Instituto de Computación – Facultad de Ingeniería
Universidad de la República

Tesis de Maestría
en Ingeniería en Computación

Predicción de series temporales utilizando redes neuronales

Horacio Paggi

Director: M.Sc. Graciela Ferreira

Montevideo, Uruguay
Diciembre de 2003

Predicción de series temporales
utilizando redes neuronales
Horacio Paggi

ISSN 1510-7264
Tesis de Maestría en Ingeniería en Computación
Instituto de Computación – Facultad de Ingeniería
Universidad de la República.

Montevideo, Uruguay, Diciembre de 2003.

Resumen

Una serie temporal consiste en un conjunto de valores que pueden ser considerados observaciones tomadas de un cierto sistema. En particular, nos interesarán las series generadas por sistemas dinámicos. Dichos sistemas, en algunos casos, pueden ser descritos por un conjunto de ecuaciones, las que modelan el mecanismo subyacente que genera las observaciones; sin embargo, en muchos sistemas reales, esas ecuaciones son desconocidas, y la única información disponible sobre los mismos es una secuencia temporal de medidas, que forma una serie temporal. Asociada a esa serie temporal, suele existir la necesidad práctica de predecirla, es decir, conocer los valores futuros a un instante t dado. Al respecto se pueden tomar dos posturas no excluyentes: desarrollar un modelo que solamente ***prediga*** los valores siguientes de la serie, a partir de un cierto conjunto de datos históricos, o bien hacer una ***reconstrucción dinámica*** del sistema, es decir, tratar de que el modelo capte además ciertas características del sistema original de forma de hacer corresponder el comportamiento a largo plazo del sistema computacional construido (el modelo) con el exhibido por los datos observados. Los dos enfoques son aplicados en este trabajo y ejemplificados con un caso de estudio: el modelado de las ventas de gas propano licuado envasado en garrafas, tanto para su ***predicción*** como para la ***reconstrucción dinámica*** del sistema asociado. Adicionalmente se describen el análisis y la selección de las herramientas informáticas disponibles para llevar a cabo esa tarea y se exponen los principales aspectos del *estado actual del arte de la teoría asociada al uso de redes neuronales aplicadas a la predicción de series temporales.*

Agradecimientos

Quisiera poder mencionar aquí a todos aquellos que me dieron su apoyo y aliento durante todo el tiempo que duraron los estudios de esta Maestría, y en especial, en el período en el cual realicé el presente trabajo, pero como acordarme de todos ellos es imposible, agradezco a:

- El CPAP, en especial a Nora Szasz que tuvo confianza en mí para la realización de esta tesis y que con su seguimiento me impulsó a seguir adelante día a día, a través de todos los vaivenes que vivió el país a través de estos últimos tres años. También quiero agradecer a las Secretarias del CPAP, especialmente a Beatriz Romero, que siempre pusieron la mejor buena voluntad para sortear todos los inconvenientes de un proyecto que recién se iniciaba, como eran las Maestrías Profesionales.

- A mi orientadora, Graciela Ferreira, que tuvo la constancia y paciencia de acompañarme cada semana, motivándome a realizar un trabajo homogéneo y de calidad, dándome siempre buenas sugerencias; que me perdone por haber usado un font tan chico...

- A Laura Bermúdez, de la Secretaría del PEDECIBA, que también estuvo siempre receptiva y dispuesta a colaborar con su granito de arena y con la mejor sonrisa.

- Al Ingeniero Nelson Ucha, a la Economista Haydée Pérez y a la Contadora Ilda Rivero, de ANCAP, que me apoyaron desde el principio en la idea de realizar una Maestría; a mis compañeros de la División Sistemas de Información de ANCAP, quienes toleraron a alguien que a veces parecía tener su cabeza en otro planeta, y de entre ellos, a Adela Casero, porque me proporcionó datos vitales para el entrenamiento de las redes, y me ayudó con el manejo del Microsoft Word y sus vericuetos. También quisiera agradecer a Bernardo Zimberg porque también aportó información imprescindible (tal como las temperaturas diarias) y porque se involucró en el trabajo deseando aprender y realizando sugerencias interesantes.

- A la bibliotecaria del InCo-PEDECIBA, Joseline Cortazzo, porque no solo me ayudó a conseguir la bibliografía necesaria haciéndome posible acceder a las bibliotecas de otros institutos a través de ella, sino que además fue una muy grata interlocutora de charlas para-académicas.

- Finalmente, mi agradecimiento profundo a mi madre, María Esther Straneo de Paggi, que me apoyó y supo convivir con un hijo que estuvo a punto de convertirse en menos que una visita en la casa.

A todos ellos les vuelvo a decir ¡Gracias!

Horacio Paggi
Diciembre/2003

Índice

Lista de abreviaturas y símbolos

Se describe a continuación la notación empleada en el documento y para las referencias utilizadas. Algunos significados pueden cambiar localmente, dependiendo del contexto.

Variables y funciones:

$\mathbf{x} = [x_1, x_2 \dots x_n]$	Vector n-dimensional
$\mathbf{y} = \{y_1, y_2 \dots y_n\}$	Representa una lista ordenada de n elementos o los elementos de una matriz: la matriz $\mathbf{H} = \{h_{ij}\}$ esta formada por elementos h_{ij} donde i representa la fila y j la columna
∇E **(w)**	Gradiente de la función escalar $E = E(\mathbf{w})$ siendo $\nabla E = [\frac{\partial E}{\partial w_1}, \dots \frac{\partial E}{\partial w_n}]$, $\mathbf{w} = [w_1, \dots w_n]$
H	Matriz hessiana de E $\mathbf{H} = \{H_{ij}\}, H_{ij} = \frac{\partial E}{\partial w_i \partial w_j}$
x_i	Entrada i-ésima de una neurona
w_j	Peso j-ésimo (tomando el conjunto de pesos como un vector)
w_{ji}	Peso de la conexión entre la neurona i y la j; la conexión va de la i a la j
$\langle X \rangle_K$	Esperanza de $X = X(K)$ sobre todos los valores de la variable aleatoria K
$E(\mathbf{w})$	Función de error. En ciertos casos, $J(\mathbf{w})$.
$f(\bullet)$	Función de salida (o de transferencia)
$f_h(\bullet)$	Función de salida para la capa oculta
$f_o(\bullet)$	Función de salida para la capa de salida
ρ	Tasa de aprendizaje en forma genérica
ρ_h	Tasa de aprendizaje para la capa oculta
ρ_o	Tasa de aprendizaje para la capa de salida
Net_k	Entrada (o potencial) de la neurona k-ésima definido como $Net_k = \sum_{i=1}^{n} x_i w_{ki}$ para el instante actual
$Net_k(n)$	Idem pero para el instante n
d_i	Salida deseada para la neurona i-ésima dadas sus entradas actuales

y_i	Salida obtenida para la neurona i-ésima dadas sus entradas actuales
d_E	Dimensión global del subespacio de la señal ("embedding"). También denotado como m
m	Dimensión global del subespacio de la señal ("embedding"). También denotado como d_E
d_L	Dimensión local del subespacio de la señal ("embedding")
d	Delay óptimo del subespacio de la señal ("embedding") . También denotado como τ.
R	Conjunto de los números reales
S, S-1, S-2, S-3	Número de semana actual, de la anterior, de la anterior a la anterior, etc. respectivamente
V, V-1, V-2, V-3	Ventas de gas de la semana actual, de la anterior, de la anterior a la anterior, etc. En general representa las ventas a predecir, aunque en algunas partes se utiliza V+1 para ello (cuando la entrada a la red esta inspirada en el teorema de Takens-Mañé)
V+1	Ventas de la semana próxima, a predecir
T, T-1, T-2, T-3	Temperatura máxima diaria promedio de la semana actual, de la semana anterior, etc.
A, A-1, A-2, A-3	Indicador de aumento de la semana actual, de la anterior, etc.
%Error CV	Porcentaje de error cometido al predecir en un paso, promediado sobre el conjunto de "cross validation"
%FNN	Porcentaje de FNNs ("False Nearest Neighbors")

Referencias:

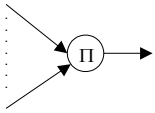
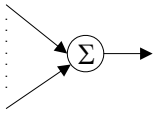
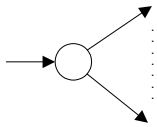
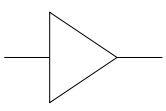
- Las referencias externas se anotan como [xx] siendo xx un número de publicación o libro de los que se detallan en la Sección V.
- Las referencias a páginas web como [*dirección de sitio web*]
- Las ecuaciones se referencian como Sección.número de ecuación

Abreviaturas:

AG	Algoritmo(s) genéticos(s)
AIC	"Akaike's Information Criterion" (Criterio de Información de Akaike)
BP	"Back-propagation"
BPTT	"Back-propagation Through Time"
BPTT(*h*)	BPTT truncada
CV	"Cross Validation"
DC	Dimensión de Correlación
deKf	"Decoupled Extended Kalman Filter"

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" (Perceptrón Multi Capa)
MSE	"Mean Squares Error"
PCA	"Principal Component Analysis" (Análisis de Componentes Principales)
RP	"Recurrence Plot"
RQA	"Recurrence Quantification Analysis"
RTRL	"Real Time Recurrent Learning"
SER	"Signal to Error Ratio"
SOM	"Self Organizing Map" (Mapas de Kohonen)
TDNN	"Time Delayed Neural Network"
TLFN	"Time Lagged Feedforward Network"

Otros símbolos:

**********	Marca el mejor modelo desde el punto de vista del MDL
→	Conexión entre dos neuronas, o flujo de entrada o de salida a una de ellas
	Una flecha de este tipo representa todas las conexiones posibles entre las neuronas que aparecen en su origen y las que figuran en su fin.
	La salida de un nodo marcado con una letra pi mayúscula es el producto aritmético de sus entradas.
	La salida de un nodo marcado con una letra sigma mayúscula es la suma aritmética de sus entradas.
	Las salidas de un nodo de este tipo son idénticas a su entrada.
	Un triángulo representa la multiplicación de su entradas por el valor escrito dentro del triángulo

Sección I: Introducción y planteo del problema

Los sistemas dinámicos evolucionan en el tiempo de acuerdo a un conjunto de reglas que generalmente son no lineales. En ellos., las condiciones actuales determinan el futuro y pueden existir varias variables de estado involucradas que interactúan entre sí. Los sistemas de ese tipo son estudiados en todas las ramas de la ciencia, desde el sistema solar a la fluctuación de los valores de las acciones en la bolsa [ANEXO VII]. En nuestro caso, estudiaremos las ventas de gas propano en garrafas, medidas en litros.

Una de las metas del estudio de los sistemas dinámicos es la ***predicción***, es decir, saber hacia dónde se encamina el sistema en consideración a partir de un cierto conjunto de datos históricos: si el precio de las acciones va a subir o bajar, si las manchas solares van a cambiar significativamente de lugar o si se va a vender más o menos gas. En nuestro caso, una predicción acertada permitirá minimizar los costos de mantenimiento de stock asociados. Esta predicción se caracteriza por tener un horizonte (alcance temporal en el futuro) relativamente cercano.

Otra área de interés es la de la ***reconstrucción dinámica*** (o modelado dinámico). La reconstrucción dinámica y la predicción de series temporales son dos áreas que tienen muchos aspectos en común pero difieren en la forma en la que la solución obtenida es evaluada: mientras que el objetivo de la predicción de series temporales desde el punto de vista clásico [52] es obtener el siguiente valor de la serie de forma que sea tan próximo como sea posible a los datos reales, el fin de la reconstrucción dinámica es capturar la dinámica (comportamiento en un futuro lejano) del sistema por medio de un modelo aproximado. Por ejemplo, mediante la reconstrucción dinámica se puede obtener un modelo con el cual realizar simulaciones del comportamiento de las ventas de gas en ciertas condiciones [5]. En el presente trabajo se consideran los dos enfoques y se discuten los modelos más adecuados para cada uno de ellos.

Si existe suficiente información disponible sobre las leyes (físicas en el caso de sistemas de la naturaleza, matemáticas en el caso general) que gobiernan el sistema, entonces puede ser adecuado un enfoque analítico, en el cual las ecuaciones que describen el mecanismo subyacente responsable de la generación de la serie temporal se derivan basándose en las leyes mencionadas. En muchos problemas reales de interés, no obstante, no tenemos suficiente información a priori como para intentar un enfoque de ese tipo. De hecho, aunque los sistemas tengan varias variables que gobiernan su comportamiento, a menudo solo se dispone de las medidas de una sola de ellas en la forma de una serie de datos dependientes del tiempo. En ese caso, en ausencia de información suficiente como para derivar las ecuaciones matemáticamente, es imperioso emplear un enfoque basado en modelos no analíticos. Un ejemplo de tal enfoque es el de las redes neuronales, en el cual una red es entrenada de forma que sus salidas sean similares a los valores observados. En el caso de un sistema determinístico, dichos valores están dados por las ecuaciones generadoras de la serie ([1], [5], [11]). Por lo tanto, en lugar de deducir explícitamente las ecuaciones que describen la dinámica subyacente del sistema, es construido un modelo neuronal que aproxime las ecuaciones ideales. Por otra parte, en el caso de un sistema dinámico estocástico, los valores observados corresponden a una realización del proceso aleatorio que genera la variable observable, y lo que se busca es hacer que la red neuronal sea un modelo estadístico de dicho proceso [1]. Las redes neuronales son atractivas para modelar sistemas dinámicos no lineales ya que son inherentemente no lineales (debido a la función no lineal de salida de las neuronas) y ya que ellas son aproximadores funcionales sólidamente fundamentados (Ver ANEXO VIII sobre los teoremas de Kolmogorov y Cybenko). Adicionalmente, para el caso de redes "feedforward" (Ver II.1.2), se sabe que constituyen una forma de modelar la densidad de probabilidad condicional de obtener una cierta salida deseada $\mathbf{d}$ a una entrada $\mathbf{x}$ (Ver [1]).

Los objetivos del presente trabajo son:

a) investigar el estado del arte de las redes neuronales aplicables a la predicción de valores de series temporales.

b) construir un modelo neuronal del sistema dinámico no lineal con el que se puedan predecir las ventas de gas propano envasado y del que se conocen unas pocas variables (tales como temperatura diaria y volumen de las ventas de gas), utilizando para ellos las herramientas informáticas que se encuentren más adecuadas. El modelo (o modelos) permitirá(n) predecir las ventas de gas en un período de al menos una semana. Esta modelización estaría asociada a la investigación de las propiedades teóricas que hacen que un modelo fuera preferible a otro.

En el esquema de modelado propuesto en este trabajo, se usan redes recurrentes y redes TLFNs (Ver ANEXO III) como topologías preferidas y los resultados obtenidos son comparados con los obtenidos con redes de tipo perceptrón

multicapa (MLP). Asimismo se buscó mejorar la calidad de la predicción por medio de un comité de redes (el "ensemble method" de Perrone). El uso de las redes recurrentes y TLFNs viene justificado no solo por la bibliografía (Ver [5] y [11]) sino que fue motivado como alternativa a explorar respecto al empleo de un MLP entrenado con "back-propagation" estático, que fue la solución dada por [15] a un problema similar de predicción de demanda de energía eléctrica. En cuanto al algoritmo de entrenamiento, utilizamos el "back-propagation" en una de sus variantes ("back-propagation through time", BPTT) fundamentalmente porque era el algoritmo proporcionado por la herramienta de software elegida para la simulación de redes neuronales. Las otras alternativas posibles al uso de los MLPs como aproximadores universales las constituyen el uso de ***redes de funciones de base radial*** ("radial basis functions networks" o "RBF networks") y las "***support vector machines***". Por motivos de extensión estas opciones no son exploradas en el presente trabajo, pudiendo consultarse al respecto la bibliografía ([1], [5], [11]).

Debido a las debilidades propias de los algoritmos de aprendizaje basados en el descenso en el gradiente cuando se aplican a redes recurrentes ([5], [54]), se incluye el estudio de otras formas de entrenamiento, tales como la del filtro de Kalman ([5], [53], [69], [21]) y los algoritmos evolutivos ([71], [4]), aunque no se realizaron experimentos con ellos en cuanto a la tarea de entrenamiento; sí se utilizaron algoritmos evolutivos para determinar la estructura de algunas de las redes (cantidad de neuronas ocultas y entradas a utilizar).

Una vez construidos los modelos neuronales, se procedió a su validación. Para ello se realizaron pruebas de ciclo abierto y de ciclo cerrado ("***open loop***" y "***closed loop***", respectivamente) [52]. En el modo de ciclo abierto, se le da a la red el valor verdadero de la serie temporal (señal) en cada instante y el objetivo es predecir el valor de la serie un paso adelante en el tiempo. En el modo de ciclo cerrado, la red opera dependiendo de sus propias salidas en cada paso de tiempo, más una copia de las entradas exógenas que tuvo en los correspondientes instantes del pasado. Para predecir varios pasos en el futuro se necesita realimentar la red con sus salidas; dicha predicción puede hacerse difícil debido a que los errores de salida son amplificados en cada paso. En el modo de ciclo cerrado esperamos que, a igualdad de entradas exógenas, las salidas de la red sigan los valores reales de la señal muy de cerca durante unos pocos instantes y luego se desvíen considerablemente. Se define el ***modo "priming"*** como un modo de trabajo consistente en, para cada uno de uno de p_L instantes consecutivos, ir sustituyendo una entrada de la red por su respectiva predicción. A p_L se lo llama ***largo del periodo de "priming"***. Por ejemplo, si la red va a procesar los datos de dos instantes hacia atrás, será $p_L = 2$. La evaluación con ciclo cerrado no puede iniciarse sin antes inicializar las entradas de la red con los valores reales de la señal (no vistos durante el entrenamiento) y luego operar en el ***modo "priming"*** durante p_L instantes. En el modo de ciclo abierto, se eligieron los modelos buscando minimizar el MSE para una cierta arquitectura (lo cual es equivalente a minimizar el porcentaje de error cometido, ver III.3.3 "Validación a corto plazo") y a similares porcentajes de error cometidos, según el MDL. En el modo de ciclo cerrado, se trató de buscar el modelo tal que su porcentaje de error iterado promedio fuese menor y además cuyos invariantes fuesen lo más próximos posibles a los de la serie original. Se realizó un estudio de los invariantes de la serie obtenida mediante el "ensemble method" y se extrajeron algunas conclusiones.

El trabajo realizado se informa según el siguiente esquema:

La ***Sección II*** contiene los principales fundamentos teóricos de la aplicación de las redes neuronales al modelado de los sistemas dinámicos Con la idea de que estos fundamentos constituyen un documento auto contenido, en lo posible, se incluyen allí la descripción del algoritmo de BP, algunas de sus variantes, incluyendo las que permiten la búsqueda del óptimo global, y diferentes topologías de red neuronal adecuadas al proceso de estos sistemas. Esta Sección se complementa con los ***Anexos I al VIII.***

La ***Sección III*** presenta cómo se aplican las redes neuronales al caso de estudio, la predicción de las ventas semanales de gas. Se describen los experimentos realizados, los modelos construidos y un resumen de los resultados obtenidos. Esta Sección se complementa con el ***Apéndice*** que contiene algunos resultados numéricos y tablas de valores utilizados.

La ***Sección IV*** plantea algunas observaciones sobre los experimentos realizados y expone las conclusiones generales del trabajo y algunos posibles trabajos futuros.

Finalmente, la bibliografía utilizada, tanto en formato de libros, publicaciones o sitios web, con un pequeño comentario sobre ellos forma la ***Sección V***.

Sección II: Redes neuronales aplicadas a la predicción de series temporales

En esta Sección se exponen algunos de los fundamentos teóricos en los que se basa el presente trabajo. Se comienza en describiendo el algoritmo de "back-propagation" en el caso estático (entradas y salidas independientes del tiempo), asociándolo a la regla del LMS ("least mean squares") y se exponen luego algunas de sus mejoras, tales como las búsqueda de segundo orden, etc. A continuación se describen los comités de redes y el método del "ensemble". Dado que se hace un uso extenso de ciertos conceptos de la teoría de los sistemas de información, se presentan las entropías y la información mutua. Posteriormente se detallan las redes dinámicas, que son un tipo de redes capaces de modelar los sistemas dinámicos en una forma natural. Finalmente se dan algunos lineamientos en lo referente al diseño de la red y de su entrenamiento.

Por más detalles sobre algunos de los puntos descriptos en esta Sección referirse a los Anexos.

II.1 El algoritmo de retro-propagación ("back-propagation")

Describiremos a continuación el algoritmo utilizado para el entrenamiento de todas las redes empleadas en este trabajo así como su justificación teórica y algunas de las variantes del método.

El aprendizaje en las redes neuronales es un aprendizaje "por ejemplos" [1]. Podemos dividir este aprendizaje en supervisado y no supervisado. Supervisado es el basado en la comparación directa entre la salida real de la red y la salida deseada correcta. A menudo se lo formula como la minimización de una función de la diferencia (error) entre la salida real y la deseada, tal como el error medio cuadrático, y cuyas variables son los pesos de la red. Para ello, se pueden aplicar algoritmos de optimización basados en el descenso en el gradiente, tales como el de retro-propagación ("***back-propagation***" o ***BP*** por sus siglas en inglés). Por otra parte, el entrenamiento no supervisado esta solamente basado en las correlaciones entre los datos de entrada y en él se supone que no hay información disponible de la salida deseada de la red, y se traduce igualmente en un proceso de cambio de los pesos de la red [60].

Por retro-propagación entendemos, básicamente, a un algoritmo de aprendizaje que se utiliza para entrenar redes multicapa[1] que fue diseñado por David Rumelhart, G.E. Hinton y R. J. Williams en 1986, a partir de trabajos de Parker en 1985 y Werbos (1974) [1]. Este algoritmo, de aprendizaje supervisado, puede ser considerado una generalización de la regla delta. Actualmente, BP es uno de los algoritmos (modelos) más populares, en parte debido a ser simple de implementar, más rápido que otros métodos y haber sido ampliamente probado en la práctica.

Las reglas de aprendizaje bases de este algoritmo son la regla delta y la de mínimos cuadrados ("least mean squares" o LMS).

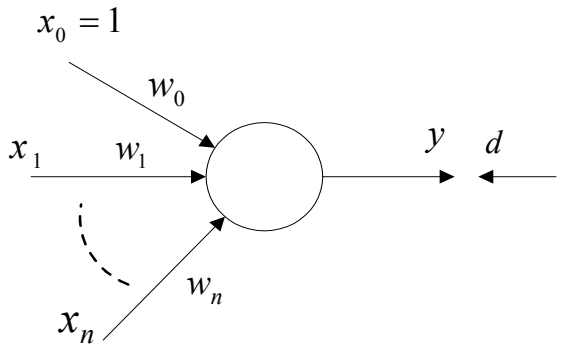
II.1.1 La regla delta

La regla delta es una extensión de la regla de aprendizaje de LMS ("least mean squares") a redes con neuronas con funciones de salida no lineales y continuas. Esta regla es local en cuanto al patrón y los pesos, es decir, requiere información sobre el patrón específico, el error cometido por la neurona y el valor de la entrada a la misma.

[1] Aunque al modelo de una red "feedforward" multicapa entrenada con este algoritmo de aprendizaje se lo suele llamar "red back-propagation" ([3] y [6])

La regla LMS vista como un proceso estocástico

Sea la siguiente neurona a entrenar cuya función de salida es lineal (por lo que llamaremos la llamaremos ***neurona lineal***):



donde las entradas son las x_i, w_i sus pesos asociados, w_0 representa el umbral de activación, ***d*** el valor deseado de salida conocido e $y = y(\mathbf{w})$ el valor de salida obtenido:

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

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

Se tratará de minimizar el error cuadrático medio (manteniendo $x_0 = 1$) definido como

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

donde $\langle \ \rangle_{\mathrm{x}}$ representa la media sobre todos los vectores de entrenamiento **x** posibles. Para buscar ese mínimo de E se podría hacer un descenso según su gradiente:

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

pero en su lugar se aproxima $\nabla E(\mathbf{w})$ por su valor instantáneo:

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

con el subíndice k significando "correspondiente a la presentación del patrón k-ésimo en la entrada" lo cual nos lleva al algoritmo:

$$\begin{cases} \mathbf{w}_0 \quad \text{arbitrario} \\ \mathbf{w}_{k+1} = \mathbf{w}_k + \rho_k [d_k - (\mathbf{x}_k)^{\mathrm{T}} \mathbf{w}_k]\mathbf{x}_k \end{cases}$$

siendo ρ_k una constante cuyas propiedades se detallan a continuación en el punto "Convergencia".

A este proceso iterativo se lo conoce como ***proceso de aproximación estocástica*** (o de Kiefer-Wolfowitz o Robbins-Monro, Ver [4]).

Convergencia

Sea la matriz de autocorrelación $\mathbf{C} = \left\langle \mathbf{x}\mathbf{x}^T \right\rangle_X$ donde $\langle \cdots \rangle$ aplicado a una matriz se entiende como el valor esperado de cada uno de sus elementos. En este caso sería $\mathbf{C} = \left\{ c_{ij} = \left\langle x_i x_j \right\rangle_X \right\}$ siendo X una variable aleatoria que representa todos los patrones de entrada $\mathbf{x}$ posibles. Se puede demostrar [4] que si el determinante de $\mathbf{C}$ no es 0 y las tasas de aprendizaje ρ_k cumplen que

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

siendo m número de patrones de entrenamiento, entonces, la sucesión de vectores $\{\mathbf{w}_k\}$ converge asintóticamente en la media cuadrática al $\mathbf{w}^*$ que produce el mínimo valor de E **:**

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

La regla delta

En el caso en que la salida ***y*** de la neurona sea $y = f\left(\sum\limits_{i=0}^{n} x_i w_i \right)$ siendo f una función diferenciable no lineal, la regla de entrenamiento obtenida es conocida como ***regla delta*** ya que lo que se busca es minimizar una "delta" (diferencia) entre los valores obtenidos y los valores deseados. Análogamente que antes, buscamos minimizar el valor medio de la diferencia $J(\mathbf{w}) = \frac{1}{2} \left\langle (\mathbf{y}(\mathbf{w}, \mathbf{x}) - d)^2 \right\rangle_X$ cuyo gradiente es aproximado usando el gradiente instantáneo

$$\nabla_k J(\mathbf{w}) = -(d_k - y_k) f'(Net_k) \mathbf{x}_k$$

$$Net_k = \sum_{i=0}^{n} (\mathbf{x}_i)_k w_i$$

$$f'(Net_k) = \left. \frac{\partial f}{\partial Net} \right|_{Net_k}$$

donde $(\mathbf{x}_i)_k$ significa "el elemento i-ésimo del patrón k-ésimo".
La regla delta nos queda entonces:

$$\begin{cases} \mathbf{w_1} \quad arbitrario \\ \mathbf{w}_{k+1} = \mathbf{w}_k + \rho_k [d_k - f(Net_k)] f'(Net_k) \mathbf{x}_k = \mathbf{w}_k + \rho_k \delta_k \mathbf{x}_k \\ \delta_k = [d_k - f(Net_k)] f'(Net_k) \end{cases}$$

Convergencia

Wittner y Denker [70] demostraron que en ciertos casos el entrenamiento por regla delta nunca converge.

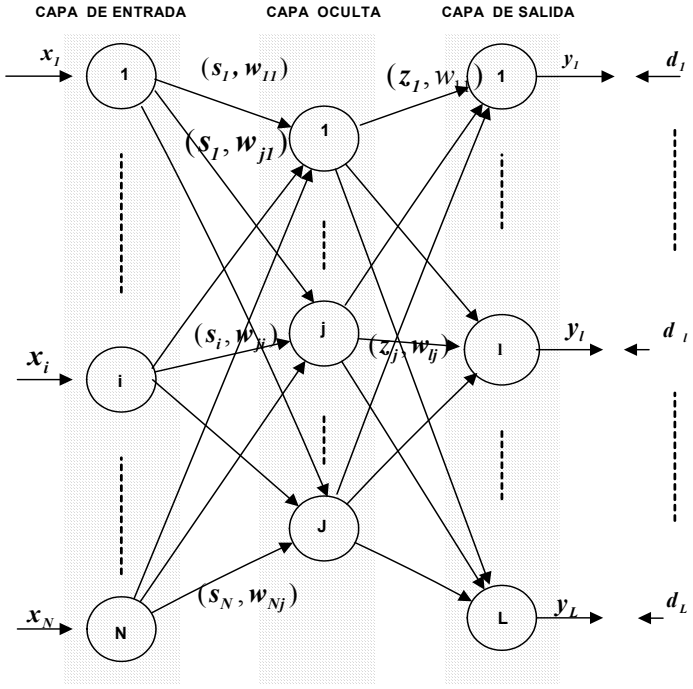
II.1.2 El algoritmo en el caso estático

Llamamos ***red estática*** a aquella cuyas salidas dependen solo de sus entradas actuales. Por ejemplo, un MLP es una red estática.

Una red se dice "***feedforward***" si existe un método de numeración de sus neuronas que las numera a todas de manera que no hay conexión entre una neurona i-ésima y la j-ésima si $i \geq j$ (Ver[11]). Esta definición esta dando por sobreentendido que las conexiones son orientadas, esto es, que la conexión ***i-j*** es distinta que la ***j-i***. Para decirlo más directamente, la red es " feedforward" si puede ser representada por un grafo dirigido acíclico [7].

Consideraremos para empezar el caso más simple: aquel en que la red asocia a cada patrón de entrada $\mathbf{x}_i$ una salida $\mathbf{y}_i$, sin importar las entradas (y por ende las salidas) que haya recibido (generado) antes; es decir, en el caso de que tenga un comportamiento estático en el tiempo.

Supongamos entonces que tenemos la siguiente red feedforward:



siendo s_i la salida de la neurona i-ésima de entrada, w_{ji} el peso entre la neurona i y la j, y z_j la salida de la neurona oculta j-ésima.

Los resultados que se presenten son generalizables a más de una capa oculta [11, Cap. 3].

En lo que respecta a las neuronas en sí, se exige que tengan funciones de salida no decrecientes y derivables con derivada continua. Cada capa puede tener distintas funciones de salida: una f_o para la de salida y otra f_h para la oculta. Estamos asumiendo aquí la misma función de salida (de igual tipo y parámetros) en toda la capa oculta, pero

esto es solo a efectos de simplificar la notación, ya que alcanzaría con agregar un índice que denotara la neurona dentro de la capa a la función *f*. La misma consideración vale para la función de salida de las neuronas de salida.
Esta red recibe como entrada un conjunto de valores (señales) escalares $[x_1,...x_n]$= **x** con **x** ∈ $\mathbf{R}^n$**.** La capa oculta tiene como salida un vector de reales **z** = $[z_1,...z_j] \in \mathbf{R}^J$ y la de salida, otro **y** = **y** (**w, x**) = $[y_1,...y_L] \in \mathbf{R}^L$**,** el cual, cuando la red esté totalmente entrenada, debería ser idéntico (o al menos muy cercano) a un vector de valores deseados **d** asociado a **x.**

Consideremos ahora un conjunto de m pares de entrada/salida $\{\mathbf{x}_k, \mathbf{d}_k\}$ donde $\mathbf{d}_k$ es un vector L dimensional representando la salida deseada para la red cuando la entrada es $\mathbf{x}_k$. Nuestro objetivo es ajustar los pesos de la red de forma que la función (correspondencia) entre las **x** y las **d** subyacente en el juego de entrenamiento sea aprendida. Ya que tenemos disponibles los valores deseados para cada entrada, podemos definir una función de error para medir el grado de aproximación obtenida para un conjunto dado de pesos de la red. Por el momento usaremos como función de error :

$$E(\mathbf{w}) = \tfrac{1}{2}\sum_{l=1}^{L}(d_l - y_l)^2$$

donde **w** representa el conjunto de todos los pesos de la red y donde hemos omitido, por simplicidad, los índices representando el número de patrón (en las secciones posteriores y en el ANEXO I se discuten otras funciones de criterio de error factibles de ser usadas).

El algoritmo visto como un problema de optimización

Luego de haber planteado una función de error, el proceso de aprendizaje queda reducido a un proceso de optimización: se desea minimizar la función de error sobre el espacio de todos los posibles pesos. Como la función E es diferenciable, si realizamos una búsqueda por descenso según el gradiente, tendremos una regla de aprendizaje. Este enfoque fue aplicado independientemente por Amari (1967, 1968), Bryson y Ho (1969), Werbos (1974) y Parker (1985).

La regla de aprendizaje incremental ([4], [5], [11])

Dado que los valores deseados para las salidas son conocidos, se puede usar la regla delta para actualizar los pesos w_{lj}, que llegan a las neuronas de salida obteniéndose:

$$\Delta w_{lj} = w_{lj}^{nuevo} - w_{lj}^{actual} = -\rho_o \frac{\partial E}{\partial w_{lj}} = \rho_o (d_l - y_l) f_o'(Net_l) z_j$$

siendo ρ_o la ***tasa de aprendizaje*** de las neuronas de la capa de salida y $l = 1, 2,...L \quad j = 0, 1,....J$. Acordamos que el umbral de activación θ_i de cada neurona es simulado por el agregado de una entrada a ella ficticia con valor fijo 1 y con peso θ_i proveniente de una neurona ficticia. No representamos esa neurona ficticia en el diagrama ni sus conexiones, aunque los θ_i deban ser ajustados como un peso más al entrenar la red. Por otra parte, f_o' representa la derivada de *f* (función de salida) respecto Net y w_{li}^{nuevo} y w_{li}^{actual} representan los pesos actualizados (es decir, luego de aplicarles la regla de actualización) y actuales (antes de aplicarla), respectivamente. Los valores z_j (salidas de la capa oculta) son calculados propagando el vector de entrada **x** a través de la red, teniendo

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

con $j = 1, 2...J$ y f_h la función de salida de la neuronas de la capa oculta (h).

La regla de aprendizaje para los pesos que llegan a la capa oculta no es tan obvia ya que no tenemos el conjunto de valores deseados a obtener en las salidas de las neuronas ocultas. Sin embargo, se podría derivar dicha regla tratando

de minimizar el error de la capa de salida. Esto equivale a propagar los errores de la capa de salida $(d_l - y_l)$ hacia atrás a través de las neuronas ocultas en un intento de estimar "dinámicamente" los valores deseados para esas unidades. A esta regla de aprendizaje la llamaremos ***retro-propagación ("back-propagation") incremental del error***, y como se observa que se puede usar la regla delta para actualizar los pesos que llegan a la capa de salida, se suele decir que es una extensión de la regla delta. Esta versión del algoritmo se dice incremental ya que la modificación de los pesos se va haciendo según se presenten los patrones. Más adelante se describe otra versión de BP, llamada ***"batch".***

Estos pesos que llegan a la capa oculta se modifican haciendo un descenso según el gradiente respecto de los pesos ocultos en la función $E(\mathbf{w})$:

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

calculando la derivada parcial con los valores actuales de los pesos. Usando la regla de la cadena para diferenciar es

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

y como $\frac{\partial Net_j}{\partial w_{ji}} = s_i$ y $\frac{\partial z_j}{\partial Net_j} = f_h'(Net_j)$ sustituyendo obtenemos la regla de aprendizaje buscada:

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

Comparando esta ecuación con la correspondiente a la capa de salida, podemos definir un "valor deseado estimado" d_j para la j-ésima neurona oculta en términos de un valor de error retropropagado:

$$d_j - z_j \xrightarrow{corresponde} \sum_{l=1}^{L} (d_l - y_l) f_o'(Net_l) w_{lj}$$

Estas ecuaciones se pueden extender a redes de más de una capa oculta, o con conexiones que salten de una capa hacia otra no inmediatamente posterior [11]

El algoritmo de "back-propagation" incremental podría esquematizarse como:

Paso 0 – Inicialización

Inicializar los pesos con valores pequeños al azar. Deben ser pequeños para evitar la parálisis de la red (volverla insensible al aprendizaje) y aleatorios para que se rompa la posible simetría que hubiera en los pesos y prevenir otros problemas de aprendizaje [4]. *Tomaremos a estos valores como "actuales"* w_{ij}^{actual} .

Además, darle a ρ_o *y* ρ_h *valores positivos y pequeños (ya que con valores grandes se corre el riesgo de pasar por encima del mínimo sin estacionarse en él)*

Paso 1 – Aplicación de un patrón

Presentar un vector cualquiera $\mathbf{x_k}$ *del conjunto de entrenamiento (elegido al azar) a la entrada de la red y propagarlo a través de ella, obteniendo los correspondientes valores de salida*[2] *a partir de los valores actuales de los pesos*

[2] En plural si se consideran los valores de salida de cada neurona de la capa de salida, singular si se toma el vector obtenido como resultado de la presentación de **x** en la entrada.

Paso 2 – Cálculo del error cometido para ese patrón

Comparar las salidas reales con las correctas $\mathbf{d_k}$ *asociadas a* $\mathbf{x_k}$. *y determinar el error cometido en la capa de salida. Calcular los cambios en los pesos que llegan a la capa de salida. usando la fórmula*

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

Paso 3 – Cálculo del cambio a efectuar

Calcular el cambio a los pesos que llegan a la capa oculta, Δw_{ji} *, usando*

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

Paso 4 – Actualización de los pesos

Actualizar todos los pesos según

$w_{lj}^{nuevo} = w_{lj}^{actual} + \Delta w_{lj}$ *y* $w_{ji}^{nuevo} = w_{ji}^{actual} + \Delta w_{ji}$ *para las capas de salida y oculta respectivamente.*

Paso 5 –-Prueba de convergencia

Repetir los pasos 1 a 5 con todos los patrones de entrenamiento. Hacer $w_{ji}^{actual} = w_{ji}^{nuevo}$ *y* $w_{lj}^{actual} = w_{lj}^{nuevo}$ *y volver al paso 1 hasta que el error de salida, para todos los vectores del conjunto de entrenamiento, haya sido reducido a un valor (previamente establecido) como aceptable. También se puede usar otro criterio de convergencia, basándose en otras funciones de error* [4].

Convergencia:

El algoritmo, cuando converge, lo hace a un mínimo que no tiene porqué ser global.

En general, no se puede demostrar que el algoritmo converja [5]. Por ello, se han sugerido varios criterios para determinar cuando detenerlo, siendo uno de ellos de la validación cruzada ("***cross validation***"). Ver ANEXO I.

El algoritmo usa una estimación instantánea para el gradiente de la superficie de error en el espacio de pesos. Por otra parte, hace una elección aleatoria del patrón a presentar o sea del punto inicial en el cual se calcula el gradiente, y por lo tanto tiene una tendencia a zigzaguear respecto a la dirección verdadera, mientras se va acercando acerca hacia un mínimo de la superficie de error, siendo esto en parte debido a que es una aplicación del método de Robbins-Monro de aproximación estocástica [5]. Como zigzaguea, tiende a converger de forma lenta.

Decimos que la serie de soluciones $\{\mathrm{x}_k \mid k = 1,2,...\}$ producidas por un algoritmo iterativo en cada una de sus iteraciones tiene ***convergencia q-lineal*** si existe un a $0 < a < 1$ tal que $e(\mathrm{x}_{k+1}) / e(\mathrm{x}_k) \le a$ siendo $e(\mathrm{x}_k)$ el error de x_k respecto x^* (solución óptima). Por ejemplo, podría ser $e(\mathrm{x}_k) = \|\mathrm{x}_k - \mathrm{x}_k{}^*\|$. Saarinen y Cybenko [61] hicieron un estudio de la tasa de convergencia de algunos de los métodos descritos en este trabajo (descenso directo en el gradiente, de Newton, gradiente conjugado, etc.), y plantean que el descenso en el gradiente común tiene una tasa de convergencia q-lineal, con un error asintótico constante igual a $(k-1)/(k+1)$ siendo k el número de condición de

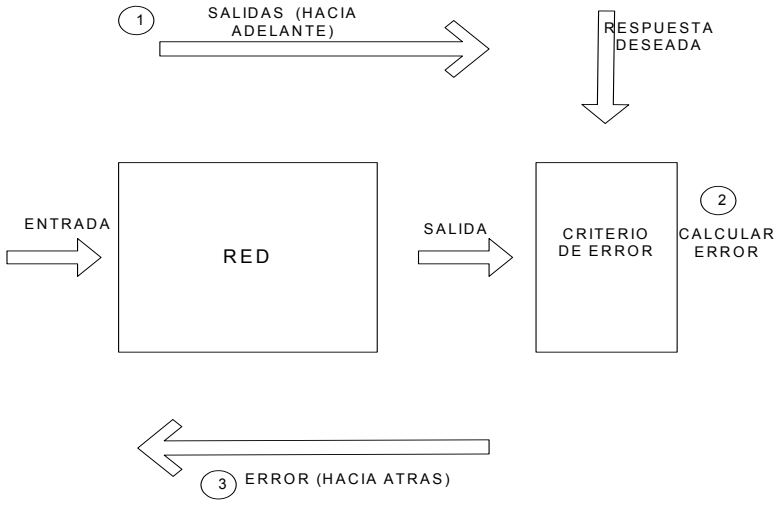
$\mathbf{H}(\mathbf{x}^*)$, $\mathbf{H}$ la matriz hessiana de E y $\mathbf{x}^*$ el mínimo global de la función de error. Adicionalmente, plantean que en ciertos casos (como cuando el jacobiano o la hessiana son "ill-conditioned"[3] en algunos de los puntos de iteración), el descenso directo en el gradiente no converge cuando se lo implementa en una computadora.

.

Gori y Maggini [29] demostraron que el algoritmo en su versión incremental converge a una solución óptima cuando se lo utiliza para entrenar una red en clasificación de patrones, si estos son linealmente separables, es más, lo hace sin importar la tasa de aprendizaje usada.

Las dos pasadas de cálculo

Podemos observar que en la aplicación del algoritmo de "back-propagation" en su versión incremental se realizan dos pasadas de cálculo: la primera es referida como la pasada ***hacia delante*** y la segunda, como la ***hacia atrás***. En la pasada hacia delante, los pesos quedan sin cambiar y se calcula la salida de cada neurona, en base a la entrada que les presentan (es decir, en base al patrón de entrada). Luego de haber calculado los valores de salida de cada neurona de salida, los comparamos con los valores deseados y obtenemos así el error para ese patrón en cada neurona de salida. Entonces comienza la pasada hacia atrás (hacia la izquierda, en nuestro grafo de red), capa por capa, y calculando el gradiente local para cada neurona y los cambios en los pesos, siempre desde la capa de salida hacia la entrada. Gráficamente es:



El algoritmo de interconexión

Este algoritmo es una implementación del algoritmo general de back-propagation incremental y se basa en usar dos redes, la original en donde se propagan los datos de entrada y la dual, en la cual se propagan los errores. Fue codificado por primera vez por Lefebvre (1991) (Ver [11]).

Si consideramos la red por capas, el **x** (entrada) en una capa es la salida (**y**) de la capa anterior, yendo de izquierda a derecha. Es decir $x_i^{capa\ L+1} = f(\sum_j w_{ij} x_j^{capa\ L})$.

[3] Una matriz se dice "ill-conditioned" cuando el cociente entre el mayor y menor valor singular en su descomposición en valores singulares es muy grande.

Gráficamente podemos plantear esto como:

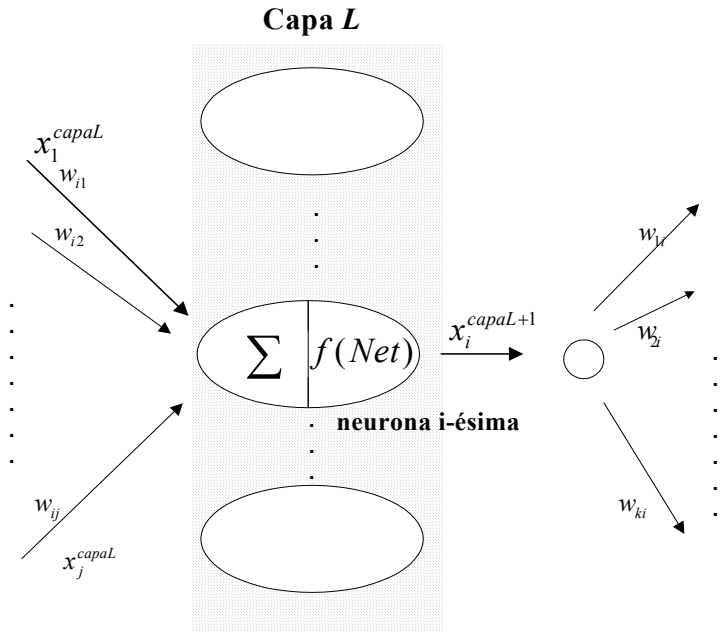


Figura II.1

donde el círculo representa un punto de bifurcación o separación de caminos.

Por otra parte, si expresamos las ecuaciones de actualización de pesos como

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

tomando $y_i(n) = s_i(n)$ o $y_i(n) = z_i(n)$ según sea la capa de salida u oculta respectivamente, entonces

$$\delta_i^{capa\ de\ salida}(n) = \varepsilon_i(n) f'(Net_i^{capa\ de\ salida}(n)) \quad \text{para la capa de salida}$$
$$\delta_i^{capa\ L}(n) = f'(Net_i^{capa\ L}(n)) \sum_k \delta_k^{capa\ L+1} w_{ki}(n) \quad \text{para las demás capas} \qquad \textbf{(II.1)}$$

Por definición, δ_i es el ***gradiente local*** [5] en la neurona i-ésima. Se cumple que

$$\delta_i(n) = \frac{\partial E(n)}{\partial Net_i(n)} = e_i(n) f_i'[Net_i(n)] \qquad \textbf{(II.2)}$$

siendo $e_i(n)$ el error en la salida de la neurona i en el instante n.

Podemos representar las ecuaciones II.1 y II.2 como:

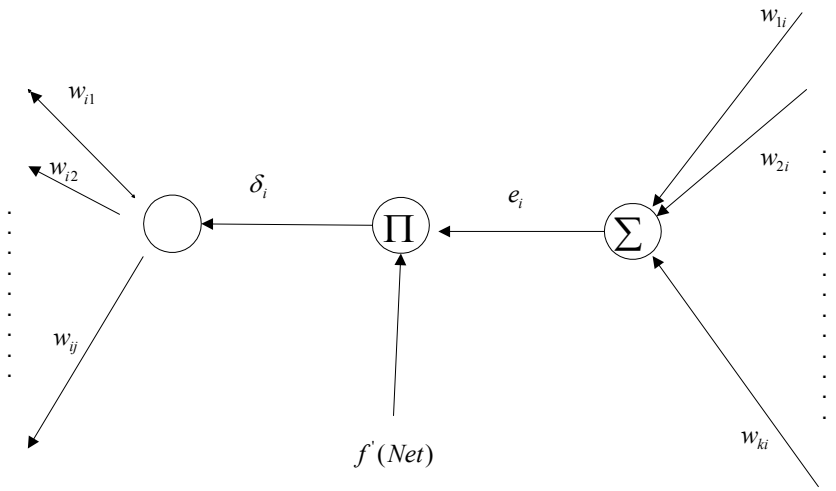


Figura II.2

La red de la figura II.2 es por definición la ***red dual*** (o adjunta) de la de la figura II.1.

La relación que existe entre ambas redes es: en la neurona i-ésima de la red original el flujo de las entradas (x_i) va de izquierda a derecha, mientras que en la dual el gradiente local (δ_i) va de derecha a izquierda, esto es, las entradas se convierten en salidas y recíproco. Además, los puntos donde se suma ($\sum$) en la red original se convierten en nodos donde se hace una separación de caminos en la dual, y a su vez los nodos donde se separaban los caminos se convierten en puntos de unión donde se suma. Los pesos de las conexiones correspondientes mantienen los mismos valores en ambas redes. El error de entrada en la red dual es multiplicado por $f'(Net_i)$ para producir δ_i. Por otra parte el gradiente local es proporcional al error local (ec. II.2). La conclusión de estas observaciones es que podemos imaginar que el error $\varepsilon_i = d_i - y_i$ producido a la salida de la red original es inyectado en la capa de entrada de la red dual y permite calcular el gradiente del error de la red original como salida de los nodos de la dual.

Por lo tanto, podemos re-escribir el algoritmo de "back-propagation" como sigue:

Paso 1 *– Inicializar los pesos de la red con valores pequeños al azar.*

Repetir *los siguientes pasos hasta que se obtenga el error deseado o se verifique la no convergencia:*

Paso 2 *– Elegir un patrón del conjunto de entrenamiento al azar, presentarlo a la red y obtener la salida de cada neurona de la red.*

Paso 3 *– Inyectar el error calculado a la salida de la red original a través de la red dual y calcular el gradiente local en cada neurona.*

Paso 4 *– Actualizar los pesos según la regla de búsqueda elegida. Por ejemplo, para el descenso directo en el gradiente, es:*

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

Fin Repetir

Vemos que el gradiente local esta disponible como un valor (señal) en los nodos de la topología dual y nos evita calcular la expresión $\delta_i^{capa\ L}(n) = f'(Net_i^{capa\ L}(n)) \sum_k \delta_k^{capa\ L+1} w_{ki}(n)$. Solo es necesario especificar la topología de la red ya que el flujo de los gradientes a través de la red dual hace los cálculos de la retro-propagación por nosotros.

Implementar el algoritmo de BP usando la red dual es mucho más versátil que codificar directamente las ecuaciones del mismo, ya que la red dual puede ser programada muy simplemente a partir de la topología original especificada. Como se verá más adelante, BP puede ser adaptado para entrenar redes recurrentes y con aprendizaje temporal, siendo el algoritmo de interconexión también generalizable a esos casos [11]. Es más, el algoritmo puede ser aplicado a una topología arbitraria, siempre que se pueda construir la red dual. Según [11], es la mejor manera posible de implementar "back-propagation" con un algoritmo, y es el método que el simulador de redes neuronales elegido utiliza, aunque no se dispone de información de cómo se implementan los dos flujos de datos.

En resumen, la ventaja de esta implementación del algoritmo es que nos evita tener que hacer los cálculos utilizando las funciones de salida para simular una red: el gradiente local siempre esta disponible en los nodos de la red dual.

Incremental vs. Batch

Una alternativa al aprendizaje incremental descrito antes es usar ***aprendizaje por lotes*** (o "***batch"***) en el cual la actualización de los pesos es realizada solo luego de que todos los patrones de entrada hayan sido presentados y calculadas todas las variaciones necesarias de los pesos, pero sin aplicarlas (recordar que estamos trabajando con un conjunto de entrenamiento finito). Formalmente, el aprendizaje por lotes se plantea sumando las partes derechas de las ecuaciones de cambio de los pesos, para todas las entradas $\mathbf{x}_k \quad k = 1,\ldots m$:

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

para las neuronas ocultas

$$\Delta w_{lj} = \sum_{k=1}^{m} \rho_o [(d_l)_k - (y_l)_k] f_o'[(Net_l)_k](z_j)_k$$

para las de salida

donde $(d_i)_k$ significa "el i-ésimo componente de $\mathbf{d}_k = [(d_1)_k, (d_2)_k \ldots (d_L)_k]$ y $(Net_j)_k$ significa el valor de Net_j para el patrón k-ésimo. Esto equivale a un descenso según el gradiente en la función objetivo

$$E(\mathbf{w}) = \tfrac{1}{2} \sum_{k=1}^{m} \sum_{l=1}^{L} [(d_l)_k - (y_l)_k])^2 \qquad \textbf{(II.3)}$$

Aunque la actualización en "batch" mueve el punto de búsqueda **w** en la dirección del verdadero gradiente en cada paso de actualización, la actualización incremental es más deseable por dos motivos: cuando se implementa el algoritmo en una computadora requiere menos almacenamiento de datos (ya que no es necesario mantener información sobre la variación acumulada de cada peso) y además hace aleatorio el camino de búsqueda en el espacio de pesos al tomar el patrón (vector) de entrada al azar entre los pares {**x**, **d**} de entrenamiento lo que permite una mayor exploración del espacio de búsqueda y, potencialmente, lleva a soluciones de mejor calidad. Usando la teoría de aproximación estocástica (Robbins-Monro) Finoff (citado por [4]) mostró que, para tasas de aprendizaje próximas a cero, la retro-propagación incremental se aproxima a la batch y produce esencialmente los mismos resultados. Además, para constantes de aprendizaje pequeñas, el elemento estocástico en el proceso de entrenamiento da al "back-propagation" incremental un carácter de casi-temple ("quasi-annealing") en el cual el gradiente acumulativo (es decir, el de la función E(**w**) de II.3) es perturbado continuamente, permitiendo a la búsqueda escapar de los mínimos locales con llanos de atracción ("basins") aplanados y pequeños (Ver [4]). Adicionalmente, cuando los datos de entrenamiento son redundantes (es decir, el juego de datos contiene varias copias del mismo par {**x, d**}) se ha encontrado que el modo incremental obtiene ventajas de la presentación de un patrón a la vez y de esa redundancia (Ver [5]).

Por todo ello, tal como se verá en la Sección III, siempre se trabajó con entrenamiento incremental.

II.1.3 Variantes del algoritmo

> Generalmente el aprendizaje con el algoritmo de retro-propagación es lento ([4] y [5]), debido a la forma de las superficies de error, caracterizadas por numerosas regiones aplanadas o con mucha pendiente, teniendo incluso áreas que son planas en la dirección de la búsqueda; se han hecho numerosos estudios para casos concretos de superficies de error. Estos problemas se ven particularmente agravados cuando la red realiza tareas de clasificación, especialmente cuando el tamaño del conjunto de entrenamiento es reducido [4]. Por ello se han propuestos muchas variaciones al algoritmo, que son en su mayoría modificaciones heurísticas que buscan aumentar la velocidad de convergencia, evitar los mínimos locales y/o mejorar la capacidad de la red de generalizar .

Mencionaremos a continuación algunas de esas modificaciones.

De la convergencia hacia el mínimo local

Estas son mejoras que buscan acelerar la velocidad de convergencia hacia un mínimo, seguramente local.

De la tasa de aprendizaje

La velocidad de la convergencia del back-propagation esta directamente asociada a la tasa de aprendizaje (ρ_o y ρ_h en las ecuaciones) por lo cual se han ideado algunas técnicas (heurísticas) para elegir su valor correcto. Algunas tratan de determinar el valor de la tasa de cada neurona en forma estática, es decir, independientemente de la etapa del aprendizaje en que se esté, por ejemplo, tomando como tasa el inverso de la raíz cuadrada del número de conexiones (o "***fan in***") que llegan a la neurona [5 Cap. 4]. Otros, han sugerido ir variando la tasa a medida que pasa el tiempo, por ejemplo,

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

con τ una constante positiva y t el número de iteración (Ver [4]).

Finalmente, algunos (como por ejemplo Almeida, citado por [11]) aumentan o disminuyen la tasa según que se vaya aproximando al mínimo o se pase sobre él. El algoritmo de Almeida utiliza una tasa de aprendizaje para cada peso y la idea es, si la componente del gradiente según ese peso tiene el miso signo en dos iteraciones consecutivas, se debe aumentar la tasa, y si cambia, se la debe disminuir. Dicho aumento se hace en forma geométrica (es decir, multiplicando por un factor. Ver [10]).

Del uso de momentos

Se ha sugerido añadir un término adicional del desarrollo de Taylor de $E(\mathbf{w})$ usado para calcular $\Delta\mathbf{w}$. A dicho término (de segundo orden), o a una aproximación a él, se lo suele llamar ***momento de primer orden*** [51] o simplemente ***momento.*** Con esto lo que se busca es acelerar la convergencia en regiones donde la superficie de error es casi plana y escapar a los mínimos locales.

La regla de aprendizaje usando momento de primer orden será entonces:

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

donde $0 < \alpha < 1$ es la ***tasa de momento***. Esta tasa puede ser constante en el tiempo o ir cambiando dinámicamente con la iteraciones (Ver [4]).

El uso de momentos está justificado por el hecho de que son aproximaciones a los métodos de segundo orden, tales como el ***método de Newton.*** En él se acutalizan los pesos como $\Delta \mathbf{w} = -\left(\mathbf{H}(\mathbf{w}^{actual})\right)^{-1} \nabla E(\mathbf{w}^{actual})$ obteniéndose el llamado. Como realizar este cálculo es computacionalmente intenso (del orden $\boldsymbol{O}$ (N^3) siendo N el número de pesos), se han sugerido varias heurísticas y aproximaciones para tener métodos de segundo orden disminuyendo el número de operaciones (Ver [4]).

La tasa de momento puede ser constante o ir variando con las iteraciones. Por ejemplo, supongamos que queremos ajustar ahora α en cada paso de la actualización de forma que la búsqueda según el gradiente sea localmente óptima, es decir, que deseamos calcular un ρ en el tiempo t tal que el error E en el tiempo $t+1$ sea mínimo. En ese caso, la tasa de momento que se obtiene [4] es

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

En forma más general, se pueden agregar más términos del desarrollo de Taylor de $E(\mathbf{w})$; por ejemplo, los de tercer orden de dicho desarrollo (llamados ***momentos de segundo orden*** [51]).

Mientras que el algoritmo en su forma "batch" y original con su planteo: $\Delta w(t) = -\rho \frac{\partial E}{\partial w(t)}$ converge a un mínimo de la función error (considerando como tal la suma de errores cuadráticos) en un tiempo t no menor a $(\lambda_{\max} / \lambda_{\min})/4$ donde $(\lambda_{\max} / \lambda_{\min})$ es el cociente entre el mayor y el menor de los valores propios de la matriz hessiana de E respecto $\mathbf{w}$ (es decir, si E_{min} es el mínimo local que se busca, E - E_{min} tiende a 0 más lentamente que $e^{-4t\frac{\lambda_{\min}}{\lambda_{\max}}}$ cuando t tiende a infinito), con el agregado de un momento de primer orden esto se mejora a $[\sqrt{(\lambda_{\max} / \lambda_{\min})}]/4$ también para el caso "batch". Sin embargo, si se agrega un momento de segundo orden, esta cota casi no baja, es más, la ventaja relativa tiende a 0 cuando $\lambda_{\max} / \lambda_{\min} \to 0$ (Ver [51]).

De los gradientes conjugados

Estos métodos se basan en consideraciones geométricas y han sido probados por varios autores en redes "feedforward" multicapa demostrando ser mejores al "back-propagation" original en velocidad de convergencia [4]. Lo que se hace es buscar el mínimo según direcciones tales que la dirección de búsqueda en la iteración k, d_k, es conjugada[4] , respecto a la hessiana de E , con todas las anteriores $d_1 \ldots d_{k-1}$, empleando para ello un proceso similar al de ortogonalización de Gram-Schmidt. Cada dirección de búsqueda estará dada por

$$d_k = -\nabla E\big|_{\mathbf{w}_k} + \beta_k d_{k-1}$$

siendo β_k un escalar: según como se lo calcule se obtendrán distintas variantes del método ([7],[1]); así por ejemplo se obtiene la fórmula de Fletcher-Reeves tomando

[4] Es decir que $d_j^T \mathbf{H} d_j = 0$

$$\begin{cases} \beta_k = \dfrac{\nabla_k^T \nabla_k}{\nabla_{k-1}^T \nabla_{k-1}} \\ \nabla_k = -\nabla E\big|_{\mathbf{w}_k} \end{cases}$$

De la función de salida

La idea aquí es evitar la saturación prematura de las neuronas (parálisis de la red). Para ello, se puede variar (perturbar) la derivada de la función de salida o alterar algún parámetro que defina la función misma, de forma que se actualicen más rápidamente los pesos en las primeras etapas del entrenamiento, para luego, ir haciendo desaparecer esa perturbación a medida que se alcanza la convergencia.

Otros criterios de convergencia

La utilización de otros criterios de convergencia, tales como funciones de error distintas a la suma de diferencias cuadráticas, implicará variaciones en el tiempo que se tarde en llegar a la solución deseada. De la misma forma, el uso de la validación cruzada ("cross validation") altera el tiempo de ejecución del entrenamiento, a la vez que permite obtener mejores resultados finales, en cuanto a la capacidad de generalización. (Ver ANEXO I).

El uso de otras funciones (de criterio) de error para evaluar el error que se esta cometiendo nos lleva a tener otros criterios de convergencia del algoritmo. Para ser lo más generales posibles la función de error que se use debe cumplir con ciertas condiciones básicas: las que caracterizan a una medida (o al menos ser no negativa, y nula si y solo si los dos puntos son idénticos), y además, para poder aplicar el algoritmo, debe ser derivable con derivada continua (ver ANEXO I). Como casos concretos, podemos citar las funciones de grado r de Minkowsky $E(\mathbf{w}) = \frac{1}{r}\sum_{i=1}^{n} |d_i - y_i|^r$ con $r > 0$ e i variando sobre el número de neuronas de salida. Cuando $r = 2$, se obtiene el conocido error cuadrático [5](digamos, a menos de una constante ½). Cuando $r = 1$, se obtiene la llamada ***norma Manhattan***, que posee interesantes propiedades matemáticas (ver ANEXO I). También se pueden utilizar medidas de error basadas en la entropía relativa ([4], [18], [16]).

Búsqueda del óptimo global

Dado que la convergencia del algoritmo de retro-propagación es hacia un óptimo que no tiene porqué ser el global, es evidente que una mejora interesante al mismo puede ser la búsqueda de "el" mínimo. Daremos a continuación una breve reseña de algunas de ellas. En el ANEXO II se pueden encontrar más detalles.

La búsqueda aleatoria en el gradiente

Esta técnica permite explorar mejor el espacio de pesos tratando de evadir los mínimos locales. Básicamente lo que se hace es añadir un ruido a los pesos que irá desapareciendo progresivamente. El método de aprendizaje que se obtiene así (llamado de Langevin) tiene una serie de propiedades interesantes: desde el punto de vista teórico, se asemeja a las técnicas de temple simulado ya que da una baño de calor a la red (el ruido que se agrega representaría la temperatura de ese baño de calor, el cual luego va desapareciendo para producir el templado) que permite obtener configuraciones de los pesos que escapen de los mínimos locales [60] y desde el punto de vista práctico, no tiene grandes requerimientos computacionales y puede ser usada junto con otras variaciones del "back-propagation".

Este método, junto con los algoritmos genéticos y el temple simulado, como se verá a continuación, trata de escapar de los mínimos locales. En este trabajo se utilizó una función del simulador de redes neuronales (llamada "jog") que suma

[5] Estamos considerando los errores instantáneos, es decir, para el patrón recién presentado en la entrada.

a cada peso, en la iteración del entrenaminto que se desee, un ruido uniforme de media el valor actual del peso respectivo y varianza un cierto valor que se especifica para todos los pesos.

Métodos no basados en derivadas ([7], [71])

Los algoritmos evolutivos (por ejemplo, los genéticos), junto con el temple simulado, son métodos de optimización llamados "sin derivada". Estos métodos permiten utilizar funciones de salida no diferenciables ya que la función de "fitness" usada por ellos no necesita serlo. Esto permite plantear funciones de error tan complejas como se requieran sin que ello influya demasiado en el tiempo de procesamiento computacional insumido.

Usando algoritmos evolutivos, se puede llegar a obtener mejores velocidades de convergencia que el temple simulado, aunque de todos modos son más lentos que los métodos basados en derivadas ([7]).

En especial, los algoritmos evolutivos permiten búsquedas en paralelo, que pueden ser implementadas en máquinas que tengan varios procesadores o que trabajen independientemente, para acelerar así masivamente el proceso. También pueden ser aplicados a problemas continuos o discretos, así como permiten la identificación tanto de la estructura de la red como de los parámetros (pesos) en los modelos complejos ([7]).

Sin embargo dado que los algoritmos genéticos hacen búsquedas en direcciones aleatorias, puede suceder que encontrar la solución óptima global requiera un tiempo considerable, si es que no lleva uno que este fuera de los límites admisibles para hallar la solución al problema. Además, se hace difícil realizar estudios analíticos de ellos, en parte debido a su aleatoriedad, por lo que la mayoría del conocimiento que se tiene de estos algoritmos proviene de estudios empíricos [7].

Existen dos formas de aplicar los algoritmos evolutivos para entrenar una red:

1) ***directamente***, tratando de hacer evolucionar un conjunto de pesos $\{\mathbf{w}_j\}$ que minimicen la función de error. Esto puede llevar a un proceso lento e ineficiente, y requerir grandes recursos de almacenamiento. El entrenamiento sería una búsqueda genética de los pesos óptimos

2) ***en forma de algoritmo híbrido***: en ese caso se considera la red dividida en dos subredes: se aplica descenso en el gradiente para la red formada por las capas oculta-de salida, y en la correspondiente a la formada por las capas entrada-oculta, se aplica un AG para los valores de salida de la capa oculta sean tales que aplicados a la entrada de la red de capas oculta-salida produzcan la salida deseada. La solución converge más rápidamente que la anterior (Ver ANEXO II y [4]).

Eliminación de pesos y neuronas

A partir del descubrimiento por parte de Baum y Haussler ([18]) de que para obtener una buena capacidad de generalización la cantidad de patrones de entrenamiento debe ser considerablemente mayor que el número de pesos [6] en la red, surge la idea de reducir el número de dichos pesos, simplificando así la estructura de la red y mejorando la capacidad de generalización para un mismo conjunto de entrenamiento. Esto es especialmente útil cuando se disponen de pocos datos de entrenamiento. Por otra parte, dado que cada neurona que tiene la red requiere cierto tiempo de entrenamiento, eliminar algunas junto con los pesos asociados aumenta la velocidad de aprendizaje. Ya que es difícil de estimar a priori el número óptimo de neuronas o pesos que deben eliminarse (o retenerse, según como se lo mire), se incluye el estudio de algunas técnicas automáticas de reducción de pesos, lo que luego dará un criterio para eliminar neuronas. Básicamente, lo que se hace es incluir en la función de error una penalización a los pesos distintos de 0, o distintos entre sí, para que , en las etapas sucesivas del entrenamiento, esos pesos vayan disminuyendo de valor absoluto en forma más que lineal , si es que no hay casi aprendizaje en la neurona para ese peso (o sea si ese peso casi no variaría usando el entrenamiento normal). Cuando todos los pesos que llegan a una neurona que no es de entrada (o los que salen de una que no es de salida) son casi nulos, esta se convierte en redundante y puede ser eliminada. A estas

[6] En realidad, es la cantidad de pesos independientes, lo que puede verse grados de libertad del problema de optimización asociado. Los pesos pueden igualarse o relacionarse entre sí para disminuir esa cantidad, utilizando técnicas como el "weight sharing" (ver ANEXO VI).

técnicas se las llama ***algoritmos de podado***. Recíprocamente, se puede comenzar con una red muy pequeña, y luego irla enriqueciendo progresivamente, obteniéndose los ***algoritmos de crecimiento*** ("growing algorithms") ([5], [1]).

Incorporación de ruido a los valores deseados

Hasta el momento se ha expuesto la incorporación de ruidos en distintas partes de la red (en los pesos, como en el caso de Langevin, o implícitamente en la entrada, al usar "back-propagation" incremental) para mejorar la convergencia. Otra opción es añadir ruido a los valores deseados y minimizar el error "instantáneo" (el correspondiente al patrón presentado en la entrada) en lugar del esperado definido como la suma de los errores cuadráticos: para cada valor deseado $d_i(t)$ se toma un nuevo valor deseado $d_i(t) + n_i(t)$ donde los n_i son ruidos con distribución normal o uniforme, con media 0 y varianza σ^2 , independientes tanto de los valores de entrada $x_i(t)$ como de los $d_i(t)$, y se calcula $\varepsilon_i = \frac{1}{2}\sum_i [d_i(t) - y_i(t)]^2$ (Ver [67]).

Se puede demostrar que, en esas condiciones, y usando funciones de salida de Borel (tales como la sigmoide, la Tanh y en general cualquier función continua, que son las comúnmente usadas en redes neuronales) los valores finales de los pesos no son afectados en un sentido estadístico:

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

Es más, esta conclusión es válida para *cualquier* arquitectura de red, habiéndose probado con redes estáticas y dinámicas, obteniéndose buenas performances [67].

Por otra parte, esta forma de agregar ruido tiene como ventaja práctica de permitir seguir usando los simuladores de redes existentes, ya que no afecta en ninguna forma la implementación del algoritmo de retro-propagación usado, al solo modificar los valores deseados

II.2 Comités de redes

Una forma de mejorar la performance de redes neuronales es usar varias redes independientes de diferente tamaño y características para resolver un mismo problema. Esa idea proviene de investigaciones realizadas sobre las propiedades del estimador (en nuestro caso, el predictor del valor a obtener) que se obtiene al promediar otros estimadores obtenidos en forma independiente [55]. La aplicación de estas ideas lleva a resultados interesantes especialmente cuando se dispone de un conjunto reducido de datos de entrenamiento. Cuando las diferentes redes se integran formando otra red, se obtiene una ***red modular*** ([12] y [11]). Estas redes poseen la ventaja de que, a igualdad de neuronas con una red común, poseen menos pesos ya que no existe conectividad total entre todas sus neuronas. Por lo tanto, el aprendizaje de estas redes es más rápido y se requieren menos ejemplares de entrenamiento. Por otra parte, no existe casi investigación sobre cómo se puede dividir una red "feedforward" en módulos en forma óptima [11].

Supongamos que entrenamos *C* redes diferentes con los mismos datos. Si eligiésemos la red que produce el mejor error en el conjunto de entrenamiento, no estaríamos tomando la mejor decisión ya que, primero, se desperdiciaría el entrenamiento de las otras redes (es decir, estaríamos descartando la información potencialmente útil almacenada en las otras redes), segundo, dado que el conjunto de "cross validation" es elegido al azar, existe una probabilidad no nula de que alguna otra red del conjunto tenga una mejor performance que la que elegimos cuando se trate de datos no vistos previamente, correspondientes a la misma función de distribución de probabilidad. Una estrategia mucho mejor es usar todas las redes entrenadas, o sea, utilizar un ***comité de redes.*** Más formalmente, un comité de redes es "un conjunto de redes neuronales entrenadas con los mismos datos, cuyas topologías suelen ser diferentes, y cuyas salidas son interpretadas como un voto para la clasificación (o un valor sugerido para la predicción)" [11]

II.2.1 El "ensemble method"

El error medio cuadrático de tomar la solución promedio de las redes (es decir, el promedio de los valores de salida $y_{COMITE} = \frac{1}{C}\sum_{i=1}^{C} y_i$) suponiendo que los errores de cada red tienen media 0 y no están correlacionados entre sí será (Perrone [55] y citado por [11]):

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

donde $\overline{E}$ es el promedio de los MSE las redes trabajando individualmente. Por lo tanto, el error cometido por el comité es C veces más pequeño que el promedio de los MSE, por lo que aumentando el tamaño de la población de redes que usamos podemos hacer que el error del comité sea tan pequeño como se quiera. Si bien esto es algo optimista, ya que en general los errores entre las redes no están incorrelacionados y en ese caso no se obtienen mejoras tan drásticas, se puede demostrar que de todos modos el error E_{COMITE} nunca es mayor que $\overline{E}$ (Ver [1]).

II.2.2 Ventajas de los comités

Una propiedad interesante de este método es que si bien estamos tomando las estimaciones como generadas por redes neuronales, las mismas podrían ser generadas por otros modelos (por ejemplo estadísticos), y en el caso de que fuesen todas redes neuronales, se podría estar incluso trabajando sobre distintos conjuntos de entrenamiento. Esto último tiene una implicación importante que pasamos a describir. Un método estándar de evitar el "overfitting" durante el entrenamiento es usar un conjunto de "cross validation". El problema es que dado que usamos ese método para evitar el "overfitting", cada estimación de la regresión "ve" solo parte de los datos y puede estar perdiendo información importante sobre el resto de la distribución de los datos, especialmente si el conjunto total de datos es pequeño. Este va a ser siempre el caso de una predicción (estimación) que use "cross validation" como regla de detención en una sola red. Sin embargo, usando el método de comités que hemos presentado (el ***"ensemble process"***, [55]), al construir el conjunto de estimadores (redes), podríamos entrenar cada una con el conjunto entero de datos y dejar que la propiedad de suavizado [7]del "ensemble process" remueva cualquier "overfitting". De esta forma, las diferentes redes habrán visto todos los datos mientras que si usáramos el método de "cross validation" para evitar "ovefitting" esto no pasaría. [55]. Otra ventaja que se obtiene es la reducción de la varianza del error debido al promediar cada salida individual [5].

II.2.3 Mejoras posibles al trabajar con comités de redes

Bishop ([1]) demuestra que para que el error del comité al ponderar sea mínimo los pesos de ponderación p_i deben ser tales que

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

siendo $\mathbf{C}$ la matriz de correlación de los errores de cada red:

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

donde $\varepsilon_i = d_i - y_i$, X es la variable aleatoria que representa todos los patrones de entrada posibles y el promedio es tomado considerando todas las condiciones iniciales posibles de la red.

En especial, aproximando $\mathbf{C}$ a partir de los datos existentes es

[7] Reducción del error en $1/C$ respecto al promedio de errores MSE

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

siendo n el número de patrón e y_{ik} el valor obtenido de salida para la red i-ésima con el patrón k-ésimo.

