Universidad
Francisco de Vitoria
UFV Madrid

Inteligencia Artificial II

Tema 5:
Modelos Deep Learning

Departamento de Ingeniería Informática
Universidad Francisco de Vitoria

Álvaro José García Tejedor
Curso 2019/20

Objetivos del tema

- Ubicación
  - Bloque II: **COMPUTACION NEURONAL**
    - Tema 2: *Aprendizaje no supervisado: Aprendizaje competitivo*
    - Tema 3: *Aprendizaje supervisado: Modelos lineales*
    - Tema 4: *Aprendizaje supervisado: Modelos no lineales*
    - Tema 5: ***Modelos Deep Learning***

- Objetivos
  - Entender el Deep Learning y las arquitecturas DNN, su uso y aplicación
  - Comprender cómo es el aprendizaje en las DNN y los problemas que presenta
  - Estudiar el modelo de Red Neuronal Convolutiva, su arquitectura y sus aplicaciones
  - Estudiar el modelo de autoencoders y su arquitectura

Contenido

1. Introducción
2. Deep Learning
    1. Definición
    2. Historia
    3. Arquitecturas DNN
    4. Tipos de aprendizaje
3. Entrenamiento
    1. El problema
    2. Soluciones
4. Convolutional Neural Networks
    1. Convolución
    2. Arquitectura
    3. Aplicaciones
5. Deep Autoencoders
    1. Definición
    2. Procesamiento
    3. Tipos

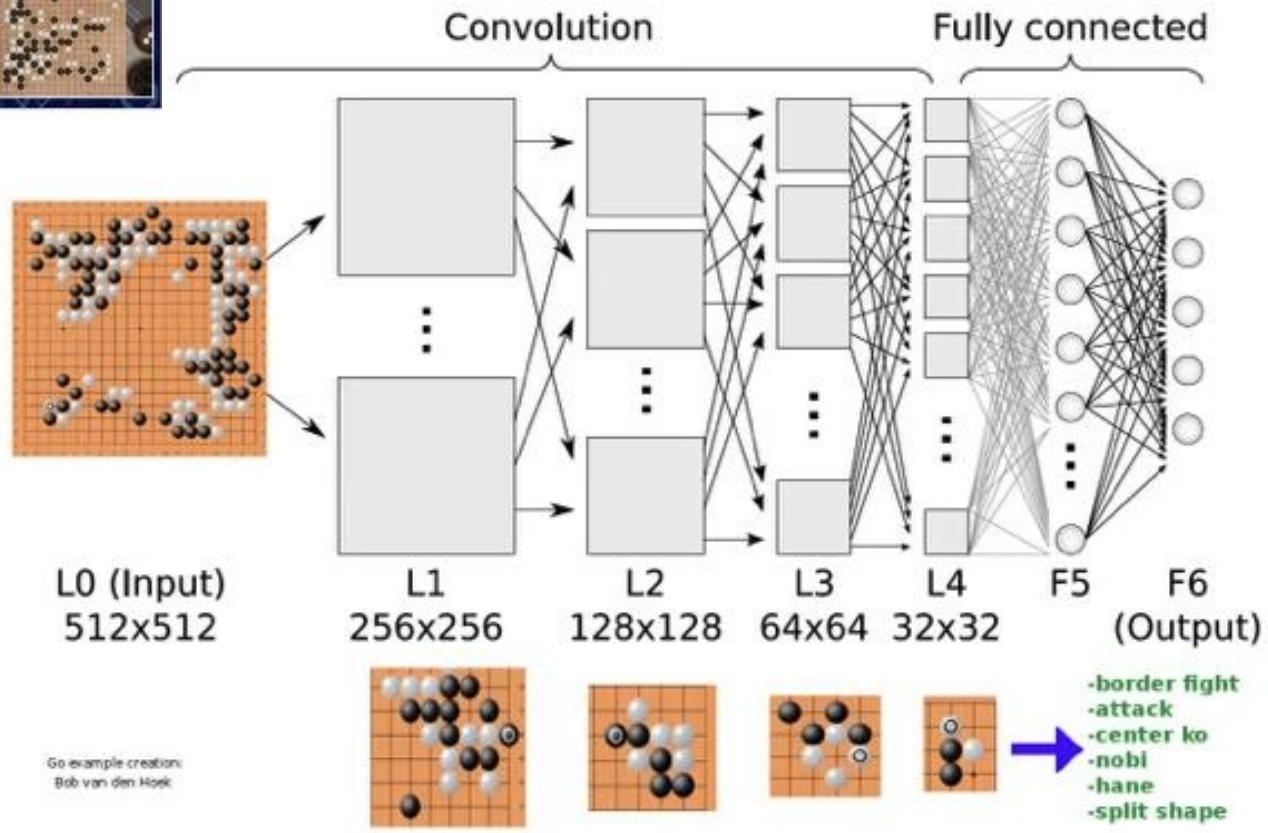
1. Introducción
2. Deep Learning
    1. Definición
    2. Historia
    3. Arquitecturas DNN
    4. Tipos de aprendizaje
3. Entrenamiento
    1. El problema
    2. Soluciones
4. Convolutional Neural Networks
    1. Convolución
    2. Arquitectura
    3. Aplicaciones
5. Deep Autoencoders
    1. Definición
    2. Procesamiento
    3. Tipos

1. Introducción

- Problema de los algoritmos de retropropagación: el error se va diluyendo de forma exponencial a medida que atraviesa capas hacia la capa de entrada
  - En una red muy profunda sólo las últimas capas se entrenan mientras que las primeras apenas sufren cambios.
  - A veces compensa utilizar redes con pocas capas ocultas que contengan muchas neuronas en lugar de redes con muchas capas ocultas que contengan pocas neuronas.

- Esto era así hasta que se desarrollaron los algoritmos y los modelos neuronales de *Deep Learning* (DL) que están formados por multiples capas
  - Un MLP con más de 2 capas ocultas ya es un modelo profundo

1. Introducción



1. Introducción
2. Deep Learning
   1. Definición
   2. Historia
   3. Arquitecturas DNN
   4. Tipos de aprendizaje
3. Entrenamiento
   1. El problema
   2. Soluciones
4. Convolutional Neural Networks
   1. Convolución
   2. Arquitectura
   3. Aplicaciones
5. Deep Autoencoders
   1. Definición
   2. Procesamiento
   3. Tipos

2.1 Definición de Deep Learning

Aprendizaje Profundo o **Deep Learning**

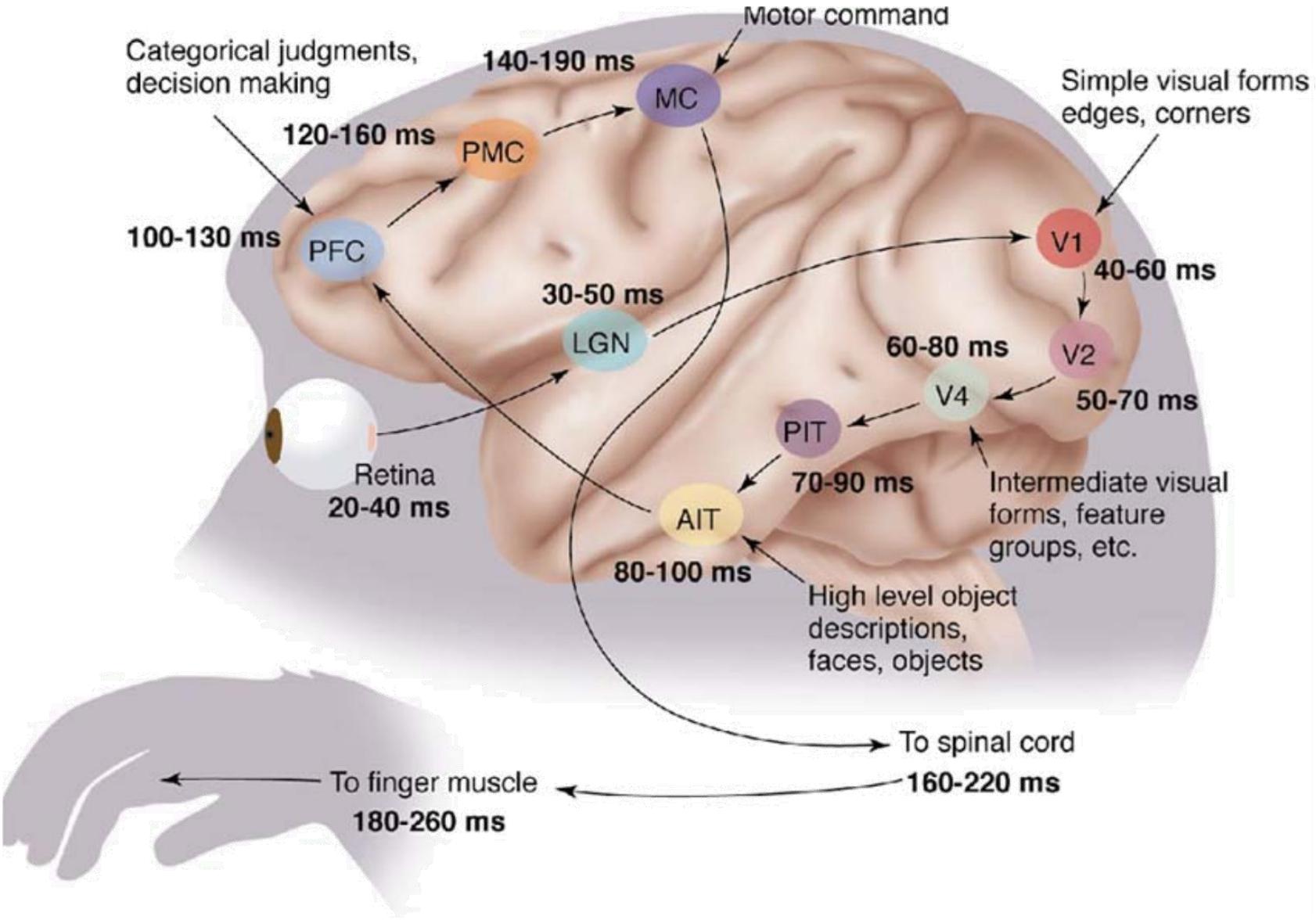
Área del *Machine Learning* que utiliza diferentes algoritmos de aprendizaje automático para modelar abstracciones de datos de alto nivel usando arquitecturas jerárquicas llamadas Redes Neuronales Profundas (*DNN*).

- Concepto de *profundidad*
  - Es la profundidad del grafo o dibujo correspondiente a una arquitectura, es decir, el camino mas largo desde un nodo de entrada hasta un nodo de salida.
  - Cuando el conjunto de elementos computacionales son neuronas, la profundidad corresponde al numero de capas de la red neuronal.

2.1 Definición de Deep Learning

- DL se basa en modelos computacionales que exhiben características similares a las del neocortex
  - El cerebro humano es una arquitectura neuronal profunda
  - El desarrollo neocortical (propuesto en 1990) fue simulado por modelos computacionales similares a las actuales DNN (*neocognitrón* de Fukushima)
  - Al igual que el neocórtex, las DNN emplean una jerarquía de filtros apilados en las que cada capa toma información de un filtro anterior (o el entorno) y pasa su salida a otras capas

2.1 Definición de Deep Learning



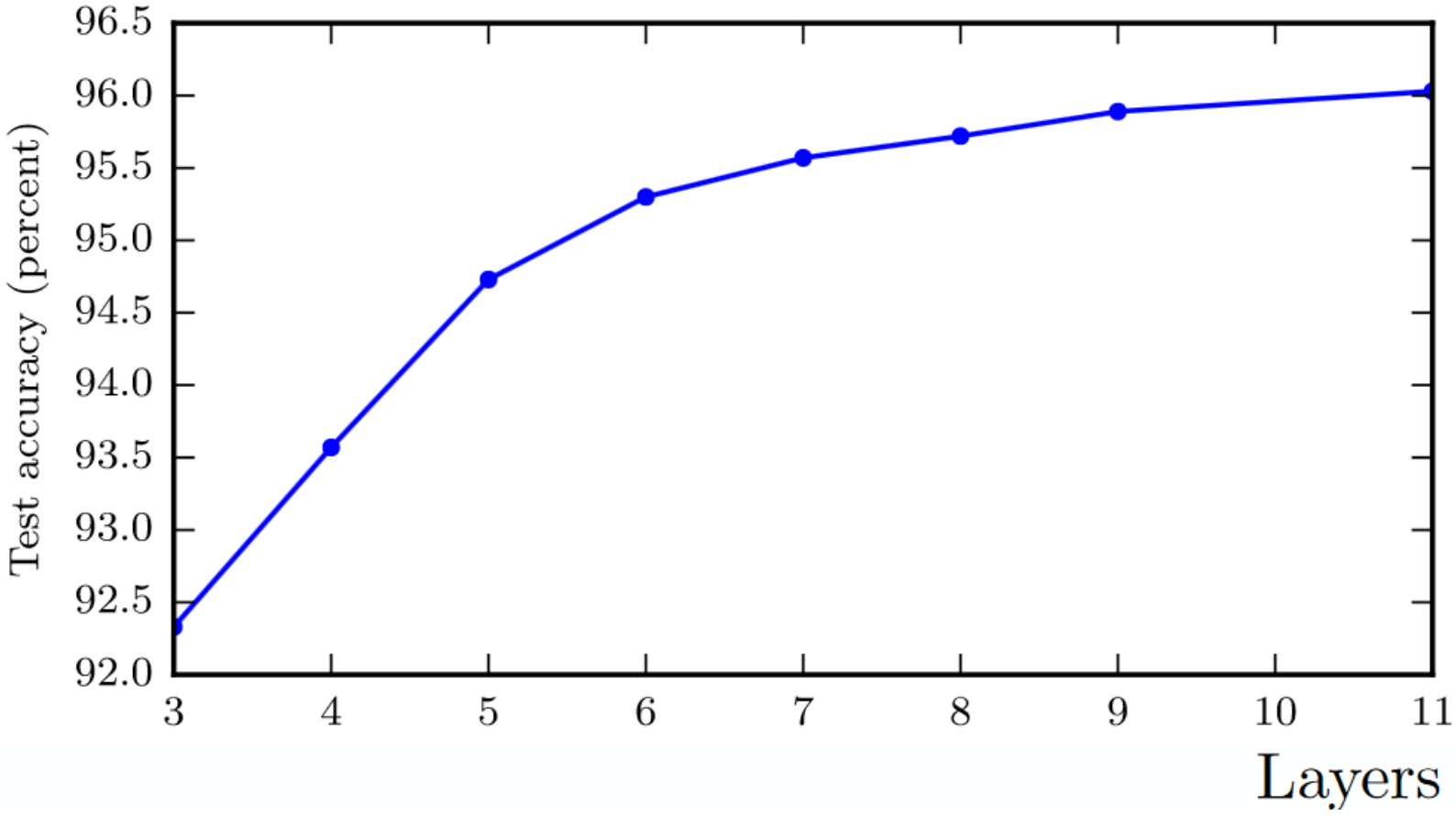
2.1 Definición de Deep Learning

- El Teorema de Aproximación Universal dice que *una única capa oculta es suficiente para aproximar* (que no aprender) *cualquier función con el grado de precisión deseado.*
  - Es decir, puede encontrar la relación entre los pares d entrada/salida del conjunto de entrenamiento
  - Pero no se garantiza que lo pueda hacer para pares nuevos (no garantiza que generalice)

