

# Title of Document

Name of Author

April 5, 2022

## 1 notas

capa densa capa oculta ap. supervisado clasificacion  
crear red python keras: <https://youtu.be/aFZEvQDTSyA?t=616>  
perceptron: unica neurona cuyo input es un vector binario y output es un numero binario. su funcion de activacion es la step function <https://es.wikipedia.org/wiki/Perceptr%C3%B3n> [https://wikimedia.org/api/rest\\_v1/media/math/render/svg/50f2b5077f8fa933c912c6ca0571d6c7d3709d83](https://wikimedia.org/api/rest_v1/media/math/render/svg/50f2b5077f8fa933c912c6ca0571d6c7d3709d83)  
umbral o sesgo o threshold  
perceptron multicapa funcion de activacion

## 2 Redes Neuronales

Formalmente una red neuronal es un modelo de deeplearning. El deeplearning (o también conocido como aprendizaje profundo) es la rama específica del aprendizaje automático (o machine learning) que usa redes neuronales.

### 2.1 Perceptrón

Un perceptrón (o neurona) es un modelo de deeplearning que recibe como input variables binarias  $x_1, \dots, x_n$  y tiene como output una única variable binaria  $y$ . A su vez, esta función depende de los parámetros reales  $\omega_1, \dots, \omega_n, u$ . A los parámetros  $\omega_i$  se les conoce como pesos y al parámetro  $u$  se le conoce como umbral (o sesgo o threshold o bias)<sup>1</sup>.

El perceptrón está definido por la siguiente función

$$y = \begin{cases} 0, & \text{si } \sum_{i=1}^n w_i x_i - u \leq 0 \\ 1, & \text{si } \sum_{i=1}^n w_i x_i - u > 0 \end{cases}$$

---

<sup>1</sup><http://neuralnetworksanddeeplearning.com/chap1.html>

El perceptrón sirve como modelo para toma de decisiones basado en otros hechos. Por ejemplo puede modelizar la siguiente toma de decisiones. Consideremos el siguiente escenario<sup>2</sup>:

- $y$  = Irnos de viaje
- $x_1$  = Tengo suficiente dinero?
- $x_2$  = Mi pareja quiere ir?
- $x_3$  = Hará buen tiempo?

### Introducir grafico

El perceptrón modela una posible toma de decisiones a la hora de decidir si irnos de viaje o no y de él se puede sacar su tabla de la verdad.

Sin embargo, un perceptrón no es capaz de modelizar cualquier toma de decisión (tabla de la verdad). Por ejemplo no es capaz de modelizar la operación lógica XOR.

Nótese que si fijamos  $w_1, \dots, w_n, u \in \mathbb{R}$ , entonces

$$H := \left\{ (x_1, \dots, x_n) \in \mathbb{R}^n \mid \sum_{i=1}^n w_i x_i - u = 0 \right\}$$

define un hiperplano de  $\mathbb{R}^n$ . Desde un punto de vista geométrico, este hiperplano es el separador entre  $\{x \in \{0, 1\}^n \mid y(x) = 0\}$  y  $\{x \in \{0, 1\}^n \mid y(x) = 1\}$ . Es decir, que el perceptrón únicamente puede hacer una separación lineal (o afín) del conjunto de puntos  $\{0, 1\}^n$ .

Como consecuencia, un perceptrón no puede reproducir la operación lógica XOR.

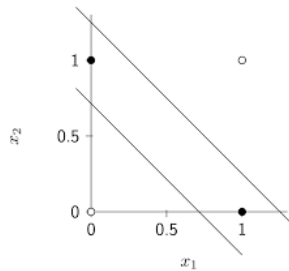


Figure 1: Visualización de la separación no lineal de la puerta XOR

<sup>2</sup><https://youtu.be/CU24iC3grq8?t=236>

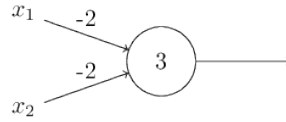


Figure 2: Red neuronal modelizando una puerta NAND

En la figura 2 se muestra la representación de un perceptrón que reproduce una puerta lógica NAND. Esta es una red con dos capas: la primera contiene dos neuronas y la segunda solo una con un umbral igual a 3. Además observamos que los pesos de las dos únicas conexiones son iguales a  $-2$ . En este caso la función de activación de la neurona de la segunda capa es

$$f(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{si } x \leq 0 \end{cases}.$$

En efecto, como se puede ver en la figura 3, la red reproduce la tabla de la verdad de la puerta lógica NAND.

