TESIS

presentada el dia 21 de febrero de 2013 en la

Universidad de la República, UdelaR

para obtener el título de

MAGÍSTER EN INGENIERÍA MATEMÁTICA

por

Ing. HORACIO PAGGI STRANEO

Instituto de Investigación: LPE/IMERL- Facultad de Ingeniería

Componentes universitarios:

UNIVERSIDAD DE LA REPÚBLICA

FACULTAD DE INGENIERÍA

Título de la tesis:

MODELS FOR TIME SERIES PREDICTION BASED ON NEURAL NETWORKS.

CASE STUDY: GLP SALES PREDICTION FROM ANCAP.

Defensa realizada el 21 de marzo de 2013 ante el comité de examinadores

Dr. Ing. Franco	ROBLEDO	Director de Tesis
Dr. Ing. Rafael	TERRA	Presidente
Dr. Gerardo	RUBINO	
Dr. Marco	SCAVINO	
Dr. Juan	KALEMKERIAN	

Abstract

A time series is a sequence of real values that can be considered as observations of a certain system. In this work, we are interested in time series coming from dynamical systems. Such systems can be sometimes described by a set of equations that model the underlying mechanism from where the samples come. However, in several real systems, those equations are unknown, and the only information available is a set of temporal measures, that constitute a time series. On the other hand, by practical reasons it is usually required to have a prediction, v.g. to know the (approximated) value of the series in a future instant t. The goal of this thesis is to solve one of such real-world prediction problem: given historical data related with the liquefied bottled propane gas sales, predict the future gas sales, as accurately as possible. This time series prediction problem is addressed by means of neural networks, using both (dynamic) reconstruction and prediction. The problem of to dynamically reconstruct the original system consists in building a model that captures certain characteristics of it in order to have a correspondence between the long-term behavior of the model and of the system.

The networks design process is basically guided by three ingredients. The dimensionality of the problem is explored by our first ingredient, the Takens-Mañé's theorem. By means of this theorem, the optimal dimension of the (neural) network input can be investigated. Our second ingredient is a strong theorem: neural networks with a single hidden layer are universal approximators. As the third ingredient, we faced the search of the optimal size of the hidden layer by means of genetic algorithms, used to suggest the number of hidden neurons that maximizes a target fitness function (related with prediction errors). These algorithms are also used to find the most influential networks inputs in some cases. The determination of the hidden layer size is a central (and hard) problem in the determination of the network topology.

This thesis includes a state of the art of neural networks design for time series prediction, including related topics such as dynamical systems, universal approximators, gradient-descent searches and variations, as well as meta-heuristics. The survey of the related literature is intended to be extensive, for both printed material and electronic format, in order to have a landscape of the main aspects for the state of the art in time series prediction using neural networks. The material found was sometimes extremely redundant (as in the case of the back-propagation algorithm and its improvements) and scarce in others (memory structures or estimation of the signal subspace dimension in the stochastic case). The surveyed literature includes classical research works ([27], [50], [52]) as well as more recent ones ([79] , [16] or [82]), which pretends to be another contribution of this thesis.

Special attention is given to the available software tools for neural networks design and time series processing. After a review of the available software packages, the most promising computational tools for both approaches are discussed. As a result, a whole framework based on mature software tools was set and used. In order to work with such dynamical systems, software intended specifically for the analysis and processing of time series was employed, and then chaotic series were part of our focus.

Since not all "randomness" is attributable to chaos, in order to characterize the dynamical system generating the time series, an exploration of chaotic-stochastic systems is required, as well as network models to predict a time series associated to one of them. Here we pretend to show how the knowledge of the domain, something extensively treated in the bibliography, can be someway sophisticated (such as the Lyapunov's spectrum for a series or the embedding dimension).

In order to model the dynamical system generated by the time series we used the state-space model, so the time series prediction was translated in the prediction of the next system state. This state-space model, together with the delays method (delayed coordinates) have practical importance for the development of this work, specifically, the design of the input layer in some networks (multi-layer perceptrons - MLPs) and other parameters (taps in the TFLNs). Additionally, the rest of the network components where determined in many cases through procedures traditionally used in neural networks: genetic algorithms.

The criteria of model (network) selection are discussed and a trade-off between performance and network complexity is further explored, inspired in the Rissanen's minimum description length and its estimation given by the chosen software. Regarding the employed network models, the network topologies suggested from the literature as adequate for the prediction are used (TLFNs and recurrent networks) together with MLPs (a classic of artificial neural networks) and networks committees. The effectiveness of each method is confirmed for the proposed prediction problem. Network committees, where the predictions are a naive convex combination of predictions from individual networks, are also extensively used.

The need of criteria to compare the behaviors of the model and of the real system, in the long run, for a dynamic stochastic systems, is presented and two alternatives are commented.

The obtained results proof the existence of a solution to the problem of learning of the dependence $Input \to Output$. We also conjecture that the system is dynamic-stochastic but not chaotic, because we only have a realization of the random process corresponding to the sales. As a non-chaotic system, the mean of the predictions of the sales would improve as the available data increase, although the probability of a prediction with a big error is always non-null due to the randomness present. This solution is found in a constructive and exhaustive way. The exhaustiveness can be deduced from the next five statements:

- the design of a neural network requires knowing the input and output dimension,the number of the hidden layers and of the neurons in each of them.
- the use of the Takens-Mañé's theorem allows to derive the dimension of the input data
- by theorems such as the Kolmogorov's and Cybenko's ones the use of multi-layer perceptrons with only one hidden layer is justified so several of such models were tested
- the number of neurons in the hidden layer is determined many times heuristically using genetic algorithms
- a neuron in the output gives the desired prediction

As we said, two tasks are carried out: the development of a time series prediction model and the analysis of a feasible model for the dynamic reconstruction of the system. With the best predictive model, obtained by an ensemble of two networks, an acceptable average error was obtained when the week to be predicted is not adjacent to the training set (7.04% for the week 46/2011). We believe that these results are acceptable provided the quantity of information available, and represent an additional validation that neural networks are useful for time series prediction coming from dynamical systems, no matter whether they are stochastic or not.

Finally, the results confirmed several already known facts (such as that adding noise to the inputs and outputs of the training values can improve the results; that recurrent networks trained with the back-propagation algorithm don't have the problem of vanishing gradients in short periods and that the use of committees - which can be seen as a very basic of distributed artificial intelligence - allows to improve significantly the predictions).

Resumen

Una serie temporal es una secuencia de valores reales que pueden ser considerados como observaciones de un cierto sistema. En este trabajo, estamos interesados en series temporales provenientes de sistemas dinámicos. Tales sistemas pueden ser algunas veces descriptos por un conjunto de ecuaciones que modelan el mecanismo subyacente que genera las muestras. sin embargo, en muchos sistemas reales, esas ecuaciones son desconocidas, y la única información disponible es un conjunto de medidas en el tiempo, que constituyen la serie temporal. Por otra parte, por razones prácticas es generalmente requerida una predicción, es decir, conocer el valor (aproximado) de la serie en un instante futuro t. La meta de esta tesis es resolver un problema de predicción del mundo real: dados los datos históricos relacionados con las ventas de gas propano licuado, predecir las ventas futuras, tan aproximadamente como sea posible. Este problema de predicción de series temporales es abordado por medio de redes neuronales, tanto para la reconstrucción como para la predicción. El problema de reconstruir dinámicamente el sistema original consiste en construir un modelo que capture ciertas características de él de forma de tener una correspondencia entre el comportamiento a largo plazo del modelo y del sistema.

El proceso de diseño de las redes es guiado básicamente por tres ingredientes. La dimensionalidad del problema es explorada por nuestro primer ingrediente, el teorema de Takens-Mañé. Por medio de este teorema, la dimensión óptima de la entrada de la red neuronal puede ser investigada. Nuestro segundo ingrediente es un teorema muy fuerte: las redes neuronales con una sola capa oculta son un aproximador universal. Como tercer ingrediente, encaramos la búsqueda del tamaño oculta de la capa oculta por medio de algoritmos genéticos, usados para sugerir el número de neuronas ocultas que maximizan una función objetivo (relacionada con los errores de predicción). Estos algoritmos se usan además para encontrar las entradas a la red que influyen más en la salida en algunos casos.La determinináción del tamaño de la capa oculta es un problema central (y duro) en la determinación de la topología de la red.

Esta tesis incluye un estado del arte del diseño de redes neuronales para la predicción de series temporales, incluyendo tópicos relacionados tales como sistemas dinámicos, aproximadores universales, búsquedas basadas en el gradiente y sus variaciones, así como meta-heurísticas. El relevamiento de la literatura relacionada busca ser extenso, para tanto el material impreso como para el que esta en formato electrónico, de forma de tener un panorama de los principales aspectos del estado del arte en la predicción de series temporales usando redes neuronales. El material hallado fue algunas veces extremadamente redundante (como en el caso del algoritmo de retropropagación y sus mejoras) y escaso en otros (estructuras de memoria o estimación de la dimensión del sub-espacio de señal en el caso estocástico). La literatura consultada incluye trabajos de investigación clásicos (([27], [50], [52])‘así como de los más reciente ([79] , [16] or [82]).

Se presta especial atención a las herramientas de software disponibles para el diseño de redes neuronales y el procesamiento de series temporales. Luego de una revisión de los paquetes de software disponibles, las herramientas más promisiorias para ambas tareas son discutidas. Como resultado, un entorno de trabajo completo basado en herramientas de software

maduras fue definido y usado. Para trabajar con los mencionados sistemas dinámicos, software especializado en el análisis y proceso de las series temporales fue empleado, y entonces las series caóticas fueron estudiadas.

Ya que no toda la "aleatoriedad" es atribuible al caos, para caracterizar al sistema dinámico que genera la serie temporal se requiere una exploración de los sistemas caóticos-estocásticos, así como de los modelos de red para predecir una serie temporal asociada a uno de ellos. Aquí se pretende mostrar cómo el conocimiento del dominio, algo extensamente tratado en la literatura, puede ser de alguna manera sofisticado (tal como el espectro de Lyapunov de la serie o la dimensión del sub-espacio de señal).

Para modelar el sistema dinámico generado por la serie temporal se usa el modelo de espacio de estados, por lo que la predicción de la serie temporal es traducida en la predicción del siguiente estado del sistema. Este modelo de espacio de estados, junto con el método de los delays (coordenadas demoradas) tiene importancia práctica en el desarrollo de este trabajo, específicamente, en el diseño de la capa de entrada en algunas redes (los perceptrones multicapa) y otros parámetros (los taps de las redes TLFN). adicionalmente, el resto de los componentes de la red con determinados en varios casos a través de procedimientos tradicionalmente usados en las redes neuronales: los algoritmos genéticos.

Los criterios para la selección de modelo (red) son discutidos y un balance entre performance y complejidad de la red es explorado luego, inspirado en el minimum description length de Rissanen y su estimación dada por el software elegido.

Con respecto a los modelos de red empleados, las topologóas de sugeridas en la literatura como adecuadas para la predicción son usadas (TLFNs y redes recurrentes) junto con perceptrones multicapa (un clásico de las redes neuronales) y comités de redes. La efectividad de cada método es confirmada por el problema de predicción propuesto. los comités de redes, donde las predicciones son una combinación convexa de las predicciones dadas por las redes individuales, son también usados extensamente.

La necesidad de criterios para comparar el comportamiento del modelo con el del sistema real, a largo plazo, para un sistema dinámico estocástico, es presentada y dos alternativas son comentadas.

Los resultados obtenidos prueban la existencia de una solución al problema del aprendizaje de la dependencia *Entrada* $\rightarrow$ *Salida* . Conjeturamos además que el sistema genrador de serie de las ventas es sinámico-estocástico pero no caótico, ya que solo tenemos una realización del proceso aleatorio correspondiente a las ventas. Al ser un sistema no caótico, la media de las predicciones de las ventas debería mejorar a medida que los datos disponibles aumentan, aunque la probabilidad de una predicción con un gran error es siempre no nuladebido a la aleatoriedad presente. Esta solución es encontrada en una forma constructiva y exhaustiva. La exhaustividad puede deducirse de las siguiente cinco afirmaciones:

- el diseño de una red neuronal requiere conocer la dimensión de la entrada y de la salida, el número de capas ocultas y las neuronas en cada una de ellas
- el uso del teorema de takens-Mañé permite derivar la dimensión de la entrada
- por teoremas tales como los de Kolmogorov y Cybenko el uso de perceptrones con solo una capa oculta es justificado, por lo que varios de tales modelos son probados
- el número de neuronas en la capa oculta es determinada varias veces heurísticamente a través de algoritmos genéticos
- una sola neurona de salida da la predicción deseada

Como se dijo, dos tareas son llevadas a cabo: el desarrollo de un modelo para la predicción de la serie temporal y el análisis de un modelo factible parala reconstrucción dinámica del sistema. Con el mejor modelo predictivo, obtenido por el comité de dos redes se logró obtener un error aceptable en la predicción de una semana no contigua al conjunto de entrenamiento (7.04% para la semana 46/2011). Creemos que este es un resultado aceptable dada la cantidad de información disponible y representa una validación adicional de que las redes neuronales son útiles para la predicción de series temporales provenientes de sistemas dinámicos, sin importar si son estocásticos o no.

Finalmente, los resultados experimentales confirmaron algunos hechos ya conocidos (tales como que agregar ruido a los datos de entrada y de salida de los valores de entrenamiento puede mejorar los resultados: que las redes recurrentes entrenadas con el algoritmo de retropropagación no presentan el problema del gradiente evanescente en periodos cortos y que el uso de de comités - que puede ser visto como una forma muy bñasica de inteligencia artificial distribuida - permite mejorar significativamente las predicciones).

Agradecimientos

Quisiera agradecer aquí a todos aquellos que me dieron su apoyo y aliento durante la realización de este trabajo, y muy especialmente:

- A mi tutor, Dr. Ing. Franco Robledo, por la confianza que tuvo en mí y por su aliento

- A mi compañera de ANCAP, la Ing. Adela Casero, que me proporcionó los datos usados para llevar a cabo esta tesis

- A mi Jefa, Marianela Moreno, por su apoyo a todas las actividades de capacitación del personal

.

- A la SCAPA de Ingeniería Matemática que financió a través de la Fundación Ricaldoni (FJR)/FING mi estadía en Chile

- Al Dr. Ing. Pablo Romero, por sus comentarios y sugerencias y la traducción de este trabajo

- Finalmente, mi agradecimiento profundo a mi madre, María Esther Straneo de Paggi, por su apoyo permanente.

A todos ellos les digo ¡Gracias!

Horacio Paggi

Marzo de 2013

Contents

Part I

State of the Art and Case Study

Terminology

Variables and functions

%FNN	False Nearest Neighbors percentage.
*	Arithmetic product of two scalars
$\mathbf{x} = [x_1, x_2, ...x_n]$	Denotes an n-dimensional vector (sometimes denoted by x when there is no risk of confusion).
$y = \{y_1, ...y_m\}$	Represents either an ordered list of n elements or elements from a matrix. For instance, the matrix $\mathbf{H} = \{h_{ij}\}$ contains the elements h_{ij} where i represents the row and j the column.
$\nabla E(\mathbf{w})$	Denotes the gradient of a scalar function $E = E(\mathbf{w})$, being $\nabla E = [\frac{\partial E}{\partial w_1}, ...\frac{\partial E}{\partial w_n}]$ and $\mathbf{w} = [w_1, ...w_n]$
$E(\mathbf{w})$	Error function. In some cases denoted by $J(\mathbf{w})$.
$\mathbf{H}$	Hessian matrix of the scalar function E : $\mathbf{H} = \{H_{ij} = \frac{\partial E}{\partial w_i \partial w_j}\}$
x_i	Denotes input number i of a neuron.
w_j	Weight from position j (regarding the set of weights as a vector)
w_{ji}	Weight of the connection from neuron i towards j.
$\langle X \rangle_K$	Expectation of X =X(K) among all the possible values of the random variable K.
$f(.)$	Output function (or transfer).
$f_h(.)$	Output function for the hidden layer.
$f_o(.)$	Output function for the output layer.
ρ	Learning rate.
ρ_h	Learning rate for the hidden layer.
ρ_o	Learning rate for the output layer.
Net_k	Input of neuron k, defined for the current instant as $Net_k = \sum_{i=1,n} x_i w_{ki}$
$Net_k(n)$	Analogously, but for instant n.
d_i	Desired output for neuron i, given its current inputs.
y_i	Real output for neuron i, given its current inputs.
d_E	Global dimension of the embedding, also denoted by m.
d_L	Local dimension of the embedding.
d	Optimal delay for the embedding, also denoted by τ.
$\mathbb{R}$	Denotes the field of real numbers.
S, S-1, S-2, S-3...	Denotes respectively the current week (S), previous (S-1), and earlier weeks.
V, V-1, V-2, V-3	Denotes the corresponding gas sales from current and previous weeks. They are the sales to be predicted, although in some parts V+1 is used (mainly when the network input is inspired in Takens-Mañé's theorem).
V+1	Sales from the next week, to be predicted.

%FNN	False Nearest Neighbors percentage.
*	Arithmetic product of two scalars
T, T-1, T-2, T-3	Average of maximun daily temperatures from current and previous weeks respectively.
A, A-1, A-2, A-3	Indicator of price increase from current and previous weeks.
%Error CV	Percentage error when a step is predicted, averaged over the cross-validation set.

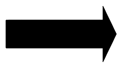
.

Acronyms

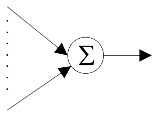
GA	Genetic Algorithm
AIC	Akaike's Information Criterion
BP	Back-propagation
BPTT	Back-propagation Through Time
BPTT(h)	Truncated BPTT
CV	Cross validation
CD	Correlation Dimension
deKf	Decoupled Extended Kalman Filter
h.n.	hidden neuron(s)
FNN	False Nearest Neighbors
IFS	Iterated Functions Systems
LMS	Least Mean Squares
LSTM	Long Short-Term Memory
MDL	Minimun Description Length
MLP	Multi Layer Perceptron
MSE	Mean Squares Error
PCA	Principal Component Analysis
RP	Recurrence Plot
RQA	Recurrence Quantification Analysis
RTRL	Real Time Recurrent Learning
SER	Signal to Error Ratio
SOM	Self Organizing Map
TDNN	Time Delayed Neural Network
TLFN	Time Lagged Feed-forward Network

Symbols

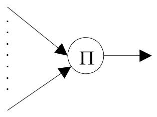
$\longrightarrow$ Connection between two neurons, or input/output flow to/from one of them.



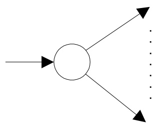
This type of arrow represents all possible connections between two neurons.



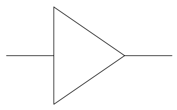
The output of a node labeled wit a capital sigma equals the arithmetic sum of its inputs.



The output of a node labeled with a capital pi equals the arithmetic product of its inputs.



The outputs of a node of this class are identical to its input.



Represents the product of their inputs by the value inside the triangle.

Chapter 1

Introduction and Problem Formulation

A time series is a set of observations x_t, each one being recorded at a specific time t. [13]. In general, those values can be regarded as measurements (observations) of a characteristic variable inside a certain system. On the other hand, one of the most common real systems are the ***dynamical systems***. These systems, as well as their associated time series, are characterized by their "dynamic effect", where *dynamic refers to the phenomena that produce time changing patterns, the characteristics of the partner at one time being interrelated with those at other times. The term [dynamic] is nearly synonymous with time-evolution or pattern of change. It refers to the unfolding events in a continuing evolutionary process*[44]. In other words, in a dynamical system the current status conditions the future, and several state variables can be interrelated and interacting. This class of systems are studied in all branches of science, from the solar system to the evolution of the stocks prices. These systems evolve with time in accordance with a set of (usually non-linear) rules which, if known, allow us to predict the complete evolution of the system. In this case, we will study the time series associated with the liquefied propane gas sales in bottles, measured in liters, trying to use some properties of the underlying dynamical system.

To summarize, in this thesis we address the ***prediction*** of a time series: a good prediction will reduce, for instance, the costs associated with stock maintenance. A prediction is characterized by a horizon (temporary scope in the future) relatively short (e.g. two steps in the future). Collaterally, we will be interested in the ***dynamic reconstruction*** (or dynamic modeling). Indeed, dynamic modeling and prediction of time series are two areas which share several aspects in common but differ in the way the obtained solution is evaluated. In the classical theory of time series prediction [59], the goal is to get the correct (or the most accurate) value in the near future (for instance, by using well known modeling techniques such as ARIMA or ARMA [13]. The goal of a dynamic reconstruction is to capture the dynamics (behavior in the long-term future) of the system by means of an appropriate model. For example, by means of dynamic reconstruction a simulation benchmark for the gas sales could be obtained [30]. In the present work both approaches are considered and several models are discussed in order to find adequate models for our case study.

Generally, when there exists enough information about the laws that govern the system (mathematical or physical in the case of natural systems), an analytical approach for prediction may be adequate, and the reconstruction is derived from the equations that describe the underlying mechanism responsible from the time series generation. In a deterministic system, such values are given by the equations that produce the series ([10], [30], [69]).

However, in several real problems, the available information is scarce as to try an analytical approach. Indeed, even when the system has several variables that govern its behavior, usually only one of them is available as a time series. In the absence of enough information to derive mathematical equations, it is mandatory to employ non-analytic approaches. A clear example is the neural networks approach, where a network is trained in such a way that its output respects a measure of similarity with the observed values. Therefore, a neural model is built in order to approximate ideal equations, instead of trying an explicit inference of the equations that describe the underlying dynamic of the system. Alternatively, in the case of a stochastic dynamical system, the observed values are samples of a possible result of the random process that produces the observed variable, and the issue is to make the neural network a statistical model of that process [1]. Neural networks are attractive to model non-linear dynamical systems, given that they are intrinsically non-linear (mainly because of the non-linear output functions of the neurons) and they approximate functions with a solid foundational background (see for example 2.10 on Kolmogorov's and Cybenko's theorems). Additionally, feedforward networks represent a feasible way to model the conditional probability density function of a certain desired output $\mathbf{d}$ given a known input $\mathbf{x}$ ([10]).

Regarding the previous elements as our point of departure, we can summarize the main goals of this work as follows:

1. reveal the state of the art of neural networks for the prediction of time series.
2. build a neural model in order to predict propane gas sales, from which only a few variables are known (such as the daily temperatures and volume of gas sales), using the most adequate computational tools. The model(s) should allow us to predict the gas sales at least in a period of one week. This modeling related to the research of the theoretical properties that make one model preferable over another one.

This thesis is structured as follows:

- Chapter 2 contains the state of the art in the application of neural networks to the modeling of dynamical systems. In order to make this document self-contained, a thorough description of the Back-Propagation (BP) algorithm is presented, including some variations to look for the global optimum. Several network topologies adequate for this are discussed, and the most promising software tools are also investigated and described.
- Chapter 3 presents the application of neural networks to a real life problem (our case study), the prediction of weekly gas sales. The experimental setup is fully described, and the main results are summarized. This chapter is connected with Chapter 5, which contains further numerical results and tables with the used data.
- Chapter 4 includes a discussion of the experimental setup and remarks the main conclusions and possible future work.
- Finally, this document is closed with bibliographical references and an appendix containing characteristics of several software packages for neural modeling.

Chapter 2

State of the Art

2.1 Basic Concepts of Neural Networks for the Time Series Prediction

This chapter revisits the foundational theory on which the present work is based. The ***back-propagation*** algorithm is first described in its static case (inputs and outputs independent of time), linking it with the ***Least Mean Squares*** (LMS) rule and further discussing several improvements, such as second order searches. Networks committees and combinations of predictors are then covered using concepts coming from ***machine learning***. The treatment is extensively related with concepts of the theory of information systems such as entropy and mutual information. Dynamical networks are then discussed, which represent a kind of networks that naturally model dynamical systems. Additionally, several criteria for the discrimination of ***outliers*** and the concept of ***minimum description length*** are explained. Finally, some guidelines on the network design and its training are included.

2.1.1 The Back-Propagation Algorithm

In this subsection we will describe the basic algorithm for the training of all the networks employed in this work, as well as its theoretical support and its major variants.

Learning in neural networks can be divided into supervised and unsupervised. The supervised technique is based on a direct comparison between the real network output and the desired output. Its custom formulation is the minimization of a difference function (error function) between the real and the desired outputs, such as the mean square error, and where the variables are the weights of the neural network. In order to minimize the error function several optimization algorithms can be applied, such as gradient descend (of which back-propagation is an instance), simulated annealing and genetic algorithms, to name a few. On the other hand, unsupervised techniques are based on the correlations between the inputs; here is not information available for the desired output and the process produces a change of weights of the neural network [72].

With the term retro-propagation we basically understand a learning algorithm used to train multilayer networks[1], designed by David Rumelhart, G.E. Hinton and R. J. Williams in 1986, from the pioneering works of Parker (1985) and Werbos (1974) [10]. This supervised

[1]The model of a multilayer feedforward network trained with this learning algorithm, is usually called "back-propagation network" ([20] y [33])

learning algorithm can be regarded as a generalization of the delta rule. Currently, BP represents one of the most popular algorithms (models), mainly because its simplicity of implementation, its speed (it is faster than many other method) and its wide acceptance and success in practice. The two learning rules in which this method is based are the least mean squares (LMS) and the delta rule.

2.1.1.1 The Delta Rule

This rule is an extension of the LMS rule for neural networks with non-linear, continous output functions. This is a local rule when we consider the pattern and its weights: it needs information about the specific pattern, the corresponding neuron error and its input, as it is detailed as follows.

LMS rule seen as a stochastic process Consider the following neuron to be trained, whose output function is linear (so we call it a ***linear neuron***):

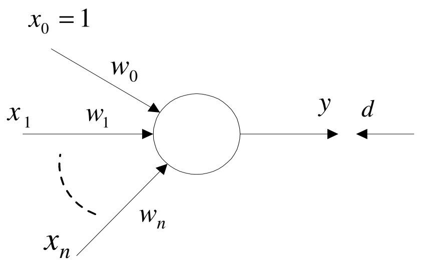


Figure 2.1

where $x_i, i = 1, ...n$ represent the inputs, w_i the weights, w_0 the activation threshold, and **d** the desired output value (assumed known) and $y = y(\mathbf{w})$ the output value the output value of the neuron:

$$y = \sum_{i=0}^{n} x_i w_i = \mathbf{x}^{\mathrm{T}}\mathbf{w}$$

with $\mathbf{x} = [x_0 \cdots x_n] \quad \mathbf{w} = [w_0 \cdots w_n]$

We will try to minimize the mean square error (with a fixed artificial input $x_0 = 1$ for the sake of notation simplicity) defined as:

$$E(\mathbf{w}) = \frac{1}{2}\left\langle (\mathbf{x}^T\mathbf{w} - d)^2 \right\rangle_{\mathrm{x}}$$

where $\langle \bullet \rangle_{\mathrm{x}}$ represents the mean among all feasible training vectors $\mathbf{x}$. In order to find the minimum for *E,* a possible technique is a gradient descend. The gradient vector is:

$$\nabla E(\mathbf{w}) = \left\langle (\mathbf{x}^{\mathrm{T}}\mathbf{w} - d)\mathbf{x} \right\rangle_{\mathrm{x}}$$

We can approximate the gradient vector $\nabla E(\mathbf{w})$ with its instantaneous value:

$$\nabla_k E = [(\mathbf{x}_k)^{\mathrm{T}}\mathbf{w}_{\mathrm{k}} - d_k]\mathbf{x}_{\mathrm{k}}$$

where the sub-index k meaning “corresponding to the presentation of the $k-$th pattern in the input”, arriving to the following algorithm:

$$\begin{cases} w_0 \quad \textit{arbitrary} \\ w_{k+1} = w_k + \rho_k[d_k - (x_k)^{\mathrm{T}}w_k]x_k \end{cases}$$

being ρ_k a constant whose properties are detailed below.

The previous iterative process is known as a ***stochastic approximation process*** (or of Kiefer-Wolfowitz or of Robbins-Monro, see [29]).

Convergence Consider the autocorrelation matrix $\mathbf{C} = \left\langle \mathbf{x}\mathbf{x}^T \right\rangle_X$ where $\langle ... \rangle$ is understood as the matrix with the corresponding expected values of its entries. In this case we have $C = \left\{ c_{ij} = \langle x_i x_j \rangle_X \right\}$ where X is a random variable representing all possible patterns for the input $\mathbf{x}$. It can be proved [29] that if $\mathbf{C}$ is non-singular and the learning rates ρ_k meet the following requirements

$$\begin{cases} \rho \geq 0 \\ \lim\limits_{m \to \infty} \sum_{k=1}^{m} \rho_k = +\infty \\ \lim\limits_{m \to \infty} \sum_{k=1}^{m} (\rho_k)^2 = \lambda \quad (finite) \end{cases}$$

being m the number of training patterns, then the sequence $\{\mathbf{w}_k\}$ asymptotically converges in mean square to the vector $\mathbf{w}^*$ that produces the minimum value for the error function E:

$$\lim_{k \to \infty} \left\langle \|\mathbf{w}_k - \mathbf{w}^*\|^2 \right\rangle_X = 0$$

The Delta rule In the case where the output $\boldsymbol{y}$ is given by $y = f\left(\sum_{i=0}^{n} x_i w_i\right)$ for a non-linear differentiable function f, the obtained training rule is known as ***delta rule***, because the issue is to minimize a "delta" (difference) between the obtained and desired values. Analogously than before, we try to minimize the expected value of the difference $J(w) = \frac{1}{2} \left\langle (y(\mathbf{w}, \mathbf{x}) - d)^2 \right\rangle_X$, whose gradient is approximated via its instantaneous gradient:

$$\begin{array}{l} \nabla_k J(\mathbf{w}) = -(d_k - y_k) f'(Net_k) x_k \\ Net_k = \sum_{i=0}^{n} (x_i)_k w_i \\ f'(Net_k) = \left. \frac{\partial f}{\partial Net} \right|_{Net_k} \end{array}$$

where $(x_i)_k$ represents the i-th element (component) for the k-th pattern.

As a consequence, the delta rule can be formulated as follows:

$$\begin{cases} \mathbf{w}_1 \quad arbitrary \\ \mathbf{w}_{k+1} = \mathbf{w}_k + \rho_k [d_k - f(Net_k)] f'(Net_k) \mathbf{x}_k = \mathbf{w}_k + \rho_k \delta_k \mathbf{x}_k \\ \delta_k = [d_k - f(Net_k)] f'(Net_k) \end{cases}$$

Note that, continuing with the example, we have assumed a scalar output y but the result can be be generalized when the output is a vector.

Convergence Wittner and Denker [87] proved that under certain circumstance the delta rule never converges.

2.1.1.2 BP Algorithm for the static case

A ***static network*** is a network which outputs only depend on the current inputs. For instance, a MLP is a static network.

A network is called ***feedforward*** if its neurons can be ordered in such a way that there is no connection between neurons i and j whenever $i \geqq j$ ([69]). This definition assumes that the network is directed: the connection $i - j$ is different to the $j - i$. In other words, a network is feedforward whenever it can be represented as an acyclic directed graph [36].

To start, let us consider the simplest case, where the network makes to correspond to every input pattern $\mathbf{x}_j$ an output $\mathbf{y}_j$, no matter the historical inputs or outputs generated before; this is, with a static behavior with time. Assume we have the following feedforward network:

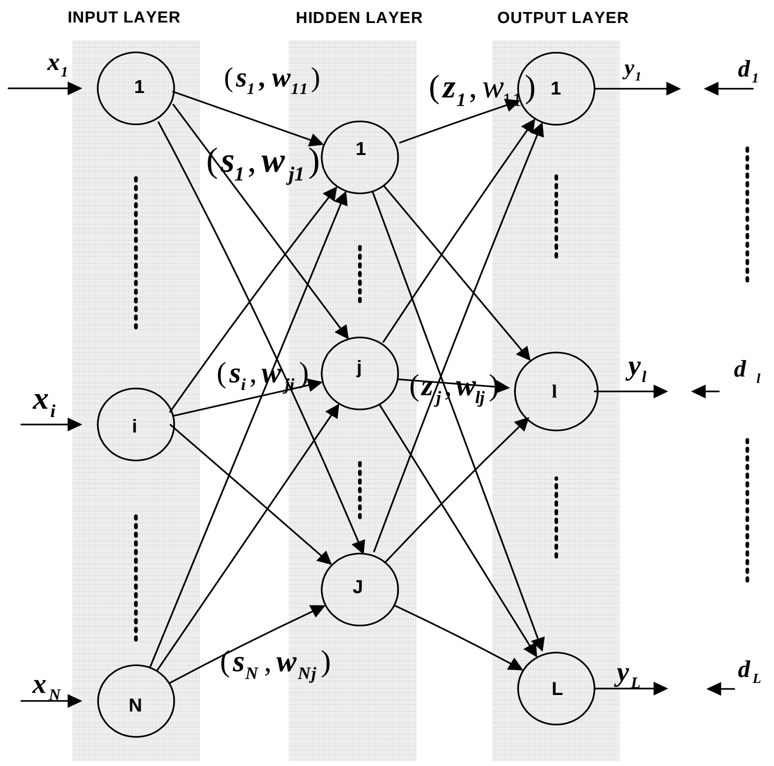


Figure 2.2

being s_i the output for the i-th input neuron, w_{ji} the weight between neurons i and j, and z_j the output for the hidden j-th neuron. The results obtained for this case can be generalized to more than one hidden layer [69].

Neurons must have non-decreasing output functions with continuous derivative. Each layer can have different output functions: f_o for the output layer and f_h for the hidden layer, respectively. Here we will assume the same output function (identical in both type and parameters) in the whole hidden layer, just for the sake of notational simplicity (it suffices to add a new index to denote a neuron inside the layer for function *f).* The same statement holds for the output functions corresponding to the output neurons.

This network receives a set of scalar values (signals) $[x_1, ...x_N] = \mathbf{x}$ as input, with $\mathbf{x} \in \mathbb{R}^N$**.** The hidden layer returns a real-valued vector $\mathbf{z} = [z_1, ...z_J] \in \mathbb{R}^J$, and the output layer another vector $\mathbf{y} = \mathbf{y}(\mathbf{w}, \mathbf{x}) = [y_1, ...y_L] \in \mathbb{R}^L$ which, when the network is fully trained, should be identical (or at least very close) to the desired vector $\mathbf{d}$ associated with $\mathbf{x}$**.**

Let us consider a set of m input/output pairs $\{\mathbf{x}_k, \mathbf{d}_k\}$ where $\mathbf{d}_k$ is an L-dimensional vector that represents the desired output for the network when the input is $\mathbf{x}_k$. The goal is to adjust the weights in order to learn correctly the correspondence between the inputs $\mathbf{x}$ and the underlying desired output values $\mathbf{d}$. As we have the desired value for each input, we can define an error function to evaluate the approximation quality for every set of network weights. For the moment we will use the following expression as the error function :

$$E(\mathbf{w}) = \frac{1}{2}\sum_{l=1}^{L}(\mathbf{d}_l - \mathbf{y}_l)^2$$

where $\mathbf{w}$ represents the whole set of network weights, and where we omitted, for simplicity, the pattern indexes (in the next subsections another feasible criteria to define error functions are discussed).

The algorithm view as an optimization problem

Since we defined an error function, the learning process is reduced to an optimization process: we wish to minimize the error function over the space of all possible weights. The function E is differentiable, hence, we can produce a local search with a traditional gradient descent and obtain a learning rule. This approach was independently analyzed by Amari (1967, 1968), Bryson and Ho (1969), Werbos (1974) and Parker (1985).

The rule of incremental learning Given that the desired values for the outputs are known, we can use the delta-rule (a special case of gradient descent) in order to update the weights w_{lj} that finish at the output neurons obtaining that:

$$\Delta w_{lj} = w_{lj}^{new} - w_{lj}^{current} = -\rho_o \frac{\partial E}{\partial w_{lj}} = \rho_o(\mathbf{d}_l - \mathbf{y}_l) f_o'(Net_l)\mathbf{z}_j$$

being ρ_o the ***learning rate*** for the neurons at the output layer, and $l = 1, 2, ...L$ $j = 0, 1, ...J$. We stick to the rule (as in Subsection 2.1.1.1) that the activation threshold θ_i for each neuron is simulated adding an an auxiliary input with a constant unit value and weight θ_i coming from an artificial neuron. We do not represent that artificial neuron in the diagram nor its connections, but the activation thresholds θ_i must be adjusted as additional weights when training the network. Additionally, f_o' represents the first derivative of f (output function) with respect to *Net,* and w_{li}^{new} and $w_{li}^{current}$ represent respectively the updated and current weights (after and before the application of the update rule). The values z_j (outputs of the hidden layer) are found by a propagation of the input vector $\mathbf{x}$ through the network:

$$z_j = f_h(\sum_{i=0}^{n} w_{ji}s_i) = f_h(Net_j)$$

with $j = 1, 2 \ldots J$ and f_h the output function for the neurons in the hidden layer (h).

The learning rule for the weights entering the hidden layer is not so obvious because we do not know the set of desired output values for the hidden layer. However, we could try to derive that rule minimizing the error for the output layer. This is equivalent to back-propagate the errors from the output layer $(d_l - y_l)$ towards the hidden neurons in an attempt to dynamically estimate the desired values for those units. This learning rule is called ***incremental back-propagation***. It is worth to notice that the delta-rule can be used to update the weights that reach the output layer. As a consequence, BP is regarded as an extension of the delta-rule. This version of the algorithm is said to be incremental, given that the weight adaptation is produced in accordance with the presentation of the input patterns. Later we will describe another version of BP, called ***batch.***

The weights that reach the hidden layer are modified according to a gradient descent with respect to the hidden weights from function $E(\mathbf{w})$:

$$\Delta w_{ji} = -\rho_h \tfrac{\partial E}{\partial w_{ji}} \; j = 1, 2...J \; i = 0, 1, 2...N$$

where the partial derivatives with the current weights must be found. By means of the chain rule we get that:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial Net_j} \frac{\partial Net_j}{\partial w_{ji}}$$

and by the facts that $\frac{\partial Net_j}{\partial w_{ji}} = s_i$ and $\frac{\partial z_j}{\partial Net_j} = f'_h(Net_j)$ we get the learning rule by replacement:

$$\Delta w_{ji} = \rho_h [\sum_{l=1}^{L} (d_l - y_l) f'_o(Net_l) w_{lj}] f'_h(Net_j) s_i$$

Comparing the last expression with the associated with the output layer, we can define an "expected estimated value" d_j for the j-th hidden layer in terms of a back-propagated error:

$$d_j - z_j \overset{corresponds}{\longrightarrow} \sum_{l=1}^{L} (d_l - y_l) f'_o(Net_l) w_{lj}$$

These equations can be extended to networks with more than one hidden layer, or networks where there are connections between non-adjacent layers [69]

The incremental back-propagation algorithm can be outlined as follows:

Step 0 – Initialization
Initialize the weights with small values chosen uniformly at random. They should be small to prevent the network paralysis (loosing capability of learning) and random to fight against possible symmetries in weights and avoid other learning problems [29]. *We take these values as "current": $w_{ij}^{current}$. Besides, assign to both ρ_o and ρ_h positive small values (with big values we risk to pass over the minimum and oscillate).*
Step 1 – Application of a pattern
Present an arbitrary vector $\mathbf{x}_k$ to the input network chosen uniformly at random from the training set, and propagate it through the network, finding the corresponding output values using the current weights.
Step 2 – Find the error for that pattern
Compare the real outputs with the correct ones $\mathbf{d}_k$ associated with $\mathbf{x}_{k.}$ and find the error at the output layer. Find the changes in the weights that reach the output layer, by means of the formula

$$\Delta w_{lj} = \rho_o(d_l - y_l) f_o'(Net_l) z_j.$$

Step 3 – Find the increments in the weights
Find the increments in the weights that reach the hidden layer,Δw_{ji}, by using

$$\Delta w_{ji} = \rho_h [\sum_{l=1}^{L} (d_l - y_l) f_o'(Net_l) w_{lj}] f_h'(Net_j) s_i$$

Step 4 – Update the weights
Update all the weights according to

$$w_{lj}{}^{new} = w_{lj}^{current} + \Delta w_{lj}$$

and

$$w_{ji}{}^{new} = w_{ji}^{current} + \Delta w_{ji}$$

for both the output and hidden layer, respectively.
Step 5 — Convergence test
Repeat Steps 1 to 5 with all the training patterns. Fix $w_{ji}^{current} = w_{ji}^{new}$ and $w_{lj}^{current} = w_{lj}^{new}$ and return to Step 1 until the output error, for all the vectors over the training set, have been reduced below an acceptable (previously established) value. Other convergence tests can be carried out, using other error functions [29].

Convergence: The algorithm, when converges, finds a local minimum that is not necessarily the global minimum.

Generally, an algorithm can not be proved to converge [30]. For that reason, several stopping criteria have been suggested, being one of them the "***cross validation***" (see 2.2.3 on page 46).

The algorithm uses an instantaneous estimation of the gradient of the surface error in the space of weights. Additionally, the pattern to be presented is randomly chosen, that is of the first point where the gradient vector will be found. As a consequence, the algorithm has a tendency to zigzag with respect to the correct direction, once it is close to a minimum of the error surface. This occurs because it is an application of the Robbins-Monro method for stochastic approximation [30]. As soon as it does zigzag trajectories, it tends to converge slowly.

We say that the series of solutions $\{x_k \quad k = 1, 2, ...\}$ produced by an iterative algorithm has ***q-linear convergence*** if there exists a real number $a : \; 0 < a < 1$ such that $e(x_{k+1})/e(x_k) \leq a$ being $e(x_k)$ the error of x_k with respect to the optimal solution $\mathbf{x}^*$. For example, a possible selection is $e(x_k) = \|x_k - x_k^*\|$. Saarinen and Cybenko [74] developed a study of the convergence rate for some methods described in this work (direct gradient descent, Newton's method, conjugated gradient, etc.), and showed that the common gradient descent has q-linear convergence rate, with an asymptotic constant error that equals $(k-1)/(k+1)$, being k the condition number of $H(\mathbf{x}*)$, H the Hessian matrix of E and $\mathbf{x}^*$ the global minimum for the error function. They also showed that under certain circumstances (e.g. when the Jacobian or Hessian matrix are "ill-conditioned"[2] in some of the iterative points), the direct gradient descent does not converge when implemented on a computer.

Gori and Maggini [25] proved the algorithm in its incremental version does converge to an optimal solution when used to train a network for pattern classification, whenever they are linearly separable. What is more, it makes the correct pattern classification independently of the choice of the learning rate.

The two calculation phases in Back-Propagation It is instructive to observe that the incremental back-propagation algorithm has two-run stages: the first run is ***forward*** whereas the second run is ***backward***. During the forward stage the weights do not change, and the output is found for each neuron, given the presented input (i.e. the input pattern). After having found the output values for each output neuron, they are contrasted with the corresponding desired values, and so we get an error for that pattern in each output neuron. Immediately, the second backward stage starts (from the right towards the left in our network), layer by layer, finding a local gradient for each neuron and the changes in the weights, always from the output layer towards the input. Both stages can be graphically summarized as follows:

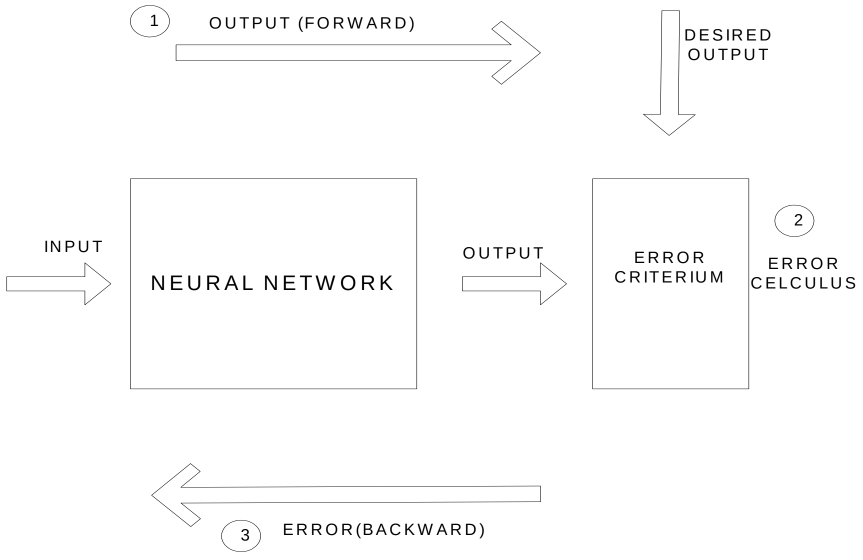


Figure 2.3

The interconnection algorithm This algorithm is an implementation of the general

[2]A matrix is said to be "ill-conditioned" when the quotient between the highest and lowest singular values of its singular value decomposition is very high.

incremental back-propagation algorithm, and is based on two networks: the original one, where the input data is propagated, and the dual network, where the errors are propagated. The interconnection algorithm was encoded for the first time by Lefebvre (1991) ([69]).

If we consider the network as a set of adjacent layers, the vector **x** (input) in one layer is the output (**y**) of the previous layer, from left to right:

$$x_i^{layer\ L+1} = f(\sum_j w_{ij} x_j^{layer\ L})$$

Graphically, we can represent the interconnection algorithm as follows:

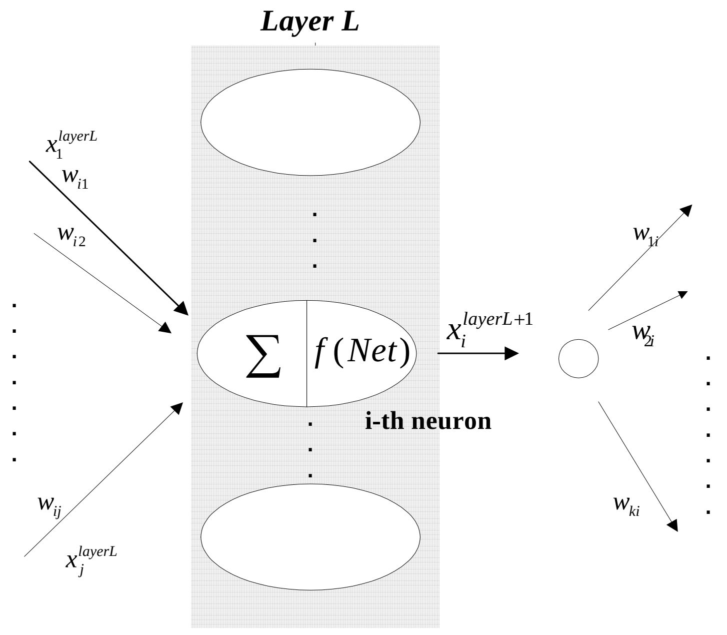


Figure 2.4

where the circle represents a bifurcation point or paths separation.

If we express the weight update with

$$\Delta w_{ji}(n) = \rho \delta_j(n) y_i(n)$$

taking $y_i(n) = s_i(n)$ *or* $y_i(n) = z_i(n)$ depending whether is a hidden or output layer, respectively, then

$$\delta_i^{output\ layer}(n) = \varepsilon_i(n) f'(Net_i^{output\ layer}(n)) \text{ for the output layer}$$

$$\delta_i^{layer\ L}(n) = f'(Net_i^{layer\ L}(n)) \sum_k \delta_k^{layer\ L+1} w_{ki}(n) \text{ for the other layers } \textbf{(Eq. 2.1)}$$

By its definition, δ_i is the ***local gradient*** [30] for the i-th neuron. It holds that

$$\delta_i(n) = \frac{\partial E(n)}{\partial Net_i(n)} = e_i(n) f_i' \left[Net_i(n)\right] \textbf{ (Eq. 2.2)}$$

being $e_i(n)$ the error from neuron i in the output layer, at step n.

We can represent Equations 2.10and 2.20 with the following diagram:

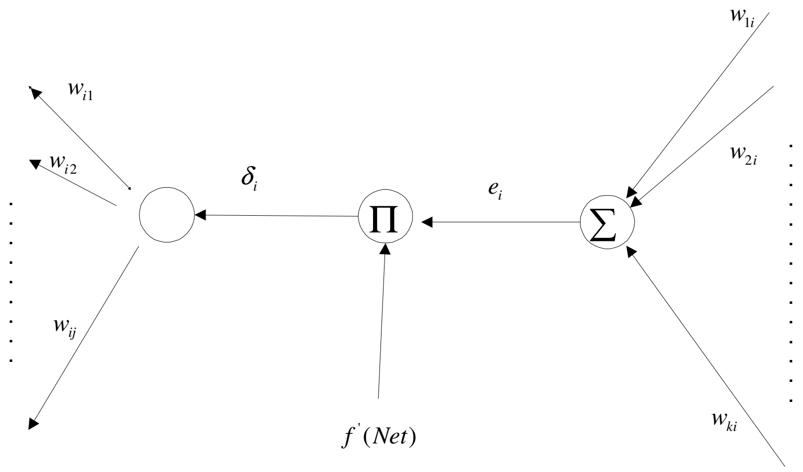


Figure 2.5

The network shown in Figure 2.5 is, by its definition, the ***dual network*** (or adjoint) of that from Figure 2.4.

The relation between both networks is that in the original the input (x_i) flows from the i-th neuron from left to right, whereas the local gradient (δ_i) in the dual network travels from right to left, briefly, the inputs are outputs in the dual and vice-versa. Besides, the adding points $(\sum)$ in the original network are separation points in the dual network, and simultaneously, separation points are now adding points in the dual network. The weights of the corresponding connections are preserved. The error in the dual network is multiplied by $f'(Net_i)$ to produce δ_i. The local gradient is proportional to the local error (Eq. 2.2).

A conclusion from these observations is that we can imagine that the error $\varepsilon_i = d_i - y_i$ produced in the output of the original network is injected to the input layer of the dual network, and allow us to find the gradient of the error in the original network as the output of the nodes in the dual network. Therefore, we can re-write the back-propagation algorithm as follows:

- ***Step 1*** *– Initialize the weights with small values uniformly chosen at random.*
- ***Step 2.***
  - ***Repeat*** *the following steps until reaching a negligible error or a non-convergence test holds:*
    - ***Step 2a*** *– Choose a pattern at random from the training set, present it to the network and get the output for each neuron of the network.*
    - ***Step 2b*** *– Inject the computed error through the dual network, and find the local gradient in each neuron.*
    - ***Step 2c*** *– Update the weights according to the search rule chosen. For instance, for the direct gradient descend we have that:*

$$\Delta w_{ij}(n) = \rho \delta_j(n) y_i(n)$$

  End Repeat

It can be seen that the local gradient is available as a value (signal) at the nodes of the dual topology, which avoid us to compute the expression

$$\delta_i^{layer\ L}(n) = f'(Net_i^{layer\ L}(n)) \sum_k \delta_k^{layer\ L+1} w_{ki}(n)$$

Instead, it suffices to have the topological specification of the network, and the computation via backward propagation in the dual network makes the rest for us. The implementation of the BP algorithm in the dual network is much more versatile than encoding its equations directly, mainly because the dual network can be easily settled up from the topology of the original network. As we will see, BP can be adapted to train recurrent networks with dynamic learning, and the interconnection algorithm can be generalized for those cases as well [69]. What is more, the algorithm can be adapted to an arbitrary topology, whenever the dual network can be built. In accordance with [69], this is the best way to implement "back-propagation", and it is the chosen method for the neural networks simulator, though there is not information at disposal about the implementation of the data flow.

To sum up, the key advantage of this algorithm is that it avoids long computations using output functions in the original network: the local gradient is always available at the nodes of the dual network.

Incremental vs. Batch An alternative to the incremental learning described before is to use ***batch learning***, where the weight updates occur only after the presentation of the whole pattern set and the complete possible variations of weights are found, with no application of them (recall that we are working with a finite training set). Formally, the batch learning is implemented considering the sum of the right-hand sides of the weight changes equations, for all the possible inputs $\mathbf{x}_k \quad k = 1, \ldots m$:

$$\Delta w_{ji} = \sum_{k=1}^{m} \rho_h [\sum_{l=1}^{L} \{(d_l)_k - (y_l)_k] f'_o[(Net_l)_k] w_{lj}\} f'_h[(Net_j)_k](s_i)_k \text{ for the hidden neurons}$$

$\Delta w_{lj} = \sum_{k=1}^{m} \rho_o[(d_l)_k - (y_l)_k] f'_o[(Net_l)_k](z_j)_k$ for the output neurons,

where $(d_i)_k$ refers to the i-th element from the desired output $d_k = [(d_1)_k, (d_2)_k \ldots (d_L)_k]$ and $(Net_j)_k$ represents Net_j for the pattern k-th. This is precisely a descent according to the gradient of the objective function

$$E(\mathbf{w}) = \tfrac{1}{2} \sum_{k=1}^{m} \sum_{l=1}^{L} [(d_l)_k - (y_l)_k])^2 \textbf{ (Eq. 2.3)}$$

The batch update moves the search point **w** in the correct gradient direction in each iteration. However, the incremental update is desirable for two reasons: its implementation in a computer saves storage (it is not needed to keep information about the cumulative weight variations) and the search path in the weight space is randomly chosen once the input pattern is picked at random among the set of training pairs {**x**, **d**}. The latter property enriches the diversification process during the incremental update, helping to better explore the search space, and, potentially, leads to solutions with better quality. By means of the stochastic approximation theory (Robbins-Monro) Finoff (cited by [29]) showed that, for learning rates close to zero, the incremental back-propagation tends to the batch technique and produces essentially the same results. Besides, for small learning rates, the stochastic element in the training process prints to the incremental back-propagation a "quasi-annealing" character, where the cumulative gradient (relative to error function $E(\mathbf{w})$ from Eq. 2.3) is permanently perturbed, thus running away from local minima with small plain attractors, or "basins" ([29]). Additionally, when the data set is redundant (there are several copies of the same pair {**x**, **d**}) the incremental mode possesses advantages for both pattern presentation and for that redundancy ([30]). For the reasons exposed before, the incremental training has been always chosen throughout this work.

2.1.1.3 Variants of the algorithm

In general, the learning with back-propagation is slow ([29] y [30]), because of the shape of the error surfaces, which present generally several plain regions or very steep ones, or even regions that are plain in the search direction. Several works studied the geometry of the error surface. These problems become more crucial when the network is designed for classification tasks, specially when the cardinality of the data set is small [29]. As a consequence, several variations to the algorithm have been proposed, most of them heuristics that try to speed-up the convergence rate, avoid local minima and/or improve the generalization capacity of the network. We mention here some of these modifications and in 2.20 they are treated more extensively.

Convergence towards local minimum These improvements try to accelerate the convergence rate towards a (surely local) minimum. See 2.2

Learning rate The convergence rate in back-propagation is directly associated with the learning rate (ρ_o and ρ_h in the equations), so several heuristics have been proposed to address the correct choosing of learning rates. Some of them try to determine the learning rate for each neuron statically, that is, independent of the learning phase, for instance, taking the inverse of square roots of the number of connections (or "***fan in***") that input the neuron ([30] Chap. 4). Other works suggest to dynamically adapt the learning rate with the time, for example,

$$\rho(t) = \rho_0 \frac{1}{1 + t/\tau}$$

where τ is a positive constant and t is the iteration [29].

Finally, other works (for example Almeida, cited by [69]) either increase or reduce the learning rate whether the search point is getting near or far away from the minimum. Almeida's algorithm takes a learning rate for each weight. The main idea is that the learning rate should be increased whenever the gradient component associated with that weight preserves its sign during two consecutive iterations; otherwise, it should be decreased. That increase (decrease) is geometric (i.e. multiplying it by a factor [51]).

Use of Moments It has been suggested to include an additional term of the Taylor series for $E(\mathbf{w})$ when used to find $\Delta\mathbf{w}$.

Such second-order term, or an approximation, is usually called first-order moment [61] or simply ***moment***. This artifice tries to speed-up the convergence when the error surface is near-plain, so to escape from local minima.

The learning rate using the first-moment order will be:

$$\Delta w_i(t) = -\rho\tfrac{\partial E}{\partial w_i} + \alpha\Delta w_i(t-1)$$

where $0 < \alpha < 1$ is the moment rate. This rate can be either constant or be dynamically changed with the iterations ([29]).

The implementation of moments is inspired in the Newton's method, where the weights are updated in the following manner

$$\Delta\mathbf{w} = -(\mathbf{H}(\mathbf{w}^{actual}))^{-1}\nabla E(\mathbf{w}^{actual})$$

Given that the implementation of Newton's method is computationally intensive (with order $O(N^3)$, being N the number of weights), several heuristics have been proposed as well as approximations to get second-order methods that reduce the computational cost (see [29]).

The moment rate can be either constant or change with the iteration number. Suppose we want to adjust α in each step such that the gradient search returns a local optimum, that is, we wish to find α for time t such that the error E at time $t+1$ is minimum. In this case, the moment rate obtained [29] is

$$\alpha(t) = \rho\frac{\nabla E(t)^T\Delta\mathbf{w}(t-1)}{\|\Delta\mathbf{w}(t-1)\|^2}$$

In the most general way, more terms from the Taylor series for $E(\mathbf{w})$ can be added; for instance, the third-order terms (called ***second-order moments*** [61]).

While the original batch algorithm with its proposal: $\Delta w(t) = -\rho\frac{\partial E}{\partial w(t)}$ converges towards a minimum for the error function (when regarded as the sum of square errors) in a certain time t not higher than $(\lambda_{\max}/\lambda_{\min})/4$, where $(\lambda_{\max}/\lambda_{\min})$ is the ratio between the highest and lowest eigenvalues for the Hessian matrix of E with respect to $\mathbf{w}$ (note that if E_{min} represent the local minimum found, then E - E_{min} tends to 0 slower than $e^{-4t\frac{\lambda_{\min}}{\lambda_{\max}}}$ when t tends to infinite), adding the first-order moment the time is improved to be not higher than $[\sqrt{(\lambda_{\max}/\lambda_{\min})}]/4$ even for the "batch" case. However, if a second-order moment is added, this bound practically does not change. What is more, the relative advantage tends to 0 when $\lambda_{\max}/\lambda_{\min} \to 0$ ([61]).

Conjugated Gradient These methods are based on geometrical considerations, and have been tested for several authors in multilayer feedforward networks, showing to be better than the original back-propagation, in terms of convergence [29]. The key is to look for the minimum using directions such that the direction search for iteration k, $\mathbf{d}_k$, is conjugated[3] , with respect to the Hessian matrix of E , with the previous $\mathbf{d}_1 \ldots \mathbf{d}_{k-1}$, employing a process similar in spirit to the Gram-Schmidt orthogonalization scheme. Each search direction will be given by

$$\mathbf{d}_k = -\nabla E|_{w_k} + \beta_k \mathbf{d}_{k-1}$$

being β_k a scalar: this parameter leads to different variants for the gradient conjugated method ([36],[10]). For instance, the Fletcher-Reeves formula takes

$$\begin{cases} \beta_k = \frac{\nabla_k^T \nabla_k}{\nabla_{k-1}^T \nabla_{k-1}} \\ \nabla_k = -\nabla E|_{w_k} \end{cases}$$

Output function The issue here is to avoid the premature neuron saturation (hence the network paralysis). To that purpose, we can add a disturbance to the derivative of the output function or alter some parameter that defines the function itself, in such a way that the weights are updated faster in the first training stages, and then, make it disappear the disturbance as long as convergence is reached.

Other convergence criteria The use of other convergence criteria, such as alternative error functions (different to the euclidean distance), will induce variations in the time to reach the desired solution. Additionally, the use of cross validation will both modify the training execution time and help to obtain better final results, when regarded the generalization capacity (see 2.2 on page 45).

The use of other error functions (criteria) lead us to have different convergence criteria for the algorithm. To the sake of the greatest generality, the error function must meet certain basic conditions: the characteristics of a distance (or at least to be non-negative and null when the two points are identical) and have first continuous derivative in order to apply the algorithm. As a concrete example, we can cite the Minkowsky function of degree *r:* $E(\mathbf{w}) = \frac{1}{r}\sum_{i=1}^{n} |d_i - y_i|^r$with $r > 0$ and i over the number of output neurons. When $r = 2$ the square error is retrieved [4](except for the constant factor 1/2). When $r = 1$ we get the ***Manhattan norm*** [41], that owns interesting mathematical properties (see 2.2 on page 45). There are other error measures based on the relative entropy ([29], [5], [4]).

2.1.1.4 Other ways of improving the solution

Search for global optimum The back-propagation algorithm converges to an optimum that is not necessarily a global one, the search for "the" minimum is an evident and appealing improvement. We will give next a brief overview of some promising techniques. The reader can find further details in 2.3 on page 50.

[3]This is that $\mathbf{d}_j^T \mathbf{H} \mathbf{d}_j = \mathbf{0}$

[4]We consider the instantaneous errors, for the recently presented pattern to the input.

Random gradient search This technique is designed to better explore the search space trying to evade local minima. The basic idea is to add noise to the weights that will be progressively vanish with the number of iterations. The so obtained learning method (called the ***Langevin method***) owns valuable properties: from a theoretical viewpoint, it is similar to simulated annealing, since a "heat bath" is given to the network (the additive noise represents the temperature of the heat bath, which will decrease with time to produce the annealing) helping to get weight settings that escape from local minima [72]. From a practical viewpoint, it can be combined with other back-propagation techniques, and has no great computational requirements.

This method, together with genetic algorithms and simulated annealing, as we will see next, tries to escape from local minima. In this work a function from the neural network simulator called "jog" has been used. The function sums to each weight (in a desired training iteration) a uniform noise with mean the current weight and specified variance for all the weights.

Methods not based on derivatives Evolutionary algorithms (for instance, genetic algorithms), together with simulated annealing are "non-derivative" methods. In these methods we can even use non-differentiable output functions, because their fitness function is not necessarily differentiable. This property permits the application of complex error functions without sacrificing the elapsed computational time.

By means of evolutionary algorithms, better convergence rates can be obtained than with simulated annealing, though they are slower than the derivative-based methods ([36]).

Evolutionary algorithms allow parallel searches, that can be implemented in multi-core or independent computers to massively accelerate the process. They can also be applied to continuous or discrete problems, and help to identify the network structure as well as the parameters (weights) in complex models ([36]).

However, given that genetic algorithms provide searches in random directions, finding the global optimum will possibly require a considerable, if not prohibitive, amount of time. Besides, it is hard to develop analytical studies of them, in part because of their randomness. As a consequence, most of the knowledge about these algorithms is derived from empirical studies ([36], [90]).

There are two ways to implement evolutionary algorithms to train a neural network:

- ***directly***, trying to evolve a set of weights $\mathbf{w}_j$ in order to minimize the error function. This method can lead to a slow and inefficient process, using large storage resources. The training would be a genetic search of the optimum weights.
- ***in a hybrid algorithm***: the network is divided into two sub-networks: the gradient descent is applied for the network with hidden and output layer, whereas the sub-network that consists of input/hidden layers a GA is introduced pointing to obtain the desired output whenever the input of the hidden/output sub-network equals the output of the input/hidden sub-network. The solution converges faster than in the direct way (see 2.3 on page 50 and [29]).

Elimination of weights and neurons The seminal works from Baum and Haussler ([5]) suggest that the number of training patterns should be considerably higher than the number of weights [5] in the network to get a good generalization capacities. A natural consequence

[5]In fact, it is the number of independent weights, or degrees of freedom for the associated optimization problem. The weights can be equal or related to reduce that quantity, by means of special techniques as "weight sharing" (see 2.8 on page 85).

is the need to reduce the number of such weights, hence simplifying the network structure and improving the generalization capacity for the same training data set. This is specially useful when few training information is available. As a result, given that each neuron needs certain time for training, the learning rate is increased when some of them are deleted together with their corresponding weights. It is certainly hard to predict the optimal number of neurons or weights that should be deleted (or kept) beforehand. Hence, some automatic weight reduction techniques were studied, that give hints to decide which neurons to keep or delete. Basically, the idea is to include a penalty in the error function to the weights which are non-zero (or different each other). In this way, after successive training steps those weights are reduced in magnitude faster than linearly, whenever the learning rate for that neuron is negligible (in other words, if that weight practically does not change with a normal training). When all weights that reach a certain neuron that is not in the input layer (or weights that depart a certain neuron that is not in the output layer) are close to null, this neuron is redundant and hence can be deleted. These techniques are called ***pruning algorithms***. Reciprocally, we can start with a very small network and enrich it progressively, defining ***growing algorithms*** ([30], [10]).

Adding noise to the desired values Up to this moment additive noise has been included in different parts of the network to improve the convergence rate (in the weights as in the case of Langevin, or implicitly at the input, via incremental back-propagation). Another option is to add noise to the desired values and minimize the "instantaneous" error (the one obtained with the presented input pattern) instead of the sum-square error: for each desired value $\mathbf{d}_i(t)$ a new desired value is taken $\mathbf{d}_i(t)+\mathbf{n}_i(t)$, where $\mathbf{n}_i$ represent independent equally distributed (normal or uniform) additive noises, with zero mean and finite variance, also independent of both the input values $\mathbf{x}_i(t)$ and the desired vectors $\mathbf{d}_i(t)$, and we compute

$$\varepsilon_i = \tfrac{1}{2}\textstyle\sum_i [\mathbf{d}_i(t) - \mathbf{y}_i(t)][\mathbf{d}_i(t) - \mathbf{y}_i(t)]^T \text{ (see [83]).}$$

It can be proved that, under these conditions and using Borel measurable output functions (such as a sigmoid or any other continuous function of the commonly used in the neural networks) the final values are not affected in the following statistical sense:

$$\langle \mathbf{y}(t) + \mathbf{n}(t) \rangle_{\mathrm{x}(t)} = \langle \mathbf{y}(t) \rangle_{\mathrm{x}(t)}$$

What is more, this conclusion holds for *every* network architecture, having proved the result for both dynamic and static networks, achieving good performances [83].

This methodology has a practical advantage: we can use the existent network simulators, because the algorithmic back-propagation implementation is in no way affected when the desired values are modified.

2.1.2 Networks Committees

A way to improve the performance of neural networks is to use several independent networks with different sizes and characteristics to address the same problem. The idea comes from previous research developed around the properties of the estimator (in our case, the predictor of the future values) obtained with an average of independent estimators [62]. The application of these ideas leads to interesting results, specially when the training data set is small. When different networks are integrated making another network a ***modular network*** is obtained ([68] y [69]). These networks, having equal number of neurons with a ordinary network, have less weights, since neurons are not fully connected. As a consequence, their learning phase is shorter and less training samples are needed. On the other hand, there is scarce research on how to divide a feedforward network into modules in an optimal fashion [69].

Suppose we train C different networks with the same data set. If we picked the network with the best error over the training set, we would not be taking the best decision as we'll see. First, we would be wasting the training of the other networks (discarding potentially useful information stored in them). Second, it is worth to remark that the cross validation set is randomly chosen, so there is a non-negligible probability that other network has better performance for new data, with the same probability distribution. A much better strategy is to use all the trained networks, that is, to use a ***network committee.*** Formally, a network committee is a "set of trained neural networks with the same data set, whose topologies tend to be different, and their output are interpreted as a vote for classification (or a suggested value for prediction)" [69].

Model combination using concepts from *"machine learning"* Each predictive model used, including parametric models, makes assumptions about the available data that sometimes are not true, providing the different models good approximations in different circumstances. A valuable approach [3] is to adequately combine the models in order to obtain complementary decisions (having a committee when all models are neural networks), in order to gain accuracy in the result. This can be realized in several ways:

a) using the same learning algorithm but using different hyper-parameters (such as the number of hidden layer of a perceptron, or the starting values of the network weights prior to the implementation of back-propagation, etc.). Distinct models can be obtained in this way, and the resulting average gains in a variance reduction in the resulting prediction.

b) taking different representations and generating models of the same input object to get the output. The idea is to have different and independent predictions, using distinct combination of dimensions at a time, because the simultaneous use would turn the model complex and would need too many data to adjust it.

c) using distinct training data sets for the different models. This can be do by extracting training sub-data sets at random ("bagging" technique). If the partition of the training data set is done in such a way that each model is specifically trained in a region over the input domain, we obtain what is called ***mixture of experts***.

We are then interested in how to combine the different models to generate the final series. Specifically, we are concerned with the combination of multiple experts: models that work in parallel, where each one is trained, their prediction is taken and a subsequent mixer returns the final decision using these predictions. Examples of this technique is voting, the mixture of experts and the "stacked generalization". Here we will analyze voting and stacked generalization, because of its use and relevance in this work.

a) Voting: This is the simplest way to combine several models: taking a weighted average from the results of each model. This method is sometimes also called "ensembles". Perrone [62] shows a remarkable property of ensembles relative to the resulting average error produced when the number of predictors is increased (see 3.4 on page 165). When the weights are identical for all the models, we have a ***simple voting***.

An alternative is to rank the accuracy of each model and assign higher weights to the more accurate models.

The voting schemes can be seen as approximations in a Bayesian context, being the weights an estimation of the a priori probabilities of the model, and the models outputs represent model's conditional probabilities (Bayesian combination of models). Analytically:

$$P(y_j = d) = \sum_{all.models.M_j} P(y_j = d|x, M_j)P(M_j)$$

being x the input data, y_j the output for the j-th model and d the desired output for the input x.

Simple voting just corresponds to a simple uniform a priori probability.

When a simple voting is chosen regarding a set of L predictors with output y_j $j = 1...L$, independents and identically distributed with mean $E(y_i)$ and variance $Var(y_i)$, and an output $\bar{y}$ is found as a function of them, then

$$E(\bar{y}) = E(\sum_j \frac{1}{L} y_j) = \frac{1}{L} L E(y_j) = E(y_j)$$

$$Var(\bar{y}) = Var(\sum_j \frac{1}{L} y_j) = \frac{1}{L^2} Var(\sum_j y_j) = \frac{1}{L} Var(y_j)$$

The expected value does not change, but the variance (and hence the mean square error) is decreased with the numbers of predictors L.

If they were not independent we would get:

$$Var(\bar{y}) = \frac{1}{L^2} Var(\sum_j y_j) = \frac{1}{L^2} \left[\sum_j Var(y_j) + 2 \sum_j \sum_{i<j} Cov(y_j, y_i) \right]$$

Note that the variance could be reduced whenever the votes are negatively correlated [3].

b) stacked generalization This technique was proposed by Wolpert and extends the voting ins the sense that the models are combined in accordance with the weights offered by a system (which is in fact another model) $f(\bullet|\varphi)$ whose parameters φ are also trained:

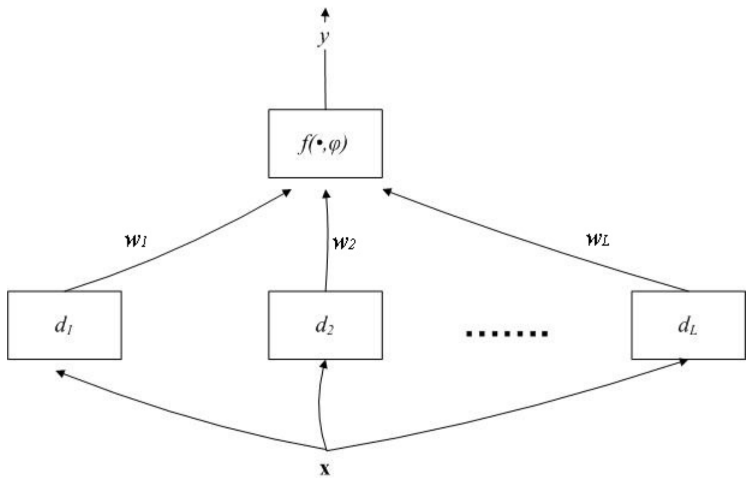


Figure 2.6

If $f(\bullet|\varphi)$ is a linear model with constraints $w_i \geq 0, \sum_i w_i = 1$ the optimum weights can be obtained by regression, as Bishop proposes [10].

When stacked generalization is performed, the models are intended to be as dissimilar as possible to be complementary. It is even recommended to have the models trained by using different algorithms (for instance, to mix a parametric predictor with a MLP).

The "ensemble method" The mean square error when the average solution for all networks is chosen (that is, the average over the output values $y_{COMMITTEE} = \frac{1}{C}\sum_{i=1}^{C} y_i$) assuming the errors in each network have zero mean and are not correlated will be (Perrone [55] and cited by [69]):

$$E_{COMMITTEE} = \frac{1}{C}\overline{E}$$

where $\overline{E}$ is the MSE averaged over all individual networks. As a consequence, the resulting error is statistically C times lower than the average of the MSE, so the resulting error coming from a network committee can be reduced as low as desired increasing the size of the networks set. While this is something optimistic, given that the errors between the different networks are not uncorrelated and in that case not drastic improvements are performed, it can be proved that the error $E_{COMMITTEE}$ is never greater than $\overline{E}$ anyway ([10]).

Advantages of the committees A valuable property of committees is that, even though we are considering estimations generated from neural networks, the same could be generated, as previously mentioned, by other kinds of models (for instance statistical models), and in the case that all the models were neural network-based, we might be working over different training sets. The latter has a remarkable corollary that we will now describe. A standard method to avoid "overfitting" when training is to use a "cross validation" set. The main drawback of this is that given that we use this method to escape from overfitting, each estimation is restricted to "see" only a subset of the whole data set, and will possibly be loosing valuable information over the rest of the data distribution, specially when the data set is small. This will always be the case when a prediction (estimation) uses "cross validation" as a stopping rule on a single network. However, using the committees method (sometimes named ***"ensemble process"*** [62]), once the set of estimators is found we might train each network with the whole data set and let the smoothing property [6] of the "ensemble process" remove any feasible overfitting. In this manner, the different networks are capable to see the whole data set, whereas if we used the cross validation method to avoid overfitting this would not occur [55]. Another clear advantage is the variance reduction in the error when the average of individual outputs is taken [30].

Possible improvements for network committees Bishop ([10]) shows that the error committed by the committee will be minimized when the weight p_i corresponding to model number i is

$$p_i = \frac{\sum_{j=1}^{C}\left(C^{-1}\right)_{ij}}{\sum_{k=1}^{C}\sum_{j=1}^{C}\left(C^{-1}\right)_{kj}}$$

where **C** denotes the correlation matrix for the errors between the networks:

$$C = \left\{C_{ij} = \langle\varepsilon_i\varepsilon_j\rangle_X\right\}$$

[6]Error reduction within a factor 1/C with respect to MSE

being $\varepsilon_i = d_i - y_i$, X the random variable that represents all possible patterns and the average is taken regarding all possible initial conditions of the networks.

In particular, if we approximate $\mathbf{C}$ from the available data we have

$$C_{ij} \approx \frac{1}{N}\sum_{n=1}^{N}(y_{in} - d_n)(y_{jn} - d_n)$$

being n the pattern number and y_{ik} the output result for network i and the pattern k.

2.1.3 Entropy and mutual information

Here we will sketch the basic properties for the entropy, the relative entropy and the mutual information. These concepts (specially the one of mutual information and relative entropy) will be useful in order to study the properties of the time series when choosing the error function, as well as when comparing the different models that predict the time series.

2.1.3.1 Entropy

Informally, the entropy is a quantification of the uncertainty for a certain random variable. Let X be a discrete random variable and $\boldsymbol{X}$ its possible outcomes. Denote its probability mass with $P(X = x) = p(x) \quad x \in X$. The absolute entropy (or Shannon entropy, or simply ***entropy***) of X is

$$H(X) = -\sum_{x \in X} p(x) \log_2 p(x)$$

Observations:

- The entropy is measured in bits. If the logarithm were natural, the entropy would be measured in ***nats***.
- We stick to the convention $0log0 = 0$. In words, the entropy is not altered when outcomes with null probability are added.
- The entropy does not depend on the outcomes *of* X, but on their probabilities.

A basic application of the entropy is that the length of the shortest binary code for an arbitrary random variable is within $H(X)$ and $H(X)$+1 ([19]).

Another interpretation for the entropy Consider a discrete random variable X. The feasibility of its outcomes x_k (noted as $x_k \in X$) can be interpreted as a message, that is, we consider this occurrence (event) as the message "X takes the value x_k". This message can be more or less expected (in other words, we can feel highly "surprised" or not with the message). The information that is carried by the message is its measure of uncertainty: indeed, a totally predictable message does not contain information at all. Denote with $P(x_k) = P(X = x_k)$ the probability that X takes the possible outcome x_k. We define the information carried by message x_k as

$$I(x_k) = log_2\left(\frac{1}{P(x_k)}\right)$$

and the entropy of X is the expected information of a message:

$$H(X) = \textstyle\sum_{x_k \in X} P(x_k) I(x_k) \qquad \textbf{(Eq. 2.4)}$$

Entropy properties From Eq. 2.4 we get that the entropy $H(X)$ can be interpreted as an average of the information carried in the random variable X [69].

The entropy is bounded by

$$0 \leq H(X) \leq log_2(N)$$

being N the possible outcomes for X. In fact, the entropy is the highest when $P(x_k) = 1/N$, precisely when the uncertainty is the highest, and minimum when$P(x_k) = 1$ for some outcome, or when there is complete certainty of the result (when X is constant).

Relative entropy The entropy of a random variable is a measure of the uncertainty about it. On the other hand, the relative entropy is a notion of distance between two probability density functions, and a measure of the inefficiency of assuming that the density function for X is q when the correct one is p. We define the ***relative entropy*** (or cross-entropy, or Kullback-Leibler distance) between two probability distribution functions $p(x)$ and $q(x)$ for the discrete random variable X with possible outcomes $\boldsymbol{X}$ as [19]:

$$D(p||q) = \sum_{x \in X} p(x) log_2 \frac{p(x)}{q(x)}$$

We stick again to the conventions $0Log(0/q) = 0$ and $pLog(p/0) = \infty$.

The relative entropy is always non-negative and assumes the value 0 if and only if $p = q$.

In the continuous case is

$$D(p||q) = \int_{-\infty}^{+\infty} p(x) log_2 \frac{p(x)}{q(x)} dx$$

where p and q are the density functions.

Application: relative entropy as an error function When the desired and real network outputs represent probability vectors it is reasonable to use the relative entropy as an error function: $D(X \| Y)$ is non-negative, convex in its two variables and gets null if and only if $x = y$. Strictly speaking, it is not a distance, because it is not symmetric and the triangular inequality does not hold. The symmetry can be obviously recovered taking as distance $d = D(X||Y) + D(Y||X)$, but this is not necessary in practice. In that case we then take $D(X \| Y) = \sum_{x_i \in X} x_i \log_2 \frac{x_i}{y_i}$. In [6] a practical application is fully detailed.

If the sum of the network outputs equals 1 (or, in the terminology of the networks simulator, they are "softmax"), then it is proved that the relative entropy criterion can be implemented by the MSE criterion (that is, to minimize the relative entropy is precisely the same as to minimize the MSE between the desired and real network outputs) [69].