- Entonces… ¿Por qué DL?
  - Las redes superficiales necesitan (exponencialmente) más anchura
  - Las redes superficiales se sobreentrenan más fácilmente
  - Las redes generalizan mejor cuanto mayor es la profundidad

2.1 Definición de Deep Learning

- Generalización en modelos DNN
  - Mejor generalización con mayor profundidad

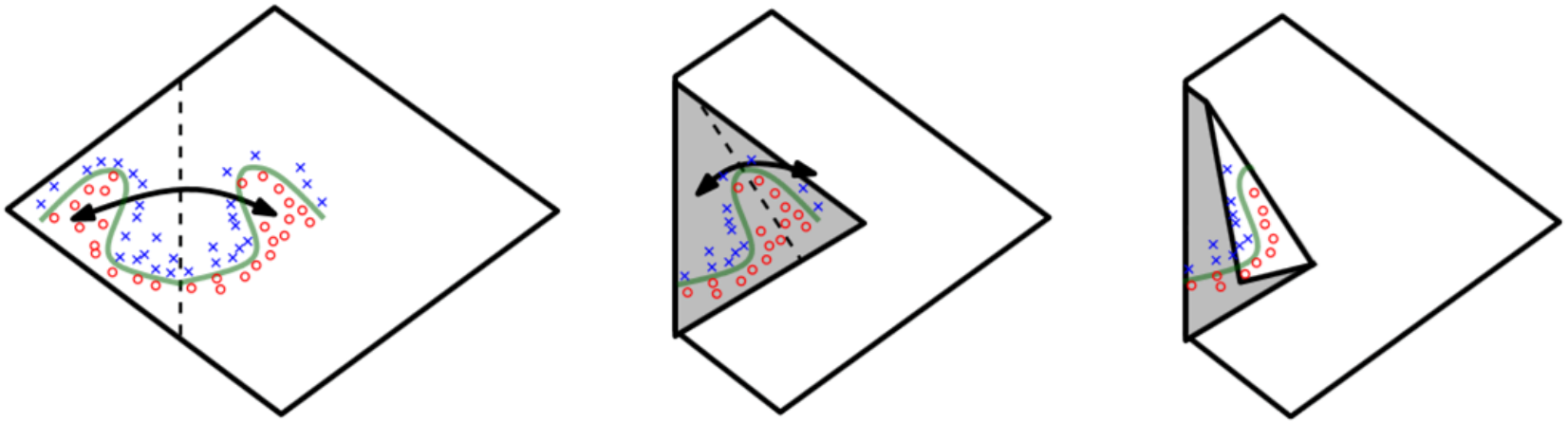


2.1 Definición de Deep Learning

- Generalización en modelos DNN
  - El algoritmo de entrenamiento encuentra la mejor representación que puede *para cada capa* comenzando desde la de entrada
  - Esto permite computar características mucho más complejas de las entradas
    - Cada capa oculta computa una transformación no lineal de la salida de la capa anterior
    - Cada capa oculta aprende a representar datos con los conceptos aprendidos de las anteriores
    - Por lo tanto, un DNN tiene más capacidad de representación que una superficial (puede aprender funciones más complejas)

2.1 Definición de Deep Learning

- Cuando se entrena una red profunda es importante usar una función de activación no lineal *f* en cada capa oculta.
  - Múltiples capas de funciones lineales calcularían solo una función lineal de la entrada
  - La composición conjunta de múltiples funciones lineales da como resultado otra función lineal
  - Por lo tanto no serían más expresivas que el uso de una sola capa de unidades ocultas.



2.2 Historia

- Las técnicas DL existen desde hace bastante tiempo aunque no se consideraban viables por:
  - La no disponibilidad de grandes cantidades de datos para entrenar a los modelos.
  - No tener algoritmos de entrenamiento que permitan entrenar las capas más profundas.
  - No tener HW en los ordenadores lo suficientemente potente como para la ejecución de los modelos.

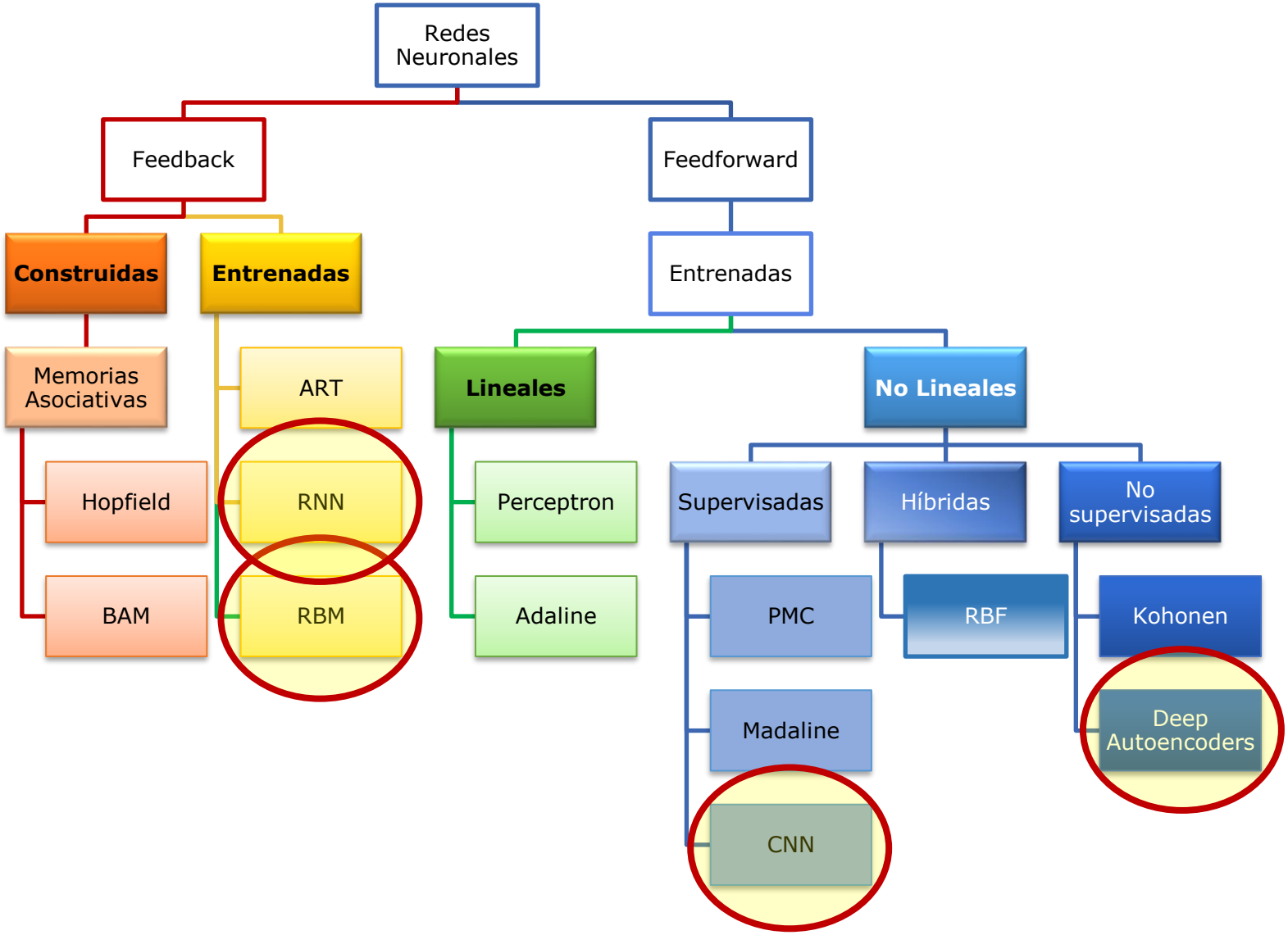
En los últimos años estos problemas se han visto solventados

- El primer éxito en el uso del aprendizaje profundo se debe a Geoffrey Hinton (2006) quien introdujo las *Redes de Creencia Profunda* (DBN) utilizando en cada capa de la red una *Maquina de Boltzmann Restringida* (RBM) para la asignación inicial de los pesos sinápticos

2.2 Historia

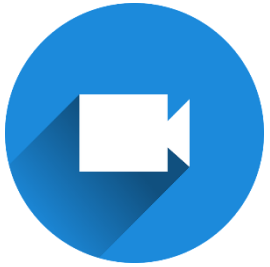
- Posteriormente se han desarrollado varias técnicas que permiten el aprendizaje en redes profundas, basadas en el *descenso estocástico de gradiente* + *backpropagation*
  - Estas técnicas permiten el entrenamiento de redes muy profundas (no hay problema en 5-10 capas ocultas)
  - Estas redes profundas se comportan mucho mejor que los redes poco profundas (MLP con una sola capa oculta)

2.3 Arquitecturas DNN



2.3 Arquitecturas DNN

- Redes Neuronales Convolucionales (CNN)
- Deep Auto-Encoders (DAE)
  - Sparse autoencoders (SAE)
  - Denoising autoencoders (DAE)
  - Contractive autoencoders (CAE)
- Máquinas de Boltzmann Restringidas (RBM)
  - Deep Belief Networks (DBN)
  - Máquinas de Boltzmann Profundas (DBM)
- Redes Neuronales Recurrentes (RNN) Feedback
  - Más de 13 modelos, incluyendo Hopfield y BAM (no DNN)
  - Fully recurrent (Basic RNN)
  - Long Short-Term Memory (LSTM)
  - Gate Recurrent Unit (GRU)
- Generative Adversarial Networks (GAN)



2.4 Tipos de aprendizaje

Aprendizaje no supervisado

- Cuando se entrena con datos sin etiquetar, cada capa de neuronas en una red profunda aprende características automáticamente al tratar de reconstruir la entrada de la que extrae sus muestras, minimizando la diferencia entre las conjeturas de la red y la distribución de probabilidad de los datos de entrada en sí.
- En el proceso, estas redes aprenden a reconocer las correlaciones entre ciertas características relevantes y los resultados óptimos.
- Tipos
  - Autoencoders
  - GAN
  - RBM

2.4 Tipos de aprendizaje

Aprendizaje supervisado

- El entrenamiento con datos etiquetados se puede aplicar a datos no estructurados, lo que le da acceso a mucha más información.
  - Conclusión: con cuantos más datos se pueda entrenar una red, más precisa será (malos modelos entrenados en una gran cantidad de datos pueden superar a buenos modelos poco entrenados).
- Una técnica muy habitual en *Deep Learning* consiste en empezar el entrenamiento supervisado, en lugar de empezar con pesos al azar, con pesos pre-entrenados, especialmente para las primeras capas.
- Tipos
  - CNN
  - RNN

2.4 Tipos de aprendizaje

Aprendizaje por refuerzo

- Modelos orientados a objetivos: aprenden cómo alcanzar un objetivo complejo o maximizar una dimensión particular a lo largo de muchos pasos
  - Ejemplo: maximizar los puntos ganados en un juego a lo largo de muchos movimientos.
  - Al igual que un niño, estos algoritmos se penalizan cuando toman las decisiones equivocadas y se los recompensa cuando toman las correctas.
- Los algoritmos de refuerzo que incorporan el aprendizaje profundo han vencido a campeones humanos
  - Juegos complejos como el Go o el ajedrez (AlphaGo, AlphaGo Zero, AlphaZero)
  - Videojuegos (videojuegos de Atari o Starcraft: AlphaStar).

1. Introducción
2. Deep Learning
    1. Definición
    2. Historia
    3. Arquitecturas DNN
    4. Tipos de aprendizaje
3. Entrenamiento
    1. El problema
    2. Soluciones
4. Convolutional Neural Networks
    1. Convolución
    2. Arquitectura
    3. Aplicaciones
5. Deep Autoencoders
    1. Definición
    2. Procesamiento
    3. Tipos

3.1 El problema

- A pesar de los beneficios de las DNN, es muy difícil entrenar arquitecturas profundas y hasta hace muy poco no se había conseguido entrenarlas satisfactoriamente
- El método clásico de descenso de gradiente para arquitecturas supervisadas no funciona por tres motivos:
  - Disponibilidad de datos
    - Datos etiquetados son difíciles de obtener
    - Dada el alto poder de expresividad, entrenar con datos insuficientes produce overfitting
  - Óptimos locales
    - El aprendizaje supervisado implica resolver un problema de optimización altamente no convexo (minimizar el error de entrenamiento en función del peso de la red)
    - En una red profunda, este problema se complica al existir muchos más óptimos locales, y el entrenamiento con descenso de gradiente (o el de gradiente conjugado) ya no funcionan bien.

3.1 El problema

- Difusión de gradientes
  - Cuando se utiliza backpropagation para calcular las derivadas, los gradientes que se propagan hacia atrás (desde la capa de salida a las capas anteriores de la red) disminuyen rápidamente en magnitud a medida que aumenta la profundidad de la red.
  - Como resultado, la derivada del error con respecto a los pesos (sensibilidad) en las capas anteriores es muy pequeña, los pesos de las capas anteriores cambian lentamente, y las capas anteriores no aprenden mucho. Este problema a menudo se llama la "*difusión de gradientes*".
  - Además, si las últimas capas tienen suficiente número de neuronas serán capaces de generalizar sin ayuda de las anteriores capas, lo que equivale a entrenar un MLP menos profundo pero cuyos datos de entrada están corruptos (salida de las primeras capas sin entrenar).