$x_1$	$x_2$	output
0	0	$f(0 \cdot (-2) + 0 \cdot (-2) + 3) = f(3) = 1$
0	1	$f(0 \cdot (-2) + 1 \cdot (-2) + 3) = f(1) = 1$
1	0	$f(1 \cdot (-2) + 0 \cdot (-2) + 3) = f(1) = 1$
1	1	$f(1 \cdot (-2) + 1 \cdot (-2) + 3) = f(-1) = 0$

Figure 3: Tabla de la verdad del perceptrón representado en la figura 2

## 2.2 Perceptrón multicapa

Las puertas NAND son universales para la computación, es decir, se puede conseguir cualquier puerta lógica como combinaciones de estas<sup>3</sup>.

## 3 Redes neuronales convolucionales

Usar una red neuronal para reconocer dígitos escritos a mano es posible pero no es lo mejor. La red recibiría los píxeles como un vector, perdiendo así la información sobre la posición relativa de los píxeles de la imagen. Es natural pensar que esta información es vital para un correcto reconocimiento.

La red, con tal de "detectar ejes o líneas", debería aprender por sí sola que relación hay entre los inputs (intensidad de los píxeles) que esta recibe.

<sup>3</sup><http://neuralnetworksanddeeplearning.com/chap1.html>

## 3.1 Convolución

### 3.1.1 Filtros

Un kernel es una matriz cuadrada  $K$  con coeficientes reales. Los kernels tienen una dimensión impar, de modo que se puede hablar de su píxel principal (central)<sup>4</sup>.

Un filtro es un tensor de tres dimensiones cuyas "capas" son kernels<sup>5</sup>. Es posible que un filtro contenga un único kernel, de modo que a veces estas palabras se usan indistintamente.

Hay distintos filtros destacables, por ejemplo:

- Filtro de desenfoque (o blur) 3x3:  $\begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$
- Filtro vertical 3x3:  $\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$
- Filtro horizontal 3x3:  $\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{pmatrix}$
- Edge detector 3x3:  $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

## 3.2 Padding

Dada una imagen  $I$  con  $c$  canales de resolución  $n \times n$  y una cantidad entera  $p$ , se conoce como hacer padding (o rellenar) con ceros a la imagen  $I$  a la operación cuyo output es la nueva imagen  $I'$  con  $c$  canales de dimensión  $n + 2p \times n + 2p$  definida como

$$I'[i, j, k] = \begin{cases} I[i - p, j - p, k] & p \leq i, j \leq n + p \\ 0 & \text{En caso contrario} \end{cases}$$

### 3.2.1 Operación de convolución

Dada una imagen  $I$  con  $c$  canales de resolución  $n \times n$  y un filtro  $F$  formado por  $c$  kernels de dimensión  $m \times m$  con  $m < n$ , la operación de convolución devuelve una imagen  $I'$  con un canal (o feature map) de dimensión, digamos,  $o \times o$ .

