APRENDIZAJE SUPERVISADO
REDES NEURONALES

Redes Neuronales

- Las redes neuronales son una de las técnicas de aprendizaje automático más utilizadas hoy en día por su capacidad de aprender a partir de conocimiento no estructurado (imágenes, sonido, texto, …).
  - Es un caso paradigmático de enfoque de aprendizaje subsimbólico

- Inicialmente inspiradas en el modelo biológico de neuronas interconectadas. Hoy en día nos interesa más su estudio desde el punto de vista matemático.

- Una red neuronal es una función capaz de representar complejas relaciones no lineales entre los datos de entrada y la variable a predecir.

- Cada neurona es una unidad de proceso distribuido paralelo.
  - Desarrollo de hardware especializado: GPUs, tensor cores, …

Evolución histórica

- **Fase inicial (1943-1969)**
  - 1943: McCulloch y Pitts
  - 1949: Hebb, modelo de memoria dinámica
  - 1951 Minsky y Edmons construyen el primer ordenador que implementa una red neuronal
  - 1957-1962: Rosenblatt, propone el perceptrón con un algoritmo de aprendizaje
  - 1969: Minsky y Papert, limitaciones teóricas del perceptrón

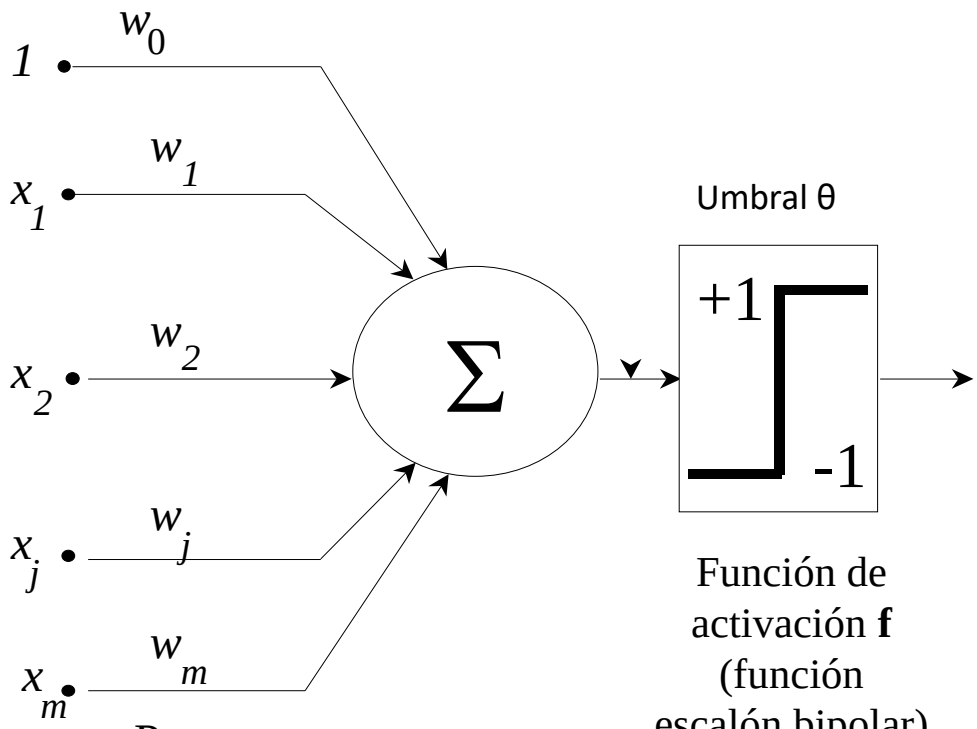
- **Resurgimiento (1975-1992)**
  - 1975: Werbos propone el algoritmo de backpropagation.
  - 1982: Hopfield, redes autoasociativas (reverberaciones estables para construir memorias)
  - 1984: Hinton, máquina de Boltzmann
  - 1986: Rumelhart, Hinton y Williams muestran experimentalmente que el aprendizaje por retropropagación genera representaciones internas en las capas ocultas del perceptrón
  - 1987: 1ª conferencia DARPA
  - 1992: max-pooling para ayudar a reconocer objetos 3D

Evolución histórica

- **Aprendizaje profundo (>2012)**
  - 2010: Ciresan, *backpropagation* usando GPUs en redes multicapa
  - 2012: Ng y Dean, aprendizaje de conceptos abstractos (ej: gatos) a partir de imágenes no etiquetadas
  - 2009-2012: acercamiento a nivel humano en varias tareas (reconocimiento de patrones en imágenes, reconocimiento de escritura, …)
  - 2013: DeepMind, aprendizaje por refuerzo profundo para jugar a juegos de Atari.
  - 2014: DeepFace, reconocimiento de caras de Facebook con rendimiento “humano”.
  - 2014: Redes Generativas Antagónicas (Ian Goodfellow *et* al.)
  - 2016: AlphaGo, capaz de ganar a jugadores de Go profesionales.

