Wavelet Probabilistic Neural Networks

E. S. García-Treviño, Pu Yang, and J. A. Barria*

***Abstract*—In this paper, a novel Wavelet Probabilistic Neural Network (WPNN), which is a generative-learning wavelet neural network that relies on the wavelet-based estimation of class probability densities, is proposed. In this new neural network approach the number of basis functions employed is independent of the number of data inputs and, in that sense, it overcomes the well-known drawback of traditional Probabilistic Neural Networks (PNN). Since the parameters of the proposed network are updated at a low and constant computational cost, it is particularly aimed at data stream classification and anomaly detection in offline settings and online environments where the length of data is assumed to be unconstrained. Both synthetic and real-world datasets are used to assess the proposed WPNN. Significant performance enhancements are attained compared to state-of-the-art algorithms.**

***Index Terms*—Probabilistic neural network, Wavelet density estimation, Wavelet frames, Wavelet probabilistic neural networks, Data stream classification, Online learning, Non-stationary environment.**

I. Introduction

THE Probabilistic Neural Network (PNN), introduced by Specht in [1], is a well known neural network architecture widely applied in machine learning problems [2], which unify several different research areas such as function approximation, density estimation and regularisation theory. PNNs use the Bayes decision rule [2] and Parzen's approach to estimate the class density in a non-parametric manner [3]. PNNs have become an effective tool for solving classification problems due to their simplicity, efficiency, and ease of model training [4]. PNN can provide a faster one-pass learning process without the use of any back-propagation-based algorithm [5]. Moreover, in data streams classification and anomaly detection tasks, where class labels might not be balanced, the architecture of a PNN, which can handle imbalanced classification tasks, is more advantageous than feed-forward neural networks.

The Orthogonal Series Estimator (OSE) [6], is a class of non-parametric estimator whose most relevant aspect is its low computation cost while, its major drawback is its inability to estimate local properties of the underlying density distribution. A subset of the OSE, the so-called Wavelet Density Estimator (WDE), inherits the good localisation capabilities (in time and frequency) of wavelet functions, while allowing local learning and local manipulation of the estimated density (see [7] for an overview). As it is pointed out in [8], since WDEs incorporate the advantages of wavelets and multiresolution analysis, they are superior to other OSEs. Wavelet Density Estimators offer flexibility in terms of convergence and smoothness, due to the availability of several families of orthogonal wavelet functions that can be used in the estimator. However, the available orthogonal functions do not have an analytical form, therefore, recursive algorithms with a high computational burden are required for evaluating these type of functions.

The well known key drawback of standard PNN [1] is the fact that one separate neuron is needed for each training data point. Hence, for classification problems involving millions of data points, the amount of computational resources required by the network becomes prohibitive [9]. The strategies to reduce complexity within the framework of a PNN can be categorised into two main groups. The first group includes: i) the approaches based on the representation of the original dataset by a new one with smaller cardinality while maintaining the statistical properties of the original one [10], and ii) removing features that have few contributions to the classification [11]. The second group considers methods based on the complexity simplification of the estimator by reducing its corresponding number of kernel components. In [11] sensitivity analysis has been applied to a PNN to reduce the number of features from the dataset as well as the pattern neurons of the PNN. However, it is designed only for offline settings. In [12] a Fourier series-based OSE is proposed to reduce the complexities of the PWE. Unfortunately, it can only be applied to single dimensions, and hence only uncorrelated features can be fed to the algorithm. We further note that, in some real-world settings, a neural network framework might be affected by intrinsic time-delays. In this context, the stability analysis for a time-delayed system [13], [14], has been recently applied for delayed neural networks [15], [16].

In this paper, we address the key drawback of PNNs by following a radically different approach. Specifically, we reformulate PNNs in order to rely on a novel density estimation paradigm that does not require one neuron for each training sample. This novel probabilistic neural network approach is based on a new density estimation approach based on wavelets. There is a great amount of work combining wavelets with artificial neural networks. In general, we can categorise all the related research work into two main categories. The first category includes all the work in which the wavelet method is used as a feature extraction/reduction block which is then followed by a neural network classifier. Note here that the wavelet block is not part of the neural network, which means that the training of the network does not involve adjusting any parameter of the wavelet block. Note also that the neural network classifier could be a feed-forward neural network or a

The First Author was with the Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2AZ, United Kingdom. He is now with U. Maya, 29060 Tuxtla Gutiérrez, Chiapas, México.

The Second and Third Authors are with the Department of Electrical and Electronic Engineering, Imperial College London, London SW7 2AZ, United Kingdom

*Corresponding author:
Tel: +44 (0)207 594 6275
E-mail address: j.barria@imperial.ac.uk (J.A. Barria)

KDE-based probabilistic neural network. The second category of papers combines wavelets and neural networks, but it is not formulated in probabilistic terms. To the best of our knowledge, the majority of the approaches from the second category are based on the seminal work of Zhang and Benveniste [17], which uses wavelets as activation functions in a feed-forward neural network structure. We note that this wavelet network paradigm is not formulated in probabilistic terms, and hence it does not require the estimation of any class-conditional probabilities as it is for the proposed approach.

The combination of PNNs and wavelets has been well studied in applications that include fault diagnosis, fingerprint recognition, and power systems. Wavelets are used to extract features in [18], [19], or as coefficients for the densities calculations in [20]. In all these applications, wavelets are not part of the neural network structure, as the density estimation paradigm is still based on kernel density estimation. Further, discriminative learning approaches, e.g. [17], [20], use back-propagation to optimise the decision boundaries but do not estimate the class conditional probabilities. To the best of our knowledge, the proposed WPNN framework is the first attempt to formulate a probabilistic neural network based on wavelet density estimators, in which wavelets are an intrinsic part of the network involved in estimating the class-conditional probabilities of each class. In this type of network, the coefficients of the wavelet basis functions are optimised by the learning algorithm. The proposed WPNN is a generative-learning-based wavelet neural network, which relies on the wavelet-based estimation of the class probability densities.

The proposed WPNN is based on a WDE proposed by García-Treviño *et al.* 2019 [21], namely the Radial Wavelet Frame Density Estimator (RWFDE), which utilises the radial B-splines scaling function to estimate the probability density function. This novel estimator renders analytic solutions and its formulation is suitable for multidimensional analysis. It is able to deal with larger data dimensions that involve more features, and more relevantly, it has good localisation capabilities and good sparsity with low and constant computational complexity. Therefore, the RWFDE enables the application of WDE within a PNN and reduces the computational burden of the standard PNN. This WPNN formulation is constructed by replacing the Parzen Windows Estimator (PWE) used in the PNN by the RWFDE.

Two are the main contributions of this paper. The first one is the performance enhancement of PNN by reformulating the network components. Note that PNNs that rely on a density estimator approach based on scaling functions have not been investigated before due to the lack of a suitable wavelet-based approach for multidimensional problems. Therefore, WPNN is the first wavelet neural network based on generative learning concepts. The second contribution is providing a novel online learning approach for anomaly detection and data stream classification while maintaining constant computation time and low computational complexity, which is perfectly suitable for real-world applications. In this paper, we define the structure of this novel probabilistic neural network approach as well as we formulate and integrate all the necessary blocks and computational operations required in order to perform the offline and online classification of input data.

The proposed WPNN benefits from the robust approximation capabilities provided by its established multiresolution analysis framework. Moreover, the model parameters can be updated instantly once a new data point arrives, as its training and evaluation processes have constant and low computation time. Note also that for the WPNN, the number of wavelet functions is independent of the number of inputs and hence perfectly suitable for anomaly detection, fast online classification for data streams or time series, where the data is ordered by timestamps with rapid arrival rate, the length is assumed to be infinite, and its statistical properties might vary over time.

The rest of the paper is organised as follows. In Section II, the theoretical background for the proposed WPNN is briefly reviewed. The formulation of WPNN is described in Section III. Section IV includes the set of experiments to assess the performance of the proposed framework. Finally, the conclusions of this work are presented in Section V.

II. Theoretical Background

A. Probabilistic Neural Networks (PNN)

The key idea behind PNNs is the approximation of class-conditional distributions of input data by a mixture of components density estimation, where the mixture components are interpreted as probabilistic neurons. There are four layers in this type of neural network with input, pattern, summation and output units in each layer, respectively. Assuming n-dimensional input data of the form $\mathbf{x} \in \mathbb{R}^{T \times n}$, where $|T|$ refers to the cardinality of $\mathbf{x}$, and n is the data dimension. The first layer distributes input to all pattern units. The second layer forms groups associated with a specific class c_q. The activation function for the pattern units is the Gaussian basis function defined by $f_{q,l}(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}\sigma^n} \exp(-\frac{1}{2\sigma^2}(\mathbf{x} - \mathbf{x}_{q,l})^T(\mathbf{x} - \mathbf{x}_{q,l}))$ with $q = 1, \ldots, C$, $l = 1, \ldots, N_q$, where C is the total number of classes, N_q is the number of pattern units for a given class c_q, and n is the data dimension. Here, the smoothing factor σ is the parameter that controls the receptive field of the Gaussian function. Note that both the input vector $\mathbf{x}$ and the centres $\mathbf{x}_{q,l}$ of the Gaussian functions are n-dimensional. Note also that the total number of pattern units is equal to the sum of the pattern units for all classes.

The outputs of the pattern units, that belong to the same class, are connected to the summation units on the third layer corresponding to that specific class. Assuming a uniform prior distribution for each class, then each summation unit estimates the class-conditional probability density function for class c_q is defined by $\hat{p}_q(\mathbf{x}|c_q) = \frac{1}{(2\pi)^{n/2}\sigma^n} \frac{1}{N_q} \sum_{l=1}^{N_q} \exp(-\frac{1}{2\sigma^2}(\mathbf{x} - \mathbf{x}_{q,l})^T(\mathbf{x} - \mathbf{x}_{q,l}))$ where $\mathbf{x}_{q,l}$ is the l-th training vector from class c_q, $\mathbf{x}$ is the test input vector, n is the data dimension and N_q is the number of training patterns in class c_q. Note that each vector $\mathbf{x}_{q,l}$ is assumed to be the centre of a Gaussian function.

Once $\hat{p}_q(\mathbf{x}|c_q)$ has been estimated for all classes using the available training data, then the best classifier defined by the Bayesian decision rule can be obtained. In the output units on the fourth layer, a Bayesian decision rule of the form: $d(\mathbf{x}) = \arg\max_q\{P(c_q)\hat{p}_q(\mathbf{x}|c_q)\}$, with $q = 1, \ldots, C$ is applied to distinguish the class c_q associated with the input vector $\mathbf{x}$, where $P(c_q)$ is the a priori class probability, which in many applications is known or otherwise is usually assumed uniform for all classes. Note that the above decision rule minimises the probability of classification error and that the training procedure of the PNN is, in fact, the construction of the network for some available training data.

B. Wavelets and Multiresolution analysis

Wavelet analysis projects and approximates the data onto a subspace using a group of basis functions in order to provide different levels of information. In the Discrete Wavelet Transform (DWT), which is one of the most important algorithms that utilises wavelet analysis [22], data is separated into two different scales, fine-scale and coarse-scale. This multiresolution separation is done using detail coefficients and approximation coefficients to project the data onto an orthogonal dyadic basis system [23].

