# Special Issue Correspondence

## Signal Classification Through Multifractal Analysis and Complex Domain Neural Networks

Witold Kinsner, *Senior Member, IEEE*, Vincent Cheung, Kevin Cannons, Joseph Pear, and Toby Martin

***Abstract*—This paper describes a system capable of classifying stochastic self-affine nonstationary signals produced by nonlinear systems. The classification and the analysis of these signals are important because these are generated by many real-world processes. The first stage of the signal classification process entails the transformation of the signal into the multifractal dimension domain, through the computation of the variance fractal dimension trajectory (VFDT). Features can then be extracted from the VFDT using a Kohonen self-organizing feature map. The second stage involves the use of a complex domain neural network and a probabilistic neural network to determine the class of a signal based on these extracted features. The results of this paper show that these techniques can be successful in creating a classification system which can obtain correct classification rates of about 87% when performing classification of such signals without knowing the number of classes.**

***Index Terms*—Classification, complex domain neural network (CNN), multifractal analysis, probabilistic neural network (PNN).**

## I. Introduction

This paper investigates the development of a software system that is capable of classifying stochastic, self-affine, nonstationary signals that originate from nonlinear systems. Such signals are often multivariate, and the system described in this paper has the ability to take these multivariate signals into account during the classification process.

The features used for classification are based on a temporal multifractal characterization of the signal, which is achieved through the computation of its variance fractal dimension trajectory (VFDT) [1]. This translation into the temporal multifractal dimension domain emphasizes the underlying complexity of the signal and more importantly, for classification, has a bounding effect. The classification based on these features is performed by a complex domain neural network (CNN) that can operate upon signal features from separate but strongly correlated signals, without losing the correlation between the signals. Furthermore, CNNs often generalize more effectively and train faster than their real-valued counterparts.

Manuscript received February 28, 2004; revised September 30, 2004. The work of W. Kinsner was financially supported in part by the Natural Sciences and Engineering Research Council of Canada through a summer scholarship and a grant. This paper was recommended by Guest Editor Y. Wang.

W. Kinsner is with the Department of Electrical and Computer Engineering and the Institute of Industrial Mathematical Sciences, University of Manitoba, Winnipeg, MB R3T 5V6, Canada and also with the Telecommunications Research Laboratories, TRLabs, Winnipeg, MB R3T 6A8, Canada (e-mail: w.kinsner@ieee.org).

V. Cheung and K. Cannons were with the Department of Electrical and Computer Engineering, Signal and Data Compression Laboratory, University of Manitoba, Winnipeg, MB R3T 5V6, Canada. They are now with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: vincent@psi.utoronto.ca; kevin.cannons@utoronto.ca).

J. Pear and T. Martin are with the Department of Psychology, University of Manitoba, Winnipeg, MB R3T 2N2, Canada (e-mail: pear@cc.umanitoba.ca; ummarti7@cc.umanitoba.ca).

Digital Object Identifier 10.1109/TSMCC.2006.871148

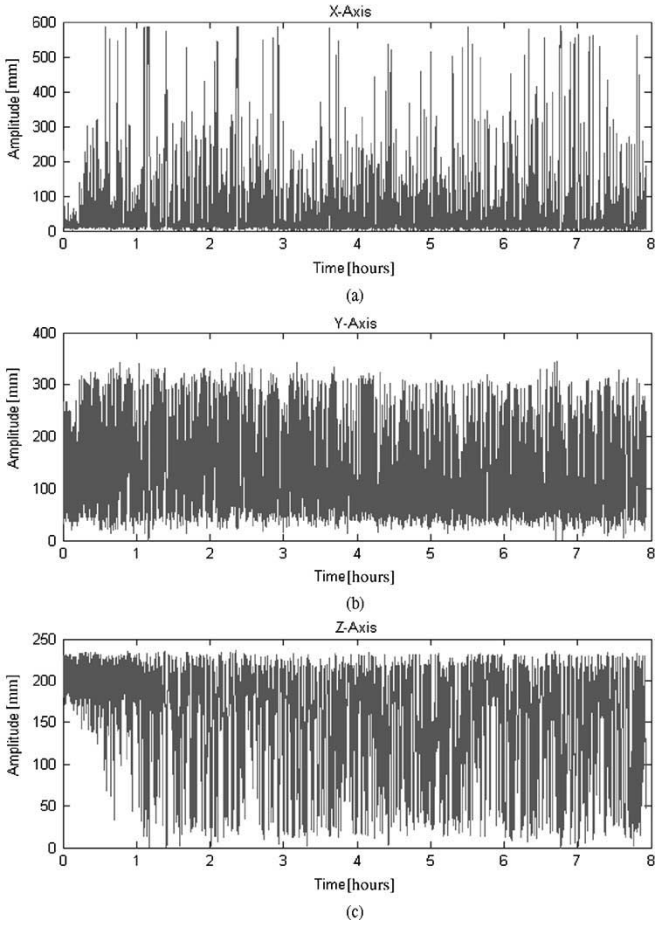


Fig. 1. Fish trajectory signal along the (a) $x$, (b) $y$, and (c) $z$ directions.

While the classification system implemented for this paper is not specific to any particular signal, spatio-temporal recordings of a Siamese fighting fish when presented with various stimuli during dishabituation experiments were used to evaluate the performance of the system. Habituation refers to a decrease in responsiveness to a repeated stimulus; the restoration of this responsiveness is referred to as dishabituation, which was performed in these experiments through the introduction of a different stimulus. A stereoscopic camera system was used to track and record the three-dimensional (3-D) Cartesian coordinates of the fish over an 8-h period. A sampling rate of ten samples/s was used for these recordings, which is valid according to the Nyquist sampling theorem and the known physical limits of the fish [2]. Each sample in the signal was recorded with an accuracy of two decimal places. Finally, the recording device performed no filtering of the signal and thus, did not alter its bandwidth.

