Daniel Vázquez Lago

Deep Learning

Con aplicaciones en Python

Índice general

Redes Neuronales

Librerías de Python

Capítulo 1

Introducción al *machine learning*

El *deep learning* o aprendizaje profundo permite a los modelos computacionales compuestos de varias capas de procesamiento apreder representaciones de datos con múltiples niveles de abstracción. Estso métodos han mejorado dramáticamente el estado del arte en el sentido del reconocimiento de voz, de imágenes, detección de objetos y otros muchos dominios como el descrubrimiento de fármacos y la genómica. El aprendizaje prfundo descubre patrones realmente complejos en grandes volúmenes de datos usando algoritmos de *backpropagation* (retropropagación) que le indica a la máquina si debería cambiar sus parámetros internos usados para calcular las representaciones en cada una de las capas a partir de las representaciones de la capa anterior. Las redes convonucionales han traido además inovaciones en el campo del procesamiento de imágenes, video, discursos y audio, mientras que las redes recurrentes tienen un papel fundamental en el procesamiento de dato secuencial como pueden ser textos y discursos.

1.1. Introducción

El aprendizaje máquina o *machine learning* ayuda a la sociedad en múltiples aspectos: desde búsquedas de infromación en la web, como filtros en redes sociales, recomendaciones en comercio electrónico... Los sistemas de aprendizaje máquina se usan paara identificar objetos en imágenes, transcribir discursos a texto plano o seleccionar los resultados relevantes en una búsqueda en internet. Cada vez más en estas aplicaciones usamos un tipo particular de técnicas llamdas aprendizaje profundo o *deep learning*.

Las primeras técnicas de aprendizaje máquina estuvieron muy limitidas en su capacidad de procesar datos crudos. Incluso décadas después de su inicio, construir un patrón eficiencia de reconocimiento de datos (un sistema de aprendizaje máquina) requería una cuidadosa construcción, además de un gran dominio en la materia para diseñar tanto la extracción como la transformación de datos crudos (como podrían ser los valores de información de cada pixel de una imagen) en una represntación interna como podría ser un vector, a partir la cual el sistema de aprendizaje, normalmente un clasificador, podría detectar patrones en la entrada (*input*)

El aprendizaje de representaciones o *represntation learning* es una sier de métodos que le permite a la máquina, a partir de datos crudos, descrubir representaciones necesitadas para la clasificación o detección. Los métodos de aprendizaje profundo son métodos de aprendizaje de represntaciónes con múltiples niveles de representación, obteniendo de un cojunto simple de modulos no lineales, los cuales transforman la representación a un solo nivel (empezando con los datos crudos) en una re-

presntación mucho más abstracta. Con la composición de suficientes transformaciones, un conjunto complejo de funciones pueden llegar a ser aprendidas por la máquina. Cuando el fin de la máquina es la clasificación, capas de representación más altas amplificarán aspectos del *input* que son importantes para la posterior discriminación, a la par que suprimirá las variaciones irrelevantes. Una imagen por ejemplo viene dada como una matriz de valores pixel, y la primera capa en general suele detectar o representar la ausencia o presencia de bordes, a través de cambios de color, luminosidad... La segunda capa detecta motivos/patrones en un conjunto particular de bordes, o variaciones pequeñas de los bordes. La tercera capa juntará todos los motivos en un conjunto mucho más grande que corresponderá con partes de objetos familiares. Las siguientes capas podrán detectar objetos como combinaciones de esas partes. El aspecto clave del aprendizaje profundo es que estas capas no están diseñadas por un humano, sino que se aprenden a partir de datos utilizando un procedimiento de aprendizaje de propósito general.

El aprendizaje profundo esta haciendo grandes avances en la resolución de problemas que hasta ahora resistían los mejores intentos de la inteligencia artificial. Se ha vuelvo una herramienta muy importante en el descubrimiento de estructuras intrincadas con grandes dimensiones y por tanto es apicable a numerosos campos de la ciencia, negocios y gobierno. Este método es el mejor en reconocimiento de imágenes y de voz, y además predice la activación de drogas, analiza información de aceleradores de partículas, recostruye circuitos cerebrales...

