Engineering Applications of Artificial Intelligence 133 (2024) 108555

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

Research paper

Regularised feed forward neural networks for streamed data classification problems

Mathys Ellis[a,*], Anna S. Bosman[a], Andries P. Engelbrecht[b,c]

[a] Department of Computer Science, University of Pretoria, Pretoria, South Africa
[b] Department of Industrial Engineering and Computer Science Division, Stellenbosch University, Stellenbosch, South Africa
[c] Center for Applied Mathematics and Bioinformatics, Gulf University for Science and Technology, Kuwait

ARTICLE INFO

Keywords:
Data streams
Classification problems
Feed forward neural networks
Quantum particle swarm optimisation
Regularisation
Concept drift

ABSTRACT

Streamed data classification problems (SDCPs) require classifiers to not just find the optimal decision boundaries that describe the relationships within a data stream, but also to adapt to changes in the decision boundaries in real-time. The requirement is due to concept drift, i.e., incorrect classifications caused by decision boundaries changing over time. Changes include disappearing, appearing or shifting decision boundaries. This article proposes an online learning approach for feed forward neural networks (FFNNs) that meets the requirements of SDCPs. The approach uses regularisation to dynamically optimise the architecture, and quantum particle swarm optimisation (QPSO) to dynamically adjust the weights. The learning approach is applied to a FFNN, which uses rectified linear activation functions, to form a novel SDCP classifier. The classifier is empirically investigated on several SDCPs. Both weight decay (WD) and weight elimination (WE) are investigated as regularisers. Empirical results show that using QPSO with no regularisation causes the classifier to completely saturate. However, using QPSO with regularisation makes the classifier efficient at dynamically adapting both its architecture and weights as decision boundaries change. Furthermore, the results favour WE over WD as a regulariser for QPSO.

1. Introduction

In the real world, data is commonly sourced from data streams (Aggarwal, 2007). *Streamed data* is sequentially-accessible and real-time by nature (Hulten et al., 2001; Aggarwal, 2007). The task of classifying streamed data, in real-time, using a set of known labels, is known as the streamed data classification problem (SDCP) (Dyer et al., 2014).

Because streamed data is based on real-world sources, the decision boundaries of SDCPs can experience changes, i.e., the decision boundaries may either disappear, appear, or shift (Rakitianskaia, 2011; Rakitianskaia and Engelbrecht, 2009, 2012). If the classifier cannot adapt to the changes in decision boundaries, then the classifier will not be able to classify correctly, i.e., the classifier will experience concept drift (Rakitianskaia and Engelbrecht, 2009; Tsymbal, 2004).

SDCPs and their potential classifiers, including FFNNs have been the subject of various studies (Hulten et al., 2001; Chu et al., 2004; Jadhav and Deshpande, 2017; Domingos and Hulten, 2000; Cui et al., 2016; Pratama et al., 2017; Telec et al., 2014; Sancho-Asensio et al., 2014; Liang et al., 2006; Aggarwal, 2007; Rakitianskaia and Engelbrecht, 2009; Krawczyk et al., 2017). To prevent concept drift in a FFNN, the FFNN must be able to dynamically optimise its architecture and to dynamically adjust its weights (Rakitianskaia and Engelbrecht, 2009, 2015b; Domingos, 2012; Zhang et al., 2000). The studies mentioned, however, showed that the subject of dynamic architecture optimisation for FFNNs with regards to SDCPs has received little attention. Literature on the superset of SDCPs, i.e. dynamic classification problems, and 3-layer FFNN classifiers have been more abundant.

Abdulkarim and Engelbrecht (2021) used a cooperative quantum particle swarm optimisation (QPSO) to train 3-layer FFNNs for non-stationary time series prediction problems. Non-stationary time series prediction problems are similar to dynamic classification problems but require the model to forecast rather than to classify (Abdulkarim and Engelbrecht, 2021; Zainuddin and Pauline, 2007).

Rakitianskaia and Engelbrecht (2009, 2012) and Rakitianskaia (2011) investigated the use of 3-layer FFNNs, trained by dynamic particle swarm optimisers (PSOs), as classifiers for dynamic classification problems (Rakitianskaia, 2011). Rakitianskaia (2011) concluded that QPSO is suitable for training FFNNs which experience concept drift. Rakitianskaia (2011) investigated the use of dynamic architecture optimisation in dynamic classification problems.

* Corresponding author.
E-mail address: mox.1990@gmail.com (M. Ellis).

https://doi.org/10.1016/j.engappai.2024.108555
Received 20 March 2023; Received in revised form 11 February 2024; Accepted 1 May 2024
Available online 16 May 2024

Aside from Rakitianskaia and Engelbrecht (2009, 2012) and Rakitianskaia (2011) there have not been any other significant investigations into the use of 3-layer FFNNs, trained by dynamic PSOs, for dynamic classification problems. Most research on 3-layer FFNNs trained by PSOs focused on static classification problems, i.e. classification problems whose data does not undergo change (van Wyk and Engelbrecht, 2016; Rakitianskaia and Engelbrecht, 2015a,b; Krogh and Hertz, 1991; Rakitianskaia and Engelbrecht, 2014a,b; Abdulkarim and Engelbrecht, 2021).

Rakitianskaia and Engelbrecht (2009, 2012) and Rakitianskaia (2011) also did not cater for the effects of saturation that are common for PSO-based training of FFNN (van Wyk and Engelbrecht, 2016; Rakitianskaia and Engelbrecht, 2015b). Rakitianskaia and Engelbrecht (2015a,b) and Dennis et al. (2020) investigated several approaches to overcome the issue of saturation. However, controlling saturation was found to be a non-trivial task. Rakitianskaia and Engelbrecht (2014b) and Bosman et al. (2018) suggested that regularisation can aid 3-layer FFNNs trained by PSOs for static classification problems by reducing saturation. Gupta and Lam (1998) showed that using weight decay (WD) regularisation can also improve the performance of FFNNs for static classification problems with noise. Studies have also shown that WD regularisation and weight elimination (WE) regularisation can improve performance with respect to accuracy and complexity for FFNNs on static classification problems (Bosman et al., 2018; Weigend et al., 1990; Rakitianskaia and Engelbrecht, 2014b; Krogh and Hertz, 1991).

Despite the existence of regularisation techniques and QPSO, no work on using them together to train FFNNs for SDCPs has been done. Lastly, SDCP literature on 3-layer FFNNs use computationally expensive activation functions (Liang et al., 2006; Telec et al., 2014; Sancho-Asensio et al., 2014). The real-time nature of streamed data requires classifiers to be as computationally inexpense as possible (Domingos and Hulten, 2000; Aggarwal, 2007). The rectified linear unit (ReLU) activation function is considered a computationally less expensive alternative to commonly used FFNN activation functions, e.g. sigmoid (Sonoda and Murata, 2017; Maas et al., 2013). Despite the ReLU activation function being computationally inexpensive, the main advantage of the ReLU activation function is that it does not saturate in the presences of positive signals (van Wyk and Engelbrecht, 2016). Furthermore, Dennis et al. (2020) showed that regularisation reduces saturation for FFNNs with ReLUs neurons that is trained by PSO (Dennis et al., 2020).

The main focus of this article is to address the above gaps in SDCP literature. This article therefore proposes a novel online learning algorithm that uses regularisation to dynamically optimise the architecture, and QPSO to dynamically adjust the weights. The learning algorithm was applied to a 3-layer FFNN, using ReLUs, to create a classifier for SDCP. Both WD and WE were considered as regularisers. The classifiers were empirically investigated on 80 SDCPs that were derived from five problem domains. The gradient-based back propagation (BP) optimisation algorithm for FFNNs was used as a benchmark for the classifiers (Werbos, 1974).

The remainder of this article is organised into eleven sections. Section 2 elaborates on SDCPs. Section 3 reviews current literature on classifiers for SDCP. Section 4 discusses the QPSO algorithm. Section 5 discusses FFNNs. Section 6 proposes several regularised FFNNs as streamed data classifiers. Section 7 presents the SDCPs used to benchmark the proposed classifiers. Section 8 presents the performance measures used to evaluate the proposed classifiers. Section 9 discusses the process used to evaluate the proposed classifiers. Section 10 analyses and discusses the performance of the proposed classifiers. Lastly, Sections 11 and 12 presents the conclusions drawn from the research conducted in this article, as well as the areas that can be investigated further in the future.

2. Streamed data classification problems

Because data streams are based on real world subjects, e.g. users and processes, and exhibit the traits of *big data*, an SDCP is considered to be a non-trivial real world problem (Santos et al., 2017; Aggarwal, 2007; Hulten et al., 2001). SDCP classifiers therefore also need to address the following issues: bounded memory, unbounded data set, changes in decision boundaries, i.e., concept drift, random dynamics, online learning, high speed data streams, one-pass, limited number of tunable control parameters, maintain low model complexity, robustness, and fault tolerance (Aggarwal, 2007; Ellis et al., 2021; Domingos and Hulten, 2000; Hulten et al., 2001; Rakitianskaia and Engelbrecht, 2012). The reader is referred to the work by Aggarwal (2007) and Ellis et al. (2021) for more information on these requirements.

The sliding window algorithm can be used to construct SDCPs from a set of patterns. Ellis et al. (2021) provided a comprehensive explanation on how to use the algorithm to construct SDCPs. An advantage of using the sliding windows algorithm to generate SDCPs, is that the algorithm's parameters can be used to estimate the problem difficulty of the SDCP, i.e., how difficult it is to accurately learn the SDCP (Ellis et al., 2021). Ellis et al. (2021) proposed four categories to describe the difficulty of a SDCP: Easy, Moderate-I, Moderate-II, and Hard. The categories describe how likely a classifier is to underfit or overfit to the patterns that are presented to it (Ellis et al., 2021).

Because the decision boundaries of SDCPs can undergo change, i.e. shift, appear, disappear, SDCPs have dynamic environments (Leskovec et al., 2014; Jadhav and Deshpande, 2017; Blackwell and Branke, 2004). Therefore, SDCPs are categorised as dynamic classification problems (Ellis et al., 2021; Blackwell and Branke, 2004). Duhain and Engelbrecht (2012) grouped dynamic environments into four categories: Quasi-static, Progressive, Abrupt, and Chaotic. Each group describes how severe the changes in decision boundaries are Duhain and Engelbrecht (2012). Change severity consists of a spatial component, i.e. the magnitude of the changes, and a temporal component, i.e. the frequency of the changes (Duhain and Engelbrecht, 2012). The Duhain and Engelbrecht classification scheme can also be used to provide additional insight into SDCPs (Ellis et al., 2021).

The reader is referred to the work by Ellis et al. (2021) for a comprehensive study on problem difficulty and problem environments of SDCP for more information.

3. Existing streamed data classifiers

Research on streamed data classifiers has gained popularity over the last two decades (Aggarwal, 2007; Sancho-Asensio et al., 2014; Singh and Chauhan, 2009). The majority of early research mostly focused on online learning and concept drift (Aggarwal, 2007; Hulten et al., 2001; Chu et al., 2004; Wang et al., 2003; Rakitianskaia and Engelbrecht, 2009; Tham, 1995; Domingos and Hulten, 2000; Tsymbal, 2004; Liang et al., 2006). Over time, research has begun to include the other streamed data classifier requirements discussed in Section 2, for example the work presented in Aggarwal (2007), Cui et al. (2016), Telec et al. (2014), Sancho-Asensio et al. (2014), Krawczyk et al. (2017), Pramod and Vyas (2012), Ngom et al. (2016), Dyer et al. (2014), Kulkarni et al. (2016), Losing et al. (2018). This section presents a minor review of streamed data classifiers to motivate the work done in this article.

Sections 3.1 and 3.2 respectively review non-artificial neural network (ANN) approaches and ANN approaches to streamed data classifiers. Section 3.3 presents the conclusions drawn from the review.

3.1. Non-artificial neural networks approaches

Two types of streamed data classifiers that have been considered in literature are decision trees and ensembles. *Decision trees* are approximators that use a tree data structure to approximate functions (Ertel, 2011; Hulten et al., 2001). On the other hand, an ensemble is a group of high variance, low bias, i.e. overfitted, classifiers working together to reduce the high variance in the outcomes of the ensemble members (Engelbrecht, 2007; Telec et al., 2014; Wang et al., 2003).

Domingos and Hulten (2000) proposed a constant memory and time per pattern, i.e. one-pass, decision tree, called the very fast decision tree (VFDT), for data streams that required learning at high speeds. Unfortunately, the VFDT does not cater for concept drift. Later, Hulten et al. (2001) modified VFDT to handle concept drift, and called the new decision tree algorithm concept-adapting very fast decision tree (CVFDT). The CVFDT was found to be better at handling concept drift compared to traditional decision tree algorithms (Hulten et al., 2001).

Wang et al. (2003) proposed an ensemble of classifiers that can be used for SDCPs. The ensemble weights the relevance of each classifier in the current epoch using an expected accuracy for the current chunk of patterns in the data stream (Wang et al., 2003). The approach addresses the concept drift, high speed, robustness and online learning requirements (Wang et al., 2003).

Babaeian et al. (2019) used an ensemble of logistic regression based classifiers, which incorporated regularisation, to detect the drowsiness of drivers. The approach performed better than prior classifiers and reduced overfitting (Babaeian et al., 2019).

Telec et al. (2014) investigated the use of an ensemble of different ANNs for modelling a data stream of real-estate market prices. Telec et al. (2014) showed ANN ensembles to be more effective than ensembles of non-ANN classifiers.

Both decision trees and ensembles are outside the scope of this article. The reader is, however, referred to the survey on decision trees by Kotsiantis (2013) and the survey on ensembles by Krawczyk et al. (2017) for more information.

3.2. Artificial neural networks

ANNs are a group of function approximators whose structures are inspired by neurons and synapses found in the human brain (McCulloch and Pitts, 1943). Various ANN approaches have been developed to solve SDCPs (Cui et al., 2016; Pratama et al., 2017; Liang et al., 2006; Sancho-Asensio et al., 2014; Rakitianskaia, 2011; Rakitianskaia and Engelbrecht, 2009).

Cui et al. (2016) introduced a new one-pass *sequence learning* ANN architecture called hierarchical temporal memory (HTM), based on the recent neuroscience discovery of synaptic integration in the cerebral cortex. Sequence learning is a type of classification problem that requires classification of the outcome from a sequence of past patterns (Cui et al., 2016). To do so, the classifier must maintain *temporal context* (Cui et al., 2016; Engelbrecht, 2007). Temporal context refers to the time-based contextual dependencies that can exist between patterns, i.e. a sequence of events that provides context for the next event (Cui et al., 2016). HTM allows for *temporal context* to be built up and maintained by the ANN. Cui et al. (2016) showed that HTM was more effective, robust, and fault tolerant than recurrent neural networks (RNNs) and other traditional ANNs in streamed data sequence learning problems. The HTM does not provide mechanisms to control model complexity, handle random dynamics, and reduce the number of tunable control parameters. The algorithm was not evaluated on SDCPs but streamed data sequence learning problems. Thus, the applicability of HTM to SDCPs is unknown.

Pratama et al. (2017) proposed an online random neural network (RdNN), called the recurrent type-2 random vector functional link network (RT2McRVFLN), to address streamed data problems (SDPs). RdNNs, first proposed by Gelenbe (1989), are ANNs which are based on the biological behaviour of neuron circuits. The mechanics of RdNNs are outside the scope of this article. The reader is referred to Gelenbe (1989) for more information. Note that the RT2McRVFLN is a type of RNN. The RT2McRVFLN was found to be competitive against other streamed data classifiers in terms of the accuracy-to-simplicity trade-off, and was able to control model complexity to some extent (Pratama et al., 2017). The algorithm did not provide any explicit means to deal with bounded memory, noisy patterns, system faults, reduce the number of tunable control parameters, and ensure patterns were only processed once during training. Furthermore, the RT2McRVFLN was not compared to traditional benchmark algorithms such as stochastic BP, and the empirical analysis was done on a limited set of three benchmark problems.

Liang et al. (2006) proposed a fast online sequence learning FFNN based on extreme learning machine (ELM) concepts, called online sequential extreme learning machine (OS-ELM) (Liang et al., 2006). ELMs are FFNNs that allow fast training by inverting the training set matrix and output matrix in order to find the optimal weights. Because there is no guarantee that the exact inverse might exist, a pseudo matrix inversion formula, i.e. the Moore–Penrose generalised inverse, is used instead of normal matrix inversion (Liang et al., 2006). The pseudo matrix inversion attempts to find an approximate inverse that minimises the least square error (Liang et al., 2006). The advantages of OS-ELM is its ability to handle situations requiring high speed data processing and little parameter tuning. The algorithm did not provide any explicit means to deal with concept drift, noisy patters, system faults, and model complexity.

Sancho-Asensio et al. (2014) proposed a robust, online, concept-drift handling classifier system for SDCPs called supervised neural constructivist system (SNCS). SNCS makes use of FFNNs, stochastic BP, and genetic algorithms (GAs) (Sancho-Asensio et al., 2014). GAs are population-based optimisers like PSOs. GAs are based on evolutionary concepts and not swarm movements (Ertel, 2011; Engelbrecht, 2007). The GA is used to select a FFNN architecture, while BP is used to adjust the weights. The algorithm did not provide any explicit means to deal with bounded memory, high-speed data streams, system faults, reduce the number of tunable control parameters, or ensure patterns were only processed once during training.