In [4] the authors propose that, in the case $y_i = x_i + \varepsilon_i \quad \forall i$, then we get

$$D(X||Y) \approx \sum_{x_i \in \mathbf{X}} \frac{\varepsilon_i^2}{x_i}$$

Conditional entropy The conditional entropy between two discrete random variables X e Y is defined as:

$$H(Y|\,X) = \sum_{x \in \mathbf{X}} p(x) H(Y\,|X = x)$$

Mutual information The ***mutual information*** is, in a first approach, a measure of the information that carries one random variable about another one, or the reduction of uncertainty of a random variable given the knowledge of another one. The entropy is then the information that a random variable has about itself. Formally, let us consider two random variables X and Y with joint probability $p(x,\ y)$, and marginal probabilities $p(x)$ and $p(y)$ respectively. The ***mutual information*** *$I(X,\ Y)$* is the relative entropy between the joint probability and the product of the marginals [19]:

$$I(X,Y) = \sum_{x\in X}\sum_{y\in Y} p(x,y)\log_2 \frac{p(x,y)}{p(x)p(y)}$$

Application to the "optimal delay" The mutual information can be used to determine the value of the "optimal delay" (see 2.9.4.2 on page 102) for the reconstruction of the state space. The ***delayed mutual information*** is a particular case of mutual information, applied to the outcomes of a certain variable $X(t)$ and its delayed version $X(t+T)$. A difference with the autocorrelation is that the mutual information takes into account non-linear correlations [31]. The delayed mutual information is computed by

$$I(X+T,X) = I(T) = \sum_{i,j} p_{ij}(T) log_2 \frac{p_{ij}(T)}{p_i p_j}$$

where $A_1, A_2, \ldots$, represents a partition of the real field, p_i is the probability to find a value from the time series inside A_i and $p_{ij}(T)$ the joint probability to have one value inside A_i and other value after T time steps inside the set A_j (see 2.9.4.2 on page 102). If we plot$I(T)$ versus T, $I(T)$ starts with a really high value ($I(0) = 1$) and when T is increased, $I(T)$ decreases until a certain time where it increases again. There are strong arguments [31] that suggest the delay where $I(T)$ reaches the first minimum should be used as the "optimal delay" of the signal sub-space.

Basic relations It can be proved ([19]) that

$$I(X,Y) = H(X) - H(X|Y)$$

As a consequence, the mutual information is the reduction of uncertainty for X given the knowledge of Y.

It can be also proved that

$$I(X,Y) = I(Y,X)$$

so X says of Y as much information as Y says of X.

Finally, the following equalities hold:

$$I(X,X) = H(X)$$

$$H(X,Y) = H(X) + H(X|Y)$$

2.1.4 Networks with dynamic behavior

Static networks are trained to produce a spatial output as an answer to a spatial input (here the term spatial means that it does not depend on time), that is, they represent static systems. However, in several applications it is required to model dynamic processes where the output is a sequence of changing values with time, corresponding to a certain input variable that is also time-dependent. The aim is to design a model capable of simulate the reality, and dynamically adapts its parameters to approximately capture the observations of the real system. Briefly, a model with a dynamic behavior. We will call such networks simply ***dynamic***. A recurrent network is a clear example. Another example is a network TLFN (time lagged feedforward network). In this subsection we will be concerned with dynamical networks, with special emphasis in recurrent networks. The recurrent architecture allows us to include the time-dependent nature of the relations between the data. Additionally, it is possible to extend multilayer feedforward networks and their associated training algorithms (e.g. BP) to the time domain in order to train these networks.

2.1.4.1 Introduction

Dynamic behavior of the data Both the classification and the prediction problems can be formulated as a certain arbitrary mapping between two vector spaces. In the static case the dimension of the input defines the dimension of the patterns space (e.g. if the inputs are two independent variables, the input space has dimension 2). In static pattern recognition (and prediction) the way from which the data are taken from the data set is irrelevant: for instance, the classification problem is the same when we shuffle the data presented to the network, because we assume the data “clusters” are not assumed to be ordered (we do not assume there is an internal sequencing for the data).

In dynamic problems the measures are not an independent sample of observations, but also time-dependent. If we change the order of the samples, we are distorting the time series x(n), so the order of the observations must be kept during the processing.

Dynamic networks seen as dynamical systems For a given a certain dynamical network we will suppose that its input is time-dependent, $I(n) = [I_1(n), \dots I_m(n)]$. Whenever the vector $I(n)$ is changed, the network will need a certain amount of time to reach a stable output, if it does. Hence, it seems natural to introduce two scales of time for the network: one associated with the external stimulus, and the other describing the time-response for the network (inertia with respect to that stimulus). To make the training feasible, it is necessary to assume that the changes in the external inputs are slower than the time that the network needs to reach an a steady state [42]. From now on we will assume this is the case, so the network has “enough” time to evolve after a change in its inputs. For example, when a fixed-point learning is performed (see 2.1.5 on page 34), we let the network evolve until an equilibrium point is met, so we can think that $I(n)$ is held constant during that period, and such equilibrium point exists. The training algorithm will continuously adjust the weights in order to reach an equilibrium point and at the same time have the values as close as possible to the desired ones. In this process we do not need to know the output values from the hidden layers. The sub-problem related with the search of attractors (equilibrium points) “as close as possible” to the desired values can be treated as a minimization of a certain error function, that will depend on the instantaneous output value and the desired ones. On the other hand, all the networks treated in this work comply that, for an arbitrary neuron i:

$$y_i(n+1) = f[\sum_{j \geq i} w_{ij} y_j(n) + \sum_{j<i} w_{ij} y_j(n+1) + I_i(n+1)] \quad i = 1, \dots N$$

where $I_i(n)$ is the exogenous input for neuron i, with the agreement that $I_i(n) = 0$ if that input does not exist.

To verify this condition it suffices to see that

1. If $w_{ij} = 0 \quad \forall j \geq i$ a feedforward network is obtained (such as TLFNs).
2. The MLP is obtained making $w_{ij} = 0 \quad \forall j \geq i$ and $I(n) = 0$.
3. The general case corresponds to a totally recurrent network (see 2.1.4.2).

So, for the neuron i we can write:

$$y_i(n+1) = G_i[y_i(n), w(n), I_i(n)] \qquad \textbf{(Eq 2.5)}$$

when we hold the inputs $I_i(n)$ and vary n. We also minimize the error via a gradient descent, so:

$$\Delta w_{ij}(n) = -\rho \frac{\partial E}{\partial w_{ij}} \qquad \textbf{(Eq. 2.6)}$$

(or other equation, corresponding to the weight update). Then, if we take the outputs $y_i(n)$ for each neuron and the weights $w(n)$ as the network states, equations Eq 2.50 and Eq. 2.60 fully describe the behavior of the network as a dynamical system. In vectorial form:

$$\mathbf{y}(n+1) = G[\mathbf{y}(n), \mathbf{w}(n), \mathbf{I}(n)]$$

This system of non-linear equations can not be analytically solved because of its complexity.

If we also consider the training dynamics it is required to add that

$$\Delta w(n) = -\rho \nabla E \big|_{w=w(n)} \quad \textbf{(Eq. 2.7)}$$

Equation Eq. 2.70 represents a gradients system that can converge towards a stable state (see [42]).

The convergence conditions for the output $y_i(n)$ towards an attractor point have been derived for general cases, though particular cases have been studies as well (for example, imposing that the weight matrix to be symmetric in a Hopfield's network).

2.1.4.2 Recurrent architecture

We define a ***recurrent network*** to be a network with one or more closed loops, this is, which topology can be represented by a directed graph with cycles, or looking the data, those networks where the output depends on the previous outputs [30]. The network in figure 2.7 is hence recursive.

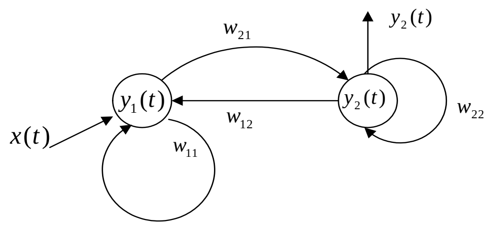


Figure 2.7

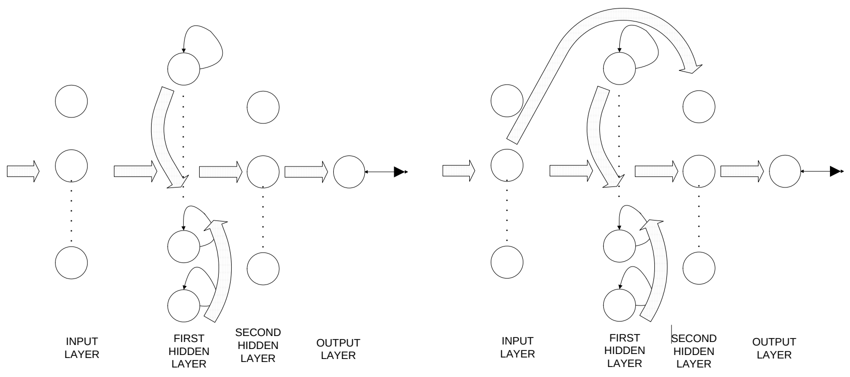
In a recurrent network, neurons can be of input, of output or of both type simultaneously, and the desired output values are given for an arbitrary set of neurons for predefined times.

Totally and partially recurrent networks

To start, the most intuitive definition is the one used by Neurosolutions:

"*A totally recurrent network connects the first hidden layer to itself with at least one recurrent connection. A partially recurrent network adds to the totally recurrent at least one forward connection from the input layer towards the next layer of the first hidden layer.*"

We can see the idea graphically in figure 2.8



Totally (right) and partially (left) recurrent networks
Figure 2.8

Príncipe gives an alternative definition of a totally recurrent network [69]:

If a network verifies that

$$\begin{cases} Net_i(n+1) = \sum_{i\leq j} w_{ij}y_j(n+1) + \sum_{i>j} w_{ij}y_j(n) + I_i(n+1) \\ y_i(n+1) = f(Net_i(n+1)) \end{cases}$$

where I_i is the external input connected with neuron i if there exists such external input, or 0 otherwise, the neurons are labeled respecting the definition of "feedforward" network at instant n+1 and we define a delay in all "feed-back" connections (that go from a higher to a lower neuron), then we say the network is ***totally recurrent***.

For example, the network from figure 2.9 is totally recurrent:

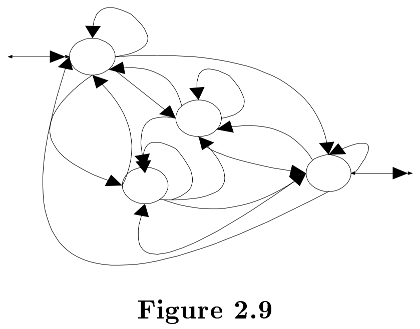


Figure 2.9

The reader is referred to [65] for a strict and formal definition for totally and partially recurrent networks.

2.1.4.3 Static vs. dynamic networks

In the case of static networks we do not have the concept of ***stability***, or better, they are always stable. Dynamic networks can be either stable or not. Stability is here understood in a uniform sense (2.3.1): the network response to a bounded input must be a finite value (bounded). This does not occur for instance in a context neuron: if $\tau > 1$ it turns ***unstable*** [51].

Static networks only have a "***long term memory***" that is defined by its weights: the information from the data set is translated during the training process into weights, using learning rates. These networks contain a historical repository associated with its memory, though they are not capable to distinguish the temporal relations between the data, because the collected information is collapsed in the weight values. Dynamic networks, instead, have a "***short term memory***", as in the case of the time lagged feedforward networks (TLFNs), or recurrent networks, that are sensitive to the sequence in which the information is presented. They also have weights, consequently long term memory, but unlike the static networks their weights capture the differences in the order of the data set within an observation window.

The short term memory structures allow to convert dynamic phenomena into static, handling a set of D successive data over the time as a point in a D-dimensional space [69].

2.1.5 On the training of recurrent networks

Error criteria A key difference between the weight adaptation in dynamic and static networks is that in the former, the local gradient is time-dependent. Indeed, the optimization problems are also different, because in dynamic networks we are interested in the overall network performance within a certain period instead of instantaneously. The most common error criterion for dynamic networks is called "***trajectory learning***", where the error is summed over all steps from $n = 0$ to $n = T$:

$$E = \sum_{n=0}^{T} E_n = \sum_{n=0}^{T} \sum_{m=1}^{N} \varepsilon_m^2(n)$$

where E_n is the instantaneous error and N the number of output neurons (the sum over all patterns was here omitted for the sake of simplicity). The time T is the length of the trajectory, related with the time-length of the temporal pattern of interest. As a common rule, the trajectory has exactly the same length of the pattern to be learned, though it could be higher. The error function is then found over a period, and we attempt to adapt the weights in order to minimize E_n over the whole time interval. Particularly, if the dynamical system represented by the network reaches the stationarity (in the sense that an output can be statically associated with an input pattern), the following error function can be used

$$E = \textstyle\sum_{m=1}^{N} \varepsilon_m^2 \qquad \textbf{(Eq. 2.8)}$$

This error criterion is used when the network is trained using a ***fixed-point learning*** [69].

Learning paradigms for dynamic networks

Point-fixed learning In this training method (only used for recurrent networks) the idea is to make correspond a static input with a desired, static output. The error function is exactly given by equation Eq. 2.8. The system must be reach a steady state. The stable value returned by the network (a value that does not change) for a certain input pattern, after reaching its steady state is, by its definition, a ***fixed-point*** ([69]). In that moment the network behaves statically. Therefore, after reaching that fixed-point, the resulting output for an input pattern can be compared with the desired output, and the error can be back-propagated.

The "back-propagation" algorithm can be further generalized to implement this kind of learning, in the following manner:

- *Present an input pattern and hold it in the input until the output is stable (it does not change)*
- *Compare the output with the desired output value, compute the error and back-propagate it through the dual network, in the same way as in static case. It is mandatory to present the error to the dual network and hold the input until the propagated error is stable.*
- *Use any desired procedure to update the weights in order to find the minimum for E. For instance, a raw gradient descent would return*

$$\Delta w_i = -\rho \frac{\partial E}{\partial w_i}$$

- *Repeat steps 1 to 4 for the next pattern.*

It must be taken into account that:

1. This training is convergent provided the original network is stable, namely, the network outputs can be stabilized. If the original network is stable, the dual onewill inherit this stability property [69]. However, the necessary time for both networks to reach a steady state can be different and even change with the iterations. Sometimes, the original and dual networks take more time to get stable as soon as the algorithm gets closer to the solution [69].
2. The learning rates must be chosen so to have a slow training. This is to allow to have a learning dynamics slower than the network's one. If this condition is not met, a family of networks are trained instead of one, and there is no certainty to get a convergent process.
3. The static back-propagation is a special case of fixed-point training, with null stabilization times.
4. A tentative initial value for the stabilization time in the fixed-point case might be 100, that is, to hold the input pattern one-hundred times more than in the static case [69].

Learning trajectories We define a ***trajectory*** as a sequence of patterns together with their respective outputs through time. in this case, we wish to train the network such as the outputs follow a specific temporal sequence. Note that we are not only interested in the last instant, but the intermediate outputs as well (the trajectory). The learning of trajectories can be implemented using RTRL or BPTT indistinctly [69]. A variant on the learning of trajectories is to store the inputs, outputs and neurons weights for the last part of the trajectory (the most recent part), obtaining a ***truncated BPTT (or BPTT(h))***. The reader is referred to 2.4 or [30] to have an overview of truncated BPTT.

2.1.5.1 Difficulties to train recurrent networks

There are several circumstances that make a given task irresolvable with a recurrent network. In a first place, it is possible that the chosen neural model is not adequate for that task, something hard to judge beforehand. In the second place, even if the model is adequate (including the number of neurons and the input/output representation), it can occur that the employed training method is not capable to return a correct set of weights. There are mainly two reasons for this drawback: the presence of local minima and long term dependencies. There is also the risk to turn the network unstable. "*Training recurrent networks with BPTT (or RTRL) is yet more an art than a science, being TLFN networks easier to be trained, and they should be the point of departure for every solution*" [69].

- ***Local minima***

The error function E defines a multidimensional surface called ***error hyper-surface***. Normally, the error hyper-surface presents a global minimum (possibly more than one "global" minimum where E assumes the same value, because of symmetries in the network) and several local minima, that may not correspond to a correct solution of the problem. These local minima are consequence of the high dimensionality of the search space, and represent the major cause of concern, given that all the learning algorithms tend to be trapped in them, specially naive local searches as the raw gradient descent. Anyway, the local minima problem is not specific of the recurrent neural networks, and affects nearly all the neural network models. Additionally, the error surface from dynamic networks tend to have very narrow valleys, and the learning rate must be carefully controlled for the training stage. The algorithms that use adaptive learning rates are the most appropriate here. In particular, the neural network simulator here chosen allows the manual and/or automatic rate adjust.

- ***Stability***

Recurrent networks can turn unstable during their training. The non-linearity in the neurons will avoid a "network explosion", keeping the outputs bounded, though the neuron outputs can oscillate between extremal values. Normally, to watch that it does not happen, and to restart the training in the unfortunate case, represent the only way-out [69].

- ***Error flow and vanishing gradient***

Consider the network flow between neurons i and j. In this case the resulting error at instant t in neuron i, represented by $\varepsilon_i(t)$, "travels" back with time (assuming we are using BPTT) until neuron j is reached at instant $s \leq t$. This error signal tries to "modify the past" hence outperforming the last result in the present with respect to the desired output $\mathbf{d}(t)$. It can be proved ([34] and [70]) that the error flow backwards (reaching neuron j) is either increased without bound and neuron j gets an error value that can make oscillate the weights, turning the whole training unstable, or the error exponentially decreases with the distance between t and s, that is, the error flow vanishes and the learning is no more possible. This turns the training both hard and slow.

This degradation problem of the error information (or equivalently, the local gradient) through non-linearities has been called the ***vanishing gradient problem***, and is linked with the ***long term dependencies problem***. From a dynamic viewpoint, if a recurrent network, as a dynamic system, has an hyperbolic attractor and is being trained with a learning gradient-based method, either the network is not able to learn when the inputs have noise or cannot determine long-term dependencies between the data [30]. In order to face these drawbacks, a training with the extended Kalman filter, second-order descend methods can be used or even to change of topology (for example, to LSTM networks). There is no mathematical solution to this problem at hand yet, even though new topologies to avoid it (LSTM networks to "avoid the vanishing gradient"), and the use of other training methods (using deKf, see 2.5 on page 70). The long-term dependence problem means that when a recurrent network is trained, the gradients can be highly attenuated, and the long-term relations will be learned with difficulty, if they can. However, if structures with linear memories are employed (LSTM networks), then the gradients are not attenuated when they are back-propagated. A mixture of non-linear neurons and neurons with memory, as in a TLFN, can lead to better results than totally recurrent topologies [69].

- ***The long-term dependence problem***

Practically all training algorithms for recurrent networks face big troubles (sometimes not surmountable) to track the information over a sequence, specially when the time between the presentation of an input and its corresponding affected output is relatively large (normally higher than 10 time-steps), because of the vanishing gradient. Formally:

Given a data source that generates a sequence $s(1), \dots s(t_u), \dots, s(t_v), \dots$ we will say that there exists a ***long-term dependence*** *between the value at instant t_v and the one at instant t_u, and we will denote it by $s(t_v) \multimap s(t_u)$, if the following conditions are met:*

1. *the value of $s(t_v)$ depends on the value $s(t_u)$*
2. $t_v \gg t_u$
3. *there is no t_w with $t_u < t_w < t_v$ such that $s(t_v) \multimap s(t_w) \multimap s(t_u)$*

As seen before, the training gradient-based algorithms for recurrent networks are usually unable to represent long-term dependences given that the current network output is insensitive to old inputs [34].

2.1.5.2 Back-propagation through time (BPTT)

There are several modifications to the "back-propagation" algorithm for dynamic networks: back-propagation through time (BPTT), recurrent back-propagation, BP with stationarity (or Pearlmutter's method) and real-time recurrent learning (RTRL). Each one is associated with a special behavior of the system that generates the inputs with time and how the network is trained. We will see BPTT next.

BPTT and RTRL are both based on ***unfolding*** the network through time, and though this is widely general from a theoretical viewpoint, it is limited in practice by the number of neurons generated by unfolding the network (for example, if we want to learn a length-3 trajectory, the number of neurons is multiplied by 3).

To be more specific with the unfolding process consider a recurrent network NN that must be trained from a certain time n_0 until a time n. Let NN * the feedforward network obtained by unfolding NN through time. NN * is built from NN in the following manner (so it always exists):

- *For each time step within the interval $(n_0, n]$ the network NN^* has a layer with k neurons, where k is the number of neurons of NN*
- *In each layer of the network NN * there is a copy of each neuron of the network NN*
- *For each instant $l \in [n_0, n]$, the connection from neuron i in layer l towards neuron j in layer $l+1$ of the network NN * is a copy of the corresponding connection from neuron i towards neuron j in NN .*

Regarding the high computational costs required to handle real applications with back-propagation through time, they are usually considered additional algorithms that do not unfold the network, such as some implementations of BPTT(h) (truncated back-propagation, see 2.4 on page 58 and [30]).

2.1.5.3 Back-propagation (BP) vs. BPTT

If the network is either static or dynamic, but feedforward (such as the TDNNs) and the desired output is known for every instant, it can be proved that BPTT is equivalent to use an ordinary BP and sum the local gradients multiplied by the inputs along the specified period [69]. However, there are cases where the output is only known in the last instant of that period. In those cases we have to use BPTT since we do not have an explicit expression for the error at each instant. If the network can be trained by BPTT in some parts and by a common back-propagation in others (such as in the case of the TDNNs), it is recommended for simplicity to apply BPTT through the whole network [69].

2.1.5.4 Recurrent BP

Let us suppose that the network under study (where time varies continuously) is recurrent and presents an equilibrium point (stationarity) for its inputs in the sense that after training, the same outputs are obtained for the same inputs no matter the instant. In [34] some necessary conditions are discussed in order to reach the steady state. In this case, a variation for the back-propagation algorithm can be proposed, called "recurrent back-propagation" (see 2.4 on page 58).

2.1.5.5 Recurrent time-dependent BP - Pearlmutter's method

Let us now consider a network that does not reach an equilibrium point for a set of fixed inputs, so the network output is always a function of the continuous time. Further, assume the output of the neurons in the network comply that

$$\tau_i \frac{dy_i}{dt} = -y_i + f(Net_i) + x_i(t) \quad i = 1,\ 2, \ldots N$$

where the τ_i are independent of time and constant, and $x_i(t),\ y_i(t)$ are continuous functions representing respectively the input to and output from the neuron i, and N is the number of neurons in the network. In this case, working on a closed time interval we can essay an "off-line" learning, called Pearlmutter algorithm. This algorithm basically uses a generalization of the sum of squares as its error function:

$$E = \frac{1}{2} \int_0^{t_1} \sum_l [d_l(t) - y_l(t)]^2 \, dt$$

The algorithm performs a descent search on the weights and simultaneously, other descent according with the numbers τ_i (see [29] and [60]).

2.1.5.6 Real Time Recurrent Learning (RTRL)

In the discrete-time case, an "on line" training can be performed (in real time, while the network is processing the presented values at the input), that does not imply major computational challenges. This method has been widely employed (see 2.4 on page 58 and [30]).

This training makes sense when the weights change slowly when compared with the changes of the output respect the inputs. We can say that this algorithm is local in time but not in space (in the topology), so in some way it is dual to the BPTT. Indeed, if we consider the weight adaptation equations we get that (see 2.4 on page 58):

$$\begin{cases} \Delta w_{ij}(n) = \rho \sum_{p=1}^{N} [d_p(n) - y_p(n)] \frac{\partial y_p}{\partial w_{ij}} \\ \Delta w_{ij} = \sum_{t=t_0}^{T} \Delta w_{ij}(n) \end{cases}$$

where n varies with time, p, i and j over the number of neurons. It can be appreciated that the gradient with respect to a given weight (in a certain instant) does depend on the derivative with respect to other weights and the error in the other neurons. That is the reason why the algorithm is called local in time but not in the topology.

2.1.5.7 BPTT vs. RTRL

The following table summarizes the main characteristics of both methods [69]:

	RTRL	BPTT
Required storage	$\boldsymbol{O}(N^3)$	$\boldsymbol{O}(NT)$
Number of required operations	$\boldsymbol{O}(N^4T)$	$\boldsymbol{O}(N^2T)$
Local in space (topology)	NO	YES
Local in time	YES	NO

being N the number of neurons in the network and T the size of the trajectory

2.1.5.8 Jordan and Elman networks

While TLFN networks are able to implement any mapping $I \rightarrow O$, there are some cases where such mapping is out of the scope for a reasonable size of a focused TLFN. Jordan and Elman proposed simple networks based on context neurons and recurrences, easy to train and able to learn such mappings by means of small topologies [69]. Their schemes are presented in Figure 2.10:

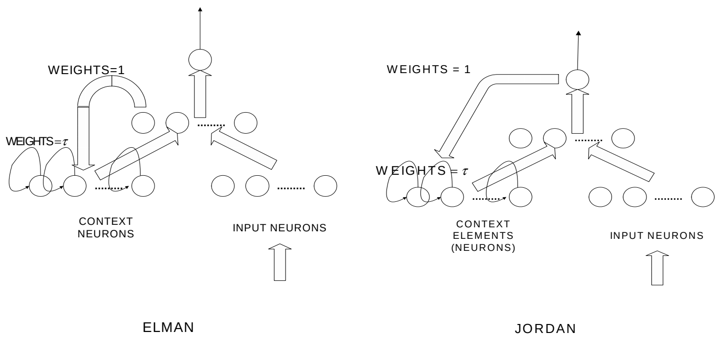


Figure 2.10

The thick arrows represent all the possible connections. Both networks have the feedback parameter, τ, fixed, and there is no recurrence between the input and the output. They can be roughly trained using the ordinary BP (and this is the case of the neural network simulator chosen here, [69]). In principle these networks are more efficient than focused architectures (see 2.1.5.12 on the facing page) to code temporal information.

Both networks have been used for sequences recognition, and because of this they are denominated ***sequential***.

2.1.5.9 Memory

These networks use a memory structure called “feed-back” memories. The neurons that possess those memories are called ***context neurons***. The memory of the context neurons is obtained using a “feed-back” loop, which corresponds to find the output summing the past input values multiplied by τ (time constant):

$$y(n) = \sum_{i=0}^{n} x(n)\tau^{n-i}$$

This can be graphically represented by:

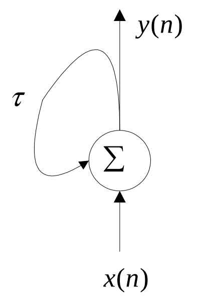


Figure 2.11

Observe than an impulse in the input x (such that $x(0) = 1,\ x(n) = 0$ if $n \neq 0$) will generate an output series $y(n) = \tau^n$. This is the reason why context neurons are called memory neurons: they “remember” past events. The time constant τ (chosen once the network is designed) should verify $0 < \tau < 1$, otherwise, the neuron response will progressively turn unstable.

2.1.5.10 Relation with AR and MA models

In linear systems, the use of previous values in the input creates the models known as Moving Average (MA), whereas the use of historic values of the output defines Auto-Regressive (AR) models. In the case of non-linear system, such as the ordinary neural networks, those two topologies correspond with NMA and NAR (the N comes from non-linear) respectively. The Jordan network is a particular case of a NAR model, whereas a special case of NMA is obtained when the context neurons are fed with the input layer. The Elman network does not have a counterpart in the theory of linear systems [68].

2.1.5.11 Computational power of recurrent systems

In a wide sense, one of the most remarkable computational capacities of the general recurrent networks is given by Siegelmann and Sontag's theorem (1991, cited by [30]): ***Every Turing machine can be simulated by a fully connected recurrent neural network with sigmoid transfer function.*** Finally, other interesting conclusion related with these networks is the ***Funahashi and Nakamura's theorem.*** Funahashi and Nakamura proved that the output of a recurrent neural network with enough hidden neurons (with a continuous-time domain) can approximate as exactly as desired any trajectory in the space state (that is, the behavior of any dynamical system). ***In other words, recurrent neural networks are universal approximators for deterministic dynamical systems*** [29].

2.1.5.12 Time Lagged Feed-forward Networks" (TLFNs) and "Time Lagged Recurrent Networks" (TLRNs)

TLFNs are non-linear networks represented by feedforward arrays of memory elements and non linear neurons. The main advantage is that they share valuable properties with feed-forward networks (for instance, stability) and additionally capture the present information from temporal input data. A TLFN is ***"focused"*** if its memory elements are restricted to the input layer ([69] and [30]). When the memory is distributed over the network, we will refer to a ***distributed TLFN*** [30] or ***"non-focused"*** [69]. When the memory elements are of type "delay line" and exist only in the input layer we have a "time delay neural network" (TDNN). Here we will generalize the definition including the case of "non-focused" TDNN (that is a "delay-line" TLFN with delay line memory elements, focused or not). When the elements with memory present an architecture with cycles (they are recurrent networks) we have the ***"time lagged recurrent networks" (TLRNs).*** These can also be focused or not.

2.1.6 Design issues

We will pay particular attention to several design issues such as the network topology, transfer functions to use, etc. in order to improve the obtained solution (smaller network, better generalization capacity, easier to train, etc.). Some of these considerations were taken into account in our experimental development.

To start, Haykin ([30]) proposes several heuristics to improve the performance of the BP algorithm:

- Heuristic 1: Each parameter of the error function that can be tuned through the learning phase must have its own learning rate. For example, each weight must have its own learning rate.

- Heuristic 2: Each learning rate should be able to vary from one iteration to another.
- Heuristic 3: When the derivative of the error function with respect to a given weight has the same algebraic sign for several consecutive iterations, the learning rate should be increased.
- Heuristic 4: When the derivative of the error function with respect to a given weight alternates during several consecutive iterations, the learning rate should be reduced.

It is worth to have in mind that the use of variable learning rates modifies the back-propagation algorithm, and the weight update is no more with a gradient search, but based on a) the partial derivatives of the error function with respect to the weights, and b) estimations for the curvature of the error surface at the current point of operation, with respect to some of the weights. Even though, the heuristics and the corresponding modifications to the algorithm have proved to be useful [30].

Now we will go through general concerns over other aspects that can affect the behavior of the algorithm.

2.1.6.1 Output function

The calculations can be greatly simplified when the logistic or hyperbolic tangent functions are used, mainly because their derivatives f' can be expressed in terms of the function f :

for the logistic:

$$\begin{aligned} f(Net) &= \left(1 + e^{-\lambda Net}\right)^{-1} \\ f'(Net) &= \lambda f(Net)[1 - f(Net)] \end{aligned}$$

and for the hyperbolic tangent:

$$\begin{aligned} f(Net) &= Tanh(\beta Net) \\ f'(Net) &= \beta[1 - f^2(Net)] \end{aligned}$$

being *Net* the input for the respective neuron:

$$Net_j = \sum_{i=1}^{p} w_{ji} x_i$$

2.1.6.2 Convergence

Normally the current weights that reach the output layer are used for the calculus of the variation of weights that reach the hidden layer, though generally a better correction is performed if the values of the updated weights from the output layer itself $w_{lj}^{new} = w_{lj}^{current} + \Delta w_{lj}$ are used instead. From a computational viewpoint, it should be taken into account that it means an additional cost coming from the re-calculation of y_i and $f_o'(Net_i)$.

If a solution from the current parameters can no longer be found (there is no convergence), we can try again changing some of them, taking a new initial weight set and/or use more hidden neurons, because this number plays its role in the learning effectiveness. What is more, if the learning is fast enough, the number of hidden neurons can be reduced, in order to optimize the computational resources. A trade-off between stability, training performance and number of hidden neurons should be met. Rumelhart (cited by [29]) developed a pruning method to delete hidden neurons with scarce activity during the learning process (neurons which their weights change little or nothing during the learning process, see 2.8 on page 85).

Bear in mind that the surface error should be explored with small increments in the weights, because we only have local information of this surface and we are not aware whether the local minimum is near or not. Using big increments can lead us to "skip" that point, oscillating around it without reaching the target. On the other hand, an extremely small increment can lead us to a very slow convergence. Normally the learning rate ρ is within 0.05 and 0.5. A good practice is to let ρ increase when the network error is reduced during the learning stage, having in mind the variation should not be big enough [33].

Although the artificial unit input to all neurons was not represented, the activation threshold θ also plays its role during the learning process, and its weight will be adjusted accordingly [33].

2.1.6.3 The training set

There is no general rule to predict the number of training pairs $(\mathbf{x}, \mathbf{d})$ to achieve a satisfactory result with an arbitrary network topology with sigmoid transfer functions. For that reason, it is better to have as many pairs as possible. Recall that a subset could be used for verification (e.g. in cross-validation, see 2.2 on page 45), what makes the number of necessary pairs for training to be increased. Baum and Haussler [5] found that in feedforward multilayer networks with Heaviside input (unit step from a certain instant) and desired binary values in $-1, +1$, the relation $d_{VC} \leq 2Wlog_2(eM)$ holds, being d_{VC} the Vapnik-Chernovenkis dimension ([10], [30]), W the number of weights in the network, M the number of neurons and e the base of natural logarithm. They derived as corollary that if a network is trained with $N \geq \frac{W}{\varepsilon}log_2(\frac{M}{\varepsilon})$ patterns with a correct classification of $100(1-\varepsilon/2)$ percent of them, there is high probability that the network correctly classifies $100(1-\varepsilon)$ percent of the remaining future observations, generated with the same probability density function [10]. They also conjectured that similar bounds hold for other transfer functions commonly used. In fact, some authors use these bounds with no more theoretical considerations [81]. For example, if If we wish to have an error classification of 0.1, we should consider around $10 * W$ training pairs [81].

The additive noise at the input helps to speed-up the convergence, even though the network will not operate finally with input noise [29].

Additionally, the number of necessary input patterns is constrained by certain characteristics of the modeled data [81] :

- The ***intrinsic dimensionality*** of the data, which is the number of independent variables at the input (whose values do not depend each other). A possible heuristic is to increment the size of the training set in a factor of 10 for each independent variable. In a time series, the intrinsic dimension is given by the embedding dimension [81].
- The ***resolution*** or granularity of the data, which is the number of divisions in each dimension (input variable). For example, if we wish to have a measure of a distance in cm. or mm. If the division scale is higher (less granularity), we will need less observations for that variable, and it will be less likely to model data noise [22].
- The ***probability distribution/probability density*** of the data: we must be sure of having enough data to cover each possible state that the system to be modeled can reach. Therefore, if we know the network input $\mathbf{x}$ (corresponding to a system state) occurs with probability $P(\mathbf{x})$, we should have at least $1/P(\mathbf{x})$ training patterns to be relatively sure that the input includes some sample of $\mathbf{x}$.
- The ***noise*** and the ***quality*** of the data: if the data is mixed with very high noise or has a bad quality (for example for being scarce in number or it not representing all

the possible inputs), a much higher volume of data could be required, since a big part of the data will be discarded.

2.1.6.4 Network dimension

In general, three layers (input, hidden, output) are enough. In fact, it can be proved that no more than one hidden layer with an adequate number of neurons is needed to solve any interpolation or classification problem (see 2.10 on page 108).

The sizes of the input and output layers are given by the nature of the problem (application).

The size of the hidden layer(s) should be carefully studied because:

1. The number of hidden layers should be such that the information available in the training set could be stored, and at the same time not as high as to loose the generalization capacity of the network.
2. The number of neurons in the hidden layer must be lower than the ones in the input, if any compression is going to be done ("feature extraction" or reduction of the dimensionality) of the input data.
3. In [81] the number of hidden neurons is suggested not to be more than twice the number of the input. The support of this rule can be found in Kolmogorov's theorem on function approximations (see 2.10).
4. Each hidden neuron consumes computational resources when the network is simulated, and a trade-off between learning performance and number of hidden neurons should be met.

2.1.6.5 Weights and learning parameters

It is recommended to choose the initial weights as small values uniformly picked at random, for instance within the open interval $(-1, +1)$; the same is applied for θ_i [81]. On the other hand, in [20] suggest to have the weights within $(-0.5, +0.5)$. Another heuristic suggested in the literature (for instance, in [29]) is to choose the initial weights within the interval $(-1/\sqrt{f}, +1/\sqrt{f})$, being f the number of connections that enter neuron i (or "***fan-in***").

Kolen and Pollac [39] discovered, for feedforward networks learning the XOR function, that there exists a structure representing convergence as function of the initial weights, similar to fractals, where there are regions with high sensibility in the weight space, so a small difference in the initial weights can lead to very different learning curves. That shows how sensitive is the retro-propagation to the choice of the initial weights, displaying a chaotic behavior. This fact could be explained by the existence of several minima of the error function, the non-zero learning parameters (rates and weights) or non-linear deterministic nature of the gradient descent approach.

The choice of the learning rate is important: in general within the interval $(0.05, 0.25)$, possibly varying with time [20] (see 2.1.2 on page 24).

Network Paralysis: if a sigmoid transfer function is used (such as the hyperbolic tangent or the logistic), we should be careful that the magnitude of the weights do not increase beyond a bound because otherwise the neurons will operate with high input values, and in that case, since the derivative of the output function is close to 0, no weight updates are produced, and the network halts its learning phase (it gets paralyzed).

2.2 Advanced techniques

2.2.1 Improvements and variations to the basic BP algorithm

Here some variations for the basic BP algorithm are covered, for both the modifications of its learning parameters and the selection of the error function.

2.2.2 Second-order methods and moments

Moments This technique consists of adding a ***moment*** term in the right part of the weights update equations

$$\Delta w_{lj} = \rho_o (d_l - y_l) f_o^{'}(Net_l) z_j \text{ (output layer)}$$

$$\Delta w_{ji} = \rho_h [\textstyle\sum_{l=1}^{L} (d_l - y_l) f_o^{'}(Net_l) w_{lj}] f_h^{'}(Net_j) s_i \text{ (hidden layer)}$$

In this way we speed-up the gradient descent when the derivatives of the error function have the same sign in two consecutive steps, avoiding an oscillation of it with each change in the sign of $\frac{\partial E}{\partial w_i}$ [30]. The corresponding equations are[7]:

$$\begin{cases} \Delta w_i(t) = -\rho \frac{\partial E}{\partial w_i} + \alpha \Delta w_i(t-1) \\ \Delta w_i(t) = w_i(t) - w_i(t-1) \end{cases} \quad \textbf{(Eq. 2.9)}$$

where α *is the **moment rate***, generally $0 < \alpha < 1$ (in the general case, $0 < |\alpha| < 1$, though it is not usual to take negative values).

The moments method is a way to increase the effective learning rate in regions in which the error surface is nearly plain, while the learning rate is close to ρ (with $0 < \rho \ll 1$) in regions with high fluctuations. Indeed, if we use a recurrence in N steps (N arbitrarily chosen) we can re-write Eq. 2.9:

$$\Delta w_i = -\rho \sum_{n=0}^{N-1} \alpha^n \frac{\partial E}{\partial w_i(t-n)} + \alpha^N \Delta w_i(t-N)$$

If the search point is taken in a nearly plain region, then $\frac{\partial E}{\partial w_i}$ will be approximately constant in each step, and the previous equation can be approximated by

$$\Delta w_i \approx -\rho \frac{\partial E}{\partial w_i(t)} \sum_{n=0}^{N-1} \alpha^n = -\frac{\rho}{1-\alpha} \frac{\partial E}{\partial w_i(t)}$$

when $0 < \alpha < 1$ and the number of steps N is high enough.

As a consequence, for plain regions, the moment term leads to increase the variation rate of the weights in a factor $1/(1-\alpha)$.

Static and dynamic moment The added moment is said to be static if its rate α does not depend on the stage of the learning phase in which one is. The moment is adaptive or dynamic whenever the moment rate changes with the time.

[7]Here we consider the weights as components of a vector, without regarding the connected neurons.

Newton's method The second-order searches (such as Newton's method) are based on a quadratic approximation $E_2(\mathbf{w})$ of the function $E(\mathbf{w})$, that is, in $E_2(\mathbf{w})$ are used the first three terms from the Taylor's series of $E(\mathbf{w})$ [29]:

$$E_2(\mathbf{w}^{current} + \Delta\mathbf{w}) = E(\mathbf{w}^{current}) + \nabla E(\mathbf{w}^{current})^T \Delta\mathbf{w} + \frac{1}{2}\Delta\mathbf{w}^T \mathbf{H}(\mathbf{w}^{current})\Delta\mathbf{w}$$

being $\mathbf{H}$ the Hessian matrix $\mathbf{H} = \left\{H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j}\right\}$

Minimize E will be roughly the same to minimize E_2(w). Particularly if we wish to minimize $E_2(\mathbf{w}^{current} + \Delta\mathbf{w})$, we must solve

$$\nabla E_2(\mathbf{w}^{cirrent} + \Delta\mathbf{w}) = 0$$

which is achieved making [29]

$$\Delta\mathbf{w} = -\left[\mathbf{H}(\mathbf{w}^{current})\right]^{-1} \nabla E(\mathbf{w}^{current}) \textbf{ (Eq. 2.10)}$$

The iterative weight updating process obtained using Eq. 2.10 is called ***Newton's method***.

The computation of $\mathbf{H}^{-1}$ is computationally intensive (it needs $O(W^3)$ operations, being W the number of weights), so some researchers (Le Cun and Becker, cited by [29]) proposed an approximation that discards the elements outside the diagonal of $\mathbf{H}$, and calculates

$$\Delta w_i = -\frac{\partial E}{\partial w_i}\left(\frac{\partial^2 E}{\partial w_i^2}\right)^{-1}$$

2.2.3 Other stopping criteria

"Cross validation" This alternative (or complementary strategy) to improve the generalization capacity of the network is based on empirical results ([50], Wieigend, Hergert and others, cited by [29]) and it is basically a criterion to stop training. In training simulation of feedforwawith an additional termwith an additional termrd networks using back-propagation with noisy data, it has been found that the generalization error monotonically decreased towards a minimum, and then started to increase even if the training error still went on decreasing.

Graphically:

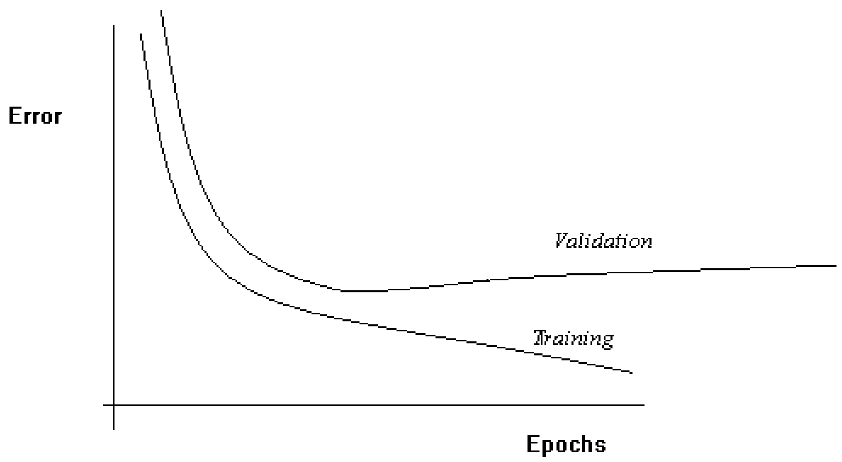


Figure 2.12

To summarize, when the training is performed with noisy data, the excessive training leads to overfitting, whereas a partial training can lead to a better approximation of the function in the sense of a better interpolation, and possibly, better extrapolation. Wang and others (cited by [29]) gave a formal justification of the generalization improvement when the learning stage is stopped before reaching the global minimum. They proved that there exists a critical moment in the training process where the network generalizes in the best possible way, and after which the generalization error is increased. Therefore, a feasible strategy to improve the network generalization capability for networks of sub-optimal size[8] is to avoid the overfitting by means of a careful control of the validation error during the training process, and stopping the training just before the error starts to grow. This strategy is known as "***cross validation***". In this method, the whole available data set is divided into two parts: a training set and a validation set. The training set is used to determine the network weights. The validation set is used to decide when to finish the training process. The training proceeds while the performance in the validation set is still improving. When there is no further improvement, the training is stopped[9].

This criterion requires an extensive data set, being inappropriate for applications where data are scarce [29]. In those cases, there are variants of it, such as the "***multi-fold cross validation***", where the N available patterns are divided into $K > 1$ subsets and the model is trained for all subsets but one, being the excluded subset used for the error validation. The process is repeated for the K subsets, using in each turn a different subset for validation. The model performance is evaluated averaging the error validation of the K tests [30].

Finally, the cross validation criterion permits to choose the network model that fits better with the data and provides a better generalization capacity, and in that is related with the MDL and the AIC. It can be proved that the use of cross validation is asymptotically equivalent (when the data set is increased without bound) to the Akaike's index [69].

Other error functions [criteria] An error function should respect the conditions that define a distance: the resulting error from an output value $\mathbf{i}$ obtained instead of a desired output $\mathbf{j}$, $d(\mathbf{i},\mathbf{j})$, should be such that

$$\begin{array}{ll} 1) & d(\mathbf{i},\mathbf{j}) > 0 \quad \forall\,(\mathbf{i},\mathbf{j}) \qquad \mathbf{i} \neq \mathbf{j} \\ 2) & d(\mathbf{i},\mathbf{i}) = 0 \\ 3) & d(\mathbf{i},\mathbf{j}) = d(\mathbf{j},\mathbf{i}) \\ 4) & d(\mathbf{i},\mathbf{j}) \leq d(\mathbf{i},\mathbf{h}) + d(\mathbf{h},\mathbf{j}) \end{array}$$

with $\mathbf{i} = \mathbf{i}(\mathbf{w}), \quad \mathbf{w} \in \mathbb{R}^{\mathbf{n}}, \quad \mathbf{i},\mathbf{j} \in \mathbb{R}^{\mathbf{m}}$ and also, if a derivative-based training will be employed (as in the case of a gradient descent), the gradient $\nabla d = \left[\frac{\partial d}{\partial w_1}, \frac{\partial d}{\partial w_2}, \cdots \frac{\partial d}{\partial w_n}\right]$ must be defined, and continuous with respect to $\mathbf{w}$. Conditions 3) and 4) are not mandatory for some error functions, such as the relative entropy.

Minkowsky error functions They are a family of parametric functions with a parameter $r \geq 1$, that can be seen as a possible generalization for the MSE. Its general form is[10]:

$$E(\mathbf{w}) = \sum_{j=1}^{n}\sum_{i=1}^{m} |(d_i)_j - (y_i)_j|^r$$

[8]It is proved that stopping the training via CV has the same effect in the network generalization capacity than changing the network size, for a given training set[84].

[9]Software tools help us to determine a stopping rule for the training stage once the error exceeds a certain threshold.

[10][11]and [4] define it in this way, whereas [29] considers $\widehat{E}(w) = \frac{1}{r}E(w)$

being n the number of input data, m the number of output neurons and meaning $(d_i)_j$ the $i-th$ element of $d_j = [(d_1)_j, (d_2)_j, \dots (d_m)_j]$", or in its instantaneous version:

$$E_I(\mathbf{w}) = \sum_{i=1}^{m} |(d_i)_j - (y_i)_j|^r$$

These error functions are associated with the so called ***L-r distances:***

$$L^r(\mathbf{y}_j, \mathbf{d}_j) = \left[\sum_{i=1}^{m} |(d_i)_j - (y_i)_j|^r\right]^{1/r} = [E_I(\mathbf{w})]^{1/r}$$

The use of these functions can lead us to a maximum likelihood estimation of the weights for arbitrary input patterns, choosing r appropriately [4].

If $r = 2$ we have the euclidean distance (and E is the sum of squares error).

Some properties of this distance for $r = 1$ (called ***Manhattan norm***) are mentioned in what follows.

When $r \to \infty$ the associated L-r distance is ([4]):

$$L^\infty = \underset{i=1,\dots m}{Max} |(d_i)_j - (y_i)_j|$$

A value for r such that $1 \leq r < 2$ gives lower weights to higher $(d - y)$ deviations (because the difference $|d_i - y_i|$ is raised to a power between 0 and 1).

For the Minkowsky error functions the weight modification is [10]:

$$\Delta w_{lj} = \rho_o \textstyle\sum_{k=1}^{n} sign[(d_l)_k - (y_l)_k] \, |(d_l)_k - (y_l)_k|^{r-1} \, f_o{}'[(Net_l)_k](z_j)_k$$

(output layer)

$$\begin{cases} \alpha = f_h{}'[(Net_j)_k](x_i)_k \\ \Delta w_{lj} = \rho_o \sum_{k=1}^{n} \sum_{l=1}^{L} \left[sign[(d_l)_k - (y_l)_k] \, |(d_l)_k - (y_l)_k|^{r-1} \, f_o{}'[(Net_l)_k](z_j)_k\right] \alpha \end{cases}$$

(hidden layer)

From a statistical viewpoint, the case $r = 1$ corresponds to minimize the conditional median of the error whereas $r = 2$ is the minimization of the conditional mean error of obtaining the network outputs conditioned to the inputs [10].

Special case: learning using the Manhattan norm The network training (including the Langevin training, see 2.1.1.3 on page 20), can be formulated in general as

$$\mathbf{w}(n + 1) = \mathbf{w}(n) + \rho(n)H\left[\mathbf{w}(n), \mathbf{x}(n)\right] \textbf{ (Eq. 2.11)}$$

where $\rho(n)$ is the learning rate and H the learning rule.

For each value of the input $\mathbf{x}(n)$, assumed random, independent and identically distributed, we will obtain a new $\mathbf{w}(n)$ from certain starting input $\mathbf{w}(0)$ and using Eq. 2.11. That sequence of values $\mathbf{w} = \{w_{ij}\}$ is a Markov chain whose states have transition probabilities:

$$P(w_{ij},n+1)-P(w_{ij},n)=\int\left[P(w'_{ij},n)W(w_{ij}\left|w'_{ij}\right.)-P(w_{ij},n)W(w'_{ij}\left|w_{ij}\right.)\right]dw'$$

(Eq. 2.12)

where the integration is over the whole variation set of $\mathbf{w}$ and being W the transition probabilities in a step:

$$W(w_{ij}\left|w'_{ij}\right.)=\left\langle\delta(w_{ij}-w'_{ij}-\rho H(w'_{ij},\mathbf{x}))\right\rangle_{\mathbf{x}}$$

where $\langle...\rangle_{\mathbf{x}}$ means the average with respect to (or over all the values of) $\mathbf{x}$ and δ the Kronecker delta function.

Equation 2.12 does not have an exact theoretical solution, so approximations are used. The equation can be re-written in a power series with baseρ , obtaining the Kramers-Moyal's series [32]:

$$\begin{aligned}&P(w_{ij},n+1)-P(w_{ij},n)=\textstyle\sum_{n=1}^{\infty}\frac{(-1)^n}{n!}\left(\frac{\partial}{\partial w_{ij}}\right)^n\left[a_n(w_{ij})P(w_{ij},t)\right]\\&a(w_{ij})=\rho^n\left\langle H^n(w_{ij},\mathbf{x})\right\rangle_{\mathbf{x}}\end{aligned}\quad\textbf{(Eq. 2.13)}$$

Additionally, if we only take the two first terms from Eq. 2.13, we obtain the Fokker-Plank approximation.

In the case of the Manhattan rule, the Kramers-Moyal's series can be exactly summed, in other words, Equation 2.12 has exact solution [32], and can be found as a sum of a finite number of terms (of course different from the Kramers-Moyal's series).

In this case, we would get

$$\Delta w_{ij}=H_{ij}(\mathrm{w},\mathbf{x})=-sign\left[\frac{\partial E(\mathrm{w},\mathbf{x})}{\partial w_{ij}}\right]$$

where $\frac{\partial E(\mathrm{w},\mathbf{x})}{\partial w_{ij}}$ represents the component that corresponds to weight w_{ij} of the instantaneous gradient (that is, for the pattern $\mathbf{x}$) of E.

Relative entropy The relative entropy (or Kullback-Leibler distance) can be as well used as an error function (see 2.1.3 on page 28).

2.3 Global optimum search techniques

We will describe a technique that can be implemented together with the back-propagation algorithm in order to reach the global optimum of the error function. Another common technique is represented by genetic algorithms, discussed in Subsection 2.2.5. Finally, other methods produce searches starting with the minimum of auxiliary functions, in a process known as "tunneling" (see [15]).

2.3.1 Langevin learning rule

Random descent gradient methods employ noise in order to perturb the function $E(\mathbf{w})$ to be minimized, hence avoiding local minima and "bad" solutions, trying to get a "good" global solution. During the search process the modifications for E are gradually deleted so the function to be minimized is effectively $E(\mathbf{w})$ when the final solution is reached.

Langevin's learning rule adds noise to the weights during the training phase, which corresponds to produce a gradient descent of a distorted function $\widetilde{E}(\mathbf{w}, \mathbf{N})$:

$$\widetilde{E}(\mathbf{w}, \mathbf{N}) = E(\mathbf{w}) + c(t)\mathbf{w}^T\mathbf{N}$$

being $E(\mathbf{w})$ the target error function to be minimized, $\mathbf{N} = [N_1, N_2 .... N_n]^T$ an array of additive white independent Gaussian noise and $c(t)$ a parameter that controls the noise magnitude [72]. The noise should be gradually reduced, so $\underset{t\to\infty}{lim}\, c(t) = 0$. A possible choice is $c(t) = \beta e^{-\alpha t}$ with $\beta \neq 0$ and $\alpha > 0$. The gradient of the perturbed function is $\nabla\widetilde{E}(\mathbf{w}, \mathbf{N}) = \nabla E(\mathbf{w}) + c(t)\mathbf{N}(t)$, and since in the general case it is $\mathbf{w}(t+1) = \mathbf{w}(t) - \rho\nabla\, g(\mathbf{w})|_{\mathbf{w}(t)}$ so when replace with the distorted case (replacing the function $g(\mathbf{w})$ by $\widetilde{E}$) we get that:

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \rho\left[\nabla\, E(\mathbf{w})|_{\mathbf{w}(t)} + c(t)\mathbf{N}(t)\right]$$

or

$$\Delta\mathbf{w} = -\rho\left[\nabla\, E(\mathbf{w})|_{\mathbf{w}(t)} + c(t)\mathbf{N}(t)\right]$$

Observe that if the noise $\mathbf{N}$ has zero mean, the search will in average follow the gradient for E:

$$\left\langle \nabla\widetilde{E}(\mathbf{x}, \mathbf{N})\right\rangle_N = \langle\nabla E(\mathbf{x})\rangle_N + c(t)\langle\mathbf{N}(t)\rangle_N = \nabla E(x)$$

where the sub-index N means "the expected value is taken over all possible noises".

The probability to produce a global optimum in this way critically depends on the noise magnitude $c(t)$. For example, when $c(t) = \beta e^{-\alpha t}$, the coefficient β controls the noise magnitude and α the vanishing rate. It is necessary to choose β big enough to explore the search space profusely. Choosing α big should dissipate the random effect rapidly so the search is prematurely translated into a deterministic one, increasing the probability of producing a local minimum. Small values for α are desirable since they allow a high number of points to be explored from the search surface, a basic element for a global optimization. However, if α is too small, the convergence will be very slow. Therefore, α must be chosen to reach a trade-off between an adequate speed of convergence and the production of a global optimum (or close to global).

This stochastic gradient rule can be used for all the gradient-based learning rules for feedforward networks, and it is called ***Langevin's learning rule***. In the case of back-propagation the Langevin's learning rule leads to

$$\Delta w_{lj} = \rho_o(d_l - y_l){f_o}'(Net_l)z_j \ + \ \rho_o c_o(t)N_{lj}(t)$$

$$\Delta w_{ji} = \rho_h \left[\sum_{l=1}^{L} (d_l - y_l){f_o}'(Net_l)w_{lj} \right] {f_h}'(net_j)x_i + \ \rho_h c_h(t)N_{ji}(t)$$

for output and hidden neurons respectively (the sub-index o and h for c mean we might use different noise magnitudes for the hidden and the output neurons)[11].

Training using Langevin's rule is usually computationally more effective than a deterministic back-propagation in order to escape from local minimum (Hoptroff and Hall, cited by [29]), giving better results in both speed of convergence and quality of the results when comparing Lavengin's rule with high-order methods when the Hessian matrix is ill-conditioned (which is common in the feed-forward networks with hidden layers [72]).

Last, it is worth to note that incremental back-propagation can be seen as a random gradient search, but randomness is in that case intrinsic to the gradient, given the nature of the "instantaneous" error minimization (when training vectors are randomly presented, as opposed with an artificially introduced randomness) used here. The noise here introduced is homogeneous, in the sense it is the same on each minimum for E, whereas the one for incremental BP is not, because it is related with intrinsic fluctuations given by the randomness of presented patterns. In incremental BP, the grater the error, the longer the learning phase, the higher the weight fluctuations as well as the greater noise and the possibilities to escape from local minima. This non-homogeneous noise gives it an advantage over Langevin's learning rule referred to escaping from local minima, which has been tested in simulations by Heskes and Kappen (cited by [29]). However, the incremental BP does not succeeds to escape from the flat spots problem while Langevin's rule does.

2.3.2 Non-gradient-based training methods

An exposition of non-derivative based training methods is presented, such as Genetic Algorithms combined with Neural Networks. A hybrid method is described, trying to exploit the best from both worlds: genetic algorithms and gradient descent searches.

2.3.2.1 Evolutionary artificial neural networks

Evolutionary artificial neural networks represent a special kind of neural networks where evolution, as well as learning, is another adaptation method. This evolution is commonly simulated by genetic algorithms or other evolutionary algorithms[12]. Some applications of evolutionary algorithms are the training itself (weight adaptation), selection of starting weights, architecture design (network structure and transfer functions), learning of the learning rule, etc. [72].

Evolutionary search procedures Evolutionary algorithms are based on the evolution of a population of competitive individuals that exchange information between each other, being specially suited when dealing with high-dimensional functions with several local optimum points [72]. In particular, genetic algorithms represent a sub-class of these algorithms, where their implementations depend on the way to encode the genetic information of the individuals, the selection process through generations and the genetic operators.

[11]With the standard convention, when $\mathbf{w}$ is considered a vector, N is taken as a vector with equal dimension. Therefore, when $\mathbf{w} = \{w_{ij}\}$ is considered, $\mathbf{N} = \{N_{ij}\}$ will have the same structure.

[12]We understand evolutionary algorithms by genetic algorithms, evolutionary programming and evolutionary strategies[90].

The general structure of a genetic algorithm can be schematically represented by the following template:

Step 1 - Generate the starting population $G(0)$ at random and let $i = 0$.
Step 2 - Repeat until the stopping criterion holds:

- *Score each individual in the population*
- *Choose ancestors from $G(i)$, using the fitness function*
- *Apply the genetic operators to the ancestors and use the results to get a new population set $G(i+1)$*
- *Update i=i+1*

Evolutionary neural networks The evolution of these networks take place roughly at three levels: weight evolution, architectural evolution and learning rates evolution. The weight evolution introduces an adaptive approach of the training, specially in the learning of recurrent networks and by reinforcement [13] where the gradient descent algorithms commonly present difficulties [72].

Weight evolution Even if BP has had a successful application in several cases, it has some deficiencies coming from its gradient descent use: it produces local minima very often, mainly for multi-modal error functions; additionally, it requires to work with differentiable error functions. An alternative approach is to implement the training stage with an evolutionary approach. Genetic algorithms could be applied to try a global optimization in the weight space. The fitness function for the evolutionary network can be defined in accordance to the different needs. The network complexity can be included in it, as well as the difference between the real and the desired outputs. On the other hand, the fitness function is not required to be differentiable: GA are not gradient-based methods.

The training phase seen as a weighting evolution has two stages:

1. ***decide the genotype (individual) representation for the connections weights (for instance, they could be represented by binary strings) and***
2. ***the evolution itself, simulated by GAs or other evolutionary algorithm.***

Different representation schemes and genetic operators can lead to very different training performances.

A typical cyclic control for the weight evolution could be this one:

[13]Reinforcement learning is a special kind of learning where the exact desired output is unknown, and is based only on the fact that the real output is either correct or not [90].

1. *Encode each genotype (individual) from the current generation as a weight set, and construct the respective network with these weights.*
2. *Score each so constructed network finding the mean square error between the real and the desired outputs*[a]*. The fitness function for each individual is determined by this error: a higher error means a lower fitness.*
3. *Cross the individuals to have offsprings, choosing the number of descendants according to the "fitness" of the current generation*
4. *Apply the "crossover" and "mutation" genetic operators to every descendant from Step 3, obtaining a new generation.*

[a]or to use other error function.

The weights could be represented by binary digits: each weight would be represented with a binary number of specific length, and the network itself is represented by the concatenation of all those weights. For instance, the network

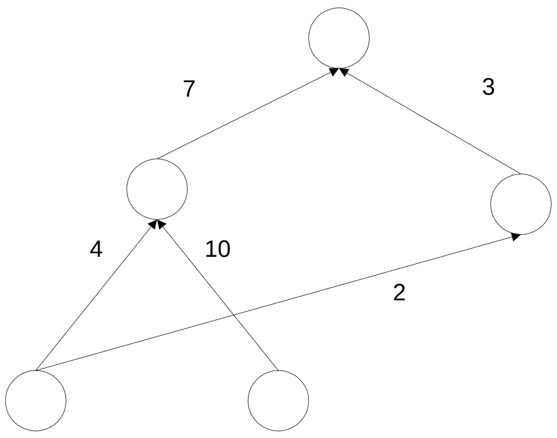


Figure 2.13

could be represented by 0100 1010 0010 0000 0111 0011. An alternative way to represent the network is by means of a set of real numbers representing its weights. For the previous network we would write (4.10.2.0.7.3).

Each representation has its own genetic operators: we cannot use a genetic operator that works on binary words wit a real number representation of the weights. In this case, the mutation operator (as bit permutation) could be replaced by the addition of a random number to the corresponding weight.

In the software used, many operators exist implementing these algorithms. For example, there are four different mutation operators (types) available: ·

- Uniform - Replaces the value of the chosen gene with a uniform random value selected between the user-specified upper and lower bounds for that gene. ·
- Boundary - Replaces the value of the chosen gene with either the upper or lower bound for that gene (chosen randomly). ·
- Gaussian - Adds a unit Gaussian distributed random value to the chosen gene. The new gene value is clipped if it falls outside of the user-specified lower or upper bounds for that gene. ·

- Non-Uniform - Increases the probability that the amount of the mutation will be close to 0 as the generation number increases. This mutation operator keeps the population from stagnating in the early stages of the evolution, then allows the genetic algorithm to fine tune the solution in the later stages of evolution. The chosen gene is mutated according to the following equations:

$$mutatedGene = \begin{cases} gene + \Delta(generationNumber, upperbound - gene)\ for\ l = 0 \\ gene - \Delta(generationNumber, gene - lowerBound)\ for\ l = 1 \end{cases}$$

  where l is a random binary value and

$$\Delta(t, y) = yr(1 - \frac{t}{max\{GenerationNumber\}})^b$$

  being

  - r a random number between 0 and 1 and
  - b is a system parameter that controls the degree of non-uniformity [68].

There are also five operators related to crossover, that we won't comment here (see [68]).

Architectural evolution and learning rules The optimal architectural design for an evolutionary network can be formulated as a search problem over the architectures space, where each point represents a possible architecture. For a given "performance" criterion (faster training stage, lower complexity, etc.), a surface is obtained where each point corresponds with the "performance" of a certain architecture; the desired design will be the "highest" point (with better performance) in the surface.

The use of evolution to determine the optimal architecture is based on some works that show the search of that structure using evolutionary algorithms returns better results than the "growing" or "pruning" methods, given the characteristics of the architectures space:

- the surface is infinite, because the number of neurons and possible connections is unbounded
- the surface is non-differentiable, because the changes in the nodes or connections are naturally discrete and can have a non-continuous effect over the network performance
- the "mapping" between an architecture and its performance is not direct, and depends on the chosen evaluation method
- similar architectures could have very different network performances
- different architectures could have similar performances

The used transfer functions can evolve together with the network topology. The learning rule can evolve as well. White and Ligomenides made evolve the learning rule in a network (cited by [72]) and after 1000 generations, starting with a population with randomly chosen rules, obtained the delta-rule.

Gradient-based vs. Evolutionary training Evolutionary training is attractive when the gradient information is not available or is computationally expensive. This training has been used during the learning stage of recurrent networks and reinforcement learning. It is interesting to observe that the same evolutionary algorithm can be used for different networks, no matter whether they are recurrent or not, saving human effort to develop training algorithms for different network types. Regularization terms or other constraints (such as "weight sharing") can be added to the fitness function, even if it is not continuous.

Evolutionary algorithms are, in general, slower than the fastest back-propagation techniques, but can work in networks where the gradient is not available. Comparisons of the convergence speeds strongly depends on the chosen back-propagation technique (a fast BP vs. a standard GA or a standard BP vs. fast GA) [72].

Hybrid training One of the major drawbacks of genetic algorithms is its inefficiency to find a local optimum, although they are good to look for the global optimum with enough computational time. The training efficiency of the evolutionary algorithms can be greatly improved adding local search procedures during its evolution, that is, combining the local search inherited from other algorithms (for instance, BP). Experimental results show that these hybrid searches are more efficient than GAs or BP alone, even though the BP technique should be trained several times to reach a competitive solution given its sensibility to the initial conditions.

A special hybrid method Hassoum [29] describes a hybrid algorithm for feedforward networks with one hidden layer. This method is based on the fact that feedforward networks with one hidden layer are universal approximators (and classifiers): this network family is rich enough to get as closer as desired to every target function or to classify every target set of patterns, provided the necessary amount of neurons can be used in the hidden layer (see 2.10 on page 108). Based on that, we divide the original network into two sub-networks working in cascade, and a genetic algorithm will be implemented over the space of feasible output values for the hidden layer combined with a gradient search (delta rule) in both networks.

Consider the network from Figure 2.8. If we had a set of output vectors from the hidden layer $\{\mathbf{h}_1, \mathrm{h}_2, \ldots \mathrm{h}_m\}$[14] such that the corresponding "mappings" are respected: $\{\mathbf{x}_k\} \to \{\mathbf{h}_k\}$ and $\{\mathbf{h}_k\} \to \{\mathbf{d}_k\}$ for $k = 1.2, ...m$, we could apply a descent gradient search to learn both fast and independently the weights for both the output and hidden layers. However, at the beginning we do not know the adequate set of output values for the hidden layer $\{\mathbf{h}_k\}$ that solves the problem. Therefore, we will use a genetic algorithm to let evolve that set of output values from the hidden layer, to get that $\{\mathbf{x}_k\} \to \{\mathbf{h}_k\}$ and $\{\mathrm{h}_k\} \to \{\mathrm{d}_k\}$.

We will encode those values with strings $\mathbf{s} = [\mathbf{s}_1, \mathbf{s}_2 \ldots \mathbf{s}_m]$ where $\mathbf{s}_i$ is obtained taking the binary representation for the output $\mathbf{h}_i$. Additionally, each search point can be represented by a matrix $\mathbf{H} = [\mathbf{h}_1\ \mathbf{h_2} \cdots \mathbf{h}_m]$ of size J x m .

[14]m is the number of available training patterns.

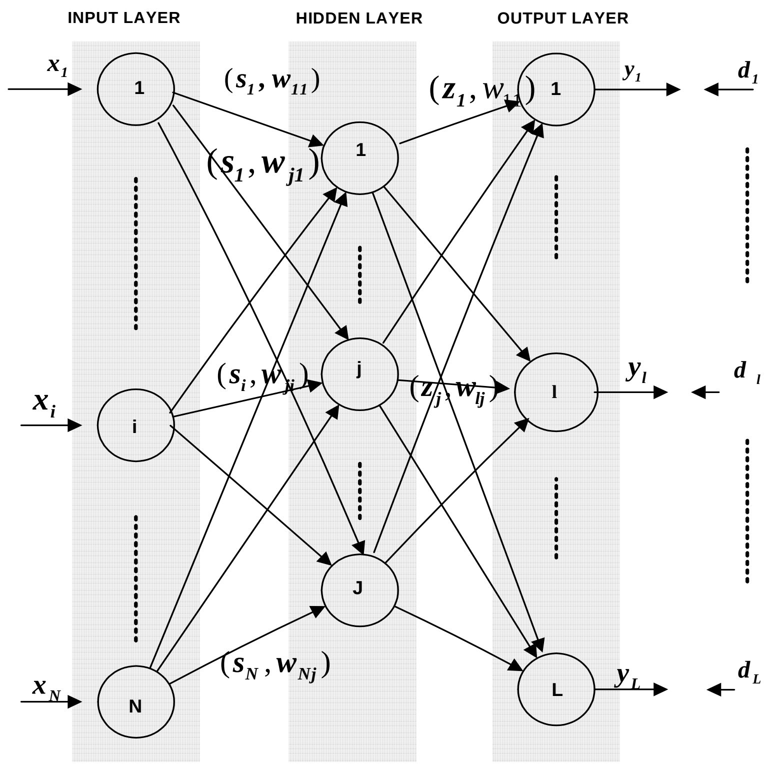


Figure 2.14

As an initial population of search points, we will generate m randomly chosen binary matrices $\{\mathbf{H}_j\} \quad j = 1, 2, ...m$. The matrix $\mathbf{H}_j$ has an associated network that possesses the same topology than the original network, being the m networks initialized with he same set of random weights.

The fitness value of the search point j (that is,the fitness of the matrix $\mathbf{H}_j$) will be proportional to the sum of squares errors over the output of the hidden/output networks:

$$E_j = \frac{1}{2} \sum_{k=1}^{m} \sum_{l=1}^{L} \left[d_k^l - \left(y_l^k \right)_j \right]^2$$

where$\left(y_l^k\right)_j$ is understood as the output of the neuron l of the hidden/output network j when the input pattern $\mathbf{x}_k$ is presented. The fitness function can have different expressions.

Some options are:

$$f(\mathbf{H}_j) = -E_j$$
$$f(\mathbf{H}_j) = 1 - \left(\frac{E_j}{\max\limits_{\{\mathrm{H}_j\}} E} \right)$$
$$f(\mathbf{H}_j) = 1/\left(E_j + \varepsilon\right)$$

being ε a small positive real. It should be taken into account the fact that different fitness functions will lead to different overall performances of the algorithm.

A template for the algorithm is:

- ***Initialization:***
  - *Starting from random weights and outputs for the hidden neurons, the weights of all m networks are adapted with respect to the training set* $\{\mathbf{x}_k, \mathbf{h}_k\} \quad k = 1, 2, ...m$, *using the delta rule*[a]. *Similarly, the weights of the connections that reach the neurons from the hidden layer are adapted according to the training set* $\{\mathbf{h}_k, \mathbf{d}_k\} \quad k = 1, 2, ...m$, *independently of the first network.*
- ***Repeat until*** *(at least one out of the m generated input-output networks have an error* E_j *lower than a specified value)*[b]
  - ***Fitness test****: after the weights are updated, every network is tested executing feedforward calculations, and the fitness is evaluated. During these calculations, the outputs from the first network are used as inputs of the second one (the one that contains the output neurons).*
  - ***Evolution:*** *Genetic operators are introduced to get the next generation* $\{\mathbf{H}_j\}$. *Regarding the crossover, the best m/2* $\mathbf{H}_j$ *(with the highest fitness values) are duplicated and temporally kept for crossover. Crossings are produced with probability* P_c *(close to 1): a pair* $\{\mathbf{H}_i, \mathbf{H}_j\}$ *is chosen without replacement over the mentioned set. If a training set* $\{\mathbf{x}_k, d_k\}$ *is poorly trained for the network i during the previous training stage, that is, the output error of this pair is substantially higher than the average error over the whole training set, then the corresponding column* $\mathbf{h}_k$ *from* $\mathbf{H}_i$ *is replaced by the column number k of* $\mathbf{H}_j$. *Crossovers can affect several column pairs of the matrices* $\mathbf{H}$.
- ***End repeat***

[a]Other learning rules can be used as well.
[b] In [29] additional stopping conditions are proposed in order to avoid endless loops.

Non-derivative based methods This topic is thoroughly covered in [65].

ALOPEX. This algorithm updates the network weights by small perturbations of its values, in accordance with the correlation between the perturbation direction and resulting change in the error regarding the whole training set. This algorithm is interesting because of its independence of the network topology and of the error function used. As a consequence, it can be used for other optimization applications different from neural networks.

Cauwenberghs' algorithm. Its learning rule is close to the one used in Alopex. A random perturbation $\mathbf{z}$ is added to the current weight vector $\mathbf{w}$, and the resulting error $E(\mathbf{z} + \mathbf{w})$ is found. This number is used to update the weight vector by:

$$w^{new} = w - \alpha z[E(w + z) - E(w)]$$

where α is the learning rate.

2.4 Dynamic networks

In this section, networks where their outputs are not only function of the "instantaneous" inputs are presented. Additionally, some design issues as well as training algorithms for these networks are revisited.

2.4.1 Training

A ***dynamic network*** is a network that behaves like a dynamical system with respect to its outputs and inputs. We will see how to apply this definition directly to the (predictive) network training.

Given the observed values of the output $\mathbf{y}$ of a discrete-time dynamical system previous to the current instant t, we will try to predict exactly $\mathbf{y}(t+p)$, being $p > 0$. As long as the quantity p is increased, the quality of the prediction will be degraded for any predictive method. Here we will try to keep the prediction accuracy for a wide range of values p. Given a dynamic neural network with input $\mathbf{x}$ and output $\mathbf{y}$, we can write:

$$\mathbf{y}(t+1) = g\left[\mathbf{y}(t),\ \mathbf{y}(t-1), ...,\ \mathbf{y}(t-n),\ \mathbf{x}(t),\ \mathbf{x}(t-1), ...,\ \mathbf{x}(t-m)\right]$$

being $n \geq m$ and g a non-linear function [29]. In order to train the network, back-propagation can be used from "static" pairs

$$\{\{\mathbf{y}(t), \mathbf{y}(t-1),\ ...\ \mathbf{y}(t-n), \mathbf{x}(t), \mathbf{x}(t-1)\ ...\ \mathbf{x}(t-m)\},\quad \mathbf{y}(t+1)\}$$

If the training phase is successful, it is expected to have the real output $\mathbf{y}(t+1)$ as close as the desired one, $\mathbf{d}(t+1)$. This technique is called ***"windowing"***. As we will see, there are also other training methods for dynamic networks.

2.4.2 Back-propagation through time (BPTT) and RTRL

Unfolding the network through time Consider the network in Figure 2.15:

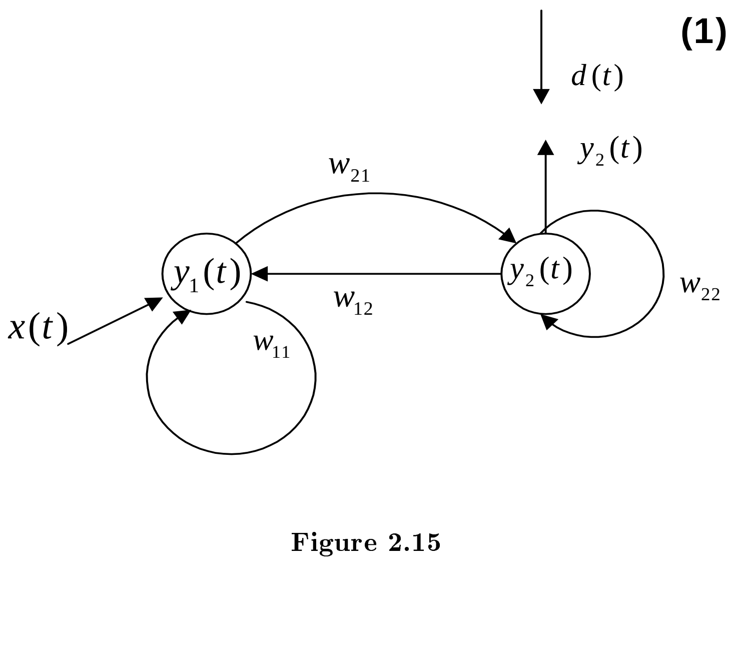


Figure 2.15

where $d(t)$ is the desired output value, $y_2(t)$ the real output for the network input $x(t)$, at instant t.

A network that behaves identically for times $t = 1, 2, 3, 4$ is obtained unfolding the network through time, in such a way that only "feedforward" layers result:

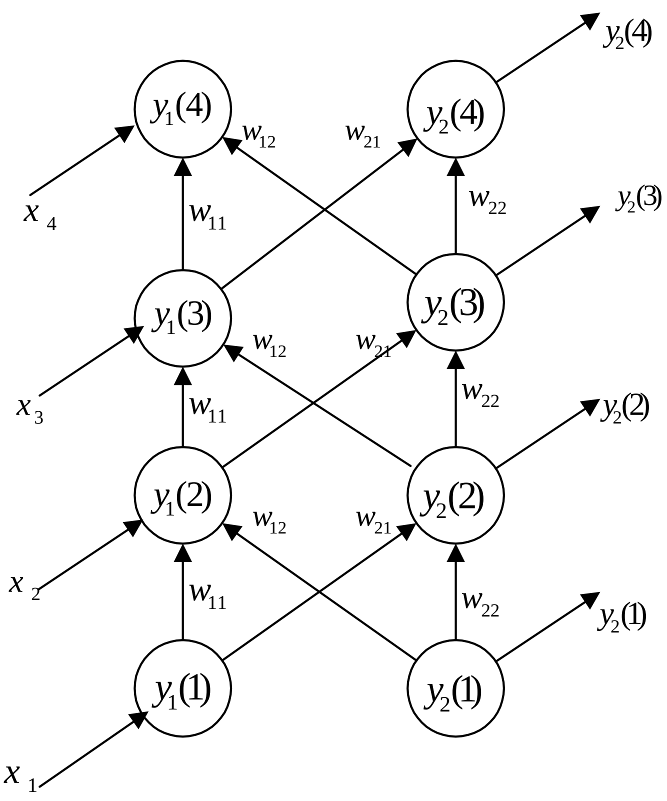


Figure 2.16

The number of resulting layers equals the unfolding (expanding) time interval T. This concept is applicable whenever T is small, since all neurons are replicated T times and hence the maximum sequence length that can be processed is constrained. Note that the connections w_{ji} from neuron j towards neuron i are identical in all the layers from the expanded network. The BP algorithm adaptation to train an unfolded network is known as "***back-propagation through time***", ***BPTT***) [30].

The training of the unfolded network can be done by epochs or in an incremental way, deriving two possible implementations for the temporal BP: the training by epochs and the real-time training.

Training an unfolded network by epochs Let us divide the training set in independent epochs, where each epoch represents a temporal training pattern[15]. Let n_0 and n_1 be respectively the beginning and ending of a certain epoch. We can define the error function

[15]The meaning of epoch is local to this section, and differs from the meaning given when referring for example to the training of MLPs.

by

$$E(n_0, n_1) = \frac{1}{2} \sum_{n=n_0}^{n_1} \sum_{j \in \mathrm{A}} e_j^2(n)$$

where A is the set of all indexes j corresponding to those neurons for which the desired outputs is specified, and $e_j(n)$ is the output error from those neurons, with respect to the desired output. We wish to find ∇E, that is, the partial derivatives of E with respect to the network weights $\mathbf{w}$. For that purpose, we will use the following algorithm based on batch back-propagation:

1. Forward pass: all input data within the interval (n_0, n_1) are presented to the network. Those data are stored, as well as the desired outputs and network weights
2. Backward pass: the local gradients are found using the previously stored information:

$$\delta_j(n) = -\frac{\partial E(n_0, n_1)}{\partial Net_j(n)} \quad \forall j \in \mathrm{A} \quad n_0 < n \leq n_1$$

This can be calculated in the following way [30]:

$$\delta_j(n) = \begin{cases} f'[Net_j(n)]e_j(n) \quad if \quad n = n_1 \\ f'[Net_j(n)][e_j(n) + \sum_{k \in \mathrm{A}} w_{jk}\delta_k(n+1)] \quad if \quad n_0 < n < n_1 \end{cases} \qquad \textbf{(Eq. 2.14)}$$

where $f'(\bullet)$ is the derivative of the output function and $e_j(n) = y_j(n) - d_j(n)$ means that $y_j(n)$ is the component number j for the output at time n. It is assumed that all neurons within the network have the same output function. Equation 2.14 is iteratively used starting at time n_1 and going back, step by step, until n_0 is reached. The number of steps here stated equals the number of time steps of the time interval (epoch).