3.1 El problema

- Hinton propuso una forma de atacar el problema (2006):

  1. Cada capa de la red es entrenada utilizando un algoritmo de pre-entrenamiento no supervisado comenzando con la capa de entrada a la red usando *Restricted Boltzmann Machines*
     - Se pueden guiar sus parámetros hacia mejores regiones dentro del hiperespacio de parámetros

  2. Después de pre-entrenar todas las capas, la red puede ser ajustada utilizando un entrenamiento supervisado:

     descenso estocástico de gradiente (SGD) + backpropagación

- Hay otras formas de minimizar o solucionar este problema

Uso de una mejor Loss function

- La *Loss Function (Función de coste)* es la magnitud que se minimizará durante el entrenamiento. Representa cuan buena es una red neuronal para un problema dado.
- Hay varios tipos de Loss Function, no solo el error cuadrático, por lo que ya no se puede hablar propiamente de "*error*", sino de "*coste*"
- La opción más habitual es reemplazar el error cuadrático por la función cross-entropy (*cross-entropy cost function*)

$$C = -\frac{1}{n}\sum_{i=1}^{n} d\ ln(y)$$

3.2 Soluciones

Uso de Optimizers

- Determinan cómo se actualizarán los pesos de la red en función de la *Loss Function*.
- Como MSE es el promedio sobre la suma de errores, cuya derivada es la suma de las derivadas, se debe calcular la dirección de Descenso de Gradiente para cada ejemplo del conjunto de datos en cada iteracion.
  - Muy ineficiente porque cada iteración del algoritmo solo mejora el error en un pequeño paso
- Alternativas
  - *Mini-batch Gradient Descent*: La regla para actualizar los pesos es igual, pero se aproxima la derivada de un pequeño mini lote del conjunto de datos y usaremos esa derivada para actualizar los pesos.
    - No se garantiza que este método dé los pasos en la dirección óptima.

3.2 Soluciones

- *SGD (Stochastic Gradient Descent)*: Versión extrema del *Mini-batch Gradient Descent* con tamaño de batch=1
  - Aproximación estocástica del descenso de gradiente.
  - Indica la dirección en la que la función tiene el ratio de aumento más pronunciada aunque no indica hasta donde se debe avanzar en esa dirección.
  - Se mejora aún más con el uso del Momento
- Algoritmos de optimización adaptativos
  - *RMSProp* (*Root Mean Square Propagation*): Usa el signo del gradiente para ver la dirección y lo combina con adaptar el tamaño del paso individualmente para cada peso (ratio de aprendizaje adaptativo) ya que los gradientes pueden variar mucho en magnitud y dirección.
  - *Adam*: Modificación del RMSProp, adaptando el ratio de aprendizaje en función de cómo estén distribuidos los parámetros. Si los parámetros están muy dispersos (sparse) el ratio de aprendizaje aumentará.

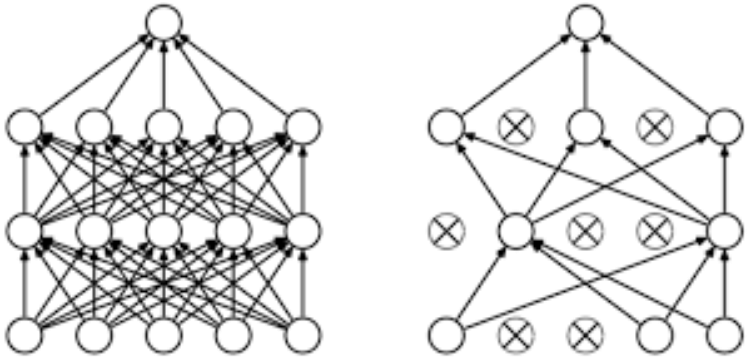
3.2 Soluciones

Uso de Métodos de Regularización

- Evitan el *overfitting* y mejoran la *capacidad de generalización*

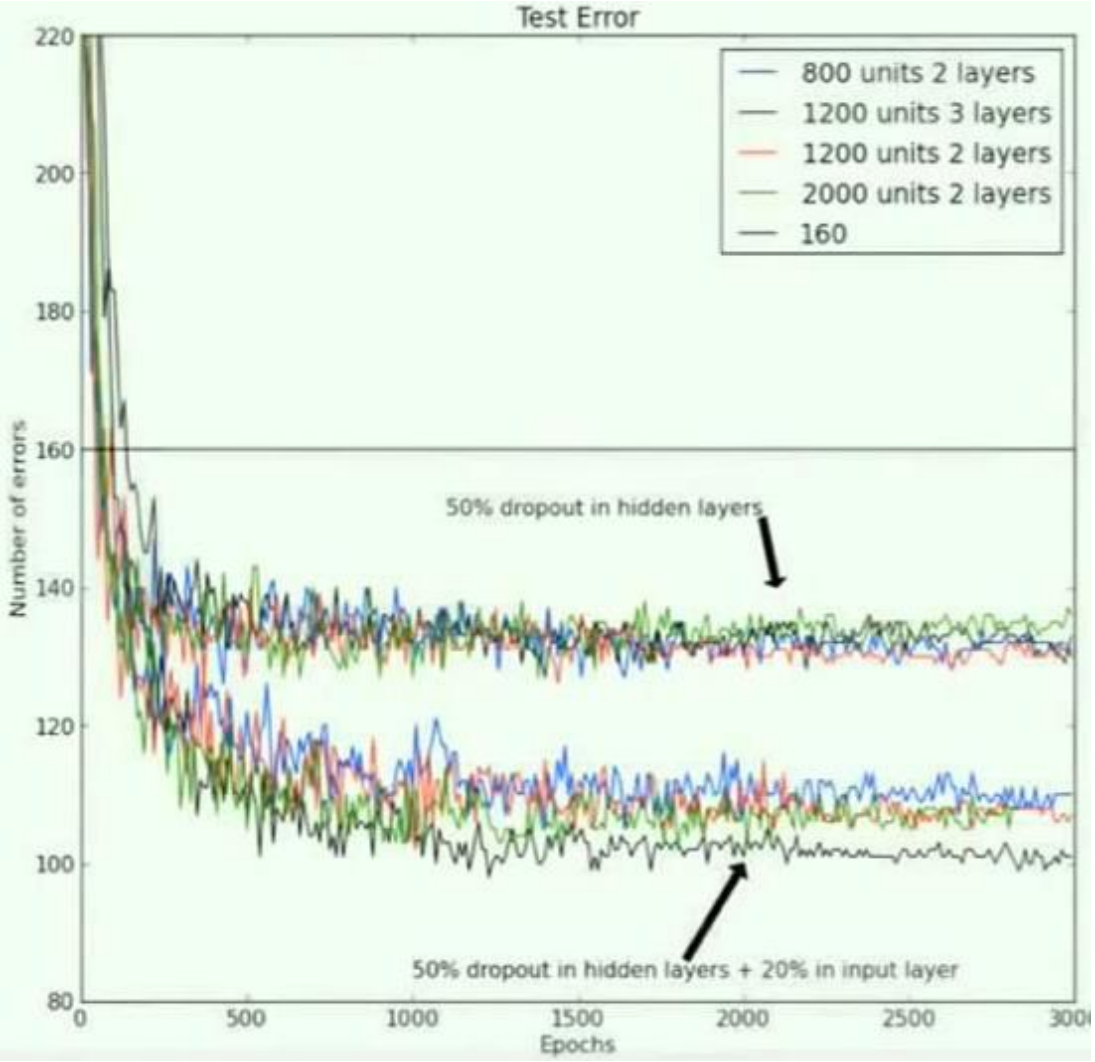
Si tu DNN no está sobre-entrenada, entonces debes de usar una más grande

- **Dropout**
  - Técnica que desconecta al azar algunas neuronas de la capa completamente conectada durante el entrenamiento
    - Cada vez que presentamos un patrón desconectamos al azar cada neurona oculta con una probabilidad de, digamos, 50%
    - Estamos entrenando al azar con 2^h arquitecturas que comparten los pesos entre ellas
  - Obliga a las capas totalmente conectadas a aprender el mismo concepto de diferentes maneras



(a) Standard Neural Net (b) After applying dropout.

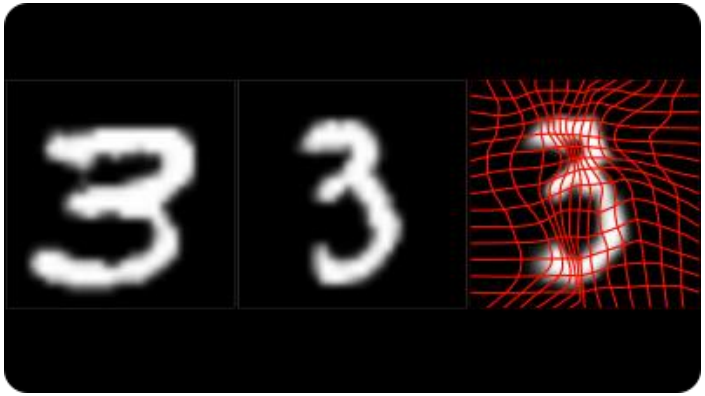
3.2 Soluciones

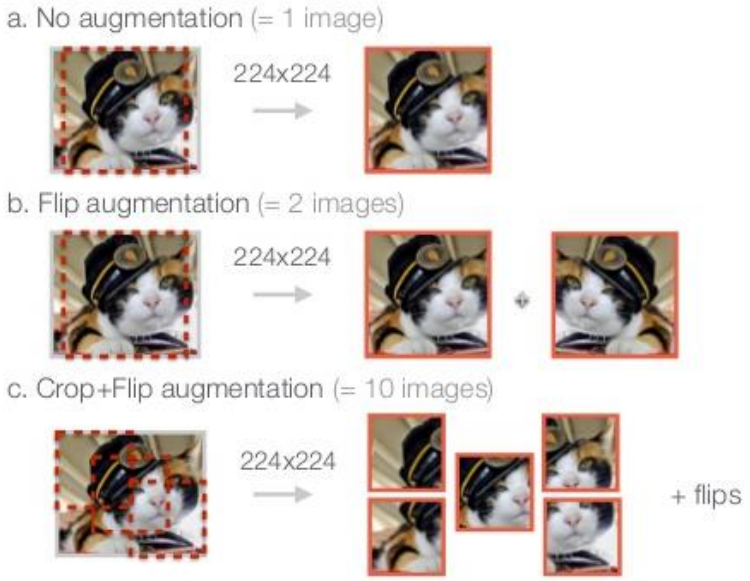


También se puede aplicar el dropout a la capa de entrada pero con menor probabilidad

3.2 Soluciones

- Magnificación de datos
  - Crear sintéticamente nuevos datos de entrenamiento aplicando algunas transformaciones en los datos de entrada.
    - Desplazamiento de imágenes
    - Valores aleatorios de RGB.
  - En ImageNet 2012, *Alex Krizhevesky* (Alexnet) usó un aumento de datos de un factor de 2048
    - El conjunto de datos utilizado para entrenar su modelo era 2048 veces mayor que al inicio





3.2 Soluciones

3.2 Soluciones

- Regularización L2 (o weight decay)
  - La forma más común de regularización. En la *Loss Function* agrega un término a cada peso que penaliza el valor al cuadrado de todos los pesos/parámetros que se optimizan.
    - Los pesos de muy alto valor son penalizados fuertemente.
    - El modelo tiende a preferir que todas las entradas a una capa se usen un poco en lugar de que algunas entradas se usen mucho.
    - El modelo se utilizará al máximo y habrá menos pesos no utilizados.
  - Además de la regularización L2, hay una regularización L1 que generalmente funciona mejor.

1. Introducción
2. Deep Learning
    1. Definición
    2. Historia
    3. Arquitecturas DNN
    4. Tipos de aprendizaje
3. Entrenamiento
    1. El problema
    2. Soluciones
4. Convolutional Neural Networks
    1. Convolución
    2. CNN
    3. Arquitectura
5. Deep Autoencoders
    1. Definición
    2. Procesamiento
    3. Tipos

4.1 Convolución

Operador de Convolución:

Operador matemático que transforma dos funciones f y g de una sola variable en una nueva función $f * g$ que representa la magnitud en la que se superponen f y una versión trasladada e invertida de g.

$$f(t) * g(t) = \{f * g\}(t) = h(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau$$

- Se puede asimilar al concepto de "*media móvil ponderada*"
- Si las funciones son discretas (por ejemplo, píxeles de una imagen) la convolución se calcula como

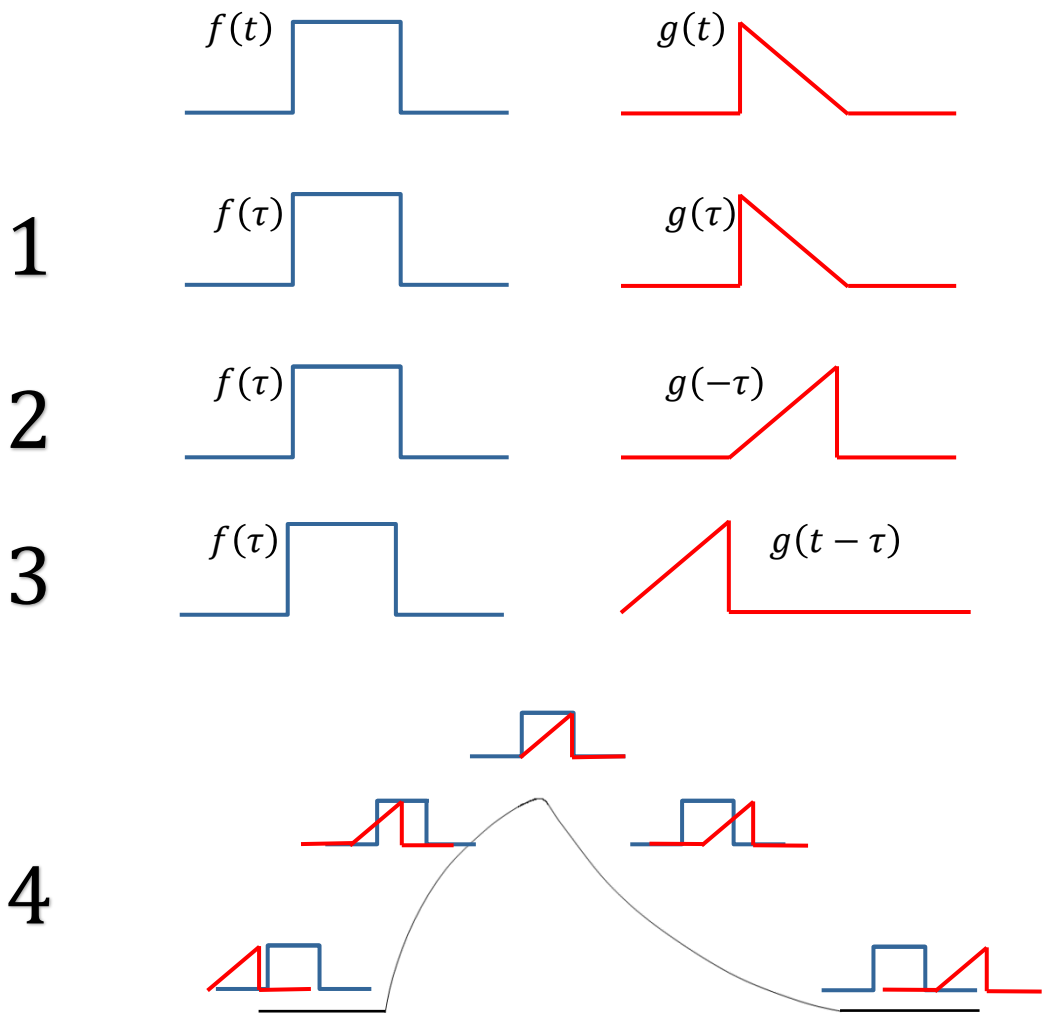
$$f(t) * g(t) = h(t) = \sum_{\tau} f(\tau)g(t-\tau)$$

