Deep learning en visión

Robótica - Grado en Ingeniería de Computadores

Departamento de Sistemas Informáticos
E.T.S.I. de Sistemas Informáticos - Universidad Politécnica de Madrid

22 de octubre de 2023

Conceptos básicos de redes de neuronas

Sistema matemático capaz de realizar predicciones a partir de datos de entrada

- Propuesta por McCulloch y Pitts en 1943
- Basada en imitar el comportamiento de una neurona biológica
- Toma ciertos estímulos de entrada, los procesa y genera una nueva salida

Neurona biológica

- Estímulos → impulsos nerviosos

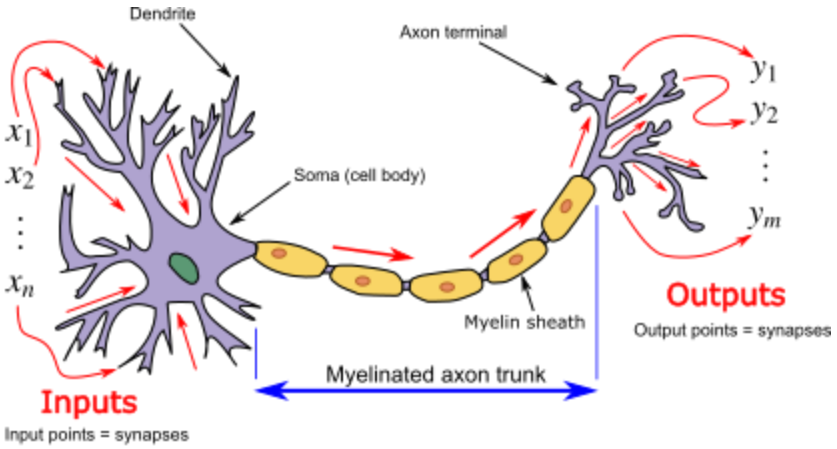
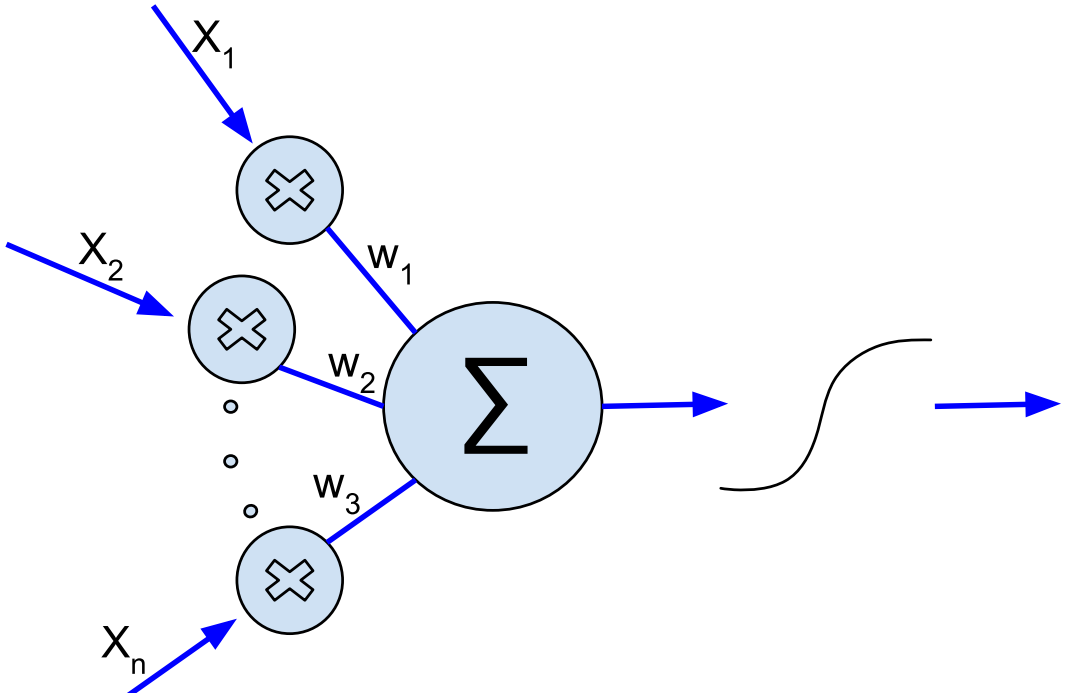


Fig.1 - Neurona biológica. Fuente: Wikipedia.

Neurona artificial

- Estímulos → cálculos matemáticos



Neurona artificial

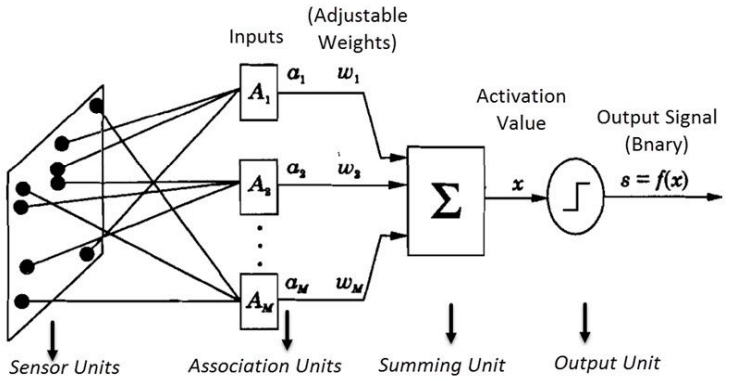


Fig.3 - Neurona artificial. Fuente: ElectronicsHub.