Multiresolution analysis is the foundation for DWT, it approximates data at different levels of resolution by orthogonally projecting them onto different spaces $\{\mathbf{V}_j\}_{j\in\mathbb{Z}}$ and $\{\mathbf{W}_j\}_{j\in\mathbb{Z}}$. Note, the space $\mathbf{W}_j$ is the complement of the space $\mathbf{V}_j$ in $\mathbf{V}_{j+1}$. The two orthonormal bases constructed by a *scaling function* ϕ and a *wavelet function* ψ are required to perform the projection. The dilated and translated version of the orthonormal basis for the space $\mathbf{V}_j$ is defined as $\phi_{j,k}(x) = 2^{j/2}\phi(2^j x - k)$, and the orthonormal basis for the space $\mathbf{W}_j$ is defined as $\psi_{j,k}(x) = 2^{j/2}\psi(2^j x - k)$.

In practice, a signal $f(x)$ at resolution 2^0 after DWT has the form $f(x) = \sum_k a_{J,k}\phi_{J,k}(x) + \sum_{j=0}^{J}\sum_k d_{j,k}\psi_{j,k}(x)$, where $a_{J,k}$ denotes the approximation coefficient at resolution 2^J, and $d_{j,k}$ denotes the wavelet coefficient at resolution 2^j. Note, that the coefficients $a_{J,k}$ and $d_{j,k}$ can be expressed as: $a_{J,k} = \langle f(x), \phi_{J,k}(x)\rangle$, $d_{j,k} = \langle f(x), \psi_{j,k}(x)\rangle$ with $J, j, k \in \mathbb{Z}$. The operator $\langle . \rangle$ denotes the inner product in the space of square integrable functions $L^2(\mathbb{R})$.

C. Wavelet Density Estimators (WDEs)

Density estimation, which is one of the fundamental problems in statistics, has been thoroughly studied in the literature, (see for example [24], [25]). WDEs is a subclass of the Orthogonal Series Estimators (OSEs) [6] that estimate an unknown square-integrable density function $p(x)$ by using a series of orthogonal basis function: $p(x) = \sum_j b_j\psi_j(x)$, where b_j is the coefficient of the j^{th} basis function, ψ_j is the basis functions in $L^2(\mathbb{R})$, and $\mathcal{J}$ is an appropriate set of indices that belongs to $\mathbb{Z}$. If $p(x)$ is a probability density function, then the coefficient b_j can be expressed as the expectation of the basis functions: $b_j = \langle p, \psi_j\rangle = \int \psi_j(x)p(x)dx = E[\psi_j(X)]$. Therefore, if there exist a group of random variables X_i, $i \in [1, N]$, with an unknown square-integrable density function $p(x)$, the approximated j^{th} coefficient in the OSEs can be expressed as $\hat{b}_j = \frac{1}{N}\sum_{i=1}^{N}\psi_j(X_i)$. Consequently, the approximated density $\hat{p}(x)$ can be calculated by: $\hat{p}(x) = \sum_j \hat{b}_j\psi_j(x)$.

The concepts above described also apply to WDEs, but WDEs can use both scaling functions ϕ and wavelet functions ψ. Therefore, in WDEs, the scaling and wavelet coefficients can be approximated and expressed as: $\hat{a}_{j_0,k} = \frac{1}{N}\sum_{i=0}^{N}\phi_{j_0,k}(X_i)$ and $\hat{d}_{j,k} = \frac{1}{N}\sum_{i=0}^{N}\psi_{j,k}(X_i)$, respectively, where j_0 indicates the coarsest scale or the lowest resolution of analysis, and J refers to the finest scale or the highest resolution. Hence, the density can be approximated using one of the following three approaches: 1) using only scaling functions with $\hat{p}(x) = \sum_k \hat{a}_{j_0,k}\phi_{j_0,k}(x)$; 2) using only wavelet functions with $\hat{p}(x) = \sum_{j=0}^{J}\sum_k \hat{d}_{j,k}\psi_{j,k}(x)$; and 3) using both scaling and wavelet functions with $\hat{p}(x) = \sum_k \hat{a}_{j_0,k}\phi_{j_0,k}(x) + \sum_{j=j_0}^{J}\sum_k \hat{d}_{j,k}\psi_{j,k}(x)$. The proposed WPNN utilises RWFDE with only scaling functions that provides a novel PNN configuration with lower computational complexity.

D. Wavelet Neural Networks (WNNs)

WNNs are a special class of neural networks that replace the activation functions of feed-forward neural networks with more powerful computing units based on wavelet transform. These networks have been thoroughly investigated as an alternative approach to traditional neural networks that rely on sigmoidal activation functions. WNNs have attracted great interest, since they incorporate some key advantages of wavelet theory (i.e., multiresolution analysis, functions with good localisation in time and frequency, sparse representation of functions). The seminal work in the context of WNNs is the $(1 + \frac{1}{2})$ layer neural network based on wavelets [17] that not only preserves the universal approximation property of neural networks, but also establishes an explicit link between the network coefficients and some appropriate transform [17]. In WNNs, the learning is performed by the standard back-propagation type algorithm as in traditional feed-forward neural networks.

A comprehensive taxonomy for WNNs organized according to the type of basis function employed is proposed in [26]. As discussed in [27], studies on wavelets have been mostly concentrated in one or two-dimensional wavelets. The reason for this is the heavy computational cost associated with the implementation of multidimensional wavelet transforms. For multidimensional problems, WNNs usually constructs multidimensional wavelets based on computing the tensor product between one-dimensional wavelets [17]. An alternative approach for multidimensional wavelets was investigated by Kugarajah and Zhang [27]. This approach is based on the use of one-dimensional radial wavelets to reduce the computational complexity of the network.

E. Multidimensional frames for neural networks

As discussed in [21], a sequence $\{\Phi_n\}_{n\in\Gamma}$ is a wavelet frame if $A\|f\|^2 \leqslant \sum_{n\in\Gamma}|\langle f, \Phi\rangle|^2 \leqslant B\|f\|^2$ where Γ is an index set, $f \in \mathbf{H}$, A and B are two constants with $B \geqslant A > 0$. If the above expression holds, a signal f can be recovered in a Hilbert space $\mathbf{H}$ by using the frame $\{\Phi_n\}_{n\in\Gamma}$.

The construction of multidimensional wavelet frames for use in neural networks was studied by [27], where two approaches to generalise a one-dimensional wavelet frame to the multidimensional case are proposed. The first approach uses only one dilation parameter for all dimensions, whereas the second approach assigns different dilation parameter for each dimension. These two approaches are named *single-scaling* and *multi-scaling*, respectively.

The RWFDE estimator proposed by García-Treviño *et al.* uses a single-scaling dilation parameter to construct a multi-dimensional frame with a reduced computational complexity.

F. Radial Wavelet Frames Density Estimators

The key idea of RWFDE is the use of *Radial B-spline scaling functions*, which are a type of novel multidimensional scaling functions, and defined as [21]:

$$\Phi_{j_0,k}(\mathbf{x}) = 2^{\frac{nj_0}{2}} N_m\big(\|(2^{j_0}\mathbf{x} - \mathbf{k})\| + \frac{m}{2}\big) \tag{1}$$

where $n \in \mathbb{Z}$ is the data dimension of $\mathbf{x}$, $j_0 \in \mathbb{Z}$ is the dilation or scale parameter, and $\mathbf{k} \in \mathbb{Z}^n$ is a vector of translation parameters. The m-th order cardinal B-spline $N_m(x)$ can be obtained by using the convolution:

$$N_1(x) = \begin{cases} 1 & \text{for } 0 \le x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$N_m(x) = \int_{-\infty}^{\infty} N_{m-1}N_1(t)dt = \int_0^1 N_{m-1}(x-t)dt \tag{3}$$

One of the advantages of using the B-spline functions is that they have explicit closed-form solutions. The analytic closed-form solutions of the first three order B-spline function $N_m(x)$ can be found in Table I, where $\phi(x)$ refers to different orders of $N_m(x)$.

TABLE I: Explicit expressions for B-spline functions.

Order	B-spline function
Linear	$\phi(x) = \begin{cases} x & \text{for } 0 \le x < 1 \\ 2-x & \text{for } 1 \le x < 2 \\ 0 & \text{otherwise} \end{cases}$
Quadratic	$\phi(x) = \begin{cases} \frac{1}{2}x^2 & \text{for } 0 \le x < 1 \\ \frac{3}{4} - (x-\frac{3}{2})^2 & \text{for } 1 \le x < 2 \\ \frac{1}{2}(x-3)^2 & \text{for } 2 \le x < 3 \\ 0 & \text{otherwise} \end{cases}$
Cubic	$\phi(x) = \begin{cases} \frac{1}{6}x^3 & \text{for } 0 \le x < 1 \\ \frac{1}{6}(-3x^3+12x^2-12x+4) & \text{for } 1 \le x < 2 \\ \frac{1}{6}(3x^3-24x^2+60x-44) & \text{for } 2 \le x < 3 \\ \frac{1}{6}(4-x)^3 & \text{for } 3 \le x < 4 \\ 0 & \text{otherwise} \end{cases}$

1) Density Estimation: The density estimation process is based on the wavelet frame theory discussed in Section II-E, where a function $f(x)$ in $L^2(\mathbb{R})$ can be approximated using a basis of scaling functions. To begin with, the coefficients $w_{j_0,k}$ is calculated from the inner product between the function $f(x)$ and the frame function $\{\Phi_{j_0,k}\}_{j_0,k\in\mathbb{Z}}$:

$$w_{j_0,k} = \langle f, \Phi_{j_0,k}\rangle = \int f(\mathbf{x})\Phi_{j_0,k}(\mathbf{x})d\mathbf{x} \tag{4}$$

If the function $f(x)$ is replaced by the density function $p(x)$, (4) becomes:

$$w_{j_0,k} = \int p(\mathbf{x})\Phi_{j_0,k}(\mathbf{x})d\mathbf{x} \tag{5}$$

Given a group of random variables X_i, $i \in [1,N]$, whose empirical density function is expressed as $p_X(\mathbf{x}) \approx \frac{1}{N}\sum_{i=0}^{N}\delta(\mathbf{x}-X_i)$ and taking into the account the property of δ functions that states that $\int f(x)\delta(x-a)dx = f(a)$, then the coefficients in (5) can be approximated as:

$$\hat{w}_{j_0,k} = \int \Phi_{j_0,k}(\mathbf{x})\frac{1}{N}\sum_{i=0}^{N}\delta(\mathbf{x}-X_i)d\mathbf{x} = \frac{1}{N}\sum_{i=0}^{N}\Phi_{j_0,k}(X_i) \tag{6}$$

The estimated density $\hat{p}(x)$ at resolution 2^{j_0} can be obtained:

$$\hat{p}(\mathbf{x}) = \sum_k \hat{w}_{j_0,k}\Phi_{j_0,k}(\mathbf{x}) \tag{7}$$

2) Convergence: Convergence rates for WDEs have been well studied in the literature (see e.g. [28], [29] and [30]) and they are formulated in general terms and apply for any type of wavelet employed. Since RWFDE is a particular case of WDE with a novel type of wavelet, its convergence rates can be expressed similarly to the ones for WDEs.

