Title of Document

Name of Author

February 18, 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`
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 $x_1, \ldots, x_n$ y tiene como output una única variable binaria y. A su vez, esta función depende de los parámetros reales $\omega_1, \ldots, \omega_n, u$. A los parámetros ω_i se les conoce como pesos y al parámetro u se le conoce como umbral (o sesgo o threshold o bias)[1].

El perceptrón está definido por la siguiente función

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

[1] `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[2]:

- 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 $w_1, \ldots, w_n, u \in \mathbb{R}$, entonces

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

define un hiperplano de $\mathbb{R}^n$. Desde un punto de vista geométrico, este hiperplano es el separador entre $\{x \in \{0,1\}^n \mid y(x) = 1\}$ 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.

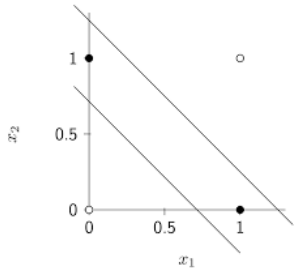

Figure 1: Visualizacación de la separación no lineal de la puerta XOR

[2] https://youtu.be/CU24iC3grq8?t=236

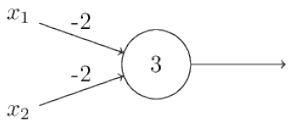

Figure 2: Red neuronal modelizando una puerta NAND

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[3].

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 píxles de la imagen. Es natural pensar que esta información es vital para un correcto reconocimiento.

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

[3] `http://neuralnetworksanddeeplearning.com/chap1.html`

3.1 Convolución

3.1.1 Filtros

Parece ser que un filtro es el conjunto de kernels de una misma convolución: https://stats.stackexchange.com/questions/154798/difference-between-kernel-and-filter-in-cn
Un filtro (o kernel) es una matriz cuadrada K con coeficientes reales. 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: $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

Nota: Los filtros suelen ser únicamente de dimension impar. Esto se debe a que de este modo siempre hay un píxel central en el kernel y así podemos saber a qué pixel del mapa de características corresponde. Dicho de otra manera, el kernel tiene que tener un pixel central del cual "se extraen sus características".[4]

3.1.2 Operación de convolución

Dada una matriz cuadrada (porque no rectangular?) M (una imagen en blanco y negro) y un filtro F, la operación de convolución devuelve una imagen M'.

Nota: es necesario que todos los inputs de las redes tengan un tamaño idéntido. Normalmente, se deciden diseñar las redes "para uso público" 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.[5]

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.

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

[5] https://ai.stackexchange.com/questions/8323/how-to-handle-rectangular-images-in-convolutional-neural-networks

Haz click aquí o aquí para ver animaciones de esta operación[6].
una alternativa seria pasar a markdown para ver

Inicialmente se le hace un padding (rodear con ceros) la imagen M con un número p de lineas. 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 producntos 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 $3x3x3$ es decir: un filtro con 3 kernels de 3×3; luego esos 3 filtros se suman (y se le suma una unidad bias) y conformarán 1 salida (cómo si fuera 1 solo canal)[7].

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 28x28x1, dando un total del 25.088 neuronas 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)[8].

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 imagenes de output será igual a la cantidad de filtros que tenga esa capa convolusional. Finalmente a cada una de estas imágenes se les suma un cierto humbral y se aplica la ReLU.[9]

Nota: por lo que veo en la libreria de tensorflow, cada kernel del filtro tiene su propio bias. tras aplicar el bias se hace la media.[10]

[6] fuente de los gifs: `https://juansensio.com/blog/042_cnns`

[7] `https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/`

[8] `https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/`

[9] `https://www.youtube.com/watch?v=HEH1clGg8o8&list=PLdxQ7SoCLQANQ9fQcJ0wnnTzkFsJHlWEj&index=50`

[10] `https://www.youtube.com/watch?v=S5MwKyYYLSM&list=PLdxQ7SoCLQANQ9fQcJ0wnnTzkFsJHlWEj&index=51`

3.2 Implementación

Una de las redes más sencillas es la LeNET-5[11][12]

4 enlaces de interes

- https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduc - ilustraciones muy chulas y utiles
- webs para ayudar a redactar la seccion:
  - https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-net
  - https://www.youtube.com/playlist?list=PLv8Cp2NvcY8DpVcsmOT71kymgMmcr59Mf - lista de reproduccion de un canal que habla sobre las arquitecturas de deres 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)
- https://www.youtube.com/watch?v=_fDvfGxwW20 - sensio (FALTA)
- https://www.youtube.com/watch?v=AwTH_0yW9_I - Ringa Tech (FALTA)
- 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 - 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[13]

[11] https://www.youtube.com/watch?v=HEH1clGg8o8&list=PLdxQ7SoCLQANQ9fQcJ0wnnTzkFsJHlWEj&index=50

[12] https://en.wikipedia.org/wiki/LeNet

[13] https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

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 predeterminada en su campo receptivo (mínimo, máximo (maxpooling), promedio (average pooling o avgpooling, etc)[14].

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

5 Generalidades sobre las imágenes

[14] https://juansensio.com/blog/042_cnns