A sample of this dishabituation signal is shown in Fig. 1. Stimuli applied during these experiments were on the $y$–$z$ plane at $X = 0$ mm and on the $x$–$y$ plane at approximately $Z = 225$ mm. Since there were no stimuli along the $y$ axis and it was the least accurate because it was resolved indirectly through the stereoscopic vision, the Y-component of the signal was not used for classification in this paper [2]. An added difficulty in analyzing these signals was that they

contained an unknown number of classes, but this was overcome using clustering algorithms.

An overview of the desired behavioral analysis and the techniques used in this paper for classification of the behavioral classes is provided in Section II. Details of the experiments performed with the dishabituation signals and the classification system are presented in Section III.

## II. Background

### A. Animal Behavioral Analysis

Experimental analysis is an effective means of discovering laws that govern the interaction between animal behavior and its environment. Because behavior is a continuous phenomenon, this effort requires that we subdivide or classify behavior in some way.

One way to form response classes is to create contingencies that are satisfied by a variety of response topographies and to study selected properties of any behavior that satisfies those contingencies. For example, a rat in an experimental chamber may be trained to depress a lever, and the frequency of this behavior will be a function of which lever presses, if any, produce access to food. This approach of defining *functional* response classes has helped to produce powerful and general learning principles, but the effects of the experimental contingencies on other properties of the behavior often go unstudied. That is, the rat can do many different things that will lower the lever and may be engaged in a variety of activities between lever presses.

The computer-aided video tracking system described here measures behavior in a way that captures its continuity and diverse topographies, but the problem of classification remains: how do we efficiently extract interesting and relevant portions of behavior and relate these behaviors to environmental events? The system can easily generate so much data that it becomes very difficult to try to identify recurring patterns through visual inspection. The techniques described in this paper are an important step towards the automated identification and analysis of *topographical* response classes. Combining this data with analysis of functional response classes is certain to help produce a more complete science of animal behavior.

### B. VFDT

Classification of signals directly in the time domain is generally impractical because of the considerable number of recorded data points. Extracting features from the signal not only reduces the classification problem size, but it also makes the classification problem easier since the features summarize the significant characteristics of the signal. The feature extraction technique used in this paper for classification is a transformation into the temporal multifractal dimension domain by computing a VFDT [1]. An advantage of using the VFDT for classification is that it emphasizes the underlying complexity of the signal and thus helping to provide the unique identification for each class. Another advantage is that the transformation provides a bounding effect because the theoretical range of fractional dimensionality of Euclidean one-dimensional (1-D) signal is between one and two. A final important advantage of the VFDT is that it can be computed in real time, which broadens its possible applications significantly.

Before proceeding with a more detailed description of the VFDT, the concepts of fractals and fractal dimensions are explained. The fish trajectory signals that are being examined in this paper are, among other things, self-affine in nature. Self-affinity means that regardless of the magnification used when viewing an object, the object's statistical properties, structure, and complexity remain constant [3]. Fractals, in the simplest sense, are self-affine entities. As a result, fractal analysis techniques serve as an appropriate means for processing the signals in this paper.

An important characteristic of a fractal is its fractal dimension. The most familiar dimensions are the Euclidean dimensions that are discrete integral numbers. For instance, a line is considered to be a 1-D object, while a square a two-dimensional (2-D) object, and a cube a 3-D object. However, it is also possible for objects to exhibit fractional dimensions. In fact, the term fractals was coined to describe objects that have a dimension that is nonintegral. The Koch curve, for example, is a fractal curve with a dimension of approximately 1.2619 [4], meaning that it has a greater complexity than a straight line, but is not quite as complex as a 2-D object.

The significance of the dimensionality of an object is that it provides valuable information regarding the object's complexity, which in this context, refers to the singularities in the object. There are an infinite number of fractal dimensions, such as the topological dimension, box-counting dimension, self-similarity dimension, and information dimension. For an object with a single power law relationship (a monofractal), all the dimensions are the same. For an object that is a mixture of monofractals (a multifractal), the values of the dimensions are different. For the purposes of this paper, only one form of fractal dimension, the variance fractal dimension [1], is utilized to perform feature extraction.

The variance fractal dimension is based on calculations involving the variance of the amplitude increments of a signal taken at different scales. The amplitude increments of a signal over a time interval $\Delta t$ adhere to the following power law relationship:

$$\mathrm{Var}[x(t_2) - x(t_1)] \sim |t_2 - t_1|^{2H} \tag{1}$$

where $x(t)$ represents the signal and $H$ is the Hurst exponent. The Hurst exponent can be calculated via a log-log plot using

$$H = \lim_{\triangle t \to 0} \frac{1}{2} \frac{\log[\mathrm{Var}(\triangle x)_{\triangle t}]}{\log(\triangle t)}. \tag{2}$$

The variance fractal dimension $D_\sigma$ is then given by

$$D_\sigma = E + 1 - H \tag{3}$$

where $E$ is the Euclidean dimension. The Euclidean dimension is equal to the number of independent variables in the signal. Thus, since this paper concentrates solely upon Euclidean 1-D signals, $E$ can be set to 1. Equation (3) then reduces to

$$D_\sigma = 2 - H. \tag{4}$$

The process of calculating the VFDT of a signal essentially involves segmenting the entire signal into numerous subsignals, or windows, and calculating the variance fractal dimension for each of these windows. The choice of the window size is a very important aspect of the process and is specific to the type of signal under analysis. Generally, the window size is chosen to match the stationarity of the signal being studied. A second important parameter of the VFDT is the window displacement, which is the number of samples that the window is shifted for each calculation of the variance fractal dimension. If the window displacement is selected such that it is smaller than the window size, then the windows used in calculating the VFDT will overlap. It is not necessary that the windows overlap; however, using overlapping windows amplifies the details of the VFDT because the same points in the original signal are involved in multiple windows while computing the VFDT. However, the displacement should

not be given an extremely small value since this would result in significant windowing artifacts because of the excessive correlation between windows. Of the two parameters, the determination of the window displacement is more subjective in nature and usually determined experimentally.

