Redes Neuronales

# Primeros pasos

Para la creación de una red neuronal hay que tener en cuenta 2 cosas, primero el objetivo que se persigue, es decir, para que se va a entrenar, y datos, muchos datos, que permitan su aprendizaje y testeo.

Una vez definido el objetivo y el o los datasets a utilizar, ya sean a través de uno público o creado propio haciendo uso de ETLs por ejemplo, toca modelizar la red neuronal. Para ello se ha de tener en cuenta lo siguiente:

* Funciones de Coste
* Funciones de activación
* Capas de convolución
* Pooling y padding

(la siguiente información la he obtenido de mi TFG)

## Funciones de coste

Las funciones de coste se encargan de determinar el error entre el valor estimado (generado por la red de neuronas) y el valor real para así poder optimizar los parámetros y permitir a la red aprender.

Entre las funciones de coste, destacan las Loss Function o Funciones de pérdida que se caracterizan por evaluar la desviación entre el valor predicho por la red de neuronas y el real, con el fin de optimizar los parámetros. Cuánto más bajo sea este valor, más precisa es la red implementada.

A continuación, se explican las más utilizadas.

### Cross Entropy Loss

Este criterio calcula la perdida de entropía cruzada entre la entrada y el objetivo.[26]

Se mide el rendimiento en un valor de probabilidad de entre 0 y 1, este valor aumenta a medida que la probabilidad predicha diverge de la etiqueta real; es decir, si obtenemos un valor de probabilidad de 0.05 daría una perdida muy alta; y al revés, obtener un valor de 0.95 daría una baja perdida.

De este modo se “penaliza” las predicciones erróneas realizadas y premiar las correctas permitiendo a la red convolucional aprender.

La clase predicha indica el valor con mayor probabilidad al que pertenece, por tanto, la función de coste es [26]:

Donde C representa a la etiqueta de la clase, con C={0,1} y indica la distribución probabilística para cada clase

### Binary Cross Entropy

Se especializa en resolver problemas de clasificación binaria, es decir; del tipo de diferenciar si en una imagen son perros o gatos, avión o coche. [27]

La Binary Cross Entropy compara cada una de las probabilidades predichas con la real, después penaliza las probabilidades en función de la distancia desde el valor esperado.

La fórmula se define como [27]:

Donde C representa a la etiqueta de la clase, con C={0,1} y p indica la probabilidad de la clase

### MSE Loss

Mide el error cuadrático medio entre cada uno de los elementos de la entrada X y el objetivo. Se usa especialmente en problemas de regresión con aprendizajes automáticos supervisados.

Consiste en la siguiente fórmula matemática:

Imagen que contiene objeto, reloj

Descripción generada automáticamente

Donde y es el valor real al que se resta el valor predicho(ŷ)

### Hinge Embedding Loss

Se encarga de medir la perdida dado un tensor x y un tensor y, con valores de 1 a -1.

Generalmente se tiende a utilizar para medir si dos entradas son similares, utilizando la distancia L por pares. Por ello, la función de coste es la siguiente [24]:

Texto

Descripción generada automáticamente

Es ideal para el aprendizaje de ilustraciones no lineales y aprendizajes semi-supervisados

## Funciones de activación

Las funciones de activación en las redes neuronales son el “cómo piensan” las neuronas, es decir; indican el resultado que va a producir la neurona dado una entrada o un conjunto de entradas. Este valor suele estar en el rango [-1.1].

A continuación, se explican más en detalle las funciones más utilizadas.

### Función de Heaviside

La función de Heaviside o función escalón unitario , fue la primera en ser utilizada en las redes neuronales. Consta de una función discontinua cuyo valor siempre devolverá 0 para los valores negativos y 1 para los positivos.

Sin embargo, para adaptarla a las redes neuronales, se modificó para que el valor de entrada fuera entre 0 y 1 , es por ello por lo que carece de sentido que para valores negativos devuelva 0, y para positivos 1, así que se reajustó. En redes neuronales, el valor umbral es 0.5, si la entrada es por debajo, devuelve 0, si es superior, devuelve 1.

### Función ReLU

La función Rectificied Lineal Unit o ReLU por sus siglas en inglés, transforma los valores introducidos, anulando los negativos y dejando los positivos sin modificar, es decir; si recibe -1, anula la entrada, si recibe 0.4, devuelve 0.4.

Su uso está muy extendido dado que permite un rápido aprendizaje a las redes neuronales. Además, aporta las siguientes ventajas [30]:

* Simplicidad de cálculo
* Capacidad de generar un valor de salida con 0, dado que los valores negativos devolverán el valor 0. De este modo las capas ocultas pueden utilizar este valor simplificando mucho el modelo
* Tiene un comportamiento lineal

La función quedaría de la siguiente manera:

Del mismo modo la representación gráfica quedaría de la siguiente manera:

Gráfico, Gráfico de líneas

Descripción generada automáticamente

*Figura 2.7.2.1 Representación gráfica de la función de activación ReLU [31]*

### Funciones de activación Sigmoideas

Las funciones sigmoideas permiten mitigar el efecto outliers, es decir; observaciones anormales que pueden afectar negativamente a los parámetros.

Este tipo de funciones tienen una representación gráfica, característica de un trazado simétrico delimitado por 2 asíntotas horizontales tal que:

*Figura 2.7.3.1 Representación de una función de activación sigmoide*

#### Función logística

Es especialmente utilizada porque convierte casi cualquier variable independiente en una probabilidad simple entre 0 y 1.