II.3 Entropías e información mutua

Se describen a continuación algunas de las propiedades de la entropía, así como de la entropía relativa e información mutua. Estos conceptos (especialmente el de información mutua y la entropía relativa) aparecerán al estudiar las propiedades de la serie temporal, como función de error a utilizar y cuando se comparen los modelos obtenidos para predecir la serie temporal.

II.3.1 La entropía

En términos informales, la entropía es una medida de la incertidumbre de una variable aleatoria. Sea X una variable aleatoria discreta, con un conjunto de valores posibles $\boldsymbol{X}$. Sea la función de probabilidad (densidad) $P(X = x) = p(x) \quad x \in X$. Se define la entropía absoluta (o entropía de Shannon o simplemente ***entropía***) de X como

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

Observaciones:

- La entropía se mide en bits, si los logaritmos se hubiesen tomado naturales, la entropía se mediría en ***nats***.
- Se utiliza la convención $0log0 = 0$. Es decir, agregar términos con probabilidad 0 no cambia la entropía.
- La entropía no depende de los valores de X, sino de sus probabilidades.

Una de las aplicaciones de la entropía es que el largo del código binario (descripción) más corto de una variable aleatoria está entre $H(X)$ y $H(X)+1$ (Ver [2]).

Otra interpretación de la entropía

Sea X una variable aleatoria discreta. La ocurrencia de cada uno de sus valores $x_k \in X$ puede ser considerada como un mensaje, es decir, consideramos esa ocurrencia como el mensaje "X toma el valor x_k". Ese mensaje puede ser más o menos esperado (o sea, su contenido puede causarnos más o menos "sorpresa"). La información que contiene ese mensaje es una medida de su aleatoriedad: un mensaje totalmente predecible no contendrá información. Sea $P(x_k) = P(X = x_k)$ la probabilidad de que X tome cada uno de esos valores x_k. Definimos la información contenida en el mensaje x_k como

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

y la entropía de X como

$$H(X) = \sum_{x_k \in \boldsymbol{X}} P(x_k) I(x_k) \qquad \textbf{(II.4)}$$

Algunas propiedades

De II.4 se deduce que la entropía $H(X)$ es un promedio de la información contenida en la variable aleatoria X [11]. La entropía está acotada por

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

siendo N el número de valores que puede tomar X. O sea que es máxima cuando P(x_k)= 1/N o sea cuando hay máxima incertidumbre y es mínima cuando P(x_k)= 1 para algún k, o sea cuando hay certeza.

II.3.2 La entropía relativa

La entropía de una variable es una medida de la incertidumbre que se tiene sobre ella. Por otra parte, la entropía relativa es una medida de la distancia entre dos distribuciones de probabilidad y es también una medida de la ineficiencia de suponer que la distribución de X es q cuando la verdadera es p. Definimos la ***entropía relativa*** (o entropía cruzada o distancia de Kullback-Leibler) entre dos funciones de probabilidad $p(x)$ y $q(x)$ de la variable aleatoria discreta X con valores en $\boldsymbol{X}$ como [2]:

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

Utilizamos aquí nuevamente las convenciones $0Log(0/q) = 0$ y $pLog(p/0)=\infty$

La entropía relativa es siempre no negativa y es 0 si y solo si $p = q$.

Aplicación: su uso como función de error

Cuando las salidas deseadas y reales de la red son vectores de probabilidad es razonable usar la entropía relativa como función de error: $D(X \| Y)$ es siempre positiva, convexa en las dos variables y se anula si y solo si $x = y$. Para ser estrictos, no es una distancia, ya que no es simétrica ni satisface la desigualdad triangular, aunque esta consideración se podría obviar tomando como distancia $d = D(X \| Y) + D(Y \| X)$, pero esto no es necesario en la práctica. En ese caso tomamos entonces $D(X \| Y) = \sum_{x_i \in \boldsymbol{X}} x_i \log_2 \frac{x_i}{y_i}$. En [17] se detalla un caso práctico donde se la utilizó.

Si las salidas de la red suman 1 (o, para utilizar la nomenclatura del simulador de redes, son "softmax"), entonces se demuestra que el criterio de la entropía relativa puede ser implementado por el criterio MSE (es decir, minimizar la entropía relativa es equivalente a minimizar el MSE entre los valores de salida reales y los deseados) [11].

En [16] se plantea que, para el caso $y_i = x_i + \varepsilon_i \quad \forall i$, entonces es

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

II.3.3 La entropía condicional

Se define la entropía condicional entre dos variables aleatorias X e Y discretas como:

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

II.3.4 La información mutua

La ***información mutua*** es, en primera aproximación, una medida de la información que una variable aleatoria contiene sobre otra, o la reducción de la incertidumbre de una variable aleatoria debido al conocimiento de la otra. La entropía se vuelve entonces la información que una variable aleatoria tiene sobre sí misma. Más formalmente, sean X e Y dos variables aleatorias con una densidad de probabilidad conjunta $p(x, y)$ y probabilidades marginales $p(x)$ y $p(y)$. La ***información mutua*** $I(X, Y)$ es la entropía relativa entre las distribuciones conjuntas y el producto de las distribuciones marginales $p(x)$ y $p(y)$ [2]:

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

Aplicación: el cálculo del "delay" óptimo

La información mutua puede ser usada para determinar el valor del "delay" óptimo (ver AVII.4.4) para la reconstrucción del espacio de estados. La ***información mutua demorada en el tiempo*** es un caso particular de información mutua aplicada a los valores de una variable $X(t)$ y a su versión $X(t+T)$ desfasada en el tiempo. A diferencia de la función de autocorrelación, la información mutua tiene en cuenta las correlaciones no lineales [32]. Lo que se hace es calcular

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

donde, para alguna partición de los números reales en intervalos $A_1, A_2, \cdots$, p_i es la probabilidad de encontrar un valor de la serie temporal en el i-ésimo intervalo A_i y $p_{ij}(T)$ es la probabilidad conjunta de que un valor de ella pertenezca al intervalo i-ésimo y que otro valor en un tiempo T posterior caiga en el j-ésimo A_j (Ver AVII.4.4). Si graficamos $I(T)$ versus T , $I(T)$ comienza con un valor muy alto ($I(0)$=1) y al aumentar T, $I(T)$ decrece y luego vuelve a crecer. Existen buenos argumentos [32] que sugieren que el valor del delay donde $I(T)$ alcanza el primer mínimo sea usado como "delay" óptimo del espacio de reconstrucción.

II.3.5 Algunas relaciones

Se puede demostrar ([2]) que

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

Por lo tanto, la información mutua es la reducción en la incertidumbre de X debido al conocimiento de Y.

También se demuestra que

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

o sea que X dice de Y tanto como Y de X.

Finalmente, se cumple que

$$I(X,X) = H(X)$$
$$H(X,Y) = H(X) + H(X \mid Y)$$

II.4 Redes de comportamiento dinámico

La serie temporal en estudio (o la que se pretende predecir) es generada a partir de un cierto sistema dinámico. Por otra parte, dicha serie va a ser predicha por otro sistema dinámico, el formado por una (o más) red(es) neuronal(es) entrenada(s) a partir de las observaciones existentes (es decir, la serie dada). Comprender la relación redes dinámicas-sistemas dinámicos es fundamental para el estudio de dichas redes, por lo que en el Anexo VII se incluye una introducción a los sistemas dinámicos.

Las redes estáticas son aquellas que son entrenadas para producir una salida espacial en respuesta a una cierta entrada espacial (significando espacial que no depende del tiempo), es decir, que representan sistemas estáticos. Sin embargo, en muchas aplicaciones se precisa modelar procesos dinámicos donde se requiere que la salida sea una secuencia de valores en el tiempo que corresponda a una cierta entrada que también es temporal. Lo que se desea entonces es un modelo que imite la realidad cambiando adaptativamente sus parámetros para aproximar las salidas observables del sistema real cuando se le dan las mismas entradas que al sistema, es decir, que se comporte dinámicamente. Un posible modelo es el de una red recurrente. Otro modelo sería el de una red con memoria, de tipo TLFN (Ver II.4.6). En esta sección nos dedicaremos a las redes dinámicas y en especial, a las recurrentes. La arquitectura recurrente permite incluir la naturaleza dependiente del tiempo de las asociaciones entre los datos. Adicionalmente, es posible extender la red feedforward multicapa y sus algoritmos de entrenamiento asociados (por ej. BP) al dominio temporal para poder entrenar a estas redes.

II.4.1 Introducción

El comportamiento dinámico referido a los datos

Tanto los problemas de clasificación como los de predicción pueden ser formulados como cierto "mapeo" arbitrario entre dos espacios vectoriales. En el caso estático la dimensión de los datos de entrada define el número de dimensiones del espacio de patrones (por ejemplo si los datos están formados por valores de dos variables independientes, el espacio de entrada es de dos dimensiones). En el reconocimiento de patrones (y también en la predicción) estático(a) la forma en que los datos son tomados del espacio de datos es irrelevante: por ejemplo, el problema de clasificación es el mismo cuando barajamos la presentación de los datos que se van a dar a la red, ya que asumimos que no hay un orden en los "clusters" de datos (o sea, no asumimos un secuenciamiento interno de los datos).

En los problemas temporales las medidas tomadas ya no son un conjunto independiente de muestras de entrada sino funciones del tiempo. Si cambiamos el orden de las muestras, estamos distorsionando la serie temporal $x(n)$, por lo que el orden de las muestras debe ser mantenido en el procesamiento temporal.

Las redes dinámicas vistas como sistemas dinámicos [8]

Dada una cierta red dinámica, supondremos que su entrada es una función del tiempo, $\mathbf{I}(n) = [I_1(n), \ldots I_m(n)]$. Sabemos que cada vez que cambia $\mathbf{I}(n)$ va a pasar cierto tiempo hasta que la salida de la red se estabilice, si es que lo hace. Parece natural introducir dos escalas de tiempo para la red: una asociada a la dependencia temporal de los estímulos externos y otra que describe la respuesta de la red (inercia con respecto a ese estímulo). Para que sea posible

algún tipo de entrenamiento es necesario asumir que los cambios temporales en las entradas externas son lentos comparados con el tiempo que se deja evolucionar la red hacia un estado de equilibrio [8]. Supondremos que este es nuestro caso y que la red tiene tiempo "suficiente" para evolucionar luego de un cambio en las entradas. Por ejemplo, cuando se realiza un aprendizaje de punto fijo (ver II.4.3), se deja evolucionar a la red hasta que alcance un equilibrio, por lo que podemos pensar en ese caso que $\mathbf{I}(n)$ es constante en el período de tiempo que tarda la red en alcanzar dicho equilibrio y que tal punto de equilibrio existe. Lo que hará el algoritmo de entrenamiento es ajustar continuamente los pesos de forma de que la red evolucione hacia algún estado estable cuyos valores en las neuronas de salidas tengan los valores más cercanos posibles a los valores deseados para la entrada. Vemos que no precisamos conocer los valores de salida de las neuronas ocultas en ningún caso. En cuanto a obtener los puntos de equilibrio (atractores) "lo más cercanos posibles" a los valores deseados, lo podemos plantear como la minimización de una cierta función de error, que será función de los valores de salida instantáneos y de los deseados. Por otra parte, todas las redes manejadas en el presente trabajo cumplen que , para una neurona i cualquiera:

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

con $I_i(n)$ la entrada exógena a la neurona i-ésima, con la convención $I_i(n) = 0$ si esa entrada no entrada no existe.

Para verificar esto alcanza por ejemplo ver que

- si $w_{ij} = 0 \quad \forall j \geq i$ se obtiene una red "feedforward" (como las TLFNs).
- el MLP se obtiene haciendo $w_{ij} = 0 \quad \forall j \geq i$ y $\mathbf{I}(n) = 0$
- el caso general corresponde a una red totalmente recurrente (Ver II.4.2).

Es decir, podemos escribir para la neurona i-ésima:

$$y_i(n+1) = G_i[y_i(n), \mathbf{w}(n), I_i(n)] \qquad \textbf{(II.5)}$$

cuando mantenemos las entradas $I_i(n)$ fijas al variar n. Dado que además minimizamos el error mediante un descenso en el gradiente, se cumple:

$$\Delta w_{ij}(n) = -\rho \frac{\partial E}{\partial w_{ij}} \qquad \textbf{(II.6)}$$

(u otra ecuación, correspondiente a la actualización de los pesos). Entonces, si tomamos como estado de la red a las salidas $y_i(n)$ de cada neurona y los pesos $\mathbf{w}(n)$, las ecuaciones II.5 y II.6 describen el comportamiento de la misma como un sistema dinámico. En forma vectorial:

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

Este sistema de ecuaciones no lineales no puede ser resuelto analíticamente debido a su complejidad.

Si consideramos además la dinámica durante el entrenamiento hay que agregar entonces

$$\Delta \mathbf{w}(n) = -\rho \nabla E \Big|_{\mathbf{w}=\mathbf{w}(n)} \qquad \textbf{(II.7)}$$

Por otra parte, la ecuación II.7 representa un sistema de gradientes que puede converger hacia un estado estable ya que la función de error juega el papel de función de Lyapunov (Ver [8]).

Las condiciones de convergencia de las salidas $y_i(n)$ hacia un valor estable no han sido derivadas en forma general, aunque sí se ha hecho para casos particulares (por ejemplo, exigiendo que la matriz de pesos sea simétrica en una red de Hopfield).

II.4.2 La arquitectura recurrente

Definimos a una ***red recurrente*** como aquella que presenta uno o más lazos ("loops") de retroalimentación, es decir, cuya topología puede ser representada por un grafo orientado con ciclos, o considerando los datos, aquella en la que la salida depende de sus salidas anteriores en el tiempo [5]. La red de la figura II.3 es por lo tanto recurrente.

En una red recurrente las neuronas pueden ser de entrada, salida o ambas cosas y en ella los valores deseados de salida se establecen para un conjunto arbitrario de neuronas en tiempos predeterminados.

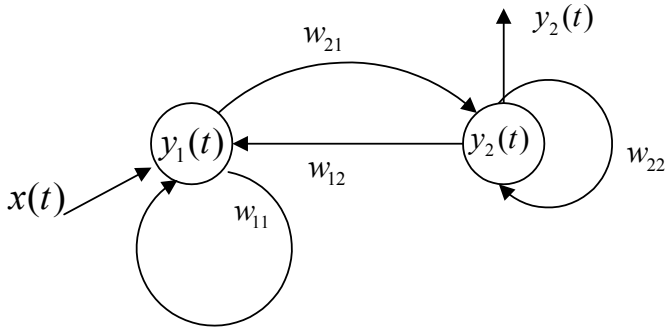


Figura II.3

Redes total y parcialmente recurrentes

Se puede ver en [54] una definición formal de redes total y de redes parcialmente recurrentes. Aquí diremos, siguiendo la definición utilizada por el simulador de redes neuronales elegido:

"*Una red totalmente recurrente conecta la primer capa oculta a sí misma a través de al menos una conexión recurrente. Una parcialmente recurrente añade a la totalmente recurrente al menos una conexión hacia delante desde la capa de entrada a la capa posterior a la primer capa oculta.*"

Podemos ver esto gráficamente en la figura II.4:

Totalmente recurrente
Parcialmente recurrente

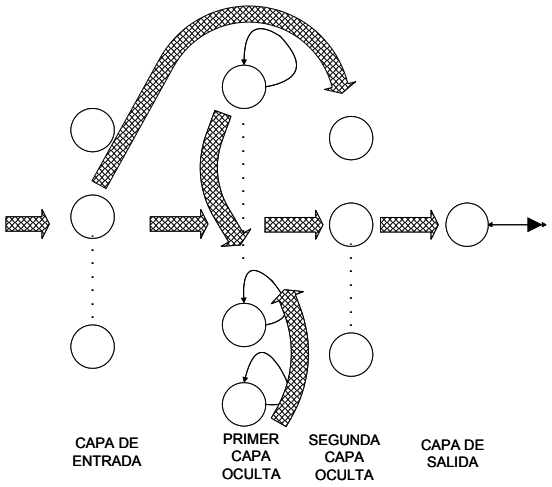


Figura II.4

Una definición alternativa de red totalmente recurrente es [11]:

Si una red cumple que

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

donde I_i es la entrada externa conectada a la neurona i si existe dicha entrada externa, 0 en otro caso y se supone que las neuronas están numeradas de forma de que se cumpla la definición de red "feedforward" para el instante $n+1$ y en los arcos que van hacia atrás ("feed-back") insertamos un "delay" de un "time step" en su salida, entonces se dice que la red es ***totalmente recurrente*** . Por ejemplo, la red de la figura II.5 es totalmente recurrente:

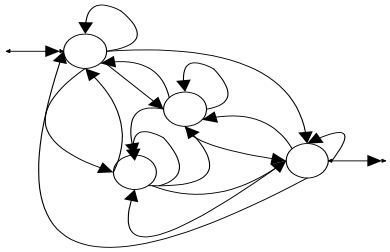


Figura II.5

Redes estáticas vs. redes dinámicas

En las redes estáticas no existe el concepto de ***estabilidad***, o mejor dicho, son siempre estables. Las dinámicas pueden no serlo. La estabilidad se entiende aquí en el sentido uniforme (AVII.1.3): la respuesta de la red a una entrada acotada debe ser un valor finito (acotado). Esto no sucede por ejemplo en una neurona de contexto: si $\tau > 1$ se vuelve ***inestable*** [10].

Las redes estáticas solo tienen memoria a largo plazo ("***long term memory"***) que esta formada por los pesos: la información contenida en los datos es convertida durante el entrenamiento a valores de pesos usando las reglas de aprendizaje. Estas redes contienen un repositorio de la información pasada que nosotros asociamos con memoria, aunque son incapaces de diferenciar las relaciones temporales entre los datos, ya que la información recolectada a través del tiempo es colapsada en los valores de los pesos. Las redes de comportamiento dinámico, o más brevemente, redes dinámicas, en cambio, tienen estructuras de memoria a corto plazo ("***short term memory"***), como en el caso de las "time lagged feedforward networks" (TLFNs, ver II.4.6), o bien conexiones recurrentes, como en el caso de las redes recurrentes, que las hace sensibles a la secuencia en que se presenta la información. Ellas también tienen pesos, por lo que también tienen memoria de largo plazo, pero a diferencia de las estáticas, sus pesos captan las diferencias en el orden de los datos dentro de la ventana de observación.

Las estructuras de memoria a corto plazo permiten convertir los fenómenos dinámicos en estáticos, al manejar una serie de valores D de entrada sucesivos en el tiempo, como un punto de un espacio D-dimensional [11].

II.4.3 Del entrenamiento de las redes recurrentes

Criterios de error

La diferencia fundamental en la adaptación de pesos en redes de comportamiento dinámico y las estáticas es que en las primeras, los gradientes locales dependen de un índice temporal. Es más, los tipos de problema de optimización son también diferentes, ya que en general se esta interesado en cuantificar la performance de la red (o sea del entrenamiento) dentro de un intervalo de tiempo en vez de instantáneamente. El criterio de error más común para redes dinámicas es el aprendizaje de trayectoria (“***trajectory learning”***) en el cual el error se suma sobre todo el tiempo desde un instante inicial $n = 0$ hasta el instante final $n = T$:

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

donde E_n es el error instantáneo y N es el número de neuronas de salida (omitimos la suma que va sobre los patrones por simplicidad de notación). El tiempo T es el largo de la trayectoria y se relaciona con el largo del patrón temporal de interés. En general, la trayectoria tiene el largo del patrón temporal a aprender, aunque puede ser mayor. La función de error es entonces calculada sobre un intervalo de tiempo y lo que se busca es adaptar los pesos para minimizar E_n sobre el intervalo de tiempo. En especial, si el sistema dinámico representado por la red alcanza el estado de estacionariedad (en el sentido de que se puede asociar a un patrón de entrada una salida, en forma estática), se puede usar como función de error

$$E = \sum_{m=0}^{N} \varepsilon_m^2 \qquad \textbf{(II.8)}$$

Este criterio de error es el usado cuando se entrena con ***aprendizaje de punto fijo*** [11].

Paradigmas de aprendizaje para redes dinámicas

Aprendizaje de punto fijo

En esta forma de entrenamiento (utilizado solo con redes recurrentes) se busca asociar a una entrada estática una salida deseada también estática. La función de error asociada es la II.8. Para que pueda ser aplicado el sistema (red) debe ser capaz de alcanzar un estado estacionario. El valor que la red da como salida estable (es decir, que no cambia) para cierto patrón de entrada, luego de haber alcanzado el estado estacionario, es, por definición, un ***punto fijo*** ([11]). En ese momento la red se comporta en forma estática. Entonces, luego de alcanzado ese punto fijo, se puede comparar la salida obtenida para un patrón de entrada, y retropropagar el error.

Se puede generalizar el algoritmo de “back-propagation” para realizar este tipo de aprendizaje de la siguiente manera:

1. *Presentar un patrón de entrada y mantenerlo fijo en la entrada de la red hasta que la salida se estabilice (es decir, que no cambie más)*
2. *Comparar la salida con el valor deseado de salida, calcular el error y retropropagar el error a través de la red dual, de la misma forma en que se hace en el caso estático. Para ello: aplicar el error en la red dual, y mantenerlo constante en la entrada hasta que el error propagado se estabilice.*
3. *Utilizar entonces el procedimiento que se desee para actualizar los pesos de forma de buscar el mínimo de E. Por ejemplo, si se eligiera descenso en el gradiente “puro” sería*

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

4. *Repetir los pasos 1 a 4 para el siguiente patrón.*

Se debe tener presente que:

- Este entrenamiento converge en tanto la red original sea estable, es decir, las salidas de la red se estabilicen. Si la red original es estable, la dual también lo será [11]. Sin embargo, los tiempos que les lleva alcanzar la estabilidad (tiempo de estabilización, Ver II.1.2) a las dos redes pueden ser distintos y cambiar de iteración a iteración, y a veces las redes original y dual tardan más en estabilizarse a medida que la el algoritmo se aproxima a la solución [11].

- Las tasas de aprendizaje deben ser elegidas de manera que permitan un entrenamiento lento. Esto es para que la dinámica del aprendizaje sea mucho más lenta que la de la red. Si ello no sucede, se estaría entrenando una familia de redes más que una red y no hay seguridad de que el proceso converja.

- Como se ve, la retro-propagación estática es un caso especial del entrenamiento de punto fijo, con tiempos de estabilización nulos.

- Un valor tentativo inicial del tiempo de estabilización, para el caso de punto fijo, podría ser 100, es decir que se mantenga el mismo patrón de entrada en la entrada de la red 100 veces más de tiempo que si fuese estática [11].

Aprendizaje de trayectorias

Definimos una ***trayectoria*** como una secuencia de patrones junto con sus respectivas salidas deseadas a lo largo del tiempo. En este caso, buscamos entrenar la red de forma que sus salidas sigan una secuencia específica en el tiempo. O sea, no solo nos interesa el valor al final del período de tiempo, sino que nos interesan además las salidas intermedias de la red (la trayectoria). El aprendizaje de trayectorias puede ser implementado usando RTRL o BPTT en forma equivalente [11]. Una variación al aprendizaje de trayectorias es guardar solo las entradas, salidas y pesos de las neuronas para la parte final de la trayectoria (la parte más reciente), obteniéndose la BPTT ***truncada (o BPPT(h))***. Por más detalles sobre BPPT truncada, Ver ANEXO III o [5, Cap.15].

Dificultades de entrenar redes recurrentes

Existen varias circunstancias que pueden hacer que una determinada tarea a llevar a cabo no pueda ser resuelta mediante una red recurrente. En primer lugar, es posible que el modelo neuronal elegido no sea el adecuado para esa tarea, lo cual puede ser difícil de evaluar en muchas ocasiones. En segundo lugar, suponiendo que el modelo elegido es el adecuado (incluyendo tanto el número de neuronas como la representación de la información de entrada y salida), es posible que el algoritmo de entrenamiento empleado no sea capaz de encontrar un valor de los pesos correcto. Los motivos de ello son principalmente dos: la existencia de mínimos locales y las dependencias a largo plazo. Existe además la posibilidad de que la red se vuelva inestable. "*Entrenar redes recurrentes con BPTT (o con RTRL) es todavía más un arte que una ciencia, siendo las redes TLFNs más fáciles de entrenar, y deberían ser el punto de partida de cualquier solución*" [11].

Mínimos locales

La función de error E define una superficie multidimensional llamada ***hipersuperficie de error*** . Normalmente, la hipersuperficie de error tiene un mínimo "global" (posiblemente varios mínimos "globales" donde E toma valores iguales, debido a simetrías de la red) y muchos mínimos locales, que pueden no corresponder a una solución correcta del problema. Estos mínimos locales son consecuencia de la elevada dimensionalidad del espacio de búsqueda y son el mayor problema, ya que casi todos los algoritmos de aprendizaje tienden a quedar atrapados en ellos, especialmente los que hacen una búsqueda local como ser los basados en el gradiente. En cualquier caso, el problema de los mínimos locales no es específico de las redes recurrentes y afecta a casi todos los modelos de redes neuronales.

Adicionalmente, la superficie de error de las redes dinámicas tienden a tener valles muy estrechos, por lo que la tasa de aprendizaje debe ser controlada con cuidado al entrenarlas. Los algoritmos que usan tasas de aprendizajes adaptativas son los más indicados aquí. En especial, el simulador de redes neuronales elegido permite el ajuste manual y/o automático de las tasas.

Estabilidad

Durante el entrenamiento las redes recurrentes pueden volverse inestables. La no linealidad en las neuronas evitará que la red "explote", es decir que sus salidas sean no acotadas, aunque las salidas de las neuronas pueden oscilar entre valores extremos. Normalmente, controlar que esto no suceda, y reiniciar el entrenamiento en el caso que así ocurra, es la única vía de salida [11].

El flujo de error y el gradiente evanescente ("vanishing gradient")

Consideremos el flujo del error entre la i-ésima neurona y la j-ésima neurona. En ese caso el error cometido en el instante t en la neurona de i-ésima, representado por $\varepsilon_i(t)$, "viaja" hacia atrás en el tiempo (suponiendo que estamos utilizando BPTT) hasta llegar a la neurona j-ésima del instante $s \leq t$. Esta señal de error intenta "modificar el pasado" de manera que se obtenga un presente más conforme con la salida deseada $\mathbf{d}(t)$. Se puede demostrar ([35] y [58]) que el flujo del error hacia atrás (es decir, el error que le llega a la neurona j-ésima) o bien explota (aumenta sin límite) y a la neurona j-ésima le llega un valor de error que puede hacer oscilar a los pesos y volver inestable el entrenamiento o bien el error decrece exponencialmente con la distancia entre t y s, es decir, el flujo de error se desvanece y no es posible realizar el aprendizaje. Esto vuelve el entrenamiento difícil y lento.

Este problema de la degradación de la información del error (o lo que es equivalente, del gradiente local) a través de las no linealidades ha sido llamado el ***problema del gradiente evanescente*** y esta asociado con el ***problema de la dependencias a largo plazo***. Desde el punto de vista dinámico, si se esta entrenando con aprendizaje basado en el gradiente a una red recurrente que como sistema dinámico tiene un atractor hiperbólico, o bien la red no es capaz de aprender cuando los valores de entrada tienen ruido o bien no es capaz de descubrir las dependencias a largo plazo entre los datos [5, Cap. 15]. Para tratar de paliar estos inconvenientes se utiliza el entrenamiento con el filtro de Kalman extendido, métodos de descenso en el gradiente de segundo orden e incluso se puede cambiar de topología: utilizando las redes LSTM. No se conoce solución matemática a este problema, aunque se han propuesto topologías nuevas de red que evitan esto (redes LSTM, ANEXO IV), y el uso de otras formas de entrenamiento (por ejemplo, usando el deKf, Ver ANEXO IV). El problema de la dependencia a largo plazo significa, en otras palabras, que cuando entrenamos una red recurrente, los gradientes pueden llegar a ser muy atenuados, por lo que las relaciones a largo plazo entre los datos van a aprenderse con mucha dificultad, si es que se llegan a aprender. Sin embargo, si se usan estructuras con memoria lineal (redes LSTM), se tiene la ventaja de que los gradientes no son atenuados cuando se los retropropaga a través de ellas. Una mezcla de neuronas con memoria y neuronas no lineales , tal como se hace en las TLFNs, puede llegar a dar mejores resultados que las topologías totalmente recurrentes [11].

El problema de las dependencias a largo plazo

La casi totalidad de los algoritmos de entrenamiento de redes recurrentes encuentran grandes problemas (a veces insalvables) para mantener la información sobre una secuencia, especialmente cuando el intervalo de tiempo entre la presencia de una determinada entrada y la salida afectada correspondiente es relativamente largo (normalmente a partir de unos 10 instantes de tiempo), debido al problema del gradiente evanescente. Más formalmente:

Dada una fuente de datos que genera una secuencia de la forma $s(1),\ldots s(t_u),\ldots,s(t_v),\ldots$ *diremos que existe una* ***dependencia a largo plazo*** *entre el valor del instante* t_v *y el del instante* t_u *y lo anotaremos como* $s(t_v) \multimap s(t_u)$ *si se cumplen las siguientes condiciones:*

1) *el valor de* $s(t_v)$ *depende del valor de* $s(t_u)$

2) $t_v \gg t_u$

3) *no existe* t_w *con* $t_u < t_w < t_v$ *tal que* $s(t_v) \multimap s(t_w) \multimap s(t_u)$

Como ya se vio, los algoritmos de entrenamiento de redes recurrentes basados en el descenso en el gradiente suelen ser incapaces de aprender las dependencias a largo plazo debido a que la salida actual de la red es muy poco sensible a una entrada antigua [35].

Retro-propagación ("back-propagation") a través del tiempo (BPTT)

Existen varias modificaciones del algoritmo de "back-propagation" asociadas a redes dinámicas: "back-propagation" a través del tiempo (BPTT), "back-propagation" recurrente, "back-propagation" dependiente del tiempo con estacionariedad (el método de Pearlmutter) y "back-propagation" en tiempo real (RTRL). Cada una se asocia a un comportamiento especial del sistema que genera las entradas en el tiempo y a cómo se entrena la red. Veremos a continuación la BPTT.

La "back-propagation" a través del tiempo, también llamada "back-propagation through time" (BPTT), así como el RTRL, se basan en el ***desdoblado*** ("unfolding") de la red en el tiempo, y si bien teóricamente es muy general, esta limitada en la práctica por la cantidad de neuronas que se generan al desdoblar la red (por ejemplo, si se quiere aprender una trayectoria de largo = 3, se triplica la cantidad de neuronas).

Para ser más específicos en cuanto a cómo se desdobla la red: sea $\mathcal{N}$ una red recurrente que debe ser entrenada a partir de un tiempo n_0 hasta el tiempo n. Sea $\mathcal{N}^*$ la red "feedforward" que resulta de desdoblar la red $\mathcal{N}$ en el tiempo. La red $\mathcal{N}^*$ se construye a partir de $\mathcal{N}$ de la siguiente forma (por lo tanto, siempre existe):

1- *Para cada paso de tiempo en el intervalo* $(n_0, n]$ *la red* $\mathcal{N}^*$ *tiene una capa conteniendo k neuronas, donde k es el número de neuronas de* $\mathcal{N}$

2- *En cada capa de la red* $\mathcal{N}^*$ *hay una copia de cada neurona de la red* $\mathcal{N}$

3- *Para cada instante (paso) de tiempo* $l \in [n_0, n]$*, la conexión de la neurona i en la capa l a la neurona j en la capa l+1 de la red* $\mathcal{N}^*$ *es una copia de la conexión de la neurona i a la j en la red* $\mathcal{N}$.

Dada las grandes cantidades de recursos computacionales que se requieren para manejar redes correspondientes a aplicaciones reales con "back-propagation through time", se suele recurrir a algoritmos que no desdoblan la red, como algunas implementaciones del BPTT(h) ("back-propagation" truncada, Ver ANEXO III y [5]).

Retro-propagación común (BP) vs. BPTT

Si la red es estática o dinámica, pero "feedforward" (tal como las TDNNs, Ver AIII.3) y la salida deseada de la red es conocida por cada instante de tiempo, se puede demostrar que usar BPTT es equivalente a utilizar BP común y sumar los gradientes locales multiplicados por las entradas a lo largo del intervalo especificado ([11] y ANEXO III). Por otra parte, hay casos en los que la salida se conoce solo en el instante final del intervalo de tiempo. En esos casos tenemos que usar BPTT ya que no tenemos un error explícito en cada instante. Si la red puede entrenarse en algunas partes por BPTT y en otras por "back-propagation" común (como en el caso de las TDNNs), es recomendable por simplicidad aplicar BPTT a toda la red [11].

BP recurrente

Supongamos que la red a estudiar, en la cual el tiempo varía en forma continua, es recurrente y presenta un punto de equilibrio para sus entradas (estacionariedad) en el sentido de que luego de entrenada la red, a igual patrón de entrada le corresponde igual patrón de salida, sin importar el instante. En [34] se discute algunas condiciones que se deben

cumplir para que se alcance este estado estacionario. En ese caso, se puede plantear el entrenamiento de la red usando una variación del algoritmo de "back-propagation" llamado "back-propagation recurrente" (Ver ANEXO III).

BP recurrente dependiente del tiempo – El método de Pearlmutter

Sea ahora una red que no alcanza un punto de estabilidad para un conjunto de entradas fijas, es decir que la salida de la red es siempre una función del tiempo y que el tiempo es continuo. Adicionalmente, supongamos que las salidas de las neuronas de la red cumplen

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

con τ_i constantes independientes del tiempo y $x_i(t)$, $y_i(t)$ funciones continuas representando la entrada y la salida a/de la neurona i-ésima y con N = número de neuronas de la red. En ese caso, suponiendo que trabajamos sobre un intervalo cerrado de tiempo, podemos ensayar usar un algoritmo de aprendizaje "off line", llamado algoritmo de Pearlmutter. El algoritmo, básicamente, utiliza como función de error una generalización de la suma de cuadrados:

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

El algoritmo realiza una búsqueda por descenso en los pesos y, simultáneamente, otra según los τ_i (ver [4] y [50]).

El algoritmo RTRL ("Real Time Recurrent Learning")

En el caso de tiempo discreto, se puede llegar a realizar un entrenamiento "on line" (en tiempo real, es decir, mientras la red va procesando los valores que se le presentan en la entrada, de ahí el nombre del método), que no implica mayores dificultades computacionales. Este método ha sido ampliamente usado en la práctica (Ver ANEXO III y [5]).

Este entrenamiento tiene sentido cuando los pesos cambian lentamente en comparación con los cambios de las salidas respecto las entradas. Podemos decir que el algoritmo es local en el tiempo pero no en el espacio (topología), (por lo que en cierta forma es el dual del BPTT), ya que si consideramos las ecuaciones de actualización de pesos a las que se llega (Ver ANEXO III):

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

donde *n* varía sobre el tiempo, y *p, i* y *j* sobre el número de neuronas, vemos que en cada instante, el gradiente respecto de un peso depende de las derivadas respecto los demás pesos y de los errores en las demás neuronas, de allí lo de ser local en el tiempo pero no en la topología.

Una comparación entre BPTT y RTRL

La siguiente tabla resume las características de los dos métodos [11, Cap. 11]:

	RTRL	BPTT
Almacenamiento requerido	$O(N^3)$	$O(NT)$
Nro. de operaciones requeridas	$O(N^4T)$	$O(N^2T)$
Local en espacio (topología)	NO	SI

Local en tiempo	SI	NO

siendo N = nro. de neuronas de la red y T = largo de la trayectoria

II.4.4 Las redes de Jordan y Elman

Si bien las redes TLFNs (ver II.4.6) son capaces de implementar cualquier correspondencia ("mapping" o "***mapeo***") E→S, hay casos en que dicho "mapeo" esta fuero del alcance de las posibilidades de una red TLFN "focused" de tamaño razonable. Jordan y Elman propusieron redes simples basadas en neuronas de contexto y recurrencias que son fáciles de entrenar y son capaces de aprender dichos "mapeos" utilizando topologías de tamaño reducido [11]. He aquí sus esquemas:

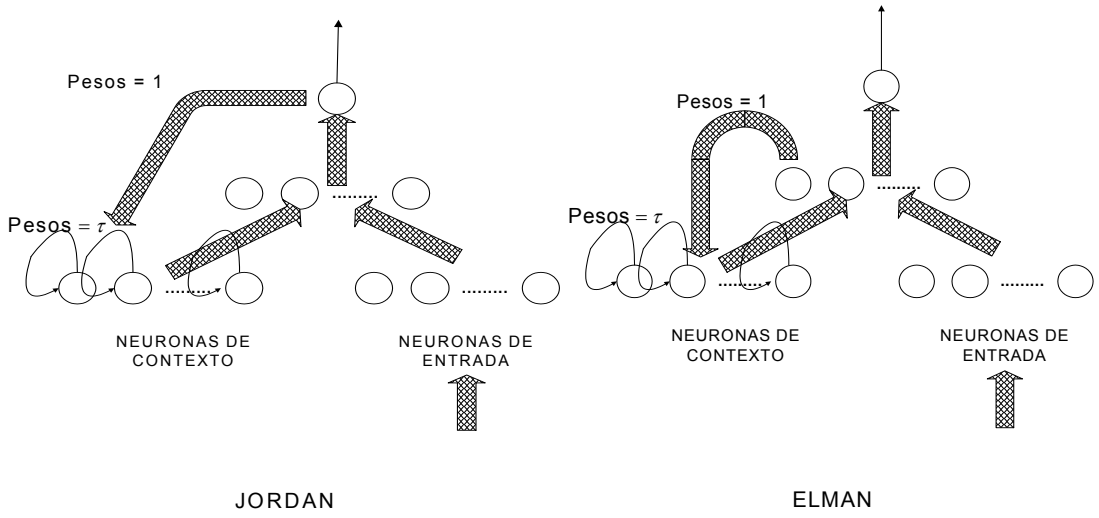


Figura II.6

En los esquemas de la figura II.6, las flechas gruesas representan todas las conexiones posibles.

Ambas redes tienen el parámetro de feedback, τ, fijo y no existe recurrencia entre la entrada y la salida. Pueden ser entrenadas en una forma aproximada usando BP común, (y es así como lo hace el simulador de redes neuronales elegido, [11]). En principio, estas redes son más eficientes que las arquitecturas "focused" (Ver II.4.6) para codificar información temporal.

Ambas redes han sido utilizadas para reconocimiento de secuencias y por ello a veces se las denomina ***secuenciales***.

La memoria

Las redes de este tipo utilizan un tipo de estructuras de memoria llamadas memorias por "feed-back". Las neuronas que poseen esa memoria se las llama ***neuronas de contexto***. La memoria de las neuronas de contexto se obtiene al utilizar un lazo de "feed-back", lo que equivale a calcular la salida en base a sumar los valores de entrada pasados multiplicados por el escalar τ (constante tiempo):

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

Podemos expresar esto gráficamente como:

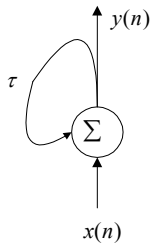


Figura II.7

Obsérvese que un impulso $x(n)$ (es decir, una entrada tal que $x(0)=1, x(n)=0$ si $n \neq 0$) que aparece cuando n=0 y desaparece cuando $n = 1, 2$....va a generar una serie de salidas $y(1)=\tau, y(2)=\tau^2, \cdots$ Es por esto que las neuronas de contexto son llamadas de memoria, porque "recuerdan" los eventos pasados. Este τ (elegido al diseñar la red) debería ser $0<\tau<1$, de otra manera, la respuesta de la neurona se vuelve progresivamente inestable.

Su relación con los modelos AR y MA

En los sistemas lineales el uso del pasado de los valores de entrada crea los llamados modelos de promedio móvil (MA) mientras que el uso de los valores pasados de la salida crea los modelos autorregresivos (AR). En el caso de sistemas no lineales, tales como las redes neuronales ordinarias, esas dos topologías se corresponden con NMA y NAR (N de no lineal) respectivamente. La red de Jordan es un caso particular de modelo NAR mientras que si alimentamos las neuronas de contexto con la capa de entrada obtenemos un caso restringido de NMA. La red de Elman no tiene contraparte en la teoría de sistemas lineales [12].

II.4.5 Potencia computacional de las redes recurrentes

En un sentido general, una de las capacidades computacionales más notables de las redes recurrentes genéricas esta dada por el teorema de Siegelmann y Sontag (1991, citados por [5]): ***Cualquier máquina de Turing puede ser simulada por una red recurrente de neuronas con función de transferencia sigmoide que estén totalmente conectadas unas con otras.*** Finalmente, otra conclusión interesante del estudio de estas redes es el ***teorema de Funahashi y Nakamura.*** Funahashi y Nakamura probaron que la salida de una red neuronal recurrente con suficientes neuronas ocultas, en la que el tiempo varía en forma continua, puede aproximar cualquier trayectoria en el espacio de estados (es decir el comportamiento de cualquier sistema dinámico) con cualquier grado de exactitud. ***O sea que las redes recurrentes son aproximaciones universales a los sistemas dinámicos (de dinámica deterministica)*** [4, Cap. 5].

II.4.6 Las "Time Lagged Feedforward Networks" (TLFNs)

Estas redes son un arreglo (combinación) "feedforward" de elementos de memoria y neuronas no lineales (es decir, con función de salida no lineal). Los elementos de memoria pueden ser de cualquier tipo y estar en cualquier capa. La ventaja de las TLFNs es que comparten algunas de las propiedades de las "feedforward" (por ejemplo, la estabilidad) y además capturan la información presente en los datos de entrada temporales. En especial, se dice que una TLFN es ***"focused"*** si los elementos de memoria están restringidos a la capa de entrada ([11] y [5]). Cuando la memoria se encuentra distribuida en toda la red, se dice que es una red ***TLFN distribuida*** [5] o "***no focused***"[11].

II.5 Algunas consideraciones de diseño

Haremos a continuación una serie de consideraciones sobre la topología de la red, funciones de transferencia a usar, etc. con la idea de mejorar la solución obtenida (una red más pequeña, con más capacidad de generalización, más fácil de entrenar, etc.). Algunas de estas consideraciones fueron tenidas en cuenta en el desarrollo de los experimentos.

Primeramente, Haykin ([5 Cap. 4]) propone varia heurísticas para mejorar la performance del algoritmo:

Heurística 1: Cada parámetro de la función de error que puede ajustarse a través del aprendizaje, debe tener su propia tasa de aprendizaje. Por ejemplo, cada peso debe tener su propia tasa de aprendizaje.

Heurística 2: Cada tasa de aprendizaje debe poder ser variada de una iteración a otra.

Heurística 3: Cuando la derivada de la función de error respecto a un peso tiene el mismo signo algebraico para varias iteraciones consecutivas del algoritmo, se debe incrementar la tasa de aprendizaje.

Heurística 4: Cuando la derivada de la función de error respecto a un peso en particular alterna para varias iteraciones consecutivas, la tasa de aprendizaje para ese peso debe ser reducida.

Hay que tener presente que el uso de tasas de aprendizaje variables modifica el algoritmo de "back-propagation" haciendo que ya no se haga una búsqueda según el gradiente, sino que los ajustes en los pesos se basan en a) las derivadas parciales de la superficie de error con respecto a los pesos y b) estimaciones de la curvatura de la superficie de error en el punto actual (operativo) respecto a algunos de los pesos. Sin embargo, estas heurísticas y las modificaciones asociadas al algoritmo han probado tener utilidad [5].

A continuación pasaremos a hacer algunas consideraciones generales sobre otros aspectos que también pueden afectar el comportamiento del algoritmo.

II.5.1 De la función de salida

Desde el punto de vista del cálculo, éste se facilita cuando se usan las funciones tales como la logística o la tangente hiperbólica ya que en esos casos *f'* se puede expresar como función de *f*:

para la logística:

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

y para la tangente hiperbólica:

$$f(Net) = Tanh(\beta Net)$$
$$f'(Net) = \beta[1 - f^2(Net)]$$

siendo Net la entrada a la neurona respectiva:

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

II.5.2 De la convergencia

Normalmente se usan los pesos actuales que llegan a la capa de salida para el cálculo de la variación de los pesos que llegan a la capa oculta, aunque generalmente se obtiene una mayor corrección si se emplean los valores de los pesos

actualizados de la capa de salida $w_{ij}^{nuevo} = w_{ij}^{actual} + \Delta w_{ij}$. Desde el punto de vista computacional, hay que tener en cuenta que esto significa un costo adicional de recalcular y_i y $f_o'(Net_i)$.

Si no se puede encontrar una solución a partir de los parámetros actuales (no hay convergencia), se puede intentar nuevamente cambiando algunos de ellos, tomando un nuevo conjunto inicial de pesos y/o usar más neuronas ocultas ya que su número interviene en la eficacia del aprendizaje Es más, si se observa un aprendizaje lo suficientemente rápido, se pueden disminuir las neuronas ocultas, a fin de optimizar recursos computacionales usados. Se puede llegar entonces a un equilibrio entre performance del entrenamiento y cantidad de neuronas ocultas. Rumelhart (citado por [4]) desarrolló un método para "podar " las neuronas ocultas que participan muy poco en el proceso de aprendizaje (o sea, aquellas cuyos pesos asociados cambian muy poco durante el mismo, Ver ANEXO VI).

Hay que tener presente que se debe avanzar sobre la superficie de error con incrementos pequeños de los pesos, ya que solo tenemos una información local de la superficie y no se sabe lo cerca o lejos que esta el punto mínimo. Usar incrementos grandes puede llevar a que pasemos "por encima" del punto mínimo, oscilando alrededor de él sin alcanzarlo nunca. A su vez, un incremento demasiado pequeño puede llevar a una convergencia muy lenta. Normalmente se suele tomar ρ entre 0,05 y 0,5. También es una buena medida aumentar ρ a medida que disminuye el error de la red en la fase de aprendizaje, aunque teniendo en cuenta que no debe ser demasiado grande [6].

Si bien no se representó la entrada ficticia a cada neurona cuyo valor es 1 (constante) y peso el umbral de activación θ de esa neurona, también interviene en el proceso de aprendizaje y se ajustará debidamente su peso [6].

II.5.3 Del conjunto de entrenamiento

No hay una regla para predecir cuantos tienen que ser los pares (**x, d**) de entrenamiento para conseguir un resultado aceptable en el caso general de una red de topología arbitraria y funciones de transferencia sigmoides. Por ello, siempre es bueno disponer de tantos pares como sea posible. Hay que recordar que un subconjunto de ellos podría ser usado para verificación (en el caso de utilizar cross-validation, Ver ANEXO I), lo que hace que aumente la cantidad necesaria de pares con respecto al caso en que todos se usan para entrenamiento. Baum y Haussler [18] hallaron que para redes multicapa "feedforward" con neuronas con funciones de salida el escalón unitario (o de Heaviside) y valores deseados {-1, +1}, se cumple que $d_{VC} \leq 2W\log_2(eM)$ siendo d_{VC} la dimensión de Vapnis-Chernovenkis([1], [5 Cap. 2]), W la cantidad de pesos de la red, M el número de neuronas y $\boldsymbol{e}$ la base de los logaritmos naturales. De allí dedujeron que si se entrena con N patrones siendo $N \geq \frac{W}{\varepsilon}\log_2(\frac{M}{\varepsilon})$ y de forma que se clasifique bien un $100(1-\varepsilon/2)$ por ciento de ellos, entonces hay una gran posibilidad de que la red clasifique bien un $100(1-\varepsilon)$ por ciento de las muestras futuras generadas con la misma distribución de probabilidad [1]. Conjeturaron además que para las funciones de transferencia comúnmente usadas se cumplen cotas similares y algunos [13] usan esas cotas sin otras consideraciones teóricas. O sea que si queremos tener un nivel de error de clasificación de 0.1, debemos considerar alrededor de 10*W pares de entrenamiento [13].

La adición de ruido a la entrada permite acelerar la convergencia, aunque la red no vaya a trabajar con entradas con ruido [4].

Adicionalmente, el número de patrones de entrada necesarios está condicionado por ciertas características de los datos que se modelan [13] a saber:

1) La ***dimensionalidad intrínseca*** de los datos, que es el número de variables de entrada independientes (es decir, cuyo valor no depende de ninguna otra de entrada). Una heurística es aumentar el tamaño del conjunto de entrenamiento en un factor de 10 por cada variable independiente. En el caso de una serie temporal, la dimensión intrínseca esta dada por la dimensión del embedding [13].

2) La ***resolución*** o granularidad de los datos, que se refiere a al número de divisiones hechas a lo largo de cada dimensión (variable de entrada). Por ejemplo, si queremos que una medida sea dada en centímetros o en centímetros y milímetros. Cuanto mayor sea la escala de división (menos granularidad), menos datos precisaremos ya que vamos a precisar menos valores muestrales de esa variable que caigan en cada uno de los rangos de división y además estaremos menos proclives a modelar ruidos de los datos [26].

3) La ***distribución de probabilidad*** de los datos: debemos asegurarnos que haya suficientes datos como para cubrir cada estado posible que pueda alcanzar el sistema a modelar. Por lo tanto, si sabemos que entrada a la red (correspondiente al estado del sistema) **x** se da con probabilidad P(**x**), deberemos tener al menos 1/P(**x**) patrones de entrenamiento para estar relativamente seguros de que la entrada incluye algún ejemplo de **x**.

4) El ***ruido*** y la ***calidad*** de los datos: si los datos contienen ruido demasiado pronunciado o son de mala calidad (por ejemplo, por ser escasos y no representar todas las entradas posibles), puede llegarse a requerir mucho más volumen de datos, ya que una buena parte de ellos será descartada.

II.5.4 Del dimensionamiento de la red

En general, tres capas (entrada, oculta, salida) son suficientes De hecho, se puede demostrar que no son necesarios más de una capa oculta, con la cantidad adecuada de neuronas, para resolver cualquier problema de clasificación o interpolación (Ver ANEXO VIII).

Los tamaños de la capas de entrada y de salida están dados por la naturaleza de la aplicación.

El tamaño de la(s) capa(s) oculta(s) deberá ser estudiado cuidadosamente ya que:

1) El número de neuronas ocultas debe ser tal que permita almacenar la información presente en el conjunto de entrenamiento y a la vez no ser tantas como para que la red pierda la capacidad de generalización .

2) El número de neuronas de la capa oculta debe ser menor que el de la de entrada, si se va a hacer alguna compresión ("feature extraction" o reducción de la dimensionalidad) de los datos de entrada.

3) En [13] se propone que el número de neuronas ocultas nunca sea más de dos veces el de las de entrada. La justificación a esto se encuentra en el teorema de Kolmogorov sobre aproximación de funciones (ver ANEXO VIII).

4) Cada neurona oculta consume recursos computacionales al momento de simular la red, y se debe llegar a una solución de equilibrio performance de aprendizaje-número de neuronas ocultas.

II.5.5 De los pesos y parámetros de aprendizaje

Es conveniente que los pesos se inicialicen con valores pequeños al azar uniformemente distribuidos, por ejemplo en el intervalo (-1 , +1) lo mismo que para θ_i [13]. Por otra parte, en [3] se sugiere que estén en el (-0.5,+0.5). Otra heurística sugerida en la bibliografía (por ejemplo, en [4]) es que los valores iniciales de los pesos se hallen en el intervalo $(-1/\sqrt{f}, +1/\sqrt{f})$ siendo f el número de conexiones que llegan a la neurona i-ésima ("***fan-in***").

Kolen y Pollac [38] descubrieron, que para redes feedforward aprendiendo el exor, que existe una estructura que representa la convergencia en función de los pesos iniciales que es similar a los fractales, donde existen regiones de gran sensibilidad en el espacio de pesos, en los que pequeñas diferencias iniciales en los pesos pueden llevar a curvas de aprendizaje muy diferentes. Eso pone de manifiesto lo sensible que es la retro-propagación a la elección de los pesos iniciales, mostrando un comportamiento caótico. Esto podría ser debido a la existencia de varios mínimos de la función de error, a los parámetros de aprendizaje (tasas y pesos) no nulos y a la naturaleza determinística no lineal del enfoque del descenso según el gradiente

La elección de la tasa de aprendizaje es importante: en general entre 0.05 a 0.25, pudiendo variarse a medida que el proceso avanza [3] (ver II.2.3).

Parálisis de la red: Si se usa una función de transferencia sigmoide (tal como la tangente hiperbólica o la logística), hay que cuidar que los valores absolutos de los pesos no crezcan demasiado ya que en ese caso las neuronas operan con valores grandes de entrada, y en ese caso, como la derivada de la función de salida es muy cercana a 0, no se producen actualizaciones en los pesos, por lo que la red deja de aprender (se paraliza).

Sección III: Caso de estudio: Predicción de las ventas semanales de gas

En esta sección se exponen algunos de los estudios y experimentos realizados sobre la serie de datos formada por las ventas semanales de gas envasado y los modelos creados para predecirla y reconstruirla dinámicamente. Primeramente se detallan los estudios realizados a los datos con el fin de determinar algunas características subyacentes, tales como autocorrelaciones, estacionalidad, etc. Luego se continua con una descripción de los resultados obtenidos con algunos modelos de red indicados en la bibliografía para la predicción de una serie de este tipo (las TLFNs y las recurrentes) y se comparan con los dados por perceptrones (MLPs). Finalmente, se utiliza el método del "ensemble" [55] y se plantean algunas conclusiones.

III.1 Herramientas utilizadas

Se utilizaron distintos productos de software dependiendo de qué aspecto del proyecto se estuviera tratando. La idea fue investigar el estado del arte de las herramientas de software necesarias para llevar adelante un proyecto de este tipo. Algunos de los requerimientos comunes a todos ellos son:

- ser adecuados para trabajar en plataformas Intel (PC) stand alone, utilizando sistema operativo Windows 95/98/2000/XP
- ser productos estables, en lo posible, es decir, no ser versiones beta ni prototipos. De esa forma se pretendía tomar contacto con herramientas que ya tuvieran cierta madurez en el mercado.
- ser actuales: que su última versión disponible fuese lo más reciente posible y que el producto siguiera existiendo en el mercado
- ser productos de distribución industrial (comercial), ya que así se investigaba el estado del arte en herramientas de uso relativamente común en la industria
- estar disponibles para usar (en el sentido de que se pudiera conseguir una copia bajo licencia, aunque más no fuera de una versión anterior), lo cual era un requerimiento excluyente, y tener al menos demos de sus últimas actualizaciones (para saber como son actualmente)
- en el caso de los simuladores de redes se requirieron además las características siguientes:
    - presentar características (facilidades) que hicieran que su uso implicara profundizar el estudio teórico de las RNA
    - presentar una interfase gráfica de ayuda al diseño de la red (opcional)
    - haber sido diseñados por equipos que integraran a investigadores reconocidos en el área
    - permitir la mayor variedad de topologías, reglas de aprendizaje, etc. posibles, especialmente de aquellas que más se adecuaban al caso de estudio.

No consideramos aquí puntos que pueden considerarse básicos cuando se adquiere un software:

- costo de las licencias (individual o de cualquier tipo)
- número de instalaciones previas: solo tuvimos en cuenta la madurez del producto, lo que seguramente refleja la cantidad de instalaciones que se hicieron: un producto cuya versión actual es la 5, es muy probable que se haya instalado en más sitios que otro destinado al mismo propósito y que esta en la versión 2.
- tamaño (importancia) de la empresa que lo desarrolla, lo que puede dar idea del soporte del producto en el futuro (si la empresa va dejar de existir o no)
- portabilidad a otras plataformas
- idioma en que esta escritas la interfase con el usuario: estaban todos en inglés, salvo el PREVia que estaba en francés (al menos la demo)
- cuestiones relacionadas con la seguridad y acceso al producto
- habilidad para recuperarse de fallas inesperadas (cortes de energía, etc.)

Clasificamos el software utilizado en herramientas auxiliares (todas las accesorias tales como procesadores de texto, etc.) y el simulador de redes en sí.

III.1.1 Herramientas auxiliares

Teniendo en cuenta todos los puntos mencionados antes, elegimos:

- para el manejo de datos, proceso de documentos, etc., las herramientas de Microsoft Office 2000. Debido a que el Microsoft Word 2000 generaba esperas inadmisibles al trabajar con este documento en su versión completa y generaba errores al momento de guardar lo realizado, se utilizó Microsoft Word 2003. Para la edición de ecuaciones, se utilizó un producto, el Mathtype ver. 5, similar al editor de ecuaciones de Word, pero más fácil de usar.

- para el manejo de estadísticas de datos y de resultados de simulaciones: consideramos el SPSS ver. 11.0 y el Statistica ver 6.0. Luego de algunas pruebas, se eligió el Statistica vistas las ayudas y funcionalidades extras e interfase que proveía.

- el archivo .pdf que se entrega junto con el presente trabajo fue creado con Adobe Acobat Writer ver 5.0, y el CD respectivo con el Nero Burning.

- dado que se requirió gran cantidad de tiempo de navegación en Internet, se hacía imprescindible utilizar una herramienta que facilitara las búsquedas en forma inteligente. Por ello utilizamos el Copernic Agent Ver 6.01 (versión freeware) que es un metabuscador que permite filtrado y agrupado de la información de Internet.

- El manejo de la serie temporal nos llevó a utilizar una serie de productos: el TISEAN, el VRA, el TSTOOL y el DATAPLORE. Los tres primeros son freeware, y el último provee una versión demo. El TSTOOL esta orientado a línea de comandos y corre bajo Matlab. El VRA y el DATAPLORE proveen muy buenas interfases gráficas, mientras que el TISEAN corre bajo ventana DOS, y no maneja gráficos. Algunas de estas herramientas se hallan descriptas en [20].

III.1.2 Simulador de redes neuronales

Para las simulaciones de las redes en sí evaluamos varios simuladores de la multitud existente en el mercado. El elegido fue el NeuroSolutions ver 4.0. Describimos a continuación algunas de las características que encontramos sobre cada uno de ellos:

ECANSE (Environment for Computer Aided Neural Software Engineering).
Es un producto desarrollado por la compañía Siemens (Austria). Algunas de sus características más relevantes son:

- es un producto certificado ISO9001
- utiliza teoría del caos para hacer análisis no lineal
- maneja algoritmos genéticos y permite utilizar lógica borrosa ("fuzzy logic")
- soporta varias topologías, tales como MLPs, redes de Hopfield, etc.

Si bien es un producto que industrialmente puede ser muy interesante, lo descartamos ya que su versión demo (la única disponible) solo salva 15 objetos y la última versión del producto apareció en marzo/98, para Windows NT. Por otra parte, la interfase con el usuario no es atractiva.

EXPO/NeuralNet
Es un producto desarrollado por Leading Market Technologies Inc. y esta muy orientado al manejo de series temporales, incluyendo una interfase especial para distintas fuentes de información on line (Reuters, Bloomberg, etc.). Es un producto que tiene una versión estudiantil gratis. Hemos probado la versión gratuita y vemos que si bien se dedica de lleno a las series temporales, casi no permite especificar el diseño de la red a utilizar.

Matlab ver 6.1 release 12 y su Neural Networks Toolbox.
La interfase del diseño de la red es pobre. Por otra parte, las únicas funciones de error que maneja son derivadas del MSE, no teniendo la posibilidad de utilizar normas L-r. Además, en cuanto a las topologías soportadas, como recurrente no estocástica solo incluye la red de Elman.

NeuroShell
Es un producto de Ward Systems Group Inc. y fue desarrollado para la predicción. Utiliza dos métodos básicos: redes neuronales y una combinación de algoritmos genéticos y estimación estadística. Puede realizar un diagnóstico de qué importancia tiene cada elemento de entrada (análisis de sensibilidad). Finalmente, es un producto de bajo costo (alrededor de 500 dólares norteamericanos). Como no conseguimos ni siquiera una versión demo del mismo, se descartó.

Adaptive Logic Network
Permite realizar aprendizaje supervisado (y en ese caso la única medida del error a usar es la distancia L2) o hebbiano. La versión beta data de noviembre/2002, por lo cual puede ser poco estable.

Attrasoft Boltzmann Machine (ABM)
Desarrollado por Attrasoft, es un producto ya maduro (su versión actual es la 2.3) que permite simular redes de Hopfield y de Boltzmann. Fue pensado para la clasificación y determinación de patrones[8]. Como se ve su diseño no esta orientado a la predicción y además es muy limitado en la cantidad de topologías distintas que permite. Finalmente, no se pudo manejar ni siquiera un demo del mismo.

PREVia
Es un software francés desarrollado por Elseware S.A. y fue pensado para la predicción de series temporales. Provee de un entorno interactivo de desarrollo de modelos. Si bien la compañía sigue manteniendo el producto, encontramos que la documentación de su versión de evaluación data de 1996, lo que nos hace sospechar que sea un producto que no se haya actualizado con el tiempo. Por otra parte, la interfase gráfica es muy buena, aunque la versión demo no permite utilizar datos propios, lo cual le quita utilidad práctica. Como dato anecdótico, es uno de los productos más caros (3500 euros) contra los aproximadamente 3000 que cuestan el Adaptive Logic Network o el NeuroSolutions.

Statistica Neural Networks ver 6.0
Fue desarrollado por StatSoft Inc (los creadores de Statistica). Según la descripción que de él provee la empresa, permite tareas tales como:

- pre y post proceso de los datos (codificación, escalamiento, normalización, rellenado de valores faltantes)
- PCA y "feature selection" de los valores de entrada, para determinar las entradas a la red más apropiadas
- generación de código C++, C# o Visual Basic
- manejo de las topologías MLP, RBF, redes de Kohonen, lineales, PCA, etc.
- trabajar con comités de redes

Dado que no nos fue posible ver siquiera una demo del producto, se descartó.

NeuroSolutions
Desarrollado por Neuro Dimensions Inc., utiliza una interfase gráfica basada en íconos mediante la cual se puede construir modularmente la red. Adicionalmente, tiene un demo disponible. Fue desarrollado con aportes de José Príncipe y Kurt Lefebvre, quienes son investigadores reconocidos en el área de redes neuronales. Presenta una buena interfase gráfica, genera código C++ (no en todas las versiones), proporciona una amplia variedad de topologías: MLPs,

[8] Dada una parte del patrón y la clase a la que pertenece, determinar el resto del mismo

RBFs, redes de Jordan y Elman, Hopfield, Kohonen, etc. Es un producto que esta en la versión 4, lo cual lo clasifica como estable. Adicionalmente, existe una versión gratis para estudiantes que se distribuye con el libro "Adaptive Neural Systems: Fundamentals through simulations". Por todo ello fue la herramienta de simulación elegida.

III.2 Análisis de los datos disponibles

III.2.1 Estudio estadístico

Dado que el diseño de la red va a ser altamente influenciado por la estructura lógica subyacente en los datos, se les ha realizado algunas pruebas estadísticas para investigar la misma.

Intuitivamente se sabía que

- el gas en garrafas es utilizado primariamente como combustible para la generación de calor, en especial, para calefacción
- las ventas de gas dependen de la temperatura ambiental (cuando hace frío, se vende más, cuando hace calor, se vende mucho menos, ya que no solo que el comprador no lo consume, sino que además trata de minimizar el tiempo que están encendidas las fuentes generadoras de calor)
- las ventas de una semana pueden depender de lo que se haya vendido las semanas anteriores
- en ciertas semanas del mes, se tiende a vender menos (por ejemplo, en la última) y en otras más (por ejemplo, en la segunda, donde se producen los cobros de sueldos)
- los aumentos en el precio de venta de gas influyen en las ventas en las semanas previas y posteriores a ellos

Los datos de los que se disponía consistían en cuaternas (fecha, temperatura mínima diaria, temperatura máxima diaria, volumen de gas vendido (en litros)), siendo las temperaturas las registradas para el día correspondiente por la Dirección Nacional de Meteorología., aunque algunas se obtuvieron consultando la prensa vía Internet durante periodos de varios meses (setiembre/2002 a abril/2003) y otras (aquellas correspondientes a los días donde no se publican diarios) se obtuvieron de la página www.weather.com que tiene un promedio de temperaturas diarias para Montevideo de los últimos 25 años. Existían datos de ventas de gas diarias desde febrero de 1996 a agosto de 2002, aunque debido a lo dispersos que resultaban en algunos años (de 1996 solo había 100 días) se decidió trabajar con los datos disponibles provistos por un datawarehouse corporativo, los cuales existían a nivel de día (ventas diarias) y a niveles semana, mes y año, desde mayo de 1999 a la fecha, que eran considerados confiables y no presentaban discontinuidades en el tiempo. La determinación de cual de los dos conjuntos de datos era preferible emplear (los del datawarehouse o los de las ventas diarias desde febrero/96) implicó realizar algunas pruebas con diferentes redes (pruebas que aquí no se detallan) para ver los resultados obtenibles. Estas pruebas insumieron cierto tiempo y la elección no fue inmediata. Se utilizaron los datos a nivel semana. A partir de las temperaturas diarias, se determinaron las temperaturas promedio semanales (Ver III.2.4).

Se realizaron las pruebas que se describen a continuación a efectos de completar el conjunto de interdependencias existentes.

Pruebas realizadas

Estudio de periodicidades

Las ventas semanales (en litros) de los últimos cuatro años son (el eje de las x representa las semanas):

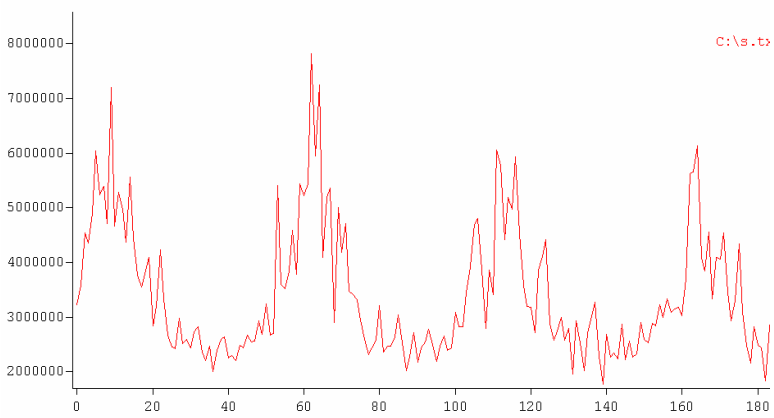


Figura III.1

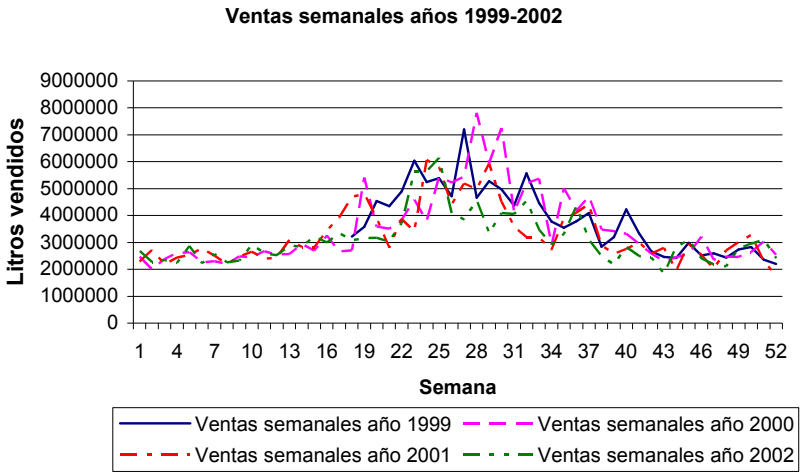


Figura III.2

Se observa la periodicidad de los datos. Pasamos a estudiar los promedios móviles. Los promedios móviles de 52 elementos (centrados en el elemento en estudio y equiponderados) de la serie temporal de ventas son:

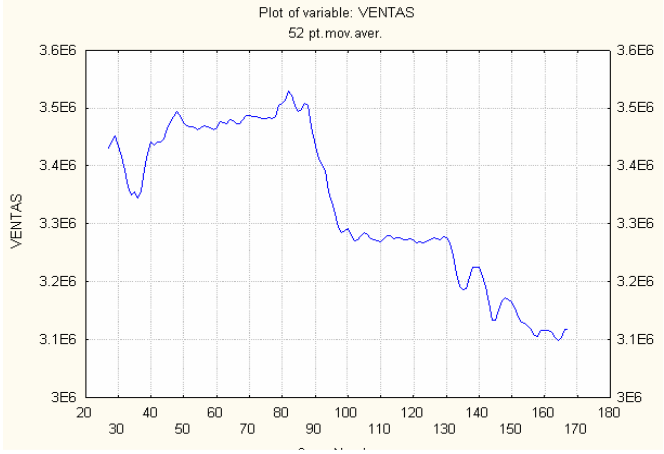


Figura III.3

y los de de 27 elementos:

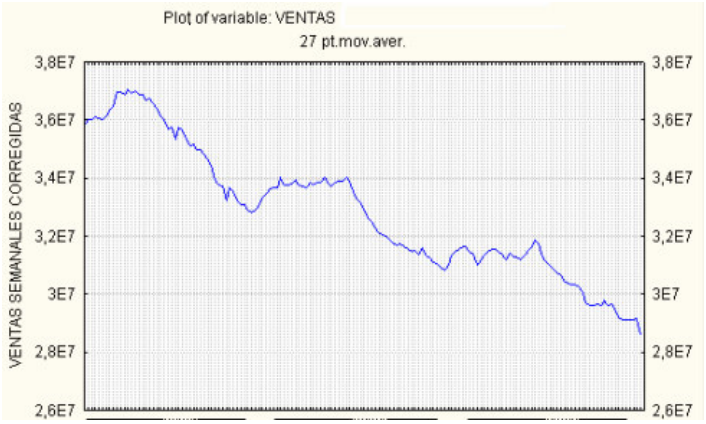


Figura III.4

En ambos casos el eje de las x representa las semanas y el de las y el promedio móvil. A esta variable suavizada la volvemos a suavizar, haciendo los promedios móviles de 27 valores consecutivos [9] y obtuvimos:

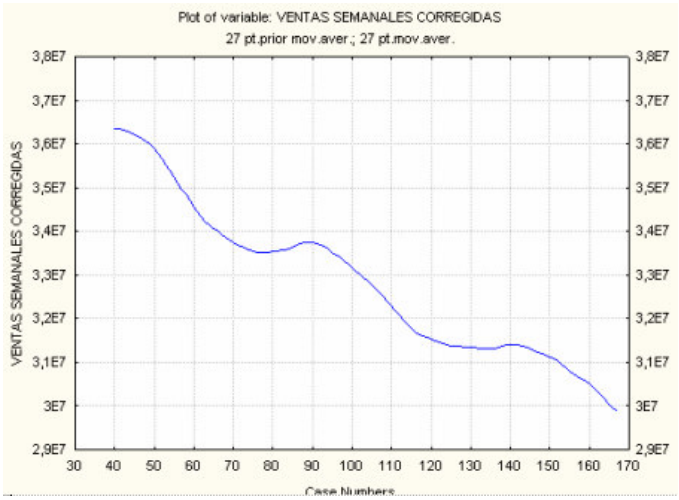


Figura III.5

En la gráfica de la figura III.5 confirmamos que existe una estacionalidad en los datos, y que el período es de alrededor de 52 semanas (es decir, un año), lo cual ya era observable en el primer gráfico de las ventas semanales para los distintos años. Esto también lo vemos si estudiamos las correlaciones lineales entre las ventas mensuales de los distintos años. Por ejemplo:

Años	ρ
2000-2001	0,90
2001-2002	0,88
2000-2002	0,87

Asimismo, parece existir una tendencia a consumir menos gas, si se considera la pendiente de la regresión lineal de las ventas semanales tal como se ve en la figura III.6.

[9] El número 27 fue elegido heurísticamente, y representa aproximadamente la mitad de las semanas que hay en un año.

