Radial Basis Functions Versus Geostatistics in Spatial Interpolations

Cristian Rusu[1] and Virginia Rusu[2]
1 Pontificia Universidad Católica de Valparaíso, Escuela de Ingeniería Informática, Av. Brasíl No. 2241, Valparaíso, Chile, cristian.rusu@ucv.cl, WWW home page: http://www.inf.ucv.cl
2 North University of Baia Mare, Faculty of Sciences, Victoriei 76, Baia Mare, Romania, crusu@vtr.net, WWW home page: http://www.ubm.ro

Abstract. A key problem in environmental monitoring is the spatial interpolation. The main current approach in spatial interpolation is geostatistical. Geostatistics is neither the only nor the best spatial interpolation method. Actually there is no "best" method, universally valid. Choosing a particular method implies to make assumptions. The understanding of initial assumption, of the methods used, and the correct interpretation of the interpolation results are key elements of the spatial interpolation process. A powerful alternative to geostatistics in spatial interpolation is the use of the soft computing methods. They offer the potential for a more flexible, less assumption dependent approach. Artificial Neural Networks are well suited for this kind of problems, due to their ability to handle non-linear, noisy, and inconsistent data. The present paper intends to prove the advantage of using Radial Basis Functions (RBF) instead of geostatistics in spatial interpolations, based on a detailed analyze and modeling of the SIC2004 (Spatial Interpolation Comparison) dataset.

1 Introduction

A key problem in many fields (including environmental monitoring) is spatial interpolation (sometimes referred as "surface modeling"). It consists of estimating the values of z variable at any location, based on set of (x_i, y_i, z_i) samples, which usually have a non-uniform distribution. Input data represent z values samples at given (x, y) locations, usually called control points. The problem occurs in geology, geophysics, meteorology, environmental sciences, agriculture, engineering, economy, medicine, social sciences, etc. [9], [19], [22], [23].
Two classes of methods are generally used in spatial interpolations: (1) triangulation, and (2) gridding. Triangulation requires a tessellation by an optimal network of

triangles, with control points at all apices. The triangles set represents an approximation of the surface. A regular array of data is generated by gridding, z parameter being estimated on the grid nodes, based on a set of control points. Gridding offers at least two major advantages over triangulation: (1) it is not necessary to sample the extreme points of the surface to be estimated, and (2) subsequent operations on grid data are facilitated. Usually gridding is not an aim by itself; it is a preliminary step for further processing.

2 Geostatistics in Spatial Interpolations

The main current approach in spatial interpolation nowadays is geostatistical. Geostatistics was originated by the application of statistical methods to the study of geological phenomenon. A complex theory was later developed, being applied not only to earth sciences, but also to many other areas: natural, economic, social phenomenon, among others. Geostatistics use regionalized variables, which values are not random; neither are exactly describable by a function. A regionalized variable may consist of a drift component and residual. A third error component has to be considered.
Geostatistical interpolation estimates values by kriging. Kriging is an exact interpolator which uses geostatistical techniques to calculate the autocorrelation between data points, and produce a minimum variance unbiased estimate, taking in consideration the spatial configuration of the underlying phenomenon.
Geostatistics is neither the only nor the best spatial interpolation method. Actually there is no “best” method, universally valid [3], [12], [19], [24]. The choice of interpolation method may vary, mainly according to the type and nature of data, and the aim of modeling. Choosing of a particular method implies to make assumptions. The understanding of initial assumptions, of the used methods, and the correct interpretation of the interpolation results are key elements of the spatial interpolation process. Comparison between methods can be made based on criteria as goodness of representation (errors in honoring control points), dependency on data distribution, number of control points that can be handled, ease of implementation, speed of computation.

3 Soft Computing Methods in Spatial Interpolations

3.1 Soft Computing Methods

Soft computing methods offer the potential of a more flexible, less assumption dependent approach in spatial interpolations. Even if their validness as spatial interpolation methods was proved by many authors, their use in practice is still limited [2], [5], [8], [10], [20], [25]. Soft Computing differs from conventional (hard) computing in that, unlike hard computing, it is tolerant of imprecision, uncertainty, partial truth, and approximation [11].

3.2 Artificial Neural Networks in Spatial Interpolations

Artificial Neural Networks (ANNs) are information processors, trained to represent the implicit relationship and processes that are inherent within a data set [1], [6], [7], [15], [16]. Sometimes spatial relationship between inputs has to be found (like in geology, for instance). Other areas require the identification of both spatial and temporal relationships (meteorology, environmental sciences, etc.).
The original inspiration for ANN was biological; so much of the terminology of ANN reflects this biological heritage. The basic structure of an ANN consists of a number of simple processing units, also known as neurons (nodes). The basic role of each node is to take the weighted sum of the inputs and process this through an activation function. A connection joins the output of one node to the input of another. Each link has a weight, which represents the strength of the connection. The values of all the weights in a network represent the current state of learning of the network, in a distributed manner. These weights are altered during the training process to ensure that the inputs produce an output that is close to the desired value.
A learning function or algorithm is used to adjust the weights of the network during the training phase. Training can be supervised or unsupervised. Hybrid training techniques and reinforcement learning are also used. During the learning period both the input and output vector are supplied to the network. The network then generates an error signal based on the difference between the actual output and the target vector. The error is used to adjust the weights of the network adequately. Following training, input data are then passed through the trained network in its non-training (recall) mode, where they are transformed within the hidden layers to provide the modeling output values.
ANNs have emerged as an option for spatial data analysis approximately a decade ago. Training data are the observation samples used to derive the predictive model. The independent (predictor) variables are known as the input variables, and the dependent variables (response) are known as the output variables. In supervised learning, an ANN makes use of the input variables and their corresponding output variables to learn the relationship between them. Once found, the trained ANN is then used to predict values for the output variables given some new input data set. For unsupervised learning, an ANN will only make use of the input variables and attempts to arrange them based on their properties, hopefully in a way that is meaningful to the analyst.

3.3 Radial Basis Functions in Spatial Interpolations

Radial Basis Functions (RBF) have various applications in practice, due to their simplicity, generality and fast learning stage [11], [13], [14]. RBF are unidirectional ANNs, of hybrid learning (incorporating both supervised and unsupervised learning). Usually RBF have a three layers' architecture: (1) input layer – sends the input information to the hidden layer, (2) hidden layer – composed by non-linear neurons (usually gaussian), and (3) output layer – composed by linear neurons.
The hidden layer's neurons work based on the distance between the input vector and the synaptic vector of each neurons (centroid). Therefore they offer a localized

response, which will have a significant intensity only if the input vector will be located near the centroid. Thus, a radial basis neuron acts as a detector that produces 1 whenever the input is identical to its weight vector, meaning that the input pattern was recognized. The output layer's neurons only compute the weighted sum of the output of the hidden layer.

The radial functions are usually symmetric, but asymmetric (ellipsoidal) functions may also be used. They will then have preferential search directions of the control points used in the interpolation, for a specific grid node. The gaussian functions are not the only type of radial functions that can be used. The type of the radial functions and their parameters are chosen based on the specific problem to solve and the characteristics of the input data.

Some of the reasons to use RBF in spatial interpolations are the following:

- depending on the radial functions type, the RBF model may offer a localized response (therefore is able to identify the local characteristics of the surface to be modeled), or a global response (identifying this way the global characteristics of the surface to be modeled),
- RBF are exact interpolators, honoring the control points when the point coincides with the grid node being interpolated,
- smoothing factors can be employed in order to reduces the effects of small-scale variability between neighboring data points.

4 RBF Versus Geostatistics

The progress made in spatial interpolation is usually presented only in journals or scientific meetings dedicated to statistics, mining, environmental etc. Users who have a different technical background often do not have in-depth knowledge of spatial interpolation methods. That is why the use of new techniques is often discouraging for newcomers. When spatial interpolation methods are integrated in software tools, they are often implemented in such a rigid way that users have no real choice in selecting the best possible method, according to the true nature of data to process, and the aim of modeling. Moreover, many required parameters are fixed, without any possible way to modify them.

The following is a comparison between RBF and geostatistics, at theoretical, correctness and efficiency levels, with special emphasis on method's usability.

4.1 Common Characteristics

The basis kernel of RBF is somehow analogous to variogram in geostatistics. The basis kernel functions define the optimal set of weights to apply to the data points when interpolating a grid node.

Both RBF and geostatistics (kriging) can be used as exact interpolators or smoothing interpolators. RBF will act like a smoothing interpolator when a smoothing factor will be incorporated to the basis function. Kriging will be a smoothing interpolator when an error nugget effect will be specified.

Both RBF and geostatistics are powerful and flexible methods, and are useful for gridding almost any type of data set. They generate quite similar results for most data sets. Computing time increases significantly when using large data sets. Precision of estimation is quite similar, excepting for small data sets, when a proper variographic study is difficult or impossible to perform, and therefore RBF give better results.

4.2 Problems with RBF

RBF architecture is actually imposed by the input data set itself. It is natural to use a number of RBF neurons equal to the number of the available control points, and to center the basis functions on the control point's locations. So a challenging problem when using ANN, the choose of the right architecture, is implicitly solved when using RBF in spatial interpolations.
Another problem to solve is the adequate choose of the type of the radial function to be used, as gaussian function is not always the best choice in spatial interpolation. Some alternative function may be multiquadric, multilog, inverse multiquadric, or natural cubic spline, among others. All these options where tested for real data sets (as the section 5 shows).
Once the radial function was chosen, setting the working parameters is by far less challenging then using geostatistics. Basically only smoothing factors have to be specified.

4.3 Problems with Geostatistics

Before actually performing the kriging, a variographic study has to be done. This may be quite a challenge, especially for inexperienced users. Based on the experimental variogram (obtained from the input data set), appropriate variogram model and adequate parameters have to be chosen. Moreover, many times different theoretical models have to be mixed in a complex all-in-one model.
The variogram is a measure of how quickly things change on the average. The underlying principle is that, on the average, two observations closer together are more similar than two observations farther apart. Because the underlying processes of the data often have preferred orientations, values may change more quickly in one direction than another. As such, the variogram is a function of direction. The variogram is a three dimensional function. There are two independent variables (the direction q, the separation distance h) and one dependent variable (the variogram value g(q,h)). The experimental variogram is a curve that displays the groups of variogram pairs on a plot of separation distance versus the estimated variogram.
Variogram modeling is not an easy or straightforward task. The development of an appropriate variogram model for a data set requires the understanding and application of advanced statistical concepts and tools. In addition, the development of an appropriate variogram model for a data set requires knowledge of the tricks, traps, pitfalls, and approximations inherent in fitting a theoretical model to real world data. An inappropriate variogram model can lead to completely false gridding results.

The development of an appropriate variogram model requires numerous decisions. These decisions can only be properly addressed with an intimate knowledge of the data at hand, and a competent understanding of the data genesis (i.e. the underlying processes from which the data are drawn).
The variogram model mathematically specifies the spatial variability of the data set and the resulting grid file. The interpolation weights, which are applied to data points during the grid node calculations, are direct functions of the variogram model. When the variogram is specified for kriging, the following parameters have to be set: sill, range, and nugget, but also the anisotropy information.

5 Study Case: SIC 2004

The Radioactivity Environmental Monitoring (REM) Group of the Institute for Environment and Sustainability at the Joint Research Center (JRC) of the European Commission has organized Spatial Interpolation Comparison Exercises (SIC97 and SIC2004). Participants were invited to estimate values of a variable observed at N locations with the help of a subset of n observed measurements. Once the participants have made their estimates, REM disclosed the true values observed at the N-n locations, so that the participants may assess the accuracy of their approach. The main objective of SIC97 and SIC2004 was to present the diversity of approaches taken by participants facing a problem that is identical for everyone, and to present the latest developments in the field of spatial statistics [3], [4], [17]. They offered an excellent occasion to test methods, compare results, and further orient research in the field of spatial interpolations.
The data used in SIC2004 were daily mean values of gamma dose rates measured in South West Germany, in an area of approximately 400 x 700 km, which includes 1008 monitoring stations. Participants were invited to estimate values of gamma dose rates variable at 808 locations, with the help of a subset of 200 observed measurements. Later on, the true 808 values where published. Additionally, 10 smaller data sets (of 200 observed measurements each one) where published, in order to allow the calibration of the methods and parameters [4]. The location of the 200 input data and the output 808 estimations are shown in fig. 1. All available SIC 2004 data sets where processed by the authors of the present paper, using various gridding methods [18], [21]. Only the results obtained by RBF and geostatistics will be shown and discussed here.
The interpolation results where compared with the real 808 values. The following statistics where used: Mean Error - ME, Mean Absolute Error - MAE, Percentage Mean Error - PMAE, Minimum Error - MIN, Maximum Error - MAX, Percentage Minimum Error - PMIN, Percentage Maximum Error - PMAX, Pearson's Coefficient of Correlation between the estimated and true values - PEAR.
The modeling results obtained by RBF are presented in a 3D view in fig. 1. The modeling results obtained by kriging are presented in a 3D view in fig. 2. Examining the two drawings, one could think that kriging brings more details, but the small differences are due, in fact, only to a different level of smoothness.

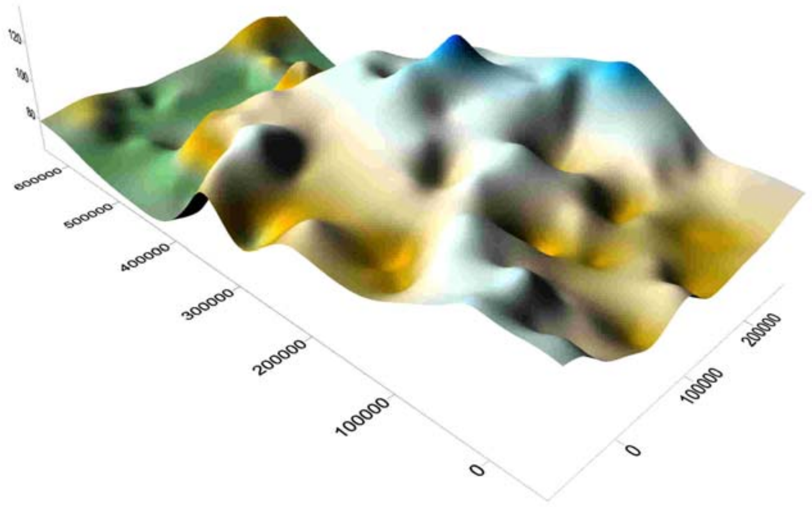


Fig. 1. Modeling results obtained by RBF

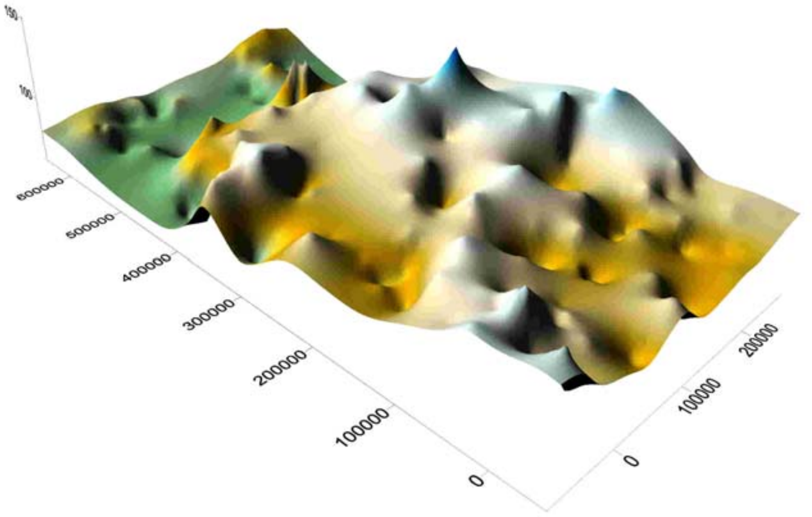


Fig. 2. Modeling results obtained by kriging

Table 1 shows the values of the above-mentioned indicators for RBF and kriging (KRG). RBF and kriging have got similar results, with a slight advantage for RBF over kriging, considering the most significant indicators (MAE, PMAE, PEAR).

