



UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA

APRENDIZAJE NO SUPERVISADO
Y EL ALGORITMO WAKE-SLEEP
EN REDES NEURONALES

TESIS
PARA OBTENER EL TÍTULO DE:
LICENCIADO EN MATEMÁTICAS APLICADAS

PRESENTA:

NORMA PELÁEZ CHÁVEZ

DIRECTOR DE TESIS:

M.C. VERÓNICA BORJA MACÍAS

HUAJUAPAN DE LEÓN, OAXACA.
NOVIEMBRE DE 2012

*Dedicado
a mis queridos padres Mario y Magdalena,
a mis hijos Job y Noemí,
y a mi esposo Alejandro.*

Agradecimientos

A mi directora de tesis M.C. Verónica Borja Macías por su paciencia, dedicación, por las aportaciones en la realización de este trabajo y por ser ejemplo de fortaleza. Asimismo quiero agradecer a mis sinodales M.C. Alejandro Hernández Tello, Dr. Cuauhtémoc Castañeda Roldán y M.C. Mario Lomelí Haro, por las aportaciones tan valiosas para la mejora de este trabajo y por el tiempo dedicado a la revisión. Al jefe de carrera Dr. José Margarito Hernández Morales y a la secretería Rita García Fernández por las facilidades otorgadas. A todos los maestros que contribuyeron en mi formación académica.

Mis agradecimientos a las maestras Alma Lidia Piceno Rivera y Luz del Carmen Álvarez Marín quienes además de compartirme sus conocimientos me brindaron su amistad. También a Lili por su amistad, apoyo y por los momentos tan bonitos que compartimos a lo largo de la carrera.

Por supuesto a mis padres por su apoyo, cariño, esfuerzo y dedicación para que lograra este triunfo en mi vida. A mis hermanos, Iraís, Baruc y Nubia, por sus muestras de cariño y apoyo; los quiero mucho.

Especialmente a mis hijos, porque han sido el motor principal para seguir adelante y porque han comprendido mis ausencias. Querido esposo, gracias por tu amor, apoyo, comprensión, por alentarme a terminar este trabajo, por tus aportaciones y sugerencias.

A Dios por la vida, por la linda familia a la que pertenezco y por la que me permitió formar.

Prefacio

La teoría sobre redes neuronales es muy amplia [1, 11, 12]. Debido quizás a esto nos encontramos con que en la literatura referente a redes neuronales se describen de manera muy general sólo los tipos de redes más representativos, o por el contrario, se enfoca al estudio de un modelo en particular. Otra de las tendencias que se ha observado es que la mayor parte de la literatura se concentra en el estudio de las redes neuronales de aprendizaje supervisado por lo que es difícil encontrar información detallada sobre redes neuronales de aprendizaje no supervisado.

Dado que en el aprendizaje no supervisado no se tiene conocimiento *a priori* de qué salida debe asociarse a cada entrada, la elección del algoritmo de aprendizaje es un punto central en el desarrollo del mismo, por ello es necesario conocer las ventajas y limitantes de cada uno de los algoritmos y así seleccionar el que mejor se adapte a las características del problema.

La gama de algoritmos de aprendizaje no supervisado es muy amplia, pero nos ha llamado la atención el algoritmo *wake-sleep*, implementado en la red conocida como máquina de Helmholtz la cual tiene conexiones hacia adelante y hacia atrás, debido a su estructura que de manera autónoma crea modelos y los compara durante fases intermedias para optimizar el aprendizaje. La literatura señala que se han obtenido buenos resultados pero no existe comparativa con otros modelos.

El objetivo principal de este trabajo es presentar de manera clara y ordenada la teoría sobre redes neuronales encaminada al estudio de los modelos de redes neuronales con aprendizaje no supervisado, así como conocer y comparar el algoritmo *wake-sleep* con otros modelos de redes con aprendizaje no supervisado. Para este fin se debe comprender la teoría básica de Redes Neuronales Artificiales (RNA). Es por ello que este trabajo está estructurado de la siguiente manera:

En el Capítulo 1 se dan a conocer, de manera cronológica, algunos investigadores que iniciaron el estudio de las RNA, se describe de manera muy general la neurona biológica y se observa su similitud con el modelo artificial, para dar paso al concepto de una Red Neuronal Artificial y sus elementos, finalizando con un pequeño y sencillo ejemplo.

Ya que nuestro objetivo es estudiar redes con aprendizaje no supervisado, se deben conocer las características del aprendizaje supervisado, para establecer las diferencias y ventajas entre el aprendizaje supervisado y no supervisado. Por ello, el Capítulo 2 aborda el tema de aprendizaje supervisado así como las principales redes con este tipo de aprendizaje: el Perceptrón, la red ADALINE y la red Retropropagación. Para cada red se muestra su estructura, su algoritmo de aprendizaje y se concluye con un ejemplo del algoritmo de aprendizaje para la red Retropropagación.

Posteriormente, en el Capítulo 3 se estudian las redes con aprendizaje no supervisado como: la red de Hopfield, los Mapas Auto-Organizativos de Kohonen (SOM, por sus siglas en inglés) y la red LVQ. Para cada red se describe: la estructura, el algoritmo de aprendizaje y se concluye con un ejemplo para una mejor comprensión del algoritmo de aprendizaje.

El Capítulo 4 está dedicado al estudio del algoritmo *wake-sleep*, en este se describe la estructura de la red, se mencionan algunos conceptos de probabilidad que se emplearán en el proceso de aprendizaje de

la red, se describe el proceso de reconocimiento (hacia adelante) y generativo (hacia atrás) del algoritmo wake-sleep, se detalla el algoritmo de aprendizaje y por último se muestra un ejemplo para ilustrar el funcionamiento del algoritmo de aprendizaje.

Recordemos que uno de nuestros objetivos es comparar el comportamiento de la máquina de Helmholtz con otras redes de aprendizaje no supervisado, es por ello que en el Capítulo 5 se eligió el problema de identificar barras horizontales y verticales en imágenes en blanco y negro, ya que el reconocimiento de patrones en imágenes digitales es la aplicación que mejor se ajusta a la mayoría de los diferentes tipos de redes. Particularmente se trabajó con imágenes de 3×3 píxeles ya que con imágenes de mayor tamaño el tiempo de ejecución aumenta exponencialmente.

Por último se realizó la comparación del desempeño y se identificó que las redes que clasificaron correctamente mayor número de patrones fueron la red LVQ y la máquina de Helmholtz.

Índice general

Prefacio	V
Lista de figuras	IX
Lista de tablas	XI
1. Introducción a las redes neuronales	1
1.1. Antecedentes	1
1.2. Neurona biológica	3
1.3. Neurona artificial	4
1.3.1. Funciones de activación	6
1.3.2. Ejemplo del funcionamiento de una neurona	7
1.4. Redes neuronales artificiales	9
1.4.1. Ejemplo del funcionamiento de una red neuronal multicapa	14
2. Aprendizaje	17
2.1. Aprendizaje supervisado	17
2.1.1. Perceptrón	17
2.1.2. ADALINE	19
2.1.3. Retropropagación	20
3. Aprendizaje no supervisado	27
3.1. Red de Hopfield	28
3.1.1. Estructura de la red	28
3.1.2. Algoritmo de aprendizaje y funcionamiento	29
3.1.3. Ejemplo	30
3.2. Mapas Auto-Organizativos de Kohonen	31
3.2.1. Estructura de la red	32
3.2.2. Algoritmo de aprendizaje	33
3.2.3. Ejemplo	34
3.3. Learning Vector Quantization	36
3.3.1. Estructura de la red	36
3.3.2. Algoritmo de aprendizaje	38
3.3.3. Ejemplo	39
4. Algoritmo Wake-Sleep	43
4.1. Estructura de la red	43
4.2. Algoritmo de aprendizaje	47
4.3. Ejemplo	49

5. Comparación de redes no supervisadas	53
5.1. Red de Hopfield	54
5.2. Red LVQ	57
5.3. Algoritmo Wake-Sleep	61
Bibliografía	67
A. Código fuente	69
A.1. Red de Hopfield	69
A.1.1. Función auxiliar: Aprendizaje_Hopfield	69
A.1.2. Función auxiliar: compara	69
A.1.3. Función principal: Hopfield	70
A.2. Red LVQ	71
A.2.1. Función principal: Redlvq	71
A.3. Algoritmo Wake-Sleep	72
A.3.1. Función auxiliar: logsig1	72
A.3.2. Función auxiliar: sample1	72
A.3.3. Función auxiliar: combinaciones	72
A.3.4. Función auxiliar: prob_pgd	73
A.3.5. Función auxiliar: KL	73
A.3.6. Función principal: wake_sleep	74
B. Conjunto de entrenamiento	77
B.1. Conjunto de entrenamiento para la red LVQ	77

Índice de figuras

1.1. Neurona biológica.	3
1.2. Neurona biológica y esquema detallado de una neurona artificial.	5
1.3. Esquema que muestra el proceso general de una neurona.	5
1.4. Esquema abreviado de una neurona con múltiples entradas.	6
1.5. Neurona con múltiples entradas y función hardlims.	8
1.6. Topología de una red multicapa.	9
1.7. Esquema detallado de una red monocapa.	10
1.8. Esquema abreviado de una red monocapa.	11
1.9. Arquitectura de red neuronal de tres capas.	11
1.10. Tabla de verdad y representación gráfica de la función XOR.	14
1.11. Topología y arquitectura del Perceptrón multicapa para la función XOR.	14
2.1. Arquitectura de la red Perceptrón.	18
2.2. Arquitectura de la red ADALINE.	19
2.3. Arquitectura de una red Retropropagación de tres capas.	21
2.4. Arquitectura de la red Retropropagación para aproximar la función XOR.	22
3.1. Topología de una red de Hopfield de 6 neuronas.	29
3.2. Arquitectura de la red de Hopfield.	30
3.3. Red SOM.	32
3.4. Vecindades.	33
3.5. Topología rectangular y hexagonal.	33
3.6. Topología de la red SOM para el ejemplo de la sección 3.2.3.	34
3.7. Estructura de la red LVQ.	37
3.8. Topología de la red LVQ del ejemplo de la sección 3.3.3.	39
3.9. Arquitectura de la red del ejemplo de la sección 3.3.3.	40
4.1. Topología de una red tipo máquina de Helmholtz.	44
4.2. Arquitectura de la máquina de Helmholtz.	47
4.3. Divergencia entre $p(d)$ y $p_G(d)$ respecto al número de iteraciones para el ejemplo 4.3.	51
5.1. Imágenes con barras horizontales y verticales.	53
5.2. Codificación de los patrones de entrada.	54
5.3. Imágenes que no tienen barras horizontales ni verticales.	54
5.4. Imágenes del conjunto de simulación que se estabilizaron en la red de Hopfield.	56
5.5. Imágenes que devuelve la red de Hopfield para los patrones estabilizados.	56
5.6. Algunas imágenes de patrones que no se estabilizaron en la red de Hopfield.	57
5.7. Imágenes de los patrones que forman las subclases.	58
5.8. Algunos patrones que la red LVQ no clasifica correctamente.	60
5.9. Topología de la máquina de Helmholtz (9-6-1).	62
5.10. Divergencia entre $p(d)$ y $p_G(d)$ respecto al número de iteraciones.	64

Lista de tablas

1.1. Funciones de activación.	7
1.2. Clasificación de las RNA de acuerdo al número de capas y tipos de conexiones.	13
1.3. Clasificación de las RNA de acuerdo al valor de entrada y mecanismo de aprendizaje.	13
3.1. Distintas formas de regiones.	37
4.1. Probabilidades asignadas a los patrones del ejemplo 4.3.	49
4.2. Distribución de probabilidad generativa después de una iteración.	50
4.3. Probabilidades generativas con 50,000 iteraciones.	51
5.1. Conjunto de entrenamiento para la red de Hopfield.	55
5.2. Comportamiento de la red de Hopfield.	55
5.3. Comportamiento de la red LVQ.	59
5.4. Comportamiento de la máquina de Helmholtz.	61
5.5. Resultados de la máquina de Helmholtz después de 60,000 iteraciones.	63
B.1. Conjunto de entrenamiento para la red LVQ.	78

Capítulo 1

Introducción a las redes neuronales

1.1. Antecedentes

El camino hacia la construcción de máquinas inteligentes comienza en la Segunda Guerra Mundial con el diseño de computadoras analógicas ideadas para controlar cañones antiaéreos o para la navegación. Algunos investigadores observaron que existían semejanzas entre el funcionamiento de estos dispositivos de control y los sistemas reguladores de los seres vivos. De este modo, combinando los avances de la electrónica de la posguerra y los conocimientos sobre los sistemas nerviosos de los seres vivos, inició el reto de construir máquinas capaces de responder y aprender como los animales.

A finales del siglo XIX y principios del siglo XX, surgieron teorías que fundamentaron el campo de las redes neuronales; principalmente las aportaciones del trabajo interdisciplinario en física, psicología y neurofisiología realizada por científicos como Hermann von Helmholtz, Ernst Mach e Ivan Pavlov. Los primeros estudios se concentraron en teorías generales de aprendizaje y visión, entre otras, pero aún no incluían modelos matemáticos específicos de operación de la neurona.

El trabajo de Warren McCulloch y Walter Pitts, de 1940 es reconocido como el origen del campo de las redes neuronales [11].

En 1949, Donald Hebb escribió *The organization of Behavior*, donde planteó el postulado que lleva su nombre el cual señala el hecho de que las conexiones nerviosas se fortalecen cada vez que se usan, un concepto fundamental de aprendizaje humano [1].

La primera aplicación práctica de una red neuronal artificial ocurre a finales de 1950 con la invención del Perceptrón y la regla de aprendizaje asociada a esta red. Frank Rosenblatt junto con sus compañeros construyeron el Perceptrón y demostraron su habilidad para realizar reconocimiento de patrones. Este modelo era capaz de generalizar, es decir, después de haber aprendido una serie de patrones podía reconocer otros similares, aunque no se le hubieran presentado anteriormente. Sin embargo, tenía una serie de limitaciones, quizás la más conocida era la incapacidad para resolver el problema de la función XOR o disyunción exclusiva y, en general, no era capaz de clasificar patrones que no fueran linealmente separables.

En 1960, Bernard Widrow y Marcian Hoff desarrollaron la red ADALINE (acrónimo que proviene de las siglas ADAPtative LInear NEuron) que sirvió como una máquina de clasificación y permitió ilustrar los principios de comportamiento adaptativo de las redes neuronales. Este modelo fue similar en capacidad y estructura al Perceptrón de Rosenblatt.

Marvin Minsky y Seymour Papert publicaron el libro *Perceptrons* en 1969, que además de contener un análisis matemático detallado del Perceptrón, consideraba que la extensión a perceptrones multinivel era completamente estéril.

Mucha gente, influenciada por Minsky y Papert, creyó que la investigación sobre redes neuronales había concluido. Esto, combinado con el hecho de que no había computadoras adecuadas para experimentar, causó que muchos investigadores dejaran el campo. Por todo lo anterior la investigación sobre redes neuronales prácticamente se suspendió por más de una década.

En 1972 Teuvo Kohonen y James Anderson separada e independientemente desarrollaron nuevas redes neuronales que podían actuar como memorias asociativas. Stephen Grossberg también estuvo muy activo durante este periodo en la investigación de redes auto-organizativas [11, 16].

El interés en las redes neuronales vaciló a finales de 1960 a causa de la carencia de nuevas ideas y computadoras lo suficientemente poderosas para experimentar, pero durante la década de 1980 se superaron ambas limitaciones e incrementó el número de investigadores en el área.

Dos conceptos nuevos fueron los responsables del renacimiento de las redes neuronales. El primero fue el uso de la mecánica estadística para explicar la operación de ciertas clases de redes recurrentes, las cuales se podían usar como una memoria asociativa. La segunda clave del desarrollo de 1980 fue el algoritmo retropropagación (backpropagation) para entrenar redes tipo Perceptrón multicapa, las cuales fueron propuestas independientemente por varios investigadores. La publicación [11] más influyente del algoritmo retropropagación fue de David Rumelhart y James McClelland. Este algoritmo fue la respuesta a la crítica hecha por Minsky y Papert en 1960.

John Hopfield en 1982 publicó un artículo para la Academia Nacional de Ciencias, en el que propuso el modelo de la red neuronal de Hopfield. Este modelo es capaz de almacenar ciertos recuerdos o patrones de una manera similar al cerebro humano y recuperar un patrón completo presentándole sólo una parte de él [1].

En 1986, el desarrollo del algoritmo retropropagación fue dado a conocer por Rumelhart, Hinton y Williams. Ese mismo año, el libro *Parallel Distributed Processing*, fue publicado por Rumelhart y McClelland, siendo este libro la mayor influencia para la aplicación generalizada del algoritmo retropropagación [1].

La creación de nuevos modelos y los buenos resultados de éstos reforzaron el campo de las redes neuronales. En los últimos años ha crecido la investigación en el área de redes neuronales y se han encontrado muchas aplicaciones. Se han aplicado de manera exitosa en diversos campos tales como medicina, comunicaciones, economía, comercio y manufactura, ya que han sido el mejor medio para identificar patrones o tendencias en datos. En seguida se detallan algunas aplicaciones específicas.

Aplicaciones en medicina. Una red neuronal entrenada localiza y clasifica en imágenes la posible existencia de tumores cancerígenos [21]; en el tratamiento de problemas del corazón, la tarea de la red neuronal es predecir el posible riesgo de intoxicación por el fármaco digoxina [4, 27, 19]; asimismo, se emplea en diagnóstico de hepatitis [3].

Procesamiento de señal. Ecuilibrar un canal consiste en recuperar la señal que, al pasar a través de un canal de comunicaciones, sufre una distorsión. Cuando la señal de origen es binaria, este problema se puede considerar como un problema de clasificación [10, 5, 22].

Economía. En las solicitudes de crédito de instituciones financieras, existen modelos de redes neuronales que de acuerdo a datos o indicadores económicos de la persona que solicita el crédito determinan la viabilidad del mismo [26]; de la misma manera, una red neuronal puede determinar el riesgo de quiebra de un banco con base en determinados parámetros económicos [18]; también se han aplicado redes neuronales de manera exitosa en pronósticos de venta, investigaciones de mercado y análisis de riesgo.

Procesamiento de imágenes. Problemas tales como reconocimiento de objetos tridimensionales, de palabras escritas a mano y de rostros han sido resueltos mediante el uso de redes neuronales [3]. Otra

aplicación consiste en Sistemas de Reconocimiento Óptico de Caracteres (OCR, por sus siglas en inglés), en los que a partir de la imagen de un texto escrito o impreso en papel o similar, crean un archivo de texto en la cual se almacenan los caracteres, palabras y párrafos identificados [24].

1.2. Neurona biológica

La revelación de la compleja estructura del sistema nervioso central tiene unos cien años aproximadamente. En la segunda mitad del siglo XIX prevalecían dos escuelas de investigadores del sistema nervioso: la de los reticularistas y la de los neuronistas o neuristas. Los primeros argumentaban que el sistema nervioso se encontraba formado por una continua e ininterrumpida red de fibras nerviosas; los segundos, defendían que el sistema nervioso estaba formado por un vasto número de simples unidades celulares interconectadas entre sí, las neuronas. Gracias al doctor Santiago Ramón y Cajal en 1888, quien aplicó un técnica en la cual coloreó las fibras nerviosas bajo una reacción de dicromato de plata, se eliminó la doctrina del reticularismo al descubrir que existen separaciones entre las membranas de las neuronas presinápticas y las postsinápticas donde tienen lugar algunos fenómenos electrolíticos y bioquímicos [23].

El cerebro consta de un gran número de células altamente interconectados, llamadas neuronas (aproximadamente 10^{11} neuronas y 10^4 conexiones por neurona), éstas son el elemento más importante en el sistema nervioso y las responsables de convertir un estímulo en señales o impulsos eléctricos de rápida conducción a través de grandes distancias en el cuerpo. La investigación detallada de la estructura interna de las células nerviosas, especialmente después de la invención del microscopio electrónico, ha revelado que todas las neuronas están constituidas por las mismas partes básicas, independientemente del tamaño y forma de la neurona. Estas partes básicas son tres: las dendritas, el soma o cuerpo de la célula y el axón. La parte central de la célula es llamada cuerpo de la célula o soma; cuyo tamaño típico es de unos 10 a 80 μm (donde 1 $\mu\text{m} = 1 \times 10^{-6}$ m.). Desde el soma se proyectan numerosas extensiones en forma de raíz llamadas dendritas. La neurona también está formada por una fibra tubular simple conocida como axón, el cual se divide en numerosas ramificaciones [1], como se puede observar en la Figura 1.1.

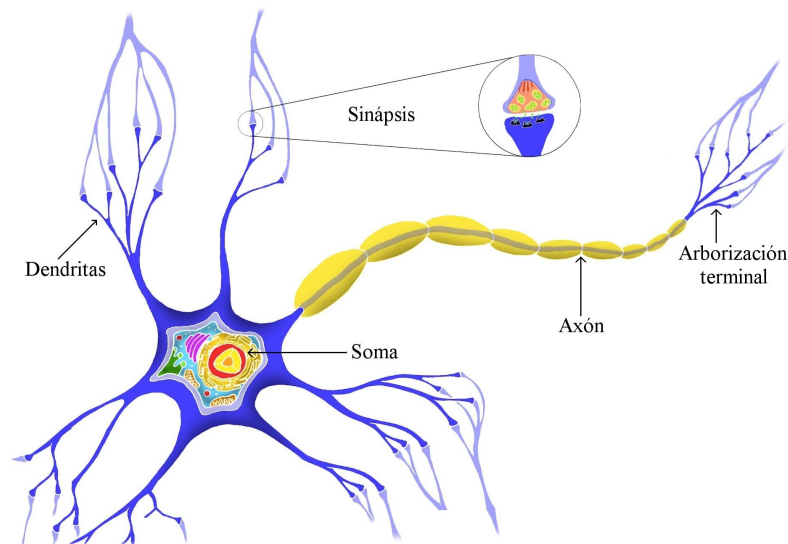


Figura 1.1: Neurona biológica.

Las dendritas reciben estímulos de otras células, éstas junto con el soma los procesan e integran, después se transmiten a lo largo del axón. El axón es la línea de transmisión de impulsos eléctricos desde el cuerpo de la célula hacia otras neuronas, y puede presentar ramas colaterales a lo largo de la rama principal. Cuando los axones alcanzan su destino final se ramifican nuevamente en lo que se llama una

arborización terminal. Se le llama sinapsis al espacio que existe entre las dendritas y axones, es un medio acuoso ionizado principalmente por iones de sodio y potasio y es ahí en donde los impulsos eléctricos se transfieren. Los iones dan ciertas propiedades de conductividad al espacio intersináptico, de manera que inhiben o potencializan la transmisión de impulsos eléctricos permitiendo a una célula influir en la actividad de otras. Es importante señalar que la sinapsis puede conectarse con el cuerpo de la célula, axones y con otras sinapsis [1, 16, 23].

Los potenciales de acción, son señales de baja frecuencia conducidas en forma muy lenta, estos no pueden saltar de una célula a otra, la comunicación entre neuronas viene siempre mediada por transmisores químicos que son liberados en las sinapsis, llamados neurotransmisores. Existen dos tipos de sinapsis: las sinapsis excitadoras, cuyos neurotransmisores provocan disminuciones de potencial en la membrana de la célula postsináptica, facilitando la generación de impulsos a mayor velocidad, y las sinapsis inhibitoras, cuyos neurotransmisores tienden a estabilizar el potencial de la membrana, dificultando la emisión de impulsos.

La neurona recibe las entradas procedentes de sinapsis excitadoras e inhibitoras, es decir, recibe impulsos amplificados o atenuados. La suma de estos impulsos en el cuerpo de la célula determinará si se activa o no, dependiendo de que si el valor de dicha suma es superior al valor de umbral de generación del potencial de acción [12].

1.3. Neurona artificial

Una neurona biológica es una estructura inmensamente compleja y puede ser modelada con distintos grados de complejidad. Si se tratara de poner en un modelo todo lo que se conoce acerca de la neurona biológica, sería imposible trabajar con éste (con la tecnología actual). Por lo tanto se debe usar un modelo simplificado que sea adecuado al uso que se requiera. Este modelo simplificado, en este caso, incluirá las señales que provienen de otras neuronas, la intensidad de la sinapsis y una función que definirá el umbral, estos elementos se describirán brevemente a continuación.

El modelo de una neurona artificial es una imitación del proceso de una neurona biológica. En la literatura existen varias formas para denominar a una neurona artificial: nodo, neuronodo, celda, unidad o elemento de procesamiento (PE), etc. En este trabajo se denominará neurona artificial.

De la observación detallada del proceso biológico, se ha desarrollado un modelo artificial simplificando el funcionamiento de la neurona biológica, este modelo consta de:

- Las entradas p_i que representan las señales que provienen de otras neuronas y que son capturadas por las dendritas.
- Los pesos W_i que son la intensidad de la sinapsis que conecta dos neuronas; tanto p_i como W_i son valores reales.
- f que es la función que establece el umbral que la neurona debe sobrepasar para activarse; este proceso ocurre biológicamente en el cuerpo de la célula.

En la Figura 1.2 se observa un esquema de una neurona artificial y su similitud con una neurona biológica.

Las señales de entrada a una neurona artificial p_1, p_2, \dots, p_R son variables continuas en lugar de pulsos discretos, como se presentan en una neurona biológica. Cada señal de entrada es potencializada o atenuada por el peso sináptico o intensidad de conexión cuya función es análoga a la función de las sinapsis de la neurona biológica. Los pesos pueden ser positivos (excitatorios), o negativos (inhibitorios), el nodo sumatorio acumula las señales ponderadas y las pasa a la salida a través de una función umbral (también conocida como función de activación o transferencia)[30].

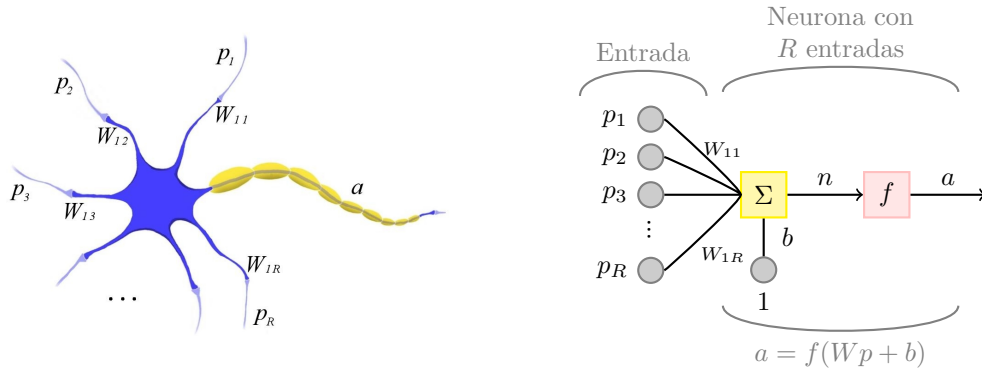


Figura 1.2: Una neurona biológica (izq.) y esquema detallado de una neurona artificial (der.).