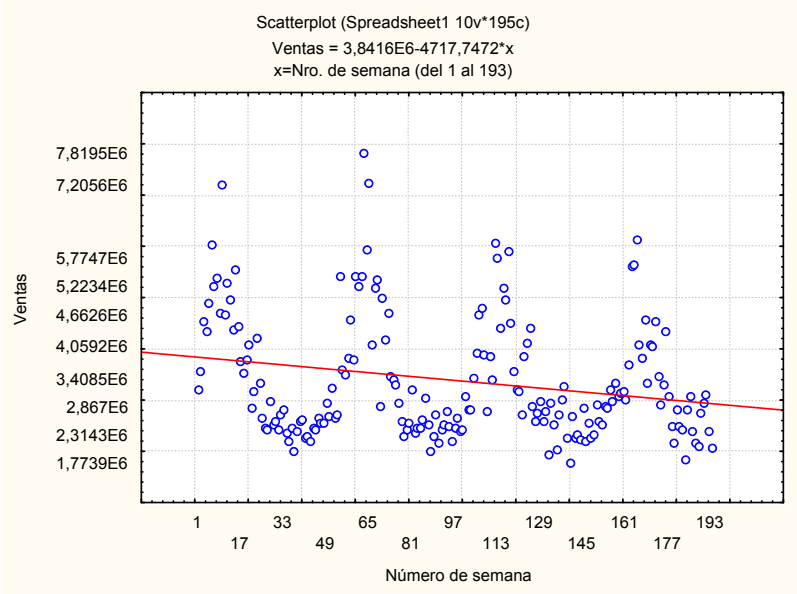


Figura III.6

A los efectos de detectar otras periodicidades, estudiamos la autocorrelación de las ventas semanales durante el período de dos años, y obtuvimos:

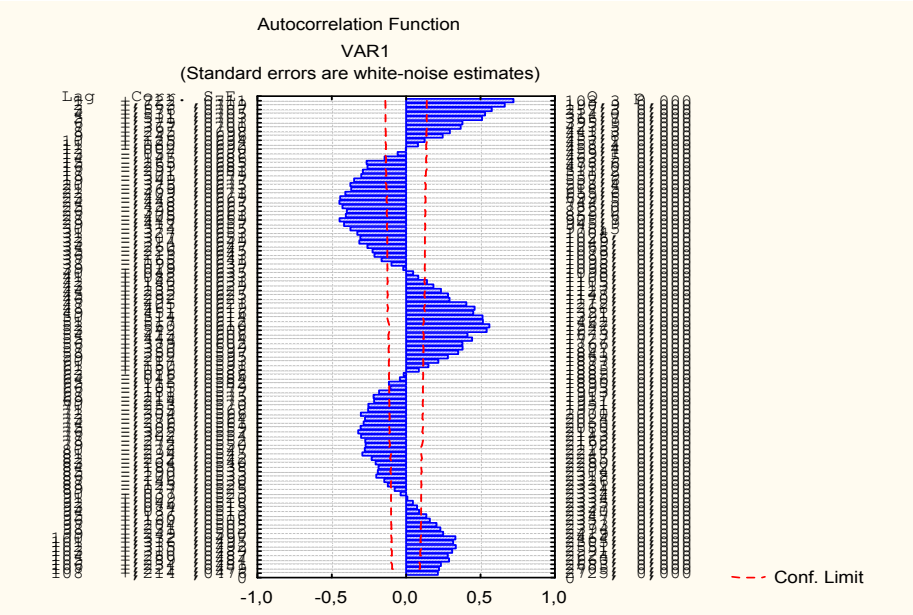


Figura III.7

En el eje vertical se representan los distintos $i = 1, 2...$ ("lags") y en el horizontal la correlación entre la serie y la serie demorada i semanas. Nuevamente, aparece una periodicidad de largo aproximado = 50.

Si estudiamos la autocorrelación de menos semanas hacia atrás las dependencias quedan:

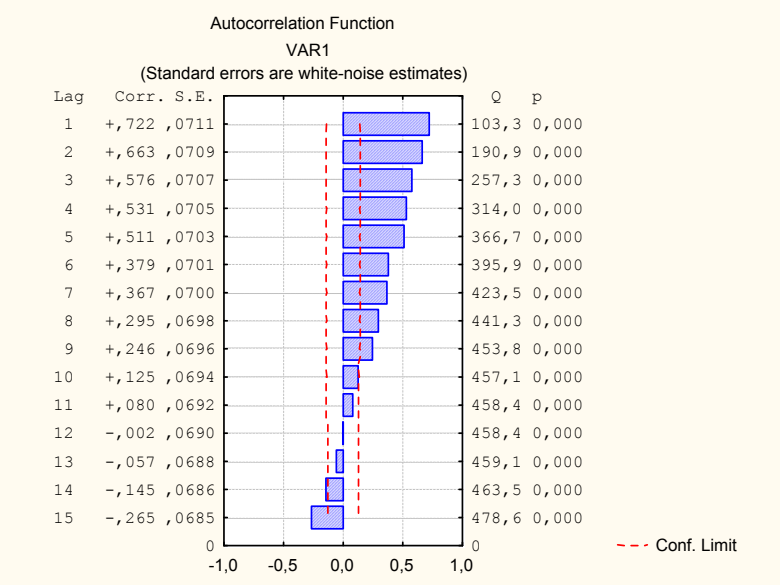


Figura III.8

Podemos observar que más allá de tres a cinco semanas hacia atrás en el tiempo la influencia en las ventas semanales en las actuales es casi nula. Esta hipótesis se verá confirmada en III.2.2 "Determinación del mAMI".

Estudio de correlaciones específicas de las ventas semanales

Dadas las ventas diarias provistas por el datawarehouse, agrupamos de a 7 días consecutivos las ventas, las temperaturas máximas y las temperaturas mínimas. Obtuvimos las siguientes correlaciones entre las sumas de las ventas y las temperaturas máximas y mínimas:

P(V,T)	***ϱ(V,t-mín)***	***ϱ(V,T-1)***	***ϱ(V,t-mín -1)***	***ϱ(V,T-2)***	***ϱ(V,t-mín-2)***
-0.77	-0.78	-0.74	-0.64	-0.62	-0.60

donde

V	ventas acumuladas en el grupo de 7 días
t-mín	temperatura mínima promedio en el grupo de 7 días
T	temperatura máxima promedio en el grupo de 7 días
t-mín-1	temperatura mínima promedio en el grupo de 7 días anterior al actual
T-1	temperatura máxima promedio en el grupo de 7 días anterior al actual
T-2	temperatura máxima promedio en el grupo de 7 días anterior al anterior al actual
t-mín-2	temperatura mínima promedio en el grupo de 7 días anterior al anterior al actual

Dado que la diferencia entre las correlaciones ***ϱ(V,T)*** vs. ***ϱ(V,t-mín)*** es mínima (0.01) y es más significativa en los casos ***ϱ(V,T-1)*** vs. ***ϱ(V,t-mín -1)*** y ***ϱ(V,T-2)*** vs. ***ϱ(V,t- mín-2)*** decidimos usar como temperatura de referencia la máxima promedio semanal.

Eliminación de la tendencia ("detrending")

La bibliografía (por ejemplo, [22]) señala la importancia de realizar el "detrending" (eliminación de la tendencia) de la serie afín de mejorar la performance del MLP; dicha tendencia puede ser la lineal, en el caso general, o la dada por otro tipo de función (por ejemplo un polinomio de grado mayor que dos, como en [22]). Consideraremos en adelante solo la lineal. Dado que la pendiente de la tendencia lineal de nuestros es suave, el desbalance[10] entre los datos de entrenamiento y "testing" va a ser pequeño, especialmente dado que solo se utilizan 17 semanas para "testing", por lo que, en una primera instancia, probamos entrenar las redes sin eliminar dicha tendencia de los datos. Esto no tendría sentido si se estuviese trabajando con los datos de las ventas de un período mucho mayor, por ejemplo, de los últimos 10 años. Otros motivos por los que no realizamos el "detrending" fueron:

- se intentó evaluar las capacidades de aprendizaje de las distintas topologías, tomando la serie original o una sub serie de ella (como al aplicar el teorema de Takens-Mañé), es decir, realizando el menor pre-proceso posible a los datos

- la red es utilizada por un usuario, por lo tanto, si se realizaba un "detrending" utilizando una función más compleja que la lineal, la reconstrucción de la predicción (es decir deshacer los cálculos de eliminación de la tendencia para obtener la predicción final) se le iba a hacer engorrosa

- podemos ver la poca influencia que tiene en la serie el "detrending", al comparar la gráfica original y la "detrended" al comparar las figuras III.9 y III.10:

Serie original:

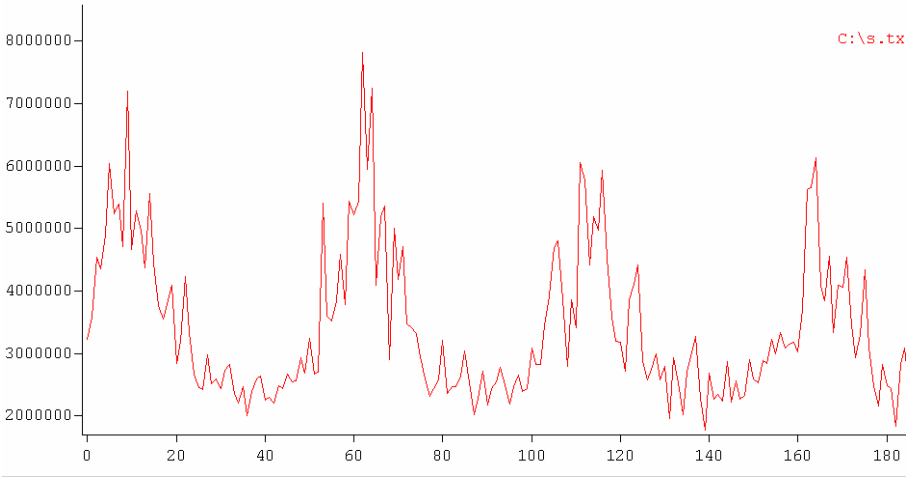


Figura III.9

[10] En el sentido de que los datos de "testing" (ventas de las semanas 2-18/2003) tengan una tendencia lineal muy marcadamente distinta de los de "training" fundamentalmente al transcurso del tiempo y los cambios que se produjeron en el mercado.

Serie sin la tendencia lineal:

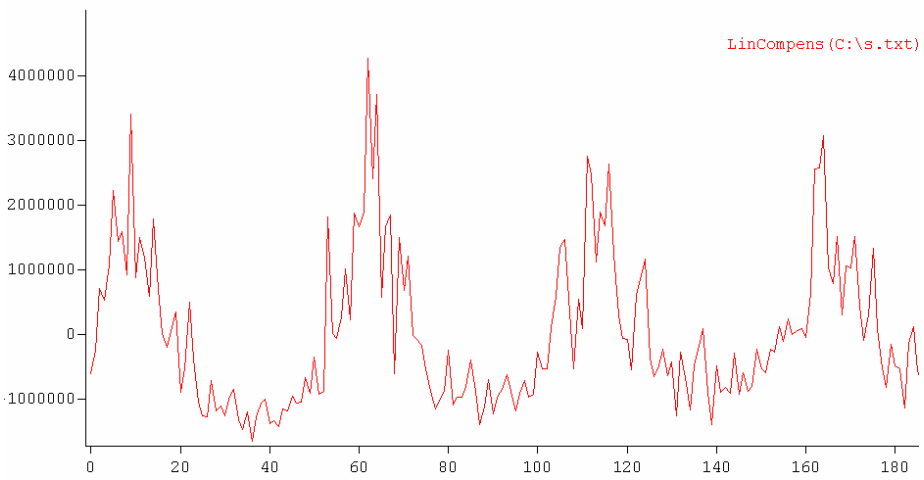


Figura III.10

Como comprobación empírica, entrenamos un MLP con V, V-3 prediciendo V+1, 11 neuronas ocultas y la serie de entrenamiento "detrended" (según la tendencia lineal). Los resultados obtenidos fueron tales que en 20 pruebas de entrenamiento, los %ErrorCV eran de más de dos dígitos. Lo mismo sucedió aun cuando el número de neuronas ocultas se determinara genéticamente.

III.2.2 Discusión del modelo de sistema a adoptar

Algunas de las consideraciones que se hicieron al plantear qué modelo de sistema a adoptar (es decir, cómo supondremos que se comporta la realidad) fueron:

1) Determinístico o estocástico?
La realidad a modelar se rige por una serie de leyes físicas que son determinísticas, pero también intervienen factores que por su complejidad solo pueden ser modelados por un proceso estocástico (por ejemplo, la reacción de los consumidores ante un aumento de precio de venta del gas). Por lo tanto el modelo del sistema tendrá que ser estocástico.

2) El sistema real es visto como totalmente aleatorio o como un sistema dinámico estocástico?
Si consideramos las gráficas de las ventas semanales en III.2.1 vemos que las ventas de los cuatro años parecen corresponder a un proceso que tiene una componente determinística y otra aleatoria. La aleatoriedad está restringida (en el sentido de que no todos los cambios de estados son posibles). Por lo tanto, el sistema aleatorio sería o bien aleatorio (pero con restricciones en cuanto a los cambios de estados posibles) o bien dinámico-estocástico. Ese sistema real da origen a un proceso aleatorio discreto de ventas semanales $\mathcal{P} = \{v(n) \quad n = 1,2...\}$

3) Cómo modelamos el sistema y el proceso $\mathcal{P} = \{v(n) \quad n = 1,2...\}$?
Sabemos, a partir del estudio de las autocorrelaciones de la serie y del AMI (ver III.2.3), que no existe una correlación significativa en las ventas de la semana i y de la j si $j > i+3$ y además sabemos que existe una gran dependencia entre las ventas de una semana y la siguiente (por el conocimiento empírico que tenemos de cómo se consume el gas para calefacción: si en una semana alguien compra una garrafa de gas, es muy poco probable que a la semana siguiente compre otra). Esto nos lleva a pensar que una cadena de Markov de tercer orden, cuyos estados estarían representados por las parejas (ventas, semana), o por las ternas

(ventas, semana, aumento). Otra alternativa sería utilizar un modelo oculto de Markov discreto (ver [41]), con restricciones, de tercer orden, cuyos estados son (temperatura, semana) o (temperatura, semana, aumento) y las salidas (observaciones) las ventas. Dado que tomamos las ventas como un número entero, que no puede ser mayor que el stock máximo almacenable, el número de estados es finito. Las mismas consideraciones valen tomando las temperaturas, por ejemplo, con un dígito decimal de exactitud. Sin embargo, estas alternativas implican la evaluación de las probabilidades de transición entre los estados, lo que, dados los escasos datos disponibles, es imposible. La misma conclusión vale para el caso de quererlo modelar como una red estocástica. Finalmente, la consideración más importante es que, aún teniendo los datos en cantidad necesaria, en este trabajo se pretende utilizar redes neuronales, lo que implicaba que el modelo de la realidad a utilizar pudiera ser representado por una red neuronal lo que descartaría a los modelos ocultos de Markov (HMM) y las cadenas de Markov. Teniendo en cuenta lo recién dicho sobre las redes estocásticas, la red a usar será no-estocástica. Por lo tanto, nos inclinaremos a utilizar como modelo del sistema real un sistema dinámico estocástico.

El modelo elegido para la realidad es:

El estado del sistema en el instante n+1, $\mathbf{x}(n+1)$ *viene dado por*

$$\mathbf{x}(n+1) = \mathbf{F}[\mathbf{x}(n), \mathbf{x}(n-1), ...\mathbf{x}(n-T), \mathbf{u}(n+1), \mathbf{u}(n)...\mathbf{u}(n-T)] + \boldsymbol{\varepsilon}(n)$$

siendo $\mathbf{F}$ y $\boldsymbol{\varepsilon}$ *definidos como en* AVII.4.4 (ambos de componentes enteros) *y* $\mathbf{u}(n)$ *las entradas (temperaturas medias semanales) en el instante* n. *Los estados estarían representados por las ternas (ventas, semana, temperatura) o (ventas, semana, temperatura, indicador de aumento). La variable de la cual se tienen mediciones son las ventas,* v, *por lo que será*

$$v(n+1) = \varphi[\mathbf{v}_{0-\tau}(n), \mathbf{t}_{0-\tau}(n), \mathbf{a}_{-1-\tau}(n), \mathbf{s}_{-1-\tau}(n)] + \delta(n)$$

con

- v, t, a, s *las ventas, temperaturas máximas semanales promedio, el indicador de aumento y el número de semana respectivamente*
- $\mathbf{v}_{x-\tau} = [v(n-x), v(n-x-1), ...v(n-\tau)]$ *y análogamente para* $\mathbf{a}_{x-\tau}$ y $\mathbf{s}_{x-\tau}$
- $\delta(n)$ *una variable aleatoria entera.*

Suponemos que no existe ruido en los valores de las temperaturas, por lo que toda la aleatoriedad es atribuible al "shift" $\boldsymbol{\varepsilon}(n)$ *que toma un número finito de valores enteros distintos, lo que hace que el sistema constituya un IFS* (ver AVII.1.1).

Este IFS tendrá un subespacio de la señal del cual deberemos determinar su "delay" óptimo y dimensión

Por otra parte, este modelo de sistema dinámico será representado mediante el modelo estadístico dado por una red neuronal [1]. Se utilizarán varias topologías de redes neuronales distintas, pero todas ellas no estocásticas. De entre esas redes, las dinámicas serán aquellas tales que cada una tendrá asociada a su vez un modelo de espacio de estados. Por ejemplo, consideremos la red de Jordan utilizada, de la forma indicada en la figura III.11. En dicha red suponemos que la capa de entrada no realiza ningún proceso sobre los datos $\mathbf{U}$ y que $\mathbf{Fc}$, $\mathbf{F1}$, $\mathbf{F2}$ y $\mathbf{F3}$ son las funciones (vectoriales) de salida representadas por la capa de contexto, la primera capa oculta, segunda capa oculta y de salida respectivamente y τ es la constante tiempo, tal como se describe en III.3.2 "Redes recurrentes". Esta red, una vez entrenada, tendrá un espacio de estados definido por los estados $\mathbf{X}^* = \mathbf{Xc}$ y la dinámica estará dada por:

$$\begin{cases} \mathbf{Xi}(n) = \mathbf{U}(n) \\ \mathbf{X1}(n) = \mathbf{F1}[\mathbf{WcXc}(n) + \mathbf{WiXi}(n)] \\ \mathbf{X2}(n) = \mathbf{F2}[\mathbf{X1}(n)] \\ \mathbf{Y}(n) = \mathbf{F3}[\mathbf{X2}(n)] \\ \mathbf{Xc}(n+1) = \mathbf{Fc}[\tau\mathbf{Xc}(n) + \mathbf{Y}(n)] \end{cases} \qquad \textbf{III.2.1}$$

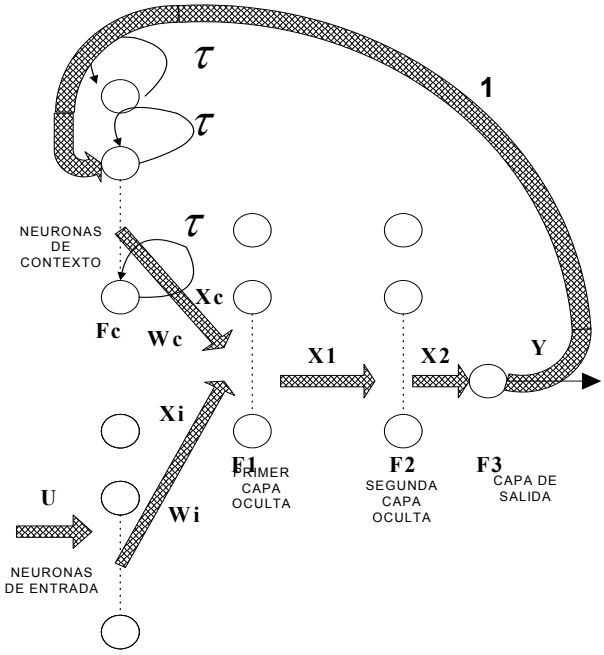


Figura III.11

Por otra parte, durante el entrenamiento se modifican los pesos de acuerdo con los pares (**U**, **d**), por lo cual en este caso los pesos **W**(*n*) correspondientes al paso *n* de entrenamiento junto con **X***, serían los que representarían el estado del sistema en el instante *n* y la dinámica del sistema estaría dada por las ecuaciones III.2.1 más las asociadas a la actualización de los pesos. El orden del sistema será entonces dim**X***=dim**X**c+ dim**W.**

III.2.3 Otros estudios realizados

Cálculo del "delay" óptimo

Para calcular el "delay" óptimo de la serie temporal utilizamos el "Average Mutual Information" de la serie respecto sí misma. Este método se puede utilizar tanto para series provenientes de sistemas determinísticos como estocásticos, ya que en su planteo no hace ninguna referencia a dicha condición del sistema, aunque en el caso estocástico los resultados obtenidos son válidos para la realización del proceso aleatorio en estudio. Realizamos un estudio del AMI de la serie utilizando la herramienta VRA y obtuvimos los resultados de la figura III.12:

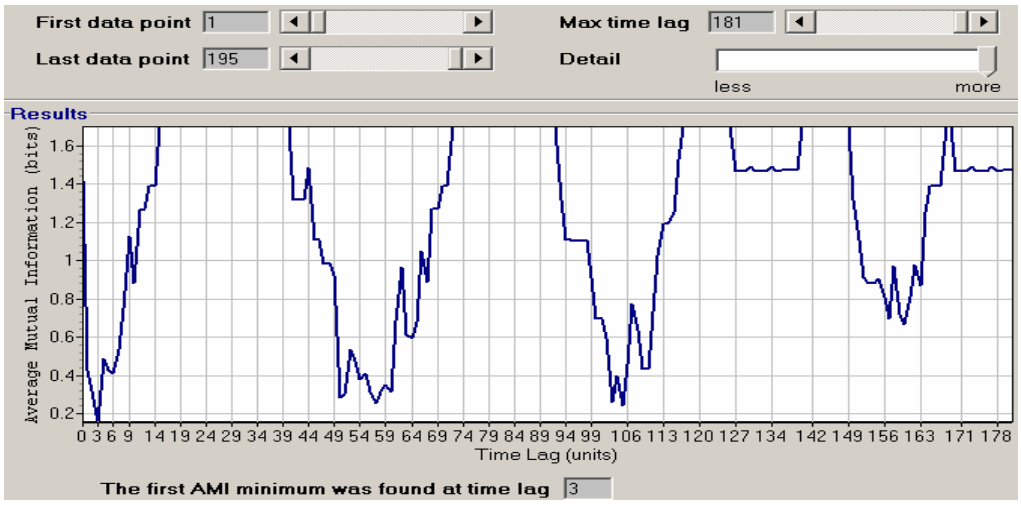


Figura III.12

Se puede observar que la herramienta maneja un parámetro "detalle" ("detail") que se refiere directamente al "detalle" (precisión con que se dibuja el gráfico del AMI) y por lo tanto, la precisión con que se calcula el primer mínimo (y por ende, el "delay" óptimo). En productos tales como el TISEAN o el DATAPLORE, en lugar de indicarse mayor o menor nivel de detalle, se deben explicitar los parámetros usados en los algoritmos respectivos: para este caso, en el TISEAN, sería el número de cajas ("boxes") a utilizar. Vemos que para no todos los niveles de detalle el "delay" óptimo *d* es 3 sino que al bajar el nivel de detalle pasa a 2 (la herramienta VRA sugiere que cuando se trabaja con el menor nivel de detalle, el "delay" óptimo puede cambiar). Estos valores concuerdan con los estudios estadísticos (III.2.1): las relaciones entre las ventas no van más allá de tres semanas.

Cálculo de la dimensión del "embedding"

Como los datos se consideran como provenientes de un sistema dinámico, a los efectos de poder minimizar las entradas a la red se calculan ciertos parámetros del sistema, tales como la dimensión del "embedding" (Ver AVII.4.4).
Dada que la serie es aproximadamente periódica de periodo 52, la dimensión del subespacio de la señal deberá ser $2 \le m \ll 52$ (Ver AVII.1.5). Como existe un ruido dinámico, a esa dimensión hay que sumarle la del "shift space", y teniendo en cuenta que el número posible de valores del ruido es finito (o sea que tiene dimensión 0), se llega a que las dimensiones posibles para el "embedding" son las mismas que si el sistema fuese dinámico determinístico. Por lo tanto, podemos calcular ese m con el método de los FNN, como si el sistema fuese determinístico.
El DATAPLORE , en su versión de demostración, arrojó un valor de m = 2 con un porcentaje de vecinos falsos muy cercano al 0% para d = 2 y 3.

Para d = 2, la gráfica de los FNN obtenida fue:

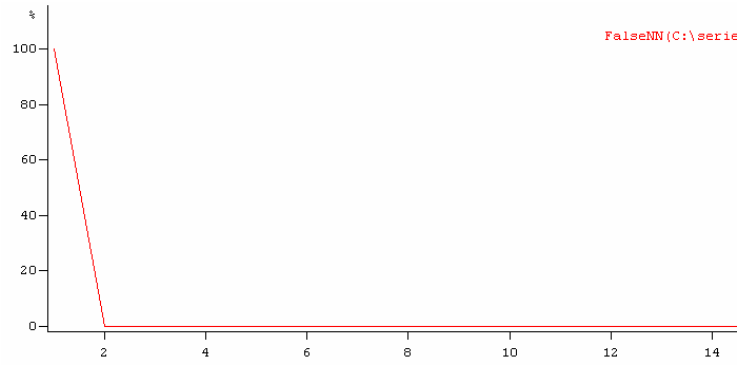


Figura III.13

Para d=3 se obtuvo la misma gráfica. Los parámetros usados fueron: dimensión máxima del "embedding" = 15, "distance tolerance" (RTOL)= 10.0 y "Loneliness tolerance" (ATOL)= 2.0

Cabe aclarar que en el presente trabajo solo consideramos la dimensión global del "embedding" debido a que: a) no disponemos de un software que calcule la dimensión local d_L (ver AVII.4.4) y b) los resultados obtenidos utilizando la dimensión global $m = d_E$ son válidos (ya que no se pierde información sobre el "embedding" al trabajar con una dimensión mayor que d_L y c) el d_E calculado fue $d_E \le 3$ lo cual no deja muchas posibilidades para d_L y por lo tanto, utilizar d_L en lugar de d_E no implicaría, en términos absolutos, una gran reducción en la dimensión.

Como verificación, probamos con m = 2 y m = 3 entrenando una red MLP de 11 neuronas ocultas en una sola capa, con d = 2 y d =3. Los mejores resultados los obtuvimos para m = 2, por lo que consideramos que esa es la dimensión buscada:

m=3 d= 2 Entradas V(i), V(i-2), V(i-4) prediciendo V(i+1). Se hicieron 20 entrenamientos El %Error CV mínimo fue del 40%

m=3 d=3 Entradas V(i), V(i-3), V(i-6) prediciendo V(i+1). Se hicieron 20 entrenamientos El %Error CV mínimo fue del 40%.

m=2 d=3 Entradas V(i), V(i-3) prediciendo V(i+1). Se hicieron 20 entrenamientos El %Error CV mínimo fue del 14.7%

m=2 d=2 Entradas V(i), V(i-2) prediciendo V(i+1). Se hicieron 20 entrenamientos El %Error CV mínimo fue del 20%.

Por lo tanto, aplicando el teorema de Takens-Mañé (para el caso estocástico), podremos predecir x_{i+1} a partir de $\{x_i, x_{i-d}, \ldots x_{i-(m-1)d}\}$. Sustituyendo por m=2, concluimos que un MLP que predijera el valor de la serie temporal en un paso (x_{i+1}) debería tener como entradas

$\{x_i, x_{i-3}\}$ para d=3

$\{x_i, x_{i-2}\}$ para d=2

siendo x_i el valor i-ésimo de las ventas semanales de gas, y como salida, una estimación de x_{i+1}

Esta aproximación siempre puede ser implementada por un MLP debido a las propiedades de aproximadores universales que tienen dichas redes (Ver AVIII.1 a AVIII.3).

"Recurrence plot" (RP) de la serie semanal

La herramienta VRA calcula la distancia euclidiana entre los vectores (en el caso del gráfico de abajo, las distancias entre todas las parejas de vectores $\{x_i, x_{i-d}\}$) y las expresa según una escala de colores o tonos de grises (mostrada en la parte superior derecha del gráfico). En los ejes se representarían la i correspondiente. Al gráfico obtenido se lo llama gráfico de recurrencia o "***recurrence plot***". La distribución de los colores y patrones nos da una idea de que existe cierto determinismo en el generador de la serie.

Realizamos un RP de la serie utilizando el VRA para los valores de "delay" óptimo d=2 y d=3 y dimensión del "embedding" m=2 y nos dio como resultado (Ver III.2.3 acerca del cálculo de d y m):

***d*=3 *m*=2** ***d*=2 *m*=2**

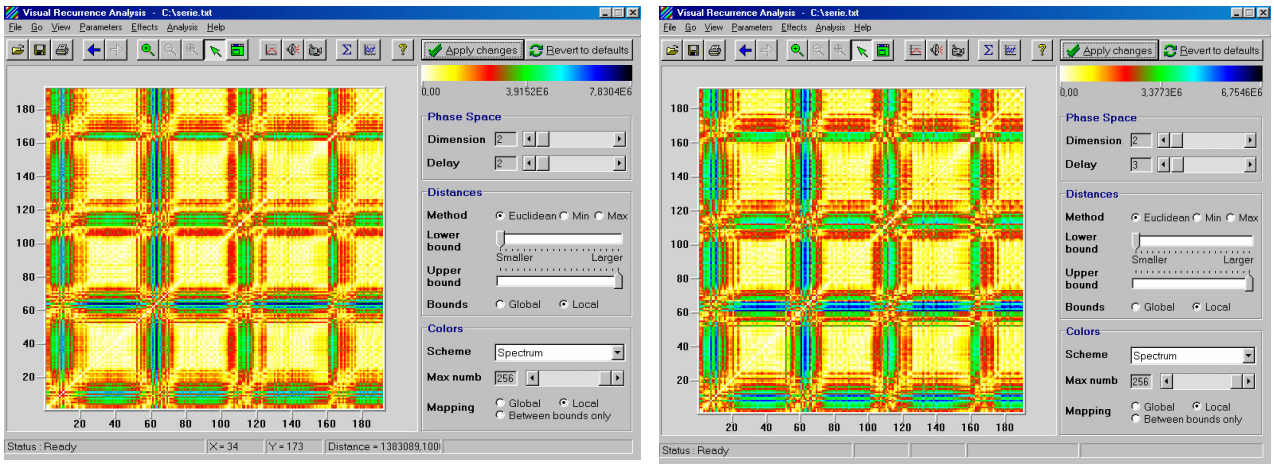


Figura III.14

Estos dos RP se complementan con el mostrado en la figura III.15 (obtenido con el TISEAN) para dar una idea de la estacionariedad del proceso:

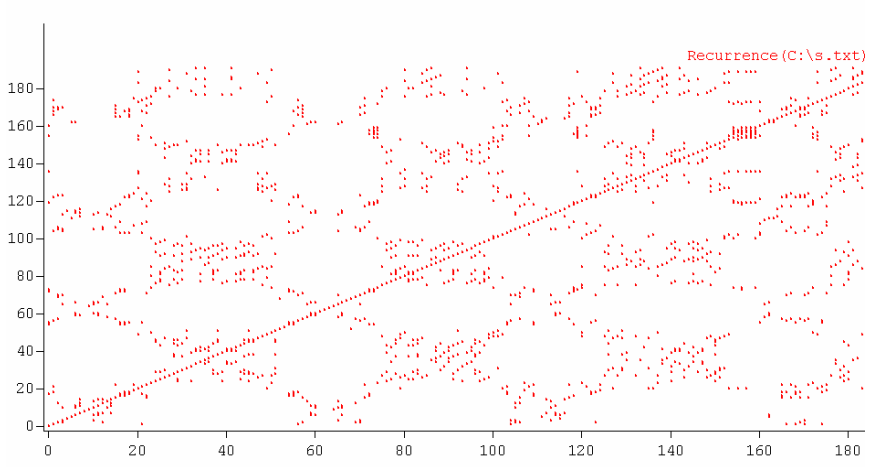


Figura III.15

Considerando la distribución de los puntos en el plano, que están dispersos uniformemente en patrones en forma de rombos que se separan de la línea de identidad, podemos concluir que el proceso es estacionario.

Entropías y resolubilidad del problema

Siguiendo a [13] tratamos de determinar si el problema tiene solución, es decir, si existía una red que pueda aprender la relación Entradas → Salidas, a partir de las entropías de las entradas y las salidas.
La relación es aprendible por una red si

a) $\dfrac{H(salidas|entradas)}{H(salidas)}$ es cercano 0

y

b) $\dfrac{I(entradas, salidas)}{H(salidas)}$ es cercano a 1

siendo $H(\bullet)$ la entropía, $H(\bullet|\bullet)$ la entropía condicional e $I(\bullet,\bullet)$ la información mutua. (Por una definición de entropías e información mutua, Ver Sec II.3).

Si bien las condiciones son simples, no pudimos contar con ningún software[11] que nos permitiera estimar las entropías ni la información mutua en forma directa a no ser que se calculen manualmente (a través de un procedimiento ad hoc) a partir de los datos disponibles. Adicionalmente encontramos el mismo problema que se vio en el caso de querer usar una cadena de Markov: para estimar estas probabilidades precisamos una cantidad de datos de la que no disponemos. Por lo tanto, este aspecto del problema no pudo ser aclarado directamente, quedando solo la posibilidad de demostrar que era resoluble en una forma constructiva, es decir, encontrando un modelo de red adecuado.

[11] El Statistica Neural Networks de StatSoft dice calcularlos, pero no pudimos conseguir ni siquiera una versión de demostración del mismo.

Predecibilidad de la secuencia de valores de ventas

Considerando los volúmenes de gas vendido como una serie temporal, nos planteamos si el sistema dinámico al que correspondía era caótico además de estocástico. Para ello utilizamos las rutinas del paquete TISEAN, en especial la Lyap_spec, con la que calculamos el espectro de exponentes de Lyapunov (Ver AVII.3.4). Obtuvimos

d	espectro $[\lambda_1, \lambda_2]$
2	[-1.218792e-01, -5.494868e-01]
3	[-1.153879e-01, -3.889482e-01]

En todos los casos el máximo exponente de Lyapunov λ_{MAX} es negativo (ver ANEXO VII) . Esto significa que no estamos en presencia de los problemas mencionados en AVII.3.4 al aplicarse el algoritmo de Sano y Sawada (utilizado por el TISEAN) al sistema estocástico y que el sistema dinámico no es caótico (al menos según la realización del proceso $\mathcal{P}$ en estudio) pero sí estocástico. Debemos aclarar que si bien el caótico es el sistema dinámico, se suele decir que la serie de observaciones que se tienen de ese sistema, y a partir de la cual se determina su condición de caótico es determinada, es caótica. O sea, para abreviar se suele decir que la serie temporal es caótica. Por otra parte, que la suma de los exponentes de Lyapunov sea negativa es consistente con el hecho de que es un sistema real.

III.3 Pruebas realizadas

Se ensayó con varios modelos de redes y para cada una de ellas se realizó un estudio de calidad de la predicción en modo "open loop" (a través de la evaluación del porcentaje de error en la predicción en un paso promediado sobre el conjunto de "cross validation") y de estabilidad (validez a largo plazo) en modo "closed loop", por medio de la serie sintética generada haciendo predicciones iteradas con cada modelo. Asimismo, dado que el algoritmo de BP realiza una búsqueda en el gradiente partiendo de un punto al azar sobre la superficie de error, se realizó el entrenamiento de cada red al menos siete veces, y se tomaron los parámetros (tasas de aprendizaje y pesos) que arrojaran los mejores valores de porcentaje de error. El entrenamiento que se realizó fue siempre de tipo incremental, lo que esta justificado por las ventajas que el mismo presenta sobre el "batch" (Ver Sección II).

Se realizaron pruebas con redes total y parcialmente recurrentes, redes de Jordan, redes TLFNs y MLPs. También se ensayó el "ensemble method" [55] que en nuestro caso (predicción usando varias redes neuronales) puede ser visto como el uso de un comité de redes ([11] y [1]).

III.3.1 Pre y post-proceso de los datos

Para que la red pueda ser entrenada, los datos deben ser codificados o re-codificados de una manera adecuada a tal efecto. Eso es lo que podríamos llamar el pre-proceso de los datos. De la misma forma, puede ser que sea necesario realizar alguna transformación a los datos de salida de la red antes de que puedan ser usables: es lo que constituye el post-proceso de datos. En la etapa de pre-proceso, algunos distinguen entre codificado y re-codificado. En el proceso de codificado, se convierten a una representación tal que puede ser usada para entrenar la red, y en el de re-codificado se trabaja sobre los datos, bien ya codificados o bien en bruto, de forma de poner de manifiesto características de importancia o de reducir su dimensionalidad. La transformada de Fourier de una serie temporal es un ejemplo de re-codificado [13]. En el caso de estudio, como casi todos los datos son numéricos, la codificación es un poco obvia. Por otra parte, dado que una de las metas de este trabajo es probar el potencial de las topologías más sofisticadas (TLFNs y redes recurrentes) al momento de aprender las características de la serie temporal, el pre proceso de los datos que se hizo (en cuanto a transformación de sus valores) fue mínimo.

Pre-proceso

Clasificación de los tipos de datos

Las entradas de la red pueden ser intrínsecamente continuas o discretas. Las continuas son:

- Temperaturas: anotadas como T+1, T, T-1....
- Volúmenes de venta: anotadas como V+1, V, V-1...

Por otra parte las discretas son:

- número de semana (S)
- indicador de si hubo o no aumento del precio de venta del gas en esa semana, las anteriores o las siguientes

Sin embargo, codificamos los datos de forma que algunas continuas se transformaran en enteras. Se codificaron los datos como:

- los volúmenes de las ventas de gas expresados en litros, como un número entero
- las semanas como un número del 1 al 53, para poner de manifiesto la característica periódica de la serie
- el indicador de aumentos de precio de venta del gas:
    - si en la semana ha habido un aumento, como aumento = 0
    - si la semana era previa a una de aumento, como aumento = -1
    - si era previa en dos semanas, aumento = -2
    - si era previa en más de dos semanas, aumento = 3
    - si era posterior a una semana con aumento, aumento = 1
    - si era posterior en dos semanas a una con aumento, aumento = 2
    - si era posterior en más de dos semanas, aumento = 3

Obsérvese que se le da el mismo valor a la variable aumento tres o más semanas antes o después de la semana en que ocurre la variación de precios: es para significar que no existe una influencia de los aumentos en las ventas (influencia especulativa antes o de retracción en las ventas después, del aumento) a largo plazo.

Representación de los datos

Representación física

Los datos de entrenamiento y prueba de la red debieron ser representados en formatos aceptados por el simulador elegido (NEUROSOLUTIONS). Lo que se hizo fue trabajar con archivos ASCII formateados en columnas, donde cada columna representa una variable (***canal***) ya sea de entrada como de salida.

Normalización de los datos

Los datos de entrada fueron normalizados automáticamente por el simulador de la siguiente forma:

- para cada variable ***i*** de entrada se calcula

$$Amp(i) = [CotaSup(i) - CotaInf(i)]/[Max(i) - Min(i)]$$
$$Despl(i) = CotaSup(i) - Amp(i) * Max(i)$$

donde

$Max(i)$	Es el máximo de los valores encontrados para esa variable
$Min(i)$	Es el mínimo de los valores encontrados para esa variable
$CotaSup(i)$	Una constante, por defecto 0.9
$CotaInf(i)$	Una constante, por defecto -0.9

- para el valor j-ésimo de la variable *i*, se calcula su valor normalizado como:

$$Valor(i, j) = Ampl(i) * Valor(i, j) + Despl(i)$$

Con esta normalización se lleva los datos de entrada de la variable *i* al rango [*CotaSup(i), CotaInf(i)*] tratando de evitar de esa forma la saturación de las neuronas durante el entrenamiento.

Este es un escalado lineal, lo que asegura que no se introduzca una estructuración en los datos que no estaba presente originalmente: si los datos estaban o no uniformemente distribuidos en el universo de valores posibles, luego del escalado lineal van a seguir estándolo. Si la transformación hubiese sido no lineal, se hubiesen magnificado los efectos de los datos de entrenamiento mal balanceados (mal distribuidos), agrupándolos en conjuntos que serían aún más dispersos [13]. A su vez, cuando se requiere la presentación de los datos de salida los datos de entrada normalizados son desnormalizados deshaciendo la transformación anterior para hacer los cálculos correspondientes y obtener así la salida Es debido a esta desnormalización que en algunos de los resultados experimentales pueden aparecer valores decimales para los valores deseados de la serie, cuando originalmente eran enteros: por ejemplo, 1821053.88 en lugar de 1821054, producto esto de los errores cometidos al normalizar-desnormalizar, ya que ese valor se tomó de la ventana del simulador como valor deseado al construir la tabla de resultados obtenidos.

Finalmente, *CotaInf y CotaSup* deberían ser adecuados a la función de salida que se use. En el caso de Tanh son las adecuadas, pero en el caso de la logística, debería ser *CotaInf* positivo y cercano a 0.

Remoción de "outliers"

Según [13] "los valores que esten anormalmente lejos del valor medio de la variable ("***outliers***") pueden tener un efecto al entrenar la red que es desproporcionado con el error que producen. Este efecto es empeorado si esos valores son producidos debido a ruidos. Por esa razón es conveniente remover los outliers antes de entrenar".
Utilizando el STATISTICA realizamos un gráfico de caja, obteniendo el gráfico de la figura III.16:

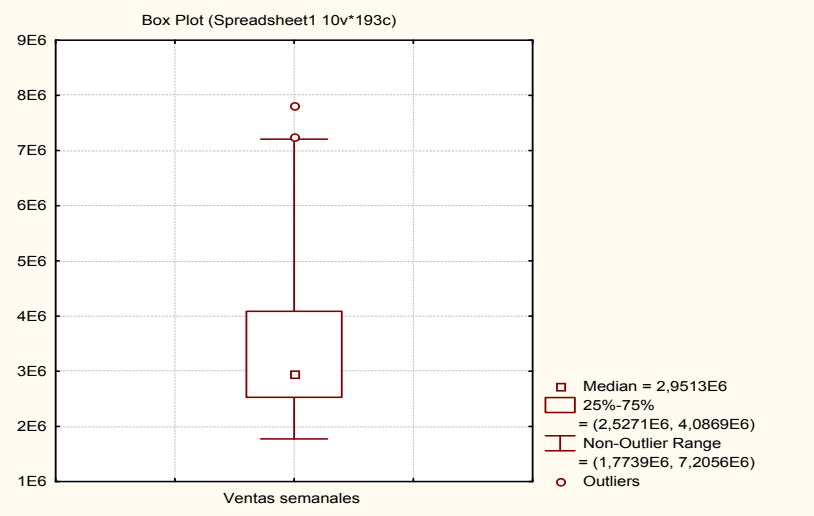


Figura III.16

es decir, serían outliers los valores correspondientes a ventas mayores que 7.2056E06 o menores a 1.7739E06. STATISTICA define un ***valor*** "outlier" si

$$\textbf{valor} > UBV + o.c.*(UBV - LBV) \quad o \quad \textbf{valor} < LBV - o.c.*(UBV - LBV)$$

donde

UBV es el extremo superior de la caja en el diagrama anterior, en este caso, el 75o. percentil

LBV es el extremo inferior de la caja, en este caso, el 25o. percentil

o.c. es el coeficiente de outlier. En este caso, usamos 1.5 (el valor por defecto para el cálculo de outliers en STATISTICA), con otros coeficientes se obtienen distintas cantidades de outliers, por ejemplo con *o.c.* = 2 se tiene un solo outlier; un *o.c.* <1.5 produciría más de dos outliers. Optamos por 1.5 por parecernos que dos era una cantidad adecuada de outliers para el volumen de datos disponibles

Es decir, tomando como centro la mediana y un coeficiente *o.c.* = 1.5 vemos que los outliers son aquellos con valor superior a 6426600= (4,0869+1,5*(4,0869-2,5271))*1000000 o inferior a 187400 = (2,5271-1,5*(4,0869-2,5271))*1000000. Se detectaron así 2 outliers: los correspondientes a las semanas 28 y 30 del año 2000. Dado que los datos son escasos, decidimos sustituir esos valores por la media de las ventas para esas semanas (es decir, el valor para la semana 28 del año 2000 es el promedio de los valores de las otras semanas 28 = 5505709,25 y análogamente para la 30) [13].

Este método de detección de "outliers" a partir de los percentiles fue preferido sobre el marcar como "outliers" a los datos que disten más de dos o tres desviaciones estándar de la media, por presuponer esto una distribución de probabilidad normal de los mismos.

Construcción de los datos de entrenamiento y testing

Los datos disponibles provienen de dos fuentes: reportes generados manualmente de las salidas (embarques) de gas diarias y de un datawarehouse con la facturación diaria resumida a nivel semanal y mensual. En las épocas de mayor demanda de gas, suele haber discrepancias entre dichos valores (por ejemplo porque se facturan ventas que se despachan a primera hora del día siguiente, antes de que las secciones administrativas esten trabajando), mientras que en el resto del año coinciden. Adicionalmente, la información del datawarehouse está disponible solo a partir de mayo/1999. La opción que se tomó fue considerar solo los datos del datawarehouse, ya que se presupone que son datos fiables, por ser los utilizados en todo el manejo contable de la empresa. Las temperaturas promedio semanales se calcularon promediando las temperaturas de lunes a domingo de la semana en cuestión. Esto introdujo una diferencia de aproximadamente 2.8% promedio respecto al promedio realizado sobre solamente los días en que se vendió gas. Esta diferencia es considerada como un ruido en la medición de la temperatura y por el momento es ignorada. (Ver IV.1.3).

Reducción de la dimensionalidad

Dado que las entradas factibles a un MLP podrían ser V, V-1,... T+1, T, T-1...para predecir V+1, se decidió en algunos casos hacer uso del teorema de Takens-Mañé para tratar de determinar cuales eran las entradas que permitían reconstruir (predecir) V+1 (ver III.3.2). Adicionalmente, como parte del pre-proceso se podría pensar en la determinación de cuales son las entradas más significativas a la red utilizando por ejemplo análisis de componentes principales (PCA) o alguna técnica de "feature extraction" (ver [1], [7], [13]). No se realizaron estas consideraciones aquí y en cambio se realizaron pruebas utilizando AGs.

Post-proceso de los datos

El único post-proceso que existió fue la desnormalización de los datos de salida, que al igual que la normalización de los datos de entrada, es un proceso automático.

III.3.2 Pruebas realizadas para diferentes topologías

Arquitecturas y modelos ensayados

Se define un ***ejemplar*** como un conjunto consecutivo de valores de la serie temporal, tantos como sea el largo de la trayectoria a aprender. Una ***época*** consiste en la presentación de todos los ejemplares a la entrada de la red.

En todos los experimentos realizados se tomó el MSE sobre el conjunto de "cross validation" como

$$MSE = \frac{\sum_{i=0}^{N} (d_i - y_i)^2}{N}$$

siendo N el número de ejemplares en el conjunto de "cross validation" y d_i, y_i los valores deseados y de salida para el ejemplar i-ésimo respectivamente y n el número de ejemplares de entrada.

El porcentaje de error promedio se calculó como

$$\%Error = \frac{100}{N} \sum_{i=0}^{N} \frac{|y_i - d_i|}{d_i}$$

teniendo N, i , d e y el mismo significado que para el MSE.

El MDL fue estimado como:

$$MDL = NLog(MSE) + \frac{1}{2} kLog(N)$$

siendo k el número de pesos de la red (Ver ANEXO V) y los logaritmos son naturales.
Adicionalmente, en el simulador de redes utilizado no se ha podido determinar, a partir de la documentación existente y de las pruebas realizadas, si se toma siempre el mismo juego de ejemplares para realizar la CV cada vez que se entrena la red, cuando se especifica que el conjunto de CV sea un porcentaje del de training. Debido a ello, siempre que se especificó el MDL sobre el conjunto de CV no es un solo MDL el que se tuvo en cuenta, sino que se entrenó un mínimo de siete veces la red y se tomó la de mejor MDL (que dado que el modelo es el mismo salvo los valores de los pesos, es equivalente a elegir la que tenga menor MSE) como referencia para ese modelo.

Siempre se entrenó utilizando como criterio de detención del entrenamiento el siguiente: "si luego de 50 épocas consecutivas no hay una disminución en el MSE del conjunto de CV, se detiene el aprendizaje".

Básicamente se ensayaron tres grandes grupos de redes: perceptrones multicapa, redes TLFNs y redes recurrentes. Para cada una de ellas se determinó el mejor modelo según su MDL sobre el conjunto de CV, y se validaron todos los modelos tal como se describe en el punto III.2.3 "Validación del modelo". Adicionalmente, se utilizó un comité de redes ("ensemble method"), para tratar de obtener una mejor predicción a partir de las dadas por las redes individualmente.

El criterio de validación a corto plazo fue que el error de predecir en un paso promediado sobre el conjunto de "cross validation" fuese no mayor que el 12%. La validación a largo plazo no fue excluyente, y se prefirió a los modelos que arrojaran menor error promedio al predecir en forma iterada.

Mejores resultados según el MDL

Se presentarán a continuación los mejores modelos encontrados, en el sentido de aquellos con mínimo MDL (ver AV.1 y AV.2).

Los mejores resultados obtenidos para cada una de los tipos de redes son:

a) para los MLPs: la mejor red MLP desde el punto de vista del MDL es aquella con entradas V(i) , V(i-2) que predice V(i+1), tres neuronas ocultas, sin ruido en sus entradas.

b) para las TLFNs: una red TDNN correspondiente a un delay óptimo de la serie d=3, memoria focused, entradas V, S, A, T-1 y salida V+1 y 8 neuronas ocultas en una capa.

c) para redes recurrentes: una red de Jordan/Elman con entradas S, A, A-1, V-1, T-1 y salida V, con una constante Tiempo = 0.5 (ver III.3.2 "Redes recurrentes") dos capas ocultas de 4 y 2 neuronas respectivamente.

d) para el "ensemble method":

En esta instancia no tenemos definido (a partir de la bibliografía consultada) cual es el MDL de un "ensemble" de modelos.

Los valores obtenidos para estos modelos se resumen en el cuadro "CUADRO RESUMEN MDL" (por el significado del modelo d), ver III.3.4):

CUADRO RESUMEN MDL

	a)	b)	c)	d)	
				d.a)	d.a)
MDL calculado sobre el conjunto de CV	**-52.94**	**-373.29**	**-23.68**	**No definido**	**No definido**
% Error sobre el conjunto de CV	**25.54%**	**13.03%**	**9.55%**	**No definido**	**No definido**
Arquitectura	**d=2, 3 neuronas ocultas en una sola capa**	**d=3, memoria focused, 8 neuronas ocultas**	**4/2 neuronas ocultas, topología ya descripta**	**No definido**	**No definido**
Nro. de pasos de predicción con error < 10%	**0**	**1**	**5**	**6**	**5**
Error de predicción en un paso para semana 2	**50.90%**	**2.85%**	**0.92%**	**4.83%**	**7.62%**
Error promedio en semana 2 a 18/2003	**--**	**23.77%**	**11.74%**	**4.37%**	**9.23%**
Error máximo	**> 50.90%**	**40.41%**	**37.22%**	**15.00%**	**29.14%**
Error mínimo	**--**	**2.75%**	**0.71%**	**0.05%**	**0.35%**

Las partes sombreadas indican los modelos donde se aplicaron AGs.

Si bien el MDL parece ser el indicador ideal para elegir el modelo a utilizar, hacer la elección según la estimación del MDL que de él da el simulador de redes neuronales puede conducir a predicciones muy pobres, como se discute en III.3.3 "Validación del modelo", por lo que hay que plantearse en qué circunstancias un modelo es válido y por lo tanto elegible.

Modelos válidos

Generalmente en la bibliografía (por ejemplo en [5]), al estudiar redes para la predicción de series temporales, en especial series caóticas determinísticas, se distingue entre modelos válidos a corto y a largo plazo.

Válidos a corto plazo

Podemos considerar las redes (modelos) válidas(os) solo a corto plazo como solo localmente válidas(os), ya que aproximan la serie unos pocos pasos en el futuro a partir de sus valores en T (por ejemplo, T = 190) instantes anteriores. Teniendo en cuenta que una de los objetivos de este trabajo es la predicción en base a una red que se re-entrenara periódicamente (por ejemplo cada semana), se detectaron siete modelos válidos a corto plazo, es decir, utilizables en la predicción de las serie temporal ya que arrojaron un %Error CV no mayor al 12%. Estos modelos podrían utilizar un número fijo de datos históricos (por ejemplo, las ventas de las 190 semanas pasadas). Los modelos válidos a corto plazo encontrados son:

- **a)** una red totalmente recurrente con entradas S, T, V-1 y salida V, 7 neuronas ocultas en una sola capa.
- **b)** red totalmente recurrente con entradas S, T, V-1, V-2, salida V, 7 neuronas ocultas en una sola capa
- **c)** una red de Jordan con entradas adicionales, dos capas ocultas de 4 y 2 neuronas (ver III.2.3.1.3)
- **d)** una red TDNN con entradas V, salida V+1 con 12 neuronas ocultas en una capa, memoria "no focused", tomando un "delay" de 3 entre las muestras
- **e)** ídem pero con entradas S, A, T-1 y salida V+1 (ver más adelante, "Las TDNNs")

f) una TLFN con memoria gamma, entradas S, A, V-1, T-1, S-2, V-2, T-2 salida V , 17 neuronas ocultas
g) red MLP de entradas T, A-1,V-1,V-2,S-3 y salida V con 7 neuronas ocultas en una sola capa

El criterio de validación fue que el %ErrorCV fuese menor que 12%.

Siempre se utilizaron funciones de transferencia Tanh, entrenando utilizando descenso en el gradiente (BPTT) mejorado con un término de momento de primer orden con tasa de momento constante.

Resumen de los modelos válidos a corto plazo: **CUADRO III.3.1**

Modelo	a)	b)	c)	d)	e)	f)	g)	Ensemble	
								d.a)	d.b)
MDL calculado sobre el conjunto de CV	62.20	75.17	-23.61	-66.3	298.52	213.13	7.92	No definido	No definido
% Error sobre el conjunto de CV	11.49%	11.86%	9.55%	11.88%	12.00%	11.49%	11.90%	No definido	No definido
Nro. de pasos de predicción con error < 10%	0	0	5	0	0	0	3	6	5
Error de predicción en un paso para semana 2	13.66%	20.42%	0.92%	13.47%	16.28%	22.55%	6.24%	4.85%	7.62%
Error promedio en semana 2 a 18/2003	14.14%	14.91%	11.11%	11.33%	23.71%	48.10%	24.94%	4.37%	9.23%
Error máximo	42.65%	51.28%	37.22%	31.04%	88.28%	116.9%	81.12%	15.00%	29.14%
Error mínimo	1.03%	0.29%	0.71%	2.28%	4.31%	3.10%	1.23%	0.05%	0.35%

***De entre estos modelos el mejor (según el MDL) fue una TDNN con memoria no focused con 12 neuronas ocultas, entradas V, V-3 y salidas V+1 (el* d)**)

Selección del modelo a utilizar para predecir

El modelo constituido por el comité de las siete redes nos dio el mejor modelo en el sentido de menor error de predicción : fue la mejor aproximación, en cuanto

- al porcentaje de error promedio al predecir las semanas 2-18/2003,
- al error cometido al predecir en el primer paso (semana 2/2003) y
- al máximo error de predicción

y se obtuvo al ponderar los valores predichos por las siete redes válidas a) a g) según los coeficientes derivados a partir de la matriz de correlación de los errores (Ver II.2.3), llamado modelo **d.a)** en los cuadros anteriores.

Los coeficientes utilizados en el modelo **d.a)** fueron:

CUADRO III.3.2

Modelo	Coeficiente
red totalmente recurrente con entradas S, T, V-1 y salida V, 7 neuronas ocultas en una sola capa.	-0.155
red totalmente recurrente con entradas S, T, V-1, V-2, salida V, 7 neuronas ocultas en una sola capa	-0.27
red de Jordan con entradas adicionales, dos capas ocultas de 4 y 2 neuronas (ver III.2.3.1.3)	0.46
red TDNN con entradas V, salida V+1 con 12 neuronas ocultas en una capa, memoria no focused, tomando un delay de 3 entre las muestras	1.31
ídem pero con entradas S, A, T-1 y salida V+1 (ver III.2.2.4)	-0.08
una TLFN con memoria gamma, entradas S, A, V-1, T-1, S-2, V-2, T-2 salida V , 17 neuronas ocultas	-0.35
red MLP de entradas T, A-1,V-1,V-2,S-3 y salida V con 7 neuronas ocultas en una sola capa	0.088

Por otra parte, dado que trabajar con siete redes es algo engorroso (ya que hay que realizar las predicciones para cada una de ellas, lo que incluso puede significar entrenarlas de nuevo), redujimos la cantidad de redes a considerar a dos, equiponderando sus predicciones. Los valores arrojados al hacer esto con una red de Jordan y una TDNN fueron todavía satisfactorios (ver III.3.3 "Modelos válidos para la predicción"). Este modelo es el **d.b)** del mismo cuadro.

El modelo elegido para predecir será entonces aquel cuyos valores son el promedio de los valores dados por una red de Jordan y una TDNN tal como se describe en III.3.3

Válidos a largo plazo

En los sistemas determinísticos se considera el estudio de la validez a largo plazo a los efectos de verificar que el modelo reproduzca el comportamiento (dinámica) del sistema original, para lo cual se suelen comparar los invariantes de la serie original y la serie predicha en modo "closed loop". Para el caso de sistemas dinámicos estocásticos se tiene una densidad de probabilidad para cada valor de los invariantes, es más, en algunos trabajos se han considerado comparaciones entre las densidades de probabilidad de algunos de los invariantes (la dimensión de correlación y la entropía de Kolmogorov-Sinai) del sistema original y de la serie generada por el modelo [63]) como forma de validación a largo plazo. Adicionalmente, en el caso estocástico, sabemos que lo más que podemos predecir es una media (la asociada a la probabilidad condicional de que el sistema pase al estado n+1 sabiendo que estuvo en los n, n-1,...n-T), mientras que el valor real de la serie será el valor asociado a dicha media más una variable aleatoria. El modelo a largo plazo estaría formado por una red que predijera esa media (que implemente la función φ de AVII.1.1 "El enfoque estocástico") y opcionalmente otro modelo para la predicción de la variable aleatoria (el δ de AVII.1.1 "El enfoque estocástico"). Ese segundo modelo podría ser un modelo paramétrico (una función de densidad ajustada), una red de mezclas de modelos ("network mixture model" , Ver [2, Cap.6]), una red "feedforward" o incluso otro modelo estadístico cualquiera. Dado que disponemos de solo 18 valores (los de "testing") para ajustar los errores cometidos no intentaremos determinar dicho modelo de predicción del error ni descartaremos soluciones basándonos en la validez a largo plazo. La comprobación de la validez a largo lazo la sustituimos por el cálculo de los valores en forma iterada para las semanas de "testing" y sus respectivos errores. Preferiremos las redes que hayan tenido menor error promedio, y en ese sentido decimos que la red tiene un comportamiento bueno a largo plazo. A su vez, dentro de los que tengan un buen comportamiento respecto a los errores iterados, elegiremos aquellos con invariantes similares a los de la serie original.

Al utilizar el "ensemble method" con las series sintéticas generadas al predecir con los distintos modelos válidos a corto plazo en modo closed loop, obtuvimos una comprobación de que el método parece representar un modelo válido en el largo plazo, cuando se promedian los resultados de modelos válidos a largo plazo y pierde la propiedad de mejorar los resultados de un conjunto cualquiera de redes: si se promedian los valores de las cinco redes que tienen un menor error promedio iterado, utilizando los coeficientes de ponderación derivados de los errores iterados, se obtiene un error intermedio a los dados por las redes en forma individual, y si los coeficientes son calculados a partir de los errores no iterados, se mejora algo (alrededor del 15%) respecto al mínimo error promedio cometido por una red en el conjunto (ver III.3.4).

Otro criterio podría haber sido, teniendo en cuenta la aplicación que va a tener el modelo, el de que un modelo es válido a largo plazo si puede predecir más de un paso en el futuro, sin reentrenar, con un error no mayor que el 12%. En ese caso, los modelos válidos hubiesen sido la red de Jordan (modelo **c)**), el MLP con aumento (modelo **g)**) y el formado por el ensemble de las otras (modelos **d.a)** y **d.b)**).

Las redes que presentaron un buen comportamiento a largo plazo en el sentido de un porcentaje de error iterado promedio bajo fueron (en orden creciente de error promedio):

1) una red de Jordan con entradas adicionales, dos capas ocultas de 4 y 2 neuronas (ver más adelante, "Redes recurrentes")
2) red totalmente recurrente con entradas S, T, V-1, salida V, 7 neuronas ocultas en una sola capa
3) una TLFN con memoria gamma, entradas S, A, V-1, T-1, S-2, V-2, T-2 salida V , 17 neuronas ocultas

siendo la red 1) la más recomendable. En el cuadro siguiente se muestran los errores obtenidos al iterar los valores en cada una de las redes, para las semanas 2 a 18/2003:

CUADRO III.3.3

	1)	2)	3)	4)	5)	6)	7)	
Semana	**Recurrente sin V-2**	**TDNN c/ aumento no focused d=3**	**TLFN Gamma con aumento d=3**	**Jordan c/ aumento**	**Recurr con v-2**	**MLP con aumento**	**TDNN pura no focused d=3**	**Ensemble con coefs. Óptimos**
2	13.66%	16.28%	22.51%	0.92%	20.42%	6.24%	15.56%	4.85%
3	6.62%	5.74%	14.27%	8.19%	10.80%	74.43%	27.13%	38.88%
4	2.89%	15.44%	5.37%	9.04%	10.51%	73.43%	4.50%	6.32%
5	3.13%	24.77%	3.77%	2.87%	23.14%	86.82%	20.22%	29.91%
6	2.91%	20.25%	0.19%	0.85%	25.94%	89.08%	15.89%	22.08%
7	42.99%	35.72%	48.88%	45.27%	30.87%	5.02%	33.97%	29.62%
8	48.23%	85.52%	45.32%	35.28%	69.05%	159.37%	15.12%	1.55%
9	34.47%	13.85%	29.72%	31.61%	65.36%	144.51%	72.40%	87.90%
10	1.82%	25.97%	5.99%	9.37%	11.45%	73.04%	36.26%	50.72%
11	6.08%	9.74%	15.34%	11.56%	7.06%	61.85%	35.10%	51.59%
12	1.06%	17.12%	5.54%	2.61%	17.20%	76.63%	76.78%	102.18%
13	22.63%	70.04%	15.10%	16.54%	39.14%	114.57%	102.39%	127.16%
14	5.72%	24.49%	19.78%	17.08%	2.24%	51.24%	31.80%	45.06%
15	2.30%	15.17%	12.89%	0.62%	3.64%	54.87%	18.15%	31.87%
16	14.58%	37.84%	6.95%	19.51%	18.44%	80.85%	4.34%	1.99%
17	17.98%	47.43%	3.08%	22.60%	17.72%	78.62%	19.32%	20.14%
18	26.66%	16.28%	27.61%	13.50%	22.32%	16.02%	38.58%	33.92%
Errores prom:	**14.92%**	**28.33%**	**16.61%**	**14.55%**	**23.25%**	**73.33%**	**33.38%**	**40.34%**

Los valores del "ensemble" se calcularon ponderando según los coeficientes dados en el cuadro III.3.2.

A corto y largo plazo

Con respecto a los modelos válidos tanto a largo como a corto plazo, es decir, a los que se podrían emplear para una reconstrucción dinámica, como ya se dijo, serán los que produjeron un menor error iterado, o sea (en orden decreciente de error):

- una red de Jordan con entradas adicionales, dos capas ocultas de 4 y 2 neuronas (ver más adelante, "Redes recurrentes")
- red totalmente recurrente con entradas S, T, V-1, salida V, 7 neuronas ocultas en una sola capa
- una TLFN con memoria gamma, entradas S, A, V-1, T-1, S-2, V-2, T-2 salida V , 17 neuronas ocultas

En ese orden de preferencia. Estos modelos serían los que se podrían recomendar a quien quisiera predecir la serie temporal utilizando una sola red.

Observamos que el "ensemble method" aplicado a los valores iterados no mejora el error promedio. Esto puede deberse a que todas las consideraciones teóricas se hicieron pensando en un error en un paso y a que la TDNN en varios casos da errores iterados muy altos (marcados en gris en el cuadro III.3.3), los que se reflejan y amplifican en el error del "ensemble" ya que su coeficiente en la mezcla es 1,31. Adicionalmente, el "ensemble" de las redes 1), 2), 3), 4) y 5) (es decir, de las que tenían un mejor comportamiento a largo plazo) ponderado según los mismos coeficientes dio un error que solamente es intermedio a los errores: 17.58% y un máximo del 52.24%, que también es intermedio.

Mejores resultados según el MDL detallados por arquitectura

Se detallan a continuación los mejores resultados obtenidos para cada una de las arquitecturas y sus variantes.

Perceptrones multicapa

Perceptrones basados solo en la serie temporal

Uno de los modelos de red ensayados fue el perceptrón multicapa (MLP), por ser una topología clásica que ya ha sido usada como punto de referencia en este tipo de trabajos ([59], [57], [15], [24]).

A partir de los resultados obtenidos en el estudio realizado para la determinación del "delay" óptimo y de la dimensión del "embedding", se realizaron pruebas con un perceptrón con

Delay	Entradas	Salidas
d=3	$\{x_i, x_{i-3}\}$	x_{i+1}
d=2	$\{x_i, x_{i-2}\}$	x_{i+1}

donde x_i representa las ventas semanales de la semana i-ésima, i=1,2,...53.

Siempre se utilizaron redes de una sola capa oculta ya que 1) en teoría una capa es suficiente para aproximar cualquier mapeo y 2) no teníamos interés en acelerar el proceso de entrenamiento de la red, porque insume unos pocos minutos (del $O(1)$) de proceso. Las cantidades de neuronas de la capa oculta fueron determinadas genéticamente a través de 1500 generaciones.

La mejor red fue una con entradas V, V-2, 3 neuronas ocultas, sin ruidos adicionados en sus entradas.

Perceptrones con entradas adicionales

Se intentó mejorar la performance de los MLPs utilizando uno que tuviera como entradas las determinadas genéticamente a partir de 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 determinándose las entradas: T, A-1, V-1,V-2, S-3 y 7 neuronas ocultas. El modelo obtenido fue el **b).**

Cuadro resumen para los MLPs:

	a)	**b)**
MDL calculado sobre el conjunto de CV	-52.94	7.92
% Error sobre el conjunto de CV	25.54%	11.90%
Arquitectura	d=2, 3 neuronas ocultas en una sola capa	7 neuronas ocultas en una capa
Nro. de pasos de predicción con error < 10%	0	3
Error de predicción en un paso para semana 2	50.90%	6.24%
Error promedio en semana 2 a 18/2003	--	26.41%
Error máximo	> 50.90%	81.12%
Error mínimo	--	1.23%

a) MLP con entradas V, V-2, 3 neuronas ocultas, sin ruidos adicionados en sus entradas

b) MLP de entradas: T, A-1, V-1,V-2, S-3 y 7 neuronas ocultas

Vemos que si bien el MDL estimado para el modelo a) es mejor que el del b), los mejores resultados corresponden al b).

En resumen, la mejor red MLP desde el punto de vista del MDL fue aquella con entradas V(i) , V(i-2) que predice V(i+1), tres neuronas ocultas, sin ruido en sus entradas, para la cual:

MDL calculado sobre el conjunto de CV: -52.94
% Error sobre el conjunto de CV: 25.54%
d=3, 11 neuronas ocultas en una sola capa
Nro. de pasos de predicción con error < 10% = 0
Error de predicción en un paso para semana 2= 50.90%
Error promedio en semana 2 a 18/2003 = --
Error máx.: > 50.90%
Error min.: --

Las redes "time lagged feedforward" (TLFNs)

Otro de los modelos de red considerado fue el de una red "time lagged feedforward". En una primera aproximación consideramos los valores arrojados por el estudio de la dimensionalidad y "delay" óptimo del "embedding", por lo que realizamos pruebas tratando de "recordar" con "delays" de 3 o 2 elementos del pasado (Ver III.2.2).

Se ensayaron memorias "focused" y "no-focused" (Ver II.4.6). Utilizamos una sola capa de neuronas ocultas, determinando su número mediante una selección genética de más de 1500 generaciones de 50 a 60 individuos cada una. El entrenamiento siempre se hizo incremental ("on line"), utilizando un término de momento con tasa de momento no adaptativa. El porcentaje de datos usados para "cross validation" fue el 15% de los datos de entrenamiento. Los tipos de elementos de memoria ensayados fueron las memorias por "delay" y las gamma, las primeras por ser a las estructuras de memorias lo que el perceptrón es a las redes neuronales, y la segunda por permitir hacer experimentos con ella (tales como cambiar la profundidad de la memoria sin necesidad de variar la topología de la red). Como en todo el resto de este trabajo, las trayectorias a aprender eran de largo 4. Para ambos tipos de memoria, se intentó en una primera aproximación predecir el valor siguiente de la serie utilizando el teorema de Takens, es decir, dándole a la red como entrada solo las ventas (la red "pura") y utilizando los parámetros de las memorias ("taps" y "delay") para implementar las entradas demoradas $\{x_i, x_{i-d}\}$ siendo d =2 o 3. En una segunda aproximación, se incorporaron a las entradas las temperaturas máximas medias semanales, la variable aumento y el número de semana, tanto para la semana a predecir como para las dos anteriores y también las ventas anteriores. Las entradas pertinentes se determinaron genéticamente, tal como se justifica en II.1.3 "Métodos no basados en derivadas" y [30].

Respecto a los algoritmos genéticos empleados y a sus parámetros, podemos anotar que:

- A efectos de uniformizar la nomenclatura con la usada por el simulador de redes elegido, cada solución factible (en nuestro caso, una red) es llamada un cromosoma. Un cromosoma esta compuesto de una colección de genes, que son simplemente los parámetros de la red que queremos optimizar genéticamente.

- El algoritmo genético crea una población inicial (una colección de cromosomas) y luego la evalúa entrenando una red que se corresponde con un cromosoma. Luego evoluciona la población a través de múltiples generaciones, tratando de encontrar los mejores parámetros para la red. Si lo que se busca es optimizar un parámetro de la red, se hace evolucionar la población (variando ese parámetro) buscando la red con mayor performance (mejor "fitness"), sino, cuando los parámetros definen la propia estructura de la red (por ejemplo cuando se quiere determinar el número de neuronas de una capa), la evolución se hace en realidad buscando la estructura que de la mejor red más que un parámetro óptimo.

- La evolución se hizo en forma generacional: la población entera fue reemplazada en cada iteración. Este método ha probado dar buenos resultados en una gran variedad de casos y tiende a evitar los mínimos locales de mejor forma que si se descartaran solo los miembros con peor valor de "fitness" en cada iteración [12].

- Cada generación constó de 50 o 60 individuos. En algunas pruebas preliminares realizadas con generaciones de 100 individuos, no se obtuvieron mejores resultados y se enlenteció demasiado el proceso.

- Se buscó minimizar el porcentaje de error de predicción en un paso sobre el conjunto de "cross validation", para obtener así una red con buenas capacidades de generalización. El porcentaje de error es significativo ya que la magnitud de los errores cometidos es siempre mayor que 1 y minimizarlo es equivalente a minimizar el MSE dado un número de ejemplares de CV fijo (Ver III.3.3 "Validación a corto plazo").

- Los cromosomas que pasaban a la generación siguiente eran elegidos con una probabilidad proporcional a su "fitness".

- La probabilidad de cruzamiento entre dos cromosomas era 0.9. El cruzamiento se realizaba en un solo punto (elegido aleatoriamente) de los genes del cromosoma: por ejemplo, dados los padres

  A 11001|010
  B 00100|111

  Luego de intercambiar los genes de los padres en el punto de cruza elegido (marcado con |), se obtienen los siguientes descendientes:

  A1 11001|111

- La probabilidad de mutación de un cromosoma era de 0.01

- Las poblaciones se hicieron evolucionar durante 1500 generaciones como mínimo, y se continuó hasta que no se encontrara mejora en la función de fitness en más de 500 generaciones consecutivas.

- En todas las redes se usó función de salida Tanh (la recomendada por [12]) en todas sus neuronas, una sola capa oculta, entrenamiento "on-line", porcentaje de datos para CV = 15%, descenso en el gradiente mejorado con un momento de tasa fija (no adaptativa). Siempre se realizó la predicción en un paso.

Se pudo observar que

- La calidad de la solución obtenida por un método genético esta fuertemente asociada al número de generaciones de individuos usados.

- Como es regla general, al aumentar las generaciones, los tiempos de entrenamiento se empiezan a disparar: para hacer la selección del número de neuronas con 1500 generaciones, se demora más de 20 minutos en un PC Pentium III a 900Mhz y 256Mbytes de memoria, contra los 15 segundos que demora entrenar[12] la red de topología fija (7 neuronas ocultas). Para el caso de 5000 generaciones con poblaciones de 100 individuos, más de 4 horas.

- Tal vez se pudieran obtener mejores resultados al usar AGs, si se hubiesen ajustado más adecuadamente los parámetros usados en la optimización, pero ello hubiese requerido conocimientos de AGs que estaban fuera del alcance de este trabajo.

Las TDNNs

Dado que las ventas correspondientes a las semanas próximas a un aumento del precio de venta del gas se ven afectadas por fenómenos especulativos, etc., incorporamos la entrada "aumento" con el significado y codificación ya visto en III.3.1.

Utilizamos para entrenamiento las semanas 18/99 a 1/2003 y para "testing" las 2/2003-18/2003. En una primera instancia, intentamos utilizar redes TDNNs en su forma "pura", es decir, tal como lo indicaría el teorema de Takens-Mañé: usando el "delay" y la "dimensión" del "embedding" obtenido en III.2.2 y como entradas, solo la serie de ventas semanales para predecir las ventas de la semana siguiente. Las capas ocultas fueron 1, con un número de neuronas determinados genéticamente, un porcentaje de datos para "cross validation" del 15%, entrenamiento "on line".