Para ello, existen diversos elementos dentro de una neurona artificial:

- **Entradas** (x_i): Los valores numéricos de entrada
- **Salida** (y): El valor de salida de la neurona
- **Pesos** (w_i): Parámetros capaces de cambiar, suponen el aprendizaje de la neurona
- **Bias** (b): Peso cuya entrada siempre es 1 y que desplaza la función de activación
- **Función de activación**: a: Participa en el cálculo de la salida de la neurona

Inferencia o propagación

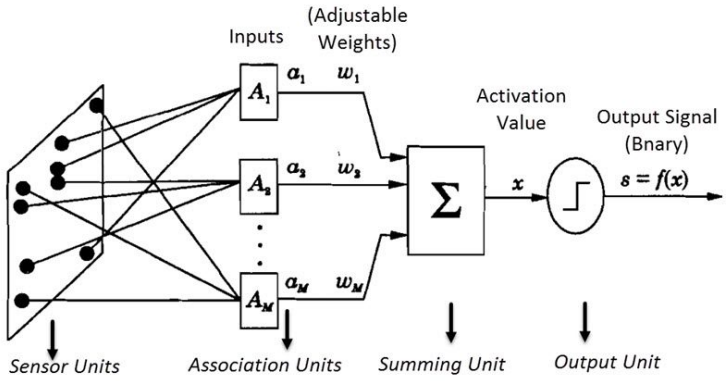


Fig.3 - Neurona artificial. Fuente: ElectronicsHub.

Se encarga de procesar la entrada y generar la salida correspondiente

1. Cada entrada x_i es multiplicada por el valor de su peso correspondiente w_i,
   - La entrada del *bias* siempre es 1, y tiene su propio peso (denominado w_0 o b)
2. Todos los productos se suman (**entrada neta**),
3. La entrada neta pasa por la función de activación a para generar la salida $\hat{y}$.

Cuando hay varias capas, **la salida de una capa es la entrada de la siguiente**

Estructura de capas

Las redes de neuronas se suelen organizar por capas

- Éstas se componen de varias neuronas
- Cuando cada neurona de una capa se conecta con todas las de la siguiente, se denomina **capa densa** o **fully connected**
- Si la red posee sólo capas densas se denomina perceptrón multicapa (MLP)

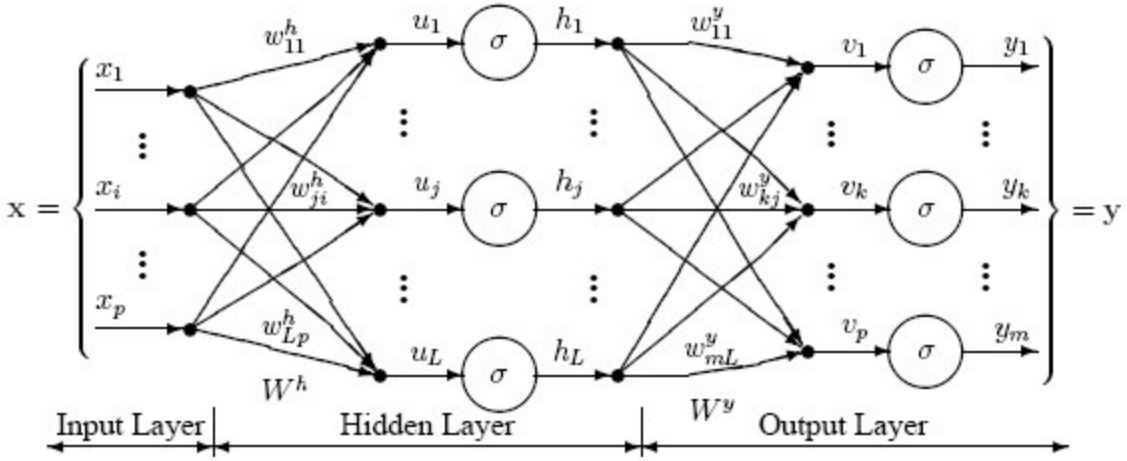


Fig.4 - Esquema de un perceptrón multicapa. Fuente: AILephant.

Funciones de activación

Las **funciones de activación** de cada neurona pueden variar; algunas de las más populares son:

Fig.5 - Algunas funciones de activación. Fuente: Neural Network Activation Functions (Code360).