4.1 Convolución

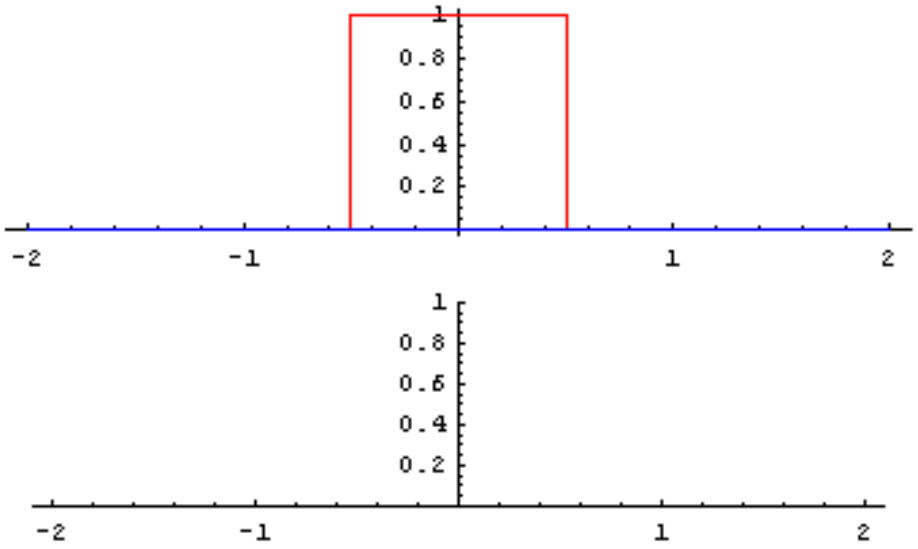
- Cálculo de la convolución de dos funciones *f* y *g*
  1. Se expresan *f* y *g* en términos de una variable ficticia τ
  2. Se refleja una de las funciones: $g(\tau) \rightarrow g(-\tau)$
  3. Se cambia la variable $\tau \rightarrow t-\tau$ lo que desplaza el origen de *g* sobre el eje τ hacia el punto t.
  4. Se varía t entre $(-\infty, +\infty)$ lo que desliza *g* sobre el eje τ. Siempre que las dos funciones se intersequen, se encuentra la integral de su producto.

- Es decir, se calcula el promedio ponderado desplazado de la función $f(\tau)$, donde la función peso es $g(-\tau)$.

4.1 Convolución



4.1 Convolución



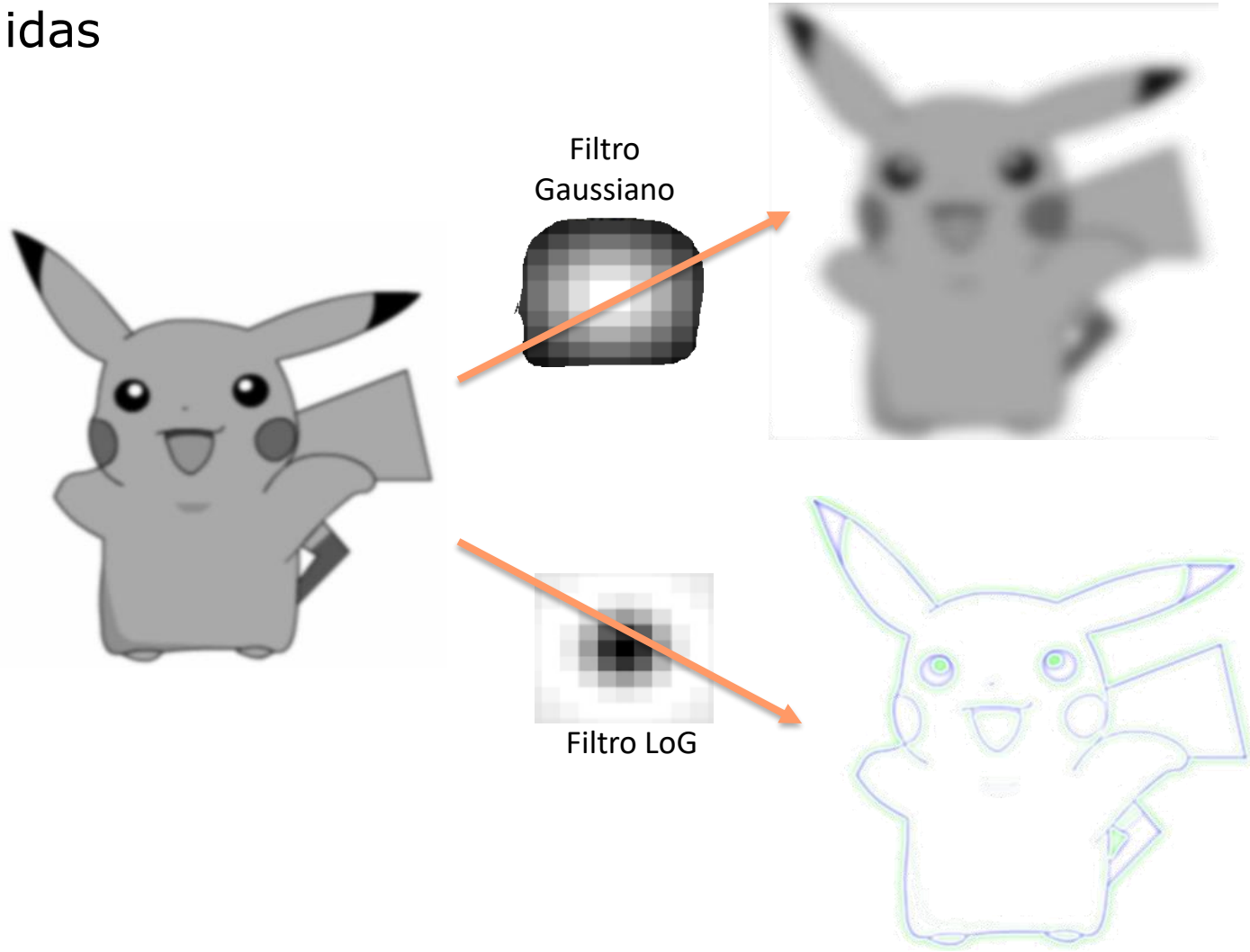
4.1 Convolución

Aplicaciones:

- Normalmente la convolución se aplica a la descripción matemática de señales, por ejemplo en
  - Acústica para describir el eco
    - Convolución del sonido original con una función que representa los objetos que lo reflejan
  - Óptica para describir manchas o sombras:
    - Una fotografía desenfocada es la convolución de la imagen correcta con el círculo borroso formado por el diafragma del objetivo
    - La sombra de un objeto es la convolución de la forma de la fuente de luz y del objeto cuya sombra se está proyectando

4.1 Convolución

- Diferentes filtros/kernels/máscaras producen diferentes salidas



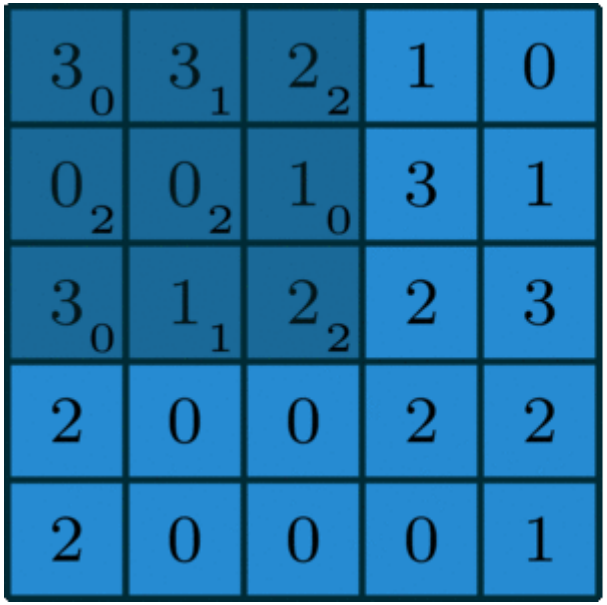
4.1 Convolución

Por qué usar la convolución

- Problemas con codificadores/clasificadores de imágenes
  - Cuanto mayor es la entrada del codificador, más pesos hay que entrenar y más lento (y peor) es el entrenamiento.
    - Para imágenes de 1.000×1.000 pixeles y 1.000 neuronas en la capa oculta, se necesita entrenar **1.000 millones** de pesos
    - Si entrenamos con fragmentos más pequeños (8×8 píxeles y 100 neuronas en la capa oculta) solo son **6.400** pesos.
  - Si usamos imágenes enteras, una imagen con un rostro centrado y otra con el mismo rostro desplazado hacia un lado, son completamente diferentes para el codificador.
    - Sin embargo… un círculo sigue siendo un círculo aunque lo desplacemos.

- Especialmente para resolver el segundo punto se puede utilizar la convolución

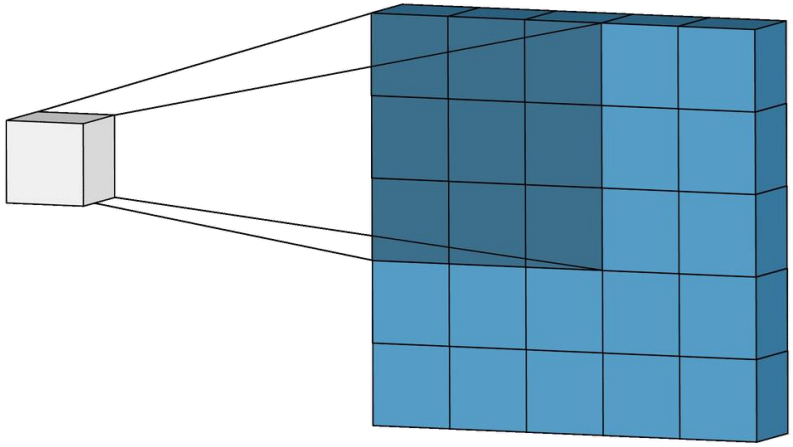
4.1 Convolución



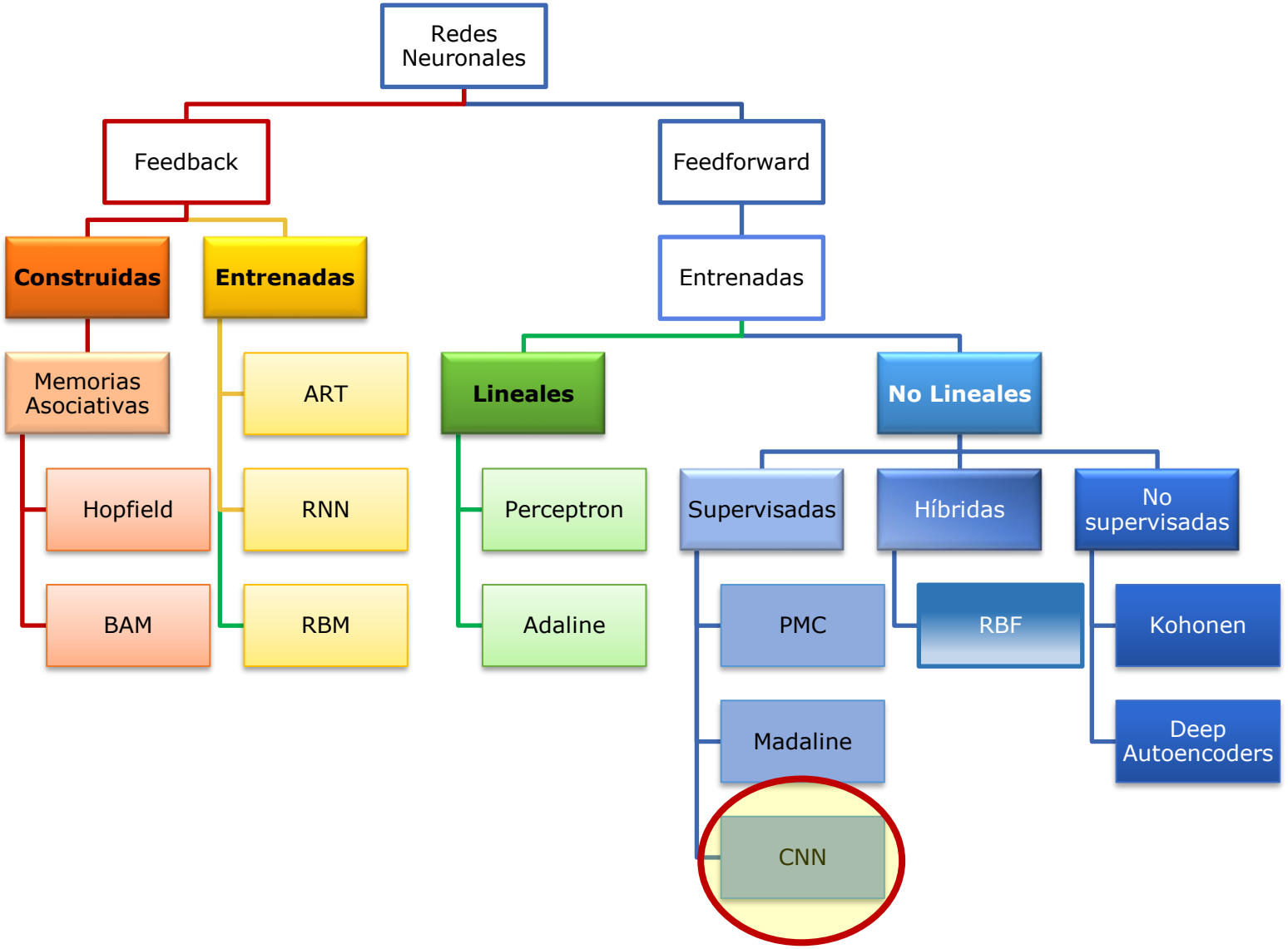
- 5×5 = 25 entradas
- 3×3 = 9 salidas
  - Si fuera una capa fully-connected estándar, tendría una matriz de pesos de 25×9 = 225 valores y cada característica de salida será la suma ponderada de cada característica de entrada.
  - La convolución permite hacer esta transformación con solo 9 parámetros

4.1 Convolución

- La convolución 2D es una operación simple:
  - Se define una *neurona*/*filtro*/*kernel* que es simplemente una pequeña matriz de pesos.
  - El *kernel* realiza una multiplicación de cada peso con la parte de la imagen de entrada sobre la que se encuentra y suma los resultados en una única celda de salida.
  - El *kernel* "*se desliza*" sobre los datos de entrada 2D repitiendo este proceso para cada ubicación, convirtiendo una matriz de características 2D en otra matriz de características 2D.



