

# UD4-REDES NEURONALES

*Curso de Especialización en Inteligencia Artificial y Big Data*

*MP Sistemas de Aprendizaje Automático*

## 1. INTRODUCCIÓN

- 1.1. ¿Qué es una neurona?
- 1.2. ¿Qué es una red neuronal?
- 1.3. Experimentación con TensorFlow Playground
- 1.4. Funciones de activación

## 2. RN DENSAMENTE CONECTADAS (FEED FORWARD NEURAL NETWORKS)

- 2.1. Definición
- 2.2. Función de pérdida (Loss function)
- 2.3. Gradiente descendiente

## 3. RN CONVOLUCIONALES

- 3.1. Definición
- 3.2. Capa de convolución (Convolutional layer)
- 3.3. Capa de pooling (Pooling layer)

## 4. OTROS TIPOS DE RN

- 4.1. RN Recurrentes

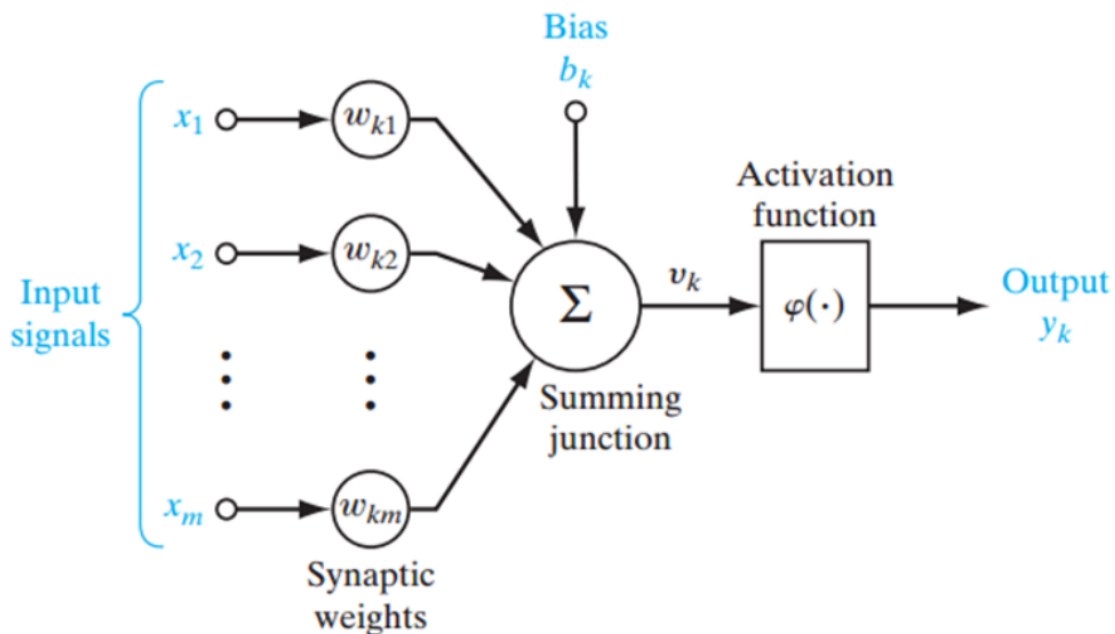
## 5. WEBGRAFÍA

## 1. INTRODUCCIÓN

### 1.1. ¿Qué es una neurona?

Una neurona artificial es una función matemática concebida como un modelo de una neurona biológica. La neurona artificial recibe una o más entradas y las suma para producir una salida. Por lo general, cada entrada se pondera por separado con un peso y la suma se pasa a través de una función no lineal conocida como función de activación.

Estructura básica de una neurona:



### 1.2. ¿Qué es una red neuronal?

Las redes neuronales artificiales (llamadas simplemente redes neuronales) son sistemas informáticos inspirados en las redes neuronales biológicas que constituyen los cerebros de los animales.

**Una red neuronal se basa en una colección de neuronas artificiales.** Cada conexión, como las sinapsis en un cerebro biológico, puede transmitir una señal a otras neuronas. Una neurona artificial recibe señales, luego las procesa y puede enviar señales a las neuronas conectadas a ella. La "señal" en una conexión es un número real, y la salida de cada neurona se calcula mediante alguna función no lineal de la suma de sus entradas.

