UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE POSGRADO EN CIENCIAS DE LA TIERRA

MODELACIÓN MATEMÁTICA Y COMPUTACIONAL DE SISTEMAS TERRESTRES

MODELO DE UNA RED NEURONAL ARTIFICIAL PARA EL PRONÓSTICO DE SERIES DE TIEMPO

TESIS

QUE PARA OPTAR POR EL GRADO DE

MAESTRO EN CIENCIAS

PRESENTA:

JUAN CARLOS ALVA PACHECO

DIRECTOR DE TESIS:

DR. TOMÁS MORALES ACOLTZI

CENTRO DE CIENCIAS DE LA ATMÓSFERA, UNAM

UNAM POSGRADO Ciencias de la Tierra

MÉXICO, CDMX, MARZO 2016.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

RESUMEN

En las series de tiempo (ST) se ha de observar el comportamiento que representa el estado de un sistema, éste definido por un conjunto de variables, contemplando las magnitudes de éstas en las ST. Analizando el cambio que pueda tener al medir la misma variable en diferentes lapsos de tiempo.

La historia de la humanidad, presenta diversas inquietudes y una de ellas es conocer el futuro, universalmente lo realiza utilizando información del presente y del pasado; llamando a este hecho, predicción. En diversas áreas científicas de la investigación se requiere conocer el comportamiento futuro de ciertos sucesos, sólo disponiendo de un conjunto de datos, por lo que se construyen modelos que aproxime la evolución de la ST.

Un modelo construido a partir de la comprensión de la naturaleza, es el de las Redes Neuronales Artificiales (**RNA**), el cual implementa modelos matemáticos idealizados en función de las neuronas biológicas, conocidos como Bio-inspirados.

Dentro del marco de las **RNA**, el modelo más apto y frecuentemente empleado en tareas como: modelado de sistemas, predicción, funciones de aproximación, clasificación de patrones, procesamiento de señales y mucho más; es el conocido perceptrón multicapa (PMC) con el algoritmo de aprendizaje retropropagación de errores (RPE), denominado 'red de retropropagación' o simplemente RPE.

Este modelo es empleado debido a sus capacidades generales que le favorecen, por ello se efectúa diversos experimentos sobre ST de funciones matemáticas no lineales, así como el de un sistema que presenta comportamiento caótico.

ABSTRACT

In the time series (TS) is to observe the behavior representing the state of a system it is defined by a set of variables, contemplating the magnitude of these in the TS. Analyzing the changes that may have to measure the same variable in different time frames.

The human history has several concerns and one of them is to know the future, universally makes using information from past and present, calling this fact prediction. In various areas of scientific research it is required to know the future behavior of certain events, having only at a set of data, so models that approximates the evolution of TS are built.

A model built from the understanding of nature, is the Artificial Neural Network (ANN), which implements a function idealized mathematical models of biological neurons, known as Bioinspired.

As part of the RNA, the model most suitable and mostly used on tasks such as system modeling, prediction, function approximation, pattern classification, signal processing and more; It is known Multilayer Perceptron (MLP) with error back propagation (BP) algorithm learning, called 'back-propagation network' or simply BP.

This model is used because of its general capabilities that favor, so various experiments ST nonlinear mathematical functions is carried out and the system having a chaotic behavior.

INTRODUCCIÓN

Trabajos con Redes Neuronales Artificiales (**RNA**) han sido implementados en diversas áreas científicas, pero también se han tenido que mejorar diferentes aspectos dentro de su ámbito, como es en el procesamiento que realizan. Trabajos como: "*A new class of multi-stable neural networks: Stability analysis and learning process"* [E. Bavafaye Haghighi *et al.*, 2015].

Se han encontrado trabajos para resolver ecuaciones diferenciales haciendo uso de las **RNA**, tales como: *"Numerical solution of sixth-order differential equations arising in Astrophysics by Neural Network"* [M. Khalid *et al.*, 2015]. *"A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks"* [Keith Rudd *et al.*, 2015]. *"Utilizing artificial neural network approach for solving two-dimensional integral equations"* [B. Asady *et al.*, 2014].

También se han de encontrar trabajos como: *"A regional neural network ensemble for predicting mean daily river water temperature"* [Jefferson Tyrell DeWeber *et al.*, 2014]. *"Constructing prediction interval for artificial neural network rainfall runoff models based on ensemble simulations"* [K.S. Kasiviswanathan *et al.*, 2013].

Actualmente existen sociedades de los cuales tienden a organizar conferencias internacionales, tales como: *International Neural Network Society* (*INNS,* http://www.inns.org/); *European Neural Network Society* (*ENNS*, http://www.e-nns.org/); *The International Joint Conference on Neural Networks* (*IJCNN*, http://www.ijcnn.org/); *The International Conference on Artificial Neural Networks* (*ICANN*).

El desglosar con más detalle, al ejemplificar y describir en cómo se puede construir un modelo de **RNA**, se ha de mostrar en este trabajo, así como los pasos y funciones que este modelo realiza, además de ser muy descriptivos sobre el proceso que efectúa el autómata, se realizan ejercicios en ST para que el modelo neuronal aprenda su comportamiento y efectúe un pronóstico, comparando lo que se obtiene con lo real.

Debemos señalar que el modelo neuronal es implementado para cada una de las ST, es decir, solo una ST (una variable) es ingresada a la **RNA** con el fin de usar sus propios datos y de ella obtener su evolución, dicho de otra forma, se trabaja en el espacio de fases; más sin en cambio, trabajos descritos en la literatura utilizan diferentes variables para obtener el comportamiento de otra distinta, trabajando en el espacio de estado; he aquí la diferencia y comparación respecto a este trabajo, siendo éste más parcial.

Se ha de someter en cuestión, si realmente sus capacidades generales que describen a las **RNA** son favorables, ya que el modelo programado es implementado en diversos ejercicios, trabajando ST con comportamientos periódicos y casi-periódicos, cuyos resultados son favorables al realizar pronóstico; más sin en cambio al trabajar con ST caóticas, para este método en cuestión presenta limitaciones.

El presente trabajo comprende tres capítulos; el primero de ellos abarca lo que son las **RNA**, mencionando ciertos modelos que hay dentro del ámbito, dando un preámbulo de como inició y se fue de desarrollando, citando algunos de los autores principales quienes demostraron la funcionabilidad de las **RNA** y referenciando sobre sus trabajos; además de hablar del modelo que es mayormente empleado y haciendo uso de éste para nuestro propósito. Antes de hacer uso de dicho modelo se desglosa el cómo se puede ingresar las ST a la **RNA**, seguido de cómo realizar pronóstico, esto, descrito en el capítulo dos, incluyendo los resultados alcanzados para las diferentes ST. En el capítulo tres se hace el análisis de lo obtenido para finalmente aportar con las conclusiones generales de todo el trabajo realizado.

CAPÍTULO 1

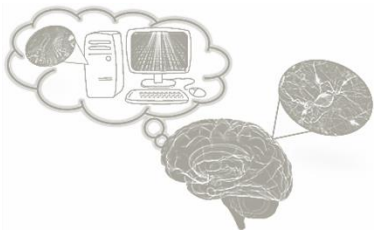
SISTEMA

NEURONAL

ARTIFICIAL

Considero al cerebro como un computador que dejará de funcionar cuando fallen sus componentes. No hay paraíso o vida después de la muerte para los computadores que dejan de funcionar, ese es un cuento de hadas de gente que le tiene miedo a la oscuridad.

Stephen Hawking



1.1 Modelo de una neurona artificial.

Las Redes Neuronales Artificiales (**RNA**), son la implementación de modelos matemáticos idealizados en la función de las 'neuronas biológicas'. Estos modelos son viables para resolver una amplia variedad de problemas, tales como: la clasificación, estimación y optimización de patrones; reconocimiento de visión, del habla y de caracteres; funciones de aproximación, **modelado de sistemas no lineales**, robótica, control, procesamiento de señal, **predicción**, economía, defensa, bioingeniería, etc. [Martín del Brío, 2007; Hilera González, 1995].

Una **RNA** es un sistema de procesamiento de información que está formado por elementos **no lineales**, llamados neuronas. Éstas están interconectadas entre ellas y son distribuidas en capas, de tal manera que emulan en forma simple la estructura y función neuronal del cerebro.

Cada modelo de neurona, es capaz de realizar algún tipo de procesamiento a partir de estímulos de entrada *(input,* x_i*)* y ofrecer una respuesta *(*salida-*output,* y_j*)*, por lo que las **RNA**, en conjunto, funcionan como redes distribuidas de cálculo en paralelo, similares a los sistemas cerebrales biológicos. Sin embargo, a diferencia de las computadoras convencionales, las cuales son programadas para realizar tareas específicas, las **RNA**, tal como los sistemas biológicos, deben ser entrenadas. Un atributo significativo de las neuronas es su habilidad para aprender, ya sea interactuando con el medio en el que se encuentran o bien, con información de entrada.

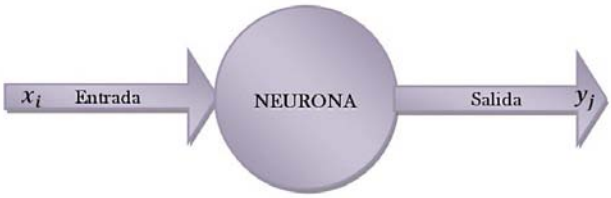


Figura 1.1 Esquema de un proceso neuronal.

Una **RNA** debe asemejarse al cerebro en dos aspectos, el conocimiento debe ser adquirido mediante un 'proceso de aprendizaje' y debe usar las conexiones neuronales (sinapsis) para almacenar la información, que posteriormente se dispongan para algún uso. Las **RNA** están inspiradas en la estructura y funcionamiento de los sistemas nerviosos, donde la neurona biológica es el elemento fundamental **(ver anexo A1)**.

El cerebro humano está constituido por un arreglo complejo de neuronas; se estima que el sistema nervioso contiene alrededor de 'cien mil millones de neuronas', que dan lugar a un sistema cognitivo robusto, con un diseño distinto al de las computadoras **(ver anexo A2)**.

En un modelo general de la neurona artificial, se le denomina como un dispositivo simple de cálculo que, a partir de un vector de entrada (*input*) procedente del exterior o de otras neuronas, proporciona una única respuesta o salida (*output*).

Sea una neurona $\boldsymbol{j}$ **(ver figura 1.2)**, los elementos que la constituyen son los siguientes:

- Conjunto de entradas $\boldsymbol{x_i}$.
- Pesos sinápticos $\boldsymbol{w_{ji}}$ de la neurona $\boldsymbol{j}$, que representan la intensidad de interacción entre cada neurona pre-sináptica $\boldsymbol{i}$ y la neurona post-sináptica $\boldsymbol{j}$.

- Regla de propagación $\boldsymbol{\sigma(w_{ji}, x_i) = \sum_i w_{ji} x_i}$, que proporciona el valor del potencial post-sináptico $\boldsymbol{h_j = \sigma(w_{ji}, x_i)}$ de la neurona $\boldsymbol{j}$, en función de sus pesos y entradas. Con frecuencia se añade al conjunto de pesos de la neurona un parámetro adicional $\boldsymbol{\theta_j}$, al que se denomina 'umbral', que se resta del potencial post-sináptico, por lo que el argumento de la función queda $\boldsymbol{h_j = \sigma(w_{ji}, x_i) - \theta_j}$, lo que representa añadir un grado de libertad a la neurona.

- Una función de activación $\boldsymbol{y_j = f(h_j)}$, que presenta simultáneamente la salida de la neurona y su estado de activación.

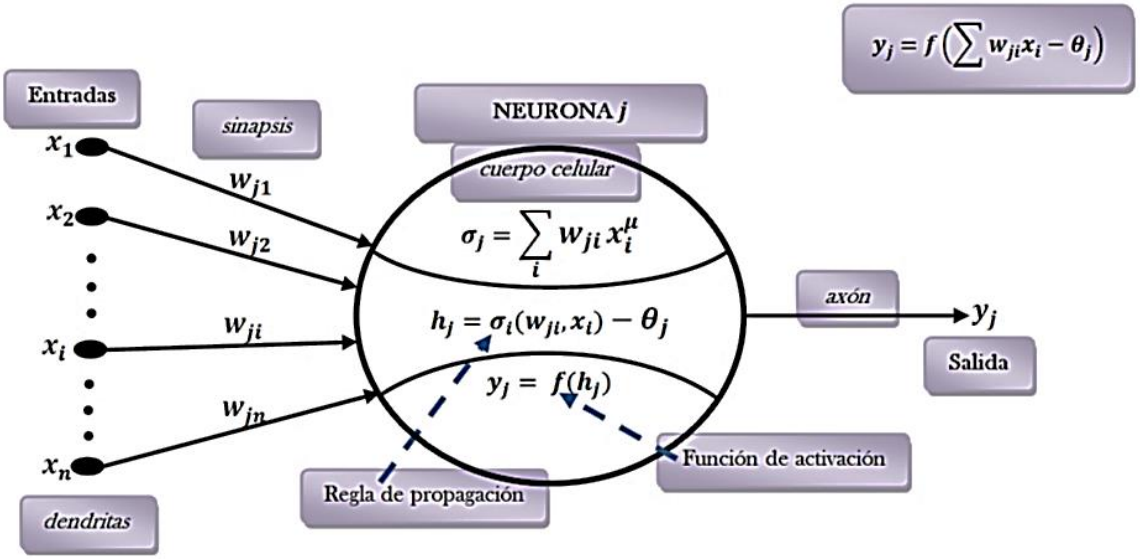


Figura 1.2 Modelo de una neurona artificial
[Modificada de: Martín del Brío, 2002; Rumelhart, 1986]

Conceptualizando los elementos que constituyen a la neurona artificial mencionados, se tiene:

Entrada – Salida (*input* - *output*). Variables que pueden ser binarias (digitales) o continuas (analógicas), dependiendo del modelo y aplicación. Para tareas de clasificación, se tendrían salidas digitales $\{\mathbf{0}, +\mathbf{1}\}$, mientras que para un problema de ajuste funcional de una aplicación multivariable continua se utilizarían salidas continuas pertenecientes a un cierto intervalo.

Peso sináptico. Define en este caso la intensidad de interacción entre la neurona pre-sináptica $\boldsymbol{i}$ y la post-sináptica $\boldsymbol{j}$. Dada una entrada positiva (procedente de un sensor o simplemente la salida de otra neurona), sí el peso es positivo tendrá a excitar a la neurona post-sináptica, sí el peso es negativo tendrá a inhibirla. Así, se habla de sinapsis excitadoras (de peso positivo) e inhibidoras (de peso negativo).

Regla de propagación. Permite obtener, a partir de las entradas y los pesos, el valor del potencial post-sináptico de la neurona:

$$\boldsymbol{h_j = \sigma_j(w_{ji}, x_i)} \qquad \textit{...(1)}$$

La función más habitual es de tipo lineal, y se basa en la 'suma ponderada' de las entradas con los pesos sinápticos:

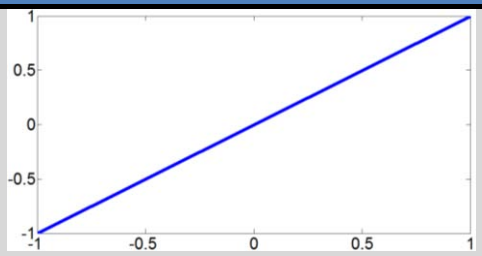
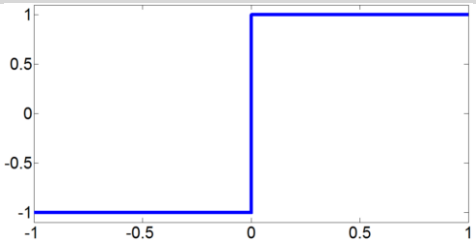
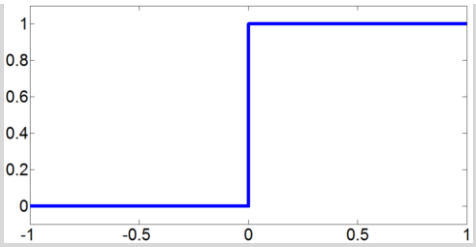
$$\boldsymbol{\sigma_j(w_{ji}, x_i) = \sum_i w_{ji} x_i} \qquad \textit{...(2)}$$

El parámetro adicional que se le agrega, en ocasiones, a la regla de propagación, para algunos modelos se le ha llamado *threshold*, es decir, 'umbral', y en otras *bias*, que no tiene una traducción clara para este caso, motivo por el que se le es llamado así a este parámetro (aunque a veces no haga el papel de umbral) [Martín del Brío, 2007].

Función de activación o de transferencia. Esta función proporciona el estado de activación actual a partir del potencial post-sináptico $\boldsymbol{h_j}$;

$$\boldsymbol{y_j = f(h_j)} \qquad \textit{...(3)}$$

La función de activación se suele considerar determinista, y en mayor parte de los modelos es monótona creciente y contínua, como se observa habitualmente en las neuronas biológicas. En ocasiones, los algoritmos de aprendizaje requieren que la función de activación cumpla la condición de ser derivable. Las más empleadas en este sentido son las funciones de tipo sigmoideo, la gaussiana y funciones sinusoidales **(ver tabla 1.1)**.

	Función	Rango	Gráfica
Identidad	$y = m \cdot x$	$[-\infty, +\infty]$	
Escalón	$y = signo\ (x)$	$\{-1, +1\}$	
	$y = H(x)$	$\{0, +1\}$	

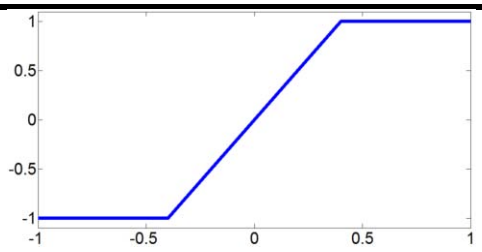
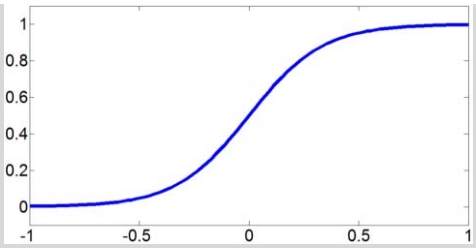
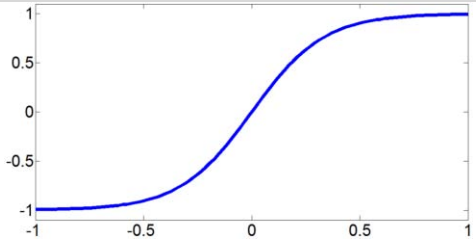
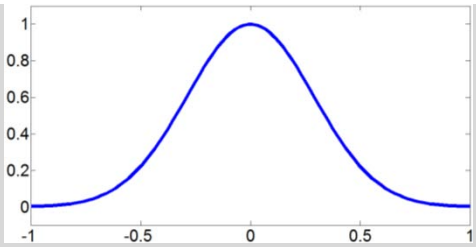
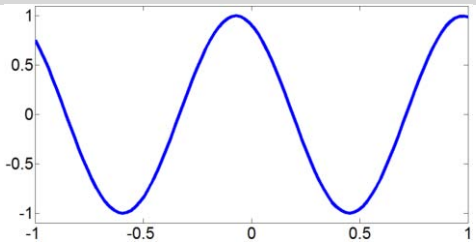
Lineal a tramos	$-1, si\ x < -l$ $y = x, si + l \leq x \leq -l$ $+1, si\ x > +l$	$[-1, +1]$	
Sigmoidea	$y = \frac{1}{1 + e^{-x}}$	$[0, +1]$	
Gaussiana	$y = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$ $y = tgh(x)$	$[-1, +1]$	
Sinusoidal	$y = A \cdot e^{-Bx^2}$	$[0, +1]$	
	$y = A \cdot sen(\omega x + \varphi)$	$[-1, +1]$	

Tabla 1.1 Funciones de activación.

Separar el concepto de regla de propagación y función de activación, permite considerar desde un punto de vista unificado varios modelos, que de otra manera, habría que tratar como casos especiales a una neurona.

1.2 Red neuronal artificial (RNA).

El número de neuronas, la disposición y el tipo de conexiones entre ellas determinan su estructura o patrón, denominada arquitectura o topología de la red neuronal. Los nodos de las **RNA** se conectan por medio de la sinapsis, esta estructura de conexiones sinápticas determina el comportamiento de la red.

Las conexiones sinápticas son direccionales, es decir, la información solamente puede propagarse en un único sentido, desde la neurona pre-sináptica a la neurona post-sináptica. Por lo general, las neuronas se suelen agruparse en unidades estructurales que comúnmente son denominadas 'capas'; y éstas a su vez, se pueden agruparse formando grupos neuronales llamados *clusters*.

La capa de entrada recibe datos o señales procedentes del exterior. La capa de salida es la que proporciona la respuesta de la red neuronal. La capa oculta, no tiene una conexión directa del exterior, recibe la información de la capa de entrada y proporciona el resultado que se envía a otra capa oculta o a la capa de salida. En el apartado de la estructura y funcionamiento de una **RNA**, se ejemplifica e ilustra la descripción de las capas.

La realización de un sistema neuronal artificial puede establecerse de la siguiente manera: el elemento de partida es la neurona artificial, que se organizará en capas; varias capas constituirán una red neuronal; junto con los módulos operacionales de una regla o dinámica de aprendizaje.

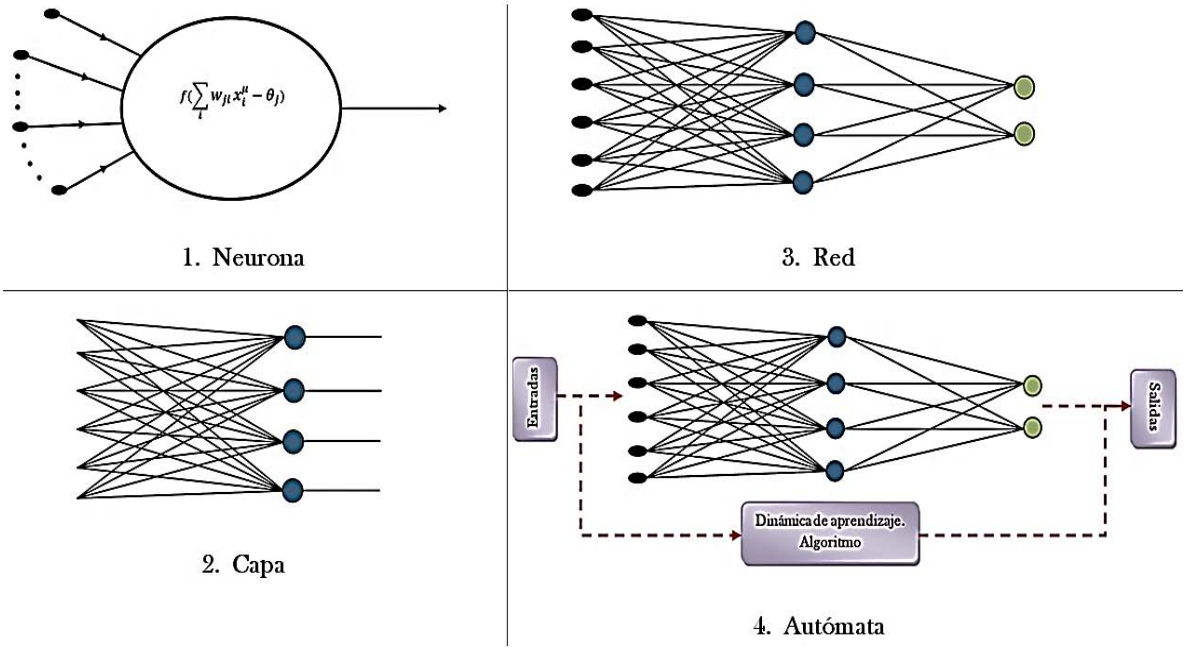


Figura 1.3 Sistema neuronal artificial (estructura jerárquica).

Se pueden establecer diferentes tipos de arquitecturas neuronales, de acuerdo con el número de capas. Así, en relación a su estructura, se hablan de redes monocapa, son aquellas compuestas por una única capa de neuronas; y redes multicapa (*layered networks*), son aquellas cuyas neuronas se organizan en varias capas.

De acuerdo al tipo de conexiones o el circular de datos en la red neuronal, se puede hablar de **redes unidireccionales** (*feedforward*), en las que su información circula en un único sentido, desde la neurona de entrada hacia la neurona de salida; y de **redes recurrentes** o **realimentadas** (*feedback*), cuya información puede circular entre las capas en cualquier sentido [Corchado J. M., 2000].

Mencionando algunas de las ventajas de las redes neuronales artificiales según Sánchez Camperos, 2006, en la siguiente tabla:

Las ventajas que ofrecen las RNA son:	
No linealidad.	El procesador neuronal es básicamente **no lineal** y, por consecuencia, la red neuronal también.
Transformación entrada (input) – salida (output).	El proceso de aprendizaje consiste básicamente en presentar a la red un ejemplo y modificar sus pesos sinápticos de acuerdo con su respuesta.
Adaptabilidad.	La red tiene la capacidad de adaptar sus parámetros, aún en tiempo real.
Tolerancia a fallas.	Debido a la interconexión masiva, la falla de un procesador no altera seriamente la operación.
Uniformidad en el análisis y diseño.	Esto permite garantizar características precisas.
Analogía con las redes biológicas.	Esto permite la utilización mutua del conocimiento de las dos áreas.

*Tabla 1.2 Ventajas que ofrecen las **RNA**.*

1.3 Mecanismo de entrenamiento (aprendizaje).

La regla de 'aprendizaje' es una de las características más importantes de una **RNA** y cuya finalidad es realizar un ajuste de los parámetros de la red neuronal, a partir de la información que se le ha de ingresar.

Comentando sobre la bio-inspiración, para hacer uso del modelo neuronal artificial, el cerebro es capaz de **'aprender'** y realizar procesos de manera rutinaria, mediante la experiencia.

Considerando que nuestro sistema nervioso coordina y controla todas nuestras conductas, desde las respuestas más simples hasta las emociones más complejas. El sistema que ha desarrollado durante la etapa prenatal de un animal, por ejemplo, posee información

indispensable para su supervivencia, pero a la vez debe modificarse cada día y **'aprender'** (que nunca permanezca igual), pues su objetivo primordial es seguir existiendo [Modificado de: González Hortensia *et al.*, 2012].

Actualmente el estudio del funcionamiento del cerebro facilita cada vez más la construcción y la explotación de los dispositivos automáticos. El problema de la confiabilidad, la probabilidad de conservar los índices de calidad en el transcurso de un tiempo dado, es clave de la automatización, resuelto brillantemente por la naturaleza viviente en el cerebro que, a nivel humano, es capaz de crear dispositivos automáticos cada vez más complejos [Pavel Simonov, 1968].

En el caso de las **RNA**, se puede considerar que el adquirir conocimiento, ha de residir en los parámetros de las conexiones neuronales, representando la intensidad entre ellas.

Para el proceso de aprendizaje (también recibe el nombre de 'algoritmo de aprendizaje' o 'modo aprendizaje') implica un cierto número de cambios para determinar un conjunto de pesos sinápticos, modificando estos valores, permitiendo que la red realice el tipo de procesamiento a desear. El entrenamiento se realiza, normalmente, a partir de la optimización de una 'función de error', que mide la capacidad actual del proceso que adquirió la red.

Un mecanismo de ajuste es: denominando $\boldsymbol{w_{ji}(t)}$ al peso que conecta la neurona pre-sináptica $\boldsymbol{i}$ con la post-sináptica $\boldsymbol{j}$ en la iteración $\boldsymbol{t}$, el algoritmo de aprendizaje, en función de las señales que en el instante $\boldsymbol{t}$ llegan procedentes del entorno, proporcionará el valor

$\Delta w_{ji}(t)$**,** que da la modificación que debe incorporarse en dicho peso y el cual quedará actualizado de la forma:

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}(t) \qquad \text{...(4)}$$

El proceso de aprendizaje es usualmente iterativo, actualizándose los pesos de igual manera, una y otra vez, hasta que la red neuronal alcanza el rendimiento deseado [Martín del Brío, 2002].

Cuando se construye un sistema neuronal, se parte de un cierto modelo de neurona y de una determinada arquitectura, antes de empezar el aprendizaje se establece valores a los pesos sinápticos como 'aleatorios', **(ver proceso de entrenamiento).**

Frecuentemente se emplean dos tipos básicos de aprendizaje que son el **'supervisado'** y el **'no supervisado'**, este último también llamado 'aprendizaje auto-organizado'. Ambos pretenden estimar funciones entrada/salida multivariable o densidades de probabilidad, la diferencia está en que el supervisado dispone de información sobre la salida deseada de la red (*target* -objetivo-), la cual le proporciona cierta información sobre estas funciones, como la distribución de las clases, etiquetas de los patrones de entrada o salidas asociadas a cada patrón, hecho que no ocurre en el 'no supervisado'.

En el **aprendizaje supervisado**, cada patrón de entrada que recibe la red, junto con la salida deseada u objetivo, los pesos son ajustados iterativamente hasta que la salida tienda a ser a lo esperado, utilizando para ello información detallada del error que comete en cada paso.

Sea $\boldsymbol{E}[\boldsymbol{W}]$ un funcional que representa el error esperado de la operación de la red, expresado en función de sus pesos sinápticos $\boldsymbol{W}$. En el aprendizaje supervisado se pretende estimar una cierta función multivariable desconocida $\boldsymbol{f}: \boldsymbol{R^n} \rightarrow \boldsymbol{R^m}$ a partir de muestras. Se pretende que el comportamiento de entrenamiento, sea minimizar el error, o bien, que éste llegue a un cierto rango aceptable, dependiendo a lo que se quiera abordar.

El **aprendizaje auto-organizado** se basa en la estimación de la función de densidad de probabilidad $\boldsymbol{p}(\boldsymbol{x})$ que describe la distribución de patrones $\boldsymbol{x}$**,** pertenecientes al espacio de entrada $\boldsymbol{R^n}$ a partir de muestras (ejemplos), reconociendo regularidades, extraer rasgos o agrupar patrones según su similitud, lo que determina la salida de la red. Para este tipo de aprendizaje, sólo se le da a la red una multitud de patrones sin adjuntar la respuesta.

También existe el tipo de **aprendizaje reforzado** (*reinforcement learning*), la red toma el error entre la entrada y la salida para maximizar los valores esperados de una función criterio, llamada señal reforzadora [Hassoon M.H., 1995] (solamente le indicamos lo bien o lo mal que está actuando, pero sin proporcionar más detalles), con este criterio la red modifica los pesos de cada neurona así como su salida. La similitud que tiene con el caso 'no supervisado', es la de no suministrar explícitamente la salida deseada. En ocasiones se denomina aprendizaje por premio-castigo.

1.4 El perceptrón.

Cuando se iniciaron las investigaciones sobre las **RNA**, de acuerdo a [Sánchez Camperos, 2006], algunos investigadores se destacaron por sus contribuciones pioneras:

- McCulloc y Pitts (1943) por introducir la idea de redes neuronales como máquinas de cálculo.
- Hebb (1949) por postular la primera regla de aprendizaje auto-organizado.
- Ronsenblat (1958) por proponer al 'perceptrón' como el primer modelo para aprendizaje supervisado.

El 'perceptrón', o también conocido como 'perceptrón simple', es un modelo unidireccional, compuesto por dos capas, una de entrada y otra de salida **(ver figura 1.4)**, con pesos ajustables y un umbral. Es la forma más simple de una red neuronal utilizada para la clasificación de patrones linealmente separables, es decir, cuyas regiones de decisión pueden ser separadas mediante una única condición lineal o hiperplano.

La importancia histórica del perceptrón radica en su carácter del entrenamiento, pues el algoritmo permite determinar automáticamente los pesos sinápticos que clasifican un conjunto de ejemplos etiquetados; considerado a este algoritmo de aprendizaje, de los denominados **'corrección de errores'**, los cuales ajustan los pesos en proporción a la diferencia existente entre la salida actual de la red y la salida deseada, con el objetivo de minimizar el error actual de la red.

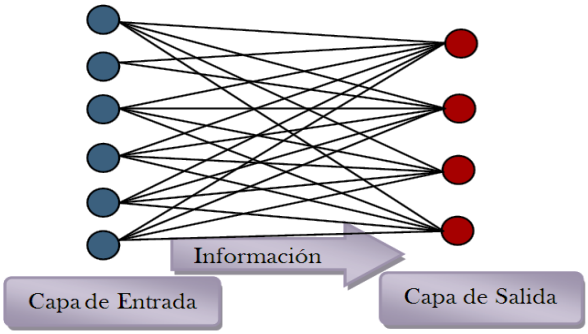


Figura 1.4 Perceptrón simple.

La operación de un perceptrón con $\boldsymbol{n}$ neuronas de entrada y $\boldsymbol{l}$ neuronas de salida, puede escribirse de esta manera:

$$\boldsymbol{y_j = H\left(\sum_{i=1}^{n} w_{ji}x_i - \theta_i\right), \qquad \forall j, \qquad 1 \leq j \leq l} \qquad \textit{...(5)}$$

La función de activación, o de transferencia, de las neuronas de la capa de salida es de tipo escalón $\boldsymbol{H(\cdot)}$ **(ver modelo de una neurona artificial).**

La forma más habitual de la regla de aprendizaje (regla del perceptrón) se expresa:

$$\boldsymbol{\Delta w_{ji}^{\mu}(t) = \varepsilon \cdot (t_j^{\mu} - y_j^{\mu})x_i^{\mu}} \qquad \textit{...(6)}$$

El ajuste de los pesos en la iteración $\boldsymbol{t}$ será:

$$\boldsymbol{w_{ji}(t+1) = w_{ji}(t) + \sum_{\mu=1}^{m} \Delta w_{ji}^{\mu}(t)} \qquad \textit{...(7)}$$

donde $\boldsymbol{m}$ es el conjunto de patrones $\boldsymbol{x^{\mu}}$ $\boldsymbol{(\mu = 1, \ldots, m)}$, y con t^{μ} que son sus salidas deseadas.

El **'ritmo de aprendizaje'** $\boldsymbol{\varepsilon}$ es de gran utilización práctica, puesto que un valor pequeño implica un aprendizaje lento, mientras que uno excesivamente grande puede conducir a oscilaciones en el entrenamiento, al introducir variaciones en los pesos excesivamente amplias.

1.5 El Perceptrón multicapa (PMC).

Si se añaden capas intermedias (ocultas) a un perceptrón simple, se obtiene un **'perceptrón multicapa (PMC)'** (*Multi-Layer Perceptron,* ***MLP*** por sus siglas en inglés). Inicialmente fue desarrollado por *Werbos P. J.* (1974), ya que al presentar desventajas el perceptrón simple, al no ser capaz de resolver problemas de clasificación que no fuesen linealmente separables [Haykin S., 1999].

La arquitectura del **PMC** suele entrenarse mediante el algoritmo denominado **'retropropagación de errores (RPE)'** (*Back-Propagation*, ***BP*** por sus siglas en inglés), pero en muchas ocasiones el conjunto $arquitectura\ PMC + aprendizaje\ RPE$ suele denominarse **'red de retropropagación'**, simplemente **RPE** [Martín del Brío, 2007] **(ver figura 1.5)**. *Werbos P. J.* fue quien también introdujo por vez primera el **RPE** en su tesis doctoral en 1974.