El mejor resultado desde el punto de vista del MDL fue para el caso de redes TDNNs con entradas solo las ventas y salida la predicción V es llamado el modelo **a)** del CUADRO III.3.4.

Las TDNNs con entradas adicionales ("aumento")

A los efectos de mejorar la predicción, añadimos variables explicativas a la entrada de la red. Trabajando con d=3 y d=2, los mejores resultados se obtuvieron con una red TDNN con d=3, memoria "focused", entradas S, A, T-1 (determinadas genéticamente a partir de las posibles entradas adicionales: S, A, T, S-1- A-1, V-1, T-1, S-2, A-2,V-2, T-2) salida V y 8 neuronas ocultas y constituye el modelo **b)** del CUADRO III.3.4.

Observamos que la adición de variables explicativas parece haber mejorado la predicción en un paso, aunque el porcentaje de error promedio de predicción (sobre el conjunto de "testing" o de CV) es mejor en el caso de la red "pura".

[12] Es decir, a partir de los datos de entrenamiento y usando el criterio de detención, entrenar la red. No se considera el tiempo necesario para realizar varios entrenamientos y ver cual arroja mejores resultados.

Las TLFNs con memoria gamma

Análogamente que para las TDNNs, intentamos aplicar el teorema de Takens-Mañé utilizando esta vez estructuras de memoria tipo gamma. Se realizaron las pruebas en las mismas condiciones que para el caso de las TDNNs, pero utilizando memorias gamma[13] de profundidad = 15. En ambos casos el largo de la trayectoria a aprender fue 4. Para el caso "focused" d = 3 se determinaron 25 neuronas ocultas y para d=2, 22 neuronas y para el "no focused", 19 y 21 para d=3 y d=2 respectivamente. La mejor red fue obtenida para el caso de memoria gamma "no focused" d=2, 21 neuronas ocultas con entradas V, V-2 y salidas V+1, y la llamamos **c)** en el CUADRO III.3.4.

También, análogamente que para las TDNNs intentamos mejorar la predicción en un paso incorporando la variable aumento y algunos datos de las semanas anteriores en forma explícita como se verá a continuación.

TLFNs con memorias gamma y entradas adicionales.

Teniendo en cuenta el MDL, la mejor red se obtuvo como aquella que tiene entradas S, A, V-1, T-1, S-2, V-2, T-2, salida V y 17 neuronas ocultas con memoria "focused", llamada red **d).**

Cuadro de resumen para las TLFN: **CUADRO III.3.4**

	a)	b)	c)	d)
MDL calculado sobre el conjunto de CV	-66.30	-373.29	48.25	213.13
% Error sobre el conjunto de CV	11.88%	13.03%	14.19%	11.49%
Arquitectura	d=3, 12 neuronas ocultas, memoria "no focused"	d=3, memoria focused, 8 neuronas ocultas	d=2, 21 neuronas ocultas, memoria "no focused"	d=3, 17 neuronas ocultas, memoria "focused"
Nro. de pasos de predicción con error < 10%	0	1	3	0
Error de predicción en un paso para semana 2	13.47%	2.85%	6.13%	18.38%
Error promedio en semana 2 a 18/2003	11.33%	23.77%	11.79%	31.96%
Error máximo	31.04%	40.41%	41.07%	53.91%
Error mínimo	2.28%	2.75%	0.56%	3.01%

a) TDNN pura "no focused", entradas solo V, d = 3
b) TDNN con d=3, memoria "focused", entradas S, A, T-1, 8 neuronas ocultas y constituye el modelo
c) TLFN, memoria tipo gamma de profundidad = 15, trayectoria a aprender = 4, "no focused" d=2, 21 neuronas ocultas con entradas V, V-2 salidas V+1
d) TLFN con memoria gamma, profundidad = 15, trayectoria a aprender = 4, entradas S, A, V-1, T-1, S-2, V-2, T-2, salida V, 17 neuronas ocultas con memoria "focused"

En resumen, la mejor red TLFN se obtuvo cuando se utilizó una TDNN con entradas adicionales y memoria "focused" (ver III.2.2.1.2).

[13] Gamma tipo 1, ver AIII.3.

Redes recurrentes

Las arquitecturas probadas de redes recurrentes fueron

a) Totalmente recurrentes:

- Entradas S, T, V-1 y salida V
- Entradas S, T, V-1, V-2 y salida V
- Entradas S, T, V-2 y salida V

b) Parcialmente recurrentes

Iguales entradas y salidas que para totalmente recurrentes

c) Redes de Jordan y Elman

Casos a) y b) (Redes total o parcialmente recurrentes):

Las funciones de transferencia que se usaron en todos los casos fueron la Tanh y 7 neuronas ocultas, en una sola capa. En una de las pruebas realizadas utilizamos como función de salida de la capa de salida la logística, que dado el rango de valores posibles de salida de la red (valores enteros positivos) parecía ser más adecuada y dar un menor error. Sin embargo, obtuvimos una peor aproximación que usando Tanh.

El entrenamiento realizado fue incremental, usando BPTT.

El conjunto de CV consistió del 15% de los datos de entrenamiento.

La predicción siempre fue en un paso.

Para ver qué tan bien se podía predecir con cada arquitectura entrenando con un número fijo de valores, se construyeron 12 juegos de entrenamiento, cada uno desfasado una semana respecto al otro, y luego se predijeron las semanas inmediatamente siguientes a la última de cada juego. Por ejemplo, ej1 corresponde al rango de semanas 18/1999 a la 19/2002 y predice la semana número 20 del año 2002, el ej2 corresponde a 19/1999 – 20/2002 y predice la 21, etc. Para cada juego de entrenamiento, se realizó el training de la red (con sus siete pruebas para elegir la que diera mejor resultado). Esto fue una rudimentaria prueba de validación del modelo.

Los mejores resultados de las redes totalmente y parcialmente recurrentes se obtuvieron mediante una red totalmente recurrente, con entradas S, T, V-1 salida V y 7 neuronas en una capa oculta en la cual (red a)):

MDL calculado sobre el conjunto de CV: 62.20
% Error sobre el conjunto de CV: 15.59%
7 ocultas, totalmente recurrente
Nro. De pasos de predicción con error < 10% =2
Error de predicción en un paso para semana 2= 1.66%
Error promedio en semana 2 a 18/2003 =20.22%
Error máx.: -- 50.66%
Error min.: -- 1.66%

Caso c) (Redes de Jordan y Elman):

Como caso particular de redes recurrentes se probó utilizar redes de Jordan y Elman. El simulador de redes provee varias topologías posibles para las redes de Jordan, dependiendo desde donde salen las entradas a las neuronas de contexto: primer capa oculta o segunda capa oculta (Ver [12]).

La red elegida tuvo la topología descripta en la figura III.17:

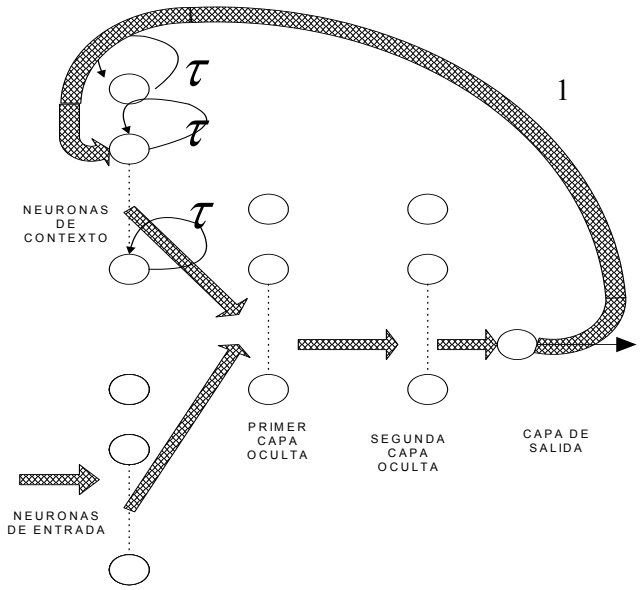


Figura III.17

Para simplificar el diagrama, las flechas anchas representan todas las conexiones posibles. Las entradas son V, S, S-1,V-1, T-1 y salida V. Porcentaje de datos de CV=15%, dos capas ocultas, cuatro neuronas en la primer capa oculta y dos en la segunda (lo que anotamos 4/2), todas las funciones de salida Tanh, entrenamiento "on line". Se probó con varios valores para la constante tiempo: 0,5, 0,3, 0,7 ,0,15, 0,07.

La constante tiempo tiene el siguiente significado: dada una neurona de contexto **Z,** se tiene

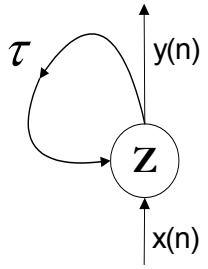


Figura III.18

donde se cumple que $y(n) = \sum_{i=0}^{n} x(i)\tau^{n-i}$. La constante tiempo es por definición τ y controla la caída exponencial con que los datos pasados (anteriores) influyen en la salida actual, es decir, controla la memoria de la neurona.

Para distintos valores de la constante tiempo (τ) se obtuvieron distintas redes, siendo la que arrojó mejores resultados una en la que Tiempo = 0.15, llamada red b), y en ella se cumplió que:

MDL calculado sobre el conjunto de CV: -15.63
% Error sobre el conjunto de CV: 16.35%
4/2 neuronas ocultas, topología ya descripta
Nro. de pasos de predicción con error < 10% = 5
Error de predicción en un paso para semana2/3003 = 1.06%
Error promedio al predecir las semanas 2 a 18 =13.37%
Error máx.: 44.11%

Error min.: 0.91%

Incorporación de los aumentos de precio de venta del gas al modelado

Una de las causas de mayor fluctuación en la serie temporal es la presencia de aumentos de precio de venta del gas. En las semanas previas a un aumento se suele registrar una mayor demanda debido a la tendencia especulativa y en las posteriores, una menor debido a la ligera retracción que provocó el aumento. Intentamos reflejar todo esto agregando una variable descriptiva, AUMENTO, que podía tomar varios valores:

-2 si la semana en cuestión era anterior en 2 semanas a aquella en la que se produce el aumento,
-1 ídem, pero una semana antes
0 si en esa semana se producía un aumento
1 si la semana es posterior a aquella en la que se produjo el aumento
2 si es posterior en dos semanas
3 en otro caso.

En caso de conflicto, se hizo primar el criterio de ser previa a ser posterior (por ejemplo, en el caso de una semana que fuese una semana posterior aun aumento y dos semanas anterior a otro, se le asignó –2. En el caso de las redes de Jordan, entrenamos una red con entradas S, A, A-1, V-1, T-1 y salida V. Utilizamos tiempo = 0.5 y en lo demás era igual a las redes de este tipo ya mencionadas. La entrenamos con datos hasta la semana 1/2003. Lo que se obtuvo, red **c)** del CUADRO III.3.5, fue:

MDL calculado sobre el conjunto de CV: -23.61
% Error sobre el conjunto de CV: 9.55%
4/2 neuronas ocultas, topología ya descripta
Nro. de pasos de predicción con error < 10% = 5
Error de predicción en un paso para semana 2/2003= 0.92%
Error promedio al predecir las semanas 2 a 18 =11.74%
Error máx.: 37.22%
Error min.: 0.71%

En resumen, la red de Jordan que arrojó mejores resultados fue una con entradas adicionales:

MDL calculado sobre el conjunto de CV: -23.61
% Error sobre el conjunto de CV: 9.55%
4/2 neuronas ocultas, topología ya descripta
Nro. de pasos de predicción con error < 10% = 5
Error de predicción en un paso para semana2/3003 = 0.92%
Error promedio al predecir las semanas 2 a 18 =11.74%
Error máx.: 37.22%
Error min.: 0.71%

Cuadro resumen para las redes recurrentes: **CUADRO III.3.5**

	a)	b)	c)
MDL calculado sobre el conjunto de CV	**62.20**	-15.63	-23.61
% Error sobre el conjunto de CV	**15.59%**	16.35%	9.55%
Arquitectura	**7 ocultas, totalmente recurrente**	4/2 neuronas ocultas, topología ya descripta	4/2 neuronas ocultas, topología ya descripta
Nro. de pasos de predicción con error < 10%	**2**	5	5
Error de predicción en un paso para semana 2	**1.66%**	1.06%	0.92%
Error promedio en semana 2 a 18/2003	**20.22%**	13.37%	11.74%
Error máximo	**50.66%**	44.11%	37.22%
Error mínimo	**1.66%**	0.91%	0.71%

a) red totalmente recurrente, entradas S, T, V-1 salida V y 7 neuronas en una capa oculta
b) red de Jordan de entradas V, S, S-1,V-1, T-1 y salida V, dos capas ocultas, cuatro neuronas en la primer capa oculta y dos en la segunda (lo que anotamos 4/2), todas las funciones de salida Tanh, constante tiempo: 0.15
c) Ídem **b)** pero con entradas S, A, A-1, V-1, T-1 y salida V, constante tiempo = 0.5

III.3.3 Validación del modelo

Como ya se esbozó en las pruebas con redes recurrentes, una forma de validar a corto plazo el modelo podría ser entrenar la red para un número ***n*** (fijo) de semanas consecutivas ("***cluster"***) y predecir el valor de la serie para la semana *n+1* siguiente a ese conjunto. Repitiendo el proceso varias veces, para distintos "clusters" de semanas, se podría tomar el porcentaje de error promedio de esas predicciones y ver así si el modelo, en cuanto a topología y algunos parámetros que lo definen, es válido (si su porcentaje de error promedio al predecir es menor que una cota predeterminada). Tomar esta validación como validación a corto plazo del modelo tiene dos inconvenientes:

- solo nos dice si la topología y algunos parámetros que definen la red (por ejemplo, la constante tiempo, el caso de las de Jordan) son adecuados al problema, ya que los pesos del modelo cambian en cada caso (al entrenar con cada conjunto de semanas)

- es sensible a la forma en que se seleccionen las semanas para formar los distintos conjuntos ("clusters") de entrenamiento

Por otra parte, si deseamos realizar una reconstrucción dinámica del sistema, agregaremos alguna restricción adicional para considerar a un modelo válido a corto plazo como válido en un sentido general: diremos entonces que un modelo construido para la predicción de la serie temporal es válido si se cumplen simultáneamente las siguientes condiciones [52] y [5]:

a) el modelo tiene un buen comportamiento a corto plazo: el modelo predice en una forma "suficientemente buena" el valor de la serie para al menos un paso en el futuro, en promedio, para el conjunto de CV.

b) El modelo captura la estructura y propiedades de la dinámica subyacente del sistema que genera la serie temporal, es decir, presenta un buen comportamiento a largo plazo.

Por lo tanto, los modelos válidos a corto plazo serán los que usemos para predecir, y de entre ellos, algunos para reconstruir el comportamiento del sistema. Adicionalmente, luego de determinados los modelos válidos a corto plazo, se elegirá entre ellos basándose en el MDL. La selección del modelo basándose solo en la estimación del MDL no es posible, ya que se pueden tener valores de dicha estimación muy bajos sin que ello signifique que se tiene ni siquiera una buena aproximación a corto plazo, puesto que un error alto de predicción puede ser compensado por la simplicidad del modelo, dando una estimación del MDL muy pequeña.

Validación a corto plazo

La validación a corto plazo es la que asociamos a las pruebas en modo "open loop". En esta modalidad, la red recibe como una entrada los valores verdaderos de la serie y el objetivo es predecir el valor de la misma un paso hacia delante.

La forma más simple de medir el éxito de la predicción es calcular el MSE promedio sobre un conjunto de N muestras (ejemplares) de datos, siendo N el número de ejemplares dejados para CV:

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

donde $d_i,\quad y_i$ representan el valor de la serie temporal en el paso siguiente y el valor predicho, respectivamente. Está claro que se desea un valor del MSE pequeño, pero se pretende además que el error de predicción sea pequeño respecto al valor predicho. Por otra parte, si se desea que la medida del error este acotada entre 0 y 1, se podría utilizar el root-mean-square (RMS):

$$\begin{cases} RMS = \dfrac{\sum_{i=1}^{N}(y_i - d_i)^2}{\sum_{i=1}^{N}(d_i - \overline{d})^2} \\ \overline{d} = \dfrac{1}{N'}\sum_{i=1}^{N'} d_i \end{cases}$$

siendo N' el número de valores deseados distintos.

Patel [52] sugiere utilizar el cociente (coeficiente) "signal-to-error" definido como

$$SER = 10\log_{10}\frac{MSS}{MSE}$$

siendo $MSS = \frac{1}{N}\sum_{i=1}^{N} d_i^2$. Lo que se buscaría entonces sería un valor alto de SER, que significaría que la serie predicha es significativa respecto al error de predicción [52].

Otra forma de medir la calidad de la predicción es considerar el porcentaje de error cometido al predecir en un paso, promediado sobre el conjunto de CV (anotado como %Error CV). Dado que los valores de la serie temporal en estudio son siempre mayores que uno, este porcentaje de error es una medida significativa del error absoluto que se esta cometiendo.

Adicionalmente, minimizar el %Error CV es equivalente a minimizar el MSE sobre el conjunto de CV, ya que si se calculan las derivadas parciales en cada caso y se igualan a 0, se obtienen los mismos puntos extremos (con la convención de que $signo(0)=0$):

$$\frac{\partial \%ErrorCV}{\partial y_i} = \frac{100}{N}\frac{1}{d_i} signo(d_i - y_i)$$

y

$$\frac{\partial MSE}{\partial y_i} = -\frac{2}{N}(d_i - y_i)$$

Utilizaremos esta métrica para decidir la validez a corto plazo de un modelo: ***un modelo será considerado válido si su porcentaje de error de predicción en un paso promediado sobre el conjunto de CV es menor que el 12%.*** La elección del 12% como límite tiene la siguiente explicación: originalmente la meta de este trabajo era lograr una cota del error promedio de predicción de un 10%. Sin embargo, tal vez debido a los escasos datos de los que disponemos, un error de esas características no fue alcanzable, por lo que tuvimos que relajar la condición a un 12%, que sí es cumplida por varios de los modelos desarrollados.

Finalmente, dado que el porcentaje de error es un indicador significativo del MSE, minimizarlo será equivalente a maximizar el SER.

Validación a largo plazo

Para saber si el modelo capturó toda la estructura y propiedades del sistema que genera la serie original, se realiza una prueba de "closed loop". En esta prueba, utilizada en la bibliografía en el modelado de sistemas determinísticos, la red opera dependiendo de sus propias salidas anteriores (en lugar de los valores reales anteriores de la serie) en cada paso de tiempo y una copia de las entradas que tuvo el sistema para los respectivos instantes en el pasado. En este esquema la evaluación no puede ser iniciada sin realizar previamente una etapa ("***priming phase***") en la que a partir de los datos conocidos (reales) se van prediciendo las entradas necesarias para los pasos posteriores, tomando en cada instante una predicción más como entrada, hasta que ya solo se predice en base a salidas de la red obteniéndose entonces la serie reconstruida de la original.

Las salidas de la red van a ir acumulando errores de manera que incluso un modelo óptimo generará trayectorias divergentes respecto a la serie original debido a la dependencia de las condiciones iniciales y de los ruidos incluidos en el vector con que se comienza la priming phase. Obsérvese que en el caso estocástico se suma la posible divergencia producida por los ruidos dinámicos. Para comparar cuantitativamente la estructura dinámica de las series, cuando la original corresponde a un sistema determinístico, calcularíamos los siguientes valores (***invariantes***) para la serie original y la reconstruida [52]:

- la dimensión m del "embedding"
- el espectro de Lyapunov completo, $\lambda_i, \quad i = 1...m$
- la dimensión de correlación ***D***
- la dimensión de Kaplan-Yorke, que se espera que sea cercana a ***D.***
- La entropía de Kolmogorov, que es calculada sumando los exponentes de Lyapunov positivos
- El horizonte de predecibilidad.

Un modelo satisfactorio, en el caso determinístico, sería aquel tal que la serie reconstruida por él tenga invariantes muy similares a los de la serie original. Como ya dijimos, en nuestro caso estocástico lo que correspondería sería comparar las densidades de probabilidad empíricas de algunos de los invariantes con las respectivas densidades del sistema original. En su lugar, debido a lo corta que es la serie del caso de estudio, nos limitaremos a elegir aquellos modelos que tengan menor error promedio al iterar las predicciones durante las semanas 2-18/2003.

Determinación de los modelos válidos

Modelos válidos para la reconstrucción dinámica

Teniendo en cuenta la condición a) de III.3.3 solo son válidos:

a) red totalmente recurrente con entradas S, T, V-1 y salida V
b) red totalmente recurrente con entradas S, T, V-1, V-2 y salida V
c) red de Jordan con entradas adicionales: ver III.3.2
d) red TDNN "pura" "no focused" con 12 neuronas ocultas y correspondiente a d=3
e) red TDNN "no focused" con entradas adicionales (para d=3), ver III.2.2
f) red TLFN con memoria gamma, entradas adicionales y "focused", d=3
g) un MLP de entradas T, A-1, V-1, V-2, S-3 y salida V con siete neuronas ocultas en una sola capa

Para verificar la condición b) predecimos en forma de "closed loop" para las semanas 2-18 para los siguientes modelos y obtuvimos sus errores de predicción iterada. Para realizar la predicción el "closed loop" en los casos en que se tomaban entradas adicionales (aumentos, temperaturas, etc.) lo que se hizo fue repetir esas entradas de los datos originales 2000-2001 (o sea, para la semana 52 de esta predicción, se tomó una temperatura media correspondiente a la semana 52 del año 2000).

Tomamos los valores reconstruidos para las semanas 2-18/2003 solamente ya que eran todos los datos disponibles, y no significa que no se pueda reconstruir un período más largo con esos modelos.

En la tabla siguiente se muestran los resultados obtenidos.

	REAL	TDNN no focused d=3	TDNN c/ aumento no focused d=3	Gamma con aumento d=3	Jordan c/ aumento	Tot recurrente sin v-2	Tot recurrente c/ v-2	MLP con aumento
Delay opt:	3	1	1	2	2	1	1	1
Dim. del embedding:	2	3	4	3	6	2	7	2
Espectro de Lyapunov:	-1.15e-1, -3.88e-1	-4.04 e-2, −5.00e-1, −1.06e00	-1.28e-1, -3.01e-1, -5.13e-1,.....	2.53e-2, -3.11e-1, -5.34e-1	-7.28e-3, -3.57e-2, -3.92e-2.....	-2.88E-1, -2.04e00	6.75e-2, -9.63e-2.....	No calculable por el método de Sano y Sawada (TISEAN)
%FNN si m=2	0%	12%	33.35%	16.7%	15%	0%	16.6%	0%
Modelo del CUADRO III.3.1		d)	e)	f)	c)	a)	b)	g)
%Error iterado promedio		33.38%	28.33%	16.61%	14.55%	14.92%	23.25%	73.33%

Todos los AMI fueron calculados con "delay" máximo=99 y detalle máximo de la gráfica de información mutua, utilizando el VRA.

Según la condición b) de III.3.3 (interpretándola como semejanza de invariantes) son válidos los modelos f), a), g), c) y d) ya que las series generadas por estos modelos tienen aproximadamente la misma dimensión de "embedding" y "delay" óptimo que la serie original. En el caso de la de Jordan, el porcentaje de vecinos falsos cuando m=2 no es tan alto como para descartarla.

Es de notar que, salvo en los casos de los modelos d) y g), las otras redes son válidas a largo plazo tanto en el sentido de tener un buen error promedio como de tener invariantes similares. La red con memoria gamma produce una serie caótica, lo cual a hace de muy dudosa aceptación. Si descartamos la red de Jordan y la de memoria gamma, siendo estrictos, decimos que el único modelo válido a largo plazo es una red totalmente recurrente de entradas S, T, V-1 y que predice V, es decir el modelo a). Relajaremos esta condición y daremos prioridad al criterio de menores errores de predicción promediados sobre el conjunto de "testing", por lo que los modelos g) y d) quedan descartados.

Los modelos válidos para realizar la reconstrucción dinámica del sistema son el a), el c) y el f)

Teniendo en cuenta el número de pasos que se pudo predecir con un error aceptable, vemos que se deberá re-entrenar como mínimo cada 4 o 5 semanas si se elige la red c) y cada semana para la a) y la f); esta cantidad de semanas deberá ser ajustada empíricamente a medida que se hacen las predicciones.

Observación: El descartar una red en base a la condición a) de II.3.3 podría ser ignorado si luego se utiliza esa red en el "ensemble method".

Modelos válidos para la predicción

En el caso de la predicción, dado que no se pretende hacer una predicción recursiva de las ventas y que el período entre utilizaciones (realización de predicciones) del modelo es de una semana, puede ser útil considerar los modelos que dan una buena aproximación local (es decir, que no necesariamente sean válidos a largo plazo), o sea, que proporcionen una aproximación aceptable (con error menor al 12% sobre el conjunto de CV) en un paso, a partir de un cierto conjunto de datos históricos, por ejemplo 190 semanas en el pasado, re-entrenando la red cada vez que se desea predecir (si se debiera predecir cada, por ejemplo, tres segundos, se debería utilizar un modelo válido en el sentido de que predijera acertadamente para muchos pasos en el futuro ya que no se podría re-entrenar entre cada predicción.). Entonces solo quedaría elegir uno de los modelos especificados en el CUADRO III.3.1. Esa elección la realizamos usando el MDL: el que tuvo menor MDL fue una red TDNN tal como se describe en III.3.2 "Las TDNNs". La red así elegida fue la que arrojó mejor resultado en el caso del aumento de la semana 7/2003. En el caso de las de Jordan, una con T=0.15, sin aumento, que si bien tuvo un MDL relativamente bajo y un escaso error de predicción en un paso sobre el conjunto de "testing", produjo un error muy alto para la predicción de la semana 7/2003: 44% y 38% para la semana 8/2003.

Dado que todas las pruebas fueron hechas en base a la predicción de solo un paso en el futuro, la red utilizada para predecir debería ser reentrenada cada semana (ver [5]).

Si se quisiera aplicar el ensemble method, se debería utilizar con dos o tres redes, ya que de otra forma el tiempo de entrenamiento se empieza a disparar y el proceso se volvería engorroso.

Si elegimos las redes a promediar utilizando el MDL, las elegidas serían la TDNN pura "no focused" correspondiente a d =3, la de Jordan con aumento y el MLP con aumento. Si se elige según el porcentaje de error de predicción promedio en un paso, serían la red de Jordan, la TDNN pura "no focused" correspondiente a d=3 y la totalmente recurrente sin V-2. Por lo tanto, el promedio de las salidas de la de Jordan con entradas adicionales y de la TDNN pura no focused con d=3 podría ser un buen estimador en un paso.

Las estimaciones que se obtendrían entonces equiponderando, para las semanas 2-18/2003 serían:

		Jordan		TDNN		Ensemble	
Semana	***Deseado***	***Predicción***	***%Error***	***Predicción***	***%Error***	***Predicción***	***%Error***
2	2344265	2365895.5	0.92%	2709111	15.56%	2537503.25	7.62%
3	2554344	2351020.25	7.95%	2802681.5	9.72%	2576850.875	0.87%
4	2568176	2383354.25	7.19%	2809267	9.39%	2596310.625	1.08%
5	2384806	2316061.25	2.88%	2726262	14.32%	2521161.625	5.41%
6	2355510	2338734.25	0.71%	2713826.75	15.21%	2526280.5	6.76%
7	4240721	2691635.25	36.52%	4117191.75	2.91%	3404413.5	24.57%
8	1716938	2356060	37.22%	2489850	45.02%	2422955	29.14%
9	1821053.875	2351360.25	29.12%	2520970	38.43%	2436165.125	25.25%
10	2572587	2351720.5	8.58%	2811378.75	9.28%	2581549.625	0.35%
11	2749742	2560280	6.59%	2900987.25	5.50%	2730633.625	0.70%
12	2518900	2424982.75	3.72%	2786055	10.61%	2605518.875	3.32%
13	2072692.875	2409411.25	16.24%	2604294.5	25.65%	2506852.875	17.32%
14	2939268	3196924.5	8.76%	3007983.75	2.34%	3102454.125	5.26%
15	2868848	3149047	9.76%	2966812.75	3.41%	3057929.875	6.18%
16	2455639	2660958.5	8.36%	2757246.5	12.28%	2709102.5	9.36%
17	2484590	2839490	14.28%	2770294.5	11.50%	2804892.25	11.42%
18	3822776	3790146.75	0.85%	3687567.75	3.54%	3738857.25	2.24%
- Errores del ensemble -							
Max= 29.14%	***Prom.=9.23%***	***Min= 0.35%***					

III.3.4 El "ensemble method"

Esta estimación de las ventas semanales es construida utilizando un promedio de las dadas por los otros modelos (Ver II.2). Cada uno de esos modelos fue entrenado utilizando CV (15% de los patrones de entrenamiento eran de CV), es decir, no utilizamos la propiedad de "suavizado" del "ensemble process". El criterio de error empleado fue el mismo en todas las redes. Perrone [55] sostiene que el entrenamiento podría ser hecho con distintos juegos de datos para las distintas topologías, afín de asegurar una mayor independencia entre los estimadores. Debido a la escasez de datos que sufrimos, no pudimos hacer esto y debimos entrenar todos los modelos con los mismos datos, lo que podría aparejar una reducción no tan importante en el error de la predicción dada por el método. Realizamos dos trabajos utilizando este método:

a) predijimos la serie para las semanas 2/2003 a 18/2003 (modo "open-loop"). Ver más adelante.

b) estudiamos cómo se comportaban los valores generados por el "ensemble" para la predicción en modo "closed loop". En este caso lo que hicimos fue promediar los valores obtenidos a partir de distintos modelos y ver si el promedio cumplía los requerimientos de validación a largo plazo. En el CUADRO III.3.6 se tienen varias variantes del ensemble (calculando los promedios aritméticos y los ponderados según el MDL y %Error CV de cada modelo). Básicamente nos planteamos si

- el ensemble obtenido a partir de los modelos válidos a corto plazo es válido a largo plazo
- el ensemble de los válidos a largo plazo es un "mejor modelo" a largo plazo
- el ensemble de dos modelos, uno válido a corto plazo y el otro no, ambos válidos a largo plazo, es válido a largo plazo

El "ensemble method" aplicado a la predicción

Ponderamos las predicciones de cuatro formas distintas: según el porcentaje de error en la predicción del primer paso (para semana 2/2003), según el MDL sobre el conjunto de CV, según el porcentaje de error sobre el conjunto de CV y equiponderación. Las predicciones de las que partimos fueron:

Predicciones en modo "open loop":

Se promediaron los valores de predicción de las siguientes redes:

REAL	Recurr sin v-2	TDNN pura no focused d=3	TDNN c/aumento no focused d=3	Gamma con aumento d=3	Jordan c/aumento	Tot recurr con v-2	MLP con aumento

Y el "ensemble" dio: **CUADRO III.3.6**

ENSEMBLE equiponderado	%error	ENSEMBLE (MDL)	%error	ENSEMBLE (%error CV)	%error	ENSEMBLE (%error 1 paso)	%error
	Max: 60.73%		Max: 48.18%		Max: 48.08%		Max: 47.77%
	Min: 1.96%		Min: 1.29%		Min: 0.41%		Min: 0.48%
	Prom: 18.36%		Prom: 15.39%		Prom: 13.98%		Prom: 14.82%
MSEs:							
2.98853E+11		2.3569E+11		2.3862E+11		2.48069E+11	
MSE promedio:	2.553E+11						

Observamos que al utilizar la siguiente forma de asignar pesos a las ponderaciones según *valor* (donde *valor* puede ser el MDL, %Error CV, etc.):

$$[\mathrm{MAX}(valor)-valor]/\mathrm{MAX}(valor)$$

algunos valores de la lista tienen ponderación 0 (los correspondientes al máximo de *valor*). Si cambiamos ligeramente la fórmula por

$$[\mathrm{MAX}(valor)-valor+1]/\mathrm{MAX}(valor)$$

eso ya no sucede. Los resultados obtenidos entonces son:

ENSEMBLE equiponderado	%error	ENSEMBLE (MDL)	%error	ENSEMBLE (%error CV)	%error	ENSEMBLE (%error 1 paso)	%error
	Max: 60.73%		Max: 48.23%		Max: 56.26%		Max: 59.68%
	Min: 1.96%		Min: 1.30%		Min: 1.04%		Min: 1.46%
	Prom: 18.36%		Prom: 15.41%		Prom: 16.76%		Prom: 18.00%
2.98853E+11		2.35929E+11		2.7126E+11		2.93166E+11	
MSE promedio:	2.7E+11						

De todas formas, esta segunda ponderación introduce un sesgo en el cálculo en el sentido de que se adiciona un término que no depende del valor ponderado:

$$[\mathrm{MAX}(valor)\text{-}valor+1]/\mathrm{MAX}(valor) = [\mathrm{MAX}(valor)\text{-}valor]/\mathrm{MAX}(valor) + 1/\mathrm{MAX}(valor)$$

Otro método de ponderación posible podría haber sido tomar cada coeficiente de ponderación como:

$$\frac{1/valor}{\sum 1/valor}$$

Al utilizar los coeficientes óptimos sugeridos en II.2.3, el "ensemble" que se obtiene es el siguiente:

Ensemble óptimo	Deseado	% Error	Squared Error
2457931,276	2344265	4,85%	1,292E+10
2512409,229	2554344	1,64%	1758524998
2562629,304	2568176	0,22%	30765833,7
2251665,499	2384806	5,58%	1,7726E+10
2356712,586	2355510	0,05%	1446212,49
4319551,107	4240721	1,86%	6214185809
1974424,165	1716938	15,00%	6,6299E+10
2079025,979	1821054	14,17%	6,655E+10
2359050,21	2572587	8,30%	4,5598E+10
2721360,883	2749742	1,03%	805487802
2470784,199	2518900	1,91%	2315130320
2272412,116	2072693	9,64%	3,9888E+10
3013488,943	2939268	2,53%	5508748380
2879069,735	2868848	0,36%	104483856
2407578,571	2455639	1,96%	2309804836
2368009,275	2484590	4,69%	1,3591E+10
3841471,572	3822776	0,49%	349524394
		Prom: 4,37%	MSE: 1,66E+10
		Max: 15,00%	
		Min: 0,05%	

El cual verificamos que es mejor que todos los anteriores, en cuanto al error promedio, al máximo y al mínimo.

El "ensemble method" aplicado a la reconstrucción dinámica

Para este punto tomamos las predicciones en forma "closed loop" para las siguientes 100 semanas a partir de la 2/2003 dadas por los modelos que se consideraban válidos en la predicción a corto plazo. Al hacer el promedio de los valores dados por las siete redes obtuvimos una serie que se ve en la figura III.19. A dicha serie se le estudiaron algunos de sus invariantes y se obtuvo que:

- tiene "delay" óptimo 1
- tiene una dimensión del "embedding" = 2
- tiene coeficientes de Lyapunov -1.8 y -1.04 (es decir, negativos)

Es decir, la serie no es muy diferente (en cuanto a las características anteriores) de la original.

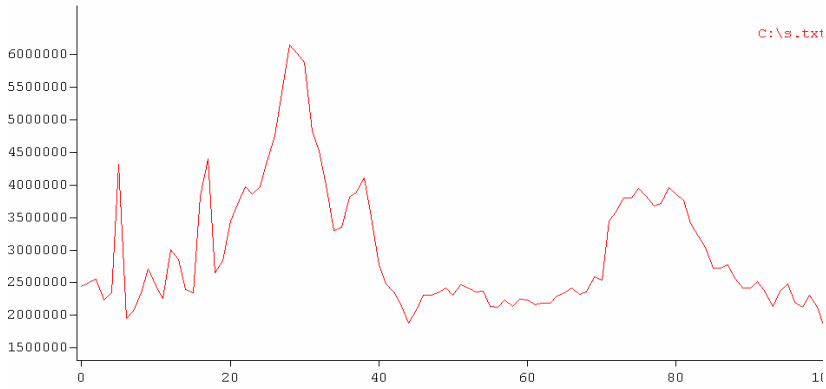


Figura III.19

Por otra parte, si promediamos solo los modelos 1), 2), 3), 4) y 5) del CUADRO III.3.3 :

- Si lo hacemos ponderando según los coeficientes obtenidos a partir de los errores de predicción iterada, obtenemos un "delay" =1, dimensión = 2, espectro de Lyapunov de coeficientes negativos, un error promedio del 17.58% y un máximo del 52.24%.

- Si en cambio tomamos los coeficientes originales (CUADRO III.3.2), el error promedio baja al 12,8%, el máximo al 48.17%, los invariantes que se obtienen son "delay" =1, dimensión = 2 y exponentes de Lyapunov negativos.

De todos modos, el error obtenido al promediar los valores de las cinco redes no es mucho mejor que el error de la red de Jordan (14.55%), por lo que, teniendo en cuenta la usabilidad del modelo, preferimos emplear una red de Jordan que entrenar a (y promediar los valores predichos por) cinco redes.

A efectos comparativos, graficamos en las figuras III.20 y III.21 algunas de las series reconstruidas versus la original (llamada "REAL") más un ruido uniforme θ en [-50000, +50000]. Graficamos además las distintas tendencias de las series generadas en la figura III.22.

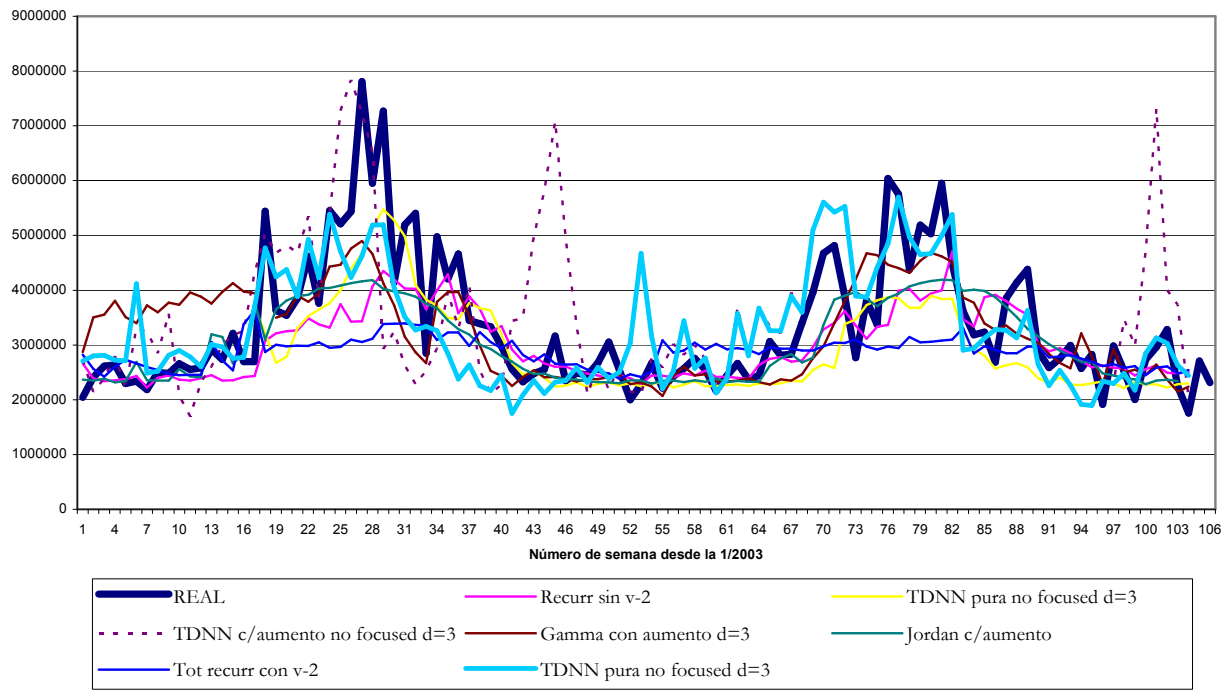


Figura III.20

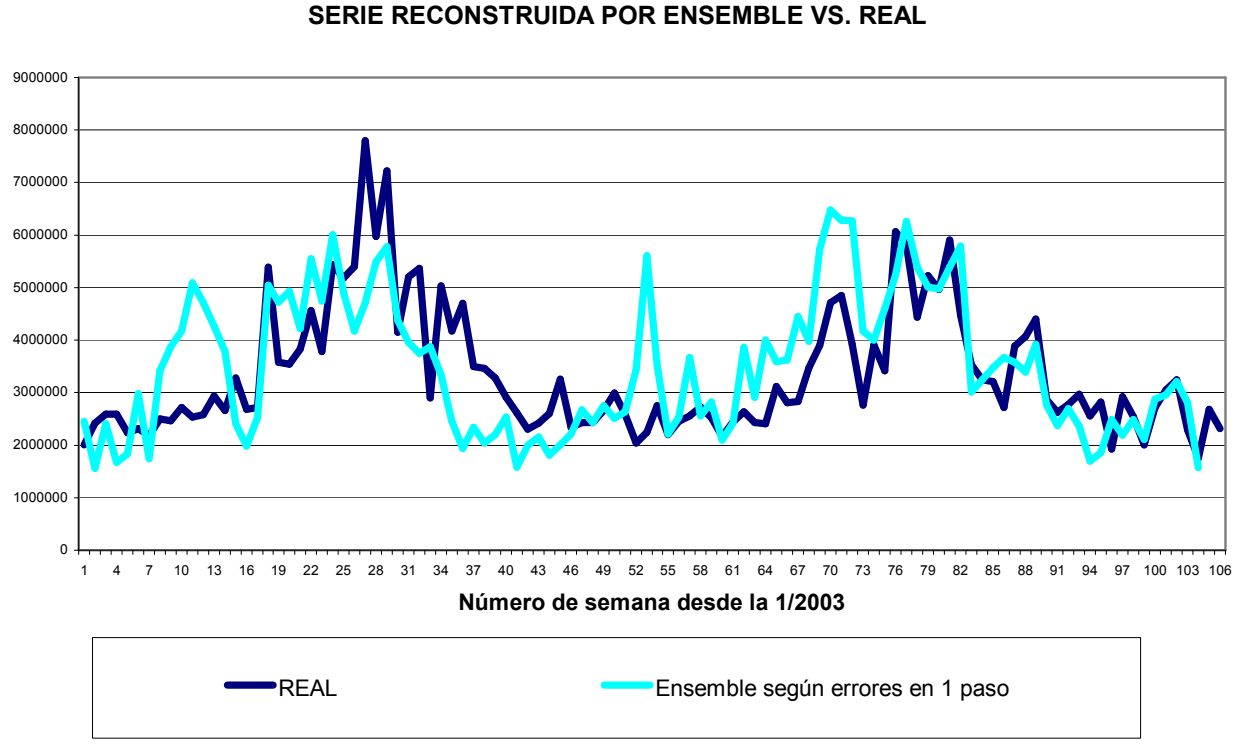


Figura III.21

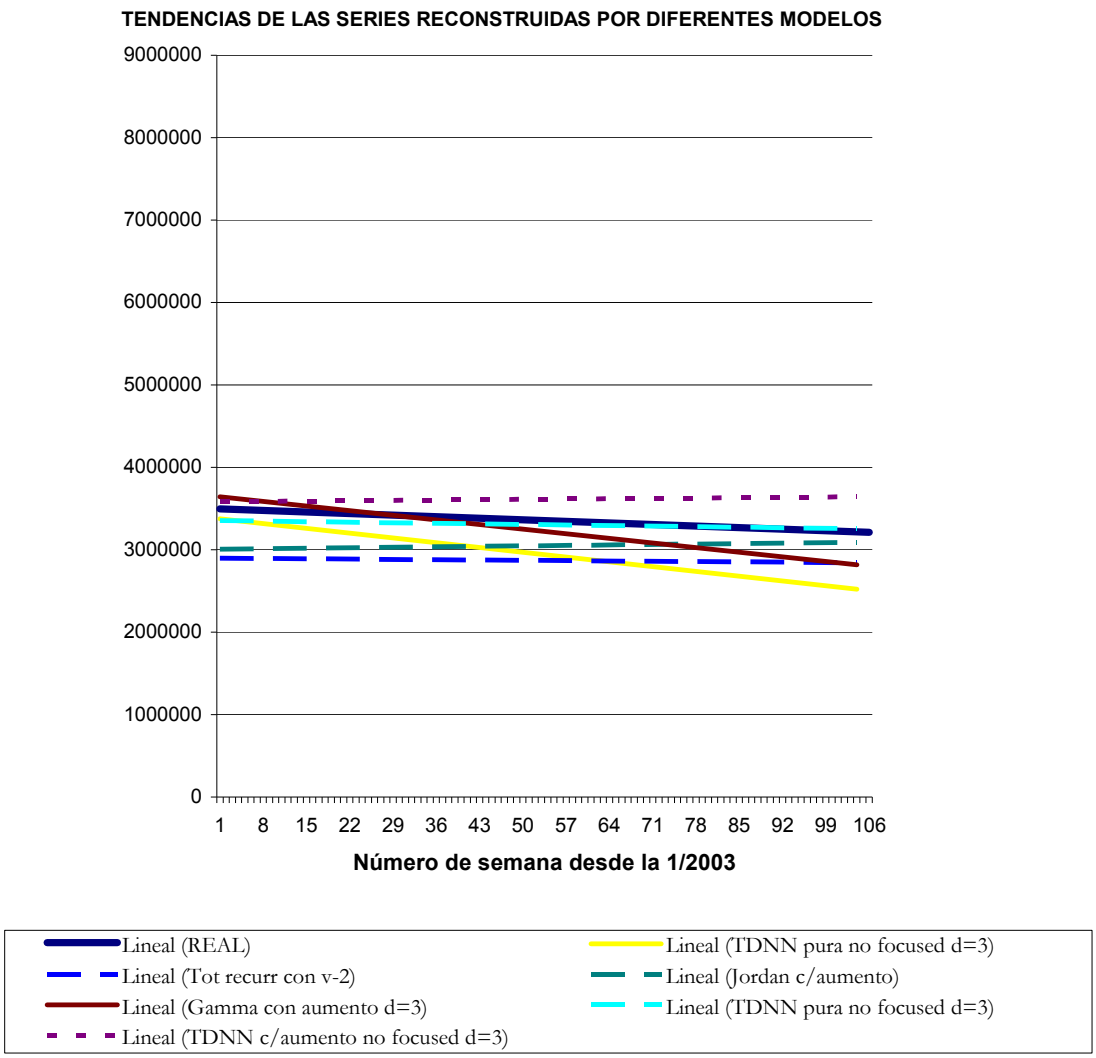


Figura III.22

Observamos que si bien tomamos la red de Jordan, la totalmente recurrente sin V-2 y la TDNN con memoria gamma como válidas a largo plazo, solo la gamma (y la TDNN pura "no focused") producen una serie que tiene la misma tendencia que la serie original. Ello tal vez se deba a que se puede considerar la tendencia como un caso de dependencia a largo plazo, por lo cual las dos redes recurrentes no logran aprender ese aspecto de los datos a causa del problema del gradiente evanescente.

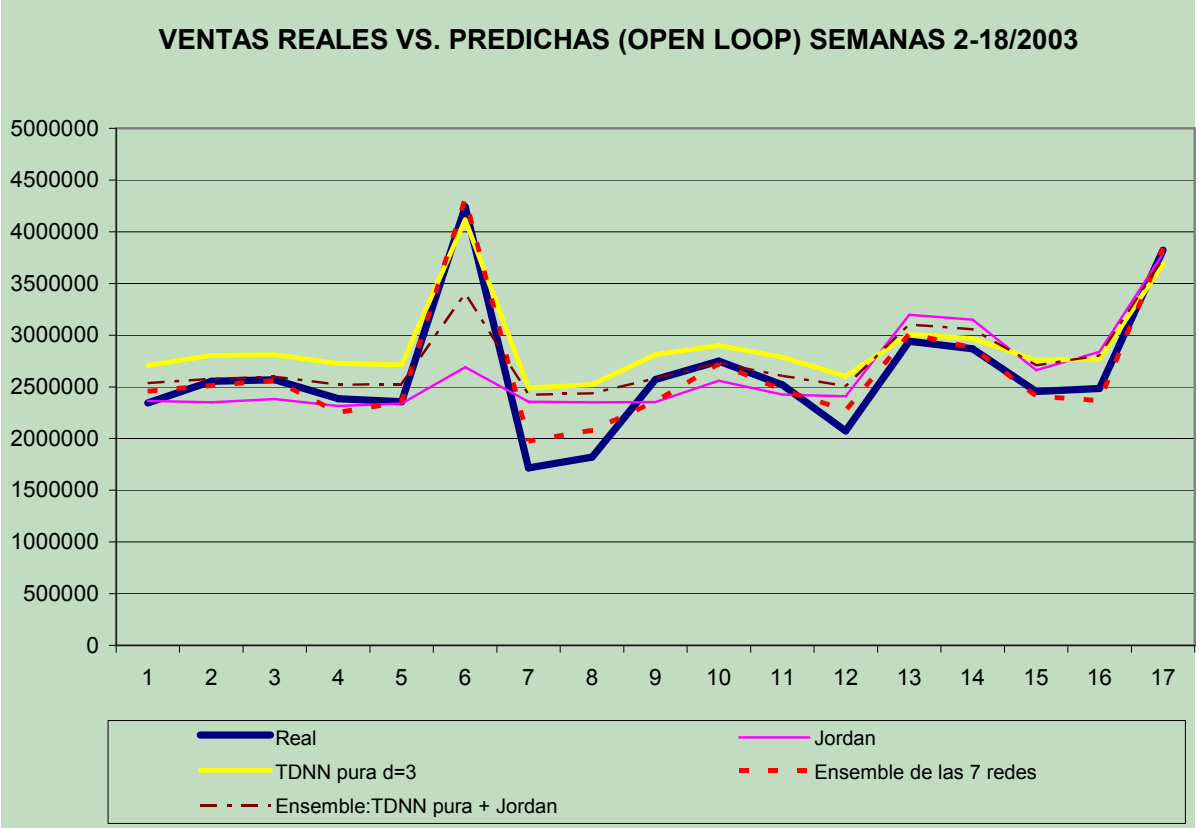


Figura III.23

Estamos en condiciones de responder las interrogantes planteadas en III.3.4:

a) el "ensemble" de los modelos válidos a corto plazo es válido a corto plazo, ya que, como vimos, mejora mucho la calidad de la predicción.

b) el "ensemble" de los modelos válidos a corto plazo utilizando los coeficientes óptimos dados en II.2.3 NO es válido a largo plazo, como se ve en el CUADRO III.3.3 (error iterado promedio = 40.34%).

c) el "ensemble" de los modelos válidos a largo plazo es también válido a largo plazo: si bien la mejora del error no es substancial, el "ensemble" es válido (en el sentido de menor error iterado) y tiene un error menor que el menor de los errores.

d) el "ensemble" de modelos válidos a largo y corto plazo da un valor intermedio a los promediados, o sea que puede considerarse no válido a largo plazo.

Sección IV: Observaciones, conclusiones y trabajo futuro

IV.1 Observaciones y conclusiones sobre los experimentos

Algunas de las observaciones sobre los experimentos y las conclusiones extraídas son:

1) Antes que nada, al haber hallado un modelo válido (cualquiera de los aptos para la predicción) que permite predecir los valores de la serie, probamos la existencia de la solución al problema del aprendizaje de la relación Entradas→Salidas planteado en III.2.3 "Entropías y resolubilidad del problema".

2) Dado que encontramos que la realización estudiada del proceso estocástico que genera las ventas no es caótica (ya que determinamos que los exponentes de Lyapunov de dicha realización son negativos, III.2.3), una mayor cantidad de datos recopilados nos permitirá hacer predicciones más exactas de la media de la distribución de probabilidad condicional de que la serie tome cierto valor dado que antes tomó ciertos otros, aunque la probabilidad de que el error de predicción sea grande siempre es no nula, debido a la naturaleza estocástica del sistema [5].

3) Si bien se podría haber entrenado y predicho usando los promedios obtenidos a partir de solamente las temperaturas de los días en que se vendió gas, en lugar de la promedio semanal, nos pareció que la elegida era la opción que reflejaba más la realidad, como se pasa a explicar. En general, hay un día extra en la suma utilizada para calcular las temperaturas promedio semanales (el domingo), que es tomado como un ruido en la medición de la temperatura (2.8% promedio), y si no incluyésemos la temperatura del domingo en el promedio semanal, estaríamos ignorando las ventas que se hacen al por menor (de los distribuidores a los consumidores finales) los domingos (especialmente los de baja temperatura), que luego se traducen en ventas al por mayor (de ANCAP a los mayoristas) el lunes siguiente.

4) En el APENDICE Sec. 1 observamos que casi todos los modelos experimentaron un error de predicción alto en la semana 7/2003. Ello se puede deber a varias causas:

- es una semana en la que se produce un aumento de precio de venta del gas

- solo se disponen de 11 ternas (S, T, A) en el conjunto de "training" para las cuales A = 0, es decir, hay solo once casos de aumentos para usar en el entrenamiento. De esos once, en seis casos se vendió más en la semana en que se produjo el aumento que en la semana anterior, en los otros cinco, se vendió menos. O sea que no solo son pocos los ejemplares de "training" con esas características, sino que además su información es poco significativa: casi no hay diferencia entre el número de semanas con aumento en las que se vende más (6) con respecto a aquellas en las que se vende menos (5). De todas maneras, algunos modelos captaron mejor que los otros ese efecto (por ejemplo, las redes TDNN con d = 3). También puede deberse a que la variable aumento haya sido codificada en forma poco adecuada.

- un dato a conocer, adicional a la fecha del aumento, es saber si el mismo fue anunciado en la prensa. De esa forma, cuando los aumentos son anunciados, se puede producir un efecto de especulación que no existe en el caso de aumentos no anunciados. No se pudo determinar cuales aumentos habían sido de cada tipo, ya sea a través de los registros de la empresa ni a través de la prensa que aparece en Internet, ya que ningún diario tenía más de un año en línea.

5) Una explicación de porqué la estimación del MDL es a veces negativa puede ser que el MSE en base al cual se calcula (III.3.2) sea el determinado a partir de las entradas normalizadas a la red (III.3.1) y no a partir de los datos en su magnitud real (del $\boldsymbol{O}(10^6)$). De esa forma, el MSE se calcularía en base a números de $\boldsymbol{O}(10^{-1})$ y la estimación del MDL resulta negativa.

6) En cuanto a los valores obtenidos con el método del "ensemble":

- promediamos siete modelos utilizando la ponderación óptima dada en II.2.3. Obtuvimos una mejora de un orden de magnitud respecto al menor MSE (el de la TDNN pura "no focused" d= 3 con 1.24E11, ver APENDICE) y de más de 30 veces respecto al MSE promedio (5.34E11). Esto tal vez se deba a los distintos modelos usados y a que los conjuntos de CV son tomados al azar al entrenar cada uno de ellos. Asimismo, el error (promediado sobre el conjunto de CV) de predecir en un paso se redujo en alrededor de 4 veces. Sin embargo, tal como se vio en III.3.2. "Válidos a largo plazo" , el "ensemble" aplicado al "closed loop" dio solamente una predicción intermedia al promediar las siete redes con coeficientes derivados de los errores de "closed loop". Cuando se utilizan los modelos con mejor comportamiento a largo plazo, el "ensemble" mejora drásticamente su error, ya sea que se prediga según los unos u otros coeficientes (12.8% y 17.58% de error promedio). Conjeturamos que el "ensemble method" pierde la propiedad de mejorar las predicciones al promediar un conjunto arbitrario de modelos cuando se trata de predicciones iteradas.

- Tal como se esperaba, los mejores resultados en el corto plazo, y también en el largo plazo, se obtuvieron al ponderar los valores dados por las redes individualmente según los coeficientes óptimos dados en II.2.3 siendo los errores considerados los obtenidos en la predicción en un paso ("open loop").

- Es significativo que como segundo valor óptimo de MSE (luego de el obtenido al usar los coeficientes dados en II.2.3) fue el correspondiente al promedio ponderado según el MDL (ver III.3.4). Eso significa que en la estimación que se hace del MDL para un conjunto de 190 ejemplares, 15% de los cuales es de CV, usando la fórmula

$$MDL = NLog(MSE) + \frac{1}{2}kLog(N)$$

siendo

k	número de pesos
N	número de ejemplares de CV
MSE	error medio cuadrático sobre el conjunto de CV

esta teniendo más peso el primer término $N\,Log(MSE)$. Esto puede deberse en parte a que: a) en el conjunto de redes hay topologías muy similares, que por lo tanto tienen sus k muy próximos o b) topologías muy dispares se implementan con un número de pesos parecidos.

Por lo tanto, dado que lo que se esta teniendo en cuenta es la cantidad de pesos y el MSE, esta estimación del MDL no refleja ciertas complejidades del modelo: por ejemplo, intuitivamente parece más compleja una red con memoria gamma que un MLP, aunque ambos se terminen implementando con la misma cantidad de pesos.

7) Observamos que el modelo de redes recurrentes no presentó los problemas típicos del gradiente evanescente en la predicción en un paso. Eso se debe a que las ventas de una semana no influyen más allá de tres semanas en el futuro; sin embargo, vimos que la tendencia (que podía ser considerada como una dependencia a largo plazo) no era aprendida ni por la red totalmente recurrente ni por la de Jordan (que también es recurrente). De la misma forma, si se tratase de predecir ventas diarias seguramente hubiesen aparecido problemas asociados al gradiente evanescente ya en la predicción.

8) Sería interesante investigar cómo se comportan las demás arquitecturas (ya que solo se probó con un caso de MLP) al usar la serie de datos "detrended".

9) En esta etapa del trabajo se utilizaron los RP interpretándolos visualmente. Una opción, para una etapa posterior, podría ser la de utilizar las medidas cuantitativas dadas por el RQA (Ver AVII.5).

10) Si bien la TLFN con memoria gamma tiene un error iterado promedio ligeramente (1.6%) mayor que el de una totalmente recurrente o una de Jordan, es capaz de captar la tendencia de la serie y luego reproducirla al trabajar en modo "closed loop" (ver III.3.3 "El ensemble method aplicado a la reconstrucción dinámica"). Al predecir, además, proporcionan resultados aceptables, lo que verifica lo ya adelantado en II.4.3 al hablar del gradiente evanescente.

11) Para resumir los resultados sobre los modelos obtenidos:

- En cuanto a los criterios posibles de evaluación:
  - Consideramos dos criterios de evaluación a corto plazo
    CC1) según el MDL estimado del modelo. Este criterio probó no ser suficiente si se aplicaba solo.
    CC2) según el porcentaje de error al predecir en un paso promediado sobre el conjunto de CV

  - Análogamente, existían varios criterios de evaluación a largo plazo:

    CL1) según el número de predicciones consecutivas a partir de la semana 2/2003 con un error menor que el 12%
    CL2) según el promedio de error al predecir en forma iterada los valores de la serie correspondientes a las series 2-18/2003 y la semejanza de invariantes entre la serie original y la reconstruida. Esta semejanza no puede constituir un criterio de selección en sí debido a la naturaleza estocástica del sistema.

- En cuanto a las topologías: se consideraron

  T1) redes en forma individual
  T2) «ensembles» de redes

La tabla siguiente muestra los resultado obtenidos:

Mejor topología según	**CC1**	**CC2**	**CL1**	**CL2**
T1	D	C	C	C
T2	No aplica	A	A	B

Siendo :

A: "Ensemble" de las siete redes descriptas en III.3.2 "Modelos válidos" según los coeficientes derivados a partir de la matriz de correlación de errores (ver II.2.3), que dio un porcentaje de error promedio de 4.37% al predecir las semanas 2-18/2003 y 6 pasos de predicción válidos. Para simplificar el manejo del modelo, se propuso utilizar el promedio equiponderado de los valores de una red de Jordan y una TDNN (ver III.3.3 "Modelos válidos para la predicción"), lo que produjo un error del 9.23% al predecir las semanas 2-18/2003 con 5 pasos de predicción válidos.

B: "Ensemble" de cinco redes, tal como se explica en III.3.4 "El 'ensemble method' aplicado a la reconstrucción dinámica", ponderadas según los coeficientes obtenidos a partir de los errores de predicción en un paso (en modo "open loop")

C: Red de Jordan con dos capas ocultas de 4 y 2 neuronas, entradas S, A, A-1, V-1, T-1 y salida V, que dio un error promedio del 11,11% al predecir las semanas 2-18/2003.

D: TDNN correspondiente a un delay óptimo $d = 3$, memoria focused, entradas V-1, S, A, T-1 y salidas V+1, 8 neuronas ocultas en una capa, MDL = -373.29.

Teniendo en cuenta la usabilidad del modelo, preferimos utilizar una red de Jordan para hacer la reconstrucción dinámica y el ensemble de dos redes (modelo **A**) para la predicción.

El re-entrenamiento se haría cada 4 (o menos) semanas, dependiendo de la calidad de los resultados obtenidos.

12) Se intentó utilizar, en algunas redes recurrentes, un "weight decay" tal como el descrito en AVI.2.1, provisto por el simulador de redes neuronales, pero los resultados no fueron esclarecedores: luego de entrenar, la matriz de pesos no ofrecía evidencias claras de cuales neuronas podían ser eliminadas. Téngase presente además que por la forma en que se construyeron las redes no se manejaron las conexiones en forma individual (de neurona a neurona) sino más bien de capa a capa, estando las capas totalmente interconectadas entre sí. Es decir, que aparte de que el "weight decay", para los

parámetros λ probados (ver AVI.2.1), no efectuó cambios muy claros en la matriz de pesos, si los hubiese efectuado, hubiese implicado construir la red a partir de sus elementos más elementales (neuronas manejadas individualmente, arcos conectores, elementos de control de "back-propagation" manejados manualmente, etc. con el consiguiente trabajo extra de diseño y el necesario manejo experto de la herramienta Dado que dicho manejo estaba fuera del alcance de este trabajo, y dado que disponíamos de otras herramientas que permitieran optimizar la estructura de la red (por ejemplo, los algoritmos genéticos), no se continuó probando con el "weight decay".

13) Un método que se podría haber ensayado fue el de Judd y otros, el "ΨΦ method", para mejorar la predicción a largo plazo. Una de las causas por la que no se lo probó, fue porque la aplicación no requería de predicciones iteradas salvo para realizar una reconstrucción dinámica: cuando se va a predecir las ventas para una semana, se tienen perfectamente determinadas las de las semanas anteriores, sin necesidad de utilizar las predicciones hechas por el modelo.

IV.2 Conclusiones generales

En esta tesis se han explorado algunas de las posibilidades de predicción de una serie temporal por medio de redes neuronales. Esto implicó una investigación del estado del arte de varias áreas: desde el software de análisis de series temporales y de los simuladores de redes neuronales a la de la teoría de sistemas y redes dinámico(a)s.

A continuación se resumen los principales resultados y conclusiones logrado(a)s:

1) ***De la selección de las herramientas de simulación de redes y de procesamiento de datos***:

- El proceso de selección de las herramientas de software necesarias fue un trabajo largo y muchas veces desalentador, ya que en general las más interesantes no estaban disponibles ni siquiera en forma de demo funcional o versión "trial". Se logró finalmente configurar un banco de pruebas basado en un conjunto de productos de software de calidad reconocida en el mercado: DATAPLORE, VRA, STATISTICA, TISEAN, NeuroSolutions y Microsoft Office 2000/XP. Aprender el manejo de algunas de ellas fue un proceso que tomó un tiempo no despreciable.

- El uso de un simulador de redes neuronales de uso industrial nos llevó a profundizar en varios aspectos, tanto de las redes neuronales en sí (por ejemplo, en lo que respecta a las redes con estructuras de memoria) como en la teoría de los sistemas dinámicos.

- A su vez, para trabajar con dichos sistemas dinámicos se tuvo que emplear software para el tratamiento de series temporales y de allí se pasó a considerar las series caóticas. Dado que no todo el "azar" encontrado era atribuible al caos, para modelar el sistema dinámico que generaba la serie temporal en estudio se debió hacer una exploración de los sistemas caótico-estocásticos y de los modelos de red capaces de predecir la serie temporal asociada a uno de ellos. Es de notar que tal vez una buena parte del estudio hecho sobre redes dinámicas (y sistemas dinámicos) hubiese sido omitido si se hubiese utilizado una herramienta de uso general que no contuviera referencias a dichas redes, para realizar las simulaciones. De esa forma, y al atacar un problema real, se aporta a este trabajo la parte de práctica profesional y la parte de tecnología al hacer la investigación del estado del arte en el área de software para tratamiento de series temporales y simulación de redes neuronales.

2) ***De los modelos (topologías) de red utilizados y del modelado del sistema dinámico:***

- En cuanto a los modelos de redes empleados, se utilizaron redes de topologías indicadas en la bibliografía como adecuadas a la tarea (TLFNs y recurrentes) junto con redes MLPs y la dada por el ensemble method y se pudo comparar la eficacia de cada una de ellas en la resolución del problema planteado.

- Dados los problemas que suelen presentar las redes recurrentes en el aprendizaje de dependencias a largo plazo, se incluyó una presentación del entrenamiento por medio del filtro de Kalman extendido y el uso de las redes LSTM.

- Se probó que la relación Entradas → Salidas de nuestro caso de estudio era aprendible, es decir, que el problema tenía solución; aún más, conjeturamos que el sistema no es caótico aunque sí estocástico (ya que solo conocimos los exponentes de Lyapunov de una realización del proceso aleatorio de las ventas) por lo que la calidad de la predicción de la media irá mejorando con el tiempo y el consiguiente aumento de la cantidad de datos de entrenamiento aunque siempre va a existir la posibilidad de una gran desviación del valor real respecto al predicho, debido a la aleatoriedad del sistema original.

- Para modelar el sistema dinámico que generaba la serie temporal utilizamos el modelo de espacio de estados, por lo que el problema de predicción de la serie fue convertido en el de predicción del siguiente estado del sistema. Este modelo de espacio de estados, junto con el método de los "delays" (coordenadas demoradas) fueron importantes en el desarrollo del trabajo práctico, más específicamente, en el diseño de la capa de entrada de algunas redes (MLPs) y de los parámetros de otras ("taps" y "delays" de las TLFNs), y ponen de manifiesto algo ya conocido: que el manejo que se haga de los datos de entrada a la red es fundamental en la calidad de los resultados a obtener. Adicionalmente, el resto de los componentes de la red fueron determinados muchas veces a través de procedimientos que se han utilizado tradicionalmente junto con las redes neuronales: los algoritmos genéticos. Es de observar que si bien existen numerosos estudios aplicables a la determinación de las entradas a la red (análisis de componentes principales (PCA), "feature extraction", etc., ver [1]), no sucede lo mismo con el diseño del resto de las capas, más específicamente, de las capas ocultas.

- Encontramos que un perceptrón multicapa es capaz de predecir los valores de la red, a corto plazo, es decir, que a pesar de su topología simple, es apto para resolver un problema como el planteado, siempre que se haga un estudio de los datos que permita determinar características de los mismos, tales como las indicadas por el Teorema de Takens-Mañé. El conocimiento del dominio del problema influyó también en ello: empíricamente se sabía que el aumento de precio de venta de un producto afecta sus ventas.

3) ***De los resultados numéricos obtenidos:***

- Se realizaron dos tareas: el desarrollo de un modelo para la reconstrucción dinámica del sistema del caso de estudio y el de otras redes adecuadas solo para la predicción de la serie temporal. En el mejor de los modelos de predicción, obtenido por "ensemble" de siete redes, se obtuvo un error promedio del 4.37% al predecir las semanas 2-18/2003, con un 4.85% de error en la predicción de la semana 2/2003. Sin embargo, a los efectos de manejar un modelo de predicción más simple, se sugirió un ensemble equiponderado de una red de Jordan y una TDNN, cuyo error promedio de predicción era del 9.23% y del 7.6% al predecir la semana 2/2003. Creemos que estos son valores aceptables dada la cantidad de datos disponibles, y que constituyen una prueba de que las redes neuronales son aplicables a casos de predicción de series temporales provenientes de sistemas dinámicos, sean o no estocásticos.

- Conjeturamos que la técnica del "ensemble" no presenta mayores ventajas al promediar valores obtenidos por medio de predicciones iteradas ("closed loop") de cada red, y arroja valores que son simplemente intermedios entre los de las redes promediadas o levemente mejores al mejor valor. Por ejemplo, se obtuvo un error promedio de 17.58% utilizando los coeficientes obtenidos a partir de los errores de "closed loop" de todas las redes aunque mejora a un 12.8% si se utilizan los coeficientes obtenidos a partir de los errores de "open loop". Por otra parte, al promediar los distintos modelos para estudiar su validez a corto y largo plazo, los promedios se realizaron utilizando diferentes ponderaciones: equiponderados, según el MDL, según el porcentaje de error al predecir un paso promediado sobre el conjunto de CV y según los coeficientes derivados de la matriz de correlación de errores dado en II.2.3. Observamos que el "ensemble" obtenido partir de varios modelos es válido a largo plazo, cuando son válidos a largo plazo. Es decir, y esta es una de las conclusiones de corte teórico de este trabajo, parecería ser que el uso de un comité de redes es particularmente beneficioso no solo porque disminuye el MSE promedio, sino porque permite obtener un modelo válido a largo plazo a partir de otros que lo son, con mejor comportamiento, siendo ésta una cuestión que, al menos en este trabajo, permanece abierta.

- No evaluamos las series obtenidas por medio de comparaciones exactas de los invariantes debido a la naturaleza estocástica del sistema original.

- La obtención de los datos y su depuración fue un proceso bastante accidentado, que finalmente nos proporcionó un conjunto de observaciones realmente escasas y, si consideramos las variables explicativas que no fuesen ventas ni temperaturas, de poca calidad. Esto nos enseñó algo que ya había sido anunciado en la bibliografía:

antes de emprender un proyecto de redes neuronales hay que asegurarse la existencia de un conjunto de datos adecuados que lo respalden y hagan factible. "Es un error capital teorizar antes de que uno tenga datos" (de Sherlock Holmes a Watson, en "Un escándalo en Bohemia", de Arthur Conan Doyle).

4) ***De los estudios teóricos suplementarios requeridos:***

- Como ya se dijo, para la construcción de los modelos de predicción y reconstrucción dinámica se hizo necesario el estudio de los sistemas dinámicos (determinísticos y estocásticos) y de sus invariantes: dimensión y "delay" del "embedding", exponentes de Lyapunov, etc. a los efectos de determinar la validez a largo plazo del modelo construido.

5) ***De los criterios de selección de modelos:***

- Si bien esto formó parte de los estudios teóricos adicionales, preferimos establecerlo en forma separada dada la importancia que tiene el tema en términos generales.

- Para poder elegir de entre los modelos válidos a los más deseables, se analizaron algunos de los criterios de selección de modelos y se eligió el del MDL. Vimos que elegir el modelo a utilizar basándose solo en la estimación que la herramienta hace del MDL no conducía a resultados positivos, por lo que decidimos primero seleccionar los modelos que produjeran resultados válidos (%Error CV < 12%) y de entre ellos, sí elegir utilizando el MDL. Posteriormente, al considerar la estimación que hace el simulador del MDL determinamos que la misma era pobre ya que no reflejaba ciertas complejidades intrínsecas del modelo, sino solo el número de pesos y el error MSE. De todas maneras, el modelo óptimo para la predicción según esta estimación del MDL resultó ser una red TDNN, es decir, una red de un tipo sugerido en la bibliografía como muy recomendable para la predicción temporal, que resultó ser válida a corto plazo y que tal vez hubiese sido un modelo también válido a largo plazo si se hubiesen tenido los suficientes datos de entrenamiento.

6) ***De la aplicación real de los modelos construidos:***

- Se introdujo al usuario del modelo de predicción en los aspectos principales de las redes neuronales, y se lo motivó a utilizarlas en otras áreas de la empresa, estableciendo las bases para la realización de futuros proyectos que arrojen resultados más positivamente contundentes. Se le proveyó de los modelos elegidos y del simulador de redes utilizado, y se le explicó brevemente cómo utilizarlo para que el modelo predijera las ventas de la semana siguiente.

- Se le insistió al usuario sobre la calidad necesaria en los datos para que las predicciones sean más acertadas, tanto a nivel de cantidad de valores de la serie que deben estar disponibles, como a nivel de los datos que se deben recopilar en forma sistemática: por ejemplo, cuando se precisó saber cuales de los aumentos de precio de venta del gas habían sido anunciados en la prensa, se encontró que esa información no estaba disponible para el usuario del modelo ni para los vínculos dentro de la empresa que trabajaban con él, por lo que se determinó que debía registrarse la fecha en la que se produce cada aumento y si fue o no anunciado.

7) ***Del relevamiento bibliográfico:***

- Se realizó una revisión bibliográfica que trató de ser amplia, tanto a nivel de material impreso como en formato electrónico, a los efectos de compendiar los principales aspectos del estado actual del arte de la predicción de series temporales utilizando redes neuronales. El material encontrado fue a veces sumamente redundante (como en el caso de las mejoras al algoritmo de BP) y muy escaso en otros casos (como en el de las estructuras de memoria o el de la estimación de la dimensión del espacio de reconstrucción en el caso estocástico). Esta revisión bibliográfica, que abarca tanto trabajos de investigación ya clásicos (por ejemplo, [30], [44] o [45]) como investigación muy reciente (tal como [63]), constituye otro aporte de esta tesis.