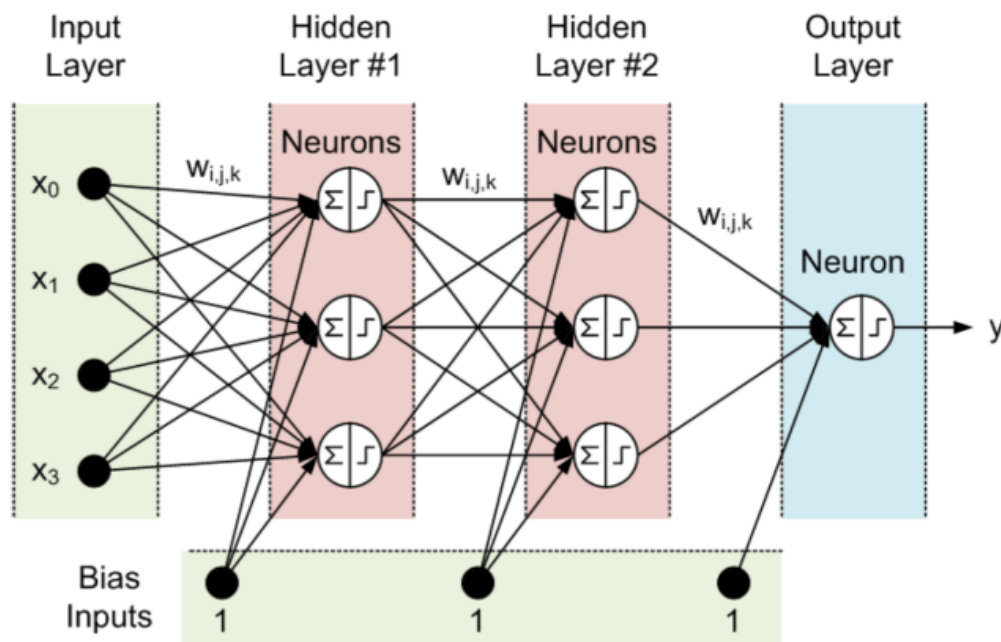
Las neuronas y las conexiones suelen tener un peso que se ajusta a medida que avanza el aprendizaje. El peso aumenta o disminuye la fuerza de la señal en una conexión. Las neuronas pueden tener un umbral tal que se envíe una señal solo si la señal agregada cruza ese umbral.

Por lo general, las neuronas se agrupan en capas con diferentes números de neuronas. Diferentes capas pueden realizar diferentes transformaciones en sus entradas. Las señales viajan desde la primera capa (la capa de entrada) hasta la última capa (la capa de salida), posiblemente después de atravesar las capas varias veces.

**La unión de todas las neuronas permite descubrir patrones complejos en los datos de entrada.**

Todos los algoritmos de Deep Learning son redes neuronales que comparten algunas propiedades básicas comunes, como que todas consisten en neuronas interconectadas que se organizan en capas.

En la siguiente figura vemos una red neuronal de 4 capas: una de entrada (input layer) que recibe los datos de entrada, una de salida (output layer) que devuelve la predicción realizada, y dos capas ocultas internas (hidden layers).



### 1.3. Experimentación con TensorFlow Playground

[TensorFlow Playground](#) es una aplicación web de visualización interactiva, escrita en JavaScript, que nos permite simular redes neuronales densamente conectadas que se ejecutan en nuestro navegador y ver los resultados en tiempo real.

Permite añadir hasta 6 capas internas con hasta 8 neuronas por capa. Al entrenar la red neuronal, vemos si lo estamos consiguiendo o no por la métrica de “Training loss”, es decir, por la función de pérdida para los datos de entrenamiento. Posteriormente, para comprobar que el modelo generaliza, se debe conseguir también minimizar la “Test loss”, es decir, el error calculado por la función de pérdida para los datos de test.

Propuestas de cambios:

- Subir el ratio of training to test data al 70%
- Dejar una única capa interna con una neurona
- Añadir 2 neuronas más a la capa interna (de forma que tenga 3 neuronas)
- Cambiar el dataset al que tiene 4 zonas cuadradas diferentes
- Cambiar el dataset al que tiene el remolino (necesitará más capas con más neuronas)
- Cambio valores de los hiperparámetros
- Adición de ruido a los datos de entrada

Conclusiones:

- **Pocas neuronas** en las capas ocultas provocarán infraajuste o **underfitting**.
- **Demasiadas neuronas** en las capas ocultas provocarán **overfitting** (la red neuronal tiene más capacidad de procesamiento de información que la cantidad de información contenida en el conjunto de entrenamiento que no es suficiente para entrenar a todas las neuronas de las capas ocultas) y **mucho más tiempo de procesamiento**.

### 1.4. Funciones de activación

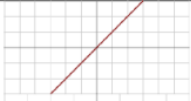




La función de activación determina, como su propio nombre indica, el nivel de activación que alcanza cada neurona una vez que ha recibido los impulsos transmitidos. Estas funciones ostentan un rol muy importante en la determinación del poder

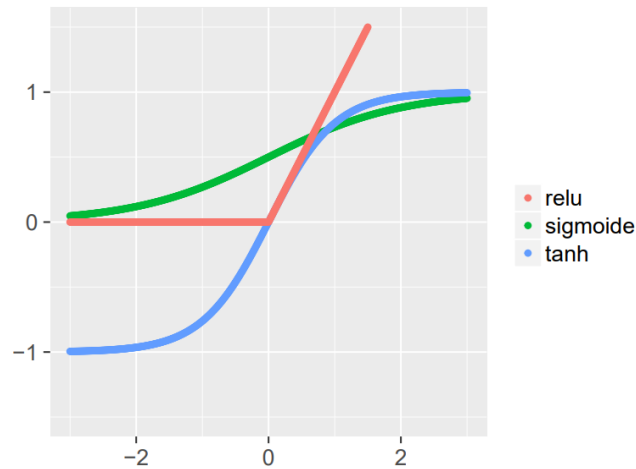
computacional de la red neuronal.