La descripción de la arquitectura más común del **PMC** se muestra en la figura 1.5, donde $\boldsymbol{x_i}$ son las entradas de la red; en la capa oculta se presentan lo pesos $\boldsymbol{w_{ji}}$ y umbrales $\boldsymbol{\theta_j}$, $\boldsymbol{y_j}$ son las salidas de ésta; $\boldsymbol{z_k}$ son las salidas de la capa final, por lo que sus pesos y umbrales

para la conexión de estas últimas capas serán $\boldsymbol{w_{kj}}$ y $\boldsymbol{\theta_k}$; por otro lado $\boldsymbol{t_k}$ son las salidas objetivo (*target*). Todo esto se expresa matemáticamente de la siguiente manera:

$$z_k^{\mu} = f\left(\sum_j w_{kj}\left[y_j^{\mu}\right] - \theta_k\right) = f\left(\sum_j w_{kj} f\left[\sum_i w_{ji}\, x_i^{\mu} - \theta_j\right] - \theta_k\right) \quad \ldots(8)$$

Siendo $\boldsymbol{f}(\cdot)$ la función de transferencia, usualmente de tipo sigmoideo **(ver modelo de una neurona artificial).**

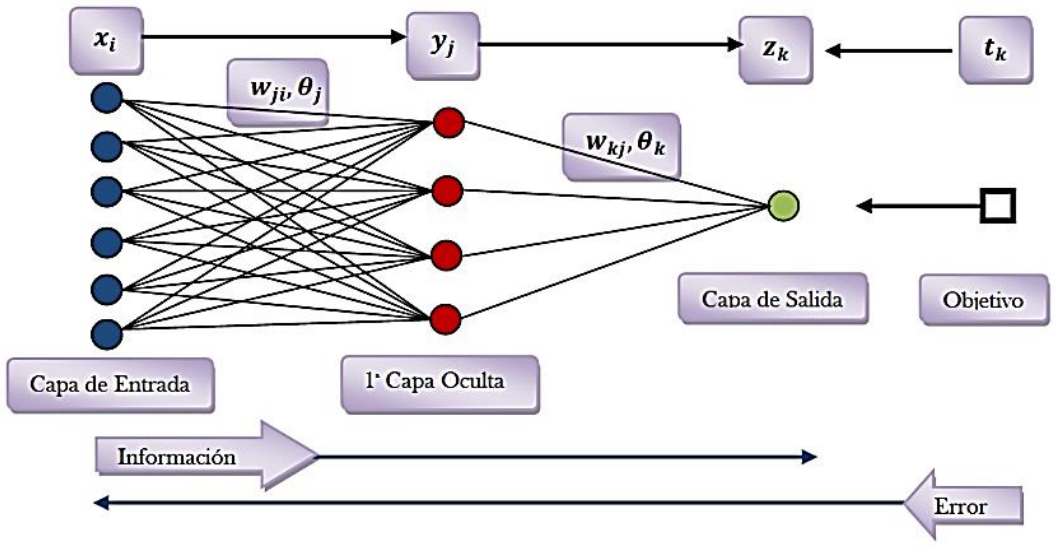


Figura 1.5 Arquitectura de una red neuronal del tipo **PMC**.

En el apartado de la estructura y funcionamiento de una **RNA**, se ejemplifica e ilustra con más detalle esta descripción del **PMC** así como su expresión matemática.

La arquitectura neuronal mostrada en la figura 1.5, es la más común, aunque existen numerosas variantes, como: incluir neuronas **no lineales** en la capa de salida (solución que se adopta especialmente en problemas de clasificación), introducir más capas ocultas, emplear otras funciones de activación, limitar el número de conexiones entre una

neurona y las de la capa siguiente, introducir dependencias temporales o arquitecturas recurrentes, etc...

El proceso que ha de realizar este tipo de red, se resumiría en dos etapas:

- Etapa hacia adelante. Se fijan los parámetros de la red y se presenta una señal de entrada a la red, que se propaga hacia adelante para producir la salida.
- Etapa hacia atrás. El error entre la salida y el objetivo, la red se propaga hacia atrás, cuyos parámetros se modifican para minimizar dicho error.

De acuerdo a Sánchez Camperos (2006), las características distintivas del **PMC**, o mejor dicho, el gran significativo que fue el agregar capas intermedias a la arquitectura neuronal (capas ocultas), son:

> El modelo de cada neurona en la red incluye una función de activación **no lineal**. Lo importante de esta característica es la **no linealidad**, pues ha de suavizar la señal que se ha de filtrar, en otras palabras, en cualquier punto existe su derivada.
>
> La red contiene una o más capas ocultas que no son parte de las entradas o salidas de la red. Estas neuronas ocultas permiten que la red aprenda tareas complejas por la extracción progresiva de las características principales de los patrones de entrada.

1.6 PMC, aproximador universal de funciones.

Un perceptrón de dos capas ocultas es suficiente para representar cualquier función (no necesariamente booleana), esto fue demostrado por Lapede y Farber [Lapades A. S., Faber R. M. 1987].

Hetcht-Nielsen [Hetcht-Nielsen, 1987, 1990], aplicando el teorema de Kolmogorov demostró que una arquitectura de características similares al perceptrón multicapa, con una única capa oculta, resultaba ser un aproximador universal de funciones.

Diversos grupos propusieron teoremas muy similares que demostraban matemáticamente que un perceptrón multicapa convencional, de una capa oculta, constituía un aproximador universal de funciones [Funahashi, 1989; Hornik *et al.*, 1989].

TEOREMA [Funahashi, 1989]

Sea $\boldsymbol{f(x)}$ una función no constante, acotada y monótona (uniforme) creciente. Sea $\boldsymbol{K}$ un subconjunto compacto (acotado y cerrado) de $\boldsymbol{R^n}$, sea un número real $\boldsymbol{\varepsilon} \in \boldsymbol{R}$, y sea un entero $k \in \boldsymbol{Z}$, tal que $k \geq 3$, que fijamos. En estas condiciones se tiene que:

Cualquier mapeo:

$$g\colon\ x \in K \ \rightarrow\ [g_1(x), g_2(x), \dots, g_m(x)] \in R^m$$

Con $\boldsymbol{g_i(x)}$ sumables en $\boldsymbol{K}$, puede ser aproximado en el sentido de la topología $\boldsymbol{L_2}$ en $\boldsymbol{K}$ por el mapeo entrada-salida representado por una 'red neuronal unidireccional' de k

capas ($k-2$ capas ocultas), con $\boldsymbol{f(x)}$ como función de transferencia de las neuronas ocultas, y funciones lineales para las capas de entrada y de salida.

En otras palabras:

$\forall \boldsymbol{\varepsilon} > \mathbf{0}$, $\exists$ un **PMC** de las características anteriores que implementa el mapeo

$$g'\colon\ x \in K \ \rightarrow\ \left[g'_1(x), g'_2(x), \dots, g'_m(x)\right] \epsilon\ R^m$$

De manera que

$$d_{L_{2(k)}}(g, g') = \left(\sum_{i=1}^{m} \int_k |g_i(x_1, \dots, x_n) - g'_i(x_1, \dots, x_n)| dx\right)^{1/2} < \varepsilon$$

Las funciones sigmoideas empleadas habitualmente en el perceptrón multicapa, cumplen las condiciones exigidas a $\boldsymbol{f(x)}$. En [Hornik *et al.*, 1989] se llega a un resultado similar, considerando funciones de activación sigmoideas no necesariamente continuas.

$$f(x) = \frac{1}{1+e^{-x}} \qquad \textit{...(9)}$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} = \tanh(x) \qquad \textit{...(10)}$$

Un **PMC** de una capa oculta puede aproximar hasta el nivel deseado de cualquier función contínua en un intervalo*, por lo tanto, las redes neuronales multicapa unidireccionales son aproximadores universales de funciones.

*El teorema permite elegir k con la restricción $k \geq 3$ si se elige $k = 3$ se tiene una sola capa oculta. No obstante, pueden emplearse más capas ocultas obteniéndose, en ocasiones, resultados más eficientes o una mejor generalización.

A partir de la expresión que define la operación de este tipo de red

$$g'_k(x) = \sum_j w'_{kj} y_j - \theta'_k = \sum_j w'_{kj} \cdot f\left(\sum_i w_{ji} x_i - \theta_j\right) - \theta'_k \qquad \textit{...(11)}$$

observando que la $\boldsymbol{g}'(\boldsymbol{x})$ dada por el perceptrón multicapa representa una cierta función $\boldsymbol{g}(\boldsymbol{x})$, como un desarrollo en funciones sigmoides $\boldsymbol{f}(\boldsymbol{x})$, lo cual posee una clara analogía con la representación convencional de una función periódica como un desarrollo en serie de *Fourier* de sinusoides [Principe, 2000]. También se ha establecido paralelismo entre el perceptrón multicapa y otros tipos de transformaciones, como la de *Gabor* o las *Wavelets*.

Autores como Cybenko (1989), Hornik (1989) han demostrado, independientemente, que el **PMC** es un aproximador universal, en el sentido de que cualquier función contínua sobre un compacto de $\boldsymbol{R^n}$, puede aproximarse con un perceptrón multicapa con al menos una capa oculta de neuronas. Este resultado sitúa al **PMC** como una nueva clase de funciones, como pueden ser también los polinomios, las funciones trigonométricas, los *splines*, etc., para aproximar o interpolar relaciones **no lineales** entre datos de entrada y salida.

La habilidad del **PMC** para 'aprender' a partir de un conjunto de ejemplos, aproximar relaciones **no lineales**, filtrar ruido en los datos, etc., hace que sea un modelo adecuado para abordar problemas reales, sin que esto indique que sean los mejores modelos para aproximar funciones. Cada una de las clases de **PMC** que sirven como aproximadores de funciones, tienen sus propias características; se conocen ciertas condiciones bajo las cuales un método es preferible a otro, pero en ningún caso se puede decir que un método

sea absolutamente el mejor. Serán las consideraciones prácticas de cada problema las que determinen la elección de un aproximador u otro.

El **PMC** es una de las arquitecturas más empleadas para resolver problemas, han sido aplicadas con éxito en una gran variedad de áreas como en: el reconocimiento de habla, reconocimiento de caracteres ópticos, reconocimiento de caracteres escritos, control de procesos, 'modelización de sistemas dinámicos', conducción de vehículos, diagnósticos médicos, 'predicción de series temporales', etc...

1.7 Aprendizaje por RPE.

Las reglas por corrección de errores, son manipuladas sobre los datos a comparar que originalmente se proponen como unidades de entrenamiento simples. Estas reglas manejan el error de la salida hasta aproximarlo a cero (según la aplicación). Algunos ejemplos son la regla del perceptrón, la regla de los cuadrados mínimos (*Least Mean Square*, LMS por sus siglas en inglés) y la regla delta generalizada [Hassoun M. H., 1995].

La regla de aprendizaje LMS es otra regla de aprendizaje que se utiliza para encontrar los pesos óptimos $\boldsymbol{W}$ y se desarrolla al iniciar la fase de entrenamiento, se inicializa el vector de pesos de manera aleatoria, después se calcula la suma de los pesos por las entradas, se compara el resultado con el valor esperado y con ello se obtiene el error, éste se utiliza para ajustar los pesos.

El algoritmo **RPE** aparece como una consecuencia natural de extender el algoritmo LMS, para ello se hace uso de la regla de la cadena, esto debido a que la función de error estará en función, tanto de los pesos de salida y los pesos de la capa oculta.

La función de error (función de coste) de la cual parte del **'error cuadrático medio'***:*

$$E(w_{ji}, \theta_j, w_{kj}, \theta_k) = \frac{1}{2}\left[\sum_k t_k^{\mu} - z_k^{\mu}\right]^2 \qquad \ldots(12)$$

Para resolver un problema en la búsqueda de un mínimo de una función, aparecen una serie de procedimientos que, generalmente, se dividen en dos grupos: métodos de 'búsqueda global' en donde se busca el 'mínimo global' de la función objetivo, proporcionando los pesos $\boldsymbol{W}$ que dan el valor más pequeño de dicha función sobre todo su dominio; métodos de 'búsqueda local', que buscan el 'mínimo más cercano' de la función objetivo en relación al estado inicial de los pesos **(ver figura 1.6).**

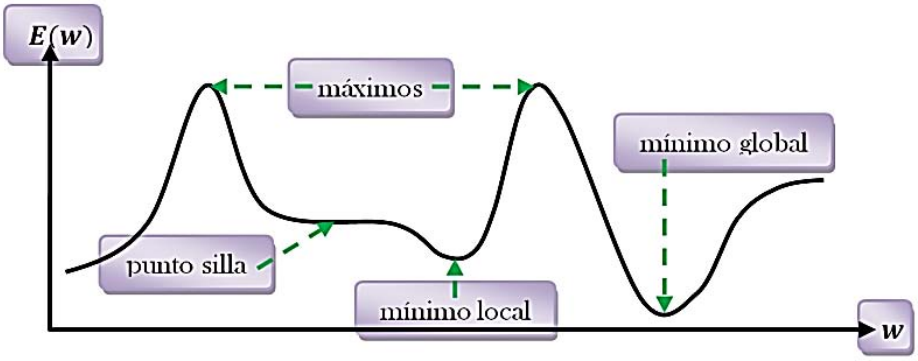


Figura 1.6 Búsqueda de un mínimo de una función.

La búsqueda de mínimos puede ser efectuada por los métodos basados en: **descenso por gradiente**, gradientes conjugados (que pueden ser la de direcciones conjugados,

conjugación Gram-Schmidt, propiedades de los residuales, método de los gradientes conjugados versión lineal y **no lineal**), descenso más inclinado, etc. [Pérez Pérez, 2001].

La minimización del error del aprendizaje **RPE** habitualmente se lleva a cabo mediante el **descenso por gradiente** **(ver figura 1.7)**; en el que se tendrá un gradiente respecto de los pesos de la capa de salida w_{kj} y otro, respecto de los de la oculta w_{ji}:

$$\delta w_{kj} = -\varepsilon \frac{\partial E}{\partial w_{kj}} \qquad ...(13)$$

$$\delta w_{ji} = -\varepsilon \frac{\partial E}{\partial w_{ji}} \qquad ...(14)$$

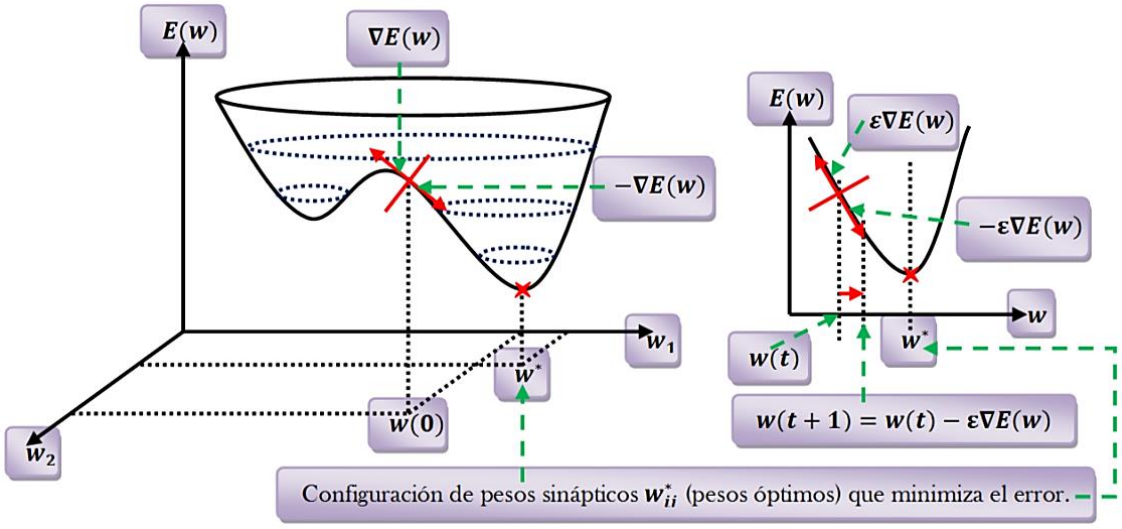


Figura 1.7 Superficie de error $E(W)$ en el espacio de los pesos W. Descenso por gradiente.

Aclarando, que la arquitectura neuronal del **PMC** y su respectivo algoritmo de aprendizaje (**RPE**), expresados en este escrito, son de tres capas (una de entrada, una oculta y una de salida), independientemente del número de neuronas que contengan cada una. Se hace esta aclaración ya que las expresiones matemáticas y figuras que se han presentado hasta ahora, han sido de este tipo de arquitectura neuronal. Esto no limita a que haya más capas

y cuya arquitectura cambie, así como su expresión matemática, simplemente se puede hacer una extensión de lo presentado.

Siguiendo con la minimización del error del aprendizaje mediante el **descenso por gradiente**. Las expresiones de actualización de los pesos se obtienen derivando, como se muestra en las ecuaciones anteriores **(ecuación 13 y 14)**, teniendo en cuenta las dependencias funcionales y aplicando adecuadamente la regla de la cadena:

$$\delta w_{kj} = -\varepsilon \Delta_k y_j \qquad \text{...(15)}$$

$$\Delta_k = [t_k - g(h_k)] \frac{\partial g(h_k)}{\partial h_k} \qquad \text{...(16)}$$

$$h_k = \sum w_{kj} y_j - \theta_k \qquad \text{...(17)}$$

$$\delta w_{ji} = -\varepsilon \Delta_j x_i \qquad \text{...(18)}$$

$$\Delta_j = \left[\sum_k \Delta_k w_{kj}\right] \frac{\partial g(h_j)}{\partial h_j} \qquad \text{...(19)}$$

$$h_j = \sum w_{ji} x_i - \theta_j \qquad \text{...(20)}$$

Donde h_k y h_j son los potenciales post-sinápticos (regla de propagación, **ver modelo de una neurona artificial**). Las actualizaciones de los umbrales se realiza haciendo uso de las operaciones anteriores.

Con estas expresiones está implícito el concepto de propagación hacia atrás de los errores (**RPE**). Primero se calcula la expresión Δ_k, que se denominará señal de error, por ser proporcional a la diferencia de la salida actual de la red y el objetivo, con el que se calcula la actualización δw_{kj} de los pesos de la capa de salida. A continuación se propaga la señal hacia atrás, proporcionando así las señales de error Δ_j correspondientes a las sinapsis de la capa oculta; con éstas se calcula la actualización δw_{ji}. El algoritmo puede extenderse fácilmente a arquitecturas con más de una capa siguiendo el mismo esquema.

1.8 Aceleración del RPE.

Debe considerarse que el algoritmo **RPE** no garantiza alcanzar el mínimo global de la función de error, tan sólo en un mínimo local, por lo que el proceso de aprendizaje puede estancarse.

Ahora bien, para resolver ciertos inconvenientes del procedimiento **RPE**, como su lenta convergencia, se propusieron algunas variantes. La primera, la propusieron *Rumelhart* en 1986, quien con un grupo de investigación en **RNA**, popularizaron el modelo **RPE** [Rumelhart D.E., 1986], incluyendo en el algoritmo un término de inercia (*momentum*), consistente en añadir al cálculo de la variación de los pesos, un término adicional proporcional al incremento de la iteración anterior, que proporciona una cierta inercia al entrenamiento:

$$\boldsymbol{\delta w_{kj}(t+1) = -\varepsilon \left.\frac{\partial E}{\partial w_{kj}}\right|_t + \alpha \cdot \delta w_{kj}(t-1)} \qquad \textit{...(21)}$$

$$\boldsymbol{\delta w_{ji}(t+1) = -\varepsilon \left.\frac{\partial E}{\partial w_{ji}}\right|_t + \alpha \cdot \delta w_{ji}(t-1)} \qquad \textit{...(22)}$$

Con $\boldsymbol{\alpha}$ un parámetro entre $[0,1]$, que se suele tomar próximo a 1 ($\boldsymbol{\alpha \approx 0.9}$).

Es ésta, una manera de aumentar el ritmo de aprendizaje efectivo en determinadas direcciones.

La razón por la que se usan funciones sigmoides como funciones de transferencia o de activación para el **RPE**, es que este algoritmo busca minimizar la función de error para los pesos mediante el cálculo del gradiente en dirección descendente. Como se debe calcular el gradiente de la función de error en cada iteración, se tendrá que garantizar su continuidad y diferenciabilidad en todo momento.

La derivada de la sigmoide es muy importante para el funcionamiento del algoritmo, ya que consiste en calcular el gradiente de la función error compuesta con la sigmoide para moverse en la dirección en la que éste disminuya. Este estilo de minimizar la función se podría pensar como un proceso físico equivalente al de poner una canica en la superficie formada por la función de error y dejarla rodar sobre ella hasta encontrar el mínimo [Rojas Raúl, 1996].

Esto siempre lleva en una dirección más adecuada, porque la derivada de la sigmoide siempre es positiva, marcándonos la dirección adecuada para movernos; con la desventaja de que algunas veces el gradiente es muy grande y en otros muy pequeños, haciendo que pueda resultar un poco complicado seguir la dirección óptima.

La red **RPE** puede emplearse en tareas de ajuste funcional (**'predicción'**, **'series temporales'**, **'modelado'**) y también para clasificación. Ésta es la red neuronal por excelencia, empleada en más del setenta por ciento de los casos.

1.9 Capacidad de generalización de la red.

La forma habitual de entender los sistemas supervisados es dividir los ejemplos disponibles en dos conjuntos: entrenamiento y generalización, normalmente en una relación $\mathbf{2/3}$ a $\mathbf{1/3}$. Usualmente, de todo un conjunto de entrenamiento se emplea aproximadamente un $\mathbf{80}\%$ de los patrones para entrenar, reservándose un $\mathbf{20}\%$ como *conjunto prueba* [Haykin S., 1999].

Con el primer conjunto de datos se entrena la red, esto es, se ajustan las conexiones de la red o pesos, y con el segundo se comprueba su capacidad de generalización. Se entiende por generalización a la capacidad que tiene la red de dar una respuesta correcta ante patrones que no han sido empleados en su entrenamiento.

Hay que considerar que una de las principales características de las redes neuronales, que las hacen especiales frente a otros métodos, es su capacidad de generalización, es decir, ante entradas desconocidas son capaces de dar salidas aproximadas a las deseadas.

En un proceso de entrenamiento, se debe considerar, por una parte, un **error de aprendizaje**, que se suele calcular con el error cuadrático medio de los resultados proporcionados por la red. Para una red suficientemente grande, puede reducirse tanto como se quiera sólo con llevar a cabo más iteraciones (épocas).

Por otra parte, existe un **error de generalización** (medida de error en *test*), que se puede medir empleando un conjunto representativo, diferentes a los utilizados en el entrenamiento. De esta manera, se puede entrenar una red neuronal haciendo uso de un

conjunto de aprendizaje, y comprobar su eficiencia real, o error de generalización, mediante un conjunto de prueba.

Si representamos a la vez lo errores de aprendizaje y de *test* durante el transcurso del entrenamiento **(ver figura 1.8)**, tienden a disminuir de igual forma, nombrado error de generalización, esto es, hasta que el error de *test* empiece a incrementar, por lo que se debe encontrar el punto óptimo o error mínimo del *test,* denominada **validación cruzada** (*cross validation*), este procedimiento es ampliamente utilizado para la red neuronal supervisada, tal es el caso para el **PMC**.

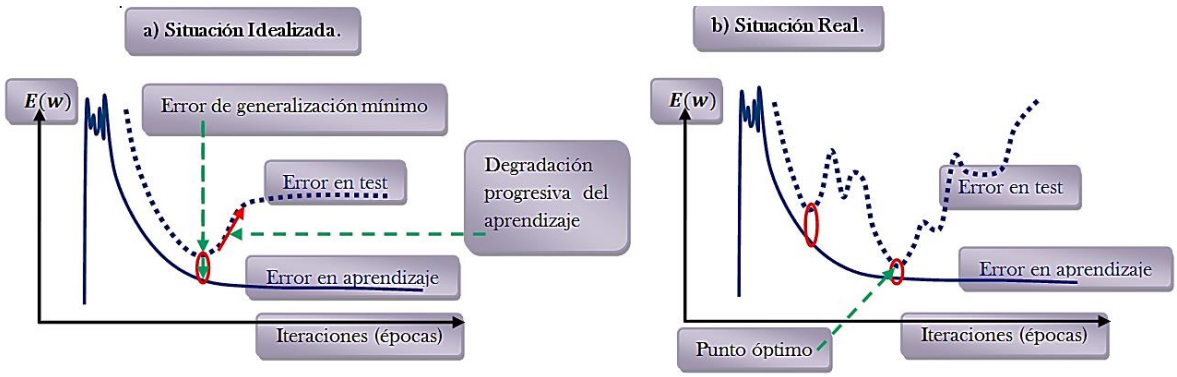


Figura 1.8 Evolución del error de aprendizaje y del error de generalización [Prechelt L., 1998].

Para una situación idealizada (Figura 1.8a), tras una fase inicial, en la que pueden aparecer oscilaciones en el valor del error, el de aprendizaje tiende a disminuir monótonamente, mientras que el de generalización al principio la red se adapta progresivamente al conjunto de aprendizaje, adaptándose al problema y mejorando la generalización, pero a partir de cierto punto comienza a incrementarse.

Para una situación más realista (Figura 1.8b), pueden presentarse varios mínimos para el conjunto *test*, debiendo detener el aprendizaje en el punto óptimo de error mínimo de generalización, y no quedarnos en el primer mínimo que aparezca en el error de prueba.

La **validación cruzada** es utilizada por el denominado **sobre-ajuste** o **sobre-aprendizaje** (*overfitting* u *overtraining*), dado que el sistema se ajusta demasiado, aprendiendo demasiado, incluso el ruido presente en los ejemplos empleados, por lo que crece el error de generalización.

En la aplicación de pronóstico de series de tiempo con **RNA**, la optimización jugará un papel importante, debido a que el pronóstico tiene importancia cuando se presenta en tiempo real.

1.10 Estructura y funcionamiento de una RNA.

Teniendo como ejemplo una arquitectura unidireccional de tres capas que se muestra en la siguiente figura, cada círculo de color es una neurona o un nodo, por lo que una red neuronal está estructurada por más de una sola neurona.

Cada capa puede contener $\boldsymbol{N}$ neuronas, así como la arquitectura de una red neuronal puede contener $\boldsymbol{M}$ capas, por lo que se puede realizar diferentes arquitecturas de una red neuronal artificial, dependiendo lo que el usuario deseé implementar.

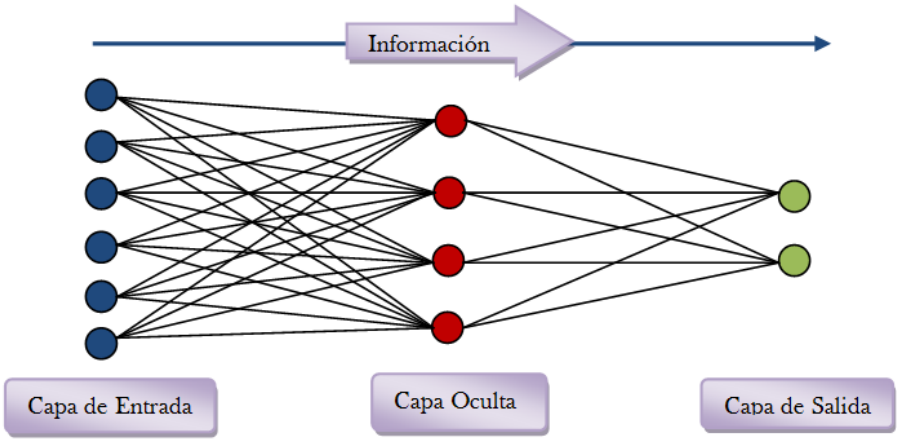


*Figura 1.9 Arquitectura unidireccional de tres capas de una **RNA**.*

Dado este ejemplo, se muestra una arquitectura de tres capas, una primera capa conocida como 'capa de entrada', en la cual es la información que se ha de procesar, filtrar, o estudiar según su objetivo establecido; para este ejemplo, se tiene 6 nodos de entrada (círculos azules), éstos se conectan con una segunda capa conocida como 'capa oculta' (la función de la capa oculta es incluir un comportamiento **no lineal**, **ver el perceptrón multicapa**) teniendo 4 nodos ocultos (círculos rojos) y dando como resultado una *'capa de salida'* con 2 nodos de salida (círculos verdes).

Pueden existir $\boldsymbol{K}$ capas ocultas con las que se deseé trabajar, por lo que se tendrá una capa de entrada y una capa de salida, teniendo que el número de capas ocultas se determina como $\boldsymbol{K = M - 2}$. En la literatura, en algunos casos, omiten la primera capa, esto es que no la consideran como una capa neuronal, ya que es la información sin procesar y es la que se ingresa al modelo. En este trabajo si se le considera como capa (capa de entrada), simplemente se aclara este punto para hacerle mención al lector cuando indague en alguna bibliografía.

Siguiendo con el ejemplo dado, la información fluye o se procesa de una capa a otra, esto es, que la capa de entrada se conecta con la capa oculta, y la capa oculta se conecta con la capa de salida, he de ahí que este ejemplo es de una arquitectura unidireccional de tres capas.

Las neuronas de una capa se han de conectar con las neuronas de la siguiente capa, conectando todos los nodos de una capa a otro nodo y así hasta que todos los nodos estén conectados de una capa a otra.

Redondeando esto último mencionado, para el ejemplo de la figura, la capa de entrada de 6 nodos (círculos azules) se conecta con la capa oculta de 4 nodos (círculos rojos), lo que se está realizando es que los 6 nodos de entrada se conectan con el primer nodo oculto **(ver figura 1.10),** estos mismos 6 nodos de entrada se conectan con el segundo nodo oculto, y así sucesivamente hasta que todos estén conectados con todos los nodos ocultos. Lo mismo ocurre con la conexión de la capa oculta a la capa de salida, los 4 nodos de la capa oculta se conectan con los 2 nodos de la capa de salida (círculos verdes), teniendo que los 4 nodos ocultos se conectan con el primer nodo de salida y estos mismos 4 nodos ocultos se conectan don el segundo nodo de salida.

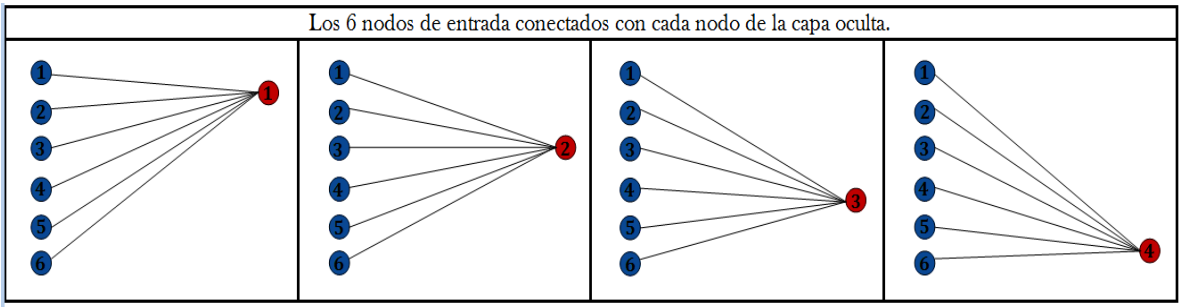


Figura 1.10 Conexión de los nodos de la capa de entrada a los nodos de la capa oculta.

Las conexiones mencionadas, se han de realizar por medio de ciertas operaciones numéricas por medio de valores, evaluados por los llamados 'pesos', éstos se ajustan durante la fase llamada de 'aprendizaje' o 'entrenamiento' obteniendo una Red Neuronal Artificial (**RNA**).

El funcionamiento de una **RNA** se da de la siguiente manera. Teniendo la información (datos a procesar) para cada vector de entrada, éste es introducido en la red copiando cada valor de dicho vector a la capa de entrada. Una vez recibida la totalidad de sus entradas, las procesa y genera una salida que es propagada a través de las conexiones, llegando ahora, como entrada a la siguiente capa de destino. Finalmente, cuando la ahora entrada ha sido completamente propagada por toda la red, se producirá un vector salida, cuyos componentes son cada uno de los valores de salida de la llamada capa de salida.

Ejemplificando sobre lo descrito del funcionamiento de una **RNA**, dado el ejemplo de una arquitectura unidireccional de tres capas ya mencionado, el esquema de funcionamiento puede describirse mediante la ecuación:

$$\vec{s} = F(w_2 \cdot F[w_1 \cdot \vec{x}]) \qquad \text{...(23)}$$

Siendo, w_1 y w_2 los pesos de la capa oculta y capa de salida, respectivamente; F se conoce como la función de activación o de transferencia (esta función de activación se presenta en la parte de la estructura de la neurona artificial); $\vec{x}$ es el vector de entrada a la red, y $\vec{s}$ es el vector de salida que la red produce.

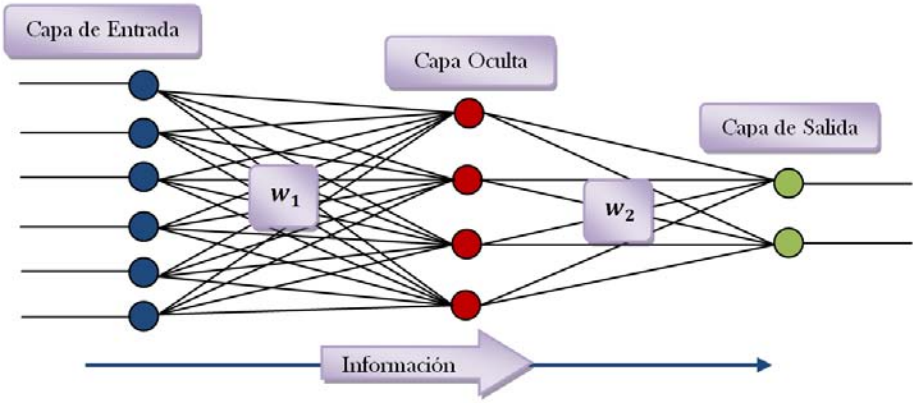


*Figura 1.11 Funcionamiento de una **RNA**.*

$\boldsymbol{w_1}$ y $\boldsymbol{w_2}$ son matrices de conexiones entre las capas de la red, por lo tanto, la expresión $(\boldsymbol{w} \cdot \vec{\boldsymbol{x}})$ es una multiplicación de matrices (vectores), de modo que la expresión de la suma ponderada de pesos y valores $\left(\sum_i \boldsymbol{w_{ji}} \cdot \boldsymbol{x_i}\right)$ es una multiplicación de matrices, esta última expresión se muestra al hablar sobre la neurona artificial, así como en la operación del perceptrón multicapa; más adelante se describirá a una manera amplia como se tomó para la realización de estas operaciones matriciales y con ello llevarla a la programación, similar a este ejemplo.

