Reconstrucción de Forzamiento en Sistemas Dinámicos

Rodrigo Villavicencio Sánchez

3 de abril de 2009

Índice general

Capítulo 1

Introducción

En esta tesis se busca, primero que nada, aprender a usar redes neuronales artificiales, como un método predictor de señales caóticas. Por otro lado, sabemos que un fenómeno natural está sujeto a distintas variables, y al analizarlo no necesariamente tenemos que saber cuales son todas las que lo afectan. Pero para predecirlo de manera aproximada no necesitamos conocer su reacción a todas las variables que lo influyen, por eso es que existen modelos de distintas complejidades, con los que se logra reproducir de mejor o peor manera el comportamiento de la naturaleza. Por ello, intentamos crear un modelo computacional con el cual se pueda predecir el comportamiento de un sistema dinámico aparentemente muy complejo, usando relativamente poca información. Para esto es necesario tener un poco de conocimiento sobre sistemas dinámicos y que tipo de redes neuronales artificiales se usan para predecirlos.

Como nos enfocamos en predecir series de tiempo artificiales, en particular en el mapeo logístico, para poder saber que tan buenas son nuestras predicciones también es indispensable saber como se comportan las series de tiempo generadas por este sistema al hacer cambios en sus parámetros. Lo que nos lleva a hablar un poco de caos y la forma en la que se hace la transición de un sistema muy simple y facilmente predecible a uno tan complicado.

Una vez familiarizado con estos temas solo queda empezar el análisis sobre como con base en el conocimiento de los errores cometidos por la red neuronal, se puede llegar a reconstruir una señal que afecta las series de tiempo de manera aparentemente imperceptible. Y aún que sabemos que

nuestros resultados se pueden mejorar, con lo que hemos logrado hasta ahora es suficiente para observar resultados exitosos. No por ello quedandonos como si el proyecto estuviera totalmente terminado, aún hay algorítmos faltantes por implementar y por mejorar para poder llegar a hacer un uso más avanzado nuestras redes neuronales, hasta llegar al tratamiento de sistemas dinámicos reales.

1.1. Sistemas Dinámicos

Los sistemas dinámicos se permiten estudiar una gran cantidad de feómenos, ya que básicamente los podemos definir como cualquier sistema que evolucione con el paso del tiempo. Una de las mayores ventajas de estos, es el éxito con el que se pueden usar para modelar una gran cantidad de problemas físicos. Para definir un sistema dinámico necesitamos dos condiciones [1]:

a) Vector de Estado: consiste de todas las variables necesarias para definir el estado inicial del sistema.

b) Función: consiste en la regla que nos determina, a partir del estado actual, cual va a ser el estado al siguiente instante; donde el "instante" lo podemos pensar como un cambio discreto o continuo.

Por dar un ejemplo sencillo, supongamos que tenemos una cuenta bancaria con \$100 y una tasa de interés anual que es del 6 % [1]. En este caso el vector de estado sería $x(0) = 100$ y la regla de evolución para saber el estado del sistema al siguiente instante $x(k+1) = 1{,}06x(k)$. El tiempo en este caso es discreto porque la regla que nos indica como va a cambiar el sistema actúa solo hasta que ha transcurrido un año. Pero esta forma de ver los cambios en los sistemas, no siempre es la más práctica para todo tipo de problemas, por ejemplo, podemos pensar en el lanzamiento vertical de una pelota desde una altura h_0 a una velocidad inicial v_0, en cuyo caso el tiempo transcurre de manera contínua; ahora, en lugar de preguntarnos cual será el siguiente estado del sistema, tiene más sentido preguntarnos como está cambiando el sistema a cada instante, lo que nos introduce inmediatamente a problemas con ecuaciones diferenciales. Pero este caso lo dejaremos a un lado puesto que el objetivo principal de esta tesis es utilizar las redes neuronales para encontrar patrones de forzamiento en series de tiempo, y a pesar de que los

fenómenos realmente interesantes son los de la vida real, en esta tesis el enfoque se dirige totalmente al caso de señales generadas artificialmente por distintos tipos de mapeos. Especialmente el mapeo logístico usando tiempo discreto, pues como ya se dijo antes, se quieren analizar series de tiempo, lo que implica que los datos que se tienen de los fenómenos analizados de esta forma no utilizan mediciones de tiempo contínuas.

Como bien sabemos, una forma sencilla de clasificar a las funciones es en lineales y no lineales. Y lo que averiguamos inmediatamente después de haberlas definido de esta forma, es que es mucho más fácil hacer operaciones algebráicas con las lineales. Siguiendo esta linea de razonamiento, al igual que con las funciones, podemos hacer una primera clasificación de los sistemas dinámicos en lineales y no lineales, obviamente esperando que el estudio de los primeros sea el más sencillo. Partiendo de esto damos la siguiente definición.

Definición: Supongamos que tenemos una función $y = f(x)$ y un sistema dinámico cuya regla de recurrencia está dada por $x(k+1) = f(x(k))$. Si la función $f(x)$ es lineal i.e. $f(x) = Ax + b$, entonces tenemos un sistema dinámico lineal [1]. Hay que notar que esta función también puede estar definida de $\mathbf{R}^n \rightarrow \mathbf{R}^n$ en cuyo caso la constante "A" sería una matriz de $n \times n$ y "b" es un vector de n dimensiones. Ahora la definición de los sistemas dinámicos no lineales es inmediata, asumiendo simplemente que la función $f(x)$ no es lineal, por ejemplo:

$$x(k+1) = \alpha x(k)(1 - x(k)). \tag{1.1}$$

A esta ecuación se le conoce como ecuación logística -discreta-[3]. Como ya habíamos dicho, en general es mucho más fácil trabajar con sistemas lineales, pero los no lineales resultan ser mucho más interesantes porque son mejores para modelar fenómenos de la vida real. Lo que se hace normalmente es aprender tanto como se pueda sobre las técnicas para resolver sistemas lineales y luego aplicarlas a los no lineales, teniendo en cuenta que así como en algunos casos pueden funcionar muy bien, en otros nos pueden ser prácticamente inútiles [1].

También hay que notar que aunque ya hemos mencionado tanto sistemas lineales como no lineales, aun no hemos sido tan generales como se puede

ser, ya que también se puede considerar que la función de recurrencia sea de la forma $x(k+1) = f(k, x(k))$, como por ejemplo $x(k+1) = kx(k)$. A este tipo de sistemas se les conoce como *sistemas dinámicos no-autónomos* [2]. Este tipo de sistemas también es muy importante en las aplicaciones, y es nuestra intención intentar estudiarlos pensando en el parámetro temporal k como un forzamiento sobre la serie de tiempo, problema que es abordado más adelante.

1.2. Redes Neuronales

El trabajo hecho en el campo de las redes neuronales artificiales fue motivado inicialmente por el interés de entender la forma en la que trabaja el cerebro de cualquier ser vivo, pues es completamente diferente a la forma en la que trabajan las computadoras hoy en día. Por ejemplo, sabemos que el cerebro está compuesto de neuronas, mientras que en una computadora tenemos 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 ($\sim 10^{-3}s$) vemos que hay una diferencia entre cinco y seis ordenes de magnitud. Sin embargo, el cerebro compensa esta falta de velocidad, teniendo una enorme cantidad de neuronas junto con una cantidad aún más impresionante de conexiones o sinápsis entre ellas. Mientras el cerebro es capáz de procesar la información que recibe a travéz de los ojos, para que logremos interactuar con nuestro entorno, o reconocer caras familiares en un ambiente cualquiera, como un simple trabajo de rutina, una computadora puede tomar días en cálculos mucho menos complejos. Además el cerebro resulta ser una maquina 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 podemos ver al cerebro como una computadora, extremadamente compleja, no lineal y capáz de operar en paralelo [9].

La forma en la que el cerebro logra aprender a realizar todos sus procesos de manera rutinaria, es mediante la experiencia, que se ve unicamente como la creación de sinapsis, que se pueden considerar como las unidades elementales tanto de estructura como de funcionamiento de la interacción entre una neurona y otra. El tipo más común de sinápsis es la de orígen químico, en la que se difunde una sustancia transmisora a lo largo de la unión en-

tre dos neuronas, que ayuda tanto en un proceso presináptico como en uno possináptico, de manera que en resumen, se convierte una señal eléctrica, en una señal química y luego nuevamente en una señal eléctrica. O de forma más simple, una sinápsis es una conexión que puede exitar o inhibir a otra neurona, pero no ambas. Otras partes importantes de las neuronas son las líneas de transmisión, o axones, y las terminales receptoras de señales, dendritas. Como podemos ver en la figura 1.1, estas dos partes son fácilmente distinguibles, pues los axones son terminales largas y sin grandes ramificaciones, mientras que las dendritas, presentan gran cantidad de ramificaciones y tienen una superficie mucho más irregular, asemejandose más a las ramas de un arbol. En la figura 1.1 se muestra una neurona piramidal, que es un tipo muy común de célula cortical, sin embargo, existe una gran variedad de células de este tipo, tanto en forma como en tamaño, dependiendo de la zona del cerebro en la que se encuentren.

Una red neuronal propensa naturalmente a guardar información, para des- pués hacerla disponible para algún uso, debe asemejarse al cerebro en dos aspectos:

1.- El conocimiento debe ser adquirido mediante un proceso de aprendizaje.

2.- Debe usar las fuerza de las conexiones neuronales -sinapsis- para almacenar la información.

Una red neuronal artificial está compuesta por simples elementos operando en paralelo. Esta idea fue inspirada en las redes neuronales de la naturaleza, en las que se realiza algún aprendizaje al crear distintas conexiones entre los elementos que la conforman. Al procedimento 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 redes neuronales artificiales se ajustan las conexiones -pesos- entre sus neuronas, permitiendonos hacer que dada una entrada (input) la red esté entrenada para dar cierto tipo de salida (output). Las redes neuronales artificiales han sido utilizadas en campos prácticos como ingenieria y finanzas haciendo tareas tales como reconocimiento de patrones, identificación, clasificación y sistemas de control entre otras [4].

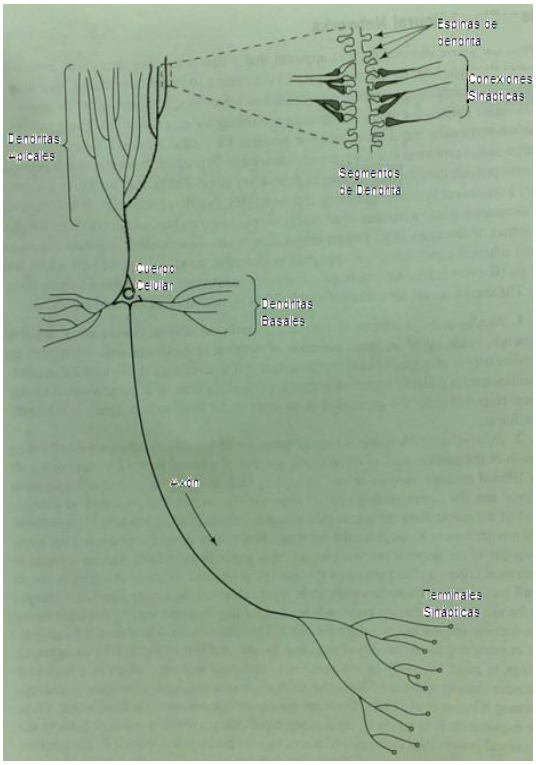

Figura 1.1: Neurona piramidal

El modelo neuronal consta basicamente de cinco elementos: la entrada, la función de transferencia, dos parámetros libres llamados “pesos” y “bias” y el objetivo al cual se quiere llegar.

