Técnicas de Inteligencia Artificial

Óscar García

Tema 5. Redes Neuronales Artificiales

UNIVERSIDAD INTERNACIONAL DE LA RIOJA

Contenidos

- Objetivos
- Introducción. Fundamento biológico
- La neurona artificial. El perceptrón
- Redes neuronales multicapa
- Redes neuronales recurrentes. Redes Hopfield
- Hacia el *Deep Learning*
- Aplicaciones y ejemplos de implementación

Objetivos

- Entender el funcionamiento básico de la **neurona artificial** y el **perceptrón**.
- Describir las **redes neuronales multicapa**.
- Diferenciar entre redes de propagación hacia adelante (**feed forward**) y redes bidireccionales (**redes recurrentes**).
- Entender la evolución de las redes neuronales hacia el ***Deep Learning***.
- Identificar **aplicaciones prácticas** de redes neuronales.

Esquema

Redes Neuronales Artificiales

Conexión entre neuronales artificiales
El objetivo del aprendizaje es determinar los pesos que determinan la relevancia de las conexiones

Redes Neuronales Prealimentadas (Feed forward)

Perceptrón
Red neuronal más sencilla
→ Regla de aprendizaje basada en el error

Redes Neuronales Multicapa
Redes unidireccionales de alimentación hacia adelante (Feed Forward)
→ Método: retropropagación del error
→ Regla de aprendizaje basada en el gradiente del error

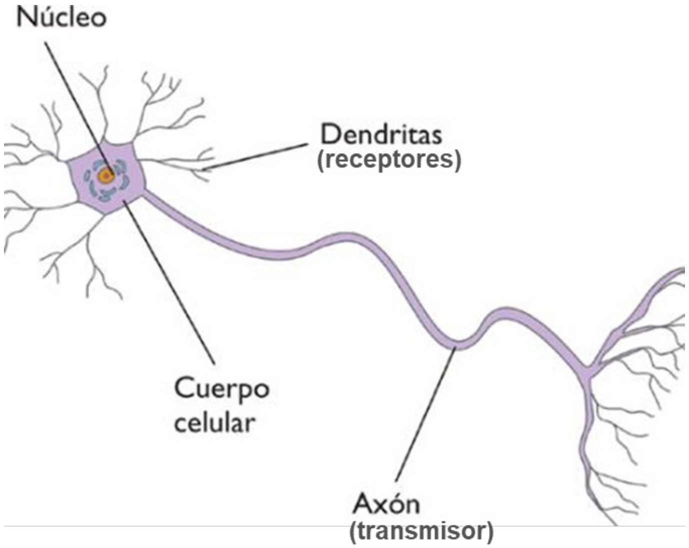
Redes Neuronales Recurrentes

Red Hopfield
→ Red monocapa auto-asociativa que memoriza información

Memoria bidireccional asociativa
Bidirectional Associative Memory (BAM)
→ Red multicapa que memoriza y asocia informaciones

Introducción. Fundamento biológico

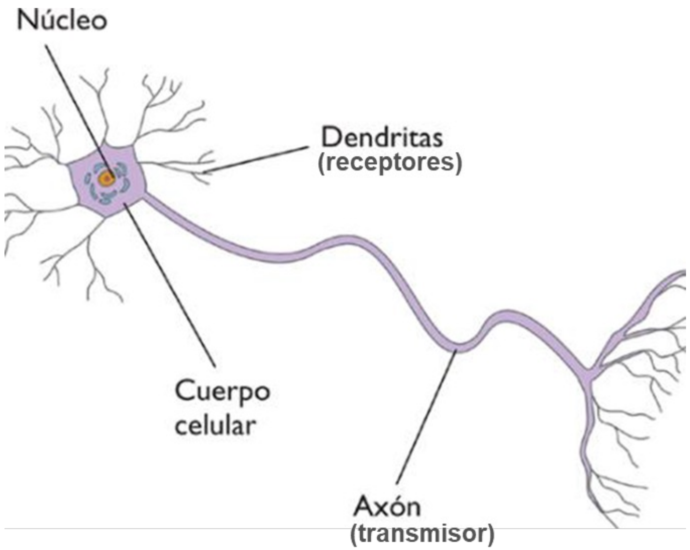
- **Redes neuronales artificiales**
  - Técnica de aprendizaje automático.
  - Emulan el funcionamiento del cerebro humano.
    - **Neuronas**: Unidades de procesamiento y transmisión
    - Un impulso nervioso excita la **dendrita**
    - Si se supera un **umbral**, se envía al **axón**
    - Se transmite a otras neuronas (**sinapsis**)

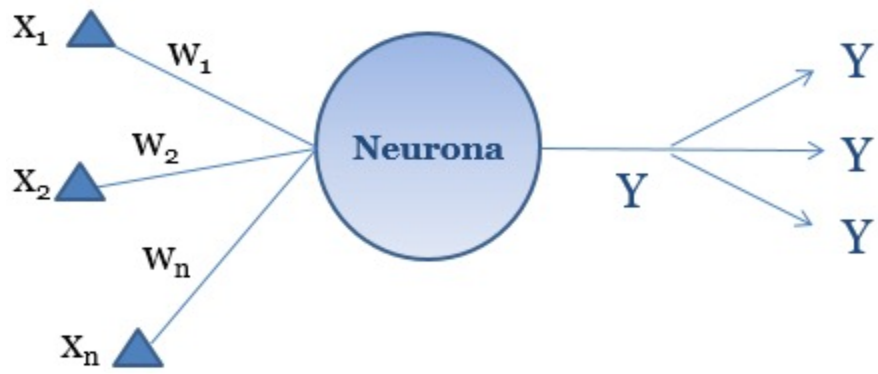


  - El cerebro procesa y almacena información creando nuevas conexiones con otras neuronas que se fortalecen o se debilitan según la información sea correcta o no.

Introducción. Fundamento biológico

- **Estructura de una neurona**

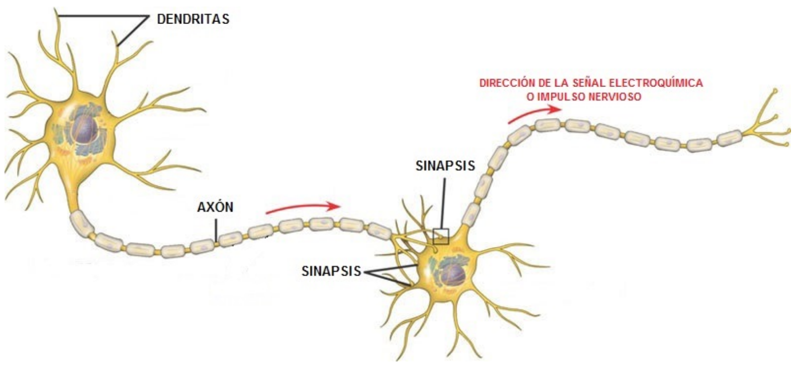




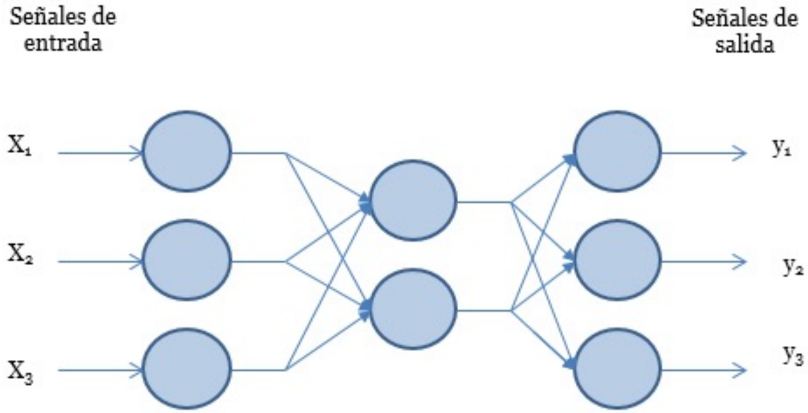
– Varias entradas: $x_1 , x_2 , \ldots , x_n$
– Una única salida Y que se puede ramificar en varias señales iguales.
– Umbral de excitación: las neuronas se excitan con un determinado nivel.

Introducción. Fundamento biológico

▸ **Estructura de una red neuronal**



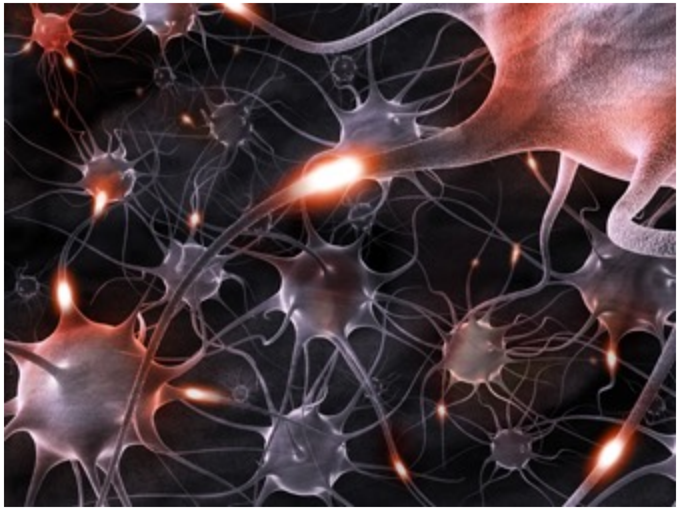
Fuente: https://sites.google.com/site/xmpanatomy/4-3-sinapsis-1



- Enlaces ponderados unen la salida a las entradas de otras neuronas.
- Pesos en los enlaces indican la relevancia de esa entrada a la neurona.
- Umbral de excitación se ajusta con los pesos.

Introducción

- **Redes neuronales artificiales son apropiadas para problemas de aprendizaje supervisado donde:**
  - Instancias con un gran número de atributos.
  - Cualquier valor de la salida (real, discreto, un vector con un conjunto de valores,…).
  - No se requiere un tiempo de entrenamiento corto (aunque para clasificar una nueva instancia si son rápidas).
  - No se requiere comprender la función objetivo.
  - Errores en los datos de entrenamiento (robustas frente a datos ruidosos).