Abdulkarim and Engelbrecht (2021) used a cooperative QPSO to train 3-layer FFNNs for non-stationary time series prediction problems. Abdulkarim and Engelbrecht (2021) concluded that FFNNs trained with dynamic PSO are effective for non-stationary time series problems, and that non-stationary time series problems also cause concept drift.

Rakitianskaia and Engelbrecht (2009, 2012) and Rakitianskaia (2011) investigated the use of 3-layer FFNNs, trained by dynamic PSOs, as classifiers for dynamic classification problems (Rakitianskaia, 2011). Rakitianskaia (2011) concluded that the dynamic PSOs, including the QPSO, are suitable for learning FFNNs which experience concept drift in online situations, and can in some cases outperform stochastic BP. Rakitianskaia (2011) did not investigate classification problems with respect to the other SDCP requirements, discussed in Section 2.

Aside from Rakitianskaia and Engelbrecht (2009, 2012), Rakitianskaia (2011), and Abdulkarim and Engelbrecht (2021), there has not been any other significant investigations into the use of 3-layer FFNNs trained by dynamic PSOs. Most research on 3-layer FFNNs trained by PSOs focused on static classification problems and classification problems that require only the handling of concept drift, and not SDCPs (van Wyk and Engelbrecht, 2016; Rakitianskaia and Engelbrecht, 2009; Bosman et al., 2018; Sancho-Asensio et al., 2014; Rakitianskaia and Engelbrecht, 2014a,b).

Gupta and Lam (1998) have shown that WD combined with the BP algorithm can handle static classification problems with noise. Furthermore, studies have shown that WD and WE can improve the accuracy and complexity performance of both BP and PSO weights adjustment algorithms (Bosman et al., 2018; Weigend et al., 1990; Rakitianskaia and Engelbrecht, 2014b; Krogh and Hertz, 1991). However, current studies (Bosman et al., 2018; Weigend et al., 1990; Rakitianskaia and Engelbrecht, 2014b; Gupta and Lam, 1998; Krogh and Hertz, 1991) considered only static classification problems and not dynamic classification problems.

3.3. Summary

Table 1 summaries the streamed data classifier requirements focused on by the classifiers reviewed in this Section 3. From the above review of current SDCP literature and Table 1, the following conclusions are made:

- Most streamed data classifiers that were investigated considered limited subsets of the SDCP requirements presented in Section 2.
- There is no significant research into the use of regularised 3-layer FFNNs to deal with SDCPs.
- The issue of saturation has been addressed for 3-layer FFNNs trained by static PSOs, but the suggested approaches have not been tested on streamed data FFNN classifiers trained by dynamic PSOs.
- There is no significant research into the use of combining regularisation and dynamic PSOs to train 3-layer FFNNs for SDCPs.

4. Quantum particle swarm optimisation

PSO is a class of stochastic population-based optimiser (Eberhart and Kennedy, 1995). PSO consists of n_p particles, known as a swarm. Each particle represents a candidate solution. The solution is stored as the particle's N-dimensional position vector, $\vec{x}$. Over the course of several optimisation iterations, the particles work together, as a collective, to find optimal solutions for the N-dimensional objective function, $f : \mathbb{R}^N \rightarrow \mathbb{R}$ (Eberhart and Kennedy, 1995; Engelbrecht, 2007).

Particle movement is determined by one or more update rules, collectively referred to as the particle's behaviour. Behaviour can vary between particles (Blackwell and Branke, 2004). *Sub-swarms* are thus formed by grouping particles based on their behaviours (Engelbrecht, 2010; Gies and Rahmat-Samii, 2004).

The standard particle update rules require each particle to have an N-dimensional *velocity vector*, $\vec{v}$. The velocity vector represents the direction and magnitude of the particle's movement at iteration t. The standard particle update rules consist of a velocity update rule,

$$v_{ij}(t+1) = \omega v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_{ij}(t) - x_{ij}(t)], \quad (1)$$

and a position update rule,

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1), \quad (2)$$

where $v_{ij}(t)$ is the jth element of the velocity vector of particle i at iteration t; $y_{ij}(t)$ is the jth element of the personal best position of particle i at the iteration t; $\hat{y}_{ij}(t)$ is the value of the jth element of the neighbourhood best position of particle i at iteration t; and $x_{ij}(t)$ is the jth element of particle i's position vector at iteration t; $r_{1j}(t)$ and $r_{2j}(t)$ are the jth elements of $\vec{r}_1$ and $\vec{r}_2$, respectively (Engelbrecht, 2007). The elements of $\vec{r}_1$ and $\vec{r}_2$ are sampled uniformly from the range $[0, 1]$ (Eberhart and Kennedy, 1995). The variables c_1 and c_2 represent the user-specified positive acceleration coefficients of the particle, and ω represents the user-specified inertia weight of the particle. PSO has been successfully implemented for a multitude of optimisation tasks, ranging from single-objective to dynamic optimisation problems (Engelbrecht, 2007; Gies and Rahmat-Samii, 2004; Blackwell and Bentley, 2002; Mendes et al., 2002; Rakitianskaia and Engelbrecht, 2014b; Zhang et al., 2000; Harrison et al., 2015; Rakitianskaia and Engelbrecht, 2009).

Optimisation problems with dynamic environments require a PSO that is able to both detect changes and handle said changes (Blackwell et al., 2008). Blackwell and Branke (2004) proposed the QPSO to address dynamic optimisation problems. QPSO adapted the standard PSO model by introducing *quantum particles* (Blackwell and Branke, 2004).

Quantum particles act as if they are electrons around the nucleus of an atom, where the nucleus is the social guide of the respective particle (Blackwell and Branke, 2004). The quantum mechanics model of an atom dictates that an electron's position is somewhere within the sphere around the nucleus of an atom (Harrison et al., 2015). The exact position of an electron can only be guessed with some degree of certainty as it is constantly changing (Blackwell and Branke, 2004). Hence, quantum particles' positions are estimated to be somewhere in a hypersphere around the social guide at a specific point in time.

These concepts are implemented by equally dividing the swarm of the QPSO into two sub-swarms, each with a different set of behaviours (Blackwell and Branke, 2004; Rakitianskaia and Engelbrecht, 2009). The first sub-swarm uses the standard particle update rules, and the second sub-swarm uses the quantum update rules (Blackwell and Branke, 2004; Rakitianskaia and Engelbrecht, 2009). The quantum update rule set consists of the position update rule (Harrison et al., 2015)

$$x_{ij}(t+1) \sim d(\hat{y}_{ij}(t), r) \quad (3)$$

where d refers to a user-defined sampling distribution and r refers to the radius around $\vec{\hat{y}}$. Quantum particles' positions are re-sampled every iteration, using the statistical distribution d, from a hypersphere around the $\vec{\hat{y}}$ particle, with a radius r (Blackwell and Branke, 2004).

The quantum update rule prevents complete convergence of the swarm, thus preventing swarm diversity loss. The combination of standard and quantum update rules allow the quantum particles to continuously explore the search space, while the standard particles exploit the quantum particles' findings (Blackwell and Branke, 2004). A larger value for r results in a larger sample area around $\vec{\hat{y}}$, thus enabling quantum particles to explore a larger portion of the search space (Blackwell and Branke, 2004). The inverse of this occurs for smaller values of r.

A commonly used sampling distribution is the uniform distribution (Blackwell and Branke, 2004; Harrison et al., 2015). However, Harrison et al. (2015) found that the *de facto* uniform distribution performs neither the best nor worst in dynamic environments. Harrison et al. (2015) suggested the use of the linear decreasing distribution with a small radius.

Note that this article focuses on QPSO instead of the other dynamic PSO variants, because QPSO is computationally less complex than other dynamic PSOs. Furthermore, there is a significant body of research on QPSO, including improvements and its successful use to train FFNNs.

5. Feed forward neural networks

The 3-layer FFNN is a class of ANN that is suitable for classification problems (LeCun et al., 1998). The objective of the 3-layer FFNN is to model the function f that describes the classification problem (Zainuddin and Pauline, 2007). That is, the function which maps a pattern of independent variables, $\vec{z}$, to the label, $\vec{t}$ with an ϵ amount of error, i.e., $f(\vec{z}) = \vec{t} + \epsilon$ (Engelbrecht, 2007; Wilamowski, 2003).

Training a FFNN is an optimisation problem that comprises of two objectives (Engelbrecht, 2001; Liu et al., 2002; Ertel, 2011; Alpaydın, 2010): (i) find the synapse weight values that minimise the error function, and (ii) find the architecture that minimises the complexity of the FFNN, i.e., the number of synapses and neurons.

Section 5.1 discusses the use of BP to adjust the weights of a FFNN. Section 5.2 discusses the use of PSO to adjust the weights of a FFNN. Section 5.3 discusses the use of regularisation to select the architecture of a FFNN.

Table 1
Comparison of the requirements focused on by the reviewed classifiers.

Requirements	Reviewed classifiers								
	VFDT (Domingos and Hulten, 2000)	*CVDFT* (Hulten et al., 2001)	*Ensemble-based* (Wang et al., 2003; Telec et al., 2014)	*HTM* (Cui et al., 2016)	*OS-ELM* (Liang et al., 2006)	*RT2MCRvFLN* (Pratama et al., 2017)	*SNCS* (Sancho-Asensio et al., 2014)	*Stocastic BP FFNN* (Rakitianskaia and Engelbrecht, 2009; Rakitianskaia, 2011; Rakitianskaia and Engelbrecht, 2012)	*QPSO FFNN* (Rakitianskaia and Engelbrecht, 2009; Rakitianskaia, 2011; Rakitianskaia and Engelbrecht, 2012)
Bounded memory	X	X		X	X			X	X
Unbounded data set	X	X		X	X	X	X	X	X
Concept drift		X	X	X		X	X		X
Random dynamics						X	X		X
Online learning	X	X	X	X	X	X	X	X	X
High speed	X	X	X	X	X	X			
One-pass	X	X		X	X				
Limited number of tunable control parameters				X	X			X	X
Maintain low model complexity						X	X		
Robustness			X	X			X		
Fault tolerance				X					

5.1. Training with back propagation

A commonly used algorithm for benchmarking training algorithms of FFNNs is the BP algorithm (Mendes et al., 2002; Rakitianskaia and Engelbrecht, 2009; Fahlman, 1989). BP conducts a local-search of the search space using gradient descent (GD) (Werbos, 1974). The BP algorithm is considered to be a static training algorithm in literature, however, Rakitianskaia (2011) showed that the algorithm can train FFNN in certain dynamic environments. Of the three learning modes, i.e., stochastic, batch, and mini-batch, the stochastic mode is the most suited for SDCP (Ellis et al., 2021). Stochastic BP immediately changes the weights of the FFNN after a pattern has been processed (Ertel, 2011; Werbos, 1974).

There are two important control parameters that must be considered for stochastic BP: (i) the momentum term (α), i.e. how much the previous weight changes influence the weight change, and (ii) the learning rate (η), i.e. how much of the current weight change is experienced by a weight (Engelbrecht, 2007; Ertel, 2011; Bosman et al., 2018). These control parameters help to combat the haphazard changes in the search trajectory and local optimum trapping that the stochastic BP can suffer from Rakitianskaia (2011), LeCun et al. (1998), Ertel (2011).

5.2. Training with particle swarm optimisation

Algorithms that use PSO to adjust FFNN weights start off by initialising n_p instances of a FFNN (Mendes et al., 2002). Each particle's position vector represents the weights and biases of a FFNN (Engelbrecht, 2007; Mendes et al., 2002). These position vectors are commonly referred to as *weight vectors*. The objective function is the FFNN's error function (Rakitianskaia and Engelbrecht, 2009). Mean square error (MSE) is typically used by PSO-based weight adjustment algorithms (Rakitianskaia and Engelbrecht, 2009; van Wyk and Engelbrecht, 2016).

Five advantages that PSO-based weight adjustment algorithms have over BP are (i) their resilience to local optimum trapping (Rakitianskaia and Engelbrecht, 2009), (ii) their ability to handle concept drift, if a dynamic PSO is used, e.g. QPSO (Harrison et al., 2015), (iii) they do not use computationally expensive derivative calculations (Mendes et al., 2002), (iv) they are computationally simple and easy to implement (Ismail and Engelbrecht, 2000), and (v) there is empirically and theoretically derived guidance on control parameter value assignment to ensure that the PSO reaches an equilibrium state (Cleghorn and Engelbrecht, 2016).

Rakitianskaia and Engelbrecht (2009) investigated the use of QPSO and charged particle swarm optimisation (CPSO) to train 3-layer FFNNs for dynamic classification problems. It was found that QPSO and CPSO performed similarly to each other, and tended to outperform BP in terms of accuracy (Rakitianskaia and Engelbrecht, 2009). Rakitianskaia and Engelbrecht (2009), however, did not consider optimising the architecture after an environment change.

Zhang et al. (2000) designed a PSO-based training algorithm for 3-layer FFNNs that incorporates architecture selection. However, Zhang et al. (2000) did not consider that dynamic architecture selection results in a dynamic optimisation problem itself, where the search space dimensionality changes over time. Neither did they consider dynamic classification problems.

A key issue that is neglected in the above works pertaining to the use of PSO to train FFNN is *saturation* (Rakitianskaia and Engelbrecht, 2015a; Dennis et al., 2020). A neuron is *saturated* when its activation value is always an asymptotic end of an activation function that is bounded (Rakitianskaia and Engelbrecht, 2015a; LeCun et al., 1998). Saturation hinders the ability of a FFNN to approximate problems, because it reduces the information capacity of a neuron, i.e., the set of possible activation values (Rakitianskaia and Engelbrecht, 2015b; Dennis et al., 2020). When PSO is used to adjust the weights, saturation can degrade performance severely and also lead to overfitting (Rakitianskaia and Engelbrecht, 2015b; van Wyk and Engelbrecht, 2016). If the FFNN begins to saturate, then the error function constantly produces similar information about the search space. Velocities thus either stagnate or explode because the memory of the PSO never changes. Both cases usually prevent the search from converging to an optimal solution (Rakitianskaia and Engelbrecht, 2015b,a, 2014b). Rakitianskaia and Engelbrecht (2015b) showed that controlling saturation in PSO is a difficult task because of the mechanics of PSO.

5.3. Regularisation

Overfitting is generally described in literature through the concepts of *bias*, i.e. how far the outputs of the ANN are from their targets, and *variance*, i.e. how scattered the outputs are for similar unseen inputs (Alpaydın, 2010; Domingos, 2012; Geman et al., 1992; Babaeian et al., 2019). An ANN overfits when it has a low bias and high variance (Alpaydın, 2010; Geman et al., 1992; Babaeian et al., 2019). While underfitting is essentially the inverse of overfitting, i.e. high bias and low variance (Alpaydın, 2010; Geman et al., 1992).

The objective of architecture selection is to prevent overfitting and underfitting by ensuring that the complexity of the FFNN architecture is appropriate for the approximation problem (Liu et al., 2002). Regularisation algorithms are a class of architecture selection algorithms which do not explicitly change the architecture, but rather neutralise the contribution of unnecessary weights by driving their weight values to zero (Bosman et al., 2018; Weigend et al., 1990).

Regularisation augments the error function by adding a regularisation (or penalty) term, E_r, to penalise structural complexity, i.e.,

$$E'_t = E_t + \lambda_r E_r \ , \tag{4}$$

where E'_t is the regularised error function, E_t is the original error function and λ_r is the regularisation coefficient (Weigend et al., 1990).

The regularisation coefficient controls how much the overall error is influenced by the regularisation term. The regularisation terms considered in this article need to be minimised, thus the larger λ_r is, the more complexity is penalised resulting in more weight values to be driven towards zero at the cost of increasing the model error (Krogh and Hertz, 1991; Gupta and Lam, 1998). Too large λ_r values may result in underfitting, because important weights might also be driven to zero (Rakitianskaia and Engelbrecht, 2014b; Babaeian et al., 2019). Whereas too small λ_r values may cause overfitting, because E'_t will be more concerned with fitting the training set than reducing model complexity, and therefore fit noise (Weigend et al., 1990; Babaeian et al., 2019). Because the exact value for λ_r is problem dependent, λ_r has to be carefully tuned to achieve a balance between model error and model complexity (Engelbrecht, 2007).

Regularisation is attractive because it is a simple on-line approach to architecture optimisation, and there is no need to decide on a threshold value to determine when a weight/neuron should be removed (Gupta and Lam, 1998; Weigend et al., 1990). Regularisation, however, does introduce an additional problem dependent parameter, i.e. λ_r. Two well-known regularisation terms are weight decay (WD) and weight elimination (WE) (Engelbrecht, 2007; Rakitianskaia and Engelbrecht, 2014b).

WD is defined as

$$E_r = \frac{1}{2} \sum_{j=1}^{n_s} w_j^2 \ , \tag{5}$$

where n_s is the number of synapses in FFNN and w_j is the jth weight in the FFNN. In essence, WD counts the number of synapses based on their weight value (Rakitianskaia and Engelbrecht, 2014b). FFNNs with larger weight values will be penalised more than FFNNs with smaller weight values (Rakitianskaia and Engelbrecht, 2014b).