Supóngase que la función de activación $\boldsymbol{F}$ que se utiliza es del tipo lineal $\boldsymbol{F(x) = c \cdot x}$, si se sustituye en la ecuación, se tiene que:

$$\vec{\boldsymbol{s}} = \boldsymbol{c} \cdot \boldsymbol{w_2} \cdot (\boldsymbol{c} \cdot \boldsymbol{w_1} \cdot \vec{\boldsymbol{x}}) \qquad \textit{...(24)}$$

Teniendo como $\boldsymbol{c} = \mathbf{1}$, entonces:

$$\vec{\boldsymbol{s}} = \boldsymbol{w_2} \cdot (\boldsymbol{w_1} \cdot \vec{\boldsymbol{x}}) \qquad \textit{...(25)}$$

Ilustrando más este ejemplo, sea:

$$w_1 = \begin{pmatrix} 3 & 2 & 3 & 1 & 3 & 5 \\ 4 & 1 & 2 & 2 & 2 & 1 \\ 1 & 2 & 1 & 5 & 1 & 1 \\ 6 & 1 & 1 & 3 & 1 & 1 \end{pmatrix}$$

$$w_2 = \begin{pmatrix} 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 2 \end{pmatrix}$$

$$\vec{x} = \begin{pmatrix} -3 \\ 1 \\ 4 \\ -2 \\ 3 \\ -1 \end{pmatrix}$$

Realizando la primera conexión entre la capa de entrada y la oculta, se tiene que:

$$\vec{y} = w_1 \cdot \vec{x} = \begin{pmatrix} 3 & 2 & 3 & 1 & 3 & 5 \\ 4 & 1 & 2 & 2 & 2 & 1 \\ 1 & 2 & 1 & 5 & 1 & 1 \\ 6 & 1 & 1 & 3 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} -3 \\ 1 \\ 4 \\ -2 \\ 3 \\ -1 \end{pmatrix} = \begin{pmatrix} 7 \\ -2 \\ -5 \\ -17 \end{pmatrix}$$

Obteniendo un vector salida de esta primera conexión y actuando ahora como entrada a la segunda conexión entre la capa oculta y la de salida, dando como resultado:

$$\vec{s} = w_2 \cdot \vec{y} = \begin{pmatrix} 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 7 \\ -2 \\ -5 \\ -17 \end{pmatrix} = \begin{pmatrix} -44 \\ -22 \end{pmatrix}$$

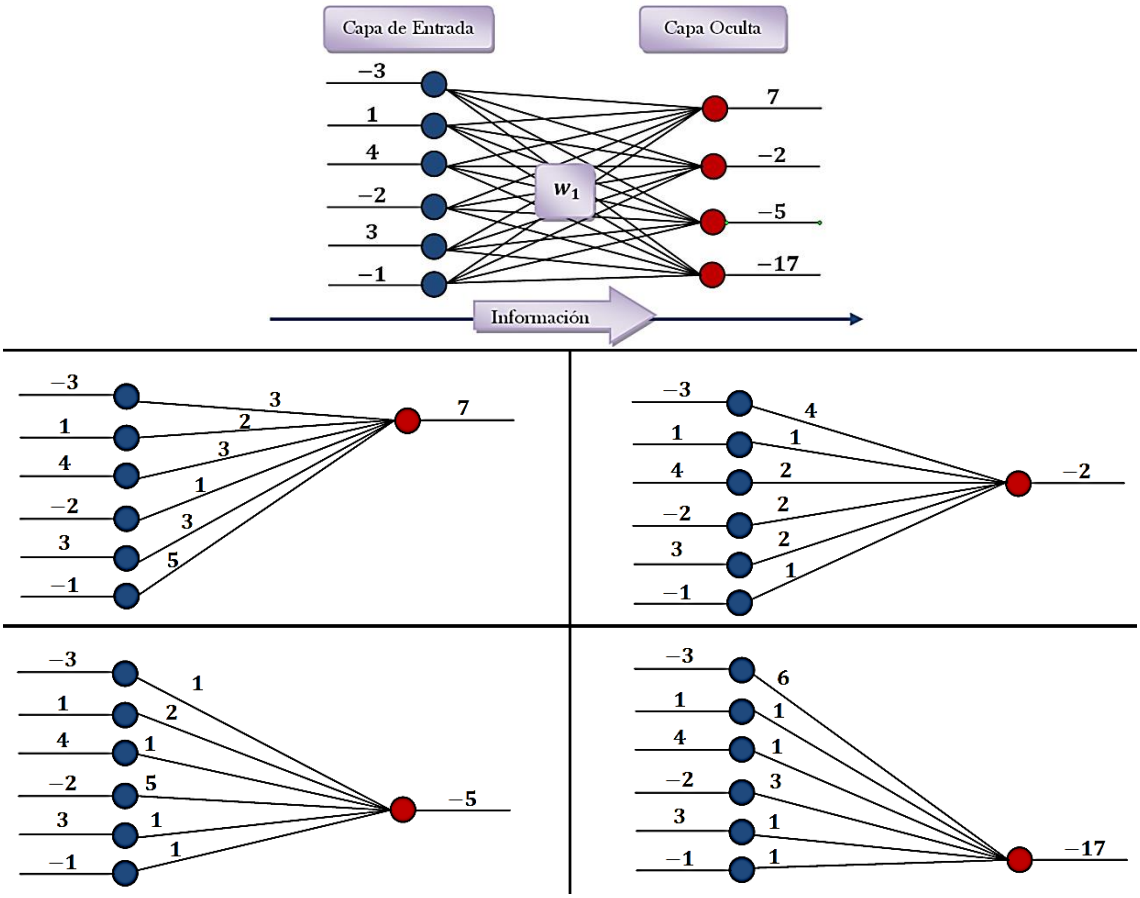


Figura 1.12 Primera conexión entre la capa de entrada y la oculta

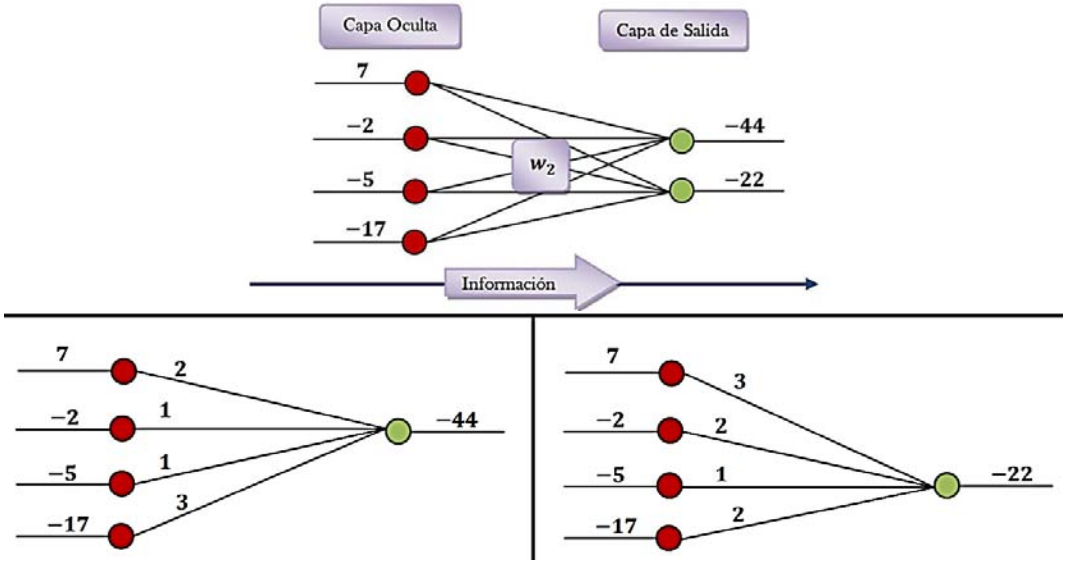


Figura 1.13 Segunda conexión entre la capa oculta y de salida.

De la ecuación 24 se puede simplificar, obteniendo:

$$\vec{s} = c^2 \cdot w_2 \cdot w_1 \cdot \vec{x} \qquad \textit{...(26)}$$

Lo cual equivaldría a una red con una sola capa de conexiones $\boldsymbol{W}$, donde:

$$W = c^2 \cdot w_2 \cdot w_1 \qquad \textit{...(27)}$$

$$\vec{s} = W \cdot \vec{x} \qquad \textit{...(28)}$$

Por lo que la misma salida se obtendría con una sola capa de conexiones:

$$W = 1^2 \cdot \begin{pmatrix} 2 & 1 & 1 & 3 \\ 3 & 2 & 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} 3 & 2 & 3 & 1 & 3 & 5 \\ 4 & 1 & 2 & 2 & 2 & 1 \\ 1 & 2 & 1 & 5 & 1 & 1 \\ 6 & 1 & 1 & 3 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 29 & 10 & 12 & 18 & 12 & 15 \\ 30 & 12 & 16 & 18 & 16 & 20 \end{pmatrix}$$

$$\vec{s} = W \cdot \vec{x} = \begin{pmatrix} 29 & 10 & 12 & 18 & 12 & 15 \\ 30 & 12 & 16 & 18 & 16 & 20 \end{pmatrix} \cdot \begin{pmatrix} -3 \\ 1 \\ 4 \\ -2 \\ 3 \\ -1 \end{pmatrix} = \begin{pmatrix} -44 \\ -22 \end{pmatrix}$$

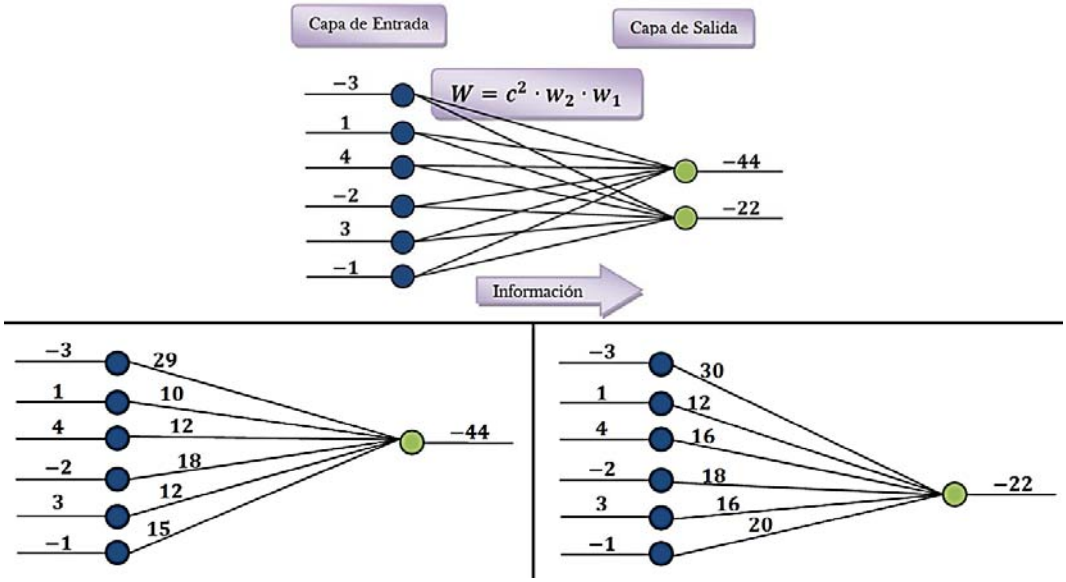


Figura 1.14 Red equivalente a la de tres tapas, esto por la función de activación lineal.

Obteniendo los mismos resultados, denota que la función de activación es lineal y que al introducir más capas (capas ocultas) en la red es irrelevante; existirá siempre una red de una sola capa de entrada y de salida equivalente a cualquier otra con un número de capas ocultas arbitrario, y su cálculo es trivial. Por lo tanto, el incluir funciones de activación lineales hará que la red pierda gran parte de su potencial. Todas las redes cuentan con funciones de activación **no lineal**, que hacen que el potencial de la red sea prominente para solucionar problema de forma genérica.

Ampliando lo ya mencionado hasta aquí, se describirán tanto los vectores (matrices) de entrada y de salida, como las matrices de pesos y sus correspondientes operaciones, siguiendo las expresiones del perceptrón multicapa (**PMC**), y con ello llevadas a la programación.

Siendo la expresión matemática para el **PMC** de una sola capa oculta ($K = 1$, $K\ capas\ ocultas$, **ver estructura de una red neuronal artificial**)

$$z_k^{\mu} = f\left(\sum_j w_{kj}\left[y_j^{\mu}\right] - \theta_k\right) = f\left(\sum_j w_{kj} f\left[\sum_i w_{ji}\, x_i^{\mu} - \theta_j\right] - \theta_k\right) \quad \textit{...(29)}$$

Empezando con los datos de entrada, que van siendo los datos históricos, series de tiempo, ejemplos, etc., cuyo objetivo es ingresar al modelo de la red neuronal y ésta hará un cierto entrenamiento (ajuste de pesos) del cual aprenderá sobre de estos datos, por lo que también se conocen a éstos como datos a entrenar, expresado como:

$$x_i^{\mu} = \begin{pmatrix} x_1^1 & x_1^2 & \cdots & x_1^m \\ x_2^1 & x_2^2 & \cdots & x_2^m \\ x_3^1 & x_3^2 & \dots & x_3^m \\ \vdots & \vdots & \ddots & \vdots \\ x_n^1 & x_n^2 & \dots & x_n^m \end{pmatrix} = X = \{x_{i\mu}\}$$

$$i = 1,2,3,\dots,n$$

$$\mu = 1,2,\dots,m$$

Donde $\boldsymbol{i}$ es el número de entradas y $\boldsymbol{\mu}$ vendría siendo el número de patrones; teniendo entonces definida la capa de entrada como $\boldsymbol{x_i^{\mu}}$.

Los pesos de la conexión entre la capa de entrada y la oculta $\boldsymbol{w_{ji}}$, estará definida como:

$$w_{ji} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1n} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{l1} & w_{l2} & w_{l3} & \dots & w_{ln} \end{pmatrix} = W_1 = \{w_{ji}\}$$

$$j = 1,2,\dots,l$$

$$i = 1,2,3,\dots,n$$

Donde $\boldsymbol{j}$ es el número de neuronas en la capa oculta, por lo que el tamaño de la matriz $\boldsymbol{w_{ji}}$ lo conforma el número de entradas $\boldsymbol{i}$ por el número de neuronas ocultas $\boldsymbol{j}$.

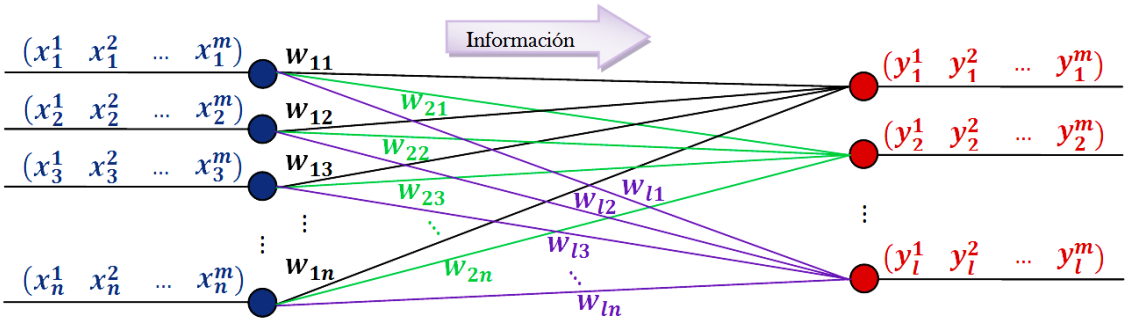


Figura 1.15 Conexión entre la capa de entrada y la capa de oculta. Entradas $\boldsymbol{x_i^{\mu}}$, pesos $\boldsymbol{w_{ji}}$ y salidas $\boldsymbol{y_i^{\mu}}$.

Siguiendo con la expresión matemática del **PMC** se tiene la siguiente suma ponderada, la cual se puede representar como una multiplicación de matrices, esto se conoce como regla de propagación proporcionando el valor del potencial post-sináptico **(ver modelo de una neurona artificial)**:

$$\sigma_j^{\mu} = \sum_i w_{ji}\, x_i^{\mu} = W_1 \cdot X = \{w_{ji}\} \cdot \{x_{i\mu}\} = \{\sigma_{j\mu}\} \qquad \textit{...(30)}$$

En varios casos, para esta regla de propagación, se presenta lo que se conoce como umbral $\boldsymbol{\theta_j}$ **(ver modelo de una neurona artificial)**, siendo este un vector y cuya operación en esta regla de propagación es una suma (resta) matricial:

$$h_j^{\mu} = \sum_i w_{ji}\, x_i^{\mu} - \theta_j = \sigma_j^{\mu} - \theta_j \qquad \textit{...(31)}$$

$$h_j^{\mu} = \sigma_j^{\mu} - \theta_j = \begin{pmatrix} \sigma_1^1 & \sigma_1^2 & \cdots & \sigma_1^m \\ \sigma_2^1 & \sigma_2^2 & \cdots & \sigma_2^m \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_l^1 & \sigma_l^1 & \cdots & \sigma_l^m \end{pmatrix} - \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_l \end{pmatrix} = \begin{pmatrix} \sigma_1^1-\theta_1 & \sigma_1^2-\theta_1 & \cdots & \sigma_1^m-\theta_1 \\ \sigma_2^1-\theta_2 & \sigma_2^2-\theta_2 & \cdots & \sigma_2^m-\theta_2 \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_l^1-\theta_l & \sigma_l^1-\theta_l & \cdots & \sigma_l^m-\theta_l \end{pmatrix}$$

$$h_j^{\mu} = \begin{pmatrix} h_1^1 & h_1^2 & \cdots & h_1^m \\ h_2^1 & h_2^2 & \cdots & h_2^m \\ \vdots & \vdots & \ddots & \vdots \\ h_l^1 & h_l^1 & \cdots & h_l^m \end{pmatrix} = H = \{h_{j\mu}\}$$

Para obtener finalmente la salida $\boldsymbol{y_j^{\mu}}$ de la conexión entre capas de entrada y oculta **(ver figura 1.15)**, se evalúa el valor del potencial post-sináptico h_j^{μ}, en una función que se le ha de llamar función de activación o de transferencia $\boldsymbol{f}(\cdot)$ **(ver modelo de una neurona artificial)**.

$$y_j^{\mu} = f\left(\sum_i w_{ji}\, x_i^{\mu} - \theta_j\right) = f\left(h_j^{\mu}\right) \qquad \textit{...(32)}$$

$$f\left(h_j^{\mu}\right) = \begin{pmatrix} f(h_1^1) & f(h_1^2) & \cdots & f(h_1^m) \\ f(h_2^1) & f(h_2^2) & \cdots & f(h_2^m) \\ \vdots & \vdots & \ddots & \vdots \\ f(h_l^1) & f(h_l^1) & \cdots & f(h_l^m) \end{pmatrix}$$

$$y_j^{\mu} = Y = \{Y_{j\mu}\} = \begin{pmatrix} y_1^1 & y_1^2 & \cdots & y_1^m \\ y_2^1 & y_2^2 & \cdots & y_2^m \\ \vdots & \vdots & \ddots & \vdots \\ y_l^1 & y_l^1 & \cdots & y_l^m \end{pmatrix}$$

$$j = 1,2,\ldots,l$$

$$\mu = 1,2,\ldots,m$$

Ahora, esta y_j^{μ} será la entrada para la conexión entre la capa oculta y la capa de salida

$$z_k^{\mu} = f\left(\sum_j w_{kj}\left[y_j^{\mu}\right] - \theta_k\right) \qquad \textit{...(33)}$$

Los pesos de la conexión entre la capa de entrada y la oculta $\boldsymbol{w_{kj}}$, estará definida como:

$$w_{kj} = \begin{pmatrix} w_{11} & w_{12} & \ldots & w_{1l} \\ w_{21} & w_{22} & \ldots & w_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ w_{r1} & w_{r2} & \ldots & w_{rl} \end{pmatrix} = W_2 = \{w_{kj}\}$$

$$k = 1,2,\ldots,r$$

$$j = 1,2,\ldots,l$$

Siendo $\boldsymbol{k}$ el número de neuronas en la capa de salida o salidas de la red (número de salidas), recordando que $\boldsymbol{j}$ es el número de neuronas en la capa oculta, así que el tamaño

de la matriz $\boldsymbol{w_{kj}}$ lo forma el número de neuronas de la capa de salida $\boldsymbol{k}$ por el número de neuronas ocultas $\boldsymbol{j}$.

De la misma forma que se realiza la conexión entre la capa de entrada y oculta, se sigue los mismos pasos para la conexión de la capa oculta y de salida, teniendo entonces una regla de propagación proporcionando el valor del potencial post-sináptico:

$$\sigma_k^{\mu} = \sum_j w_{kj} y_j^{\mu} = W_2 \cdot Y = \{w_{kj}\} \cdot \{Y_{j\mu}\} \qquad \textit{...(34)}$$

$$h_k^{\mu} = \sigma_k^{\mu} - \theta_k = H = \{h_{k\mu}\} \qquad \textit{...(35)}$$

Evaluando esto último en una función de activación o de transferencia $\boldsymbol{f}(\cdot)$, obteniendo la salida de esta conexión:

$$z_k^{\mu} = f(h_k^{\mu}) \qquad \textit{...(36)}$$

$$z_k^{\mu} = \begin{pmatrix} z_1^1 & z_1^2 & \dots & z_1^m \\ z_2^1 & z_2^2 & \dots & z_2^m \\ \vdots & \vdots & \ddots & \vdots \\ z_r^1 & z_r^2 & \dots & z_r^m \end{pmatrix} = Z = \{z_{k\mu}\}$$

$$k = 1,2,\dots,r$$

$$\mu = 1,2,\dots,m$$

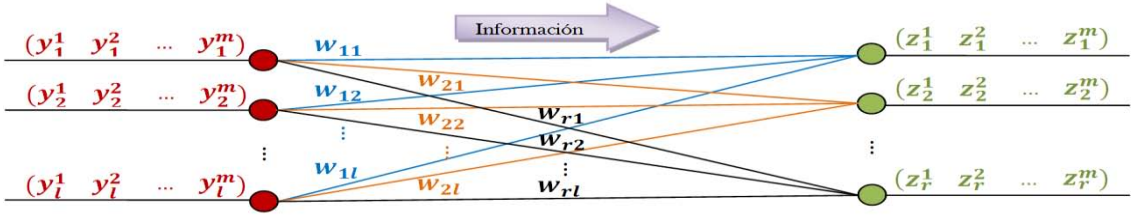


Figura 1.16 Conexión entre la capa oculta y la capa de salida.
Entradas $\boldsymbol{y_i^{\mu}}$, pesos $\boldsymbol{w_{kj}}$ y salidas $\boldsymbol{z_k^{\mu}}$.

1.11 Función XOR.

La función **XOR**, también conocida como **OR-exclusivo**, es una función lógica que representa la función de la desigualdad, es decir, la salida es 'verdadera' si las entradas no son iguales, de otro modo el resultado es 'falso' **(ver tabla 1.3)**. Esta función lógica, es lo que se conoce como una función booleana, por lo que suman, multiplican, niegan o afirman, incluyen o excluyen según sus propiedades lógicas.

Expresión 1	Falso	Falso	Verdadera	Verdadera
Expresión 2	Falso	Verdadera	Falso	Verdadera
Resultado	Falso	Verdadera	Verdadera	Falso

Tabla 1.3 Función booleana.

Bajo la estructura del **PMC** y lo ilustrativo en la realización de las operaciones, se da un ejemplo conocido como **XOR**, siguiendo este ejemplo de [Isasi Viñuela P. *et al.*, 2004, p.50]. Se toma este ejemplo para corroborar que las operaciones concuerden con los resultados de la bibliografía y con lo programado, examinando las operaciones en las capas oculta y de salida, esta verificación no realiza ningún tipo de entrenamiento. Posteriormente se retoma este ejemplo [Isasi Viñuela P. *et al.*, 2004, p.50], ahora ya aplicando un algoritmo de entrenamiento, que es mediante el **RPE** **(ver mecanismos de entrenamiento)**, haciendo una pequeña modificación en la estructura de la red, la cual beneficia en los resultados.

Cálculo de las activaciones para un **PMC** con dos neuronas de entrada, dos ocultas y una salida **(ver figura 1.17)**. Considerando los patrones que definen la función lógica **XOR**:

	Patron_1	Patron_2	Patron_3	Patron_4
Entrada_1	0	0	1	1
Entrada_2	0	1	0	1
Salida	0	1	1	0

*Tabla 1.4 Función lógica **XOR**.*

Capa de entrada, donde $\mu = 1,2,3,4$

$$x_i^{\mu} = \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

Si los pesos y umbrales de la red toman los valores:

Para la capa oculta:

$$w_{ji} = \begin{pmatrix} 5.191129 & 5.473012 \\ 2.758669 & 2.769596 \end{pmatrix}$$

$$b_j = \begin{pmatrix} 1.90289 \\ 4.127002 \end{pmatrix}$$

Capa de salida

$$w_{kj} = (5.839709 \quad -6.186834)$$

$$b_k = (2.570539)$$

Siendo estos parámetros los óptimos para aproximar la función **XOR**, se introducen en la red, cuya arquitectura está formada de dos neuronas de entrada ($i = 1,2$), dos neuronas

en la capa oculta ($j = 1{,}2$) y en la capa de salida una sola neurona ($k = 1$) **(ver figura 1.17),** utilizando como función de activación una función sigmoidea $f(h) = \frac{1}{1+e^{-h}}$ **(ver modelo de una neurona artificial).**

Las salidas (resultados) de la red para los patrones de la función **XOR** toma los valores:

Capa de entrada		Capa oculta		Capa de salida
x_1^μ	x_2^μ	y_1^μ	y_2^μ	z_1^μ
0	0	0.129781	0.015875	0.128883
0	1	0.972618	0.204662	0.863310
1	0	0.964023	0.202889	0.858615
1	1	0.999843	0.802384	0.154993

*Tabla 1.5 Entradas y salidas de la **RNA** para la función **XOR**.*

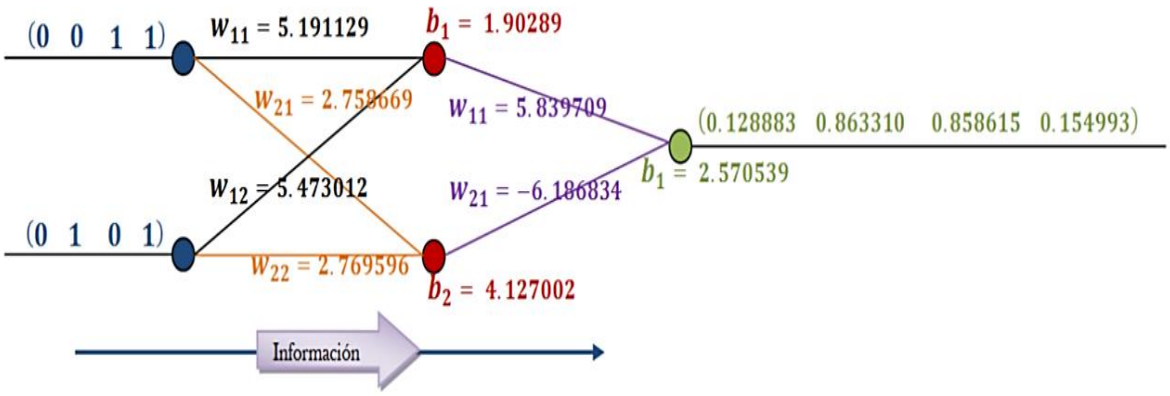


*Figura 1.17 Arquitectura para la función **XOR**.*

Dados los parámetros y resultados mostrados, se compara con lo que arroja nuestro programa, obteniendo lo siguiente:

```
[Isaisi Viñuela Pedro et al., 2004]
Salidas de la CO =
    0.1298    0.0159
    0.9726    0.2047
    0.9640    0.2029
    0.9998    0.8024

Salida =
    0.1289
    0.8633
    0.8586
    0.1550

Error =
    0.0397
```

Resultado 1.1

Se observa que los resultados son idénticos, solo que en el diez milésimo valor es redondeado. El error que se muestra es el error entre lo deseado, que es $\boldsymbol{t} = (0 \quad 1 \quad 1 \quad 0)$ y la salida de la red $\boldsymbol{z_1^{\mu}}$, este error se calcula por medio del error cuadrático medio **(ver mecanismos de entrenamiento)**, teniendo que estas salidas son una aproximación al objetivo (*target*), por lo que si se redondea estos valores se obtiene lo que se desea, siendo que 0.1289 y 0.1550 están cercanos al 0, los valores 0.8633 y 0.8586 están cercanos al 1.

Esto último, sobre el error y redondear los resultados obtenidos, dependiendo el usuario, que tan minucioso sea con lo que se obtenga de la **RNA**, dado que los resultados pueden que sean exactos o que maneje un cierto error muy cercano a cero, esto depende de lo que se deseé obtener y el objetivo en la implementación de una **RNA**.

Para este ejemplo de la función **XOR**, se puede decir que, con los resultados dados, es aceptable y que la **RNA** aproxima dicha función. Más sin en cambio, podemos ser

rigurosos con estos resultados por lo que a continuación se muestra este mismo ejemplo, pero implementando un algoritmo de aprendizaje y haciendo una modificación en la estructura de la red.

Poniendo a prueba el funcionamiento de este tipo de red multicapa y el aprendizaje por retropropagación de errores (**RPE**). Se realizaron varias pruebas agregando más de dos neuronas en la capa oculta, dejando la misma arquitectura de tres capas: una de entrada, una oculta y una de salida; dentro de estas capas se tienen las mismas dos entradas y una sola neurona de salida, con el mismo número de patrones y cuya función de activación seguirá siendo la función sigmoidea $\boldsymbol{f(h) = \frac{1}{1+e^{-h}}}$.

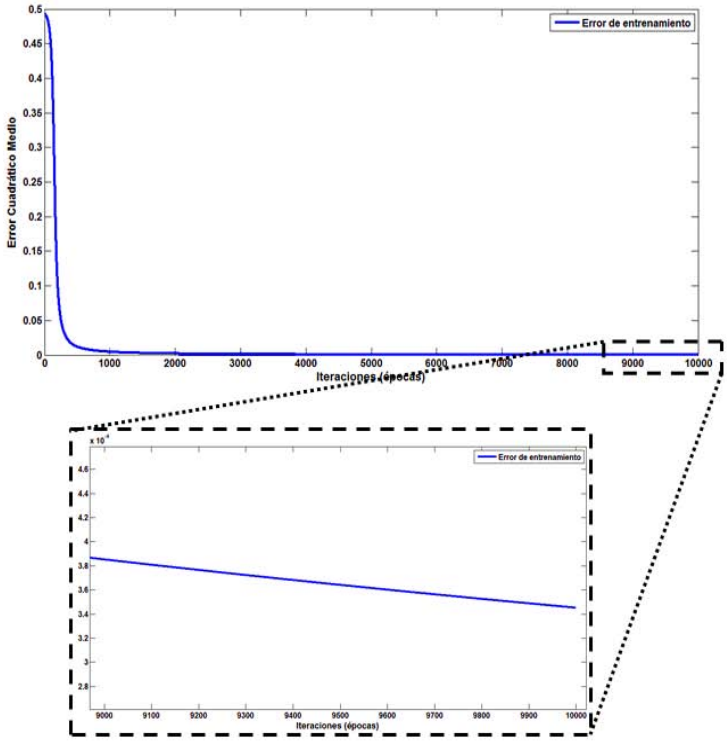
Observando la evolución de los errores obtenidos a partir de los gradientes para el ajuste de pesos como para los umbrales, se llega a lo siguiente, se dejan constantes los umbrales igual a menos uno, sin que esta sufra un ajuste. Para obtener resultados minuciosos se requiere desde tres a no más de seis neuronas en la capa oculta, con ello se obtiene un error menor al mostrado con los parámetros que se dan en el ejemplo citado.

Se menciona no más a seis neuronas en la capa oculta puesto que si aumenta su número, no se garantiza la disminución del error o bien no tendría un mayor efecto en su descenso.

Al dejar constantes los umbrales, la red se enfocará en los ajustes de los pesos y así encontrar una función óptima en un menor tiempo de proceso, además de evitar oscilaciones en el error, ya que el entrenamiento va ajustando tanto los pesos como los umbrales y éstos pueden diferir en los resultados causando perturbaciones en el error.

Se muestra a continuación el resultado, implementando tres neuronas en la capa oculta **(ver figura 1.18)**, obteniendo un error entre lo que da la red y lo deseado de 0.00034518. Se realizan diez mil iteraciones, donde en cada iteración hay un ajuste sobre los pesos y al observar el comportamiento del error en las últimas iteraciones, aún se da un descenso de éste aunque no sea tan apreciable, esto es porque el error está en un valor dado de 10^{-4}.

```
Error mínimo de entrenamiento:
   3.4518e-04

Salidas deseadas
     0     1     1     0

Salidas obtenidas
    0.0171    0.9866    0.9866    0.0064

PESOS Y UMBRALES OPTIMOS:
*Capa oculta
Wji =
    4.1563    4.1747
    7.1963   -5.0571
   -5.0715    7.2309

bj =
    -1
    -1
    -1

*Capa de salida
Wkj =
   14.0200  -10.4648  -10.4638

bk =
    -1

Elapsed time is 8.701548 seconds.
```



Resultado 1.2

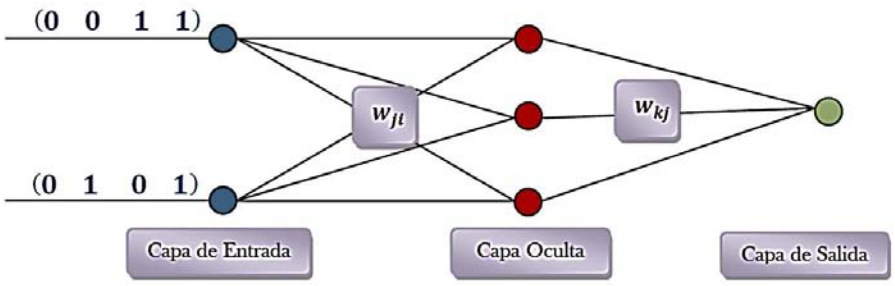


*Figura 1.18 Arquitectura para la función **XOR** con 3 neuronas ocultas.*

Las salidas obtenidas de la red están más cercanas al 0 y 1. $Z_k^{\mu} = (0.0171 \quad 0.9866 \quad 0.9866 \quad 0.0064)$, siendo $k = 1$ y $\mu = 4$. También, se muestra en los resultados los pesos y umbrales óptimos para reproducir esta función de aproximación.

Realizando otro experimento, del cual se implementan 6 neuronas en la capa oculta **(ver figura 1.19)**, se alcanzó un error de entrenamiento de $1.9964e-04$ (menor al anterior resultado).

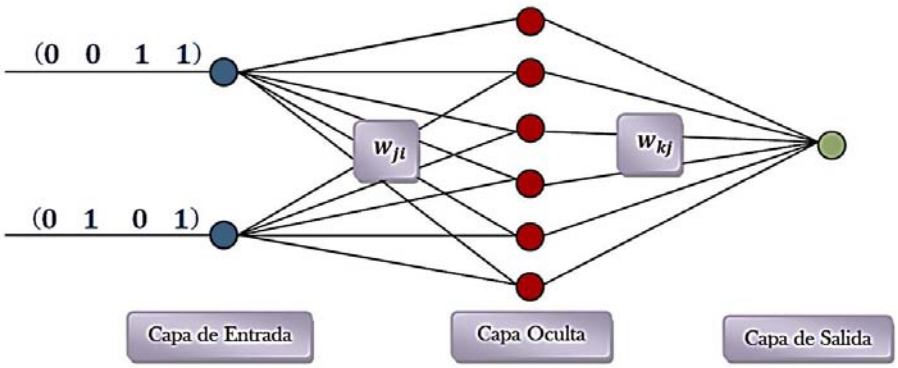
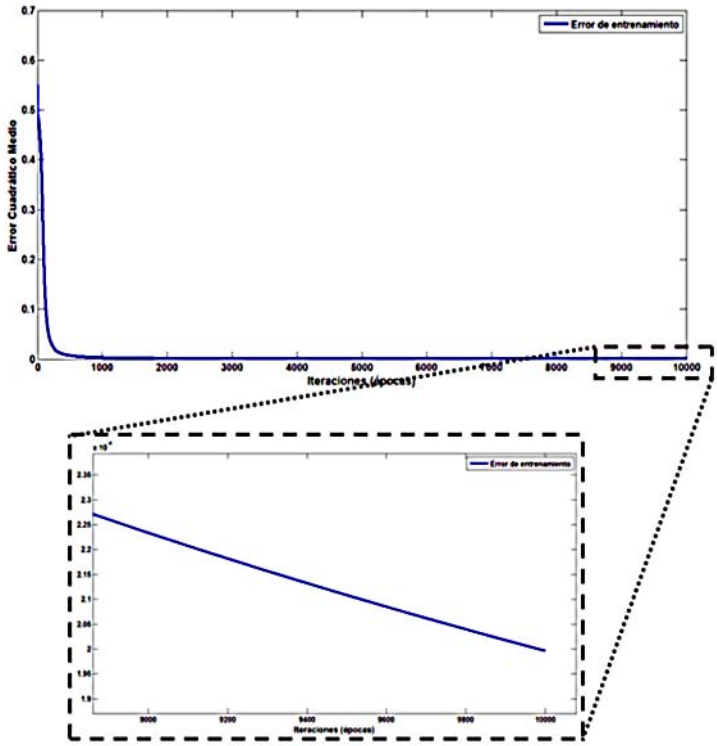


*Figura 1.19 Arquitectura para la función **XOR** con 6 neuronas ocultas.*

Teniendo el mismo número de iteraciones en los resultados mostrados y experimentos realizados, se puede decir que el error va decreciendo pero no es muy notorio, por lo que este comportamiento también es visible al estar aumentando el número de neuronas en la capa oculta, considerando entonces que un mayor número de neuronas ocultas no garantiza mejores resultados.

```
Error minimo de entrenamiento:
   1.9964e-04

Salidas deseadas
     0     1     1     0

Salidas obtenidas
    0.0127    0.9908    0.9893    0.0062

PESOS Y UMBRALES OPTIMOS:
*Capa oculta
Wji =
    1.3359    1.0698
   -4.8165    7.0851
    4.0200   -3.1966
    6.3957   -4.3251
    3.6033    3.4340
    2.6292    2.3834

bj =
   -1
   -1
   -1
   -1
   -1
   -1

*Capa de salida
Wkj =
    2.2795  -11.4919   -3.9781   -8.2404    8.6746    5.4367

bk =
   -1

Elapsed time is 13.692159 seconds.
```



Resultado 1.3

Hasta aquí se ha visto el tipo de red que se puede construir, dando un ejemplo usual que se encuentra en la literatura y probando que nuestro algoritmo programado es adecuado para su funcionamiento.