IV.3 Trabajo futuro

Existen varias posibles áreas en las que se podría continuar o profundizar el presente trabajo:

- a nivel práctico,

  o pasar a predecir las ventas de gas diarias, con todo la consiguiente investigación sobre las variables descriptivas necesarias, reducción de dimensionalidad de la entrada, etc. asociada;

  o generación de código ejecutable, de forma que el usuario final de la red solo tenga que cargar algunos archivos con datos y ejecutar una aplicación para obtener su predicción.

  o continuar con la experimentación con otras profundidades de memorias gamma y probar el entrenamiento de las redes recurrentes utilizando deKf, ya que en el caso de ventas diarias seguramente va a aparecer el problema del gradiente evanescente de una forma más problemática, y no como aquí, que solo impedía aprender la tendencia.

- a nivel teórico,

  o profundizar en el conocimiento de la interacción de los algoritmos genéticos con las redes neuronales. Por ejemplo, cuando se construyen redes basadas en "comités de expertos", ¿las redes que constituyen los distintos expertos pueden ser entrenadas mediante algoritmos genéticos en forma independiente, concurrentemente con el entrenamiento (también por algoritmos genéticos) de la que actúa como "compuerta" selectiva? Esto podría corresponderse con un símil del Teorema de Kolmogorov en el sentido de que la evolución genética de una red puede ser descripta a partir de la evolución de sus componentes.

  o estudiar las propiedades a largo plazo del modelo generado por un comité de redes, y en su forma más general, por un promedio ("ensemble") de estimadores de una variable temporal. En qué condiciones el ensemble de modelos es válido a largo plazo?

  o analizar como se puede calcular el MDL (y su respectiva estimación) de un ensemble de modelos, a los efectos de poder elegir el "mejor" ensemble, es más, dada una red que tiene asociado un cierto MDL, ¿cómo se puede descomponer en subredes de forma que el comité de redes que formen tenga un MDL mejor que el de la red original?

  o continuar con el estudio de los sistemas dinámicos estocásticos, por ejemplo en lo que respecta a la determinación de la dimensión del subespacio de la señal (tal vez intentando establecer una generalización probabilística del método de los FNN).

Sección V: Bibliografía

Se detallan a continuación los libros, publicaciones y principales sitios web consultados en la elaboración del presente trabajo. Adicionalmente se puede utilizar la bibliografía que aparece en cualquiera de los textos mencionados.

V.1 Documentación impresa consultada

V.1.1 Libros

[1] *Bishop, Christopher* -- **Neural Networks for pattern recognition**. Clarendon Press, Oxford, 1996.

[2] *Cover, Thomas M. – Thomas, Joy A.* -- **Elements of Information Theory**. John Wiley &Sons, 2002.

[3] *Freeman, James - Skapura, David* -- **Neural Networks. Algorithms, Applications and Programming Techniques.** Addison Wesley, 1992.

[4] *Hassoum, Mohamad* -- **Fundamentals of artificial Neural networks.** MIT Press, 1995.

[5] *Haykin, Simon* -- **Neural Networks. A comprehensive foundation.** Prentice Hall, 1999.

[6] *Hilera, José - Martínez, Víctor* -- **Redes Neuronales Artificiales- Fundamentos, modelos y aplicaciones**. Addison Wesley Iberoamericana, 1995.

[7] *Jang, Jyh-Shing – Sun, Chuen-Tsai – Mizutani, Eiji* -- **Neuro Fuzzy and Soft computing. A computational approach to learning and machine intelligence**. Prentice Hall, 1997.

[8] *Linggard, R. –- Myers, D.J. – Nightingale, C.* -- **Neural Networks for vision, speech and natural language.** Chapman & Hall, 1992.

[9] *Ljung, Lennar* -- **System identification: Theory for the user**. Prentice Hall, 1995.

[10] *Motulsky, Harvey – Christopoulos, Arthur* -- **Fitting models to biological data using linear and nonlinear regression. A practical guide to curve fitting.** Graph Pad Software Inc., 2002.

[11] *Principe, José – Euliano, Neil – Lefebvre, W. Curt* -- **Neural and Adaptive Systems Fundamentals through simulations**. John Wiley & Sons, 2000.

[12] *Principe, José – Euliano, Neil – Lefebvre, W. Curt* -- **Neurosolutions Ver. 4. User's guide.**

[13] *Swingler, Kevin* -- **Applying Neural Networks: A practical guide**. Academic Press, 1996.

V.1.2 Publicaciones

[15] *Afamado, Laura – Carratú, Alejandra – Sansonetti, Daniele* -- **Redes neuronales artificiales aplicadas a la predicción de la demanda eléctrica horaria.** Proyecto de taller V, InCo, Facultad de Ingeniería.

[16] *Baldi, Pierre – Chauvin, Yves y otros* -- **Assessing the accuracy of prediction algorithms for classificacion: An overview**. *Bioinformatics*, Vol .16, No. 5, 412-424, (2000)

[17]] *Baum, Eric B. – Wilczek, Frank* -- **Supervised learning of probability distributions by neural networks.** *Neural Information Processing Systems* (D. Z. Anderson, ed.), American Institute of Physics, 1988.

[18] *Baum, Eric – Haussler, David* -- **What size net gives valid generalization?** Neural Computation, No. 1, pp. 151-160, 1989.

[19] *Bengtsson, Thomas* -- **An improved Akaike Information Criterion for state-space model selection**. National Center for atmosferic research, Department of Statistics, University of Missouri-Columbia, Estados Unidos, 2003.

[20] *Belaire Franch, Jorge – Contreras Bayarri, Dulce* -- **Recurrence plots in nonlinear time series analysis: free software. Reporte Técnico** DT 01-01. Universidad de Valencia. 2001.

[21] *Bishop, G. – Welch, G.* -- **An introduction to the Kalman filter**. Informe técnico TR 95-041. University of North Carolina at Chapel Hill. Department of Computer Science. 2002.

[22] *Castiglioni, Filippo* -- **Forecasting price increments using an artificial neural network**. Advanced Complex Systems, Vol. 1, pp. 1-12, 2000.

[23] *Chen, Ying – Brooks, Richard y otros* -- **Efficient global optimization for image registration**. IEEE Transactions on knowledge and data engineering, Vol 14 No. 1. Enero/Febrero 2002.

[24] *Colombert, Isabelle – Ruelland, Allan y otros* -- **Models to predict cardiovascular risk: comparison of CART, multilayer perceptron and logistic regression.** Proceedings of the AMIA Symposium, pp. 156-160, 2000.

[25] *Fueda, Kaoru – Yanagawa, Takashi* -- **Estimating the embedding dimension and delay time from chaotic time series with dynamic noise.** Journal of Japan Statistic Society, Vol 31, No. 1, 2001.

[26] *Gao, Qiong – Li, Ming – Vitányi, Paul* -- **Appliying MDL to learning best model granularity.** Artificial Intelligence, Vol. 121 , No. 1-2, pp. 1-29. 2000.

[27] *Gers, F.A. – Schmidhuber,J. – Cummings,F.* -- **Learning to forget: Continual prediction with LSTM**. Neural computation, Vol. 12 No. 10, 2000.

[28] *Ginsburg, Iris – Horn, David* -- **Combined neural networks for time series analysis. Advances in Neural Information Processing Systems** - 6 - (NIPS*93), 224-231. Morgan Kaufmann, San Mateo, CA , 1994.

[29] *Gori, M. – Maggini, M.* -- **Optimal convergence of on line backpropagation.** *IEEE Transactions on Neural Networks*, Vol. 7, No. 1, pp. 251-253, Enero 1996.

[30] *Han, Seung-Soo – May, Gary* -- **Optimization of Neural Network structure and learning parameters using genetic algorithms**. Eigth International Conference on Tools with Artificial Intelligence (ICTAI '96), Toulouse, France. IEEE Computer Society, 1996, ISBN 0-8186-7686-8. pp. 200-206, 1996.

[31] *Hansen, Mark – Yu, Bin* -- **Model selection and the principle of minimun description length.** Journal of the American Statistics Association, Vol. 96.

[32] *Hegger, Rainer – Kantz, Holger – Schreiber, Thomas* -- **Practical implementation of nonlinear time series methods: The TISEAN package.** CHAOS, nro. 9, pp. 413-435. 1999.

[33] *Heskes, Tom M – Kappen, Bert* -- **On line learning processes in artificial neural networks**. Mathematical Foundations of Neural Networks, ed. Taylor, J., Elsevier, Amsterdam, pp. 199-233, 1993.

[34] *Heskes, Tom – Wiegerink, Win* -- **On Line Learning with time correlated examples**. Department of medical Physics and Biophysics, University of Nijmegen. The Nederlands

[35] *Hochreiter, Sepp – Schmidhuber, Jürgen* -- **Long short-term memory.** Vol 9, No. 8, 1997.

[36] *Judd, Kevin – Small, Michael* -- **Toward log term prediction.** Physica D, Vol 136, pp. 31-34, 2000.

[37] *Kelly, Patrick y otros*. -- **Digital diffusion network for image segmentation –** Proceedings of the 1995 International conference on Image Processing (IEEE). Vol 3.

[38] *Kolen, John F. – Pollack, Jordan B.* -- **Back propagation is sensitive to initial conditions**-. Laboratory for Artificial Intelligence Research. The Ohio State University, 1991.

[39] *Latora , Vito – Baranger, M.* -- **Kolmogorov-Sinai Entropy-Rate vs. Physical Entropy**. Physical Review Letters Vol. 82, 1999.

[40] *Leen, Todd K. – Moody, John E.* -- **Stochastic Manhatan Learning: An exact time-evolution for the ensemble dynamics**. Online Learning in Neural Networks, The Newton Institute Series, Cambridge University Press, Cambridge, 1999.

[41] *Martins, José Antonio* – **Avaliacao de diferentes tecnicas para reconhecimento de fala.** Tesis doctoral. Universidade Estadual de Campinas, 1997.

[42] *Moody, John* -- **Prediction risk and architecture selection for neural networks**. Aparece en 'From statistics to neural networks: Theory and pattern recognition applications', NATO ASI series F, editores V. Cherkassky, J.H. Friedman y H. Wechlsler. Springer Verlag, 1994.

[43] *Moody, John* -- **The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems.** Advances in Neural Information Processing Systems, No. 4, San Mateo, California. Morgan Kauffmann, 1992.

[44] *Morgan, N. – Boulard, H.*-- **Generalization and parameter estimation in feedforward Nets: some experiments**. Aparece en `Advances in Neural Information Processing Systems 2', D. Touretzky, editor, Morgan Kaufmann, San Mateo, CA, pp. 630--637.

[45] *Muldoon, M.R. - Broomhead, D.S. – Huke, J.P.* -- **Delay Embedding in the Presence of Dynamical Noise.** Dyn. and Stab. Systems. Vol 13. 1998.

[46] *Murata, Noburu – Yoshizawa, Shuji* -- **Network information criterion. Determining the number of hidden units for an artificial neural network model.** IEEE Transactions on Neural Networks, Vol 5, noviembre, pp. 865-872. 1994.

[47] *Myung, In – Pitt, Mark – Zhang, Shaobo – Balasubramanian, Vijay* -- **The use of MDL to select among computational models of cognition.** Advances in Neural Information Processing Systems 13, Papers tomados del 'Neural Information Processing Systems (NIPS) 2000', Denver, CO, USA. MIT Press. pp. 38-44. 2001.

[48] *Nowlan, Steven – Hinton, Geoffrey* -- **Simplifying neural networks by soft sharing**. Neural Computation, Vol 4, No. 4, pp. 473--493, 1992

[49] *Olofsen, Erik* -- **The identification of strange attractors using experimental time series**. Tesis de Maestría. Twente University, Holanda, 1991.

[50] *Pearlmutter, Barak* -- **Dynamic Recurrent neural Networks**. School of computer Science, Carnegie Mellon University. Reporte Técnico No. CMU-CS-90-196. 1990.

[51] *Pearlmutter, Barak* -- **Gradient Descent: Second order momentum and Saturating Error**. Advances in Neural Information Processing Systems 4, pp. 887-894. Morgan Kaufmann, 1992.

[52] *Patel, Gaurav S.* -- **Modeling Nonlinear Dynamics with Extended Kalman Filter trained recurrent multilayer perceptrons**. Tesis de Maestría. Mc. Master University, 2000.

[53] *Pérez-Ortiz, Juan Antonio y otros* -- **Kalman Filters improve LSTM network performance in problems insolvable by traditional recurrent nets**. Neural Networks, Vol. 16, No. 2, 2003.

[54] *Pérez-Ortiz, Juan Antonio* -- **Modelos predictivos basados en redes neuronales recurrentes de tiempo discreto** Tesis doctoral, Universidad de Alicante, España. 2002.

[55] *Perrone, Michael Peter* -- **Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization.** Tesis doctoral. Brown University, 1993.

[56] *Petry, Adriano* -- **Estudo sobre Aplicabilidade da Teoria de Sistemas Dinâmicos Não-Lineares para o Reconhecimento Automático de Locutor**. Exame de Qualificação, Curso de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, 2000.

[57] *Popescu, I. y otros* -- **Prediction of outdoor propagation path loss with neural networks**. Informatica, Vol. 10 Nro. 2, pp. 231-234. Universidad de Timisoara, Rumania, 1999.

[58] *Príncipe, José – Kuo, Jyh-Ming* -- **Dynamic modeling of chaotic time series with neural networks.** Neural Information Processing Systems, pp. 311-318, Conferencia en Cambridge, Massachussets, 1995.

[59] *Raudys, Aistis – Mockus*, *Jonas* -- **Comparison of ARMA and multilayer perceptron based methods for economic time series forecasting**. Informatica, Vol 10 Nro. 2 pp. 231-244. Institute of Mathematics and Informatics, Lituania, 2001.

[60] *Rögnvaldsson, Thorsteinn* -- **On Langevin Updating in Multilayer Perceptrons**. Neural Computation. Vol. 6 No. 5, Setiembre 1994.

[61] *Saarinen, S. – Bramley, R. – Cybenko, G.* -- **Ill-Conditioning in neural network training problems.** SIAM Journal on Scientific Computing Vol. 14, No. 693, 1993.

[62] *Schittenkopf, Christian - Dorffner, Georg - Dockner, Engelbert J.* -- **On Nonlinear, Stochastic Dynamics in Economic and Financial Time Series** Studies in Nonlinear Dynamics & Econometrics: Vol. 4 No. 3. 2000.

[63] *Small, Michael* -- **Estimating the distribution of dynamic invariants: Illustrated with an application to human photo-plethysmographic time series**. Nonlinear Sciences, agosto/2003.

[64] *Stark, Jaroslav* -- **Delay Reconstruction: Dynamics v Statistics.** *Nonlinear Dynamics and Statistics*, ed A.I. Mees, Birkhauser, 2001.

[65] *Stark, J. - Broomhead, D.S. – Davies, M.E.- Huke, J.* -- **Delay Embeddings of Forced Systems: II Stochastic Forcing.** Journal of Nonlinear Science, Setiembre/1999.

[66] *Wang, Changfeng – Venkatesh, Santosh – Judd, Stephen* – **Optimal stopping and effective complexity in learning.** Advances in Neural Information Processing Systems, Vol. 6, Morgan Kaufmann, San Francisco, pp. 303-310, 1994.

[67] *Wang, Chuang – Principe, Jose.* -- **Training Neural Networks With Additive Noise in The Desired Signal**. IEEE Transactions on Neural Networks, Vol. 10, No. 6, pp. 1511-1517, Nov 1999.

[68] *Williams, Ronald – Zipser, David* -- **A learning algorithm for continually running fully recurrent neural networks**. Neural Computation, pp. 270-280, 1989.

[69] *Williams, Ronald* – **Some observations on the use of the extended Kalman filter as a recurrent network learning algorithm**. Technical Report NU-CCS-92-1. Boston: Northeastern University, College of Computer Science, 1992.

[70] *Wittner, Ben – Denker, John* -- **Strategies for teaching layered networks classification tasks**. American Institute of Physics, 1988.

[71] *Yao, Xin* -- **Evolutionary artificial neural networks**. Aparece en *Encyclopedia of computer science and technology*, A. Kent y otros editores, Vol 33, pp. 137-170, Marcel Dekker Inc., New York, 1995.

V.2 Documentación utilizada en formato electrónico

A continuación se listan los principales sitios web utilizados para el desarrollo de este trabajo y se da un breve comentario sobre ellos.

http://www.emsl.pnl.gov:2080/proj/neuron/neural/gateway/USA.html Artificial Neural Networks: Tiene vínculos relacionados con universidades, grupos de desarrollo y compañías relacionadas con las RNA. De acceso libre.

http://neuralnetworks.ai-depot.com/Applications.html Neural networks warehouse: software, ensayos, etc. De acceso libre.

http://nips.djvuzone.org/ NIPS (Neural Information Processing Systems) on line es una muy completa recopilación de publicaciones sobre redes neuronales con más de 15000 páginas de texto (1958 artículos) disponibles. Adicionalmente, el formato de los trabajos por defecto es el DJVU, que es uno de (por no decir el) más los compactos al momento de descargar los papers. El "driver" necesario para manejarlos se puede obtener desde el propio sitio. De acceso libre.

http://www.neurocolt.com/abs/ Neural Networks and Computational Learning Theory . De acceso libre.

http://www.1t.uom.gr/pdp/DigitalLib/Neural/Neu_soft.htm El sitio de la Universidad de Macedonia, Grecia, sobre Tecnologías de Proceso Distribuido Paralelo de la información. De acceso libre.

http://citeseer.nj.nec.com Es la biblioteca digital del NEC Research Institute. Muy completa y de libre acceso.

http://www.computer.org/publications/dlib/index.htm Biblioteca digital de la IEEE. Es otra fuente muy completa y confiable de información: incluye todos los artículos aparecidos en las diferentes publicaciones de la IEEE. El acceso irrestricto es por suscripción, o se pueden descargar artículos sueltos, algunos de los cuales son con costo.

http://www.5campus.com Es el sitio web de la Facultad de Ciencias Empresariales de Zaragoza, España. Posee algunos artículos interesantes, como el de Mireya Arellano "Introducción al Análisis Clásico de Series de Tiempo"

http://www.mdl-research.org Un sitio dedicado a la investigación asociada al MDL. De libre acceso.

http://apnonlin.zweb.com/csp_man El manual on-line del cspX. Una fuente interesante de conocimientos sobre procesos caóticos. De libre acceso, pero el cspX no es freeware.

http://www.mpipks-dresden.mpg.de/~tisean/ El manual on line del TISEAN. Realmente provee toda una gama de definiciones y explicaciones sobre sistemas dinámicos y series temporales. De libre acceso

www.mathworld.com El sitio del producto Mathematica y otras enciclopedias científicas. Muy recomendable y útil a la hora de buscar alguna definición puntual, ya que además tiene las referencias bibliográficas de donde fueron tomadas. De libre acceso.

http://home.netcom.com/~eugenek/download.html La página de descarga del VRA. Junto con el producto se incluye documentación y ejemplos. De libre acceso.

V.3 Algunos comentarios sobre la bibliografía manejada

Realizaremos a continuación un breve comentario sobre las obras consultadas. No se comentan las publicaciones debido a su gran cantidad.

Para empezar encontramos distintos niveles en ellas, ya sea de formalismo, ya sea de generalidad (o alcance) de los temas expuestos. Para ser más claros: podemos distinguir obras más o menos claras en su exposición, o más o menos completas en cuanto a temática. Entre las especialmente claras en lo que toca a su redacción y en cuanto a la fundamentación matemática que da a sus conclusiones, encontramos las de Freeman y Skapura, de Haykin y de Bishop. En especial, la de Bishop (aunque su título pueda hacer pensar que esta restringida al reconocimiento de patrones) y la de Freeman y Skapura son textos que encontramos balanceados entre extensión del contenido, profundidad y claridad. También profundo y claro, y muy completo, es el de Haykin, en él se tratan los temas en forma concisa sin perder por ello contenido ni claridad por lo que puede ser utilizado como libro de referencia.

Hassoum, por su parte, si bien es riguroso en sus conclusiones, bastante completo en su alcance y provee una gran cantidad de referencias puntuales (es decir, para cada tema que se va tratando) a los trabajos originales, puede ser poco claro en su exposición y resulta un libro algo difícil de utilizar como texto de consulta.

Una obra con un enfoque eminentemente práctico y muchas buenas ideas a tener presentes en el desarrollo de un proyecto de redes neuronales, es la de Swingler, sin embargo, maneja topologías muy simples, no incluyendo las redes estocásticas, las recurrentes ni las TLFNs.

El libro de Hilera y Martínez es muy similar en cuanto a los puntos tratados al de Freeman y Skapura, aunque sus fundamentaciones a veces no poseen el rigor matemático de aquel. Sin embargo, tiene un atractivo interesante: esta escrito en español.

La obra de Príncipe, Euliano y Lefebvre es completa en ciertos aspectos, aunque no aborda el tema de redes estocásticas. Muy extensa en el tratamiento de series temporales, incluye anexos con algunos de los fundamentos de la geometría y álgebra utilizados en el texto. En cuanto a las justificaciones, las separa del resto del material, explicitando así que son algo opcional. Un inconveniente que presenta es que fue pensado para ser leído en forma secuencial (algo similar ocurre con el de Hassoum), por lo que su consulta puntual puede hacerse dificultosa. El libro incluye una versión en CD con todo el texto de la obra y los ejemplos interactivos usando la herramienta NeuroSolutions. Esos ejemplos no son simples demos: el lector puede cambiar los parámetros, etc. y ver como reacciona la red. Por último, esta obra tiene un glosario donde se explican muchos términos técnicos usados comúnmente en la práctica (tales como "rattling", canal, época, etc.).

Si bien esta orientado a sistemas fuzzy, el libro de Jang, Sun y Mizutani condensa en pocas hojas parte de los principales aspectos de las RNA: optimización por descenso en el gradiente, variaciones y mejoras a ella (método de Newton, etc.), back-propagation, back-propagation extendida para redes recurrentes (BPTT y RTRL), aprendizaje no supervisado, etc. Es un texto adecuado para consultar conceptos en forma aislada. Como no se extiende en cada tema, se hace necesario recurrir a otras obras si se quieren más detalles.

En cuanto al libro de Motulsky, si bien no es un texto dedicado a las RNA, nos fue especialmente útil para aclarar algunos aspectos prácticos del índice de Akaike en la comparación de modelos. Algo similar sucede con el de Cover y Thomas, con el de Ljung y con el de Linggard y otros: fueron consultados por temas muy puntuales. En especial, este último describe muy claramente el comportamiento de una red neuronal como sistema dinámico, aunque las aplicaciones que trae como ejemplo se basan casi exclusivamente en MLPs.

Por último, el manual del simulador usado fue de ayuda, aunque no tiene una organización "de libro": en realidad, esta más bien pensado para ser leído al azar, vía el "help on line" de la herramienta. Es muy extenso (unas 1000 hojas) y muchas veces no aclara conceptos teóricos.

Para resumir estos comentarios, las calificamos en un cuadro según los siguientes criterios:

- Formalismo: el grado de formalismo y rigor matemático utilizados en las diferentes deducciones y demostraciones
- Generalidad y extensión: el alcance de la obra en cuanto a temas distintos tratados.
- Ejemplificación: si la obra incluye ejemplos o ejercicios.
- Idioma en que esta escrita la versión usada: si bien este punto no califica la calidad del texto, sí permite hacerse una idea de lo difícil que puede ser leerlo durante cierto tiempo ininterrumpido.
- Inclusión de fundamentos matemáticos: si el libro incluye algún anexo con parte de los fundamentos de la matemática requerida para entender la obra.
- Tipo de utilización: si se puede usar como texto de consulta o debe ser leído secuencialmente desde el comienzo.
- Inclusión de referencias: canto se discrimina dentro del texto las distintas referencias utilizadas, es decir, si las referencias aparecen indicadas dentro del texto, que son consideradas como lo mejor, (por ejemplo, luego de una definición, dar la referencia de quien definió el concepto de esa forma), a nivel de capítulo al que corresponden o solo a nivel del libro completo (lo peor).

Los valores posibles para cada ítem fueron:

* Pobre o deficiente
** Normal
*** Muy bueno

y en el caso del idioma, solo indicamos I – Inglés y E – Español

NA: no aplica.

El cuadro asociado es:

Ítem/Obra	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[9]	[8]	[10]	[11]	[12]	[13]
Formalismo	**	***	***	***	***	**	***	NA	NA	NA	*	NA	*
Generalidad	**	NA	**	**	***	**	**	NA	NA	NA	**	**	**
Ejemplos	**	**	**	**	**	**	**	NA	**	NA	***	**	*
Idioma	I	I	I	I	I	E	I	I	I	I	I	I	I
Fundamentos matemáticos	***	*	*	*	*	*	*	**	*	*	***	*	*
Para consulta	***	***	**	*	***	**	***	**	**	***	*	***	***
Referencias	***	**	**	***	**	**	**	**	*	**	**	*	*

ANEXOS

ANEXO I: Mejoras y variaciones al algoritmo de BP básico

En este anexo se describen algunas variaciones al algoritmo de "back-propagation", ya sea en cuanto a la modificación de algunos de los parámetros de aprendizaje así como en la de la función de error.

AI.1 Métodos de segundo orden y de momentos

AI.1.1 Los momentos

La técnica de los "momentos" consiste en la adición de un término ***momento*** en la parte derecha de las reglas de actualización de pesos en las ecuaciones

$$\Delta w_{lj} = \rho_o (d_l - y_l) f_o^{'}(Net_l) z_j \quad \text{(capa de salida)}$$

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

Lo que se consigue es que el término que se agrega acelere el descenso cuando las derivadas de la función de error tienen el mismo signo en dos instantes consecutivos y no lo deja oscilar con cada cambio en el signo de la derivada $\frac{\partial E}{\partial w_i}$ [5, Cap.4]. Las ecuaciones correspondientes son[14]:

$$\begin{cases} \Delta w_i(t) = -\rho \dfrac{\partial E}{\partial w_i} + \alpha \Delta w_i(t-1) \\ \Delta w_i(t) = w_i(t) - w_i(t-1) \end{cases} \qquad \textbf{(AI.1)}$$

con α = ***tasa de momento***, generalmente $0<\alpha<1$ (en una forma más general, $0<|\alpha|<1$, aunque no es usual utilizar valores negativos de α)

El método del momento es una forma de aumentar la tasa efectiva de aprendizaje en regiones de la superficie de error casi planas mientras que se mantiene una tasa de aprendizaje cercana a ρ (con $0<\rho \ll 1$) en regiones con altas fluctuaciones. En efecto, si usamos una recursión en N (N arbitrario) pasos podemos rescribir la ecuación (AI.1) como

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

Si se toma el punto de búsqueda en una región casi plana, entonces $\frac{\partial E}{\partial w_i}$ va a ser casi constante en cada paso, y la ecuación anterior puede ser aproximada como

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

cuando $0<\alpha<1$ y N grande.

[14] Pasamos aquí a anotar los pesos como elementos de un vector, sin considerar las neuronas que conectan.

Por lo tanto, para regiones aplanadas, el término de momento lleva a aumentar la tasa de variación de los pesos en un factor $1/(1-\alpha)$.

El momento estático

Se dice que el momento añadido es estático si su tasa α no depende de la etapa del aprendizaje en que se este.

El momento adaptativo o dinámico

En este caso la tasa de momento irá cambiando con el tiempo. Hay varias técnicas propuestas para ello:

AI.1.2 El método de Newton

Las búsquedas de segundo orden (tales como el método de Newton) se basan a en una aproximación cuadrática $E_2(\mathbf{w})$ de la función $E(\mathbf{w})$, es decir, en $E_2(\mathbf{w})$ se usan los primeros tres términos del desarrollo de Taylor de $E(\mathbf{w})$ [4]:

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

siendo $\mathbf{H}$ la matriz hessiana $\mathbf{H} = \left\{ H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j} \right\}$

Minimizar E será aproximadamente igual a minimizar $E_2(\mathrm{w})$, y en especial si queremos minimizar $E_2(\mathbf{w}^{actual} + \Delta\mathbf{w})$, tendremos que resolver

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

lo que se consigue haciendo [4]

$$\Delta\mathbf{w} = -\left[\mathbf{H}(\mathbf{w}^{actual})\right]^{-1} \nabla E(\mathbf{w}^{actual}) \qquad \textbf{(AI.2)}$$

Al cálculo iterativo de los cambios en los pesos usando (AI.4) se lo llama ***algoritmo de Newton***.

El cálculo del $\mathbf{H}^{-1}$ es muy costoso computacionalmente hablando (requiere $\boldsymbol{O}(W^3)$ con W el número de pesos), por lo que algunos investigadores (Le Cun y Becker, citados por [4]) propusieron una aproximación a ésta que descarta las elementos fuera de la diagonal de $\mathbf{H}$, y que calcula

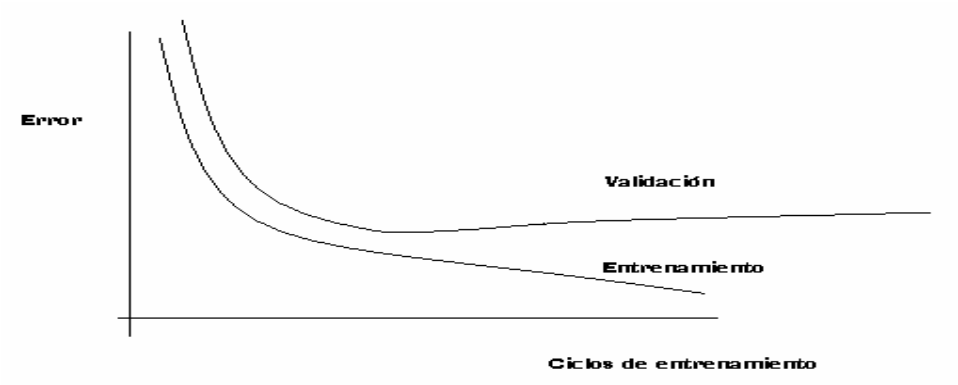
$$\Delta w_i = -\frac{\partial E}{\partial w_i}\left(\frac{\partial^2 E}{\partial w_i^2}\right)^{-1}$$

AI.2 Otros criterios de detención

AI.2.1 "Cross validation"

Esta alternativa (o estrategia complementaria) para mejorar la capacidad de generalización de una red está basada en resultados empíricos ([44], Wieigend, Hergert, etc. citados por [4]) y es básicamente un criterio de detención del entrenamiento. En simulaciones de entrenamiento de redes "feedforward" usando retro-propagación con datos que incluían ruido, se encontró que el error de generalización decrecía monótonamente a un mínimo para luego empezar a crecer aunque el error de entrenamiento continuara decreciendo.

Gráficamente:



Para resumir, cuando se entrena con datos con ruido, el entrenamiento excesivo lleva en general al "overfitting", mientras que un entrenamiento parcial puede llevar a una mejor aproximación de la función conocida en el sentido de una mejor interpolación, y posiblemente, mejor extrapolación. Wang y otros (citado por [4]) dieron una justificación formal al fenómeno de mejora de la generalización al detener el aprendizaje antes que el mínimo global del error de entrenamiento sea alcanzado. Ellos demostraron que existe un momento crítico en el proceso de entrenamiento en el cual la red entrenada generaliza en la mejor forma posible, y que luego de ese punto el error de generalización aumenta. Por lo tanto, una estrategia factible para mejorar la generalización en redes de tamaño no óptimo[15] es evitar el sobre-entrenamiento controlando cuidadosamente la evolución del error de validación durante la etapa el entrenamiento y deteniendo el entrenamiento justo antes de que dicho error empiece a crecer. Esta estrategia es conocida como "***cross validation***". En este método todo el conjunto de datos disponibles es dividido en dos partes: conjunto de entrenamiento y conjunto de validación. El conjunto de entrenamiento es usado para determinar los valores de los pesos de la red. El conjunto de validación es usado para decidir cuando terminar el entrenamiento. El entrenamiento continúa mientras que la performance en el conjunto de validación continúe mejorando. Cuando deja de mejorar, se detiene el entrenamiento[16].

Hay que tener presente que este criterio requiere tener abundantes datos, lo que no lo hace apropiado para aplicaciones que sufren de escasez de ellos [4]. En esos casos, se pueden utilizar variantes de "cross validation", tales como la "***multifold cross validation***" en la cual se dividen los N patrones disponibles en K subconjuntos, $K>1$, se entrena el modelo para todos los subconjuntos excepto uno y el error de validación es medido para el subconjunto que se excluyó. El proceso se repite para los K subconjuntos, usando cada vez un subconjunto distinto para validación. La performance del modelo es evaluada promediando el error de validación en las K pruebas [5].

Finalmente, el criterio de cross validation permite elegir el modelo de red que mejor se ajuste a los datos y que conduzca a una mejor capacidad de generalización, y en ello se relaciona con el MDL y el AIC (ver ANEXO V). En efecto, aquí elegiríamos el modelo tomando la red tal que provee el mínimo error en el conjunto de "cross validation", es decir, elegir el modelo m-ésimo con $m = \{\min_k E_{CV}(k)\}$ y E_{CV} es el error calculado para el conjunto de "cross validation". Se puede demostrar que el uso de "cross validation" es asintóticamente (o sea, cuando el conjunto de datos crece indefinidamente) equivalente al índice de Akaike [11].

---

[15] En [66] se demuestra que detener el entrenamiento utilizando CV tiene el mismo efecto en la capacidad de generalización de la red que variar el tamaño de la misma, para un conjunto de entrenamiento dado.

[16] Las herramientas de software suelen permitir establecer cuánto se permite crecer el error antes de detener el entrenamiento.

AI.2.2 Otras funciones [de criterio] de error

Las funciones de error que se usen deberían cumplir las condiciones que definen una distancia: el error cometido al obtener un cierto valor $\mathbf{i}$ como salida de la red en vez de un esperado $\mathbf{j}$ (deseado), llamémoslo $d(\mathbf{i},\mathbf{j})$ será tal que

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) $d(\mathbf{i},\mathbf{j}) \leq d(\mathbf{i},\mathbf{h}) + d(\mathbf{h},\mathbf{j})$

con $\mathbf{i} = \mathbf{i}(\mathbf{w}),\ \mathbf{w} \in \mathbf{R}^n,\quad \mathbf{i},\mathbf{j} \in \mathbf{R}^m$ y además, si se va a emplear un entrenamiento basado en derivadas (tal como el descenso en el gradiente), debe estar definido el gradiente $\nabla d = \left[\frac{\partial d}{\partial w_1}, \frac{\partial d}{\partial w_2}, \cdots \frac{\partial d}{\partial w_n}\right]$ el que en general se exige que sea continuo respecto **w.** Las condiciones 3) y 4) no son exigidas para algunas funciones de error, tales como la entropía relativa.

Las funciones de error de Minkowsky

Son una familia de funciones paramétricas, de parámetro $r \geq 1$, que pueden ser vistas como una posible generalización de la suma de errores cuadráticos. Su forma general es[17]:

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

siendo n el número de ejemplares de entrada, m la cantidad de neuronas de salida y significando $(d_i)_j$ "el elemento i-ésimo de $\mathbf{d}_j = [(d_1)_j, (d_2)_j, \ldots (d_m)_j]$", o en su versión instantánea:

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

Estas funciones de error están asociadas a las llamadas ***distancias L-r:***

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

El uso de estas funciones puede llevar a una estimación de máxima verosimilitud de los pesos para patrones de entrada arbitrarios, si se elige apropiadamente r. [16]

Si $r = 2$ obtenemos la distancia euclidiana (y E es la suma de errores cuadráticos).

Algunas propiedades de esta medida para $r = 1$ (llamada ***Norma Manhattan***) son mencionadas más adelante.

---

[17] [1] y [16] las definen de esta forma, [4] como $\widehat{E}(\mathbf{w}) = \frac{1}{r}E(\mathbf{w})$

Cuando $r \to \infty$ la distancia asociada L-r es $L^{\infty} = \underset{i=1,\ldots m}{Max} \left|(d_i)_j - (y_i)_j\right|$ (Ver [16])

Un r tal que $1 \le r < 2$ da menos peso a las desviaciones (d-y) grandes (ya que la diferencia $|d_i - y_i|$ se eleva a una potencia entre 0 y 1).

Para las funciones de error de Minkowsky la modificación de los pesos es [1]:

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

(capa de salida)

y

$$\Delta w_{ji} = \rho_h \sum_{k=1}^{n} \left[\sum_{l=1}^{L} signo[(d_l)_k - (y_l)_k] \left|(d_l)_k - (y_l)_k\right|^{r-1} w_{lj} f_o'[(Net_l)_k] \right] f_h'[(Net_j)_k](x_i)_k$$

(capa oculta)

Desde el punto de vista estadístico, el caso r = 1 equivale a minimizar la mediana condicional del error mientras que si r = 2 corresponde a la minimización de la media condicional del error de obtener las salidas de la red respecto a las deseadas condicionadas a las entradas dadas [1].

Un caso especial: el aprendizaje según la norma Manhattan

El entrenamiento de la red (incluido el de Langevin, Ver AII.1), puede plantearse en forma genérica como

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \rho(n)\mathcal{H}\left[\mathbf{w}(n), \mathbf{x}(n)\right] \qquad \textbf{(AI.3)}$$

donde $\rho(n)$ es la tasa de aprendizaje y $\mathcal{H}$ representa la regla de aprendizaje.
Para cada valor de la entrada $\mathbf{x}(n)$, valores que se suponen aleatorios e iid, se va a obtener un nuevo $\mathbf{w}(n)$ a partir de cierto $\mathbf{w}(0)$ inicial y la aplicación de AI.3. Esa secuencia de valores de $\mathbf{w} = \{w_{ij}\}$ es una cadena de Markov cuyos estados tienen probabilidades tales que [40]:

$$P(w_{ij}, n+1) - P(w_{ij}, n) = \int \left[P(w'_{ij}, n) W(w_{ij} \left| w'_{ij}) - P(w_{ij}, n) W(w'_{ij} \right| w_{ij}) \right] dw' \qquad \textbf{(AI.4)}$$

integrándose en todo el dominio de variación de w y siendo W las probabilidades de transición en un paso:

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

donde $\langle \ldots \rangle_{\mathbf{x}}$ significa el promedio respecto (o sobre todos los valores) de $\mathbf{x}$ y δ la función delta de Kronecker**.**

La ecuación AI.4 no tiene una solución teórica exacta, y por ello se suelen realizar aproximaciones. La ecuación puede plantear como un desarrollo en serie de potencias de ρ , obteniéndose el desarrollo de Kramers-Moyal [33]:

$$P(w_{ij}, n+1) - P(w_{ij}, n) = \sum_{n=1}^{\infty} \frac{(-1)^n}{n!} \left(\frac{\partial}{\partial w_{ij}} \right)^n \left[a_n(w_{ij}) P(w_{ij}, t) \right]$$

$$a_n(w_{ij}) = \rho^n \left\langle \mathcal{H}^n(w_{ij}, \mathbf{x}) \right\rangle_{\mathbf{x}} \qquad \textbf{(AI.5)}$$

Adicionalmente, si solo tomamos como aproximación los dos primeros términos de la suma de (AI.5), o sea para n=1 y n=2, obtenemos la ecuación (aproximación) de Fokker-Plank [40].

En el caso de la regla de Manhattan el desarrollo de Kramers-Moyal puede ser sumado exactamente, o lo que equivalente, la ecuación (1) tiene solución exacta [33], y puede ser calculada como una sumatoria de un número finito de términos (distintos de los del desarrollo de Kramers-Moyal).

En ese caso, obtendríamos

$$\Delta w_{ij} = \mathcal{H}_{ij}(\mathrm{w}, \mathbf{x}) = -signo\left[\frac{\partial E(\mathrm{w}, \mathbf{x})}{\partial w_{ij}}\right]$$

donde $\frac{\partial E(\mathrm{w}, \mathbf{x})}{\partial w_{ij}}$ representa la componente correspondiente al peso ij del gradiente instantáneo (o sea, para el patrón x) de E.

La entropía relativa

La entropía relativa (o distancia de Kullback-Leibler) puede ser utilizada como función de error (Ver II.3.2).

ANEXO II: Técnicas de búsqueda del óptimo global

Se describe a continuación una técnica que puede ser usada junto con el algoritmo de "back-propagation" para alcanzar el mínimo global de la función de error. Otra técnica común es el uso de AGs, como se verá en AII.2. Finalmente, algunos realizan búsquedas a partir del mínimo de una función auxiliar (suavizada) en un proceso llamado "tunnelling" (ver [22]).

AII.1 El aprendizaje de Langevin

Los métodos de descenso aleatorio en el gradiente utilizan un ruido para perturbar la función $E(\mathbf{w})$ a minimizar de forma de evitar ser atrapados en un mínimo local que lleve a una solución "mala", y de esa forma obtener una solución global "buena". Durante el proceso de búsqueda las perturbaciones de E son eliminadas gradualmente de forma de que la función que esta siendo minimizada sea efectivamente $E(\mathbf{w})$ al alcanzar la solución final.

En la regla de aprendizaje de Langevin, lo que se hace es añadir ruido a los pesos durante el entrenamiento, lo cual es equivalente a realizar una búsqueda descendente según el gradiente de una función perturbada $\widetilde{E}(\mathbf{w},\mathbf{N})$:

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

con $E(\mathbf{w})$ función de error a minimizar, $\mathbf{N} = \left[N_1, N_2 .... N_n\right]^T$ es un vector de ruidos blancos independientes y $c(t)$ es un parámetro que controla la magnitud del ruido [60]. El ruido añadido debe ser reducido en forma gradual, de forma que $\lim_{t\to\infty} c(t) = 0$. Por ejemplo, podría ser $c(t) = \beta e^{-\alpha t}$ con $\beta \neq 0$ y $\alpha > 0$. El gradiente de la función perturbada es $\nabla\widetilde{E}(\mathbf{w},\mathbf{N}) = \nabla E(\mathbf{w}) + c(t)\mathbf{N}(t)$, y como, en el caso general, es $\mathbf{w}(t+1) = \mathbf{w}(t) - \rho\nabla g(\mathbf{w})\big|_{\mathbf{w}(t)}$ cuando la función de error a minimizar es g(**w**), entonces, sustituyendo g(**w**) por $\widetilde{E}$ es:

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

o sea

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

Observamos que si el ruido **N** tiene media **0**, entonces, en promedio la búsqueda va a seguir el gradiente de E:

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

donde el subíndice N significa "media sobre los valores de los ruidos".

La probabilidad de que esta búsqueda aleatoria en el gradiente conduzca a la solución mínima global depende en forma crítica de la amplitud del ruido $c(t)$. Por ejemplo, en el caso de $c(t) = \beta e^{-\alpha t}$, el coeficiente β controla la amplitud del ruido y α la tasa de decrecimiento. Es necesario un β suficientemente grande como para que la búsqueda explore una gran parte del espacio de búsqueda. Para un α grande, los efectos aleatorios se atenúan muy rápidamente y la búsqueda se reduce prematuramente a una búsqueda determinística por el gradiente, lo que aumenta la probabilidad de quedarnos solo con un mínimo local. Los valores pequeños de α son deseables porque permiten que la búsqueda explore un número de puntos lo suficientemente grande de la superficie de búsqueda, lo cual es necesario para una optimización global. Sin embargo, un valor de α demasiado pequeño se traducirá en una convergencia muy lenta. Por lo tanto, α debe ser elegido de forma que se alcance un balance entre una velocidad de convergencia adecuada y la obtención de una solución óptima (o casi óptima).

Esta búsqueda estocástica en el gradiente puede ser aplicada a todas las reglas de aprendizaje basadas en el descenso según el gradiente, para redes "feedforward", y cuando los pesos se actualizan según ella, se suele llamar ***regla de aprendizaje de Langevin***. En el caso de "back-propagation", la retro-propagación tipo Langevin nos llevaría a

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

para las neuronas de salida y ocultas respectivamente (los subíndices o y h para c están implicando que se podrían usar diferentes magnitudes de ruido para las neuronas ocultas y para las de salida)[18].

El entrenamiento según la regla de Langevin suele ser computacionalmente más efectivo que un "back-propagation" determinístico y escapar a los mínimos locales (Hoptroff y Hall, citados por [4]), dando mejores resultados tanto en velocidad de convergencia como en calidad de la solución que otros métodos de mayor orden cuando la matriz hessiana es "ill-condioned", es decir, cuando el cociente del menor valor propio de $\mathbf{H}$ sobre el mayor es muy cercano a 0 (o el equivalente a la exactitud de punto flotante simple precisión de la computadora donde se realiza el proceso). Esto es importante, ya que las redes "feedforward" con capas ocultas tienden a tener esta característica [60].

Finalmente, es de notar que la versión incremental del "back-propagation" también puede ser vista como una búsqueda aleatoria en el gradiente, aunque la aleatoriedad en ese caso es intrínseca al gradiente en sí, debido a la naturaleza de la minimización de un error "instantáneo" (a la presentación al azar de los vectores de entrenamiento, como opuesta a una aleatoriedad introducida artificialmente, como lo hicimos aquí. El ruido que introdujimos aquí es homogéneo, en el sentido de que es el mismo en cada mínimo de E, mientras que en el BP incremental es no-homogéneo, ya que se relaciona con las fluctuaciones intrínsecas debidas a la aleatoriedad de los patrones presentados. En BP incremental, cuanto mayor es el error, más hay que aprender, mayores son las fluctuaciones de los pesos, mayor es el ruido y mayor es la posibilidad de escapar de los mínimos locales. Este ruido no-homogéneo le da una ventaja sobre el aprendizaje de Langevin en cuanto a escapar de mínimos locales, lo cual ha sido probado en simulaciones por Heskes y Kappen (citados por [4]). Sin embargo, el BP incremental no logra escapar del problema de los flat spots, lo que sí hace el método de Langevin.

AII.2 Métodos de entrenamiento no basados en derivadas

Se exponen aquí varios métodos de entrenamiento no basados en las derivadas de la función de salida, tales como los Algoritmos Genéticos combinados con las Redes Neuronales. En especial, se describe un método híbrido que intenta aunar las ventajas de los AGs y de las búsquedas por descenso en el gradiente.

AII.2.1 Redes neuronales artificiales evolutivas

Las redes neuronales evolutivas son un tipo especial de redes neuronales en el que la evolución es otra forma de adaptación aparte del aprendizaje. Esta evolución es a menudo simulada por algoritmos genéticos u otros algoritmos evolutivos[19]. Algunas de las aplicaciones de la evolución son el entrenamiento en sí (adaptación de los pesos), la selección de los pesos iniciales, el diseño de la arquitectura (estructura de la red y funciones de transferencia), aprendizaje de la regla de aprendizaje, etc. [60].

[18] Con la convención habitual, cuando $\mathbf{w}$ se considera vector se toma $\mathbf{N}$ como un vector de igual dimensión; cuando se considera $\mathbf{w} = \{w_{ij}\}$, $\mathbf{N}$ tendrá la misma estructura $\mathbf{N} = \{N_{ij}\}$

[19] Por algoritmos evolutivos entendemos los algoritmos genéticos, la programación evolutiva y las estrategias de evolución [71].

Procedimientos evolutivos de búsqueda

Los algoritmos evolutivos se basan en hacer evolucionar una población de individuos que compiten e intercambian información entre sí, siendo especialmente buenos al manejar funciones definidas en espacios de alta dimensionalidad que contienen varios óptimos locales [60]. En especial, los algoritmos genéticos son una clase de esos algoritmos donde cada uno está definido por su método de codificar los individuos, su mecanismo de selección y sus operadores genéticos.

La estructura general de un algoritmo genético puede ser esquematizada como:

1 – Generar la población inicial G(0) al azar y hacer $i = 0$.

2 – ***Repetir***

- *Evaluar cada individuo en la población*
- *Elegir progenitores dentro de G(i) basándose en su función de fitness en G(i)*
- *Aplicar los operadores genéticos a los progenitores y usar los resultados para formar G(i+1)*
- *Hacer* $i=i+1$

hasta que *se satisfaga el criterio de terminación*

Evolución en las redes neuronales evolutivas

La evolución en estas redes se puede dar, por decirlo a grandes rasgos, en tres niveles: la evolución de los pesos, la evolución de las arquitecturas y la evolución de las tasas de aprendizaje. La evolución de los pesos introduce un enfoque adaptativo del entrenamiento, especialmente en el aprendizaje de redes recurrentes y por refuerzo ("reinforcement") [20] donde los algoritmos por descenso en el gradiente suelen presentar dificultades [60]

La evolución de los pesos

Si bien BP ha tenido una aplicación exitosa en muchos casos, al ser un algoritmo basado en el descenso en el gradiente tiene algunos defectos debido a ello: a menudo conduce a soluciones que son mínimos locales y es muy indeficiente en encontrar el mínimo global si la función de error es multimodal; adicionalmente, exige trabajar con funciones de error diferenciables. Una forma de evitar estos problemas es plantear el entrenamiento de los pesos usando una red neuronal evolutiva. Se podría entonces emplear procedimientos de búsqueda global tales como los AGs para hallar una solución cercana a la óptima en el espacio de pesos. La función de "fitness" para la red evolutiva puede ser definida de acuerdo a las distintas necesidades. En ellas se puede incluir, además de la diferencia entre la salida real y la deseada, la complejidad de la red (o sea, algún término de regularización). Por otra parte, la función de "fitness" no necesita ser diferenciable ya que los AGs no dependen de la información del gradiente.

El planteo del entrenamiento como una evolución de los pesos tiene dos etapas:

a) ***decidir la representación de los genotipos (individuos) de los pesos de conexión (por ejemplo, si se van a representar como cadenas ("strings") binaria(o)s o no) y***

b) ***la evolución en sí, simulada por AGs u otros algoritmos evolutivos.***

Diferentes esquemas de representación y operadores genéticos pueden conducir a performances de entrenamiento muy distintas.

[20] El aprendizaje por refuerzo es un tipo especial de aprendizaje no supervisado donde la salida exacta deseada es desconocida, y esta basado solo en la información de si la salida real es correcta o no [71]

Un ciclo típico en la evolución de los pesos de conexión podría ser:

1. *Codificar cada genotipo (individuo) en la generación actual como un conjunto de pesos y construir la correspondiente red con esos pesos.*

2. *Evaluar cada una de esas redes construidas calculando su error cuadrátco medio entre las salidas reales y las deseadas[21]. El "fitness" de cada individuo esta determinado por el error: a mayor error, menor fitness.*

3. *Reproducir un número de hijos para cada individuo en la generación actual de acuerdo a su "fitness"*

4. *Aplicar operadores genéticos tales como el de cruzamiento ("crossover") y el de mutación a cada individuo hijo generado en el paso 3 y obtener una nueva generación.*

En cuanto a la representación de los pesos, una posible es la binaria: cada peso es representado por un número binario de cierta longitud y la red esta representada por la concatenación de todos los pesos de la red. Por ejemplo, la red

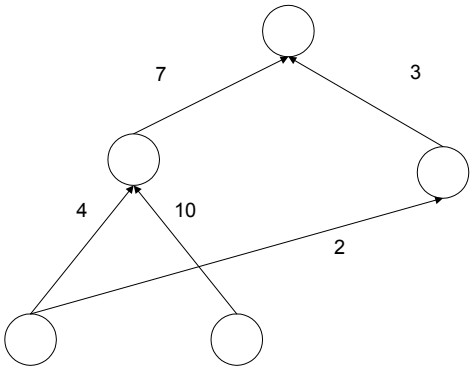


Figura AII.1

se podría representar como 0100 1010 0010 0000 0111 0011. Otra forma de representar la red sería mediante un conjunto de números reales que representarían sus pesos. Para la red anterior, se escribiría entonces: (4,10,2,0,7,3).
Cada una de las representaciones tiene sus operadores genéticos por ejemplo, en la de números reales no se pueden aplicar los operadores estándar que trabajan sobre cadenas binarias. En ese caso, se suele sustituir la mutación (como permutación de bits) por la adición de un número aleatorio al peso correspondiente.

La evolución de la arquitectura y de las reglas de aprendizaje

El diseño de la arquitectura óptima para una red evolutiva puede ser formulado como un problema de búsqueda en el espacio de las arquitecturas donde cada punto representa una arquitectura. Dado cierto criterio de "performance" (por ejemplo, entrenamiento más rápido, menor complejidad, etc.), se obtiene una superficie donde cada punto de ella corresponde a la "performance" de una cierta arquitectura; el diseño buscado será el punto más "alto" (con mejor valor del criterio de "performance") en esa superficie.

El uso de la evolución para determinar la arquitectura óptima para la red se basa en algunos trabajos que demuestran que la búsqueda de dicha estructura utilizando algoritmos evolutivos da mejores resultados que los algoritmos de crecimiento ("growing") y podado ("pruning"), debido a las características del espacio de las arquitecturas:

- la superficie es infinitamente grande, ya que el número de neuronas y conexiones posibles no es acotada
- la superficie es no diferenciable, ya que los cambios en el número de nodos o conexiones son discretos y pueden tener un efecto discontinuo en la performance de la red

[21] o usar otra función de error

- el "mapeo" entre una arquitectura y su performance es indirecto y dependiente del método de evaluación usado
- arquitecturas similares pueden tener performances muy diferentes
- distintas arquitecturas pueden tener performances parecidas.

Junto con la topología de la red se pueden hacer evolucionar las funciones de transferencia usadas. También se pueden hacer evolucionar la regla de aprendizaje. Por ejemplo, White y Ligomenides (citados por [60]) hicieron evolucionar en una red la regla de aprendizaje, y luego de 1000 generaciones, partiendo de una población de reglas al azar, llegaron a la regla delta.

Comparación entre el entrenamiento evolutivo y el basado en el gradiente

El entrenamiento evolutivo es atractivo cuando la información del gradiente no esta disponible o es muy costosa de conseguir. Este entrenamiento se ha usado en aprendizaje de redes recurrentes y aprendizaje por refuerzo. Es interesante que el mismo algoritmo evolutivo se pueda usar para distintas redes sin importar si son recurrentes o redes de otro tipo, lo que ahorra mucho esfuerzo humano en desarrollar algoritmos de entrenamiento para distintos tipos de redes. Se pueden agregar a la función de "fitness" términos de regularización u otras restricciones, tales como "weight sharing", sin importar ni siquiera es continua.

El entrenamiento evolutivo es en general más lento que las variantes rápidas de "back-propagation", pero puede trabajar con redes donde la información del gradiente no está disponible. Las comparaciones de cuales son más rápidos dependen en gran parte de si se esta comparando un AG estándar vs. uno de BP rápido o un BP estándar contra un AG rápido [60].

El entrenamiento híbrido

Uno de los mayores problemas de los algoritmos genéticos es su ineficiencia en la búsqueda del óptimo local aunque son buenos buscando en forma global. La eficiencia del entrenamiento evolutivo puede ser mejorada significativamente incorporando procedimientos de búsqueda local en la evolución, es decir, combinando la búsqueda local de otros algoritmos (por ejemplo, BP). Los resultados experimentales prueban que estas búsquedas híbridas son más eficientes que los AGs o BP por separado, aún teniendo en cuenta el hecho de que se debe entrenar con BP varias veces para encontrar una buena solución debido a su sensibilidad a las condiciones iniciales.

Un caso especial de método híbrido

Hassoum [4] describe un algoritmo híbrido para redes "feedforward" de una capa oculta. Este es un método que se basa en que las redes "feedforward" con una sola capa oculta son aproximadores (y clasificadores) universales: una red de este tipo puede aproximar cualquier función o clasificar cualquier conjunto arbitrario de patrones, siempre que la capa oculta tenga el número suficiente de neuronas (Ver AVIII). Basándonos en ello, dividiremos la red original en dos redes, que trabajan en cascada y lo que se hará es una búsqueda usando AGs en el espacio de los valores posibles de salida para la capa oculta y una búsqueda según el gradiente (regla delta) en las dos redes.

Sea la red de la figura AII.2. Si tuviéramos un conjunto de vectores de salida de la capa oculta $\{\mathbf{h}_1, \mathbf{h}_2, \ldots \mathbf{h}_m\}$ [22] tal que se cumplan los "mapeos" (correspondencias) $\{\mathbf{x}_k\} \rightarrow \{\mathbf{h}_k\}$ y $\{\mathbf{h}\}_k \rightarrow \{\mathbf{d}_k\}$ para k = 1,2,...m, podríamos aplicar una búsqueda basada en el descenso según el gradiente para aprender rápida e independientemente los pesos para tanto la capa de salida como la oculta. Sin embargo, inicialmente no conocemos el conjunto adecuado de valores de salida de la capa oculta $\{\mathbf{h}_k\}$ que resuelve el problema. Por lo tanto, usaremos un algoritmo genético para hacer evolucionar ese conjunto de valores de salida ocultos de manera que $\{\mathbf{x}_k\} \rightarrow \{\mathbf{h}_k\}$ y $\{\mathbf{h}_k\} \rightarrow \{\mathbf{d}_k\}$.

[22] m = número de patrones de entrenamiento disponibles

Codificaremos los valores de salida de la capa oculta como la cadena ("string") $\mathbf{s} = [\mathbf{s}_1, \mathbf{s}_2 \ldots \mathbf{s}_m]$ donde $\mathbf{s}_i$ es un string formado a partir de la representación binaria de la salida $\mathbf{h}_i$. Adicionalmente, cada punto de búsqueda puede ser representado en forma equivalente como una matriz $\mathbf{H} = [\mathbf{h}_1\ \mathbf{h}_2 \cdots \mathbf{h}_m]$ de J x m.

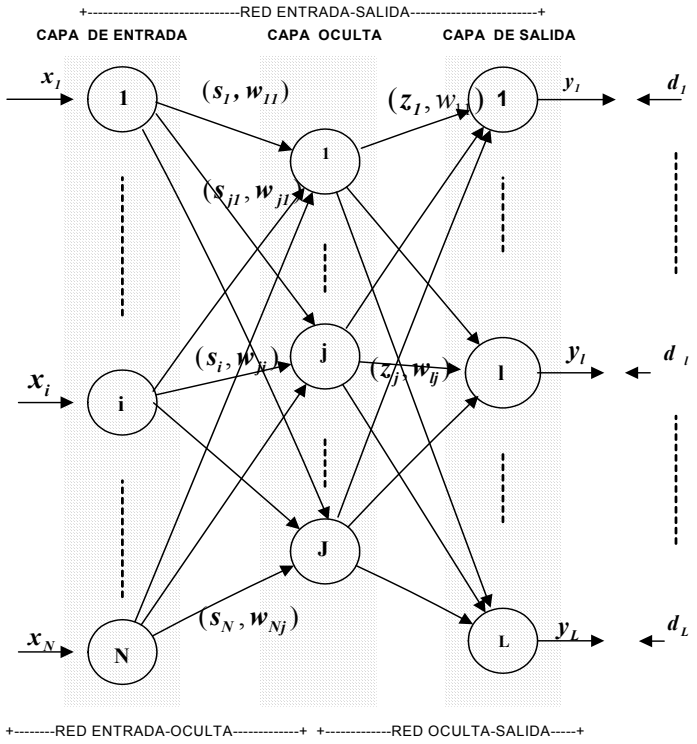


Figura AII.2

Como población inicial de puntos de búsqueda, generamos m matrices binarias al azar $\{\mathbf{H}_j\}\quad j = 1, 2, \ldots m$ La matriz j-ésima tiene asociada una red que tiene la misma topología que la red original, siendo las m redes inicializadas con el mismo conjunto de pesos al azar.

El valor de "fitness" del punto de búsqueda j-ésimo (o sea la matriz $\mathbf{H}_j$) será función de la suma de errores cuadráticos de la salida de la red j-ésima oculta-salida:

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

entendiéndose por $\left(y_l^k\right)_j$ a la salida de la *l*-ésima neurona de salida en la red oculta-salida *j* cuando se presenta el patrón de entrada $\mathbf{x}_k$. La función de "fitness" puede tener varias formas.
Algunas posibilidades son:

$$f(\mathbf{H}_j) = -E_j$$

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

$$f(\mathbf{H}_j) = 1/\left(E_j + \varepsilon\right)$$

con ε un número positivo y pequeño. Hay que tener en cuenta que las distintas funciones llevarán a distintas performances del algoritmo.

El esquema del algoritmo es:

Inicializar:

Comenzando con valores aleatorios de los pesos y de los valores de salida de las neuronas ocultas, se adaptan los pesos en cada una de las m redes respecto al juego de entrenamiento $\{\mathbf{x}_k, \mathbf{h}_k\}$ $k = 1, 2, ...m$ usando la regla delta[23]. Similarmente, los pesos de las conexiones que llegan a las neuronas de la capa de salida son adaptados según el juego de entrenamiento $\{\mathbf{h}_k, \mathbf{d}_k\}$ $k = 1, 2, ...m$, independientemente de las primer red.

Repetir hasta que *(al menos una de las redes entrada-salida generadas tenga un error E_j menor que un valor preespecificado)*[24]

Prueba de " fitness"*:*

Luego de que se actualizan los pesos, se prueba cada red ejecutando los cálculos "feedforward" y se evalúa su "fitness". En estos cálculos, las salidas de la primera red sirven como entradas de la segunda (la que contiene las neuronas de salida).

Evolución:

A continuación, se aplican los operadores genéticos para obtener la siguiente generación de $\{\mathbf{H}_j\}$. Respecto al cruzamiento, los m/2 $\mathbf{H}_j$ con los mayores valores de "fitness" se duplican y se mantienen temporalmente para cruzamiento. Se realizan cruzamientos con probabilidad P_c (cercanas a 1): se elige aleatoriamente un par $\{\mathbf{H}_i, \mathbf{H}_j\}$ sin reemplazo del conjunto temporario mencionado. Si un par $\{\mathbf{x}_k, \mathbf{d}_k\}$ de entrenamiento es aprendido pobremente por la red i durante la fase de entrenamiento anterior, o sea, si el error de salida debido a este par es substancialmente mayor que el error promedio de la red i sobre el conjunto completo de entrenamiento, entonces la columna correspondiente $\mathbf{h}_k$ de $\mathbf{H}_i$ es reemplazada por la k-ésima columna de $\mathbf{H}_j$. Los cruzamientos pueden afectar varios pares de columnas en las matrices $\mathbf{H}$.

Fin repetir

AII.2.2 Otros métodos no basados en derivadas

La bibliografía asociada a este punto es [54].

[23] Se pueden utilizar otras reglas de aprendizaje.

[24] En [4] se proponen condiciones adicionales de detención para evitar caer en "loops" infinitos.

ALOPEX. Este algoritmo actualiza los pesos de la red mediante pequeñas perturbaciones de sus valores, en uno u otro sentido según la correlación entre los sentidos de las perturbaciones recientes y el cambio en el error cometido sobre todo el conjunto de entrenamiento. Este algoritmo presenta el interés de ser independiente de la topología de la red y de la función de error usada, por lo que puede ser usado además para cualquier otra tarea de optimización distinta a las redes neuronales.

Algoritmo de Cauwenberghs. Aquí se sigue una regla de aprendizaje parecida a la del Alopex. Se suma una perturbación aleatoria **z** al vector de pesos actual **w** y se calcula el error resultante $E(\mathbf{z}+\mathbf{w})$. Este nuevo error se utiliza para actualizar el vector de pesos como:

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

donde α es la tasa de aprendizaje.

ANEXO III: Redes dinámicas

En este anexo presentaremos las redes cuyas salidas no solo dependen de sus entradas "instantáneas" así como los algoritmos de entrenamiento asociados.

AIII.1 Entrenamiento

Una ***red dinámica*** es una red tal que se comporta como un sistema dinámico respecto a sus entradas y salidas. Veremos cómo aplicar esta definición directamente al entrenamiento de la red para la predicción.

Dados los valores observados de la salida **y** de un sistema dinámico obtenidos en tiempos discretos anteriores a *t,* trataremos de predecir exactamente **y**(*t*+*p*), donde $p > 0$. Si bien cuando p aumente la calidad del valor predicho se degradará para cualquier método predictivo, trataremos de mantener la exactitud de la predicción para un amplio rango de valores de *p*. Dada que esta predicción es realizada por una red neuronal de entrada **x** y salida **y**, como el sistema modelado y la red son dinámicos podríamos anotar:

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

con $n \geq m$ y g una función no lineal [4]. Para entrenar la red podemos utilizar "back-propagation" a partir de los pares "estáticos"

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

Si el entrenamiento es exitoso, se esperaría que la salida **y**(*t*+1) obtenida se aproxime a la real, **d**(*t*+1). A esta técnica se la llama ***"windowing"***. Además de esta forma de entrenar una red dinámica existen otros métodos, como se verá a continuación.

AIII.1.2 "Back-propagation" a través del tiempo (BPTT) y RTRL

Desdoblado de la red en el tiempo

Sea la siguiente red de ejemplo:

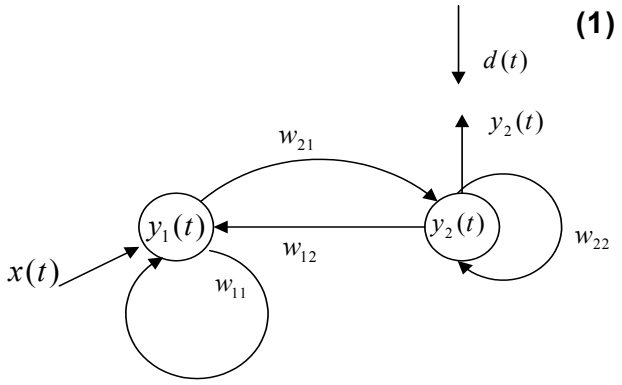


Figura AIII.1

donde $d(t)$ es el valor de la salida deseada, $y_2(t)$ es el de salida obtenido y $x(t)$ la entrada a la red, todo ello en el instante *t*.

Una red que se comporta idénticamente a esta para t = 1, 2, 3, 4 es la que se obtiene al desdoblar la red en el tiempo, de forma que solo nos queden capas "feedforward":

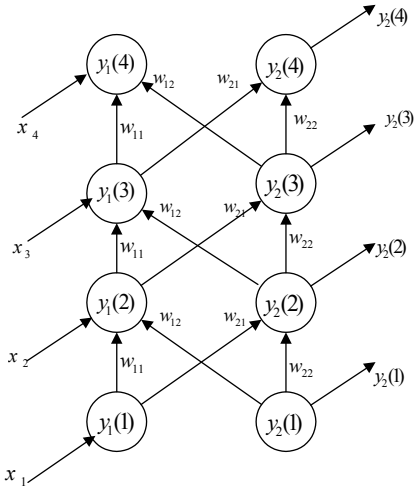


Figura AIII.2

En forma general, el número de capas resultantes es igual al intervalo de tiempo que se desdobla (expande), digamos *T*. Esta idea es aplicable cuando *T* es pequeño ya que todas las neuronas de la red son replicadas *T* veces y por lo tanto limita el largo máximo de las secuencias que pueden ser procesadas. Nótese que las conexiones w_{ij} de la neurona j a la i son idénticas en todas las capas de la red expandida. La adaptación del algoritmo de BP para entrenar la red desdoblada es lo que se conoce como "***back-propagation" a través del tiempo*** (o "back-propagation through time", ***BPTT***) [5].

El entrenamiento de la red desdoblada puede hacerse por épocas o en forma incremental, lo que conduce a dos implementaciones del BP temporal, correspondiendo una al entrenamiento por épocas y la otra al entrenamiento en tiempo real.

Entrenamiento por épocas de la red desdoblada

Dividamos los datos de entrenamiento en épocas independientes donde cada época representa un patrón (temporal) de entrenamiento[25]. Sea n_0 el instante de comienzo de una época y sea n_1 el instante final. Dada esta época, podemos definir la función de error como

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

donde A es el conjunto de todos los índices *j* correspondientes a aquellas neuronas en la red para las cuales esta especificada la salida deseada y $e_j(n)$ es el error en la salida de esas neuronas medida con respecto a la respuesta deseada. Deseamos calcular ∇E, es decir, las derivadas parciales de la función E respecto a los pesos **w** de la red. Para ello, utilizaremos el algoritmo siguiente que se basa en el BP "batch":

1- Pasada hacia delante ("forward"): se presentan todos los datos de entrada del intervalo (n_0, n_1) a la red y se guardan (almacenan) los datos presentados, salidas deseadas y pesos de la red

[25] Esta acepción de época es local a este tema y difiere de la que generalmente se usa al hablar de entrenamiento de MLPs.

2- Pasada hacia atrás ("backward"): se realiza esta pasada sobre los datos almacenados en el paso 1) para calcular los gradientes locales:

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

Esto se puede calcular como [5]:

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

donde $f'(\bullet)$ es la derivada de la función de salida con respecto a su argumento y $e_j(n) = y_j(n) - d_j(n)$ significando $y_j(n)$ la componente j-ésima de la salida en el instante n. Se supone que todas las neuronas de la red tienen la misma función de salida. La ecuación AIII.1 se usa repetidamente empezando por el instante n_1 y yendo hacia atrás, paso a paso, hasta el instante n_0, el número de pasos indicados aquí es igual al número de incrementos de tiempo ("time steps") correspondientes a la época.

3- Una vez que se ha realizado el cálculo de la BP hasta el instante $n_0 + 1$, se aplica el siguiente ajuste a los pesos de la neurona j:

$$\Delta w_{ji} = -\rho \frac{\partial E(n_0, n_1)}{\partial w_{ji}} = \sum_{n=n_0+1}^{n_1} \Delta w_{ji}(n-1) = \rho \sum_{n=n_0+1}^{n_1} \delta_j(n) x_i(n-1)$$

donde $x_i(n-1)$ es la entrada aplicada a la sinapsis i-ésima de la neurona *j* en el instante n-1 [5].

Al comparar este procedimiento con el modo "batch" de BP común vemos que la diferencia básica entre ellas es que las respuestas deseadas están especificadas para las neuronas en varios niveles de la red ya que la capa de salida real esta replicada varias veces al desdoblar la red.

El término de BPTT proviene de que no solo retropropagamos el cálculo desde la salida hacia la entrada, sino que además lo hacemos en el tiempo (del instante final hacia el inicial).

Vemos que este algoritmo no es local en el tiempo, aunque sí en la topología: en el cálculo de δ interviene más de un valor del índice temporal, pero no sus valores en otras neuronas de la red.

A los efectos prácticos se suelen utilizar versiones truncadas del BPTT tal como la siguiente:

Back propagation truncada:

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

- Se utiliza como error el error instantáneo
- Los ajustes a los pesos se realizan teniendo en cuenta $-\nabla E$ en el instante n. Las actualizaciones se hacen "on line", mientras la red trabaja.

- El nombre de truncada proviene de que se almacenan las entradas, pesos y salidas deseadas (paso 1 del algoritmo anterior) para un número fijo de instantes de tiempo de cada época. A ese número fijo se lo llama ***profundidad de truncamiento.*** Toda la información más antigua que esa profundidad es ignorada. Este truncamiento se realiza ya que sino el tiempo necesario para hacer los cálculos y los requerimientos de almacenamiento podrían crecer linealmente con el tiempo hasta hacer que el proceso se volviese impracticable (Ver [5]).

El algoritmo de BPTT no es muy usado debido a su limitación de trabajar con secuencias de tiempo (trayectorias *T*) cortas, por lo que en algunos casos se usan otros tales como el de "back-propagation" recurrente [4].

Entrenamiento de la red desdoblada en tiempo real: el algoritmo RTRL

El método RTRL ("real time recurrent learning") fue propuesto por Williams y Zipser ([68] y citados en [4]) y permite entrenar a la red mientras esta trabaja (de ahí lo de "real time"), aunque tiene el inconveniente de ser computacionalmente muy costoso. De todos modos, este método tiene la generalidad del enfoque de back-propagation a través del tiempo sin sufrir de los problemas asociados al aumento de memoria necesaria al crecer T.

El algoritmo de BPTT se derivó suponiendo que los pesos permanecían fijos durante toda la variación de n en $[n_0, n_1]$. Como lo que pretendemos hacer es un entrenamiento en tiempo real y poder trabajar con intervalos de tiempo no acotados, lo que haremos es plantear una condición menos restrictiva y simplemente incrementaremos cada peso en $\Delta w_{ji}(n-1)$ sin acumular las variaciones para realizarlas todas juntas posteriormente. Una desventaja potencial de este procedimiento es que no sigue más el gradiente negativo del error total a lo largo de la trayectoria. Sin embargo, es totalmente análogo a lo que sucede con el método de retro-propagación incremental respecto al de retro-propagación "batch": mientras que el algoritmo resultante no puede ya garantizar que se siga el gradiente negativo del error total, las diferencias prácticas son en general pequeñas, con las dos versiones volviéndose casi idénticas cuando la tasa de aprendizaje es pequeña [68]. Se obtiene así el algoritmo de entrenamiento en tiempo real ("real-time") RTRL.

AIII.1.3 Otras variaciones del algoritmo de "back-propagation"

Pasamos a describir dos variaciones del algoritmo de "back-propagation" en las que el tiempo varía en forma continua.