Because WD penalises the total number of weights that can be assigned to the synapses, the training algorithm is forced to allocate weights to only the necessary synapses. Therefore, the training algorithm will drive the weights of irrelevant synapses to zero (Gupta and Lam, 1998).

A drawback of WD is that it does not associate any notion of relevance to the values of weights. WD, therefore, penalises a weight value with the same aggressiveness regardless if the value is relevant or not (Bosman et al., 2018; Krogh and Hertz, 1991).

WE was proposed by Weigend et al. (1990) to overcome the issue of equally aggressive penalisation of relevant and irrelevant weight values seen in WE, and is defined as

$$E_r = \sum_{j=1}^{n_s} \frac{\frac{w_j^2}{w_0^2}}{1 + \frac{w_j^2}{w_0^2}} , \tag{6}$$

where w_0 is a non-zero threshold which controls the level at which weight values become irrelevant. The larger w_0 is, the less large weights are penalised, because they are considered more relevant, i.e. needed to model the problem better, and vice versa (Weigend et al., 1990).

Bosman et al. (2018) found that WE both smooths the gradients of the minima and introduces additional minima into the search space. The weight value threshold parameter, w_0, controls the sharpness of the introduced minima, i.e. the magnitude of their gradients. The value of w_0, however, is problem dependent (Bosman et al., 2018; Rakitianskaia and Engelbrecht, 2012). The main drawback of WE is thus that it introduces another control parameter that needs to be tuned (Engelbrecht, 2007).

6. Regularised feed forward neural networks as streamed data classifiers

Regularised 3-layer FFNNs have not been investigated thoroughly as streamed data classifiers, despite their ability to manage the complexity of the data. To address this gap in the research, this article proposes four regularised 3-layer FFNN classifiers for SDCPs. The term *regularised classifier* is used in this article to describe any classifier that uses regularisation.

The remainder of the section is organised as follows. Section 6.1 describes the FFNN architecture used for the purpose of this study. Section 6.2 describes two BP-based regularisation learning algorithms. Section 6.3 describes two QPSO-based regularisation learning algorithms. Section 6.4 presents the four streamed data classifiers.

6.1. Architecture

The 3-layer FFNN architecture proposed in this article uses summation units to calculate the net input sum (*net*) (Engelbrecht, 2007; Ertel, 2011). Furthermore, the proposed architecture uses the popular ReLU function to calculate activation values of the neurons (Mendes et al., 2002; Rakitianskaia and Engelbrecht, 2009; van Wyk and Engelbrecht, 2016; Ertel, 2011; LeCun et al., 2015). Because the ReLU function has a low computational complexity and stronger gradients, the function is often preferred over bounded activation functions, such as sigmoid (LeCun et al., 2015; van Wyk and Engelbrecht, 2016; Maas et al., 2013). The ReLU function calculates the activation value as

$$f_{ReL}(net) = max\{\lambda(net - \theta), 0\} \tag{7}$$

where the steepness of the curve is controlled by λ, and θ controls the amount the curve is shifted by van Wyk and Engelbrecht (2016), LeCun et al. (1998). Note that for the purpose of this article λ was set to 1 and θ was learnt via the bias neurons.

Initialisation of the weights of a FFNN around zero, in a uniform manner, helps to prevent the FFNN from favouring some of the regions in the search space over others (LeCun et al., 1998). The weight of each synapse, including the synapses of bias neurons, were initialised uniformly within the range (Engelbrecht, 2007; Fernández-Redondo and Hernández-Espinosa, 2001; LeCun et al., 1998)

$$\left[\frac{-1}{\sqrt{fanin}}, \frac{1}{\sqrt{fanin}}\right] \tag{8}$$

where $fanin$ is the number of synapses that provide input into the neuron, which is associated with the synapse being initialised (Wessels and Barnard, 1992; LeCun et al., 1998).

The number of input neurons (n_i) are fixed to the input dimensionality of the SDCP, i.e. I. The hidden layer selected in a way that ensures the architecture is *over-parametrised*, i.e. the number of hidden neurons (n_h) is more than the number needed for the particular problem. Note that this article investigates problems where the number of hidden neurons required has been empirically determined.

The input layer and the hidden layer of the architecture also contains a single bias neuron each. The input signal of the bias neurons is fixed at -1. The reader is referred to Appendix C in the supplementary material of this article for more information on the structure of the proposed architecture.

The architecture assumes the use of binary encoded targets, e.g. each element in a target vector is either 0 or 1 (Potdar et al., 2017). If the SDCP only has two classes, then *binary encoding* to encode the target vectors, otherwise, *one-hot encoding* is used (Potdar et al., 2017; Harris and Harris, 2016). The number of output neurons (n_k) is therefore dependent on the how many discrete classes an SDCP has. If there are two classes, then the FFNN uses one output neuron, otherwise the FFNN uses one output neuron per class (Harris and Harris, 2016).

Because of the ReLU activation function, the values provided by the output neurons are bounded below by 0. Binary classification target vectors, however, need the values of output neurons to be bounded to the range $[0, 1]$. The architecture uses thresholds to constrain the value of an output neuron, o_k, to the range $[0, 1]$ as follows (Engelbrecht, 2007; Ertel, 2011; Rakitianskaia, 2011):

$$o_k = \begin{cases} 1, & \text{if } o_k \geq 1.0, \\ o_k, & \text{otherwise.} \end{cases} \tag{9}$$

6.2. Back propagation learning algorithms

The proposed architecture requires the weights of the hidden-to-input layer and output-to-hidden layer to change. Because this article makes use of stochastic learning the training error is calculated per pattern using the sum square error (SSE) function, as follows

$$SSE_p = \sum_{k=0}^{n_k} (t_{p,k} - o_{p,k})^2 \tag{10}$$

where p is a pattern, $o_{p,k}$ is the value of the kth output after pattern p has been evaluated, and $t_{p,k}$ is the kth target of pattern p (Ertel, 2011; Alpaydın, 2010). The contribution to the training error of each neuron, i.e., the error signal, is calculated using the relevant derivative calculations. The error signals are augmented with the learning rate and momentum to calculate the change in weight required for each neuron.

Note that because of the piecewise nature of f_{ReL}, the activation function is not differentiable at 0 (Maas et al., 2013; van Wyk and Engelbrecht, 2016; Sonoda and Murata, 2017). In practice, when training with BP, f'_{ReL} is considered to be 1 if $net_{o_k} > 0$, otherwise f'_{ReL} is 0 (van Wyk and Engelbrecht, 2016; Maas et al., 2013; LeCun et al., 2015).

To incorporate WD or WE regularisation into the BP learning algorithm is a relatively straightforward process (Bosman et al., 2018; Weigend et al., 1990). The regularisation error function, Eq. (4), with E_r as the chosen penalty term, i.e., Eq. (5) or Eq. (6), must be used as the new training error function instead of SSE_p. To do so, the partial derivative of the product between the regularisation coefficient (λ_r) and the penalty term, was calculated and incorporated into the weight adjustment calculations. The reader is referred to Appendix D in the supplementary material of this article for more information on the backpropagation learning algorithms.

6.3. Quantum particle swarm optimisation learning algorithms

The QPSO weights adjustment algorithm in this article utilises the MSE, calculated as,

$$MSE = \frac{\sum_{p=1}^{|D|} \sum_{k=1}^{n_k} (t_{p,k} - o_{p,k})^2}{|D| \times n_k} \tag{11}$$

where $D = \{(\vec{z}_p, \vec{t}_p) | p = 1, \dots, |D|\}$ (Rakitianskaia and Engelbrecht, 2012; Twomey and Smith, 1995). The algorithm uses the Von Neumann topology to select the neighbourhood best particle (Rakitianskaia, 2011; Rakitianskaia and Engelbrecht, 2012; Engelbrecht, 2007; Kennedy and Mendes, 2002; van Wyk and Engelbrecht, 2016). The reader is referred to Appendix E in the supplementary material of this article for more information on the QPSO learning algorithm.

Application of WD and WE regularisation to QPSO weights adjustment algorithm requires even fewer modifications than needed with the BP learning algorithms. Regularisation is achieved by changing the training error function from MSE to the regularisation error, Eq. (4), with Et as MSE and E_r as the chosen penalty term, i.e., Eq. (5) or Eq. (6).

6.4. Novel streamed data classifiers

By combining the proposed architecture and four regularisation learning algorithms, the following four streamed data classifiers are formed:

- *BP-WD*, which uses the WD BP-based learning algorithm to train the FFNN. BP-WD has three tunable control parameters, i.e. α, η and λ_r.
- *BP-WE*, which uses the WE BP-based learning algorithm to train the FFNN. BP-WE has four tunable control parameters, i.e. α, η, λ_r and w_0.
- *QPSO-WD*, which uses the WD QPSO-based learning algorithm to train the FFNN. QPSO-WD has seven tunable control parameters, i.e. ω, c_1, c_2, n_p, distribution d, r and λ_r.
- *QPSO-WE*, which uses the WE QPSO-based learning algorithm to train the FFNN. QPSO-WE has eight tunable control parameters, i.e. ω, c_1, c_2, n_p, distribution d, r, λ_r and w_0.

Because of the online learning requirement for SDCPs, no stopping conditions are used to terminate the learning algorithms of the four streamed data classifiers. However, each training set is only trained for one epoch to enforce the one-pass requirement.

Note that the term *proposed classifiers* is used to refer to BP-WD, BP-WE, QPSO-WD and QPSO-WE for the remainder of this article.

7. Benchmark streamed data classification problems

This article uses a suite of 80 SDCPs to empirically evaluate the classifiers proposed in Section 6. The suite is broken up equally into five problem domains, namely (Street and Kim, 2001; Harries, 1999; Rakitianskaia and Engelbrecht, 2009, 2012; Ellis et al., 2021):

- **Moving hyperplane**. An artificial domain that splits a 10 dimensional input space into two classes using a hyperplane that moves over time.
- **Dynamic sphere**. An artificial domain that splits a three dimensional input space into two classes using a hypersphere that changes over time.
- **Sliding thresholds**. An artificial domain that splits a two dimensional input space into three classes using two parallel lines that change over time. This problem domain also simulates the presence of irrelevant inputs.
- **SEA concepts**. An artificial domain that splits a three dimensional input space into two classes using a constraint that changes over time. This problem domain simulates the data sets that contain noisy patterns at a level of 10%.

Table 2
Benchmark problems labels.

		w_f			
		1	*2*	*5*	*10*
w_s	*1*	A1	A2	A3	A4
	2	B1	B2	B3	B4
	5	C1	C2	C3	C4
	10	D1	D2	D3	D4

Table 3
3-Layer FFNN architectures found by Rakitianskaia's (Rakitianskaia, 2011).

Parameter	Problem domain				
	Hyperplane	*Sphere*	*Thresholds*	*SEA*	*Electricity*
n_i	10	3	2	3	6
n_h	6	4	3	4	6
n_k	1	1	3	1	1

- **Electricity pricing**. A real world domain that splits a six dimensional input space, based on the electricity pricing data from the state of New South Wales, Australia, into two classes.

These benchmark problems were used because the characteristics of the suite have been extensively analysed (Ellis et al., 2021; Rakitianskaia, 2011; Rakitianskaia and Engelbrecht, 2012). Ellis et al. (2021) labelled each benchmark problem in a problem domain according to Table 2. A4, D1, D4 and A1 represent the four extreme benchmark problems in each problem domain (Ellis et al., 2021). Furthermore, Ellis et al. (2021) identified A1 problems as *streamed benchmark problems*, because the sliding window algorithm did not alter the occurrence of the patterns in the data set of the problem domain.

Rakitianskaia (2011) determined optimised 3-layer FFNNs architectures for each of the five problem domains. The optimal n_h found in Rakitianskaia (2011) is, thus, a good basis for determining the over-parametrised architectures that were used in this article. This article used over-parametrised architectures that were obtained by multiplying the optimised n_h value with a factor of two. Table 3 lists the optimal architecture configurations determined by Rakitianskaia (2011).

8. Performance measures

The classifiers proposed in Section 6 were empirically compared to a set of baseline classifiers. Section 8.1 presents the baseline classifiers used in this article. Furthermore, the performance measures used in this article needed to make provision for the design requirements of SDCP. Section 8.2 discusses the methodology used to measure the performance of a streamed data classifier. Because streamed data classifiers have a variety of aspects to consider, a broad range of performance categories needs to be analysed. Sections 8.4 to 8.10 discusses the various measures used in this article to measure the saturation, accuracy, structural complexity, computational complexity, control parameter impact, weight distribution, and swarm diversity of the classifiers.

8.1. Baseline classifiers

The following two non-regularised versions were used to compare the four proposed classifiers against:

- *BP-N*, which uses BP learning algorithm to train the proposed architecture (presented in Section 6.1). BP-N has two control parameters that need to be tuned, namely α, η.
- *QPSO-N*, which uses Algorithm 2 to train the proposed architecture (presented in Section 6.1). QPSO-N has six control parameters that need to be tuned, namely ω, c_1, c_2, n_p, d, r. In the comparisons of QPSO-N with QPSO-WD and QPSO-WE, the control parameters ω, c_1, c_2, n_p and d were fixed in QPSO-N to the same values used for QPSO-WD and QPSO-WE.

Note that for the remainder of this article, the term *BP classifiers* refers to the BP-N, BP-WD and BP-WE classifiers. Likewise, the term *QPSO classifiers* refers to the QPSO-N, QPSO-WD and QPSO-WE classifiers.

8.2. Measuring performance of streamed data classifiers

Performance measures of a streamed data classifier need to consider the following two issues (Ellis et al., 2021; Morrison, 2003):

1. *Multiple environments*. That is, performance may be different for each environment instance. Thus, the metric needs to be applied per epoch and averaged. The collective mean of an error, $\bar{C}_{proformancemetric}$, is commonly used in literature to address this issue (Morrison, 2003; Rakitianskaia and Engelbrecht, 2012; Ellis et al., 2021).
2. *Only one pattern is available per epoch*. That is, the one-pass requirement, bounded memory requirement, and sequential availability of SDCPs force the training and generalisation errors to be calculated using the same pattern during an epoch. Generalisation errors must therefore be calculated before the FFNN is trained with the pattern, and training errors must be calculated afterwards.

Because certain performance measures in literature require a set of patterns, this article created a special set that consisted of the 30 most recent patterns. The set is called the "memory set", and is exclusively used for the purpose of analysis. The memory set is based on the mechanics of the *predictive sequential error* suggested by Gama et al. (2009).

8.3. Saturation performance measures

Rakitianskaia and Engelbrecht (2015a) proposed the following metric to measure saturation in the hidden neurons,

$$\varphi_{b_w} = \frac{\sum_{b=1}^{B} |\bar{g}'_b| f_b}{\sum_{b=1}^{B} f_b} \tag{12}$$

where f_b is the frequency, i.e. number of hits, for bin b, B represents the number of activation value bins created using the binning width b_w, and $\bar{g}'_b$ is the average activation value, scaled to the range $[-1, 1]$.

The saturation level measured by φ_{b_w} is in the range $[0, 1]$. If φ_{b_w} is zero, then there is no saturation. On the other hand, if the value is one, then the neurons are completely saturated (Rakitianskaia and Engelbrecht, 2015a). A value less than 0.5 indicates a normal distribution of activation values, while a value of 0.5 indicates a uniform distribution of activation values (Rakitianskaia and Engelbrecht, 2015a).

Because Eq. (12) requires bounded activation functions, the metric had to be altered for it to work with ReLU activation functions. An upper bounds parameter was introduced in order to bound the activation values. The value of the upper bound parameter estimates the level of activation that would cause ReLU hidden neurons to saturate the FFNN in a detrimental way. No research on what bounds would be appropriate for ReLU activation functions has been done, therefore, several upper bounds were investigated (Rakitianskaia and Engelbrecht, 2015a; Dennis et al., 2020).

Rakitianskaia and Engelbrecht (2015b) argued that saturation in the output neurons was necessary for classification, but premature saturation in the hidden layers was unwanted. Thus, the smaller the activation values in the hidden neurons, the lower the chance of premature saturation (Rakitianskaia and Engelbrecht, 2015b). Reasonable values for the upper bound are, therefore, values greater or equal to the upper bounds of the active ranges of the output neurons.

Three upper bounds, i.e. 1, 5 and 10, were chosen near to the upper bound of the active ranges of the output neurons, i.e. 1. To obtain a single saturation metric this article combined the three parametrisations of Eq. (12), as follows:

$$\varphi_{0.1} = \min\{\varphi_{0.1,1}, \varphi_{0.1,5}, \varphi_{0.1,10}\} \tag{13}$$

where $\varphi_{0.1,1}$, $\varphi_{0.1,5}$ and $\varphi_{0.1,10}$ is calculated at the start of an epoch. The binning width (b_w) was set to 0.1 in all three cases to ensure that all three measures had 10 or more bins, as recommended by Rakitianskaia and Engelbrecht (2015a). The minimum of $\varphi_{0.1,1}$, $\varphi_{0.1,5}$ and $\varphi_{0.1,10}$ was used, because $\varphi_{0.1,1}$, $\varphi_{0.1,5}$ and $\varphi_{0.1,10}$, will not necessarily saturate at the same time. Eq. (12) is generally applied to a set of patterns, thus for the purpose of this article two saturation measures were utilised: φ_g, which is $\varphi_{0.1}$ applied to the generalisation set, and φ_m, which is $\varphi_{0.1}$ applied to the memory set.