A continuación, en el siguiente capítulo, se adentrará sobre las ST, problema adoptado para este trabajo, describiendo y ejemplificando el cómo podemos ingresar las ST a la **RNA**, además de ilustrar de cómo realizar el pronóstico sobre la misma ST; dando lugar a los resultados alcanzados para ST como: función seno, función con ruido y sistema de *Lorenz*.

CAPÍTULO 2

SERIES DE TIEMPO Y SU PRONÓSTICO

La predicción es muy difícil, sobre todo si se trata del futuro.

Niels Bohr

2.1 Series de tiempo.

Una serie de tiempo (ST), es la colección de mediciones de cierto suceso o experimento registrados secuencialmente en el tiempo, a intervalos iguales. Comúnmente la notación de una ST es de la forma:

$$x(t) = \{ x(1), x(2), x(3), \dots, x(n) \}$$

$$t = 1,2,3, \dots, n$$

donde $x(t)$ es el valor registrado del proceso en el instante t.

El intervalo de tiempo puede ser en segundos, minutos, horas, días, meses, años, etc., o bien, cierto paso de tiempo que genere la secuencia.

Las ST que describen un sistema, como las manchas solares, consumo eléctrico, temperatura, etc., no es fácil ni inmediato conocer las ecuaciones que describen la relación explícita entre el valor de la serie en un instante de tiempo y sus valores anteriores, sino que sólo se dispone de un conjunto de datos observados. Estos datos requieren una interpretación para poder construir un modelo que aproxime la evolución de la serie temporal y permita predecir su comportamiento en el futuro.

2.2 Predicción.

Conocer la evolución o comportamiento a lo largo del tiempo –valor(es) futuro(s)- se puede formular a partir de un conjunto de muestras de la ST. Prediciendo el valor

inmediato siguiente al instante actual t, es la predicción a un paso de tiempo, siendo $x(t+1)$ el valor inmediato, utilizando un cierto número de muestras anteriores $\{x(t), x(t-1), x(t-2), \dots\}$, ésta es la ST que describe un sistema.

La predicción en múltiples pasos de tiempo consiste en pronosticar el comportamiento de la serie desde el instante inmediato, hasta donde permita hacer dicha predicción

$$\{ x(t+1), x(t+2), x(t+3), \dots, x(t+1+\tau)\}$$

teniendo un intervalo de predicción limitado $[t+1, t+1+\tau]$, siendo τ un número natural que representa el horizonte de predicción, conocido como límite de predictibilidad, a partir de la información disponible $\{x(t), x(t-1), x(t-2), \dots\}$.

Cuando no se dispone de las ecuaciones que describan el comportamiento de la ST, se recurre a 'métodos aproximativos' para construir modelos que permitan resolver el problema de predicción, cuyo comportamiento dinámico de dicha serie es la dificultad de predicción de la ST.

Un gran número de importantes aplicaciones prácticas en ciencias de la tierra puede ser considerado matemáticamente como un mapeo (*mapping*); surge la necesidad hacia mejorar la exactitud de la interpolación del desarrollo de la aproximación de la red neuronal. Las técnicas de emulación de la red neuronal han sido desarrolladas en los mapeos (*mappings*) con alta aproximación y exactitud de interpolación.

2.3 Manejo de la RNA en ST

En este apartado se describe la manera de introducir la serie de tiempo a nuestro modelo de red neuronal artificial, este manejo de la **RNA** en ST puede diferir de otros modelos y trabajos que el lector haya visto y/o manejado.

Con lo descrito hasta ahora, el objetivo principal es hacer uso de un modelo de autómata para hacer pronóstico mediante las series de tiempo que se tengan disponible. Es simple hacer una descripción de ello, teniendo nuestros datos disponibles, en este caso n datos (valores) $x(t) = \{ x(1), x(2), x(3), \dots, x(n) \}$, se introducen al modelo neuronal para obtener uno o más valores futuros $\{ x(t+1), \dots, x(t+1+\tau) \}$.

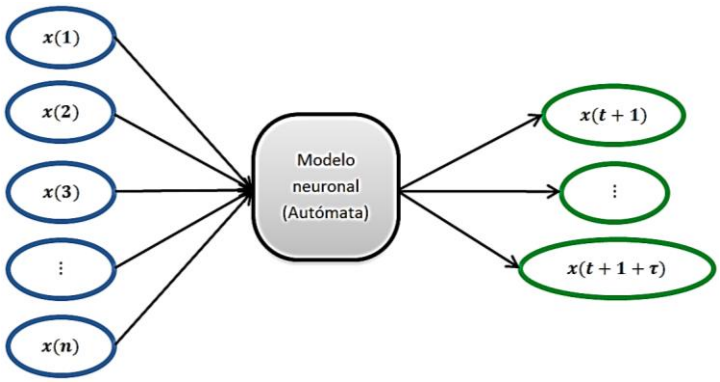


*Figura 2.1 Uso de la **RNA** en ST.*

Basados en lo descrito de la estructura y funcionamiento de una red neuronal, en la figura 2.1 se tiene que cada dato lo podemos considerar como nodo de entrada y que cada valor futuro es un nodo de salida, teniendo que $\boldsymbol{x(t) = x_i^{\mu}}$, esto es siguiendo la expresión matemática del sistema neuronal artificial, siendo $\boldsymbol{\mu}$ el patrón o conjunto de patrones, en este caso $\boldsymbol{\mu = 1}$ y por lo tanto se tiene que los nodos de entrada y salida se comportan como vector columna **(ver estructura y funcionamiento de una RNA).**

Como se ha de manejar una red neuronal supervisada en este trabajo, cuando se esté en la etapa de entrenamiento se supervisara que tan bueno está aprendiendo para que arroje los valores deseados, éstos son el objetivo (*target*) de la **RNA**, comparando los valores de salida con los del *target* en el cálculo de su error.

Por ejemplo, si se tiene un conjunto de valores de 15 datos, que representan nuestra serie de tiempo, queremos conocer al menos 3 datos inmediatos (futuros), se puede realizar lo siguiente. Utilizando los primeros 12 datos como nodos de entrada y los 3 valores restantes como conjunto objetivo para supervisar la red, ilustrándolo de esta manera:

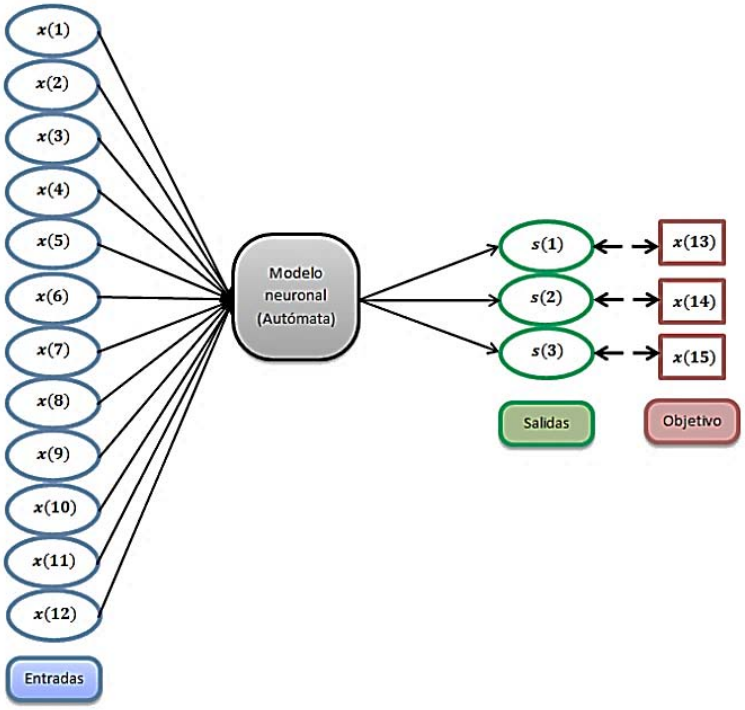


Figura 2.2 Ejemplo 1 de una ST y sus valores futuros.
***RNA** supervisada.*

Al término del entrenamiento, se ha de obtener una salida, ya sea idéntico o aproximado a lo deseado. Entonces este autómata ha aprendido cierto comportamiento de la serie por

lo que se puede realimentar el modelo con las salidas obtenidas o bien con el conjunto de datos objetivo para realizar un posible pronóstico y conocer los valores futuros, considerando que al realimentar la **RNA** ya no está efectuando ningún tipo de entrenamiento, esto se puede ilustrar de la siguiente manera:

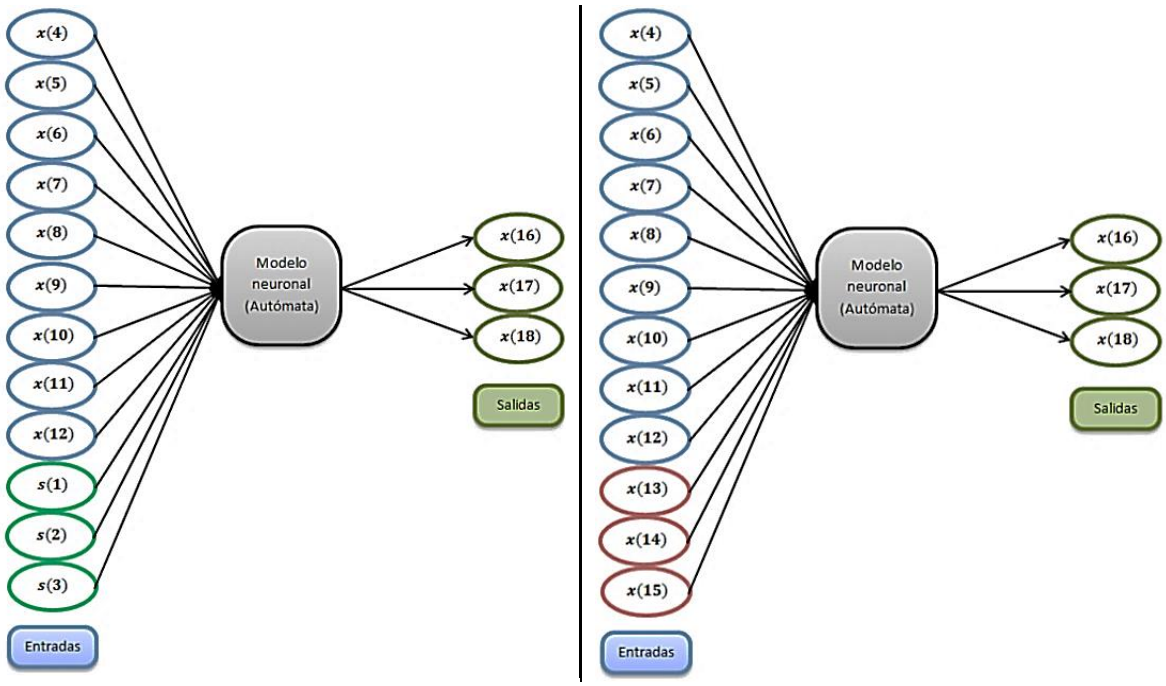


Figura 2.3 Pronóstico de la ST.
*Realimentación de la **RNA** supervisada (ejemplo 1).*

Se observa en la figura 2.3, que al realimentar la red se realiza un desfase, en este caso de tres valores, o tres pasos de tiempo (*delay*), ya que al inicio, se cuentan con 15 datos, de éstos se utilizan los 12 primeros para ser entrenados y los siguientes son usados como objetivo, al realizar el pronóstico se recorre la ST, ignorando los tres primeros datos y utilizando los tres valores de salida y objetivo secuencialmente, y así se obtienen los siguientes valores futuros.

Este ejemplo es una forma en cómo utilizar la **RNA** en ST y hacer un pronóstico, teniendo en cuenta, que este ejemplo es particular, ya que como se tienen doce entradas y se obtienen tres salidas, siempre tendrá esta estructura si se utilizara nuevamente para realimentar la red y hacer pronóstico hasta donde se permita, esto último es en la decisión del pronosticador, ya que se podría tener que los valores futuros obtenidos por la red neuronal no sean aptos o acordes a lo que realmente describe el sistema de la ST.

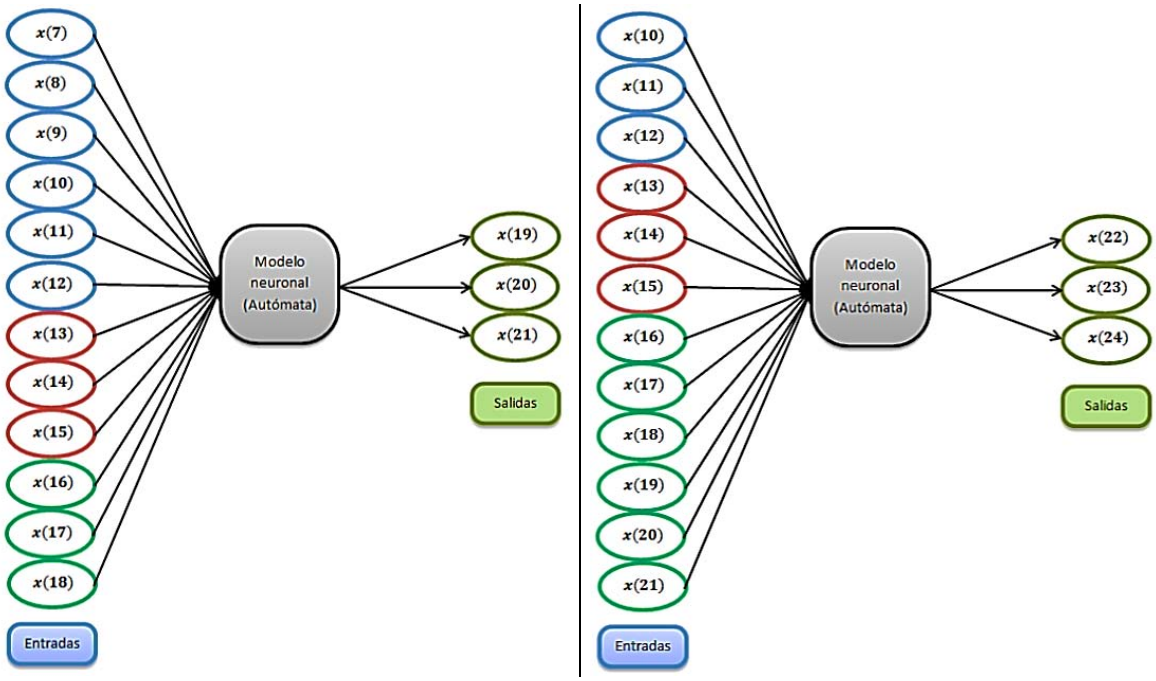


Figura 2.4 Pronóstico de la ST.
*Realimentación de la **RNA** supervisada (ejemplo 1).*

Se ilustra en la figura 2.4 una realimentación a la **RNA** con los valores futuros obtenidos de ésta, recordando que en esta fase de hacer predicción no se realiza ningún tipo de entrenamiento, ya que desde un inicio se realizó con la ST que se tenía.

Entonces, en un inicio la ST contaba con 15 valores $x(t) = \{ x(1), x(2), x(3), \ldots, x(14), x(15) \}$, entrenada la **RNA** con esta ST, se realiza la

predicción, obteniendo los valores futuros deseados $x(t+\tau) = \{\, x(16), x(17), x(18), \ldots, x(23), x(24) \,\}$, basados en las figuras 2.3 y 2.4, dependiendo los valores predichos si son o no acordes al sistema que describa. Se tendrá un límite a la predictibilidad $\boldsymbol{\tau}$, ya que se podría hacer pronóstico hasta el fin de los tiempos con este modelo, pero, ¿qué tan buenos o aproximados serán estos datos predichos con lo real?; se darían diversas respuestas y conclusiones, ya que las ST describen diversos sistemas dinámicos y cuyo comportamiento sean diferentes uno con el otro, por lo que más adelante se corroborará esta cuestión contemplando el límite a la predictibilidad con las pruebas de este trabajo.

Se seguirá explicando e ilustrando las formas que podemos utilizar las ST disponibles en la red neuronal, ya que se pueden ingresar los datos en diferente forma y podemos obtener diferentes resultados que puedan influir en el pronóstico.

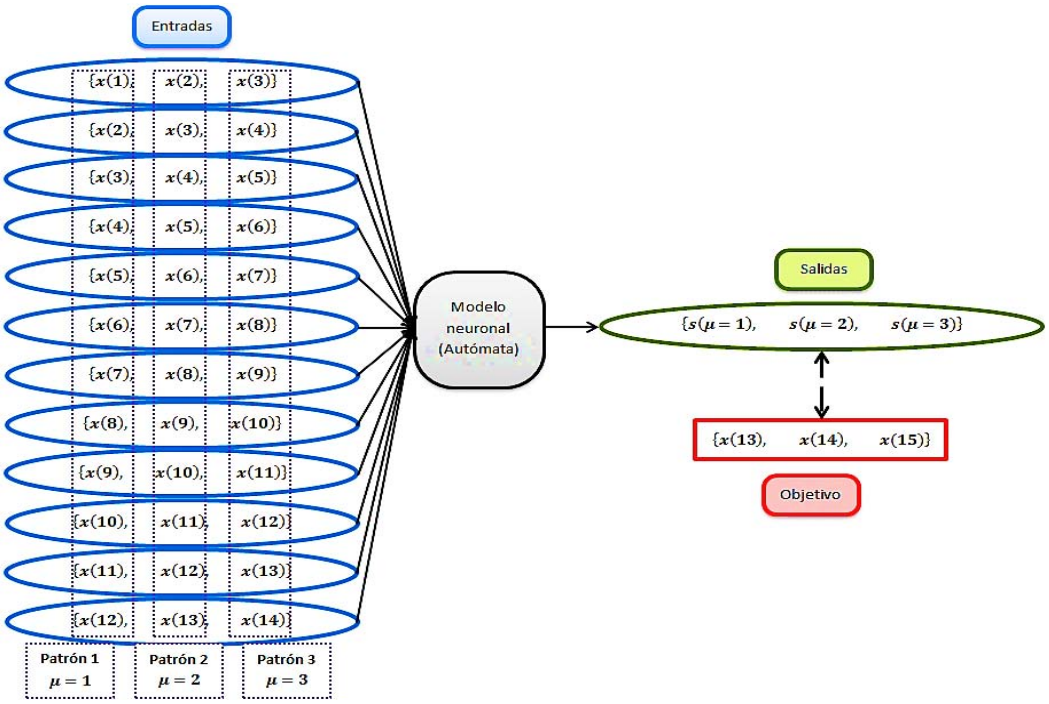


*Figura 2.5 **RNA** supervisada, ejemplo 2.*

Dado el primer ejemplo en este apartado del manejo de la **RNA** en ST, se tiene que $\boldsymbol{\mu = 1}$, lo cual corresponde como un patrón, pero se puede trabajar las ST con más de un solo patrón. Por ejemplo, de la misma serie de 15 valores, sea ahora $\boldsymbol{\mu = 3}$ y que sólo queremos avanzar un paso de tiempo, o sea pronosticar un solo valor.

Se usarán los primeros 14 datos de la ST para entrenar la **RNA** y lo supervisarán los valores mostrados en la figura 2.5 $\{x(13), x(14), x(15)\}$, siendo estos el objetivo de la red neuronal. Se mencionó que un solo valor es pronosticado, por lo que solo hay un nodo de salida que contiene tres valores, esto es porque $\boldsymbol{\mu = 3}$, al igual que cada nodo de entrada contiene tres valores.

	Patron_1	Patron_2	Patron_3
Entrada_1	$x(1)$	$x(2)$	$x(3)$
Entrada_2	$x(2)$	$x(3)$	$x(4)$
Entrada_3	$x(3)$	$x(4)$	$x(5)$
Entrada_4	$x(4)$	$x(5)$	$x(6)$
Entrada_5	$x(5)$	$x(6)$	$x(7)$
Entrada_6	$x(6)$	$x(7)$	$x(8)$
Entrada_7	$x(7)$	$x(8)$	$x(9)$
Entrada_8	$x(8)$	$x(9)$	$x(10)$
Entrada_9	$x(9)$	$x(10)$	$x(11)$
Entrada_10	$x(10)$	$x(11)$	$x(12)$
Entrada_11	$x(11)$	$x(12)$	$x(13)$
Entrada_12	$x(12)$	$x(13)$	$x(14)$
Salida	$s(\mu = 1)$	$s(\mu = 1)$	$s(\mu = 3)$
Objetivo	$x(13)$	$x(14)$	$x(15)$

*Tabla 2.1 Entradas y salidas de la **RNA** supervisada, con tres patrones.*

Para pronosticar en este segundo ejemplo se hace un desfase de un paso de tiempo, contemplando que el valor pronosticado será $\boldsymbol{x(16)}$. A la vez podemos observar que tan bien estará haciendo la predicción, ya que los otros dos valores del nodo de salida serían aproximados o idénticos a $\{x(14), x(15)\}$, entonces tendremos el valor futuro en el nodo de salida del patrón tres **(ver tabla 2.2)**.

	Patron_1	Patron_2	Patron_3
Entrada_1	$x(2)$	$x(3)$	$x(4)$
Entrada_2	$x(3)$	$x(4)$	$x(5)$
Entrada_3	$x(4)$	$x(5)$	$x(6)$
Entrada_4	$x(5)$	$x(6)$	$x(7)$
Entrada_5	$x(6)$	$x(7)$	$x(8)$
Entrada_6	$x(7)$	$x(8)$	$x(9)$
Entrada_7	$x(8)$	$x(9)$	$x(10)$
Entrada_8	$x(9)$	$x(10)$	$x(11)$
Entrada_9	$x(10)$	$x(11)$	$x(12)$
Entrada_10	$x(11)$	$x(12)$	$x(13)$
Entrada_11	$x(12)$	$x(13)$	$x(14)$
Entrada_12	$x(13)$	$x(14)$	$x(15)$
Salida (deseada)	$x(14)$	$x(15)$	$\boldsymbol{x(16)}$

Tabla 2.2 Pronóstico a un paso de tiempo (ejemplo 2).

Basados en estos dos ejemplos, se tiene una ST de tiempo de 15 valores, ahora se tendrá tres conjuntos, un primer conjunto será el de entrada, que va siendo el conjunto a entrenar; el segundo es el que estará supervisando el entrenamiento, cuyo conjunto es el objetivo; el tercero se usará para probar la red neuronal, conocido como *'test'* **(ver capacidad**

de generalización de la red), este último es un posible pronóstico el cual se mide al momento cuando está entrenando la red.

Utilizando para este ejemplo (ejemplo 3), un nodo de entrada y un nodo de salida. El nodo de entrada contendrá el conjunto de entrenamiento, para este caso pondremos cinco valores secuenciales, que van siendo los primeros datos de la ST $\{x(1), x(2), x(3), x(4), x(5)\}$, entonces tendremos lo que es un vector fila **(ver estructura y funcionamiento de una RNA),** en contraste con el primer ejemplo de este apartado. Dada esta forma de ingresar la ST al modelo, el nodo de salida poseerá también cinco valores, por lo que $\boldsymbol{\mu = 5}$, ya que la forma que se ha manejado y descrito en el presente trabajo es, el número de patrones que ingresa a la **RNA** será el mismo número de patrones en la salida, independientemente del número de nodos (neuronas) que se manejen tanto para la entrada como para la salida.

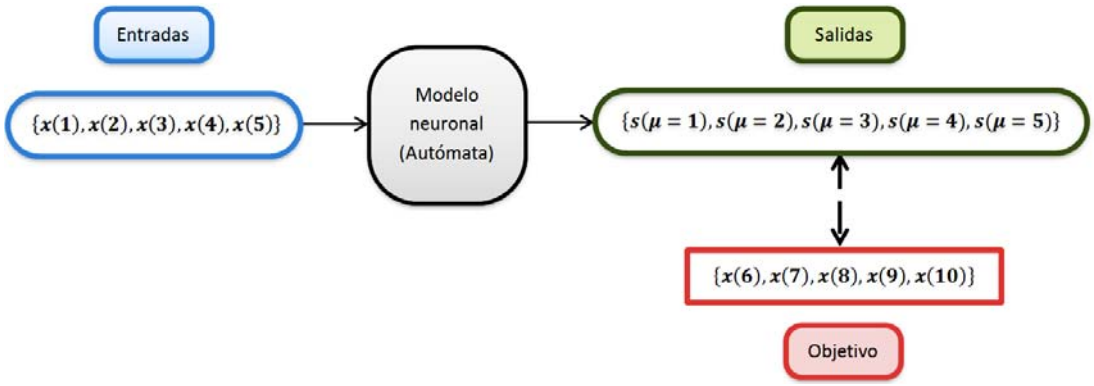


*Figura 2.6 **RNA** supervisada con un nodo de entrada y uno de salida, con cinco patrones. Fase de entrenamiento (ejemplo 3).*

Para este ejemplo 3, el conjunto objetivo será los cinco valores seguidos del primer conjunto $\{x(6), x(7), x(8), x(9), x(10)\}$, y el conjunto de prueba serán los últimos cinco valores de la ST $\{x(11), x(12), x(13), x(14), x(15)\}$.

Para probar la **RNA**, el conjunto *test* se utilizará como objetivo, obteniendo un error entre éste y la salida (error de prueba), se espera que la salida sea igual o aproximada a la del objetivo, tal como se pretende al supervisar la red al ser entrenada.

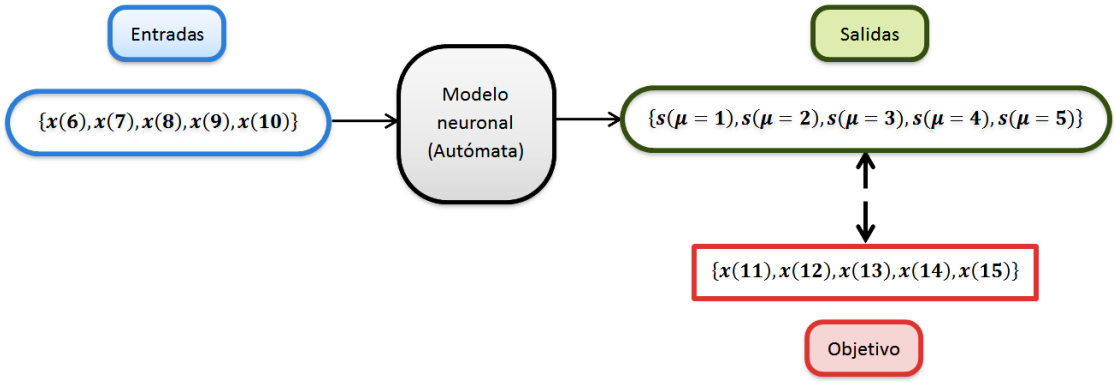


*Figura 2.7 Prueba de la **RNA** supervisada (ejemplo 3).*

Al probar la red, se hace un desplazamiento, así como se ha realizado en los dos ejemplos anteriores en el pronóstico, en este ejemplo, las salidas obtenidas de la red se utilizan como entradas, por lo que se está recorriendo un ventaneo de la ST **(ver figura 2.8).**

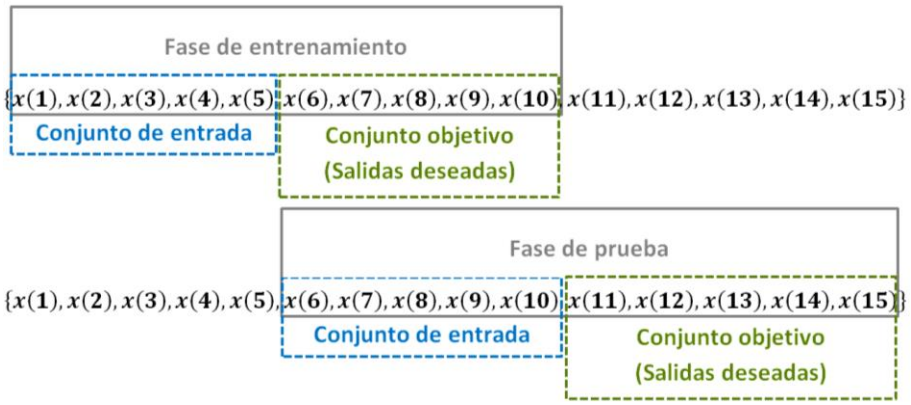


Figura 2.8 Ventaneo de la ST, etapa de entrenamiento y de prueba.

Recordando que al estar entrenando se mide el error entre la salida y el objetivo, ajustando los parámetros de la **RNA**, que en esencia son los pesos. Con estos pesos, que

se utilizaron para obtener el error de entrenamiento, se utilizan para realizar el *test*, por lo que ahora el conjunto de prueba se usa como objetivo.

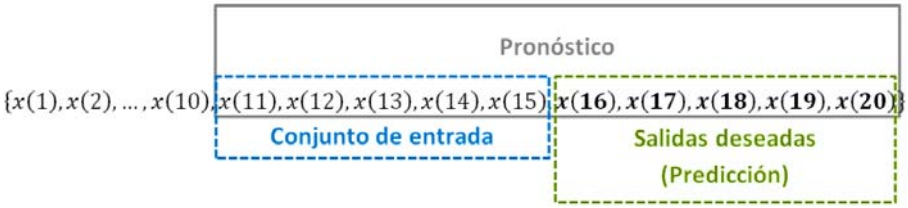


Figura 2.9 Pronóstico para el ejemplo 3.

La predicción se ha de realizar similar a los ejemplos anteriores, de la misma forma como se prueba la red en este ejemplo, por lo que se desplaza este ventaneo utilizando las últimas salidas de la red que son las obtenidas al realizar el *test* **(ver figura 2.9)**, obteniendo cinco valores futuros deseados, que serían los valores secuenciales a la ST $\{x(16), x(17), x(18), x(19), x(20)\}$.

Entrenamiento					
Entrada	$x(1)$	$x(2)$	$x(3)$	$x(4)$	$x(5)$
Salida	$s(\mu = 1)$	$s(\mu = 1)$	$s(\mu = 3)$	$s(\mu = 4)$	$s(\mu = 4)$
Objetivo	$x(6)$	$x(7)$	$x(8)$	$x(9)$	$x(10)$
Prueba					
Entrada	$x(6)$	$x(7)$	$x(8)$	$x(9)$	$x(10)$
Salida	$s(\mu = 1)$	$s(\mu = 1)$	$s(\mu = 3)$	$s(\mu = 4)$	$s(\mu = 4)$
Objetivo	$x(11)$	$x(12)$	$x(13)$	$x(14)$	$x(15)$
Pronóstico					
Entrada	$x(11)$	$x(12)$	$x(13)$	$x(14)$	$x(15)$
Salida deseada	$x(16)$	$x(17)$	$x(18)$	$x(19)$	$x(20)$

Tabla 2.3 Fase de entrenamiento, de prueba y pronóstico del ejemplo 3.

El siguiente ejemplo (ejemplo 4), se conservan los tres conjuntos: de entrada, objetivo y prueba; pero para estos conjuntos el desplazamiento del ventaneo de la ST está

sobrepuesto **(ver figura 2.10)**. Tal como en el ejemplo 3, se utilizará un nodo de entrada y un nodo de salida, solo que esta vez contendrán trece valores, por lo que el conjunto para entrenar está dado por los trece primeros valores. Haciendo un *delay* de uno en la ST, se tiene el conjunto objetivo y para el conjunto de prueba se aplica nuevamente otro paso de tiempo de uno.

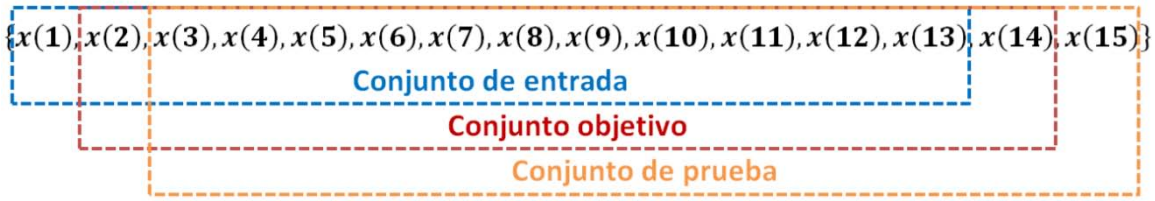


Figura 2.10 Ventaneo de conjuntos sobrepuestos.

Al emplear trece valores en el conjunto a entrenar, ha de abarcar la mayor parte de historia de la ST. Y como bien se ha mencionado, el mismo número de patrones que entra a la **RNA** es el mismo número de patrones que sale, por ello tanto el conjunto objetivo como el de prueba también contienen trece valores.

Al realizar el pronóstico, se ingresa la salida obtenida al realizar el *test*, que serían los valores deseados $\{x(3), x(4), x(5), \dots, x(14), x(15)\}$, entonces la salida predicha deseada sería:

$$\{x(4), x(5), x(6), x(7), x(8), x(9), x(10), x(11), x(12), x(13), x(14), x(15), \boldsymbol{x(16)}\}$$

Obteniendo el valor futuro a un paso de tiempo en el último patrón $\boldsymbol{x(16)}$, similar al del ejemplo 2.

Del ejemplo 3, también se puede variar el número de entradas, similar al ejemplo 2. En este caso, se quiere una **RNA** con cuatro nodos de entrada y un nodo de salida, considerando los tres conjuntos **(ver figura 2.11)**, para este ejemplo 5, se hace un *delay* de

uno en el conjunto que será entrenado, esto se hace para obtener los cuatro nodos de entrada, esto implica que cada nodo contenga 4 patrones **(ver figura 2.12),**

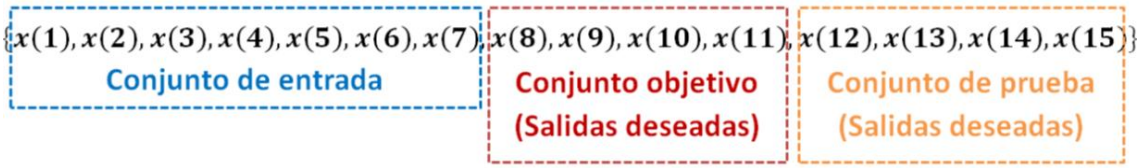


Figura 2.11 ST de quince datos divididos en tres conjuntos.