La Figura 1.3 muestra una idea más clara de este proceso, en donde se puede observar que cada señal es ponderada por un peso, éstas se suman para después aplicar la función de activación y así obtener la salida correspondiente.

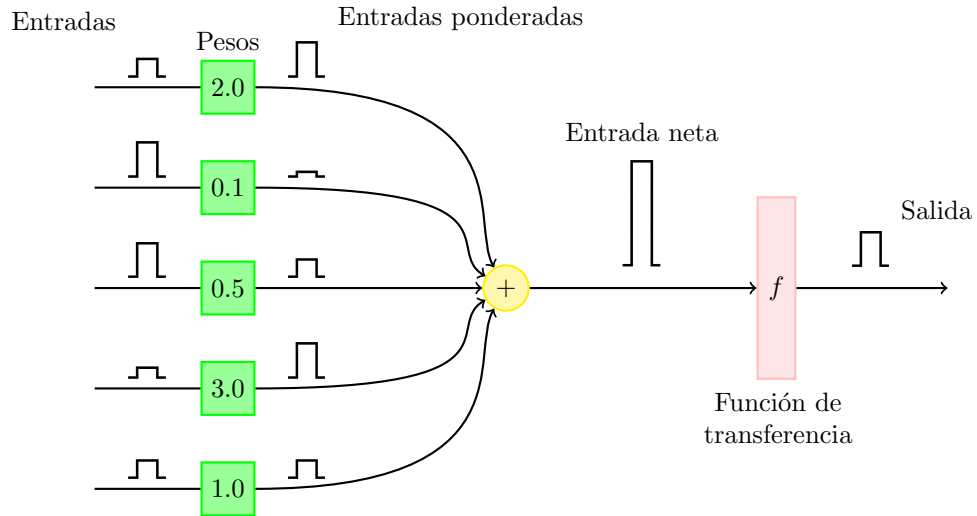


Figura 1.3: Esquema que muestra el proceso general de una neurona.

Definición 1.1 Una neurona artificial es un modelo compuesto por un vector p que representa las señales de entrada, un vector de pesos W que representa las conexiones sinápticas de la neurona biológica y que amplifica o atenúa la entrada p , un escalar b que representa el bias o ganancia, una función de activación f y un escalar a , que representa la salida dada por la siguiente expresión

$$a = f(Wp + b)$$

donde $W = (W_{11} \ W_{12} \ \dots \ W_{1R})$ y $p = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{pmatrix}$, al producto de los pesos por la entrada más el bias se le denomina entrada neta (n) luego $a = f(n)$ con:

$$n = W_{11}p_1 + W_{12}p_2 + \dots + W_{1R}p_R + b.$$

El esquema detallado de una neurona artificial se ilustra en la Figura 1.2. Los subíndices de la matriz de pesos, W , representan los términos involucrados en la conexión, el primer subíndice representa la neurona destino y el segundo, representa la fuente de la señal que alimenta a la neurona. Por ejemplo, los subíndices de W_{1R} indican que este peso es la conexión desde la R -ésima entrada a la primera neurona. Esta convención se hace más útil cuando hay más de una neurona.

Cuando se tiene una neurona con demasiados parámetros la notación de la Figura 1.2, puede resultar inapropiada y se emplea la notación abreviada representada en la Figura 1.4.

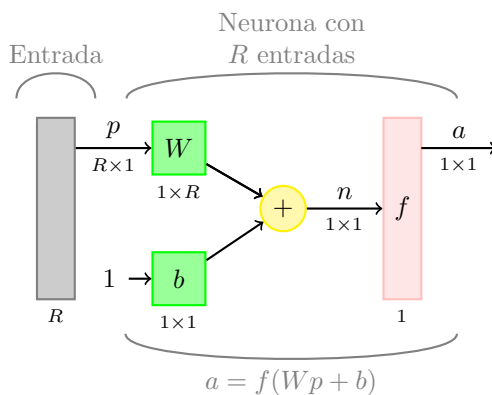


Figura 1.4: Esquema abreviado de una neurona con múltiples entradas.

En el esquema simplificado el vector de entrada p es representado por la barra sólida vertical, en color gris, a la izquierda. Las dimensiones de p se muestran en la parte inferior de la variable como R y bajo el nombre de la entrada aparece $R \times 1$, indicando que el vector de entrada es un vector columna de R elementos. Las entradas van a la matriz de pesos W , en color verde, la cual tiene R columnas y solo una fila para el caso de una sola neurona. Una constante con valor 1 entra a la neurona multiplicada por la ganancia escalar b , en verde. En amarillo, la entrada neta a la neurona, es decir la suma de todas sus entradas, y en rosa la función de activación f . La salida de la red a , es en este caso un escalar.

1.3.1. Funciones de activación

En la sección 1.2 se explicó el proceso biológico (potencial de acción) mediante el cual las neuronas se activan o no. Este comportamiento de la neurona biológica se modela en una neurona artificial mediante la función de activación. Una función de activación de una neurona artificial define la salida de esa neurona dada una entrada o conjunto de entradas.

Existen diferentes funciones de activación o de transferencia [12, 30], como las funciones Limitador fuerte, Limitador fuerte simétrico (ambas conocidas como funciones tipo escalón) o la función de tipo Lineal. Las funciones de transferencia más comunes se definen en la Tabla 1.1 además de que muestra algún ejemplo de la red en la que se emplea cada función y el nombre y símbolo empleado en MATLAB para identificarla.










Nombre	Función	Red	Nombre en MATLAB	Símbolo
Limitador fuerte	$a = \begin{cases} 0, & n < 0 \\ 1, & n \geq 0 \end{cases}$	Perceptrón	hardlim	
Limitador fuerte simétrico	$a = \begin{cases} -1, & n < 0 \\ +1, & n \geq 0 \end{cases}$	Perceptrón	hardlims	
Lineal positiva	$a = \begin{cases} 0, & n < 0 \\ x, & n \geq 0 \end{cases}$	Perceptrón	poslin	
Lineal	$a = n$	ADALINE	purelin	
Lineal saturado	$a = \begin{cases} 0, & n < 0 \\ n, & 0 \leq n \leq 1 \\ 1, & n > 1 \end{cases}$	ADALINE	satlin	
Lineal saturado simétrico	$a = \begin{cases} -1, & n < -1 \\ n, & -1 \leq n \leq 1 \\ +1, & n > 1 \end{cases}$	ADALINE	satlins	
Sigmoidal logarítmico	$a = \frac{1}{1+e^{-n}}$	Retropropagación	logsig	
Tangente sigmoidal hiperbólica	$a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$	Retropropagación	tansig	
Competitiva	$a = \begin{cases} a(i^*) = 1 & \text{si } i^* \\ & \text{es la neurona ganadora} \\ a(i) = 0 & \\ & \text{en otro caso} \end{cases}$	Learning Vector Quantization	compet	

Tabla 1.1: Funciones de activación.

1.3.2. Ejemplo del funcionamiento de una neurona

Se ilustrará con un ejemplo el funcionamiento de una neurona. Se implementará la función lógica AND, tomando -1 como el valor de verdad falso y 1 como el valor de verdad verdadero, luego su tabla de verdad es:

x	y	x AND y
-1	-1	-1
1	-1	-1
-1	1	-1
1	1	1

Se establecen los pesos $W = (1 \ 1)$ y el bias (umbral) en $b = -0.5$, además se empleará la función de activación *hardlims* como se muestra en el esquema de la red en la Figura 1.5.

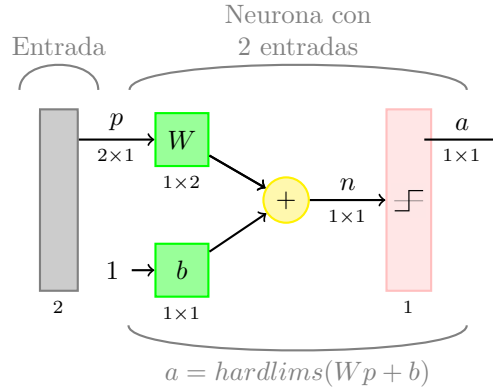


Figura 1.5: Neurona con múltiples entradas y función hardlims.

Se calcula la salida de la red para cada una de las entradas $p_1 = \begin{pmatrix} -1 \\ -1 \end{pmatrix}$, $p_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$, $p_3 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$ y $p_4 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ de la siguiente manera:

$$a_i = \text{hardlims}(n_i) = \text{hardlims}(Wp_i + b)$$

$$a_1 = \text{hardlims} \left((1 \ 1) \begin{pmatrix} -1 \\ -1 \end{pmatrix} - 0.5 \right) = \text{hardlims}(-2.5) = -1$$

$$a_2 = \text{hardlims} \left((1 \ 1) \begin{pmatrix} 1 \\ -1 \end{pmatrix} - 0.5 \right) = \text{hardlims}(-0.5) = -1$$

$$a_3 = \text{hardlims} \left((1 \ 1) \begin{pmatrix} -1 \\ 1 \end{pmatrix} - 0.5 \right) = \text{hardlims}(-0.5) = -1$$

$$a_4 = \text{hardlims} \left((1 \ 1) \begin{pmatrix} 1 \\ 1 \end{pmatrix} - 0.5 \right) = \text{hardlims}(1.5) = 1.$$

De esta manera se comprueba que la neurona devuelve los respectivos valores deseados para cada entrada como se muestra en la tabla de verdad del conectivo AND.

Aunque la definición de red dice que la entrada es un vector, se puede agrupar el conjunto de entradas y presentarlas como matriz, así p está formada por, $p = (p_1 \ p_2 \ p_3 \ p_4)$; la única consideración que se debe hacer es que el bias debe multiplicarse por un vector fila de tantos unos como patrones tenga el conjunto de entrada.

Así la salida de la red se calcula mediante la siguiente expresión:

$$a = \text{hardlims}(n) = \text{hardlims}(Wp + b).$$

Se calcula la entrada neta, es decir, la suma de las entradas por los pesos más el bias:

$$\begin{aligned} n &= Wp + b \\ &= (1 \ 1) \begin{pmatrix} -1 & 1 & -1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix} - (0.5 \ 0.5 \ 0.5 \ 0.5) \\ &= (-2.5 \ -0.5 \ -0.5 \ 1.5). \end{aligned}$$

La salida definitiva de la red se produce al aplicar la función de activación *hardlims* a cada una de las entradas del vector n .

$$a = \text{hardlims}(n) = (-1 \quad -1 \quad -1 \quad 1).$$

En donde la primera componente de a corresponde a la salida de la primera entrada, la segunda componente a la segunda entrada, así sucesivamente.

1.4. Redes neuronales artificiales

Las neuronas se pueden agrupar en sistemas más complejos conocidos como redes neuronales. El término red neuronal se emplea tanto para identificar redes biológicas como el cerebro humano, como para redes neuronales artificiales, que se refieren a simulaciones computacionales, mecánicas, eléctricas o a modelos de redes neuronales biológicas.

Definición 1.2 Una capa consta de un conjunto de neuronas, un vector de entrada p , donde cada entrada del vector p se conecta a cada neurona a través de la matriz de pesos W que agrupa los vectores de pesos de cada una de las neuronas, un vector bias b que agrupa los bias de cada neurona y una función de activación f que será la misma para todas las neuronas.

Definición 1.3 Una red neuronal está formada por un grupo interconectado de capas compuesta por una o varias matrices de pesos W , sus respectivos vectores de ganancias (bias) b , un vector de entradas p , un vector de salida a , y una o varias funciones de activación.

De acuerdo a la ubicación de la capa en la Red Neuronal Artificial (RNA), ésta recibe diferentes nombres (Figura 1.6):

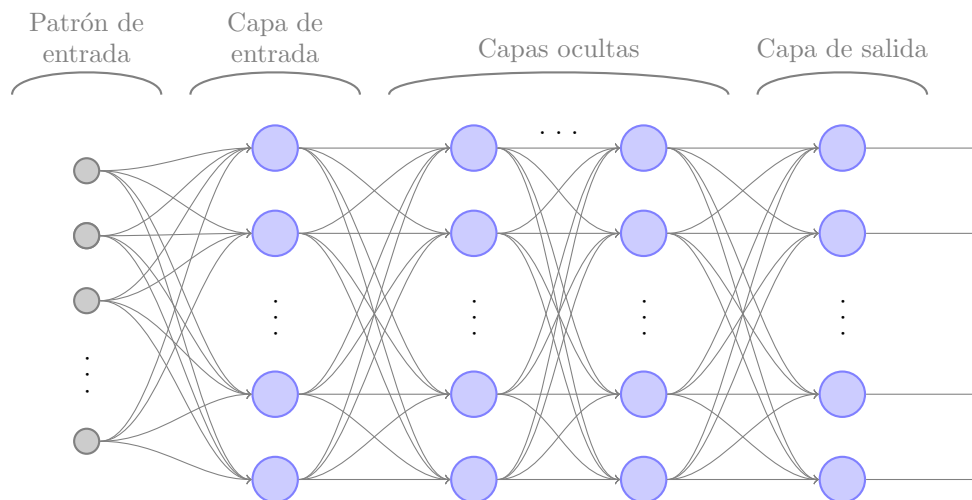


Figura 1.6: Topología de una red multicapa.

Capa de entrada: Recibe las señales de la entrada de la red.

Capa oculta: Es aquella que no tiene contacto con los datos ni de entrada ni de salida, recibe las señales de capas previas y los propaga a capas posteriores. El número de capas ocultas puede variar e incluso ser igual a cero.

Capa de salida: Recibe la información de la capa anterior y transmite la respuesta al medio externo [30, 12].

Con los elementos mencionados anteriormente las redes neuronales se pueden clasificar de acuerdo al número de capas.

Definición 1.4 *Red neuronal monocapa.* Es la red neuronal más sencilla ya que tiene una sola capa de neuronas que pondera y suma las entradas para obtener la salida. La capa incluye una matriz de pesos, un vector de ganancias y una función de transferencia [28].

En la Figuras 1.7 y 1.8 se observan los esquemas detallado y abreviado, respectivamente, de una red de una sola capa con S neuronas, en la cual, cada una de las R entradas se conecta a cada una de las S neuronas, la matriz de pesos tiene ahora S filas lo mismo que el vector bias y tienen la siguiente forma:

$$W = \begin{pmatrix} W_{11} & W_{12} & \dots & W_{1R} \\ W_{21} & W_{22} & \dots & W_{2R} \\ \vdots & \vdots & \ddots & \vdots \\ W_{S1} & W_{S2} & \dots & W_{SR} \end{pmatrix} \quad \text{y} \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{pmatrix}.$$

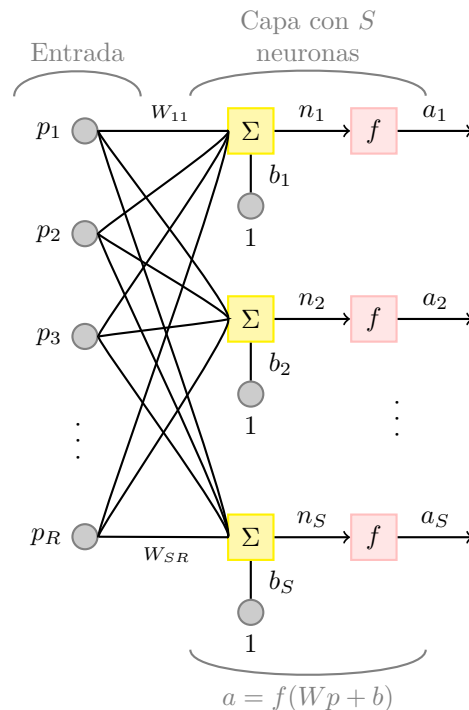


Figura 1.7: Esquema detallado de una red monocapa.

Definición 1.5 *Red neuronal multicapa.* Es una generalización de la red monocapa, en el que la salida de una capa se toma como entrada para otra capa surgiendo así capas ocultas entre la entrada y la salida [28].

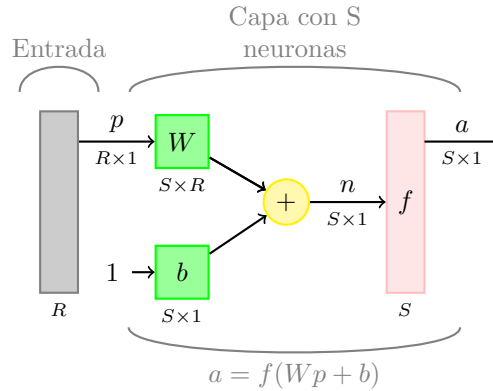


Figura 1.8: Esquema abreviado de una red monocapa.

Si se considera una red con varias capas, o red multicapa, cada capa tendrá su propia matriz de pesos W , su propio vector de ganancias b , un vector de entradas netas n , una función de activación f y un vector de salida a .

Para la red que se muestra en la Figura 1.9 se tienen R entradas, S^1 neuronas en la primera capa (donde el superíndice indica el número de capa), S^2 neuronas en la segunda capa y S^3 neuronas en la tercera capa, las salidas de las capas 1 y 2 son las entradas a las capas 2 y 3 respectivamente, así la capa 2 se puede ver como una red de una capa con $R = S^1$ entradas, $S = S^2$ neuronas y una matriz de pesos W^2 de dimensiones $S^2 \times S^1$; la entrada a la segunda capa es a^1 y la salida es a^2 . Similarmente, la capa 3 se puede ver como una red de una capa con $R = S^2$ entradas, $S = S^3$ neuronas y una matriz de pesos W^3 de dimensiones $S^3 \times S^2$, la entrada a la tercera capa es a^2 y la salida es a^3 .

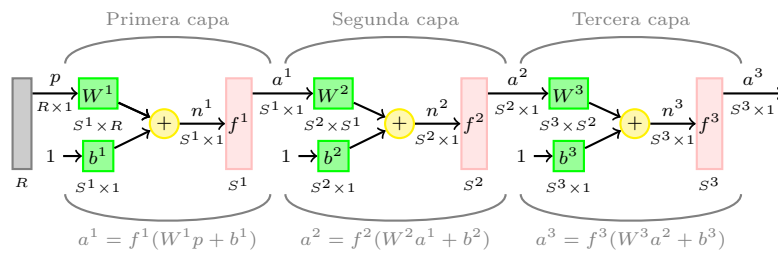


Figura 1.9: Arquitectura de red neuronal de tres capas.

En general las redes neuronales se pueden clasificar de diversas maneras [12]:

Según su topología. Se distingue entre redes con conexiones hacia adelante, conexiones hacia atrás y conexiones autorrecurrentes.

- Redes con conexiones hacia adelante, *feedforward*, las señales se propagan hacia adelante a través de las capas de la red.
- Redes con conexiones hacia atrás, *feedback*, las salidas de las neuronas de capas posteriores se conectan a las entradas de las capas anteriores.
- Redes que disponen de conexiones tanto hacia adelante como hacia atrás o redes *feedforward/feedback*.

- Redes recurrentes, éstas redes se caracterizan por la existencia de conexiones de realimentación. Estas conexiones pueden ser entre neuronas de diferentes capas o de la misma capa.
- Redes autorrecurrentes, en este tipo de redes se presentan autoconexiones, es decir, conexiones de una neurona consigo misma.

Mecanismo de aprendizaje. Una de las clasificaciones que se realiza en las redes neuronales corresponde al tipo de aprendizaje que utilizan, así se pueden distinguir:

- Redes con aprendizaje supervisado.
- Redes con aprendizaje no supervisado¹.

Tipo de aprendizaje. Es otro criterio que se utiliza para diferenciar las reglas de aprendizaje, se basa en considerar si la red puede aprender durante su funcionamiento o si el aprendizaje supone la desconexión de la red.

- Aprendizaje OFF LINE, se distingue entre una fase de aprendizaje o entrenamiento y un fase de operación o funcionamiento, existiendo un conjunto de datos de entrenamiento y un conjunto de datos de prueba que serán utilizados en la fase correspondiente. En este tipo de aprendizaje la matriz de pesos permanece fija después que termina la etapa de entrenamiento de la red.
- Aprendizaje ON LINE, no se distingue entre fase de entrenamiento y de operación, de tal forma que los pesos varían dinámicamente siempre que se presenta una nueva información al sistema.

Tipo de asociación entre la información de entrada y salida. Las redes neuronales son sistemas que almacenan cierta información aprendida; esta información se registra de forma distribuida en los pesos asociados a las conexiones entre neuronas. Por tanto puede imaginarse una red como cierto tipo de memoria que almacena datos de forma estable y que se grabarán en dicha memoria como consecuencia del aprendizaje de la red, estos datos podrán ser leídos a la salida como respuesta a cierta información de entrada, comportándose la red como lo que habitualmente se conoce por memoria asociativa; es decir, cuando se aplica un estímulo la red responde con una salida asociada a dicha información de entrada. Existen dos formas de realizar esta asociación entre entrada-salida:

- Heteroasociación: la red neuronal aprende parejas de datos $\{(A_1, B_1), (A_2, B_2), \dots, (A_N, B_N)\}$, de forma que cuando se presente cierta información de entrada A_i , deberá responder generando la correspondiente salida asociada B_i .
- Autoasociación: la red aprende ciertas informaciones A_1, A_1, \dots, A_N de tal forma que cuando se le presente cierta información de entrada, le asociará el dato almacenado más parecido. La principal misión de un red autoasociativa es reconstruir determinada información de entrada que se presenta incompleta o distorsionada.

En función de los valores de entrada-salida:

- Cuando tanto los datos de entrada como de salida son valores reales, comúnmente están normalizados y su valor absoluto es menor que la unidad. Cuando esto ocurre, las funciones de activación también son continuas, de tipo lineal o sigmoideal.
- Las redes cuyos valores de entrada son discretos o binarios $\{0, 1\}$, ó $\{1, -1\}$, generando también respuestas de tipo binario; las funciones de activación son de tipo escalón.

Las Tablas 1.2 y 1.3 muestran la clasificación de las RNA de acuerdo al número de capas y a los valores de entrada, respectivamente.

¹Los tipos de aprendizaje supervisado y no supervisado se describirán con detalle en los capítulos 2 y 3 respectivamente.

Número de capas	Tipo de conexiones	Ejemplos
Redes monocapa	Conexiones autorrecurrentes	Brain State in A Box Additive Grossberg (AG) Shunting Grossberg (SG) Optimal Linear Associative Memory
	Conexiones no autorrecurrentes	Hopfield Boltzmann Machine Cauchy Machine
Redes multicapa	Conexiones hacia adelante/feedforward	Adaline/Madaline Perceptrón Linear Adaptative Memory (LAM) Drive Reinforcement Retropropagación
	Conexiones hacia atrás/feedback	Mapas de Kohonen LAM
	Conexiones feedforward/feedback	ART (Adaptative Resonance Theory) BAM (Bidirectional Associative Memory)

Tabla 1.2: Clasificación de las redes neuronales de acuerdo al número de capas y tipos de conexiones.

	Entrada	Aprendizaje	Ejemplo
Redes Neuronales	Entrada binaria	Supervisado	Red Hamming Red Hopfield
		No supervisado	Art1 Art2
	Entrada continua	Supervisado	Perceptrón Retropropagación
		No supervisado	Mapas de Kohonen o LAM

Tabla 1.3: Clasificación de las redes neuronales de acuerdo al valor de entrada y mecanismo de aprendizaje.

1.4.1. Ejemplo del funcionamiento de una red neuronal multicapa

En la sección 1.1 se mencionó que una de las limitaciones de la red Perceptrón era la de clasificar patrones que no son linealmente separables. Dos conjuntos de puntos A y B son *linealmente separables* en un espacio n -dimensional si existen $n + 1$ números reales w_1, w_2, \dots, w_n, b , de manera que cada punto $(x_1, x_2, \dots, x_n) \in A$ satisface $\sum_{i=1}^n w_i x_i \geq b$ y cada punto $(x_1, x_2, \dots, x_n) \in B$ satisface $\sum_{i=1}^n w_i x_i \leq b$. En otras palabras dos conjuntos A y B en un espacio n -dimensional son linealmente separables si existe un hiperplano de tal manera que todos los puntos de A se encuentran de un mismo lado del hiperplano mientras que todos los puntos del conjunto B se encuentran del lado contrario del hiperplano.

La función XOR produce salida correcta ó 1 cuando sus dos entradas no coinciden, e incorrecta ó 0 cuando sus entradas coinciden, vea la tabla de verdad y su representación gráfica en la Figura 1.10. Como se puede observar en este caso no existe una recta que divida el plano en dos de tal modo que los dos puntos en azul (verdaderos) y los dos puntos en rojo (falsos) queden separados por medio de esta recta. Este problema de clasificación no se puede resolver mediante un Perceptrón simple pero sí empleando una red Perceptrón multicapa. Veamos la solución, tomemos una red Perceptrón con dos neuronas de entrada y una de salida, cuya topología y arquitectura se muestran en la Figura 1.11.

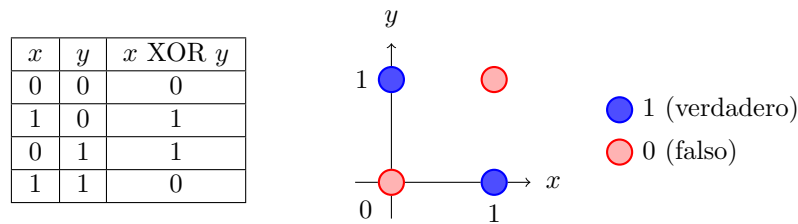


Figura 1.10: Tabla de verdad y representación gráfica de la función XOR.

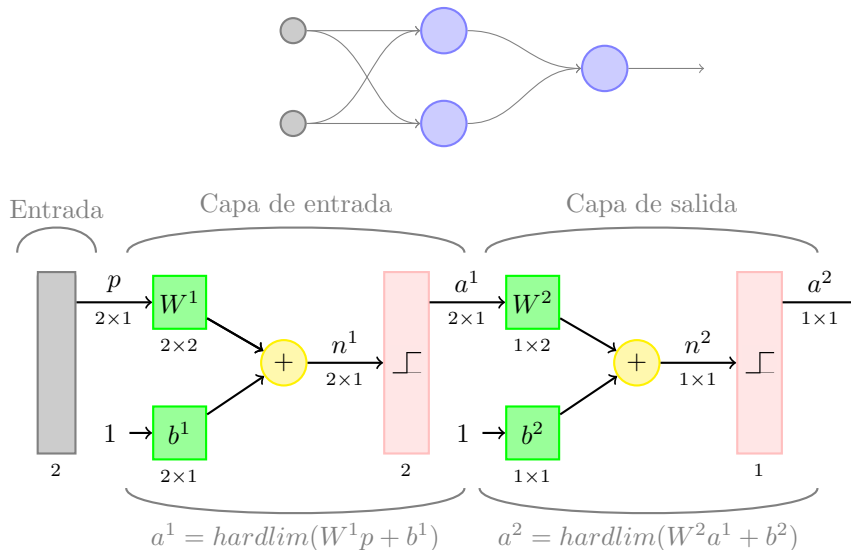


Figura 1.11: Topología y arquitectura del Perceptrón multicapa para la función XOR.