8.4. Accuracy performance measures

MSE, given by Eq. (11), and percentage correct classifications (PCC) were used evaluate accuracy performance (Twomey and Smith, 1995; Rakitianskaia, 2011):

$$PCC = \frac{\sum_{p=1}^{|D|} y_p}{|D|} \times 100 \tag{14}$$

where y_p is defined as

$$y_p = \begin{cases} 1, & \text{if } \vec{t}_p = \vec{o}_p \\ 0, & \text{otherwise} \end{cases} \tag{15}$$

Note that MSE measures the error magnitude, whereas PCC measures the correct pattern classification (Twomey and Smith, 1995).

The accuracy of the classifiers was measured using the following five metrics:

1. MSE at the start of an epoch, MSE_g.
2. MSE based on the memory set, MSE_m. MSE_m was measured to determine how well the classifiers remembered concepts learnt from the last 30 patterns in terms of MSE.
3. MSE at the end of an epoch, MSE_t.
4. PCC at the start of an epoch, PCC_g.
5. PCC based on the memory set, PCC_m. PCC_m was measured to determine how well the classifiers remembered concepts learnt from the last 30 patterns in terms of PCC.

Note that PCC based on the training set was not measured, because PCC was not used to train the classifiers.

8.5. Structural complexity performance measures

Structural complexity refers to the number of neurons and synapses in a FFNN (Guan and Li, 2001; Liu et al., 2002; Bosman et al., 2018; Rakitianskaia and Engelbrecht, 2014b; Engelbrecht, 2001). Because of regularisation, the proposed classifiers change the structural complexity by reducing weight values towards zero. Thus, the optimal structural complexity and effective structural complexities need to be compared.

The optimal structural complexity was considered to be the structural complexity of the optimal architectures presented in Table 3. The effective structural complexity of a 3-layer FFNN can be calculated if a smaller architecture that has similar accuracy can be found (Engelbrecht, 2001). Pruning algorithms are well-suited for this purpose. The effective structural complexity was calculated by adapting the variance nullity pruning algorithm by Engelbrecht (2001).

Variance nullity testing is a statistical hypothesis testing technique, which tests whether the effect that a change in the weight of a synapse has on the outcomes of a ANN, i.e. the *sensitivity* of a synapse, varies statistically significantly across a set of patterns (Engelbrecht, 2001). If the variance of the sensitivity is not significant, i.e., the test statistic is less than the critical value, then the synapse is considered to be irrelevant and can be pruned (Engelbrecht, 2001). Note that the memory set was used for these performance measures. The reader is referred to the work by Engelbrecht (2001) for more information on the variance nullity pruning algorithm and its implementation.

This article used the variance nullity testing in the following way: The effective number of synapses (n_{s_e}) was calculated by counting the number of weights that do not have a test statistic below the critical value. The effective number of neurons (n_{n_e}) was determined by pruning the FFNN using the following two steps:

1. First, the irrelevant synapses were removed in such a way that the outputs of the FFNN will always produce values. In other words, each output must at a minimum be connected to either an input neuron or a bias neuron, directly or indirectly.
2. Second, all neurons that are not outputs, and do not have any incoming and outgoing synapses left after the first step, must be removed.

The number of remaining neurons is n_{n_e}. The pruning algorithm, however, needs to be configured in terms of α_v, the confidence level, and σ_0^2, the maximum amount of variance that an irrelevant synapse can display. Based on the recommendations made by Engelbrecht (2001), σ_0^2 was set to 0.0001 and α_p was set to 0.01. Furthermore, to be effective, the pruning algorithm requires a set of patterns. In this article, the pruning algorithm made use of the memory set and was executed at the end of every epoch to determine n_{s_e} and n_{n_e}.

Using the above method for determining the effective structural complexity, the structural complexity of the classifiers was measured by the following two oversize ratios:

1. The *hidden neuron oversize ratio*, calculated as
$$n_{h_{or}} = \frac{n_{h_e}}{n_{h_o}} \tag{16}$$
where n_{h_e} is the number of hidden layer neurons obtained by the adapted variance nullity pruning algorithm, and n_{h_o} is the number of hidden layer neurons, including bias neurons, in the optimal architecture for the particular problem domain. Because of the oversize factor of 2, $n_{h_{or}} \in [0, 2]$. If $n_{h_{or}} = 1$, then the optimal and effective architecture contained the same number of hidden neurons.
2. The *hidden neuron oversize ratio*, calculated as
$$n_{s_{or}} = \frac{n_{s_e}}{n_{s_o}} \tag{17}$$
where n_{s_e} is the number of synapses in the architecture obtained by the adapted variance nullity pruning algorithm, and n_{s_o} is the number of synapses in the optimal architecture for the particular problem domain. Because the oversize factor of 2 also doubled the number of synapses, $n_{s_{or}} \in [0, 2]$. If $n_{s_{or}} = 1$, then the optimal and effective architecture contained the same number of synapses.

8.6. Computational complexity performance measures

The lower bound of the computational complexity of the classifiers to process one pattern was estimated as follows:

$$\Omega_{FFNN} = 2n_a + n_s \tag{18}$$

where the number of synapses represented by n_s, and the number of activated neurons, n_a, is given by

$$n_a = n_n - (n_i + n_b)$$

where n_b, n_i, and n_n represents the number of bias neurons, the number of input neurons, and the number of neurons, respectively.

The lower bound of the effective computational complexity of the FFNNs, Ω_{FFNN_e}, was calculated by applying Eq. (18) to the effective models produced by the adapted variance nullity pruning algorithm at the end of every epoch.

The effective reduction in computational complexity due to the learning algorithm was derived from Ω_{FFNN} and Ω_{FFNN_e} as follows:

$$\Omega_r = 100 \times \frac{\Omega_{FFNN} - \Omega_{FFNN_e}}{\Omega_{FFNN}} \tag{19}$$

where Ω_r represents the percentage by which the learning algorithm reduced the computational complexity of the classifier to process one pattern. The remainder of this article refers to Ω_r as the *complexity reduction measure*.

8.7. Overfitting performance measures

Overfitting can be detected by the trigger,

$$MSE_g > \bar{MSE}_g + \sigma_{MSE_g} \tag{20}$$

where $\bar{MSE}_g$ is the moving average of MSE_g and σ_{MSE_g} is the standard deviation of the MSE_g used to calculate $\bar{MSE}_g$ (Engelbrecht, 2007).

The moving average period for $\bar{MSE}_g$ is specified in epochs. The larger the moving average period is, the less sensitive the detection method becomes to fluctuations in MSE_g, because the moving average is including more generalisation errors. Large moving average periods can result in false negatives when detecting overfitting. On the other hand, the smaller the moving average period is, the more sensitive the method becomes. Too small moving average periods can lead to false positives when detecting overfitting. The moving average period thus controls the overfitting detection sensitivity (Engelbrecht, 2007).

Two disadvantages of Eq. (20) are that the equation does not consider the training error, MSE_t, and that the equation introduces an additional control parameter that is problem dependent (Engelbrecht, 2007).

Röbel (1994) suggested an alternative overfitting detection trigger which is defined as

$$\rho(t) > \varphi_\rho(t) \tag{21}$$

where $\rho(t)$ is called the *generalisation factor* at epoch t, and $\varphi_\rho(t)$ is the threshold at epoch t. The two terms are defined as follows (Röbel, 1994):

$$\rho(t) = \frac{MSE_g(t)}{MSE_t(t)} \tag{22}$$

with

$$\varphi_\rho(t) = min\{\varphi_\rho(t-1), \bar{\rho} + \sigma_\rho, 1.0\} \tag{23}$$

where the moving average of ρ is given by $\bar{\rho}$, and σ_ρ is the standard deviation of the moving average (Röbel, 1994; Engelbrecht, 2007). The overfitting detection approach therefore results in overfitting being detected when the generalisation error is larger than the training error, or when the two errors are moving significantly away from each other (Röbel, 1994; Engelbrecht, 2007).

The above measures were used in this article to create two overfitting metrics as follow:

1. The boolean result of the overfitting constraint equation (21), O_ρ, at the end of every epoch. This measure returned 1 if a classifier was overfitting for an epoch, otherwise 0 was returned.
2. The boolean result of the overfitting constraint equation (20), O_{MSE_g}, at the start of every epoch. This measure returned 1 if a classifier was overfitting for an epoch, otherwise 0 was returned.

Both O_ρ and O_{MSE_g} make use of moving averages. To ensure that the moving average is smooth enough for the data stream in question, the moving average period should be chosen in accordance to the length of the data stream. A moving average period of 3% the length of the data stream was used. This allowed the measures to be comparable across different SDCPs, regardless of the length of the data streams.

8.8. Control parameter impact on performance measures

The more tunable control parameters there are, the more parameter configurations need to be evaluated during parameter tuning. Another factor is the size of the value set used for each control parameter. The larger a value set, the more parameter configurations there are that need to be tested. The number of control parameter configurations tested for a classifier ($|D_c|$) was calculated as follows:

$$|D_c| = \prod_{i=1}^{n_c}(|\mathcal{V}_i|) \tag{24}$$

where n_c is the number of tunable control parameters, and $\mathcal{V}_i$ is the set of potential values for the ith control parameter.

The impact of the control parameters on saturation, accuracy, and complexity performance of the classifiers was analysed by comparing the differences in the performance measures to the differences in the number of control parameter configurations of the classifiers.

8.9. Weight distribution performance measures

The regularisation algorithms maintain model complexity by altering the weight values (Bosman et al., 2018; Weigend et al., 1990). This article therefore recorded the weights frequency distribution during each epoch.

The weights frequency distribution, Ξ_w, was constructed by binning the weights using a binning width at the end of every epoch. The frequency of each bin was calculated by counting the number of weights that fall into the range of that bin (Rakitianskaia and Engelbrecht, 2015a, 2014b). The binning width was set to 0.1 for the weights in the range $[-1, 1]$, and 1 for the weights in the ranges of $[-5, -1)$ and $(1, 5]$. Any weights larger or smaller than the covered ranges were counted under their nearest bin. These bin widths were used because the values of weights in a FFNN with regularisation tended to be less than one, and a saturated FFNN tended to have weight values that were significantly larger than one (Rakitianskaia and Engelbrecht, 2014b; Krogh and Hertz, 1991; Bosman et al., 2018; Rakitianskaia and Engelbrecht, 2015b,a).

8.10. Swarm diversity performance measures

Swarm diversity describes how different the particles' positions in the swarm are, i.e. the spread of the particles in the N-dimensional search space (Engelbrecht, 2007; Olorunda and Engelbrecht, 2008). Swarm diversity indicates the exploration–exploitation states of a PSO, and provides a means to analysis how a PSO algorithm adapts to changes over time (Olorunda and Engelbrecht, 2008; Rakitianskaia, 2011; Blackwell et al., 2008). In the context of PSOs, exploration manifests in diverse swarms and exploitation manifests in non-diverse swarms (Olorunda and Engelbrecht, 2008; Rakitianskaia, 2011).

Swarm diversity, $\mathcal{D}$, can be measured as the average Euclidean distance between particles and the swarm's centre, as follows (Olorunda and Engelbrecht, 2008):

$$\mathcal{D} = \frac{\sum_{i=1}^{n_p}\sqrt{\sum_{n=1}^{N}(x_{ij} - \bar{x}_j)^2}}{n_p} \tag{25}$$

where n_p is the number of particles in the swarm; x_{ij} is the jth element of the position vector of particle i in the swarm; and $\bar{x}_j$ is the jth element of the average position vector calculated over all the particle positions in the swarm.

9. Benchmark process

Because the empirical study in this article compares different classifiers, the control parameters of the classifiers needed to be optimised, and the classifiers themselves needed to be benchmarked. Section 9.1 describes the method used to optimise the control parameters of the classifiers. Section 9.2 describes how the performance of the classifiers was evaluated.

Table 4
Control parameter values that were tested.

Parameter	Value sets
	BP parameters
α	$[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$
η	$[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]$
	QPSO parameters
n_p	30
U	Linear decreasing
r	$[0.1, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 3.5, 5.0]$
r_1 and r_2 range	$[0, 1]$
c_1 and c_2	1.496180
ω	0.729844
	WD parameters
λ_r	$[0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5]$
	WE parameters
λ_r	$[0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5]$
w_0	$[0.005, 0.0075, 0.01, 0.025, 0.05, 0.1, 0.5, 1.0]$

9.1. Control parameter tuning methodology

Each of the classifiers used in the investigation had control parameters that required to be tuned. Various studies (Bosman et al., 2018; Rakitianskaia and Engelbrecht, 2014b; Harrison et al., 2015; Eberhart and Shi, 2000; LeCun et al., 1998) have indicated good ranges for the control parameters. These ranges were used to construct the value sets for the purpose of control parameter tuning. Furthermore, it was assumed that the control parameters were dependent on each other and the problem domain. To address these dependencies the tuning process tested all of the control parameter configurations for each classifier on each of the streamed benchmark problems.

Because the classifiers made use of stochastic learning algorithms, the collective mean of MSE_g, $\bar{C}_{MSE_g}$, for each combination was sampled 30 times to ensure a valid statistical sample (Helbig and Engelbrecht, 2013). The optimal parameter configuration for a particular classifier in particular problem domain was therefore the parameter configuration that resulted in the lowest average $\bar{C}_{MSE_g}$ for the streamed benchmark problem of the problem domain.

The control parameter values sets used by the control parameter tuning process are listed in Table 4. The BP value sets were used by the BP classifiers. The QPSO value sets were used by the QPSO classifiers. The WD value sets were used by the WD-specific parameters in BP-WD and QPSO-WD. The WE value sets were used by the WE-specific parameters in BP-WE and QPSO-WE.

The architecture control parameters n_i, n_h, and n_o were set per problem domain as discussed in Section 7. The control parameters w, c_1, c_2, d, and n_p were fixed because the values in Table 4 have been found to be optimal or *de facto* in ANN and PSO literature (Eberhart and Shi, 2000; Engelbrecht, 2007; Harrison et al., 2015; Rakitianskaia and Engelbrecht, 2014a; van Wyk and Engelbrecht, 2016). The value sets of the remaining control parameters were selected in a way to ensure that the ranges of the control parameter values, suggested by literature, were uniformly covered. Values for α and η were restricted to the range of $[0, 1]$ as suggested by BP literature (Engelbrecht, 2007; Ertel, 2011; LeCun et al., 1998). The ranges for α and η were covered by using increments of 0.1.

QPSO literature suggests the use of the linearly-decreasing distribution with small r values should be used when sampling the positions of quantum particles (Harrison et al., 2015). This prevents potentially very random position vectors from causing the swarm to search haphazardly (Harrison et al., 2015). The proposed range, thus, focused mostly on small values. Harrison et al. (2015) investigated optimisation problems and not classification problems, some large values were included in the value set of r.

Table 5
Benchmark parameters for classifiers.

Parameter	Problem				
	Hyperplane	*Sphere*	*Thresholds*	*SEA*	*Electricity*
			BP-N		
α	0.1	0.2	0.1	0.1	0.1
η	0.1	0.1	0.1	0.1	0.3
			BP-WD		
α	0.1	0.2	0.1	0.1	0.1
η	0.1	0.1	0.1	0.1	0.3
λ_r	0.0001	0.0001	0.0001	0.0001	0.0001
			BP-WE		
α	0.1	0.2	0.1	0.1	0.1
η	0.1	0.1	0.1	0.1	0.1
λ_r	0.01	0.0001	0.0005	0.0001	0.01
w_0	0.005	0.05	0.5	0.05	0.05
			QPSO-N		
r	5.0	5.0	0.1	5.0	5.0
			QPSO-WD		
r	0.1	0.1	0.1	0.1	0.25
λ_r	0.5	0.01	0.01	0.001	0.01
			QPSO-WE		
r	0.1	0.1	0.25	0.1	0.25
λ_r	0.05	0.01	0.01	0.01	0.1
w_0	0.5	0.5	0.5	0.1	1.0

To allow comparison between the classifiers using WD and WE, the same regularisation coefficient (λ_r) value set was used for BP-WD, QPSO-WD, BP-WE and QPSO-WE. Bosman et al. (2018) recommended that, when using weights elimination, λ_r should use the range $[0.001, 0.1]$. Hence, the value set chosen for λ_r focused more around the range $[0.001, 0.1]$, but allowed for values outside the range also to be considered, because WD might require such values for SDCPs. Furthermore, Bosman et al. (2018) recommended that w_0 should be set to 0.01. Hence, the value set chosen for w_0 focused on values around 0.01.

The optimal parameter configurations found for each classifier in each problem domain are presented in Table 5.

9.2. Benchmarking methodology

Next, the six classifiers were benchmarked using the optimal parameter configurations in Table 5. The benchmarking simulations measured the performance of the six classifiers on the 80 benchmark problems, using the performance measures discussed in Section 8.

Each performance measure was sampled 30 times per *classifier-benchmark pair*, i.e. the combination of a classifier and a benchmark problem. Performance measures were calculated for each epoch in each run, for each classifier-benchmark pair.