Table 1. Statistics of RBF and kriging interpolation results

	ME	MAE	PMAE	MIN	MAX	PMIN	PMAX	PEAR
RBF	-1,41	9,15	9,20	-61,14	44,53	0,00	53,91	0,78
KRG	-1,31	9,28	9,34	-58,39	47,28	0,06	55,82	0,77

Various type of RBF where tested. Table 2 compares the results of applying the following functions: multiquadric - MQ, multilog - MLOG, inverse multiquadric - INVMQ, natural cubic spline - SPLINE.

Table 2. Statistics of interpolation results using various RBF types

	ME	MAE	PMAE	MIN	MAX	PMIN	PMAX	PEAR
MQ	-1,41	9,15	9,20	-61,14	44,53	0,00	53,91	0,78
MLOG	-1,19	9,82	9,94	-71,82	34,15	0,00	45,30	0,77
INVMQ	-12,18	199,40	190,78	-2017	1446	0,42	2004	-0,01
SPLINE	-2,94	53,23	54,59	-635,54	704,15	0,03	706,15	0,17

MQ and MLOG functions give the best results, but INVMQ and SPLINE should not be used in this particular case. The importance of choosing the right type of function is now obvious. Multiquadric-type radial functions offer a more global response than the gaussian-like type, so their use is particularly justified for rather sparse data, like SIC2004 data sets.

The results and the execution time are quite similar for RBF and kriging, but the easy of use of RBF is overwhelming, comparing to the use of kriging. When using RBF, user has to choose only the radial functions type and the smoothing parameter. When using kriging, a complex variogram modeling has to be done.

6 Conclusions

As we saw, spatial interpolation is a key problem in many fields, including environmental monitoring. Even if the main current approach is geostatistical, it is neither the only nor the best spatial interpolation method. There is no “best” method, universally valid. Choosing a particular method implies to make assumptions. The understanding of initial assumption, of the methods used, and the correct interpretation of the interpolation results are key elements of the spatial interpolation process.

A powerful alternative to geostatistics in spatial interpolation is the use of the soft computing methods. They offer the potential for a more flexible, less assumption dependent approach. ANNs are well suited for this kind of problems, due to their ability to handle non-linear, noisy, and inconsistent data. Particularly useful prove to be RBF.

Both RBF and geostatistics are powerful and flexible methods, and are useful for gridding almost any type of data set. They generate quite similar results for most data sets. Precision of estimation is quite similar, excepting for small data sets, when a proper variographic study is difficult or impossible to perform, and RBF give better results. RBF and geostatistics (kriging) can be used both as exact interpolators and smoothing interpolators.

Using RBF is easier than using geostatistics, even for inexperienced users. As section 4 shows, the geostatistics problems in spatial interpolations are far more complicated than the RBF problems. The development of an appropriate variogram model for a data set requires the understanding and application of advanced statistical concepts and tools. In addition, the development of an appropriate variogram model for a data set requires knowledge of the tricks, traps, pitfalls, and approximations inherent in fitting a theoretical model to real world data. An inappropriate variogram model can lead to completely false gridding results. Variogram modeling is especially difficult for relatively small data sets.

The above-mentioned conclusions where proved based on a detailed analyze and modeling of the SIC2004 (Spatial Interpolation Comparison) dataset, as the 6th section shows. That is way we strongly recommend the use of RBF in spatial interpolations.

7 References

[1] Aguilar M. and others: *Evaluación de diferentes técnicas de interpolación espacial para la generación de modelos digitales del terreno agrícola*, Mapping Interactivo, Electronic Journal, April, 2002.

[2] Demyanov V. and others: *Neural Network Residual Kriging Application For Climatic Data*, in Mapping radioactivity in the environment, edited by Dubois G, Malczewski J., and De Cort M., European Commission, Joint Research Center, Luxembourg, 2003, pp. 176-191.

[3] Dubois G.: *Spatial Interpolation Comparison 97: Foreword and Introduction*, Journal of Geographic Information and Decision Analysis, vol. 2, no. 2, 1998, pp. 1-10.

[4] Dubois G, Galmarini S.: *Introduction to the Spatial Interpolation Comparison (SIC) 2004 Exercise and Presentation of the Datasets*, Applied GIS, Vol. 1, No. 2, 2005, ISSN 1832-5505 (DOI: 10.2104/ag050009).

[5] Gilardi N., Bengio S.: *Local Machine Learning Models for Spatial Data*, Journal of Geographic Information and Decision Analysis, vol. 4, no. 1, 2000, pp. 11-28.

[6] Graubard S.: *El nuevo debate sobre la inteligencia artificial: sistemas simbólicos y redes neuronales*, Gedisa, Barcelona, España, 1999.

[7] Hilera J., Martínez V.: *Redes Neuronales Artificiales. Fundamentos, modelos y aplicaciones*, Alfaomega, Madrid, Spain, 2000.

[8] Huang Y., Wong P., Gedeon T.: *Spatial interpolation on overlapping partition surfaces using an optimized dynamic fuzzy-reasoning-based function estimator*, in

Mapping radioactivity in the environment, edited by Dubois G, Malczewski J., and De Cort M., European Commission, Joint Research Center, Luxembourg, 2003, pp. 201-210.
[9] Jones Th., Hamilton, D., Johnson C.: *Contouring Geological Surfaces with the Computer*, Van Nostrand Reinhold, New Zork, 1986.
[10] Lee S., Cho S., Wong P.: *Rainfall Prediction Using Artificial Neural Networks*, Journal of Geographic Information and Decision Analysis, vol. 2, no. 2, 1998, pp. 253-264.
[11] Mártin B., Sanz A.: *Redes Neuronales y Sistemas Difusos*, Alfaomega, Madrid, Spain, 2002.
[12] Nagy D. and others: *Comparison of various methods of interpolation and gridding*, XX General Assembly, IUGG, Vienna, Austria, 1991.
[13] Orr M.: *Introduction to Radial Basis Function Networks,* Center for Cognitive Science, University of Edinburgh, Scotland, 1996.
[14] Orr M.: *Recent Advances in Radial Basis Function Networks*, Institute for Adaptative and Neural Computation, Edinburgh University, Scotland, 1999.
[15] Pao Y.: *Adaptive Pattern Recognition and Neural Networks*, Addison-Wesley, Readings, USA, 1989.
[16] Peter A.: *Retele neuronale pentru aproximarea si predictia seriilor de timp*, Presa Universitară Clujeană, Cluj-Napoca, Romania, 2000.
[17] Rusu C.: *Interpolarea spatiala a datelor utilizand pachetul de programe ZAZA. Studiu de caz: modelarea cantitatilor de precipitatii dupa un accident nuclear*, Lucrările seminarului de creativitate matematică, vol. 9 (1999-2000), Universitatea de Nord Baia Mare, Romania, pp. 135-146.
[18] Rusu C.: *Modelado de superficies por métodos basados en redes neuronales artificiales*, Proyecto No. 209.736/2004, Pontificia Universidad Católica de Valparaíso, Chile, 2005.
[19] Rusu C.: *Modelarea suprafetelor asistata de calculator, cu aplicatii in geologie*, PhD Thesis, Universitatea Tehnică Cluj-Napoca, Romania, September 2001.
[20] Rusu C.: *Neural Network Methods in Surface Modeling. Preliminary Notes*, Creative Mathematics, Vol. 13 (2004), North University of Baia Mare, Romania, pp. 111-120.
[21] Rusu C., Rusu V.: *Uso de funciones de base radial en monitoreo ambiental*, Actas de la XXXI Conferencia Latinoamericana de Informática (CLEI 2005), Cali, Colombia, 10-14.10.2005.
[22] Swan A.R.H., Sandilands M.: *Introduction to Geological Data Analysis*, Blackwell Science, Oxford, 1995.
[23] Watson D.: *nngridr. An Implementation of Natural Neighbor Interpolation*, Vol. I of the Natural Neighbour Series, Claremont, Australia, 1994.
[24] Wingle W.: *Examining Common Problems Associated with Various Contouring Methods*, Particularly Inverce-Distance Methods, Using Shaded Relief Surfaces, Geotech ´92 Conference Proceedings, Lakewood, Colorado, USA, September, 1992.
[25] Wong K.W. and others: *Rainfall Prediction Using Neural Fuzzy Technique*, in Mapping radioactivity in the environment, edited by Dubois G, Malczewski J., and De Cort M., European Commission, Joint Research Center, Luxembourg, 2003, pp. 213-221.

Neural Networks applied to wireless communications

Georgina Stegmayer[1] and Omar Chiotti[2]

[1] C.I.D.I.S.I., Universidad Tecnológica Nacional, Lavaise 610, 3000 Santa Fe, Argentina. e-mail: gstegmay@frsf.utn.edu.ar
[2] INGAR-CONICET, Avellaneda 3654, 3000 Santa Fe, Argentina. e-mail: chiotti@ceride.gov.ar

Abstract. This paper presents a time-delayed neural network (TDNN) model that has the capability of learning and predicting the dynamic behavior of nonlinear elements that compose a wireless communication system. This model could help speeding up system deployment by reducing modeling time. This paper presents results of effective application of the TDNN model to an amplifier, part of a wireless transmitter.

1 Introduction

In new generation wireless communications - i.e. third generation (3G) standards such as WCDMA (Wideband Code Multiple Division Access) and UMTS (Universal Mobile Telecommunications System) towards which most of the current cellular networks will migrate - system component modeling has become a critical task inside the system design cycle, due to modern digital modulation schemes [1].

New standards may introduce changes in the behavior of the devices that are part of the system (e.g. mobile phones and their internal components) mainly due to the modulation schemes they use, generating nonlinearities in the behavior and memory effects (when an output signal depends on past values of an input signal). Memory effects in the time-domain cause the output of an electronic device to deviate from a linear output when the signal changes, resulting in the deterioration of the whole system performance since the device begins behaving nonlinearly. In this work we are interested in modeling the nonlinear behavior that an amplifier can have inside a wireless transmission.

Amplifiers are a major building block of modern RF digital wireless transmitters (i.e. cellular phones). Figure 1 shows a simplified block diagram of what a cellular phone communication would be. The voice coming from the phone speaker (analog signal) has to be digitalized to be transmitted through the wireless network, and this is the task of an Analog/Digital converter. The digitalized voice then has to be compressed to reduce bit rate and bandwidth. It is also codified, to format the data so the receiver can detect and minimize errors by doing the reverse operation. After that, a modulator adds the carrier signal to the data signal. The signal has to reach an antenna from the cellular phone with enough strength to guarantee the communication. But the signal suffers from attenuation and needs amplification before that. Therefore, the final element of the chain is a power amplifier (PA) which amplifies the signal before it travels to the nearer antenna and to the receiver side of the communications chain.

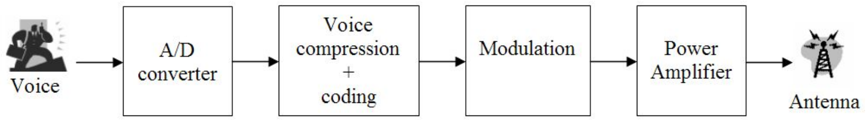


Fig. 1. Simplified block diagram of a digital wireless transmitter.

An amplifier works by increasing the magnitude of an applied signal. Amplifiers can be divided into two big groups: linear amplifiers, which produce an output signal directly proportional to the input signal, and power amplifiers which have the same function as the first ones, but their objective is to obtain maximum output power. A PA can work in different "classes": in Class A if it arrives at the limit of the linearity; and class B when it works in nonlinear regime. Moreover, in wireless communications, the transmitter itself introduces nonlinearities when operating near maximum output power [2].

Nonlinear behavior modeling has been object of increasing interest in the last years [3][4] since classical techniques that were traditionally applied for modeling are not suitable anymore. That is why new techniques and methodologies have been recently proposed, as for example neural network (NN) based modeling applied to PA modeling[5].

Neural networks, as a measurement-based technique, may provide a computationally efficient way to relate inputs and outputs, without the computational complexity of full circuit simulation or physics level knowledge [6], therefore significantly speeding up the analysis process. No knowledge of the internal structure is required and the modeling information is completely included in the device external response.

Although the NN approach has been largely exploited for static simulation, their application to dynamic systems modeling is a rather new research field. In this paper we present a new NN model for modeling nonlinear elements that belong ro a communications chain, using a network which takes into account device nonlinearity and memory effects. In particular, this paper presents the results of the proposed model to nonlinear PA modeling.

The organization of the paper is the following: in the next Section, NN-based modeling of electronic components is presented. Section 3 explains the neural network model presented in this paper and shows its architecture and parameters. In Section 4, measurements and validation results are shown. Finally, the conclusions are reported in Section 5.

2 Neural network-based modeling

Neural network-based models are nowadays seen as a potential alternative for modeling electronics elements having medium-to-strong memory effects along with high-order nonlinearity. NNs are preferred over traditional methods (i.e. equivalent-circuit, empirical models) because of their speed in implementation and accuracy. A NN model can be used during the stage of system design for a rapid evaluation of its performance and main characteristics. The model can be directly trained with measurements extracted from the real system, speeding up the design cycle. NN models can be more detailed and rapid than traditional equivalent-circuit models, more exact and flexible than empirical models, and easier to develop when a new technology is introduced. By profiting from their potential to learn a device behavior based on simulated or measured records of its input and output signals, they were used in nonlinear modeling and design of many microwave circuits and systems [7].

The increasing number of electronic devices models proposed using NNs that have appeared in the last years [8][13][10] shows their importance and interest. Many topologies of NNs are reported in the literature for modeling different types of circuits and systems, with different kinds of linear and nonlinear behavior [11]. However, until very recently, NNs for modeling were applied almost exclusively to instantaneous behavior of the input variables alone. Although this approach has been largely exploited for static simulation, their application to dynamic system is a rather new research field. Recently have appeared NN-based models taking into account the dynamic phenomena in RF microwave devices [12].

For representation of a system which has a nonlinear behavior and is dynamic, intending by dynamic not only that the device characteristic varies over time but also that it depends on past values of its controlling input variables, not any NN topology can be used. A neural model which includes time-dependence into the network architecture is the time-delayed neural network (TDNN), a special type of the well-known multilayer-perceptron (MLP). TDNNs have been successfully applied for solving the temporal processing problems in speech recognition, system identification, control and signal modeling and processing [13]. They are suited for dynamic systems representation because the continuous time system derivatives are approximated inside the model by discrete time-delays of the model variables.

A TDNN is based on the feedforward MLP neural network with the addition of tapped delay lines (Z^{-1}) which generate delayed samples of the input variables. They are used to add the history of the input signals to the model, needed for memory effects modeling. The TDNN entries include not only the current value of the input signal, but also its previous values, as illustrates figure 2. The memory depth M of the element or system analyzed is reflected on the length of the taps. The strategy followed to set the system memory is dictated by the bandwidth accuracy required.

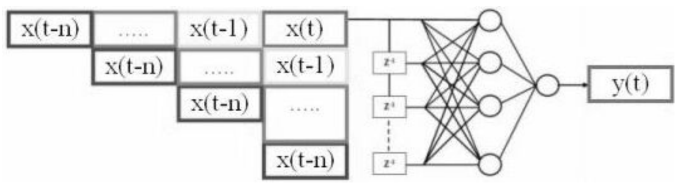


Fig. 2. Time-Delayed neural network (TDNN) model and its corresponding input data.

In this paper we propose the application of a TDNN model for modeling nonlinear and dynamic behavior of devices or elements, parts of a communication system. The model proposed is explained in detail in the next Section.

3 TDNN model

The proposed model has the classical three layers topology for universal approximation in a MLP: the input variable and its delayed samples, the nonlinear hidden layer and a linear combination of the hidden neurons outputs at the output neuron. The architecture of the TDNN is shown in figure 3.