3. Once performed the BP calculations until step n_0+1, the following weight adjustment for neuron j takes place:
4. $$\Delta w_{ji} = -\rho \frac{\partial E(n_0, n_1)}{\partial w_{ji}} = \sum_{n=n_0+1}^{n_1} \Delta w_{ji}(n-1) = \rho \sum_{n=n_0+1}^{n_1} \delta_j(n) x_i(n-1)$$

   being $x_i(n-1)$ the input applied to the synapse i of neuron j at time n-1 [30].

When this technique is compared with the ordinary BP batch we can see that the main difference is that the desired neuron outputs are specified at different network levels, because the real output layer is replicated several times when the network is unfolded.

The term BPTT comes from the fact that we do not only back-propagate the calculations from the output towards the input, but we employ this technique through time (from the last time towards the starting one).

This algorithm is not local with time, but it is in the topology: to find δ values corresponding to different instants are needed, but not their values over other neurons.

For practical reasons, truncated versions of the BPTT are implemented, as the next one:

Truncated back-propagation:

- The instantaneous error is used $E(n) = \frac{1}{2} \sum_{j \in \mathrm{A}} e_j^2(n)$

- The weight adjustments are performed considering $-\nabla E$ at step n. The weight update is “on line”, while the network is working.
- The term “truncated” comes from the fact that inputs, weights and desired outputs are stored (step 1 of the previous algorithm) for a finite number of steps in each epoch. This fixed number is called ***truncation depth.*** All the historical information older than this depth is ignored. This truncation is mandatory; otherwise, the computational time and storage needs could increase linearly with time, turning the process impracticable (see [30]).

BPTT is not widely used because of its limitation to work with short trajectories of length $= T$, so in some cases recurrent back-propagation is preferred instead [29].

Training an unfolded network in real time: RTRL The real time recurrent learning (RTRL) method was proposed by Williams and Zipser ([85] cited by [29]), and allows us to train the network while it works (that is the reason for the term “real time”), though it is computationally expensive. All in all, this method has the generality of the back-propagation approach through time without suffering the problem of the increasing of the needed memory when T augments.

BPTT has been derived assuming the weights were fixed during the whole variation of n within $[n_0, n_1]$. Given that we pretend to perform a real-time training working with (possible) unbounded time intervals, we can try a less-restrictive condition, increasing each weight a value $\Delta w_{ji}(n-1)$ without summing the variations, but summing and applying all of them together afterward. A potential drawback from this procedure is that the negative gradient of the total error is no longer followed along the whole trajectory. Nevertheless, it is totally analogous to the incremental batch back-propagation: while the resulting algorithm cannot ensure that the negative gradient is followed, the practical differences between the two versions are small in general, with both versions turning almost identical when the learning rate is small [85]. The real-time recurrent learning is hence obtained.

2.4.3 Other variations of the back-propagation algorithm

We continue describing two variants of back-propagation, for the case in which time varies in a continuous way.

Recurrent Back-propagation This algorithm is used in totally recurrent networks in which the time varies in a continuous domain, and allows us to train a recurrent network to learn static associations $I \to O$ [29]. Consider a network with N output functions y_i, weights w_{ij} and output function $f(Net_i)$. We say that a neuron i is within the input layer if it receives an element x_i^k of the input pattern $\mathbf{x}_k$. We will also say that non-input neurons have associated an input $x_i^k \equiv 0$. The output neurons are those (by definition) which have desired output values d_i^k. In general, a neuron can be simultaneously in the the input and the output sets, or can be hidden in the sense it is neither an input nor output neuron.

From now on, we will omit the supra-index k that recalls the pattern, to simplify the notation.

When the network is in equilibrium, this is, with no output variations $\mathbf{y}^*$ whenever the input is held constant, the following relation holds [29]:

$$y_i^* = f(Net_i*) = f\left(\sum_{j\in A} w_{ij} y_j^* + x_i\right) \qquad i = 1...N$$

where $\mathbf{y}^* = [y_1^* \; y_2^* \; y_3^* .... \; y_N^*]^T$

Assume the network has converged towards an equilibrium state $\mathbf{y}^*$ as response to a| fixed input $\mathbf{x}$. Therefore, if the output neuron i answers with y_i^* but the desired output d_i was expected, an error E_i is produced.

We will try to adjust the weights in order to minimize

$$E = \frac{1}{2}\sum_{i=1}^{N}(d_i - y_i^*)^2 = \frac{1}{2}\sum_{i=1}^{N}E_i^{*2}$$

being $E_i^* = 0$ if neuron i is not an output neuron.

The weight update rule will be

$$\Delta w_{pq}^* = -\rho\frac{\partial E}{\partial w_{pq}} = \rho\sum_{i=1}^{N}E_i^*\frac{\partial y_i^*}{\partial w_{pq}}$$

It can be proved that [29]

$$\Delta w_{pq}^* = \rho f'(Net_p^*)y_q^*\sum_{i=1}^{N}E_i^*(\mathbf{L}^{-1})_{ip}$$

where the element $i\,j$ of the matrix $\mathbf{L}$ is $\mathbf{L}_{ij} = \delta_{ij} - f'(Net_i^*)w_{ij}$ and δ_{ij} is the Kronecker's delta function (1 provided $i = j$, 0 otherwise).

Since $\mathbf{L}$ is square with size N and the computation of its inverse requires $O(N^3)$ operations, an indirect method is applied instead to find the weight variations. The justification of the algorithm can be found in [29].

The learning algorithm for the recurrent back-propagation will be then (recall we will omit the supra-index that denote the pattern):

1. *An input pattern* $\mathbf{x}$ *is chosen and presented to the network*[a]*. A solution* $\mathbf{y}^*$ *is found by iteratively solving the equations*

$$y_i^* = f(Net_i) = f\left(\sum_{j\in A}w_{ij}y_j^* + x_i\right) \qquad i = 1, 2, \cdots N$$

2. *The solutions* $\mathbf{y}*$ *are used to compute the error*

$$E_i^* = \mathbf{y}_i^* - \mathbf{d}_i$$

3. *The vectors* $\mathbf{z} = \{z_i^*\}$ *are found by solving the equations*

$$E_i^* = -\,z_i + \sum_{j\in A}f'(Net_j^*)w_{ji}z_j \quad i = 1, 2 \ldots N$$

4. *Weights are adjusted using the formula*

$$\Delta w_{pq}^* = \rho f'(Net_p^*)y_q^*z_p^*$$

5. *Another pattern is presented in the input, and all previous steps are repeated, until an acceptable error is produced.*

[a] To the original network, not to the adjoint.

Observations:

- It can be proved that the incremental back-propagation algorithm is a special case of this, for feedforward networks.
- The algorithmic derivation assumed the equilibrium $\mathbf{y}^*$ does exist. Simard and others (cited by [29]) proved that for any recurrent network, we can always find cases where such stability is not achieved. Nevertheless, in practice it suffices to choose starting weights small enough to reach an equilibrium state.
- Instead of solving the system of equations form Step 3, another way to compute the vector $\mathbf{z}^* = [z_1^*, \ldots z_N^*]$ is by means of a recurrent network topologically similar to the original, called ***adjoint network***, in which the weights w_{ij} are replaced by $f'(Net_i^*)w_{ij}$, the output functions for all neurons are linear, all inputs are replaced by 0 and E_i^* is taken as input for neuron i. For example, the adjoint network from the original network in Figure 2.15 is depicted in Figure 2.17:

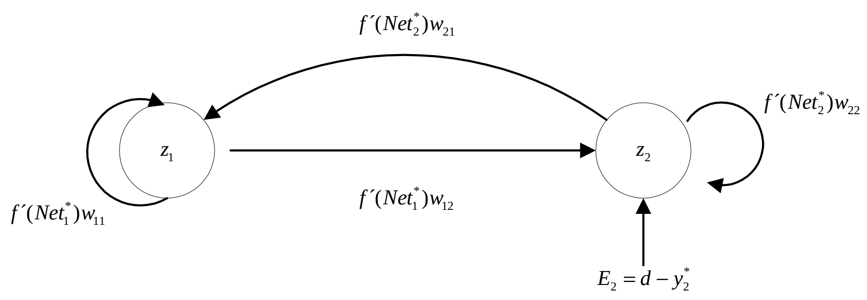


Figure 2.17

In this network the time also varies continuously.

2.4.4 Memory elements

Here we will describe some of the key aspects of the memory structures, and the networks that use them.

Types of memory elements In order to exploit the temporal structure of the input data, the network must have access to the time dimension. A possible way to achieve this is having structures that store the past of such data, called ***memory structures*** ("short term memories" or memory neurons). As a counterpart, we use the term long-term memory when we refer to the information stored at the network weights. Basically, a memory structure translates a sequence of samples in a point of the reconstruction space (see 2.9 on page 88). To add memory structures within the network, we use neurons specifically dedicated to store both the history of data and their respective outputs. Other way to manage temporal data is to employ the data inputs as windows ("***windowing"***), assigning windows (small sequences) of the time series and the respective outputs as network inputs. However, the introduction of memory structures has several benefits because the network is able to:

1. choose the temporal window size that better fits to the task
2. choose the ponderation weight of the data samples in the window that better reduces the output error

3. receive only the current observation from the world outside (the input data), such as a the biological archetype[16]does and keep the history of them

Recurrent networks can encode the temporal information, but they are complex. There are less complex topologies such as TLFNs that can store historical information as well, but they are still more complex than the static ones.

In its most general way, a memory element has the structure shown in Figure 2.18:

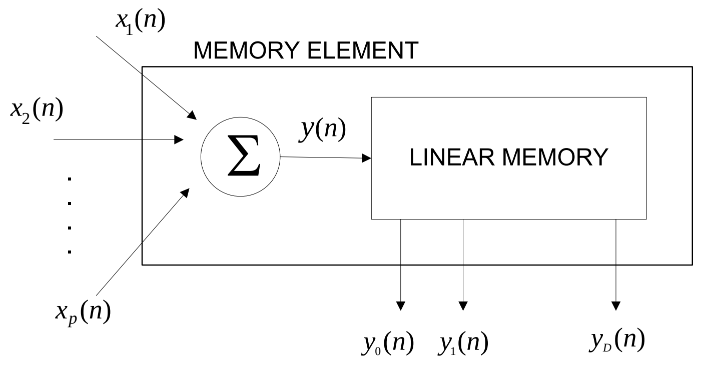


Figure 2.18

It is composed of a node that sums the inputs, plus a linear memory, described below. The output of a general memory element is multidimensional. The different types of memory elements are obtained by using distinct kinds of linear memories. We will see now two particular cases of them.

Delay line memories In these memory elements the linear memory is a ***"delay line".***

The delay line can be schematically shown as follows:

[16]Biological systems have notion of time and learn based on sequential observations in steps, but in multiple time-steps (windows).

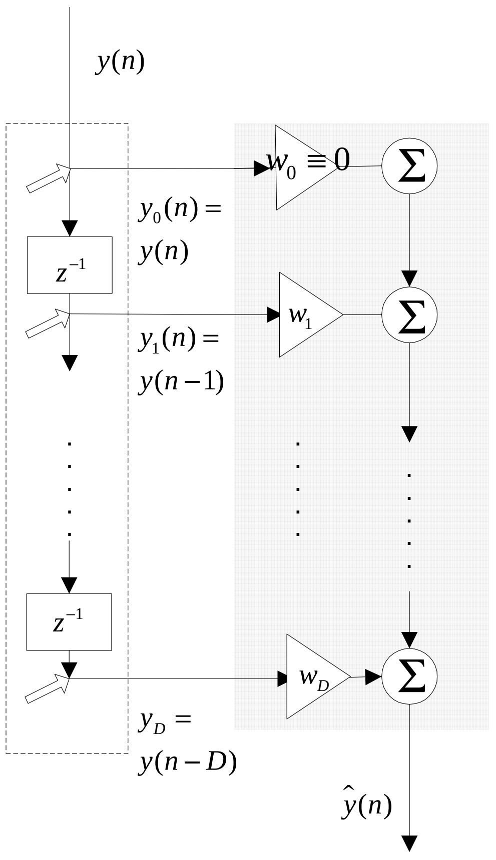
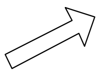


Figure 2.19

where z^{-1} represents a delay function[17] and the triangle the multiplication of the input signal by the value written inside it. So, given the inputs $x(n)$ the output is y$(n) = [y(n), \cdots y(n-D)]$.

The points marked with big arrows



are called ***"taps"***. The meaning of the shaded area and $\widehat{y}(n)$ are described below.

At step n, the output of a memory element of this kind, with D taps, only stores samples that occurred during the interval $[n-D+1, n)$, this is $D-1$ samples from the past. This kind of memory implements a sliding window with capacity D. This constant is called ***memory depth*** (see 2.4.4 on page 69). This kind of memory is employed by the TDNNs.

Feedback memories These memories are obtained from linear ones, in the way illustrated in Figure 2.20:

[17]In fact, the z-transform is used in the blocks. Therefore, z^{-1} is the z-transform for the unit impulse.

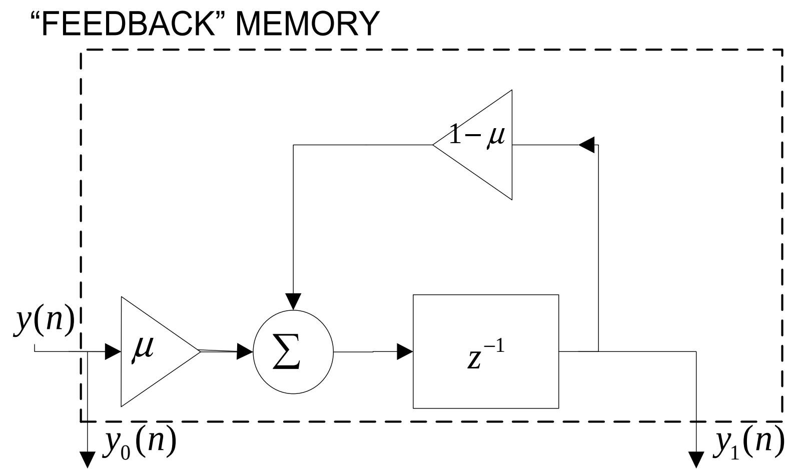


Figure 2.20

These memory elements are called ***context elements (or context neurons)*** and are used for instance in the Jordan/Elman´s networks.

Memory traces In a "feed back" memory, the past samples are not exactly preserved, since the outputs are the sum of the current input and a weighted version of the recently past output:

$$y_1(n) = (1 - \mu)y_1(n - 1) + \mu y_0(n)$$

so the historical information is progressively degraded, and that is why we term ***memory trace:*** a modified version of the input data. The idea of the trace is generalized to other memory structures, where previous samples are stored in a modified way. In the case of feed back memories the trace is $y_1(n)$. It also can be proved that in that case, the $x(n)$ projected over the trace space is [69]:

$$\widehat{y}(n) = y_1(n)w_1$$

Observe that the context memory has a single trace $y_1(n)$, whereas the "delay line" has D: $y_1(n), \dots y_D(n)$, each one corresponding with the outputs $y(n-1)$, $y(n-2), \dots y(n-D)$.

Generalized "feedforward" memory element The previous memory can be further generalized into a ***generalized feedforward memory element***, which uses a linear memory. This linear memory is such that the memory traces $y_k(n)$ are recursively found from the previous trace $y_{k-1}(n)$. It can be proved that a linear memory has the following output whenever $y(n)$ is injected at the input:

$$\begin{cases} y_0(n) = y(n) & \forall n \\ y_1(n) = g_0(n) * y_0(n) & \\ y_k(n) = g(n) * y_{k-1}(n) & k \geq 2 \end{cases}$$

where the symbol * here represents the convolution operation of sequences, $g(n)$ is a causal time function, normalized and time-invariant [18], called ***memory kernel*** and k represents the ***tap*** number. According to the selection of $g_0(n)$ and g, different types of linear memories are obtained, and as a consequence, different memory elements.

Schematically:

[18] A function $g(n)$ is causal when $g(n) = 0 \quad \forall n < 0$, and normalized when $\sum_0^\infty |g(n)| = 1$.

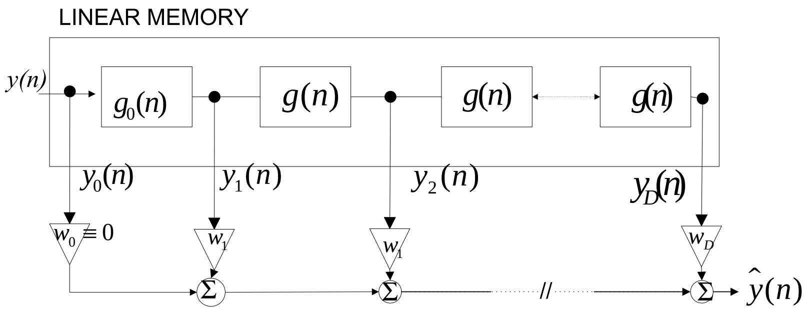


Figure 2.21

where the triangle represents the multiplication of the signal times the factor indicated inside it, and the circle represents the input sums. The number D-1 is called memory ***order***. In order to avoid confusion, we will assume that the input to the linear memory is $u(n)$. Consider the input vectors $\mathbf{u}(n) = [u(n), u(n-1), ...u(n-D)]$. The projection of $u(n)$ over the trace subspace (called ***memory trace subspace***) is $\widehat{u}(n) = [\widehat{u}(n), \widehat{u}(n-1), \ldots \widehat{u}(n-D)]$ where $\widehat{u}(n) = \sum_{k=0}^{D} w_k y_k(n)$, with $w_0 = 0$. Different selections for $g(n)$ *and* $g_0(n)$ provide different memory models. For example,

$$\begin{cases} g(n) = \delta(n-1) \\ g_0(n) = \delta(n) \end{cases}$$

(δ represents Kronecker's function) retrieves the "delay line memory". For the case of feed back memories it is

$$\begin{cases} y_0(n) = y(n)\ \forall n \\ y_1(n) = (1-\mu)y_1(n-1) + \mu y_0(n) \end{cases}$$

being the number of taps $D = 2$ and $g_0(n) = \mu(1-\mu)^n$ (see [69]).

The most appropriate "kernel" selection for a specific application is a current research field [69].

Gamma memory type I When

$$\begin{cases} g(n) = \mu(1-\mu)^n \quad n \geq 1 \\ g_0(n) = \delta(n) \end{cases}$$

we get the gamma memory type I. In this case it can be proved that,

$$g_k(n) = \begin{pmatrix} n-1 \\ k-1 \end{pmatrix} \mu^k (1-\mu)^{n-k} \quad n \geq k \quad k \geq 1$$

being $\begin{pmatrix} n \\ p \end{pmatrix} = \frac{n(n-1)...(n-p+1)}{p!}$.

The name of this memory comes form the fact that these functions are discrete versions of the gamma function integrand. This memory is stable (in the sense that the outputs are bounded over time) provided $0 < \mu < 2$ (see [69][69]). It is schematically represented in Figure 2.22:

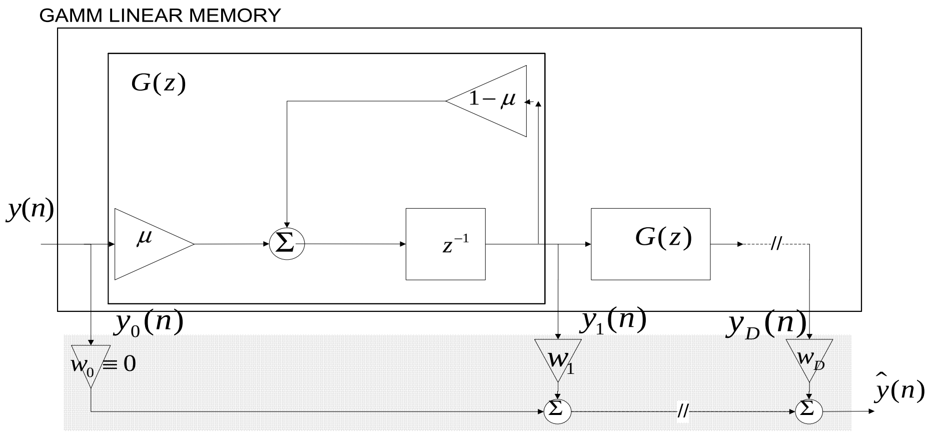


Figure 2.22

From Figure 2.22 it can be observed that the gamma memory unifies the delay and the context memories in a single model. Analytically:

$$\begin{cases} y_0(n) = y(n) \quad \forall n \\ y_k(n) = (1-\mu)y_k(n-1) + \mu y_{k-1}(n) \quad k = 1, 2, \ldots D \end{cases}$$

Properties of the gamma memories

The depth M for gamma memory (see below) is $M = D/\mu$. This memory has the property (which the delay lines do not have) of to separate the depth from the memory order. Consider for example a particular application that needs a memory with 100 samples but the memory traces can be modeled with three weights (free parameters). A delay line would require 100 taps and as a consequence 100 parameters (weights) from which 97 would be null. In fact, they could not be null because of the noise, with a consequent performance degradation. In a gamma memory it suffices to choose M=100, D=3 and $\mu = D/M$=0.03. These memory can represent N past samples in D taps ($D < N$) [69].

There exists a variant, called gamma type II memory that is not described here but fully covered in [69]. This memory type I is used by the neural network simulator chosen here.

Gamma type memories have been widely used ([68], [76]).

On the other hand, when in our case study we consider these memory types, the number of "taps" is associated with the dimension of the signal subspace.

Laguerre's memory This memory is obtained choosing

$$\begin{cases} g_0(n) = (1-\mu)^n\sqrt{1-(1-\mu)^2} \\ g(n) = (1-\mu)^n + (1-\mu)^{n+2} \end{cases}$$

The main advantage of this memory is that it leads to faster learning than gamma types, specially when μ is either close to 0 or 2 [69]. In this case

$$\begin{cases} y_0(n) = y(n) \quad \forall n \\ y_1(n) = (1-\mu)y_1(n-1) + y_0(n)\sqrt{1-(1-\mu)^2} \\ y_k(n) = (1-\mu)y_k(n-1) + y_{k-1}(n-1) - (1-\mu)y_{k-1}(n) \quad k = 2, \ldots D \end{cases}$$

Memory filters Whenever a memory element is used within a network, its outputs are connected with (common) adjacent neurons by means of serial connections with corresponding weights (shaded in previous schemes, sometimes seen as ***a linear combiner***). The set of the memory elements together with the linear combiner is called a ***memory filter*** ([69]), which is what is finally implemented in the simulator.

A fine overview of the links between gamma memories, Laguerre memory and filter theory can be found in [69].

These memories have also been widely used (e.g. [67][70][76]).

Depth and memory resolution For a certain generalized memory element we define the ***memory depth*** denoted with M, by

$$\begin{cases} M = \sum_{n=0}^{\infty} n y_D(n) \\ y_k(n) = g(n) * y_{k-1}(n) \end{cases}$$

The ***memory resolution*** R is the number of taps in the structure per sample or time unit. A memory with small depth just stores its content by a relatively short period, whereas a big depth allow to hold the content for long periods. A memory with high resolution is capable to store information over the sequence with high granularity, whereas low resolution corresponds to less detailed information. It holds that $R.M = D$ [69]. As a consequence, if we wish to increase the resolution, we will necessary reduce the depth. For example, in a context neuron the depth is $1/\mu$ and resolution is μ.

The following table summarized the depths and resolutions for the different described memories:

Depth vs. Resolution	Depth	Resolution
"Delay line"	D	1
Context	$1/\mu$	μ
Gamma type I	D/μ	μ

2.5 Alternatives to avoid the "vanishing gradient"

2.5.1 Training with Kalman filters

Filtering an input signal it to apply a transformation to it in such a way that certain characteristic of the signal are eliminated, in order to get a desired output. Therefore, the term "filter" will be equivalent in this context to "algorithm".

The ***linear Kalman filter*** helps to estimate the state of a linear dynamic system whose model is not completely known but accessed through a measure process with a certain level of noise. The filter uses incomplete information from that model to recursively improve the estimation of the system's state given by these measurements. The prediction is recursively found regarding the estimations from previous iterations. This is not the case of a gradient descent: normally, the derivatives of the error function only take into account the distance between the current and the desired output without regarding historical training data.

The extended Kalman filter is an adaptation of the linear case for non-linear systems.

The decoupled Kalman filter is based on the extended Kalman filter and is intended to manage the computational complexity corresponding to networks of considerable size. The whole available information provided to the network until the current instant is used every time, including the computed derivatives from the first iteration of the learning process. However, the algorithm works in such a way that just the last iteration results need to be explicitly stored (see [86] and [11]).

To summarize, the Kalman filter problem can be stated in the following manner*: using all the observations, consistent with a set of vectors* $\mathbf{d}_i \quad i = 1, \ldots n$*, find for each* $n \geq 1$ *the state estimation* $\mathbf{w}_i$ *that minimizes the mean square error between the observed data and desired outputs of a system with state* $\mathbf{w}_i$ *([30]).*

In order to apply the Kalman filter to a neural network, the idea is to see the network as a non-linear discrete-time deterministic dynamic system whose states are given by the weights $\mathbf{w}(n)$.

2.5.2 Linear Kalman filter

The linear Kalman filter tries to estimate the state $\mathbf{w}(n) \in \mathbb{R}^m$ of a linear discrete-time dynamical system in which the following expression holds:

$$\mathbf{w}(n+1) = \mathbf{A}(n)\mathbf{w}(n) + \mathbf{B}(n)\mathbf{u}(n) + \omega(n) \quad \textbf{(Eq. 2.15)}$$

where $\mathbf{u}(n)$ is the input function of the system at time n, from which we get $\mathbf{d}(n) \in \mathbb{R}^n$ that

$$\mathbf{d}(n) = \mathbf{H}(n)\mathbf{w}(n) + \upsilon(n) \quad \textbf{(Eq. 2.16)}$$

being $\mathbf{A}(n)$, $\mathbf{B}(n)$ and $\mathbf{H}(n)$ known matrices, and $\omega(n)$ and $\upsilon(n)$ represent the noises of the process and measurements respectively. It is assumed they are white noises with zero mean, and diagonal covariance matrices $\mathbf{Q}(n)$ and $\mathbf{R}(n)$:

$$\mathbf{Q}(n) = \left\langle \omega(n)\omega^T(n) \right\rangle$$
$$\mathbf{R}(n) = \left\langle \upsilon(n)\upsilon^T(n) \right\rangle$$

In each iteration the filter uses the estimation of the current state and of the current covariance to find an a priori estimation of the next iteration (it "projects forward in time" the variance and covariance of the state estimation). In the next step, the real measurements are used to improve the estimation and get an a posteriori estimator. This process can also be seen as a prediction-correction cycle.

Let $\widehat{\mathbf{w}}_{-}(n)$ be an a priori estimator of the state for the current step n considering the previous historical knowledge:

$$\widehat{\mathbf{w}}_{-}(n) = \mathbf{A}(n)\mathbf{w}(n-1) + \mathbf{B}(n)\mathbf{u}(n-1) \ \textbf{(Eq. 2.17)}$$

The posteriori state estimation, $\widehat{\mathbf{w}}(n)$, is obtained as a linear combination of both the a priori estimation $\widehat{\mathbf{w}}_{-}(n)$ and the weighted difference between the real measurement $\mathbf{d}(n)$ and a prediction of the measure , $\mathbf{H}(n)\widehat{\mathbf{w}}_{-}(n)$:

$$\widehat{\mathbf{w}}(n) = \widehat{\mathbf{w}}_{-}(n) + \mathbf{K}(n)\left[\mathbf{d}(n) - \mathbf{H}(n)\widehat{\mathbf{w}}_{-}(n)\right]$$

The expression $\mathbf{d}(n) - \mathbf{H}(n)\widehat{\mathbf{w}}_{-}(n)$ is called ***residual or measure innovation***, and reflects the discrepancy between the predicted measurement and the real one. $\mathbf{K}$ is called the ***gain matrix*** for the Kalman filter.

Let us consider now the a priori and posteriori estimation errors:

$$\begin{aligned}\mathbf{e}_{-}(n) &= \mathbf{w}(n) - \widehat{\mathbf{w}}_{-}(n)\\ \mathbf{e}(n) &= \mathbf{w}(n) - \widehat{\mathbf{w}}(n)\end{aligned}$$

The covariances of the a priori and posteriori estimation errors are respectively:

$$\begin{aligned}\mathbf{P}_{-}(n) &= \left\langle \mathbf{e}_{-}(n)\mathbf{e}_{-}(n)^T \right\rangle\\ \mathbf{P}(n) &= \left\langle \mathbf{e}(n)\mathbf{e}(n)^T \right\rangle\end{aligned}$$

The gain matrix $\mathbf{K}$ is here chosen to minimize the posteriori covariance error. A possible choice is

$$\mathbf{K}(n) = \mathbf{P}_{-}(n)\mathbf{H}(n)^T\left[\mathbf{H}(n)\mathbf{P}_{-}(n)\mathbf{H}(n) + \mathbf{R}(n)\right]^{-1} \ \textbf{(Eq. 2.19)}$$

The covariance of the a priori estimation error is:

$$\mathbf{P}_{-}(n) = \mathbf{A}(n)\mathbf{P}(n-1)\mathbf{A}(n)^T + \mathbf{Q}(n) \ \textbf{(Eq. 2.20)}$$

whereas the covariance of the a posteriori estimation error is:

$$\mathbf{P}(n) = \left[\mathbf{I} - \mathbf{K}(n)\mathbf{H}(n)\right]\mathbf{P}_{-}(n) \ \textbf{(Eq. 2.21)}$$

The resulting algorithm for the linear case is then:

```
Initialize the diagonal elements of $\mathbf{P}(0)$, $\mathbf{Q}(1)$, $\mathbf{R}(1)$
Repeat for n= 1.2...
    • Calculate (Eq 2.17)
    • Calculate (Eq. 2.20)
    • Calculate (Eq. 2.19)
    • Calculate (Eq. 2.18)
    • Calculate (Eq. 2.21)
End repeat
```

The recursive nature of the filter makes the estimation of the system's state to depend of all historical measurements, but without an explicit consideration.

The filter's "performance" can be further improved by means of the matrices $\mathbf{Q}(n)$ and $\mathbf{R}(n)$. These matrices can be either chosen before the filter operation or modified dynamically. Hence, $\mathbf{R}(n)$ will be adjusted according to our confidence in the mechanism responsible of the measurements. On the other hand, $\mathbf{Q}(n)$ can be interpreted as the uncertainty in our model $\mathbf{w}(n+1) = \mathbf{A}(n)\mathbf{w}(n) + \mathbf{B}(n)\mathbf{u}(n) + \omega(n)$.

Since this algorithm propagates the covariance matrix, it is called covariance Kalman's filter. The a posteriori estimation for $\mathbf{P}$ might not be definite non-negative, because of the accumulated calculation errors (an unacceptable fact since it is a covariance matrix). Therefore, the "square root Kalman filter" has been suggested in order to propagate $\mathbf{P}$ (this is to use another equation instead of Eq. 2.21, see [30]).

2.5.2.1 Extended Kalman filter

Normally, the process to estimate or the measurement equation (corresponding to Eq. 2.15 and Eq. 2.16) are non-linear. In this case it is mandatory to include an approximation provided the linear Kalman filter is chosen. A Kalman filter that linearizes around the current mean and variance is called ***extended Kalman filter***. There are several proposals to apply a Kalman filter to non-linear systems. Here we will develop one of them.

Consider a system whose vector state $\mathbf{w}(n) \in \mathbb{R}^m$ respects the following equation:

$$\mathbf{w}(n+1) = f[\mathbf{w}(n), \mathbf{u}(n)] + \omega(n) \quad \textbf{(Eq. 2.22)}$$

with a measurement $\mathbf{d} \in \mathbb{R}^N$ that is

$$\mathbf{d}(n) = h[\mathbf{w}(n), \mathbf{u}(n)] + v(n) \quad \textbf{(Eq. 2.23)}$$

where the variables $\omega(n)$, $v(n)$ represent, as before additive white noises with zero mean, of the process and of the measurement, respectively and $\mathbf{Q}(n)$ and $\mathbf{R}(n)$ their covariance matrices. Functions $\mathbf{f}$ and $\mathbf{h}$ are non-linear, and relate the state at step n with the next one through the respective measurement $\mathbf{d}(n)$.

Through a linearization of the current estimation using the derivatives of the state and measurement functions we get a set of equations equivalent to the linear case. Such linearization is similar to a Taylor series around the current estimation using partial derivatives of the process equation Eq. 2.22 and of the measurement equation Eq. 2.23:

$$\begin{aligned}\mathbf{w}(n+1) &\approx \hat{\mathbf{w}}^-(n+1) + \mathbf{A}[\mathbf{w}(n) - \hat{\mathbf{w}}(n)] + \mathbf{W}\omega(n)\\ \mathbf{d}(n) &\approx \hat{\mathbf{d}}^-(n) + \mathbf{H}[\mathbf{w}(n) - \hat{\mathbf{w}}^-(n)] + \mathbf{V}v(n)\end{aligned}$$

(see [65] and [66] for further details). Hence, the a priori state estimation, $\widehat{\mathbf{w}}-(n)$ is now approximated making

$$\widehat{\mathbf{w}}-(n) = \mathbf{f}[\widehat{\mathbf{w}}(n-1), \mathbf{u}(n-1)] \quad \textbf{(Eq. 2.24)}$$

and the a priori covariance of the error is found with

$$\mathbf{P}-(n) = \mathbf{A}(n-1)\mathbf{P}(n-1)\mathbf{A}(n-1)^T + \mathbf{W}(n-1)\mathbf{Q}(n-1)\mathbf{W}(n-1)^T \quad \textbf{(Eq. 2.25)}$$

being **A** the Jacobian of **f** with respect to the state

$$\mathbf{A}(n) = \left\{ a_{ij} = \frac{\partial f_i[\widehat{\mathbf{w}}(n), \mathbf{u}(n)]}{\partial w_j} \quad i,j = 1,2\dots m \right\}$$

and **W** is the matrix with partial derivatives of **f** respect to the noise ω:

$$\mathbf{W}(n) = \left\{ b_{ij} = \frac{\partial f_i[\widehat{\mathbf{w}}(n), \mathbf{u}(n)]}{\partial \omega_j} \quad i,j = 1,2\dots m \right\}$$

The gain matrix $\mathbf{K}(n)$ can be obtained in this case from the following expression

$$\mathbf{K}(n) = \mathbf{P}\text{-}(n)\mathbf{H}(n)^T \left[\mathbf{H}(n)\mathbf{P}\text{-}(n)\mathbf{H}(n)^T + \mathbf{V}(n)\mathbf{R}(n)\mathbf{V}(n)^T\right]^{-1} \quad \textbf{(Eq. 2.26)}$$

where **H** is here the Jacobian with the partial derivatives of **h** with respect to the state, and **V** the one with partial derivatives of **h** with respect to[19] v:

$$\begin{cases} \mathbf{H}(n) = \left\{ h_{ij} = \frac{\partial h_i[\widehat{w}\text{-}(n), u(n)]}{\partial w_j} \quad i = 1,2\dots N \quad j = 1,2\dots m \right\} \\ \mathbf{V}(n) = \left\{ \mathrm{v}_{ij} = \frac{\partial h_i[\widehat{w}\text{-}(n), u(n)]}{\partial v_j} \quad i = 1,\dots N \; j = 1,2\dots m \right\} \end{cases}$$

$$\widehat{\mathbf{w}}(n) = \widehat{\mathbf{w}}\text{-}(n) + \mathbf{K}(n)\{\mathbf{d}(n) - \mathbf{h}[\widehat{\mathbf{w}}\text{-}(n), \mathbf{u}(n)]\} \quad \textbf{(Eq. 2.27)}$$

Finally, the covariance **P** for the a posteriori error has a similar form that the linear case (Eq. 2.21):

$$\mathbf{P}(n) = [\mathbf{I} - \mathbf{K}(n)\mathbf{H}(n)]\mathbf{P}^{-}(n) \quad \textbf{(Eq. 2.28)}$$

though it should be taken into account that $\mathbf{H}(n)$ is found in a different way than in the linear case.

The basic operation of the extended Kalman filter can be summarized through the following steps:

- *Initialize the elements of* $\mathbf{P}(0)$, $\mathbf{Q}(1)$, $\mathbf{R}(1)$
- ***Repeat*** *for* $n = 1, 2...$
  - *The state estimations are projected (predicted) and the covariance error from step* n *to* $n+1$ *using Eq. 2.24 and Eq. 2.25*
  - *The new a priori estimations are used to obtain the corrected estimations for the measurement* $\mathbf{d}(n)$. *Equations Eq. 2.27 and Eq. 2.28, in that order, give a posteriori state estimations and a posteriori covariance error.*
  - *Analogously to the linear case,* $\mathbf{R}(n)$ *and* $\mathbf{Q}(n)$ *are parameters of the algorithm that should be carefully adjusted in each step to get good results* ([65] and [66]).
- ***End repeat***

[19]Observation: while the elements v_{ij} are scalars, they should be denoted with italics to keep coherence with the rest of the work. In this equation they were written how they look to avoid confusion with the Greek letter v.
The a posteriori state estimation also uses **K** to weight the difference between the real measurement and the prediction:

2.5.2.2 The Global extended Kalman filter

When the Kalman filter is used to train neural networks (recurrent or not), the learning is regarded as a filtering problem, in which the optimal network parameters are estimated recursively from the filter equations [30]. The algorithm is specially useful for online learning situations, where the weights are continuously adjusted, though it can be employed in an off-line process as well (Feldkamp and Puskorius, 1994, cited by [65]).

The network state, denoted by $\mathbf{w}(n)$, for a certain input $\mathbf{u}(n)$, is precisely given by the weight values. The algorithm assumes the optimal weight value is stationary: $\mathbf{w}(n+1) = \mathbf{w}(n)$ after a minimum (either local or global) is reached . If we consider the network as a dynamic system, it can be seen that the equation describing the weight behavior is linear and respects the following expression:

$$\mathbf{w}(n+1) = \mathbf{A}(n)\mathbf{w}(n) + \mathbf{B}(n)\mathbf{u}(n) + \omega(n)$$

with $\mathbf{A}(n) = \mathbf{I}$, $\mathbf{B}(n) = \mathbf{0}$ and $\omega(n) = 0 \quad \forall n$ (we will consider this equality shortly). By "weights" we also refer to the neuron activation thresholds, such as previously agreed in the paragraph 2.1.1.1.

Therefore, this equation assumes the system is found in a stable optimum. The state corresponds with a local (or global) minimum of the error surface.

The measurement $\mathbf{d}(n)$ corresponds to the desired output of the neural network. This is the case of a non-linear equation, of the form:

$$\mathbf{d}(n) = \mathbf{h}[\mathbf{w}(n), \mathbf{u}(n)] + \upsilon(n) \quad \textbf{(Eq. 2.29)}$$

but with the shape

$$\mathbf{d}(n) = \mathbf{y}(n) + \upsilon(n) \quad \textbf{(Eq. 2.30)}$$

being $\mathbf{y}(n)$ the network output when $\mathbf{u}(n)$ is injected in the input.

Since the state equation is linear, the Kalman filter will use Equations (Eq. 2.17) and (Eq. 2.20) with $\mathbf{A}(n) = \mathbf{I}$, $\mathbf{B}(n) = \mathbf{0}$ and $\mathbf{Q}(n) = \mathbf{0}$ for all n. The non-linearity from Equation (Eq. 2.29) adds the remaining equations of the filter: the non-linear filter equations can be interpreted replacing $\mathbf{h}(n)$ by $\mathbf{y}(n)$ in (Eq. 30) when the network uses weights $\mathbf{w}(n)$.

The Jacobian $\mathbf{V}(n)$ used to find the matrix $\mathbf{K}$ normally equals the identity matrix, $\mathbf{V}(n) = \mathbf{I}$, as a result of the difficulty to correctly estimate its value. It is assumed, then, that its influence is in some way "hidden" inside $\mathbf{R}(n)$ (see [65]).

The partial derivatives that are within $\mathbf{H}(n)$ are normally found in an analogous way to that of BPTT or RTRL ([65] and [59]).

Summing up all the previous elements, we would get a full version of the so called ***global extended Kalman filter***. Nevertheless, for its real use as a training algorithm it is recommended to introduce some modifications. A justification for the denomination "global" is given next.

2.5.2.3 Decoupled extended Kalman filter

When we work with networks of certain size, the state vector $\mathbf{w}(n)$ can have a considerable number of components. That produces that the required calculations with matrices such as

$\mathbf{H}(n)$ requires a high quantity of computational resources, even for networks of modest size. The ***"decoupled extended Kalman filter"*** or deKf, reduces this complexity [30].

To achieve that the problem has a computationally tractable size, the dKef divides the network weights in g groups, obtaining corresponding state vectors $\mathrm{w}_i \quad i = 1, 2 \ldots g$. We will have so many groups as neurons within the network [20] and two weights will be in the same group whenever they are input of the same neuron. The decoupled version, as a consequence, applies the extended Kalman filter to each neuron independently in order to estimate the optimal value of the weights that reach it. In this manner, only the local inter-dependences are considered during the training process.

The main difference between the decoupled version and the global one is the substitution of the matrix $\mathbf{H}$ by g matrices with the following shape:

$$\mathbf{H}_k(n) = \left\{ h_{ij} = \frac{\partial y_i(n)}{\partial w_j^{(k)}} \quad i = 1, 2 \ldots N \quad j = 1, 2 \ldots m \right\}$$

where $w_j^{(k)}$ is the j-th weight from group k, and N is the number of neurons. We assumed m is the number of weights from group k. The matrix $\mathbf{H}(n)$ is then simply the concatenation of the matrices $\mathbf{H}_i(n)$:

$$\mathbf{H}(n) = (\mathbf{H}_1(n), \mathbf{H}_2(n), \ldots \mathbf{H}_g(n))$$

It can be observed that the deKf can be reduced to the global one when $g = 1$.

The corresponding decoupled algorithm is then:

1. *Let g=N, the number of neurons of the network*
2. *Initialize the network weights,* $\widehat{\mathbf{w}}_i(0) \quad i = 1, 2 \ldots g$
3. *Initialize the diagonal elements from* $\mathbf{R}(1)$ *and* $\mathbf{P}_i(0)\, i = 1.2...g$.
4. *For* $n = 1.2....i = 1.2...g$ *compute the following vectors:*

$$\begin{aligned}
&\widehat{\mathbf{w}}_i^-(n) = \widehat{\mathbf{w}}_i^-(n-1)\\
&\mathbf{P}_i^-(n) = \mathbf{P}_i^-(n-1)\\
&\mathbf{K}_i(n) = \mathbf{P}_i^-(n)\mathbf{H}_i(n)^T \left(\textstyle\sum_{j=1}^{g} \mathbf{H}_j(n)\mathbf{P}_{\mathbf{j}}^-(n)\mathbf{H}_j(n)^T + \mathbf{R}(n) \right)^{-1}\\
&\widehat{\mathbf{w}}_i(n) = \widehat{\mathbf{w}}_i^-(n) + \mathbf{K}_i(n)[\mathbf{d}(n) - \mathbf{y}(n)]\\
&\mathbf{P}_i(n) = [\mathbf{I} - \mathbf{K}_i(n)\mathbf{H}_i(n)]\mathbf{P}_i^-(n)
\end{aligned}$$

where $\mathbf{d}(n)$ *is the desired network output at step n and* $\mathbf{y}(n)$ *the real output for the corresponding input* $\mathbf{u}(n)$. *It is worth to notice that the two first equations are not necessarily the algorithm implementation, and we can work with the a posteriori estimations of the previous steps directly.*

5. *Update* $\mathbf{R}(n)$.

[20]This is not true for the case of LSTM networks. See [56]

2.5.2.4 Convergence

The non-linear structure of the deKf produces several numerical implementation difficulties, that make the iterations diverge from the correct solution. A way to avoid this divergence [30] is to add noise to the process equation, making $\omega(n) \neq 0$. The only change with respect to the previously described way for the deKf is the equation $\mathbf{P}_i(n) = [\mathbf{I} - \mathbf{K}_i(n)\mathbf{H}_i(n)]\mathbf{P}_i^-(n)$ from Step 4 of the algorithm, that turns into

$$\mathbf{P}_i(n) = [\mathbf{I} - \mathbf{K_i}(n)\mathbf{H_i}(n)]\mathbf{P}_i^-(n) + \mathbf{Q_i}(n)$$

Additionally, to avoid divergence, the introduction of $\mathbf{Q}_i(n)$ has the secondary effect in the probability reduction of being clamped in local minima. Normally the same matrix is used for all groups, so we will simply refer to $\mathbf{Q}(n)$. The initial values for $\mathbf{R}(1)$ and $\mathbf{Q}(1)$ are discussed next.

2.5.2.5 Initial parameters for the algorithm

The parameters to be adjusted at the beginning of the deKf are:

1. the initial value of the covariance matrix of the a posteriori error $\mathbf{P}_i(0)$; usually taken as $\mathbf{P}_i(\mathit{0}) = \delta\mathbf{I}$, where δ is a positive real constant and $\mathbf{I}$ the identity matrix.
2. the diagonal element of the initial covariance matrix of the noise of the measurement $\mathbf{R}(1)$; these elements are generally adjusted from a certain initial value and are gradually reduced when the training advances.
3. the diagonal elements of the covariance matrix of the error at the beginning of the process $\mathbf{Q}(1)$; also gradually reduced when the training is developed.

The elements adjustment for the covariance matrices consists of giving a certain value and then gradually reduce them according to a certain reduction rate *T*. For example, the values of the diagonal of $\mathbf{R}(0)$ are updated with a value R_{max} adjusted with a rate T until a terminal value R_{min} is reached.

A possible equation to apply this evolution is [65]:

$$R(n) = \frac{R_{max} - R_{min}}{e^{n/T}} + R_{min}$$

2.5.2.6 Computational cost

Since the deKf is based on the computation of the error derivative, which can be implemented both with BPTT or RTRL, the deKf computational cost is at least equal to the algorithm employed to find such derivatives. In [30] a comparison of storage use and necessary operations for RTRL, BPTT and deKf is given.

Finally, for MLPs networks it can be noticed that the global extended Kalman filter tends to converge in less iterations than the standard BP [59].

2.6 LSTM networks

In the case of recurrent networks formed only by non-linear neurons the meaningful events of the input sequence cannot be very distant from each other, because the errors flowing back with time either decay exponentially or grow up without bound (the "vanishing gradient problem", see 2.1.5 on page 34 and [34]). This limits recurrent networks to problems that only present jump with short time jumps (less than 10 time-steps) between the relevant inputs and the desired signals. Long short-term memory (LSTM) networks solve this problem by forcing a constant error flow, allowing LSTM networks to learn tasks that require to store information from relevant events for more than 1000 time-steps . These networks can be trained either with a gradient descent or a decoupled extended Kalman filter.

We will describe their architecture and general operation [34].

2.6.1 Architecture

The basic unit in a hidden layer of an LSTM network is the ***memory block***, that contains one or more ***memory cells*** and a pair of neurons that work as input and output gates for all the cells within the block. Each memory cell has in its core a linear neuron connected to itself called "Constant Error Carrousel" (CEC). This recurrent connection forces the error flow to be constant with time. Only that CEC carries the error trace when it flows back with time. The errors from other neurons behave as in other recurrent networks. In the absence of new inputs to the cell, the local error within a CEC flows back with time with no variation [23].

The scheme for a cell c is:

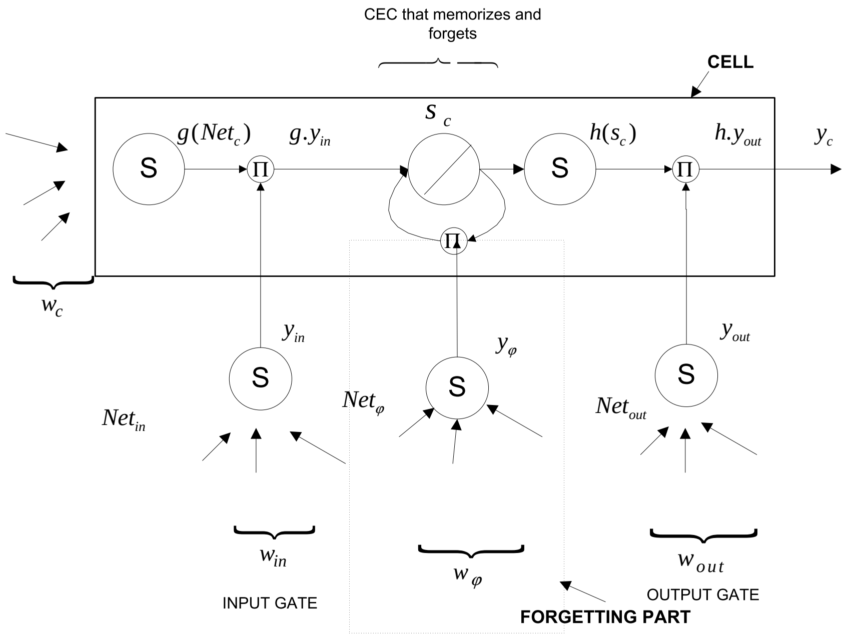


Figure 2.23

where the S means that the transfer function is sigmoid, with rank $[-1,1]$ in the case of function h, and with rank $[-2,2]$ in the case of g, whereas the diagonal line (/) represents a linear transfer function. The gates use logistic sigmoid functions within the rank $[0, 1]$. The cell is represented by the box in the solid line. The dotted box does not exist in cells that do not possess the capacity to forget (see below), and it is replaced by a recurrent connection in the linear neuron with unit weight (this is the case of a standard LSTM cell).

The input gate can be used to decide if a certain information is stored in the cell or not. Simultaneously, the output gate is placed to protect other neurons from the perturbations produced by irrelevant memory contents stored at the cell.

The error signals stored at the CEC cannot change, but can be superimposed with different error signals flowing within the cell (at different moments) by its output gate. Essentially, the gates open and close (for example, they close with a value y_{in} and y_{out} close to 0) and the access to the error flow is constant through the CEC.

These cells are grouped into memory blocks, and share the gates.

2.6.2 Training

LSTM networks can be trained either with gradient descend based algorithms or the deKf. In the first case, the algorithm proposed by Hochreiter is linear in both space and time, with less than ten operations in each step and neuron. On the other hand, the deKf allow us to speed-up convergence, though the long-term memory capacities are reduced (the length of the periods between correlated events tends to reduce) [70].

2.6.3 LSTM networks with reset

Sometimes the input values at a CEC, s_c, tend to increase linearly during the presentation of the time series, resulting in the neuron saturation with output function h. This makes a) the derivative of h to be practically null, blocking the new coming errors, and b) the output of a cell equals the "output gate", this is, the cell of the whole memory will degenerate in a common neuron (trainable with BPTT), leaving its memory role. The solution to this problem is given in [23] using "forget gates" (showed in dotted line in the Figure 2.21) that learn to "reset" memory blocks once it happens.

2.6.4 An example

An example of how to use a three-layer LSTM network with a memory block with two cells and recurrence limited to the hidden layer could be the one is shown in Figure 2.24:

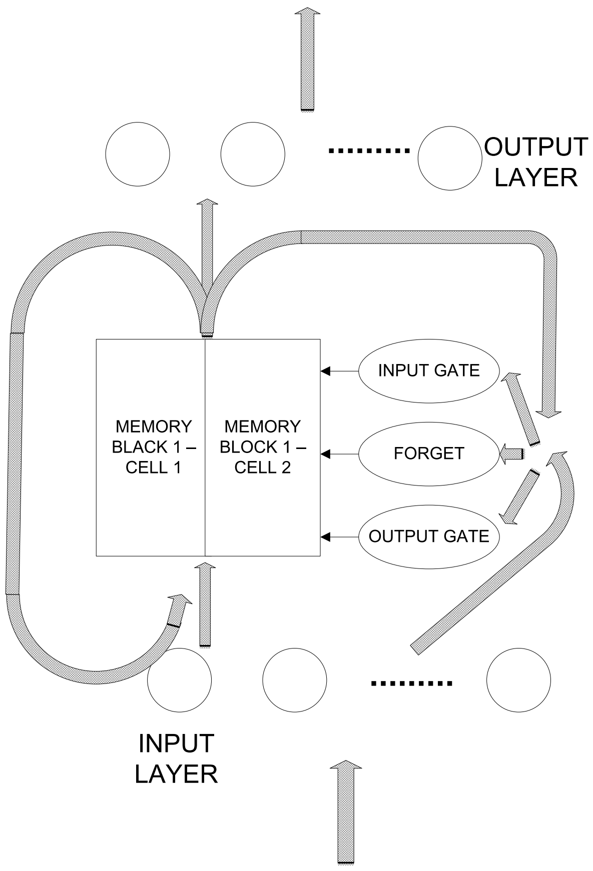


Figure 2.24

2.7 Model selection criteria: AIC and MDL

Different criteria can be employed to choose between several models. To start, a possible one is to select the model that produces the lowest average error over the training set, under equal conditions (the same number of epochs, percentage of patterns used for CV, etc.). However, the use of this criterion does not necessarily result in the lowest prediction error. We might instead use a subset of patterns for the training and use the remaining subset for "testing", to determine how well the model works for prediction, and choose the model with the lowest average error for this process. Even though this criterion is better than the first (takes into account the prediction error) it is completely empirical and does not have any theoretical foundation. It is also interesting to consider the linear correlation between the real and the desired outputs. When that correlation is high and positive, we know that the model will produce outputs with a similar behavior than the real ones. Furthermore, even if the "testing" error criterion makes a model preferable than other, the model with the highest correlation coefficient could be better. The prediction error, as well as the correlation coefficient and "testing" error, are provided by the chosen simulator.

Given that the prediction task of a time series is intended, an alternative criterion could be to choose the model whose error varies less in a long-term, this is, a ***robust*** model. In other words, if the model was trained with the patterns corresponding to the steps $1....p$, and predictions for steps $p+1$, $p+2$,...$p+k$ are performed (where k is variable), we choose the model for which k is maximum, respecting an error bound predetermined.

Given that the generalization capacity (prediction) of a model will depend on its complexity[21], we can summarize the previous criteria in two indicators directly provided by the simulator: the Akaike's information criterion (AIC) and the Rissanen's Minimun Description Length (MDL). More specifically, these indicators have into account the model complexity, the training performance (error) and the number of available patterns. There are several other criteria like the Moody's GPE ([49]), the BIC, etc. that are not detailed here for the sake of brevity, and because its discussion is out of our scope.

2.7.1 Akaike's Information Criterion (AIC)

Index deduction A quantification of the "distance" between two probability density functions can be their asymmetric divergence, relative entropy or Kullback-Leibler distance. The idea in the ***Akaike's criterion*** is to minimize this distance between the adjusted model and the true one [8].

Suppose we have a data set $y_1, y_2 \dots$ generated according to a certain "true" probability density function $f(\mathbf{y}|\ \theta_0)$ where θ is a k-dimensional parameter. Let $\Phi(k)$ be a family of parametric (empirical) densities, $\Phi(k) = \{\mathrm{f}(\mathbf{y}|\theta_k)|\theta_k \in \Omega(k) \subset \mathbb{R}^k\}$, θ_k^* the parametric estimation obtained maximizing the likelihood $f(\mathbf{y}|\ \theta)$ over $\Omega(k)$ and $f(\mathbf{y}|\ \theta_k^*)$ the resulting empirical density. Our goal is to find over the family set $F = \{\Phi(k_1), \Phi(k_2), \dots \Phi(k_L)\}$ the model $f(\mathbf{y}|\ \theta_k)\quad k \in \{k_1 \dots k_L\}$ that better fits $f(\mathbf{y}|\ \theta_0)$.

In order to determine which of the adjusted models $f(\mathbf{y}|\theta_{k_1}^*), f(\mathbf{y}|\theta_{k_2}^*) \dots f(\mathbf{y}|\theta_{k_L}^*)$ is the closest to $f(\mathbf{y}|\ \theta_0)$, we need a dissimilarity metric between the real model $f(\mathbf{y}|\ \theta_0)$ and an "approximator" $f(\mathbf{y}|\ \theta_*)$.

This metric will be the relative entropy between them: $d(\theta_0, \theta_*) = \langle Log[f(\mathbf{y}|\theta_0)/f(\mathbf{y}|\theta_*)]\rangle_{\mathbf{y}}$

If we denote $\delta(\theta_0, \theta_*) = \langle -2 \log f(\mathbf{y}|\theta_*)\rangle_{\mathbf{y}}$ we can write

$$2d(\theta_0, \theta_*) = \delta(\theta_0, \theta_*) - \delta(\theta_0, \theta_0)$$

[21]The generalization capacity is monotonically decreasing with complexity, probably because of overfitting.

Since $\delta(\theta_0, \theta_0)$ does not depend on θ_*, any classification of a models set based on $\delta(\theta_0, \theta_*)$ will be identical to the classification corresponding with $d(\theta_0, \theta_*)$. Therefore, in order to choose the model we can use $\delta(\theta_0, \theta_*)$ instead of $d(\theta_0, \theta_*)$.

Now, for a maximum likelihood estimator θ_k^* the exact divergence between the adjusted and the real model is given by $\delta(\theta_0, \theta_k^*)$, but to find $\delta(\theta_0, \theta_k^*)$ we would need to know the correct density $f(\mathbf{y}|\, \theta_0)$, which is obviously not possible. That is why Akaike proposed to use $-2\log f(\mathbf{y}|\theta_k^*)$ as a biased estimator of $\delta(\theta_0, \theta_k^*)$, where the bias can be approximated by $2k$. This approximation is valid provided

1. $f(\mathbf{y}|\theta_0) \in \Phi_i$ for some family Φ_i
2. a set of regularity conditions holds, to assure the asymptotic properties for the maximum likelihood estimator θ_k^* (see [8]).

This approximation for $\delta(\theta_0, \theta_k^*)$ is called ***Akaike's Information Criterion (AIC):***

$$AIC = -2\log f(\mathbf{y}|\theta_k^*) + 2k$$

If these two conditions holds, then $\langle AIC\rangle_{\mathrm{y}} \xrightarrow[sample\ size \text{ tends to } inifinite]{\text{asymptotically}} \langle\delta(\theta_0, \theta_k^*)\rangle_{\mathrm{y}}$, so, for big samples (data sets), the AIC-based model selection criterion should asymptotically conduct to the closest model of $f(\mathbf{y}|\, \theta_0)$ in the Kullback-Leibler sense. As a consequence, the use of AIC for small data sets is limited. For that reason the ***second-order AIC*** (or corrected AIC) is used instead:

$$AICc = AIC + \tfrac{2k(k+1)}{N-k-1}$$

Where N is the number of data points and k the number of model parameters. This approximation holds only when $N \geq 2 + k$ [51].

Particularly, if we assume that the errors (differences between the model output and the real values) are normally distributed, then the AIC can be approximated by

$$AIC = N\log\left(\frac{\sum_{i=1,N}(y_i - y_i^*)^2}{N}\right) + 2(k+1)$$

where N is the number of data points, the logarithms are natural and y_i^* represent the real values corresponding to the values y_i estimated by the model [51]. With this formula, the AIC depends on the chosen values to express the data, so an isolated AIC cannot have interpretation. Instead, the difference of the results obtained for different models over the same data is relevant. Finally, it can be proved that the probability to choose the correct model is

$$P = \frac{e^{-0.5\alpha}}{1 + e^{-0.5\alpha}}$$

being α the AICc differences [51]

Use In our application, the neural network simulator computes:

$$\begin{aligned} MSE &= \frac{\sum_{j=0}^{p}\sum_{i=0}^{N}(d_{ij}-y_{ij})^2}{Np} \\ AIC(k) &= N\log(MSE) + 2k \end{aligned}$$

where N is the number of input patterns, p the number of output neurons k the weights in the network.

Sub-models and AIC application Consider two parametric models $\mathfrak{M}_1(\mathbf{y}|\mathbf{x},\theta_1)$ and $\mathfrak{M}_2(\mathbf{y}|\mathbf{x},\theta_2)$ with $\theta_1 \in \mathbb{R}^{m_1}$ $\theta_2 \in \mathbb{R}^{m_2}$ $m_1 < m_2$, being $\mathbf{y}$ the output data for the input $\mathbf{x}$. We say that $\mathfrak{M}_1(\mathbf{y}|\mathbf{x},\theta_1)$ is a sub-model of $\mathfrak{M}_2(\mathbf{y}|\mathbf{x},\theta_2)$, and we denote it with

$$\mathfrak{M}_1(\mathbf{y}|\mathbf{x},\theta_1) \subset \mathfrak{M}_2(\mathbf{y}|\mathbf{x},\theta_2)$$

if by restricting some components of θ_2 (or relations of its components) we get the model $\mathfrak{M}_1(\mathbf{y}|\mathbf{x},\theta_1)$. For example, in the case of multilayer networks, the number of input and output neurons are the same in both models, but if the number of units in the hidden layer is bigger in the second model ($\mathfrak{M}_2(\mathbf{y}|\mathbf{x},\theta_2)$), taking the weights that arrive/reach to the additional neurons as zero we obtain the model $\mathfrak{M}_1(\mathbf{y}|\mathbf{x},\theta_1)$.

It can be proved that the AIC index and its generalizations (such as the NIC - Network Information Criterion) are only applicable in hierarchical models (where an inclusion relation holds) such as the previous case [53].

2.7.2 Minimun Description Length (MDL)

2.7.2.1 Introduction

The complexity of a model is determined by two independent causes: the number of free parameters and the functional way in which those parameters are combined. For example, intuitively the models $y = \theta x$ and $y = x^\theta$ have different complexities. When these dimensions vary, different improvements in the fit of the model to the data are achieved, not necessarily meaning an enhance of the generalization capacity. The chosen model should be complex enough to describe the data accurately, but without reaching an overfitting, hence loosing its generalization capacity. The MDL was defined by Rissanen (1978), associated with the data encoding, and it allows us to choose between several models, considering its parameters and functional way which they are related.

Suppose we want to transmit a message D from a source to a destination, with minimum length (where the length could be, for example, the number of encoding bits). A possible approach is to try a static encoding for the message D, assuming the source and destination are independent. However, if there are aspects in the data that are repeated systematically and that are not previously known by the receiver, we could expect to transmit a shorter message if we first detail a model specification $\mathfrak{M}$ that captures those aspects, using a message with length L($\mathfrak{M}$) and afterward a second message explaining how the real data differ from the one predicted by the model $\mathfrak{M}$. We could see L($\mathfrak{M}$) as a model complexity metric, since a more complex model would need more information to describe it. The necessary message to send the information of the discrepancy has a length given by L($D|\mathfrak{M}$), and can be considered as an error term. Therefore, the total message length is

$$\textit{Description length} = \underbrace{\mathrm{L}(D|\mathfrak{M})}_{error} + \underbrace{\mathrm{L}(\mathfrak{M})}_{complexity}$$

The objective of choosing the ***minimum description length*** (MDL) leads naturally to an instance of the Occam's razor: a very simple model will be a poor predictor, so the errors will be big, leading to a big additive error term on the right hand. If we use instead a much more complex model a lower error term will be produced, but if it is too complex an extremely high amount of information will be needed to describe it, making the complexity term high in the right hand side again. Intuitively we hope the minimum description length will occur when the model $\mathfrak{M}$ gives an exact representation for the process that generated the data, and also hope this model to have the best generalization properties, in average.

2.7.2.2 Formal definition

Given the vector with parameters θ and the observed data $\mathbf{y}$ represented by the random variable Y, the Fisher matrix $\mathcal{I}(\theta)$ is defined by

$$\mathcal{I}(\theta) = \left\{ \mathcal{I}_{ij}(\theta) = \left\langle \frac{\partial Log\ f(y|\theta)}{\partial \theta_i} \frac{\partial Log\ f(y|\theta)}{\partial \theta_j} \right\rangle_Y \right\}$$

being $P(\mathbf{y}|\theta)$ the probability density function for Y (see [19]).

Let $\Phi = \{ \mathit{\Phi}(y,\theta) \quad \theta \in \mathbb{R}^k \ y \in \mathbb{R}^m \}$ be a family of parametric models with parameter $\theta \in \mathbb{R}^k$ such that each one has an associated likelihood function $\varphi(\mathbf{y}|\theta)$ for a set of observed data $\mathbf{y}$. For this family, we have by definition [54]:

$$MDL = -\underbrace{Log\varphi(\mathbf{y}|\theta^*)}_{A} + \underbrace{\frac{k}{2} Log\left(\frac{N}{2\pi}\right)}_{B} + \underbrace{Log \int d\theta \sqrt{\det(\mathcal{I}(\theta))}}_{C}$$

where

Log is the natural logarithm

θ^* is the maximum likelihood estimation for θ

k is the number of parameters for the model

N is the sample size

$I(\theta)$ is the Fisher information matrix

Interpretation

The term A gives, from the Log likelihood, the model goodness of fit of the model respect to the data. The terms B and C give us the intrinsic model complexity: B measures the size of the problem whereas C gives a complexity idea associated with its functional form by means of the Fisher matrix $\mathcal{I}(\theta)$ [54]. Additionally, these terms have a geometric interpretation related with Riemann's spaces [54].

It can be proved that to minimize the MDL is the same to maximize the Bayesian probability $P_\varphi = P(\varphi|\mathbf{y})$, this is, maximize the probability that the "real" model $\mathit{\Phi}_0$ (that corresponds with the density where the observed data $\mathbf{y}$ is generated) is inside the family Φ [54].

When we refer to the calculus of the MDL, the chosen neural network simulator estimates it by

$$MDL(k) = N Log MSE + \frac{k}{2} Log N$$

where:

- N is the number of pattern
- $MSE = \frac{1}{Np} \sum_{j=0}^{p} \sum_{i=0}^{N} (d_{ij} - y_{ij})^2$
- y_{ij} the network output for pattern i from neuron j
- d_{ij} the desired output for pattern i from neuron j
- p the number of output neurons

The reader can find valuable connections between MDL, information theory and statistics in [28].

2.8 Reduction of the network´s degrees of freedom

In order to address problems that require large networks it is necessary to minimize the network size. Additionally, it is less probable that a network of minimum size fall in overfitting (that learns the noise or the idiosyncrasy of the data) and hence it will generalize better. The optimum size can be achieved by several manners:

- choosing the number of weights (no matter their values) and the neurons the network has
- letting the network grow (***"network growing"***): we start with a small network and include neurons in the hidden layer, progressively, until the desired performance is achieved. Marchand, Lebiere and others developed this kind of algorithms ([29]). We will not cover this methodology here.
- pruning the network (***"network pruning"***): we start here with a big network (typically, a multilayer perceptron) with an adequate level of performance for the problem to be solved, and the network is pruned by means of weight deletion through a selective and ordered manner.
- imposing certain restrictions to the weights so they are related and the network degrees of freedom are reduced (such as in "weight sharing").

In the following paragraphs we will revisit pruning and weight sharing methods.

2.8.1 Pruning techniques

Weight decay One of the most simple pruning algorithms is through weight decay. Each weight decays to 0 within a rate proportional to its magnitude so some connections disappear[22] unless they are reinforced (used in the learning stage).

The ***weight decay*** in the weight update equations can be implemented adding a regularization term to the function E that penalizes big weights:

$$J(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\sum_{i=1}^{n} w_i^2$$

where $0 < \lambda < 1$ and n the number of weights. If a gradient descent search is performed to look for the minimum of $J(\mathbf{w})$ we get the update rule:

$$\Delta w_i = -\rho\frac{\partial J}{\partial w_i} = -\rho\frac{\partial E}{\partial w_i} - \rho\lambda w_i$$

showing an exponential decay for w_i if there is no learning stage (that is, if before the application of the degradation $\Delta w_i = -\rho\frac{\partial E}{\partial w_i} \approx 0$ we replace and get $\Delta w_i \approx -\rho\lambda w_i(t)$ showing the mentioned variation).

The function $J(\mathbf{w})$ discourages the use of big weights, since a single weight with big value "costs" much more than several small weights: if two connections reach a neuron with possible weights w and 0 or $w/2$ and $w/2$, it will be desirable the second case with two equal weights since $\left(\frac{w}{2}\right)^2 + \left(\frac{w}{2}\right)^2 < w^2 + 0^2$. This leads to delete big weights, even though they are needed to model the data.

[22]The weight turns so small that it can be taken as zero, so the connection associated with it disappears.

Other technique to delete weights was proposed by Weigend (cited by [29] and [30]) is based on taking

$$J(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\sum_{i=1}^{n}\left(1+\frac{w_i^2}{w_0^2}\right)^{-1}\frac{w_i^2}{w_0^2}$$

being n the number of weights and $0 < \lambda < 1$, in which the penalty term helps to regulate the weight magnitudes, and w_0 is a free positive parameter that should be determined. For big values of w_0 this technique is equivalent to the previous weight decay, so its tends to have several small weights. On the other hand, if w_0 is small, less big weights are favored. It is worth to notice that when $|w_i| \gg w_0$, the term cost $\frac{w_i^2}{w_0^2}\left(1+\frac{w_i^2}{w_0^2}\right)^{-1}$ tends to 1 (so its contribution in $J(\mathbf{w})$ tends to $\lambda/2$), justifying the interpretation of this term as a penalty to use big weights. in practice w_0 is close to unit. The weight deletion is highly sensible to the penalty factor λ. Some authors look for the best factor λ by means of statistical methods (Wahba, cited by [69]).

Last, an alternative function can be considered [69]

$$J = E + \lambda\sum_i |w_i|$$

The previous ideas have been applied to the concept of neurons elimination. It would start with an excess of hidden neurons and tsome of them would be dynamically discarded (the redundant ones, with all null weights).

Pruning techniques based on the Hessian matrix These techniques are based on the second-order partial derivatives of the error function to simplify the network. They start with a trained network, and try to identify the parameters (weights) such that if they are deleted from the network, the error function is increased minimally. There are two algorithms inspired in this idea: "***optimal brain damage***" (OBD) of Le Cun, that assumes that the Hessian matrix is diagonal, and the "***optimal brain surgeon***" (OBS) that does not require the Hessian to be diagonal. A thorough description of OBS can be found in [30].

2.8.2 Regularization theory

Given the relation $\mathbf{F}(\mathbf{x}) = \mathbf{y}$ where $\mathbf{F}$ and $\mathbf{y}$ are known but $\mathbf{x}$ must be found, the problem is said to be "ill posed" when a small change in the dependant variables produces an enormous change in the solution. The regularization theory was proposed by Tikhonov and Arsenin in order to face ill-posed problems. For example, when we try to determine the correct number of neurons the network should have and we only have information restricted to the training set, we have an ill-posed problem, since we do not have access to the performance in the testing set [69]. The basic idea of the regularization theory is to modify the optimization problem turning it more restrictive so its solution has less variability. A ***regularization term*** is then added to the error function. The obtained error function is then

$$E_{new} = E + \lambda E_r$$

where E is the original error function, E_r is the regularization and λ the ***regularization constant***) that regulates the influence of the regularization versus the original error E, and is experimentally determined. Tikhonov regulators (E_r) try to find more regular solutions for the optimization problem, hence they penalize the curvature of the original solution (see [48]). When these techniques are introduced in a training algorithm, it is used to choose regularization functions which allow an efficient computation of its derivatives with respect to the different weights, for example:

$$E_r = \sum_{i=1}^{n} w_i^2$$

where n is the number of weights. A first-order regularization term can be useful as well [69].

The regularization is closely related with the weight decay and the optimal brain damage. Indeed, as we recently saw, weight decay is equivalent to add a regularization term that is a function of the weights:

$$E_{new} = E + \lambda \sum_{i=1}^{N} w_i^2.$$

2.8.3 Weight sharing and Soft Weight Sharing

Weight sharing is a method where several weights are controlled by a single parameter (this is $p = w_k = w_c = ...$), which helps to improve the generalization, reducing the number of free weights in the network [29]. Normally, it is very hard to decide which weights should be equal unless the network topology for the specific application is known beforehand.

Another way to reduce the number of independent weights is grouping them into “clusters” of values, where each weight is normally distributed with unknown mean and variance, but they will be adjusted when the training process is performed. Nowlan and Hinton [56] found that this returns better results than using as a regularization term the sum of its squares. This is equivalent to add an adequate regularization term to the error function to obtain an automatic weight sharing method. The regularization term is in this case

$$E_R = -\sum_i \ln \left[\sum_j \alpha_j P_j(w_i) \right]$$

where each $P_j(w_i)$ is a normal density with mean μ_j and variance σ_j, the factors α_j represent mixture proportions of normal densities $P_j(w_i)$ with $\sum_j \alpha_j = 1$, and w_i represents an arbitrary network weight. It is assumed that the parameters α_j, μ_j and σ_j are adapted while the network learns. The use of multiple adaptive normal distributions allows the implementation of a ***soft weight sharing***, where the learning algorithm decides by itself which weights should be linked together in the same cluster. If all distributions begin with high variance, the starting weight grouping in subsets will be very weak (or soft). The groupings turn dissimilar when the network progressively learns and simultaneously the variances are decreased. Therefore, the groupings converge to certain subsets which depend on the particular task that is being learned.[29].

To summarize, the correction term leads to a non-supervised weight clustering (weight sharing) that will be regulated according to the characteristics of the training set ([56] and [29]).

2.9 Introduction to dynamical systems

We present here an overview of the theoretical foundations of dynamical systems, which will help us to have an interpretation of the dynamic networks as well as to train networks with a reduced input dimensionality, thanks to the application of the Takens-Mañé's theorem. The bibliography of this section is based on [30] and [31] in this case for deterministic systems, while in several publications timely mentioned in the stochastic case [38].

2.9.1 State-space model

A system will be called ***dynamic*** if its outputs are not only function of its current inputs, but the previous ones [43]. Formally (and more restrictively) Boccara defines: "Consider a system with a ***state-space*** S whose elements represent possible states of the system, a ***time*** t (that can be either discrete or continuous) and an ***evolution law*** (a rule that gives the state of the system at time t from the knowledge of the history). A system like this, where the evolution can be described in terms of a non-linear set of differential equations is a dynamical system" [12]. Some authors ([9]) define a dynamical system as a pair (S, F) where S is a non-empty set (the "state-space") and F a function (called the evolution law): $F : S \to S$. Therefore, if x_t is the system state at time t, then [23]

$$x_{t+1} = F(x_t)$$

is the system state at time $t + 1$ [9]. Generally S is a metric space. Using this ***state-space model*** we can represent a system as a set of N state variables that, given their values in any instant, permit to predict the future evolution of the system. The number of variables involved is called the ***order of the system***. Using this model we can say, formally, that a dynamical system is a model where its state variables vary with time for a certain input [30].

A system is either ***autonomous*** if its evolution law does not depend explicitly on time, or ***non-autonomous*** otherwise. A non-autonomous system can always be described by an autonomous system with higher dimensionality [12].

If time is a continuous variable, the dynamical system will be described by means of differential equations of the form

$$\frac{dx}{dt} = F(x(t))$$

whereas in the discrete-time case it will be

$$x_{t+1} = F(x_t)$$

as previously defined.

In our case study we are concerned with discrete-time autonomous systems.

[23] In the next paragraphs we will be using the italics notation for x, F, etc., even though they are vectors, in order to be coherent with the notation used in the bibliography

2.9.1.1 State-space

Given an instant t, the system state can be represented by a single point $P(t)$ in an N-dimensional space. This space can be either euclidean or not, even though our interest is focused here on the euclidean case. Such space is called ***state-space***. The curve described by those points $P(t)$ is called the system's ***trajectory*** (or orbit).

The mathematical expression for the state change with time can assume several forms, depending on the system to be modeled, but a basic classification is deterministic or stochastic.

Deterministic approach A dynamical system can be either deterministic or stochastic. In the present context, we will refer to a deterministic system when the past observations can lead to determine the current state with no error, this is, we refer to "operationally deterministic", so it can be modeled reasonably well deterministically given the past knowledge without regarding its ultimate deterministic or stochastic nature. In this case the states change deterministically from the current ones and possibly some exogenous variables. These systems are called ***deterministic dynamical systems***. For example, if the system is not influenced by any exogenous inputs, we will write:

$$x_j(n+1) = F_j[x(n)] \qquad j = 1, 2 \dots N$$

being

- N the system order
- $x(n) = [x_1, \dots x_N]$ the system state
- $F_j(\bullet)$ a non-linear function that does not explicitly depend on n and is non-zero for some n. This characteristic of the functions F_j holds for all dynamical systems that are considered in this work.

This equation can be written in its vectorial form:

$$x(n+1) = F[x(n)]$$

If instead the system is influenced by a set of exogenous factors $u(n) = [u_1(n), u_2(n) \dots u_M(n)]$ it will then be:

$$x(n+1) = F[x(n), u(n)]$$

Stochastic approach "A stochastic (or random) dynamical system is described by a tern (S, Γ, Q) where S is the state-space, Γ a family of operators in S (interpreted as the whole admissible movement laws) and Q a probability distribution on Γ. The system evolution is informally described in the following manner: initially, the system is in some state $x_0 \in S$; an element α_1 is randomly chosen (by Fortune) according to the distribution Q and the system turns to a state $x_1 = \alpha_1(x)$. Again, independently of α_1, Fortune chooses α_2 from Γ using the same distribution Q and the system state at step 2 is $x_2 = \alpha_2(x_1)$, and this process is repeated. The initial state x_0 can be a random variable X_0 chosen independently of the operators α_i" [9]. In other words, when the system states change in such a way that the next state depends on the current one, the inputs and a random vector:

$$x(n+1) = F[x(n), u(n), \omega(n)]$$

being $\omega(n)$ a random vector (called ***dynamic noise***) and $\mathbf{F}$ a vector with non-linear functions, we have a ***dynamical stochastic (or random) system*** [24]. The general approach includes systems with exogenous inputs that though here were explicitly stated, can be considered assuming unpredictable values and hence be included in $\omega(n)$, and systems that are sampled in irregular time intervals [80]. The variation space for $\omega(n)$ is called ***shift space*** and the noise is said to be a ***shift.*** In its most general case that noise varies over a compact manifold (see below).

In the case of stochastic dynamical systems, we can only make a statistical prediction of the following system states.

Stochastic dynamical systems are interesting for several reasons. For example, if we wish to model a deterministic system with high order, we can model it by means of a stochastic dynamical system then gaining robustness. Additionally, in some economical, financial (or even physical) applications it is sometimes necessary to model systems that tend to have both a deterministic component and a stochastic one. When $\omega(n)$ can only have a finite number of results, this is, where in each instant, $\mathbf{F}$ is chosen within certain finite set $\{F_1, F_2 \dots F_Z\}$ an ***iterated functions system*** (IFS) is obtained. In this case, shift space dimension equals 0.