Perceptrón simple

x
Vector de entrada con *m* variables y una constante



$$\hat{y} = \begin{cases} 1 & \mathbf{w} \cdot \mathbf{x} > \theta \\ -1 & \mathbf{w} \cdot \mathbf{x} \leq \theta \end{cases}$$

Combinación lineal de las entradas

$$\mathbf{w} \cdot \mathbf{x} = \sum_{j=1}^{m} w_j x_j + w_0$$

Perceptrón simple

- Para cada ejemplo de entrenamiento (x_i,y_i) el perceptrón calcula

$$\hat{y}_i = f\left(w_0 + \sum_{j=1}^{m} w_j x_{i,j}\right)$$

- En función del valor real (y_i) y de la salida de la red ($\hat{y}_i$) se calcula el error para cada ejemplo de entrenamiento

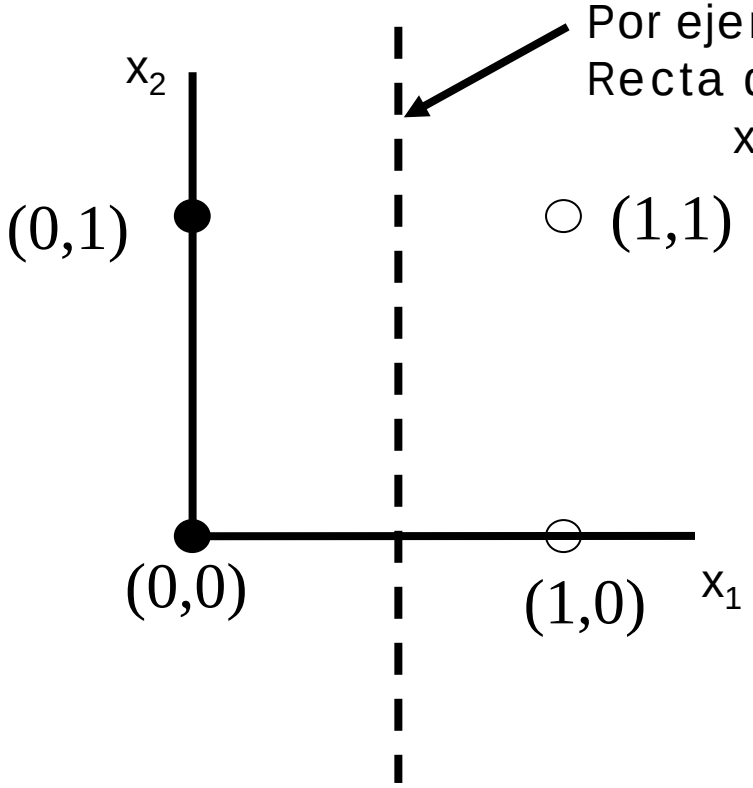
- **Entrenar la red significa encontrar los valores de los pesos w_j que minimicen el error total**
  - Búsqueda en el espacio de pesos

- Compromiso entre tiempo, error y nivel de generalización

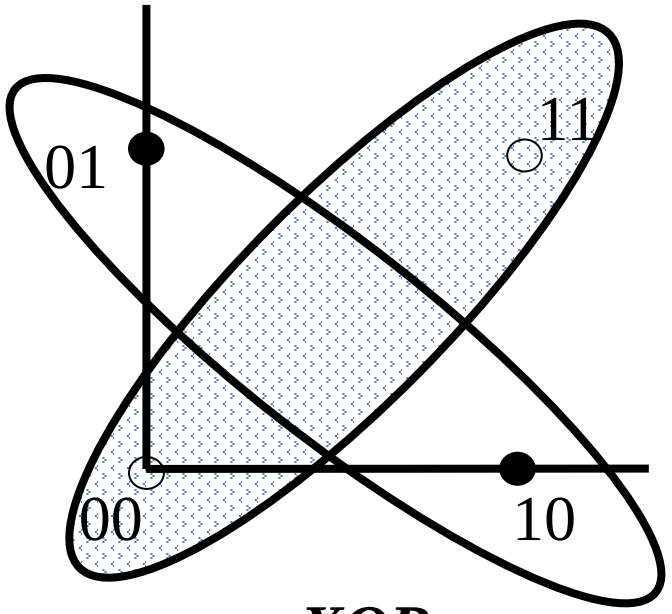
Limitaciones del Perceptrón

Función de decisión del ejemplo anterior:

Cualquier recta que separe las clases c1 y c2



...pero No existe recta que separe estas clases:



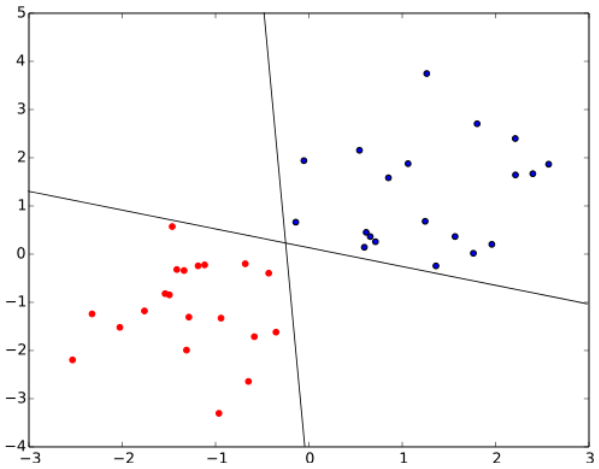
XOR

...necesitamos dos rectas para separar estas clases

... y cómo se consiguen?

Idea base del perceptrón multicapa

- El perceptrón simple es una combinación lineal de las variables de entrada, y por tanto permite "configurar" un hiperplano que separe dos clases
  - El hiperplano será eficaz solamente si las clases son linealmente separables, es decir, si una clase puede quedar a un lado del hiperplano y la otra al otro