This sliding window computation is significant in that it reveals the changes in the variance fractal dimension of the signal over time, which is shown in this paper to provide valuable information for classification purposes. Such a signal that produces a nonconstant VFDT is a temporal multifractal signal. Conversely, should the VFDT be computed from a simple monofractal signal, then the resulting VFDT will simply be constant.

The representation of a signal by its VFDT facilitates classification by the CNN. The VFDT provides a usable set of features upon which classification can be performed because of the dimensionality reduction, underlying feature exemplification, and boundedness that resulted from the transformation into the multifractal domain.

### C. CNN

Complex domain three-layer feedforward neural networks [5] are used in this paper to perform classification based upon the variance fractal dimensions.

Neural networks that work with real-valued inputs are sufficient for most situations, but when the inputs to the neural network are naturally represented as complex numbers, it is advantageous to use a neural network that takes this representation into account. The fish trajectory signal examined in this paper is a multivalued signal where each sample consists of three values, one for each of the Cartesian coordinate axes. For this particular signal, the majority of the significant information lies along two of the axes. Correspondingly, the signal can be viewed as a complex valued signal, whereby samples from the $x$ axis are used as the real part of the samples in the complex valued signal and samples from the $z$ axis are used as the imaginary part. The reason for this representation is that it emphasizes that the trajectories along each of the axes are not independent; rather, they are strongly correlated.

Complex valued data can be provided to a real domain neural network by separating the components of the complex values and providing them separately as inputs; however, any correlation between the components is lost. While in theory, real-valued neural networks have the same ability as CNNs, in practice, the training of CNNs is typically faster and they often generalize better, especially when only a sparse training set is available [5].

The architecture of complex domain three-layer feedforward neural networks is similar to their real domain counterparts; the main differences are that each input value and weight is a complex number consisting of both a real and imaginary part. The activation function used in this paper for the neurons is a scaled version of the hyperbolic tangent function $\tanh(1.5\ x)$ [6] which is applied to the magnitude of the complex valued input and then multiplied by the unit vector of the input so that the output of the activation function maintains the same direction as the input [5].

This paper uses a single output neuron for each class in order to perform classification. Since the output neurons result in binary decisions for the inclusion or exclusion of an input to a particular class, it is inefficient to employ complex-valued outputs as it does not aid in making the classification decision. Thus, for classification purposes, the imaginary part of the output of these neurons is discarded and the decisions are based solely upon the real part of the output. The network is trained to have the output neuron corresponding to the input vector's class produce a value of 0.9 and with the rest of the output neurons producing $-0.9$. The classification decision for a given input is then based upon which output neuron produces the highest activation.

The network is restricted to a single hidden layer in this paper since additional layers would increase the complexity of the network as well as the training time substantially, while not adding to the network's ability to classify or generalize significantly because a three-layer neural network is sufficient in nearly all situations. The number of neurons to place in this layer has a significant effect on the network's ability to perform the desired operation and the speed at which it executes and trains. The heuristic which specifies that the number of hidden neurons should be set to the geometric mean of the number of input and output neurons is used in this paper. Through experimentation, it was discovered that this heuristic provides a good balance in that there are enough neurons to learn the desired function without simply memorizing the inputs and the network is restrained to a reasonable size so that the training and execution time of the network is acceptable.

The training of the network is performed using a squared error cost function and the standard backpropagation algorithm extended to operate with complex values. The partial derivates of the error of the output with respect to the real and imaginary parts of the weights is used as the error gradient to indicate the direction with which to modify the weights. The modifications of the weights in each epoch is given by

$$w_{\mathrm{new}_{\mathrm{real}}} = w_{\mathrm{old}_{\mathrm{real}}} - \alpha \frac{\partial \varepsilon}{\partial w_{\mathrm{old}_{\mathrm{real}}}} \tag{5a}$$

$$w_{\mathrm{new}_{\mathrm{imag}}} = w_{\mathrm{old}_{\mathrm{imag}}} - \alpha \frac{\partial \varepsilon}{\partial w_{\mathrm{old}_{\mathrm{imag}}}} \tag{5b}$$

where $\varepsilon$ is the output error, $w$ is the weight in the network, the real and imag subscripts indicate the real and imaginary parts of the weights, and $\alpha$ is the learning rate. The full derivation of the error gradients shown in (5a) and (5b) can be found in [5].

### D. Probabilistic Neural Network (PNN)

An alternative to the CNN for classification is the PNN. The PNN is an implementation of the Bayes optimal decision rule in the form of a neural network [7]. PNNs have a number of advantages over traditional neural networks in that they tend to train orders of magnitude faster and their classification accuracy asymptotically approaches Bayes optimal decision rule. However, PNNs require a comparatively large amount of memory and more time to execute.

As a means of introduction, the mathematical concepts behind PNNs will first be presented, followed by a description of how these ideas can be molded into the framework of a multilayer feedforward neural network. The first concept that requires introduction is the Bayes optimal decision rule and how it pertains to classification. Consider a case in which there are a number of objects known to be derived from a number of different classes. Simply put, the goal of a classifier is to identify to what class a new unidentified object belongs. The Bayes optimal decision rule for determining which class an unidentified object should be assigned to is expressed as follows, in which the object is assigned to class $i$ provided that

$$h_i c_i f_i(X) > h_j c_j f_j(X), \qquad j \neq i \tag{6}$$

where $h_k$ is the prior probability that new object belongs to class $k$, $c_k$ is the cost of misclassifying an object that belongs to class $k$, and $f_k$ is the probability density function (PDF) of class $k$. It must also be noted that $X$ is the input vector that will be classified. Typically, when dealing with PNNs, the prior probabilities and cost of misclassification

are not known *a priori*, and are made to be equal; hence, Bayes optimal decision rule reduces to

$$f_i(X) > f_j(X), \qquad j \neq i. \tag{7}$$

This rule indicates that if the PDFs of the different classes are known, then the best classification decision can be made by merely making a few simple comparisons. Unfortunately, the probability distributions of the different classes are not usually known and are much too complicated to attempt to approximate with simpler distributions.

