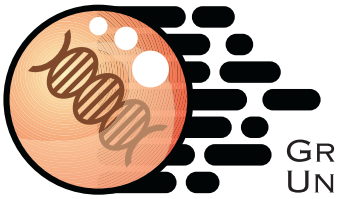

GRUPO DE VIDA ARTIFICIAL
UNIVERSIDAD DE CHILE

Introducción a la Vida Artificial
Apuntes para el Tutorial 2002/2

Pedro Ortega C., <peortega@dcc.uchile.cl>

28 de noviembre de 2002

Índice general

Capítulo 1

Introducción

Una definición:

> La Vida Artificial se suele definir como la ciencia que trata de situar la vida "tal como es" dentro del contexto de la vida "tal como podría ser". (Christopher Langton).

Esta definición la sitúa claramente dentro de las Ciencias de la Vida, y al lado de la Biología. Es decir, la Vida Artificial estudia la Vida como un fenómeno universal, del cual, por el momento, sólo se conoce un ejemplo, la vida en la Tierra. ¿Es este el único tipo de vida posible, o la vida en la Tierra se trata de un "accidente congelado"?

Para ello, por medios teóricos y computacionales, se estudia lo que es común a todos los seres vivos.

La Vida Artificial abarca muchas áreas del conocimiento, pues integra ciencias que van desde la Biología, Psicología, Física, y las Matemáticas.

La ciencia de la Vida Artificial como tal se encuentra dentro de las *Ciencias de la Complejidad*, que estudia los fenómenos subyacentes y comunes a todos los sitemas complejos, como ecosistemas, economías y culturas, modelándolos mediante la interacción de *elementos simples*.

En ambos temas, el principal centro de investigación es el Instituto de Santa Fe (`http://www.santafe.edu`) para el Estudio de los Sistemas Complejos. Hoy en día, la Sociedad Internacional para la Vida Artificial, fundada principalmente por los investigadores del Instituto de Santa Fe, es la institución más importante en el tema. Su página Web puede visitarse en `http://www.alife.org`.

Capítulo 2

El Tutorial

2.1. Motivación

El Grupo de Vida Artificial de la Universidad de Chile (GVA) es un grupo organizado de estudiantes de ingeniería que centra su interés en todo el área de la Vida Artificial. Actualmente se compone por los miembros de la página Web del grupo ubicada en `http://gva.li2.uchile.cl`.

El objetivo principal del grupo es dar a conocer las diversas técnicas que integran esta ciencia, y realizar experimentos para motivar el desarrollo de tecnologías de vanguardia en forma dinámica y entretenida.

Las técnicas básicas forman un conjunto de métodos alternativos de la computación y del área eléctrica, que le pueden servir a cualquier ingeniero civil como herramienta de apoyo en la resolución y optimización de problemas.

¡Estás invitado para formar parte del grupo y participar activamente!

2.2. Objetivo General

Al término de este tutorial, conocerás a nivel básico las principales características de una variedad de herramientas de análisis, reconocimiento de patrones y métodos de optimización no lineal, siendo capaz de utilizarlas en problemas reales de tu ámbito de conocimiento.

2.3. Objetivos Específicos

1. Aplicar Redes Neuronales en la clasificación y reconocimiento de patrones.

2. Aplicar métodos de la Computación Evolucionaria (Algoritmos Genéticos, Programación Genética) para la optimización.

3. Analizar resultados y conocer los límites de las herramientas utilizadas.

2.4. Conocimientos Previos

Es necesario que poseas conocimientos de programación en algún lenguaje de programación como C, Java u otro, aunque preferentemente trabajaremos con C. Esto es necesario para poder implementar las los algoritmos en los laboratorios.

Además, debes conocer Cálculo Matricial y el Cálculo en Varias Variables a nivel básico.

2.5. Metodología

El grupo se reune dos veces semanales, en los módulos 2.5 y 4.5, es decir, todos los martes y jueves desde las 16:15 hasta las 17:45 horas, en donde se realizan alternadamente tutoriales teóricos y sesiones de Laboratorio. Cada semana se ve un tema diferente. En las clases teóricas se da a conocer y se analiza una herramienta, la cual luego se pone a prueba en el laboratorio.

Al final del tutorial, deberás presentar un miniproyecto grupal que utilice la materia vista.

2.6. Contenidos

Estudiar Vida Artificial no es una tarea simple. El número de temas que se pueden tratar son muchísimos. En algunos libros, se centra más bien en la teoría de los Sistemas Complejos (Caos). Sin embargo, nuestro enfoque es más bien lúdico; más orientado a las *técnicas* necesarias para crear Vida Artificial que en la teoría misma.

En otras palabras, el contenido corresponde a la parte llamada *dura*, es decir, aquellas que permiten realizar simulaciones en tiempo real basadas en arquitecturas complejas.

El contenido *tentativo*, por semana, es:

1. Tutorial: Introducción al área. La Neurona y su modelo matemático, el Perceptrón. Laboratorio: Entrenamiento del perceptrón para la detección de imágenes.

2. Tutorial: Redes Neuronales Multicapa (Feedforward Multilayer Neural Network). El Algoritmo de Retropropagación (BP). Laboratorio: Clasificación de Cáncer de Mamas.

3. Tutorial: Combinadores Adaptivos (Adalines). Cancelación de Ruído en Señales. Laboratorio: Eliminación de ruído en un canal de comunicación.

4. Tutorial: Aprendizaje no supervisado. Mapas de Kohonen. Laboratorio: Clasificación no-supervisada de países del mundo y comparación con estudios socio-económicos.

5. Tutorial: Redes Neuronales Recurrentes. Predicciones de tiempo utilizando Time-Delay-Neural-Networks (TDNN). Laboratorio: Predicción de la órbita del Sol.

6. Tutorial: Redes Neuronales Recurrentes y Memoria Asociativa. Redes de Hopfield. Laboratorio: Implementación de una memoria asociativa y experimentación.

7. Tutorial: Teorema de Kolgomorov de aproximación universal. Límites en la complejidad. Entropía de Shannon. La maldición de la dimensionalidad. El dilema Sesgo-Varianza. El poder de la Aleatoriedad. Laboratorio: Tema libre.

8. Tutorial: Sistemas complejos no integrables. Autómatas Celulares. Laboratorio: Experimentación con Autómatas Celulares, para juegos y simuladores de sistemas físicos.

9. Tutorial: Introducción a la Computación Evolucionaria. El Algoritmo Genético Simple.Introducción a la Programación Genética. Laboratorio: Video sobre la Programación Genética de John Koza.

10. Tutorial: Principales deficiencias del Algoritmo Genético Simple. Optimización Multimodal. Crowding Determinístico. Laboratorio: Imitación de imágenes mediante un algoritmo genético.

11. Tutorial: Introducción a la Robótica. Laboratorio: Interacción con robots simples. Experiencia con brazo robótico.

12. Presentación de los proyectos.

2.7. Actividades de Evaluación

El tutorial no tiene evaluación y sólo pretende enseñar por via de la entretención. La idea es que los participantes se motiven con el tema y desarrollen un miniproyecto que utilice la materia vista en los tutoriales, y que participen activamente en los eventos del grupo. Lo que sí va a existir son mini-ejercicios que van sin nota, para soltar la mano.

2.8. Los Apuntes

Para poder aprender bien la materia, se recomienda fuertemente imprimir los apuntes de clase en clase como guía y material de apoyo, ya que estos contendrán los ejemplos que desarrollaremos en clases. Los apuntes se irán publicando en el sitio `http://gva.li2.uchile.cl`. Léelos por lo menos una vez antes de asistir a clases. Además se recomienda como trabajo personal que resuelvas todos los ejercicios que se plantean, pues es la única manera de "soltar la muñeca". No necesitarás más que una calculadora común, aunque una que pueda realizar cálculo matricial te facilitaría mucho la vida.

También se publicarán los laboratorios en la Web, más las herramientas de desarrollo. Estos son para que puedas experimentar un poco en tu propia casa.

Capítulo 3

Introducción a las Redes Neuronales

En este capítulo introduciremos a un modelo matemático muy simplificado del cerebro. Se trata de un método de representación de funciones que utiliza redes de unidades de cómputo muy sencillas, y de cómo estas redes son capaces, mediante un entrenamiento adecuado, de aprender estas representaciones a partir de ejemplos.

Una de estas redes representa a una función de una forma muy parecida a como un circuito eléctrico representa a una función Booleana sólo mediante compuertas lógicas sencillas. Estás redes son especialmente útiles para representar funciones muy complicadas que toman y entregan valores contínuos.

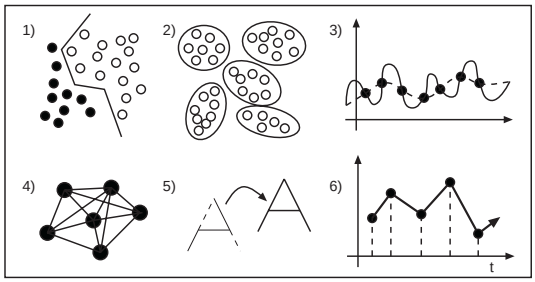

Figura 3.1: Tareas que una Red Neuronal puede realizar.

3.1. ¿Para qué sirve una Red Neuronal?

Las Redes Neuronales se pueden utilizar en muchas aplicaciones diferentes. En particular, podemos mostrar las siguientes, que son de mucho interés.

1. Clasificación de Patrones: La tarea de clasificación de patrones consiste en asignarle a una entrada a la red, que puede ser por ejemplo una imagen, un sonido u otro, una clase. Aplicaciones comúnes de este tipo son reconocimiento de caractéres, de voz, de circuitos eléctricos y de cardiogramas.

2. Clustering / Categorización: También conocido como clasificación de patrones no supervisada, se disponen de datos de entrenamiento sin previa clasificación. La tarea de la red consiste en hallar diferencias entre los datos y categorizarlos. Aplicaciones son Data Mining, compresión de datos, y análisis de datos exploratorios.

3. Aproximación de Funciones: En una aproximación se dispone de un conjunto de pares de vectores $(\vec{x}_i, \vec{y}_i)$ que corresponden a una función *desconocida* $\mu(\cdot)$, tal que para cada

entrada $\vec{x}_i$ se tiene que $\mu(\vec{x}_i) = \vec{y}_i$. Entonces, la tarea consiste en aproximar está función mediante una función $\hat{\mu}(\cdot)$.

4. Optimización: Una gran variedad de problemas en las matemáticas, estadísticas, ingeniería, ciencia, medicina y economía pueden ser planteados como problemas de optimización. La meta de un algoritmo de optimización consiste en hallar una solución que satisfaga un conjunto de restricciones de tal manera que la función objetivo sea maximizada o minimizada. Como ejemplo, el problema del Vendedor Viajero, es un problema clásico muy difícil.

5. Memoria Asociativa: Normalmente en los computadores de hoy en día, una entrada en la memoria sólamente se puede obtener si se conoce su ubicación en la memoria, y dicha dirección es totalmente independiente del contenido. Más aún, si se comete un pequeño error en el cálculo de la dirección, se obtiene un resultado totalmente distinto al deseado. La Memoria Asociativa es un tipo de memoria, que como dice su propio nombre, puede ser consultada directamente por su contenido. Incluso si la consulta contiene distorsiones o pequeño ruído, la Memoria Asociativa es capaz de hallar la entrada deseada y restaurarla.

6. Predicción: Dado un conjunto de n muestras $y(t_1), y(t_2), \ldots ..., y(t_n)$ que corresponden a una secuencia de tiempo $t_1, t_2, \ldots, t_n$ la tarea consiste en predecir la muestra $y(t_{n+m})$ en un futuro t_{n+m}. La predicción tiene una importante aplicación en la toma de decisiones en los negocios, ciencia e ingeniería. La predicción del stock de mercado y el clima/tiempo son aplicaciones típicas de estas técnicas.

En este tutorial, veremos variantes para todas estas aplicaciones, menos la de optimización, ya que para eso aplicaremos los Algoritmos Genéticos.

3.2. ¿Cómo funciona el Cerebro?

La forma exacta de cómo funciona el cerebro y provoca la conciencia, es uno de los misterios más grandes de la ciencia. Desde siempre la humanidad ha sabido que golpes en la cabeza pueden llevar a pérdida de la memoria, inconciencia e incluso alteraciones en el comportamiento. Sin embargo, no ha sido hasta el siglo XVII que se ha propuesto el cerebro como el lugar de alojamiento del pensamiento.

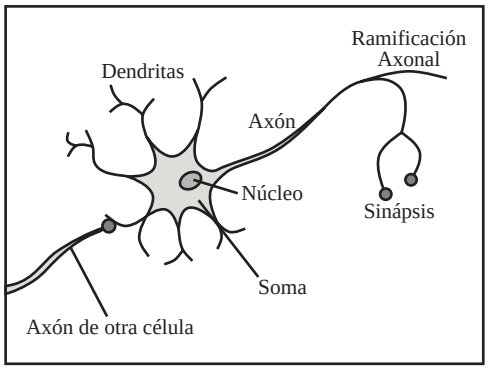

Figura 3.2: La neurona.

Lo que sabemos es que la *neurona*, es la unidad fundamental del tejido nervioso, incluyendo el cerebro. Cada neurona consiste en un cuerpo celular, o *soma*, que contiene al núcleo célular. Las ramificaciones que salen de ella consisten en

fibras llamadas *dendritas* y una fibra larga llamada *axón*. Las dendritas se ordenan en forma de arbusto en torno al soma mientras que el axón tiene extensiones de un centímetro comúnmente, y en casos extremos llega a tamaños de hasta un metro, hasta conectarse finalmente con otra neurona por sus dendritas a través de una junturas denominadas *sinápsis* (Figura 3.2).

Las señales se transmiten mediante un complicado proceso electroquímico. Las sinápsis emiten unas sustancias (neurotransmisores) que ingresan a las dendritas de la segunda célula, provocando así una disminución en su potencial eléctrico. Si este potencial disminuye hasta un cierto *umbral de activación* se dispara la neurona. El impulso se transmite por el axón hacia las demás neuronas. Existen dos tipos de sinápsis: las *inhibitorias* y las *excitatorias*.

Sin embargo, quizás el descubrimiento más importante es el hecho que las neuronas muestran *plasticidad*, es decir, varían la intensidad de sus interconecciones según cómo han sido exitadas en el largo plazo. Se piensa que es este el mecanismo que forma la base para el aprendizaje.

El verdadero potencial del cerebro sin embargo no radica en la forma de una neurona, sino en la forma en que ellas están interconectadas entre sí. La arquitectura y el paralelismo en que operan estas unidades la llevan a realizar tareas sumamente complejas, cómo reconocer patrones, almacenar memoria, asociar ideas. Y aunque estas arquitecturas claves nos son y serán quizás por cuánto tiempo desconocidas, hay una cosa que es cierta: *cerebros provocan mentes.*

3.3. El perceptrón

El primer modelo que estudiaremos es el perceptrón, que consiste en una sola unidad básica. La figura (3.3) muestra una unidad perceptrón. Esta recibe una entrada a través de sus conexiones de entrada, y a partir de ella calcula la salida.

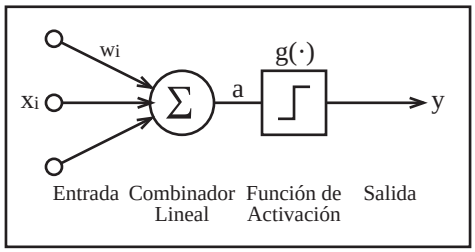

Figura 3.3: Una unidad.

Esta unidad se constituye de las siguientes componentes:

1. Unidades de Entrada: Ilustrados mediante círculos, estos son los elementos que estimulan a la red, y pueden estar encendidos (=1) ó apagados (=0), permitiéndonos representar un patrón *booleano*. Se denotan con x_i, donde $i = 1, \ldots, N$.

2. Conexiones: Las conexiones son aquellas a través de las cuales se propaga la señal que conforma al patrón. Están etiquetadas con $j = 1, \ldots, m$. Cada conexión tiene asociado un peso w_j que indica que tan fuerte es.

3. Combinador Lineal: Esta componente capta las señales que provienen de la entrada y las *combina* en una sóla.

4. Función de Activación: Este módulo recibe una única señal proveniente del combinador lineal y detecta si esta es positiva o negativa. Dependiendo de esto emite una de dos senãles diferentes.

5. Umbral: El umbral w_0 es un valor que actúa en el combinador lineal aumentando ó disminuyendo su respuesta. Este puede interpretarse como el valor que determina el límite entre la activación y la intactividad de la neurona. En la práctica, este se comporta como un peso adicional.

Lo que hace esta neurona artificial es emitir una respuesta dependiendo de como ha sido estimulada. Por ejemplo, si le presentamos el patrón (Verdadero, Verdadero, Falso) que se representa mediante el vector (1.0, 1.0, 0.0), el Perceptrón calcula una respuesta a partir de las intensidades de las interconexiones, emitiendo finalmente un 1 ó 0.

Matemáticamente, un perceptrón funciona de la siguiente manera:

Evaluación Perceptrón

1. Partimos con un patrón $\vec{x} = (x_1, \ldots, x_N)$.

2. Multiplicamos cada peso con la componente del patrón correspondiente y sumamos los resultados junto con el umbral.

$$\begin{aligned} a &= \sum_{i=1}^{N} w_i x_i + w_0 \\ &= w_1 x_1 + w_2 x_2 + \ldots + w_N x_N + w_0 \end{aligned} \tag{3.1}$$

3. Aplicamos la función de activación $g(\cdot)$ al resultado anterior. La función $g(\cdot)$ es

$$g(x) = \begin{cases} 1 & \text{si } x \geq 0 \\ 0 & \text{si } x < 0 \end{cases} \tag{3.2}$$

Luego, aplicado al resultado anterior, tenemos

$$y = g(a) = g(\sum_{i=1}^{N} w_i x_i + w_0), \tag{3.3}$$

obteniéndo así la respuesta del perceptrón. Recordemos que w_0 corresponde al *umbral*.

Observemos que en la fórmula que calcula el combinador lineal, se puede hacer una análogía entre el producto entre las entradas y los pesos $w_i x_i$ con el umbral w_0, pues si consideramos una entrada adicional artificial x_0 que vale 1 por definición, la combinación lineal puede escribirse como:

$$\sum_{i=1}^{N} w_i x_i + w_0 = \sum_{i=0}^{N} w_i x_i$$

Adoptaremos esta notación simplificada en lo que sigue.

Desarrollemos el siguiente ejemplo.

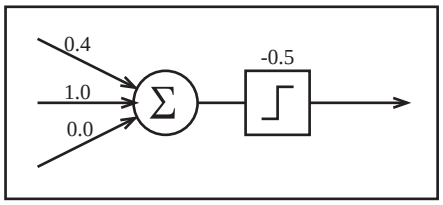

Figura 3.4: Ejemplo

Ejemplo 1 *Para el Perceptrón de la figura (3.4), se tienen las entradas* $(0,0), (0,1)$ *y* $(1,1/2)$. *Los resultados son:*

1. $g(0{,}5 \cdot 0 + 0{,}5 \cdot 0 - 0{,}5) = g(-0{,}5) = 0$
2. $g(0{,}5 \cdot 0 + 0{,}5 \cdot 1 - 0{,}5) = g(0{,}0) = 1$
3. $g(0{,}5 \cdot 1 + 0{,}5 \cdot 1 - 0{,}5) = g(0{,}5) = 1$

Ejercicio 1 *Para el perceptrón de la figura (3.5), calcular la respuesta del perceptrón ante las siguientes entradas:*

$(0,0,0)$	$(0,1,1)$	$(1,1,0)$
$(1,0,0)$	$(0,0,1)$	$(1,1,1)$
$(0,1,0)$	$(1,0,1)$	$(1/2,1,1/2)$

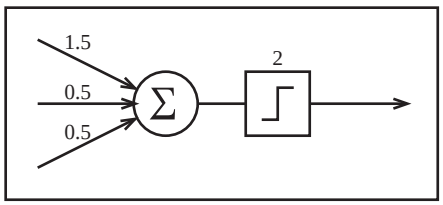

Figura 3.5: Ejercicio

Dada que la respuesta del Perceptrón es un valor binario, es decir, 0 ó 1, el Perceptrón está especialmente adecuado para representar funciones booleanas. En particular, veamos las 3 compuertas lógicas (figura 3.6).

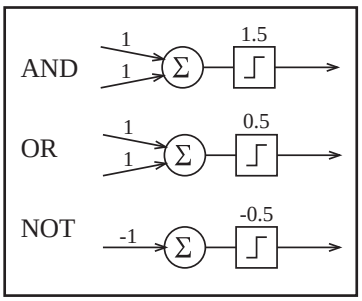

Figura 3.6: Las 3 compuertas lógicas más usadas.