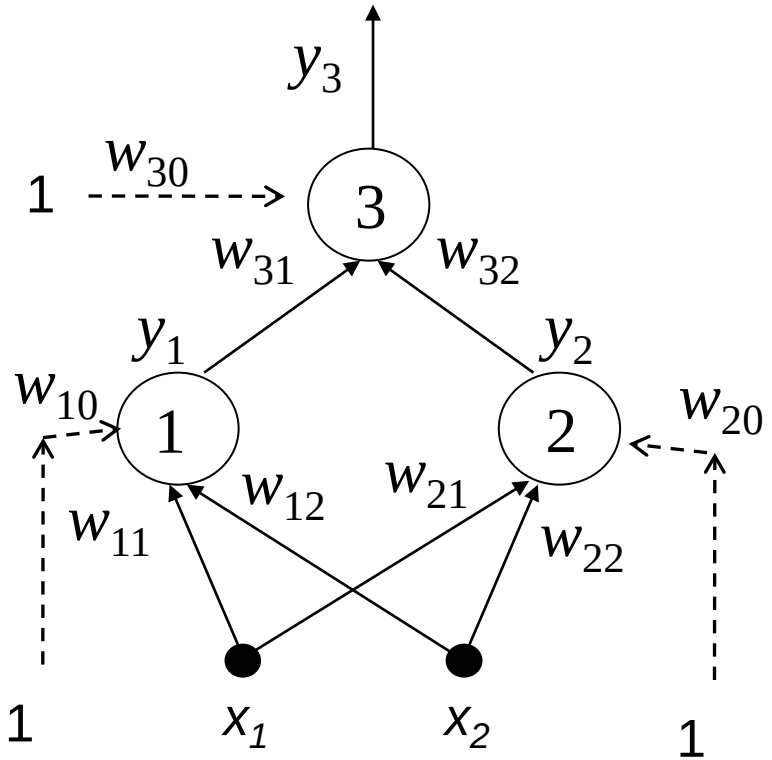


Dos de las posibles rectas que separan ambas clases (Imagen de Wikimedia.org)

- En el perceptrón multicapa la combinación de neuronas en diferentes capas, permite crear diferentes hiperplanos que al combinarse linealmente en la siguiente capa crean hipersuperficies no lineales que permiten separar las clases

Perceptrón Multicapa

Ejemplo: solución para XOR



Pesos

w_{10}=-0.5 w_{11}= 1 w_{12}= 1
w_{20}=-1.5 w_{21}= 1 w_{22}= 1
w_{30}=-0.5 w_{31}= 1 w_{32}= -1.5

Salidas de las neuronas

$y_1 = f(w_{11}x_1 + w_{12}x_2 + w_{10}) = f(x_1 + x_2 - 0.5)$
$y_2 = f(w_{21}x_1 + w_{22}x_2 + w_{20}) = f(x_1 + x_2 - 1.5)$
$y_3 = f(w_{31}y_1 + w_{32}y_2 + w_{30}) = f(y_1 - 1.5\, y_2 - 0.5)$

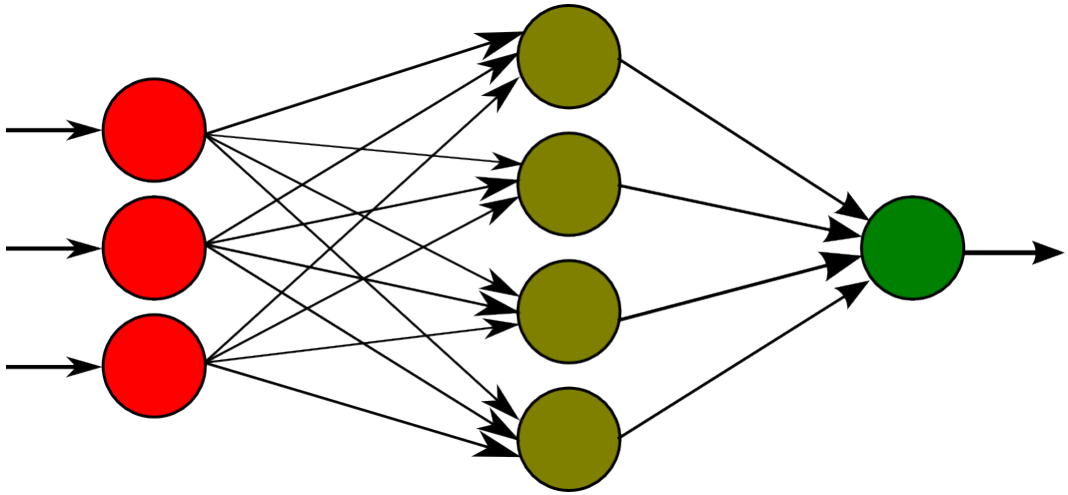
Resultados

x_1	x_2	y_1	y_2	y_3 = XOR
0	0	$f(-0.5) = 0$	$f(-1.5) = 0$	$f(-0.5) = 0$
0	1	$f(0.5) = 1$	$f(-0.5) = 0$	$f(0.5) = 1$
1	0	$f(0.5) = 1$	$f(-0.5) = 0$	$f(0.5) = 1$
1	1	$f(1.5) = 1$	$f(0.5) = 1$	$f(-1.0) = 0$

Función de activación: escalón unidad (f (x) = 1 si x > 0; f (x) = 0 si x <= 0)

Perceptron Multicapa (MLP)

- El perceptrón multicapa (*multi-layer perceptron* o *MLP*) es una red neuronal de tipo perceptrón con **al menos** tres capas
  - 1 **capa de entrada**, tantas neuronas como variables de entrada
  - 1 o más **capas ocultas** con un número variable de neuronas cada una de ellas con una función de activación
  - 1 **capa de salida** con una o varias neuronas con su función de activación
    - 1 neurona si es un problema de clasificación binario o de regresión simple, o tantas neuronas como clases en problemas de clasificación múltiple
- La capa intermedia hace que el MLP sea un "aproximador universal de funciones" capaz de modelar **relaciones no lineales** entre las variables de entrada y de salida.