En este caso, se establecen los valores de los pesos y bias como:

$$W^1 = \begin{pmatrix} -0.6 & -0.6 \\ 2 & 2 \end{pmatrix}, \quad W^2 = (1 \ 1),$$

$$b^1 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}, \quad b^2 = (-1.5),$$

y se emplea la función de activación *hardlim* en ambas capas. Primero se calcula la entrada neta, n^1 de la capa de entrada, para el conjunto de patrones:

$$\begin{aligned} n^1 &= W^1 p + b^1 \\ &= \begin{pmatrix} -0.6 & -0.6 \\ 2 & 2 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0.4 & 0.4 & -0.2 \\ -1 & 1 & 1 & 3 \end{pmatrix}. \end{aligned}$$

Posteriormente se aplica la función de activación *hardlim* para obtener las salidas de la capa de entrada:

$$a^1 = \text{hardlim}(n^1) = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}.$$

De esta manera a^1 , que representa el estado de activación de las neuronas de la capa de entrada, se convierte en la entrada para la capa de salida.

Para calcular la salida de la red, primero se calcula la entrada neta n^2 de la capa de salida:

$$\begin{aligned} n^2 &= W^2 a^1 + b^2 \\ &= (1 \ 1) \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} + (-1.5 \ -1.5 \ -1.5 \ -1.5) \\ &= (-0.5 \ 0.5 \ 0.5 \ -0.5). \end{aligned}$$

Por último se aplica la función de activación a n^2 :

$$a = a^2 = \text{hardlim}(n^2) = (0 \ 1 \ 1 \ 0).$$

Se compara la salida de la red con la tabla de verdad del operador XOR, de la Figura 1.10 y se observa que se obtuvieron las salidas esperadas.

Capítulo 2

Aprendizaje

Desde el punto de vista biológico, el aprendizaje es el proceso en el que se adaptan las sinapsis para que la red responda de un modo distinto a los estímulos del medio.

En el contexto de RNA, el aprendizaje es un proceso mediante el cual la red modifica sus pesos en respuesta a una información de entrada. Durante el proceso de aprendizaje, los pesos de las conexiones de la red sufren modificaciones, por lo tanto se puede afirmar que este proceso ha terminado (la red ha aprendido) cuando los valores de los pesos permanecen estables [12].

Un aspecto importante respecto al aprendizaje en las redes neuronales es el conocer cómo se modifican los valores de los pesos, es decir; cuál es la regla de aprendizaje para cambiar el valor asignado a las conexiones (pesos) cuando se pretende que la red aprenda una nueva información [12].

Por lo tanto una clasificación de las RNA se realiza de acuerdo al tipo de aprendizaje utilizado. De esta manera se tienen: redes neuronales con aprendizaje supervisado y no supervisado.

2.1. Aprendizaje supervisado

En el aprendizaje supervisado, se presenta una entrada a la capa de entrada junto con una respuesta deseada para la capa de salida, el proceso de aprendizaje se basa en la comparación entre la salida que calculó la red y la respuesta deseada, generando un error, el error se utiliza para cambiar los parámetros de la red (pesos) de modo que resulte un mejor rendimiento [2, 31].

A continuación se darán algunos ejemplos de redes con aprendizaje supervisado.

2.1.1. Perceptrón

La red tipo Perceptrón es un modelo matemático de una neurona biológica propuesto por Frank Rosenblatt en 1959 en su obra *Principles of Neurodynamics*, los Perceptrones son redes de propagación hacia adelante basados en unidades binarias. El objetivo de esta red es aprender, usando parte de la información, de tal modo que siempre que se presente un patrón de entrada p la salida sea la salida esperada, digamos t .

Estructura de la red

La red Perceptrón consta de una capa de S neuronas conectadas a R entradas a través de una matriz de pesos W como se muestra en la Figura 2.1. W_{ij} es la intensidad de conexión de la j -ésima entrada a la i -ésima neurona [15].

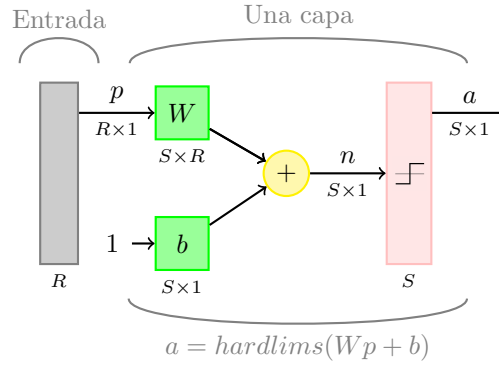


Figura 2.1: Arquitectura de la red Perceptrón.

En esta red pueden emplearse principalmente dos funciones de transferencia; *hardlim* con salidas 0 y 1 o *hardlims* con salidas -1 y 1; su uso depende del valor de salida que se espera para la red; sin embargo, la función *hardlims* es preferida sobre la *hardlim*, ya que en esta última al tener un cero multiplicando algunos de los valores resultantes del producto de las entradas por el vector de pesos, ocasiona que éstos no se actualicen y que el aprendizaje sea más lento.

Algoritmo de aprendizaje

Sea $P = \{p_1, p_2, \dots, p_Q\}$ un conjunto de entradas al que llamaremos conjunto de entrenamiento y $T = \{t_1, t_2, \dots, t_Q\}$ el conjunto de salidas esperadas, donde para el vector de entrada p_i tenemos que t_i es la salida deseada; ε el error mínimo aceptable. El algoritmo de entrenamiento del Perceptrón para una neurona con R elementos de entrada se puede resumir en los siguientes pasos:

1. Se inicializa la matriz de pesos $W(0)$ y el valor de la ganancia $b(0)$, (la notación $W(k)$ y $b(k)$, indican el valor de la matriz de pesos y el bias, respectivamente, en la k -ésima iteración), por lo general se asignan valores aleatorios pequeños. El valor del bias b puede tomarse como el valor de un peso adicional asociado con una entrada cuyo valor es 1.
2. Se presenta a la red un nuevo patrón, en forma de pares entrada-salida, $\{p_i, t_i\}$.
3. Se calcula la salida de la red para el vector de entrada p_i por medio de:

$$a_i = f(Wp_i + b)$$

donde f puede ser la función *hardlim* o *hardlims*.

4. Cuando la red no retorna la salida deseada, es necesario alterar el valor de los pesos, para minimizar el error y así aumentar las posibilidades de que la clasificación sea correcta, entonces usando el error, $e_i = t_i - a_i$, la modificación de los pesos y el bias se define del siguiente modo:

$$W(k+1) = W(k) + e_i p_i = W(k) + (t_i - a_i) p_i^T$$

$$b(k+1) = b(k) + e_i.$$

5. Volver al paso 2. Este proceso se repite hasta que $e = 0$ o bien $e < \varepsilon$ para cada uno de los Q patrones del conjunto de entrenamiento.

El Perceptrón será capaz de aprender a clasificar todas sus entradas, en un número finito de pasos, siempre y cuando el conjunto de los patrones de entrada sea linealmente separable [11, 12].

2.1.2. ADALINE

Al mismo tiempo que Frank Rosenblatt trabajaba en el modelo del Perceptrón, Bernard Widrow y su estudiante Marcian Hoff introdujeron el modelo de la red Adaline y su regla de aprendizaje llamada algoritmo LMS (Least Mean Square).

La red ADALINE es similar al Perceptrón, excepto en su función de transferencia, la cual es una función de tipo lineal en lugar de un limitador fuerte como en el caso del Perceptrón. Esta red presenta la misma limitación del Perceptrón, ambas redes sólo pueden resolver problemas linealmente separables, sin embargo, el algoritmo LMS es más potente que la regla de aprendizaje del Perceptrón ya que minimiza el error cuadrático medio, la regla sirvió de inspiración para el desarrollo de otros algoritmos, éste es el gran aporte de esta red [11].

El significado del término ADALINE cambió ligeramente a finales de los años sesenta cuando decayó el estudio de las redes neuronales, inicialmente se llamaba ADAptive LINEar NEuron (Neurona Lineal Adaptativa), para pasar después a ser ADAptive LINEar Element (Elemento Lineal Adaptativo). La red ADALINE consta de una única neurona, como tal no es técnicamente una red neuronal, sino una neurona.

La red ADALINE es adaptativa en el sentido de que existe un procedimiento bien definido para modificar los pesos con objeto de hacer posible que la red proporcione el valor de salida correcto para la entrada dada.

Estructura de la red

La arquitectura de la red ADALINE se muestra en la Figura 2.2, en donde se puede observar que la salida de la red está dada por:

$$a = \text{purelin}(Wp + b) = Wp + b.$$

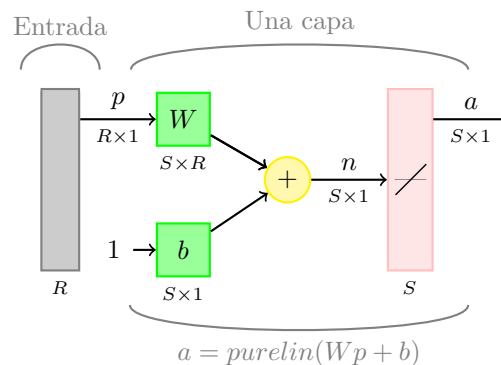


Figura 2.2: Arquitectura de la red ADALINE.

La aplicación más común de la red ADALINE es como filtro adaptativo, que sirve para eliminar el ruido de una señal de entrada. Por ejemplo cuando se diseña un filtro digital por medio de software con un programa normal, el programador debe saber exactamente como se especifica el algoritmo de filtrado y cuales son los detalles de las características de las señales; si se necesitan modificaciones, o si cambian las características de la señal, es necesario reprogramar. Cuando se emplea una red tipo ADALINE, ésta toma la entrada y la salida deseada, y se ajusta a sí misma para filtrar la señal de forma adecuada.

Algoritmo de aprendizaje

El algoritmo de aprendizaje de este tipo de redes toma como entradas un valor α denominado *tasa de aprendizaje*¹, ε el error mínimo aceptable, $P = \{p_1, p_2, \dots, p_Q\}$ el conjunto de entrada y $T = \{t_1, t_2, \dots, t_Q\}$ el conjunto de salidas deseadas. Una vez conocidos estos parámetros el algoritmo puede expresarse como el siguiente proceso iterativo:

1. Se inicializa la matriz de pesos $W(0)$ y el valor de la ganancia $b(0)$ en ceros.
2. Se presenta a la red un vector o patrón de entrada, p_i .
3. Se obtiene la salida lineal:

$$a_i = Wp_i + b$$

y se calcula la diferencia respecto a la salida deseada $e_i = (t_i - a_i)$.

4. Se actualizan los pesos:

$$W(k+1) = W(k) + 2\alpha e_i p_i^T = W(k) + 2\alpha(t_i - a_i)p_i^T.$$

5. Se repiten los pasos 2 al 4 con los Q vectores de entrada.
6. Si el error cuadrático medio es aceptable, es decir si:

$$E^2 = \frac{1}{Q} \sum_{k=1}^Q e_k^2 < \varepsilon$$

entonces termina el proceso de aprendizaje; en otro caso, se repite desde el paso 2.

2.1.3. Retropropagación

La red Retropropagación (Backpropagation) es un tipo de red con aprendizaje supervisado. Su funcionamiento consiste en el aprendizaje de un conjunto predefinido de pares entradas-salidas, empleando un ciclo de propagación-adaptación de dos fases: una vez que se ha aplicado un patrón a la entrada de la red como estímulo, éste se propaga desde la primera capa a través de las capas intermedias u ocultas de la red, hasta generar una salida. La señal de salida se compara con la salida deseada y se obtiene un error para cada una de las salidas. El error se propaga hacia atrás, partiendo de la capa de salida, pasa por las capas ocultas hasta llegar a la capa de entrada. Las neuronas de las capas ocultas y de entrada, sólo reciben una fracción del total del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite capa por capa, hasta que todas las neuronas de la red hayan recibido un error que describa su contribución relativa al error total. Basándose en el error percibido, se actualizan los pesos de conexión de cada neurona, para hacer que la red converja hacia un estado que permita clasificar correctamente todos los patrones de entrenamiento. La importancia de este proceso consiste en que, a medida que se entrena la red, las neuronas de las capas intermedias se organizan a sí mismas de tal modo que las distintas neuronas aprenden a reconocer distintas características del conjunto de entradas.

Después del entrenamiento, cuando se les presente un patrón arbitrario que contenga ruido o que esté incompleto, las neuronas de la capa oculta de la red responderán con una salida activa si el patrón tiene aquella característica que las neuronas individuales hayan aprendido a reconocer durante su entrenamiento. Y a la inversa, las unidades de las capas ocultas tienen una tendencia a inhibir su salida si el patrón no contiene la característica a reconocer, y para la cual han sido entrenadas. Esta característica importante que se exige a los sistemas de aprendizaje es la capacidad de *generalización*, entendida como la facilidad de dar salidas satisfactorias a entradas que a la red no se le han presentado nunca en su fase de entrenamiento [12].

¹Para mayores detalles sobre la elección de este parámetro puede consultar [11], Capítulo 10.

Varias investigaciones [11, 30] han demostrado que durante el proceso de entrenamiento, la red Retropropagación tiende a desarrollar relaciones internas entre neuronas con el fin de organizar los datos de entrenamiento en clases. Esta tendencia se puede extrapolar, para llegar a la hipótesis consistente de que todas las neuronas de las capas ocultas de una red tipo Retropropagación son asociadas a características específicas del patrón de entrada como consecuencia del entrenamiento.

Estructura de la red

La estructura típica de una red multicapa se ilustra en la Figura 2.3, en la cual se observa que la salida de la primera capa, es la entrada a la segunda y la salida de la segunda capa es la entrada a la tercera y así sucesivamente. Cada capa puede tener diferente número de neuronas, e incluso distinta función de transferencia siempre y cuando la derivada exista, por ejemplo, las funciones *tansig*, *logsig* y *purelin*.

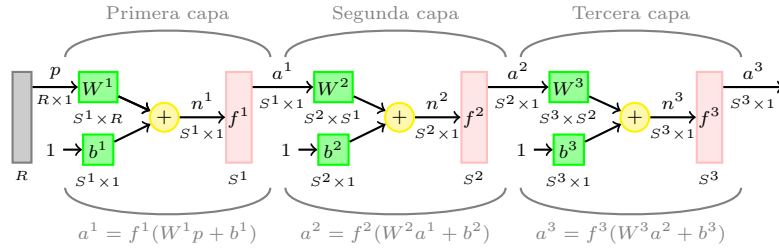


Figura 2.3: Arquitectura de una red Retropropagación de tres capas.

Algoritmo de aprendizaje

Dado el conjunto de entrenamiento $P = \{p_1, p_2, \dots, p_Q\}$ y el conjunto de salidas esperadas $T = \{t_1, t_2, \dots, t_Q\}$, M el número de capas, α la tasa de aprendizaje, ε el error mínimo aceptable y la derivada de la función de activación de cada capa, es decir, \dot{f}^k para $1 \leq k \leq M$ (k indica el número de capa, no el orden de la derivada) entonces el algoritmo de aprendizaje es el siguiente:

1. Se inicializa aleatoriamente la matriz de pesos $W(0)$ y el bias $b(0)$.
2. Se propaga la entrada hacia adelante en la red:

$$a^0 = p_i,$$

$$a^{m+1} = f^{m+1}(W^{m+1} a^m + b^{m+1})$$

para $m = 0, 1, \dots, M - 1$,

$$a = a^M.$$

3. Se definen y propagan hacia atrás las sensibilidades (s^i) de la red de la siguiente forma:

$$s^M = -2\dot{F}^M(n^M)(t - a)$$

$$s^m = \dot{F}^m(n^m)(W^{m+1})^T s^{m+1}$$

para $m = M - 1, \dots, 2, 1$ donde

$$\dot{F}^m(n^m) = \begin{pmatrix} \dot{f}^m(n_1^m) & 0 & \dots & 0 \\ 0 & \dot{f}^m(n_2^m) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \dot{f}^m(n_{s^m}^m) \end{pmatrix}$$

y n^m es la entrada neta de la m -ésima capa.

- Se actualizan los pesos y los bias usando alguna regla de aprendizaje para este tipo de red, por ejemplo, el gradiente descendente²:

$$W^m(k+1) = W^m(k) - \alpha s^m (a^{m-1})^T$$

$$b^m(k+1) = b^m(k) - \alpha s^m.$$

- Se repiten los pasos 2 a 4 con los Q vectores de entrada.
- Si el error E_P para el conjunto de entrenamiento P es aceptable:

$$E_P = \frac{1}{2} \sum_{k=1}^Q e_k^2 < \varepsilon$$

El proceso de aprendizaje concluye, de lo contrario se repite desde el paso 2.

Ejemplo

Se desea usar la red de la Figura 2.4 para aproximar la función XOR, considerando 0 como falso y 1 como verdadero, la cual tiene dos capas; dos neuronas en la capa de entrada con función de activación *logsig* y una neurona en la capa de salida con función de activación *purelin*. En este ejemplo se toma como tasa de aprendizaje $\alpha = 0.1$ y error mínimo aceptable $\varepsilon = 0.00001$.

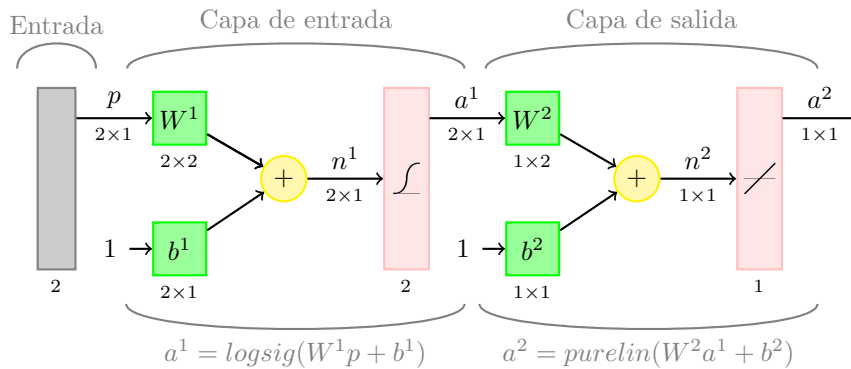


Figura 2.4: Arquitectura de la red Retropropagación para aproximar la función XOR.

Dado el conjunto de entrenamiento:

$$P = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\},$$

y el conjunto de salidas esperadas:

$$T = \{0, 1, 1, 0\}.$$

Los valores iniciales aleatorios para los pesos y bias de la primera y segunda capa, respectivamente, son:

$$W^1(0) = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 1 \end{pmatrix}, \quad b^1(0) = \begin{pmatrix} 0.4 \\ 0.2 \end{pmatrix},$$

$$W^2(0) = (1 \ 1), \quad b^2(0) = (0.6).$$

²No es la única regla de aprendizaje, para ver otras variantes puede consultar [11], Capítulo 12.

Como entrada inicial se toma³:

$$a_1^0 = p_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Se calcula la salida de la primera capa:

$$\begin{aligned} a_1^1 &= f^1(W^1 a_1^0 + b^1) \\ &= \text{logsig} \left(\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.4 \\ 0.2 \end{pmatrix} \right) \\ &= \text{logsig} \begin{pmatrix} 0.4 \\ 0.2 \end{pmatrix} \\ &= \begin{pmatrix} \frac{1}{1+e^{-0.4}} \\ \frac{1}{1+e^{-0.2}} \end{pmatrix} \\ &= \begin{pmatrix} 0.5987 \\ 0.5498 \end{pmatrix}. \end{aligned}$$

La salida de la segunda capa es:

$$\begin{aligned} a_1 = a_1^2 &= f^2(W^2 a_1^1 + b^2) \\ &= \text{purelin} \left((1 \ 1) \begin{pmatrix} 0.5987 \\ 0.5498 \end{pmatrix} + 0.6 \right) = 1.7485. \end{aligned}$$

El error es: $e_1 = t_1 - a_1 = 0 - 1.7485 = -1.7485$.

Para calcular las sensibilidades de la red, se emplearán las derivadas de las funciones de activación.

Las derivadas de las funciones logsig $f^1(n) = \frac{1}{1+e^{-n}}$ y purelin $f^2(n) = n$ son:

$$\dot{f}^1(n) = \frac{d}{dn} \left(\frac{1}{1+e^{-n}} \right) = \left(1 - \frac{1}{1+e^{-n}} \right) \left(\frac{1}{1+e^{-n}} \right) = (1 - a_1^1) (a_1^1)$$

$$\dot{f}^2(n) = \frac{d}{dn} (n) = 1.$$

Se calculan las sensibilidades, primero para la segunda capa y después para la primera capa:

$$\begin{aligned} s^2 &= -2\dot{F}^2(n^2)e_1 \\ &= -2(1)(-1.7485) = 3.4970 \\ s^1 &= \dot{F}^1(n^1)(W^2)^T s^2 \\ &= \begin{pmatrix} (1 - 0.5987)(0.5987) & 0 \\ 0 & (1 - 0.5498)(0.5498) \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} (3.4970) = \begin{pmatrix} 0.8402 \\ 0.8656 \end{pmatrix}. \end{aligned}$$

Se actualiza la matriz de pesos y el bias:

$$W^2(1) = W^2(0) - \alpha s^2 (a_1^1)^T = (1 \ 1) - 0.1(3.4972) (0.5987 \ 0.5498) = (0.7906 \ 0.8077)$$

$$b^2(1) = b^2(0) - \alpha s^2 = 0.6 - 0.1(3.4972) = 0.2503$$

$$W^1(1) = W^1(0) - \alpha s^1 (a_1^0)^T = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 1 \end{pmatrix} - 0.1 \begin{pmatrix} 0.8402 \\ 0.8656 \end{pmatrix} (0 \ 0) = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 1 \end{pmatrix}$$

³En este ejemplo la notación a_i^j , indica la salida de la red en la capa j , correspondiente al i -ésimo patrón de entrada y a^0 se puede considerar como la entrada.

$$b^1(1) = b^1(0) - \alpha s^1 = \begin{pmatrix} 0.4 \\ 0.2 \end{pmatrix} - 0.1 \begin{pmatrix} 0.8402 \\ 0.8656 \end{pmatrix} = \begin{pmatrix} 0.3160 \\ 0.1134 \end{pmatrix}.$$

Con estos pesos y bias actualizados, se toma otra entrada inicial del conjunto de entrenamiento:
Sea

$$a_2^0 = p_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Realizando el procedimiento anterior, la salida de la primera capa es:

$$a_2^1 = \text{logsig} \left(\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0.3160 \\ 0.1134 \end{pmatrix} \right) = \begin{pmatrix} 0.6934 \\ 0.7528 \end{pmatrix}.$$

La salida de la segunda capa es:

$$a_2 = a_2^2 = \text{purelin} \left(\begin{pmatrix} 0.7906 & 0.8077 \end{pmatrix} \begin{pmatrix} 0.6934 \\ 0.7528 \end{pmatrix} + 0.2503 \right) = 1.4065.$$

El error es : $e_2 = t_2 - a_2 = 1 - 1.4065 = -0.4065$.

Se calculan las sensibilidades:

$$\begin{aligned} s^2 &= -2(1)(-0.4065) = 0.8130 \\ s^1 &= \begin{pmatrix} (1 - 0.6934)(0.6934) & 0 \\ 0 & (1 - 0.7528)(0.7528) \end{pmatrix} \begin{pmatrix} 0.7906 \\ 0.8077 \end{pmatrix} (0.8130) = \begin{pmatrix} 0.1367 \\ 0.1222 \end{pmatrix}. \end{aligned}$$

Se actualiza la matriz de pesos y el bias

$$W^2(2) = \begin{pmatrix} 0.7906 & 0.8077 \end{pmatrix} - 0.1(0.8130) \begin{pmatrix} 0.6934 & 0.7528 \end{pmatrix} = \begin{pmatrix} 0.7342 & 0.7465 \end{pmatrix}$$

$$b^2(2) = 0.2503 - 0.1(0.8130) = 0.1690$$

$$W^1(2) = \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 1 \end{pmatrix} - 0.1 \begin{pmatrix} 0.1366 \\ 0.1222 \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.5 & 0.4863 \\ 0.5 & 0.9878 \end{pmatrix}$$

$$b^1(2) = \begin{pmatrix} 0.3160 \\ 0.1134 \end{pmatrix} - 0.1 \begin{pmatrix} 0.1366 \\ 0.1222 \end{pmatrix} = \begin{pmatrix} 0.3023 \\ 0.1012 \end{pmatrix}.$$

Se repite el procedimiento anterior, tomando como entrada inicial:

$$a_3^0 = p_3 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

y se obtienen los pesos y bias:

$$W^2(3) = \begin{pmatrix} 0.7124 & 0.7261 \end{pmatrix}, \quad b^2(3) = \begin{pmatrix} 0.1374 \end{pmatrix},$$

$$W^1(3) = \begin{pmatrix} 0.4950 & 0.4863 \\ 0.4946 & 0.9878 \end{pmatrix}, \quad b^1(3) = \begin{pmatrix} 0.2973 \\ 0.0958 \end{pmatrix}.$$

Tomando como entrada inicial el último patrón del conjunto de entrenamiento:

$$a_4^0 = p_4 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Se obtienen los pesos y bias:

$$\begin{aligned} W^2(4) &= (0.5096 \quad 0.5111), & b^2(4) &= -0.1219, \\ W^1(4) &= \begin{pmatrix} 0.4636 & 0.4549 \\ 0.4679 & 0.9611 \end{pmatrix} & b^1(4) &= \begin{pmatrix} 0.2659 \\ 0.0691 \end{pmatrix}. \end{aligned}$$

Puesto que se han presentado todos los patrones del conjunto de entrenamiento se calculará el error:

$$E_p = \frac{1}{2} \sum_{k=1}^Q e_k^2 = 2.4643.$$

Como $E_p > \varepsilon$, se repite el proceso anterior con todos los patrones del conjunto de entrenamiento, hasta que se cumpla $E_p < \varepsilon$.

La condición de paro se cumple con 1232 iteraciones cuando $E_p = 9.9364 \times 10^{-6} < \varepsilon$. Las matrices de pesos y bias correspondientes son:

$$\begin{aligned} W^1(1232) &= \begin{pmatrix} -3.9279 & 3.4724 \\ -1.9650 & 1.8642 \end{pmatrix}, & b^1(1232) &= \begin{pmatrix} -2.6266 \\ 0.5809 \end{pmatrix}, \\ W^2(1232) &= (2.7550 \quad -2.6684), & b^2(1232) &= (1.5281). \end{aligned}$$

Si se realiza la simulación de la red, empleando las matrices de pesos y los vectores bias obtenidos, se tiene como resultado:

$$T = \{(0.0027), (1.003), (0.9974), (4.494 \times 10^{-4})\}$$

la cual se aproxima a la salida esperada

$$T = \{0, 1, 1, 0\}$$

con un margen de error de ± 0.003 .