Sin embargo, la utilidad más importante radica en la capacidad del Perceptrón de *aprender* una función booleana. Supongamos que tenemos un conjunto de muestras que provienen de una función $f(\cdot)$ desconocida, para las cuales tenemos los resultados. Es decir, tenemos parejas $(\vec{x}_p, d_p)$, donde p es el índice que corre sobre todos los patrones, y queremos que la red sea capaz de aprender una función $\hat{f}(\cdot)$ tal que para todo p, $\hat{f}(\vec{x}_p) = d_p$. Veremos que si está función $f(\cdot)$ cumple ciertas condiciones, el Perceptrón efectivamente logra interpolarla.

3.4. El Algoritmo de Aprendizaje del Perceptrón

El siguiente algoritmo se basa en una idea muy sencilla. Se le presentan sucesivamente todos los ejemplos al Perceptrón, hasta que entregue un resultado erróneo. Cuando sucede esto, se revisa como influyó cada uno de los pesos (y el umbral) en el resultado. Si un peso provocó que la respuesta sea demasiado alta, se dismuye y viceversa. Esto se repite hasta que el Perceptrón entregue el resultado deseado para *todas* las entradas.

La regla de actualización de los pesos es bastante simple. primero, podemos calcular el error en el resultado:

$$\varepsilon = d - y \tag{3.4}$$

es decir, la diferencia entre la respuesta *deseada* d y la respuesta del Perceptrón, y. A partir del error ε se actualiza cada peso mediante la regla:

$$w_i \leftarrow w_i + \mu x_i \varepsilon \tag{3.5}$$

Es decir, al peso w_i se le suma un valor proporcional al error provocado por la componente x_i de la entrada. Esta regla también se le aplica al umbral considerando a su entrada de valor 1 constante. Es fácil ver que esto provoca una disminución del error. El parámetro μ es el *factor de aprendizaje*, que regula qué tan bruscas

deben ser estas modificaciones. Un valor grande de μ modifica los pesos más rápidamente, pero puede llevar, si es demasiado grande, a cambios exagerados que no logran converger. Por otro lado, si μ es demasiado pequeño este problema no ocurre pero la convergencia es mucho más lenta.

Como sabemos que sólo mediante esta sencilla regla el Perceptrón converge a una aproximación adecuada de la función, no importa cómo inicializamos los pesos. En particular, se aconseja, por razones que veremos más tarde, inicializar los pesos aleatoriamente.

Ahora podemos formalizar el algoritmo de aprendizaje:

Aprendizaje-Perceptrón
Entrada: Ejemplos $\{\vec{x}_p\}$, Factor de Aprendizaje μ

1. Inicializar los pesos w_i aleatoriamente.
2. Repetir
   - a) Para cada ejemplo $\vec{x}$
     - 1) $y \leftarrow g(\sum_{i=0}^{N} w_i x_i)$
     - 2) $\varepsilon \leftarrow d - y$
     - 3) Para cada w_i
       - a' $w_i \leftarrow w_i + \mu x_i \varepsilon$

   hasta que todos los ejemplos sean predichos correctamente ó hasta que se haya alcanzado un criterio de parada.

Este sencillo algoritmo genera el aprendizaje del Perceptrón.

Ejemplo 2 *Disponemos de la siguiente tabla que muestra los datos y el resultado esperado. Queremos entrenar al Perceptrón partiendo por la situación de la figura (3.7) con un factor de aprendizaje de* $\mu = 0{,}3$.

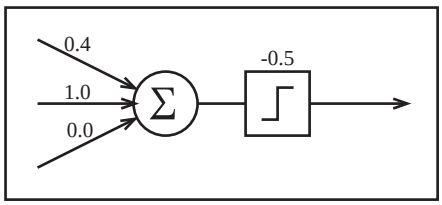

Figura 3.7: Ejemplo

Patrón	x_1	x_2	x_3	d
1	0	0	0	0
2	0	1	0	1
3	1	1	1	0

La siguiente tabla muestra la evolución de los pesos a medida que avanza el algoritmo.

Patrón	w_0	w_1	w_2	w_3	$\sum w_i x_i$	y	ε
1	$-0{,}5$	0,4	1,0	0,0	$-0{,}5$	0	0
2	$-0{,}5$	0,4	1,0	0,0	0,5	1	0
3	$-0{,}5$	0,4	1,0	0,0	0,9	1	-1

Hay error, luego actualizamos los pesos:

$$w_0 \leftarrow -0{,}5 + 0{,}3 \cdot 1 \cdot (-1) = -0{,}8$$

$$w_1 \leftarrow 0{,}4 + 0{,}3 \cdot 1 \cdot (-1) = 0{,}1$$

$$w_2 \leftarrow 1{,}0 + 0{,}3 \cdot 1 \cdot (-1) = 0{,}7$$

$$w_3 \leftarrow 0{,}0 + 0{,}3 \cdot 1 \cdot (-1) = -0{,}3$$

Probamos nuevamente,

Patrón	w_0	w_1	w_2	w_3	$\sum w_i x_i$	y	ε
3	$-0{,}8$	0,1	0,7	$-0{,}3$	$-0{,}3$	0	0
1	$-0{,}8$	0,1	0,7	$-0{,}3$	$-0{,}8$	0	0
2	$-0{,}8$	0,1	0,7	$-0{,}3$	$-0{,}1$	0	1

Actualizamos,

$$w_0 \leftarrow -0{,}8 + 0{,}3 \cdot 1 \cdot 1 = -0{,}5$$

$$w_1 \leftarrow 0{,}1 + 0{,}3 \cdot 1 \cdot 0 = 0{,}1$$

$$w_2 \leftarrow 0{,}7 + 0{,}3 \cdot 1 \cdot 1 = 1{,}0$$

$$w_3 \leftarrow -0{,}3 + 0{,}3 \cdot 1 \cdot 0 = -0{,}3$$

Patrón	w_0	w_1	w_2	w_3	$\sum w_i x_i$	y	ε
2	$-0{,}5$	0,1	1,0	$-0{,}3$	0,5	1	0
3	$-0{,}5$	0,1	1,0	$-0{,}3$	0,3	1	-1

$$w_0 \leftarrow -0{,}5 + 0{,}3 \cdot 1 \cdot (-1) = -0{,}8$$

$$w_1 \leftarrow 0{,}1 + 0{,}3 \cdot 1 \cdot (-1) = -0{,}2$$

$$w_2 \leftarrow 1{,}0 + 0{,}3 \cdot 1 \cdot (-1) = 0{,}7$$

$$w_3 \leftarrow -0{,}3 + 0{,}3 \cdot 1 \cdot (-1) = -0{,}6$$

A continuación seguiremos la evolución de los pesos omitiendo los pasos intermedios:

Patrón	w_0	w_1	w_2	w_3	$\sum w_i x_i$	y	ε
3	$-0{,}8$	$-0{,}2$	0,7	$-0{,}6$	$-0{,}9$	0	0
1	$-0{,}8$	$-0{,}2$	0,7	$-0{,}6$	$-0{,}8$	0	0
2	$-0{,}8$	$-0{,}2$	0,7	$-0{,}6$	$-0{,}1$	0	1
2	$-0{,}5$	$-0{,}2$	1,0	$-0{,}6$	0,5	1	0
3	$-0{,}5$	$-0{,}2$	1,0	$-0{,}6$	$-0{,}3$	0	0
1	$-0{,}5$	$-0{,}2$	1,0	$-0{,}6$	$-0{,}5$	0	0

Y así obtuvimos un Perceptrón que aprendió exitósamente la función en 15 iteraciones, con umbral y pesos $w_0 = -0{,}5$, $w_1 = -0{,}2$, $w_2 = 1{,}0$, $w_3 = -0{,}6$.

Ejercicio 2 *Dados los datos*

Patrón	x_1	x_2	d
1	0	0	0
2	0	1	1
3	1	1	0

y un Perceptrón inicial $w_0 = 1$, $w_1 = 1$, $w_2 = 1$, *y un factor de aprendizaje* $\mu = 0{,}3$, *entrenarlo hasta obtener un conjunto de pesos que predigan los resultados correctamente.*

Ejercicio 3 *Dado un Perceptrón inicial* $w_0 = 0{,}5$, $w_1 = 0$, $w_2 = 0$, *y un factor de aprendizaje de* $\mu = 0{,}5$ *entrenarlo hasta obtener un conjunto de pesos que aproximen a las funciones:*

1. *OR*
2. *AND*
3. *NAND:* $nand(x, y) = \neg(x \wedge y)$

3.5. La mala Noticia

Sin embargo, hay un problema muy severo que limita la aplicabilidad de los Perceptrones: su capacidad de representación. La pregunta que nos hacemos es: ¿cuáles son las funciones booleanas que un Perceptrón es capaz de representar? El problema es que un peso en particular w_j sólo puede influir en el resultado de la combinación lineal en un sentido. Es decir, sin importar qué entrada se le presenta al Perceptrón, el peso w_j siempre va a causar un aumento en el resultado si w_j es positivo, o una disminución en caso contrario.

Un poco de geometría nos ayudará a entender mejor este problema. Supongamos que tenemos tres funciones booleanas, AND, OR y XOR (OR exclusivo). Si los dibujamos en un plano, marcando aquellos que valen cero con blanco y los otros con negro, resultan los dibujos de la figura (3.8).

Las dos primeras funciones, es decir, AND y OR, pueden ser representadas por un Perceptrón. Sin embargo, XOR no. La diferencia radica en que AND y OR son funciones *linealmente separables*. Es decir, se puede trazar una línea que separa los puntos que corresponden a unos de los ceros. Si la función tiene tres parámetros, el dibujo correspondería a un cubo como el de la figura (3.9), y en este caso tendríamos que hallar

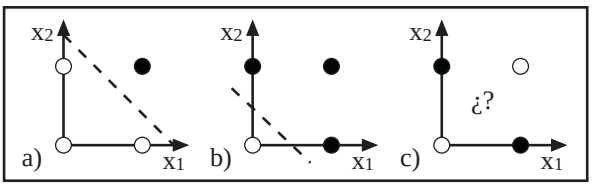

Figura 3.8: Tres funciones booleanas y los planos separadores.

un plano separador en vez de una línea. En más dimensiones, buscamos *hiperplanos separadores*.

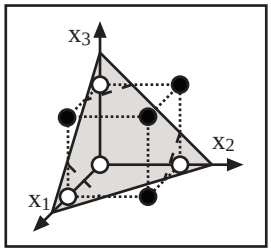

Figura 3.9: El plano separador.

Esto no ocurre para las funciones como XOR, que no tienen plano separador, i.e. no son linealmente separables. Desgraciadamente, la gran mayoría de las funciones booleanas no son separables.

Capítulo 4

Redes Neuronales Multicapa

A fines de los años 1950, Rosenblatt y otros han descrito por primera vez, en su libro *Perceptrons*, a las Redes Neuronales Multicapa. Si bien en su libro se centraron más bien en los perceptrones, mencionaron que "las Redes Neuronales Multicapa son un problema importante de investigación". Sin embargo, no entendían como había que entrenar dichas redes, ya que si bien era posible calcular los errores en su salida, no quedaba claro cómo había que adaptar los pesos para producir el resultado deseado.

Sin embargo, ya en 1969 el algoritmo más importante de entrenamiento hoy en día, Retropropagación del Error (*Error-Backpropagation* ó simplemente BP) fue descubierto por Bryson y Ho, y ha pasado desapercibido por razones computacionales y sociológicas hasta 1985.

4.1. El Problema de las Fronteras

Ya hemos visto que las únicas funciones representables por Perceptrones son las linealmente separables. Si tenemos clases cuya separación no puede trazarse mediante un hiperplano, ¿cómo resolvemos el problema?

La figura (4.1) muestra un problema no linealmente separable. Este posee una frontera curva que separa las dos clases presentes. Una propuesta sería tratar de aproximar esta frontera mediante polígonos (en más dimensiones se denominan poliedros), como indica la figura.

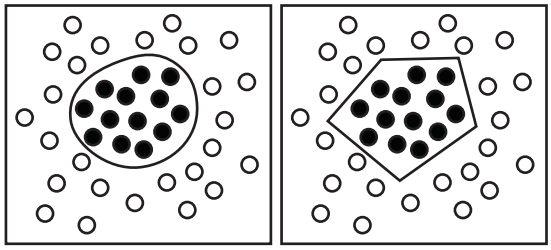

Figura 4.1: Una aproximación de la frontera.

El problema se puede seguir complicando más aun, pues basta pensar en regiones no convexas (a), regiones con sacados (b) e incluso en grupos de regiones no conexas (c) de la figura (4.2).

Para resolver este problema hay que recordar que un Perceptrón, en el fondo, construye internamente un hiperplano separador. ¿Podemos combinar Perceptrones para representar regiones más complicadas? Afortunadamente, la respuesta es sí. Y es en este instante en donde entran las Redes Neuronales Multicapa en el escenario.

Probemos qué sucede si combinamos los Per-

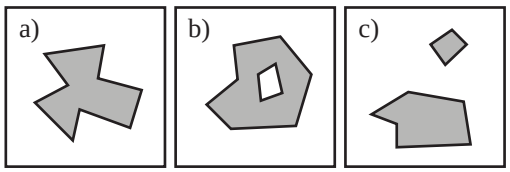

Figura 4.2: Regiones Problemáticas.

ceptrones. Supongamos que tenemos varios que separan el conjunto en dos regiones, como en la figura (4.3). Tomemos las *respuestas* de estos Perceptrones para alimentar a una unidad tipo AND, que ya sabemos construir. De esta manera, podemos formar intersecciones de regiones (convexas). Tomando los resultados de estas intersecciones y conectándolos a una unidad OR, formamos las uniones de estas regiones.

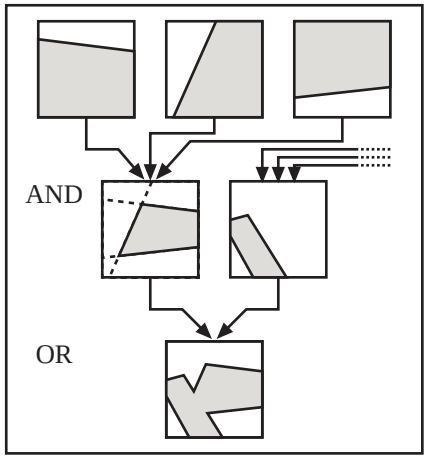

Figura 4.3: Combinando Perceptrones.

Tomando en cuenta este último hecho, y mirando la figura (4.4), resulta evidente que mediante este método, es decir, tres capas de Perceptrones con un número lo suficientemente grande de unidades, se puede aproximar cualquier región ó frontera de decisión (a) a una precisión arbitraria (b y c).

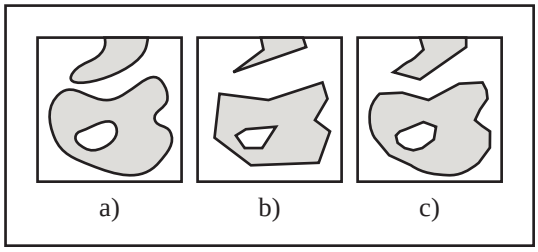

Figura 4.4: Aproximación arbitraria de la frontera de decisión (a).

Además, observando la arquitectura de la red, vemos que las conexiones entre las unidades no son arbitrarias: siempre van de una capa a la siguiente. Unidades de una capa dada no están conectadas ni entre ellas mismas, ni con unidades de capas anteriores. Si bien la construcción anterior es plausible, se ha demostrado que incluso bastan dos capas de pesos para aproximar cualquier función contínua, lo cual es un tremendo resultado. Es debido a las razones anteriores que a este tipo de red se le dió el nombre de *Two Layer Feedforward Neural Network*, es decir, Redes Neuronales de dos Capas de Alimentación Progresiva. En lo que sigue las llamaremos Redes Neuronales Multicapa por simplicidad.

4.2. Arquitecturas de Redes Neuronales

Ya hemos visto las Redes Neuronales Multicapa, y su capacidad de aproximar cualquier frontera de decisión. ¿Cuáles son las diferentes arquitecturas, y para qué sirven?

Las Redes Neuronales son herramientas flexibles que sirven para una variedad muy am-

plia de problemas. Aun cuando estos últimos poseen características muy distintas, pueden ser resueltas en parte por una arquitectura neuronal genérica, como las Redes Neuronales Multicapa, pero a veces el resultado no es óptimo e incluso insatisfactorio.

Bajo ciertas condiciones, las redes neuronales son capaces de realizar computación universal, pudiendo así implementar cualquier algoritmo. Una forma de hacerlo sería construyendo un red compleja a partir de las compuertas lógicas, como un circuito eléctrico. Sin embargo, para que estas mantengan su capacidad de aprendizaje, el problema de diseñar una arquitectura adecuada se convierte en un verdadero arte. Hasta el día de hoy, no se sabe mucho de esto.

Es así como se han investigado algunas arquitecturas, pero estas en general son muy sencillas y fueron diseñadas para resolver problemas muy específicos.

La gran diferencia que se hace entre los tipos de redes neuronales está en la clasificación entre redes recurrentes y feedforward. Las redes recurrentes pueden tener cualquier forma, incluso pueden contener ciclos. Esto les permite tener un *estado interno*, el cual les da el poder de emitir respuestas según lo observado, a diferencia de las feedforward cuya respuesta sólo depende de la entrada y nada más. Sin embargo, el problema de las redes recurrentes es que no se conoce bien su comportamiento debido a la complejidad de las ecuaciones de recurrencia no-lineales que describen estos sistemas. En algunos casos experimentales, estos no convergen u oscilan. Por otro lado, las tipo feedforward se conocen bien hasta un cierto grado, sobre todo las de dos capas intermedias como las que veremos a continuación.

4.3. Redes Neuronales Multicapa

La red neuronal que nosotros llamaremos como Red Neuronal Multicapa es la más usada, pues es sumamente simple y a la vez poderosa. El uso principal que se le ha dado es el de clasificar patrones y aproximar funciones contínuas, aunque con un par de trucos esta puede desempeñar una gran variedad de tareas. Concentrémonos antes en su forma.

La Red Neuronal Multicapa se compone de tres capas de unidades, con dos capas de pesos que las conectan entre sí. En la literatura generalmente se denota como Two Layer Feedforward Neural Network, pero también hablan de Three Layer Neural Network, dependiendo de si se cuentan las capas de pesos ó unidades respectivamente. Estas tres capas de unidades reciben nombres diferentes. La primera es la *capa de entrada*, ya que es aquí donde ingresan los parámetros a la red. La última es la *capa de salida*, por razones obvias. La segunda se denota capa intermedia ó más frecuentemente *capa oculta*. La estructura general la podemos ver en la figura (4.5).

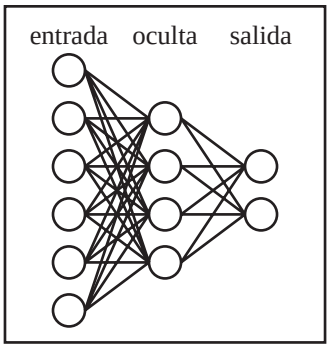

Figura 4.5: Red Neuronal Multicapa.

Generalmente, todas las unidades entre una capa y otra están totalmente interconectadas. Luego, teniendo n unidades en una y m en la próxima capa hay nm conexiones y por ende nm pesos si absorvemos los umbrales. Estos los representaremos mediante matrices de tamaño $m \times n$, donde el número de columnas corresponde al número de unidades de la capa previa más la unidad asociada al umbral y el número de filas a la capa posterior. Como son dos capas de pesos, hacen falta dos de estas matrices.

El número de unidades en cada capa suele determinarse de la siguiente manera. Para la entrada, generalmente se le asocia una unidad a cada variable del problema, igual que en un Perceptrón. Para la capa de salida, tenemos dos alternativas. Si se trata de un problema de clasificación de patrones en, digamos, c clases distintas, entonces le asociamos a cada una de las clases una unidad, es decir, un total de c unidades de salida. Para aproximar una función, tenemos que mantener en mente que el resultado de una neurona de salida será un valor *en el intervalo* $[0; 1]$. La capa intermedia es más dificil de determinar, pues en general esta depende de la complejidad del problema mismo. Un problema complicado necesitará más unidades ocultas que otro que tiene fronteras de separación sencillas. Como regla, puede probarse con un número igual al promedio entre la entrada y la salida.

En principio, podríamos utilizar el mismo tipo de unidades que las que hemos visto para el Perceptrón. Sin embargo, debido a un problema que veremos después al revisar el algoritmo de aprendizaje, necesitamos reemplazar la función de activación por otra *contínua* que la aproxime (Figura 4.6).

En general, la familia de funciones que tienen una forma parecida a la función escalón que utiliza el Perceptrón, tienen forma de S, y debido a esto se denominan *sigmoides*. Las tres sigmoides más utilizadas son:

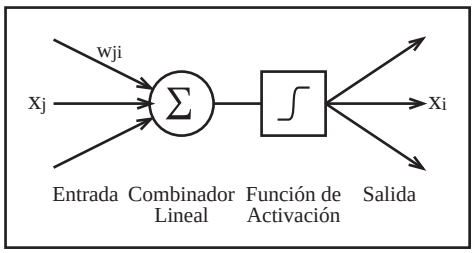

Figura 4.6: Una unidad.