Funciones de activación

- Las funciones de activación son funciones no lineales que permiten modelar funciones complejas “apilando” capas en la red neuronal.
  - Sin función de activación cada capa calcularía una combinación lineal de las salidas de la capa anterior. Y una combinación lineal de combinaciones lineales sigue siendo una combinación lineal ➔ no ganaríamos expresividad
  - Con las funciones de activación se calculan combinaciones lineales de funciones no lineales
- Las funciones de activación pueden ser de distintos tipos: logística, tangente hiperbólica, rectificada lineal (ReLU)
  - Todas ellas monótonas crecientes y derivables
    - Es necesario que sean derivables para poder entrenar la red (como veremos más adelante).
  - Las funciones de activación de la capa de salida dependen de los rangos de las variables que estamos modelando.
    - ¿La salida es una probabilidad? Logística
    - ¿La salida es un número positivo? ReLU
    - ¿La salida es un número entre -1 y 1? Tangente hiperbólica
    - ¿La salida es un número real? Identidad (sin función de activación)

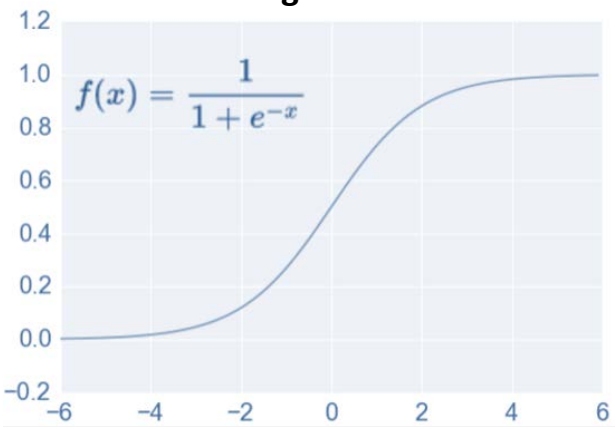
Tipos de función de activación

- **Logística**: Función sigmoidal que varía entre 0 y 1
  - Permite representar la probabilidad de pertenecer o no a la clase
  - La función sigmoidal es acotada, monótona creciente y diferenciable

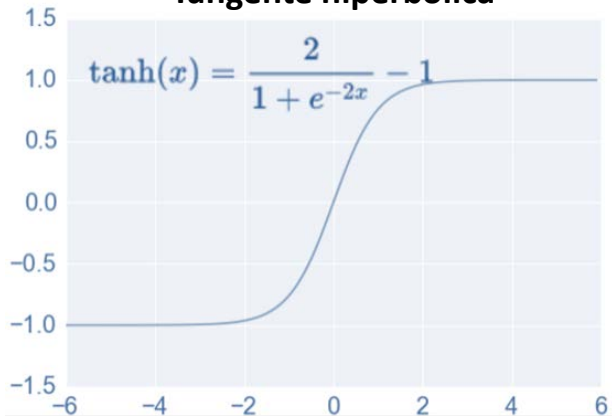
- **Tangente hiperbólica**: Función sigmoidal que varía entre -1 y 1
  - Permite asociar valores negativos a entradas negativas lo que la hace adecuada para separar entre dos clases

- **Identidad:** Útil en la capa de salida para problemas de regresión no acotados

Logística

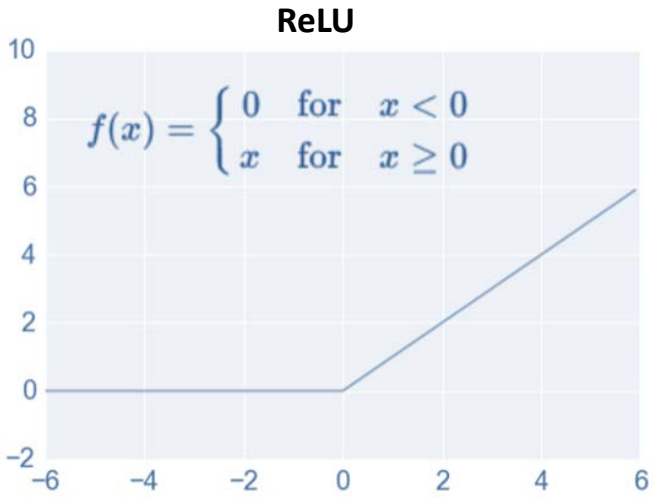


Tangente hiperbólica



Tipos de función de activación

- **Rectificada linear (ReLU):** Función con dos tramos, uno lineal que varía entre 0 e Infinito (muy usada en Deep learning)
  - Al ser esencialmente una función lineal reduce mucho el tiempo de cálculo
  - Al no estar acotada en un extremo no satura, lo cual es beneficioso en la fase de aprendizaje y para modelar problemas de regresión
  - A diferencia de las anteriores, puede tomar el valor cero (las otras se aproximan infinitamente), lo que inhibe la activación de la neurona y reduce los tiempos de cálculo en redes muy grandes



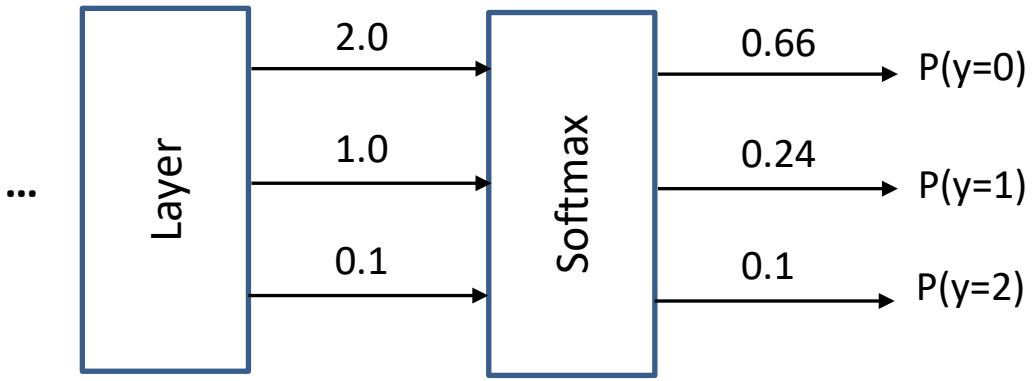
La función de activación de la capa de salida queda determinada por el rango de valores de la función que queremos modelar.