1.2. Aprendizaje supervisado

1.3. Retropopagación en el entrenamiento de arquitectura multicapa

1.4. Redes neuronales convolucionales

1.5. Representaciones distribuidas y procesamiento de lengauje

1.6. Redes neuronales recurrentes

1.7. El futuro del *deep learning*

El aprendizaje sin supervisión ha revivido el interés en el *deep learning*, aunque ha sido opacado por el éxito rotundo el aprendizaje supervisado. Es de esperar que el aprendizaje no supervisado se haga cada vez mas importante, sobretodo teniendo en cuenta que tanto los humanos como el resto de los animales aprendemos sin una supervisión.

La visión humana es un proceso activo que muestrea secuencialmente el conjunto óptico de forma inteligente y específica para cada tarea, utilizando una fóvea pequeña y de alta resolución con un entorno grande y de baja resolución. Esperamos que la mayor parte del desarrollo de algoritmos de visión venga dado por sistemas que son entrenados *end-to-end* con combinaiones de ConvNets

y RNNs que usen el aprendizaje reeforzado para decidir a donde mirar. Sistemas que combinenen aprendizaje profundo y lenguaje reforzado son todavía muy nuevos, pero ya han demostrado su efectividad en el uso de sistemas de visión pasiva en la clasficiación de elementos, además de ser capaces de jugar o aprender a jugar a diferentes videojuegos.

El lenguaje natural es otro área en la que el aprendizaje profundo tendrá un impacto en los próximos años, ya que esperamos que los procesos RNN permitan entender frases o incluso documentos enteros, lo que nos lelvará a procesos mucho mejores en tanto en cuanto sean capaces de aprender estrategias para seleccioanr partes del texto. Finalmente, también se espera que procesos mayores en la inteligencia artificial permitirá pasar a través de sistemas que combinen la representación en leguaje y un razonamiento complejo.

Capítulo 2

Fundamentos del aprendizaje máquina

2.1. Clasficiación Binaria

Para poder avanzar en el tema, primero tenemos que definir algunos de los términos que hemos usado de manera intuitiva: aprendizaje, experiencia, mejora y tarea. Empecemos con la tarea de la clasificación binaria.

Supongamos un problema abstracto en el cual tenemos los datos de la siguiente forma:

$$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \tag{2.1}$$

donde $x \in \mathbb{R}^n$ e $y = \pm 1$. A pesar de que D represente todos los datos, nosotros solo tenemos acesso a un subconjunto de ellos, $S \in D$. Usando S, nuestra tarea será generar un proceso computacional que implemente la función $f : x \to y$ tal que podemos usar f para predecir los datos no pertenecientes a S, esto es, $x_i, y_i \notin S$, tal que $f(x_i) = y_i$. Si dennotamos U como el conjunto contario a S ($U = \bar{S}$), podemos suponer que una medida de la eficiencia/calidad de nuestro sistema para realizar esta tarea vendrá dada por una función error sobre los datos no vistos:

$$E(f, D, U) = \frac{\sum_{x_i, y_i \in U} [f(x_i) \neq y_i]}{|U|} \tag{2.2}$$

Medimos el rendimiento de la tarea usando la función error $E(f, D, U)$ sobre los datos no conocidos U. EL tamaño de los datos conocidos S será nuestra "experiencia". Lo que nosotros queremos hacer es crear algoritmos que generen funciones f (comunmente denominadas modelos). En general a los valores x se le llaman valores de entradas o *inputs*, mientras que a y valores de salida o *outputs*.

Como en cualquier otra disciplina, las características computacionales es un aspecto relevante, pero además de tener en cuenta los costos computacionales, tendremos que tratar de hallar el modelo f que tenga el menor errror posible $E(f, D, S)$ con el menor $|S|$ posible.

Ejemplo 2.1: Web de Comercio Electrónico

Supongamos que queremos implementar en una página web una página personalizada para usuarios registrados, que les muestre los productos que más dispuestos estarían a comprar. La web registra la información de los usuarios. ¿Cómo se relaciona este problema con la clasificación binaria?