In order to circumvent the problem of unknown probability distributions, Parzen introduced a method for approximating the PDFs of each class by using training samples from each of the classes [8]. Mathematically, the PDF for a single class can be approximated using the following, where $\sigma$ is a smoothing parameter, $W$ is the weighting function, $X$ is the unknown input sample to be classified, $X_{ik}$ is the $k$th training input from the $i$th class, $n_i$ is the number of training inputs for class $i$, and $g_i(x)$ is the PDF estimate for class $i$:

$$g_i(X) = \frac{1}{n_i \sigma} \sum_{k=1}^{n_i} W\left(\frac{X - X_{ik}}{\sigma}\right). \tag{8}$$

As the number of sample inputs $n$ for class $i$ increases, the PDF estimate approaches the true PDF for class $i$. When all of the PDF estimations are calculated, they are used in (7) in place of the true PDFs and the classification decision is made. By using a Guassian weighting function, the following is obtained, where $p$ is the length of the input vector $X$:

$$g_i(X) = \frac{1}{(2\pi)^{p/2}\sigma^p n_i} \sum_{k=1}^{n_i} \exp\left[-\frac{\|X - X_{ik}\|^2}{2\sigma^2}\right]. \tag{9}$$

The scaling parameter $\sigma$ dictates how wide an area the weighting function takes into account when determining the contributions of each training sample. If $\sigma$ is selected to be too small, then only those training samples that are extremely close to the unknown input will have any major contribution to the estimated PDFs. As a result, the classification system simply breaks down to a nearest neighbor classifier. If, however, the $\sigma$ selected is too large, then even training samples that are separated by a great distance from the unknown input will have a large contribution to the estimated PDFs, causing the system to display matched filter behavior. Hence, some effort must be put forth to find an appropriate scaling parameter for the particular classification problem. Fortunately, in most problems, there is a range of values that will yield acceptable results [9]. Since there is typically a plateau of $\sigma$ values producing optimal results, extremely intense algorithms do not have to be utilized to find an exact superior $\sigma$ value.

Now that the mathematics behind the PNN has been described, the manner in which it is mapped to the architecture of a multilayer feedforward network, as first performed by Specht [7], will be summarized. The architecture of a PNN is organized into four layers: the input layer, pattern layer, summation layer, and output layer, as illustrated in Fig. 2.

The input layer accepts input values from the outside world and then distributes those inputs to each of the neurons in the pattern layer. Each neuron in the pattern layer corresponds to a training sample of the PNN. The function of each pattern layer neuron is to compute the result of the weighting function. There is a single summation layer neuron for each class. A summation layer neuron accepts the result of the weighting functions for each training vector belonging to its class and then calculates the sum in (8). There is only one neuron in the output layer in the basic PNN. The output neuron performs the classification of the input according to the Bayes optimal decision rule from (7). This neuron accepts all the results from each of the summation neurons and selects the class that generates the largest sum.

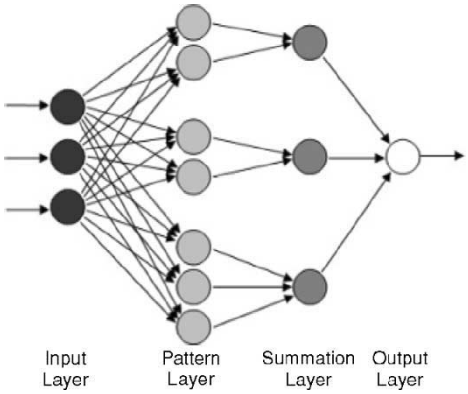


Fig. 2. Basic architecture of a PNN.

Learning in this neural network entails the optimization of the scaling parameter $\sigma$. The particular cost function that is optimized in this paper is jackknifing or leave-one-out cross-validation. At its roots, single $\sigma$ optimization is merely a single variable optimization problem. Optimization problems are typically easily resolved provided the function one that is attempting to optimize is known. The process of differentiating the function and setting the result to zero yields the function's minima and maxima. However, in the case of optimizing $\sigma$, this process cannot be done easily and other optimization means must be used. The only way to find the globally optimal $\sigma$ value is to search the entire problem space. However, this is obviously not a practical solution as it would require an infinite amount of time. Fortunately, with PNNs there is usually a rather broad range of acceptable $\sigma$ values; hence, a simple optimization algorithm can be used to produce acceptable results. Consequently, a simple and fast two step line minimization algorithm is used here [10].

The learning procedure used in this paper begins by attempting to bound the minimum. This process entails the selection of several trial $\sigma$ values within a given range and then evaluating each trial $\sigma$ to observe how it performs. When complete, the algorithm returns three $\sigma$ values. The middle $\sigma$ produces the best results out of the trial $\sigma$ values. The first and third $\sigma$ values returned are the trial points to the immediate left and right of the middle $\sigma$ value. Because of the nature of the problem space, the bounded minimum in this interval is generally quite optimal.