Las funciones de activación de las capas internas son un parámetro más de la Red Neuronal aunque actualmente ReLU es la opción más habitual.

Tipos de función de activación

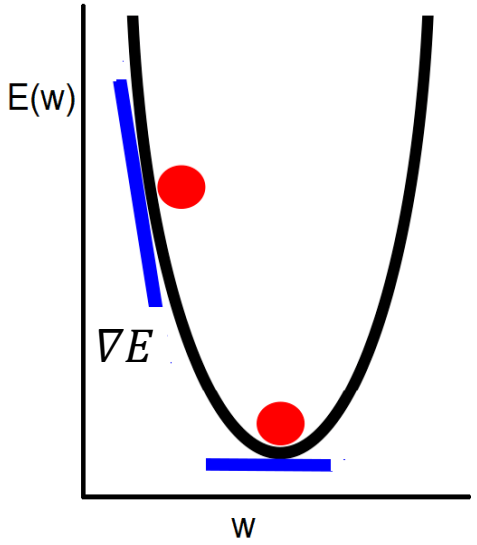
- **Softmax:** Transforma un conjunto de valores numéricos al rango [0,1] con suma total 1
  - Habitualmente se emplea en la capa de salida en problemas de clasificación multiclase (n clases ➔ n salidas)
  - Permite re-interpretar las salidas de la red como probabilidades (valores entre 0 y 1 que suman 1)
    - La probabilidad de que el ejemplo pertenezca a cada una de las n clases

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_k \exp(x_k)} \text{ para i} = 1, \ldots, n$$



Descenso por gradiente

- Dada una función de error E(w_1, .., w_s) diferenciable, podemos encontrar uno de sus mínimos (locales) mediante el algoritmo de descenso de gradiente
  - Partimos de un punto aleatorio *p* con un cierto error *E(p)*
  - Calculamos el gradiente de la función $\nabla E = (\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_s})$ en el punto p
    - El vector gradiente indica la dirección en la que la función de error crece más rápido
  - Calculamos un nuevo punto p = p $-\alpha \nabla E$ y volvemos al paso anterior.
    - α indica cuanto nos desplazamos y se conoce como tasa de aprendizaje (*learning rate*)



En cada paso del algoritmo disminuimos el error hasta llegar a un mínimo local donde el gradiente será 0.

Si la función no es convexa no se garantiza encontrar un mínimo global.

Algoritmo de retropropagación

- Idea: disminuir el error de la red mediante descenso de gradiente
- Se inicializan los pesos de forma aleatoria y con valores pequeños
- Para cada ejemplo *k* del conjunto de entrenamiento se hacen dos fases:
  - Hacia delante
    - Calculamos la salida de la red para el ejemplo $\hat{y}_k$
    - Calculamos el error (*loss value*) entre la salida real de la red y la salida deseada para ese ejemplo $E_k(y_k, \hat{y}_k)$
  - Hacia atrás
    - Calculamos el gradiente de la función de error con respecto a los pesos de la red (mide cuánto afecta un pequeño cambio en cada peso al error) para el ejemplo *k*

$$\nabla E_k = \left(\frac{\partial E_k}{\partial w_{10}}, \ldots, \frac{\partial E_k}{\partial w_{rs}}\right)$$

      - Se calcula usando la regla de la cadena desde la capa final de la red hacia atrás
    - Ajustamos los pesos para que reduzcan el error respecto al ejemplo k

$$w' = w - \alpha \nabla E_k$$

Algoritmo de retropropagación

- De esa forma reducimos el error con respecto al ejemplo *k*, pero en realidad nos interesa reducir el error total con respecto a todos los ejemplos

$$E_{total} = \sum E_k$$

- Así que en realidad calculamos el gradiente medio y modificamos los pesos de la red en dirección contraria

$$\nabla E_{total} = \frac{1}{n} \sum \nabla E_k$$

$$w' = w - \alpha \nabla E_{total}$$

- La actualización de la red para todo el conjunto de entrenamiento recibe el nombre de **época** (*epoch*)
  - El entrenamiento de una red puede requerir entre decenas y miles de épocas

Cálculo del error

- Es importante utilizar una función de error que sea derivable para entrenar la red. A esta función se le llama **función de pérdida** (loss).

- Error en problemas de regresión: error cuadrático

$$MSE = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2$$

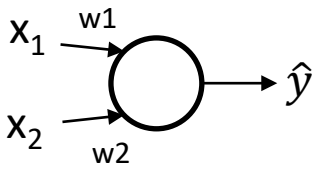
- Error en problemas de clasificación: entropía cruzada. Por ejemplo, para dos clases:

$$CE = \frac{1}{n}\sum_{i}^{n} y_i \log(\hat{y}_i) - (1 - y_i)\log(1 - \hat{y}_i)$$

- El error de la red se calcula como la suma de los errores en cada uno de los ejemplos de entrada.

