UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA

MAESTRÍA EN SIMULACIÓN NUMÉRICA Y CONTROL

TRABAJO PRACTICO 2:

PERCEPTRONES SIMPLE, MULTICAPA Y BACKPROPAGATION

Presentada como requisito parcial para la aprobación del curso Sistemas Adaptativos y Redes Neuronales.

Estudiante:
Fredy Andrés Mercado Navarro
DNI: 94.872.342

Profesor titular:
Sergio Lew

Clases Prácticas:
Ana Laura Vadnjal

8 de diciembre de 2013
Buenos Aires, Argentina

Resumen

Un perceptrón es una red de capas de neuronas alimentadas hacia adelante, que puede tener varias entradas y varias salidas, donde las salidas son función de las entradas. En el perceptrón se busca actualizar los pesos sinápticos de forma iterativa a través del algoritmo de propagación del error hacia atrás (Backpropagation). Existen perceptrones simples, sin capas ocultas y perceptrones de varias capas. El perceptrón simple puede resolver solamente aquellos problemas que sean linealmente separables, esto es, problemas cuyas salidas estén clasificadas en dos categorías diferentes y que permitan que su espacio de entrada sea dividido en estas dos regiones por medio de un hiperplano de características similares. Un ejemplo de este tipo de problemas son las funciones lógicas OR y AND. Para aquellos problemas que no son linealmente separables se emplean perceptrones multicapa, como es el caso del aprendizaje de la función XOR.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Teoría

1.1. Perceptron Simple

Es un método de aprendizaje supervisado. Consiste de un proceso iterativo, donde vamos hallando términos w_{ij} mejores partiendo de un punto de partida arbitrario. El perceptrón simple solo posee una capa por definición y es una red con aprendizaje hacia adelante (feed forward). La matriz de pesos w_{ij} es, por definición, asimétrica, es decir, todas las conexiones son unidireccionales, lo que significa que no existe una función de energía.

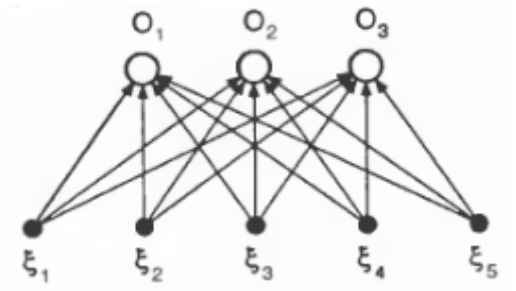

Figura 1.1: Perceptrón simple.

$$O_i = g\left(\sum_{k=1}^{N} w_{ik}\xi_k - \theta_i\right)$$

$$\underline{O} = \begin{bmatrix} O_1 \\ O_2 \\ ... \\ O_{NN} \end{bmatrix}$$

N: cantidad de entradas.
i: neurona i.
ξ_k^μ: patrón de entrada. Como vector es $\underline{\xi}^\mu$.
ζ_i^μ: patrón de salida particular (patrón objetivo).
O_i^μ: patrón de salida actual.
μ: número de patrón.

La salida deseada es:

$$O_i^\mu = \zeta_i^\mu$$

Los pesos w_{ij} se convierten en un vector de pesos $\underline{w} = (w_1, w_2, ..., w_N)$, una componente para cada entrada. Para cada patrón μ.

$$sgn(\underline{w} \cdot \underline{\xi_\mu}) = \zeta^\mu$$

$$sgn\left(\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \cdot \underline{P}\right) = \zeta^\mu$$

Utilizamos la función signo para obtener ± 1 en lugar de 0 o 1.

1.1.1. Algoritmo de aprendizaje

Este algoritmo es útil para problemas linealmente separables. Los pasos para actualizar los pesos sinápticos son:

1. Recorrer los patrones uno por uno.
2. Para cada patrón recorrer las unidades de salida una por una.
3. Para cada unidad de salida comparar si la salida actual es la deseada.
4. Si lo es no realizar ningún cambio.
5. Si no, aplicar la regla de Hebb, es decir, agregar a cada conexión algo proporcional al producto de la entrada y la salida deseada.

Dicho de otra forma:

$$w_{ik}^{new} = w_{ik}^{old} + \Delta w_{ik}$$
$$\Delta w_{ik} = \eta \left(\zeta_i^\mu - O_i^\mu\right) \xi_k^\mu$$

η: tasa de aprendizaje.

1.2. Perceptrón Multicapa

El perceptrón simple no aplica para redes hacia adelante (feed-forward) con capas intermedias (ocultas) entre las capas de entrada y salida. Una red con una sola capa oculta puede representar cualquier función Booleana, incluyendo funciones que no son linealmente separables como la función XOR, que trataremos en el punto práctico 2. Se usará el método de propagación del error hacia atrás para hacer que el perceptrón multicapa aprenda una función particular.

1.2.1. Propagación hacia atrás

Este algoritmo da una receta para ir cambiando los pesos w_{pq} en cualquier red hacia adelante. Las unidades escondidas se denotan por V_j, las entradas por ξ_k y las salidas por O_i. w_{jk} son conexiones entre las entradas y las unidades escondidas y W_{ij} son las conexiones entre las unidades escondidas y las de salida. i se refiere a unidades de salida, j a unidades escondidas y k a una terminal de entrada, μ indica el patrón, N la cantidad de unidades de entrada y p la cantidad de patrones de entrada (μ=1,2,...,p).

