Reconocimiento asociativo de estímulos en redes neuronales

Trabajo de Fin de Grado
Curso 2020/2021

UNIVERSIDAD COMPLUTENSE
MADRID

Facultad de Ciencias Matemáticas
Doble Grado en Ingeniería Informática y Matemáticas

Autor:
Pablo Sanz Sanz

Director:
Valeriy Makarov Slizneva

Madrid, 6 de julio de 2021

Resumen

La Inteligencia Artificial es un campo que nace en la década de 1950 y que se ha ido desarrollando lentamente hasta nuestros días. Sin embargo, con el rápido aumento en la capacidad de cómputo que hemos experimentado estos últimos años, esta área está cobrando un mayor impulso y está creciendo a un ritmo acelerado. Una de las herramientas utilizadas en Aprendizaje Automático, que forma parte de la Inteligencia Artificial, son las redes neuronales artificiales, que nacen pretendiendo imitar el funcionamiento del cerebro. En este trabajo, vamos a estudiar varios tipos de redes neuronales artificiales, desde el tipo de neurona más simple, el perceptrón, hasta redes complejas, como las convolucionales. Además, en el espíritu de imitar el funcionamiento del cerebro, vamos a estudiar un modelo matemático de las llamadas neuronas conceptuales, capaces de activarse ante determinados conceptos. Al contrario que los modelos típicos de Aprendizaje Automático, que funcionan mejor con pocas dimensiones, estas neuronas se aprovechan de la alta dimensionalidad de la información para ser capaces de separar conceptos entre sí.

Palabras clave

Inteligencia Artificial, aprendizaje automático, perceptrón. red neuronal, retropropagación, red convolucional, neurona conceptual.

Abstract

Artificial Intelligence is a branch of science born in the 1950s, which has been developed relatively slowly to the present day. However, with the rapid increase in computational power we are experimenting lately, this area is gaining pace and it is growing rapidly. One of the tools used in Machine Learning –a part of Artificial Intelligence– is artificial neural networks. These networks were born as an imitation of the human brain. In this project, we are going to study several types of artificial neural networks. We will start from the simplest neuron, the perceptron, and end with complex ones, like convolutional neural networks. Additionally, in the spirit of imitating brain operation, we will study a mathematical model of the so-called conceptual neurons. These neurons can trigger in the presence of certain concepts. In contrast with typical Machine Learning models, which behave better with few dimensions, these neurons take advantage of the high dimensionality of information to separate better different concepts.

Keywords

Artificial Intelligence, machine learning, perceptron, neural network, backpropagation, convolutional network, conceptual neuron.

Índice general

Capítulo 1

Introducción

El término Inteligencia Artificial se comenzó a utilizar en los años 50 del siglo XX gracias a McCarthy, que lo usó para dar título a una importante conferencia [1]. En 1950, el famoso matemático británico, Alan Turing, se preguntaba en su artículo *Computing machinery and intelligence* si "las máquinas son capaces de pensar" [2]. Esto lo plasmó en su Juego de la imitación, la llamada prueba de Turing. El propósito de dicha prueba es lograr que una máquina pueda ser capaz de hacerse pasar por un humano en una interacción textual de pregunta-respuesta con ella. La prueba de Turing fue superada por primera vez en 2014, cuando el programa ruso *Eugene* hizo creer al 33 % de los jueces que se trataba de un niño ucraniano de 13 años [3].

Sin embargo, el objetivo de la Inteligencia Artificial es más general que el de alcanzar la inteligencia humana. La Inteligencia Artificial trata de resolver problemas habitualmente difíciles para los humanos tomando decisiones por sí misma. Por ejemplo, existen sistemas recomendadores ampliamente utilizados en tiendas *online*; la rama de visión artificial sirve para tareas como la conducción autónoma, que últimamente está cobrando gran importancia gracias a modelos como YOLO [4]; y el aprendizaje por refuerzo se ha utilizado con gran éxito para lograr programas capaces de jugar a diferentes juegos de forma sobrehumana, como el reciente algoritmo *AlphaZero* [5]. Además, con la mejora que ha experimentado la Informática en los últimos años, especialmente en la velocidad de cómputo, cada vez tenemos sistemas inteligentes más potentes.

Uno de los elementos que tienen en común muchos de estos sistemas son las redes neuronales artificiales. Originalmente, estas redes pretendían, de alguna manera, imitar el funcionamiento de sus homólogas biológicas para resolver tareas complejas. Sin embargo, finalmente se han quedado como un aproximador universal de funciones. Como tal, en principio serían capaces de ajustarse a cualquier función que pudieran representar [6]. Por lo tanto, podemos utilizar una red neuronal para predecir el precio de una vivienda a partir de una serie de datos de entrada; pero también pueden ser utilizadas para reconocimiento de objetos en imágenes o para muchas otras tareas.

Aquí vamos a estudiar redes neuronales desde dos enfoques distintos: usando aprendizaje supervisado, el clásico y más utilizado para este tipo de problemas, y con aprendizaje no supervisado, que veremos en el contexto de neuronas conceptuales. Lo veremos más en detalle más adelante, pero la diferencia fundamental entre ambos enfoques se encuentra en el objetivo que se busque. El aprendizaje supervisado se utiliza para generalizar desde una serie de ejemplos etiquetados a nuevos ejemplos de los que se desconoce su valor asociado, mientras que el aprendizaje no supervisado trata de encontrar una separación en los datos para ayudar en su clasificación.

El objetivo final de este trabajo es estudiar un modelo artificial de un tipo de neuronas biológicas denominadas conceptuales. La existencia de dichas neuronas aún está en cuestión, pero cada vez se tiene mayor certeza de que son una realidad. Las neuronas conceptuales estarían asociadas a un concepto cada una, en contra de la creencia anterior de que los conceptos se formaban por la interacción de millones de neuronas, y se dispararían ante la presencia de un estímulo relacionado con dicho concepto. Por ejemplo, un artículo publicado en 2005 demostró que ciertas neuronas conceptuales se activan cuando el sujeto observa una foto de la actriz Jennifer Aniston o escucha su nombre, pero no lo hacen ante fotos de otras personas [7].

Sin embargo, antes de llegar a ello, primero debemos estudiar los conceptos básicos relacionados con redes neuronales. En concreto, vamos a comenzar con el tipo de neurona más simple, el perceptrón, y cómo se unen para formar redes neuronales, denominadas densas o completamente conectadas. Estas redes tienen una serie de pesos sinápticos que son los que determinan la función que aproxima. Habitualmente se tienen cientos de miles de pesos sinápticos, e incluso millones, y calcularlos a mano sería una tarea imposible. Por ello, se han desarrollado algoritmos de optimización capaces de automatizar esta tarea, llamados algoritmos de entrenamiento. En particular, primero vamos a estudiar cómo entrenar una sola neurona, para después analizar en detalle el algoritmo de retropropagación (*backpropagation*, en inglés), utilizado en redes neuronales generales. Para comprobar la teoría estudiada vamos a realizar una serie de experimentos de clasificación de imágenes.

A continuación, vamos a dedicar un capítulo a explicar las redes convolucionales, un tipo algo distinto de redes neuronales que funcionan mejor para problemas con imágenes, como los de nuestros experimentos. Por lo tanto, vamos a repetirlos con el objetivo de obtener un resultado mejor que el anterior.

Finalmente, vamos a poder estudiar las neuronas conceptuales y el porqué de su funcionamiento. Vamos a ver también un ejemplo de aplicación de estas neuronas al caso de detección de un determinado individuo, dentro de una clase más amplia. En concreto, vamos a tratar de diferenciar un gato particular de entre todos los demás gatos. Para ello, necesitaremos usar todo lo estudiado hasta el momento: redes densas, convolucionales y neuronas selectivas. Por último, vamos a analizar la dinámica de aprendizaje de estas neuronas utilizando la regla de Oja, que permite entrenarlas automáticamente, de una forma más parecida a lo que ocurre en el cerebro.

Capítulo 2

Redes neuronales prealimentadas

Las redes neuronales artificiales son modelos computacionales que tratan de imitar los procesos de toma de decisiones de los sistemas neuronales biológicos, como el que tiene el ser humano o cualquier animal. Las redes neuronales biológicas (Figura 2.1) constan de multitud de células nerviosas, llamadas neuronas. Por ejemplo, en el cerebro humano existen unas $8{,}6 \times 10^{10}$ [8]. Las neuronas están conectadas entre ellas a través de las dendritas (entradas) y del axón (salida). La transmisión de la información se lleva a cabo mediante impulsos nerviosos, llamados potenciales de acción, que comienzan en una neurona emisora y pasan a través de su axón hasta las dendritas de la neurona receptora. Pero estas conexiones no son todas igual de relevantes: hay algunas más prioritarias que otras y, de hecho, algunas son excitatorias y otras inhibitorias. Las sinapsis excitatorias aumentan el potencial eléctrico de la neurona y, de esta forma, la hacen más proclive a generar un potencial de acción en la salida. El efecto de las sinapsis inhibitorias es el contrario. Todo esto lo tratarán de imitar las redes neuronales artificiales.

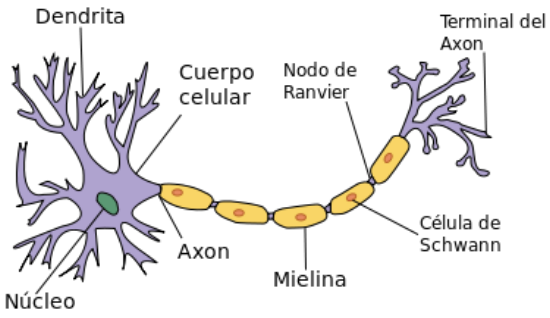
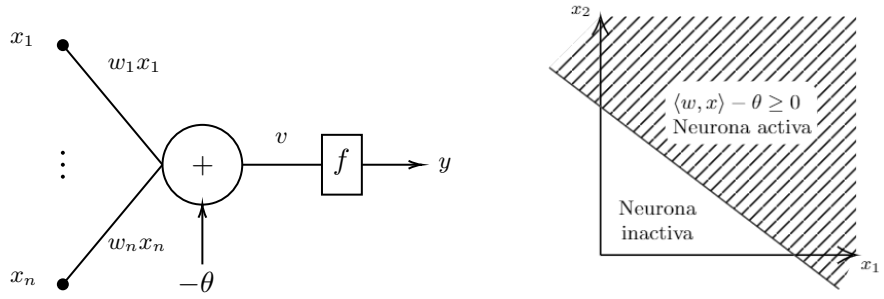


Figura 2.1: Esquema de una neurona biológica [9].

2.1. El perceptrón

Vamos a comenzar definiendo el concepto básico de neurona artificial. El primer modelo y de los más simples que existen es el perceptrón, introducido por el psicólogo Frank Rosenblatt en 1958 [10] en base a los trabajos realizados por McCulloch y Pitts en 1943 [11]. Pese a su simpleza, es el más utilizado para construir redes neuronales artificiales más complejas hoy en día.

Definición 2.1 (Perceptrón)**.** Un perceptrón (Figura 2.2a) se puede considerar como una



(a) (b)

Figura 2.2: a) Diagrama de un perceptrón. b) Ejemplo de funcionamiento aplicado en $\mathbb{R}^2$, usando la función escalón como función de activación y un umbral de activación θ. La neurona será capaz de reconocer la región del plano marcada, es decir, se activará cuando reciba estímulos de dicha zona y permanecerá inactiva en la región complementaria.

aplicación de la forma

$$\begin{aligned} F : \mathbb{R}^n \times \mathbb{R}^n &\to \mathbb{R} \\ (x; w) &\mapsto y = F(x; w) = f(\langle w, x\rangle), \end{aligned}$$

donde $f : \mathbb{R} \to \mathbb{R}$ se denomina función de activación, $x \in \mathbb{R}^n$ es el vector de entrada (datos a procesar), $w \in \mathbb{R}^n$ es el denominado vector de pesos sinápticos (parámetros de la neurona) e $y \in \mathbb{R}$ es la salida de la neurona. Con $\langle \cdot, \cdot \rangle$ denotamos el producto escalar de dos vectores.

De esta forma, la neurona recibe un estímulo $x \in \mathbb{R}^n$, su núcleo computa el potencial eléctrico de la membrana $v = \langle w, x\rangle$ y produce una salida $y = f(v)$.

La función de activación f permite que la salida de la neurona se encuentre entre unos determinados límites. Existen numerosas funciones de activación y entre las más típicas podemos encontrar las siguientes.

- La función escalón (se utilizaba originalmente en [10]):

$$H(u) = \begin{cases} 0 & u < 0 \\ 1 & u \geq 0. \end{cases} \tag{2.1}$$

  Esta función se puede utilizar en problemas de clasificación binaria.

- La función rectificada lineal (*Rectified Linear Unit*)

$$\mathrm{ReLU}(u) = \max\{0, u\}, \tag{2.2}$$

  que se utiliza habitualmente en neuronas de capas intermedias por su simplicidad.

- La función sigmoide

$$\mathrm{S}(u) = \frac{1}{1 + e^{-u}}, \tag{2.3}$$

  que se utiliza a menudo como salida de la red para obtener la probabilidad de pertenecer a una clase, ya que devuelve valores en el intervalo $(0, 1)$.

El vector de pesos sinápticos w permite que la neurona produzca la salida esperada al ser estimulada, y se obtiene mediante un proceso de aprendizaje (entrenamiento), que se explicará en el Capítulo 3.

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Cuadro 2.1: Tabla de verdad de la función xor. Esta función es cierta si y sólo si es cierta una de sus entradas, pero no ambas a la vez.

Típicamente, además del vector de pesos sinápticos, el perceptrón suele tener también un parámetro de sesgo θ (Figura 2.2), que en nuestro contexto vamos a denominar umbral de activación. El umbral de activación se puede añadir a la función del perceptrón de la forma

$$y = F(x; w, \theta) = f(\langle w, x\rangle - \theta) = f(\langle \hat{w}, \hat{x}\rangle) = \hat{F}(\hat{x}; \hat{w}),$$

donde $\hat{x} = (x, 1) \in \mathbb{R}^{n+1}$, $\hat{w} = (w, -\theta) \in \mathbb{R}^{n+1}$ y $\hat{F} : \mathbb{R}^{n+1} \times \mathbb{R}^{n+1} \to \mathbb{R}$. En nuestro contexto, θ corresponde al potencial mínimo necesario para que la neurona se active, pues vamos a usar típicamente la función ReLU como función de activación. Así, $F(x; w, \theta) = \max\{0, \langle w, x\rangle - \theta\}$, por lo que decimos que la neurona está inactiva ($y = 0$) si su potencial $\langle w, x\rangle$ no supera el umbral θ y activa en caso contrario ($y > 0$).

Por tanto, la idea original del perceptrón consistía en dividir el espacio de entradas $\mathbb{R}^n$ en dos semiespacios $H_1 = \{x \in \mathbb{R}^n : \langle w, x\rangle - \theta \leq 0\}$ y $H_2 = \{x \in \mathbb{R}^n : \langle w, x\rangle - \theta > 0\}$, de forma que los estímulos $x \in H_1$ se clasificarán como de la primera clase y aquellos $x \in H_2$ como de la segunda (Figura 2.2b). La recta de separación de ambos semiespacios depende, por tanto, de los pesos sinápticos w y del umbral θ.