Función de error (pérdida o *loss*)

Se usa para calcular el error entre salida deseada y real

- Para un ejemplo: coste (*coste*); para muchos: pérdida (*loss*),

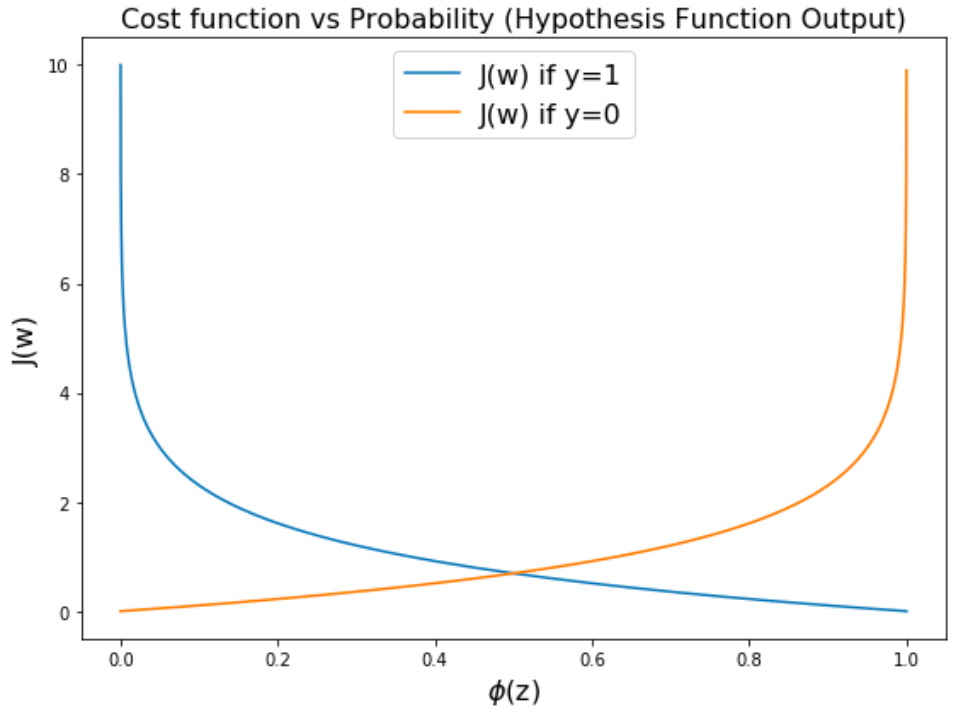


Fig.6 - Entropía cruzada (clasificación). Fuente: Analytics Yogi.

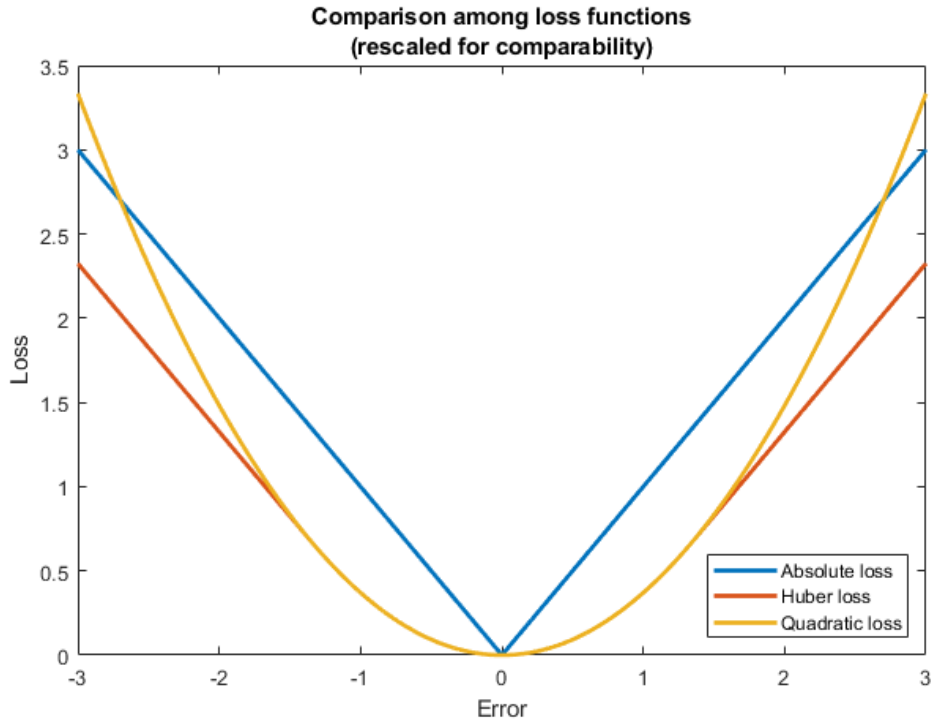


Fig.7 - Ejemplos de *loss* en regresión. Fuente: StatLect.

