Aprendizaje Automatizado

Redes Neuronales Artificiales

Introducción

- Una forma de emular características propias de los humanos: memorizar y asociar hechos.
- Se aprende de la **experiencia**.
- El cerebro humano es el ejemplo más perfecto de sistema capaz de adquirir conocimiento.
- Se modela artificialmente ese sistema.

MODELO NEURONAL
ARQUITECTURA DE REDES

Definición

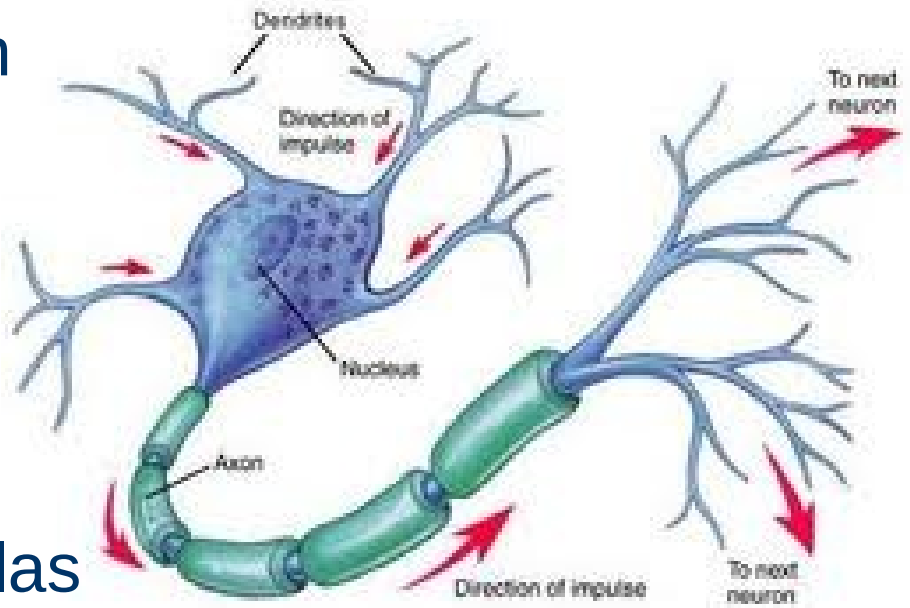
- Una Red Neuronal Artificial (RNA) es un sistema para el tratamiento de la información, cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano: **la neurona**.
- Las neuronas son un componente relativamente simple pero conectadas de a miles forman un poderoso sistema.

Descripción de las RNA

- Unidades de procesamiento que intercambian datos o información.
- Se utilizan para reconocer patrones, incluyendo imágenes, manuscritos, tendencias financieras, etc.
- Tienen la capacidad de aprender y mejorar su funcionamiento.

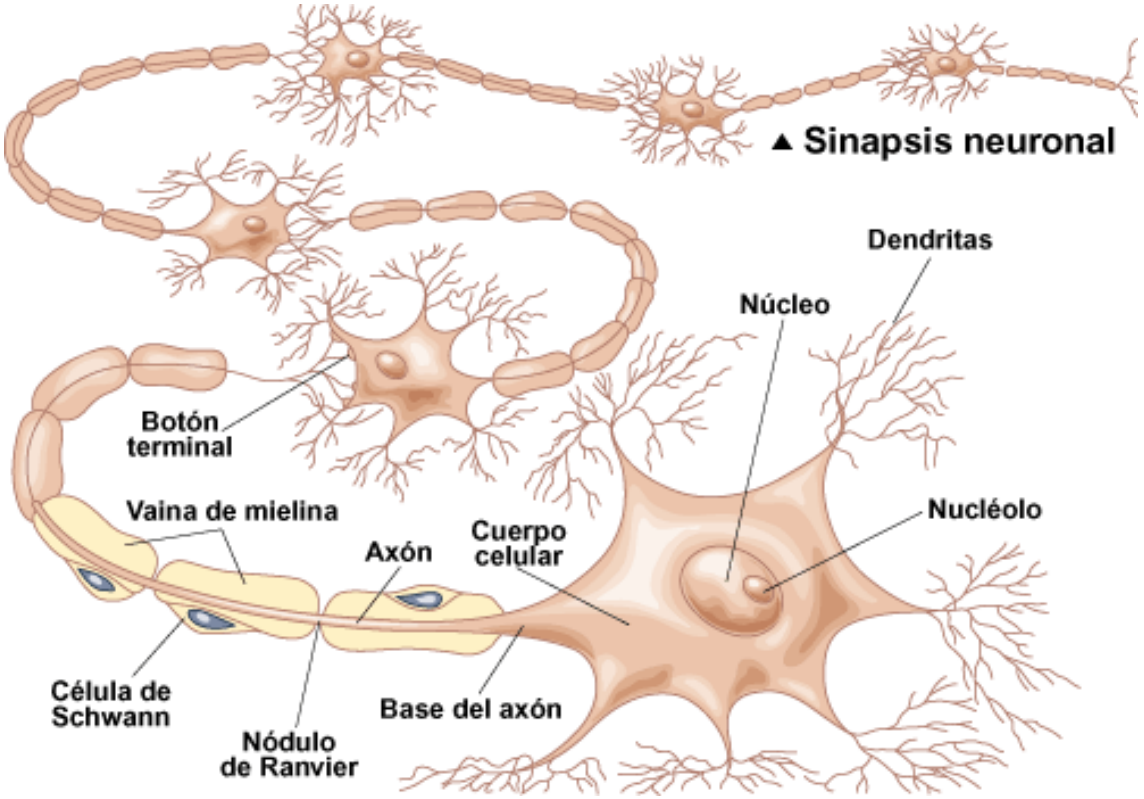
Fundamentos - El modelo biológico

- El cerebro humano contiene más de cien mil millones de neuronas.
- La clave para el procesamiento de la información son las conecciones entre ellas llamadas **sinápsis**.



Estructura biológica

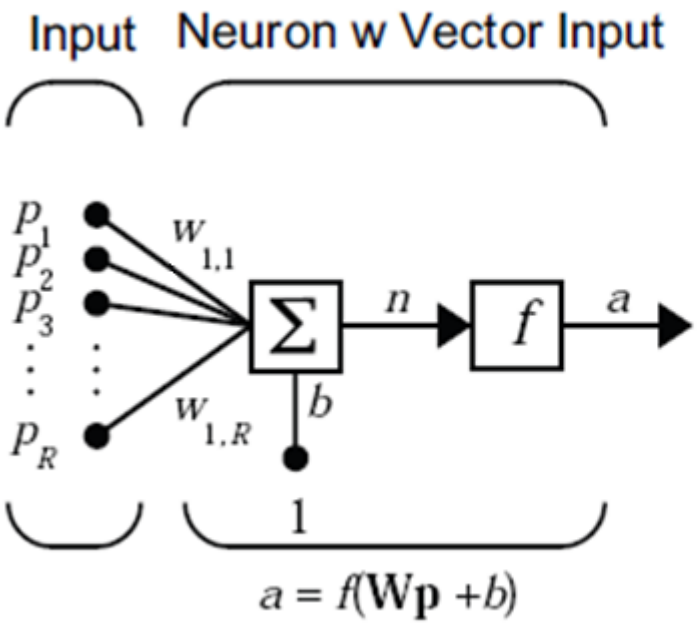
- Las **dendritas** son la vía de entrada de las señales que se combinan en el cuerpo de la neurona.
- El **axón** es el camino de salida de la señal generada por la neurona.