Esta claro que el problema relaciona productos con personas. Centremonos entonces en la relación entre un solo producto y un solo usuario. Lo que nosotros queremos saber es si va a ser comprado. Podemos denotar el estado "deseado" con $y = +1$ mientras que el estado "no deseado" por $y = -1$. Lo siguiente que debemos hacer es coger información del producto (tipo de producto...) y la información histórica de búsqueda y compra del usuario, representado todo esto en un vector $x \in \mathbb{R}^n$. Basados en esta información y en el mapeo conocido $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, entonces podemos generar la función modelo $f : x \to y$ a partir lo cual podemos determinar que productos querrá comprar y llenarle la página con esta información. Podríamos medir lo "buena" que es esta medida en función de si el usuario compra los productos mostrados, evaluando luego la función error con los datos predichos y si los compra o no.

2.2. Regresión

2.3. Regularización

Capítulo 3

Redes neuronales de propagación hacia adelante

En este capítulo trataremos algunos de los conceptos básicos en las redes de propagación hacia adelante o en ingles *feed forward neuronal networks*. Estos conceptos servirán más adelante para entender conceptos mucho más complejos. De manera abstracta, una red neuronal puede ser pensado como una función $f_\theta : x \to y$ tal que coge la entrada $x \in \mathbb{R}^n$ y produce una salida $y \in \mathbb{R}^m$, cuyo comportamiento está parametrizado por $\theta \in \mathbb{R}^p$, com por ejemplo $y = f_\theta(x) = \theta \cdot x$.

3.1. Unidades

Las **unidades** (*units*) son los elementos más básicos dentro de una red neuronal. Una unidad es una función que coje un valor de entrada $x \in \mathbb{R}^n$ y produce un escalar. Cada unidad está parametrizada por un peso $w \in \mathbb{R}^n$ y un término de *bias* (permite al modelo desplazar la función de activación hacia la izquierda o derecha) denotado por b, tal que el valor de salida de cada unidad puede ser descrito como

$$f\left(\sum_{i=1}^{n} x_i \cdot w_i + b\right) \tag{3.1}$$

donde $f : \mathbb{R} \to \mathbb{R}$ es lo que llamamos **función de activación**. Existe una gran cantidad de funciones de activación, y son, en general, funciones no lineales.

3.2. Estrucutra de una red neuronal

Las unidades se organizan en **capas** (*layers*), con cada capa coteniendo una o varias unidades. La última capa se le denomina *capa de salida*, y todas las capas anteriores *capas ocultas*. El número de unidades de una capa se le llama ancho o espesor de capa, y no todas las capas deben tener el mismo espesor, aunque las dimensiones deberán estar alineadas. El número de capas se denomina **profundidad** de la red. El *input* de cada capa debe venir dado de la capa anterior excepto la primera capa, mientras que la salida de la red neuronal será el *output* de la última capa. Diseñar una red neuronal implica, entre otras cosas, definir la estrucutra general de la red, incluyendo el número de capas y el ancho de las mismas.

3.3. Evaluando la salida de una red neuronal

Para cada valor de entrada x (vector n-dimensional) se produce una salida computable por la red neuronal denotada por $\hat{y}$. Ahora, habrá que comparar la como de buena es la predicción de nuestra red neuronal $\hat{y}$ e y. Ahora es cuando entra a trabajar la función de pérdida. La **función de pérdida** mide el nivel de diferencia entre $\hat{y}$ e y, que denotamos por l. Existen decensas de funciones de pérdida en función de la tarea: clasificación binaria, multi-clasificaión, regresión... Una función de péridda típicamente compara las diferencias entre y e $\hat{y}$ sobre un número de puntos mas que sobre un único punto.

3.4. Entrenando la red neuronal

Asumiendo la misma notación, denotando por θ como el conjunto de todos los pesos y los términos *bias* de todas las capas de la red. En un principio podemos suponer que θ es inicicializada con valores aleatorios, denotado por f_{NN} a la función que representa la red neuronal. Con estos valores aleatorios, debemos calcular $\hat{y} = f_{NN}(x,\theta)$ y obtener $l(\hat{y},y)$. Luego tendríamos que calcular el grandiente de la función de pérdida y denotarla por $\nabla l(f_{NN}(x,\theta),y)$. El siguiente θ en ser usado será aquel calculado como $\theta_s = \theta_{s-1} - \alpha \cdot l(f_{NN}(x,\theta),y)$, siendo α un parámetro libre que podremos seleccionar en función de valores anteriores. Luego el θ final será aquel que nos de un valor razonalbe de $l(\hat{y},y)$.