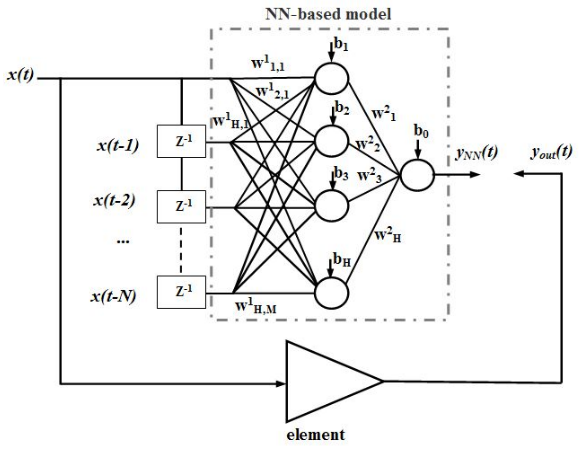


Fig. 3. Time-Delayed neural network (TDNN) to model a communications chain component.

The input layer has as inputs the samples of the independent variable, together with its time-delayed values. The model here presented shows a one-input variable dependence (x) of one output variable (y), but the model can be easily extended to more input/output variables. The variable can have a delay tap between 0 and N, then the total number of input neurons is M ($M = N + 1$).

In the hidden layer, the number h of hidden neurons varies between 1 and H. The hidden units have a nonlinear activation function. In general, the hyperbolic tangent (tanh) will be used because this function is usually chosen in the electronics field for nonlinear behavior. In our model we have used H = 10, but if necessary, the number of neurons in the hidden layer can be changed to improve the network accuracy. The hidden neurons receive the sum of the weighted inputs plus a corresponding bias value for each neuron b_h. All the neurons have bias values. This fact gives more degrees of freedom to the learning algorithm and therefore more parameters can be optimized, apart from weights, to better represent a nonlinear system.

The weights between layers are described with the usual perceptron notation, where the sub-indexes indicate the origin and the destination of the connection weight, e.g. $w_{j,i}$ means that the weight w relates the destination neuron j with the origin neuron i. However, a modification in the notation was introduced, adding a super-index to the weights, to more easily identify to which layer they belong. Therefore, w^1 means that the weight w belongs to the connection between the inputs and the first hidden layer, and w^2 means that the weight w belongs to the connection between the first hidden layer and the second one (in our particular case, the second hidden layer happens to be the output layer). In this second group of connections weights, the sub-index which indicates the destination neuron has been eliminated because there is only one possible destination neuron (the output).

This unique output neuron has a linear activation function which acts as a normalization neuron (this is usual choice in MLP models). Therefore, the output of the proposed TDNN model is calculated as the sum of the weighted outputs of the hidden neurons plus the corresponding output neuron bias (b_0), yielding equation 1.

$$y_{NN}(t) = b_0 + \left[\sum_{h=1}^{H} w_h^2 \tanh\left(b_h + \sum_{i=0}^{N} w_{h,i+1}^1 x(t-i) \right) \right] \tag{1}$$

Network initialization is an important issue for training the TDNN with the back-propagation algorithm, in particular in what respects speed of execution. In this work the initial weights and biases of the model are calculated using the Nguyen-Widrow initial conditions [14] which allow reducing training time, instead of a purely random initialization.

Once the TDNN model has been defined, it is trained with time-domain measurements of the element output variable under study ($y_{out}(t)$), which is expressed in terms of its discrete samples. To improve network accuracy and speed up learning, the inputs are normalized to the domain of the hidden neurons nonlinear activation functions (i.e for the hyperbolic tangent tanh, the interval is $[-1; +1]$). The formula used for normalization is shown in equation 2.

$$x^{norm} = \frac{2\,(x - min\{x\})}{(max\{x\} - min\{x\})} - 1 \tag{2}$$

During training, network parameters are optimized using a backpropagation algorithm such as the Levenberg-Marquardt [15], chosen due to its good performance and speed in execution. To evaluate the TDNN learning accuracy, the mean square error (mse) is calculated at each iteration k of the algorithm, using equation 3, where P is the number of input/output pairs in the training set, y_{out} is the output target and y_{NN} is the NN output.

$$mse = \frac{1}{P}\sum_{k=1}^{P} E(k)^2 = \frac{1}{P}\sum_{k=1}^{P} \left(y_{out}(k) - y_{NN}(k)\right)^2 \qquad (3)$$

The good generality property of a NN models says that it must perform well on a new dataset distinct from the one used for training. Even a excessive number of epochs or iterations on the learning phase could make performance to decrease, causing the over-fitting phenomena. That is why, to avoid it, the total amount of data available from measurements is divided into training and validation subsets, all equally spaced. We have used the "early-stopping" technique [16], where if there is a succession of training epoch in which accuracy improves only for the training data and not for the validation data, over-fitting has occurred and the learning is terminated. The obtained results are shown in the next section.

4 Measurements and validation results

For training the neural model, a dedicated test-set for accurate PA characterization has been used. It provides static and pulsed DC characterization, scattering parameter measurements, real-time load/source-pull at fundamental and harmonic frequencies, and gate and drain time-domain RF waveforms. The measurements are carried out with a Microwave Transition Analyzer and a large-signal Vector Network Analyzer.

Complete characterization was performed for different input power levels and different classes of operation at 1 GHz on a 2 ns window, as shows figure 4. Class A is biased at 50% IDSS, and class B with $IDS = 0$. A 1 mm total gate periphery GaN HEMT based on SiC with IDSS = 700 mA, has been measured at 1 GHz. The power sweep ranged from -21 to +27 dBm. Only the first 4 harmonics are taken into account.

The basic idea to characterize the nonlinear models for IDS is to collect input-output data with different tuned-load terminations, mapping the widest region in the I-V characteristic. Time-domain data have been collected only for load-pull characterization results in this case, that is three tuned-loads, 50 ohm, the best output power (Pout) and the best output efficiency (PAE). The dynamic load curves corresponding to the selected loads are rather close in class A operation, whereas they are fairly open in class B operation. This suggested to use the all three selected loads from class B characterization, and only 50 ohm load data from class A. The other characterization data will be

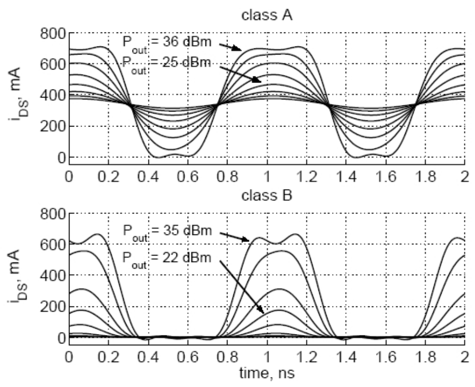


Fig. 4. Time-domain waveforms at 1 GHz at increasing power for class A (top), and B (bottom) at 50 Ohm load used for training.

used for model validation. In principle, however, this method does not need full load-pull characterization, but only to map gate and drain nonlinear models for different generic tuned-loads, in order to concern the widest region in the I-V characteristic. Input data vectors of VGS and VDS for each load and power level have been first copied and delayed as many times, to represent the network inputs, as necessary to account for memory, and then joined together to train the TDNN model with all the working classes, the selected load terminations and the power levels, simultaneously. This is shown in figure 5.

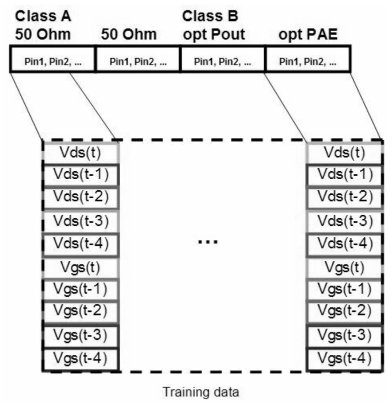


Fig. 5. TDNN model training data organization.

We report here some results for class A and B, at VDS=30V. The validation test includes the best Pout load, that has not been used for model training, and some intermediate points for input power levels not included in the training set. Results obtained show a rather good agreement between experimental and modelled data, in fact the relative mean square error is lower than 1e-04 (figure 6).

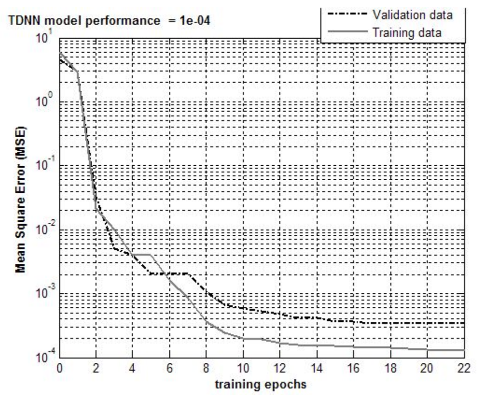


Fig. 6. TDNN model performance.

Figure 7 reports the output PA time-domain waveforms at increasing output power at 50 Ohm load: the upper plots are relative to a class A condition, while the bottom ones refer to class B operations. The left figures report the measurements, while the right ones show the TDNN simulation result. Figure 8 shows the comparison between the training (left) and validation (right) measurement data used for class A operation and the TDNN model response.

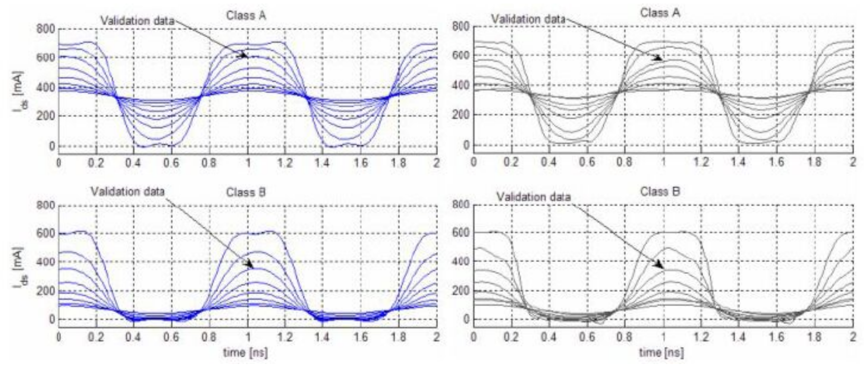


Fig. 7. Time-domain waveforms at 1 GHz at increasing power for class A (top), and B (bottom). Measurements (left), TDNN output (right). The pointed arrow waveform refers to a validation data subset.

From the comparison between measurements and model output, the waveforms good agreement in general. Also for the power levels excluded from the training data, used only for model validation. This is a key result to prove the model predictive capabilities. The model is also capable of recognizing the class in which the device is working and in consequence give a reasonable output response.

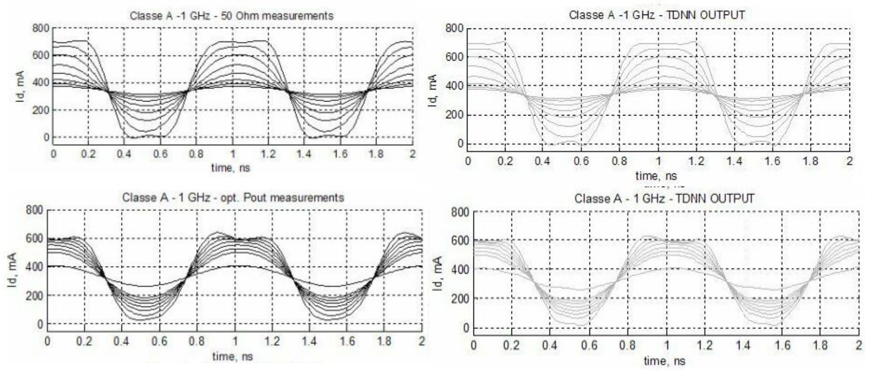


Fig. 8. Time domain waveforms at 1 GHz at increasing power for class A at 50 Ohm (used for training), and optimum Pout (used for validation). Measurements (left), TDNN output (right).

5 Conclusions

In this paper, a model that has the capability of learning and predicting the dynamic behavior of nonlinear PAs, based on a Time-Delayed Neural Network (TDNN), has been proposed. Validation and accuracy of the TDNN model in the time-domain showed good agreements between the TDNN model output data and measurements.

The TDNN model can be trained with input/output device measurements or simulations, and a very good accuracy can be obtained in the device characterization easily and rapidly. These properties make this kind of models specially suitable for new wireless communications components modeling, which are mostly nonlinear and require speed, accuracy and simplicity when designing and building the model.

6 Acknowledgement

The authors would like to thank Prof. Marco Pirola and Vittorio Camarchia, from Politecnico di Torino at Turin (IT), for providing the measurements used in this work. Also, we would like to thank Banco Ro and Red Universia for the financial support under project number INV-1334.

References

1. Evci C, Barth U, Sehier P, Sigle R. (2003) The path to beyond 3G systems: strategic and technological challenges. In: Proc. 4^{th} Int. Conf. on 3G Mobile Communication Technologies, pp. 299-303
2. Elwan H, Alzaher H, Ismail M (2001) New generation of global wireless compatibility. IEEE Circuits and Devices Magazine 17: 7-19
3. Ahmed A, Abdalla MO, Mengistu ES, Kompa G. (2004) Power Amplifier Modeling Using Memory Polynomial with Non-uniform Delay Taps. In: Proc. IEEE 34^{th} European Microwave Conf., pp. 1457-1460
4. Ku H, Kenney JS (2003) Behavioral Modeling of Nonlinear RF Power Ampli-fiers considering memory effects. IEEE Trans. Microwave Theory Tech. 51: 2495-2504
5. Root D, Wood J (2005) Fundamentals of Nonlinear Behavioral Modeling for RF and Microwave design. Artech House, Boston
6. Zhang QJ, Gupta KC, Devabhaktuni VK (2003) Artificial Neural Networks for RF and Microwave Design - From Theory to practice. IEEE Trans. Microwave Theory Tech. 51: 1339-1350
7. Zhang QJ, Gupta KC (2000) Neural Networks for RF and Microwave Design. Artech House, Boston
8. Schreurs D, Verspecht J, Vandamme E, Vellas N, Gaquiere C, Germain M, Borghs G (2003) ANN model for AlGaN/GaN HEMTs constructed from near-optimal-load large-signal measurements. IEEE Trans. Microwave Theory Tech. 51: 447-450
9. Liu T, Boumaiza S, Ghannouchi FM (2004) Dynamic Behavioral Modeling of 3G Power Amplifiers Using Real-Valued Time-Delay Neural Networks. IEEE Trans. Microwave Theory Tech. 52: 1025-1033
10. Ahmed A, Srinidhi ER, Kompa G (2005) Efficient PA modelling using Neural Network and Measurement setup for memory effect characterization in the power device. In: Proc. IEEE MTT-S International Microwave Symposium, pp. 1871-1874
11. Alabadelah A, Fernandez T, Mediavilla A, Nauwelaers B, Santarelli A, Schreurs D, Tazón A, Traverso PA (2004) Nonlinear Models of Microwave Power Devices and Circuits. In: Proc. 12^{th} GAAS Symposium, pp. 191-194
12. Xu J, Yagoub MCE, Ding R, Zhang QJ (2002) Neural-based dynamic modeling of nonlinear microwave circuits. IEEE MTT-S Int. Microwave Symp. Dig. 1: 1101-1104
13. Liu T, Boumaiza S, Ghannouchi FM (2004) Dynamic Behavioral Modeling of 3G Power Amplifiers using real-valued Time-Delay Neural Networks. IEEE Trans. Microwave Theory Tech. 52: 1025-1033
14. Nguyen D, Widrow B (1990) Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights. In: Proc. IEEE Int. Joint Conf. Neural Networks, vol. 3, pp. 21-26
15. Marquardt DW (1963) An algorithm for least-squares estimation of non-linear parameters. Journal of the Society for Industrial and Applied Mathematics 11: 431-441
16. Sjoberg J, Ljung L (1995) Overtraining, regularization and searching for a minimum, with application to neural networks. Int. J. Control 62: 1391-1407

Anomaly Detection using prior knowledge: application to TCP/IP traffic

Alberto Carrascal[1], Jorge Couchet[2], Enrique Ferreira[3] and Daniel Manrique[4]
1: NEIKER: Instituto Vasco de Investigación y Desarrollo, acarrascal@neiker.net
2: Shell Corporation, Uruguay, jorge.couchet@shell.com
3: Facultad de Ingeniería y Tecnologías, Universidad Católica del Uruguay, enferrei@ucu.edu.uy
4: Dpto. Inteligencia Artificial. Facultad de Informática. Univ. Politécnica de Madrid, dmanrique@fi.upm.es

Abstract. This article introduces an approach to anomaly intrusion detection based on a combination of supervised and unsupervised machine learning algorithms. The main objective of this work is an effective modeling of the TCP/IP network traffic of an organization that allows the detection of anomalies with an efficient percentage of false positives for a production environment. The architecture proposed uses a hierarchy of Self-Organizing Maps for traffic modeling combined with Learning Vector Quantization techniques to ultimately classify network packets. The architecture is developed using the known SNORT intrusion detection system to preprocess network traffic. In comparison to other techniques, results obtained in this work show that acceptable levels of compromise between attack detection and false positive rates can be achieved.