La salida de la neurona es comparada para ser igual, o tan parecida como se pueda, al objetivo; para lograr la igualdad lo que se hace es que dada una entrada se ajustan los dos parámetros libres para que al pasar por la función de transferencia obtengamos el valor deseado. Es decir, si tenemos una entrada p, un peso w, un bias b, una función de transferencia f y una salida a como en la figura 1.2, buscamos que la salida $a = f(wp+b)$ sea igual al objetivo, ya que de lograrlo, se puede hacer que la red aprenda a realizar algún trabajo en específico. En nuestro caso, reconocer patrones dentro de distintos tipos de series de tiempo.

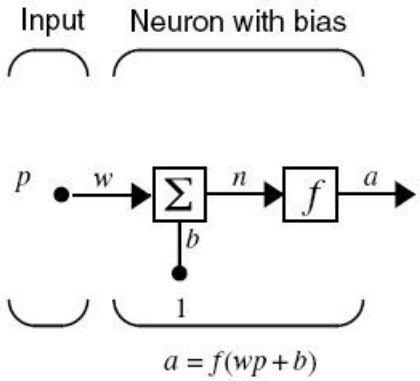

Figura 1.2: Unidad Neuronal con parámetros

Por la discución anterior, ahora sabemos que la alta efectividad de las redes neuronales para resolver problemas, radica básicamente en dos puntos, su estructura en paralelo, y su capacidad de aprender y generalizar un problema generando resultados rasonables a pesar de mostrarles un comportamiento nunca antes visto en las variables que lo afectan.

Existen varios tipos de funciones de transferencia, pero generalmente se usan funciones escalón o sigmoides; dependiendo de la tarea a la que se vaya a entrenar la red. Las más importantes son la función escalón, la recta identidad, y la función log-sigmoide como se muestra en la figura 1.3.

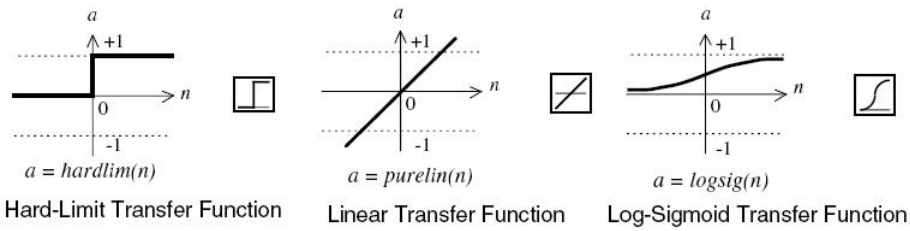

Figura 1.3: Funciones de transferencia más importantes;(a)Escalón, (b)Recta Identidad (c)Log-Sigmoide, de izquierda a derecha

Las funciones escalón son especialmente útiles cuando se quiere entrenar

a la red para hacer clasificación, puesto que si tenemos un conjunto de elementos que queremos separar a partir de una característica en particular, simplemente se le asigna uno de los valores del escalón a los que tengan la carácterística deseada y el otro valor a los que no la tengan. Por ejemplo, pensando en la función escalon mostrada en la figura 1.3, podríamos querer clasificar un grupo de personas en zurdos y diestros, simplemente asignando un valor $n = 1$ a la característica de ser zurdo y $n = -1$ a la de ser diestro. De esta forma al pasar por la función de transferencia se divide el conjunto en dos partes. Pero aún más, esta función se puede usar para clasificar considerando que tan dominante es dicha característica en el conjunto considerado. Por ejemplo, ahora podríamos considerar también sumarle a cada individuo el número de hijos zurdos que tiene y cambiar el valor crítico en el que se da el salto del escalón, de manera que si lo movemos a la derecha una unidad, dividimos al conjunto en los que zurdos, con al menos un hijo zurdo contra el resto. En general, a cada uno de los elementos del conjunto, le podemos medir alguna característica y su importancia, asignandole un valor numérico y cambiando el valor crítico de la función escalón, de manera que se le asigna una constante a todos los elementos con una "medida" mayor al valor crítico y una constante distinta a los que tengan una medida menor, pudiendo dividir un conjunto en dos grupos con una característica que los distinga uno del otro [4]. Lo que sería un funcionamiento similar al utilizado en los circuitos digitales.

Las neuronas con una función de transferencia lineal son usadas normalmente para crear filtros lineales, estos son redes neuronales lineales, que como es de esperarse, solo son útiles para problemas linealmente separables; al igual que las redes creadas con la función escalón, con la ventaja de que con estas neuronas se puede crear un red que se adapte a los cambios que recibe del exterior logrando minimizar mejor el error relacionado con la tarea que realiza. Las aplicaciones más comunes de estas redes son cancelación de errores, procesamiento de señales y sistemas control [4].

Finalmente, la función de transferencia *Log-sigmoide* es la más importante para nosotros, ya que es la que se usa generalmente para los problemas con la regla de aprendizaje *Retropropagación*, como en nuestro caso. Una de las principales razones por la que se usa esta función es porque es diferenciable. Este método de aprendizaje fue usado por distintos grupos de investigación en distintos contextos y fue descubierto y redescubierto, pero no fue si no

hasta 1985 que se formalizó su uso en trabajos de inteligencia artificial; desde entonces ha sido uno de los algoritmos más usados y estudiados en redes neuronales [5]. En la *retropropagación* se entrena la red con las entradas y sus respectivos objetivos hasta que se logra una buena aproximación de la función buscada. Estas redes, con sus pesos, una capa de neuronas con función sigmoide y otra capa con funciones lineales son capaces de aproximar cualquier función mientras tengan un número finito de discontinuidades [4].

La razón por la que se usan funciones sigmoides como funciones de transferencia o de activación para la *retropropagación* es que este algoritmo busca minimizar la función de error para los pesos mediante el cálculo del gradiente en dirección descendiente. Como se debe calcular el gradiente de la función de error en cada iteración, tenemos que garantizar su continuidad y diferenciabilidad en todo momento, lo cual no se cumple con la función escalón. La expresión general de la sigmoide es $s_c(x) = \frac{1}{1+e^{-cx}}$ y dependiendo de la constante c que se escoja, la función se comporta como se puede ver en la función superior izquierda de la figura 1.4.

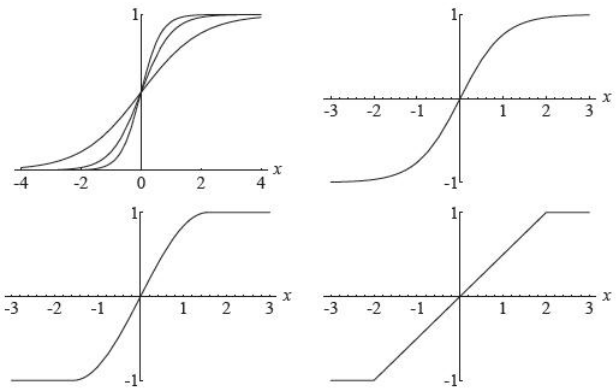

Figura 1.4: Cuatro tipos de funciones sigmoides

Entre más grande es el parámetro c, más cercana es la sigmoide a la función escalón. Por otro lado, si nombramos $s(x) = s_1(x)$ podemos expresar la derivada en términos de la misma función de la siguiente forma:

$$\frac{d}{dx}s(x) = \frac{e^{-x}}{(1+e^{-x})^2} = s(x)(1-s(x)).$$

Por razones de optimización en el aprendizaje, a veces es útil definir como función de transferencia una sigmoide simétrica como la que se muestra en la parte superior derecha de la figura 1.4. La ecuación correspondiente es $S(x) = 2s(x) - 1 = \frac{1-e^{-x}}{1+e^{-x}}$, que corresponde a la tangente hiperbólica pero con argumento $x/2$. Las otras dos funciones también pueden ser usadas como funciones de transferencia, pero si se usara una función como la que se muestra en la parte inferior derecha de la 1.4 habría que tener cuidado con no acercarse a los dos puntos donde se hacen los picos, pues la derivada no está definida. La derivada de la sigmoide es un 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 este 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 error y dejarla rodar sobre ella hasta encontrar el mínimo [5]. Esto simpre nos lleva en una dirección más o menos correcta, porque la derivada de la sigmoide siempre es positiva, marcandonos la dirección adecada para movernos; con la desventaja de que algunas veces el gradiente es muy grande y otras muy pequeño, haciendo que pueda resultar un poco complicado seguir la dirección óptima.

Capítulo 2

Mapeo Logístico

Para introducir la ecuación logística podemos comenzar preguntandonos por la evolución de la población de una especie. Como primer intento, podemos pensar que entre más individuos existan mayor será el aumento de la población, es decir, si en promedio por cada individuo hubiera un aumento κ, nuestro primer modelo para el cambio de población resultaría ser

$$\begin{aligned} x(n+1) - x(n) &= \kappa x(n) \\ \Rightarrow x(n+1) &= (\kappa + 1)x(n), \end{aligned}$$

donde "x" es la cantidad de individuos en la población. En este caso es sencillo observar que al iterar la regla de correspondencia, llegamos a que

$$x(n+1) = (1+\kappa)^n x(n).$$

Y suponiendo que inicialmente tenemos X_0 individuos, es claro que el crecimiento de la población es exponencial, pues se obtiene $x(n+1) = X_0(1+\kappa)^n$, que en cualquiera de los casos posibles ($\kappa > 0$ o $\kappa < 0$) se ve un crecimiento exponencial [3]. Pero a cualquiera le parecería extraño que la población pudiera crecer desmesuradamente y de forma tan rápida, pues debe llegar un momento en el que el alimento empieze a escasear y debido a la competencia la población no pueda aumentar de igual manera o hasta empiece a disminuir. La forma más simple de resolver este problema es añadiendo otro término lineal. El término correcto a agregar nos lleva la ecuación

$$x(n+1) - x(n) = \kappa x(n)(L - x(n))$$

Donde L se puede pensar como el máximo número de individuos que puede haber en la población. Ecuación que haciendo un cambio de variable se puede llevar a la forma de la ecuación (1.1) o una forma más abreviada,

$$x_{n+1} = \mu x_n(1 - x_n). \tag{2.1}$$

Con esto hemos llegado al modelo de este sistema, que es representado por iteraciones de la función $f_\mu(x) = \mu x(1-x)$. A este mapeo f se le conoce como *mapeo logístico* (o familia logística debido al parámetro μ). Esta familia de ecuaciones, por sencilla que parezca, resulta tener un comportamiento que puede llegar a ser tan complicado como uno quiera. Para familiarizarnos con este tipo de funciones, primero trabajaremos con un conjunto muy similar a la logística, la "*familia cuadrática*", que está definido por ecuaciones del tipo

$$Q_c(x) = x^2 + c. \tag{2.2}$$

Podemos esperar que sus comportamientos sean muy parecidos por ser las dos ecuaciones de segundo grado. Y si hacemos un estudio un poco más profundo de este conjunto de ecuaciones podemos encontrar que su semejanza yace en su tendencia a un comportamiento caótico. Para mostrar esto, podemos hacerlo gráficamente, que talvez no sea el camino más formal pero creo que puede resultar ser el más sencillo.