3.5. Derivando funciones coste usando Maxima Verosimilitud

En esta sección nos centraremos en como derivar las funciones pérdida usando el método de Máxima Verosimilitud. Especificamente, veremos como noralmente usamos las funciones pérdidas en el aprendizaje profundo, con la entropía cruzada binaria, entropía cruzada y error cuadrático.

La idea de la Máxima Verosimilitud es intentar obtener el θ que maximiza una función, en general suele ser la probabilidad $P(D|\theta)$ siendo esto la *probabilida de que ocurra D condicionado a θ*. Una vez fijado θ, vemos que probabilidad hay de obtener los datos D. Lógicamente el mejor θ será el que sea capaz de reproducir los datos D, que es lo que viene a decir "maximizar $P(D|\theta)$". En función del problema planteado, de la forma de los datos D, pues tendremos una forma de $P(D|\theta)$ u otra.

3.5.1. Entropía binaria cruzada

La Entropía Binaria Cruzada viene dada por la siguiente expresión:

$$-\sum_{i=1}^{n} y_i \log f(x_i,\theta) + (1-y_i)\log(1-f(x_i,\theta)) \tag{3.2}$$

es la mejor función de pérdida cuando hablamos de clasificación binaria. Es usada cuando la red neuronal se usa para predecir la probabilidad de un resultado. En algunos casos, la capa de salida tiene una sola unidad con una función sigmoide apropiada como función de activación.

3.5.2. Entropía cruzada

La entropía cruzada viene dada por la expresión

$$-\sum_{i=1}^{n} y_i \log f(x_i, \theta) \tag{3.3}$$

es la función recomendada para la función de pérdida para la multi-clasficación. Esta función de pérdidad debería de trabajar, a priori, para predecir la probabilidad de determinados resultados de cada una de las clases. En muchos casos, la capa de salida tiene unidades *softmax*.

3.5.3. Error cuadrático

El error cuadrático viene dado por la expresión

$$\sum_{i=1}^{i} (y - \hat{y})^2 \tag{3.4}$$

y se suele usar para problemas de regresión. La capa de salida tiene una única undiad.

3.6. Tipos de Unidades/Capas/Funciones de Activación

Las propiedes de las funciones de activación más comunes:

- En teoría, cuando una función de activación es no lineal, una capa de dos redes neuronales puede *aproximar cualquier función* (siempre y cuando se le de un número adecuados entre las capas ocultas). Por eso quereos funciones no lineales en general.
- Una función de activación debería ser continuamente diferencialbe, lo que permitirá al gradiente ser calculado y poder usar métodos basados en el gradiente, lo que nos permitirá encontrar los parámetros que minimicen la función error sobre los datos.
- Una función con rango finito lleva a una activiada mucho más estable.
- Las funciones suaves son preferidas (por razones puramente empíricas).
- Deberían ser simétricas respecto al origen y comportarse como función identidad cerca del mismo $f(x) = x$.

3.6.1. Unidad Lineal

La unidad lineal es la unidad más simple, ya que transforma el *input* tal que $y = wx + b$. Tal y como indica el nombre, la unidad es lineal, y se suele usar para generar la media de una distribución gausiana. Las unidades lineales hacen que el aprendizaje basado en gradientes sea una tarea bastante sencilla.

3.6.2. Unidad Sigmoide

La Unidad Sigmoide da el siguiente valor de salida a partir de la entrada

$$y = \frac{1}{1+e^{-(wx+b)}} \tag{3.5}$$

siendola función de activación

$$f(x) = \frac{1}{1+e^{-x}} \tag{3.6}$$

Las unidades sigmoides pueen ser usadas en las capas de salida junto con la entropía binaria cruzada para problemas de clasificación binaria. La salida de esta unidad puede ser modelada como una función de Bernoulli sobre la y condicionada a x.