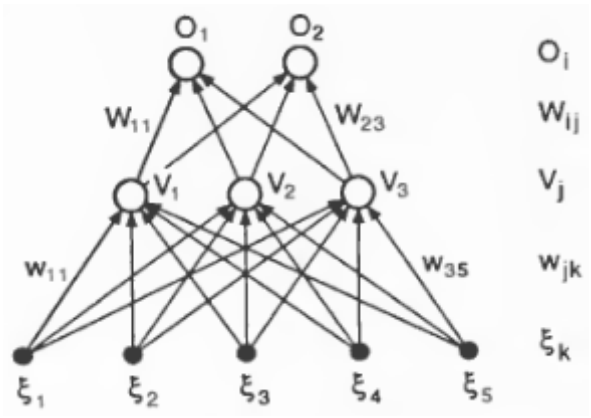

Figura 1.2: Red hacia adelante de dos capas. Notación para unidades y pesos.

Para un patrón μ, la unidad escondida j recibe una entrada neta

$$h_j^\mu = \sum_k w_{jk}\xi_k^\mu$$

y produce la salida

$$V_j^\mu = g(h_j^\mu) = sgn(h_j^\mu)$$

La unidad de salida i recibe entonces

$$h_i^\mu = \sum_j W_{ij}V_j^\mu$$

y produce para la salida final

$$O_i^\mu = g(h_i^\mu) = sgn(h_i^\mu)$$

Los umbrales pueden implementarse como una unidad de entrada extra fijada en 1 y conectada a todas las unidades de la red.

La medida del error es

$$E[w] = \frac{1}{2}\sum_{\mu i}(\zeta_i^\mu - O_i^\mu)^2$$

que es una función contínua y diferenciable de cada peso.

Para las conexiones que van de la última capa oculta hacia la capa de salida la regla del gradiente descendente resulta en:

$$\Delta W_{ij} = \eta \sum_{\mu} \delta_i^{\mu} V_j^{\mu}$$
$$\delta_i^{\mu} = g'(h_i^{\mu})[\zeta_i^{\mu} - O_i^{\mu}]$$

Para las conexiones entre la capa de entrada y la primera capa oculta tenemos:

$$\Delta w_{jk} = \eta \sum_{\mu} \delta_j^{\mu} \xi_k^{\mu}$$
$$\delta_j^{\mu} = g'(h_j^{\mu}) \sum_{i} W_{ij} \delta_i^{\mu}$$

La regla general es:

$$\Delta w_{pq} = \eta \sum_{patrones} \delta_{salida} \times V_{entrada}$$

Importante

- La regla de actualización es local. El cambio de los pesos para una conexión sólo depende de las cantidades de los dos extremos de esa conexión.

Es normal usar una función sigmoidea para la función de activación $g(h)$. Debe ser diferenciable y saturarse en ambos extremos.

$$g(h) = \tanh \beta h$$
$$g'(h) = \beta(1 - g(h)^2)$$

β a menudo es 1.

1.2.2. Procedimiento Back-propagation

Tomando un patrón μ a la vez. Consideremos una red con M capas $m = 1, 2, ..., M$ y usemos V_i^m para la salida de la i-ésima unidad en la m-ésima capa. V_i^o será sinónimo para ξ_i, la i-ésima entrada. m etiqueta capas, no patrones. w_{ij}^m es la conexión desde V_j^{m-1} hasta V_i^m. El procedimiento de propagación hacia atrás sigue los siguientes puntos:

1. Inicializar los pesos a valores aleatorios pequeños.

2. Seleccionar un patrón ξ_k^{μ} y aplicarlo a la capa de entrada ($m = 0$) de modo que

$$V_k^0 = \xi_k^{\mu} \ \forall \ k$$

3. Propagar la señal hacia adelante a través de la red usando

$$V_i^m = g(h_i^m) = g\left(\sum_j w_{ij}^m V_j^{m-1}\right)$$

para cada i y m hasta que las salidas finales V_i^M hayan sido calculadas.

4. Calcular los deltas para las capas de salida

$$\delta_i^M = g'(h_i^M)[\zeta_i^\mu - V_i^\mu]$$

comparando las salidas actuales V_i^M con las salidas deseadas ζ_i^μ para el patrón μ que está siendo considerado.

5. Calcular los deltas para las capas anteriores propagando el error hacia atrás

$$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m$$

para $m = M, M-1, ..., 2$ hasta que se calcule una delta para cada unidad (neurona).

6. Usar

$$\Delta w_{ij}^m = \eta \delta_i^m V_j^{m-1} \tag{1.1}$$

para actualizar todas las conexiones de acuerdo a

$$w_{ij}^{new} = w_{ij}^{old} + \Delta w_{ij} \tag{1.2}$$

7. Volver al paso 2 y repetir para el próximo patrón [4].

1.3. Error cuadrático medio

El cálculo del error cuadrático medio que se empleará para la estimación del error entre los datos de salida deseados y las salidas actuales es:

$$ECM = \frac{1}{n} \sum_{i=1}^{n} (O_i - \zeta_i)^2$$

Donde n es la cantidad de datos de los vectores de salida (4 para dos entradas y 16 para 4 entradas binarias), O_i es una componente del vector de la salida actual (estimada) y ζ_i es una componente del vector de la salida deseada.

Capítulo 2

Ejercicios teóricos

2.1. Punto 1

Explicar la siguiente ecuación:

$$C(p+1, N) = C(p, N) + D \tag{2.1}$$

donde $C(p, N)$ es la cantidad de combinaciones posibles de p puntos en posición general, pertenecientes a dos clases distintas, que pueden ser separados por un hiperplano en un espacio de dimensión N (Ver *Capacity of the simple perceptron*, Hertz, Krogh y Palmer, pág. 113).