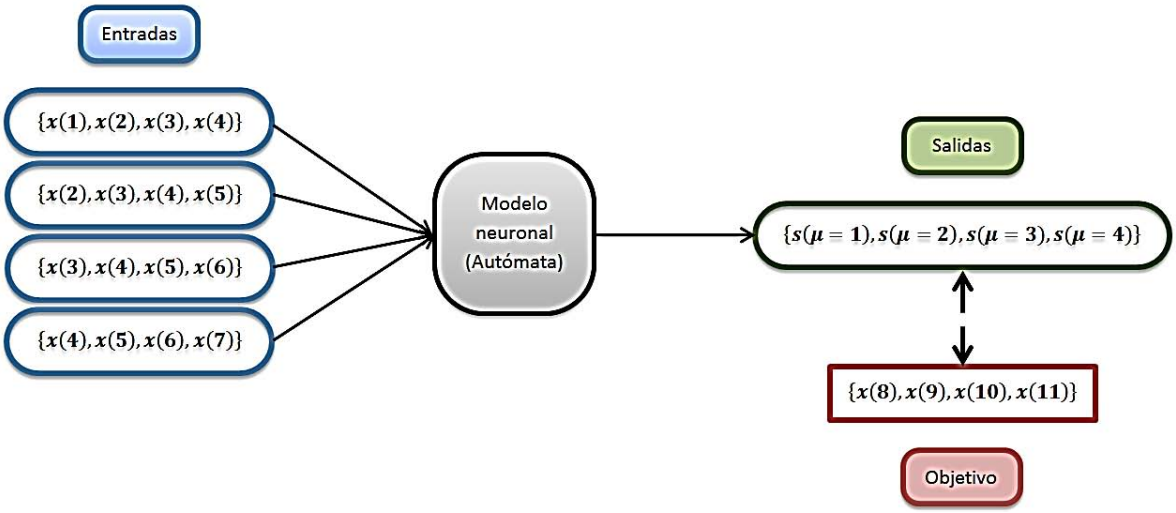


*Figura 2.12 **RNA** supervisada con cuatro nodos de entrada y uno de salida, Cada nodo con cuatro patrones. Fase de entrenamiento (ejemplo 5).*

Al probar el modelo neuronal se desplaza el ventaneo de cada conjunto, teniendo que el nuevo conjunto objetivo será el conjunto de prueba, y las salidas obtenidas de la red, que se esperan sean idénticas o aproximadas al del objetivo, formarán parte del conjunto de entrada **(ver figura 2.13).**

Terminando el entrenamiento y con ello, obteniendo los parámetros óptimos para el autómata, se puede realizar el pronóstico, realizándolo de la misma forma cuando se prueba la **RNA**. Entonces, se recorre el ventaneo de cada conjunto, obviando que en esta

fase no hay conjunto objetivo, simplemente se espera que los valores futuros deseados sean los secuenciales de la ST.

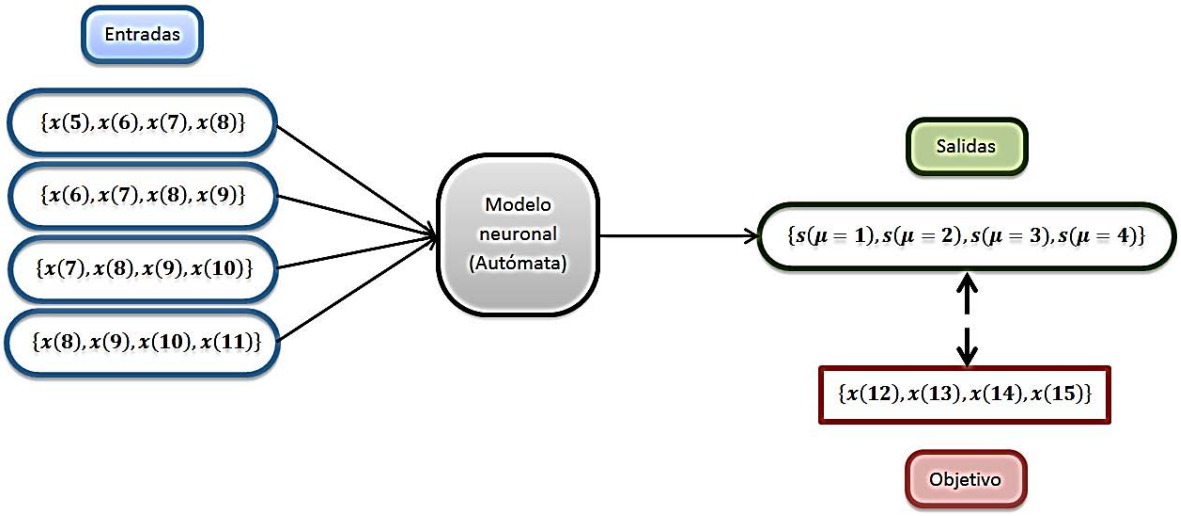


*Figura 2.13 **RNA** supervisada con cuatro nodos de entrada y uno de salida, Cada nodo con cuatro patrones. Fase de prueba (ejemplo 5).*

La forma como se hace la predicción, es utilizar las salidas deseadas de la **RNA**, realimentándola con éstas y utilizando los últimos valores del anterior conjunto de entrada, obteniendo cuatro valores futuros secuenciales a la ST. Por ejemplo, si se quiere seguir ocupando este modelo para realizar pronóstico y obtener los siguientes valores futuros deseados$\{x(20), x(21), x(22), x(23)\}$, se tendría al conjunto de entrada con los valores $\{x(13), x(14), x(15), x(16), x(17), x(18), x(19)\}$, recordando que la estructura que tiene la red neuronal es de cuatro nodos de entrada.

Hasta ahora, con los ejemplos descritos, el 'usuario' puede experimentar diferentes estructuras para la **RNA**, implicando la forma de ingresar la ST al modelo. Todo esto, con el propósito de realizar predicción. Analizando los valores que estaría obteniendo, tomando

la decisión si la predicción sea o no óptima, o bien, que esté dentro de un rango aceptable y le sea eficiente en su objetivo de lo que esté llevando acabo **(ver función XOR)**.

Dependiendo la cantidad de datos que se tengan, se pueden usar $\boldsymbol{n}$ nodos de entrada como $\boldsymbol{m}$ patrones, así como el tamaño del *delay* que se llegará a usar, esto es a consideración al usuario al estar experimentando.

La predicción que se ha de realizar, como se ha visto en los ejemplos, podemos obtener desde un valor futuro a múltiples valores, el resultado de éstos es primordial ya que pueden, o no, ser acordes a la dinámica de la ST.

Teniendo estas nociones de como ingresar las ST al modelo de una **RNA**, a continuación se desglosa un par de apartados; en uno, indica cómo tratar las ST antes de introducirlas al autómata, esto es como un pequeño tratamiento que se describe frecuentemente en la literatura. En el siguiente, se especifica algunos parámetros al comenzar y el estar experimentando en el entrenamiento, que de igual manera, es como se maneja en la literatura.

2.3.1 Pre-procesado.

En este apartado se describe sobre la elección de los datos de aprendizaje, así como su acondicionamiento para poderlos ingresar al modelo de la **RNA**.

La condición que se debe imponer a los ejemplos (patrones) que se presentan al modelo, para llevar a cabo el entrenamiento, es que sean un número suficiente como para ser

representativos del sistema que queremos modelar. Refiriéndose a que queden cubiertas todas las situaciones que pueden darse, es decir, que los valores de las entradas y salidas cubran uniformemente todo su rango.

Masters T., 1993, menciona que: es aconsejable, pero no indispensable, que los datos que se proporcionan a la red, posean:

Buena distribución, es decir, sigan una distribución estándar o uniforme.

Rangos de valores parecidos para todas las entradas.

Rangos acotados dentro del intervalo de trabajo de la función de activación empleada en las capas ocultas y de salida de la **RNA**.

Si se trabaja con datos que no cumplan lo anterior, la **RNA** podrá, en algunos casos, seguir dando buenos resultados pero invertirá un mayor tiempo en conseguirlo. Si se desea cambiar el tipo de distribución de los datos, es preciso aplicar una transformación, el interés de ello radica en no enmascarar la información ya que puede: al moverse en un pequeño intervalo de cierta entrada se produzcan variaciones importantes en la salida de la red neuronal, mientras que en otro intervalo mayor, las variaciones pueden no afectar tanto a la salida. Las transformaciones más frecuentes se basan en el empleo de logaritmos, raíces cuadradas y cúbicas.

Para modificar el rango de valores de las entradas se aplica un escalado. El objetivo es acotar sus valores entre $[\mathbf{0}, +\mathbf{1}]$ o $[-\mathbf{1}, +\mathbf{1}]$, límites de la función de activación para evitar la saturación de ésta, al realizar su respectiva regla de propagación (en ocasiones se

recomienda que el acotamiento sea entre $[-\mathbf{0.7}, +\mathbf{0.7}]$ para tratar de mantener la operación de la neurona más cerca de una zona casi-lineal de operación). Para ello se busca el mínimo y el máximo del rango de entradas y se transforma el intervalo de forma lineal.

Un escalado que proporciona con frecuencia buenos resultados, es al aplicar dos transformaciones consecutivas:

En primer lugar se realiza un estandarizado, es decir, se transforma las variables de forma que presenten media cero y varianza uno. Se calcula de la manera usual la media $(\bar{\boldsymbol{x}})$ y la desviación estándar $(\boldsymbol{\sigma})$ de cierta entrada, siendo su expresión: $\boldsymbol{x}' = \frac{(\boldsymbol{x}-\bar{\boldsymbol{x}})}{\boldsymbol{\sigma}}$

A continuación, pueden escalarse los nuevos valores al intervalo $[\mathbf{0}, +\mathbf{1}]$ o $[-\mathbf{1}, +\mathbf{1}]$ mediante una transformación lineal.

A partir de este escalado conseguimos que todas las entradas tengan la misma importancia al igualar su variabilidad (gracias al estandarizado) y su rango de valores, sin importar el rango y distribución inicial.

Existen otros tipos de pre-procesamiento más complejos, como la transformación de *Fourier*, análisis de componentes principales o técnicas de extracción de características (*feature extration*). Algo tan sencillo como cambiar las entradas por otras que sean cocientes de ciertas variables (*ratios*) puede dar muy buenos resultados.

2.3.2 Procesado (proceso de entrenamiento).

Los parámetros de la red que pueden modificarse en los experimentos son: pesos iniciales, ritmo de aprendizaje, número de neuronas ocultas y parada de entrenamiento.

Inicialización de pesos. El método más extendido consiste en realizar una inicialización aleatoria en un cierto intervalo, como por ejemplo, entre $[-\mathbf{0.3} \ +\mathbf{0.3}]$. No obstante, existen otros enfoques de inicialización más inteligente que la meramente aleatoria; se trata de aplicar algoritmos de minimización de funciones a esta elección de pesos iniciales, como la regresión lineal, *simulated anneling* y algoritmos genéticos.

Ritmo de aprendizaje. También se puede considera como una tasa de aprendizaje, este parámetro desempeña un papel crucial en tal proceso, ya que controla el tamaño de los cambios en los pesos de las neuronas. Toma valores muy dispares, desde $\mathbf{0.5}$ a $\mathbf{0.00001}$ y aún menor; uno de los motivos de esta variabilidad reside en el que el aprendizaje *batch* –aprendizaje por lotes-, al acumular el cambio en los pesos propuestos por cada patrón, se obtiene un cambio mayor cuantos más patrones se tengan en el conjunto de aprendizaje.

Elegir un ritmo de aprendizaje demasiado pequeño implica que la **RNA** realiza cambios pequeños en sus pesos, los cual es perjudicial en dos sentidos: evita escapar mínimos locales y disminuye la rapidez de convergencia. Por otra parte, decidirse por un valor alto puede ocasionar grandes variaciones en los pesos sinápticos, lo cual puede conducir a inestabilidades en la **RNA** o a saturar sus neuronas.

Una conclusión es: interesa un ritmo de aprendizaje que varíe durante el entrenamiento, adaptándose a las necesidades. Hay muchos enfoques de este tipo, desde simples reglas heurísticas hasta un control mediante reglas borrosas de su valor. Al margen de estas técnicas más sofisticadas, se recomienda, en general, emplear el mayor ritmo de aprendizaje que no provoque excesivas oscilaciones en los errores mostrados por la **RNA**, modificándolo a mano si es necesario.

Neuronas ocultas. No existe una receta que indique el número de neuronas ocultas que, para un problema dado, deban emplearse. En la literatura hacen mencionar que gracias a algunos resultados, se tiene constancia teórica de que basta con una capa oculta para resolver cualquier problema de aproximación funcional con un perceptrón, aunque en ocasiones se recurra a dos por cuestiones prácticas. Si se coloca un número excesivo de neuronas ocultas ocurrirá algo parecido en el ajuste de unos datos experimentales a un polinomio, empleando uno de grado demasiado elevado, el cual podría sobrar grados de libertad, aunque ajustan perfectamente, los casos que nos proporcionan (conjunto de aprendizaje), se apartaría de la técnica general y fallaría ante nuevos casos (conjunto de *test*). Por otro lado, si el número de neuronas no es suficiente, no se obtendrá un error aceptable ni siquiera para los datos que queremos ajustar.

Para decidir el número de neuronas ocultas a emplear, se puede recurrir a técnicas como las siguientes:

- Recetas. Unas son de tipo geométrico, como por ejemplo que la **RNA** tenga aspecto de pirámide (el número de neuronas en cada capa va decreciendo desde

las entradas hacia las salidas). Otras imponen condiciones relativas al número de patrones disponibles, intentando ajustar los grados de libertad a la cantidad de ejemplos, en aras de una buena generalización (se sugiere que el número de pesos de la red debe ser de la décima parte del de patrones). No obstante, ninguna de estas reglas es infalible.

- Prueba y error. Partiendo de un número sumamente pequeño de neuronas ocultas (por ejemplo, dos), se procede a realizar el entrenamiento con validación cruzada. El proceso se repite para distintas arquitecturas, cada vez con más neuronas ocultas, hasta llegar a la arquitectura que proporciona el resultado óptimo para los de aprendizaje y de *test*.
- Métodos dinámicos (arquitecturas evolutivas). Consisten en que sea el propio algoritmo el que, en función de los ejemplos de aprendizaje, ajuste la arquitectura de la red mediante la creación, destrucción o compartición de neuronas o conexiones durante el entrenamiento (o al término de éste). La introducción de estas técnicas complica el algoritmo de aprendizaje, garantizando su convergencia pero disminuyendo su rapidez. En cambio, no asegura un determinado grado de generalización; además, alguna de estas técnicas requiere un ajuste cuidadoso de sus parámetros de control. La ventaja principal es que evita la clásica prueba/error para encontrar la topología ideal del sistema neuronal artificial.

Parada del entrenamiento. No interesa prolongar indefinidamente el entrenamiento, pues llega un momento en el que se pierde generalización y tan sólo se memorizan los detalles

(ruido) del conjunto de aprendizaje. La decisión habitual es quedarse con los pesos de la **RNA** en la iteración para la cual se obtuvo el mínimo error en el conjunto de *test*.

Al finalizar la fase de entrenamiento, se almacenan los pesos ideales y con éstos se está en disposición de aplicar la **RNA** sobre los casos nuevos (no empleados en el entrenamiento) para medir su eficiencia de forma completamente objetiva. La evaluación de los resultados obtenidos es comprobando que se siguen obteniendo resultados dentro del margen de error deseado, procediendo a emplear la **RNA** dentro de su entorno de trabajo real.

2.4 Pruebas de la RNA en ST y su pronóstico.

Se muestra enseguida los experimentos realizados para este trabajo, en general, se trabaja con un modelo de **RNA** del tipo perceptrón multicapa, cuya arquitectura es de tres capas, una capa de entrada, una oculta y una de salida. Se recurrió a la prueba y error para el número de neuronas en la capa oculta, es decir, se fue variando el número de éstas analizando los resultados que arrojaba mediante el comportamiento del error.

El algoritmo de aprendizaje empleado en el autómata es del tipo **RPE**, que es de '**retropropagación de errores'**, teniendo entonces que es una **RNA** supervisada, dando seguimiento a los ejemplos mostrados en el apartado de manejo de la **RNA** en ST de este capítulo 2. Recordando que este algoritmo va modificando los pesos sinápticos entre las capas al calcular el error entre la salida de la red y el objetivo **(ver aprendizaje por RPE).**

La función de transferencia empleada fue la del tipo sigmoideo $\boldsymbol{f(h) = 1/1 + e^{-h}}$, tanto para la capa oculta como de salida, esto fue siguiendo la literatura, ya que recomiendan esta función para este tipo de **RNA**, de igual manera se siguió esta corriente en el uso del **PMC** para la predicción en ST.

En cada experimento se pueden presentar diferentes estructuras, esto es en la arquitectura de la red, conteniendo diferente número de nodos en la capa de entrada y la capa oculta, pero conservan lo que es un PMC de una capa oculta.

2.4.1 Función seno.

La función seno, bien conocida, es una señal periódica, la cual se repiten los patrones entre un rango de $[-\mathbf{1}, +\mathbf{1}]$.

Conociendo el comportamiento en un periodo, éste se repite secuencialmente, teniendo así una ST armónica suave como se muestra en la siguiente figura. Esta ST que se muestra, contiene $\mathbf{1000}$ datos y cuyo periodo es de $\mathbf{100}$. Se puede aplicar esta función como $\boldsymbol{y = \sin(x)}$, teniendo una variable independiente $\boldsymbol{x}$ y cuyo resultado es $\boldsymbol{y}$; para nuestro caso, se tiene los valores de esta última variable que están dentro del rango de la función.

Se menciona que se recurre a la prueba y error para el número de neuronas en la capa oculta, de igual manera fue para encontrar el número de nodos en la capa de entrada, variando a su vez un *delay* en el conjunto de entrada, tal como se ejemplificó anteriormente.

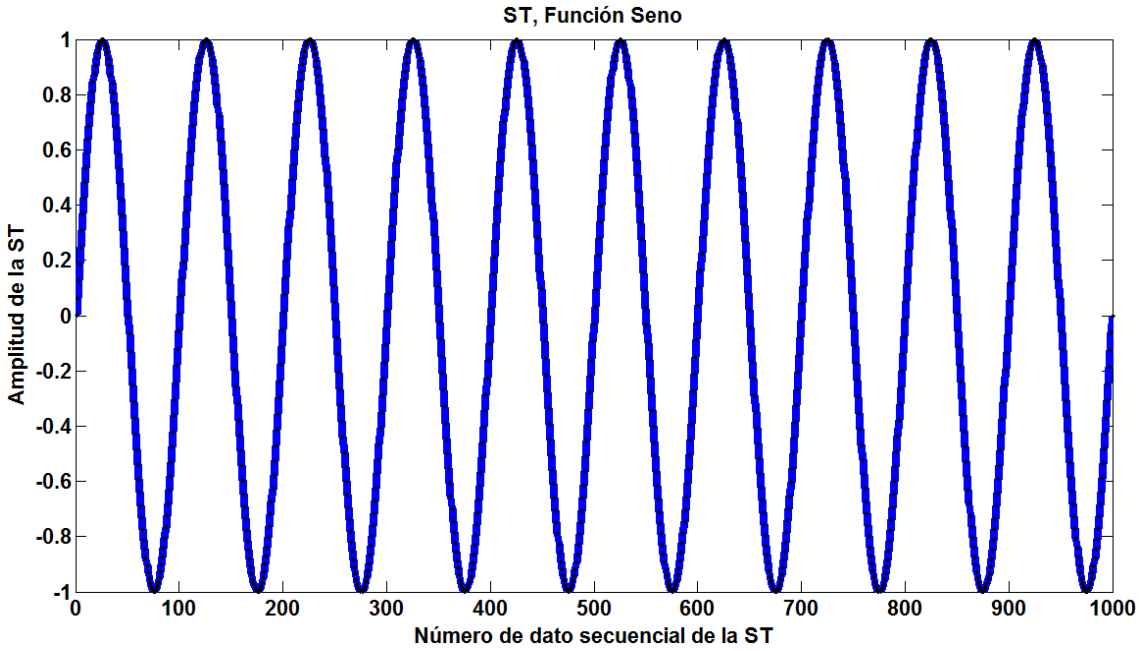


Figura 2.14 Función seno, ST de 1000 datos.

De inicio se pretende usar 100 patrones, primero por la periodicidad que tiene la ST; otra, porque al medir el error mediante, el error cuadrático medio, hace la suma de la comparación de cada valor dado, entonces, es más factible medir el error entre un grupo de 100 datos.

Manteniendo constantes los nodos de salida, que será de uno. También se mantiene constante el ritmo de aprendizaje, el cual va modificando los pesos para encontrar un mínimo global **(ver aprendizaje por RPE)**. Otra constante es el acelerador del **RPE**, esto es muy útil, ya que reduce el tiempo de convergencia; por ejemplo, si se llega a un mínimo en la iteración $\mathbf{100}$, con el acelerador lo podría encontrar en la iteración $\mathbf{30}$.

La parada del entrenamiento estará dada por el número de iteraciones (también se le llaman épocas), recordando que el método de aprendizaje es iterativo, en cada época se

modifican los pesos, al mismo tiempo que se calcula el error, por eso se puede analizar el comportamiento de éste.

Algo que no sufrirá cambios, cuando la **RNA** esté aprendiendo, serán los llamados umbrales, éstos podrían sufrir un ajuste tal como lo hacen los pesos, pero en esta prueba de la función seno, se observó que el autómata realiza un mejor entrenamiento cuando este parámetro se mantiene constante, en la literatura y en la práctica el umbral toma el valor de $\mathbf{0}$ o $\mathbf{1}$, al experimentar, se decide que este parámetro sea igual a $\mathbf{0}$.

Realizando diversos experimentos sobre esta ST, con los parámetros ya descritos, se llega al siguiente resultado, determinando una estructura para la **RNA**. El número de nodos en la capa de entrada será de $\mathbf{29}$, para la capa oculta se contemplan $\mathbf{11}$ neuronas y $\mathbf{1}$ en la de salida, cada nodo contiene $\mathbf{100}$ valores. En el conjunto de entrada, que son los utilizados para ser entrenados, se aplica un *delay* de $\mathbf{3}$, teniendo $\mathbf{184}$ datos para este conjunto.

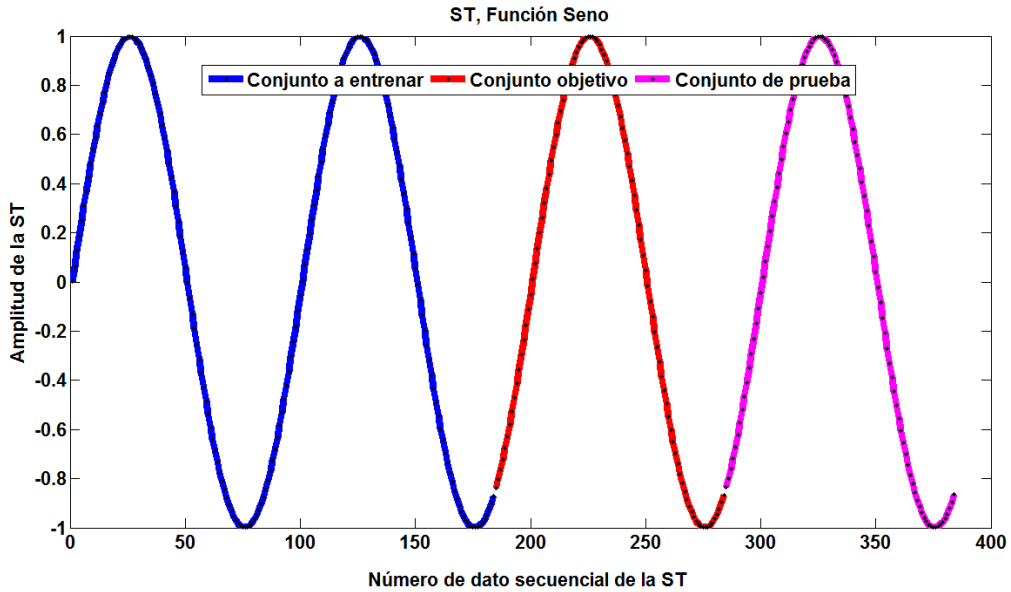


Figura 2.15 Conjunto de entrada y objetivo, función seno.

Sean los conjuntos de entrada y objetivo, el primero contempla los primeros $\mathbf{184}$ datos de la ST, mientras el segundo son los siguientes $\mathbf{100}$ valores secuenciales **(ver figura 2.15).**

El número de iteraciones realizadas es de $\mathbf{300\,000}$. La tasa de aprendizaje es $\boldsymbol{\varepsilon = 0.1}$ y $\boldsymbol{\alpha = 0.9}$ para el acelerador del **RPE**. El error mínimo obtenido durante el entrenamiento es de $\mathbf{5.96056}e^{-07}$, el cual va convergiendo hasta la última iteración **(ver figura 2.16).** Observando que el comportamiento del error va disminuyendo conforme va iterando, se muestra el error en diferentes figuras para distinguir su descenso.

Al entrenar el modelo, se prueba simultáneamente, adquiriendo un error de prueba de $\mathbf{5.96085}e^{-07}$ (en la misma iteración del error mínimo de entrenamiento), siendo muy idénticos ambos errores. Es ligeramente apreciable en las figuras del comportamiento del error que ambos errores sigan una misma tendencia, siendo muy idénticos estos errores.

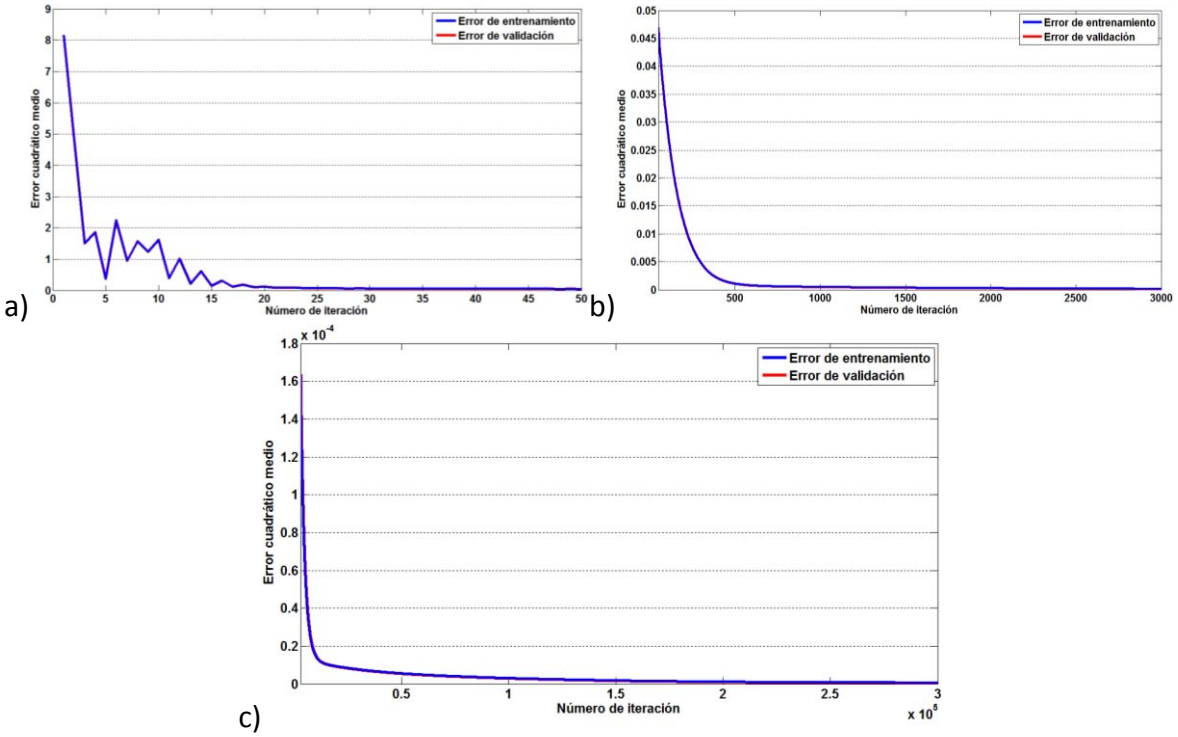


Figura 2.16 Comportamiento del error de la ST función seno.
a) Primeras 50 iteraciones, b) iteración 50 a 3000, c) iteración 3000 a 300000.

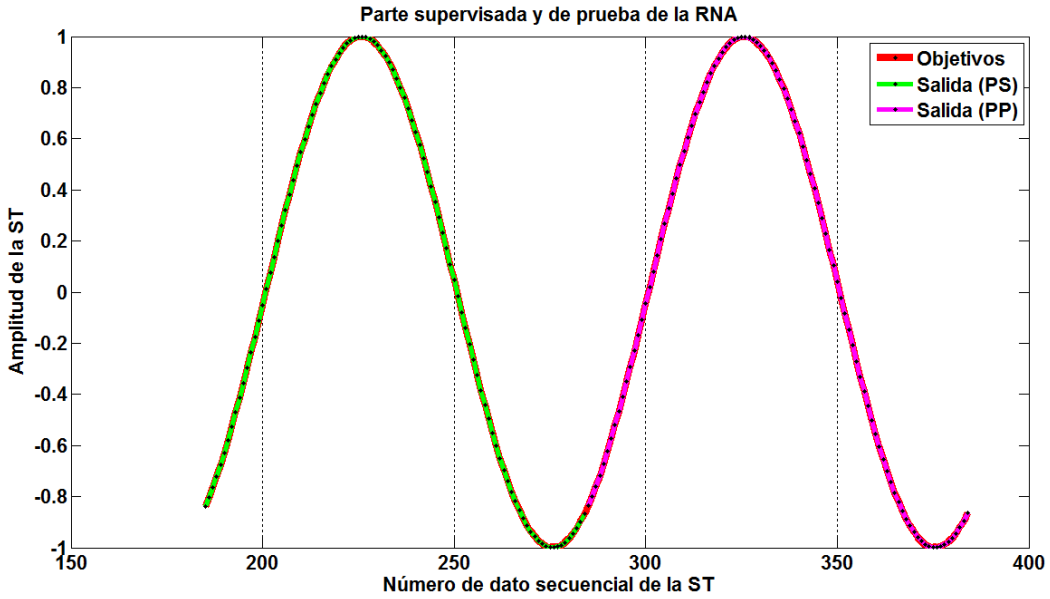


Figura 2.17 Comparación entre el objetivo y las salidas obtenidas de la RNA.

En la figura 2.17 se muestra la parte supervisada (PS) y de prueba (PP) de la ST, con la salida adquirida del modelo. Observando que la **RNA** aprendió la forma del conjunto de entrenamiento, llegando a ser similar al objetivo, obviando con cierto error, $\mathbf{5.96056}e^{-07}$ en la parte supervisada y $\mathbf{5.96085}e^{-07}$ en la parte de prueba.

Para realizar el pronóstico, se realimenta la red neuronal con las salidas obtenidas (ejemplificado y descrito al inicio de este capítulo), obteniendo 100 valores futuros cada vez que se es realimentada la red.

Considerando que se usaron menos de 200 datos para el entrenamiento, se hace una comparación con todas las salidas obtenidas del autómata, es decir, se calcula el error de lo pronosticado con el resto de la ST original, teniendo alrededor de 800 datos para calcularlo. Hay que hacer hincapié que el resto de los datos de la ST original, no se utilizaron para el entrenamiento, ni para realimentar a la **RNA**, simplemente, como se tienen disponibles, se puede analizar los datos reales con los predichos.

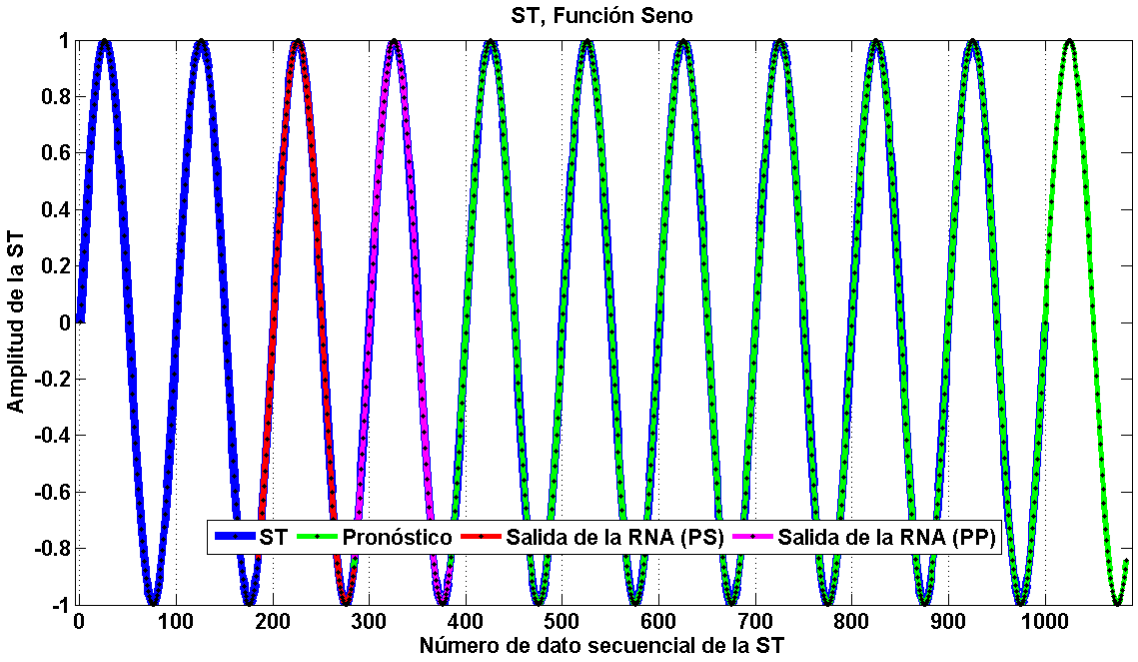


Figura 2.18 Pronóstico de la ST de la función seno.

Usando el **ECM**, se obtiene un error de $\mathbf{7.84051}e^{-06}$ respecto a las salidas de la **RNA** y lo pronosticado con la ST, lo cual se ve razonable, ya que el error calculado en el entrenamiento fue para cien valores, lo cual va sumando el error en cada cien.

2.4.2 Función seno con ruido.

Sea la ST mostrada en la figura 2.20 de una función seno con ruido, ST que contempla $\mathbf{1000}$ datos secuenciales. La estructura que se emplea para el autómata es igual que en el anterior problema, siendo esta una arquitectura de $\mathbf{29}$ nodos de entrada, $\mathbf{11}$ neuronas ocultas y un nodo de salida.

Esta ST, aunque contenga ruido, presenta una periodicidad, debido a que su función base es una función seno, por lo que se emplean $\mathbf{100}$ patrones en cada nodo, representando

éste la periodicidad de la ST. El *delay* que se maneja sobre la ST es de $\mathbf{4}$, diferente al que se manejó en ejercicio anterior.

