Title of Document

Name of Author

June 26, 2023

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 matemático. 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 tomar 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

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

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 $\{x \in \{0,1\}^n \mid y(x) > 0\}$. Es decir, que el perceptrón únicamente puede hacer una separación lineal (o afín) del conjunto de puntos $\{0,1\}^n$.

En particular, un perceptrón no puede reproducir la operación lógica XOR.

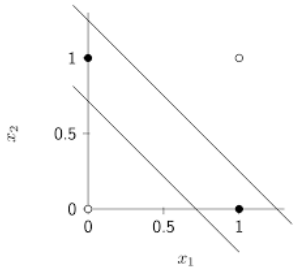

Figure 1: Visualizacación de la separación no lineal de la puerta XOR

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

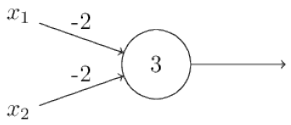

Figure 2: Perceptron modelizando una puerta NAND

En la figura 2 se muestra una representación gráfica 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`

Otro aspecto a mejorar es la invarianza por traslación[4]. Una red entrenada para reconocer números centrados en la imagen no sería capaz de reconocerlos si estos están a un lado de la imagen (no es capaz de generalizaar las formas y depende masivamente de la posición).

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 elemento principal o (central)[5].

Un filtro es un tensor de más de una dimensión cuyas "capas" son kernels[6]. 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: $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

3.1.2 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, que denotaremos con $Pad(I, p)$, compuesta por c canales de dimensión $n + 2p \times n + 2p$ definida como

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

[4] https://www.youtube.com/watch?v=dAetIuYz46g

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

[6] https://stats.stackexchange.com/questions/154798/difference-between-kernel-and-filter-in-cnn

3.1.3 Operación de convolución/correlación

Nota: A continuación se va a definir la operación de correlación. En el ámbito de redes neuronales, se le suele llamar convolución. Esto se debe a que la única diferencia entre una correlación y una convolución es un flip de una de las entradas y en el caso de las redes, la red aprenderia a clasificar bien con los pesos "flipeados"[7].
Sea I una imagen con c canales de resolución $n \times n$ y sea F un filtro formado por c kernels de dimensión $m \times m$ con $m < n$. Sean $p, s \in \mathbb{N}$. La operación de convolución de I y F con un stride s y padding p es una imagen $Conv(I, F, p, s)$ con un único canal (o feature map) de dimensión, digamos, $o \times o$.

El elemento i, j de $Conv(I, F, p, s)$ viene definido[8] por

$$Conv(I, F, p, s)[i, j] := \sum_{\alpha=0}^{c} \sum_{\beta=0}^{m} \sum_{\gamma=0}^{m} Pad(I, p)[i \cdot s + \beta, j \cdot s + \gamma, \alpha] \cdot F[\beta, \gamma, \alpha]$$

donde

$$0 \leq i \cdot s + \beta < n + 2p \quad \forall i, \beta$$

y

$$0 \leq j \cdot s + \gamma < n + 2p \quad \forall j, \gamma.$$

Por lo tanto tenemos que

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

Nótese que si $n+2p-m$ no es múltiplo de s, entonces habrán filas y columnas de la imagen que se saltarán. Además, los píxeles de las esquinas y bordes tienen menor importancia en el cálculo. Este es el motivo por el cual se introduce el padding: Añadir filas y columnas redundates para que al convolución se haga con todos los píxeles de la imagen por igual.

Nota: Los frameworks tensorflow y keras admiten como padding la opción 'same' que quiere decir que se ajustará el padding adecuado para que tras la convolución no se pierda tamaño[9]

3.1.4 visualización de la operación de convolución

Haz click aquí, aquí o aquí para ver animaciones de esta operación[10].
una alternativa seria pasar a markdown para ver

[7] `https://youtu.be/cUa3Jug3TiA?list=PLWzLQn_hxe6ZlC9-YMt3nN0Eo-ZpOJuXd&t=590`
[8] Inspirado en `https://ichi.pro/es/matematicas-de-las-redes-neuronales-convolucionales-30697178023872` y a su vez traducido de `https://towardsdatascience.com/convolutional-neural-networks-mathematics-1beb3e6447c0`
[9] `https://youtu.be/l16RxAmP9QE?list=PLdxQ7SoCLQANQ9fQcJ0wnnTzkFsJHlWEj&t=1376`
[10] fuente de los gifs: `https://juansensio.com/blog/042_cnns`

3.1.5 interpretación de la convolución

3.1.6 Estructura de la capa de convolución

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)[11].

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)[12].