Ejemplo

Red neuronal



Función de activación: sigmoide

Función de error: $E = \frac{1}{2}(y - \hat{y})^2$

Cálculo del gradiente

$$z = w_0 + x_1 w_1 + x_2 w_2 \qquad \hat{y} = \frac{1}{1 + e^{-z}} \qquad \nabla E = (\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2})$$

Sólo voy a calcular una de las componentes del gradiente a modo de ejemplo usando la regla de la cadena:

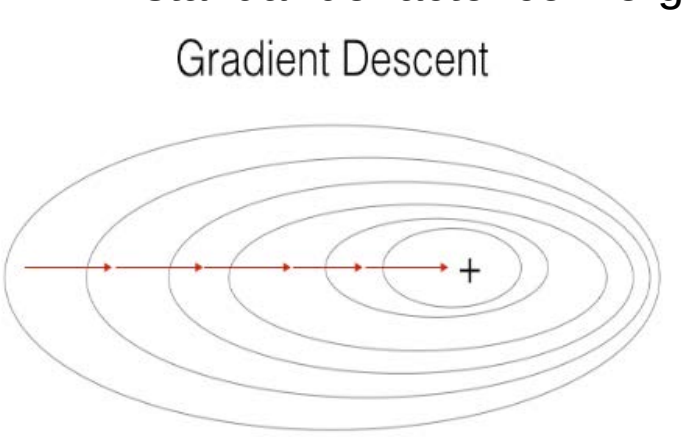
$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_1} \qquad \frac{\partial E}{\partial \hat{y}} = -(y - \hat{y}) \qquad \frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y}) \qquad \frac{\partial z}{\partial w_1} = x_1$$

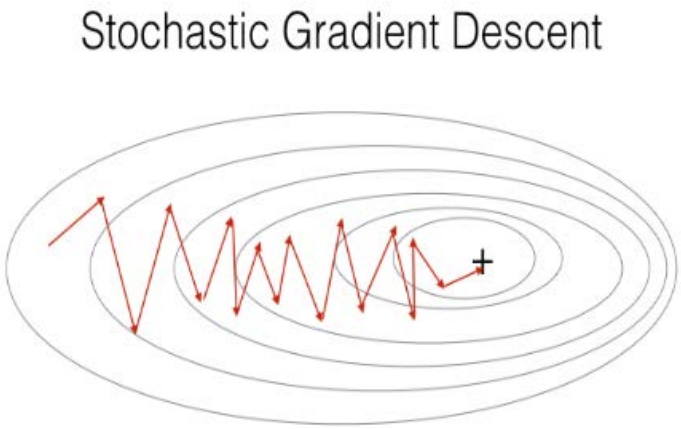
Una vez tengo el gradiente puedo evaluarlo en cualquier punto y me dará un vector de 3 componentes. Calculo el gradiente medio para todos los puntos del conjunto de entrenamiento y actualizo los pesos:

$$w'_0 = w_0 - \alpha \frac{\partial E_{total}}{\partial w_0} \qquad w'_1 = w_1 - \alpha \frac{\partial E_{total}}{\partial w_1} \qquad w'_2 = w_2 - \alpha \frac{\partial E_{total}}{\partial w_2}$$

Minibatches

- El gradiente nos indica la dirección que mejor minimiza el error pero para calcularlo es necesario recorrer todos los ejemplos de entrenamiento en cada paso
  - Esto es muy costoso computacionalmente
- Una alternativa sería calcular el gradiente a cada ejemplo (GD estocástico)
  - La convergencia fluctúa mucho
- Lo que se hace es dividir los ejemplos de entrenamiento en conjuntos pequeños (32, 64, …) y dar muchos más pasos en cada *epoch*.
  - Búsqueda más eficiente pero con mucho más ruido
  - Estándar de facto: converge mucho antes





Regularización

- Para evitar el sobreaprendizaje podemos utilizar varias estrategias
  - Conseguir más ejemplos de entrenamiento
  - Interrumpir antes el proceso de entrenamiento
  - Utilizar técnicas de regularización

- Regularización L2

$$Error = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2 + \frac{\lambda}{2n}\sum_{i}^{n}\|w\|^2$$

  - La idea es penalizar los pesos demasiado grandes para que la red generalice mejor.
  - λ es la tasa de regularización (permite controlar cuánto regularizar)
  - Otras técnicas usadas: regularización L1 y *dropout* (eliminación probabilística de nodos).

Consideraciones de uso del percetrón multicapa

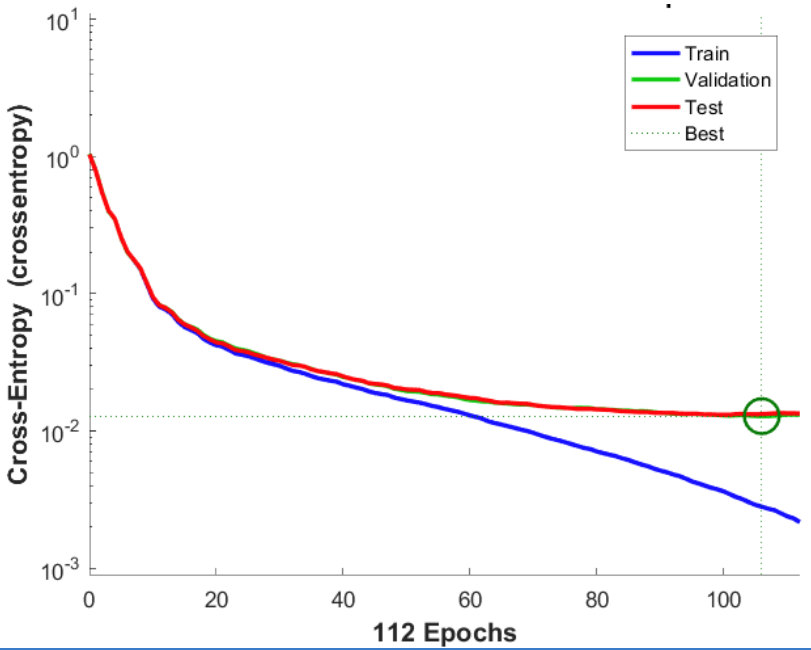
- Los datos de entrada deben ser numéricos
  - Las variables categóricas binarias se pueden adaptar con 0 y 1
  - Las variables categóricas múltiples como un vector binario de tantos elementos como clases
- El MLP puede tratar con variables sin normalizar o sin estandarizar
  - Sin embargo, la normalización o estandarización de las variables disminuye el tiempo de entrenamiento y evita los “mínimos locales” en la optimización del error
- En principio el MLP es capaz de tratar con variables irrelevantes, ya que les acabará asignando peso cero
  - La arquitectura del perceptrón no será necesariamente más compleja (no requerirá de más neuronas en la capa oculta)
  - Sin embargo, añadir variables irrelevantes hace que sean necesarios más datos de entrenamiento