Additionally, what we do know from the system (whatever its type) is a series of observations of a variable x such as

$$x(n+1) = \varphi[x(n), x(n-1), \dots x(n-T)] + \delta(n)$$

being φ a certain function and δ noise, called ***observational noise***.

2.9.1.2 Equilibrium states

In the deterministic case, a state vector $\overline{\mathbf{x}}$ is called ***of equilibrium*** (or steady state) provided $F(\overline{\mathbf{x}}) = \overline{\mathbf{x}}$. Observe the trajectory degenerates in the equilibrium point itself whenever it is reached. If the system has inputs $\mathbf{u}(n)$ the steady state must verify that

$$F(\overline{\mathbf{x}}, \mathbf{u}(n)) = \overline{\mathbf{x}}.$$

2.9.1.3 Stability

For a deterministic dynamical system, there are several definitions of stability. An equilibrium point $\overline{\mathbf{x}}$ is

1. ***uniformly stable*** if $\forall \varepsilon > 0 \quad \exists \delta > 0 \ : \ \|\mathbf{x}(0) - \overline{\mathbf{x}}\| < \delta \Rightarrow \|\mathbf{x}(n) - \overline{\mathbf{x}}\| < \varepsilon \quad \forall n > 0$

The trajectory for the system rests eventually close to the equilibrium point after some finite time with any desired proximity, provided the starting point $\mathbf{x}(0)$ is close enough to $\overline{\mathbf{x}}$.

2. ***asymptotically stable*** if both condition 1) holds and $\exists \delta > 0 \ : \ \|\mathbf{x}(0) - \overline{\mathbf{x}}\| < \delta \Rightarrow \lim_{n \to \infty} \mathbf{x}(n) = \overline{\mathbf{x}}$. That is, if the starting point $\mathbf{x}(0)$ is close enough to $\overline{\mathbf{x}}$, then the trajectory $\mathbf{x}(n)$ will converge to the equilibrium point $\overline{\mathbf{x}}$.

[24] When the evolution of the system is non-deterministic and only the transition probabilities to change from one state to another is known for each instant t instead, we have a ***random process***: a family of measurable mappings over the space Ω of possible outcomes, in the state-space X. As a consequence, this state-space must be measurable. Formally we can say that a dynamical system is stochastic if its state-space is a metric measurable space, whereas it is deterministic if the state-space is just a metric space [12].

3. ***Globally asymptotically stable*** provided conditions 1) and 2) hold for all the possible starting point $\mathbf{x}(0)$.

This definition implies in particular that the system cannot have other equilibrium points, and needs that all the trajectories are bounded. In other words, global asymptotic stability implies the system will reach the equilibrium point no matter the starting point [30].

These definitions do not take into consideration the system inputs (external stimuli) so they are applicable when those stimuli are kept (almost) constant in time.

2.9.1.4 Dissipative Systems

A deterministic dynamical system is ***dissipative*** provided

$$\int_S [\mathbf{F}(\mathbf{x}) \cdot \mathbf{n}] dS \ < \ 0 \quad \forall \mathbf{x} \in S$$

being S an oriented, compact and closed surface contained in the state-space and $\mathbf{n}$ the normal vector in each point of S (pointing outside).

In dissipative systems, the volumes for starting conditions tend to contract with time.

2.9.1.5 Reconstruction space

Consider a deterministic dynamical system where the time series $x(n)$ of observations is known here and we always know the N-1 previous values $x(n\text{-}1)$,...$x(n\text{-}N\text{+}1)$, being N not necessarily the system order, and let $\mathbf{x}(n) = [x(n-1), ...x(n-N+1)]$. The space whose dimensions are $x_1 \equiv x(n-1), x_2 \equiv x(n-2) \ldots x_{N-1} \equiv x(n-N+1)$[25] is by definition the ***reconstruction space*** [69]. The points determined by each one of those coordinated vectors over the reconstruction space follow a curve called ***reconstructed trajectory*** for the series[26]. The vectors $x(p) = [x(p-1), ...x(p-N+1)]$ and $[x(p), ...x(p-N+2)]$ that are obtained in the reconstruction space when we move p are correlated by the structure of the time series $x(n)$. Figure 2.25 shows an example for the case $N = 3$.

[25] x_1 is not the same as $x(1)$: the idea is that there is one dimension for each component of the vector x. *The axis* x_1 can be regarded as the x, the x_2 *as the* y, *etc.*

[26] In fact it is the trajectory reconstructed from the system using the available data from the series, but, as in other cases, we refer to the reconstructed trajectory for the series, with an abuse of language.

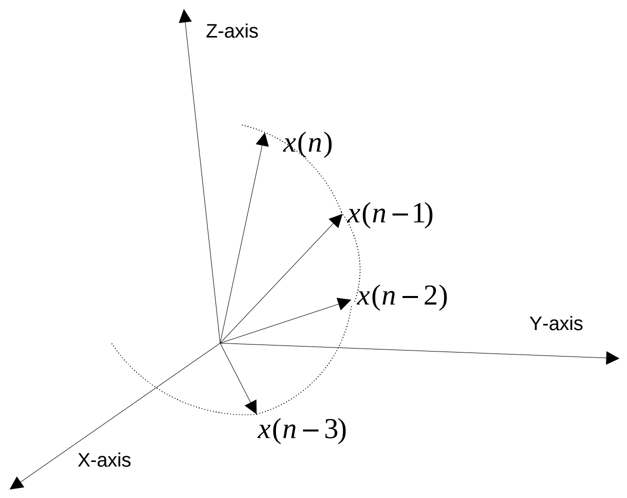


Figure 2.25

The structure of the time series will restrict the possible placements for the mentioned vectors $\mathbf{x}(p)$, restricting the trajectory for the series to a much lower dimension than the reconstruction one. This subspace is called the ***signal subspace***[27]. A great dimensionality reduction can be obtained in this way: for example, if $x(n)$ is periodic the number of necessary dimensions of the reconstruction space equals the period (in samples), mean while, given this trajectory will be a closed curve (because $x(n)$ is periodic), much less dimensions will be required to represent this trajectory. For example, if $x(n)$ is a sinusoid it is associated with an elliptic trajectory. No matter the length of the period, the ellipse can be represented with two dimensions. On the other hand, a random noise does not have any temporal structure and fills any space of arbitrary dimension. The same happens with static data patterns: we need a reconstruction space of maximum dimension (the dimension of $\mathbf{x}(n)$), since we do not know a priori any relation between the data components. If the dimensions of the reconstruction space are well chosen, the reconstructed trajectory does not cut itself, and there exists a one-to-one correspondence between the real trajectory $x(n)$, and the reconstructed one. The dimension of the reconstruction space depends on the particular application of the reconstructed trajectory. For example, in the previous case, if we reconstruct the signal trajectory only from the axis x and y, we would obtain a curve (trajectory) B (see Figure 2.26), with other properties. This signal subspace (so called by [69]) corresponds to the embedding mentioned throughout the rest of this work. We sticked to the most common denomination in the literature about this topic.

[27]The term signal is here inherited from digital signal processing applications. In the current context we could call it "series subspace", but we prefer to stick to the terminology used in the related literature.

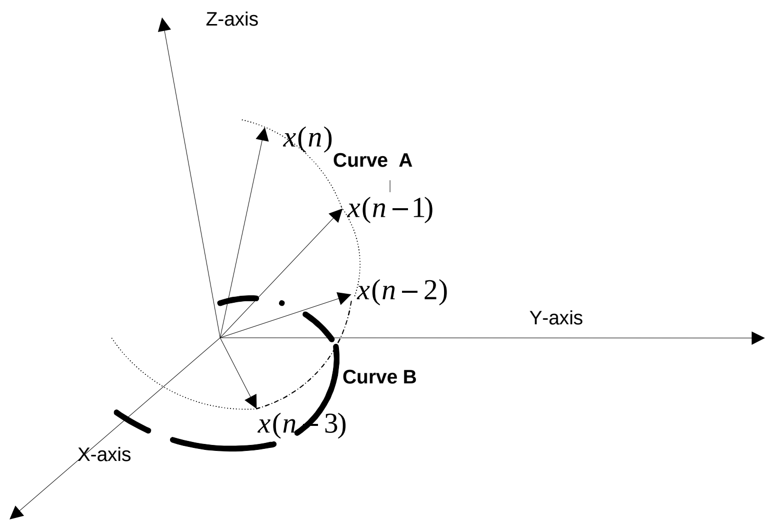


Figure 2.26

We will see later that this reduction in the dimension of the reconstruction space can be used to reduce the input dimensionality of the neural network to be implemented. In the case of a stochastic dynamical system it is proved that the situation is basically the same, and there is a bijective correspondence between the series and the points of the reconstructed trajectory, for each possible value of ω (excepting, perhaps for a finite values then ω) [80].

2.9.2 Attractors and manifolds

2.9.2.1 Manifolds

The concept of manifold is commonly used in the descriptions of dynamical systems. In simple terms, a ***manifold*** is a k-dimensional surface embedded[28] in the state-space. A manifold is a non-euclidean space that generalizes the notion of curves and surface representation in $\mathbb{R}^n$ [12].

Definition: A map $f : X \to Y$ is called a diffeomorfism if f carries X homeomorfically into Y and if both f and f^{-1} have continuous partial derivatives of any order [46].

Definition: $\mathfrak{M}$ is a n-dimensional manifold if, for all point x in $\mathfrak{M}$ there is some neighborhood $N(x) \subseteq \mathfrak{M}$ that contains x and exists a diffeomorphism $h : N(x) \to \mathbb{R}^n$ that maps $N(x)$ in a open subset of $\mathbb{R}^n$ [78][46].

In dissipative systems[29] it is common to find "manifolds" of states that attract the trajectories that pass through a neighborhood of it, called ***attractors***. These attractors are bounded sets of the state-space where the volume starting points converge[30]. The attractor can be either a single point and is therefore a ***point attractor*** , a periodic trajectory or ***limit cycle*** that is stable in the sense that closed trajectories asymptotically approximate to it, or a ***strange attractor***.

[28]In what follows we will see "The notion of embedding"

[29]The theory is developed assuming the system will keep the same inputs, whenever it has anyone (or they have non-significant variation) during the time we let the system evolve towards a stable state.

[30]The starting conditions here include the possible inputs in the starting point.

The idea of attractor can be generalized for stochastic dynamical systems. In this case, for an attractor we can think that trajectories will eventually enter a neighborhood of it with unit probability when $t \to \infty$.

Each attractor has an associated region, called ***attraction domain,*** such that every starting point inside it makes the evolution convergent towards that attractor.

Let A be the Jacobian of $\mathbf{F}$ evaluated at $\overline{\mathbf{x}}$. If all the eigenvalues of A are lower than one in magnitude, $\overline{\mathbf{x}}$ is called ***hyperbolic.*** We will see the meaning of these attractors to study training problems associated with recurrent neural networks.

A dynamical system can have several attractors. Starting from a certain point, the system trajectory will either converge to one of them or not. The conditions for a dynamical system to have convergence are called Lyapunov's conditions [42].

In general, we have a set of observations (measurements) of a certain system variable, that have evolved through time towards one attractor, so the problem focuses on the study of "the" attractor, this is, the attractor to which the observations are getting closer to.

2.9.2.2 Strange Attractors and chaos

Strange attractors are characteristic in deterministic dynamical systems of order higher than 2. A system with a strange attractor exhibits a chaotic behavior: for starting points inside the attraction domain of that attractor, the behavior is deterministic (it behavior is governed by fixed rules) but that behavior is so complex that is seems random. We define a ***strange attractor*** as an attractor such that the orbits that visit the attraction domain and have similar starting conditions tend to separate with time. A system with a strange attractor is said to be ***chaotic.*** An alternative definition is to say that *the system is chaotic if it has some positive Lyapunov exponent* [30].

In deterministic chaotic systems the observed randomness does not disappear with the more data gathering in time, so, the prediction for the next system state will have the same expected error, no matter the number of previously known observations. What is more: there is an extremely high sensibility to starting conditions: if two realizations start with very similar starting points $\mathbf{x}$ and $\mathrm{x} + \varepsilon$ their trajectories will diverge between each other and their separation will exponentially increase with time.

In the case of stochastic systems, we do not find in the related literature a definition for "chaotic", specially referred with the Lyapunov coefficients, even though in [50] it is mentioned a system can be simultaneously chaotic and stochastic. On the other hand, we could not find a definition of "general" or "average" Lyapunov exponent either, that has into account all the realizations of the random process that would correspond with the idea of trajectory of a deterministic system.

Example: the Lorenz's attractor The Lorenz's system[31] is a three-dimensional system described by the following differential equations:

$$\begin{cases} \frac{\partial x}{\partial t} = \sigma[x(t) - y(t)] \\ \frac{\partial y}{\partial t} = x(t)[r - z(t)] - y(t) \\ \frac{\partial z}{\partial t} = x(t)y(t) - bz(t) \end{cases}$$

where b, r, and σ are parameters. Starting with an arbitrary point $[x_0, y_0, z_0]$ and using certain values for the parameters r and b we can obtain three time series (one for each variable). The described trajectory for the typical case $r = 28$, $\sigma = 10$ and $b = 8/3$ can be seen from different perspective angles in the following illustration:

[31] In the literature this author sometimes appears as Lorentz.

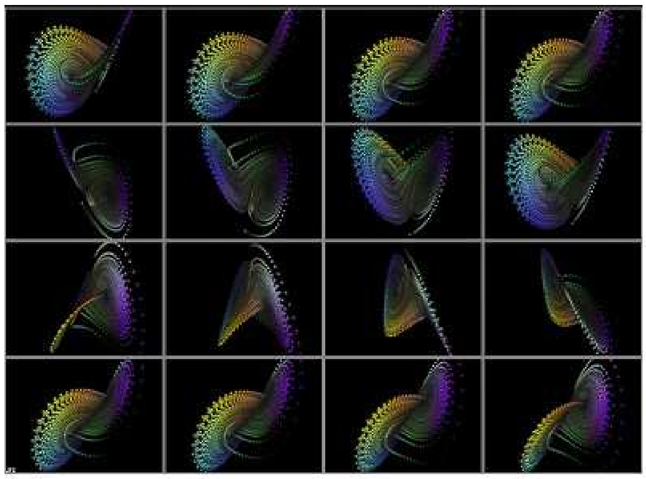


Figure 2.27

A dynamical system can be both chaotic and stochastic. For instance, if we add a random noise to the x coordinate to the Lorenz's attractor, we obtain a stochastic and chaotic dynamical system. In this case, the aspect of the Lorenz's attractor is translated from the one in Figure 2.27 to a set of fuzzy points, in some way related with the original one: for instance the dimension of the new attractor equals the dimension of the original attractor plus the dimension of the noise space (in this case, 1) [52].

2.9.3 Invariant characteristics

The invariant characteristics of an attractor of a dynamical system (or simply ***invariants***) are those properties that are preserved after a non-linear "smooth" and invertible transformation[32] to the states of the system. Formally, a measure μ is invariant under a transformation $F: S \rightarrow S$ if it holds that

$$\mu(F^{-1}A) = \mu(A) \qquad \forall A \in \mathcal{B}$$

where $\mathcal{B}$ is a $\sigma-$algebra in the ground set S.

Example of invariants are the optimum delay and the embedding dimension [33]. Other characteristic properties of a dynamic system are the generalized dimensions, the Kolmogorov's entropy, the Lyapunov's exponents, the Kaplan-Yorke's dimension and the predictability horizon [34]. These invariant characteristics lead to a full characterization of a system, and hence permit to determine how similar behaviors two systems have. This comparison criterion is very useful for a dynamic system reconstruction since we can discern how well a model reproduces the dynamics of the system.

[32]This is, a homeomorphism: a continuous one-to-one mapping, with continuous inverse.

[33]When the embedding dimension is mentioned, it is the minimum dimension of the reconstruction space such that the reconstructed trajectory does not cut itself.

[34]All these invariants are estimated by numerical methods that use a finite data set, in contrast with the definitions of dimensions for infinite data sets. Extrapolations are applied, and they can fail for several reasons, giving low reliable estimations [25]. Here we will not discuss the quality (robustness) for each estimation, and we will focus to use the ones provided by the most known software packages.

In the stochastic case, we get a probability density function (or probability distribution in the discrete case) for each of the invariants[35] and the problem is then to compare the densities/distributions for each one of the invariants generated by the model and the original system. Up to date, few works have been published concerning this issue [79].

2.9.3.1 Generalized dimensions and entropy

The attractors of a dissipative deterministic dynamical system generally have a fractal structure (in the sense they are similar to parts of themselves). The ***generalized dimensions*** are a class of quantities that characterize this "fractability" and they are invariants. Some of those dimensions are the information dimension and the correlation dimension, defined in what follows.

Generalized dimensions The study of the generalized dimensions of an attractor provides us information on its structure. For systems with high degrees of freedom (even deterministic) a trajectory in the state-space presents a rather irregular shape. The study of the generalized dimensions allows us to discover certain features of that shape. Let us divide the state-space (assuming it is bounded) into $N(\varepsilon)$ hypercubes of side ε and we will study the probability distributions p_i through the attractor when $\varepsilon \to 0$. The distribution p_i is defined by the probability to find a point of the trajectory inside hypercube i:

$$p_i = \lim_{N \to \infty} \frac{N_i}{N}$$

being N the total number of points in the trajectory and N_i the points inside hypercube i with size ε.

The ***generalized dimensions*** (or Renyi's dimensions) of an attractor are defined by

$$D_q = \frac{1}{q-1} \lim_{\varepsilon \to 0} \frac{Log \sum_{i=1}^{N(\varepsilon)} p_i^q}{Log\varepsilon} \quad \textbf{(Eq. 2.31)}$$

When $q = 0$, 1 and 2, the respective dimensions are called Hausdorff-Besicovitch's or ***fractal dimension*** D_0, ***information dimension*** D_1 and ***correlation dimension*** D_2 [63]. The dimension D_0 is used in Chapter 3 to estimate the dimension the signal subspace should have. On the other hand, D_2 can be approximated by

$$D_2 = \lim_{\varepsilon \to 0} \frac{LogC(\varepsilon)}{Log\varepsilon}$$

being

$$C(\varepsilon) = \frac{1}{N(N-1)} \sum_{j=1}^{N} \sum_{\substack{i=1 \\ i \neq j}}^{N} u(\varepsilon - \|x_i - x_j\|)$$

with $u(\bullet)$ the Heaviside function and x_i and x_j two points of the trajectory. The correlation dimension is one of the invariants provided by many tools of time series standard analysis (for example, DATAPLORE and TISEAN).

Alternatively, some authors [30] define the generalized dimensions as follows. Consider an attractor for a deterministic dynamical system, the probability measure $C(q,r)$ that returns

[35]In the case of Lyapunov exponents we could consider the distribution of the biggest exponent.

the probability that two points in the state-space $\mathbf{x}(n)$ and $\mathbf{x}(k)$ [36] (placed in the attraction domain) are separated a distance r is, by definition, the ***correlation function*** ([30] Chap. 14):

$$C(q,r)=\frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{N-1}\sum_{\substack{k=1\\k\neq n}}^{N}u\left(r-\|\mathbf{x}(n)-\mathbf{x}(k)\|\right)\right)^{q-1}$$

being N the number of data points (observed values of the series), q a real non-negative parameter and $u(\bullet)$ the Heaviside function.

This correlation function is an invariant attractor. Additionally, for practical purposes, it is usually considered its behavior for small values of r . In that case, when $r\to 0$, if the limit exists it is [30]

$$C(q,r)\approx r^{(q-1)D_q}\ \textbf{(Eq. 2.32)}$$

The value D_q is defined as the ***fractal dimension*** [37] of the attractor. Taking natural logarithms on both sides of Eq. 2.32 and regarding that the approximation holds when r is sufficiently small, we get that

$$D_q=\lim_{r\to 0}\frac{Log\left[C(q,r)\right]}{(q-1)Log(r)}$$

When $q=0$ the Hausdorff-Besicovitch is obtained, when $q\to 1$ the information dimension is, and the attractor correlation dimension D_2 is retrieved if $q=2$.

Interpretation for the generalized dimensions The difference among the different dimensions is mainly what is considered in each one: once q is increased, "dense" regions of the attractor are taken into account (density regarding number of points) [63]. It can be proved from the definition of D_q that $D_0=\lim_{\varepsilon\to 0}\frac{LogN(\varepsilon)}{Log(1/\varepsilon)}$ being $N(\varepsilon)$ the minimum number of hypercubes of side ε needed to cover the whole set of points of the attractor. The dimension D_0 would give us an idea of how "irregular" the attractor is. For example, for a square we have D_0=2 while for the Lorenz attractor it is within 2 and 3. This dimension also provides the degrees of freedom for the system. In order to have a predictable system (in the deterministic case) that value should be small. For systems with a high value of D_0 it may be more appropriate a stochastic model rather a deterministic one. On the other hand, D_1 is a lower bound for D_0, and these dimensions tend to have similar values [63].

Generalized entropies The ***Kolmogorov-Sinai's entropy*** can be defined as the average rate of information loss about the initial conditions through time. The temporal predictability of a deterministic system is inversely proportional to its Kolmogorov's entropy, so, no reliable predictions are feasible indefinitely because of the information loss associated with this entropy. Formally it is defined in the following manner. Consider the state-space divided in hypercubes (boxes) with size r^d and let $P_{i_0\ldots i_{d-1}}$be the joint probability that the state $\mathbf{x}(t=0)$ is in box i_0, $\mathbf{x}(t=1)$ in box i_1,...and $\mathbf{x}(t=(d-1)\Delta t)$ in box i_{d-1}, being Δt the sampling time. The Kolmogorov-Sinai's entropy[38] is [63]

[36]These points are equivalent to $P(t)$from Subsection 2.8.1.

[37]Haykin calls fractal dimension to the family of generalized dimensions, even though the term fractal dimension of an object is reserved to its Hausdorff-Besicovitch dimension, D_0

[38]Also known as Kolmogorov's entropy.

$$K = - \lim_{\substack{\Delta t \to 0 \\ r \to 0 \\ d \to \infty}} \frac{1}{d\Delta t} \sum_{i_0,\ldots i_{d-1}} P_{i_0 \ldots i_{d-1}} Log(P_{i_0 \ldots i_{d-1}})$$

where the sum is over all feasible boxes $i_0, i_1, \ldots i_{d-1}$. This entropy can be used to classify dynamical systems, since it is infinite for stochastic systems, positive and finite for chaotic systems and null otherwise [58]. Furthermore, it can be computed as the sum of the Lyapunov positive exponents[40]. In a general way, the ***generalized entropies*** are defined by [63]

$$K_q = - \lim_{\substack{\Delta t \to 0 \\ r \to 0 \\ d \to \infty}} \frac{1}{d\Delta t(q-1)} Log \sum_{i_0,\ldots i_{d-1}} (P_{i_0 \ldots i_{d-1}})^q$$

2.9.3.2 Exponents and Lyapunov's spectrum

Lyapunov´s exponents can be defined as the average (rate) at which diverge (or converge) two initially neighboring trajectories in the attraction domain of an attractor, and characterize the sensibility of a deterministic system to the initial conditions. To visualize this notion imagine the evolution of a spheroid with starting radius $\varepsilon(0)$ in an n-dimensional space. As the points move along the attractor trajectory, the spheroid will evolve to an ellipsoid, since each dimension (axis) has associated a different variation rate. Assume the length of the main axis of that ellipsoid over direction i is $\varepsilon_i(n)$. In that case, the Lyapunov exponent i is defined as:

$$\lambda_i = \lim_{n \to \infty} sup \left[\frac{1}{n} \log_2 \frac{\varepsilon_i(n)}{\varepsilon_i(0)} \right]$$

where n represents time. It is measured in bits/s (note we use logarithm in base 2). If we used natural logarithm instead, it would be measured in nats/s. [53].

The sign of Lyapunov exponents has a special meaning. A positive exponent represents divergence of the trajectories of a deterministic dynamical system in the considered dimension ($\lambda_i > 0$ implies there is divergence in direction i), whereas a negative exponents represents convergence. For a system to be considered chaotic it should have at least one positive Lyapunov exponent. If the system has more than one positive exponent, the biggest one typically presents more influence. A null exponent means there are no changes (there are no variations in the dimension of the spheroid) in that direction.

Real physical systems (those that exist in Nature), are dissipative, and their sum of Lyapunov exponents is non-positive [30].

As we previously defined, Lyapunov exponents are global quantities, since they are a limit, and hence they do not provide information on the local divergence rate for initially close trajectories, that can have different signs, depending on the attractor's region. In terms of prediction, there are areas where predictions can be feasible in the short-term (some steps in the future) and others where it is impossible because of the exponential increase in the errors. This phenomenon of change in the local divergence rates has been called ***effective (or local) Lyapunov exponents phenomenon*** [77].

The ***Lyapunov spectrum*** is formed by $\lambda_i \quad i = 1, \ldots d_E$ where each λ_i is a ***Lyapunov global exponent.***

In our case we find the global Lyapunov spectrum, using routines of TISEAN though cspX tool finds it locally. DATAPLORE also finds it globally.

Stochastic case In the case of stochastic dynamical systems a probability density function/probability distribution would be obtained for each invariant (see 2.9.3 on page 95), for example, the global Lyapunov exponents. Nevertheless, the both theoretical and practical interpretation of the exponents (global and local ones as well) is difficult in this case[77], even more if the same algorithms to find Lyapunov exponents in deterministic systems are directly applied (such as the one of Sano and Sawada, used by TISEAN) to a time series coming from a stochastic dynamical system. In this case, fake positive exponents can be obtained [Tanaka, Aihara and others, cited by [77]]. A partial solution to avoid fake positive exponents is to find the maximum Lyapunov exponent based on confidence intervals, as proposed by Gençay (cited by [77]). Moreover, since for each starting point we have a probability density function over all the possible states for the system at time n, the system sensibility with respect to initial points can be thought as the distance (in the sense of Kullback-Leibler) between probability distributions at time n the one corresponding to initial states x and $x+\varepsilon$. In contrast with Lyapunov exponents, this sensibility is local, and depends on the initial states [77]. In this case, taking the distance:

$$K(p\|\, q) = D(p\|q) + D(q\|p)$$

being D the Kullback-Leibler distance between p and q we obtain a function that satisfy the properties stated in 2.2.3. K was so defined to be symmetric.

The distance $K(p\|\, q)$ can be approximated by [77]:

$$K(x_n, x_n + \varepsilon) \approx \varepsilon^T I(\overline{x_n})\varepsilon$$

where

- $I(\overline{x_n})$ is the Fisher information matrix:

$$I(\overline{x}_n) = \left\{ I_{ij} = \left\langle \frac{\partial Log\rho(x_{n+1}|\overline{x}_n)}{\partial \overline{x}_i} \frac{\partial Log\rho(x_{n+1}|\overline{x}_n)}{\partial \overline{x}_j} \right\rangle_{x_{n+1}} \right\}$$

- m is the order of the system
- $n - m + 1 \leq i, j \leq n$
- $\rho(x_{n+1}\,|\overline{x}_n)$ is the conditional distribution of the system to be in state $n+1$ provided it was in $x_n, x_{n-1}, \ldots x_{n-m+1}$.

This approximation has the inconvenient that the conditional distributions of the states should be known; a practical alternative appears in [77] where a sensibility analysis for the starting conditions with a mixture distribution network is presented (MDN, see [10] Chap. 2) to represent the conditional distributions $\rho(x_{n+1}\,|\overline{x}_n)$.

2.9.3.3 Kaplan-Yorke's dimension

Given the Lyapunov spectrum, Kaplan and Yorke suggested the dimension (called of Kaplan-Yorke or of Lyapunov):

$$D_{KY} = K + \frac{\sum_{i=1}^{K} \lambda_i}{|\lambda_{K+1}|}$$

where K is an integer that verifies

$$\left\{ \begin{array}{l} \sum_{i=1}^{K} \lambda_i > 0 \\ \sum_{i=1}^{K+1} \lambda_i < 0 \end{array} \right.$$

For a chaotic deterministic system generally is $D_{KY} = D_2$ [30]. However, depending on the method used to estimate D_{KY}, we can have D_{KY} and D_2 very close or not, depending on whether the noise in the data was is taken into account or not (some estimations for D_{KY} considers it but others do not) [59].

2.9.3.4 Predictability horizon

As we said, due to the sensibility to the initial conditions in a deterministic chaotic system, the orbits starting from two near points separates exponentially with time (the greater the exponent, the lesser quantity of values that can be predicted). This is the reason why deterministic chaotic systems are predictable in the short-term but in not the long-term. Sort-term predictability[39] is defined as the ***predictability horizon*** and is mainly determined by the biggest Lyapunov exponent. Moreover, we estimate the length of the predictability by [30]:

$$HOP \approx \frac{1}{\lambda_{MAX}}$$

for chaotic processes. For deterministic non-chaotic systems, the horizon is not defined, or it would be theoretically infinite, even though the prediction errors accumulate and make the trajectories (the real versus the predicted one) diverge. For stochastic dynamical systems, the horizon cannot be thought as the number of steps we can predict without having high errors, since the randomness of the states makes that can be predicted only the expected value of the random variable with law $P(x_{n+1}\,|x_n)$ even though the observed value is far away from it. Anyway, since the predictability horizon is associated with long-term predictions, usually found via an iteration of short-term predictions, some improvements can be introduced to this prediction types, such as the ones proposed by Judd and Small with the $\mathbf{\Psi\Phi}$ method [79]. In this method the key is to correct systematic prediction errors that could commit the predictor $\mathbf{\Phi}$. In other words, a number of values are predicted by means of predictor $\mathbf{\Phi}$ and then the correction parameters $\mathbf{\Psi}$ are determined in order to minimize the mean square error between the desired and corrected values, and finally use that corrector to correct the predicted values in the future. For example, if f_n represent the desired values to be predicted, z_n are the predicted values for the instant $n,$ and if we take $\mathbf{\Psi}(z_n) = \mathbf{A}z_n$ for a certain matrix $\mathbf{A}$, then the estimation of $\mathbf{A}$ such that $\|f_n - \mathbf{A}z_n\|^2$is minimized is

$$\widehat{\mathbf{A}} = (\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T\mathbf{F}$$

being $\mathbf{Z}$ a matrix such that the i-th column is the prediction for time n and $\mathbf{F}$ the matrix such that column n is the desired value for time n[79].

2.9.4 Embeddings and the the Takens-Mañé's theorem

2.9.4.1 Formal statement

We will give here both a formal statement and an intuitive interpretation. The formal statement is:

[39]Better, the number of steps in the future that can be predicted.

Theorem:
Consider a compact m-dimensional manifold M. For pairs (φ, y) such that $\varphi \in Diff^2(\mathrm{M})$ and $y \in C^2(\mathrm{M}, \mathbb{R})$ is a generic property that the mapping

$$\Phi_{(\varphi,y)} : \mathrm{M} \to \mathbb{R}^{2m+1}$$

defined by

$$\Phi_{(\varphi,y)} = (y(x), y(\varphi(x)), ...y(\varphi^{2m}(x)))$$

is an embedding, where

- "generic" here means open and dense
- $Diff^2(\mathrm{M})$ is the set of functions $C^r : \mathrm{M} \to \mathrm{M}$ with inverses

An equivalent statement was formulated by Takens [35]:

Theorem:
Consider a compact m-dimensional manifold M, and $\varphi : \mathrm{M} \to \mathrm{M}$ a diffeomorphism with two properties: the generic points of φ with periods not higher than $2m$ are finite, and if x is an arbitrary periodic point with period $k \leq 2m$ then the eigenvalues of the derivative of φ^k are all different in M.
Then, for a generic y such that $y \in C^2(\mathrm{M}, \mathbb{R})$, the mapping

$$\Phi_{(\varphi,y)} : \mathrm{M} \to \mathbb{R}^{2m+1}$$

(defined as in the previous theorem) is an embedding.

The attractor of a deterministic system is obtained drawing the evolution of the state variables in the state-space. This is only possible when we have access to all the variables of the system, which rarely occurs in practice. Instead, it is common to have some function M defined over the possible system states, whose values form a single time series (of measures) $s(n) = M(\mathbf{x}(n))$.

Takens and Mañé[40] proved, independently, that for a time series the original attractor can be reconstructed, using the ***delayed coordinate method***. Afterward, it was proved that this holds also for the stochastic case ([80]). The dynamic reconstruction of the system from a time series of observations, both in deterministic and stochastic scenarios, are "ill-posed" inverse problems. The direct problem is to describe the trajectories given the dynamics, and in the inverse version we are given the trajectories and the dynamics must be described. The inverse problem is "ill-posed" since (depending on the data quality) the found solution can be either stable or unstable, be only one or even may not exist. Some previous knowledge about the input/output mapping is sometimes included in the predictive model in order to deal with those "ill-posed" problems [30]. A way to do it is to use embedding parameters: dimension and optimal delay computed from the observed data $s(n)$ in the deterministic system. We will see that this idea can also be used in the stochastic case.

[40]Mañé, Ricardo: Uruguayan mathematician that simultaneously with Takens, proved the theorem. The literature usually refers to this theorem as of Takens or "of the embedding".

2.9.4.2 Delays method

From the series x_n we get the vectors

$$\mathbf{x}(n) = [x_n,\ x_{n-\tau} \dots x_{n-(d_E-1)\tau}]$$

where

- d_E is the ***"embedding dimension"*** [41]
- τ is the ***"embedding delay"***, chosen as a multiple of the sampling time (the inter-sampling time is considered as 1).

2.9.4.3 The notion of embedding

A set X is ***embedded*** in another set Y when the properties of Y (connectivity, algebraic or others) restricted to X are the same. For example, the rational set is embedded in reals and integers embedded in rationals. We say there is an ***embedding*** from X to Y. In the geometric case, a sphere is embedded in $\mathbb{R}^3$. Formally:

Definition: Given two manifolds X, Y and a mapping $f : X \to Y$ we say that f is an embedding if f is a diffeomorfism of X onto its image $f(X)$ [78]

By an abuse of language we write "embedding dimension" to refer the dimension of the co-domain (Y) of the embedding.

2.9.4.4 The Takens-Mañé's theorem

The theorem assumes the existence of both d_E and τ, and in that case[42], the vector function $\mathbf{x}(t) = [x_t,\ x_{t-\tau} \dots x_{t-(d_E-1)\tau}]$ is an embedding from the attractor over the set of points $\mathbf{x}(t)$. Exploiting the continuity properties of embeddings [43], we can predict $\mathbf{x}_{t+\tau}$ from $\mathbf{x}(t)$, provided the parameters d_E and τ are adequate, even though the theorem does not give further details of them, stating that it is enough that $d_E \geq 2D + 1$ there D is the fractal dimension of the attractor. This means that, from almost any observations set (time series) of the system, we can answer a wide range of questions concerning the dynamics of the original system by only studying the dynamics in the space defined by the delayed values of the time series.

In general, again with an abuse of terminology, we talk about the "dimension of the attractor", to refer the dimension of the signal subspace, d_E.

2.9.4.5 Determination of the parameters of the embedding

According to the Takens-Mañé's theorem, the parameters d_E and τ must be determined such that $d_E \geq 2D + 1$, but afterward it was proved that it suffices that $d_E \geq D + 1$ holds, and τ can be arbitrarily chosen provided the data have no noise and present infinite size. Given that these assumptions are not realistic, we must determine both d_E and τ using methods based on the data. The first to be determined is the optimal delay, since it is necessary to find then the embedding dimension.

[41] In the reconstruction space we take $\tau = 1$ so we have $d_E = N$ and have an intuitive notion for the dimension of the signal subspace.

[42] The theorem needs additional (little restrictive) conditions, and the related literature usually ignores them. For example, a requirement is that the system must not contain periodic trajectories with period τ or 2τ, and it must contain a finite number of rest points.

[43] Since they are continuous mapping by definition.

Deterministic case

Optimal delay The optimal delay τ for the embedding is defined as the needed value to obtain $x(n), x(n-\tau), x(n-2\tau)$... independent enough as to have a base for the signal subspace ([30] Chap. 14). If the delay is too small, then the elements of the vector $\mathbf{x}(n)$ are too similar (they are too correlated) and are vulnerable to noise effects (it is said there is a redundancy problem). On the other hand, if it is chosen big enough, the elements are extremely dissimilar and the vectors $\mathbf{s}$ are sparsely distributed in the state-space (there is a problem of irrelevance). Both situations make difficult to reconstruct the attractor. In the method of mAMI ("minimum average mutual information") the average mutual information $I(n)$ between the series x_n and its delayed version x_{n+t} is found for $t = 1, 2 \ldots$by

$$I(t) = \sum_{n=1,2\ldots} P(x_n, x_{n+t}) \log_2 \left[\frac{P(x_n, x_{n+t})}{P(x_n)P(x_{n+t})}\right]$$

where $P(\bullet)$ is the probability law and $P(\bullet, \bullet)$ denotes the joint probability. The value for t where the first minimum for $I(\bullet)$ occurs is suggested as a good estimation for the optimal delay τ of the embedding. Additionally, the AMI shows non-linear correlations that could not be clearly seen if only linear correlations were used instead.

An alternative option is to choose τ as the ***time correlation*** for the time series: it is the number τ such the autocorrelation $C(\bullet)$ falls at its half of its first value [63]:

$$\begin{cases} C(\tau) = \frac{1}{2}C(0) \\ C(k) = \sum_{m=-\infty}^{+\infty} x_m x_{m+k} \end{cases}$$

Global and local embedding dimension The dimension of the signal subspace (embedding) is not necessarily the same than the one of the attractor, D, and its suffices to find a case where $d_E \geq D + 1$. Such dimension d_E is called the ***global dimension*** of the embedding. The procedure to find it is based on geometrical considerations, and was established by Kennel and others (cited by [31]). The method finds the ***nearest neighbor*** of each point in a given dimension, and it is checked whether those points are still close neighbors in a higher dimension or not. This method has as its main advantage that it does not need very large data sets. Essentially, it determines when the points at the dimension d are neighbors of other by virtue of the projection for the attractor in a sufficient small dimension:

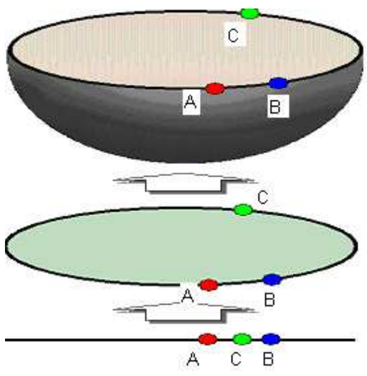


Figure 2.28

Observe how the points A and B appear as the nearest neighbors for C when they are projected over a line in a uni-dimensional space, but they appear much far away when they are projected over a disk of two dimensions. By contrast, the points A and B are near in the two-dimensional space and they still be near in the three-dimensional space. By a close examination of the situation in successive dimensions one, two, three,... until there are no incorrect (or false) neighbors, we could establish, by geometrical considerations, a value for the embedding dimension. If we plot the percentage of false nearest neighbors as a function of the embedding dimension, the curve should vanish when the adequate dimension for the embedding is achieved.

The algorithm can be sketched as follows (see [31] for a justification):

1. *For each vector $x(i) = [x_i, x_{i-1}, x_{i-2} \dots x_{i-n}]$ in the time series, find the nearest neighbor $y(j) = [y_j, y_{j-1}, y_{j-2} \dots y_{j-n}]$ in an n-dimensional space.*
2. *Find the distance $\|x(i) - y(j)\|$.*
3. *Take the following values for the time series, x_{i+1} and y_{j+1} and find*
$$R_i = \frac{|x_{i+1} - y_{j+1}|}{\|x(i) - y(j)\|}.$$
4. *If $R_i$exceeds a threshold heuristically obtained beforehand R_τ, this point is marked as it has a false nearest neighbor.*
5. *If the percentage of points for which $R_i > R_T$ is below a certain value, n equals the embedding dimension. Else, increase n in 1 and the process (all the previous steps) is repeated.*

While this method is quite intuitive and easy to implement, it cannot be directly used, since (among other aspects) it needs a heuristic estimation for the thresholds (R_τ determines whether a point is a false nearest neighbor or not, and the percentage R_T of false nearest neighbors that are accepted to determine the correct dimension is found). The chosen tool (VRA) is a slightly modified implementation, where there are no free parameters. Since Y$_j$ is the nearest neighbor of X$_i$, we can think it is an approximation to it, or x_{n+1} is predicted by y_{n+1}, so R is the prediction error. The idea is that when the attractor is completely unfolded in n dimensions, the distance R between the $(n+1)$-th components from X and Y will be small. In order to detect whether the nearest neighbor recently found is false or not, we compare R with the error that would have produced using the trivial predictor (the one that uses x_n as a predictor of x_{n+1}). Formally, if $|x_{n+1} - y_{n+1}| = |x_{n+1} - x_n|$ the following neighbor is entitled as false.

Since the algorithm is intended to find the global dimension for the embedding, it is called ***global FNN*** method.

As an additional tool some products (like cspX) find the ***local dimension*** of the embedding, d_L, such that $d_L \leq d_E$, which represents the active degrees of freedom that govern the local evolution of the system. For instance, suppose we have a periodic orbit or limit cycle as the system's steady state. Then, the movement of the points over the state-space is in a uni dimensional object, but it needs $d_E = 2$ or 3 to develop an attractor an see this, even though the dynamics over the limit cycle is uni-dimensional. So, cspX uses the Local FNN method, that adds some information over the geometry of the attractor when compared with the Global FNN method.

Stochastic case In the stochastic case, to find the dimension with the FNN method is no longer useful since it is based on the continuity of the deterministic case

(if $\mathbf{x}(j)$ is the nearest neighbor to $\mathbf{y}(j)$ then x_{j+1} must be close to y_{j+1} for a certain dimension d_E).

Therefore, some considerations on probability theory are mandatory. Fueda and Yanagawa ([21]), for the case where the time series complies that

$$x_t = F(x_{t-\tau}, x_{t-2\tau}, \dots x_{t-d\tau}) + \varepsilon_t \quad t = 1, 2...$$

being

- F a non-linear field (vectorial function) with a vector as its argument
- ε_t is dynamic noise
- τ is the optimal delay
- d the dimension of the signal subspace

and the following relations are respected:

$$\begin{cases} F(x_{t-\tau}, x_{t-2\tau}, \dots x_{t-d\tau}) = \langle x_t \,|x_{t-\tau}, x_{t-2\tau}, \dots x_{t-d\tau} \rangle_{x_t} \\ \langle \varepsilon_t \,|x_{t-\tau}, x_{t-2\tau}, \dots x_{t-d\tau} \rangle_{\varepsilon_t} = 0 \\ \langle x_n^2 \rangle < \infty \end{cases}$$

plus certain conditions for the dynamic noise (see [21]), define the embedding dimension d_0 with an optimal delay τ_0 if an only if the following conditions simultaneously hold:

- There exist two natural numbers d_0 and τ_0
- $P\left[F(x_{t-\tau}, x_{t-2\tau}, \dots x_{t-d\tau}) \neq F(x_{t-\tau_0}, x_{t-2\tau_0}, \dots x_{t-d_0\tau_0})\right] = 1$ for all $d < d_0$ and $\tau > 0$
- $P\left[F(x_{t-\tau}, x_{t-2\tau}, \dots x_{t-d\tau}) = F(x_{t-\tau_0}, x_{t-2\tau_0}, \dots x_{t-d_0\tau_0})\right] = 1$ for all pairs $(d, \tau) \in B(d_0, \tau_0)$ being

$$B(d_0, \tau_0) = \{(d, \tau) \,|\{\tau_0, 2\ \tau_0, \dots d_0\tau_0\} \subset \{\tau, 2\tau, \dots d\tau\}\}$$

They also proposed a statistical estimation for the optimal delay and dimension for that case [21].

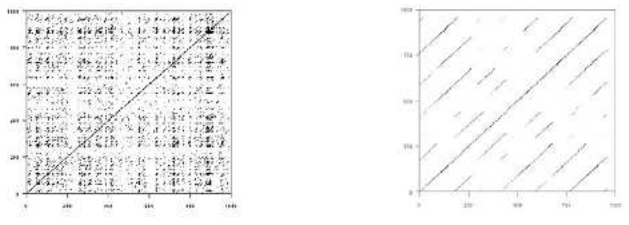
2.9.5 Recurrence analysis and recurrence plots

R***ecurrence analysis*** is a graphical method designed to locate hidden pattern that are repeated in time (in the sense of repetitive behaviors that are not easily detected), non-stationarity and changes in the structure of a time series. Let x_n be the series under study and $x_i^m = [x_i, x_{i+d}, \dots x_{i+(m-1)d}]$ the vectors where m is the embedding dimension and d the optimal delay. From the vectors x_i^m a symmetric matrix is obtained with the euclidean distance among all the possible pairs, and these distances are visualized by means of a graphic called ***recurrence plot***, where each distance of that matrix is associated with a color (for example, the color is "colder" for longer distances; this is the way that the VRA proceeds: longer distances are bluish, whereas the shorter ones are represented with reddish). A recurrence plot is a solid square plot that has pixels whose color corresponds to the magnitude for the values of a two-dimensional matrix and whose coordinates correspond to the place for the values of the data in the matrix. It is rather usual to establish a critical radius ε and to mark a point as dark only if the resulting distance is not higher than ε (as in the case of TISEAN). A black and white plot is hence obtained. The vectors compared with themselves give a null distance, explaining the presence of the diagonal from the lower left to the upper right cornet (***identity line***) and it appears in all recurrence plots.

If the series analyzed is periodic then the recurrence plot shows line segments parallel to the identity line that correspond to the sequences of pairs (i,j), $(i+1,j+1)...(i+k,j+k)$ such that $x_i^m, x_{i+1}^m \dots x_{i+k}^m$ are close to $x_j^m, x_{j+1}^m \dots x_{j+k}^m$ for some j given i. These segments a certain attractor region is re-visited. On the other hand, if the series is white noise, the recurrence plot does not show structure but is formed by uniformly distributed points. The non-stationarity is expressed by a tendency of all points to be close to the identity line [7].

This visual analysis helps us to state in Chapter 3 the stationarity of the process. On the other hand, several indicators complement well the recurrence plot, so the conclusions are not merely extracted from visual appreciations (these indicators are for example of determinism, for recurrence, etc.). These indicators numerically represent the graphical features from the RP, and that technique is called Recurrence Quantification Analysis (RQA).

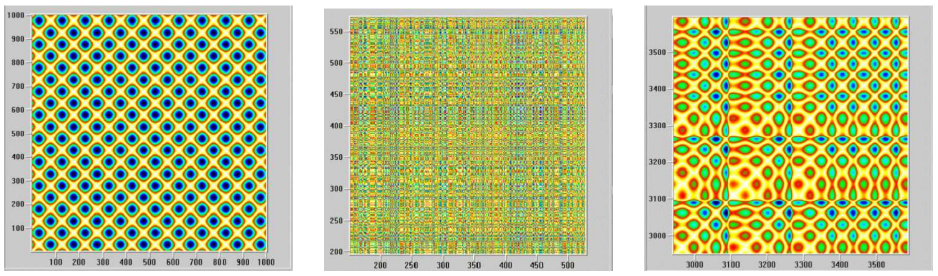
TISEAN produces the RP of Figure 2.29:



white noise ***chaotic series (Rossler's system)***

Figure 2.29

On the other hand, by means of VRA tool the recurrence plots obtained for different series are presented in Figure 2.30:



sin(x) ***white noise*** ***chaotic series***

Figure 2.30

2.9.6 Prediction and dynamic reconstruction

The ***prediction*** of a time series, as a way to predict the future behavior of the subjacent system, has been one of the key areas in science and engineering. Given some observations of the system behavior in the past, it is desired to make certain predictions over its future behavior and to determine how exact they are beforehand. In our case, we wish to predict weekly sales of propane gas in bottles, given the historical information of sales.

The other approach is ***dynamic reconstruction***. Dynamic reconstruction and prediction of time series are two areas with several common aspects, but differ in the way the final results are evaluated. The goal of time series prediction from a classical viewpoint is to obtain the closest possible values to the real data. On the other hand, dynamic reconstruction tries to dynamically capture the time series by means of a model. In other words, prediction is related with short-term evolution, whereas dynamic reconstruction with a long-term behavior [53]. This does not mean that it is impossible to predict in the long-term. Indeed, dynamic reconstruction tries to have those predictions to be the best possible ones in the long-term. In order to check whether the model captures the dynamics of the system, invariant characteristics can be used (see 2.9.3 on page 95).

Prediction may be thought as it were associated to a local or global model . When the model is local, the attractor is partitioned in smaller regions, and several models are constructed for each region. Local models, as a consequence, vary from region to region, and given the discontinuities that exist in the attractor zone where the models overlap, undesirable behavior can be produced in the long-term, so the local models are poor for a dynamic reconstruction [59]. However, when a global model is considered, the dynamics of each region determined by the attractor is considered jointly. A global model can be obtained for example with a MLP using all the available data. On the other hand, local models might be obtained from local AR-MA models, be based on SOMs (Self Organizing Maps) or even polynomials that approximate a time series locally.

Dynamic reconstruction allows us to obtain a model of the dynamic system, making feasible to carry out test (experiments) over the system behavior under special circumstances. Observe that both in prediction or reconstruction using neural networks, we did not limit to an only network, but we can use even several networks simultaneously: for example, one network for prediction could be chosen, and another for error prediction (under the deterministic case). This approach is applied in [24].

The data impose some limitations to the model from the point of departure, since a model will be good in those attractor regions for which enough data is available. By this reason, the quality of a model, specially a global one, is limited by the best quality obtained for the less represented attractor regions over the training set.

2.10 MLPs as universal approximators

We will comment in this section two fundamental theorems about MLPs as universal approximators of every continuous function with domain in $\mathbb{R}^n$. As a corollary of Kolmogorov's theorem, it is possible to approximate an arbitrary continuous function by means of several perceptrons working in parallel. Cybenko's theorem states that feedforward networks with a single hidden layer and one linear output neuron are able to approximate every continuous function $f : \mathbb{R}^n \to \mathbb{R}$ with any desired level of accuracy. The bibliography for this section is basically [29].

2.10.1 Kolmogorov's theorem

This classical theorem can be stated in this way:

Kolmogorov's Theorem:

Consider an arbitrary continuous function f with real arguments $f(x_1, x_2 \ldots x_n)$ where $\mathrm{x} = [x_1 \ldots x_n] \in [0,1]^n \quad n \geq 2$. *Then, f can be represented by*

$$f(x_1 \ldots x_n) = \sum_{j=1}^{2n+1} g_j \left[\sum_{i=1}^{n} \varphi_{ij}(x_i)\right]$$

where the functions g_j have a single real-valued argument and φ_{ij} are monotonically increasing continuous non-differentiable functions independent of f [29].

The theorem states any continuous function with multiple real arguments can be expressed as a finite sum of single argument functions. The direct application of this theorem to justify that feedforward networks are universal approximators has been widely discussed, mainly for two reasons:

1. The functions φ_{ij} are non-differentiable. The use of this kind of functions in a network would produce problems of extreme sensibility with respect to input values [10]
2. the functions g depend on the function f itself to be represented, in opposition with the practical operation of a neural network: in general, fixed output functions are considered and then the number of hidden neurons, weights and activation thresholds are adjusted until a satisfactory output is produced. To illustrate the idea, a graphical scheme for Kolmogorov's theorem is shown in Figure 2.31, with a network that implements the proposed decomposition:

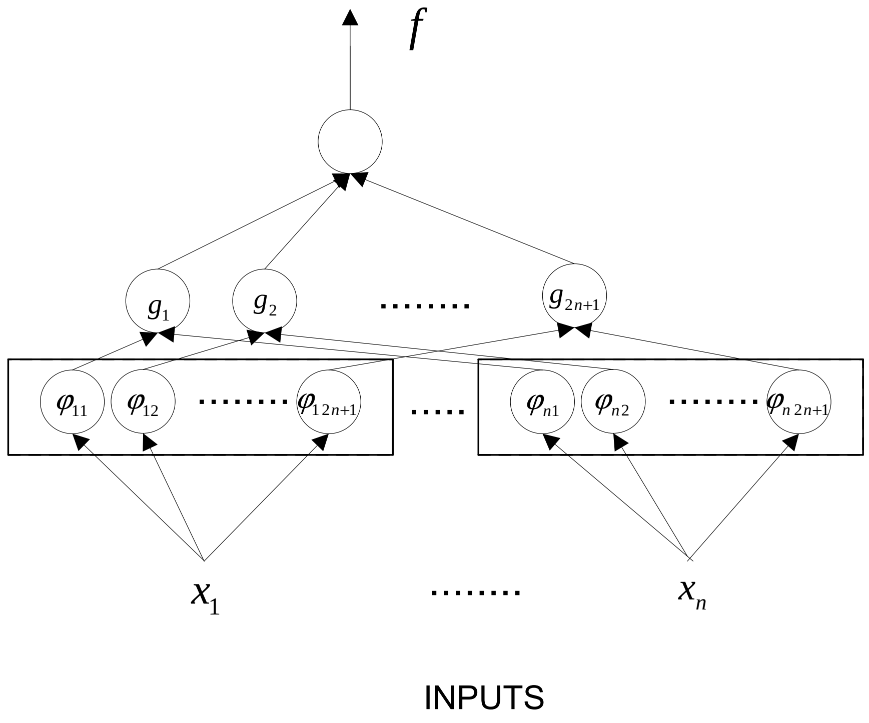


Figure 2.31

The number of hidden neurons is fixed in Kolmogorov's theorem and the output functions depend on the function that is being approximated. Anyway, although its poor practical application, the theorem points out the feasibility to use multilayer feedforward networks working in parallel (the boxes would be working in parallel) in order to approximate a desired function.

2.10.2 The Cybenko's theorem

The mathematical proof that MLPs that use sigmoid output functions are universal approximators was independently given by Cybenko, Hornik and Funahashi (cited by [29]), even though the credits are usually given to Cybenko because his neat and elegant proof.

The theorem can be stated as follows:

Cybenko's Theorem:

Consider a continuous sigmoid function $\varphi(\bullet)$. *Given an arbitrary continuous function* $f : [0,1]^n \to \mathbb{R}$ and $\varepsilon > 0$, *there exist vectors* $w_1, w_2 \dots w_N$, θ *and a parametrized function* $G(x, w, \alpha, \theta)$: $[0,1]^n \to \mathbb{R}$ *such that*

$$\begin{cases} |G(x,w,\alpha,\theta) - f(x)| < \varepsilon \quad \forall x \in [0,1]^n \\ G(x,w,\alpha,\theta) = \sum_{i=1}^{N} \alpha_j \varphi(w_j^T x + \theta_j) \end{cases}$$

being $w_j \in \mathbb{R}^n, \quad \alpha_j, \theta_j \in \mathbb{R}, \quad w = [w_1, \dots w_N] \quad \alpha = [\alpha_1, \dots \alpha_n] \quad \theta = [\theta_1, \dots \theta_n]$

The theorem states that a MLP network with one hidden layer is able to approximate every continuous multivalued function with an arbitrary level of accuracy. Additionally, Hornik et al. (cited by [29]) proved the network can also approximate the derivatives of *f*, even though f is piecewise differentiable only.

2.10.3 Generalization

Hornik (cited by [29]) proved that a sufficient condition for an universal approximator is that the output functions of the hidden layer are continuous, bounded and non-constant. Non-sigmoid functions can also be used, such a Bernstein's polynomials (bell-shaped).

MLP networks with a single hidden layer are also universal classifiers: it suffices to define a function f over the input pattern set $f(x) = j \Leftrightarrow x \in P_j$ with $f : A^n \to \{1, 2 \ldots k\} \quad A^n \subset \mathbb{R}^n$ compact and $P_1 \ldots P_k$ a partition of A^n.

2.11 Outliers determination

Given a set of points in an N-dimensional space, an outlier is a point that is "extremely far away" from the others. More formally,

Definition a): "An outlier is an observation that is at an abnormal distance of the other values in a random sample of a population" [55].

Other definition takes into account statistical properties of populations:

Definition b): "An outlier is a case (an instance) that does not follow the same model of the rest of the data and seems to come from another probability distribution" [14].

We will adopt definition a). This definition leaves the opportunity to the analyst to decide what is considered "abnormal". On the other hand, the outliers processing (that is, to see how they will be considered) is necessary: in accordance with [81] "the values that are abnormally far away from the mean value of the random variable (**outliers**) can have an effect when training the network that is out of proportion. This effect can be even worse if those values are produced by noises. For that reason, it is recommended to remove the outliers before training". The determination of outliers from only a single dimension of the data points (in our case, sales), without regarding the logical relations that the dimensions have, can lead to poor results, so in this work the Mahalanobis's distance is suggested.

2.11.1 The Mahalanobis's distance

2.11.1.1 Introduction

This distance is more adequate than the euclidean one when the spatial distribution of points are not spherically symmetric, and no special distribution (in a statistical sense) for the points is required. Intuitively, if the shaded set is the set of available data (see Fig. 2.32), and **A** and **B** two points in it, with barycenter μ, it is not appropriate to say that, even if **A** and **B** are at the same euclidean distance from μ, that they have "equivalent" positions, because **A** is in a region with high points density whereas **B** is not[44]:

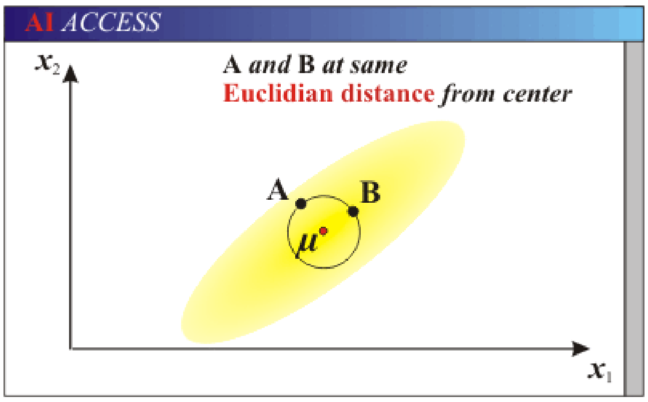


Figure 2.32

The euclidean distance that is

[44]Taken from [55]

$$d(\mathbf{A}, \mu) = \sum_{i=1}^{N} (a_i - \mu_i)^2$$

is then replaced by the Mahalanobis's distance, in such a way that **A** and **B** are considered "equivalents" in the following case:

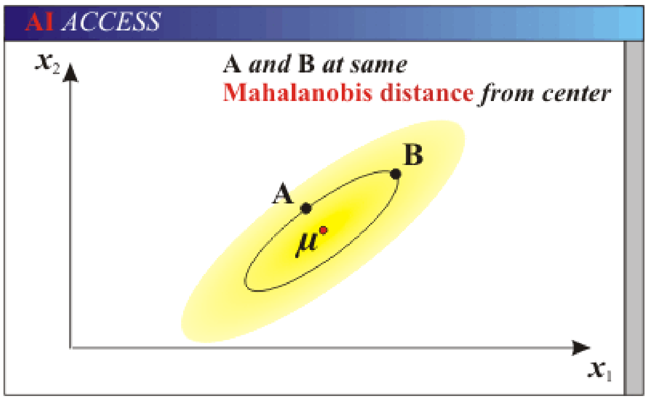


Figure 2.33

2.11.1.2 Formal definition

Formally, the Mahalanobis's distance of a vector $\mathbf{x} = [x_1, ...x_N]$ to a set of points with mean $\mu = [\mu_1, ...,\mu_N]$ and covariance matrix **S** is defined by

$$D_M(\mathbf{x}) = \sqrt{(\mathbf{x} - \mu)^T \mathbf{S}^{-1} (\mathbf{x} - \mu)}$$

If the covariance matrix equals the identity, the Mahalanobis's distance reduces to the euclidean one.

Intuitively, this distance is the "normalized" one between the point in question and and the rest of the points.

Mahalanobis used it for the first time to identify similarities in skulls from certain characteristics measurements [57].

Application

In this work we will apply the previous concept of outlier to the distance values D_M obtained for each point: we replace the values $\{V_1, V_2, ...\}$ by $\{D_M(V_1), D_M(V_2)...\}$ for the determination of outliers, using all the dimensions (temperature, relative humidity, etc.).

2.12 Used tools

Different software products were used. Therefore, a study of the state of the art of the software tools needed in order to execute all the tasks of this project was performed. Some common requirements to all of them were to be:

1. Adequate to work in stand alone Intel (PC) platforms, under Windows 7 operating system (optionally Linux), 32 bits.
2. Stable products: not beta versions or prototypes. It was pretended to have contact with mature tools within the market.
3. Updated software: that the latest version was as updated as possible, and the product should have support/updates
4. Commonly used products in the industry
5. Available for its use as a licenced copy, at least of an earlier version, and to have demos of its last updates; this requirement was excluding
6. For the neural network software further characteristics were required:
   (a) present features that drive us to study in depth the theory of artificial neural networks
   (b) present a graphical interface with a network design assistance or help (optional)
   (c) be designed by well known research teams in the field (the underlying algorithms hence have a solid theoretical background)
   (d) allow the largest variety of topologies and learning rules, mainly those appropriate for time series prediction.

Here we did not address considerations that usually are made when software is acquired:

1. licensing costs
2. number of previous installations: we only consider the product maturity, possibly related with the number of installations (a product in its fifth release is probable to have been installed more times than other with the same purpose but in its second version released).
3. company´s/organization's size (relevance) that released the software, giving an idea of the support in a future (if the company/organization will eventually exist or not)
4. portability to other hardware and software platforms
5. language of the user interface (English or Spanish)
6. issues related with security and product access
7. ability to cope with unexpected failures (blackouts, etc.)

We classified the used software in auxiliary tools (all the accessory software such as text processors and others) and the neural network software itself.

2.12.1 Auxiliary tools

Regarding the previous considerations, we chose:

1. Microsoft Excel 2010 for handling and processing spreadsheets.
2. LyX 2.0.50 and Jabref 2.8.10for the writing of this document and the bibliography management.
3. SPSS ver. 11.0 and Statistica ver. 6.0 for the management of statistical data and simulation analysis. After some tests, Statistica was chosen regarding its additional functionalities and graphic interface.
4. TISEAN, VRA, TSTOOL and DATAPLORE to handle the time series. The first three product are free-ware, and the last provides a demo version. TSTOOL is command line oriented and operates in Matlab. VRA and DATAPLORE provide very good graphical interfaces, while TISEAN has a command line interface and has no graphics. Some of these tools are described in [7].

2.12.2 Neural networks software

We distinguish between simulators and development frameworks.

2.12.2.1 Simulators

They are software applications that simulate the behavior of artificial or natural neural networks (that is, there are pre-set network models within the tool). They can be classified into research simulators, data analysis simulators or simulators oriented to teaching the theory of neural networks.

- Research simulators

These programs allow us to research the properties of neural networks through simulation. Among the most common simulators we have the Stuttgart Neural Network Simulator (SNNS), Emergent, JavaNNS and Neural Lab. Sometimes, simulation is the only possibility: in the case of biological neural networks, the only possible approach is this one. In that case physical, chemical or biological properties of neural tissues are simulated by means of tools like Neuron, GENESIS, NEST and Brian. Other simulators are XNBC and BNN Toolbox for MATLAB.

- Simulators for data analysis

A key difference with the previously mentioned simulators is that data analysis simulators primarily focus on data mining and prediction; they tend to have preprocessing functionalities. Most of these simulators use either back-propagation networks or self-organized maps as main topologies. They present the advantage that they are relatively easy to use, and have in contrast their limited capacities. Some of these simulators can be integrated with other computational environments such as Microsoft Excel.

- Simulators for teaching neural networks

The first simulators that did not require programming skills for its use was the Parallel Distributed Processing (PDP), that appeared in 1986-1987, fact that encouraged several researchers from different fields to use it. This simulator evolved and currently is known as Emergent. By 1997 the tLearn package was released, with the idea of to return to a small and simple simulator for novice users. This tool has not been updated since 1999. Last, Basic Prop, launched in 2011, is a simulator distributed as a neutral platform (.JAR) providing most of the functionalities offered by tLearn.

2.12.2.2 Development environments

The development environments for neural networks differentiate from the previously mentioned software in the sense that a) they help to develop own user networks and b) they support neural networks deployment out of the development environment (for instance, generating a .dll module that reads a file and performs a classification or prediction from the read data). We can find two types of such environments:

- Component-based environments

In this kind of environment the neural network is built by connecting adaptive filters which allows a great flexibility, since network and user components can be built as well. In some cases this allows to work with both adaptive and non-adaptive filters simultaneously. The data flow is controlled by a control system that is exchangeable, as well as the learning algorithms. These environments allow the development in environments such as .NET and Java, even in other platforms such as embedded systems. Examples of such environments are Peltarion's Synapse, NeuroSolutions from NeuroDimension and Neuro Laboratory of Scientific Software. As free and open source instances Encog and Neuroph can be cited.

A drawback of these environments is that they are much more complex than simulators, hence require higher learning effort.

- Customizable environments

These environments are based on programming libraries of different languages (typically Java and C++) that contain the neural network functionalities and that must be used through the adequate programming.

2.12.2.3 Standards

In order to share network models between distinct applications an XML-based language has been developed, called Predictive Model Markup Language (PMML). It provides a way to define and share neural models from different applications (obviously, PMML compliant). Its users may develop models within a certain platform and then use it in another one to view, evaluate or analyze it. Some of the products that use it are:

- R: produces PMML for neural networks and other *"machine learning"* models
- SAS Enterprise Miner: produces PMML for several data mining models, including neural networks.
- SPSS: produces PMML for neural networks.
- STATISTICA: produces PMML for neural networks, traditional statistical models and data mining.

2.12.2.4 Tools analyzed

Concretely, we evaluated several tools for network simulation from the multiple existent in the market and chose NeuroSolutions ver. 6.0. We will describe the main features found on each one of them:

ECANSE (Environment for Computer Aided Neural Software Engineering).

Developed by Siemens (Austria). Its most relevant features include:

1. ISO9001 certified
2. Uses chaos theory for non-linear analysis
3. Includes genetic algorithms and fuzzy logic
4. Supports several topologies, such as MLPs, Hopfield networks and others.

While it is an industrially interesting product, it has been discarded since its demo version (the only available) stores only 15 objects, and the latest product version was released in march/98, for Windows NT. Furthermore, the user's interface is not appealing.

EXPO/NeuralNet

Developed by Leading Market Technologies Inc., this product is highly oriented to time series, including a special interface for different sources of online information (Reuters, Bloomberg, etc.). This product has a free version for students. We have tested the free version and determined that even though it is fully dedicated to time series it practically does not allow to specify the network design to be used.

Matlab and its Neural Network Toolbox.

Its design interface is poor. Furthermore, the only available error functions are derived from the MSE, not being able to choose L-r norms. The non-stochastic recurrent networks supported include only the Elman's and no other option. It supports feedforward networks, RBF networks, self-organized maps and dynamic networks as well.

NeuroShell

A product from Ward Systems Group Inc. developed for prediction. It uses two basic methods: neural networks and a combination of genetic algorithms and statistical estimation. It can provide a diagnostic of the importance of each input element (sensibility analysis). It has been discarded, since we could not access even to a demo version.

Adaptive Logic Network

It permits to develop supervised learning (in that case the only error function is the L2 distance) or hebbian. An ALN network is a special kind designed for a basic high-speed classification (assign to a pattern class).

Attrasoft

Developed by Attrasoft, Attrasoft Boltzmann Machine (ABM) is a product that permits to simulate Hopfield and Boltzmann networks. It was thought for pattern determination and classification[45]. Its design is not prediction-oriented and the number of possible topologies is limited as well. Last, a demo was not available. Other prediction-oriented products are Predictor and PredictorPro, but they are discontinued.

PREVia

[45]Given a portion of the pattern and a class where it is, determine the rest of it.

Developed by Elseware S.A., this product was designed for time series prediction. It provides an interactive environment for model development. Even though the company still (theoretically) supports this product, we found the documentation of the evaluated version dates from 1996 and its web page inside the company (Elseware) does not exist any more, so we suspect this product has been discontinued.

NeuroSolutions

Developed by Neuro Dimensions Inc., its interface is based on icons, from which the network can be modularly constructed. Additionally, it has an available trial version. The development credits are in part for José Príncipe and Kurt Lefebvre, known researchers in the field of neural networks. It presents a nice graphical interface and generates C++ code (not in all versions), and permits a wide variety of topologies: MLPs, RBFs, Jordan and Elman networks, Hopfield, Kohonen, and others. This product is currently in its version number 6, and it is stable. Furthermore, there is a free version for students, distributed with the book entitled "Adaptive Neural Systems: Fundamentals through simulations". Regarding all these elements, NeuroSolutions was the simulation tool here chosen.

2.12.2.5 Free software tools

In this paragraph we would like to point-out there is a wide variety of free software tools in the market, that cannot be classified into the previous categories. We will mention here some of them [1]:

- ***Neuroph*** is a Java-based framework that can be used to develop standard neural network architectures.
- ***Fast Artificial Neural Network Library (FANN)*** implements multilayer neural networks in C. It is multi-platform, versatile and easy to use. It has adaptations for C++, PHP, PERL, etc.
- ***ffnet*** es a neural network solution for Python. it has some interesting features such as arbitrary connectivity, automatic data normalization and network export to Fortran, among others.
- ***Scene*** is a visual framework that use neural networks and fuzzy classification rules.
- ***FACON*** is a package designed for optimization and neural networks based on Scilab.
- ***Java Data Mining Package (JDMP)*** is a library that provides methods to analyze data via machine learning algorithms (e.g. clustering, classification, graphical models, neural networks, Bayesian networks, etc.).
- ***Icon Sciengy RPF!*** is a Windows application for data mining with self-organized networks. It also allows time series prediction.
- ***OpenAIL [Open Artificial Intelligence Library].*** is a library that provides a toolbox with algorithms used in artificial intelligence (for neural networks, genetic algorithms and others).
- ***OpenNN [Open Neural Networks Library]***. OpenNN is an open source class library written in C++ that implements neural networks.
- ***Recognic*** Is a project that tries to construct high-scale distributed neural networks, used for pattern recognition of time series (such as the ones found in natural language, music and video).

- ***Calamari*** is a collection of Java APIs created to implement genetic algorithms, neural networks and vehicle simulation.
- ***jFAN2*** Is a project (jFAN successor) that tries to develop a Java API to use a neuro-fuzzy network called FAN (Free Associative Neurons).
- ***Amanda Neural Network Project***
- ***Soft-Comuting Library*** is a set of libraries written in C that include neural networks, evolutionary programming, fuzzy systems, etc.

To conclude, the box in the APPENDIX (based on another similar from Colorado University [18]) shows additional features.

Chapter 3

Case study: Gas sales prediction

In this chapter some studies and experiments performed using the time series of the weekly bottled liquefied propane gas[1] sales, and the models designed to predict and dynamically reconstruct this time series are described. Firstly, preliminary experiments are detailed in order to determine some underlying characteristics, such as autocorrelation, stationarity and others. Then, the results obtained with the network models mentioned in the literature as the most adequate for time series prediction (such as TLFNs and recurrent networks) are presented and they are compared with the ones derived from classical architectures such as the multilayer perceptrons (MLPs). Finally, the results produced by different networks committees are presented [55], closing the chapter with an analysis and discussion of the results.

3.1 Data analysis

3.1.1 Statistical study

Since the network design will be highly affected by the underlying relationships of the data, we performed several statistical tests in order to understand them.

Intuitively it is known that:

1. bottled gas is primarily used as a fuel for heat generation (that is, for heating).
2. gas sales depend on the atmospheric temperature (when it is hot, sales go down, and the clients not only stop buying but also try to minimize the usage of heat sources; when it is cold, sales increase), as well as other climatic variables (wind, rain, aspect of the sky, etc.).
3. the sales in a week depends on the sales of previous weeks.
4. in certain weeks of a month, there is a tendency to sell less (the last week) and in others more (in the second, where people generally receive their salaries).
5. the gas price increases directly impact the sales in the previous and following weeks.

[1]In the rest of this work we will refer to the “bottled liquefied propane gas” more briefly as “gas”

The available data is a set of n-tuples (date, minimum daily temperature, peak daily temperature, relative daily humidity, daily rains, daily average of the wind velocity, heliophany[2], gas sales (in liters) for the day), where the temperatures are registered at the INIA's station Las Brujas[3]. There were daily gas sales records from February 1996 until week number 23/2012, even though for an adequate model determination we used the data until week 45/2011 and from weeks 46/2011 to 23/2012 for validation. Due to the dispersion of some data during several years (in 1996 there were only 100 days with information), we decided to work with the available data provided by a corporate data-warehouse, which is considered reliable and did not present discontinuities over the time, so the information was available at the level of days (daily sales) and at level of weeks, months and years, from may 1999 to the date. The determination of the data set to be used (the ones from the data-warehouse or the others with daily sales since February/96) required to develop special tests with different networks (tests omitted here) to analyze the resulting errors. These tests were time-consuming and the decision was by no means immediate. The data were used at the level of weeks trying to smooth the variations present in the daily sales. From the daily data the weekly averages were determined. Additionally, the information of the dates where occurs an increase in the price of gas cylinder was available as well.

In the following paragraphs we describe tests in order to complete the knowledge about the set of existent inter-dependencies.

[2]Heliophany represents the hours of sunlight, and it is related with the fact that the instrument to measure it, the heliophanograph, records the time between dawn and dusk in which receives direct sun radiation. The presence of clouds determines whether the radiation perceived by the instrument will be diffuse or not, possibly cutting the register in the affirmative case. Therefore, although there is enough available energy, its density is not enough to be registered. In other words, it is an indicator of the number of clouds in the sky. A day with poor daylight seems to encourage the use of heating, when other variables are the same.

[3]INIA is the acronym of : "Instituto Nacional de Investigaciones Agrarias",

3.1.1.1 Periodicity

Weekly gas sales (in liters) from September/2000 are shown in Figure 3.10(the x axis represents weeks), being in blue (light grey) the linear tendency.

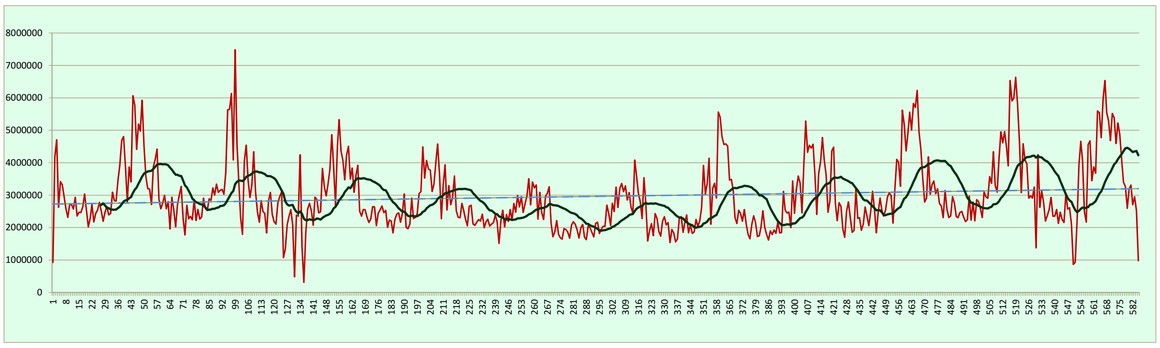


Fig. 3.1 Weekly sales of bottled gas (in m^3)

The graphic with moving averages of 27 weeks (black line) shows a periodicity of about 52 weeks in the series.

Additionally, it seems to exist a mild tendency to consume more gas, if the slope of the linear regression for week sales is considered, shown in Figure 3.1.

If we study the linear correlations between *monthly* sales from different years, we will confirm there is stationarity in the data, and the period is around 52 weeks (this is, one year). For example, the linear correlation between*weekly* sales gives us:

Years	Correlation
2001-2002	0.51
2002-2003	0.65
2003-2004	0.77

whereas the correlations between *monthly* sales (comparing one month with respect to the same month of the previous year) it yields:

2000-2001	0.90
2001-2002	0.88
2000-2002	0.87

In order to check this periodicity and find others, we studied the autocorrelation for weekly sales during the period, and we obtained the results from Table 3.1.1.

Both in Table 3.1.10and Figure 3.20 we have distinct lags (i) in the vertical axis, and the horizontal the correlation between the series and the delayed series with a delay of i weeks. Again, we can appreciate a periodicity of around 50 weeks.