10. Analysis of regularised feed forward neural networks as streamed data classifiers

The results obtained from the benchmarking process in Section 9.2 were analysed using the methods presented in Section 10.1. The results are discussed in Section 10.2. Lastly, Section 10.3 summaries the findings made in Section 10.1 by evaluating the primary objective of the study.

10.1. Result analysis methodology

The benchmarking results were analysed using descriptive statistics, Mann–Whitney-U-based ranking, and performance trends. Each of these analysis techniques are discussed in Sections 10.1.1, 10.1.2, and 10.1.3, respectively.

10.1.1. Descriptive statistics

The performance measures of each benchmark run was summarised using the collective mean approach. In the case of the weights frequency distribution (Ξ_w), each frequency bin was summarised using the collective mean approach to get the collective mean of the weights frequency distribution, $\bar{C}_{\Xi_w}$, for each run. Afterwards, the collective means for each performance measure were aggregated over the 30 runs for each classifier-benchmark pair using the descriptive statistical measures mean and standard deviation. The article notates the mean ($\bar{x}$) and standard deviation (σ) using the notation $\bar{x} \pm \sigma$.

The benchmark results were further aggregated on four additional levels:

- *Classifier-domain* level per performance measure. This aggregation level grouped results of the classifiers for a particular performance measure by the benchmark problem domains, i.e. the hyperplane, sphere, thresholds, SEA, and electricity domains.
- *Classifier-difficulty* level per performance measure. This aggregation level grouped results of the classifiers for a particular performance measure by the benchmark problem difficulties.
- *Classifier-environment* level per performance measure. This aggregation level grouped results of the classifiers for a particular performance measure by the problem environments, i.e. abrupt, progressive, and chaotic. Quasi-static was left out as there were no quasi-static benchmark problems in the benchmark problem suite that was used.
- *Classifier-measure* level per classifier. This aggregation level grouped results of the classifiers by performance measure over all the benchmark problems.

10.1.2. Mann–Whitney-U-based ranking

A series of Mann-Whitney U (MWU) pair-wise comparisons were done to see if the performance of one classifier was significantly different from another classifier on the same benchmark problem. If the two classifiers were found to be significantly different from each other, then the classifier whose performance measure had a more favourable median was considered the winner. Otherwise, the classifiers were considered tied. A two-tailed MWU test was used, because the hypothesis test was based on weather or not the performance of the classifiers differed statistically. The MWU tests were performed using a confidence interval of 95%.

The total number of wins, ties and losses for each classifier were tallied per performance measure. The classifier with the least number of losses for the performance measure was the winner and was assigned the highest rank, i.e. 1. If there was a tie between the number of losses, then the classifier with more wins was considered better. This ranking approach was adopted from Helbig and Engelbrecht (2013). The notation *winning percentage/drawing percentage/losing percentage* is used in this article to represent the MWU-based ranking results, where the percentages were each calculated as follows:

$$\frac{\text{number of wins/ties/losses}}{\text{wins} + \text{ties} + \text{losses}} \times 100$$

In the case where a rank was needed, the notation was extended as follows: *rank (winning percentage/drawing percentage/losing percentage)*. If a classifier, for example, had the result 2(40/50/10), then the classifier was ranked second best out of the six classifiers. Furthermore, the classifier won 40%, tied 50%, and lost 10% of the time in all the pair-wise comparisons between the classifier and the other classifiers in the pool. The MWU ranks should be compared to the ranks in the same row of a results table.

10.1.3. Performance trends

Trend analysis was carried out on the various performance measures. The 30 run values of a performance measure per epoch were averaged. It is possible that an epoch average could have infinitely large values, because of its performance measure, e.g. swarm diversity. In

Table 6
MWU-based ranking of the classifiers with regards to the performance measures (Wins/Ties/Losses percentages).

Measure	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
φ_g	2(65.25/16.25/18.50)	3(50.50/15.50/34.00)	**1(70.00/12.00/18.00)**	*6*(*0.00*/0.00/*100.00*)	4(52.25/6.75/41.00)	5(31.50/10.50/58.00)
φ_m	2(64.75/16.75/18.50)	3(50.25/16.25/33.50)	**1(70.25/12.25/17.50)**	*6*(*0.00*/0.00/*100.00*)	4(53.50/6.00/40.50)	5(31.25/8.75/60.00)
Saturation rank	2(65.00/16.50/18.50)	3(50.38/15.88/33.75)	**1(70.13/12.13/17.75)**	*6*(*0.00*/0.00/*100.00*)	4(52.88/6.38/40.75)	5(31.37/9.63/59.00)
MSE_g	**1(75.25/19.50/5.25)**	3(59.00/17.75/23.25)	2(64.25/19.50/16.25)	*6*(*0.00*/1.00/*99.00*)	5(29.75/3.25/67.00)	4(37.75/7.00/55.25)
MSE_m	**1(78.50/18.00/3.50)**	3(63.50/17.50/19.00)	2(64.50/18.00/17.50)	*6*(*0.50*/1.00/*98.50*)	5(21.25/3.50/75.25)	4(41.50/2.50/56.00)
PCC_g	**1(67.00/13.75/19.25)**	2(46.50/13.50/40.00)	3(39.25/14.00/46.75)	4(46.00/5.75/48.25)	*6*(*31.00*/2.00/*67.00*)	5(43.75/4.00/52.25)
PCC_m	**1(72.00/10.25/17.75)**	2(50.25/9.25/40.50)	4(44.00/9.50/46.50)	3(50.75/6.25/43.00)	*6*(*24.00*/1.50/*74.50*)	5(39.00/3.25/57.75)
Accuracy rank	**1(73.19/15.38/11.44)**	2(54.81/14.50/30.69)	3(53.00/15.25/31.75)	*6*(*24.31*/3.50/*72.19*)	5(26.50/2.56/70.94)	4(40.50/4.19/55.31)
$n_{s\,or}$	*6*(*7.50*/15.50/*77.00*)	5(15.25/16.00/68.75)	4(43.25/6.25/50.50)	**1(94.25/3.25/2.50)**	3(49.50/10.75/39.75)	2(58.75/11.25/30.00)
$n_{h\,or}$	5(6.75/20.50/72.75)	*6*(*6.00*/20.00/*74.00*)	4(27.00/15.75/57.25)	**1(99.50/0.50/0.00)**	3(58.00/9.75/32.25)	2(64.50/10.00/25.50)
Ω_r	*6*(*8.00*/14.75/*77.25*)	5(14.50/15.25/70.25)	4(43.00/6.25/50.75)	**1(94.50/4.00/1.50)**	3(50.25/9.00/40.75)	2(60.00/10.25/29.75)
Complexity rank	*6*(*7.42*/16.92/*75.67*)	5(11.92/17.08/71.00)	4(37.75/9.42/52.83)	**1(96.08/2.58/1.33)**	3(52.58/9.83/37.58)	2(61.08/10.50/28.42)
Accuracy-complexity rank	4(40.30/16.15/43.55)	5(33.36/15.79/50.84)	3(45.38/12.33/42.29)	**1(60.20/3.04/36.76)**	*6*(*39.54*/6.20/*54.26*)	2(50.79/7.34/41.86)
Overall rank	2(48.53/16.26/35.20)	3(39.03/15.82/45.15)	**1(53.63/12.26/34.11)**	*6*(*40.13*/2.03/*57.84*)	5(43.99/6.26/49.76)	4(44.32/8.10/47.58)

the case where epoch averages had a potentially broad range of values, e.g. $[10^{-300}, 10^{300}]$, a logarithmic scale was used.

Next, all trend lines were smoothed using a moving average with a period of 3% of the number of patterns in the SDCP. This allowed a metric's performance trends to be compared across different SDCPs.

The sample standard deviation of the moving average was also determined for D and MSE_g. Two bands were formed along each moving average trend. The *positive band* represents the moving average plus the standard deviation. The *negative band* represents the moving average minus the standard deviation. These bands represented the volatility of the performance trend. A larger channel meant more volatility in the performance trend, and vice versa.

Lastly, the Pearson correlation coefficient was used to quantify the level of linear correlation between the performance trends of the measures, where needed.

10.2. Discussion

Table 6 presents the overall rankings for the classifiers based on the results of the MWU pairwise comparisons of the saturation levels, accuracy performance, and complexity performance. The rank of a classifier for each of the three performance categories, i.e. saturation, accuracy and complexity, was determined using the average wins-ties-loses percentages of the classifier in each category. The accuracy-complexity rank of the classifier was determined by averaging the average wins-ties-loses percentages for the accuracy and complexity performance categories. The overall rank of the classifier was determined by averaging the average wins-ties-loses percentages for the three performance categories. Note that Appendix A in the supplementary material of this article presents the detailed results on which Table 6 is based.

Figs. 1 and 2 illustrate the rank categories, i.e. the saturation, accuracy, complexity, accuracy-complexity, and overall results of Table 6. Fig. 1 plots the inverted rank, i.e., a higher rank means better performance, for the five rank categories against the classifiers. Fig. 2 plots the MWU-comparison winning percentages for the five rank categories against the classifiers.

Note that, due to the sheer volume of performance trend graphs used by this analysis, the graphs have been omitted from this section. However, selected graphs are presented in Appendix B in the supplementary material of this article for the benefit of the reader.

Remarks on accuracy. BP-N was the best overall at generalising and remembering patterns in SDCPs. Furthermore, BP-N was capable of good levels of accuracy in various dynamic environments. This went against the expectations of the study, because BP is considered a static weights adjustment algorithm .

As expected, the MSE measures showed QPSO-N to have the worst accuracy on all the problem domains. On the contrary, the PCC measures showed that QPSO-N performed very well, at times surpassing the PCC results of the regularised QPSO classifiers. These contrasting observations were most likely the result of complete saturation.

The idea of complete saturation is supported by the fact that PCC results were near to 12.5% in the thresholds domain, whereas the PCC results for the other four problem domains were near to 50%. This is because the thresholds domain is the only problem domain among the five problem domains that had three target classes, while the rest had two target classes. Consider the following example:

If a two target class classifier is completely saturated, then the outputs will always be one or the other class. Thus, a saturated two target class classifier has a 50% chance of being correct. Saturation, however, can only occur after some period of training. The probability of the classifier classifying a pattern correctly should, therefore, be higher than 50%. The same idea can be applied to a three target class classifier, however, the classifier should be able to classify more than 12.5% of the patterns correctly. The 12.5% is derived by multiplying the 50% probabilities of the three mutually exclusive outputs together, i.e. $50\% \times 50\% \times 50\%$.

The above example also explains why the MSE values showed poor performance for QPSO-N, because these values were at the extreme values of zero and one most of the time.

Regularisation degraded the accuracy performance of the BP weights adjustment algorithm. However, regularisation, especially WE, improved the MSE performance of the QPSO weights adjustment algorithm for SDCPs.

The low MSE_t values of all the classifiers, except QPSO-N, showed that the classifiers were able to learn the decision boundaries from patterns in SDCPs accurately. However, the training and generalisation accuracies of these classifiers showed signs of overfitting.

The regularised QPSO classifiers were not good at remembering, whereas the BP classifiers were only able to forget some of what they had learnt. This was evident in the memory accuracy trends, i.e. MSE_m and PCC_m, that were usually worse than the generalisation accuracy trends, i.e. MSE_g and PCC_g. Furthermore, the MSE_m trends of the regularised QPSO classifiers were most of the time significantly worse than their MSE_g and MSE_t trends. Note that the MSE_t trends came close to zero in most problems, and in some cases stayed close to zero for the duration of the problem. Both BP and QPSO weight adjustment algorithms were unable to adjust the rates at which they learnt and unlearned information as needed. A possible solution would be to dynamically adjust η for the BP classifiers and λ_r for all the classifiers.

The MSE and PCC performance measures provided conflicting conclusions. This was in-line with the findings by Twomey and Smith

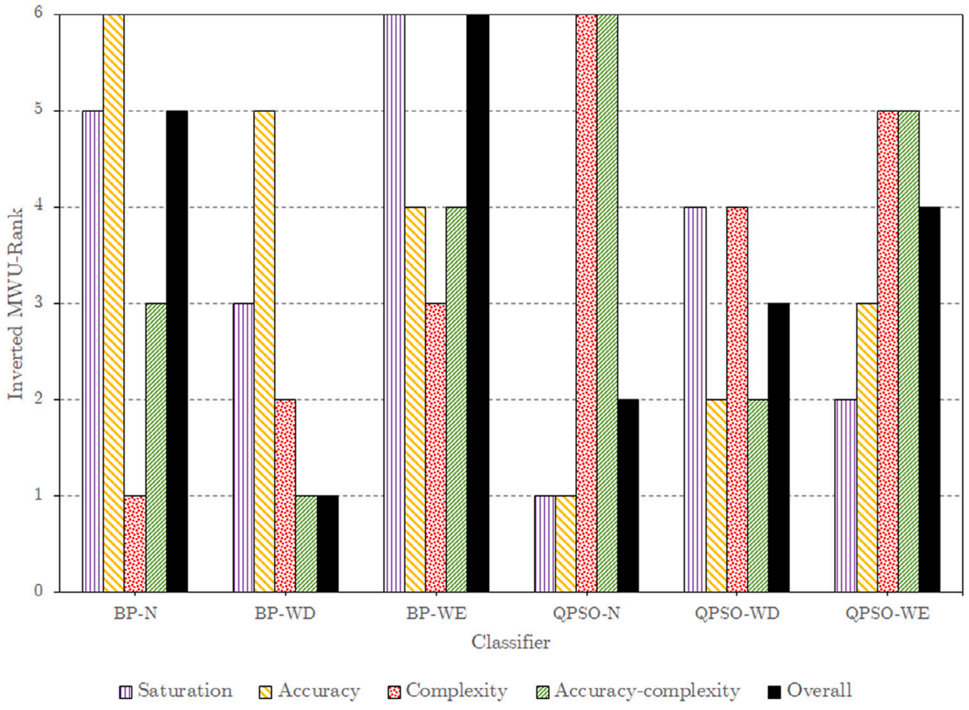


Fig. 1. Classifier versus overall inverted MWU-Rank.

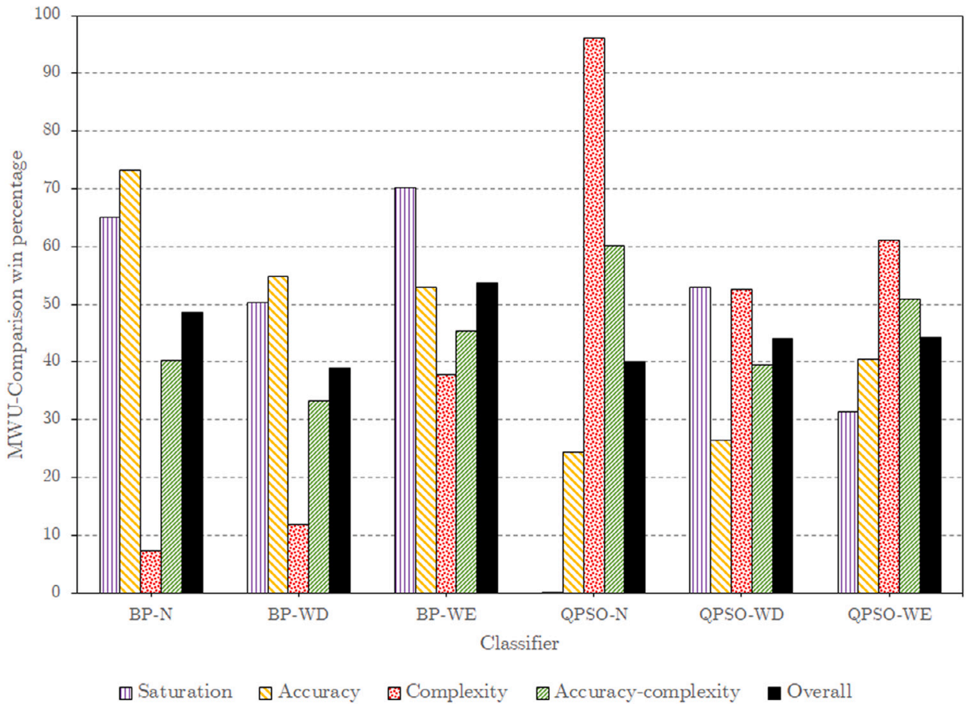


Fig. 2. Classifier versus overall MWU-Comparison winning percentages.

(1995) for static classification problems. Both measures should, therefore, be considered in studies that look at streamed data stream classifiers.

The regularised QPSO classifiers showed potential at handling noisy problems, while the BP classifiers did not. This was evident from the SEA domain, where the regularised QPSO classifiers significantly outperformed the BP classifiers according to the PCC measures, with QPSO-WD and QPSO-WE achieving very similar results. Note that the regularised QPSO classifiers did not fair so well on the real-world electricity problem.

Furthermore, high dimensional SDCPs had a detrimental effect on the accuracy of the regularised QPSO classifiers. This was not the case for the BP classifiers.