Capítulo 3

Aprendizaje no supervisado

Las redes neuronales artificiales con aprendizaje no supervisado son aquellas que no necesitan un asesor externo para realizar su aprendizaje. La red no recibe información por parte del entorno que le indique si la salida generada en respuesta a una entrada es o no correcta. El aprendizaje no supervisado consiste en que la red descubra por sí misma características, regularidades, correlaciones o categorías en los datos de entrada y se obtengan de forma codificada en la salida. En algunos casos, la salida representa el grado de similitud entre la información que se le está presentado en la entrada y la que se le ha mostrado en el pasado. En otro caso podría realizar un *clustering* o establecimiento de categorías, indicando con la salida de la red a qué categoría pertenece la información presentada como entrada, siendo la propia red quien deba encontrar las categorías apropiadas a partir de correlaciones en las entradas presentadas.

Es interesante conocer qué tipo de proceso puede realizar la red, qué problemas puede resolver, qué representa la salida de la red. Existen varias posibilidades [16]:

- **Análisis de las componentes principales.** Se trata de detectar cuáles de las componentes del conjunto de entrada caracterizan en mayor grado al conjunto de datos, de forma que las demás pudieran eliminarse sin pérdida significativa de información.
- **Agrupamiento.** A partir de un conjunto de entrada se desea conocer si hay un conjunto de datos que están bien representados por un patrón, y de qué manera se agrupan los datos a estos patrones representativos que ha encontrado la red. Es decir, si se puede dividir el conjunto de datos en diferentes clases, decidir a qué clase pertenece cada dato y qué caracteriza a cada una de las clases.
- **Prototipado.** Igual que en el caso anterior, pero en vez de obtenerse como resultado a qué clase pertenece el dato de entrada, se obtiene un prototipo o ejemplar de la clase al que pertenece dicho dato de entrada.
- **Codificación.** Se obtiene, a la salida de la red, una versión codificada del dato de entrada, es decir, un dato de menor dimensión que mantenga el máximo de información que le sea posible. Esto podría utilizarse para la compresión de datos antes de transmitirse por un canal de ancho de banda limitado, por el cual la señal original no podría ser transmitida, siempre y cuando exista la posibilidad de construir una red capaz de restaurar el dato original.
- **Extracción y relación de características.** La idea es realizar un mapa topológico de los datos de entrada, a través del diseño de la red, de tal forma que patrones de entrada parecidos, produzcan respuestas similares en neuronas cercanas. De esta forma se espera que si existe una organización global de los patrones de entrada, ésta sea observada en la salida.

Como ya se mencionó, en el aprendizaje no supervisado no existe ninguna información externa que indique si se están produciendo resultados erróneos, ni que ayude a decidir cómo y en qué grado se van a modificar las conexiones. En general se consideran dos criterios con los que se van a modificar las conexiones, que son [12, 16]:

- **Aprendizaje Hebbiano.** Este tipo de aprendizaje se basa en el postulado formulado por Donald Hebb en 1949 [11]:
Cuando un axón de la célula A está lo suficientemente cerca para excitar a una célula B, y toma parte en el proceso de activación de dicha célula, se produce algún tipo de cambio metabólico en una de las células (o en las dos), que hace que la eficacia con la que A activaba a B se vea incrementada. Esta regla de modificación sináptica no depende de ningún factor externo; simplemente hace que las células se refuerzen unas a otras, a partir de los estados obtenidos de las neuronas tras la presentación de cierto estímulo.
- **Aprendizaje competitivo.** En esta forma de aprendizaje las neuronas compiten unas con otras con el fin de llevar a cabo una tarea dada. Con este tipo de aprendizaje se pretende que cuando se presente a la red cierta información de entrada sólo una de las neuronas de salida, o una en un grupo de neuronas, se active, quedando como neurona vencedora. Este aprendizaje sólo modifica las conexiones de la neurona ganadora, por eso recibe el nombre de, *el que gana se lo lleva todo*, (en inglés *winner takes all*).

Algunas influencias en la historia del aprendizaje no supervisado fueron [6]: la regla de Hebb en 1949, la cual relaciona métodos estadísticos con experimentos neurofisiológicos sobre plasticidad; Donald MacKay en 1956, quien adoptó una aproximación teórica cibernética; David Marr en 1970, quien hizo un postulado de aprendizaje no supervisado acerca de la meta de aprendizaje en su modelo de la neocorteza; Geoffrey Hinton y Terrence Sejnowski en 1986 inventaron un modelo de aprendizaje llamado la Máquina de Boltzmann; Horace Barlow en 1992, quien sugirió maneras de caracterizar códigos neurales¹ [25].

Quizás la forma más novedosa de aprendizaje no supervisado en el campo de la Redes Neuronales es el mapa auto-organizativo de Kohonen (Self organizing feature map SOM). El aprendizaje competitivo es usado para una amplia variedad de campos bajo una amplia variedad de nombres, el más común es *cluster analysis*; la forma principal de aprendizaje competitivo es el vector quantization [32].

3.1. Red de Hopfield

Una de las mayores contribuciones en el área de las redes neuronales fue desarrollado por el físico John Hopfield en la década de los 80, quien propuso un modelo neuronal no lineal, conocido como la red de Hopfield [14]. La red de Hopfield es presentada como un modelo de memoria asociativa de patrones, en el sentido de que es capaz de recuperar patrones almacenados a partir de información incompleta sobre los patrones o incluso a partir de patrones con ruido [16].

3.1.1. Estructura de la red

El modelo original de Hopfield consiste en una red discreta de una sola capa con S neuronas cuyos valores de salida son binarios: -1 y 1, siendo las funciones de activación de tipo escalón. Posteriormente, Hopfield desarrolló una versión continua con entradas y salidas continuas y funciones de activación tipo sigmoideal [12]. En este trabajo sólo se tratará la versión discreta.

La red de Hopfield está formada por una sola capa de S neuronas, cada una se conecta a todas las demás salvo a ella misma, como se muestra en la Figura 3.1.

La matriz de pesos $W = W_{ij}$ de $S \times S$, posee las siguientes particularidades [16].

- Es una matriz simétrica, es decir $W_{ij} = W_{ji}$ para $i, j \in \{1, \dots, S\}$. Esto implica que el peso de conexión de la neurona i a la neurona j es igual al peso de conexión de la neurona j a la neurona i .
- Los elementos de la diagonal de la matriz son iguales a cero, es decir, $W_{ii} = 0$, para $i \in \{1, \dots, S\}$, debido a que en esta red no existen conexiones autorrecurrentes.

¹El código neural es la manera en que el sistema nervioso codifica la información.

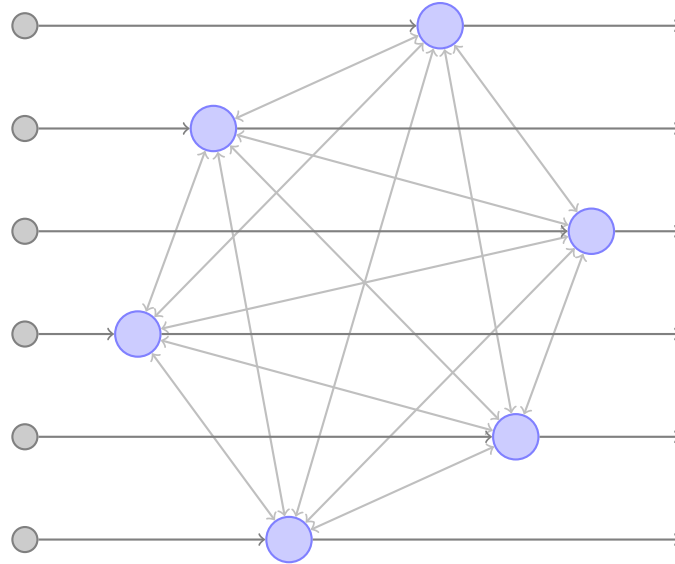


Figura 3.1: Topología de una red de Hopfield de 6 neuronas.

La red de Hopfield tiene un mecanismo de aprendizaje OFF LINE. Por lo tanto se distinguen dos fases de operación, llamadas fase de aprendizaje y fase de funcionamiento, o bien, fase de almacenamiento y fase de recuperación.

3.1.2. Algoritmo de aprendizaje y funcionamiento

Sean p_1, p_2, \dots, p_Q los patrones que la red debe aprender. Si cada vector consta de S entradas, entonces la red tendrá S neuronas.

ETAPA DE APRENDIZAJE:

Construir la matriz de pesos W a partir de los Q vectores de entrada:

$$W = \sum_{i=1}^Q (p_i^T p_i - I)$$

donde I es la matriz identidad que anula los pesos de las conexiones autorrecurrentes.

ETAPA DE FUNCIONAMIENTO:

1. La red recibe como entrada inicial un patrón o conjunto de patrones (vectores fila), digamos:

$$E = e_1, e_2, \dots, e_Q.$$

2. Se realiza la multiplicación de los vectores de entrada por la matriz de pesos W que se calculó en la fase de aprendizaje, y se aplica la función de transferencia f de tipo escalón²

$$a(1) = f(EW).$$

3. Se repite el paso 2. tomando como entrada la salida anterior, a_0 . Este proceso continúa hasta que las salidas de las neuronas se estabilizan, es decir, hasta que,

$$a(k) = a(k + 1)$$

²En esta red no se considera un bias.

La arquitectura de la red de Hopfield que permite un funcionamiento recurrente se puede observar en la Figura 3.2, donde el símbolo **D** significa delay o retraso, el cual posibilita justamente la recurrencia de la red.

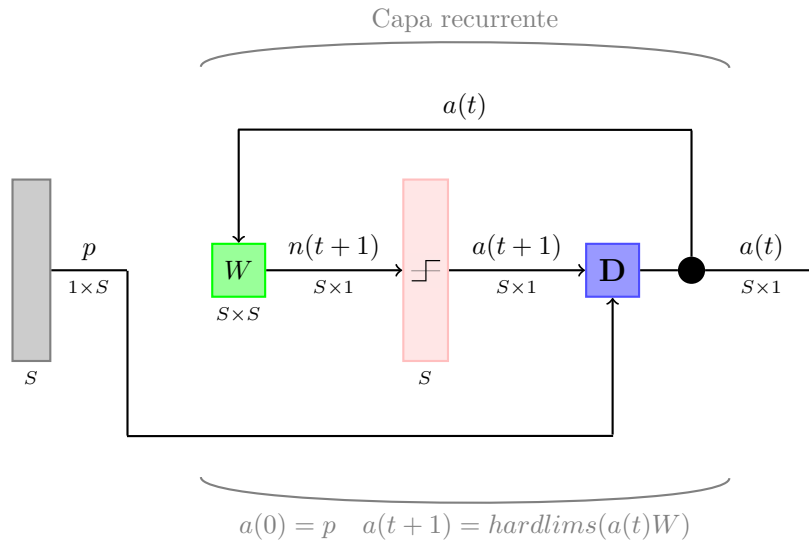


Figura 3.2: Arquitectura de la red de Hopfield.

3.1.3. Ejemplo

En este ejemplo se diseñará una red de Hopfield para el reconocimiento de dos patrones de cuatro entradas.

Sean $p_1 = (1 \ 1 \ -1 \ -1)$ y $p_2 = (-1 \ -1 \ 1 \ 1)$ los dos patrones que la red debe aprender.

Ya que son dos patrones, $Q = 2$ y como cada patrón tiene 4 entradas, la red tendrá 4 neuronas para que cada una reciba el valor de una entrada.

ETAPA DE APRENDIZAJE:

Se construye la matriz de pesos W :

$$\begin{aligned}
 W &= \sum_{i=1}^2 (p_i^T p_i - I) \\
 &= \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix} (1 \ 1 \ -1 \ -1) + \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \end{pmatrix} (1 \ 1 \ -1 \ -1) - 2 \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
 &= \begin{pmatrix} 0 & 2 & -2 & -2 \\ 2 & 0 & -2 & -2 \\ -2 & -2 & 0 & 2 \\ -2 & -2 & 2 & 0 \end{pmatrix}.
 \end{aligned}$$

Una vez finalizada la fase de aprendizaje ya puede ser utilizada como memoria asociativa, de tal forma que al presentarle un nuevo patrón iterará hasta generar como salida el patrón más parecido a los almacenados durante el aprendizaje.

ETAPA DE FUNCIONAMIENTO:

1. Sea $p = (1 \ -1 \ -1 \ -1)$.
2. La salida de la red después de la primera iteración es:

$$\begin{aligned} a(1) &= f \left((1 \ -1 \ -1 \ -1) \begin{pmatrix} 0 & 2 & -2 & -2 \\ 2 & 0 & -2 & -2 \\ -2 & -2 & 0 & 2 \\ -2 & -2 & 2 & 0 \end{pmatrix} \right) \\ &= \text{hardlims}(2 \ 6 \ -2 \ -2) \\ &= (1 \ 1 \ -1 \ -1). \end{aligned}$$

3. Se repite el proceso anterior tomando como entrada la salida $a(1)$

$$\begin{aligned} a(2) &= \text{hardlims} \left((1 \ 1 \ -1 \ -1) \begin{pmatrix} 0 & 2 & -2 & -2 \\ 2 & 0 & -2 & -2 \\ -2 & -2 & 0 & 2 \\ -2 & -2 & 2 & 0 \end{pmatrix} \right) \\ &= (1 \ 1 \ -1 \ -1). \end{aligned}$$

Se observa que se repite la salida de la primera iteración, por lo que se ha llegado a una situación de estabilidad, en la que la red ha generado como salida el patrón más parecido al presentado como entrada; en este caso estaría asociando a p con p_1 .

3.2. Mapas Auto-Organizativos de Kohonen

Existen evidencias que demuestran que en el cerebro hay neuronas que se organizan en muchas zonas, de forma que las informaciones captadas del entorno a través de los órganos sensoriales se representan internamente en forma de mapas bidimensionales. Por ejemplo, en el sistema visual se han detectado mapas del espacio visual en zonas del córtex (capa externa del cerebro), también en el sistema auditivo se detecta una organización según la frecuencia a la que cada neurona alcanza mayor respuesta (organización tonotópica) [12].

Esto sugiere, por lo tanto, que el cerebro podría poseer la capacidad inherente de formar *mapas topológicos* de las informaciones recibidas del exterior.

A partir de estas ideas, Teuvo Kohonen presentó en 1982 un modelo de red neuronal con capacidad para formar *mapas de características* de manera similar a como ocurre en el cerebro. El objetivo de Kohonen era *demostrar que un estímulo externo por sí solo, suponiendo una estructura propia y una descripción funcional del comportamiento de la red, era suficiente para forzar la formación de los mapas*;[12].

Este modelo tiene dos variantes, denominadas LVQ (Learning Vector Quantization) y SOM (Self Organizing Map). En este trabajo primero se presentará la red SOM.

El aprendizaje en el modelo de Kohonen es de tipo OFF-LINE, por lo que se distingue una etapa de aprendizaje y otra de funcionamiento. Esta red utiliza un aprendizaje no supervisado de tipo competitivo, en el que las neuronas compiten por activarse y sólo una de ellas permanece activa ante una determinada información de entrada a la red. De esta manera los pesos de las conexiones se ajustan en función de la neurona que haya resultado vencedora.

Durante la etapa de entrenamiento, se presenta a la red un conjunto de entrenamiento (vectores de entrenamiento) para que ésta establezca, en función de la semejanza entre los datos, las diferentes categorías (una por neurona), que servirán durante la fase de funcionamiento para realizar clasificaciones de nuevos datos que se presenten a la red. En el caso de existir más patrones de entrenamiento que neuronas, más de uno deberá asociarse con la misma neurona, es decir pertenecerán a la misma clase.

En este modelo el aprendizaje no concluye después de presentarle una vez todos los patrones de entrada, sino que habrá que repetir el proceso varias veces para refinar el mapa topológico de salida, de tal forma que cuantas más veces se presenten los datos, tanto más se reducirán las zonas de neuronas que se deben activar ante entradas parecidas, consiguiendo que la red pueda realizar una clasificación más selectiva.

3.2.1. Estructura de la red

Una red SOM (Self Organizing Map) es una red neuronal monocapa. La capa es a la vez de entrada y de salida por lo tanto es la encargada de recibir y procesar la información procedente del exterior y formar el mapa topológico³. Normalmente, las neuronas de la capa de salida se organizan en forma de mapa bidimensional como se muestra en la Figura 3.3.

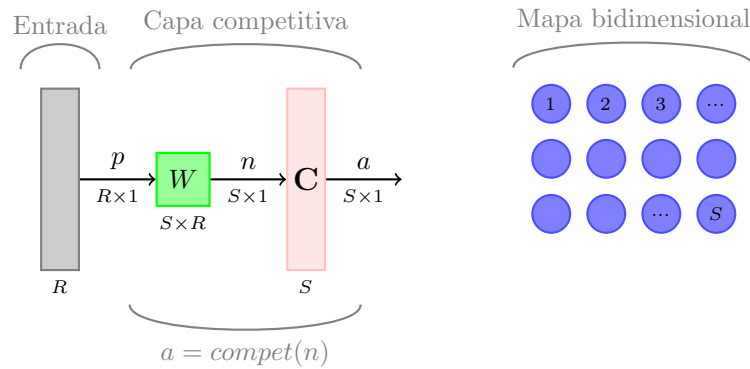


Figura 3.3: Red SOM.

La función de transferencia competitiva produce un 1 para el elemento de salida a_{i^*} correspondiente a la neurona ganadora i^* ; las otras entradas del vector de salida a son cero.

Un concepto muy importante en la red de Kohonen es la zona de vecindad, o vecindario alrededor de la neurona vencedora i^* ; la *vecindad* $N_{i^*}(d)$ (vea [11]) contiene los índices de todas las neuronas que están dentro de un radio d de la neurona ganadora i^* es decir:

$$N_{i^*}(d) = \{j, d_{i^*j} \leq d\}.$$

Los pesos de las neuronas que se encuentren en esta zona, $N_{i^*}(d)$, serán actualizados junto con el peso de la neurona ganadora.

Para ilustrar el concepto de vecindad, considérense los diagramas mostrados en la Figura 3.4. El primer diagrama muestra la vecindad de la neurona 13 con un radio $d = 1$ y el segundo diagrama muestra la vecindad de la misma neurona ahora con un radio $d = 2$ si se emplea la distancia rectilínea o Manhattan.

³Un mapa topológico es un mapeo que preserva relaciones de vecindad.

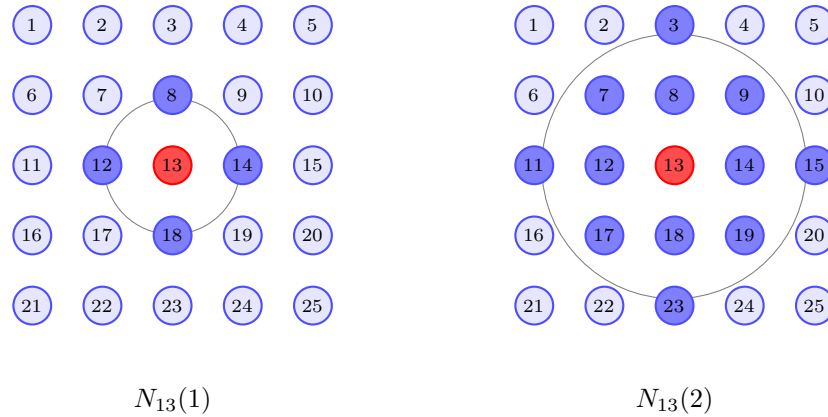


Figura 3.4: Vecindades.

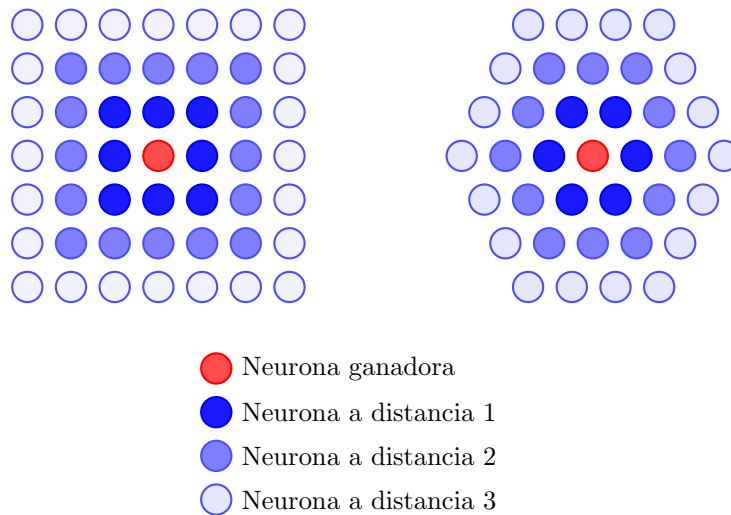


Figura 3.5: Topología rectangular (izq.) y hexagonal (der.).

Como se puede observar:

$N_{13}(1) = \{8, 12, 13, 14, 18\}$ y $N_{13}(2) = \{3, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18, 19, 23\}$, así por ejemplo, si la neurona 13 fuera la vencedora y se fija la distancia en 1 se actualizarán los pesos de las neuronas en $N_{13}(1)$.

Es posible elegir diferentes topologías de vecindades, por ejemplo, Kohonen sugirió vecindades rectangulares y hexagonales, véase la Figura 3.5. De la misma manera, se pueden elegir diferentes definiciones de distancia para calcular qué neuronas están cercanas a la neurona ganadora [15, 11].

3.2.2. Algoritmo de aprendizaje

Sea $P = \{p_1, p_2, \dots, p_Q\}$ el conjunto de entrenamiento, α la tasa de aprendizaje, N el número de neuronas en la capa de entrada y M el número de neuronas en la capa de salida. El algoritmo de aprendizaje para establecer los valores de los pesos de las conexiones es el siguiente [11, 12, 20]:

1. Seleccionar la topología de la red para determinar que nodos son adyacentes a otros.
2. Fijar el número k de iteraciones, generalmente se elige $k \geq 500$.
3. Inicializar la distancia d y la matriz de pesos W con valores aleatorios pequeños.
4. Seleccionar un patrón de entrada p_i y calcular la distancia entre p_i y los pesos W_j asociados a cada neurona de salida, usando, por ejemplo, el cuadrado de la distancia Euclídea (otra puede ser la distancia Euclídea):

$$d^2 = \sum_{j=1}^N (p_{ij} - W_j(k))^2.$$

5. Elegir la neurona ganadora i^* ; será aquella que tenga el valor mínimo obtenido en el paso anterior.
6. Se actualizan los pesos de la neurona vencedora, así como los pesos de las neuronas vecinas

$$W_i(k+1) = W_i(k) + \alpha(k)(p_i - W_i(k)), \text{ para } i \in N_{i^*}(d).$$

La tasa de aprendizaje α , decrece con el número de iteraciones: $0 < \alpha(k+1) \leq \alpha(k) \leq 1$.

En la elección de este parámetro suele utilizarse alguna de las siguientes expresiones: $\alpha(k) = \frac{1}{k}$, o bien, $\alpha(k) = \alpha_1 \left(1 - \frac{k}{\alpha_2}\right)$, donde $\alpha_1 = 0.1$ ó $\alpha_1 = 0.2$ y α_2 igual a un valor próximo al número de iteraciones.

7. Se repite el proceso a partir del paso 4, presentando todos los patrones del conjunto de entrenamiento, hasta completar el número de iteraciones dado en el paso 2.

3.2.3. Ejemplo

Sea $P = \{(1, 1, 0, 0), (0, 0, 0, 1), (1, 0, 0, 0), (0, 0, 1, 1)\}$, el conjunto de entrenamiento, $R = 4$ el número de neuronas de entrada, $S = 2$ el número de neuronas en la capa, $\alpha = 0.6$ la tasa de aprendizaje, $k = 500$ el número de iteraciones; la topología de la red que resuelve este ejemplo es como se muestra en la Figura 3.6, de tal modo que las neuronas no se consideran vecinas.

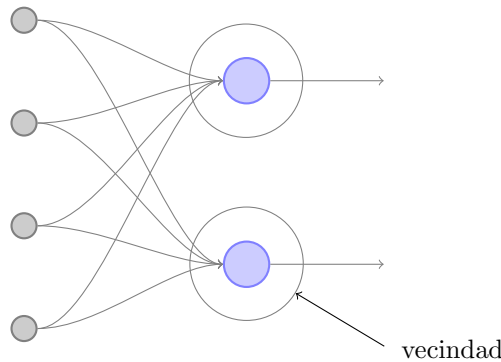


Figura 3.6: Topología de la red SOM para el ejemplo de la sección 3.2.3.

Se inicializa la matriz de pesos:

$$W(0) = \begin{pmatrix} 0.2 & 0.6 & 0.5 & 0.9 \\ 0.8 & 0.4 & 0.7 & 0.3 \end{pmatrix}.$$

Donde los elementos del primer renglón, llamémoslo W_1 , son los pesos que se conectan con la neurona 1, y el segundo renglón W_2 corresponde a los pesos de conexión entre las entradas y la neurona 2.

Se entrena la red con el primer vector de entrada $p_1 = (1, 1, 0, 0)$, para ello se calcula el cuadrado de la distancia Euclídea entre la entrada y los pesos correspondientes a cada neurona.

$$d^2(p_1, W_1) = (0.2 - 1)^2 + (0.6 - 1)^2 + (0.5 - 0)^2 + (0.9 - 0)^2 = 1.86$$

$$d^2(p_1, W_2) = (0.8 - 1)^2 + (0.4 - 1)^2 + (0.7 - 0)^2 + (0.3 - 0)^2 = 0.98.$$

Como la distancia de la neurona 2 es menor, la segunda neurona es la ganadora. A continuación se actualizan los pesos correspondientes a la neurona ganadora únicamente (ya que las vecindades no incluyen otras neuronas), es decir, los pesos del segundo renglón, así:

$$\begin{aligned} W_2(1) &= (0.2 \ 0.6 \ 0.5 \ 0.9) + 0.6 \left((1 \ 1 \ 0 \ 0) - (0.8 \ 0.4 \ 0.7 \ 0.3) \right) \\ &= (0.92 \ 0.76 \ 0.28 \ 0.12). \end{aligned}$$

De esta manera la matriz de pesos W actualizada, es:

$$W(1) = \begin{pmatrix} 0.2 & 0.6 & 0.5 & 0.9 \\ 0.92 & 0.76 & 0.28 & 0.12 \end{pmatrix}.$$

Se entrena la red usando el segundo vector de entrada, $p_2 = (0, 0, 0, 1)$. Se calcula el cuadrado de la distancia Euclídea entre el vector de entrada p_2 y los pesos de conexión que van a la neurona 1:

$$d^2(p_2, W_1) = (0.2 - 0)^2 + (0.6 - 0)^2 + (0.5 - 0)^2 + (0.9 - 1)^2 = 0.66$$

$$d^2(p_2, W_2) = (0.92 - 0)^2 + (0.76 - 0)^2 + (0.28 - 0)^2 + (0.12 - 1)^2 = 2.28.$$

Se elige la distancia menor, lo que significa que la neurona ganadora es la neurona 1. Por lo tanto se actualizan los pesos correspondientes a la neurona 1:

$$\begin{aligned} W_1(2) &= (0.2 \ 0.6 \ 0.5 \ 0.9) + 0.6 \left((0 \ 0 \ 0 \ 1) - (0.2 \ 0.6 \ 0.5 \ 0.9) \right) \\ &= (0.08 \ 0.24 \ 0.20 \ 0.96). \end{aligned}$$

Así la matriz de pesos W es:

$$W(2) = \begin{pmatrix} 0.08 & 0.24 & 0.20 & 0.96 \\ 0.92 & 0.76 & 0.28 & 0.12 \end{pmatrix}.$$

Se repite el proceso anterior para el tercero y cuarto vector de entrada, p_3 y p_4 , respectivamente. Después de 500 iteraciones la matriz de pesos es⁴:

$$W(500) = \begin{pmatrix} 0 & 0 & 0.7 & 1 \\ 1 & 0.3 & 0 & 0 \end{pmatrix}.$$

En seguida se realiza la simulación de la red empleando la matriz de pesos que se obtuvo en el proceso de entrenamiento, y se observa cómo clasifica los patrones.

