Introducción a las Redes Neuronales mediante el paquete neuralnet

El paquete *neuralnet* es un paquete de muy sencilla utilización que permite la aplicación, visulaización e implementación de redes neuronales. En este trabajo se detallan algunas de sus características generales, se muestran algunos ejemplos de uso junto con el detalle de algunas de sus fortalezas y sus limitaciones.

Se entiende por *Red Neuronal* a un conjunto de técnicas cuyo origen viene de la inteligencia artificial. En su versión más básica, el modelo más simple de red neuronal es lo que se conoce como *Perceptron* que no es más que una función de tipo signoidal cuya teoría fue desarrollada por *Frank Rosenblatt* a finales de los 50, la cual está basada en el modelo biológico de *Neurona de Mc Culloch y Pitts* de principios de los años 40.

$$\text{Función signoidad: } \text{sgn(x)} = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Esta técnica desarrollada en 1958, fué puesta en duda cuando *Minsky* y *Papert* en 1969 muestran por un lado que el un simple *Perceptron* no puede resolver problemas no lineales y aunque la combinación de varios perceptrones si podría hacerlo, sin embargo hubo que esperar hasta mitad de la década de los 80 para que se tengan desarrollados los *Algoritmos de Propagación de Errores hacia Adelante* que si permiten ya estimar los pesos de las unidades de las capas intermedias.

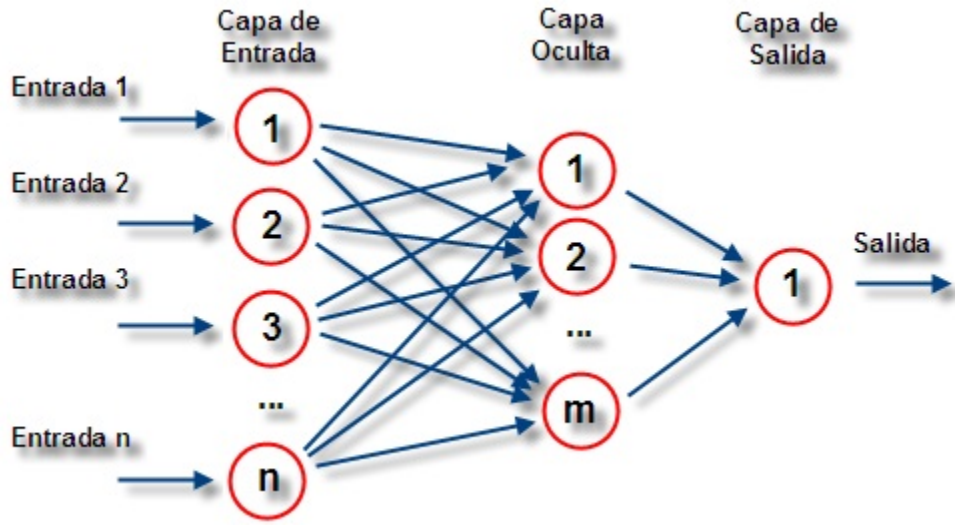

Figure 1: Esquema de Red Neuronal con una capa oculta

Así si tomamos los ejemplos clásicos para implementar los operadores lógicos AND, OR y XOR en un espacio de inputs de dimensión 2 cuya tabla de verificación se implementan con los siguientes vectores en R:

```
input1 = c(1,1,0,0)
input2 = c(1,0,1,0)
outputAND = c(1,0,0,0)
outputOR  = c(1,1,1,0)
outputXOR = c(0,1,1,0)

datos1 = cbind(input1,input2,outputAND)
```

```
datos2 = cbind(input1,input2,outputOR)
datos3 = cbind(input1,input2,outputXOR)
```

Se obtienen los conjuntos de datos *datos1*, *datos2* y *datos3* que vienen a implementar respectivamente cada uno de dichos operadores lógicos. La pregunta es ¿Hasta qué punto dichos vectores pueden ser aprendidos correctamente por una red neuronal debidamente entrenada? Se observa precisamente el fenómeno de *Minksy* y *Paper*, si se entrenan para cada conjunto de datos unas 100 redes neuronales sin capa oculta según el siguiente código:

```
library(neuralnet)

matriz1 <- matrix(nrow = 4, ncol = 100)
matriz2 <- matrix(nrow = 4, ncol = 100)
matriz3 <- matrix(nrow = 4, ncol = 100)

for (i in 1:100){

  set.seed(i)
  modelo1 <- neuralnet(outputAND ~ input1 + input2, data = datos1)
  matriz1[,i] = as.vector(prediction(modelo1)$rep[,3])
  modelo2 <- neuralnet(outputOR ~ input1 + input2, data = datos2)
  matriz2[,i] = as.vector(prediction(modelo2)$rep[,3])
  modelo3 <- neuralnet(outputXOR ~ input1 + input2, data = datos3)
  matriz3[,i] = as.vector(prediction(modelo3)$rep[,3])

}
```