Estructura biológica

- En las terminaciones de las **sinápsis** se encuentran unas vesículas que contienen unas sustancias químicas llamadas neurotransmisores, que propagan señales electroquímicas de una neurona a otra.
- La neurona es estimulada por sus entradas y cuando alcanza cierto umbral, se dispara o activa pasando una señal hacia el axón.

Neurona artificial

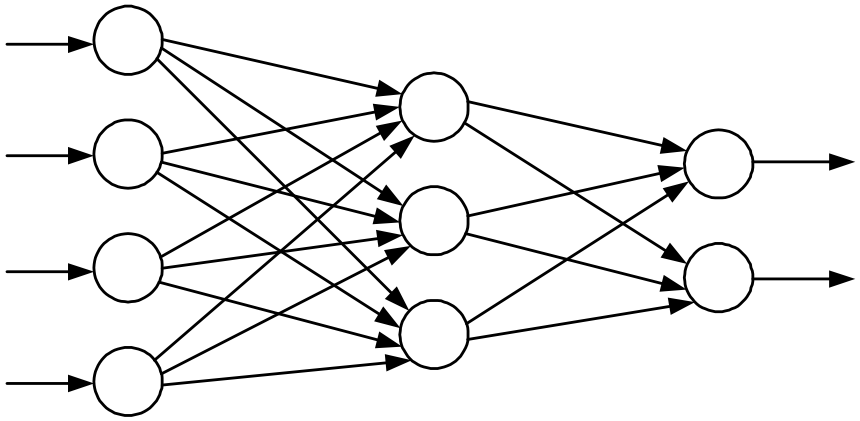


Where...

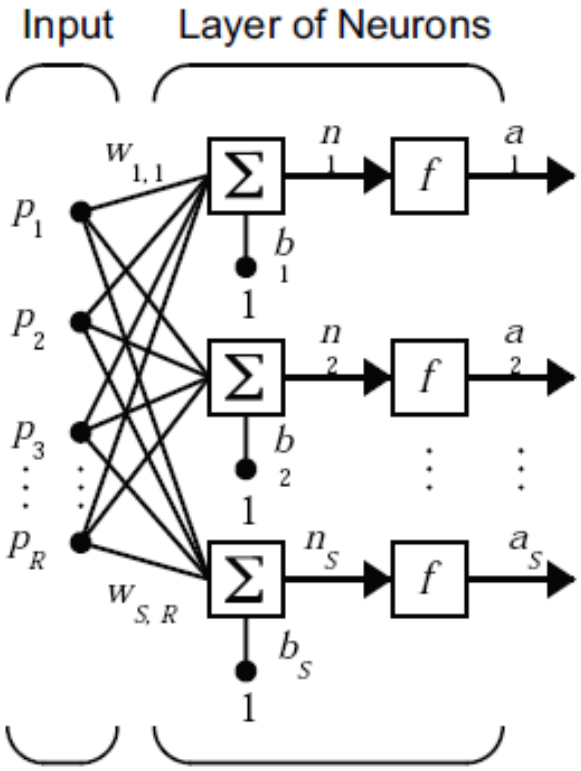
R = number of elements in input vector

Estructura Artificial

- Se interconectan neuronas en tres tipos de capas:
  - De entrada: reciben estímulos externos.
  - Oculta: elementos internos de procesamiento (se pueden estructurar en varias capas).
  - De salida: reciben la información procesada y retornan la respuesta del sistema al exterior.



Una capa de neuronas



Where...

R = number of elements in input vector

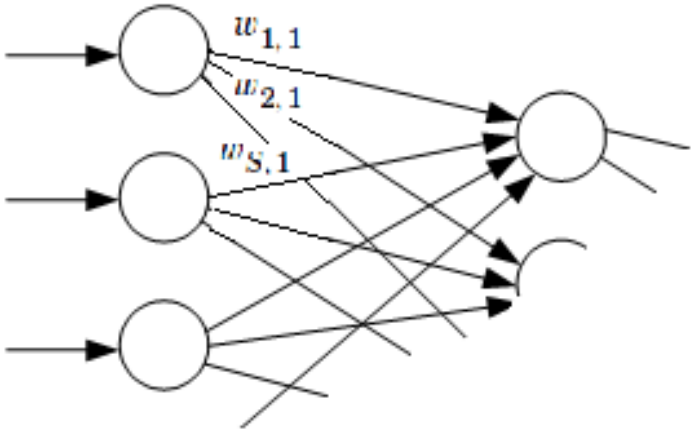
S = number of neurons in layer

Una capa de neuronas

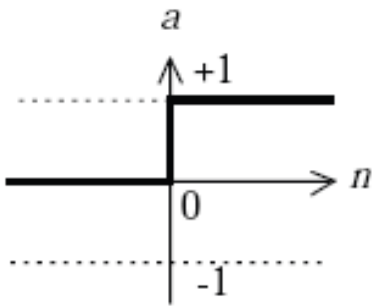
$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ & & & \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

La fila 1 de la matriz de pesos, contiene los pesos de las conexiones que van hacia la primer neurona.

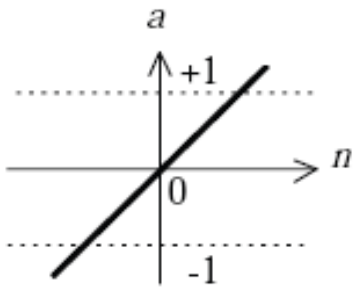
La columna 1 de la matriz de pesos, contiene los pesos de las conexiones que salen desde la primer entrada.



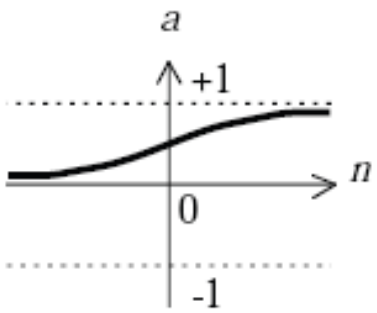
Funciones transferencia



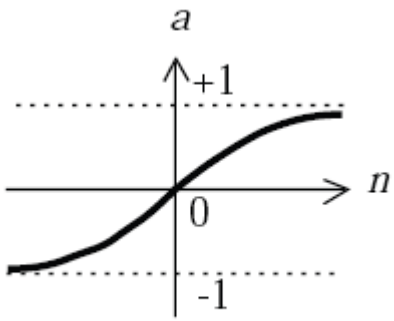
$a = hardlim(n)$



$a = purelin(n)$



$a = logsig(n)$



$a = tansig(n)$

Características de las RNA

- Arquitectura (Topología).
  - Número de capas.
  - Número de neuronas por capa.
  - Tipo de conexiones. Normalmente, todas las neuronas de una capa reciben señales de la capa anterior (más cercana a la entrada) y envían su salida a las neuronas de la capa posterior (más cercana a la salida de la red).
- Tipo de aprendizaje.

Work flow

El diseño general del trabajo con RNA tiene los siguientes pasos básicos:

- Collect data
- Create the network
- Configure the network
- Initialize the weights and biases
- Train the network
- Validate the network (post-training analysis)
- Use the network

Redes *feedforward*

- Las neuronas de una capa se conectan con las neuronas de la capa siguiente (hacia adelante).
- Las más conocidas son:
  - Perceptrón
  - Adaline
  - Backpropagation
- Son útiles en aplicaciones de reconocimiento o clasificación de patrones.