Primero hay que aclarar la representación gráfica de iterar una función. Sabemos que el iterar n-veces la función $f(x)$, es como pensar que en la composición de $f(x)$ con sigo misma, lo que gráficamente se hace proyectando verticalmente el valor de x_i en la función que nos interesa para luego proyectarlo horizontalmente sobre la función identidad y finalmente volver a proyectarlo sobre la función que se está iterando. Lo que se hace es dibujar en una misma gráfica la función *identidad* y la función que queramos repetir. Lo siguiente es escoger un punto x_0 que es del que se va a iniciar la serie. Al proyectarlo verticalmente sobre la función que se le quiere aplicar, lo que se está haciendo es obtener el punto $(x_0, f(x_0))$. Si ahora proyectamos horizontalmente este punto sobre la identidad, vamos a obtener el punto $(f(x_0), f(x_0))$ y ahora solo hay que pensar este punto como si hubieramos empezado desde él, entonces llamamos $x_1 = f(x_0)$ y para lograr la primera iteración de la función solo hay que volver a proyectarlo verticalmente, obteniendo así el punto $(x_1, f(x_1)) = (f(x_0), f(f(x_0))) = (f(x_0), f^2(x_0))$. Haciendo esto muchas veces, se obtiene una imágen como la de la figura 2.1.

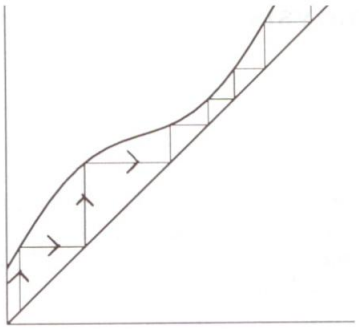

Figura 2.1: Análisis Gráfico

Al camino formado por estos puntos lo llamamos la *órbita* de $f(x)$. Cuando se itera una función con sigo misma, puede aparecer cierto tipo de puntos en los que el comportamiento de la órbita es particularmente disitinto al resto de los de la función; si al aplicar la función $f(x)$ a x_0 tenemos que $x_0 = f(x_0) = f^2(x_0) = ... = f^n(x_0)$ decimos que este es un punto fijo [6]. Además el comportamiento alrededor de estos puntos se puede dividir en tres tipos, atractor, repulsor o punto neutro, entendiendo que si es del primer tipo, la serie de puntos $x0$, $f(x_0)$, ...,$f^n(x_0)$ tiende al punto fijo y si es del segundo tipo, los puntos de la serie son cada vez más lejanos de él. Para definirlos con mayor formalidad decimos que si x_0 es un punto fijo de la función $f(x)$ entonces es *atractor* si $|f'(x_0)| < 1$, *repulsor* si $|f'(x_0)| > 1$, y *neutro* o *indiferente* si $|f'(x_0)| = 1$ [7].

Es importante mencionar esto, porque aunque para este trabajo no es tan relevante conocer este tipo de puntos, el análisis gráfico es especialmente útil para observar de manera sencilla este tipo de comportamientos. La razón de esto es que para encontrar analíticamente los puntos fijos de una función, hay que resolver la ecuación $f(x) = x$ que es justamente la definición de punto fijo. Pero hacer esto gráficamente implica únicamente buscar los puntos donde hay intersección de la función y la identidad; y en muchos casos ni si quiera es necesario calcularlos para saber que existen. Por ejemplo, en la figura 2.2 se grafíca la función $f(x) = \sqrt{x}$ que sabemos intersecta a la recta identidad en "0" y "1" y si iteramos esta función en estos puntos vemos que no cambian.

Además, al dibujar la órbita de $f(x)$ también podemos ver si los puntos

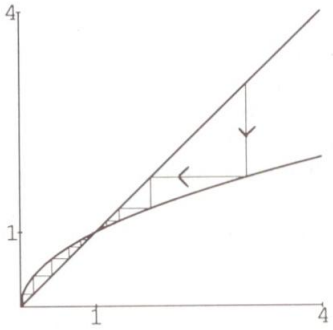

Figura 2.2: Puntos fijos de $f(x) = \sqrt{x}$

fijos son atractores o repulsores. Como en la figura 2.2 donde podemos ver que el *cero* es un atractor y el *uno* un repulsor. Ahora nos enfocamos en la *familia cuadrática* para entender un poco mejor su dinámica buscando primero que nada, puntos fijos. Del análisis gráfico vemos que existe un valor crítico de c tal que si estamos por arriba de él, no hay puntos fijos y todas las órbitas tienden a infinito (como en la figura 2.3a); si es es exactamente el valor crítico $c^{\dagger}$ entonces solo hay un punto fijo y para valores menores que $c^{\dagger}$ tenemos dos, y puede ser probado que con esto ahora aparece también un atractor, como podemos ver en la figura 2.3b.

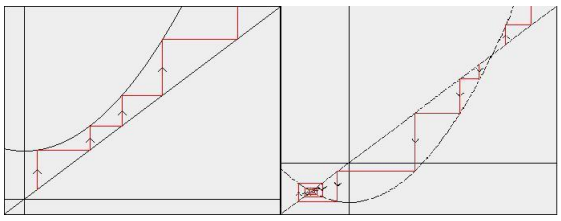

Figura 2.3: (a)No hay puntos fijos (b)Aparecen dos puntos fijos

Para encontrar el valor de $c^{\dagger}$ primero hay que buscar los puntos fijos de la familia cuadrática Q_c. Así que igualando la ecuación (3.4) con la función

identidad, vemos que necesitamos que

$$Q_c(x) = x^2 + c = x \Rightarrow x^2 - x + c = 0, \tag{2.3}$$

cuyas soluciones son:

$$p_+ = \frac{1}{2}(1 + \sqrt{1 - 4c}),$$
$$p_- = \frac{1}{2}(1 - \sqrt{1 - 4c}).$$

Ahora es fácil ver que si $1 - 4c < 0$ las raices no son reales y por lo tanto no tendremos puntos fijos, además al evaluar la derivada de Q_c verémos que todos los puntos son repulsores. Por otro lado, si $1 - 4c = 0$ solo tendrémos un punto fijo, que al ser evaluado en la derivada $Q_c'(x)$ resulta ser neutro. Y finalmente, si $1 - 4c > 0$ tendrémos dos puntos fijos, en los que evaluando la derivada vemos que si $c < 1/4$ entonces p_+ es repulsor y si $-3/4 < c < 1/4$ entonces p_- es atractor, pero si $c < -3/4$ p_- también es repulsor. Deteniendonos a observar lo que acaba de ocurrir con los puntos fijos, vemos que de tener solamente uno con $c = 1/4$, este es transforma en dos puntos fijos. A este comportamiento se le llama *bifurcación*. Ahora sabemos como es la dinámica de f_c cuando $c > -3/4$, pero ¿que pasa si ahora analizamos Q_c^2?. Buscando puntos fijos, sabemos que p_+ y p_- son factores, entonces también lo es $(x - p_+)(x - p_-) = x^2 - x + c$ así que para buscar nuevos puntos fijos hay que resolver

$$\frac{Q_c^2}{x^2 - x + c} = x^2 + x + c + 1 = 0.$$

Las soluciónes a esta ecuación son los puntos $q_\pm = \frac{1}{2}(-1 \pm \sqrt{-4c - 3}$, que también son puntos fijos pero de peridodo dos. Lo que nos da una nueva bifurcación; evaluando las derivadas, se puede comprobar que estos dos puntos son atractores en el intervalo $-5/4 < c < -3/4$ -como podemos ver en la figura 2.4- y repulsores si $c < -5/4$.

Es inmediato suponer que de mantenerse este comportamiento, el periodo de los ciclos conforme se avanza en el número de iteraciones, pueda llegar a ser muy grande, ¿ Pero qúe tan grande?. Para saber esto, nos fijamos en la función $Q_{-2}(x)$ y hacemos el análisis gráfico de ella y sus primeras iteraciones

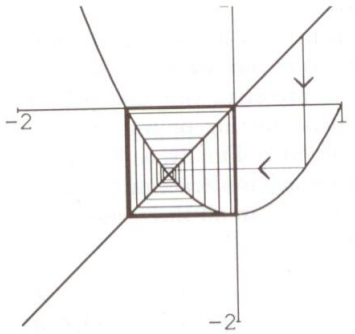

Figura 2.4: Puntos fijos de periodo dos

[6]. Como podemos ver en la figura 2.5a, esta función en específico, proyecta el intervalo I=[-2,2] en él mismo, pero con la particularidad de que cada punto -con excepción del "-2"- tiene dos preimágenes, pues tanto el intervalo [-2,0] como el [0,2] cubren totalmente a I al aplicarles $Q_{-2}(x)$. Esto significa que tras la segunda iteración de la función, ya solo necesitamos 1/2 de I para cubrir completamente la imágen, por eso ahora la figura 2.5b tiene dos vayes en vez de uno, como la figura 2.5a. De la misma manera, argumentamos que tras la tercera iteración, tendrémos cuatro valles en vez de dos, tal como se ve en la figura 2.5c, y al fijarnos en el número de veces que se cruzan estas funciones con la función identidad, podemos contar fácilmente el número de puntos fijos que hay tras cada iteración, demostrando de esta forma que la función $f^n_{-2}(x)$ tiene al menos 2^n puntos fijos de periodo n en el intervalo [-2,2].

Tal y como dijimos en un principio, esta familia tiene un comportamiento muy parecido al de la familia logística, y se puede hacer un análisis similar al anterior para la función (2.1) pero usando $\mu = 4$. En la figura 2.6 se puede apreciar lo complicada que puede llegar a ser la órbita de algunos puntos de esta función, ya que al igual que con la familia cuadrática, también se pueden llegar a tener órbitas de periodo n.