2.1.1. Desarrollo

La ecuación 2.1 surge en el contexto de la determinación del poder computacional de un perceptrón a raíz de la siguiente pregunta:

Cuántas dicotomías es posible encontrar si lanzo una distribución aleatoria de p puntos en un espacio de N dimensiones, si cada punto puede tomar una de dos características (clases)?

Entiéndase por dicotomía a un estado de los p puntos que pueda ser dividido por un hiperplano, entendiendo que un hiperplano es un sub-espacio de dimensión $(N-1)$. Si los puntos están en un espacio bidimensional $(N = 2)$, el hiperplano correspondiente es una línea recta que separa dos grupos de puntos. Cada grupo con una característica distintiva.

La respuesta a la pregunta anterior se dotó con el símbolo $C(p, N)$, donde p es la cantidad de patrones de entrada (a almacenar) y N la cantidad de unidades o neuronas de entrada, que corresponde a la dimensión del problema. Para poder responder a la pregunta correctamente se utiliza el método de inducción matemática, donde se demostrará el cálculo para el aumento de las dicotomías para un paso de (p) a $(p+1)$ patrones, con N constante, así:

1. Suponer que tenemos un grupo de p patrones (puntos) en un espacio de dimensión N. La cantidad de dicotomías que tendría este grupo de puntos estaría dada entonces por $C(p, N)$.

2. Supongamos ahora que al grupo de puntos anterior agregamos otro punto (patrón) P que se encuentre en posición general, al igual que los demás. Estar en posición general indica que el punto posee independencia lineal y un hiperplano que pase por el origen no cortará más de un punto en cierta posición.

3. Ahora, algunas de las dicotomías del grupo original de p puntos pueden conseguirse con hiperplanos que pasen por el origen O y por el punto P. A ésta cantidad de dicotomías se le llamará D.

4. Para cada una de las D dicotomías hay dos nuevas dicotomías, la primera si corremos el punto infinitesimalmente hacia una clase y la segunda si lo corremos hacia la otra clase. Esto ocurre porque cuando los puntos están en posición general, cualquier hiperplano que pase a través de P permite clasificarlo sin afectar la dicotomía de los p puntos que estaba dada por ese mismo hiperplano.

5. Para las $C(p, N) - D$ dicotomías restantes, los dos nuevos posibles conjuntos que incluyen P deben ser separables, y como consecuencia, habrá una dicotomía nueva para cada dicotomía de los p puntos.

Los puntos anteriores nos llevan a que el número $C(p + 1, N)$ de dicotomías lineales del grupo que incluye el punto P está dado por:

$$C(p + 1, N) = C(p, N) - D + 2D = C(p, N) + D$$

Recordemos que D es la cantidad de dicotomías lineales del grupo de puntos p que pudieron haber sido separadas por un hiperplano que pase a través del punto P, pero este número es simplemente $C(p, N-1)$, porque al imponer la condición de que el hiperplano pase por el punto P estamos convirtiendo el problema en uno de dimensión (N-1), ya que los puntos podrían separarse (clasificarse) sin necesidad de que se encuentren en el espacio de dimensión N, simplemente proyectándolos en un espacio de dimensión (N-1). Una muestra de esto es tener un espacio de dimensión 3 y proyectar los puntos sobre un espacio de dimensión 2 (un plano perpendicular a la línea que une el origen y el punto P), que es justo lo que se observa en la Figura 5.12 del texto de Hertz, Krogh y Palmer [4].

La observación anterior permite obtener la expresión:

$$C(p + 1, N) = C(p, N) + C(p, N - 1)$$

La explicación de la ecuación anterior es un paso intermedio para finalmente demostrar, iterando para $p, p - 1, p - 2, ..., 1$ que la cantidad de dicotomías posibles para una cantidad de puntos aleatorios (patrones) p en un espacio de N dimensiones es:

$$C(p, N) = 2 \sum_{i=0}^{N-1} \binom{p-1}{i}$$

Este punto teórico fue desarrollado con ayuda de las referencias [2], [3] y [6].

2.2. Punto 2

Demostrar que cualquier función binaria puede ser implementada con algún perceptrón multicapa.

2.2.1. Desarrollo

Los perceptrones multicapa integran las capacidades de los perceptrones simples, que solo tienen una capa de entrada y una de salida. Estos perceptrones simples pueden modelar funciones con entradas linealmente separables, mientras que los perceptrones multicapa pueden modelar funciones que no son linealmente separables, como la función OR exclusiva.

Dentro del grupo de las 16 funciones binarias que existen tenemos un grupo de funciones con salidas que son linealmente separables y otro grupo con funciones con salidas que no lo son. Sabemos que el perceptron simple puede modelar las que son linealmente separables, y esta ventaja se aprovecha para modelar las demás funciones binarias que no son linealmente separables. Una clave para modelar las funciones no separables son las funciones NOR y NAND, las cuales son ambas linealmente separables y pueden ser representadas por un perceptrón simple. Al combinar estas funciones en dos capas es posible generar las demás funciones booleanas que no pudieron ser modeladas con el perceptrón simple. Esto nos demuestra que cualquier función binaria puede ser representada por un perceptrón multicapa, mejor aún, es posible hacerlo con un perceptrón de una sola capa oculta [1].

2.3. Punto 3

El borde de decisión (decision boundary) de un perceptrón simple es un hiperplano. Explique cómo es el límite de decisión de un perceptrón de dos o tres capas (siempre con una sola neurona de salida).