Redes *feedforward-Multiple-layer*

Multiple-layer networks are quite powerful!!!

- For instance, a network of two layers, where the first layer is sigmoid and the second layer is linear, can be trained to approximate any function (with a finite number of discontinuities) arbitrarily well.
- This kind of two-layer network is used extensively

Mecanismo de Aprendizaje

- Biológicamente se acepta que la información memorizada en el cerebro se relaciona con los valores sinápticos de las conexiones.
- En las RNA se considera que el conocimiento se encuentra representado en los pesos de las conexiones.
- El proceso de aprendizaje se basa en cambios en estos pesos.

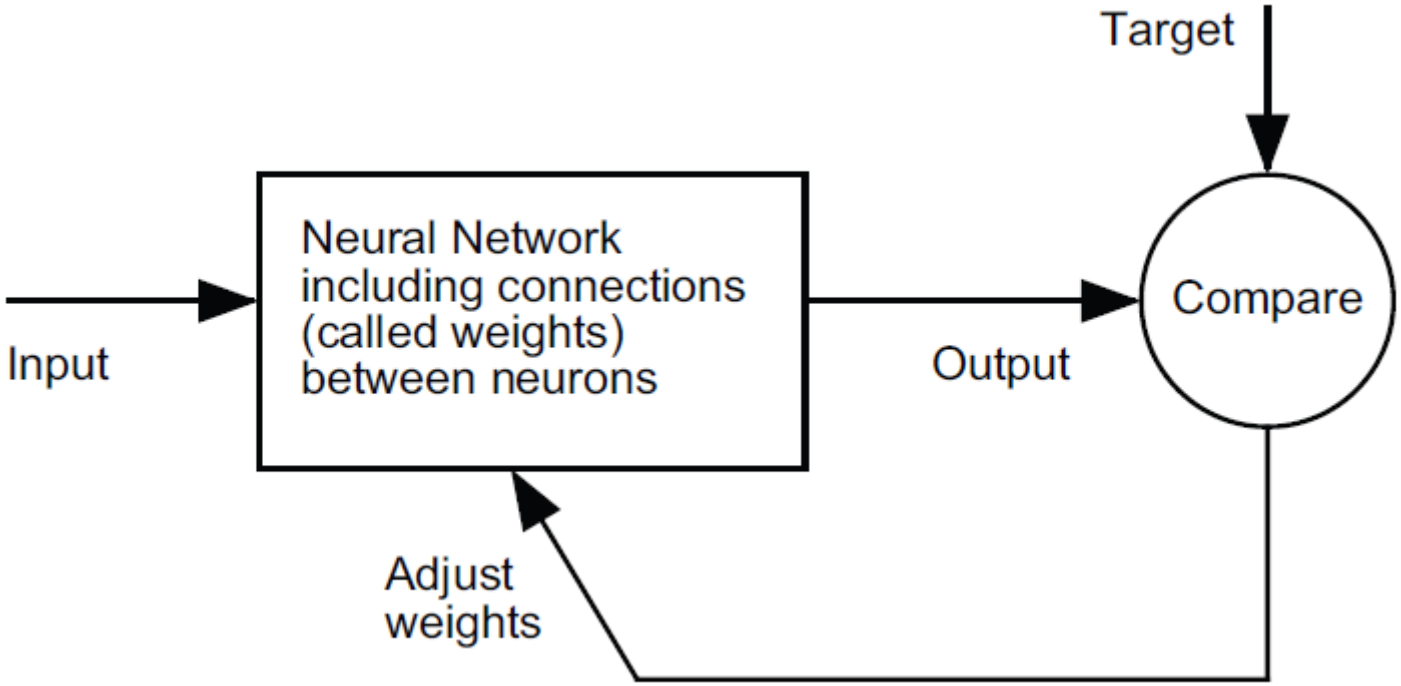
Mecanismo de Aprendizaje

- Los cambios en el proceso de aprendizaje se reducen a destrucción, modificación y creación de conexiones entre las neuronas.
- La creación de una conexión implica que el peso de la misma pasa a tener un valor distinto de cero.
- Una conexión se destruye cuando su valor pasa a ser cero.

Reglas de Aprendizaje

- Una regla de aprendizaje es un procedimiento para modificar los pesos y umbrales de una red.
- Este proceso también puede ser llamado *algoritmo de entrenamiento*.
- El aprendizaje puede ser *supervisado* o *no supervisado*.

Entrenamiento



PERCEPTRÓN

EL MODELO
NEURAL NETWORK TOOLBOX - MATLAB

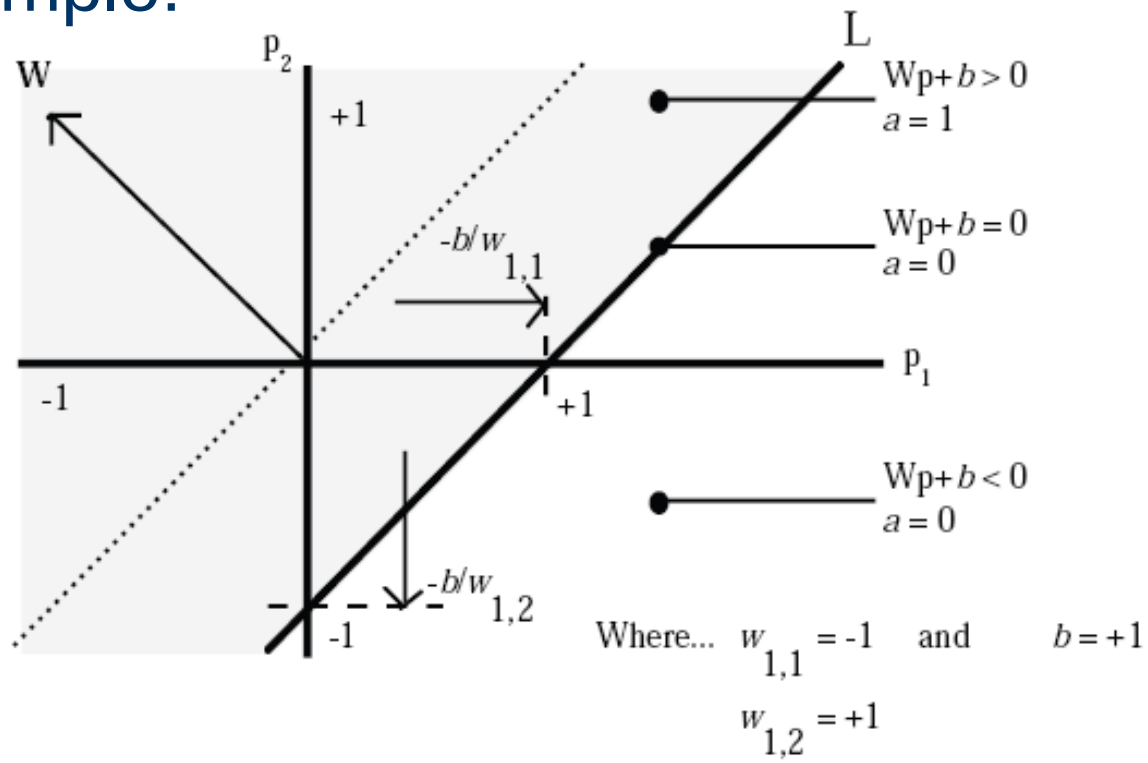
La neurona perceptrón

- Utiliza la función transferencia escalón (`hardlim`).
- Produce un 1 si la entrada a la función es mayor o igual que 0 y produce un 0 en caso contrario.
- Esto permite clasificar vectores de entrada dividiendo el espacio de entrada en dos regiones.