La función de activación se encarga de devolver una salida a partir de un valor de entrada, normalmente el conjunto de valores de salida en un rango determinado como (0,1) o (-1,1).

Se buscan funciones que las derivadas sean simples, para minimizar con ello el coste computacional.

Link muy explicativo sobre los diferentes tipos de funciones de activación:  
<https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/>

Name	Visualization	$f(x) =$	Notes
Linear (= Identity)		$x$	Not useful for hidden layers
Heaviside Step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	Not differentiable
Rectified Linear (ReLU)		$\begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	Surprisingly useful in practice
Tanh		$\frac{2}{1+e^{-2x}} - 1$	A soft step function; ranges from -1 to 1
Logistic ('sigmoid')		$\frac{1}{1+e^{-x}}$	Another soft step function; ranges from 0 to 1



### Función de activación Softmax

En la teoría de la probabilidad, la salida de la función softmax se puede utilizar para representar una distribución categórica, es decir, una distribución de probabilidad sobre  $K$  diferentes resultados posibles. Por tanto, la función softmax se utiliza para clasificación multiclase. Podríamos considerar la función softmax como una generalización de la función sigmoid que permite clasificar más de dos clases. La función de activación softmax nos garantiza que todas las probabilidades estimadas son entre 0 y 1 y que suman 1.

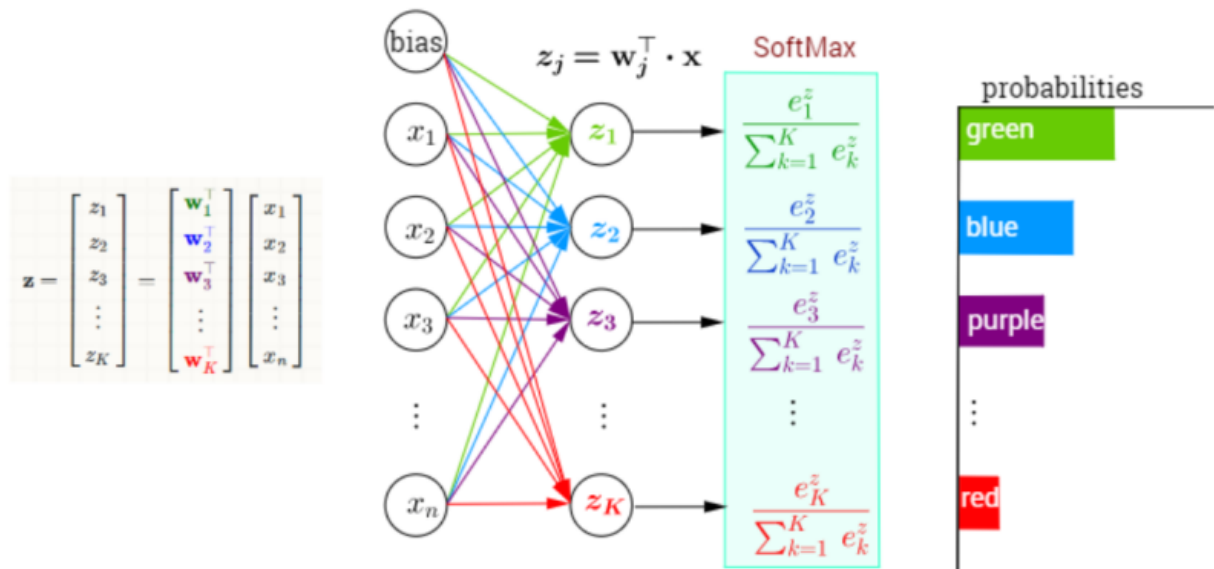
A nivel matemático, softmax usa el valor exponencial de las evidencias calculadas y, luego, las normaliza de modo que sumen uno, formando una distribución de probabilidad.

$$\sigma : \mathbb{R}^K \rightarrow [0, 1]^K$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

Ejemplo de la función de activación softmax:

## Multi-Class Classification with NN and SoftMax Function



## 2. RN DENSAMENTE CONECTADAS (FEED FORWARD NEURAL NETWORKS)

### 2.1. Definición

Cuando todas las neuronas de una capa están conectadas con todas las neuronas de la capa anterior, la capa se denomina completamente/densamente conectada.

