UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

Teorema de Aproximación Universal: prueba constructiva basada en conjuntos semisimpliciales

Autor/es

MARÍA VILLOTA MIRANDA

Director/es

JÓNATAN HERAS VICENTE

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Matemáticas

Departamento

MATEMÁTICAS Y COMPUTACIÓN

Curso académico

2019-20

publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Matemáticas

Teorema de Aproximación Universal: prueba constructiva basada en conjuntos semisimpliciales

Realizado por:

María Villota Miranda

Tutelado por:

Jónathan Heras Vicente

23 de junio de 2020

Resumen

Las Redes Neuronales artifciales son aproximadores universales. El Teorema de Aproximación Universal prueba que dada una función definida en un conjunto compacto en un espacio n-dimensional existe una red neuronal que aproxima dicha función. Sin embaro, este resultado sólo prueba la existencia y no proporciona una método para construir esa red neuronal. En este trabajo veremos una prueba constructiva en la que los conjuntos semisimpliciales jugarán un papel fundamental.

Abstract

Artificial Neural Networks are universal approximators. The Universal Approximation Theorem proves that given a continuous function on a compact set on a n-dimensional space, then there exists a neural network which approximates such function. Such a result proves the existence, but it does not provide a method for finding it. We will see a constructive approach to the proof wherein simplicial complexes will have an instrumental role.

Índice general

Capítulo 1

Introducción

La existencia de funciones complicadas y difíciles de manejar hace que, en muchas ocasiones, sea más eficiente y cómodo trabajar con aproximaciones. Para ello, se buscan funciones sencillas, como los polinomios, capaces de modelar esas funciones difíciles. A lo largo de los cuatro años del Grado de Matemáticas, hemos visto distintas técnicas de aproximación entre las que se encuentran las series de Taylor, los polinomios interpoladores, los polinomios de Chebyshev, los splines, las series de Fourier [1, 2]. El objetivo de este Trabajo de Fin de Grado es construir una red neuronal artificial capaz de aproximar una función definida entre espacios triangulables. La base de esta construcción es puramente matemática y se necesitan conceptos de topología para poder comprender el trabajo. En especial, los conjuntos semisimpliciales. A fin de lograr dicho objetivo, nos apoyaremos en el artículo: *Two hidden-layer feedforward networks are universal approximators: A constructive approach* [3] en el que se muestra cómo computar dicha red neuronal.

Las redes neuronales artificiales, más comúnmente conocidas como redes neuronales, representan una tecnología arraigada en muchas disciplinas: neurociencia, matemáticas, estadística, física, ciencias de la computación e ingeniería. Las redes neuronales tienen aplicaciones en diversos campos como por ejemplo el reconocimiento de patrones, procesamiento de señales, modelado y control, análisis de series temporales [4]. Esto se debe en gran medida a la importante propiedad que poseen: la habilidad de aprender a partir de unos datos de entrada.

Una red neuronal no deja de ser un algoritmo diseñado para simular la forma en la que el cerebro realiza una tarea o una función. Para ello, las redes neuronales utilizan interconexiones masivas de unidades simples de computación conocidas como neuronas. Pero, ¿que es lo que hace que las redes neuronales hayan tenido tanto éxito estos últimos años? Los beneficios de utilizar redes neuronales son los siguientes:

- Son estructuras masivamente paralelizables.

- Su habilidad para aprender y generalizar, es decir, de predecir.

Las redes neuronales no dejan de ser un caso particular del *aprendizaje automático*. El aprendizaje automático es una rama de la *Inteligencia Artificial* cuyo objetivo es desarrollar técnicas que permitan a los ordenadores aprender sin ser programados de manera explícita [5]. Todos los algoritmos del aprendizaje automático tienen una base matemática sólida y aunque muchas veces se vean como cajas negras, la fiabilidad de todos ellos se puede probar gracias a las matemáticas.

El Capítulo 2 está dedicado exclusivamente al estudio de las redes neuronales. En él veremos la evolución desde la *neurona o perceptrón* a las redes neuronales. En el Capítulo 3 se introducirán las nociones y resultados básicos y necesarios sobre los conjuntos semisimpliciales, como la *subdivisión baricéntrica* y el *Teorema de Aproximación Simplicial*, que nos ayudarán a entender y posteriormente a construir la aproximación de la función a través de una red neuronal. Esta construcción la veremos en el Capítulo 4, en el que también estudiaremos el teorema principal de redes neuronales: el *Teorema de Aproximación Universal*. Por último, veremos las conclusiones y, a continuación, los Apéndices A y B en los que encontraremos las definiciones y resultados complementarios que se necesitarán para comprender las demostraciones que se proporcionan en el trabajo y el código que hemos programado para mostrar algún ejemplo del trabajo respectivamente.

Capítulo 2

Redes Neuronales Artificiales

En este capítulo presentamos las nociones básicas sobre redes neuronales artificiales que necesitaremos para entender el resto de trabajo. En concreto, presentamos la forma más simple de una red neuronal artificial, el perceptrón, y sus limitaciones. Veremos además cómo se combinan los perceptrones para formar redes neuronales artificiales más complejas, y su mecanismo de aprendizaje. Una descripción más detallada sobre este tema puede verse en [4, 5].

Las redes neuronales artificiales son modelos matemáticos inspirados en el funcionamiento de las redes neuronales biológicas [4]. En concreto, se parecen en dos aspectos:

- El conocimiento es adquirido del entorno por la red mediante un proceso de aprendizaje.
- La fuerza de conexión interneuronal, conocida como pesos, se usa para almacenar el conocimiento adquirido.

Esto fue lo que motivó en 1943 al neurofisiólogo McCulloch y al matemático Pitts a desarrollar un modelo de redes neuronales [4]. En 1958, el psicólogo Frank Rosenblantt, basándose en el modelo de McCulloch y Pitts desarrolló un modelo simple de neurona, denominado *perceptrón* y una regla de aprendizaje que consiste en la corrección del error [6]. Sin embargo, este modelo presentaba una serie de limitaciones que presentamos a continuación.

2.1. El perceptrón simple

Antes de dar la definición de *perceptrón* conviene introducir la notación que utilizaremos en el trabajo con el fin de facilitar la lectura del mismo.

Dados $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, W una matriz en $\mathbb{R}^{n \times m}$ y $\phi : \mathbb{R} \to \mathbb{R}$; denotaremos por $\phi_{W,b}$ a la función $\phi^m(xW + b)$ dada por la siguiente fórmula:

$$\begin{array}{rccc} \phi^m: & \mathbb{R}^m & \to & \mathbb{R}^m \\ & (x_1,\ldots,x_m) & \to & (\phi(x_1),\ldots,\phi(x_m)) \end{array}$$

A esta función $\phi_{W,b}$ se le denomina *capa*, y existen tres tipos: la capa de entrada, la capa de salida y la capa oculta. La capa de entrada no es una capa como tal ya que no es una función, sino que se corresponde con el vector x. Las capas ocultas son todas las capas intermedias entre la capa de entrada y la de salida. A continuación veremos que el perceptrón sólo tiene dos capas, una de entrada y otra de salida. Más adelante, veremos que las redes neuronales artificiales están formadas por una capa de entrada, una de salida y una o varias capas ocultas.

Definición 2.1 (Perceptrón simple o neurona)**.** *Dados $\phi : \mathbb{R} \to \mathbb{R}$, $W \in \mathbb{R}^{n\times 1}$ y $b \in \mathbb{R}$ el perceptrón simple, o neurona, es una función $f : \mathbb{R}^n \to \mathbb{R}$ definida por $f(x) = \phi_{W,b}(x) = y$*

Una representación gráfica del perceptrón simple puede verse en la figura 2.1. Según la notación que hemos establecido antes, el perceptrón no deja de ser una capa con $m = 1$. Recordar que la capa de entrada es el vector x. En total, el perceptrón está formado por una capa de entrada y otra de salida.

A partir de la definición anterior podemos decir que los elementos básicos de una neurona son los siguientes:

1. La *entrada* se corresponde con el vector $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ en el que cada x_i representa una señal de entrada.

2. $W = (w_1, \ldots, w_n)^T$ agrupa al conjunto de *pesos*. Cada w_i representa el peso asociado a la entrada x_i.

3. El valor b es conocido como el *sesgo* o *umbral de activación*, y es el valor mínimo que tiene que tomar la combinación lineal de la entrada y los pesos para activar la función de activación.

4. La función $\phi : \mathbb{R} \to \mathbb{R}$ se conoce como la *función de activación* y determina la salida de la neurona a partir de las entradas, los pesos y el sesgo: $y = f(x) = \phi_{W,b}(x) = \phi(xW+b)$.

El objetivo del perceptrón es ser capaz de modelar una función $g : \mathbb{R}^n \to \mathbb{R}$ a partir de una nube de puntos, o dataset, dado por $\{(x, g(x))\}$. Para ello en la definición del perceptrón hay dos componentes que no han sido fijadas y que se pueden modificar para tal fin: la función de activación y el valor de los pesos. Mientras que la función de activación es algo que se tiene que fijar al definir una neurona concreta (esto se conoce como *hiperparámetro*), los pesos y el sesgo son aprendidos (y por lo tanto se los llama *parámetros*). A continuación pasamos a describir el proceso iterativo que se sigue para realizar el aprendizaje de los pesos, y las funciones de activación más utilizadas.

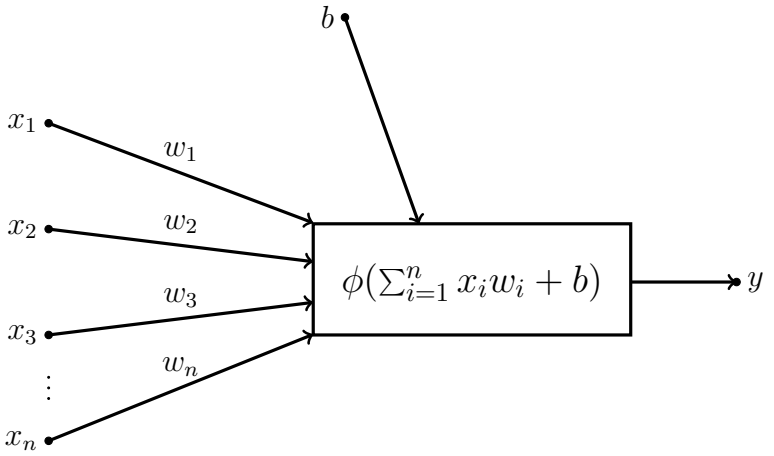


Figura 2.1: Representación de una neurona

2.1.1. Proceso de aprendizaje del perceptrón

La regla de aprendizaje del perceptrón simple nos proporciona los valores de los pesos y del sesgo a partir de un proceso adaptativo e iterativo [7]. Para poder aplicar esta regla hace falta un *conjunto de entrenamiento*.

Definición 2.2 (Conjunto de entrenamiento)**.** *Se denomina conjunto de entrenamiento a una colección de puntos* $\{(x^0, z^0), (x^1, z^1), \ldots, (x^r, z^r)\}$ *donde para cada* $i = 0, \ldots, r$ *se tiene que* $x^i \in \mathbb{R}^n$*. Los* $z^i \in \mathbb{R}$ *se corresponden con el resultado esperado para* x^i*.*

Un ejemplo muy sencillo de conjunto de entrenamiento podría ser los precios de una casa teniendo en cuenta el número de metro cuadrados, el número de años desde su construcción y el número de habitaciones: $\{((70, 15, 2), 90.000), ((150, 2, 3), 168.000), ((200, 30, 3), 383.000)\}$. Una vez que tenemos el conjunto de entrenamiento, lo que esperamos del perceptrón es que lo aprenda y sea capaz de predecir los precios de casas no contempladas en dicho conjunto.

Para ello, el siguiente paso que debemos realizar en este proceso de aprendizaje es establecer el valor de una constante $\eta^k \in \mathbb{R}$ para entrenar al perceptrón. Se trata de un hiperparámetro positivo que recibe el nombre de *tasa de aprendizaje* y se encarga de controlar dicho proceso de aprendizaje. Si el valor de esta constante es fijo en todas las iteraciones, $\eta^k = \eta$, tendremos una regla de aprendizaje con incremento fijo. Cuando la tasa de aprendizaje toma valores grandes, los pesos se modifican rápidamente y cuando toma valores pequeños, los pesos cambian paulatinamente, decimos entonces que el perceptrón aprende poco a poco.

Una vez que tenemos un conjunto de entrenamiento y la tasa de aprendizaje establecidos, se comienza dando valores aleatorios a los pesos y al sesgo del perceptrón, se pasan las distintas r entradas del dataset y se van modificando iterativamente los pesos y el sesgo siempre que la salida y no coincida con la salida deseada z^i . El valor de los pesos en la iteración $k + 1$, denotados por w^{k+1}, viene dado por:

$$\begin{cases} w_j^{k+1} = w_j^k + \Delta w_j^k \\ \Delta w_j^k = \eta^k (z^{k \ mod \ r} - \phi_{w^k,b^k}(x^{k \ mod \ r}))x_j^{k \ mod \ r} \end{cases}$$

Simplificamos la notación, tomamos $i = k \ mod \ r$ y denotamos por y_k^i a $\phi_{W^k,b^k}(x^i)$. Por lo tanto la regla anterior nos queda de la siguiente manera.

$$w_j^{k+1} = w_j^k + \eta^k (z^i - y_k^i)x_j^i \tag{2.1}$$

El sesgo se modifica de la misma manera. A este lo podemos considerar como el peso correspondiente a una nueva señal de entrada igual a $x_{n+1} = -1$ y como valor del peso, el propio valor del sesgo $w_{n+1} = b$.

La regla de aprendizaje es un método de detección de error y corrección. Sólo aprende cuando se equivoca. Veamos cuales son los pasos a seguir para aplicarla.

Algoritmo de aprendizaje

Paso 0: Inicialización

Inicializamos los pesos con valores aleatorios. Ir al paso 1 con $k = 1$.

Paso 1: k-ésima iteración

Sea $i = k \ mod \ r$, tomamos el punto del conjunto de entrenamiento (x^i, z^i) y calculamos y_k^i.

Paso 2: Corrección de los pesos

Si $z^i \neq y_k^i$ modificar los pesos según la expresión:

$$w_j^{k+1} = w_j^k + \eta^k (z^i - y_k^i)x_j^i$$

Paso 3: Parada

Si los pesos no se han modificado en p iteraciones, con $p \in \mathbb{N}$, es decir,

$$w_j^i = w_j^k \text{ con } i = k - p, \ldots, k - 1$$

se finaliza el proceso. También podemos llegar a una parada tras un número marcado de iteraciones.

En caso contrario, se incrementa k en una unidad y se vuelve al paso 1.

En el Apéndice B hemos programado el perceptrón y su algoritmo de aprendizaje además de los ejemplos que siguen.

Antes de ver un ejemplo de este proceso de aprendizaje es necesario explicar la otra componente del perceptrón: las funciones de activación.

2.1.2. Tipos de función de activación

La función de activación es una componente muy importante del perceptrón, y en general de las redes neuronales, ya que sin ella, el perceptrón solo sería capaz de modelar relaciones lineales. Podemos encontrar una descripción detallada de múltiples funciones de activación en [4, 8]. En la tabla 2.1 presentaremos las funciones de activación más utilizadas.

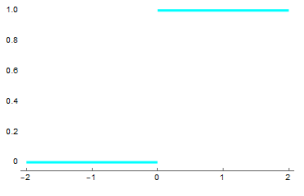
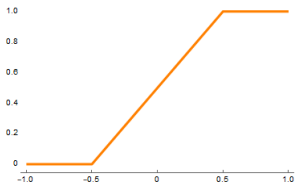
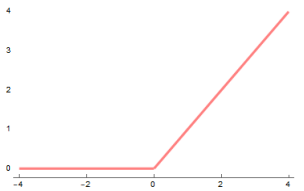
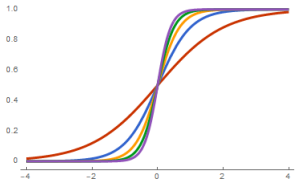
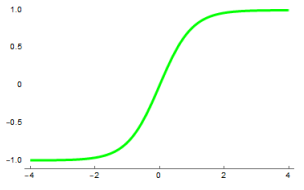
Nombre de la función	Función	Gráfica
Función de paso	$f(v) = \begin{cases} 1 & si \quad v \geq 0 \\ 0 & si \quad v < 0 \end{cases}$	
Función lineal a trozos	$f(v) = \begin{cases} 1 & si \quad v \geq 1/2 \\ v + 1/2 & si \quad -1/2 < v < 1/2 \\ 0 & si \quad v \leq -1/2 \end{cases}$	
RELU	$f(v) = max(0, v)$	
Sigmoide	$f_a(v) = \frac{1}{1+e^{-av}}$	
Tangente hiperbólica	$f(v) = tanh(v) = \frac{2}{1+e^{(-2v)}} - 1$	

Tabla 2.1: Funciones de activación

Una vez explicadas la regla de aprendizaje y las distintas funciones de activación veremos un ejemplo en el que el perceptrón es incapaz de aprender. Hablamos entonces de las limitaciones del perceptrón.

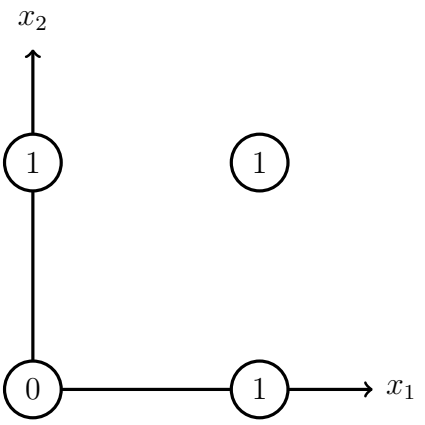


Figura 2.2: Función OR

2.2. Limitaciones del perceptrón simple

Comenzamos mostrando un ejemplo de función que el perceptrón sí que es capaz de modelar.

Ejemplo 2.1 (Función lógica OR)**.** *La función lógica OR es una función* $OR : \{0,1\} \times \{0,1\} \to \{0,1\}$ *que se define según la tabla 2.2*

x_1	x_2	$OR(x_1, x_2) = z$
0	0	0
1	0	1
0	1	1
1	1	1

Tabla 2.2: Función OR

Una representación gráfica de esta función viene dada en la figura 2.2 donde el valor esperado para cada para $(x1, x2)$ viene dado dentro del punto.

Nuestro conjunto de entrenamiento son los puntos de la función OR $\{((0,0),0)$, $((1,0),1)$, $((0,1),1)$, $((1,1),1)\}$. Para este ejemplo tomaremos como función de activación la función de paso. Ahora, ya solo nos falta entrenar al perceptrón simple. Para ello, seguimos los pasos que vimos en la subsección 2.1.1. Establecemos la tasa de aprendizaje, $\eta = 0.3$, y tomamos valores aleatorios para w_1, w_2 y $b = w_3$:

$$w_1^1 = 1.5,\ w_2^1 = 1.5,\ w_3^1 = 2$$

Con estos datos establecemos el separador $1.5x_1 + 1.5x_2 - 2 = 0$. Los puntos que se queden por debajo tomarán el valor de 0 y los puntos que se queden por encima o que estén en el plano tomarán el valor de 1. Esto lo podemos ver en la figura 2.3a. En ella observamos que hay casos que están bien clasificados, el (1,1) y el (0,0), y otros que no, como el (1,0) y el (0,1).

Proseguimos con el algoritmo de aprendizaje. El siguiente paso es calcular la salida en cada uno de los 4 casos.

- Caso 1: $x^0 = (0,0), z^0 = 0$

$$v = 1.5 \cdot 0 + 1.5 \cdot 0 - 2 = -2 < 0 \Rightarrow y_1^0 = 0$$

Como $z^0 = y_1^0 = 0$ el punto (0,0) está bien clasificado

- Caso 2: $x^1 = (1,0), z^1 = 1$

$$v = 1.5 \cdot 1 + 1.5 \cdot 0 - 2 = -0.5 < 0 \Rightarrow y_1^1 = 0$$

El punto (1,0) está mal clasificado ya que $z = 1 \neq y_1^1 = 0$. Aplicamos entonces la regla del perceptrón 2.1:

$$w_1^2 = w_1^1 + 0.3(z^1 - y_1^1)x_1 = 1.5 + 0.3 \cdot 1 \cdot 1 = 1.8$$

$$w_2^2 = w_2^1 + 0.3(z^1 - y_1^1)x_2 = 1.5 + 0.3 \cdot 1 \cdot 0 = 1.5$$

$$w_3^2 = w_3^1 + 0.3(z^1 - y_1^1)x_3 = 2 + 0.3 \cdot 1 \cdot (-1) = 1.7$$

Los nuevos valores de los pesos y el sesgo, $w_1^2 = 1.8, w_2^2 = 1.5, b^2 = 1.7 = w_3^2$, determinan un nuevo separador $1.8x_1 + 1.5x_2 - 1.7 = 0$. Como podemos observar en la figura 2.3b, al actualizar los pesos y el sesgo, el perceptrón ha aprendido y ya clasifica correctamente el punto (1,0). Sin embargo, para el punto (0,1) sigue haciendo una clasificación errónea, el perceptrón tiene que seguir aprendiendo.

- Caso 3: $x^2 = (0,1), z^2 = 1$

$$v = 1.8 \cdot 0 + 1.5 \cdot 1 - 1.7 = -0.2 < 0 \Rightarrow y_2^2 = 0$$

Actualizamos los pesos y el sesgo según la regla de aprendizaje.

$$w_1^3 = w_1^2 + 0.3(z^2 - y_2^2)x_1 = 1.8 + 0.3 \cdot 1 \cdot 0 = 1.8$$

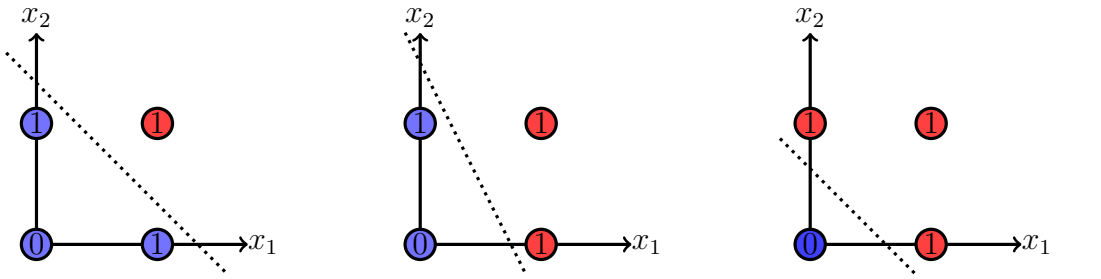
$$w_2^3 = w_2^2 + 0.3(z^2 - y_2^2)x_2 = 1.5 + 0.3 \cdot 1 \cdot 1 = 1.8$$

$$w_3^3 = w_3^2 + 0.3(z^2 - y_2^2)x_3 = 1.7 + 0.3 \cdot 1 \cdot (-1) = 1.4$$

Los nuevos valores de los pesos y el sesgo, $w_1^3 = 1.8, w_2^3 = 1.8, b^3 = 1.4 = w_3^3$, determinan un nuevo separador $1.8x_1 + 1.8x_2 - 1.4 = 0$

- Caso 4: $x^3 = (0,0), z^3 = 0$

$$v = 1.8 \cdot 1 + 1.8 \cdot 1 - 1.4 = 2.2 < 0 \Rightarrow y_3^3 = 1$$



(a) $1.5x_1 + 1.5x_2 - 2 = 0$ (b) $1.8x_1 + 1.5x_2 - 1.7 = 0$ (c) $1.8x_1 + 1.8x_2 - 1.4 = 0$

Figura 2.3: Aprendizaje del perceptrón función OR

Como $z = 1 = y^3$ el punto (1,1) está bien clasificado, entonces los pesos y el sesgo no se modifican.