nnd4db

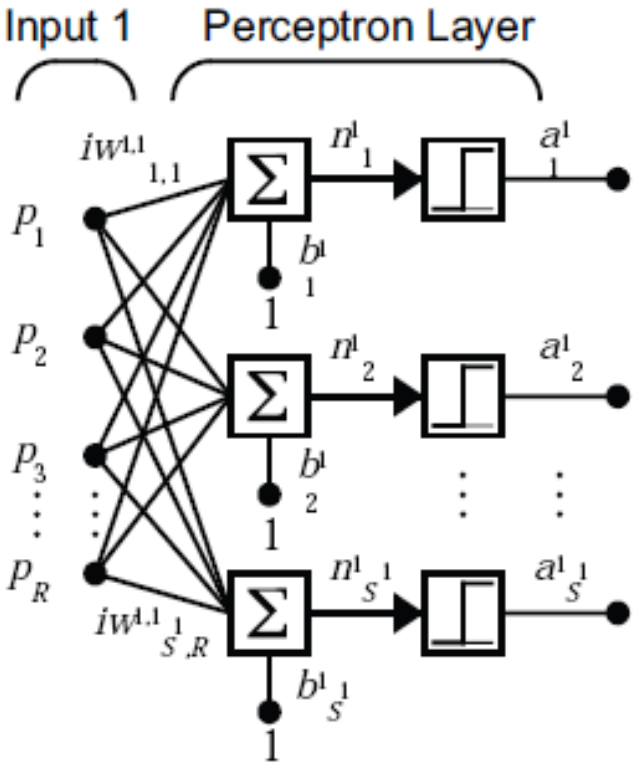
La neurona perceptrón

- Ejemplo:



Arquitectura

- La red Perceptrón consiste de una única capa oculta de S neuronas perceptrón conectadas a R entradas a través de conexiones con pesos $w_{i,j}$.
- Los índices `i` y `j` indican que $w_{i,j}$ es el peso de la conexión desde la j-ésima entrada hacia la i-ésima neurona.



Creación de un perceptrón con Matlab

```
net = perceptron
net = configure(net, P, T);

P: vector de entrada de tamaño n x m,
n: cantidad de atributos de la entrada
m: cantidad de entradas
T: clasificación de las entradas
   (tamaño 1 x m)
```

Creación de un perceptrón con Matlab (Antes)

```
net = newp(PR, S)
```

PR: matriz de tamaño R * 2 donde la fila i indica los valores mínimo y máximo de la i-ésima entrada.

S: número de neuronas en la capa oculta.

Ejemplo:

```
net = newp([-2 2; -2 2], 1)
```

Inicialización

```
net = init(net); % pesos y bias en cero

net.IW{1, 1} = [3, 4];
net.b{1} = 5;

net.inputweights{1, 1}.initFcn = 'rands';
net.biases{1}.initFcn = 'rands';
```

Simulación

- Para obtener la salida de una red se realiza una simulación:

Ejemplo:

```
p1 = [1;1];
a1 = sim(net, p1);
o
a1 = net(p1);
```

Regla de Aprendizaje de la red Perceptrón I

- El perceptrón es entrenado con aprendizaje supervisado.
- Un conjunto de pares de E/S (patrón/target) representan el comportamiento deseado.
- El objetivo es reducir el error e, que es la diferencia entre el valor deseado `t` y el resultante a:

$$e = t - a$$

Regla de Aprendizaje de la red Perceptrón II

- Se puede probar que la regla de aprendizaje del pereptrón converge hacia una solución en un número finito de pasos.
- La estrategia consiste en modificar el vector **w** haciendo que éste apunte hacia los patrones cuya salida es **1** y en contra de los patrones cuya salida es **0**.
- `learnp` es la función que realiza el cálculo de la variación de **w**.

Regla de Aprendizaje de la red Perceptrón III

- **CASO 1:** a = t (e = 0). El vector **w** no se altera.
- **CASO 2:** a = 0 y t = 1 (e = 1). El vector de entrada **p** se suma a **w**. Esto provoca que **w** se cierre hacia **p**, incrementando la posibilidad de que **p** sea clasificado como **1** en el futuro.
- **CASO 3:** a = 1 y t = 0 (e = -1). A **w** se le resta el vector de entrada **p**. Esto provoca que **w** se aleje de **p**, incrementando la posibilidad de que **p** sea clasificado como 0 en el futuro.

Regla de Aprendizaje de la red Perceptrón IV

- En consecuencia:

$$W = W + ep'$$

- Si consideramos el umbral como un peso cuya entrada siempre es 1 tenemos:

$$b = b + e$$

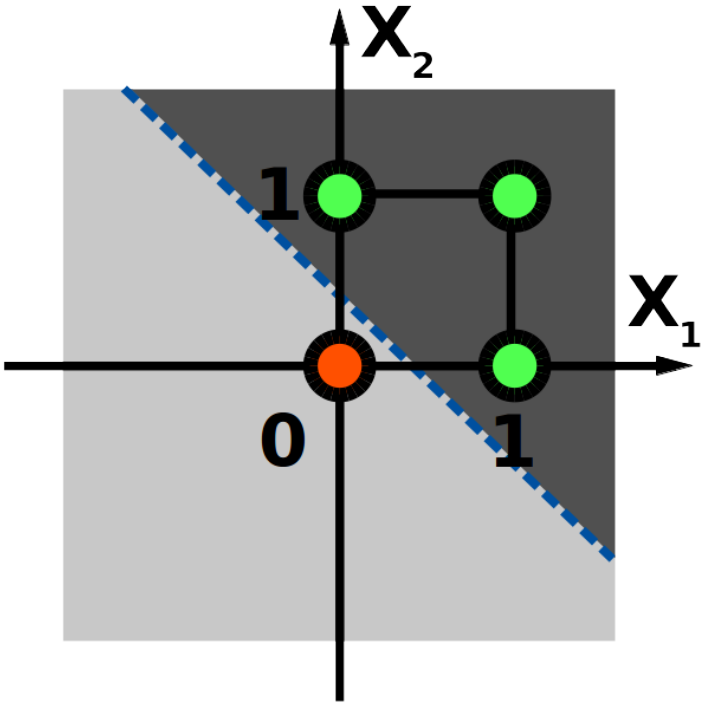
nnd4pr

Regla de Aprendizaje de la red Perceptrón V

- El proceso de encontrar nuevos pesos se repite hasta que el error sea aceptable.
- Se garantiza la convergencia en un número finito de pasos para todos los problemas que pueden ser resueltos por un perceptrón (estos son todos los problemas de clasificación "linealmente separables").

Perceptrón - clasificación

- AND?
- OR?
- XOR?



Entrenamiento I

- Se utilizan repetidamente `sim` y `learnp` introduciendo entradas al perceptrón y actualizando sus pesos.
- La función que lleva a cabo este loop es llamada `train`.
- La función `train` realiza una pasada por todas las entradas calculando para cada una la salida y el error.

Entrenamiento II

- Después de obtener el error para una entrada se realiza el correspondiente ajuste y se introduce la entrada siguiente.
- Se necesita una pasada posterior por todas las entradas para evaluar si se ha alcanzado el objetivo.
- Estas pasadas se denominan épocas.