The problem difficulty classification scheme proposed by Ellis et al. (2021) was shown to be valid for SDCPs at a high-level. However, QPSO-N did not follow the classification scheme. This was due to the

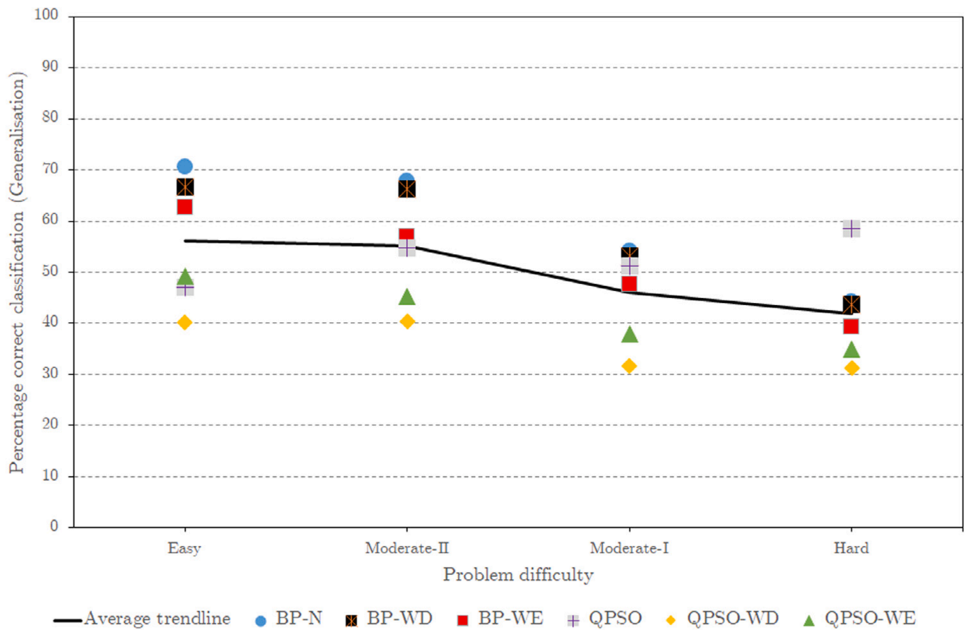


Fig. 3. PCC_g versus problem difficulty scatter plot.

complete saturation of QPSO-N. Fig. 3 illustrates the above by plotting the mean of the PCC_g in Table A.9e against the problem difficulties for each classifier.

SDCPs that have low temporal severity, e.g. abrupt, allowed classifiers to achieve the best levels of accuracy. Thus, longer environment instances allowed for better accuracy to be achieved, because there is more time to learn.

Remarks on saturation. Of the three saturation components for φ_v and φ_g, the components using an upper bound of one, i.e. $\varphi_{0.1,1_v}$ and $\varphi_{0.1,1_g}$, correlated the most with their corresponding saturation measures for the problem domains. The average Pearson correlation coefficients for φ_v and $\varphi_{0.1,1_v}$ was 0.9302 ± 0.2506. The average Pearson correlation coefficients for φ_g and $\varphi_{0.1,1_g}$ was 0.9118 ± 0.2652. The other two components, however, correlated significantly less with a maximum average Pearson correlation coefficient of 0.7317 ± 0.3506. This meant that the activation values of the classifiers were mostly in the range $[0, 1]$ most often regardless of classifier or problem domain. Using only the $\varphi_{0.1,1}$ component as a measure of saturation in the hidden neurons should, therefore, be sufficient for streamed data classifiers using ReLU activation functions.

The saturation measures, i.e. φ_v and φ_g, very closely reflected the same outcomes for most of the problem domains. The SEA problem domain, however, exhibited some minor, but negligible variations between φ_v and φ_g. The average Pearson correlation coefficient for the two measures was 0.995 ± 0.0112. The very high correlation between the two measures indicates that only one of the two measures need to be measured. Because φ_g does not violate the one-pass requirement of SDCPs, and is computationally less complex than φ_v, it should be preferred over φ_v.

The complete saturation of QPSO-N was confirmed by the results of the saturation measures. This supported the explanation as to why the QPSO-N showed abnormal accuracy performance.

The BP classifiers saturated significantly less than the QPSO classifiers the majority of the time. Thus the BP weights adjustment algorithm was less prone to saturation than the QPSO weights adjustment algorithm. However, the same was not true for the QPSO weights adjustment algorithm. This was evident from the fact that QPSO-N saturated almost completely all the time.

Regularisation helped to reduce saturation in both the regularised BP classifiers and the regularised QPSO classifiers. Regularisation, however, was more successful in reducing saturation for the regularised QPSO classifiers than for the regularised BP classifiers.

The behaviour of the regularisation approaches differed over different weights adjustment algorithms. BP-WE had the lowest levels of saturation for the BP classifiers. QPSO-WD had the lowest levels of saturation for the QPSO classifiers. However, WD caused the BP weights adjustment algorithm to become less effective at handling saturation.

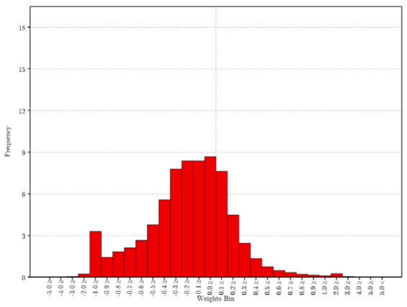
The regularised QPSO classifiers had significantly more consistent saturation levels than the BP classifiers across the problem domains, difficulties, and environments. Hence, the regularised QPSO classifiers were better at controlling saturation in the hidden neurons than the BP classifiers.

Noise caused the BP classifiers to saturate more. The presence of noise or irrelevant information increased the rate of the saturation performance trends for the BP classifiers, especially if there was prolonged exposure to patterns with these characteristics. On the other hand, noise caused the QPSO classifiers to saturate less.

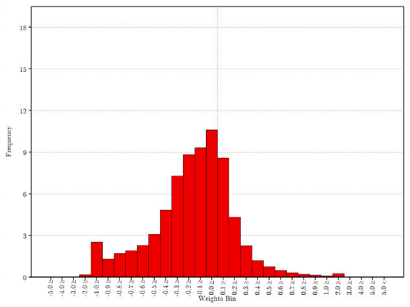
The performance trends of the BP classifiers also showed that problem dimensionality effects saturation. That is, an increase in dimensionality leads to saturation trends rising faster to higher levels, and becoming more volatile. Lower dimensionality SDCPs were better for the regularised QPSO classifiers than for the BP classifiers, and allowed the regularised QPSO classifiers to perform closer to the BP classifiers without the threat of rising saturation.

Furthermore, the problem difficulty results show that the more patterns there were in the SDCP, the more saturated the classifiers became. This relationship, however, was much more prevalent for the BP classifiers than for the QPSO classifiers. On the other hand, the problem environment results showed that as temporal severity decreased and spatial severity increased, the more saturated the classifiers became. The BP classifiers were more susceptible to this phenomenon than the QPSO classifiers.

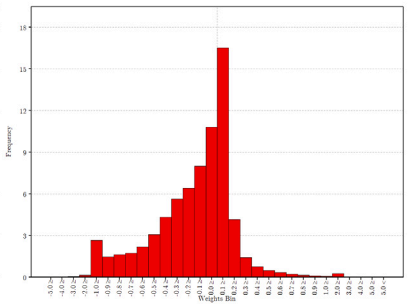
Remarks on weights. Fig. 4 presents the aggregated weights frequency distributions (Ξ_w) for the classifiers. The scale of the y-axis for the graphs differ as follows. The BP classifiers all have the same y-axis scale, and the QPSO classifiers all have the same y-axis scale. Note that small artefacts occurred when the bin intervals changed from 0.1 to 1. However, this is to be expected because the new bin interval is ten times greater than the previous bin interval. BP-N had a broad normal distribution with a negative mean. BP-WD did not alter this distribution drastically, but BP-WE did. BP-WE shifted most of the distribution to the negative side. The regularised BP classifiers in general had narrow



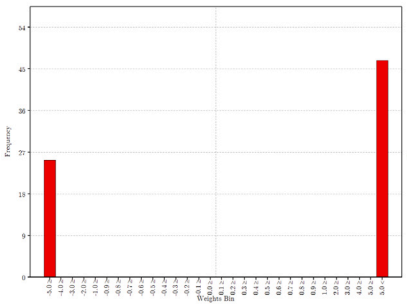
(a) BP-N



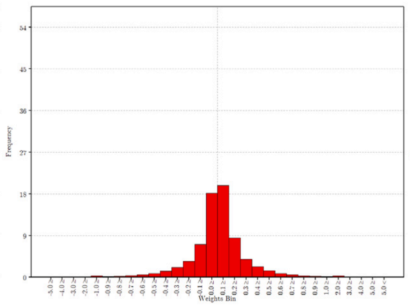
(b) BP-WD



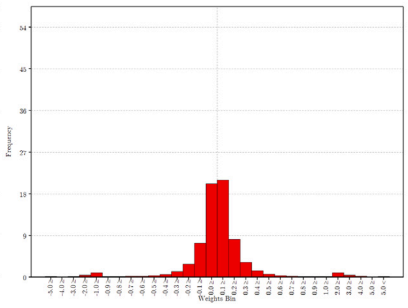
(c) BP-WE



(d) QPSO-N



(e) QPSO-WD



(f) QPSO-WE

Fig. 4. Weight frequency distribution graphs for the classifiers.

weight distributions, that were skewed to the negative side. Furthermore, BP-WE saturated completely on the electricity A4 problem. The average weight magnitudes for the electricity domain, however, was very small.

The above observations show a large number of zero and negative weights for the BP classifiers. Too many zero and/or negative weights will result in net input signals that are zero or less. Thus, neurons with net input signals of zero or less will have activation values with zero gradients due to the ReLU activation function. The BP weight adjustment algorithm will *lose control over the weights*, i.e. not be able to adjust the weights that result in such net input signals, because the algorithm is gradient-based. Once control is lost the BP algorithm will not be able to adjust the weight again. This phenomenon is known as *dying ReLU* and is typically found in deep neural network (DNN) that are trained with gradient-based training algorithms (Lu et al., 2020). Because SDCPs are dynamic and unbounded in the real-world, weights need to be constantly adjusted. The BP classifiers would eventually "die" and fail to learn the SDCPs. It can be argued that regularisation adds additional training information, i.e. model complexity, which will still provide a gradient in the case of weights being negative. However, the optimisation will only be focused on optimising model complexity and not accuracy. Furthermore, regularisation only drove weights faster to zero. Thus adding regularisation to the BP weights adjustment algorithm did not help. A possible solution for the problem of losing control over the weights could be to reinitialise the weights when the majority of weights are less than or equal to zero, or when accuracy starts to worsen. Another possible solution suggested by Lu et al. (2020) for dying ReLU neural networks is to initialise the weights using an asymmetric initialisation process, instead of the traditional symmetric initialisation processes.

Saturation in the BP classifiers was, therefore, caused by a loss of control over the weights, because of the combined effect of zero gradients and dynamic environments. The ReLU FFNN classifiers must, therefore, not use the BP weights adjustment algorithm when dealing with SDCPs.

QPSO-N, on the other hand, had all its weights distributed to the edges of negative and positive portions of the distributions. This is a definitive confirmation of complete saturation. The regularised QPSO classifiers both changed the weight distribution of QPSO-N to a narrow normal distribution around zero. The weight distribution of QPSO-WE was narrower than the weight distribution of QPSO-WD, but the weight value range of QPSO-WE was greater.

The above observations showed that the QPSO weights adjustment algorithm was immune to the zero gradients of the ReLU activation functions. QPSO-N, however, still suffered from saturation due to an uncontrolled explosion of weights. The difference between the QPSO-N and the regularised QPSO classifiers was that the regularised QPSO classifiers had more information about the search space to control the weights, namely complexity performance. Hence, the *lack of search space information*, i.e. boundaries, was the cause of saturation for the QPSO classifiers.

Remarks on overfitting. The raw MSE performance trends (refer to Appendix B in the supplementary material of this article) revealed extremely volatile trends that would either bounce between zero and very high levels, or remain at zero for several epochs. This was a side effect of the classifiers learning each pattern for only one epoch, i.e. the one-pass requirement.

The overfitting measure O_ρ produced meaningless values, because of the division by zero that occurred due to the presence of zeros in the raw MSE_t trend. The overfitting measure O_ρ, therefore, cannot work with SDCPs. An alternative overfitting measure that considers MSE_t and MSE_g needs to be developed. A possible starting point can be looking at the difference between MSE_t and MSE_g. Another starting point could be to add a very small negligible value to MSE_t, e.g. 1×10^{-16}.

The zero MSE_t values did not present problems for the overfitting measure, O_{MSE_g}. The O_{MSE_g} results, however, left much to be desired. The biggest concern is that O_{MSE_g} suffered from many false positives. The main reason for the false positives was the high volatility of the MSE_g values. This can be seen in the large standard deviation bands in the moving average trend graphs.

Another issue with O_{MSE_g} was that it only relied on the moving average of MSE_g and did not consider MSE_t. This lead to false positives in the cases where the trend volatility was high but the directions in which MSE_g and MSE_t moved were the same, i.e. correlated.

The average Pearson correlation coefficient for MSE_g and MSE_t was 0.6137 ± 0.3711. Thus when MSE_g increased or decreased so too did MSE_t most of the time. Hence, the classifiers were generally not overfitting, but O_{MSE_g} indicated that the classifiers were overfitting. A possible solution to this problem is to compare the direction of the trends of both MSE_g and MSE_t.

O_{MSE_g} trends for the completely saturated QPSO-N, which experienced overfitting early on, revealed a further issue with O_{MSE_g}. That is, O_{MSE_g} did not consider if overfitting happened in prior environments, which lead to false negatives. A possible solution could be to integrate the saturation measure into the overfitting measure. Another possible solution is to convert the measure into a flip-flop operator, i.e. stays in a state until a certain event occurs.

The findings above indicate that an alternative overfitting measure that mitigates problems of division by zero, only considering MSE_g, and dynamic environments is required in order to detect overfitting in SDCPs successfully.

Remarks on complexity. The synapse oversize ratio ($n_{s_{or}}$) and the complexity reduction measure (Ω_r) produced the same rankings for the problem domains, problem difficulties, and problem environments. Therefore, either measure can be used to estimate the lower bound computational complexity of a FFNN. Furthermore, the values for $n_{s_{or}}$ for BP-N exceeded 1. Architecture selection should therefore be employed by FFNN streamed data classifiers.

The mean values for $n_{s_{or}}$ were significantly lower than the mean values of hidden neuron oversize ratio ($n_{h_{or}}$) for all the problem domains, problem difficulties, and problem environments. Thus, as expected, synapses have a higher chance of being irrelevant than hidden neurons. Architecture selection algorithms for SDCP should, therefore, function at a synapse level rather than at a neuron level.

The above observations also provided evidence that the pruning algorithm proposed by Engelbrecht (2001) was able to determine the effective model for the FFNN classifiers.

The BP classifiers had the worst complexity performance results, therefore, the BP weights adjustment algorithm was not naturally proficient at maintaining complexity. On the other hand, QPSO-N had the best complexity performance, however, QPSO-N was completely saturated. Complete saturation thus improves complexity performance, because complete saturation degrades information capacity thereby rendering various hidden neurons irrelevant.

WE helped both weight adjustment algorithms, i.e. BP and QPSO, to achieve better complexity performance than WD. However, not all regularisation approaches aided the complexity performance of the BP weights adjustment algorithm, for example WD.

QPSO-WD and QPSO-WE were outperformed by QPSO-N with regards to complexity performance. However, both classifiers maintained reasonable saturation levels. Thus, regularisation helped to boost the complexity performance of the QPSO weights adjustment algorithm without allowing unwanted saturation in the hidden neurons.

WD and WE regularisation significantly improved the performance trends of the QPSO weights adjustment algorithm through the stabilisation of the complexity reduction trend. The performance trends suggested that only saturation in the hidden neurons that was necessary for accuracy performance was preserved by regularised QPSO classifiers. QPSO-WE was the best at distinguishing between necessary and unnecessary saturation for most problems. QPSO-WD, however, reduced all saturation regardless. This is mainly due to the difference in the aggressiveness of the weight penalisation for the two regularisation terms.

Generally, complexity and saturation levels correlated to the extent that the more saturated the hidden neurons of the classifiers were, the less complexity they required. For instance, the regularised QPSO classifiers had an average Pearson correlation coefficient between φ_g and Ω_r of 0.6616 ± 0.3112.

Noise affected complexity performance. That is, the more a classifier captured noise, the less the classifier could reduce model complexity. This could be seen in the complexity results of the BP classifiers for the SEA problems begin worse than for other problems. The complexity performance of the regularised QPSO classifiers did not degrade when faced with noise in the SEA problems, because regularisation unlearned the noise as per their accuracy performance.

An increase in the problem difficulty resulted in a decrease in the complexity performance of the classifiers, and vice versa. Thus, the problem difficulty classification scheme proposed by Ellis et al. (2021) could potentially be useful in determining a SDCP classifier's complexity performance at a high-level.

Furthermore, the more times a unique input–target pair was repeated, the more fitted the classifiers became, and accordingly, the more complexity was reduced. The degree of fitting of the classifiers was also significantly influenced by temporal severity, but not significantly by spatial severity.

Abrupt environments resulted in the best complexity performance, and progressive environments the worst.

The dimensionality of a problem had inconsistent effects on complexity performance between the BP classifiers and the QPSO classifiers. That is, higher dimensional problems, i.e. the hyperplane and electricity domains, lead to better complexity performance for the BP classifiers. On the other hand, higher dimensional problems, lead to worse complexity performance for the QPSO classifiers.