Let the density p to belong to the Besov space $\mathbf{B}^s_{p,q}(\mathbb{R}^d)$, that is $p \in \mathbf{B}^s_{p,q}(\mathbb{R}^d)(p,q \in [1,\infty], s > d/p)$, then the linear estimator defined in (7) with $2^{j_0} \sim \big(\frac{N}{\ln N}\big)^{\frac{1}{2(s-d/p)+d}}$ satisfies

$$\sup_{\mathbf{x}\in[0,1]^d} |\hat{p}(\mathbf{x}) - p(\mathbf{x})| = \mathrm{O}_{a.s.}\left(\frac{N}{\ln N}\right)^{\frac{1}{(s-d/p)/(2(s-d/p)+d)}} \tag{8}$$

In the empirical assessment of RWFDE carried out in [21], the Mean Integrated Squared Error (MISE) for this type of estimators outperforms traditional WDEs as well as other recently published estimator variants.

3) Hyper Parameters: The estimator RWFDE has three parameters to be selected by the user based on the different target scenarios. The first one is the order of the B-spline scaling functions. Similar to the bandwidth parameter in the Gaussian function in PNN, the order of the B-spline scaling functions controls the degree of smoothness. The second one is the resolution parameter j_0, which also affects the smoothness, and it needs to be chosen in an appropriate way according to the complexity of the estimated density function and the amount of data available. The larger the j_0, the more details will be extracted but also tends to overfit. Fig. 1 depicts the variation of smoothness when applying different j_0 to the different number of available data points by keeping the order of the B-spline scaling functions unchanged. The third parameter, α, is used only for non-stationary data and it corresponds to a sliding window with size P, $\alpha = 1/P$.

G. The RWFDE Formulation

The formulation of RWFDE has three steps: *Initialisation*, *Updating Process*, and *Evaluation Process* [21]. Note, data normalisation is needed, since the input data $\mathbf{x}$ is assumed to lie within the interval $[0,1]^n$, where n denotes its corresponding dimension.

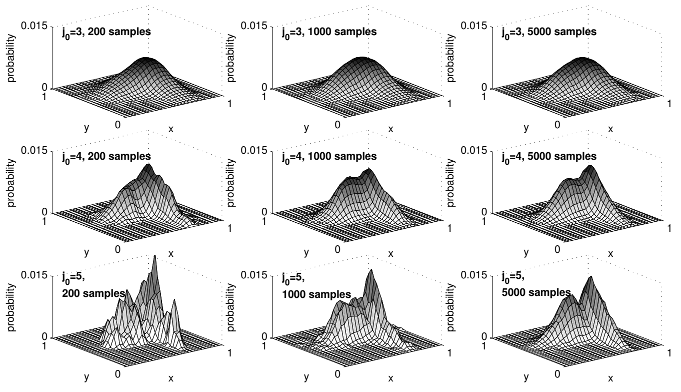

Fig. 1: The smoothing effect of different values of j_0 on a density function estimated from different number of data points. (The basis function used is the radial *Quadratic* B-spline scaling function).

1) *RWFDE Initialisation:* This step initialises the parameters for WPNN: i) The support of a radial B-spline scaling function $\Phi(\mathbf{x})$ with any dilation and translation parameters j_0 and k is $[2^{-j_0}(k-\frac{m}{2}), 2^{-j_0}(k+\frac{m}{2})]$. ii) The number of frame functions required to form a frame within the support is $2^{j_0}+2u+1$, where $u=1$ for *Linear* and *Quadratic* order B-spline and $u=2$ for *Cubic* order. Therefore, the translation parameters k are given by $k \in \{-u, ..., 0, 1, 2, ..., 2^{j_0}+u\}$. The multivariate translation vector $\mathbf{k} \in \mathbb{Z}^n$ can then be constructed by creating all combinations of the translation parameter k across the data dimension n, see [21] for details.

2) *Updating Process:* This step aims to update the coefficients $\hat{w}_{j_0,k}$ for the batch, and online environments. Note that as discussed in RWFDE [21], Equation (6) can be optimised and hence, for the coefficient $\hat{w}$, the l-th relevant frame function using different orders of $\Phi(\mathbf{x})$ at the timestamp t for the data $\mathbf{x}$ can be obtained for specified j_0, k:

$$\begin{aligned}\hat{w}_l^t &= \hat{w}_l^{t-1} + \frac{1}{t}\Big(\Phi_{j_0,k}(\mathbf{x}_t) - \hat{w}_l^{t-1}\Big) \\ &= \hat{w}_l^{t-1} + \frac{1}{t}\Big(2^{\frac{nj_0}{2}}\phi\big(\|(2^{j_0}\mathbf{x}_t - \mathbf{k}_l)\| + \frac{m}{2}\big) - \hat{w}_l^{t-1}\Big)\end{aligned} \tag{9}$$

$$\begin{aligned}\hat{w}_l^t &= (1-\alpha)\hat{w}_l^{t-1} + \alpha\big(\Phi_{j_0,k}(\mathbf{x}_t)\big) \\ &= (1-\alpha)\hat{w}_l^{t-1} + \alpha\Big(2^{\frac{nj_0}{2}}\phi\big(\|(2^{j_0}\mathbf{x}_t - \mathbf{k}_l)\| + \frac{m}{2}\big)\Big)\end{aligned} \tag{10}$$

where $\phi(x)$ represents different orders of the B-spline function $N_m(x)$, (9) is the updating process in the offline and online stationary environment, and (10) is the updating process of the online non-stationary environment. The index $t \in \{1, 2, ..., N\}$ is the number of data points that have been processed so far. The forgetting factor α relates to a window of size P to analyse the latest P data points ($\alpha = 1/P$).

Note that the number of frame functions depends on the data dimension of $\mathbf{x}$, the order of Φ and the scale j_0; therefore, an optimisation process, so-called *find relevant frame functions*, was introduced by the authors in [21] and can be found in Algorithm 1 in Section III-D. Hence, a set of relevant frame functions will be used to update the coefficients, such that their resulting evaluations of $2^{\frac{nj_0}{2}}\phi\big(\|(2^{j_0}\mathbf{x}_t - \mathbf{k}_l)\| + \frac{m}{2}\big)$ are different from zero.

3) *Evaluating Process:* Once the coefficients $\hat{w}$ are found, the estimated density $\hat{p}(x)$ can be derived using (7). Details can be found in [21].

III. The proposed wavelet probabilistic neural network

There is a great body of research dedicated to the study of PNNs. However, to the best of our knowledge, none of the existing approaches has considered a PNN with a wavelet-based density estimator. In this section, the novel formulation of RWFDE-based PNN will be discussed.

A. *Wavelet Probabilistic Neural Networks (WPNN)*

Similar to the PNN, the proposed WPNN is also a four-layered network, with input, pattern, summation and output layers; however, they differ on the units within the pattern layer. While the former relies on Gaussian functions, the latter considers multidimensional radial wavelet frames based on radial B-spline scaling functions from [21]. Note here that the replacement of these activation functions implies a substantial change in the density estimation paradigm employed by the network. The structure of the proposed WPNN is depicted in Fig. 2, where the translation vector $\mathbf{k}$ is a multivariate vector which contains all the combinations of the translation parameter k across the data dimension n. The sub-indices of k in the figure refer to the translation parameter k at the M-th row, n-th column in $\mathbf{k}$, where M depends on the length of k, and n depends on the data dimension of $\mathbf{x}$.

The formulation for the proposed WPNN can be divided into three main steps: network construction, network training and network testing.

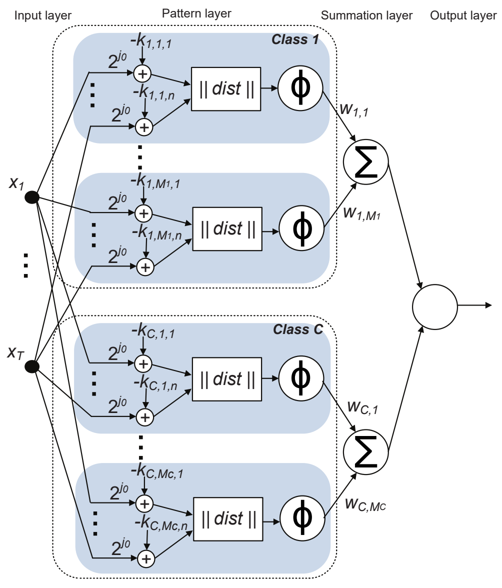

Fig. 2: Proposed WPNN.

B. *Online and adaptive learning in WPNN*

The key advantage of the proposed WPNN is that, since it is based on WDEs, it allows the recursive computation of network weights. The coefficients can therefore be dynamically updated when new data points arrive (Section III-D). Two different online learning algorithms are proposed: i) For the stationary environment, whenever new data arrives, the coefficients will be discounted according to the current timestamp; ii) For the non-stationary environment, as the underlying characteristics of the data vary over time, a forgetting factor α is defined to discount the historical network coefficients.

C. *WPNN construction*

The initial step is the construction of the network (as in the *RWFDE Initialisation*), where the translation parameter k for the m-th order of B-spline and the dilation parameter j_0 are defined (see also Section II-G).

D. *WPNN training*

In the training step, the empirical coefficients w (network weights) are estimated for each basis function of all classes, using only available training data. Using (6), (9), and (10), the coefficients w for the l-th basis function associated with the class c_q at timestamp t (for the offline setting and online environments, respectively) for specified j_0, k are given by:

$$\hat{w}^t_{q,l} = \hat{w}^{t-1}_{q,l} + \frac{1}{t}\Big(2^{\frac{nj_0}{2}}\phi\big(\|(2^{j_0}\mathbf{x}_t - \mathbf{k}_l)\| + \frac{m}{2}\big) - \hat{w}^{t-1}_{q,l}\Big) \quad (11)$$

$$\hat{w}^t_{q,l} = (1-\alpha)\hat{w}^{t-1}_{q,l} + \alpha\Big(2^{\frac{nj_0}{2}}\phi\big(\|(2^{j_0}\mathbf{x}_t - \mathbf{k}_l)\| + \frac{m}{2}\big)\Big) \quad (12)$$

Note that relevant frame functions are selected for the training process in this step by using Algorithm 1, hence $l = 1, 2, ..., M_q$. Algorithm 2, Algorithm 3 and Algorithm 4 present the pseudocode for the above training procedure in offline, online stationary and online non-stationary environments, respectively. Note that unlike the standard neural networks or WNNs, the proposed WPNN does not require back-propagation, which is also one of the advantages of using PNN.

1) WPNN offline training: This algorithm (Algorithm 2) runs only once, and the outputs are the coefficients for every class. In the pseudocode, the set of indices of the relevant frame functions is denoted as $\mathbf{b}$. The expression $|\mathbf{b}|$ is the cardinality of $\mathbf{b}$, the symbol $\setminus$ as in $\mathbf{f}\setminus\mathbf{b}$ removes the elements in $\mathbf{b}$ from $\mathbf{f}$.

2) WPNN online training: The two online training processes update the coefficients in the stationary and non-stationary environments, and are presented in Algorithm 3 and Algorithm 4, respectively.

It is worth mentioning that Algorithm 2 and Algorithm 3 use the same updating policy (11). However, in the case of Algorithm 2 all the training data $\mathbf{x}$ will be provided, and hence this algorithm will only run once. In contrast, Algorithm 3 will be called whenever a new data point is available: the algorithm is only aware of the last data point and the last calculated coefficient w for a specific class. Regarding Algorithm 4, it utilises the updating policy (12). Similar to Algorithm 3, it aims to update the coefficients instantly when a new data point is available.