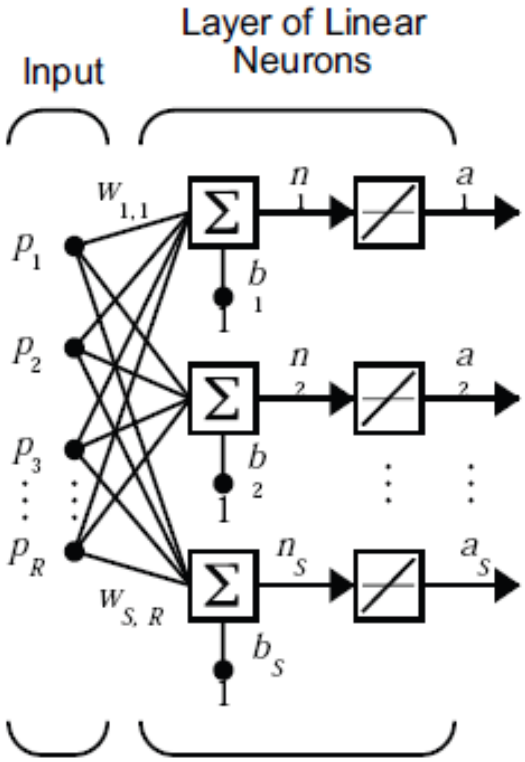
RED ADALINE

REDES LINEALES ADAPTATIVAS

Redes lineales

- Son similares al perceptrón pero su función transferencia es lineal.
- Al igual que el perceptrón estas redes resuelven problemas linealmente separables.
- En este caso minimiza el promedio de los errores al cuadrado.
- La función `linearlayer` crea una red lineal (Antes `newlin`).

Arquitectura de la ADAptive LInear NEtwork



Error cuadrático medio (LMS)

- Sea un conjunto de Q entradas y targets:

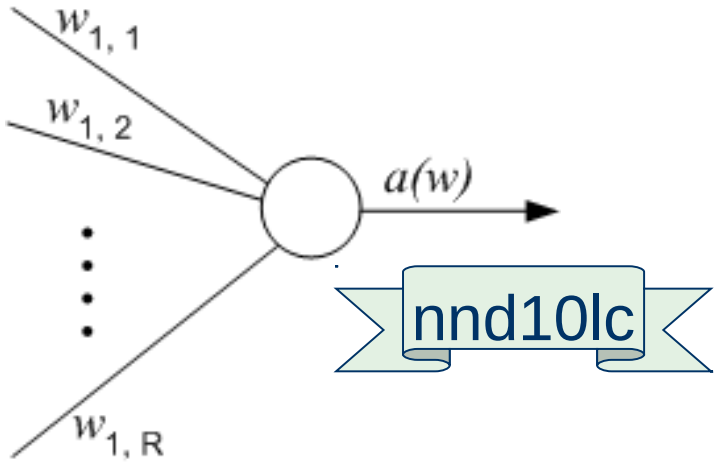
$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

Se aplican las Q entradas calculando cada error como la diferencia entre el objetivo y la salida. Se desea minimizar el promedio de los cuadrados de estos errores:

$$mse = \frac{1}{Q}\sum_{k=1}^{Q} e(k)^2 = \frac{1}{Q}\sum_{k=1}^{Q} (t(k) - a(k))^2$$

Algoritmo LMS (*least mean square*)

- También llamado algoritmo de Widrow-Hoff.
- La idea es ajustar los pesos para minimizar el *mse* (mean square error).
- ¿Cuáles deben ser los próximos pesos?



$$e^2(w) = (t - a(w))^2$$

El gradiente apunta hacia la dirección de crecimiento más rápido

Algoritmo LMS (*least mean square*)

$$\nabla E = \frac{1}{Q} \sum_{k=1}^{Q} \nabla e^2(w)$$

$$W = W - \alpha \nabla E$$

Algoritmo LMS (*least mean square*)

https://la.mathworks.com/help/deeplearning/ug/linear-neural-networks.html

REDES BACKPROPAGATION

Redes Backpropagation I

- Son una generalización de las redes lineales.
- Admiten múltiples capas y funciones derivables no lineales.
- Con una capa sigmoidea y una capa de salida lineal son capaces de aproximar cualquier función con un número finito de discontinuidades.

Redes Backpropagation II

- Utilizan la regla de aprendizaje de Widrow-Hoff con una serie de variaciones.
- El término backpropagation se refiere a cómo se aplica el cálculo del gradiente hacia capas anteriores (el nuevo peso deseado en la capa *s* es el target de la capa *s-1*).
- La función `feedforwardnet` crea una red backpropagation (Antes `newff`).

nnd12sd1

Redes Backpropagation III

- Las múltiples capas con funciones no lineales permiten que estas redes puedan resolver problemas "no linealmente separables".
- Una capa de salida con función transferencia lineal permite que la red produzca valores más allá del (-1, 1).
- Si se quiere restringir la salida, por ejemplo al (0, 1) se puede utilizar la función sigmoidea para la capa de salida.

Neural Network Toolbox (Matlab)

```
>> load house_dataset
```

- La red backpropagation más comunmente utilizada posee una capa oculta con 20 neuronas.

```
>> net = feedforwardnet(20);
>> net = configure(net, houseInputs, houseTargets);
```

- La cantidad de neuronas de entrada tanto como la de salida quedan determinadas por las características del dataset.

```
>> net = train(net, houseInput, houseTargets)
```