Table 3.1.1: Correlations between the series and the lagged series

Lags	**Autocorrel.**	**Lags**	**Autocorrel.**	**Lags**	**Autocorrel.**
1	0.752879	76	0.049008	151	-0.278802
2	0.634452	77	0.049008	152	-0.263451
3	0.634452	78	0.097857	153	-0.263451
4	0.600686	79	0.097857	154	-0.25766
5	0.600686	80	0.122335	155	-0.25766
6	0.586435	81	0.122335	156	-0.25201
7	0.586435	82	0.212874	157	-0.25201
8	0.550584	83	0.212874	158	-0.263028
9	0.550584	84	0.277901	159	-0.263028
10	0.45983	85	0.277901	160	-0.260536
11	0.45983	86	0.314865	161	-0.260536
12	0.398416	87	0.314865	162	-0.230249
13	0.398416	88	0.334148	163	-0.230249
14	0.386852	89	0.334148	164	-0.226067
15	0.386852	90	0.378154	165	-0.226067
16	0.324083	91	0.378154	166	-0.251682
17	0.324083	92	0.463978	167	-0.251682
18	0.243134	93	0.463978	168	-0.232207
19	0.243134	94	0.50147	169	-0.232207
20	0.162037	95	0.50147	170	-0.182431
21	0.162037	96	0.511494	171	-0.182431
22	0.127499	97	0.511494	172	-0.142942
23	0.127499	98	0.526375	173	-0.142942
24	0.079797	99	0.526375	174	-0.120443
25	0.079797	100	0.551903	175	-0.120443
26	0.005726	101	0.551903	176	-0.120591
27	0.005726	102	0.595384	177	-0.120591
28	-0.076934	103	0.595384	178	-0.059243
29	-0.076934	104	0.559504	179	-0.059243
30	-0.135389	105	0.559504	180	0.006193
31	-0.135389	106	0.506891	181	0.006193
32	-0.151108	107	0.506891	182	0.048404
33	-0.151108	108	0.523388	183	0.048404
34	-0.177557	109	0.523388	184	0.051851
35	-0.177557	110	0.485985	185	0.051851
36	-0.212317	111	0.485985	186	0.097884
37	-0.212317	112	0.475039	187	0.097884
38	-0.22673	113	0.475039	188	0.156828
39	-0.22673	114	0.390929	189	0.156828
40	-0.242474	115	0.390929	190	0.211243
41	-0.242474	116	0.345113	191	0.211243
42	-0.244379	117	0.345113	192	0.241186
43	-0.244379	118	0.315587	193	0.241186
44	-0.274942	119	0.315587	194	0.264273
45	-0.274942	120	0.263605	195	0.264273
46	-0.305592	121	0.263605	196	0.331158
47	-0.305592	122	0.188816	197	0.331158
48	-0.297054	123	0.188816	198	0.368145
49	-0.297054	124	0.11152	199	0.368145
50	-0.281425	125	0.11152	200	0.38142
51	-0.281425	126	0.050646	201	0.38142
52	-0.280846	127	0.050646	202	0.415255
53	-0.280846	128	0.02346	203	0.415255
54	-0.2919	129	0.02346	204	0.420624
55	-0.2919	130	-0.029474	205	0.420624
56	-0.28612	131	-0.029474	206	0.426533
57	-0.28612	132	-0.070272	207	0.426533
58	-0.255216	133	-0.070272	208	0.421202
59	-0.255216	134	-0.124083	209	0.421202
60	-0.245878	135	-0.124083	210	0.394999
61	-0.245878	136	-0.130937	211	0.394999
62	-0.242029	137	-0.130937	212	0.374965
63	-0.242029	138	-0.158848	213	0.374965
64	-0.209794	139	-0.158848	214	0.353318
65	-0.209794	140	-0.198008	215	0.353318
66	-0.15582	141	-0.198008	216	0.321105
67	-0.15582	142	-0.232962	217	0.321105
68	-0.09715	143	-0.232962	218	0.286001
69	-0.09715	144	-0.227006	219	0.286001
70	-0.096469	145	-0.227006	220	0.237502
71	-0.096469	146	-0.235363	221	0.237502
72	-0.075632	147	-0.235363		
73	-0.075632	148	-0.25569		
74	-0.015313	149	-0.25569		
75	-0.015313	150	-0.278802		

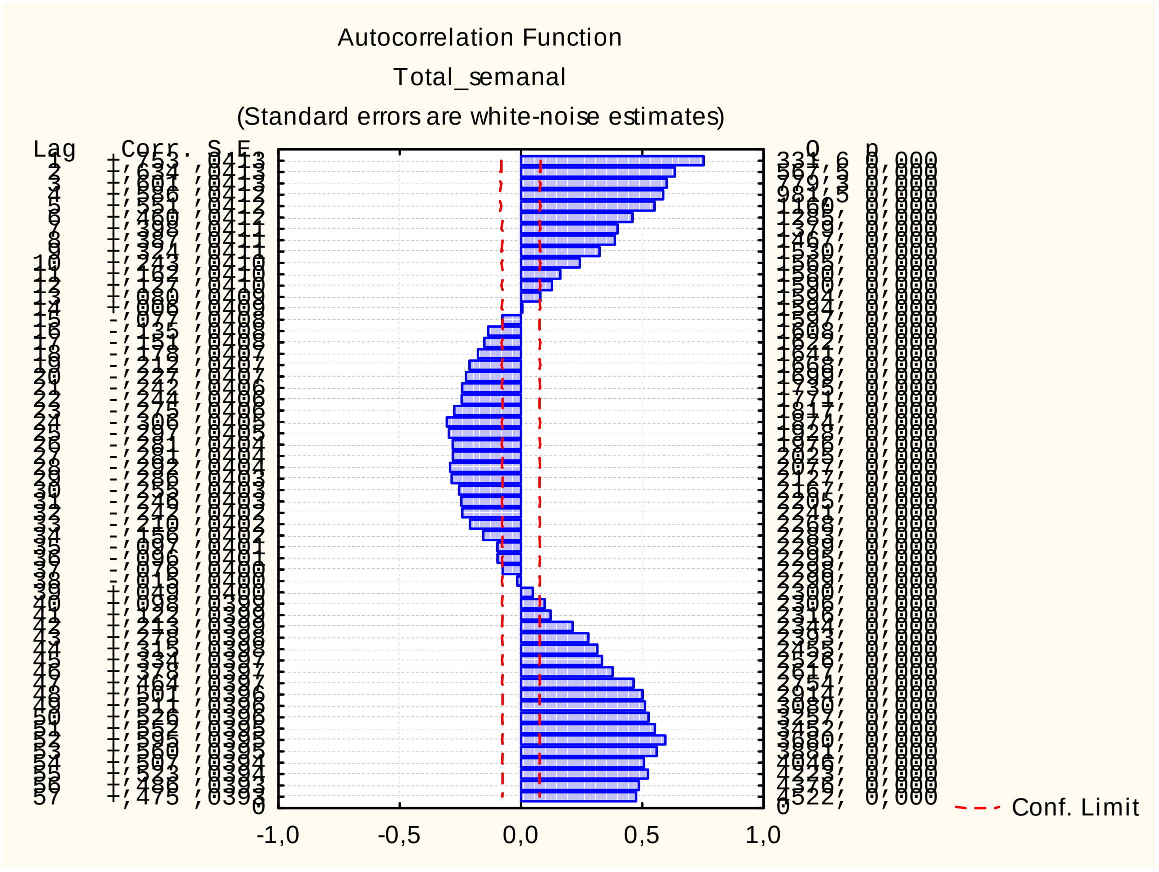


Figure 3.2

We can notice that beyond 3 to 5 weeks back in time, the influence of the weekly sales in the current week's ones is extremely small. This hypothesis will be confirmed in 2.9.4.5.

3.1.1.2 Specific correlations of the weekly sales

Based on the daily sales provided by the data-warehouse, we grouped the information (sales, temperature and corresponding data) into 7 consecutive days. We obtained the following correlations between the sum of the sales and the Maximum-Minimum temperatures:

$\rho(V,T)$	$\rho(V,t-Min)$	$\rho(V,T-1)$	$\rho(V,t{-}Min{-}1)$	$\rho(V,T-2)$	$\rho(V,t-Min-2)$
-0.68	-0.38	-0.66	-0.34	-0.57	-0.27

where:

- ***V*** is the sum of the sales in the group of 7 days
- ***t-Min*** is the minimum temperature averaged in the group of 7 days
- ***T*** is the maximum temperature averaged over the group of 7 days

- ***t-Min-1*** is the minimum temperature averaged in the group of 7 prevoius days to the group of 7 days at hand
- ***T-1*** is the maximum temperature averaged over the group of 7 previous to the group of 7 days at hand
- ***T-2*** maximum temperature averaged over the group of 7 previous days before the previous group
- ***t-Min-2*** is the minimum temperature averaged over the group of 7 previous days before the previous group

Studying these correlations we decided to use the maximum averaged temperature as our temperature of reference.

3.1.1.3 Detrending (tendency deletion)

The literature (for instance, [15]) points-out the importance of to perform the "detrending" (tendency deletion) of the time series in order to improve the performance of a MLP. Such a tendency can be linear, in the general case, or given by another function type (a polynomial with degree higher than two [15]). We will consider from now on only the linear trend. Since the slope of a linear tendency in our data is quite soft[4], the mismatches[5] between training data and testing will be small, so, as a first stage, we tried to train the networks without deleting such data tendency. Other reasons that explain why detrending was not implemented include:

1. we tried to evaluate the learning capacities from different topologies, taking the original series or a sub-series from it (as when the Takens-Mañé theorem is applied), performing the least preprocessing possible to the data.
2. the network is used by a user, and, if a detrending was introduced (using even a function as simple as the linear one), the reconstruction of the prediction (this is, to undo the calculations made for the tendency deletion to get the final prediction) would be something tricky

We trained an MLP with inputs V-3 to predict V, 20 hidden neurons and the original training series, as an empirical checkup for the irrelevance of detrending. The obtained results produced an error %Error CV of around 45%. The same happened when the number of hidden neurons was genetically determined. The we tried with the detrended series (subtracting the straight line to the series, found solving minimum squares) with remarkably worse results (errors higher than 90%), even selecting genetically the number of neurons from the hidden layer. This suggests that perhaps the periodic components for the series should be discarded more than its tendency.

3.1.2 Model Discussion

Some questions concerning the system model to be adopted were:

[4]However, the data seem to have a double periodicity (one is with a period of one year, and the other with period of 12 years), so a more sophisticated method avoiding cyclic components should be used (for instance, the Fast Fourier Transform), that for extension reasons we do not include it here.

[5]In the sense that the testing data have a linear tendency clearly different than the training data, essentially because the different times considered and the respective changes in the market.

- Deterministic or stochastic?

The reality to model is parially governed by physical laws and relationships that are deterministic (for instance, "in winter it´s cold") but there are another factors involved that due to their complexity can be only modeled by stochastic processes (for example, the customers reaction when the gas price is increased). Therefore, the model should include a stochastic component.

- The real system is completely random as a whole or as a stochastic dynamical system?

If we consider the evolution of the weekly gas sales we see its seems to correspond to a process with a deterministic component and another stochastic one. The randomness is constrained (in the sense that not all transitions are possible). Therefore, the system will be either random (with constraints in the possible transitions) or stochastic dynamical. That real system gives rise to a discrete random process of weekly sales $P = \{v(n),\ n = 1, 2...\}$

- How can we model the system and the process $P = \{v(n) \quad n = 1, 2...\}$?

We know, from the study of the series autocorrelation and the AMI, that there is no significant correlation in the sales from weeks i and j provided $j > i+3$. Besides, there is a strong dependence between a certain week and the next one (intuitive by our empirical knowledge of the reality: if someone buys a gas cylinder, it is uncommon that he/she buys another one in the next week). These observations lead us to think about a third-order Markov chain, whose states would be represented by couples (sales, weeks), or terns (sales, weeks, price increase). Other approach could be a third-order discrete Hidden Markov Model (HMM, see [45]), with constraints, whose states are (temperature, week) or (temperature, week, price increase)[6] and outputs (observations) the sales. Since we take the sales as a non-negative integer, that cannot be higher than the maximum storage of the selling company, the number of states is finite. The same consideration holds for temperature, for example, with a decimal digit of accuracy. However, these alternatives require to define the transition probabilities between the states, which is impossible using the scarce information available. The same conclusion holds when we consider a stochastic network model. Finally, the most important consideration is that, even in the case we had enough information, in this work we intend to use the neural networks approach, so the model of the reality could be represented by a neural network, hence discarding HMMs and Markov chains. Having in mind the previous comments concerning stochastic networks, our network will be non-stochastic. As a consequence, given that intuitively we should include the randomness in the model because the real systems has it, and considering the previously mentioned data scarcity, we model the real system as a stochastic dynamical system.

More specifically, the chosen model for the real system is:

[6]or (temperature, rain, heliophany,..., week)

The system state in step $n+1$, $\mathbf{x(n+1)}$ is given by

$$\mathbf{x}(n+1) = \mathbf{F}[x(n), \mathbf{x}(n-1), ...\mathbf{x}(n-T), \mathbf{u}(n+1), \mathbf{u}(n)...\mathbf{u}(n-T)] + \varepsilon(n)$$

being $\mathbf{F}$ *and* ϵ *defined as in 2.9.4.5 (both with integer components) and* $\mathbf{u}(n)$ *the inputs (average week temperatures and other atmospheric variables) at step (week) n.*
The states are represented by the tern (sales, week, temperature) or quadruple (sales, week, temperature, price increase). In some cases (for certain network models) we will consider the system such that the states are represented by n-tuples with higher dimension: (sales, week, temperature, Rel. humidity, heliophany...) or (sales, week, price increase, Rel. humidity, heliophany...).
The output variable from which we have measurements are the sales, v, so:

$$v(n+1) = \varphi[v_{0-\tau}(n), t_{0-\tau}(n), a_{-1-\tau}(n), s_{-1-\tau}(n)] + \delta(n)$$

with
- *v, t, a, s, n the sales, maximum week temperatures, averaged maximum temperature, price increase and week number respectively. This generalizes to the case of using the other atmospheric variables.*
- $v_{x-\tau} = [v(n-x), v(n-x-1), ...v(n-\tau)]$ *and analogously for* $a_{x-\tau}$ *and* $s_{x-\tau}$
- $\delta(n)$ *an integer random variable.*

We assume that there is no noise in the temperature values, so the whole randomness is attributed to the "shift" $\varepsilon(n)$ *that takes a finite number of different integer values, and makes the system to be an IFS (see 2.9.1 on page 88).*

This IFS will have a signal subspace from which we must determine its dimension and we also need to find the optimal τ.

On the other hand, this dynamical system will be represented by means of a neural network [10]. We will use several neural network topologies: all of them non-stochastic. Among these networks, the dynamical ones will be those such that each one will have in turn an associated state-space model. Let us consider for example the Jordan network, in Figure 3.3. In such network we assume the input layer does not perform any process over the data $\mathbf{U}$ and that $\mathbf{F}c$, $\mathbf{F}1$, $\mathbf{F}2$ and $\mathbf{F}3$ are (multivalued) output functions represented by the context layer, the first hidden layer, second hidden layer and output layer respectively and τ is the time constant, as described in 2.4 on page 58. This network, once trained, will have a state-space defined by states $\mathbf{X}^* = \mathbf{X}\mathrm{c}$ and its dynamics will be governed by:

$$\begin{cases} \mathbf{X}\mathrm{i}(n) = \mathbf{U}(n) \\ \mathbf{X}1(n) = \mathbf{F}1[\mathbf{W}c\mathbf{X}c(n) + \mathbf{W}i\mathbf{X}\mathrm{i}(n)] \\ \mathbf{X}2(n) = \mathbf{F}2[\mathbf{X}1(n)] \\ \mathbf{Y}(n) = \mathbf{F}3[\mathbf{X}2(n)] \\ \mathbf{X}c(n+1) = \mathbf{F}c[\tau\mathbf{X}c(n) + \mathbf{Y}(n)] \end{cases} \qquad \textbf{(Eq. 3.1)}$$

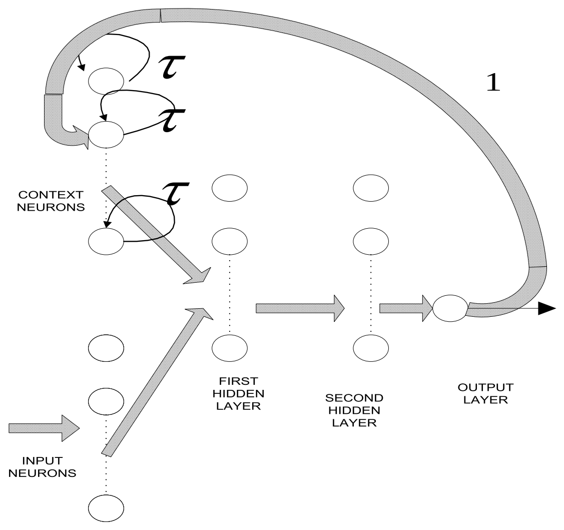


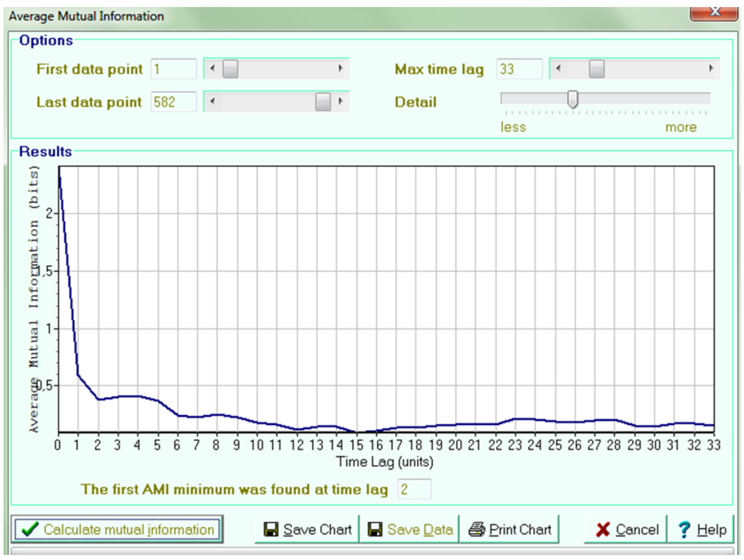
Figure 3.3

Additionally, the weights are modified during the training stage, in accordance with the pairs $(\mathbf{U}, \mathbf{d})$, so in this case the weights for training step n , $\mathbf{W}(n)$, together with $\mathbf{X}*$, would represent the system state at step n and the system dynamics given by Eq 3.10plus the associated with a weight update rule. The system order will then be $dim\mathbf{X}* = dim\mathbf{Xc} + dim\mathbf{W}$.

3.1.3 Other studies

3.1.3.1 Optimal delay

In order to find the optimal delay of a time series we used the "Average Mutual Information" (AMI) of the series respect to itself. This method can be used for both the series coming from deterministic or stochastic systems, since its proposal does not make any reference to the nature of the system, even though in the stochastic case the obtained results are only valid for the random process realization under study. We performed a study of AMI for the series using VRA tool, and we obtained the results from Figure 3.4:


Average Mutual Information

First data point 1

Last data point 582

Max time lag 33

The first AMI minimum was found at time lag 2

Figure 3.4

We can observe that the tools defines a parameter called "detail" that refers to the resolution to draw the AMI, and hence the accuracy to express the first minimum (and therefore, the accuracy of the optimal delay). In products such as TISEAN or DATAPLORE, instead of giving the detail, it is required to make an explicit enumeration of the used parameters in the respective algorithms, e. g. for the case of TISEAN, the number of boxes to use. The optimal delay is 3 for all but some detail levels. Indeed, when the detail is reduced, the optimal delay is turned to 2 (the VRA tool suggests that when the minimum detail is chosen, the optimal delay can change). These values agree with the statistical studies previously performed: the correlation for sales is no further than three weeks.

3.1.3.2 Embedding dimension

Since the data is considered to come from a dynamical system, in order to minimize the network input we studied certain parameters, such as the embedding dimension (see 2.9 on page 88).

Based on the fact that the series is nearly periodic with a period of 52 weeks, the signal subspace dimension should be $2 \leq m \ll 52$ (see 2.9 on page 88). Provided there exist a dynamic noise, that dimension must be added to the shift space dimension, and since the possible values for the noise is finite (so it has null dimension), we conclude that the possible dimensions for the embedding are the same that if the system were deterministic. Therefore, we can find the embedding dimension m with the FNN method, as if the system were deterministic.

DATAPLORE, in its demo version, produced a value of $m = 2$ with a percentage of false nearest neighbors very close to 0% when $d = 2$ and 3.

For $d = 2$, the resulting FNN plot was:

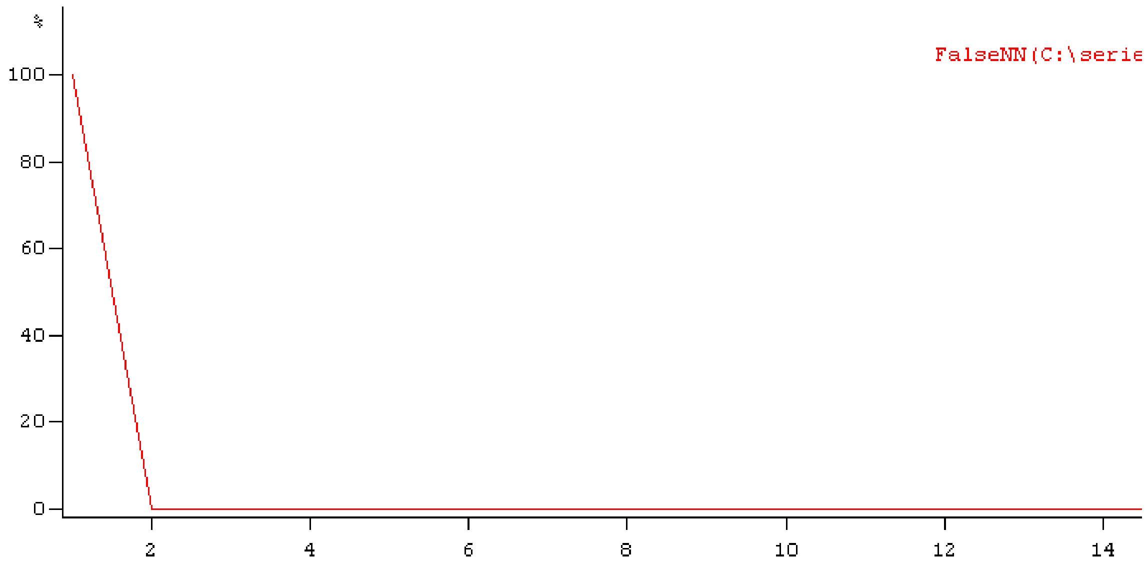


Figure 3.5

For $d = 3$ the same plot was obtained. The parameters introduced were: maximum embedding dimension = 15, distance tolerance (RTOL) = 10.0 and Loneliness tolerance (ATOL) = 2.0.

It is worth to point-out that in the present work we only consider the global embedding dimension since:

- we do not have a software that finds the local dimension d_L
- the results with the global dimension $m \equiv d_E$ are valid (we do not loose information about the embedding when we work with higher dimensions than d_L
- the value for d_E was $d_E \leq 3$, so we do not have many choices for d_L and hence using $d_L$instead of d_E would not imply, in absolute terms, a great dimensional reduction.

We carried-out tests with $m = 2$ and $m = 3$ training a MLP network of 11 hidden neurons in a single layer, with $d = 2$ and $d = 3$. The best results were obtained for $m = 2$, so we considered this is the correct dimension:

- $m = 3, d = 2$ Inputs V(i), V(i-2), V(i-4) predicting V(i+1), with 20 trainings. The minimum %Error CV was 38%.
- $m = 3, d = 3$ Inputs V(i), V(i-3), V(i-6) predicting V(i+1), with 20 trainings. The minimum %Error CV was 40%.
- $m = 2, d = 3$ Inputs V(i), V(i-3) predicting V(i+1), with 20 trainings. The minimum %Error CV was 45%.
- $m = 2, d = 2$ Inputs V(i), V(i-2) predicting V(i+1), with 20 trainings. The minimum %Error CV was 29%.

As a consequence, applying the Takens-Mañé's theorem (for the stochastic case), we can predict x_{i+1} from $\{x_i, x_{i-d}, \ldots x_{i-(m-1)d}\}$. Replacing with $m = 2$, we conclude an MLP that predicts the time series for a single step (x_{i+1}) should have as inputs

$$\{x_i, x_{i-3}\} \text{ for } d = 3$$

$$\{x_i, x_{i-2}\} \text{ for } d = 2$$

being x_i the gas sales for week number i, and as an output, an estimation for x_{i+1}.

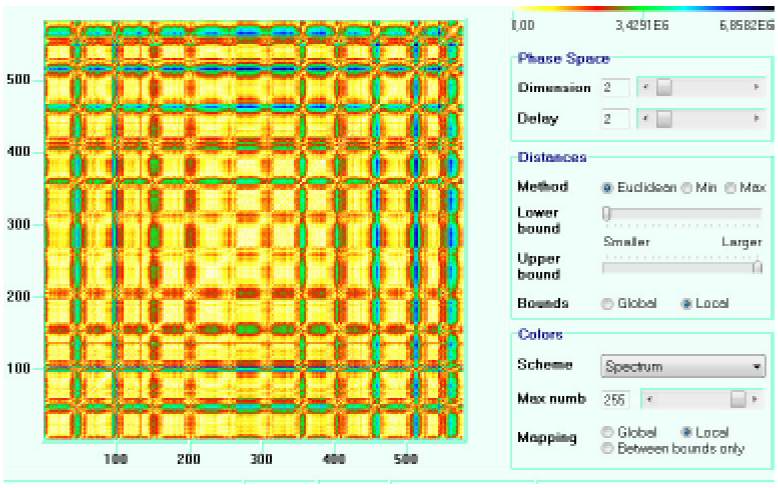
This approximation can always be implemented by a MLP network, due to the universal approximation property of these networks.

3.1.3.3 "Recurrence plot" (RP) of the time series

The tool VRA finds the euclidean distances between the vectors (the distance between all pairs of vectors $\{x_i,\ x_{i-d}\}$) and express them with a color scale (shown in the upper right side of the graphic). In the axis the corresponding index i is represented. The obtained graphic is called a "**recurrence plot**". The color distribution and pattern give us an idea that there exists a certain determinism in the series generator.

We performed a RP of the series by means of the VRA tool for the optimal delay values $d = 2$ and $d = 3$ and embedding dimension $m = 2$, yielding the results shown in Figure 3.6:

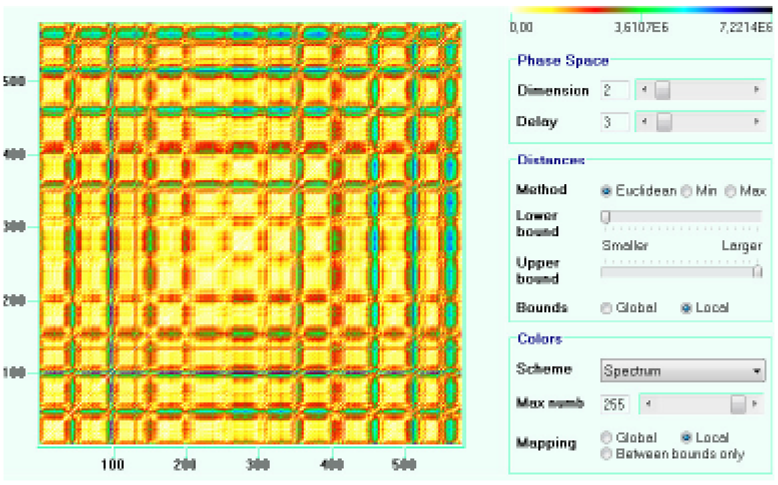
$d = 3, m = 2$**(upper) and** $d = 2, m = 2$ **(lower)**



Phase Space
Dimension 2
Delay 2

Distances
Method [✓ Euclidean] Min Max
Lower bound
Smaller Larger
Upper bound
Bounds Global [✓ Local]

Colors
Scheme Spectrum
Max numb 255
Mapping Global [✓ Local] Between bounds only



Phase Space
Dimension 2
Delay 3

Distances
Method [✓ Euclidean] Min Max
Lower bound
Smaller Larger
Upper bound
Bounds Global [✓ Local]

Colors
Scheme Spectrum
Max numb 255
Mapping Global [✓ Local] Between bounds only

Figure 3.6

Considering the points distribution in the plane (graphic), that are uniformly disperse in patterns in regions that escape from the identity line, we can conclude the process is stationary.

3.1.3.4 Entropies and problem resolvability

Following the approach from [81] we try to determine whether the problem has solution or not, precisely, whether there exists a network able to learn the relation Inputs $\rightarrow$ Outputs, from the input and output entropies.

The relation can be learned by a network whenever

$$\begin{cases} \frac{H(outputs|inputs)}{H(outputs)} \text{ is close to } 0 \\ \frac{I(inputs,\, outputs)}{H(outputs)} \text{ is close to } 1 \end{cases}$$

being $H(\bullet)$ the entropy, $H(\bullet\,|\bullet)$ the conditional entropy and $I(\bullet,\bullet)$ the mutual information.

Although these conditions are simple, we could not determine any software able to estimate the mutual information directly, unless we manually calculate it (through an ad hoc procedure) from the available data. Additionally, we faced the same problem as we encountered when we intended to use a Markov chain: we need a quantity of data that is not available. Therefore, these aspect of the problem could not be directly elucidated, remaining the possibility to prove the resolvability in a constructive way, this is, finding an adequate network model.

3.1.3.5 Predictability of the gas sales sequence

Regarding the volumes of gas sales as a time series, we wonder whether the corresponding dynamical system is either chaotic or not. For that purpose we used the routines of the TISEAN package, specially the Lyap_spec, from which we could find the spectrum of Lyapunov exponents (see 2.9 on page 88). We obtained for embedding dimensions of $m = 2$ and 3:

$$[\lambda_1, \lambda_2]$$

$$-1.591010e-001 \quad -8.755421e-001$$

In all cases the maximum Lyapunov exponent λ_{MAX} was negative (see 2.9 on page 88). This means we are not facing the problems mentioned in 2.9.3.2 on page 98 when the Sano and Sawada´s algorithm is used (by TISEAN) with the stochastic system and that the dynamical system is non-chaotic (at least according to the realization of the process P under study) but stochastic. Additionally, given that it is close to zero, it evidences a cyclic behavior of the system. We must point-out that while the dynamical system is chaotic, it is usually said that the series of observations for the system is chaotic. Briefly, that the time series is chaotic. The fact that the sum of Lyapunov exponents yields a negative value is consistent with the fact that it is a real system.

It is worth to notice that in order to determine the Lyapunov exponents we have applied the algorithms corresponding to a deterministic dynamical system, with no regards of the stochastic part of it. This has been done previously by other researchers ([37][75][88]) even though it is not completely correct due to the random component. Another suggested idea is to estimate the Kullback-Leibler distance between the probability density functions for the system values obtained from two initially close states $\mathbf{x}(t)$ and $\mathbf{y}(t)$. This distance has a similar meaning to that of the obtained Lyapunov exponents (in the sense of convergence or divergence of two trajectories) [77].

3.2 Tests and Experimental results

Tests were carried out for different network models and an analysis of the prediction quality was developed in the open loop mode (by means of the error evaluation in one-step predictions, averaging over the cross validation set and over the weeks called "production weeks": 32 to 45/2011) and of stability (long-term validation) in the closed loop mode, by means of the synthetic series generated by iterated predictions with each model. The data was divided into three sets:

- training set = weeks until 31/2011. From that set the cross validation samples were automatically extracted.
- production set = weeks 32 to 45/2011 from which the weighting coefficients for the networks committees were computed.
- final set = weeks 46/2011 to 23/2012 from which the final tests were carried out comparing the models.

Additionally, since BP algorithm develops a gradient-based search starting from a random point on the error surface, the training stage for each network was performed at least seven times, and the weights and learning rates were chosen as the ones that produced the best results (lowest percentage error). The training mode was always incremental, justified by its advantages over the batch one.

Tests were carried out with total and partially recurrent networks, Jordan networks, TLFNs and MLPs networks. The networks committees were essayed as well ("ensembles" [55]) with different weights. By reasons of extension, we do not included experiments with radial base functions, even though they have also used for time series prediction (for example, [26], [89], [47]).

3.2.1 Data processing

For training purposes, the data must be encoded adequately. This process is called ***preprocessing***. Analogously, it can be necessary some data transformation to the network output data before it can be useful: this constitutes the ***post-processing***. During a preprocessing stage, many authors distinguish between encoding and re-encoding. In the encoding process, the data is converted into a representation such that can be useful to train the network, whereas in re-encoding we just work with the raw encoded data or we try to highlight important features of them or reduce their dimensionality. The Fourier transform of a time series is an example of re-encoding [81]. In our case study, since nearly all the data are numerical, the encoding is rather obvious. Additionally, since one of the goals of this work is to test the power of the most sophisticated topologies (TLFNs and recurrent networks) in order to learn the main characteristics of a time series, the preprocessing (related with a transformation of its values) was minimum.

3.2.1.1 Preprocessing

Classification of data types The network inputs can be either discrete or intrinsically continuous. The continuous are:

1. Temperatures (denoted by T+1, T, T-1, ...), winds, precipitations, relative humidity and heliophany.
2. Volume of sales: denoted by V+1, V, V-1...

On the other hand, the discrete inputs are:

1. Week number (S).
2. Indicator of the existence of a price increase in the current, previous or next week.

However, we encode in such a way that some continuous variables are translated into integers:

1. the gas sales volume in liters as an integer number.
2. the weeks with a number form 1 to 53, to highlight the periodic characteristic of the series.
3. the indicator of price increase in the gas sales:
   (a) if in the current week occurred a price increase, we choose increase = 0
   (b) if the current week is previous to one with increase we choose increase = -1
   (c) if there is an increase in two weeks, we choose increase = -2
   (d) if there is an increase in more than two weeks, we choose increase = -3
   (e) if the week is exactly after one with increase, we choose increase = 1
   (f) if the current week is two weeks after an increase, we choose increase = 2
   (g) if the current week is more than two weeks after an increase, we choose increase = 3

   Observe that the same value is assigned when the price increase occurs three or more weeks after or before the current week: this means there is no influence of the increases in the sales (speculative influence with increase of sales before a price increase or sales reduction after it) in the long-term.

Data representation

Physical representation The training and test data sets were represented in adequate formats for the chosen simulator (NEUROSOLUTIONS). ASCII files were considered to work with, formatted in columns, where each variable represents a ***channel*** (in the tool terminology) be either input or output.

Data normalization The input data were normalized automatically by the software tool in the following manner:

- for each input variable i it is computed

$$Amp(i) = [UpBound(i) - LowBound(i)]/[Max(i) - Min(i)]$$
$$Shift(i) = UpBound(i) - Amp(i) * Max(i)$$

where

- $Max(i)$ is the maximum value found for variable i
- $Min(i)$ is the minimum value found for variable i
- $UpBound(i)$ is a constant, by default 0.9
- $LowBound(i)$ is a constant, by default -0.9

- the $j$ht value of variable i is normalized by:

$$Value(i,j) = Ampl(i) * Value(i,j) + Shift(i)$$

With this normalization, all values from variable i are now within the range $[LowBound(i), UpBound(i)]$ which helps avoid neuron saturation during the training phase.

This is a linear scaling, which keeps the original structure of the data: if the values were uniformly distributed in the universe, the linear scale would be uniformly distributed as well. If a non-linear transformation were implemented, the "mis-balance" in the training data would have been magnified, grouping them in more sparse sets [81]. Besides, when the output data must be presented, the normalized data are denormalized, undoing the transformation and hence obtaining the output. Due to these denormalizations, in some experimental results the desired values for the series appear as a decimal number, when they were originally integer. For instance, 1821053.88 instead of 1821054, as a product of the propagation of the errors in the normalization-denormalization process.

Finally, LowBound and UpBound should be adequate to the output function to be used. In the case of hyperbolic tangent the previous values are adequate, but when the logistic curve is used, we should work with a LowBound positive and close to zero.

Outliers deletion Clasically, outliers are removed in order to "smooth" the data behavior. However, their removal means a loss of information that can be signaling a fact that is ignored due to the scarce data about it. In fact, it might be that the information provided by these outliers were of a paramount importance for the user. If outliers were numerous, a deeper study of their respective circumstances and of the outliers determination criterion could be done, in order to understand their subjacent generating causes, and a model specialized in the processing of the "extreme data" (outliers and neighbors) could be developed.

Two different approaches can be used to determine an outlier.

1) Consider the sales only and define an outlier as sales that "escape" from the "data mass": we performed a "box and whiskers" plot, retrieving the graphic from Figure 3.7:

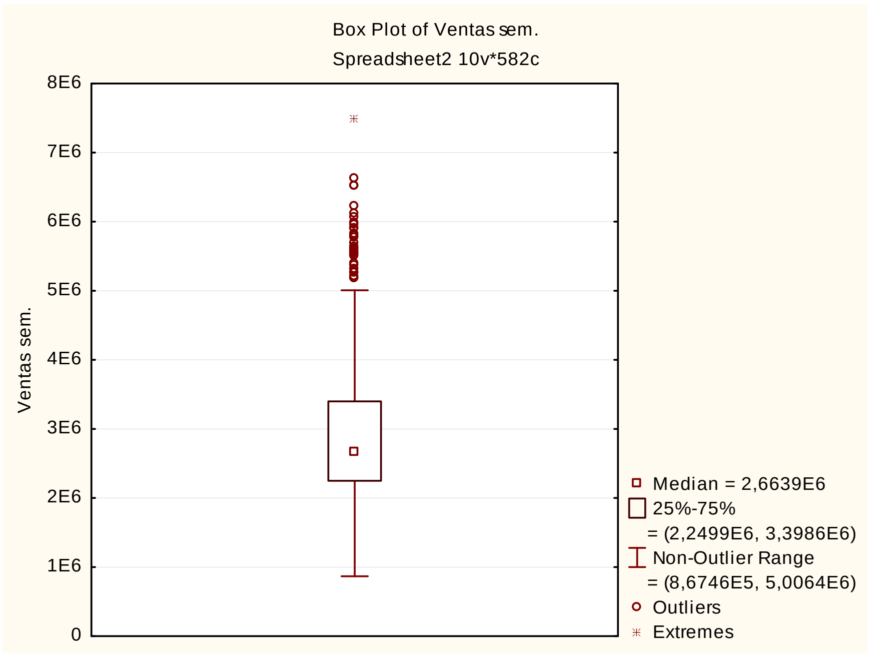


Figure 3.7

this means that outliers would be sales values greater than $5.0064E6$ or smaller than $8.6746E5$

The box and whiskers plot is a useful tool to describe the data behavior around the median; it uses the median, upper and lower quartiles (75th and 25th percentile respectively).

More formally, a *value* is an outlier if

$$value > UBV + o.c. * (UBV - LBV) \text{ or } value < LBV - o.c. * (UBV - LBV)$$

where

- UBV is the upper bound value for the box shown in the previous diagram, in this case the 75th percentile.
- LBV is the lower bound value, in this case the 25th percentile.
- o.c. is the outlier coefficient. In this case we used 1.5 (the default value for outlier calculations in STATISTICA). With other coefficients different number of outliers are obtained; for example with $o.c. = 2$ a single outlier is obtained, whereas an $o.c. < 1.5$ would produce more than two outliers. We chose 1.5 since 2 outliers seem an adequate number of outliers for the available volume of data. Some authors define "moderate outliers" and "extreme outliers" according whether the factor o.c. used to determine the outlier is 1.5 or 3 [55]. We are working then with moderate outliers.

To summarize, taking the median as the center and an outlier coefficient $o.c. = 1.5$ we can see the outliers as the values above $3.3986E6 + 1.5 * (3.3986E6 - 2.2499E6)$ or below $2.2499E6 - 1.5 * (3.3986E6 - 2.2499E6)$. Many outliers were detected. Since data are scarce, we might replace that value for an average of the sales in the contiguous weeks [81].

2) An alternative is to consider the other dimensions that determine the sales (temperature, heliophany, precipitations, wind velocity, relative humidity) and use the Mahalanobis distance between each vector (sales, temperature, heliophany, wind, precipitations, relative humidity) and the respective mean (barycenter):

$$(\overline{sales}, \overline{temperature}, \overline{heliophany}, \overline{wind}, \overline{precipitations}, \overline{relative\,humidity})$$

Figure 3.8 presents the graphic with the so-obtained distances, the horizontal line representing the limit for the distance of a point in order to be considered an outlier:

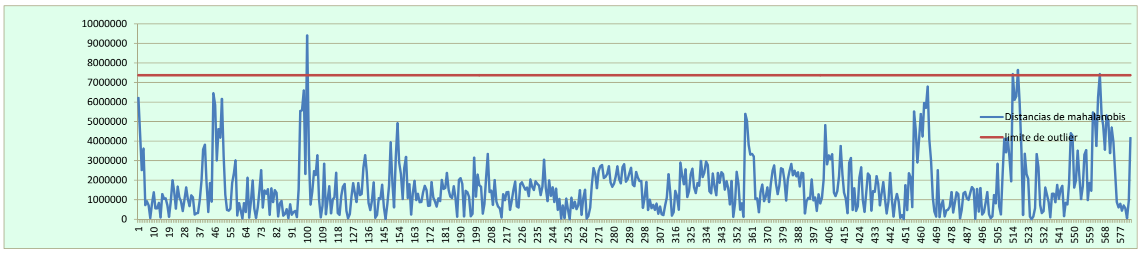


Figure 3.8

With a statistical analysis of this series we have

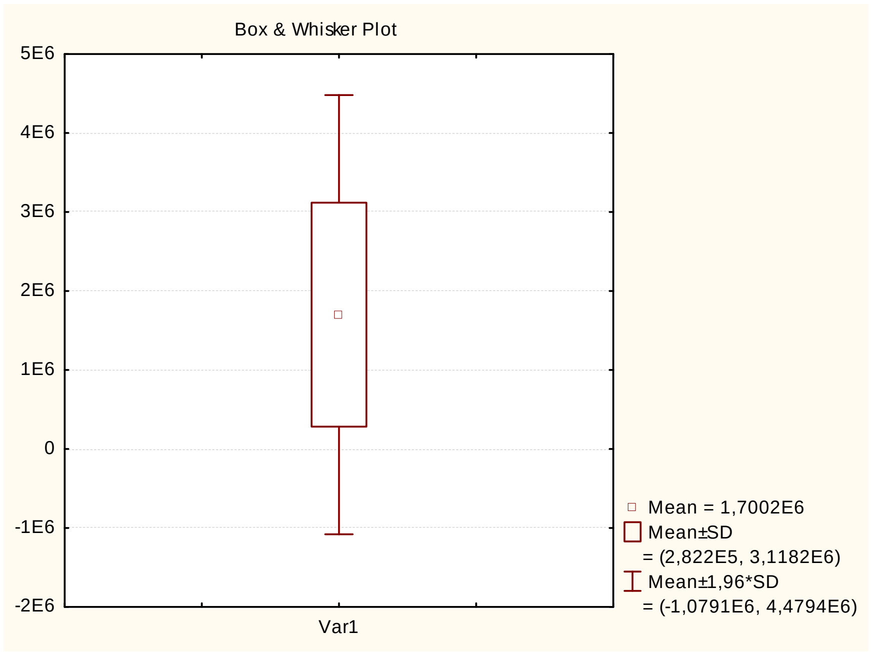


Figure 3.9

so the distances such that $distance > 3.1182E6 + 1.5 * (3.1182E6 - 2.822E5) = 7372200$ would mean extremely rare combination of sales, temperatures... that might be ignored. That threshold distance is marked with a solid horizontal line in the previous graphic. The outliers would be in this case the ones for weeks 99, 513, 516, 564. Specifically:

Week 27/2002 : 7481043

Week 28/2010: 6531135

Week 31/2010: 6636083

Week 27/2011: 6532075

In a first stage we decided not to remove the outliers and observe the results. Then, for the best networks, we tried using a deletion of outliers from the series (being replaced by an average of their adjacent values, in all dimensions). We did not consider for that deletion the “final values” nor "production values".

Data construction for training and testing The available data about the sales come from two sources: reports generated by a data-warehouse with the daily billings summarized at week and month levels, and reports of the the gas sales from the Accounting Department of the company. In the epochs of higher gas demand, there may be some discrepancies between these sources (for example because some sales are dispatched at the first time of the next day, before office hours), while in the rest of the year they match. Additionally, the data-warehouse is available only since may/1999. Here we decided to use only the data-warehouse data, since we considered that its information was reliable, because it is used by the whole Accounting Management Area of the company and ignored the other source. About the weather information, the data were provided by a meteorological station (“Las Brujas”). The average temperatures were found taking the temperatures from Monday to Sunday. This element introduced a difference of around 2.8% in average with respect to the average of the days on which the company bills gas (Monday-Saturday). This difference is considered as a noise in the temperature measurement and we ignored it (see 4.2 on page 171).

Dimensionality reduction Given that the feasible inputs of a MLP might be V, V-1,... T+1, T, T-1...in order to predict V+1, we decided to apply the Takens-Mañé’s theorem to try to determine the necessary inputs to reconstruct (predict) V+1. Additionally, as a part of the preprocessing we could wonder which are the most meaningful network inputs, using for instance principal component analysis (PCA) or a feature extraction technique (see [10], [36], [81]). Here we did not employ these techniques here. Instead, genetic algorithms were used for an optimal network design.

3.2.1.2 Post-processing

The only post-processing applied was the data denormalization, which is an automatic process in our case (as the data normalization is).

3.2.2 Tests for different topologies

3.2.2.1 Architectures and models considered

- We define a ***sample*** as a consecutive set of values of the time series, as long as the trajectory to be learned. An ***epoch*** is the presentation of all samples to the network input.

- In all tests carried out we considered the MSE over a set of patterns (for example the sets from cross validation set) defined as:

$$MSE = \frac{\sum_{i=0}^{N}(d_i - y_i)^2}{N}$$

being N the number of samples in the set, d_i, y_i the desired and output values for the sample i respectively.

- The averaged percentage error was found by

$$\%Error = \frac{100}{N}\sum_{i=0}^{N}\frac{|y_i - d_i|}{d_i}$$

where N, i , d and y have the same meaning that for the MSE.

- The MDL was estimated using the following formula:

$$MDL = NLog(MSE) + \frac{1}{2}kLog(N)$$

being k the number of network weights (see 2.7 on page 80) and the logarithms are natural.

- Additionally, in the chosen networks simulator we could not determine, from the existent documentation and carried out tests, if the same sample set for CV was considered each time the network was trained, once the CV set is specified a percentage of the training set. Due to that fact, the MDL over the CV set is not only one MDL into account, but the network was trained at least seven times and the best MDL was selected (since the model is the same with the exception of its weights, it is equivalent to choose the one with the lowest MSE) as the reference for that model.

We always trained using as a stopping criterion the following:

"if after 50 consecutive epochs there is no MSE reduction over the CV set, the learning phase is stopped".

Basically, three big classes of networks were tested: multilayer perceptrons, TLFNs (including TLRNs) and recurrent networks (such as Jordan/Elman networks). The best model of each one was determined in terms of %Error in the production set (weeks 32 to 45/2011). All models were validated according to the rules stated in 3.3 on page 158. Besides, a voting scheme ("ensemble method") was defined, in order to get a better prediction from the individual networks.

The short-term validation criterion was to check whether the averaged prediction error over the production was higher than 10% or not. The long-term validation was not performed exhaustively, since the final object of this work is the prediction of the time series and not the underlying model. By that reason the long-term validation was done, as an illustration for the perceptron which had produced optimal results. Finally, after the best models were determined the predictions for the weeks 46/2011 to 23/2012 were performed as the final evaluation of the prediction quality.

Best results As a first attempt it sounds adequate to choose the network according to the MDL[7](choosing the minimum MDL, see 2.7 on page 80). However, since we are concerned with the sales predictions at least for one week in the future, no matter the model complexity, we will choose first counting the number of time-steps with admissible error predictions (v. g. not higher than 10%) and then using the estimated MDL[8]. The best models found were then (for each one of the network classes):

a) for MLPs: the best MLP network is the one with inputs V(i) , V(i-2) that predicts V(i+1), 25 hidden neurons with two layers (25/2), with no additive noise in the inputs.

[7]In fact an estimation for the MDL is used instead, automatically found by the simulator, as described in 3.2.2 on the previous page

[8]This does not mean the model is considered as valid for prediction, see 3.3 on page 158.

b) for TLFNs: a TDNN network corresponding with an optimal delay for the series $d = 2$, non-focused memory, only V as input and output V+1, with 16 hidden neurons in a single layer.

c) for recurrent networks: a Jordan/Elman network with inputs V,S,S-1,V-1, T-1 and output V+1, with time constant $\tau = 0.15$ (see 2.1.4.2 on page 32) two hidden layers with 4 and 2 neurons respectively.

d) for voting schemes the best combination is the one that produces the lowest error over the final data, and is obtained with $\alpha = 0.5$ using weights computed from the production data. Specifically, in Table “ERRORS WITH FINAL DATA AND ENSEMBLES" is:

d.a) Ensemble of networks b) and c), weighted according to the optimal coefficients suggested by Bishop [11].

d.b) Idem but the coefficients found using the error variances and $\alpha = 0.5$ (see 3.2.3.1 on page 141).

Observe that for a networks ensemble we have no definition for the MDL (at least from the consulted literature).

The resulting values for these models are summarized in Table “ERRORS SUMMARY” and “ERRORS WITH FINAL DATA AND ENSEMBLES” (for the meaning of model d), see 2.1.2 on page 24):

ERRORS SUMMARY

	a)	b)	c)
MDL computed over the CV set	**64.09**	**81.08**	**-68.79**
% Error over the CV set	**30.22%**	**23.35%**	**20.63%**
Model	**d=2, 25/2 25 hidden neurons in two layers**	**d=2, memory no-focused, 16 hidden neurons in one layer**	**4/2 hidden neurons, Jordan/ Elman as described**
Time-steps with error < 10%	**1**	**2**	**7**
Prediction error in a time-step for week 32/2011	**0.30%**	**4.40%**	**8.85%**
Averaged Error from week 32 to 45/2011	**13.95%**	**23.12%**	**13.02%**
Maximum Error	**30.64%**	**28.53%**	**29.20%**
Minimum Error	**0.30%**	**0.26%**	**1.64%**

ERRORS WITH FINAL DATA AND ENSEMBLES

	a)	b)	c)	d)	
				d.a)	d.b)
MDL found over the CV set	**N/A**	**N/A**	**N/A**	**Non- defined**	**Non- defined**
% Error over the CV set	**N/A**	**N/A**	**N/A**	**Non- defined**	**Non- defined**
Model	**d=2, 25/2 25 hidden neurons in two layers**	**d=2, no-focused memory, 16 hidden neurons**	**4/2 hidden neurons, Jordan/ Elman as described**	**b)+c) and optimal weighting**	**b)+c) considering the error variance**
Time-steps with prediction error < 10%	**0**	**1**	**7**	**0**	**2**
Time-step prediction error for week 46/2011	**25.40%**	**5.19%**	**24.55%**	**13.01%**	**7.04%**
Average Error from week 46/2011 to 23/2012	**28.17%**	**27.05%**	**29.28%**	**27.95%**	**17.56%**
Maximum Error	**219.07%**	**213.71%**	**223.52%**	**217.68%**	**104.64%**
Minimum Error	**1.09%**	**0.73%**	**7.33%**	**4.84%**	**3.87%**

N/A: Non-applicable.

Although the %Error in production seems to be the ideal indicator to choose the model to be used, the choice only based on this indicator can lead to very poor predictions, as it is discussed in 3.3 on page 158, so it is pertinent to wonder the circumstances where a model is valid and hence eligible.

Observation:

A reason of why the MDL estimation is sometimes negative can be that the MSE is calculated from the normalized inputs and not from the original values (which real magnitudes are in the order of millions). In this way, the MSE is found using numbers in $(-1, 1)$, and the estimation would give negative values.

3.2.3 Valid models

3.2.3.1 Short-term valid models

We can consider the networks (models) valid in the short-term as locally valid, since they approximate the series a few steps in the future from T known steps in the past. The goal of this work is the prediction using a neural network that will be periodically re-trained (where the period should be determined). Three models were detected to be possibly valid[9] in the short-term (see Table "ERRORS SUMMARY") so they are useful for this time series forecast since they produced a one time-step prediction error (in week 32/2011) not higher than 10%. We assumed that 10% is a reasonable limit for the prediction error, even though this value should be discussed with the user (precisely, the person who uses the neural model as a tool in their routine tasks). Additionally, it should be agreed with the user what is an acceptable error variance. In this work we did not include any selection based on that value, but we used it to weight the values in a committee. It would be also interesting to choose the models whose production errors have an acceptable variance and that predict acceptably good the series in one time-step..

These models might use a fixed number of historical data (for example, the sales of the last 300 weeks) and not all the available data. The Jordan/Elman network (the network g) in the Table "SUMMARY RECURRENT NETWORKS" might be added to the short-term valid networks, as well as the models obtained by a combination (ensemble) of these networks.

In all the cases the transfer function was the hyperbolic tangent, using the gradient-based method (BPTT) modified with a first-order term (moment) with constant moment rate.

Among the elementary models the best one according to the MDL was a Jordan/Elman network with inputs V,S,S-1,V-1, T-1 and output V, with time constant $\tau = 0.15$ ***with two hidden layers of 4 and 2 neurons respectively)***

Model selection for prediction The model obtained as two-networks ensemble produced the best result in the sense of the lowest prediction error: it gave the best approximation, regarding both

1. the average percentage error to predict for the weeks 46/2011 to 23/2012, and
2. the maximum prediction error

and it was obtained weighting the predicted values for the two valid networks b) and c) from Table "ERRORS WITH FINAL DATA AND ENSEMBLES" according to the coefficients derived using the following weights:

$$p_j = \frac{\frac{1}{\%EP_j + Var(\%EP_j)^{\alpha}}}{\sum\limits_j \frac{1}{\%EP_j + Var(\%EP_j)^{\alpha}}}$$

[9]Since we additionally want to have an acceptable error in the final data set, it is required to study the averaged error over those data.

being $\%EP$ the average error percentage produced using the production data and taking $\alpha = 0.5$, called model **d.b)** in Table "ERRORS WITH FINAL DATA AND ENSEMBLES".

When four networks were intended to be used in the same committee the coefficients employed were (some values for α are omitted):

COEFFICIENTS FOR COMMITTEES

	Coefficient			
Model	**"optimum"**	$\alpha = 1$	$\alpha = 0.5$	$\alpha = 0.2$
MLP a) from Table "ERRORS SUMMARY"	0.110	0.210	0.220	0.235
TDNN b) from Table "ERRORS SUMMARY"	0.137	0.238	0.243	0.248
TLFN f) from Table "SUMMARY FOR TLFNs"	0.020	0.326	0.301	0.273
Jordan/Elman network c) from Table "ERRORS SUMMARY"	-0.167	0.226	0.235	0.244

By optimum we understand the ones suggested by [11] and detailed in 2.1.2 on page 24. On the other hand, since working with four network can be hard (the predictions should be performed for each one, and sometimes it means to train them again), we reduced the number of networks to two. The values so obtained with a Jordan/Elman network with a TDNN committee were satisfactory (see 3.3.0.12 on page 163). This model is called **d.b)** from the same table.

The chosen model to predict will be the average of the values predicted by a Jordan/Elman network and a TDNN (see 3.3 on page 158)

In the following figures we can observe the errors improvements obtained when a network committee is implemented, using different weights:

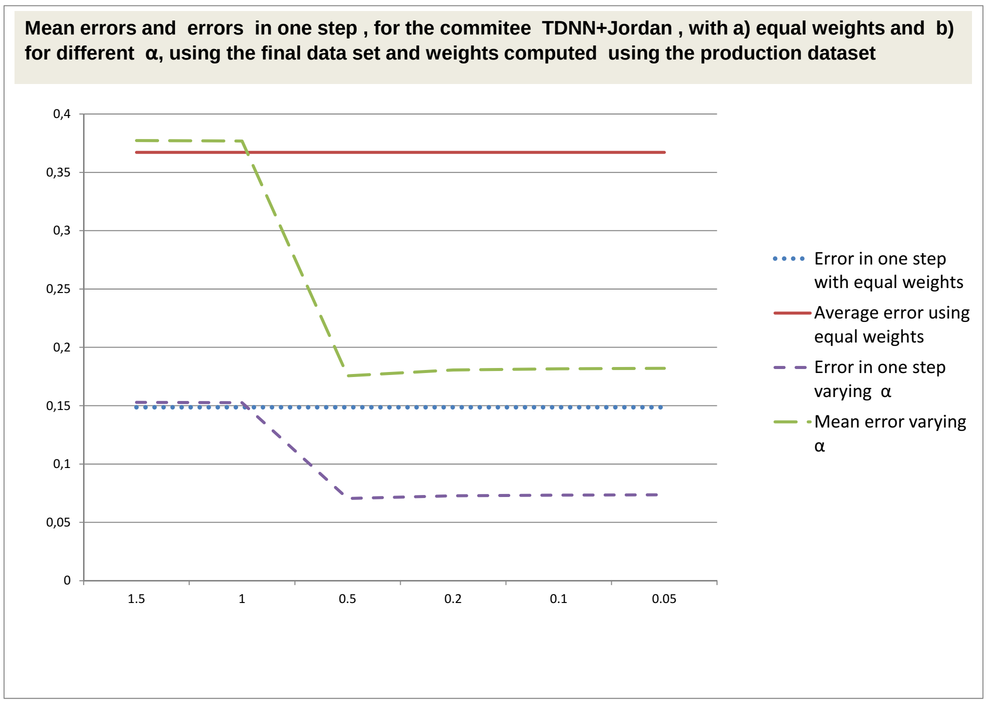


Figure 3.10

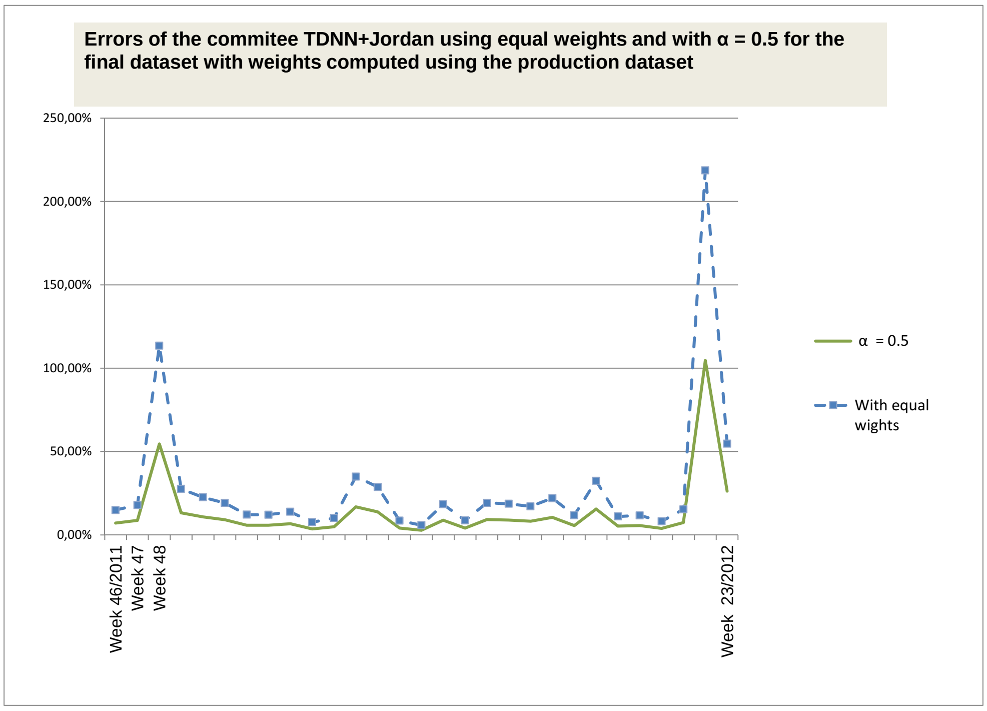


Figure 3.11

Long-term valid models In the case of deterministic systems the study of long-term validation is usually made to check that the model can reproduce the dynamic behavior of the original system, usually considering the invariants of the original system and of the predicted series in a closed loop mode. We will not focus on the discussion of what means “similar” for those invariants. For stochastic dynamical systems a probability density function corresponds to each invariant. Moreover, in some works comparisons between the probability densities of some invariants (the correlation dimension and Kolmogorov-Sinai’s entropy) between the original system and the series generated by the model [79]) as a long-term validation criterion. In the stochastic case, we further know that the most we are able to predict is a mean value (associated with the conditional probability that the system visits a certain state in step n+1 knowing the past states n, n-1,...n-T), while the real value for the series will be the value associated to this mean, plus a random variable. A long-term model could then be constituted by a network that predicts that mean (that implements function φ mentioned in 2.9.1.1 on page 89) and optionally other model for the prediction of the random variable (the δ in 2.9.1.1 on page 89). That second model could be non-parametric (for instance a density adjustment), a networks mixture model, a feed-forward network or even another statistical model. Since we have only 14 (32 a 45/2011) values to adjust the resulting errors, we will not try to determine such error prediction model, nor discard solutions based on the long-term validation. The long-term validation checking could be replaced by an iterated calculation for the production weeks[10] and their respective errors. In that case, we prefer those network with the lowest average error, and in that sense we would say the network has a good long-term behavior. Simultaneously, among those models with good behavior in the long-term with respect to the iterated errors, we will choose those models with "similar" invariants compared with the original series.

Another criterion could have been, given the final application for the model, to say for instance that we have a valid long-term model whenever it is able to predict a time-step ahead at least, without re-training, with an acceptable error upper-bounded error, for data that is non-adjacent to the training data. In this case, the model would have been the d.b) from Table “ERROR SUMMARY WITH FINAL DATA AND ENSEMBLES”.

The networks that presented a good behavior in the long-term, in the sense of the lowest iterated averaged percentage error, were (in increasing average error order):

1. The Jordan network from c) previously mentioned
2. TDNN b) from Table “ERRORS SUMMARY”
3. MLP a) from Table “ERRORS SUMMARY”

being the network 1 the most advisable. In Table 5.4.10the resulting errors are shown, obtained via an iteration of the values for each network, for the weeks 32 to 45/2011 and are graphically illustrated in Figure 3.12.

[10]The denomination that appears in the literature for the network performance testing data is just “testing data”. However, here we stick to the denomination used by the simulator for the data set that the network never received as input and from which the performance is calculated.

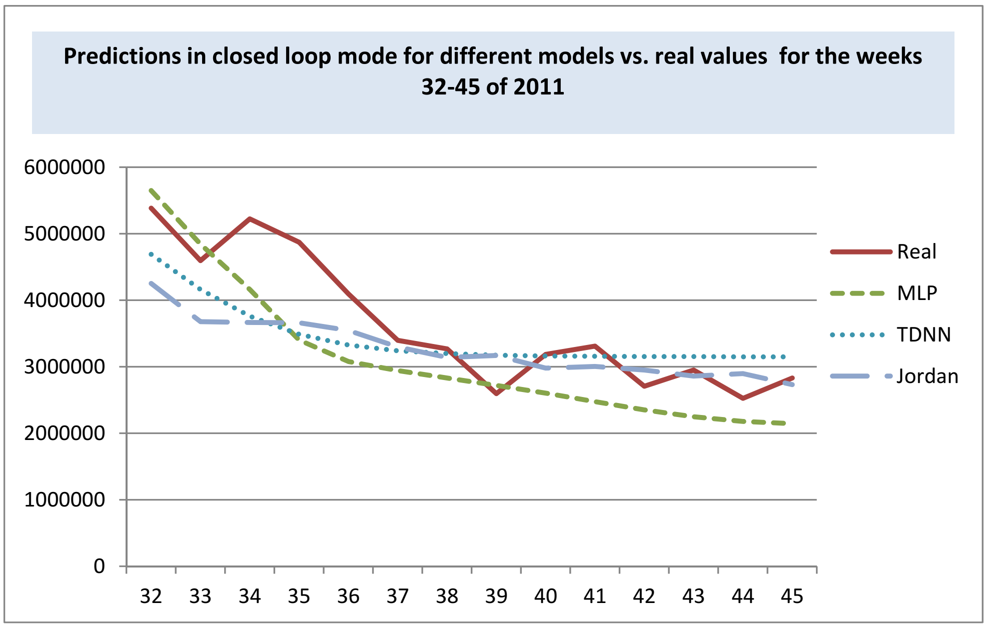


Figure 3.12

Models with valid short and long-term behavior With respect to the models valid for both long and short-terms, this is, that could be potentially employed for both prediction and dynamic reconstruction, they will be the ones with the lowest iterated error (ordered by increasing error):

- a Jordan network with additional inputs, two hidden layers with 4 and 2 neurons (see 2.1.4 on page 30)
- a totally recurrent network with inputs S, T, V-1, output V, 7 hidden neurons in a single hidden layer
- a TLFN with gamma memory, inputs S, A, V-1, T-1, S-2, V-2, T-2 output V , 17 hidden neurons

(in that order of preference). If we ignore the constraint that the one-step prediction error (or average error) be lower than 10%, these models might be recommended to users that intend to predict the time series using only one network. If the constraint is not relaxed, there would not exist elementary models valid in the long-term, even though an ensemble between a TDNN and a Jordan or between a MLP and Jordan networks would achieve the long-term condition.

We can appreciate that the ensemble method with optimal coefficients applied to the iterated values does not imply any improvement (see Tables 5.3.10and 5.3.2) comparing the average and one-step prediction errors, even though it reduces its variance. The variance is reduced even more with $\alpha = 0.05$, but the error is increased in the three-networks ensemble (Table 5.3.4). Finally, for the MLP+Jordan (the committee obtained with the best MLP and the best Jordan/Elman network) with optimal weights found from the production set and using $\alpha = 0.5$ an acceptable value yields (see Table 5.3.60 "Errors for different alphas, MLP+Jordan, closed loop, prod. weights") . The same happens with the TDNN+Jordan ensemble.

3.2.4 Best results classified by architecture

Here we detail the best results for each one of the architectures and its variants.

3.2.4.1 Multi-layer perceptrons

3.2.4.2 Perceptrons based-on the time series only

MLPs were one of the models under study, representing a classical topology used as point of departure in works similar in nature to ours ([71], [64], [2], [17]).

From the obtained results for the determination of the optimal delay and embedding dimension, tests were carried out with a perceptron with

Delay	Inputs	Outputs
$d = 3$	$\{x_i,\ x_{i-3}\}$	$\{x_{i+1}\}$
$d = 2$	$\{x_i,\ x_{i-2}\}$	$\{x_{i+1}\}$

where x_i represents the weekly sales for the week i, $i = 1.2, ...53$.

We always used networks with a single hidden layer or two hidden layers at most, because 1) in theory, one layer is enough to approximate any smooth function and 2) we have no interest to speed-up the training phase, since it consumes few minutes of computational effort. The number of neurons in within the hidden layer were genetically determined through 1500 generations.

The best network was one with inputs V, V-2, 25/2 hidden neurons, with no additive noise in its inputs (Network **a)** from Table MLP SUMMARY).

3.2.4.3 Perceptrons with additional inputs

We tried to improve the overall performance of the MLPs using networks with inputs genetically determined from the set T, S, A, V-1, A-1, S-1, T-1, V-2, A-2, S-2, T-2, V-3, A-3, S-3, T-3 and also the average wind, precipitations and heliophany for that week. The obtained model was **b)** from Table **"MLPs SUMMARY".**

MLPs SUMMARY

	a)	b)	c)
MDL found over the CV set	***64.09***	21.79	-60.21
% Error over the CV set	***30.22%***	23.31%	16.50%
Model	***d=2, 25 hidden neurons in the first layer and 2 in the second***	24 hidden neurons in one layer	4/5 hidden
Time-steps with prediction error < 10%	**1**	0	0
Time-step error prediction for week 32/2011	***0.30%***	28.50%	34.16%
Average error from week 32 to 45/2011	***13.95%***	27.99%	21.47%
Maximum error	***30.64%***	53.81%	34.16%
Minimum error	***0.30%***	27.99%	1.94%

a) MLP with inputs V, V-2, 25/2 hidden neurons, and no additive noise in the inputs.

b)MLP with inputs S,V,V-1,V-2, Aver T Max, Aver T min, Aver precip, Aver wind, Aver heliophany and 24 hidden neurons[11].

c) MLP with inputs: S,V,V-1,V-2, Aver T max, Aver T max previous, Aver T min, Aver precip, 4 neurons in the first hidden layer and 5 in the second one.

Observe that even if the estimated MDL allows to choose between models of similar complexity, the intuitive choice (Network a) here does not match with the minimum MDL choice, perhaps because the complexities of networks a) and c) are completely dissimilar.

To summarize, although the best MLP from the MDL perspective was that with inputs S, V, V-1, V-2, Aver T max, Aver T max previous, Aver T min, Aver precip, that predicts V(i+1) with 4/5 hidden neurons, with no additive noise in its inputs, it is preferred a network with inputs V, V-2 predicting V+1, with 25/2 hidden neurons for which (see Table 5.1.6):

MDL found over the CV set: 64.09
% Error over the CV set: 30.22%
25/2 hidden neurons in two layers
Time-steps with prediction error < 10% = 1
Prediction error for week 32/2011 = 0.30%
Average error from week 32 to 45/2011 = 13.95%
Max. error: 30.64%
Min. error: 0.30%

We tested with other MLP as well, as can be seen in 5 on page 175.

[11]Using the notation explained at the beginning of this chapter, it would suffice to speak about T and t, but we prefer here to denote in a most descriptive way: "Aver T Max".

Perceptrons with additive noise in the inputs

Remarkable improvements were observed in the prediction errors when a MLP network is trained with the time series altered with an additive noise, see 5 on page 175.

Results with preprocessing (outliers removed)

In some cases valuable improvements were obtained by treating the outliers of the series (determined using the Mahalanobis's distance) and then using the modified series to train the network:

Table 3.2.1: Results obtained by treating the outliers in the series

Network a)	**W/original series**	**W/treated outliers**
Max. Error	30.64%	21.00%
Min. Error	0.30%	2.85%
Aver. Error	13.95%	10.42%
Prediction steps	1	4
Network b)		
Max. Error	53.81%	57.40%
Min. Error	6.01%	6.21%
Aver.Error	27.99%	33.42%
Prediccion steps	0	0
Network c)		
Max. Error	49.09%	34.16%
Min. Error	3.33%	1.94%
Aver.Error	26.78%	21.48%
Prediccion steps	0	0

It can be appreciated that the outliers processing significantly reduces the prediction error (maximum and average) in a time-step for networks a) and c) but worsens the results for b).

Time lagged feed-forward networks (TLFNs)

Another network class considered here was the "time lagged feedforward networks" (including TLRNs). In a first attempt, we exploited the results obtained by the study of the embedding's dimension and optimal delay, so we performed tests "remembering" using delay = 3 or delay = 2 past elements.

We studied both focused and non-focused memories (see 2.4 on page 58). We used a single hidden layer, finding the number of hidden neurons by means of genetic selection with more than 1500 generations of 50 to 60 individuals each. The training was always on-line, using a moment term with a non-adaptive moment rate. The percentage of data used data for cross-validation was 15% of the training data set. The memory elements under consideration were delay memories and gamma (type I), the formers because they represent memory structures as classical as perceptrons are respect the neural networks, and the former because several experiments can be performed with them (such as to change the memory depth with no need of changing the topology), even though some tests were also carried with Laguerre's memories. As in the remaining of this work, all trajectories to be learned had length 4. For both memory types, we tried as a first approximation to predict the next value of the series using Takens's theorem, this is, presenting only sales to the network inputs (a "pure" network) and using the memory parameters (taps and delays) to implement the delayed inputs $\{x_i,\ x_{i-d}\}$ being $d = 2$ or 3. As a second approach, we added averaged maximum temperatures, the increase indicator, other meteorological data and a week number, for the current week and two previous weeks as well as previous sales. These inputs were genetically determined, as justified in 2.3.2.1 on page 57 and [27].

With respect to the genetic algorithms employed and its parameters, we can remark that:

1. In order to keep the terminology consistent with the one used by the network simulator chosen here, each feasible solution (in our case, a network) is called a chromosome. A chromosome is composed by a collection of genes, that are precisely the network parameters that we want to optimize genetically.

2. The genetic algorithm creates an initial population (a collection of chromosomes) and then scores it training networks that correspond with every chromosome. Then the population is evolved through multiple generations, trying to find the best network parameters. If the issue is to optimize a certain network parameter, the population is evolved (changing that parameter) looking for the network with the highest fitness. When the parameters define the own network structure (for example when the number of neurons within a layer must be determined), the evolution searches the network with the best structure, rather than an optimal parameter.

3. The evolution is performed via generations: the whole population is replaced after each iteration. This method has shown good results in a great variety of cases, and tends to avoid local minima in a better way than when the members with lowest fitness are discarded during each iteration [68].

4. Each generation had 50 or 60 individuals. In some preliminary tests performed with populations of 100 individuals, better results were not found, and the process turned out to be unnecessarily long and slow.

5. We tried to minimize the one-step prediction error percentage over the cross validation set, in order to find a network with good generalization capacities. The percentage error is significant, since the magnitude of the resulting errors is always higher than 1, and minimize it is equivalent to minimize the MSE given a fixed number of CV exemplars (see 3.3.0.8 on page 158).

6. The chromosomes that still exist in the next generation (that passes to the next generation) were chosen with a probability proportional to their fitness.

7. The crossover probability between two chromosomes was 0.9. This crossover is produced in a single point (randomly chosen) from the genes of the chromosome. For example, given the parents A and B:

A 11001|010

B 00100|111

After a gen exchange from the parents in the cross point randomly chosen (marked with a |), the following successors are obtained:

A1 11001|111

B1 00100|010

No test were made with other crossover operators (see 2.3.2.1 on page 51)

8. The mutation probability of a chromosome is 0.01. The uniform mutation operator was chosen.

9. The population was evolved for at least 1500 generations, and it can evolve even more, and the stopping criterion was the absence of improvement in the fitness in more than 500 consecutive generations.

10. In all networks the hyperbolic tangent output function was chosen (recommended by [68]) in all its neurons, a single hidden layer, on-line training, percentage for the CV set= 15%, improved gradient descent with a fixed moment rate (non-adaptive). One-step predictions were always considered.

It could be appreciated that

- The final quality of a genetic method strongly depends on the number of generations.
- As a general rule, the training phase rapidly augments when the number of generations is increased: when the number of hidden neurons is chosen with 1500 generations, the execution takes more than 10 minutes in an ordinary desktop PC (core 2 duo, with 4 GB of RAM), versus 15 seconds that takes to train[12] a network with fixed topology (with 7 hidden neurons). For the case of 5000 generations with a population of 100 individuals, it takes more than 1.50 hours.
- Perhaps, better results could be obtained with a careful adjustment of the optimization parameters in the genetic algorithm. This adjustement would exceed the scope of this work.

[12]This means to train the network from the training data and using the corresponding stopping criterion. We do not consider the time required to to perform several training phases and select the one with the best results.

3.2.4.4 TDNNs

We used for training the weeks 35/2000 to 31/2011 and the weeks 32/2011-45/2011 for testing. In a first attempt, we tried to use "pure" TDNNs, such as described in the Takens-Mañé´s theorem: using the delay and embedding dimension previously obtained and as inputs only the weekly sales in order to predict the sales for the next week. A single hidden layer was considered, with a number of neurons genetically determined, a "cross validation" percentage of 15%, an on-line training.

*The best result was given by TDNNs with the sales as the only input and V as the output prediction, which is called Model **b)** from Table "TLFNs SUMMARY".*

3.2.4.5 TDNNs with additional inputs

In order to improve the prediction, we added explanatory variables to the input. Working with $d = 3$ and $d = 2$, the best results were obtained with a TDNN with $d = 3$, focused memory, inputs S, A, T-1 (genetically determined among the possible inputs: S, A, T, S-1-A-1, V-1, T-1, S-2, A-2,V-2, T-2) output V and 8 hidden neurons which constitutes the Model **c)** from *Table "TLFNs SUMMARY"*.

We can appreciate the introduction of explanatory variables seems to improve the MDL but the one-step prediction, even though the average percentage prediction error (over the "testing" set or CV set) is better in the "pure" TDNN.

Results with preprocessing (outliers)

The best TDNN was trained with the series with the outliers treated. An improvement of more than 50% in the maximum, minimum and average errors was achieved, with a variance reduction in magnitude of one order. See Table 5.1.18 from 5 on page 175.

3.2.4.6 TLFNs with gamma memory (TLRNs)

Analogously to TDNNs, we tried to apply the Takens-Mañé's theorem using gamma memory type structures. Tests were carried out under the same scenarios of TDNNs, but with gamma memory types[13] of $depth = 15$, even though we tried other depths as well. In both cases the length for the trajectory to be learned was 4. For the focused case with $d = 3$, 25 hidden neurons were determined, for $d = 2$, 22 neurons, whereas for the "non-focused", 19 and 21 for $d = 3$ and $d = 2$ respectively. The best TLFN network was obtained with a non-focused gamma memory with $d = 2$, 21 hidden neurons with inputs V, V-2 and output V+1, that we Model **e)** in Table "TLFNs SUMMARY".

Analogously for TDNNs we tried to improve the one-step prediction performance adding the price increase indicator variable and some previous weeks data explicitly, explained in what follows.

TLFNs with gamma memory and additional inputs

Taking the MDL into account, the best network had inputs S, V-1, T-1, S-2, V-2, T-2, precipitations, winds, heliophany, output V and 17 hidden neurons with focused memory, called Model **h)**.

[13]Gamma type 1

TLFNs SUMMARY

	a)	*b)*	*c)*	*d)*	e)	f)	g)	h)
MDL found over the CV set	*84.41*	*81.08*	*-373.29*	*547.74*	**335.22**	**1428**	**357.02**	**618.59**
% Error over the CV set	*25.82%*	*23.35%*	*13.03%*	*24.63%*	**19.18%**	**22.58%**	**20.12%**	**25.63%**
Time-steps with prediction error< 10%	*0*	*2*	*0*	*0*	**0**	**0**	**0**	**0**
Prediction error in week 32/2011	*29.45%*	*4.40%*	*49.73%*	*24.48%*	**21.78%**	**14.04%**	**21.79%**	**24.48%**
Average error for weeks 32 to 45/2011	*23.12%*	*23.12%*	*31.58%*	*12.15%*	**13.06%**	**12.15%**	**13.06%**	**12.15%**
Maximum Error	*40.81%*	*28.53%*	*40.41%*	*25.61%*	**29.63%**	**35.45%**	**29.63%**	**25.61%**
Minimum Error	*2.58%*	*0.26%*	*2.75%*	*0.68%*	**2.49%**	**35%**	**2.49%**	**0.68%**

Architectures:

a) TDNN pure non-focused, input V, $d = 3$, 16 hidden neurons.

b) Idem a) and $d = 2$.

c) TDNN with $d = 2$, focused memory, inputs S, A, T-1, 8 hidden neurons and $depth = 10$.

d) Idem c) $depth = 3$.

e) TLFN, gamma memory type with $depth = 15$, $trajectory = 4$, non-focused, $d = 2$, 15 hidden neurons with inputs V, V-2 output V+1.

f) TLFN genetically constructed. Inputs: T, T-1, Aver. precipitations, Aver. wind, gamma memory with $depth = 10$ and $trajectory = 4$, focused, 18 hidden neurons.

g) TLFN with gamma memory, predicts V, Inputs: V-1, T-1, 8 hidden neurons.

h) TLFN Inputs: S, T,V-1, V-2, T-1, Aver. precipitations, Aver. wind, predicts V+1, $taps = 3$, $delay = 1$

In summary, the best TLFN network was the b).

Tests changing the memory type

For networks e) and f) we replaced their memories by Laguerre's ones, with no improvements.

Predictions for the final data set

TDNN b) produced an acceptable one-step prediction, whereas no TLFN output any acceptable prediction.

Recurrent networks

The recurrent architectures tested here were

a) Totally recurrent:

1. Inputs S, T, V-1 and output V
2. Inputs S, T, V-1, V-2 and output V
3. Inputs S, T, V-2 and output V

b) Partially recurrent The same inputs and output than the totally recurrent ones

c) Jordan and Elman's networks

Cases a) and b) (Totally or partially recurrent networks):

The hyperbolic tangent transfer function was used in all the cases, and 7 hidden neurons, in a single layer.

In one of the tests performed we used the logistic as the output function, since its range (positive integer values) seemed to be more adequate and would return a lower error. However, we obtained a worse approximation than using hyperbolic tangent.

The training phase employed an incremental weight updating, using BPTT.

The CV set contained a 15% of the training set.

The prediction was always for one-step ahead.

The best results from totally and partially recurrent networks were obtained with a partially recurrent network, with inputs S, T, V-1 output V and 4/2 neurons in two hidden layers (Network c), where:

MDL found over the CV set: -97.28
% Error over the CV set: 21.01%
4/2 hidden neurons, partially recurrent
Time-steps with prediction error $< 10\% = 0$
Prediction error in week 32/2011= 20.77%
Average error from week 32 to 45/2011 =10.72%
Max. Error: 22.29%
Min. Error: 1.36%

Case c) (Jordan and Elman's networks):

As a particular case of recurrent networks we tried Jordan and Elman's networks. The network simulator provides several possible topologies for Jordan's networks, depending whether the context neurons reach the first or second hidden layers (referred here by one-step or two-step links, respectively, see [68]).

The chosen network had the topology described in Figure 3.13:

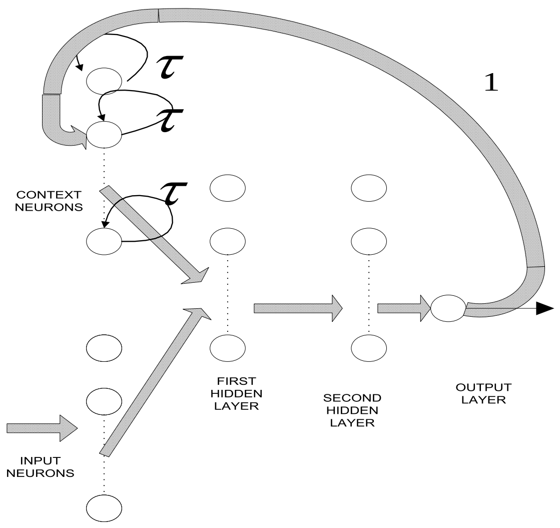


Figure 3.13

In order to simplify the diagram, the wide arrows represent all possible connections. The inputs are V, S, S-1,V-1, T-1 and output V+1. Percentage of CV data=15%, two hidden layers, four neurons in the first hidden layer and four in the second (denoted by 4/2), all output functions are hyperbolic tangent and on-line training. several values for the time constant were tested $\tau =: 0.5, 0.3, 0.7, .0.15, 0.07$.