Otra forma de apreciar la evolución de la familia logística, es graficando sus puntos fijos contra los distintos valores de μ; haciendo esto, en el intervalos 2,4 a 4 se obtiene una gráfica como la de la figura 2.7. Aquí podemos observar el rápido aumento de puntos fijos conforme se hace más grande el

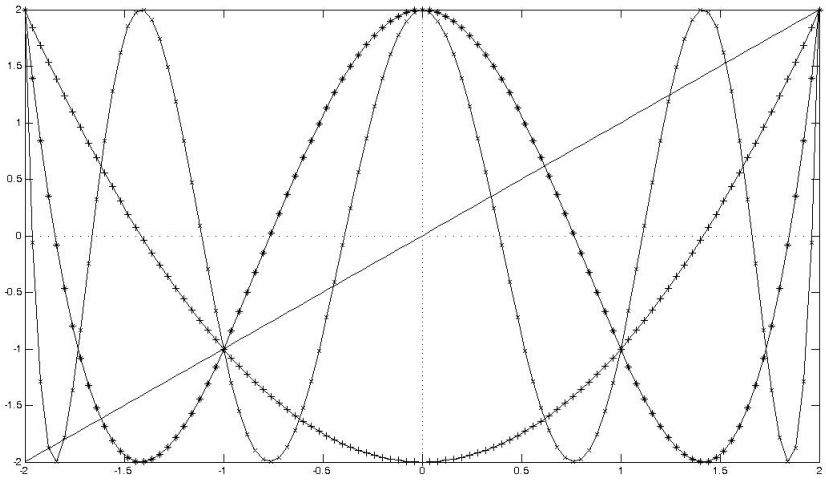

Figura 2.5: (a)Función cuadrática (+). (b)Primera iteración de la función cuadrática (*). (c)Tercera iteración de la función cuadrática (x).

parámetro μ. En este tipo de gráfica, también es posible hacer acercamientos de diversas regiones y observar una figura prácticamente idéntica a la original, como pasaría en un fractal. Es importante notar, que mientras el parámetro libre permanezca en el intervalo $[0, 4]$ todos los puntos $x \in [0, 1]$ van a permanecer dentro de este mismo intervalo, pero si hacemos que $\mu > 4$ vamos a empezar a tener un comportamiento distinto, pues ahora empieza a haber puntos tales que al iterar la función tienen una órbita que tiende a infinito. Esto nos lleva a hablar de otra caracterstica muy interesante de la familia logstica, que es su comportamiento caótico para el rango de valores del parámetro $\mu > 4$. Pero ¿qué es el caos?. Para dar una definición formal del caos, tenemos que empezar definieniendo algunos conceptos necesarios para entenderlo, así que los explicaremos brevemente.

Primero que nada, hay que ver que no todos los puntos del conjunto $I = [0, 1]$ tengan órbitas que tiendan a infinito, pues esto ya no sería interesante. Para ver esto, podemos fijarnos que es lo que está pasando gráficamente. Al tener f_μ con $\mu > 4$, caemos en un caso donde cualitativamente hablando, la función se ve como en la figura 2.8. Lo más importante de esta

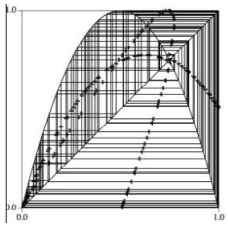

Figura 2.6: Análisis gráfico de una órbita de $4x(1-x)$

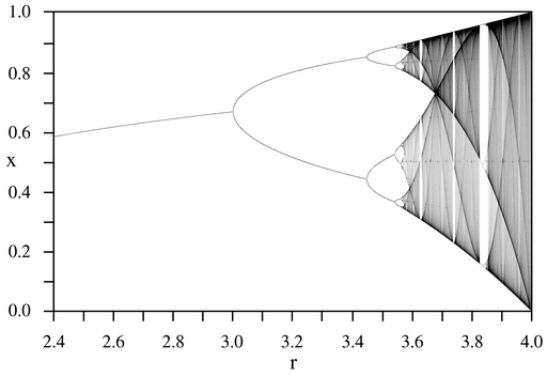

Figura 2.7: Puntos fijos de la familia logística vs. r= $\mu \in [2{,}4, 4]$

figura, es que con ella podemos ver que los intervalos I_0 e I_1 volveran a ser mapeados sobre el intervalo I, por lo tanto hasta la primera iteración, la óbita de estos puntos no dejará la zona de interés y aún no es posible saber si tenderán a infinito. Mientras tanto, los puntos del intervalo A_0 quedan fuera de I de inmediato y su órbita tiende a infinito.

Recordando ahora la figura 2.5, podemos imaginarnos como se ve esta misma evolución para la logística con parámetro $\mu > 4$. Lo que nos lleva a darnos cuenta que para la segunda iteración ahora hay dos intervalos más, que también serán mapeados al intervalo A_0, y si seguimos iterando la función, volveremos a encontrar nuevos intervalos que serán mapeados eventualmente al A_0. También notemos que estos intervalos son cada vez más pequeños. Esto se asemeja mucho al conjunto de los *tercios intermedios de Cantor*. Que consiste en una función que toma un intervalo dado de números reales, lo

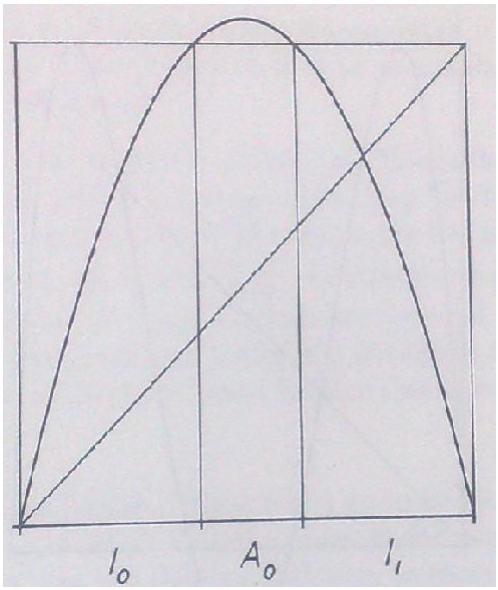

Figura 2.8: La logística con $\mu > 4$ en el intervalo $I = [0, 1]$

divide en tres partes iguales y quita del conjunto al intervalo del centro. Al iterar esta función sobre los nuevos intervalos, van quedando conjuntos de intervalos cada vez más pequeños pero cada vez más abundantes, además de siempre ser cerrados, tal como en el caso de la logística. El objetivo de hacer esta comparación, es que en cualquier curso de cálculo, se ve que el conjunto de puntos que queda de iterar esta función no solo no esta vacío, sino que es infinito. Así que aceptando que el comportamiento de estas dos funciones es análogo, ahora sabemos que el conjunto de puntos en I_0 e I_1 cuya órbia siempre está contenida en el intervalo I, no está vacío.

Ahora retomamos los conceptos necesarios para entender la definición de caos; el primero es tener un sistema dinámico que sea muy sensible a las condiciones iniciales. Lo que quiere decir que no importa cuan parecidas creamos que son las órbitas de dos puntos en un principio, siempre vamos a encontrar una iteración n o algún tiempo t y algún punto para el cual las órbitas ya no son en absoluto cercanas. La forma matemática de pedir esto, se reduce simplemente, a encontrar una distancia "δ" mayor que cero -para una métrica dada previamente- tal que siempre exista un punto muy cercano

al que nos interesa, que cumple con la condición de que para un número finito de iteraciones, la distancia entre las dos órbitas es mayor que "δ".

El segundo concepto necesario para definir el caos, es la *transitividad topológica*. Para este concepto necesitamos pensar en cuales quiera dos conjuntos abiertos dentro del dominio de la función. Si existe algún número finito de iteraciones tal que al aplicarle la función a uno de los conjuntos, resulta que su intersección es distinta del vació, significa que el sistema es topológicamente transitivo.

Y el último concepto que hay que recordar es el de densidad de un conjunto A dentro de otro más grande B. Que requiere que para todo punto en B, y para cualquier vecindad de tamaño δ, existe un punto en A que también está dentro de la vecindad de tamaño δ.

Ahora, si tenemos un conjunto V y una función $f : V \to V$. Decimos que el sistema es caótico si se cumplen las tres condiciones anteriores, es decir [7]:

1.- f es muy sensible a las condiciones iniciales.

2.- f es topológicamente transitiva.

3.- Los puntos periódicos son densos en V.

Es posible demostrar que en el caso $\mu > 4$ el comportamiento de la familia logística se vuelve caótico. Pues a pesar de que hay muchos valores para los cuales la órbita tiende a infinito, haciendo uso de *Dinámica Simbólica* se llega a que la familia logística bajo esta condición, puede ser tratada como un conjunto más sencillo de analizar, haciendo uso del *mapeo de cambio* -*shift map*-, el cual trabaja sobre un conjunto donde todos los elementos son simples series de *ceros* y *unos*. La ventaja de este conjunto, es que por construcción, cada uno de sus elementos es equivalente a conocer la órbita exacta de alguno de los puntos que no escapan bajo las iteraciones de la logística, lo que es muy conveniente para verificar si el conjunto es caótico; solo hay que definir una métrica adecuada.

Para construir este conjunto, nos fijamos primero en la figura 2.8. Como podemos ver siempre vamos a tener dos intervalos disconexos, I_0 e I_1, así que

a todos los puntos del primer intervalo les asignaremos el número *cero* y a los del segundo el número *uno*. Sabemos que solo los puntos en estos dos intevalos van a seguir quedando dentro del dominio que nos interesa después de la primera iteración, así que el resto podemos olvidarlo, pues su órbita simplemente tiende a infinito. Al haber aplicado por primera vez la función a estos intervalos, tendremos que algunos puntos siguen cayendo en el intervalo en el que estaban anteriormente, pero como la función hace que se pueda mapear completamente el intervalo I con cualquiera de los dos intervalos menores, también tendrémos puntos que cambian de intervalo o que quedan en la región en la que bajo una iteración más abandonan el intervalo I. Nos olvidamos nuevamente de los puntos que tienden a infinito, pero a los sobrantes, les podemos asignar una segunda cifra de acuerdo con el intervalo en el que se encuentren. De manera que a un punto que estuvo ambas veces en el mismo intervalo, lo tendremos clasificado hasta ahora con dos números iguales i.e. "00" o "11", mientras que los que cambiaron de intervalo, ahora los clasificaremos como "01" o "10". Ahora recordamos que el comportamiento de este sistema es como el del conjunto de los *tercios intermedios de Cantor* por lo que sabemos que deben existir una infinidad de puntos cuya órbita nunca tiende a infinito, por lo que siguiendo con el razonamiento anterior la secuencia de números asociada a estos puntos es infinita, pues los podemos iterar una infinidad de veces. De esta forma llegamos a definir el *espacio de secuencias*, $\Sigma_2 = \{s = (s_0 s_1 s_2 ...) | s_j = 0 \text{ o } 1\}$ [7], como las secuencias infinitas que identifican a los puntos que nos interesan del intervalo I.