La capacidad de cálculo de una sola neurona es bastante limitada, y esto lo podemos comprobar con uno de los ejemplos más famosos: la función booleana xor ($\oplus$), definida por la tabla de verdad del Cuadro 2.1. Consideremos una neurona con dos entradas $F : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$. Debemos construir un vector de pesos $w \in \mathbb{R}^2$ que haga que $F((0,0); w) = 0$, $F((0,1); w) = 1$, $F((1,0); w) = 1$ y $F((1,1); w) = 0$. Sin embargo, no podemos dividir el plano (x_1, x_2) mediante una recta $\langle w, x\rangle = \theta$ de forma que los puntos queden separados correctamente (Figura 2.3). Esto lo formalizamos en la siguiente proposición.

Proposición 2.2. *La función xor no puede ser modelizada por un perceptrón.*

Demostración. Como trabajamos con clasificación binaria, vamos a utilizar la función escalón H definida en (2.1) como función de activación. Así, la función que aplica el perceptrón es $F(x; w) = H(\langle w, x\rangle - \theta)$ para cierto $\theta \in \mathbb{R}$. La respuesta del perceptrón a la entrada $x = (0,0)$ debe ser $y = 0$, por lo que $\langle w, (0,0)\rangle - \theta = -\theta < 0$. Debemos obligar a que las entradas $(0,1)$ y $(1,0)$ produzcan $y = 1$. Por tanto, $\langle w, (0,1)\rangle, \langle w, (1,0)\rangle \geq \theta$, por lo que si $w = (w_1, w_2)$, entonces $w_1, w_2 \geq \theta$. Finalmente, la entrada $(1,1)$ debería producir $y = 0$. Pero se tiene $\langle w, (1,1)\rangle - \theta = w_1 + w_2 - \theta \geq 2\theta - \theta = \theta > 0$, por lo que $y = 1$ y llegamos a una contradicción. □

Se hace necesario, por tanto, disponer de más neuronas para poder dividir el espacio en regiones más complejas.

2.2. Redes neuronales con capas densas

Según el teorema sobre el aprendizaje del perceptrón formulado por Rosenblatt en 1961 [6], una red neuronal de perceptrones es capaz de resolver cualquier problema que se pueda

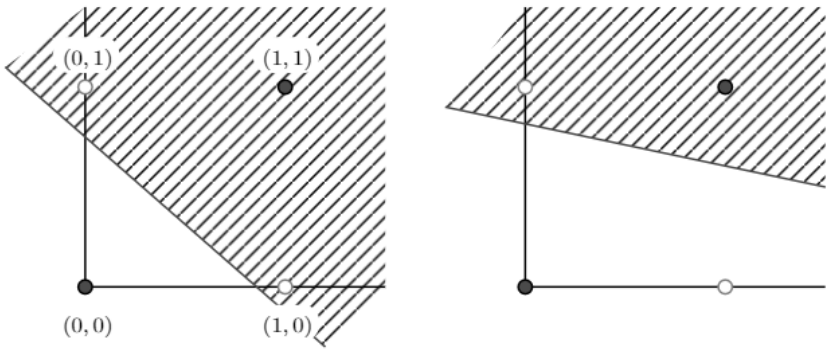


Figura 2.3: Ninguna recta puede separar los puntos (entradas del perceptrón) $(0,0)$ y $(1,1)$ de $(0,1)$ y $(1,0)$, por lo que el perceptrón no puede implementar la función xor.

representar con esta. Estas redes pueden estar formadas por una o más capas de neuronas acopladas entre sí de forma parecida a las redes biológicas. Comencemos definiendo el concepto de redes neuronales.

Definición 2.3 (Capa de neuronas)**.** Una capa de m perceptrones es una aplicación F de la forma

$$\begin{aligned} F : \mathbb{R}^n \times \mathcal{M}_{m\times n}(\mathbb{R}) &\to \mathbb{R}^m \\ (x; W) &\mapsto y = F(x; W) = \mathbf{f}(Wx), \end{aligned}$$

donde $x \in \mathbb{R}^n$ es el vector de datos de entrada a la capa, $W \in \mathcal{M}_{m\times n}(\mathbb{R})$ es la matriz de pesos sinápticos, cuyas filas son los vectores de pesos de cada una de las neuronas, y $\mathbf{f}(u_1, \ldots, u_m) = (f_1(u_1), \ldots, f_m(u_m))^T$, para las funciones de activación $f_i : \mathbb{R} \to \mathbb{R}$. A menudo tendremos $f_i \equiv f$ para todo $i \in \{1, \ldots, m\}$.

De nuevo, podemos considerar, además, los sesgos o umbrales de activación $\theta = (\theta_1, \ldots, \theta_m)^T$ de cada neurona. Por otra parte, debemos notar que la dimensión de los datos de entrada n es, en general, distinta de la dimensión de salida m.

Utilizando una capa de neuronas en lugar de una sola podemos obtener resultados de procesamiento de información más complejos. Por ejemplo, podemos utilizar una capa de m neuronas para un problema de clasificación difusa en m conjuntos, entendiendo cada una de las salidas y_j, $j \in \{1, \ldots, m\}$, como la probabilidad de que el estímulo pertenezca a la clase j. Para ello, se suele utilizar la función de activación llamada softmax, o función exponencial normalizada:

$$y_j = \sigma_j(u) = \frac{e^{u_j}}{\sum_{i=1}^m e^{u_i}}, \quad j \in \{1, \ldots, m\}.$$

Ahora podemos definir un perceptrón multicapa de la siguiente forma.

Definición 2.4 (Perceptrón multicapa completamente conectado)**.** Un perceptrón multicapa es una red neuronal formada por la composición de k capas de n_i neuronas cada una, de la forma $F^{(i)} : \mathbb{R}^{n_{i-1}} \times \mathcal{M}_{n_i\times n_{i-1}}(\mathbb{R}) \to \mathbb{R}^{n_i}$ con $i \in \{1, \ldots, k\}$. Es decir, es la aplicación $F : \mathbb{R}^{n_0} \times \mathcal{M}_{n_1\times n_0}(\mathbb{R}) \times \cdots \times \mathcal{M}_{n_k\times n_{k-1}}(\mathbb{R}) \to \mathbb{R}^{n_k}$ definida como

$$F(x; W^{(1)}, \ldots, W^{(k)}) = F^{(k)}(\cdots F^{(2)}(F^{(1)}(x; W^{(1)}); W^{(2)}) \cdots ; W^{(k)}). \tag{2.4}$$

La primera capa se denomina capa de entrada, la última capa de salida y las intermedias son las capas ocultas.

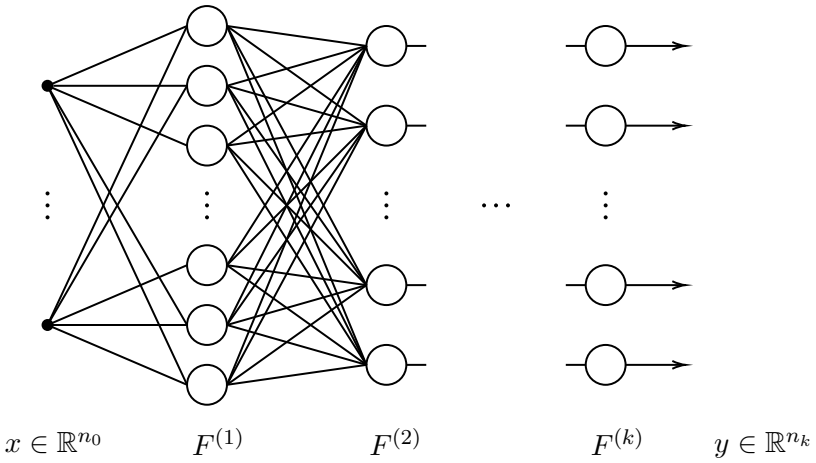


Figura 2.4: Esquema de una red neuronal multicapa. La red recibe el vector de entrada x (a la izquierda), que se ha representado componente a componente. Este vector se envía a cada neurona de la primera capa. Estas computan la función F_1 y generan un vector en $\mathbb{R}^{n_1}$ a la salida de la primera capa, que se envía a la segunda, y así sucesivamente. Finalmente, se obtiene la salida y, representada de nuevo componente a componente.

Además, las redes neuronales se clasifican en dos tipos.

Definición 2.5 (Redes profundas)**.** Supongamos que una red neuronal de k capas ocultas aplica una función no lineal F. Entonces la red se denomina profunda (*deep neural network*) si $k > 1$ y poco profunda (*shallow neural network*) cuando $k = 1$.

Gráficamente, una red neuronal se representa como en la Figura 2.4. Estamos suponiendo que las capas sucesivas están completamente conectadas entre sí (también se les denomina capas densas), lo que significa que cada una de las salidas de la capa i-ésima está conectada con todas de las neuronas de la capa $i + 1$. Finalmente, podemos añadir que existen otros tipos de capas más complejas que no tienen por qué seguir este esquema.

Capítulo 3

Aprendizaje supervisado

Hasta ahora habíamos considerado los vectores de pesos sinápticos como parámetros fijos de la red neuronal. Sin embargo, como comentamos al principio, estos parámetros se pueden ajustar a un problema concreto a través de un proceso llamado entrenamiento o aprendizaje. Para ello, debemos contar con un conjunto de ejemplos

$$\mathcal{X} = \{(x_1, y_1), \ldots, (x_N, y_N)\}, \tag{3.1}$$

formado por pares en los que $x_i \in \mathbb{R}^n$ es una observación (dato de entrada, por ejemplo, una imagen) e $y_i \in Y$ es el valor correspondiente observado en la salida (por ejemplo, la clase a la que pertenece la imagen i-ésima). En problemas de clasificación es habitual que $Y = \mathbb{Z}_\ell$, siendo ℓ el número de clases existentes, y en problemas de regresión a menudo $Y = \mathbb{R}$.

Usando el conjunto de aprendizaje $\mathcal{X}$ podemos entrenar la red neuronal para que adecúe sus pesos para conseguir una salida lo más parecida posible a la esperada en Y. Vamos a explicar primero la forma de entrenar una sola neurona y después explicaremos el algoritmo de retropropagación (*backpropagation*) utilizado en redes multicapa.

3.1. Entrenamiento de una neurona

Por simplicidad, supongamos que $Y = \mathbb{R}$. Para entrenar un único perceptrón es suficiente con calcular su salida $\hat{y}_i = F(x_i; w) \in \mathbb{R}$ obtenida para cada vector de entrada $x_i \in \mathbb{R}^n$ y compararla con la salida deseada $y_i \in \mathbb{R}$, con $(x_i, y_i) \in \mathcal{X}$, utilizando algún tipo de función de error L. Una función de error típica es el error cuadrático medio

$$L(w; \mathcal{X}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, \tag{3.2}$$

donde $N = |\mathcal{X}|$ es la cantidad de datos de entrenamiento. En adelante, por simplicidad se utilizará L sin especificar sus argumentos, siempre que no haya ambigüedad.

El objetivo consiste en minimizar la función de error en w, ya que $\hat{y}_i$ depende de los pesos sinápticos. Para ello, existen varios algoritmos, como por ejemplo el de descenso de gradiente, que requiere que la función de activación f sea derivable. Este algoritmo modifica los pesos en la dirección contraria al vector gradiente de la función L, pues este apunta hacia la dirección de máximo aumento. Dados los pesos iniciales w de la neurona, los cambiamos por

$$w' = w - \alpha \nabla_w L, \tag{3.3}$$

donde $\alpha > 0$ es un hiperparámetro, denominado tasa de aprendizaje, y $\nabla_w L$ es el vector gradiente de L con respecto a w. En el caso de una neurona lineal ($f \equiv \mathrm{id}$, $\hat{y}_i = w^T x_i$) podemos expresar (3.2) matricialmente como

$$L = w^T R w - 2w^T p + \frac{1}{N}\sum_{i=1}^{N} y_i^2, \tag{3.4}$$

siendo $R = 1/N \sum_{i=1}^{N} x_i x_i^T \in \mathcal{M}_n$ y $p = 1/N \sum_{i=1}^{N} y_i x_i \in \mathbb{R}^n$, una matriz y un vector constantes, respectivamente, por lo que

$$\nabla_w L = 2(Rw - p).$$

Observamos que la función de error (3.4) es convexa, por lo tanto, iterando el algoritmo (3.3) con α suficientemente pequeño llegaríamos a minimizar la función de error, cuando $\nabla_w L = 0$, lo que implica que los pesos óptimos son

$$w^* = R^{-1} p.$$

Por lo tanto, en el caso de una neurona lineal podemos hallar directamente los valores de pesos sinápticos óptimos. Para funciones de activación no lineales se pueden aplicar estrategias numéricas de descenso de gradiente, como por ejemplo la regla delta [12].

3.2. *Backpropagation*

Para el caso del perceptrón multicapa, se utiliza ampliamente el algoritmo de retropropagación del error (*backpropagation*, en inglés). Este algoritmo apareció en la década de 1970, pero no fue hasta 1986 cuando se le empezó a dar importancia, gracias al trabajo seminal de Rumelhart, Hinton y Williams [13]. La idea inicial es la misma que en la Sección 3.1. El algoritmo consiste en una pasada hacia delante en la que se calculan las salidas de la red $\hat{y}_i = F(x_i; W)$ para cada dato x_i y otra hacia atrás en la que se actualizan los pesos W usando (3.3).

Supongamos que tenemos una red neuronal de k capas $F : \mathbb{R}^{n_0} \to \mathbb{R}^{n_k}$, en la que la capa m-ésima aplica la función $F^{(m)}$ con pesos $W^{(m)} = (w_{ij}^{(m)})_{i,j}$, para $m \in \{1, \ldots, k\}$. Para ver cómo actualizar los pesos de acuerdo con el algoritmo de descenso de gradiente vamos a fijarnos primero en la capa de salida. El vector de pesos $\omega = w_i^{(k)}$ de su neurona i-ésima se actualizará de acuerdo con (3.3) utilizando la siguiente función de error:

$$L = L_i = \frac{1}{N}\sum_{j=1}^{N} \varepsilon_{ij}, \tag{3.5}$$

donde, de nuevo, N es el número de ejemplos de los que disponemos y

$$\varepsilon_{ij} = \frac{1}{2}(\hat{y}_{ij} - y_{ij})^2$$

es el error cometido en el ejemplo j-ésimo por la neurona i.

Ahora bien,

$$\nabla_\omega L = \left(\frac{\partial L}{\partial \omega_1}, \ldots, \frac{\partial L}{\partial \omega_{n_{k-1}}}\right)^T,$$

con

$$\frac{\partial L}{\partial \omega_\ell} = \frac{1}{N}\sum_{j=1}^{N} \frac{\partial \varepsilon_{ij}}{\partial \omega_\ell} \tag{3.6}$$

para $\ell \in \{1, \ldots, n_{k-1}\}$.