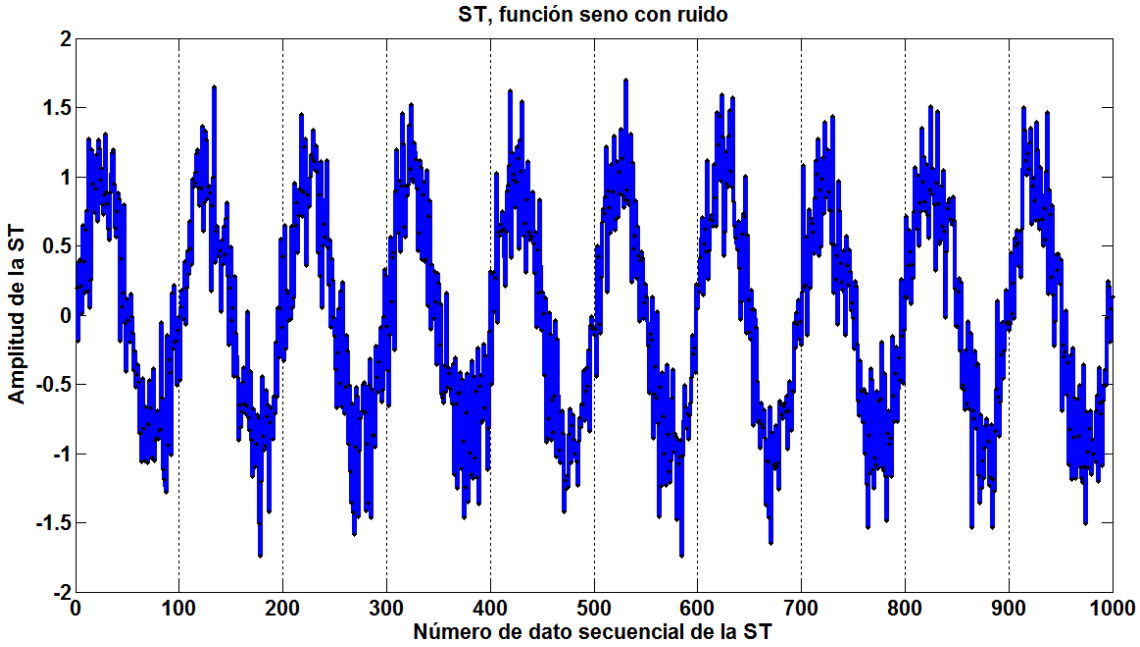


Figura 2.19 ST de la función seno con ruido.

Los parámetros que se emplean en la **RNA**, como la tasa de aprendizaje y el acelerador del **RPE** toman valores de $\boldsymbol{\varepsilon = 0.1}$ y $\boldsymbol{\alpha = 0.9}$ respectivamente. Los llamados umbrales se manejan constantes en toda la etapa de aprendizaje y toman valores de $\mathbf{0}$.

El número de iteraciones manejadas en diversos ejercicios fue desde $\mathbf{300}$ a $\mathbf{300\,000}$, esto para observar lo siguiente.

Debido a que la ST contempla ruido, el comportamiento de los errores de entrenamiento y prueba son diferentes, además de que en que cierta iteración estos errores empiezan a oscilar, al contrario de lo que ocurre con una función seno, siendo esta función periódica y que repite sus datos (valores) conforme avanza secuencialmente. En cambio, la función con ruido, aunque también contempla una periodicidad, no necesariamente repite sus

valores de un periodo a otro. Por ello la diferencia que se observa en el comportamiento de los errores, por lo que en este problema se ha de aplicar lo que se conoce como validación cruzada **(ver capacidad de generalización de la red).**

El resultado final que se adquiere es mediante la realización de ejecutar el algoritmo programado veinte veces, debido a que cada vez que se corre el programa y éste termina, se obtiene un resultado, al efectuarlo varias veces, el autómata arroja resultados distintos, siendo éstos parte de la solución del problema, por último se promedian estas veinte salidas obteniendo un resultado final. En estos aprendizajes las iteraciones se dejan en $\mathbf{3\,000}$, ya que si se aumenta el número de éstas llegamos al problema de generalización. Se mencionó con anterioridad, que se manejó hasta $\mathbf{300\,000}$ iteraciones, esto para mostrar lo que se conoce como sobre-ajuste (sobre-aprendizaje), el cual se analiza en el siguiente capítulo. Por tanto, las salidas que arroja el modelo neuronal son promediadas, obteniendo así un resultado final para este problema.

Reiterando que la ST de la función seno con ruido contiene $\mathbf{1000}$ datos, utilizando alrededor de $\mathbf{200}$ para ser entrenadas, el resto de la ST los hemos de comparar con el resultado obtenido para medir el error entre éstos. Además, se hace una comparación entre la ST de función seno con la ST de función seno con ruido, obteniendo un error para comparar éste con el error del resultado adquirido.

El error entre la ST de función seno respecto con la de ruido es de $\mathbf{3.4055}$, el error entre el resultado conseguido y la ST de la función seno con ruido es $\mathbf{3.4782}$, esta medición comprende $\mathbf{788}$ datos.

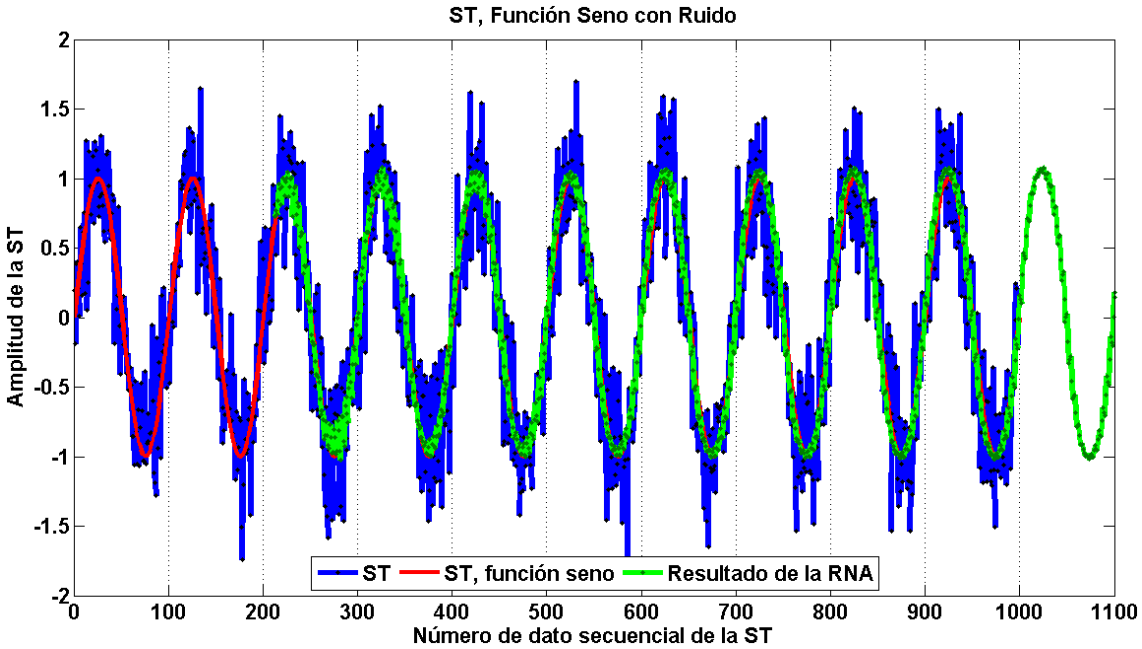


Figura 2.20 Resultado final para la ST función seno con ruido y comparación de la ST función seno.

Un promedio del error de aprendizaje y prueba al ejecutar el algoritmo programado varias veces es de $\mathbf{0.3612}$ y $\mathbf{0.3643}$, respectivamente. Estos errores localizados en un punto óptimo donde el error de generalización es mínimo.

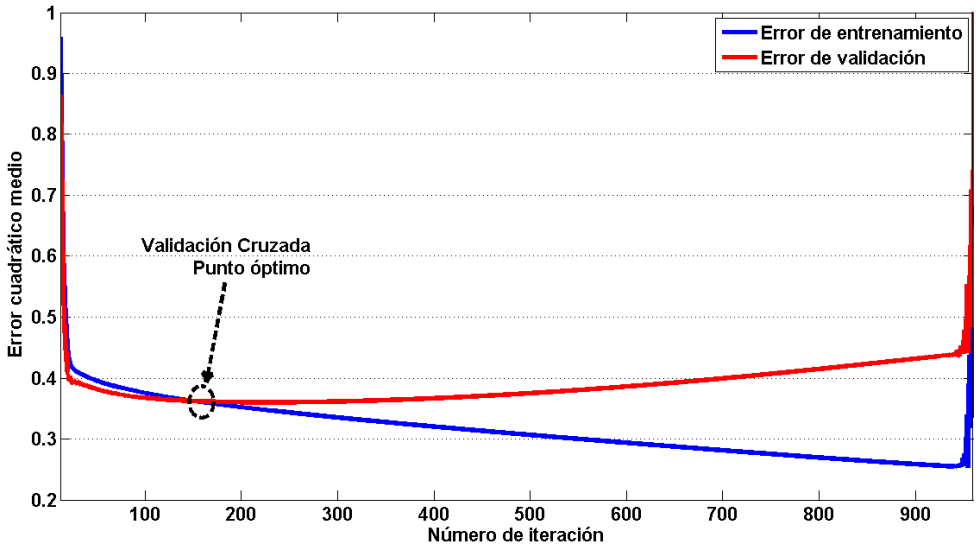


Figura 2.21 Error de generalización, validación cruzada.

Hasta ahora se han resuelto y expuesto ST que contemplan una periodicidad, siendo **no lineales**. Para el siguiente problema la cuestión cambia, ya que se maneja una ST muy diferente a estos anteriores, conteniendo más datos en la ST debido al comportamiento que tiene.

2.4.3 Función casi-periódica.

Una función casi-periódica la podemos construir mediante la siguiente función:

$$\boldsymbol{f(z)} = (\boldsymbol{cos(z)} + \boldsymbol{sin}(\sqrt[2]{\boldsymbol{2}} \cdot \boldsymbol{z}));$$

Observando que este tipo de funciones la podemos construir con funciones periódicas, incluyendo dentro de su argumento de la función un número irracional como es la raíz cuadrada de dos.

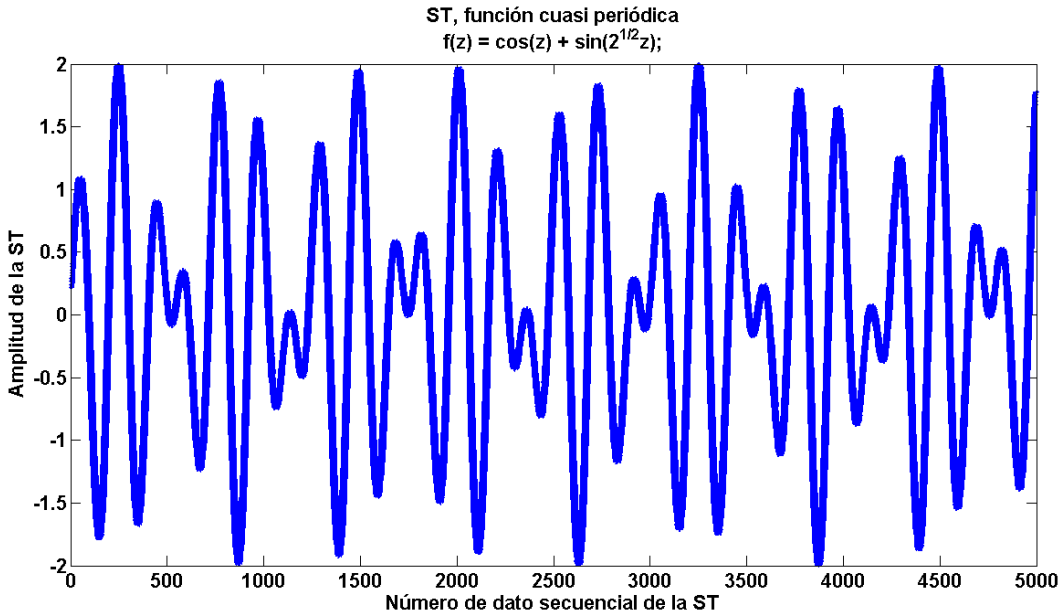


Figura 2.22 Función casi periódica, ST de 5000 datos.

La serie de tiempo de esta función se creó dentro del dominio$(-20 \cdot \pi, 20 \cdot \pi)$, adquiriendo $5\ 000$ puntos de muestreo entre el dominio dado **(ver figura 2.22).**

La estructura neuronal que se encontró y manejó fue de 7 nodos de entrada, 8 neuronas ocultas y una de salida, cada nodo o neurona contiene 1000 datos (patrones), aplicando un corrimiento de 45 (*delay*) para cada nodo de entrada.

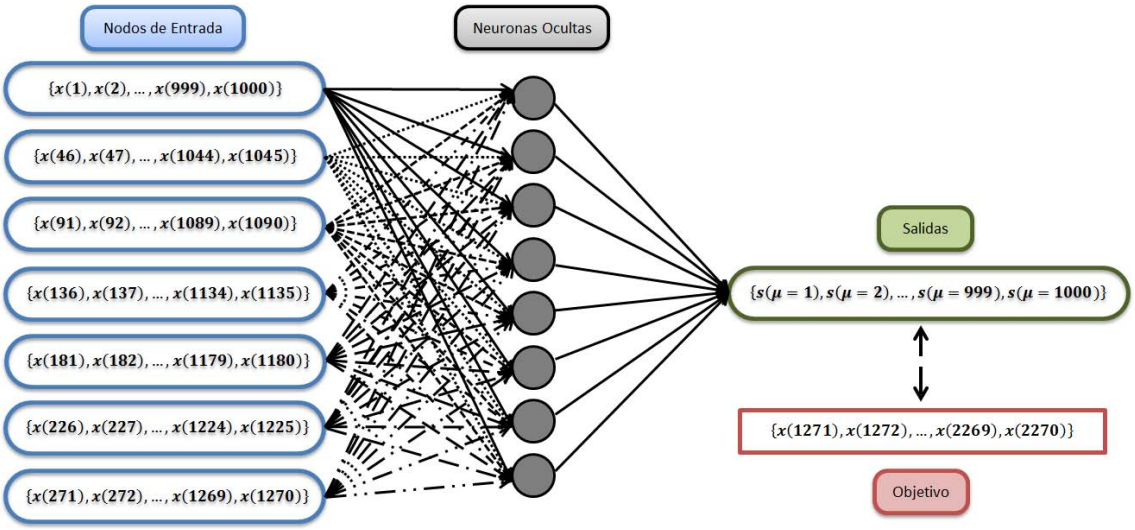


Figura 2.23 Estructura neuronal para la función casi periódica.

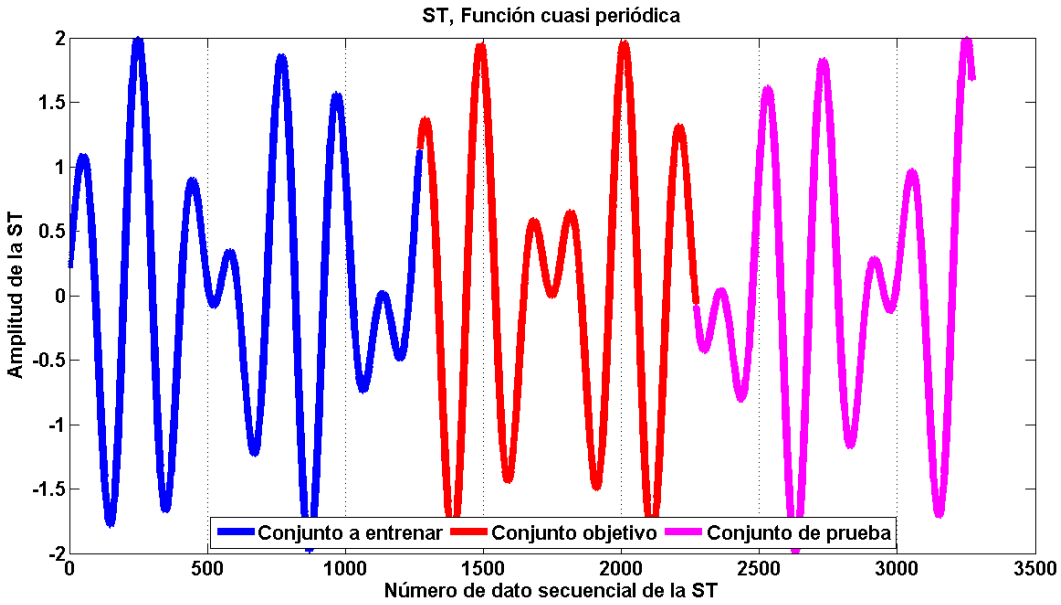


Figura 2.24 Conjunto de datos para la función casi periódica.

Los primeros 1270 datos de la serie se utilizan para ser entrenados (línea azul, $\{x(1), \dots, x(1270)\}$), los siguientes mil datos son utilizados para supervisar el entrenamiento (línea roja, $\{x(1271), \dots, x(2270)\}$), otros mil datos posterior al conjunto objetivo ($\{x(2271), \dots, x(3270)\}$) son para probar la **RNA**.

El error que se obtiene en el aprendizaje es de 0.001072, este error es la comparación entre el conjunto objetivo con el resultado obtenido de la **RNA**; el error de prueba es de 0.001887. Reiterando que la comparación, se obtiene con el error cuadrático medio.

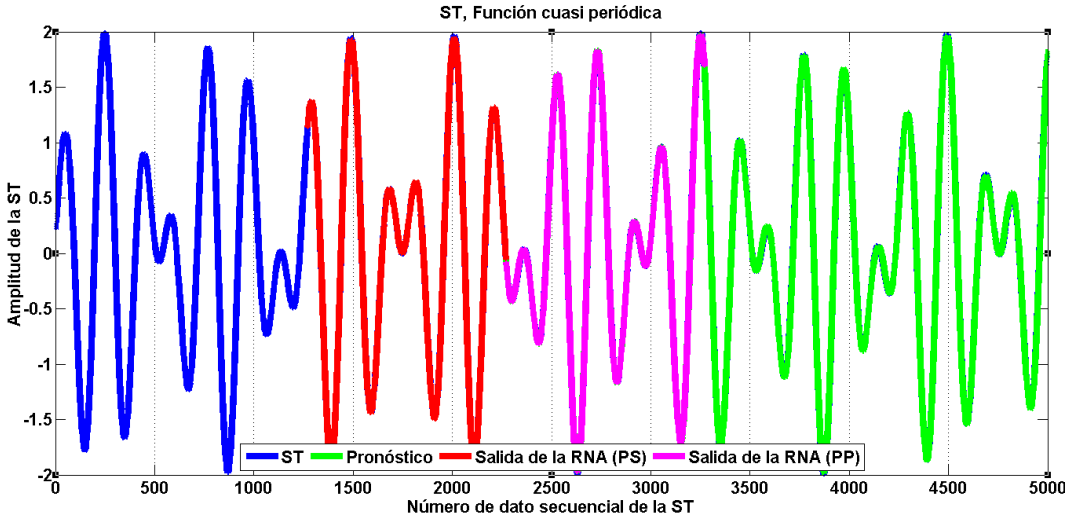


Figura 2.25 Resultados para la función casi periódica.

La ST original creada de la función $f(z) = \left(cos(z) + sin\left(\sqrt[2]{2} \cdot z\right)\right)$, es descrita por la línea de color azul. Las líneas de color rojo y purpura son las salidas (resultados) dadas por el autómata siendo éstas de los conjuntos objetivo y de prueba. La línea de color verde describe el pronóstico.

Tomando en cuenta que al realizar pronóstico se están dando 1000 valores futuros, cada vez que se realimenta la RNA obtenemos 1000 valores pronosticados. Observando que el resultado obtenido se asemeja a la original.

Otro ejemplo que se aplicó dentro de esta índole de funciones fue $\boldsymbol{f(z) = sin(z) + sin(\sqrt[2]{2} \cdot z)}$, manejando el mismo dominio e igual obteniendo 5000 puntos de muestreo.

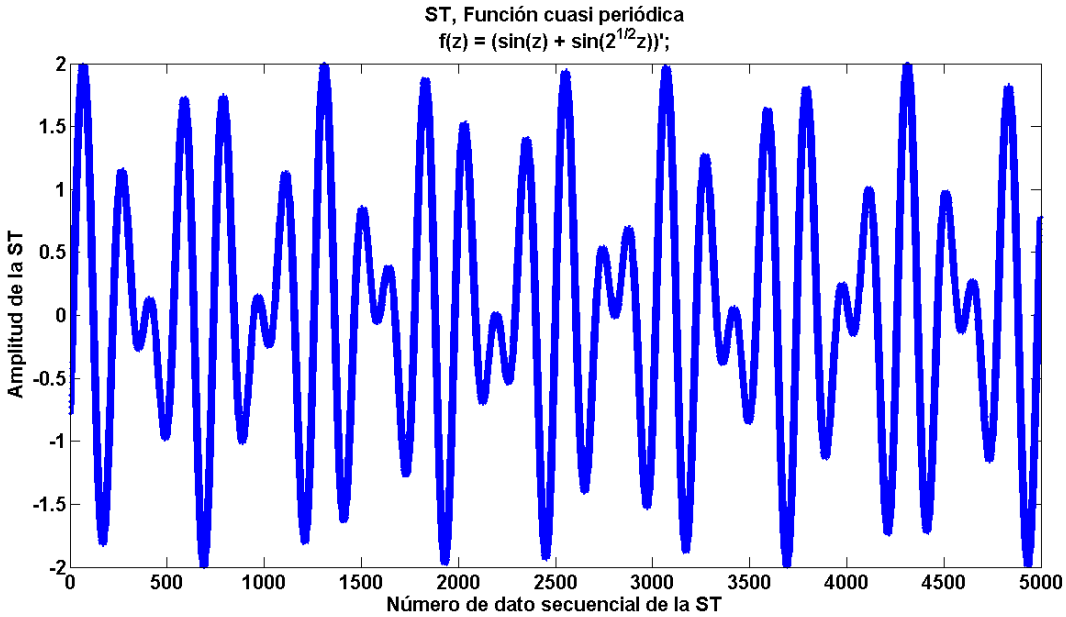


Figura 2.26 Segunda función casi periódica, ST de 5000 datos.

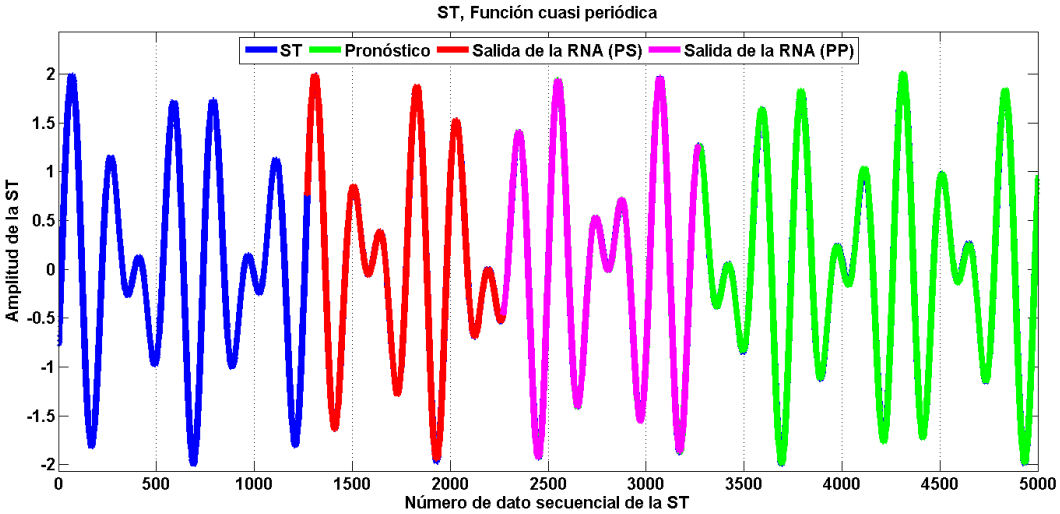


Figura 2.27 Resultados para la segunda función casi periódica.

Aplicando la misma estructura neuronal, el mismo número de patrones y de corrimiento, se obtiene en esta serie de tiempo el error de entrenamiento es 0.001136 y el de prueba 0.002678, siendo del mismo orden al ejercicio anterior.

2.4.4 Lorenz.

Edward Lorenz (1917 – 2008) fue un matemático y meteorólogo del *MIT* (*Massachusetts Institute of Technology*, por sus siglas en inglés), pionero en el desarrollo de la teoría del caos y quien introdujo el concepto de atractores extraños y acuñó el término efecto mariposa. En 1963, publicó su trabajo titulado 'flujo determinístico no periódico', el cual detalla el comportamiento de un modelo matemático simplificado de ecuaciones diferenciales ordinarias, que representa el funcionamiento de la atmósfera. Las ecuaciones de *Lorenz* son:

$$\dot{x} = -\sigma(x - y)$$

$$\dot{y} = -xz + rx - y$$

$$\dot{z} = xy - bz$$

Es de notar que este sistema tiene dos términos **no lineales**: 'xy' y 'xz'. Siendo 'x' el gradiente (variación) de giro convectivo, 'y' la variación horizontal de la temperatura y 'z' la variación vertical de temperatura, siendo también las respectivas soluciones de las ecuaciones del modelo de *Lorenz*.

Las constantes que maneja este modelo de *Lorenz* son $\boldsymbol{\sigma}$, $\boldsymbol{r}$ y $\boldsymbol{b}$, respectivamente son: número de *Prandtl*, el cual relaciona las pérdidas de energía dentro del fluido a la conducción térmica; el número de *Rayleigh*, medida adimensional de la diferencia de temperatura entre las placas; razón entre la altura vertical de la capa de fluido y la extensión horizontal de los vórtices convectivos.

Las soluciones de las ecuaciones del modelo de *Lorenz* pueden representar ST, tiendo para cada variable $\boldsymbol{x}$, $\boldsymbol{y}$ y $\boldsymbol{z}$ una ST. Con dichas soluciones, se puede construir el famoso atractor extraño de *Lorenz*, conocido como alas de mariposa **(ver tabla 2.4)**, contando con tres ST, mostradas en las figuras siguientes.

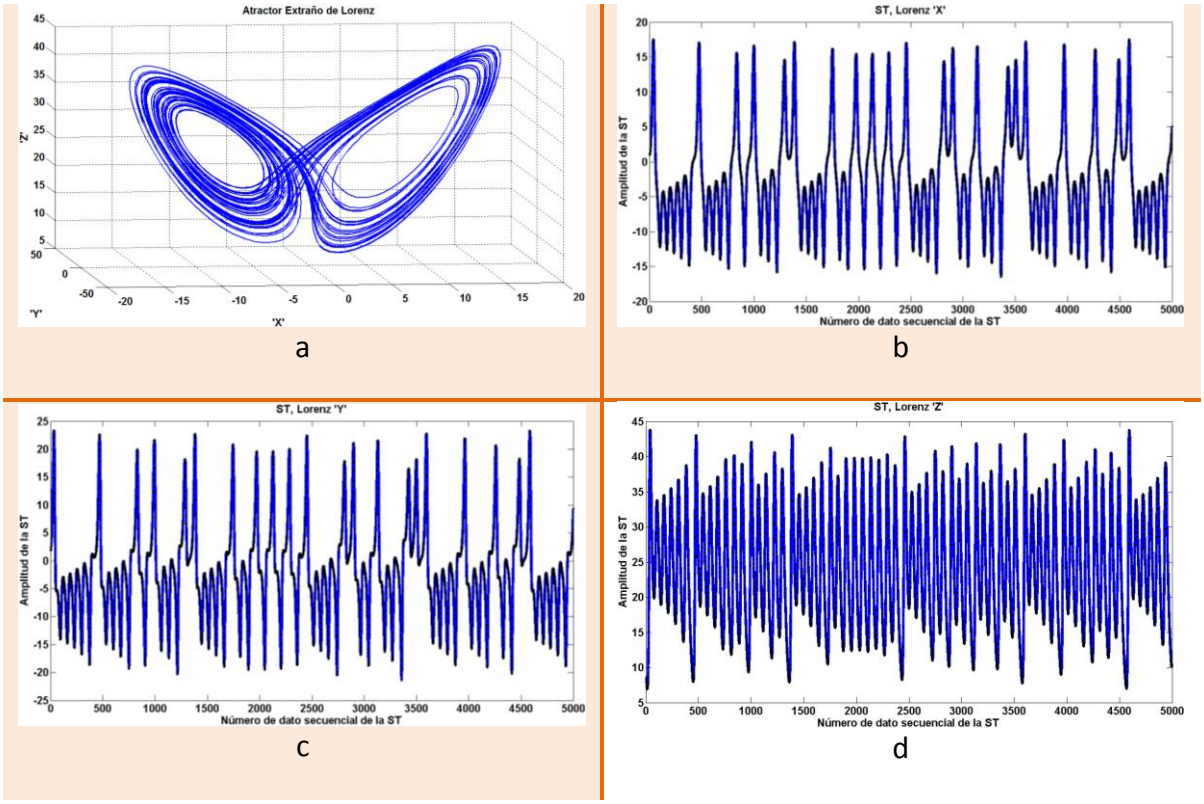


Figura 2.28 Modelo de Lorenz, atractor (a) y ST de las variables '$\boldsymbol{x}$', '$\boldsymbol{y}$' y '$\boldsymbol{z}$' (b, c, d, respectivamente).

Las ST trabajadas de este modelo, contienen $5\,000$ datos cada una, realizando experimentos para cada variable.

Trabajando primero con la variable *'z'* (variación vertical de temperatura), hemos de observar que su comportamiento de la ST es más suave **(ver figura 2.28d)**, teniendo mínimos y máximos menos notorios a los que presentan las ST de las variables 'x' y **'y'** de *Lorenz*.

Para todas las ST de *Lorenz* que se ingresaron a la **RNA**, se trabajó de modo diferente en comparación al de los dos ejercicios anteriores, teniendo sobrepuestos a los conjuntos de entrada, objetivo y de prueba **(ver Figura 2.29)**, haciendo referencia al ejemplo cuatro descrito al inicio de este capítulo **(ver Figura 2.10)**. Se ingresaron de este modo debido a la cantidad de datos manejados, además de su comportamiento variado, queriendo que el conjunto a entrenar contenga suficiente historia para la red. Si se realizaba de las formas como el seno y seno con ruido, no se lograba tener un buen entrenamiento además de no obtener un error mínimo de generalización.

La arquitectura generada fue de $\mathbf{10}$ nodos de entrada, $\mathbf{9}$ neuronas ocultas y una de salida. Los parámetros de la **RNA** fueron $\boldsymbol{\varepsilon = 0.1}$ (tasa de aprendizaje) y $\boldsymbol{\alpha = 0.9}$ (acelerador del **RPE)**. Los umbrales fueron constantes en todo el entrenamiento tomando valor de $\mathbf{0}$. Cada nodo contempla $1\,000$ datos, el número de iteraciones máximos fueron de $300\,000$. El *delay* o el corrimiento que se aplicó en la ST para cada nodo de entrada fue de 15, teniendo entonces que este número será el número de valores futuros obtenidos cada vez que se realimenta la **RNA**.

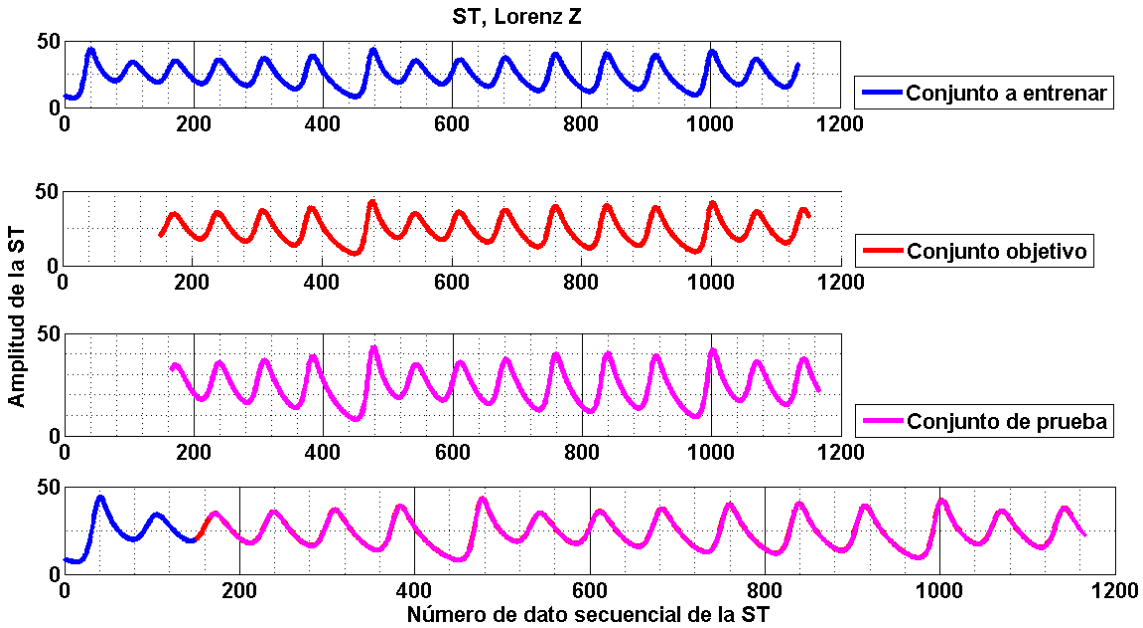


Figura 2.29 Conjuntos de entrada, objetivo y prueba para la ST de Lorenz de la variable 'z'.

Los errores obtenidos tanto para el de entrenamiento y de prueba fueron idénticos 0.0036, teniendo que su comportamiento del error fueron semejantes.

Los resultados para el pronóstico fueron peculiares, debido que en este problema hemos de encontrar un límite a la predictibilidad. Al obtener los pesos con el error mínimo de entrenamiento, parámetros que constituyen el modelo neuronal sobre dicha ST.

Al realizar el pronóstico, hemos de tener 15 valores futuros cada vez que realimentamos a la **RNA**; al realimentar el autómata se usan las salidas que se obtienen, dando como resultado lo que se muestra en la figura 2.30. Teniendo en cuenta que el número de datos empleados para el entrenamiento fueron 1160, el resto de la ST original se usa para comparar con respecto a las salidas de la **RNA**.

Podemos observar que nuestro pronóstico empieza muy bien, siguiendo el comportamiento de la ST; pero, al ir extendiendo esta predicción hemos de notar que ésta

empieza a desvariar, aunque sigue mostrando la misma tendencia que la original; más sin en cambio llega un punto donde el pronóstico se comporta muy diferente a la ST, mostrando un resultado de tipo ruidoso y cuya predicción es obsoleta.