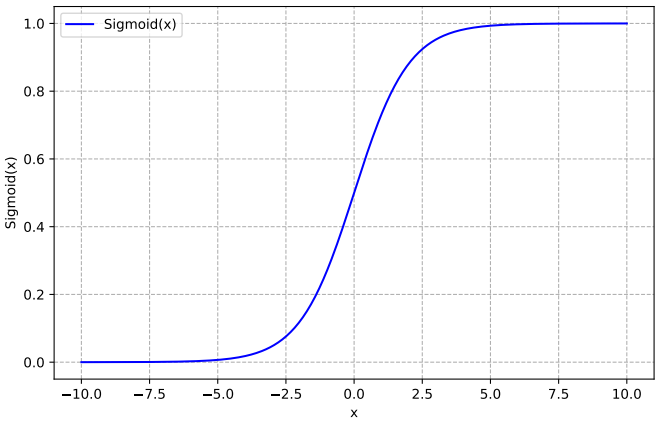

Figura 3.1: Función sigmoide.

3.6.3. Capa *Softmax*

Las capas *Softmax* (*Softmax layers*) se usa típicamente como una capa de salida para tareas de multi-clasificación junto con la función error Entropía Cruzada. Su función es normalizar las salidas de las capas anteriores de tal modo que todas sumen uno. Normalmente, las unidades de capas anteriores nos darán un valor no normalizado acerca de como el valor de entrada se relaciona o depende con determinada clásica. La capa softmax normalizada lo que hace es, efectivamente, dar el valor de probabilidad de cada clase.

3.6.4. ReLU

La *Rectified Linear Unit* (Unidad Lineal Rectificada) usada junto con la función de transformación lineal, usa la siguiente función de activación

$$f(x) = \text{máx}(0, \omega x + b), \tag{3.7}$$

Es usada en general como capa oculta. Se puede probar que las unidades ReLU dan gradientes muy consistentes, lo que mejora los métodos basados en estos.

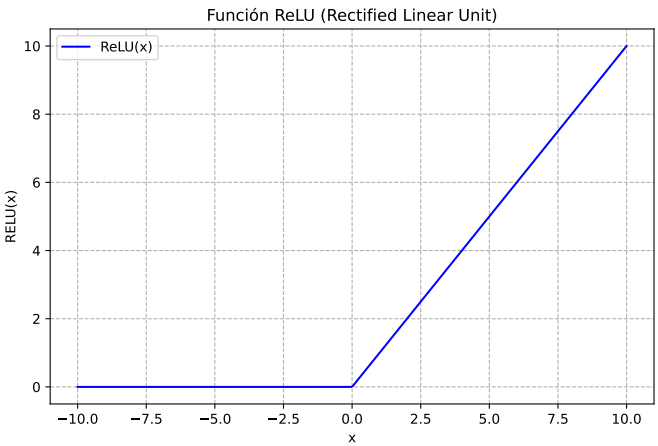

Figura 3.2: Función sigmoide.

3.6.5. Tangente Hiperbólica

La tangente hiperbólica trnaforma el *input* tal y como sigue:

$$y = \tanh(\omega x + b) \tag{3.8}$$

siendo claramente la función de activacióin

$$f(x) = \tanh(x) \tag{3.9}$$

Se suele unidad como capa/unidad oculta, y en conjunción con la función de transoformación lineal.

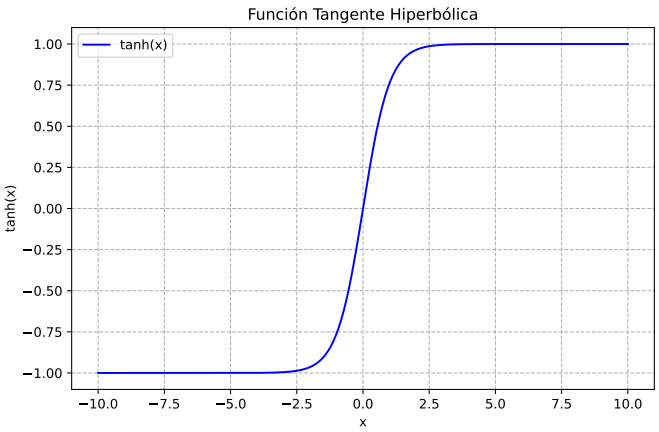

Figura 3.3: Función sigmoide.