The minimum bounded by this interval is then determined with a sectioning algorithm, which usually locates the minimum after a small number of trial points. This portion of the line minimization algorithm begins with the best $\sigma$ produced in the bounding stage. The algorithm compares the distance between the current minimum $\sigma$ value and its left neighbor, as well as the current minimum $\sigma$ value and its right neighbor. The algorithm selects the larger of the two intervals in order to search for a better minimum. The value of the new trial $\sigma$, located in the larger of the two intervals, is dictated by the particular sectioning algorithm used. In bisectioning, the most intuitive case, the trial $\sigma$ is the point in the middle of the interval; of course, the interval could be sectioned in any number of other ways. If the trial $\sigma$ produces a better result than the current best $\sigma$, then the optimal $\sigma$ and its bounds are updated. On the other hand, if the trial $\sigma$ does not outperform the current best $\sigma$, the optimal $\sigma$ remains the same, but the bounding points are altered to narrow the search interval. This process is repeated until it is determined that the current optimal $\sigma$ produces sufficiently accurate

results or the bounded interval gets so small that new trial $\sigma$ points are only slightly different from the current best $\sigma$.

### *E. Kohonen Self-Organizing Feature Map (SOFM)*

Kohonen SOFMs [11] can be used as a clustering algorithm to determine the classes of signals. SOFMs can also be used to perform feature extraction upon the VFDT prior to classification. SOFMs are neural networks that employ unsupervised competitive learning algorithms. These neural networks are referred to as topology-preserving in that the neighborhood relations of the data are preserved and structure is imposed upon the neurons in the network. This clustering of the data based on their relations allows for the discovery of the underlying structure of the data.

An SOFM is composed of just two layers of neurons: an input layer and an output layer. The neurons in the output layer simply output the Euclidean distance (or the square of the Euclidean distance) between its weights $w_{ji}$ and the input, as shown in the following, where $n$ is the number of inputs to the network:

$$\text{out}_j = \sum_{i=1}^{n} (w_{ji} - \text{input}_i)^2. \tag{10}$$

The neuron that has the smallest output is declared the winner. It is this winner that is allowed to have its weights updated and hence the competitive nature of the learning. Training of the network is performed by modifying the weights of the winner neuron to make it more closely resemble the following input, where $w$ is the neuron weight, index $k$ signifies the winner neuron, index $i$ is the input neuron, and $\eta$ is the learning rate:

$$w_{ki_{\text{new}}} = w_{ki_{\text{old}}} + \eta \left(\text{input}_i - w_{ki_{\text{old}}}\right) \times k|\text{out}_k < \text{out}_j \forall j \neq k. \tag{11}$$

In addition, the surrounding neurons also have their weights updated during training, in order to encourage groupings and clusterings. It is this group learning from which the SOFM attains its power. The aim of the training is to have similar inputs activate neurons in the same area. The function that dictates which nearby neurons also learn is referred to as the neighborhood function. There are many different neighborhood functions that can be used, such as a constant, linearly decreasing, exponentially decreasing, or Mexican hat, which uses negative learning rates in an attempt to isolate the groupings more distinctly [12]. However, the neighborhood function has little overall effect on the SOFM as long as enough training is permitted for the network to converge to a steady state. This paper uses a constant neighborhood function where all neurons within a designated distance, or radius, of the winner neuron have their weights updated at the same rate as the winner neuron. A heuristic is used whereby the radius of the neighborhood function decreases along with the learning rate as the training progresses. Using this technique, the SOFM initially roughly defines the clusters when the radius and learning rate are high and then refines these clusters as training progresses and the radius and learning rate are decreased.

There are two common ways in which the neurons in the output layer can be organized. The first is simply in a 1-D array, and the second is in a 2-D matrix. The main difference between these layouts is how the neighborhood function is defined, in that for the 2-D case, there are neighboring neurons in multiple directions that may have their weights updated. This paper does not consider this type of SOFM as the signals used in this paper are represented as 1-D vectors and do not require the extra complexity of the 2-D SOFM.

The primary manner in which SOFMs are used in this paper is for feature extraction of signals. The number of weights in the SOFM is typically less than the length of the signal that is used to train it. Hence, when the SOFM is presented with segments from the signal, the weights of the SOFM have to adjust such that they represent the predominant characteristics that are present in the signal. In other words, the weights are adjusted such that they represent the most prominent features in the signal. When training is complete, the set of weights contained in the SOFM are commonly referred to as the codebook, as it contains a set of codes representative of the signal. These weights, or codes, within the codebook are known as codewords, and each codeword represents a specific feature that the SOFM found within the signals. In addition to the reduction in the number of features representing the signal, the use of SOFMs provides some translational robustness in that signals which are mere translations of one another result in very similar codebooks.

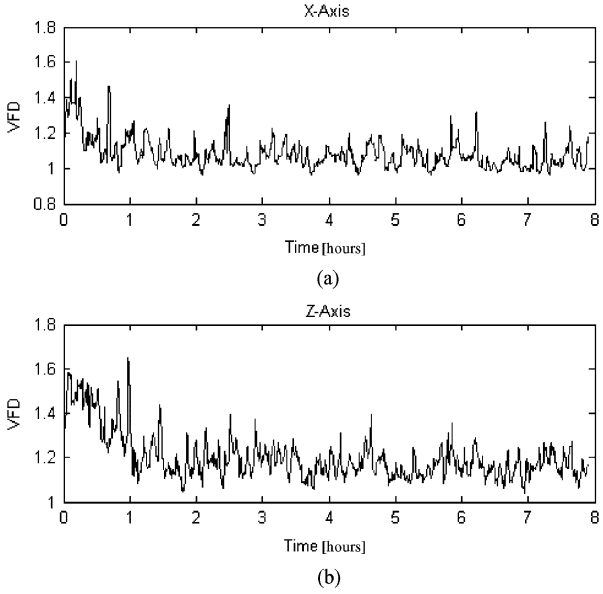


Fig. 3. VFDTs of the fish trajectory signal from Fig. 1.

The second way in which SOFMs are used for this paper is to derive the different classes from the signals by exploiting the SOFM's clustering abilities. Clustering algorithms are not the main focus of this paper but are utilized to form the training and testing sets for the purpose of classification since the annotations of the signals used in this paper were not provided. A more thorough discussion of the use of SOFM for clustering can be found in [2].