Así, el problema se divide en calcular la derivada del error de cada ejemplo x_j concreto respecto de cada una de las componentes de ω por separado. Utilizando la regla de la cadena, podemos expresar esto de la siguiente forma:

$$\frac{\partial \varepsilon_{ij}}{\partial \omega_\ell} = \frac{\partial \varepsilon_{ij}}{\partial v_i^{(k)}} \frac{\partial v_i^{(k)}}{\partial \omega_\ell}, \tag{3.7}$$

donde

$$v_i^{(k)} = \langle \omega_i^{(k)}, \xi^{(k-1)} \rangle \tag{3.8}$$

y $\xi^{(r)}$ se define recursivamente de la forma

$$\begin{aligned} \xi^{(1)} &= f^{(1)}(W^{(1)} x_j); \\ \xi^{(r)} &= f^{(r)}(W^{(r)} \xi^{(r-1)}), \quad r > 1. \end{aligned}$$

Esto procede de que en ε_{ij} el único término no constante es $\hat{y}_{ij}$, que podemos escribir como

$$\hat{y}_{ij} = f_i^{(k)}(v_i^{(k)}).$$

La primera derivada de (3.7) se denota $\delta_i^{(k)}$ y la segunda se puede calcular de la siguiente forma:

$$\frac{\partial v_i^{(k)}}{\partial \omega_\ell} = \frac{\partial}{\partial \omega_\ell} \sum_{r=1}^{n_{k-1}} \omega_r \xi_r^{(k-1)} = \xi_\ell^{(k-1)}.$$

Así,

$$\frac{\partial \varepsilon_{ij}}{\partial \omega_\ell} = \delta_i^{(k)} \xi_\ell^{(k-1)}. \tag{3.9}$$

Por tanto, para el cálculo necesitamos conocer $\xi_\ell^{(k-1)}$, que no es más que la salida de la neurona ℓ-ésima de la capa anterior, y que podemos calcular y almacenar durante la fase hacia delante; pero también debemos calcular

$$\delta_k^{(k)} = \frac{\partial \varepsilon_{ij}}{\partial v_i^{(k)}} = f_i^{(k)\prime}(v_i^{(k)}), \tag{3.10}$$

por lo que es suficiente con conocer la derivada de la función $f_i^{(k)}$ y sustituirla en $v_i^{(k)}$ durante la pasada hacia delante.

Ahora veamos lo que sucede para las capas ocultas $h \in \{1, \ldots, k-1\}$. En principio no tenemos información acerca del resultado intermedio que debiera haber generado, como ocurre con la capa final. Sin embargo, podemos utilizar el error final y la regla de la cadena en la neurona i para derivar en función del vector de pesos $\omega = w_i^{(h)}$.

Como el resultado de la neurona i-ésima de la capa h se propaga hacia todas las neuronas de la siguiente capa, afectará a todas las componentes de la salida final de la red neuronal. Por tanto, durante la fase hacia detrás debemos tener en cuenta todos los posibles caminos por los que se pueda llegar a dicha neurona. Esto se hace de la siguiente forma:

$$\delta_i^{(h)} = \frac{\partial \varepsilon_{ij}}{\partial v_i^{(h)}} = \sum_{r=1}^{n_h} \frac{\partial \varepsilon_{ij}}{\partial v_r^{(h+1)}} \frac{\partial v_r^{(h+1)}}{\partial v_i^{(h)}} = \sum_{r=1}^{n_{h+1}} \delta_r^{(h+1)} \frac{\partial v_r^{(h+1)}}{\partial v_i^{(h)}}. \tag{3.11}$$

Cada uno de los $\delta_r^{(h+1)}$ se calcula de forma similar a como se ha explicado en (3.10) y las otras derivadas se obtienen de (3.8):

$$\frac{\partial v_r^{(h+1)}}{\partial v_i^{(h)}} = \frac{\partial}{\partial v_i^{(h)}} \sum_{q=1}^{n_h} w_{rq}^{(h+1)} f_q^{(h)}(\langle w_q^{(h)}, \xi^{(h-1)} \rangle) = w_{ri}^{(h+1)} f_i^{(h)\prime}(v_i^{(h)}). \tag{3.12}$$

Finalmente, sustituyendo (3.12) en (3.11) tenemos el algoritmo completo. Es un algoritmo de programación dinámica en el que se precalcula todo lo necesario durante la etapa hacia delante para luego poder utilizarlo durante la fase hacia detrás sin tener que repetir operaciones.

En todo momento hemos asumido que las funciones $\mathbf{f}^{(r)}$ son derivables y en muchos libros, como en [14] o en [15], se suele utilizar directamente la función sigmoide (2.3), que cumple dicho requisito. No obstante, en las capas ocultas se suele utilizar la función ReLU (2.2) pues, como ya se mencionó, es mucho más simple e igualmente añade no linealidad a la red. Sin embargo, esta función no es derivable en 0, por lo que no cumpliría las precondiciones del algoritmo, pero se sigue utilizando dando valor 0 a la derivada en dicho punto. En la práctica esto no supone un problema, ya que el algoritmo de descenso de gradiente sigue teniendo un buen comportamiento.

3.2.1. Sobreaprendizaje

Uno de los objetivos del aprendizaje es conseguir que la red tenga capacidad de generalizar, es decir, poder predecir el resultado para datos nuevos de manera correcta.

Pero un problema que puede ocurrir es que la red se "aprenda de memoria" los ejemplos de entrenamiento, es decir, ajuste demasiado sus pesos a ellos, y cuando encuentre un ejemplo nuevo no sea capaz de generalizar. Este problema se denomina sobreaprendizaje (*overfitting*, en inglés). Por tanto, es habitual dividir el conjunto de ejemplos $\mathcal{X}$ en dos disjuntos: el conjunto de entrenamiento X y el de prueba o test $\bar{X}$. De esta forma, sólo se entrenará la red con los ejemplos de entrenamiento y se podrá estimar cómo se va a comportar realmente con datos que la red aún no ha visto usando el conjunto de test. Así, si ha habido sobreaprendizaje se podrán tomar medidas para evitarlo, como terminar el proceso de aprendizaje antes de que esto ocurra.

3.3. Ejemplos de aprendizaje

En esta sección vamos a ilustrar el funcionamiento de una red neuronal con capas densas con ejemplos prácticos. Para ello vamos a usar *Matlab* [16] con la extensión *Neural Network Toolbox* [17], que permite crear y entrenar redes neuronales de cualquier tipo. Estos ejemplos y los que aparecen en secciones posteriores se pueden encontrar en el siguiente repositorio de GitHub: `https://github.com/Soy-yo/tfg-matematicas`. Los ejemplos se encuentran bajo el directorio *examples* y algunos otros programas que han servido de ayuda para la realización de este trabajo se encuentran en *other_scripts*.

3.3.1. Reconocimiento de dígitos escritos a mano

Vamos a usar como primer ejemplo el reconocimiento de dígitos escritos a mano, usando el conjunto de datos MNIST [18]. Este *dataset* contiene 70 000 imágenes de tamaño 28×28 píxeles con números escritos a mano, como se muestra en la Figura 3.1, y sus respectivas etiquetas. Los dígitos desde el 0 hasta el 9 aparecen con una distribución relativamente uniforme. El conjunto está dividido en un subconjunto de entrenamiento de 60 000 ejemplos y uno de test con 10 000 imágenes.

Figura 3.1: Ejemplos de dígitos escritos a mano (conjunto MNIST).

El objetivo de este experimento será entrenar una red neuronal con el conjunto de entrenamiento que sea capaz de generalizar al conjunto de test y conseguir la mayor precisión de reconocimiento de dígitos posible.

Para ello, vamos a construir una red de tres capas:

1. Una capa de entrada de $28 \times 28 = 784$ neuronas (una por píxel).
2. Una capa densa oculta de 512 neuronas con función de activación ReLU.
3. Una capa de salida de 10 neuronas con función de activación softmax.

La capa de entrada aceptará las imágenes como vectores de $\mathbb{R}^{784}$ y la capa de salida devuelve la probabilidad de que la imagen pertenezca a cada una de las diez clases. Por tanto, la clasificación se hará eligiendo la etiqueta que tenga mayor probabilidad. Además, de cara a la automatización en *Matlab* y a un mejor cálculo de la función de error, hay que añadir una capa extra que permite al programa tanto obtener el resultado de la red, como calcular la entropía cruzada, que es la función de error que se suele utilizar en este tipo de problemas:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\langle y_i, \log \hat{y}_i\rangle, \tag{3.13}$$

entendiendo log como el logaritmo natural "vectorial" $\log u = (\log u_1, \ldots, \log u_n)^T$ para $u \in \mathbb{R}^n$.

Una vez definida la arquitectura de la red podemos pasar a entrenarla. El entrenamiento se llevó a cabo durante 4 épocas (número de veces que se muestran todos los ejemplos de entrenamiento), con una tasa de aprendizaje $\alpha = 0{,}01$ y utilizando el algoritmo de descenso de gradiente estocástico para actualizar los pesos en el algoritmo de *backpropagation*. Además, se ha utilizado un conjunto extra de validación, un subconjunto del conjunto de entrenamiento que no se va a utilizar para entrenar, sino que se va a mostrar a la red cada cierto número de iteraciones para obtener una aproximación de su comportamiento real durante el entrenamiento, por si fuera necesario terminarlo antes de tiempo.

Todo esto ha llevado a la red a entrenar durante 2:25 minutos[1] obteniendo una precisión en el conjunto de validación del 97,12 %. El progreso del aprendizaje lo podemos observar en la Figura 3.2. Después se evaluó el rendimiento de la red entrenada con el conjunto de test, donde obtuvo una precisión de 97,73 %, algo superior a la obtenida en validación.

Podemos clasificar el resultado obtenido como muy bueno. La Figura 3.3 muestra ejemplos de predicción y la probabilidad de la misma. En este caso, ha logrado predecir correctamente todos los ejemplos con alta seguridad. Este éxito es debido a que el conjunto de datos que hemos utilizado es relativamente simple. No obstante, las capas densas no son las apropiadas para este tipo de problemas, ya que pierden cualquier relación espacial que hubiera en las imágenes.

[1] Usamos una GPU *Nvidia RTX 3060 Ti* de 8 GB de VRAM. *Matlab* permite realizar el entrenamiento en tarjetas gráficas (GPU), que aumentan la velocidad de cómputo de las operaciones tensoriales con respecto al uso de CPU.

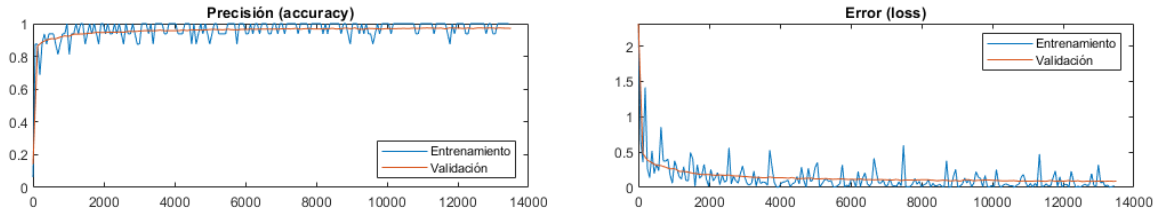


Figura 3.2: Precisión (izquierda) y error L (derecha) en los conjuntos de entrenamiento y validación para el *dataset* MNIST a lo largo del entrenamiento. En unas 500 iteraciones se llega a una precisión superior al 90 % de aciertos.

Figura 3.3: Ejemplos de reconocimiento de dígitos. Para cada imagen se indica el dígito predicho por la red junto con la probabilidad con que lo estima. En este caso, la red tiene una certeza superior al 97 % para todos los ejemplos, excepto para el cuatro, que pudiera parecer también un nueve y le da alrededor de un 19 % de serlo.

3.3.2. Clasificación de vehículos y animales en imágenes

En este ejemplo el objetivo es construir una red neuronal capaz de determinar qué vehículo o animal se muestra en una imagen de entre los diez posibles: avión, coche, pájaro, gato, ciervo, perro, rana, caballo, barco o camión. El *dataset* CIFAR-10 [19] está compuesto por 60 000 imágenes a color de tamaño 32×32 píxeles etiquetadas y dividido en un conjunto de entrenamiento de 50 000 ejemplos y otro de test de 10 000. Podemos ver algunos ejemplos en la Figura 3.4.

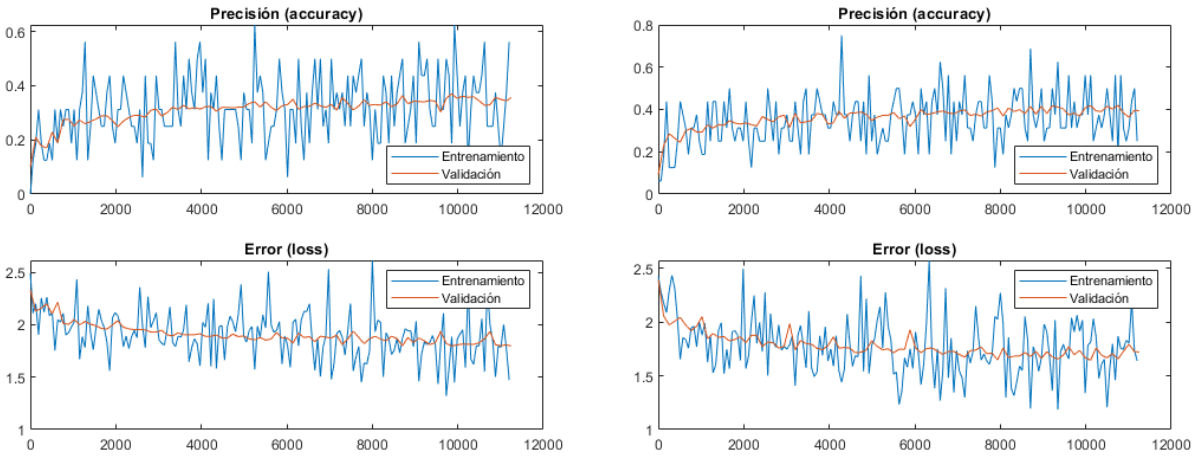
Vamos a realizar este experimento para demostrar que este tipo de problemas no son los apropiados para las redes neuronales densas y que deberíamos usar redes convolucionales en su lugar (Capítulo 4).

Para comprobar esto se han entrenado dos redes neuronales similares, una con las imágenes en blanco y negro para reducir la cantidad de neuronas necesarias en la capa de entrada y otra con las imágenes a color. Ambas redes constan de una capa de entrada de tamaños $1024 = 32 \times 32$ y $3072 = 32 \times 32 \times 3$, respectivamente, dos capas ocultas y una capa de salida de diez neuronas con activación softmax. Las capas ocultas usan ReLU como función de activación y las de la primera red constan de 512 y 1024 neuronas cada una, mientras que las de la segunda tienen 1024 y 2048, respectivamente, ya que la entrada es de mayor tamaño.

El entrenamiento se llevó a cabo con los mismos parámetros que en la Sección 3.3.1 y también se utilizó un 10 % de los datos como conjunto de validación.

El proceso de entrenamiento se muestra en la Figura 3.5. Tras alrededor de 2:15 minutos de entrenamiento para cada red, se obtuvieron los siguientes resultados: un 35,62 % de precisión en el conjunto de validación en blanco y negro y un 39,36 % en el de color; y un 34,69 % y 38,60 % en los de test, respectivamente. Además, comparando las Figuras 3.2 y 3.5 observamos la gran diferencia en las funciones de error, que en este caso no logra bajar de 1,5 en ninguna

Figura 3.4: Ejemplos de imágenes del conjunto CIFAR-10 a color y en blanco y negro con sus respectivas etiquetas.



