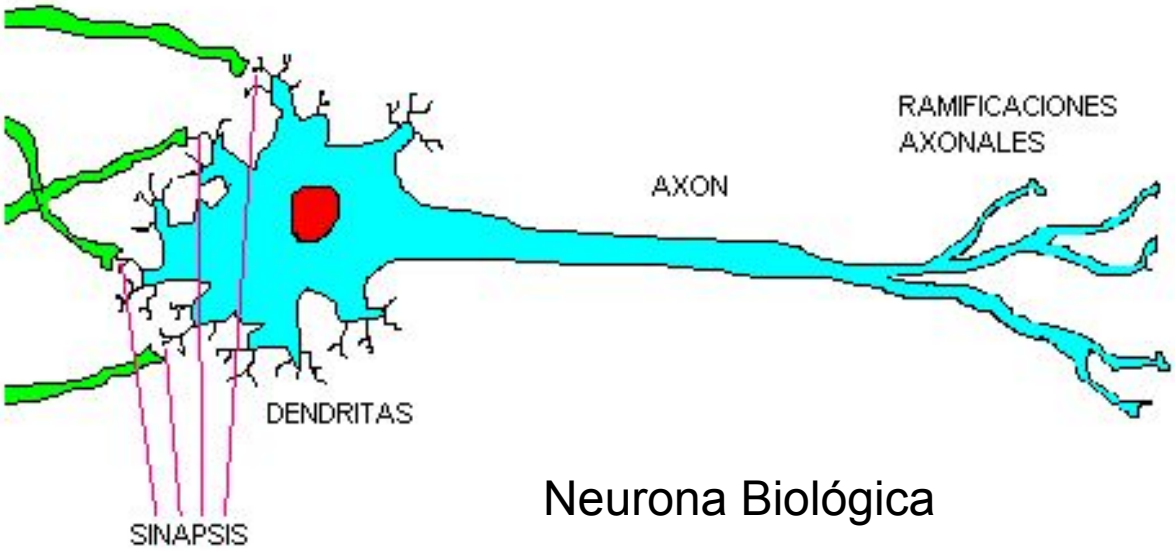
Clasificación

Redes Neuronales

Por Elizabeth León Guzmán

Redes Neuronales Biológicas

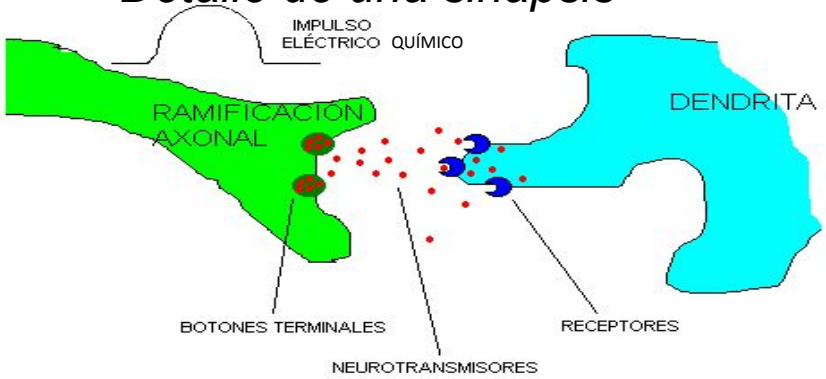


Neurona Biológica

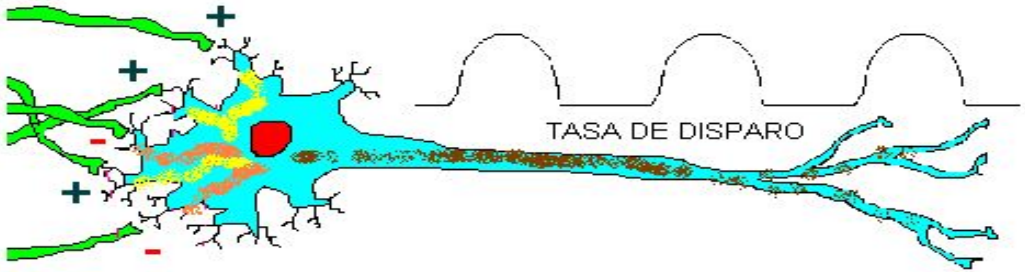
Dibujos tomados de http://www.iiia.csic.es/~mario/rna/tutorial/RNA_marcos.html