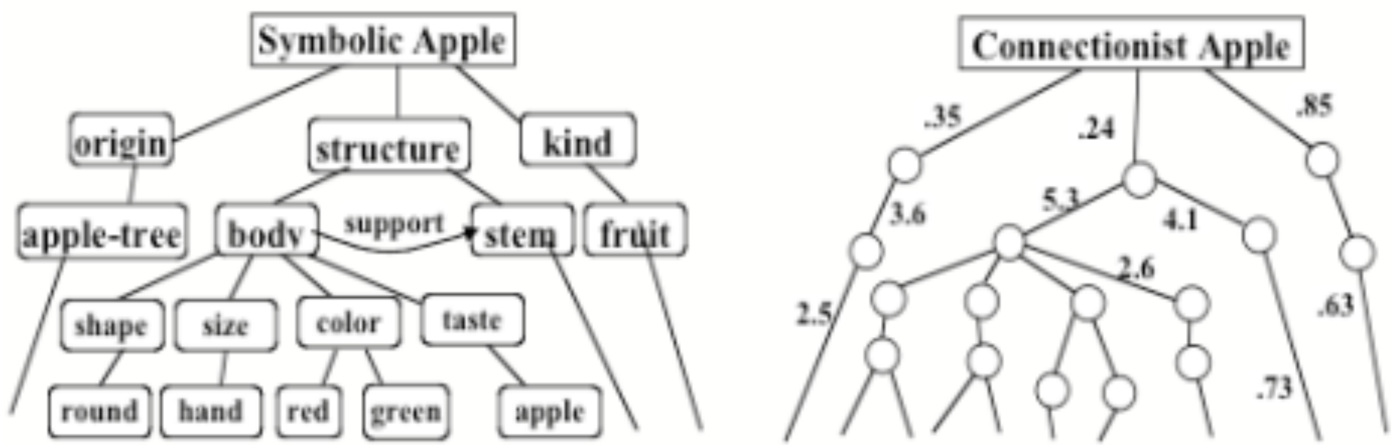


Introducción

- **Contrapartidas:**
  - Tiempos de entrenamiento elevados (no así el de ejecución).
  - Es difícil comprender la función objetivo.
  - Los pesos que se aprenden son difíciles de interpretar.
- Nuevas corrientes relacionadas en este sentido:
  - ***Inteligencia Artificial Explicable***
    - (XAI – *eXplainable Artificial Intelligence*)
  - Algoritmos híbridos neurosimbólicos, unen:
    - Explicabilidad de lógica simbólica
    - Precisión de métodos conexionistas

Introducción

- **Sistemas neurosimbólicos**, combinan:
  - **Modelos simbólicos**: árboles, reglas, fuzzy logic…
    - Dificultad para generalizar conocimiento
  - **Modelos conexionistas** (neuronales)
    - Dificultad para explicar los resultados



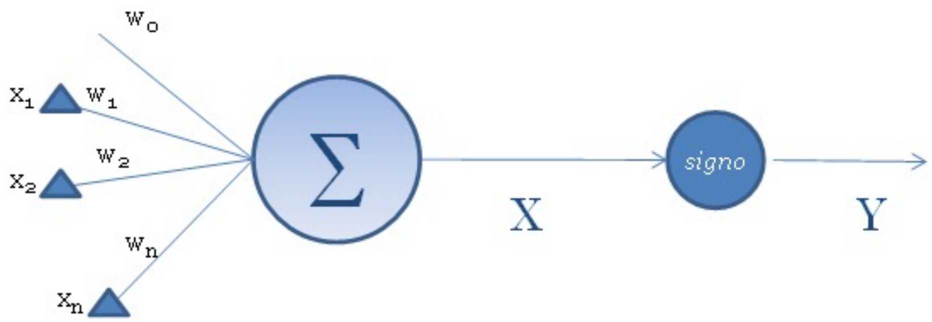
Fuente: Minsky, Marvin Lee, and Seymour Papert. Perceptrons: An Introduction to Computational Geometry. Expanded ed. Cambridge, Mass: MIT Press, 1988

Introducción

- Algunas **aplicaciones** de las redes neuronales
  - Reconocimiento de voz
  - Reconocimiento de escritura
  - Conducción de vehículos autónoma
  - Telemedicina: análisis de imágenes
  - Telecomunicaciones: en redes móviles e inalámbricas
  - Fintech: estudio de patrones de uso de tarjetas de crédito con el fin de detectar fraudes
  - Web maps: predecir áreas probables de ser solicitadas en un futuro con el fin de ser precargadas en una caché y mejorar la experiencia del usuario
  - Marketing: predicción de parámetros como el CTR
    - *Click Through Rate*

La neurona artificial. El perceptrón

- **Perceptrón:** red neuronal artificial más sencilla



– **Entrada ponderada:**

$$X = w_0 + \sum_{i=1}^{n} x_i w_i$$

$\boldsymbol{x_i}$: entrada $\boldsymbol{i}$
$\boldsymbol{w_i}$: peso de la entrada $\boldsymbol{x_i}$
$\boldsymbol{w_0}$: umbral

– **Salida utilizando la función signo:**

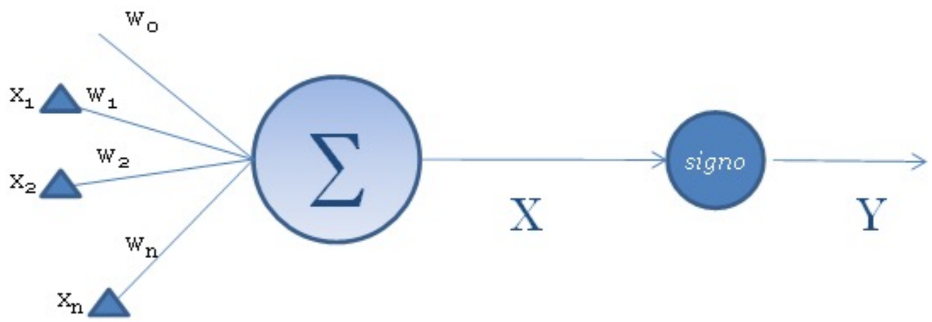
$$Y = \begin{cases} +1 & si\ X \geq 0 \\ -1 & si\ X < 0 \end{cases}$$

– **Función de activación:**

- Función signo, función escalón, función lineal…

La neurona artificial. El perceptrón

- **Perceptrón:**



  - Objetivo del aprendizaje: dado un conjunto de datos de entrenamiento (una serie de entradas a esa red y las salidas correspondientes), escoger los pesos (w_0 , w_1 , w_2 , … , w_n) que se ajusten mejor a esas entradas y salidas.
  - Regla de aprendizaje del perceptrón:

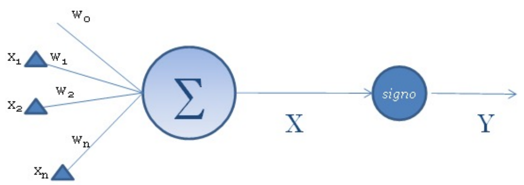
$$\boldsymbol{w_i(t+1) = w_i(t) + \alpha \times x_i(t) \times e(t)} \qquad e(t) = y_d(t) - y(t)$$

$\boldsymbol{e(t)}$: error (*coste, pérdida, loss*) (diferencia entre la salida esperada y la salida real en la iteración $\boldsymbol{t}$)

$\boldsymbol{\alpha}$: tasa de aprendizaje (valor entre 0 y 1 que pondera la relevancia del error en la última iteración)

La neurona artificial. El perceptrón

- **Algoritmo de aprendizaje del perceptrón:**



1. Asignar valores aleatorios en el intervalo [$-0.5, 0.5$] al umbral w_0 y a los pesos w_1 , w_2 , … , w_n.
2. Activar el perceptrón aplicando las entradas $x_1(t)$, $x_2(t)$, … , $x_n(t)$ y la salida deseada $y_d(t)$. Calcular la salida real en la iteración $y(t)$ utilizando una función de activación:

$$Y = \begin{cases} +1 & si\ X \geq 0 \\ -1 & si\ X < 0 \end{cases} \qquad X = w_0 + \sum_{i=1}^{n} x_i w_i$$

3. Actualizar los pesos :

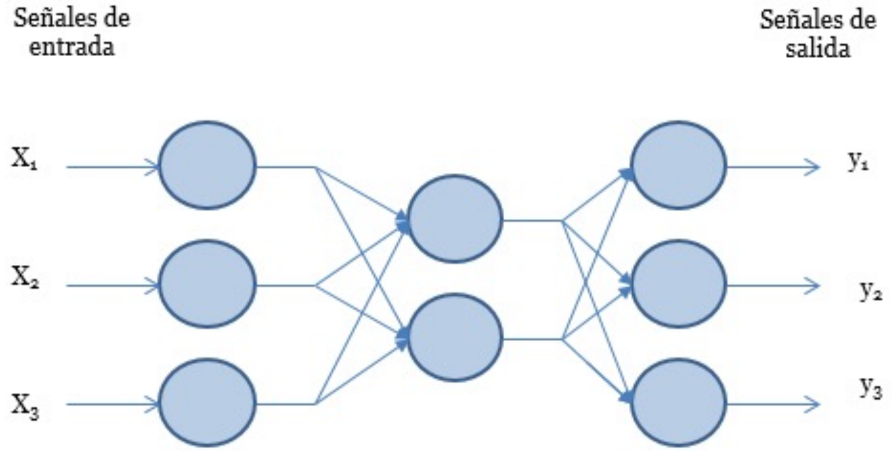
$$w_i(t+1) = w_i(t) + \alpha \times x_i(t) \times e(t) \qquad e(t) = y_d(t) - y(t)$$

4. Retornar al paso 2 hasta alcanzar convergencia.

En cada iteración se procesaría uno de los datos de entrenamiento disponibles.

La neurona artificial. El perceptrón

- **Red neuronal artificial:**
  - **Arquitectura de la red:**
    - Número de capas: capa de entrada, capa de salida y capas intermedias de neuronas.
    - Número de neuronas.
    - Conexiones entre neuronas.



  - **Función de activación**: función signo, función escalón, etc.
  - **Algoritmo de aprendizaje**: regla de aprendizaje para ajustar los pesos.

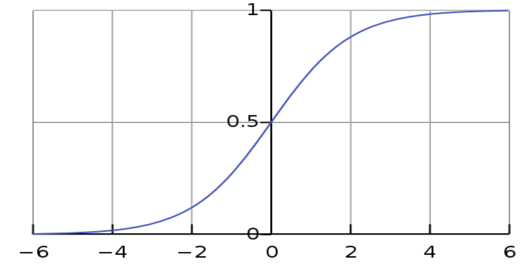
Funciones de activación

- Devuelve 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.
  - En el método de retropropagación, en la regla del gradiente, aplicaremos diferenciación

Fuente: http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/

Funciones de activación

- **Sigmoide**
  - Satura y mata el gradiente.
  - Lenta convergencia.
  - No está centrada en el cero.
  - Está acotada entre 0 y 1.
  - Buen rendimiento en la última capa.



Fuente: https://en.wikipedia.org/wiki/File:Logistic-curve.svg

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

Fuente: http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/

Funciones de activación

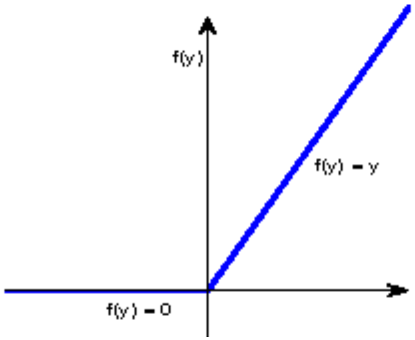
- **Tangente hiperbólica**
  - Muy similar a la signoide
  - Satura y mata el gradiente.
  - Lenta convergencia.
  - Centrada en 0.
  - Está acotada entre -1 y 1.
  - Se utiliza para decidir entre una opción y la contraria.
  - Buen desempeño en redes recurrentes.

$$f(x) = \frac{2}{1 + e^{-2x}} - 1$$

Fuente: http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/

Funciones de activación

- **ReLU – Rectified Lineal Unit**
  - Anula las entradas negativas y mantiene el valor de las entradas positivas.
  - Activación Sparse – sólo se activa si son positivos.
  - No está acotada.
  - Se pueden morir demasiadas neuronas.
  - Se comporta bien con imágenes.
  - Buen desempeño en redes convolucionales.