(a) Blanco y negro.

(b) A color.

Figura 3.5: Resultados de clasificación de vehículos y animales en una red con capas densas: precisión y error en los conjuntos de entrenamiento y validación para el *dataset* CIFAR-10. a) Resultados para el entrenamiento con imágenes en blanco y negro. b) Resultados con las imágenes originales.

de las dos redes.

Las redes con capas densas pierden toda la información espacial y de color, lo que es fundamental en problemas de clasificación de imágenes. Sin embargo, podemos afirmar que sí que han logrado generalizar los datos de entrenamiento, puesto que el rendimiento final ha sido más de tres veces mejor que el de un modelo aleatorio, el cual lograría un 10 % de precisión.

Capítulo 4

Redes convolucionales

En este capítulo se va a tratar un tipo de redes neuronales diseñadas específicamente para resolver problemas de visión por ordenador.

Como se ha comprobado en la Sección 3.3.2, las redes neuronales densas no son las apropiadas a la hora de tratar con imágenes. Esto es debido a que este tipo de redes únicamente son capaces de encontrar patrones globales en la entrada, es decir, si un mismo patrón aparece en lugares diferentes de una imagen, la red neuronal necesitaría aprender dos patrones por separado, aunque en realidad sea el mismo pero trasladado. Las redes convolucionales, por su parte, son invariantes a las traslaciones, pero también son capaces de detectar rotaciones o cambios en la perspectiva. Esto lo logran utilizando un tipo de procesado distinto sobre las imágenes de entrada.

El objetivo de las redes convolucionales, por tanto, consiste en aplicar sucesivamente una operación llamada convolución sobre la imagen de entrada para encontrar las zonas más relevantes de la misma. En la Figura 4.1 podemos ver un ejemplo de la convolución denominada desenfoque gaussiano, que se utiliza a menudo en programas de edición fotográfica.

4.1. Convolución

Comencemos definiendo la operación convolución [20].

Definición 4.1 (Convolución)**.** Sean $f, g : \mathbb{R}^n \to \mathbb{R}$. Se define la convolución de f y g como

$$(f * g)(x) = \int_{\mathbb{R}^n} f(u)g(x-u)\,\mathrm{d}u \tag{4.1}$$

allá donde la integral esté definida. Aquí, g se denomina el núcleo de la convolución.

Es habitual que f ó g no estén definidas en todo $\mathbb{R}^n$. En estos casos vamos a considerar su extensión tomando un valor nulo en aquellos puntos en los que no estén definidas.

Para el caso de un dominio discreto, es decir, si $f, g : \mathbb{Z}^n \to \mathbb{R}$, se define la convolución utilizando sumas infinitas.

$$(f * g)(k) = \sum_{i \in \mathbb{Z}^n} f(i)g(k-i), \tag{4.2}$$

donde $k \in \mathbb{Z}^n$. En concreto, las funciones que vamos a utilizar en este capítulo estarán definidas en $\mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n}$, donde $\mathbb{Z}_m = \{0, \ldots, m-1\}$, por lo que en el resto de puntos se definen como 0 y podemos escribir (4.2) utilizando sumas finitas.

Figura 4.1: Ejemplo de aplicación de la convolución desenfoque gaussiano sobre una imagen. A la izquierda la imagen original y a la derecha la imagen resultante tras aplicar la convolución con el núcleo gaussiano.

4.1.1. Relación de la convolución con el producto de polinomios

En este trabajo vamos a utilizar las convoluciones aplicadas al procesamiento de imágenes, pero tienen muchos otros usos en multitud de campos. Por ejemplo, sirven para calcular los coeficientes de un producto de dos polinomios. Sean $p(x) = a_0 + a_1x + \cdots + a_nx^n$, $q(x) = b_0 + b_1x + \cdots + b_mx^m$ y $r(x) = p(x)q(x) = c_0 + c_1x + \cdots + c_{nm}x^{nm}$ y definamos $\alpha : \mathbb{Z}_{n+1} \to \mathbb{R}$ y $\beta : \mathbb{Z}_{m+1} \to \mathbb{R}$ como $i \mapsto \alpha(i) = a_i$ y $j \mapsto \beta(j) = b_j$, respectivamente. Entonces se puede comprobar que $\gamma : \mathbb{Z}_{nm+1} \to \mathbb{R}$, $\gamma(k) = c_k$ se puede calcular como

$$\gamma(k) = (\alpha * \beta)(k) = \sum_{i=0}^{k} \alpha(i)\beta(k-i),$$

ya que si $i < 0$, $\alpha(i) = 0$ (usando la extensión de α a $\mathbb{Z}$) y si $i > k$, $\beta(k-i) = 0$.

4.2. Estructura de una red convolucional

4.2.1. Capas convolucionales

Una red convolucional está dividida también en múltiples capas, igual que las redes densas, pero cada una de estas capas recibe una entrada de tipo diferente y aplica operaciones distintas.

La entrada de una capa convolucional es un tensor de rango tres o cuatro. En este contexto vamos a entender un tensor como una generalización de los vectores y matrices para almacenar información de forma ordenada, aunque su definición real es algo más compleja de la que aquí se muestra.

Definición 4.2 (Tensor)**.** Un tensor de rango $n \geq 0$ es una aplicación $x : \mathbb{Z}_{m_1} \times \cdots \times \mathbb{Z}_{m_n} \to \mathbb{R}$. A cada uno de los elementos del producto se les denomina ejes y el número de componentes m_i de cada eje es su dimensión.

Por simplicidad a la hora de escribir, vamos a denotar $\mathcal{T}(m_1, \ldots, m_n)$ al conjunto de tensores de rango n y dimensiones $m_1, \ldots, m_n$ en cada eje, respectivamente.

Además, con : se denotará la proyección del tensor sobre el eje indicado por la componente en que se encuentre:

$$x(i_1, \ldots, i_{k-1}, :, i_{k+1} \ldots, i_n) = y \in \mathcal{T}(m_k), \quad y(j) = x(i_1, \ldots, i_{k-1}, j, i_{k+1}, \ldots, i_n).$$

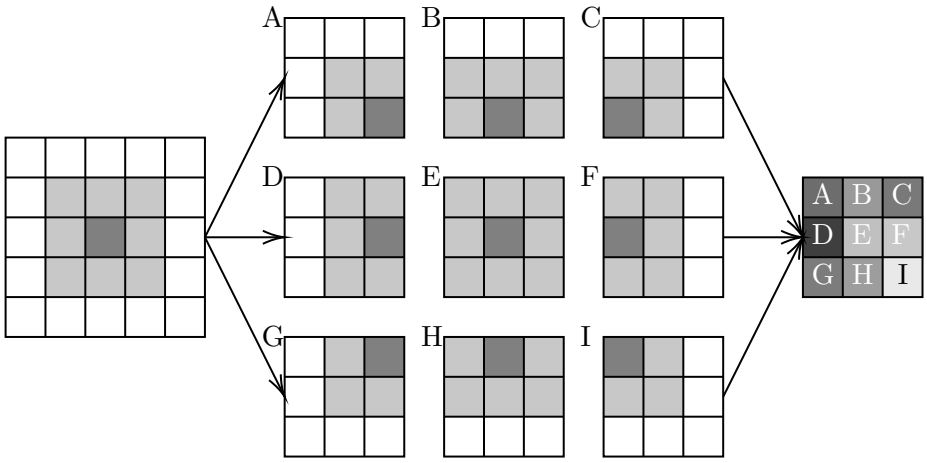


Figura 4.2: Ejemplo de aplicación de un filtro ω de tamaño 3×3 a una entrada x 5×5. El filtro se puede aplicar en las nueve posiciones que se muestran diferenciadas y produce una salida y 3×3. La intensidad del gris de cada píxel en la salida se refiere al posible valor resultante tras aplicar el filtro a la parte correspondiente de la entrada.

Observamos que un tensor de rango 0 representa un escalar ($x \in \mathbb{R}$), uno de rango 1 un vector $\mathbf{x} = (x_0, \ldots, x_{n-1}) \in \mathbb{R}^n$ ($x(i) = x_i$, tal que $0 \leq i < n$) y uno de rango 2 una matriz $X = (x_{ij})_{i,j} \in \mathcal{M}_{m \times n}(\mathbb{R})$ ($x(i,j) = x_{ij}$, con $0 \leq i < m$ y $0 \leq j < n$, usando 0 como primer índice).

Así, la entrada de una capa de una red convolucional es un tensor $x \in \mathcal{T}(H, W, D)$, donde H representa la altura de la entrada, W la anchura y D la profundidad. Específicamente, la entrada de la red convolucional será habitualmente una imagen, con una altura y anchura H y W, en píxeles, fijadas por la capa de entrada, y profundidad $D = 3$, ya que tienen un canal para cada color básico: rojo, verde y azul. Cada uno de estos canales representa la intensidad de dicho color y viene dada, en general, por valores en $[0, 1]$ o $\{0, \ldots, 255\}$. Para el caso de imágenes en blanco y negro, como las de la Sección 3.3.1, será suficiente con utilizar un único canal ($D = 1$) que represente el brillo del píxel, lo que reduce la cantidad de pesos necesarios en la red.

Si quisiéramos trabajar con vídeos en lugar de imágenes deberíamos añadir un eje más de T dimensiones para el número de fotogramas que se toman, pero como aquí sólo nos vamos a centrar en imágenes el rango de los tensores va a ser siempre 3.

La salida de una capa será un nuevo tensor $y \in \mathcal{T}(H', W', D')$, de rango 3. En las capas ocultas de la red la profundidad D' no significa el número de canales de color, sino el número de rasgos que se han detectado y suele ser un número mucho mayor que 3.

Por último, cada capa dispone de un tensor $\omega \in \mathcal{T}(h, w, D, D')$ de D' filtros, o núcleos, de tamaño $h \times w \times D$. Este tensor es el que contiene los pesos de la capa. Cada uno de estos filtros se aplica en cada una de todas las posibles posiciones de la entrada donde es posible, de la forma que se muestra en la Figura 4.2, produciendo un valor cada vez. Por tanto, para una entrada $x \in \mathcal{T}(H, W, D)$ se obtiene un tensor de salida $y \in \mathcal{T}(H', W', D')$, con $H' = H - h + 1$ y $W' = W - w + 1$. Así, para que esto sea posible es necesario que $h \leq H$ y $w \leq W$.

La aplicación de un filtro es un caso especial de una convolución aplicada a tensores, pero que produce tensores como resultado. Esto lo vamos a hacer calculando la convolución normalmente, pero ignorando los resultados que no nos interesan.

Con todo esto, podemos definir las capas convolucionales formalmente.

Definición 4.3 (Capa convolucional)**.** Una capa convolucional con núcleo de dimensiones (h, w) y D' filtros es una aplicación de la forma

$$F : \mathcal{T}(H, W, D) \times \mathcal{T}(h, w, D, D') \times \mathcal{T}(D') \to \mathcal{T}(H', W', D')$$
$$(x; \omega, \theta) \mapsto y = F(x; \omega, \theta),$$

con $H' = H - h + 1$ y $W' = W - w + 1$ que cumple

$$y(:, :, k) = f(x * \omega(:, :, :, k) - \theta(k)),$$

donde $\omega \in \mathcal{T}(h, w, D, D')$ es el tensor de pesos de todos los canales, $\theta \in \mathcal{T}(D')$ es el tensor de sesgos o umbrales de activación de los filtros, $x \in \mathcal{T}(H, W, D)$ es el tensor de entrada e $y \in \mathcal{T}(H', W', D')$ el de salida. La función $f : \mathcal{T}(H', W', D') \to \mathcal{T}(H', W', D')$ es la función de activación, que se define mediante funciones $f_{ijk} : \mathbb{R} \to \mathbb{R}$ como

$$f(u)(i, j, k) = f_{ijk}(u(i, j, k))$$

para cada i, j y k. Es habitual que $f_{ijk} \equiv g$ para cierta $g : \mathbb{R} \to \mathbb{R}$.

Comprobemos que las dimensiones del tensor de salida coinciden con lo esperado. Sean $\omega_\ell = \omega(:, :, :, \ell)$ y $v_\ell = x * \omega_\ell$ para cierto $\ell \in \mathbb{Z}_{D'}$. Por (4.2) tenemos que

$$v_\ell(i, j, k) = \sum_{r=-\infty}^{\infty} \sum_{s=-\infty}^{\infty} \sum_{t=-\infty}^{\infty} x(r, s, t)\omega_\ell(i - r, j - s, k - t), \quad (i, j, k) \in \mathbb{Z}^3, \tag{4.3}$$

extendiendo a $\mathbb{Z}^3$. Pero $\omega_\ell \in \mathcal{T}(h, w, D)$, por lo que $\omega_\ell(i-r, j-s, k-t) = 0$ si $r \notin [i, i-h+1]$, $s \notin [j, j - w + 1]$ o $t \notin [k, k - D + 1]$; y $x \in \mathcal{T}(H, W, D)$, lo que implica que $x(r, s, t) = 0$ cuando $r \notin \mathbb{Z}_H$, $j \notin \mathbb{Z}_W$ o $t \notin \mathbb{Z}_D$. Así, podemos simplificar (4.3) de la siguiente forma:

$$\begin{cases} v_\ell(i, j, k) = \displaystyle\sum_{r=i}^{i+h-1} \sum_{s=j}^{j+w-1} \sum_{t=k}^{k+D-1} x(r, s, t)\omega_\ell(i - r, j - s, k - t), & (i, j, k) \in \mathbb{Z}_H \times \mathbb{Z}_W \times \mathbb{Z}_D \\ v_\ell(i, j, k) = 0 & \textit{e.o.c.} \end{cases} \tag{4.4}$$

Pero, por un lado, cuando $k \geq 1$ se alcanza $x(:, :, D)$, donde ya no está definida la entrada, por lo que podemos ignorar este índice, ya que siempre es 0. Por otro lado, cuando $i \geq H'$ o $j \geq W'$ se alcanzan $x(H, :, :)$ o $x(:, W, :)$, donde tampoco está definida, por lo que podemos reducir la definición de v_ℓ a $\mathcal{T}(H', W')$, lo que queríamos.

Muchas veces interesa que las capas convolucionales puedan aplicar los filtros correctamente a los bordes de la entrada, pues podrían contener información relevante. Para lograr esto se suele ampliar la entrada por los bordes con un marco de tamaño suficiente para que la salida de la capa preserve el tamaño exacto que tenía la entrada. Por ejemplo, al aplicar filtros 3×3 a una entrada de tamaño $H \times W$ podemos añadir un borde de un píxel a cada lado, de forma que la nueva entrada sea de tamaño $(H + 2) \times (W + 2)$ y, por tanto, la salida tenga una altura $H' = (H + 2) - 3 + 1 = H$ y anchura $W' = W$. De esta forma, además, se evita reducir el tamaño de la entrada y permite crear redes más profundas. Esta operación se denomina *padding* o relleno.

En la Figura 4.3 podemos ver un ejemplo del resultado de aplicar una capa convolucional (sin relleno) de 8 filtros 3×3 prefijados a una determinada imagen en blanco y negro. La imagen procede del conjunto de datos CIFAR-10 y es tan sólo uno de los canales de color de uno de los ejemplos. Como podemos ver, obtendríamos ocho imágenes que después podrían ser interpretadas en su conjunto como $\mathcal{T}(30, 30, 8)$ por una capa posterior.