Para $p_1 = (1, 1, 0, 0)$

Se calcula la distancia entre los pesos de la neurona 1 y p_1 :

$$d^2(p_1, W_1) = (0 - 1)^2 + (0 - 1)^2 + (0.7 - 0)^2 + (1 - 0)^2 = 3.49.$$

Se calcula la distancia entre los pesos de la neurona 2 y p_1 :

$$d^2(p_1, W_2) = (1 - 1)^2 + (0.3 - 1)^2 + (0 - 0)^2 + (0 - 0)^2 = 0.49.$$

La neurona 2 es la ganadora.

⁴Desde la iteración 5 la matriz de pesos ya no cambia, debido a que es un ejemplo con un conjunto de entrenamiento muy pequeño.

Para $p_2 = (0, 0, 0, 1)$.

Se calcula la distancia entre los pesos de la neurona 1 y p_2 :

$$d^2(p_2, W_1) = (0 - 0)^2 + (0 - 0)^2 + (0.7 - 0)^2 + (1 - 1)^2 = 0.49.$$

Se calcula la distancia entre los pesos de la neurona 2 y p_2 :

$$d^2(p_2, W_2) = (1 - 0)^2 + (0.3 - 0)^2 + (0 - 0)^2 + (0 - 1)^2 = 2.09.$$

Gana la neurona 1.

Si se realizan las mismas operaciones para los patrones p_3 y p_4 , se tiene que: a los patrones p_1 y p_3 los clasifica con la neurona 2, y a los patrones p_2 y p_4 los clasifica con la neurona 1.

Si se presenta un nuevo patrón a la red, por ejemplo $p_5 = (1, 1, 1, 0)$, que es “muy parecido” a p_1 , se espera que la neurona ganadora sea la neurona 2. Se calcula la distancia desde los pesos de la neurona 1 y p_5 :

$$d^2(p_5, W_1) = (0 - 1)^2 + (0 - 1)^2 + (0.7 - 1)^2 + (1 - 0)^2 = 3.09.$$

Se calcula la distancia desde los pesos de la neurona 2 y p_5 :

$$d^2(p_5, W_2) = (1 - 1)^2 + (0.3 - 1)^2 + (0 - 1)^2 + (0 - 0)^2 = 1.49.$$

Ganó la neurona 2. Lo cual significa que la red clasifica de manera adecuada.

3.3. Learning Vector Quantization

La red Learning Vector Quantization (LVQ), emplea tanto aprendizaje supervisado como no supervisado. Es una red bicapa en la que cada neurona de la primera capa es asignada a una clase, con varias neuronas generalmente asignadas a la misma clase. En la segunda capa cada clase es asignada a una neurona. El número de neuronas en la primera capa, S^1 , debe ser mayor o al menos igual que el número de neuronas en la segunda capa, S^2 [11], si se identifican N subclases, entonces, S^1 debe contener al menos N neuronas.

Al igual que con redes competitivas, cada neurona en la primera capa de la red LVQ aprende un vector prototipo, el cual permite a la neurona clasificar una región del espacio de entrada, sin embargo en lugar de calcular la distancia entre la entrada y el vector de pesos por medio del producto punto, la red LVQ calcula la distancia directamente. Una ventaja de hacer el cálculo de la distancia directamente, es que los vectores no necesitan ser normalizados; cuando los vectores son normalizados la respuesta de la red será la misma sin importar la técnica que se utilice.

3.3.1. Estructura de la red

La estructura de la red LVQ se muestra en la Figura 3.7. La entrada neta a la primera capa de la red es

$$n_i^1 = - \| W_i^1 - p^T \|,$$

donde $W_i^1 = (W_{i1}^1 \ W_{i2}^1 \ \dots \ W_{iS}^1)$, S es el número de neuronas en la capa de entrada y p^T es el transpuesto de p . La entrada neta se puede expresar en forma de vector como:

$$n^1 = - \begin{pmatrix} \| W_1^1 - p^T \| \\ \| W_2^1 - p^T \| \\ \vdots \\ \| W_{S^1}^1 - p^T \| \end{pmatrix}$$

y la salida de la primera capa de la red LVQ es $a^1 = \text{compet}(n^1)$.

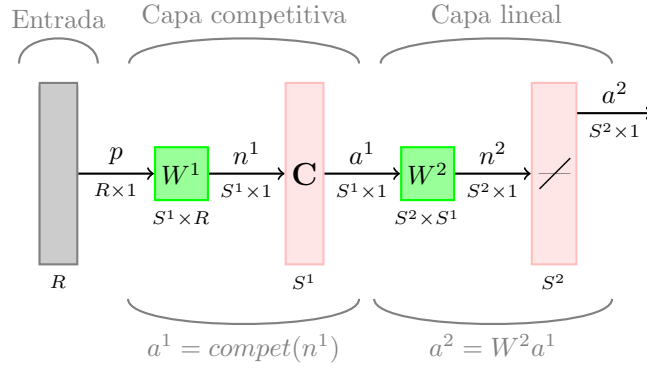


Figura 3.7: Estructura de la red LVQ.

Así, la neurona cuyo vector de pesos esté cercano al vector de entrada tendrá salida 1 y las otras neuronas tendrán salida 0; en este aspecto la red LVQ se comporta igual a las redes competitivas, la única diferencia consiste en la interpretación, mientras que en las redes competitivas la salida no cero representa una clase del vector de entrada, para la red LVQ, indica más bien una subclase, y de esta forma muchas neuronas (subclases), conforman una clase.

La segunda capa de la red LVQ es usada para combinar subclases en una sola clase; esto es realizado por la matriz de pesos W^2 . Las columnas de W^2 representan subclases y los renglones representan clases. W^2 tiene un solo 1 en cada columna, los demás elementos son cero. La fila en la cual se presenta el 1 indica cual es la clase a la que la subclase pertenece, es decir:

$$(W_{ki}^2 = 1) \implies \text{la subclase } i \text{ pertenece a la clase } k.$$

Una propiedad importante de esta red es que permite crear clases complejas combinando subclases. Una capa competitiva estándar tiene la limitación de que sólo puede crear regiones de decisión convexas⁵, del mismo modo que una red Perceptrón: con una capa sólo puede establecer dos regiones separadas por un hiperplano en el espacio de entrada de los patrones, con dos capas dependiendo del número de neuronas, puede formar regiones convexas en el espacio mediante la intersección de las regiones delimitadas por las neuronas de la primera capa, con tres o más niveles prácticamente se pueden separar regiones arbitrarias como se puede observar en la Tabla 3.1. En el caso de las redes competitivas la red LVQ al igual que el Perceptrón multicapa soluciona esta limitación.

Estructura	Regiones de decisión	Ejemplos
Una capa	Medio espacio limitado por un hiperplano	
Dos capas	Regiones cerradas o convexas	
Tres capas	Arbitraria complejidad limitada por el número de neuronas	

Tabla 3.1: Distintas formas de regiones.

⁵Dados dos puntos cualesquiera de una región si el segmento de recta que los une queda contenido en la región diremos que esta es convexa.

Debido a que la red LVQ combina aprendizaje competitivo con aprendizaje supervisado, necesita un conjunto de entrenamiento acompañado de las salidas esperadas, es decir parejas de la forma $\{p_1, t_1\}, \{p_2, t_2\}, \dots, \{p_Q, t_Q\}$, que describan el comportamiento de la red.

3.3.2. Algoritmo de aprendizaje

Antes de que suceda el aprendizaje, cada neurona en la segunda capa es asignada a una neurona de salida, así se genera la matriz de pesos W^2 ; por lo general, igual número de neuronas ocultas son conectadas a cada neurona de salida, para que cada clase pueda ser conformada por el mismo número de regiones convexas. En otros casos esta asignación es proporcional a la cantidad de patrones que pertenecen a esa clase respecto al total, pero esto implicaría aportar conocimiento del medio exterior.

Todos los elementos de W^2 son cero excepto los que cumplan la siguiente condición:

$$\text{Si la neurona } i \text{ es asignada a la clase } k \implies W_{ki}^2 = 1.$$

Una vez que W^2 ha sido definida, nunca será alterada. Los pesos W^1 son actualizados por medio de la regla de Kohonen.

El algoritmo de aprendizaje de la red LVQ, procede de la siguiente manera:

Dados el conjunto de entrenamiento y el de salidas esperadas de la red $P = \{p_1, p_2, \dots, p_Q\}$ y $T = \{t_1, t_2, \dots, t_Q\}$, α la tasa de aprendizaje y ε el error mínimo aceptable:

1. En cada iteración, se presenta un vector de entrada p_i a la red y se calcula la distancia de la matriz de pesos al vector prototipo.

$$n^1 = - \begin{pmatrix} \left\| W_1^1 - p_i^T \right\| \\ \left\| W_2^1 - p_i^T \right\| \\ \vdots \\ \left\| W_{S^1}^1 - p_i^T \right\| \end{pmatrix}.$$

2. Las neuronas ocultas compiten; si la neurona ganadora es i^* entonces el i^* -ésimo elemento de a^1 se fija en 1 y los demás en cero, es decir.

$$a^1 = \text{compet}(n^1).$$

3. Se multiplica a^1 por W^2 para obtener la salida final a^2 , la cual tiene solamente un elemento no cero, k^* , indicando que el patrón p_i está siendo asignado a la clase k^* .
4. Se emplea la regla de Kohonen⁶ para mejorar la capa oculta de la red, en dos formas:

- a) Si p_i es clasificado correctamente, los pesos de la neurona ganadora oculta $W_{i^*}^1$ se hacen tender hacia p_i

$$W_{i^*}^1(q) = W_{i^*}^1(q-1) + \alpha(p_i^T - W_{i^*}^1(q-1)), \text{ si } a_{k^*}^2 = t_{k^*} = 1.$$

Aquí q es el número de iteraciones, puesto que k representa la clase.

- b) Si p_i fue clasificado incorrectamente, entonces la neurona oculta incorrecta ganó la competencia y por lo tanto sus pesos $w_{i^*}^1$ se alejan de p_i

$$W_{i^*}^1(q) = W_{i^*}^1(q-1) - \alpha(p_i^T - W_{i^*}^1(q-1)), \text{ si } a_{k^*}^2 = 1 \neq t_{k^*} = 0.$$

⁶No es la única regla de aprendizaje, en el toolbox de redes neuronales de MATLAB se considera una variante.

5. Se repite el proceso anterior hasta que la media del error sea menor o igual a ε .

El algoritmo de aprendizaje irá presentando a la red todos los patrones del conjunto de entrenamiento P . El resultado será que los valores de las conexiones de cada neurona de entrada se acercarán a los vectores que cayeron dentro de la clase, para la cual ellos forman una subclase y se alejarán de los vectores que cayeron en otras clases.

3.3.3. Ejemplo

Se ilustrará el funcionamiento de la red LVQ, buscando que clasifique correctamente los siguientes patrones, cuyas clases se han definido arbitrariamente:

$$clase1 : \left\{ p_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, p_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\}, clase2 : \left\{ p_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, p_4 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}.$$

Los vectores esperados asociados a cada una de las entradas son:

$$\left\{ p_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, t_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\}, \left\{ p_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, t_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\}$$

$$\left\{ p_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, t_3 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}, \left\{ p_4 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, t_4 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}.$$

Si se escogen dos subclases para cada una de las dos clases existentes, entonces tendremos cuatro subclases, lo que determina que debe haber cuatro neuronas en la primera y la matriz de pesos para la capa de salida es:

$$W^2 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}.$$

La topología y arquitectura de la red que resuelven este ejemplo se muestran en las Figuras 3.8 y 3.9.

Se inicializa $W^1(0)$ con valores aleatorios, de la siguiente forma:

$$\begin{aligned} W_1^1(0) &= (0.25 \quad 0.75) & W_2^1(0) &= (0.75 \quad 0.75) \\ W_3^1(0) &= (1 \quad 0.25) & W_4^1(0) &= (0.5 \quad 0.25). \end{aligned}$$

Sea $\alpha = 0.5$ la tasa de aprendizaje y $\varepsilon = 0$.

En cada iteración del proceso de aprendizaje se presenta un vector de entrada, se calcula la respuesta, y por último se ajustan los pesos.

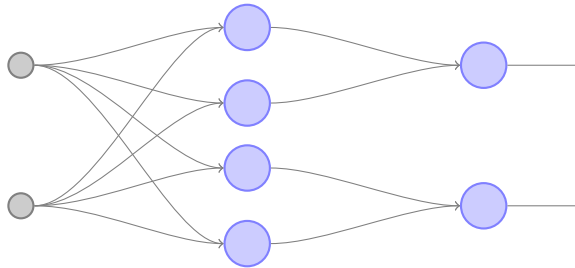


Figura 3.8: Topología de la red LVQ del ejemplo de la sección 3.3.3.

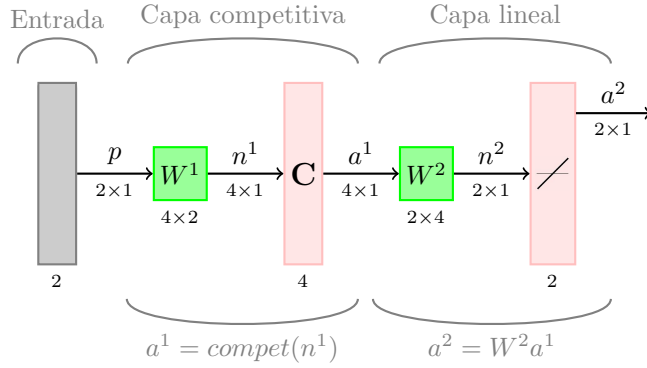


Figura 3.9: Arquitectura de la red del ejemplo de la sección 3.3.3.

En este ejemplo se empezará presentando $p_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$,

Se calcula la distancia entre el vector de entrada y las matrices de pesos:

$$\begin{aligned}
 a_1^1 &= \text{compet}(n^1) = \text{compet} \begin{pmatrix} -\|W_1^1 - p_1^T\| \\ -\|W_2^1 - p_1^T\| \\ -\|W_3^1 - p_1^T\| \\ -\|W_4^1 - p_1^T\| \end{pmatrix} \\
 &= \text{compet} \begin{pmatrix} -\| \begin{pmatrix} 0.25 & 0.75 \end{pmatrix} - \begin{pmatrix} 0 & 1 \end{pmatrix} \| \\ -\| \begin{pmatrix} 0.75 & 0.75 \end{pmatrix} - \begin{pmatrix} 0 & 1 \end{pmatrix} \| \\ -\| \begin{pmatrix} 1.00 & 0.25 \end{pmatrix} - \begin{pmatrix} 0 & 1 \end{pmatrix} \| \\ -\| \begin{pmatrix} 0.50 & 0.25 \end{pmatrix} - \begin{pmatrix} 0 & 1 \end{pmatrix} \| \end{pmatrix} = \text{compet} \begin{pmatrix} -0.3536 \\ -0.7906 \\ -1.2500 \\ -0.9014 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.
 \end{aligned}$$

El resultado indica que la primera neurona de la capa de entrada es la más cercana al vector p_1 . Se calculará la salida de la red, la cual indicará a qué clase pertenece el vector de entrada p_1

$$a_1 = a_1^2 = W^2 a_1^1 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

La salida de la red, indica que el vector p_1 , pertenece a la clase 1, lo cual es correcto. Se actualizará la matriz de pesos correspondiente a ese vector:

$$\begin{aligned}
 W_{1^*}^1(1) &= W_{1^*}^1(0) + 0.5(p_1^T - W_{1^*}^1(0)) \\
 &= \begin{pmatrix} 0.25 & 0.75 \end{pmatrix} + 0.5 \left(\begin{pmatrix} 0 & 1 \end{pmatrix} - \begin{pmatrix} 0.25 & 0.75 \end{pmatrix} \right) \\
 &= \begin{pmatrix} 0.125 & 0.875 \end{pmatrix}.
 \end{aligned}$$

Se toma la siguiente entrada y se realiza el proceso anterior:

$$p_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Se calcula la distancia entre el vector y la matriz de pesos,.

$$a_2^1 = \text{compet} \begin{pmatrix} -\| \begin{pmatrix} 0.125 & 0.875 \end{pmatrix} - \begin{pmatrix} 1 & 0 \end{pmatrix} \| \\ -\| \begin{pmatrix} 0.750 & 0.750 \end{pmatrix} - \begin{pmatrix} 1 & 0 \end{pmatrix} \| \\ -\| \begin{pmatrix} 1.000 & 0.250 \end{pmatrix} - \begin{pmatrix} 1 & 0 \end{pmatrix} \| \\ -\| \begin{pmatrix} 0.500 & 0.250 \end{pmatrix} - \begin{pmatrix} 1 & 0 \end{pmatrix} \| \end{pmatrix} = \text{compet} \begin{pmatrix} -1.2374 \\ -0.7906 \\ -0.2500 \\ -0.5590 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}.$$

El resultado indica que la tercera neurona oculta es la más cercana al vector p_2 , para saber a que clase pertenece se realiza:

$$a_2 = a_2^2 = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

La salida de la red indica que el vector p_2 , pertenece a la clase 2, lo cual es incorrecto, por lo tanto se actualizan los pesos de la siguiente forma:

$$\begin{aligned} W_{3^*}^1(1) &= W_{3^*}^1(0) - 0.5(p_2^T - W_{3^*}^1(0)) \\ &= (1 \quad 0.25) - 0.5((1 \quad 0) - (1 \quad 0.25)) \\ &= (1 \quad 0.375). \end{aligned}$$

Se presenta el siguiente vector de entrada a la red:

$$p_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

y se obtiene la salida $a_3 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, la cual es incorrecta, y se actualizan los pesos.

Se presenta el último vector de entrada a la red, y se repite el procedimiento para:

$$p_4 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Se obtiene la salida $a_4 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, la cual es correcta, y se actualizan los pesos.

Las matrices actualizadas después de aplicar todos los patrones del conjunto de entrenamiento a la red son:

$$\begin{aligned} W_1^1(1) &= (0.125 \quad 0.875) & W_2^1(1) &= (0.625 \quad 0.625) \\ W_3^1(1) &= (1 \quad 0.375) & W_4^1(1) &= (0.25 \quad 0.125). \end{aligned}$$

De esta manera se tiene que:

$$W^1(1) = \begin{pmatrix} 0.125 & 0.875 \\ 0.625 & 0.625 \\ 1 & 0.375 \\ 0.25 & 0.125 \end{pmatrix}.$$

Después de 16 iteraciones la matriz de pesos con la cual se obtiene el resultado correcto es:

$$W^1(16) = \begin{pmatrix} 0.0078 & 0.9922 \\ 0.7188 & 0.2188 \\ 1 & 0.8906 \\ 0.0313 & 0.0313 \end{pmatrix}.$$

Capítulo 4

Algoritmo Wake-Sleep

Recientes experimentos y avances teóricos, sugieren que los recuerdos se pueden reorganizar en la corteza cerebral durante el sueño. Las representaciones de objetos y eventos están ampliamente distribuidas en la corteza cerebral; por ejemplo, la representación de un violín puede estar almacenada en diversas áreas como: la corteza visual, para la forma, la corteza auditiva, para el sonido, la corteza parietal, para la forma en que puede ser sostenido y la corteza motora, para la forma en que se toca [29].

Los problemas surgen cuando las nuevas experiencias deben integrarse con la información existente que se encuentra ampliamente distribuida. El cerebro debe resolver dos problemas durante el aprendizaje: dónde hacer los cambios necesarios para crear una nueva memoria y cómo hacer los cambios para que sean compatibles con los recuerdos almacenados previamente.

Las redes neuronales con aprendizaje supervisado [13] presentan dos problemas: requieren de un maestro que especifique la salida deseada de la red y requieren de algún método para comunicar el error a todas las conexiones. El algoritmo wake-sleep evita ambos problemas. Cuando no existe una señal externa para comparar, se requiere otro objetivo para forzar a las neuronas ocultas a extraer la estructura subyacente. En el algoritmo wake-sleep la meta es aprender patrones que permitan reconstruir la entrada con precisión.

Hinton et. al. [29, 13] han proporcionado un marco teórico para la creación de representaciones jerárquicas eficientes de memoria en los modelos de redes neuronales: la máquina de Helmholtz [7].

La máquina de Helmholtz tiene conexiones de “reconocimiento” de abajo hacia arriba, que se emplean para convertir las entradas sensoriales en representaciones. Las conexiones “generativas” de arriba hacia abajo, se emplean para reconstruir las entradas sensoriales a partir de las representaciones.

En la fase *wake* del algoritmo de aprendizaje, la red o la máquina de Helmholtz, está guiada por las conexiones o pesos de reconocimiento y se modifican los pesos generativos para incrementar la probabilidad de tener una mejor reconstrucción de la entrada sensorial. En la fase *sleep* la red es guiada por las conexiones generativas, en esta fase se modifican las conexiones de reconocimiento para producir una mejor representación fantasmiosa [9]. En ambos casos se usará la regla delta, también conocida como la regla del gradiente descendente, para actualizar los pesos.

Por una curiosa coincidencia, la idea de que el sistema perceptivo utiliza modelos generativos fue defendido por Helmholtz, así que a cualquier red neural que se ajuste a un modelo generativo de datos, se le llama una máquina de Helmholtz.

4.1. Estructura de la red

La red neuronal conocida como máquina de Helmholtz tiene dos conjuntos de conexiones diferentes. Las conexiones de reconocimiento (de abajo hacia arriba) se usan para convertir el vector de entrada en una

representación a través de una o más capas ocultas. Las conexiones generativas (de arriba hacia abajo) se usan para reconstruir una aproximación del vector de entrada como se muestra en la Figura 4.1. El algoritmo de aprendizaje para esos dos conjuntos de conexiones se puede emplear para diferentes tipos de neuronas estocásticas¹; en este trabajo se emplearán solamente neuronas estocásticas binarias cuyos estados de activación son 1 ó 0.

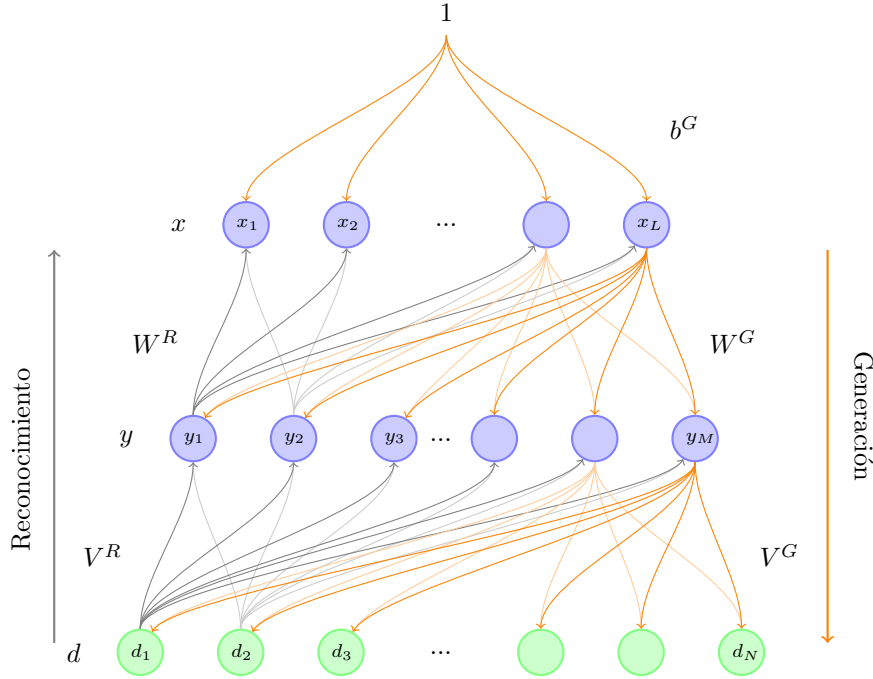


Figura 4.1: Topología de una red tipo máquina de Helmholtz.

La entrada neta de una neurona, en la fase wake está dada por:

$$n_{y_j} = \sum_{i=1}^N V_{ji}^R d_i + b_j$$

$$n_{x_k} = \sum_{j=1}^M W_{kj}^R y_j + b_k$$

donde n_{y_j} es la entrada neta de la neurona y_j y n_{x_k} es la entrada neta de la neurona x_k , V^R y W^R son las matrices de pesos de reconocimiento, particularmente, V_{ji}^R es el peso de conexión de reconocimiento de la neurona i a la neurona j , d_i es el estado de activación de la neurona i en la capa anterior.

En forma matricial, la entrada neta para las neuronas de la capa oculta y de salida respectivamente, está dada por:

$$n_y = V^R d + b$$

$$n_x = W^R y + b.$$

Similarmente, la entrada neta de una neurona, en la fase sleep es:

$$n_{x_k} = b_k^G$$

$$n_{y_j} = \sum_{k=1}^L W_{jk}^G x_k + b_j$$

$$n_{d_i} = \sum_{j=1}^M V_{ij}^G y_j + b_i$$

¹Una neurona estocástica es aquella cuya función de activación depende de una variable aleatoria.

donde b^G , W^G y V^G son las matrices de pesos generativos, por ejemplo W_{jk}^G es el peso de conexión generativo de la neurona k a la neurona j , x_k es el estado de activación de la neurona k en la capa anterior, b_j es el bias de la neurona j .