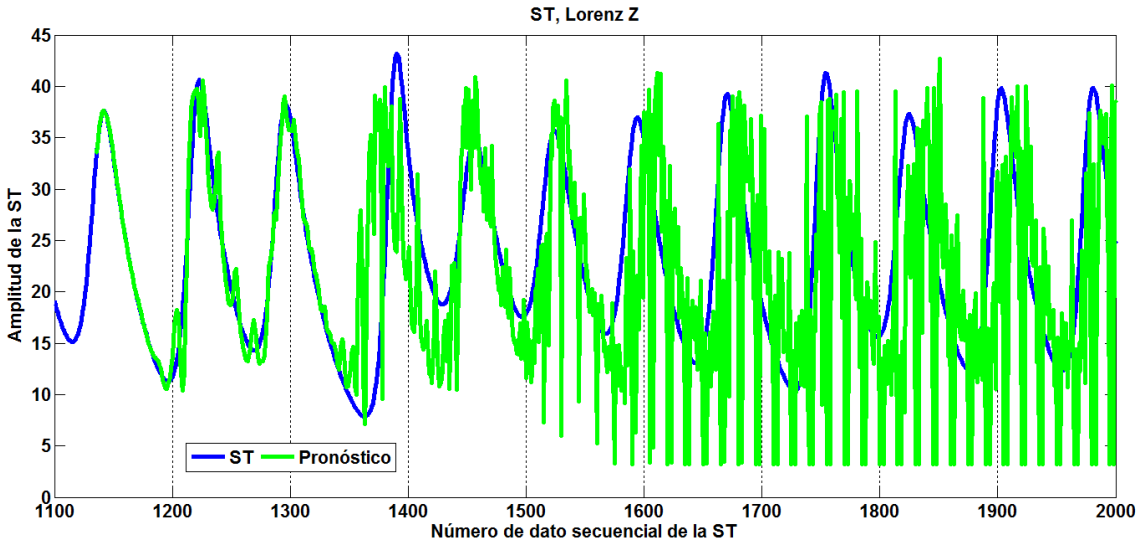


Figura 2.30 Pronóstico sobre la ST de Lorenz de la variable 'z'.

Recalcando que este pronóstico que se efectuó, fue realimentando a la **RNA** con las salidas de esta misma. Pero, hemos de observar que hasta cierto punto, nuestro pronóstico es aceptable, teniendo entonces como límite de predictibilidad el mismo *delay* manejado sobre la ST.

Ahora bien, para el usuario, pronosticador o tomador de decisiones, comúnmente han de ajustar o rectificar el resultado obtenido con lo real, por lo que en vez de usar las salidas obtenidas de la red para realimentarla nuevamente, se corrigen usando la ST original y al realimentar la red con éstos, obtenemos los siguientes valores futuros. Efectuando esto último mencionado, mostrando los resultados en las figuras 2.31 y 2.32, recordando que al efectuar el pronóstico, ya no se hace ningún entrenamiento de la red.

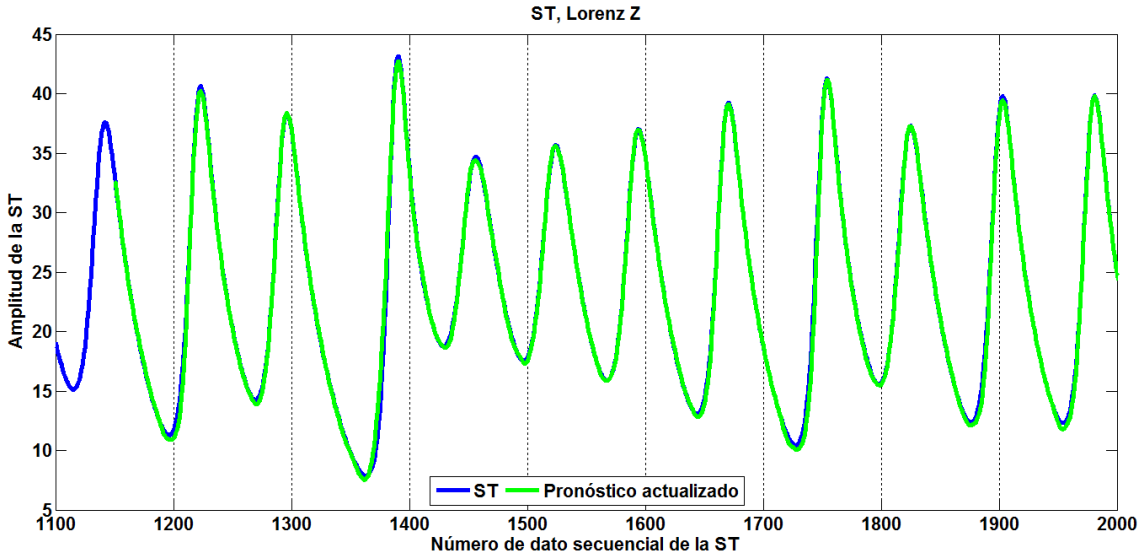


*Figura 2.31 Pronóstico variable '**z**', actualizado las salidas antes de realimentar a la **RNA**.*

Comparando los resultados obtenidos del modelo neuronal con la ST, de 3840 datos, fue de 0.27581, observando que el comportamiento de lo pronosticado con la ST, es idéntico.

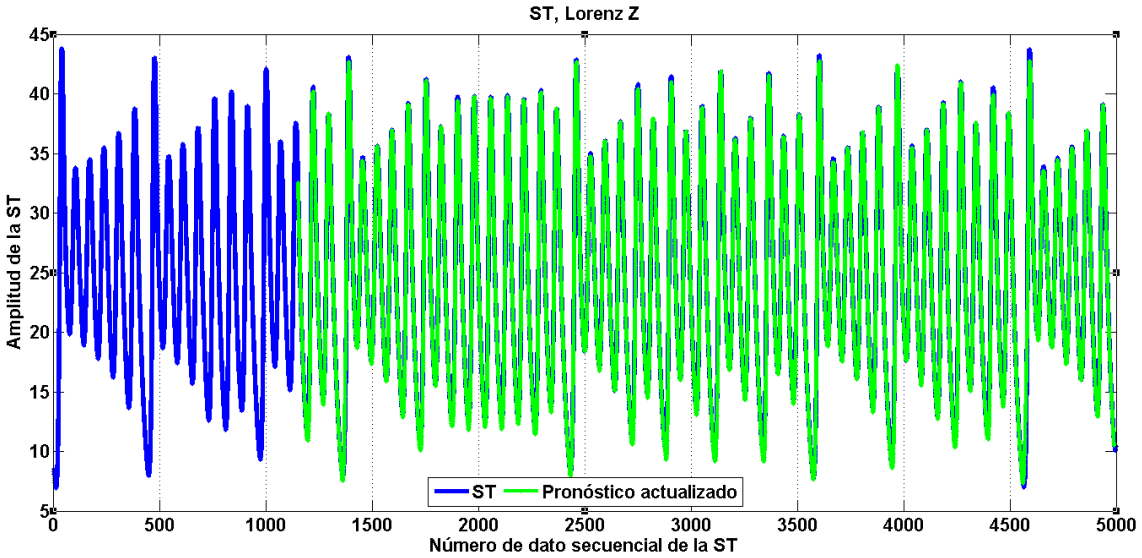


*Figura 2.32 Pronóstico variable '**z**', actualizado las salidas antes de realimentar a la **RNA**.*

Al analizar y ejecutar diferentes modelos neuronales, el valor del *delay* es demasiado grande, ya que estamos dando por hecho de que éste representa el límite a la

predictibilidad; por lo que se hace una reducción de este corrimiento, encontrando además una diferente arquitectura.

Esta reducción del *delay* y la diferente arquitectura, se encontró al experimentar con las dos siguientes ST de las variables '$\boldsymbol{x}$' y '$\boldsymbol{y}$' del sistema de *Lorenz*, ya que se ha de implementar la misma arquitectura para cada una de éstas.

Teniendo en cuenta que, al efectuar el pronóstico se corregirán las salidas antes de realimentar a la **RNA,** para que nuevamente se obtengan los siguientes valores futuros. El *delay* que ahora se manejó fue de $\mathbf{5}$ y la nueva arquitectura es de $\mathbf{2}$ nodos de entrada con $\mathbf{25}$ neuronas ocultas y una de salida, teniendo entonces un número de datos a entrenar de 1005, menores al del anterior, respetando los 1000 datos en cada nodo. Los parámetros que se manejan en la red se mantienen iguales que al ejercicio precedente, lo único que se modificó fue el número de corrimiento y la arquitectura.

Por lo tanto, para la ST de '$\boldsymbol{z}$' se obtiene que el error de entrenamiento como de prueba es de 0.01387 y 0.01368, respectivamente, el error entre las salidas obtenidas del autómata y la ST es de 0.06077, haciendo la comparación de 3990 datos, este último error es menor comparado al anterior, recalcando que esta comparación contiene mayor número de datos. Observando que estos cambios, mejoraron el pronóstico.

Con esta estructura de modelo neuronal, se toma para las siguientes dos ST del sistema de *Lorenz*, '$\boldsymbol{x}$' y '$\boldsymbol{y}$', teniendo en cuenta que su comportamiento de éstas son diferentes respecto de **'$\boldsymbol{z}$'**, presentando una mayor dificultad de aprendizaje para la **RNA** por los máximos y mínimos que contienen dichas ST.

En resumen, para todas las variables del sistema de Lorenz, se manejó una arquitectura neuronal de $\mathbf{2}$ nodos de entrada, $\mathbf{25}$ neuronas ocultas y un nodo en la capa de salida, siendo esta estructura un **PMC**. Los parámetros de la **RNA** fueron $\boldsymbol{\varepsilon = 0.1}$ (tasa de aprendizaje) y $\boldsymbol{\alpha = 0.9}$ (acelerador del **RPE)**. Los umbrales fueron constantes en todo el entrenamiento tomando valor de $\mathbf{0}$. Cada nodo contempla $1\,000$ datos, el número de iteraciones máximos fueron de $300\,000$.

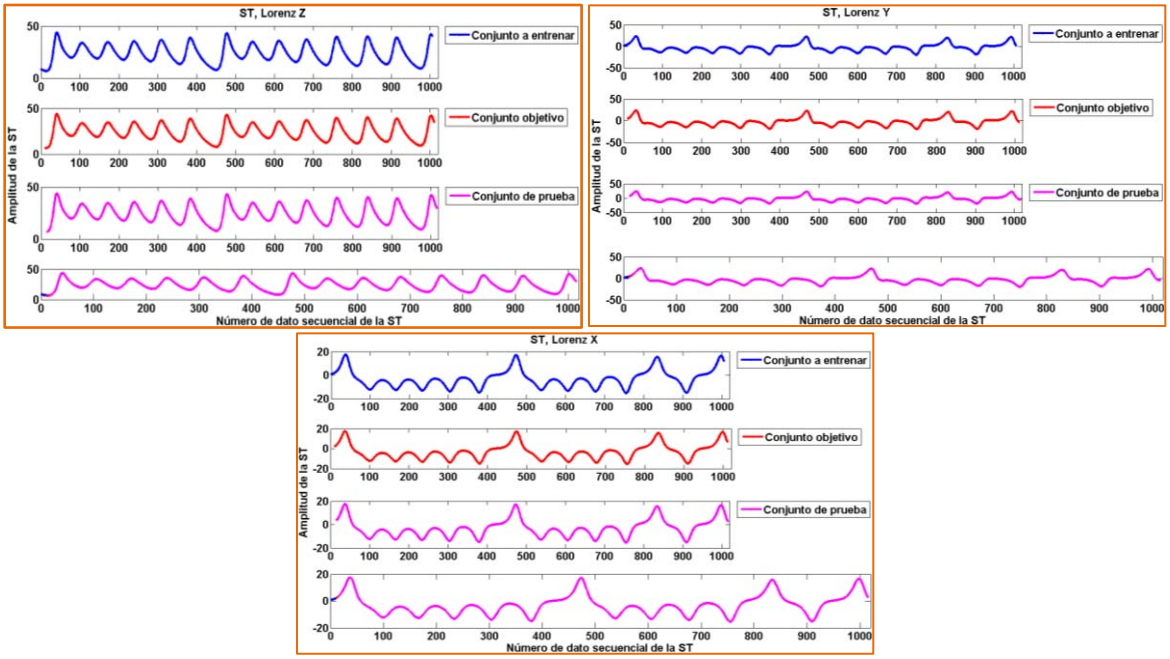


Figura 2.33 Conjuntos de entrada, objetivo y prueba para la ST de Lorenz.

El error que presenta en el pronóstico de la variable *'$\boldsymbol{x}$'* (gradiente de giro convectivo) es de 0.48253 y de la variable *'$\boldsymbol{y}$'* (variación horizontal de la temperatura) es de 1.16315.

Presentando errores de entrenamiento y de prueba para $\boldsymbol{x}$, respectivamente, de 0.07551 y 0.07538, para $\boldsymbol{y}$ se tienen 0.18134 y 0.18367.

Mostrando en la figura 2.34 los resultados para las ST del sistema de *Lorenz*.

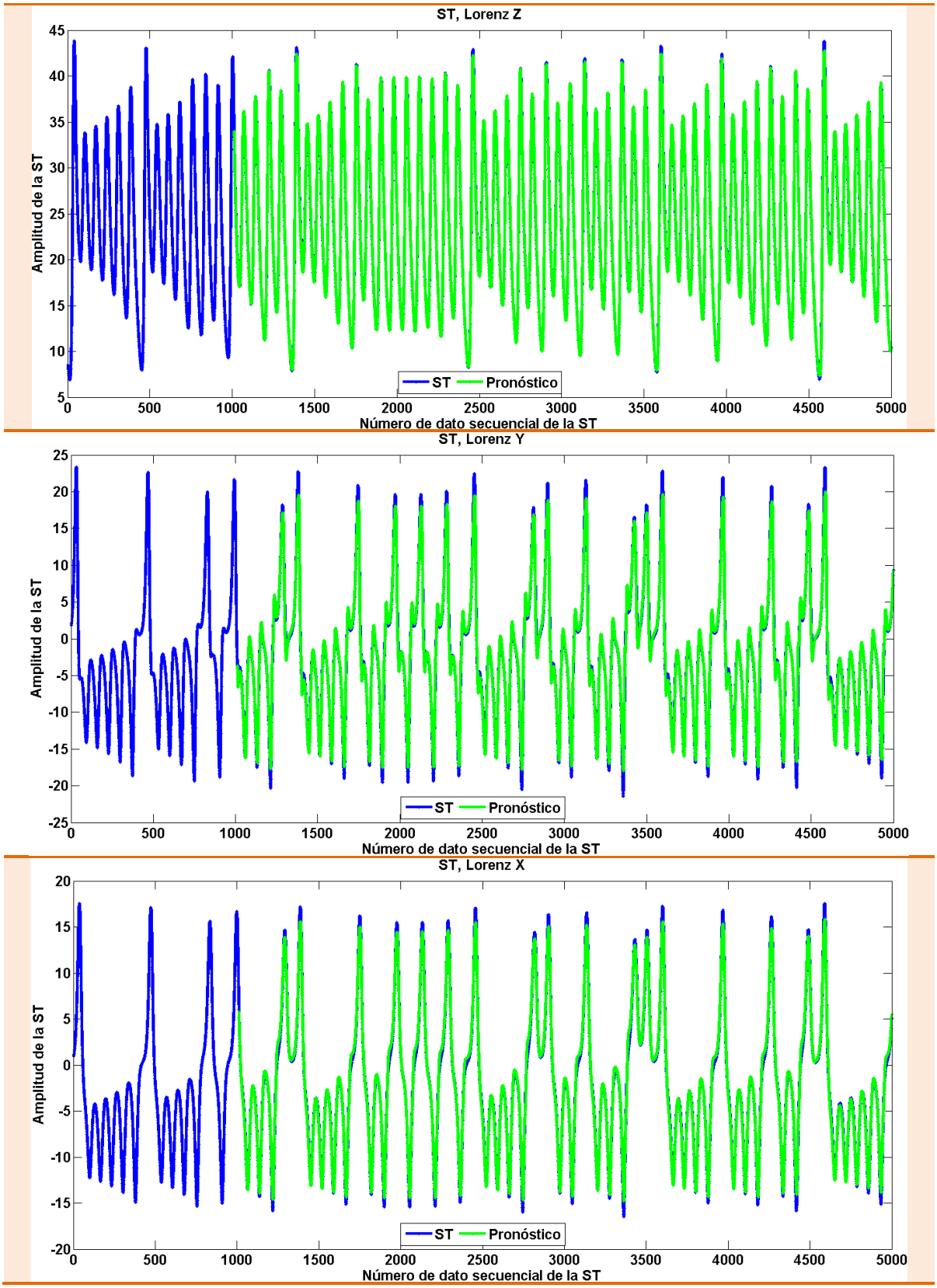


Figura 2.34 Pronósticos de las ST del sistema de Lorenz.
*Actualizado las salidas antes de realimentar a la **RNA**.*

Con las ST pronosticadas, se puede reconstruir el atractor de Lorenz, considerando que los datos pronosticados en total son de 3990 puntos.

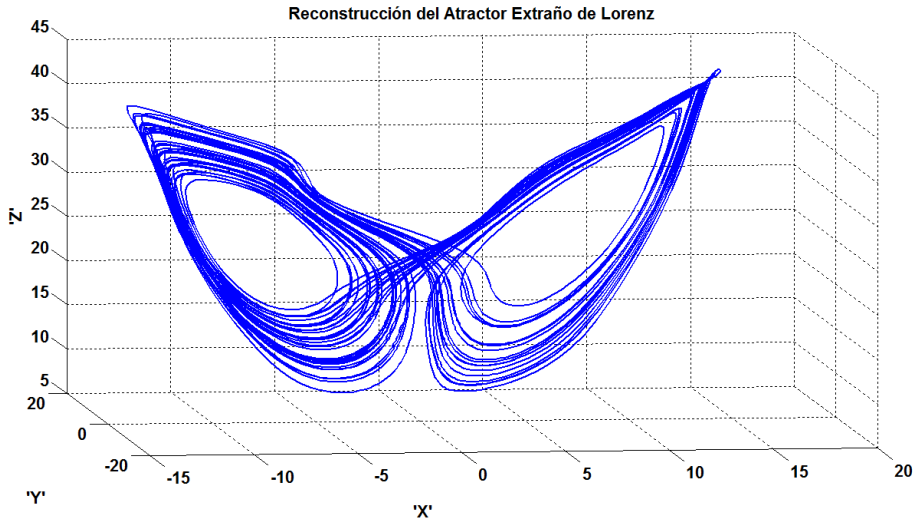


Figura 2.35 Reconstrucción del atractor de Lorenz, con la ST pronosticadas.

Apreciando que el atractor reconstruido es similar al original, mostrando las famosas alas de mariposa del sistema de *Lorenz*.

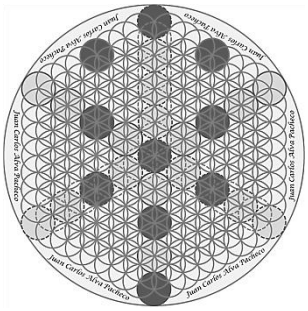
Con todo lo que se trabajó para todas y cada una de las ST en todo este proyecto presente, se puede dar a lugar a las conclusiones generales, expresadas en el último capítulo, previo a un análisis general de cada problema que se han mostrado en los apartados de este capítulo.

CAPÍTULO 3

ANÁLISIS DE RESULTADOS Y CONCLUSIONES GENERALES

La ciencia se compone de errores, que a su vez, son los pasos hacia la verdad.

Julio Verne



Los experimentos ejecutados, modelos obtenidos y realización de todo el proceso de los algoritmos programados; se efectuaron en una computadora personal laptop *DELL*, modelo *Inspiron 3520*, procesador *Intel(R) Inside Core(TM) i3*, memoria *RAM 4.00 GB*, sistema operativo de *64 bits*, *Windows 7*.

Los algoritmos para el modelo de la **RNA** y todas las funciones que se toman para realizar el procesado y pronóstico en las ST, fueron programadas sobre el *software Matlab* (*Mathworks Matlab R2012a*), siendo éste un lenguaje de computación técnico de alto nivel y un entorno interactivo para el desarrollo de algoritmos, visualización de datos, análisis de datos y cálculo numérico. Se trabajó en este software debido a su entorno de desarrollo para la gestión de código, archivos y datos; por las interfaces y funciones gráficas bidimensionales para la visualización de datos y sus herramientas para hacerlas personalizadas.

Respecto a algunos parámetros que se manejan en el modelo de la **RNA**, se hace mención en los capítulos anteriores de los llamados umbrales (*bias*), éstos se manejaron como constantes, esto es, que no sufrieran un cambio como los pesos. Se toman de esta manera para no alterar el comportamiento de los errores, ya que al estar entrenando se modifican los pesos, y si se quisiera, también los umbrales, teniendo más trabajo el autómata al estar ajustando estos dos parámetros, por ello se fijan, ayudando al proceso de aprendizaje en que sea más ágil y que el error converja en un mínimo. En la literatura, los umbrales han de tomar el valor de $\mathbf{0}$, $\mathbf{1}$ o $-\mathbf{1}$, probando con cada uno de éstos para la optimización del modelo.

El *delay*, este desfasamiento implementado al introducir los datos al modelo neuronal, fue para dar información a la **RNA** sobre la ST, realizando un corrimiento para cada nodo de entrada. Haciendo una mención sobre la teoría del caos y reconstrucción del atractor, es que este *delay* recrea una representación en el espacio de fase del sistema dinámico estudiado a partir de una serie escalar temporal [Peña Maciel, 2012].

Este último parámetro se obtiene con ayuda de un software conocido como ***VRA*** (*Visual Recurrence Analysis*, por sus siglas en inglés), el cual es un software escrito en *C++* por Eugene Konov, tiene como función efectuar el análisis topológico, cualitativo y cuantitativo, además de la predicción no paramétrica de ST **no lineales** y caóticas; fácil de usar ya que posee una interfaz gráfica.

Recordando que el proceso que realiza la **RNA** son operaciones matriciales, teniendo diferentes matrices en cada capa para un modelo de **PMC**; matrices como los datos que son ingresados al modelo y matrices de los pesos. Se menciona esto debido al tiempo de procesado, siendo la duración que le toma al algoritmo programado terminar su tarea. Esto va dependiendo desde el número de iteraciones (número de veces que es ajustado/modificado el peso), al número de nodos o neuronas en cada capa, además de la cantidad de datos que contiene cada nodo (el número de patrones). Al estar experimentando se toman los tiempos de procesado, considerando que un mayor número de neuronas, ya sea en la capa de entrada y/u oculta, aumenta el tiempo de ejecución, este aumento también es debido al número de iteraciones y de igual forma a la cantidad de patrones utilizados. Por lo que entre más iteraciones, más datos y más neuronas, hace que el tiempo de procesado se prolongue.

3.1 Función seno.

Siendo la función seno una función **no lineal**, la **RNA** aprendió su comportamiento, lo cual muestra que el método es adecuado para este tipo de ST, no obstante, esta función seno es periódica y su comportamiento es suave a comparación de una con ruido. Considerando que esta función fue asequible para el modelo en cuestión.

Sobre el proceso realizado para obtener el resultado, mostrado en el anterior capítulo a este, se realizaron distintos experimentos, variando la cantidad de nodos de entrada y neuronas ocultas. Para esta función siempre se consideraron $\mathbf{100}$ patrones para cada neurona, siendo este valor la periodicidad de la ST. Resaltando que la cantidad de nodos en capa de entrada y oculta para la estructura de la **RNA**, fue dada a prueba y error, realizando varios experimentos y repitiéndolos por lo menos diez veces, esto para estimar cual fuese más apto para realizar un buen aprendizaje y con ello realizar pronóstico sobre la ST.

El tiempo que le llevó al algoritmo programado en terminar $300\ 000$ iteraciones, con $\mathbf{29}$ nodos de entrada, $\mathbf{11}$ en la capa oculta y $\mathbf{1}$ en la salida, con una cantidad de $\mathbf{100}$ datos en cada nodo, fue aproximadamente de $4\ hrs$. Esta arquitectura logró simular la función seno con un error de un orden a la millonésima potencia $\mathbf{5.96056}e^{-07}$ (en su etapa de entrenamiento) y al realizar el pronóstico sigue el comportamiento de la función seno, logrando simular muy bien dicha función.

Concluyendo, que funciones bien portadas, como es la función seno, la **RNA** logra asemejar dicha función, haciendo notar que el número de datos ingresados para ser

entrenados fue alrededor de una tercera parte de mil datos que son con los que se contaban, por lo que nuestro modelo es óptimo para este tipo de ST.

Los pesos óptimos, que son cuando la **RNA** obtuvo un mínimo error en el entrenamiento, solo han de funcionar exclusivamente para esta ST y con la arquitectura con la que se trabajó, es decir, si ingresáramos otra ST, por ejemplo otra función seno pero con una discretización o periodicidad distinta (valores que no sean iguales pero que representen la función seno) no forzosamente simularía dicha ST como la que se obtuvo al principio, podría asemejarse, pero se tendría una diferencia entre la nueva ST respecto a las salidas de la **RNA**.

Por tanto, los pesos adquiridos para simular la ST son únicos para cada una, no obstante, se podría utilizar dicha arquitectura y realizar el entrenamiento para así obtener sus propios pesos óptimos y reproducir la nueva ST. Esto último mencionado es aplicado para el siguiente ejercicio.

3.2 Función seno con ruido.

Se generó la ST de la función seno con ruido, con la misma ST de la función seno del ejercicio anterior, aplicándole ruido.

Se tomó la misma arquitectura del ejercicio anterior, que es de $\mathbf{29}$, $\mathbf{11}$ y $\mathbf{1}$ nodos de entrada, ocultos y salida respectivamente. Realizando su entrenamiento para esta ST de la función seno con ruido, mostrando los resultados en el anterior capítulo, donde los pesos

óptimos son los que se obtienen en la iteración donde se intersecan los errores de entrenamiento y prueba, esto se conoce como validación cruzada **(ver capacidad de generalización de la red)**. Se considera que estos pesos son óptimos, ya que si se tomaran los pesos en la iteración del error mínimo de entrenamiento, estos pesos solo funcionarían muy bien para el conjunto objetivo que es el que supervisa el aprendizaje de la **RNA**, asemejando muy bien su comportamiento para este conjunto, pero estos pesos no funcionan al realizar pronóstico, ya que aprendió el ruido y no la tendencia (forma) que sigue la función, a esto se le conoce como sobre-aprendizaje.

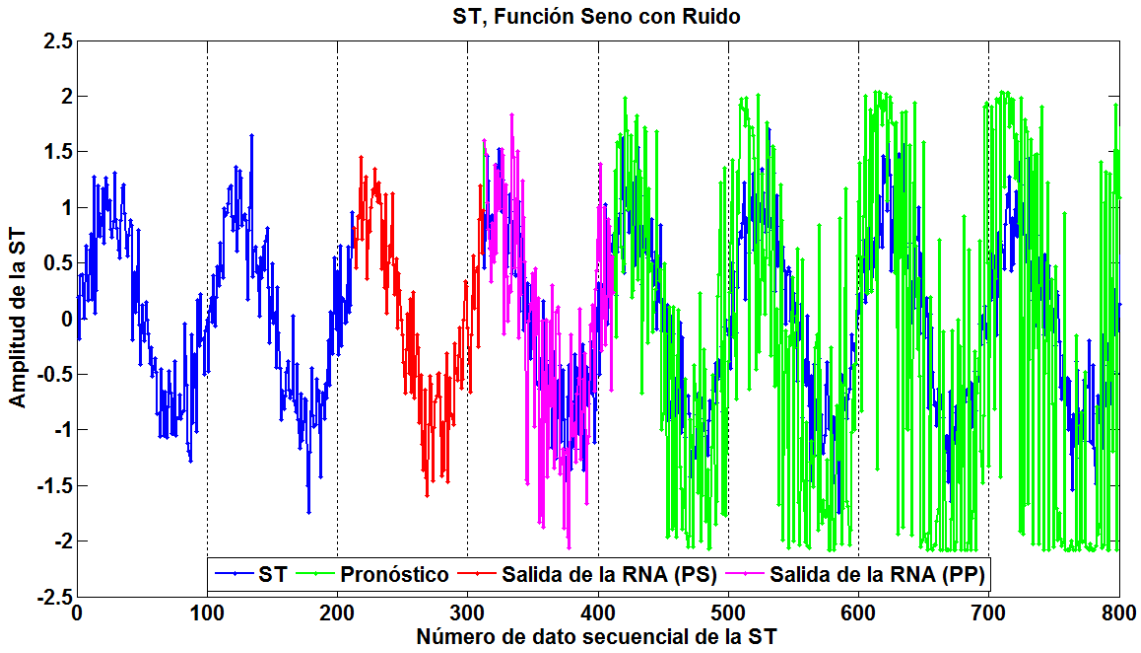


*Figura 3.1 Salida de la **RNA** con sobre-aprendizaje.*

Hemos de observar que el modelo neuronal también puede aprender el ruido, en este caso la parte que es supervisada, pero al realizar pronóstico, éste no es servible, ya que pierde la forma que contiene la ST original.

El tiempo de procesado de este problema, fue variado, ya que se experimentó con números de iteraciones de 300, $3\,000$ y $300\,000$; el primero fue porque frecuentemente se encontraban los errores de generalización óptimos antes de este número de iteración; el segundo para contemplar que el error de entrenamiento fuera disminuyendo, además de encontrar el comienzo de oscilaciones de este error y el de prueba; el de $300\,000$ iteraciones fue para observar lo que realmente puede conllevar a que la **RNA** tenga sobre-aprendizaje. Teniendo $12\ seg$, $2\ min$ y $4\ hrs$, como tiempos de procesado y reiterando que entre más número de iteraciones más el tiempo de procesado, siendo éste lo que tarda la **RNA** en aprender.

Se experimentó con pesos dados en la iteración del error mínimo de prueba, éstos se localizaban antes de que comenzaran a incrementar su error y antes de que el error de entrenamiento disminuyera, o bien, antes de la validación cruzada; simplemente se experimentó esto para observar los resultados que logra arrojar la **RNA**, siendo estos no benéficos para el pronóstico, por lo que se concluye que los pesos óptimos son alcanzados cuando hay una menor distancia entre el error de entrenamiento y de prueba, tal como se explica en el apartado capacidad de generalización de la red del capítulo uno.

El comportamiento del error, tanto de aprendizaje como de prueba, llegan a una iteración donde empiezan a oscilar, esto se debe a que la ST entrenada contiene ruido además de considerar que en tal iteración empieza el sobre ajuste. Al seguir iterando, no obstante la oscilación, el error de entrenamiento tiende a bajar.

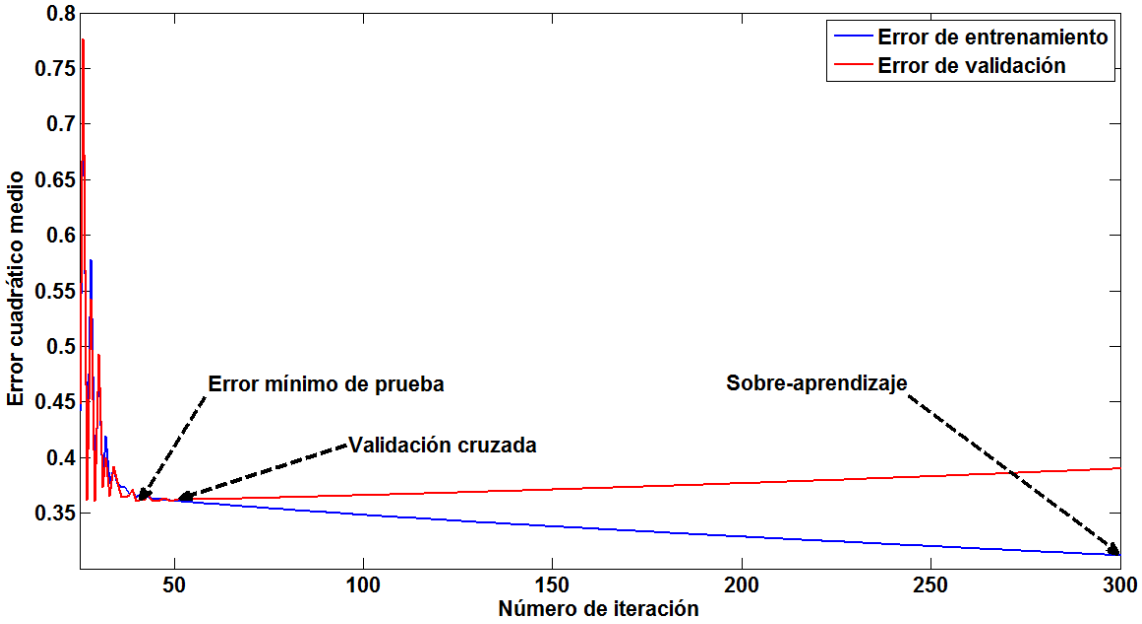


Figura 3.2 Comportamiento de errores de la ST función seno con ruido.

Lo que cambia para este problema con respecto al anterior fue el delay, para la función seno se consideró un *delay* de 3 y 4 para la función con ruido, estos valores son los que arroja el **VRA**, considerando que estos *delays* son los aptos para cada ST en cuestión, sin embargo, también se experimentó con el *delay* de 3 sobre la ST con ruido, pero al observar los errores adquiridos arroja un mejor resultado el de 4. Concluyendo que cada ST tendrá su propio *delay*, independientemente de la arquitectura que se es manejado en la **RNA**.

Como bien se menciona la ST con ruido fue generada con la ST anterior, por lo que se hizo una comparación y así analizar el error de ésta con lo que se obtuvo con la **RNA**, los datos comparados son los no utilizados para entrenar, que son $\mathbf{788}$ datos. Los errores son de $\mathbf{3.4055}$, error entre la ST de función seno respecto con la de ruido y $\mathbf{3.4782}$**,** error entre el resultado conseguido y la ST de la función seno con ruido. Observamos que el error son muy cercanos uno del otro, siendo bueno para este problema, ya que lo que se desea

obtener es que la autómata aprenda la forma, la tendencia que tiene la ST, además de que la función base de la ST con ruido es la función seno, llegando a ser similar el resultado a esta función.

Lo que logra la **RNA** es aprender la forma de la ST original, no tanto el ruido que contiene. Esto sería conveniente para los tomadores de decisiones, ya que muchas veces no se desea pronosticar valores exactos, sino que este dentro de un rango asequible para proporcionar resoluciones.

En la literatura, hacen mencionar que las **RNA** también hacen función del filtrado de señales, probándolo en este ejercicio con los resultados obtenidos; no obstante, siendo minuciosos, hemos de encontrar algo de ruido en el resultado final, por ello la diferencia de errores precedentemente mencionados, aunque se podría mejor el algoritmo y así obtener un ST exacta al seno que es lo apetecible que alcanzara el modelo neuronal, ya que esta respuesta sería la señal filtrada.

El pronóstico manejado en estos dos problemas, han sido a número de patrones, es decir, cada vez que realimentamos la **RNA** obtenemos $\mathbf{100}$ valores futuros, que es el número de patrones contenidos en cada nodo. También podemos hacer pronóstico a número de *delay*, teniendo tal número de valores pronosticados cada vez que se realimenta el autómata, pero al analizar los resultados bajo este último mencionado, obtenemos los mismos errores, por lo que no hay una diferencia o afectación en la forma de hacer predicción.

3.4 Función casi-periódica

Al entrenar y pronosticar funciones periódicas, se ha observado que la **RNA** lo realiza dentro de las expectativas favorables, para el caso de presentar funciones casi-periódicas es un ejercicio que se debe considerar, puesto que hay ST que presentan este tipo de comportamientos, un ejemplo clásico es el de las temperaturas superficiales del mar, tomadas para los estudios del suceso conocido como 'el niño', siendo que este sistema tiende a tener un comportamiento casi-periódico.

Por ello se realizó este tipo de ejercicios, obteniendo resultados muy favorables para la predicción de éste. Se presentaron dos ejercicios, ambos muy semejantes, simplemente cambiando una función coseno por una función seno, por ello se aplica la misma estructura neuronal obtenida de una.

Para el proceso de aprendizaje se realizó un numero de iteraciones de $300\ 000$, con el fin de observar el comportamiento del error, no necesariamente en la última iteración se obtiene un error mínimo, ya que debido a la estructura y parámetros manejados llega a estancarse el error, encontrando entonces un error mínimo local. Dado este número de iteraciones, la cantidad de datos señalados a entrenar y el número de neuronas manejadas, el tiempo de procesado es relativamente grande, tardando alrededor de 5 horas.