Hecha esta caracterización se define a $d[s,t] = \sum_{i=0}^{\infty} \frac{|s_i - t_i|}{2^i}$ donde $s, t \in \Sigma_2$ como la distancia entre dos puntos del conjunto, la cual también funciona como métrica sobre Σ_2. Ahora solo nos falta la función para el *mapeo de cambio*. La definimos como $\sigma : \Sigma_2 \to \Sigma_2$ tal que $\sigma(s_0 s_1 s_2 ...) = (s_1 s_2 s_3 ...)$, que simplemente elimina la primera entrada de la secuencia. Con estas definiciones ahora es fácil demostrar varias propiedades de este mapeo, como su continuidad al igual que las tres definiciones necesarias para que un conjunto sea considerado caótico. Y como ambos mapeos son esencialmente iguales, demostrar formalmente el *mapeo de cambio* es caótico implica que también lo es el mapeo logístico.
En nuestro caso no necesitamos trabajar con la logística en el régimen caótico, justamente porque debido a la alta sensibilidad a las condiciones iniciales es imposible hacer predicciónes realmente confiables sobre su comportamiento. Pero es importante conocer como es que se da esta transición, pues en caso

de usar constantes con las cuales hay pocos puntos periódicos, la serie de tiempo artificial no será suficientemente "desafiante" para la red, mientras que de estar usando constantes para las cuales el número de puntos fijos es muy grande, lo natural es esperar que las predicciones que se hacen sobre el sistema dinámico tengan errores mucho más grandes y nunca sepamos si la red está suficientemente preparada para atacar problemas reales. Además, de ser posible, hay que saber exactamente en que momento se da la transición al caos, tal como en el caso del mapeo logístico, ya que podemos evitar completamente el régimen caótico.

Capítulo 3

Reconstrucción de Forzamiento usando Redes Neuronales

Para realizar la reconstrucción del forzamiento, lo primero que hay que hacer es encontrar una función con la que se pueda modelar el comportamiento de la serie de tiempo que estamos analizando. Para esto se usa la red neuronal, que como ya sabemos al ser entrenada irá ajustando sus *pesos* para poder predecir el comportamiento de los datos. Esto significa que se encuentra una relación -con d-dimensiones- del tipo $x_{t+1} = f(\bar{x}_t, \alpha_t)$, donde $\bar{x}_t = (x_t, x_{t-1}, ..., x_{t-d+1})$. El parámetro α_t es el que realmente nos interesa para poder hacer el modelo de la serie no estacionaria, ya que con este parámetro es con el que queremos modelar los efectos ya sea de fuerzas externas o de grados de libertad internos que no son modelados por la función f, puesto que las escalas de tiempo T en las que tienen efectos notorios sobre el sistema dinámico que analizamos son mucho mayores a los intervalos de tiempo usados para la serie [8].

3.1. Una red de ejemplo

Antes de empezar a describir más profuntamente la reconstrucción hacemos una red neuronal sencilla para darnos una mejor idea de que es lo que se está haciendo en el programa cuando se llama a las funciones necesarias para crear, entrenar y simular una red neuronal artificial. En este pequeño ejercicio haremos una red usando funciones gaussianas para aproximar la señal, mejor

conocida como red de funciones radiales (*Radial Basis Function network*) y también intentaremos hacer que aprenda a predecir el mapeo logístico. Más adelante usaremos una red "*Feedforward*" para modelar la logística y otros mapeos, y aunque no es la misma, los resultados deben ser similares.

La pequeña red que vamos a hacer puede usar *gaussianas* normalizadas o no normalizadas, así que como el cambio en la programación de estas funciones es sencillo la haremos de las dos formas para ver en que aspectos se nota la diferencia. La función aproximada, o dicho de otra forma, la función que queremos se asemeje a la logística para nuestro ejemplo, la definimos como:

$$\begin{aligned} \varphi(x) &= \sum_{i=1}^{N} a_i \rho_i(x), \quad \text{donde} \\ \rho_i(x) &= e^{-\beta(x-c_i)^2}, \end{aligned} \tag{3.1}$$

para el caso no normalizado. Y para el caso normalizado tomamos la función aproximada como:

$$\begin{aligned} \varphi(x) &= \frac{\sum_{i=1}^{N} a_i \rho_i(x)}{\sum_{i=1}^{N} \rho_i(x)} = \sum_{i=1}^{N} a_i u_i(x), \quad \text{donde} \\ u_i(x) &= \frac{\rho_i(x)}{\sum_{i=1}^{N} \rho_i(x)}. \end{aligned} \tag{3.2}$$

En estas ecuaciónes, el parámetro N nos indica el número de gaussianas que estamos usando para aproximar, y por lo tanto el número de datos de la serie que usaremos como centros para estas funciones.

De manera similar al razonamiento usado en las series de Fourier, para lograr que estas funciones se asemejen a la logística, necesitamos encontrar los valores de los pesos a_i. Y para esto podemos usar dos métodos de entrenamiento, uno siguiendo la dirección en la que desciende el gradiente de la función $H_t(a_i)$ que queremos aproximar, fijandonos en su dependencia de los pesos. De manera que

$$a_i(t+1) = a_i(t) - \nu \frac{d}{da_i} H_t(a_i),$$

nos sirve para actualizar los pesos en cada iteración. En esta ecuación a la constante ν se le conoce como el *parámetro de aprendizaje*, que nos indica de que tamaño daremos el paso contra el gradiente de la función. Es importante que no sea ni muy pequeño ni muy grande, ya que en el primer caso tardaríamos mucho en encontrar el mínimo de la función, que es justo lo que nos interesa; mientras que si es muy grande podríamos nunca encontrarlo por estar siempre dando vueltas alrededor de él.

El segundo método de entrenamiento es usando un operador de proyección [10], con el cual la actualización de los pesos se reduce a calcular

$$a_i(t+1) = a_i(t) + \nu[y(t) - \varphi(x(t), a_i)]\frac{f_i(x)}{\sum_{i=1}^{N} f_i^2(x)}, \tag{3.3}$$

para cada iteración. Donde $f_i(x) = \rho_i(x)$ o $f_i(x) = u_i(x)$ según se trate del caso sin normalizar o del normalizado respectivamente.

Ahora, para crear la red solo necesitamos decidir los valores de unos cuantos parámetros. Por ejemplo, nosotros usaremos $N = 5$, así que también escogemos cinco datos de la serie como centros de nuestras funciones gaussianas. El segundo parámetro a escoger es el parámetro de aprendizaje, que tomamos como $\nu = 0{,}3$. Y finalmente tenemos que escoger una serie de números que nos servirá como pesos iniciales para las ecuaciones (3.1) y (3.2) según sea el caso no normalizado o normalizado respectivamente.

El siguiente paso es entrenar la red para obtener la señal de la logística. Para esto le aplicamos la función $\phi(x)$ al valor x_0 con el que se inició la serie de tiempo original. Una vez obtenido el nuevo valor usamos la ecuación (3.3) para actualizar los pesos y mejorar el desempeño de nuestra función. A este proceso, tal como fue descrito se le cuenta como una época, y al final de esta debemos tener un conjunto de pesos más adecuados que los anteriores para predecir el siguiente valor al que queremos acercarnos. Hacemos esto para una serie de 50 datos y como podemos ver en la figura 3.4, con el entrenamiento se logra reconstruir aceptablemente bien la señal.

En esta misma figura, podemos apreciar que claramente al usar las funciones normalizadas se logra entrenar mucho mejor a la red, aún con una menor cantidad de épocas. Para no hacerlo a simple vista, calculamos error

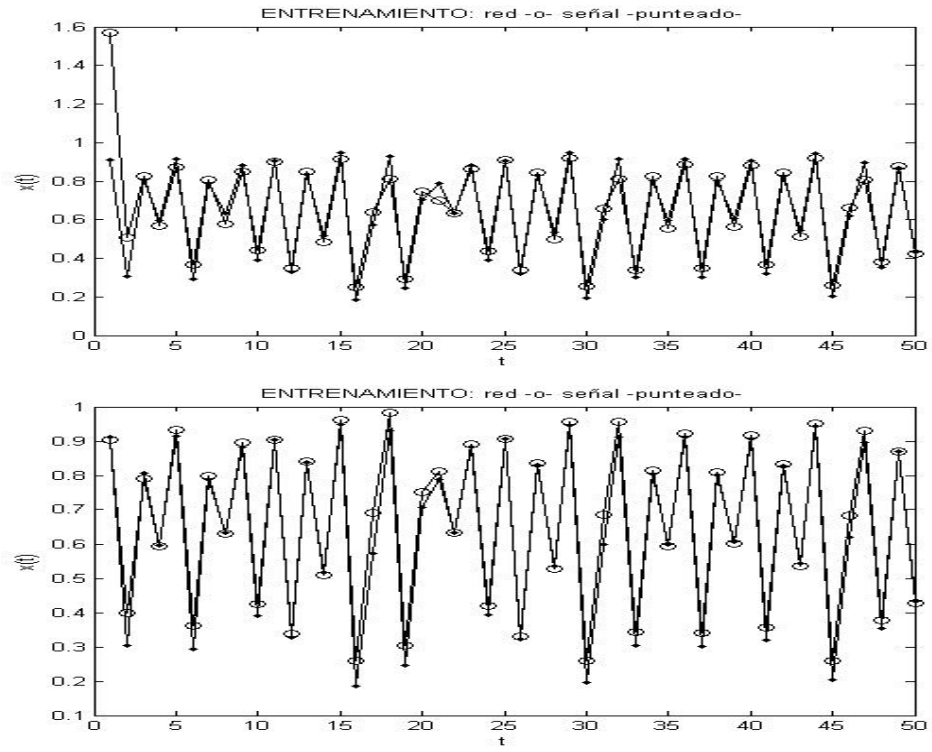

Figura 3.1: Desempeño del entrenamiento para la red con funciones sin normalizar (arriba) y funciones normalizadas (abajo).

cuadrático medio de ambas señales obteniendo que para el caso sin normalizar es 0.0951 mientras que para las funciones normalizadas es 0.0044, notando que para esta última, solo se realizaron dos épocas con la finalidad de apreciar ambas series de tiempo en la imágen, ya que al usar 5 épocas el error se vuelve del orden de 10^{-8} haciendonos imposible notar la diferencia entre ambas señales.

El último paso es el de la predicción, que en mi opinión es el más importante, pues lo que queremos es predecir la señal adecuadamente, sin importar tanto que tan bueno haya sido el entrenamiento. En la figura 3.2 podemos ver las predicciones hechas por ambas redes, siendo prácticamente las mismas. En estos casos no cambia notablemente si usamos 2, 5, o 10 épocas durante el entrenamiento. Como estamos usando una red muy pequeña, y únicamente

con un parámetro por neurona, las predicciones no logran aproximar bien la función más allá de diez datos al futuro.