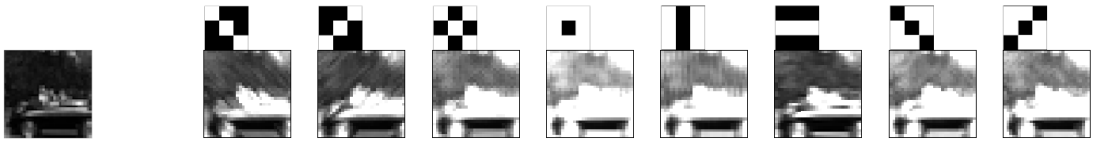


Figura 4.3: Ejemplo de aplicación de una serie de filtros sobre una imagen en blanco y negro. A la izquierda la imagen original, de tamaño 32×32, y a la derecha los resultados de tamaño 30×30 tras aplicar cada uno de los ocho filtros. Las matrices de cada filtro se muestran en la parte superior.

Los filtros que se usan en cada capa no se suelen establecer a mano, sino que se aprenden de la misma forma que en las capas densas, aplicando el algoritmo de retropropagación para calcular cada uno de los pesos dentro de los filtros, lo que permite que la red pueda utilizar los más apropiados para cada problema concreto.

4.2.2. Otras capas de las redes convolucionales

Existen redes convolucionales con arquitecturas muy diversas, pero entre las más básicas podemos encontrar aquellas que están formadas por varios grupos de capas convolucionales terminados en un un grupo de capas densas. Primero, las capas convolucionales obtendrán una representación de la imagen como un tensor de tamaño $H' \times W' \times D'$; después, esta representación se aplana, es decir, se convierte a un vector de $\mathbb{R}^{H'W'D'}$; y, finalmente, se utilizan una serie de capas densas para predecir el resultado final.

Se entiende como grupo de capas convolucionales a la sucesión de una o más capas convolucionales que terminan con una capa de reducción de dimensionalidad. Una capa de reducción de dimensionalidad particiona la entrada en rectángulos de tamaño $h \times w$ y aplica una operación prefijada, por lo que no tiene pesos que entrenar. En concreto, las capas de reducción de dimensionalidad que más se utilizan son las llamadas *max-pooling*, las cuales toman el valor máximo de cada rectángulo. De esta forma, si la entrada tiene tamaño $H \times W \times D$, la salida será un tensor de tamaño $\lfloor H/h \rfloor \times \lfloor W/w \rfloor \times D$.

Esta reducción sirve para añadir más canales sin que esto suponga un gran coste computacional y para disminuir el número de pesos que utilizarán las capas densas finales, mientras se mantiene la información más relevante.

Por ejemplo, las redes convolucionales basadas en la arquitectura VGG [21] están formadas por múltiples grupos de capas convolucionales que mantienen el tamaño de la entrada y usan filtros de tamaño 3×3 y cada uno de ellos terminado en una capa de *max-pooling* de tamaño de ventana 2×2. Habitualmente, dentro de un mismo grupo se suele mantener el número de canales y al pasar al grupo siguiente se duplica, ya que cada una de las otras dos dimensiones se ha reducido a la mitad. Un ejemplo de esta arquitectura es el de la Figura 4.4, que se utilizará en la Sección 4.3. De esta forma, se pueden crear redes bastante profundas, es decir, con un gran número de capas, pues el tamaño únicamente se ve reducido al pasar de un grupo al siguiente.

4.3. Ejemplos de aprendizaje

Vamos a estudiar los mismos ejemplos que en la Sección 3.3, pero esta vez utilizando redes neuronales convolucionales para comprobar que se obtienen mejores resultados con estas. En ambos casos se han utilizado parámetros de entrenamiento similares a los usados en sus respectivos ejemplos con capas densas para que los resultados sean comparables.

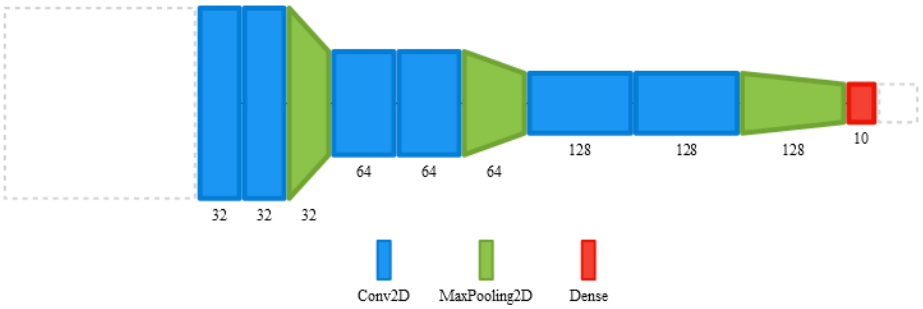


Figura 4.4: Ejemplo de una red basada en la arquitectura VGG [22].

En la Sección 3.3, utilizando capas densas para el conjunto de datos MNIST obtuvimos un 97,73 % de precisión en el conjunto de test. Ahora vamos a usar una red convolucional de tres grupos de capas convolucionales y un clasificador final completamente conectado de 10 neuronas, una por dígito, como la vez anterior. Cada grupo de capas convolucionales está formado por una capa convolucional de filtros 3×3 con función de activación ReLU y una capa de reducción de dimensionalidad *max-pooling* con filtros 2×2. De esta forma la entrada, de tamaño inicial 28×28, se reduce a 3×3 tras pasar todas las capas convolucionales. En cuanto al número de filtros, la primera aplica 16, la segunda 32 y la tercera 64. Por tanto, esta red está basada en VGG.

Una vez creada la red se ha entrenado durante cuatro épocas con los mismos parámetros que en el caso anterior. Esta vez el entrenamiento ha tardado 5:18 minutos, más del doble que cuando se usaron capas densas (2:25 minutos) y se ha obtenido un resultado mejor: 98,87 % de precisión en el conjunto de test. La Figura 4.5a un resumen del entrenamiento. Desde el punto de vista del error cometido esto es una gran mejora, puesto que la red densa cometía un 2,27 % de error, mientras que la convolucional tan sólo un 1,13 %, aproximadamente la mitad.

La diferencia en el tiempo de entrenamiento se debe a que la operación de convolución es mucho más lenta que la que aplican las capas densas, que no es más que una multiplicación de matrices. Además, en este caso estamos usando más capas que la otra vez, por lo que el número de operaciones a aplicar es también mayor.

Sin embargo, a pesar de tener más capas en realidad la red convolucional tiene menos parámetros. Esto es debido a que estas redes aplican un filtro pequeño a lo largo de toda la imagen, por lo que el número de pesos necesarios es menor. En concreto, la red densa usaba 407 050 pesos, contando con los umbrales de activación, mientras que la red convolucional tan sólo necesita 29 066, un 7 % de los que utiliza la completamente conectada.

Veamos ahora qué sucede con el conjunto de datos CIFAR-10. Esta vez sólo vamos a utilizar las imágenes a color, ya que ahora no afecta tanto al tamaño de la red. Recordemos que en la Sección 3.3.2 obtuvimos un 38,60 % de precisión en el conjunto de test, valor con amplio margen de mejora.

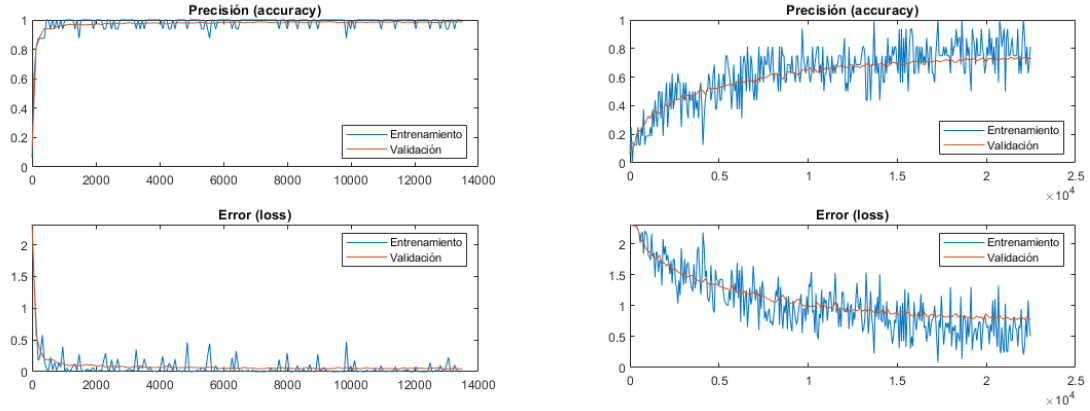
Para este ejemplo vamos a usar una red convolucional igual que la anterior, salvo porque cada grupo de capas convolucionales va a tener dos capas en vez de una y se va a comenzar con 32 filtros en el primer grupo, en vez de 16. Así, el segundo grupo usará 64 filtros y el tercero 64. La Figura 4.4 muestra la arquitectura de esta red. También se ha aumentado a ocho el número de épocas, ya que tras las cuatro primeras las gráficas del error y precisión continuaban mejorando (Figura 4.5b).

Tras 13:02 minutos esta red logró un 72,95 % de precisión en el conjunto de test, cerca del doble que lo que consiguió la otra vez. Con lo cual, de nuevo vemos claramente la gran

diferencia que hay entre estos dos tipos de redes para problemas con imágenes.

De nuevo, comparando el número de pesos de cada red sigue necesitando muchos menos parámetros la convolucional, a pesar de tener más filtros que en el caso anterior. Y es que, mientras que la red densa tenía 5 266 442 pesos, la convolucional únicamente necesita 307 498 (menos incluso que la red completamente conectada para MNIST), alrededor de un 6 % de la cantidad anterior.

El Cuadro 4.1 muestra una comparativa de los resultados entre los dos tipos de redes.



(a) MNIST (b) CIFAR-10

Figura 4.5: Entrenamiento con la red convolucional. a) Precisión y error en el conjunto MNIST. b) Precisión y error en CIFAR-10. Parece que aún no ha terminado de converger, por lo que se podrían haber mejorado los resultados.

	Red densa			Red convolucional		
	Precisión	Error	Pesos	Precisión	Error	Pesos
MNIST	97,73 %	2,27 %	407 050	98,87 %	1,13 %	29 066
CIFAR-10	38,60 %	61,40 %	5 266 442	72,95 %	27,05 %	307 498

Cuadro 4.1: Precisión y error obtenidos en el conjunto de test para cada problema con cada una de las redes, además del número de pesos sinápticos utilizados por cada una.

Capítulo 5

Aprendizaje Hebbiano

Hasta ahora, hemos trabajado con un modelo de aprendizaje denominado supervisado. Esto quiere decir que en el conjunto de entrenamiento conocemos cuál es el resultado esperado y corregimos el modelo para que sus resultados se acerquen lo más posible a ellos. Por tanto, nuestro conjunto de entrenamiento $\mathcal{X}$ estaba compuesto por pares $(x_i, y_i) \in \mathbb{R}^n \times Y$. Por contra, en los modelos basados en aprendizaje no supervisado no se dispone del valor y_i esperado, sino que se espera que la máquina sea capaz de encontrar diferencias en los datos de entrada. Por tanto, no se busca que el modelo prediga un valor asociado a un dato de entrada, sino que asocie entradas parecidas entre ellas, y será cuestión del analista decidir el significado detrás de dicha relación.

Por ejemplo, uno de los algoritmos de aprendizaje no supervisado más típicos es el llamado k-medias (k-*means*, en inglés) [23]. Este algoritmo sirve para crear k particiones en el conjunto de datos. Comienza eligiendo k centroides aleatorios en $\mathbb{R}^n$ para cada uno de los subconjuntos y seleccionando los puntos más cercanos a cada centroide como pertenecientes a dicho conjunto. Tras esto, se calculan los nuevos centroides de cada cluster y se repite la operación de nuevo. Este proceso se itera hasta lograr una distancia entre los antiguos centroides y los nuevos menor que un determinado umbral. Lo habitual también es elegir varias opciones de k y quedarse con aquella que genere clusters con la menor entropía.

Este tipo de aprendizaje se puede utilizar, por ejemplo, para formar grupos de usuarios de una aplicación. Conociendo diferentes datos de los usuarios, como el género, la edad, etc., y sus hábitos de uso de la aplicación, podemos utilizar aprendizaje no supervisado para formar grupos de usuarios y así saber a quién dirigir la aplicación y de qué forma.

En este capítulo nos vamos a centrar en un tipo de neuronas, denominadas neuronas conceptuales, que imitan a sus homólogas en el cerebro y que se entrenan utilizando el llamado aprendizaje Hebbiano (no supervisado) [24].

5.1. Neuronas conceptuales

Hasta el momento se había creído que los conceptos abstractos en el cerbero humano requerían de una compleja interacción entre millones de neuronas. Sin embargo, va cobrando peso una teoría que dice que son neuronas individuales las que se hacen cargo de un concepto. Estas se denominan neuronas conceptuales [25].

Vamos a desarrollar una neurona artificial capaz de imitar el trabajo de dichas neuronas conceptuales. Estas neuronas forman dos capas, la capa selectiva y la capa conceptual. Aquí, nos vamos a centrar únicamente en la primera de ellas, ya que, por la complejidad añadida,

la segunda queda fuera del objeto de este proyecto.

Las diferencias fundamentales entre una capa de neuronas selectivas y una de perceptrones, como las del Capítulo 2, son el número de neuronas y la forma en que se entrenan. En cuanto al número de neuronas m, este debe ser mucho mayor que el número de estímulos N que se le vayan a presentar. De esta forma, en el mejor de los casos, será posible que cada neurona se alinee con un solo estímulo. Entraremos más en detalle en cuanto al entrenamiento en la Sección 5.3.

Por lo demás, estas neuronas son muy similares a las que ya hemos estudiado: disponen de un vector de pesos sinápticos $w \in \mathbb{R}^n$ y un umbral de activación $\theta \in \mathbb{R}$, que supondremos que será constante y global a todas las neuronas de la capa. La salida $y \in \mathbb{R}$ de una neurona ante un estímulo $x \in \mathbb{R}^n$ se calcula de la misma forma:

$$y = F(\langle w, x \rangle - \theta),$$

donde F es la función ReLU (2.2). Por tanto, la salida de la capa selectiva será el vector de $\mathbb{R}^m$ formado por cada una de las salidas de las m neuronas.

Igual que antes, estamos tratando de separar los estímulos mediante un hiperplano, pero con la diferencia de que ahora nuestro objetivo es separar un único estímulo de los demás, en lugar de separar conjuntos de estímulos entre sí. Para lograr esto, podemos aprovechar la denominada "bendición de la dimensionalidad" [26]. Este término es la contraposición a la "maldición de la dimensionalidad", que se ha utilizado habitualmente en el campo de Optimización y que significa que la alta dimensionalidad de los datos de entrada complica seriamente el tratamiento de los problemas. En este caso, la "bendición de la dimensionalidad" nos es de utilidad porque existen diferentes teoremas de separabilidad que versan sobre la probabilidad de separar un punto de otro utilizando un hiperplano. Uno de los resultados principales es el siguiente teorema.