Se observa que uno de los ejemplos, el correspondiente al XOR, no es rasonablemente "bien aprendido" tal y como se observa en los siguientes resultados:

Resultados de Simulaciones

AND	Media	Varianza	OR	Media	Varianza
0	-0.0965	0.0069	0	0.0032	0.0001
0	0.0947	0.0082	1	0.9798	0.0002
0	0.0884	0.0048	1	0.9775	0.0002
1	0.9084	0.0312	1	1.0413	0.0004

XOR	Media	Varianza
0	0.2997	0.0503
1	0.5797	0.0209
1	0.5974	0.0220
0	0.5342	0.0160

En este sentido, un resultado que prevé justamente este comportamiento es el siguiente, siguiendo la línea desarrollada en [Fine, T. L. 1999]

Teorema 1 (Cota superior)

El número de dicotomías linealmente separables de un conjunto de datos S (que en este caso está formado por los 4 inputs de dimensión 2), está acotado superiormente por:

$$D(n,d) = \begin{cases} 2\sum_{i=0}^{d} \binom{n-1}{i} & \text{si } n > d+1 \\ 2^n & \text{si } n \leq d+1 \end{cases}$$

Dado que en el caso bajo estudio se tiene que $n = 4$ y $d = 2$, se deduce que $D(n,d) = 14 < 2^4 = 16$, y por tanto, no todas las dicotomías posibles pueden ser separables. Nótese que los puntos del conjunto de datos son los vértices del cuadro de lado 1 y vértices:

$$S = \{(0;0), (0;1), (1;0), (1;1)\}$$

En este sencillo caso, llama la atención que cuando se trata de separar en planos distintos los puntos de vértices opuestos, como son los pares: (0;1), (1;0) y (0;0), (1;1) esto no se logra adecuadamente con un perceptrón y eso que los puntos están en lo que se denomina *Posición General*.

Definición 1 (Posición General)

Se dice que n puntos en un espacio de dimensión d, están en posición general, cuando para ningún k entero con 1 < k < d+2, k de esos puntos estarían contenidos en un sub-espacio afín de dimensión k - 2.

Así pues, en los caso de *Posición General*, se cumpliría que se alcanza el máximo número de linealidades separables que se podría tener, pero a pesar de eso, en el caso que aquí se considera, de tan sólo 4 puntos y de dimensión 2, algunas de las 16 posibilidades no se logran (si no la crítica al perceptrón planteada anteriormente no habría tenido lugar) y además, si la citada condición de *Posición General* no se cumpliese, dicha cantidad sería aún menor.

Teorema 2 (Cota superior estricta)

Si dado un conjunto S de n puntos, se tiene que éstos no están en posición general, entonces el número de dicotomías linealmente separables es estrictamente menor a la cota dada en el Teorema 1

Funcionamiento de una Red Neuronal: Un ejemplo sencillo

A continuación se trata de mostrar los conceptos anteriores mediante una red neuronal de una única capa proporcionada por la librería R *neuralnet*. Obsérvese que en este caso, no se hace uso de perceptrones en sí, sino de funciones continuas como se verá, esto no supondrá mayor problema como se analiza más adelante, porque muchos de los resultados obtenidos para el caso discreto de un perceptrón se mantienen para el caso continuo.

Para comenzar se va a tratar de separar en 2 grupos unos 6 puntos que están en posición general en un espacio de dos dimensiones $d = 2$. Para ello se considera el siguiente conjunto de puntos en el plano:

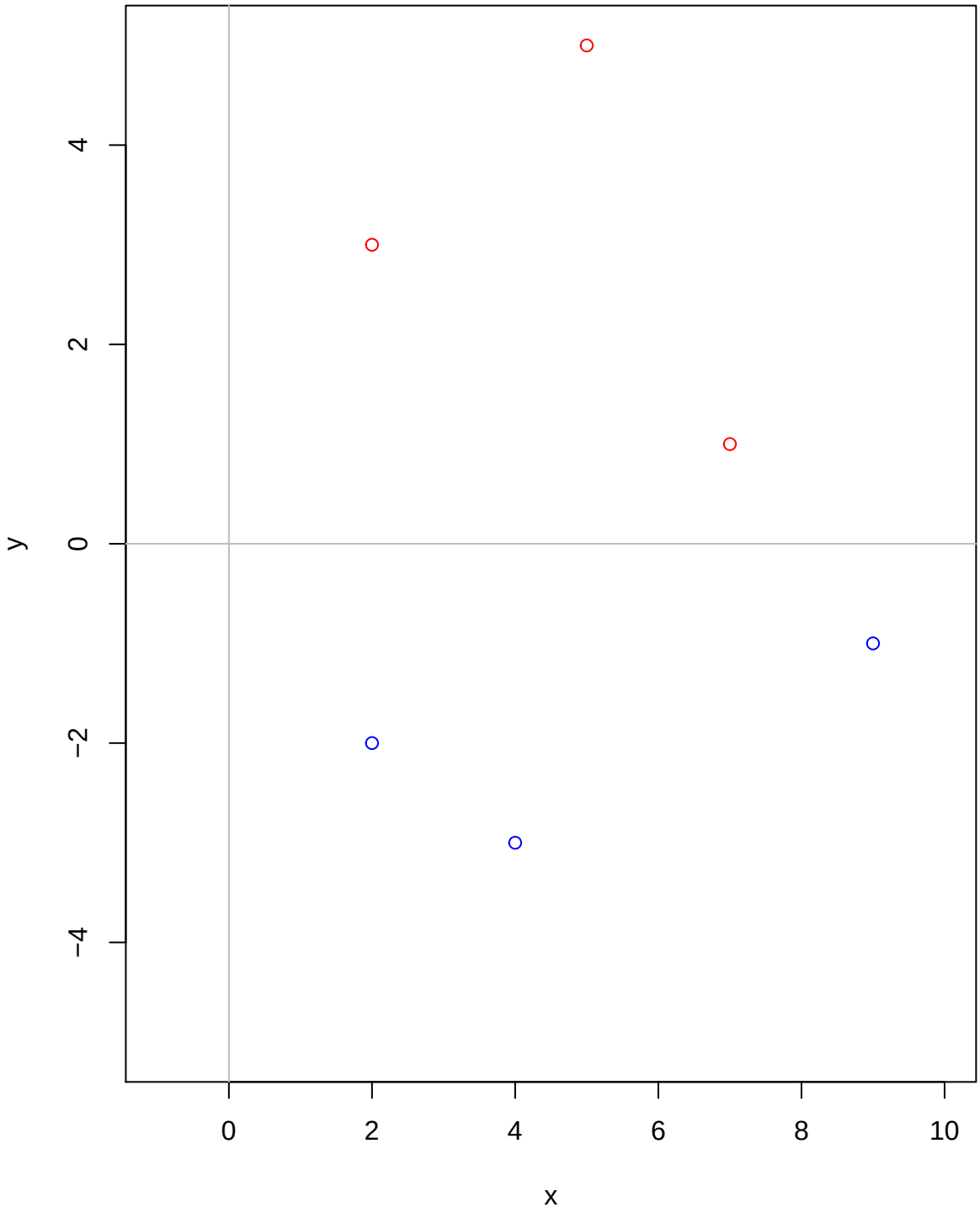

Figure 2: Representación del conjunto de puntos

$$S = \{(2;3),(5;5),(7;1),(-2;-2),(4;-3),(9;-1)\}$$

Cuya representación en el plano cartesiano es la siguiente:

Cuando se utiliza el paquete *neuralnet* se obtiene una red con el siguiente código que se muestra a continuación:

```
library(neuralnet)

#Puntos del plano considerados
input1 = c(2,5,7,2,4,9)
input2 = c(3,5,1,-2,-3,-1)
output = c(1,1,1,0,0,0)
datos1 = cbind(input1,input2,output)

#Se entrena la red fijando una semilla aleatoria de pesos iniciales
#para su reproducción posterior
set.seed(1)
modelo1 <- neuralnet(output ~ input1 + input2, data = datos1)

#Se muestran los puntos de origen y previsiones que se reproducen
#en el excel adjunto
prediction(modelo1)$rep[,1]
```

```
## Data Error:  0;

## 1 2 3 4 5 6
## 4 2 9 7 2 5
```

```
prediction(modelo1)$rep[,2]
```

```
## Data Error:  0;

##  1  2  3  4  5  6
## -3 -2 -1  1  3  5
```

```
prediction(modelo1)$rep[,3]
```

```
## Data Error:  0;

##              1              2              3              4              5
## -0.01570899083 -0.01223092263  0.02291357340  0.97584151044  1.01779572794
##              6
##  1.01784919429
```

En este ejemplo se han dejado los resultados originales de R (sin un mejor tratamiento que se podría realizar), para poderlos explicar de modo sencillo:

- La opción asociada a *rep[,1]*, tiene que ver con las componentes x del conjunto de datos original, hay que fijarse que están indexados por los números de la primera fila y desordenados con respecto a la entrada inicial (el orden va de menor a mayor según *rep[,3]*)

- La opción asociada a *rep[,2]*, tiene que ver con las componentes y del conjunto de datos original, hay que fijarse que están indexados por los números de la primera fila y desordenados con respecto a la entrada inicial (el orden va de menor a mayor según *rep[,3]*)
- La opción asociada a *rep[,3]*, tiene que ver con las predicciones que realizaría la red sobre el mismo conjunto de entrenamiento, es decir serían los datos ajustados (el fitting). Como se observa en este caso, los valores son muy próximos, caso por caso, a los reales ya que estamos ante un caso de datos separables

En definitiva, si se dibuja la arquitectura de red y los parámetros que la componen estaríamos ante el siguiente caso:

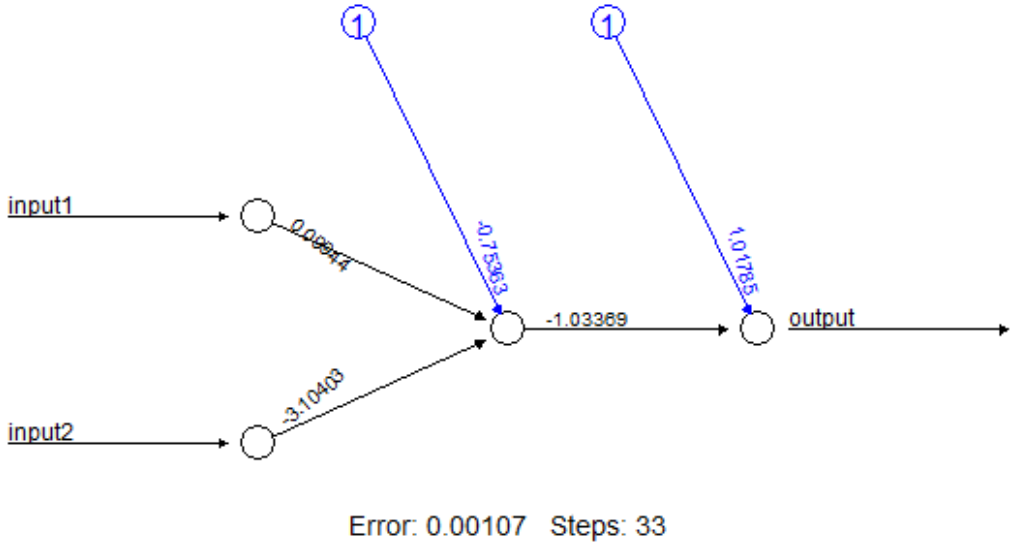

Figure 3: Arquitectura de Red Neuronal

La cual es fácilmente reproducible en una hoja de cálculo Excel tal y como se observa en la siguiente figura:

	A	B	C	D	E	F	G	H	I	J	K	L
1	INPUT1	INPUT2	OUTPUT	RED		PAR_INP11	0,09944	PAR_INP12	-1,03369		OUTPUT1	OUTPUT2
2	4	-3	0	-0,01571		PAR_INP21	-3,10403	BIAS2	1,01785		0,999871	-0,01571
3	3	-2	0	-0,01223		BIAS1	-0,75363				0,996836	-0,01257
4	9	-1	0	0,022914							0,962506	0,022917
5	7	1	1	0,975842							0,040637	0,975844
6	2	3	1	1,017796							5,19E-05	1,017796
7	5	5	1	1,017849							1,41E-07	1,01785

Figure 4: Simulación del modelo en Excel

En la simulación de este modelo se observa lo siguiente:

- Hay 2 variables inputs, las cuáles son metidas en el primer módulo neuronal mediante la función:

$$g(x_1; x_2) = \frac{1}{1 + e^{-[p_{11}x_1 + p_{21}x_2 + b_1]}}$$

- Posteriormente, se obtiene la salida final mediante la función lineal siguiente:

$$o(x_1; x_2) = p_{12} g(x_1; x_2) + b_2$$

Nótese que en este proceso es necesario estimar al menos unos 5. Para finalizar esta sesión se dejan las siguientes cuestiones a discutir:

¿Cuántos parámetros harían falta por tanto en el caso de una regresión lineal? ¿Qué modelo se interpreta mejor? ¿Cuál modelo clasifica mejor las nuevas entradas? ¿Cómo influye la entrada de nueva información en la capacidad de aprendizaje de estos modelos?

BIBLIOGRAFÍA

Fine, L. T. 1999 *Feedforward Neural Network Methodology* Springer-Verlag New York Berlin Heidelberg ISBN 0-387-98745-2