Podemos ver en la figura 2.3c que el perceptrón ya clasifica correctamente todos los casos. Por lo tanto, en las siguientes iteraciones de la regla, al ser $z = y$, los pesos y el sesgo no se actualizarán dando lugar al fin del algoritmo de aprendizaje.

Este ejemplo lo podemos ver también en el Notebook que encontramos en el Apéndice B.

Acabamos de ver que el perceptrón es capaz de simular la función lógica OR. Esto nos hace preguntarnos si el perceptrón es capaz de simular otras funciones lógicas como las funciones AND o XOR. En el siguiente ejemplo veremos qué sucede cuando el perceptrón intenta modelar la función XOR.

Ejemplo 2.2 (Función lógica XOR)**.** *La función lógica XOR es una función* $XOR : \{0,1\} \times \{0,1\} \to \{0,1\}$ *que se define según la tabla 2.3.*

x_1	x_2	$XOR(x_1, x_2) = z$
0	0	0
1	0	1
0	1	1
1	1	0

Tabla 2.3: Función XOR

Al igual que en el ejemplo anterior, esta función se puede representar como se muestra en la figura 2.4.

El conjunto de entrenamiento viene determinado por los valores de la tabla 2.3. Procedemos de la misma manera que en el ejemplo anterior. Tomamos unos pesos y un sesgo aleatorio:

$$w_1^1 = 1, w_2^1 = 1, w_3^1 = 0.5$$

Hace falta establecer también el valor del ratio de aprendizaje: $\eta = 0.25$. Con los datos establecemos el separador $x_1 + x_2 - 0.5 = 0$. Los puntos que se queden por debajo de dicho

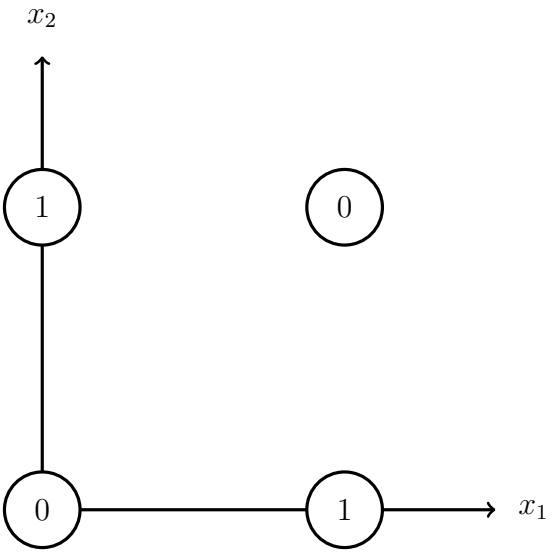
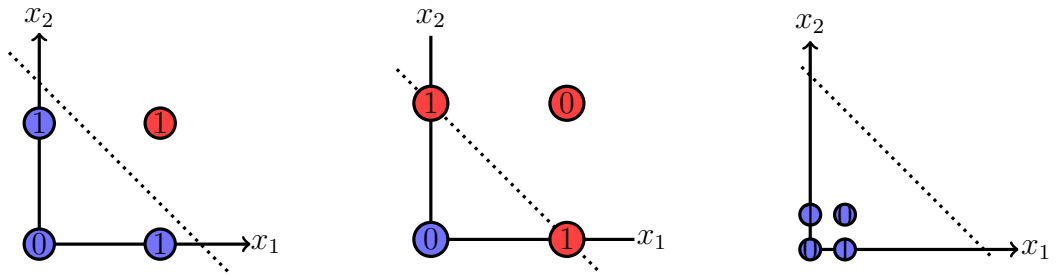


Figura 2.4: Función XOR



(a) $x_1 + x_2 - 0.5 = 0$ (b) $0.75x_1 + 0.75x_2 - 0.75 = 0$ (c) $0.25x_1 + 0.25x_2 - 1.25 = 0$

Figura 2.5: Aprendizaje del perceptrón función XOR

separador, el perceptrón devolverá el valor 0 y los puntos estén en el plano o se queden encima de él, devolverá el valor 1. Esto lo podemos ver en la figura 2.5a. En ella observamos que hay casos que están bien clasificados, el (1,0), el (0,1) y el (0,0), y otros que no como el (1,1).

Continuamos con el algoritmo de aprendizaje. El siguiente paso es calcular la salida en cada uno de los 4 casos.

- Caso 1: $x^0 = (0, 0), z^0 = 0$

  $$v = 0 + 0 - 0.5 = -0.5 < 0 \Rightarrow y_1^0 = 0$$

  Como $z^0 = y_1^0 = 0$ el punto (0,0) está bien clasificado.

- Caso 2: $x^1 = (1, 0), z^1 = 1$

  $$v = 1 + 0 - 0.5 = 0.5 > 0 \Rightarrow y_1^1 = 1$$

  El punto (1,0) está bien clasificado ya que $z^1 = 1 = y_1^1$.

	w_1	w_2	b	η	z	y	x_1	x_2
1	1	1	0.5	0.25	0	1	1	1
2	0.75	0.75	0.75	0.25	0	1	1	1
3	0.5	0.5	1.	0.25	0	1	1	1
4	0.25	0.25	1.25	0.25	0	0	1	1

Tabla 2.4: Variación de los pesos y el sesgo

- Caso 3: $x^2 = (0, 1), z^2 = 1$

$$v = 0 + 1 - 0.5 = 0.5 > 0 \Rightarrow y_1^2 = 1$$

El punto (0,1) está bien clasificado ya que $z^2 = 1 = y_1^2$.

- Caso 4: $x^3 = (1, 1), z^3 = 0$

$$v = 1 + 1 - 0.5 = 1.5 > 0 \Rightarrow y_1^3 = 1$$

El punto (1,1) está mal clasificado ya que la salida esperada es $z^3 = 0$. Por lo tanto hace falta actualizar los pesos y el sesgo según la regla de aprendizaje.

$$w_1^2 = w_1^1 + 0.25(z^3 - y_1^3)x_1 = 1 + 0.25 \cdot (-1) \cdot 1 = 0.75$$

$$w_2^2 = w_2^1 + 0.25(z^3 - y_1^3)x_2 = 1 + 0.25 \cdot (-1) \cdot 1 = 0.75$$

$$w_3^2 = w_3^1 + 0.25(z^3 - y_1^3)x_3 = 0.5 + 0.25 \cdot (-1) \cdot (-1) = 0.75$$

Los nuevos valores de los pesos y el sesgo, $w_1^2 = 0.75, w_2^2 = 0.75, w_3^2 = 0.75$, determinan un nuevo separador $0.75x_1 + 0.75x_2 - 0.75 = 0$

En la figura 2.5b, observamos que el perceptrón sigue clasificando bien los puntos $(0, 0)$, $(1, 0)$, $(0, 1)$, sin embargo, sigue sin clasificar correctamente el punto $(1, 1)$.

Para ver cómo se van modificando los pesos y el sesgo en las siguientes iteraciones hemos programado una función en Mathematica. El código lo encontramos en el Apéndice B. Esta función nos devuelve la tabla que vemos en la figura 2.4. En ella observamos que en la cuarta iteración el perceptrón ya clasifica bien el punto $(1, 1)$. Obtenemos los siguientes valores para los pesos y el sesgo:

$$w_1^4 = 0.25, \ w_2^4 = 0.25, \ w_3^4 = 1.25$$

Se corresponde con el separador $0.25x_1 + 0.25x_2 - 1.25 = 0$. Este separador aparece representado gráficamente en la figura 2.5c. El perceptrón ya clasifica bien el punto $(1, 1)$ pero lo hace sacrificando la correcta clasificación de los puntos $(0, 1)$ y $(1, 0)$.

En el Apéndice B podemos ver como el perceptrón no consigue modelar la función XOR. Esto se debe a que este proceso, para este caso, no converge independientemente

de los valores que se tomen, por lo tanto, el perceptrón es incapaz de simular la función lógica XOR. ¿Qué es lo que hace que el perceptrón sea capaz de simular la función OR pero no la XOR? Para responder esta pregunta hace falta introducir el concepto de *separabilidad lineal*.

2.2.1. Separabilidad lineal

Introducimos los conceptos necesarios para entender la separabilidad lineal [9, 10].

Definición 2.3 (Hiperplano)**.** *Sea E un espacio euclídeo de dimensión n, se denomina hiperplano de E a un subespacio afín de dimensión n-1.*

Definición 2.4 (Separabilidad lineal)**.** *Sean A y B dos conjuntos de puntos en un espacio euclídeo E de dimensión n, se dice que A y B son linealmente separables si existen n+1 números reales $w_1, \ldots, w_n, k$ tales que:*

- $\forall a \in A, \sum_{i=1}^{n} w_i a_i \geq k$*, con a_i la i-ésima componente de a.*
- $\forall b \in B, \sum_{i=1}^{n} w_i b_i < k$*, con b_i la i-ésima componente de b.*

Geométricamente, A y B son linealmente separables si existe un hiperplano E que separa los puntos de A y los de B. Según la definición dada, el hiperplano podría contener puntos de A y vendría dado por la siguiente ecuación:

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n = b$$

Si volvemos a los ejemplos que hemos visto en la sección anterior podemos observar que claramente al graficar la función OR, figura 2.2, los puntos son linealmente separables. En cambio, los puntos de la función XOR, figura 2.4, no lo son. Veamos que en efecto, no lo son.

Si el dataset XOR es linealmente separable, entonces existen w_1, w_2 y b tales que para todo $((x1, x2), y)$ del dataset XOR se verifica que $w_1 x_1 + w_2 x_2 < b$ si $y = 0$, y $w_1 x_1 + w_2 x_2 \geq b$ si $y = 1$. Por lo tanto:

$$w_1 \cdot 0 + w_2 \cdot 0 < b$$
$$w_1 \cdot 1 + w_2 \cdot 0 \geq b$$
$$w_1 \cdot 0 + w_2 \cdot 1 \geq b$$
$$w_1 \cdot 1 + w_2 \cdot 1 < b$$

De estas inecuaciones obtenemos que $b > 0$, $w_1 \geq b$ y $w_2 \geq b$ y con la última inecuación obtenemos una contradicción: $w_1 + w_2 < b$. Entonces, la función XOR no el linealmente separable.

En el siguiente teorema veremos como el perceptrón solo es capaz de clasificar correctamente aquellos puntos que son linealmente separables.

Teorema 2.1 (Convergencia del perceptrón para clasificación binaria)**.** *Sea el conjunto de entrenamiento dado por* $\{(x^1, z^1), (x^2, z^2), \ldots, (x^k, z^k)\}$ *donde* $x^i \in \mathbb{R}^n$ *y* $z^i \in \{0, 1\}$ *es la salida esperada de* x^i, $\forall i \in \{1, \ldots, k\}$. *El perceptrón simple con la función de paso como función de activación clasifica correctamente todos los puntos* x^i *si y solo si dicho conjunto es linealmente separable.*

Demostración. Veamos la implicación hacia la derecha, suponemos que existe un perceptrón simple de n entradas que clasifica todos los puntos correctamente, es decir, que para todo x^i, $\phi_{W,b}(x^i) = z^i$, por lo tanto existen distintos números reales $w_1, \ldots, w_n$ como pesos y $b > 0$ como el sesgo tales que:

$$z^j = \begin{cases} 1 & si \quad \sum_{i=1}^n w_i x_i^j + b \geq 0 \\ \\ 0 & si \quad \sum_{i=1}^n w_i x_i^j + b < 0 \end{cases}$$

Restando b a ambos lados, nos queda lo siguiente:

$$z^j = \begin{cases} 1 & si \quad \sum_{i=1}^n w_i x_i^j \geq -b \\ \\ 0 & si \quad \sum_{i=1}^n w_i x_i^j < -b \end{cases}$$

Los valores de los pesos y el sesgo corresponden con los números reales de la definición de linealmente separable. Por lo tanto podemos definir el conjunto A formado por todos los x^i cuyo valor $z^i = 1$ y el conjunto B formado por los x^i que hacen que $z^i = 0$. Por lo tanto, los conjuntos A y B son linealmente separables y entonces el conjunto $\{(x^1, z^1), (x^2, z^2), \ldots, (x^k, z^k)\}$ es también linealmente separable.

Veamos la implicación hacia la izquierda, suponemos que los puntos son linealmente separables, entonces existen dos conjuntos A y B en los que podemos agrupar nuestros puntos. Sin pérdida de generalidad podemos escoger $A = \{x^i \mid z^i = 1\}$ y $B = \{x^i \mid z^i = 0\}$ con $1 \leq i \leq k$. Por la definición de linealmente separable tenemos lo siguiente.

$$\begin{cases} \sum_{i=1}^n w_i x_i^j \geq b \quad si \quad x^j \in A \\ \\ \sum_{i=1}^n w_i x_i^j < b \quad si \quad x^j \in B \end{cases}$$

Pero los números reales $w_1, \ldots, w_n, -b$ son exactamente los pesos y el sesgo del perceptrón, por lo tanto ya hemos encontrado el perceptrón que clasifica correctamente todos los puntos. □

Acabamos de ver que el perceptrón sólo permite resolver problemas que son linealmente separables, y los problemas reales raramente lo son. Estas limitaciones supusieron el abandono de muchos científicos en una búsqueda sin éxito de un algoritmo de aprendizaje capaz de

implementar funciones de cualquier tipo. Sin embargo, estas limitaciones hicieron plantearse la necesidad de introducir capas intermedias o capas ocultas entre la capa de entrada y la de salida. Surgieron así, las redes neuronales artificiales. No fue hasta 1986 cuando Rumelhart, Hinton y Williams desarrollaron un algoritmo de aprendizaje, el *algoritmo de propagación hacia atrás*, con el que se puede entrenar redes con múltiples perceptrones conectados calculando los pesos y el sesgo a partir de un conjunto de entradas y salidas esperadas [4].

En las secciones siguientes veremos estos avances que hacen que las redes neuronales artificiales sean estructuras tan interesantes.

2.3. Redes neuronales artificiales

Las redes neuronales artificiales están formadas por perceptrones que están conectados unos a otros haciendo así posible que la información de entrada viaje por toda la red produciendo al final unos valores de salida. A continuación proporcionamos una definición formal.

Definición 2.5 (Red neuronal artificial)**.** *Una red neuronal artificial, o RNA, es una función $N_{W,\theta} : \mathbb{R}^n \to \mathbb{R}^m$ que depende de un conjunto de pesos W y un conjunto de parámetros θ que involucra a la descripción de la función de activación, a las capas, la conexión entre neuronas y cualquier otra consideración en su arquitectura.*

La forma en la que se conectan los perceptrones determinan distintas arquitecturas. La más común y la que necesitamos introducir para este trabajo es la *red neuronal hacia adelante* (*feedforward network*). No obstante, existe una gran variedad de arquitecturas, en este trabajo carece de sentido profundizar en ellas, por lo que nos limitaremos a mostrar una imagen en la que aparezcan otros tipos, ver figura 2.6. En [11] encontramos más información sobre las distintas arquitecturas que existen.

Definición 2.6 (Red neuronal multicapa hacia adelante)**.** *Una red neuronal multicapa hacia adelante definida en un espacio n-dimensional real es una función $N : \mathbb{R}^n \to \mathbb{R}^m$ tal que para cada $x \in \mathbb{R}^n$, $N(x)$ es la composición de las funciones*

$$N(x) = f_{k+1} \circ \cdots \circ f_1(x)$$

donde $k \in \mathbb{Z}$ con $k \geq 1$ es el número de capas ocultas, cada f_i es una capa y son de la forma

$$\begin{array}{rccll} f_i\colon & \mathbb{R}^{d_{i-1}} & \to & \mathbb{R}^{d_i} & 1 \leq i \leq k+1 \\ & y & \to & f_i(y) = \phi_{W^{(i)},b_i}(y) & \end{array}$$

siendo $W^{(i)} = \mathbb{R}^{d_i \times d_{i-1}}$ una matriz real, $b_i \in \mathbb{R}^{d_i}$ el término de sesgo y ϕ la función de activación.

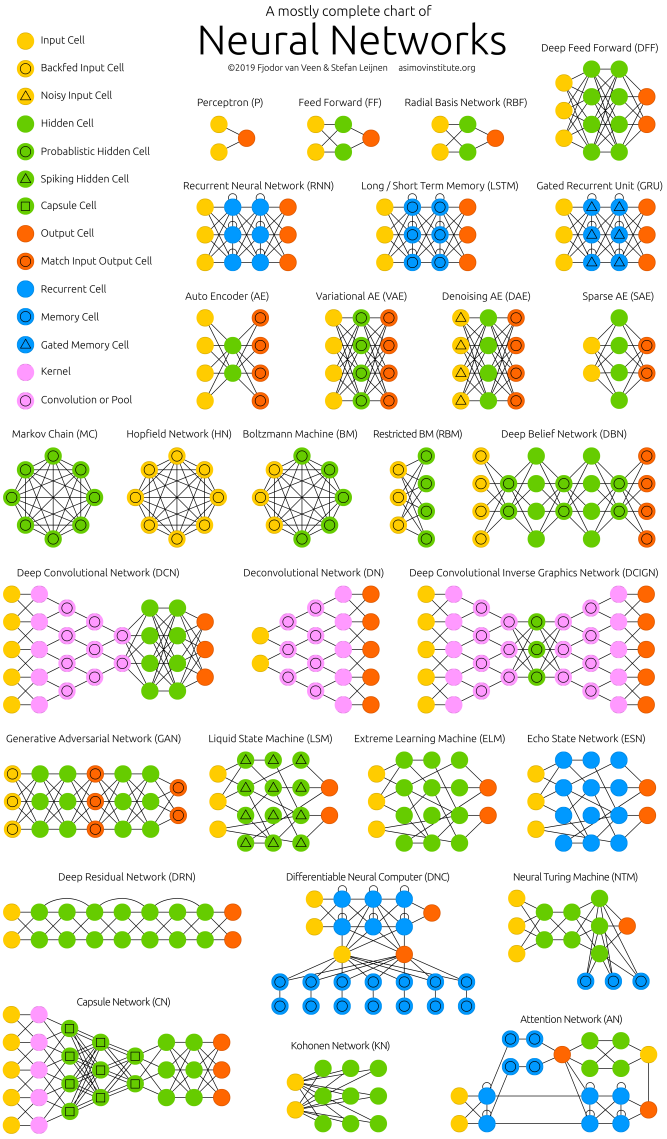


Figura 2.6: Otras arquitecturas

Notar que $d_0 = n$ y $d_{k+1} = m$. Estos valores se corresponden con el tamaño de la capa de entrada y con la de salida respectivamente. Los demás d_i con $0 < i < k + 1$ representan la amplitud de la i-ésima capa capa oculta.

Definición 2.7 (Amplitud de la RNA hacia adelante)**.** *Dada una red neuronal multicapa hacia adelante la amplitud de su i-ésima capa oculta es el valor que toma $d_i \in \mathbb{Z}$ con $1 \leq i < k$. La amplitud de la red neuronal multicapa hacia adelante es el $max\{d_0, \ldots, d_{k+1}\}$.*

Para representar una RNA hacia adelante se utiliza una secuencia de números. Esta secuencia consiste simplemente en las distintas amplitudes de la red desde la capa inicial hasta la final seguidas de un guión. Por ejemplo en la figura 2.7 la RNA se le denominaría 3-2-3-2. Una vez visto todo los conceptos necesarios para entender la RNA hacia adelante veamos un ejemplo.

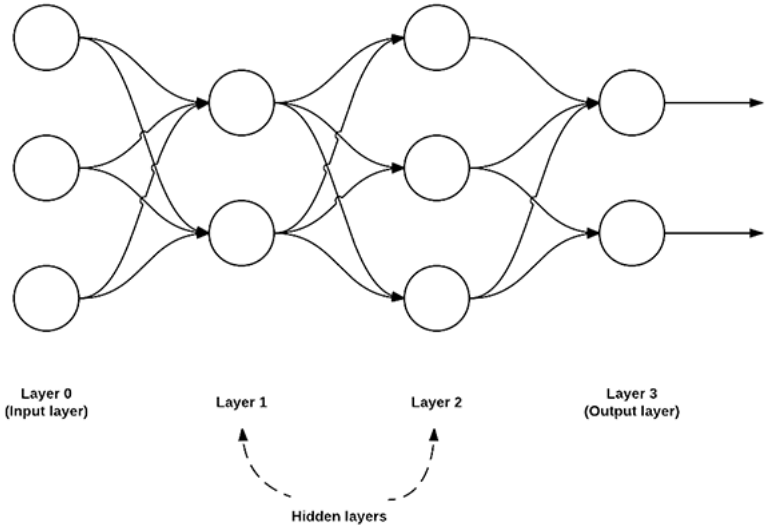


Figura 2.7: Red neuronal multicapa hacia delante o red 3-2-3-2

Ejemplo 2.3. *Sea la red 5-3-2, se trata de una red multicapa hacia adelante con una capa oculta. La amplitud de cada capa es:* $d_0 = 5, d_1 = 3, d_2 = 2$ *y la amplitud de la red es* $max\{5, 3, 2\} = 5$.