The time constant τ has the following meaning: given a context neuron **Z,** we have that

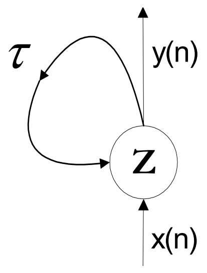


Figure 3.14

where $y(n) = \sum_{i=0}^{n} x(i)\tau^{n-i}$ holds. The time constant τ controls the exponential decay by which the past data impacts the current outputs, in other words, it controls the neuron memory.

For different values of the time constant τ different networks were obtained, where the network with the best results had $\tau = 0.15$ (Network b), with the following performance:

MDL found over the CV set: -69.78
% Error over the CV set: 20.63%
4/2 hidden, described topology
Time-steps with prediction error < 10% =7
Prediction error in week 32/2011= 8.85%
Average error from week 32 to 45/2011 =13.02%
Max. Error: 29.20%
Min. Error: 1.64%

Results with the treated series

When the chosen Jordan/Elman was trained with the outliers previously treated, both the maximum error and error variance improved, even though the one-step prediction was not acceptable any more (see Table 1.38).

3.2.4.7 Adding the price increases to the model

One of the causes of fluctuation of the time series is the presence of increments in the price of gas sales. In previous weeks of an increase there is usually a higher demand, by speculative reasons, whereas in the following week there is a slight reduction triggered by the increase. We intended to reflect this adding a descriptive variable, INCREASE, that can assume different values:

- -2 if the current week is before two weeks to that were an increase occurs,
- -1 Idem, but one week,
- 0 if the increase occurs in the current week,
- 1 if the current week is exactly after the increase,
- 2 if the current week occurs exactly two weeks after the increase, or
- 3 otherwise.

In a conflicting case, we give priority to the fact of being previous rather than following the increase (for instance, if the current week that is just after an increase and two weeks before another increase, the current week is assigned -2). In the case of Jordan networks, we trained a network with inputs S, A, A-1, V-1, T-1 and output V. We used time $\tau = 0.07$ and the remaining variables are identical to these previously mentioned networks. The network obtained, Model **g)** in the Table "RECURRENT NETWORK SUMMARY", was:

MDL found over the CV set: -59.15
% Error over the CV set: 20.97%
4/2 hidden, described topology
Time-steps with prediction error < 10% =6
Prediction error in week 32/2011= 9.94%
Average error from week 32 to 45/2011 =16.84%
Max. Error: 34.66%
Min. Error: 2.19%

In summary, the Jordan network that produced the best results had inputs V,S,S-1,V-1,T-1:

MDL found over the CV set: -69.78 %
%Error over the CV set: 20.63%
4/2 hidden neurons, described topology
Time-steps with prediction error < 10% = 7
Prediction error in week 32/2011= 8.85%
Average error from week 32 to 45/2011 = 13.02%
Max. Error: 29.20%
Min. Error: 1.64%

Summary for recurrent networks:

SUMMARY RECURRENT NETWORKS

	a)	b)	c)	d)	f)	g)	h)
MDL found over the CV set	175.97	-70.54	**133.43**	-0.70	*-69.78*	*-59.15*	-15.09
% Error over the CV set	27.13%	25.22	**28.15%**	19.78%	*20.63%*	*20.97%*	21.94
Model	9 hidden	9 hidden	**4/2 hidden partially recur-rent**	4/2 hidden, totally recurrent	*4/2 hidden, with links in context neurons in a single forward step, time= 0.15*	*4/2 hidden neurons topology described, tie constant = 0.07, 2 steps forward*	7
Time-steps with prediction error < 10%	0	0	**0**	0	*7*	*6*	0
Prediction error with for week 32	34.85%	25.06%	**20.77%**	25.94%	*8.85%*	*9.94*	25.06%
Average error for weeks 32 to 45	16.34%	11.34%	**10.72%**	13.82%	*13.02%*	*16.84%*	19.28%
Maximum Error	34.85%	25.06%	**1%**	25.94%	*29.20%*	*34.66%*	131.17%
Minimum Error	5.29%	0.52%	**1.36%**	3.05%	*1.64%*	*2.19%*	0.52%

a) Totally recurrent with inputs S,T, V-2 and output V, 9 hidden neurons.
b) Idem, partially recurrent.
c) Partially recurrent, inputs S, T, V-1 output V and 4/2 hidden neurons.
d) Idem but totally recurrent.
f) Jordan network with inputs V, S, S-1,V-1, T-1 and output V, two hidden layers, four neurons in the first and two in the second, all output function hyperbolic tangent, time constant: 0.15.
g) Idem f) but with inputs S, A, A-1, V-1, T-1 and output V, time constant = 0.07 and context links two steps forward.
h) Totally recurrent network with inputs S,T,V-1, V-2 and output V, 7 hidden layers.

Some tests were not included in this box. For more details, see 5 on page 175.

3.3 Model validation

Generally in the literature (for example in [30]), in order to study networks for time series prediction, specially chaotic and deterministic, short-term from long-term validities of the models are distinguished.

A way to validate a model in the short-term could be to train the network for a fix number n of consecutive weeks (a "cluster"), and to predict the time series for week $n+1$, the next week under study. Repeating the process several times for different clusters of adjacent weeks, the average percentage error for those predictions can be found checking hence whether the model, regarding its topology and parameters that define it, is valid (if the the averaged prediction error is lower than a certain threshold) or not. This short-term validation method has at least two drawbacks:

- it just tells us whether the topology and some parameters that define the network (for instance, the time constant in Jordan/Elman's networks) are adequate to the problem, since the weights of the model change in each case (when training with each set of weeks)
- it is sensitive to the selection (composition) of the clusters

On the other hand, if we wish both to predict and perform a dynamic reconstruction of the system, we will add some constraint to consider a short-term model valid as a long-term model as well: a model constructed for time series prediction is valid in both senses if the following conditions are met ([59] and [30]):

1. the model has a good short-term behavior: the model predicts "sufficiently well" at least for one time-step in the future (week 32/2011) and satisfactorily for the final data set.
2. the model captures both the structure and properties of the underlying system dynamics that generates the time series, this is, it presents a well behavior in the long-term.

As a consequence, we will use valid models for the short-term in order to predict, and among them, some to reconstruct the system behavior. Additionally, after the valid short-term models are determined, the selection of them will be performed by means of the MDL criterion. The model selection based only on a MDL estimation is not feasible, since low values for the MDL can be obtained not implying that the model has an acceptable short-term behavior, for example, a high prediction error can be compensated by the model simplicity, producing a very low MDL estimation.

3.3.0.8 Short-term validation

The short-term validation is associated with tests in "open loop" mode. Under this mode, the trained network receives the real values of the time series as an input, and the goal is to predict it a step further, given these past values.

The easiest way to evaluate the prediction success is to compute the averaged MSE over a set of N data samples (exemplars), being N the number of CV exemplars:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(d_i - y_i)^2$$

where d_i, y_i represent respectively the real value and the predicted one for the time series in the next time-step, respectively . It is clear we wish low values for the MSE, and further, we want to have the prediction error to be small in relation with the magnitude for the predicted value. On the other hand, if we wish to have an error bounded between 0 and 1, the root-mean-square (RMS) could be used instead:

$$\begin{cases} RMS = \left[\sum_{i=1}^{N}(y_i - d_i)^2\right] / \left[\sum_{i=1}^{N}(d_i - \overline{d})^2\right] \\ \overline{d} = \frac{1}{N'}\sum_{i=1}^{N'} d_i \end{cases}$$

being $N´$ the number of different desired values.

Patel [59] recommends to use the signal-to-error ratio, defined by

$$SER = 10\log_{10}\frac{MSS}{MSE}$$

being $MSS = \frac{1}{N}\sum_{i=1}^{N} d_i^2$. The idea is to have big values for the SER, meaning the predicted value is meaningful (high in magnitude) in relation with its respective prediction error [59].

Another way to measure the prediction quality is to consider the resulting prediction error for a time-step, averaging over the CV set (denoted by %Error CV). Since the values of the time series under study are always above the unit, this percentage is a significant measure of the absolute error produced.

Additionally, to minimize the %Error CV is equivalent that to minimize the MSE over the CV set, since if we find the partial derivatives in each case and equal to zero, the same extreme points are obtained:

$$\frac{\partial \%ErrorCV}{\partial y_i} = \frac{100}{N}\frac{1}{d_i}sign(d_i - y_i)$$

and

$$\frac{\partial MSE}{\partial y_i} = -\frac{2}{N}(d_i - y_i)$$

We will use a variation for this distance to decide the short-term validity of a model: since we have no certainty that the simulator always uses the same CV set, we slightly modify the definition replacing the CV set by the final data (final weeks):

A model will be valid in the short-term provided the following conditions are met:

- ***its prediction error (as a relative error) is less than 10% at least in one time-step (for week 32/2011 and the next ones) and***
- ***its average error over the final data is less than 10%.***

The choice of 10% is arbitrary and should be verified with the model user. Additionally, since the error percentage is a significant indicator for the MSE, to minimize this percentage is equivalent to maximize the SER.

3.3.0.9 Long-term validation

In order to know whether the model captured the whole structure and properties of the system that generates the original series, a "closed loop" mode test is performed. In this test, used in the related literature in the modeling of deterministic systems, the network operates depending on its proper previous outputs (instead of the previous real series values) in each time-step, and on a copy of the inputs the system had in the past. In this mode the evaluation cannot be started without a previous ***"priming phase"***, in which from the real (known) values the necessary inputs for future steps are predicted, taking in each step another additional prediction as an input, until it is predicted considering the network´s outputs only[14], obtaining finally the reconstructed signal from the original one.

The network outputs will accumulate errors in such a way that even an optimal model will produce divergent trajectories with respect to the original series due to the dependence to the initial conditions and the noises included in the vector with witch the priming phase starts. Observe that in the stochastic case the possible divergence caused by dynamic noises is also present. In order to compare quantitatively the dynamic structure for the two series, when the original corresponds to a deterministic system, we would find the following values (***invariants***) for both the original and reconstructed series [59]:

- embedding dimension m
- complete Lyapunov spectrum, $\lambda_i, \quad i = 1...m$
- correlation dimension $\boldsymbol{D}$
- Kaplan-Yorke's dimension, expected to be close to $\boldsymbol{D}$
- Kolmogorov entropy, found summing the positive Lyapunov exponents
- predictability horizon

A satisfactory model, in the deterministic case, would be the one in which the reconstructed series has very close invariants (ideally, the same) to that of the original series. It is worth to note that it is not a trivial task to decide whether the sets of invariants is similar or not. As previously mentioned, in the stochastic case we could compare the probability density functions of the invariants with the respective ones of the original system. Another choices could be:

- Statistically study the original versus generated series (by "closed loop") and check whether they come from the same probability density functions or not (using Mann-Whitney's U test) and use the Lavene's test to determine whether they have the same variance or not [76].
- Invariants generalization: see which is the meaning of the different invariants and try to find a correspondence in the case of stochastic systems. For example, since the Lyapunov exponents determine the convergence of the obtained trajectories from two given states, we can generalize this concept to the stochastic case and determine the distance between the probability density functions of the states obtained from two starting points, using the Kullback-Leibler distance. The negative Lyapunov exponents for the deterministic case would be translated into "close" density functions. Besides, it should be studied which is the analogy of other invariants under a stochastic scenario.

[14]plus the necessary inputs that are not generated by the network (such as the temperature)

Considering the small size and high dimensionality of our time series, we will instead chose those models with the lowest averaged error error when the predictions are iterated for the weeks 32-45/2011 ***and we will say a model is valid in the long-term if that error is below 10%.***

3.3.0.10 Determination of the valid models

3.3.0.11 Valid models for dynamic reconstruction

Having the one-step prediction error-bounded condition in mind, the only models feasible of a long-term analysis are shown in the Table “ERRORS SUMMARY” and some of the shown in the Table “ERRORS WITH FINAL DATA AND ENSEMBLES” and additionally several networks committees as we will see (we include the networks committees in our discussion in what follows). Adding the condition that the average error in “closed loop” $< 10\%$ only the ensembles remain.

To verify the condition that the model captures both the structure and underlying system properties generated by the time series, it could be predicted in a closed loop mode for weeks 32-45 for the chosen models and get the iterated prediction errors. To make a closed loop prediction in the cases where additional inputs are required (increments, temperatures, etc.) the procedure here developed was to repeat the inputs of the original data.

We took the reconstructed values only for weeks 32-45/2011, since they represent the whole available data (for further tests the last weeks were included), and it does not mean we cannot reconstruct a longer period with those models.

The following table shows the results. As an illustration[15] we include some invariants for the optimal MLP (in which it was iterated more than 30 times):

Invariant	*Opt. MLP*
Opt. delay:	3
Embedding dimension:	2
Lyapunov spectrum:	-3.64 E-1 -1.61E0
%FNN if m=2	0% if $\varepsilon < 0.0001$
Kaplan-Yorke's estimated dimension	0.0
%Average iterated Error (over 40 predictions)	20.74%

All AMIs were found using maximum delay=99 and maximum graphical detail for the mutual information, using the VRA.

According to the second condition from 3.3 on page 158 (considered as a similarity of invariants), this optimal MLP would be a valid system representation, since its Lyapunov exponents are negative and it has the same optimal delay and embedding dimension. If we choose the condition that average prediction error over the testing data set must be less than 10%, we have

The valid model preferred for dynamic reconstruction of the system is the committee TDNN+Jordan using $\alpha = 0.5$, even though the MLP+Jordan one might be used

[15]We do not develop here the calculation of the invariants for other networks since a) we want to have a model that predicts satisfactorily, no matter whether it models well the original system, and b) we do not know how to compare invariants and decide if a model is valid or not.

Considering the number of time-steps that could be predicted with an acceptable error in closed loop, we see it is mandatory to repeat the training phase every one or two weeks. This week period should be empirically adjusted as long the predictions are produced.

On the other hand, if we use the Mann-Whitney's U-test to both the series generated by the MLP and the original one to check whether they come from the same probability law, we get that:

	Range	Average Rang	Range sum
Var 1	40	67.4	2696.00
Var 2	144	99.47	14324
Total	184		

Contrast statistics

	Var 1
Mann-Whitney's U	1876.000
Wilcoxon's W	2696.000
Z	-3.369
Sig. asymptotically (bi-lateral)	.001

Grouping Variable Var 2

where Var 1 corresponds to the sales values and Var 2 indicated the group where the value is (1 - real series, 2 - synthetic series). A total of 144 randomly chosen consecutive values from the real series were chosen such as they included the corresponding 40 weeks of the generated series.

We can see that, with level $\alpha = 0.05$, the series present different distribution, and this network could be discarded as a non-reliable model for the real system.

Observation: a discarded network from condition 1) of 3.3 on page 158 might be used if that network is considered in a committee. This idea was considered with the TLFN.

3.3.0.12 Valid models for prediction

In the case of prediction, since it is not intended to provide a recursive sales prediction and the period between the different runs of the model (needed for new predictions) is equal a week at least, we can train again every time we decide to have a prediction (if we needed to predict online, with only three seconds between runs, it should be required a model valid in the sense it must predict correctly several steps in the future, since it would not be possible to train between predictions). Then, we only need to design a model for the short-term prediction. Since all the tests were carried out based on one-step prediction in the future, the network used should be trained weekly (see [30]).

If we wanted to use a voting method, it should be done with two or three networks only; otherwise, the training time (including the data processing and handling needed) would be excessive.

If the model selection is done using the averaged percentage of error prediction over the production set in a single step, the best two networks would be the Jordan's and the MLP. As a consequence, the averaged outputs of the Jordan's network with additional inputs and the MLP might be a good sales estimator in a single step even though we are not taking into account the results for the final weeks. We will conclude the same option if we just based on the MDL. To Exploit the properties of error smoothing that voting schemes have, we will also include here the best TDNN.

The estimations that would have been obtained with equal weighting, and using optimal weights suggested by [10] for the weeks 32-45/2011 can be found in 5 on page 175.

3.4 Ensemble Method

This estimation of week sales is constructed using an average of the ones given by different models (topologies). Each one of those models was trained using CV (15% of the training patterns were for CV), so we do not use the smoothing property of the ensemble process. The employed error criterion was the same for all networks. Perrone supports the training might be done with distinct data sets for different topologies, to get sure a high independence among the different estimators. Due to the scarce data available, we could not do this, hence we trained all the models with the same data, which will be translated into an reduction in the error prediction not as meaningful. We performed two works using this method:

1. we predicted the series from weeks 32/2003 to 45/2011 ("open-loop" mode). These values and the respective errors were used to construct the weights (called production weights in Chapter 5) from which the prediction are averaged for weeks 46/2011 a 23/2012).

2. we studied the behavior of the generated values for the ensemble for the closed loop mode. In this case we averaged the obtained values from different models and checked whether the long-term validation requirements hold. In the tables from 5 on page 175 several variants for the ensemble method can be found (averaging for equal weights, weights found using the error variances and the optimal Bishop coefficients for groups of two, three and four networks).

3.4.0.13 Committees used for prediction

We weighted the prediction in accordance with the three previously mentioned ways. We started with open loop predictions and averaged using four, three and two networks. The case with four networks included a TLFN essentially to use the smoothness property of the ensembles, then a test was carried-out with the three networks which presented a prediction error for one step lower than 10%, and finally with the different network pairs with this characteristic.

Committees used for dynamic reconstruction

We took the predictions in a "closed loop" mode for the weeks 32 to 45/2011. By taking averages (the one obtained by means of four networks, the one of TDNN and Jordan network) and comparing them with the values predicted by individual networks we obtained a series whose errors can be seen in Figure 3.15:

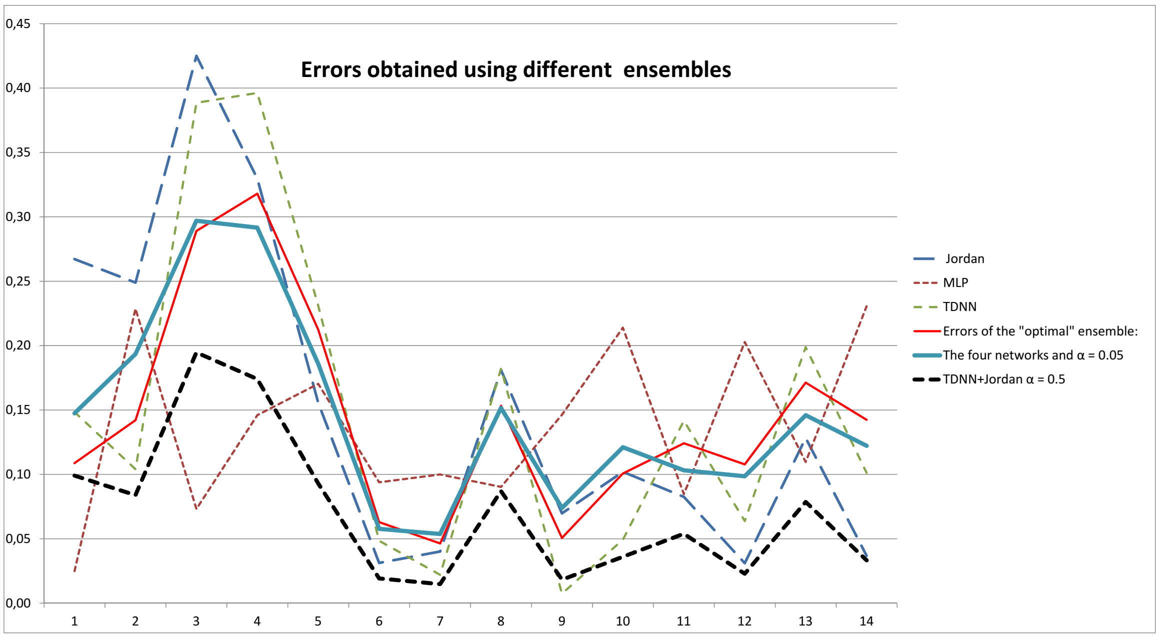

Errors obtained using different ensembles
0,45
0,40
0,35
0,30
0,25
0,20
0,15
0,10
0,05
0,00
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Jordan
MLP
TDNN
Errors of the "optimal" ensemble:
The four networks and α = 0.05
TDNN+Jordan α = 0.5

Figure 3.15

Chapter 4

Concluding remarks and Future work

4.1 Remarks and experimental conclusions

The proposed modeling for this work has used both TLFNs and recurrent networks (see 2.3 on page 50) as preferred topologies chosen beforehand, and the obtained results are compared with multilayer perceptrons (MLP). Additionally, by using networks committees, we intended to get an improvement in the prediction performance. The implementation of recurrent networks and TLFNs is not only justified by the related literature ([30] and [69]) but also as an alternative to be explored with respect to the use of a static MLP trained via back-propagation, a solution given by [2] to a similar problem (prediction of the electrical demand), and that constitutes a paradigmatic network class inside artificial neural networks. We used back-propagation as the training algorithm, in one of its flavors - "back-propagation through time" (BPTT) - essentially because these algorithms are provided by the software tool here chosen for neural networks simulation. Other possible alternatives to MLPs as universal approximators such as the usage of ***radial basis function networks*** (or "RBF networks") and ***support vector machines*** were not further explored in the present work mainly by reasons of extension. The reader can find an overview of these topics in the bibliography ([10], [30], [69] , [73] ,[89], [16]).

Due to the intrinsic weaknesses of gradient descent-based learning algorithms for recurrent networks ([30], [65]), other training rules are visited in the state of the art included here, such as the Kalman filters ([30], [66], [86], [11]) and Genetic Algorithms ([90], [29]), even though further experiments were not developed with them. Genetic algorithms were used to determine the structure of some networks (the number of neurons at both the input and hidden layers).

Once the neural model was constructed, a validation process took place. Several open-loop and closed-loop tests were carried-out [59]. In the open-loop mode, the real values of the time series (signal) is introduced, and the issue is to predict the real value of the series one step ahead in time. In the closed-loop mode, the network operated according to both its own output in each time-step and a copy of its exogenous inputs related with previous steps. In order to predict several steps in the future it is mandatory to feedback the network with its outputs. Such predictions could have great errors provided the output errors are amplified in each step. In this mode we expect than when the exogenous inputs are the same, the network outputs follow the real signal closely, at least during few time-steps, and then they can have considerable deviation[1]. In the open-loop mode, the models were chosen trying to minimize the MSE for each architecture (which is equivalent to minimize the resulting percentage error, see 3.3.0.8 on page 158) and when similar percentage errors were produced, according to the MDL estimated. In the closed-loop mode, the search model was guided regarding both the minimization of the iterated average percentage error and the similarity of the invariants with the ones of the original time series. The best MLP was considered to study the invariants of the obtained time series, as well as the Mann-Whitney's U test was performed to check whether both series (original and synthetic) presented the same probability density function or not.

Some of the most remarkable conclusions from the experimental development are:

1. A valid model for a satisfactory prediction of the time series was found. We proved the existence of the solution to the Input$\rightarrow$ Output learning problem proposed in Section "Entropy and problem resolvability".

2. Provided we found the stochastic realization for sales was non-chaotic (its Lyapunov exponents are non-negative), the greater possible data set will help us to produce more accurate predictions for the mean of the conditional probability for the series to assume a certain outcome given a historical knowledge. However, the stochastic nature of the system makes the probability to produce high prediction errors will always be non-zero [30].

3. Even though we could have trained and predicted by using averages not only from daily temperatures corresponding to days in which sales occurs instead of weekly averages, we considered the best chosen option was the one that best reflects reality. In fact, there is an extra-day in the sum used to find the weekly average temperature (Sunday), that is taken as noise in the temperature measurement (2.8% average), and if we did not include Sundays' temperatures for the weekly average, we would had be discarding real retail sales (from distributors to end consumers) on Sundays (mainly the ones with low temperature), that are then translated into wholesales (from ANCAP to wholesale buyers) on Monday.

4. Additionally to the datey of the price increase, it would be interesting to know whether it was correspondingly announced in the press. This would mean the increase could either have had more or less real impact to the population (end users). In this way, when increases are announced, a speculation effect can appears that does not exist when an increase is not announced. The corresponding types of the increases (announced or not) could not be determined neither from the records in the company industry nor through Internet press, since no newspaper have more than one year's issues online.

5. When we consider networks committees: we mixture the values from several models (sometimes four, others three or two) using the optimal mixture from "Network committees", others using the error variance and finally with equal weighting. In the first case, when the three best networks were used, the average error for the iterated predictions was improved in nearly 50% as well as the variance error. When it was used for the final data set valuable improvements were obtained as well (see 5 on page 175). The error variance is important since it gives an idea of how "reliable" the prediction is.

[1]Passing fro open to closed-loop takes certain amount of time: the ***"priming mode"*** is defined as a mode in which for each one of the p_L consecutive steps, an input is replaced by its respective prediction. The constant p_L is called ***"priming"*** period length. For example, if the network will process two steps back, it will be $p_L = 2$. The closed-loop evaluation cannot be started without starting the network inputs with the corresponding real signal values (not seen during the training stage) and then operate in its priming mode during p_L steps.

6. We confirmed that the addition of noise to both inputs and outputs in the training data improves the obtained results (at least, in the case of a MLP).T
7. In the estimation for the MDL produced using the formula

$$MDL = NLog(MSE) + \frac{1}{2}kLog(N)$$

being:

- k the number of weights
- N the number of CV samples
- MSE over the CV samples

the first term $N\,Log(MSE)$ can have higher weight, depending on the combination of values at hand. It is possibly due to:

- in the networks set there are similar topologies, with similar values for k, or
- dissimilar topologies are implemented with a similar number of weights.

Therefore, since we take into account the number of weights and MSE, this estimation for the MDL does not reflect certain complexities of the model: for instance, a network with gamma memories seems more complex than a MLP, even though both are implemented using the same number of weights

8. The recurrent network model did not face the vanishing gradient problem for one-step predictions. This is in part due to the fact that weekly sales does not impact more than three weeks in the near future. It can be observed from Figure 4.10that the linear tendency (considered as a long-term dependency) is learned by the Jordan network. On the contrary, if daily sales had been predicted, more probably the vanishing gradient problem would had arisen.

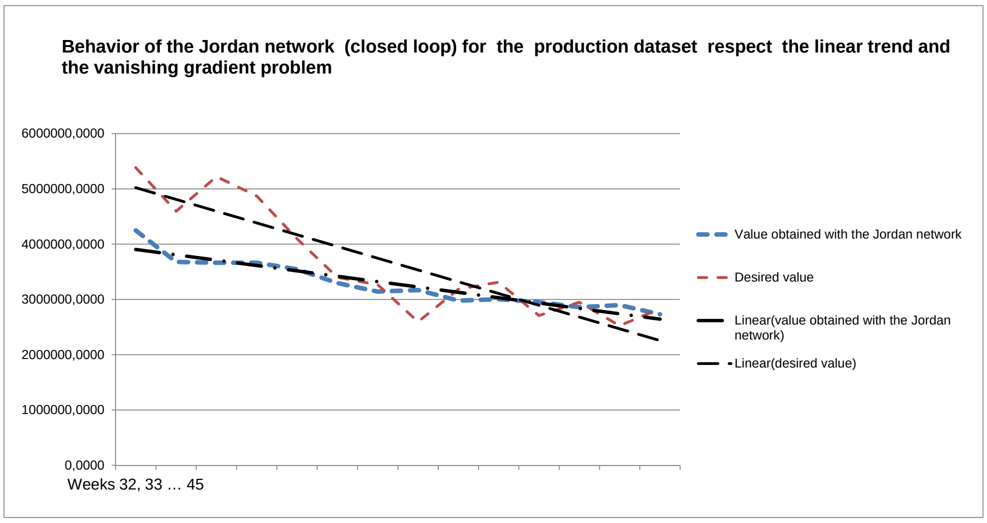


Figure 4.1

9. To summarize the results for the obtained models:

(a) On the possible evaluation criteria:
   - We consider two short-term evaluation criteria: a) MDL. This criterion proved not to be enough to be implemented alone, and b) the error percentage to predict in a step averaged over the whole final data (weeks 46/2011-23/2012).

(b) Analogously, there exist several long-term evaluation criteria:
   i. iterated averaged error over the weeks 32-45/2011 to be lower than 10%.
   ii. similarity between invariants of the synthetic and of the original time series. This similarity cannot constitute a selection criterion *per se*, due to the stochastic nature of the system and the vagueness for the term "similarity". A precise notion for an acceptable quantification of the similarity is out of our scope.
   iii. statistical similarity between the synthetic and original series (probability density functions, variances, etc.).

10. Regarding the topologies, we tried:

    (c) individual networks
    (d) networks committees

    Not all selection criteria are applicable to all topologies. For instance, the MDL is not practicable for an ensemble.

    From the tests here performed it is concluded that an adequate prediction model is the ensemble of the optimal TDNN and the optimal Jordan/Elman network with weights obtained from errors calculated using the productions weeks and $\alpha = 0.5$, with a predictability of two weeks in the near future.

    The re-training period depends on the period of the year where the prediction is performed: if the period is of high-demand (June - July - August) it is convenient to re-train every week, since the ensemble chart with the final data (weeks 46/2011 to 23/2012) shows a successful prediction for only one week. If the period represents a low-demand, even twelve weeks can be predicted advance with no re-training, as can be seen in the ensemble using production data (weeks 32nd. to 45th. correspond to spring/summer, explaining the low demand).

11. The weight decay rule provided by the network simulator was attempted to use in some recurrent networks, but their results were not enlightening: clear evidences for neuron deletions were not offered by the resulting weight matrix after training. Connections were not handled individually (from neuron to neuron), but rather from layer to layer, being the layers totally interconnected each other. Apart from the fact that the weight decay rule did not perform clear changes in the weights matrix, for a tested set of parameters λ (see 2.8 on page 85); if it had it would had meant to construct the network from its most elementary components (individually handled neurons, connections, control elements for back-propagation, etc.) with the need of additional network design and expert tool managing. Since that tool expertise was out of the scope of this work and we had other elements for the optimization of the network structure (for example, genetic algorithms), the weight decay rule was not further tested.

12. Judd's methods as the "$\Psi - \Phi$" might have been tested here to improve long-term predictions. One of the main causes from which these methods were not carried-out is that the primary goal if this work did not required iterated predictions (in order to perform a dynamic reconstruction): when weekly sales should be predicted, all previous sales have perfectly determined, with no need to use the predictions produced by the model.

4.2 General conclusions

In this thesis different methodologies for predictions for time series have been explored by means of neural networks. This implied the study of the state of the art covering different areas: from software analysis of time series and neural network simulators to the theory of dynamic systems.

Here we remark the main results and its corollaries:

1. On the selection of neural network tools and data processing:
    (a) A full framework based on mature software products: DATAPLORE, VRA, STATISTICA, TISEAN, NeuroSolutions and Microsoft Office 2010/Windows 7 was set and is now available as part of this thesis.
    (b) The use of an industrial neural network simulator lead us to an in-depth exploration of several aspects of both neural networks itself (for example, network memory structures) and the theory of dynamical systems.
    (c) In order to work with such dynamical systems, software intended specifically for the analysis and processing of time series was employed, and then chaotic series was part of our focus. Since not all "randomness" was attributable to chaos, in order to characterize the dynamical system generating the time series, an exploration of chaotic-stochastic systems was required, as well as network models to predict a time series associated to one of them. Here we pretended to show how the knowledge of the domain, something extensively treated in the bibliography, can be someway sophisticated (such as the Lyapunov's spectrum for a series or the embedding dimension). It is worth to note that a great deal of the topics on dynamic networks (and dynamic systems) would be simply omitted if another general software tool nor related to such networks to perform simulations. In this way, given that a real problem is faced, the part of the professional practice and technology use is added when dealing with issues such as the state of the art in time series processing techniques and tools for neural networks.
2. On network models (topologies) and the modeling of dynamical systems:
    (a) Regarding the employed network models, the network topologies suggested from the literature as adequate for the prediction task were used (TLFNs and recurrent networks) together with MLPs (a classic of artificial neural networks) and networks committees. The effectiveness of each method could be confirmed for the proposed prediction problem.
    (b) It was proved that the *Input* $\rightarrow$ *Output* relation in our case study can be learned, this is, the problem has solution. Furthermore, we conjecture that the system is non-chaotic but also dynamic-stochastic (since we could only determine the Lyapunov's exponents of a realization of the sales random process), so the prediction quality should improve with time and the corresponding increase of the size of the training set, even though it will always exist a non-zero probability of a big deviation between the real and predicted values, due to the randomness of the original system and others factors that affect consumption not studied here.
    (c) In order to model the dynamical system generated by the time series we used the state-space model, so the time series prediction was translated in the prediction of the next system state. This state-space model, together with the delay method (delayed coordinates) had practical importance for the development of this work,

specifically, the design of the input layer in some networks (MLPs) and other parameters (taps and delays in the TFLNs), and showed something already known: the data handling is essential for the prediction performance. Additionally, the rest of the network components where determined in many cases through procedures traditionally used in neural networks: genetic algorithms. Even though there are several studies to determine network inputs (PCA, feature extraction and others, see [10]), the case is not the same for the rest of the network layers design, more specifically, the hidden layers.

(d) We found that a multilayer perceptron might predict the value of the time series, when the predicted one is just after the training data set (it's adjacent to it). We say it might predict because we should further test this with more training sets and predictions, since here we predicted only once and a good result was produced. In that case, a MLP, even its simple topology, would show to be fit to solve the problem here addressed. This is possible in virtue of the study made of the data set, that helped us to determine meaningful features such as the ones used in Takens-Mañé's theorem.

3. On the numerical results:

   (a) Two tasks were developed: the development of a time series prediction model and the analysis of a feasible model for the dynamic reconstruction of the system. With the best predictive model, obtained by an ensemble of two networks, an average error of 17.56% was obtained when the weeks 46/2012 - 23/2012 were predicted, with a 7.04% for the week 46/2011. This ensemble of a Jordan network and a TDNN used $\alpha = 0.5$ and weights found from the resulting errors in the series prediction for the weeks 32-45/2011. We believe that these results are acceptable provided the quantity of information available, and represent an additional validation that neural networks are useful for time series prediction coming from dynamical systems, no matter whether they are stochastic or not.

   (b) The data gathering and debugging was a long process that finally gave us a set of scarce observations given their dimensionality and, in the case of the increases of price, poor quality. Here it is worth to recall a previously stated in the literature: a major concern before starting a neural network project is to make sure the existence of an adequate data set that supports and makes it feasible. "It is a capital mistake to theorize before having data" (Sherlock Holmes to Watson, in "A scandal in Bohemia", from Arthur Conan Doyle).

4. On the criteria of model selection:

   (a) In order to choose, at least theoretically, from the valid models the more desirable ones, several theoretic model selection criteria were analyzed, and the MDL was initially chosen. We noticed that choosing the model only from the tool estimation of the MDL would not conduct to better results, so we decided first to select the models with valid results ($\%Error < 10\%$) and among them, choose one according to the MDL. Afterward, regarding the MDL estimation provided by the simulator, we determined that it was poor since it did not reflect intrinsic model complexities, but only the weights number and the MSE.

5. On the bibliographical survey:

   (a) The survey of the related literature tried to be extensive, for both printed material and electronic format, in order to have a landscape of the main aspects for the state of the art in time series prediction using neural networks. The material found was sometimes extremely redundant (as in the case of the BP algorithm and its improvements) and scarce in others (memory structures or estimation of the signal subspace dimension in the stochastic case). The surveyed literature includes classical research works ([27], [50], [52]) as well as more recent ones ([79] , [16] or [82]), which pretends to be another contribution of this thesis.

4.3 Future work

There are several possible areas where the present work could be extended or further analyzed:

1. From a practical viewpoint,
   (a) try a daily sales prediction, with the consequent associated research on the necessary descriptive variables, input dimensionality reduction, among many others.
   (b) we performed a visual interpretation of the RP. An alternative choice could be the usage of the quantitative measurements given by RQA.
   (c) generate executable code, so the end-user just loads a small data file and executes an application in order to have a prediction.
   (d) automate the training and model generation (committee), since it should be used every two weeks.
   (e) go on with the experimentation and look for other memory structures and try the training of recurrent neural networks using deKf, since in the daily sales it will probably appear the vanishing gradient problem when using BP.
   (f) determine a confidence interval for the error prediction, also, for d_E and τ and study the sensibility of the results respect these values.
   (g) use neuro-fuzzy systems (the vagueness that models the fuzzy parts would include unknown randomnesses, though vagueness and randomness are distinct concepts).
2. From a theoretical viewpoint,
   (a) analyze how to find the MDL of an ensemble, in order to choose the "best" ensemble. What is more, a related question is the following: given a network with a known MDL, how can we decompose it such that the resulting network committee has better MDL than the original network? Is it possible?
   (b) go on with the study of stochastic dynamical systems, covering more extensively the generalization of the distinct invariant characteristics used to characterize a deterministic system, translated into the stochastic case.

Chapter 5

Numerical results

We detail here the numerical results obtained when the different studied architectures were simulated.

5.1 Results for different networks

5.1.1 Perceptrons

5.1.1.1 Perceptrons based on the sales only

We trained with the available data until the first week of 2003 and tried to predict the next weeks. We obtained:

For a delay = 3 and a single hidden layer, hidden neurons genetically determined in 1500 generations, networks with 5, 8 and 11 hidden neurons in different executions were found, with the values from Tables 1.10to 1.3.

For a delay = 2, 23 neurons were genetically determined in a single hidden layer, obtaining the values from Table 1.4.

Even though the results for a delay = 2 produced the lowest maximum error, the error that is obtained beyond the second week is unacceptable, so we choose the one with 25/2 hidden neurons.

Table 5.1.1: **MLP d=3, IN=[V, V-3],OUT=[V+1], 5 hidden layers**

Desired	**Real**	**%Error**
538559	2622025.05	51.31%
4595666	2829096.39	38.44%
5222869	2435064.98	53.38%
4870710	2538441.79	47.88%
4096739	2721694.49	33.56%
3398617	2433936.24	28.38%
3266710	2425071.57	25.76%
2595111	2557350.61	1.46%
3185410	2642169.38	17.05%
3311189	2845542.79	14.06%
2706442	3299803.35	21.92%
2950426	2747326.01	6.88%
2524455	2757714.93	9.24%
2831359	2923698.8	3.16%
	Min.	1.46%
	Max.	53.38%
	Var	0.029190315
	Mean	25.18%

Table 5.1.2: **MLP d= 3, IN=[V, V-3],OUT=[V+1], 8 hidden neurons**

Desired	**Real**	**%Error**
5385591	2545322.78	52.74%
4595666	2751808.46	40.12%
5222869	2324529.38	55.49%
4870710	2456917.18	49.56%
4096739	2662072.71	35.02%
3398617	2318370.11	31.78%
3266710	2300191.92	29.59%
2595111	2510092.96	3.28%
3185410	2668133.35	16.24%
3311189	2901146.88	12.38%
2706442	3381922.10	24.96%
2950426	2811186.04	4.72%
2524455	2807675.50	11.22%
2923698.8	3048061.87	4.25%
	Min.	3.28%
	Max.	55.49%
	Var.	0.031466608
	Mean	26.53%

Table 5.1.3: **MLP d=3, IN=[V, V-3],OUT=[V+1], 11 hidden neurons**

Desired	**Real**	**%Error**
5385591	2544790.79	52.75%
4595666	2783529	39.43%
5222869	2299002.45	55.98%
4870710	2445947.8	49.78%
4096739	2694405.55	34.23%
3398617	2307474.20	32.11%
3266710	2303710.50	29.48%
2595111	2549924.75	1.74%
3185410	2713362.69	14.82%
3311189	2938233.48	11.26%
2706442	3383395.76	25.01%
2950426	2850735.34	3.38%
2524455	2848847.77	12.85%
2923698.8	3068787.44	4.96%
	Min.	1.74%
	Max.	55.98%
	Var.	0.032395909
	Mean	26.27%

Table 5.1.4: **MLP d=2, IN=[V, V-2],OUT=[V+1], 23 hidden neurons**

Desired	**Real**	**%Error**
5385591	3380695.60	37.23%
4595666	3412203.30	25.75%
5222869	3356184.8	35.74%
4870710	3353750	31.14%
4096739	3365627.54	17.85%
3398617	3287489.48	3.27%
3266710	3166704.47	3.06%
2595111	3092196.26	19.15%
3185410	3023961.92	5.07%
3311189	3015429.45	8.93%
2706442	3076840.79	13.69%
2950426	3037039.66	2.94%
2524455	3005896.13	19.07%
2923698.8	2991817.88	2.33%
	Min.	2.33%
	Max.	37.23%
	Var.	0.014636008
	Mean	16.09%

Table 5.1.5: **MLP d=2, IN=[V, V-2],OUT=[V+1], 11 hidden neurons**

Desired	**Real**	**%Error**
5385591	2987823.86	44.52%
4595666	2810752.65	38.84%
5222869	2765960.68	47.04%
4870710	2972951.8	38.96%
4096739	2816072.1	31.26%
3398617	2807653.2	17.39%
3266710	2878188.58	11.89%
2595111	3001438.62	15.66%
3185410	2951389.22	7.35%
3311189	3164537.1	4.43%
2706442	3051822.55	12.76%
2950426	2955065.72	0.16%
2524455	3111184.4	23.24%
2923698.8	3007307.88	2.86%
	Min.	0.16%
	Max.	47.04%
	Var.	0.024276261
	Mean	21.17%

Table 5.1.6: **MLP d=2, IN=[V, V-2],OUT=[V+1], 25/2 hidden neurons**

Desired	**Real**	**%Error**
5.385.591	5.369.625	0.30%
4.595.666	6.003.580	30.64%
5.222.869	4.162.727	20.30%
4.870.710	5.214.485	7.06%
4.096.739	5.048.485	23.23%
3.398.617	3.584.121	5.46%
3.266.710	3.189.445	2.37%
2.595.111	3.046.616	17.40%
3.185.410	2.325.407	27.00%
3.311.189	2.844.855	14.08%
2.706.442	2.799.341	3.43%
2.950.426	2.381.773	19.27%
2.524.455	2.621.688	3.85%
2.831.359	2.240.779	20.86%
	Max.	30.64%
	Min.	0.30%
	Var.	0.009399434
	Mean	13.95%

Table 5.1.7: **MLP d=2 hidden layer genetically determined, IN=[V, V-2],OUT=[V+1], 24/19 hidden neurons**

Desired	Real	%Error
5385591	5432890.71	0.88%
4595666	5223643.12	13.66%
5222869	4201601.55	19.55%
4870710	5205019.98	6.86%
4096739	4620936.68	12.80%
3398617	3732458.68	9.82%
3266710	3261112.75	0.17%
2595111	3129181.19	20.58%
3185410	2632957.59	17.34%
3311189	2959952.06	10.61%
2706442	3138001.61	15.95%
2950426	2723236.46	7.70%
2524455	2789542.75	10.50%
2923698.8	2516186.05	13.94%
	Min.	0.17%
	Max.	20.58%
	Var.	0.00353696
	Mean	11.45%

5.1.1.2 Perceptrons with additional inputs

Table 5.1.8: **MLP IN=[S,V,V-1,V-2, Aver T max, Aver T min, Aver precip, Aver wind and Aver heliophany] OUT=[V] and 24 h. n.**

Desired	Real	%Error
5385591	3850550.78	28.50%
4595666	2453265.29	46.62%
5222869	2412499.12	53.81%
4870710	3850550.78	20.94%
4096739	3850550.78	6.01%
3398617	3911953.74	15.10%
3266710	1952965.27	40.22%
2595111	1952965.27	24.74%
3185410	2412499.12	24.26%
3311189	3911953.74	18.14%
2706442	3850550.78	42.27%
2950426	2453265.29	16.85%
2524455	1952965.27	22.64%
2923698.8	3850550.78	31.70%
	Min.	6.01%
	Max.	53.81%
	Var.	0.016817568
	Mean	27.99%

Table 5.1.9: **MLP IN=[S,V,V-1,V-2, Aver T max, Aver T min, Aver precip, Aver wind and Aver heliophany] OUT=[V] and 24 h. n. with averaged outliers**

—Real—	—Desired—	%Error	%Error c/series 'raw'
6396654.06	5385591	18.77%	28.50%
6396654.04	5304230	20.59%	46.62%
6396654.02	5222869	22.47%	53.81%
6396654.06	4870710	31.33%	20.94%
2454284.37	4096739	40.09%	6.01%
2431019.43	3398617	28.47%	15.10%
1391519.4	3266710	57.40%	40.22%
1210138.49	2595111	53.37%	24.74%
1441023.32	3185410	54.76%	24.26%
2515808.8	3311189	24.02%	18.14%
2538243.27	2706442	6.21%	42.27%
1660754.31	2950426	43.71%	16.85%
2104717.09	2524455	16.63%	22.64%
1940898.01	2831359	31.45%	31.70%
	Max.	57.40%	53.81%
	Min.	6.21%	6.01%
	Var.	0.02283291	0.01811123
	Mean	32.09%	27.99%

Table 5.1.10: **MLP, 25/2 h. n. with averaged outliers**

—Real—	—Desired—	%Error	%Error w/'raw' series
5010062.38	5385591	6.97%	0.30%
5009859.72	4595666	9.01%	30.64%
4964788.64	5222869	4.94%	20.30%
5009405.86	4870710	2.85%	7.06%
4957009.09	4096739	21.00%	23.23%
3266391.53	3398617	3.89%	5.46%
3104308.41	3266710	4.97%	2.37%
3094606.41	2595111	19.25%	17.40%
2698098.9	3185410	15.30%	27.00%
3078110.04	3311189	7.04%	14.08%
3073879.12	2706442	13.58%	3.43%
2789227.49	2950426	5.46%	19.27%
3004107.31	2524455	19.00%	3.85%
2475868.68	2831359	12.56%	20.86%
	Max.	21.00%	30.64%
	Min.	2.85%	0.30%
	Var.	0.00393321	0.00101
	Mean	10.42%	13.95%

Table 5.1.11: **MLP IN=[S,V,V-1,V-2, Aver T max, Aver T max ant, Aver T min, Aver precip], OUT=[V], 4/5 h. n.**

Desired	**Real**	**%Error**
5385591	3546135.16	34.16%
4595666	4684982.27	1.94%
5222869	4390901.55	15.93%
4870710	4362510.24	10.43%
4096739	2812792.45	31.34%
3398617	2319103.05	31.76%
3266710	2583629.15	20.91%
2595111	2257849.76	13.00%
3185410	2297732.07	27.87%
3311189	2375374.35	28.26%
2706442	2190424.87	19.07%
2950426	2237717.05	24.16%
2524455	2151973.65	14.75%
2923698.8	2131353.16	27.10%
	Min.	1.94%
	Max.	34.16%
	Var.	0.008195003
	Mean	21.48%

Table 5.1.12: **MLP IN=[S,V,V-1,V-2, Aver T max, Aver T max ant, Aver T min, Aver precip], OUT=[V], 4/5 h. n. and averaged outliers**

—Real—	**—Desired—**	**%Error**	**%Error w/the 'raw' series**
6396654.06	4322848.16	47.97%	34.16%
6396654.06	4704700.49	35.96%	1.94%
6396654.02	4615842.33	38.58%	15.93%
6396654.06	4533067.62	41.11%	10.43%
2454284.37	2760189.93	11.08%	31.34%
2431019.43	2352757.43	3.33%	31.76%
1391519.4	2733201.01	49.09%	20.91%
1210138.49	2328879.92	48.04%	13.00%
1441023.32	2303584.75	37.44%	27.87%
2515808.8	2408480.43	4.46%	28.26%
2538243.27	2210639.97	14.82%	19.07%
1660754.31	2269354.32	26.82%	24.16%
2104717.09	2253164.99	6.59%	14.75%
1940898.01	2147712.62	9.63%	27.10%
	Max.	49.09%	34.16%
	Min.	3.33%	1.94%
	Var.	0.03139309	0.00882539
	Mean	26.78%	21.48%

5.1.1.3 Perceptrons with noise in their inputs

In order to improve the prediction performance, we tried to train many MLPs adding white Gaussian noise with zero mean and different variances (here denoted with V) to their inputs $\{v_i, v_{i-2}\}$. We worked with a MLP of 25/2 hidden neurons trying to improve the obtained values . The desired training values had the same noise. The production values had no noise.

The following results were obtained:

Table 5.1.13: **MLP with noise in the inputs**

Real	**V=0.1**	**%Error**	**V=0.7**	**%Error**	**V=5**	**%Error**
5385591	5270791.79	2.13%	4999080.45	7.18%	5306477.74	1.47%
4595666	5097101.01	10.91%	4989490.61	8.57%	5063299.83	10.18%
5222869	3947577.74	24.42%	4933516.50	5.54%	3967021.46	24.05%
4870710	5086722.23	4.43%	4987691.01	2.40%	5068747.25	4.07%
4096739	4476781.61	9.28%	4968133.28	21.27%	4440368.44	8.39%
3398617	3430242.97	0.93%	3412186.28	0.40%	3411060.54	0.37%
3266710	3137937.44	3.94%	3159124.13	3.29%	3143618.15	3.77%
2595111	3017414.09	16.27%	3160763.77	21.80%	3048998.66	17.49%
3185410	2554761.74	19.80%	2738723.52	14.02%	2592861.77	18.60%
3311189	2880651.03	13.00%	3102371.07	6.31%	2924746.23	11.67%
2706442	3025008.53	11.77%	3175407.93	17.33%	3046206.61	12.55%
2950426	2635176.42	10.68%	2903701.38	1.58%	2668106.11	9.57%
2524455	2733256.9	8.27%	2935130.86	16.27%	2759503.52	9.31%
2831359	2473645.65	12.63%	2541098.59	10.25%	2506370.55	11.48%
	Max.	24.42%	**Max.**	21.80%	**Max.**	24.05%
	Min.	0.93%	**Min.**	0.40%	**Min.**	0.37%
	Mean	10.61%	**Mean**	9.73%	**Mean**	10.21%
	Var.	0.0044	**Var.**	0.0053	**Var.**	0.0045

If these values are compared with the ones obtained with the series without noise (Table 5.1.6), it can be observed a remarkable improvement.

The noise at the input improves the number of time-steps to be predicted with an error lower than 10% (four steps), reduces the averaged mean error (in less than 20%) as well as the maximum error, although the error is increased in the first step nearly in 100%. Anyway, for the other experiments, we still prefer the a MLP with 25/2 h. n. trained without noise.

Table 5.1.14: **MLP with noise in the inputs**

Real	**V=10**	**%Error**	**V=15**	**%Error**	**V=25**	**%Error**	**V=50**	**%Error**
5385591	4971361.77	7.69%	4587776.57	14.81%	5300530.07	1.58%	5015379.92	6.87%
4595666	4962473.89	7.98%	4530527.83	1.42%	5129171.41	11.61%	5016193.09	9.15%
5222869	4888843.55	6.40%	4026728.43	22.90%	4069921.03	22.07%	4929103.36	5.62%
4870710	4955746.41	1.75%	4469369.72	8.24%	5115149.85	5.02%	4963044.58	1.90%
4096739	4936530.99	20.50%	4257730.15	3.93%	4582538.70	11.86%	4963750.32	21.16%
3398617	3404526.25	0.17%	3554383.88	4.58%	3531621.27	3.91%	3494764.84	2.83%
3266710	3119253.35	4.51%	2871880.15	12.09%	3160783.50	3.24%	3230581.45	1.11%
2595111	3118858.83	20.18%	2771253.61	6.79%	3024744.76	16.56%	3206853.18	23.57%
3185410	2707867.70	14.99%	2400491.70	24.64%	2540721.24	20.24%	2704436.06	15.10%
3311189	3054923.63	7.74%	2684863.85	18.92%	2842373.74	14.16%	3066556.83	7.39%
2706442	3142066.86	16.10%	2806102.19	3.68%	3009465.34	11.20%	3200740.70	18.26%
2950426	2871670.60	2.67%	2451025.85	16.93%	2620236.06	11.19%	2895743.39	1.85%
2524455	2900147.17	14.88%	2545099.35	0.82%	2692732.98	6.67%	2931820.22	16.14%
2831359	2496854.07	11.81%	2375881.26	16.09%	2465179.22	12.93%	2511263.34	11.31%
	Max.	20.50%	**Max.**	24.64%	**Max.**	22.07%	**Max.**	23.57%
	Min.	0.17%	**Min.**	0.82%	**Min.**	1.58%	**Min.**	1.11%
	Mean	9.81%	**Mean**	11.13%	**Mean**	10.87%	**Mean**	10.16%
	Var.	0.0045	**Var.**	0.0064	**Var.**	0.0039	**Var.**	0.0057

5.1.2 Time lagged feed-forward networks (TLFNs)

5.1.2.1 TDNNs

Table 5.1.15: **TDNN IN=[V], non-focused, d=3/d=2 and 16 h. n.**

	d = 3		**d = 2**	
Real	**Desired**	**%Error**	**Real**	**%Error**
3799405.75	5385591	29.45%	5.148.473	4.40%
4079347.10	4595666	11.23%	4.670.389	1.63%
4127097.87	5222869	20.98%	4.380.066	16.14%
3646733.10	4870710	25.13%	4.549.058	6.60%
3208287.79	4096739	21.69%	3.786.131	7.58%
3770845.24	3398617	10.95%	3.389.907	0.26%
3182323.74	3266710	2.58%	3.107.519	4.87%
2311872.85	2595111	10.91%	2.839.890	9.43%
2260900.61	3185410	29.02%	2.472.319	22.39%
2056515.98	3311189	37.89%	2.366.377	28.53%
2014541.25	2706442	25.56%	2.460.661	9.08%
2140712.10	2950426	27.44%	2.248.732	23.78%
1767209.15	2524455	30.00%	2.226.246	11.81%
1675857.14	2831359	40.81%	2.099.799	25.84%
	Mean	23.12%	**Mean**	23.12%
	Max.	40.81%	**Max.**	28.53%
	Min.	2.58%	**Min.**	0.26%
	Var.	0.01191569	**Var.**	0.00880866

Table 5.1.16: **TDNN, d = 2, IN=[S,A,T-1], depth = 3, focused, 8 n.o.**

Real	**Desired**	**%Error**
4067209.27	5385591	24.48%
4329889.74	4595666	5.78%
3885165.75	5222869	25.61%
3967233.83	4870710	18.55%
4233890.78	4096739	3.35%
3375568.49	3398617	0.68%
3445484.53	3266710	5.47%
2702415.87	2595111	4.13%
2661346.25	3185410	16.45%
2856301.34	3311189	13.74%
2601172.70	2706442	3.89%
3671938.79	2950426	24.45%
2440305.53	2524455	3.33%
2261918.02	2831359	20.11%
	Mean	12.15%
	Min.	0.68%
	Max.	25.61%
	Var.	0.0086%

Table 5.1.17: **TDNN IN=[V], d = 2, 8 n.o., non-focused, depths 10 and 3**

Real depth = 10	**Desired**	**%Error**	**Real depth = 3**	**%Error**
2706919.00	5385591	49.74%	3070786.56	42.98%
2548499.00	4595666	44.55%	3094480.78	32.67%
2805003.00	5222869	46.29%	3751714.91	28.17%
3102691.86	4870710	36.30%	3454092.88	29.08%
3256292.40	4096739	20.52%	3355358.31	18.10%
3048952.56	3398617	10.29%	2872302.55	15.49%
2935446.85	3266710	10.14%	2585331.29	20.86%
2693865.33	2595111	3.81%	2418317.13	6.81%
2544763.74	3185410	20.11%	2272311.82	28.67%
2390895.63	3311189	27.79%	2380599.72	28.10%
2126422.76	2706442	21.43%	2304077.48	14.87%
2054803.34	2950426	30.36%	2262917.70	23.30%
2089660.87	2524455	17.22%	2245843.56	11.04%
1999752.89	2831359	29.37%	2091471.88	26.13%
	Max.	49.74%	**Max.**	42.98%
	Min.	3.81%	**Min.**	6.81%
	Var.	0.018652663	**Var.**	0.008424231
	Mean	10.42%	**Mean**	23.31%

Table 5.1.18: **TDNN IN=[V], non-focused, d=2, 8 h. n., mem. depth =10**

Desired	**%Error 'Raw'**	**Real**	**%Error**
5385591	49.74%	4896442.58	9.99%
4595666	44.55%	4525929.81	1.54%
5222869	46.29%	4311756.92	21.13%
4870710	36.30%	4533113.94	7.45%
4096739	20.52%	4522433.60	9.41%
3398617	10.29%	3926118.09	13.44%
3266710	10.14%	3031274.12	7.77%
2595111	3.81%	3230834.15	19.68%
3185410	20.11%	2881314.32	10.55%
3311189	27.79%	2804121.71	18.08%
2706442	21.43%	3269879.35	17.23%
2950426	30.36%	2819818.96	4.63%
2524455	17.22%	2715954.95	7.05%
2831359	29.37%	2566478.18	10.32%
Mean	26.28%		11.30%
Max.	49.74%		21.13%
Min.	3.81%		1.54%
Var.	0.02008748		0.003174

5.1.2.2 TLFNs

Table 5.1.19: **TLFN IN=[V, V-1],OUT=[V+1], 15 h. n., mem. gamma focused, depth = 15**

–Real–	–Desired–	–%Error–
4212190.66	5385591	21.79%
4368849.18	4595666	4.94%
4427442.41	5222869	15.23%
4644067.36	4870710	4.65%
3368218.01	4096739	17.78%
3056515.42	3398617	10.07%
2421923.85	3266710	25.86%
2481071.85	2595111	4.39%
2512210.49	3185410	21.13%
3413040.8	3311189	3.08%
3132387.38	2706442	15.74%
3129872.42	2950426	6.08%
2587303.76	2524455	2.49%
1992388.59	2831359	29.63%
	Mean	13.06%
	Min.	2.49%
	Max.	29.63%
	Var.	0.0084

Table 5.1.20: **TLFN IN=[V,V-1],OUT=[V+1], focused, depth = 10, 4 h. n.**

–Real–	–Desired–	–%Error–
4265433.83	5385591	20.80%
3905378.67	4595666	15.02%
3725727.9	5222869	28.67%
4294161.11	4870710	11.84%
3686130.51	4096739	10.02%
3135993.41	3398617	7.73%
2753895.48	3266710	15.70%
2775329.82	2595111	6.94%
2079445.04	3185410	34.72%
1919878.44	3311189	42.02%
2108143.61	2706442	22.11%
1863600.12	2950426	36.84%
1732740.41	2524455	31.36%
1797692.37	2831359	36.51%
	Mean	22.88%
	Min.	6.94%
	Max.	42.02%
	Var.	0.0145

Table 5.1.21: **TLFN with IN=[V,V-1] OUT=[V+1], focused, 8 h. n., depth = 10**

–Real–	–Desired–	%Error
4212190.66	5385591	21.79%
4368849.18	4595666	4.94%
4427442.41	5222869	15.23%
4644067.36	4870710	4.65%
3368218.01	4096739	17.78%
3056515.42	3398617	10.07%
2421923.85	3266710	25.86%
2481071.85	2595111	4.39%
2512210.49	3185410	21.13%
3413040.8	3311189	3.08%
3132387.38	2706442	15.74%
3129872.42	2950426	6.08%
2587303.76	2524455	2.49%
1992388.59	2831359	29.63%
	Mean	13.06%
	Max.	29.63%
	Min.	2.49%
	Var.	0.008421

Table 5.1.22: **TLFN with IN=[V,V-1], OUT=[V+1], focused, 4 h. n., depth = 10**

–Real–	–Desired–	%Error
4265433.83	5385591	20.80%
3905378.67	4595666	15.02%
3725727.9	5222869	28.67%
4294161.11	4870710	11.84%
3686130.51	4096739	10.02%
3135993.41	3398617	7.73%
2753895.48	3266710	15.70%
2775329.82	2595111	6.94%
2079445.04	3185410	34.72%
1919878.44	3311189	42.02%
2108143.61	2706442	22.11%
1863600.12	2950426	36.84%
1732740.41	2524455	31.36%
1797692.37	2831359	36.51%
	Min.	6.94%
	Max.	42.02%
	Var.	0.01344097
	Mean	22.88%

Table 5.1.23: **TLFN d=2, focused, IN=[S,T,V-1,V-2,T-1, Aver. rains, Aver. winds], OUT=[V], "delay" =1 taps =3**

–Real–	–Desired–	%Error
4067209.27	5385591	24.48%
4329889.74	4595666	5.78%
3885165.75	5222869	25.61%
3967233.83	4870710	18.55%
4233890.78	4096739	3.35%
3375568.49	3398617	0.68%
3445484.53	3266710	5.47%
2702415.87	2595111	4.13%
2661346.25	3185410	16.45%
2856301.34	3311189	13.74%
2601172.70	2706442	3.89%
3671938.79	2950426	24.45%
2440305.53	2524455	3.33%
2261918.02	2831359	20.11%
	Mean	12.15%
	Max.	25.61%
	Min.	0.68%
	Var.	0.00792079

Table 5.1.24: **TLFN IN=[T,T-1,Aver. rains, Aver. winds], Laguerre mem.**

–Real–	–Desired–	%Error
3439409.99	5385591	36.14%
3597115.19	4595666	21.73%
4304683.09	5222869	17.58%
4005499.8	4870710	17.76%
3185782.49	4096739	22.24%
2770620.89	3398617	18.48%
2976354.91	3266710	8.89%
2724329.09	2595111	4.98%
2731410.85	3185410	14.25%
2792854.9	3311189	15.65%
2620308.39	2706442	3.18%
2713044.37	2950426	8.05%
2492539.72	2524455	1.26%
2274217.62	2831359	19.68%
	Mean	14.99%
	Max.	36.14%
	Min.	1.26%
	Var	0.007917255

Table 5.1.25: **TLFN IN=[V,V-2] OUT=[V+1], non-focused, d=2, traj. = 4 depth = 15, 15 h. n., Laguerre mem**

–Real–	–Desired–	–%Error–
3431501.21	5385591	36.28%
3459194.07	4595666	24.73%
4245208.95	5222869	18.72%
4150950.48	4870710	14.78%
4465854.03	4096739	9.01%
3182226.87	3398617	6.37%
3100878.97	3266710	5.08%
2906852.50	2595111	12.01%
2757917.49	3185410	13.42%
2814461.84	3311189	15.00%
2340551.89	2706442	13.52%
2529519.93	2950426	14.27%
2406396.66	2524455	4.68%
1875004.36	2831359	33.78%
	Mean	12.15%
	Max.	36.28%
	Min.	4.68%
	Var.	0.0065295

Table 5.1.26: **TLFN genetically determined inputs, IN=[T,T-1,Aver week precip, Aver week wind],gamma mem depth = 10, traj. = 4, focused, 18 h. n.)**

–Real–	–Desired–	%Error
4629682.73	5385591	14.04%
4424991.9	4595666	3.71%
4587028.99	5222869	12.17%
4425043.17	4870710	9.15%
3940091.10	4096739	3.82%
3557357.99	3398617	4.67%
3383543.67	3266710	3.58%
2936739.74	2595111	13.16%
3044483	3185410	4.42%
2983662.60	3311189	9.89%
2645666.60	2706442	2.25%
2801464.65	2950426	5.05%
2596599.74	2524455	2.86%
1827594.38	2831359	35.45%
	Mean	12.15%
	Max.	35.45%
	Min.	2.25%
	Var.	0.00789147

Table 5.1.27: **TLFN d= 2, non-focused, IN=[V].16 h. n., Laguerre mem. traj. = 4 depth = 15**

Real	Desired	%Error
4631178.46	5385591	14.01%
4291083.30	4595666	6.63%
3845678.46	5222869	26.37%
4147868.58	4870710	14.84%
3994048.38	4096739	2.51%
3319743.79	3398617	2.32%
3039956.49	3266710	6.94%
2559644.35	2595111	1.37%
2569840.56	3185410	19.32%
2372675.58	3311189	28.34%
2151710.25	2706442	20.50%
2216310.25	2950426	24.88%
2125683.84	2524455	15.80%
1943785.19	2831359	31.35%
	Mean	15.37%
	Max.	31.35%
	Min.	1.37%
	Var.	0.009702791

5.1.3 Recurrent networks

5.1.3.1 Totally and partially recurrent networks

Table 5.1.28: **Totally recurrent, IN=[S,T,V-2] and OUT=[V], 9 h. n.**

Real	**Desired**	**%Error**
3508947.36	5385591	34.85%
3470358.03	4595666	24.49%
3487748.81	5222869	33.22%
3384714.39	4870710	30.51%
3392335.27	4096739	17.19%
2997767.22	3398617	11.79%
2990242.8	3266710	8.46%
2751853.39	2595111	6.04%
2743128.86	3185410	13.88%
2724732.24	3311189	17.71%
2849719.36	2706442	5.29%
2697677.25	2950426	8.57%
2677456.20	2524455	6.06%
2528242.87	2831359	10.71%
	Mean	16.34%
	Max.	34.85%
	Min.	5.29%
	Var.	0.010086

Table 5.1.29: **Partially recurrent, IN=[S,T,V-2], OUT=[V], 9 h. n.**

Real	**Desired**	**%Error**
4035700.46	5385591	25.06%
4375097.04	4595666	4.80%
4039548.36	5222869	22.66%
4013737.28	4870710	17.59%
3680771.19	4096739	10.15%
3380942.89	3398617	0.52%
3094850.04	3266710	5.26%
2734810.43	2595111	5.38%
2666281.51	3185410	16.30%
2718473.41	3311189	17.90%
2685091.23	2706442	0.79%
2665050.21	2950426	9.67%
2454387.21	2524455	2.78%
2269644.54	2831359	19.84%
	Mean	11.34%
	Max.	25.06%
	Min.	0.52%
	Var.	0.00594664

Table 5.1.30: **Partially recurrent IN=[S,T,V-1], OUT=[V], 4/2 h. n.**

Real	**Desired**	**%Error**
4266846.79	5385591	20.77%
4530596.67	4595666	1.42%
4321543.9	5222869	17.26%
4157797.11	4870710	14.64%
4152276.9	4096739	1.36%
3684612.18	3398617	8.42%
3437375.73	3266710	5.22%
3173625.91	2595111	22.29%
3040158.82	3185410	4.56%
2776118.24	3311189	16.16%
2824970.75	2706442	4.38%
2685730.33	2950426	8.97%
2331847.49	2524455	7.63%
2348906.30	2831359	17.04%
	Mean	10.72%
	Max.	22.29%
	Min.	1.36%
	Var.	0.00461852

Table 5.1.31: **Totally recurrent IN=[S,T,V-1], OUT=[V], 4/2 h. n.**

Real	**Desired**	**%Error**
3988787.52	5385591	25.94%
4121912.28	4595666	10.31%
4424381.50	5222869	15.29%
4100939.44	4870710	15.80%
3815312.52	4096739	6.87%
3557906.21	3398617	4.69%
3675816.47	3266710	12.52%
3246851.06	2595111	25.11%
2852422.13	3185410	10.45%
2949399.56	3311189	10.93%
2623949.91	2706442	3.05%
2478495.35	2950426	16.00%
2233029.83	2524455	11.54%
2120387.76	2831359	25.11%
	Mean	13.82%
	Max.	25.94%
	Min.	3.05%
	Var.	0.00451061

Table 5.1.32: **Totally recurrent IN=[S,T,V-1,V-2], OUT=[V], 4/2 n.o.**

Real	**Desired**	**%Error**
3591264.69	5385591	33.32%
4394579.61	4595666	4.38%
4271784.91	5222869	18.21%
3811672.20	4870710	21.74%
3615269.10	4096739	11.75%
3576227.99	3398617	5.23%
3054037.8	3266710	6.51%
2748575.65	2595111	5.91%
2629334.46	3185410	17.46%
2506132.36	3311189	24.31%
2688247.87	2706442	0.67%
2711887.66	2950426	8.08%
2467136.88	2524455	2.27%
2371997.03	2831359	16.22%
	Mean	12.58%
	Max.	33.32%
	Min.	0.67%
	Var.	0.00607216

Table 5.1.33: **Partially recurrent IN=[S,T,V-1,V-2], OUT=[V], 4/2 n.o.**

Real	**Desired**	**%Error**
3417188.18	5385591	36.55%
4254813.62	4595666	7.42%
4275128.20	5222869	18.15%
3809178.4	4870710	21.79%
3324703.76	4096739	18.85%
3389733.49	3398617	0.26%
3225629.02	3266710	1.26%
2632569.12	2595111	1.44%
2813228.8	3185410	11.68%
2551772.44	3311189	22.93%
2474051.53	2706442	8.59%
2804165.51	2950426	4.96%
2242462.66	2524455	11.17%
2184932.36	2831359	22.83%
	Mean	13.42%
	Max.	36.55%
	Min.	0.26%
	Var.	0.010281

5.1.3.2 Jordan and Elman networks

Table 5.1.34: **Jordan Network IN=[V,S,S-1,V-1,T-1], OUT=[V+1], 4/2 h. n. and t =0.5**

Real	Desired	%Error
4222608.35	5385591	21.59%
3489409.44	4595666	24.07%
3784060.46	5222869	27.55%
3461227.50	4870710	28.94%
2908412.36	4096739	29.01%
2634581.84	3398617	22.48%
2594637.11	3266710	20.57%
2394885.24	2595111	7.72%
2548000.39	3185410	20.01%
2458880.77	3311189	25.74%
2319273.62	2706442	14.31%
2382911.65	2950426	19.23%
2253661.23	2524455	10.73%
2009807.93	2831359	29.02%
	Mean	21.50%
	Max.	29.02%
	Min.	7.72%
	Var.	0.004586714

Table 5.1.35: **Jordan Network IN=[V,S,S-1,V-1,T-1], OUT=[V+1], 4/2 hidden neurons and t = 0.7**

Real	Desired	%Error
4279007.42	5385591	20.55%
3470667.71	4595666	24.48%
4049518.61	5222869	22.47%
3867396	4870710	20.60%
3257518.05	4096739	20.49%
2711083.14	3398617	20.23%
2588234.18	3266710	20.77%
2470570.83	2595111	4.80%
2447492.04	3185410	23.17%
2511474.29	3311189	24.15%
2337763.18	2706442	13.62%
2321220.04	2950426	21.33%
2278765.99	2524455	9.73%
2049861.93	2831359	27.60%
	Mean	19.57%
	Max.	27.60%
	Min.	4.80%
	Var.	0.003762748

Table 5.1.36: **Jordan Network IN=[V,S,S-1,V-1,T-1], OUT=[V+1], 4/2 hidden neurons and t=0.15**

Real	Desired	%Error
4909197.61	5385591	8.85%
4671036.35	4595666	1.64%
4756557.78	5222869	8.93%
4650881.91	4870710	4.51%
4193070.91	4096739	2.35%
3184838.61	3398617	6.29%
3338036.9	3266710	2.18%
3134288.19	2595111	20.78%
2790258.46	3185410	12.41%
2615972.81	3311189	21.00%
2160622.10	2706442	20.17%
2088978.97	2950426	29.20%
2084272.8	2524455	17.44%
2079475.60	2831359	26.56%
	Mean	13.02%
	Max.	29.20%
	Min.	1.64%
	Var.	0.008867245

Table 5.1.37: **Jordan Network IN=[V,S,S-1,V-1,T-1], OUT=[V+1], 4/2 hidden neurons and t=0.15 with averaged outliers**

Real	Desired	%Error	%Error with raw
3816204.17	3350217.72	13.91%	8.85%
3430731.45	3438128.62	0.22%	1.64%
4551299.84	4007572.32	13.57%	8.93%
3964113.01	3641926.99	8.85%	4.51%
3017542.43	2635434.41	14.50%	2.35%
2821100.80	2281052.17	23.68%	6.29%
2761331.36	2392399.05	15.42%	2.18%
2350775.01	2217238.94	6.02%	20.78%
2335365.26	2294538.47	1.78%	12.41%
2281026.77	2275009.71	0.26%	21.00%
2240502.47	2190907.00	2.26%	20.17%
2297207.19	2235658.90	2.75%	29.20%
2160023.31	2161358.57	0.06%	17.44%
2111803.43	1941485.53	8.77%	26.56%
	Max.	23.68%	29.20%
	Min.	0.06%	1.64%
	Var.	0.00535965	0.00886724
	Mean	8.00%	13.02%

Table 5.1.38: **Jordan Network IN=[S,A,A-1,V-1,T-1], OUT=[V], 4/2 h. n., 2 steps forward, t= 0.15**

Real	Desired	%Error
4737864.23	5385591	12.03%
4102870.72	4595666	10.72%
4639510.64	5222869	11.17%
4457937.76	4870710	8.47%
3725618.32	4096739	9.06%
3090968.42	3398617	9.05%
2958100.53	3266710	9.45%
2846920.51	2595111	9.70%
2729482.64	3185410	14.31%
2801609.91	3311189	15.39%
2562996.12	2706442	5.30%
2347144.01	2950426	20.45%
2250263.26	2524455	10.86%
2169104.31	2831359	23.39%
	Mean	12.10%
	Max.	23.39%
	Min.	5.30%
	Var.	0.002368846

Table 5.1.39: **Jordan Network IN=[S,A,A-1,V-1,T-1], OUT=[V], 4/2 hidden neurons, 2 steps forward, t= 0.07**

Real	**Desired**	**%Error**
4850337.27	5385591	9.94%
4262774.81	4595666	7.24%
4925139.4	5222869	5.70%
4511722.08	4870710	7.37%
4251453.68	4096739	3.78%
3472899.51	3398617	2.19%
2572386.86	3266710	21.25%
2117394.48	2595111	18.41%
2125275.60	3185410	33.28%
2163597.31	3311189	34.66%
2118643.94	2706442	21.72%
2116931.33	2950426	28.25%
2113055.94	2524455	16.30%
2105492.33	2831359	25.64%
	Mean	16.84%
	Max.	34.66%
	Min.	2.19%
	Var.	0.012155459

5.1.4 Output values of the different models for the final weeks

Table 5.1.40: **Output values of the best networks fot rhe final weeks**

Week	**Real**	**MLP**	**%Error**	**TDNN**	**%Error**	**Jordan**	**%Error**	**TLFN**	**%Error**
Week 46/2011	3078353	2296334	25.40%	3238138	5.19%	2322546	24.55%	3616954	17.50%
Week 47/2011	2249274	2585221	14.94%	2889754	28.47%	2084337	7.33%	2731955	21.46%
Week 48/2011	1117710	2773817	148.17%	2718236	143.20%	2053070	83.69%	2311505	106.81%
Week 49/2011	2826068	2267234	19.77%	2087095	26.15%	2007780	28.95%	1646248	41.75%
Week 50/2011	2896135	2165031	25.24%	2410848	16.76%	2075896	28.32%	2234517	22.84%
Week 51/2011	3034267	2581102	14.93%	2766602	8.82%	2141056	29.44%	2469145	18.62%
Week 52/2011	2425932	2636085	8.66%	2687073	10.76%	2099541	13.45%	2470963	1.86%
Week 1/2012	2424835	2742220	13.09%	2535217	4.55%	1949008	19.62%	2374722	2.07%
Week 2/2012	2366759	2330057	1.55%	2784586	17.65%	2130386	9.99%	2878174	21.61%
Week 3/2012	2390686	2329581	2.56%	2497232	4.46%	2134087	10.73%	2656307	11.11%
Week 4/2012	2386675	2305988	3.38%	2158580	9.56%	2132619	10.64%	2323563	2.64%
Week 5/2012	1634413	2315326	41.66%	2274765	39.18%	2136972	30.75%	2333384	42.77%
Week 6/2012	2966234	2313724	22.00%	2095125	29.37%	2131812	28.13%	2169201	26.87%
Week 7/2012	2388941	2181183	8.70%	2552408	6.84%	2142363	10.32%	2575628	7.81%
Week 8/2012	2398604	2690888	12.19%	2416233	0.73%	2136887	10.91%	2389445	0.38%
Week 10/2012	2694209	2314627	14.09%	2257755	16.20%	2142840	20.46%	2398532	10.97%
Week 11/2012	2539613	2318533	8.71%	2492633	1.85%	2146064	15.50%	2438918	3.96%
Week 12/2012	2941051	2482970	15.58%	2586973	12.04%	2164618	26.40%	2649110	9.93%
Week 13/2012	2931686	2385975	18.61%	2570936	12.31%	2199412	24.98%	2665433	9.08%
Week 14/2012	2938726	2671336	9.10%	2563163	12.78%	2310411	21.38%	2566969	12.65%
Week 15/2012	3224139	2664017	17.37%	2784438	13.64%	2245003	30.37%	2841395	11.87%
Week 16/2012	2640735	2669521	1.09%	2874972	8.87%	2253810	14.65%	2889522	9.42%
Week 17/2012	3630020	2865102	21.07%	2676583	26.27%	2230038	38.57%	2731920	24.74%
Week 18/2012	3491800	2446816	29.93%	3050277	12.64%	3162688	9.43%	3144851	9.94%
Week 19/2012	3556866	3005873	15.49%	3137099	11.80%	3146960	11.52%	3117903	12.34%
Week 20/2012	3329068	2973495	10.68%	3141766	5.63%	2976280	10.60%	3271411	1.73%
Week 21/2012	2575804	2990178	16.09%	3149934	22.29%	2788705	8.27%	3142106	21.99%
Week 22/2012	914036	2916377	219.07%	2867434	213.71%	2957089	223.52%	2959157	223.75%
Week 23/2012	5685877	2406561	57.67%	2120020	62.71%	3030713	46.70%	2180430	61.65%
		Max.	219.07%		213.71%		223.52%		223.75%
		Min.	1.09%		0.73%		7.33%		0.38%
		Mean	28.17%		27.05%		29.28%		26.56%
		Var.	0.208205		0.201017		0.163441		0.19097

Table 5.1.41: **MLP+Jordan committee,final weeks, production weights and different alphas**