4.2 CNN


Redes Neuronales
Feedback
Feedforward
Construidas
Entrenadas
Entrenadas
Memorias Asociativas
ART
Lineales
No Lineales
Hopfield
RNN
Perceptron
Supervisadas
Híbridas
No supervisadas
BAM
RBM
Adaline
PMC
RBF
Kohonen
Madaline
Deep Autoencoders
CNN

Redes Neuronales Convolucionales

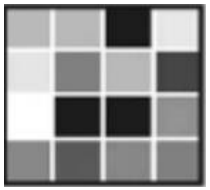
Redes que pueden aprender relaciones entrada-salida basadas en operaciones de convolución, donde la entrada es una imagen

- Características de las CNN
  - Basada en cómo se estructuran realmente las neuronas en nuestro sistema visual.
    - Los patrones de conectividad entre neuronas asemejan la organización del córtex visual de los animales
    - Las neuronas corticales responden a estímulos en un área del campo visual (*campo receptivo*) que se solapa parcialmente con el de otras neuronas cubriendo así todo el campo visual
  - Reemplazan la multiplicación de matrices del MLP ($\vec{X}$ x $\overrightarrow{W_i}$) por la convolución ($\vec{X} * \overrightarrow{W_i}$)
  - Cada conjunto de convoluciones extrae jerárquica e incrementalmente alguna *característica* de la imagen que se procesa

4.2 CNN

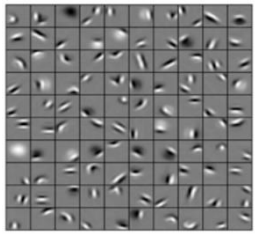
- La operación de convolución recibe como *entrada* una imagen y aplica sobre ella n *neuronas*. Cada una devuelve un *mapa de características* de la imagen original
  - Un mapa de características es una matriz donde cada celda es una combinación lineal de los pixeles de entrada con los valores del kernel (pesos).
  - Las características de salida corresponden a las entidades de entrada ubicadas en la capa de entrada aproximadamente en la misma posición que la celda de salida.
  - Estas características son invariantes respecto a la posición dentro de la imagen original y corresponden a cada posible ubicación del filtro en la imagen original.
  - El mismo filtro (neurona) sirve para extraer la misma característica en cualquier parte de la entrada ya que se replica en todo el campo visual.

4.2 CNN



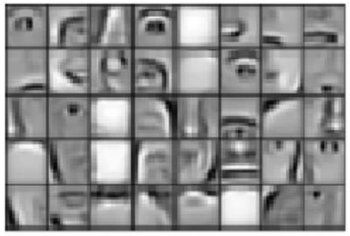
Capa 1: La red identifica píxeles con distinto nivel de gris/color

Características de bajo nivel



Capa 2: La red aprende a identificar bordes y formas simples

Características de nivel medio



Capa 3: La red aprende a identificar formas más complejas y objetos

Características de alto nivel

Capa 4: La red aprende qué formas y objetos se pueden usar para definir una cara humana

Clasificador entrenable

Capa 5: La red clasifica la salida

Las sucesivas capas operan no sobre la imagen original, sino sobre los mapas de características producidas por la capa anterior

4.2 CNN

- Aplicaciones de las CNN

  - 1D CNN: texto, señales secuenciales
    - Clasificación de texto
    - Reconocimiento de género musical
    - Modelos acústicos para reconocimiento de hablas
    - Predicción de series temporales (poco)

  - 2D CNN: imágenes, señales en frecuencias (habla y audio)
    - Detección de objetos, localización, reconocimiento

  - 3D CNN: video, imágenes volumétricas/tomográficas
    - Reconocimiento/comprensión de video
    - Análisis de imágenes biomédicas
    - Análisis de imágenes hiperespectrales

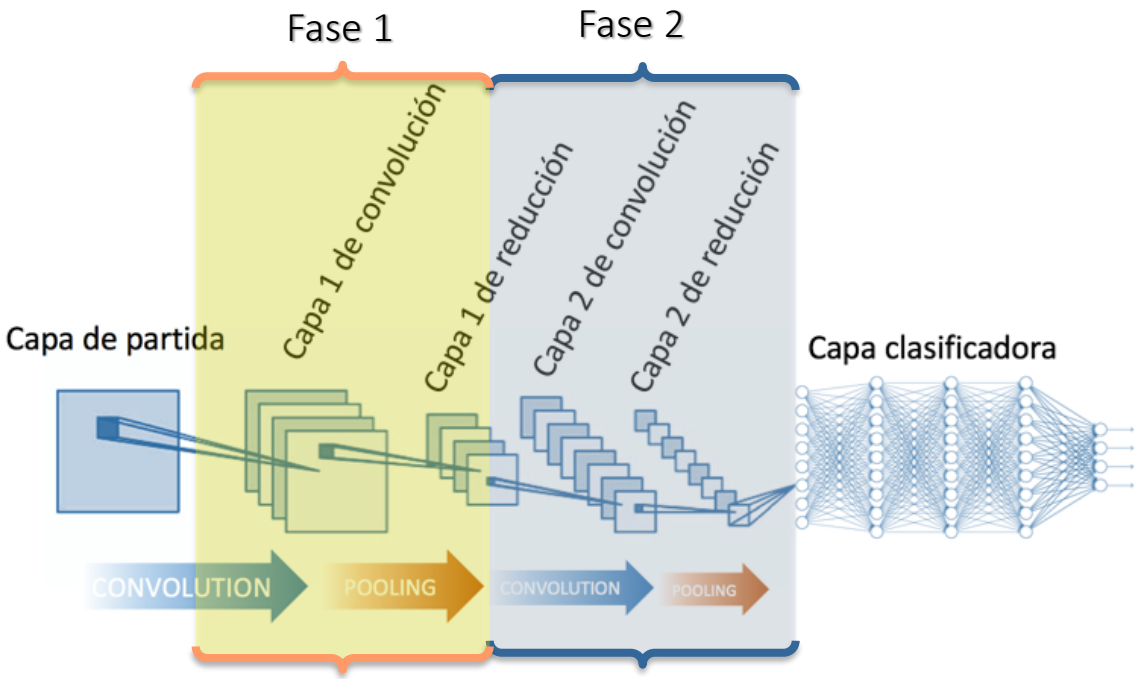
4.3 Arquitectura

- Arquitectura de las CNN
  - *Feedforward*: Variación (compleja) del MLP que requiere un pre-procesamiento mínimo
  - *Locally connected*: Existen capas, pero se restringen las conexiones entre capas (no es fully connected)
    - Cada neurona oculta se conecta a una región pequeña y contigua de pixeles de la entrada
    - Cada neurona oculta se conecta a un subconjunto de unidades de la capa anterior lo que reduce drásticamente el número de pesos
    - Dos neuronas consecutivas de una capa intermedia se especializan en regiones solapadas de la capa anterior.
  - *Capas*: Input, Output y múltiples Hidden layers del tipo:
    - Capa *convolucional*
    - Capa de *pooling*
    - Capa *clasificadora (fully connected)*

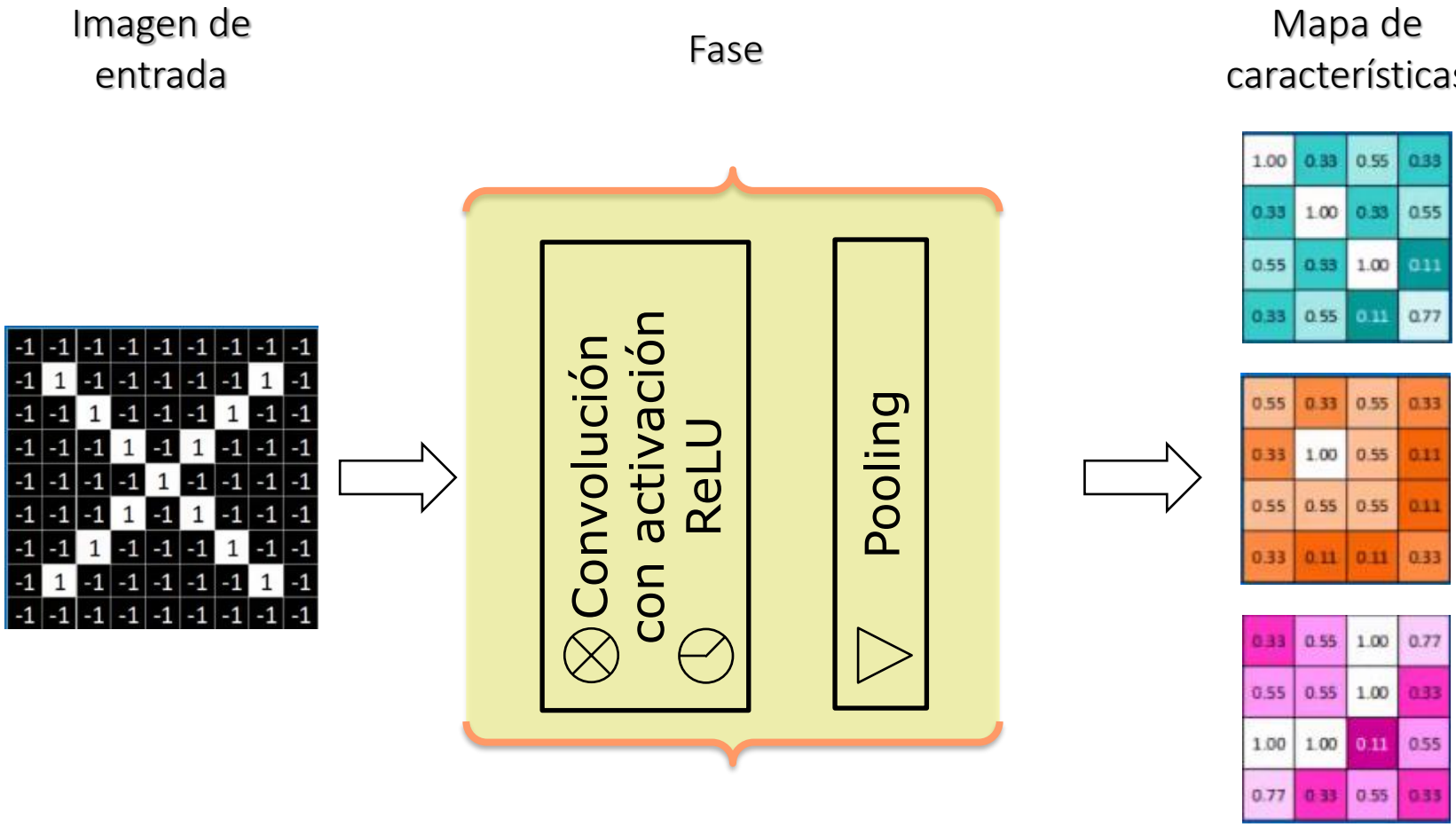
4.3 Arquitectura

- Una o más de fases de extracción de características acabadas en un clasificador
- Cada fase genera un conjunto de *feature maps* o *mapas de características*

Fase = capa de convolución + pooling



4.3 Arquitectura



4.3 Arquitectura

Capa de convolución

- *Capa compuesta por neuronas convolucionales que detecta patrones aplicando varios filtros de 1 o más dimensiones a los datos que le llegan*
  - Cada filtro es una neurona
  - Las neuronas realizan la convolución de una imagen que recibe como *input* con un *filtro o kernel* devolviendo como output un *mapa de características* de la imagen original
  - Los pesos que se han de ajustar durante el entrenamiento son los parámetros del filtro
  - Un fan-in pequeño (cada neurona acepta solo un pequeño numero de señales de entrada) ayuda al método del descenso estocástico de gradiente (SGD) a actuar a través de varias capas sin que su acción se desvanezca (se reduce la difusión de gradientes)

4.3 Arquitectura

- La salida de cada CN es una matriz que se obtiene por combinación lineal de las salidas de las neuronas de la capa anterior operadas con filtro de esa conexión y llevada a una función de activación no lineal *f*

$$\vec{Y_j} = f\left(\sum_i \overrightarrow{K_{ij}} * \vec{Y_i}\right)$$

  - Donde
    - $\vec{Y_j}$ matriz de salida de la neurona j
    - $\vec{Y_i}$ salidas de las neuronas de la capa anterior
    - $\overrightarrow{K_{ij}}$ filtro (kernel) de la conexión
    - f función de activación ReLU

4.3 Arquitectura

- El filtro se va desplazando sobre toda la matriz de datos de entrada. En cada desplazamiento:
  1. Se computa la convolución de los datos de entrada con el filtro, generando un valor escalar de salida del filtro
  2. Se calcula la función de activación no lineal para ese valor
  3. Se desplaza el filtro un número de posiciones (stride) solapando parcialmente la región sobre la que operó en la convolución anterior
  4. Se vuelve a 1