Matricialmente:

$$\begin{aligned}n_x &= b^G \\n_y &= W^G x + b \\n_d &= V^G y + b.\end{aligned}$$

La probabilidad de que una neurona u con entrada neta n se active depende de la composición de las funciones de transferencia *logsig* y *sample*.

$$a_u = \text{sample}(\text{logsig}(n)).$$

La función *sample* se define de la siguiente manera:

$$\text{sample}(x) = \begin{cases} 1 & \text{con probabilidad } p \\ 0 & \text{con probabilidad } 1 - p \end{cases}$$

Algunos conceptos básicos de probabilidad

Se describirán algunos conceptos de probabilidad, que se emplearán en las siguientes secciones:

- Cuando se asigna una probabilidad a un par de vectores binarios, $p : \{0, 1\}^N \times \{0, 1\}^M \rightarrow [0, 1]$, se escribe $p(xy)$ en lugar de $p(x, y)$
- La distribución de probabilidad conjunta de dos vectores binarios aleatorios X y Y , es $p(xy) = \text{Prob}[X = x \text{ y } Y = y]$.
- La probabilidad condicional de x dado y es $p(x|y) = p(xy)/p(y)$.
- Las variables aleatorias X y Y son independientes si $p(xy) = p(x)p(y)$.
- Si X y Y son independientes entonces $p(x|y) = p(x)$.
- Un vector binario aleatorio D con distribución $p(d)$, se puede describir como una distribución conjunta sobre todas las N posibles combinaciones en $d : p(d) = p(d_1 d_2 \dots d_N)$.
- El vector binario aleatorio D es independiente si $p(d_1 d_2 \dots d_N) = p(d_1)p(d_2) \dots p(d_N)$
- Una variable aleatoria es idénticamente distribuida si existe alguna constante de probabilidad p_0 , tal que para todo $i = 1, 2, \dots, N$:

$$p(d_i) = \begin{cases} p_0 & \text{cuando } d_i = 1 \\ 1 - p_0 & \text{cuando } d_i = 0 \end{cases}$$

- Gracias a la propiedad de ser independiente e idénticamente distribuido, se tiene que la probabilidad de un vector binario específico, d , se puede reescribir de la siguiente forma:

$$p(d) = \prod_i p_0^{d_i} (1 - p_0)^{1-d_i}.$$

- Dadas dos distribuciones de probabilidad $p_A(d)$ y $p_B(d)$, una manera de cuantificar que tan diferentes son es empleando la divergencia de Kullback-Leibler, también conocida como la entropía relativa de A a B :

$$KL[p_A(D), p_B(D)] = \sum_d p_A(d) \log \frac{p_A(d)}{p_B(d)}.$$

El proceso generativo

El conjunto de entrenamiento está compuesto de patrones binarios, donde cada patrón d tiene asignada alguna probabilidad $p(d)$. El objetivo de este proceso es la generación de un patrón d , empleando la cadena $1 \rightarrow x \rightarrow y \rightarrow d$. Con la misma probabilidad que tiene asignada originalmente. Dadas las conexiones generativas es posible conocer con qué probabilidad aparecerá cierto patrón x , luego se puede calcular la distribución de probabilidad generativa de x , $p_G(x)$, usando la matriz de pesos b^G . Similarmente, $p_G(y|x)$ está determinada por la matriz de pesos W^G y $p_G(d|y)$ está determinada por la matriz de pesos V^G . Ya que las neuronas en cada capa son condicionalmente independientes dado el estado de activación de las neuronas en la capa anterior se tiene que:

- $p_G(x) = \prod_k p_G(x_k)^{x_k} [1 - p_G(x_k)]^{1-x_k}$

donde $p_G(x_k) = \text{logsig}(n_{x_k}) = \text{logsig}(b_k^G)$.

- $p_G(y|x) = \prod_j p_G(y_j|x)^{y_j} [1 - p_G(y_j|x)]^{1-y_j}$

donde $p_G(y_j|x) = \text{logsig}(n_{y_j}) = \text{logsig}(\sum_{k=1}^L W_{jk}^G x_k + b_j)$.

- $p_G(d|y) = \prod_i p_G(d_i|y)^{d_i} [1 - p_G(d_i|y)]^{1-d_i}$

donde $p_G(d_i|y) = \text{logsig}(n_{d_i}) = \text{logsig}(\sum_{j=1}^M V_{ij}^G y_j + b_i)$.

El proceso de reconocimiento

En el proceso generativo se crea una fantasía² y se consideran a los patrones x y y como los generadores de d . La tarea para el proceso de reconocimiento es tomar un nuevo patrón d y reportar los estados de x y y que lo generaron. Para obtener la distribución de probabilidad de reconocimiento $p_R(xy|d)$, al igual que en el proceso de generación se tiene independencia condicional de x y y dado d , lo cual por definición es:

$$p_R(xy|d) = p_R(x|y)p_R(y|d).$$

Los factores $p_R(x|y)$ y $p_R(y|d)$, están determinados por las conexiones ascendentes en la máquina de Helmholtz, llamados los pesos de reconocimiento V^R y W^R .

Las ecuaciones para las probabilidades de los pesos de reconocimiento son:

- $p_R(x|y) = \prod_k p_R(x_k|y)^{x_k} [1 - p_R(x_k|y)]^{1-x_k}$

donde $p_R(x_k|y) = \text{logsig}(n_{x_k}) = \text{logsig}(\sum_{j=1}^M W_{kj}^R y_j + b_k)$.

²En el proceso de reconocimiento a partir de una imagen (entrada) se obtiene una salida, en el proceso generativo a partir de una constante se genera una imagen, dado que esta fase también se conoce como la fase *sleep* a la imagen la llamamos sueño o fantasía.

$$\blacksquare p_R(y|d) = \prod_k p_R(y_j|d)^{y_j} [1 - p_R(y_j|d)]^{1-y_j}$$

$$\text{donde } p_R(y_j|d) = \text{logsig}(n_{y_j}) = \text{logsig}(\sum_{i=1}^N V_{ji}^R d_i + b_j).$$

4.2. Algoritmo de aprendizaje

Una máquina de Helmholtz puede aprender la estructura del conjunto de entrenamiento, es decir, aprender su distribución de probabilidad como se demuestra en [6, 7]. Para esto, la idea es usar el método del gradiente descendente que involucrará dos fases, una modificará los pesos generativos y otra los pesos de reconocimiento.

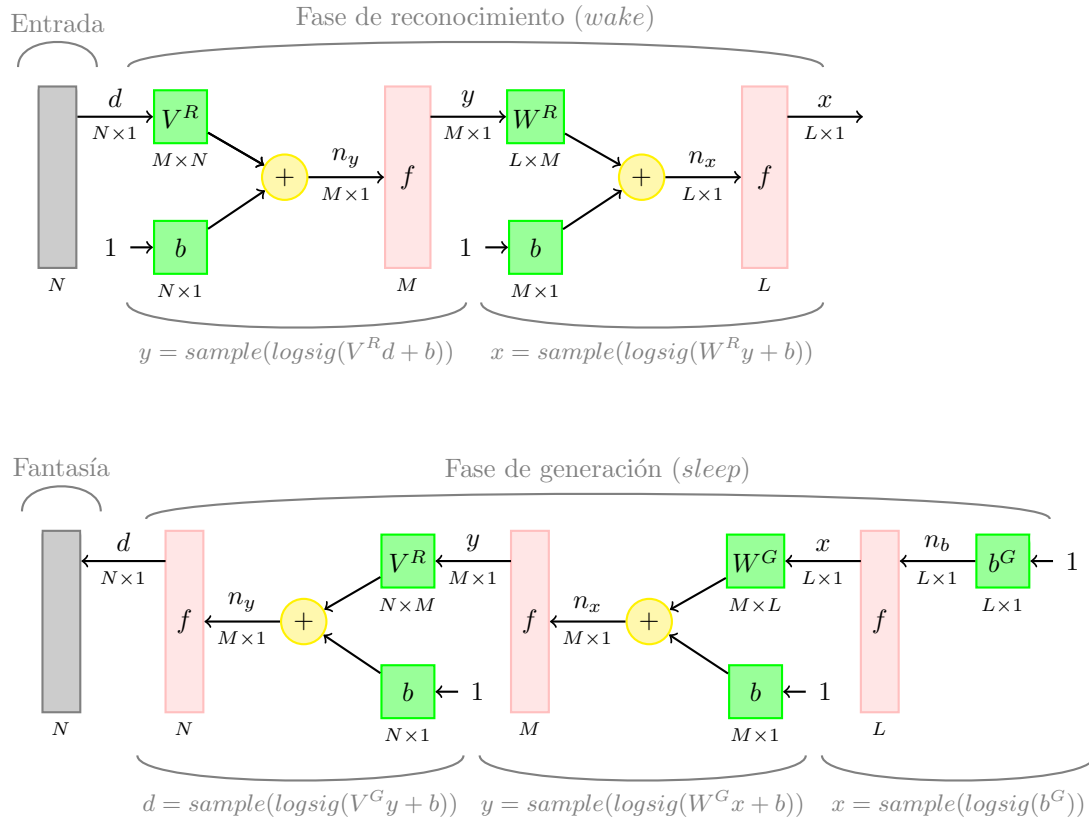


Figura 4.2: Arquitectura de la máquina de Helmholtz.

En la Figura 4.2 se puede observar la arquitectura de la máquina de Helmholtz y el proceso de aprendizaje. Primero se muestra el proceso de la fase de reconocimiento, en la cual se presenta el vector de entrada d , se obtiene la salida de la primera capa y aplicando la composición de funciones sample y logsig a la multiplicación del vector de entrada d con la matriz de pesos V^R más el bias; la salida de la primera capa es a la vez entrada para la segunda capa, realizando un procedimiento análogo se obtiene x .

En la segunda parte se muestra la fase de generación, en esta fase se parte de una constante igual a 1, se multiplica con la matriz de pesos b^G y se aplica la composición de funciones para obtener el estado de activación de x , luego y y por último la fantasía d . Veamos detalladamente el algoritmo de aprendizaje.

Dado el conjunto de entrenamiento D , con una distribución de probabilidad $p(D)$, una divergencia mínima aceptable α y $\varepsilon_V, \varepsilon_W, \varepsilon_b$, las tasas de aprendizaje:

1. Se inicializan las matrices de pesos de reconocimiento (W^R, V^R) y de generación (W^G, V^G, b^G) en cero.

2. Fase wake

- a) Obtener una muestra d del conjunto de entrenamiento D , de acuerdo a la distribución $p(D)$ y realizar el proceso de reconocimiento.

$$y = \text{sample}(\text{logsig}(V^R d + b))$$

$$x = \text{sample}(\text{logsig}(W^R y + b)).$$

- b) Se ajustan los pesos generativos, utilizando la regla delta³:

$$b^G(q) = b^G(q-1) + \varepsilon_b(x - \xi)$$

$$W^G(q) = W^G(q-1) + \varepsilon_W(x - \psi)(x + b)^T$$

$$V^G(q) = V^G(q-1) + \varepsilon_V(d - \delta)(y + b)^T.$$

Donde:

$$\xi = \text{logsig}(b^G)$$

$$\psi = \text{logsig}(W^G x + b)$$

$$\delta = \text{logsig}(V^G y + b).$$

3. Fase sleep

- a) Se transmite la señal de sueño a través de las conexiones de generación, empezando con una constante igual a 1:

$$x = \text{sample}(\text{logsig}(b^G * 1))$$

$$y = \text{sample}(\text{logsig}(W^G x + b))$$

$$d = \text{sample}(\text{logsig}(V^G y + b)).$$

- b) Se ajustan los pesos de reconocimiento, utilizando la regla delta

$$V^R(q) = V^R(q-1) + \varepsilon_V(y - \psi)(d + b)^T$$

$$W^R(q) = W^R(q-1) + \varepsilon_W(x - \xi)(y + b)^T.$$

Donde:

$$\psi = \text{logsig}(V^R d + b)$$

$$\xi = \text{logsig}(W^R y + b).$$

4. Repetir los pasos 2 y 3 hasta que la divergencia entre la distribución de probabilidad y la distribución de probabilidad generativa sea menor que el mínimo aceptable, es decir :

$$KL[p(D), p_G(D)] = \sum_d p(d) \log \frac{p(d)}{p_G(d)} < \alpha.$$

³La demostración de que el gradiente, de la función de "error" tiene la forma especificada se da basándose en la teoría física conocida como mecánica estadística, interpretando el "error" como energía libre, por lo que queda fuera de los alcances de este trabajo. Para mayor información consultar las referencias [6, 8, 17].

4.3. Ejemplo

Para ilustrar el funcionamiento del algoritmo wake-sleep, se empleará una máquina de Helmholtz para aproximar la función XOR. Sea $d \in \{0, 1\}^2$, existen 4 posibles patrones en el conjunto de entrenamiento. Se asignarán probabilidades a cada patrón: 0.5 para el patrón cuya respuesta sea verdadera y 0 para el patrón cuya respuesta sea falsa como se muestra en la Tabla 4.1.

d	$p(d)$
[0,0]	0
[1,0]	0.5
[0,1]	0.5
[1,1]	0

Tabla 4.1: Probabilidades asignadas a los patrones del ejemplo 4.3.

Se usará una máquina de Helmholtz con dos neuronas en la capa de entrada, dos en la segunda y una neurona en la tercera.

Se establecen las tasas de aprendizaje: $\varepsilon_V = 0.15$, $\varepsilon_W = 0.01$, $\varepsilon_b = 0.01$, la divergencia mínima aceptable $\alpha = 0.1$ y se procede al entrenamiento.

1. Se inicializan las matrices de pesos, tanto de reconocimiento como de generación:

$$V^G(0) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad W^G(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad b^G(0) = (0),$$

$$V^R(0) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}, \quad W^R(0) = (0 \ 0).$$

2. Fase wake.

Se toma una muestra del conjunto de entrenamiento:

$$d = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Y se hace fluir el patrón d a través de los pesos de reconocimiento:

$$y = \text{sample} \left(\text{logsig} \left(\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right) \right) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$x = \text{sample} \left(\text{logsig} \left((0 \ 0) \begin{pmatrix} 1 \\ 1 \end{pmatrix} + (1) \right) \right) = (1).$$

Se calculan los valores de ξ , ψ , δ , que se emplean para la modificación de los pesos:

$$\xi = \text{logsig}(b^G) = \text{logsig}(0) = 0.5$$

$$\psi = \text{logsig} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix} (1) + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) = \begin{pmatrix} 0.7311 \\ 0.7311 \end{pmatrix}$$

$$\delta = \text{logsig} \left(\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) = \begin{pmatrix} 0.7311 \\ 0.7311 \end{pmatrix}.$$

Se actualizan los pesos usando la regla delta:

$$b^G(1) = 0 + 0.01(1 - 0.5) = 0.005$$

$$W^G(1) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + 0.01 \left(\begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 0.7311 \\ 0.7311 \end{pmatrix} \right) (1+1)^T = \begin{pmatrix} 0.0054 \\ 0.0054 \end{pmatrix}$$

$$\begin{aligned} V^G(1) &= \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + 0.15 \left(\begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.7311 \\ 0.7311 \end{pmatrix} \right) \left(\begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)^T \\ &= \begin{pmatrix} 0.0807 & 0.0807 \\ -0.2193 & -0.2193 \end{pmatrix}. \end{aligned}$$

3. Fase sleep.

Se genera un sueño o fantasía usando los nuevos pesos generativos.

$$x = \text{sample}(\text{logsig}(0.005)) = 1$$

$$y = \text{sample} \left(\text{logsig} \left(\begin{pmatrix} 0.0054 \\ 0.0054 \end{pmatrix} 1 + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) \right) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$d = \text{sample} \left(\text{logsig} \left(\begin{pmatrix} 0.0807 & 0.0807 \\ -0.2193 & -0.2193 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) \right) = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Se calculan ψ y ξ :

$$\psi = \text{logsig} \left(\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) = \begin{pmatrix} 0.7311 \\ 0.7311 \end{pmatrix}$$

$$\xi = \text{logsig} \left(\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right) = \begin{pmatrix} 0.7311 \\ 0.7311 \end{pmatrix}.$$

Se actualizan pesos:

$$\begin{aligned} V^R(1) &= \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + 0.15 \left(\begin{pmatrix} 1 \\ 0 \end{pmatrix} - \begin{pmatrix} 0.7311 \\ 0.7311 \end{pmatrix} \right) \left(\begin{pmatrix} 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)^T \\ &= \begin{pmatrix} 0.0807 & 0.0807 \\ -0.2193 & -0.2193 \end{pmatrix} \end{aligned}$$

$$\begin{aligned} W^R(1) &= \begin{pmatrix} 0 & 0 \end{pmatrix} + 0.01 (1 - 0.7311) \left(\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right)^T \\ &= \begin{pmatrix} -0.0146 & -0.0146 \end{pmatrix}. \end{aligned}$$

4. Se calcula la distribución de probabilidad generativa, utilizando las ecuaciones:

$$p_G(xyd) = p_G(d|y)p_G(y|x)p_G(x) \quad \text{y} \quad p_d = \sum_{xy} p_G(xyd)$$

y se obtienen los resultados que se muestran en la Tabla 4.2.

d	$p(d)$	$p_G(d)$
[0,0]	0	0.1083
[1,0]	0.5	0.1549
[0,1]	0.5	0.3040
[1,1]	0	0.4328

Tabla 4.2: Distribución de probabilidad generativa después de una iteración.

5. Se calcula la divergencia entre $p(d)$ y $p_G(d)$

$$KL[p(d), p_G(d)] = 0.8347.$$

6. Como no se ha alcanzado la condición de paro, se repite el proceso anterior.

Después de 50,000 iteraciones se alcanza la condición de paro. La distribución de probabilidad generativa se muestra en la Tabla 4.3 y la gráfica en la Figura 4.3 muestra la divergencia entre $p(d)$ y $p_G(d)$ cada 1,000 iteraciones.

d	$p(d)$	$p_G(d)$
[0,0]	0	0.0470
[1,0]	0.5	0.4810
[0,1]	0.5	0.3773
[1,1]	0	0.0948

Tabla 4.3: Probabilidades generativas con 50,000 iteraciones.

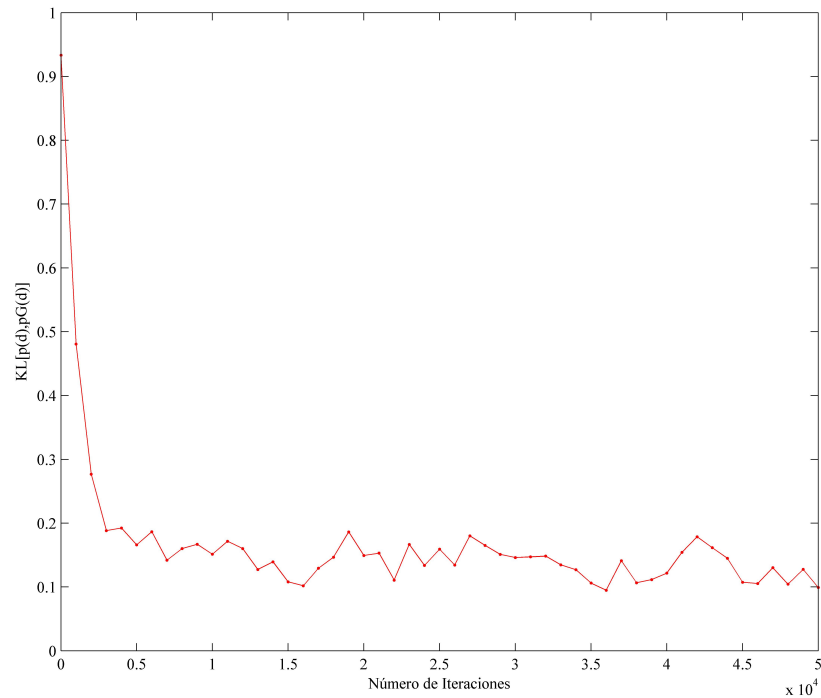


Figura 4.3: Divergencia entre $p(d)$ y $p_G(d)$ respecto al número de iteraciones para el ejemplo 4.3.

Observe que la divergencia entre $p(d)$ y $p_G(d)$ decreció y alcanzó la condición de paro, pero el error, es decir la diferencia real entre las probabilidades $p(d)$ y $p_G(d)$ es de ± 0.13 aproximadamente. Si se desea un menor error se debe fijar la divergencia mínima en un valor mucho más pequeño.

Capítulo 5

Comparación de redes no supervisadas

Debido a que cada tipo de red se ha especializado para resolver ciertos tipos de problemas, resulta difícil atacar un mismo problema con diferentes redes. Sin embargo la aplicación que se ajusta mejor a la mayoría de los diferentes tipos de redes es la de reconocimiento de imágenes digitales. Es por eso que se eligió el problema de identificar barras horizontales y verticales en imágenes en blanco y negro. Cabe mencionar que la idea no es original, esta aplicación ya se ha hecho en otros trabajos [9, 13]. El trabajo original, se implementó con imágenes de 4×4 pixeles. En este trabajo se usarán imágenes (patrones) de 3×3 pixeles, ya que se desea comparar los resultados con los obtenidos en [17], además de que con imágenes de mayor tamaño el número de patrones aumenta exponencialmente, por lo que el tiempo de ejecución se incrementa considerablemente.

Dado que se emplean imágenes de 3×3 pixeles, se tienen $2^9 = 512$ patrones distintos, pero de éstos sólo son de interés los que tienen barras verticales o barras horizontales, como se muestra en la Figura 5.1.

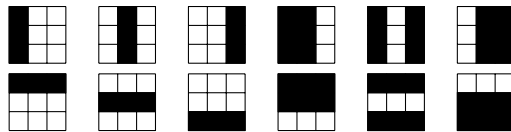


Figura 5.1: Imágenes con barras horizontales y verticales.

De esta manera el conjunto de entrenamiento consiste de imágenes en blanco y negro con barras verticales u horizontales. Para codificar las imágenes en blanco y negro, se emplean valores binarios ya sean -1 y 1 o bien 0 y 1, esta primera opción se muestra en la Figura 5.2.

Una vez descrito el problema se identificó que las redes que mejor se adaptan para resolverlo son la red de Hopfield y la red LVQ, pues son algoritmos que se han aplicado principalmente al reconocimiento de imágenes y a problemas de clasificación [12, 16], es por ello que únicamente se compararon esas redes con la máquina de Helmholtz implementando el algoritmo wake-sleep.

Es posible trabajar en cualquier lenguaje de programación imperativo; sin embargo dado que en MATLAB existe el toolbox de redes neuronales, además de que nos permite trabajar muy fácilmente con matrices, se optó por este lenguaje de programación. Se usó la versión de MATLAB 7.8.0 (Release 2009a) y los programas se ejecutaron en una computadora con Sistema Operativo Windows 7 Ultimate de 32 bits, Memoria RAM de 4Gb, Procesador Intel(R) core(TM) i3-2120CPU@ 3.30GHz. en el laboratorio de Matemáticas Aplicadas de la Universidad Tecnológica de la Mixteca.

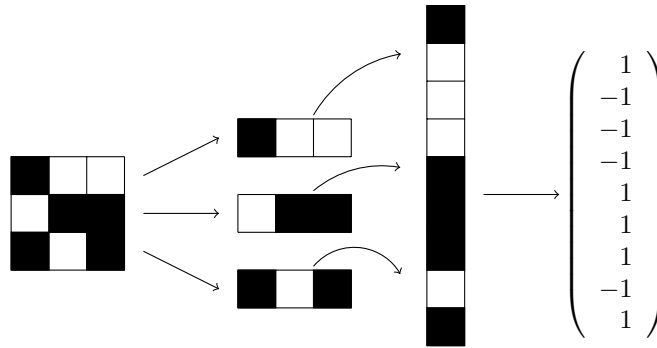


Figura 5.2: Codificación de los patrones de entrada.

5.1. Red de Hopfield

Antes de describir la forma en que se va a resolver el problema se aclara que para esta red no se usará el toolbox de MATLAB, pues se necesita la versión discreta de la red de Hopfield, misma que no está implementada en el toolbox de MATLAB¹.

Se trabajará con valores binarios -1 y 1; donde cada pixel blanco estará representado por -1 y cada pixel negro por 1. En este caso, el conjunto de entrenamiento constará de las doce imágenes descritas al inicio del capítulo y se agregan dos más, que corresponden a las imágenes que no son ni horizontales ni verticales como se muestra en la Figura 5.3.



Figura 5.3: Imágenes que no tienen barras horizontales ni verticales.

Para obtener resultados óptimos con esta red los vectores correspondientes a los patrones del conjunto de entrenamiento deben ser ortogonales entre sí como se muestra en [12], en este caso, no es así. Por ejemplo los patrones del conjunto de entrenamiento, p_1 () y p_2 () no son ortogonales. Aún así se analizará su comportamiento y que tanto puede aprender.

La red estará compuesta por 9 neuronas, una por pixel. Durante la fase de entrenamiento, se almacenan en la red las 14 imágenes, de manera codificada, como se muestra en la Tabla 5.1.

Se calcula la matriz de pesos de la red a partir de los 14 patrones binarios que representan las imágenes:

$$W = (p_1^T p_1 - I) + (p_2^T p_2 - I) + \dots + (p_{14}^T p_{14} - I).$$

Esto se hace mediante la función **Aprendizaje_Hopfield**(véase la sección A.1.1 del apéndice).

Después se pasa a la fase de funcionamiento en la que se se harán pasar los patrones por la red para averiguar la salida que ésta les asigna, es decir se simulará el funcionamiento de la red, para ello se emplearán todos los posibles patrones (512) a éstos se les llamará el conjunto de simulación. La fase de funcionamiento en este caso se detiene hasta que las salidas de las neuronas se estabilizan, lo cual ocurrirá cuando dejen de cambiar de valor. Tomando en consideración que la condición de paro no siempre es alcanzable, debido a que en general los patrones del conjunto de entrenamiento no son ortogonales entre sí, (particularmente ya se comprobó que hay al menos dos que no son ortogonales), es necesario modificar

¹La versión de la red de Hopfield implementada en el toolbox de MATLAB es una modificación del modelo original, propuesta por: Jian-Hua Li, Anthony N.Michel y Wolfgang Porod, para mayores detalles puede consultar [15].

Barras verticales	$p_1 = (-1 \quad -1 \quad 1 \quad -1 \quad -1 \quad 1 \quad -1 \quad -1 \quad 1)$
	$p_2 = (-1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1)$
	$p_3 = (1 \quad -1 \quad -1 \quad 1 \quad -1 \quad -1 \quad 1 \quad -1 \quad -1)$
	$p_4 = (1 \quad 1 \quad -1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1 \quad -1)$
	$p_5 = (-1 \quad 1 \quad -1 \quad -1 \quad 1 \quad -1 \quad -1 \quad 1 \quad -1)$
	$p_6 = (1 \quad -1 \quad 1 \quad 1 \quad -1 \quad 1 \quad 1 \quad -1 \quad 1)$
Barras horizontales	$p_7 = (-1 \quad -1 \quad -1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1)$
	$p_8 = (1 \quad 1 \quad 1 \quad -1 \quad -1 \quad -1 \quad 1 \quad 1 \quad 1)$
	$p_9 = (-1 \quad -1 \quad -1 \quad 1 \quad 1 \quad 1 \quad -1 \quad -1 \quad -1)$
	$p_{10} = (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad -1 \quad -1 \quad -1)$
	$p_{11} = (-1 \quad -1 \quad -1 \quad -1 \quad -1 \quad -1 \quad 1 \quad 1 \quad 1)$
	$p_{12} = (1 \quad 1 \quad 1 \quad -1 \quad -1 \quad -1 \quad -1 \quad -1 \quad -1)$
Ni horizontales ni verticales	$p_{13} = (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1)$
	$p_{14} = (-1 \quad -1 \quad -1 \quad -1 \quad -1 \quad -1 \quad -1 \quad -1 \quad -1)$

Tabla 5.1: Conjunto de entrenamiento para la red de Hopfield.

el algoritmo fijando un límite de iteraciones.