1 Introduction

Nowadays, Information Technology (IT) constitutes a necessity in most organizations. Actually, companies of all sizes have their vital infrastructure based on IT for all their activities. This strong dependence has its risks, e.g. an interruption of the IT services can cause severe problems, endangering the company's assets and image or even worse, its clients as well [1].

The stability of an IT platform may be affected in several ways. The main sources of instability are the following: problems related to hardware; application problems; inadequate personal training and Information Security [2].

In the last years we have seen a steep rise in the importance of the Information Security as a main issue for companies and consequently, the amount of resources invested in technological solutions to this problem has increased accordingly. Table

1, taken from CERT [3], shows the increasing risks associated to Information Security since 1990.

Table 1. Number of security incidents reported to CERT annually.

Year	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004
#Incidents	252	406	773	1,334	2,340	2,412	2,573	2,134	3,734	9,859	21,756	52,658	82,094	137,529	204,625

The growth showed in security incidents makes the development of efficient techniques for intrusion detection a necessity. Mainly, there are two ways to approach the development of an Intrusion Detection System (IDS): Misuse Detection (MD) and Anomaly Detection (AD) [4].

Techniques based on Misuse Detection work with patterns, usually called Signatures, which are configured to match attacks based on some known system vulnerability. Most IDS available today correspond to the MD type, since they are easier to implement. However, MD has some important drawbacks that affect its effectiveness: first, they are somewhat rigid, only able to detect those attacks for which a signature is available. Secondly, a signature database has to be available and maintained regularly and manually since signatures can only be created once a type of attack has been detected and therefore has compromised several systems already. Finally, an intruder with sufficient knowledge of signatures may modified the attacks slightly to avoid known signatures, cheating the IDS based on them.

The Anomaly Detection approach uses Machine Learning (ML) algorithms [5] to model normal activity in an organization. In this way it may detect deviations that can be considered abnormal or suspicious. Most of the drawbacks attributed to MD systems can be overcome by the use of an anomaly detection IDS, which may be able to adapt dynamically and automatically to the relevant characteristics of an organization's activities. In this way, it would not be necessary to know the attacks beforehand to detect them, improving the response time to a security attack. Consequently, the majority of the recent research in the Intrusion Detection area is focused in this direction as can be seen in [6,7,8].

One of the main issues with AD systems is a high percentage of false positive detections (*Normal* cases classified as *Attacks*). This is a very important issue to resolve for practical purposes. In a typical system the percentage of normal traffic is considerably larger than abnormal traffic, therefore, an IDS with a high percentage of false positives could potentially generate an alert file with most of its records due to false positives instead of real anomalies.

Several methods that use ML techniques such as Support Vector Machines (SVM) or K-Nearest Neighbor (KNN) to build an AD system. They have shown a high rate of detection but also a high percentage of false positives as well, making them very difficult to implement in a real system [8]. Recently, some other works have made use of Self Organizing Maps (SOM) [9] to address the issue of false positives. They show a comparable detection rate with a significant decrease in the number of false positive detections [10, 11]. In these works the SOMs are trained using information at packet and connection levels obtained with the IDS called Bro [12]. Although the results are positive, the percentage of false positives is still large to use such a detection system in a real situation. There may be a better way to exploit the information acquired by the SOMs to classify the incoming traffic.

In this work a new architecture aiming to solve the AD problem is presented. It is based on a hierarchy of SOMs combined with LVQ [9] to reduce the percentage of false positives. Only packet level information is used to analyze its contribution in the detection process. The system is implemented using the IDS Snort [13].

1.1 Self Organizing Maps

A SOM [9] is a type of neural network with a competitive unsupervised learning algorithm that performs a transformation of the input space. In general, it consists of a 2D dimensional map of neuron-like units. Each unit has an n-by-1 weight vector m_i associated, with n the dimension of the input space. That structure also determines a neighborhood relationship between the units. The basic SOM has a fix structure and number of units. The number of units determines the granularity of the transformation affecting the overall sensitivity and generalization ability of the map.

During the iterative training process, the unit weights will be adjusted to find common features, correlations and categories within the input data. Because of this, it is usually said that the neurons self organize themselves. Actually, the map tends to approximate the probability density of the input data. Weight vectors tend to zones where there is more input data and few units will cover zones of the input space where there is less information.

During a training step an input vector *x* is randomly chosen and the unit weight vector closer to *x* is found. That defines a winner unit c, such that

$$c = \arg\min_i \left\| x - m_i \right\| \tag{1}$$

where the symbol ||.|| refers to a norm, usually the Euclidean. The weight vectors of the unit c and its neighboring units are adjusted to get closer to the input data vector. A common update rule is given by

$$m_i(k+1) = m_i(k) + \alpha(k)\,(x - m_i(k)),\ i \in N_c(k) \tag{2}$$

with k denoting the training step, *x(k)* is the input vector chosen from the input data, $N_c(k)$ is the neighborhood set of unit c and α(k) is the learning rate at step k. The learning rate α(k) goes between 0 and 1 and decreases with k. Training evolves in two phases. During the first phase “big” values of α are used (from 0.3 to 0.99) while the second phase sees smaller values of α (below 0.1). $N_c(k)$ is usually fixed. Bigger neighborhoods are sometimes used in the first training phase.

1.2 Learning Vector Quantization

Learning Vector Quantization (LVQ) may also be considered a type of neural network like the SOM architecture [9]. An LVQ network has a set of units and weight vectors m_i associated to them. There are several training algorithms for LVQ. The algorithm used in this paper, LVQ1, is a supervised learning algorithm for classification, i.e. each input vector has a class assigned to them that the network would like to learn. Initially, each unit is assigned to a class. At step k, given a vector *x* randomly chosen from the input data, we find the weight vector closer to it m_c, with *c* given by (1). The vector m_c is updated in the following way:

$$m_c(k+1) = \begin{cases} m_c(k) + \alpha(k)(x - m_c(k)), \text{ if } x \text{ in same class as } m_c \\ m_c(k) - \alpha(k)(x - m_c(k)), \text{ else} \end{cases} \tag{3}$$

where α(k), the learning rate, is bound between 0 and 1 and can be constant or decrease monotonically with each time step.

1.3 Fundamentals of TCP/IP

Transmission Control Protocol (TCP) and Internet Protocol (IP) refers to the most widely used protocols to send and receive data through a network system. Because they work together at different levels of the system (transport and network layers respectively), they are usually named together as TCP/IP.

TCP/IP specifies how to establish and close a connection between processes in different parts of the network and how to send and receive messages between them. A TCP entity accepts messages from a process and breaks them up in pieces up to 64K bytes to send as *datagrams* by the corresponding IP entity. It is up to TCP to guarantee that all datagrams are received and the original message reassembled correctly. TCP datagrams have header and data sections. The minimum TCP header is 20 bytes long. It mainly contains source and destination addresses, packet sequence number for reassembly, several flags for connection purposes (URG, ACK, EOM, RST, SYN and FIN), a header checksum, some optional parameters and its own length.

An IP entity takes TCP datagrams, add its own IP header to generate what is called *network packets* and sends them to its destination. The IP header, which is also 20 bytes minimum, contains source and destination addresses, header and total length, protocol, type of service, flags, checksum and other attributes. In particular, type of service (1 byte) allows the user to select the quality of service it wants, from speed for voice connections to reliability for file transfer uses.

This is a very brief overview of TCP/IP, interested readers should refer to [14] for a complete explanation of computer networks and protocols. In this paper the information contained in the TCP and IP headers of the network packets is used to detect anomalies in network traffic.

2 Anomaly Detection based on SOM/LVQ

The three-layer architecture proposed is shown in Figure 1. The input to the classifier proposed is a vector comprising the main attributes of a window of predefined length of network packets. The output gives the class to which the input is classified: *Normal* or *Attack.*

The first layer examines the variation of each attribute over the time window separately. The second layer correlates the information from the first layer between attributes and makes a three-class decision: *Normal*, *Attack* or *Indefinite*. The class *Indefinite* introduced refers to the vectors that are *close* to the *Normal* class but have some *Attack* elements. The third layer decides whether the vectors in class *Indefinite* should be in *Normal* or *Attack* using a larger SOM network.

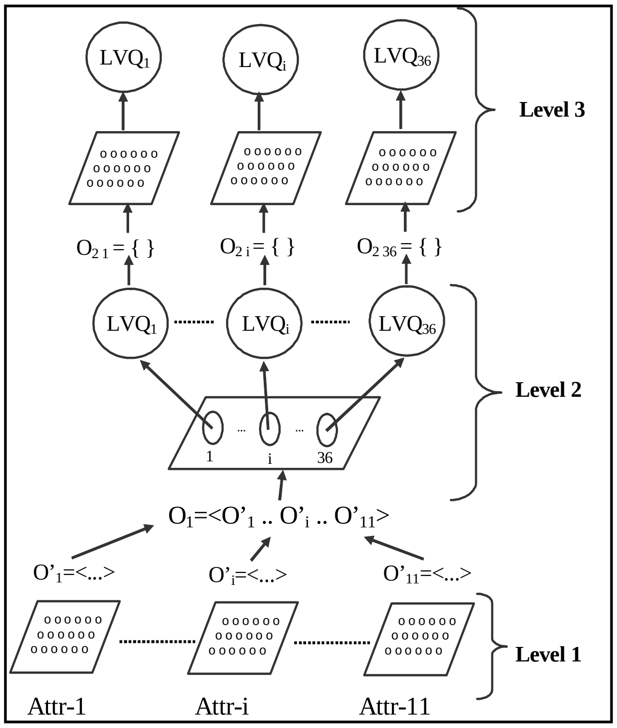


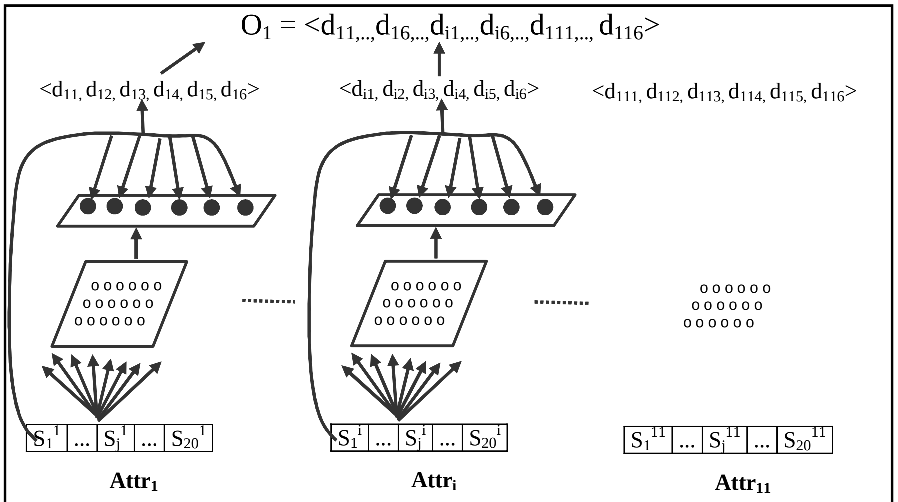
Fig. 1. Main architecture proposed.

2.1 First Level - Feature Clustering

The first level is composed of eleven SOMs, one per attribute selected to characterize TCP traffic. Each SOM takes an input vector of dimension twenty, given by a time window of length twenty of the selected attribute as shown in Figure 2. The goal at this level is to identify the main features of each attribute to obtain a model of the traffic analyzed. Once these SOMs are trained, six units per SOM are selected to reduce the dimensionality of the information to be passed to the second layer. The criteria used to select which units to select are two: the use of a *Potential Function* [10], or the selection of three units associated to normal traffic and three units associated to attack traffic.

2.2 Second Level - Aggregation and Classification

The second level of the architecture, shown in Figure 3, consists in a 6-by-6 SOM and a set of LVQs, one per SOM unit. The input vector to this SOM is composed of the distances of the input vector to the first layer, to all the units selected in the first level SOMs. Therefore, the dimension of the input vector to the second level is 6 times 11. The objective of this SOM is to capture the correlations between the features found by the SOMs of the first level, in order to make a better characterization of the traffic being studied.


$O_1 = <d_{11},..,d_{16},..,d_{i1},..,d_{i6},..,d_{111},.., d_{116}>$
$<d_{11}, d_{12}, d_{13}, d_{14}, d_{15}, d_{16}>$
$<d_{i1}, d_{i2}, d_{i3}, d_{i4}, d_{i5}, d_{i6}>$
$<d_{111}, d_{112}, d_{113}, d_{114}, d_{115}, d_{116}>$
S_1^1 ... S_j^1 ... S_{20}^1
S_1^i ... S_j^i ... S_{20}^i
S_1^{11} ... S_j^{11} ... S_{20}^{11}
Attr$_1$
Attr$_i$
Attr$_{11}$

Attr 1 = Codification of TCP Flags
Attr 2 = IP Protocol Number
Attr 3 = IP Type of Service
Attr 4 = TCP Windows
Attr 5 = Packet Size
Attr 6 = Codification of <Src Port / Src IP, Dest Port / Dest IP>
Attr 7 = Destination Port
Attr 8 = Source Port
Attr 9 = Source IP
Attr 10 = Destination IP
Attr 11 = Codification of TCP Options

Fig. 2. Attributes extracted and information flow at the first level.

Once the SOM is trained, each LVQ associated to each unit is tuned in a supervised manner using the subset of inputs to the SOM that makes the unit associated to the LVQ the *winner* in the SOM sense. The label used to train the LVQs for each input is defined as:

$$MyLabel = \frac{\#\,attack\ packets}{\#\,packets} \tag{4}$$

where the number is computed over the time window of the input vector.

It is possible to train the LVQ network in two ways: using the values of *MyLabel* as defined or discretizing the values of *MyLabel* in *Normal*, *Attack* and *Indefinite*, where: $0 < MyLabel(Indefinite) < Attack\_threshold$. In tests, the value of *Attack_threshold* used was 30 %. In the last case, the system is making a 3-class decision at this level, leaving the final classification of the packets labeled in the *Indefinite* class to the next level.

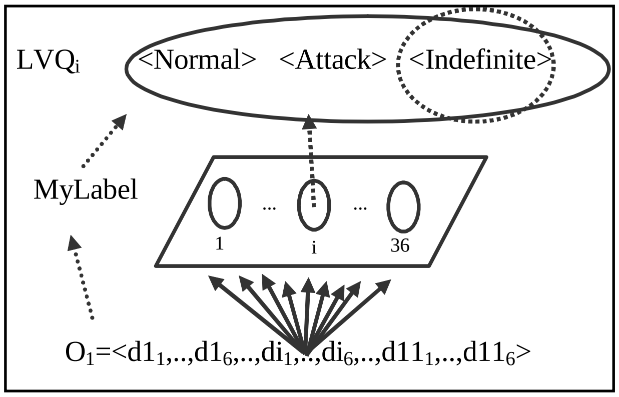

LVQi
<Normal> <Attack> <Indefinite>
MyLabel
1
i
36
O1=<d11,..,d16,..,di1,..,di6,..,d111,..,d116>

Fig 3. Second level processing.

2.3 Third Level - Indefinite Class Processing

At this level the architecture proposed analyses those input windows that have a relatively low percentage of attack packets, i.e. lower than *Attack_threshold*. Each LVQ from the second layer is associated to a 10-by-10 SOM that is trained with the packets linked to the windows classified as *Indefinite* by that particular LVQ. A LVQ network is assigned to each SOM to make the final classification using as input the distances of each packet to all the units in the associated SOM. The overall processing is shown in Figure 4.