Algorithm 1: RWFDE_find_relevant_frame_for_given_datapoint ($\mathbf{x}_t$,j_0,m,$\mathbf{k}$,M)

```
Input: $\mathbf{x_t}$: The latest training data $\mathbf{x}$ at the timestamp t; $j_0$: Index associated
       to the base resolution; $m$: The order of the $B$-spline function
       employed as scaling function; $\mathbf{k}$: Translation vectors for the frame
       functions in [0,1]; $M$: Number of frame functions in [0,1].
Output: $\mathbf{b} = \{b_1, b_2, ..., b_B\}$: Set of indices of the translation vectors for
        the frame functions relevant for $\mathbf{x}_t$.
$n = dim(\mathbf{x_t})$;
$\mathbf{h} = [h_1, h_2, ..., h_n] = [0, 0, ..., 0]$;
$\mathbf{x} = \mathbf{x_t}$;
for $j \leftarrow 1$ to $M$ do
    for $d \leftarrow 1$ to $n$ do
        if $\mathbf{x_d} \geq 2^{-j_0}(\mathbf{k}_j - \frac{m}{2})$ and $\mathbf{x_d} \leq 2^{-j_0}(\mathbf{k}_j + \frac{m}{2})$ then
            $h_d \leftarrow h_d + 1$;
            if $d$ == 1 then
                $A_{d,h_d} \leftarrow j$;
            else
                $A_{d,h_d} \leftarrow (j-1)M^{d-1}$;
$\mathbf{b} = A_{1,*}$;
for $d \leftarrow 1$ to $n-1$ do
    $\mathbf{g} \leftarrow \{\}$;
    for $i \leftarrow 1$ to length($\mathbf{b}$) do
        for $p \leftarrow 1$ to columns($A$) do
            $\mathbf{g} \leftarrow concatenate(\mathbf{g}, (b_i + A_{d+1,p}))$;
    $\mathbf{b} \leftarrow \mathbf{g}$;
```

Algorithm 2: WPNN_offline_training ($\mathbf{X}, C, j_0, m, M, \mathbf{k}$)

```
Input: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{c_q}\}$: Collection of data vectors for all the
       classes; $C$: Number of classes; $j_0$: Index associated to the base
       resolution; $m$: Order of the $B$-spline function employed as scaling
       function; $M$: Number of frame functions in [0,1]; $\mathbf{k}$: Translation
       vectors for the frame functions in [0,1].
Output: $\hat{\mathbf{W}} = \{\hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_C\}$: Set of empirical coefficients for the basis
        functions of all the classes.
for $q \leftarrow 1$ to $C$ do
    $N_q$ = length($\mathbf{x}_q$);
    for $t \leftarrow 1$ to $N_q$ do
        $\mathbf{b}$ = RWFDE_find_relevant_frame_for_given_datapoint($\mathbf{x}_{q,t}$,$j_0$, $\mathbf{k}$,
         $M$);
        $M_q = |\mathbf{b}|$;
        for $l \leftarrow 1$ to $M_q$ do
            $\hat{w}_{q,b_l} =$
              $\hat{w}_{q,b_l} + \frac{1}{t}\Big(2^{\frac{nj_0}{2}}\phi\big(\|(2^{j_0}\mathbf{x}_{q,t} - \mathbf{k}_{b_l})\| + \frac{m}{2}\big) - \hat{w}_{q,b_l}\Big)$;
        $\mathbf{f} = [1, 2, ..., M]$;
        $\mathbf{f}\setminus\mathbf{b} = [1, 2, ..., M]\setminus[b_1, b_2, ..., b_B]$;
        for $l \leftarrow 1$ to $M - |b|$ do
            $\hat{w}_{q,f_l} = \hat{w}_{q,f_l} - \frac{\hat{w}_{q,f_l}}{t}$
    $\hat{\mathbf{w}}_q = [\hat{w}_{q,1}, \hat{w}_{q,2}, \ldots, \hat{w}_{q,M}]$;
```

E. *WPNN testing*

The evaluation of a trained WPNN consists of using the test data to estimate the corresponding probability for each class. The most probable generative model for each testing data point is chosen as its corresponding class. According to this, we first need to obtain the class-conditional probability density function for the class c_q, which is defined by

$$\hat{p}_q(\mathbf{x}_i|c_q) = \sum_{l=1}^{M_q} 2^{\frac{nj_0}{2}}\hat{w}_{q,l}\phi\big(\big\|2^{j_0}\mathbf{x}_i - \mathbf{k}_{q,l}\big\| + \frac{m}{2}\big) \qquad (13)$$

```
Algorithm 3: WPNN_online_stationary_training ($\mathbf{x_t}, \hat{w}_q, j_0, m, M, \mathbf{k}, t$)
Input: $\mathbf{x_t}$: The latest training data $\mathbf{x}$ at the timestamp t; $w_q$ The set of latest coefficients for class $c_q$; $j_0$: Index associated to the base resolution; $m$: The order of the $B$-spline function employed as scaling function; $M$: Number of frame functions in [0,1]; $\mathbf{k}$: Translation vectors for the frame functions in [0,1]; $t$: Current timestamp/the $t$-th data in the dataset.
Output: $\hat{w}_q$: Set of network coefficients for the class $c_q$.
b = RWFDE_find_relevant_frame_for_given_datapoint($\mathbf{x}_t$,$j_0$, $\mathbf{k}$,$M$);
$M_q = |\mathbf{b}|$;
for $l \leftarrow 1$ to $M_q$ do
    $\hat{w}_{q,b_l} = \hat{w}_{q,b_l} + \frac{1}{t}\left(2^{\frac{nj_0}{2}}\phi(\|(2^{j_0}\mathbf{x}_t - \mathbf{k}_{b_l})\| + \frac{m}{2}) - \hat{w}_{q,b_l}\right)$;
$\mathbf{f} = [1, 2, ..., M]$;
$\mathbf{f}\backslash\mathbf{b} = [1, 2, ..., M]\backslash[b_1, b_2, ..., b_B]$;
for $l \leftarrow 1$ to $M - |b|$ do
    $\hat{w}_{q,f_l} = \hat{w}_{q,f_l} - \hat{w}_{q,f_l}/t$
```

```
Algorithm 4: WPNN_online_non_stationary_training ($\mathbf{X_t}, \hat{w}_q, j_0, m, M, \mathbf{k}, \alpha$)
Input: $\mathbf{x_t}$: The latest training data $\mathbf{x}$ at the timestamp t; $\hat{w_q}$: The set of latest coefficients for class $c_q$; $j_0$: Index associated to the base resolution; $m$: The order of the $B$-spline function employed as scaling function; $M$: The number of frame functions in [0,1]; $\mathbf{k}$: Translation vectors for the frame functions in [0,1]; $\alpha$: The forgetting factor, it is the inverse of the size of a sliding window.
Output: $\hat{w}_q$: Set of network coefficients for the class $c_q$.
b = RWFDE_find_relevant_frame_for_given_datapoint($\mathbf{x}_t$,$j_0$, $\mathbf{k}$,$M$);
$M_q = |\mathbf{b}|$;
for $l \leftarrow 1$ to $M_q$ do
    $\hat{w}_{q,b_l} = (1-\alpha)\hat{w}_{q,b_l} + \alpha\left(2^{\frac{nj_0}{2}}\phi(\|(2^{j_0}\mathbf{x}_t - \mathbf{k}_{b_l})\| + \frac{m}{2})\right)$;
$\mathbf{f} = [1, 2, ..., M]$;
$\mathbf{f}\backslash\mathbf{b} = [1, 2, ..., M]\backslash[b_1, b_2, ..., b_B]$;
for $l \leftarrow 1$ to $M - |b|$ do
    $\hat{w}_{q,f_l} = \hat{w}_{q,f_l} - \alpha\hat{w}_{q,f_l}$
```

Then the output unit computes the decision rule defined in (14) to obtain the class c_q associated with the training vector $\mathbf{x}_i$.

$$d(\mathbf{x}_i) = \arg\max_q\{P(c_q)\hat{p}_q(\mathbf{x}_i|c_q)\}, \quad q = 1, \dots, C \tag{14}$$

where $P(c_q)$ is the a prior class probability, which is assumed to be uniform for all classes. Algorithm 5 presents the WPNN testing step for a specific class c_q. For a specific data point $\mathbf{x_i}$, the predicted class can be obtained using (14) and selecting the maximum probabilities over different classes.

```
Algorithm 5: WPNN_testing ($\mathbf{X}, \mathbf{K}, \mathbf{w}, j_0, m, \mathbf{k}, M$)
Input: $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{N_{\text{testing}}}\}$: Set of $N_{\text{testing}}$ testing data; $\hat{w}$: Set of network coefficients for the basis functions; $j_0$: Index associated to the base resolution; $m$: The order of the $B$-spline function employed as scaling function; $\mathbf{k}$: Translation vectors for the frame functions in [0,1]; $M$: number of frame functions in [0,1].
Output: $\hat{\mathbf{p}} = \hat{p}(\mathbf{x}_1), ..., \hat{p}(\mathbf{x}_{N_{\text{testing}}})$: Set of probabilities for the testing data $\mathbf{x}$.
for $i \leftarrow 1$ to $N_{\text{testing}}$ do
    b = RWFDE_find_relevant_frame_for_given_datapoint($\mathbf{x}_i$,$j_0$, $\mathbf{k}$,$M$);
    $M_q = |\mathbf{b}|$;
    for $l \leftarrow 1$ to $M_q$ do
        $z_l = 2^{\frac{nj_0}{2}} w_{b_l}\phi(\|2^{j_0}\mathbf{x}_i - \mathbf{k}_{b_l}\| + \frac{m}{2})$;
    $\hat{p}_q(\mathbf{x}_i|c_q) = \sum_{l=1}^{M_q} z_{b_l}$;
```

F. WPNN parameters selection

Three are the tuning parameters for the proposed WPNN. The first one is the order of the B-spline scaling functions selected for the pattern units. The second parameter is j_0, the dilation or scale parameter of the B-spline scaling functions. By changing these two parameters, we can control the degree of smoothness of the densities for each class and then modify the discrimination capabilities of the network. Note here that the selection of j_0 will depend on the size of the available training data as well as on the complexity of the classification problem. And the third parameter, for the online non-stationary environment, is the forgetting factor α that is used to discount the historical network coefficients. The larger the α, the more emphasis will be put on recent data. The size of α will need to be modified based on the characteristic of the datasets, such as the frequency of concept drift.

The effect of increasing the number of training data and the effect of modifying the base resolution j_0 on the classification results are both shown in Fig. 3.

G. Coefficient adjustment: WPNN performance improvement

In this section, we propose a Coefficient Adjustment Operator (CAO), which is applied to a selected subset of the empirical coefficients of a trained WPNN, to improve generalisation and minimise overfitting in the offline setting. In order to apply this operator, the experimental data has to include training, validation and test sets. The improvement in performance is evaluated using the validation set. In this way, we verify that any increase in accuracy over the training set actually yields an increase in accuracy over the validation set.

Note that this operator is not directly aimed at improving the estimation of class-conditional densities. Instead, its objective is to modify the decision boundary. According to Donoho *et al.* [31] the procedure, so-called wavelet thresholding, is similar to the adjustment operator proposed, which can improve convergence because they suppress noise artefacts and peaks. See [31] for more comprehensive and theoretical analysis.