La red se simuló usando los 512 patrones, con distintos límites de iteración, esto se hace mediante la función **Hopfield** que se muestra en la sección A.1.3 del apéndice A y se obtuvieron los resultados que se muestran en la Tabla 5.2.

No. de iteraciones	% de clasificaciones correctas	No. de patrones estabilizados	Tiempo de ejecución
100	27.34	140	0.0728 min
1000	27.34	140	0.7379 min
5000	27.34	140	3.7005 min
10000	27.34	140	7.3726 min

Tabla 5.2: Comportamiento de la red de Hopfield.

Como se puede observar el número de patrones que se estabilizaron (140) no cambia con el número de iteraciones, más aún se trata exactamente de los mismos patrones. ¿Qué pasa si el programa se ejecuta con un número menor a 100 iteraciones? Se realizaron diferentes pruebas y se observó que a partir de 4 iteraciones el resultado de la red es el mismo, es decir, los patrones estables (140) son los mismos. Por otro lado, se observó que los patrones que no se estabilizan alternan entre dos salidas distintas en cada iteración por lo que no tiene sentido colocar un límite de iteración mayor.

En las Figuras 5.4 y 5.5 se muestran las imágenes del conjunto de simulación que se estabilizaron y las imágenes resultantes después de 4 iteraciones, respectivamente. Se observó que la red dió como resultado imágenes que se pueden clasificar de la siguiente manera:

- Imágenes completamente blancas
- Imágenes completamente negras
- Imágenes con barras verticales
- Imágenes con barras horizontales
- Imágenes con ambas barras (horizontales y verticales)

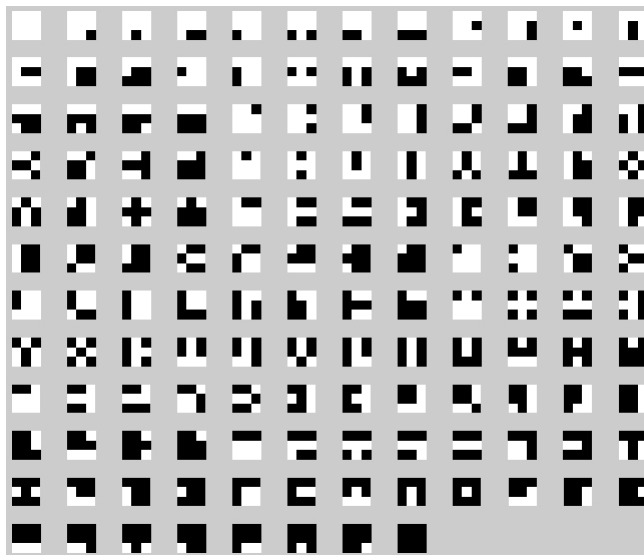


Figura 5.4: Imágenes del conjunto de simulación que se estabilizaron en la red de Hopfield.

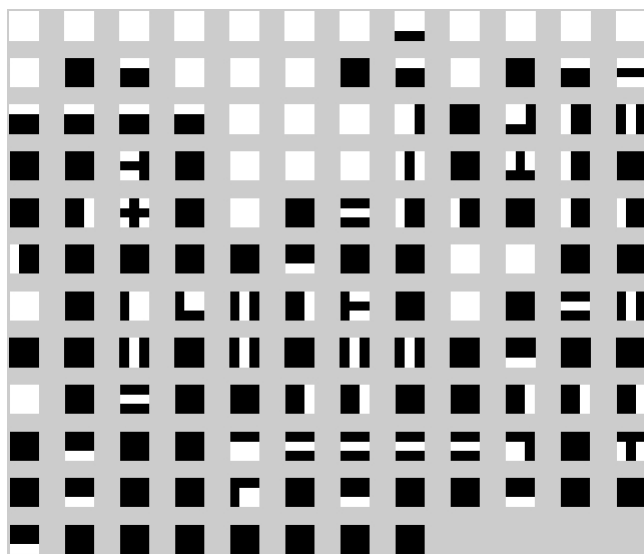


Figura 5.5: Imágenes que devuelve la red de Hopfield para los patrones estabilizados.

Se comparan los patrones de entrada con la salida que da la red, en los casos en que la salida se estabilizó (140 patrones) y se observa lo siguiente:

1. Las imágenes o patrones del conjunto de entrenamiento que tienen a lo más 2 píxeles negros (aproximadamente 20% de ruido), la red las clasificó como imágenes completamente blancas, se puede decir que eliminó el ruido.
2. Clasifica como imágenes completamente negras aquellas que cumplen con las siguientes restricciones:
 - Aquellas imágenes que tienen 4 ó 5 píxeles negros y que además no forman al menos una barra horizontal, vertical o ambas.

- Las imágenes que tienen de 7 a 9 píxeles negros, aún cuando estas imágenes tengan barras horizontales verticales o ambas
- 3. La red clasifica de manera correcta las imágenes en las cuales están bien definidas las barras verticales (que no tienen píxeles de más), y completa la barra de aquellas imágenes a las que les falta a lo más un píxel (10%), siempre y cuando tengan una guía, esto es, una barra dentro de la imagen que esté bien definida.
- 4. Para clasificar las imágenes con barras horizontales, se aplica el mismo criterio que con las verticales.
- 5. Por último la red clasifica de manera adecuada las imágenes mixtas (con ambas barras), como se puede observar en las Figuras 5.4 y 5.5 la salida es idéntica a la entrada.

Las 140 imágenes que se estabilizaron caen en alguna de las cinco categorías anteriores, por otro lado el conjunto de imágenes que no se estabilizaron (372), no se muestra debido a que son demasiadas, pero se muestran algunos ejemplos en la Figura 5.6, la primera fila corresponde a las imágenes de los patrones del conjunto de simulación y la segunda y tercera fila corresponde a las imágenes de los patrones que la red generó en las iteraciones 4 y 5. Se observa que los patrones estarán oscilando entre dos salidas diferentes, siempre que el número de iteraciones sea par se obtendrán los patrones que se generaron en la iteración 4 y cuando el número de iteraciones sea impar la red dará como salida los patrones que se muestran en la iteración 5.

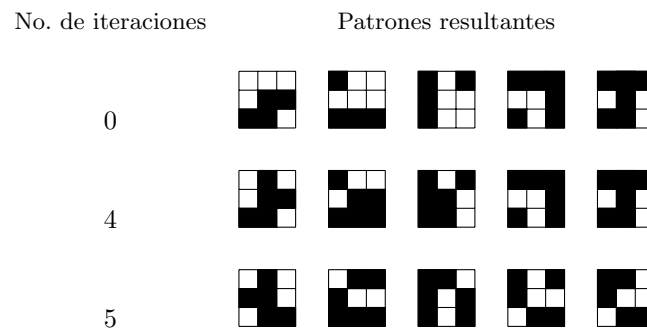


Figura 5.6: Algunas imágenes de patrones que no se estabilizaron en la red de Hopfield.

También se observa que no se estabilizan aquellas imágenes que forman diagonales o que tienen píxeles dispersos (es decir, no tienen barras horizontales, verticales o una mezcla de éstas de manera definida). Por otra parte hay imágenes del conjunto de simulación que tienen barras horizontales con 10% de ruido y se esperaría que les quitara el ruido, pero lo que hace es agregar píxeles. Lo mismo ocurre cuando la imagen tiene barras en ambas direcciones.

La red se desempeña de manera adecuada considerando que el conjunto de entrenamiento no es ortogonal. Una desventaja de esta red, para este ejemplo, es que el número de patrones que se no se estabilizó es muy alto, la red sería incapaz de dar una respuesta en esos casos.

5.2. Red LVQ

La red LVQ emplea de igual forma aprendizaje supervisado como no supervisado para formar clasificaciones, por lo tanto tiene mayor capacidad para distinguir clases, en este problema se pueden distinguir cuatro clases, las cuales son, imágenes que tengan barras horizontales y verticales a la vez, sólo barras

horizontales, sólo barras verticales, y aquellas imágenes que no tengan ni barras horizontales ni verticales. Para esta red se emplearán valores binarios 1 y 0; 1 para un pixel negro y 0 para un pixel blanco. El conjunto de entrenamiento se muestra en la Tabla B.1, del apéndice B.

Como ya se mencionó se pueden formar cuatro clases, específicamente se tiene que:

- Clase 1: Imágenes con barras verticales.
- Clase 2: Imágenes con barras horizontales.
- Clase 3: Imágenes mixtas, aquellas que tienen tanto barras horizontales como verticales.
- Clase 4: El resto de las imágenes.

Dadas las clases e ignorando el ruido se identifican 50 subclases, éstas se muestran en la Figura 5.7, en donde: el renglón 1 corresponde a imágenes verticales, el renglón 2 corresponde a imágenes horizontales, del renglón 3 al 8 corresponden a imágenes mixtas y en el renglón 9 se muestran las imágenes de los patrones que no son ni horizontales ni verticales².

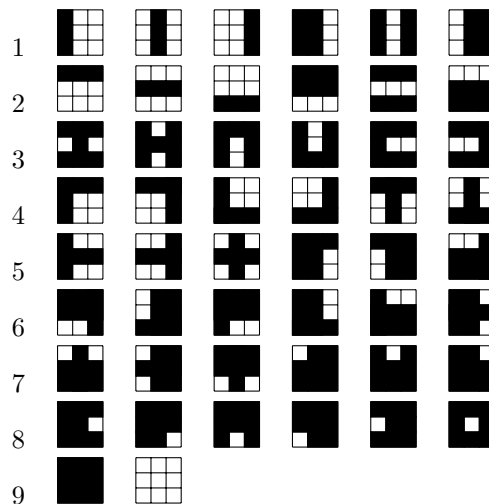


Figura 5.7: Imágenes de los patrones que forman las subclases.

Una vez analizado el problema se puede definir la topología de la red LVQ que lo resolverá. La red constará de 50 neuronas en la capa de entrada, que corresponden al número de subclases y 4 neuronas en la capa de salida, que corresponden al número de clases. La primera capa emplea la función de activación *compet* y la segunda capa la función de activación *purelin*.

El algoritmo se implementó haciendo uso de las funciones del toolbox de redes neuronales en MATLAB, por lo tanto se hace una modificación al algoritmo de aprendizaje que se describió en la sección 3.3.2, el programa **Redlvq** que aparece en el apéndice A incluye la modificación que es la introducción de un vector que contiene el porcentaje de patrones que pertenecen a cada clase, y que se llamará *PC* (Porcentaje de Clase).

En este problema en las clases 1 y 2 hay 6 imágenes en cada una, en la clase 3 hay un total de 36 imágenes (barras horizontales y verticales) y en la clase 4 hay 464 imágenes (ni horizontales ni verticales).

²La imagen con todos sus píxeles negros así como la que tiene todos sus píxeles blancos no se consideran mixtas, dado que no se pueden identificar barras horizontales o verticales.

De esta manera el vector PC está dado como sigue:

$$PC = [6/512 \quad 6/512 \quad 36/512 \quad 464/512]$$

Con las características descritas se entrena la red, implementando el programa **Redlvq** que se muestra en la sección A.2.1 del apéndice A, se ejecutó el programa variando el límite de iteraciones y la tasa de aprendizaje de fijó en 0.01 (se eligió esta tasa ya que si se elige una tasa de aprendizaje más alta la modificación de los pesos será muy grande y los patrones oscilarán), obteniendo los resultados que se muestran en la Tabla 5.3:

No. de iteraciones	% de clasificaciones correctas	No. de patrones	Tiempo de ejecución
100	58.40	299	0.15min
200	62.11	318	0.28min
300	63.87	327	0.42min
400	63.09	323	0.55min
500	63.09	323	0.7min
1000	56.45	289	1.25min
1500	44.73	229	2.05min
2000	45.31	232	2.48min
5000	32.23	165	7min
10000	33.01	169	14min
20000	41.02	210	30min

Tabla 5.3: Comportamiento de la red LVQ.

Se observa que con 300 iteraciones la red clasifica adecuadamente mayor número de patrones y conforme las iteraciones aumentan el número de patrones clasificados no aumenta considerablemente. En la siguiente página en la Tabla 5.8 se pueden apreciar algunos patrones que la red no clasifica correctamente.

Los patrones que la red no clasifica correctamente son las barras verticales, pues pertenecen a la clase 1 y la red los clasifica en la clase 4. Sin embargo se observa que clasifica de manera satisfactoria las imágenes que tienen barras horizontales, pues de las 6 imágenes que tienen éstas barras, sólo clasificó de manera incorrecta 2 imágenes. También se observa que la red clasifica de manera adecuada los patrones mixtos dado que clasificó incorrectamente solo 6 de los 36 patrones. Imágenes con barras verticales con a lo más 10% de ruido los clasifica dentro de la clase 4. Las imágenes horizontales con a lo más 10% de ruido los clasifica dentro de la clase 3. Y las imágenes restantes que corresponden a la clase de imágenes que no son ni horizontales ni verticales las clasifica en la clase 3 o bien en la clase 2.

La cantidad de entradas que son clasificadas correctamente es mayor que en la red de Hopfield. Esto puede deberse a que esta red emplea tanto aprendizaje supervisado como no supervisado. A diferencia de la red de Hopfield, esta red da una respuesta para cada entrada. Sin embargo no es muy buena clasificando imágenes con barras verticales, pues patrones sin ruido alguno son clasificadas incorrectamente, como se muestra en la Tabla 5.8.

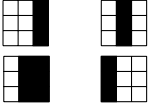
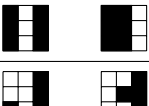
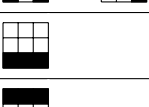
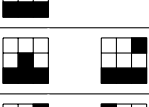
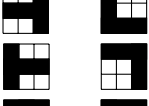
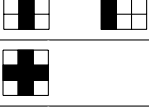
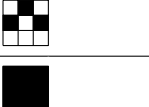
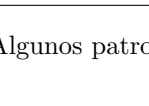


Imagen	Clase correcta	Clase resultante
	1	4
	1	3
	1	4
	2	4
	2	3
	2	3
	3	4
	3	4
	4	2
	4	3

Figura 5.8: Algunos patrones que la red LVQ no clasifica correctamente.

5.3. Algoritmo Wake-Sleep

Como ya se describió en un capítulo anterior el algoritmo wake-sleep se emplea en redes con aprendizaje no supervisado, donde no existe un asesor externo que indique si la salida de la red es o no correcta, en este ejemplo a cada patrón se le asigna una probabilidad, y el objetivo es que la red reproduzca la distribución de probabilidad.

Dado que se tienen 512 posibles patrones (los cuales pueden incluir líneas verticales y horizontales y se desean clasificar), se define la distribución de probabilidad distinta de cero para aquellos patrones con barras verticales y patrones con barras horizontales, además estas probabilidades deben ser diferentes entre sí para distinguir unas de otras, en este caso, la probabilidad de que un patrón tenga barras verticales es el doble de la probabilidad de que tenga barras horizontales (se eligió una razón de 2 entre las probabilidades, pero puede elegirse otra siempre que sea distinta de 1 para distinguir entre las clases). Como hay 6 patrones en cada caso, entonces:

- La probabilidad del patrón d si contiene barras verticales es, $p(d) = 2/18$.
- La probabilidad del patrón d si contiene barras horizontales es, $p(d) = 1/18$.
- La probabilidad del patrón d en cualquiera de los otros 500 casos es, $p(d) = 0$.

La estructura de la máquina de Helmholtz puede variar pero con el fin de comparar los resultados aquí obtenidos con los resultados presentados en [17], se emplea la misma topología que propone el artículo mencionado, así la red consta de 6 neuronas en la capa de entrada, que se llamará y ; y 1 neurona en la capa de salida, que se llamará x ; como se muestra en la Figura 5.9 en la página siguiente.

Se implementó en MATLAB el algoritmo descrito en la sección 4.2, mediante el programa **wake-sleep** el cual se describe en la sección A.3.6 en el apéndice A. En base a [17] se establecieron las tasas de aprendizaje de la siguiente manera: $\varepsilon_V = 0.15$, $\varepsilon_W = 0.01$. Y dado que no se especifica la tasa de aprendizaje, ε_b , para los pesos de reconocimiento que conectan a la constante 1 con la capa de salida, se hicieron varias pruebas y se observó que con el valor de la tasa de aprendizaje $\varepsilon_b = 0.01$ se replicaron los resultados del artículo citado³.

El objetivo es minimizar la diferencia entre la distribución de probabilidad y la distribución de probabilidad generativa, esta diferencia se puede medir mediante la divergencia de Kullback-Leibler dada por $KL[p(D), p_G(D)]$. En la Tabla 5.4 se muestra la divergencia entre las distribuciones de probabilidad, en diferentes iteraciones.

Número de iteraciones	Divergencia (KL)	Tiempo de ejecución
1000	2.356	0.059min
5000	1.3769	0.22min
10000	0.8446	0.41min
30000	0.3617	1.20min
60000	0.4137	2.37min

Tabla 5.4: Comportamiento de la máquina de Helmholtz.

El programa se ejecutó variando el número de iteraciones, en la página 63 se pueden observar los resultados que corresponden a 60,000 iteraciones en la Tabla 5.5. Ésta muestra únicamente patrones cuya probabilidad $p_G(d)$ generada por la máquina de Helmholtz es superior a 0.001.

³Tómese en cuenta que el aumento o decremento de la tasa de aprendizaje sirve para que el aprendizaje sea más rápido o más lento, si la tasa de aprendizaje es muy grande se corre el riesgo de que el aprendizaje no sea estable.

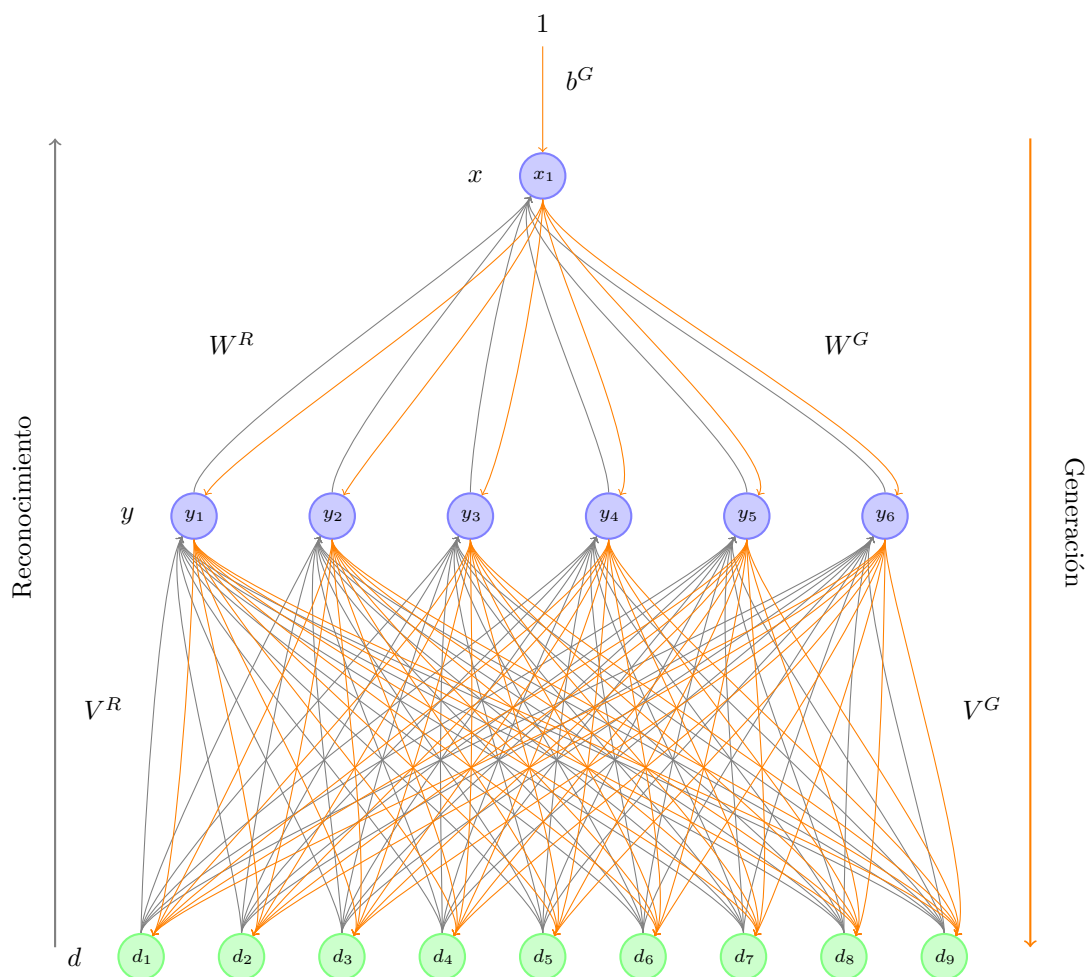


Figura 5.9: Topología de la máquina de Helmholtz (9-6-1).

	Patrón d	$p(d)$	$p_G(d)$
Barras verticales	$d_1 = (0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1)$	0.11	0.1007
	$d_2 = (0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1)$	0.11	0.0989
	$d_3 = (1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0)$	0.11	0.1014
	$d_4 = (1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0)$	0.11	0.0986
	$d_5 = (0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0)$	0.11	0.0940
	$d_6 = (1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1)$	0.11	0.1098
Barras horizontales	$d_7 = (0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)$	0.0556	0.0182
	$d_8 = (1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1)$	0.0556	0.0279
	$d_9 = (0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0)$	0.0556	0.0133
	$d_{10} = (1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0)$	0.0556	0.0345
	$d_{11} = (0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1)$	0.0556	0.0566
	$d_{12} = (1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)$	0.0556	0.0044
Ni horizontales ni verticales	$d_{13} = (0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0)$	0	0.0183
	$d_{14} = (1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0)$	0	0.0169
	$d_{15} = (0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0)$	0	0.0155
	$d_{16} = (0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0)$	0	0.0155
	$d_{17} = (0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1)$	0	0.0151
	$d_{18} = (0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1)$	0	0.0151
	$d_{19} = (0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1)$	0	0.0128
	$d_{20} = (1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1)$	0	0.0128
	$d_{21} = (1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1)$	0	0.0123
	$d_{22} = (1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0)$	0	0.0084
	$d_{23} = (1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0)$	0	0.0081
	$d_{24} = (1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)$	0	0.0061
	$d_{25} = (1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1)$	0	0.0040
	$d_{26} = (1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0)$	0	0.0033
	$d_{27} = (1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1)$	0	0.0026
	$d_{28} = (1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1)$	0	0.0023
	$d_{29} = (1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1)$	0	0.0022
	$d_{30} = (1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0)$	0	0.0021
	$d_{31} = (1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0)$	0	0.0020
	$d_{32} = (1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0)$	0	0.0016
	$d_{33} = (1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1)$	0	0.0014
	$d_{34} = (1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)$	0	0.0013
	Otros	0	<0.001

Tabla 5.5: Resultados de la máquina de Helmholtz después de 60,000 iteraciones.

Si se toma como margen de error 0.025, entonces se puede definir que:

- Un patrón d será clasificado como vertical si $p_G(d) \in [0.085, 0.135]$,
- Un patrón d será clasificado como horizontal si $p_G(d) \in [0.0306, 0.0806]$,
- Un patrón d no pertenece a ninguna de las dos clases (horizontal o vertical), si $p_G(d) < 0.001$.

Una vez definidos los intervalos, se analizan los resultados obtenidos. Se observa que la red clasificó de manera correcta las barras verticales, pues la probabilidad generativa, $p_G(d)$, de los 6 patrones con barras verticales sin ruido están en este intervalo $[0.085, 0.135]$. No sucede lo mismo con las barras horizontales, dado que de 6 patrones con barras horizontales solo clasificó de manera correcta a 2 patrones que son d_{10} y d_{11} . Por otra parte se observa que al patrón d_{13} (que corresponde a una barra horizontal con 20 % de ruido) le asigna una probabilidad muy cercana a la del patrón d_7 (que es también una barra horizontal), la red no los clasifica como barras horizontales, pero la máquina de Helmholtz está generalizando al asignar valores cercanos a patrones muy parecidos.

En la Figura 5.10, se muestra el comportamiento de la divergencia graficando su valor cada 500 iteraciones; como se puede observar, la divergencia va decreciendo, lo cual indica que las distribuciones se aproximan. De manera similar a la red LVQ, la máquina de Helmholtz da una respuesta para cada entrada, además se obtiene la distribución de probabilidad, la cual se puede almacenar en una tabla sin necesidad de simular la red. Las imágenes que contienen cierta cantidad de ruido, tienen valores muy cercanos a patrones sin ruido, esto indica que la red está tratando de generalizar, es decir, si se tienen patrones muy similares, se espera que los clasifique en la misma categoría.

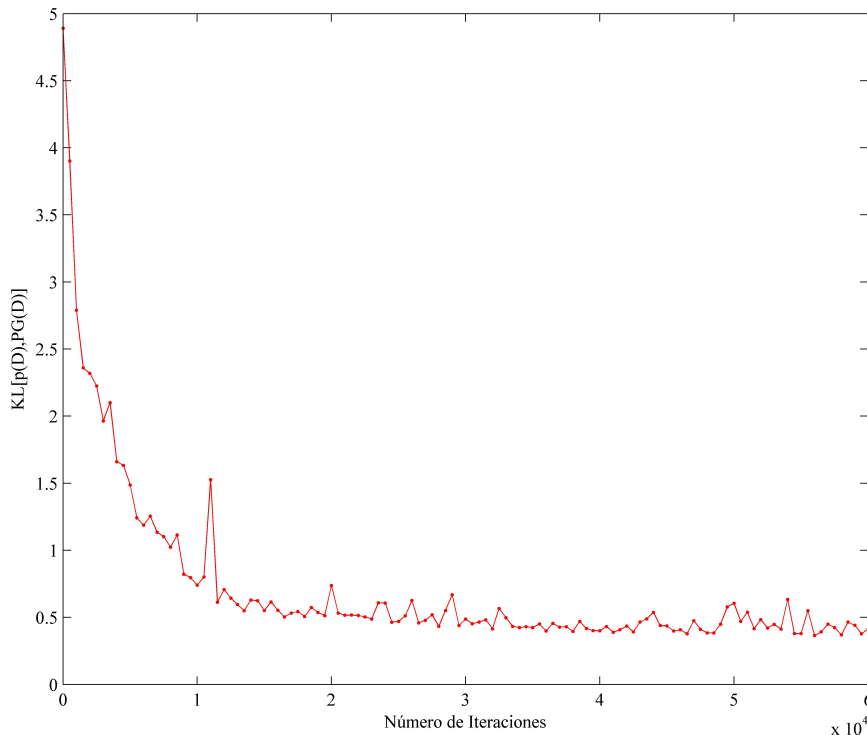


Figura 5.10: Divergencia entre $p(d)$ y $p_G(d)$ respecto al número de iteraciones.