"Back-propagation" recurrente [4]

Este algoritmo es usado para redes totalmente recurrentes en las cuales el tiempo varía en forma continua y permite entrenar a una red recurrente para aprender asociaciones estáticas E → S. Sea una red de N neuronas con salidas, y_i pesos w_{ij} y función de salida $f(Net_i)$. Definimos que una neurona i-ésima es de entrada si recibe un elemento x_i^k del patrón de entrada $\mathbf{x}_k$. También diremos que las neuronas que no son de entrada tiene asociado un $x_i^k = 0$. Las neuronas de salida son aquellas que tienen valores deseados de salida d_i^k. En general, una neurona puede pertenecer al conjunto de las de entrada y al de las de salida a la vez, o puede ser oculta en el sentido de que no es ni de entrada ni de salida.

En adelante, omitiremos el superíndice k indicativo del patrón para simplificar la notación.

Sabemos que cuando la red se halla en equilibrio, es decir, sin variaciones en la salida $\mathbf{y}^*$ si se mantiene la entrada constante, se cumple que [4]:

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

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

Supongamos que la red ha convergido a un estado de equilibrio $\mathbf{y}^*$ en respuesta a una entrada **x.** Por lo tanto, si la neurona *i* es de salida y responde con y_i^*, mientras que se esperaba obtener una salida d_i, se tiene un error E_i

Buscaremos ajustar los pesos de forma de minimizar

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

siendo $E_i^* = 0$ si la neurona i-ésima no es de salida.
La actualización de los pesos será

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

Se puede demostrar que [4]

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

donde el elemento *ij* de la matriz **L** es $\mathbf{L}_{ij} = \delta_{ij} - f'(Net_i^*) w_{ij}$ y δ_{ij} es la función delta de Kronecker (1 si $i = j$, 0 en otro caso).

Como **L** es *N*x*N* y su inversión requiere $\boldsymbol{O}(N^3)$ operaciones, se utiliza un método indirecto para el cálculo de las variaciones de los pesos. La justificación del algoritmo puede encontrarse en [4].

El algoritmo de aprendizaje del "back-propagation" recurrente sería entonces (para simplificar la notación omitimos los superíndices *k* que denotan el patrón):

1. *Se toma un patrón de entrada* ***x*** *y se lo presenta a la red*[26]*. Se halla una solución* $\mathbf{y}^*$ *resolviendo iterativamente las ecuaciones*

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

2. *Las salidas estacionarias obtenidas,* y^**, se utilizan para calcular*

$$E_i^* = y_i^* - d_i$$

3. *Se calculan los* z_i^* *solución de las ecuaciones*

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

4. *Se ajustan los pesos usando la fórmula*

[26] A la red original, no a la adjunta.

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

5. *Se toma otro patrón de entrada y se repiten los pasos anteriores, hasta que se obtenga el error deseado.*

Observaciones:

- Se puede demostrar que el algoritmo de retro-propagación incremental es un caso especial de este, para una red "feedforward" .

- La deducción del algoritmos se hizo suponiendo que el estado de equilibrio **y*** existe. Simard y otros (citados por [4]) demostraron que para cualquier red recurrente, siempre existen casos en los que no se llega a dicha estabilidad. Sin embargo, en la práctica alcanza con elegir lo suficientemente pequeños los pesos iniciales para que la red converja casi siempre a un estado de equilibrio (es decir, para que exista **y***).

- En vez de resolver el sistema de ecuaciones del Paso 3, otra forma de calcular $\mathbf{z}^{*} = [z_1^{*}, \ldots z_N^{*}]$ sería mediante una red recurrente topológicamente similar a la original llamada ***red adjunta***, en la que se sustituyen los pesos w_{ij} por $f'(Net_i^{*}) w_{ij}$, las funciones de salida de las neuronas son lineales, se toman todas las entradas como 0 y a E_i^{*} como entrada de la neurona i-ésima de salida de la red original. Por ejemplo, para la red de la figura AIII.1 sería

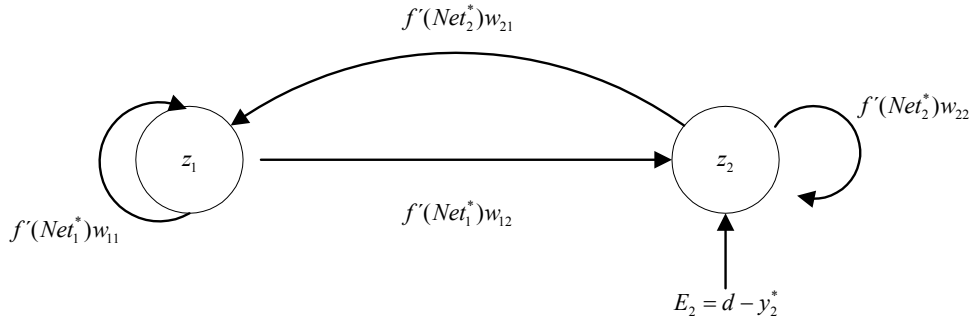


Figura AIII.3

En esta red el tiempo también variaría en forma continua..

AIII.2 Los elementos de memoria

Describiremos aquí algunos de los aspectos principales de las estructuras de memoria y de las redes que hacen uso de ellas.

AIII.2.1 Tipos de elementos de memoria

Para explotar la estructura temporal de los datos de entrada, la red debe tener acceso a la dimensión tiempo. Una forma de lograr esto es utilizando estructuras que almacenen el pasado de dichos datos, llamadas ***estructuras de memoria*** (o neuronas de memoria, "short term memories" o memorias a corto plazo). Como contrapartida, se habla de memoria a largo plazo para referirse a la información almacenada en los pesos de la red. Básicamente, una estructura de memoria

transforma una secuencia de muestras en un punto del espacio de reconstrucción (Ver AVII.1.5). Para incorporar estructuras de memoria dentro de la red, usamos neuronas dedicadas a almacenar tanto la historia de los datos de entrada como de salida de ellas. Otra forma de manejar los datos temporales sería emplear la entrada de datos por ventanas ("***windowing***"), es decir, dando ventanas (pequeñas secuencias) de la serie de datos y de las salidas asociadas como entradas a la red (Ver AIII.1.1). Sin embargo, la inclusión de estructuras de memoria tiene ventajas ya que la red puede:

- elegir el tamaño de la ventana temporal que mejor se ajuste a la tarea
- elegir la ponderación de las muestras de datos en la ventana que más disminuya el error de salida
- recibir solo la muestra actual del mundo exterior (de los datos de entrada), tal como lo hace el arquetipo biológico[27] y a partir de ella ir llevando la historia de ellos.

Las redes recurrentes pueden codificar también la información temporal, pero son complejas. Existen topologías menos complejas que las recurrentes tales como las TLFNs que también pueden hacerlo, aunque son may complejas que las estáticas.

En su forma más general el elemento de memoria tiene la estructura de la figura AIII.4:

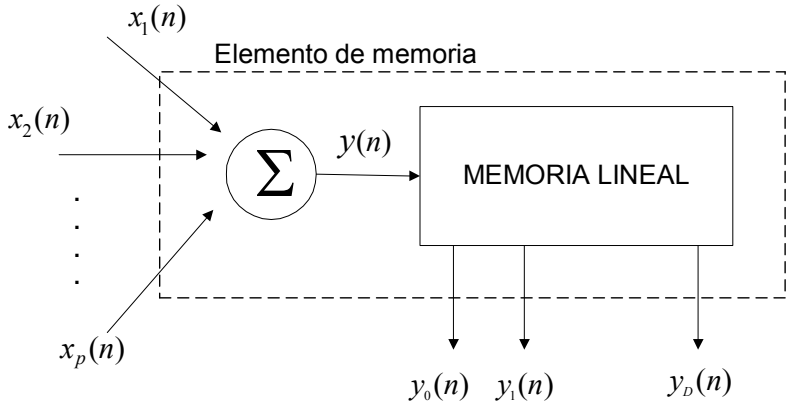


Figura AIII.4

Es decir, esta formado por un nodo que suma las entradas más una memoria lineal, que se describe más adelante. Como se ve, la salida del elemento de memoria genérico es multidimensional. Los diferentes tipos de elementos de memoria se obtienen al utilizar distintos tipos de memorias lineales. A continuación veremos dos casos particulares de ellos.

[27] Los sistemas biológicos tienen noción del tiempo y aprenden en base a secuencia de observaciones en el tiempo que se van dando de a una y no en ventanas.

Las memorias por "delay": la "delay line"

En estos elementos de memoria, la memoria lineal es una la línea de "delays" (***"delay line"***).
La "delay line" se puede esquematizar como:

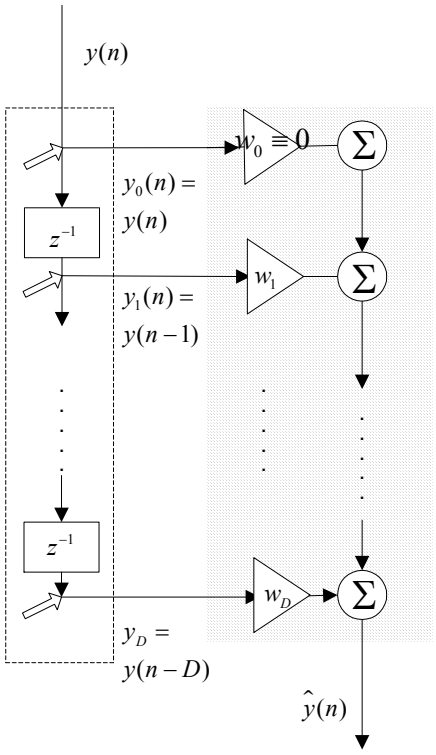


Figura AIII.5

donde z^{-1} representa la función de retardo[28] y el triángulo la multiplicación de su entrada por el valor escrito en él. Es decir, dadas las entradas $x(n)$ la salida es $\mathrm{y}(n) = [y(n), \cdots y(n-D)]$. A los puntos señalados con flechas grandes (⇗) se los llama ***"taps"***. El significado del área sombreada y de $\hat{y}(n)$ se verá a continuación.

En el instante *n*, la salida de un elemento de memoria de este tipo, con D taps, almacena solo muestras que ocurrieron en el tiempo [*n-D+1, n*), es decir, *D-1* muestras en el pasado. Por lo tanto, este tipo de memoria implementa una ventana que incluye D muestras. Se dice entonces que la ***profundidad de la memoria*** es D (Ver más adelante "Profundidad y resolución de la memoria"). Este tipo de memoria es la que se emplea en las TDNNs.

[28] En realidad, lo que se suele indicar dentro del bloque es la transformada z de la función que aplica el bloque a la entrada para obtener la salida. En este caso, z^{-1} es la transformada z del impulso unitario.

Memorias por "feed back"

Estas memorias son obtenidas mediante memorias lineales de la forma indicada por la figura AIII.6:

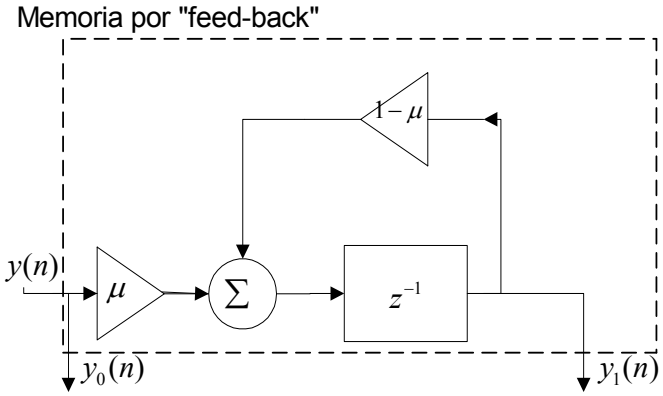


Figura AIII.6

A los elementos de memoria así obtenidos se los llama ***elementos (o neuronas) de contexto*** y son por ejemplo los usados en las redes de Jordan/Elman.

Trazas de memoria

Vemos que en una memoria por "feed back" las muestras pasadas no se preservan exactamente, ya que las salidas de la memoria son la suma de las entradas actual y una versión ponderada de la salida pasada:

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

por lo que la información del pasado se distorsiona progresivamente, y es por ello que se habla de ***traza de memoria***: una versión modificada de los datos de entrada. La idea de traza se generaliza a otros tipos de memorias, donde las muestras anteriores solo se almacenan en forma modificada. En el caso de memorias por "feed back" la traza es $y_1(n)$. Se puede demostrar además que en ese caso, la proyección de la entrada $x(n)$ sobre el espacio de trazas es [11]:

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

Observamos que la memoria de contexto tiene una sola traza $y_1(n)$ mientras que la "delay line" tiene *D:* $y_1(n),\ldots y_D(n)$ que se corresponden con las salidas $y(n-1), y(n-2),\ldots y(n-D)$.

Generalización: el elemento "feedforward" de memoria generalizado.

Las memorias anteriores pueden generalizarse a una memoria (***elemento "feedforward" de memoria generalizado***) el cual utiliza una memoria lineal. Esta memoria lineal es tal que las trazas de memoria $y_k(n)$ se calculan recursivamente a partir de la traza anterior $y_{k-1}(n)$. Se puede demostrar que esa memoria lineal es tal que cuando se le aplica una entrada $y(n)$ su salida es

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

donde * es el operador de convolución, $g(n)$ es una función del tiempo que es causal, normalizada e invariante en el tiempo [29], llamada ***núcleo (o "kernel") de memoria*** y k representa el número de ***"tap"***. Según la elección de $g_0(n)$ y g se obtienen distintos tipos de memorias lineales, y por ende, distintos elementos de memoria.

Esquemáticamente:

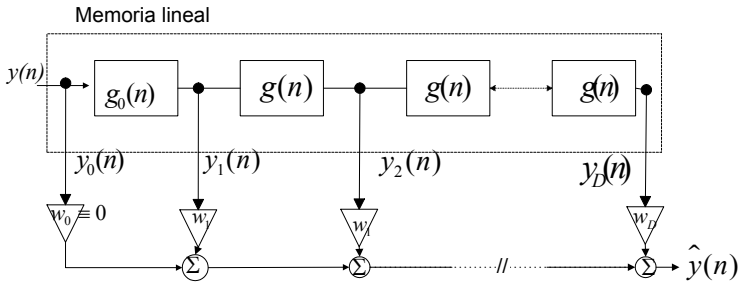


Figura AIII.6

donde el triángulo representa la multiplicación de su entrada por la constante indicada adentro, y análogamente el círculo, la suma de sus entradas. Al número D-1 se lo llama ***orden*** de la memoria. Para evitar confusión, supondremos ahora que la entrada a la memoria lineal es $u(n)$. Sean los vectores $\mathbf{u(n)}=[u(n), u(n-1),...u(n-D)]$ formados a partir de las entradas**.** La proyección de $\mathbf{u}(n)$ sobre el espacio generado por las trazas (llamado ***espacio de trazas de memoria***) es $\hat{\mathbf{u}}(n) = [\hat{u}(n), \hat{u}(n-1), \ldots \hat{u}(n-D)]$ formado igual que $\mathbf{u(n)}$ y tal que $\hat{u}(n) = \sum_{k=0}^{D} w_k y_k(n)$ con $w_0 = 0$. Como se dijo, diferentes selecciones de $g(n)$ *y* $g_0(n)$ proveen distintos modelos de memoria. Por ejemplo,

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

(siendo δ la función delta de Kronecker) da la memoria de "line delay". Para el caso de las memorias por "feed back" es

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

siendo el número de "taps" $D = 2$ y $g_0(n) = \mu(1-\mu)^n$ (Ver [11]).

La selección del "kernel" generador más apropiado para una aplicación dada es un tema todavía en estudio [11].

La memoria gamma tipo I

Cuando $\begin{cases} g(n) = \mu(1-\mu)^n \quad n \geq 1 \\ g_0(n) = \delta(n) \end{cases}$ se obtiene la memoria gamma tipo I. Se puede demostrar que en ese caso,

[29] Se dice que la función $g(n)$ es causal si $g(n) = 0 \quad \forall n < 0$ y que sea normalizada es que $\sum_0^\infty |g(n)| = 1$.

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

siendo $\binom{n}{p} = \frac{n(n-1)\ldots(n-p+1)}{p!}$.

Estas funciones son versiones discretas de los integrandos de la función gamma, de allí su nombre.
Esta memoria gamma es estable (en el sentido de que sus salidas sean valores acotados al transcurrir el tiempo) si $0 < \mu < 2$ (Ver [11]). Esquemáticamente, es

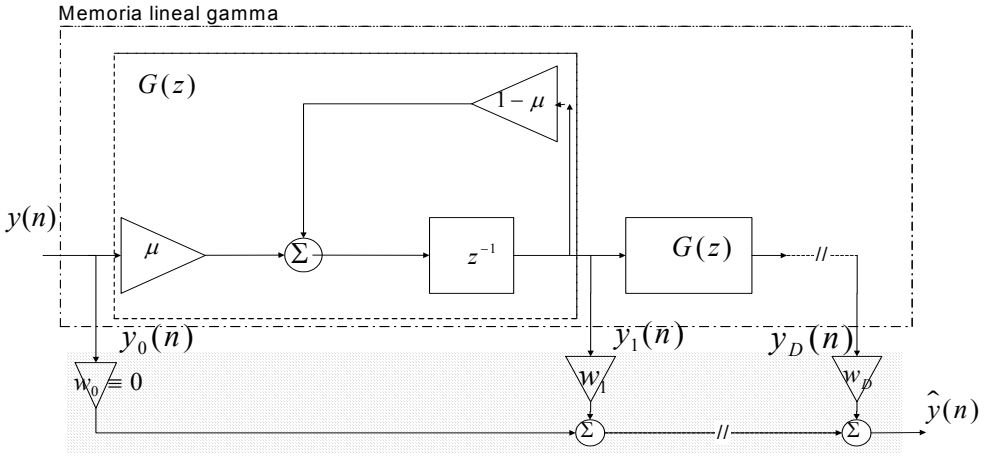


Figura AIII.7

Del esquema se puede observar que la memoria gamma unifica las memorias por "delay" y las de contexto en un solo modelo. Analíticamente es:

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

Algunas propiedades de la memoria gamma

La profundidad M de la memoria gamma (Ver AIII.2.3) es $M = D/\mu$. La memoria gamma tiene la propiedad (que no tiene la "delay line") de separar la profundidad del orden de memoria. Por ejemplo, sea una aplicación dada que requiere una memoria de 100 muestras pero que es tal que con tres pesos (parámetros libres) ya alcanza para modelar las trazas de memoria. Una "delay line" requeriría 100 taps y por lo tanto 100 parámetros (pesos) de los cuales 97 serían nulos, aunque en realidad, debido a los ruidos, todos los pesos van a ser no nulos, lo que producirá una "performance" pobre de la red. En una memoria gamma, alcanza con hacer M=100, D=3 y $\mu = D/M$ =0.03. La memoria puede representar en D taps N muestras del pasado (D<N) [11].

Existe una variación de la memoria gamma, la gamma tipo II que no es descripta aquí pero sí en [11]. Esta memoria tipo I es la usada por el simulador de redes neuronales elegido.

Por otra parte, al implementar en la Sección III una red utilizando este tipo de memorias, el número de "taps" se correspondió con la dimensión del subespacio de la señal.

La memoria de Laguerre

Esta memoria se obtiene tomando

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

La ventaja de esta memoria es que permite un aprendizaje más rápido que la gamma, especialmente cuando μ es cercano a 0 o 2 [11]. En este caso

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

AIII.2.2 Los filtros de memoria

Cada vez que se utilizan elementos de memoria dentro de una red, sus salidas son conectadas a las neuronas (comunes) siguientes utilizando una serie de conexiones con sus correspondientes pesos (que en los esquemas anteriores fueron mostrados en regiones sombreadas y que pueden ser vistos como un ***combinador lineal*** o "linear combiner"). Al conjunto elemento de memoria y combinador lineal se lo llama ***filtro de memoria*** ([11]) y es lo que finalmente se implementa en el simulador.

Una descripción detallada de la relación entre las memorias gamma, de Laguerre y la teoría del filtrado se puede encontrar en [11].

AIII.2.3 Profundidad y resolución de la memoria

Para un elemento de memoria generalizado se define M ***= profundidad de la memoria*** como

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

Se define la ***resolución de memoria*** R como al número de "taps" en la estructura por muestra o unidad de tiempo. Una memoria de poca profundidad solo almacena su contenido por un período relativamente corto, mientras que una de gran profundidad mantiene su contenido durante más tiempo. Una memoria con alta resolución es capaz de almacenar información sobre la secuencia de entrada a un nivel fino mientras que una de poca lo hace a un nivel menos detallado. Se cumple que $R.M = D$ [11]. Por lo tanto, para un número de taps D dado, aumentar la resolución significa disminuir la profundidad. Por ejemplo, para la neurona de contexto, la profundidad es $1/\mu$ y la resolución es μ.

En el cuadro AIII.1 se resumen las distintas profundidades y resoluciones de las memorias descritas:

CUADRO AIII.1

	PROFUNDIDAD	RESOLUCIÓN
"Delay line"	D	1

Contexto	$1/\mu$	μ
Gamma tipo I	D/μ	μ

ANEXO IV: Alternativas para evitar el "gradiente evanescente"

IV.1 El entrenamiento usando los filtros de Kalman

El filtrado consiste en a partir de un conjunto de datos (señal) de entrada, realizar cierta transformación de forma de que se eliminen ciertas características de ellos para obtener una salida deseada. Por lo tanto, hablar de "filtro" será equivalente, en este contexto, a hablar de "algoritmo".

El ***filtro de Kalman lineal*** permite estimar el estado de un sistema dinámico lineal cuyo modelo no es completamente conocido y al que se accede a través de un proceso de medición que posee cierto nivel de ruido. El filtro permite utilizar la información incompleta del modelo para mejorar de forma recursiva la estimación del estado del sistema proporcionada por la medición. Por otra parte, al calcular la predicción en forma recursiva considera las estimaciones en los instantes anteriores. Esto no sucede en los casos de descenso en el gradiente: normalmente, las derivadas de la función de error solo toman en cuenta la distancia entre la salida actual y la correspondiente salida deseada sin usar a la hora de actualizar los pesos ninguna información sobre la historia anterior del entrenamiento.

El filtro de Kalman extendido es una adaptación del lineal que permite trabajar con sistemas no lineales.

El filtro de Kalman extendido desacoplado se basa en el filtro de Kalman extendido para poder manejar la complejidad computacional de los cálculos correspondientes a las redes de cierto tamaño. En todo momento se utiliza toda la información suministrada a la red hasta el instante actual, incluidas todas las derivadas calculadas desde la primera iteración del proceso de aprendizaje. Sin embargo, el algoritmo funciona de tal modo que solo es necesario almacenar explícitamente los resultados de la última iteración (Ver [69] y [21]).

Resumiendo: *el problema del filtrado de Kalman puede plantearse como: usando todos los datos observados, consistentes en un conjunto de vectores* $\mathbf{d}_i \quad i = 1, \ldots n$ *encontrar para cada* $n \geq 1$ *la estimación del estado* $\mathbf{w}_i$ *que minimice el error medio cuadrático entre los valores observados y las salidas de un sistema con ese estado* $\mathbf{w}_i$ ([5]).

Para aplicar el filtrado de Kalman a una red neuronal, lo que se hace es considerarla como un sistema dinámico determinístico no lineal de tiempo discreto cuyos estados están dados por los pesos $\mathbf{w}(n)$.

AIV.1.1 El filtro de Kalman lineal

El filtro de Kalman intenta estimar el estado $\mathbf{w}(n) \in \mathrm{R}^m$ de un sistema dinámico lineal de tiempo discreto en el que se cumple:

$$\mathbf{w}(n+1) = \mathbf{A}(n)\mathbf{w}(n) + \mathbf{B}(n)\mathbf{u}(n) + \boldsymbol{\omega}(n) \qquad \textbf{(AIV.1)}$$

donde $\mathbf{u}(n)$ es la entrada del sistema en el instante n del cual se tiene $\mathbf{d}(n) \in \mathrm{R}^m$ tal que

$$\mathbf{d}(n) = \mathbf{H}(n)\mathbf{w}(n) + \boldsymbol{\upsilon}(n) \qquad \textbf{(AIV.2)}$$

siendo $\mathbf{A}(n)$, $\mathbf{B}(n)$ y $\mathbf{H}(n)$ matrices conocidas y $\boldsymbol{\omega}(n)$ y $\boldsymbol{\upsilon}(n)$ representan el ruido del proceso y de la medición respectivamente. Se asume que se tratan de ruidos blancos de medias nulas y con matrices de covarianza diagonales $\mathbf{Q}(n)$ y $\mathbf{R}(n)$:

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

En cada instante el filtro utiliza la estimación del estado actual y de la covarianza actual para obtener una estimación a priori para el siguiente instante ("proyecta hacia delante en el tiempo" la covarianza y la estimación del estado). Posteriormente, en el instante siguiente, utiliza los resultados de la medición real para mejorar esta estimación y obtener una estimación a posteriori. Este proceso también puede verse como un ciclo de predicción y corrección.

Sea $\hat{\mathbf{w}}^{-}(n)$ la estimación a priori del estado en el instante n a partir del conocimiento anterior al paso n:

$$\hat{\mathbf{w}}^{-}(n) = \mathbf{A}(n)\mathbf{w}(n-1) + \mathbf{B}(n)\mathbf{u}(n-1) \qquad \textbf{(AIV.3)}$$

La estimación a posteriori del estado, $\hat{\mathbf{w}}(n)$ se obtiene como una combinación lineal de la estimación a priori $\hat{\mathbf{w}}^{-}(n)$ y la diferencia ponderada entre la medición real $\mathbf{d}(n)$ y una predicción de la medida , $\mathbf{H}(n)\hat{\mathbf{w}}^{-}(n)$:

$$\hat{\mathbf{w}}(n) = \hat{\mathbf{w}}^{-}(n) + \mathbf{K}(n)\left[\mathbf{d}(n) - \mathbf{H}\hat{\mathbf{w}}^{-}(n)\right] \qquad \textbf{(AIV.4)}$$

A la expresión $\mathbf{d}(n) - \mathbf{H}\hat{\mathbf{w}}^{-}(n)$ se la denomina ***residuo o innovación de la medida*** y refleja la discrepancia entre la medición predicha y la real. $\mathbf{K}$ es la llamada ***matriz de ganancia*** de Kalman.

Consideremos ahora los errores de la estimación a priori y de la estimación a posteriori:

$$\mathbf{e}^{-}(n) = \mathbf{w}(n) - \hat{\mathbf{w}}^{-}(n)$$
$$\mathbf{e}(n) = \mathbf{w}(n) - \hat{\mathbf{w}}(n)$$

Las covarianzas del error de estimación a priori y a posteriori son respectivamente:

$$\mathbf{P}^{-}(n) = \left\langle \mathbf{e}^{-}(n)\mathbf{e}^{-}(n)^{T} \right\rangle$$
$$\mathbf{P}(n) = \left\langle \mathbf{e}(n)\mathbf{e}(n)^{T} \right\rangle$$

La matriz de ganancia **K** se elige de manera de que se minimice la covarianza del error a posteriori. Una posibilidad para ello es

$$\mathbf{K}(n) = \mathbf{P}^{-}(n)\mathbf{H}(n)^{T}\left[\mathbf{H}(n)\mathbf{P}^{-}(n)\mathbf{H}(n) + \mathbf{R}(n)\right]^{-1} \qquad \textbf{(AIV.5)}$$

La covarianza del error de la estimación a priori es:

$$\mathbf{P}^{-}(n) = \mathbf{A}(n)\mathbf{P}(n-1)\mathbf{A}(n)^{T} + \mathbf{Q}(n) \qquad \textbf{(AIV.6)}$$

y la covarianza del error a posteriori se obtiene de

$$\mathbf{P}(n) = \left[\mathbf{I} - \mathbf{K}(n)\mathbf{H}(n)\right]\mathbf{P}^{-}(n) \qquad \textbf{(AIV.7)}$$

El algoritmo para el caso lineal sería entonces:

1. *Inicializar los elementos de la diagonal de* $\mathbf{P}(0), \quad \mathbf{Q}(1), \quad \mathbf{R}(1)$
2. ***Repetir*** *para n= 1,2...*

    1. *Calcular (AIV.3)*
    2. *Calcular (AIV.6)*
    3. *Calcular (AIV.5)*
    4. *Calcular (AIV.4)*
    5. *Calcular (AIV.7)*

    Fin repetir

La naturaleza recursiva del filtro hace que la estimación del estado del sistema esté en función de todas las mediciones del pasado pero sin tenerlas que considerarlas explícitamente.

El "rendimiento" ("performance") del filtro puede mejorarse mediante el manejo de las matrices $\mathbf{Q}(n)$ y $\mathbf{R}(n)$. Estas matrices pueden fijarse antes del funcionamiento del filtro o pueden ir cambiándose dinámicamente. Así, $\mathbf{R}(n)$ podrá ser ajustada en función de nuestra confianza en el mecanismo responsable de la medición. Por otra parte, con $\mathbf{Q}(n)$ se puede modelar nuestra incertidumbre en el modelo $\mathbf{w}(n+1) = \mathbf{A}(n)\mathbf{w}(n) + \mathbf{B}(n)\mathbf{u}(n) + \boldsymbol{\omega}(n)$

Dado que este algoritmo propaga la matriz de covarianza es llamado algoritmo de Kalman de filtrado de covarianza. Debido a que la estimación de $\mathbf{P}$ a posteriori puede llegar a no ser definida no negativa, a causa de los errores de cálculo acumulados (lo cual es inaceptable ya que es una matriz de covarianza), se ha sugerido una mejora (el "square root Kalman filter") en la forma de propagar $\mathbf{P}$ (es decir, utilizar otra ecuación en vez de la AIV.7, ver [5]).

AIV.1.2 El filtro de Kalman extendido

Normalmente el proceso a estimar o la ecuación de medición (correspondientes a las ecuaciones AIV.1 y AIV.2) son no lineales. En ese caso debe hacerse, por lo tanto, una aproximación si se quiere aplicar el filtro de Kalman lineal. Un filtro de Kalman que linealiza en torno a la media y a la covarianza actual se denomina un ***filtro de Kalman extendido***. Existen varias propuestas para aplicar el filtro de Kalman a sistemas no lineales. Desarrollaremos a continuación una de ellas.

Sea un proceso cuyo vector de estado $\mathbf{w}(n) \in \mathbf{R}^m$ cumple que

$$\mathbf{w}(n+1) = \mathbf{f}[\mathbf{w}(n), \mathbf{u}(n)] + \boldsymbol{\omega}(n) \qquad \textbf{(AIV.8)}$$

con una medición $\mathbf{d} \in \mathbf{R}^N$ que es

$$\mathbf{d}(n) = \mathbf{h}[\mathbf{w}(n), \mathbf{u}(n)] + \boldsymbol{\upsilon}(n) \qquad \textbf{(AIV.9)}$$

donde las variables $\boldsymbol{\omega}(n), \quad \boldsymbol{\upsilon}(n)$ representan, como antes, el ruido blanco de media cero del proceso y de la medida respectivamente y siendo $\mathbf{Q}(n)$ y $\mathbf{R}(n)$ sus matrices de covarianza. Las funciones $\mathbf{f}$ y $\mathbf{h}$ son funciones no lineales que relacionan el estado en el instante n con el estado en el instante $n+1$ y con la medición $\mathbf{d}(n)$ respectivamente.

Mediante la linealización de la estimación actual usando las derivadas de las funciones del estado y de medida se llega a una serie de ecuaciones equivalentes a las del caso lineal. Dicha linealización es similar a un desarrollo de Taylor alrededor de la estimación actual usando las derivadas parciales de la ecuación del proceso AIV.8 y de medida AIV.9:

$$\mathbf{w}(n+1) \approx \hat{\mathbf{w}}^-(n+1) + \mathbf{A}[\mathbf{w}(n) - \hat{\mathbf{w}}(n)] + \mathbf{W}\boldsymbol{\omega}(n)$$
$$\mathbf{d}(n) \approx \hat{\mathbf{d}}^-(n) + \mathbf{H}[\mathbf{w}(n) - \hat{\mathbf{w}}^-(n)] + \mathbf{V}\boldsymbol{\upsilon}(n)$$

(ver [54] y [53] para más detalles). Así, la estimación a priori del estado , $\widehat{\mathbf{w}}^-(n)$ se aproxima ahora haciendo

$$\widehat{\mathbf{w}}^-(n) = \mathrm{f}[\widehat{\mathbf{w}}(n-1), \mathbf{u}(n-1)] \qquad \textbf{(AIV.10)}$$

y la covarianza del error a priori se calcula con

$$\mathbf{P}^-(n) = \mathbf{A}(n-1)\mathbf{P}(n-1)\mathbf{A}(n-1)^T + \mathbf{W}(n-1)\mathbf{Q}(n-1)\mathbf{W}(n-1)^T \qquad \textbf{(AIV.11)}$$

donde $\mathbf{A}$ es el jacobiano de $\mathbf{f}$ respecto al estado

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

y $\mathbf{W}$ es la matriz de derivadas parciales de $\mathbf{f}$ respecto al ruido $\boldsymbol{\omega}$:

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

La matriz de ganancia $\mathbf{K}(n)$ se obtiene en este caso a partir de la ecuación

$$\mathbf{K}(n) = \mathbf{P}^{-}(n)\mathbf{H}(n)^T \left[\mathbf{H}(n)\mathbf{P}^{-}(n)\mathbf{H}(n)^T + \mathbf{V}(n)\mathbf{R}(n)\mathbf{V}(n)^T \right]^{-1} \qquad \textbf{(AIV.12)}$$

donde $\mathbf{H}$ es aquí el jacobiano de las derivadas parciales de $\mathbf{h}$ respecto al estado y $\mathbf{V}$ es el de las derivadas parciales de $\mathbf{h}$ respecto a υ:

$$\mathbf{H}(n) = \left\{ h_{ij} = \frac{\partial h_i[\widehat{\mathbf{w}}^{-}(n), \mathbf{u}(n)]}{\partial w_j} \quad i = 1, 2 \ldots N \quad j = 1, 2 \ldots m \right\}$$

$$\mathbf{V}(n) = \left\{ \mathrm{v}_{ij} = \frac{\partial h_i[\widehat{\mathbf{w}}^{-}(n), \mathbf{u}(n)]}{\partial \upsilon_j} \quad i = 1, \ldots N \;\; j = 1, 2 \ldots m \right\}$$

Observación: Si bien los elementos v_{ij} son escalares, y deberían ser anotados en itálica para ser coherente con el resto del documento, en esta ecuación se los escribió así para no confundirlos con la letra griega υ.
La estimación a posteriori del estado utiliza también aquí $\mathbf{K}$ para ponderar la diferencia entre la medición real y una predicción de la medida:

$$\widehat{\mathbf{w}}(n) = \widehat{\mathbf{w}}^{-}(n) + \mathbf{K}(n)\{\mathbf{d}(n) - \mathrm{h}[\widehat{\mathbf{w}}^{-}(n), \mathbf{u}(n)]\} \qquad \textbf{(AIV.13)}$$

Finalmente, la covarianza $\mathbf{P}$ del error a posteriori tiene una forma similar a la del caso lineal (Ec. AIV.7):

$$\mathbf{P}(n) = [\mathbf{I} - \mathbf{K}(n)\mathbf{H}(n)]\mathbf{P}^{-}(n) \qquad \textbf{(AIV.14)}$$

aunque debe tenerse en cuenta que ahora $\mathbf{H}(n)$ se calcula de manera diferente al caso lineal.

La operación básica del filtro de Kalman extendido se puede resumir en los siguientes pasos:

1. *Inicializar los elementos de* $\mathbf{P}(0), \quad \mathbf{Q}(1), \quad \mathbf{R}(1)$
2. ***Repetir*** *para n=1,2...*

    1. *Se proyectan (predicen) las estimaciones del estado y de la covarianza del error del instante n al n+1 usando para ello las ecuaciones AIV.10 y AIV.11*
    2. *Se utilizan las nuevas estimaciones a priori obtenidas para obtener las estimaciones corregidas al considerar la medición* $\mathbf{d}(n)$*. Las ecuaciones AIV.13 y AIV.14, en ese orden, dan las estimaciones a posteriori del estado y de la covarianza del error a posteriori.*
    3. *Al igual que para el caso lineal,* $\mathbf{R}(n)$ *y* $\mathbf{Q}(n)$ *son parámetros del algoritmo que deben ser ajustados con cuidado en cada paso para obtener buenos resultados* ([54] y [53]).

Fin repetir

AIV.1.3 El filtro extendido global

Al aplicar el filtro de Kalman al entrenamiento de redes neuronales (recurrentes o no), el aprendizaje se considera como un problema de filtrado en el que los parámetros óptimos de la red se estiman en forma recursiva a partir de las ecuaciones del filtro [5]. El algoritmo es especialmente idóneo para situaciones de aprendizaje en línea, en las que los pesos se ajustan continuamente, aunque también puede aplicarse al procesamiento fuera de línea (Feldkamp y Puskorius, 1994, citados por [54]).

Comencemos considerando que el estado de la red, que denotaremos por $\mathbf{w}(n)$, para una entrada dada $\mathbf{u}(n)$, viene dado por los valores de sus pesos. El algoritmo asume que el valor óptimo de los pesos es estacionario: $\mathbf{w}(n+1) = \mathbf{w}(n)$ luego de que se ha alcanzado un mínimo, ya sea local o global. Si consideramos la red como un sistema dinámico, vemos que la ecuación describe el comportamiento de los pesos es lineal y sigue la forma

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

con $\mathbf{A}(n)=\mathbf{I}$, $\mathbf{B}(n)=\mathbf{0}$ y $\boldsymbol{\omega}(n) = \mathbf{0} \quad \forall n$ (esta última igualdad la consideraremos más adelante). Por "los pesos" estamos entendiendo también los umbrales de activación de las neuronas, tal como se convino en II.1.1.

Por lo tanto, esta ecuación asume que el sistema se encuentra en un estado óptimo y estable. Este estado puede corresponder a un mínimo local o global de la superficie de error.

La medida $\mathbf{d}(n)$ corresponde a la salida deseada de la red neuronal. Se trata, por lo tanto, de una ecuación no lineal como la

$$\mathbf{d}(n) = \mathrm{h}[\mathbf{w}(n), \mathbf{u}(n)] + \boldsymbol{\upsilon}(n) \qquad \textbf{(AIV.15)}$$

pero con la forma

$$\mathbf{d}(n) = \mathbf{y}(n) + \boldsymbol{\upsilon}(n) \qquad \textbf{(AIV.16)}$$

donde $\mathbf{y}(n)$ es la salida de la red cuado se aplica a sus entradas $\mathbf{u}(n)$.

Debido a que la ecuación de estado es lineal, el filtro de Kalman utilizará las ecuaciones (AIV.3) y (AIV.6) con $\mathbf{A}(n)=\mathbf{I}$, $\mathbf{B}(n)=\mathbf{0}$ y $\mathbf{Q}(n) = \mathbf{0}$ para todo n. La no linealidad de la ecuación (AIV.15) añade las restantes ecuaciones del filtro: las ecuaciones del filtro no lineal se interpretan sustituyendo $\mathbf{h}(n)$ por $\mathbf{y}(n)$ en (AIV.16) cuando la red utiliza los pesos $\mathbf{w}(n)$.

El jacobiano $\mathbf{V}(n)$ que se utiliza para calcular la matriz $\mathbf{K}$ se hace normalmente igual a la identidad, $\mathbf{V}(n)=\mathbf{I}$, ante la dificultad de una estimación correcta de su valor. Se asume, entonces, que su influencia está de alguna manera "oculta" dentro de $\mathbf{R}(n)$ (Ver [54]).

Las derivadas parciales que forman la matriz $\mathbf{H}(n)$ se calculan normalmente en forma análoga a como se hace en BPTT o RTRL ([54] y [52]).

Con todo lo anterior, ya tendríamos una versión del denominado ***filtro de Kalman extendido global***. No obstante, para su utilización real como algoritmo de entrenamiento es aconsejable la introducción de algunas modificaciones. La justificación de la denominación "global" se verá a continuación.

AIV.1.4 El filtro de Kalman extendido desacoplado

Cuando se trabaja en redes de cierto tamaño, el vector de estado $\mathbf{w}(n)$ puede tener un número considerable de componentes. Ello ocasiona que los cálculos realizados con matrices como $\mathbf{H}(n)$ requieran una cantidad elevada de recursos computacionales, incluso para redes de tamaño moderado. El ***filtro de Kalman extendido desacoplado*** ("decoupled extended Kalman filter", o deKf) reduce esta complejidad [5].

Para lograr que el problema tenga un tamaño computacionalmente tratable el dKef divide los pesos de la red en g grupos, obteniéndose otros tantos vectores de estado $\mathrm{w}_i \quad i = 1,2 \ldots g$. Habrá tantos grupos como neuronas en la red [30] y dos pesos pertenecerán al mismo grupo si forman parte de la entrada de la misma neurona. La versión desacoplada, por lo tanto, aplica el filtro de Kalman extendido a cada neurona independientemente para estimar el valor óptimo de los pesos que llegan a ella. De esta forma solo se consideran las interdependencias locales durante el entrenamiento.

La principal diferencia entre la versión desacoplada y la global es la sustitución de la matriz $\mathbf{H}$ por g matrices de la forma

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

donde $w_j^{(k)}$ es el j-ésimo peso del grupo k y N es el número de neuronas. Se ha supuesto que m es el número de pesos del grupo k. Con esto la matriz $\mathbf{H}(n)$ es simplemente la concatenación de las matrices $\mathbf{H}_i(n)$:

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

Se puede observar que el deKf se reduce al global cuando $g = 1$.

El algoritmo correspondiente a la versión desacoplada queda entonces:

1. *Hacer g igual al número de neuronas de la red*
2. *inicializar los pesos de la red,* $\widehat{\mathbf{w}}_i(0) \quad i = 1,2 \ldots g$
3. *Inicializar los elementos de la diagonal de* $\mathbf{R}(1)$ *y* $\mathbf{P}_i(0)$ i=1,2...g.
4. *Para n =* $1,2 \ldots .$ $i = 1,2 \ldots g$ *calcular las siguientes ecuaciones:*

$$\widehat{\mathbf{w}}_i^-(n) = \widehat{\mathbf{w}}_i^-(n-1)$$

$$\mathbf{P}_i^-(n) = \mathbf{P}_i^-(n-1)$$

$$\mathbf{K}_i(n) = \mathbf{P}_i^-(n)\mathbf{H}_i(n)^T \left(\sum_{j=1}^{g} \mathbf{H}_j(n)\mathbf{P}_j^-(n)\mathbf{H}_j(n)^T + \mathbf{R}(n) \right)^{-1}$$

$$\widehat{\mathbf{w}}_i(n) = \widehat{\mathbf{w}}_i^-(n) + \mathbf{K}_i(n)[\mathbf{d}(n) - \mathbf{y}(n)]$$

$$\mathbf{P}_i(n) = [\mathbf{I} - \mathbf{K}_i(n)\mathbf{H}_i(n)]\mathbf{P}_i^-(n)$$

donde $\mathbf{d}(n)$ *es la salida deseada de la red en el instante n e* $\mathbf{y}(n)$ *es la salida real de la red para la entrada* $\mathbf{u}(n)$*. Nótese que las dos primeras ecuaciones no son necesarias en la implementación del algoritmo y puede trabajarse con las estimaciones a posteriori del instante anterior directamente.*

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

Convergencia

La forma no lineal del deKf provoca numerosas dificultades numéricas a la hora de su implementación que hace que el filtro diverja de la solución correcta. Una forma de evitar esta divergencia [5] es añadir ruido a la ecuación del proceso,

[30] No es válido para el caso de las redes LSTM. Ver [48]

haciendo que $\boldsymbol{\omega}(n) \neq 0$. El único cambio sobre la forma descripta anteriormente para el deKf es la ecuación $\mathbf{P}_i(n) = [\mathbf{I} - \mathbf{K}_i(n)\mathbf{H}_i(n)]\mathbf{P}_i^-(n)$ del paso 4 del algoritmo, que se convierte en

$$\mathbf{P}_i(n) = [\mathbf{I} - \mathbf{K}_i(n)\mathbf{H}_i(n)]\mathbf{P}_i^-(n) + \mathbf{Q}_i(n)$$

Además de evitar la divergencia, la introducción de $\mathbf{Q}_i(n)$ tiene el efecto secundario de hacer que el algoritmo tenga menor propensión a quedarse atrapado en los mínimos locales. Normalmente se usa la misma matriz para todos los grupos, por lo que hablaremos simplemente de $\mathbf{Q}(n)$. Los valores iniciales de $\mathbf{R}(1)$ y $\mathbf{Q}(1)$ se discuten a continuación.

Parámetros iniciales del algoritmo

Los parámetros a ajustar en la inicialización del deKf son:

- el valor inicial de la covarianza del error a posteriori $\mathbf{P}_i(0)$; este valor se suele inicializar como $\mathbf{P}_i(0) = \delta\mathbf{I}$ donde δ es una constante positiva e **I** la matriz identidad.

- los elementos diagonales de la matriz inicial de covarianza del ruido de la medida $\mathbf{R}(1)$; estos elementos se ajustan generalmente desde un valor inicial a valores más bajos conforme avanza el entrenamiento.

- los elementos de la diagonal de la matriz de covarianza inicial del error del proceso $\mathbf{Q}(1)$; estos valores también se ajustan en forma decreciente.

El ajuste de los elementos de las matrices de covarianza consiste en darles un valor inicial e ir decrementándolo paulatinamente según una determinada tasa de reducción *T*. Por ejemplo, consideremos que los valores de la diagonal de la matriz $\mathbf{R}(0)$ se inicializan con un valor $\mathbf{R}_{\max}$ y se ajustan con una tasa *T* hasta alcanzar el valor $\mathbf{R}_{\min}$.

Una posible ecuación a aplicar para obtener esta evolución es [54]:

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

Costo computacional

Dado que el deKf se apoya en el cálculo de la derivada del error, que puede realizarse tanto con BPTT como RTRL, el costo es, por lo tanto, como mínimo igual al del algoritmo de cálculo de derivadas utilizado. En [5, Cap. 15] se puede ver una comparación de los recursos de almacenamiento y operaciones necesarias para RTRL, BPTT y deKf.

Por último, para redes MLPs se ha encontrado que el filtro de Kalman extendido global suele converger en menos iteraciones que el BP estándar [52].

AIV.2 Las redes LSTM

En las redes recurrentes formadas por neuronas no lineales solamente los eventos significativos en la secuencia de entrada para el entrenamiento no pueden estar muy alejados en el tiempo entre sí ya que los errores al fluir hacia atrás en el tiempo o bien decaen exponencialmente o bien crecen sin medida (problema de los gradientes evanescentes o

"vanishig gradients") (Ver II.4.3 y [35]). Esto limita a las redes recurrentes a problemas que tienen solo saltos de tiempo cortos (menos de 10 instantes de tiempo) entre entradas relevantes y señales deseadas. Las redes LSTM ("long short-term memory") resuelven este problema al forzar un flujo de error constante, lo que permite que las redes LSTM aprendan tareas que requieren almacenar información de eventos relevantes por hasta más de 1000 instantes de tiempo ("***timesteps"***) . Estas redes pueden ser entrenadas por descenso en el gradiente o bien usando el filtro extendido desacoplado de Kalman ("decoupled extended Kalman filter" o deKf).

Describiremos a continuación su arquitectura y funcionamiento en general [35].

AIV.2.1 Arquitectura

La unidad básica en la capa oculta de una red LSTM es el ***bloque de memoria***, que contiene una o más ***celdas de memoria*** y un par de neuronas que trabajan como ***compuertas*** ("gates") de entrada y salida a todas las celdas del bloque. Cada celda de memoria tiene en su núcleo una neurona lineal conectada a sí misma llamada el "Constant Error Carrousel" (CEC). Esa conexión recurrente fuerza a que el flujo de error sea constante en el tiempo. Solo esa CEC es la que lleva la pista del error a medida que él fluye hacia atrás en el tiempo. Los errores de las demás neuronas se comportan como en las demás redes recurrentes. En ausencia de nuevas entradas a la celda, el error local del CEC fluye hacia atrás en el tiempo sin crecer ni decrecer [27].

El esquema para la celda c es:

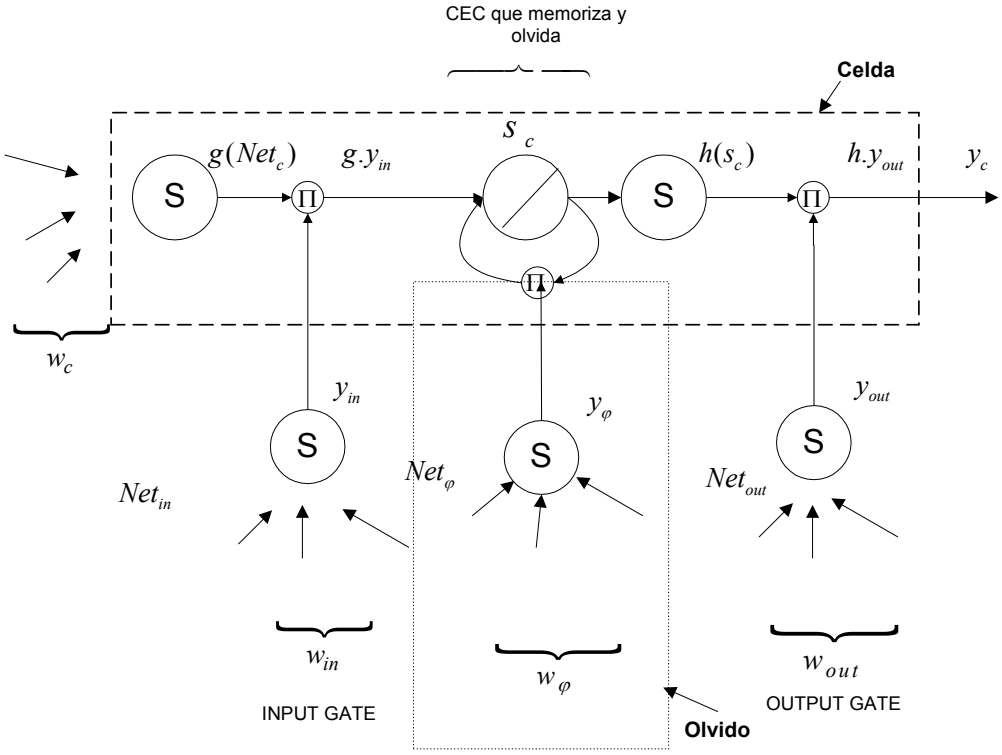


Figura AIV.1

donde la S significa que la función de transferencia es sigmoide, de rango [-1, 1] en el caso de la función h, y en el rango [-2, 2] en el de g, mientras que la línea en diagonal representa a la función de transferencia lineal. Las compuertas usan sigmoides logísticas en el rango [0, 1]. La celda es el recuadro en línea cortada. El recuadro en punteado ("olvido") no existe en celdas que no tienen la capacidad de olvidar (ver más adelante), y es sustituido por una conexión recurrente en la neurona lineal con peso = 1 (este es el caso de la celda LSTM estándar).

La "input gate" se puede usar para decidir si se almacena cierta información en la celda. Simultáneamente, la "output gate" es colocada para proteger a otras neuronas de la perturbación producida por contenidos de memoria irrelevantes almacenados en la celda.

Las señales de error almacenadas dentro del CEC no pueden cambiar, pero pueden superponerse diferentes señales de error fluyendo dentro de la celda (en distintos momentos) vía su "output gate". Esencialmente, las compuertas ("gates") abren y cierran (por ejemplo, cierran mediante un valor y_{in} y y_{out} cercanos a 0) el acceso al flujo de error constante a través del CEC.

Estas celdas se agrupan en bloques de memoria, y comparten las compuertas.

AIV.2.2 Entrenamiento

Las redes LSTM se entrenan bien con algoritmos derivados del descenso en el gradiente diseñados a propósito o bien con el deKf. En el primer caso, el algoritmo propuesto por Hochreiter es lineal en espacio y tiempo, usa $O(1)$ operaciones por instante (paso) de tiempo y neurona. El deKf por su parte permite obtener convergencia más rápida, aunque bajan las capacidades de memoria a lago plazo (es decir que el largo de los períodos que pueden existir entre eventos correlacionados a aprender tiende a disminuir) [58].

AIV.2.3 Redes LSTM con olvido

A veces los valores de entrada al CEC, s_c, tienden a crecer linealmente durante la presentación de la serie temporal, causando la saturación de la neurona con función de salida h. Esto hace que a) la derivada de h sea casi nula, bloqueando los errores que llegan y b) hace que la salida de la celda iguale la salida de la "output gate", esto es, la celda de memoria entera va a degenerar en una neurona común (entrenable con BPTT), dejando de funcionar como memoria. La solución al problema se da en [27] usando compuertas "de olvido" (marcado en punteado) que aprenden a "resetear" bloques de memoria una vez que esto sucede.

AIV.2.5 Un ejemplo

Un ejemplo de cómo se utilizaría una red LSTM de tres capas y un bloque de memoria con dos celdas con la recurrencia limitada a la capa oculta podría ser el mostrado en la figura AIV.2:

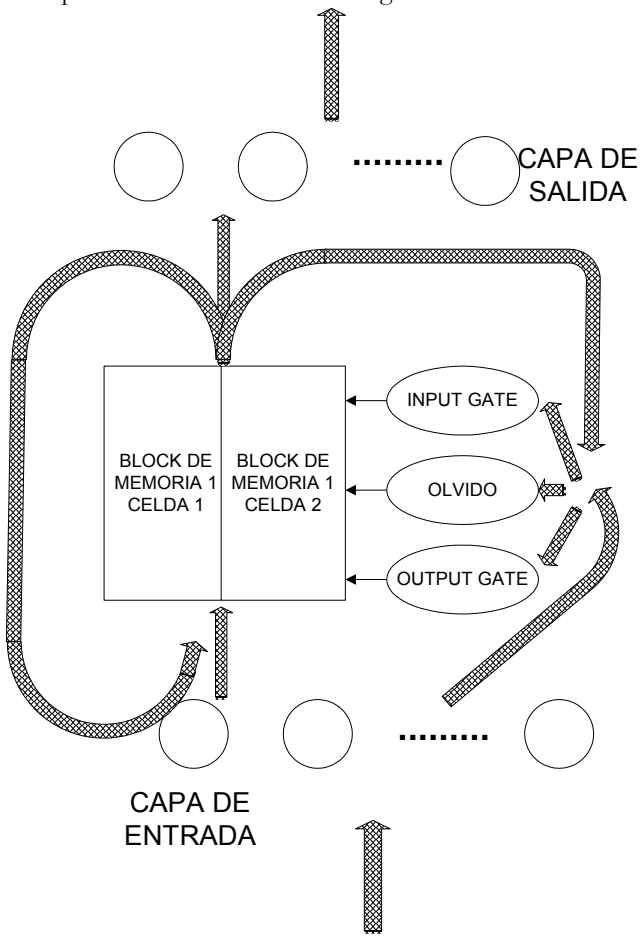


Figura AIV.2

ANEXO V: Criterios de selección de modelos: el AIC y el MDL

Para elegir un modelo entre varios posibles, pueden utilizarse distintos criterios. Para empezar, el que produzca menor error promedio sobre el conjunto de de entrenamiento en igualdad de condiciones (igual cantidad de épocas, porcentaje de patrones usados para CV, etc.) no necesariamente será el que produzca un menor error de predicción. Podríamos entonces dejar un conjunto de patrones sin usar durante el entrenamiento, usarlos como prueba ("testing") para ver qué tan bien predice cada modelo, y elegir aquel con el menor de estos errores promedio. Si bien este es un criterio más acertado que el anterior, ya que empieza a tener en cuenta el error de predicción, es totalmente empírico y no tiene un fundamento teórico. Es interesante también considerar la correlación lineal entre los valores deseados a obtener y los valores realmente obtenidos. Cuando dicha correlación es alta y positiva, sabemos que el modelo producirá datos que tendrán un comportamiento similar a los reales, es más, aun cuando el criterio de error de "testing" pareciera hacer preferible otro modelo, el que tuviera mayor coeficiente de correlación podría ser preferible. Tanto el error de predicción, el de "testing" como el coeficiente de correlación son proporcionados por el simulador elegido.

Dada la tarea de predicción de una serie temporal que se pretende realizar, otro criterio podría ser elegir aquel modelo cuyo error varíe menos al predecir etapas futuras a medida que ese futuro es más lejano respecto a los datos de entrenamiento, o sea, el que sea más ***robusto***. En otras palabras, si se entrenó el modelo con los patrones correspondientes a los instantes 1....p, y se predice para la p+1, p+2,...p+k elegiremos aquel modelo para el cual k es máximo, sin que se supere una cota de error de predicción prefijada.

Dado que sabemos que la capacidad de generalización (predicción) del modelo va a depender de su complejidad[31], podemos resumir los criterios anteriores en dos indicadores que son proporcionados en forma directa por el simulador: el índice de información de Akaike (AIC) y el Minimun Description Lenght (MDL) de Rissanen. Para ser más específicos, estos dos indicadores tienen en cuenta la complejidad del modelo, la performance (error) de entrenamiento y el número de ejemplares (patrones) disponibles. Junto con ellos existen muchos otros definidos como por ejemplo el GPE de Moody ([42]), el BIC, etc. que no se detallan por motivos de brevedad y por quedar su discusión fuera del alcance del presente trabajo.

AV.1 El índice de Akaike (AIC)

AV.1.1 Deducción del índice

Una medida de la "separación" entre dos distribuciones de probabilidad es su divergencia asimétrica, entropía relativa o de Kullback-Leibler. Lo que se intenta en el ***criterio de Akaike*** es minimizar esta separación entre el modelo ajustado y el verdadero [19].

Supongamos que los datos $\mathbf{y}_1, \mathbf{y}_2 \cdots$ son generados de acuerdo a una densidad de probabilidad "verdadera" $f(\mathbf{y} \mid \boldsymbol{\theta}_0)$ donde $\boldsymbol{\theta}$ es un parámetro k dimensional. Sea $\boldsymbol{\Phi}(k)$ una familia de densidades (empíricas) paramétricas, $\boldsymbol{\Phi}(k) = \{\mathrm{f}(\mathbf{y}|\boldsymbol{\theta}_k) \mid \boldsymbol{\theta}_k \in \boldsymbol{\Omega}(k) \subset \mathbf{R}^k\}$ y sean $\boldsymbol{\theta}_k^*$ la estimación del parámetro obtenida al maximizar la función de verosimilitud $f(\mathbf{y} \mid \boldsymbol{\theta})$ sobre $\boldsymbol{\Omega}(k)$ y $f(\mathbf{y} \mid \boldsymbol{\theta}_k^*)$ la densidad empírica resultante. Nuestra meta es buscar en un conjunto de familias $\boldsymbol{\mathcal{F}} = \{\boldsymbol{\Phi}(k_1), \boldsymbol{\Phi}(k_2), \ldots \boldsymbol{\Phi}(k_L)\}$ el modelo $f(\mathbf{y} \mid \boldsymbol{\theta}_k) \quad k \in \{k_1 \ldots k_L\}$ que mejor aproxime $f(\mathbf{y} \mid \boldsymbol{\theta}_0)$.

[31] A mayor complejidad, menor capacidad de generalización, seguramente debido al overfitting.

Para determinar cual de los modelos ajustados $f(\mathbf{y}\mid\boldsymbol{\theta}^*_{k_1}), f(\mathbf{y}\mid\boldsymbol{\theta}^*_{k_2})\ldots f(\mathbf{y}\mid\boldsymbol{\theta}^*_{k_L})$ se parece más a $f(\mathbf{y}\mid\boldsymbol{\theta}_0)$ necesitamos una medida de la disparidad entre el modelo verdadero $f(\mathbf{y}\mid\boldsymbol{\theta}_0)$ y uno "aproximante" $f(\mathbf{y}\mid\boldsymbol{\theta}_*)$.

Esa medida será la entropía relativa entre ellos: $d(\boldsymbol{\theta}_0,\boldsymbol{\theta}_*)=\left\langle Log[f(\mathbf{y}\mid\boldsymbol{\theta}_0)/f(\mathbf{y}\mid\boldsymbol{\theta}_*)]\right\rangle_{\mathbf{y}}$

Si llamamos $\delta(\boldsymbol{\theta}_0,\boldsymbol{\theta}_*)=\left\langle -2\log f(\mathbf{y}\mid\boldsymbol{\theta}_*)\right\rangle_{\mathbf{y}}$ podemos escribir

$$2d(\boldsymbol{\theta}_0,\boldsymbol{\theta}_*)=\delta(\boldsymbol{\theta}_0,\boldsymbol{\theta}_*)-\delta(\boldsymbol{\theta}_0,\boldsymbol{\theta}_0)$$

Como $\delta(\boldsymbol{\theta}_0,\boldsymbol{\theta}_0)$ no depende de $\boldsymbol{\theta}_*$, cualquier clasificación de un conjunto de modelos a elegir que se base en $d(\boldsymbol{\theta}_0,\boldsymbol{\theta}_*)$ será idéntica a la clasificación correspondiente a $\delta(\boldsymbol{\theta}_0,\boldsymbol{\theta}_*)$. Por lo tanto, para elegir el modelo podemos usar $\delta(\boldsymbol{\theta}_0,\boldsymbol{\theta}_*)$ en vez de $d(\boldsymbol{\theta}_0,\boldsymbol{\theta}_*)$.

Ahora bien, para un estimador de máxima verosimilitud $\boldsymbol{\theta}^*_k$ la divergencia exacta entre el modelo ajustado y el verdadero está dada por $\delta(\boldsymbol{\theta}_0,\boldsymbol{\theta}^*_k)$, pero calcular $\delta(\boldsymbol{\theta}_0,\boldsymbol{\theta}^*_k)$ requeriría conocer la densidad verdadera $f(\mathbf{y}\mid\boldsymbol{\theta}_0)$, lo cual obviamente no es posible. Es por ello que Akaike propuso usar $-2\log f(\mathbf{y}\mid\boldsymbol{\theta}^*_k)$ como un estimador sesgado de $\delta(\boldsymbol{\theta}_0,\boldsymbol{\theta}^*_k)$, con un sesgo que puede ser aproximado por $2k$. Esta aproximación del sesgo es válida siempre que

- para alguna de las familias $\boldsymbol{\Phi}_i$ se cumple que $f(\mathbf{y}\mid\boldsymbol{\theta}_0)\in\boldsymbol{\Phi}_i$
- se cumplen un conjunto de condiciones de regularidad que aseguren las propiedades asintóticas del estimador de máxima verosimilitud $\boldsymbol{\theta}^*_k$ (Ver [19]).

A esa aproximación de $\delta(\boldsymbol{\theta}_0,\boldsymbol{\theta}^*_k)$ se la llama ***Akaike's Information Criterion (AIC):***

$$AIC=-2\log f(\mathbf{y}\mid\boldsymbol{\theta}^*_k)+2k$$

Si se cumplen estas dos condiciones, entonces $\left\langle AIC\right\rangle_{\mathrm{y}} \xrightarrow[\text{al crecer el tamaño de la muestra}]{\text{asintoticamente}} \left\langle\delta(\boldsymbol{\theta}_0,\boldsymbol{\theta}^*_k)\right\rangle_{\mathrm{y}}$, por lo que, para muestras grandes, la selección del modelo basada en el AIC debería conducir a modelos que son los más cercanos a $f(\mathbf{y}\mid\boldsymbol{\theta}_0)$ en el sentido de la entropía de Kullback-Leibler. Por lo tanto, la utilidad del AIC en el caso de trabajar con pequeñas muestras de datos, es limitada. Es por ello que se desarrolla el ***AIC de segundo orden*** (o corregido), AICc:

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

Donde N es el número de puntos de datos y k el número de parámetros del modelo. Esta aproximación solo es válida cuando $N\geq 2+k$ [10].

En especial, si suponemos que los errores (diferencias entre los valores dados por el modelo y los reales) tienen una distribución normal, entonces se puede aproximar el AIC por

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

donde N es el número de puntos de datos, los logaritmos son naturales y los y_i^* son los valores reales correspondientes a los y_i estimados por del modelo [10]. Según esta fórmula, el valor del AIC depende de las medidas elegidas para expresar los datos, por lo que un valor aislado de AIC no puede ser interpretado, sino que solo interesa la diferencia entre los valores obtenidos para dos modelos que se basan en los mismos datos. Finalmente, si se quiere cuantificar cuánto es más posiblemente correcto un modelo que otro (es decir, cual es la probabilidad de haber elegido el modelo correcto), se puede demostrar que esa probabilidad es

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

siendo α la diferencia entre los AICc [10]

AV.1.2 Uso

En nuestra aplicación, el simulador de redes neuronales calcula:

$$MSE = \frac{\sum_{j=0}^{p}\sum_{i=0}^{N}(d_{ij}-y_{ij})^2}{Np}$$

$$AIC(k) = N\log(MSE)+2k$$

donde N es el número de patrones de entrada, p el de neuronas de salida y k el número de pesos de la red.

AV.1.3 Submodelos y aplicabilidad del AIC

Sean dos modelos paramétricos $\mathfrak{M}_1(\mathbf{y}\mid\mathbf{x},\boldsymbol{\theta}_1)$ y $\mathfrak{M}_2(\mathbf{y}\mid\mathbf{x},\boldsymbol{\theta}_2)$ con $\boldsymbol{\theta}_1\in\mathbf{R}^{m_1}$ $\boldsymbol{\theta}_2\in\mathbf{R}^{m_2}$ $m_1<m_2$ y siendo $\mathbf{y}$ los datos de salida dados los datos de entrada **x.** Decimos que $\mathfrak{M}_1(\mathbf{y}\mid\mathbf{x},\boldsymbol{\theta}_1)$ es un submodelo de $\mathfrak{M}_2(\mathbf{y}\mid\mathbf{x},\boldsymbol{\theta}_2)$ y anotamos

$$\mathfrak{M}_1(\mathbf{y}\mid\mathbf{x},\boldsymbol{\theta}_1)\subset\mathfrak{M}_2(\mathbf{y}\mid\mathbf{x},\boldsymbol{\theta}_2)$$

si restringiendo algunas componentes de $\boldsymbol{\theta}_2$ a valores fijos o que cumplan ciertas relaciones fijas, obtenemos el modelo $\mathfrak{M}_1(\mathbf{y}\mid\mathbf{x},\boldsymbol{\theta}_1)$. Por ejemplo, en el caso de redes multicapa, el número de neuronas de entrada y salida son las mismas en los dos modelos, pero si el número de unidades de la capa oculta es mayor en el segundo modelo ($\mathfrak{M}_2(\mathbf{y}\mid\mathbf{x},\boldsymbol{\theta}_2)$), tomando los pesos que llegan/salen a/desde las neuronas adicionales iguales a 0, obtenemos el modelo $\mathfrak{M}_1(\mathbf{y}\mid\mathbf{x},\boldsymbol{\theta}_1)$.

Se puede demostrar que el índice AIC, así como algunas de sus generalizaciones (como el NIC- Network Information Criterion) son solo aplicables al caso de modelos jerárquicos (o sea entre los que existe una relación de "contenido en") como la anterior [46].

AV.2 El "Minimun Description Lenght" (MDL)

AV.2.1 Introducción

Al estudiar la complejidad de un modelo existen al menos dos dimensiones independientes que la afectan: el número de parámetros libres y la forma funcional en que se combinan esos parámetros. Por ejemplo, intuitivamente los

modelos $y = \theta x$ y $y = x^{\theta}$ tienen distintas complejidades. Al variar estas dos dimensiones, se obtienen las diferentes mejoras al ajuste de modelo sobre los datos, sin mejorar necesariamente la capacidad de generalización. El modelo elegido deberá ser lo suficientemente complejo como para describir los datos con exactitud, pero sin llegar al overfitting y perder capacidad de generalización. El MDL fue definido por Rissanen (1978) asociado a la codificación de datos y permite elegir entre varios modelos, tomando en cuenta los parámetros y la forma funcional en que se relacionan.

Imaginemos que se quiere transmitir un conjunto de datos *D* desde un emisor a un receptor, tratando de usar un mensaje de largo mínimo (donde el largo podría ser medido, por ejemplo, por la cantidad de bits que lleve codificarlo). Un enfoque podría ser simplemente transmitir el mensaje *D* codificado usando algún esquema de codificación fijo, suponiendo que el emisor y el receptor son independientes. Sin embargo, si hay aspectos que se repiten sistemáticamente en los datos, aspectos que no son conocidos previamente a la recepción de los datos por el receptor, podría esperarse que fuese posible transmitir un mensaje más corto si primero transmitimos información especificando el modelo $\mathfrak{M}$ que captura esos aspectos, usando para ello un mensaje de largo L($\mathfrak{M}$) y luego un segundo mensaje especificando como los datos reales difieren de los predichos por el modelo $\mathfrak{M}$. Podríamos ver a L($\mathfrak{M}$) como una medida de la complejidad del modelo, ya que un modelo más complejo va a requerir más información para describirlo. El mensaje necesario para enviar la información de discrepancia tiene un largo dado por $\mathrm{L}(\boldsymbol{D} \mid \mathfrak{M})$ y puede ser visto como un término de error. Por lo tanto, el largo total del mensaje que es enviado es

$$\text{Largo de la descripción} = \underbrace{\mathrm{L}(\boldsymbol{D} \mid \mathfrak{M})}_{error} + \underbrace{\mathrm{L}(\mathfrak{M})}_{complejidad}$$

Vemos que el objetivo de elegir la descripción de largo mínimo (***minimun description lenght***, MDL) lleva a una forma natural de la navaja de Occam: un modelo muy simple va a ser un predictor pobre de los datos, por lo que los errores van a ser grandes y esto llevará a un término de error grande en la ecuación anterior. Si usamos un modelo más complejo, tendremos un menor término de error, pero si es demasiado complejo va a requerir mucha información para especificarlo lo que llevará a un término de complejidad grande en la ecuación. Intuitivamente esperamos que la descripción más corta ocurra cuando el modelo $\mathfrak{M}$ da una representación exacta del proceso que generó los datos, y también esperamos, que este modelo tenga las mejores propiedades de generalización, en promedio.

AV.2.2 Definición formal

Dado el vector de parámetros $\boldsymbol{\theta}$ y dados los datos observados **y** representados por la variable aleatoria Y**,** la matriz de Fisher $I(\boldsymbol{\theta})$ se define como

$$I(\boldsymbol{\theta}) = \left\{ I_{ij}(\boldsymbol{\theta}) = \left\langle \frac{\partial Log\ f(\mathbf{y} \mid \boldsymbol{\theta})}{\partial \boldsymbol{\theta}_i} \frac{\partial Log\ f(\mathbf{y} \mid \boldsymbol{\theta})}{\partial \boldsymbol{\theta}_j} \right\rangle_Y \right\}$$

siendo $P(\mathbf{y} \mid \boldsymbol{\theta})$ la función de densidad de probabilidad de Y (Ver [2 Cap. 12]).

Sea una familia $\boldsymbol{\Phi} = \{\Phi(\mathbf{y}, \boldsymbol{\theta}) \quad \boldsymbol{\theta} \in \mathbf{R}^k \ \mathbf{y} \in \mathbf{R}^m\}$ de modelos paramétricos de parámetro $\boldsymbol{\theta} \in \mathbf{R}^k$ tal que cada uno tiene asociada una función de verosimilitud $\boldsymbol{\varphi}(\mathbf{y} \mid \boldsymbol{\theta})$ para un conjunto de datos observados **y.** Para esa familia, se tiene por definición [47]:

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

donde

Log es el logaritmo natural

$\boldsymbol{\theta}^*$ es el valor de $\boldsymbol{\theta}$ que maximiza la verosimilitud

k es el número de parámetros del modelo

N es el tamaño de la muestra de datos

$I(\boldsymbol{\theta})$ es la matriz de información de Fisher

Interpretación:

El término A nos da, a partir del Log de la verosimilitud, una medida de la bondad (o falta) de ajuste del modelo respecto a los datos Los términos B y C nos dan la complejidad intrínseca del modelo: el B a través de la cantidad de parámetros k y el C da una idea de la complejidad asociada a la forma funcional debido al uso de la matriz $I(\boldsymbol{\theta})$ [47]. Adicionalmente, a estos dos términos se les puede dar una interpretación geométrica muy interesante relacionada con los espacios de Riemman [47].

Se puede demostrar que minimizar el MDL es equivalente a maximizar la probabilidad bayesiana $P_{\boldsymbol{\varphi}} = P(\varphi \mid \mathbf{y})$, es decir, maximizar la probabilidad de que el modelo "verdadero" Φ_0 (aquel al que corresponde la densidad de probabilidad con la cual se generan los datos $\mathbf{y}$) pertenezca a la familia $\boldsymbol{\Phi}$ [47].