1. *Sigmoide Logística*: La más utilizada, y la que usaremos nosotros. Toma valores entre cero y uno (Figura 4.7).

$$\text{sigm}(x) = \frac{1}{1 + e^{-x}}$$

   Una de las gracias de la sigmoide logística es que en un problema de clasificación de *dos clases* con dos unidades de salida, sus resultados pueden ser interpretados como *probabilidades*. Es decir, para un resultado de la red de $(3/4, 1/4)$, la probabilidad de que el patrón pertenezca a la clase 1 ó 2 es $3/4$ ó $1/4$ respectivamente.

2. *Tangente Hiperbólica*: Toma valores en el intervalo $[-1; 1]$ (Figura 4.8).

$$\text{sigm}(x) = \frac{e^x - e^x}{e^x + e^x}$$

3. *Softmax*: Si queremos generalizar a más de dos clases la idea de que las salidas de la red neuronal se puedan interpretar como probabilidades, usamos, para la unidad i-ésima de una capa, la función

$$\text{sigm}_i(x_1, x_2, \ldots) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

donde la sumatoria corre sobre todas las unidades de la misma capa. En el fondo, estamos normalizando los resultados de la función de activación provenientes de una misma capa, para que la suma de las salidas dé 1.

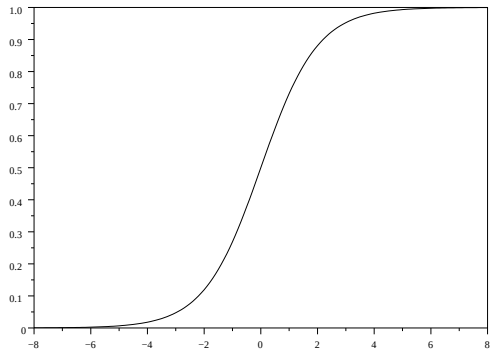

Figura 4.7: Sigmoide Logística.

La elección de estas funciones que parecen tan extrañas a primera vista es muy sencilla. Se les puede calcular su *derivada*, elemento vital para el algoritmo de aprendizaje, y estas tienen expresiónes muy sencillas.

Veamos cómo se calcula el resultado de una Red Neuronal Multicapa. Para esto, usemos la notación empleada en la figura (4.9) para denotar las diferentes componentes. Los x_i son las entradas, y_j las ocultas y z_k las salidas. Los pesos w_{ji} y w_kj conectan a las unidades $i - j$ y $j - k$ respectivamente. Los subíndices están al revés porque esto equivale a la notación empleada para las matrices. Recordemos que el combi-

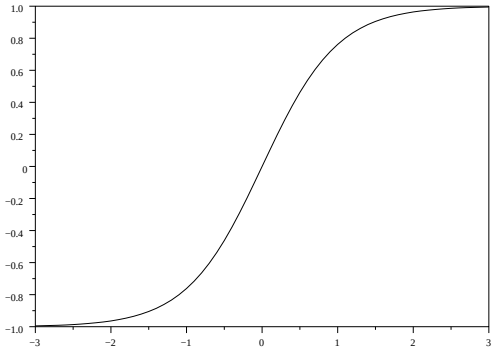

Figura 4.8: Tangente Hiperbólica.

nador lineal del Perceptrón computa

$$\sum_{i=1}^{N} w_i x_i + w_0.$$

El combinador lineal de una Red Neuronal Multicapa hace lo mismo. La combinación lineal de la unidad oculta $j = 1$ sería

$$\sum_{i=1}^{N} w_{1i} x_i + w_{10}$$

donde w_{10} es el umbral de la primera unidad oculta. Para la unidad $j = 2$ tenemos

$$\sum_{i=1}^{N} w_{2i} x_i + w_{20}$$

Ídem para las demás unidades.

Podemos resumir todas estas operaciones escribiéndolo matricialmente: La matriz de pesos multiplica al vector de entrada y a este se le suma el vector de umbrales.

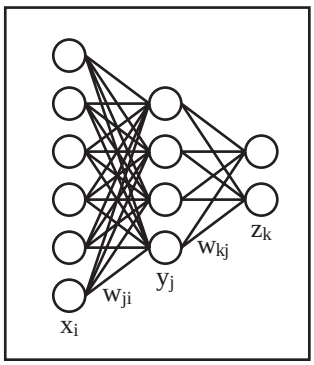

Figura 4.9: Notación empleada.

$$\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1N} \\ w_{21} & w_{22} & \dots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & \dots & w_{MN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix} + \begin{pmatrix} w_1 0 \\ w_2 0 \\ \vdots \\ w_N 0 \end{pmatrix}$$

con lo que la notación para los pesos w_{ji}, w_{kj} en vez w_{ij} y w_{jk} respectivamente cobra sentido. Análogamente, para la segunda capa de pesos, tenemos

$$\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1M} \\ w_{21} & w_{22} & \dots & w_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ w_{L1} & w_{L2} & \dots & w_{LM} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{pmatrix} + \begin{pmatrix} w_1 0 \\ w_2 0 \\ \vdots \\ w_L 0 \end{pmatrix}$$

Ojo: no confundir ambas matrices. El elemento w_{10} de la primera matriz es distinto al w_{10} de la segunda. Ídem los umbrales. Resumamos los elementos que necesitamos para representar una red neuronal y luego veamos cómo se calcula el resultado.

Red Neuronal Multicapa

1. 3 Vectores que representan la entrada y los resultados de cada capa, $\vec{x}$, $\vec{y}$ y $\vec{z}$ de N, M y L componentes respectivamente.
2. 2 matrices de pesos, $(w)_{ji}$ de $M \times N$ y $(w)_{kj}$ de $L \times M$.
3. 2 vectores de umbrales para la primera y la segunda capa, $(w)_{j0}$ y $(w)_{k0}$ de M y L componentes respectivamente.
4. Opcionalmente 2 vectores para los resultados intermedios (las combinaciones lineales) $\vec{a}$ y $\vec{b}$ de M y L componentes respectivamente.

Y el algoritmo para calcular el resultado es

Respuesta Red Neuronal Multicapa

Entrada: Ejemplo $\vec{x}$

1. Para $j = 1, \dots, M$
   - *a)* $a_j \leftarrow \sum_{i=1}^{N} w_{ij} x_i + w_{j0}$
   - *b)* $y_j \leftarrow \text{sigm}(a_j)$
2. Para $k = 1, \dots, L$
   - *a)* $b_k \leftarrow \sum_{j=1}^{M} w_{jk} y_j + w_{k0}$
   - *b)* $z_k \leftarrow \text{sigm}(b_k)$
3. Retornar $\vec{z}$

Ejemplo 3 *Para la Red Neuronal Multicapa de la figura (4.10), calcular los resultados para las entradas (0.0 -1.5 2.0), (3.0 1.0 0.0) y (-0.5 -2.0 3.0). Los umbrales son los números por encima de las unidades.*

Primero armamos las dos matrices de pesos. La primera es

$$\begin{pmatrix} 3{,}0 & 0{,}5 & -0{,}5 \\ -2{,}5 & 1{,}0 & -0{,}5 \end{pmatrix},$$

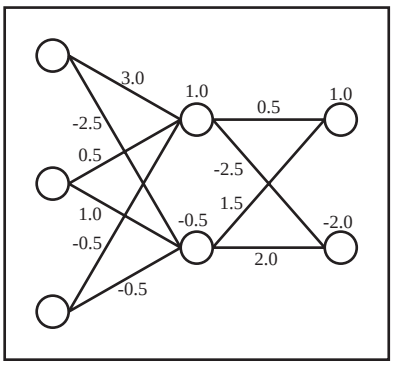

Figura 4.10: Ejemplo.

y la segunda

$$\begin{pmatrix} 0{,}5 & 1{,}5 \\ -2{,}5 & 2{,}0 \end{pmatrix}.$$

Los vectores umbrales son:

$$\begin{pmatrix} 1{,}0 \\ -0{,}5 \end{pmatrix}$$

y

$$\begin{pmatrix} 1{,}0 \\ -2{,}0 \end{pmatrix}.$$

Luego, podemos comenzar el cálculo. Primero para (0.0 -1.5 2.0):

$$\begin{pmatrix} 3{,}0 & 0{,}5 & -0{,}5 \\ -2{,}5 & 1{,}0 & -0{,}5 \end{pmatrix} \begin{pmatrix} 0{,}0 \\ -1{,}5 \\ 2{,}0 \end{pmatrix} = \begin{pmatrix} -1{,}75 \\ -2{,}5 \end{pmatrix}$$

$$\begin{pmatrix} -1{,}75 \\ -2{,}5 \end{pmatrix} + \begin{pmatrix} 1{,}0 \\ -0{,}5 \end{pmatrix} = \begin{pmatrix} -0{,}75 \\ -3{,}0 \end{pmatrix}$$

Aplicamos la sigmoide logística a la combinación lineal,

$$sigm(-0{,}75) = 0{,}321 \qquad sigm(-3{,}0) = 0{,}047$$

Luego la primera capa resulta,

$$\vec{y} = \begin{pmatrix} 0{,}321 \\ 0{,}047 \end{pmatrix}$$

Repitiendo el cálculo para la segunda capa,

$$\begin{pmatrix} 0{,}5 & 1{,}5 \\ -2{,}5 & 2{,}0 \end{pmatrix} \begin{pmatrix} 0{,}321 \\ 0{,}047 \end{pmatrix} = \begin{pmatrix} 0{,}231 \\ -0{,}709 \end{pmatrix}$$

$$\begin{pmatrix} 0{,}231 \\ -0{,}709 \end{pmatrix} + \begin{pmatrix} 1{,}0 \\ -2{,}0 \end{pmatrix} = \begin{pmatrix} 1{,}231 \\ -2{,}709 \end{pmatrix}$$

$$\vec{z} = sigm\left(\begin{pmatrix} 1{,}231 \\ -2{,}709 \end{pmatrix} \right) = \begin{pmatrix} 0{,}774 \\ 0{,}062 \end{pmatrix}$$

que es el resultado de la red para el primer ejemplo. El segundo ejemplo,

$$\begin{pmatrix} 3{,}0 & 0{,}5 & -0{,}5 \\ -2{,}5 & 1{,}0 & -0{,}5 \end{pmatrix} \begin{pmatrix} 3{,}0 \\ 1{,}0 \\ 0{,}0 \end{pmatrix} = \begin{pmatrix} 9{,}5 \\ -6{,}5 \end{pmatrix}$$

$$\begin{pmatrix} 9{,}5 \\ -6{,}5 \end{pmatrix} + \begin{pmatrix} 1{,}0 \\ -0{,}5 \end{pmatrix} = \begin{pmatrix} 10{,}5 \\ -7{,}0 \end{pmatrix}$$

$$\vec{y} = sigm\left(\begin{pmatrix} 10{,}5 \\ -7{,}0 \end{pmatrix} \right) = \begin{pmatrix} 1{,}000 \\ 0{,}001 \end{pmatrix}$$

$$\begin{pmatrix} 0{,}5 & 1{,}5 \\ -2{,}5 & 2{,}0 \end{pmatrix} \begin{pmatrix} 1{,}000 \\ 0{,}001 \end{pmatrix} = \begin{pmatrix} 0{,}502 \\ -2{,}498 \end{pmatrix}$$

$$\begin{pmatrix} 0{,}502 \\ -2{,}498 \end{pmatrix} + \begin{pmatrix} 1{,}0 \\ -2{,}0 \end{pmatrix} = \begin{pmatrix} 1{,}502 \\ -4{,}498 \end{pmatrix}$$

$$\vec{z} = sigm\left(\begin{pmatrix} 1{,}502 \\ -4{,}498 \end{pmatrix} \right) = \begin{pmatrix} 0{,}818 \\ 0{,}011 \end{pmatrix}$$

Y finalmente para el tercer ejemplo: El segundo ejemplo,

$$\begin{pmatrix} 3{,}0 & 0{,}5 & -0{,}5 \\ -2{,}5 & 1{,}0 & -0{,}5 \end{pmatrix} \begin{pmatrix} -0{,}5 \\ -2{,}0 \\ 3{,}0 \end{pmatrix} = \begin{pmatrix} -4{,}0 \\ -2{,}25 \end{pmatrix}$$

$$\begin{pmatrix} -4{,}0 \\ -2{,}25 \end{pmatrix} + \begin{pmatrix} 1{,}0 \\ -0{,}5 \end{pmatrix} = \begin{pmatrix} -3{,}0 \\ -2{,}75 \end{pmatrix}$$

$$\vec{y} = sigm\left(\begin{pmatrix} -3{,}0 \\ -2{,}75 \end{pmatrix}\right) = \begin{pmatrix} 0{,}047 \\ 0{,}060 \end{pmatrix}$$

$$\begin{pmatrix} 0{,}5 & 1{,}5 \\ -2{,}5 & 2{,}0 \end{pmatrix} \begin{pmatrix} 0{,}047 \\ 0{,}060 \end{pmatrix} = \begin{pmatrix} 0{,}114 \\ 0{,}003 \end{pmatrix}$$

$$\begin{pmatrix} 0{,}114 \\ 0{,}003 \end{pmatrix} + \begin{pmatrix} 1{,}0 \\ -2{,}0 \end{pmatrix} = \begin{pmatrix} 1{,}114 \\ -1{,}997 \end{pmatrix}$$

$$\vec{z} = sigm\left(\begin{pmatrix} 1{,}114 \\ -1{,}997 \end{pmatrix}\right) = \begin{pmatrix} 0{,}753 \\ 0{,}120 \end{pmatrix}$$

Luego, los resultados son (0.774 0.062), (0.818 0.011) y (0.753 0.120).

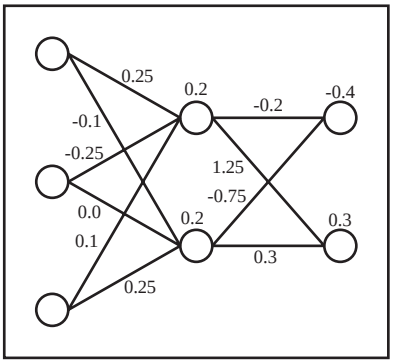

Figura 4.11: Ejercicio.

Ejercicio 4 *Para la Red Neuronal Multicapa de la figura (4.11), plantear las matrices de pesos y los umbrales, y luego calcular las respuestas de las red neuronal para las entradas (1.0 -1.0 -1.0) y (0.2 0.0 -0.75).*

4.4. Algoritmo de Retropropagación del Error

Para entender la problemática que involucra el entrenamiento de una Red Neuronal Multicapa, veamos un sencillo ejemplo intuitivo, dado por la figura (4.12).

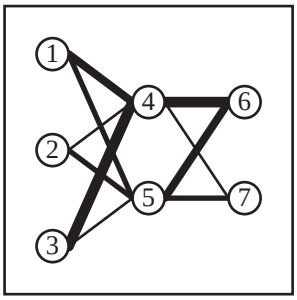

Figura 4.12: Una red simple, donde el grosor de la conexión refleja su intensidad.

Supongamos que nosotros le presentamos un ejemplo a la red, la cual responde con un resultado. Si este es correcto, no hacemos nada. Si hay un error, entonces queremos perturbar levemente los pesos de la red para que este error disminuya. El problema es cómo modificar los pesos para que esto ocurra.

La figura (4.13) muestra esqueméticamente los errores en la salida. Igual que en el algoritmo de aprendizaje del Perceptrón, lo que queremos hacer es averiguar si los pesos de la red influyen positiva- ó negativamente sobre el error, y luego modificarlos en el sentido opuesto en una cantidad pequeña dada por el factor de aprendizaje. Sin embargo, esta idea no se puede aplicar tal como está, pues los pesos de la primera capa influyen en la respuesta a través de los de la segunda.

El algoritmo descubierto por Bryson y Ho es

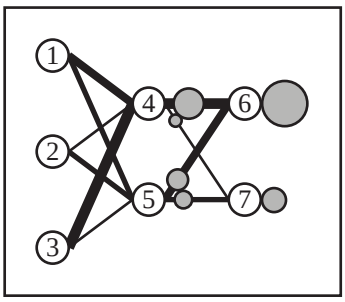

Figura 4.13: Los errores que se producen en la respuesta representados por los círculos se propagan hacia atrás en trozos proporcionales a las intensidades de las conexiones.

muy ingenioso, pues es una forma sensible de dividir el error entre los pesos de la red, para así poder ajustarlos según su grado de participación en el resultado.

Lo que se hace es sencillo. *El truco para lograr nuestro objetivo consiste en determinar los pesos que contribuyen al error y moverlos en el sentido opuesto en una cantidad proporcional a su grado de influencia.* En la capa de salida, la regla de actualización para los pesos es muy similar a la de los Perceptrones. Hay dos diferencias: en vez de utilizar los valores x_i de la entrada, se utilizan los provenientes de la capa oculta, es decir, los y_j; además, la regla contiene un término adicional para el gradiente de la función de activación. Si denotamos al error $\varepsilon_k = (d_k - z_k)$, entonces la regla de actualización es:

$$w_{kj} \leftarrow w_{kj} + \mu y_j \varepsilon_k g'(b_k) \tag{4.1}$$

donde $g'(\cdot)$ es la derivada de la función de activación $g(\cdot)$ y μ el ya familiar factor de aprendizaje. Ahora vemos que si no hubiésemos cambiado la función de activación escalón por una sigmoide entonces no podríamos calcular este término.

Por conveniencia, definamos un nuevo término de error Δ_k:

$$\Delta_k = g'(b_k)\varepsilon_k \tag{4.2}$$

con lo que nuestra regla de actualización se convierte en

$$w_{kj} \leftarrow w_{kj} + \mu y_j \Delta_k \tag{4.3}$$

Para modificar las conexiones para la primera capa necesitamos definir una cantidad análoga al término de error que se produce en la salida de la red. Aquí es donde aplicamos la retropropagación. La idea es que el nodo j-ésimo es "responsable" de una fracción del error Δ_k en cada uno de los nodos de salida a los cuales está conectado. Luego, los Δ_k son divididos en trozos acorde a las intensidades entre el nodo oculto y los de salida, y propagados para proveer los Δ_j de la capa oculta (Figura 4.13). Siguiendo esta idea, la regla de propagación de los valores Δ es

$$\Delta_j = g'(a_j) \sum_{k=1}^{L} w_{kj} \Delta_k \tag{4.4}$$

Con esta simplificación, la regla de actualización para la primera capa es prácticamente idéntica a la de la segunda:

$$w_{ji} \leftarrow w_{ji} + \mu x_i \Delta_j \tag{4.5}$$

Un detalle antes de exponer el algoritmo. Una de las gracias de las funciones de activación que hemos enunciado antes es que las derivadas se expresan de manera muy sencilla (Cuadro 4.1):

Enunciamos entonces el algoritmo.

Retropropagación del Error
Entrada: Ejemplos $\{\vec{x}_i\}$, Factor de Aprendizaje μ

1. Inicializar los pesos y umbrales aleatoriamente.

Nombre	Logística	Tanh
$g(x)$	$\frac{1}{1+e^{-x}}$	$\frac{1-e^{-2x}}{1+e^{-2x}}$
$g'(x)$	$g(x)(1-g(x))$	$1-g(x)^2$

Cuadro 4.1: Las funciones de activación más comunes y sus derivadas.

2. Repetir

   a) Para cada ejemplo $\vec{x}$

   1) $\vec{z} \leftarrow \text{RNM}(\vec{x})$
   2) $\vec{\varepsilon} \leftarrow \vec{d} - \vec{z}$
   3) Para $k = 1, \ldots, L$,
      $\Delta_k \leftarrow g'(b_k)\varepsilon_k$
   4) Para $k = 1, \ldots, L$ y $j = 1, \ldots, M$
      $w_{kj} \leftarrow w_{kj} + \mu y_j \Delta_k$
   5) Para $k = 1, \ldots, L$
      $w_{k0} \leftarrow w_{k0} + \mu \Delta_k$
   6) Para $j = 1, \ldots, M$,
      $\Delta_j \leftarrow g'(a_j) \sum_{k=1}^{L} w_{kj}\Delta_k$
   7) Para $j = 1, \ldots, M$ y $i = 1, \ldots, N$
      $w_{ji} \leftarrow w_{ji} + \mu x_i \Delta_j$
   8) Para $j = 1, \ldots, M$
      $w_{j0} \leftarrow w_{j0} + \mu \Delta_j$

   hasta que la red haya convergido.

donde $\text{RNM}(\vec{x})$ es la respuesta de la red ante el ejemplo $\vec{x}$. Una observación importante es con respecto al orden en que las entradas se le deben presentar a la red. Una pasada completa de todas las muestras se denomina *época*. Se aconseja determinar tras cada época un nuevo orden, para mejorar la convergencia.