Ejemplo `train - init`

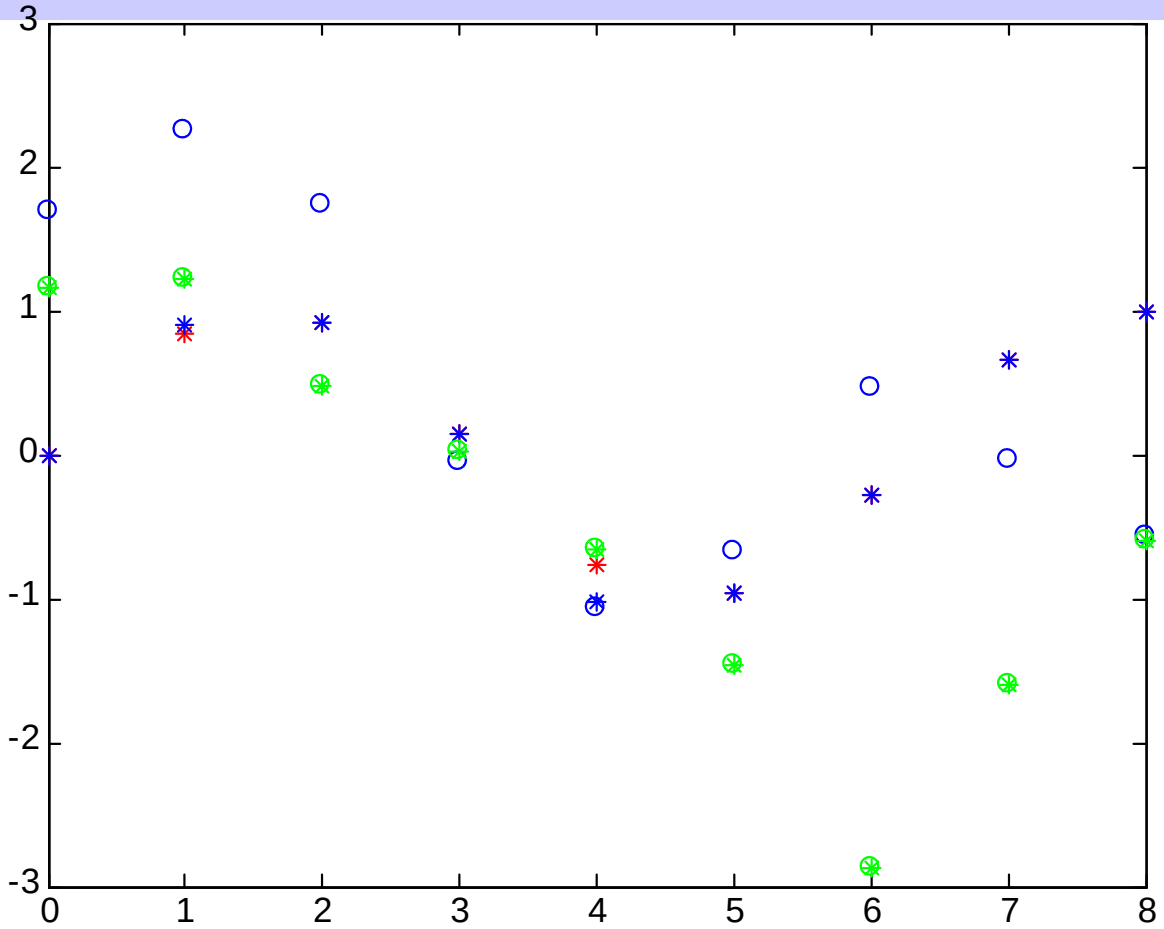
```
x = [0 1 2 3 4 5 6 7 8];
t = [0 0.84 0.91 0.14 -0.77 -0.96 -0.28 0.66 0.99];

net = feedforwardnet(10);
net = configure(net,x,t);
y1 = net(x);

net = train(net,x,t);
y2 = net(x);
plot(x,t,'o',x,y1,'x',x,y2,'*')

net = init(net);
y3 = net(x);
net = train(net,x,t);
y4 = net(x);
plot(x,t,'*r',x,y1,'ob',x,y2,'*b', x, y3, 'og', x, y4, '*g')
```

Ejemplo `train - init`



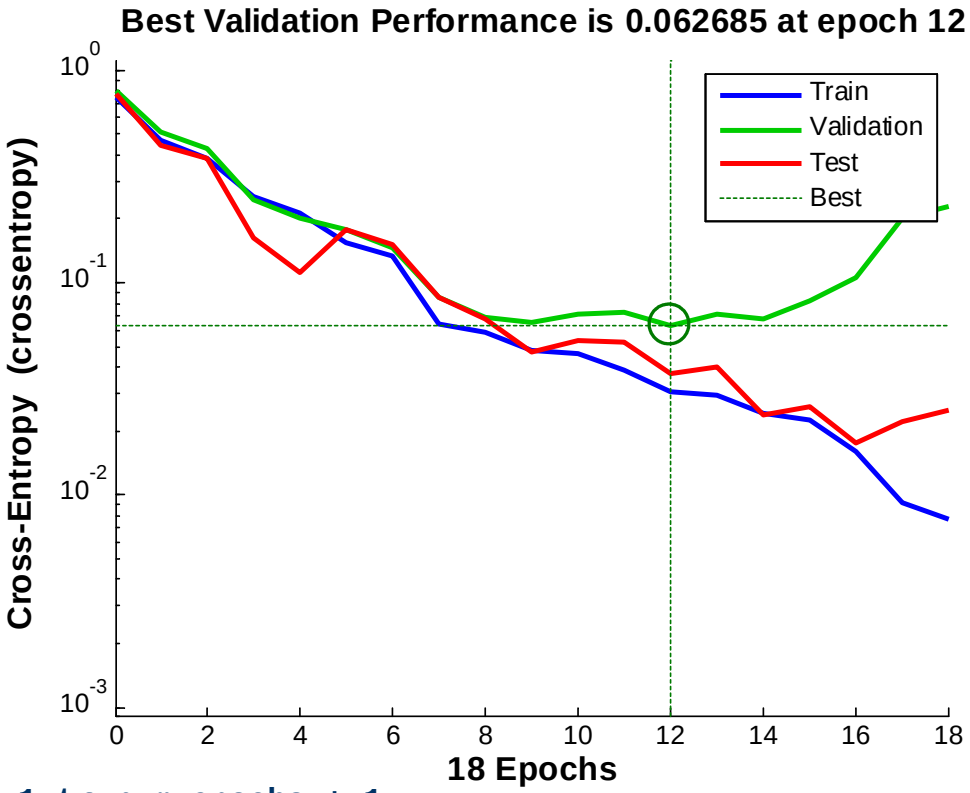
Reconocimiento de patrones

- Se puede entrenar redes para reconocimiento de patrones.
- Son redes feedforward que pueden classificar entradas en clases (conjuntos categóricos).
- Los datos objetivo (target) se presentan como vectores cuyos valores son todos cero excepto un 1 en la posición i, donde i es la clase que representan.

Reconocimiento de patrones

```
[x,t] = iris_dataset;
net = patternnet(10);
[net, tr] = train(net,x,t);
view(net)
y = net(x);
perf = perform(net,t,y); % tr.best_tperf
plotperform(tr);
classes = vec2ind(y);
```

Errores




```
xx = 1:tr.num_epochs + 1;
plot(xx, tr.perf, 'b', xx, tr.tperf, 'r', xx, tr.vperf, 'g')
```

Conjuntos de entrenamiento, validación y test

- El dataset es dividido aleatoriamente en tres conjuntos:
  - 60% de los datos se utilizan para entrenamiento (ajuste de pesos y umbrales).
  - 20% de los datos se utilizan para validación (ajuste de otros parámetros - learning rate, arquitectura, etc).
  - 20% de los datos se utilizan para test (valoración del modelo obtenido).

Conjuntos de entrenamiento, validación y test

- En Matlab el tamaño de los conjuntos se puede observar en `tr.trainInd`, `tr.valInd` y `tr.testInd`
- El entrenamiento continúa mientras se reduce el error de validación.
- Esta es una técnica muy utilizada para evitar el sobreentrenamiento.

Simulación de las redes

- Una vez entrenada la red, se la puede utilizar:

```
>> y = sim(net, p);
```

o

```
>> y = net(p)
```

- Para un conjunto de nuevos datos se espera un error similar al calculado para el conjunto de test.

Optimizar el modelo

- Inicializar nuevamente la red
- Cambiar el número de capas y neuronas internas.
  - Se puede agregar como cuarto argumento un arreglo con los nombres de las funciones transferencia a usar en cada capa.
- Por último, usar datos adicionales generalmente mejora el aprendizaje.

Redes Neuronales - Resumen (I)

- Capacidad de representación: muy alta. Fronteras de representación no lineales.
- Legibilidad: ninguna. El modelo resultante consiste en vectores de pesos para las conexiones entre neuronas.
- Tiempo de cómputo on-line: Rápido. Las operaciones son sumas y multiplicaciones.

Redes Neuronales - Resumen (II)

- Tiempo de cómputo off-line: muy lento. Gran cantidad de pesos a ajustar iterativamente.
- Parámetros a ajustar: complicado. Se debe ajustar la estructura de la red, el tamaño del paso (*learning rate*), condiciones de terminación, etc.
- Robustez ante ejemplos de entrenamiento ruidosos: alta.
- Sobreajuste: puede ocurrir. Se debe controlar.