Conclusiones

El primer objetivo de este trabajo fue estudiar redes con aprendizaje no supervisado, por ello, del Capítulo 1 al Capítulo 2 se estudiaron algunos conceptos preliminares sobre RNA, en el Capítulo 3, se logró tener una idea general de la topología, arquitectura y funcionamiento de algunas RNA con aprendizaje no supervisado, lo cual ayudó a la elección de las redes que se emplearon para el problema de identificar barras horizontales y verticales en imágenes de 3×3 píxeles.

Otro objetivo de este trabajo era el comparar el desempeño de algunas redes con aprendizaje no supervisado, para ello se tomó el problema de reconocimiento de barras horizontales y verticales en imágenes en blanco y negro. Se eligió MATLAB como el lenguaje apropiado para la codificación de los algoritmos ya que nos permite trabajar muy fácilmente con matrices además de que tiene implementado el toolbox de Redes Neuronales. Se implementaron los algoritmos **Hopfield.m**, **Redlvq.m** y **wake-sleep.m**, se analizaron los tiempos de ejecución así como el desempeño de las redes y se observó que: el tiempo de ejecución es proporcional al desempeño de la red pues se registraron los siguientes tiempos, 0.0728min para la red de Hopfield, 0.42min para la red LVQ y 2.37min para la máquina de Helmholtz; las redes que mejor se desempeñaron en la resolución de este problema, es decir, las redes que clasificaron correctamente el mayor número de patrones fueron la red LVQ y la máquina de Helmholtz. Esto no significa que la red de Hopfield no es adecuada, pues recordemos que los patrones del conjunto de entrenamiento, no eran ortogonales entre sí.

En cuanto a la red LVQ y la Máquina de Helmholtz son diferentes parámetros los que se emplean, por ejemplo la primera emplea ambos tipos de aprendizaje y por lo tanto se puede apreciar la salida de la red dada la entrada mientras que en la máquina de Helmholtz el desempeño de la red se evalúa de acuerdo a la distribución de probabilidad generada, y como se pudo observar en los resultados del Capítulo 5 la divergencia entre la distribución de probabilidad de los patrones y la distribución de probabilidad generada por la red sí está disminuyendo.

Dado que no hay un criterio para elegir la topología de una red para resolver un problema particular, la elección de los parámetros de la red generalmente se hace a prueba y error, tal fue el caso de la elección de la tasa de aprendizaje para el algoritmo wake-sleep; en el caso de la red LVQ el número de subclases pudo ser mayor o bien menor, es a criterio del programador y considerando las características del problema a resolver.

Una ventaja de las Redes Neuronales Artificiales es que son fáciles de implementar, además de que para problemas de clasificación y reconocimiento de imágenes han dado buenos resultados.

Bibliografía

- [1] Anderson, James A. *An introduction to neural networks*. Cambridge, Mass.: Mit Press, 1997.
- [2] Ajith, Abraham. Artificial Neural networks. *Handbook of Measuring System Design*. Oklahoma State University, Stillwater, OK, USA.
- [3] Awodele, Oñudele, Jegede, Olawale. Neural networks and its Application in Engineering. *Proceedings of Informing Science and IT Education Conference (InSITE)*. Dept. of Computer Science and Mathematics, Babcock University, Nigeria, 2009.
- [4] Camps, Gustavo. *Redes neuronales aplicadas en el problema de la intoxicación por digoxina*. Proyecto final de carrera Ingeniería Electrónica, Universitat de Valencia, 1998.
- [5] Chen, S.; et al. Reconstruction of Binary Signals Using an Adaptive Radial Basis-Function Equalizer. *Signal Processing*, vol 22, pp 77-93, Enero, 1991.
- [6] Dayan, Peter. Unsupervised Learning. *The MIT Encyclopedia of the Cognitive Sciences*
- [7] Dayan, Peter; Hinton, Geoffrey E.; Neal Radford M.; Zemel Richard S. *The helmholtz Machine*. Science, 14 diciembre 1994.
- [8] Dayan, Peter. *Helmholtz Machines and Wake-Sleep Learning*. Handbook of Brain Theory and Neural Networks, 2.
- [9] Frey, Brendan J.; Dayan, Peter; Hinton, Geoffrey E. *A simple algorithm that discover efficient perceptual codes*. Computational and Biological Mechanism of Visual Coding, Cambridge University Press, New York NY, 1997.
- [10] Gibson, Gavin J.; Siu, Sammy; Cowan Colin F.N. The Application of Nonlinear Structures to the Reconstruction of Binary Signals. *IEEE Transactions on Signal Processing*, vol 39, pp 1877-1884, Agosto, 1991.
- [11] Hagan, Martin T.; Demuth, Howard B.; Beale, Mark. *Neural Network Desing*. Estados Unidos de América: PWS Publishing Company, 1996.
- [12] Hilera, José R. y Martínez, Víctor J. *Redes neuronales artificiales (Fundamentos, modelos y aplicaciones)*. Madrid, España: Alfaomega Ra-Ma, 2000.
- [13] Hinton, Geoffrey E.; Dayan, Peter; Frey, Brendan J., Radford Neal M. *The wake-sleep algorithm for unsupervised neural networks*. Department of Computer Science, University of Toronto, 6 King's College Road, Toronto M5S 1A4, Canada. Science 1995.
- [14] Hopfield, John. Neural networks and phisycal systems with emergent collective computational abilities. *Proceedins of the National Academy of Science*, vol. 81, pp 3088-3092.
- [15] Hudson Beale; Mark, Hagan; Martin T, Demuth; Howard B. *Neural Network Toolbox. Users Guide*. The MathWorks, Inc. 1992-2012.
- [16] Isasi Viñuela, Pedro y Galván León, Inés M. *Redes de neuronas artificiales. Un enfoque práctico*. Madrid: Pearson educación, S.A., 2004.

- [17] Kirby, Kevin G. *A tutorial on Helmholtz Machine*. Department of Computer Science, Northern Kentucky University, June 2006.
- [18] Martín del Brío, Bonifacio y Sanz Molina, Alfredo. *Redes de neuronales y sistemas borrosos*. Tercera edición. Madrid, España: Alfaomega Ra.Ma., 2007.
- [19] Martín, José D. *Implementación de redes neuro-difusas para ser aplicadas en problemas de clasificación y modelización*. Proyecto final de carrera, Ingeniería Electrónica, Universitat de Valencia, 1999.
- [20] Mehrotra, Kishan; Mohan, Chilukuri K.; Ranka, Sanjay. *Elements of Artificial Neural Networks*. 2a. edición, MIT Press, 1997.
- [21] Moallemi, Ciamac. Clasifying Cells for Cancer Diagnosis Using Neural Networks. *IEEE Expert*, Diciembre, 1991.
- [22] Molina, Javier. *Uso de comités de expertos en problemas de clasificación*. Proyecto final de carrera, Ingeniería Electrónica, Universitat de Valencia, 1999.
- [23] Müller, Berndt; Reinhardt, Joachim; Strickland, Michael T. *Neural networks: an introduction*. 2a. edición, New York: Springer, 1995.
- [24] Pavlidis, Theo, Mori, S. Special Issue on Optical Character Recognition, *Proceedings of the IEEE*, Vol. 80, No. 7, Julio, 1992.
- [25] Shadlen, Michael N, Newsome William T. *Noise, neural codes and cortical organization* Departamento de Bioingeniería, universidad de Tacumán.
- [26] Serrano, Antonio J.; Soria, Emilio; Martín, José D. *Redes neuronales artificiales*. Escuela Técnica Superior de Ingeniería. Departamento de Ingeniería Electrónica. Universidad de Valencia. Curso 2009-2010.
- [27] Soria, Emilio. et al. Application of an Artificial Neural Network with a Piecewise-Linear Activation Function to Determine the End of the T-Wave in an ECG". *World Congress on Medical Physics and Biomedical Engineering*, Niza, Septiembre, 1997.
- [28] Soria, Emilio; Blanco, Antonio. *Redes neuronales artificiales. Autores científico-técnico y académicos*.
- [29] Terrence J. Sejnowski. *Neural networks. Sleep and memory*. Howard Hughes Medical Institute, Salk Institute for Biological Studies, La Jolla, California 92037, USA, and Department of Biology, University of California at San Diego, La Jolla California 92093, USA.
- [30] Características principales de las redes neuronales. Universidad Tecnológica de Pereira. <http://ohm.utp.edu.co/neuronales> [consulta: 21 de mayo de 2012]
- [31] Chakraborty, RC. Fundamentals of Neural Networks: IA course lecture 37-38 <http://www.myreaders.info/html/artificial-intelligence.html> [consulta: 18 de octubre de 2012]
- [32] Sarle, Warren S., Cary, NC. What does unsupervised learning learn?, <ftp://ftp.sas.com/pub/neural/FAQ.html> [consulta: 18 de octubre de 2012]

Apéndice A

Código fuente

A.1. Red de Hopfield

A continuación se presentan las funciones realizadas en MATLAB, para crear una red de Hopfield.

A.1.1. Función auxiliar: Aprendizaje_Hopfield

Con esta función se realiza la etapa de aprendizaje de la red de Hopfield, en la cual se crea la matriz de pesos de la red.

```
function[M]=Aprendizaje_Hopfield(E)
% Aprendizaje_Hopfield: Esta funcion crea la matriz de pesos de una red de
% Hopfield dado el conjunto de entrenamiento.
%Entrada:
    %E: Es una matriz de nxm donde n es numero de patrones del conjunto de
    %entrenamiento y m es el numero de entradas de cada patron.
%Salida:
    %M: Es una matriz de mxm que almacena los pesos de la red, donde m es
    %como se definio anteriormente.
[r,c]=size(E);
    I=eye(c);
M=zeros(c);
    for i=1:r
        M=M+(E(i,:)'*E(i,.)-I);
    end
end
```

A.1.2. Función auxiliar: compara

La función compara es una función auxiliar cuyo objetivo es indicar qué patrones se estabilizaron así como el porcentaje de éstos. Cabe aclarar que de acuerdo al algoritmo que se describió en la sección 3.1.2, es necesaria la comparación entre la salida que se obtuvo en la iteración $k - 1$ con la que se obtuvo en la iteración k .

```
function[E,Estab]=compara(P,S)
%compara: Es una funcion que indica que patrones del conjunto de
%simulacion se estabilizaron.
%Entradas:
    % P: Es una matriz de nxm, donde n es el numero de patrones del
```

```

% conjunto de simulacion y m es el numero de entradas de cada
% patron.
% S: Es una matriz con las mismas características de P.
%Salidas:
% E: Es un escalar que devuelve el porcentaje de los patrones que se
% estabilizaron.
% Estab: Es un vector columna de dimension n cuyas componentes son unos
% y ceros, 1 si el patron se estabilizo y 0 en otro caso; n es el
% numero de patrones del conjunto de simulacion.

cont=0;
r=size(P,1);
Estab=zeros(r,1);
for i=1:r
    if P(i,:)==S(i,:)
        cont=cont+1;
        Estab(i)=1;
    end
end
E=cont/r;
end

```

A.1.3. Función principal: Hopfield

La función Hopfield es la función principal que crea y simula una red de Hopfield; en la cual se emplean las funciones auxiliares descritas anteriormente.

```

function[S,E,Estab]=Hopfield(P,PS,Li)
%Hopfield: Funcion que crea y simula una red de hopfield.
%Entradas:
%P : Matriz de entrenamiento de nxm donde n es el numero de patrones
%del conjunto de entrenamiento y m es el numero de entradas de cada
%patron.
%PS: Matriz de simulacion de rxm, r es el numero de patrones del
%conjunto de simulacion y m es el numero de entradas de cada patron.
%Li: Limite de iteraciones.
%Salidas:
%S : Matriz de las mismas dimensiones que PS y en donde se almacena la
%salida de la red.
%E: Escalar que devuelve el porcentaje de los patrones que se
%estabilizaron.
%Estab: Vector columna de dimension r con ceros en los indices de los
%patrones del conjunto de simulacion que no se estabilizaron y unos si
%se estabilizaron.

M=Aprendizaje_Hopfield(P); % Se crea la matriz de pesos a partir
                          % del conjunto de entrenamiento.

[r,c]=size(P);
S=zeros(r,c);
for it=1:Li                % Se simula hasta que se cumpla el numero de
    for k=1:r              % iteraciones.
        S(k,:)=PS(k,:)*M;
    end
end
S=hardlims(S);

```

```

        end
        if it==Li-1
            P1=S;
        end
        PS=S;
    end
    [E,Estab]=compara(S,P1);
end

```

A.2. Red LVQ

La siguiente función crea y simula la red LVQ, empleando las funciones predefinidas en el toolbox de redes neuronales de MATLAB.

A.2.1. Función principal: Redlvq

La función Redlvq, es la función principal que simula la red LVQ. Da como resultado una estructura de MATLAB denominada net que almacena la topología y pesos de la red que se creó, así como un vector que indica a qué clase pertenece cada patrón después de simular la red. El conjunto de entrenamiento para esta red se almacena de manera diferente que en la red de Hopfield, en este caso cada columna representa un patrón de entrada y el último elemento de cada columna representa la clase a la cual pertenece el patrón.

La regla de aprendizaje que usa esta red es la de Kohonen, identificada en el toolbox de redes neuronales como learnlvq1, la cual se describió en la sección 3.3.2.

```

function [net,Yc]=Redlvq(pD,pS,S1,PC,LR,LI)
%Redlvq: Funcion que crea y simula una red LVQ.
%Entradas:
    %pD: Conjunto de entrenamiento de (n+1)xm donde n es numero de entradas de cada
    %patron y el renglon n+1 contiene la clase a la cual pertenece el patron;
    %m es el numero de patrones del conjunto de entrenamiento.
    %pS: Conjunto de simulacion.
    %S1: Numero de neuronas en la capa de entrada.
    %PC: Vector fila de dimension c, donde c es el numero de clases y
    %almacena los porcentajes de cada clase
    %LR: Tasa de aprendizaje.
    %LI: Limite de iteraciones.
%Salidas:
    %net: Estructura tipo net que almacena la red.
    %Yc: Vector fila de dimension m(numero de patrones del conjunto de
    %simulacion) donde se almacena la salida de la red.
r=size(pD,1);
T=ind2vec(pD(r,:)); %Convierte el vector de clases en vector objetivo.
targets=full(T);
net=newlvq(pD(1:r-1,:),S1,PC,LR); % newlvq: crea una red LVQ, con la regla
    % de aprendizaje "learnlv1"
net.trainParam.epochs=LI; %Entrena la red de acuerdo al numero de
    %iteraciones indicadas.
net=train(net,pD(1:r-1,:),T);
Y=sim(net,pS(1:r-1,:)); % Simula la red despues de completar las
    % iteraciones.
Yc=vec2ind(Y); % Muestra el resultado de la simulacion,
end

```

A.3. Algoritmo Wake-Sleep

A.3.1. Función auxiliar: logsig1

La función logsig ya existe en MATLAB, solo que para minimizar el tiempo de ejecución del programa se creó la función logsig1.

```
function out = logsig1(in)
%logsig1: Funcion de transferencia logaritmica.
%Entrada:
    %in: Puede ser un escalar, un vector (renglon o fila), o una matriz de mxn, donde m
    %es el numero de entradas del patron y n es el numero de patrones (en el caso de que
    %sea una matriz) del conjunto de entrenamiento.
%Salida:
    %out: Puede ser un escalar, un vector (renglon o columna), o una matriz cuya dimension
    %depende de la entrada.
out = 1 ./ (1 + exp(-in));
end
```

A.3.2. Función auxiliar: sample1

Al igual que la función logsig, la función sample ya existe en MATLAB, pero hace cosas distintas a lo que se requiere para este ejemplo, por eso se creó la función sample1.

```
function r=sample1(p)
%sample1: Es una funcion estocastica que produce 1 con probabilidad p y
% 0 con probabilidad 1-p.
%Entrada:
    %p: Matriz de nxm o vector de dimension n, donde n es el numero de entradas
    %del patron y m es el numero de patrones del conjunto de entrenamiento.
    %Las entradas de p estan entre cero y uno.
%Salida:
    %r: Matriz o vector cuyas dimensiones son las mismas que la entrada,
    %y cuyos elementos son ceros y unos.
n=size(p,1);
r=rand(n,1)<p;
end
```

A.3.3. Función auxiliar: combinaciones

Con esta función se creó el conjunto de simulación que se empleó en la red de Hopfield, la red LVQ y la máquina de Helmholtz.

```
function [C]=combinaciones(N)
%combinaciones: Funcion que realiza todas las posibles combinaciones dado
%un vector o patron binario.
%Entrada:
%      N: Numero de entradas del patron.
%Salida:
%      C: Matriz con valores binarios con todas las posibles combinaciones
C=zeros(N,2^N);
for i=1:N
    k=2^(N-i);
    for j=1:2:2^i
        C(i,j*k+1:j*k+k)=1;
    end
end
end
```

A.3.4. Función auxiliar: prob_pgd

```
function[prob_pgd]=prob_pgd(bG,WG,VG,pD)
%prob_pgd: Funcion que calcula la distribucion de probabilidad generativa
%           de cada patron.
%Entradas:
%           bG: Matriz de pesos de Lx1 donde L es el numero de neuronas en la
%           capa de salida.
%           WG: Matriz de pesos de MxL donde M es el numero de neuronas en la
%           capa oculta y L el numero de neuronas en la capa de salida.
%           VG: Matriz de pesos de NxM donde N es el numero de neuronas en la
%           capa de entrada y M el numero de neuronas en la capa oculta.
%Salidas:
%           prob_pgd: Matriz de (n+1)xc donde el renglon n+1 contiene la pro-
%           babilidad generativa de cada patron, c es el numero de patrones.

L=size(bG,1);
M=size(WG,1);
N=size(VG,1);
prob_pgd=[pD(1:N,:);zeros(1,2^N)];
PGX=zeros(2^L,2^M);
PGY=zeros(2^L,2^M);
PGD=zeros(2^L,2^M);
x=combinaciones(L);
y=combinaciones(M);
for di=1:2^N
    d=pD(1:N,di);
    for xi=1:2^L
        for yj=1:2^M
            pgx=logsig1(bG);
            PGX_int=pgx.^x(:,xi).*(1-pgx).^(1-x(:,xi));
            PGX(xi,yj)=prod(PGX_int);
            pgy=logsig1(WG*x(:,xi)+ones(M,1));
            PGY_int=(pgy.^y(:,yj)).*(1-pgy).^(1-y(:,yj));
            PGY(xi,yj)=prod(PGY_int);
            pgd=logsig1(VG*y(:,yj)+ones(N,1));
            PGD_int=pgd.^d.*(1-pgd).^(1-d);
            PGD(xi,yj)=prod(PGD_int);
        end
    end
    PGXYD=PGD.*PGY.*PGX;           %Calculando la distribucion de proba-
                                %bilidad conjunta.
    prob_pgd(N+1,di)=sum(sum(PGXYD)); %Calculando la distribucion de proba-
    %                               %bilidad generativa.
end
end
```

A.3.5. Función auxiliar: KL

```
function[E]=KL(N,pD,pgD)
%KL: Funcion que calcula la diferencia entre la distr. de probabilidad p(d) y
%la distribucion de probabilidad generativa pG(d).
%Entradas:
%           N: Numero de neuronas de la capa de entrada
%           pD: Matriz donde cada columna es un patron con su respectiva probabilidad
%           pgD:Matriz en la cual cada columna es un patron con su respectiva
```

```

%           probabilidad generativa
%Salida:
%           E: Escalar que mide la diferencia en las distribuciones de probabilidad.

mini=1e-020;
kl=zeros(1,2^N);
for i=1:2^N
    if pD(N+1,i)==0
        pD(N+1,i)=mini;
    end
    if pgD(N+1,i)==0
        pgD(N+1,i)=mini;
    end
    kl(:,i)=(pD(N+1,i)*log(pD(N+1,i)/pgD(N+1,i)));
end
E=sum(kl);
end

```

A.3.6. Función principal: wake_sleep

```

function[bG,WG,VG,WR,VR,prob_pgD,E]=wake_sleep(N,M,L,epb,epW,epV,d,pD,LI,Lim_KL)
%wake_sleep: Funcion que implementa la maquina de Helmholtz con aprendizaje
%           wake-sleep.
%Entradas:
%N: Numero de neuronas en la capa de entrada.
%M: Numero de neuronas en la capa oculta.
%L: Numero de neuronas en la capa de salida.
%epb: Tasa de aprendizaje para la matriz de pesos bG que conecta a una
%     constante igual a 1 con la capa superior.
%epW: Tasa de aprendizaje para la matriz de pesos W.
%epV: Tasa de aprendizaje para la matriz de pesos V.
%d: Matriz de nxm, donde n es el numero de entradas del patron y m es
%   el numero de patrones que contiene el conjunto de entrenamiento.
%pD: Matriz que (n+1)xc, donde el renglon n+1 contiene la probabilidad
%   de cada patron y c es el numero de patrones posibles.
%LI: Escalar que contiene el numero de iteraciones.
%Lim_KL: Escalar que indica cada cuantas iteraciones se calculara la
%        diferencia entre la distribucion de probabilidad y distribuci-
%        on de probabilidad generativa.
%Salidas:
%bG: Matriz de pesos de Lx1 que conecta a la cte 1 con la capa de salida
%WG: Matriz de pesos de MxL que conecta a la capa de salida con la capa
%   oculta.
%VG: Matriz de pesos de NxM que conecta a la capa oculta con la capa de
%   entrada.
%VR: Matriz de pesos de MxN que conecta a la capa de entrada con la capa
%   oculta.
%WR: Matriz de pesos de LxM que conecta a la capa oculta con la capa de
%   salida.
%prob_pgD: Matriz de dimension (n+1)xc, donde el renglon (n+1) corres-
%   ponde a la probabilidad generativa de cada patron y c es el numero
%   de patrones.

```



```

%Se inicializan los pesos VG,WG,bG en cero.
bG=zeros(L,1);
WG=zeros(M,L);
VG=zeros(N,M);
VR=zeros(M,N);
WR=zeros(L,M);
r=size(d,2);
n=1;

for cont=0:LI
    if mod(cont,Lim_KL)==0
        [prob_pgD]=prob_pgD(bG,WG,VG,pD);
        E(1,n)=KL(N,pD,prob_pgD);
        n=n+1;
    end
    c= randi(r,1,1);
    %FASE WAKE
        y=sample1(logsig1(VR*d(:,c)+1)); %Activacion de las neuronas de la
                                        %capa oculta empleando la composicion
                                        %de las funciones de activacion.
        x=sample1(logsig1(WR*y+1)); %Activacion de las neuronas de la
                                        %capa superior empleando la composi-
                                        %cion de las funciones de activacion.
        chi=logsig1(bG); % chi, sigma y delta: variables que
                        %forman parte del gradiente para
                        %actualizar pesos generativos.

        sigma=logsig1(WG*x+1);
        delta=logsig1(VG*y+1);
    %Actualizando pesos generativos por la regla delta
        bG=bG+epb*(x-chi);
        WG=WG+epW*(y-sigma)*(x'+1);
        VG=VG+epV*(d(:,c)-delta)*(y'+1);
    %FASE SLEEP
        xs=sample1(logsig1(bG));
        ys=sample1(logsig1(WG*xs+1));
        ds=sample1(logsig1(VG*ys+1));
        sigma=logsig1(VR*ds+1);
        chi=logsig1(WR*ys+1);
    %Actualizando pesos de reconocimiento por la regla delta
        VR=VR+epV*(ys-sigma)*(ds'+1);
        WR=WR+epW*(xs-chi)*(ys'+1);
end
X=0:Lim_KL:LI;
plot(X,E,'d-r'); % Grafica la diferencia de la
                %distribuciones cada numero de ite-
                %raciones(indicadas por el usuario),
                %partiendo de la iteracion 0.

end

```


Apéndice B

Conjunto de entrenamiento

B.1. Conjunto de entrenamiento para la red LVQ

En la página siguiente se muestra la tabla con los 50 patrones que se emplearon como conjunto de entrenamiento para el ejemplo de la red LVQ, todos ellos al igual que el resto de los conjuntos de entrenamiento y simulación fueron introducidos en MATLAB mediante archivos *.mat que permiten el almacenamiento de variables de tipo matriz como sería nuestro caso.

Barras verticales. Clase 1	$p_1=(0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1)$
	$p_2=(0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1)$
	$p_3=(1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0)$
	$p_4=(1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0)$
	$p_5=(0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0)$
	$p_6=(1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1)$
Barras horizontales. Clase 2	$p_7=(0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)$
	$p_8=(1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1)$
	$p_9=(0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0)$
	$p_{10}=(1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0)$
	$p_{11}=(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1)$
	$p_{12}=(1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)$
Barras horizontales y verticales. Clase 3	$p_{13}=(1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1)$
	$p_{14}=(1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1)$
	$p_{15}=(1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1)$
	$p_{16}=(1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1)$
	$p_{17}=(1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1)$
	$p_{18}=(1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1)$
	$p_{19}=(1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0)$
	$p_{20}=(1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1)$
	$p_{21}=(0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1)$
	$p_{22}=(1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 1)$
	$p_{23}=(1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0)$
	$p_{24}=(0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 1 \ 1)$
	$p_{25}=(0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1)$
	$p_{26}=(1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0)$
	$p_{27}=(0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0)$
	$p_{28}=(1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0)$
	$p_{29}=(1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1)$
	$p_{30}=(0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)$
	$p_{31}=(1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0 \ 1)$
	$p_{32}=(0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1)$
	$p_{33}=(1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0)$
	$p_{34}=(1 \ 1 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1)$
	$p_{35}=(1 \ 0 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)$
	$p_{36}=(1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0)$
	$p_{37}=(0 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)$
	$p_{38}=(0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1)$
	$p_{39}=(1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0)$
	$p_{40}=(0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)$
	$p_{41}=(1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)$
	$p_{42}=(1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)$
	$p_{43}=(1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1)$
	$p_{44}=(1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0)$
	$p_{45}=(1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1)$
	$p_{46}=(1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1)$
	$p_{47}=(1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 1)$
	$p_{48}=(1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1)$
Ni horizontales ni verticales. Clase 4	$p_{49}=(1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1)$
	$p_{50}=(0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0)$

Tabla B.1: Conjunto de entrenamiento para la red LVQ.