Con la matriz de pesos encontrados se realiza tanto la prueba como el pronóstico, obteniendo que el resultado es muy similar al original, obteniendo 1000 pasos de tiempo futuros cada vez que realimentamos la **RNA**, lo cual obtenemos una gran ventaja para este tipo de ST.

El corrimiento que se maneja en la ST es nuevamente obtenido y empleado por el *software* VRA, dando un valor de 45, obteniendo el mismo corrimiento para ambas funciones presentadas en el apartado correspondiente a este. También, por este mismo valor para ambos ejercicios se recurre a la misma arquitectura empleada.

En el capítulo dos hemos ejemplificado de como ingresar los datos a la **RNA** así como de realizar el pronóstico. Experimentando de la forma del ejemplo 4, donde los conjuntos de entrada, conjunto objetivo y conjunto de prueba, están sobrepuestos, forma también aplicada para estos dos problemas, adquiriendo resultados no mejores que los descritos. Sin embargo, esta forma de los conjuntos sobrepuestos, la hemos adoptado para lo que fue el caso de *Lorenz*, dado que su comportamiento de esta ST es muy diferente a las anteriores, ingresando más datos para que aprenda y realizando diversos experimentos para obtener resultados convincentes a la hora de pronosticar, donde encontramos un límite a la predictibilidad de la ST.

3.4 Lorenz

Hemos visto que con los resultados que se obtienen con la **RNA**, para las ST del sistema de *Lorenz*, la acción de predecir tiene un límite. Como se ha venido expresando a lo largo de este trabajo, hemos de hacer predicción al realimentar al modelo neuronal con las salidas obtenidas de esta misma, pero como es de esperar, no podemos usar este pronóstico a largo tiempo, ya que se pierde el comportamiento con lo real, siendo esto obsoleto, comprobando y reiterando que no se puede predecir más de lo que se desearía para un sistema caótico como es el sistema de Lorenz, por lo que un sistema como éste siempre presentará dificultades para realizar pronóstico además de estar restringido.

Sin embargo, se puede mejorar el pronóstico, esto se realiza al actualizar los resultados obtenidos antes de realizar nuevamente la predicción, esto es, corregir las salidas antes de ingresarlas nuevamente a la **RNA**, al realizar esto último se realimenta el modelo para obtener los siguientes valores futuros, y así repetir nuevamente el proceso de corregir y realimentar, obteniendo los resultados ya mostrados. Teniendo muy en consideración que este pronóstico siempre está limitado, ya que sería necesario corregir antes de conocer el siguiente paso de tiempo, además es indispensable corregir para seguir usando el modelo particular de la ST en cuestión.

El seguir usando el modelo neuronal para una ST en especial, nos referimos a que no necesitamos entrenar nuevamente al autómata, ya que ésta, aprendió relativamente lo

suficiente, por lo que sólo se realimentaría a la **RNA** con los valores correspondientes al paso de tiempo.

El delay manejado en principio para las ST, fueron de $\mathbf{15}$ para la variable '$\boldsymbol{z}$', $\mathbf{15}$ para '$\boldsymbol{y}$' y $\mathbf{16}$ para '$\boldsymbol{x}$', se menciona con anterioridad que estos corrimientos son los adquiridos mediante el *software* **VRA**, aplicando estos para cada ST correspondiente, pero al estar trabajando sobre estos valores adoptados para las variables '$\boldsymbol{x}$' y '$\boldsymbol{y}$', observamos que nuestro pronóstico de estar actualizando las salidas antes de realimentar a la red, no son muy convincentes, por lo que se opta de reducir el *delay*, tomando una tercera parte de éste, aplicando entonces un corrimiento de $\mathbf{5}$ para cada ST del sistema de *Lorenz*.

Mostrando mejores resultados esta reducción del *delay*, encontrando además una diferente arquitectura para estas variables de *Lorenz*, ya que nunca se dejó de seguir experimentando (prueba y error), poniendo a prueba más y más a la **RNA**, ya que la arquitectura empleada para el entrenamiento ha de ocupar alrededor de una quinta parte de la ST del sistema de *Lorenz*.

Se utilizó la misma arquitectura, en cada ST para el sistema de *Lorenz*, ya que cada variable representa parte del sistema, teniendo que si la estructura neuronal es apta para una de ellas, ésta puede ser igual para las demás, que en efecto, esto ocurre, pero se presenta diferentes errores en lo pronosticado, uno mayor que otro, por lo que aún podríamos seguir experimentando para encontrar diferentes arquitecturas en cada ST, aunque éstas sean parte del mismo sistema.

Las diferencias y modificaciones del modelo neuronal para las ST del sistema de Lorenz respecto al de las del seno y seno con ruido, además de cómo se manejaron los conjuntos de entrada, objetivo y de prueba, se realizaron ajustes en la función de activación, recordando que en todos los entrenamientos para cada ejercicio y experimento se ocupó la función de activación del tipo sigmoide $\boldsymbol{f(h) = 1/\left(1 + e^{-B \cdot h}\right)}$ **(ver Tabla 1.1 Funciones de activación)**, ésta puede cambiar su forma al aumentar o disminuir su argumento $\boldsymbol{B}$ y cuyo comportamiento puede ser más suave o contrario.

Para las ST de seno y seno con ruido se tomó el valor de $\boldsymbol{B = 1}$, pero para las ST del sistema de *Lorenz* se tomó como $\boldsymbol{B = 0.1}$. Esta modificación que se hace es para que el aprendizaje sea más óptimo, ya que si se dejaba como en la del seno, la **RNA** presentaba un estancamiento en el error sin que este disminuyera y al realizar este ajuste en el argumento de la función de activación, obtenemos un descenso en el error, siendo esto un mayor aprendizaje para el autómata.

Los comportamientos de los errores de entrenamiento y de prueba, tienden a ser similares, pero se ha de presentar cierta oscilación en el transcurso del aprendizaje, esto se presenta generalmente al manejar demasiados datos en el entrenamiento. Observamos que en el problema de la función seno, el comportamiento de errores no presenta ninguna oscilación, pero, en experimentos realizados para esta función, se ingresaron una mayor cantidad de datos de los descritos, observando que el comportamiento del error en cierta iteración empezaba a oscilar, tal como ocurre en el problema del seno con ruido. Sin embargo, aunque se tengan presentes estas variaciones,

los errores de entrenamiento y de prueba van decreciendo conforme el proceso va iterando.

También se encontró que la oscilación va dependiendo del número de neuronas manejas en la estructura, ya que si se tienen un mayor número de neuronas, las oscilaciones aumentan, aunque, conforme se va iterando, la oscilación va decreciendo. Con un número adecuado de neuronas, es posible que se presenten estas variaciones, pero serán menores, manejando una amplitud de milésimas o mucho menos.

Los pesos óptimos fueron alcanzados en el error mínimo de entrenamiento, estos pesos es la base para adoptar el modelo adquirido de cierta ST, recordando que al estar iterando se van modificando estos pesos, modificación que describe el aprendizaje del autómata. Teniendo entonces que entre más disminuye el error de entrenamiento, la red aprende más, por ello, se aplicó un número de $300\,000$ iteraciones en cada proceso de aprendizaje, obteniendo el modelo y resultados ya descritos, con esta cantidad de iteraciones se midieron los tiempos de procesado, siendo estos alrededor de $8\,hrs$, recordando que la cantidad de datos en el entrenamiento fueron de 1000 para cada nodo.

Para todos los problemas manejados y descritos en este trabajo, y logrados los resultados presentes, podemos dar a concluir lo siguiente.

3.4 Conclusión general.

El detallar y ejemplificar desde las operaciones que realiza la **RNA**, el cómo está constituida la estructura de un PMC y las funciones que ha de realizar, el comparar ejemplos clásicos que se logran encontrar en la literatura, expresar las formas que podemos ingresar las ST, es una aportación que se da en este trabajo, ya que es reducida la bibliografía en que podemos encontrar el paso a paso de todo lo expuesto.

El objetivo estuvo siempre sobre las ST, ya que éstas son el elemento principal de este trabajo, en el cual radica el problema y el estudio, que es realizar un pronóstico sobre estas mismas. Hay que hacer mención que cada ST se trabajó individualmente, es decir, que sólo se utiliza una ST (una variable), de esta se utiliza parte de su historia (datos) para ingresarla al modelo y obtener su pronóstico, dicho de otra manera, se trabajó en el espacio de fases. Haciendo referencia a trabajos descritos en la literatura, hacen uso de más de una variable para obtener otra diferente, trabajando en el espacio de estado, por ejemplo, se quiere hacer pronóstico para precipitación, se podrían ingresar ST de las variables humedad, temperaturas máximas, etc., ingresando todas estas variables al modelo para obtener la evolución de precipitación.

En nuestro caso, se trabajó en el espacio de fases, ya que estamos usando la misma variable para obtener la evolución que esta ha de presentar. Esto es similar a lo que se trabaja en sistemas dinámicos y reconstrucción del atractor, siendo una aportación más a este trabajo.

Se podría haber trabajado en el espacio de estado para las ST del sistema de *Lorenz* experimentado, por ejemplo, al ingresar las ST de 'x' y 'y' al modelo para obtener 'z', y de esta manera hacer sus variaciones para obtener cada una de las variable; si se logrará pronosticar una sola variable, podríamos conocer la evolución de las demás. Pero como hemos de poner a prueba las capacidades de la **RNA** se trabajó con una sola variable para cada problema desarrollado.

La cantidad necesaria de neuronas manejadas en cada estructura empleada, fue por ensayo y error, se realizó de esta manera para, cuando al tener una óptima estructura, se encontrara alguna relación de ciertos patrones que pudiesen ser descritos en el análisis de las ST. Por ejemplo, al ingresar las ST en el VRA, podemos obtener diversos parámetros como la dimensión de correlación, dimensión de *embedding*, etc., parámetros que se utilizan en el análisis de series de tiempo, de estos parámetros coincidir con algún número en la estructura de la **RNA**, esto sería de gran utilidad y de gran fundamento para la construcción en la arquitectura de las redes. Otro ejemplo que se hubiera empleado es el de las Wavelets, donde obtenemos la periodicidad de la ST, número de varianza, etc., y con ello encontrar alguna relación con la construcción del autómata, ya que si se tiene, se puede utilizar y evitar el ensayo y error reduciendo el número de experimentos.

Lo descrito en el anterior párrafo, es manejado desde un punto de vista de minería de datos, ya que se obtiene información sobre las ST y cuya finalidad es emplearla al configurar la estructura neuronal y la cantidad de datos necesarios a usar.

Desafortunadamente no hemos logrado encontrar algún patrón, alguna información que se asemeje con lo empleado. En el trabajo expuesto se manejó lo que fue el corrimiento o *delay*, se optó manejar este parámetro para reducir los casos de experimentación de ensayo y error, valiéndonos del software VRA, éste nos da el *delay* de cada ST, pero hay que ser cautelosos en manejar y adoptar ciertos patrones que puedan arrojar los softwares y/o modelos para el análisis de ST, ya que éstos pueden ser, o no, idóneos para el funcionamiento de la **RNA** y que ésta de resultados favorables.

El aprendizaje de la **RNA** logra hacerlo de buena forma, pero hay que señalar que este aprendizaje es bueno sólo para las ST de comportamiento suave, como lo es la función seno o bien para funciones casi-periódicas, y cuyos pronósticos logran ser fiables. También hemos observado que el autómata puede aprender muy bien el ruido, pero al pronosticar, este no es favorable, por lo que la cuestión es de ¿Qué tanto se quiere que aprenda la **RNA**?, en lo descrito del seno con ruido, lo importante es que aprenda la tendencia que esta tiene, logrando hacer esto mediante la validación cruzada, más sin en cambio, no se garantiza de que el resultado llegue a encontrar que la matriz de pesos sea óptima, en este caso que el aprendizaje junto con la validación cruzada llegue a reacondicionar la matriz de pesos, y el resultado que se logró encontrar es debido a que el ruido integrado a la función seno se repita, siendo el mismo ruido para cada periodo de la ST, y de esta manera logra que la **RNA** haga un buen pronóstico describiendo la forma que es la función sin ruido.

Hemos puesto a prueba las capacidades generales que han de favorecer el uso de **RNA**, en específico al modelo de PMC, pero al experimentar con las ST del sistema de *Lorenz*,

encontramos un límite a la predictibilidad, que en efecto esto ocurre en ST con cierto comportamiento caótico, más sin en cambio, aunque se tenga esta delimitación al pronosticar, se logra utilizar el mismo modelo una y otra vez, obteniendo un pronóstico servible, pero siendo este limitado, debido a las mismas limitaciones que tiene el modelo, por lo que las **RNA**, en este caso, no logra superar a los modelos que hacen pronóstico, que usualmente es como trabajan. Además de que si se lograra pronosticar este tipo de ST, dejarían de tener comportamiento caótico, ya que por definición dentro de la teoría del caos es imposible predecir.

Para la cantidad de datos a entrenar, haciendo un promedio con los problemas expuestos en este trabajo, se utilizaron una quinta parte del total de las ST que se tenían a disposición, logrando así un adecuado aprendizaje con una cantidad de datos no necesariamente grande, más sin en cambio, esto es contradictorio puesto que en el problema de la función seno, se utilizan un número de datos mayor a lo que contiene en uno de sus periodos, ya que en un solo periodo contiene toda la información necesaria para la evolución de la función, por lo que se ha de suponer que la **RNA** necesita, además de conocer todos los posibles valores de la ST, conocer un poco más de lo necesario aunque ya se haya incluido dicha información o se esté repitiendo.

Desafortunadamente lo desfavorable, es la oscilación en el error, esto se presenta al trabajar con las ST de *Lorenz*, ya que ésta atrasa el tiempo de convergencia, o bien, se pierde dicha convergencia al no encontrar un error mínimo. Se menciona que aunque haya oscilación, el error sigue disminuyendo, pero no es conveniente de que se presente estas variaciones. Siendo esto una desventaja, pero se puede mejorar esto si se encuentra

una estructura neuronal que sea óptima para reducir, o bien, desaparecer dicha oscilación. No obstante, el sistema de *Lorenz* presenta comportamiento caótico, por lo que es congruente que el error presente oscilación y que no llegue a converger.

El tiempo de procesado es fundamental y más si se realiza un pronóstico a tiempo real, por ejemplo, cada media hora se toman mediciones, mismo paso de tiempo que se ha de pronosticar, por lo que el tiempo de ejecución del algoritmo debe ser de inmediato. O bien, semejante al de este trabajo, siendo que el proceso de ejecución le lleve un tiempo mucho mayor, puede tardar hasta un mes, pero este modelo podrá ser rentable para varios pasos de tiempo, por ejemplo, usar el mismo modelo para toda la temporada de verano.

Para nuestro caso, aunque se haya tardado cierto tiempo en proceso de aprendizaje, este modelo se adopta para una gran cantidad de pasos hacia adelante, recordemos la función seno, basto con entrenar alrededor de 200 datos, pronosticando 800 pasos adelante para hacer una comparación con la ST original, mostrando buenos los resultados. Las ST del sistema de *Lorenz* se puede apegar un poco más a la realidad, por ello presenta un límite a la predictibilidad, pero este modelo se reutiliza, ordinario a los modelos que realizan pronóstico, por lo que al usar y ajustar resultados una y otra vez, se puede mostrar un comportamiento muy similar al original. Ahora bien, el número secuencial que fue tomado para cada ST, podrían representar horas, días, meses, dependiendo la escala con la que se haya trabajado la ST en cuestión. Por ello se puede decir que el modelo es funcional, a pesar del tiempo de procesado, pero también con ciertas limitaciones que es al pronosticar ST caóticas, más sin en cambio favorables para ST periódicas y casi-periódicas.

La idea de utilizar este tipo de métodos es propuesta por la llamada bio-inspiración, debido a que se ha de recurrir a la naturaleza para resolver problemas, en este caso se recurre a la naturaleza del funcionamiento del cerebro, en específico el funcionamiento de las neuronas. Por ello es muy atractivo de recurrir a este tipo de métodos, pero hay que ser cautelosos en cómo y dónde aplicarlo, ya que como hemos observado en este trabajo, en el modelo que implementamos, la **RNA** es fiable para ST con comportamiento periódico y casi-periódico, sin embargo, ST que presentan comportamiento caótico, no han de ser superados por la **RNA**, quedando al par de un modelo tradicional que pronostica, llegando al mismo punto de partida que es la teoría del caos, la cual es impredecible.

La **RNA** pueden ser muy beneficiosos en otras aplicaciones, en lo que hemos estudiado y aplicado, son adecuados para interpolar o predecir de forma periódica y casi-periódica. No obstante, para quienes quisieran recurrir al entorno de este tema, se le hace las siguientes recomendaciones para trabajos futuros que es, mejorar el tiempo de procesado para continuar la experimentación de prueba y error, o bien, al realizar el análisis de ST y obtener ciertos parámetros, para utilizarlos en la estructura de la **RNA**, siempre y cuando sean fundamentados y que estos sean confiables. Hoy en día, podemos encontrar trabajos que mejoran las estructuras de la **RNA** para un PMC (empleado en este trabajo), evitando el ensayo y error, pero hay que hacer atribución que al estar trabajando de esta manera se logra una mayor comprensión en el entorno de la **RNA**, ya que el estudio de este trabajo se realizó desde cómo opera, llevándolo a la programación y funcionamiento, conociendo más a profundidad este método.

En la literatura hemos de encontrar aplicaciones comúnmente en áreas de estudio como ingeniería electrónica, computacional, mecánica, etc., y no es de sorprenderse, ya que de un lugar nace esto de las **RNA**, en el transcurso del tiempo, este tema ha sido tomado y se hace uso en diversas áreas científicas, como en la economía para realizar pronóstico, para el área de robótica, bioingeniería, en el uso militar. Encontramos también en el ámbito de ciencias de la tierra, como es en el área de atmosfera, que se utiliza para predicción; o bien, en registros de pozos, haciendo uso para el mejoramiento de imágenes de éstos. En fin, una infinidad de usos que podemos encontrar. Por lo que invito al lector, al usuario, al experto en este tema, difunda los trabajos, se empape del tema, que se mejore el uso de esta herramienta, que realicen temas que demuestren matemáticamente el uso fiable de los modelos neuronales, en fin, seguir avanzando y explotar el tema, dando a conocer sus alcances y limitaciones, que han de presentar cualquier método y/o modelo, señalando que un modelo sólo asemeja a la realidad.

ANEXOS

Anexo A1. Neurona biológica.

Las **RNA** están inspiradas en la estructura y funcionamiento de los sistemas nerviosos, donde la neurona biológica es el elemento fundamental.

El cerebro humano está constituido por un arreglo complejo de neuronas; se estima que el sistema nervioso contiene alrededor de **'cien mil millones de neuronas'**, que dan lugar a un **sistema cognitivo robusto**, con un diseño distinto al de las computadoras. Vistas al microscopio, este tipo de células puede presentarse en múltiples formas, aunque muchas de ellas tienen un aspecto similar muy peculiar **(ver figura A1.1)**; con un cuerpo celular o **soma**, que es donde se localiza el núcleo, del que surge un denso árbol de ramificaciones (árbol dendrítico compuesto por dendritas); las **dendritas** es por donde las neuronas reciben o atrapan las señales provenientes de otras neuronas, y del cual parte una fibra tubular denominada **axón**, ésta se extiende desde el cuerpo de la célula, hasta eventualmente ramificarse en filamentos y sub-filamentos, que también se ramifica en su extremo final para conectar con otras neuronas (cada neurona puede conectarse con otras diez mil neuronas, en promedio).

En los extremos de las ramificaciones del axón, se encuentran las uniones sinápticas o **sinapsis** y representa el aprendizaje adquirido hacia las otras neuronas. El axón de una neurona típica realiza algunos miles de sinapsis con otras.

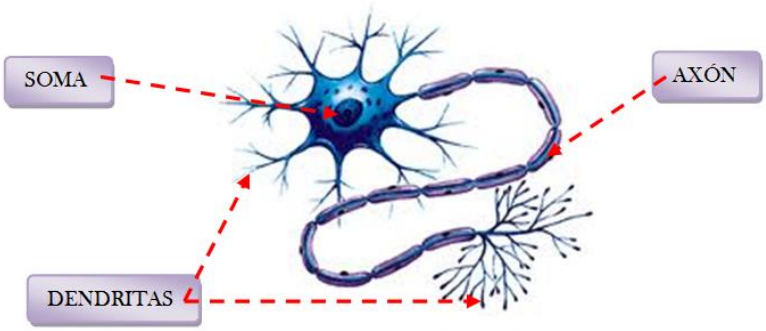


Figura A1.1 Estructura General de una Neurona Biológica Típica

Figura modificada, toma de: Tema 5. Fundamentos Biológicos de la Psicología, http://psicologiavprado.blogspot.mx/p/tema-5-fundamentos-biologicos-de-la.html [consulta: 02-10-15]

En 1888, Santiago Ramón Cajal (científico aragonés el cual inicia la historia de las redes neuronales artificiales, descubridor de la estructura neuronal del sistema nervioso) demostró que el sistema nervioso estaba compuesto por una red de células individuales, 'las neuronas', ampliamente interconectadas entre sí. No sólo observó al microscopio los pequeños vacíos que separaban unas neuronas de otras, sino que también estableció que la información fluye en la neurona, desde las dendritas hacia el axón, atravesando el soma.

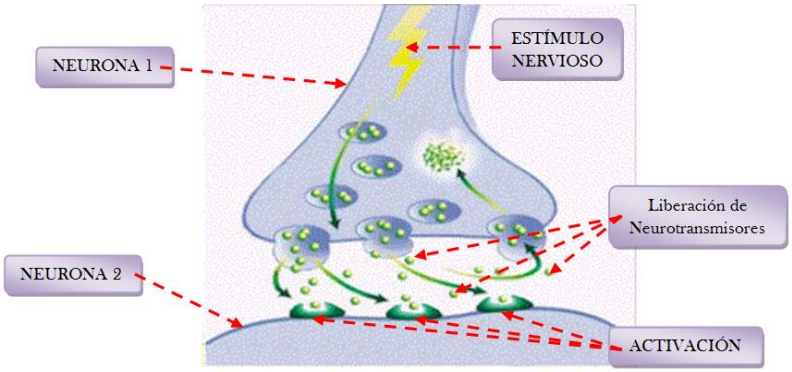


Figura A1.2 Transmisión de una señal (Sinapsis), de la Neurona 1 a la Neurona 2.

Figura modificada, toma de: Sinapsis: Comunicación entre neuronas. http://www.iibce.edu.uy/uas/neuronas/abc.htm [consulta: 02-10-15].

La transmisión de una señal (típicamente en forma de un tren de pulsos) de una célula a otra, a través de la sinapsis, se realiza mediante un proceso químico complejo, en la que algunas sustancias específicas favorecen la transmisión, y son emitidas del lado de la

neurona emisora con el objetivo de incrementar o disminuir el potencial eléctrico dentro del cuerpo de la célula receptora. Si este potencial alcanza un umbral determinado, un pulso de duración e intensidad fija, es enviado a través del axón. Es entonces cuando se dice que la neurona ha disparado. El pulso se ramifica a través de las arborizaciones del axón hasta alcanzar las uniones sinápticas con las otras células.

Después de disparar, la célula tiene que esperar un intervalo de tiempo conocido como periodo de refracción, antes de que pueda volver a disparar otra vez. Cabe aclarar que una sinapsis, en realidad, no es una conexión física **(ver figura A1.2)**. En resumen, la sinapsis se convierte de una señal eléctrica a una señal química y luego nuevamente en una señal eléctrica, o de forma más simple, una sinapsis es una conexión que puede **excitar** o **inhibir** a otra neurona, pero no ambas.

Las Redes Neuronales Biológicas (**RNB**) pueden establecerse como grupos de neuronas activas especializadas en tareas como: cálculos matemáticos, posicionamiento y memoria [Simpson P.K., 1990].

Anexo A2. Comparación de las RNA con el ordenador.

El campo de las **RNA** fue motivado por entender la forma en la que trabaja el cerebro humano así como la de cualquier ser vivo, a diferencia de la forma que trabajan las computadoras.

Se sabe que el cerebro está compuesto de neuronas, mientras que en una computadora se tienen componentes electrónicos como compuertas lógicas. Ahora, si comparamos el tiempo que dura un evento en una compuerta de silicón ($\sim 10^{-9}$ *s*) con la duración de un evento en una neurona ($\sim 10^{-5}$ *s*), vemos que hay una diferencia entre cinco y seis órdenes de magnitud. Sin embargo, el cerebro compensa esta falta de rapidez, teniendo una enorme cantidad de neuronas, junto con una cantidad aún más impresionante de conexiones o sinapsis entre ellas.

Mientras el cerebro es capaz de procesar la información que recibe a través de los ojos, para que logremos interactuar con nuestro entorno, o reconocer caras familiares en un ambiente cualquiera, una computadora puede tomar días en cálculos mucho menos complejos. Además, el cerebro resulta ser una máquina mucho más eficiente energéticamente hablando, usando alrededor de 10^{-16} *J/s* por operación, mientras que muchas computadoras gastan alrededor de 10^{-6} *J/s* por operación. En conclusión, si quisiéramos tener una computadora comparable con el cerebro, tendría que ser una máquina extremadamente compleja, no lineal y capaz de operar en paralelo [Simon Haykin, 1994].

La **RNA** está compuesta por elementos simples operando en paralelo. Esta idea está inspirada en las **RNB**, en ellas se realiza algún aprendizaje al crear distintas conexiones entre los elementos que la conforman. Al procedimiento usado para llevar a cabo el proceso de aprendizaje se llama algoritmo de aprendizaje, que consiste en simplemente modificar los pesos de las conexiones sinápticas, de igual manera, en las **RNA** se ajustan las conexiones (pesos) entre sus neuronas, permitiéndonos hacer que dada una entrada (*input*), la red esté entrenada para dar cierto tipo de salida (*output*).

Los actuales ordenadores son máquinas de *Von Neumann*, en esencia una máquina de procesamiento (*hardware*) que actúa ejecutando en serie (una tras otra) una secuencia de instrucciones o programa (*software*), que almacena en su memoria.

La máquina está compuesta por cuatro unidades básicas: unidad de entrada de información, unidad de salida, unidad de procesamiento (compuesta a su vez por la unidad lógico-aritmética y la unidad de control) y memoria. Este sistema constituye una máquina universal de cómputo en el sentido de *Turing*, por lo que puede llevar a cabo, en principio, cualquier tarea sólo con programarla de forma adecuada. *Alan Turing* emprendió el estudio formal de la computación, para lo que introdujo una máquina ideal conocida como máquina de *Turing* [Conrad M., 1992; Hopcroff J.E., 1984].

Para un ordenador, el verdadero corazón de éste, es el microprocesador, potente y complejísimo circuito electrónico, que en la actualidad puede integrar varios millones de componentes electrónicos en un espacio de alrededor de un centímetro cuadrado [Martín del Brío, 1999]. La mayor parte de los microprocesadores no se encuentran dentro de los

ordenadores, como se podría pensar, sino contenidos en todo equipo que contenga cierta cantidad de electrónica, como puedan ser hornos de microondas, aparatos de televisión, videos, cámaras de video, lavadoras, etc. Estos microprocesadores que se dedican a tareas de control se denominan microcontroladores, integrando en una única pastilla todos los elementos del computador: CPU (*Central Processing Unit*), memoria, entradas y salidas, etc. [Martín del Brío, 1999].

La neurona es en realidad un pequeño procesador, sencillo, lento y poco fiable (a diferencia de nuestros potentes microprocesadores), sin embargo, en nuestro cerebro cohabitan unos 'cien mil millones de neuronas' operando en paralelo y a diferentes escalas. Es aquí donde reside el origen de su poder de cómputo. Aunque individualmente las neuronas sean capaces de realizar procesamientos muy simples, ampliamente interconectadas a través de las sinapsis (cada neurona puede conectarse con otras diez mil en promedio) y trabajando en paralelo pueden desarrollar una actividad global de procesamiento enorme.

BIBLIOGRAFÍA

[Aréchiga Hugo, 2001]

Aréchiga Hugo, ***El universo interior***, la ciencia para todos 182, Fondo de Cultura Económica, 2001.

[Asady B. *et al.*, 2014]

Asady B., Hakimzadegan F., Nazarlue R., ***Utilizing artificial neural network approach for solving two-dimensional integral equations***, Springerlink.com, 2014

[Bavafaye Haghighi, 2015]

Bavafaye Haghighi E., Palma G., Rahmatib M., Yazdanpanahc M. J., ***A new class of multi-stable neural networks: Stability analysis and learning process***, Neural Networks 65, 53-64, 2015. http://dx.doi.org/10.1016/j.neunet.2015.01.010

[Braun Eliezer, 1996]

Braun Eliezer, ***Caos, fractales y cosas raras***, la ciencia para todos 150, Fondo de Cultura Económica, 1996.

[Corchado J. M., 2000]

Corchado, J. M., Díaz, F., Borrajo, L., Fernández, F., ***Redes neuronales artificiales, un enfoque práctico***, servicio de publicaciones de Universidad de Vigo, España, 2000.

[Cybenko, 1989]

Cybenko, G., ***Approximation by superposition of a sigmoidal function***, Mathematics of control, Signals and systems 2, 303-314, Springer-Verlag New York Inc., 1989.

[Funahashi, 1989]

Funahashi, K. I., ***On the approximate realization of continuous mappings by neural networks***, Neural Networks 2, pp. 183-192, Elsevier Science Ltd. Oxford, UK, 1989.

[González Hortensia *et al.*, 2012]

González Hortensia, Arce Humberto, ***El caos, un intento por dar sentido a la realidad***, Antologías de la revista ciencias 2, México: siglo veintiuno, UNAM, Facultad de ciencias, 2012.

[Hassoun M. H., 1995]

Hassoun, M. H., ***Fundamentals of artificial neural network***, the MIT Press, London, England, 1995.

[Haykin S., 1999]

Haykin S., ***Neural networks, a comprehensive foundation***, 2a edición, Prentice Hall, New Jersey, USA, 1999.

[Hetcht-Nielsen, 1987]
Hetcht-Nielsen R., ***Kolmogorov's mapping neural networks existence theorem***, Proc. Int. Conf. on Neural Networks, III, pp. 11-13, 1987.
[Hetcht-Nielsen, 1990]
Hetcht-Nielsen R., ***Neurocomputing***, Addison Wesley, 1990.
[Hilera González, 1995]
Hilera González J. R., Martínez V. J., ***Redes neuronales artificiales, fundamentos, modelos y aplicaciones***, Addison-Wesley, Madrid, España, 1995.
[Hornik *et al.*, 1989]
Hornik K., Stichcombe M., White H., ***Multilayer feedforward networks are universal approximators***, Neural Networks, 2, pp. 359-366, 1989.
[Isasi Viñuela P. *et al.*, 2004]
Isasi Viñuela P. M., Galván León I. M., ***Redes neuronales artificiales. Un enfoque práctico***, Pearson Educación, S. A., Madrid, 2004.
[Kasiviswanathan K. *et al.*, 2013]
Kasiviswanathan K. S., Cibin R., Sudheer K. P., Chaubey I., ***Constructing prediction interval for artificial neural network rainfall runoff models based on ensemble simulations***, Journal of Hydrology 499, 275–288, 2013. http://dx.doi.org/10.1016/j.jhydrol.2013.06.043
[Khalid M. *et al.*, 2014]
Khalid M., Sultana Mariam, Zaidi Faheem, ***Numerical solution of sixth-order differential equations arising in Astrophysics by neural network***, International Journal of Computer Applications, 107-6, 2014.
[Lapades A. S., Faber R. M. 1987]
Lapades A. S., Faber R. M., ***Nonlinear signal processing using neural network: Prediction and system modelling***, Los Álamos, technical report L-UR-87-2662, 1987
[Martin del Brío, 2002]
Martin del Brío Bonifacio, Sanz Molina Alfredo, ***Redes neuronales y sistemas difusos***, 2a edición ampliada y actualizada, Alfaomega, México, c2002.
[Martin del Brío, 2007]
Martin del Brío Bonifacio, Sanz Molina Alfredo, ***Redes neuronales y sistemas difusos***, 3a edición revisada y ampliada, Alfaomega, México, D.F., 2007.
[Masters T., 1993]
Masters T., ***Practical neural networks recipes in C++***, Academic Press, 1993.

[Nastos P. *et al.*, 2014]
Nastos P. T., Paliatsos A. G., Koukouletsos K. V., Larissi I. K., Moustris K. P., ***Artificial neural networks modeling for forecasting the maximum daily total precipitation at Athens, Greece***, Atmospheric Research 144, 141–150, 2014. http://dx.doi.org/10.1016/j.atmosres.2013.11.013

[Pavel Simonov, 1968]
Pavel Simonov Ezras Asratian, ***La función del cerebro***, Grijalbo, S. A. DINA, versión en español, 1968.

[Peña Maciel, 2012]
Peña Maciel Daniel, ***Análisis de series de tiempo de variables atmosféricas aplicando la teoría de sistemas dinámicos no lineales y un enfoque de minería de datos: Estado de Tlaxcala***, Tesis de Licenciatura, Facultad de Ingeniería, UNAM, 2012.

[Pérez Pérez, 2001]
Pérez Pérez Edgar, ***La precipitación pluvial en la ciudad de México como sistema dinámico caótico y su modelación y pronóstico con redes neuronales artificiales***, Tesis de Maestría, Facultad de Ingeniería, UNAM, 2001.

[Principe, 2000]
Principe J. C., Euliano N. R., Lefebvre W. C., ***Neural and adaptive system. Fundamentals through simulations***, John Wiley, 2000.

[Ravi P. *et al.*, 2011]
Ravi P. Shukla, Krishna C. Tripathi, Avinash C. Pandey, I.M.L. Das, ***Prediction of Indian summer monsoon rainfall using Niño indices: A neural network approach***, Atmospheric Research 102, 99–109, 2011.
www.sciencedirect.com/science/article/pii/S0169809511001918

[Rojas Raul, 1996]
Rojas Raul, ***Neural networks: A systematic introduction***, Springer-Verlag, Berlin, 1996.

[Rumelhart D. E., 1986]
Rumelhart D. E., Hinton G. E., Willians R. J., ***Learning representations by backpropagating errors***, Nature, 323, pp. 533-6, 1986.

[Rudd Keith *et al.*, 2015]
Rudd Keith, Ferrari Silvia, A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks, Neurocomputing 155, 277–285, 2015. http://dx.doi.org/10.1016/j.neucom.2014.11.058

[Sametband Moisés, 1999]
Sametband Moisés José, ***Entre el orden y el caos: La complejidad***, la ciencia para todos 167, Fondo de Cultura Económica, 1999.

[Sánchez Camperos, 2006]

Sánchez Camperos, Edgar Nelson., ***Redes neuronales: Conceptos fundamentales y aplicaciones a control automático***, Pearson Educación, Madrid-México, 2006.

[Tyrell DeWeber *et al.*, 2014]

Tyrell DeWeber Jefferson, Wagner Tyler, ***A regional neural network ensemble for predicting mean daily river water temperature***, Journal of Hydrology 517, 187–200, 2014. http://dx.doi.org/10.1016/j.jhydrol.2014.05.035

[Wang Xin *et al.*, 2015]

Wang X., Yu J., Li C., Wang H., Huang T., Huang, J., ***Robust stability of stochastic fuzzy delayed neural networks with impulsive time window***, Neural Networks, 2015, http://dx.doi.org/10.1016/j.neunet.2015.03.010

[Werbos P. J., 1974]

Werbos P. J., ***Beyond regression: New tools for prediction and analysis in behavioral sciences***, Doctoral Dissertation, Applied Mathematics, Harvard University, Noviembre, 1974.