Ejemplo 4 *Para la red de la figura (4.14), la misma del ejemplo (3), aplicar una pasada para los ejemplos (0.0 -1.5 2.0), (3.0 1.0 0.0) y (-0.5 -2.0 3.0) cuyas respuestas deseadas son (0.5 0.0), (1.0 0.0) y (0.8 0.2) respectivamente. El factor de aprendizaje es* $\mu = 0{,}2$.

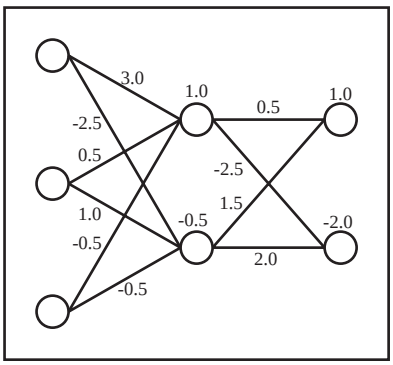

Figura 4.14: Ejemplo.

Ya hemos calculado la respuesta de la red neuronal para el primer ejemplo. De hecho, RNM(0.0 -1.5 2.0) = (0.774 0.062).

Calculemos las actualizaciones para la primera entrada. Vamos a necesitar los resultados intermedios obtenidos en el cálculo anterior. Estos se pueden apreciar en la figura (4.16).

El vector de errores es

$$\vec{\varepsilon} = \begin{pmatrix} 0{,}5 \\ 0{,}0 \end{pmatrix} - \begin{pmatrix} 0{,}774 \\ 0{,}062 \end{pmatrix} = \begin{pmatrix} -0{,}274 \\ -0{,}062 \end{pmatrix}$$

Los dos Δ *son*

$$\begin{aligned} \Delta_1 &= g'(b_1)\varepsilon_1 \\ &= g(b_1)(1-g(b_1))\varepsilon_1 \\ &= 0{,}774(1-0{,}774)(-0{,}274) \\ &= -0{,}048 \end{aligned}$$

y

$$\begin{aligned} \Delta_2 &= g'(b_2)\varepsilon_2 \\ &= g(b_2)(1-g(b_2))\varepsilon_2 \end{aligned}$$

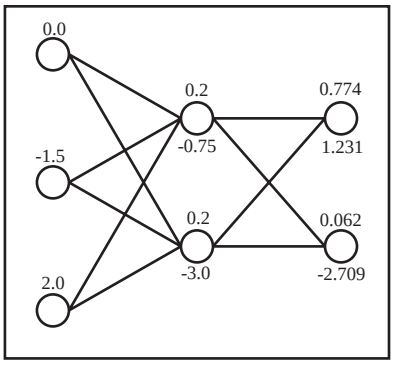

Figura 4.15: Los resultados intermedios para la 1ra entrada. Los valores encima y debajo de las unidades son sus salidas y sus combinaciones lineales respectivamente.

$$\begin{aligned} &= 0{,}062(1-0{,}062)(-0{,}062) \\ &= -0{,}004 \end{aligned}$$

Y los pesos de la segunda capa se modifican entonces como

$$\begin{aligned} w_{10} &= w_{10} + \mu\Delta_1 \\ &= 1{,}0 + 0{,}2 \cdot (-0{,}048) \\ &= 0{,}990 \\ w_{11} &= w_{11} + \mu y_1 \Delta_1 \\ &= 0{,}5 + 0{,}2 \cdot 0{,}321 \cdot (-0{,}048) \\ &= 0{,}497 \\ w_{12} &= w_{12} + \mu y_2 \Delta_1 \\ &= 1{,}5 + 0{,}2 \cdot 0{,}047 \cdot (-0{,}048) \\ &= 1{,}500 \\ w_{20} &= w_{20} + \mu\Delta_1 \\ &= -2{,}0 + 0{,}2 \cdot (-0{,}004) \\ &= -2{,}000 \\ w_{21} &= w_{21} + \mu y_1 \Delta_2 \\ &= -2{,}5 + 0{,}2 \cdot 0{,}321 \cdot (-0{,}004) \\ &= -2{,}500 \\ w_{22} &= w_{22} + \mu y_2 \Delta_2 \\ &= 2{,}0 + 0{,}2 \cdot 0{,}047 \cdot (-0{,}004) \\ &= 2{,}000 \end{aligned}$$

Los Δ de la capa oculta son

$$\begin{aligned} \Delta_1 &= g'(a_1) \sum_{k=1}^{L} w_{k1}\Delta_k \\ &= 0{,}321(1-0{,}321)(0{,}497 \cdot (-0{,}048) + (-2{,}5) \cdot (-0{,}004)) \\ &= -0{,}015 \end{aligned}$$

y

$$\begin{aligned} \Delta_2 &= g'(a_2) \sum_{k=1}^{L} w_{k2}\Delta_k \\ &= 0{,}047(1-0{,}047)(1{,}5 \cdot (-0{,}048) + 2{,}0 \cdot (-0{,}004)) \\ &= -0{,}079 \end{aligned}$$

Luego actualizamos los pesos de la primera capa:

$$\begin{aligned} w_{10} &= w_{10} + \mu\Delta_1 \\ &= 1{,}0 + 0{,}2 \cdot (-0{,}015) \\ &= 0{,}997 \\ w_{11} &= w_{11} + \mu x_1 \Delta_1 \\ &= 3{,}0 + 0{,}2 \cdot 0{,}0 \cdot (-0{,}015) \\ &= 3{,}000 \\ w_{12} &= w_{12} + \mu x_2 \Delta_1 \\ &= 0{,}5 + 0{,}2 \cdot (-1{,}5) \cdot (-0{,}015) \\ &= 0{,}504 \\ w_{13} &= w_{13} + \mu x_3 \Delta_1 \\ &= -0{,}5 + 0{,}2 \cdot 2{,}0 \cdot (-0{,}015) \\ &= -0{,}506 \\ w_{20} &= w_{20} + \mu\Delta_2 \\ &= -0{,}5 + 0{,}2 \cdot (-0{,}079) \\ &= -0{,}515 \\ w_{21} &= w_{21} + \mu x_1 \Delta_2 \\ &= -2{,}5 + 0{,}2 \cdot 0{,}0 \cdot (-0{,}079) \end{aligned}$$

$$
\begin{aligned}
& = -2{,}500 \\
w_{22} &= w_{22} + \mu x_2 \Delta_2 \\
& = 1{,}0 + 0{,}2 \cdot (-1{,}5) \cdot (-0{,}079) \\
& = 1{,}024 \\
w_{23} &= w_{23} + \mu x_3 \Delta_2 \\
& = -0{,}5 + 0{,}2 \cdot 2{,}0 \cdot (-0{,}079) \\
& = -0{,}532
\end{aligned}
$$

La figura (4.16) refleja los resultados de esta primera actualización de las conexiones. Ahora repetimos el proceso para los ejemplos 2 y 3.

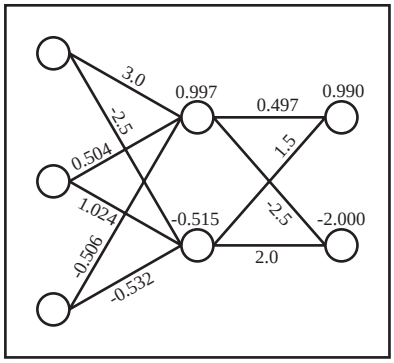

Figura 4.16: Los resultados intermedios para la 2da entrada y los pesos actualizados.

Presentando el ejemplo 2 y calculando el error en la salida, se obtiene:

$$\vec{\varepsilon} = \begin{pmatrix} 1{,}0 \\ 0{,}0 \end{pmatrix} - \begin{pmatrix} 0{,}816 \\ 0{,}011 \end{pmatrix} = \begin{pmatrix} 0{,}184 \\ -0{,}011 \end{pmatrix}$$

Y aplicando el resto del algoritmo de retropropagación:

$$\Delta_k = \begin{pmatrix} 0{,}0277 \\ 0{,}000 \end{pmatrix}$$

$$(w)_{kj} = \begin{pmatrix} 0{,}502 & 1{,}500 \\ -2{,}500 & 2{,}000 \end{pmatrix}$$

$$(w)_{k0} = \begin{pmatrix} 0{,}996 \\ -2{,}001 \end{pmatrix}$$

$$\Delta_j = \begin{pmatrix} 0{,}014 \\ 0{,}041 \end{pmatrix}$$

$$(w)_{ji} = \begin{pmatrix} 3{,}009 & 0{,}507 & -0{,}506 \\ -2{,}475 & 1{,}032 & -0{,}532 \end{pmatrix}$$

$$w_{j0} = \begin{pmatrix} 1{,}000 \\ -0{,}508 \end{pmatrix}$$

Y por último para el tercer ejemplo.

$$\vec{\varepsilon} = \begin{pmatrix} 0{,}8 \\ 0{,}2 \end{pmatrix} - \begin{pmatrix} 0{,}749 \\ 0{,}118 \end{pmatrix} = \begin{pmatrix} 0{,}051 \\ 0{,}082 \end{pmatrix}$$

$$\Delta_k = \begin{pmatrix} 0{,}010 \\ 0{,}009 \end{pmatrix}$$

$$(w)_{kj} = \begin{pmatrix} 0{,}503 & 1{,}500 \\ -2{,}500 & 2{,}000 \end{pmatrix}$$

$$(w)_{k0} = \begin{pmatrix} 0{,}998 \\ -1{,}999 \end{pmatrix}$$

$$\Delta_j = \begin{pmatrix} -0{,}017 \\ 0{,}031 \end{pmatrix}$$

$$(w)_{ji} = \begin{pmatrix} 3{,}010 & 0{,}514 & -0{,}516 \\ -2{,}478 & 1{,}019 & -0{,}513 \end{pmatrix}$$

$$w_{j0} = \begin{pmatrix} 0{,}997 \\ -0{,}501 \end{pmatrix}$$

Con lo que obtuvimos los pesos de la red resultante (Figura 4.17), para una pasada.

Ejercicio 5 *Realizar una pasada del algoritmo de retropropagación del error para los siguientes pares de entrada-respuesta:*

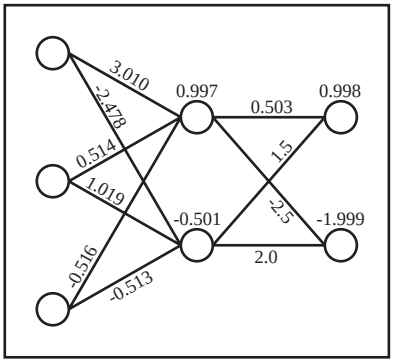

Figura 4.17: La configuración resultante tras una pasada del algoritmo de retropropagación del error.

Entrada	*Respuesta*
(0.6 0.0)	*(0.5 0.5)*
(0.7 -0.8)	*(0.7 0.4)*
(0.1 1.0)	*(1.0 1.0)*

La red neuronal es la que se ilustra en la figura (4.18), y el factor de aprendizaje a utilizar es $\mu = 1{,}0$.

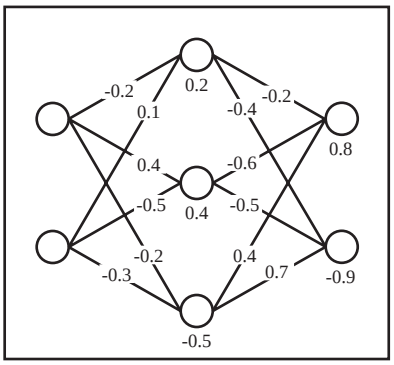

Figura 4.18: Ejercicio.

Capítulo 5

Combinadores Lineales Adaptivos (Adalines)

En este capítulo veremos modelos neuronales más sencillos que las Redes Neuronales Multicapa. Estos son simples en su diseño y sus propiedades los hacen ideales para el diseño de filtros canceladores de ruído. Son tremendamente usados en la práctica. Se trata de los *Combinadores Lineales Adaptivos*, en la literatura generalmente denominados *Adalines* (ADaptive LINear Element). Fueron desarrollados en 1959 por Widrow y Hoff. Se trata de una red de una sola capa de pesos con múltiples entradas, una salida y un combinadores lineal (sin función de activación) como ilustra la figura (5.1).

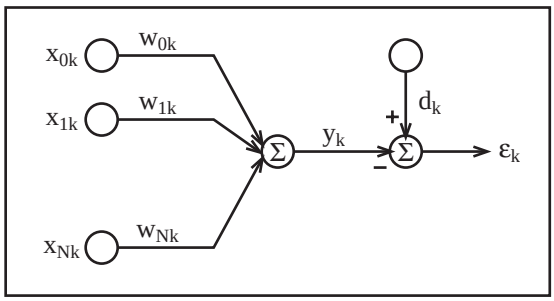

Figura 5.1: El Combinador Lineal Adaptivo con la respuesta deseada y la señal de error.

La respuesta y_k entregada por el combinador es

$$y_k = \vec{w_k} \cdot \vec{x_k} = \sum_{i=1}^{N} w_{ik} x_{ik} \tag{5.1}$$

A primera vista, pareciera tratarse de un tipo de red más básico que el mismo Perceptrón, ya que no hay función de activación. En rigor esto es cierto, pero el uso que se les da es completamente diferente, y en ese papel se desempeñan bastante bien.

El empleo de los Combinadores Lineales Adaptivos se relaciona más bien con señales (temporales), y sus pesos se van ajustando dinámicamente durante su operación.

Pensemos en N señales $x_0, \ldots, x_N$. En general, estas provienen de funciones contínuas, pero se trabaja sobre *muestras* de ellas, i.e. mediciones a intervalos de tiempo constante, que en la figura ejemplar (5.2) vienen representadas por los puntos. La muestra k-ésima de la señal x_i la denotamos con x_{ik}. En una combinación lineal, superponemos las señales ponderando cada una de ellas por un factor w_i diferente, resultando en una nueva señal como el de la figura. En el combinador, estas ponderaciones también varían, y

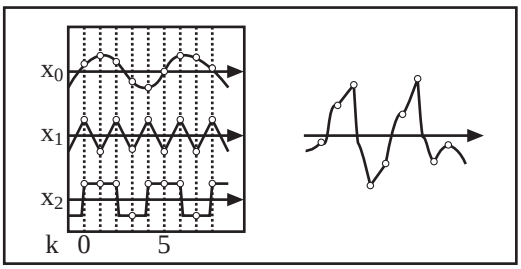

Figura 5.2: Una combinación lineal de tres señales.

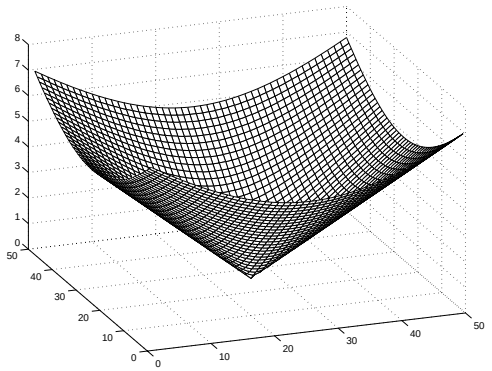

Figura 5.3: Error absoluto para dos pesos

el valor de w_i en el instante k se denota por w_{ik}.

Para entrenar la red y obtener el resultado deseado, lo que se hace es comparar la salida del combinador con el valor deseable d_k del instante d_k. El error viene dado por la diferencia

$$\varepsilon_k = |d_k - y_k| \tag{5.2}$$

Una forma equivalente para eliminar este error es minimizando el error cuadrático,

$$\varepsilon_k^2 = (d_k - y_k)^2 \tag{5.3}$$

Para minimizar el error, empleamos el *método del gradiente*. Para entender el método veamos la ilustración del error. La figura (5.3) representa el error absoluto y la figura (5.4) el error cuadrático para un combinador de dos pesos.

De las figuras se desprende que efectivamente una minimización del error cuadrático es equivalente a aquella del error absoluto. La idea del método del gradiente consiste en modificar los pesos en la dirección del máximo descenso. Como no existen óptimos locales nos aseguramos de que eso funciona. La dirección del máximo descenso viene dada por la contraria al gradiente, luego si $\varepsilon_k^2 = (d_k - y_k)^2$, entonces

$$\begin{aligned}
\nabla_k(\varepsilon_k^2) &= \begin{pmatrix} \frac{\partial}{\partial w_{1k}} \\ \frac{\partial}{\partial w_{2k}} \\ \vdots \\ \frac{\partial}{\partial w_{Nk}} \end{pmatrix} (\varepsilon_k^2) \\
&= 2\varepsilon_k \begin{pmatrix} \frac{\partial}{\partial w_{1k}} \\ \frac{\partial}{\partial w_{2k}} \\ \vdots \\ \frac{\partial}{\partial w_{Nk}} \end{pmatrix} (d_k - y_k) \\
&= 2\varepsilon_k \begin{pmatrix} \frac{\partial}{\partial w_{1k}} \\ \frac{\partial}{\partial w_{2k}} \\ \vdots \\ \frac{\partial}{\partial w_{Nk}} \end{pmatrix} (d_k - \sum_{i=1}^{N} w_{ik} x_{ik}) \\
&= -2\varepsilon_k \vec{x_k}
\end{aligned} \tag{5.4}$$

El gradiente ∇_k corresponde a aquel asociado a los pesos w_{ik} del instante k.

Utilizando este valor como aproximación, es decir, el valor instantáneo del gradiente del error, podemos construir una regla sencilla para actualizar los pesos llamada *regla LMS* (Least Mean

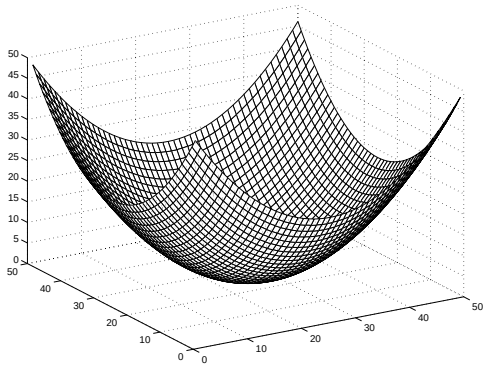

Figura 5.4: Error cuadrático para dos pesos

Square). El método del gradiente dice que para minimizar la función de error, hay que ejecutar

$$\vec{w_{k+1}} = \vec{w_k} - \mu \nabla_k(\varepsilon_k^2) \tag{5.5}$$

donde μ es el factor de aprendizaje que regula el tamaño de las modificaciones. Y reemplazando el valor para el gradiente tenemos la regla buscada

$$\vec{w_{k+1}} = \vec{w_k} + 2\mu\varepsilon_k\vec{x_k} \tag{5.6}$$

Luego, estamos listos para enunciar el algoritmo LMS para un instante de tiempo k:

Algoritmo LMS
Entrada: Muestra de señal $\vec{x}_k$, Resultado deseado d_k, Factor de Aprendizaje μ

1. $y_k \leftarrow \sum_{i=1}^{N} w_{ik}x_{ik}$
2. $\varepsilon_k \leftarrow (d_k - y_k)$
3. Para cada peso w_{ik}
   a) $w_{i(k+1)} \leftarrow w_{ik} + 2\mu\varepsilon_k x_{ik}$

Ejemplo 5 *Se tienen tres señales de entrada y una salida deseada dadas por el cuadro a continuación. Los pesos iniciales son* $w_{10} = 3{,}0$, $w_{20} = 1{,}0$ *y* $w_{30} = 1{,}5$ *y el factor de aprendizaje es* $\mu = 0{,}5$. *Calcular la respuesta* y_k *del combinador lineal adaptivo.*

k	1	2	3	4	5	6
x_1	0,5	1,0	1,5	1,0	0,5	1,0
x_2	0,8	0,9	0,1	−0,8	0,0	−0,3
x_3	0,0	0,94	0,88	0,80	0,70	0,58
d	15,0	12,9	4,3	−11,6	−1,9	−2,5
k	7	8	9	10	11	12
x_1	1,5	1,0	0,5	1,0	1,5	1,0
x_2	0,7	1,0	0,4	−0,5	−1,0	−0,5
x_3	0,44	0,29	0,14	−0,02	−0,18	−0,33
d	16,4	19,0	8,0	−1,3	−4,7	0,8

La primera respuesta se obtiene inmediatamente aplicando la combinación lineal

$$\begin{aligned} y_1 &= w_{11}x_{11} + w_{21}x_{21} + w_{31}x_{31} \\ &= 3{,}0 \cdot 0{,}5 + 1{,}0 \cdot 0{,}8 + 1{,}5 \cdot 0{,}0 \\ &= 2{,}3 \end{aligned}$$

Luego el error vale

$$\varepsilon_1 = 15{,}0 - 2{,}3 = 12{,}7$$

Actualizando los pesos, obtenemos

$$\begin{aligned} w_{12} &= w_{11} + 2\mu\varepsilon_1 x_{11} \\ &= 3{,}0 + 2 \cdot 0{,}5 \cdot 12{,}7 \cdot 0{,}5 \\ &= 9{,}35 \end{aligned}$$

$$\begin{aligned} w_{22} &= w_{21} + 2\mu\varepsilon_1 x_{21} \\ &= 1{,}0 + 2 \cdot 0{,}5 \cdot 12{,}7 \cdot 0{,}8 \\ &= 11{,}16 \end{aligned}$$

$$\begin{aligned} w_{32} &= w_{31} + 2\mu\varepsilon_1 x_{31} \\ &= 1{,}5 + 2 \cdot 0{,}5 \cdot 12{,}7 \cdot 0{,}0 \\ &= 1{,}5 \end{aligned}$$

Veamos la tabla siguiente, que contiene todas las iteraciones del algoritmo.

k	1	2	3	4
w_1	3,000	9,350	1,446	11,864
w_2	1,000	11,160	4,046	4,741
w_3	1,500	1,500	−5,930	0,182
y	2,300	20,804	−2,645	8,217
ε	12,700	−7,904	6,945	−19,817
k	5	6	7	8
w_1	−7,953	−1,430	7,471	−2,653
w_2	20,595	20,595	17,925	13,201
w_3	−15,672	−6,539	−1,376	−4,346
y	−14,947	−11,401	23,149	9,288
ε	13,047	8,901	−6,749	9,712
k	9	10	11	12
w_1	7,059	4,819	9,217	9,205
w_2	22,913	21,121	18,922	18,930
w_3	−1,530	−2,157	−2,245	−2,244
y	12,481	−5,698	−4,692	0,481
ε	−4,481	4,398	−0,008	0,319

Observando la evolución del error, vemos que inicialmente este oscila de manera descontrolada, hasta que en los últimos pasos tiende a anularse.