Capítulo 4

Redes Neuronales Convolucionales

Las redes neuronales convolucionales o CNN (*Convolutional Neural Network*) son, en esencia, redes neuronales que emplen métodos convolucionales en vez de usar capas completamente conectadas. Las CNNs son increíblemente exitosas en su aplicación a problemas en los que los datos de entrada sobre los que se van a realizar predicciones tienen una topología conocida en forma de malla, como una serie temporal (que es una malla unidimensional) o una imagen (que es una malla bidimensional).

4.1. Operacion Convolucional

La operación convolucional en una dimensión consta de dos partes, un *input* $I(t)$ y un *kernel* $K(a)$, tal que la **operación convolucional** es

$$s(t) = \sum_a I(a) \cdot K(t-a) \tag{4.1}$$

Una forma equivalente

$$s(t) = \sum_a I(t-a) \cdot K(a) \tag{4.2}$$

En la literatura del aprendizaje máquina y su implementación en *software*, la convolución y la correlación cruzada son intercambiables. La esencia de la operación es qeu el Kernel, mucho más pequeño que el input. El valor de la convolción $s(t)$ será, como tal, la salida de la operación convolucion. Evidentemente podemos generalizar este resultado para *inputs* y *kernels* de varias dimensiones, tales como

$$s(m,n) = (I * K)(m,n) = \sum_a \sum_b I(a,b) \cdot K(m-a, n-b) \tag{4.3}$$

Podemos pensar en la convolución como el producto de un vector/matriz pequeño sobre un área de un valor grande (input) centrado dando como resultado un único valor. Repetiremos este proceso sobre varias "áreas" del input obteniendo un output vector/matriz.

En las imagenes Figura 4.1 y Figura 4.2 podemos ver dos diferentes tipos de redes neuronales. En este caso es evidente ver que, para el mismo número de entradas y salidas, la densamente conectada tiene muchas mas conexiones (y por tanto más pesos) que la red convolucional. Debido a esto, lógicamente las interacciones que producen las salidas serán menores, o que llamamos red de interacción escasa o dispersa (*sparse interaction*). Dado que los parámetros y pesos se comparten a lo largo de la capa

convolucional, ya que se usa siempre el mismo kernel en la interacción de una capa a otra, pues tendremos muchos menos parámetros que en una red no convolucional.

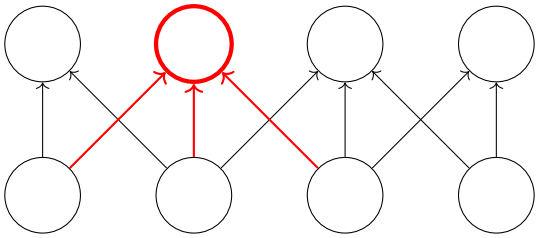

Figura 4.1: Red Neuronal Convolucional

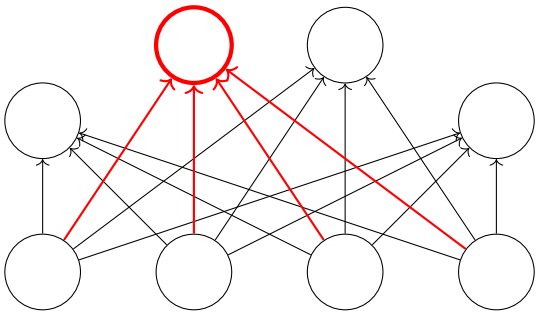

Figura 4.2: Red Neuronal con capas densamente conmunicadas.