Algoritmo de retropropagación (*backpropagation*)

Es el encargado de adaptar la red de neuronas a su cometido específico

$$\triangle w_i = w_i - \alpha \cdot \delta_i$$

- δ_i es directamente proporcional al loss en la última capa
- δ_i es directamente proporcional al error de la capa inmediatamente posterior
  - De ahí lo de retropropagación, el error se va propagando de la última a la primera capa
- Para calcular la dirección del error se usa el gradiente de la función de activación

Entrenamiento

Entrenamiento en un esquema de aprendizaje supervisado:

1. **Inferencia**: Se calcula la salida de la red para una entrada,
2. **Calculo del error**: Se comparan las salidas real y deseada,
3. **Retropropagación**: Se modifican los pesos según el error.

Esta operación se realiza iterativamente hasta que el error sea lo suficientemente bajo

- Cada iteración se denomina **época** o *epoch* y es una vuelta a todo el conjunto de entrenamiento
- A cada porción de datos de entrenamiento se le denomina **lote** o *batch*

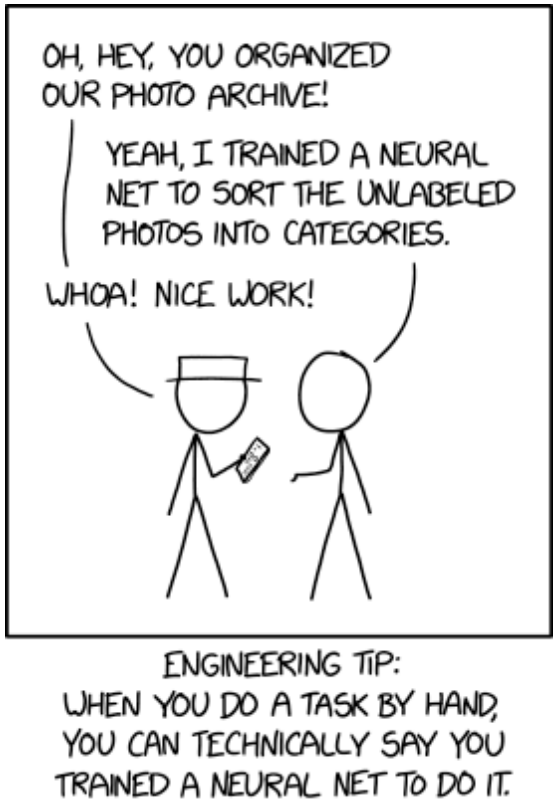


Fig.8 - Neurona artificial.
Fuente: XKCD.

Entrenamiento de redes neuronales

Durante la creación de modelos, se suelen usar tres conjuntos de datos:

- **Entrenamiento** (*training*): Usado para entrenar el modelo,
- **Validación** (*validation*): Usado para validar el modelo **durante el entrenamiento**,
  - Suele ser un subconjunto del conjunto de entrenamiento
- **Testeo** (*testing*): Usado para probar el modelo una vez entrenado.
  - Debe ser un conjunto de datos no visto por el modelo durante el entrenamiento

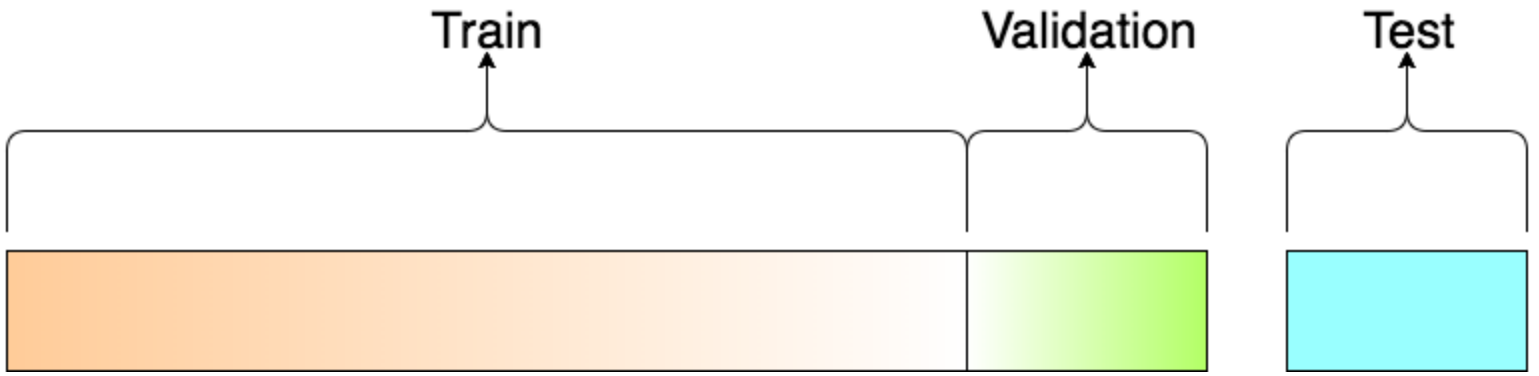


Fig.9 - Representación de los conjuntos de entrenamiento, validación y test. Fuente: Towards Data Science.