5.1. Aplicación de las Adalines

Como hemos mencionado anteriormente, a pesar de su simplicidad, las Adalines tienen un campo muy amplio de aplicación. Tal vez la aplicación más importante sea la cancelación de ruído.

Supongamos que estamos registrando una señal, ya sea desde un micrófono (audio) u otro dispositivo y queremos transmitirla lo más limpia posible a un destino. Es difícil evitar que la señal capturada esté libre del ruído proveniente del medio ambiente. Lo que se puede hacer para resolver este problema es capturar una segunda señal de referencia y tratar de eliminar el ruído de la señal primaria guiándonos por la segunda.

Más formalmente, se pretende obtener una señal s a partir de otra ruidosa $s + n_0$, donde n_0 es un ruido no correlacionado con s. n_1 es otro ruído correlacionado con n_0, pero no con s. Además, s, n_0 y n_1 se suponen estadísticamente estacionarios y de valor medio cero.

Luego, el montaje del cancelador de ruído adaptivo sería el que se puede apreciar en la figura (5.6).

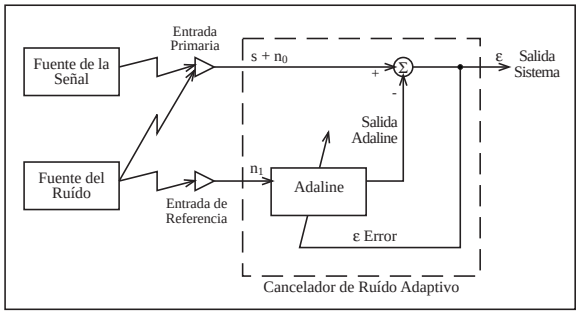

Figura 5.5: Montaje del Cancelador de Ruído

Podemos ver fácilmente a partir de la construcción del cancelador que la función que cumple el Adaline es tratar de imitar la señal primaria $s + n_0$. Es decir, trata de construir una respuesta y que se asemeja lo mejor posible a $s + n_0$. Sin embargo esto no le puede resultar, pues su entrada n_1 sólamente se relaciona con el ruído n_0 y no con s. En conclusión, la respuesta del Adaline y a lo más puede resultar similar a n_0 y no a s, con lo que el error y por ende la

respuesta del sistema cancelador será:

$$\varepsilon = s + n_0 - y \approx s + n_0 - n_0 = s$$

Es decir, la respuesta del cancelador sería la señal limpia s. Existe un resultado teórico más fuerte aún, que dice que si la señal de referencia no estuviera correlacionada con el ruído n_0, entonces los pesos de la Adaline se anulan y por lo tanto este se apaga como filtro.

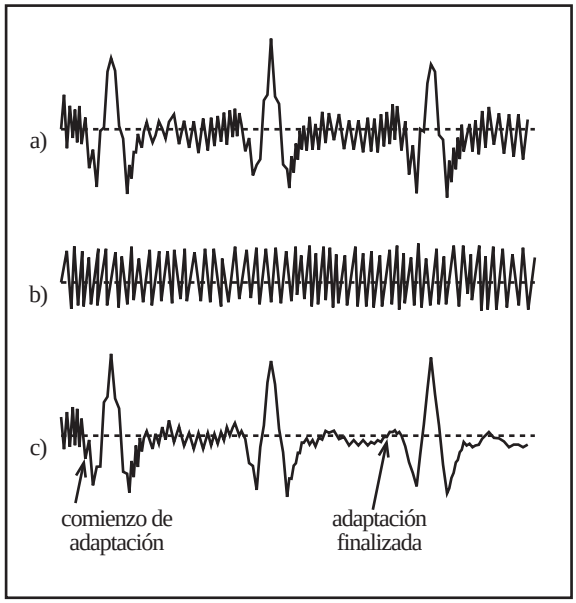

Figura 5.6: Cancelación del ruído electrocardiográfico: (a) señal primaria; (b) entrada de referencia; (c) salida del cancelador.

Capítulo 6

Mapas de Kohonen

En las redes neuronales que hemos visto hasta ahora, la estrategia global que se ha adoptado para la clasificación ha sido la de partir con una red genérica que sea capaz de representar o aproximar una gran variedad de funciones. El entrenamiento de la red consiste en presentarle iterativamente ejemplos que han sido previamente clasificados por algún experto, y ajustar sus pesos tratando de disminuir el error en la respuesta. En el fondo, podemos ver a la red neuronal como una poderosa función multifacética con un número finito de perillas o parámetros que nosotros podemos ajustar y así modelar a una función particular que queremos aproximar.

Hemos visto que esta aproximación resulta muy versátil y los resultados más que satisfactorios. A pesar de este hecho, hay algunas críticas que se le pueden hacer. Una de las desventajas de la Red Neuronal Multicapa es que si bien puede aproximar cualquier función, una vez entrenada es una caja negra; la interpretación de su significado es una tarea imposible realizar ya para un red de mediano tamaño, la que la información que contiene se encuentra distribuída en todas sus conexiones. Ni hablar de un orden topológico de sus neuronas que nos permita tratar de identificar a un grupo de ellas como responsables de cumplir una función determinada.

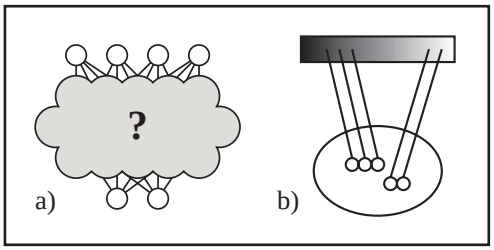

Figura 6.1: Interpretatividad de las redes neuronales: (a) La Red Neuronal Multicapa es una caja negra; (b) El cerebro humano presenta un orden topológico que mapea las neuronas vecinas con estímulos parecidos.

Sin embargo, no ocurre lo mismo con el cerebro humano. Se puede deducir una organización topológica en la corteza cerebral observando las disfunciones y alteraciones en el comportamiento de individuos que han sufrido lesiones cerebrales como hemorragias, tumores o malfunciones. Regiones muy específicas cumplen diferentes roles. Más aún, los estudios sugieren que existen verdaderos mapas en donde la ubicación de una neurona está directamente asociada a una señal de modalidad y calidad particular. Nombremos algunos ejemplos:

1. Corteza Visual Primaria:
   - *a*) Campo visual
   - *b*) Mapa de orientación de líneas
   - *c*) Mapa de colores
2. Corteza Auditiva
   - *a*) Mapa Tonotópico
3. Otros mapas sensitivos
   - *a*) Mapa Somatotópico de la superficie de la piel
4. Mapa motriz
   - *a*) Prácticamente idéntico al Somatotópico.
5. Mapas de Alto Nivel
   - *a*) Usualmente están desordenados o contienen una métrica más complicada que no se interpreta fácilmente.
   - *b*) Algunas células responden a patrones complejos de estímulos o se relacionan con cualidades sensitivas abstractas.
   - *c*) En la comprensión del lenguaje existen mapas semánticos.

En general, se puede concluir que la información del cerebro se encuentra espacialmente organizada.

6.1. Modelamiento de Espacios No-Homegéneos

En la vida real, muchas veces nos enfrentaremos a problemas en que disponemos de un conjunto de patrones de alta dimensionalidad, i.e. los patrones tienen un número N elevado de componentes, dentro de los cuales queremos encontrar relaciones entre ellos. Una forma de enfrentar el análisis sería definirse una distancia entre los patrones que sirva como indicador del *nivel de relación*. Mientras más corta la distancia entre dos patrones, más relacionados están. Si la dimensión N es alta, aparecen generalmente tres problemas.

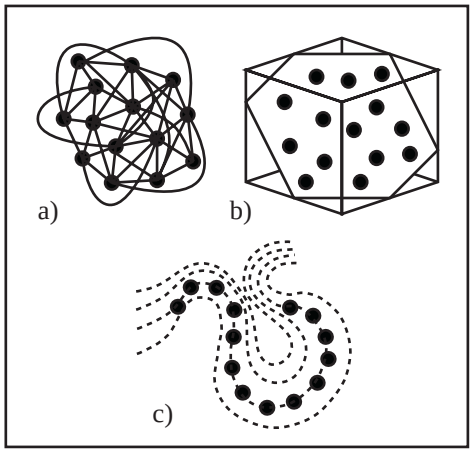

Figura 6.2: Problemas en el análisis: (a) Se fijó un valor para la distancia. Una conexión entre dos patrones indica que ellos tienen esta separación. El análisis se vuelve imposible; (b) Si bien los patrones tienen 3 componentes, el problema en realidad está confinado en un espacio de 2 dimensiones; (c) El espacio en el cual se encuentran los patrones es curvo.

1. Es difícil realizar un análisis a partir de la medida de distancia sola, pues entender una relación que se extiende sobre tantas dimensiones es una tarea complicada.
2. En muchos problemas, la dimensión real del conjunto de patrones es menor que la que

se sospecha, incluso pueden haber componentes que son redundantes.

3. El espacio de los patrones no es homogéneo. Esto significa que a pesar que dos patrones tengan la misma distancia real, mirado desde el punto de vista de lo que se pretende resolver, uno puede estar más cercano al otro. Dicho de manera informal, el espacio de los patrones es curvo.

Un algoritmo adecuado debe ser capaz de lidiar con estos tres problemas simultáneamente. La propuesta que veremos a continuación es aquella diseñada por el islandés Kohonen, que se conocen bajo el nombre de los *Mapas de Kohonen*.

Para distinguir ambos espacios que hemos mencionado, usaremos el término *Espacio de Patrones* para denotar a aquel dentro del cual viven los patrones y que tiene una dimensión N igual al número de sus componentes, y *Espacio del Objetivo* a aquel intrínsecamente relacionado con el problema.

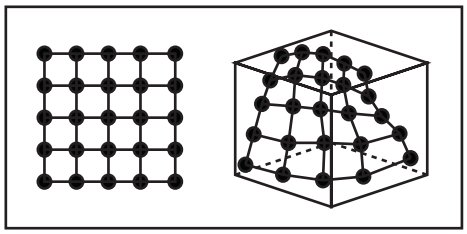

Figura 6.3: Modelamiento de un subespacio no-homogéneo. En este caso, se aproxima una superficie curva dentro de un espacio de tres dimensiones.

Todo parte con un conjunto de patrones, digamos que disponemos de P de ellos, que viven en un espacio de alta dimensionalidad, siendo cada uno de ellos caracterizado por N variables. La idea de Kohonen consiste en darse un espacio objetivo de dimensión M estrictamente inferior a la original. Para $M = 1$ tendríamos una línea, para $M = 2$ una superficie, para $M = 3$ un cuerpo, etc. Generalmente, se utiliza $M = 2$. Luego, llenamos este espacio con puntos de referencia equiespaciados formando una grilla. Una vez formada esta grilla, se acomoda dentro del espacio de los patrones modelando cuya topología subyacente. La figura (6.3) ilustra este procedimiento para un espacio de patrones de dimensión $N = 3$ y un espacio objetivo de $M = 2$.

Si se hace esto correctamente, se obtiene como resultado una representación del espacio objetivo real. Obviamente ésta sólo es una aproximación en donde los puntos de referencia quedan correctamente emplazados y los espacios *entremedio* quedan interpolados. Se podría aumentar el número de referencias para mejorar la aproximación pero el problema se vuelve computacionalmente prohibitivo, y a partir de cierto punto empeora su calidad.

Esta es la idea que formalizaremos a continuación.

6.2. Mapa Auto-Organizativo de Kohonen

En inglés conocidos como *Self-Organizing Maps* o simplemente *SOM*, estos mapas son muy utilizados como herramientas de análisis para descubrir relaciones entre un conjunto grande de datos. Se utiliza mucho en aplicaciones de *Data Mining*.

Las características principales son:

1. SOM permite proyectar espacios de señales de alta dimensionalidad en otros de dimen-

sión inferior, permitiendo incluso visualizarlos si $M = 1, 2$ ó 3.

2. Es una red neuronal con forma de grilla. Se le presenta secuencialmente un patrón tras otro, y cada uno estimula a una neurona la cual luego es ajustada levemente junto a sus células vecinas.

3. El aprendizaje es no-supervisado: Eso quiere decir que el aprendizaje no requiere de una etiquetación previa de los patrones por medio de un experto. Como el nombre sugiere, estos mapas se autoorganizan y descubren por si solos una categorización adecuada, acumulando los patrones relacionados dentro de regiones contiguas del espacio objetivo.

4. Las localidades de las respuestas de los estímulos tienden a ordenarse y asentarse sobre grupos específicos de células como si se estuviera generando un sistema de coordenadas con sentido, dedicado a las diferentes características que presentan las entradas.

5. Cada célula actúa como un decodificador para un tipo especial de señal, pues sólo responde ante un grupo reducido de estímulos parecidos.

6. Esta segmentación espacial de grupos de estímulos (en el espacio de patrones) y su organización topológica en el espacio objetivo resulta ser un método muy eficiente para encontrar relaciones entre subconjuntos.

7. La idea de los mapas es escalable a dimensiones y tamaños arbitrarios, aunque actualmente se sugieren mapas jerárquicos para problemas de gran tamaño.

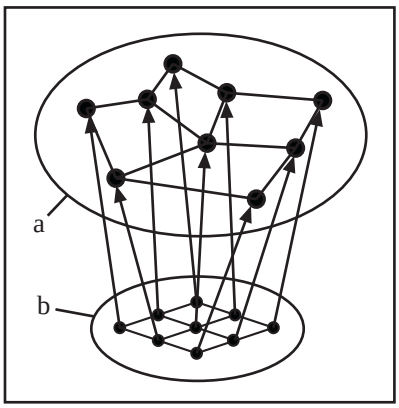

Figura 6.4: El espacio objetivo (b) está conformado por un arreglo de vectores prototipos que representan puntos en el espacio de patrones (a).

El Mapa de Kohonen está conformado por los siguientes elementos (Figura (6.4)):

1. Arreglo de Vectores Prototipos: Se dispone de un arreglo M-dimensional de vectores que corresponden a los puntos de referencia. Estos se denotan técnicamente como *Vectores Prototipo*, y consisten en elementos N-dimensionales.

2. Espacio de patrones: Se constituye por un espacio N-dimensional donde cada eje x_i toma valores iguales a los que la componente i-ésima de los patrones puede asumir. Los Vectores Prototipo son sólamente algunos miembros de este espacio.

3. Espacio objetivo: El arreglo de Prototipos corresponde al espacio objetivo, el cual es representado intrínsecamente por la disposición de los prototipos dentro del arreglo M-dimensional. Dos elementos contiguos son dos puntos vecinos de distancia 1 en el espacio objetivo.

Como los Mapas de Kohonen son redes neuronales, la gran pregunta que sigue es ¿Cómo hay que entrenar la red para que logre representar el mapa deseado? La idea en la cual se basa el entrenamiento es muy sencilla, y se conoce bajo el nombre de *Cuantización Vectorial*.

6.3. Aprendizaje no supervisado del Mapa de Kohonen

Ya mecionado, el método utilizado en el entrenamiento de la red neuronal es la Cuantización Vectorial, que responde al esquema de los *Aprendizajes Competitivos*. Esto significa, que ante cada patrón del conjunto de entrenamiento que se le presenta a la red, todas la neuronas compiten por adjudicarse el estímulo, y sólo una entre ellas gana.

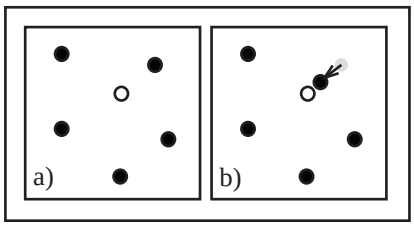

Figura 6.5: Cuantización Vectorial: El estímulo (blanco) es comparado con los prototipos (negros). El más cercano resulta ganador, y tiene derecho a acercarse al estímulo en una cantidad proporcional a la distancia que existe entre ellos.

En la figura (6.5) se ve un ejemplo. Los prototipos los denotaremos con $\vec{m}_l$, donde l es el índice que corre sobre todos los prototipos y van de $l = 1, \ldots, L$. En la Cuantización Vectorial, se presenta un patrón $\vec{x}^p$ el cual es comparado con todos los prototipos mediante una distancia previamente establecida, por ejemplo, la distancia euclidiana. En esta comparación, el prototipo con menor distancia al estímulo, $\vec{m}_c$, resulta vencedor, y se ajusta según la regla

$$\vec{m}_c \leftarrow \vec{m}_c + \mu(t)(\vec{x}^p - \vec{m}_c) \qquad (6.1)$$

donde $\mu(t)$ es el factor de aprendizaje y $0 \leq \mu(t) < 1$.

A este sencillo procedimiento le agregamos la noción de vecindad para completar el algoritmo de aprendizaje. Se define una vecindad N_c alrededor de la neurona ganadora, en el espacio objetivo. Entonces, en vez de mover sólo el prototipo ganador, movemos a todos los que se encuentran en la vecindad. Esta regla adicional fuerza la formación del orden topológico que estamos buscando. Usualmente, N_c es grande al comienzo, y decrece a medida que transcurre el algoritmo, terminando en $N_c = \{\vec{m}_c\}$, es decir, moviendo sólo a la neurona ganadora.

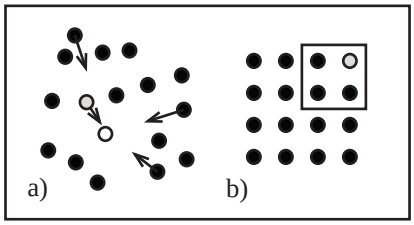

Figura 6.6: El ganador (gris) se mueve hacia el estímulo junto con los prototipos dentro de su vecindad (b) en el espacio objetivo.

Además $\mu(t)$ es una función del tiempo, es decir, del número de iteraciones. La función $\mu(t)$ debe ser monotónicamente decreciente, y se recomienda que parta desde un valor cercano a 1 y termine en otro cercano a cero. La elección de $\mu(t)$ de esta forma pretende inducir un ordenamiento global durante el comienzo del algoritmo, mientras que en la etapa final se realiza un

ajuste fino.

Veamos el algoritmo.

Entrenamiento Mapa de Kohonen
Entrada: ejemplos $\{\vec{x}^p\}$

1. Inicializar los prototipos $\vec{m}_l$
2. Inicializar μ y N_c
3. Repetir

   a) Para cada ejemplo $\vec{x}^p$

      1) Encontar $\vec{m}_c$ tal que minimice

$$\|\vec{x}^p - \vec{m}_l\|$$

      2) Para cada $\vec{m}_l \in N_c$

$$\vec{m}_l \leftarrow \vec{m}_l + \mu(\vec{x}^p - \vec{m}_l)$$

   b) Actualizar μ y N_c

   hasta que no se produzcan cambios notables en el mapa.

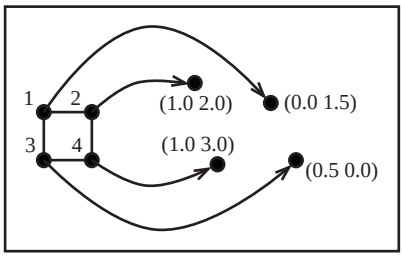

Figura 6.7: Ejemplo.