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.[13]

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.[14]
Nota: Por lo general se suele escoger que la cantidad de filtros sea una potencia de 2 por algún tipo de cuestión de eficiencia[15]

3.2 Implementación

Una de las redes más sencillas es la LeNET-5[16][17]

sensio -arquitecturas de cnns famosas: `https://www.youtube.com/watch?v=IKB8rzGU8Wo`

[11] `https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/`
[12] `https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/`
[13] `https://www.youtube.com/watch?v=HEH1clGg8o8&list=PLdxQ7SoCLQANQ9fQcJOwnnTzkFsJHlWEj&index=50`
[14] `https://www.youtube.com/watch?v=S5MwKyYYLSM&list=PLdxQ7SoCLQANQ9fQcJOwnnTzkFsJHlWEj&index=51`
[15] `https://youtu.be/XkqgTaWle0I?list=PLWzLQn_hxe6ZlC9-YMt3nNOEo-ZpOJuXd&t=61`
[16] `https://www.youtube.com/watch?v=HEH1clGg8o8&list=PLdxQ7SoCLQANQ9fQcJOwnnTzkFsJHlWEj&index=50`
[17] `https://en.wikipedia.org/wiki/LeNet`

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 (VISTO, TODO) (interesante pero no creo que sea necesario meterlo aqui). El video empieza a partir del min 6:20. Hbala sobre comolos filtros detectan cada vez cosas más complejas y como intepretar cada unon delos filtros. Seria interesante reproducir eso. Toda esta visualizacióon la hace con "l microscopio de openAI": https://microscope.openai.com/models
- https://www.youtube.com/watch?v=_fDvfGxwW20 - sensio (VISTO, TODO) - interesante a partir del min 14:30 donde muestra una implementacionde una cnn usando pytorch. tambien son interesantes algunos gifs que ya he sacado de su web
- https://www.youtube.com/watch?v=AwTH_0yW9_I - Ringa Tech (HECHO)
- 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)
- https://stackoverflow.com/questions/67409505/python-2d-convolution-optimization - curiosa implementacion en python de una visualizacion ascii de la convolucion
- https://www.youtube.com/watch?v=OYXkoZXEZc4&list=PLkgbkukKg_Nr-CAmU1yGbhJ8UELVDJ07I - sensio (FALTA) - lista de reproduccion en la que parece que implementa unlocalizador de objetos y segmentador de imagenes desde cero. seguramente lo hagaa partir de torch pero no esta mal

4.1 Max-Pooling (deberia ser poolig en general porque tambien hay average 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[18]

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)[19].

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

5 Generalidades sobre las imágenes

5.1 Canales de las imágenes

Las imágenes en blanco y negro se pueden representar con un único canal, es deicr, con una matriz de enteros en el rango $[0, 255]$ o floats en el rango $[0, 1]$.

Si la imagen es de color, entonces se representan mediante los tres canales RGB. De modo que una imagen en color se representa mediante un tensor (martriz) tridimensional.

En general una "imagen" podría tener tantos canales como se quisiera. Por ejemplo las redes neuronales convolucionales producen imagenes con muchos canales.

5.1.1 Orden de los canales de las imágenes

No hay un consenso en el orden de los indices que deben representar la altura, anchura y canales de una imagen. Por ejemplo una imagen con c canales, una altura h y una anchura w podría representarse de las dos siguientes formas[20].

1. Channels First: Tensor de dimensiones $c \times h \times w$ (Como por ejemplo pytorch)
2. Channels Last: Tensor de dimensiones $h \times w \times c$ (como por ejemplo tensorflow y numpy)

La manera más rápida de pasar de un formato a otro (permutar los ejes) es usando numpy.transpose[21]

[18] https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/
[19] https://juansensio.com/blog/042_cnns
[20] https://machinelearningmastery.com/a-gentle-introduction-to-channels-first-and-channels-last-image-formats-f
[21] https://stackoverflow.com/questions/43829711/what-is-the-correct-way-to-change-image-channel-ordering-betwee

5.2 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.[22]

6 Redes convolucionales

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

6.1 Capa de convolución

6.1.1 Filtros

6.1.2 padding

6.1.3 definicion forma/funcionamiento de la operacion de convolución

6.1.4 visualizacion de la convolución

aqui deberian ir gifs y visualizacion del stride etc

6.1.5 interpretacion de la convolucion

filtros lineas verticales etc. estaria bien preparar un script con algun ejemplo real para ver que pasa

6.1.6 Estructura de la capa de convolucion

consiste de varios fltros y tienen bias y relu.