Teorema 5.1. *Sean $n \geq 2$, $X = \{x_1, \ldots, x_N\} \subset B_n(0,1) \coloneqq \{x \in \mathbb{R}^n : \|x\| < 1\}$, $w \in \mathbb{R}^n$ y $\theta \in \mathbb{R}$ que cumplen $\langle w, x_1 \rangle > \theta$. Entonces, si $P(\mathrm{sep}(x, Y))$ es la probabilidad de que el punto x se separe del conjunto Y mediante el hiperplano $H = \{x \in \mathbb{R}^n : \langle w, x \rangle = \theta\}$ (es decir, $\langle w, x \rangle > \theta$ y $\langle w, y \rangle \leq \theta$ para todo $y \in Y$), se tiene que*

$$P(\mathrm{sep}(x_1, X \setminus \{x_1\})) \geq \left(1 - \frac{1}{2} r^n\right)^{N-1},$$

donde

$$r = \sqrt{1 - \frac{\theta^2}{\|w\|^2}}.$$

Demostración. Comencemos suponiendo que $N = 2$. Entonces queremos probar que

$$P(\mathrm{sep}(x_1, \{x_2\})) \geq 1 - \frac{1}{2} r^n.$$

Sea $A = \{x \in B_n(0,1) : \langle w, x \rangle > \theta\}$, el conjunto de puntos de la bola que quedan "por encima" de H. Con esto, la probabilidad de separar x_1 de x_2 viene dada por la probabilidad de que x_2 no pertenezca a A, es decir,

$$p = P(\mathrm{sep}(x_1, \{x_2\})) = 1 - \frac{V(A)}{V(B_n(0,1))},$$

donde V denota el volumen del conjunto. Ahora bien, sabemos que la distancia entre el origen y A es $\mathrm{dist}(0, A) = \theta / \|w\|$ y, por tanto, usando el Teorema de Pitágoras, A está contenido en

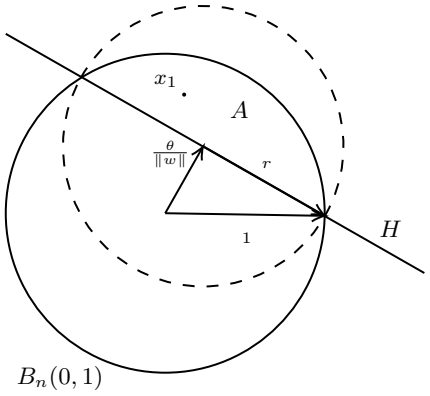


Figura 5.1: Esquema de la situación en el Teorema 5.1 para $n = 2$.

media bola de radio $r = \sqrt{1 - \theta^2/\|w\|^2}$ (Figura 5.1). Con esto,

$$p \geq 1 - \frac{1}{2}\frac{V(B_n(0,r))}{V(B_n(0,1))}.$$

Por otro lado, gracias a [27] sabemos que

$$V(B_n(0,R)) = \frac{1}{2}\frac{\pi^{n/2}}{\Gamma(n/2+1)}R^n,$$

con $R > 0$ y siendo Γ la función gamma

$$\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}\,\mathrm{d}t.$$

Por tanto,

$$p \geq 1 - \frac{1}{2}r^n. \tag{5.1}$$

Esto podemos entenderlo como un experimento de Bernoulli, por lo tanto, teniendo en cuenta los N puntos tendremos $N-1$ experimentos de Bernoulli independientes, con probabilidad p. Así, la variable aleatoria que representa el número de puntos que quedan separados de x_1 es una binomial $B \sim B(N-1,p)$, con

$$P(B = k) = \binom{N-1}{k} p^k (1-p)^{N-1-k}.$$

Como el caso que nos interesa es aquel con $k = N-1$, utilizando (5.1) tenemos

$$P(\mathrm{sep}(x_1, X \setminus \{x_1\}) = P(B = N-1) = p^{N-1} \geq \left(1 - \frac{1}{2}r^n\right)^{N-1}.$$

□

Para comprobar esto, hemos realizado un experimento numérico que consiste en lo siguiente: se han generado $N = 1000$ puntos uniformemente distribuidos en $B_n(0,1)$ para valores de n entre 1 y 30 y se ha elegido uno aleatoriamente como x_1. Después se han elegido un vector de pesos sinápticos w y un umbral de activación θ relativamente "ajustados" a x_1, como si ya se hubiera entrenado la neurona de forma que detecte x_1. Finalmente, se comprueba si x_1 se ha podido separar de los demás y se calcula la probabilidad que se obtendría según el Teorema 5.1. Repitiendo este experimento numerosas veces, utilizando el método de Monte

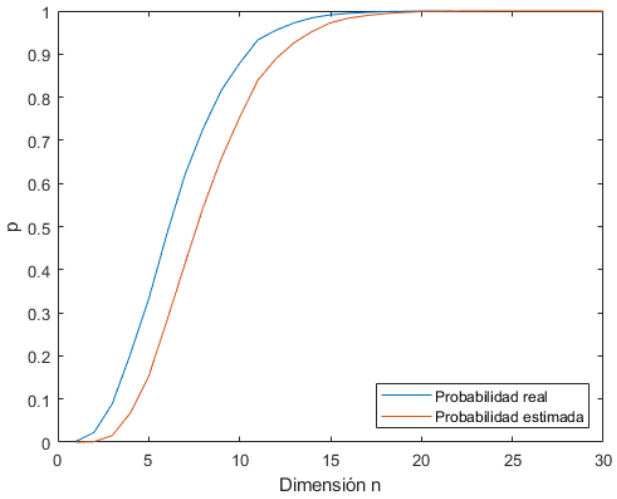


Figura 5.2: Probabilidad real y estimada de separar x_1 del resto de puntos.

Carlo, podemos estimar la probabilidad real de separar el punto, es decir, la obtenida empíricamente, y la dada por el teorema para comprobar que es correcto. La Figura 5.2 muestra el resultado de este experimento. De ella se puede destacar que aproximadamente a partir de una dimensión $n = 20$ se tiene una probabilidad cercana a 1 de separar un punto del conjunto.

Nosotros no vamos a trabajar necesariamente en la bola unidad, pero se tienen resultados similares en otros tipos de conjuntos compactos, ya que la demostración del Teorema 5.1 está basada en los volúmenes del conjunto total y de la parte comprendida sobre H.

5.2. Ejemplo de detección de un determinado individuo

Vamos a ver ahora un ejemplo de funcionamiento de este tipo de neuronas, antes de pasar a desarrollar su proceso de aprendizaje.

Para este ejemplo vamos a utilizar de nuevo el conjunto de datos CIFAR-10, pero en esta ocasión, únicamente vamos a utilizar dos de sus clases, por simplificar. En concreto, vamos a usar las clases "gato" (*cat*) y "coche" (*automobile*). El objetivo de este experimento es construir un modelo capaz de identificar un único individuo de entre otros muchos de la misma especie. En este caso, vamos a tratar de diferenciar un gato concreto de los del conjunto de test, pero esta técnica se podría utilizar para otras aplicaciones, como por ejemplo en seguridad, para encontrar a un cierto individuo mediante una cámara de videovigilancia.

Para ello, vamos a utilizar una red neuronal que genere una representación de las imágenes. Después, usaremos esta representación como entrada de una neurona selectiva que sea capaz de detectar individuos concretos.

Por simplicidad, la red únicamente estará compuesta por una capa convolucional 3×3 de 32 filtros, con su respectiva capa de *max-pooling* 2×2, y por una cabeza formada por una red densa con dos neuronas con función de activación softmax para clasificar entre ambas categorías. Tomaremos como representación, por tanto, la salida de la capa de reducción de dimensionalidad y la de las neuronas densas antes de aplicar la función de activación, lo que da lugar a una entrada de la neurona selectiva de 7202 dimensiones.

Una vez entrenada la red para clasificar gatos y coches con la base de datos CIFAR-10, añadiremos 8 imágenes "caseras" de un gato concreto que no aparece en CIFAR-10. La

Figura 5.3: Imágenes originales del gato añadido al conjunto.

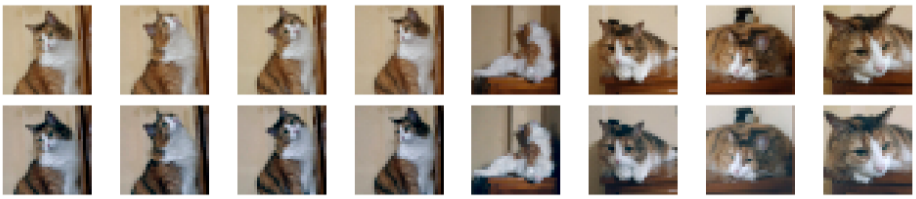


Figura 5.4: Imágenes del gato recortadas y reducidas. Arriba con el brillo y contraste originales y abajo tras ajustarlos a los del conjunto de datos.

Figura 5.3 muestra las imágenes elegidas. Estas imágenes han sido tomadas con un teléfono móvil, por lo que no tienen el tamaño adecuado para la red. Así que el primer paso es reducirlas a 32×32 píxeles. Además, como estos ejemplos son ajenos al conjunto, puede ser que tengan un brillo o un contraste distintos que el resto de las imágenes y que esto provoque resultados erróneos. Para evitarlo, aplicaremos primero un preprocesamiento de las imágenes. Calculamos la media $\bar{\mu}_i$ y la desviación típica $\bar{\sigma}_i$ por cada canal i en el conjunto de entrenamiento. Después, ajustaremos cada imagen añadida X utilizando

$$Y_i = \frac{X_i - \mu_i}{\sigma_i}\bar{\sigma}_i + \bar{\mu}_i,$$

donde μ_i y σ_i son la media y la desviación típica del canal i en la imagen X, respectivamente. Las imágenes resultantes se muestran en la Figura 5.4.

Tras esto, evaluaremos las imágenes con la red para comprobar que las clasifica correctamente y usaremos sus salidas intermedias para generar la entrada de una neurona selectiva. Después, para evitar que la neurona se active "accidentalmente" con estímulos con normas grandes, utilizaremos una normalización L_2, lo que hará que todos los estímulos tengan norma 1. Este proceso lo repetimos en los elementos del conjunto de test de CIFAR-10 que la red clasifica como gatos para tener las mismas condiciones en todas las imágenes.

Finalmente, vamos a "entrenar" una neurona selectiva para reconocer las imágenes de los gatos añadidos, pero ninguna más. Esto lo haremos, utilizando como estímulo ante el que se tiene que activar la media de las representaciones de $n = 4$ ejemplos de entrenamiento:

$$\bar{X} = \frac{1}{n}\sum_{j=1}^{n} X_j.$$

En concreto, hemos escogido para este propósito las dos primeras imágenes y las dos últimas por ser diferentes entre sí, lo que debería ayudar a la neurona a generalizar. Para que la neurona se dispare ante este estímulo es suficiente con tomar

$$w = \frac{\bar{X}}{\|\bar{X}\|} \tag{5.2}$$

como pesos sinápticos y

$$\theta = \min_{1 \le j \le n} \{\langle w, X_j \rangle\} - \varepsilon \tag{5.3}$$

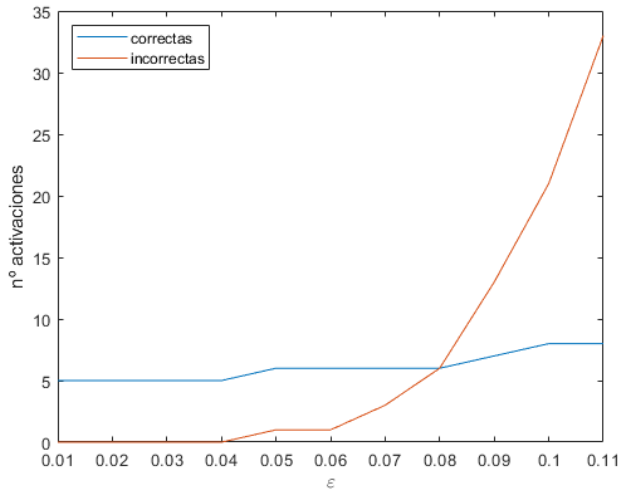


Figura 5.5: Número de activaciones en el individuo que estamos tratando de diferenciar (denominadas correctas) y en el conjunto de test de CIFAR-10 (incorrectas).

como umbral de activación para un cierto $\varepsilon > 0$. De esta forma, para la representación X de una cierta imagen de entrenamiento, se tiene

$$\langle w, X\rangle - \theta \geq \langle w, X\rangle - \langle w, X\rangle + \varepsilon > 0,$$

luego la neurona se activa. Sin embargo, tenemos que elegir un ε suficientemente pequeño para que la neurona no se active con imágenes de CIFAR-10, pero sí lo haga ante otras imágenes de este gato. Para ello, vamos a comprobar el valor que debe tomar ε.

La Figura 5.5 muestra el resultado de este experimento. Para valores pequeños de ε (hasta 0,04) la neurona se activa con las cuatro imágenes de entrenamiento. Sin embargo, consigue activarse con otro de los ejemplos del gato concreto y ninguno de los del conjunto de datos, como buscábamos. A partir de $\varepsilon = 0{,}05$ es capaz de activarse con otro ejemplo, pero también lo hace para uno de CIFAR-10. Después, el umbral ya es bastante alto y el número de ejemplos en los que se activa incorrectamente comienza a crecer exponencialmente. Finalmente, es necesario usar un umbral de $\varepsilon = 0{,}1$ para que la neurona logre reconocer todos los ejemplos de nuestro individuo, pero, a cambio, se activa ante 21 imágenes en las que no debería. En cualquier caso, debemos tener en cuenta que la red neuronal clasificó 997 imágenes como gatos, luego la neurona tan sólo se está activando erróneamente en un 2 % de los ejemplos.

Estos resultados pueden deberse a que la red neuronal es pequeña y no es capaz de abstraer características generales de las imágenes, de forma que los vectores que representan un mismo individuo aparecieran más cercanos en el espacio.

5.3. Regla de Oja

Ahora nuestra misión será encontrar la forma de hallar el vector de pesos sinápticos w de cada neurona para que se alinee con un estímulo y no con los demás. Para ello, vamos a considerar un modelo de aprendizaje distinto al que llevábamos usando hasta ahora. Esta vez el proceso de aprendizaje será continuo, es decir, la neurona se entrenará según reciba un estímulo y no después de un conjunto de ellos. Además, por ser un aprendizaje continuo, cada estímulo se va a mostrar a la neurona durante un tiempo T determinado. Por simplicidad, vamos a usar $T = 1$. Por tanto, ahora todos los elementos que aparecen son funciones del tiempo t.

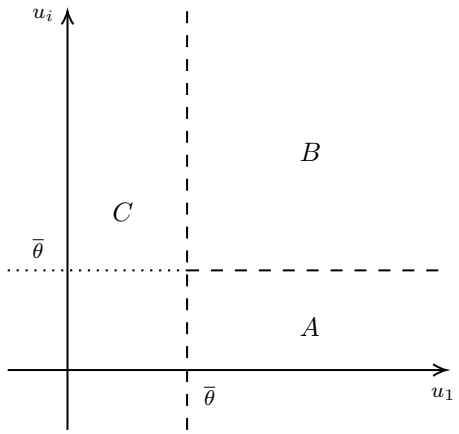


Figura 5.6: Regiones A, B y C a considerar en el plano de fases (u_1, u_i).