## III. Experimental Work

### *A. Computational Setup*

The dishabituation signals were segmented into lengths of 4096 samples, the longest period of constant behavior of the fish, and the classes of each of the segments were determined through clustering of the $x$ and $z$ axis segments in the time domain with a Kohonen SOFM [2]. To train the classification system, a training set of 612 segments, each consisting of 4096 samples, from nine recordings was used. The testing set applied to the system was made up of 544 segments from eight recordings that were not used for the training set.

No filtering was performed upon the signals prior to the computation of the VFDTs used to construct the training and testing sets. The VFDTs were computed using a window size of 2048 samples, the largest window in which the fractal dimension of the signals remained constant. This length of 2048 samples, or about 3.4 min, as determined by the signal's monofractality can be considered as a weak-stationarity

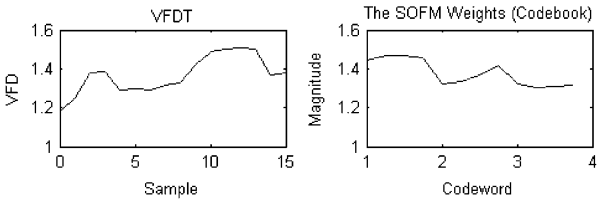


Fig. 4. VFDT segment and its codebook.

measure. A window displacement of 256 samples was used as it was discovered experimentally to give a good resolution of the VFDT while yielding a substantial dimensionality reduction.

Fig. 3 shows the VFDT of the signal from Fig. 1 along the $x$ and $z$ directions. The first thing to note about the VFDT plot is that the fractal dimensions of the signal changes, indicating that it is multifractal in time. It can further be noted that the samples of the VFDT are bounded dimensions between 1 and 2, which is essential for the classification process. Additionally, the VFDT plots visually seem to correspond to the time domain plots in that they tend to emphasize some of the characteristics in the original signal; the most exemplary characteristic being the initial large changes in the VFDT signals which correspond to the irregular motion of the fish as seen in the time-domain plot.

In the configurations of the system using the SOFM, feature extraction is performed upon these VFDT segments. The SOFMs can be used to extract features from VFDT segments which consist of a single recording or multiple recordings. In order to facilitate the extraction of features from segments involving multiple recordings, the recordings are merely appended to each other prior to being presented to the SOFM.

For the formal experiments used to verify the different configurations of the classification system, the SOFM was configured to use three codewords, each of which had a length of four. This combination means that 12 features are extracted from each VFDT segment. The configuration of three codewords with a length of four apiece was selected as it yielded very good experimental results. Scenarios where fewer codewords were used usually produced decent results and decreased the problem size even further. However, the overall classification accuracy decreases with fewer codewords.

Fig. 4 shows a sample segment of a VFDT and its corresponding SOFM codebook. The codebook contains the codewords most representative of the signal and it should also be noted that the classification problem size is reduced during the transition from the VFDT segment to the SOFM codebook in that the number of features used to represent the signal segment is reduced. The generation of all the codebooks for the entire training and testing sets, whether it be for a single recording signal or a multiple recording signal, requires no more than a third of a second when implemented in Java and run on a 1-GHz AMD Athlon computer. In the configurations of the system involving the SOFM component, it is these codebooks and their codewords that are provided as inputs into the neural networks rather than the VFDT as is done when the SOFM module is not used.

### B. CNN Experiment

The results of the classification of the input vectors in the testing set using the CNN are shown in the confusion matrix in Table I. Overall, the classification system performed well at a correct classification rate of nearly 87%.

The size of each class in the training and testing sets was proportional to their frequency of occurrence in the signals. While the first class had the smallest representation, it was so distinct that all but one of input vectors of this class was correctly classified. Input vectors from the remaining classes were also classified at a high rate, giving confidence to the abilities of the system. As the development of the testing set involved randomness in selecting the input vectors to use for testing, the 95% confidence interval for the classification rate is provided under the confusion matrix in order to bound the true classification rate of the system. The confidence interval was computed by considering each classification of the input vectors in the testing set to be a Bernoulli trial.

TABLE I
CNN EXPERIMENT CONFUSION MATRIX

| Expected | Experimental | | | | Correct classification rate (%) |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | |
| 1 | 23 | 0 | 0 | 1 | 95.83 |
| 2 | 3 | 127 | 8 | 8 | 86.99 |
| 3 | 0 | 11 | 151 | 26 | 80.32 |
| 4 | 0 | 13 | 3 | 170 | 91.40 |

Average Correct Classification Rate: 86.58%; 95% Confidence Interval: [83.72%, 89.44%].

TABLE II
CNN WITH SOFM EXPERIMENT CONFUSION MATRIX

| Expected | Experimental | | | | Correct classification rate (%) |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | |
| 1 | 24 | 0 | 0 | 0 | 95.83 |
| 2 | 2 | 129 | 10 | 5 | 88.36 |
| 3 | 0 | 13 | 150 | 25 | 79.79 |
| 4 | 0 | 16 | 10 | 160 | 86.02 |

Average Correct Classification Rate: 85.11%; 95% Confidence Interval: [82.12%, 88.10%].

The above experiment was repeated using SOFMs to perform feature extraction upon the VFDT prior to classification with the CNN. As the confusion matrix in Table II indicates, in general, the classification results when using the SOFMs were slightly lower than when the SOFMs were excluded, but they are essentially equal when confidence intervals are taken into account. Making use of the SOFMs did, however, reduce the number of inputs into the CNN from 16 in experiment one to 12 in experiment two, which caused a reduction in the training and execution times of the system. Hence, the SOFM stage did not improve the overall classification accuracy of the system, but it did reduce the classification problem size, which expedites the classification process. Thus, the classification rates remained almost the same despite the fact that fewer features were used for classification.