α=	**1.5**	**1**	**0.5**	**0.2**	**0.1**	**0.05**	**Equal-weights**
	24.11%	24.07%	**11.37%**	11.97%	12.16%	12.24%	24.98%
	10.75%	10.73%	**5.01%**	5.31%	5.40%	5.44%	11.14%
	111.90%	111.71%	**52.30%**	55.30%	56.26%	56.71%	115.93%
	23.51%	23.47%	**11.16%**	11.72%	11.89%	11.96%	24.36%
	25.85%	25.80%	**12.22%**	12.85%	13.05%	13.14%	26.78%
	21.41%	21.38%	**10.22%**	10.70%	10.84%	10.90%	22.19%
	10.67%	10.65%	**5.07%**	5.32%	5.40%	5.43%	11.06%
	15.79%	15.76%	**7.50%**	7.87%	7.98%	8.03%	16.36%
	5.57%	5.56%	**2.69%**	2.80%	2.83%	2.85%	5.77%
	6.41%	6.40%	**3.09%**	3.22%	3.26%	3.27%	6.65%
	6.77%	6.75%	**3.25%**	3.39%	3.43%	3.45%	7.01%
	34.95%	34.89%	**16.40%**	17.31%	17.60%	17.73%	36.21%
	24.19%	24.15%	**11.46%**	12.05%	12.22%	12.30%	25.07%
	9.18%	9.16%	**4.34%**	4.57%	4.63%	4.67%	9.51%
	11.15%	11.13%	**5.25%**	5.53%	5.62%	5.66%	11.55%
	16.67%	16.65%	**7.92%**	8.31%	8.43%	8.48%	17.28%
	11.68%	11.66%	**5.57%**	5.83%	5.91%	5.95%	12.11%
	20.26%	20.23%	**9.64%**	10.11%	10.25%	10.31%	20.99%
	21.04%	21.00%	**9.97%**	10.48%	10.63%	10.70%	21.80%
	14.71%	14.69%	**7.04%**	7.36%	7.45%	7.50%	15.24%
	23.04%	23.00%	**10.97%**	11.50%	11.66%	11.73%	23.87%
	7.60%	7.58%	**3.69%**	3.83%	3.87%	3.88%	7.87%
	28.78%	28.73%	**13.72%**	14.38%	14.57%	14.65%	29.82%
	19.00%	18.96%	**8.80%**	9.34%	9.52%	9.61%	19.68%
	13.04%	13.01%	**6.12%**	6.46%	6.56%	6.61%	13.51%
	10.27%	10.25%	**4.84%**	5.10%	5.18%	5.22%	10.64%
	11.76%	11.74%	**5.49%**	5.81%	5.91%	5.96%	12.18%
	213.60%	213.24%	**100.82%**	106.13%	107.76%	108.51%	221.30%
	50.37%	50.28%	**23.68%**	24.97%	25.38%	25.56%	52.19%
Mean	27.73%	27.68%	**13.09%**	13.78%	13.99%	14.08%	28.72%
Var.	0.167650	0.167074	**0.037164**	0.041274	0.042594	0.043212	0.179940
Max.	213.60%	213.24%	**100.82%**	106.13%	107.76%	108.51%	221.30%
Min.	5.57%	5.56%	**2.69%**	2.80%	2.83%	2.85%	5.77%

Table 5.1.42: **TDNN+Jordan committee, final weeks, prod. weights/calc. weights**

	%Error	
	Prod. weights	**Calc. weights**
	13.01%	35.69%
	19.93%	-4.83%
	119.15%	49.45%
	27.28%	30.56%
	21.43%	34.97%
	17.15%	41.30%
	11.85%	15.00%
	10.64%	28.29%
	14.55%	5.58%
	6.99%	14.34%
	10.00%	11.26%
	35.77%	25.90%
	28.87%	27.42%
	8.25%	12.32%
	4.84%	16.77%
	17.92%	22.91%
	7.37%	23.35%
	17.84%	34.66%
	17.43%	32.27%
	16.26%	26.33%
	20.40%	40.00%
	11.21%	17.98%
	31.24%	45.65%
	11.34%	7.58%
	11.69%	11.36%
	7.64%	13.46%
	16.62%	0.20%
	217.68%	229.16%
	56.24%	37.49%
Mean	27.95%	30.57%
Var.	0.17980964	0.1644907
Max.	217.68%	229.16%
Min.	4.84%	-4.83%

Table 5.1.43: **TDNN+Jordan committee, final weeks, prod. weights/calc. weights**

	%Error **prod. weights**	**Calc. weights**
	24.83%	23.84%
	9.84%	0.98%
	104.99%	29.88%
	25.92%	36.61%
	27.30%	30.89%
	24.65%	41.55%
	11.87%	17.45%
	17.46%	25.07%
	7.20%	17.03%
	8.03%	17.55%
	8.24%	16.70%
	34.35%	21.64%
	26.11%	33.25%
	9.78%	11.67%
	11.33%	9.84%
	18.36%	25.78%
	13.26%	21.17%
	22.83%	35.43%
	22.88%	30.30%
	17.32%	31.63%
	26.08%	41.22%
	10.17%	25.97%
	32.79%	53.18%
	16.20%	7.68%
	12.83%	8.21%
	10.63%	10.53%
	10.85%	1.74%
	222.05%	227.23%
	50.32%	37.54%
Mean	28.91%	30.21%
Var.	0.1670551	0.16213871
Max.	222.05%	227.23%
Min.	7.20%	0.98%

Table 5.1.44: **Committee of the three best networks, final weeks, prod. weights**

$\alpha =$	**1.5**	**1**	**0.5**	**0.2**	**0.1**	**0.05**	**Equal weights.**
	17.81%	17.82%	17.90%	18.03%	18.22%	18.10%	18.38%
	18.59%	18.57%	18.41%	18.21%	18.33%	18.13%	16.91%
	118.91%	118.93%	119.32%	119.91%	121.00%	120.22%	125.02%
	30.57%	30.53%	30.10%	29.59%	29.77%	29.36%	24.96%
	23.13%	23.13%	23.18%	23.24%	23.44%	23.26%	23.44%
	17.95%	17.96%	17.97%	17.97%	18.11%	17.96%	17.73%
	8.00%	8.02%	8.25%	8.50%	8.58%	8.59%	10.96%
	8.90%	8.94%	9.23%	9.56%	9.68%	9.70%	12.42%
	13.86%	13.83%	13.50%	13.07%	13.12%	12.87%	9.73%
	7.66%	7.65%	7.52%	7.36%	7.40%	7.28%	5.92%
	6.23%	6.25%	6.37%	6.48%	6.53%	6.52%	7.86%
	38.99%	38.97%	38.82%	38.68%	39.00%	38.64%	37.20%
	26.73%	26.73%	26.70%	26.65%	26.85%	26.62%	26.50%
	8.33%	8.33%	8.36%	8.39%	8.47%	8.41%	8.62%
	5.30%	5.32%	5.54%	5.82%	5.91%	5.94%	7.94%
	14.99%	15.01%	15.16%	15.32%	15.45%	15.38%	16.92%
	7.04%	7.06%	7.21%	7.37%	7.45%	7.44%	8.69%
	15.31%	15.34%	15.56%	15.80%	15.95%	15.90%	18.01%
	15.41%	15.44%	15.71%	16.01%	16.17%	16.13%	18.63%
	13.90%	13.91%	13.96%	13.98%	14.08%	13.98%	14.42%
	17.59%	17.62%	17.87%	18.12%	18.29%	18.22%	20.46%
	8.72%	8.72%	8.68%	8.60%	8.64%	8.55%	8.20%
	27.44%	27.46%	27.56%	27.63%	27.85%	27.65%	28.64%
	14.65%	14.66%	14.88%	15.19%	15.38%	15.35%	17.33%
	12.69%	12.69%	12.71%	12.75%	12.86%	12.77%	12.94%
	6.52%	6.54%	6.74%	6.97%	7.05%	7.07%	8.97%
	17.75%	17.73%	17.54%	17.33%	17.45%	17.24%	15.55%
	220.34%	220.33%	220.22%	220.10%	221.92%	220.05%	218.77%
	57.71%	57.69%	57.51%	57.33%	57.79%	57.25%	55.69%
Mean	27.62%	27.63%	27.67%	27.72%	27.96%	27.74%	28.17%
Var.	0.185444767	0.1854107	0.185258702	0.185209	0.188328673	0.1852376	0.18472053
Max.	220.34%	220.33%	220.22%	220.10%	221.92%	220.05%	218.77%
Min.	5.30%	5.32%	5.54%	5.82%	5.91%	5.94%	5.92%

Table 5.1.45: **Committee errors for different alphas and four networks, final weeks, prod. weight**

$\alpha =$	**1.5**	**1**	**0.5**	**0.2**	**0.1**	**0.05**	**Equal weights**
	18.09%	18.14%	18.16%	18.14%	18.18%	18.12%	18.15%
	18.10%	18.02%	18.01%	18.05%	18.09%	18.09%	18.06%
	120.01%	119.79%	119.93%	120.25%	120.57%	120.47%	120.30%
	29.36%	29.34%	29.29%	29.23%	29.27%	29.20%	29.25%
	23.27%	23.31%	23.31%	23.29%	23.34%	23.27%	23.30%
	18.00%	18.08%	18.06%	17.99%	18.01%	17.93%	18.00%
	8.63%	8.67%	8.68%	8.67%	8.69%	8.66%	8.68%
	9.74%	9.82%	9.84%	9.81%	9.84%	9.79%	9.82%
	12.89%	12.85%	12.80%	12.77%	12.78%	12.75%	12.77%
	7.30%	7.32%	7.29%	7.25%	7.26%	7.22%	7.26%
	6.56%	6.59%	6.58%	6.56%	6.57%	6.55%	6.57%
	38.58%	38.53%	38.53%	38.58%	38.66%	38.61%	38.60%
	26.65%	26.65%	26.64%	26.61%	26.67%	26.60%	26.63%
	8.41%	8.43%	8.43%	8.42%	8.44%	8.41%	8.42%
	5.95%	5.99%	6.02%	6.02%	6.04%	6.01%	6.02%
	15.41%	15.45%	15.45%	15.43%	15.46%	15.41%	15.44%
	7.47%	7.53%	7.54%	7.50%	7.52%	7.47%	7.51%
	15.95%	16.03%	16.04%	15.99%	16.02%	15.95%	16.00%
	16.17%	16.24%	16.26%	16.23%	16.27%	16.20%	16.24%
	14.03%	14.07%	14.06%	14.01%	14.03%	13.97%	14.02%
	18.28%	18.37%	18.37%	18.32%	18.35%	18.27%	18.33%
	8.60%	8.63%	8.61%	8.55%	8.56%	8.51%	8.56%
	27.72%	27.79%	27.77%	27.70%	27.75%	27.65%	27.72%
	15.28%	15.26%	15.31%	15.40%	15.45%	15.45%	15.41%
	12.75%	12.75%	12.76%	12.77%	12.80%	12.78%	12.78%
	7.08%	7.11%	7.13%	7.14%	7.16%	7.13%	7.14%
	17.20%	17.14%	17.13%	17.16%	17.20%	17.19%	17.17%
	220.06%	220.08%	220.07%	220.03%	220.49%	220.01%	220.15%
	57.20%	57.13%	57.12%	57.18%	57.30%	57.22%	57.20%
Mean	27.75%	27.76%	27.77%	27.76%	27.82%	27.76%	27.78%
Var.	0.18504417	0.18484475	0.18490537	0.18512727	0.18594995	0.18528731	0.18531708
Max,	220.06%	220.08%	220.07%	220.03%	220.49%	220.01%	220.15%
Min.	5.95%	5.99%	6.02%	6.02%	6.04%	6.01%	6.02%

Table 5.1.46: **Error variation for TDNN+Jordan committee, final weeks, prod. weights**

$\alpha =$	**1.5**	**1**	**0.5**	**0.2**	**0.1**	**0.05**	**Equal weights**
	15.28%	15.26%	**7.04%**	7.28%	7.34%	7.36%	14.87%
	18.40%	18.37%	**8.66%**	8.85%	8.89%	8.90%	17.90%
	116.61%	116.44%	**54.56%**	55.93%	56.22%	56.34%	113.45%
	28.32%	28.28%	**13.18%**	13.55%	13.63%	13.67%	27.55%
	23.17%	23.14%	**10.75%**	11.07%	11.14%	11.17%	22.54%
	19.66%	19.64%	**9.08%**	9.38%	9.44%	9.47%	19.13%
	12.44%	12.42%	**5.79%**	5.95%	5.99%	6.00%	12.11%
	12.42%	12.40%	**5.73%**	5.92%	5.96%	5.98%	12.09%
	14.21%	14.18%	**6.65%**	6.81%	6.85%	6.86%	13.82%
	7.81%	7.80%	**3.61%**	3.73%	3.75%	3.76%	7.60%
	10.38%	10.37%	**4.83%**	4.97%	5.00%	5.01%	10.10%
	35.94%	35.89%	**16.78%**	17.22%	17.32%	17.35%	34.97%
	29.55%	29.51%	**13.77%**	14.15%	14.23%	14.26%	28.75%
	8.82%	8.81%	**4.09%**	4.22%	4.24%	4.25%	8.58%
	5.98%	5.97%	**2.75%**	2.85%	2.87%	2.88%	5.82%
	18.84%	18.81%	**8.76%**	9.01%	9.07%	9.09%	18.33%
	8.92%	8.90%	**4.10%**	4.24%	4.28%	4.29%	8.68%
	19.76%	19.73%	**9.15%**	9.43%	9.50%	9.52%	19.22%
	19.16%	19.14%	**8.88%**	9.15%	9.21%	9.24%	18.65%
	17.56%	17.53%	**8.14%**	8.39%	8.44%	8.47%	17.08%
	22.62%	22.59%	**10.47%**	10.80%	10.87%	10.90%	22.01%
	12.09%	12.07%	**5.61%**	5.78%	5.81%	5.83%	11.76%
	33.32%	33.28%	**15.47%**	15.93%	16.03%	16.07%	32.42%
	11.34%	11.33%	**5.30%**	5.44%	5.47%	5.48%	11.04%
	11.99%	11.97%	**5.58%**	5.74%	5.77%	5.78%	11.66%
	8.34%	8.33%	**3.87%**	3.98%	4.01%	4.02%	8.12%
	15.71%	15.68%	**7.37%**	7.54%	7.58%	7.59%	15.28%
	224.71%	224.39%	**104.64%**	107.55%	108.19%	108.45%	218.62%
	56.23%	56.15%	**26.26%**	26.95%	27.10%	27.16%	54.71%
Mean	37.73%	37.68%	17.56%	18.06%	18.17%	18.21%	36.71%
Var.	0.363556382	0.36251442	0.078877908	0.08329992	0.084295592	0.0846875	0.344105463
Max.	224.71%	224.39%	104.64%	107.55%	108.19%	108.45%	218.62%
Min.	8.34%	8.33%	3.87%	3.98%	4.01%	4.02%	8.12%

Table 5.1.47: **Optimun ensemble's errors, final weeks, calc. weights, four networks**

	%Error
	2.56%
	19.69%
	67.99%
	34.95%
	19.84%
	25.66%
	18.26%
	13.44%
	27.23%
	13.61%
	19.74%
	24.85%
	37.35%
	8.77%
	-0.53%
	25.08%
	10.25%
	26.16%
	21.07%
	28.23%
	30.46%
	25.71%
	48.51%
	-12.36%
	6.79%
	6.32%
	12.30%
	217.78%
	48.90%
Mean	28.57%
Max.	217.78%
Min.	-12.36%
Var.	0.15837759

These errors are found as a linear combination of the individual errors using "optimal" coefficients. If the previously negative errors are considered, in magnitude we get that:

Mean	29.46%
Max.	217.78%
Min.	0.53%
Var.	1477.5722

5.2 Results for networks committees

Table 5.2.1: **Optimal committee. Errors found without iterations**

	%Error
	12.07%
	4.74%
	14.15%
	9.35%
	6.72%
	3.88%
	3.85%
	11.85%
	8.03%
	11.05%
	0.31%
	5.14%
	1.75%
	34.02%
Mean	9.21%
Max.	34.02%
min.	0.31%
Var.	0.0062904

Table 5.2.2: **Ensemble of the three best networks, final weeks, production weights**

	Error obtained with prod. weights	**calc. weights**
	11.41%	15.50%
	24.37%	24.07%
	145.20%	120.21%
	24.12%	39.34%
	19.34%	21.35%
	10.59%	15.06%
	10.08%	1.89%
	7.13%	0.68%
	12.64%	20.81%
	3.82%	9.32%
	7.61%	2.33%
	40.02%	44.17%
	27.06%	26.47%
	7.40%	7.36%
	4.26%	-0.04%
	15.50%	10.44%
	3.91%	2.26%
	13.05%	8.08%
	14.20%	7.91%
	11.56%	10.82%
	14.69%	9.62%
	6.38%	7.55%
	24.54%	22.23%
	18.10%	12.59%
	12.96%	12.75%
	7.18%	1.76%
	20.44%	23.68%
	215.33%	221.90%
	61.24%	63.86%
Mean	27.38%	26.34%
Var.	0.2022145	0.199333973
Max.	215.33%	221.90%
min.	3.82%	-0.04%

Table 5.2.3: **Errors of the MLP+Jordan committee, production data and different alphas (calc. weights)**

$\alpha =$	1.50	1	**0.5**	0.20	0.10	0.05	Equal weights
	4.41%	4.40%	**2.15%**	2.23%	2.25%	2.26%	13.48%
	15.58%	15.55%	**7.13%**	7.61%	7.78%	7.86%	14.12%
	14.11%	14.08%	**6.57%**	6.96%	7.08%	7.14%	13.03%
	5.58%	5.57%	**2.62%**	2.76%	2.81%	2.83%	12.91%
	12.35%	12.33%	**5.66%**	6.04%	6.17%	6.23%	13.41%
	5.67%	5.66%	**2.68%**	2.82%	2.86%	2.88%	12.20%
	2.20%	2.19%	**1.03%**	1.09%	1.11%	1.11%	16.22%
	18.42%	18.39%	**8.72%**	9.17%	9.30%	9.36%	17.38%
	19.02%	18.98%	**8.86%**	9.38%	9.55%	9.63%	17.21%
	16.93%	16.90%	**8.04%**	8.44%	8.56%	8.61%	16.93%
	11.39%	11.37%	**5.50%**	5.73%	5.79%	5.82%	16.85%
	23.39%	23.35%	**11.11%**	11.66%	11.83%	11.90%	17.57%
	10.27%	10.26%	**4.95%**	5.16%	5.22%	5.24%	16.46%
	22.88%	22.84%	**10.84%**	11.39%	11.56%	11.63%	17.93%
Mean	13.01%	12.99%	**6.13%**	6.46%	6.56%	6.61%	15.41%
Var.	0.004648	0.004632	**0.001031**	0.001144	0.001181	0.001198	0.000429
Max.	23.39%	23.35%	**11.11%**	11.66%	11.83%	11.90%	17.93%
min.	2.20%	2.19%	**1.03%**	1.09%	1.11%	1.11%	12.20%

Table 5.2.4: **Errors for different alphas and four networks, production data, calculated weights**

$\alpha =$	**1.5**	**1**	**0.5**	**0.2**	**0.1**	**0.05**	**Equal weights**
	17.81%	17.82%	17.90%	18.03%	18.07%	18.10%	18.16%
	18.59%	18.57%	18.41%	18.21%	18.13%	18.13%	18.05%
	118.91%	118.93%	119.32%	119.91%	119.99%	120.22%	120.47%
	30.57%	30.53%	30.10%	29.59%	29.41%	29.36%	29.16%
	23.13%	23.13%	23.18%	23.24%	23.24%	23.26%	23.29%
	17.95%	17.96%	17.97%	17.97%	17.95%	17.96%	17.95%
	8.00%	8.02%	8.25%	8.50%	8.55%	8.59%	8.68%
	8.90%	8.94%	9.23%	9.56%	9.66%	9.70%	9.83%
	13.86%	13.83%	13.50%	13.07%	12.92%	12.87%	12.70%
	7.66%	7.65%	7.52%	7.36%	7.30%	7.28%	7.22%
	6.23%	6.25%	6.37%	6.48%	6.50%	6.52%	6.56%
	38.99%	38.97%	38.82%	38.68%	38.62%	38.64%	38.59%
	26.73%	26.73%	26.70%	26.65%	26.60%	26.62%	26.59%
	8.33%	8.33%	8.36%	8.39%	8.39%	8.41%	8.42%
	5.30%	5.32%	5.54%	5.82%	5.90%	5.94%	6.05%
	14.99%	15.01%	15.16%	15.32%	15.34%	15.38%	15.43%
	7.04%	7.06%	7.21%	7.37%	7.42%	7.44%	7.51%
	15.31%	15.34%	15.56%	15.80%	15.86%	15.90%	15.99%
	15.41%	15.44%	15.71%	16.01%	16.08%	16.13%	16.25%
	13.90%	13.91%	13.96%	13.98%	13.97%	13.98%	13.98%
	17.59%	17.62%	17.87%	18.12%	18.18%	18.22%	18.31%
	8.72%	8.72%	8.68%	8.60%	8.56%	8.55%	8.51%
	27.44%	27.46%	27.56%	27.63%	27.62%	27.65%	27.66%
	14.65%	14.66%	14.88%	15.19%	15.28%	15.35%	15.49%
	12.69%	12.69%	12.71%	12.75%	12.75%	12.77%	12.79%
	6.52%	6.54%	6.74%	6.97%	7.03%	7.07%	7.16%
	17.75%	17.73%	17.54%	17.33%	17.25%	17.24%	17.16%
	220.34%	220.33%	220.22%	220.10%	219.88%	220.05%	220.01%
	57.71%	57.69%	57.51%	57.33%	57.22%	57.25%	57.18%
Mean	27.62%	27.63%	27.67%	27.72%	27.71%	27.74%	27.76%
Var.	0.185444	0.185410	0.185258	0.185208	0.184886	0.185237	0.185245
Max.	220.34%	220.33%	220.22%	220.10%	219.88%	220.05%	220.01%
min.	5.30%	5.32%	5.54%	5.82%	5.90%	5.94%	6.05%

Table 5.2.5: **MLP+Jordan committee, production data, calculated weights**

	Errors
	6.02%
	11.22%
	12.68%
	5.35%
	9.25%
	6.02%
	2.24%
	19.66%
	17.23%
	18.71%
	14.64%
	25.92%
	12.95%
	24.67%
Mean	13.33%
Var.	0.005294256
Max.	25.92%
min.	2.24%

Table 5.2.6: **TDNN+Jordan committee, production data, calculated weights**

$\alpha =$	**1.5**	**1**	**0.5**	**0.2**	**0.1**	**0.05**	**Equal weights**
	6.81%	6.80%	**3.15%**	3.25%	3.27%	3.28%	6.62%
	1.68%	1.68%	**0.78%**	0.80%	0.81%	0.81%	1.63%
	12.88%	12.86%	**6.03%**	6.18%	6.21%	6.22%	12.53%
	5.71%	5.71%	**2.67%**	2.74%	2.75%	2.76%	5.56%
	5.11%	5.10%	**2.40%**	2.45%	2.46%	2.47%	4.97%
	3.36%	3.36%	**1.54%**	1.60%	1.61%	1.62%	3.27%
	3.63%	3.62%	**1.70%**	1.74%	1.75%	1.75%	3.53%
	15.53%	15.50%	**7.19%**	7.41%	7.46%	7.48%	15.10%
	17.88%	17.85%	**8.37%**	8.58%	8.62%	8.64%	17.40%
	25.46%	25.42%	**11.89%**	12.20%	12.27%	12.29%	24.76%
	15.03%	15.01%	**6.96%**	7.18%	7.22%	7.24%	14.62%
	27.23%	27.19%	**12.66%**	13.02%	13.11%	13.14%	26.49%
	15.03%	15.01%	**6.98%**	7.19%	7.23%	7.25%	14.62%
	26.93%	26.89%	**12.54%**	12.89%	12.97%	13.00%	26.20%
Mean	13.02%	13.00%	**6.06%**	6.23%	6.27%	6.28%	12.67%
Var.	0.008097	0.008074	**0.001757**	0.001855	0.001877	0.001886	0.007664
Max.	27.23%	27.19%	**12.66%**	13.02%	13.11%	13.14%	26.49%
min.	1.68%	1.68%	**0.78%**	0.80%	0.81%	0.81%	1.63%

Table 5.2.7: **committee with 4 networks, production data, calculated weights.**

$\alpha =$	**1.5**	**1.0**	**0.5**	**0.2**	**0.1**	**0.05**	**Equal weights**
	7.71%	7.69%	7.45%	7.15%	7.05%	7.01%	6.90%
	8.39%	8.40%	8.64%	9.02%	9.16%	9.23%	9.40%
	14.09%	14.09%	14.16%	14.27%	14.30%	14.34%	14.38%
	7.07%	7.06%	6.98%	6.89%	6.87%	6.86%	6.83%
	8.45%	8.46%	8.66%	8.96%	9.06%	9.12%	9.25%
	4.15%	4.15%	4.15%	4.16%	4.16%	4.16%	4.17%
	3.32%	3.32%	3.30%	3.27%	3.26%	3.26%	3.25%
	14.87%	14.88%	14.98%	15.09%	15.12%	15.15%	15.19%
	15.20%	15.24%	15.64%	16.14%	16.27%	16.36%	16.55%
	17.68%	17.72%	17.97%	18.21%	18.25%	18.30%	18.38%
	8.14%	8.17%	8.39%	8.60%	8.64%	8.66%	8.73%
	17.89%	17.95%	18.42%	18.94%	19.06%	19.14%	19.33%
	8.46%	8.49%	8.69%	8.87%	8.90%	8.93%	8.99%
	28.12%	28.09%	27.81%	27.46%	27.34%	27.31%	27.18%
Mean	11.68%	11.69%	11.80%	11.93%	11.96%	11.99%	12.04%
Var.	0.004470	0.004472	0.004482	0.004494	0.004488	0.004500	0.004507
Max.	28.12%	28.09%	27.81%	27.46%	27.34%	27.31%	27.18%
Min.	3.32%	3.32%	3.30%	3.27%	3.26%	3.26%	3.25%

Table 5.2.8: **TDNN+Jordan committee, production data, calculated weights**

	%Error
	6.20%
	1.63%
	13.22%
	5.76%
	5.47%
	2.70%
	3.79%
	14.02%
	18.35%
	25.49%
	13.56%
	25.97%
	14.09%
	26.13%
Mean	12.60%
Var.	0.007701835
Max.	26.13%
min.	1.63%

5.3 Iterated predictions for each model

Table 5.3.1: **Iterated predictions ('closed loop') for each model**

Week	**Real**	**=MLP=**	**%Error**	**=TDNN=**	**%Error**	**=Jordan=**	**%Error**
32/2011	5385591	5646154.44	4.84%	4689340.60	12.93%	4250138.91	21.08%
33/2011	4595666	4840941.73	5.34%	4162812.52	9.42%	3679669.71	19.93%
34/2011	5222869	4159463.73	20.36%	3761387.81	27.98%	3665198.06	29.82%
35/2011	4870710	3399105.99	30.21%	3488404.93	28.38%	3663092.32	24.79%
36/2011	4096739	3079483.62	24.83%	3327994.57	18.76%	3545097.89	13.47%
37/2011	3398617	2940344.58	13.48%	3241489.84	4.62%	3295762.67	3.03%
38/2011	3266710	2829340.70	13.39%	3196553.49	2.15%	3140578.69	3.86%
39/2011	2595111	2719539.43	4.79%	3173566.00	22.29%	3168777.74	22.11%
40/2011	3185410	2602746.81	18.29%	3161883.64	0.74%	2977822.02	6.52%
41/2011	3311189	2477129.17	25.19%	3155964.34	4.69%	3004659.83	9.26%
42/2011	2706442	2352101.9	13.09%	3152969.36	16.50%	2949781.58	8.99%
43/2011	2950426	2247517.88	23.82%	3151455.06	6.81%	2861529.70	3.01%
44/2011	2524455	2178816.65	13.69%	3150689.67	24.81%	2896076.23	14.72%
45/2011	2831359	2143040.98	24.31%	3150302.88	11.26%	2730897.15	3.55%
		Max	30.21%		28.38%		29.82%
		min.	4.79%		0.74%		3.01%
		Var.	0.00692363		0.00913421		0.00819588
		Mean	16.83%		13.67%		13.15%

Table 5.3.2: **Errors of the opt. committee of the best three nets, 'closed loop', prod. weights**

	%Error
	10.87%
	14.21%
	28.90%
	31.80%
	21.24%
	6.29%
	4.63%
	15.33%
	5.07%
	10.07%
	12.41%
	10.78%
	17.12%
	14.24%
Mean	14.50%
Var.	0.006175759
Max.	31.80%
min.	4.63%

Table 5.3.3: **Errors of the best three nets for different alphas ,'closed loop', calculated weights**

$\alpha =$	**1.5**	**1**	**0.5**	**0.2**	**0.1**	**0.05**	**Equal weights**
	14.21%	13.97%	13.35%	13.77%	14.16%	14.39%	14.68%
	19.46%	19.53%	19.72%	19.62%	19.52%	19.46%	19.38%
	28.75%	28.32%	27.16%	27.91%	28.62%	29.03%	29.56%
	28.57%	28.29%	27.53%	28.01%	28.47%	28.73%	29.07%
	18.53%	18.50%	18.42%	18.46%	18.51%	18.53%	18.56%
	5.92%	5.99%	6.18%	6.05%	5.94%	5.87%	5.79%
	5.56%	5.65%	5.89%	5.74%	5.59%	5.51%	5.40%
	14.90%	14.78%	14.47%	14.67%	14.86%	14.98%	15.12%
	7.68%	7.82%	8.20%	7.97%	7.74%	7.61%	7.45%
	12.48%	12.66%	13.15%	12.85%	12.55%	12.38%	12.17%
	10.25%	10.21%	10.11%	10.17%	10.22%	10.26%	10.29%
	10.30%	10.50%	11.04%	10.69%	10.36%	10.17%	9.92%
	14.46%	14.38%	14.19%	14.30%	14.42%	14.48%	14.56%
	12.69%	12.90%	13.45%	13.08%	12.74%	12.54%	12.28%
Mean	14.55%	14.54%	14.49%	14.52%	14.55%	14.57%	14.59%
Var.	0.005269	0.005063	0.004549	0.004873	0.005204	0.005404	0.005668
Max.	28.75%	28.32%	27.53%	28.01%	28.62%	29.03%	29.56%
min.	5.56%	5.65%	5.89%	5.74%	5.59%	5.51%	5.40%

Table 5.3.4: **Errors for different alphas of the best three nets,'closed loop', prod. weights**

$\alpha =$	1.50	1	0.50	0.20	0.10	**0.05**	Equal weight.
	14.96%	14.97%	14.95%	14.83%	14.78%	**14.75%**	14.68%
	19.13%	19.14%	19.20%	19.30%	19.33%	**19.34%**	19.38%
	30.23%	30.25%	30.15%	29.88%	29.76%	**29.70%**	29.56%
	29.59%	29.60%	29.51%	29.30%	29.21%	**29.17%**	29.07%
	18.68%	18.68%	18.65%	18.60%	18.59%	**18.58%**	18.56%
	5.69%	5.68%	5.70%	5.74%	5.76%	**5.76%**	5.79%
	5.24%	5.23%	5.26%	5.33%	5.35%	**5.37%**	5.40%
	15.32%	15.32%	15.29%	15.21%	15.18%	**15.16%**	15.12%
	7.16%	7.16%	7.21%	7.33%	7.37%	**7.39%**	7.45%
	11.83%	11.82%	11.89%	12.02%	12.08%	**12.10%**	12.17%
	10.41%	10.41%	10.38%	10.34%	10.32%	**10.31%**	10.29%
	9.62%	9.61%	9.65%	9.78%	9.83%	**9.86%**	9.92%
	14.74%	14.74%	14.70%	14.63%	14.60%	**14.59%**	14.56%
	12.00%	11.99%	12.02%	12.14%	12.19%	**12.22%**	12.28%
Mean	14.61%	14.61%	14.61%	14.60%	14.60%	**14.59%**	14.59%
Var.	0.006049	0.006056	0.005997	0.005841	0.005777	**0.005744**	0.005668
Max.	30.23%	30.25%	30.15%	29.88%	29.76%	**29.70%**	29.56%
min.	5.24%	5.23%	5.26%	5.33%	5.35%	**5.37%**	5.40%

Table 5.3.5: **MLP+Jordan committee's errors, 'closed loop' and calculated weights**

$\alpha =$	**1.5**	**1**	**0.5**	**0.2**	**0.1**	**0.05**	**Equal weight.**
	15.45%	15.85%	**8.90%**	9.17%	9.41%	9.55%	14.60%
	25.27%	25.93%	**16.61%**	16.40%	16.19%	16.06%	23.88%
	26.36%	27.04%	**15.51%**	15.87%	16.18%	16.36%	24.91%
	25.17%	25.83%	**15.65%**	15.73%	15.78%	15.81%	23.78%
	17.24%	17.69%	**11.49%**	11.30%	11.11%	11.00%	16.29%
	6.62%	6.79%	**4.72%**	4.55%	4.39%	4.29%	6.26%
	7.41%	7.60%	**5.23%**	5.06%	4.89%	4.79%	7.00%
	14.35%	14.73%	**9.00%**	9.02%	9.03%	9.03%	13.56%
	11.43%	11.72%	**7.98%**	7.74%	7.50%	7.37%	10.80%
	16.72%	17.15%	**11.67%**	11.32%	10.98%	10.78%	15.80%
	8.85%	9.08%	**5.87%**	5.78%	5.69%	5.64%	8.36%
	12.37%	12.70%	**9.13%**	8.71%	8.31%	8.09%	11.69%
	12.59%	12.92%	**8.23%**	8.14%	8.05%	8.00%	11.90%
	14.14%	14.51%	**10.42%**	9.95%	9.50%	9.24%	13.36%
% **Mean**	15.28%	15.68%	**10.03%**	9.91%	9.79%	9.71%	14.44%
Var.	0.004131	0.00435	**0.001453**	0.001502	0.001554	0.001588	0.003689
Max.	26.36%	27.04%	**16.61%**	16.40%	16.19%	16.36%	24.91%
min.	6.62%	6.79%	**4.72%**	4.55%	4.39%	4.29%	6.26%

Table 5.3.6: **MLP+Jordan committee's errors, 'closed loop' and prod. weights**

$\alpha =$	**1.5**	**1**	**0.5**	**0.2**	**0.1**	**0.05**	**Equal weight**
	14.09%	14.07%	**9.78%**	9.78%	9.76%	9.75%	22.47%
	23.05%	23.01%	**15.58%**	15.75%	15.81%	15.84%	22.70%
	24.04%	24.00%	**16.62%**	16.64%	16.62%	16.61%	22.66%
	22.96%	22.92%	**15.70%**	15.79%	15.81%	15.82%	22.59%
	15.73%	15.70%	**10.60%**	10.73%	10.78%	10.80%	22.56%
	6.04%	6.03%	**4.01%**	4.09%	4.12%	4.13%	22.76%
	6.76%	6.75%	**4.50%**	4.58%	4.61%	4.63%	23.30%
	13.09%	13.07%	**8.94%**	9.00%	9.01%	9.02%	23.85%
	10.42%	10.40%	**6.95%**	7.07%	7.12%	7.14%	24.21%
	15.25%	15.22%	**10.17%**	10.35%	10.41%	10.45%	24.70%
	8.07%	8.06%	**5.45%**	5.51%	5.53%	5.54%	25.03%
	11.29%	11.27%	**7.43%**	7.60%	7.67%	7.71%	25.69%
	11.49%	11.47%	**7.78%**	7.86%	7.88%	7.90%	26.26%
	12.90%	12.88%	**8.49%**	8.69%	8.77%	8.81%	26.87%
Mean	13.94%	13.92%	**9.43%**	9.53%	9.57%	9.58%	23.97%
Var.	0.003437	0.003425	**0.001649**	0.001649	0.001644	0.001641	0.000228
Max.	24.04%	24.00%	**16.62%**	16.64%	16.62%	16.61%	26.87%
min.	6.04%	6.03%	**4.01%**	4.09%	4.12%	4.13%	22.47%

Table 5.3.7: **Errors ensemble opt. 'closed loop', MLP+Jordan, prod. weights/calculated weights**

	Errors	
	Calculated weights	**Prod. weights**
	7.39%	18.71%
	23.27%	24.22%
	14.43%	30.88%
	18.32%	26.90%
	16.73%	16.05%
	8.12%	5.19%
	8.78%	5.99%
	10.86%	15.11%
	13.08%	9.50%
	19.13%	13.90%
	8.43%	8.32%
	16.81%	8.78%
	11.35%	12.22%
	19.13%	10.08%
Mean	13.99%	14.70%
Var	0.00251553	0.006281167
Max.	23.27%	30.88%
min.	7.39%	5.19%

Table 5.3.8: **TDNN+Jordan committee,'closed loop', prod. weights**

	%Error
	22.80%
	20.11%
	41.30%
	35.17%
	18.05%
	3.69%
	3.41%
	18.14%
	4.91%
	8.46%
	10.20%
	4.19%
	15.16%
	5.81%
Mean	15.10%
Var.	0.014116371
Max.	41.30%
min.	3.41%

Table 5.3.9: **TDNN+Jordan committee,'closed loop', prod. weights**

$\alpha =$	**1.5**	**1**	**0.5**	**0.2**	**0.1**	**0.05**	**Equal weight**
	21.36%	21.33%	**9.90%**	10.20%	10.27%	10.30%	20.78%
	18.14%	18.11%	**8.39%**	8.66%	8.72%	8.74%	17.65%
	41.81%	41.75%	**19.46%**	20.01%	20.13%	20.18%	40.68%
	37.31%	37.25%	**17.41%**	17.87%	17.97%	18.01%	36.30%
	19.87%	19.84%	**9.29%**	9.52%	9.58%	9.60%	19.33%
	4.10%	4.09%	**1.91%**	1.96%	1.97%	1.98%	3.98%
	3.19%	3.19%	**1.48%**	1.52%	1.53%	1.54%	3.11%
	18.67%	18.65%	**8.70%**	8.94%	8.99%	9.01%	18.17%
	3.97%	3.96%	**1.82%**	1.89%	1.90%	1.91%	3.86%
	7.77%	7.76%	**3.60%**	3.71%	3.74%	3.75%	7.56%
	11.52%	11.50%	**5.39%**	5.52%	5.55%	5.57%	11.21%
	4.87%	4.87%	**2.28%**	2.34%	2.35%	2.36%	4.74%
	16.81%	16.79%	**7.86%**	8.06%	8.10%	8.12%	16.35%
	7.09%	7.08%	**3.33%**	3.41%	3.42%	3.43%	6.90%
Mean	15.46%	15.44%	**7.20%**	7.40%	7.45%	7.46%	15.04%
Var.	0.014721	0.014679	**0.003194**	0.003373	0.003413	0.003429	0.013933
Max.	41.81%	41.75%	**19.46%**	20.01%	20.13%	20.18%	40.68%
min.	3.19%	3.19%	**1.48%**	1.52%	1.53%	1.54%	3.11%

Table 5.3.10: **Results, via iteration, of a MLP used to find the invariants**

Sales
5646154.44
4840941.73
4159463.73
3399105.99
3079483.62
2940344.58
2829340.70
2719539.43
2602746.81
2477129.17
2352101.90
2247517.88
2178816.65
2143040.98
2326947.25
2381064.25
2426377.25
2395783.75
2446592.25
2294147.25
2312018.25
2214266.75
2050533.25
2447648.50
2567263.75
2489502.50
2457000.50
2343001
2312216.25
2351420.50
2382318.75
2374639.75
204224.50
2355913.25
2364289.25
2501101
2777317.25
2942121
3431798
3759009.75

5.4 Data set

5.4.1 Data set for validation

Table 5.4.1: **Data set for final validation (weekly averaged)**

Week	**Sales**	**Max. T**	**Min. T**	**Rel. Humidity.**	**Precip.**	**Wind**	**Helioph.**
Week 46/2011	3078353	27.20	12.71	61.33	0.16	170.28	12.41
Week 47/2011	2249274	27.25	15.10	63.83	0.16	128.93	10.51
Week 48/2011	1117710	25.05	14.26	66.00	0.33	211.57	11.25
Week 49/2011	2826068	27.11	14.86	68.16	0.16	150.01	12.60
Week 50/2011	2896135	22.28	16.06	75.83	1.00	208.85	7.55
Week 51/2011	3034267	28.317	14.63	64.00	0.33	138.70	9.45
Week 52/2011	2425932	28.18	14.88	62.33	0.00	180.27	12.60
Week 1/2012	2424835	28.76	14.51	60.05	0	164.41	12.70
Week 2/2012	2366759	30.20	17.71	58.83	0.16	184.11	11.48
Week 3/2012	2390686	33.53	17.18	54.33	0.16	194.91	12.35
Week 4/2012	2386675	27.05	15.83	64.00	0.33	195.05	11.75
Week 5/2012	1634413	31.93	20.60	68.16	0.33	142.13	8.16
Week 6/2012	2966234	28.85	17.85	65.83	0.00	158.13	8.43
Week 7/2012	2388941	33.18	19.88	64.33	0.50	163.73	8.66
Week 8/2012	2398604	27.19	16.08	68.05	0.25	152.35	9.24
Week 10/2012	2694209	29.45	19.65	77	0.33	123.98	7.96
Week 11/2012	2539613	24.83	15.03	73.83	0.16	128.70	7.70
Week 12/2012	2941051	25.41	14.53	73.00	0.50	109.08	8.10
Week 13/2012	2931686	22.53	10.70	65.50	0.33	203.95	9.30
Week 14/2012	2938726	25.92	12.84	70.60	0.20	130.84	7.82
Week 15/2012	3224139	24.45	13.86	78.83	0	90.33	5.91
Week 16/2012	2640735	25.31	11.86	73.60	0.33	103.25	8.56
Week 17/2012	3630020	15.45	6.75	71.16	0.66	111.17	6.06
Week 18/2012	3491800	21.40	8.175	72.52	0.11	122.46	8.16
Week 19/2012	3556866	20.90	12.53	76.50	0.50	139.05	4.63
Week 20/2012	3329068	22.11	10.10	73.16	0.16	151.88	7.75
Week 21/2012	2575804	19.13	15.63	87.16	0.33	75.58	0.43
Week 22/2012	914036	18.50	15.50	87	0.00	86.10	0.50
Week 23/2012	5685877	18.50	15.50	87.00	0.00	86.10	0.50

5.4.2 Values of the real series

Table 5.4.2: **Data set for sales used to construct the networks**

Week	**Sales**	**Week**	**Sales**
Week 35/2000	926755	Week 10/2006	2126979
Week 36/2000	4179261	Week 11/2006	2164678
Week 37/2000	4706674	Week 12/2006	1816436
Week 38/2000	2627357	Week 13/2006	1951619
Week 39/2000	3416080	Week 14/2006	2041524
Week 40/2000	3315804	Week 15/2006	2038337
Week 41/2000	2946162	Week 16/2006	2693360
Week 42/2000	2587267	Week 17/2006	2435101
Week 43/2000	2314257	Week 18/2006	2061311
Week 44/2000	2709274	Week 19/2006	2747744
Week 44/2000	2709274	Week 20/2006	2497153
Week 45/2000	2572616	Week 21/2006	3246319
Week 46/2000	2933942	Week 22/2006	2622512
Week 47/2000	2359168	Week 23/2006	3208380
Week 48/2000	2473501	Week 24/2006	3357450
Week 49/2000	2463478	Week 25/2006	3111614
Week 50/2000	2634574	Week 26/2006	3284742
Week 51/2000	3034834	Week 27/2006	2853060
Week 52/2000	2539880	Week 28/2006	3078257
Week 53/2000	2019749	Week 29/2006	2632276
Week 1/2001	2291729	Week 30/2006	2444469
Week 2/2001	2714202	Week 31/2006	4079808
Week 3/2001	2174609	Week 32/2006	3612956
Week 4/2001	2432859	Week 33/2006	3064508
Week 5/2001	2540535	Week 34/2006	2800561
Week 6/2001	2775286	Week 35/2006	2280058
Week 7/2001	2502048	Week 36/2006	3537141
Week 8/2001	2196746	Week 37/2006	2741801
Week 9/2001	2473843	Week 38/2006	1587098
Week 10/2001	2654655	Week 39/2006	1904643
Week 11/2001	2390933	Week 40/2006	2120507
Week 12/2001	2436329	Week 44/2006	1751901
Week 13/2001	3090102	Week 45/2006	2431799
Week 14/2001	2829358	Week 46/2006	2280129
Week 15/2001	2820350	Week 47/2006	1876845
Week 16/2001	3448604	Week 48/2006	1735020
Week 17/2001	3931186	Week 49/2006	2192247
Week 18/2001	4691133	Week 50/2006	2335422
Week 19/2001	4803830	Week 51/2006	2086710
Week 20/2001	3882958	Week 52/2006	2153375
Week 21/2001	2796097	Week 53/2006	1542989
Week 22/2001	3864962	Week 1/2007	1940214
Week 23/2001	3408514	Week 2/2007	1835315
Week 24/2001	6063085	Week 3/2007	1561202
Week 25/2001	5774657	Week 4/2007	1648127
Week 26/2001	4415792	Week 5/2007	2288112

Continues in the next page...

Table 5.4.2 – Continued

Week	**Sales**	**Week**	**Sales**
Week 27/2001	5187932	Week 6/2007	2340295
Week 28/2001	4978862	Week 7/2007	1856677
Week 29/2001	5926167	Week 8/2007	2178414
Week 30/2001	4519588	Week 9/2007	2387049
Week 31/2001	3577718	Week 10/2007	1839173
Week 32/2001	3202177	Week 11/2007	2080375
Week 33/2001	3186564	Week 12/2007	1821560
Week 34/2001	2715797	Week 13/2007	1867146
Week 35/2001	3865311	Week 14/2007	2249850
Week 36/2001	4106679	Week 15/2007	2043826
Week 37/2001	4417628	Week 16/2007	2209278
Week 38/2001	2881843	Week 17/2007	2612835
Week 39/2001	2579913	Week 18/2007	3913849
Week 40/2001	2750153	Week 19/2007	3035823
Week 41/2001	2993723	Week 20/2007	3363584
Week 42/2001	2579861	Week 21/2007	4137460
Week 43/2001	2797988	Week 22/2007	2104115
Week 44/2001	1957547	Week 23/2007	3210528
Week 45/2001	2936830	Week 24/2007	3366970
Week 46/2001	2538715	Week 25/2007	2916113
Week 47/2001	2027196	Week 26/2007	5561243
Week 48/2001	2709882	Week 27/2007	5398958
Week 49/2001	3007134	Week 28/2007	4797002
Week 50/2001	3268912	Week 29/2007	4562853
Week 51/2001	2286447	Week 30/2007	4577326
Week 52/2001	1773867	Week 31/2007	4512109
Week 1/2002	2687093	Week 32/2007	3466389
Week 2/2002	2271150	Week 33/2007	3484006
Week 3/2002	2346260	Week 34/2007	3147108
Week 4/2002	2238878	Week 35/2007	2309977
Week 5/2002	2866988	Week 36/2007	2126661
Week 6/2002	2219256	Week 37/2007	2533570
Week 7/2002	2555820	Week 38/2007	2429710
Week 8/2002	2263271	Week 39/2007	2197202
Week 9/2002	2326495	Week 40/2007	2551922
Week 10/2002	2907213	Week 41/2007	2108353
Week 11/2002	2589854	Week 42/2007	1789974
Week 12/2002	2527123	Week 43/2007	1655688
Week 13/2002	2887433	Week 44/2007	2097765
Week 14/2002	2842291	Week 45/2007	2360214
Week 15/2002	3221716	Week 46/2007	2152498
Week 16/2002	3001184	Week 47/2007	1725665
Week 17/2002	3341405	Week 48/2007	1749327
Week 18/2002	3087029	Week 49/2007	2021910
Week 19/2002	3149088	Week 50/2007	2516246
Week 20/2002	3179749	Week 51/2007	2004826
Week 21/2002	3027566	Week 52/2007	1786796
Week 22/2002	3713998	Week 1/2008	1616077
Week 23/2002	5630234	Week 2/2008	1896702

Continues in the next page...

Table 5.4.2 – Continued

Week	Sales	Week	Sales
Week 24/2002	5647959	Week 3/2008	1786087
Week 25/2002	6133903	Week 4/2008	1924555
Week 26/2002	4086884	Week 5/2008	1826086
Week 27/2002	7481043	Week 6/2008	2171090
Week 28/2002	4561935	Week 7/2008	1834858
Week 29/2002	3336950	Week 8/2008	1847698
Week 30/2002	2298535	Week 9/2008	3119016
Week 30/2002	1795398	Week 10/2008	2537237
Week 31/2002	4059187	Week 11/2008	2449622
Week 32/2002	4541634	Week 12/2008	2477690
Week 33/2002	3460739	Week 13/2008	2002338
Week 34/2002	2929769	Week 15/2008	3437607
Week 35/2002	3312203	Week 16/2008	2443131
Week 36/2002	4337180	Week 17/2008	3181962
Week 37/2002	3096599	Week 18/2008	3588418
Week 38/2002	2487257	Week 19/2008	3366375
Week 39/2002	2166852	Week 20/2008	2453671
Week 40/2002	2829997	Week 21/2008	3951939
Week 41/2002	2489479	Week 22/2008	5283446
Week 42/2002	2440010	Week 23/2008	4321461
Week 43/2002	1836245	Week 24/2008	4537587
Week 44/2002	2824424	Week 25/2008	4451132
Week 45/2002	3084545	Week 26/2008	4568490
Week 46/2002	2400476	Week 27/2008	3629043
Week 47/2002	2180859	Week 28/2008	2407967
Week 48/2002	2108846	Week 29/2008	3666894
Week 49/2002	2767276	Week 30/2008	4029739
Week 50/2002	2951331	Week 31/2008	4773272
Week 51/2002	3101019	Week 32/2008	4076976
Week 52/2002	2411754	Week 33/2008	3628467
Week 1/2003	2091221	Week 34/2008	2473349
Week 2/2003	2344265	Week 35/2008	2826566
Week 3/2003	2554344	Week 36/2008	4360510
Week 4/2003	2087222	Week 37/2008	4479319
Week 5/2003	2384806	Week 38/2008	2749200
Week 6/2003	2355510	Week 39/2008	2215935
Week 7/2003	4240721	Week 40/2008	2769579
Week 8/2003	1404384	Week 41/2008	2659052
Week 9/2003	1821054	Week 42/2008	1955810
Week 10/2003	2572587	Week 43/2008	1704084
Week 11/2003	2749742	Week 44/2008	2498404
Week 12/2003	2518900	Week 45/2008	2793180
Week 13/2003	2072693	Week 46/2008	2303739
Week 14/2003	2939268	Week 47/2008	1853996
Week 15/2003	2868848	Week 48/2008	1913062
Week 16/2003	2455639	Week 49/2008	3186749
Week 17/2003	2484590	Week 50/2008	2284180
Week 18/2003	3822776	Week 51/2008	2302637
Week 19/2003	3264968	Week 52/2008	1917573

Continues in the next page...

Table 5.4.2 – Continued

Week	**Sales**	**Week**	**Sales**
Week 20/2003	2980751	Week 1/2009	2106193
Week 21/2003	3338559	Week 2/2009	2634985
Week 22/2003	3795066	Week 3/2009	2351758
Week 23/2003	4861309	Week 4/2009	2064283
Week 24/2003	3859993	Week 5/2009	2491609
Week 25/2003	2686226	Week 6/2009	3119109
Week 26/2003	4486447	Week 7/2009	2547601
Week 27/2003	5328098	Week 8/2009	1839200
Week 28/2003	4366524	Week 9/2009	2466054
Week 29/2003	4081855	Week 10/2009	2911494
Week 30/2003	3472574	Week 11/2009	2527839
Week 31/2003	4235990	Week 12/2009	2355366
Week 32/2003	4504910	Week 13/2009	2543897
Week 33/2003	3499123	Week 14/2009	2941673
Week 34/2003	3833747	Week 15/2009	3077187
Week 35/2003	3089664	Week 16/2009	2978107
Week 36/2003	3609030	Week 17/2009	2138566
Week 37/2003	3919008	Week 18/2009	3201548
Week 38/2003	2505294	Week 19/2009	4102988
Week 39/2003	2352757	Week 20/2009	4007709
Week 40/2003	2559015	Week 21/2009	3273680
Week 41/2003	2549628	Week 22/2009	5617576
Week 42/2003	2321544	Week 23/2009	5180040
Week 43/2003	2156849	Week 24/2009	4365749
Week 44/2003	2262453	Week 25/2009	4921847
Week 45/2003	2644424	Week 26/2009	5557854
Week 46/2003	2639170	Week 27/2009	5006407
Week 47/2003	2063080	Week 28/2009	5826676
Week 48/2003	2441794	Week 29/2009	5705439
Week 49/2003	2545541	Week 30/2009	6227846
Week 50/2003	2664210	Week 31/2009	4940653
Week 51/2003	2462348	Week 32/2009	4406095
Week 52/2003	2530282	Week 33/2009	3512060
Week 1/2004	2010711	Week 34/2009	2787517
Week 2/2004	2232971	Week 35/2009	2909381
Week 3/2004	2204838	Week 36/2009	4179106
Week 4/2004	1837343	Week 37/2009	3029413
Week 5/2004	2257356	Week 38/2009	3330570
Week 6/2004	2417491	Week 39/2009	3435229
Week 7/2004	2453689	Week 40/2009	3046156
Week 8/2004	2165516	Week 41/2009	3193715
Week 9/2004	2436447	Week 42/2009	2741613
Week 10/2004	3037165	Week 43/2009	2663604
Week 11/2004	2010665	Week 44/2009	2307745
Week 12/2004	1968738	Week 45/2009	3139445
Week 13/2004	2108793	Week 46/2009	2556855
Week 14/2004	2905879	Week 47/2009	2313595
Week 15/2004	2277141	Week 48/2009	2356704
Week 16/2004	2315125	Week 49/2009	2952435

Continues in the next page...

Table 5.4.2 – Continued

Week	Sales	Week	Sales
Week 17/2004	2424359	Week 50/2009	2752478
Week 18/2004	3048647	Week 51/2009	2340506
Week 19/2004	3110173	Week 52/2009	2304976
Week 20/2004	4486423	Week 1/2010	2460717
Week 21/2004	3531813	Week 2/2010	2499677
Week 22/2004	4069386	Week 3/2010	2323663
Week 23/2004	3812879	Week 4/2010	2185857
Week 24/2004	3771381	Week 5/2010	2241975
Week 25/2004	3113831	Week 6/2010	2966363
Week 26/2004	3353187	Week 7/2010	2231558
Week 27/2004	4018478	Week 8/2010	2789168
Week 28/2004	4577138	Week 9/2010	2207965
Week 29/2004	3643350	Week 10/2010	2859274
Week 30/2004	2274765	Week 11/2010	2801928
Week 31/2004	3330015	Week 12/2010	2577976
Week 32/2004	3936423	Week 13/2010	2308383
Week 33/2004	2541321	Week 14/2010	3098129
Week 34/2004	3289351	Week 15/2010	2948625
Week 35/2004	2708966	Week 16/2010	29046
Week 36/2004	2934089	Week 17/2010	3569217
Week 37/2004	3589900	Week 18/2010	3365704
Week 38/2004	2510938	Week 19/2010	4336038
Week 39/2004	2315608	Week 20/2010	3293971
Week 40/2004	2305684	Week 21/2010	3093574
Week 41/2004	2744836	Week 22/2010	4112841
Week 42/2004	2531931	Week 23/2010	4950567
Week 43/2004	2258744	Week 24/2010	4614168
Week 44/2004	2054699	Week 25/2010	4960458
Week 45/2004	2657164	Week 26/2010	4500692
Week 46/2004	2687454	Week 27/2010	3904511
Week 47/2004	2112851	Week 28/2010	6531135
Week 48/2004	2070644	Week 29/2010	5905227
Week 49/2004	2164784	Week 30/2010	5989815
Week 50/2004	2248424	Week 31/2010	6636083
Week 51/2004	2200349	Week 32/2010	5817367
Week 52/2004	2411323	Week 33/2010	4676220
Week 1/2005	1999391	Week 34/2010	3080303
Week 2/2005	2178662	Week 35/2010	4581311
Week 3/2005	2258625	Week 36/2010	4083486
Week 4/2005	2045641	Week 37/2010	3966731
Week 5/2005	2107681	Week 38/2010	2908668
Week 6/2005	2150651	Week 39/2010	2979651
Week 7/2005	2383880	Week 40/2010	2908662
Week 8/2005	2178316	Week 41/2010	3248016
Week 9/2005	1514920	Week 42/2010	1375272
Week 10/2005	2213110	Week 43/2010	4245950
Week 11/2005	2533642	Week 44/2010	2628521
Week 12/2005	2024704	Week 45/2010	3126566
Week 13/2005	2400506	Week 46/2010	2778568

Continues in the next page...

Table 5.4.2 – Continued

Week	Sales	Week	Sales
Week 14/2005	2191309	Week 47/2010	2199673
Week 15/2005	2552563	Week 48/2010	2371753
Week 16/2005	2224823	Week 49/2010	2556968
Week 17/2005	2743075	Week 50/2010	2931188
Week 18/2005	2475998	Week 51/2010	2350386
Week 19/2005	2990511	Week 52/2010	2352637
Week 20/2005	2638878	Week 1/2011	2558949
Week 21/2005	2950930	Week 2/2011	2131942
Week 22/2005	2469329	Week 3/2011	2473954
Week 23/2005	2730435	Week 4/2011	2249915
Week 24/2005	2975941	Week 5/2011	2156809
Week 25/2005	3505573	Week 6/2011	3050521
Week 26/2005	2703008	Week 7/2011	2573061
Week 27/2005	3405088	Week 8/2011	2611030
Week 29/2005	3223184	Week 9/2011	2101574
Week 30/2005	3312664	Week 10/2011	867459
Week 31/2005	2260459	Week 11/2011	929321
Week 32/2005	3085029	Week 12/2011	2198028
Week 33/2005	2461010	Week 13/2011	3925263
Week 34/2005	2252468	Week 14/2011	4659060
Week 35/2005	2960554	Week 15/2011	4054998
Week 36/2005	3057640	Week 16/2011	2482409
Week 37/2005	3256333	Week 17/2011	2169366
Week 38/2005	2141784	Week 18/2011	4560781
Week 39/2005	1718958	Week 19/2011	4671129
Week 40/2005	1868168	Week 20/2011	3446401
Week 41/2005	2214655	Week 21/2011	3868174
Week 42/2005	1790966	Week 22/2011	3676501
Week 43/2005	1681442	Week 23/2011	5594894
Week 44/2005	1644086	Week 24/2011	5540264
Week 45/2005	1965016	Week 25/2011	4771888
Week 46/2005	1938285	Week 26/2011	5966606
Week 47/2005	1864924	Week 27/2011	6532075
Week 48/2005	1675378	Week 28/2011	5547618
Week 49/2005	2074166	Week 29/2011	5280350
Week 50/2005	2180600	Week 30/2011	4682671
Week 51/2005	2103779	Week 31/2011	5519564
Week 52/2005	1935984	Week 32/2011	5385591
Week 1/2006	1679207	Week 33/2011	4595666
Week 2/2006	1989111	Week 34/2011	5222869
Week 3/2006	2095454	Week 35/2011	4870710
Week 4/2006	1699000	Week 36/2011	4096739
Week 5/2006	1627846	Week 37/2011	3398617
Week 6/2006	2052770	Week 38/2011	3266710
Week 7/2006	1995528	Week 39/2011	2595111
Week 8/2006	1851658	Week 40/2011	3185410
Week 9/2006	1713908	Week 41/2011	3311189
		Week 42/2011	2706442
		Week 43/2011	2950426

Continues in the next page...

Table 5.4.2 – Continued

Week	**Sales**	**Week**	**Sales**
		Week 44/2011	2524455
		Week 45/2011	981767

5.4.3 Values with noise

Table 5.4.3: **Sales with different added noises**

V	**V=0.1**	**V=0.7**	**V=5**	**V=10**	**V=15**	**V=25**	**V=50**
4515921	4523113	4553418	4560731	4533299	4550628	4538218	
3466389	3497134	3513503	3491294	3497326	3504252	3496443	3499196
3484006	3522067	3509683	3506420	3494447	3507289	3519288	3517659
3147108	3183175	3191555	3174085	3155007	3173641	3153494	3150754
2309977	2339920	2322089	23156	2329347	2317501	2353295	2324322
2126661	2163134	2160280	2134669	2154007	2152748	2159772	2167177
2533570	2574823	2545002	2551168	2551129	2575172	2563233	2577017
2429710	2464691	2476106	2475373	2453404	2444563	2441828	2447077
2197202	2234678	2214910	2218437	2231467	2224199	2201668	2231442
2551922	2565244	2601255	2597023	2583189	2554493	2589384	2556902
2108353	2121727	2153700	2129335	2136302	2143414	2108672	2133512
1789974	1815614	1798522	1807138	1813988	1813403	1818027	1813932
1655688	1703054	1662763	1661499	1677462	1703509	1698820	1697780
2097765	2124046	2113157	2143063	2144420	2124875	2111623	2104444
2360214	2373719	2374113	2409669	2363816	2407222	2383424	2398549
2152498	2159021	2173703	2175571	2175293	2182930	2189517	2156741
1725665	1735007	1764566	1738878	1743250	1742728	1741376	1729866
1749327	1767575	1755086	1786798	1777354	1793878	1785194	1751021
2021910	2064735	2025357	2053979	2069128	2040089	2065478	2024319
2516246	2551324	2559635	2521629	2530244	2516690	2541651	2538208
2004826	2006468	2044132	2014696	2018896	2020152	2051472	2043222
1786796	1798440	1789324	1821167	1822587	1807141	1825542	1827580
1616077	1636229	1659464	1646498	1621296	1644254	1619415	1650514
1896702	1926831	1932608	1942216	1910986	1943250	1901862	1933081
1786087	1788088	1810351	1808319	1812063	1788580	1793386	1834774
1924555	1967498	1973347	1932861	1926103	1960518	1972997	1968483
1826086	1850645	1862497	1828195	1844139	1843364	1860079	1866104
2171090	2196113	2216770	2192417	2184101	2202318	2204285	2217609
1834858	1863882	1865081	1879759	1872020	1878330	1883289	1868051
1847698	1884735	1859197	1885961	1875774	1875653	1860919	1858566
3119016	3163881	3147926	3129379	3128998	3148286	3137014	3158778
2537237	2572317	2553259	2585485	2585408	2545285	2562676	2563715
2449622	2472950	2477246	2490115	2461891	2462622	2469827	2486588
2477690	2508140	2493877	2524305	2524818	2521061	2516215	2523154
2002338	2015379	2007180	2046647	2025520	2029076	2010310	2008607
3437607	3437875	3479560	3457250	3459945	3475734	3469729	3446026
2443131	2459150	2450252	2451782	2457948	2443411	2489520	2481717
3181962	3184689	3225275	3214713	3223418	3204217	3190834	3226978
3588418	3602456	3630474	3601057	3591529	3595241	3607918	3633435
3366375	3388726	3376119	3407675	3367756	3384289	3411362	3387624
2453671	2492918	2500110	2466389	2474709	2481197	2477912	2489819
3951939	3978763	3970244	3970913	3988050	3993272	3968438	3978390
5283446	5291389	5324013	5311628	5329260	5324295	5313763	5291079
4321461	4358596	4359437	4323293	4364487	4343577	4348221	4349524
4537587	4541939	4564975	4563078	4562227	4547514	4575115	4577598
4451132	4484941	4472647	4471489	4478545	4488800	4497660	4458364

Continues in the next page . . .

Table 5.4.3 – Continued

V	V=0.1	V=0.7	V=5	V=10	V=15	V=25	V=50
4568490	4615337	4572042	4608775	4594901	4607535	4582248	4598488
3629043	3654597	3668284	3678245	3673497	3668253	3635720	3641494
2407967	2431864	2454084	2435463	2436920	2450266	2455686	2438332
3666894	3680865	3679016	3697398	3694027	3680208	3682762	3696751
4029739	4076676	4046167	4074792	4072141	4057586	4055909	4077872
4773272	4811643	4793747	4797223	4777071	4798510	4777962	4786444
4076976	4107078	4110228	4090291	4093634	4104902	4118591	4090427
3628467	3650979	3654285	3630853	3657060	3650518	3668446	3666960
2473349	2499365	2499117	2523010	2475678	2509067	2508358	2480754
2826566	2839362	2840265	2863034	2876349	2868009	2834457	2837515
4360510	4367982	4368551	4362077	4368916	4360991	4409682	4365700
4479319	4515041	4515932	4513510	4525879	4506020	4487684	4490505
2749200	2763972	2775581	2791430	2788992	2785279	2765219	2756895
2215935	2227418	2256964	2237917	2256685	2235440	2245626	2233704
2769579	2796561	2808887	2789851	2770333	2817197	2770455	2777324
2659052	2665174	2664257	2689378	2660249	2660546	2693680	2680174
1955810	1978814	1956757	1994844	1999823	1964143	1995829	1961313
1704084	1746261	1724554	1742764	1744560	1750635	1738572	1726063
2498404	2507499	2500607	2544758	2547475	2505464	2535986	2517461
2793180	2833810	2829820	2816626	2823239	2796594	2835617	2818936
2303739	2317712	2350281	2347434	2325211	2309668	2313705	2307982
1853996	1873967	1894796	1876408	1885076	1898101	1870967	1894929
1913062	1952863	1958692	1958201	1939097	1932340	1919031	1936141
3186749	3216280	3193885	3196003	3227764	3204730	3192500	3197705
2284180	2312279	2304681	2317718	2327816	2330266	2284729	2290703
2302637	2307700	2304776	2309909	2303324	2314134	2307852	2331493
1917573	1923194	1923234	1926584	1918307	1918930	1924760	1945292
2106193	2138597	2118136	2142674	2130784	2129755	2134711	2128589
2634985	2684855	2670302	2664957	2680630	2644798	2671087	2664095
2351758	2396099	2372297	2352765	2353710	2391382	2398903	2372825
2064283	2072865	2095190	2098015	2107688	2102202	2078489	2089171
2491609	2518381	2507102	2511889	2538812	2511822	2530469	2541541
3119109	3159415	3147975	3152214	3119674	3132081	3129376	3156748
2547601	2584777	2568103	2557460	2553026	2594472	2594715	2570076
1839200	1867083	1880708	1875096	1866653	1859254	1856316	1868608
2466054	2501644	2497716	2493859	2506746	2475671	2488047	2492302
2911494	2927254	2924090	2920750	2958063	2953468	2948006	2927113
2527839	2545746	2576966	2536223	2547238	2554239	2557214	2573064
2355366	2358670	2396097	2405086	2368053	2391276	2374894	2379736
2543897	2556824	2563225	2569762	2570145	2580368	2567444	2566942
2941673	2953307	2955080	2951998	2988964	2963518	2953577	2953818
3077187	3098302	3084521	3121328	3088787	3122466	3097766	3095009
2978107	3001026	3021702	3027988	3021826	3017513	2998794	2990045
2138566	2174400	2164121	2188427	2165617	2158523	2180208	2180279
3201548	3242500	3236676	3221645	3210583	3236549	3234120	3203809
4102988	4141581	4111916	4122472	4149078	4143948	4143685	4126480
4007709	4034099	4033178	4054696	4016191	4052361	4025212	4020025
3273680	3308547	3314860	3290384	3292285	3289656	3291911	3292886
5617576	5652622	5623043	5666440	5662717	5629603	5653198	5627661

Continues in the next page ...

Table 5.4.3 – Continued

V	V=0.1	V=0.7	V=5	V=10	V=15	V=25	V=50
5180040	5186959	5192003	5209790	5183276	5212119	5193455	5228634
4365749	4373251	4366352	4398386	4369903	4394714	4377373	4381730
4921847	4962407	49528	4949552	4921917	4930472	4956623	4942336
5557854	5567892	5591989	5589994	5586398	5562303	5600245	5593025
5006407	5016467	5029941	5055591	5010046	5039313	5040024	5025664
5826676	5855870	5832591	5855849	5854676	5848630	5869242	5836915
5705439	5716690	5755039	5711146	5726580	5739363	5735873	5750415
6227846	6275532	6228674	6229905	6229147	6259868	6234005	6239064
4940653	4944814	4955904	4977863	4977947	4951824	4944679	4941149
4406095	4451577	4418417	4434855	4433769	4419070	4430476	4421973
3512060	3524503	3552226	3536357	3556412	3547078	3553146	3512683
2787517	2807578	2787877	2810408	2792367	2794133	2814903	2834359
2909381	2958377	2928165	2918333	2913891	2955973	2923683	2922915
4179106	4192488	4202923	4214185	4190334	4195196	4209314	4188261
3029413	3077083	3045902	3069980	3040222	3037991	3051922	3056907
3330570	3344776	3361123	3356849	3343694	3357172	3354980	3362468
3435229	3458157	3449523	3436098	3443904	3447356	3479218	3475420
3046156	3065048	3061188	3086603	3052189	3052380	3067020	3048020
3193715	3236367	3216565	3204319	3213587	3227866	3237348	3208717
2741613	2748166	2776170	2789535	2760175	2779505	2775235	2780992
2663604	2679079	2688908	2672313	2683425	2696206	2691314	2690714
2307745	2345857	2329548	2337030	2328728	2331767	2318084	2356610
3139445	3155953	3171790	3159250	3176449	3181293	3177194	3180686
2556855	2581805	2591556	2576253	2584853	2580306	2604567	2572167
2313595	2329259	2325909	2330537	2325150	2337409	2336917	2314003
2356704	2404418	2401227	2378004	2394477	2400820	2378764	2383470
2952435	2956879	2986576	2980290	2980936	2957557	2962051	2986992
2752478	2792478	2783219	2798199	2762964	2762239	2782209	2753116
2340506	2376432	2373579	2345172	2351536	2375640	2367840	2374678
2304976	2336575	2309680	2319170	2314853	2327037	2313711	2318645
2460717	2461566	2482035	2499200	2473546	2494408	2469249	2491917
2499677	2502618	2548491	2520851	2529895	2502234	2544758	2545216
2323663	2371144	2333913	2327920	2348190	2362676	2341484	2328359
2185857	2194707	2213107	2225582	2196578	2196428	2195954	2212984
2241975	2244742	2267659	2285670	2244233	2272392	2264552	2243474
2966363	2969600	3004855	29718	2992175	2998933	2999471	3001470
2231558	2239532	2253477	2240362	2267896	2273238	2259646	2239152
2789168	2802810	2829263	2825474	2833687	2793718	2800653	2814550
2207965	2221933	2216598	2243969	22466	2250529	2222170	2240417
2859274	2871235	2884130	2859944	2891058	2891024	2892998	2906652
2801928	2839977	2833143	2811526	2838065	2818135	2848907	2803635
2577976	2589851	2613920	2627404	2593477	2582705	2620958	2623327
2308383	2331728	2324852	2355552	2317123	2326249	2357382	2322258
3098129	3133512	3123451	3115007	3140256	3124621	3111983	3138088
2948625	2988384	2952592	2967035	2996034	2968240	2949140	2966855
29046	2916818	2938311	2906777	2921013	2925734	2954051	2943557
3569217	3601026	3581472	3569775	3616250	3584055	3591598	3587252
3365704	3381426	3384360	3383851	3393755	3383476	3394188	3408794
4336038	4349895	4378710	4363453	4359310	4373657	4345027	4378437

Continues in the next page ...

Table 5.4.3 – Continued

V	V=0.1	V=0.7	V=5	V=10	V=15	V=25	V=50
3293971	3298557	3301880	3316025	3315342	3305660	3342048	3300307
3093574	3103212	3126713	3133202	3133226	3104905	3131963	3139386
4112841	4136564	4159971	4115459	4114137	4146377	4115836	4146861
4950567	4982107	4977581	4974373	5000423	5000429	4993551	4998910
4614168	4648760	4620608	4635507	4637056	4638982	4628837	4652579
4960458	5003294	5007906	4974231	4979479	4993286	4962284	4999069
4500692	4531255	4546291	4518875	4514420	4539144	4534005	4550502
3904511	3917086	3944001	3910531	3914438	3907815	3907887	3918184
6531135	6549030	6533535	6550838	6560174	6548733	6554073	6551071
5905227	5907186	5919392	5934577	5949235	5910243	59238	5914466
5989815	6009119	6007434	6017709	6000389	6032493	6013373	5992923
6636083	6643415	6677964	6638282	6642730	6659542	6650349	6643795
5817367	5862825	5855150	5824084	5828264	5831523	5832894	5855520
4676220	4693780	4717021	4693560	4696408	4678051	4677669	4717411
3080303	3089388	3125896	3080867	3111128	3110398	3096756	3087856
4581311	4596390	4607979	4581326	4582443	4613149	4613399	4592269
4083486	4104650	4098691	4105364	4089142	4100606	4100285	4087763
3966731	3995390	3977506	3976200	3993161	3967997	3970502	3986951
2908668	2915841	2937068	2952127	2917077	2932166	2910632	2947995
2979651	2982481	3003399	3015724	2991193	3029163	2991965	2986699
2908662	2921146	2913357	2933778	2957381	2914202	2924478	2919490
3248016	3252934	3284807	3253201	3256962	3270963	3271097	3297862
1375272	1406712	1389698	1378982	1401373	1386685	1399334	1422390
4245950	4295494	4277137	4262119	4286139	4268677	4289294	4251637
2628521	2677853	2637696	2642936	2669960	2657837	2639246	2640965
3126566	3131633	3131952	3129279	3133821	3127135	3173723	3158203
2778568	2815410	2826002	2814566	2800058	2793372	2822854	2789386
2199673	2210377	2229061	2245222	2200323	2239364	2230571	2218921
2371753	2377900	2401985	2407066	2397743	2420469	2410801	2410205
2556968	2587091	2561112	2557095	2559235	25728	2560522	2589611
2931188	2935853	2946834	2949594	2961551	2958152	2936968	2964689
2350386	2361781	2367248	2368668	2371195	2396707	2382536	2389868
2352637	2388574	2383140	2380523	2399173	2365157	2373715	2389603
2558949	2598766	2590508	2600858	2598588	2577642	2584330	2560789
2131942	2173166	2154697	2170593	2153767	2155109	2151918	2149376
2473954	2499970	2517028	2482011	2496722	2499448	2506698	2487913
2249915	2292303	2279671	2292500	2257795	2295199	2276998	2250056
2156809	2180427	2168674	2201554	2163871	2170136	2170373	2165150
3050521	3090757	3072390	3095774	3063947	3068619	3077166	3092327
2573061	2591275	2580054	2583960	2595648	2602860	2576206	2590079
2611030	2657098	2625932	2642443	2656678	2616383	2629384	2639766
2101574	2124405	2102925	2134931	2111571	2102789	2115257	2117597
867459	897278	912469	873711	893388	912510	880084	916426
929321	950935	965459	940460	951162	950014	972042	946329
2198028	2199581	2198686	2226381	2228129	2220590	2227143	2209181
3925263	3949922	3967097	3968864	3958884	3935407	3931490	3954551
4659060	4697157	4676085	4687641	4669725	4662133	4689157	4695986
4054998	4097852	4067458	4062767	4101023	4085850	4063466	4065923
2482409	2482629	2524574	2523014	2486998	2529005	2518922	2500080

Continues in the next page . . .

Table 5.4.3 – Continued

V	V=0.1	V=0.7	V=5	V=10	V=15	V=25	V=50
2169366	2206930	2213868	2211308	2175690	2214623	2205657	2190442
4560781	4581366	4573554	4594780	4602139	4600018	4604784	4610142
4671129	4698010	4713711	4694831	4692694	4716401	4708451	4686113
3446401	3465606	3491549	3458821	3477855	3460787	3486894	3467776
3868174	39046	3913861	3899261	3910808	3912460	3871201	3901006
3676501	3706511	3697759	3698477	3690894	3716110	3698533	3702469
5594894	5603787	5617914	5601279	5631282	5634269	5637396	5604929
5540264	5571243	5548799	5563071	5582399	5586711	5570818	5565907
4771888	4791111	4778080	4801202	4820783	4804612	4812140	4777263
5966606	5994126	5999974	6004091	6010155	6002782	5979185	5976956
6532075	6569000	6541711	6538099	6556623	6547879	6549906	6571880
5547618	5553333	5581931	5586644	5589488	5548636	5568564	5564356
5280350	5321685	5284557	5300900	5289315	5298381	5302223	5288539
4682671	4715150	4724715	4714712	4712986	4703578	4703824	4695211
5519564	5543872	5539640	5566954	5566075	5550572	5522852	5554922
5385591	5385999	5414256	5433983	5405638	5426730	5424267	5427336
4595666	4598186	4638784	4637778	4628413	4610968	4607697	4597214
5222869	5258633	5233573	5263613	5254689	5265681	5240726	5223115
4870710	4883640	4905255	4909035	4909746	4911116	4907825	4902171
4096739	4116010	4137913	4111926	4139095	4119773	4098683	4116935
3398617	3435449	3442502	3432393	3445530	3440696	3434202	3399641
3266710	3266919	3288433	3287592	3282997	3316180	3303340	3314020
2595111	2630196	2599569	2623331	2604749	2610708	2613799	2601557
3185410	3192755	3214499	3231067	3187417	3193967	3203210	3193315
3311189	3315066	3331912	3315827	3341124	3360103	3318872	3340250
2706442	2735099	2725649	2723590	2742883	2752435	2740312	2753165
2950426	2986375	2964278	2990949	2953778	2960182	2983472	2985259
2524455	2568497	2572212	2571194	2542381	2530980	2560636	2537483
981767	1004779	1015236	1007564	990848	1025589	1014508	1019723

5.4.4 Temperature and other atmospheric variables

Table 5.4.4: **Temperature and other atmospheric variables**

Week	Sales	T Max. Aver.	T min. Aver.	Rel. Humidity Aver.	Precip. Aver.	Wind Aver.	Heliophany Aver.
Week 35/2000	926755	14,1000	1,7000	76,1000	0	105,4000	10,3000
Week 36/2000	4179261	19,6333	8,5667	85,8667	9,1500	183,8333	6,6500
Week 37/2000	4706674	13,1667	8,3167	84,6000	7,3167	184,1667	1,5333
Week 38/2000	2627357	17,6000	9,4750	87,3250	8,0750	205,2750	6,7500
Week 39/2000	3416080	20,8400	6,3400	73,4000	0	118,5400	10,8200
Week 40/2000	3315804	16,1833	10,1167	89,2667	5,5167	181,1167	5,7667
Week 41/2000	2946162	21,4500	8,9667	71,2500	0,3000	177,4333	9,8167
Week 42/2000	2587267	24,2600	14,2200	90,6000	1,3000	119,0200	5,8400
Week 43/2000	2314257	18,8833	11,9333	92,7833	1,7333	207,3167	3,5500
Week 44/2000	2709274	23,6333	11,3833	81,6500	0,6167	152,0333	9,4667
Week 44/2000	2709274	0	0	0	0	0	0
Week 45/2000	2572616	21,9000	13,7000	91,0400	5,1000	86,2400	6,2200
Week 46/2000	2933942	20,5400	9,0800	67,7600	0,3200	200,3400	12,0600
Week 47/2000	2359168	24,2667	11,0167	73,5000	6,0333	157,4333	10,1667
Week 48/2000	2473501	24,6833	12,9500	75,5167	0	98,0667	11,6833
Week 49/2000	2463478	27,4200	13,9000	74,6200	0	91,2200	11,3000
Week 50/2000	2634574	28,5333	16,1667	75,9667	4,6000	130,9500	10,7500
Week 51/2000	3034834	26,6167	11,7667	69,1667	0,1667	123,3000	11,9167
Week 52/2000	2539880	27,8000	16,3400	79,5800	14,9600	119,5600	8,5600
Week 53/2000	2019749	33,8000	20,5000	76,4800	0,5600	106,7800	11,6400
Week 1/2001	2291729	26,8000	14,8167	71,1333	4,7500	182,6667	10,8167
Week 2/2001	2714202	27,5000	17,2667	83,8167	9,9500	142,2000	10,1000
Week 3/2001	2174609	30,8500	19,4667	84,8667	3,8667	77,6333	9,7833
Week 4/2001	2432859	27,6167	15,1833	78,5500	1,3333	109,3333	11,1167
Week 5/2001	2540535	30,1500	17,9500	82,7667	7	128,2000	10,0500
Week 6/2001	2775286	27,7167	14,5167	80,8667	5	59,5333	10,8667
Week 7/2001	2502048	31,6667	20,3667	86,0167	13,9833	112,4167	9,2500
Week 8/2001	2196746	28,8200	21,6600	93,1600	5,5000	154,9000	2,9400
Week 9/2001	2473843	29,1167	17,8667	85,0333	0,2333	80,3500	8,9333
Week 10/2001	2654655	30,5667	19,4167	83,9500	0	125,5333	10,5500
Week 11/2001	2390933	21,9167	16,0333	91,0167	26,5833	251,2500	2,5500
Week 12/2001	2436329	23,0500	15,3167	93,2000	0,6333	116,3833	6,4167
Week 13/2001	3090102	22,5833	13,2667	86,1167	0,5833	0	6,4167
Week 14/2001	2829358	24,3000	14,0500	88,8500	0,6000	0	6,9250
Week 15/2001	2820350	23,3500	12,1667	80,3667	2,1667	161,6833	9,3667
Week 16/2001	3448604	19,2800	7,8000	85,1800	0	77,7800	7,2600
Week 17/2001	3931186	15,3600	8,7000	89,6800	3,5000	150,8600	4,3400
Week 18/2001	4691133	16,5000	9,9000	94,3667	1,4333	100,1167	5
Week 19/2001	4803830	15,9167	7,9167	92,9500	0,4167	72,6500	3,1500

Continues in the next page . . .