Consideremos el proceso de aprendizaje en una neurona. Vamos a utilizar la regla de Oja modificada [28] para actualizar sus pesos sinápticos $w(t) \in \mathbb{R}^n$:

$$\dot{w} = \alpha y(\beta^2 s - vw), \tag{5.4}$$

donde $v(t) = \langle w(t), s(t)\rangle$ es el potencial de la membrana, $y(t) = F(v(t) - \theta)$ es la señal de salida, $s(t) = \sqrt{3/n}\, x_i$, si $t \in [i-1, i)$, con $i \in \{1, \dots, N\}$, es el estímulo mostrado en tiempo t y $\alpha > 0$ y $\beta > 0$ son, respectivamente, la tasa de aprendizaje y un parámetro de orden.

Denotemos ahora

$$u_i(t) = \langle w(t), x_i\rangle \in \mathcal{C}^1((i-1, i)). \tag{5.5}$$

Decimos que la neurona responde a un estímulo x_i si $y(t) > 0$ (o lo que es equivalente, $u_i(t) > \sqrt{n/3}\,\theta$) y que no responde en caso contrario. Notemos que, a diferencia de v, cada u_i hace siempre referencia al mismo estímulo x_i, por lo que para cada $t \in [i-1, i)$, $v(t) = \sqrt{3/n}\, u_i(t)$.

Sin pérdida de generalidad, supongamos que la neurona se activa ante algún estímulo (en caso contrario no habría nada que estudiar) y que la primera activación ocurre en tiempo 0 (siempre podemos reordenar los estímulos para conseguir que esto suceda), es decir, $u_1(0) > \sqrt{n/3}\,\theta$; y denotemos $\bar{\theta} = \sqrt{n/3}\,\theta$. Considerando ahora el plano $(u_1(t), u_i(t))$, $t \in [0, 1)$ en función del valor de los pesos w y para cierto $i \neq 1$, diferenciamos tres regiones (Figura 5.6):

A) La región donde la neurona sólo se activa para x_1,

$$A \coloneqq \{(u_1, u_i) : u_1 > \bar{\theta}, u_i \leq \bar{\theta}\}.$$

B) La región donde la neurona se activa para ambos estímulos,

$$B \coloneqq \{(u_1, u_i) : u_1 > \bar{\theta}, u_i > \bar{\theta}\}.$$

C) La región donde la neurona no se activa para ningún estímulo,

$$C \coloneqq \mathbb{R}^2 \setminus (A \cup B).$$

Veamos ahora la dinámica del sistema si en $t = 0$ $(u_1, u_i) \in B$ a medida que se van actualizando los pesos de la neurona. Utilizando (5.4) y operando se obtiene

$$\dot{u_i} = \frac{3}{n}\alpha F(u_1 - \bar{\theta})(\beta^2\langle x_1, x_i\rangle - u_1 u_i), \quad i \in \{1, \dots, L\}. \tag{5.6}$$

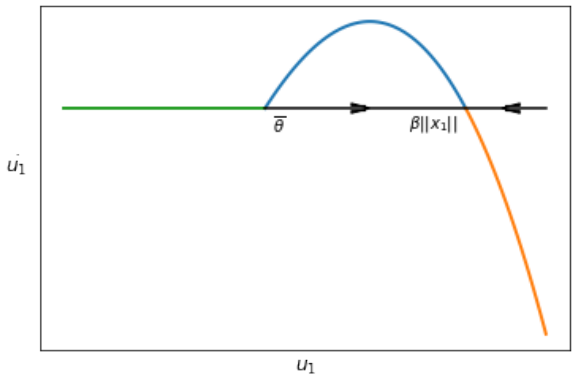


Figura 5.7: Dinámica del problema de valor inicial (5.7).

Proposición 5.2. *Sea* $\beta > \frac{\bar{\theta}}{\|x_1\|}$ *y supongamos* $u_1(0) > \bar{\theta}$*, es decir, la neurona se activa ante el primer estímulo. Entonces:*

a) $u_1(t) > \bar{\theta}$ *para todo* $t \in [0, 1)$*. Es decir, la trayectoria en el plano* (u_1, u_i) *no sale del conjunto* $A \cup B$ *y la neurona sigue detectando el primer estímulo.*

b) *Para* $t > 0$ *la trayectoria se acerca exponencialmente a la recta* $\{(\beta\|x_1\|, c) : c \in \mathbb{R}\}$*. Si* $\alpha \to \infty$*, entonces* $\lim_{t\to 1} u_1(t) = \beta\|x_1\|$*.*

Demostración. Observamos que la dinámica de $u_1(t)$ no depende de u_i ($i \in \{1, \dots, N\}$). Por lo tanto, tenemos el siguiente problema de valor inicial:

$$\begin{cases} \dot{u_1} = \begin{cases} 0 & u_1 \leq \bar{\theta} \\ \frac{3}{n}\alpha(u_1 - \bar{\theta})(\beta^2\|x_1\|^2 - u_1^2) & u_1 > \bar{\theta} \end{cases} \\ u_1(0) = c_0 > \bar{\theta}. \end{cases} \tag{5.7}$$

a) Observamos que el intervalo $(\bar{\theta}, \infty)$ es invariante y tiene un único punto de equilibrio $u_1^* = \beta\|x_1\|$ estable. El signo de $\dot{u_1}$ en $(\bar{\theta}, \infty)$ coincide con el de $\beta^2\|x_1\|^2 - u_1^2$. Por tanto, el problema sigue la dinámica mostrada en la Figura 5.7.

Vemos que si $c_0 \in (\bar{\theta}, \beta\|x_1\|)$, u_1 es creciente, y si $c_0 \in (\beta\|x_1\|, \infty)$ es decreciente, pero siempre se tiene $u_1(t) > \beta\|x_1\| > \bar{\theta}$ para todo $t \in [0, 1)$, como queríamos probar.

b) Cuando $t \to \infty$ es obvio que $u_1(t) \to \beta\|x_1\|$ (las trayectorias tienden al punto de equilibrio). Como por (a) sabemos que $u_1 > \bar{\theta}$, tenemos que

$$\dot{u_1} = \frac{3}{n}\alpha(u_1 - \bar{\theta})(\beta^2\|x_1\|^2 - u_1^2),$$

y veamos los tres posibles casos:

(1) Si $c_0 = \beta\|x_1\|$ nos encontramos en el punto de equilibro, luego el resultado se tiene trivialmente: $u_1(t) = \beta\|x_1\|$.

(2) Si $c_0 \in (\bar{\theta}, \beta\|x_1\|)$, entonces

$$u_1 - \bar{\theta} \geq c_0 - \bar{\theta} > 0 \text{ y } \beta\|x_1\| + u_1 \geq \beta\|x_1\| + c_0 > 0$$

y podemos introducir la subsolución $\underline{u}(t)$ que cumple

$$\begin{cases} \dot{\underline{u}} = \alpha D(\beta\|x_1\| - \underline{u}) \\ \underline{u}(0) = c_0, \end{cases} \tag{5.8}$$

donde $D = 3/n\,(c_0 - \bar{\theta})(\beta\|x_1\| + c_0) > 0$ es una constante.

Para resolver (5.8) tomemos $z = \underline{u} - \beta\|x_1\|$ con $z(0) = c_0 - \beta\|x_1\|$. Entonces $\dot{z} = -\alpha D z$, por lo que podemos escribir $z(t) = z(0)e^{-\alpha D t}$ y concluimos que

$$\underline{u}(t) = \beta\|x_1\| + (c_0 - \beta\|x_1\|)e^{-\alpha D t}.$$

De esta forma, como sabemos que $\underline{u}(t) \leq u_1(t) < \beta\|x_1\|$ para todo $t > 0$, la solución $u_1(t)$ está acotada y además cumple que

$$0 < \beta\|x_1\| - u_1(t) \leq \beta\|x_1\| - \underline{u}(t),$$

luego

$$0 < \beta\|x_1\| - u_1(t) \leq (\beta\|x_1\| - c_0)e^{-\alpha D t} \to 0,$$

por lo que debe ser que $u_1(t) \to \beta\|x_1\|$, al menos exponencialmente, con exponente proporcional a α.

(3) Si $c_0 \in (\beta\|x_1\|, \infty)$, se tiene

$$c_0 - \bar{\theta} \geq u_1 - \bar{\theta} > 0 \text{ y } \beta\|x_1\| + c_0 \geq \beta\|x_1\| + u_1 > 0$$

y podemos introducir la supersolución $\overline{u}(t)$ que cumple

$$\begin{cases} \dot{\overline{u}} = \alpha D(\beta\|x_1\| - \overline{u}) \\ \overline{u}(0) = c_0, \end{cases} \tag{5.9}$$

donde $D > 0$ es la constante definida anteriormente. Como la ecuación es la misma que (5.8), la solución también será la misma:

$$\overline{u}(t) = \beta\|x_1\| + (c_0 - \beta\|x_1\|)e^{-\alpha D t},$$

por lo que tenemos también el resultado anterior por el que $u_1(t) \to \beta\|x_1\|$, exponencialmente.

□

Gracias a esta proposición podemos elegir el hiperparámetro α. Supongamos que queremos que la neurona aprenda un estímulo en su primera presentación (por ejemplo, x_1 durante $t \in [0, 1)$). Entonces podemos definir la tolerancia $\delta > 0$ de forma que $|\beta\|x_1\| - c_0|e^{-\alpha D} < \delta$. Luego, cuando $c_0 \to \bar{\theta}$ se tiene

$$\alpha > \frac{1}{D} \log \frac{|\beta\|x_1\| - \bar{\theta}|}{\delta}.$$

Ya hemos analizado lo que ocurre con u_1 mientras se muestra el estímulo x_1. Ahora vamos a comprobar qué sucede con u_i para $i \neq 1$. Pero antes de continuar vamos a enunciar un teorema que nos va a ser útil más adelante.

Teorema 5.3 (Condición de Lyapunov)**.** *Sea $\{X_k\}_{k=1}^n$ una sucesión de variables aleatorias independientes con esperanzas $\mathbb{E}[X_k] = \mu_k < \infty$ y varianzas $Var(X_k) = \sigma_k^2 < \infty$, y sea $s_n^2 = \sum_{k=1}^n \sigma_k^2$. Si existe $\delta > 0$ tal que*

$$\lim_{n\to\infty} \frac{1}{s_n^{2+\delta}} \sum_{k=1}^{n} \mathbb{E}[|X_k - \mu_k|^{2+\delta}] = 0, \tag{5.10}$$

entonces se tiene, en sentido de distribuciones,

$$\frac{1}{s_n} \sum_{k=1}^{n} (X_k - \mu_k) \xrightarrow{d} \mathcal{N}(0, 1).$$

La ecuación (5.10) define una de las condiciones bajo las que se cumple el Teorema Central del Límite [29], que fue probada por Lyapunov en 1901. Habitualmente será suficiente con tomar $\delta = 1$.

Supongamos entonces que $u_i(0) < \bar{\theta}$, es decir, $(u_1(0), u_i(0)) \in A$. Vamos a comprobar que es posible que $(u_1(t), u_i(t)) \in B$ para cierto $t > 0$, pero con una probabilidad muy baja, la cual podemos acotar superiormente.

Teorema 5.4. *Supongamos $(u_1(0), u_i(0)) \in A$. Entonces la probabilidad de que exista algún $t > 0$ tal que $(u_1(t), u_i(t)) \in B$ viene dada por $P(\xi > 0)$, donde*

$$\xi = \|x_1\|^2 - b^2\langle x_1, x_i\rangle = \sum_{k=1}^{n} x_{1k}^2 - b^2 \sum_{k=1}^{n} x_{1k}x_{ik}. \tag{5.11}$$

Demostración. Sea $\beta = b\beta_0$, con $b > 1$ y $\beta_0 = \bar{\theta}/\|x_1\|$ y consideremos la frontera de A, $\Gamma := \{(u_1, \bar{\theta}) : u_1 > \bar{\theta}\}$. Para que el estado del sistema en el punto (u_1, u_i) cruce de la región A a B es necesario que $\frac{\mathrm{d}u_i}{\mathrm{d}t}\Big|_\Gamma > 0$. Impongamos que la trayectoria no cruce la frontera con B, es decir, que no exista tal $t > 0$. Entonces, tomando la ecuación (5.6) en Γ imponemos

$$\dot{u_i}\big|_\Gamma = \frac{3}{n}\alpha(u_1 - \bar{\theta})(\beta^2\langle x_1, x_i\rangle - \bar{\theta}u_1) < 0.$$

Ya sabemos que $u_1(t) > \bar{\theta} + \varepsilon$, para cierto $\varepsilon > 0$, por tanto,

$$\dot{u_i}\big|_\Gamma < \frac{3\alpha\varepsilon}{n}(\beta^2\langle x_1, x_i\rangle - \bar{\theta}^2),$$

y es suficiente con que $\beta^2\langle x_1, x_i\rangle < \bar{\theta}^2$ para que la derivada sea negativa. Esto significaría que la neurona no se va a activar nunca ante el estímulo x_i, puesto que en Γ el campo vectorial de $(u_1 u_i)$ está dirigido hacia abajo. Reescribiendo, tenemos

$$b^2\frac{\bar{\theta}^2}{\|x_1\|^2}\langle x_1, x_i\rangle < \bar{\theta}^2,$$

luego

$$\xi = \|x_1\|^2 - b^2\langle x_1, x_i\rangle > 0.$$

Por tanto, tenemos que la probabilidad de que ninguna de las trayectorias pueda pasar a B queda definida como $P(\xi > 0)$. □

A partir de ahora, vamos a denotar b a la definida en la demostración anterior: $b = \beta\|x_1\|/\bar{\theta}$. Suponiendo que tenemos una dimensión neuronal n suficientemente grande, queremos ver si podemos aplicar el Teorema 5.3 sobre ξ de forma que tienda a una distribución $\mathcal{N}(\mu, \sigma)$.