### C. Additional Experiments

Additional experiments were performed using a PNN as the classifier and the results are shown in Table III. For the first experiment, the $x$ axis fractal dimensions were used for classification by the PNN. As the table shows, when using just the $x$ axis fractal dimensions, the PNN based system was only able to attain an average classification rate of 67%. Despite this overall poor performance, the system did function quite well with respect to the first two classes. However, the network was simply not able to differentiate between the last two classes with much success in this first experiment.

The second experiment was identical to the first, except that the $z$ axis signals were used. As shown in Table III, this second configuration of the PNN-based classification system was not very effective in differentiating between the first three classes. However, this configuration of the system was quite successful in classifying inputs of the fourth class. It should be noted that when classification was performed

TABLE III
PNN EXPERIMENTS

| Signal | Classification rate (%) | | | | Average classification rate (%) |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | |
| X | 100 | 92 | 59 | 50 | 67 |
| Z | 63 | 29 | 47 | 91 | 58 |
| X & Z | 100 | 95 | 84 | 95 | 91 |

TABLE IV
PNN WITH SOFM EXPERIMENTS

| Signal | Classification rate (%) | | | | Average classification rate (%) |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | |
| X | 100 | 90 | 70 | 39 | 66 |
| Z | 63 | 16 | 68 | 90 | 61 |
| X & Z | 96 | 91 | 86 | 86 | 88 |

upon the $x$ axis information, as was done in the first experiment using the PNN, the system was able to classify inputs from the first two classes, but failed to perform well on the class 4 inputs. Hence, the information contained in the $x$ and $z$ axes is somewhat complementary in that the information in the $x$ axis signal can be used to identify inputs from classes 1 and 2 and the $z$ axis signals can be used to identify signals of class 4. Another observation from these two experiments is that the results from the first experiment were superior to those attained through the second experiment, which seems to indicate that the $x$ axis of the fish trajectory signals contains more important information than the $z$ axis.

After observing the results of the first two experiments involving the PNN, the goal of the third experiment was to determine the accuracy of the PNN-based system when presented with both the $x$ and $z$ axes. Hence, in this experiment, the PNN uses the exact same inputs as were utilized in the CNN experiments. As the final row of Table III indicates, the system used for this third experiment performed at a consistently high rate for all of the different classes. The individual classification rates for each class were all above 90% except for class three. When the $x$ and $z$ axes were used independently for classification, as was the case for experiments one and two, class three consistently had a fairly low classification rate. Thus, it is not all that surprising that when the PNN-based classification system uses information from both $x$ and $z$ axis that it still has difficulties with class three.

While the results for this last experiment gave slightly higher classification rates than those with the CNN, they are comparable when confidence intervals are taken into account. However, there were some differences in the training and execution times. The PNN trained two orders of magnitude faster than the CNN, while the trained CNN performed classification nearly an order of magnitude faster than the PNN.

Similar to what was done with the CNN experiments, the PNN experiments were repeated using SOFMs to perform feature extraction upon the VFDT prior to classification. Table IV shows that the addition of the SOFMs results in very similar effects as were seen when the SOFMs were incorporated with the CNN-based systems. Namely, the PNN with SOFM system yielded classification rates almost as high as those when the SOFMs were not used, while requiring less training and execution time.

Further details of these additional experiments can be found in [2].

## IV. DISCUSSION

The experiments performed upon the various configurations of the system using the fish trajectory signals provided some valuable insight into the abilities of the different systems. First, configurations using and the corresponding systems not incorporating SOFMs obtained average classification rates that were approximately equal. Essentially, the SOFMs were successful in removing the redundant information content contained within the signals so that fewer features were used in the classification process while still maintaining high classification rates. Because the SOFMs reduced the size of the classification problem, the systems that made use of the SOFMs were faster than the equivalent systems that did not use the SOFMs.

A second general observation that can be made from these experiments is that the classification systems that made use of only the $x$ axis information outperformed those systems which only used the $z$ axis information. This indicates that the $x$ axis contains more important information than the $z$ axis. Configurations that made use of both axes simultaneously, however, substantially outperformed the systems that only utilized one axis. When a classification system only makes use of one of the $x$-or $z$ axis, it is effectively taking only part of the information content of the signal into account. The systems making use of both the axes used all the information in the signals when making classification decisions, which provided higher classification rates. In improving the classification rate above configurations utilizing only a single axis, the neural networks when presented with both axes demonstrated that they were also able to differentiate between which axis information was important when making each individual classification decision. More specifically, the $x$ axis seemed to contain the most important information regarding classes one and two while the $z$ axis seemed to provide the most information regarding class four. Both the CNN and PNN were able to use the complementary information from each of the signals in order to achieve greater classification accuracy.

A third observation is related to the training and execution times of the two neural networks used to perform classification based on the selected features. It was stated in Section II-D that PNNs train orders of magnitude faster than most other neural networks and this fact is corroborated by the results of the experiments performed. When comparing the training time of the experiment from Table III on both the $x$ and $z$ axes to that of Table I, the PNN trained more than 50 times faster than the CNN. In comparing the experiments in Tables II and IV, the CNN was approximately 100 times slower than the PNN. In terms of execution time, PNNs were stated to have slower execution times than other neural networks, and again, the experiments confirmed this fact. In the experiments performed in this paper, once trained properly, the CNNs were able to classify input vectors more than 8 times faster than the PNNs.

## V. CONCLUSION

This work was done to demonstrate the feasibility of classification of self-affine signals by using variance fractal dimensions and CNN, without any prior knowledge of the number of classes in the signal. This paper has shown that a multifractal characterization of self-affine signals through variance fractal dimensions is an effective means of feature extraction as it provided a sufficient metric upon which to classify the signals used in this paper. Furthermore, the use of CNNs upon two separate, yet strongly correlated signals were used and shown to be effective in classifying these signals based on its variance fractal dimensions.