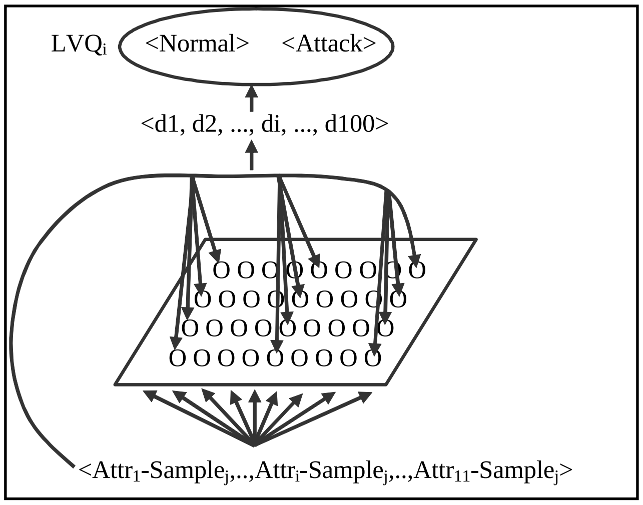


Fig. 4. Third level processing.

3 Experimental Results

The main objective in this work is to find an acceptable compromise between the Detection Rate (DR) and False Positive Rate (FPR) defined by:

$$DR = \frac{\#\,\text{Attacks Detected}}{\text{Total Number of Attacks}}, \quad FPR = \frac{\#\,\text{of False Positives}}{\text{Total Number of Normals}}$$

Data for experiments is taken from the DARPA 1998 Intrusion Detection Problem [15]. Two sets were built: the training set contains **1:514,848** records while the test set has **765,029** records. Results shown are computed over the test sets for networks trained using the training set. Training and test sets were generated from DARPA data using SNORT to extract the TCP/IP attributes selected. Only information at the packet level is used. The preprocessor developed works as a plugin to SNORT. It is also able to extract attributes at other levels (e.g. TCP connection) and protocols (e.g. UDP and ICMP). The implementation of SOM and LVQ is based on the package SOM_PAK [16]. Some modifications to this package were necessary to handle large data files as required for this application.

The packet attributes used, shown in Figure 2, were selected by its relevance in the TCP/IP protocol and hence its importance to model network traffic. The attributes that may have several values in the same packet, such as the TCP flags, were coded to reduce the dimensionality of the problem.

Tables 2, 3 and 4 summarize the results obtained with the most relevant experiments run on the architecture proposed. They use the following notation:

Classification: Refers to the amount of classifications performed by the system.

Correct: Represent the number of correct classifications per class made.

Deviations: Show the number of incorrect classifications of each type made.

The initials **N**, **I** and **A** are used for *Normal*, *Indefinite* and *Attack* respectively. In the case of deviations we use, for example, **N-I** to indicate the packets belonging to the *Normal* class but classified as *Indefinite*.

It should be noted that, at the third level the *Indefinite* class is sorted between *Normal* and *Attack*. At this level the SOM is not trained with a time window but with each of the twenty individual packets that composed each window. Due to this, each pattern at the second level is transformed into twenty patterns at the third level.

Table 2. Use of Potential function and *My Label* for training.

	Classification			**Correct**			**Deviations**					
	N	**I**	**A**	**N**	**I**	**A**	**N-I**	**N-A**	**I-N**	**I-A**	**A-N**	**A-I**
Level 2	396674	132355	236000	227250	7657	208396	41845	14382	28584	13222	140840	82853
Level 3	522269	0	71111	481170	0	50926	0	20185	0	0	41099	0

Table 3. Use of Potential function and *My Label* discretized for training.

	Classification			**Correct**			**Deviations**					
	N	**I**	**A**	**N**	**I**	**A**	**N-I**	**N-A**	**I-N**	**I-A**	**A-N**	**A-I**
Level 2	648290	66580	50159	239197	11238	33256	36754	7526	28848	9377	380245	18588
Level 3	859386	0	429934	701701	0	112103	0	317831	0	0	157685	0

Table 4. Use of 3 centers associated to *Normal* and 3 to *Attack*, and *My Label* discretized.

	Classification			Correct			Deviations					
	N	**I**	**A**	**N**	**I**	**A**	**N-I**	**N-A**	**I-N**	**I-A**	**A-N**	**A-I**
Level 2	413132	44929	306968	272431	3922	301335	6591	4455	44363	1178	96338	34416
Level 3	637367	0	261213	246029	0	233798	0	27415	0	0	391338	0

It can be observed that the best results are achieved when the selection of units to represent level one information is based on the classes to discriminate, in this case three for each class. The performance of this option is then:

$$DR = \frac{301335 + 233798/20}{301335 + 96338 + 34416} = 72\% \quad FPR = \frac{4455 + 27415/20}{272431 + 6591 + 4455} = 2\%$$

With respect to the results achieved in [10], the value of FPR is improved by 73% while DR decreased by only 19 %. In general, Table 5 resumes some results obtained in previous works where ML techniques were used for AD [8, 10].

Table 5. Results of other works using AD methods

Method	**Detection Rate**	**False Positive Rate**
Clustering	93%	10%
K-NN	91%	8%
SVM	98%	10%
SOM Hierarchy	89%	7.6%

It can be noted that the DR achieved here is below the ones obtained in the works presented in Table 5. A better inspection of the experiments also show that:

- An important percentage of the attacks presented in the data sets used corresponds to the *user to root* type. This type of attacks are quite difficult to model using TCP/IP protocol characteristics only.
- The prototype implemented is classifying TCP/IP packets up to now. Since a connection usually consists of hundreds of packets, it is expected that the **DR** may improve once the connection information is added to the system.

4 Conclusions

The AD method developed in this work introduces an efficient mechanism to reduce the false positives getting closer to generate sufficiently reliable alerts to be able to use an AD based IDS in a production environment.

It can also be observed that packet information is an important component in the detection of attacks. This work serves as a guide as to which packet information to use to model traffic for this purpose. However, the need to use connection level information is pointed out as well to reach an efficient DR with low FPR.

This investigation is based on a priori knowledge of traffic packets combined with ML techniques to set up the model and pattern recognition techniques for traffic classification. However, many steps are developed in an empirical manner which limits the conclusions that can be made. Therefore, there is the need to develop formal ways to obtain bounds on rates and efficiency of the AD methods.

From the results of this work we are following this research in two directions. First, we are exploring more efficient ML techniques for the intrusion detection problem. Besides, we are developing a theoretical framework to obtain a deeper understanding of the problem which may allow us to get robust models that are feasible to implement in the real world.

5 References

1. CSI, Computer Security Institute. *2004 CS//FBI Computer Crime and Security Survey*. (2004); http://www.gocsi.com/.

2. C. Kruegel, F. Valeur, and G. Vigna, *Intrusion Detection and Correlation - Challenges and Solutions* (Springer Verlag, New York, 2005).

3. CERT Coordination Center, CERT/CC Statistics (1988-2005); http://www.cert.org/stats/.

4. E. Carter. *Cisco Secure Intrusion Detection System* (Cisco Press, 2001).

5. T.M. Mitchell. *Machine Learning* (McGraw-Hill, 1997).

6. D.S. Kim, H.-N. Nguyen, and J.S. Park, Genetic algorithm to improve SVM based network intrusion detection system, 19th International Conference on Advanced Information Networking and Applications, Vol. 2, (2005), pp.155-158.

7. M. Markou, and S. Singh, Novelty Detection: A Review, Part II: Neural Network Based Approaches. Signal Processing, Vol. 83 (2003), pp. 2499-2521.

8. E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S Stolfo, A Geometric Framework for Unsupervised Anomaly Detection: Detecting intrusions in unlabeled data, In D. Barbara and S. Jajodia, editors, Applications of Data Mining in Computer Security (Kluwer, 2002).

9. T.Kohonen. *Self Organizing Maps*, Third Extended Edition (Springer, 2001).

10. H.G. Kayacik. *Hierarchical Self Organizing Map Based IDS on KDD Benchmark*. Master’s thesis, Dalhousie University (2003).

11. P. Lichodzijewski, A.N. Zincir-Heywood, and M.I. Heywood, Host -based Intrusion Detection Using Self-Organizing Maps, IEEE World Congress on Computational Intelligence, International Joint Conference on Neural Networks, IJCNN (2002).

12. Bro. Intrusion detection system (2005); http://www.bro-ids.org/

13. Snort. open source network intrusion prevention and detection system (2005); http://www.snort.org.

14. A. S. Tanenbaum. *Computer Networks* (2nd Edition, Prentice-Hall, 1989).

15. MIT Lincoln Laboratory (1998); http://www.ll.mit.edu/IST/ideval/data/data index.html.

16. SOM_PAK**,** Helsinki University of Technology, Laboratory of Computer and Information Science (2005); http://www.cis.hut.fi/research/som_lvq_pak.shtml.

A study on the ability of Support Vector Regression and Neural Networks to Forecast Basic Time Series Patterns

Sven F. Crone[1], Jose Guajardo[2], and Richard Weber[2]
1 Lancaster University, Department of Management Science, Lancaster LA1 4YX, Lancaster, UK s.crone@lancaster.ac.uk
2 University of Chile, Department of Industrial Engineering, Republica 701, Santiago, Chile {jguajard,rweber}@dii.uchile.cl

Abstract. Recently, novel learning algorithms such as Support Vector Regression (SVR) and Neural Networks (NN) have received increasing attention in forecasting and time series prediction, offering attractive theoretical properties and successful applications in several real world problem domains. Commonly, time series are composed of the combination of regular and irregular patterns such as trends and cycles, seasonal variations, level shifts, outliers or pulses and structural breaks, among others. Conventional parametric statistical methods are capable of forecasting a particular combination of patterns through ex ante selection of an adequate model form and specific data preprocessing. Thus, the capability of semi-parametric methods from computational intelligence to predict basic time series patterns without model selection and preprocessing is of particular relevance in evaluating their contribution to forecasting. This paper proposes an empirical comparison between NN and SVR models using radial basis function (RBF) and linear kernel functions, by analyzing their predictive power on five artificial time series: stationary, additive seasonality, linear trend, linear trend with additive seasonality, and linear trend with multiplicative seasonality. Results obtained show that RBF SVR models have problems in extrapolating trends, while NN and linear SVR models without data preprocessing provide robust accuracy across all patterns and clearly outperform the commonly used RBF SVR on trended time series.

1 Introduction

Support Vector Regression (SVR) and Artificial Neural Networks (NN) have found increasing consideration in forecasting theory, leading to successful applications in time series and explanatory forecasting in various domains, including business and management science [1, 2]. Methods form computational intelligence promise attractive features to business forecasting, being data driven, semi-parametric learning machines, permitting universal approximation of arbitrary linear or nonlinear functions from examples without a priori assumptions on the model structure, often outperforming conventional statistical approaches of ARIMA- or exponential smoothing- methods.

Despite their theoretical capabilities, NN as SVR are not established forecasting methods in business practice. Recently, substantial theoretical criticism of NN has raised skepticism regarding their ability to forecast even simple time series patterns of seasonality or trends without prior data preprocessing [3]. While all novel methods must ultimately be evaluated in an objective experiment using a number of empirical time series, adequate error measures and multiple origins of evaluation [4], the fundamental questions to their ability to approximate and generalize basic time series patterns must be evaluated beforehand. Time series can generally be characterized by the combination of basic regular patterns: level, trend, season and residual errors. For trend, a variety of linear, progressive, degressive and regressive patterns are feasible. For seasonality, an additive or multiplicative combination with level and trend further determines the shape of the final time series. Consequently, we evaluate SVR and NN on a set of artificially created time series derived from previous publications. We evaluate the comparative forecasting accuracy of each method to reflect their ability of learning and forecasting fundamental time series patterns relevant to empirical forecasting tasks.

This paper is organized as follows. First, we provide a brief introduction to SVR and NN in forecasting time series of observations. Section three presents the artificially generated time series and the experimental design. This is followed by the experimental results and their discussion. Conclusions are given in section 4.

2 Modelling SVR and NN for Time Series Prediction

2.1 Support Vector Regression

We apply the common Support Vector Regression (SVR) algorithm as proposed by Vapnik [5], which uses an ε-insensitive loss function for predictive regression problems. This function allows a tolerance degree to errors not greater than ε. The description is based on the terminology used in [6, 7]. Let $\{(x_1,y_1),\ldots.., (x_\ell,y_\ell)\}$, where $x_i \in R^n$ and $y_i \in R$, be the training data points available to build a regression model. The SVR algorithm applies a transformation function Φ to the original data points from the initial Input Space, to a higher-dimensional Feature Space *F*. In this new space, we construct a linear model, which corresponds to a non-linear model in the original space[1]:

[1] When Φ is the identity function, the Feature Space is equivalent to the Input Space, and the model constructed is linear in the original space.

$$\Phi : R^n \rightarrow F, w \in F$$
$$f(x) = \langle w, \Phi(x) \rangle + b$$

The goal when using the ε-insensitive loss function is to find a function that fits current training data with a deviation less or equal to *ε*, and at the same time is as flat as possible. This means that one seeks for a small weight vector w; one way to do that is e.g. by minimizing the quadratic norm of the vector *w* [6]. As this problem could be infeasible, slack variables *ξi, ξi** are introduced to allow error levels greater than *ε*, arriving to the formulation proposed in [5]:

$$Min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{\ell}(\xi_i + \xi_i^*)$$
$$s.t. y_i - \langle w, \Phi(x_i) \rangle - b \le \varepsilon + \xi_i$$
$$\langle w, \Phi(x_i) \rangle + b - y_i \le \varepsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \ge 0, i = 1,2,\ldots,\ell$$

This is known as the primal problem of the SVR algorithm. The objective function takes into account generalization ability and accuracy in the training set, and embodies the structural risk minimization principle [8]. Parameter *C* measures the trade-off between generalization ability and accuracy in the training data, and parameter ε defines the degree of tolerance to errors. To solve the problem stated above, it is more convenient to represent the problem in its dual form. For this purpose, a Lagrange function is constructed, and once applying saddle point conditions, it can be shown that the following solution is obtained [8]:

$$w = \sum_{i=1}^{\ell}(\alpha_i - \alpha_i^*)\Phi(x_i)$$

$$f(x) = \sum_{i=1}^{\ell}(\alpha_i - \alpha_i^*)K(x_i, x) + b$$

Here, α_i and α_i^* are the dual variables, and the expression $K(x_i,x)$ represents the inner product between $\Phi(x_i)$ and $\Phi(x)$, which is known as the kernel function [8]. The existence of such a function allows us to obtain a solution for the original regression problem, without explicitly considering the transformation $\Phi(x)$ applied to the data. In our experiments we use radial basis functions (RBF) and linear kernel functions.

Limited research has been conducted to investigate the ability of SVR for predicting different time series patterns. Experiments performed by Hansen et. al [9] compare SVR performance with 3 statistical methods (e.g. ARIMA) on predicting 9 different patterns present in real world time series. Among other patterns, they tried trends, seasonality, cycles, and combinations of them. Their experiments show SVR models outperforming the other methods on 8 of the 9 patterns; particularly, they obtained very good results using SVR for extrapolating linear and non linear trends. Guajardo et al. [10] compared SVR with ARMAX models for predicting seasonal time series in a weekly sales forecasting domain for 5 different products. Their experiments show that SVR were slightly better than ARMAX models, succeeding in extrapolating seasonal patterns (without trends) with SVR.

2.2 Neural Networks

Forecasting with non-recurrent NN may encompass prediction of a dependent variable $\hat{y}$ from lagged realizations of the predictor variable y_{t-n} , 1 or i explanatory variables x_i of metric, ordinal or nominal scale as well as lagged realizations thereof, $x_{i,t-n}$. Therefore, NNs offer large degrees of freedom towards the forecasting design, permitting explanatory or causal forecasting through estimation of a functional relationship of the form $\hat{y} = f\left(x_1, x_2, \dots, x_z\right)$, as well as general transfer function models and simple time series prediction. Following, we present a brief introduction to modelling NN for time series prediction; a general discussion is given in [11, 12].