Compromiso sesgo-varianza

En inglés *bias-variance tradeoff*, es el equilibrio entre dos tipos de error:

- **Sesgo** (*bias*): Error por supuestos demasiado simples (*underfitting*)
- **Varianza** (*variance*): Error por excesiva sensibilidad a los datos (*overfitting*)

Modelos y complejidad:

- **Modelo simple**: Puede no capturar las relaciones subyacentes en los datos
- **Modelo complejo**: Puede sobreajustarse, capturando ruido y anomalías

Implicaciones:

- **Sobreajuste** (*overfitting*): Modelos muy complejos con alta varianza
- **Subajuste** (*underfitting*): Modelos muy simples con alto sesgo
- Objetivo: Equilibrio donde el modelo es lo suficientemente flexible para capturar la verdadera relación, pero no tanto como para ser engañado por el ruido

¿Cómo detectar altos sesgo o varianza?

Detección de un alto sesgo:

- **Indicador**: Error elevado en entrenamiento
- **Soluciones**: Aumentar complejidad, más épocas, revisar arquitectura y calidad de datos.

Deteción de una alta varianza:

- **Indicador**: Gran diferencia entre errores de entrenamiento y validación
- **Soluciones**: Más datos, usar regularización (ej. dropout), disminuir complejidad, aumentado de datos

Problemas del gradiente

Recordando qué es el gradiente

Vector que indica la dirección y magnitud de máximo crecimiento de una función

- En redes neuronales nos dice **cómo actualizar los pesos para minimizar el error**
- Se obtiene al derivar la función de *loss* respecto a cada peso
  - Así sabemos cuánto cambiaría la función de pérdida si ajustáramos un peso específico

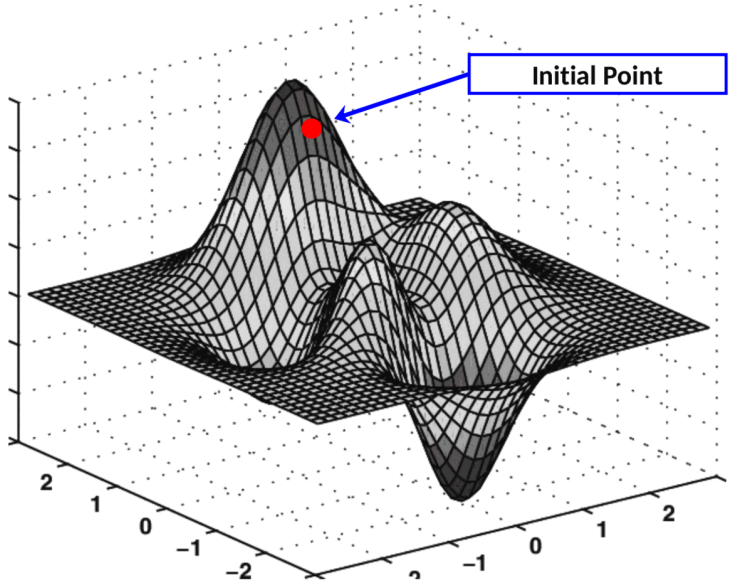


Fig.10 - Gradiente calculándose para varios pesos y su evolución a lo largo de 10 iteraciones. Fuente: Towards AI.

Problema del desvaniecimiento del gradiente

Ocurre cuando los **gradientes** de las capas más alejadas a la entrada **tienden a disminuir exponencialmente** a medida que se retropropagan a través de la red

- Provoca actualizaciones mínimas, más cuanto más cercanas a la entrada
- Aprendizaje lento o nulo: Las capas profundas no convergen (o sí, pero muy lento)
- Mayor cuanto más capas (multiplicación continua de valores pequeños)

La principal causa es la naturaleza de ciertas funciones de activación

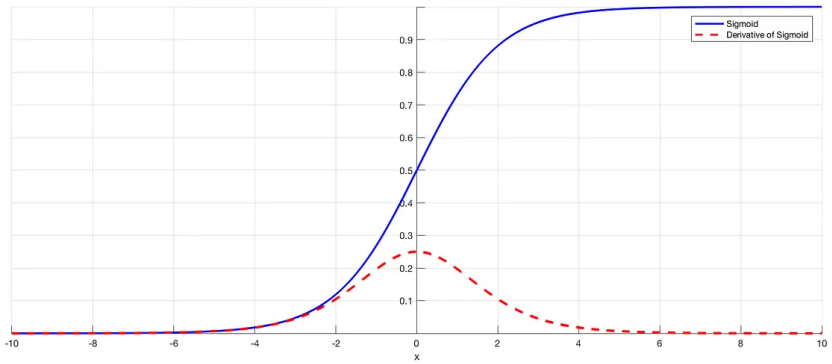


Fig.11 - La sigmoide comprime muchos valores de entrada a un rango limitado. Fuente: Towards Data Science.

Problema de la Explosión del Gradiente

Ocurre cuando los **gradientes** se vuelven **excesivamente grandes**, causando actualizaciones de pesos desproporcionadas, haciendo que la red sea inestable

- Es el fenómeno opuesto al desvanecimiento del gradiente
- Actualizaciones inestables, red que oscila y no converje, o incluso que diverje a infinito
- Riesgo de Overfitting: La red puede adaptarse demasiado a particularidades del conjunto de entrenamiento.
- Valores NaN: Las actualizaciones excesivas pueden llevar a valores numéricos indeseados o no válidos en la red.