---

<sup>4</sup><https://datascience.stackexchange.com/questions/23183/why-convolutions-always-use-odd-numbers-as-filter-size>

<sup>5</sup><https://stats.stackexchange.com/questions/154798/difference-between-kernel-and-filter-in-cnn>

Sean  $i$  y  $j$  tales que

$$\frac{m-1}{2} \leq i, j < n - \frac{m-1}{2}.$$

El elemento  $i, j$  de  $I'$  viene definido<sup>6</sup> por

$$I'[i, j] = \sum_{\alpha=1}^c \sum_{\beta=-\frac{m-1}{2}}^{\frac{m-1}{2}} \sum_{\gamma=-\frac{m-1}{2}}^{\frac{m-1}{2}} I[i + \beta, j + \gamma, \alpha] \cdot F\left[\alpha + \frac{m-1}{2}, \beta + \frac{m-1}{2}\right]$$

$$I'[i, j] = \sum_{\alpha=1}^c \sum_{\beta} \sum_{\gamma} I[i + \beta, j + \gamma, \alpha] F[\beta, \gamma, \alpha]$$

Por lo tanto tenemos que  $o = n - m + 1$ .

A grosso modo, la convolución genera una nueva matriz conocida como feature map o mapa de características mediante productos escalares de  $F$  y submatrices de  $M$ .

Haz click aquí o aquí para ver animaciones de esta operación<sup>7</sup>.

una alternativa sería pasar a markdown para ver

Inicialmente se le hace un padding (rodear con ceros) la imagen  $M$  con un número  $p$  de líneas. De este modo,  $M$  pasa de tener dimensiones  $n \times n$  a  $(n + 2p) \times (n + 2p)$  (se añade una fila/columna por todos los lados).

A continuación se fija el stride  $s$ , es decir, la cantidad de píxeles que se desplazará horizontalmente el filtro. Finalmente se genera una nueva matriz  $M'$  donde sus entradas son productos componente a componente del filtro  $F$  y submatrices de  $M$ .

En particular la  $M'$  tendrá dimensiones  $o \times o$  donde

$$o = \left\lfloor \frac{n + 2p - m}{s} \right\rfloor + 1$$

NOTA: si la imagen fuera a color, el kernel realmente sería de  $3 \times 3 \times 3$  es decir: un filtro con 3 kernels de  $3 \times 3$ ; luego esos 3 filtros se suman (y se le suma una unidad bias) y conformarán 1 salida (cómo si fuera 1 solo canal)<sup>8</sup>.

En realidad, no aplicaremos un solo kernel. Por ejemplo, en esta primera convolución podríamos tener 32 kernels, con lo cual realmente obtendremos 32 matrices de salida con dimensión  $28 \times 28 \times 1$ , dando un total del 25.088 neuronas

<sup>6</sup>Inspirado en <https://ichi.pro/es/matematicas-de-las-redes-neuronales-convolucionales-30697178023872>

<sup>7</sup>fuelle de los gifs: [https://juansensio.com/blog/042\\_cnns](https://juansensio.com/blog/042_cnns)

<sup>8</sup><https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador>

para nuestra PRIMER CAPA OCULTA de neuronas. Parecen muchas neuronas para una imagen cuadrada de apenas 28 pixeles. El número aumentaría drásticamente si tomáramos una imagen de entrada de 224x224x3 (que aún es considerado un tamaño pequeño)<sup>9</sup>.

Cuando tratamos con imágenes con más de un canal (por ejemplo rgb), los filtros deben tener el mismo número de canales (kernels?). De modo que cada kernel hace una convolución con su canal correspondiente. Posteriormente se cogen todas esos outputs y se hace la media. De este modo, tras convolucionar el filtro con la imagen de más de un canal, se sigue teniendo una imagen con un único canal como output. En consecuencia, la cantidad de imágenes de output será igual a la cantidad de filtros que tenga esa capa convolucional. Finalmente a cada una de estas imágenes se les suma un cierto umbral y se aplica la ReLU.<sup>10</sup>

Nota: por lo que veo en la librería de tensorflow, cada kernel del filtro tiene su propio bias. tras aplicar el bias se hace la media.<sup>11</sup>

### 3.3 Implementación

Una de las redes más sencillas es la LeNET-5<sup>1213</sup>

## 4 enlaces de interes

- <https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/> - ilustraciones muy chulas y utiles
- webs para ayudar a redactar la seccion:
  - <https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>
  - <https://www.youtube.com/playlist?list=PLv8Cp2NvcY8DpVcsmOT71kymgMmcr59Mf> - lista de reproduccion de un canal que habla sobre las arquitecturas de redes conocidas
  - <https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8>
- <https://www.youtube.com/watch?v=V8j1oENVz00> - dotcsv (HECHO)
- <https://www.youtube.com/watch?v=ysqpl6w6Wzg> - dotcsv (parte 2) (interesante pero no creo que sea necesario meterlo aqui)