The CAO is defined by

$$\delta_\lambda(w_{q,l}) = \begin{cases} w_{q,l} + \lambda, & \text{if } w_{q,l} \in \mathcal{F}; \\ 0, & \text{otherwise}; \end{cases} \tag{15}$$

where $w_{q,l}$ is a given empirical coefficient and the set $\mathcal{F}$ includes the coefficients of all those basis functions whose support includes at least one misclassified data point of the validation set. Note that when (15) is applied to any of the coefficients of the subset $\mathcal{F}$ its corresponding value is increased by an amount equal to λ.

To evaluate the effect of using CAO with the proposed WPNN, we perform an experiment with a synthetic spiral dataset similar to Fig. 3. In this assessment, we evaluate different configurations of the proposed neural network varying the B-spline function employed, its resolution and the value of the CAO applied. We also vary the number of data points considered for the dataset. The 25 percent of the dataset is used for each of training and validation, while the remaining 50 per cent is used for testing. Each configuration is evaluated by a hundred times randomly selecting training, validation and testing sets and the average accuracy for these 100 experiments is reported. The results obtained for all experiments are presented in Table II.

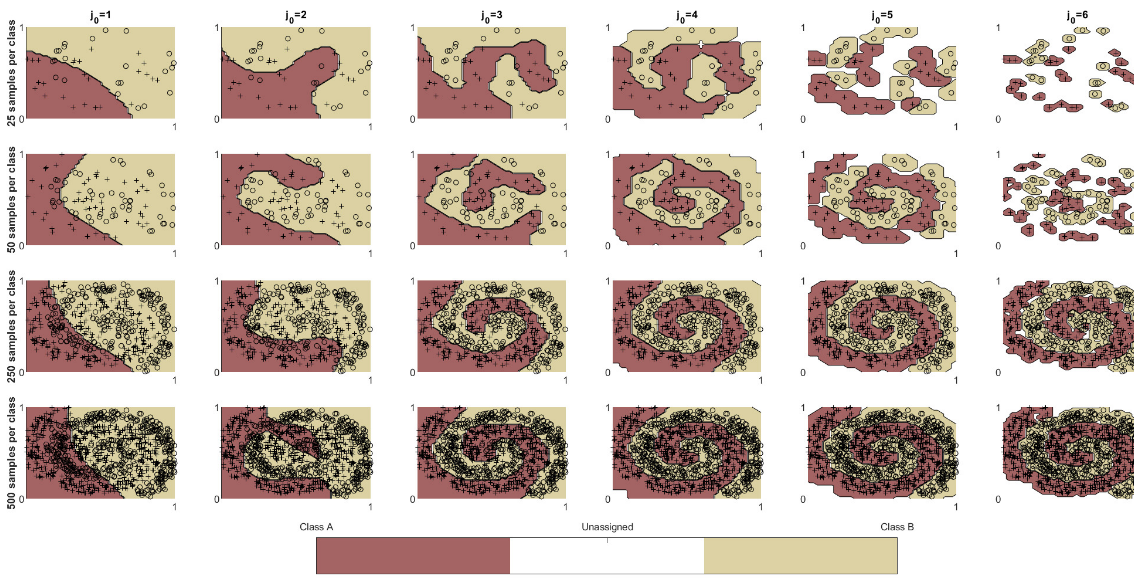

Fig. 3: The effect of increasing the number of data and modifying the base resolution j_0 on the classification results of the proposed WPNN. This example experiment uses radial *Quadratic* B-spline scaling functions.

There are three key observations from Table II. The first one is that for resolutions higher than 3, algorithms with CAO report the highest accuracy in around 80 per cent of the cases, that is in 29 out of the 36 scenarios, where considering the three B-spline functions evaluated and the four dataset settings. This means that the CAO improves the WPNN performance when the algorithm is working with the finest resolutions. Note that higher B-spline resolutions are better for applications involving more complex decision boundaries. The second observation is that the use of CAO becomes more relevant in classification scenarios when there is not enough data to characterise the different classes. Note that for WPNN configurations with resolutions equal to 3 and 4 CAO only improves performance in 1 out of 6 experiments with the largest dataset, the one with 5000 data points. The third observation (related to the first one) is that the use of CAO is not recommended for WPNNs with B-spline functions with coarser resolutions as even a small modification of its coefficients bring large changes in the decision boundary. This can be observed in the 12 scenarios for $j_0 = 3$, where only in 17 per cent of them, CAO improved accuracy.

The effect of applying the coefficient adjustment operator on the spiral dataset is shown in Fig. 4. The testing accuracy in this figure is increased from 97.0 to 97.2 after application of the coefficient adjustment.

H. Time Complexity of the online version of WPNN

Table III shows the time complexity of WPNN when processing one data point in an online environment. The analysis has been separated into four steps: network parameter initialisation, find relevant frame functions for the data, evaluation, and network parameter update. Note that R is the length of translation parameters, n is the data dimension, M_q is the cardinality of the relevant frame function, n_c is the number of classes in the classification task, m is the order of B-spline.

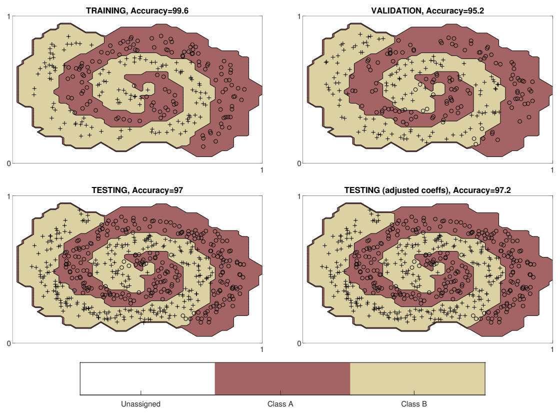

Fig. 4: The effect of applying a coefficient adjustment operator λ on the coefficients of trained WPNN estimated from 1000 data points for each class using *Quadratic* B-spline scaling functions with $j_0 = 5$.

The complexity is independent of the cardinality of the dataset, but depends on the parameters settings and the latest available data point. Therefore, the proposed WPNN has the ability to provide low and constant computation time for data stream classification.

IV. Performance assessment

In this section, the performance of the proposed WPNN is assessed using synthetic and real-world datasets in both offline and online settings [1]. The empirical evaluation includes three types of experiments: i) Offline classification of publicly available datasets; ii) Online classification of stationary datasets; and iii) Online classification of non-stationary data

[1]The source code is released at https://github.com/puyangdl/WPNN.

TABLE II: Experimental Evaluation for the Coefficient Adjustment Operator on Spiral Dataset

		Linear B-spline				*Quadratic* B-spline				*Cubic* B-spline			
Resolution	CAO	n=100	n=500	n=1000	n=5000	n=100	n=500	n=1000	n=5000	n=100	n=500	n=1000	n=5000
$j_0 = 3$	No CAO	**68.12**	68.14	67.08	**64.44**	**65.64**	**65.03**	**64.24**	**62.36**	**65.14**	**65.35**	**64.25**	**62.95**
	λ=0.0005	68.04	**68.62**	**67.21**	63.80	64.62	58.13	54.17	53.07	61.38	53.33	51.15	51.24
	λ=0.0010	68.06	68.35	66.87	63.43	64.22	56.97	53.14	52.28	60.66	52.31	50.31	50.28
	λ=0.0050	68.02	66.93	65.47	61.80	63.60	54.78	51.21	50.59	59.44	51.71	50.00	50.00
	λ=0.0100	67.78	65.87	64.34	60.57	63.18	53.95	50.72	50.09	59.08	51.68	50.00	50.00
	λ=0.0500	67.46	62.15	60.14	56.08	62.04	52.88	50.53	50.00	58.82	51.66	50.00	50.00
$j_0 = 4$	No CAO	76.48	93.22	92.78	83.99	78.72	78.65	**74.60**	**70.76**	72.18	**74.77**	**74.41**	**72.12**
	λ=0.0005	**77.26**	93.35	**92.92**	**84.52**	**78.82**	**78.85**	74.39	68.41	**72.30**	73.38	69.65	63.92
	λ=0.0010	77.26	**93.36**	92.92	84.48	78.78	78.79	74.21	67.45	72.24	72.92	68.43	62.07
	λ=0.0050	77.26	93.33	92.85	84.17	78.40	78.48	73.42	63.67	72.08	71.20	64.49	57.35
	λ=0.0100	77.26	93.34	92.76	83.77	78.30	78.29	72.85	61.28	72.02	70.23	62.68	55.44
	λ=0.0500	77.18	93.01	92.07	80.81	78.40	77.44	70.52	55.20	71.54	67.62	58.56	51.79
$j_0 = 5$	No CAO	65.40	88.51	94.91	97.35	76.12	94.01	96.42	**96.04**	79.32	94.63	**95.66**	88.14
	λ=0.0005	**66.46**	89.79	95.33	97.37	76.66	94.17	96.45	96.04	79.56	94.70	95.65	**88.20**
	λ=0.0010	66.46	89.79	95.33	**97.37**	76.64	94.19	96.44	96.03	79.64	94.72	95.63	88.06
	λ=0.0050	66.46	89.84	**95.37**	97.36	76.64	**94.23**	96.44	96.02	**79.68**	**94.74**	95.55	87.49
	λ=0.0100	66.46	89.82	95.36	97.35	76.62	94.20	**96.46**	96.00	79.60	94.67	95.50	87.10
	λ=0.0500	66.44	**89.88**	95.36	97.28	**76.80**	94.12	96.33	95.94	79.46	94.32	95.26	85.90
$j_0 = 6$	No CAO	55.66	70.18	80.42	97.38	61.68	85.42	93.81	98.27	68.40	92.27	96.28	98.26
	λ=0.0005	**56.14**	**72.28**	82.75	97.65	62.74	86.87	94.48	98.28	**69.50**	92.82	**96.40**	**98.27**
	λ=0.0010	56.14	72.28	82.75	97.65	62.74	86.87	**94.50**	**98.28**	69.50	**92.84**	96.39	98.26
	λ=0.0050	56.14	72.28	**82.75**	97.66	62.74	**86.88**	94.48	98.27	69.50	92.80	96.39	98.24
	λ=0.0100	56.14	72.28	82.75	**97.66**	**62.76**	86.87	94.48	98.28	69.46	92.83	96.38	98.25
	λ=0.0500	56.14	72.27	82.73	97.63	62.76	86.86	94.43	98.25	69.44	92.80	96.32	98.26

Frequency	Step	Time Complexity
Run only once	Parameter Initialisation	O(R^n)
Run whenever new data arrives	Find relevant frame function	O($n(R + M_q)$)
	Evaluation	O($n_c + M_q nm$)
	Parameter update	O($M_q nm$)

TABLE III: Time complexity of WPNN

datasets. All the experiments were run using MATLAB 2014a[2] on Windows 10 OS with Intel i9-9900K CPU and 32G RAM.

As the WPNN is a reformulation of a PNN, we first compare their performance. In accordance to [32]: the bandwidth/smoothing parameter σ is proportional to the standard deviation of the dataset and the average distance between data points. The resulting smoothing parameter $\hat{\sigma}$ is estimated using algorithm *Gap-Based Estimation* in [32]. The chosen datasets have standard deviations between 0.01 to 0.2, and the resulting $\hat{\sigma}$ between 0.002 to 0.07. Therefore, for PNN, we consider five different values for the smoothing parameter of the Gaussian kernel that can cover this range: 0.001, 0.01, 0.05, 0.1 and 0.2. Regarding WPNN, we test them with B-splines of the three orders depicted in Table I and with five different values of j_0: $1, 2, 3, 4$ and 5. The highest j_0 is chosen to be 5 as in [21] showed that overfitting happened when $j_0 \geq 6$.