Redes Neuronales Biológicas

Detalle de una sinapsis



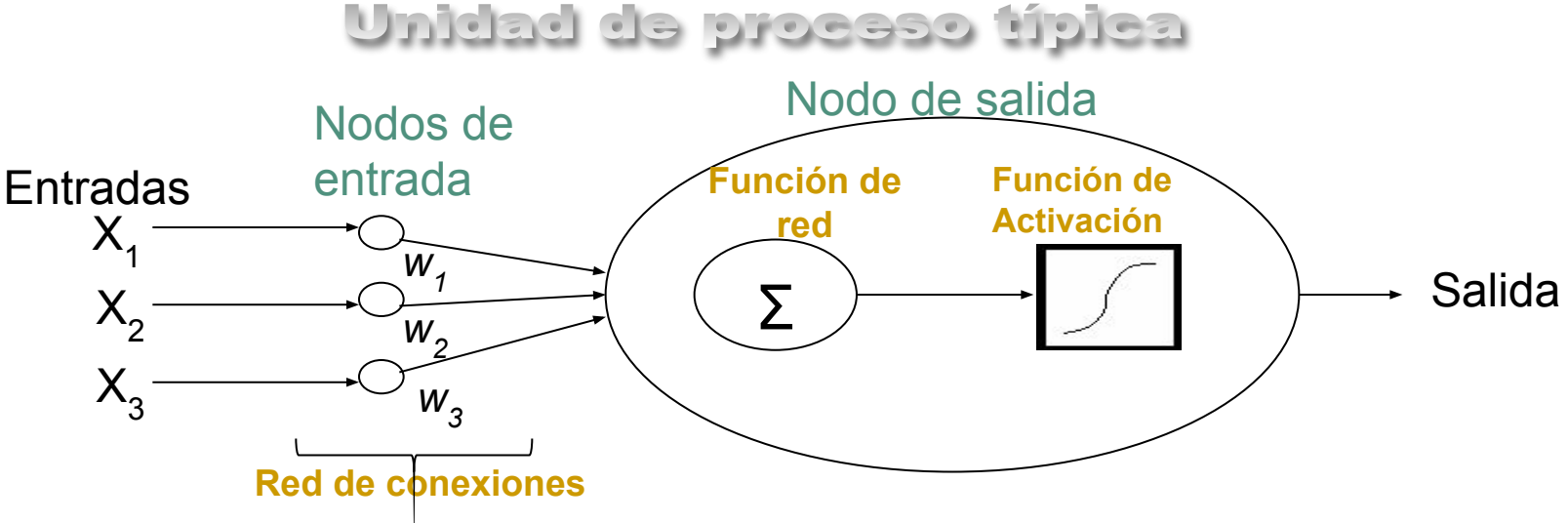
Activación y disparo de una neurona



Dibujos tomados de http://www.iiia.csic.es/~mario/rna/tutorial/RNA_marcos.html

Redes Neuronales Artificiales

- Simular sistemas neuronales biológicos
- Las neuronas se modelan mediante **unidades de proceso**



Unidades de proceso

Una unidad de proceso se compone de:

- una **red de conexiones** de entrada,
- una **función de red**, encargada de computar la entrada total combinada de todas las conexiones,
- un **núcleo central de proceso**, encargado de aplicar la función de activación, y
- **la salida**, por dónde se transmite el valor de activación a otras unidades.