2.3.1. Desarrollo

El perceptrón simple (con una capa de entrada y una de salida) forma bordes de decisión de medio plano, separando los patrones de entrada por clases o grupos. A continuación se explica como son los bordes de decisión para un perceptron de dos y tres capas con una sola neurona de salida, ya que el comportamiento de los perceptrones con varias salidas es más complejo debido a que las regiones de decisión están bordeadas por curvas suaves en lugar de segmentos de líneas rectas.

Bordes de decisión para un perceptrón de dos capas

Un perceptrón de dos capas puede formar regiones de decisión convexas en el espacio ocupado por los patrones de entrada. Estas regiones incluyen polígonos convexos y regiones no limitadas como las que se aprecian en la fila del medio de la Figura 2.1. El término convexo significa que cualquier línea que una puntos sobre el borde de una región sólo pasa a través de puntos por dentro de esa misma región. Las regiones convexas están formadas por la intersección de regiones de medio plano formadas por cada nodo en la primera capa del perceptrón multicapa. Cada nodo de la primera capa se comporta como un perceptrón simple. Las regiones convexas tienen a lo sumo tantos lados como nodos tenga la primera capa (ver fila del medio y columna 5 de la Figura 2.1).

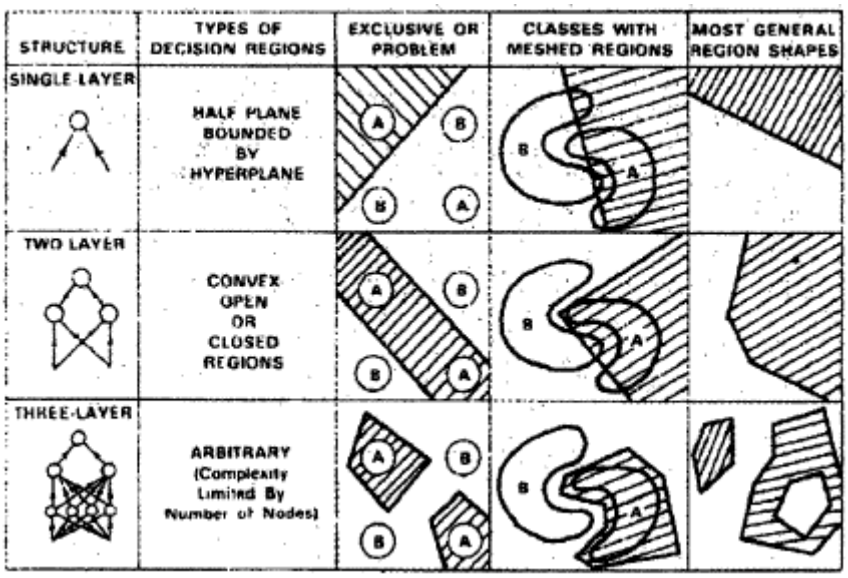

Figura 2.1: Tipos de bordes de decisión que pueden ser formados por perceptrones simples y perceptrones de varias capas con una y dos capas de unidades escondidas. Lo anterior para dos entradas y una sola salida. Las partes sombreadas corresponden a regiones de decisión para la clase A. Tomada de [5].

Bordes de decisión para un perceptrón de tres capas

Un perceptrón de tres capas puede formar regiones de decisión arbitrariamente complejas. Esto puede demostrarse construyendo la red pensando en particionar la región de decisión en hipercubos pequeños (cuadrados, cuando se tienen dos entradas). Cada hipercubo requiere 2N nodos en la primera capa, uno para cada lado del hipercubo, y un nodo en la segunda capa que tome la salida lógica AND de los nodos de la primera capa. Los hipercubos se asignan a las regiones de decisión apropiadas conectando la salida de los nodos de la segunda capa solo al nodo de salida que corresponde a la región de decisión en la que se encuentra el hipercubo de ese nodo y realizando una operación logica OR en cada nodo de salida. Este procedimiento de construcción es capaz de generar las regiones desconectadas y no convexas que se muestran en la fila más baja de la Figura 2.1.

Finalmente, los perceptrones de tres capas pueden generar, a diferencia de los de dos capas, una región de decisión totalmente cerrada, también puede tener varias regiones del mismo tipo cerradas y aisladas entre si, y regiones cerradas dentro de regiones de clase diferente, como también se puede apreciar en la columna 5, fila 4 de la Figura 2.1 [5].

Capítulo 3

Ejercicios prácticos

3.1. Punto 1

Implemente un perceptrón simple que aprenda la función lógica OR de 2 y de 4 entradas.

3.1.1. Desarrollo

En el Cuadro 3.1 se observa la tabla de verdad para la función OR de 2 entradas, donde SALIDA=1 si A=1 o B=1, y en el Cuadro 3.2 la tabla de verdad para 4 entradas, donde SALIDA=1 si A, B, C o D son iguales a 1.

Se elaboró un programa en Matlab para aprender estas dos funciones. El usuario elige entre correr para 2 o 4 entradas y elige el número máximo de iteraciones que permitirá que corra el programa, al igual que el valor de la constante de aprendizaje. Los pesos iniciales a partir de los cuales comenzará el proceso de aprendizaje y actualización de pesos se toman aleatorios con la función *rand*(). Por cada iteración se corre un ciclo que barre los patrones enseñados unos por uno. Si la salida de cada patrón es diferente a la salida deseada entonces se actualiza el vector de pesos como indican las ecuaciones 1.1 y 1.2.