4.2. Operacion de Agrupamiento

La **operación de Agrupamiento** o *pooling operation* es una técnica que se usa junto con la convolución en las CNN. La idea detrás de la operación de *pooling* es que la ubicación exacta de la característica no es un problema si, de hecho, esta ha sido detectada. Simplemente proporciona invariancia ante traslaciones. Por ejemplo, supongamos que la tarea en cuestión es aprender a detectar rostros en fotografías. Supongamos también que los rostros en la fotografía están inclinados (como generalmente ocurre) y que tenemos una capa de convolución que detecta los ojos. Nos gustaría abstraer la ubicación de los ojos en la fotografía de su orientación. La operación de pooling logra esto y es un componente importante de las CNN.

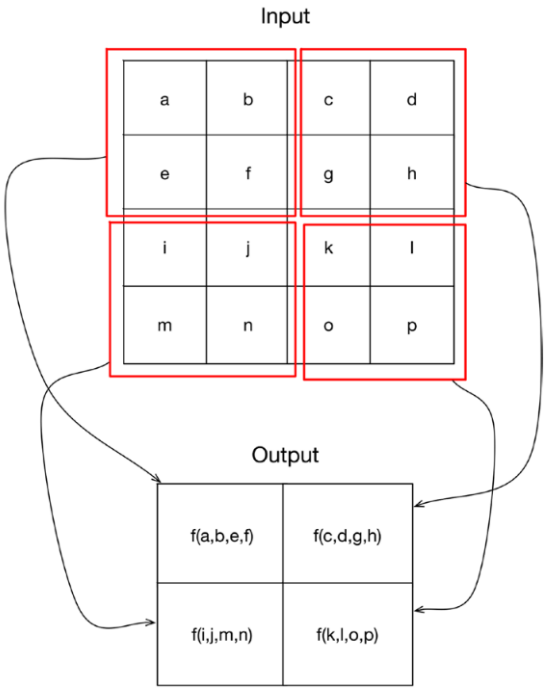

Figura 4.3: Ejemplo de una operación de Agrupamiento.

Lo que hacen estas operaciones es coger porciones de los *inputs* y aplicarles la función f, produciendo una salida. Esta función f, normalmente una operación máx (puede haber otras), actúa sobre porciones regulares (al menos en 2 dimensiones) y produce como resultado una salida dimensionalmente mucho más pequeña. En la Figura 4.3 representamos esta idea gráficamente.

4.3. Bloque básico de Convolución-Detector-Pooling

Ahora veremos como es un bloque básico en una red neuronal convolucional, donde mezclamos tanto procesos convolucionales como operaciones de agrupamiento. En la imagen Figura 4.3 vemos como una serie de inputs pasan a través de una red convolucional llegando pues a una capa o zona llamada detector (*detector stage*) que es básicamente una función de activación no-lineal. La salida normalmente se pasa a otras capas (convolucionales o densamente conectadas). Lógicamente se pueden aplicar múltiples bloques de Convolución-Detector-Pooling en paralelo, utilizando la misma entrada y produciendo múltiples salidas o mapas de características (feature maps).

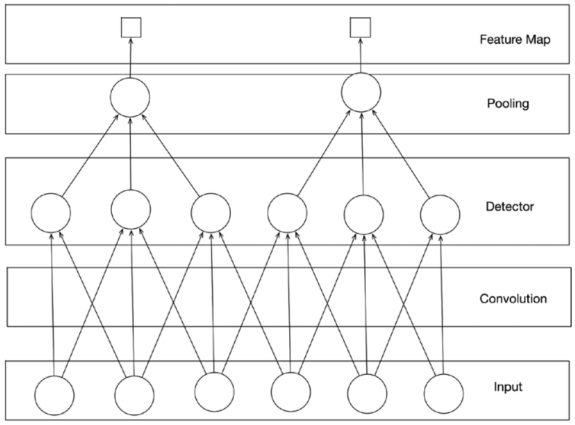

Figura 4.4: Ejemplo de una operación de un bloque básico de detector-convolución-pooling.