En cuanto al cálculo del MDL, el simulador de redes neuronales usado lo estima como

$$MDL(k) = NLogMSE + \frac{k}{2} LogN$$

donde

$$MSE = \frac{1}{Np} \sum_{j=0}^{p} \sum_{i=0}^{N} (d_{ij} - y_{ij})^2$$

p es el número de neuronas de salida

N nro. de patrones

y_{ij} salida obtenida de la red para el ejemplar (patrón) i-ésimo en la neurona j-ésima

d_{ij} salida deseada para el ejemplar (patrón) i-ésimo en la neurona j-ésima

Finalmente, una relación detallada de las relaciones entre el MDL, la teoría de la información y la estadística se puede encontrar en [31].

ANEXO VI: Disminución de los grados de libertad de la red

Para resolver los problemas que requieren redes de gran tamaño se hace necesario minimizar el tamaño de la red. A su vez, es menos probable que una red neuronal de tamaño mínimo caiga en el overfitting (es decir, que aprenda los ruidos o idiosincracia de los datos de entrenamiento) y por lo tanto generalizará mejor. El tamaño óptimo de la red se puede lograr de varias maneras:

- manejando el número de pesos (sin importar su valor) y neuronas que tiene la red

  a) haciendo crecer la red (***"network growing"***): se comienza con una red pequeña y se le van agregando neuronas a las capas ocultas a medida que se ve que no se puede cumplir con el criterio de performance. Marchand, Lebiere y otros desarrollaron algoritmos de este tipo [4, Pág. 226]. No nos ocuparemos aquí de esta metodología.

  b) podando la red (***"network pruning"***): en este caso se comienza con una gran red (típicamente, un perceptron multicapa) con una performance adecuada para el problema a resolver y se la va podando por medio de la eliminación de ciertos pesos en una manera selectiva y ordenada.

- imponiendo ciertas restricciones a los pesos de forma que no sean independientes entre sí y se disminuya el número de grados de libertad (tal como se hace en el "weight sharing").

A continuación se verán algunos métodos de podado y de "weight sharing".

AVI.2 Técnicas de podado

AVI.2.1 Algoritmo de "weight decay"

Uno de los algoritmos más simples de podado es el que trabaja a través de la degradación de los pesos, en el que cada peso decae hacia 0 a una tasa proporcional a su magnitud de forma que la conexión desaparece[32] a no ser que sea reforzada (es decir, usada en el aprendizaje).

La degradación de pesos (***"weight decay"***) en las ecuaciones de actualización de pesos pueden ser implementada añadiendo un término de regularización (Ver AVI.3) a la función E que penalice a los pesos grandes:

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

con $0 < \lambda < 1$ y n el número de pesos. Si hacemos una búsqueda en el gradiente para el mínimo de $J(\mathbf{w})$ llegamos a la regla de actualización de pesos:

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

lo que muestra una degradación exponencial de w_i si no hay aprendizaje (es decir, si antes de aplicar la degradación $\Delta w_i = -\rho\frac{\partial E}{\partial w_i} \approx 0$ sustituyendo sería $\Delta w_i \approx -\rho\lambda w_i(t)$ lo que señala el mencionado tipo de variación).

[32] O sea, el peso se vuelve tan pequeño que se puede tomar como nulo, es decir que la conexión asociada a él en la red, desaparece.

La función $J(\mathbf{w})$ desestimula el uso de pesos grandes, ya que un solo peso de gran valor "cuesta" mucho más que varios chicos: si a una neurona llegan dos conexiones con pesos posibles w y 0 o $w/2$ y $w/2$, se va a preferir los dos pesos de $w/2$ ya que $\left(\frac{w}{2}\right)^2+\left(\frac{w}{2}\right)^2 < w^2+0^2$. Esto puede hacer que se tienda a eliminar los pesos grandes, aunque se necesiten para modelar los datos.

Otra forma de eliminar pesos la propuso Weigend (citado por [4]y [5]) que se basa en tomar

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

con n el número de pesos y $0 < \lambda < 1$, en la que el término de penalización ayuda a regular las magnitudes de los pesos y w_0 es un parámetro libre positivo que debe ser determinado. Para w_0 grandes este procedimiento es equivalente al de degradación de pesos exponencial anterior y por lo tanto favorece a varios pesos pequeños mientras que si w_0 es pequeño, menos pesos grandes son favorecidos. También hay que notar que cuando $|w_i| \gg w_0$, el costo del término $\frac{w_i^2}{w_0^2}\left(1+\frac{w_i^2}{w_0^2}\right)^{-1}$ se aproxima a 1 (y entonces su peso en $J(\mathbf{w})$ se aproxima a $\lambda/2$), lo que justifica la interpretación del término de la penalización como una contra (pena) al uso de pesos grandes. En la práctica lo que se usa es un w_0 cercano a la unidad. Hay que notar que esta eliminación de pesos es muy sensible a la elección de λ. Algunos buscan el mejor valor de λ usando métodos estadísticos (Wahba, citado por [11]).

Para finalizar, se puede tomar [11]

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

Las ideas precedentes han sido aplicadas al concepto de eliminación de neuronas. Se comenzaría con un exceso de unidades ocultas y dinámicamente se descartarían a las redundantes que serían aquellas a las cuales no llegan pesos no nulos.

AVI.2.3 Técnicas de podado basadas en la hessiana

Estas técnicas se basan en usar información de las derivadas segundas de la función de error para poder simplificar la red. Parten de la premisa de que la red ya fue entrenada y tratan de identificar los parámetros (pesos) tales que si se borran de la red, la función de error aumenta en forma mínima. Hay dos algoritmos asociados a esta técnica: el "***optimal brain damage"*** (OBD) de Le Cun, que supone que la hessiana es diagonal, y el "***optimal brain surgeon"*** (OBS) que no hace tal supuesto. Una descripción completa del OBS se puede encontrar en [5].

AVI.3 La teoría de la regularización

Dada la relación $\mathbf{F(x) = y}$ con $\mathbf{F}$ e $\mathbf{y}$ conocidas y $\mathbf{x}$ a calcular, el problema se dice "ill posed" si un pequeño cambio en las variables dependientes produce un enorme cambio en las independientes (solución). La teoría de regularización fue propuesta por Tikhonov y Arsenin para trabajar con problemas "ill-posed". Por ejemplo, cuando tratamos de saber cual es el número de neuronas que debe tener la red y solo tenemos información restringida al conjunto de entrenamiento, el problema es ill-posed ya que no tenemos acceso a la performance en el conjunto de testing [11]. La idea básica de la teoría de la regularización es modificar el problema de optimización de forma de que se vuelva más

restringido y la solución tenga menos variabilidad. Lo que se hace es añadir un término extra ***(término de regularización)*** a la función de error. La función de error que se obtiene entonces es

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

donde E es la función de error original, E_r es el de regularización y λ es un parámetro (***constante de regularización***) que pondera la influencia del regularizador versus el error E y que se determina experimentalmente. Los regularizadores de Tikhonov (E_r) buscan soluciones más "suaves" al problema de optimización y por lo tanto penalizan la curvatura de la solución original (Ver [43]). Cuando se aplican en algoritmos de entrenamiento se suelen elegir regularizadores tales que sus derivadas respecto a los pesos puedan ser calculadas eficientemente, por ejemplo:

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

donde ***n*** es el número de pesos. Un término de regularización de primer orden también puede ser útil [11].

La regularización esta muy relacionada con la técnica del "weight decay" y el "optimal brain damage", es más, como recién se vio, el "weight decay" es equivalente a agregar un término regularizador que es función de los pesos, es decir,

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

AVI.3.2 "Weight sharing" y "Soft Weight Sharing"

Por ***compartir pesos ("weight sharing")*** entendemos a un método en el que varios pesos de la red son controlados por un solo parámetro (es decir, $p = w_k = w_c = \cdots$) lo que permite mejorar la generalización, ya que impone restricciones de igualdad entre los pesos, reduciendo la cantidad de parámetros (pesos) libres de la red, lo que a su vez lleva a una mejor capacidad de generalización [4]. Normalmente decidir qué pesos van a igualarse es muy difícil si no se conoce muy bien la topología de la red a usar y la aplicación específica.

Otra forma de reducir la cantidad de pesos independientes, es agrupando los mismos en "clusters" de valores, donde cada peso tiene distribución normal de media y varianza desconocidas, pero que se van adaptando a medida que el proceso de entrenamiento avanza. Nowlan y Hinton [48] hallaron que esto da mejores resultados que usar como término corrector la suma de los cuadrados de los pesos. Esto es equivalente a agregar un término adecuado de corrección a la función de error para obtener un método automático de "weight sharing". El término de regularización en ese caso es

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

donde cada $P_j(w_i)$ es una densidad normal de media μ_j y varianza σ_j, los α_j son las proporciones de la mezcla de las densidades normales $P_j(w_i)$ con $\sum_j \alpha_j = 1$, y w_i representa un peso arbitrario de la red. Se supone que los parámetros α_j, μ_j y σ_j se adaptan a medida que la red aprende. El uso de múltiples distribuciones normales adaptativas permite la implementación de un ***"soft weight sharing"*** en el cual el algoritmo de aprendizaje decide por sí mismo qué pesos deberían ser vinculados juntos en el mismo cluster. Si todas las distribuciones empiezan con una gran varianza, el agrupamiento inicial de pesos en subconjuntos va a ser muy débil (soft). A medida que la red aprende y las varianzas disminuyen, las formas de las campanas de las P_j se van haciendo más estrechas, los agrupamientos se vuelven más y más distintos y convergen a subconjuntos que dependerán de la tarea que esta siendo aprendida.[4].

En resumen, se puede decir que este término de corrección introducido lleva a un clustering no supervisado de los pesos ("weight sharing") que se regulará según las características del conjunto de entrenamiento ([48] y [4]).

ANEXO VII: Introducción a los sistemas dinámicos

Presentaremos en este anexo los fundamentos teóricos de los sistemas dinámicos, los que nos llevarán a poder interpretar el comportamiento de las redes dinámicas y que nos permitirán entrenar redes con una dimensionalidad de entradas muy reducida, al aplicar el Teorema de Takens-Mañé. La bibliografía de esta sección está basada en [5] y en [32] en el caso de sistemas determinísticos, y en varias publicaciones que se mencionan oportunamente para el caso estocástico.

AVII.1 El modelo de espacio de estados ("state-space model")

Diremos que un sistema es ***dinámico*** si sus salidas son función no solo de sus entradas actuales, sino además de las anteriores [9]. Para estudiar estos sistemas trabajaremos con un modelo matemático de ellos, el ***modelo de espacio de estados*** ("state-space model"), mediante el cual modelamos el sistema como un conjunto de N variables de estado que permiten, dado su valor en cualquier instante, predecir la evolución futura del sistema. Al número de variables de estado se lo llama ***orden del sistema***. Utilizando este modelo podemos decir, más formalmente, que un sistema dinámico es aquel cuyas variables de estado varían con el tiempo para una entrada dada [5].

AVII.1.1 El espacio de estados

En un instante dado t, el estado del sistema esta descrito por un único punto $\mathrm{P}(t)$ en un espacio N-dimensional. Este espacio puede o no ser euclidiano aunque nuestro interés aquí esta enfocado al caso euclidiano. A dicho espacio se lo llama ***espacio de estados***. La curva descripta por esos puntos $\mathrm{P}(t)$ al variar el tiempo se llama ***trayectoria*** (u órbita) del sistema.

El planteo matemático del cambio de estado puede tomar varias formas, dependiendo del sistema a modelar, pero básicamente podemos distinguir el caso determinístico del estocástico. Consideraremos siempre el caso en que el tiempo varía en forma discreta.

El enfoque determinístico

En el presente contexto, cuando hablamos de sistema "determinístico" queremos significar que los valores pasados de sus observaciones permiten la predicción de sus valores futuros sin error, es decir, que nos referimos a "operacionalmente determinístico", o sea que puede ser modelado razonablemente bien en forma determinística dados los conocimientos que tenemos de él sin considerar su naturaleza última de determinístico o estocástico. En este caso los estados cambian en forma determinística a partir de los estados actuales y posiblemente alguna variable que representa las entradas exógenas. A estos sistemas los llamaremos ***sistemas dinámicos determinísticos***. Por ejemplo, si el sistema no es influenciado por ninguna entrada exógena, escribiremos:

$$x_j(n+1) = F_j[\mathbf{x}(n)] \qquad j = 1, 2 \ldots N$$

siendo

N orden del sistema

$\mathbf{x}(n) = [x_1, \ldots x_N]$ variable de estado del sistema

$F_j(\bullet)$ una cierta función no lineal que no depende explícitamente de n y tal que no es nula para algún n. Esta característica de las F_j se mantiene para todos los modelos de sistemas dinámicos que se consideran en este trabajo.

Esta ecuación puede ser escrita en forma vectorial como:

$$\mathbf{x}(n+1) = \mathbf{F}[\mathbf{x}(n)]$$

Si en cambio el sistema es influenciado por un conjunto de factores exógenos $\mathbf{u}(n) = [u_1(n), u_2(n) \ldots u_M(n)]$ será entonces:

$$\mathbf{x}(n+1) = \mathbf{F}[\mathbf{x}(n), \mathbf{u}(n)]$$

El enfoque estocástico

Cuando los estados cambian de forma que el estado siguiente depende del actual, de las entradas al sistema y de un vector aleatorio :

$$\mathbf{x}(n+1) = \mathbf{F}[\mathbf{x}(n), \mathbf{u}(n), \boldsymbol{\omega}(n)]$$

siendo $\boldsymbol{\omega}(n)$ un vector aleatorio (llamado ***ruido dinámico*** o multiplicativo) y $\mathbf{F}$ un vector de funciones no lineales de argumento vectorial. se dice que el sistema es ***dinámico estocástico***. Este caso incluye a los sistemas con entradas exógenas, que aunque aquí se explicitaron, se pueden considerar que toman valores impredecibles y por lo tanto incluirlos en $\boldsymbol{\omega}(n)$, y a los sistemas que son muestreados en períodos irregulares de tiempo [65]. Al espacio de variación de $\boldsymbol{\omega}(n)$ se lo llama "***shift space***" y se dice que el ruido es un "***shift".*** En su forma más general se suele tomar a dicho ruido variando en un "manifold" compacto (Ver AVII.2).

En el caso de los sistemas dinámicos estocásticos, solo podemos hacer una predicción estadística de los siguientes estados del sistema.

Los sistemas dinámicos estocásticos son interesantes por varias razones: por ejemplo, si se desea modelar un sistema determinístico, que tendría que ser modelado por un modelo determinístico de un orden muy alto, al modelarlo como un sistema dinámico estocástico puede obtenerse un modelo más robusto. Adicionalmente, en aplicaciones de economía y finanzas y de la física a menudo es necesario modelar sistemas que muestran tener una componente determinística y una estocástica. Cuando $\boldsymbol{\omega}(n)$ solo puede tomar un número finito de valores, es decir, que en cada instante, $\mathbf{F}$ es elegida de entre un cierto conjunto finito $\{\mathbf{F}_1, \mathbf{F}_2 \ldots \mathbf{F}_Z\}$ se obtiene un "***iterated functions system"*** (IFS). En ese caso, la dimensión del "shift space" es 0.

Adicionalmente, lo que conocemos del sistema, cualquiera sea su tipo, es una serie de observaciones de una variable x del sistema de forma que

$$x(n+1) = \varphi[x(n), x(n-1), \ldots x(n-T)] + \delta(n)$$

siendo φ una cierta función y δ un ruido, llamado ***ruido observacional***.

AVII.1.2 Estados de equilibrio

En el caso determinístico, un vector de estado $\bar{\mathbf{x}}$ se dice de ***equilibrio*** (o estacionario) si $\mathbf{F}(\bar{\mathbf{x}}) = \bar{\mathbf{x}}$. Por lo tanto, en el punto de equilibrio, la trayectoria del sistema degenera en el punto en sí. Si el sistema tuviese entradas $\mathbf{u}(n)$ el estado de equilibrio sería tal que

$$\mathbf{F}(\bar{\mathbf{x}}, \mathbf{u}(n)) = \bar{\mathbf{x}}.$$

AVII.1.3 Estabilidad

Para un sistema dinámico determinístico, existen varias definiciones de estabilidad. El punto de equilibrio $\bar{\mathbf{x}}$ se dice

a) ***uniformemente estable*** si $\forall \varepsilon > 0 \quad \exists \delta > 0 : \left\| \mathbf{x}(0) - \bar{\mathbf{x}} \right\| < \delta \Rightarrow \left\| \mathbf{x}(n) - \bar{\mathbf{x}} \right\| < \varepsilon \quad \forall n > 0$

O sea, la trayectoria del sistema se puede hacer permanecer en un pequeño entorno del punto de equilibrio si el estado inicial $\mathbf{x}(0)$ es cercano a $\bar{\mathbf{x}}$

b) ***asintóticamente estable*** si cumple la condición a) y además $\exists \delta > 0 : \left\| \mathbf{x}(0) - \bar{\mathbf{x}} \right\| < \delta \Rightarrow \lim_{n \to \infty} \mathbf{x}(n) = \bar{\mathbf{x}}$

Es decir, si el estado inicial $\mathbf{x}(0)$ es suficientemente próximo a $\bar{\mathbf{x}}$ entonces la trayectoria descripta por el vector $\mathbf{x}(n)$ se aproximará a $\bar{\mathbf{x}}$ con el transcurso del tiempo.

c) ***Globalmente asintóticamente estable*** si cumple a) y b) para todo $\mathbf{x}(0)$.

Esta definición implica que el sistema no puede tener otros estados de equilibrio y requiere que cada trayectoria del sistema sea acotada en todo momento. En otras palabras, estabilidad asintótica implica que el sistema va a alcanzar un estado estacionario cualesquiera sean sus condiciones iniciales [5].

Estas definiciones suponen se hacen sin tomar en cuenta las entradas al sistema (estímulos externos) por lo que se pueden aplicar a aquellos sistemas en los cuales se mantienen constantes (o casi constantes) dichas entradas mientras varía el tiempo n.

AVII.1.4 Sistemas disipativos

Un sistema dinámico determinístico se dice ***disipativo*** si

$$\int_S [\mathbf{F}(\mathbf{x}) \cdot \mathbf{n}] dS < 0 \quad \forall \mathbf{x} \in S$$

siendo S una región de superficie S contenida en el espacio de estados y $\mathbf{n}$ el vector normal en cada punto de S (apuntando hacia fuera).

En los sistemas disipativos, los volúmenes de condiciones iniciales tienden a contraerse con el tiempo.

AVII.1.5 El espacio de reconstrucción

Sea un sistema dinámico determinístico del cual se conoce una serie temporal $x(n)$ de observaciones y tal que siempre se conocen los N-1 valores anteriores $x(n-1),...x(n-N+1)$, siendo N no necesariamente el orden del sistema, y sea $\mathbf{x}(n) = [x(n-1),...x(n-N+1)]$. El espacio cuyas dimensiones son $x_1, x_2 \ldots x_{N-1}$ [33] es por definición el ***espacio de reconstrucción*** [11]. Los puntos determinados por cada uno de esos vectores de coordenadas en el espacio de reconstrucción siguen una curva llamada la ***trayectoria reconstruida*** de la serie[34]. Los vectores $\mathbf{x}(p) = [x(p-1),...x(p-N+1)]$ y $[x(p),...x(p-N+2)]$ que se van obteniendo en el espacio de reconstrucción al variar p están relacionados por la estructura de la serie temporal $x(n)$. En la figura AVII.1 se puede ver un ejemplo para el caso $N = 3$.

[33] x_1 no es lo mismo que $x(1)$: la idea es que hay una dimensión por cada componente del vector **x.** El eje x_1 puede considerarse como el x, el x_2 como el y, etc.

[34] En realidad, es la trayectoria reconstruida del sistema a partir de los datos de la serie, pero, como en otros casos, se termina hablando de la trayectoria *de la serie* por un abuso del lenguaje.

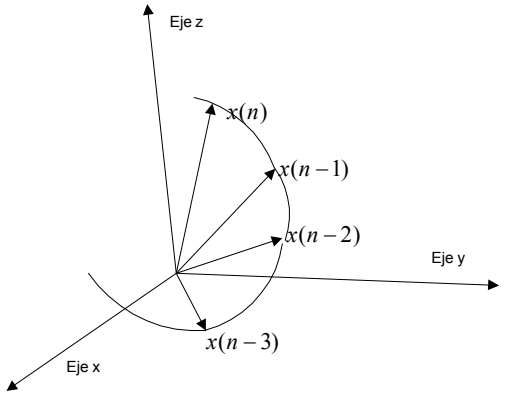


Figura AVII.1

La estructura de la serie temporal restringirá las posibles ubicaciones de los vectores **x**(*p*) mencionados, limitando la trayectoria de la serie a un espacio de mucho menos dimensionalidad que el de reconstrucción. Este subespacio se llama ***subespacio de la señal***[35]. Se puede obtener de esta forma una gran reducción de dimensionalidad: por ejemplo, si $x(n)$ es periódica el número de dimensiones necesarias del espacio de reconstrucción para representar esa señal es el largo del período de la señal en muestras, mientras que, como su trayectoria será una curva cerrada (por ser periódica $x(n)$), se requerirán muchas menos dimensiones para representar la trayectoria reconstruida. Por ejemplo, una $x(n)$ sinusoidal tiene asociada una trayectoria que es una elipse. No importa cual sea el largo del período medido en muestras, la elipse se puede representar en un espacio de dos dimensiones. Por otra parte, un ruido aleatorio no tiene estructura temporal y llena un espacio de cualquier dimensión. Lo mismo sucede con los patrones de datos estáticos: precisamos un espacio de reconstrucción de dimensión máxima, o sea la dimensión del vector **x**(*n*), ya que no conocemos a priori ninguna relación entre los componentes de los datos. Si las dimensiones del espacio de reconstrucción están bien elegidas, la trayectoria reconstruida no se corta consigo misma y existe una correspondencia 1 a 1 entre $x(n)$, la trayectoria real y la reconstruida. La dimensión usada del espacio de reconstrucción depende de los fines para los que se esta haciendo la reconstrucción de la trayectoria. Por ejemplo, en el caso anterior, si reconstruimos la trayectoria de la señal solo a partir de los ejes x e y, obtendríamos la curva (trayectoria) B (Figura AVII.2), con otras propiedades. Este subespacio de la señal (llamado así por [11]) corresponde al "embedding" que se menciona en el resto del trabajo. Nos atuvimos así a la denominación más utilizada en la bibliografía para el tema.

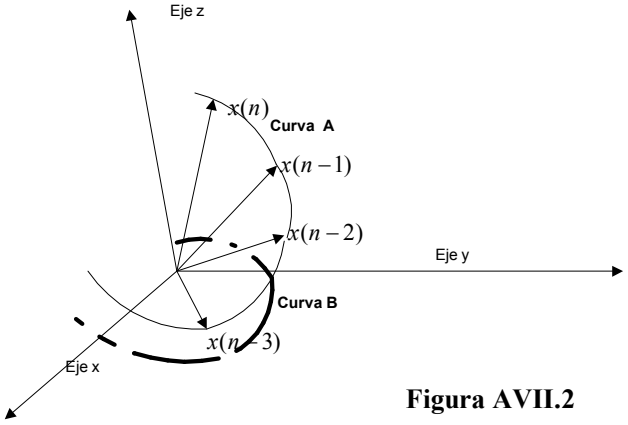


Figura AVII.2

[35] Lo de "señal" proviene de la aplicación de estos conceptos al tratamiento digital de señales. En el contexto actual podríamos llamarlo "subespacio de la serie", pero preferimos mantener la denominación manejada en la bibliografía.

Más adelante se verá que se puede utilizar esta reducción en la dimensión del espacio de reconstrucción para reducir a su vez la dimensionalidad de la entrada a la red neuronal a utilizar. En el caso de un sistema dinámico estocástico se demuestra que la situación es básicamente la misma, existiendo una correspondencia biunívoca entre la serie y los puntos de la trayectoria reconstruida**,** para cada valor de ω (salvo tal vez para un conjunto de valores de ω finito) [65].

AVII.2 Atractores y "manifolds"

Entendemos por "***manifold***" a una superficie k dimensional embebida[36] en el espacio de estados y que está definida por un conjunto de ecuaciones $M_j(\mathbf{x}) = 0 \quad j = 1,\ldots k \quad k < N$ siendo $M_j(\mathbf{x})$ cierta función de los $\mathbf{x}$ pertenecientes al espacio de estados [www.mathworld.com].

En los sistemas disipativos[37] es común que existan "manifols" de estados que atraen hacia sí las trayectorias del sistema que pasen por cierto entorno suyo, llamados ***atractores***. Estos atractores son conjuntos acotados del espacio de estados hacia los cuales convergen los volúmenes de condiciones iniciales[38] del espacio de estados al crecer t. El atractor puede ser un solo punto y se dice que es un ***atractor puntual*** , puede ser una trayectoria periódica: ***ciclo limite*** ("limit cycle") que es estable en el sentido de que trayectorias cercanas se aproximan a él asintóticamente o ser un ***atractor extraño***.

Esta idea de atractor se podría generalizar al caso de sistemas dinámicos estocásticos, si bien la bibliografía no menciona nada al respecto podríamos pensarlo como que las trayectorias se aproximan a él de forma que la probabilidad de que el estado del sistema este fuera de un entorno del atractor al $t \to \infty$ es 0.

Cada atractor tiene asociada una región llamada ***dominio de atracción*** tal que cualquier punto (condición inicial del sistema) perteneciente a ella hace que el estado del sistema derive hacia ese atractor.

Sea A el jacobiano de **F** evaluado en $\bar{\mathbf{x}}$. Si todos los valores propios de A son menores a 1 en valor absoluto, entonces $\bar{\mathbf{x}}$ se dice ***hiperbólico.*** Veremos el significado de estos atractores al estudiar los problemas de entrenamiento asociados a las redes recurrentes.

Un sistema dinámico puede poseer varios atractores. A partir de un estado inicial dado, se convergirá a alguno de ellos. Las condiciones para que se dé esa convergencia son las llamadas condiciones de Lyapunov [8].

En general, lo que se tiene es un conjunto de observaciones temporales (mediciones) de alguna variable del sistema, que habrá ido evolucionando en el tiempo hacia uno de sus atractores, por lo que el problema pasa a centrarse en el estudio de "el" atractor del sistema, es decir, de aquel hacia el cual han ido evolucionando las observaciones.

AVII.2.1 Atractores extraños y caos

Los ***atractores extraños*** ("strange attractors") son característicos de los sistemas dinámicos determinísticos de orden mayor que 2. Un sistema con un atractor extraño exhibe un comportamiento caótico: para un conjunto de condiciones iniciales que están en el dominio de atracción del atractor, es determinístico (su comportamiento esta gobernado por reglas fijas) aunque muestra un comportamiento tan complicado que parece aleatorio. Definimos un atractor extraño como aquel atractor tal que las órbitas (trayectorias) que pasen por el dominio de atracción y que tengan condiciones iniciales semejantes, tienden a separarse con el tiempo. Un sistema que posea un atractor extraño se dice ***caótico.*** Una definición alternativa es decir que *el sistema es caótico si tiene algún exponente de Lyapunov* (AVII.3.4) *positivo* [5].

En los sistemas caóticos determinísticos la aleatoriedad observada no desaparece por más que se recopilen datos del sistema, es decir, que las predicciones que se hagan del siguiente estado del sistema tienen el mismo error esperado, sea cual sea el número de observaciones conocidas; adicionalmente existe una sensibilidad extrema a las condiciones

[36] Ver AVII.4.2

[37] La teoría se desarrolla suponiendo que si el sistema tiene entradas, estas no varían (o tienen una variación no significativa) durante el tiempo en que se deja evolucionar el sistema hacia un estado estable.

[38] Las condiciones iniciales incluyen aquí las posibles entradas en el momento inicial.

iniciales: si dos sistemas idénticos comienzan con condiciones iniciales ligeramente distintas, $\mathbf{x}$ y $\mathbf{x}+\boldsymbol{\varepsilon}$ sus trayectorias van a divergir entre sí y su separación va a aumentar exponencialmente con el tiempo.

En el caso de los sistemas estocásticos, no encontramos en la bibliografía una definición de "caótico", en especial referida a los coeficientes de Lyapunov, aunque en [45] se menciona que un sistema puede ser caótico y estocástico a la vez. Por otra parte, tampoco encontramos una definición de exponente de Lyapunov "general" o "promedio", es decir, que tenga en cuenta todas las realizaciones del proceso aleatorio que se correspondería con la idea de "trayectoria" de un sistema determinístico.

Un ejemplo: el atractor de Lorenz

El sistema de Lorenz[39] de tres dimensiones es descrito por las ecuaciones diferenciales:

$$\begin{cases} \dfrac{\partial x}{\partial t} = \sigma[x(t) - y(t)] \\ \dfrac{\partial y}{\partial t} = x(t)[r - z(t)] - y(t) \\ \dfrac{\partial z}{\partial t} = x(t)y(t) - bz(t) \end{cases}$$

donde b, r, y σ son parámetros. Partiendo de un valor inicial $[x_0, y_0, z_0]$ y utilizando ciertos valores para los parámetros r, σ y b se obtienen tres series temporales (una para cada variable de estado). La trayectoria descripta para el caso típico $r = 28$ $\sigma = 10$ y $b = 8/3$ puede ser vistas desde distintos ángulos en la siguiente ilustración:

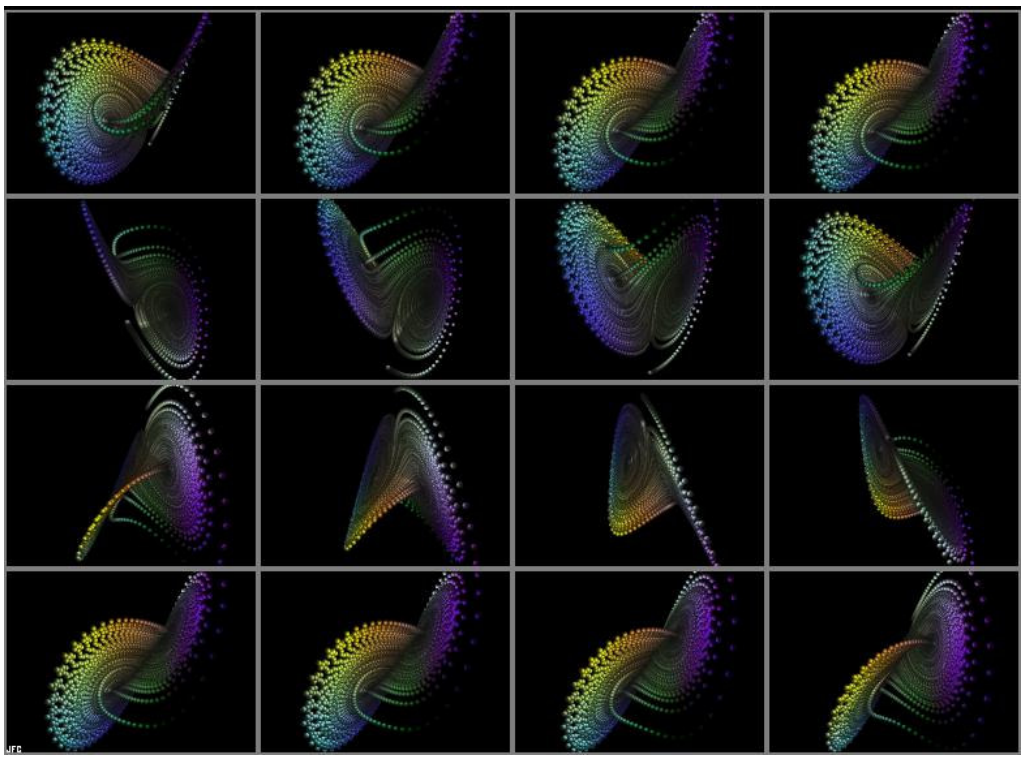


Figura AVII.3

[39] En la bibliografía a veces aparece como *Lorentz*

Un sistema dinámico puede ser caótico y estocástico a la vez: si por ejemplo le sumamos un ruido aleatorio a la coordenada x del atractor de Lorenz, obtenemos un sistema dinámico estocástico y caótico. En ese caso, el aspecto del atractor de Lorenz deja de ser el de la figura anterior y pasa a ser un conjunto de puntos más o menos borroso (en la acepción común de la palabra) que de todos modos esta relacionado con el original: por ejemplo, la dimensión del nuevo atractor es la dimensión del original más la del espacio de ruidos añadidos (en este caso, 1) [45].

AVII.3 Características invariantes

Las características invariantes de un atractor de un sistema dinámico (o simplemente ***invariantes***) son aquellas propiedades que se mantienen luego de haber realizado una transformación no lineal "suave" [40]a los estados del sistema. Son invariantes por ejemplo el "delay" óptimo y la dimensión del "embedding" [41]. Otras propiedades características de un sistema dinámico son las dimensiones generalizadas, la entropía de Kolmogorov, los exponentes de Lyapunov, la dimensión de Kaplan-Yorke y el horizonte de predecibilidad [42]. Estas características permiten caracterizar a un sistema y por lo tanto comparar qué tanto se asemejan los comportamientos de dos sistemas. Este criterio de comparación es muy útil al momento de realizar la reconstrucción dinámica del sistema (Ver AVII.6) ya que permite discernir qué tan bien un modelo reproduce la dinámica del sistema.

En el caso estocástico, lo que se obtiene es una función de densidad de probabilidad empírica para cada uno de los invariantes[43] y el problema es entonces comparar las funciones de densidad de cada uno de los invariantes generadas por el modelo y las del sistema original. A la fecha, muy pocos trabajos se han publicado al respecto [63]

AVII.3.1 Dimensiones generalizadas y entropías

Los atractores de un sistema dinámico disipativo determinístico generalmente poseen una estructura fractal (en el sentido de que son similares a partes de sí mismos). Las ***dimensiones generalizadas*** son una clase de cantidades que caracterizan esta fractabilidad y son invariantes. Algunas de dichas dimensiones son la dimensión de información y la dimensión de correlación que se definen a continuación.

AVII.3.2 Las dimensiones generalizadas

El estudio de las dimensiones generalizadas de un atractor nos proporciona información sobre la estructura que el mismo presenta. Para los sistemas con varios grados de libertad (aun siendo determinísticos) una trayectoria en el espacio de estados se presenta como una figura altamente irregular. El estudio de las dimensiones generalizadas nos permite descubrir ciertas características dentro de esa figura. Para ello dividamos el espacio de estados (el cual suponemos que es acotado) en $N(\varepsilon)$ hipercubos de lado ε y estudiaremos la distribución de probabilidades p_i a lo largo del atractor cuando $\varepsilon \to 0$. La distribución de probabilidad p_i es definida como la probabilidad de que haya un punto de la trayectoria en el i-ésimo hipercubo:

$$p_i = \lim_{N\to\infty} \frac{N_i}{N}$$

[40] un homeomorfismo: una transformación continua biunívoca.
[41] Cuando se habla de dimensión del embedding a secas, se esta refiriendo a la mínima dimensión del espacio de reconstrucción tal que la trayectoria reconstruida no se corta consigo misma.
[42] Dado que todos estos invariantes se calculan por métodos numéricos que utilizan el conjunto de datos dados, el cual es finito, y como las dimensiones se definen sobre conjuntos infinitos, lo que se hace es hacer extrapolaciones, que pueden fallar por varias causas, y dar estimaciones poco confiables [29]. No discutiremos aquí la calidad (robustez) de cada estimación, y nos limitamos a utilizar las dadas por los paquetes de software más conocidos.
[43] En el caso de los exponentes de Lyapunov se podría considerar la distribución del mayor exponente.

donde N es el número total de puntos de la trayectoria y N_i es el número de puntos contenidos en el i-ésimo hipercubo de lado ε.

Se definen entonces las ***dimensiones generalizadas*** (o de Renyi) del atractor como

$$D_q = \frac{1}{q-1}\lim_{\varepsilon\to 0}\frac{Log\sum_{i=1}^{N(\varepsilon)} p_i^q}{Log\,\varepsilon} \qquad \textbf{(AVII.1)}$$

Cuando q = 0, 1 y 2 las dimensiones correspondientes son la ***dimensión fractal*** D_0 o de Hausdorff-Besicovitch, la ***dimensión de información*** D_1 y la ***dimensión de correlación*** D_2 [56]. La dimensión D_0 es utilizada para estimar la dimensión que debe tener el subespacio de la señal en la Sección III. Por otra parte, D_2 puede aproximarse como

$D_2 = \lim_{\varepsilon\to 0}\frac{LogC(\varepsilon)}{Log\,\varepsilon}$ siendo $C(\varepsilon) = \frac{1}{N(N-1)}\sum_{j=1}^{N}\sum_{\substack{i=1\\ i\neq j}}^{N} u(\varepsilon - \|\mathbf{x}_i - \mathbf{x}_j\|)$ con $u(\bullet)$ la función de Heaviside y $\mathbf{x}_i$ y $\mathbf{x}_j$ dos puntos de la trayectoria. La dimensión de correlación es uno de los invariantes cuyo cálculo se proporciona en forma estandard en la mayoría de las herramientas de análisis de series temporales (por ejemplo, el DATAPLORE y el TISEAN).

Alternativamente, algunos (por ejemplo [5]) definen las dimensiones generalizadas como sigue. Dado un atractor de un sistema dinámico determinístico, una medida $C(q,r)$ de la probabilidad de que los puntos en el espacio de estados $\mathbf{x}(n)$ y $\mathbf{x}(k)$ [44] (situados en el dominio de atracción de un atractor) estén separados por una distancia *r* es por definición la ***función de correlación*** [5, Cap. 14]:

$$C(q,r) = \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{N-1}\sum_{\substack{k=1\\ k\neq n}}^{N} u\left(r - \|\mathbf{x}(n) - \mathbf{x}(k)\|\right)\right)^{q-1}$$

siendo N el número de puntos de datos (valores observados de la serie), q un parámetro real no negativo y $u(\bullet)$ la función de Heaviside (escalón unitario).

Esta función de correlación es un invariante del atractor. Adicionalmente, a los efectos prácticos se suele considerar su comportamiento para valores pequeños de r. En ese caso, cuando $r\to 0$, si existe el límite, es [5]

$$C(q,r) \approx r^{(q-1)D_q} \qquad \textbf{(AVII.2)}$$

Al valor D_q lo definen como la ***dimensión fractal*** [45] del atractor. Tomando logaritmos a ambos lados de AVII.2 y considerando que la aproximación tiene validez cuando r es muy pequeño, se tiene

[44] Estos puntos son los equivalentes a los $\mathbf{P}(t)$ de AVII.1.1

[45] Haykin llama dimensión fractal a la familia de dimensiones generalizadas, aunque generalmente se conoce como dimensión fractal de un objeto a su dimensión de Hausdorff-Besicovitch, D_0

$$D_q = \lim_{r\to 0} \frac{Log\left[C(q,r)\right]}{(q-1)Log(r)}$$

Cuando q = 0 se obtiene la dimensión de Hausdorff-Besicovitch, cuando $q \to 1$ la dimensión de información y si q = 2 , D_2 es la dimensión de correlación del atractor.

Interpretación de las dimensiones generalizadas:

La diferencia entre las distintas dimensiones esta en lo que consideran cada una: a medida que aumenta q son tenidos en cuenta cada vez más las regiones más "densas" (en número de puntos) del atractor [56]. Se puede demostrar a partir de la definición AVII.1 que $D_0 = \lim_{\varepsilon\to 0} \frac{LogN(\varepsilon)}{Log(1/\varepsilon)}$ siendo $N(\varepsilon)$ el número mínimo de hipercubos de lado ε necesarios para cubrir todo el conjunto de puntos del atractor. La dimensión D_0 daría una idea de qué tan "recortado" es el atractor. Por ejemplo, para un cuadrado es D_0 =2 mientras que para el atractor de Lorenz esta entre 2 y 3. Esta dimensión da además una medida de los grados de libertad del sistema. Para que el sistema sea previsible (se puedan prever sus estados futuros) en el caso determinístico, ese valor debe ser pequeño. Para los sistemas que presentan dimensiones D_0 altas puede ser más apropiado un modelado estocástico que determinístico. Por otra parte, D_1 es el límite inferior de D_0 y en general suelen tener valores similares [56].

AVII.3.3 Las entropías generalizadas

La ***entropía de Kolmogorov-Sinai*** puede ser definida como la tasa media de pérdida de información sobre las condiciones iniciales del sistema durante su evolución. La predecibilidad temporal de un sistema determinístico es inversamente proporcional a su entropía de Kolmogorov, es decir, no se pueden hacer previsiones futuras confiables para tiempos grandes debido a la pérdida de información asociada a la entropía . Formalmente se la define como sigue. Supongamos que tenemos el espacio de estados dividido en hipercubos (cajas) de tamaño r^d y sea $P_{i_0\ldots i_{d-1}}$ la probabilidad conjunta de que el estado $\mathbf{x}(t=0)$ este en la caja i_0, $\mathbf{x}(t=1)$ en la i_1,...y $\mathbf{x}(t=(d-1)\Delta t)$ en la i_{d-1} siendo Δt el intervalo de muestreo. Entonces la entropía de Kolmogorov-Sinai[46] es [56]

$$K = -\lim_{\substack{\Delta t\to 0\\ r\to 0\\ d\to\infty}} \frac{1}{d\Delta t} \sum_{i_0,\ldots i_{d-1}} P_{i_0\ldots i_{d-1}} Log(P_{i_0\ldots i_{d-1}})$$

donde la suma se realiza sobre todas las posibles cajas $i_0, i_1, \ldots i_{d-1}$. Esta entropía puede ser usada para clasificar sistemas dinámicos, ya que es infinita para sistemas estocásticos, positiva y finita para sistemas caóticos y nula en otro caso [49]. Por otra parte puede ser calculada como la suma de los exponentes positivos de Lyapunov [39]. En forma general, se definen las ***entropías generalizadas*** como [56]

$$K_q = -\lim_{\substack{\Delta t\to 0\\ r\to 0\\ d\to\infty}} \frac{1}{d\Delta t(q-1)} Log \sum_{i_0,\ldots i_{d-1}} (P_{i_0\ldots i_{d-1}})^q$$

AVII.3.4 Los exponentes y el espectro de Lyapunov

[46] También conocida como de Kolmogorov

Los exponentes de Lyapunov pueden ser definidos como la medida (tasa) promedio a la que divergen (o convergen) dos trayectorias inicialmente vecinas en el dominio de atracción de un atractor y caracterizan la sensibilidad de un sistema determinístico a las condiciones iniciales. Para visualizarlos imaginemos la evolución de un esferoide de radio inicial $\varepsilon(0)$ en un espacio n-dimensional. A medida que los puntos se desplazan a lo largo de la trayectoria del atractor, el esferoide va a evolucionar en un elipsoide, ya que cada dimensión (eje) tiene asociada una tasa de variación distinta. Supongamos que el largo del eje principal del elipsoide según la dirección i es $\varepsilon_i(n)$. En ese caso, el i-ésimo exponente de Lyapunov se define como

$$\lambda_i = \lim_{n\to\infty} Supremo\left[\frac{1}{n}\log_2\frac{\varepsilon_i(n)}{\varepsilon_i(0)}\right]$$

donde n representa el tiempo. Por utilizar logaritmos en base dos, el exponente se mide en bits/seg., si se utilizaran logaritmos naturales, sería en nats/seg. [46].

El signo de los exponentes de Lyapunov tiene un significado especial. Un exponente positivo representa la divergencia de las trayectorias de un sistema dinámico determinístico que se da en la dimensión considerada ($\lambda_i > 0$ implica que hay divergencia en la dirección i-ésima), mientras que un exponente negativo representa convergencia. Para que un sistema sea considerado caótico deberá tener al menos un coeficiente de Lyapunov positivo. Si el sistema tiene más de un exponente positivo, el mayor es el que tiene, típicamente, más influencia. Un exponente nulo significa que no hay cambios (que no hay aumento ni disminución en esa dimensión del esferoide) en esa dirección.

En un sistema realizable físicamente (o sea que puede existir en la Naturaleza), que será disipativo, la suma de los coeficientes de Lyapunov es no positiva [5].

Tal como los definimos, los exponentes de Lyapunov son cantidades globales, ya que son un límite, y por lo tanto no proveen información sobre la tasa local de divergencia de trayectorias inicialmente próximas, que pueden tener cualquier signo, dependiendo de la región del atractor donde se esté. En términos de predicción, existen entonces áreas donde se pueden realizar predicciones a corto plazo (varios pasos en el futuro) y otras donde esto es imposible debido al crecimiento exponencial de los errores. Este fenómeno de cambio de tasas de divergencias locales ha sido llamado ***fenómeno de los exponentes de Lyapunov efectivos (o locales)*** [62].

El ***espectro de Lyapunov*** está formado por $\lambda_i \quad i = 1,\ldots d_E$ siendo cada λ_i un ***exponente*** global ***de Lyapunov.***

En nuestro caso calculamos el espectro de Lyapunov global, utilizando las rutinas del TISEAN aunque el cspX lo calcula en forma local. El DATAPLORE también lo calcula globalmente.

Caso estocástico

En el caso de sistemas dinámicos estocásticos lo que se obtiene es una función de densidad de probabilidad de cada uno de los invariantes (ver AVII.3), entre ellos, los exponentes globales de Lyapunov. Sin embargo, la interpretación del significado de los exponentes (tanto globales como locales) en ese caso es difícil, tanto a nivel teórico como práctico [62] y más aún si se aplican directamente los algoritmos de determinación de los exponentes de Lyapunov (tales como el de Sano y Sawada, que es el usado por el Tisean) a una serie temporal proveniente de un sistema dinámico estocástico. En ese caso, se pueden obtener exponentes positivos que en realidad son falsos [Tanaka, Aihara y otros, citados por [62]]. Una forma parcial de evitar esta determinación de exponentes positivos falsos es utilizando una estimación del máximo exponente de Lyapunov basada en intervalos de confianza tal como la propuesta por Gençay [Gençay citado por [62]]. Por otra parte, dado que para cada estado inicial se tiene una densidad de probabilidad de los posibles estados en los que puede estar el sistema en el instante n, la sensibilidad del sistema respecto a las condiciones iniciales puede pensarse como la distancia (en el sentido de Kullback-Leibler) entre las densidades de probabilidad en el instante n: la correspondiente al estado inicial $\mathbf{x}$ y la correspondiente al $\mathbf{x}+\boldsymbol{\varepsilon}$. En contraste con los exponentes de

Lyapunov, esta sensibilidad es local, y depende del estado inicial [62]. En ese caso, tomando como distancia entre las distribuciones p y q a:

$$K(p\|q) = D(p\|q) + D(q\|p)$$

siendo D la distancia de Kullback-Leibler entre p y q[47].

Se puede aproximar $K(p\|q)$ como [62]:

$$K(\mathbf{x_n}, \mathbf{x_n} + \boldsymbol{\varepsilon}) \approx \boldsymbol{\varepsilon}^T \mathbf{I}(\overline{\mathbf{x_n}})\boldsymbol{\varepsilon}$$

donde

$\mathbf{I}(\overline{\mathbf{x_n}})$ la matriz de información de Fisher:

$$\mathbf{I}(\overline{\mathbf{x}}_n) = \left\{ I_{ij} = \left\langle \frac{\partial Log\,\rho(\mathbf{x}_{n+1}|\overline{\mathbf{x}}_n)}{\partial \overline{x}_i} \frac{\partial Log\,\rho(\mathbf{x}_{n+1}|\overline{\mathbf{x}}_n)}{\partial \overline{x}_j} \right\rangle_{\mathbf{x}_{n+1}} \right\}$$

$$n - m + 1 \le i, j \le n$$

m orden del sistema

$\rho(\mathbf{x}_{n+1}|\overline{\mathbf{x}}_n)$ la distribución de probabilidad condicional de que le sistema se encuentre en el estado n+1 si antes estuvo en $\mathbf{x}_n, \mathbf{x}_{n-1}, \ldots \mathbf{x}_{n-m+1}$. Este planteo tiene el inconveniente de que requiere conocer las distribuciones condicionales de los estados; una alternativa práctica aparece en [62] donde se da un método para el análisis de sensibilidad a las condiciones iniciales utilizando una red de mezcla de distribuciones (MDN, Ver [1] Cap. 2) para representar $\rho(\mathbf{x}_{n+1}|\overline{\mathbf{x}}_n)$.

AVII.3.5 La dimensión de Kaplan-Yorke

Dado el espectro de Lyapunov, Kaplan y Yorke sugirieron la dimensión (llamada de Kaplan-Yorke o de Lyapunov):

$$D_{KY} = K + \frac{\sum_{i=1}^{K} \lambda_i}{|\lambda_{K+1}|}$$

donde K es un entero que cumple

$$\begin{cases} \sum_{i=1}^{K} \lambda_i > 0 \\ \sum_{i=1}^{K+1} \lambda_i < 0 \end{cases}$$

Para un proceso caótico determinístico, generalmente es $D_{KY} = D_2$ [5]. Sin embargo, según el procedimiento que se utilice para calcular D_{KY}, pueden D_{KY} y D_2 ser o no muy próximos, dependiendo especialmente de si en los dos casos se tomaron a la vez en cuenta los ruidos en los datos (algunas estimaciones de D_{KY} lo tienen en cuenta y otras no) [52].

[47] K fue definida así para que fuera una medida simétrica

AVII.3.6 El horizonte de predecibilidad

Como ya se dijo, debido a la sensibilidad a las condiciones iniciales de un sistema caótico determinístico, dos puntos que originalmente eran cercanos se apartan exponencialmente al pasar el tiempo. A mayor exponente, menor el número de muestras (valores de la serie temporal) que pueden ser predichos adecuadamente hasta que las dos trayectorias diverjan significativamente. Por lo tanto, los sistemas caóticos determinísticos son predecibles en el corto plazo pero no predecibles a largo plazo. La predecibilidad[48] a corto plazo es definida como ***horizonte de predecibilidad*** y esta principalmente determinada por el mayor exponente de Lyapunov. Es más, estimamos el largo de esta predecibilidad como [5]:

$$HOP \approx \frac{1}{\lambda_{MAX}}$$

para el caso de procesos caóticos. Para procesos determinísticos no caóticos, el horizonte no está definido, o en teoría sería infinito, aunque los errores de predicción se van acumulando y hacen que finalmente las trayectorias (real y predicha) diverjan. Para sistemas dinámicos estocásticos, el horizonte no puede pensarse como cuantos pasos puedo predecir sin cometer grandes errores, ya que la aleatoriedad en los estados hace que lo más que se pueda predecir es la media de la distribución $P(\mathbf{x}_{n+1}|\mathbf{x}_n)$ aunque el valor observado diste mucho de ella. De todos modos, dado que el horizonte de predecibilidad esta asociado a las predicciones a largo plazo, las que se suelen obtener iterando un predictor a corto plazo, se pueden hacer algunas mejoras a este tipo de predicción, tales como las propuestas por Judd y Small con el método ***ΨΦ*** [63]. En ese método se pretende corregir, mediante una transformación (por ejemplo lineal) de los valores predichos, la influencia de los errores de predicción sistemáticos que pudiera cometer el predictor ***Φ***. O sea: se predicen una serie de valores en forma iterada mediante el predictor ***Φ*** y luego se estiman los parámetros del corrector ***Ψ*** de forma de minimizar el error cuadrático entre los valores deseados y los corregidos, y finalmente utilizar este corrector para los corregir los valores a predichos en el futuro. Por ejemplo, si $\mathbf{f}_n$ son los valores deseados a predecir, $\mathbf{z}_n$ los valores predichos para el instante n, y si se toma $\Psi(\mathbf{z}_n) = \mathbf{A}\mathbf{z}_n$ siendo $\mathbf{A}$ una matriz, entonces la estimación de $\mathbf{A}$ tal que $\|\mathbf{f}_n - \mathbf{A}\mathbf{z}_n\|^2$ es mínimo es

$$\widehat{\mathbf{A}} = (\mathbf{Z}^T\mathbf{Z})^{-1}\mathbf{Z}^T\mathbf{F}$$

siendo $\mathbf{Z}$ una matriz tal que la columna n-ésima es la predicción para el instante n-ésimo y $\mathbf{F}$ es una matriz tal que su columna n-ésima es el valor deseado para el instante n-ésimo (Ver [63]).

AVII.4 Los "embebimientos" ("embeddings") y el teorema de Takens-Mañé

El atractor de un sistema determinístico es obtenido al dibujar la evolución del vector de estados en el espacio de estados. Esto es posible solo cuando se tiene acceso a todas las variables de estado del sistema, lo cual es raro; en su lugar, lo que se suele es tener alguna función M de medida definida sobre los estados del sistema, cuyos valores forman una única serie temporal (de medidas) $s(n) = M(\mathbf{x}(n))$. Takens y Mañé[49] probaron, independientemente, que a partir de esta serie temporal se puede reconstruir el atractor original, utilizando para ello el ***método de los "delays"*** ("delayed coordinate method"). Posteriormente se demostró que esto también es válido para el caso estocástico ([65]). En el caso determinístico y también en el estocástico, la reconstrucción de la dinámica del sistema a partir de una serie temporal de observaciones es un problema inverso "ill-posed". El problema directo es, dada la dinámica describir las trayectorias que seguirá, y el problema inverso, dadas las trayectorias seguidas, describir la dinámica. El problema inverso es "ill-posed" debido a que, dependiendo de la calidad de los datos, la solución hallada puede no ser estable, no ser única o incluso no existir. Para disminuir los problemas asociados a esa calidad de "ill-posed", lo que se puede

[48] Mejor dicho, cuantos pasos en el futuro se puede predecir.

[49] Mañé, Ricardo: matemático uruguayo fallecido que probó el teorema simultáneamente con Takens. La bibliografía suele referirse a ese teorema como *de Takens* a secas o *del embedding*.

hacer es incluir en el modelo predictivo reconstruido alguna forma de conocimiento previo sobre el mapeo entrada→salida [5]. Una forma de hacerlo es al utilizar los parámetros del "embedding": dimensión y "delay" óptimo obtenidos a partir de los datos $s(n)$ observados del sistema determinístico. Esta idea también se utiliza en el caso estocástico, como se verá.

AVII.4.1 El método de los "delays"

A partir de la serie x_n se forman los vectores

$$\mathbf{x}(n) = [x_n, x_{n-\tau} \ldots x_{n-(d_E-1)\tau}]$$

donde

d_E es la llamada ***dimensión del "embedding "***[50]

τ es el ***"delay" óptimo*** (o más brevemente, "delay") del "embedding", que es tomado como un múltiplo del tiempo de muestreo (intervalo entre muestras, que como se dijo, tomamos igual a 1).

AVII.4.2 La noción de "embebimiento" ("embedding")

Un conjunto X esta ***embebido*** en otro Y cuando las propiedades (de conectividad, algebraicas, etc.) de Y restringidas a X son las mismas que las propiedades de X. Por ejemplo, los racionales están embebidos en los reales y los enteros en los racionales. Decimos que hay un embebimiento ("***embedding"***) de X a Y o de Y sobre X. En el caso geométrico, una esfera esta embebida en $\mathbf{R}^3$. [www.mathworld.com].

Asociada a esta idea aparece la siguiente definición más formal:

Una función $f : X \to Y$ *es un "embedding" si es continua con inversa* $f^{-1} : f(X) \to X$ *continua.*

Por un abuso de lenguaje, hablamos de la "dimensión del embedding" para referirnos a la dimensión del co-dominio (Y) del "embedding".

AVII.4.3 El teorema de Takens-Mañé

El teorema asume la existencia de d_E y τ, y en ese caso[51], el vector (la función vectorial) $\mathbf{x}(t) = [x_t, x_{t-\tau} \ldots x_{t-(d_E-1)\tau}]$ es un "embedding" del atractor sobre el conjunto de los puntos $\mathbf{x}(t)$. Aprovechando las propiedades de continuidad que tienen los "embeddings" [52], se puede predecir $x_{t+\tau}$ a partir de $\mathbf{x}(t)$, si los parámetros d_E y τ son los adecuados, aunque el teorema no da mayores detalles de ellos, salvo que es suficiente que $d_E \geq 2D+1$ siendo D la dimensión fractal del atractor. Esto significa que, para casi cualquier conjunto de

---

[50] En AVII.1.5 tomamos $\tau = 1$ por lo que en ese era $d_E = N$ y coincide con la noción intuitiva de dimensión del subespacio de la señal.

[51] El teorema exige condiciones adicionales muy poco restrictivas, por lo que la bibliografía utilizada generalmente las ignora como por ejemplo que el sistema no contenga trayectorias periódicas de periodo $\boldsymbol{\tau}$ o $2\boldsymbol{\tau}$ y que contenga solo un número finito de puntos de equilibrio.

[52] Ya que son aplicaciones continuas por definición.

observaciones (serie temporal) del sistema, podemos responder una amplia gama de interrogantes sobre la dinámica del sistema original con solo estudiar la dinámica en el espacio definido por los valores demorados de la serie.

En general, por otro abuso del lenguaje, se habla de la "dimensión del atractor", para referirse a la dimensión del subespacio de la señal, d_E.

AVII.4.4 Determinación de los parámetros del "embedding"

Según el teorema de Takens-Mañé, se deben determinar los parámetros d_E y τ de forma que $d_E \geq 2D+1$ aunque posteriormente se demostró que alcanza con que $d_E \geq D+1$ y el τ puede ser arbitrario siempre que los datos no tengan ruido y sean de largo infinito (que se tengan infinitas mediciones). Dado que estas suposiciones no son realistas, se deben determinar d_E y τ utilizando métodos basados en los datos. Lo primero que se determina es el "delay óptimo" ya que es usado para calcular luego la dimensión del "embedding".

Caso determinístico

" Delay" óptimo

Se define el "delay" óptimo τ del "embedding" como el "delay" necesario para obtener elementos $\mathbf{x}(n)$ lo suficientemente independientes como para que puedan formar una base para el subespacio de la señal [5, Cap 14]. Si el "delay" es demasiado pequeño, entonces los elementos del vector $\mathbf{x}(n)$ son demasiado similares (están demasiado correlacionados) y son más proclives a los efectos del ruido (se dice que hay un problema de redundancia), por otra parte, si es demasiado grande, entonces los elementos son demasiado disímiles y los vectores **s** se distribuyen en el espacio de estados en una forma dispersa (hay un problema de irrelevancia). Ambas situaciones dificultan la reconstrucción del atractor. En el método mAMI ("minimum average mutual information") la información mutua promedio $I(n)$ entre la serie x_n y su versión demorada x_{n+t} se calcula para $t = 1, 2 \ldots$ como

$$I(t) = \sum_{n=1,2\ldots} P(x_n, x_{n+t}) \log_2 \left[\frac{P(x_n, x_{n+t})}{P(x_n) P(x_{n+t})} \right]$$

donde $P(\bullet)$ es la probabilidad del argumento y $P(\bullet,\bullet)$ denota distribución conjunta. El valor de t donde se da el primer mínimo de $I(\bullet)$ es sugerido como una buena estimación del "delay" óptimo τ del "embedding". Adicionalmente, el AMI pone de manifiesto correlaciones no lineales que pueden no verse claramente si se estudian solo las correlaciones lineales.

Otra opción podría ser elegir el τ como el ***tiempo de correlación de la serie***: el τ tal que la autocorrelación $C(\bullet)$ de la serie cae a la mitad de la que tenía inicialmente [56]:

$$\begin{cases} C(\tau) = \dfrac{1}{2} C(0) \\ C(k) = \displaystyle\sum_{m=-\infty}^{+\infty} x_m x_{m+k} \end{cases}$$

Dimensión global y dimensión local del "embedding"

La dimensión del subespacio de la señal ("emdbedding") no tiene porqué ser la misma que la del atractor, D, y alcanza con que sea $d_E \geq D+1$. A dicha dimensión la llamamos ***dimensión global*** d_E del "embedding". El procedimiento seguido para determinarla se basa en consideraciones geométricas y fue establecido por Kennel y otros (citados por [32]). El método encuentra el ***vecino más próximo*** ("nearest neighbor") de cada punto en una dimensión dada, y luego verifica que esos puntos todavía sean vecinos próximos en una dimensión mayor. Este método tiene como gran

ventaja la de no requerir muestras muy grandes de datos. En esencia, lo que se hace es determinar cuando los puntos en la dimensión *d* son vecinos de otro por virtud de la proyección del atractor en una dimensión demasiado baja:

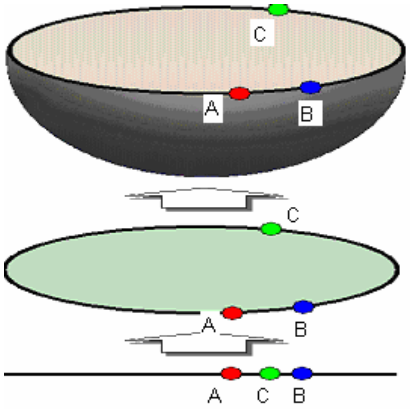


Figura AVII.4

Obsérvese como el punto A y el B aparecen como los vecinos más próximos al C cuando se los proyecta sobre una línea en un espacio unidimensional, pero aparecen como mucho más apartados cuando se los proyecta sobre un disco en un espacio de dos dimensiones. Por contraste, los puntos A y B son cercanos en el espacio de dos dimensiones y continúan siéndolo en el de tres dimensiones. Examinando la situación en la dimensión uno, dos, tres, etc. hasta que no haya vecinos incorrectos (o falsos), se podría establecer, solamente a partir de consideraciones geométricas, un valor para la dimensión del embedding. Si graficamos el porcentaje de vecinos más próximos falsos ("false nearest neighbors") en función de la dimensión del "embedding", la curva debería caer hasta cerca de 0 cuando la dimensión adecuada del "embedding" es alcanzada.

El algoritmo puede esquematizarse como sigue (por su justificación, ver [32]):

1. *Para cada vector* $\mathbf{x}(i) = [x_i, x_{i-1}, x_{i-2} \ldots x_{i-n}]$ *en la serie temporal buscar su vecino más próximo* $\mathbf{y}(j) = [y_j, y_{j-1}, y_{j-2} \cdots y_{j-n}]$ *en un espacio n-dimensional.*
2. *Calcular la distancia* $\|\mathbf{x}(i) - \mathbf{y}(j)\|$
3. *Tomar los siguientes valores de la serie temporal,* x_{i+1} *e* y_{j+1} *y calcular*

$$R_i = \frac{|x_{i+1} - y_{j+1}|}{\|\mathbf{x}(i) - \mathbf{y}(j)\|}$$

4. *Si* R_i *excede un umbral preestablecido heurísticamente* R_τ*, este punto es marcado como que tiene un vecino más próximo falso.*

5. *Si el porcentaje de puntos para los cuales es* $R_i > R_T$ *es menor que un valor preestablecido, ese n es la dimensión del embedding, sino, aumentar n en 1 y repetir los pasos anteriores*

Si bien este método es bastante intuitivo y fácil de implementar, no es utilizable directamente, ya que, entre otros aspectos, requiere la determinación heurística de dos valores de umbral (el R_τ que determina cuando un punto es o no un vecino más próximo falso y el porcentaje R_T de vecinos más próximos falsos al que se debe llegar para decir que se encontró la dimensión correcta). Lo que hace la herramienta elegida (VRA) es implementarlo ligeramente modificado, de forma que no tenga parámetros a establecer. Dado que Y_j es el vecino más cercano a X_i, puede pensarse que es una aproximación a él, o que x_{n+1} es predicho por y_{n+1}, por lo que R es el error de predicción. La idea es que cuando el atractor esta completamente desdoblado en n dimensiones, la distancia R entre los componentes (n+1)-ésimos de X e Y va a ser pequeña. Para detectar si el vecino más próximo recién hallado es falso, comparamos R con los errores que se habrían cometido usando un predictor trivial (el que usa x_n como predictor de x_{n+1}). O sea, formalmente, si $|x_{n+1} - y_{n+1}| = |x_{n+1} - x_n|$ el siguiente vecino es rotulado como falso.

Dado que este algoritmo es utilizado para hallar la dimensión global del "embedding", es llamado de ***Global FNN***.

Adicionalmente a la global, algunas herramientas (como el cspX) calculan la ***dimensión local*** del "embedding", d_L tal que $d_L \leq d_E$ la cual representa el número de grados de libertad activos que gobiernan la evolución local del sistema. Por ejemplo, supongamos que se tiene una órbita periódica o ciclo límite como estado estacionario del sistema. Entonces, el movimiento de los puntos del espacio de estados es sobre un objeto unidimensional, pero se precisa un $d_E = 2$ o 3 para poder desarrollar el atractor y ver esto, aunque la dinámica al moverse sobre el ciclo límite es unidimensional. Por ello el cspX utiliza el método de Local FNN que lo que hace es añadir alguna información sobre la geometría del atractor a la usada el método de Global FNN.

Caso estocástico

En el caso estocástico, al calcular la dimensión el método de los FNN ya no sería válido ya que se basa en la continuidad del caso determinístico:

(si **x**(*j*) es el vecino más próximo a **y**(*j*) entonces

x_{j+1} debe ser cercano a y_{j+1} para una dimensión d_E adecuada)

y habría que hacer alguna consideración basándose en la teoría de probabilidades. Por ello, Fuera y Yanagawa ([25]), para el caso en que la serie temporal cumpla

$$x_t = F(x_{t-\tau}, x_{t-2\tau}, \ldots x_{t-d\tau}) + \varepsilon_t \quad t = 1, 2 \ldots$$

siendo

F una función vectorial no lineal de argumento vectorial

ε_t un ruido dinámico
τ el "delay" óptimo
d la dimensión del subespacio de la señal

y que cumple:

$$\begin{cases} F(x_{t-\tau}, x_{t-2\tau}, \ldots x_{t-d\tau}) = \left\langle x_t \middle| x_{t-\tau}, x_{t-2\tau}, \ldots x_{t-d\tau} \right\rangle_{x_t} \\ \left\langle \varepsilon_t \middle| x_{t-\tau}, x_{t-2\tau}, \ldots x_{t-d\tau} \right\rangle_{\varepsilon_t} = 0 \\ \left\langle x_n^2 \right\rangle < \infty \end{cases}$$

más ciertas condiciones que debe cumplir el ruido dinámico (ver [25]), definen a la dimensión del "embedding" d_0 con un delay óptimo τ_0 si y solo si se cumplen simultáneamente:

- existen d_0 y τ_0 y son finitos, enteros y no negativos
- $P\left[F(x_{t-\tau}, x_{t-2\tau}, \ldots x_{t-d\tau}) \neq F(x_{t-\tau_0}, x_{t-2\tau_0}, \ldots x_{t-d_0\tau_0}) \right] = 1$ para todo $d < d_0$ y $\tau > 0$
- $P\left[F(x_{t-\tau}, x_{t-2\tau}, \ldots x_{t-d\tau}) = F(x_{t-\tau_0}, x_{t-2\tau_0}, \ldots x_{t-d_0\tau_0}) \right] = 1$ para todo par $(d, \tau) \in B(d_0, \tau_0)$ siendo

$$B(d_0, \tau_0) = \{(d, \tau) \,|\, \{\tau_0, 2\tau_0, \ldots d_0\tau_0\} \subset \{\tau, 2\tau, \ldots d\tau\}\}$$

Asimismo propusieron un método de estimación estadística del delay óptimo y de la dimensión para ese caso [25]. Dado que en nuestro caso el estado depende del instante anterior (las ventas de una semana dependen de las ventas de la semana anterior), en el modelo que plantean debería ser $\tau = 1$, lo que contrasta con lo obtenido a partir del estudio del AMI (ver III.2.3) donde se concluye que $\tau > 1$, por lo que concluimos que este modelo no es aplicable a nuestro caso y por lo tanto utilizaremos otro procedimiento para determinar la dimensión del atractor. Se han desarrollado otras formas de estimar la dimensión del atractor[53], pero son referidas a un "delay" = 1.

AVII.5 El análisis de recurrencia y los gráficos de recurrencia

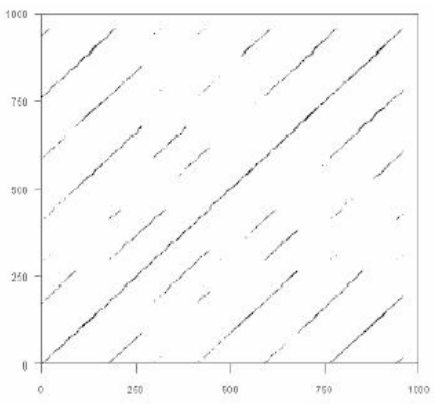
El ***análisis de recurrencia*** ("recurrence analysis") es un método gráfico diseñado para localizar patrones ocultos que se repiten a lo largo del tiempo (en el sentido de comportamientos repetitivos que no son fácilmente detectables), no estacionariedad y cambios en la estructura de una serie temporal. Sea x_n la serie a estudiar y sean los vectores $\mathbf{x}_i^m = [x_i, x_{i+d}, \ldots x_{i+(m-1)d}]$ donde m es la dimensión del "embedding" y d el "delay" óptimo. A partir de los vectores $\mathbf{x}_i^m$ se forma una matriz simétrica con las distancias euclidianas entre todos los pares posibles de ellos y esas distancias son visualizadas mediante un gráfico llamado ***gráfico de recurrencia*** (o "***recurrence plot"***) en el que cada distancia de esa matriz se asocia con un color (por ejemplo, a mayor distancia más "frío" el color; esa es la manera en que procede el VRA: las mayores distancias corresponden a los azules y las menores a los rojos). Por lo tanto, un "recurrence plot" es un gráfico cuadrado sólido consistente de pixeles cuyo color corresponde a la magnitud de los valores de una matriz bidimensional y cuyas coordenadas corresponden a las ubicaciones de los valores de los datos en la matriz. También es bastante usual establecer un radio crítico ε y marcar un punto como oscuro solo si la distancia correspondiente es no mayor que ε (tal es el caso del TISEAN). Se obtiene así un gráfico en blanco y negro. Los vectores comparados consigo mismos dan una distancia nula, lo que explica la presencia de la línea diagonal que va desde el vértice inferior izquierdo hasta el superior derecho (***línea de identidad***) y que aparece en todos los recurrence plots.

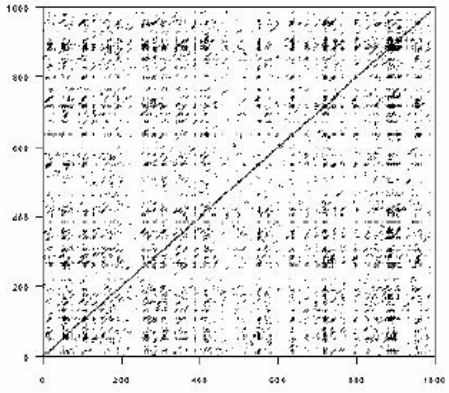
Si la serie analizada es periódica entonces el recurrence plot muestra segmentos de recta paralelos a la línea de identidad que corresponden a secuencias de pares (i,j), $(i+1,j+1)...(i+k,j+k)$ tales que $\mathbf{x}_i^m, \mathbf{x}_{i+1}^m \ldots \mathbf{x}_{i+k}^m$ son cercanos a $\mathbf{x}_j^m, \mathbf{x}_{j+1}^m \ldots \mathbf{x}_{j+k}^m$ para algún j dado i. Esos segmentos indican que se esta repitiendo la visita a una misma región de un atractor en diferentes instantes. Por otra parte, si la serie es un ruido blanco, entonces el "recurrence plot" no muestra estructura alguna sino que esta formado por puntos distribuidos homogéneamente. La no estacionariedad se expresa como una tendencia de todos los puntos del gráfico a estar cerca de la línea de identidad [20].

Este análisis visual nos permite establecer en la Sección III la estacionariedad del proceso. Por otra parte, para que las conclusiones extraídas no queden solo fundamentadas por apreciaciones visuales, se han desarrollado indicadores (por ejemplo, de determinismo, de recurrencia, etc.) que representan numéricamente las características gráficas del RP y a esa técnica se la llama RQA ("Recurrence Quantification Análisis").