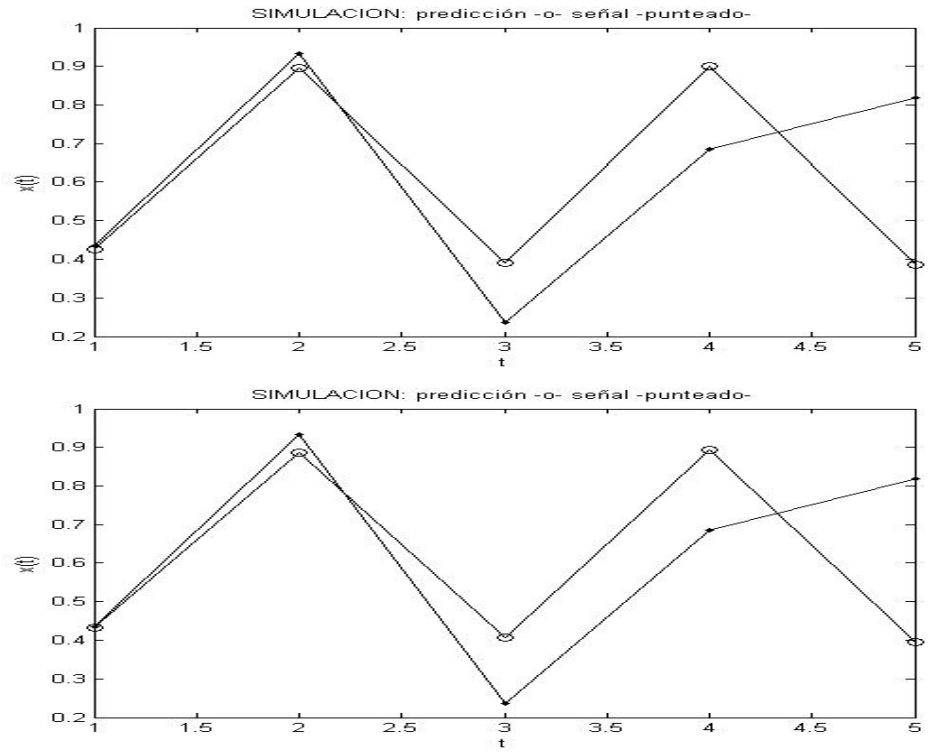

Figura 3.2: Simulación de la logística para la red con funciones sin normalizar (arriba) y funciones normalizadas (abajo).

El error en estas predicciones fue de 0.0522 para la función sin normalizar y de 0.1274 para la normalizada. Con la diferencia de que al disminuir las predicciones a 5, o hasta 3 datos, el desempeño de ambas, en especial el de la función normalizada, mejora notablemente. Un punto importante es que aquí se está mostrando la señal producida por la logística si ruido ni forzamiento, pero al tomarlos en cuenta, mejoran las dos simulaciónes, pero la de la función normalizada además supera la de las funciones sin normalizar. Usando las redes de matlab, en las que se usan métodos de entrenamiento mucho más sofisticados, y algunos parámetros más, es de esperarse que las predicciones sean mejores aún considerando puntos mucho más lejanos. Pero

este ejemplo nos sirve para darnos cuenta de lo complejo que se vuelve el problema, aún con las restricciones que le impusimos a los parámetros de nuestra red para simplificarla y mantener un ejemplo sencillo.

3.2. El algoritmo de reconstrucción

El modelo para la reconstrucción consiste en separar los datos en N_{int} intervalos de longitud M, de manera que $M \ll T$ para poder simplificar nuestro modelo a uno donde el parámetro α_t es constante. La forma más natural de definir esta constante sería como el promedio del parámetro α_t correspondiente a cada valor del mismo intervalo. Por lo tanto, para el m-ésimo intervalo, tendríamos que $x_{t+1}^{(m)} \simeq f(\bar{x}_t^{(m)}, \alpha^{(m)})$ con el parámetro α tomando en cuenta las consideraciones anteriores. Hecho esto, hacemos la aproximación $x_{t+1}^{(m)} - f(\bar{x}_t^{(m)}, \alpha^{(k)}) \simeq \frac{\partial f}{\partial \alpha}(\bar{x}_t^{(m)}, \alpha^{(k)})(\alpha^m - \alpha^k)$ con los intervalos m y k suficientemente cermanos. Haciendo un simple promedio sobre el tiempo, renombramos esta igualdad como

$$E_k^m = A_k^m \Delta\alpha_k^m, \tag{3.4}$$

donde:

$$\begin{aligned} E_k^m &= \langle x_{t+1}^{(m)} - f(\bar{x}_t^{(m)}, \alpha^{(k)}) \rangle_t, \\ A_k^m &= \langle \frac{\partial f}{\partial \alpha}(\bar{x}_t^{(m)}, \alpha^{(k)}) \rangle_t, \quad \text{y} \\ \Delta\alpha_k^m &= \alpha^{(m)} - \alpha^{(k)}. \end{aligned}$$

Para no confundirnos en esta notación hay que notar que el índice libre es m mientras que el índice k simplemente nos señala que intervalo estamos usando como referencia para nuestra aproximación. Es decir, el error E_k^m es el promedio de la resta de los datos conocidos del m-ésimo intervalo menos los datos obtenidos para este mismo intervalo pero usando la predicción de la red entrenada con los datos del k-ésimo intervalo. Ahora, como aún suponiendo que nuestra serie de tiempo cumple con todas las condiciones necesarias para que la red neuronal pueda hacer un buen modelo, la ecuación (3.4) no tiene solución para $\Delta\alpha$, pues no tenemos manera de conocer A_k^m. Sin embargo, como $\frac{\partial f}{\partial \alpha}(\bar{x}_t^{(m)}, \alpha^{(k)})$ y $\frac{\partial f}{\partial \alpha}(\bar{x}_t^{(m)}, \alpha^{(m)})$ son aproximadamente iguales, puesto que deben tener el mismo orden de magnitud debido a que suponemos que α cambia lentamente, podemos intercambiar estos términos para poder resolver el

sistema.

Dicho esto podemos olvidarnos del subíndice k, y si además nos fijamos solamente en los intervalos contiguos -i.e. $m = k + 1$- la ecuación (3.4) se vuelve

$$E_k^{k+1} = A^{k+1}\Delta\alpha_k^{k+1}. \tag{3.5}$$

Pero esto también se puede hacer entrenando con el intervalo $k + 1$ y prediciendo para el k-ésimo intervalo, de manera que la ecuación

$$E_{k+1}^{k} = A^{k}\Delta\alpha_{k+1}^{k}, \tag{3.6}$$

también es válida. Tomando ahora la división de la ecuación (3.5) entre la ecuación (3.6) nos queda la regla

$$\frac{E_k^{k+1}}{E_{k+1}^k} = -1 \cdot \frac{A^{k+1}}{A^k} \quad \text{pues} \quad \Delta\alpha_k^{k+1} = -\Delta\alpha_{k+1}^k.$$

Si utilizamos esta regla sobre la ecuación (3.6) para encontrar el valor de $\Delta\alpha$, usando intervalos con índices cada vez más pequeños, encontramos que la regla de correspondencia para calcularlas se puede generalizar como

$$\begin{aligned}
\Delta\alpha_k^{k+1} &= E_{k+1}^k \cdot \frac{-1}{A^k} \\
&= E_{k+1}^k \cdot \frac{E_k^{k-1}}{E_{k-1}^k} \cdot \frac{(-1)^2}{A^{k-1}} \\
&= E_{k+1}^k \cdot \frac{E_k^{k-1}}{E_{k-1}^k} \cdot \frac{E_{k-1}^{k-2}}{E_{k-2}^{k-1}} \cdot \frac{(-1)^3}{A^{k-2}} \\
&\vdots \\
&= E_{k+1}^k \cdot \frac{E_k^{k-1}}{E_{k-1}^k} \cdot \frac{E_{k-1}^{k-2}}{E_{k-2}^{k-1}} \cdots \frac{E_2^1}{E_1^2} \frac{(-1)^{k+1}}{A^1} \\
&= (-1)^{k+1} \frac{E_{k+1}^k}{A^1} \prod_{r=1}^{k-1} \frac{E_{r+1}^r}{E_r^{r+1}}.
\end{aligned} \tag{3.7}$$

Lo único que no conocemos de la ecuación (3.7) para encontrar $\Delta\alpha$ es el valor de A^1, pero como este solo nos cambiaría la escala de nuestros datos,

podemos escoger cualquier valor, así que por simplicidad lo volvemos *uno*. Ahora tenemos un algoritmo para reconstruir el forzamiento de cualquier serie de tiempo. Simplemente hay que usar la red neuronal para que haga un modelo de nuestro sistema y con base en el análisis hecho anteriormente, solo tenemos que fijarnos en los errores de las predicciones para encontrar los valores de los $\Delta\alpha$'s.

Hasta ahora no hemos hecho ninguna aclaración acerca del programa de redes neuronales. Simplemente hemos dicho que hay que dividir la serie de tiempo en N intervalos de tamaño M, y que para hacer la predicción de los datos del intervalo siguiente -o anterior- al que usamos para entrenar, tomamos como entrada los datos "t", "$t-1$", ..., "$t-d+1$" para obtener como salida el dato "$t+1$". Pero tenemos que recordar que el objetivo de aprender a predecir este tipo de series de tiempo, es aplicarlo a problemas reales, pero en estos, la cantidad de mediciones puede llegar a ser muy poca como para tener suficientes intervalos separados de un tamaño aceptable como para encontrar un verdadero forzamiento; a diferencia de la serie de tiempo artificial, en la que tenemos el control sobre los datos para obtener los intervalos tanto en la cantidad como de la longitud deseada. Por esto, es que es importante tener un programa que también pueda hacer este trabajo pero tomando en cuenta intervalos traslapados. Primero que nada analizaremos que es lo que se puede lograr sin usar intervalos traslapados y luego veremos como es que se afectan los resultados si introducimos esta variante.

El forzamiento de la serie se introduce haciendo que la constante μ de la ecuación (2.1) varie siguiendo la relación

$$\mu(t) = \mu_0 - Kcos(\frac{2\pi t}{T})e^{-\frac{t}{T}}, \tag{3.8}$$

donde "K" es una constante y "T" es tomado como la mitad del total de los datos de la serie. Este forzamiento tiene la forma de un *coseno* cuya amplitud disminuye de forma exponencial, como se puede ver en la figura 3.3, lo que nos permite hacer que la señal sea aún más complicada de lo que sería usando simplemente una constante.