A. Offline classification

This experiment evaluates the performance of the proposed WPNN (Algorithm 2) using five benchmark classification datasets from different sources [33], [34]. The assessment is based on a stratified shuffle and split cross-validation, which divides the datasets into training, validation, and test sets. For the construction of each set, we randomly select 25%, 25%, and 50% of the examples for training, validation, and test sets, respectively. The network configurations that provide the best performance in the validation set are subsequently used by the test set. We repeat the experiment 100 times and the average accuracies are used. Note that only numerical features will be used as they can be normalised within the interval $[0, 1]^n$ (see Section II-G). Note also that Test2 in Table IV refers to the parameter adjustment operation mentioned in Section III-G by assigning $\lambda = [0.01, 0.001, 0.0001]$ to the coefficients within the support. The average results for the best network configurations are shown in Table IV.

Other ML algorithms such as SVM with linear, radial basis function and polynomial kernels, Random Forest (RF), Feed-Forward Neural Networks (FFNN) and Gradient Boosting (GB) are also used to evaluate the performance of WPNN in the offline setting[3]. The average accuracies of 100 trials from the best algorithm are given in Table V.

From these results, we can see that the proposed WPNN outperforms or matches the performance of PNN and other ML algorithms within 1% difference. Furthermore, in four out of the five datasets evaluated, the proposed WPNN shows better results than other ML algorithms.

B. Online classification in a stationary environment

This experiment evaluates the online learning and classification capabilities of the proposed WPNN (Algorithm 3). The assessment is based on a synthetic spiral dataset similar to Fig. 4, consisting of 20 thousand data points. The results reported for the online environments are the average results of 100 repeated trials.

The classification performance is compared using the sliding window-based *Prequential error* approach [37] which monitors the evolution of learning as it progresses. For this type of error, each individual data point is used to test the model before it is used for training, and hence the accuracy is constantly updated at each timestamp. We use a sliding window that selects the latest P errors obtained from the learning process and calculates the prequential error [37]. For the PNN, four different settings are used for online learning. A sliding window with size $N = [100, 500, 1000, N_{available}]$

[2]Note that for benchmarking with alternative algorithms, Python 3.6 was used due to the alternative algorithms being implemented in Python.

[3]Note that scikit-learn [35] and Pytorch [36] are used to implement such algorithms.

TABLE IV: Accuracy for the Offline Classification Experiment

				PNN			WPNN											
							Linear B-spline				*Quadratic* B-spline				*Cubic* B-spline			
n	**Dataset**	l	n	Train	Valid	Test	Train	Valid	Test	Test2	Train	Valid	Test	Test2	Train	Valid	Test	Test2
1	Iris	150	4	99.74	95.28	95.27	96.33	95.81	**95.51**	**95.53**	97.26	95.83	95.36	95.20	98.23	95.69	95.27	95.27
2	Fourclass	862	2	100.0	99.68	**99.64**	99.84	99.13	99.18	99.23	99.94	99.55	99.48	99.48	99.96	99.53	99.51	99.51
3	Banana	5300	2	91.12	89.98	**90.13**	91.09	90.04	89.45	89.68	92.19	90.53	89.55	89.66	91.43	89.92	89.89	89.98
4	Newthyroid	215	5	99.87	93.70	93.61	96.42	92.83	93.07	93.50	96.36	93.30	93.50	93.64	97.49	94.26	**93.98**	**93.95**
5	Titanic	2201	3	79.41	78.61	**78.44**	77.78	77.14	77.18	77.18	78.14	77.73	77.56	77.56	78.15	77.94	77.78	77.78

TABLE V: Offline Classification Accuracy - WPNN vs Other Machine Learning algorithms

		WPNN							
n	**Dataset**	**Order**	**Train**	**Valid**	**Test**	**Method**	**Train**	**Valid**	**Test**
1	IRIS	*Linear*	96.33	95.50	**95.53**	SVM-linear	94.59	89.47	94.67
2	Fourclass	*Cubic*	99.96	99.53	**99.51**	RF	100.00	97.22	96.75
3	Banana	*Cubic*	91.43	89.92	**89.98**	RF	100.00	87.77	88.91
4	Newthyroid	*Cubic*	97.49	94.26	**93.98**	FFNN	96.23	98.15	92.59
5	Titanic	*Cubic*	78.15	77.94	77.78	GB	78.91	80.73	**78.29**

is used to select the most recent N data points for training, where $N_{available}$ is the number of available training data so far. In the following sections default PNN stands for PNN ($N = N_{available}$). For the online learning process, the PNN requires N data points inside the sliding window, while the WPNN only requires the last available data point.

The evaluation metrics are accuracy and computation time. Two accuracies are reported: first, the *instantaneous accuracy*, which is defined as the classification accuracy in the current sliding window, and second, the *accumulated accuracy*, which is the average accuracy of all the evaluated sliding windows so far. The computation time is defined as the time taken to train the network with the readily available data points plus the time taken to predict the class of one future data point. Note that the above metrics consider a sliding window moving forward one data point at each timestamp.

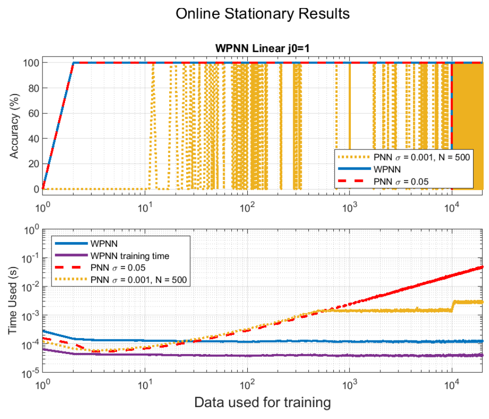

Fig. 5: Instantaneous accuracy for the online experiment in a stationary context.

Fig. 5 shows how, for the latest available data, the instantaneous accuracy increases steadily at the beginning of the training process for the WPNN and default PNN ($N = N_{available}$). The accuracy only drops when the number of training data $N = 1 \times 10^4$, which is the first time that a second class appears. However, the computation time for the WPNN algorithm remains constant. In contrast, the computation time of the default PNN grows exponentially as more training data becomes available. In respect to the confusion matrices (Fig. 6), both the WPNN and the default PNN present similar performance, with only one misclassified data point. Regarding the best window setting of PNN ($N = 500$) the results show, as expected, a low computation time (due to the constant number of training data used) but at the expense of a much lower instantaneous accuracy.

Regarding the configuration of the WPNN and the PNN, the best parameter for the WPNN, in this case, is *Linear* order B-splines with $j_0 = 1$. For the default PNN, a larger bandwidth parameter is used, $\sigma = 0.05$, providing higher a degree of smoothness than smaller σ values, whereas for the PNN ($N = 500$), $\sigma = 0.001$ is used.

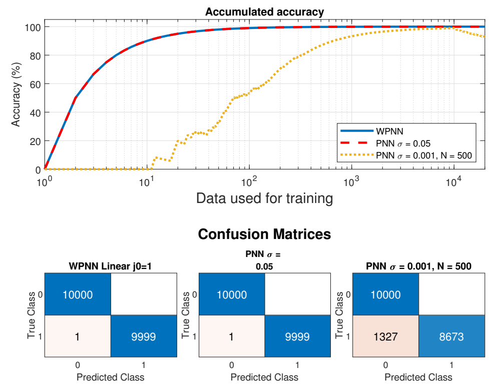

Fig. 6: Online stationary classification: Accumulated accuracy and the confusion matrix for the best testing configurations of WPNN and PNN.

C. Online classification in a non-stationary environment

In this section, a synthetic dataset consisting of 20 thousand data points with two different Gaussian distribution parameters is used to test Algorithm 4. Fig. 7 depicts the distribution of this dataset. A set of forgetting factors $\alpha = [\frac{1}{100} \frac{1}{300} \frac{1}{500} \frac{1}{700} \frac{1}{1000}]$ are used to discount the network parameters as in [21]. Here we also use the *Pre-quential error* to test the performance.

Results from this experiment are shown in Fig. 8 and 9. Similar observations as in the stationary environment arise: first, both the default PNN and the WPNN have similar

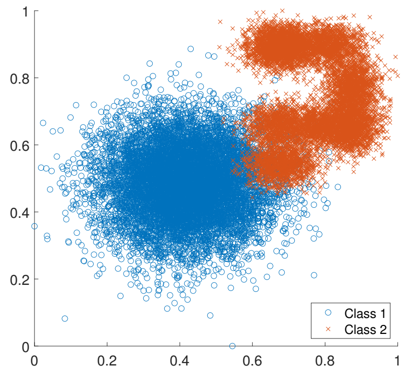

Fig. 7: Example of the non-stationary dataset.

classification performance with only one misclassified data point when the first data of the second class appears. Regarding the best result out of the three settings for the PNN, as fewer data points are involved in the training, the computation time of the chosen PNN ($N = 500$) is lower when compared to the default PNN. However, the instantaneous accuracy (Fig. 8) is also lower than the WPNN and the default PNN.

For these tests, the best configurations are: WPNN with *Linear* order B-spline with $j_0 = 1$, $\alpha = \frac{1}{100}$, default PNN with $\sigma = 0.01$, and PNN ($N = 500$) with $\sigma = 0.001$.

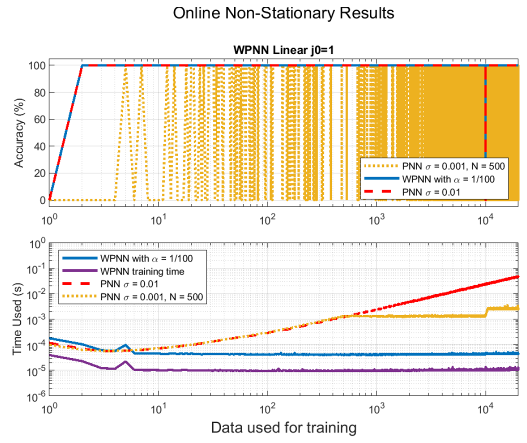

Fig. 8: Results for the online experiment in a non-stationary context.

D. *Online classification using real-world datasets*

In this section, the non-stationary version of the algorithm (Algorithm 4) is assessed. For this purpose, the dataset *HTTP* derived from *KDD Cup 99* [38] is selected, to detect intrusion attacks that vary over time (and their concepts might also drift), as it contains only 0.4% of anomalies. This dataset contains three features from the original *KDD Cup 99*: *'duration'*,*'src_byte'* and *'dst_byte'*. The preprocessing is done according to [39].

The accumulated accuracy and the computation time of both the WPNN and the PNN are shown in Fig. 10. The evaluation metrics (obtained from the confusion matrices for each method) are Classification Rate, Precision, Recall and F1-score (shown in Table VI). The proposed WPNN has higher Classification Rate, Precision and F1-score and has less than 0.4% difference in the Recall value compared to the default PNN. Regarding the best window setting of the PNN ($N = 1000$): although it predicts the results faster than the default PNN, due to fewer data points involved in the training, its overall performance is lower.