**Ejemplos disponibles en Drive:** demoKeras1.ipynb, demoKeras2.ipynb

Etapas para crear una RN densamente conectada:

1. **Creación del modelo y definición de cada una de las capas.**
  - `model = new Sequential()`
  - `model.add(Dense(<número de neuronas>, activation=<función de activación ('sigmoid', 'softmax', 'relu'...))`
2. **Configuración del proceso de aprendizaje: método compile**
  - **función de coste (loss function):** evalúa el grado de error entre las salidas calculadas y las salidas deseadas de los datos de entrenamiento. El objetivo está en reducir dicho valor en cada iteración. Explicación en el apartado 2.2.
  - **optimizador (optimizer):** es la manera que tenemos de indicar los detalles del algoritmo de optimización que permite a la red neuronal calcular los pesos de los

parámetros durante el entrenamiento a partir de los datos de entrada y de la función de coste definida. Habitualmente utilizaremos la *sgd* (explicada en el apartado 2.3)

- **métrica (metrics):** es la que usaremos para monitorizar el proceso de aprendizaje y prueba de nuestra red neuronal. Si indicamos “accuracy” solo tendremos en cuenta la fracción de datos que son correctamente clasificados, es decir, la proporción entre las predicciones correctas que ha hecho el modelo del total de predicciones)
3. **Entrenamiento con los datos de training: método fit**
    - **epochs:** número de veces que usaremos todos los datos en el proceso de aprendizaje.
    - **verbose:** permite ver el avance del entrenamiento así como una estimación de cuánto tarda cada época.
  4. **Evaluación con los datos de testing: método evaluate:**

Para redes neuronales que clasifiquen, volveremos a referirnos a la **matriz de confusión**:

		Predicted condition	
Total population = P + N		Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

*TP:* cantidad de positivos que fueron clasificados correctamente como positivos por el modelo.

*TN:* cantidad de negativos que fueron clasificados correctamente como negativos por el modelo.

*FN:* cantidad de positivos que fueron clasificados incorrectamente como negativos.

*FP:* cantidad de negativos que fueron clasificados incorrectamente como positivos.

- Si utilizamos la precisión (accuracy), conoceremos la proporción entre las predicciones correctas que ha hecho el modelo del total de predicciones, ya que  $\text{Presicion} = (TP + TN) / (TP + TN + FP + FN)$
- Si utilizamos el recall: sabremos como de bien el modelo evita los falsos negativos,



ya que  $\text{Recall} = \text{TP}/(\text{TP}+\text{FN})$

## 5. Generación de predicciones: método predict

## 2.2. Función de pérdida (Loss function)

Existen diferentes métricas del error según trabajamos con problemas de regresión o clasificación.

Métricas para problemas de **regresión**:

- Error cuadrático medio:  $(\frac{1}{n} \sum (y_i - \hat{y}_i)^2)$
- Raíz cuadrada del error cuadrático medio

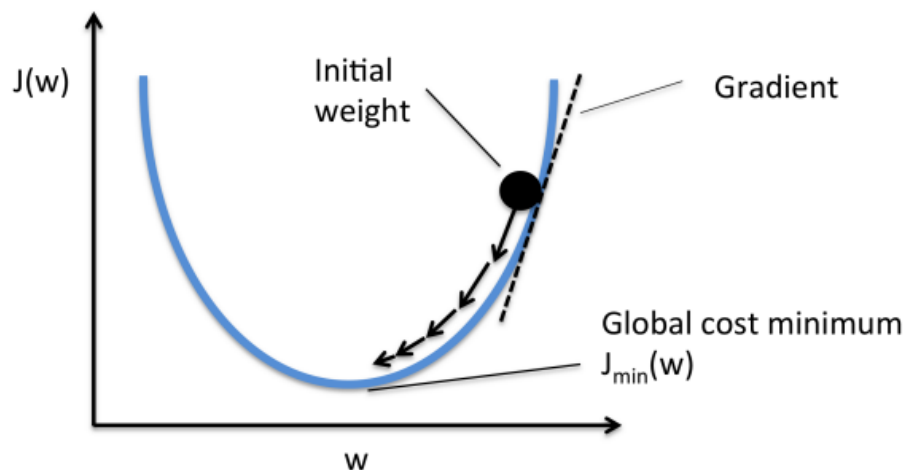
Métricas para problemas de **clasificación**:

- Logistic Loss
- Logarithmic Loss:  $-\frac{1}{N} \sum \sum y_{ij} \log(p_{ij})$
- Accuracy

## 2.3. Gradiente descendiente

El descenso de gradiente es un algoritmo de optimización de primer orden.

Encuentra un mínimo local de una función usando el descenso de gradiente, uno da pasos proporcionales al negativo del gradiente de la función en el punto actual.



*Ilustración del descenso de gradiente en una serie de conjuntos de niveles.*

La función de pérdida toma las predicciones de la red y las compara con los targets objetivo. De esta forma, calcula una puntuación de distancia, capturando cómo de bien

funciona la red neuronal.

El truco fundamental en el aprendizaje profundo es utilizar esta puntuación como una señal de retroalimentación para ajustar un poco el valor de los pesos, en una dirección que reducirá la puntuación de pérdida para el lote actual de ejemplos (gradiente negativo).