A diferencia del desvanecimiento del gradiente, donde el aprendizaje se detiene, la explosión del gradiente puede hacer que la red aprenda patrones incorrectos o simplemente falle.

Soluciones a los problemas del gradiente

Algunas soluciones son de diseño o de parametrización:

- **Funciones de activación**: Optar por funciones como ReLU o sus variantes (Leaky ReLU, Parametric ReLU) que no saturan en regiones extensas
- **Inicialización de pesos**: Utilizar técnicas como la inicialización He o Xavier/Glorot, que consideran el tamaño de las capas anteriores y siguientes.
- **Optimizadores Avanzados**: Como Adam, RMSprop o Adagrad, que ajustan dinámicamente las tasas de aprendizaje.

Otras son ya técnicas específicas de estos problemas

- **Gradient Clipping**: Establece un umbral para limitar el tamaño del gradiente
- **Batch Normalization**: Normaliza la activación de las neuronas dentro de una capa para mantenerlas en un rango deseado.

Perceptrón multicapa para procesar imágenes

¿Cómo procesar imágenes?

Las imágenes son matrices, pero las redes neuronales trabajan con vectores

- Así que no nos queda otra que aplanar la imagen en un vector unidimensional

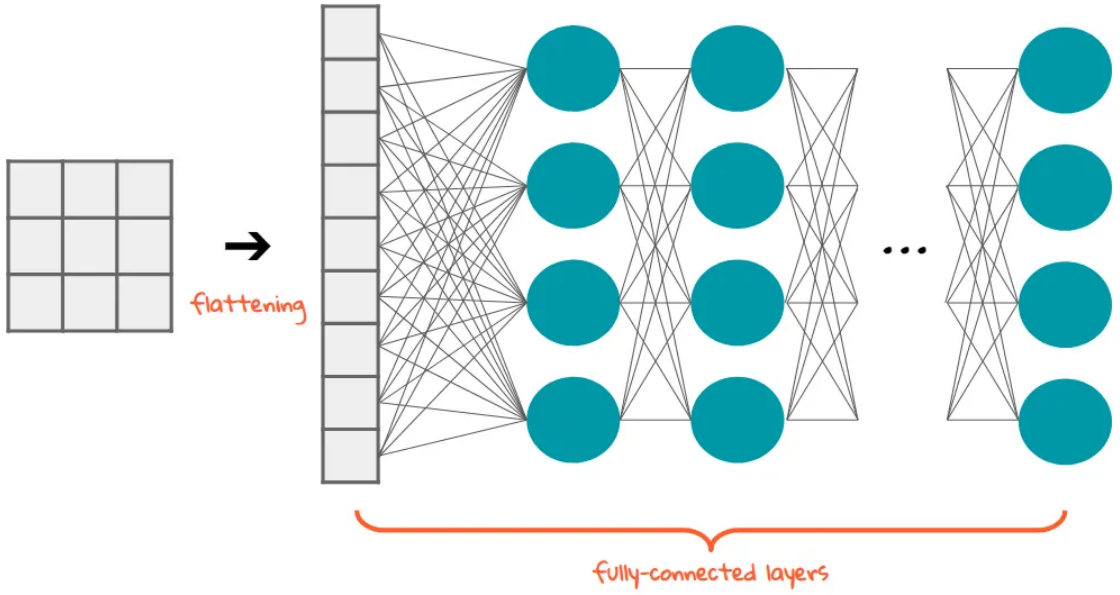


Fig.12 - La entrada al perceptrón debe ser pasada a vector antes de ser procesada. Fuente: Towards Data Science.

En `keras` existe la capa `Flatten` precisamente para realizar esta operación

Implementando un perceptrón multicapa

El siguiente notebook contiene un ejemplo de clasificador de imágenes usando un perceptrón multicapa como red neuronal

Ejercicio: 2.2. Clasificación de dígitos con un perceptrón multicapa.ipynb[1]

1 https://githubtocolab.com/etsisi/Robotica/blob/main/Notebooks/2.2. Clasificación de dígitos con un perceptrón multicapa.ipynb

Problemas del perceptrón

Al transformar la matriz en vector **se pierde la información espacial** de la imagen

- En particular, todas las relaciones de color y distancia

Fig.13 - Aplanar una imagen hace que perdamos su información espacial. Fuente: SuperDataScience.

Y también está la **enorme magnitud de las redes** creadas de esta manera

- Ejemplo: Imagen de 512×512 píxeles $\rightarrow$ ¡786.432 neuronas de entrada!