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 20/2001	3882958	21,4800	10,1400	93,8400	2,3600	108,0600	5,5000
Week 21/2001	2796097	21,1667	13,3833	96,8167	3,9833	111,7333	4
Week 22/2001	3864962	20,2167	9,4500	93,3000	12,4833	92,1500	5,0667
Week 23/2001	3408514	20,0667	13,8833	96,3333	6	236,1667	3,1000
Week 24/2001	6063085	12,7200	2,8000	89,9600	0,0800	62,6800	6,2200
Week 25/2001	5774657	17,0333	4,9500	92,7167	0	77,6833	7,8833
Week 26/2001	4415792	17,0667	6,8000	89,5000	4,4000	137,1333	4,6500
Week 27/2001	5187932	14,6167	5,3000	88,3500	0,8333	129,2833	4,3833
Week 28/2001	4978862	16,4600	9,7400	97,6400	4,9000	206,7800	1,9200
Week 29/2001	5926167	10,8400	3,0400	85,0600	0,1400	148,8800	5,6600
Week 30/2001	4519588	24,4833	13,3667	89,2500	0	162,3333	6,7000
Week 31/2001	3577718	18,4500	6,9667	91,0500	2,2500	118,7333	7,2500
Week 32/2001	3202177	18,2500	13,3833	94,8167	8,7500	105,1500	0,7167
Week 33/2001	3186564	19,4600	6,6200	85,1000	0	69,3000	8,9600
Week 34/2001	2715797	19,7600	9,2000	89,3000	7,4600	124,7200	5,6800
Week 35/2001	3865311	16,1600	8,3200	86,9600	1,6200	193,6800	5,9800
Week 36/2001	4106679	16,5600	5,0800	79,2000	0	155,3200	6,7800
Week 37/2001	4417628	22,6200	7,3800	80,3800	0	99,0400	9,7200
Week 38/2001	2881843	20,1500	12,4167	86,9167	0,6333	258,2167	3,8500
Week 39/2001	2579913	18,7800	13,7000	95,7600	5,6800	133,5200	2,3000
Week 40/2001	2750153	19,2000	10,7800	89,9000	8,4200	171,5200	4,9800
Week 41/2001	2993723	21,5600	15,2000	95,2200	5,6800	208,3400	2,0600
Week 42/2001	2579861	24,8000	12,9600	81,9200	17,3000	141,6200	7,5400
Week 43/2001	2797988	22,2333	12,2667	87,3167	2,4500	144,8833	8,7833
Week 44/2001	1957547	20,8750	11,9000	86,5000	8	157,4250	7,9750
Week 45/2001	2936830	21,5400	11,9800	73,1400	0,7800	132,8400	10,1000
Week 46/2001	2538715	27,9800	17,2400	84,0200	0,9000	145,0800	8,1800
Week 47/2001	2027196	22,7000	10,6200	80,6200	0,4400	85,8000	9,8000
Week 48/2001	2709882	24,5200	12,9400	87,7400	3,5000	154,7800	8,6800
Week 49/2001	3007134	27,5000	15,0600	82,3600	0	139,4200	11,5200
Week 50/2001	3268912	25,8833	14,7667	79,5000	1,7833	124,4167	8,6833
Week 51/2001	2286447	31,7800	17,7400	72,9200	0	145,7000	12,3600
Week 52/2001	1773867	25,8500	17,2500	88,8500	0,8000	134,1750	9,0500
Week 1/2002	2687093	29,7833	16,5167	80,6667	0,2500	133,2333	12,6333
Week 2/2002	2271150	24,3200	13,0400	73,8000	0,8000	112,7600	10,8400
Week 3/2002	2346260	29,9800	16,2800	71,3800	0	98,2600	9,9000
Week 4/2002	2238878	28,9400	17,7800	75,5000	2,8000	207,8000	11,1200
Week 5/2002	2866988	25,9167	17,1167	90,4000	2,6667	140,0667	5,1667
Week 6/2002	2219256	28,2750	16,4000	82,5000	0	98,1500	10,9750
Week 7/2002	2555820	24,3600	17,8400	90,2200	3,7600	167,1800	4,2600

Continues in the next page . . .

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 8/2002	2263271	30,6600	17,2400	69,1800	4,3000	172,7400	8,6600
Week 9/2002	2326495	27,3400	17,8800	90,0600	12	100,8600	7,2000
Week 10/2002	2907213	27,7000	18,6200	93,0200	15,6000	129,8000	5,6400
Week 11/2002	2589854	22,9800	16,9600	93,2200	6,2800	155,2800	4,8800
Week 12/2002	2527123	21,9250	13,4750	89,7750	13,7500	111,1500	5,2250
Week 13/2002	2887433	23,1000	12,3600	86,2000	2,4800	118,7200	8,6000
Week 14/2002	2842291	22,8200	16,7800	93,3800	4,5200	59,6800	2,9200
Week 15/2002	3221716	20,9167	12,8167	85,9667	4,5333	147,8667	6,8167
Week 16/2002	3001184	18,3000	8,6500	94,4000	1,8500	58,9500	4,4500
Week 17/2002	3341405	21,3000	9	92,6750	1,1000	53,7500	6,2750
Week 18/2002	3087029	21,4800	7,1000	88,9200	0	111	9
Week 19/2002	3149088	21,1400	16,1200	95,9600	21,8600	247,1600	1,2200
Week 20/2002	3179749	18,8600	7,8200	91,3400	0	127,9600	6,8000
Week 21/2002	3027566	17,3400	9,6000	90,2400	12,5800	131,1200	4,7400
Week 22/2002	3713998	19,7200	11,5600	91,1000	3,9200	130,5200	4,5800
Week 23/2002	5630234	11,6500	0,7750	84,3250	0	110,0750	7,8250
Week 24/2002	5647959	14,4250	5,7250	87,1250	0	168,3000	6,6250
Week 25/2002	6133903	13,9400	3,4800	93,3200	0	88,2200	5,4200
Week 26/2002	4086884	13,4800	8,7600	98,1200	11,4800	149,6600	1,2800
Week 27/2002	7481043	12,9167	3,0167	87,4000	0,3333	126,7333	5,2000
Week 28/2002	4561935	17,7000	7,2600	91,8600	0	138,6600	6,6600
Week 29/2002	3336950	14,6600	4,3000	89,6800	1,9200	124,1200	4,0600
Week 30/2002	2298535	15,9000	6,6000	88,5000	0,5333	192	4,3333
Week 30/2002	1795398	11,5500	6,4000	87,7000	0	107,7500	4,9000
Week 31/2002	4059187	13,8200	5,5200	90,4200	0,1600	180,6800	5,7800
Week 32/2002	4541634	16,2200	2,3200	84,5200	0	152,3800	8,0600
Week 33/2002	3460739	17,6400	8,2400	89,3000	8,7000	166	5,7400
Week 34/2002	2929769	21,6000	13,0167	84,1500	2,8333	370,4667	5,4500
Week 35/2002	3312203	15,4200	5,5000	84,0800	2	177,8000	6,0400
Week 36/2002	4337180	15,7857	6,8857	90,2000	3,8000	206,6286	6,5429
Week 37/2002	3096599	20,0833	9,8833	89,4667	4,1167	192,2667	5,6833
Week 38/2002	2487257	21,6500	9,7500	83,9833	0	177,6333	8,4667
Week 39/2002	2166852	22,2833	11,9500	88,7500	3,3000	189,1667	7,2167
Week 40/2002	2829997	24,2000	14,9500	86,0833	7,4833	176,1000	6,5000
Week 41/2002	2489479	23,1500	12,6500	86,9500	1,3167	217,0333	5,7167
Week 42/2002	2440010	22,6667	9,5333	80,1000	1,8667	188,6500	9,6833
Week 43/2002	1836245	22,5600	9,3400	79,0600	1,0800	153,1800	8,9400
Week 44/2002	2824424	21	10,9000	83,7667	8,1667	196,6333	8,1167
Week 45/2002	3084545	27,1333	13,7500	80,0500	4,6667	224,7333	10,1500
Week 46/2002	2400476	26,1500	12,7667	77,0500	0,3000	146,4333	11,2333

Continues in the next page . . .

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 47/2002	2180859	24,4000	14,9333	87,6833	7,1333	183,6167	7,5833
Week 48/2002	2108846	26,5800	15,2800	85,2600	0,2800	128,0400	7,7800
Week 49/2002	2767276	24,2600	15,1600	88,9200	0,0800	157,5200	8,0200
Week 50/2002	2951331	27,0400	14,5600	74,0400	3,8000	190,8600	10,8800
Week 51/2002	3101019	27,4800	14,0400	74,1400	11,9000	153,8000	10,5400
Week 52/2002	2411754	21,6000	16	93,5000	28,7000	240,9500	2,7500
Week 1/2003	2091221	28,3400	15,0400	77,3200	1,9400	164,0400	12,1800
Week 2/2003	2344265	31,5000	17,5500	73,9500	0	171,3167	12,1667
Week 3/2003	2554344	27,6667	14,5500	74,8000	3,7333	163,1500	9,9333
Week 4/2003	2087222	34,6600	18,9400	71,7600	0	188,7000	12,2400
Week 5/2003	2384806	28,3333	18,8167	89,8167	4,2000	141,4167	5,2500
Week 6/2003	2355510	27,1200	17,6000	91,0400	5,4400	162,8400	6,6600
Week 7/2003	4240721	23,9000	11,7000	77,2333	0,6500	173,0167	7,3000
Week 8/2003	1404384	28,8000	18,3250	88,2250	7,9000	123,8250	7,1500
Week 9/2003	1821054	30,7500	17,6750	87,9000	0	118,7500	8,1000
Week 10/2003	2572587	24,6833	15,7333	87,6667	2,3000	169,7167	6,4333
Week 11/2003	2749742	26	12,4333	74,2833	0,7833	170,0500	9,6333
Week 12/2003	2518900	25,9500	15,5333	88,0833	0,1333	170,4667	6,3000
Week 13/2003	2072693	20,7167	11,8333	87,7667	0,5000	120,0333	4,8833
Week 14/2003	2939268	20,4500	9,9167	79,5333	0,2167	264,9000	7,7500
Week 15/2003	2868848	23,8750	8,3000	82,3250	1,5000	141,9250	8,8500
Week 16/2003	2455639	22,3500	13,0833	89,5333	1,7167	128,9500	3,5667
Week 17/2003	2484590	18,3600	7,8200	91,8200	0,0400	137,7600	4,6000
Week 18/2003	3822776	18,3000	6,5167	88,3500	0,0667	156,9667	6,2333
Week 19/2003	3264968	21,9500	13,5667	93,4000	4,3833	164,4000	2,4667
Week 20/2003	2980751	17,1833	12,9500	95,9000	11,4667	288,1667	1,9167
Week 21/2003	3338559	18,9000	7,6333	84,4833	0,1667	141,6333	7,5000
Week 22/2003	3795066	15,2667	4,5833	81,7167	0,2333	150,6000	6,1167
Week 23/2003	4861309	16,3500	5,2500	92,5333	0,3333	134,3333	4,5000
Week 24/2003	3859993	18,4200	6,7400	92,6400	3,8400	166,6600	5,4200
Week 25/2003	2686226	16,1200	7,4200	89,3800	12,5600	187,5200	3,1800
Week 26/2003	4486447	15,1000	7,0167	95,6500	6,6333	98,7333	3,2500
Week 27/2003	5328098	12,3167	3,4667	85,2833	0,7167	231,7667	6,7000
Week 28/2003	4366524	15,9000	5,0800	86,0800	2,1000	171,2600	8,0800
Week 29/2003	4081855	13,6167	0,0833	86,9500	0	120,0833	8,3833
Week 30/2003	3472574	17,8000	7,6167	90,3000	3,6333	162,0833	6,6000
Week 31/2003	4235990	13,6000	7,1667	93,2833	8,4667	236,3000	2,2333
Week 32/2003	4504910	16,2667	3,9333	90,0333	0	133,8500	7,1667
Week 33/2003	3499123	18,1000	8,0167	80,5500	0,3333	186,6500	3,2667
Week 34/2003	3833747	11,8400	3,6200	89,3800	3	154,8200	4,6600

Continues in the next page ...

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 35/2003	3089664	22,5500	10,0833	85,4000	6,1333	238,0500	5,5833
Week 36/2003	3609030	14,0667	4,7333	81,5000	3,5000	190,2000	7,4667
Week 37/2003	3919008	15,7000	4,4333	83,8333	0,6333	151,4500	6,5667
Week 38/2003	2505294	20,4167	9,4167	91,0833	8,8000	196,2833	6,7833
Week 39/2003	2352757	20,7800	7,8400	86,8200	0	139,6600	8
Week 40/2003	2559015	19,5333	9,2333	78,8667	3,3333	198,5500	8,0167
Week 41/2003	2549628	25,8167	11,5667	80,5500	0	223,5000	9,4000
Week 42/2003	2321544	24,6333	10,9000	80,8333	2,1667	176,1500	8,2167
Week 43/2003	2156849	22,1667	9,1333	74,7667	0,6667	178,2167	11,1167
Week 44/2003	2262453	25	11,6667	81,5833	0	154,3667	9,8333
Week 45/2003	2644424	23,0500	11,1500	86,3000	8,7500	225,5833	8,1500
Week 46/2003	2639170	23,1833	12,5667	84,8500	14,5500	249,1667	8,4500
Week 47/2003	2063080	26,2833	13,8833	79,3500	3,4500	177,2333	11,1000
Week 48/2003	2441794	22,6500	12,6667	83,9667	1,9667	172,5833	9,7500
Week 49/2003	2545541	23,3000	11,5667	75,9833	8,6667	199,6500	9,4167
Week 50/2003	2664210	26,5500	13,7500	78,3000	2,9000	196,2333	12,0667
Week 51/2003	2462348	25,5600	15,3800	74,7200	0,7600	203,8000	7,9200
Week 52/2003	2530282	27,0800	11,9400	69,5200	1,0600	204,2600	13,5800
Week 1/2004	2010711	29,7333	19,2167	88,6667	1,4333	142,1500	8,5833
Week 2/2004	2232971	26,4167	14,4167	79,9667	1,2333	162,9333	12,5333
Week 3/2004	2204838	28,0833	18,3833	86,9667	0	151,6000	10,8833
Week 4/2004	1837343	30,6833	18,9167	82,2833	0,2833	159,4000	9,9333
Week 5/2004	2257356	26,4167	16,3333	84,9000	0	173,0500	11,4833
Week 6/2004	2417491	30,4833	18,1500	77,0667	9,0833	182,9167	9,6167
Week 7/2004	2453689	23,7167	15	83,1167	2	219,5333	9,8333
Week 8/2004	2165516	26,7167	14,0667	85,6333	0	145,2333	11
Week 9/2004	2436447	29,1000	14,6000	75,0500	1,9500	173,5500	9,9833
Week 10/2004	3037165	24,9667	13,6167	85,0500	2,0667	149,7667	9,5833
Week 11/2004	2010665	26,1833	15,3500	88,6500	0	124,9167	8,2667
Week 12/2004	1968738	30,9833	14,4167	74,2000	0	147,8167	10,2167
Week 13/2004	2108793	31,2833	19,2667	84,9000	0	182,4500	7,8667
Week 14/2004	2905879	26,7000	17,1000	87,3000	1,0500	141,8500	4,9000
Week 15/2004	2277141	25,2500	15,8833	89,7000	13,3667	152,5000	4,6667
Week 16/2004	2315125	19,2667	11,8500	82,9500	12,4167	206,1167	5,0833
Week 17/2004	2424359	20,3000	8,4600	85,7400	9,6000	154,5200	7,6800
Week 18/2004	3048647	19,2167	11,2167	92,0667	0,2500	132,5000	5,5000
Week 19/2004	3110173	17,8333	10,5667	92,2500	0,2833	152,4167	4,6333
Week 20/2004	4486423	15,2167	5,3667	87,6167	0	146,5333	7,3167
Week 21/2004	3531813	14,7833	8,3000	93,8000	4,9667	189,7167	3,4833
Week 22/2004	4069386	16,3167	7,1333	96,6167	0	98,8833	3,4667

Continues in the next page . . .

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 23/2004	3812879	15,3833	5,5667	89,8500	0	205,0333	4,6333
Week 24/2004	3771381	17,5167	4,3167	90,2000	0	170,3500	4,5833
Week 25/2004	3113831	15,6167	6,1500	98,1833	10,7833	107,8833	2,3833
Week 26/2004	3353187	16,7667	9,6000	96,3667	1,0833	172,1500	3,0500
Week 27/2004	4018478	12,8333	3,5667	84,5000	1,2500	205,4667	5,8833
Week 28/2004	4577138	11,8667	3,4667	91,7833	0	115	4,8333
Week 29/2004	3643350	19,2000	10,0833	86,5833	0,8333	246,3167	5,8333
Week 30/2004	2274765	17,3167	8,0167	91,1000	5,7667	157,3167	5,3167
Week 31/2004	3330015	16,5833	9,8833	92,4167	2,9167	216,1833	3,4667
Week 32/2004	3936423	20,3500	7,1833	85,2333	0	190,9333	8,8500
Week 33/2004	2541321	14,3667	11,2833	99,7333	5,4833	209,7333	0
Week 34/2004	3289351	18,5000	4,8600	82,6800	1,1200	148,3600	8,7000
Week 35/2004	2708966	19,7167	9,1500	86,3333	0,8333	224,7333	5,8667
Week 36/2004	2934089	15,4600	8,1800	87,7200	6,8000	164,0800	3,9600
Week 37/2004	3589900	19,9500	6,6333	83,6833	0	157,4333	8,0500
Week 38/2004	2510938	20,2667	7,6833	79,0833	0,3333	177,0500	9,0833
Week 39/2004	2315608	18,9167	8,2833	81,8833	0	188	7,3167
Week 40/2004	2305684	20,8833	8,3333	84,7333	3,5000	182,2000	8,2000
Week 41/2004	2744836	20,5800	12	86,2600	17,5000	198,4400	6,8800
Week 42/2004	2531931	22,1500	9,3333	78,0333	7,3667	178,9000	9,7667
Week 43/2004	2258744	21	9,6667	79,7333	0,4667	169,7000	5,6333
Week 44/2004	2054699	20,8400	12,6800	89,2000	11,2000	231,9800	6,5800
Week 45/2004	2657164	19,4333	12,1500	91,3167	4,1500	218,4500	3,9333
Week 46/2004	2687454	20,6667	11,4500	86,6167	1,2500	185,0500	8,5333
Week 47/2004	2112851	25,8500	15,7500	90,4333	3,5833	131,6167	7,4333
Week 48/2004	2070644	29,4500	16,4333	80,5333	0	182,7167	11,1500
Week 49/2004	2164784	26,5000	15,0167	81,8833	2,8833	200,2333	10,8833
Week 50/2004	2248424	28,7833	15,4000	75,7500	1,3333	197,2167	11,4167
Week 51/2004	2200349	27,7200	12,6000	68,7000	0	195,2800	12,6000
Week 52/2004	2411323	28,1400	16,6400	84,6000	0,3800	165,7600	10,2000
Week 1/2005	1999391	33,6167	21,0333	84,3500	1,9167	164,7333	9,9500
Week 2/2005	2178662	29,7333	16,5000	75,1667	1,2500	197,9500	10,5667
Week 3/2005	2258625	28,7667	13,3667	68,8833	2,0333	182,3000	12,7167
Week 4/2005	2045641	29,5833	14,2833	63,6000	0	208,0167	12,1000
Week 5/2005	2107681	24,0333	12,4000	81,9167	25	215,6667	10,4167
Week 6/2005	2150651	26,1000	19,3167	95,2167	6,7667	199,5667	4,4000
Week 7/2005	2383880	27,5000	15,6167	86,3833	0	133,7000	10,3333
Week 8/2005	2178316	29,8833	18,7833	91,6000	5	101,0833	6,5333
Week 9/2005	1514920	26,3200	13,9200	81,7600	1,2000	153,0800	9,2600
Week 10/2005	2213110	24,9333	15,0833	84,5667	3,4833	142,7667	5,9333

Continues in the next page . . .

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 11/2005	2533642	25,3667	15,5333	90,1167	0,3667	127,1667	7,8333
Week 12/2005	2024704	23,5400	12,3000	82,2800	0,5600	130,7400	8,0200
Week 13/2005	2400506	22,2833	14,2167	89,6667	7,3833	169,9000	4,0500
Week 14/2005	2191309	23,9000	10,6167	83,5500	17,4667	135,9833	7,8667
Week 15/2005	2552563	20,1333	11,2667	93,2333	12,2667	146,4667	5,7000
Week 16/2005	2224823	23,2000	9,1000	86,4833	0,1667	122,3333	9,2000
Week 17/2005	2743075	18,1333	5,5333	87,7167	0,7333	107	8,8667
Week 18/2005	2475998	21,4833	11	85,6667	4,4667	161,8667	6,5667
Week 19/2005	2990511	19,1667	10,6833	95,3167	2,1000	132,3667	2,1500
Week 20/2005	2638878	16,3833	7	88,8000	1,7167	116	5,2000
Week 21/2005	2950930	18,0833	6,2500	89,3667	0	161,1667	6,7500
Week 22/2005	2469329	20,4167	11,4333	96,7000	20,2667	194,1500	3,9000
Week 23/2005	2730435	20,5667	12,0167	97,2167	3,3500	175,2500	3,8167
Week 24/2005	2975941	14,9667	10,3000	96,4500	6,8833	186,6500	1,7667
Week 25/2005	3505573	14,9333	4,6333	90,6000	0	84,1333	7,5167
Week 26/2005	2703008	19,4000	10,9000	97,3167	8,8000	116,7000	1,4000
Week 27/2005	3405088	14,1167	4,5000	88,2833	0,1000	143,0167	6,3000
semana 29/2005	3223184	12,3600	3,5200	94,8000	0,4000	161,3200	3,9200
Week 30/2005	3312664	17,0333	7,5667	95,4500	0,2167	145,5667	4,0667
Week 31/2005	2260459	21,6833	10,3833	92,1500	0	230,0167	4,5000
Week 32/2005	3085029	17,4333	4,0500	89,3833	0,3333	104,7333	8,4833
Week 33/2005	2461010	17,7500	6,4167	92,5333	1,5333	164,3833	6,0167
Week 34/2005	2252468	17,9800	8,5400	93,5000	6,7200	282,8200	2,9600
Week 35/2005	2960554	13,0833	5,8333	91,9833	2,5333	185,4333	4,1333
Week 36/2005	3057640	19,3333	6,3500	89,8167	6,8667	190,9500	6,8167
Week 37/2005	3256333	13,0333	3,0667	89,7833	2,4333	155,9000	6,4500
Week 38/2005	2141784	22,2000	13,5500	92,5167	0,4500	165,9500	5,0333
Week 39/2005	1718958	20,5500	6,9667	80,2667	4,0500	178	9,5500
Week 40/2005	1868168	18,3500	7,4167	86,3500	0,6667	173,3167	7,4500
Week 41/2005	2214655	22,3167	8,5000	82,4667	0	127,7333	10,6500
Week 42/2005	1790966	26,7333	11,3833	74,4500	0	146,0333	10,0333
Week 43/2005	1681442	18,2000	8,7500	87,2833	5,9333	164,4333	6,6167
Week 44/2005	1644086	22,8000	8,9000	75,0800	1,9000	155,4200	9,1200
Week 45/2005	1965016	22,7500	8,3167	75,3333	0	171,7500	12,4000
Week 46/2005	1938285	27,4500	14,8500	77,3500	2,2167	159,4667	10,7167
semana 47/2005	1864924	29,9500	15,0167	65,9167	0,0167	179,4500	10,7000
Week 48/2005	1675378	25,3000	12,6500	75,9667	1,7500	196,4333	7,8667
Week 49/2005	2074166	22,7167	11,1167	72,2333	0,2333	202,0167	10,4500
Week 50/2005	2180600	27,6833	14,8167	80,7833	1,3500	188,5500	9,0667
Week 51/2005	2103779	26,5000	12,7833	76,0333	0,4500	184,4667	11,0333

Continues in the next page . . .

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 52/2005	1935984	31,8800	14,9200	66,9200	0	172,6800	12,2600
Week 1/2006	1679207	30,4400	14,3800	76,4600	0,9000	154,6200	9,2600
Week 2/2006	1989111	27,2833	17,9000	88,3500	9,8833	235,4833	3,8833
Week 3/2006	2095454	26,1500	14,7833	86,4333	6,0500	163,8833	9,2000
Week 4/2006	1699000	29,7833	15,3333	78,0333	1	149,8000	11,1167
Week 5/2006	1627846	27,4667	15,9833	89,0667	2,5500	149,4500	7,3333
Week 6/2006	2052770	25	12,5833	81,8333	0	140,1500	10,6333
Week 7/2006	1995528	31,4333	16,3167	79,3167	0	141,0167	10,4667
Week 8/2006	1851658	27,9667	18,3333	86,6667	6,1833	164,8000	6,8167
Week 9/2006	1713908	25,5667	14,1667	87,9333	6,7500	150,1833	7,3333
Week 10/2006	2126979	25,8500	13,4000	80,7167	0	128,3500	8,6500
Week 11/2006	2164678	25,1333	17,1833	93,1500	19,6667	114,9500	3,8333
Week 12/2006	1816436	25,3667	14,5167	87,4500	0	144,4000	8,5000
Week 13/2006	1951619	23,3000	10,9667	86,1833	0,8000	109,0500	8,1167
Week 14/2006	2041524	25,7333	14	93,7833	0	111,5500	6,2500
Week 15/2006	2038337	23,4600	11,4200	89,2400	0	135,1000	8,6400
Week 16/2006	2693360	20,6333	8,7333	83,4833	4,6333	168,7667	5,6833
Week 17/2006	2435101	23,8167	10,1667	90,3833	0,1333	123,4833	6,0500
Week 18/2006	2061311	18,9800	8,8200	93,7400	0,2800	99,0600	5,7400
Week 19/2006	2747744	20,2667	7,7500	90,8500	0	95,7000	7,6833
Week 20/2006	2497153	18,5167	7,1333	89,1667	1,4333	113,3000	6,1667
Week 21/2006	3246319	17,7333	7,5667	89,6333	0,1667	139,0167	6,4500
Week 22/2006	2622512	17,0167	7,1333	93,3833	0	151,9000	4,1167
Week 23/2006	3208380	19,8000	8,2500	90,7333	11,6000	157,8000	4,5833
Week 24/2006	3357450	14,2500	4,0167	93,8167	0,1667	119,3167	5,6500
Week 25/2006	3111614	18,3667	8,2667	87,6500	1,8833	140,3500	5,4833
Week 26/2006	3284742	14,5167	7,3500	96,4500	2,3000	447,3333	3,3667
Week 27/2006	2853060	19,2217	17,8883	96,1833	0,1667	473,6948	2,4333
Week 28/2006	3078257	17,4800	17,4800	92,5000	0,3333	1287,5128	4,1500
Week 29/2006	2632276	20,9160	20,9160	84,6000	0,2000	332,9360	5,6800
Week 30/2006	2444469	15,1967	15,1967	93,1667	0,1667	3051,3657	4,2333
Week 31/2006	4079808	13,8450	3,1950	91,5167	0	422,6833	3,7500
Week 32/2006	3612956	15,4833	7,3000	95,1667	5,6833	136,3833	2,5667
Week 33/2006	3064508	14,3000	5,3500	92,5000	0	156,7167	5,2500
Week 34/2006	2800561	19,9400	6,5600	83,5400	0,2400	168,3800	7,1600
Week 35/2006	2280058	17,5700	10,1950	86,9250	0	1073,7750	4,4750
Week 36/2006	3537141	16,1417	16,1417	83,6667	0,1667	420,6677	8,4000
Week 37/2006	2741801	20,2617	20,2617	82	0,1667	243,9655	8,0667
Week 38/2006	1587098	22,4517	22,4517	79,6667	0	326,0790	8,1167
Week 39/2006	1904643	21,0200	21,0200	81,3333	0	1349,2655	7,1833

Continues in the next page . . .

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 40/2006	2120507	21,8850	21,8850	88	0,3333	936,6892	6,8000
Week 44/2006	1751901	20,8800	19,6400	84,8400	0,2000	168,9000	8,9800
Week 45/2006	2431799	20,8000	20,8000	76,3333	0,1667	176,8833	11,5333
Week 46/2006	2280129	25,7000	25,7000	82	0,3333	191,9333	8,3333
Week 47/2006	1876845	28,2000	28,2000	78,3333	0,1667	156,5167	11,1000
Week 48/2006	1735020	26,4500	25,5000	81,7500	0,5333	162,0667	9,8333
Week 49/2006	2192247	26,9667	26,9667	79,3333	0,1667	166,1667	11,3000
Week 50/2006	2335422	31,3667	31,3667	81,6667	0,5000	173,6833	7,2333
Week 51/2006	2086710	28,6667	28,6667	87,6667	0,3333	159,7000	8,9833
Week 52/2006	2153375	29,4800	29,4800	80,6000	0,4000	184,4000	7,7200
Week 53/2006	1542989	27,5000	27,5000	92,8000	0,2000	176,0600	7,4400
Week 1/2007	1940214	30,5167	30,5167	83	0	184,5000	10,1500
Week 2/2007	1835315	27,9500	27,9500	71,1667	0	169,3167	9,7333
Week 3/2007	1561202	29,5833	29,5833	86,3333	0,1667	161,2500	6,4333
Week 4/2007	1648127	30,3000	30,3000	75,8000	0	145,3400	12,2000
Week 5/2007	2288112	29,6000	29,6000	80,1667	0,5000	177,4000	8,2167
Week 6/2007	2340295	29,7333	29,7333	84	0,1667	217,7333	7,4167
Week 7/2007	1856677	29,5600	29,5600	84	0,4000	173,8800	9,5800
Week 8/2007	2178414	26,5000	26,5000	91,4000	0,8000	132,9200	3,8800
Week 9/2007	2387049	25,8500	25,8500	92,5000	0,5000	151,0833	6,0333
Week 10/2007	1839173	27,1200	27,1200	92,2000	0,6000	126,8800	7,1800
Week 11/2007	2080375	27,1000	27,1000	87,6667	0,1667	112,4000	8,6000
Week 12/2007	1821560	23,3000	23,3000	98,8333	1	126,2667	1,5167
Week 13/2007	1867146	25,3333	25,3333	92,5000	0,1667	108,3167	8,4417
Week 14/2007	2249850	23,3833	23,3833	86,8333	0,1667	149,8833	6,4500
Week 15/2007	2043826	25,5500	25,5500	96	0,1667	164,3167	3,7500
Week 16/2007	2209278	19,3833	19,3833	90	0,5000	109,5333	6,4667
Week 17/2007	2612835	21,9200	19,3000	95,5400	0,4000	151,9400	5,5200
Week 18/2007	3913849	15,4667	15,4667	84,3333	0,3333	193,6333	7,3333
Week 19/2007	3035823	17,8857	17,8857	86,5714	0,1429	95,2200	5,8286
semana 20/2007	3363584	14,6800	14,6800	90,8000	0	161,1600	7,7200
Week 21/2007	4137460	12,4167	12,4167	91,8333	0,1667	176,6833	4,3500
Week 22/2007	2104115	17,6000	17,6000	88	0	80,3000	6,5750
Week 23/2007	3210528	14,0429	14,0429	95,8571	0,1429	143,3571	4,0143
Week 24/2007	3366970	14,4000	14,4000	93,8571	0,2857	147,3286	4,4857
Week 25/2007	2916113	14,0286	14,0286	88,1429	0	94,3857	7,9143
Week 26/2007	5561243	15,5833	15,5833	90,8333	0	113,2000	4,1000
Week 27/2007	5398958	10,4333	10,4333	91	0,3333	149,3167	4,1833
Week 28/2007	4797002	16,1714	16,1714	79,5714	0	136,5714	5,3857
Week 29/2007	4562853	10,6857	10,6857	88,1429	0,2857	179,3714	5,4571

Continues in the next page . . .

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 30/2007	4577326	11,4167	11,4167	94,3333	0,3333	162,4667	2,5833
Week 31/2007	4512109	14,3500	14,3500	87,3333	0,1667	172,4833	5,5000
Week 32/2007	3466389	15,4000	15,4000	94,1667	0,6667	182,1333	4,9000
Week 33/2007	3484006	14,4000	14,4000	92,6667	0	159,6167	7,0833
Week 34/2007	3147108	16,3333	16,3333	87,5000	0,1667	146,0500	7,2000
Week 35/2007	2309977	24,6000	24,6000	91,8333	0,1667	171,4167	4,2000
Week 36/2007	2126661	20,6833	20,6833	97,8333	0,6667	177,3333	0,9667
Week 37/2007	2533570	16,5833	16,5833	93,8333	0,3333	194,5833	4,6500
Week 38/2007	2429710	19,5500	19,5500	84,1667	0	166,9000	9,3833
Week 39/2007	2197202	22,3500	22,3500	97,3333	0,6667	144,8667	3,2167
Week 40/2007	2551922	17,3333	17,3333	96,1667	0,8333	197,5333	2,9500
Week 41/2007	2108353	23,2200	23,2200	89	0,2000	162,5600	9,4400
Week 42/2007	1789974	26,7000	26,7000	77,8333	0,1667	180,2500	10,8167
Week 43/2007	1655688	25,6000	25,6000	80,6000	0,2000	234,5000	9,1000
Week 44/2007	2097765	23,2667	23,2667	81,3333	0,5000	215,7833	7,1167
Week 45/2007	2360214	20,8000	20,8000	81,3333	0,8333	230,9667	9,1333
Week 46/2007	2152498	24,2333	24,2333	77,8333	0,5000	202,9333	9,7167
Week 47/2007	1725665	27,8167	27,8167	77,3333	0	157,6667	12,7833
Week 48/2007	1749327	28,1500	28,1500	71,8333	0,1667	185,1000	11,1833
Week 49/2007	2021910	27,2833	27,2833	70,6667	0,1667	193,6500	11,0333
Week 50/2007	2516246	27,9833	27,9833	75,6667	0	191,0667	12,1833
Week 51/2007	2004826	26,6200	26,6200	93,4000	0,2000	168,8400	5
Week 52/2007	1786796	30,5600	30,5600	77,4000	0,4000	186,9600	11,0200
Week 1/2008	1616077	30,0167	30,0167	75	0,3333	165,8167	9,3167
Week 2/2008	1896702	28,1833	28,1833	80,1667	0,1667	168,5333	12,0667
Week 3/2008	1786087	29,4667	29,4667	74,8333	0	219,3500	10,1167
Week 4/2008	1924555	29,5667	29,5667	86,5000	0,5000	177,8000	9,7000
Week 5/2008	1826086	30,4333	30,4333	83	0,1667	167,0333	9,8000
Week 6/2008	2171090	28,4333	28,4333	84,6667	0,5000	153,3833	10,7167
Week 7/2008	1834858	29,4500	29,4500	87,3333	0	148,0333	7,1417
Week 8/2008	1847698	26,6667	26,6667	95,6667	0,6667	170,8033	4,7167
Week 9/2008	3119016	24,1500	24,1500	97,1667	0,8333	123,8500	3,5833
Week 10/2008	2537237	24,0500	24,0500	85,3333	0	98,5950	10,0833
Week 11/2008	2449622	27,4250	27,4250	88	0,5000	139,0250	8,6000
Week 12/2008	2477690	25,8000	25,8000	88	0,1667	107,9733	7,3000
Week 13/2008	2002338	23,6833	23,6833	90,6667	0,3333	106	7,9167
Week 15/2008	3437607	22,8333	22,8333	71,5000	0,1667	121,0583	9,0833
Week 16/2008	2443131	24,8333	24,8333	87,3333	0	97,7500	8,1333
Week 17/2008	3181962	17,6000	17,6000	87	0,4000	154,2000	7,7800
Week 18/2008	3588418	19,1833	19,1833	86,8333	0	90,0667	8,2167

Continues in the next page . . .

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 19/2008	3366375	22,9667	22,9667	84,8333	0	140,6867	6,6333
Week 20/2008	2453671	21,8500	21,8500	98,1667	0,8333	203,8583	2,8833
Week 21/2008	3951939	13,9333	13,9333	91	0	159,5967	4,6833
Week 22/2008	5283446	16,8167	16,8167	92,3333	0,1667	109,7817	5,3833
Week 23/2008	4321461	15,2000	15,2000	80,1667	0,1667	231,0917	6,7333
Week 24/2008	4537587	14,2000	14,2000	89,1667	0,3333	234,5317	4,8667
Week 25/2008	4451132	15,8500	15,8500	97,8333	0,1667	77,2150	4,0667
Week 26/2008	4568490	16,8500	16,8500	95,3333	0,1667	155,9267	3,7667
Week 27/2008	3629043	20,4167	20,4167	96,3333	0,3333	152,3950	4,5833
Week 28/2008	2407967	23,1800	23,1800	86,6000	0	185,7020	7,4200
Week 29/2008	3666894	13,3833	13,3833	95,8333	0,3333	178,8833	3,3833
Week 30/2008	4029739	15,9167	15,9167	98,6667	0	149,1500	3,4667
Week 31/2008	4773272	15,2333	15,2333	82,1667	0,1667	143,8000	7,5500
Week 32/2008	4076976	18,3667	18,3667	93,5000	0,1667	196,5833	5,0667
Week 33/2008	3628467	15,8400	15,8400	86,8000	0	185,9000	7,6800
Week 34/2008	2473349	18,1400	18,1400	90,4000	0,4000	236,2800	6,0600
Week 35/2008	2826566	15,4167	15,4167	89,6667	0,1667	227,1833	3,7500
Week 36/2008	4360510	15,9167	15,9167	91,8333	0	230,1500	23,9167
Week 37/2008	4479319	17,6000	17,6000	92,6667	0	185,4500	5,5000
Week 38/2008	2749200	21,6667	21,6667	88,8333	0	190,1167	9,1000
Week 39/2008	2215935	19,5167	19,5167	91,5000	0,1667	138,5000	3,7000
Week 40/2008	2769579	22,1167	22,1167	77,8333	0	189,1667	9,7500
Week 41/2008	2659052	19,9000	19,9000	91,1667	0,3333	160,4000	5,8167
Week 42/2008	1955810	25,2833	25,2833	82	0,1667	138,4333	8,6417
Week 43/2008	1704084	22,7333	22,7333	81,3333	0,1667	195,4833	11,4000
Week 44/2008	2498404	30,2833	30,2833	77,5000	0	178,4000	9,9333
Week 45/2008	2793180	27,8500	27,8500	73,1667	0	197,0500	10,4000
Week 46/2008	2303739	26,7333	26,7333	69,8333	0	227,7000	12,1833
Week 47/2008	1853996	34,4667	34,4667	73,1667	0	213,8500	6,5333
Week 48/2008	1913062	24,9833	24,9833	86,5000	0,5000	178,2667	6,4833
Week 49/2008	3186749	29,3333	29,3333	78	0	203,3833	8,8833
Week 50/2008	2284180	31,5833	31,5833	72,3333	0	228,6000	12,6500
Week 51/2008	2302637	28,9800	28,9800	80,4000	0	197,7000	8,4600
Week 52/2008	1917573	25,4400	25,4400	70,8000	0	180,6200	10,7400
Week 1/2009	2106193	32,2000	32,2000	57,8333	0	221,1833	12,3500
Week 2/2009	2634985	31,1833	31,1833	78	0,3333	197,3833	9,3667
Week 3/2009	2351758	32,5000	32,5000	61,3333	0	238,9333	12,7667
Week 4/2009	2064283	25,3333	25,3333	94,1667	0,5000	282,0833	4,6333
Week 5/2009	2491609	27,9667	27,9667	79	0,1667	185,3000	8,8667
Week 6/2009	3119109	29,6333	29,6333	79,1667	0,1667	188,8000	10,4167

Continues in the next page ...

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 7/2009	2547601	30,7333	30,7333	90	0,3333	161,5667	7,8833
Week 8/2009	1839200	27,5400	27,5400	87	0	147,2000	9,5000
Week 9/2009	2466054	25,7833	25,7833	95,6667	0,5000	124,9667	5,5167
Week 10/2009	2911494	26,3000	26,3000	93,5000	0,5000	241,2000	7,1333
Week 11/2009	2527839	26,2667	26,2667	83,5000	0,1667	146,2500	9,1417
Week 12/2009	2355366	27,9333	27,9333	91,3333	0	104,1667	7,7167
Week 13/2009	2543897	22,9500	22,9500	89,6667	0,3333	143,0833	6,3417
Week 14/2009	2941673	25,6600	25,6600	83	0	109,6600	9,6100
Week 15/2009	3077187	23,9333	23,9333	85,8333	0,1667	148,4333	8,2667
Week 16/2009	2978107	23	23	90	0,1667	120,0667	7,3667
Week 17/2009	2138566	24,9400	24,9400	84,4000	0	132,3000	9,0800
Week 18/2009	3201548	22,5167	22,5167	83,3333	0	129,0500	7,3000
Week 19/2009	4102988	16,4833	16,4833	94,8333	0,6667	203,8000	3
Week 20/2009	4007709	23,4000	23,4000	95,5000	0,1667	143,1000	5,3000
Week 21/2009	3273680	15,8500	15,8500	92,1667	0,1667	120,1500	5,2333
Week 22/2009	5617576	13,5333	13,5333	89,8333	0,1667	243,0500	4,6667
Week 23/2009	5180040	13,6667	13,6667	96,5000	0	89,9000	4,9167
Week 24/2009	4365749	18,0167	18,0167	94,5000	0,3333	146,6500	4,4167
Week 25/2009	4921847	12,9833	12,9833	91,3333	0,1667	119,2167	6,8333
Week 26/2009	5557854	15,5833	15,5833	90,3333	0,1667	195,3833	4,9333
Week 27/2009	5006407	13,7333	13,7333	92,5000	0,1667	127,7833	4,6667
Week 28/2009	5826676	12,6167	12,6167	96,8333	0,6667	155,6333	3,6500
Week 29/2009	5705439	12,2000	12,2000	91,5000	0,5000	266,6167	5,0833
Week 30/2009	6227846	12,3500	12,3500	89,3333	0	119,8167	5,7667
Week 31/2009	4940653	15,5167	15,5167	94,1667	0	133,4833	5,7167
Week 32/2009	4406095	23,0333	23,0333	80,6667	0,1667	240,6833	5,5333
Week 33/2009	3512060	15,1333	15,1333	89,3333	0,1667	130,1167	5,6333
Week 34/2009	2787517	26,3200	26,3200	85,6000	0	210,8800	7,9800
Week 35/2009	2909381	15,7667	15,7667	98	0,8333	228,9667	1,0667
Week 36/2009	4179106	14,5667	14,5667	88,5000	0,3333	173,4333	5,9167
Week 37/2009	3029413	18,1000	18,1000	96,3333	0,3333	220,0667	4,3833
Week 38/2009	3330570	18,9833	18,9833	87	0	163,8500	8,1000
Week 39/2009	3435229	15,9667	15,9667	86,1667	0,1667	152,1667	7,9333
Week 40/2009	3046156	18,8833	18,8833	85,8333	0,1667	188,3000	8,7333
Week 41/2009	3193715	18,4167	18,4167	81,1667	0,1667	163,5167	8,3000
Week 42/2009	2741613	22,2500	22,2500	87,1667	0,3333	152,1167	7,1500
Week 43/2009	2663604	24,4000	24,4000	85,3333	0,3333	186,9500	7,0333
Week 44/2009	2307745	21,5333	21,5333	89,6667	0,3333	178,6833	5
Week 45/2009	3139445	25,9833	25,9833	79,3333	0,3333	157,2667	10,2833
Week 46/2009	2556855	23,6833	23,6833	91	0,5000	201,4833	5,4167

Continues in the next page . . .

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 47/2009	2313595	26,3000	26,3000	93,1667	0,5000	163,8000	6,6333
Week 48/2009	2356704	22,4667	22,4667	83,8333	0,3333	158,6833	8,8500
Week 49/2009	2952435	24,8667	24,8667	86,1667	0,1667	167,0667	9,2000
Week 50/2009	2752478	27,8167	27,8167	87	0,1667	177,4833	8,8667
Week 51/2009	2340506	26,8800	26,8800	96,2000	0,6000	151,4400	4,8400
Week 52/2009	2304976	28,1400	28,1400	84,2000	0,2000	140,0400	7,6800
Week 1/2010	2460717	28,5000	28,5000	84,3333	0,1667	131,2667	9,7500
Week 2/2010	2499677	28,5833	28,5833	78,6667	0,1667	191,9167	9,4333
Week 3/2010	2323663	29,7833	29,7833	87	0,3333	165,4667	9,4500
Week 4/2010	2185857	31,5167	31,5167	86,5000	0	150,5333	11,2000
Week 5/2010	2241975	28,6000	28,6000	97,1667	0,6667	136,2167	3,1500
Week 6/2010	2966363	31,6333	31,6333	85,8333	0	150,0500	9,9333
Week 7/2010	2231558	27,7167	27,7167	91,5000	0,6667	196,5000	5,8500
Week 8/2010	2789168	24,2000	24,2000	87,6667	0,3333	172,8500	8,2167
Week 9/2010	2207965	28,2400	28,2400	93,2000	0,2000	121,7800	7,3800
Week 10/2010	2859274	26,4000	26,4000	88	0	234,4000	7,6000
Week 11/2010	2801928	25,7667	25,7667	73,3333	0,1667	376,8500	7,9500
Week 12/2010	2577976	26,2833	26,2833	77,8333	0	296,0667	10
Week 13/2010	2308383	27,4750	27,4750	90,7500	0	95,3500	6,4750
Week 14/2010	3098129	23,3333	23,3333	87	0	106,2667	8,8667
Week 15/2010	2948625	21,4833	21,4833	91,5000	0,6667	151,5833	3,8000
Week 16/2010	2904667	20,3000	20,3000	91	0	80,2600	7,2800
Week 17/2010	3569217	21,3600	21,3600	86,8000	0	128,7400	9,1200
Week 18/2010	3365704	19,4167	19,4167	90,1667	0	124,1167	5,1167
Week 19/2010	4336038	20,9000	20,9000	83	0	141,9000	6,7167
Week 20/2010	3293971	20,2000	20,2000	85	0	186,6667	1,8167
Week 21/2010	3093574	18,5667	18,5667	89,8333	0,5000	224,5167	3,2500
Week 22/2010	4112841	17,2333	17,2333	79,5000	0	95,9667	6,1500
Week 23/2010	4950567	15,1667	15,1667	80,5000	0,1667	148,1000	6,0833
Week 24/2010	4614168	16,6500	16,6500	87	0,3333	143,6500	4,6167
Week 25/2010	4960458	16,3833	16,3833	71,5000	0,1667	178,2833	5,8667
Week 26/2010	4500692	18,7167	18,7167	84,3333	0	139,7000	4,5333
Week 27/2010	3904511	16,1333	16,1333	88,1667	0,3333	165,6833	3,6667
Week 28/2010	6531135	9,6000	9,6000	75,5000	0,6667	182,6167	5,1167
Week 29/2010	5905227	14,1500	14,1500	81,8333	0,1667	145,4167	5,7333
Week 30/2010	5989815	14,8714	14,8714	80,1429	0,5714	188,0857	5,1143
Week 31/2010	6636083	12,1000	12,1000	78,5000	0,3333	110,4500	6,6000
Week 32/2010	5817367	14,5833	14,5833	78,6667	0,6667	169,3000	4,9000
Week 33/2010	4676220	20,1000	20,1000	71,5000	0,1667	145,2833	7,0667
Week 34/2010	3080303	19,6200	19,6200	82	0	136,4000	4,8800

Continues in the next page . . .

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 35/2010	4581311	15,2833	15,2833	86,5000	1	258,1833	1,2333
Week 36/2010	4083486	22,2333	22,2333	75,8333	0,1667	156,3667	8,2167
Week 37/2010	3966731	16,8833	16,8833	74,8333	0,3333	208,1833	5,8333
Week 38/2010	2908668	21,4000	21,4000	72,8333	0	156,2667	7,9667
Week 39/2010	2979651	17,4500	17,4500	75,1667	0,3333	252,9000	4,5500
Week 40/2010	2908662	22,1600	22,1600	69	0	128,0600	8,6400
Week 41/2010	3248016	21,2333	21,2333	76,1667	0,5000	177,8500	6,2833
Week 42/2010	1375272	21,9667	21,9667	70,1667	0	133,6167	9,4000
Week 43/2010	4245950	20,1833	20,1833	72,1667	0,1667	189,8833	8,8833
Week 44/2010	2628521	24,3400	24,3400	70,4000	0,2000	141,1400	8,8200
Week 45/2010	3126566	19,7500	19,7500	69,8333	0,3333	165,4000	9,8000
Week 46/2010	2778568	24,1167	24,1167	69,6667	0,5000	191,0167	8,6833
Week 47/2010	2199673	26,8833	26,8833	69,3333	0	152,4000	8,2000
Week 48/2010	2371753	27,9333	27,9333	52,3333	0	177,3667	11,8167
Week 49/2010	2556968	29,7833	29,7833	53,6667	0,3333	207,4000	10,3500
Week 50/2010	2931188	29,0333	29,0333	56,5000	0	215,6833	10,1167
Week 51/2010	2350386	33,2400	33,2400	54,6000	0	193,4000	13,0400
Week 52/2010	2352637	31,8200	31,8200	58,6000	0	179,7800	12,0200
Week 1/2011	2558949	30,3333	30,3333	65,8333	0,1667	141,6167	8,3500
Week 2/2011	2131942	30,4000	30,4000	66,3333	0,1667	181,5833	11,3167
Week 3/2011	2473954	29,7833	29,7833	54,3333	0,1667	192,5500	11,3333
Week 4/2011	2249915	32,1667	32,1667	60,5000	0	175,9167	10,2167
Week 5/2011	2156809	30,4500	30,4500	58,1667	0,1667	210,3500	10,9000
Week 6/2011	3050521	26,3833	26,3833	69,1667	0,3333	184,9500	6,4667
Week 7/2011	2573061	30,9500	30,9500	58,8333	0,3333	178,0333	8,6333
Week 8/2011	2611030	26,1167	26,1167	74	0,5000	184,5333	6,9167
Week 9/2011	2101574	29,0833	29,0833	64,8333	0	177,3833	10,8833
Week 10/2011	867459	31,4500	31,4500	66,5000	0,5000	162,7250	7,6500
Week 11/2011	929321	24,4833	24,4833	65,8333	0,5000	138,4833	8,4333
Week 12/2011	2198028	26,3333	26,3333	70,3333	0,6667	162,7667	6,0500
Week 13/2011	3925263	26,2500	26,2500	68	0	129,1833	10,1833
Week 14/2011	4659060	22,9667	22,9667	69,5000	0,1667	109,4667	8,7167
Week 15/2011	4054998	24,2000	24,2000	71,8333	0,6667	119,0833	6,9167
Week 16/2011	2482409	19,5200	19,5200	67,6000	0,4000	132,9600	6,1800
Week 17/2011	2169366	24,3500	24,3500	77,2500	0,2500	114,0250	6,7500
Week 18/2011	4560781	18,7833	18,7833	70,6667	0	135,4167	8,5167
Week 19/2011	4671129	20,0833	20,0833	77	0,1667	100,3667	6,0333
Week 20/2011	3446401	21,1667	21,1667	75,1667	0	122,5333	7,7167
Week 21/2011	3868174	15,4667	15,4667	76	0,5000	107,9333	4,1000
Week 22/2011	3676501	15,3000	15,3000	74,6667	0,1667	114,6667	4,4500

Continues in the next page ...

Table 5.4.4 – Continued

Week	Sales	T Max. Aver.	T min. Aver.	Rel. humidity Aver.	Precip. Aver.	wind Aver.	heliophany Aver.
Week 23/2011	5594894	15,2333	15,2333	80	0,1667	107,4833	4,4000
Week 24/2011	5540264	16,4667	16,4667	82,8333	0,8333	187,2000	2,8333
Week 25/2011	4771888	14,3667	14,3667	82,5000	0,5000	170,3000	4,1667
Week 26/2011	5966606	11,9714	11,9714	67,8571	0	219,2000	7,2429
Week 27/2011	6532075	12,9000	12,9000	69,2857	0	170,0143	7,4429
Week 28/2011	5547618	17,8143	17,8143	78,5714	0,2857	150,5571	5,0429
Week 29/2011	5280350	13,7286	13,7286	79	0,2857	173,3286	4,2143
Week 30/2011	4682671	15,9429	15,9429	76,2857	0,4286	188,3857	5,6857
Week 31/2011	5519564	13,3833	13,3833	73,6667	0,3333	185,2167	5,8667
Week 32/2011	5385591	19,0857	19,0857	74,4286	0,1429	215,3571	4,5571
Week 33/2011	4595666	13,7429	13,7429	79,7143	0,4286	247,1714	3,7857
Week 34/2011	5222869	13,1333	13,1333	72,8333	0,5000	176,8500	5,6333
Week 35/2011	4870710	15,4714	15,4714	72,8571	0	205,5000	7,4571
Week 36/2011	4096739	19,7143	19,7143	63,5714	0,1429	181,4000	8,0429
Week 37/2011	3398617	20,6000	20,6000	65,6667	0,3333	214,6167	7,5333
Week 38/2011	3266710	18,0333	18,0333	66,1667	0	186,7667	8,4000
Week 39/2011	2595111	22,4167	22,4167	68,1667	0,3333	185,6667	7,4333
Week 40/2011	3185410	19,5333	19,5333	75,8333	0,3333	156,6833	3,4167
Week 41/2011	3311189	20,6500	20,6500	77	0,3333	178,3333	6,9667
Week 42/2011	2706442	22,0500	22,0500	70	0	181,3667	10,2500
Week 43/2011	2950426	19,7667	19,7667	67,1667	0,3333	200,8333	6,5667
Week 44/2011	2524455	24,3167	24,3167	64,3333	0,1667	203,4667	10,0500
Week 45/2011	981767	26	26	55	0	201,7000	9,2000

Bibliography

[1] Sourceforge, a site dedicated to making open source projects successful.

[2] Alejandra Sansonetti Daniele Afamado, Laura Carratú. Redes neuronales artificiales aplicadas a la predicción de la demanda eléctrica horaria. Master's thesis, Facultad de Ingeniería, UdeLaR.

[3] Ethem Alpaydin. *Introduction to Machine Learning, second edition (Adaptive Computation and Machine Learning series).* MIT Press; 2 edition, 2010.

[4] Yves Baldi, Pierre Chauvin. Assessing the accuracy of prediction algorithms for classificacion: An overview. *Bioinformatics*, 16:412–424, 2000.

[5] David Baum, Eric Haussler. What size net gives valid generalization? *Neural Computation*, 1:151–160, 1989.

[6] Frank Baum, Eric B Wilczek. Supervised learning of probability distributions by neural networks. *Neural Information Processing Systems*, 1988.

[7] Dulce Belaire Franch, Jorge Contreras Bayarri. Recurrence plots in nonlinear time series analysis: free software. Technical report, Universidad de Valencia, 2001.

[8] Thomas Bengtsson. An improved akaike information criterion for state-space model selection. Technical report, Center for atmosferic research, Department of Statistics, University of Missouri-Columbia, 2003.

[9] Mukul Bhattacharya, Rabi Majumdar. *Random Dynamical Systems. Theory and applications.* Cambridge University Press, 2007.

[10] Christopher Bishop. *Neural Networks for pattern recognition.* Clarendon Press, Oxford, 1996.

[11] G. Bishop, G Welch. An introduction to the kalman filter. Technical report, University of North Carolina at Chapel Hill.Department of Computer Science., 2002.

[12] Nino Boccara. *Modelling Complex Systems.* Springer, 2004.

[13] Richard Brocwell, Peter Davis. *Time series: theory and methods.* Springer Verlag, 1987.

[14] Mark Brodley, Carla Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[15] Filippo Castiglioni. Forecasting price increments using an artificial neural network. *Advanced Complex Systems*, 1:1–12, 2000.

[16] S. Chitra, A. Uma. An ensemble model of multiple classifiers for time series prediction. *International Journal of Computer Theory and Engineering*, 2, no. 3:454–458, 2010.

[17] Allan Colombert, Isabelle Ruelland. Models to predict cardiovascular risk: comparison of cart, multilayer perceptron and logistic regression. In *Proceedings of the AMIA Symposium*, 2000.

[18] University of Colorado. Comparison of neural network simulators, Setiembre 2012.

[19] Joy A Cover, Thomas M. Thomas. *Elements of Information Theory.* John Wiley &Sons, 2002.

[20] David Freeman, James Skapura. *Neural Networks. Algorithms, Applications and Programming Techniques.* Addison Wesley, 1992.

[21] Takashi Fueda, Kaoru Yanagawa. Estimating the embedding dimension and delay time from chaotic time series with dynamic noise. *Journal of the Japan Statistic Society*, 31 No. 1, 2001.

[22] Ming Vitányi Paul Gao, Qiong Li. Appliying mdl to learning best model granularity. *Artificial Intelligence*, 121:1–29, 2000.

[23] J Cummings F. Gers, F.A Schmidhuber. Learning to forget: Continual prediction with lstm. *Neural computation*, 12 No.10, 2000.

[24] David Ginsburg, Iris Horn. Combined neural networks for time series analysis. advances in neural information processing systems. *NIPS*93*, 6:224–231, 1994.

[25] M. Gori, M. Maggini. Optimal convergence of on line backpropagation. *IEEE Transactions on Neural Networks*, 7 No. 1:251–253, 1996.

[26] Mohammed Awad Héctor Pomares Ignacio Rojas Osama Salameh Mai Hamdon. Prediction of time series using rbf neural networks: A new approach of clustering. *The International Arab Journal of Information Technology*, 6, 2009.

[27] CityplaceGary Han, Seung-Soo May. Optimization of neural network structure and learning parameters using genetic algorithms. In IEEE Computer Society, editor, *Eigth International Conference on Tools with Artificial Intelligence (ICTAI '96)*, pages 200–206, Toulouse, country-regionFrance., 1996.

[28] Bin Hansen, Mark Yu. Model selection and the principle of minimun description length. *Journal of the American Statistics Association*, 96, 1998.

[29] Mohamad Hassoum. *Fundamentals of artificial Neural networks.* MIT Press, 1995.

[30] Simon Haykin. *Neural Networks. A comprehensive foundation.* Prentice Hall, 1999.

[31] Holger Schreiber Thomas Hegger, Rainer Kantz. Practical implementation of nonlinear time series methods: The tisean package. *CHAOS*, 9:413–435, 1999.

[32] Bert Heskes, Tom M Kappen. *On line learning processes in artificial neural networks.* Elsevier, Amsterdam, 1993.

[33] Víctor Hilera, José Martínez. *Redes Neuronales Artificiales- Fundamentos, modelos y aplicaciones.* Addison Wesley Iberoamericana, 1995.

[34] Jürgen Hochreiter, Sepp Schmidhuber. Long short-term memory. 8 No. 8, 1997.

[35] J.P Huke. Embedding nonlinear dynamical systems: a guide to taken's theorem. *Manchester University*, 2006.

[36] Chuen-Tsai Jang, Jyh-Shing Sun. *Neuro Fuzzy and Soft computing. A computational approach to learning and machine intelligence.* Prentice Hall, 1997.

[37] Brush J. Holzfuss-J. Kadtke, J.B. Global dynamical equations and lyapunov exponents from noisy chaotic time series. *Int. J. Bifurcation Chaos*, 3:607–616., 1993.

[38] V.I. Klyatskin. *Dynamics Of Stochastic Systems.* Elsevier, 2005.

[39] Jordan B. Kolen, John F. Pollack. Back propagation is sensitive to initial conditions-. Technical report, Laboratory for Artificial Intelligence Research. The Ohio University, 1991.

[40] M. Latora, Vito Baranger. Kolmogorov-sinai entropy-rate vs. physical entropy. *Physical Review Letters*, 82, 1999.

[41] John E. Leen, Todd K Moody. Stochastic manhatan learning: An exact time-evolution for the ensemble dynamics. Cambridge University Press, Cambridge, 1999.

[42] D.J. Nightingale C. Linggard, R. Myers. *Neural Networks for vision, speech and natural language.* Chapman & Hall, 1992.

[43] Lennar Ljung. *System identification: Theory for the user.* Prentice Hall, 1995.

[44] David G. Luemberger. *Introduction to dynamic systems: theory, models, and applications.* John Wiley & Sons, 1979.

[45] José Antonio Martins. *Avaliacao de diferentes tecnicas para reconhecimento de fala.* PhD thesis, Universidade Estadual de Campinas, Brasil, 1997.

[46] John Willard Milnor. *Topology from the Differentiable Viewpoint.* Princeton University Press, 1997.

[47] Xiaoyan Pengb Xiaohong Chenc Garba Inoussaa Min Gana, Hui Penga. A locally linear rbf network-based state-dependent ar model for nonlinear time series modeling. In *Information Sciences.* Elsevier, 2010.

[48] John Moody. The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems. *Advances in Neural Information Processing Systems*, 4, 1992.

[49] John Moody. *Prediction risk and architecture selection for neural networks.* Springer Verlag, 1994.

[50] H. Morgan, N. Boulard. *Generalization and parameter estimation in feedforward Nets: some experiments*, pages 630–637. Morgan Kaufmann.

[51] Arthur Motulsky, Harvey Christopoulos. *Fitting models to biological data using linear and nonlinear regression. A practical guide to curve fitting.* Graph Pad Software Inc., 2002.

[52] D.S. Huke J.P. Muldoon, M.R. Broomhead. Delay embedding in the presence of dynamical noise. *Dyn. and Stab. Systems.*, 13, 1998.

[53] Shuji Murata, Noburu Yoshizawa. Network information criterion. determining the number of hidden units for an artificial neural network model. *IEEE Transactions on Neural Networks*, 5:865–872, 1994.

[54] Mark Zhang Shaobo Balasubramanian Vijay Myung, In Pitt. The use of mdl to select among computational models of cognition. *Advances in Neural Information Processing Systems*, 13:38–44, 2001.

[55] NIST/SEMATECH. e-handbook of statistical methods.

[56] Geoffrey Nowlan, Steven Hinton. Simplifying neural networks by soft sharing. *Neural Computation*, 4:473–493, 1992.

[57] National Institute of Science, editor. *On the generalised distance in statistics*, volume 2. National Institute of Science, India, 1936.

[58] Erik Olofsen. The identification of strange attractors using experimental time series. Master's thesis, Twente University, Holanda,, 1991.

[59] Gaurav S. Patel. Modeling nonlinear dynamics with extended kalman filter trained recurrent multilayer perceptrons. Master's thesis, Mc.Master University, 2000.

[60] Barak Pearlmutter. Dynamic recurrent neural networks. Technical report, School of computer Science, Carnegie Mellon University, 1990.

[61] Barak Pearlmutter. *Gradient Descent: Second order momentum and Saturating Error*, pages 887–894. Morgan Kaufmann, 1992.

[62] Michael Peter Perrone. *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization.* PhD thesis, Brown University, 1993.

[63] Adriano Petry. Estudo sobre aplicabilidade da teoria de sistemas dinâmicos não-lineares para o reconhecimento automático de locutor. Master's thesis, Universidade Federal do Rio Grande do Sul, 2000.

[64] I. Popescu. Prediction of outdoor propagation path loss with neural networks. *Informatica*, 10:231–234, 1999.

[65] Juan Antonio Pérez-Ortiz. *Modelos predictivos basados en redes neuronales recurrentes de tiempo discreto.* PhD thesis, Universidad de Alicante, España, 2002.

[66] Juan Antonio Pérez-Ortiz. Kalman filters improve lstm network performance in problems insolvable by traditional recurrent nets. *Neural Networks*, 16, 2003.

[67] Celebi S.. Principe, J. C. Jyh-ming Kuo. An analysis of the gamma memory in dynamic neural networks. *IEEE transactions on neural networks*, 5(2):331–337, 1994.

[68] Neil Lefebvre W. Curt Principe, José Euliano. *Neurosolutions. User's guide.*

[69] Neil Lefebvre W. Curt Principe, José Euliano. *Neural and Adaptive Systems Fundamentals through simulations.* John Wiley & Sons, 2000.

[70] Jyh-Ming Príncipe, José Kuo. Dynamic modeling of chaotic time series with neural networks. In *Neural Information Processing Systems*, pages 311–318, Cambridge, Massachussets, 1995.

[71] Jonas Raudys, Aistis Mockus. Comparison of arma and multilayer perceptron based methods for economic time series forecasting. *Informatica*, 10:231–244, 2001.

[72] Thorsteinn Rögnvaldsson. On langevin updating in multilayer perceptrons. *Neural Computation*, 6, 1994.

[73] Milos Farkas Igor Rosipal, Roman Koska. Prediction of chaotic time-series with a resource-allocating rbf network. *Neural Processing Letters*, Volume 7:185–197, 1998.

[74] R. Cybenko G. Saarinen, S. Bramley. Stateplaceill-conditioning in neural network training problems. *SIAM Journal on Scientific Computing*, 14, 1993.

[75] Y. Sano, M. Sawada. Measurement of the lyapunov spectrum from a chaotic time series. *Physical Review Letters*, 55:1082–1085, 1985.

[76] Yadri Pal Mahesh Sattari, Taghi Yuredki. Performance evolution of artificial neural networks approaches in reservoir inflow. *Journal of Applied Mathematical Modelling*, 36:2649–2657, 2012.

[77] Georg Dockner Engelbert J. Schittenkopf, Christian Dorffner. On nonlinear, stochastic dynamics in economic and financial time series. *Studies in Nonlinear Dynamics & Econometrics*, 4, 2000.

[78] Anan R. Shastri. *Elements of Differential Topology.* CRC Press, 2011.

[79] Michael Small. Estimating the distribution of dynamic invariants: Illustrated with an application to human photo-plethysmographic time series. *Nonlinear Sciences*, 2003.

[80] D.S. Davies M.E. Huke J. Stark, J. Broomhead. Delay embeddings of forced systems: Ii stochastic forcing. *Journal of Nonlinear Science*, 1999.

[81] Kevin Swingler. *Applying Neural Networks: A practical guide.* Academic Press, 1996.

[82] Luna Christie Tjung. Comparison study on neural network and ordinary least squares model to stocks' prices forecasting. *Academy of Information and Management Sciences Journal*, 1 issue 1(35):1, January 2012.

[83] Jose. Wang, Chuang Principe. Training neural networks with additive noise in the desired signal. *IEEE Transactions on Neural Networks*, 10:1511–1517, 1999.

[84] Santosh Judd Stephen Wang, Changfeng Venkatesh. Optimal stopping and effective complexity in learning. *Advances in Neural Information Processing Systems*, 6:303–310, 1994.

[85] David Williams, Ronald Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, pages 270–280, 1989.

[86] Ronald Williams. Some observations on the use of the extended kalman filter as a recurrent network learning algorithm. Technical report, Boston Northeastern University, College of Computer Science,, 1992.

[87] John Wittner, Ben Denker. Strategies for teaching layered networks classification tasks. Technical report, American Institute of Physics, 1998.

[88] Swift J. B. Swinney BL. Vastano J.A. "Determining Lyapunov exponents from a time series" Physica D 16 pp.285-317 Wolf, A. Determining lyapunov exponents from a time series. *Physica*, 16:285–317, 1985.

[89] Lili Ma Xusong Xu. Rbf network-based chaotic time series prediction and its application in foreign exchange market. In *Proceedings Of The International Conference On Intelligent Systems And Knowledge Engineering (ISKE 2007)*, 2007.

[90] Xin Yao. Evolutionary artificial neural networks. In A. Kent, editor, *Encyclopedia of computer science and technology*, volume 33, pages 137–170. Marcel Dekker Inc., New York, 1995.

Part II

APPENDIX

APPENDIX

Additional properties of the tools mentioned in Chapter 2 and of the other ones are described next..

Tool	**Licence**	**Platforms**	**Community**	**Languages**	**Focus on**	**Coding style**
emergent Neural Network Simulation System 6.0.1 *Dr. Randy O'Reilly*	GNU GPL	Windows, OS X, Linux, Unix	Mailing List	C++	Networks	Visual scripts,
XNBC 9.10-i *Dr. Jean-François VIBERT*	GNU GPL	Windows, OS X, Linux, Unix		C++	Neurons Networks	Visual
Wolfram Mathematica Neural Networks 1.1.1 *Wolfram Research Inc.*	©, Propietary	Windows, OS X, Linux, Unix				Lua scripts
Torch5 0.80 *Ronan Collobert* *Samy Bengio* *Leon Bottou*	GNU GPL	OS X, Linux, Unix		C	Networks	Phtyton scripts C++
Topographica Neural Map Simulator 0.9.7 *Dr. James A. Bednar*	GNU GPL	Windows	Mailing List	C++ Python	Networks	Visual
Stuttgart Neural Network Simulator 4.3 *Dr. Andreas Zell*	FOSS	Windows , Unix	Mailing List	C++	Networks	Configuration files
SpikeNET 1.02 *Arnaud Delorme,* *Simon Thorpe*	GNU GPL			C++	Networks	Visual scripts
SpikeFun 0.63 *Ivan Dimkovic*		Windows		C,C++	Networks	Configuration files

Tool	**Licence**	**Platforms**	**Community**	**Languages**	**Focus on**	**Coding style**
Simbrain 3.0alpha *Jeff Yoshimi*	GNU GPL	Java	forum	Java	Networks	Visual scripts
ECANSE 2.02 *Siemens*	©, Propietary	Windows		Visual C++		Visual scripts
SNNAP 8.1 *Dr. John Byrne,* *Dr. Douglas Baxter*	©, Propietary	Java	forum	Java	Neurons Networks	Visual
PyDSTool 0.88 *Dr. Robert Clewley*	BSD	Python	Wiki	Python	Neurons	Scripts de Python
Peltarion Synapse 1.5.0 *Peltarion*	©, Propietary	Windows	forum	.Net		Visual scripts Componentes .NET
PSICS 1.0.9 *Dr. Matthew Nolan*	GNU GPL	Java		Java Fortran		Visual
PDPTool 2.02 *Jay McClelland*	GNU GPL	Windows, OS X		Matlab	Networks	Visual MATLAB
PCSIM 0.5.4 *Dr. Thomas Natschlager* *Dr. Pecevski Dejan*	GNU GPL	OS X, Linux, Unix	Mailing List	C++ Python	Neurons	Scripts Java
Neuroph 2.4 *Zoran Sevarac* *Ivan Goloskokovic* *Jon Tait*	FOSS	Java	forum	Java	Networks	Java
Neuron-C 6.44 *Dr. Robert Smith*	GNU GPL	OS X Linux Unix		C++	Neurons Networks	Visual scripts Scripts compilados
NeuroSolutions 6.02 *NeuroDimension, Inc*	©, Propietary	Windows Linux (sin GUI) Unix (sin GUI)		Visual C++	Networks	Visual C MATLAB
NeuroJet 3.0 *William Levy* *Ben Hocking*	GNU GPL	Window OS X Linux	Wiki	C, C++	Networks	MATLAB Leng. especifico

Tool	Licence	Platforms	Community	Languages	Focus on	Coding style
Aprotim Sanyal						
Nengo 1.2 *Chris Eliasmith* *Terry Stewart* *Bryan Tripp*	GNU GPL	Windows Linux OSX		Java Python	Networks Neurons	GUI
NEURON 7.1 *Dr. Michael Hines*	GNU GPL	Windows OS X Linux Unix	forum	C, C++ Fortran	Networks Neurons	Visual scripts Compiled Scripts
NEST 2.0 beta *Markus Diesmann* *Jochen Martin Eppler* *Marc-Oliver Gewaltig*	©, Propietary	Windows OS X Linux Unix	Mailing List	C++ Python	Networks	Scripts Python
Mvaspike 1.0.17 *Mohamed Ghaïth Kaabi* *Dominique Martinez*			Mailing List		Neurons Networks	
MOOSE 1.20beta *Upinder S. BhAlla* *Niraj Dudani* *Subhasis Ray*	GNU GPL	Windows OS X Linux Unix	Mailing List	C	Neurons Networks	Visual
MATLAB Neural Network Toolbox 6.0.4 *Mathworks*	©, Propietary	Windows OS X Linux Solaris 64-bit			Neurons	Visual MATLAB
LENS2.63 *Dr. Douglas Rohde*	FOSS	Windows OS X Linux		C	Networks	Visual
KInNeSS 0.3.4 RC2 *Dr. Anatoli Gorchetchnikov*	GNU GPL	Linux		C++		
JavaNNS 1.1 *Dr. Andreas Zell*	©, Propietary	Java		Java	Neurons	Visual
HHsim 3.1	GNU GPL	Windows		Matlab	Neurons	

Tool	**Licence**	**Platforms**	**Community**	**Languages**	**Focus on**	**Coding style**
Dr. David S. Touretzky *Mahtiyar Bonakdarpour* *Mark V. Albert*		OS X Linux Unix				Visual
GENESIS 2.3 *Dr. Dave Beeman* *Dr. James Bower*	GNU GPL	Windows Mac Linux	Mailing List	C		Visual scripts Compiled Scripts
FANN 2.1.0beta *Steffen Nissen*	GNU GPL	Windows OS X Linux Unix	forum	C	Neurons Networks	Visual Bindings c/lengs.
Encog 2.4 *Jeff Heaton*	FOSS	Java .NET	forum	Java, C#	Networks	Java, C# Other langs. .NET
Catacomb2 2.111 *Robert Cannon*	GNU GPL	Java		Java	Neurons Networks	Visual
CX3D *Frédéric Zubler*	GNU GPL	Java		Java	Neurons	Java
CNS r341 *Jim Mutch*	GNU GPL	Windows OS X Linux Solaris 64-bit	forum	C	Neurons Networks	MATLAB C
Brian 1.2.1 *Romain Brette* *Dan Goodman*	GNU GPL	All	forum	Python	Neurons Networks	Scrips Python
Basic Prop 1.2 *Fred Cummins*		All		Java	Neurons	Not required experience in programming
(iNVT) iLab Neuromorphic Vision C++ Toolkit 3.1 *Dr. Laurent Itti* *Dr. Christof Koch*	GNU GPL	All	forum	C++	Networks	C++

Note: In ***red italic*** are shown the tools that work with PyNN

Tool	***Backpropagation***	***Self Organizing***	***Constraint Satisfaction***	***Supported Neurons***	***Parallel processing***
emergent Neural Network Simulation System 6.0.1	Cross-entropy Feedforward (SRN)	Competitive Learning (hard/soft) Hebbian (soft/hard/ZSH Kohonen)	Boltzmann Binary/continuous Hopfield GRAIN	Point Biological	MPI GPU
XNBC 9.10-i	Feedforward Radial Basis Recurrent		Hopfield	Point Biological	
Wolfram Mathematica Neural Networks 1.1.1					
Torch5 0.80					
Topographica Neural Map Simulator 0.9.7		Hebbian		Point / Biological	
Stuttgart Neural Network Simulator 4.3 D	Quickprop RPROP Backpercolation Contraptopagación	Hebbian Kohonen Delta		Point	
SpikeNET 1.02				Biological	
SpikeFun 0.63				3D	
Simbrain 3.0alpha		Hebbian SOM Short Term Plasticity	Hopfield IAC	Point Biological	
Siemens ECANSE 2.02		Self-organising maps			
SNNAP 8.1			Hebbiano	Point Biological	
PyDSTool 0.88				Point	
Peltarion Synapse 1.5.0	Quickprop Levenberg-Marquardt Recurrent	Self-organising maps Hebbian Kohonen Competitive Learning	Hopfield	Point	
PSICS 1.0.9				Point 3D Biological	
PDPTool 2.02	Backpropagation Recurrent	Competitive Learning		Point	
PCSIM 0.5.4	STPD	possible		Point Biological	MPI
Neuroph 2.4	MLP with backprop.	Kohonen Hebbian Competitive	Hopfield	Biological	
Neuron-C 6.44	possible	possible	possible	Biological	MPI
NeuroSolutions 6.02	Feedforward generaliz. Levenberg-Marquardt	Competitive Learning Hebbian			GPU
NeuroJet 3.0		possible	possible	Biological	
Nengo 1.2				Point Biological	
NEURON 7.1	possible	possible	possible	Point	

Tool	***Backpropagation***	***Self Organizing***	***Constraint Satisfaction***	***Supported Neurons***	***Parallel processing***
				3D Biological	MPI
NEST (NEural Simulation Tool) 2.0 beta				Point Biological	MPI
Mvaspike 1.0.17				Point Biological	
MOOSE (Multiscale Object-Oriented Simulation Environment) 1.20beta					
MATLAB Neural Network Toolbox 6.0.4	RPROP	Hebbian LVQ Widrow-Hoff Kohonen			
LENS 2.63	Quickprop Feed-forward Recurrent RBPTT/CRBPTT	Kohonen	Boltzmann	Point	
KInNeSS 0.3.4 RC2	Linux		C++		
JavaNNS 1.1	Quickprop RPROP Backpercolation Counterpropagation	Hebbian Kohonen Delta		Point	
HHsim 3.1				Biological	
GENESIS 2.3	possible	possible	possible	Point Biological	
FANN 2.1.0beta	RPROP iRPROP Quickprop	Kohonen	No	Point	
Encog 2.4	Counterpropagation Elman Recurrent	Kohonen RSOM Competitive Learning	Hopfield Boltzmann	Point	GPU
Catacomb2				Biological	
CX3D (CorteX simulation in 3D)				Biological	
CNS (Cortical Network Simulator) r341	3D convolutional networks			Biological	GPU
Brian 1.2.1		STDP short-term plasticity		Point 3D Biological	GPU
Basic Prop 1.2	SI	No	No		
(iNVT) iLab Neuromorphic Vision C++ Toolkit 3.1				Point	MPI