Configurando el perceptrón multicapa

- La arquitectura del MLP la determina el número de capas (a la hora de contarlas se suele ignorar la de entrada) y el número de neuronas de cada capa
  - En principio, un MLP con una única capa oculta y “suficientes” neuronas, puede resolver igual un problema que un MLP con más capas ocultas
  - Las capas ocultas añaden “representaciones” de los datos con un nivel de abstracción mayor y en principio es más fácil llegar a la solución a partir de ellas
    - En esto se basa grosso modo el deep learning que usa gran cantidad de capas intermedias
  - Determinar el número de capas y de neuronas puede hacerse probando varias configuraciones y determinando la mejor mediante validación cruzada

Configurando el perceptrón multicapa

- El MLP cuenta también con parámetros (tasa de aprendizaje, regularización) que controlan el sobreaprendizaje del modelo
  - La validación cruzada también nos ayuda a ajustar el modelo del MLP y no sobreaprender los datos de entrenamiento
  - De hecho, también hay estrategias de regularización que controlan el error en validación en paralelo al entrenamiento y detienen este último si el error de validación aumenta



Parámetros del perceptrón multicapa

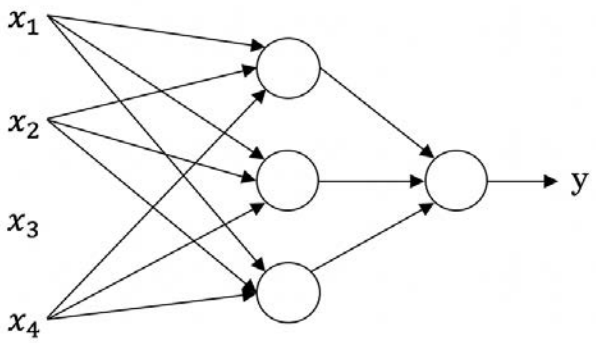
- Número de capas y número de neuronas por capa
  - Intuición: necesitaremos una red más compleja cuando más compleja sea la función que tratamos de aproximar
- Funciones de activación de las capas intermedias
  - Las de la capa de salida suelen quedar determinada por el tipo de problema
- Tasa de aprendizaje
  - Cuanto más pequeña mejores resultados puede encontrar pero más tiempo de entrenamiento
  - Hay estrategias para modificarla dinámicamente
- Tipo de regularización y tasa de regularización
  - Regularización L1 o L2, dropout, etc.
- Criterio para dejar de entrenar
  - Evitar sobreaprendizaje vs encontrar buenos mínimos del error
- ¡Ojo! a veces el problema no está en la red sino en los datos

Elegir los parámetros de la red

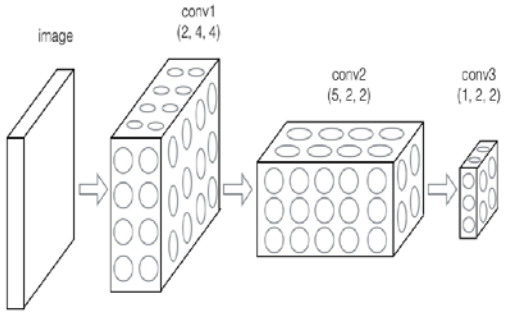
- Cualquier estrategia de validación cruzada sirve, por ejemplo:
  - Dividir el conjunto de datos en 3 trozos: entrenamiento, validación y test.
  - Elegir una configuración de parámetros, entrenar la red con el conjunto de entrenamiento y evaluar su funcionamiento con el conjunto de validación.
  - Una vez he encontrado una buena configuración evaluamos su funcionamiento con el conjunto de test.
  - **Lo importante es no usar el mismo subconjunto de datos para entrenar, configurar y evaluar.**
- Podemos simplificar la búsqueda de configuraciones de parámetros buscando el valor de cada uno de forma secuencial
  - Optimizar uno de ellos, dejarlo fijo y optimizar el siguiente
  - Estrategia voraz: mucho más rápido pero no es óptimo
- Cada entrenamiento puede tardar minutos, horas, días…

Otros tipos de redes neuronales

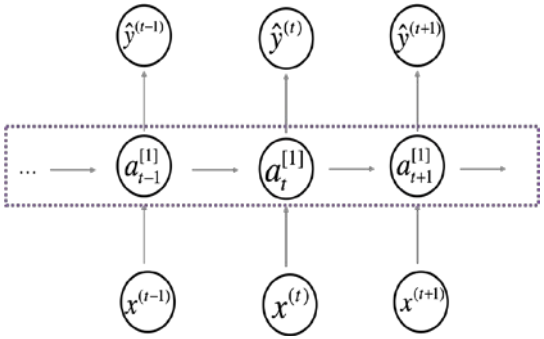
- **Perceptrón multicapa**: rede neuronal básica
- **Redes convolucionales**: tratamiento de imágenes
- **Redes recurrentes**: tratamiento de secuencias y series temporales (Hopfield, Boltzmann, LSTM)
- **Arquitecturas híbridas** y ad-hoc: topologías mixtas creadas a medida para un dominio concreto.
- **Mapas auto-organizados**: red neuronal para aprendizaje no supervisado