6.2 Capa de pooling

hablar del max pooling o average pooling

6.3 Estructura clásica de una cnn

6.4 backpropagration

hablar de como se optmizan los filtros

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

6.5 representacion grafica de una capa convolucional/redes convolucionales

un cuadrado mas o menos gordito etc

6.6 aruitecturas de cnns famosas

hablarde las arquitecturas o implementar alguna

7 Historia

La siguiente información ha sdo obtenida de ChatGPT:

La arquitectura LeNet-5 fue presentada por primera vez en un artículo titulado "Gradient-Based Learning Applied to Document Recognition" publicado por Yann LeCun, Léon Bottou, Yoshua Bengio, y Patrick Haffner en 1998. En este artículo, los autores demostraron el rendimiento superior de la arquitectura LeNet-5 en la tarea de reconocimiento de dígitos escritos a mano en comparación con otras técnicas de la época. Desde entonces, la arquitectura LeNet-5 se ha utilizado como base para muchas otras arquitecturas de redes neuronales convolucionales.

Antes de la introducción de LeNet-5, se utilizaron varias técnicas para reconocimiento de dígitos escritos a mano, como la segmentación de imágenes, el análisis de características manuales y el uso de redes neuronales simples. En particular, el algoritmo de K-nearest neighbors (K-NN) y el algoritmo de redes neuronales de retropropagación se utilizaban comúnmente para esta tarea. Sin embargo, estos métodos no eran tan efectivos como LeNet-5 y tenían limitaciones significativas en términos de precisión y eficiencia en el entrenamiento y la inferencia.

La arquitectura que superó a LeNet-5 en términos de rendimiento y popularidad fue AlexNet, presentada en 2012 por Alex Krizhevsky, Ilya Sutskever y Geoffrey Hinton. AlexNet fue entrenada en un conjunto de datos mucho más grande que LeNet-5, utilizando técnicas como la regularización y el aumento de datos para reducir el sobreajuste y mejorar la precisión. AlexNet también presentó varias innovaciones, como el uso de unidades de activación no lineales ReLU, la normalización por lotes y la utilización de una GPU para acelerar el entrenamiento. Estas técnicas permitieron a AlexNet lograr una precisión sin precedentes en la tarea de clasificación de imágenes, lo que allanó el camino para el desarrollo de arquitecturas aún más grandes y sofisticadas en los años siguientes.

Después de AlexNet, se desarrollaron muchas arquitecturas más grandes y sofisticadas, que lograron una precisión aún mayor en tareas de visión por computadora. Entre las más importantes se encuentran:

- VGG (2014): presentada por Karen Simonyan y Andrew Zisserman, esta arquitectura consta de varias capas convolucionales muy profundas y fue la primera en demostrar que el aumento del número de capas puede mejorar significativamente la precisión.
- Inception (2014): desarrollada por Christian Szegedy y su equipo en Google, esta arquitectura utiliza filtros de múltiples tamaños en paralelo y una estructura de bloques en cascada para mejorar la eficiencia computacional y la precisión.
- ResNet (2015): desarrollada por Kaiming He y su equipo en Microsoft, esta arquitectura utiliza conexiones de salto (skip connections) para permitir que la información fluya directamente a través de las capas sin ser atenuada por la función de activación, lo que permite construir redes extremadamente profundas con una precisión aún mayor.
- DenseNet (2016): desarrollada por Gao Huang y su equipo en el Laboratorio de IA de Google, esta arquitectura utiliza conexiones densas entre capas para permitir que cada capa reciba información de todas las capas anteriores en la red, lo que reduce el riesgo de desvanecimiento del gradiente y mejora la precisión.
- EfficientNet (2019): desarrollada por Mingxing Tan y Quoc V. Le en Google, esta arquitectura utiliza un enfoque de escalado compuesto para optimizar el tamaño, la profundidad y la resolución de la red para obtener la mejor relación rendimiento-eficiencia para una tarea dada.

Cada una de estas arquitecturas ha introducido innovaciones significativas y ha establecido nuevos récords de precisión en tareas de visión por computadora.

La primera red neuronal convolucional para detección de objetos se llamó R-CNN (Region-based Convolutional Neural Network) y fue presentada por Ross Girshick en 2014. La arquitectura de R-CNN se basa en el uso de una red neuronal convolucional para extraer características de la imagen y un clasificador SVM para clasificar cada región propuesta. R-CNN fue un gran avance en la tarea de detección de objetos y logró superar con creces los métodos tradicionales basados en características de bajo nivel y clasificadores manuales. Desde entonces, han surgido muchas otras arquitecturas para la detección de objetos, incluyendo Fast R-CNN, Faster R-CNN, YOLO (You Only Look Once) y SSD (Single Shot Detector).