Funciones de red o propagación

- Calcula el valor de base o entrada total a la unidad
- La función de red es típicamente la sumatoria ponderada de todas las entradas recibidas:
  - Entradas multiplicadas por el peso o valor de las conexiones.
  - Equivale a la combinación de las señales excitatorias e inhibitorias de las neuronas biológicas.

$$net_j(X, W_j) = \sum_{i=1}^{n} x_i w_{ij}$$

Función lineal de base (LBF)

$$net_j(X, W_j) = \sqrt{\sum_{i=1}^{n} (x_i - w_{ij})^2}$$

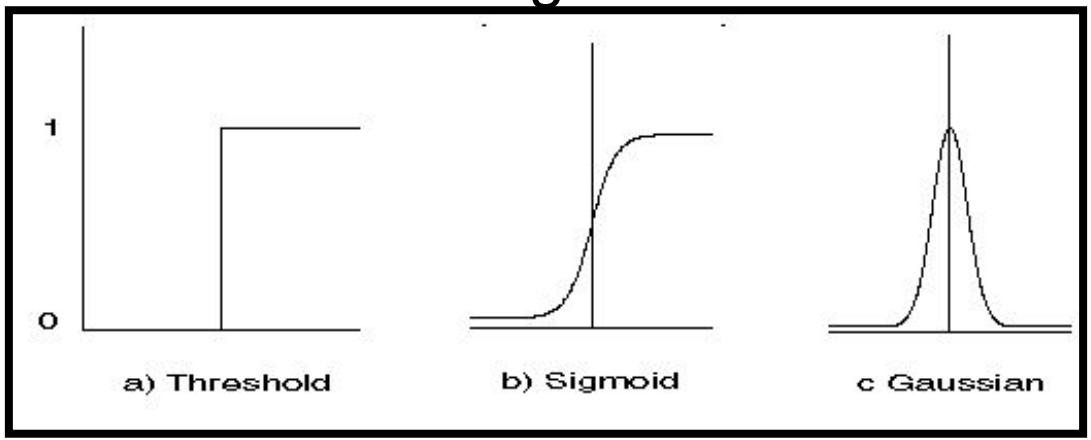
Función radial de base (RBF)

Función de Activación

- Característica principal o definitoria de las neuronas (la que mejor define el comportamiento de la neurona)
- Se encarga de calcular el nivel o estado de activación de la neurona en función de la entrada total
- Se usan diferentes tipos de funciones, desde funciones simples de umbral a funciones no lineales.

Función de Activación

- La función de activación suele ser alguna función de umbral o una función sigmoidal.



escalón unitario: la función devuelve 0 por debajo del valor crítico (umbral) y 1 por encima.

Logística:basada en la función sigmoide

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

con un perfil parecido al escalón de una función de umbral, pero continua.

Conexiones ponderadas

- Equivalen a las conexiones sinápticas
- El peso de la conexión equivale a la fuerza o efectividad de la sinapsis.
- Las existencia de conexiones determina si es posible que una unidad influya sobre otra
- El valor de los pesos y el signo de los mismos definen el tipo (excitatorio/inhibitorio) y la intensidad de la influencia.

Salida

- Calcula la salida de la neurona en función de la activación de la misma
- Normalmente no se aplica más que la función identidad, y se toma como salida el valor de activación.
- El valor de salida cumpliría la función de la tasa de disparo en las neuronas biológicas.