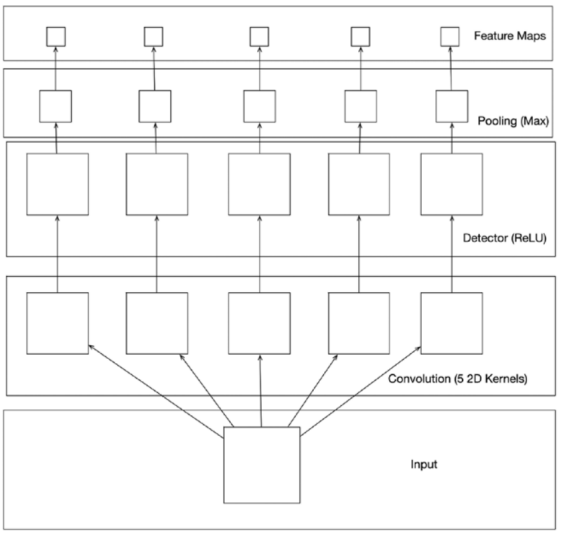

Figura 4.5: Ejemplo de Kernels dando varias salidas.

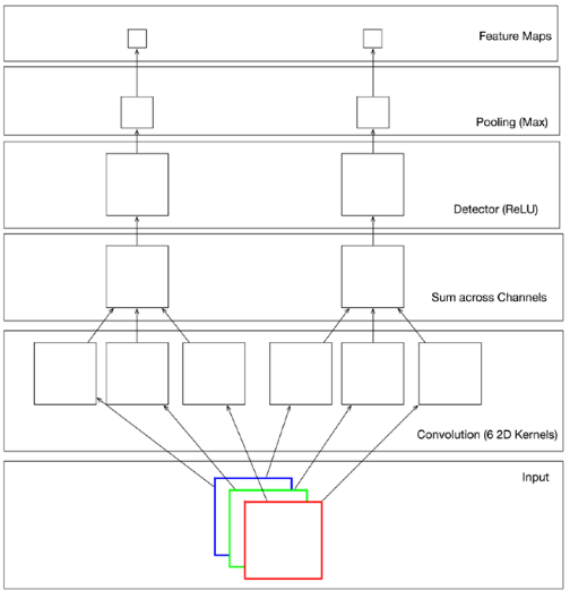

Figura 4.6: Convolución con varios canales.

4.4. Idea detrás de las redes neuronales convolucionales

A lo largo de este capítulo hemos visto los conceptos básicos de las redes neuronales convolucionales, centrándonos en la operación convolución, la operación de agrupamiento o *pooling* y como estas interactuan dentro de una CNN. Ahora bien, ¿Por qué se usan las CNNs? la primera idea que tenemos que considerar es su eficiencia. Las CNN, que en principio vienen a remplazar al menos una de las *fully connected network*, en realidad tienen una pacaidad menor que estas, pero que a su vez, al tener menos parámetros únicos, deberían ser mucho más fáciles de entrenar (o al menos eficientemente). Otro de los aspectos a considerar vienen dados opr que, en realidad, los filtros manejados por la convolución, son capaces de detectar bordes, formas... Luego además debemos pensar en la operación agrupamiento, ya que esta en realidad nos permite separar el hecho de ser detectada una característica de la ubicación exacta de la característica (borde, forma...). Un filtro que detecta líneas

rectas puede detectar este patrón en cualquier parte de la imagen, pero la operación de *pooling* conserva el hecho de que la característica fue detectada (max *pooling*). El útimo motivo es que la serie de capas de convolución y *pooling* genera las características, y una red neuronal estándar aprende la función final de clasificación o regresión. Es importante distinguir este aspecto de las CNN del aprendizaje automático tradicional. En el aprendizaje automático tradicional, un experto diseñaba manualmente las características y las introducía en una red neuronal. En el caso de las CNN, estas características o representaciones se aprenden directamente a partir de los datos.

Capítulo 5

Redes Recurrentes

Las redes neuronales recurrentes son, en esencia, es un tipo de red que contiene conexiones recurrentes, es decir, conexiones en las que la salida de una neurona en un instante se utiliza como entrada para otra neurona en el instante siguiente. Esto permite a las RNNs capturar dependencias temporales y patrones secuenciales.

Capítulo 6

Theano

Capítulo 7

Keras

Capítulo 8

PyTorch

Capítulo 9

Tensor Flow

[1]

Bibliografía

[1] Marcos Sánchez-Élez. *Introducción a la programación en VHDL*.