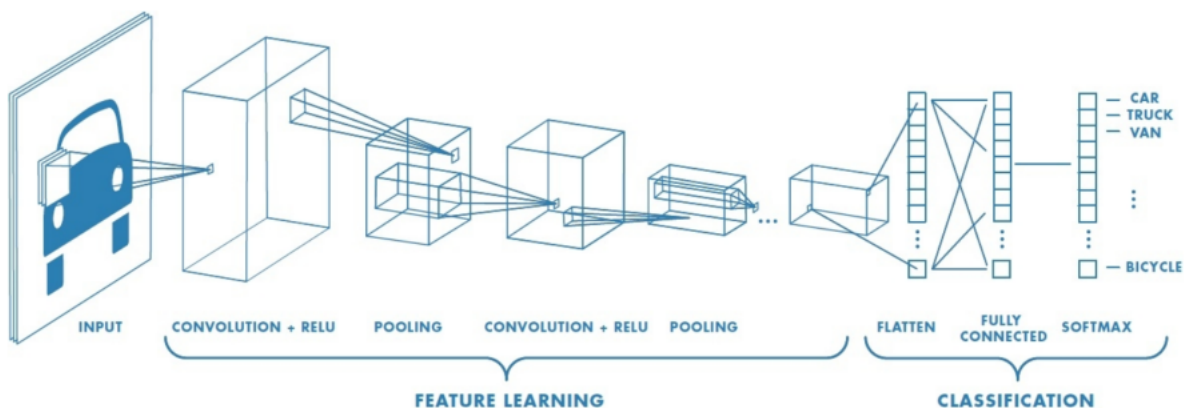
El gradiente ( $\nabla J(\theta)$ ) se calcula por retropropagación.

Calcula el gradiente de una función de pérdida con respecto a todos los pesos en la red (regla de la cadena) y lo usa para actualizar los pesos para minimizar la función de pérdida.

### 3. RN CONVOLUCIONALES

#### 3.1. Definición

- Similares a las Densamente Conectadas.
- El modelo de redes neuronales convolucionales (CNN) se puede aplicar a tareas de reconocimiento visual.
- La arquitectura de una CNN está diseñada para aprovechar la estructura de matriz de los datos.
- Jerarquía de representaciones con creciente nivel de abstracción.
- Cada etapa es un tipo de transformación de característica entrenable. Por ejemplo, una primera capa convolucional aprende elementos básicos como aristas, y una segunda capa convolucional aprende patrones compuestos de elementos básicos aprendidos en la capa anterior. Y así sucesivamente en cada capa hasta ir aprendiendo patrones muy complejos.

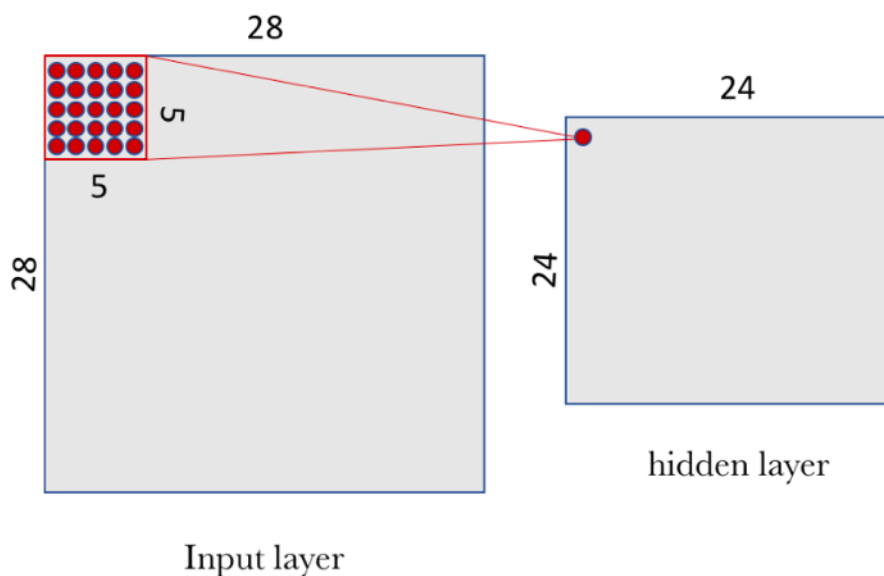


### 3.2. Capa de convolución (Convolutional layer)

La diferencia fundamental entre una capa densamente conectada y una convolucional es que **la capa densa aprende patrones globales en su espacio global de entrada, mientras que la capa convolucional aprende patrones locales dentro de la imagen en pequeñas ventanas de dos dimensiones.**

Tomemos como referencia el caso del MNIST, dónde como entrada de nuestra red neuronal podemos pensar en un espacio de neuronas de dos dimensiones  $28 \times 28$  (height=28, width=28, depth=1). Una primera capa de neuronas ocultas conectadas a las neuronas de la capa de entrada que hemos comentado realizarán las operaciones convolucionales que acabamos de describir. Pero **no se conectan todas las neuronas de entrada con todas las neuronas de este primer nivel de neuronas ocultas, sino que solo se hace por pequeñas zonas localizadas del espacio de las neuronas de entrada que almacenan los píxeles de la imagen.**