The best configuration of the WPNN used for the *HTTP* dataset is *Cubic* order B-spline with $j_0 = 2$, $\alpha = \frac{1}{9}$; $\sigma = 0.001$ for the default PNN, and $\sigma = 0.01$ for PNN ($N = 1000$). The magnitude of the forgetting factor α was chosen taking into account the characteristic of the data stream. If the data in the stream shows abrupt changes, a larger value of α should be used to increase the sensitivity of the estimator (which could be interpreted as using a smaller window size to emphasise the more recent data). Note that it is not feasible to repeat default PNN 100 times for the *HTTP* dataset due to the significantly increased time complexity when 500,000 data points are used. Here, the default PNN was repeated four times as it takes too long to finish each trial, and the average results are reported in this section.

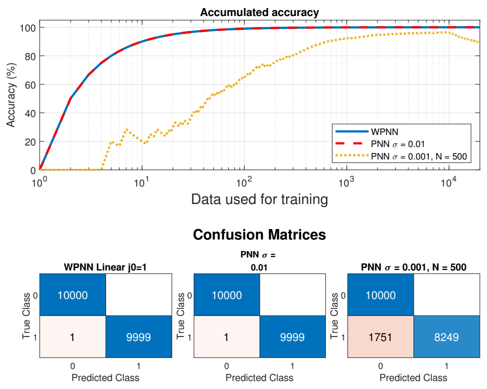

Fig. 9: Online non-stationary classification: Accumulated accuracy and the confusion matrix for the best testing configurations of WPNN and PNN.

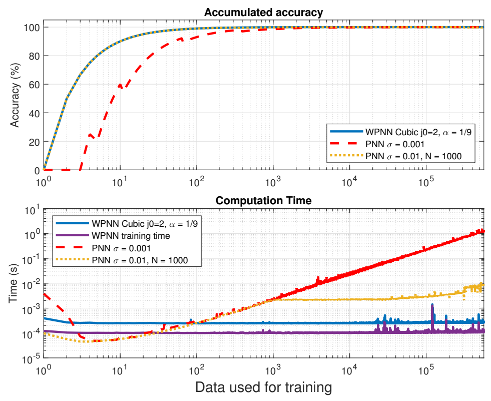

Fig. 10: *HTTP* - Accumulated accuracy and the computation time for the best configurations of WPNN and PNN.

	WPNN	PNN ($N = N_{available}$)	PNN ($N = 100$)
Classification Rate	**0.9998**	0.9994	0.9978
Precision	**0.9634**	0.8810	0.9567
Recall	0.9869	**0.9910**	0.4595
F1-score	**0.9750**	0.9327	0.6208

TABLE VI: *HTTP* - Resulting Classification Rate, Precision, Recall, and F1-score for the Class *attack*.

	WPNN	PNN ($N = N_{available}$)	PNN ($N = 100$)
Classification Rate	**0.9986**	0.9979	0.9958
Precision	0.9756	0.9705	**0.9963**
Recall	**0.9906**	0.9796	0.9014
F1-score	**0.9831**	0.9751	0.9465

TABLE VII: *NSL-KDD*: Resulting Classification Rate, Precision, Recall, and F1-score for the Class *attack*.

On a follow-up assessment, we use an advanced KDD dataset named *NSL-KDD* [40] with fewer redundant data points than the original *KDD Cup 99*. A similar preprocessing as with the previous dataset is performed and a subset of *NSL-KDD* is used for the evaluation. The obtained dataset consists of three attributes with 4.3% of attacks. The accumulated accuracy, computation time and the evaluation metrics for both the WPNN and the PNN are shown in Fig. 11 and Table VII. Table VII shows that the WPNN outperformed both PNNs in terms of Classification Rate, Recall and F1-score.

For this assessment, the best configuration of the WPNN and the PNN, for dataset *NSL-KDD* are: *Cubic* order B-spline with $j_0 = 3$, $\alpha = \frac{1}{15}$; $\sigma = 0.001$ for default PNN, and $\sigma = 0.001$ for PNN ($N = 100$).

The main observations found in these two experiments are: i) WPNN can predict the class effectively when only a few data points are provided to the model (e.g. at the beginning of the training). ii) the accuracy performance of the WPNN and the PNN tends to be similar when a sufficient amount of data points are provided. iii) the computation time for the WPNN is lower and more stable than the default PNN when the number of data points used for training is greater than 10^2. This is due to the fact that for the PNN, more neurons are required when more data becomes available. In contrast, the WPNN utilises the radial B-spline function that does not depend on the size of the dataset. Therefore, the time and space complexity of the WPNN remains constant. iv) A sliding window that chooses N most recent data for training can reduce and maintain the computation time, but it also reduces the prediction performance.

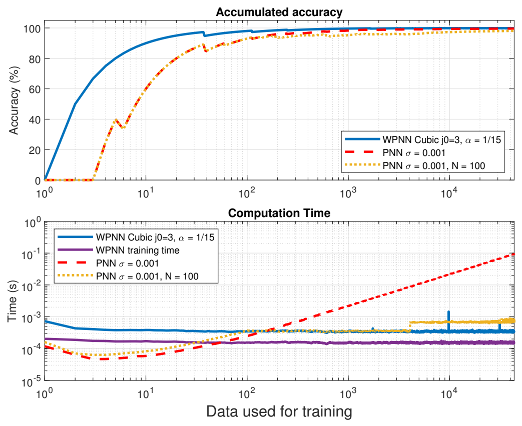

Fig. 11: *NSL-KDD* - Accumulated accuracy and the computation time for the best configurations of WPNN and PNN.

E. Benchmarking with alternative algorithms

Several methods have been proposed for data stream classification and anomaly detection in offline settings and online environments. The most common approach is the density-based algorithms, where anomalies are associated with a lower density area compared to normal data points that are grouped together to have high densities. Local Outlier Factor (LOF) [41] is one of its representatives. In this context, a low computational time LOF approach, suited for online environments, has been proposed in [42]. Regarding online data stream anomaly detection, the training and evaluation processes should match the arrival rate of the data points. Recently, the offline density-based method *STARE* [43] has been reported to have fast inference speed. Other Machine Learning (ML) methods for anomaly detection based on One-class SVM, Random Forest and Principal Component Analysis can be found in [44]–[47].

In this section, traditional and state-of-the-art algorithms for anomaly detection and data stream classification are included in the assessment (see Table VIII). The selected dataset is *HTTP*. An additional dataset *HTTP2* is also created by adding Gaussian noise with different mean and covariance (affecting 10000 data points each), in ten different time periods of the original *HTTP* dataset.

The results of the WPNN and benchmark algorithms are given in Table IX. For the offline algorithms, the evaluation metrics are the time to train and test the data, Area Under Curve (AUC), and F1-score for the class *attack* [45]. For the online algorithms, the evaluation time to classify the classes, AUC and F1-score are reported [46][4].

Reference	**Algorithm**	**Online/Offline**
LOF [41]	Local Outlier Factor	Offline
One-Class SVM [44]	SVM	Offline
PidForest [45]	Random Forest	Offline
STARE [43]	Kernel Density Estimator	Offline
online LOF [42]	Incremental Local Outlier Factor	Online
osPCA [47]	Over-sampling Principal Conponent Analysis	Online
iMondrian [46]	Isolation Mondrian forest	Online

TABLE VIII: Comparison of the proposed algorithm with benchmarks

From Table IX it can be seen that, for the online non-stationary scenario, the main advantage of the WPNN is its superior classification performance with fast computation time.

The best configuration of the WPNN used for HTTP in Table IX is *Linear* order B-spline with $j_0 = 2$, $\alpha = 1/20$; where for dataset HTTP2, *Linear* order B-spline, with $j_0 = 4$, $\alpha = 1/10$ are used. The order of the B-spline and value of j_0 should be chosen according to the complexity of the

[4]Note: osPCA did not finish within the same of order of magnitude to the ones reported and hence it is not included in the table.

	HTTP			HTTP2		
	Total Time (s)	AUC	F1	Total Time (s)	AUC	F1
LOF	**12.954**	0.353	0.010	**18.506**	0.346	0.010
One-Class SVM	1794.159	**0.995**	0.618	1545.193	0.824	0.042
PIDforest	158.762	0.989	0.416	156.009	0.931	0.104
WPNN	246.524	0.991	**0.967**	356.744	**0.996**	**0.802**
	Evaluation Time (s)	AUC	F1	Evaluation Time (s)	AUC	F1
online LOF	**157.210**	0.637	0.016	**92.515**	0.351	0.018
iMondrian	2758.200	0.979	0.008	2240.589	0.629	0.012
WPNN	164.470	**0.991**	**0.967**	220.379	**0.996**	**0.802**

TABLE IX: Results of the benchmark algorithms on the real-world datasets *HTTP* and *HTTP2*. The best case in terms of Computation time, AUC and F1-Score have been highlighted.

probability density being estimated. Note that there is a trade-off between computation time and accuracy. For example, if a higher-order B-spline and j_0 are used for *HTTP*, (i.e. *Cubic* order, $j_0 = 2$), the AUC and F1-score increase to 0.994, 0.975, respectively. However, the total time will increase from 246.5 s to 688.2 s.

The performance of the WPNN is also compared with *STARE* by using two additional datasets from YahooLab [48], which contains 67 real and 100 synthetic files, respectively. The evaluation metrics for this experiment are: Classification Rate, Precision, Recall and F1-score for the class *attack*. In this assessment, both WPNN stationary and non-stationary algorithms (i.e. Algorithms 3 and 4) are used for the YahooLab dataset since it contains stationary and non-stationary streams.

	YahooA1		YahooA2		HTTP		HTTP 2	
	WPNN	STARE	WPNN	STARE	WPNN	STARE	WPNN	STARE
CR	**0.9577**	0.9514	**0.9969**	0.9865	**0.9998**	0.9997	**0.9981**	0.9846
Precision	**0.6507**	0.2022	**0.6081**	0.1192	**0.9634**	0.9483	**0.6735**	0.1486
Recall	0.5869	**0.5974**	0.4717	**0.4893**	0.9869	**0.9873**	**0.9910**	0.6246
F1-score	**0.5442**	0.3021	**0.5204**	0.1918	**0.9750**	0.9674	**0.8020**	0.2401

TABLE X: Comparative results on the datasets *YahooLab* (time series), *HTTP* and *HTTP2*. The best case in terms of Classification Rate (CR), Precision, Recall, and F1-score have been highlighted.

Table X compares the WPNN and the *STARE* algorithm[5]. From this table, the WPNN outperforms *STARE* in terms of Classification Rate, Precision and F1-score in datasets *YahooA1, YahooA2 and HTTP*, but with a slight drop in the Recall. However, the higher value of precision and F1-score show that the WPNN is more reliable when predicting the anomaly class. In the case of the *HTTP2* dataset, WPNN outperforms all the benchmark algorithms (see Table IX & X) in terms of AUC, Classification Rate, Precision, Recall and F1-score.

The time complexity of *STARE* can be determined as $O(W + rN_G^2)$ [43], where W is the sliding window size, N_G is the number of non-empty sub-windows, and r is the ratio of the number of density changed sub-windows over N_G. The time complexity of WPNN is given in Table III. Hence, *STARE* has higher computational complexity as it relies on the window size which is case dependent and always has a larger size (in general, $W >> R, n, n_c, M_q, m$). Further, the speed of processing and predicting the class of a newly available data point for the WPNN algorithm is on average $0.290, 0.388, 0.348, 0.375$ ms for datasets *HTTP, HTTP2, YahooA1, YahooA2*, respectively.