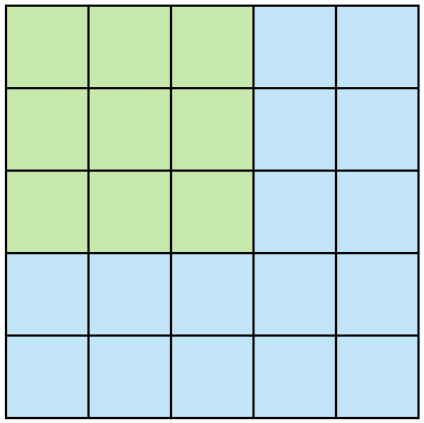
- El tamaño de la matriz de salida es

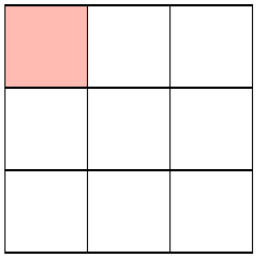
$$(N \times N) * (F \times F) = (N - F + 1) \times (N - F + 1)$$

  - *N* lado matriz de entrada
  - *F* lado del filtro

4.3 Arquitectura

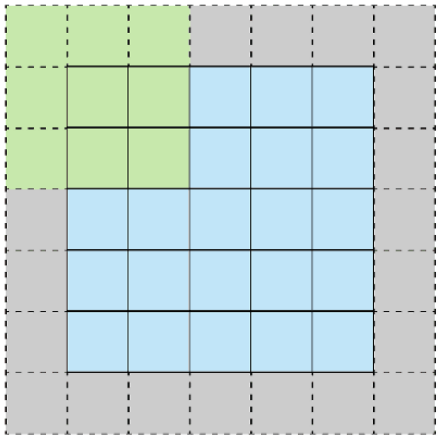
- Parámetros a determinar a la hora de definir los filtros:
  - Dimensiones de los filtros: se deciden al diseñar la capa
  - Stride: cuantas filas/ columnas avanzamos en cada operación de convolución. Se define para cada filtro de la capa
  - Padding: añadir ceros simétricamente a la matriz de entrada (en filos y/o columnas) para mantener las mismas dimensiones en la matriz de salida (zero padding).

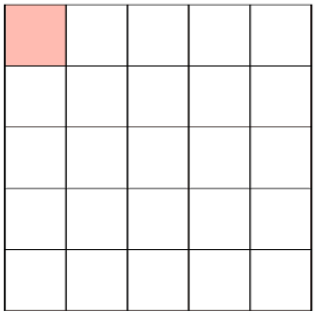




Stride 1

Feature Map





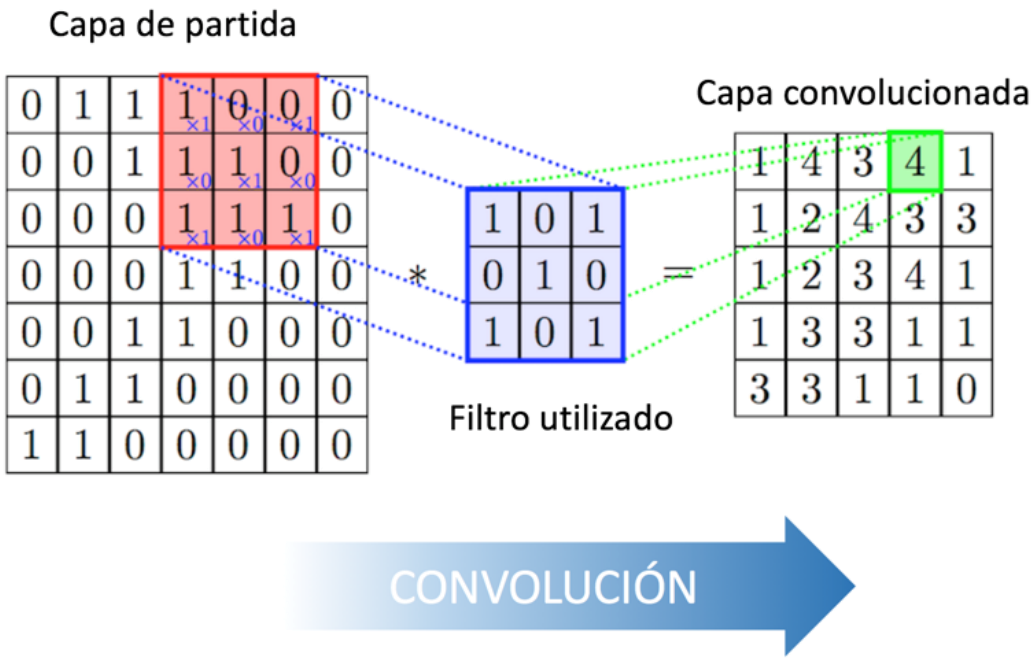
Stride 1 with Padding

Feature Map

4.3 Arquitectura

Ejemplo de convolución de un filtro de 3×3 sobre una matriz de entrada de 7x7.

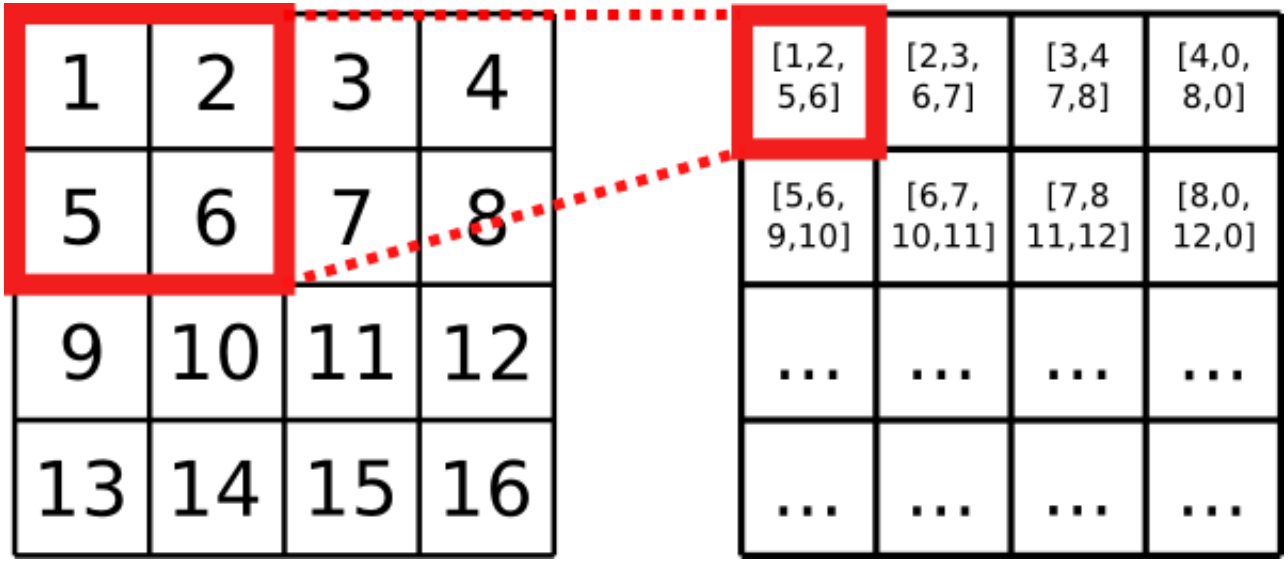
- Stride = 1
- Sin Zero padding



4.3 Arquitectura

Ejemplo de desplazamiento de un filtro de 2×2 sobre una matriz de entrada de 4x4.

- Stride = 1
- Con Zero padding

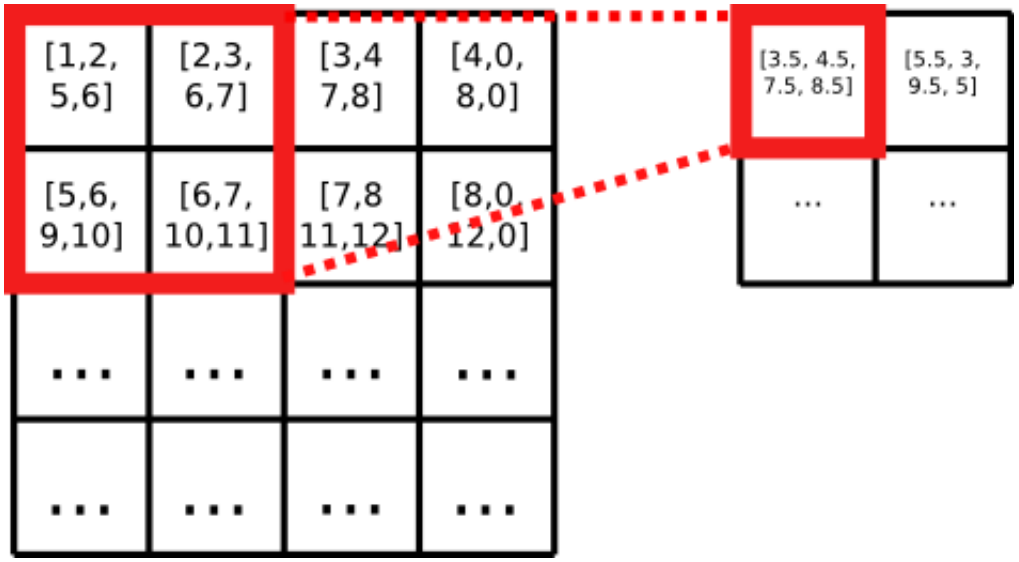


Capa de Pooling

- Capa que agrupa las características (listas de números) de varias coordenadas contiguas de la imagen con alguna función de agrupación
  - Disminuyen la resolución de la red poco a poco
  - Disminuyen el número de pesos a entrenar en la siguiente fase de convolución
  - Pueden ser
    - *Max pooling*: selecciona el máximo de una ventana de neuronas en uan capa y la lleva a la siguiente
    - *Mean pooling*: selecciona la media de una ventana de neuronas en una capa y la lleva a la siguiente
- Esta etapa reduce la resolución de la imagen y permite hacer una nueva fase de convolución y pooling hasta que nos quede una imagen de 1×1 pixeles con muchísima información

4.3 Arquitectura

Ejemplo de pooling con una ventana de 2x2 y cálculo del promedio



4.3 Arquitectura

Capa de clasificación

- Los datos llegan a la capa de clasificación reducidos a una serie de características únicas para cada imagen de entrada.
- Esta capa *clasifica* esas características asignando una probabilidad a cada etiqueta de las aprendidas durante el entrenamiento.
- Arquitectura *fully connected* compuesta por un MLP con
  - Capa(s) oculta(s) no lineal(es): función de activación Relu
  - Capa de salida (que también es la capa OUTPUT de la CNN)
- La salida de cada neurona se obtiene de igual manera que en el MLP

$$y_j = f\left(\sum_i w_{ij} \cdot y_i\right)$$

4.3 Arquitectura

Output layer

- *Última capa del MLP* y de la *red*, con tantas neuronas como clases tiene que clasificar la CNN

Tipo de salida	Función de activación	Loss funtion
Binaria	Sigmoidea	Binary cross-entropy
Discreta	SoftMax	Discrete cross-entropy
Continua	Lineal	MSE

- Modelo de regresión Softmax
  - Este modelo generaliza la regresión logística a problemas de clasificación donde la salida *y* puede tomar más de dos valores posibles.
  - Por ejemplo, problemas donde el objetivo es distinguir entre 10 clases diferentes.

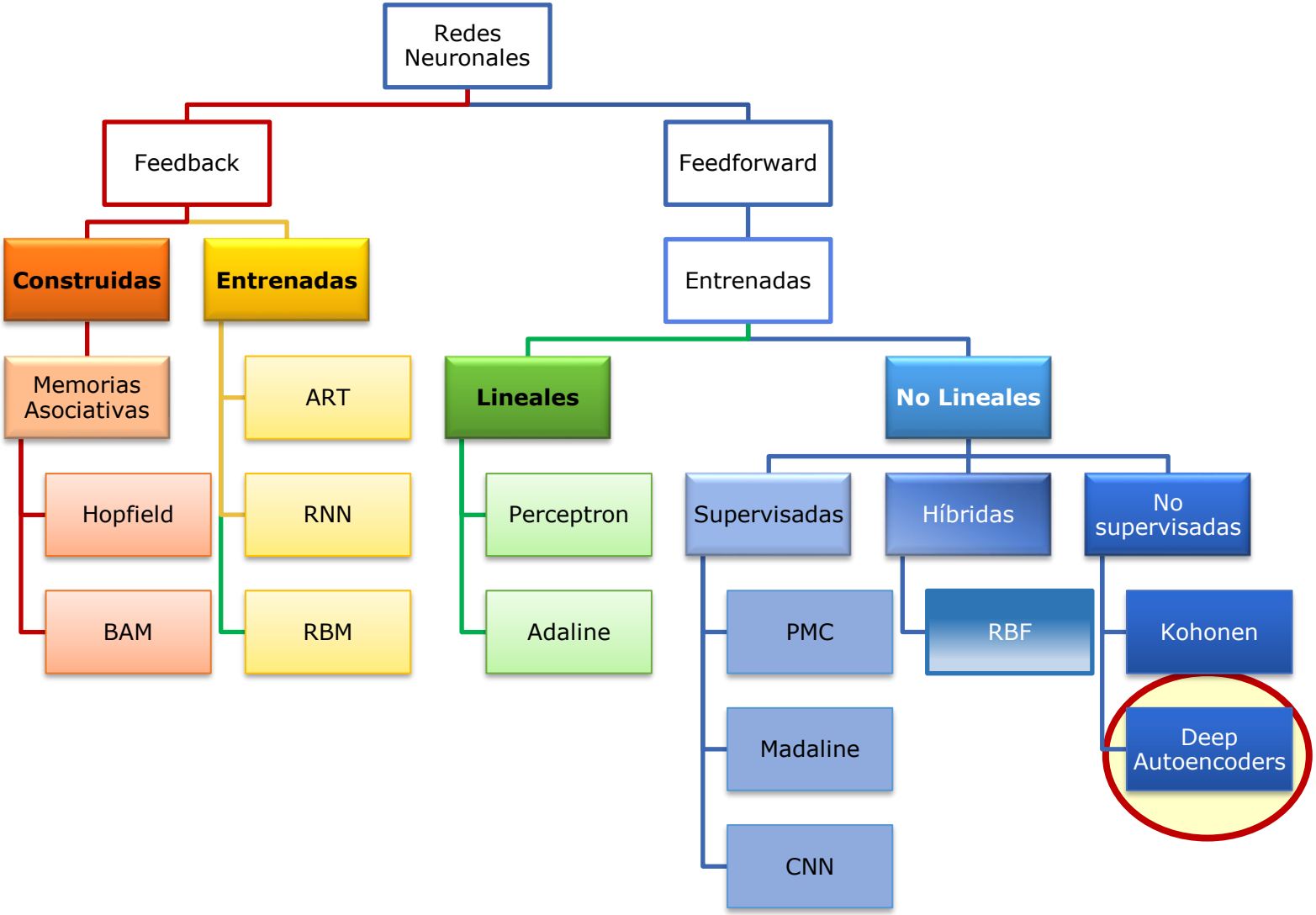
4.3 Arquitectura

Hiperparámetros de la CNN

- Convolución (para cada capa):
  - Número de características (neuronas/filtros)
  - Dimensión de cada filtro (pesos/tamaño del filtro)
  - Tamaño del stride
  - Padding
- Pooling
  - Tamaño de la ventana
  - Tipo de pooling (max/mean)
- Clasificador (fully connected)
  - Número de capas
  - Número de neuronas