Forecasting time series with NN is generally based on modelling the network in analogy to a non-linear autoregressive AR(p) model [2, 13]. At a point in time t, a one-step ahead forecast $\hat{y}_{t+1}$ is computed using p=n observations $y_t, y_{t-1}, \dots, y_{t-n+1}$ from n preceding points in time $t, t\text{-}1, t\text{-}2, \dots, t\text{-}n\text{+}1$, with n denoting the number of input units of the NN. This models a time series prediction as of

$$\hat{y}_{t+1} = f\left(y_t, y_{t-1}, \dots, y_{t-n+1}\right) .$$

The architecture of a feed-forward Multilayer Perceptron (MLP), a well researched NN paradigm, of arbitrary topology is displayed in figure 1.

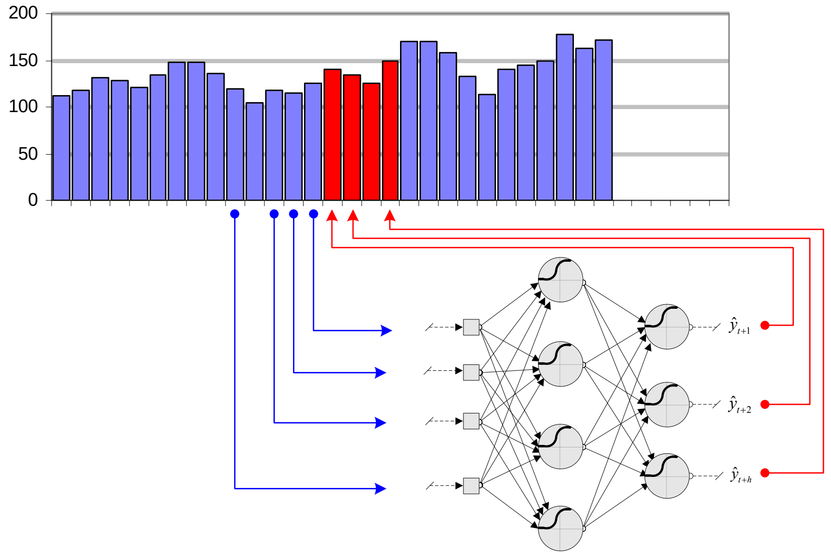


Fig. 1. Autoregressive MLP application to time series forecasting with a MLP of arbitrary topology, using n input neurons for observations in $t, t\text{-}1, t\text{-}2, \dots, t\text{-}n\text{-}1$, m hidden units, h output units for time periods $t\text{+}1, t\text{+}2, \dots, t\text{+}h$ and a two layers of trainable weights. The bias is displayed within the units.

Data is presented to the MLP as a sliding window over the time series observations. The task of the MLP is to model the underlying generator of the data during training, so that a valid forecast is made when the trained NN is subsequently presented with a new input vector value.

The network paradigm of MLP offers extensive degrees of freedom in modelling for prediction tasks. Structuring the degrees of freedom, each expert must decide upon the selection and sampling of datasets, the degrees of data preprocessing, the static architectural properties, the signal processing within nodes and the learning algorithm in order to achieve the design goal, characterized through the objective function or error function. For a detailed discussion of these issues and the ability of NN to forecast univariate time series, the reader is referred to [2].

3 Experiments and Results

3.1 Description of the Artificial Time Series

We evaluate a set of five artificial time series of monthly retail sales motivated from Pegel's original classification, later extended by Gardner to incorporate degressive trends. Time series are composed of regular patterns of different forms of linear, progressive, degressive or regressive trends T, additively or multiplicatively combined with seasonality S, a constant level L and residual noise E. In addition, empirical time series are impacted by irregular patterns such as level shifts and pulses, which are disregarded. To evaluate the ability of different computational intelligence methods we create a set of benchmark time series for the most common regular time series patterns: linear trend and different forms of seasonality. Consequently, we create individual time series patterns and combine them accordingly, overlaying each with additive noise.

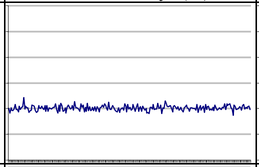
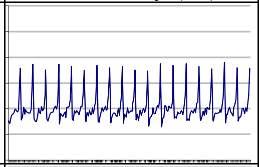
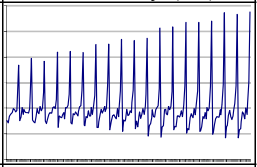
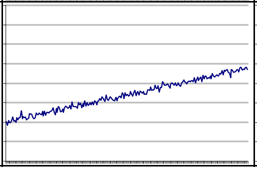
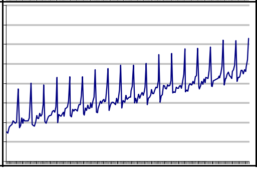
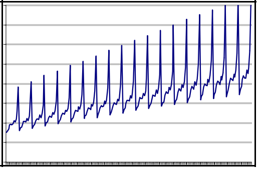
	No Seasonality (E)	Additive Seasonality (S_A)	Multiplicative Seasonality (S_M)
No Trend (L)			
Linear Trend (T_L)			

Fig. 2. Basic time series patterns of artificial time according to the Pegels- and Gardner-classification, combining Level, Trend and Seasonality with a medium additive noise level.

In contrast to Pegel's classification, a time series with multiplicative seasonality $L+S_M+E$ cannot display an increasing seasonality in the absence of level changes, it equals the pattern of additive seasonality and was consequently omitted from further analysis. Consequently, we create a set of five time series including a stationary time series $L+E$ (E), seasonality without trend $L+S_A+E$ (S_A), linear trend $L+T_L+E$ (T_L), linear trend with additive seasonality $L+T_L+S_A+E$ (T_LS_A) and linear trend with multiplicative seasonality depending on the level of the time series $L+T_L*S_M+E$

(T_LS_M). The residual error term follows a Gaussian distribution $N(0,\sigma^2)$ applying a medium level of noise $\sigma^2 = 25$. The original time series data was taken from the experiments of [3] and represent monthly retail sales. All time series considered an additive noise term to allow an estimation of final forecasting accuracy in relationship to the original noise level. Each time series consists of 228 observations.

3.2 Experimental Design

This research investigates whether the five patterns described above can be accurately predicted with RBF SVR, Linear SVR and NN models. For each series, we defined a lag structure including the 13 previous observations as attributes for predicting the next series value (one period ahead prediction); thus, a total of 215 data points remain to build and parameterize models. Data was sequentially divided into training, validation and test sets using 119, 48 and 48 observations respectively; training data is used to build the model, validation data for parameter selection purposes, and test data to evaluate the accuracy on a hold-out data set. All models are parameterized using only training and validation data, withholding all information in the test set (also for scaling etc.) to assure valid ex ante testing. Data was transformed only by applying linear scaling into a [-0.5, 0.5] interval to avoid saturation effects, using minimum and maximum values only from the training and validation data. No other preprocessing procedures such as deseasonalization or detrending were carried out.

As mentioned in section 2.1., SVR models require setting of two parameters: *C* and ε. In addition, one needs to select an appropriate kernel function to carry out the transformation to a higher dimensional feature space. The RBF kernel function, which is the kernel function most widely utilized for regression (see e.g. [6, 14, 15]), requires the definition of an additional parameter σ. Our heuristic approach for RBF SVR parameter selection can be summarized as follows:

- First, we determine starting values for the C and ε parameters on each time series by using the empirical rules proposed by Cherkassky and Ma [14], leading to E {C=0.67538; ε=0.020373}, S_A {C=0.86224; ε=0.0056657}, T_L {C=0.70709; ε=0.0043011}, T_LS_A {C=0.74641; ε=0.0064901} and T_LS_M {C=0.76968; ε=0.0064652}.
- Second, we search for ‘good’ values of the RBF kernel parameter σ using the predetermined parameters C and ε, and evaluate 45 different alternatives for σ={0.001; 0.01; 0.03; 0.05; 0.08; 0.1; 0.3; 0.5; 0.8; 1; 1.3; 1.5; 1.8; 2; 2.3; 2.5; 2.8; 3; 3.3; 3.5; 3.8; 4; 4.3; 4.5; 4.8; 5; 5.3; 5.5; 5.8; 6; 7; 8; 9; 10; 15; 20; 25; 50; 80; 100; 200; 300; 400; 500; 1000}. The value of σ which generates the model with the lowest mean absolute error (MAE) in the validation set is defined as the base parameter for the kernel function. As result, we now have heuristic starting values for the three parameters of the SVR model, C’, ε’ and σ’.
- Third, we define a grid around base parameters C’, ε’ and σ’, and retain the best combination of parameters to be the final values used in the SVR model. In our experiments, we tried five different values for each parameter C, ε, σ (factors

0.5, 0.75, 1, 1.25 and 1.5 over the initial values), thus creating a grid of 125 possible parameter settings. The parameter candidate of the grid is selected by using the lowest MAE on the validation set as before.

The scheme for Linear SVR is very similar, but without considering parameter σ. Thus, second step for base parameter σ is not carried out, and the third step involves only 25 different combinations for C and ε. (for additional details see [10]).

For NN models, we used the backpropagation algorithm to train multiple candidates of multilayer perceptron (MLP) networks. The network topology was obtained using a grid search of different hidden nodes {0, 2,…, 20} and activation functions {sigmoid; tanh} with fixed number of input and output nodes, selecting the architecture with the lowest MAE on the validation set. The final model was initialized 20 times using an (13-8-1) architecture comprised of 13 input nodes, 8 hidden nodes and a single output node for *t+1* predictions, applying a sigmoid transfer function between the input and hidden layers, and a linear function between hidden and output layers. As for SVR models, we selected the network with the lowest validation mean absolute error (MAE) to calculate the test error results.

3.3 Experimental Results and Discussion

To evaluate our models we used the root mean squared error (RMSE), mean absolute error (MAE), and mean absolute percentage error (MAPE). Test set errors obtained using SVR and MLP models for each one of the five series analyzed in this paper are shown in Table 1. As can be seen from Table 1, RBF SVR has the best performance (denoted in bold) on a level time series superimposed with white noise (E) and additive seasonality (S_A) patterns across all error measures. Linear SVR is the best method for predicting linear trend (T_L) and linear trend with multiplicative seasonality patterns (T_LS_M), while MLPs provide best results for linear trends with additive seasonality (T_LS_A) pattern.

Table 1. Forecasting accuracy on the test set for RBF and linear SVR models and MLP

Series	RBF SVR			Linear SVR			MLP		
	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE
Series E	**4.670**	**3.776**	1.387	5.036	4.108	1.496	4.851	3.946	**1.264**
Series S_A	**4.746**	**3.739**	**0.039**	6.961	5.637	0.058	5.787	4.766	0.048
Series T_L	11.501	10.408	0.046	**5.876**	**4.811**	**0.021**	6.058	4.966	0.022
Series T_LS_A	21.267	17.678	0.075	7.915	6.280	0.028	**7.083**	**5.878**	**0.027**
Series T_LS_M	14.758	10.842	0.043	7.927	**6.305**	**0.029**	**7.673**	6.454	0.030
Sum	56.942	46.443	1.590	33.715	27.141	1.632	**31.452**	**26.010**	**1.391**

Since we evaluated artificially constructed time series we can estimate the part of the forecasting errors caused by the artificially created noise, which due to its random nature cannot be forecasted. This permits an analysis to what extent each method was capable of separating noise from structure of varying complexity on the unbiased error measure of MAE. In applying the true mean of the Gaussian residuals as an optimal forecast, we estimate a MAE of 3.801 as a lower bound forecast error for all time series on the test set. It becomes apparent, that RBF SVR exceeds even a 'perfect' forecast for series E and S_A, which can be attributed to the randomness of the data inherent in all ex ante evaluations of forecasting experiments. In contrast, RBF SVR significantly underperform on trended time series patterns, indicating inadequacies of the chosen kernel function. On the contrary, linear SVR shows a more robust prediction of all time series patterns. As the forecasts deviates only slightly from the lower bound in comparison to the level of the time series, as would be reflected in the MAPE, SVR with linear kernel functions may be considered a robust method in forecasting arbitrary time series patterns without preprocessing. Similarly, MLPs forecast all time series patterns robustly and without preprocessing with a comparative high accuracy close to linear SVR and the lower bound.

In summarizing over all time series, applying an equal weight to each of the time series patterns, MLPs robustly outperform RBF SVR on all three error measures of MAE, MAPE and RMSE, whereas MLPs also moderately outperform linear SVR. This indicates that while particular kernel functions enable the SVR to outperform alternative parameterizations, MLPs or linear SVR may prove a more robust alternative in using a single method to forecast a set of time series of different patterns. In addition to these distance based error measures, we evaluate the relative performance by ranking each method by the individual error measure, provided in Table 2.

Table 2. Forecasting accuracy measured by ranks of methods for each error measure

	Rank by RMSE			Rank by MAE			Rank by MAPE		
	SVR RBF	SVR linear	MLP	SVR RBF	SVR linear	MLP	SVR RBF	SVR linear	MLP
Series E	**1**	3	2	**1**	3	2	2	3	**1**
Series S_A	**1**	3	2	**1**	3	2	**1**	3	2
Series T_L	3	**1**	2	3	**1**	2	3	**1**	2
Series T_LS_A	3	2	**1**	3	2	**1**	3	2	**1**
Series T_LS_M	3	2	**1**	3	**1**	2	3	**1**	2
Sum of Ranks	11	11	**8**	11	10	**9**	12	10	**8**

The findings by ranked error measures confirm little differences between linear SVR and MLPs, with MLPs providing the best results for the limited test design provided across all error measures. SVR with RBF kernel, the most frequently used implementation in time series prediction with SVR to date, performs significantly worse than the other methods.

As must be expected, different error measures identify different 'best' methods. In particular, RMSE and MAPE are considered to be biased error measures. To limit biases in the absence of a true objective function which could motivate the use of a particular error measure, we assume equal weight to each error and focus our conclusions on the MAE. To confirm the results of model accuracy from a statistical point of view, we performed a paired-samples t test on the absolute values of the residuals over the test set data points. Results obtained show that differences between model errors are statistically significant when comparing RBF SVR to Linear SVR (t=7.337; df=239; p<0.001), and RBF SVR to NN (t=6.999; df=239; p<0.001), although not when comparing NN to Linear SVR (t=-0.989; df=239; p=0.324). This indicates that no significant difference in forecasting accuracy between the methods of linear SVR and MLP may be derived from these experiments. Consequently, we need to extend this evaluation on additional time series and variations of MLPs. Results suggest that RBF SVR can predict seasonal patterns but no trends, while linear SVR and NN seem to be able to extrapolate trend as well as seasonal patterns accurately and without preprocessing. By examining the residuals of the models, it can be observed that RBF SVR systematically underestimate hold-out sample observations for trended series, which corresponds to saturation effects.

4 Conclusion

We have examined the ability of RBF SVR, linear SVR and MLP for predicting five basic artificial time series patterns: stationary, seasonality, linear trends, linear trend with additive seasonality, and linear trend with multiplicative seasonality. Results obtained using multiple error measures show that while RBF SVR outperform other methods on non-trended data, they do not provide robust results across all patterns. For time series with trend components, linear SVR and MLP significantly outperform RBF SVR models, which severely underestimate out-of-sample observations, consistently lagging behind upward trends. RBF SVR errors have shown to be statistically significantly higher than linear SVR and NN errors. MLP demonstrate robust performance, providing the highest overall forecasting accuracy in across time series and different statistical error measures and rank based metrics.

Our results confirm previous findings by Guajardo et. al [10], demonstrating accurate forecasts of seasonal time series without trends using RBF SVR, even outperforming established statistical methods such as ARIMAX. Also, they confirm results by Hansen et. al [9], who accurately predicted both linear and nonlinear trends using SVR, outperforming other methods such as ARIMA on several patterns. We assume that Hansen et al. also used linear kernels, as they did not fully document the kernel functions applied. A preliminary hypothesis for our poor results obtained with RBF SVR in extrapolating trend patterns lies in the linear nature of this trend. Previous publications report similar problems of closely related RBF-neural networks in predicting trends and instationary time series. While SVR with linear kernel functions and MLP with linear activation functions in the output units may be particularly suited to extrapolate linear trends, we did not conduct experiments as to their ability to extrapolate non-linear trends.