SELECCIÓN DEL MODELO A UTILIZAR

Criterios de selección del modelo

- Dos decisiones fundamentales:
  - El tipo de modelo (árboles de decisión, redes neuronales, modelos probabilísticos, etc.)
  - El algoritmo utilizado para construir o ajustar el modelo a partir de las instancias de entrenamiento (existen varias maneras de construir árboles de decisión, varias maneras de construir redes neuronales, etc.)

Selección del modelo y/o algoritmo

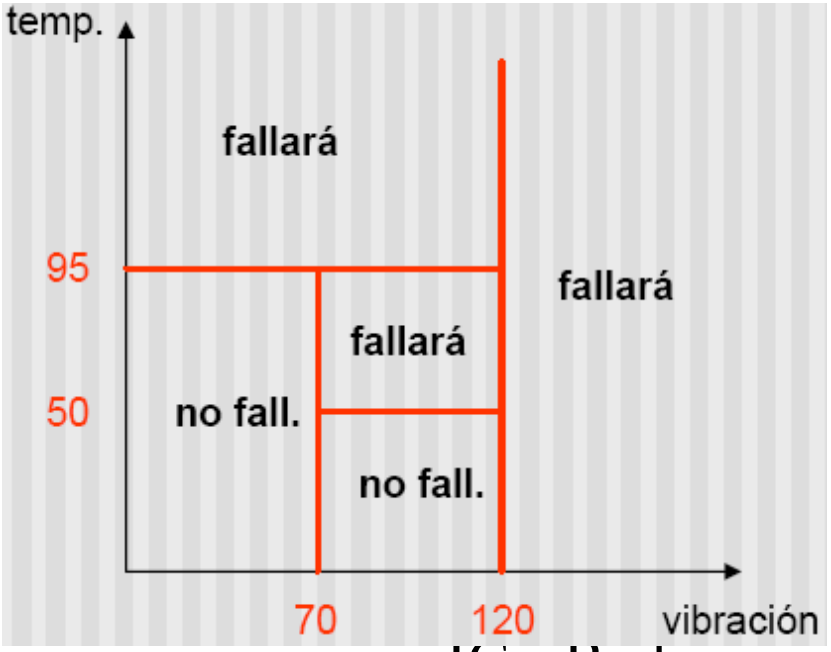
- Capacidad de representación.
- Legibilidad.
- Tiempo de cómputo on-line.
- Tiempo de cómputo off-line.
- Dificultad de ajuste de parámetros.
- Robustez ante el ruido.
- Sobreajuste.
- Minimización del error.

Selección del modelo y/o algoritmo
Capacidad de representación (I)

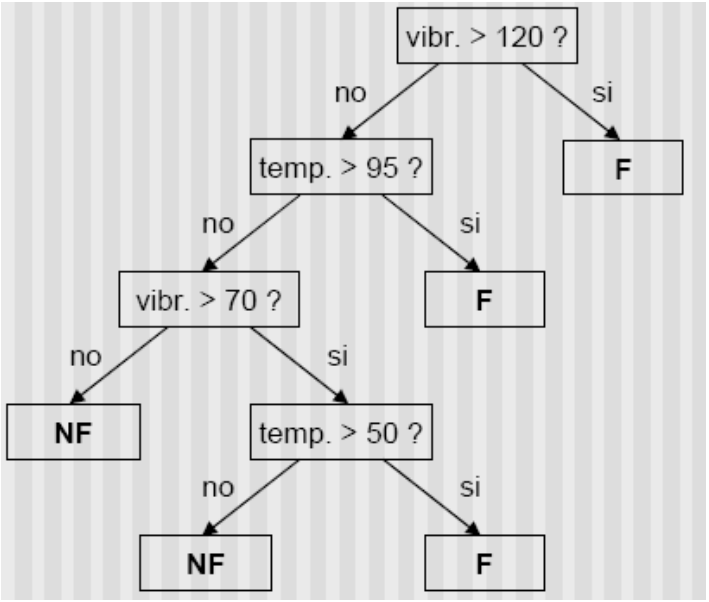
- Relacionado con el tipo de fronteras de decisión que se pueden expresar.
- Fronteras de decisión: separación de clases distintas.
- Cada modelo crea diferentes fronteras.

Selección del modelo y/o algoritmo
Capacidad de representación (II)

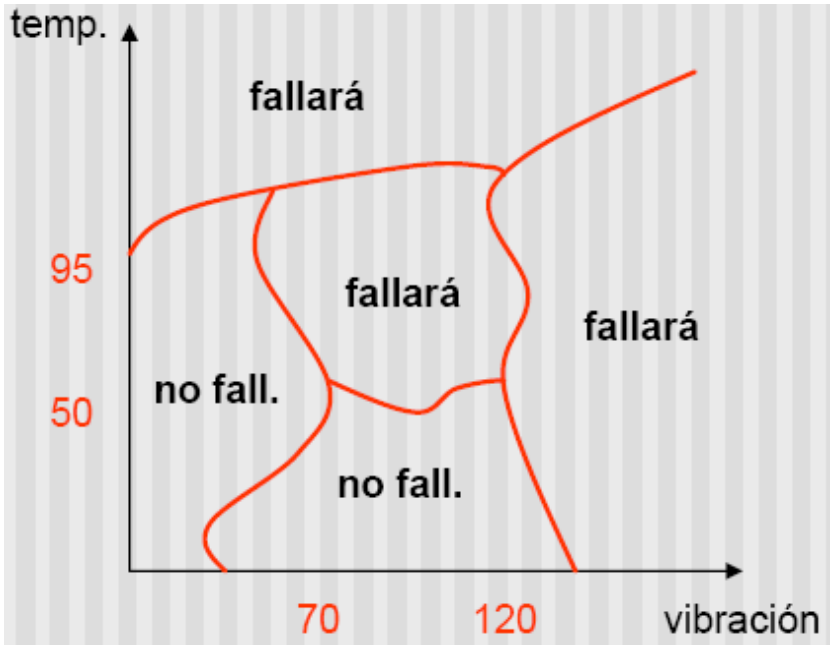
- Ejemplo con sólo dos atributos:



Árboles de decisión: fronteras perpendiculares a los ejes



Selección del modelo y/o algoritmo Capacidad de representación (III)



Redes Neuronales (NN), fronteras no lineales:

- Mayor capacidad de representación.
- Permiten representar conceptos más complejos que los árboles de decisión.
- Se estudiarán más adelante.

Selección del modelo y/o algoritmo Legibilidad (I)

- Capacidad de ser leído e interpretado por un humano.
- Árboles de decisión: fáciles de entender e interpretar:
  - conjunto de reglas.
  - en los niveles más altos están los atributos más importantes.
- Redes neuronales: difíciles (o imposibles) de interpretar:
  - pesos de conexiones entre neuronas.

Selección del modelo y/o algoritmo Legibilidad (II)

- Un modelo legible puede ofrecer información sobre el problema que se estudia (ej. indicar qué atributos afectan la probabilidad de fallo de una máquina y cómo).
- Un modelo no legible sólo puede ser usado como un clasificador (ej. permite predecir si una máquina fallará o no aplicando el modelo).

Selección del modelo y/o algoritmo
Tiempo de cómputo on-line (I)

- Es el tiempo necesario para clasificar una instancia:
  - Árboles de decisión: tiempo necesario para recorrer el árbol, evaluando las funciones lógicas de cada nodo.
  - Redes neuronales: tiempo necesario para realizar las operaciones (sumas, productos, sigmoides) incluidas en la red.