Si pensamos en una ventana del tamaño de  $5 \times 5$ , iremos recorriendo toda la capa de  $28 \times 28$  de entrada que contiene la imagen. **Esta ventana se va deslizando a lo largo de toda la capa de neuronas. Por cada posición de la ventana hay una neurona en la capa oculta que procesa esta información.**



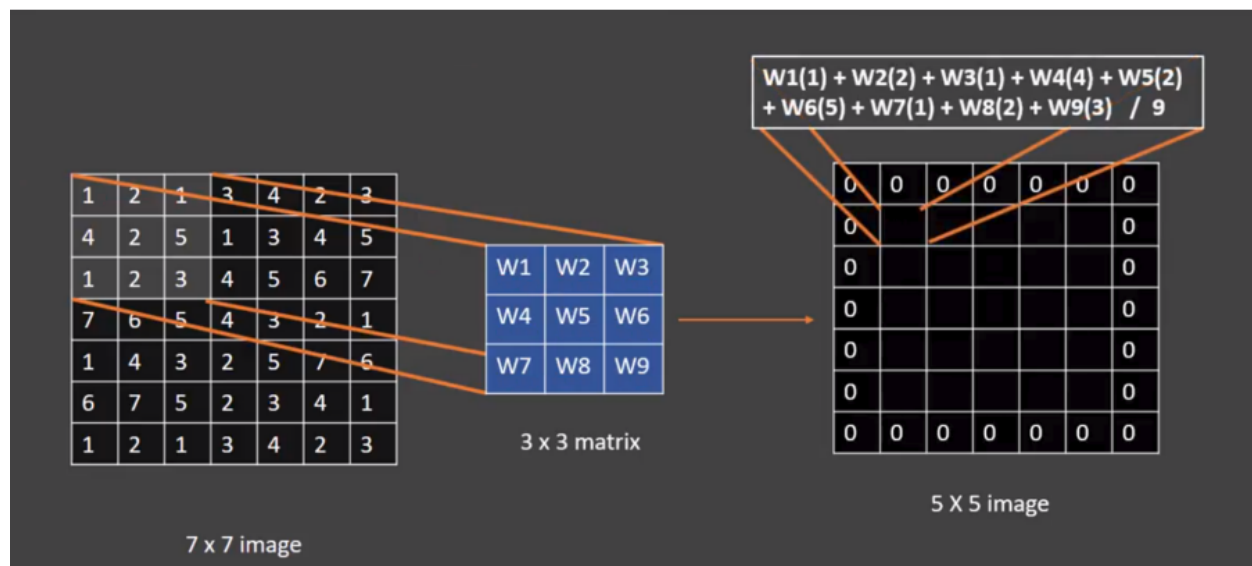
Visualmente, empezamos con la ventana en la esquina arriba-izquierda de la imagen, y

esto le da la información necesaria a la primera neurona de la capa oculta. Después, deslizamos la ventana una posición hacia la derecha para “conectar” las 5×5 neuronas de la capa de entrada incluidas en esta ventana con la segunda neurona de la capa oculta. Y así, sucesivamente, vamos recorriendo todo el espacio de la capa de entrada, de izquierda a derecha y de arriba abajo.

Observemos que **si tenemos una entrada de 28×28 píxeles y una ventana de 5×5 esto nos define un espacio de 24×24 neuronas en la primera capa del oculta**, debido a que solo podemos mover la ventana 23 neuronas hacia la derecha y 23 hacia abajo antes de chocar con el lado derecho (o inferior) de la imagen de entrada.

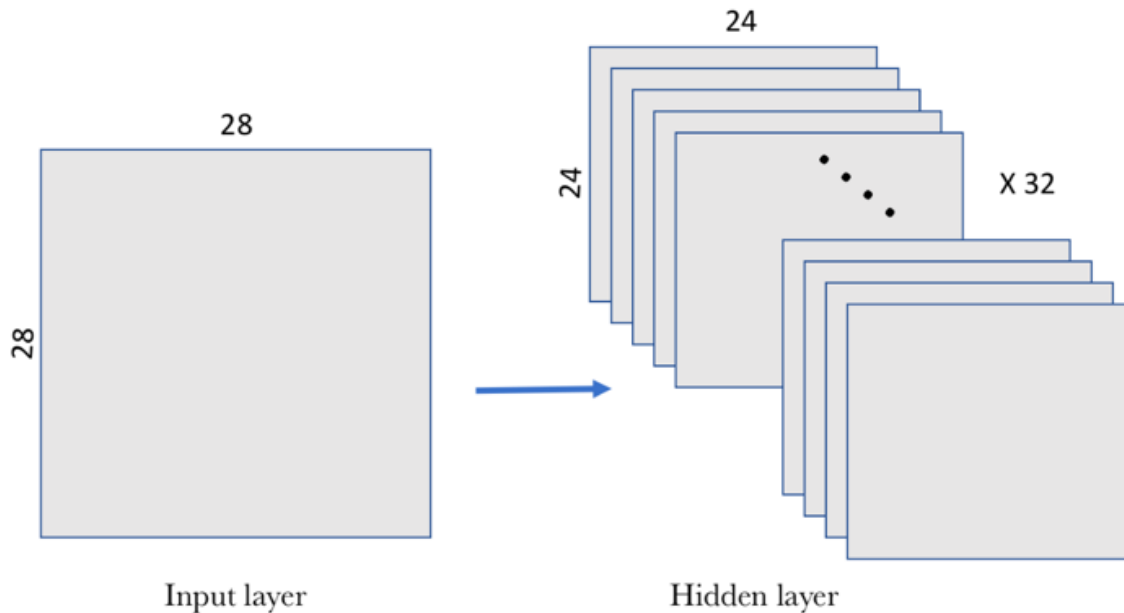
En nuestro caso de estudio, y siguiendo el formalismo ya presentado previamente, para “conectar” cada neurona de la capa oculta con las 25 neuronas que le corresponden de la capa de entrada usaremos un valor de sesgo  $b$  y una matriz de pesos  $W$  de tamaño 5×5 que llamaremos **filtro** (o kernel/filter en inglés).