El proceso descrito anteriomente se repite hasta que no existe ninguna diferencia entre la salida deseada y la salida actual. Se empleó la función signo, lo que garantiza que el error podrá descender hasta cero para este caso donde tenemos una función binaria. Para el caso práctico de este ejercicio representamos el 0 por el -1 y el 1 por el +1. A todas las entradas de este punto práctico y de los siguientes se les adicionó una entrada fija en 1 para tener en cuenta el umbral. Los pesos obtenidos para el caso de 2 entradas son $w_1 = 0{,}7502$, $w_2 = 0{,}2344$ y $w_3 = 0{,}6387$. Para el caso de 4 entradas se obtuvieron

Patrón	Entrada A	Entrada B	Salida A o B
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	1

Cuadro 3.1: Tabla de verdad - Puerta OR de 2 entradas.

$w_1 = 0{,}1816$, $w_2 = 0{,}1655$, $w_3 = 0{,}5952$, $w_4 = 0{,}3869$ y $w_5 = 1{,}0898$. Con estos pesos se podría comprobar el aprendizaje de la función OR de 2 y 4 entradas.

En la Figura 3.1 se observa la clasificación del perceptron simple para la función OR de 2 entradas, donde se tienen 4 puntos en el espacio de clases de dos dimensiones, +1 y -1. Aquí se observa que el perceptrón logra clasificar los patrones (puntos) de entrada, separando aquellos que satisfacen la función de los que sí lo hacen. En este caso, graficamos la línea normal al vector de pesos, que separa una clase (-1) de la otra (+1).

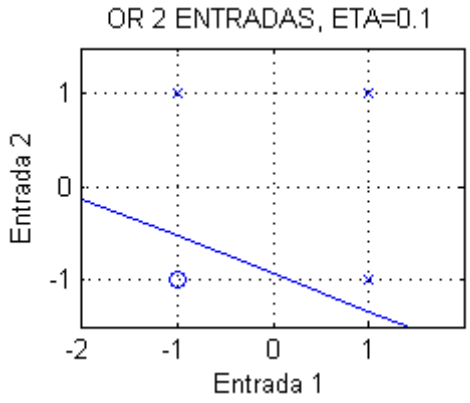

Figura 3.1: Clasificacion de la función OR de 2 entradas.

3.1.2. Conclusión

Se logró implementar el perceptrón simple para la función OR de 2 y 4 entradas. Para ambos casos el vector de pesos $\underline{w}$ converge en pocas iteraciones. Para 2 entradas en 3 iteraciones o menos, y para 4 entradas en 7 iteraciones o menos. Esto depende de las componentes aleatorias iniciales de $\underline{w}$.

3.2. Punto 2

Implemente un perceptrón multicapa que aprenda la función lógica XOR de 4 entradas (utilizando el algoritmo Backpropagation).

3.2.1. Desarrollo

Se implementará también el aprendizaje de la función XOR para 2 entradas, aunque el ejercicio no lo pida. En el Cuadro 3.3 se encuentra la tabla de verdad para la función lógica XOR de 2 entradas y en el Cuadro 3.4 la tabla de verdad para la función de 4 entradas. Para dos entradas la salida es igual a 1 si A y B son diferentes y 0 si A y B son iguales. La función XOR es asociativa, así que para 4 entradas la salida es igual a 1 si la suma de entradas iguales a 1 es impar.

Para el aprendizaje de la función XOR de 2 y 4 entradas se elaboró un programa de MATLAB que lee los números de entrada reemplazando ceros por -1. Se adiciona también una entrada fija siempre en 1 para reemplazar el uso de un umbral. Las constantes η y

Patrón	A	B	C	D	Salida A o B
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	1	0	1
4	0	0	1	1	1
5	0	1	0	0	1
6	0	1	0	1	1
7	0	1	1	0	1
8	0	1	1	1	1
9	1	0	0	0	1
10	1	0	0	1	1
11	1	0	1	0	1
12	1	0	1	1	1
13	1	1	0	0	1
14	1	1	0	1	1
15	1	1	1	0	1
16	1	1	1	1	1

Cuadro 3.2: Tabla de verdad - Puerta OR de 4 entradas.

Patrón	A	B	Salida A o B
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

Cuadro 3.3: Tabla de verdad - Puerta XOR de 2 entradas.

Patrón	A	B	C	D	Salida A o B
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	1	0	1
4	0	0	1	1	0
5	0	1	0	0	1
6	0	1	0	1	0
7	0	1	1	0	0
8	0	1	1	1	1
9	1	0	0	0	1
10	1	0	0	1	0
11	1	0	1	0	0
12	1	0	1	1	1
13	1	1	0	0	0
14	1	1	0	1	1
15	1	1	1	0	1
16	1	1	1	1	0

Cuadro 3.4: Tabla de verdad - Puerta XOR de 4 entradas.

β se mantienen constantes, al igual que el número de neuronas de la única capa oculta que se utilizó para la red del perceptrón. El usuario define también un error cuadrático medio mínimo que debe mejorarse para detener la corrida del ciclo de iteraciones de actualización de los pesos de las conexiones entre neuronas del perceptrón.

En términos generales, el algoritmo calcula la salida de las neuronas de la capa oculta y luego la salida de la neurona de la capa de salida. Luego calcula un delta por neurona para cada capa. Si el error cuadrático medio entre las salidas deseadas y las actuales es mayor que el permitido el ciclo se repite hasta que el ECM actual sea menor que el ECM permitido.