Usamos el mapeo logístico para generar una serie de tiempo, y hacemos dos pruebas para saber que la predicción de la red es aceptable. La primera implica revizar como se ve la señal predicha por la red comparada con el

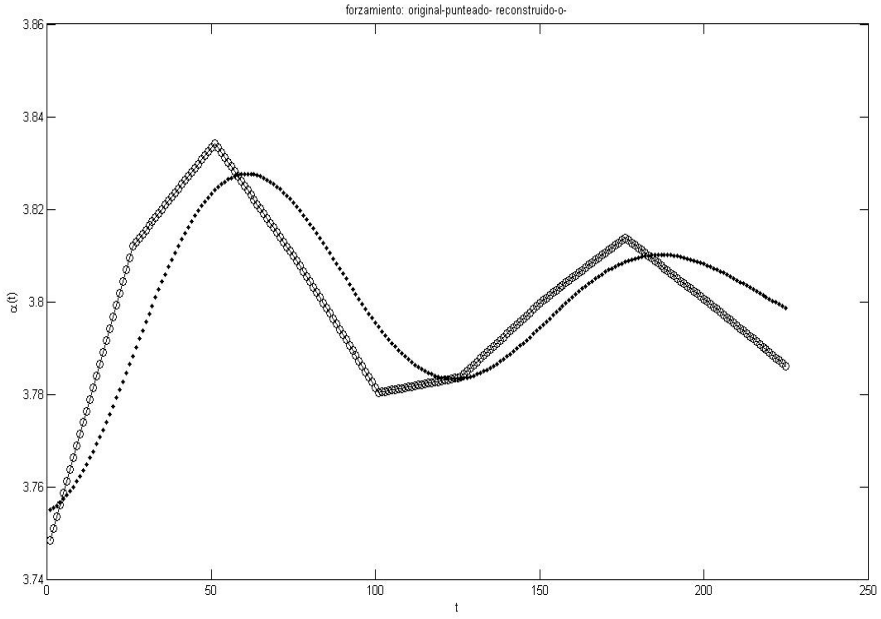

Figura 3.3: Forzamiento y reconstrucción tomando $\mu_0 = 3{,}8$ y $k = 0{,}045$.

objetivo, tanto con las predicciones al futuro como con las predicciones al pasado. Esta prueba es la más sencilla de hacer pues no necesitamos hacer mucho procesamiento de los datos, ya que de cualquier forma hay que obtener ambos vectores para más adelante calcular los errores promedio que necesitamos para la reconstrucción del forzamiento y no hay que esperar hasta que se termine de entrenar la red con todos los datos, porque se pueden observar las predicciones después del entrenamiento de cada intervalo. Así que graficando ambas curvas observamos comportamientos similares al mostrado en la figura 3.4 en el que a simple vista se aprecia que la red funciona correctamente.

La segunda prueba que hacemos para saber que la predicción de datos hecha por la red es adecuada, es hacer la gráfica de los datos x_{t+1} vs. x_t, para observar primero que nada, que se forma la parábola de la función de recurrencia -o la función del mapeo pertinenete-. Graficando tanto la serie como las predicciones obtenemos la figura 3.7, en la que aunque no podemos apreciar que tan buenas son las predicciones cuantitativamente hablando,

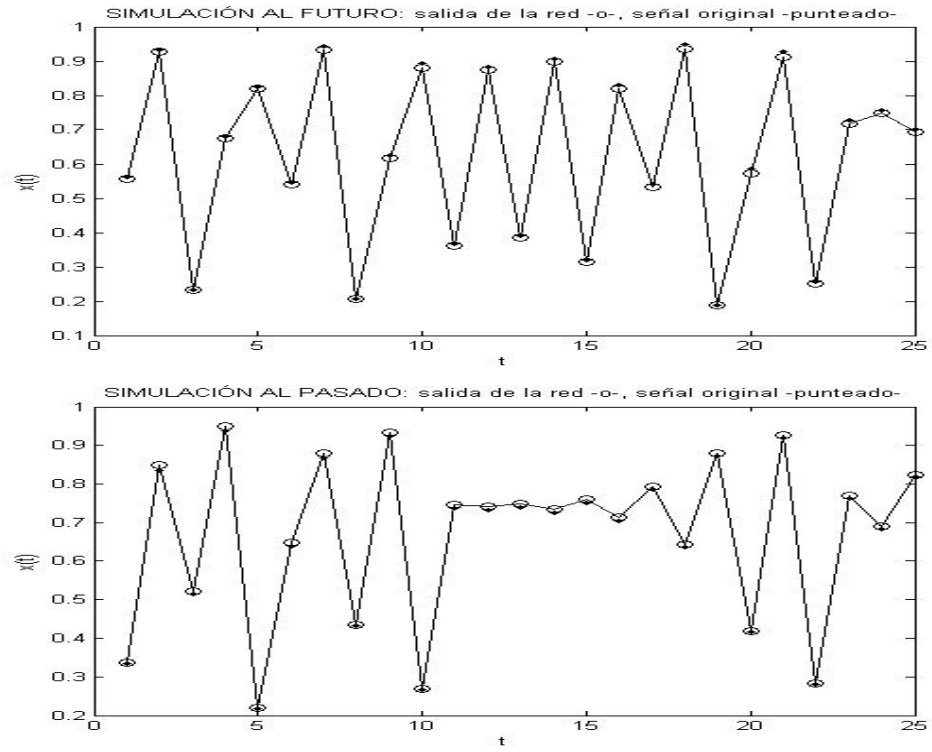

Figura 3.4: Predicción al futuro y al pasado. Figura superior e inferior respectivamente

al menos podemos ver que tienen la suficiente coherencia con la señal que queremos predecir, como para generar casi la misma parábola, lo cual es un detalle muy importante, pues a simple vista con la primera prueba solo podemos decir que las señales son parecidas, pero no tenemos ningún otro punto de comparación como para decir que tan significativa es su cercanía, ya que al parecer señales de ruido podrían no darnos ninguna tendencia clara al graficarlas como se hace en este método. Aunque no podemos saber si los datos predichos que quedan justo sobre los originales corresponden al mismo dato de la señal, el hecho de que las parábolas estén encimadas nos basta para suponer que las predicciones se están haciendo suficientemente bien como para continuar con la reconstrucción y hacer un análisis más cuantitativo.

A pesar de que a simple vista el primer método de comparación no nos

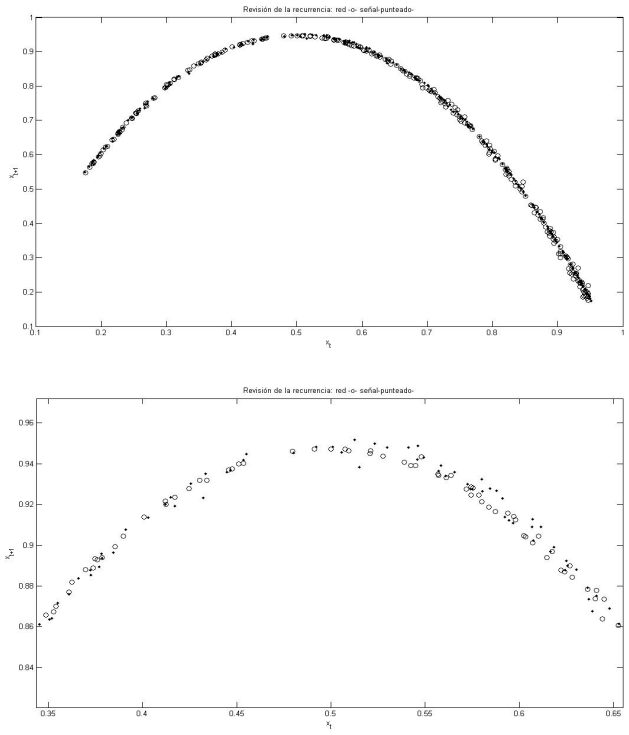

Figura 3.5: Gráfica de x_t vs. x_{t+1} para la señal de la logística.

permite decir mucho, en realidad este tiene una ventaja más importante, y no solo es que, como ya dijimos, no se necesita hacer un gran esfuerzo procesando los datos pues es prácticamente inmediato observar ese tipo de figuras, sino que además nos permite también hacer un análisis cuantitativo de los resultados, pues es muy sencillo calcular el error de la predicción y compararlo con los errores en otras corridas, ayudandonos a decidir cuales son los valores más indicados para los parámetros de la red. Por ejemplo el número de capas y de neuronas que necesita, o bien la cantidad de épocas que hay que entrenarla. Valores típicos de estos errores son 1×10^{-2} para el error cometido en la predicción de la señal completa, mientras que en el entrenamiento de la red se alcanzan desempeños del orden de 1×10^{-4}.

Como podemos ver en la figura 3.3 el forzamiento reconstruido se asemeja mucho al verdadero, pero inmediatamente nos podemos dar cuenta que para

mejorarlo, tendríamos que aumentar el número de intervalos en los que dividimos la señal completa para poder encontrar más pendientes y hacer que nuestra reconstrucción no sea tan "cuadrada" y se parezca más a la curva buscada. El mayor problema que se tiene al querer usar más intervalos es que al tener un número fijo de datos, habría que reducir el tamaño de los intervalos que conforman los vectores de entrada, lo que inmediatamente se reflejaría en una pérdida de precisión en la reconstrucción, pues para la red es más fácil aproximar la señal si tiene 1000 datos, que si tiene 10 datos para entrenarse. Entonces, una forma de evitar la pérdida de información para el vector de entrada, es modificar el programa para que pueda haber traslape entre un intervalo y otro. Pero como podemos observar en la figura 3.6, el resultado de aceptar el traslape es que ahora observamos una señal, que aunque es más suave, también está mucho más descompuesta que antes de usar el traslape. Por ahora dejaremos el traslape de los intervalos a un lado, para probar el funcionamiento del programa con señales generadas por otro tipo de mapeos.

El hecho de que la reconstrucción empeore en vez de mejorar, se lo atribuimos al algoritmo que usamos para calcular el forzamiento, ya que este involucra la división de los errores promedio cometidos por la red. El problema de esto es que al usar traslape, el error que comete la red es más pequeño, pues no todos los datos son desconocidos, pero al ser las predicciones también ligeramente azarosas -pues nunca se hacen dos corridas iguales- los errores relativos varían mucho más, volviendo el algoritmo mucho más inestable [8].

Sin considerar intervalos encimados, sabemos que es posible hacer una reconstrucción bastante buena del forzamiento introducido usando como generador de la señal el mapeo logístico, probaremos el desempeño del programa usando más series generadas artificialmente. El mapeo de Moran-Ricker, usado generalmente para modelar poblaciones de peces, dependiendo de la cantidad de *padres y huevos* disponibles, que tiene como ecuación,

$$x_{t+1} = x_t e^{r(1-\frac{x_t}{k})},$$

donde la cantidad de individuos de la población al siguiente intervalo de tiempo depende, obviamente, de los individuos en el intervalo anterior y donde la constante r se interpreta como el índice de crecimiento y la constante k como el límite de individuos que puede haber en el sistema.