En la siguiente imagen, vemos otro ejemplo partiendo de una capa de entrada de 49 neuronas (imagen de 7x7) y un filtro con ventana de 3x3, generará una capa oculta resultante de 25 neuronas (matriz de 5x5) con un **padding** de relleno a ceros alrededor del margen de la imagen (mejora el resultado del barrido que se realiza en la ventana que se va deslizando)



Es muy importante tener en cuenta que **en las redes convolucionales se usa el mismo filtro (la misma matriz  $W$  de pesos y el mismo sesgo  $b$ ) para todas las neuronas de la capa oculta.**

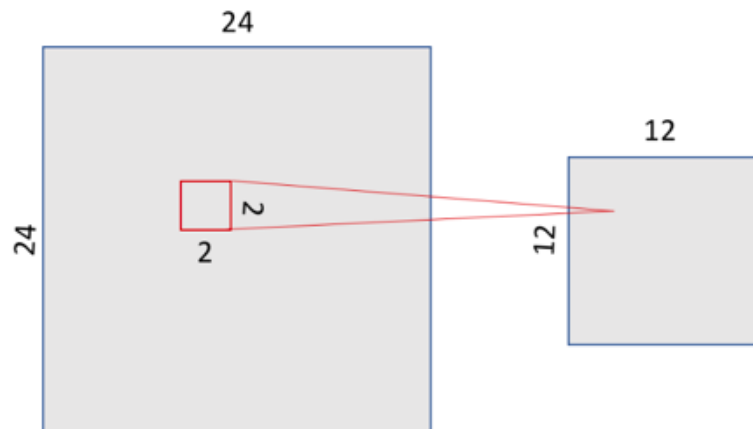
Pero un filtro definido por una matriz  $W$  y un sesgo  $b$  solo permiten detectar una característica concreta en una imagen; por tanto, **para poder realizar el reconocimiento de imágenes se propone usar varios filtros a la vez, uno para cada característica que queramos detectar**. En el siguiente ejemplo, vemos que se aplican 32 filtros, donde cada filtro recordemos que se define con una matriz  $W$  de pesos compartida de  $5 \times 5$  y un sesgo  $b$ .



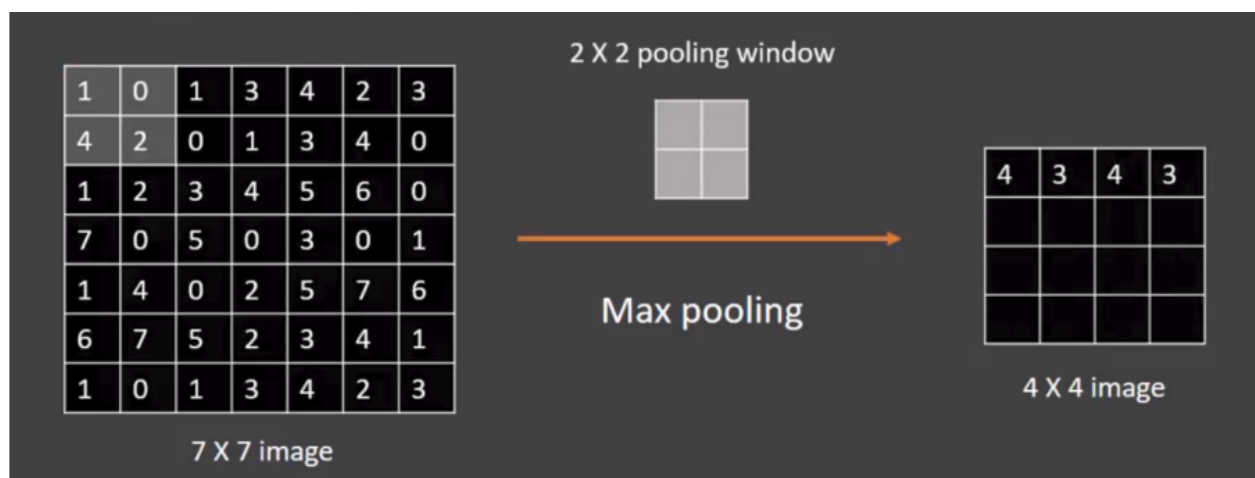
### 3.3. Capa de pooling (Pooling layer)