Fundamentos de las redes convolucionales

Operación de convolución

Las redes **convolucionales** son redes adaptadas al **tratamiento de imágenes**

- Y se apoyan en la **convolución**, que es el producto punto entre dos matrices

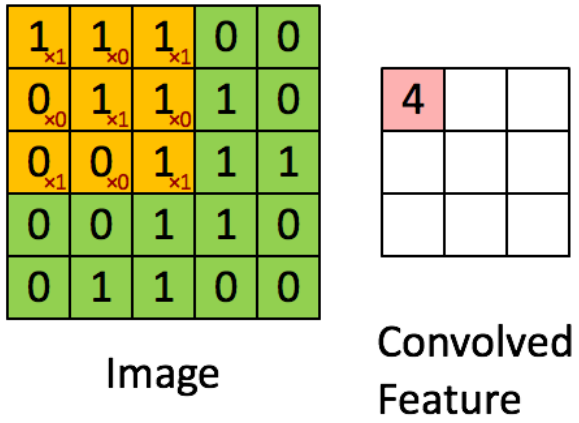


Fig.13 - Operación de convolución aplicada a una imagen. Fuente: IceCream Labs.

Para ello deben existir dos elementos fundamentales:

- Imagen de entrada: Matriz tridimensional de datos
- Filtro o *kernel*: Matriz con la que realizar la operación de convolución

Campo receptivo

La salida de la convolución es una extracción de características de la imagen

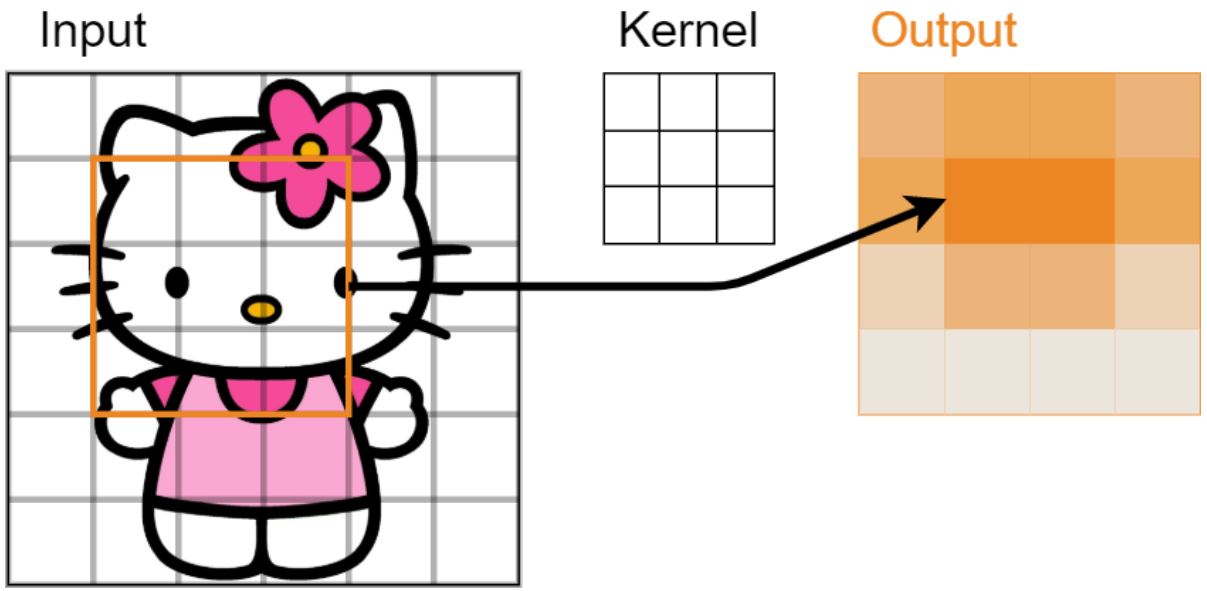


Fig.14 - Extracción de características asociadas a un filtro concreto.

El **campo receptivo** de cada celda de salida se **activa** al detectar la **característica**

- Diferentes filtros detectarán diferentes características

¿Y qué aprende la red?

Una red de convolución sustituye las primeras capas **densas** por **convolucionales**

- El bloque convolucional extrae características
- El bloque denso predice de acuerdo a características, (y no a píxeles en bruto)

Cada capa convolucional está compuesta por una serie de filtros de igual tamaño

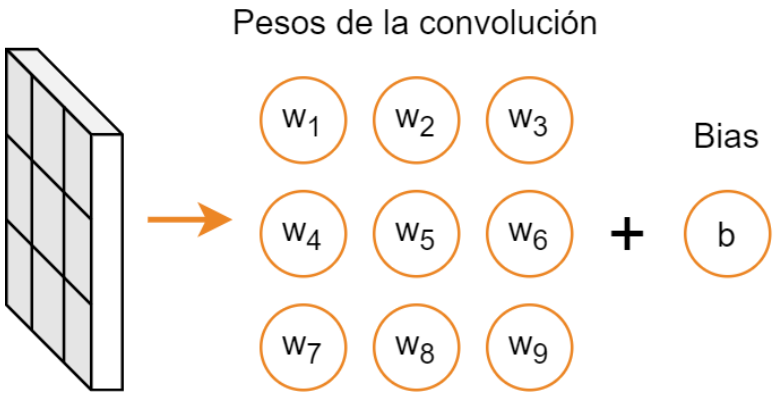


Fig.15 - Los pesos de las capas de convolución son los filtros.