Para el caso de la función XOR de 2 entradas, las entradas al perceptrón son 3. Como se empleó solo una capa oculta la cantidad de pesos entre la capa de entrada y la de salida es, para el ejemplo de la Figura 3.2, de $3 \times 6 = 18$ pesos, mientras que los pesos de la capa oculta hacia la capa de salida son sólo 6, uno por neurona. Para el caso de la función XOR de 4 entradas, las entradas al perceptrón serían 5, los pesos entre la capa de entrada y la oculta $5 \times 10 = 50$ y de la capa oculta hacia la capa de salida 10 (ver variables W_1 y W_2 en el programa de Matlab).

En la Figura 3.2 se observa como progresa el ECM para el aprendizaje de la función XOR de 2 entradas y en la Figura 3.3 para la función XOR de 4 entradas.

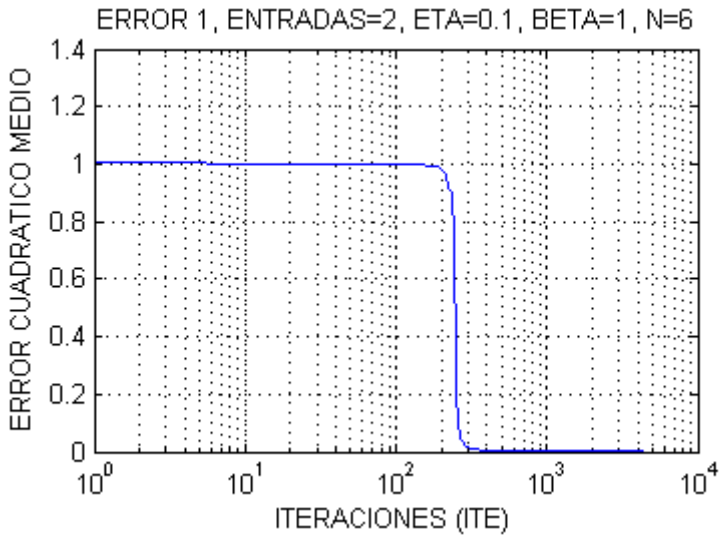

Figura 3.2: Curva de error cuadrático medio con eje x logarítmico para perceptron simple de 2 entradas.

3.2.2. Conclusión

Se implementó un algoritmo para el aprendizaje de la función XOR de 2 y 4 entradas para un perceptron con una sola capa oculta. El algoritmo de propagación del error hacia atrás calcula un delta de peso para cada neurona de cada capa. El algoritmo diseñado permite variar el número de neuronas de la capa oculta, pero no el número de capas. Mientras menor es el ECM permitido mejores son las aproximaciones de las salidas ac-

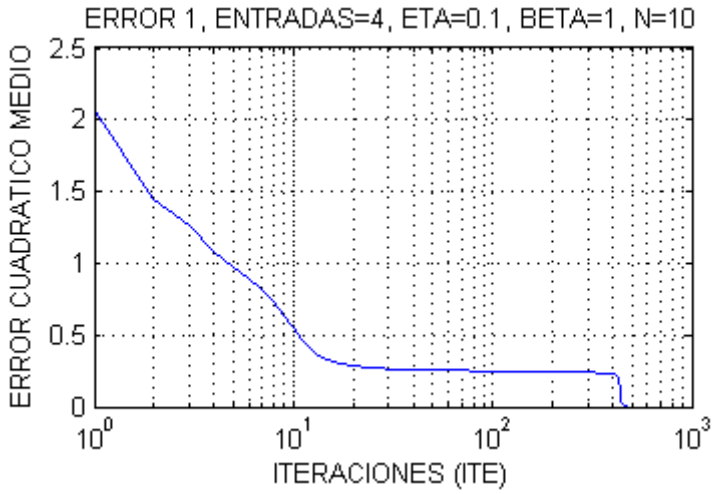

Figura 3.3: Curva de error cuadrático medio con eje x logarítmico para perceptron simple de 4 entradas.

tuales comparadas con las deseadas y mejores las estimaciones de los pesos de cada capa de conexiones que llevan al aprendizaje de la función.

3.3. Punto 3

Implemente una red con aprendizaje Backpropagation que aprenda la siguiente función:

$$f(x, y, z) = \text{sen}\,(x) + \cos\,(y) + z$$

donde x e y están en el intervalo $[0, 2\pi]$ y z está en el intervalo $[-1, 1]$.

3.3.1. Desarrollo

La función tiene 3 entradas (4 con el umbral), x, y, z y una salida $f(x, y, z)$. El propósito del ejercicio es enseñar a la red uno por uno todos los patrones, que serían combinaciones de x, y, z e ir corrigiendo los pesos con base en la diferencia entre $f(x, y, z)$ y la salida actual. A continuación se resume lo que se realizó.

En primer lugar, el algoritmo que se utilizó para el ejercicio es en escencia el mismo algoritmo de un perceptrón multicapa feedforward con backpropagation del punto 2. Los parámetros que determina el usuario para correr el programa son: las constantes η y β, un número máximo de iteraciones (MAXITER), el número de neuronas de la única capa oculta (NNEURONAS), el error cuadrático medio permisible (MIN_ECM), la cantidad de datos a evaluar por cada eje coordenado (NDATOS) y un número (PROD) que tiene la función de escalar los valores de $f(x, y, z)$ enseñados a un intervalo $[-1, 1]$, ya que estamos utilizando la función sigmoidea tanh para el umbral. El número usado para escalar $f(x, y, z)$ enseñado es $PROD = 1/3$ y el usado para escalar de vuelta la salida es su inverso, es decir 3, dado que el valor máximo de $f(x, y, z)$ es $+3$ y el valor mínimo es -3.