These issues will be evaluated in an extended set of experiments currently under investigation by the authors, increasing the number of time series patterns and considering additional kinds of trend patterns, also evaluating results against established statistical forecasting methods as benchmarks. Additionally, we will evaluate the influence of preprocessing procedures such as deseasonalization to evaluate alternative perspectives on the problem of extrapolating time series patterns.

Acknowledgement: This work has been supported in part by the Millennium Nucleus "Complex Engineering Systems" (www.sistemasdeingenieria.cl).

References

1. K. P. Liao and R. Fildes, The accuracy of a procedural approach to specifying feedforward neural networks for forecasting, Computers & Operations Research 32 (8) (2005) 2151-2169.
2. G.P. Zhang, B.E. Patuwo, and M.Y. Hu, Forecasting with artificial neural networks: The state of the art, *International Journal of Forecasting*, 1, **14,** 35-62 (1998)
3. G.P. Zhang and M. Qi, Neural network forecasting for seasonal and trend time series, *European Journal of Operational Research* **160,** 501-514 (2005).
4. L. J. Tashman, Out-of-sample tests of forecasting accuracy: an analysis and review, *International Journal of Forecasting* 16 (4) (2000) 437-450.
5. V.Vapnik, *The nature of statistical learning theory* (Springer, New York, 1995).
6. A.J. Smola and B. Schölkopf, A Tutorial on Support Vector Regression, NeuroCOLT Technical Report NC-TR-98-030, 1998 (Royal Holloway College, University of London, UK).
7. K. Müller, A. Smola, G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik, in: Advances in Kernel Methods: Support Vector Learning/ Using Support Vector Machines for Time Series Prediction, edited by B. Schölkopf, J. Burges, and A. Smola (MIT Press, 1999) , pp. 243-254.
8. V.Vapnik, *Statistical Learning Theory* (John Wiley and Sons, New York, 1998).
9. J.V. Hansen, J.B. McDonald, and R.D. Nelson, Some evidence on forecasting time-series with Support Vector Machines, *Journal of the Operational Research Society*, 1, 1-11, 2005.
10. J. Guajardo, J. Miranda, and R. Weber, A Hybrid Forecasting Methodology using Feature Selection and Support Vector Regression, 5th International Conference on Hybrid Intelligent Systems HIS 2005 (Rio de Janeiro, Brazil, 2005), pp. 341-346.
11. C. M. Bishop, Neural networks for pattern recognition. Clarendon Press; Oxford University Press, Oxford, 1995.
12. S. Haykin, Neural networks: a comprehensive foundation, 2nd ed. Prentice Hall, Upper Saddle River, N.J., 1999.
13. A. Lapedes, R. Farber, and Los Alamos National Laboratory, Nonlinear signal processing using neural networks: prediction and system modelling, Los Alamos National Laboratory, Los Alamos, N.M. LA-UR-87-2662, 1987.
14. V. Cherkassky and Y. Ma, Practical selection of SVM parameters and noise estimation for SVM regression, *Neural Networks* **17**(1), 113-126 (2004).
15. D. Mattera and S. Haykin, in: Advances in Kernel Methods: Support Vector Learning/ Support Vector Machines for Dynamic Reconstruction of a Chaotic System, edited by B. Schölkopf, J. Burges and A. Smola (MIT Press, 1999), pp. 211-242.

Neural Plasma

Daniel Berrar and Werner Dubitzky
Systems Biology Research Group, School of Biomedical Sciences,
University of Ulster, Northern Ireland
{dp.berrar, w.dubitzky@ulster.ac.uk,
WWW home page: http://research.bioinformatics.ulster.ac.uk/~dberrar/

Abstract. This paper presents a novel type of artificial neural network, called *neural plasma*, which is tailored for classification tasks involving few observations with a large number of variables. Neural plasma learns to adapt its classification confidence by generating artificial training data as a function of its confidence in previous decisions. In contrast to multilayer perceptrons and similar techniques, which are inspired by topological and operational aspects of biological neural networks, neural plasma is motivated by aspects of high-level behavior and reasoning in the presence of uncertainty. The basic principles of the proposed model apply to other supervised learning algorithms that provide explicit classification confidence values. The empirical evaluation of this new technique is based on benchmarking experiments involving data sets from biotechnology that are characterized by the small-*n*-large-*p* problem. The presented study exposes a comprehensive methodology and is seen as a first step in exploring different aspects of this methodology.

1 Introduction

Recent experimentation techniques in biology are probing deeper and deeper into biological phenomena. These so-called high-throughput technologies (measuring thousands of systems parameters in a single experiment) are heralding a paradigm shift (a) from traditional hypothesis-driven to data-driven research in molecular biology and (b) to a systems or systemic, as opposed to reductionistic, approach, attempting to model entire systems in order to understand study their holistic properties and dynamic properties. However, the noisy and high-dimensional data sets generated by these methods present considerable analytical and computational challenges. This study addresses this problem by analyzing high-dimensional gene expression data obtained from DNA microarray experiments investigating cancer. DNA microarrays are a high-throughput technology facilitating the simultaneous measurement of activity and interaction of thousands of genes in a single experiment [1]. This technology has led to the discovery of new biomarkers for disease diagnosis and prognosis, promoted the development of novel drugs for cancer therapy, and has provided new insights into the genesis and progression of multiple types of cancer.

Because of its importance to diagnostic and prognostic analysis, automated classification has attracted considerable interest in the context of microarray data analysis. For example, cancer types could be successfully classified based on the specific expression signatures [2,3,4]. However, microarray data classification presents substantially new challenges. First, microarray data exhibit high levels of noise due to various sources of systematic and random errors, including missing values. Second, microarray data are beset by a double 'curse' consisting of *high dimensionality* and *data set sparsity* [5]. Such data usually contain few (in the order of 10^2) observations (samples) and many (in the order of 10^4) parameters (genes). Many genes contain redundant or irrelevant information. Further, many data sets contain a relatively high number of classes but few cases per class. The curse of dimensionality in microarray data is commonly addressed by feature selection and dimension reduction techniques. However, the number of remaining genes that are significantly differently expressed in different classes can still be immense compared to the relatively small number of cases per class. This poses severe problems to an inductive learning of a classification function from such data. A desirable solution to the dimensionality problem would be to increase the number of cases. However, this is often not feasible because of (*i*) the limited number of available patients or specimens, and (*ii*) the relatively high costs of microarray experiments in terms of money and time.

Confidence values convey information about the class membership of the cases and are used in model fusion approaches such as bagging and boosting. Bagging involves a repeated random sampling (with replacement) of the original training set to generate *m* bootstrapped data sets. In noisy bagging, the bootstrapped data sets are disturbed by random noise and have shown to improve the generalization ability of ensembles of neural networks [6]. Adaptive boosting (Adaboost) creates several different models and combines their predictions using a weighted voting scheme (e.g., majority voting). Here, *k* different training set replicas are sampled adaptively (with non-uniform sampling probabilities and replacement) from the learning set. The predictions of the combined model are generated using a weighted voting scheme. The adaptive sampling procedures increase the probability of a hard-to-classify case to be sampled based on the performance of the classifier in the previous iteration. Cases that are most often misclassified are assigned an increased probability for being sampled in the next round.

The study presented in this paper is necessarily and intentionally comprehensive as it attempts to expose and discuss various elements of a full methodology rather than only a single method. As a consequence, not all parts of the presented methodology are discussed and evaluated in detail. It is our plan to explore and investigate different aspects of this comprehensive methodology in more detail in the future. This paper focuses on how the confidence values computed in the learning phase can be used for optimization of a single classifier in the context of the small-*n*-large-*p* problem. We present a model that calibrates its confidence in classification processes. In the learning phase, the model generates artificial training data as a function of its confidence in previous decisions and uses these data for calibrating its confidence in subsequent classifications. These artificial data play a pivotal role in determining the model's form or structure and performance, and have led to the model's name. (The Greek word *plasma* means 'to be formed' or 'molded'.)

2 Confidence in Classification

In practical applications without precise definition of costs for false positives and false negative classifications, exact characterization of the reward and penalty associated with a given prediction is not possible. Information-theoretic approaches typically translate a classifier's confidence into reward and penalty scores. This is based on the following rationale: *Misclassification with high confidence is more severe than misclassification with low confidence*. Let C be the real class associated with case $\mathbf{x}$ and $\hat{p}(C|\mathbf{x})$ be the model's confidence that the case belongs to C. Then, a *reward-penalty function* $R(\hat{p})$ can be defined as follows [7].

$$R(\hat{p}) = 1 + \log_2 \hat{p}(C \mid \mathbf{x}) \tag{1}$$

Key properties of this function are that it is not symmetrical with respect to rewards and penalties, and that the discrepancy becomes larger for higher confidence values. Extreme confidences that entail a misclassification, $\hat{p}(\neg C|\mathbf{x} \in C) = 1$, lead to a penalty of $-\infty$, whereas the maximum reward for a correct classification is only 1. To avoid extreme confidences, we force the minimum and maximum confidences towards $\hat{p}_{\min} = 0.5/(N + 1)$ and $\hat{p}_{\max} = (N + 0.5)/(N + 1)$, where N is the number of cases in the learning set [8]. For example, if a training set contains $n = 100$ cases, then the maximum confidence for a single classification is $\hat{p}_{\max} = 0.995$.

Korb *et al.* showed that if a model predicts a class with probability $\hat{p}$, and the real class will actually occur with frequency $f = \hat{p}$, then this model can be expected to obtain the highest reward [7]. Such a model is called *perfectly calibrated*. Miscalibration measures how much the probability estimates deviate from the *frequency of truth of events* [7]. Korb *et al.* proposed to measure a model's miscalibration by partitioning the range of a model's confidence values into cells, so that each cell contains at least ten confidence values and as few as possible above ten [7]. Then, the frequency of truth within the cells is compared with the confidence values that they contain. The miscalibration is defined as follows:

$$miscalibration = \sqrt{\sum_{i=1..n} \sum_{j=1..m} \frac{\left(\sum_k f_{ik} m^{-1} - \hat{p}_{ij}\right)^2}{m-1}} \tag{2}$$

where n is the number of partitions of the range of confidence values; m is the number of confidence values in the ith cell; k is the index of confidence values in the ith cell; f_{ik} is 1 if the kth prediction in the ith cell is correct, 0 otherwise; and $\hat{p}_{ij}$ is the jth confidence value in the ith cell. Korb's measure of miscalibration can be used to derive a measure to quantify the model's *timidity* by considering only those confidence values $\hat{p}_{ij}$ that lead to a correct classification.

3 Jittering

Jittered data (jitter) refers to data that is deliberately corrupted by artificial noise. Several studies have demonstrated that the generalization ability of neural networks can be significantly improved by injecting jitter into the data, particularly when the size of the training set is small [9,10]. The concept of jittering has been successfully applied to tasks that are characterized by the curse of dimensionality. Van Someren *et al.* followed this strategy to model robust genetic networks from time-course gene expression data [11]. Provided that the noise amplitude is small, jittering is equivalent to Tikhonov regularization [12]. Adding jitter can lead to an increased classification error in the training phase, but to a decreased error in the test phase. Chawla *et al.* investigated classification problems that involve imbalanced classes, i.e., data sets with classification categories that are not (approximately) equally balanced [13]. They presented the method of SMOTE, an approach for over-sampling the minority class using synthetic training cases. The generation of these synthetic cases is effectively a jittering approach that improves the classification performance in the context of skewed class distributions [13]. Empirical results have shown that SMOTE performs better than over-sampling with replacement of the minority class; it also performs better than under-sampling of the majority class [13].

Consider the classification problem that involves the learning of the mapping from a vector $\mathbf{x}$ to a class label y, where $\mathbf{x}$ is a p-dimensional vector of gene expression data and y is a discrete variable (e.g., a cancer class). The jittered version of this vector is $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\varepsilon}$, and $\tilde{\mathbf{x}}$ has class label y. The noise vector $\boldsymbol{\varepsilon} = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_p)$ has a distribution of mean m_ε and standard deviation s_ε.

For cancer microarray data sets, we often observe that genes exhibit a similar expression profile in samples of the same cancer type. We propose that the magnitude of the noise level takes into account the magnitude of the actual expression levels; otherwise, the class-discriminatory effect of low-level expressed genes might vanish.

Let the i^{th} original expression profile be $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})$. The jittered version of this vector, $\tilde{\mathbf{x}}_i$, is given by Equation 3 as follows:

$$\tilde{\mathbf{x}}_i = (\tilde{x}_{i1}, \tilde{x}_{i2}, \ldots, \tilde{x}_{ip}) \text{, with } \tilde{x}_{ik} = (\beta_{ik}\alpha_{ik}\rho_{ik} + 1)x_{ik} \tag{3}$$

where β_{ik}, α_{ik}, and ρ_{ik} are random variables, and $\beta_{ik} \in \{1, 0\}$, $\alpha_{ik} \in \{-1, 1\}$, and $\rho_{ik} \in [\rho_{\min}, \rho_{\max}]$, with $\rho_{\min}, \rho_{\max} \in]0, 1[$. The values 1 and 0 are equally likely for β_{ik}, so that β_{ik} controls the number of variables (i.e., genes) to be jittered. If $\beta_{ik} = 0$, then the k^{th} component of the i^{th} jittered expression profile is identical to the k^{th} component of the i^{th} original profile. If $\beta_{ik} = 1$, then the k^{th} component of the i^{th} jittered expression profile is a jittered version of the k^{th} component of the i^{th} original profile. This noise is determined by both α_{ik} and ρ_{ik}.

4 Calibration Using Jittering

Equation 3 provides a general means for generating a jittered expression profile. When adding jittered duplicates to a data set, three questions need to be answered: (1) How many jittered cases should be added?, (2) Which cases are candidates for jittering?, and (3) Which distribution (type and parameters) of distortion noise should be chosen?

We can distinguish two situations: (*i*) all confidence values within a cell lead to a correct classification, and (*ii*) at least one confidence value leads to a misclassification. Consider the latter case first. If a cell contains a value that leads to a misclassification, then we decide that the respective training case should be jittered. If all confidence values in a cell lead to a correct classification, and if all cases were classified with confidence 1, then the contribution to the timidity component of the miscalibration would be zero, but such extreme confidences are not allowed (see above). Suppose that each probability in a cell is relatively high, for instance, each confidence is $\hat{p} = 0.95$. Then this cell's contribution to (the square root of) the timidity is $10\times(1-0.95)^2/9 = 0.003$, which may be deemed sufficiently small. If the confidence values are all relatively small, e.g., 0.70, then the cell's contribution to (the square root of) the timidity is 0.10, which can be considered rather large. The confidence values might be too small to be judged valuable. Therefore, if the contribution to timidity in a cell is greater than a small positive threshold δ, then *all* respective training cases within this cell should be jittered.

Neural plasma is based on the probabilistic neural network (PNN) [14]. Figure 1 depicts the topology of neural plasma, illustrated for two classes of three cases each and two test cases.

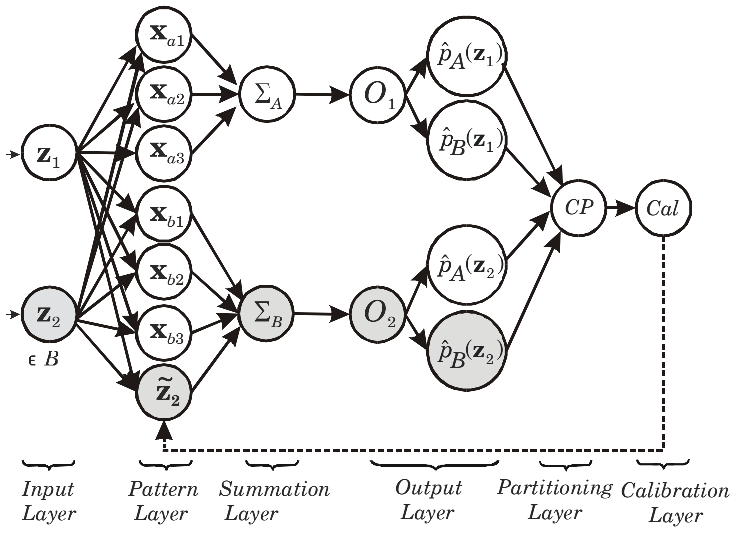


Fig 1. The topology of neural plasma.