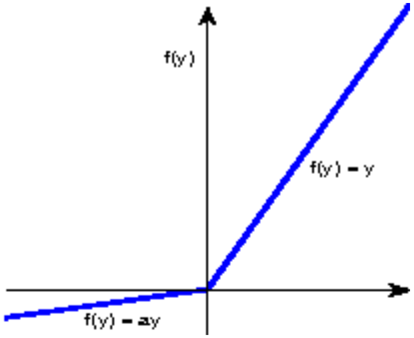


$$f(x) = x^+ = \max(0, x) = \begin{cases} 0 & si\ x < 0 \\ x & si\ x \geq 0 \end{cases}$$

Fuente: http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/
https://www.semanticscholar.org/paper/Investigation-of-parametric-rectified-linear-units-Sivadas-Wu/dd804425d9f394471ba974cda4925f50c21d8f02

Funciones de activación

- **Leaky ReLU – Leaky Rectified Lineal Unit**
  - Similar a la función ReLU.
  - Penaliza los negativos mediante un coeficiente rectificador.
    - Coeficiente $a \in (0,1)$
    - No está acotada.
  - Se comporta bien con imágenes.
  - Buen desempeño en redes convolucionales



$$f(x) = \begin{cases} ax & si\ x < 0 \\ x & si\ x \geq 0 \end{cases}$$

Fuente: http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/
https://www.semanticscholar.org/paper/Investigation-of-parametric-rectified-linear-units-Sivadas-Wu/dd804425d9f394471ba974cda4925f50c21d8f02

Funciones de activación

- **Softmax**
  - Se utiliza cuando queremos tener una representación en forma de probabilidades.
  - El sumatorio de todas las probabilidades de las salidas es 1.
  - Esta acotada entre 0 y 1.
  - Muy diferenciable.
  - Se utiliza para para normalizar tipo multiclase.
  - Buen rendimiento en las últimas capas.

$$f(Z)_j = \frac{e^{Z_j}}{\sum_{k=1}^{K} e^{Z_k}}$$

Fuente: http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/

Hiperparámetros

- **Función de activación**
- **Tasa de aprendizaje (*learning rate*)**
- **Epoch:** Ciclos en los que se ejecutan los algoritmos de forma completa. Iteraciones si elegimos un *batch*.
- **Batch:** Cantidad de muestras con las que se entrena a la vez. Optimización y velocidad. Ejecuciones con menos datos. Paralelizar
- **Número de capas y neuronas**
- **Función de pérdidas (*lost function*):** Mide el desempeño (entropía para clasificación o error cuadrático medio para regresiones)
- **Algoritmo de optimización:** Descenso del gradiente, Adam

Tres conjuntos de datos : *Training*, *Validation* y *Test*

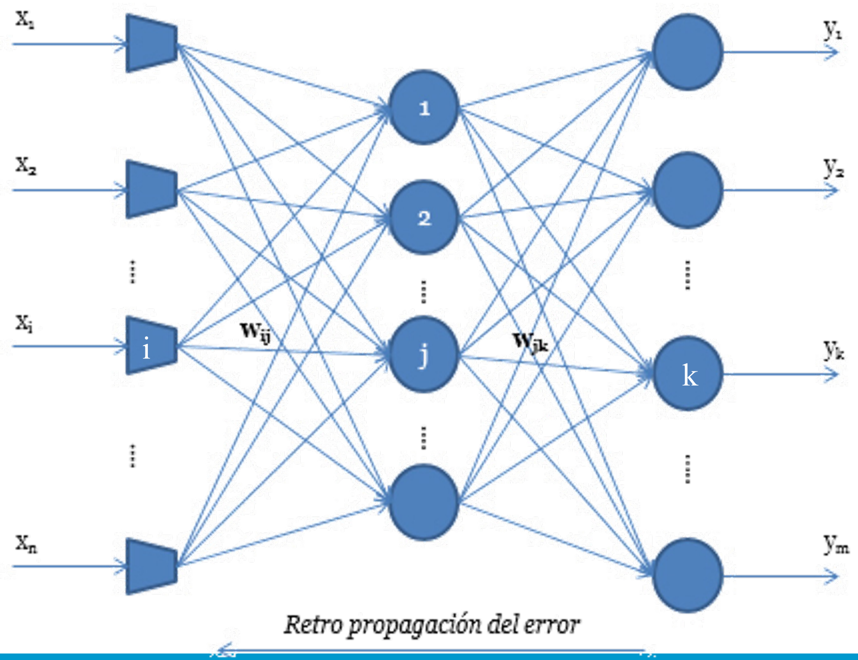
¿Qué función de activación y de pérdidas debo utilizar?

Problem Type	Output Type	Final Activation Function	Loss Function
Regression	Numerical value	Linear	Mean Squared Error (MSE)
Classification	Binary outcome	Sigmoid	Binary Cross Entropy
Classification	Single label, multiple classes	Softmax	Cross Entropy
Classification	Multiple labels, multiple classes	Sigmoid	Binary Cross Entropy

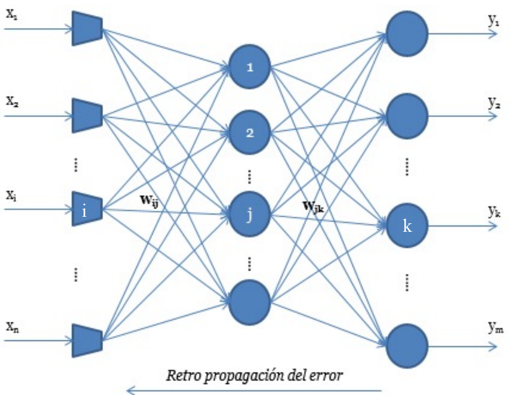
Fuente: https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8

Redes neuronales multicapa

- **Características:**
  - Redes unidireccionales de alimentación hacia adelante (*feedforward*).
  - Al menos una capa intermedia oculta
    - Comerciales: 1 ó 2 capas ocultas con 10 a 1 000 neuronas.
    - Deep Learning: hasta 1 000 capas ocultas (*resnet*, visión artificial)
- **Redes de retropropagación:**
  - Método de aprendizaje: back-propagation (retropropagación del error).
  - Neuronas conectadas con todas las de la capa anterior y posterior.



Redes neuronales multicapa

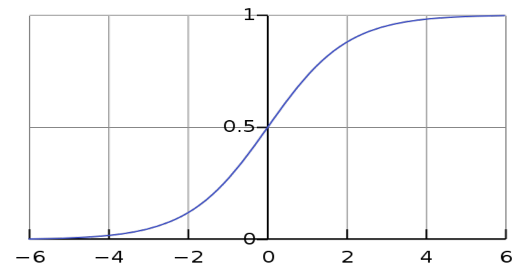


- **Método de back–propagation:**
  - Entrada ponderada:

$$X = w_0 + \sum_{i=1}^{n} x_i w_i$$

  - Función de activación: función sigmoide

$$Y = \frac{1}{1 + e^{-X}}$$



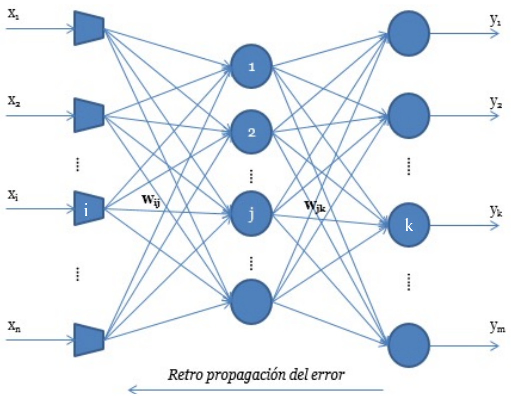
Fuente: https://en.wikipedia.org/wiki/File:Logistic-curve.svg

  - Regla de aprendizaje: gradiente del error

$$\delta_k(t) = \frac{\partial y_k(t)}{\partial X_k(t)} \cdot e_k\,(t) = \frac{\partial y_k(t)}{\partial X_k(t)} \cdot \left(y_{d,k}(t) - y_k(t)\right)$$

$X_k(t)$: entrada ponderada a la neurona k en la iteración t
$y_k(t)$: salida real de la neurona k en la iteración t
$y_{d,k}(t)$: salida esperada en la neurona k

Redes neuronales multicapa



- **Método de back–propagation:**
  - Gradiente del error para una neurona $\boldsymbol{k}$ en la capa de salida:

$$\delta_k(t) = \frac{\partial y_k(t)}{\partial X_k(t)} \cdot e_k\,(t) = \frac{\partial y_k(t)}{\partial X_k(t)} \cdot \left(y_{d,k}(t) - y_k(t)\right)$$

función sigmoide $Y = \dfrac{1}{1+e^{-X}}$

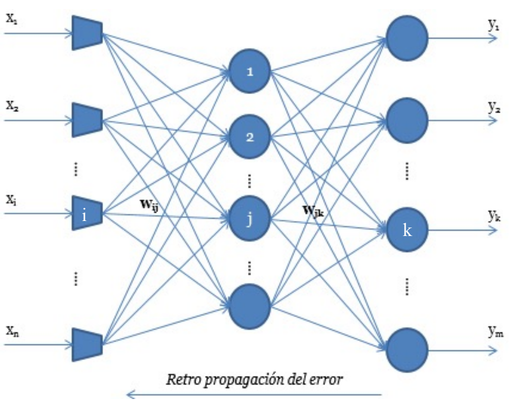
$$\delta_k(t) = y_k(t) \cdot \left(1 - y_k(t)\right) \cdot e_k\,(t) = y_k(t) \cdot \left(1 - y_k(t)\right) \cdot \left(y_{d,k}(t) - y_k(t)\right)$$

  - Pesos de los enlaces entre la neurona $\boldsymbol{j}$ y la neurona $\boldsymbol{k}$ en la capa de salida ($\boldsymbol{w_{jk}}$) – Reajuste de los pesos:

$$w_{jk}(t+1) = w_{jk}(t) + \alpha \cdot y_j(t) \cdot \delta_k(t)$$

$\boldsymbol{y_j(t)}$: salida de la neurona $\boldsymbol{j}$ en la capa oculta
$\boldsymbol{\alpha}$: tasa de aprendizaje

Redes neuronales multicapa



- **Método de back–propagation:**

  – Gradiente del error para una neurona ***j*** en la capa oculta

$$\delta_j(t) = y_j(t) \cdot \left(1 - y_j(t)\right) \cdot \sum_{k=1}^{m} \delta_k(t) w_{jk}(t)$$