La expresión utilizada para escalar $f(x, y, z)$ es entonces:

Parámetro	Valor
Constante de aprendizaje η	0.1
Constante (steepness parameter) β	1
Limite de iteraciones MAXITER	10000
Neuronas de capa oculta NNEURONAS	20
Error permisible MIN_ECM	0.001
Datos por eje coordenado NDATOS	10
Total de patrones enseñados a la red	1000
Número de entradas (coordenadas+umbral)	3+1=4

Cuadro 3.5: Parámetros utilizados para el aprendizaje de la función.

$$f(x, y, z)_{escalada} = (\sin x + \cos y + z) \times PROD$$

A diferencia del ejercicio 2, en éste ejercicio le enseñamos a la red 1000 patrones (10x10x10 datos), donde tendremos una entrada por cada coordenada y otra fija en 1 para el umbral. La ejecución del programa sigue el mismo algoritmo del ejercicio 2, donde se realiza una actualización de pesos cada vez que se enseña un nuevo patrón. Al final, el programa sale del ciclo iterativo cuando el Error Cuadrático Medio desciende por debajo de MIN_ECM.

3.3.2. Resultados

Para demostrar la funcionalidad del programa para aprender una función se corrió un ejercicio con los parámetros consignados en el Cuadro 3.5.

En la Figura 3.4 se aprecia el gráfico logaritmico de la convergencia del error cuadrático medio en función del número de iteraciones.

Para comprobar lo anterior se compararon gráficamente los valores del campo escalar $f(x, y, z)$ manteniendo constantes dos de las tres variables, lo que da como resultado un gráfico 2D que corresponde al valor de la función sobre una línea recta producto de la intersección de los dos planos que define cada una de las coordenadas que se deja constante.

En la Figura 3.5 se aprecia el gráfico de $f(x, y, z)$ manteniendo constantes z e y en el intervalo $0 \leq x \leq 2\pi$. Los valores de la salida deseada y la salida actual coinciden bastante bien. La forma de la función que predomina es claramente la de la función seno, que es la que transforma los valores de x.

La Figura 3.6 muestra resultados similares a los de la Figura 3.5. Aquí la forma de la función que predomina es la de la función coseno, que es aquella que transforma los valores de Y, mientras que la Figura 3.7 muestra también buena coincidencia entre los valores deseados y la salida del perceptrón. Todo indica que los pesos fueron modificados exitosamente y el aprendizaje del perceptrón se realizó correctamente.

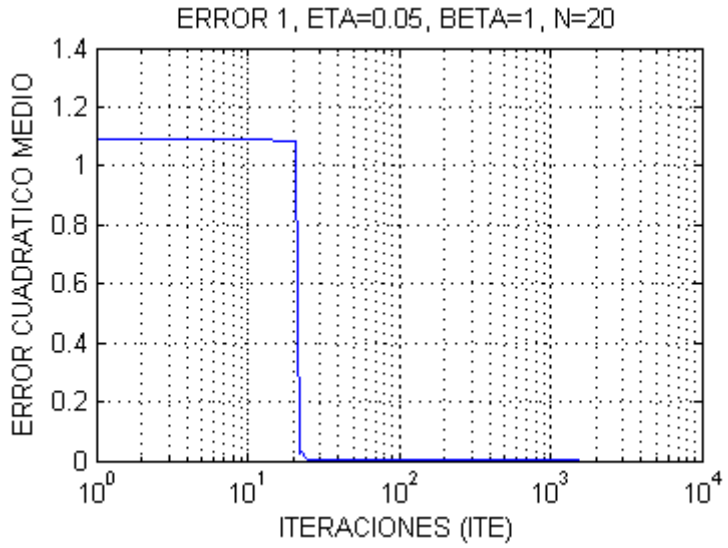

Figura 3.4: Curva de error cuadrático medio con eje x logarítmico para perceptron multicapa de 4 entradas aprendiendo una función f(x,y,z).

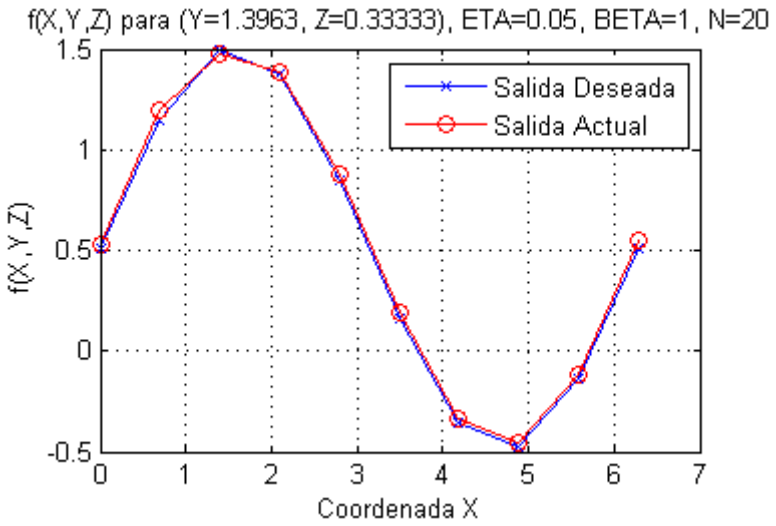

Figura 3.5: Curvas de f(x,y,z) deseada y de la salida actual del perceptron para z e y constantes. $0 \leq x \leq 2\pi$.