Lastly, the complexity results and correlation coefficients showed that there was no consistent relationship between accuracy and complexity performance.

Remarks on control parameters. Table 7 presents the impact of the control parameters for each classifier on the control parameter tuning process. The table includes the number of control parameters of each classifier (n_c), and the number control parameter configurations tested ($|D_c|$). To determine the impact made by these control parameter numbers, the table also includes the five MWU-based rank categories that were presented in Table 6.

Fig. 5 illustrates n_c in terms of the red bars, and $|D_c|$ in terms of the black line. Note that the y-axis values on the left side of the graph are for the red bars and the values on the right side are for the black line. Figs. 5, 1 and 2 was used to determine the impact that the control parameters had on the performance of the classifiers.

The results show that QPSO-N was the classifier with the lowest number of parameter configurations, but QPSO-N was also the classifier with the worst saturation and accuracy ranks. On the other hand, BP-WE was the classifier with the most parameter configurations, with 579 times more parameter configurations than QPSO-N. The large difference, however, lead to the highest saturation rank and overall rank, but also significantly worse accuracy and complexity scores. These results suggest that the classifiers with many or very few control parameters did not provide enough gain in either the performance, or the time taken to tune the control parameters to justify the loss in the other.

QPSO-WD had the second least number of parameter configurations to test, with eight times more configurations than QPSO-N. QPSO-WD placed fourth overall and displayed average performance in the other rank categories. On the other hand, BP-WD had the second most number of parameter configurations to test, with 72 times more configurations than QPSO-N, but also had no remarkable performance statistics in any of the performance categories. This provided further support that the classifiers with too many or too few control parameters did not provide a good performance versus tuning time trade-off.

BP-N and QPSO-WE had the middle most number of parameter configurations, but also the most impressive performance. QPSO-WD had 64 times more parameter configurations than QPSO-N, and BP-N had only 9 times more parameter configurations than QPSO-N. Considering the overall ranks of BP-N and QPSO-WE, the BP-N had the best trade-off between performance and time taken to tune the control parameters, and the QPSO-WE had the second best trade-off. The results also indicate that BP-N was the most suitable classifier out

Table 7
Comparison between the control parameters and the overall performance of the classifiers.

	Classifier					
	BP-N	*BP-WD*	*BP-WE*	*QPSO-N*	*QPSO-WD*	*QPSO-WE*
n_c	2	3	*4*	**1**	2	3
$\lvert D_c \rvert$	81	648	*5184*	**9**	72	576
Saturation rank	2(65.00|16.50|18.50)	4(50.38|15.88|33.75)	**1**(**70.13**|**12.13**|**17.75**)	*6*(*0.00*|0.00|*100.00*)	3(52.88|6.38|40.75)	5(31.37|9.63|59.00)
Accuracy rank	**1**(**73.19**|**15.38**|**11.44**)	2(54.81|14.50|30.69)	3(53.00|15.25|31.75)	*6*(*24.31*|3.50|*72.19*)	5(26.50|2.56|70.94)	4(40.50|4.19|55.31)
Complexity rank	*6*(*7.42*|16.92|*75.67*)	5(11.92|17.08|71.00)	4(37.75|9.42|52.83)	**1**(**96.08**|**2.58**|**1.33**)	3(52.58|9.83|37.58)	2(61.08|10.50|28.42)
Accuracy-complexity rank	4(40.30|16.15|43.55)	*6*(*33.36*|15.79|*50.84*)	3(45.38|12.33|42.29)	**1**(**60.20**|**3.04**|**36.76**)	5(39.54|6.20|54.26)	2(50.79|7.34|41.86)
Overall rank	2(48.53|16.26|35.20)	*6*(*39.03*|15.82|*45.15*)	**1**(**53.63**|**12.26**|**34.11**)	5(40.13|2.03|57.84)	4(43.99|6.26|49.76)	3(44.32|8.10|47.58)

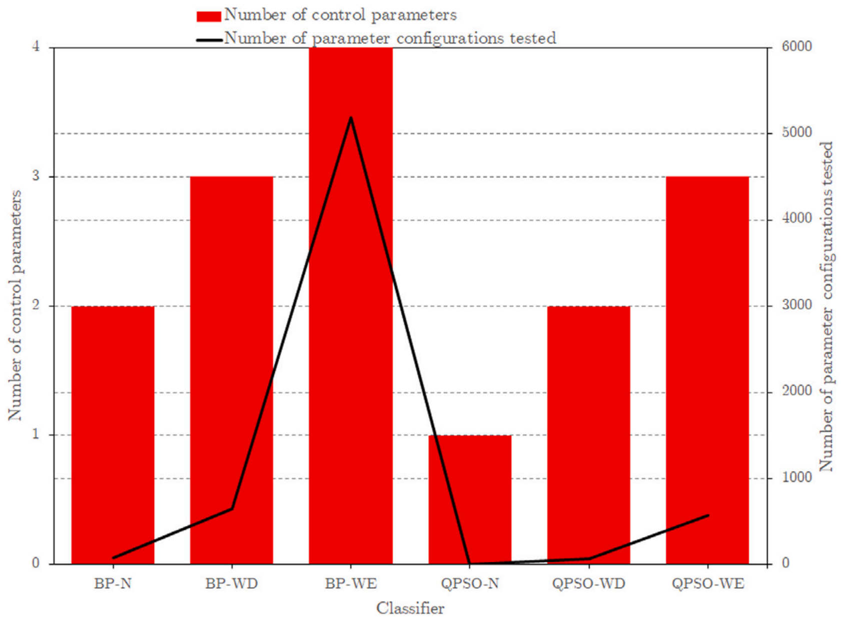


Fig. 5. Control parameter statistics for classifiers.

of the BP classifiers tested for SDCPs, and QPSO-WE was the most suitable classifier out of the QPSO classifiers tested for SDCPs.

Lastly, the following observations were made about the optimal values found for the control parameters:

The value ranges of α and η for the BP classifiers could both be reduced to $[0, 0.4]$. The finding that the BP classifiers kept their learning rate very low, showed that the BP classifiers required a very low learning rate and momentum for SDCPs in order to cope with the dynamic environments.

The values of the radius (r) control parameter for the regularised QPSO classifiers never exceed 0.25, and were mostly 0.1. The recommendation by Harrison et al. (2015) to keep the radius of QPSO small was, therefore, supported by the empirical analysis.

The ranges and values of regularisation coefficient (λ_r) control parameter was found to be problem domain and weights adjustment algorithm dependent. Furthermore, values for the weights relevancy threshold (w_0) control parameter should be selected from a wide range, i.e. $(0, 1]$, when optimising the parameter for streamed data classifiers.

Remarks on swarm diversity. The swarm diversity trends for the A1 problems (refer to Appendix B in the supplementary material of this article) of the hyperplane, sphere, thresholds and electricity domains represent the typical shape of the swarm diversity trend experienced by the regularised QPSO classifiers in these domains. Decreasing spatial severity of the A1 problems lead to more volatile trends, while decreasing the temporal severity of the A1 problems resulted in smoother trends.

Swarm diversity for QPSO-N was extreme in all of the problem domains. Swarm diversity for QPSO-WE was extreme for the noisy SEA problems. The MWU ranking indicated that swarm diversity of QPSO-WE was significantly less than the swarm diversity of the other classifiers 100% of the time for the SEA problems. This supports the findings that only a negligible number of runs for the SEA problems lead to the extreme swarm diversity. These observations showed that complete or high levels of saturation increased the weight magnitudes which in turn lead to high swarm diversity for the QPSO classifiers. Thus, complete saturation prevented the QPSO classifiers from exploiting a good solution.

Remarks on benchmark suite. All the classifiers had significantly worse performance trends for the electricity problems than for the other problems. This suggests that a real-world domain could be more challenging for the classifiers than expected. The benchmark suite should acquire more real-world problem domains to further investigate this. Additionally, the benchmark suite should also acquire artificial problems with extremely high temporal severity, because this appeared to be an fundamental characteristic of real-world problems.

Benchmark SDCP that are quasi-static in relation to the other benchmark SDCPs must also be added, because none of the benchmark problems represented quasi-static environments.

Benchmark problems with significantly more patterns should be constructed from the problem domains, because the data streams did not allow the streamed classifiers to train long enough to confirm the effect of the performance trends that progressed slowly.

10.3. Evaluation of primary objective

The primary objective of this analysis was to determine if using regularisation with QPSO to train FFNN that deal with SDCPs was effective. To evaluate the primary objective, the following nine hypotheses were considered.

Hypothesis 1, the proposed classifiers would outperform their counterparts, which were not regularised, on all of the SDCPs. The overall statistical ranks validated the hypothesis. BP-WE was considered the best BP classifier, and QPSO-WE was considered the best QPSO classifier. BP-WE, however, was 20% less accurate than BP-N, and ranked only marginally better than BP-N overall. BP-N had both the best overall accuracy rank and required significantly less parameter configuration to be tested, than BP-WE. Furthermore, BP-WD performed significantly worse than BP-N overall. Under consideration of all these aspects,hypothesis 1 was only valid for the QPSO classifiers.

Hypothesis 2, the QPSO-WD would outperform BP-WD on most of the SDCPs. The overall statistical ranks invalidated this hypothesis. QPSO-WD, however, did have a significantly better complexity performance rank than BP-WD. The saturation rank of the two classifiers also did not differ significantly. Furthermore, QPSO-WD had significantly less parameter configurations to test than BP-WD. Another problematic aspect with BP-WD was that the classifier suffered from a loss of control over its weights. Under consideration of all these aspects, hypothesis 2 was found to be partially validated.

Hypothesis 3, the QPSO-WE would outperform BP-WE on most of the SDCPs. The overall statistical ranks invalidated this hypothesis. QPSO-WE, however, did have a significantly better complexity performance rank. QPSO-WE also had significantly less parameter configurations to test than BP-WE. Like BP-WD, BP-WE also suffered from the problematic loss of control over its weights. Hypothesis 3 was therefore partially validated.

Hypothesis 4, the BP-WE and QPSO-WE would outperform their WD counterparts on all of the SDCPs. The overall statical ranks validated this hypothesis. Furthermore, the empirical analysis showed that the uniform weight penalisation approach by WD caused the under performance.

Hypothesis 5, the proposed classifiers would have lower effective model complexity than their non-regularised counterparts on all of the SDCPs. The complexity ranks validated this hypothesis for the BP classifiers, but not for the QPSO classifiers. QPSO-N suffered from complete saturation. The complete saturation rendered the QPSO-N unusable and resulted in the abnormally high complexity performance. The complexity performance of QPSO-N is, therefore, considered as meaningless. Thus, hypothesis 5 is also validated for the QPSO classifiers.

Hypothesis 6, the proposed classifiers would have lower levels of saturation than their non-regularised counterparts on all the SDCPs. The saturation ranks validated this hypothesis for the QPSO classifiers, but only partially validated the hypothesis for the BP classifiers. That is, only BP-WE had significantly lower saturation levels than BP-N. Furthermore, there were rising saturation trends amongst all of the BP classifiers due to the loss of control over their weights. This loss of control was accelerated through the use of regularisation. Thus, the empirical analysis invalidated hypothesis 6 for all the proposed BP classifiers.

Hypothesis 7, the performance of the proposed classifiers would not scale well with an increase in noise. The empirical analysis validated this hypothesis for the BP classifiers. The regularised QPSO classifiers, however, performed very well in the noisy SEA SDCPs. QPSO-WE showed the most potential for noisy SDCPs.

Hypothesis 8, the BP-WD and BP-WE would not be able to handle the dynamic environments of the SDCPs as effectively as QPSO-WD and QPSO-WE would. Between the regularised BP and regularised QPSO classifiers, accuracy performance tended to favour the regularised BP classifiers, regardless of the dynamic environment of the SDCPs. The accuracy performance trends of the regularised BP classifiers, however, showed signs of overfitting to previous environment instance, whereas the regularised QPSO classifiers did not. Furthermore, the regularised QPSO classifiers maintained stability in both saturation and complexity performance regardless of the environment of an SDCP. The empirical analysis, therefore, only had enough evidence to partially validate hypothesis 8. SDCPs with significantly more patterns need to be tested.

Hypothesis 9, the QPSO-WD and QPSO-WE would be able to maintain their swarm diversity when dealing with the SDCPs. The empirical analysis validated this hypothesis. Note that QPSO-WE did show some loss of swarm diversity control in a negligible amount of runs belonging to the SEA B1 problem.

All in all, three of the nine hypothesis were empirically validated, namely hypotheses 4, 5, 9. The remaining six hypotheses were partially validated, namely 1, 2, 3, 6, 7, 8. Furthermore, if only the QPSO regularised classifiers are considered then only three of the 9 hypotheses were partially validated, namely 2, 3, and 8, while the remaining 6 hypotheses were validated.

11. Conclusion

The article set out to investigate the application of regularised FFNNs, trained by QPSO, as classifiers for SDCPs. This article, therefore, proposed an online learning algorithm based on QPSO and regularisation to train FFNNs for SDCPs. WD and WE were used as regularisers. Because regularisation literature with regards to SDCPs is limited, a BP variant of the learning algorithm was also proposed. The learning algorithms were each applied to a 3-layer FFNNs architecture, which used ReLU activation functions and summation units. The four resulting classifiers were named QPSO-WD, QPSO-WE, BP-WD, and BP-WE.

The investigation empirically evaluated the proposed classifiers by pitting them against each other, and their non-regularised counterparts, namely QPSO-N and BP-N, on 80 benchmark problems. The benchmark problems were from an SDCP benchmark suite whose characteristics have been documented by Ellis et al. (2021).

The results of the benchmarking process were analysed using a statistically-sound approach employing descriptive statistics, MWU-based ranking, and performance trend analysis. The empirical analysis revealed the following:

The BP-N classifier learned SDCPs quickly and accurately. BP-N was the most accurate classifier of all the classifiers. BP-N, however, had a tendency to experience increasing saturation as the data stream progressed. The same increasing saturation trend occurred in BP-WD and BP-WE, with BP-WE completely saturating in some cases.

Saturation in the BP classifiers was caused by a loss of control over the weights, i.e. the learning algorithm was unable to adjust the weights. This loss of control was a result of the combined effect that the dynamic environments and the zero gradients of the ReLU activation function had on the BP algorithm. Furthermore, regularisation only accelerated the loss of control.

QPSO-WE provided well-rounded performance, and overall showed the most potential as a streamed data classifier. The main issue with the QPSO-WE was its accuracy performance. On the other hand, the aggressive weight penalisation of WD resulted in very stable complexity and saturation performance, but degraded accuracy performance significantly. The only time QPSO-WD was found worth considering was in noisy SDCPs.

QPSO-N completely saturated all the time. Unlike the BP classifiers, saturation in the QPSO classifiers was caused by a lack of search space information, i.e. boundaries imposed by additional constraints on the classifier.

With regards to complexity performance, the regularised QPSO classifiers managed to get effective architectures close to the size of the optimal architectures found by Rakitianskaia (2011). In some cases even better. The regularised BP classifiers failed to get architectures close to the optimal architectures.

Architecture selection should, therefore, be employed by fully connected FFNN streamed data classifiers, preferably with dynamic weights adjustment algorithms. Furthermore, architecture selection algorithms for SDCPs should function at a synapse-level instead of a neuron-level.

Lastly, the performance trends for accuracy measures experienced high levels of volatility. This volatility was found to be one of the main causes for the unforeseen behaviours of the classifiers. The one-pass requirement was identified as the main culprit, because it does not allow the classifier to see the entire environment instance per epoch.

The above findings showed that regularised FFNN classifiers, using a dynamic weight adjustment algorithm, such QPSO, has potential. However, several improvements need to be made to make the QPSO classifiers suitable for SDCPs.

12. Future work

The following future work is derived from the findings made during the course of this article:

The empirical analysis revealed that (i) the current set of benchmark SDCPs needs to be expanded on, e.g., add SDCPs with longer data streams; (ii) the tendency for regularised QPSO classifiers to get both low MSE and PCC errors needs to be addressed, e.g., constrain the range of the output neurons to $[0.1, 0.9]$; (iii) methods to reduce the volatility of the accuracy performance trends for SDCP classifiers need to be investigated, e.g., use fading factors to calculate the training error as suggested by Gama et al. (2009); (iv) a low computational complexity noise-filtering mechanism for SDCP classifiers is needed to curb the negative effects of noise; (v) additional forms of controlling saturation in the QPSO classifiers need to be investigated, e.g., velocity clamping and search space boundaries; and (vi) alternatives to the various components making up the proposed classifiers should be researched, e.g., the usage of other dynamic PSOs weight adjustment algorithms, using product units instead of summation units, using sigmoid and other bounded activation functions, and using other regularisers.

The pruning algorithm by Engelbrecht (2001) proved effective in determining the effective model complexity. Future research into adapting it to replace regularisation should be done. If done successfully, this would allow effective computational complexity to be realised during learning. A starting point would be to see if basing the effective reduction in complexity, Ω_r, on the generalisation set, instead of the memory set, returns similar results to what was observed during the empirical analysis.

Finding alternative overfitting measures for the streamed data classifiers is another topic for future research, because the current measures proved inadequate. If alternatives are not found, then applying algorithms to SDCP that are dependent on detecting overfitting would not be possible.