[5]Note that the evaluation metrics for *STARE* are calculated based on the *attack ID* generated by the algorithm.

We note that, as some of the probability density functions in the YahooLab dataset are much more complex than others, three different orders of B-spline, $j_0 = 1 : 10$, and $\alpha = 1/P$, with $P = [1 : 10]$, are used by the WPNN versions. We also note that, in the online scenario, *STARE* as well as all other KDE based classification algorithms rely on sliding windows. Hence, the storage of each data point inside the sliding window is needed. In applications involving the analysis of a large number of data points (millions or billions), the use of such algorithms becomes prohibitive. In contrast, the proposed WPNN approach, which only requires the storage of the last data point, since it is based on an exponential discounting strategy, has the same computational and space complexity independent of any window size.

V. Final Remarks

In this paper, a novel wavelet-based PNN suitable for time-series and data stream applications is proposed. The performance of this network has been compared to several standard and state-of-the-art algorithms. Both synthetic and real-world datasets have been used to assess it. Significant performance enhancements are attained compared to state-of-the-art algorithms. In particular, for the online environment, it has the key advantage of constant computation time and improved classification performance.

Although the proposed WPNN can achieve excellent online performance by just changing the magnitude of the network parameter w, future research could focus on hyper-parameter automation to adapt to dynamic and complex data variations.

The proposed WPNN has constant computational complexity and involves relatively simple calculations. Hence, it is particularly suited for its implementation in cheap general-purpose electronic hardware. This opens an interesting opportunity to add adaptive and decentralized learning to a vast number of devices either connected or not connected to the internet.

Other possible applications include website traffic analysis, server resources monitoring, and network intrusion detection. Such applications have characteristics of time-ordered, specific number of features involved, imbalanced classes, fast arrival rate, and possibly concept drifts over time. The proposed WPNN can perfectly adapt to such characteristics in online environments by recursively updating the network parameters in low and constant time complexity without the model retraining step. It can be deployed in either stationary or non-stationary online environments.

References

[1] D.F. Specht. Probabilistic neural networks for classification, mapping, or associative memory. In *Neural Networks, 1988., IEEE International Conference on*, pages 525 –532 vol.1, 1988.

[2] V. Georgiou, P. Alevizos, and M. Vrahatis. Novel approaches to probabilistic neural networks through bagging and evolutionary estimating of prior probabilities. *Neural Processing Letters*, 27(2):153–162, 2008.

[3] E. Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[4] K. Z. Mao, K. C. Tan, and W. Ser. Probabilistic neural-network structure determination for pattern classification. *IEEE Transactions on Neural Networks*, 11(4):1009–1016, 2000.

[5] C.M. Bishop. *Neural networks for pattern recognition*. Clarendon press Oxford, 1995.

[6] N.N. Céncov. Evaluation of an unknown distribution density from observations. *Soviet Mathematics Doklady*, 3:1559–1562, 1962.

[7] B. Vidakovic. *Statistical Modeling by Wavelets*. Wiley New York, 1999.

[8] A.A. Safavi, J. Chen, and J.A. Romagnoli. Wavelet-based density estimation and application to process monitoring. *AIChE Journal*, 43(5):1227–1241, 2004.

[9] M. Heinen and P. Engel. An incremental probabilistic neural network for regression and reinforcement learning tasks. In Konstantinos Diamantaras, Wlodek Duch, and Lazaros Iliadis, editors, *Artificial Neural Networks ICANN 2010*, volume 6353 of *Lecture Notes in Computer Science*, pages 170–179. Springer Berlin / Heidelberg, 2010.

[10] W. Xiaoxia, P. Tino, M. A. Fardal, S. Raychaudhury, and A. Babul. Fast parzen window density estimator. In *Neural Networks, 2009. IJCNN 2009. International Joint Conference on*, pages 3267–3274, 2009.

[11] P. A. Kowalski and M. Kusy. Sensitivity analysis for probabilistic neural network structure reduction. *IEEE Transactions on Neural Networks and Learning Systems*, 29(5):1919–1932, 2018.

[12] A. V. Savchenko. Probabilistic neural network with complex exponential activation functions in image recognition. *IEEE Transactions on Neural Networks and Learning Systems*, 31(2):651–660, 2020.

[13] Zhichen Li, Huaicheng Yan, Hao Zhang, Xisheng Zhan, and Congzhi Huang. Improved inequality-based functions approach for stability analysis of time delay system. *Automatica*, 108:108416, 2019.

[14] Zhichen Li, Huaicheng Yan, Hao Zhang, Yan Peng, Ju H. Park, and Yong He. Stability analysis of linear systems with time-varying delay via intermediate polynomial-based functions. *Automatica*, 113:108756, 2020.

[15] Z. Li, Y. Bai, C. Huang, H. Yan, and S. Mu. Improved stability analysis for delayed neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(9):4535–4541, 2018.

[16] Z. Li, H. Yan, H. Zhang, X. Zhan, and C. Huang. Stability analysis for delayed neural networks via improved auxiliary polynomial-based functions. *IEEE Transactions on Neural Networks and Learning Systems*, 30(8):2562–2568, 2019.

[17] Q. Zhang and A. Benveniste. Wavelet networks. *Neural Networks, IEEE Transactions on*, 3(6):889–898, 1992.

[18] Qing Yang, Lei Gu, Dazhi Wang, and Dongsheng Wu. Fault diagnosis approach based on probabilistic neural network and wavelet analysis. In *2008 7th World Congress on Intelligent Control and Automation*, pages 1796–1799, 2008.

[19] Seok Won Lee and Boo Hee Nam. Fingerprint recognition using wavelet transform and probabilistic neural network. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No.99CH36339)*, volume 5, pages 3276–3279 vol.5, 1999.

[20] F. Lin and K. Lu. Design of fuzzy probabilistic wavelet neural network controller and its application in power control of grid-connected pv system during grid faults. In *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1725–1732, 2016.

[21] E.S. García Treviño, V. Alarcón Aquino, and J.A. Barria. The radial wavelet frame density estimator. *Computational Statistics & Data Analysis*, 130:111 – 139, 2019.

[22] M. Thuillard. A review of wavelet networks, wavenets, fuzzy wavenets and their applications. *Advances in Computational Intelligence and Learning*, pages 43–60, 2002.

[23] S.G. Mallat. *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, 3rd edition, 2008.

[24] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall/CRC, 1998.

[25] D.W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley-Interscience, 1992.

[26] E. Ribes-Gómez, S. McLoone, and G. Irwin. A taxonomy for wavelet neural networks applied to nonlinear modelling. *International Journal of Systems Science*, 39(6):607–627, 2008.

[27] T. Kugarajah and Q. Zhang. Multidimensional wavelet frames. *Neural Networks, IEEE Transactions on*, 6(6):1552–1556, 1995.

[28] D.L. Donoho, I.M. Johnstone, G. Kerkyacharian, and D. Picard. Density estimation by wavelet thresholding. *The Annals of Statistics*, 24(2):508–539, 1996.

[29] Elias Masry. Multivariate probability density estimation by wavelet methods: Strong consistency and rates for stationary time series. *Stochastic Processes and their Applications*, 67(2):177–193, 1997.

[30] Huijun Guo and Junke Kou. Strong uniform convergence rates of wavelet density estimators with size-biased data. *Journal of Function Spaces*, 2019, 2019.

[31] David L. Donoho, Iain M. Johnstone, Gérard Kerkyacharian, and Dominique Picard. Density estimation by wavelet thresholding. *The Annals of Statistics*, 24(2):508 – 539, 1996.

[32] M. Zhong, D. Coggeshall, E. Ghaneie, T. Pope, M. Rivera, M. Georgiopoulos, G. C. Anagnostopoulos, M. Mollaghasemi, and S. Richie. Gap-based estimation: Choosing the smoothing parameters for probabilistic and general regression neural networks. *IEEE International Conference on Neural Networks - Conference Proceedings*, pages 1870–1877, 2006.

[33] M. Lichman. UCI machine learning repository, 2013.

[34] Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, Salvador García, Luciano Sánchez, and Francisco Herrera. Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic & Soft Computing*, 17, 2011.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[36] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[37] J. Gama, P.P. Rodrigues, and R. Sebastião. Evaluating algorithms that learn from data streams. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1496–1500. ACM, 2009.

[38] Stephen D. Bay, Dennis F. Kibler, Michael J. Pazzani, and Padhraic Smyth. UCI machine learning repository, 1999.

[39] Kenji Yamanishi, Jun-Ichi Takeuchi, Graham Williams, and Peter Milne. On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. *Data Mining and Knowledge Discovery*, 8(3):275–300, 2004.

[40] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, 2009.

[41] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[42] D. Pokrajac, A. Lazarevic, and L. J. Latecki. Incremental local outlier detection for data streams. In *2007 IEEE Symposium on Computational Intelligence and Data Mining*, pages 504–515, 2007.

[43] Susik Yoon, Jae-Gil Lee, and Byung Suk Lee. Ultrafast local outlier detection from a data stream with stationary region skipping. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1181–1191, 2020.

[44] Bernhard Schölkopf, Robert C Williamson, Alex J Smola, John Shawe-Taylor, and John C Platt. Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588, 2000.

[45] Parikshit Gopalan, Vatsal Sharan, and Udi Wieder. PIDForest: Anomaly detection via partial identification. *Advances in Neural Information Processing Systems*, 32, 2019.

[46] Haoran Ma, Benyamin Ghojogh, Maria N Samad, Dongyu Zheng, and Mark Crowley. Isolation mondrian forest for batch and online anomaly detection. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3051–3058. IEEE, 2020.

[47] Yuh-Jye Lee, Yi-Ren Yeh, and Yu-Chiang Frank Wang. Anomaly detection via online oversampling principal component analysis. *IEEE transactions on knowledge and data engineering*, 25(7):1460–1470, 2012.

[48] YahooLabs. S5 - A Labeled Anomaly Detection Dataset, version 1.0, 2015.

Edgar S. García-Treviño received the Ph.D. degree in electrical and electronic engineering from Imperial College London, London, U.K., in 2014. From 2014 to 2019, he has been a Post-Doctoral Scholar at several universities and institutes in Mexico. Currently, he is a senior researcher at the Universidad Maya.

His current research interests include online algorithms for data analysis as well as novel artificial intelligence frameworks.

Pu Yang received the M.Eng. degree in electrical and electronic engineering from Imperial College London, London, U.K., in 2019. He is currently pursuing the Ph.D. degree in electrical and electronic engineering with Imperial College London, London, U.K.

His research interests include anomaly detection, online learning, and deep learning.

Javier A. Barria (M'02) received the Ph.D. degree in electrical and electronic engineering from Imperial College London, London, U.K., in 1994.

He is currently a Reader with the Intelligent Systems and Networks Group, Department of Electrical and Electronic Engineering, Imperial College London. He has been the joint holder of several European Union Framework and U.K. Engineering and Physical Sciences Research Council project contracts, mainly concern with his research interest on monitoring strategies on networked systems.

Dr. Barria is a Fellow of the Institution of Engineering and Technology and a Chartered Engineer in the U.K. He was a British Telecom Research Fellow from 2001 to 2002. He is an Associated Editor of the IEEE Transactions on Intelligent Transportation Systems