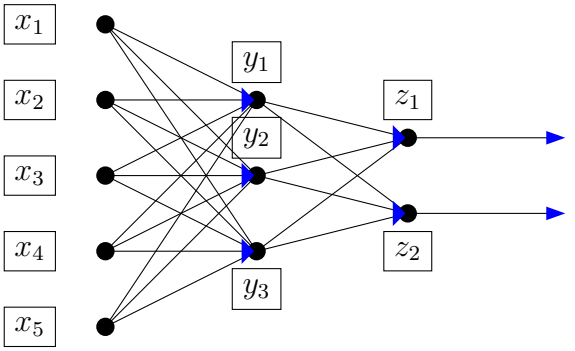


Figura 2.8: Ejemplo

Por la definición 2.6 tenemos que esta red neural es una función $N : \mathbb{R}^5 \to \mathbb{R}^2$ *con* $N = f_2 \circ f_1(x)$, $x \in \mathbb{R}^5$. f_1 *y* f_2 *son la capa oculta y capa de salida, respectivamente.* $f_1 : \mathbb{R}^5 \to \mathbb{R}^3$ *y* $f_2 : \mathbb{R}^3 \to \mathbb{R}^2$. *Para cada función hay asociada una matriz de pesos y un vector de sesgos. En este caso* $W^{(1)}$ *es una matriz* 5×3 *y* $W^{(2)}$, *una* 3×2.

$$W^{(1)} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \\ w_{51} & w_{52} & w_{53} \end{bmatrix} \qquad W^{(2)} = \begin{bmatrix} v_{11} & v_{12} \\ v_{21} & v_{22} \\ v_{31} & v_{32} \end{bmatrix}$$

Cada w_{ij} *es el peso asociado a la entrada* x_i *y a* y_j, *con* $1 \leq i \leq 5$ *y* $1 \leq j \leq 3$.

Cada v_{ij} es el peso asociado a y_i y a z_j, con $1 \leq i \leq 3$ y $1 \leq j \leq 2$

Al igual que vimos el algoritmo de aprendizaje del perceptón, a continuación veremos como aprende una RNA.

2.4. Propagación hacia atrás

Lo que hace que las RNA sean tan interesantes es su capacidad de aprender y formarse a sí mismas. Lo hacen mediante un proceso denominado *propagación hacia atrás* [4]. Este método modifica los pesos de manera que estos sean capaces de almacenar lo mejor posible el conocimiento adquirido. Puede ser visto como un proceso iterativo de optimización en el que se busca minimizar el error. En esta sección vamos a ver brevemente la *regla Delta*, esta regla trata de determinar los pesos de manera que se minimice la función del error cuadrático siguiente:

$$E = \frac{1}{2} \sum_{i=1}^{k} \sum_{j=1}^{m} (z_j^i - y_j^i)^2$$

El proceso que se sigue para encontrar el mínimo de esta función se conoce como *regla del descenso por gradiente.*

2.4.1. Descenso por gradiente

Este método tiene su base en conceptos de análisis matemático, en particular algunos relacionados con el cálculo de derivadas parciales.

Definición 2.8 (Derivada parcial)**.** *Sea $f : A \to \mathbb{R}^n$, con A un abierto de $\mathbb{R}^n$, y $x \in A$. La derivada parcial de f respecto a la j-ésima variable en el punto x se define como*

$$\frac{\partial f(x)}{\partial x_j} = \lim_{h \to 0} \frac{f(x + he_j) - f(x)}{h},$$

si existe este límite. Aquí, e_j representa un vector de $\mathbb{R}^n$ cuyas componentes son todas 0 excepto la j-ésima que toma el valor 1.

Definición 2.9 (Gradiente)**.** *Dados un abierto A de $\mathbb{R}^n$ y una función $f : A \to \mathbb{R}$ derivable, el gradiente o vector gradiente de f se define como el vector cuyas componentes son las derivadas parciales de f:*

$$\nabla f(x) = \left(\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_n} \right) \quad \forall x = (x_1, \ldots, x_n) \in A$$

Geométricamente, si f toma valores en $\mathbb{R}$ podemos interpretarla como una superficie en $\mathbb{R}^{n+1}$, donde las componentes de cada punto tengan la siguiente forma:

$$(x_1, \ldots, x_n, f(x_1, \ldots, x_n)), \text{ donde } (x_1, \ldots, x_n) \in A \subseteq \mathbb{R}^n$$

El gradiente de una función está asociado a la dirección en la que se producen los mayores cambios en la función. En términos de la superficie anterior, el gradiente determina la dirección hacia la cual la *pendiente* del vector tangente a la superficie es mayor. Si nos desplazamos una pequeña distancia sobre la superficie en la dirección opuesta a dicho vector (determinada por $-\nabla f(x)$), nos encontramos en un punto cuyo valor de f es menor. Es decir, hemos *descendido* en la superficie hacia un punto de menor valor.

Esa es la idea básica que se sigue en el método del descenso por gradiente. Comenzando en un punto cualquiera x^k, generalmente aleatorio, se calculan nuevos puntos x^{k+1} de forma que:

$$x^{k+1} = x^k - \eta \nabla f(x^k)$$

donde η es un factor positivo que determina la magnitud de cambio. De forma general podemos definir el método como sigue:

$$\begin{cases} x^1 & aleatorio \\ \\ x^{k+1} = x^k - \eta \nabla f(x^k) & k \geq 1 \end{cases}$$

Podemos suponer lo siguiente:

$$f(x^1) \geq f(x^2) \geq f(x^3) \geq \ldots$$

luego el método del descenso por gradiente converge hacia un mínimo local de la función f. El interés radica en poder determinar cuándo dicho mínimo es global. Si la función es convexa este hecho está garantizado (incluidas en el Apéndice A), por lo tanto, en estos casos, el descenso de gradiente converge a un mínimo global. Sin embargo, esto no tiene por qué ocurrir siempre, si la función tiene un gran número de máximos y mínimos locales, resulta complicado determinar el comportamiento del descenso por gradiente. En concreto, la función error suele tener un gran número de mínimos locales y el algoritmo de aprendizaje puede quedar atrapado en un mínimo local no deseado, ya que en él, el gradiente vale cero.

Para prevenir esta situación, Hinton y Williams (1986) [12] propusieron que se tuviera en cuenta también el gradiente de la iteración anterior y realizar un promedio con el gradiente de la iteración actual. Este promedio produce un efecto de reducción drástica de las fluctuaciones del gradiente en iteraciones consecutivas. Por tanto, se modifica la regla de retropropagación teniendo en cuenta el descenso seguido en el paso anterior y descendiendo por una dirección

intermedia entre la dirección marcada por el gradiente (en sentido opuesto) y la dirección seguida en la iteración anterior, como se expresa en la siguiente ecuación.

$$\Delta w_{ij}^{k+1} = \alpha \Delta w_{ij}^{k} + \eta \frac{\partial E}{\partial w}$$

α es la *constante de momentos* y toma valores entre $0 \leq \alpha < 1$. Es la que controla el grado de modificación de los pesos teniendo en cuenta la modificación en la etapa anterior. Cuando $\alpha = 0$ tenemos la regla Delta.

En nuestro trabajo, no necesitaremos utilizar la regla de aprendizaje para las RNA ya que con el método que proporcionaremos, los pesos y el sesgo se calcularan explícitamente sin tener que realizar el proceso del descenso por gradiente. Esto lo veremos en el capítulo 4, en él también veremos que las *RNA feed forward* son aproximadores universales. Para ver esto será necesario introducir la noción de conjunto semisimplicial además de otros resultados importantes. Todo esto lo vemos en el siguiente capítulo.

Capítulo 3

Conjuntos semisimpliciales

Los *conjuntos semisimpliciales*, también llamados *complejos simpliciales*, son una estructura de datos ampliamente utilizada para representar espacios topológicos. En concreto, estos objetos matemáticos sirven para representar una descomposición de un espacio topológico en piezas simples llamadas *símplices*. En este capítulo proporcionamos las definiciones y resultados necesarios sobre conjuntos semisimpliciales para llegar a demostrar el *Teorema de Aproximación Simplicial* [3], que es un resultado fundamental para nuestro trabajo sobre redes neuronales. Los conceptos que veremos en este capítulo están basados en los vistos en el curso de Geometría y Toplogía de Superficies de la Universidad de La Rioja [13].

3.1. Nociones básicas de los conjuntos semisimpliciales

Empezamos definiendo las piezas más simples de estos conjuntos semisimpliciales.

Definición 3.1 (i-símplice)**.** *Sean* $\{v_0, \ldots, v_i\}$ $i+1$ *puntos independientes del espacio afín* $\mathbb{R}^n$ *con* $i \leq n$*. Un* i*-símplice es una tupla* $\sigma = (v_0, \ldots, v_i)$ *y su realización es un conjunto convexo denotado por* $|\sigma|$ *y dado por*

$$|\sigma| = \{\lambda_0 v_0 + \cdots + \lambda_i v_i \mid \sum_{j=0}^{i} \lambda_j = 1,\ 0 \leq \lambda_j \in \mathbb{R}\}$$

Denotamos por $dim(\sigma)$ *a la dimensión de* σ*.*

Dados dos símplices, τ y σ, se dice que τ es una cara de σ si $\emptyset \neq \tau \subseteq \sigma$. Si τ es distinto de σ se dice que τ es una *cara estricta* de σ y se denota por $\sigma \succ \tau$. A los 0-símplices se les llama vértices; a los 1-símplices, aristas; y a los 2-símplices, triángulos. La frontera de un símplice σ, denotada por $bd(\sigma)$, está formada por las uniones de todas las caras estrictas de dicho símplice y el interior de σ es $int(\sigma) = |\sigma| - |bd(\sigma)|$.

Cuando juntamos distintos símplices, se forma una estructura más compleja llamada conjunto semisimplicial. Veamos en qué consiste.

Definición 3.2 (Conjunto semisimplicial)**.** *Un conjunto semisimplicial es un par* (X, S)*, donde* X *es un conjunto de puntos en* $\mathbb{R}^n$ *y* S *una familia de subconjuntos finitos de* X *verificando las siguientes propiedades:*

1. *Si* $x \in X \Rightarrow$ *el 0-símplice* $(x) \in S$*.*
2. *Si* $\tau \in S \Rightarrow \tau \neq \emptyset$*.*
3. *Si* $\sigma \in S$ *y* $\sigma \succ \tau \Rightarrow \tau \in S$*.*

Se dice que el conjunto semisimplicial (X, S) *tiene dimensión* m *con* $m = sup\{dim(\sigma) | \sigma \in S\}$*.*

En un conjunto semisimplicial (X, S), la familia de los símplices S determina al conjunto X, al considerar símplices de dimensión 0. Por este motivo, a un conjunto semisimplicial se le puede denotar simplemente por S.

Todo conjunto semisimplicial, S, tiene asociado un subespacio del espacio euclídeo $\mathbb{R}^m$ que recibe el nombre de realización.

Definición 3.3 (Realización)**.** *Sea* (X, S) *un conjunto semisimplicial. Sea* V *el espacio vectorial real libre sobre el conjunto* X *(es decir, los elementos de* V *son combinaciones lineales finitas de la forma* $\lambda_0 v_0 + \cdots + \lambda_k v_k, v_i \in X, \lambda_i \in \mathbb{R}$*). Se llama realización canónica del conjunto semisimplicial* (X, S) *al siguiente subconjunto de* V*:*

$$|(X, S)| = \{\lambda_0 v_0 + \cdots + \lambda_n v_n \mid \sum_{i=0}^{n} \lambda_i = 1,\ 0 \leq \lambda_i \in \mathbb{R}, (v_0, \ldots, v_n) \in S\}$$

dotado de la topología final respecto a la familia de inclusiones $|(v_0, \cdots, v_n)| \to |(X, S)|$*, donde* $(v_0, \cdots, v_n) \in S$*.*

Es decir, la realización de un conjunto semisimplicial consiste en la unión de las realizaciones de todos sus símplices con la topología final. Notar que si S es finito la realización es un espacio compacto ya que es cerrado y acotado.

Veamos un ejemplo de los conceptos introducidos hasta ahora.

Ejemplo 3.1. *Consideramos el conjunto semisimplicial,* S *que viene representado en la figura 3.1. Hasta ahora hemos visto la definición de* i*-símplice, conjunto semisimplicial y realización. Primero veamos un ejemplo de los distintos símplices que encontramos en* S*.*

- ***0-símplices:*** *son los vértices de la figura:* $(v_0), (v_1), \ldots, (v_6)$*.*
- ***1-símplices:*** *se corresponden con las aristas del conjunto, podemos tomar como ejemplo los símplices* (v_0, v_1)*,* (v_2, v_3)*,* (v_3, v_6)*.*

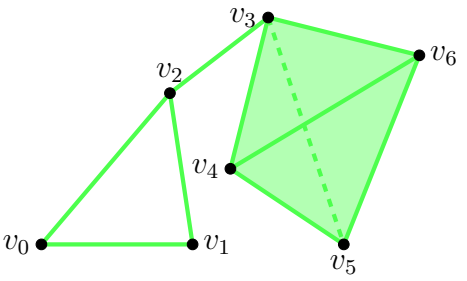


Figura 3.1: Ejemplo conjunto semisimplicial

- ***2-símplices:*** *En nuestro conjunto hay 3:* (v_3, v_4, v_5), (v_4, v_5, v_6), (v_3, v_5, v_6).
- ***3-símplices:*** *Solo hay uno y es* (v_3, v_4, v_5, v_6).

Como ejemplo de ***cara*** *de este 3-símplice podríamos tomar a* (v_3) *o a* (v_3, v_4) *o también a* (v_4, v_5, v_6). *La* ***frontera*** *del 2-símplice* $\sigma = (v_3, v_5, v_6)$ *es* $bd(\sigma) = \{(v_3), (v_5), (v_6), (v_3, v_5), (v_3, v_6), (v_5, v_6)\}$ *y entonces* $int(\sigma)$ *sería el* ***interior*** *del triángulo (lo que resulta de quitar las aristas al triángulo).*

El conjunto semisimplicial S *estaría formado por la lista de todos los 0-símplices, 1-símplices, 2-símplices y 3-símplices. Y la realización* $|S|$ *sería el subespacio de* $\mathbb{R}^3$ *que hemos representado en la figura 3.1.*

Definición 3.4 (Subconjunto semisimplicial)**.** *Sea* S *un conjunto semisimplicial, se dice que* L *es un subconjunto de* S *si es un conjunto semisimplicial tal que* $L \subseteq S$.

A continuación, vamos a ver el subconjunto semisimplicial que más vamos a utilizar en el trabajo, el j-esqueleto.

Definición 3.5 (j-esqueleto)**.** *Sea* S *un conjunto semisimplicial, el subconjunto semisimplicial formado por todos los símplices de* S *de dimensión igual o menor que* j *se denomina* j*-esqueleto de* S *y se denota como* $S^{(j)}$. *El* 0*-esqueleto de* S *consiste en el conjunto de vértices.*

Ejemplo 3.2. *Sea el conjunto semisimplicial el definido anteriormente en la figura 3.1. Veamos un ejemplo de* j*-esqueleto. Tomando* $j = 1$, *el* 1*-esqueleto de* S, $S^{(1)}$, *está formado por todos los vértices y aristas que hay en* S. *Encontramos una representación del mismo en la figura 3.2.*

A continuación, introducimos la noción de estrella de un símplice σ en un conjunto semisimplicial S. Intuitivamente, es el conjunto de símplices en S que tiene a σ como cara.

Definición 3.6 (Estrella)**.** *Sea* S *un conjunto semisimplicial y sea* $\sigma \in S$. *Se llama estrella de* σ *y se denota por* $str(\sigma)$ *al subconjunto semisimplicial:*

$$str(\sigma) = \{\tau \in S \mid \exists \rho \in S,\ \tau \subset \rho,\ \rho \bigcap \sigma \neq \emptyset\}$$

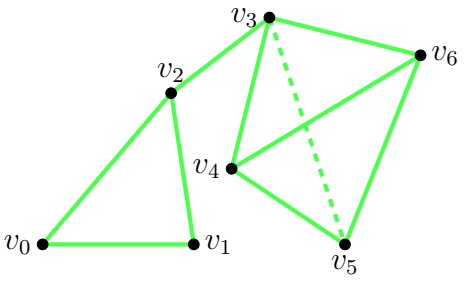


Figura 3.2: 1-esqueleto del conjunto semisimlicial de la figura 3.1

Denotamos por $N(\sigma) = \cup_{\tau \in st(\sigma)} int(\tau)$ al conjunto abierto formado por los interiores de los símplices que forman la estrella de σ.

A continuación se definen los conceptos de maximal y adyacente que aparecerán en la demostración del Teorema de Aproximación Simplicial.

Definición 3.7 (Símplice maximal y adyacente)**.** *Sea S un conjunto semisimplicial y $\sigma \in S$ un símplice, se dice que σ es maximal si no es una cara de ningún otro símplice en S. Se dice que σ' es adyacente a σ si σ y σ' comparten una cara en S.*

A continuación veremos un ejemplo con las últimas nociones introducidas.

Ejemplo 3.3. *Tomando el conjunto semisimplicial definido en la figura 3.1, calculamos la estrella del símplice (v_0). Para ello, buscamos todos los símplices que verifiquen $\sigma \succ (v_0)$. Se cumple para $\sigma = \{(v_0, v_1), (v_0, v_2)\}$. Para tener entero al conjunto hace falta incluir todas las caras de estos símplices. Por lo tanto, str $((v_0)) = \{(v_0), (v_1), (v_2), (v_0, v_1), (v_0, v_2)\}$. Podemos encontrar una representación de la misma en la figura 3.3.*

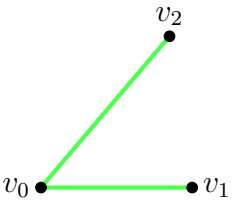


Figura 3.3: Estrella de v_0 del conjunto semisimplicial de la figura 3.1

En cuanto a los símplices maximales tanto (v_0, v_1, v_2) como (v_3, v_4, v_5, v_6) lo son porque no son cara de ningún otro símplice. Los símplices (v_1, v_2) y (v_2, v_3) son adyacentes ya que comparten al vértice (v_2). Del mismo modo que lo son (v_3, v_4, v_6) y (v_4, v_5, v_6), que comparten la arista (v_4, v_6).

Una vez visto este ejemplo, pasamos a introducir unos conceptos que nos permitirán medir el *tamaño* de un símplice de un conjunto semisimplicial.

Definición 3.8 (Diámetro y medida)**.** *Sea S un conjunto semisimplical, el diámetro de un*

símplice σ en S se define como :

$$diam(\sigma) = max\{\|x - y\| \mid x, y \in \sigma\}$$

donde $||\cdot||$ es una métrica.

La medida de S se define como:

$$m(S) = max\{diam(\sigma) | \forall \sigma \in S\}$$

3.2. Subdivisión baricéntrica

Los conjuntos semisimpliciales son estructuras de datos combinatorios que se utilizan para modelar espacios topológicos [3]. Una manera de obtener un modelo más refinado a partir de uno ya existente consiste en subdividir el que ya teníamos en piezas más pequeñas de forma que el resultado sea topológicamente equivalente al que teníamos.

Definición 3.9 (Subdivisión)**.** *Sean S y S' conjuntos semisimpliciales, se dice que S' es una subdivisión de S si:*

1. $|S| = |S'|$
2. *Si existe $\sigma' \in S'$ entonces existe $\sigma \in S$ tal que $\sigma' \subseteq \sigma$*

La subdivisión baricéntrica es un ejemplo concreto de subdivisión de conjuntos semisimpliciales. Antes de introducir la definición de subdivisión baricéntrica nos hace falta saber qué es el baricentro de un i-símplice.

Definición 3.10 (Baricentro)**.** *El baricentro de un i-símplice $\sigma = (v_0, \ldots, v_i)$ se denota por $b(\sigma)$ y viene dado por*

$$b(\sigma) = \sum_{j=0}^{i} \frac{1}{i+1} v_j$$

Con esta definición, la idea de subdivisión baricéntrica surge de manera natural. La definición que proporcionaremos de subdivisión baricéntrica viene dada de manera inductiva.

Definición 3.11 (Subdivisión baricéntrica)**.** *Sea S un conjunto semisimplicial, la subdivisión baricéntrica del 0-esqueleto de S está definida como el conjunto de vértices de S, esto es, $Sd\ S^{(0)} = S^{(0)}$, $Sd\ S$ denota la subdivisión baricéntrica de S. Asumiendo que tenemos $Sd\ S^{(i-1)}$, la subdivisión baricéntrica del (i-1)-esqueleto de S, $Sd\ S^{(i)}$ se construye añadiendo el baricentro de todos los i-símplices de S como nuevos vértices, cada uno de estos vértices que surgen de un i-símplice σ se conecta con todos los símplices que subdividen la frontera de σ.*

La subdivisión baricéntrica de un conjunto semisimplicial S, con $dim(S) = m$, se corresponde con la subdivisión baricéntrica del m-esqueleto de S, por lo que $Sd\ S = Sd\ S^{(m)}$. La iterada aplicación de la subdivisión baricéntrica se denota como $Sd^n\ S$ donde $n \in \mathbb{N}$ es el número de iteraciones.

A continuación veamos un ejemplo de cómo, a partir de un conjunto semisimplicial S, construir su subdivisión baricéntrica $Sd\ S$.

Ejemplo 3.4. *Sea S el conjunto semisimplicial que aparece en la figura 3.4.*

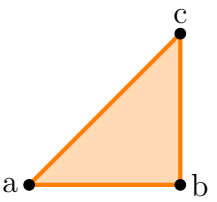


Figura 3.4: Conjunto semisimplicial S

Para construir la subdivisión baricéntrica del conjunto S seguimos los pasos que nos marcan la definición.

- 0-***esqueleto:*** *$Sd\ S^{(0)} = S^{(0)}$. En la figura 3.5 encontramos una representación del mismo.*

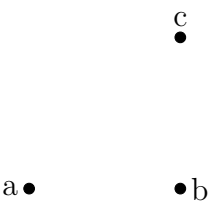


Figura 3.5: $Sd\ S^{(0)}$

- 1-***esqueleto:*** *Calculamos primero los baricentros de cada 1-símplice:*

$$b((a,b)) = \tfrac{1}{2}a + \tfrac{1}{2}b = d,\ b((a,c)) = \tfrac{1}{2}a + \tfrac{1}{2}c = f,\ b((b,c)) = \tfrac{1}{2}b + \tfrac{1}{2}c = e$$

Una vez que ya tenemos los baricentros, lo unimos con cada uno de los símplices que subdividen la frontera del 1-símplice asociado al baricentro. La representación de $Sd\ S^{(1)}$ está en la figura 3.6.

- 2-***esqueleto:*** *En este caso solo tenemos un 2-símplice, calculamos su baricentro.*

$$b((a,b,c)) = \frac{1}{3}a + \frac{1}{3}b + \frac{1}{3}c = g$$

Unimos este punto a todos los símplices que surgen de la subdivisión de la frontera de (a,b,c). Por ejemplo, en la subdivisión de la cara (a,b) aparecen dos nuevos símplices:

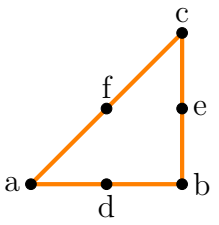


Figura 3.6: $Sd\ S^{(1)}$

(a, d) y (b, d). Uniendo estos símplices al baricentro nos quedan (a, d, g) y (b, d, g). Hacemos esto con todas las caras estrictas de (a, b, c) y nos queda el siguiente conjunto semisimplicial representado en la figura 3.7.