$\boldsymbol{m}$: número de neuronas en la capa de salida
$\boldsymbol{\delta_k(p)}$: gradiente del error para la neurona $\boldsymbol{k}$ en la capa de salida
$\boldsymbol{w_{jk}(p)}$: peso de la conexión entre la neurona $\boldsymbol{j}$ y la neurona $\boldsymbol{k}$
$\boldsymbol{y_j(t)}$: salida de la neurona $\boldsymbol{j}$ en la capa oculta para la iteración $\boldsymbol{t}$

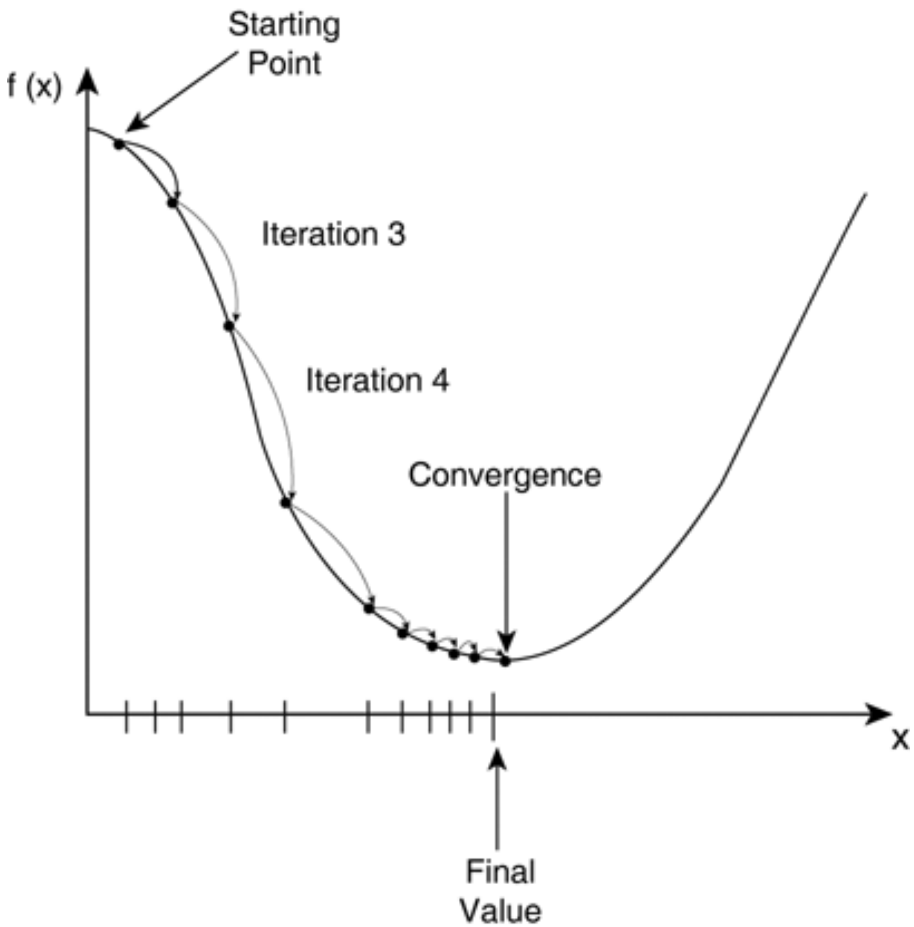
$$y_j(t) = \frac{1}{1 + e^{-X_j(t)}}$$

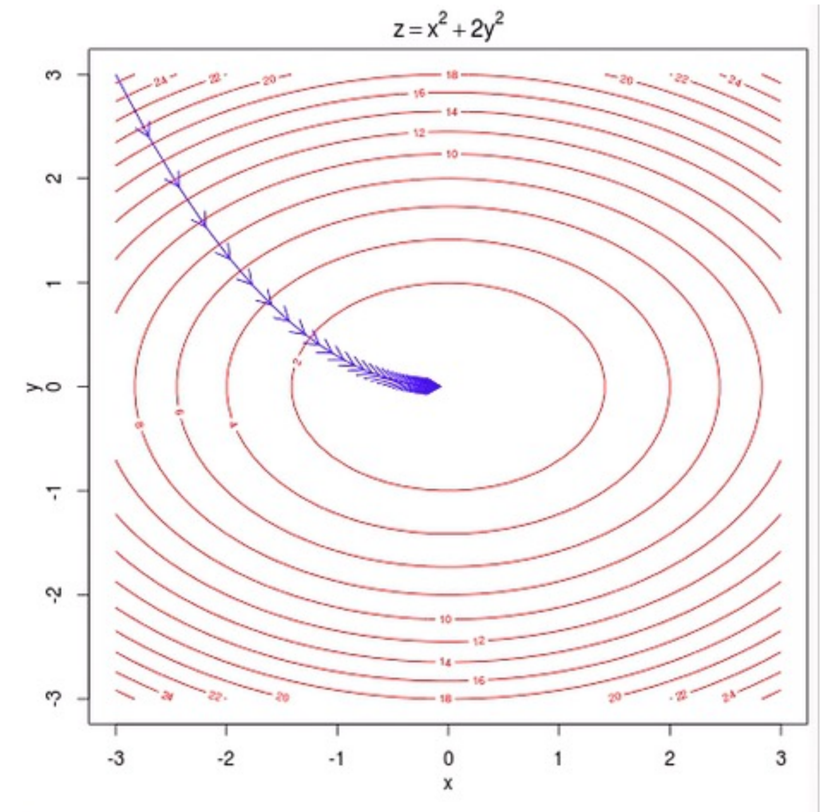
$$X_j(t) = w_{0j} + \sum_{i=1}^{n} x_i(t) \times w_{ij}(t)$$

$\boldsymbol{w_{ij}(t)}$: peso de la conexión entre la neurona $\boldsymbol{i}$ y la neurona $\boldsymbol{j}$ (capa oculta)

Redes neuronales multicapa

- **Descenso del gradiente**

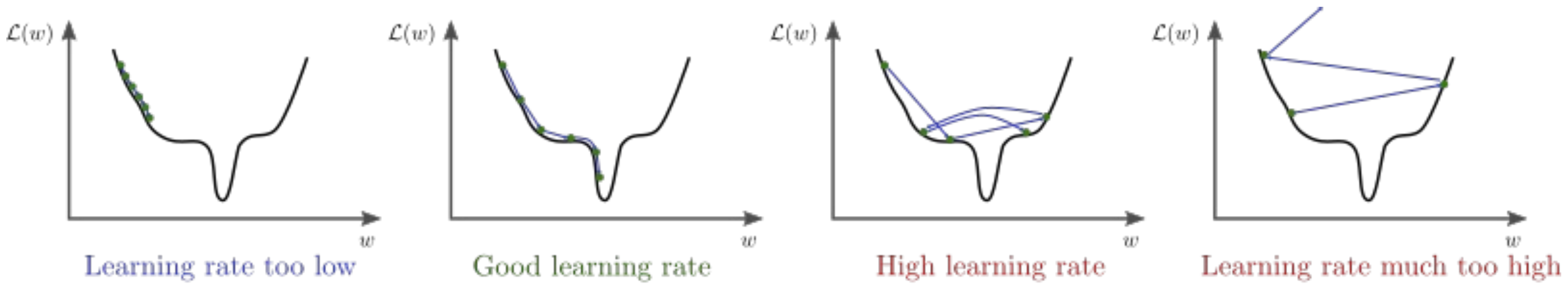


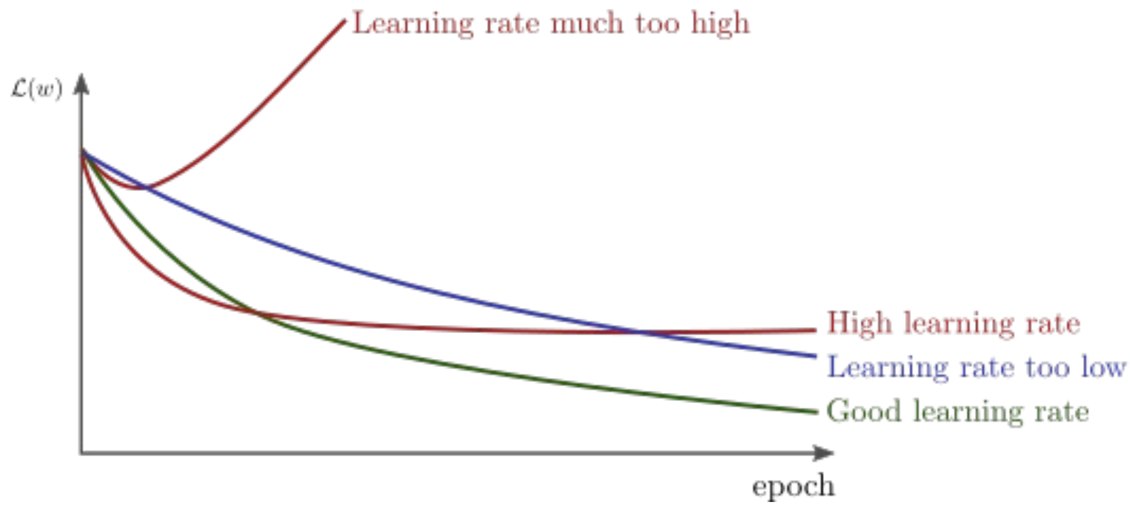


Fuente: http://www.cs.us.es/~fsancho/?e=165

Redes neuronales multicapa

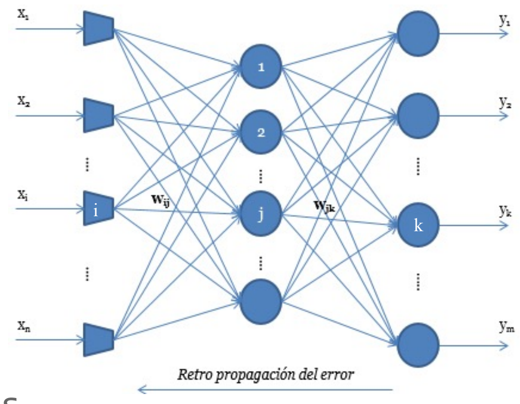
- **Tasa de aprendizaje**





Fuente: https://medium.com/metadatos/todo-lo-que-necesitas-saber-sobre-el-descenso-del-gradiente-aplicado-a-redes-neuronales-19bdbb706a78

Redes neuronales multicapa



- **Algoritmo de aprendizaje de redes multicapa de 3 capas:**

1. Iniciar los pesos con valores aleatorios pequeños.
2. Para cada dato de entrada $x_1(t), x_2(t), \dots, x_n(t)$ calcular las salidas de la red al activarla.
3. Ajustar los pesos mediante retropropagación de errores:

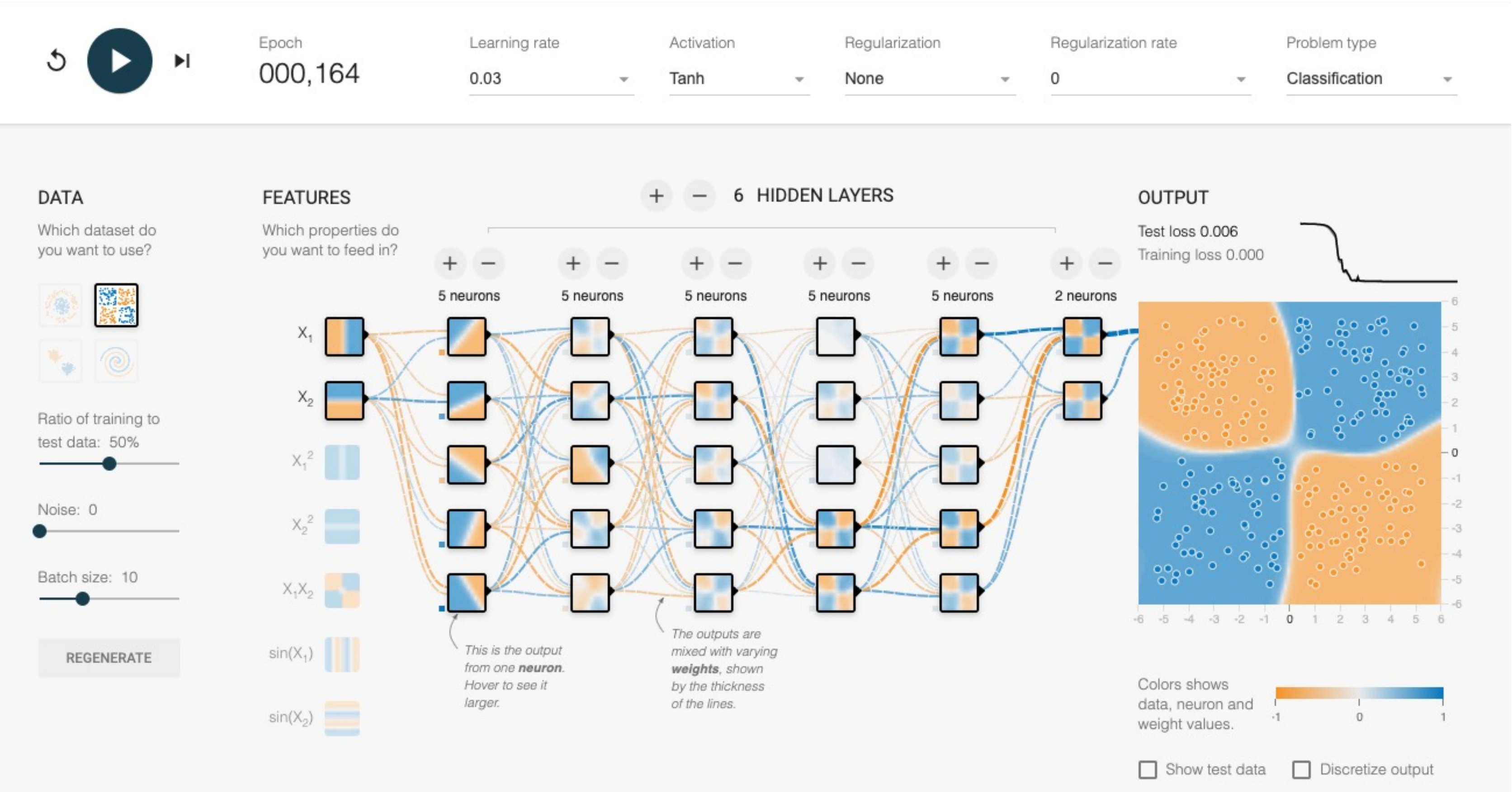
   Fase 1. Calcular el gradiente del error para las neuronas de la capa de salida $\delta_k(t) = y_k(t) \cdot (1 - y_k(t)) \cdot e_k(t) = y_k(t) \cdot (1 - y_k(t)) \cdot \left(y_{d,k}(t) - y_k(t)\right)$ y reajustar los pesos $w_{jk}(t+1) = w_{jk}(t) + \alpha \cdot y_j(t) \cdot \delta_k(t)$

   Fase 2. Calcular el gradiente del error para las neuronas en la capa oculta $\delta_j(t) = y_j(t) \cdot \left(1 - y_j(t)\right) \cdot \sum_{k=1}^{m} \delta_k(t) w_{jk}(t)$ y reajustar los pesos
   $w_{ij}(t+1) = w_{ij}(t) + \alpha \cdot x_i(t) \cdot \delta_j(t)$

4. Repetir los pasos 2 y 3 hasta que se cumpla la convergencia (errores pequeños) o un criterio de parada.

Redes neuronales multicapa

- **Tensorflow Playground**

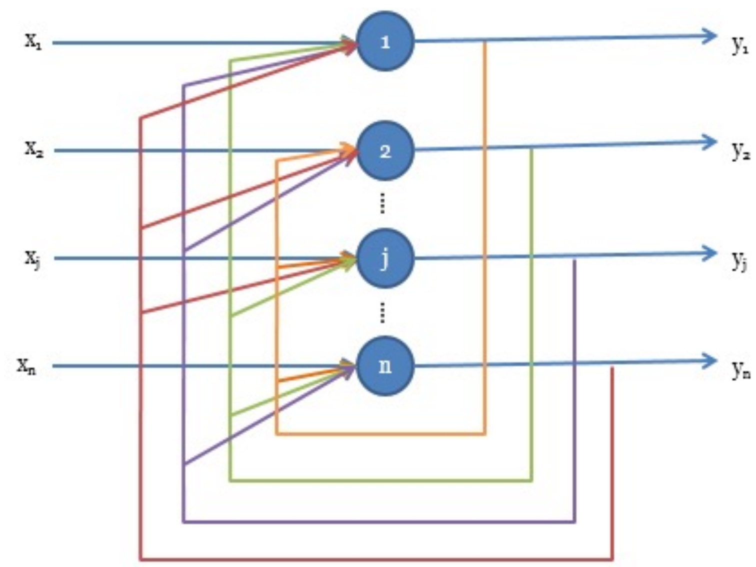


Fuente: https://playground.tensorflow.org/

Redes neuronales recurrentes. Hopfield Network

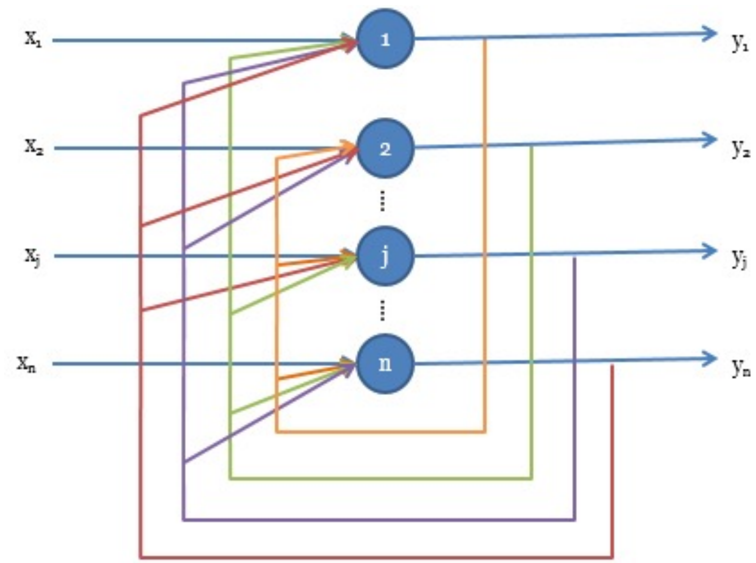
- **Características:**
  - Emulan las características asociativas de la memoria humana.
  - Las salidas de las neuronas alimentan las entradas a las otras neuronas: bucle.

- **Red Hopfield:**
  - Red autoasociativa que puede almacenar varias informaciones durante la etapa de aprendizaje.
  - Similar a una memoria.



Redes neuronales recurrentes. Hopfield Network

- **Red Hopfield monocapa:**
  - Una única capa con n neuronas.
  - Salidas retroalimentan las entradas de las otras neuronas (no autoretroalimentación).
  - Salidas binarias o números reales.
  - Función de activación más habitual: función signo

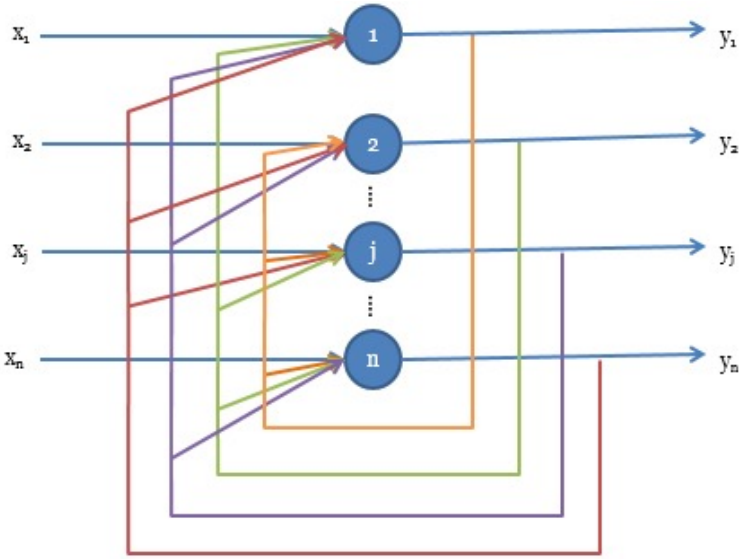


$$y^{signo} = \begin{cases} +1, si\ X > 0 \\ -1, si\ X < 0 \\ Y, si\ X = 0 \end{cases}$$

  - Estado de la red: conjunto de salidas $[y_1, y_2, \ldots y_n]$
  - Objetivo de la red: almacenar unos determinados estados $Y_1, Y_2, \ldots Ym \ldots Y_M$, denominados memorias fundamentales.

Redes neuronales recurrentes. Hopfield Network

- **Memorias fundamentales**
  - En el entrenamiento la red almacena los estados $Y_1, Y_2, \ldots Ym \ldots Y_M$
  - Una vez finalizado el aprendizaje
    - Si a la entrada se presenta una de las memorias fundamentales, la red se estabiliza ofreciendo a la salida la misma información.
    - Si la entrada no coincide con una memoria fundamental, la red evoluciona ofreciendo una salida lo más parecida a las informaciones almacenadas.