Selección del modelo y/o algoritmo
Tiempo de cómputo on-line (II)

- Este tiempo se consume cada vez que se debe clasificar una nueva instancia.
- Algunas aplicaciones requieren clasificar miles de instancias.
  - Ejemplo: clasificación de cada uno de los píxeles de una imagen área de un cultivo, río, ruta, etc.
  - Se requiere clasificar millones de instancias.
  - El tiempo de cómputo es muy importante.

Selección del modelo y/o algoritmo Tiempo de cómputo off-line (I)

- Es el tiempo necesario para construir o ajustar el modelo a partir de los ejemplos de entrenamiento.
  - Árboles de decisión: tiempo necesario para elegir la estructura del árbol, los atributos a situar en cada nodo y la optimización mediante la poda.
  - Redes neuronales: tiempo necesario para ajustar los pesos de las conexiones (puede tomar valores muy grandes).

Selección del modelo y/o algoritmo
Tiempo de cómputo off-line (II)

- Sólo se consume una vez, cuando mediante la utilización de los ejemplos de entrenamiento se genera y selecciona el resultado (modelo) más adecuado.
- Dependiendo de la aplicación no es un problema que el tiempo de cómputo off-line sea elevado (se deja una computadora procesando uno o tres días enteros).

Selección del modelo y/o algoritmo
Dificultad de ajuste de parámetros

- Se prefieren los algoritmos con pocos (o ningún) parámetros que ajustar.
- Se prefieren algoritmos con muy poca sensibilidad a la modificación de sus parámetros.
- Hay modelos muy difíciles de ajustar mediante parámetros (puede ocurrir con redes neuronales).

Selección del modelo y/o algoritmo
Robustez ante el ruido

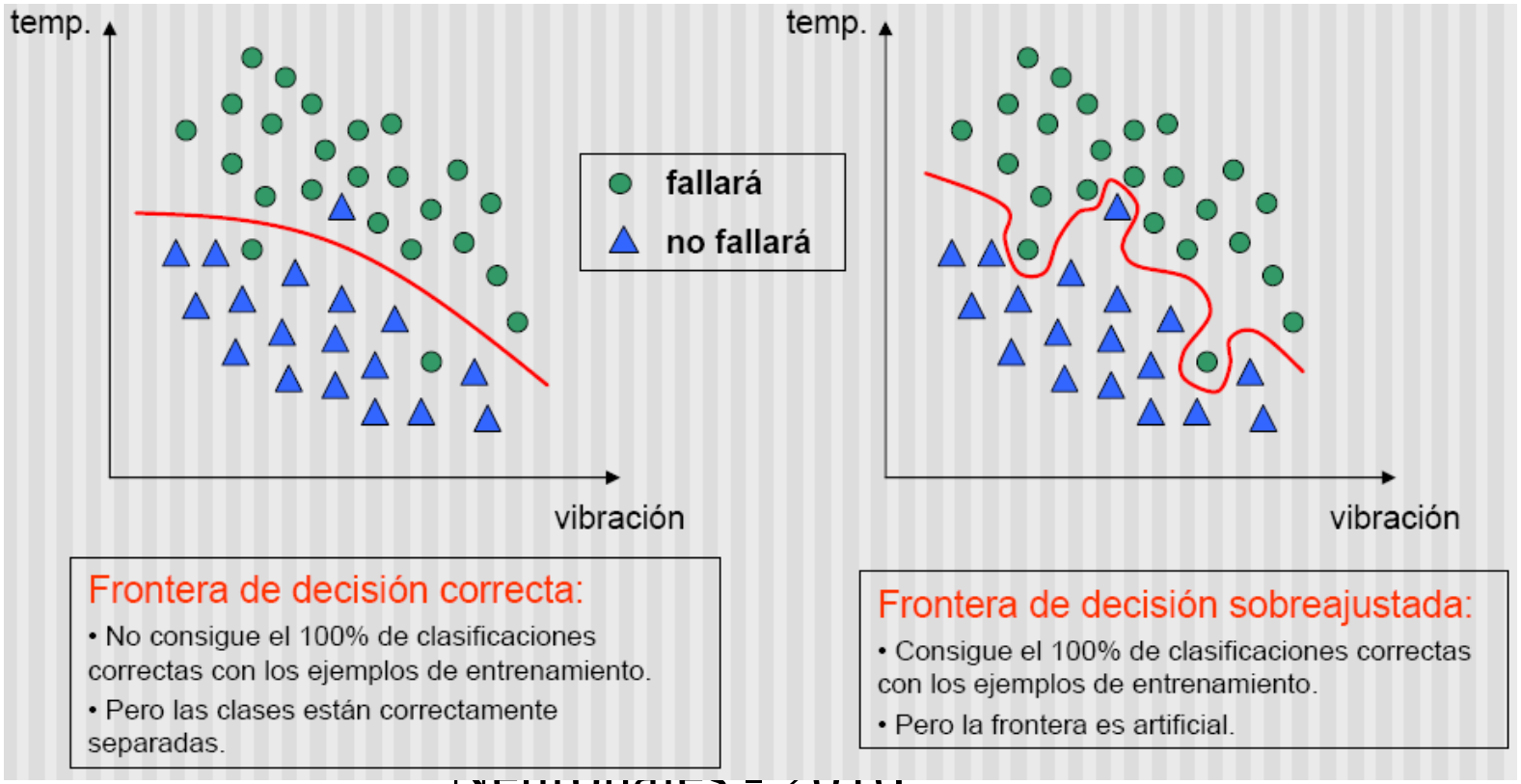
- Instancia de entrenamiento ruidosa:
  - etiquetada incorrectamente (ejemplo: una máquina que no falló, etiquetada como que sí falló).
  - algún atributo no está valorizado.
- Algunos algoritmos pueden funcionar adecuadamente aunque haya instancias ruidosas en el conjunto de entrenamiento (ej. árboles de decisión, redes neuronales).
- Otros algoritmos no ofrecen buenos resultados (ej. k-vecinos más cercanos).

Selección del modelo y/o algoritmo Sobreajuste (*overfitting*). (I)

- Problema muy común.
- El modelo está demasiado ajustado a las instancias y no funciona adecuadamente con nuevos casos.
- El modelo no es capaz de generalizar.
- Normalmente, fronteras de decisión muy complejas producen sobreajuste.

Selección del modelo y/o algoritmo Sobreajuste (*overfitting*). (I)

- Ejemplo con dos atributos:



Selección del modelo y/o algoritmo
Minimización del error

Ejemplo: Resultados arrojados para el problema de clasificar los datos de Contraceptive Method Choice (CMC), con diferentes métodos.

	Árboles de decisión	Redes neuronales	Naive-Bayes con normales	Naive-Bayes con histogramas	k-primeros vecinos
Medias	26.41	23.711339	33.9851806	25.4706348	26.094287
Desviaciones estándar	2.96739691	4.40077215	4.67207077	3.70521908	3.50755602

Bibliografía

- Machine Learning - Tom Mitchell – McGrawHill
- Neural Network Toolbox (For Use with MATLAB®) - Howard Demuth, Mark Beale.
- Curso de doctorado "Aprendizaje Automatizado y Data Mining" Grupo de Ingeniería de Sistemas y Automática (Universidad Miguel Hernández) http://isa.umh.es/asignaturas/aprendizaje/index.html