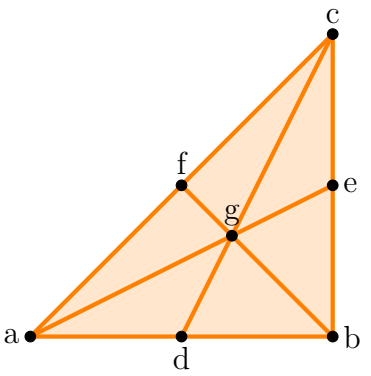


Figura 3.7: $Sd\ S^{(2)}$

Notar que el tamaño de los símplices de $Sd\ S$ ha decrecido con respecto al de los símplices de S. En la figura 3.8, extraída de [14], podemos observar como decrece el tamaño de los símplices cuando calculamos la segunda subdivisión baricéntrica.

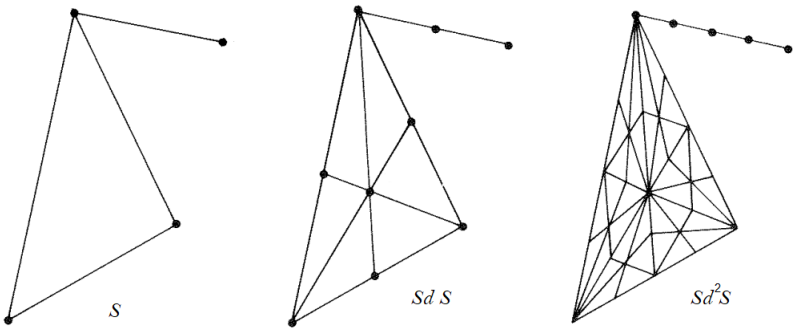


Figura 3.8: Divisiones baricéntricas

Y es que, el tamaño de los símplices de $Sd^n\ S$ decrece rápidamente cuando n crece. Esto es un hecho general que probaremos a continuación. Para la demostración necesitaremos hacer uso del siguiente lema.

Lema 3.1. *Dado S un conjunto semisimplicial, el conjunto $Sd\ S$ es igual a la colección de todos los símplices de la forma:*

$$(b(\sigma_1), b(\sigma_2), \ldots, b(\sigma_n))$$

donde $\sigma_1, \sigma_2, \ldots, \sigma_n$ son símplices de S y verifican $\sigma_1 \succ \sigma_2 \succ \ldots \succ \sigma_n$.

Demostración. Lo probamos por inducción.

Caso base $Sd\ S^{(0)}$

Es inmediato ver que los símplices de $Sd\ S^{(0)}$ son de esa manera. Recordamos que Sd $S^{(0)}$ está formado por el conjunto de los vértices del complejo simplicial S, como $b(v) = v$ con v vértice del complejo, tenemos que Sd $S^{(0)}$ queda determinado por todos los símplices de la forma $(b(v))$, con v vértice se S.

Suponemos cierta la proposición para p. Veamos para $p+1$.

Caso p+1 $Sd\ S^{(p+1)}$

Para construir $Sd\ S^{(p+1)}$ seguimos los pasos que nos indica la definición de la subdivisión baricéntrica. Tomamos σ un $(p+1)$-símplice de S y calculamos $b(\sigma)$. Una vez calculado el baricentro tenemos que unir este punto con todos los símplices que surgen de la subdivisión de la frontera de σ. Estos símplices están en Sd $S^{(p)}$, así que, por la hipótesis de inducción tenemos que son de la forma $(b(\sigma_1), \ldots, b(\sigma_n))$ con $\sigma_1 \succ \sigma_2 \succ \ldots \succ \sigma_n$ y σ_1 es una cara estricta de σ. Al unir estos símplices de Sd $S^{(p)}$ con el baricentro nos quedan símplices de la siguiente manera $(b(\sigma), b(\sigma_1), \ldots, b(\sigma_n))$. □

Ahora, veremos en el siguiente teorema que dado un $\epsilon > 0$ podemos encontrar una subdivisión baricéntrica de manera que el diámetro de los símplices sea menor que dicho ϵ.

Teorema 3.2. *Dado un conjunto semisimplicial finito S y un $\epsilon > 0$, existe un entero n tal que $m(Sd^n\ S) \leq \epsilon$*

Demostración. Como S es finito, $|S|$ es un subespacio del espacio euclídeo $\mathbb{R}^{dim(S)}$ compacto ya que es cerrado y acotado. Sabemos que todas las métricas en $\mathbb{R}^n$ son equivalentes [2]. Recordamos que dos métricas, $||\cdot||_1$ $||\cdot||_2$, son equivalentes si existen constantes $C_1, C_2 > 0$ tales que para cualquier $x \in \mathbb{R}^n$ se verifica

$$C_1||x||_1 \leq ||x||_2 \leq C_2||x||_2$$

Por lo tanto es irrelevante la métrica que utilicemos. Consideramos la métrica del máximo $||x|| = max\{|x_i| : 1 \leq i \leq dim(S)\}$.

Una vez que ya hemos establecido la métrica, veamos cómo vamos a abordar la demostración.

1. Primero vamos a probar que dado σ un p-símplice en S y $\tau \in Sd\ \sigma$ se cumple

$$diam(\tau) \leq \frac{p}{p+1} diam(\sigma)$$

2. Probado 1, la demostración del teorema es inmediata.

Para demostrar 1 nos va a ser muy útil probar primero que dado un símplice σ de S se verifica que $\forall z \in \sigma$

$$||z - b(\sigma)|| \leq \frac{p}{p+1} diam(\sigma) \tag{3.1}$$

Lo probamos para $z = v_0$

$$||v_0 - b(\sigma)|| = \left|\left| v_0 - \sum_{i=0}^{p} \frac{1}{p+1} v_i \right|\right| = \left|\left| v_0 - \left(\frac{1}{p+1} v_0 + \ldots + \frac{1}{p+1} v_p \right) \right|\right| =$$

$$= \left|\left| \frac{p+1}{p+1} v_0 - \frac{1}{p+1} v_0 - \frac{1}{p+1} v_1 - \ldots - \frac{1}{p+1} v_p \right|\right| = \left|\left| \frac{p}{p+1} v_0 - \frac{1}{p+1} v_1 - \ldots - \frac{1}{p+1} v_p \right|\right| =$$

$$= \left|\left| \left(\underbrace{\frac{1}{p+1} + \ldots + \frac{1}{p+1}}_{p \ veces} \right) v_0 - \frac{1}{p+1} v_1 - \ldots - \frac{1}{p+1} v_p \right|\right| = \left|\left| \sum_{i=1}^{p} \frac{1}{p+1} (v_0 - v_i) \right|\right| \leq$$

$$\leq \sum_{i=1}^{p} \frac{1}{p+1} ||v_0 - v_i|| \leq \frac{1}{p+1} ||v_0 - v_1|| + \ldots + \frac{1}{p+1} ||v_0 - v_p|| \leq$$

$$\leq \frac{p}{p+1} max\{||v_0 - v_i|| : i = 1, \ldots, p\} \leq \frac{p}{p+1} diam(\sigma)$$

Lo hemos probado para v_0, es análogo probarlo para los demás $v_j \in \sigma$. Por lo tanto queda probado para cualquier $z \in \sigma$.

A continuación probamos 1. Sea σ un p-símplice y $\tau \in Sd\ \sigma$ se verifica

$$diam(\tau) \leq \frac{p}{p+1} diam(\sigma)$$

Utilizamos la inducción.

- **Caso base** $p = 0$. Este caso es trivial ya que el diámetro de un vértice es 0.
- Suponemos que se cumple para dimensiones menores que p
- **Lo probamos para** p. Como $\tau \in Sd\ \sigma$, $\tau = (b(\sigma_1), \ldots, b(\sigma_n))$ con σ_1 cara de σ y $\sigma_1 \succ \ldots \succ \sigma_n$, por lo tanto, $diam(\tau) = max\{||b(\sigma_i) - b(\sigma_j)|| : i, j = 1, \ldots n \wedge \sigma_i \succ \sigma_j\}$. Sin pérdida de generalidad, tomamos $b(s)$ y $b(s')$ como los puntos que hacen máximos el diámetro de τ. Tenemos entonces que $diam(\tau) = ||b(s) - b(s')||$ con $s \succ s'$.

  Diferenciamos ahora dos casos: si $s = \sigma$ y si $\sigma \succ s$.

Veamos el caso $\boxed{s = \sigma}$: Sea $s' = (v_0, v_1, \ldots, v_m)$ con $m < p$

$$||b(\sigma) - b(s')|| = \left|\left|b(\sigma) - \sum_{i=0}^{m} \frac{1}{m+1} v_i\right|\right| =$$

$$\left|\left|\frac{m+1}{m+1} b(\sigma) - \frac{1}{m+1} v_0 - \frac{1}{m+1} v_1 - \cdots - \frac{1}{m+1} v_m\right|\right| =$$

$$= \left|\left|\frac{b(\sigma) - v_0}{m+1} + \frac{b(\sigma) - v_1}{m+1} + \ldots + \frac{b(\sigma) - v_m}{m+1}\right|\right| \leq \frac{1}{m+1}||b(\sigma) - v_0|| + \cdots +$$

$$+ \frac{1}{m+1}||b(\sigma) - v_m|| \overset{3.1}{\leq} \underbrace{\frac{1}{m+1}\frac{p}{p+1} diam(\sigma) + \cdots + \frac{1}{m+1}\frac{p}{p+1} diam(\sigma)}_{m+1 \ veces} =$$

$$= \frac{p}{p+1} diam(\sigma)$$

Por lo tanto para $s = \sigma$ se verifica que $diam(\tau) \leq \frac{p}{p+1} diam(\sigma)$.

Veamos ahora para $\boxed{\sigma \succ s}$. Sea $dim(s) = q < p$

$$||b(\sigma) - b(s')|| \overset{H.I.}{\leq} \frac{q}{q+1} diam(s) \leq \frac{p}{p+1} diam(s) \leq \frac{p}{p+1} diam(\sigma)$$

La primera desigualdad se sigue por la hipótesis de inducción, la segunda del hecho que $f(x) = \frac{x}{x+1}$ crece para $x > 0$ y la tercera del hecho de que $\sigma \succ s$.

Una vez demostrado 1, probar que dado un $\epsilon > 0$, existe un entero n tal que $m(Sd^n\ S) \leq \epsilon$ es trivial. Recordamos que la medida de un conjunto semisimplicial se corresponde con el diámetro máximo del conjunto. Sea $dim(S) = k$ y sea $\nu \in Sd^n\ S$ tal que ν verifique que $diam(\nu) = m(Sd^n\ S)$. Por la desigualdad probada en 1 tenemos que:

$$m(Sd^n\ S) = diam(\nu) \leq \frac{k}{k+1} m(Sd^{n-1}\ S) \leq \frac{k}{k+1}\frac{k}{k+1} m(Sd^{n-2} S) \leq \cdots \leq \left(\frac{k}{k+1}\right)^n m(S)$$

Si n es lo suficientemente grande, se verifica $\left(\frac{k}{k+1}\right)^n m(S) < \epsilon$. □

A continuación, veremos una de las principales herramientas que utilizaremos en este trabajo, el *Teorema de Aproximación Simplicial*.

3.3. Teorema de Aproximación Simplicial

Un conjunto semisimplicial puede verse como un modelo matemático de una región de un espacio n-dimensional [3]. Mediante el uso de subdivisiones, podemos obtener conjuntos semisimpliciales más y más ajustados. Si pensamos ahora en funciones entre conjuntos semisimiplicales, estas funciones pueden considerarse como extensiones de funciones más simples. Curiosamente, tales funciones pueden considerarse como aproximaciones de funciones continuas

definidas en la realización de espacios topológicos que modelan los conjuntos semisimpliciales. Formalicemos estas nociones.

Definición 3.12 (mapa de vértices)**.** *Sean S y L dos conjuntos semisimpliciales. Un mapa de vértices es una función $\varphi : S^{(0)} \to L^{(0)}$ con la propiedad de que los vértices de cada símplice en S se asocian a los vértices de los símplices de L.*

Un mapa de vértices φ se puede extender a una función continua $\varphi_c : |S| \to |L|$ de la siguiente manera.

Definición 3.13 (función semisimplicial)**.** *Sean S y L dos conjuntos semisimpliciales y sea $\varphi : S^{(0)} \to L^{(0)}$ un mapa de vértices. La función semisimplicial φ_c inducida por φ se define de la siguiente forma. Sea $x \in |S|$ entonces existe un i-símplice $\sigma = (v_0, \ldots, v_i)$ en S y números reales $\lambda_j \geq 0$ tal que*

$$\sum_{j=0}^{i} \lambda_j = 1 \quad y \quad x = \sum_{j=0}^{i} \lambda_j v_j$$

Entonces

$$\varphi_c(x) = \sum_{j=0}^{i} \lambda_j \varphi(v_j)$$

Cualquier mapa de vértices φ induce una función simplicial φ_c, pero, si queremos que dicha función sea una aproximación entre las realizaciones de ambos conjuntos semisimpliciales hace falta añadir una restricción sobre la estrella de cada símplice.

Definición 3.14 (Aproximación simplicial)**.** *Sea S y L conjuntos semisimpliciales y sea $g : |S| \to |L|$ una función continua. Una función simplicial $\varphi_c : |S| \to |L|$ inducida por el mapa de vértices $\varphi : S^{(0)} \to L^{(0)}$ es una aproximación simplicial si para cada vértice v de S se verifica:*

$$g(N(v)) \subset N(\varphi(v))$$

Recordar que $N(v)$ es el conjunto abierto formado por los interiores de los símplices que forman la estrella de v. A esta condición se la conoce como condición de la estrella.

Existen resultados fuertes en conjuntos semisimpliciales. De entre ellos nos interesa el *Teorema de Aproximación Simplicial* que asegura la existencia de una función simplicial que se aproxima a funciones continuas de manera arbitraria.

Antes de dar este resultado veremos una proposición con la que entenderemos mejor la demostración.

Proposición 3.3. *Sea X un espacio secuencialmente compacto y sea ϑ un cubrimiento de X, entonces existe algún número real $\lambda > 0$ tal que para cada $x \in X$, $\exists U \in \vartheta$ con $B(x, \lambda) \subset U$.*

Demostración. Para esta demostración hace falta introducir nuevos conceptos que se alejan del tema del trabajo. La prueba de esta proposición aparece en el Apéndice y se corresponde con la proposición A.2. □

Definición 3.15 (Número de Lebesgue)**.** *A tal número λ que aparece en la proposición 3.3 se le llama número de Lebesgue para el cubrimiento abierto ϑ. Notemos que no es único, si $\lambda > 0$ es número de Lebesgue para ϑ, entonces para todo ϵ con $0 \leq \epsilon \leq \lambda$, ϵ también lo es.*

Teorema 3.4 (Teorema de Aproximación Simplicial)**.** *Sean S y L conjuntos semisimpliciales finitos y $g : |S| \to |L|$ es continua, entonces existe un entero suficientemente grande $t > 0$ tal que $\varphi_c : |Sd^t\ S| \to |L|$ es una aproximación semisimplicial de g.*

Demostración. Consideramos el cubrimiento abierto de $|S|$ formado por $\vartheta = \{g^{-1}(N(u))|u$ vértice de $L\}$. Como $|S|$ es un espacio métrico y compacto es secuencialmente compacto (ver Teorema A.3) y como ϑ es un cubrimiento de $|S|$ existe número de Lebesgue λ. Se elige $n \in \mathbb{N}$ tal que cada símplice de $Sd^n\ S$ tiene un diámetro menor que $\lambda/2$. Entonces todas las estrellas de S tienen diámetro menor que λ, por lo que se verifica que para cualquier v vértice de S, $N(v) \subset g^{-1}(N(u))$. Como $\varphi(v) = u$, se verifica la condición de la estrella que implica la existencia de una aproximación simplicial. □

Los resultados más importantes de esta sección y que utilizaremos en demostraciones futuras son el Teorema 3.2 y el Teorema de Aproximación Simplicial. Este último es el pilar para obtener un resultado cercano al Teorema de Aproximación Universal.

Capítulo 4

Teorema de Aproximación Universal

Dentro de los resultados más importantes en RNA se encuentran los Teoremas de Aproximación Universal [15, 16]. Estos nos aseguran que existe una RNA hacia adelante, con sólo una capa oculta y una función de activación no polinomial, que puede aproximar una función continua en un conjunto compacto de $\mathbb{R}^n$. Sin embargo, estos resultados sólo prueban la existencia y no nos proporcionan un método para encontrar dicha RNA.

En este capítulo demostraremos el *Teorema de Aproximación Universal* y proporcionaremos un método constructivo para encontrar los pesos de una red multicapa hacia adelante con dos capas ocultas que aproxime cualquier función continua entre dos conjuntos triangulables en espacios métricos. Para ello, necesitaremos los resultados ya vistos en el capítulo 3 sobre conjuntos semisimpliciales y los resultados que veremos más adelante como la extensión del Teorema de Aproximación Simplicial y la extensión del Teorema de Aproximación Universal.

Por último, remarcar la importancia de la amplitud de las RNAs. Existen funciones continuas sobre conjuntos compactos que no pueden ser aproximadas por una RNA independientemente de la profundidad de la red. Varios estudios recientes han demostrado que una RNA no es capaz de aproximar una función si la amplitud de dicha red no es mayor a una cota [17].

4.1. Teorema Aproximación Universal

En esta sección enunciaremos el Teorema de Aproximación Universal y veremos una prueba para la cual se necesitará conocer las definiciones de *denso, álgebra* y el teorema de *Stone-Weirstrass*. Cabe destacar que la siguiente demostración no es el objetivo de este trabajo, pero se incluye a continuación para contextualizar e indicar sus limitaciones.

Veamos las definiciones y resultados necesarios.

Definición 4.1 (Denso)**.** *Sean A y B dos subconjuntos del espacio métrico X, se dice que A*

es denso en B si para todo $x \in B$ y $\epsilon > 0$ existe un $y \in A$ tal que

$$||x - y|| < \epsilon$$

Siendo $|| \cdot ||$ la métrica de X.

Definición 4.2 (Álgebra)**.** *Dado una familia A de funciones reales definidas en un conjunto K se dice que es un álgebra si es cerrado por sumas, multiplicaciones y multiplicaciones por escalares.*

Teorema 4.1 (Stone -Weirstrass)**.** *Sea K un conjunto compacto en un espacio topológico cualquiera y $C(K)$ el espacio de las funciones reales continuas definidas sobre K. Si A es un álgebra de $C(K)$ tal que*

- *A no se anula, lo que significa que para todo $x \in K$ existe una función $f \in A$ tal que $f(x) \neq 0$.*
- *A separa puntos, es decir, que para cada par de puntos $x, y \in K$, con $x \neq y$, existe una función f en A tal que $f(x) \neq f(y)$.*

entonces A es denso en $C(K)$.

Encontramos una prueba a este teorema en [18]. A continuación enunciamos el Teorema de Aproximación Universal.

Teorema 4.2 (Teorema de Aproximación Universal)**.** *Sea K un subconjunto compacto de $\mathbb{R}^n$, para cualquier $\epsilon > 0$ y cualquier función $g \in C(K)$, siendo $C(K)$ el conjunto de funciones continuas en K, existe una red neuronal hacia adelante con una capa oculta $N : \mathbb{R}^n \to \mathbb{R}$ definida como $N(x) = f_2(x) \circ f_1(x)$ con $f_1(x) = \phi_{W^{(1)},b_1}(x)$ y $f_2(x) = \phi_{W^{(2)},b_2}(x)$ tal que N es una aproximación de la función g de manera que para todo $x \in K$*

$$|N(x) - g(x)| < \epsilon$$

Este teorema es equivalente a decir que el conjunto de RNA hacia adelante con una capa oculta es denso en el conjunto $C(K)$. El objetivo principal de la prueba que sigue es probar este hecho.

Demostración. Para la realización de la prueba vamos a aplicar el Teorema de Stone - Weirstrass (Teorema 4.1). Para ello definimos los conjuntos sobre los que vamos a trabajar: $K \subset \mathbb{R}^n$ un conjunto compacto, $M^n = \{g \,|g : \mathbb{R}^n \to \mathbb{R}\}$ el conjunto de redes neuronales hacia adelante con una capa oculta, y denotamos por $C^{i,j}$ al conjunto de todas las funciones que van de $\mathbb{R}^i$ a $\mathbb{R}^j$.

Una vez definidos todos los conjuntos, lo siguiente que hacemos es comprobar que se verifiquen las hipótesis del teorema de Stone -Weirstrass:

- El conjunto M^n es obviamente un *álgebra* en K ya que $\forall N_1, N_2 \in M^n$ se verifica que $N_1 + N_2$, $N_1 \cdot N_2$, $\alpha N_1 \in M^n$ con $\alpha \in \mathbb{R}$.
- Lo siguiente que tenemos que ver es que M^n *separa puntos* en K. Notar que si $N \in M^n$ $N = f_2 \circ f_1$ con $f_1 \in C^{n,d}$ y $f_2 \in C^{d,1}$. Sea $g \in C^{d,1}$, si $x, y \in K$ con $x \neq y$, existe una función $f \in C^{n,d}$ tal que $g(f(x)) \neq g(f(y))$. Es muy fácil ver esta implicación, se eligen $a, b \in \mathbb{R}^d$ con $a \neq b$ y una función $h \in C^{n,d}$ tal que $h(x) = a$ y $h(y) = b$. Por lo tanto, $g(h(x)) \neq g(h(y))$ y decimos que M^n separa puntos en K.
- Lo último que tenemos que ver es que M^n *no se anula*. Existe $N \in M^n$ que es constante y distinta de 0 con $N = f_2 \circ f_1$. Para ver esto tomamos $b \in \mathbb{R}^d$ tal que $f_2(b) \neq 0$ y establecer $f_1(x) = b$. Entonces para cualquier $x \in K$, $f_2(f_1(x)) = f_2(b)$, esto nos asegura que M^n no se anula.

Se cumplen todas las hipótesis del teorema de Stone-Weistrass que implica que M^n es denso en el espacio de las funciones continuas en K. □

En el uso práctico del Teorema de Aproximación Universal existen dos importantes inconvenientes:

1. La profundidad de la red crece exponencialmente [19].
2. La demostración no es constructiva, solo nos asegura la existencia. Por lo tanto, no nos proporciona un algoritmo para construir una red neuronal.

Veremos más adelante como solventar estos problemas. A continuación introducimos los espacios triangulables que jugarán un papel muy importante en dicha solución.