Ejemplo 6 *Para la figura (6.7), realizar dos pasadas del algoritmo de entrenamiento del Mapa de Kohonen para los patrones*

Patrón	x_1	x_2
1	3,5	1,0
2	1,5	0,5

con μ y el radio para N_c, $\rho(N_c)$ iguales a

Pasada	μ	$\rho(N_c)$
1	0,5	1
2	0,3	0

utilizando la distancia euclidiana.

Primera pasada: Presentamos el primer patrón y calculamos el ganador.

$$\begin{aligned}\|\vec{x}^1 - \vec{m}_1\| &= \sqrt{(3{,}5-1{,}0)^2+(1{,}0-2{,}0)^2}\\ &= 2{,}693\end{aligned}$$

$$\begin{aligned}\|\vec{x}^1 - \vec{m}_2\| &= \sqrt{(3{,}5-0{,}0)^2+(1{,}0-1{,}5)^2}\\ &= 3{,}536\end{aligned}$$

$$\begin{aligned}\|\vec{x}^1 - \vec{m}_3\| &= \sqrt{(3{,}5-1{,}0)^2+(1{,}0-3{,}0)^2}\\ &= 3{,}202\end{aligned}$$

$$\begin{aligned}\|\vec{x}^1 - \vec{m}_4\| &= \sqrt{(3{,}5-0{,}5)^2+(1{,}0-0{,}0)^2}\\ &= 3{,}162\end{aligned}$$

Luego, $\vec{m}_1$ resulta ganador. Como la vecindad de $\vec{m}_1$ es $N_c = \{\vec{m}_1, \vec{m}_2, \vec{m}_3, \vec{m}_4\}$, entonces actualizamos a todos los vectores.

$$\begin{aligned}\vec{m}_1 &\leftarrow \vec{m}_1 + \mu(\vec{x}^1 - \vec{m}_1)\\ &= (1{,}0; 2{,}0) + 0{,}5(2{,}5; -1{,}0) = (2{,}25; 1{,}5)\end{aligned}$$

$$\begin{aligned}\vec{m}_2 &\leftarrow \vec{m}_2 + \mu(\vec{x}^1 - \vec{m}_2)\\ &= (0{,}0; 1{,}5) + 0{,}5(3{,}5; -0{,}5) = (1{,}75; 1{,}25)\end{aligned}$$

$$\begin{aligned}\vec{m}_3 &\leftarrow \vec{m}_3 + \mu(\vec{x}^1 - \vec{m}_3)\\ &= (1{,}0; 3{,}0) + 0{,}5(0{,}5; -2{,}0) = (1{,}25; 2{,}0)\end{aligned}$$

$$\begin{aligned}\vec{m}_4 &\leftarrow \vec{m}_4 + \mu(\vec{x}^1 - \vec{m}_4)\\ &= (0{,}5; 0{,}0) + 0{,}5(3{,}0; 1{,}0) = (2{,}0; 0{,}5)\end{aligned}$$

Ahora presentamos el segundo ejemplo. Las distancias obtenidas son:

$$\|\vec{x}^2 - \vec{m}_1\| = 1{,}250$$

$$\|\vec{x}^2 - \vec{m}_2\| = 0{,}791$$

$$\|\vec{x}^2 - \vec{m}_3\| = 1{,}521$$

$$\|\vec{x}^2 - \vec{m}_4\| = 0{,}500$$

En esta ronda gana $\vec{m}_4$, y su vecindad nuevamente es todo el conjunto de prototipos. Así que los movemos a todos.

$$\vec{m}_1 \leftarrow (2{,}25; 1{,}5) + 0{,}5(-0{,}75; -1{,}00) = (1{,}875; 1{,}000)$$

$$\vec{m}_2 \leftarrow (1{,}75; 1{,}25) + 0{,}5(-0{,}25; -0{,}75) = (1{,}625; 0{,}875)$$

$$\vec{m}_3 \leftarrow (1{,}25; 2{,}0) + 0{,}5(0{,}25; -1{,}50) = (1{,}375; 1{,}250)$$

$$\vec{m}_4 \leftarrow (2{,}0; 0{,}5) + 0{,}5(-0{,}5; 0{,}0) = (1{,}75; 0{,}5)$$

Procedemos entonces a la segunda pasada del algoritmo. Presentamos el primer ejemplo, y las distancias resultantes son:

$$\|\vec{x}^1 - \vec{m}_1\| = 1{,}625$$

$$\|\vec{x}^1 - \vec{m}_2\| = 1{,}879$$

$$\|\vec{x}^1 - \vec{m}_3\| = 2{,}140$$

$$\|\vec{x}^1 - \vec{m}_4\| = 1{,}820$$

ganando nuevamente $\vec{m}_1$. Sin embargo, μ y N_c han variado, siendo este último igual a $N_c = \{\vec{m}_1\}$.

$$\vec{m}_1 \leftarrow (1{,}875; 1{,}000) + 0{,}3(1{,}625; 0{,}0) = (2{,}363; 1{,}000)$$

Para el segundo ejemplo, calculemos por última vez las distancias:

$$\|\vec{x}^2 - \vec{m}_1\| = 0{,}997$$

$$\|\vec{x}^2 - \vec{m}_2\| = 0{,}395$$

$$\|\vec{x}^2 - \vec{m}_3\| = 0{,}760$$

$$\|\vec{x}^2 - \vec{m}_4\| = 0{,}250$$

Como $\vec{m}_4$ es el más cercano, lo actualizamos:

$$\vec{m}_4 \leftarrow (1{,}75; 0{,}5) + 0{,}3(-0{,}25; 0{,}0) = (1{,}675; 0{,}5)$$

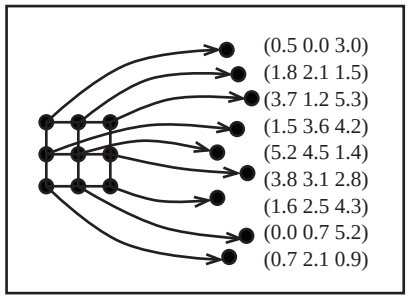

Figura 6.8: Ejercicio.

Ejercicio 6 *Para el Mapa de Kohonen de la figura (6.8), resolver dos pasadas del algoritmo de entrenamiento con los patrones siguientes:*

Patrón	x_1	x_2
1	5,5	1,0
2	2,5	0,5
3	1,0	3,5

con μ y el radio para N_c, $\rho(N_c)$ iguales a

Pasada	μ	$\rho(N_c)$
1	0,8	1
2	0,5	0

6.3.1. Comentarios Generales

Podemos diferenciar dos maneras de utilizar los mapas de Kohonen. La primera es la utilización del Mapa de Kohonen como herramienta de análisis. Queremos encontrar relaciones entre nuestros patrones que no conocemos. Se aconseja para este caso utilizar mapas cuya dimensión objetivo sea pequeña, por ejemplo bidimensional. Esto nos permite graficar los datos que etiquetamos previamente con un nombre (¡no con una clase!). Luego, inspeccionando visualmente el mapa resultante podemos determinar las relaciones que existen entre ellos. En general, hay que tener cuidado con un aspecto. Si la dimensión intrínseca de los datos es superior al espacio objetivo, entonces nuestra proyección puede resultar imperfecta (Figura 6.9).

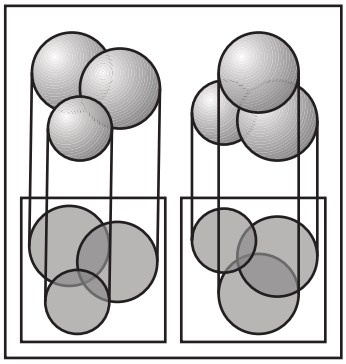

Figura 6.9: La dimensión del conjunto de datos es superior al del mapa, luego en la proyección pueden resultar traslapes de las categorías que además dependen del punto de vista desde el cual se están observando.

Esto no ocurre si la dimensión de los datos es igual o inferior. Si se construyen varios mapas a partir del mismo conjunto de datos, entonces al compararlos entre ellos podemos observar esencialmente el mismo mapa rotado, si la dimensión del mapa *coincide* con la de los datos. Si es inferior, se preservan las relaciones de vecindad de los datos proyectados (Figura 6.10).

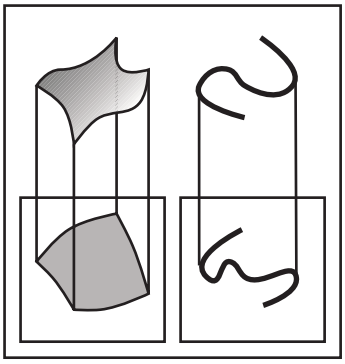

Figura 6.10: La dimensión del conjunto de datos es igual o inferior al del mapa, luego las proyecciones no presentan mayores problemas a la hora de interpretarlos.

La interpretación de los mapas resultantes procede de manera sencilla. Supongamos que queremos identificar relaciones entre diferentes bebidas. Entonces, creamos patrones cuyas componentes representan valores como precio, nivel de azucar, etc. Aplicando el entrenamiento sobre estos datos, las bebidas quedarán clasificadas según sus diferentes características. En el mapa se podrán identificar diferentes grupos de bebidas, cada uno constituído por una zona del mapa.

La segunda aplicación de los Mapas de Kohonen es el aprendizaje de una función de mapeo entre eventos que se observan en un medio ambiente a un dispositivo de control. Como ejemplo, pensemos en un brazo robótico que se mueve en un espacio de dos dimensiones con coordenadas x e y (Figura (6.11)). El brazo tiene dos motores

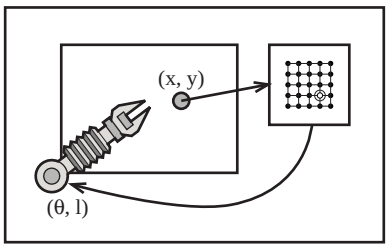

Figura 6.11: Un brazo robótico que calcula el ajuste de los motores a partir de un Mapa de Kohonen.

que lo controlan: uno que determina el ángulo θ y el otro el largo l de estiramiento del brazo. Se pretende saber cómo deben ajustarse ambos motores para que el brazo se localice en una posición deseada, sin realizar cálculos explícitos. Para esto, nuestro espacio de patrones es el espacio bidimensional de las coordenadas (x, y) y el objetivo aquel con coordenadas (θ, l). Este mapeo se obtiene entrenando la red neuronal con posiciones escogidas aleatoriamente $\vec{x}^p = (x^p, y^p)$. Como resultado obtenemos un mapa neuronal que nos permite realizarle pedidos como "aproxímate a la posición (2;3) en donde se ubica una ampolleta", excitando la neurona que ajusta a los motores adecuadamente. Esto lo hace sin que un experto haya realizado los cálculos y programado previamente.

Otras características que deben aclararse son las siguientes:

1. Como el aprendizaje se basa en un proceso estocástico, el mapeo depende sobre todo de un gran número de iteraciones. Como recomendación, se deben emplear por lo menos 500 iteraciones por cada neurona. El número de variables de entrada no influye en esta decisión.

2. Para aproximadamente 1000 pasadas, $\mu(t)$ debiera comenzar con valores altos próximos a 1, y luego decrecer paso a paso monotónicamente hasta un valor cercano a cero, por ejemplo $\mu(t^*) = 0{,}02$ el cual se mantiene constante durante el resto del algoritmo para forzar un esquema de entrenamiento global seguido de un ajuste fino.

3. La elección de una vecindad N_c es crítica para un buen desempeño, pues el mapa puede resultar no estar globalmente ordenado. Para evitarse problemas, se debe comenzar con un N_c amplio y decrecerlo a medida que avanza el algoritmo.

Capítulo 7

Redes de Hopfield

7.1. La Naturaleza de una Memoria Asociativa

Generalmente, "recordar" algo significa asociar una idea con un estímulo. Por ejemplo, alguien puede decir el nombre de una celebridad, y nosotros le asociaremos una cara y quizás algunas películas (en el caso de que se trate de un actor) en las que haya actuado. O tambien alguna imagen que vemos podemos asociarla a un recuerdo de la infancia. Estudios han mostrado que sobre todo el sentido del olfato opera de esta manera.

Un ejemplo más técnico sería por ejemplo el de reconocer dígitos (Figura 7.1). Se nos presenta un número que puede estar levemente distorsionado (con respecto a uno estereotípico como el que se aprende en el colegio), por ejemplo debido a la fuente que se utilizó al escribirlo en el computador, o la letra del escritor, si fue escrito a mano. Asimismo, si se nos muestra sólo una parte de la imagen, tambien somos capaces de reconstruir el número completo. En general, el problema de asociación puede resumirse como el de reconstruir un patrón cuando se dispone de él más un pequeño ruído que lo altera.

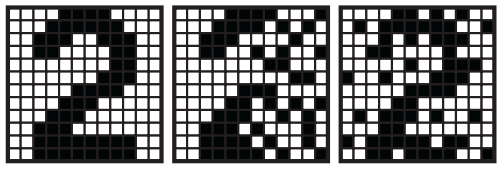

Figura 7.1: El dígito 2: a) Imagen original; b) corrupción de la imagen en la mitad derecha; c) corrupción leve en toda la imagen.

El paradigma subyacente puede ser descrito como sigue. Se dispone de un conjunto de datos dentro de los cuales existe un orden o un patrón que los interrelaciona, y que están almacenados en una memoria. Luego, este conjunto de datos interrelacionados puede ser visto como un sólo *patrón grabado* en memoria. En los ejemplos anteriores, estos corresponden al sector que alberga los datos sobre la celebridad o aquel de los recuerdos de la infancia. En el caso de los dígitos, el conjunto de imágenes que se parecen a aquella típica del dígito en cuestión. Cuando se es presentada una parte del patrón en forma de estímulo, el resto del patrón es recuperado y *asociado* a él.

Los computadores convencionales tambien pueden realizar este tipo de operación, pero en forma muy limitada. El software tradicional que

se emplea para este propósito es la base de datos. Ella puede almacenar un gran número de datos, y luego puede ser consultada. Sin embargo, una búsqueda no puede realizarse en forma arbitraria, pues la llave o palabra que se busca debe entregarse en forma exacta, es decir, sin distorsiones.

7.2. Analogías Físicas de la Memoria

Las redes neuronales que se emplean para modelar una memoria pueden verse como un ejemplo específico pertenecientes a una clase más amplia de sistemas físicos con un comportamiento análogo. Esto nos permite interpretar su forma de operar como un sistema dinámico, cuyo estado puede cuantificarse en términos de *energía*.

Pensemos en un sistema conformado por un balde y una pelota que se puede mover libremente dentro de ella, como ilustra la figura (7.2).

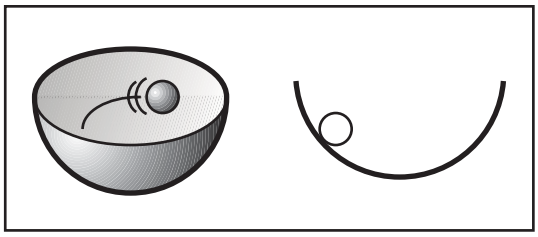

Figura 7.2: Una balde y una pelota, primero en 3D y luego la sección transversal en 2D

Supongamos que la pelota se libera desde un punto del borde del balde con un impulso inicial. Esta se movería de un lado hacia el otro, hasta detenerse en el fondo del balde.

Lo que ocurre puede describirse en forma sencilla considerando la energía del sistema. Inicialmente, la pelota parte con una *energía potencial*, que corresponde al esfuerzo que se requirió para levantarla hasta ese punto. Una vez que esta parte y acelera, gana en *energía cinética*. Eventualmente llega al estado de reposo en el fondo del balde que corresponde a aquel en que ambas energías, la potencial y la cinética, se han anulado. El punto es que a pesar de que la dinámica del sistema puede ser diferente en todos los experimentos, finalmente (debido a la pérdida de energía por la fricción y otros factores) converge siempre al mismo punto. Una vez que alcanza ese estado, no se sale de ahí. Es debido a esta razón que tal punto se denomina *punto estable* o *fijo*.

Existe otra forma de ver este mismo proceso, que es aquel que nos permite enlazar con nuestro concepto de memoria. Podemos pensar que la pelota siempre finaliza su movimiento en el mismo punto debido a que *recuerda* dónde se halla el fondo del balde. Podemos reforzar nuestra analogía otorgándole al sistema un sistema de coordenadas y asignándole a la pelota un vector posición $\vec{x} = (x, y, z)$. El fondo del balde tiene posición $\vec{x}_0$ y corresponde al patrón grabado. Luego, la posición de la pelota se puede reescribir como $\vec{x} = \vec{x}_0 + \Delta\vec{x}$, donde $\Delta\vec{x}$ corresponde al ruído o distorsión del patrón que contiene $\vec{x}$.

Ahora, si en vez de pensar en un balde con una sola depresión, lo reemplazamos por una superficie corrugada que almacena varios patrones, el asunto se vuelve más interesante.

Si hacemos partir la pelota sobre algún punto de la superficie, eventualmente se detendrá en una de sus depresiones, revocando al patrón almacenado. Nuevamente, cada uno de ellos corresponde a un mínimo del sistema.

Por ende tenemos dos maneras diferentes de ver lo que está sucediendo. La primera es pensar que el sistema converge hacia un mínimo; la segunda es que se revoca al patrón grabado

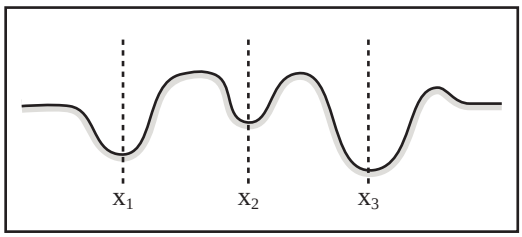

Figura 7.3: Una superficie con tres patrones almacenados.

más cercano al punto de partida. Si queremos construir una red que se comporta según estas ideas, debemos considerar los siguientes elementos claves:

1. El estado del sistema está completamente descrito por un vector $\vec{x}$.
2. Hay un conjunto de patrones $\vec{x}_1$, $\vec{x}_2$, ... que corresponden, en el caso de la superficie corrugada, a sus depresiones.
3. El sistema evoluciona en el tiempo, partiendo desde una posición arbitraria, a uno de sus mínimos, disminuyendo su energía en cada paso. Este proceso corresponde al de revocar un patrón grabado en la memoria.

7.3. La Red de Hopfield

Consideremos la red neuronal de tres unidades ilustrada en la figura (7.4).

Observemos que cada nodo está conectado con todos los demás, menos consigo mismo, y que los pesos que conectan a dos unidades, uno de ida y otro de vuelta, son simétricos. Es decir, dadas dos unidades distintas i y j, $w_{ji} = w_{ij}$ y $w_{ii} = 0$. Notemos que esta red, al contrario a las que hemos visto hasta ahora, no tiene un

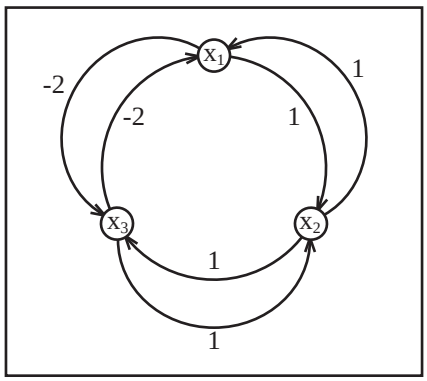

Figura 7.4: Una red de Hopfield.

flujo dirigido de información: no es de tipo *feed-forward*, ya que el valor que emite una neurona puede pasar por diferentes nodos hasta volver a la neurona original. En este caso, nos enfrentamos por primera vez a una red *recurrente*.

Fijemos los valores que pueden asumir las neuronas en 0 ó 1. El estado de la red viene dado entonces por el valor de estas tres variables x_1, x_2 y x_3. Supongamos que inicializamos a esta red con valores arbitrarios, y luego escogemos a un nodo al azar y lo actualizamos. La actualización ocurre de la siguiente manera.

Si queremos actualizar el nodo i-ésimo, calculamos

$$x_i = g(\sum_{j \neq i} w_{ij} x_j) \tag{7.1}$$

donde $g(\cdot)$ es la función de activación escalón, que vale 1 si su argumento es mayor igual a cero ó cero en caso contrario. w_{ij} denota al peso que *sale* de j y *entra* a i.

Desarrollemos un ejemplo para ilustrar esta idea.

Ejemplo 7 *Desarrollar un cuadro que muestre*

todas las posibles transiciones que pueden ocurrir en la red de Hopfield de la figura (7.4).

La tabla es muy sencilla. Primero, sabemos que con tres nodos podemos formar ocho combinaciones diferentes. A partir de estos ocho estados, podemos actualizar el nodo 1, 2 ó 3.