La red aprenderá qué filtros extraen las características para el problema a resolver

Tamaño de la salida de una capa convolucional

Supongamos que tenemos:

- Una imagen de entrada de $n \times m$ píxeles y c canales,
- Un filtro de $f \times g$ píxeles,
- Una capa de convolución con k filtros,

Entonces la salida de la capa convolucional será de:

$$(n - f + 1) \times (m - g + 1) \times k$$

La salida es una nueva imagen **más pequeña** de las características extraídas

Padding en la convolución

Muchas convoluciones reducen el tamaño de la imagen de salida

- Lo que es un problema en redes muy profundas

¿Cómo se soluciona? Con un relleno de ceros (*padding*) en la imagen de entrada

- Filtro de $n \times m \rightarrow \lfloor n/2 \rfloor$ arriba y abajo, $\lfloor m/2 \rfloor$ derecha e izquierda

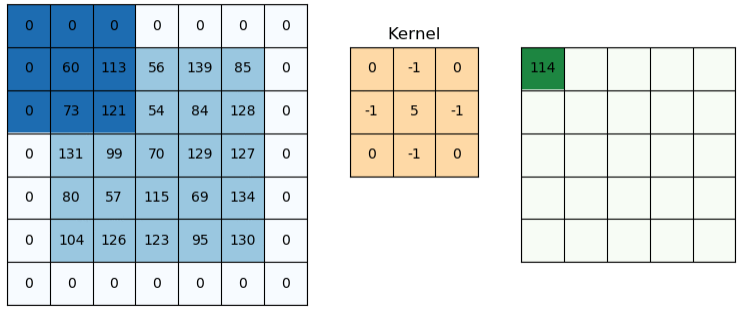


Fig.16 - Los pesos de las capas de convolución son los filtros.

Así la imagen de **salida** tendrá el **mismo tamaño** que la de **entrada**

Salto (*stride*) en la operación de convolución

Un *stride* es el número de píxeles que se desplaza el filtro al realizar la convolución

- No tienen por qué coincidir ambos desplazamientos (y por defecto son 1)

La fórmula de la dimensión cuando hay:

- Una **dimensión** de entrada de n píxeles,
- Un **filtro** de f píxeles,
- Un **salto** de s píxeles,
- Un **padding** de p píxeles

Es la siguiente:

$$\frac{n - f + 2p}{s} + 1$$

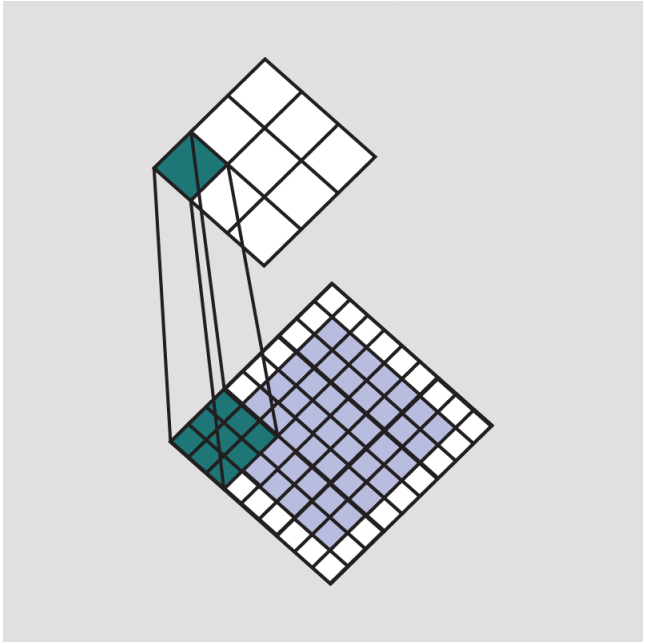


Fig.17 - Filtro de 3×3 con desplazamiento de 3 en ambas direcciones. Fuente: The Startup

Reducción dimensional en redes convolucionales

La operación principal para ello es el *pooling*, cuyas dos variantes son:

- *MaxPooling*: Se queda con el valor máximo de la ventana
- *AveragePooling*: Se queda con el valor medio de la ventana

Es similar a una operación de convolución, pero sin pesos

Fig.18 - Ejemplo de operación de *max pooling*. Autor: Parva Shah.

Normalmente el filtro y el salto (stride) son iguales, y suelen ser de 2×2

Clasificador con redes convolucionales

El siguiente notebook contiene un ejemplo de clasificador redes convolucionales como red neuronal

Ejercicio: 2.3. Entendiendo las redes de convolución.ipynb[2]

2 https://githubtocolab.com/etsisi/Robotica/blob/main/Notebooks/2.3. Entendiendo las redes de convolución.ipynb

¡GRACIAS!