Perceptrón multicapa



Convolutional NN



Recurrent NN

MÁS SOBRE APRENDIZAJE AUTOMÁTICO

Aplicaciones de aprendizaje automático

- Procesado de voz
- Procesado de imágenes
  - Clasificación de imágenes
  - Análisis de pruebas médicas
  - Monitorización y control
- Sensorización, control y robótica
- Diagnóstico
  - Diagnóstico médico
  - Averías
  - Morosidad
- Segmentación de clientes y estudios de mercado
- Control y robótica
- Bioinformática
- Finanzas
- Etc

Otras técnicas de aprendizaje automático

- Las diferentes técnicas de aprendizaje automático pueden ayudar a solucionar un problema y la elección de una o de otra depende de las características del problema que tengamos
  - Problema de clasificación, de regresión, de detección de relaciones entre variables o de ocurrencia de sucesos, detección de anomalías o de fraudes
  - Volumen de datos (número de ejemplos)
  - Calidad de los datos, p.ej. valores perdidos y escasos (sparsity)
  - Cantidad de las variables y calidad o relevancia de las mismas
  - Etc.

- Además, la complejidad de los datos en situaciones reales hace que la pericia de la persona al frente del problema sea determinante
  - Entendiendo el problema y cómo los datos pueden resolverlo
  - Limpiando y transformando los datos,
  - Seleccionando las variables relevantes,
  - Interpretando los resultados
  - Ayudando a traducir los resultados en decisiones útiles

Otras técnicas de aprendizaje automático

- Hay una gran cantidad de técnicas de aprendizaje automático y cada técnica puede tener su punto fuerte sobre otras. Hay muchas más de las que hemos visto:
  - Máquinas de Vectores de Soporte (*Support Vector Machines*)
  - Reglas asociativas
  - Naïve Bayes
  - …
- Hoy día en problemas de aprendizaje supervisado suelen obtener muy buenos resultados las técnicas que agregan muchos "predictores" sencillos (*ensemble learning*)
  - En lugar de usar un único predictor que aprenda "todo" el conjunto de datos, se combinan múltiples predictores sencillos que se "especializan" en partes del conjunto de datos.
  - Las dos estrategias más usadas para combiner predictors son:
    - Votación (*bagging*): por ejemplo, los random forests son multiples árboles de decision sencillos donde cada uno clasifica (vota) la clase a la que pertenece el ejemplo nuevo y la clase que se le asigna es la que gane la votación por mayoría
    - *Boosting*: Iterativamente se concatenan predictores de forma que cada uno de ellos intenta "corregir" los errores que cometió el predictor anterior, es decir, aprender a predecir lo que el anterior erró

Retos y nuevas tendencias del aprendizaje automático

- **Data Science**
  - La ciencia de datos es un área que combina la estadística, la minería de datos, el aprendizaje automático, la visualización de datos y la recuperación y el procesado de datos de fuentes diversas (bases de datos, sensores, ficheros, etc)
  - Surge para hacer frente a los desafíos del almacenamiento masivo de datos y la necesidad de convertir dichos datos en información y en soporte a la toma de decisiones

- **Big Data**
  - Aunque el término se usa a menudo inadecuadamente, se refiere a tres “V” :
    - Variedad: los tipos de datos son diversos (imágenes, sensores, texto, etc.)
    - Velocidad: supone un reto tratar los datos en tiempo real y en algún caso se descarta su almacenamiento
    - Volumen: los datos que se manejan tienen un volumen tal que no cabe en la memoria de un ordenador
  - A menudo se añaden Variabilidad (cambiantes) y Veracidad (confiables)
  - Las tecnologías big data son aquellas que procesan los datos en la nube en paralelo y que reformulan algoritmos de aprendizaje estadístico en paralelo o en tiempo real

Retos y nuevas tendencias del aprendizaje automático

- **Deep Learning**
  - El deep learning es un paradigma de aprendizaje fundamentalmente basado en redes neuronales
  - Se caracteriza por la existencia de numerosas capas intermedias de forma que en la red los objetos se expresan como una composición de primitivas de sofisticación creciente
  - La capacidad de aprendizaje es muy elevada y se obtienen resultados muy buenos en visión artificial, reconocimiento del habla, procesamiento de lenguaje natural, etc.

Retos y nuevas tendencias del aprendizaje automático

- **Explainable Artificial Intelligence**
  - Depender de las decisiones de un sistema de IA que no sabemos cómo funciona puede generar frustración o desconfianza
    - Especialmente en dominios críticos: sistemas médicos, coches autónomos
  - En el aprendizaje automático abundan las técnicas *opacas* como las redes neuronales (deep learning) que cada vez se implantan más
  - La IA Explicable busca que los modelos expliquen por qué toman una decisión y no otra, así como sus fortalezas y limitaciones

Referencias

- **Witten, I.H., Frank, E., Hall, M.A.**
  Data Mining
  Elsevier, 2017 (versión electrónica en la Biblioteca UCM)

- **Chollet, F.**
  Deep Learning with Python
  Manning, 2017 (https://www.manning.com/books/deep-learning-with-python)

- **James, G. et al.**
  An Introduction to Statistical Learning with Applications in R
  Springer, 2013 (http://www-bcf.usc.edu/~gareth/ISL/)

- **Borrajo, D., González, J., Isasi, P.**
  Aprendizaje automático
  Sanz y Torres, 2006

- **Sierra, B.**
  Aprendizaje automático
  Pearson Prentice-Hall, 2006

- **Hilera, J.R., Martínez, V. J.**
  Redes Neuronales Artificiales
  RA-MA, 1995