1. Introducción
2. Deep Learning
   1. Definición
   2. Historia
   3. Arquitecturas DNN
3. Aprendizaje
   1. El problema del entrenamiento
   2. Tipos de aprendizaje
   3. Overfitting
4. Convolutional Neural Networks
   1. Convolución
   2. CNN
   3. Arquitectura
5. Deep Autoencoders
   1. Definición
   2. Procesamiento
   3. Tipos

5.1 Definición



5.1 Definición

- Modelo neuronal feedforward no supervisado de compresión de datos en el que las funciones de compresión y descompresión son llevadas a cabo por distintas capas que son:
  1. Específicas de los datos: solo pueden comprimir datos similares a los usado para entrenamiento. Diferente de otros algoritmos de compresión (como MPEG-2 Audio Layer III que contiene suposiciones sobre "sonido" en general, pero no sobre tipos específicos de sonidos).
  2. Con pérdida: las salidas descomprimidas se degradarán en comparación con las entradas originales (similar a la compresión MP3 o JPEG).
  3. Aprenden automáticamente a partir de ejemplos. El aprendizaje es auto-supervisado (una subclase de no supervisado), ya que la entrada y la salida es la misma (y por lo tanto no requiere pares entrada/salida)

5.1 Definición

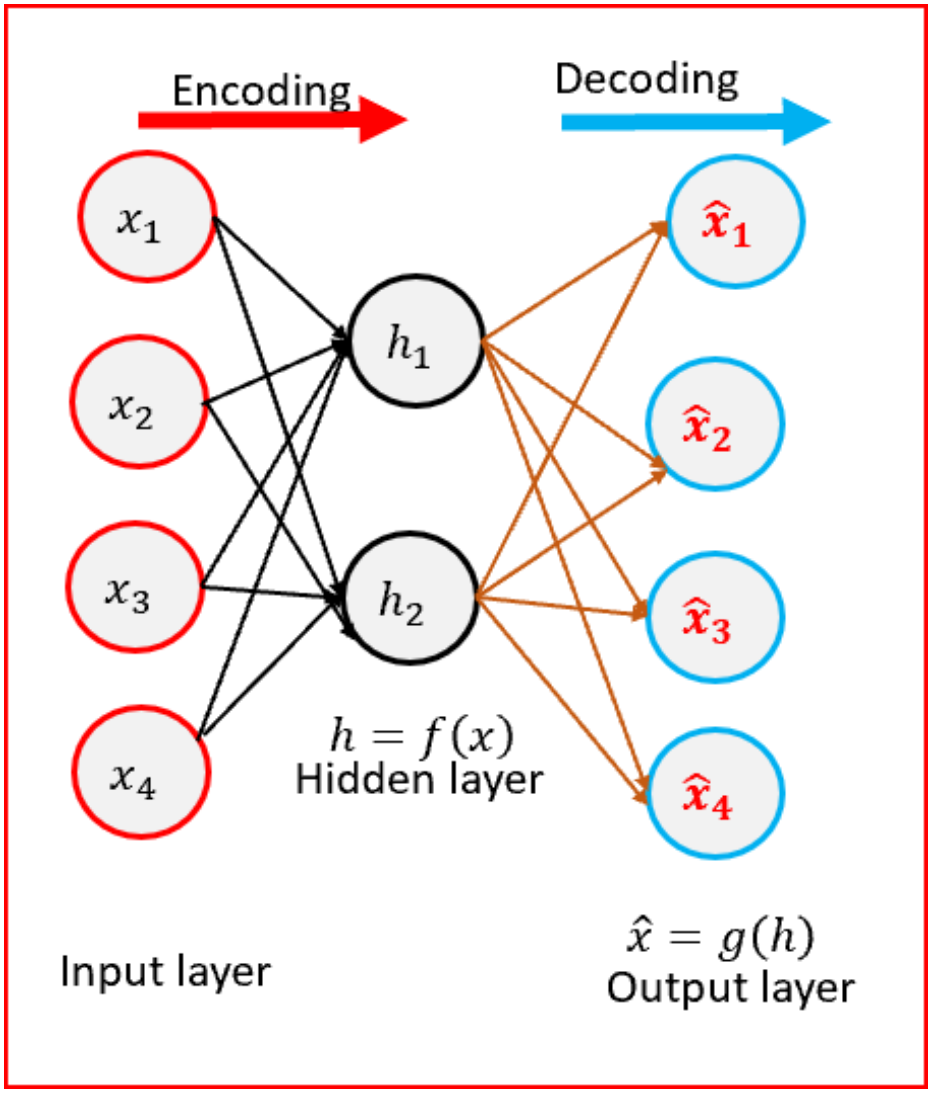
- El algoritmo del autoencoder necesita:
  - Una *función codificadora*: Codifica un input $\vec{x}$ mediante una función $h = f(\vec{x})$
  - Una *función decodificadora*: Decodificar los valores de $f(\vec{x})$ utilizando una función $\hat{x} = g(f(\vec{x}))$, creando así un output idéntico al input original.
  - Una *función de distancia (loss function)* que mide la cantidad de información perdida entre la representación comprimida y la representación descomprimida. El objetivo es minimizar el error de reconstrucción entre el input $\vec{x}$ y el output $\hat{x}$.

- ***f*** y ***g*** se implementan como capas de neuronas cuya función de activación es diferenciable respecto a la función de distancia de forma que los parámetros de codificación y decodificación (los pesos) se pueden optimizar para minimizar el *loss* utilizando *Stochastic Gradient Descent + backpropagation*.

5.1 Definición

- Arquitectura
  - Capa de entrada: No hace procesamiento
  - Capa oculta: Realiza la codificación
  - Capa de salida: Realiza la decodificación

- Las capas de codificación y decodificación pueden ser convolucionales, MLP o cualquier otra arquitectura diferenciable y no lineal.



5.2 Procesamiento

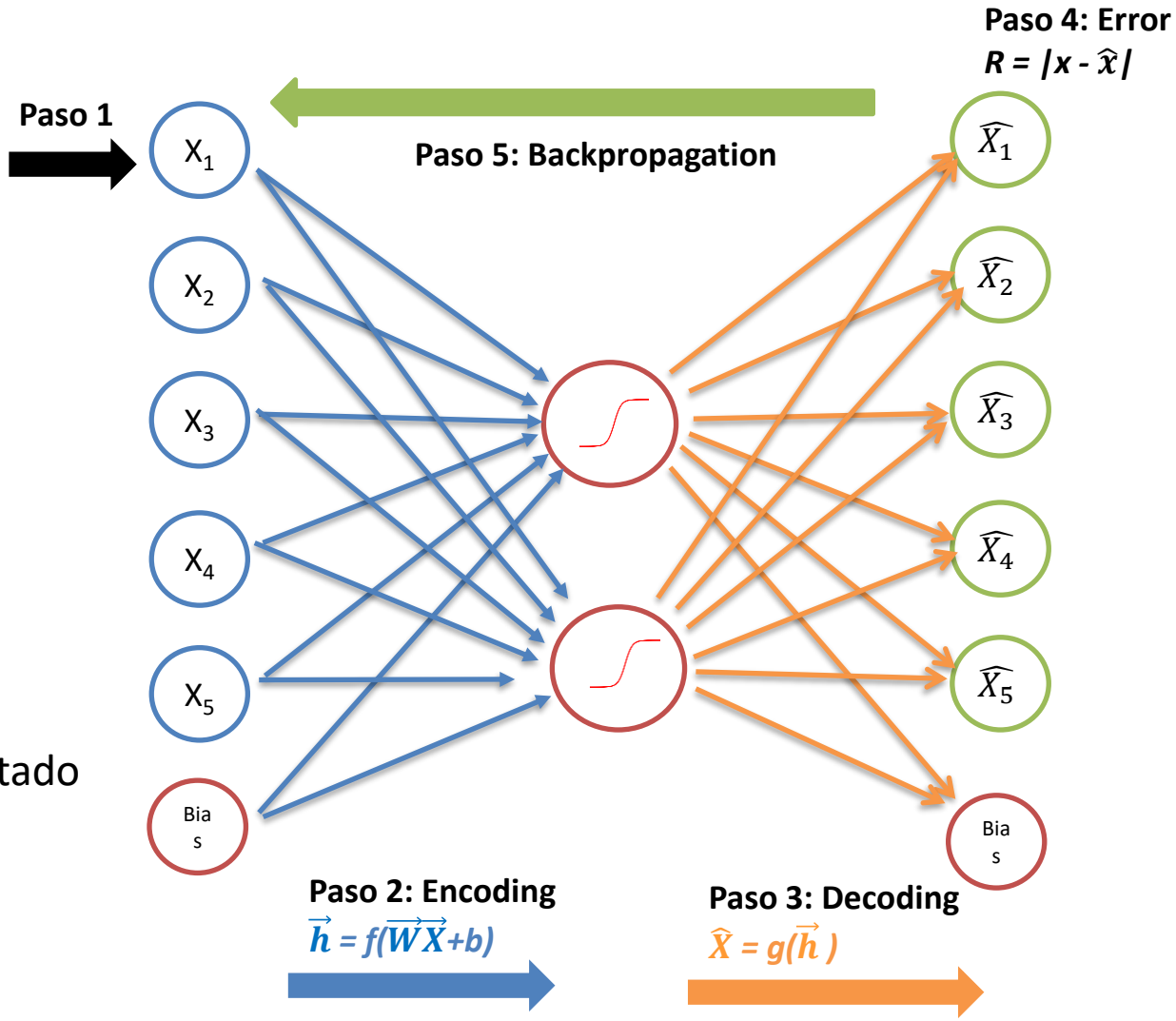
- Se procesa un input, del que se van a extraer características (features):
  - Se codifica el input (capa de entrada → capa oculta). Se obtiene una codificación de la entrada
  - Se decodifica el valor codficado (capa oculta → capa de salida). Se obtiene una reconstrucción de la entrada.

- Se calcula el valor de *loss*, comparando el input y el output (error de reconstrucción)

- Se minimiza el error de reconstrucción usando backpropagation. Los pesos se actualizan en función de cuán responsables han sido del error.

5.2 Procesamiento

	Canción 1	Canción 2	Canción 3	Canción 4	Canción 5
Cliente 1	0	0	1	1	0
Cliente 2	1	0	0	1	0
Cliente 3	1	0	1	0	1
Cliente 4	0	0	0	1	0



Base de datos de canciones y usuarios a los que les han gustado o no las canciones:

- **1** la canción ha gustado
- **0** la canción no ha gustado

5.2 Procesamiento

1. Se toma como input la primera fila del Dataset $\vec{X}$.

2. Se codifica en otro vector $\vec{h}$ con menor dimensionalidad que el del input (en este caso, 2). Como los valores del vector de entrada varían entre 0 y 1 se utilizará la función sigmoide para la codificación del vector.
   1. $\vec{X}$ es el vector de entrada
   2. $\vec{W}$ es el vector de pesos aplicados al input
   3. b es el término de sesgo, siendo éste el output de la red neuronal cuando tiene "zero input"

$$\vec{h} = sigmoide(\vec{W}\vec{X} + b)$$

3. Se decodifica $\vec{h}$ para recrear el input, con la misma dimensión que el vector de entrada.

$$\hat{X} = g(\vec{h})$$

5.2 Procesamiento

4. Se calcula el *error de reconstrucción* (R), diferencia entre el input ($\vec{X}$) y el output ($\hat{X}$)

$$R = \left|\vec{X} - \hat{X}\right|$$

$$R = \left|\vec{X} - g\left(f\left(\vec{X}\right)\right)\right|$$

5. Se propaga el error hacia atrás con *backpropagation*. Los pesos se actualizan con SGD (*Stochastic Gradient Descent*) en función de cuánto han sido responsables por el error.

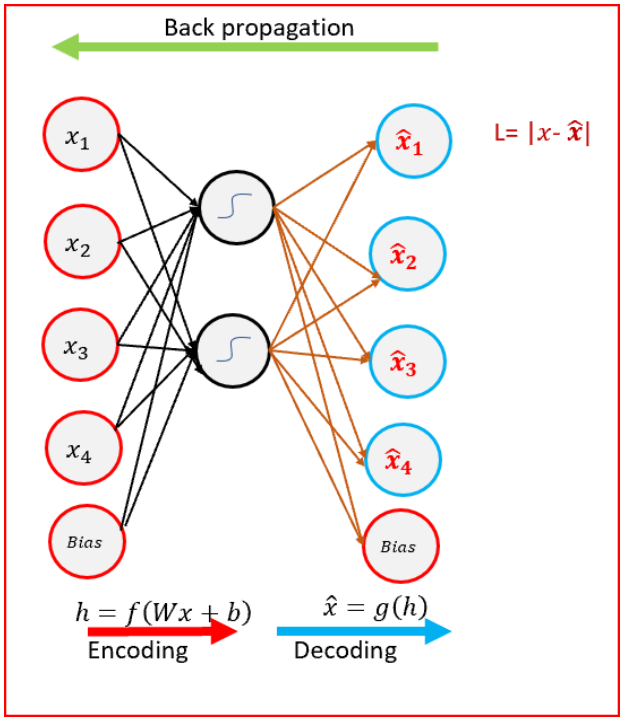
6. Se repiten todos los pasos del 1 al 5 por cada uno de los vectores de entrada disponibles hasta alanzar el mínimo error posible.