El TISEAN produce el RP de la figura AVII.5::

serie caótica (sistema de Rossler): **ruido blanco:**





ı dimensión del atra

Figura AVII.5

Por otra parte, utilizando la herramienta VRA los recurrence plots obtenidos para distintas series se muestra en la figura AVII.6:

función *seno(x)* **ruido blanco** **serie caótica**

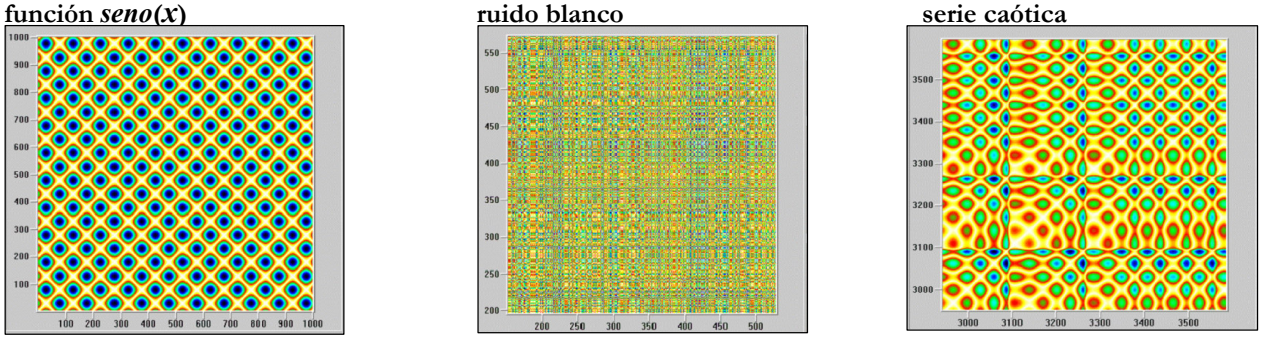


Figura AVII.6

AVII.6 Predicción y reconstrucción dinámica

Los sistemas dinámicos evolucionan con el tiempo de acuerdo a un conjunto de reglas que generalmente son no lineales. En el caso determinístico, las condiciones actuales determinan el futuro y puede haber varias variables involucradas que interactúan entre sí para determinar el estado del sistema. En el caso estocástico, las condiciones actuales (el estado) determina solo en parte el futuro, ya que el azar interviene en forma de ruido dinámico. Procesos de ese tipo son estudiados en todas las ramas de la ciencia, desde el sistema solar a la fluctuación de los valores de las acciones en la bolsa. Por otra parte, una serie temporal consiste en un conjunto de observaciones (medidas) tomadas de un cierto sistema. En algunos casos, el sistema puede ser descrito por un conjunto de ecuaciones, las que forman el mecanismo subyacente que genera esas observaciones. En muchos sistemas reales, esas ecuaciones son desconocidas, y el único conocimiento disponible es una secuencia temporal de medidas, es más, aunque esos sistemas tengan varias variables que gobiernan su comportamiento, a menudo solo se dispone de las medidas de una sola variable en la forma de una serie de datos. El estudio de esa serie de datos y del sistema asociado puede hacerse a nivel de predicción o de reconstrucción dinámica.

La ***predicción*** de series temporales, como forma de predecir el comportamiento futuro del sistema, ha sido una de las áreas claves de estudio de la ciencia y la ingeniería. Dadas algunas observaciones del comportamiento del sistema en el pasado, se desea hacer ciertas predicciones sobre su comportamiento futuro y conocer qué tan exactas son; en nuestro caso, deseamos predecir de las ventas semanales de gas propano en garrafas a partir de las ventas semanales históricas.

El otro enfoque es el de la ***reconstrucción dinámica***. La reconstrucción dinámica y la predicción de series temporales son dos áreas que tienen muchos aspectos en común pero difieren en la forma en la que los resultados finales son evaluados. El objetivo de la predicción de series temporales desde el punto de vista clásico es obtener un valor que sea tan próximo como sea posible a los datos reales. Por otra parte, el fin de la reconstrucción dinámica es capturar la dinámica de la serie temporal por medio de un modelo. O sea, la predicción se relaciona con la evolución a corto plazo de los datos mientras que la reconstrucción dinámica se ocupa del comportamiento dinámico a largo plazo [46]. Esto no significa que no se puedan realizar predicciones a largo plazo, es más, la reconstrucción dinámica busca que esas predicciones a largo plazo sean las mejores posibles. Para verificar si el modelo ha captado la dinámica del sistema se utilizan las medidas invariantes (ver AVII.3).

La predicción puede pensarse asociada a un modelado que puede ser tanto global como local. Al modelar localmente, el atractor es particionado en regiones más pequeñas y se construyen varios modelos basados en cada una de ellas. Los modelos locales, por lo tanto, varían de región a región y debido a las discontinuidades que existen en las zonas del atractor donde los modelos se superponen (unen), se producen comportamientos a largo plazo indeseados, por lo que los modelos locales resultan pobres para la reconstrucción dinámica [52]. En cambio, al modelar globalmente se considera la dinámica de cada región del atractor en forma conjunta. Un modelado global se podría realizar por ejemplo utilizando un MLP que utilizara todos los datos disponibles, por otra parte, los modelos locales se podrían

formar por ejemplo a partir de modelos AR-MA locales, modelos locales basados en SOM (redes de Kohonen) o incluso polinomios que aproximen la serie localmente.

La reconstrucción dinámica nos permite obtener un modelo del sistema dinámico, lo que a su vez hará posible realizar algunas pruebas (experimentos) sobre el comportamiento del sistema en ciertas situaciones. Obsérvese que tanto para la predicción como para la reconstrucción, al utilizar redes neuronales, no se ha hablado de una red sola, sino que pueden utilizarse varias a la vez: por ejemplo, se podría utilizar una red para la predicción de la serie y otra para la predicción del error (en el caso determinístico). Ese enfoque es aplicado en [28].

Los datos imponen limitaciones al modelo desde el punto de partida, ya que un modelo será bueno solo en aquellas regiones del atractor para las cuales existen suficientes datos disponibles. Por esto, la calidad de un modelo, especialmente de uno global, esta limitada por la mejor calidad obtenible en él para las áreas del atractor menos representadas en el conjunto de entrenamiento.

ANEXO VIII: Los MLPs como aproximadores universales

Comentaremos en esta sección dos teoremas que fundamentan el uso de los MLPs como aproximadores universales de cualquier función continua de argumento en $\mathbf{R}^n$. Del teorema de Kolmogorov básicamente se desprende que es factible aproximar una función mediante varios perceptrones trabajando en paralelo, y del de Cybenko que una red feedforward con una sola capa oculta y una neurona de salida lineal es capaz de aproximar cualquier función $f : \mathbf{R}^n \to \mathbf{R}$ con un grado de precisión arbitrario. La bibliografía de este anexo es básicamente [4].

AVIII.1 El teorema de Kolmogorov

Este teorema clásico puede enunciarse como:

> *Sea una función* $\boldsymbol{f}$ *continua arbitraria de argumento real* $f(x_1, x_2 \ldots x_n)$ *con* $\mathrm{x}=[x_1 \ldots x_n] \in [0,1]^n \quad n \geq 2$ *. La función* $\boldsymbol{f}$ *puede ser representada como* $f(x_1 \ldots x_n) = \sum_{j=1}^{2n+1} g_j \left[\sum_{i=1}^{n} \varphi_{ij}(x_i) \right]$ *siendo las* g_j *funciones de una variable y* φ_{ij} *funciones continuas monótonamente crecientes, no diferenciables e independientes de* $\boldsymbol{f}$ [4].

El teorema indica que se puede expresar una función continua multivaluada en términos de sumas de un número finito de funciones de una variable. La aplicación directa de este teorema a la justificación de que las redes feedforward sean aproximadores universales ha sido muy discutida básicamente por dos razones:

a) las funciones φ_{ij} no deben ser "suaves" (diferenciables). El uso de funciones no suaves en una red conduciría a problemas de sensibilidad extrema con respecto a los valores de entrada [1]

b) las funciones g dependen de la función f a representar, lo cual es la situación inversa que se suele encontrar al trabajar con redes neuronales: en general se consideran funciones de salida fijas y luego se ajusta el número de neuronas ocultas y los pesos y umbrales de activación hasta obtener una aproximación lo suficientemente buena. Para aclarar la idea, consideremos lo que expresa el teorema de Kolmogorov gráficamente, planteando una red que implemente la descomposición propuesta:

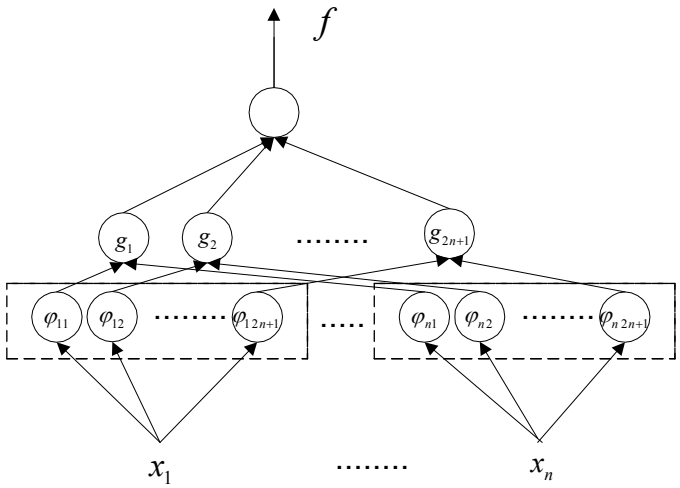


Figura AVIII.1

En el teorema de Kolmogorov el número de neuronas ocultas es fijo y las funciones de salida dependen de la función que se este aproximando. De todos modos, a pesar de su poca aplicabilidad práctica, el teorema señala la factibilidad de utilizar redes "feedforward" multicapa trabajando en paralelo (los recuadros punteados estarían trabajando en paralelo) para aproximar cualquier función.

AVIII.2 El teorema de Cybenko

La prueba matemática de que los MLPs que usan funciones de salida sigmoides continuas son aproximadores universales fue dada independientemente por Cybenko, Hornik y Funahashi (citados por [4]) aunque se le suele llamar teorema de Cybenko por haber dado éste la demostración más elegante y concisa.

El teorema se puede enunciar como:

Sea $\varphi(\bullet)$ *una función continua de tipo sigmoide. Dada cualquier función real* $f:[0,1]^n \to \mathbf{R}$ y un $\varepsilon > 0$, *existen vectores* $\mathbf{w}_1, \mathbf{w}_2 \ldots \mathbf{w}_N$, $\boldsymbol{\alpha}$ y $\boldsymbol{\theta}$ *y una función parametrizada* $G(\bullet, \mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\theta}): [0,1]^n \to \mathbf{R}$ *tal que*

$$\begin{cases} \left|G(\mathbf{x},\mathbf{w},\boldsymbol{\alpha},\boldsymbol{\theta}) - f(\mathbf{x})\right| < \varepsilon \quad \forall \mathbf{x} \in [0,1]^n \\ G(\mathbf{x},\mathbf{w},\boldsymbol{\alpha},\boldsymbol{\theta}) = \sum_{i=1}^{N} \alpha_j \varphi(\mathbf{w}_j^T \mathbf{x} + \theta_j) \end{cases}$$

siendo $\mathbf{w}_j \in \mathbf{R}^n,\ \alpha_j, \theta_j \in \mathbf{R}, \quad \mathbf{w} = [w_1, \ldots w_N] \quad \boldsymbol{\alpha} = [\alpha_1, \ldots \alpha_n] \ \boldsymbol{\theta} = [\theta_1, \ldots \theta_n]$

El teorema plantea que una red MLP con una capa oculta es capaz de aproximar cualquier función multivariada continua con un grado de exactitud arbitrario. Adicionalmente, Hornik et al. (citados por [4, Cap. 2]) demostraron que la red puede aproximar además las derivadas de *f*, aunque la función *f* solo sea diferenciable de a tramos.

AVIII.3 Generalización

Hornik (citado por [4, Cap. 2]) demostró que una condición suficiente de aproximador universal es que las funciones de salida de la capa oculta sean continuas, acotadas y no constantes. Se puede incluso utilizar funciones no sigmoides, tales como los polinomios de Bernstein (en forma de campana) [4, Cap. 2].

Las redes MLP de una sola capa oculta también son clasificadores universales: alcanza con definir una función f sobre los patrones de entrada $f(\mathbf{x}) = j \Leftrightarrow \mathbf{x} \in P_j$ con $f: A^n \to \{1,2 \ldots k\} \quad A^n \subset \mathbf{R}^n$ cerrado y acotado y $P_1 \ldots P_k$ una partición de A^n

APENDICE: Resultados numéricos obtenidos

En este Apéndice se detallan los resultados numéricos obtenidos al simular las distintas arquitecturas estudiadas. Sobre los criterios de detención, algoritmos de entrenamiento y otros detalles de las mismas, ver III.3.2 a III.3.4.

Sección I: Resultados asociados a las distintas redes

I.1 Los perceptrones

Entrenamos con los datos disponibles hasta la primer semana del 2003 y tratamos de predecir las siguientes.

Obtuvimos:

Para "delay" = 3 **CUADRO AP.1**

		23 ocultas 100 generaciones		8 ocultas 1500 generaciones		11 ocultas 2500 generaciones	
Semana	***Deseado***	***Predicción***	***%Error***	***Predicción***	***%Error***	***Predicción***	***%Error***
semana 2/2003	2.344.265	3.918.987	67,17%	3729404	59,09%	2528898	7,88%
semana 3/2003	2.554.344	3.921.908	53,54%	3731272	46,08%	2528903	1,00%
semana 4/2003	2.568.176	4.009.106	56,11%	3783818	47,33%	2529378	1,51%
semana 5/2003	2.384.806	4.057.595	70,14%	3818994	60,14%	2529458	6,07%
		Min:		***Min:***		***Min:***	
		53,54%		***46,08%***		***1,00%***	
		Max:		***Max:***		***Max:***	
		70,14%		***60,14%***		***7,88%***	
		%Error CV	***44.27%***	***%Error CV***	***31.47%***	***%Error CV***	***14.72%***
		MDL CV	***139.18***	***MDL CV***	***-3.25***	***MDL CV***	***85.70***

Comparando los tres valores de MDL obtenidos verificamos experimentalmente cómo el MDL expresa la complejidad del modelo y a la vez su capacidad de generalización.

Y para "delay" = 2 **CUADRO AP.2**

		1 oculta 1500 generaciones		3 ocultas		7 ocultas	
Semana	***Deseado***	***Predicción***	***%Error***	***Predicción***	***%Error***	***Predicción***	***%Error***
semana 2/2003	2.344.265	4162348	77,55%	3537515	50,90%	4603951	96,39%
semana 3/2003	2.554.344	4231696	65,67%	3660479	43,30%	4631781	81,33%
semana 4/2003	2.568.176	4176919	62,64%	3623348	41,09%	4606232	79,36%
semana 5/2003	2.384.806	4109947	72,34%	3549275	48,83%	4592137	92,56%
		Min:		***Min:***		***Min:***	
		62,64%		***41,09%***		***79,36%***	
		Max:		***Max:***		***Max:***	
		77,55%		***50,90%***		***96,39%***	
		% Error CV	***47.51%***	***% Error CV***	***25.54%***	***% Error CV***	***34.15%***
		MDL CV	***-39.41***	***MDL CV***	***-52.94***	***MDL CV***	***-7.62***

****

El modelo marcado con ********** es el que produjo mejores resultados.
Los valores predichos por la red correspondiente a d=3 y 20 neuronas ocultas para las semanas 6/2003 a la 18/2003

CUADRO AP.3

Semana	*Deseado*	*Predicción*	*%Error*
6	2355510	2477747	4.93%
7	4240721	3250724	30.45%
8	1716938	2238874.75	23.31%
9	1821053.875	2536343.75	28.20%
10	2572587	2686690.75	4.25%
11	2749742	2607091.5	5.47%
12	2518900	2311633	8.97%
13	2072692.875	2288366	9.42%
14	2939268	2827329.25	3.96%
15	2868848	2584882.75	10.99%
16	2455639	2432513.25	0.95%
17	2484590	2399320.75	3.55%
18	3822776	2904267.5	31.63%
%Error:			
Max:	Min:	Promedio:	
31.63%	0.95%	12.78%	

I.1.1 Perceptrones con entradas adicionales

El MLP de entradas T, A-1, V-1, V-2, S-3 y salida V, con 7 neuronas ocultas predijo:

%Error CV: 11.90%
MDL CV: 7.92

CUADRO AP.4

Semana	*Deseado*	*Predicción*	*%Error*
2	2344265	2490637.5	6.24%
3	2554344	2490677.25	2.49%
4	2568176	2689453.5	4.72%
5	2384806	2708356.25	13.57%
6	2355510	2694399	14.39%
7	4240721	2705313.25	36.21%
8	1716938	2693956.25	56.90%
9	1821053.875	2730151.25	49.92%
10	2572587	2797738.75	8.75%
11	2749742	2783593.5	1.23%
12	2518900	2898533.5	15.07%
13	2072692.875	3754147.25	81.12%
14	2939268	3729677.25	26.89%
15	2868848	3094771.5	7.88%
16	2455639	3658980.75	49.00%
17	2484590	4234557	70.43%
18	3822776	3980261.75	4.12%
%Error:			
Promedio:	Mínimo	Máximo	
26.41%	1.23%	81.12%	

I.1.2 Perceptrones con ruido en sus entradas

A los efectos de mejorar los errores obtenidos, intentamos entrenar los MLP añadiendo un ruido de distribución normal de media 0 y distintas varianzas a sus entradas $\{v_i, v_{i-2}\}$ y $\{v_i, v_{i-3}\}$ para d=2 y d=3 respectivamente. Trabajamos con un MLP de 7 neuronas ocultas en una capa.

Se obtuvieron los siguientes resultados:

CUADRO AP.5

	***d* = 3**		***d* = 2**	
Varianza	***%Error CV***	***MDL CV***	***%Error CV***	***MDL CV***
0.1	26.54%	17.71	32.06%	-11.16
0.7	30.68%	19.52	31.30%	-12.29
10	24.67%	30.31	***14.45%***	***-33.35***
25	30.71%	30.27	23.80%	-15.19
50	40.65%	32.46	31.72%	-7.01

Vemos que en el caso d=2, *varianza* = 10, se obtiene una mejora importante del error del 34.15% al 14.45%, lo que de todos modos no significa que el error mejorado sea aceptable.

Las predicciones para esta red fueron:

CUADRO AP.6

Semana	***Deseado***	***Predicción***	***%Error***
2	2344265	2772042.25	18.25%
3	2554344	2784929.25	9.03%
4	2568176	2948155.75	14.80%
5	2384806	2687669.25	12.70%
6	2355510	2678346.75	13.71%
7	4240721	3482456.5	17.88%
8	1716938	2818310.5	64.15%
9	1821053.88	2619462	43.84%
10	2572587	2768996.25	7.63%
11	2749742	3015281.75	9.66%
12	2518900	2743920.75	8.93%
13	2072692.88	2731737	31.80%
14	2939268	2797307.5	4.83%
15	2868848	3082142.5	7.43%
16	2455639	2818102.25	14.76%
17	2484590	2860754	15.14%
18	3822776	3113414	18.56%
			Max: 64.13%
			Min: 4.83%
			Promedio: 18.42%

I.2 Las "time lagged feedworward networks" (TLFNs)

I.2.1 Las TDNNs

Los resultados obtenidos fueron:

CUADRO AP.7

		***d*=3**				***d*=2**			
		TDNN pura focused 3 neuronas ocultas		***TDNN pura no focused 12 neuronas ocultas***		***TDNN pura focused 29 neuronas ocultas***		***TDNN pura no focused 6 neuronas ocultas***	
Semana	***Deseado***	***Predicción***	***%Error***	***Predicción***	***%Error***	***Predicción***	***%Error***	***Predicción***	***%Error***
2	2344265	2899728.25	19.16%	2709111	13.47%	2548934	8.73%	2823800.75	20.46%
3	2554344	2895117.25	11.77%	2802681.5	8.86%	2614089	2.34%	2901880.5	13.61%
4	2568176	2863337.5	10.31%	2809267	8.58%	2620536.25	2.04%	2907194.25	13.20%
5	2384806	2829208.75	15.71%	2726262	12.52%	2557628	7.25%	2838486.25	19.02%
6	2355510	2842025.25	17.12%	2713826.75	13.20%	2551201	8.31%	2827856	20.05%
7	4240721	.	.	4117191.75	3.00%	3955789.5	6.72%	3697800.75	12.80%
8	1716938	.	.	2489850	31.04%	2507750.75	46.06%	2619184.25	52.55%
9	1821053.88	.	.	2520970	27.76%	2508937.5	37.77%	2650281.25	45.54%
10	2572587	.	.	2811378.75	8.49%	2622663.25	1.95%	2908893.25	13.07%
11	2749742	.	.	2900987.25	5.21%	2741738.75	0.29%	.	.
12	2518900	.	.	2786055	9.59%	2599021	3.18%	.	.
13	2072692.88	.	.	2604294.5	20.41%	2517445	21.46%	.	.
14	2939268	.	.	3007983.75	2.28%	2958120.5	0.64%	.	.
15	2868848	.	.	2966812.75	3.30%	2866368.25	0.09%	.	.
16	2455639	.	.	2757246.5	10.94%	2576746.75	4.93%	.	.
17	2484590	.	.	2770294.5	10.31%	2586260.25	4.09%		.
18	3822776	.	.	3687567.75	3.67%	3902742.75	2.09%	.	.
			Max: - Min: - Prom: -		***Max: 31.04% Min: 2.28% Prom: 11.33%***		***Max: 46.06% Min: 0.09% Prom: 9.29%***		***Max: - Min: - Prom: -***
		MDL CV	***-43.60***	***MDL CV***	***-66.30***	***MDL CV***	***85.59***	***MDL CV***	***-1.60***
		%Error CV	***22.36%***	***%Error CV***	***11.88%***	***%Error CV***	***15.52%***	***%Error CV***	***15.62%***

****

Hemos señalado con ********** al modelo que produjo el mejor MDL.

Las TDNNs con entradas adicionales: caso "no focused"

Caso d=3

Utilizamos una red TDNN (elementos de memoria formados en base a "delays"), recordando 2 muestras hacia atrás (los parámetros del simulador fueron "taps" = 2, "delay" =3) que incorporara la variable explicativa ***aumento*** y las ventas y temperaturas de las semanas anteriores. Las entradas posibles S, A, T, S-1, A-1, V-1, T-1, S-2, A-2, V-2, T-2

así como el número de neuronas de la capa oculta se determinaron genéticamente, obteniéndose luego de 4000 generaciones eran

S, T, S-1, S-2, T-2

y tendría 12 neuronas ocultas siendo la salida V.

Caso d=2

Por un proceso genético similar al anterior se obtuvo una red con 6 neuronas ocultas, entradas S, S-1, A-1, V-1, S-2, T-2 y salida V.

Al entrenar estas redes con datos hasta la semana 1/2003 los valores predichos para $d = 2$ y $d = 3$ fueron:

CUADRO AP.8

		***d*=3**		***d*=2**	
Semana	***Deseado***	***Predicción***	***%Error***	***Predicción***	***%Error***
2	2344265	2725854	14,00%	3341127	42.52%
3	2554344	2143003	19,19%	3058119	19.72%
4	2568176	2384944	7,68%	3241484	26.22%
5	2384806	2795620	14,69%	3253538	36.43%
6	2355510	2212919.25	6.44%	2586422.75	9.80%
7	4240721	3312516.75	28.02%	3314502.75	21.84%
8	1716938	3232728.25	46.89%	2456285.5	43.06%
9	1821054	2849647.75	36.10%	2921442.5	60.43%
10	2572587	3573579.75	28.01%	3376316.5	31.24%
11	2749742	2071973.25	32.71%	3490172	26.93%
12	2518900	1696007.88	48.52%	3821577.25	51.72%
13	2072693	2316095.5	10.51%	3538906.5	70.74%
14	2939268	2600537.75	13.03%	4024598.25	36.93%
15	2868848	2992610.75	4.14%	4513100.5	57.31%
16	2455639	3175352.25	22.67%	4499301	83.22%
17	2484590	3255864.25	23.69%	4323762.5	74.02%
18	3822776	4324198	11.60%	4594992.5	20.20%
			Max:		***Max:***
			48.52%		***83.22%***
			Min:		***Min:***
			4.14%		***9.80%***
			Promedio:		***Promedio:***
			24.02%		***41.90%***
		MDL CV	***298.52***	***MDL CV***	***-210.67***
		%Error CV	***12.00%***	***%Error CV***	***12.32***

TDNNs con entradas adicionales: caso "focused"

Para el caso d=3 las entradas fueron S, A , T-1 salida V y 8 neuronas ocultas y para el caso d=2, cinco neuronas ocultas y entradas T, A-2, T-2 y salida V.

Los resultados de predecir con una red entrenada con datos hasta la semana 1/2003 fueron:

CUADRO AP.9

			***d*=3**		***d*=2**	
	Semana	***Deseado***	***Predicción***	***%Error***	***Predicción***	***%Error***
2		2344265	2412940	2.85%	2670121.25	12.20%
3		2554344	3637782	29.78%	2712556.25	5.83%
4		2568176	3641215.75	29.47%	2586457.75	0.71%
5		2384806	2503000.75	4.72%	2682445.25	11.10%
6		2355510	2422016	2.75%	2685393.25	12.28%
7		4240721	3515371	20.63%	2972097.25	42.68%
8		1716938	2881339	40.41%	2640245.25	34.97%
9		1821053.88	3027552.75	39.85%	2616442	30.40%
10		2572587	3618814	28.91%	2785151.25	7.63%
11		2749742	3616153.25	23.96%	2728307	0.79%
12		2518900	3673113.75	31.42%	2776468.5	9.28%
13		2072692.88	3415678	39.32%	2889818.25	28.28%
14		2939268	3625913.75	18.94%	3023271.25	2.78%
15		2868848	3807132.5	24.65%	2915322.5	1.59%
16		2455639	3667331.5	33.04%	2907498	15.54%
17		2484590	3494947.5	28.91%	3061728.75	18.85%
18		3822776	3658224.25	4.50%	3291744.75	16.13%
			Max:		***Max:***	
			40.41%		***42.68%***	
			Min:		***Min:***	
			2.75%		***0.71%***	
			Promedio:		***Promedio:***	
			23.77%		***14.77%***	
			MDL CV	***-373.29***	***MDL CV***	***24.69***
			%Error CV	***13.03%***	***%Error CV***	***24.69%***

I.2.2 Redes con memorias gamma

CUADRO AP.10

		***d*=3**				***d*=2**			
		Gamma pura focused		***Gamma pura no focused***		***Gamma pura focused***		***Gamma pura no focused***	
		25 neuronas ocultas		***19 neuronas ocultas***		***22 neuronas ocultas***		***21 neuronas ocultas***	
Semana	***Deseado***	***Predicción***	***%Error***	***Predicción***	***%Error***	***Predicción***	***%Error***	***Predicción***	***%Error***
2	2344265	2775104.25	18.38%	2906748.25	23.99%	2453890.75	4.68%	2487860.5	6.13%
3	2554344	2843574.5	11.32%	2979218.75	16.63%	2580935.25	1.04%	2731029.25	6.92%
4	2568176	2848257	10.91%	2984151	16.20%	2590577.5	0.87%	2737373.25	6.59%
5	2384806	2787933.25	16.90%	2920379.5	22.46%	2475679.25	3.81%	2656562.5	11.40%
6	2355510	2778644.5	17.96%	2910512.75	23.56%	2459810	4.43%	2644317	12.26%
7	4240721	3561748	16.01%	3722857.5	12.21%	4154968	2.02%	3693048.5	12.91%
8	1716938	2599171.75	51.38%	2716639.25	58.23%	2242693	30.62%	2422141.25	41.07%
9	1821053.88	.	.	.	.	2264972	24.38%	2452753.5	34.69%
10	2572587	.	.	.	.	2593686.75	0.82%	2739404.5	6.48%
11	2749742	.	.	.	.	2732434.25	0.63%	2824301	2.71%
12	2518900	.	.	.	.	2556970.25	1.51%	2714956.25	7.78%
13	2072692.88	.	.	.	.	2337053.25	12.75%	2535542.25	22.33%
14	2939268	.	.	.	.	2909712.25	1.01%	2921915.75	0.59%
15	2868848	.	.	.	.	2840607	0.98%	2884868.25	0.56%
16	2455639	.	.	.	.	2516807.5	2.49%	2686930	9.42%
17	2484590	.	.	.	.	2534777.5	2.02%	2699650.5	8.66%
18	3822776	.	.	.	.	3849856	0.71%	3439939.75	10.01%
			Max:		***Max:***		***Max:***		***Max:***
			-		***-***		***30.62%***		***41.07%***
			Min:		***Min:***		***Min:***		***Min:***
			-		***-***		***0.63%***		***0.56%***
			Prom:		***Prom:***		***Prom:***		***Prom:***
			-		***-***		***5.57%***		***11.79%***
	%Error CV		***17.65***		***18.80***		***16.82***		***14.19***
	MDL CV		***75.13***		***204.81***		***51.46***		***48.25***

Redes con memorias gamma y entradas adicionales: caso "no focused"

Para el caso d=3, las entradas determinadas fueron T, A-2, T-2 y 12 neuronas ocultas y para d=2, T, S-2, V-2 y 25 neuronas ocultas. Las salidas eran V en los dos casos. Las predicciones fueron en este caso:

CUADRO AP.11

			***d*=3**		***d*=2**	
	Semana	***Deseado***	***Predicción***	***%Error***	***Predicción***	***%Error***
2		2344265	2606598	10.06%	1818022	28.95%
3		2554344	2593900.5	1.52%	3065454	16.67%
4		2568176	2498843	2.77%	2267771.75	13.25%
5		2384806	2613217.75	8.74%	2681316	11.06%
6		2355510	2525607	6.73%	2926280.75	19.50%
7		4240721	2807214	51.07%	3379081.75	25.50%
8		1716938	2514221.75	31.71%	2641476.25	35.00%
9		1821053.88	2549454.75	28.57%	2306044.75	21.03%
10		2572587	2657207.25	3.18%	3146541.5	18.24%
11		2749742	2592116.25	6.08%	2807913.75	2.07%
12		2518900	2746054	8.27%	2787752.25	9.64%
13		2072692.88	2779837.5	25.44%	3534051	41.35%
14		2939268	2925210.25	0.48%	3580263.5	17.90%
15		2868848	2980254.5	3.74%	2989888	4.05%
16		2455639	2968079.75	17.27%	3070085	20.01%
17		2484590	3006479	17.36%	3717033.25	33.16%
18		3822776	3310502.5	15.47%	3620478	5.59%
				Max:		***Max:***
				51.07%		***41.35%***
				Min:		***Min:***
				0.48%		***2.07%***
				Promedio:		***Promedio:***
				14.03%		***19.00%***
			%Error CV	***16.87%***	***%Error CV***	***20.57%***
			MDL CV	***214.88***	***MDL CV***	***555.59***

Redes con memorias gamma y entradas adicionales: caso "focused"

Cuando d=3, las entradas son S, A, V-1, T-1, S-2, V-2, T-2 salida V y 17 neuronas ocultas, mientras que cuando d=2, las entradas son S-1, A-1, V-1 S-2, T-2 y salida V con 25 neuronas ocultas.

Las predicciones realizadas fueron:

CUADRO AP.12

		d = 3		*d* = 2	
Semana	***Deseado***	***Predicción***	***%Error***	***Predicción***	***%Error***
2	2344265	2872066.25	18.38%	2263150.75	3.58%
3	2554344	3503731.75	27.10%	3155498.75	19.05%
4	2568176	3551868.75	27.70%	3222242.75	20.30%
5	2384806	3807154.5	37.36%	2244558.75	6.25%
6	2355510	3510817.5	32.91%	2408558	2.20%
7	4240721	3397964.5	24.80%	2704284.75	56.81%
8	1716938	3724796.75	53.91%	2630635.5	34.73%
9	1821053.88	3597756.25	49.38%	3467937.5	47.49%
10	2572587	3772434	31.81%	3459004.25	25.63%
11	2749742	3734098.75	26.36%	3577835	23.15%
12	2518900	3961649.25	36.42%	3816317.5	34.00%
13	2072692.88	3883296.5	46.63%	3740005.25	44.58%
14	2939268	3756403.25	21.75%	3931191.5	25.23%
15	2868848	3971334	27.76%	3728484	23.06%
16	2455639	4132017.5	40.57%	3558023.5	30.98%
17	2484590	3976627.75	37.52%	3808675.5	34.76%
18	3822776	3941217.75	3.01%	4174012.25	8.41%
			Max:		***Max:***
			53.91%		***56.81%***
			Min:		***Min:***
			3.01%		***2.20%***
			Promedio:		***Promedio:***
			31.96%		***25.90%***
		%Error CV	***11.49%***	***%Error CV***	***12.64%***
		MDL CV	***213.13***	***MDL CV***	***483.93***

I.3 Las redes recurrentes

I.3.1 Redes total/parcialmente recurrentes

Los resultados obtenidos al predecir en un paso para 12 juegos de datos de entrenamiento (ej1 ...ej12) (Ver III.3.2 "Redes recurrentes" y III.3.3) se presentan a continuación:

CUADRO AP.13

		Totalmente Recurrente sin ventas-2		Totalmente Recurrente con ventas-2		Parcialmente Recurrente sin ventas-2		Parcialmente Recurrente con solo ventas-2		Parcialmente Recurrente con ventas-2		Totalmente recurrente solo con ventas-2	
	Valor Deseado	Predicción	% error	Predicción	% error	Predicción	% error	Predicción	% error	Predicción	% error	Predicción	% error
Caso													
ej1	3179749	3245813.5	2.08	3359301.5	1.72	3359301.5	5.65	3377355.75	6.21	3617101	13.75	3318300	4.36
ej2	3027566	3228567	6.64	3053562.25	2.39	3053562.25	0.86	3153633	4.16	3632618	19.98	3102986.25	2.49
ej3	3713998	3339395.5	10.09	3643268.75	6.61	3643268.75	1.90	3791146.75	2.08	3552330	4.35	3602935	2.99
ej4	5630243	5251174	6.73	5368739	2.86	5368739	4.64	5598328	0.57	5022313	10.80	5181746.5	7.97
ej5	5647959	5652651.5	0.08	6158455.5	0.45	6158455.5	9.04	6444480.5	14.10	5671084	0.41	5148381.5	8.85
ej6	6133903	6040241	1.53	5615924	3.70	5615924	8.44	5551379	9.50	5861538	4.44	5792694.5	5.56
ej7	4086884	4769349	16.70	4985252	2.10	4985252	21.98	4631253.5	13.32	5671325	38.77	4488325.5	9.82
ej8	3844403	3378308.5	12.12	4269624	11.21	4269624	11.06	3948619	2.71	3142223	18.26	3987942.5	3.73
ej9	4561935	3966971.75	13.04	4231994.5	0.32	4231994.5	7.23	4386108.5	3.85	4545565	0.36	4237661	7.11
ej10	3336950	3564627.75	6.82	3894742.5	4.26	3894742.5	16.72	3845397.75	15.24	3747938	12.32	3748329.5	12.33
ej11	4093933	4254818	3.93	4499364	1.83	4499364	9.90	4431014.5	8.23	4122404	0.70	4045892	1.17
ej12	4059187	4176860	2.90	4214487.5	4.00	4214487.5	3.83	4425522	9.02	4540177	11.85	4484246.5	10.47
			Error min		***Error min***		***Error min***		***Error min***		***Error min***		***Error min***
			0.08		***0.32***		***0.86***		***0.57***		***0.36***		***1.17***
			Error Max		***Error Max***		***Error Max***		***Error Max***		***Error Max***		***Error Max***
			16.70		***11.21***		***21.98***		***15.24***		***38.77***		***10.47***
			Error Prom.		***Error Prom.***		***Error Prom.***		***Error Prom.***		***Error Prom.***		***Error Prom.***
			6.88		***3.45***		***8.43***		***6.71***		***10.22***		***6.40***
		Entrenando con datos de 18/1999-27/2002:											
		MDL CV	***62.20***	***MDL CV***	***75.17***	***MDL CV***	***76.20***	***MDL CV***	***76.28***	***MDL CV***	***85.33***	***MDL CV***	***69.69***
		Error CV	***11.49%***	***Error CV***	***11.86%***	***Error CV***	***16.97***	***Error CV***	***21.38***	***Error CV***	***18.42%***	***Error CV***	***16.33%***

****

La red que mejor se comportaba (considerando ya sea el MDL o el % de error) era la totalmente recurrente que no incluía las ventas de la antepenúltima semana (V-2), señalada como **********. Sin embargo, dado que predecir con esa red ("modelo viejo") requería conocer una estimación de la temperatura máxima media semanal de la semana a predecir, se probó usar un modelo alternativo ("modelo nuevo"), con entradas S, V-1, T-1 y salida V, 7 neuronas ocultas en una sola capa. Ese modelo arrojó el siguiente comportamiento al predecir a partir de la semana 28 del año 2002:

CUADRO AP.14

MODELO NUEVO		**MODELO VIEJO**	
MDL CV	***51.60***	***MDL CV***	***62.20***
Error CV	***15.59%***	***Error CV***	***11.49%***

O sea que se obtuvo una mejora en los resultados numéricos (en cuanto al MDL) y en la "usabilidad" de la red (no se precisa conocer la estimación de temperatura de los siete día próximos, ya que con ese modelo se predicen las ventas de a semana a partir de las temperaturas de la misma) pero se pierde en la exactitud de la estimación.

Utilizando el modelo nuevo obtuvimos (entrenando con datos hasta semana 1/2003):

CUADRO AP.15

Semana	*Deseado*	*Predicción*	*% Error*
2	2344265	2305934	1.66%
3	2554344	2335691	9.36%
4	2568176	2303485	11.49%
5	2384806	2284858	4.37%
6	*2355510*	2176097	7,62%
7	*4240721*	2320609	45,28%
8	*1716938*	2127601	23,92%
9	*1821054*	1908363	4,79%
10	*2572587*	1885198	26,72%
11	*2749742*	1889300	31,29%
12	*2518900*	1885948	25,13%
13	*2072693*	1888062	8,91%
14	*2939268*	1888733	35,74%
15	*2868848*	1877860	34,54%
16	*2455639*	1889078	23,07%
17	*2484590*	1885356	24,12%
18	*3822776*	1886340	50,66%
Errores:			
	Min:	***Max:***	***Prom:***
	1.66%	***50.66%***	***20.22%***

I.3.2 Redes de Jordan y Elman

Los valores obtenidos para los distintos valores de la constante tiempo fueron:

CUADRO AP.16

		Tiempo=0.5		*Tiempo=0.3*		*Tiempo=0.7*		*Tiempo=0.15*		*Tiempo=0.07*	
S	***Deseado***	**Predicción**	**% Error**	**Predicción**	**% Error**	**Predicción**	**% Error**	**Valor predicho**	***% Error***	**Predicción**	**% Error**
2	***2344265***	2414434	2.99%	2442117.5	4.17%	2463019	5.07%	2369037.75	*1.06%*	2394427.5	2.14%
3	***2554344***	2434795.75	4.68%	2448866.75	4.13%	2475831.25	3.07%	2410865.5	*5.62%*	2399399	6.07%
4	***2568176***	2455168	4.40%	2456400.5	4.35%	2505332.75	2.45%	2441289.5	*4.94%*	2407551.5	6.25%
5	***2384806***	2416266	1.32%	2442252	2.41%	2463920.75	3.32%	2426736.75	*1.76%*	2418028.25	1.39%
6	***2355510***	2272649.75	3.52%	2397150	1.77%	2258008	4.14%	*2376950.5*	*0.91%*	2337167.5	0.78%
7	***4240721***	2154249.75	49.20%	2314520	45.42%	2179094.75	48.61%	*2370348.5*	*44.11%*	2355156	44.46%
8	***1716938***	2143677.75	24.85%	2307053	34.37%	2209532	28.69%	*2370348.5*	*38.06%*	2355156	37.17%
9	***1821054***	2133047	17.13%	2332147.5	28.07%	2170073.75	19.17%	*2370348.5*	*30.16%*	2355156	29.33%
10	***2572587***	2121040.25	17.55%	2339341.5	9.07%	2173539	15.51%	*2369738.25*	*7.89%*	2361895.5	8.19%
11	***2749742***	2093311.75	23.87%	2334373.25	15.11%	2124619	22.73%	*2346486.75*	*14.67%*	2350962.5	14.50%
12	***2518900***	2128832.5	15.49%	2346548.75	6.84%	2278125.5	9.56%	*2436547*	*3.27%*	2378742.5	5.56%
13	***2072693***	2160436.25	4.23%	2369910	14.34%	2374546.25	14.56%	*2505165*	*20.87%*	2415120.75	16.52%
14	***2939268***	2104769	28.39%	2391671.25	18.63%	2277299.75	22.52%	*2416106*	*17.80%*	2422233.75	17.59%
15	***2868848***	2228633.5	22.32%	2395532.25	16.50%	2530665.5	11.79%	*2642621.25*	*7.89%*	2462884	14.15%
16	***2455639***	2140395.75	12.84%	2403819	2.11%	2406818.75	1.99%	*2505525.75*	*2.03%*	2451038	0.19%
17	***2484590***	2176536.5	12.40%	2459902.75	0.99%	2500862.5	0.65%	*2591815.5*	*4.32%*	2529483.75	1.81%
18	***3822776***	2366586	38.09%	2526185	33.92%	2766177.5	27.64%	*2982325.75*	*21.99%*	2685770	29.74%
			- Errores - Promedio: 16.66% ***Máximo:*** 49.20% ***Mínimo:*** 1.32%		***- Errores - Promedio:*** 14.25% ***Máximo:*** 45.42% ***Mínimo:*** 0.99%		***- Errores - Promedio:*** 14.20% ***Máximo:*** 48.61% ***Mínimo:*** 0.65%		***- Errores - Promedio:*** 13.37% ***Máximo:*** 44.11% ***Mínimo:*** 0.91%		***- Errores - Promedio:*** 13.87% ***Máximo:*** 44.46% ***Mínimo:*** 0.19%
		MDL CV -6.71 %Error CV 18.12		***MDL CV -9.35 %Error CV 17.8***		***MDL CV -4.99 %Error CV 20.36***		***MDL CV -15.63 %Error CV 16.35***		***MDL CV -4.62 %Error CV 19.45***	

****

El gráfico siguiente muestra la variación del error al cambiar la constante tiempo:

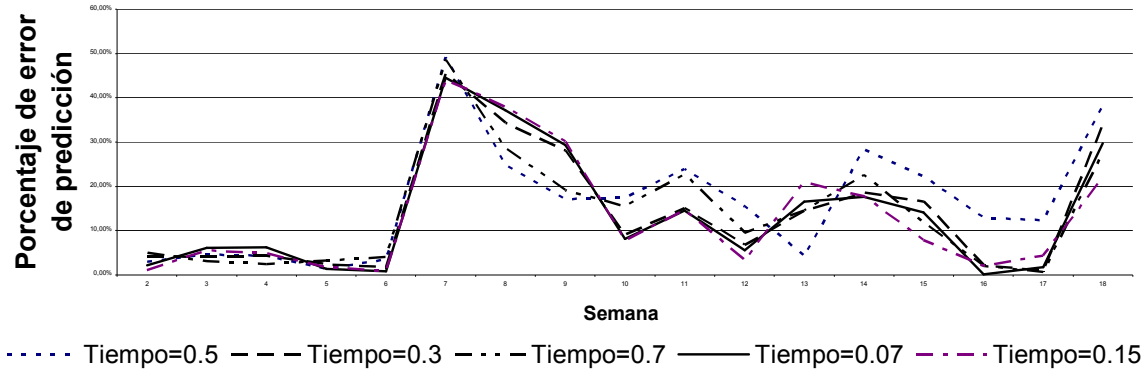


Figura AP.1

El número de neuronas ocultas en el caso de las redes de Jordan no puede ser determinado mediante AGs.

Adicionalmente, al modelo con T=0.15 lo sometimos a las mismas pruebas que a las redes recurrentes generales y obtuvimos

CUADRO AP.17

Caso	*Deseado*	*Predicción*	*%Error*
Ej1	3179749	3689735	16.04%
Ej2	3027566	3306850	9.22%
Ej3	3713998	3909721.25	5.27%
Ej4	5630243	4713152	16.29%
Ej5	5647959	5079704	10.06%
Ej6	6133903	5486171.5	10.56%
Ej7	4086884	4730231	15.74%
Ej8	3844403	3394742.75	11.70%
Ej9	4561935	3861846.5	15.35%
ej10	3336950	3827292	14.69%
ej11	4093933	3770501.25	7.90%
ej12	4059187	3995172.25	1.58%

Promedio:
11.20%
Máximo:
16.29%
Mínimo:
1.58%

Para verificar si los errores relativamente altos obtenidos eran debido a la falta de datos (ya que estábamos entrenando con datos hasta la semana 19/2002) intentamos repetir las pruebas, pero esta vez entrenando desde la semana 18/99 hasta la semana 51/2202, de la 19/99 a la 52/2002, etc. y obtuvimos

CUADRO AP.18

Semana	*Deseado*	*Predicción*	*%Error*
52/2002	2411754	2377379.5	1.43%
53/2002	2091221	2487752.5	18.96%
1/2003	2344265	2522370.75	7.60%
2/2003	2554344	2651609.75	3.81%

Promedio:
8.28%
Máximo:
18.96%
Mínimo:
1.43%

Vemos que aumentando la cantidad de datos de entrenamiento se mejora el error medio al predecir en un paso, lo que nos hace suponer que el modelo es válido (desde este punto de vista) para el problema y que con solo aumentar la cantidad de datos de entrenamiento se podrían obtener predicciones aceptables (Ver Sección IV).

I.3.3 Los aumentos de precios del gas

a) Red recurrente con entradas adicionales

Probamos una red totalmente recurrente con entradas S, A, S-1, A-1, V-1, T-1, salida V, 7 neuronas ocultas, entrenada con datos de semanas 18/1999-27/2002.

Obtuvimos: **CUADRO AP.19**

%Error CV	***17.51%***
MDL CV	***234.34***

Y al predecir:

CUADRO AP.20

Semana	*Deseado*	*Predicción*	*%Error*
28/2002	4561935	4204716	7.83%
29/2002	3336950	4278299.5	28.21%
.	4093933	4500833.5	9.94%
.	4059187	4779422	17.74%
.	4541634	4132965.25	9.00%
.	3460739	4062009.5	17.37%
	2929769	3462199.5	18.17%
	3312203	3614359	9.12%

Entrenando ahora con las semanas 18/1999-1/2003:

CUADRO AP.21

%Error CV	***17.7%***
MDL CV	***95.53***

Semana	*Deseado*	*Predicción*	*%Error*
2	2344265	3045558.5	29.92%
3	2554344	3102180	21.45%
4	2568176	3141033.75	22.31%
5	2384806	2734501.75	14.66%
6 a 18	.	.	.

b) Red de Jordan con entradas adicionales

Entrenamos ahora una red de Jordan como la descripta en III.3.2 "Redes de Jordan y Elman" con datos de la semana 18/1999-1/2003. Obtuvimos: ***%Error CV= 9.55% y MDL CV = -23.61*** y al predecir:

CUADRO AP.22

Semana	*Deseado*	*Predicción*	*%Error*
2	2344265	2365895.5	0.92%
3	2554344	2351020.25	7.95%
4	2568176	2383354.25	7.19%
5	2384806	2316061.25	2.88%
6	2355510	2338734.25	0.71%
7	4240721	2691635.25	36.52%
8	1716938	2356060	37.22%
9	1821053.875	2351360.25	29.12%
10	2572587	2351720.5	8.58%
11	2749742	2560280	6.59%
12	2518900	2424982.75	3.72%
13	2072692.875	2409411.25	16.24%
14	2939268	3196924.5	8.76%
15	2868848	3149047	9.76%
16	2455639	2660958.5	8.36%
17	2484590	2839490	14.28%
18	3822776	3790146.75	0.85%
-Errores-			
Prom: 11.74%	***Max: 37.22%***	***Min: 0.71%***	

Sección II: Predicciones iteradas para cada modelo

II.1 Predicciones iteradas para cada modelo

CUADRO AP.23

S	TDNN c/ aumento no focused d=3	Gamma c/ aumento d=3	Jordan c/ aumento	Tot recurr sin v-2	Tot recurr con v-2	MLP con aumento	TDNN pura no focused d=3
2	2725854	2872066.25	2365895.5	2664598.25	2822918.25	2490638	2709111
3	2407717.75	2918828.5	2345087.75	2385331.5	2830241.5	4455428	1861450.875
4	2171759.25	2706151	2336118.75	2493860.25	2838143.5	4453966	2452651.75
5	2975567.5	2474640.25	2316289.5	2459519.5	2936578.75	4455372	1902520.125
6	2832614	2351067	2335409.25	2424108	2966598.75	4453693	1981185.125
7	2726056	2167751.5	2321078.5	2417835.75	2931549.25	4453487	2799980.25
8	3185262.25	2494977.25	2322727.25	2545062	2902430	4453195	1976601.25
9	2073295.5	2362264.75	2396675.25	2448719.25	3011366.25	4452618	3139486.25
10	1904548.875	2418519.25	2331532.5	2619279.5	2867185.25	4451630	3505448.75
11	2481988	2327889.5	2431872	2582476.25	2943923.25	4450433	3714963.25
12	2950251.25	2379253.75	2453240.25	2545482	2952273.75	4449135	4452816.5
13	3524494.75	2385580	2415442.5	2541720.25	2883998.75	4447440	4194948.5
14	3659193.5	2357815.75	2437209.25	2771028.5	2873459.75	4445338	3873928.5
15	3304101.25	2499003.75	2886731	2802748.5	2973202.25	4443016	3389642.75
16	3384835.25	2626370.25	2934743.75	2813569.5	2908413.75	4440921	2349079.25
17	3663042	2561004.25	3046107.75	2931241	2924896.25	4437961	2004470
18	3200593.5	2767140	3306533	2803790.5	2969710	4435146	2347767.5
	5023961	3903211.25	3098392.75	3074444.5	2873124	4439900	4773325.5
	4670281.5	3499327.75	3653358.75	3211345.5	3005997.75	4435294	4246283
	4811737.5	3563877.75	3813149	3251905.75	2976368.75	4431574	4374336
	4684623.5	3901803.5	3895910.5	3269873.75	2986044.75	4427344	3882241
	5346856.5	3786259.5	3914454.75	3486589.25	2980338.5	4424020	4925221
	3859412.25	3941230.25	4033153.75	3377728.5	3046558.75	4421474	4227981.5
	5378501	4431828	4042976.5	3318587.75	2947342.75	4420295	5385771
	7277906	4464540	4085191.5	3742577	2966649.25	4419194	4710205
	7834570.5	4761751.5	4131369	3428461.5	3094625.75	4418414	4234484
	7241757	4896754.5	4164659.75	3432185	3052076.5	4417810	4621244
	6518082.5	4662937.5	4185984	4069502	3112956	4417309	5188501
	2924586.25	4130520.75	4027618	4347851	3384847.75	4416926	5193821.5
	3240792.5	3720516.25	3973918.25	4197618.5	3386186.25	4416562	4041010.75
	2627338.25	3149411.25	3941789	4030047.25	3392303.75	4416505	3512285.5
	2280069	2863743.25	3880421	4025356	3368306.75	4416032	3274541.5
	2526387	2667262	3752341.75	3651128	3355432.75	4415797	3338714.5
	2931991.75	3787465	3675268	3995882.75	3090252.5	4415691	3259072.5
	3971525.5	3962504.5	3450328.75	4290579	3230487	4415740	2843186.75
	3309714.25	3966049.75	3286657.5	3578034.25	3228881.75	4415706	2377475.75
	4077188	3532980.5	3190969.5	3889599	2979683	4415618	2634758
	2524643.5	2992561.25	3005508.25	3660844.25	3232148.5	4415641	2257162.75
	2113601.5	2525344.25	2928148.25	3238521.25	3052039.25	4415476	2171367.75
	2266100.25	2438093.25	2797047.75	3341613.5	2936631.5	4415484	2447572
	3437452.25	2253917.75	2694814.5	2913098.75	3074614.25	4415466	1751842.25
	3496854.5	2416508	2567038.25	2699645	2814509.5	4415361	2086420.5
	4981016.5	2535396	2470038.5	2798333.25	2698908.75	4415405	2348731
	5812365.5	2411327.5	2474841.75	2665616.5	2828124.5	4415378	2116093.5
	7084077.5	2417537.5	2407862.75	2605112.75	2658430.25	4415264	2324285
	4934594	2383191.5	2375907.25	2609905	2638913.25	4415273	2352392.25
	3363712.25	2334317.25	2364236.5	2521534	2649166.25	4415232	2586090.5
	2095770.125	2376314.25	2331608	2476607	2549160.75	4415187	2319977.25
	2622559.25	2376863.25	2320461.25	2444500	2511913	4415185	2590436
	2177510.25	2407894.75	2314624.75	2387008.5	2482174.5	4415158	2380614.25
	2595571.5	2459923.25	2298400.25	2446800	2385577	4415136	2505935
	2144344.25	2285106.25	2355960	2392262.75	2464013	4415130	3030244.75
	2114210.75	2317845.25	2352413	2290458.25	2415980	4415048	4674927
	2757274.25	2238623.75	2295187.25	2444320	2518355.75	4415060	3156668.5
	2593301	2068441.25	2345902.75	2438400.5	3086551	4429382	2201167
	3022525.5	2431138.5	2352098.5	2409009	2862849	4454530	2519294
	2819759	2605096.75	2322422.75	2479836.75	2897851.5	4454524	3445007.25
	2999357.75	2445878.5	2357372.75	2444466.75	3043870.5	4454194	2577651.75
	2732957.75	2465669.5	2329832.25	2530911	2914489.75	4453825	2760868.5

S	TDNN c/ aumento no focused d=3	Gamma c/ aumento d=3	Jordan c/ aumento	Tot recurr sin v-2	Tot recurr con v-2	MLP con aumento	TDNN pura no focused d=3
	2185942.25	2331075	2388695.75	2413566	3018125	4453285	2127431.75
	2499068.75	2332734.25	2322081.75	2422004	2919068	4452529	2392946.5
	3683638.25	2350555.5	2347169.75	2399795	2943646	4451607	3572365.75
	2833785.25	2393421.75	2329155.75	2383010.75	2908527.75	4450463	2800110
	3649318.25	2325430.75	2329000.5	2633902.5	2835237.25	4448967	3675154
	3336134.75	2278537.5	2620402.25	2743783.5	2960416.5	4447228	3263594.5
	3212781.25	2370472.25	2755487.75	2781003.25	2931534.75	4445619	3258123.75
	3957439.75	2348325.25	2831951.25	2695255.75	2933069	4443340	3895346.75
	3589466	2464340	2682264.25	2718729	2900526	4440784	3595923.25
	5094709.5	2753935.25	2767431.75	2961729.75	2898432.75	4437758	5097816
	5607821.5	2975396	3368387	3286179	2978660.75	4434963	5606585.5
	5440208	3399334	3828164.25	3405710.5	3039832.25	4432529	5427151
	5528296.5	3806346.75	3896191	3616637.25	3038997.5	4430074	5526858.5
	3905413.75	4199325	3972484	3356464.5	3102201.75	4427410	3890743.75
	3863176.25	4672846	3842102.25	3114524.25	2970895	4425252	3873138.75
	4408244	4641828.5	3706251.75	3333629.5	2916512.75	4422955	4413520
	4856729.5	4462479.5	3859315.5	3368767.5	2970605	4421342	4850343.5
	5713278.5	4405741	3938037.25	3994758	2941447.75	4420195	5699807
	4962849	4313761.5	4068606	4023105.75	3144857.25	4419188	4969667
	4647921.5	4535236	4127791.5	3810749.5	3047795.25	4418485	4648782.5
	4660849	4681336	4169402	3939251.5	3059029.75	4417805	4666300
	4990246.5	4617473.5	4189634.5	3999172.75	3079353.75	4417313	4992253
	5396318	4511290.5	4184221.75	4703869	3095886	4416928	5383938.5
	2905316.25	3865058.5	3989219	3472803.5	3313599.25	4416627	2905946.75
	2936762.5	3770290.5	4010070.75	3336933.25	2845045	4416320	2940297.75
	3123935	3390883	3987147	3871563.5	2975345.5	4416319	3125832.5
	3287556	3271937.5	3870546.75	3914941.5	2910963	4416096	3283951.75
	3273627.25	3357804.25	3693760.5	3800593.25	2850215.5	4415976	3268934
	3128843.25	3208804.25	3511354.25	3663145.25	2849393.25	4415866	3130820
	3626057	3114539.25	3294780	3550209.25	2969990.5	4415776	3628804.75
	2645769.5	3044097.5	3160153.5	3032931.25	2972692.75	4415634	2648635.25
	2261111.5	2830737	3016616.25	2881031.75	2774276	4415564	2258959.75
	2539504.25	2672689.25	2899243.5	2942456.75	2787410.5	4415611	2536799
	2265975.25	2572820.5	2802033.25	2850518.75	2819172.5	4415537	2265686.25
	1916582.5	3214200.75	2689261.75	2712791.25	2751083.75	4414890	1917644.625
	1896890.5	2795209.75	2692499.75	2593186.75	2695284.75	4414945	1896875.125
	2339028	2283089.5	2482234.75	2623616.5	2615227.75	4415028	2337002.5
	2294949	2923766	2445618.25	2585733	2654057.5	4415384	2294086.25
	3464168.5	2500251.5	2394321	2572073	2582087.5	4415056	2480690
	2982346.75	2548289.75	2372265.25	2444532.5	2612633	4415156	2175594
	4740759.5	2453317.25	2286357.75	2558035.25	2449142.25	4415235	2849420
	7320171	2642396	2348553.5	2623002.75	2585265.75	4415269	3130695
	3997452	2368026.75	2364550.75	2492269.5	2607843	4415141	3029624
	3731293.5	2148407	2338800.25	2483376.25	2476565.75	4415091	2629359.5
	2027473.5	2240141.75		2501589.75	2544252.5	4415122	2441442.25

II.2 Valores obtenidos al aplicar el "ensemble method" a las siete redes

CUADRO AP.24

REAL	Recurr sin v-2	TDNN pura no focused d=3	TDNN c/aumento no focused d=3	Gamma con aumento d=3	Jordan c/aumento	Tot recurr con v-2	MLP con aumento
2344265	2664598.25	2709111	2725854	2872066.25	2365895.5	2822918.25	2490637.5
2554344	2389453.25	2802681.5	2143003	3503731.75	2351020.25	2561632.75	2490677.25
2568176	2340697.25	2809267	2384944	3551868.75	2383354.25	2421361.75	2689453.5
2384806	2355009.25	2726262	2795620	3807154.5	2316061.25	2601041.25	2708356.25
2355510	2379824.25	2713826.75	2212919.25	3510817.5	2338734.25	2728270	2694399
4240721	2432194.25	4117191.75	3312516.75	3397964.5	2691635.25	2662632.75	2705313.25
1716938	2222872	2489850	3232728.25	3724796.75	2356060	2597380.75	2693956.25
1821054	2402239.75	2520970	2849647.75	3597756.25	2351360.25	2539963.25	2730151.25
2572587	2440831	2811378.75	3573579.75	3772434	2351720.5	2471412.25	2797738.75

REAL	Recurr sin v-2	TDNN pura no focused d=3	TDNN c/aumento no focused d=3	Gamma con aumento d=3	Jordan c/aumento	Tot recurr con v-2	MLP con aumento
2749742	2367811	2900987.25	2071973.25	3734098.75	2560280	2451534.5	2783593.5
2518900	2350705.75	2786055	1696007.88	3961649.25	2424982.75	2455161.5	2898533.5
2072693	2393497	2604294.5	2316095.5	3883296.5	2409411.25	2453608.75	3754147.25
2939268	2447383.5	3007983.75	2600537.75	3756403.25	3196924.5	3050511.25	3729677.25
2868848	2350822	2966812.75	2992610.75	3971334	3149047	2720753	3094771.5
2455639	2354613.5	2757246.5	3175352.25	4132017.5	2660958.5	2537948.75	3658980.75
2484590	2415046.75	2770294.5	3255864.25	3976627.75	2839490	3382451.25	4234557
3822776	2433650.5	3687567.75	4324198	3941217.75	3790146.75	3631092	3980261.75
MDL	62.2	-66.3	298.52	213.13	-23.61	75.17	7.92
%error sobre CV	11.49	11.88	12	11.49	9.55	11.86	11.9
%Error pred en 1 paso	13.66%	15.56%	16.28%	22.51%	0.92%	20.42%	6.24%
%prom error pred en 1 paso	14.14%	13.04%	23.71%	48.10%	11.11%	14.91%	24.94%
Coef(MDL)	0.791638751	1.22209567	0	0.286044486	1.079090178	0.74819108	0.97346911
Coef(MDL) normaliz	0.155207177	0.23960174	0	0.056081334	0.211564353	0.14668891	0.19085649
Coef(%Error)	0.393079781	0.30874358	0.277021417	0	0.959017717	0.09311838	0.72267497
Coef(%Error) normaliz	0.142748333	0.11212134	0.100601321	0	0.348270724	0.03381628	0.26244201
Coef(%Error CV)	0.0425	0.01	0	0.0425	0.204166667	0.01166667	0.00833333
Coef(%error CV) normaliz	0.133159269	0.03133159	0	0.133159269	0.639686684	0.03655352	0.02610966
MSE	3.7761E+11	1.24E+11	4.693E+11	1.561E+12	2.051E+11	2.97E+11	7.02E+11
MSE prom:	5.3384E+11						

Y el ensemble dio:

CUADRO AP.25

ENSEMBLE equiponderado	%error	ENSEMBLE (MDL)	%error	ENSEMBLE (%error CV)	%error	ENSEMBLE (%error 1 paso)	%error
2664440.107	13.66%	2613726.06	11.49%	2503788.119	6.80%	2531421.25	7.98%
2606028.536	2.02%	2587398.709	1.29%	2535128.473	0.75%	2429994.545	4.87%
2654420.929	3.36%	2608311.171	1.56%	2555998.592	0.47%	2506797.447	2.39%
2758500.643	15.67%	2620689.022	9.89%	2553312.38	7.07%	2528449.163	6.02%
2654113	12.68%	2625738.02	11.47%	2535556.949	7.64%	2480512.516	5.31%
3045635.5	28.18%	3030902.034	28.53%	2795104.503	34.09%	2879506.21	32.10%
2759663.429	60.73%	2544093.935	48.18%	2542420.102	48.08%	2537081.072	47.77%
2713155.5	48.99%	2569756.144	41.11%	2546202.86	39.82%	2533556.988	39.13%
2888442.143	12.28%	2658044.442	3.32%	2583189.926	0.41%	2660000.483	3.40%
2695754.036	1.96%	2704539.872	1.64%	2703486.324	1.68%	2576811.096	6.29%
2653299.376	5.34%	2680953.394	6.43%	2644493.824	4.99%	2506828.043	0.48%
2830621.536	36.57%	2799427.936	35.06%	2646385.78	27.68%	2774012.209	33.84%

ENSEMBLE equiponderado	%error	ENSEMBLE (MDL)	%error	ENSEMBLE (%error CV)	%error	ENSEMBLE (%error 1 paso)	%error
3112774.464	5.90%	3146898.26	7.06%	3174254.208	7.99%	3143612.736	6.95%
3035164.429	5.80%	2954423.249	2.98%	3129468.619	9.08%	2970204.191	3.53%
3039588.25	23.78%	2891416.138	17.75%	2840629.403	15.68%	2937536.122	19.62%
3267761.643	31.52%	3166710.186	27.45%	2988495.747	20.28%	3197516.05	28.69%
3684019.214	3.63%	3576456.067	6.44%	3625569.068	5.16%	3683249.661	3.65%
	Max:		Max:		Max:		Max:
	60.73%		48.18%		48.08%		47.77%
	Min:		Min:		Min:		Min:
	1.96%		1.29%		0.41%		0.48%
	Prom:		Prom:		Prom:		Prom:
	18.36%		15.39%		13.98%		14.82%
MSEs:							
2.98853E+11		2.3569E+11		2.3862E+11		2.48069E+11	
MSE promedio:	2.553E+11						

Observamos que al utilizar la siguiente forma de asignar pesos a las ponderaciones según *valor* (donde *valor* puede ser el MDL, %Error CV, etc.):

$$[\mathrm{MAX}(valor)-valor]/\mathrm{MAX}(valor)$$

algunos valores de la lista tienen ponderación 0 (los correspondientes al máximo de *valor*). Si cambiamos ligeramente la fórmula por

$$[\mathrm{MAX}(valor)-valor+1]/\mathrm{MAX}(valor)$$

eso ya no sucede. Los resultados obtenidos entonces son:

CUADRO AP.26

ENSEMBLE equiponderado	%error	ENSEMBLE (MDL)	%error	ENSEMBLE (%error CV)	%error	ENSEMBLE (%error 1 paso)	%error
2.664.440.107	13.66%	2.613.958.144	11.50%	2.607.625.969	11.23%	2653617.47	13.20%
2.606.028.536	2.02%	2.587.483.965	1.30%	2.580.954.922	1.04%	2.591.706.115	1.46%
2.654.420.929	3.36%	2.608.522.185	1.57%	2.619.614.137	2.00%	2.642.410.035	2.89%
2.758.500.643	15.67%	2.621.319.693	9.92%	2.685.936.373	12.63%	2.739.783.269	14.88%
2654113	12.68%	2.625.867.873	11.48%	2.612.185.976	10.90%	2.639.988.574	12.08%
3045635.5	28.18%	3.030.969.459	28.53%	2.957.035.895	30.27%	3.032.118.942	28.50%
2.759.663.429	60.73%	2.545.080.453	48.23%	2.682.835.918	56.26%	2.741.553.755	59.68%
2713155.5	48.99%	2.570.412.386	41.15%	2.654.113.154	45.75%	2.698.543.064	48.19%
2.888.442.143	12.28%	2.659.098.818	3.36%	2.780.490.528	8.08%	2.869.855.746	11.56%
2.695.754.036	1.96%	2.704.499.665	1.65%	2.698.488.539	1.86%	2.686.076.639	2.32%
2.653.299.376	5.34%	2680826.84	6.43%	2.650.185.316	5.21%	2.641.382.222	4.86%
2.830.621.536	36.57%	2.799.570.688	35.07%	2.765.467.063	33.42%	2.826.015.706	36.35%
3.112.774.464	5.90%	3.146.742.098	7.06%	3.134.516.608	6.64%	3.115.283.518	5.99%
3.035.164.429	5.80%	2.954.792.747	3.00%	3.068.514.849	6.96%	3.029.879.155	5.61%
3039588.25	23.78%	2.892.094.222	17.77%	2.969.226.996	20.91%	3.031.285.118	23.44%
3.267.761.643	31.52%	3167172.63	27.47%	3.169.000.019	27.55%	3.262.046.344	31.29%
3.684.019.214	3.63%	3.576.948.311	6.43%	3.663.348.479	4.17%	3.683.956.602	3.63%

ENSEMBLE equiponderado	%error	ENSEMBLE (MDL)	%error	ENSEMBLE (%error CV)	%error	ENSEMBLE (%error 1 paso)	%error
Promedio:	18.36%	Promedio:	15.41%	Promedio:	16.76%	Promedio:	18.00%
Max:	60.73%	Max:	48.23%	Max:	56.26%	Max:	59.68%
Min:	1.96%	Min:	1.30%	Min:	1.04%	Min:	1.46%

II.3 Valores utilizados como serie "real"

Los presentes valores de V* = (ventas+ruido) se utilizaron como valores de la serie "real" al construir las gráficas de las figuras III.20 y III.21 en III.3.4. Los valores de V* son tales que

$$V^* = \text{Ventas} + \text{ruido} = \lfloor \text{Ventas}+(1-2\theta)*50000 \rfloor$$

siendo θ un ruido con distribución uniforme en [0,1].

CUADRO AP.27

Semana desde 2/2000	Ventas desde semana 2/2000	V*
1	2006424	2001126
2	2386149	2419314
3	2590385	2592467
4	2639002	2592186
5	2256854	2233371
6	2299689	2312192
7	2205381	2182158
8	2481571	2501894
9	2445292	2464917
10	2669840	2712436
11	2550980	2535519
12	2569388	2576831
13	2940807	2929680
14	2690792	2663443
15	3249692	3277631
16	2664822	2682177
17	2709932	2718590
18	5409103	5393425
19	3593923	3574638
20	3513923	3540773
21	3840294	3823885
22	4584969	4568117
23	3788969	3785538
24	5432254	5430168
25	5223373	5177066
26	5439448	5400956
27	7819469	7791119
28	5953168	5971851
29	7247720	7227237
30	4099455	4148971
31	5191378	5208287
32	5363454	5368294
33	2898552	2903207
34	5003839	5038390
35	4179261	4176486
36	4706674	4697399
37	3480607	3492826
38	3416080	3460559
39	3315804	3284378
40	2946162	2914168
41	2587267	2608282
42	2314257	2302561
43	2445341	2421546
44	2572616	2599588
45	3209741	3256197
46	2359168	2337934
47	2473501	2435419
48	2463478	2425394
49	2634574	2663622
50	3034834	2986901
51	2539880	2580881
52	2019749	2045496
53	2291729	2246616
54	2714202	2754256
55	2174609	2205834
56	2432859	2448096
57	2540535	2555893
58	2775286	2732972
59	2502048	2524059
60	2196746	2162517
61	2473843	2437450
62	2654655	2641039
63	2390933	2432782
64	2436329	2408670
65	3090102	3121503
66	2829358	2807817
67	2820350	2829850
68	3448604	3470529
69	3925615	3897506
70	4691133	4715702
71	4803830	4852516
72	3882958	3916515
73	2796097	2767103
74	3864962	3913917
75	3408514	3416096
76	6063085	6061835
77	5774657	5809971
78	4415792	4431295
79	5187932	5227636
80	4978862	4965946
81	5926167	5903145
82	4519588	4475712
83	3577718	3527924
84	3202177	3249297
85	3186564	3205469
86	2715797	2712288
87	3865311	3891552
88	4106679	4073907
89	4417628	4404379
90	2881843	2866132
91	2579913	2622913
92	2750153	2774454
93	2993723	2968119
94	2579861	2562235
95	2797988	2819717
96	1957547	1931158
97	2936830	2924050
98	2538715	2540395
99	2027196	2007210
100	2709882	2714291
101	3007134	3047743
102	3268912	3241528
103	2286447	2297830
104	1773867	1756692
105	2687093	2679874
106	2271150	2313631
107	2346260	2357416

II.4 Valores del "ensemble" de las siete redes

CUADRO AP.28

Recurr sin v-2	TDNN pura no focused d=3	TDNN c/aumento no focused d=3	Gamma con aumento d=3	Jordan c/aumento	Tot recurr con v-2	MLP con aumento	Ensemble óptimo	*Deseado*	% Error
2664598,25	2709111	2725854	2872066,25	2365895,5	2822918,25	2490637,5	2457931	*2344265*	*4,85%*
2389453,25	2802681,5	2143003	3503731,75	2351020,25	2561632,75	2490677,25	2512409	*2554344*	*1,64%*
2340697,25	2809267	2384944	3551868,75	2383354,25	2421361,75	2689453,5	2562629	*2568176*	*0,22%*
2355009,25	2726262	2795620	3807154,5	2316061,25	2601041,25	2708356,25	2251665	*2384806*	*5,58%*
2379824,25	2713826,75	2212919,25	3510817,5	2338734,25	2728270	2694399	2356713	*2355510*	*0,05%*
2432194,25	4117191,75	3312516,75	3397964,5	2691635,25	2662632,75	2705313,25	4319551	*4240721*	*1,86%*
2222872	2489850	3232728,25	3724796,75	2356060	2597380,75	2693956,25	1974424	*1716938*	*15,00%*
2402239,75	2520970	2849647,75	3597756,25	2351360,25	2539963,25	2730151,25	2079026	*1821054*	*14,17%*
2440831	2811378,75	3573579,75	3772434	2351720,5	2471412,25	2797738,75	2359050	*2572587*	*8,30%*
2367811	2900987,25	2071973,25	3734098,75	2560280	2451534,5	2783593,5	2721361	*2749742*	*1,03%*
2350705,75	2786055	1696007,88	3961649,25	2424982,75	2455161,5	2898533,5	2470784	*2518900*	*1,91%*
2393497	2604294,5	2316095,5	3883296,5	2409411,25	2453608,75	3754147,25	2272412	*2072693*	*9,64%*
2447383,5	3007983,75	2600537,75	3756403,25	3196924,5	3050511,25	3729677,25	3013489	*2939268*	*2,53%*
2350822	2966812,75	2992610,75	3971334	3149047	2720753	3094771,5	2879070	*2868848*	*0,36%*
2354613,5	2757246,5	3175352,25	4132017,5	2660958,5	2537948,75	3658980,75	2407579	*2455639*	*1,96%*
2415046,75	2770294,5	3255864,25	3976627,75	2839490	3382451,25	4234557	2368009	*2484590*	*4,69%*
2433650,5	3687567,75	4324198	3941217,75	3790146,75	3631092	3980261,75	3841472	*3822776*	*0,49%*

Errores cometidos:

CUADRO AP.29

Ensemble óptimo	Deseado	% Error	Squared Error
2457931,276	2344265	4,85%	1,292E+10
2512409,229	2554344	1,64%	1758524998
2562629,304	2568176	0,22%	30765833,7
2251665,499	2384806	5,58%	1,7726E+10
2356712,586	2355510	0,05%	1446212,49
4319551,107	4240721	1,86%	6214185809
1974424,165	1716938	15,00%	6,6299E+10
2079025,979	1821054	14,17%	6,655E+10
2359050,21	2572587	8,30%	4,5598E+10
2721360,883	2749742	1,03%	805487802
2470784,199	2518900	1,91%	2315130320
2272412,116	2072693	9,64%	3,9888E+10
3013488,943	2939268	2,53%	5508748380
2879069,735	2868848	0,36%	104483856
2407578,571	2455639	1,96%	2309804836
2368009,275	2484590	4,69%	1,3591E+10
3841471,572	3822776	0,49%	349524394
		Prom: 4,37%	MSE: 1,66E+10
		Max: 15,00%	
		Min: 0,05%	