---

<sup>9</sup><https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

<sup>10</sup><https://www.youtube.com/watch?v=HEH1clGg8o8&list=PLdxQ7SoCLQANQ9fQcJ0wnnTzkFsJHlWEj&index=50>

<sup>11</sup><https://www.youtube.com/watch?v=S5MwKyYYLSM&list=PLdxQ7SoCLQANQ9fQcJ0wnnTzkFsJHlWEj&index=51>

<sup>12</sup><https://www.youtube.com/watch?v=HEH1clGg8o8&list=PLdxQ7SoCLQANQ9fQcJ0wnnTzkFsJHlWEj&index=50>

<sup>13</sup><https://en.wikipedia.org/wiki/LeNet>

- [https://www.youtube.com/watch?v=\\_fDvfGxwW20](https://www.youtube.com/watch?v=_fDvfGxwW20) - sensio (FALTA)
- [https://www.youtube.com/watch?v=AwTH\\_0yW9\\_I](https://www.youtube.com/watch?v=AwTH_0yW9_I) - Ringa Tech (FALTA)
- [https://www.youtube.com/watch?v=0zbhg79i\\_Bs](https://www.youtube.com/watch?v=0zbhg79i_Bs) - indio cnn desde 0 (OPCIONAL CREO)
- [https://www.youtube.com/watch?v=AwTH\\_0yW9\\_I&t=584s](https://www.youtube.com/watch?v=AwTH_0yW9_I&t=584s) - operadores importantes: operador Sobel y Algoritmo de Canny (OPCIONAL)

## 4.1 Max-Pooling

Si hiciéramos una nueva convolución a partir de esta capa, el número de neuronas de la próxima capa se iría por las nubes (y ello implica mayor procesamiento)! Para reducir el tamaño de la próxima capa de neuronas haremos un proceso de subsampling en el que reduciremos el tamaño de nuestras imágenes filtradas pero en donde deberán prevalecer las características más importantes que detectó cada filtro. Hay diversos tipos de subsampling, yo comentaré el “más usado”: Max-Pooling<sup>14</sup>

Estas capas también aplican un filtro sobre su entrada, pero en este caso es un solo filtro que además no tiene parámetros sino que aplica una función pre-determinada en su campo receptivo (mínimo, máximo (maxpooling), promedio (average pooling o avgpooling, etc)<sup>15</sup>.

Haz click aquí para ver un gif sobre el max-pooling.

## 5 Generalidades sobre las imágenes

### 5.1 Input de las redes neuronales

La arquitectura de una red neuronal podría adaptarse para tener como input una imagen con dimensiones cualesquiera.

Sin embargo, normalmente se deciden diseñar las redes "para uso público" y para recibir imágenes cuadradas y que cada uno haga el reshape correspondiente. No hay ninguna limitación en las posibles estructuras de las CNNs que limiten la forma del input.<sup>16</sup>

## 6 Redes convolucionales

En realidad la seccion deberia tener estas subsecciones aunque quizas en otro ordenador (por ejemplo la itepretacion/idea deberia ir primero)

<sup>14</sup><https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/>

<sup>15</sup>[https://juansensio.com/blog/042\\_cnns](https://juansensio.com/blog/042_cnns)

<sup>16</sup><https://ai.stackexchange.com/questions/8323/how-to-handle-rectangular-images-in-convolutional-neural-networks>

**6.1 definicion formal, funcionamiento**

**6.2 visualizacion de la convolución**

**6.3 interpretacion de la convolucion**

filtros lineas verticales etc

**6.4 representacion grafica de una capa convolucional/redes convolucionales**

un cuadrado mas o menos gordito etc

**6.5 backpropagation**