4.2. Espacios triangulables

Acabamos de ver que el Teorema de Aproximación Universal es válido para cualquier subconjunto compacto de $\mathbb{R}^n$, sin embargo, el resultado que demostraremos más adelante sólo se cumple para espacios triangulables. Para ello hace falta introducir el concepto de triangulación.

Definición 4.3. *Una triangulación de un espacio topológico X consiste en un conjunto semisimplicial S y un homeomorfismo $\tau : X \to |S|$.*

Un espacio triangulable es aquel que admite una triangulación. Los conjuntos compactos y triangulables se difieren en una propiedad topológica bastante técnica. Los espacios triangulables son todos los compactos y localmente contractibles que pueden incrustarse en $\mathbb{R}^n$ para algún n. No todos los conjuntos compactos en un espacio métrico son localmente contractibles (se da una definición formal en el Apéndice, definición A.4), sin embargo, los conjuntos

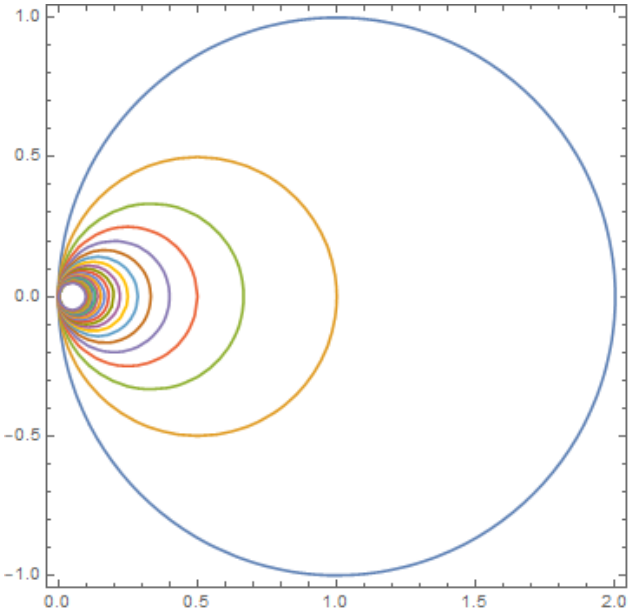


Figura 4.1: Pendiente Hawaiano

que no lo son, son muy raros en $\mathbb{R}^n$ y esta propiedad topológica no tiene aplicación práctica en problemas del mundo real. Por lo tanto, podemos decir que el resultado que probaremos para espacios triangulables es cierto en una gran cantidad de problemas de redes neuronales.

Veamos un ejemplo de un espacio en $\mathbb{R}^2$ que sea compacto pero no localmente contractible. Se trata del pendiente hawaiano, este espacio se define de la siguiente manera.

$$\mathbb{H} = \bigcup_{n=1}^{\infty} \left\{ (x,y) \in \mathbb{R}^2 \mid \left(x - \frac{1}{n}\right)^2 + y^2 = \left(\frac{1}{n}\right)^2 \right\}$$

Mostramos a continuación una representación de dicho espacio en la figura 4.1. Este espacio no es triangulable ya que es compacto pero no es localmente contractible. En [20] encontramos una explicación de porqué este espacio no es localmente contractible.

Para los teoremas y demostraciones que siguen necesitaremos conocer el siguiente concepto.

Definición 4.4 (Medida inducida por una triangulación)**.** *Sea X un espacio topológico triangulable y (S,τ) una triangulación de X. La medida de X inducida por (S,τ) se define como*

$$\tilde{m}_{(S,\tau)}(X) = max\{\tilde{diam}(\sigma) | \sigma \in S\}$$

donde $\tilde{diam}(\sigma) = max\{||x-y|| : x = \tau^{-1}(a),\ y = \tau^{-1}(b)\ con\ a,b \in \sigma\}$ es el diámetro extendido de un símplice.

Ya hemos visto todos los conceptos y resultados necesarios para poder construir nuestra RNA. Esto lo vamos a hacer utilizando el Teorema de Aproximación Simplicial que nos asegura la existencia de una aproximación simplicial de la función que queremos aproximar.

Solo faltaría modelar esa aproximación simplicial a través de una RNA hacia adelante. En la siguiente sección veremos cómo se hace.

4.3. RNA hacia adelante y funciones simpliciales

Dada una función simplicial $\varphi_c : |S| \to |L|$ donde los símplices de los conjuntos semisimpliciales S y L son maximales y con dimensiones máximas, vamos a ver cómo construir una RNA hacia adelante con dos capas ocultas que simule dicha función.

Teorema 4.3. *Dada una función simplicial $\varphi_c : |S| \to |L|$ donde los conjuntos semisimpliciales S y L están compuestos por símplices maximales con dimensiones maximales, podemos definir explícitamente una red neuronal hacia adelante con dos capas ocultas N_φ tal que $\varphi_c(x) = N_\varphi$ para cualquier $x \in |S|$.*

Demostración. Asumimos que $|S| \subset \mathbb{R}^n$ y $|L| \subset \mathbb{R}^m$ y consideramos que S está compuesto por k n-símplices maximales $\{\sigma_1, \ldots, \sigma_k\}$ donde $\sigma_i = (v_0^i, \ldots, v_n^i)$ $\forall i$ y L está compuesto por l m-símplices maximales $\{\mu_1, \ldots, \mu_l\}$ donde $\mu_j = (u_0^j, \ldots, u_m^j)$ $\forall j$.

Consideramos la RNA hacia adelante N_φ con la siguiente arquitectura:

1. Capa de entrada, compuesta por $d_0 = n$ neuronas.
2. Capa oculta primera, compuesta por $d_1 = k(n+1)$ neuronas.
3. Capa oculta segunda, compuesta por $d_2 = l(m+1)$ neuronas.
4. Capa de salida con $d_3 = m$ neuronas.

La idea es codificar el conjunto semisimplicial en las capas ocultas de la RNA. Lo primero, un punto $x \in \mathbb{R}^n$ se transforma en un vector de dimensión $k(n+1)$. Este vector se puede ver como la juxtaposición de k vectores de dimensión $n+1$. Uno por cada uno de los k símplices en S. Cada vector de dimensión $n+1$ representa las coordenadas baricéntricas de x respecto del símplice correspondiente. Si el punto x no pertenece a alguno de los k símplices, las coordenadas pueden salir deformadas (números mayores que uno o negativos). Estas coordenadas las limpiaremos con una función de autocorrección. La matriz $W^{(1)}$ y el sesgo b_1 los obtenemos de las ecuaciones para calcular las coordenadas baricéntricas de la siguiente manera:

$$W^{(i)} = \begin{pmatrix} W_1^{(1)} \\ \vdots \\ W_k^{(1)} \end{pmatrix} \in M_{k(n+1)\times n}$$

donde $W_i^{(1)} \in M_{(n+1)\times n}$ es

$$\begin{pmatrix} v_0^i & \dots & v_n^i \\ 1 & \dots & 1 \end{pmatrix}^{-1} = (W_i^{(i)}|B_i)$$

y $B_i \in \mathbb{R}^{n+1}$ es

$$b_1 = \begin{pmatrix} B_1 \\ \vdots \\ B_k \end{pmatrix}$$

La matriz de pesos entre la primera y la segunda capa, $W^{(2)}$, codifica el mapa de vértices. Como S es codificado por la primera capa oculta con $k(n+1)$ neuronas y L es representado por las $l(m+1)$ neuronas, la matriz está compuesta únicamente por 0s y 1s. El valor 1 representa que los vértices correspondientes están relacionados a través del mapa de vértices y el 0, que no lo están.

La matriz $W^{(2)}$ se define como sigue:

$$W^{(2)} = (W^{(2)}_{s_1,s_2}) \in M_{l(m+1)\times k(n+1)}$$

con

$$s_1 = r + 1 + (j-1)(m+1) \quad j = 1, \dots, l \quad r = 1, \dots, m$$

$$s_2 = t + 1 + (i-1)(n+1) \quad i = 1, \dots, k \quad t = 1, \dots, n$$

y

$$W^{(2)}_{s_1,s_2} = \begin{cases} 1 & si \quad \varphi(v_t^i) = u_r^j \\ \\ 0 & en\ otro\ caso \end{cases}$$

con $b_2 = 0$

La salida de la segunda capa oculta puede ser vista como la juxtaposición de l vectores de dimensión $m+1$. Un vector por cada símplice del conjunto semisimplicial L. Cada vector representa las coordenadas baricéntricas de $\varphi_c(x)$ con respecto al símplice correspondiente. Estas coordenadas están degeneradas, razón por la cual realizamos una división en $\phi^3(y)$. Esta función transforma el vector y en las coordenadas cartesianas de $\varphi_c(x)$. Entonces:

$$W^{(3)} = (W_1^{(3)}, \dots, W_l^{(3)}) \in M_{m\times l(m+1)}$$

siendo $W_j^{(3)} = (u_0^j, \dots, u_m^j)^T$ y $b_3 = 0$

Ya tenemos todos los pesos y los sesgos necesarios para construir la RNA, ya solo nos falta definir las funciones. Para la primera capa:

$$\phi^1(x) = g^k(W^{(1)}x + b_1)$$

La función g^k está definida de la siguiente manera:

$$\begin{array}{rccc} g^k: & \mathbb{R}^{k(n+1)} & \to & \mathbb{R}^{k(n+1)} \\ & y = \begin{pmatrix} y_1 \\ \vdots \\ y_k \end{pmatrix} & \to & \begin{pmatrix} g(y_1) \\ \vdots \\ g(y_k) \end{pmatrix} \end{array}$$

siendo $y_i \in \mathbb{R}^{n+1}$ y la función $g : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1}$ una función correctora que hace cero todas las componentes si hay alguna negativa o mayor que 1. La definimos de la siguiente manera:

$$g(y) = \begin{cases} \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{R}^{n+1} & si\ algun\ y_j < 0 \\ \\ y & otro\ caso \end{cases}$$

Para la segunda capa definimos la siguiente función de activación:

$$\phi^2(x) = W^{(2)}x + b_2$$

En la tercera capa

$$\phi^3 = \frac{\sum_{j=1}^{l} W_j^{(3)} y_j}{t}$$

con $y = \begin{pmatrix} y_1 \\ \vdots \\ y_l \end{pmatrix} \in M^{l(m+1)}$ donde $y_j \in \mathbb{R}^{m+1}$ y t es el número de símplices en los que aparece x. Luego $N_\varphi = f_3 \circ f_2 \circ f_1$ siendo $f_i(y) = \phi^i_{W^{(i)},b_i}(y) = \phi^i(y)$ para $i = 1, 2, 3$. □

La demostración original que se ofrece en [3] presenta fallos en la construcción. En primer lugar, el denominador de la función de ϕ^3 es un vector, por lo que no tiene sentido. La solución a este problema nos la dieron los propios autores del artículo enviándonos una versión mas actualizada del mismo. Sin embargo, en la nueva demostración seguía habiendo errores. El primero consistía en esas coordenadas baricéntricas "deformadas", encontrábamos componentes que eran negativas o mayores que 1. Este problema lo hemos solucionado añdiendo una función autocorrectora que limpia las coordenadas absurdas. Este arreglo hace que la

función ϕ^3 definida en el artículo nuevo quede inservible, por lo tanto, hemos buscado una función que tuviera la misma esencia.

Este teorema que acabamos de ver establece que una RNA hacia adelante con dos capas ocultas puede actuar equivalentemente a una función simplicial. Además, se muestra la arquitectura y la computación específica. Veamos un ejemplo práctico de este teorema. En el capítulo 2 vimos que el perceptrón era incapaz de modelar la función XOR, veamos si es capaz una RNA definida del modo anterior.

Ejemplo

Sea f la función XOR que viene dada de la siguiente manera.

$$\begin{array}{rccl} f: & [-1,1]\times[-1,1]\subset\mathbb{R}^2 & \rightarrow & [-\sqrt{2},\sqrt{2}]\subset\mathbb{R} \\ & (x,y) & \rightarrow & sgn(x)sgn(y)d((x,y),(0,0)) \end{array}$$

donde $sgn(x)$ representa el signo de x, y $d((x,y),(0,0))$ sería la distancia Euclídea del punto (x,y) al origen.

En la figura 4.2 presentamos una triangulación del espacio de entrada S y de salida L. Como vemos $|S|\subset\mathbb{R}^2$ y según la triangulación que hemos escogido tenemos 4 $2-$símplices maximales $\{\sigma_1,\sigma_2,\sigma_3,\sigma_4\}$ donde cada símplice es de la forma: $\sigma_1=(v_0,v_1,v_4)$; $\sigma_2=(v_1,v_2,v_4)$; $\sigma_1=(v_2,v_3,v_4)$; $\sigma_2=(v_3,v_0,v_4)$. Y respecto al conjunto de salida tenemos $|L|\subset\mathbb{R}$. Con la triangulación que hemos considerado tenemos 2 $1-$símplices maximales $\{\mu_1,\mu_2\}$ donde cada símplice es de la forma: $\mu_1=(u_0,u_1)$; $\mu_2=(u_1,u_2)$.

Para determinar la función simplicial $\varphi_c:|S|\rightarrow|L|$ empezamos definiendo el mapa de vértices $\varphi:S^{(0)}\rightarrow L^{(0)}$. Para ello, calculamos el valor correspondiente de cada vértice v de S, $f(v)$, que se corresponde con un vértice de L. Nuestro mapa de vértices es el siguiente.

$$\varphi(v_0)=u_0,\ \varphi(v_1)=u_1,\ \varphi(v_2)=u_0,\ \varphi(v_3)=u_2,\ \varphi(v_4)=u_1$$

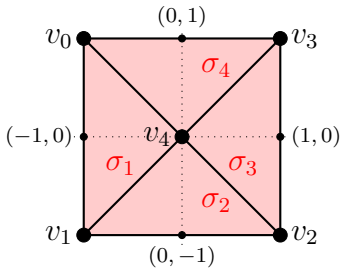
Por lo tanto siendo $x=\sum_{j=0}^{i}\lambda_j v_j$ con $\sum_{j=0}^{i}\lambda_j=1$ la función simplicial es la siguiente:

$$\varphi_c(x)=\sum_{j=0}^{i}\lambda_j\varphi(v_j)$$

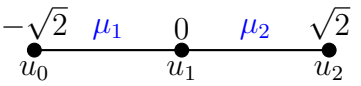
.

La demostración del teorema nos dice que la arquitectura de la RNA va a ser de la siguiente manera:

- Capa de entrada: $n=2$ neuronas
- Primera capa oculta: como $k=4$ y $n+1=3$ tendrá 12 neuronas



(a) Conjunto semisimplicial S



(b) Conjunto semisimplicial L

Figura 4.2: Triangulación del espacio de entrada y de salida

- Segunda capa oculta: como $l = 2$ y $m + 1 = 2$ tendrá 4 neuronas
- Capa de salida: $m = 1$ neuronas

Procedemos a calcular la matriz $W^{(1)} \in M_{12\times 2}$ y $b_1 \in \mathbb{R}^{12}$, para ello calculamos las submatrices:

$\boxed{W_1^{(1)}}$

$$\begin{pmatrix} v_0 & v_1 & v_4 \\ 1 & 1 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} -1 & -1 & 0 \\ 1 & -1 & 0 \\ 1 & 1 & 1 \end{pmatrix}^{-1} = \left(\begin{array}{cc|c} -1/2 & 1/2 & 0 \\ -1/2 & -1/2 & 0 \\ 1 & 0 & 1 \end{array}\right) = (W_1^{(1)}|B_1)$$

$\boxed{W_2^{(1)}}$

$$\begin{pmatrix} v_1 & v_2 & v_4 \\ 1 & 1 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} -1 & 1 & 0 \\ -1 & -1 & 0 \\ 1 & 1 & 1 \end{pmatrix}^{-1} = \left(\begin{array}{cc|c} -1/2 & -1/2 & 0 \\ 1/2 & -1/2 & 0 \\ 0 & 1 & 1 \end{array}\right) = (W_2^{(1)}|B_2)$$

$\boxed{W_3^{(1)}}$

$$\begin{pmatrix} v_2 & v_3 & v_4 \\ 1 & 1 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 1 & 0 \\ -1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}^{-1} = \left(\begin{array}{cc|c} 1/2 & -1/2 & 0 \\ 1/2 & 1/2 & 0 \\ -1 & 0 & 1 \end{array}\right) = (W_3^{(1)}|B_3)$$

$\boxed{W_4^{(1)}}$

$$\begin{pmatrix} v_3 & v_0 & v_4 \\ 1 & 1 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & -1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}^{-1} = \left(\begin{array}{cc|c} 1/2 & 1/2 & 0 \\ -1/2 & 1/2 & 0 \\ 0 & -1 & 1 \end{array}\right) = (W_4^{(1)}|B_4)$$

Por lo tanto tenemos que $\phi^1(x) = g^k(W^{(1)}x + b_1)$.

Calculamos ahora $W^{(2)} \in M_{4\times 12}$. Para la construcción de esta matriz resulta más intuitivo si colocamos los símplices de L μ_i en las filas y los símplices de S σ_i en las columnas. De esta manera nos será más fácil rellenar la matriz. Veamos un ejemplo de cómo se rellena la matriz. Tomamos el vértice $v_0 \in \sigma_1$ y calculamos su valor $\varphi(v_0) = u_0$, por lo tanto nuestro mapa de vértices envía el vértice v_0 a u_0. Vamos a las columnas de v_0 y colocamos un 1 en la fila que se corresponda con la de u_0 y marcamos un 0 en las demás. Hacemos esto con los demás vértices de S hasta completar la matriz y nos queda lo siguiente:

$$
\begin{array}{cc}
 & \begin{array}{cccccccccccc} \multicolumn{3}{c}{\sigma_1} & \multicolumn{3}{c}{\sigma_2} & \multicolumn{3}{c}{\sigma_3} & \multicolumn{3}{c}{\sigma_4} \end{array} \\
 & \begin{array}{cccccccccccc} v_0 & v_1 & v_4 & v_1 & v_2 & v_4 & v_2 & v_3 & v_4 & v_3 & v_0 & v_4 \end{array} \\
\begin{array}{c} \mu_1 \\ \\ \mu_2 \\ \end{array} & \begin{array}{c} u_0 \\ u_1 \\ u_1 \\ u_2 \end{array} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}
\end{array}
$$

Sea $y = \phi^1(x)$, $\phi^2(y) = W^{(2)}y$.

Por último calculamos la matriz $W^{(3)} \in M_{1\times 4}$

$$W^3 = (W_1^{(3)}, W_2^{(3)}) = (u_0, u_1, u_1, u_2) = (-\sqrt{2}, 0, 0, \sqrt{2})$$

Con ϕ^3 definida como hemos visto en la demostración.

Una vez que ya tenemos la RNA, $N_\varphi = \phi^3(\phi^2(\phi^1(x)))$, podemos comprobar que en efecto es una aproximación de la función f. Considerando el punto $x = (-1, 1)^T$, la función f nos devuelve $f(x) = -\sqrt{2}$, veamos qué nos devuelve nuestra RNA.

Calculamos $\phi^1(x)$:

$$
\phi^1(x) = \begin{bmatrix} -1/2 & 1/2 \\ -1/2 & -1/2 \\ 1 & 0 \\ -1/2 & -1/2 \\ 1/2 & -1/2 \\ 0 & 1 \\ 1/2 & -1/2 \\ 1/2 & 1/2 \\ -1 & 0 \\ 1/2 & 1/2 \\ -1/2 & 1/2 \\ 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} -1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1 \\ 2 \\ -1 \\ 0 \\ 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} \xrightarrow{g^k} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = y
$$

$$\text{Calculamos } \phi^2(y) = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \\ 0 \end{bmatrix} = z$$

Por último aplicamos $\phi^3(z)$. Notar que $x = (-1, 1)^T$ es vértice de dos símplices, por lo tanto $t = 2$.

$$\phi^3(z) = \frac{\begin{pmatrix} -\sqrt{2} & 0 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 & \sqrt{2} \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \end{pmatrix}}{2} = \frac{-2\sqrt{2}}{2} = -\sqrt{2}$$

En efecto, obtenemos el mismo resultado. Hemos construido una RNA que simula la función simplicial φ_c. Esta función es una aproximación de la función f que para cualquier v vértice de S verifica que $\varphi_c(v) = f(v)$. Sin embargo, si tomamos un $x \in |S|$ tal que x no sea vértice de S, $\varphi_c(x) \neq f(x)$, es más, el error máximo que se comete es próximo a la unidad. Esto lo podemos ver en el Apéndice B. Nuestro objetivo es buscar una función simplicial φ_c que se aproxime a la función f tan cerca como queramos ya que, una vez definida φ_c, por el Teorema 4.3 podemos construir una RNA que simule a φ_c. En la siguiente sección veremos cómo conseguir una función simplicial más próxima a f.

4.4. Extensión del Teorema de Aproximación Simplicial

En esta sección, proporcionaremos una extensión del Teorema de Aproximación Simplicial (Teorema 3.4) y un algoritmo explícito para construir una aproximación simplicial tan próxima como queramos a la función $g : |S| \to |L|$ con S y L dos conjuntos semisimpliciales.

Observar que el Teorema de Aproximación Simplicial se refiere a cualquier función continua. La continuidad es una propiedad de las funciones en espacios topológicos que no necesariamente se cumple en espacios métricos. El resultado siguiente introduce el concepto de métrica en la aproximación simplicial.

Proposición 4.4. *Dado $\epsilon > 0$ y una función continua $g : |S| \to |L|$ con S y L conjuntos*

semisimpliciales, existen $t_1, t_2 > 0$ *tal que* $\varphi_c : |Sd^{t_1} S| \to |Sd^{t_2} L|$ *es una aproximación simplicial de* g *y* $||g - \varphi_c|| < \epsilon$.

Demostración. Por el Teorema 3.2 existe t_2 tal que $m(Sd^{t_2} L) \leq \epsilon$. Entonces, por el Teorema de Aproximación Simplicial (Teorema 3.4) existe t_1 tal que $\varphi_c : |Sd^{t_1} S| \to |Sd^{t_2} L|$ es una aproximación simplicial de g: ver la figura 4.3.

$$\begin{array}{ccc} |S| & \xrightarrow{g} & |L| \\ \uparrow & & \uparrow \\ S & & L \\ \downarrow{\scriptstyle Sd} & & \downarrow{\scriptstyle Sd} \\ Sd^{t_1} S & & Sd^{t_2} L \\ \uparrow & & \uparrow \\ |Sd^{t_1} S| & \xrightarrow{\varphi_c} & |Sd^{t_2} L| \end{array}$$

Figura 4.3: Esquema aproximación simplicial.