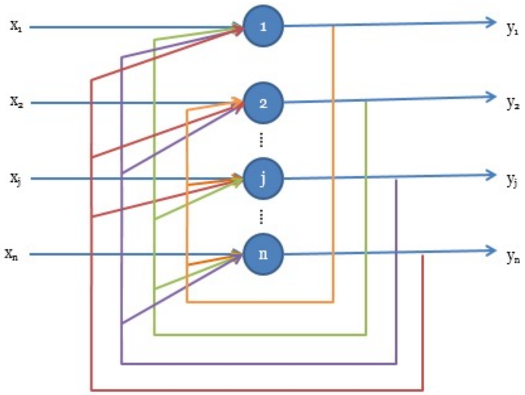


Redes neuronales recurrentes. Hopfield Network

- **Algoritmo de entrenamiento**

1. Cálculo de la matriz de pesos.

Peso $\boldsymbol{w_{ij}}$ entre las neuronas $\boldsymbol{i}$ y $\boldsymbol{j}$:



$$w_{ij} = \begin{cases} \sum_{m=1}^{M} y_{m,i} y_{m,j}, & i \neq j \\ 0 & i = j \end{cases}$$

$\boldsymbol{y_{m,i}}$, $\boldsymbol{y_{m,j}}$: elementos $\boldsymbol{i}$ y $\boldsymbol{j}$ del vector de salida deseado $\boldsymbol{Y_m}$

$\boldsymbol{M}$: número de estados a memorizar

Expresión matricial de los pesos:

$$W = \left(\sum_{m=1}^{M} Y_m Y_m^T \right) - MI$$

$\boldsymbol{Y_m}$: vector n-dimensional que se quiere memorizar

$\boldsymbol{MI}$: matriz identidad de dimensiones $n \times n$

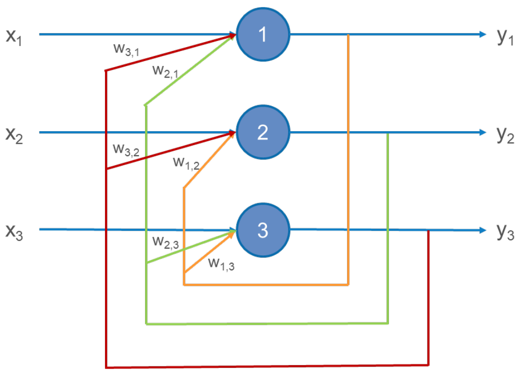
Redes neuronales recurrentes. Hopfield Network

- **Algoritmo de entrenamiento - Ejemplo**
  1. Cálculo de la matriz de pesos.

     Almacenar las memorias fundamentales:

     $Y_1 = [1,1,1]$

     $Y_2 = [-1,-1,-1]$

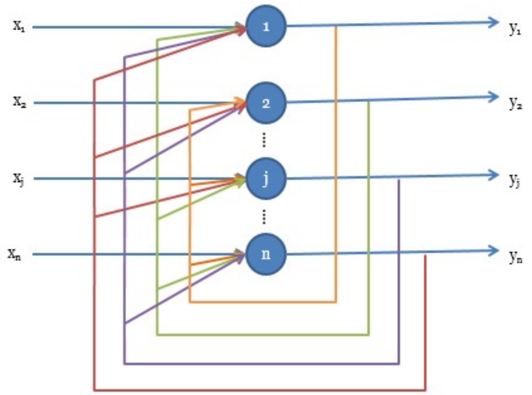


$$Y_1 Y_1^T - I = \begin{bmatrix}1\\1\\1\end{bmatrix}\begin{bmatrix}1 & 1 & 1\end{bmatrix} - \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix} = \begin{bmatrix}0 & 1 & 1\\1 & 0 & 1\\1 & 1 & 0\end{bmatrix}$$

$$Y_2 Y_2^T - I = \begin{bmatrix}-1\\-1\\-1\end{bmatrix}\begin{bmatrix}-1 & -1 & -1\end{bmatrix} - \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix} = \begin{bmatrix}0 & 1 & 1\\1 & 0 & 1\\1 & 1 & 0\end{bmatrix}$$

$$W = \begin{bmatrix}0 & w_{12} & w_{13}\\w_{21} & 0 & w_{23}\\w_{31} & w_{32} & 0\end{bmatrix} = \begin{bmatrix}0 & 2 & 2\\2 & 0 & 2\\2 & 2 & 0\end{bmatrix}$$

Redes neuronales recurrentes. Hopfield Network



- **Algoritmo de entrenamiento**
  2. Comprobación de que la red es capaz de memorizar las memorias fundamentales.

     Para cada memoria fundamental Y_m de n componentes, $m=1,2,\ldots M$, si se da una entrada tal que

     $$x_{m,i} = y_{m,i} \qquad i=1,2,\ldots.n$$

     Y si la función de activación es la función signo, se ha de cumplir lo siguiente:

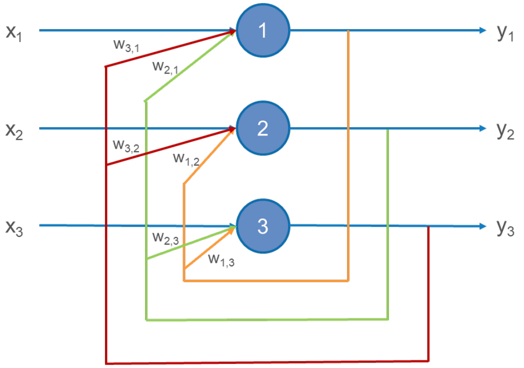
     $$y_{m,i} = signo\left(\sum_{j=1}^{n} w_{ij}\, x_{m,j} - w_{0,i}\right)$$

     Expresión matricial:

     $$Y_m = signo(W X_m - W_0)$$

Redes neuronales recurrentes. Hopfield Network

- **Algoritmo de entrenamiento - Ejemplo**
  2. Comprobación de que la red es capaz de memorizar las memorias fundamentales.



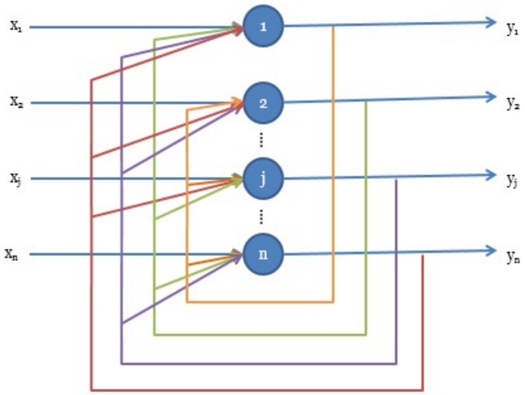
Si los umbrales w_0 son iguales a 0, se puede comprobar cómo efectivamente, a partir de una entrada [1 1 1] igual a una de las memorias fundamentales Y_1, la red obtiene la salida correspondiente a esa memoria fundamental:

$$Y_1 = signo\left(\begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Redes neuronales recurrentes. Hopfield Network

- **Algoritmo de entrenamiento**

  3. Comprobación con entradas de prueba de que la red recupera un estado estable.



Se toma un vector $X = [x_0, x_1, \ldots x_n]$ diferente las memorias fundamentales.

Se aplica como entrada a la red obteniendo como salida:

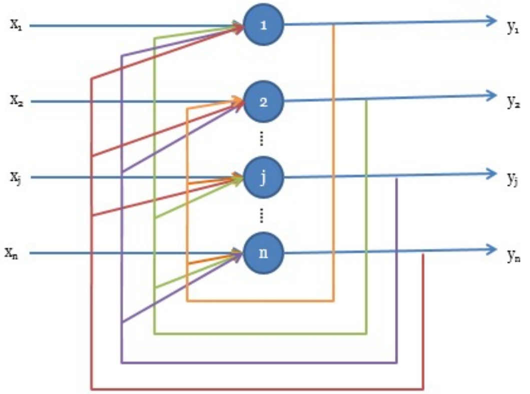
$$y_i(0) = signo\left(\sum_{j=1}^{n} w_{ij}\, x_j(0) - w_{0,i}\right)$$

$x_j(0)$: componente j de la entrada en la iteración t = 0
$y_i(0)$: salida de la neurona *i* en la iteración t=0

Expresión matricial:

$$Y(0) = signo(WX(0) - W_0)$$

Redes neuronales recurrentes. Hopfield Network



- **Algoritmo de entrenamiento**

3. Comprobación con entradas de prueba de que la red recupera un estado estable.

Se repite el paso anterior hasta que el vector de estado o la salida se mantiene estable, no cambia.

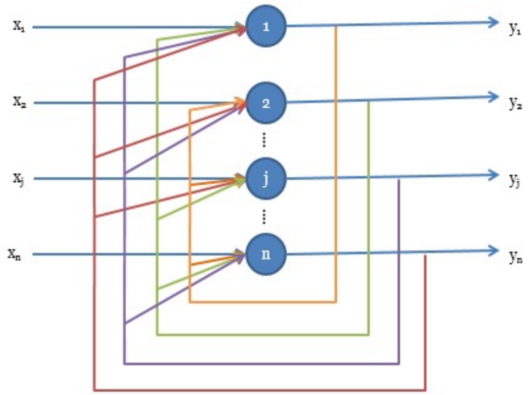
Condición de estabilidad:

$$y_i(t+1) = signo\left(\sum_{j=1}^{n} w_{ij}\, y_j(t) - w_{0,i}\right)$$

Expresión matricial:

$$Y(t+1) = signo(WY(t) - W_0)$$

Redes neuronales recurrentes. Hopfield Network



- **Algoritmo de entrenamiento**
  3. Comprobación con entradas de prueba de que la red recupera un estado estable.

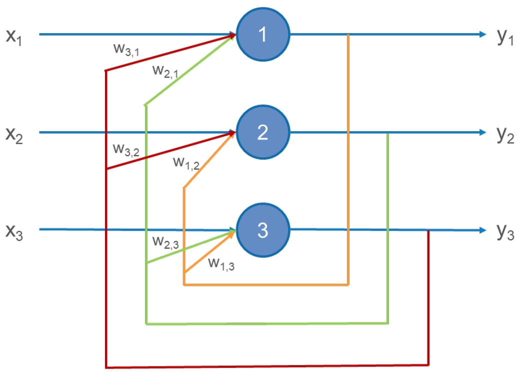
     En cada iteración, las neuronas reciben como entrada las salidas de las otras neuronas multiplicadas por el peso correspondiente.

     - **Red de Hopfield síncrona**: la actualización de las salidas de las neuronas de la red se realiza de forma simultánea.
       - En la iteración $(t+1)$ todas las neuronas van a utilizar como entradas las salidas generadas por las otras neuronas en la iteración t.
     - **Red de Hopfield asíncrona**: se actualiza solo la salida de una neurona en cada iteración → la salida a la que converge dependerá del orden de activación de las neuronas.

Redes neuronales recurrentes. Hopfield Network

- **Algoritmo de entrenamiento - Ejemplo**

3. Comprobación con entradas de prueba de que la red recupera un estado estable.



Si se tiene una entrada de prueba [-1 1 1], la salida en la iteración t=0 es:

$$Y(0) = signo\left(\begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}\begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

La salida no coincide con la entrada, la red no está estable. Esa salida se aplica de nuevo a la entrada y se obtiene en la iteración t=1:

$$Y(1) = signo\left(\begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Redes neuronales recurrentes. Hopfield Network

- **Ejemplo patrones**
  - Aprendizaje de dos patrones correspondientes a dos figuras de 2×2 píxeles:

Figura 1:

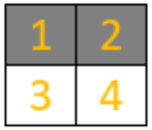
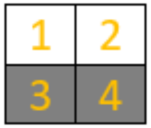


Figura 2:



  - Píxel gris → valor 1 - Píxel blanco → valor -1
  - Valores de entrada:

$$Y_1 = [1 \quad 1 \quad -1 \quad -1]$$

$$Y_2 = [-1 \quad -1 \quad 1 \quad 1]$$

  - En este caso: dos vectores de información ($M = 2$) de cuatro elementos ($N = 4$) que contienen los valores de los píxeles.
  - Red de cuatro neuronas.