Babaeian and Mohammad (2021) expanded on the work of Babaeian et al. (2019) by investigating clustering methods to address instances where training labels are not available or missing. These scenarios are can occur with real world data streams. Adapting the approaches proposed in this article to address streamed data clustering problems is thus another area of future research.

Lastly, the streamed benchmark problem approach to tuning control parameters proved effective, but not ideal. A self-adaptive version of the proposed classifiers should, therefore, be investigated in the future. A possible starting point is looking at the self-adaptive QPSO proposed by Pamparà and Engelbrecht (2018).

CRediT authorship contribution statement

Mathys Ellis: Conceptualization, Data curation, Investigation, Methodology, Software, Visualization, Writing – original draft. **Anna S. Bosman:** Supervision, Writing – review & editing. **Andries P. Engelbrecht:** Conceptualization, Methodology, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Mathys Ellis and Anna S. Bosman reports financial support was provided by the National Research Foundation of South Africa.

Data availability

Data will be made available on request.

Acknowledgements

The financial assistance of the National Research Foundation (NRF) of South Africa towards this research is hereby acknowledged. The research has been supported by the NRF Thuthuka grant number TTK210316590115. Opinions expressed and conclusions arrived at, are those of the authors and are not necessarily to be attributed to the NRF.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.engappai.2024.108555.

References

Abdulkarim, S.A., Engelbrecht, A.P., 2021. Time series forecasting with feedforward neural networks trained using particle swarm optimizers for dynamic environments. Neural Comput. Appl. 33 (7), 2667–2683.

Aggarwal, C.C., 2007. Data Streams: Models and Algorithms, first ed. Springer.

Alpaydın, E., 2010. Introduction to Machine Learning, second ed. The MIT Press.

Babaeian, M., Francis, K.A., Dajani, K., Mohammad, M., 2019. Real-time driver drowsiness detection using wavelet transform and ensemble logistic regression. Int. J. Intell. Transp. Syst. Res. 17, 212–222.

Babaeian, M., Mohammad, M., 2021. Applying HRV based online clustering method to identify driver drowsiness. In: Proceedings of the Proceedings of the Annual Computing and Communication Workshop and Conference. IEEE, pp. 0012–0021.

Blackwell, T.M., Bentley, P.J., 2002. Dynamic search with charged swarms. In: Proceedings of the Annual Conference on Genetic and Evolutionary Computation. Morgan Kaufmann Publishers Incorporated, pp. 19–26.

Blackwell, T., Branke, J., 2004. Multi-swarm optimization in dynamic environments. In: Applications of Evolutionary Computing. EvoWorkshops. In: Lecture Notes in Computer Science, vol. 3005, Springer, pp. 489–500.

Blackwell, T., Branke, J., Li, X., 2008. Particle swarms for dynamic optimization problems. Swarm Intell. 193–217.

Bosman, A., Engelbrecht, A., Helbig, M., 2018. Fitness landscape analysis of weight-elimination neural networks. Neural Process. Lett. 48 (1), 353–373.

Chu, F., Wang, Y., Zaniolo, C., 2004. An adaptive learning approach for noisy data streams. In: Proceedings of the International Conference on Data Mining. IEEE, pp. 351–354.

Cleghorn, C.W., Engelbrecht, A., 2016. Particle swarm optimizer: The impact of unstable particles on performance. In: Proceedings of the Symposium Series on Computational Intelligence. IEEE, pp. 1–7.

Cui, Y., Surpur, C., Ahmad, S., Hawkins, J., 2016. A comparative study of HTM and other neural network models for online sequence learning with streaming data. In: Proceedings of the International Joint Conference on Neural Networks. IEEE, pp. 1530–1538.

Dennis, C., Engelbrecht, A.P., Ombuki-Berman, B.M., 2020. An analysis of activation function saturation in particle swarm optimization trained neural networks. Neural Process. Lett. 52, 1123–1153.

Domingos, P., 2012. A few useful things to know about machine learning. Commun. ACM 55 (10), 78–87.

Domingos, P., Hulten, G., 2000. Mining high-speed data streams. In: Proceedings of the SIGKDD International Conference on Knowledge Discovery and Data Mining, Vol. 6. ACM, pp. 71–80.

Duhain, J.G.O.L., Engelbrecht, A.P., 2012. Towards a more complete classification system for dynamically changing environments. In: Proceedings of the Congress on Evolutionary Computation. IEEE, pp. 1–8.

Dyer, K.B., Capo, R., Polikar, R., 2014. COMPOSE: A semisupervised learning framework for initially labeled nonstationary streaming data. IEEE Trans. Neural Netw. Learn. Syst. 25 (1), 12–26.

Eberhart, R., Kennedy, J., 1995. A new optimizer using particle swarm theory. In: Proceedings of the International Symposium on Micro Machine and Human Science. IEEE, pp. 39–43.

Eberhart, R.C., Shi, Y., 2000. Comparing inertia weights and constriction factors in particle swarm optimization. In: Proceedings of the Congress on Evolutionary Computation, Vol. 1. IEEE, pp. 84–88.

Ellis, M., Bosman, A.S., Engelbrecht, A.P., 2021. Characterisation of environment type and difficulty for streamed data classification problems. Inform. Sci. 569, 615–649.

Engelbrecht, A., 2001. A new pruning heuristic based on variance analysis of sensitivity information. IEEE Trans. Neural Netw. 12 (6), 1386–1399.

Engelbrecht, A.P., 2007. Computational Intelligence: An Introduction, second ed. John Wiley and Sons Ltd.

Engelbrecht, A.P., 2010. Heterogeneous particle swarm optimization. In: Proceedings of the International Conference on Swarm Intelligence. Springer, pp. 191–202.

Ertel, W., 2011. Introduction to Artificial Intelligence, First ed. Springer.

Fahlman, S.E., 1989. Faster-learning variations on back-propagation: An empirical study. In: Touretzky, D., Hinton, G., Sejnowski, T. (Eds.), Proceedings of the 1988 Connectionist Models Summer School. Morgan Kaufmann Publishers Incorporated, pp. 38–51.

Fernández-Redondo, M., Hernández-Espinosa, C., 2001. Weight initialization methods for multilayer feedforward. In: Proceedings of the European Symposium on Artificial Neural Networks. D-Facto public, pp. 119–124.

Gama, J., Sebastião, R., Rodrigues, P.P., 2009. Issues in evaluation of stream learning algorithms. In: Proceedings of the SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM.

Gelenbe, E., 1989. Random neural networks with negative and positive signals and product form solution. Neural Comput. 1 (4), 502–510.

Geman, S., Bienenstock, E., Doursat, R., 1992. Neural networks and the bias/variance dilemma. Neural Comput. 4 (1), 1–58.

Gies, D., Rahmat-Samii, Y., 2004. Vector evaluated particle swarm optimization (VEPSO): optimization of a radiometer array antenna. In: Proceedings of the Antennas and Propagation Society Symposium, Vol. 3. IEEE, pp. 2297–2300.

Guan, S.U., Li, S., 2001. Incremental learning with respect to new incoming input attributes. Neural Process. Lett. 14 (3), 241–260.

Gupta, A., Lam, S.M., 1998. Weight decay backpropagation for noisy data. Neural Netw. 11 (6), 1127–1138.

Harries, M., 1999. Splice-2 Comparative Evaluation: Electricity Pricing. Tech. Rep. UNSW-CSE-TR-9905, Artificial Intelligence Group, School of Computer Science and Engineering, University of New South Wales, Sydney 2052, Australia.

Harris, S.L., Harris, D.M., 2016. Digital Design and Computer Architecture, ARM ed. Morgan Kaufmann.

Harrison, K., Ombuki-Berman, B.M., Engelbrecht, A.P., 2015. The effect of probability distributions on the performance of quantum particle swarm optimization for solving dynamic optimization problems. In: Proceedings of the Symposium Series on Computational Intelligence. IEEE, pp. 242–250.

Helbig, M., Engelbrecht, A.P., 2013. Analysing the performance of dynamic multi-objective optimisation algorithms. In: Proceedings of the Congress on Evolutionary Computation. IEEE, pp. 1531–1539.

Hulten, G., Spencer, L., Domingos, P., 2001. Mining time-changing data streams. In: Proceedings of the Seventh SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 97–106.

Ismail, A., Engelbrecht, A.P., 2000. Global optimization algorithms for training product unit neural networks. In: Proceedings of the International Joint Conference on Neural Networks. IEEE, pp. 132–137.

Jadhav, A., Deshpande, L., 2017. An efficient approach to detect concept drifts in data streams. In: Proceedings of the 7th International Advance Computing Conference. IEEE, pp. 28–32.

Kennedy, J., Mendes, R., 2002. Population structure and particle swarm performance. In: Proceedings of the Congress on Evolutionary Computation, Vol. 2. IEEE, pp. 1671–1676.

Kotsiantis, S.B., 2013. Decision trees: a recent overview. Artif. Intell. Rev. 39 (4), 261–283.

Krawczyk, B., Minku, L.L., Gama, J., Stefanowski, J., Wozniak, M., 2017. Ensemble learning for data stream analysis: A survey. Inf. Fusion 37, 132–156.

Krogh, A., Hertz, J.A., 1991. A simple weight decay can improve generalization. In: Proceedings of the 4th International Conference on Neural Information Processing Systems. NIPS '91, Morgan Kaufmann Publishers Incorporated, pp. 950–957.

Kulkarni, R.V., Patil, S.H., Subhashini, R., 2016. An overview of learning in data streams with label scarcity. In: Proceedings of the International Conference on Inventive Computation Technologies, Vol. 2. pp. 1–6.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444.

LeCun, Y., L., Orr, G.B., Müller, K.R., 1998. Efficient BackProp. In: Neural Networks: Tricks of the Trade. In: Lecture Notes in Computer Science, vol. 7700, Springer, pp. 9–50.

Leskovec, J., Rajaraman, A., Ullman, J.D., 2014. Mining of Massive Datasets, second ed. Cambridge University Press.

Liang, N.Y., Huang, G.B., Saratchandran, P., Sundararajan, N., 2006. A fast and accurate online sequential learning algorithm for feedforward networks. IEEE Trans. Neural Netw. 17 (6), 1411–1423.

Liu, D., Chang, T.S., Zhang, Y., 2002. A constructive algorithm for feedforward neural networks with incremental training. IEEE Trans. Circuits Syst. I 49 (12), 1876–1879.

Losing, V., Hammer, B., Wersing, H., 2018. Incremental on-line learning: A review and comparison of state of the art algorithms. Neurocomputing 275, 1261–1274.

Lu, L., Shin, Y., Su, Y., Karniadakis, G.E., 2020. Dying ReLU and initialization: Theory and numerical examples. Commun. Comput. Phys. 28 (5), 1671–1706.

Maas, A.L., Hannun, A.Y., Ng, A.Y., 2013. Rectifier nonlinearities improve neural network acoustic models. In: Proceedings of the ICML Workshop on Deep Learning for Audio, Speech, and Language Processing.

McCulloch, W.S., Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. 5 (4), 115–133.

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. IEEE, pp. 1895–1899.

Morrison, R.W., 2003. Performance measurement in dynamic environments. In: Branke, J. (Ed.), Proceedings of GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems.

Ngom, B., Boly, A., Chiky, R., 2016. "Forgetting functions" in the context of data streams for the benefit of decision-making. In: Proceedings of the International Workshop on Computational Intelligence for Multimedia Understanding. pp. 1–5.

Olorunda, O., Engelbrecht, A.P., 2008. Measuring exploration/exploitation in particle swarm using swarm diversity. In: Proceedings of the Congress on Evolutionary Computation. IEEE, pp. 1128–1134.

Pamparà, G., Engelbrecht, A.P., 2018. Self-adaptive quantum particle swarm optimization for dynamic environments. In: Swarm Intelligence. Springer International Publishing, pp. 163–175.

Potdar, K., Pardawala, T.S., Pai, C.D., 2017. A comparative study of categorical variable encoding techniques for neural network classifiers. Int. J. Comput. Appl. 175, 7–9.

Pramod, S., Vyas, O.P., 2012. Data stream mining: A review on windowing approach. Glob. J. Comput. Sci. Technol. Softw. Data Eng. 12 (11).

Pratama, M., Angelov, P.P., Lu, J., Lughofer, E., Seera, M., Lim, C.P., 2017. A randomized neural network for data streams. In: Proceedings of the International Joint Conference on Neural Networks. IEEE, pp. 3423–3430.

Rakitianskaia, A., 2011. Using Particle Swarm Optimisation to Train Feedforward Neural Networks in Dynamic Environments (Master's thesis). University of Pretoria.

Rakitianskaia, A., Engelbrecht, A.P., 2009. Training neural networks with PSO in dynamic environments. In: Proceedings of the Congress on Evolutionary Computation. IEEE, pp. 667–673.

Rakitianskaia, A.S., Engelbrecht, A.P., 2012. Training feedforward neural networks with dynamic particle swarm optimisation. Swarm Intell. 6 (3), 233–270.

Rakitianskaia, A.S., Engelbrecht, A.P., 2014a. Training high-dimensional neural networks with cooperative particle swarm optimiser. In: Proceedings of the Congress on Evolutionary Computation. IEEE, pp. 4011–4018.

Rakitianskaia, A., Engelbrecht, A., 2014b. Weight regularisation in particle swarm optimisation neural network training. In: Proceedings of the Symposium on Swarm Intelligence. IEEE, pp. 1–8.

Rakitianskaia, A., Engelbrecht, A., 2015a. Measuring saturation in neural networks. In: Proceedings of the Symposium Series on Computational Intelligence. IEEE, pp. 1423–1430.

Rakitianskaia, A., Engelbrecht, A., 2015b. Saturation in PSO neural network training: Good or evil? In: Proceedings of the Congress on Evolutionary Computation. IEEE, pp. 125–132.

Röbel, A., 1994. The Dynamic Pattern Selection Algorithm: Effective Training and Controlled Generalization of Backpropagation Neural Networks. Tech. Rep., Institut fur Angewandte Informatik, Technische Universitat, Berlin.

Sancho-Asensio, A., Orriols-Puig, A., Golobardes, E., 2014. Robust on-line neural learning classifier system for data stream classification tasks. Soft Comput. 18 (8), 1441–1461.

Santos, A.F.C., Teles, Í.P., Siqueira, O.M.P., de Oliveira, A.A., 2017. Big data: A systematic review. In: Information Technology - New Generations: 14th International Conference on Information Technology. Springer, pp. 501–506.

Singh, Y., Chauhan, A.S., 2009. Neural networks in data mining. J. Theor. Appl. Inf. Technol. 5 (6), 37–42.

Sonoda, S., Murata, N., 2017. Neural network with unbounded activation functions is universal approximator. Appl. Comput. Harmon. Anal. 43 (2), 233–268.

Street, W.N., Kim, Y., 2001. A streaming ensemble algorithm (SEA) for large-scale classification. In: Proceedings of the Proceedings of the SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 377–382.

Telec, Z., Trawiński, B., Lasota, T., Trawiński, G., 2014. Evaluation of neural network ensemble approach to predict from a data stream. In: Proceedings of the International Conference on Computational Collective Intelligence. Springer, pp. 472–482.

Tham, C.K., 1995. On-line learning using hierarchical mixtures of experts. In: Proceedings of the Fourth International Conference on Artificial Neural Networks. IET, pp. 347–351.

Tsymbal, A., 2004. The Problem of Concept Drift: Definitions and Related Work. Tech. Rep., Trinity College, Dublin.

Twomey, J.M., Smith, A.E., 1995. Performance measures, consistency, and power for artificial neural network models. Math. Comput. Modelling 21 (1–2), 243–258.

Wang, H., Fan, W., Yu, P.S., Han, J., 2003. Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the SIGKDD International Conference on Knowledge Discovery and Data Mining, Vol. 9. ACM, pp. 226–235.

Weigend, A.S., Rumelhart, D.E., Huberman, B.A., 1990. Generalization by weight-elimination with application to forecasting. In: Proceedings of the Conference on Advances in Neural Information Processing Systems, Vol. 3. Morgan Kaufmann Publishers Incorporated, pp. 875–882.

Werbos, P.J., 1974. Beyond Regression: New Tools for Prediction and Analysis in the Behavioural Sciences (Ph.D. thesis). Harvard University, Boston.

Wessels, L.F.A., Barnard, E., 1992. Avoiding false local minima by proper initialization of connections. IEEE Trans. Neural Netw. 3 (6), 899–905.

Wilamowski, B.M., 2003. Neural network architectures and learning. In: Proceedings of the International Conference on Information Technology. IEEE, pp. TU1–TU12.

van Wyk, A.B., Engelbrecht, A.P., 2016. Analysis of activation functions for particle swarm optimised feedforward neural networks. In: Proceedings of the Congress on Evolutionary Computation. IEEE, pp. 423–430.

Zainuddin, Z., Pauline, O., 2007. Function approximation using artificial neural networks. Int. J. Syst. Appl. Eng. Dev. 1 (4), 173–178.

Zhang, C., Shao, H., Li, Y., 2000. Particle swarm optimisation for evolving artificial neural network. In: Proceedings of the International Conference on Systems, Man, and Cybernetics, Vol. 4. IEEE, pp. 2487–2490.