Además, $||g - \varphi_c|| \leq \epsilon$ porque $m(Sd^{t_2}\ L) \leq \epsilon$. Veamos esta última implicación, como $m(Sd^{t_2}\ L) \leq \epsilon$ esto significa que la distancia máxima de los vértices de un simplice de $Sd^{t_2}\ L$ es menor o igual a ϵ. Dado un vértice $v \in Sd^{t_2}\ L$ se verifica que $g(v) = \varphi_c(v)$, por lo tanto $||g - \varphi_c|| \leq \epsilon$. □

Teorema 4.5. *Dada una función* $g : |S| \to |L|$ *con* S *y* L *conjuntos semisimpliciales y dado un* $\epsilon > 0$*, se puede definir una RNA hacia adelante con dos capas ocultas* N *tal que* $||N - g|| \leq \epsilon$.

Demostración. Por la Proposición 4.4 existe una aproximación simplicial, φ_c, de g tal que $||g - \varphi_c|| < \epsilon$. Por lo tanto por el Teorema 4.3 existe N_φ tal que $\varphi_c = N_\varphi$. □

4.5. Extensión del Teorema de Aproximación Universal

Los resultados de las secciones anteriores pueden ser extendidos a espacios triangulables. En este caso, la propuesta es constructiva si se conoce el homeomorfismo de la triangulación. En las anteriores secciones, hemos probado que una función continua entre conjuntos triangulables puede ser aproximada usando el Teorema de Aproximación Simplicial (Teorema 3.4).

En esta sección usando la nueva version del Teorema de Aproximación Simplicial (Teorema 4.8) podemos dar una versión constructiva del Teorema de Aproximación Universal que puede aproximar cualquier función continua todo lo cerca que se quiera.

Proposición 4.6. *Sea (S, τ) una triangulación del espacio X. Para cualquier $\epsilon > 0$ existe un n y un $\gamma > 0$ tal que si $m(Sd^n S) \leq \gamma$ entonces $\tilde{m}_{(Sd^n S, \tau_S)}(X) \leq \epsilon$.*

Demostración. Considerar $x, y_0 \in |S|$. Si x e y_0 están en $\sigma_0 \in S$ entonces $||x - y_0|| \leq diam(\sigma_0)$ y $||\tau(x) - \tau(y_0)|| \leq \tilde{diam}(\sigma_0)$. Repetimos el razonamiento para $Sd\ S$, tomamos $x, y_0 \in |Sd\ S|$. Puede ocurrir que ambos puntos x e y_0 estén o no estén en el mismo símplice de $Sd\ S$. Veamos los dos casos.

- Si x e y_0 pertenecen al mismo símplice, tomar $y_1 = y_0$.
- Si no pertenecen al mismo símplice, tomar y_1 tal que $x, y_1 \in \sigma_1 \in Sd\ \sigma_0$. Por lo tanto $||x - y_1|| \leq diam(\sigma_1) \leq diam(\sigma_0)$. Además $||\tau(x) - \tau(y_0)|| \leq \tilde{diam}(\sigma_1) \leq \tilde{diam}(\sigma_0)$.

Iteramos el proceso y obtenemos lo siguiente. $||x - y_n|| \leq diam(\sigma_n) \leq \ldots \leq diam(\sigma_1) \leq diam(\sigma_0)$ y $||\tau(x) - \tau(y_n)|| \leq \tilde{diam}(\sigma_n) \leq \ldots \leq \tilde{diam}(\sigma_1) \leq \tilde{diam}(\sigma_0)$. Definimos la secuencia $\{y_i\}_{i=0}^n$ que converge a x, por lo tanto, dado un $\epsilon > 0$ existe un n tal que $||\tau(x) - \tau(y_n)|| \leq \epsilon$. Sin pérdida de generalidad suponemos que $\tilde{diam}(\sigma_n) = \tilde{m}_{(Sd^n\ S, \tau)}(X)$. Entonces podemos considerar $\gamma = m(Sd^n\ S)$. □

Corolario 4.7. *Dado un $\epsilon > 0$ y una triangulación (S, τ) de X, existe t tal que $\tilde{m}(Sd^t S) \leq \epsilon$.*

Demostración. Por el Teorema 3.2 existe t' tal que $m(Sd^{t'} S) \leq \gamma$. Y por la Proposición 4.6, existe t tal que $\tilde{m}(Sd^t S) \leq \epsilon$. □

Finalmente, vamos a dar el resultado principal del capítulo. Dada una función continua g entre dos espacios triangulables X e Y, podemos obtener dos conjuntos semisimpliciales S y L asociados a ellos y una aproximación simplicial φ_c entre ellos que aproxima la función g.

Proposición 4.8. *Sean X,Y dos espacios topológicos triangulables, $g : X \to Y$ una función continua y $\epsilon > 0$. Entonces, existen dos triangulaciones (S, τ_S) y (L, τ_L) de X e Y, respectivamente, y una aproximación simplicial $\varphi_c : |Sd^{t_1} S| \to |Sd^{t_2} L|$ tal que $||g - \tau_L^{-1} \circ \varphi_c \circ \tau_S|| \leq \epsilon$.*

Demostración. Por el corolario 4.7, existe t_2 tal que $\tilde{m}_{(Sd^{t_2} L, \tau_L)}(Y) \leq \epsilon$. Luego, por el Teorema 3.4, existen t_1 y un mapa de vértices $\varphi : (Sd^{t_1} S)^{(0)} \to (Sd^{t_2} L)^{(0)}$ tal que $\varphi_c : |Sd^{t_1} S| \to |Sd^{t_2} L|$ es una aproximación simplicial de $\tau_L \circ g \circ \tau_S^{-1}$. Teniendo en cuenta que $|Sd^{t_1} S| = |S|$ y $|Sd^{t_2} L| = |L|$. Finalmente, como $\tilde{m}_{(Sd^{t_2} L, \tau_L)}(Y) \leq \epsilon$ entonces $||g - \tau_L^{-1} \circ \varphi_c \circ \tau_S|| \leq \epsilon$. Podemos ver un esquema en la figura 4.4.

□

Teorema 4.9. *Dada una función continua $g : X \to Y$, dos triangulaciones (S, τ_S) y (L, τ_L) de X e Y, respectivamente y un $\epsilon > 0$. Podemos definir una RNA hacia adelante N con dos capas ocultas que se aproxima a g de manera que $|N - g| < \epsilon$.*

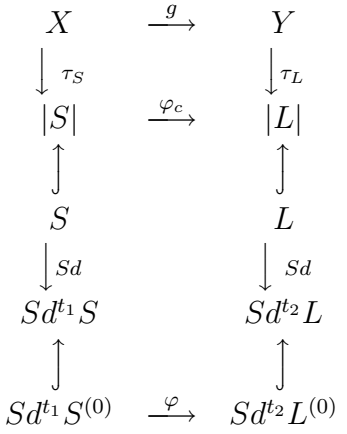


Figura 4.4: Esquema aproximación univeral.

Demostración. Por la Proposición 4.8, existe una aproximación simplicial φ_c de g tal que $|g - \varphi_c| \leq \epsilon$. Además, por el Teorema 4.3, existe N tal que $N = \varphi_c$ en todo su dominio. Por lo tanto, N aproxima g verificando que $|N - g| \leq \epsilon$. □

Una de las limitaciones de este resultado es el hecho de que partimos con el complejo simplicial ya construido, cuando normalmente, tenemos una nube de puntos sin ningún tipo de estructura. Por lo tanto, es necesario construir una triangulación a partir de los mismos. Esto lo veremos en la siguiente sección.

4.6. Complejos de Česch

En esta última sección vamos a ver cómo a partir de una nube de puntos en un espacio métrico podemos computar su conjunto semisimplicial asociado.

Definición 4.5 (Complejo de Česch)**.** *Si $N = \{p_0, p_1, \ldots, p_k\}$ con $p_i \in \mathbb{R}^d$ para todo $0 \leq i \leq k$ y $\epsilon > 0$, el complejo de Česch de radio ϵ, denotado por $C_\epsilon(N)$, es el conjunto semisimplicial cuyos m-símplices se corresponden con los subconjuntos $\sigma \subset N$ que verifican*

$$\bigcap_{p_i \in \sigma} D(p_i, \epsilon) \neq \emptyset$$

Denotando por $D(x, r)$ a la bola cerrada con centro x y radio r.

Dependiendo del tamaño de ϵ obtenemos distintos complejos de Česch. Esto lo vemos en el siguiente ejemplo.

Ejemplo 4.1. *Sean los puntos $N = \{v_0 = (0, 2.4), v_1 = (1.4, 0), v_2 = (-1.4, 0)\}$ de $\mathbb{R}^2$, computamos los distintos complejos de Česch que surgen al tomar distintos radios. En la figura 4.5 vemos tres complejos distintos.*

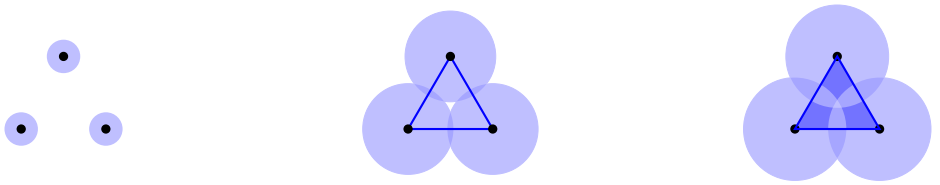


Figura 4.5: Complejos de Česch para $\epsilon = 0.5$, $\epsilon = 1.5$, $\epsilon = 1.7$

Como vemos para el radio $\epsilon = 0.5$*, el complejo de Česch está formado simplemente por los* 0*-símplices:* $C_{0.5}(N) = \{(v_0), (v_1), (v_2)\}$*. Con radio* $\epsilon = 1.5$ *obtenemos el siguiente conjunto* $C_{1.5}(N) = \{(v_0),\ (v_1),\ (v_2),\ (v_0, v_1),\ (v_0, v_2),\ (v_1, v_2)\}$ *y por último,* $C_{1.7}(N) = \{(v_0),\ (v_1),\ (v_2),\ (v_0, v_1),\ (v_0, v_2),\ (v_1, v_2),\ (v_0, v_1, v_2)\}$

Existen otros complejos, como el complejo de Vietoris-Rips [21], que ofrecen distintas triangulaciones a partir de una nube de puntos. En cualquier caso, todos ellos nos transforman el conjunto de puntos en un complejo simplicial. Una vez que tenemos el complejo simplicial, podemos proseguir con el método constructivo que hemos visto a lo largo del trabajo. De este modo, es posible construir redes neuronales que modelen relaciones entre nubes de puntos.

Capítulo 5

Conclusiones

Con el descubrimiento de las redes neuronales artificiales y su algoritmo de aprendizaje (propagación hacia atrás) se devolvió la ilusión a muchos científicos que perdieron su motivación tras conocer las limitaciones del perceptrón. En estos últimos 40 años, se han hecho grandes aportaciones al estudio de las redes neuronales, entre ellas se encuentra la demostración de que las redes neuronales son aproximadores universales.

En este trabajo hemos visto un método efectivo para construir una RNA hacia adelante con dos capas ocultas que aproxima cualquier función continua entre espacios triangulables. Se trata de un método que no requiere ningún tipo de entrenamiento, en el que los pesos y el sesgo se calculan de manera exacta. Esto es una gran ventaja ya que el proceso de entrenamiento es complejo y muy poco eficiente. Otra ventaja de este método es que sabemos a priori la arquitectura de la RNA, es decir, sabemos el número de capas ocultas y el número de neuronas necesarias para obtener la precisión deseada. Sin embargo, tiene tres inconvenientes. Puede ser una tarea difícil encontrar los homeomorfismos entre espacios triangulables y los conjuntos semisimpliciales. La segunda desventaja es que el resultado clásico de aproximación es válido para conjuntos compactos y el resultado que hemos visto en este trabajo solo es válido para conjuntos triangulables. No obstante, casi todos los problemas del mundo real están cubiertos. Y, el último inconveniente hace referencia a la magnitud de las redes neuronales. Con una triangulación relativamente sencilla obtenemos un gran número de neuronas en las capas ocultas y esto provoca un alto coste computacional a la hora de construir dichas redes.

Por último, dentro de los problemas que nos hemos topado al realizar este trabajo, se encuentran las erratas de la demostración constructiva en la que se basa todo el trabajo. Sin embargo, tanto los autores de [3] como mi tutor, me han ayudado para llegar a una solución. También hemos tenido dificultades para encontrar una notación que fuera sencilla y manejable ya que la que nos proporciona el artículo sobre el que se basa el trabajo es bastante confusa y difícil de manejar.

Bibliografía

[1] Miguel Angel Hernández Verón. Apuntes de métodos numéricos. Universidad de La Rioja, 2018.

[2] Manuel Bello Hernández. Apuntes de análisis real y funcional. Universidad de La Rioja, 2019.

[3] Rocio Gonzalez-Diaz, Miguel A. Gutiérrez-Naranjo y Eduardo Paluzo-Hidalgo. Two-hidden-layer feedforward networks are universal approximators: A constructive approach. arXiv:1907.11457, 2019.

[4] Simon Haykin. *Neural Networks - A Comprehensive Foundation*. Pearson Education, 2001.

[5] Jónathan Heras. Apuntes de aprendizaje automático. Universidad de La Rioja, 2020.

[6] José Manuel Anguas Pérez. Apuntes. El Perceptrón. Universidad de Sevilla.

[7] Juan Miguel Ortiz de Lazcano. Apuntes. El Perceptrón Simple. Universidad de Málaga.

[8] Avinash Sharma. The Theory Of Everything understanding activation functions in neural networks. `https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0`.

[9] Fernando Revilla. Hiperplanos. `http://fernandorevilla.es/blog/2014/05/19/hiperplanos/`, 2014.

[10] R.Rojas. *Neural Networks. A systematic Introduction*. Springer, 1996.

[11] The Asimov Institute the neural network zoo. `https://www.asimovinstitute.org/neural-network-zoo/`.

[12] José Manuel Anguas Pérez. Apuntes. Redes Neuronales Multicapa. Universidad de Sevilla.

[13] Mª Teresa Rivas. Apuntes de geometría y topología de superficies. Universidad de La Rioja, 2019.

[14] James R. Munkres. *Elements of Algebraic Topology*. Addison-Wesley Publishing Company, 1984.

[15] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control Signal and Systems*, 2:303–314, 1989.

[16] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, March 1991.

[17] Boris Hanin y Mark Sellke. Approximating continuous functions by relu nets of minimal width. arXiv:1710.11278, 2017.

[18] Julio Bernués. El teorema de stone-weierstrass. *La Gaceta de la RSME*, 13(4):705–711, 2010.

[19] Ibrohim Nosirov y Jeffrey M. Hokanson. A numerical investigation of the minimum width of a neural network. arXiv:1910.13817, 2019.

[20] nLab locally contractible space. `https://ncatlab.org/nlab/show/locally+contractible+space`.

[21] Jorge Cordero. *Simplicial complexes associated with cloud points*. Eindhoven University of Technology, 2018.

Apéndice A

Definiciones y Teoremas

Definición A.1. *Dado un conjunto convexo V de $\mathbb{R}^n$, una función $f : V \to \mathbb{R}^n$ se dice convexa si $\forall x, y \in V$ y $\forall t \in [0,1]$, se tiene lo siguiente.*

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$$

Proposición A.1. *Si f es convexa y existe un $x \in \mathbb{R}^n$ tal que $f(x)$ es un mínimo de f, entonces dicho $f(x)$ es un mínimo global de f.*

Demostración. Supongamos que para un cierto x, $f(x)$ es un mínimo de f. Suponemos que existe un y tal que verifica que $f(x) > f(y)$, es decir, $f(x)$ no es un mínimo global.

Como f es convexa, se debe cumplir que:

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y),$$

pero f tiene un mínimo en x, por lo que existe en un entorno E de x tal que:

$$f(z) \geq f(x), \quad \forall z \in E$$

Entonces también existe $t^* \in (0,1)$ tal que:

$$f(t^*x + (1-t^*)y) > t^*f(x) + (1-t^*)f(y)$$

que es absurdo porque f es convexa. Por tanto, no puede existir dicho y, concluyendo que f tiene un mínimo global en x. □

Definición A.2 (Secuencialmente compacto)**.** *Diremos que un espacio métrico X es secuencialmente compacto si toda sucesión en X posee alguna subsucesión convergente.*

Proposición A.2 (Lema del número de Lebesgue asociado a un cubrimiento)**.** *Sea (X, d) un espacio métrico secuencialmente compacto y sea ϑ un cubrimiento abierto de X. Entonces existe algún número real $\epsilon = \epsilon(\vartheta) > 0$ tal que para cada $x \in X$, $\exists U_x \in \vartheta$ con $B(x, \epsilon) \subset U_x$*

Demostración. Supongamos que (X, d) es un espacio métrico secuencialmente compacto. Sea ϑ un cubrimiento abierto de X, supongamos que ϑ no tiene número de Lebesgue asociado. Entonces para cada $n \in \mathbb{N}$, $\frac{1}{n} > 0$ no es número de Lebesgue para ϑ. Por lo tanto, $\exists x_n \in X$ tal que $\forall U \in \vartheta$ se tiene que $B(x_n, \frac{1}{n}) \subset U$.

Consideramos la sucesión $(x_n)_{n\in\mathbb{N}}$. Como (X, d) es secuencialmente compacto existe una subsucesión $(x_{n_k})_{k\in\mathbb{N}}$ convergente, es decir que $\exists x_0 \in X$ tal que $(x_{n_k}) \to x_0$. Como ϑ es un cubrimiento abierto de X existe un $U_o \in \vartheta$ tal que $x_0 \in U_0$. Por ser U_0 abierto $\exists \epsilon > 0$ tal que $B(x_0, \epsilon) \subset U_0$. Como $(x_{n_k}) \to x_0$ considerando $\frac{\epsilon}{2} > 0$, existe $N \in \mathbb{N}$ tal que para todo $n_k > N$ se tiene $d(x_{n_k}, x_0) < \frac{\epsilon}{2}$. Ahora consideramos un n_k verificando $n_k > max\{N, \frac{2}{\epsilon}\}$. Entonces $B(x_{n_k}, \frac{1}{n_k}) \subset B(x_0, \epsilon) \subset U_0$. Absurdo, contradice la elección de los términos de la sucesión. □

Teorema A.3 (Caracterización de la compacidad en espacios métricos)**.** *Sea X un espacio métrico, entonces son equivalentes:*

1. *X es compacto*
2. *X es secuencialmente compacto.*

Demostración. Encontramos una prueba a este teorema en [13]. □

Definición A.3. *Sean X y Y espacios y sean $f_0, f_1 : X \to Y$ funciones. Se dice que f_0 es homotópica a f_1, denotado por $f_0 \simeq f_1$ si existe $F : X \times I \to Y$ tal que para cualquier $x \in X$ F verifica $F(x, 0) = f_0(x)$ y $F(x, 1) = f_1(x)$. A esa función F se le denomina homotopía de f_0 a f_1.*

Definición A.4 (Localmente contractible)**.** *Un espacio X se dice que es localmente contractible en el punto $x_0 \in X$ si para cualquier $\epsilon > 0$ existe $\delta > 0$ con $\delta < \epsilon$ y una homotopía $F : B(x_0, \delta) \times I \to B(x_0, \epsilon)$ tal que $F(x, 0) = x_0$ y $F(x, 1) = x$ para cada $x \in B(x_0, \delta)$.*

Apéndice B

Notebooks

Los noteboks de Google son documentos realizados en la plataforma de Google Colaboratory. Esta plataforma es un producto de Google Research y permite que todos los usuarios de Google puedan escribir y ejecutar código arbitrario de Python en el navegador.

A continuación mostraremos en orden los archivos que siguen:

1. Documento de Mathematica que hemos utilizado para calcular de forma resumida los valores de los pesos y el sesgo en el ejemplo del aprendizaje del perceptrón para la función lógica XOR.
2. Notebook en el que mostramos el aprendizaje del perceptrón de las funciones lógicas OR y XOR. Podemos acceder a este notebook a través del siguiente link: Perceptrón.
3. Notebook en el que construimos la RNA hacia adelante como indica la demostración del Teorema 4.3. Al igual que en el caso anterior, podemos acceder al notebook pinchando en RNA.