Ejemplo basado en el presentado en: https://es.slideshare.net/mentelibre/redes-neuronales-de-hopfield

Redes neuronales recurrentes. Hopfield Network

- **Ejemplo patrones**
  - Obtención de la matriz W:

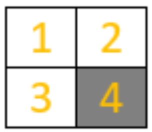
$$Y_1Y_1^T - \quad I = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 1 & -1 & -1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ -1 & -1 & 0 & 1 \\ -1 & -1 & 1 & 0 \end{bmatrix}$$

$$Y_2Y_2^T - \quad I = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & -1 & 1 & 1 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 & -1 \\ 1 & 0 & -1 & -1 \\ -1 & -1 & 0 & 1 \\ -1 & -1 & 1 & 0 \end{bmatrix}$$

$$W = \begin{bmatrix} 0 & 2 & -2 & -2 \\ 2 & 0 & -2 & -2 \\ -2 & -2 & 0 & 2 \\ -2 & -2 & 2 & 0 \end{bmatrix}$$

Redes neuronales recurrentes. Hopfield Network

- **Ejemplo patrones**
  - Prueba:



$$Y_p = [-1 \quad -1 \quad -1 \quad 1]$$

  - Primera iteración:

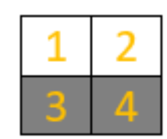
$$Y_pW = [-1 \quad -1 \quad -1 \quad 1]\begin{bmatrix} 0 & 2 & -2 & -2 \\ 2 & 0 & -2 & -2 \\ -2 & -2 & 0 & 2 \\ -2 & -2 & 2 & 0 \end{bmatrix} = [-2 \quad -2 \quad 6 \quad 2]$$

$$S = [-1 \quad -1 \quad 1 \quad 1]$$

  - Segunda iteración:

$$Y_pW = [-1 \quad -1 \quad 1 \quad 1]\begin{bmatrix} 0 & 2 & -2 & -2 \\ 2 & 0 & -2 & -2 \\ -2 & -2 & 0 & 2 \\ -2 & -2 & 2 & 0 \end{bmatrix} = [-6 \quad -6 \quad 6 \quad 6]$$

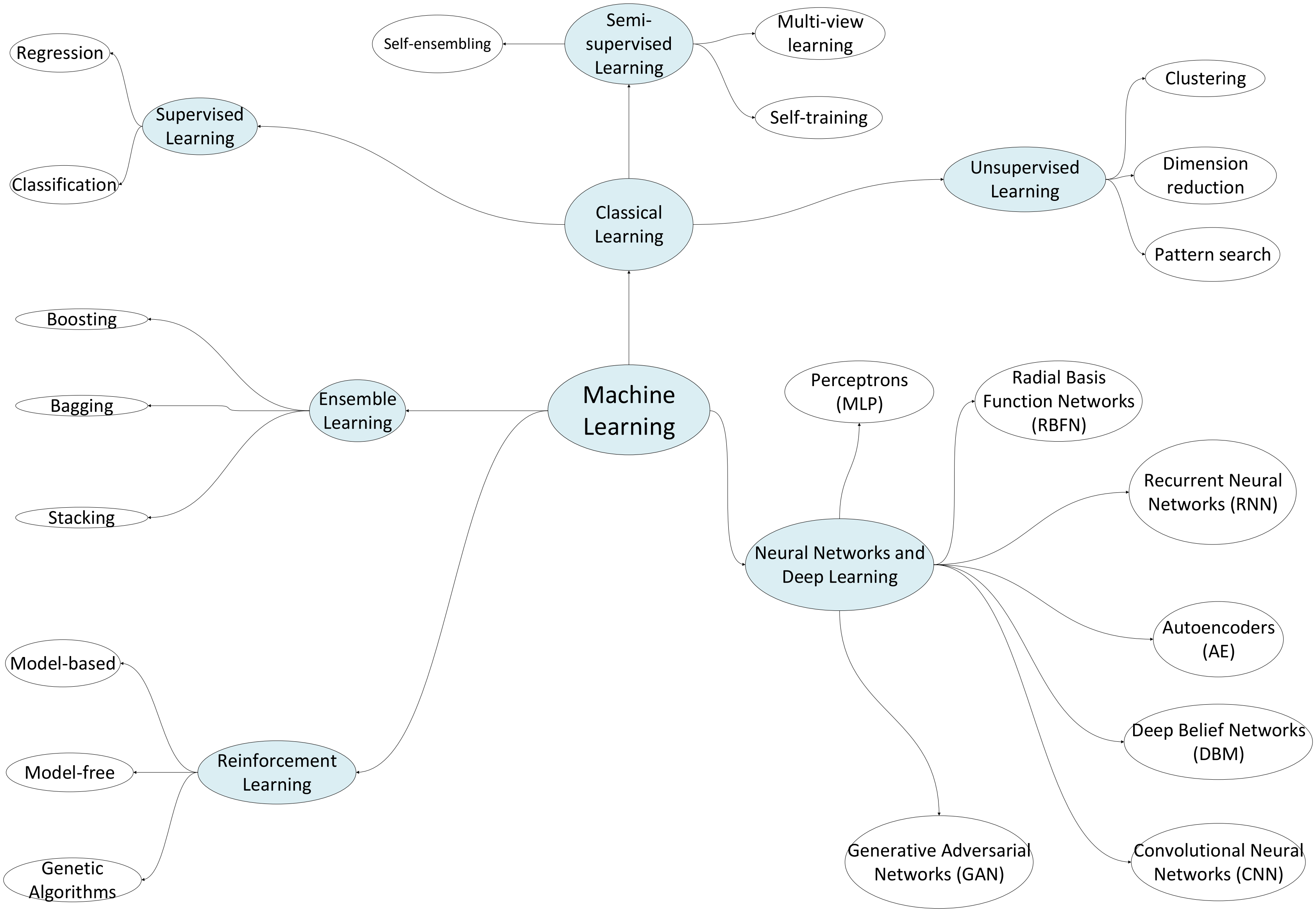
$$S = [-1 \quad -1 \quad 1 \quad 1]$$



Redes neuronales recurrentes. Hopfield Network

- **Ventajas**
  - Recuperar memorias incompletas o información completa a partir de información incompleta.
- **Limitaciones**
  - No siempre el estado estable que se alcanza es una memoria fundamental.
  - Limitación de capacidad: número de informaciones que puede ser aprendido o almacenado es limitado.
  - No pueden asociar una información con otra.
    - Red recurrente de dos niveles: memorias bidireccionales asociativas capaces de asociar informaciones diferentes.
- **Aplicaciones**
  - Reconocimiento de imágenes o de patrones
  - Problemas de optimización

Hacia el *Deep Learning*



Hacia el *Deep Learning*

- Redes de retropropagación presentan limitaciones
  - No son eficaces en problemas complejos en los que el número de capas intermedias es elevado
  - Proceso de entrenamiento lento y costoso
- Aparición del *Deep Learning*
  - Redes neuronales con alto número de nodos y capas
  - Volumen elevado de datos (*Big Data*)
  - Nuevos algoritmos más eficientes y eficaces
  - Permiten aplicar técnicas no supervisadas en capas intermedias para que éstas aprendan automáticamente en base a la experiencia conceptos no conocidos
  - Extracción de características y clasificación en la misma red

Aplicaciones de redes neuronales (y el DL)

Todos los algoritmos supervisados y no supervisados

- Identificación de objetos en imágenes y vídeos
- Reconocimiento de voz
- Síntesis de voz
- Análisis de sentimientos y reconocimiento de emociones del habla
- Procesamiento de imágenes
- Transferencia de estilos
  - Aplicación del estilo de pintura de Van Gogh a cualquier fotografía, por ejemplo
- Procesamiento del lenguaje natural (*Natural Language Processing*)
  - Traducción automática

Ejemplos con Python

- Tomaremos el dataset **iris de Fisher**
  - https://www.openml.org/d/61
  - https://www.openml.org/data/get_csv/61/dataset_61_iris.arff
  - Cuatro características
    - Longitud y anchura de pétalo
    - Longitud y anchura de sépalo
  - Tres tipos de flores (clases)
    - *Iris setosa (50)*
    - *Iris virginica (50)*
    - *Iris versicolor (50)*

Largo de sépalo	Ancho de sépalo	Largo de pétalo	Ancho de pétalo	Especies
5.1	3.5	1.4	0.2	I. setosa
4.9	3.0	1.4	0.2	I. setosa
4.7	3.2	1.3	0.2	I. setosa

Ejemplos con Python

- `pip install pandas`
- `pip install scikit-learn`
  - Random Forest
  - k-NN
  - Clasificador Naïve Bayes
  - SVM (*Support Vector Machine*)
  - MLP (*Multi-Layer Perceptron*)
- `pip install matplotlib`
- `pip install seaborn`
  - Basado en matplotlib

Ejemplos con Python: explorando los datos

```python
# Load libraries
from pandas import read_csv

url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.arff"
# La siguiente línea no es necesaria usando esta fuente, pues ya incluye la
# cabecera
#names = ['sepallength', 'sepalwidth', 'petallength', 'petalwidth', 'class']
#dataset = read_csv(url, names=names)
dataset = read_csv(url)

# mostramos la "forma", debería haber 150 entradas con 5 atributos cada una
print(dataset.shape)
#> (150 , 5)

# mostramos las 3 primeras entradas para echar un vistazo
print(dataset.head(3))
#>   sepallength  sepalwidth  petallength  petalwidth        class
#>0          5.1         3.5          1.4         0.2  Iris-setosa
#>1          4.9         3.0          1.4         0.2  Iris-setosa
#>2          4.7         3.2          1.3         0.2  Iris-setosa
```

Ejemplos con Python: explorando los datos

```
# mostramos un resumen estadístico de los datos
print(dataset.describe())
#>        sepallength  sepalwidth  petallength  petalwidth
#>count    150.000000  150.000000   150.000000  150.000000
#>mean       5.843333    3.054000     3.758667    1.198667
#>std        0.828066    0.433594     1.764420    0.763161
#>min        4.300000    2.000000     1.000000    0.100000
#>25%        5.100000    2.800000     1.600000    0.300000
#>50%        5.800000    3.000000     4.350000    1.300000
#>75%        6.400000    3.300000     5.100000    1.800000
#>max        7.900000    4.400000     6.900000    2.500000

# distribución por clases
print(dataset.groupby('class').size())
#>Iris-setosa        50
#>Iris-versicolor    50
#>Iris-virginica     50
#>dtype: int64
```