Además de las capas convolucionales que acabamos de describir, las redes neuronales convolucionales acompañan a la capa de convolución con unas capas de pooling, que suelen ser aplicadas inmediatamente después de las capas convolucionales. **Su función es reducir progresivamente el tamaño espacial de la representación para reducir la cantidad de parámetros y computación en la red. La capa de agrupación opera en cada mapa de características de forma independiente.**

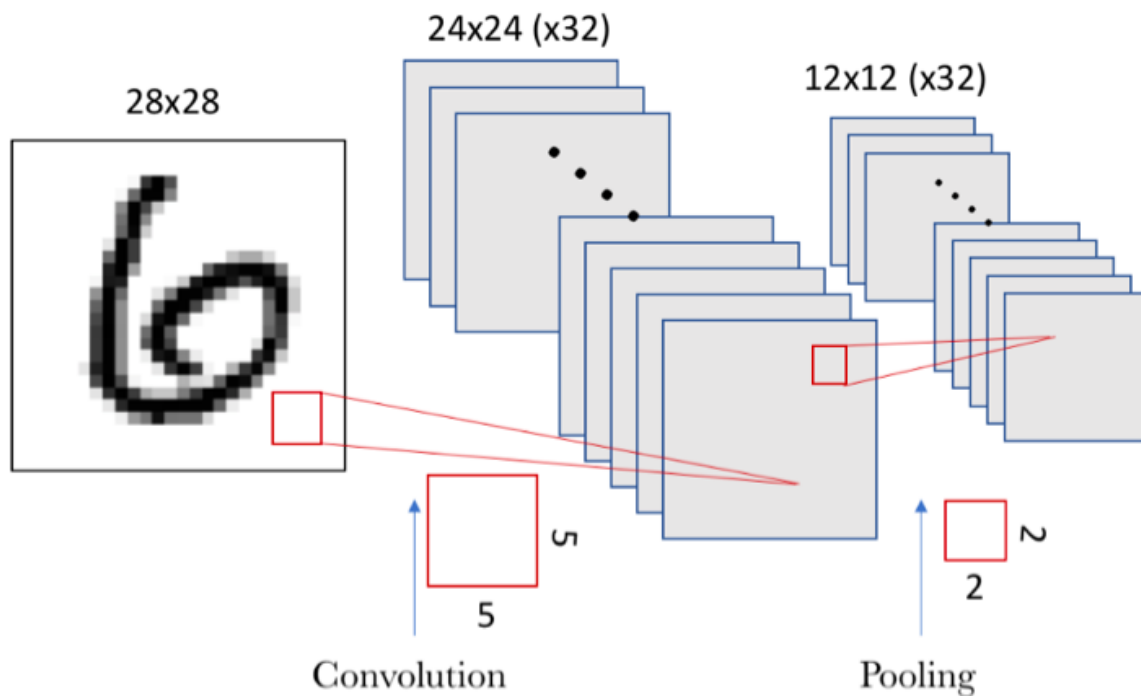
Hay varias maneras de condensar la información, pero una habitual, y que usaremos en nuestro ejemplo, es la conocida como **max-pooling**. En el siguiente ejemplo, vemos que se define una ventana de entrada de  $2 \times 2$  y se queda con el valor máximo de los existentes en la ventana. De esta forma, dividimos por 4 el tamaño de la salida de la capa de pooling, quedando una imagen de  $12 \times 12$ .



En la siguiente captura tenemos una captura de una imagen 7x7 donde se le aplica una ventana 2x2 con un **stride** (longitud del paso de avance) de 2 en lugar de 1 y por tanto obtenemos un mapa de caracteres de 4x4



En la siguiente imagen, vemos gráficamente el resultado de combinar la convolución con pooling:



## 4. OTROS TIPOS DE RN

### 4.1. RN Recurrentes

Limitación con Feed Forward Neural Networks (y CNN):

- No diseñadas para datos secuenciales.
- Dimensión fija de entradas.
- No modela la memoria.

Las redes neuronales recurrentes permiten:

- Para mapear predicciones en secuencias.
- Esto crea un estado interno de la red que le permite exhibir relaciones temporales dinámicas.
- Aplicaciones: subtítulos de imágenes, análisis de sentimientos, respuesta a preguntas, reconocimiento de voz, series temporales, generación de música, traducción automática, vehículos autónomos, ...

## 5. WEBGRAFÍA

- Libro “Python Deep Learning: Introducción práctica con Keras y TensorFlow 2” de Jordi Torres de la editorial Marcombo.
- “Introduction to Deep Learning with TensorFlow and Keras libraries with some examples in biomedical research” de Roger Borràs. Universitat Autònoma de Barcelona
- [100 Days of ML Code](#)