5.3 Tipos de Autoencoders

- **Undercomplete Autoencoders**

Los autoencoders tienen como objetivo identificar y capturar las características más importantes en los datos.

Este tipo de autoencoders tiene una capa oculta menor que la capa de entrada y así consiguen obtener mejor las características de los datos.



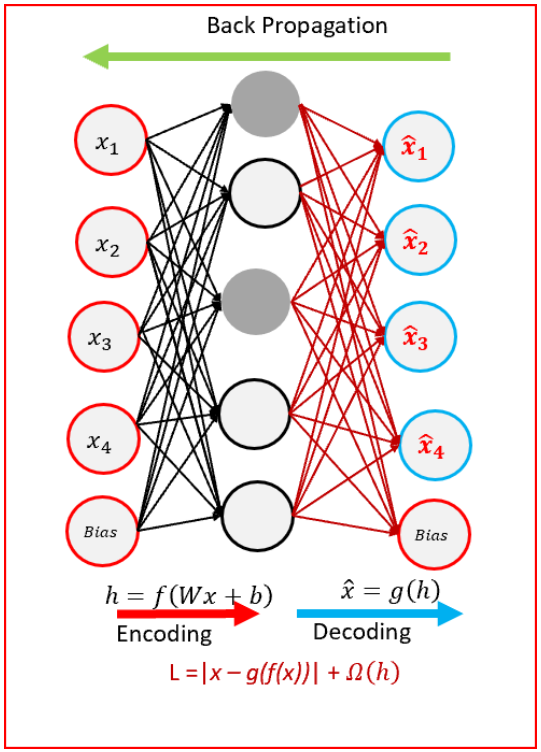
5.3 Tipos de Autoencoders

- **Sparse Autoencoders**

Estos autoencoders tienen una mayor capa oculta, que también consigue obtener las características más importantes, pero evita que la salida de la red copie la entrada recibida.

La decodificación toma únicamente los valores más altos de activación de las neuronas de la capa oculta y ponen a cero la salida de las demás neuronas de la capa oculta.

De esta forma se fuerza que no todas las neuronas sean activadas al mismo tiempo, aunque en cada nueva entrada se activarán neuronas distintas.

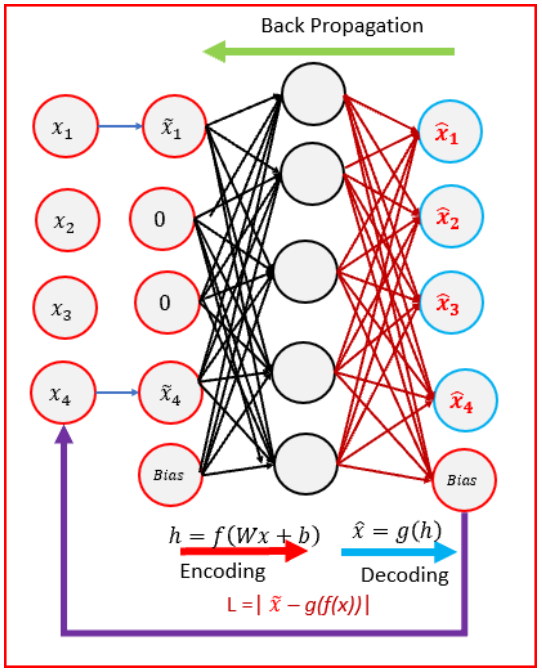


5.3 Tipos de Autoencoders

- **Denoising Autoencoders**

Estos autoencoders añaden ruido de forma intencionada a los datos antes de ser introducidos en la red. Así se fuerza a que la red aprenda realmente las características más importantes de los datos en lugar de copiarlos en su salida.

La red deberá eliminar este ruido y generar una salida que será comparada con el dato de entrada antes de haberle añadido el ruido, y así se minimizará la función de error.

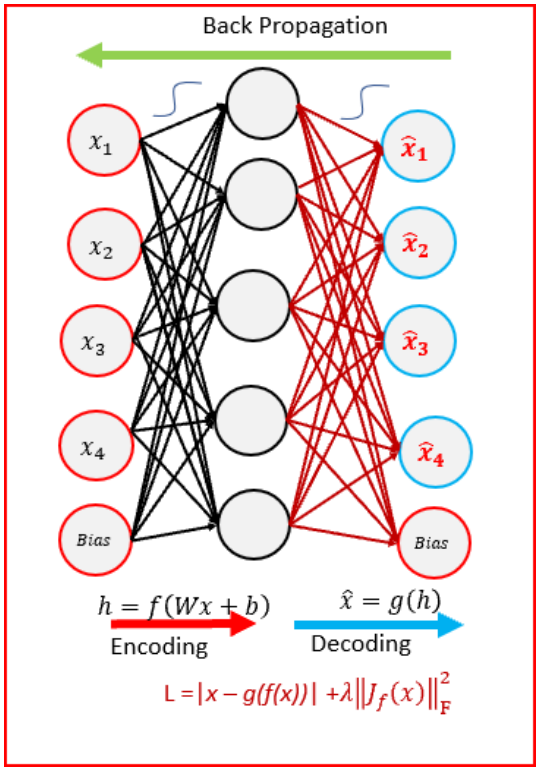


5.3 Tipos de Autoencoders

- **Contractive Autoencoders**

Tienen como objetivo obtener una representación robusta del aprendizaje para que sea menos sensible a las pequeñas variaciones en los datos.

Son una mejor alternativa que los Denoising autoencoders cuando se intenta extraer las características de los datos de una forma más fiable, concisa y útil.



5.3 Tipos de Autoencoders

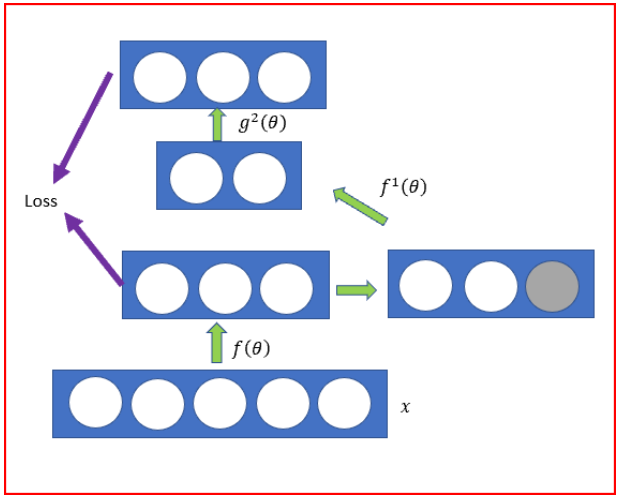
- **Stacked Denoising Autoencoders**

Consisten en una red neuronal multicapa compuesta por múltiples Autoencoders.

Se trata de modelos *"Deep"*.

Al añadir más de una única capa oculta se consigue poder trabajar con datos de mayor dimensionalidad, permitiendo reducirlos y así obtener mejor sus características.

Cada nueva capa oculta compactará más los datos que la capa anterior, hasta tener una versión "concentrada" de la entrada.



5.3 Tipos de Autoencoders

- **Deep Autoencoders**

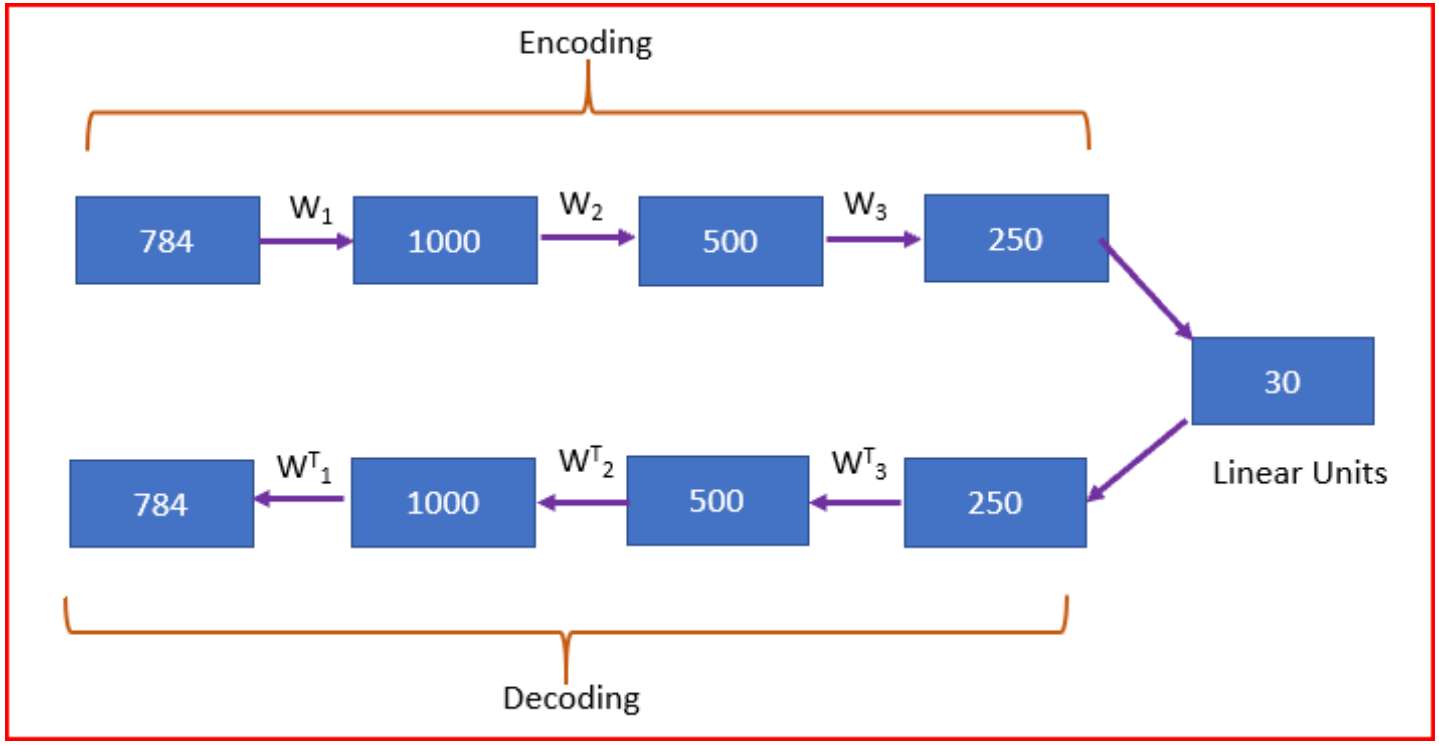
Consiste en dos redes neuronales simétricas y conectadas, donde una se encargará de la codificación y la otra de la decodificación.

La red encargada del *encoding*, a partir de la entrada, irá reduciendo la dimensionalidad de esta, manteniendo las características más importantes, hasta obtener la versión reducida y concentrada de esta entrada.

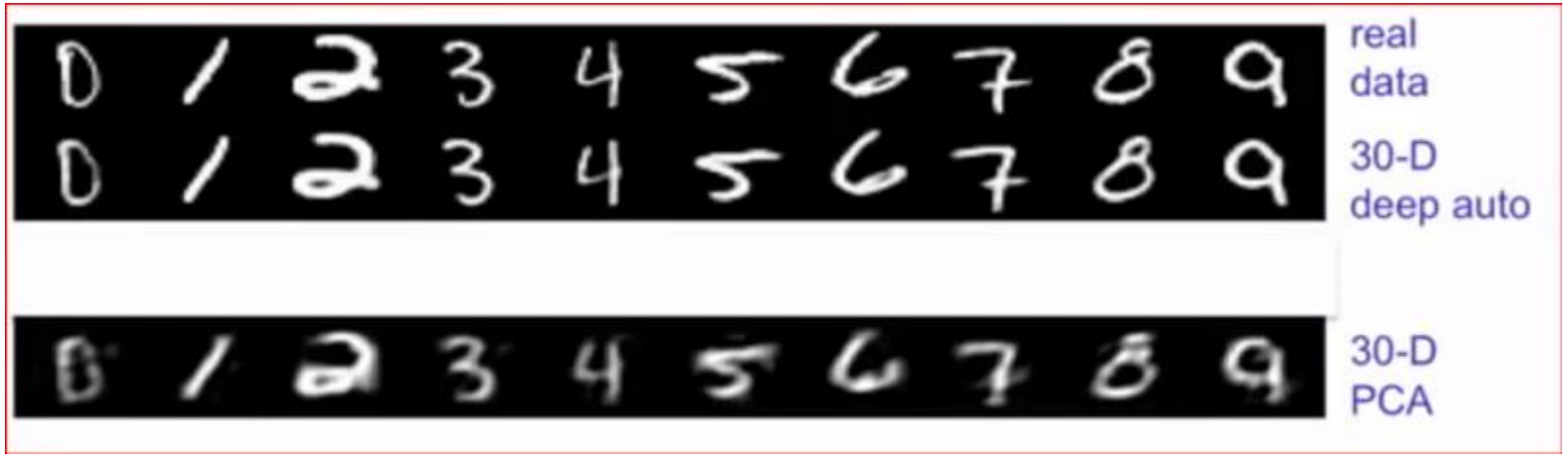
La red encargada del *decoding* recibe como entrada esta versión reducida y concentrada producida por la red *encoding*, y volverá a aumentar su dimensionalidad hasta el tamaño original a partir de únicamente sus características.

Por lo general, cuentan con 4 o 5 capas por cada red, tratándose de modelos profundos.

5.3 Tipos de Autoencoders



5.3 Tipos de Autoencoders



5.3 Tipos de Autoencoders

- Aplicaciones
  - Reducción de la dimensionalidad para datos no lineales. Codifica la entrada en la capa oculta a una dimensión más pequeña en comparación con la dimensión de entrada. La capa oculta se decodifica más tarde como salida con la misma dimensión que la entrada.
  - Motores de recomendación. Uso de Deep Autoencoders para comprender las preferencias del usuario para recomendar películas, libros, etc.
  - Extracción de características. En el proceso de minimizar el error de reconstrucción para reducir el error, aprende algunas de las características (features) importantes presentes en la entrada. La codificación genera un nuevo conjunto de características combinación de las características originales.
  - Reconocimiento de imágenes: Uso de varios codificadores apilados juntos para aprender diferentes características de una imagen.