Although the classification system created for this paper was shown to perform quite successfully in classifying the nonstationary, self-similar, stochastic, multivariate dishabituation signal, there are a number of extensions that would prove to be valuable in analyzing other signals. First, the classification system could be modified to incorporate hypercomplex input signals involving $n$-dimensional signals, as there are many examples of multivariate signals composed of more than two

significant and correlated components. Second, the fractal dimension trajectory can be generalized to represent both the spatial and temporal multifractal characteristic of a signal through the Reńyi fractal dimension spectrum trajectory [13]. This representation would be valuable for classification, as some signals are not monofractal in a window, rather they are multifractal in both space and time.

## REFERENCES


[1] W. Kinsner, "Batch and real-time computation of a fractal dimension based on variance of a time series," Univ. Manitoba, Winnipeg, MB, Canada, Tech. Rep. DEL94-6, Jun. 1994.

[2] V. Cheung and K. Cannons "Signal classification through multifractal analysis and neural networks," B.S. thesis, Dept. Elect. Comput. Eng., Univ. Manitoba, Winnipeg, MB, Canada, 2003.

[3] B. Mandelbrot, *The Fractal Geometry of Nature*. San Francisco, CA: Freeman, 1982.

[4] H. O. Peitgen, H. Jürgens, and D. Saupe, *Chaos and Fractals: New Frontiers of Science*. New York: Springer-Verlag, 1992.

[5] T. Masters, *Signal and Image Processing with Neural Networks: A C++ Sourcebook*. New York: Wiley, 1994.

[6] B. L. Kalman and S. C. Kwasny, "Why tanh? Choosing a sigmoidal function," in *Proc. Int. Joint Conf. Neural Networks*, vol. 4, Jun. 1992, pp. 578–581.

[7] D. F. Specht, "Probabilistic neural networks for classification, mapping, or associative memory," in *Proc. IEEE Int. Conf. Neural Networks*, vol. 1, Jul. 1988, pp. 525–532.

[8] E. Parzen, "On estimation of a probability density function and mode," *Ann. Math. Statist.*, vol. 33, no. 3, pp. 1065–1076, 1962.

[9] P. Wasserman, *Advanced Methods in Neural Computing*. New York: Van Nostrand, 1993.

[10] T. Masters, *Advanced Algorithms for Neural Networks: A C++ Sourcebook*. New York: Wiley, 1995.

[11] T. Kohonen, *Self-Organization and Associative Memory*. Berlin, Germany: Springer-Verlag, 1984, pp. 255–9.

[12] R. Eberhart and R. Dobbins, *Neural Network PC Tools: A Practical Guide*. San Diego, CA: Academic, 1990.

[13] W. Kinsner, "Fractal and chaos engineering," Dept. Elect. Comput. Eng., Univ. Manitoba, Winnipeg, MB, Canada, Lecture Notes, 2004, p. 941.


# Cognitive Informatics Models of the Brain

Yingxu Wang, *Senior Member, IEEE,* and Ying Wang

***Abstract*—The human brain is the most complicated organ in the universe and a new frontier yet to be explored by an interdisciplinary approach. This paper attempts to develop logical and cognitive models of the brain by using cognitive informatics and formal methodologies. This paper adopts a memory-based approach to explore the brain and to demonstrate that memory is the foundation for any kind of natural or artificial intelligence. Logical structures of memories are explored, and cognitive models of the brain are proposed. Cognitive mechanisms of the brain, including hypotheses and theories on the thinking engine of the brain, long-term memory establishment, and roles of perceptive eye movement, and sleep in long-term memory development, are investigated. The models and theories can be applied to explain a wide range of fundamental phenomena in psychology, cognitive science, physiology, computing, and neural science.**

***Index Terms*—Cognitive informatics, cognitive models, memory, object-attribute-relation (OAR) model, the brain, thinking engine.**

## I. INTRODUCTION

The history of human quest to understand the brain is certainly as long as the human history itself. The study of the brain was originally conducted in the domain of philosophy. Plato (428–347 B.C.) observed that the philosophy begins in human wonder, a powerful desire to understand the world, not merely to act in it as animals do. Aristotle (394–322 B.C.) perceived psychology as the study of the soul which differentiates the animate world from the inanimate one. Psychology, as we know it, began with Rene Descartes (1596–1650), who proposed that the brain functions like a machine, in 1649. Descartes also created a framework for thinking about mind and body for philosophers and psychologists. Then, 200 years later, Wilhelm Wundt (1832–1920) founded psychology as a science disciplinary by initiating a link between physiology and philosophy via an experimental approach in 1873 [12].

It is recognized that in computing, software engineering, informatics, and artificial intelligence, almost all hard problems that are yet to be solved, share a common root in the understanding of the mechanisms of the natural intelligence and the cognitive processes of the brain. This leads to an emerging discipline of research known as cognitive informatics [4], [11].

This paper develops the cognitive informatics models of the brain. A main thread adopted in this paper is the memory-based approach that perceives memory as the foundation for any natural and artificial intelligence. Another major thread of this work is the relationship between the inherited and the acquired life functions. Structures of memories are explored in Section II. Cognitive models of the natural intelligence are developed in Section III. Cognitive mechanisms of the brain, including the hypotheses on long-term memory establishment, roles of sleeping in memory, and the cognitive mechanisms of eyes as the perceptual browser of the mind, are investigated in Section IV. On the basis of these analyses, a number of functional and cognitive mechanisms and processes of the brain will be revealed.

Manuscript received January 6, 2004; revised September 8, 2004. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada. This paper was recommended by Guest Editor W. Kinsner.

Yingxu Wang is with the Theoretical and Empirical Software Engineering Research Center, University of Calgary, Calgary, AB T2N 1N4, Canada (e-mail: Yingxu@ucalgary.ca).

Ying Wang is with the Fourth City Hospital, Shaanxi 710004, China.

Digital Object Identifier 10.1109/TSMCC.2006.871151