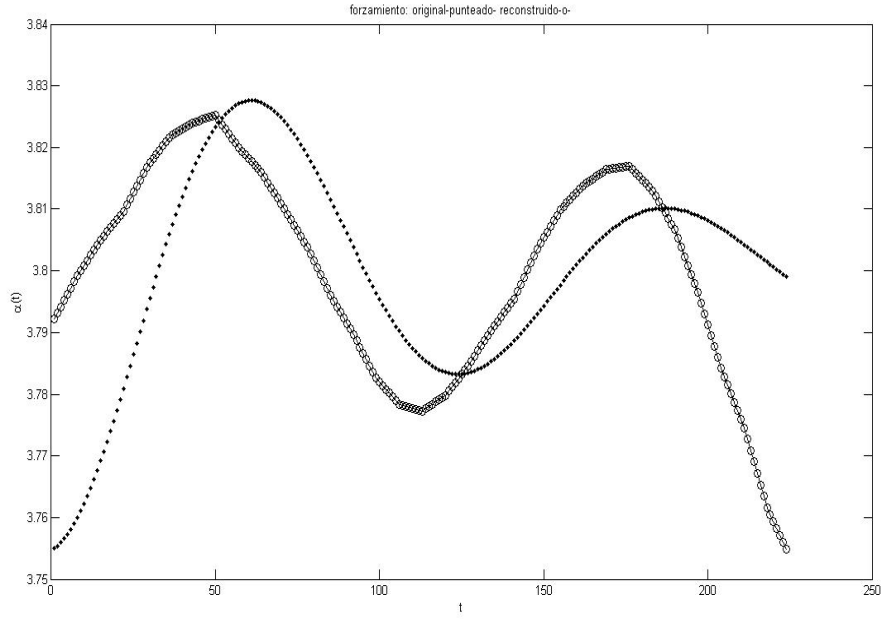

Figura 3.6: Forzamiento y reconstrucción con un traslape del 70 % del intervalo.

También usaremos el mapeo de Hassell, originalmente creado para modelar poblaciónes de insectos, y cuya ecuación es

$$x_{t+1} = \lambda \frac{x_t}{(1 + x_t)^{\beta}},$$

que podría pensarse como una generalización del modelo de Moran-Ricker, pues es una versión limitada del de Hassell, ya que con los parámetros ajustados adecuadamente se obtienen cualitativamente las mismas funciones.

Al igual que en el caso del mapeo logístico, en la figura **??** podemos ver que al graficar los datos x_t vs. x_{t+1} de los mapeos de Moran-Ricker y de Hassell, también se puede predecir con exactitud la señal generada por modelos un poco más sofisticados que el mapeo logístico.

El modelo de Hassell, es un poco más interesante que los anteriores,

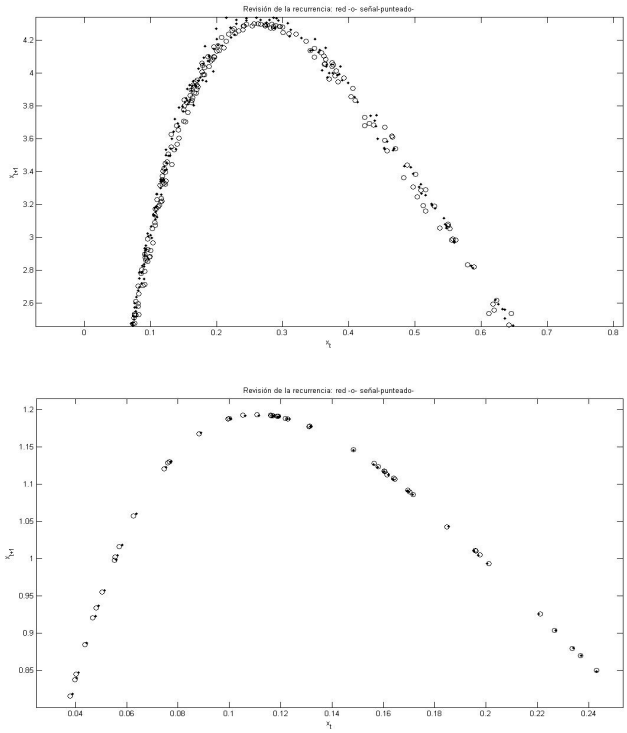

Figura 3.7: Gráfica de x_t vs. x_{t+1} para la señal del mapeo de Moran-Ricker (figura superior) y de Hassell (figura inferior).

porque en este podemos forzar el parámetro λ si queremos revisar la reconstrucción para variaciones lineales, pero también podemos forzar β para ver los efectos no lineales. En ambos casos hemos podido reconstruir la fuerza con éxito, pero como era de esperarse, es un poco mejor para el caso de variaciones lineales.

Todas estas predicciones se han hecho usando una red neuronal de *retroalimentación* -feedforward- pero aparte de esta, también podemos usar funciones de tipo *radial*; que consiste en generar la regla de correspondencia de los datos de la señal a partir de sumar funciones de tipo gaussiana. Con este método, se genera una función con la que la diferencia entre la señal real y la predicha es mucho más pequeña que en el caso anterior.

Capítulo 4

Conclusiones

Como hemos visto en el capítulo anterior, la red que se construyó para analizar las series de tiempo, genera resultados muy buenos tanto para encontrar una función que reproduzca los modelos analíticos que le presentamos, como para encontrar el forzamiento intoducido en esta señal. Pero hasta este momento tenemos dos problemas con estos resultados. El primero es que a pesar de que logramos encontrar el forzamiento con un pequeño error ($\sim 10^{-2}$), este error no es lo suficientemente pequeño como para ser comparado con el modelo que estamos intentando reproducir, en el que para la logística se alcanzan errores mucho más chicos ($\sim 10^{-4}$) [8]. La consecuencia inmediata de este problema es que al probar nuestra red con la serie de tiempo real de la temperatura global, usada en la [11] no podemos reproducir resultados parecidos, debido a que como la red aún no tiene la capacidad de hacer predicciones tan buenas como las deseadas obtengamos forzamientos completamente distintos uno de otro, haciendonos pensar que nuestra red sigue siendo muy inestable. Lo que nos lleva al segundo problema. Para hacer las predicciones de estas series de tiempo, se dispone de muy pocos datos para cada intervalo si se quiere que estn separados. Por lo que como ya habíamos previsto anteriormente, se tiene que modificar el programa para trabajar con intervalos traslapados, metiendonos directamente a buscar una manera de mejorar el problema de la estabilidad de los errores.

En la referencia [8] se presenta una propuesta para suavizar el forzamiento, de manera que se logra que el resultado producido por la red usando traslape sea menor que la que no lo usa, que es justo lo que esperamos, pues implica conocer con más detalle el valor del forzamiento en cada punto. La

propuesta consiste en minimizar el error:

$$\varepsilon = \sum_{r=1}^{n}[\sum_{k=1}^{N_{int}-r} P_r(E_k^{k+r} - A^{k+r}\Delta\alpha_k^{k+1})^2 + \sum_{k=r+1}^{N_{int}} P_r(E_k^{k-r} - A^{k-r}\Delta\alpha_{k-r}^{k})^2] + \eta \sum_{k=1}^{N_{int}-1} \Delta^2 A^k. \quad (4.1)$$

Donde $\Delta A^k = A^{k+1} - A^k$, η es un factor de escala que se usa para igualar los ordenes de magnitud entre las dos partes de la ecuación y el resto de los parámetros son los que ya conocíamos. Lo interesante de este error es que considera los errores cometidos cuando la red es entrenada para predecir los valores de un intervalo que no es inmediatamente aledaño, de manera que al resolver el sistema de ecuaciones para $\Delta\alpha$ se tienen más restricciones, dando como consecuencia una solución más estable. Por eso es que se incluye una función P_r, que se usa para darle mayor o menor importancia a los distintos intervalos dependiendo de su cercanía al intervalo que se usó para el entrenamiento. Para minimizar esta ecuación se usa la técnica de *gradiente descendiente*, en la que tal como su nombre lo dice, se calcula el gradiente del error en un punto inicial, en este caso con los parámetros obtenidos por la red neuronal, y se sigue la dirección opuesta a la que crece el gradiente. Tomando como variables los términos $\Delta\alpha_k^m$ y A^k. Además hay que minimizar el error para muchos valores de η de manera que se pueda apreciar con que valor se tiene el mejor ajuste para la curva del forzamiento.

El caso más simple que se puede tomar para la función de peso es $P_r = \delta_{r,1}$, donde la ecuación (4.1) se convierte en:

$$\varepsilon = \sum_{k=1}^{N_{int}-1} (E_k^{k+1} - A^{k+1}\Delta\alpha_k^{k+1})^2 + \sum_{k=2}^{N_{int}} (E_k^{k-1} + A^{k-1}\Delta\alpha_{k-1}^{k})^2 + \eta \sum_{k=1}^{N_{int}-1} \Delta^2 A^k.$$

Solucionar estos problemas exitosamente, nos permitiría usar este programa no solo para reproducir los resultados ya obtenidos, existe una cantidad enorme de fenómenos sobre los que se tienen series de tiempo y se trabaja en su predicción, así que el perfeccionamiento de esta técnica nos permitiría buscar forzamientos en distintos tipos de fenómenos abriendo también la pregunta, ¿Qué los origina?. Por ejemplo, existen cálculos sobre la potencia promedio de los huracanes en distintas zonas del planeta, pero en términos generales se ha percibido un aumento en ella. Con este trabajo nos gustaría buscar si este aumento en verdad es una tendencia que se pueda percibir como un forzamiento en estos fenómenos meteorologicos, y más aún, si está relacionado con el calentamiento global.

Bibliografía

[1] Edward R. Scheinerman, *Invitation to Dynamical Systems*, Prentice Hall, Inc., (1996).

[2] James T. Sandefur, *Discrete Dynamical Systems*, Oxford University Press, (1990).

[3] Boris Hasselblatt & Anatole Katok, *A First Course in Dynamics with a Panorama of Recent Developments*, Cambridge University Press, (2003).

[4] Howard Delmuth, Mark Beale & Martin Hagan, *Neural Network Toolbox User's Guide*, The MathWorks, Inc., (2008).

[5] R.Rojas, *Neural Networks*, Springer-Verlag, Berlin, (1996).

[6] Robert L. Devaney, *A First Course in Chaotic Dynamical Systems*, Addison-Wesley Publishing Company, Inc., (1992).

[7] Robert L. Devaney, *An Introduction to Chaotic Dynamical Systems*, Second Edition, Addison-Wesley, (1989).

[8] P.F. Verdes, P.M. Granitto, H.D. Navone, y H.A. Ceccatto, Phys. Rev. Lett. 87, 124101 (2001).

[9] Simon Haykin, *Neural Networks*, Macmillan College Publishing Company, (1994).

[10] Wikipedia: Radial basis function network.

[11] P.F. Verdes, Phys. Rev. Lett. 99, 048501 (2007)