Estados				
Inicial		*Resultante*		
Número	$x_1x_2x_3$	*1*	*2*	*3*
1	*0 0 0*	*5*	*3*	*2*
2	*0 0 1*	*2*	*4*	*2*
3	*0 1 0*	*7*	*3*	*4*
4	*0 1 1*	*4*	*4*	*4*
5	*1 0 0*	*5*	*7*	*5*
6	*1 0 1*	*2*	*8*	*4*
7	*1 1 0*	*7*	*7*	*7*
8	*1 1 1*	*4*	*8*	*7*

Como ejemplo del cálculo que lleva a esta tabla, veamos el estado 1. Si a partir del estado 1, es decir, $x_1 = 0$, $x_2 = 0$ *y* $x_3 = 0$, *actualizamos el nodo 1, tenemos que su valor cambia a:*

$$\begin{aligned} x_1 &= g(w_{12}x_2 + w_{13}x_3) = g(1 \cdot 0 - 2 \cdot 0) \\ &= g(0) = 1 \end{aligned}$$

Luego la configuración resultante es $x = (1, 0, 0)$, *es decir, el estado 5. Actualizando el nodo 2, se tiene*

$$\begin{aligned} x_2 &= g(w_{21}x_1 + w_{23}x_3) = g(1 \cdot 0 + 1 \cdot 0) \\ &= g(0) = 1 \end{aligned}$$

Obteniendo así la configuración $x = (0, 1, 1)$, *el estado 3. Análogamente con el tercer nodo:*

$$\begin{aligned} x_3 &= g(w_{31}x_1 + w_{32}x_2) = g(-2 \cdot 0 + 1 \cdot 0) \\ &= g(0) = 1 \end{aligned}$$

llegando al estado 2 (es decir, $x = (0, 0, 1)$*).*

A partir de la tabla obtenida en el ejemplo, podemos trazar un diagrama de estados (Figura 7.5). Los arcos muestran las posibles transiciones que pueden ocurrir.

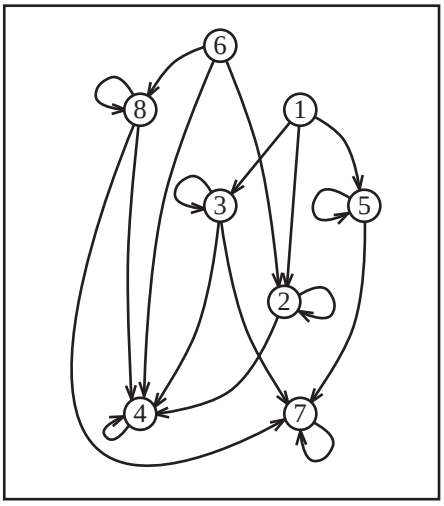

Figura 7.5: Diagrama de transición de estados de la red de Hopfield anterior.

Los estados están representados por círculos con su número asociado. Los arcos representan las posibles transiciones desde un estado hacia el otro. Están organizados de manera que reflejen su nivel de energía: los estados de más arriba tienen mayor energía que los de más abajo. A estas alturas, lo más importante es notar que, indepediente del punto de partida, siempre se llegra ó al estado 4 ó al 7. En otras palabras, una vez que la red se haya hecho un camino hacia uno de estos puntos, permanece allí para siempre. Estos dos corresponden a los puntos fijos del sistema, y representan a los patrones $(0, 1, 1)$ y $(1, 1, 0)$.

7.4. Energía de la Red

A modo de completitud, mencionaremos el término de Energía asociado a las redes de Hopfield, sin realizar algún desarrollo formal. Ésta está definida por

$$E = -\frac{1}{2}\sum_{i,j} w_{ji}x_i x_j \tag{7.2}$$

Se puede demostrar que esta cantidad decrece ante cualquier actualización de un nodo del sistema.

7.5. Actualización Asícrona versus Síncrona

Hasta ahora sólo hemos considerado la actualización del valor de una neurona a la vez. Cualquier nodo es un candidato a ser actualizado, lo que hace que operen en forma *asíncrona*; es decir, no existe una coordinación entre ellas en el tiempo. Pensemos ahora en el otro extremo, en la cual todas las neuronas son modificadas en el mismo instante. A esto se le denomina actualización *sincronizada*. Para operar de esta forma, hay que asegurarse de que los valores resultantes de cada neurona se calculen en forma separada, y recién una vez que se dispongan todos los resultados se modifica la red. Por ende se requiere disponer de dos vectores de estado: uno que almacene los valores antes y otro después de la actualización.

Al trabajar en forma sincronizada, el comportamiento de la red se vuelve *determinística*, el cual reduce las diferentes transiciones a partir de un estado a una sola, eliminando la naturaleza probabilística de la red. Este cambio altera la forma de los diagramas de transición.

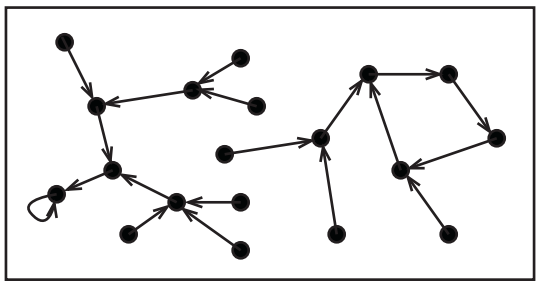

Figura 7.6: Diagrama de transición de una red de Hopfield sincronizada.

La primera diferencia resulta obvia. Desde cada estado sólamente emerge un arco, simplificando los diagramas de transición. La segunda diferencia es que ahora pueden existir *ciclos de múltiples estados*, las cuales pueden ser útiles para grabar secuencias de eventos o patrones. Es fácil verficar que los puntos fijos de las redes asíncronas siguen siéndolos en el caso sincronizado.

7.6. Buscando los Pesos

Hasta ahora hemos descrito la dinámica de la red de Hopfield, pero todavía no se ha mencionado nada sobre cómo hallar los pesos para resolver un problema en particular. De hecho, el mismo Hopfield, en su publicación original (1982) no entregó un método para entrenar la red, se limitó a dar una prescripción de cómo encontrar un conjunto adecuado de pesos, dado un set de patrones a ser grabados. A continuación se presenta una regla de aprendizaje inspirada *biológicamente*: la regla de Hebb.

7.6.1. Cómo grabar Patrones

La idea en la cual se basa la prescripción consiste en el deseo de capturar, en los pesos de la red, *correlaciones locales* entre las diferentes salidas de la red al encontrarse en un punto estable.

Consideremos dos nodos que en promedio, sobre el conjunto completo de patrones, tienden a tener el mismo valor. En otras palabras, ellas tienden a valer ambas 0 ó 1, formando las parejas (0,0) ó (1,1). La última pareja será mejor reforzada si el peso que los une tiene un valor positivo, ya que así ambas se van a contribuir mutuamente a producir la respuesta "1". Supongamos ahora que los nodos tienden a producir valores opuestos, es decir, (0,1) ó (1,0). Ambas configuraciones se refuerzan mediante un peso negativo en su interconexión, ya que así una neurona encendida tiende a apagar a la vecina. Notemos que la configuración (0,0) no es apoyada explícitamente, pero su aparición se ve favorecida si *no se aplica ninguna de las reglas anteriores.*

Matemáticamente podemos resumir las reglas anteriores en una ecuación. Primero, los patrones deben traducirse a unos que en lugar de tener 0 tengan -1. Es decir, el patrón (0, 1, 0) se convierte en (-1, 1, -1). Luego, se calcula cada uno de los pesos mediante la regla

$$w_{ij} = \sum_{p=1}^{P} v_i^p v_j^p \tag{7.3}$$

donde $\vec{v}^1, \vec{v}^2, \ldots \vec{v}^P$ son los patrones, y v_i^p denota a la i-ésima componente del p-ésimo patrón.

7.7. Comentarios Finales

Antes de finalizar este capítulo, es necesario mencionar algunas propiedades de las Redes de Hopfield.

7.7.1. La Regla de Hebb

El uso de una "receta" para fijar los valores de la red de Hopfield para grabar patrones pareciera ir en contra de la filosofía de las redes neuronales, pues recordemos que precisamente una de las gracias era que ellas aprendían a partir de ejemplos. Sin embargo, podemos ver la ecuación (7.3) como un atajo de un proceso adaptivo que tendría lugar si empleáramos el siguiente algoritmo de aprendizaje:

1. Presentarle un patrón a la red neuronal, es decir, hacer $x_i = v_i$ para todas todos los nodos x_i de la red y componentes v_i del patrón.

2. Si dos nodos tienen el mismo valor, entonces incrementar levemente el peso que los conecta. Si tienen valores contrarios, decrementar levemente.

Estos pasos se repiten contínuamente hasta que la red haya grabado los patrones. La regla en el segundo paso del algoritmo corresponde a

$$w_{ij} \leftarrow w_{ij} + \mu v_i v_j$$

donde μ es el factor de aprendizaje que tiene valores entre 0 y 1. Resulta evidente que la ecuación (7.3) es simplemente la acumulación de repetidos pasos del algoritmo anterior. La regla mencionada corresponde a la más sencilla perteneciente a una gran familia de reglas de aprendizaje llamadas *Reglas de Hebb* debido a D. O. Hebb. En su libro *The Organization of Behaviour* (1949) postuló que

> Cuando un axón de una célula A está lo suficientemente cerca para poder exitar a una célula B, y ella participa en forma repetida o persistente en la activación

de la segunda, entonces en una o en ambas neuronas ocurre un proceso de crecimiento o una alteración metabólica de manera que la eficiencia de A para activar a B se incrementa.

7.7.2. Capacidad de Almacenamiento

¿Qué tan buena es la receta para almacenar patrones dada por la ecuación (7.3)? Claramente, a medida que aumenta el número de patrones a almacenar, la precisión del grabado debe disminuir. En un trabajo empírico de la publicación de 1982, Hopfield mostró que aproximadamente la mitad de los patrones se guarda correctamente en una red de $N = 10m$ unidades. Los patrones restantes no quedaban bien almacenados. Un análisis más riguroso lo realizó McCliece et al. (1987) y mostró en forma teórica que el número máximo de patrones que una red de N nodos puede almacenar es

$$\frac{N}{2}\log N$$

Para $N = 100$ esto nos da $m = 11$ patrones.

Además, ocurren otros inconvenientes. La receta de aprendizaje no hace distinción entre un patrón y su negativo, y es por esta razón que esta misma regla almacena al patrón y a su inverso simultáneamente (Figura 7.7).

Además, si la red *parte* en un estado que corresponde a un máximo de la energía, la red tampoco evoluciona y permanece en ese mismo estado, que a pesar de ser un punto crítico del sistema, es inestable.

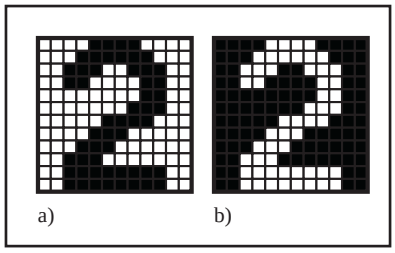

Figura 7.7: La regla de almacenamiento graba simultáneamente al patrón deseado y a su inverso.

Capítulo 8

Algoritmos Genéticos

8.1. Introducción

¿Cómo evolucionan los seres vivos? Ésta es la pregunta que se hacía *John Holland* en su infancia, cuando notaba que la naturaleza creaba seres cada vez más perfectos, en el sentido que existe una tendencia a la *adaptación* al medio ambiente. Lo curioso es que todo el proceso evolutivo se basa en pequeñas *interacciones locales* entre individuos y los elementos que los rodean.

Para responderse esta pregunta, planificaba capturar los elementos y mecanismos cruciales para poder reproducir la estrategia de la naturaleza y observar su desempeño. En los años 60, cuando impartía el curso titulado *Teoría de Sistemas Adaptivos*, junto con sus alumnos, comenzaron a desarrollar las ideas que más tarde darían orígen a los *Algoritmos Genéticos*.

Los Algoritmos Genéticos conformaron la semilla de una un conjunto de técnicas englobadas bajo el nombre de *Computación Evolucionaria*. El objetivo de estas consiste en:

1. Optimizar una función objetivo que puede ser no-lineal, o que se conozca sólo parcialmente.
2. Imitar los procesos adaptivos de los sistemas naturales, y diseñar sistemas artificiales que retengan los mecanismos importantes de los procesos naturales.

Inicialmente, las ideas de Holland no tuvieron gran aceptación. Esto se debía principalmente a los elevados costos computacionales que implica la optimización con estos métodos. Sin embargo, a mediados de los años '80, con la aparición de los computadores de altas prestaciones y bajo costo, cambió completamente el panorama. Las técnicas de la Computación Evolucionaria fueron capaces de resolver algunos problemas de la ingeniería que antes eran dificilmente resueltos vía técnicas de optimización tradicional. En algunos casos, no exisitían soluciones en absoluto. Desde ese entonces, el desarrollo en el área ha sido espectacular.

8.2. Ideas Centrales

Para ilustrar las ideas centrales de la Computación Evolucionaria, pensemos en la naturaleza como un actor que tiene un objetivo: construir seres vivos que sobrevivan por más tiempo en su medio ambiente.

Entonces, la estrategia central de la naturaleza consiste en los siguientes pasos:

1. Proponer soluciones potenciales: Crea un conjunto de invididuos, cada uno con características diferentes.

2. Evaluar la calidad de cada una de las soluciones: Se verifica el nivel de adaptación de los individuos a su medio ambiente.

3. Recombinación de las soluciones: Se seleccionan los individuos mejor adaptados y se cruzan para dar orígen a una nueva generación.

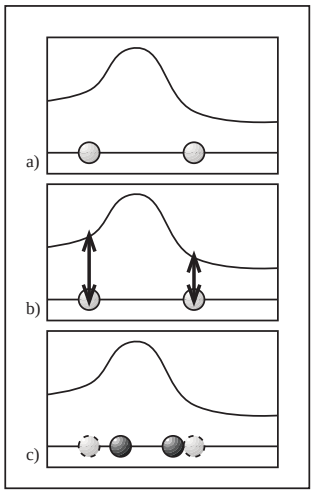

Figura 8.1: Estrategia de la selección natural.

La estrategia anterior se conoce bajo el nombre *selección natural*, y ella es la responsable del proceso de optimización al que estan sometidos los seres vivos.

8.3. Características Principales

Los Algoritmos Genéticos exhiben algunas propiedades importantes que las distinguen de los métodos clásicos de optimización, las cuales eventualmente pueden ser decisivas a la toma de tener que escoger entre ellas:

1. No debe conocerse el gradiente de la función objetivo: Los métodos clásicos requieren disponer de una expresión para gradiente. Sin embargo, esto no siempre es posible, sobre todo cuando se trata de funciones muy complicadas.

2. La optimización es de naturaleza estocástica: Cuando se dispone de un algoritmo fijo para la optimización, el espacio de búsqueda suele ser muy limitado, ya que los caminos que se exploran suelen ser sencillos, sin abarcar grandes regiones. El algoritmo genético sin embargo procede en forma cuasi-aleatoria y puede por ende llegar a regiones que no se habían contemplado.

3. Es un algoritmo independiente del problema, lo cual lo hace un algoritmo *robusto*, por ser útil para cualquier problema, pero a la vez *débil*, pues no está especializado en ninguno.

4. Su implementación es sencilla, en contraposición a los algoritmos de optimización específicos que requieren estructuras de datos avanzadas y un fuerte bagaje matemático.

5. Su ejecución generalmente requiere la utilización extensa de recursos computacionales. Esta es una desventaja de los algoritmos genéticos. El proceso de optimización generalmente necesita mucho espacio en memoria (principal) y tiempo de ejecución debido al alto grado de paralelismo con que opera la búsqueda.

8.4. El Algoritmo Genético Simple

El *Algoritmo Genético Simple* es el primer Algoritmo Genético que estudiaremos. Es el más sencillo, pero tiene desventajas que revisaremos más adelante. Los elementos básicos son:

1. *Población*: Consiste en el conjunto de *individuos* que exploran el espacio de búsqueda.
2. *Individuo ó Cromosoma*: Cada uno constituye una *solución potencial* del problema a optimizar. Se representan mediante *tiras genéticas* que codifican el conjunto de parámetros de la función objetivo. Generalmente, son tiras binarias, es decir, un vector de *ceros* y *unos*.
3. *Función de Adaptación ó Fitness*: La función objetivo del problema. Corresponde a una función $f(x_1, x_2, \ldots, x_N)$ con resultados reales, donde las variables x_j con $j = 1, \ldots, N$ son los parámetros codificados por un individuo.
4. *Operadores Genéticos*: Corresponden a los operadores que se efectúan sobre la población entera, y los cuales conforman el proceso de selección natural. Estos pueden desglosarse en:
   a) Selección
   b) Muestreo
   c) Cruce
   d) Mutación

Un detalle que debe mencionarse es el uso de los índices. La notación que emplearemos de ahora en adelante es que se utilizará j para denotar las componentes e i para identificar a los individuos. Luego, $x_j i$ es la j-ésima componente del i-ésimo individuo.

8.4.1. Codificación de las Variables

Como se ha mencionado, los individuos consisten en tiras genéticas que codifican un set de variables. Hay que tener clara la diferencia entre la *representación* y la *interpretación* de un individuo. Revisemos estos dos aspectos:

1. *Genotipo*: Corresponde a la tira genética c en sí con componentes c_j de índice $j = 1, \ldots, M$ llamados *genes*. Estos genes pueden asumir valores provenientes de un *alfabeto* (conjunto finito de símbolos ó números) a elección, como por ejemplo $\{0, 1\}$ (alfabeto binario). Algunos posibles cromosomas binarios de largo 6 son

   $$011010 \quad 010010 \quad 101101 \quad 100110$$

2. *Fenotipo*: El genotipo codifica a un conjunto de variables x_j con $j = 1, \ldots, N$ que constituyen a los parámetros de la función de adaptación. Se emplea el término *fenotipo* para denotar a estos últimos. Intuitivamente, la diferencia entre genotipo y fenotipo radica en que el primero corresponde a la codificación y el segundo a la manifestación de las propiedades genéticas de un individuo. Siguiendo con el ejemplo anterior, un cromosoma de largo 6 podría contener dos variables x_1 y x_2 provenientes de $\mathbb{N}$, codificadas en binario: los primeros 3 bits corresponden a x_1 y los restantes a x_2. Si el cromosoma es 001111 (el genotipo), entonces su fenotipo es

   $$x_1 = 2^3 \cdot 0 + 2^2 \cdot 0 + 2^1 \cdot 1 = 1$$

$$x_2 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 1 = 7$$

El proceso de pasar del fenotipo x al genotipo c se llama *codificación*, y el inverso *decodificación*. Esto se escribe como

$$\text{codificación:} \quad x = x(c)$$

$$\text{decodificación:} \quad c = c(x)$$

Para insistir que se trata de relaciónes *uno a uno* ó biyectivas.

La manera en que ésto se haga depende de cada problema. Es decir:

- el número de variables,
- la precisión de éstas
- y su dispoción dentro del cromosoma,

vienen dados por la naturaleza del problema y la exactitud deseada del resultado.

En cuanto al tamaño del alfabeto a utilizar, existe un teorema proveniente de la *teoría de los esquemas* de John Holland que afirma que el alfabeto óptimo, en cuanto al largo de los strings versus la riqueza de expresión, tiene cardinalidad 2. Es por esta razón que en lo que sigue emplearemos sólo *cromosomas binarios*. Tal como ya lo hemos mencionado anteriormente, los elementos de un cromosoma binario pueden verse como *bits* para establecer una analogía con la terminología empleada en la computación.

8.4.2. Anatomía del Algoritmo

La mejor manera de aprender los Algoritmos Genéticos es viendo un ejemplo introductorio en donde aplicaremos cada uno de los pasos necesarios.

Imaginémonos una población de cuatro individuos, representados cada uno por medio de un *string* o cromosoma de cuatro bits. Sea nuestra función objetivo

$$f(x) = 2x.$$

El objetivo consiste en *maximizar* esta función sobre el dominio $x \in \{0, 1, 2, \ldots, 31\}$. Ahora pensemos en cuatro cromosomas que han sido generados aleatoriamente antes de la ejecución del Algoritmo Genético, conformando nuestra población inicial. Los valores del fitness correspondientes provienen de la función objetivo $f(x)$ (Cuadro 8.1).

Los c_i son los cromosomas, los $x_i = x(c_i)$ los valores de los cromosomas decodificados y los $f_i = f(x_i)$, para simplificar la notación. Los valores en la columna $f_i / \sum_j f_j$ entregan la probabilidad de selección de cada cromosoma. Así, inicialmente 11000 tiene un 38.1 % de probabilidad de ser seleccionado, 00101 tiene un 7.9 % de probabilidad, etc. Los resultados del muestreo (el proceso de selección real) están dados por la columna ♯ *Real de Copias*. Como esperado, estos valores se parecen mucho a aquellos dados por la columna de ♯ *Esperado de Copias*.