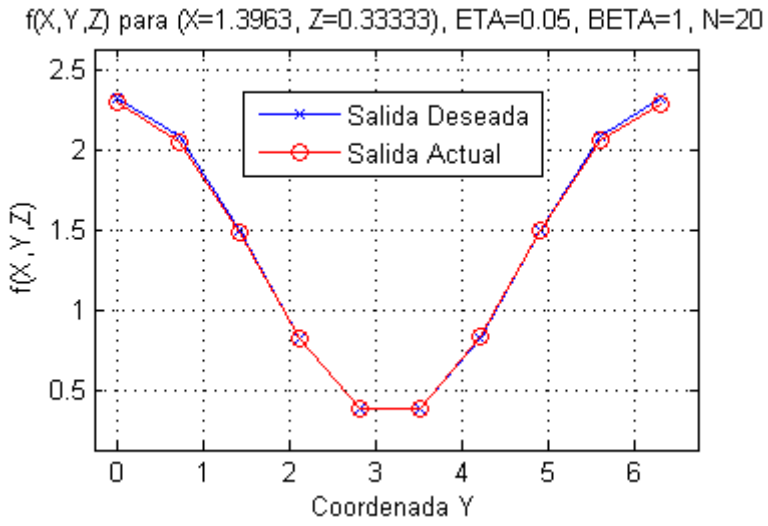

Figura 3.6: Curvas de f(x,y,z) deseada y de la salida actual del perceptron para x y z constantes. $0 \leq y \leq 2\pi$.

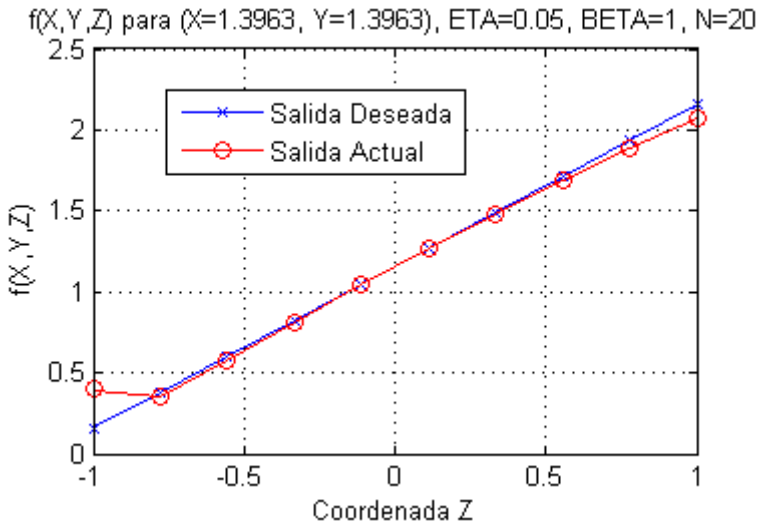

Figura 3.7: Curvas de f(x,y,z) deseada y de la salida actual del perceptron para x e y constantes. $-1 \leq z \leq 1$.

Capítulo 4

Comentarios y Conclusiones

- La cantidad de dicotomías que se pueden obtener a partir de una distribución aleatoria de patrones, para un perceptrón simple, está dada por una expresión que relaciona la cantidad de patrones de entrada y la cantidad de unidades de la primera capa. La ecuación fue deducida por inducción matemática teniendo en cuenta las dicotomías que resultan al adicionar un nuevo patrón e imponer que el hiperplano pase por el punto que define el estado de dicho patrón y el origen. Se aprovecha la proyección de los puntos sobre un plano de dimensión (N-1) para concluir que las dicotomías que se suman a las ya conocidas son las que resultan de hallar las dicotomías para el mismo número de puntos (patrones) pero con una dimensión menor (N-1).

- Los bordes de decisión del perceptron de dos capas pueden formar regiones de decisión convexas abiertas o cerradas, mientras que los bordes de decisión del perceptron de tres capas pueden formar regiones arbitrariamente complejas que pueden tener zonas de decisión aisladas y también zonas excluidas dentro de una región de decisión cerrada.

- El aprendizaje del perceptrón multicapa depende de parámetros de aprendizaje como η y β. Mientras menor sea la constante de aprendizaje más tiempo le tomará aprender a la red, sin embargo, esto garantiza una mejor aproximación de la solución deseada al emplear pasos de iteración más cortos que facilitan hallar mínimos locales.

- La facilidad de aprendizaje del perceptrón depende también del número de neuronas de las capas ocultas y del número de capas. En este trabajo práctico se demostrón que las funciones lógicas OR de 2 y 4 entradas pueden se aprendidas por un perceptrón simple, y que la función XOR de 2 y 4 entradas puede ser aprendida por un perceptrón multicapa con una sola capa oculta.

- El perceptrón con una sola capa oculta y función de activación sigmoidea tanh puede aprender una función contínua de varias variables. Dado que la salida del perceptrón estudiado va entre -1 y 1 se escaló la función de entrada para enseñar a la red valores que estuvieran dentro de ese intervalo. Luego se escaló de vuelta la salida para obtener los valores originales de $f(x, y, z)$.

Bibliografía

[1] Coppin, B. *Artificial Intelligence Illuminated.* Jones and Bartlet Publishers, 2004.

[2] Engel, A., and den Broeck, C. V. *Statistical Mechanics of Learning.* Cambridge University Press, 2001.

[3] Hassoun, M. H. *Fundamentals of Artificial Neural Networks.* The MIT Press, 1995.

[4] Hertz, J., Krogh, A., and Palmer, R. G. *Introduction to the theory of neural computation.* Addison-Wesley, 1990.

[5] Lippmann, R. P. An introduction to computing with neural nets. *IEEE ASSP Magazine 3*, 4 (1987), 4–22.

[6] Sampolinsky, H. *Introduction: The Perceptron.* MIT, 2013.