Función lógica XOR

```
aprende[{w1_, w2_, b_, η_, z_, y_, x1_, x2_} ] :=
 Block[{k1, k2, k3}, k1 = w1 + η (z – y) x1 ;
 bloquea
  k2 = w2 + η (z – y) x2 ; k3 = b – η (z – y);
  {k1, k2, k3, η, z,
   If[k1 * x1 + k2 * x2 – k3 < 0, 0, 1], x1, x2}]
   si

PesosySesgo =
  NestList[aprende, {1, 1, 0.5, 0.25, 0, 1, 1, 1}, 3];
  lista de resultados anidados

TableForm[PesosySesgo,
forma de tabla
 TableHeadings → {Range[1, 10, 1],
 cabeceras de tabla   rango
   {"w₁", "w₂", "b", "η", "z", "y", "x₁", "x₂"}}]
```

	w_1	w_2	b	η	z	y	x_1	x
1	1	1	0.5	0.25	0	1	1	1
2	0.75	0.75	0.75	0.25	0	1	1	1
3	0.5	0.5	1.	0.25	0	1	1	1
4	0.25	0.25	1.25	0.25	0	0	1	1

PERCEPTRÓN

Algoritmo de aprendizaje

Definimos el perceptrón como hemos visto:

$$y = \phi_{W,\ b}(x) = f(Wx + b)$$

con f la función de paso:

$$f(v) = \begin{cases} 1 & si \quad v \geq 0 \\ 0 & si \quad v<0 \end{cases}$$

In [0]:
```
import numpy as np
```

Definimos la función de paso:

In [0]:
```
def funcionPaso(v):
  if v<0: x=0
  else: x=1
  return x
```

Definimos el perceptrón:

In [0]:
```
def perceptron (w1,w2,b):
  return lambda x1,x2: funcionPaso(w1*x1+w2*x2-b)
```

Definimos unas funciones para mostrar los datos de entrada y para mostrar las predicciones que realiza el perceptrón:

In [0]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def muestra_dataset(dataset):
    X, Y = dataset
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])
    plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=cm_bright,
                    edgecolors='k')
    plt.show()

def muestra_dataset_predict(dataset,w1,w2,b):
    X, Y = dataset
    cm = plt.cm.RdBu
    cm_bright = ListedColormap(['#FF0000', '#0000FF'])
    y_pred = [perceptron(w1,w2,b)(x1,x2)==1 for (x1,x2) in X]
    plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap=cm_bright,
                    edgecolors='k')
    plt.show()
```

FUNCIÓN OR

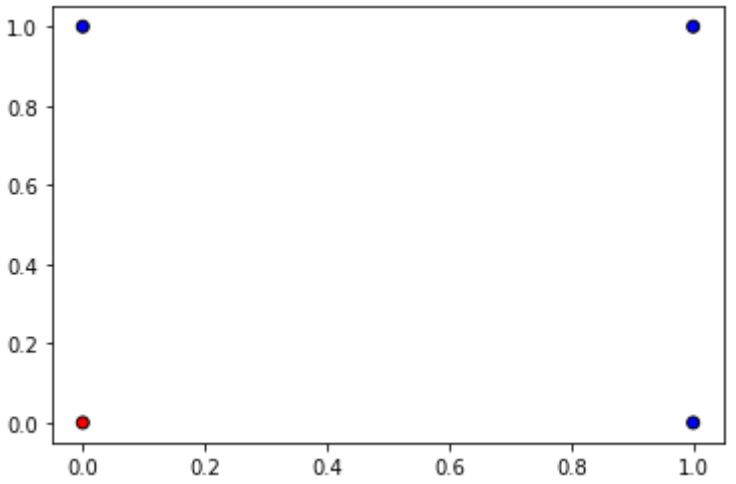
Introducimos los datos:

- En X almacenamos los distintos puntos
- En Y almacenamos en el mismo orden los valores de los puntos de X

In [0]:
```
X=np.array([[0,0],[1,0],[0,1],[1,1]])
Y=np.array([0,1,1,1])
```

Mostramos la función OR

In [0]:
```
datos=(X,Y)
muestra_dataset(datos)
```

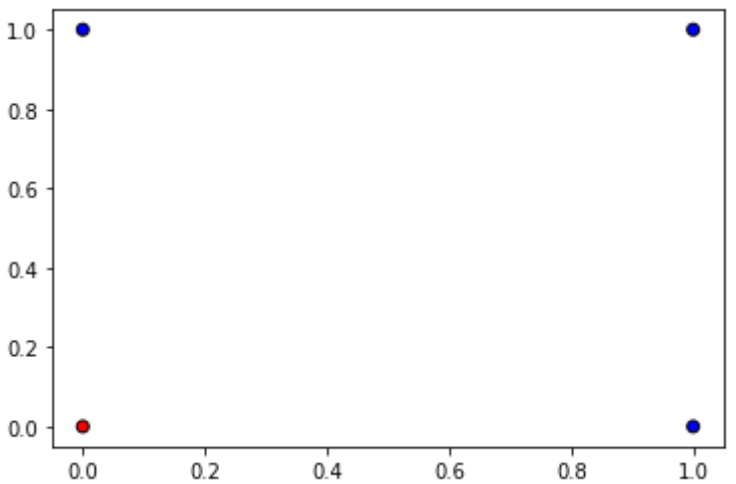


Tomamos como valores de los pesos y el sesgo valores aleatorios.

In [0]:
```
import random
random.seed(0)
w1 = random.random()
w2 = random.random()
b = random.random()
```

Mostramos la predicción que realiza el perceptrón con esos pesos y ese sesgo.

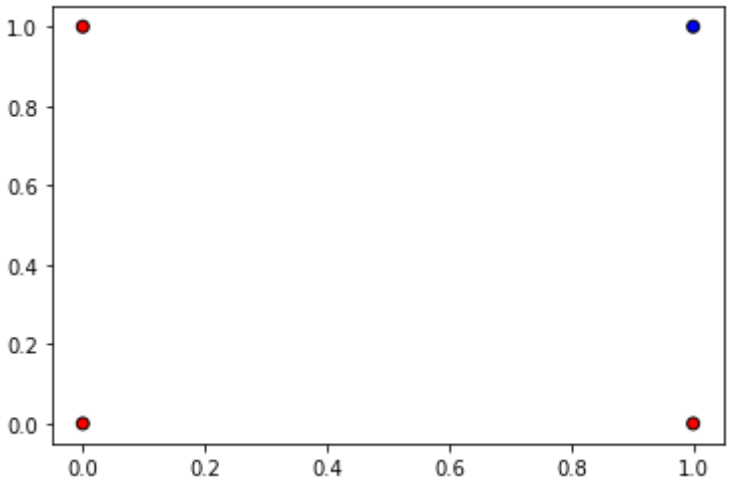
In [0]:
```
muestra_dataset_predict(datos,w1,w2,b)
```



Ha coincidido que con los valores aleatorios ya realiza una buena aproximación. Sin embargo, tomando los valores del trabajo no ocurre lo mismo.

In [0]:
```
w1,w2,b=1.5,1.5,2
```

In [0]:
```
muestra_dataset_predict(datos,w1,w2,b)
```



Definimos la regla de aprendizaje con la siguente función.

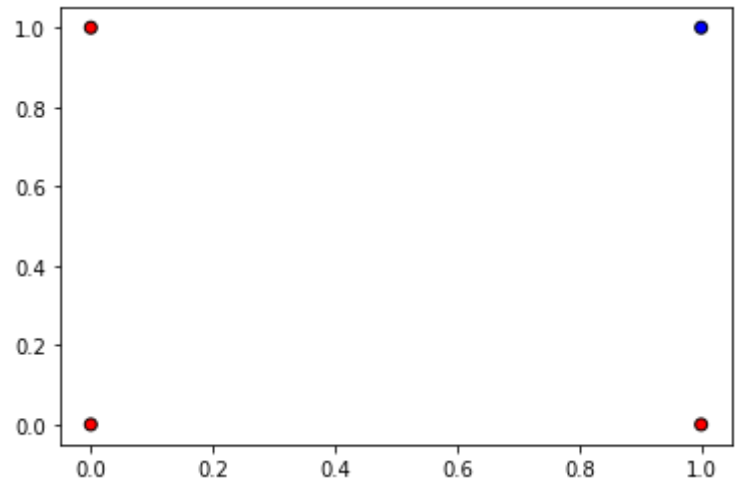
In [0]:
```
def actualizaPesos(w1,w2,b,alpha,X,y):
  e= y-perceptron(w1,w2,b)(X[0],X[1])
  w1new = w1 + alpha*X[0]*e
  w2new = w2 + alpha*X[1]*e
  bnew = b - alpha*e
  return (w1new,w2new,bnew)
```

Veamos como va aprendiendo nuestro perceptrón.

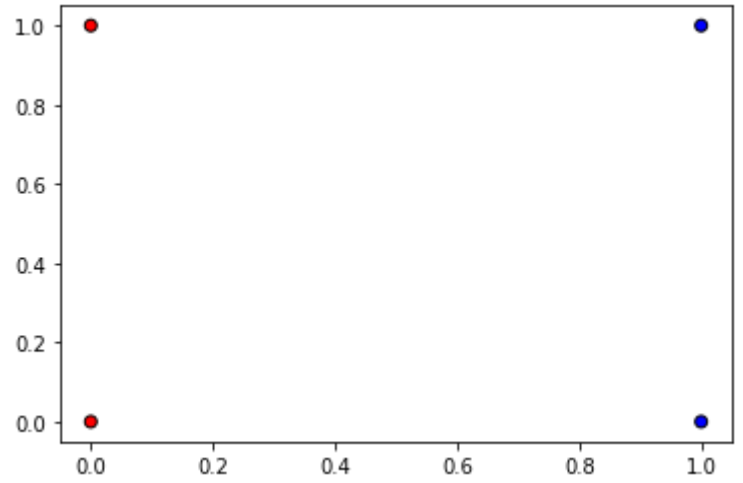
In [0]:

```
alpha=0.3
i=0
while i<4:
  w1,w2,b=actualizaPesos(w1,w2,b,alpha,X[i],Y[i])
  print("Iteración " + str(i+1))
  print(w1,w2,b)
  muestra_dataset_predict(datos,w1,w2,b)
  if Y[i]== perceptron(w1,w2,b)(X[i][0],X[i][1]): i+=1
```

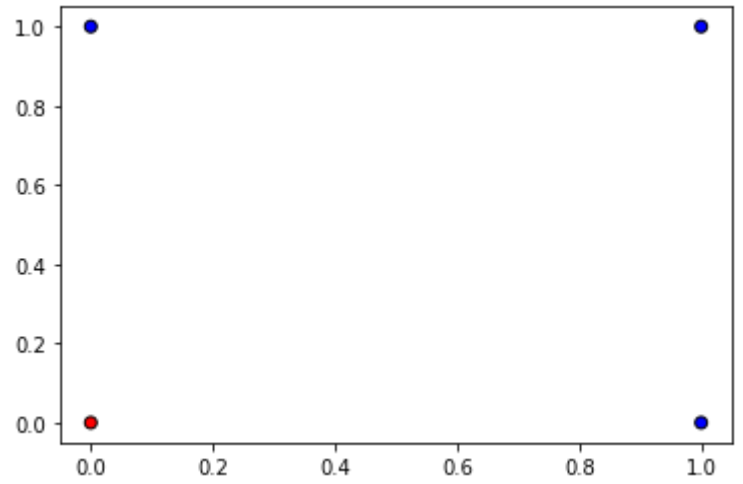
Iteración 1
1.5 1.5 2.0



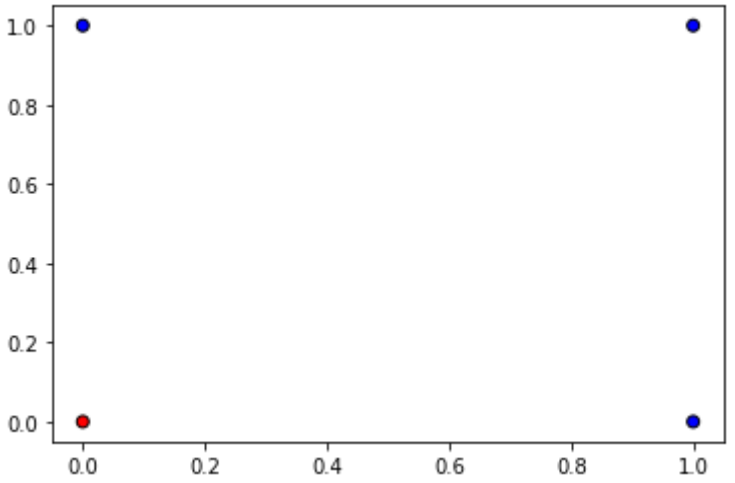
Iteración 2
1.8 1.5 1.7



Iteración 3
1.8 1.8 1.4



Iteración 4
1.8 1.8 1.4



Como vemos a partir de la tercera iteración, el perceptrón ya aproxima perfectamente todos los puntos.

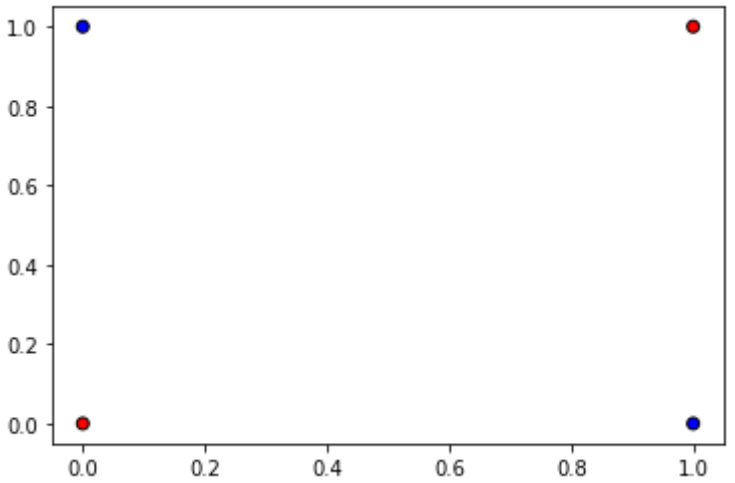
FUNCIÓN XOR

Hacemos lo mismo que antes. Almacenamos los puntos y sus valores en las variables X e Y respectivamente.

```
In [0]: X=np.array([[0,0],[1,0],[0,1],[1,1]])
        Y=np.array([0,1,1,0])
```

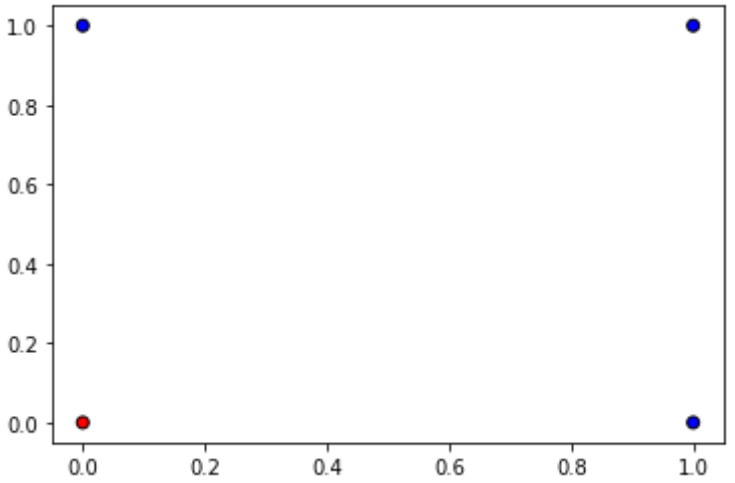
Mostramos los datos

```
In [0]: datos=(X,Y)
        muestra_dataset(datos)
```



Tomando los mismos pesos y el mismo sesgo que en el trabajo, obtenemos la siguiente predicción.

In [0]:
```
w1,w2,b=1,1,0.5
muestra_dataset_predict(datos,w1,w2,b)
```

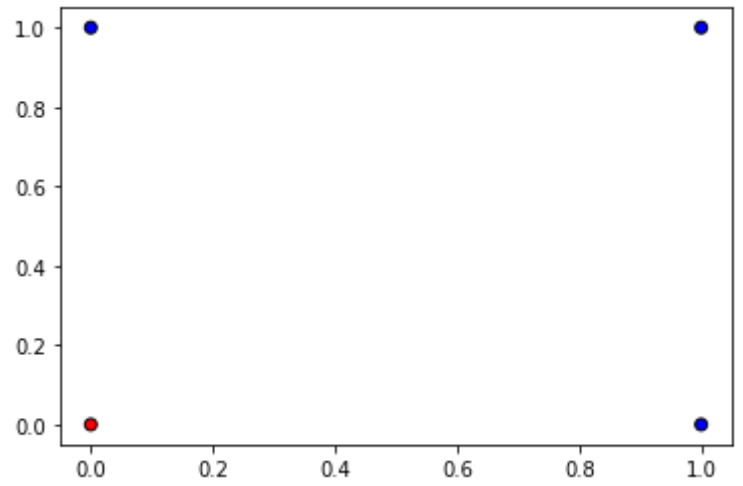


Veamos ahora los distintos pasos que realiza el algoritmo de aprendizaje

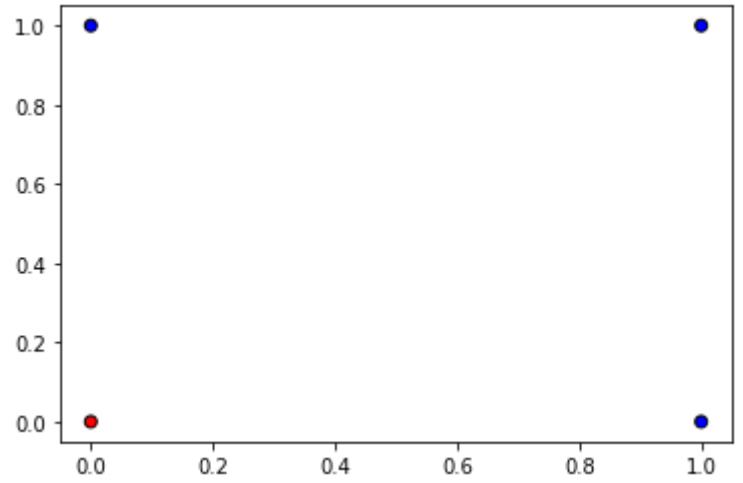
In [0]:

```
alpha=0.25
i=0
while i<10:
  print("Iteración " + str(i+1))
  print(w1,w2,b)
  w1,w2,b=actualizaPesos(w1,w2,b,alpha,X[i%4],Y[i%4])
  muestra_dataset_predict(datos,w1,w2,b)
  if Y[i%4]== perceptron(w1,w2,b)(X[i%4][0],X[i%4][1]): i+=1
```

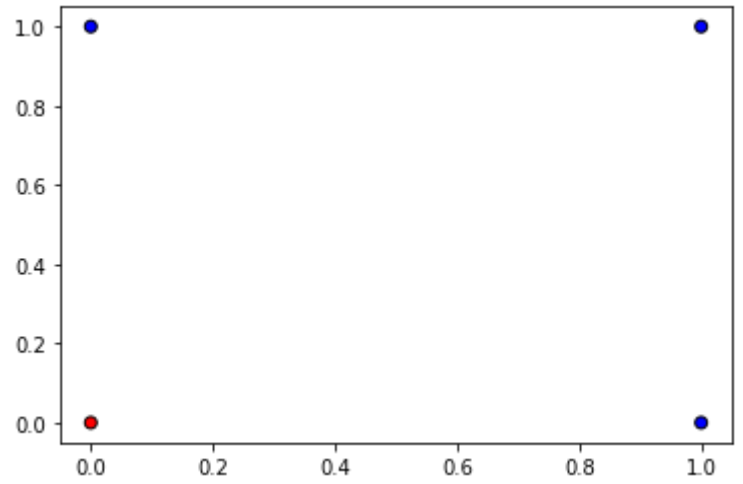
Iteración 1
1 1 0.5



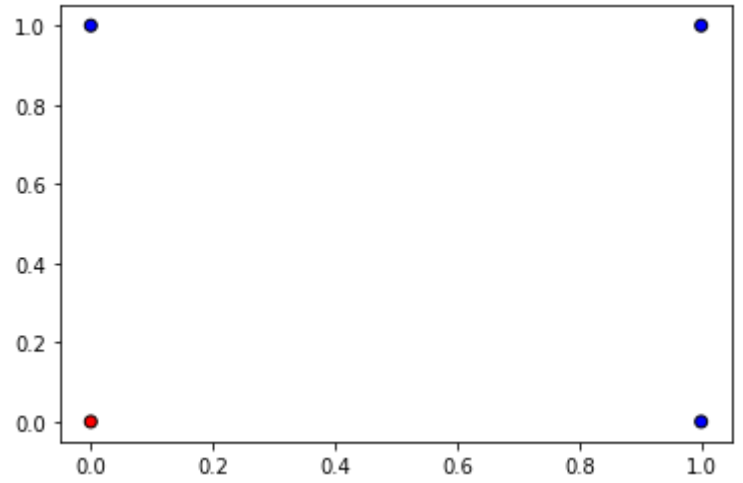
Iteración 2
1.0 1.0 0.5



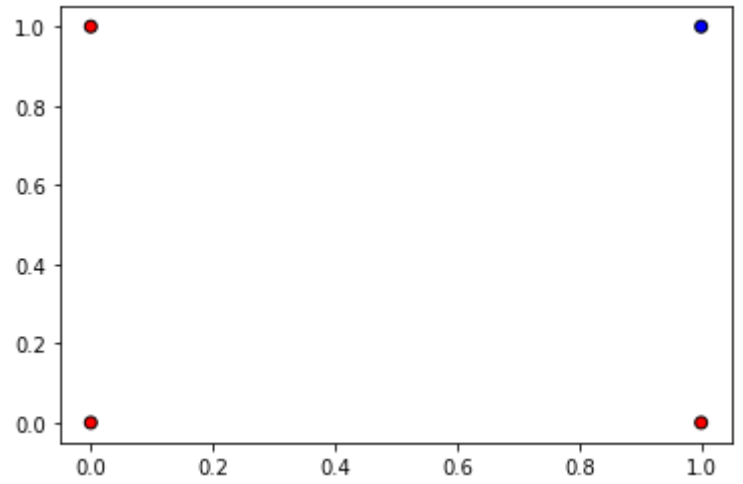
Iteración 3
1.0 1.0 0.5



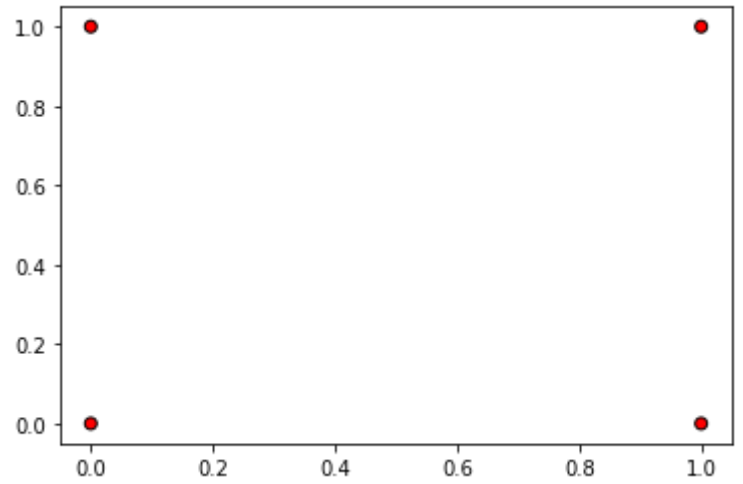
Iteración 4
1.0 1.0 0.5



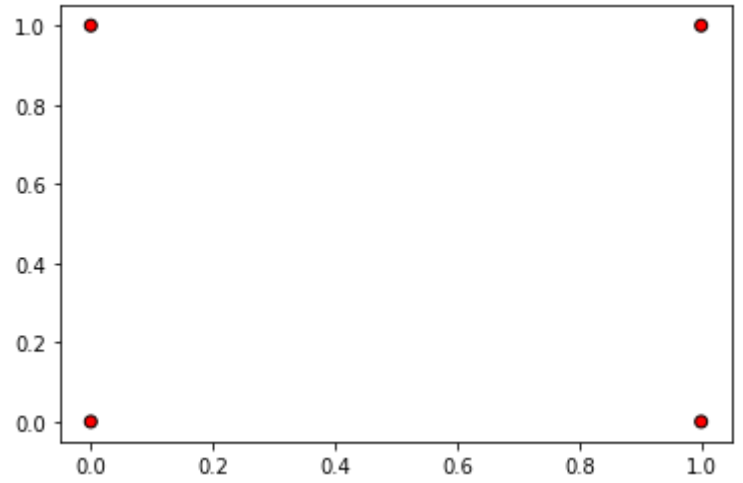
Iteración 4
0.75 0.75 0.75



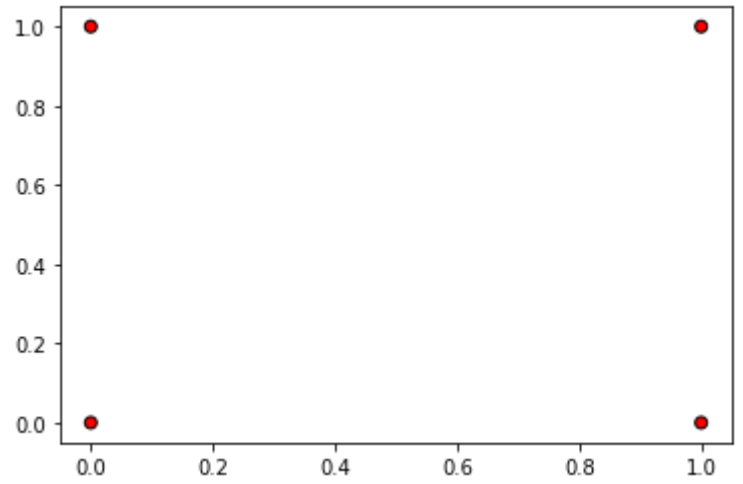
Iteración 4
0.5 0.5 1.0



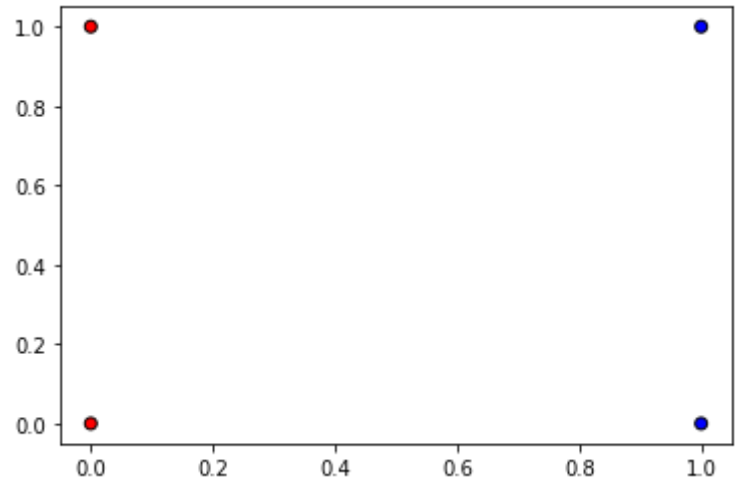
Iteración 5
0.25 0.25 1.25



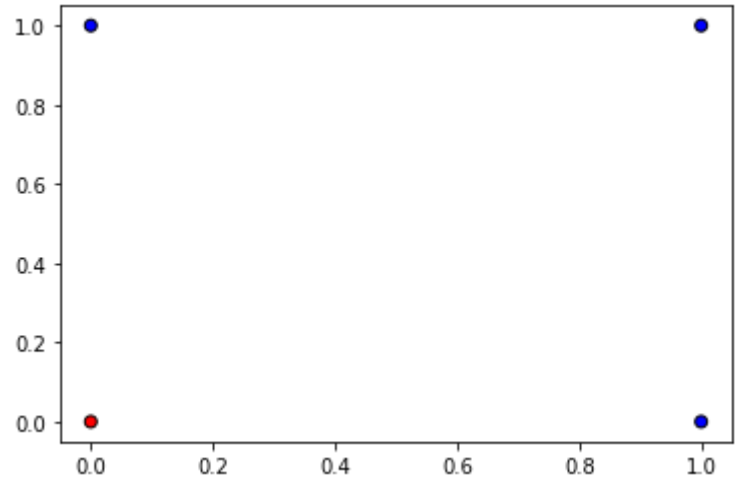
Iteración 6
0.25 0.25 1.25



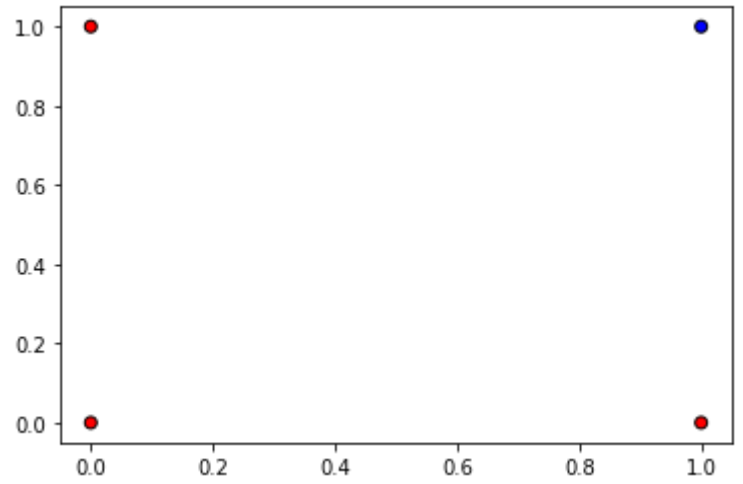
Iteración 6
0.5 0.25 1.0



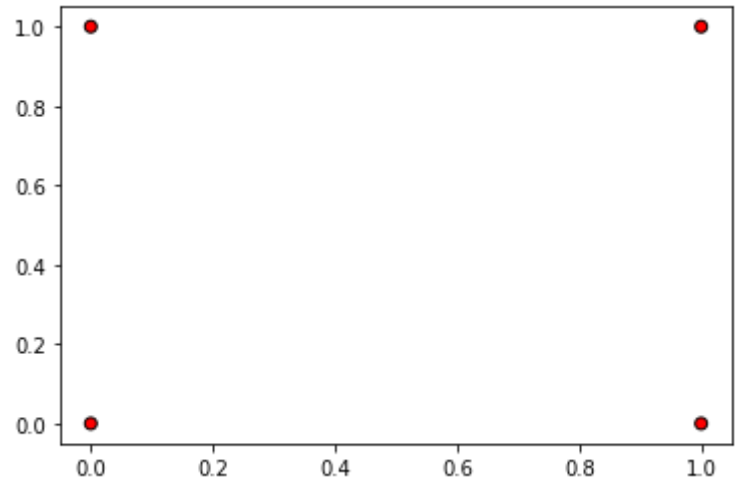
Iteración 7
0.75 0.25 0.75



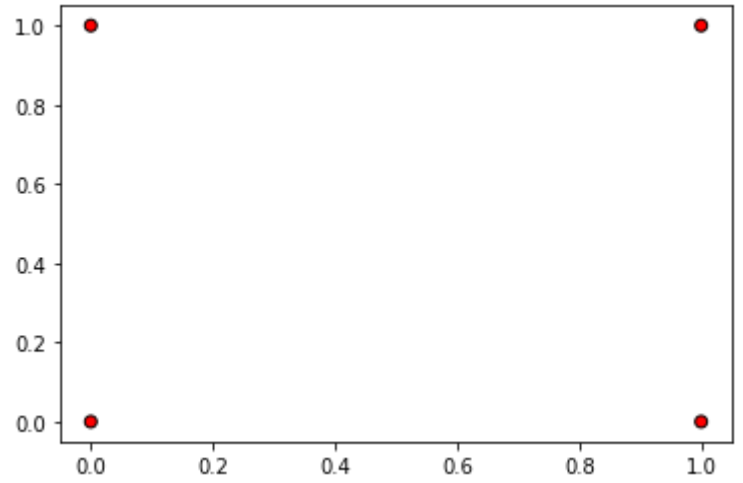
Iteración 8
0.75 0.5 0.5



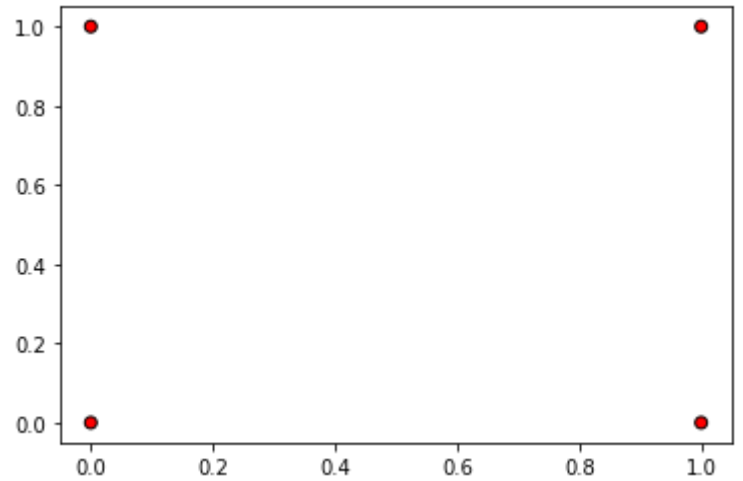
Iteración 8
0.5 0.25 0.75



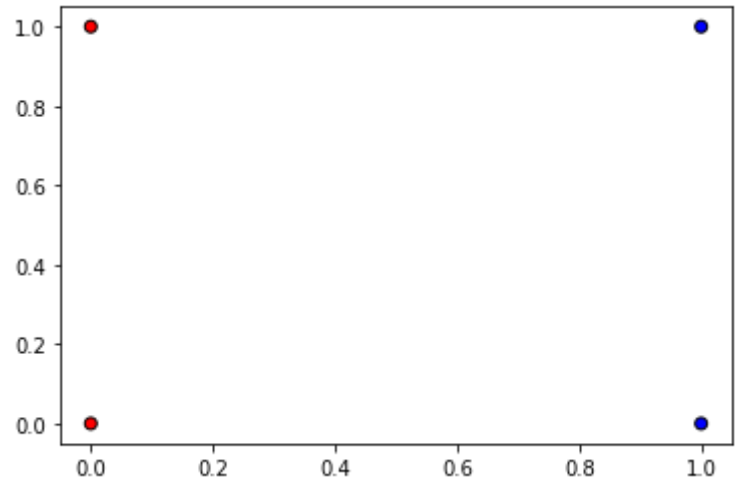
Iteración 9
0.25 0.0 1.0




```
Iteración 10
0.25 0.0 1.0
```




```
Iteración 10
0.5 0.0 0.75
```



Vemos que no es capaz de aproximar la función XOR, como hemos visto en el trabajo, esto se debe a que dicha función no es linealmente separable.

Aproximación con RNA y complejos simpliciales

En este notebook vamos a construir la RNA hacia adelante con 2 capas ocultas

In [ ]:
```
import numpy as np
import numpy.linalg
import math
```

Funciones de activación

A continuación construimos las funciones ϕ^1, ϕ^2, ϕ^3

In [ ]:
```
def phi1(W,y,b):
  return g(W.dot(y)+b,len(y))

def phi2(W,y):
  return (W.dot(y[0]),y[1])

def phi3(W,a):
  y=a[0]
  nsim=a[1]
  m=len(W)
  l= int(len(W[0])/(m+1))
  Num=0
  p=[]
  for i in range (0,l):
    Wj=W[:m,i*(m+1):(i+1)*(m+1)]
    yj=y[i*(m+1):(i+1)*(m+1)]
    z=float(Wj.dot(yj))
    Num=Num +z
  return Num/nsim
```

Función autocorrectora

In [ ]:
```
def g(v,n):
  k=int(len(v)/(n+1))
  m=0
  for j in range (0,k):
    t=False
    for i in range (0,n+1):
      if v[j*(n+1)+i]<0: t=True
    if t:
      for i in range (0,n+1):
        v[j*(n+1)+i]=0
    else: m+=1
  return v,m
```

Construcción de matrices

Construimos las matrices W_1, W_2, W_3 y el sesgo b_1

In [ ]:
```
def matrizW1(SimplicialComplexS,dimV):
  n=len(SimplicialComplexS[0])
  m=len(SimplicialComplexS[0][0])
  S=[]
  W,b=[],[]
  for i in range(0,n):
    s=[]
    for j in range(0,m):
      s.append(SimplicialComplexS[1][SimplicialComplexS[0][i][j]])
    S.append(s)
  for i in range (0,n):
    M=np.array(S[i]).transpose().reshape(dimV,m)
    A=np.concatenate((M, np.ones((1,m), dtype=np.int32)),axis=0)
    Wb= np.linalg.inv(A)
    W.append(Wb[:,:-1])
    b.append(Wb[:,m-1])
  W1=W[0]
  b1=b[0]
  for i in range (0,n-1):
    W1=np.concatenate((W1,W[i+1]),axis=0)
    b1=np.concatenate((b1,b[i+1]),axis=0)
  return W1,b1.reshape(n*m, -1)
```

In [ ]:
```
def matrizW2(SimplicialComplexS,SimplicialComplexL,mapaVertices):
  nS=len(SimplicialComplexS[0])
  mS=len(SimplicialComplexS[0][0])
  nL=len(SimplicialComplexL[0])
  mL=len(SimplicialComplexL[0][0])
  L=[]
  for i in range (0,nL):
    for j in range (0,mL):
      L.append(SimplicialComplexL[0][i][j])
  S=[]
  for i in range (0,nS):
    for j in range (0,mS):
      s=[]
      u=mapaVertices[SimplicialComplexS[0][i][j]]
      for k in range (0,len(L)):
        if u==L[k]: s.append(1)
        else: s.append(0)
      S.append(s)
  return(np.array(S).transpose())
```

In [ ]:
```
def matrizW3(SimplicialComplexL):
  n=len(SimplicialComplexL[0])
  m=len(SimplicialComplexL[0][0])
  L=[]
  for i in range (0,n):
    for j in range(0,m):
      L.append(SimplicialComplexL[1][SimplicialComplexL[0][i][j]])
  return np.array(L).transpose()
```

Construcción de la RNA

Construimos una clase: *NeuralNetwork* con ella nos será más sencillo introducir los datos y calcular las predicciones.

In [ ]:
```
class NeuralNetwork:
  #Le pasamos el Complejo simplicial S, Complejo simplicial L, mapa de v
értices y la dimensión de los vértices en S
    def __init__(self,SimplicialComplexS,SimplicialComplexL,mapaVertices
,n):
        self.weights = [matrizW1(SimplicialComplexS,n),matrizW2(Simplici
alComplexS,SimplicialComplexL,mapaVertices),matrizW3(SimplicialComplexL
)]

    def predict(self, x):
      return phi3(self.weights[2],phi2(self.weights[1],phi1(self.weights
[0][0],x,self.weights[0][1])))

    def capa1oculta(self,x): #Coordenadas baricéntricas de x respecto S
      return phi1(self.weights[0][0],x,self.weights[0][1])
    def capa2oculta(self,x):#Coordenadas 'baricéntricas' de y respecto L
      return phi2(self.weights[1],phi1(self.weights[0][0],x,self.weights
[0][1]))
    def W1(self):
      return self.weights[0][0]
    def b1(self):
      return self.weights[0][1]
    def W2(self):
      return self.weights[1]
    def W3(self):
      return self.weights[2]
```

Ejemplo

Realizamos el ejemplo del trabajo.

$$\begin{array}{rccc} f: & [-1,1]\times[-1,1]\subset\mathbb{R}^2 & \to & [-\sqrt{2},\sqrt{2}]\subset\mathbb{R} \\ & (x,y) & \to & sgn(x)sgn(y)d((x,y),(0,0)) \end{array}$$

Siendo $sgn(x)$ el signo de x y $d((x,y),(0,0))$ la distancia Euclidia del punto (x,y) al origen.

Tomamos la triangulación vista en el trabajo:

In [ ]:
```
Svertices = {'v0':[-1,1],'v1':[-1,-1],'v2':[1,-1],'v3':[1,1],'v4':[0,0]}
Lvertices = {'u1':[-math.sqrt(2)],'u2':[0],'u3':[math.sqrt(2)]}

S = [['v0','v1','v4'],['v1','v2','v4'],['v2','v3','v4'],['v3','v0','v4'
]]
L = [['u1','u2'],['u2','u3']]


mapaVertices = {'v0':'u1','v1':'u3','v2':'u1','v3':'u3','v4':'u2'}
SimplicialComplexS = (S,Svertices)
SimplicialComplexL = (L,Lvertices)
```

Tomamos también la misma función simplicial: $\varphi_c : |S| \to |L|$ con mapa de vértices $\varphi : S^{(0)} \to L^{(0)}$.

$$\varphi(v_0) = u_0,\ \varphi(v_1) = u_1,\ \varphi(v_2) = u_0,\ \varphi(v_3) = u_2,\ \varphi(v_4) = u_1$$

Ahora, siendo $x = \sum_{j=0}^{i} \lambda_j v_j$ con $\sum_{j=0}^{i} \lambda_j = 1$ la función simplicial es la siguiente:

$$\varphi_c(x) = \sum_{j=0}^{i} \lambda_j \varphi(v_j)$$

.

Con la siguiente línea de código construimos nuestra Red neuronal hacia adelante con 2 capas ocultas.

In [ ]:
```
NN=NeuralNetwork(SimplicialComplexS,SimplicialComplexL,mapaVertices,2)
```

Mostramos las matrices:

In [ ]:
```
NN.W1()
```

Out[ ]:
```
array([[-0.5,  0.5],
       [-0.5, -0.5],
       [ 1. ,  0. ],
       [-0.5, -0.5],
       [ 0.5, -0.5],
       [ 0. ,  1. ],
       [ 0.5, -0.5],
       [ 0.5,  0.5],
       [-1. ,  0. ],
       [ 0.5,  0.5],
       [-0.5,  0.5],
       [ 0. , -1. ]])
```

In [ ]:
```
NN.b1()
```

Out[ ]:
```
array([[-0.],
       [-0.],
       [ 1.],
       [-0.],
       [-0.],
       [ 1.],
       [ 0.],
       [ 0.],
       [ 1.],
       [ 0.],
       [ 0.],
       [ 1.]])
```

In [ ]:
```
NN.W2()
```

Out[ ]:
```
array([[1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0],
       [0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1],
       [0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1],
       [0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0]])
```

In [ ]:
```
NN.W3()
```

Out[ ]:
```
array([[-1.41421356,  0.        ,  0.        ,  1.41421356]])
```

Como vemos, obtenemos las mismas matrices que en el trabajo.

Procedemos a realizar alguna predicción.

Tomamos en primer lugar el punto: $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$

In [ ]:
```
x=np.array([[-1,1]]).transpose()
```

In [ ]:
```
NN.predict(x)
```

Out[ ]:
```
-1.4142135623730951
```

In [ ]:
```
x=np.array([[1,1]]).transpose()
```

In [ ]:
```
NN.predict(x)
```

Out[ ]:
```
1.4142135623730951
```

Realizamos más predicciones, tomamos ahora $\begin{bmatrix} -1 \\ -1 \end{bmatrix}$

In [ ]:
```
x=np.array([[-1,-1]]).transpose()
```

In [ ]:
```
NN.predict(x)
```

Out[ ]:
```
1.4142135623730951
```

Tomamos ahora un punto que no sea vértice: $\begin{bmatrix} -1/2 \\ 0 \end{bmatrix}$

Definimos la función f y calculamos el valor de este punto

In [ ]:
```
def f(x,y):
  if x!=0 and y!=0:
    return (x/abs(x))*(y/abs(y))*math.sqrt(x**2+y**2)
  if x==0:
    return (y/abs(y))*math.sqrt(x**2+y**2)
  if y==0:
    return (x/abs(x))*math.sqrt(x**2+y**2)
```

In [ ]:
```
f(-1/2,0)
```

Out[ ]:
```
-0.5
```

In [ ]:
```
x=np.array([[-1/2,0]]).transpose()
```

In [ ]:
```
NN.predict(x)
```

Out[ ]:
```
1.4142135623730951
```

Veamos a continuación la representación de distintos puntos y con su valor calculado con la función y con la RNA

Tomamos 50 puntos aleatorios en $[-1, 1] \times [-1, 1]$ y calculamos su valor.

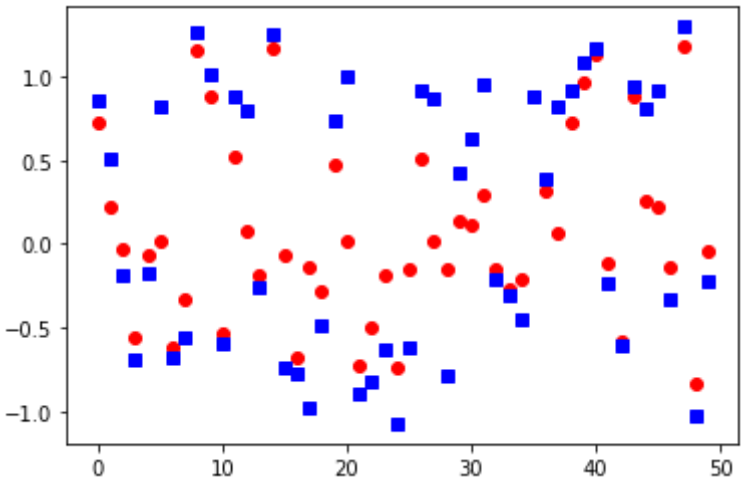
In [ ]:
```
import random
random.seed(0)
X,Y=[],[]
for i in range (0,50):
  l=[]
  l.append(random.uniform(-1,1))
  l.append(random.uniform(-1,1))
  X.append(l)
  Y.append(f(l[0],l[1]))
X=np.array(X)
datos=(X,Y)
```

In [ ]:
```
Y1=[NN.predict(np.array([[x1,x2]]).transpose()) for (x1,x2) in X]
```

A continuación graficamos los valores de los 50 puntos.

- En el **eje x** pintamos los 50 puntos.
- En el **eje y** los valores que obtenemos para ese punto:
  - En *azul* los valores exactos.
  - En *rojo* los valores aproximados.

In [ ]:
```
plt.plot(Y1,'ro',Y,'bs')
plt.show()
```



Calculamos el error.

In [ ]:
```
E=[abs(Y1[i]-Y[i]) for i in range (0,50)]
```

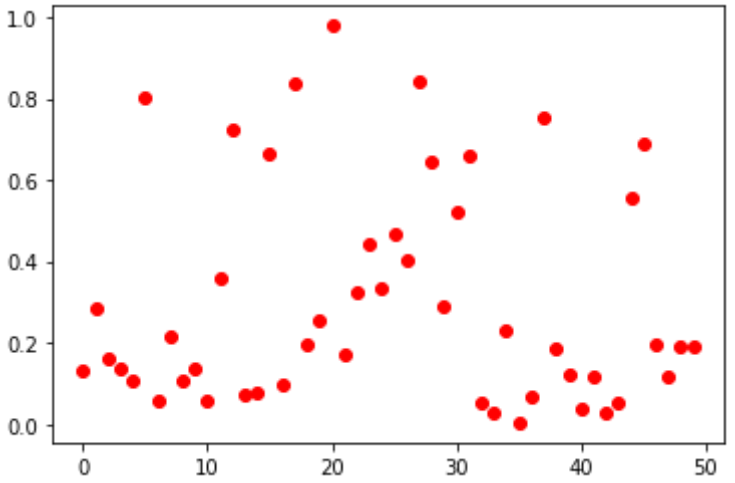
Error máximo:

In [ ]:
```
max(E)
```

Out[ ]:
```
0.9796324978247023
```

Graficamos el error:

In [ ]:
```
plt.plot(E,'ro')
plt.show()
```



El error máximo que cometemos es próximo a la unidad