The first part of neural plasma – input, pattern, summation, and output layer – is identical to the basic PNN. The difference consists in the *partitioning layer* and the *calibration layer*. The cell partitioning neuron *CP* receives the computed class posteriors and partitions them into cells of approximately equal size in such a way that each cell is guaranteed to contain at least ten elements and as few as possible above that number. The calibration neuron *Cal* determines the model's calibration with respect to boldness and timidity and determines which cases are candidates for jittering. Then, the calibration neuron generates jittered cases according to Equation 3 and feeds these cases back to the pattern layer. Consider the shaded parts in Figure 1. The case $\mathbf{z}_2$ is a member of class *B*. This case is assigned to one of the classes *A* or *B*, depending on which estimated class posterior is the highest. The neuron O_2 outputs these posteriors for $\mathbf{z}_2$. Suppose that $\hat{p}(B\,|\,\mathbf{z}_2)$ is the highest, i.e., leading to a correct classification, but $\hat{p}(B\,|\,\mathbf{z}_2)$ is still too small with respect to the calibration criterion. Or suppose that $\hat{p}(B\,|\,\mathbf{z}_2)$ is *not* the highest, leading to a misclassification of $\mathbf{z}_2$. In both cases, the calibration layer will generate a jittered duplicate of this case, $\tilde{\mathbf{z}}_2$, and add it to the pattern layer. We propose a *k*-fold sampling procedure with the sampling methodology as shown in Figure 2.

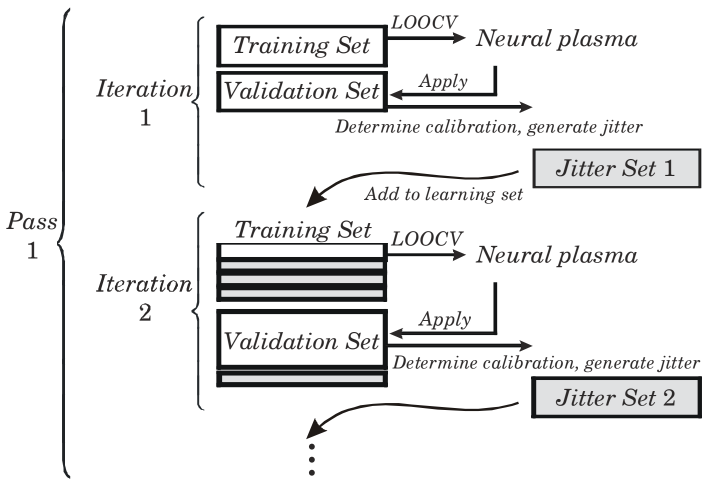


Fig. 2. One pass in the cross-validation procedure.

The learning set is randomly split in half into a training set and a validation set. Using the training set in leave-one-out cross-validation (LOOCV), the model determines the optimal kernel bandwidth. To classify the cases of the validation set, neural plasma uses that bandwidth that produces the smallest LOOCV error in the training set. Based on the performance on the validation set, the model determines its miscalibration. Based on the miscalibration, neural plasma generates jittered data. For each cell, the candidate cases for jittering are determined as follows. If at least one confidence value leads to a misclassification, then the misclassified cases are jittered. Otherwise, if all confidence values entail a correct classification, but the contribution to the timidity in a cell is greater than the threshold $\delta = 0.01$, then *all* cases in this cell are jittered.

The amount of jittered data in the i^{th} iteration represents the i^{th} *jitter set* that is added to the learning set in the $(i+1)^{\text{th}}$ iteration. Here, the learning cases are randomly mixed with the jittered cases of the previous iteration. The learning set for iteration #2 comprises now the original learning cases from iteration #1 plus the jittered cases.

In iteration #2, the model constructs the training and the validation set in such a way that they both comprise roughly the same number of cases. The jittered cases have a three times higher chance of being sampled for the training set than for the validation set. Using the training set again in LOOCV, the model optimizes the bandwidth and classifies the cases of the validation set. Again, depending on miscalibration, the model generates jittered data. The jitter set resulting from iteration #2 is mixed with the learning set and split into a training and a validation set for the next iteration. As before, jittered cases have a three times higher chance of being sampled for the training set than for the validation set. With an increasing number of iterations, both the training set and the validation set grow in size. The unequal sampling probability for jittered and original cases to be selected for the sets guarantees that the model is trained, relatively, on more artificial data and validated on more original data.

Consider Figure 2 and suppose that the depicted iterations are repeated, with the test set being the same. One *pass* encompasses n iterations with an *identical* test set. The performance – both on the test and the validation set – can vary in the iterations, because both the generation and the sampling of the jittered data are stochastic. After multiple passes have been performed, one model emerges with the smallest miscalibration. Let the number of passes be m. For example, if the model's miscalibration in the 7th iteration in the 10th pass is smaller than the miscalibration of the remaining ($m \times n - 1$) models, then this model is selected. The training and the validation set – including the jittered data – of this model are merged to one set, the *best jitter-inflated set*. The entire procedure involving m passes of n iterations represents one fold in a k-fold cross-validation. Neural plasma uses the best jitter-inflated set to classify the cases of the test set of the kth fold. For the present study, neural plasma uses $m = 20$ passes with $n = 10$ iterations each.

There exists a trade-off between too little and too much noise. In general, too few jittered cases will not have the desired regularization effect, whereas too many will increase the computational time and, more importantly, result in a 'blurring' of the data set, i.e., previously separated classes may become overlapping. The effect of the jittered cases will also depend on the characteristics of the data set at hand, for example, on the amount of measurement noise that the data set already contains. It has been suggested to determine the type of the noise distribution and the respective parameters using cross-validation procedures [9]. For example, ten-fold cross-validation can be repeated with different choices for these settings (e.g., uniform sampling of ρ_{ik} from (0, 0.05], (0.05, 15.0], etc.), and those parameters that provide for smallest mean classification error are considered optimal for the data set at hand. In the present study, we found that a uniform sampling of ρ_{ik} from (0.15, 0.25] provides for an acceptable trade-off between too little and too much noise for the three data sets investigated.

5 Materials and Methods

The experiments in this study comprise three well-studied, publicly available microarray data sets: (*i*) the NCI60 data set comprising gene expression profiles of 60 human cancer cell lines of various origins [2]. The data set contains 60 cases from nine cancer classes and 1,405 genes. The NCI60 data set is further pre-processed using principal component analysis and the first 23 'eigengenes' explaining over 75% of the total variance are selected. (*ii*) The ALL data set represents the expression profiles of 327 acute lymphoblastic leukemia samples [4]. This data comprises ten classes and the expression profiles of a total of 12,600 genes. (*iii*) The GCM data set contains 16,063 gene expression profiles of 198 specimens (190 primary tumors and eight metastatic samples) of predominantly solid tumors of 14 cancer types [15].

For the ALL and the GCM data set, feature selection was performed as follows. Based on the learning set L_i only, we determined the signal-to-noise (S2N) weight for each gene with respect to each class [16]. Then, we performed a permutation test involving a random permutation of the class labels and the re-computation of the S2N weights. This procedure was repeated 1,000 times to assess the significance of the signal-to-noise weights for the unpermuted class labels [17]. Based on the S2N weights and associated p-values, we selected the top-ranking genes per class; all other genes were discarded from further analysis. This approach was repeated ten times to generate ten pairs, each consisting of a filtered learning set L_i and a test set T_i with the corresponding genes. Information contained in the test sets was not used in any way for feature selection.

Neural plasma and boosting are related approaches, but there exist two fundamental differences: (*i*) Neural plasma is trained on jittered duplicates, and (*ii*) boosting is a multi-model approach for generating an ensemble of classifiers. Less robust or 'brittle' classifiers such as decision trees often benefit from boosting [18]. We compare neural plasma with PNN and boosted decision trees C5.0.

The performance of the models is assessed in a 10-fold repeated random sampling procedure. In short, the procedure produces $i = 1..10$ pairs of learning sets L_i and test sets T_i with original data. L_i comprises ~70% and T_i comprises ~30% of the original cases. Notice that the learning and test cases are identical for all models, and the test sets are never used for model selection or feature selection to avoid feature selection bias [19].

6 Results

Table 1 shows the 95%-confidence intervals for the prediction accuracy of the models, averaged over the ten test sets.

Table 1. 95%-confidence intervals for the true average prediction accuracy (in %).

	NCI60	*ALL*	*GCM*
Neural plasma	79.3 ± 6.4	77.9 ± 2.4	78.9 ± 3.6
PNN	76.7 ± 6.7	77.4 ± 2.4	79.6 ± 3.6
2-fold boosted C5.0	64.3 ± 7.6	68.6 ± 2.7	64.5 ± 4.3
3-fold boosted C5.0	58.5 ± 7.8	71.0 ± 2.7	63.0 ± 4.3
4-fold boosted C5.0	62.4 ± 7.6	72.6 ± 2.6	66.5 ± 4.2
5-fold boosted C5.0	62.4 ± 7.6	72.5 ± 2.6	68.0 ± 4.2

There exist only relatively small differences between neural plasma and PNN for the ALL and GCM data sets. However, on the data set comprising the smallest number of cases, NCI60, neural plasma achieved a remarkably higher accuracy than PNN. Next, we assess whether the differences in performance between neural plasma and the best-boosted trees are statistically significant. Let p_{Ai} be the observed proportion of test cases misclassified by model A and let p_{Bi} be the observed proportion of misclassified test cases by model B during the i^{th} cross-validation fold. Assume that in each fold N cases are used for learning and M cases are used for testing. The statistic for the *variance-corrected resampled paired t-test* is then given by Equation 4 as follows [20].

$$T_c = \frac{\overline{p}}{\sqrt{(k^{-1} + M / N)s^2}} \sim t_{k-1} \tag{4}$$

Empirical results show that this corrected statistic drastically improves on the standard resampled t-test with respect to Type I error [20].

The difference in performance on NCI60 between neural plasma and 2-fold boosted C5.0 is significant (*P* = 0.03). The difference in performance on GCM between neural plasma and 5-fold boosted C5.0 is significant (*P* = 0.003). However, the difference in accuracy on the ALL data set (77.9 ± 2.4% for neural plasma vs. 72.6 ± 2.6% for 4-fold boosted C5.0) is not significant (*P* = 0.06).

7 Discussion and Conclusions

Neural plasma methodology presented in this study involves several elements. Given the space limitations, not all aspects of this methodology are discussed in exhaustive detail. The neural plasma approach distinguishes itself from other neural networks with respect to two critical aspects. First, in contrast to multilayer perceptron and similar techniques, neural plasma does not attempt to mimic the topology (neurons, synapses, activation potentials, etc.) of biological neural networks. Instead, it focuses on characteristics related to intelligent behavior and reasoning, such as *timidity* (and its opposite: *boldness*). Thus, neural plasma is potentially useful for classification problems that require explicit representation of these notions in the decision process. Future work on neural plasma will concentrate on further evaluating and interpreting these concepts in the context of decision and reasoning theory.

Second, neural plasma generates artificial training cases as a function of its performance and thereby increases the learning set artificially. Within the context of high-throughput applications on biology and biotechnology, this is a novel approach to tackling the dimensionality problem in classification problems. In contrast to our approach, the SMOTE algorithm by Chawla *et al.* generates synthetic training cases only for the minority class [13].

How could the model's calibration be computed more effectively and efficiently? Neural plasma determines the miscalibration as a function of the frequency of truth in the cells. However, the partitioning into cells, each containing approximately ten elements, is based only on the empirical results by Korb *et al.* [7]. Which cross-validation procedure should be chosen, and which sampling procedure for the original and jittered cases should be adopted? The jittered data were sampled for the training set with a three times higher probability than the original cases, so that the model is trained on more artificial data and validated on more original data; however, other sampling ratios need to be investigated. What is considered a 'timid' classification is clearly context-dependent and can be controlled by the threshold δ, which was set to 0.01 in the present study. Future work will focus on the model's sensitivity to overfitting and on how these empirically determined parameters could be optimized.

In summary, we believe that the neural plasma methodology represents an interesting framework for exploring classification tasks in the context of faculties such as timidity and boldness, which are inherent factors of human reasoning. The evaluation presented in this study focuses on a limited set of criteria of a more comprehensive framework. As such, this study is seen as a first step in presenting and exploring this framework. Future work will explore different aspects in more detail.

8 References

[1] Brown, P.O. and Botstein, D. (1999) Exploring the new world of the genome with DNA microarrays. *Nat. Gen.* **21**(1): 33–37.

[2] Ross, D.T., Scherf, U., Eisen, M.B., *et al.* (2000) Systematic variation in gene expression patterns in human cancer cell lines. *Nat. Gen.* **24**(3):227−235.

[3] Alizadeh, A.A., Eisen, M.B., Davis, R.E., *et al.* (2000) Distinct types of diffuse large B-cell lymphoma identified by gene expression profiling. *Nature* **403**:503−511.

[4] Yeoh, E.J., Ross, M.E., Shurtleff, S.A., *et al.* (2002) Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling. *Cancer Cell* **1**:133−143.

[5] Somorjai, R.L., Dolenko, B., and Baumgartner, R. (2003) Class prediction and discovery using gene microarray and proteomics mass spectroscopy data: curses, caveats, cautions. *Bioinformatics* **19**(12):1484−1491.

[6] Raviv, Y., and Intrator, N. (1996) Boostrapping with noise: An effective regularization technique. *Connection Science* **8**(3–4):355–372.

[7] Korb, K.B., Hope, L.R., Hughes, M.J. (2001) The evaluation of predictive learners: some theoretical and empirical results. *Proc. 12th Europ. Conf. Machine Learning*, 276–287.

[8] Dowe, D.L., Farr, G.E., Hurst, A.J., Lentin, K.L. (1996) Information-theoretic football tipping. *Proc. 3rd Austr. Conf. Math. & Computers in Sport*, Australia, 233–241.

[9] Koistinen, P., and Holmström, L. (1992) Kernel regression and backpropagation training with noise. *Advances in Neural Inf. Proc. Sys.* **4**:1033−1039.

[10] Reed, R., Oh., S., Marks, R.J. (1992) Regularization using jittered training data. *Proc. Int. J. Conf. Neural Networks*, Baltimore MD, III147–III152.

[11] van Someren, E.P., Wessels, L.F.A., Reinders M.J.T, Backer, E. (2001) Robust genetic network modeling by adding noisy data. *Proc. IEEE Workshop on Nonlinear Signal and Image Processing*, Baltimore, Maryland.

[12] Bishop, C.M. (1994) Training with noise is equivalent to Tikhonov regularization. *Neural Computation* **7**:108−116.

[13] Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P. (2002) SMOTE: Synthetic Minority Over-sampling Technique. *J. Art. Int. Res.* **16**:321−357.

[14] Specht, D.F. (1990) Probabilistic neural networks. *Neural Networks* **3**:109−118.

[15] Ramaswamy, S., Tamayo, P., Rifkin, R., *et al.* (2001) Multiclass cancer diagnosis using tumor gene expression signatures. *Proc. Natl. Acad. Sci. USA.* **98**(26), 15149−15154.

[16] Slonim, D., Tamayo, P., Mesirov, J., *et al.* (2000) Class prediction and discovery using gene expression data. *Proc. 4th Ann. Int. Conf. Comp. Mol. Biol.*, Tokyo, Japan, 263–272.

[17] Radmacher, M.D., McShane, L.M., Simon, R. (2002) A paradigm for class prediction using gene expression profiles. *J. Comp. Bio.* **9**(3):505−511.

[18] Duda R.O., Hart P.E., Stork D.G. (2001) *Pattern Classification*. 2nd ed., John Wiley & Sons, New York, p. 461.

[19] Ambroise, C. and McLachlan, G.J. (2002) Selection bias in gene extraction on the basis of microarray gene expression data. *Proc. Natl. Acad. Sci. USA* **98**:6562−6566.

[20] Nadeau, C., and Bengio, Y. (2003) Inference for generalization error. *Machine Learning* **52**:239–281.