Lema 5.5. *Si $x_1, x_i \sim \mathcal{U}([-1, 1]^n)$, la distribución de ξ definida en (5.11) converge a una distribución normal con media $\mu = n/3$ y varianza $\sigma^2 = n(4/45 + b^4/9)$.*

Demostración. Podemos escribir $\xi = \sum_{k=1}^{n} X_k$, con $X_k = x_{1k}^2 - b^2 x_{1k}x_{ik}$. Así, podemos calcular la media μ_k de X_k como

$$\mu_k = \mathbb{E}[X_k] = \sum_{k=1}^{n} \mathbb{E}[x_{1k}^2],$$

ya que $x_{1k}, x_{ik} \sim \mathcal{U}([-1, 1])$ son independientes con media 0. Para evaluar $\mathbb{E}[x_{1k}^2]$, con $k \in \{1, \ldots, n\}$, usamos la función de densidad $p(x_{1k}) = 1/2$:

$$\mu_k = \mathbb{E}[x_{1k}^2] = \int_{-1}^{1} \frac{1}{2}x^2 \,\mathrm{d}x = \frac{x^3}{6}\Big|_{-1}^{1} = \frac{1}{3}.$$

Evaluemos ahora la varianza $\sigma_k^2 = \mathbb{E}[X_k^2] - \mu_k^2$. El segundo término ya lo conocemos, así que desarrollemos el primero.

$$\begin{aligned}\mathbb{E}[X_k^2] &= \mathbb{E}[(x_{1k}^2 - b^2 x_{1k} x_{ik})^2] \\ &= \mathbb{E}[x_{1k}^4] + b^4 \mathbb{E}[x_{1k}^2]\mathbb{E}[x_{ik}^2] \\ &= \frac{1}{5} + \frac{1}{9} b^4,\end{aligned}$$

de nuevo por la independencia entre x_{1k} y x_{ik} y porque

$$\mathbb{E}[x_{1k}^4] = \int_{-1}^{1} \frac{1}{2} x^4 \, \mathrm{d}x = \left.\frac{x^5}{10}\right|_{-1}^{1} = \frac{1}{5}.$$

Por tanto,

$$\sigma_k^2 = \frac{1}{5} + \frac{1}{9} b^4 - \frac{1}{9} = \frac{4}{45} + \frac{1}{9} b^4.$$

Ahora podemos aplicar el Teorema 5.3 con $s_n^2 = n\sigma_k^2$ (para cualquier $k \in \{1, \ldots, n\}$) y $\delta = 1$, pues se cumple que

$$\lim_{n\to\infty} \frac{1}{s_n^3} \sum_{k=1}^{n} \mathbb{E}[|X_k - \mu_k|^3] = \lim_{n\to\infty} \frac{n}{n^{3/2}\sigma_k^3} \mathbb{E}[|X_k - \mu_k|^3] = \lim_{n\to\infty} \frac{1}{\sqrt{n}\sigma_k^3} \mathbb{E}[|X_k - \mu_k|^3] = 0,$$

ya que tanto σ_k^3, como $\mathbb{E}[|X_k - \mu_k|^3]$ son constantes con respecto a n.

Por tanto,

$$\frac{1}{s_n} \sum_{k=1}^{n} (X_k - \mu_k) = \frac{1}{s_n} \sum_{k=1}^{n} \left(X_k - \frac{1}{3}\right) = \frac{\xi - n/3}{s_n} \xrightarrow{d} \mathcal{N}(0, 1),$$

lo que significa que

$$\xi \sim \mathcal{N}\left(\frac{n}{3}, \sqrt{n}\sqrt{\frac{4}{45} + \frac{b^4}{9}}\right).$$

□

En la Figura 5.8 se puede apreciar una comprobación numérica del Lema 5.5. Se puede observar que aun para valores relativamente bajos de n se obtiene una buena aproximación de la distribución esperada.

Una vez que tenemos tanto la media como la desviación típica podemos calcular la probabilidad que estábamos buscando: la probabilidad de que ninguna de las trayectorias pueda pasar a B.

$$P(\xi > 0) = P\left(\frac{\xi - \mu}{\sigma} > \frac{-\mu}{\sigma}\right) = 1 - P\left(Z \leq \frac{-\mu}{\sigma}\right) = \Phi\left(\frac{\mu}{\sigma}\right) = \Phi\left(\sqrt{\frac{n}{4/5 + b^4}}\right),$$

donde $Z \sim \mathcal{N}(0, 1)$. La Figura 5.9 muestra esta probabilidad en función de n. Se puede observar claramente que para valores de b cercanos a 1 y valores relativamente bajos de n se obtiene una probabilidad de 0,95 de que no se traspase la frontera entre las regiones A y B.

Los resultados que hemos obtenido acerca de la probabilidad de que no se atraviese la frontera entre ambas regiones vienen dados únicamente para la pareja de estímulos x_1 y x_i. Pero podemos comprobar qué ocurrirá con los restantes estímulos. Esto viene dado por el siguiente teorema.

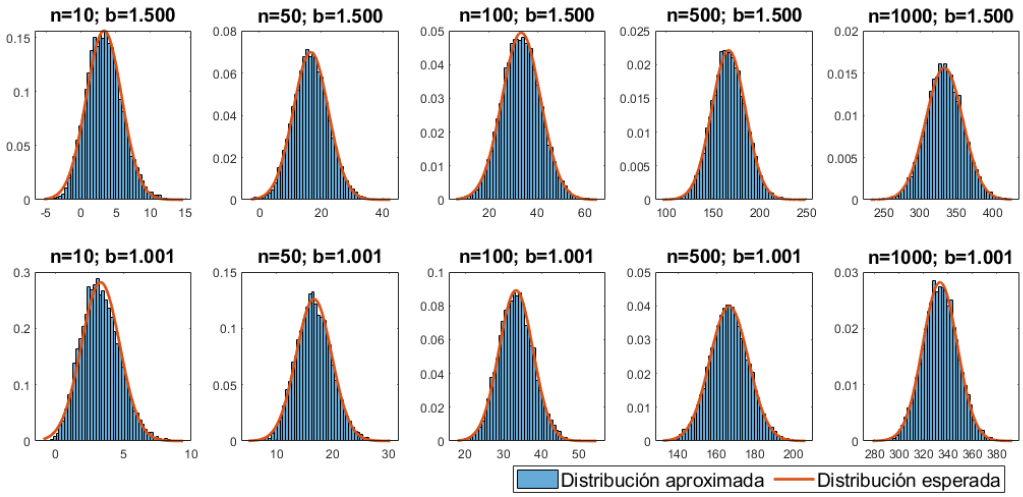


Figura 5.8: Comprobación de que la variable aleatoria ξ converge a una distribución normal con media $\mu = n/3$ y varianza $\sigma^2 = n(4/45 + b^4/9)$ para distintos valores de b y n.

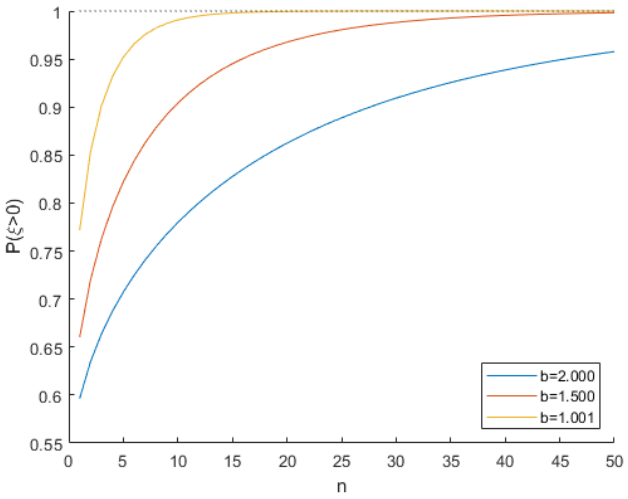


Figura 5.9: Probabilidad de que el estado del sistema en el plano (u_1, u_i) no pase de la región A a B, en función de la dimensión n.

Teorema 5.6. *Supongamos* $(u_1(0), u_{i_j}(0)) \in A$ *para* $i_j \in \{2, \dots, N\}$ *y* $j \in \{1, \dots, \ell\}$, *con* $x_k \sim \mathcal{U}([-1,1]^n)$ *para todo* $k \in \{1, \dots, N\}$. *Entonces la probabilidad* P *de que exista algún* $t > 0$ *tal que* $(u_1(t), u_{i_j}(t)) \in B$ *viene acotada por la siguiente expresión:*

$$P \geq \Phi^{\ell}\left(\sqrt{\frac{n}{4/5 + b^4}}\right). \tag{5.12}$$

Es decir, tras el aprendizaje del primer estímulo la neurona no se excitará en respuesta a ℓ *estímulos subumbrales en* $t = 0$.

La demostración de este teorema se sigue a partir de los resultados mencionados arriba.

Finalmente, podemos concluir que el aprendizaje Hebbiano puede funcionar bien en redes de neuronas de alta dimensión.

Capítulo 6

Conclusiones

A lo largo de este trabajo hemos tratado diferentes tipos de redes neuronales artificiales. En particular, hemos ido estudiando las piezas que las componen, de forma que pudiéramos construir redes cada vez más complejas, y hemos hecho diferentes experimentos para comprobar su funcionamiento.

Partiendo desde el concepto más básico, el perceptrón, hemos sido capaces de construir capas de estos y redes multicapa, las denominadas redes densas o completamente conectadas. Además, hemos estudiado el algoritmo de retropropagación (*backpropagation*), un algoritmo con cierto nivel de complejidad que permite a las redes neuronales aprender de manera automática, en base a una serie de ejemplos de entrenamiento.

Aparte del modelo teórico y el análisis del algoritmo, resultaba interesante comprobar su funcionamiento con ejemplos reales. Esto lo hemos hecho con varios programas en *Matlab*, utilizando sus extensiones para simplificar el trabajo. Hemos obtenido resultados satisfactorios en el primero de ellos, clasificando dígitos escritos a mano, aun no siendo este el campo de aplicación de este tipo de redes neuronales. Sin embargo, con otro ejemplo más complejo, clasificar diferentes animales y vehículos, las redes densas no se comportaban tan bien. Es por eso que se hacía necesario estudiar las redes convolucionales, que son más adecuadas para problemas relacionados con el tratamiento de imágenes.

De nuevo, hemos estudiado el concepto de convolución aplicada sobre imágenes, el elemento más simple que compone las redes convolucionales, para ser capaces de entender el funcionamiento de las mismas. Además, hemos analizado varios tipos de capas que pueden formar parte de una red convolucional. Repitiendo los experimentos anteriores con una red convolucional observamos que, de hecho, reducíamos el error cometido a aproximadamente la mitad utilizando incluso menos pesos sinápticos que en el caso de las redes densas.

Finalmente, hemos presentado un modelo matemático de una neurona artificial que se comporta como las neuronas selectivas en el cerebro. Hemos visto teóricamente que tienen una alta capacidad de separar un punto concreto dentro de un conjunto de estímulos y lo hemos comprobado experimentalmente siendo capaces de reconocer un cierto individuo de entre otros muchos de su especie. Sin embargo, el cálculo de los pesos sinápticos y del umbral de activación lo hemos hecho "a mano" usando los resultados teóricos recientemente publicados en la literatura. Por ello, después hemos realizado un primer acercamiento a un modelo dinámico de aprendizaje que permita que estas neuronas se alineen con un único estímulo y así imitar más fielmente el funcionamiento de las neuronas selectivas biológicas. Este último avance podría ser un problema para estudiar en el futuro, como por ejemplo, en el marco de un TFM.

Bibliografía

[1] N. J. Nilsson. *Artificial intelligence: a new synthesis*. Morgan Kaufmann, 1998.

[2] A. M. Turing. «Computing machinery and intelligence». En: *Parsing the turing test*. Springer, 2009, págs. 23-65.

[3] T. Guardian. *Eugene the Turing test-beating 'human computer' – in 'his' own words*. [Online; último acceso el 22 de junio de 2021]. URL: https://www.theguardian.com/technology/2014/jun/09/eugene-person-human-computer-robot-chat-turing-test.

[4] J. Redmon, S. Divvala, R. Girshick *et al.* «You only look once: Unified, real-time object detection». En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 779-788.

[5] D. Silver, T. Hubert, J. Schrittwieser *et al.* «A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play». En: *Science* 362.6419 (2018), págs. 1140-1144.

[6] F. Rosenblatt. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. Inf. téc. Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[7] R. Q. Quiroga, L. Reddy, G. Kreiman *et al.* «Invariant visual representation by single neurons in the human brain». En: *Nature* 435.7045 (2005), págs. 1102-1107.

[8] F. A. Azevedo, L. R. Carvalho, L. T. Grinberg *et al.* «Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain». En: *Journal of Comparative Neurology* 513.5 (2009), págs. 532-541.

[9] Wikimedia Commons. *Neurona*. [Online; último acceso el 7 de junio de 2021]. 2007. URL: https://commons.wikimedia.org/wiki/File:Neurona.svg.

[10] F. Rosenblatt. «The perceptron: a probabilistic model for information storage and organization in the brain». En: *Psychological review* 65.6 (1958), pág. 386.

[11] W. S. McCulloch y W. Pitts. «A logical calculus of the ideas immanent in nervous activity». En: *The bulletin of mathematical biophysics* 5.4 (1943), págs. 115-133.

[12] B. Kröse y P. van der Smagt. «An introduction to neural networks». En: (1993).

[13] D. E. Rumelhart, G. E. Hinton y R. J. Williams. «Learning representations by back-propagating errors». En: *nature* 323.6088 (1986), págs. 533-536.

[14] D. Graupe. *Principles of artificial neural networks*. Vol. 7. World Scientific, 2013.

[15] R. Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[16] *MATLAB version 9.10.0.1602886 (R2021a)*. The Mathworks, Inc. Natick, Massachusetts, 2021.

[17] *Deep Learning Toolbox version 14.2*. The Mathworks, Inc. Natick, Massachusetts, 2021.

[18] Y. LeCun, C. Cortes y C. Burges. «MNIST handwritten digit database». En: *ATT Labs [Online; último acceso el 29 de junio de 2021]*. 2 (2010). URL: http://yann.lecun.com/exdb/mnist.

[19] A. Krizhevsky, G. Hinton *et al. Learning multiple layers of features from tiny images*. Inf. téc. Canadian Institute For Advanced Research, 2009.

[20] I. I. Hirschman y D. V. Widder. *The convolution transform*. Courier Corporation, 2012.

[21] K. Simonyan y A. Zisserman. «Very deep convolutional networks for large-scale image recognition». En: *arXiv preprint arXiv:1409.1556* (2014).

[22] A. Bauerle, C. Van Onzenoodt y T. Ropinski. «Net2Vis-A Visual Grammar for Automatically Generating Publication-Ready CNN Architecture Visualizations». En: *IEEE transactions on visualization and computer graphics* (2021).

[23] J. MacQueen *et al.* «Some methods for classification and analysis of multivariate observations». En: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, págs. 281-297.

[24] D. O. Hebb. «The organization of behavior; a neuropsycholocigal theory». En: *A Wiley Book in Clinical Psychology* 62 (1949), pág. 78.

[25] C. C. Tapia, I. Tyukin y V. A. Makarov. «Universal principles justify the existence of concept cells». En: *Scientific reports* 10.1 (2020), págs. 1-9.

[26] D. L. Donoho *et al.* «High-dimensional data analysis: The curses and blessings of dimensionality». En: *AMS math challenges lecture* 1.2000 (2000), pág. 32.

[27] F. W. Olver, D. W. Lozier, R. F. Boisvert *et al. NIST handbook of mathematical functions hardback and CD-ROM*. Cambridge university press, 2010.

[28] E. Oja. «Simplified neuron model as a principal component analyzer». En: *Journal of mathematical biology* 15.3 (1982), págs. 267-273.

[29] P. Billingsley. *Probability and Measure*. Tercera edición. John Wiley y Sons, 1986, pág. 362.

[30] J. McGonagle, G. Shaikouski, C. Williams *et al. Backpropagation*. [Online; último acceso el 26 de marzo de 2021]. URL: https://brilliant.org/wiki/backpropagation/.

[31] M. Nielsen. *How the backpropagation algorithm works*. [Online; último acceso el 3 de mayo de 2021]. URL: http://neuralnetworksanddeeplearning.com/chap2.html.

[32] F. Chollet *et al. Deep learning with Python*. Vol. 361. Manning New York, 2018.