#### Softmax

También conocida como función exponencial normalizada, es una generalización de la función logística. (“Función Softmax - Softmax function - abcdef.wiki”)

Es una función de activación no lineal que permite re-escalar las n dimensiones de la entrada en una salida de rango [0,1] y cuyo sumatorio es 1.[29]

La función quedaría de la siguiente manera:

Donde es el valor de todas las otras neuronas de la capa.

Este tipo de funciones se suele utilizar exclusivamente en la salida de la red, dado que el objetivo en las salidas es sumar 1. Esto se conoce como normalización. Especialmente es útil en problemas de clasificación.

## Capas convolucionales

En el uso de redes neuronales para clasificación y detección de objetos, hace falta un proceso conocido como convolución. Para ello, cada píxel de la imagen ha de tener un valor numérico, dado por su color, luminancia….

Una vez se obtengan los valores de cada píxel, se realizarán pequeños grupos de pixeles cercanos y se aplicará un producto escalar con otra matriz conocida como kernel.

Este Kernel no se aplica solo una vez a un conjunto de pixeles, si no que se aplica a cada conjunto seleccionado. Tras ello se obtiene una matriz de convolución sobre la que se aplica una de las funciones de activación anteriormente vistas.

Diagrama

Descripción generada automáticamente

*Figura 2.8.2.1 Proceso de convolución [32]*

Después de aplicar la función de activación, se obtiene un mapa de características de la imagen. Normalmente no existe una sola capa de convolución, sino que se aplica múltiples veces este proceso, tal y como se puede apreciar en la figura 2.8.2.2. En este caso se trata de la arquitectura de VGG Net. Primero se le aplica a la imagen 2 convoluciones consecutivas de 224x224x64, tras una capa de Pooling, cuyo proceso se explicará en el siguiente punto. De este modo se obtiene un mapa de características más preciso de la imagen permitiendo reconocer mejor la forma de los objetos. Además, se puede observar cómo utiliza la función de activación ReLU para las convoluciones, y Softmax para la salida.

Diagrama

Descripción generada automáticamente

*Figura 2.8.2.2 Arquitectura de VGG Network [24]*

## Pooling

La capa de reducción o Pooling se utiliza especialmente para agrupar los pixeles o resultados obtenidos tras una capa de convolución, con el fin de reducir las dimensiones espaciales para la siguiente capa convolucional.

Esta operación también se conoce como reducción de muestreo, ya que al este proceso conlleva también a la pérdida de información, pero agiliza mucho el proceso de computación. Sin embargo, esto puede ayudar a reducir las probabilidades de sobreajuste.[32]

### Max-Pooling

Es un proceso de Pooling que recoge el máximo valor de cada parte del mapa, creando un mapa de características reducido. Por lo general se utiliza siempre después de un proceso de convolución. Tal y como se puede apreciar en la figura 2.9.1.2, de cada partición realizada en el mapa, se obtiene el máximo valor y se introduce en el mapa reducido. Este método es el más utilizado hoy en día.

Imagen de la pantalla de un celular con letras

Descripción generada automáticamente con confianza media

*Figura 2.9.1.1 Max-pooling de 2x2 [33]*

### Average Pooling

Este proceso de Pooling , realiza la media entre los valores de cada parte del mapa para crear el mapa reducido, tal y como se puede observar en la figura 2.9.2.1

Imagen de la pantalla de un celular con texto e imágenes

Descripción generada automáticamente con confianza baja

*Figura 2.9.1.1 Average pooling de 2x2 [34]*

## Padding

Padding consiste en introducir pixeles con un valor 0 alrededor de la imagen, como se puede apreciar en la figura 2.10.1. Es especialmente útil cuando la información de la imagen no está disponible en el centro de esta, sino en los bordes y esquinas, debido a que en los procesos de convolución estas esquinas aparecen muy pocas veces en los kernel escogidos, y esta información apenas influye en el mapa de características. Es por ello por lo que al introducir esos pixeles 0 sin valor, se permite en el proceso de convolución, que las esquinas y bordes aparezcan e influyan igual que el resto de los pixeles en el mapa de características.

Además, permite tras realizar Pooling, se mantenga el tamaño original de la imagen, pero con la información concentrada en el centro gracias a esta.

# Modelización de la red neuronal

Tras haber comprendido la teoría, vamosa un ejemplo práctico

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

La primera línea de \_\_init\_\_ llama a la función super() que permite la herencia múltiple de las clases base (en este caso, nn.Module).

La función forward(x) define cómo se procesan los datos a través de la red. La función view(-1, 32\*16\*16) aplana el tensor de salida de la segunda capa convolucional para que pueda ser ingresado a la capa totalmente conectada.

Usamos la función de activación ReLU después de cada capa (excepto la última), que añade no linealidad al modelo. Después de la última capa, utilizamos la función de activación sigmoide para obtener una salida entre 0 y 1, que puede interpretarse como la probabilidad de que la imagen sea de un perro.

La función Softmax se utiliza comúnmente en la capa de salida de una red neuronal en problemas de clasificación multiclase, ya que transforma los valores de salida de la red en probabilidades que suman 1 entre todas las clases.

En este caso, sin embargo, estamos enfrentando un problema de clasificación binaria (la imagen es un perro o no es un perro). Por lo tanto, en lugar de Softmax, es común utilizar la función sigmoide en la capa de salida para la clasificación binaria. La función sigmoide transforma los valores de salida de la red en una probabilidad entre 0 y 1, lo que puede interpretarse como la probabilidad de que la imagen sea de un perro.

Del mismo modo es más eficiente utilizar BCELoss para calcular la pérdia que hacer Softmax en la Salida