Después de seleccionar a los individuos, el Algoritmo Genético los agrupa de a pares aleatoriamente. Para cada pareja (por ejemplo, $A =$ 11000 y $B =$ 10110), el Algoritmo Genético decide si debe ó no realizar un cruce. Si no lo hace, entonces ambos cromosomas pasan a formar parte de la nueva población. Si lo hace, entonces se escoge un *punto de cruce* aleatorio dentro del cromosoma y luego se procede de la siguiente manera. Supongamos que el punto escogido se encuentra en la segunda posición, es decir, $A = 11|000$ y $B = 10|110$. Entonces los cromosomas se cruzan, obteniéndose los hijos $A' = 11110$ y $B' = 10000$. Cada hijo se compone de un segmento proviente de cada padre. Estos hijos se

Cuadro 8.1: Cuatro Cromosomas y sus Valores de Fitness

i	String c_i	Fitness $f(x_i) = 2x_i$	$f_i / \sum_j f_j$	♯ Esperado de Copias $f_i/\overline{f}$	♯ Real de Copias
1	11000	48	0.381	1.524	2
2	00101	10	0.079	0.317	0
3	10110	44	0.349	1.397	1
4	01100	24	0.191	0.762	1
	Suma	126	1.000	4.000	4
	Promedio	31.5	0.250	1.000	1
	Máximo	48	0.381	1.524	2

sitúan después en la nueva población, formando parte de la nueva generación.

Ahora, el Algoritmo Genético invoca al operador de mutación sobre cada gen de toda la población, cambiándolo con una probabilidad muy baja (usualmente del orden de $p \leq 0{,}01$).

Después de aplicar sobre la población actual los operadores de selección, cruce y mutación, estos cromosomas resultantes pasan a ser la población que representa a la nueva generación, como indica el cuadro (8.2). En aquella del ejemplo, el fitness promedio de la población aumentó aproximadamente en un 30 por ciento y el fitness máximo en 25 %. Este sencillo proceso se repite durante varias generaciones hasta alcanzar un criterio de parada.

Enunciamos el Algoritmo Genético Simple para poder discutir parte por parte su forma de operar.

Algoritmo Genético Simple

1. Crear una población inicial aleatoria $P = \{c_i\}$ de cromosomas c_i, $i = 1, \ldots, |P|$.
2. Para cada cromosoma $c \in P$ evaluar el fitness $f(x)$.
3. Mientras no se cumpla el criterio de parada:
   - *a*) Aplicar *selección* sobre la población P.
   - *b*) Reemplazar P por una nueva población P obtenida mediante *muestreo*.
   - *c*) Aplicar el *cruce* sobre P.
   - *d*) Aplicar la *mutación* sobre P.
   - *e*) Evaluar la población P.
4. Retornar P

donde los operadores genéticos están escritos en itálica para destacarlos.

En el fondo, el Algoritmo Genético Simple consiste en emplear la estrategia o esquema anterior. La gracia consiste en que, dependiendo del tipo de optimización que se desea realizar, se pueden emplear unos u otros operadores genéticos más apropiados. Para esto, se

Cuadro 8.2: Población después de Aplicar Selección, Muestreo y Cruce

Tras la Reproducción	Pareja	Punto de Cruce	Después de Cruce	Fitness $f(x_i) = 2x_i$
11|000	x_3	2	11110	60
1|1000	x_4	1	11100	56
10|110	x_1	2	10000	32
0|1100	x_2	1	01000	16
Suma				154
Promedio				41
Máximo				60

tiene un catálogo muy grande de operadores estándar, de los cuales a continuación sólo mencionaremos los más comunes.

8.4.3. Selección

Igual que en la naturaleza, son los mejores individuos de una población los que tienen mayor probabilidad de generar descendencia. Así se asegura de que mejoren las raza de generación en generación. En un algoritmo genético, este proceso de *calificación* se realiza mediante dos operaciones: *selección*, que veremos en este apartado; y *muestreo* [1].

Para generar una nueva población, debe determinarse primero cuánta descendencia va a tener cada individuo. Esto se realiza asignándole un número *esperado de copias* a cada inviduo basado en el fitness. Esto es precisamente lo que hace el operador de selección σ. En la práctica, este es un operador que *modifica al fitness* de cada individuo según ciertos criterios, preparándolos para el proceso de muestreo.

Existen muchos operadores de selección distintos. Cada uno está diseñado para modificar el fitness de una manera diferente. No es necesario que se aplique uno sólo a la vez, pueden aplicarse varios de ellos consecutivamente para combinar los efectos. Sin embargo, hay que tener en cuenta que los operadores en general no son conmutativos, y por ende sí importa el orden en el cual se aplican.

Veamos algunos de los operadores de selección.

Selección Proporcional

Es el operador de selección más frecuente. Lo único que hace es asignarle a cada individuo un número de copias esperado proporcional a su fitness. Es decir, un individuo de fitness 30 debería tener tres veces más descendencia que aquel con fitness 10. La selección proporcional se define por

$$\sigma_k = \frac{f_k}{\overline{f}} \tag{8.1}$$

donde f_k es el fitness del individuo y $\overline{f}$ el promedio sobre toda la población.

Hay que tener en cuenta que este operador solamente sirve si la función de adaptación es may-

[1] En la literatura, las operaciones de *selección* y *muestreo* se tratan como un solo operador, que se denomina *selection operator*. Sin embargo, esto no es del todo correcto y genera confusiones que evitaremos en este texto.

or que cero y si se trata de una maximización.

Transformación Lineal

A veces, la selección proporcional por sí sola no sirve. Esto se puede deber

- Se tiene un problema de *minimización*.
- Se desea *atenuar* ó *pronunciar* la curva definida por la función de adaptación.
- Los valores del fitness deben ser convertidos a estrictamente positivos para luego aplicarles la selección proporcional.

Como puede intuirse a partir de las razones mencionadas, generalmente la transformación lineal sirve como un *preprocesamiento* del fitness para luego aplicarle la selección proporcional.

La transformación lineal está definida por

$$\sigma_k = af_k + b \tag{8.2}$$

donde a y b son constantes. Veamos cómo fijar estos valores para resolver los problemas anteriormente mencionados.

1. Si el problema es de minimización, entonces deben convertirse los fitness pequeños a grandes y viceversa. Entonces, una solución puede ser

   $$\sigma_k = -f_k + (b - \min_i f_i)$$

   donde $b \geq 0$.

2. Recordemos que en la selección proporcional, el número esperado de copias que recibe un individuo es proporcional a su fitness. Si la curva de fitness tiene *peaks* muy pronunciados, puede ser que un individuo de un máximo local se tome a toda la población, eliminando la riqueza genética. Por el otro lado, si la curva no es demasiado plana, entonces el número esperado de copias que recibe un individuo en un óptimo prácticamente no se diferencia de los demás.

   Para atenuar, debe usarse $0 < a < 1$, y para pronunciar, $a > 1$.

3. Para convertir los fitness a estrictamente positivos, debe escogerse un b lo suficiente grande para que todos los fitness resulten positivos.

Selección por Ranking

En el caso de la selección proporcional, el número de descendientes que se le otorga al mejor individuo crece exponencialmente, y puede llegar a llenar rápidamente a la población, cosa que puede llevar a una convergencia prematura. A veces uno quisiera limitar el número esperado de copias que reciben los individuos de la población dentro de ciertos márgenes, para asegurar la mantención de un nivel de diversidad genética.

Para lograr este objetivo, es necesario independizarse de las proporciones presentes en los fitness de los individuos, manteniendo nada más que el orden.

Una forma de hacer esto es creando un *ranking* de los individuos, del peor (el nmero $k = 0$) al mejor ($k = n$), y calculando su número esperado de copias mediante la fórmula

$$\sigma_k = \frac{2\delta}{n-1}k + (1 - \delta)$$

Donde $0 < \delta \leq 1$ detemina la desviación del número esperado de copias del caso equitativo, en el cual todos los individuos obtienen un sólo descendiente.

No es difícil ver que esta fórmula corresponde simplemente a la ecuación de la recta que le asigna al primer individuo (el peor) $(1 - \delta)$ y al último (mejor) $(1 + \delta)$. Luego, si $\delta \approx 0$ entonces $\sigma_0 \approx 1$ y $\sigma_n \approx 1$, y en el caso extremo con $\delta = 1$ tenemos $\sigma_0 = 0$ y $\sigma_n = 2$.

8.4.4. Muestreo

Una vez que se le hayan aplicado todos los modificadores necesarios al fitness, los cuales en parte reflejan el número esperado de copias que recibirá cada individuo, hay que proceder a efectuar el muestreo. Este proceso consiste en general en seleccionar aleatoriamente a los individuos de la población antigua para llenar a otra nueva (la descendencia) del mismo tamaño. La distribución de probabilidad con la cual se seleccionan a los individuos se modela usando los números esperados de copias obtenidos en el paso anterior.

Existen diversos tipos de muestreo, algunos mejores que otros en relación a los siguientes criterios:

- Sesgo: Existe si el número esperado de copias de un individuo obtenido en el proceso de selección es diferente a aquel dado por el muestreo. Es decir, puede ser que un individuo tenga un número de copias igual a 2.6, pero el muestreo está hecho de tal manera que a lo más saque 2 copias.
- Dispersión: A veces, a pesar de que el muestreo asegure sesgo cero, puede ser que el *rango* del número de copias se extienda sobre un conjunto muy grande. Por ejemplo, si el número esperado es 3.5 copias, el método de la *ruleta* efectivamente garantiza ese número esperado, pero tambien puede ocurrir que no saque 0, 1, 2, ..., hasta n copias, y por ende la dispersión del método es máxima.
- Complejidad: Un criterio muy importante a considerar es el nivel de recursos computacionales que exige el método, medidos en espacio (RAM y disco duro) y tiempo. Un método puede ser muy eficiente, pero demorarse un tiempo tan largo que no vale la pena tomarlo en consideración, ya que si no los resultados tardarían semanas en ser calculados. Este valor se expresa en términos del número de entradas, y la notación es asintótica. Por ejemplo, $t = O(n^3)$ significa que el tiempo empleado crece cúbicamente con el número de individuos.

Veamos ahora los diferentes tipos de muestreo.

Ruleta

El nombre de este método resulta muy descriptivo. Básicamente, el muestre consiste en determinar uno tras otro a los individuos seleccionados simulando un juego de la ruleta. Es decir, se utilizan los valores del fitness para dividir una torta en trozos proporcionales al fitness, y luego se determina al azar en qué sector cae la bolita.

Este método es el más utilizado y el original propuesto por Holland. Sin embargo, no está excento de problemas. Como los individuos seleccionados se determinan uno tras otro de manera independiente, en un caso extremo un individuo puede llegar a llenar a la población descendiente con sus copias.

- Sesgo: 0
- Dispersión: máxima
- Complejidad: $t = O(n^2)$

SUS (Stochastic Universal Sampling)

Después de hallar los defectos relacionados con el método de la ruleta, Holland propuso una alternativa mejorada. Consiste en una ruleta con n punteros equiespaciados que se giran a la vez. Se realiza una sóla jugada, la cual determina simultáneamente a todos los individuos copiados.

- Sesgo: 0
- Dispersión: 0
- Complejidad: $t = O(n)$

Torneo (Tournament)

Inspirada en torneos, éste método consiste en

1. Repetir q veces los siguientes pasos:
2. Mezclar la población.
3. Formar grupos de a q individuos.
4. Determinar el ganador de cada grupo, *sin retirarlo de la población*. Cada uno recibe una copia dentro de la nueva población.

- Sesgo: 0
- Dispersión: limitada en q
- Complejidad: $t = O(n)$

8.4.5. Cruce

Tras la etapa de muestreo de la población, se dispone de una nueva población tentativa Q la cual se utiliza en esta etapa de cruce para generar la descendencia. Este proceso consiste en mezclar aleatoriamente la población Q y luego formar parejas de individuos, los cuales darán orígen a 2 *hijos*.

En algunos casos, la elección correcta de un método de cruce puede marcar la diferencia entre el éxito y el fracaso de una optimización, ya que un operador bien escogido aprovechará la lógica existente en la codificación de los cromosomas para generar descendencia que tiene sentido. A veces existen restricciones en el problema en sí, los cuales idealmente deben ser preservados en el cruce.

Técnicamente, un cruce se define a partir de una *máscara* de 0's y 1's. Para cada pareja, se genera una máscara usando algún criterio preestablecido. Luego, el primero hijo se forma tomando algunos genes, aquellos marcados con 0 en la máscara, del primer padre y los otros, los marcados con 1, del segundo padre. El segundo hijo se construye de la manera inversa. Por ejemplo, sean los padres $p_1 = 11110000$ y $p_2 = 01010101$ y la máscara $m = 00110011$. Los hijos resultantes serían $h_1 = 11|01|00|01$ y $h_2 = 01|11|01|00$.

Cruce de un punto

El cruce original propuesto por Holland. Consiste en determinar aleatoriamente un punto de cruce. Los bits a la izquierda de la máscara se marcan con 0, los otros con 1.

Si bien ésta manera de generar la máscara parece plausible, hay que destacar que está sesgada, y tiende a proteger los bordes de los cromosomas en detrimento de los centros.

Cruce de dos puntos

Una primera propuesta para arreglar el sesgo en que incurre el cruce de un punto es con del cruce de dos puntos. Imaginemos una tira genética en la cual unimos los extremos, formando un anillo. Queremos que el cruce consista en

aislar un trozo de la cinta, el cual se lo asignamos al primer padre, y el resto al segundo. Esto es equivalente a determinar 2 puntos aleatoriamente en la máscara y marcar a todas las posiciones entre ambos puntos con 1's y las demás con 0's.

Una de las ventajas de este método es que se independiza de la posición de los genes, en contraposición al cruce de un punto. Además, tiende a preservar la estructura general de los cromosomas, ya que estos sólo se rompen en los puntos de cruce.

Cruce de n-puntos

Si bien parece tentador generalizar el método de los n-puntos, no tiene mayores efectos en los resultados.

Cruce uniforme

Si llevamos la idea de los n-puntos de cruce al extremo, llegamos al cruce uniforme. En este, cada bit de la máscara se determina lanzando un moneda al aire. Este cruce tiene la gran ventaja de que es completamente insesgado, trata a todas las estructuras que pudiesen aparecer en las tiras genéticas de la misma manera. Generalmente, ésta es la mejor opción cuando no se dispone absolutamente ningún *conocimiento a priori* de la forma de los resultados.

Sin embargo, no se emplea ninguna política de conservación de estructuras genéticas, precisamente debido a la ausencia de sesgo. Luego, muchos se abstienen de su utilización pues la consideran "destructiva".

Cruce balanceado

Este método es muy similar al cruce de dos puntos. También se visualiza a la máscara como una cinta circular, pero ésta se subdivide en dos trozos del mismo largo, logrando así que los hijos se compongan de una mitad de un padre y la otra del segundo padre.

En la práctica, la máscara se genera partiendo con una que se encuentra inicialmente en blanco, con todas sus posiciones en 0, y luego se determina un punto de cruce al azar, marcando, desde ahí en adelante, la mitad de las posiciones con 1's. Si durante este proceso se sale del borde derecho, se parte nuevamente desde la izquierda.

8.4.6. Mutación

Supongamos que después de un número determinado de iteraciones del algoritmo genético, todos los individuos de la población tienen la m-ésima posición en 0. Utilizando selección, muestreo y cruce nada más no hay forma de recuperar el bit 1 en la posición j, ya que estos operadores anteriores trabajan sólo reciclando (recombinando) los genes existentes. Sin embargo, ese gen perdido podía corresponder a uno clave en la solución de la optimización. Para asegurar una diversidad genética, se introduce la mutación. En la naturaleza, esta biodiversidad se hace presente mayoritariamente a través de los *rayos cósmicos*, de los cuales se cree que han sido los impulsores de la evolución.

Existe una serie de diferentes tipos de mutaciones, casi en su totalidad empleados en problemas con restricción, ya que todos se basan principalmente en la *mutación clásica*, la única que mencionaremos en este tutorial.

Mutación Clásica

Consiste en cambiar aleatoriamente un bit del cromosoma. En realidad, existe una probabilidad, generalmente muy baja, de cambiar cada

bit de la población. En la práctica, se realizan diferentes aproximaciones para producir la mutación debido al alto costo que puede implicar la generación de un número aleatorio para cada bit presente en la población.

8.5. Aleatoriedad

¿Porqué funcionan los algoritmos genéticos? Más bien parece un proceso netamente estocástico sin rumbo definido. ¿Porqué no simplemente utilizar un algoritmo clásico para la optimización?

Un proceso de optimización consiste básicamente en una búsqueda en un espacio de soluciones potenciales. Supongamos que disponemos de un robot al cual le podemos dar instrucciones (en forma de un programa computacional) para que busque en este espacio una solución óptima. Si la estructura de las soluciones tiene una forma conocida y sencilla de calcular, uno podría pedirle al robot que busque en tales regiones para acelerar el proceso de exploración. En este caso, nuestro robot viene haciendo el papel de un ente *no creativo*, pues simplemente sigue las instrucciones que nosotros le hemos indicado en nuestro programa.

Sin embargo, ¿cómo hacemos para que nuestro robot manifieste un cierto nivel de creatividad? Además, ¿qué es la creatividad? No existe consenso en cuanto a esta última pregunta, pero podemos hacer un primer intento por entender este concepto mediante un ejemplo clásico (y por cierto muy famoso).

Imaginémonos un mono, digamos un chimpancé, al que le entregamos una máquina de escribir con infinita tinta y papel. Este mono toma la máquina y se larga a escribir. El mono, si bien tiene un cierto nivel de conciencia e inteligencia, podemos ver las secuencias de letras, puntuaciones y espacios en blanco como secuencias netamente aleatorias, y por ende carentes de sentido. Ahora bien, si esperamos suficiente tiempo a su lado, leyendo los papel que éste produce, ¡veremos que en algún instante escribirá la obra completa de Shakespeare! Un simple cálculo usando la teoría de probabilidades afirma este hecho.

Si bien este experimento puede parecer descabellado en principio, lo que se pretende decir es que la creatividad puede ser vista como una recombinación de elementos de una manera novedosa, inesperada, es decir, aleatoria. Existen corrientes filosóficas que efectivamente creen que la creatividad pura consiste en la aleatoriedad.

Aterrizando esta idea en términos de algoritmos genéticos, la aplicación de aleatoriedad puede resultar especialemente fructífera en aquellos problemas en donde no se conocen las regiones que contienen las soluciones "buenas", y en donde es necesario que se improvisen exploraciones en direcciones no sospechadas. Para nuestro robot, esto significa que no le estamos dando un rumbo fijo en su búsqueda, sino que le estamos dando la opción de realizar caminatas hacia lugares no considerados.

Sin embargo, es evidente que tal lujo implica un costo muy elevado en términos de tiempo, pues si bien nos aseguramos que la búsqueda resulte exitosa, nos arriesgamos a que ésta demore demasiado tiempo. El mecanismo de exploración empleado por la naturaleza se acelera principalmente debido a la introducción de *paralelismo* y de la *guía*, dada en nuestro caso por el uso de muchos robots que interactúan entre sí para intercambiar información sobre las regiones prometedoras.

8.6. Exploración versus Explotación

Siguiendo la analogía con los robots, pensemos que estos se están empleando para buscar oro. Inicialmente, disponemos los robots en forma totalmente aleatoria para que busquen al mineral preciado.

Sin embargo, al aparecer los primeros hallazgos, ¿cómo distribuímos nuestros recursos (los robots) en forma óptima? Esta no es una pregunta sencilla de responder, pues hay diferentes aspectos que hay que considerar. Por un lado, podemos exigir que todos los robots sigan buscando en toda la región, sin importar que algún robot en particular haya dado con oro. A esto se le llama *exploración*. Sin embargo, podríamos haber redistribuído nuestros robots de manera que muchos de ellos sigan buscando en las proximidades inmediatas de un hallazgo prometedor. Esto corresponde a la *explotación*.

Un buen algoritmo genético debe lidiar con este problema, manteniendo un equilibrio entre exploración y explotación. Hasta el día de hoy, este es un problema abierto, pero existen aproximaciones para el caso de n-agentes con un n pequeño. En la literatura, este problema se denomina *el problema del tragamonedas de n palancas* ("the n-handed bandit problem"), que modela la disyuntiva entre jugar una u otra palanca con diversos premios y probabilidades de ganar.

Los resultados actuales han mostrado que una estrategia prometedora parece ser la de asignar un número de recursos que crece exponencialmente a aquellas regiones que dan más ganancias.

Capítulo 9

Bibliografía

1. Christopher M. Bishop (1997). Neural Networks for Pattern Recognition. *Oxford University Press.*

2. Stuart Russell, Peter Norvig (1995). Artifitial Intelligence: A Modern Approach. *Prentice Hall.*

3. Sergios Theodoridis, Konstantinos Koutroumbas (1999). Pattern Recognition. *Academic Press.*

4. Pablo Estévez (2001). EM753 - Apuntes de Teoría de Redes Neuronales. *Universidad de Chile.* `http://cipres.cec.uchile.cl/~em753`

5. Pablo Estévez (2002). EM754 - Apuntes de Computación Evolucionaria. *Universidad de Chile.* `http://cipres.cec.uchile.cl/~em754`