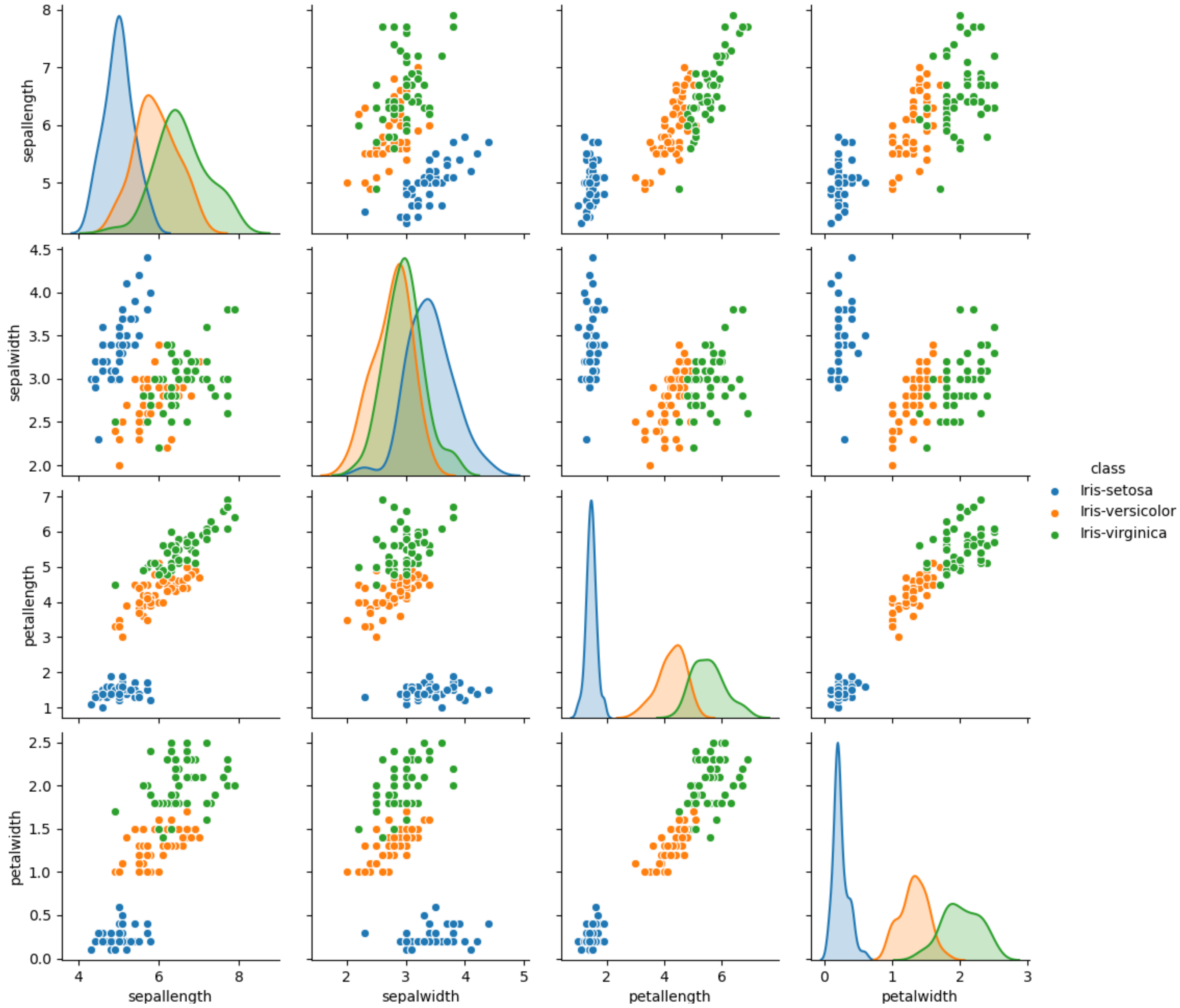
Ejemplos con Python: explorando los datos

```
# Carga de librerías
from pandas import read_csv
import matplotlib.pyplot as plt
import seaborn as sns

# Cargamos el dataset
url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.arff"
dataset = read_csv(url)

sns.pairplot( data=dataset, vars=("sepallength","sepalwidth","petallength","pe
talwidth"), hue="class" )
plt.show()
```

Ejemplos con Python: explorando los datos



Ejemplos con Python: comparativa de algoritmos

- **Para ello, separaremos nuestro dataset: usaremos un 80% de los datos para entrenar los algoritmos y un 20% de los datos para hacer los tests de predicción.**
  - **Ésta suele ser una proporción habitual.**

- Además, utilizaremos una **validación cruzada estratificada de 10 veces (*k-fold*)** para estimar la precisión del modelo.

- **Lo habitual es normalizar los datos**, especialmente cuando trabajamos con redes neuronales, pero en este ejemplo nos saltamos ese paso.

Ejemplos con Python: comparativa de algoritmos

```
# Load libraries
import numpy as np
import pandas as pd
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier

# inicializamos la semilla para generar números aleatorios
np.random.seed(0)

# Cargamos el dataset
url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.arff"
dataset = read_csv(url)
```

Ejemplos con Python: comparativa de algoritmos

```
# Dividimos el dataset en 80% de datos para entrenar y 20% para testear
array = dataset.values
X = array[:,0:4]
y = array[:,4]

X_train, X_validation, Y_train, Y_validation = train_test_split(X, y, test_s
ize=0.20, random_state=1, shuffle=True)

# Cargamos los algoritmos
models = []
models.append(("RF", RandomForestClassifier(n_jobs=2, random_state=0)))
models.append(("KNN", KNeighborsClassifier()))
models.append(("NB", GaussianNB()))
models.append(("SVM", SVC(gamma='auto')))
models.append(("MLP", MLPClassifier(activation="relu", alpha=1e-
05, batch_size="auto",beta_1=0.9, beta_2=0.999, early_stopping=False, epsilo
n=1e-
08, hidden_layer_sizes=(3, 3),learning_rate="constant", learning_rate_init=0
.001, max_iter=200, momentum=0.9, nesterovs_momentum=True, power_t=0.5, rand
om_state=1, shuffle=True, solver="lbfgs", tol=0.0001, validation_fraction=0.
1, verbose=False, warm_start=False)))
```

Ejemplos con Python: comparativa de algoritmos

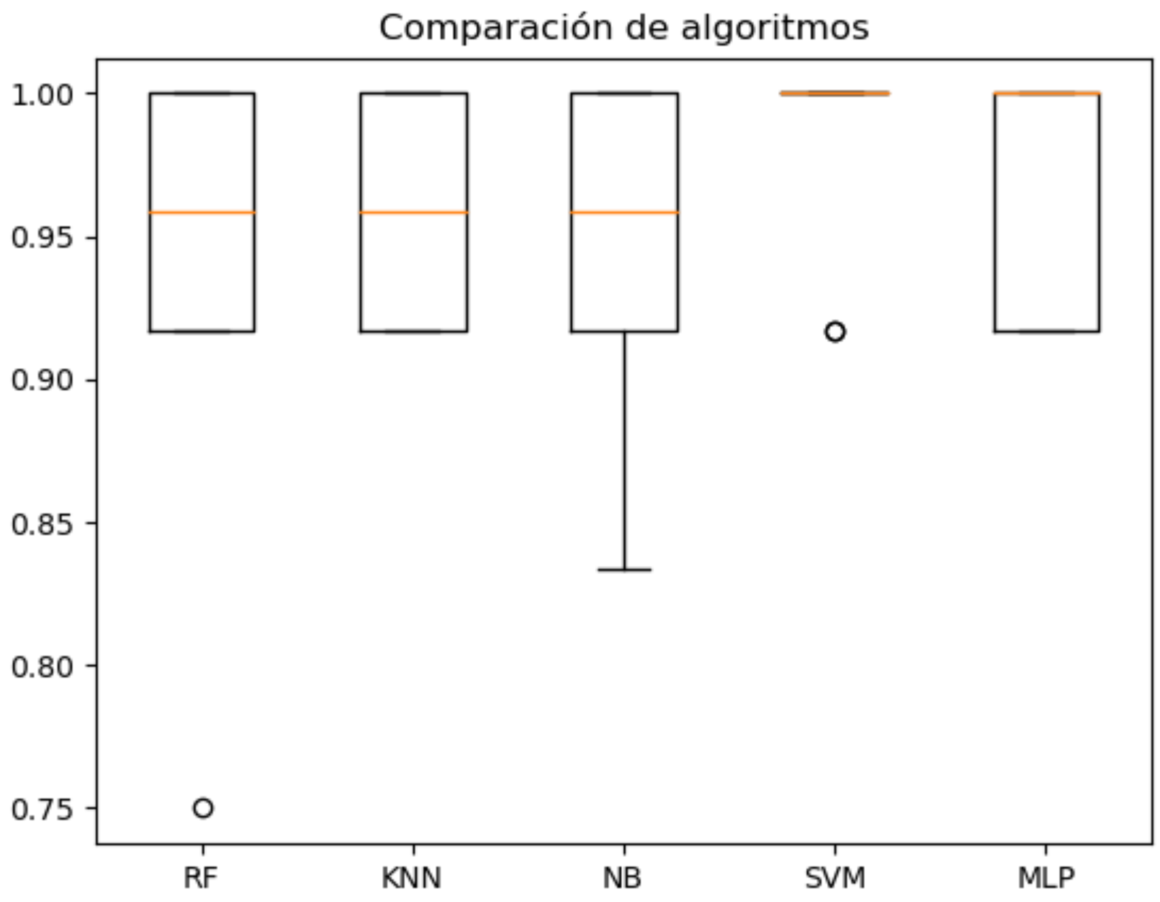
```
# evaluamos cada modelo por turnos
results = []
names = []

for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfold, scoring=
'accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))

# Comparación de algoritmos
pyplot.boxplot(results, labels=names)
pyplot.title('Comparación de algoritmos')
pyplot.show()

#>RF: 0.941667 (0.075000)
#>KNN: 0.958333 (0.041667)
#>NB: 0.950000 (0.055277)
#>SVM: 0.983333 (0.033333)
#>MLP: 0.966667 (0.040825)
```

Ejemplos con Python: comparativa de algoritmos



Resumen

Redes Neuronales Artificiales

Conexión entre neuronales artificiales
El objetivo del aprendizaje es determinar los pesos que determinan la relevancia de las conexiones

Redes Neuronales Prealimentadas (Feed forward)

- **Perceptrón** – Red neuronal más sencilla → Regla de aprendizaje basada en el error
- **Redes Neuronales Multicapa** – Redes unidireccionales de alimentación hacia adelante (Feed Forward)
  - → Método: retropropagación del error
  - → Regla de aprendizaje basada en el gradiente del error

Redes Neuronales Recurrentes

- **Red Hopfield** → Red monocapa auto-asociativa que memoriza información
- **Memoria bidireccional asociativa** – Bidirectional Associative Memory (BAM) → Red multicapa que memoriza y asocia informaciones

Gracias por vuestra atención
¿Dudas?

Imagen por Peggy und Marco Lachmann-Anke
Licencia: Creative Commons Zero

UNIVERSIDAD INTERNACIONAL DE LA RIOJA unir

www.unir.net