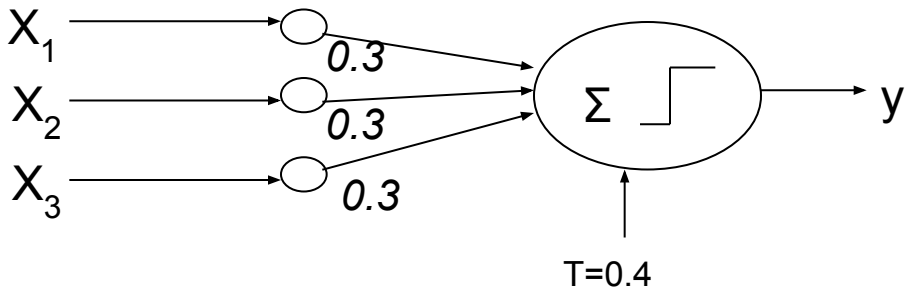
Ejemplo

Conjunto de datos

X_1	X_2	X_3	y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1

y es -1 si al menos 2 de las entradas es cero, y 1 si al menos 2 de las entradas son mayores de cero

Perceptron

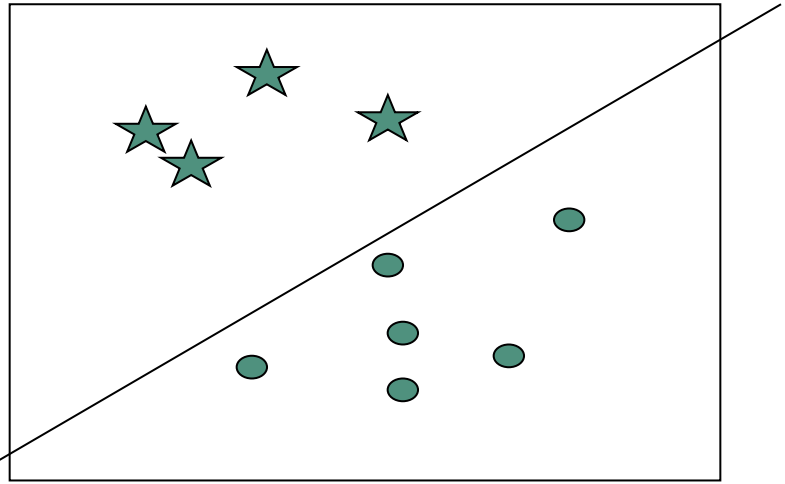


$$\hat{y} = \begin{cases} 1, & si \quad 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 > 0 \\ -1, & si \quad 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 < 0 \end{cases}$$

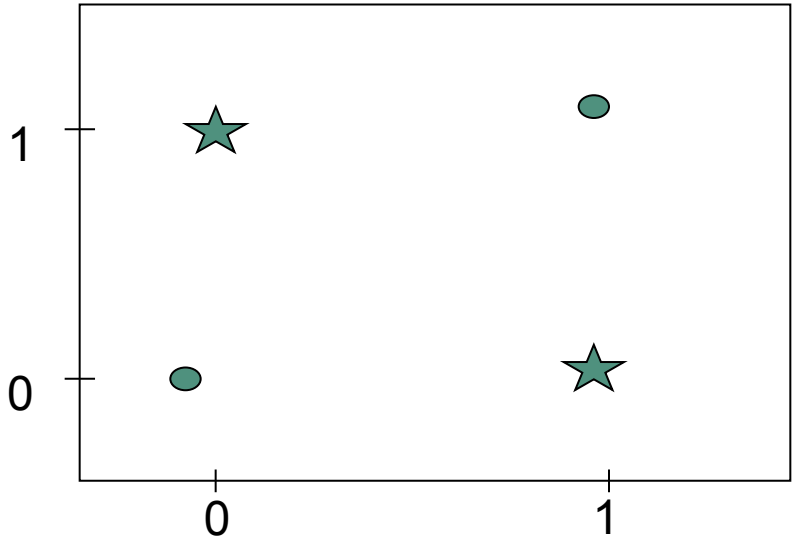
$$x_1 = 1, x_2 = 1, x_3 = 0 \qquad \hat{y} = 1$$
$$x_1 = 0, x_2 = 1, x_3 = 0 \qquad \hat{y} = -1$$

$$\hat{y} = sign(w_d x_d + w_{d-1} x_{d-1} + \ldots + w_2 x_2 + w_1 x_1 - t$$

Linealmente separables



Linealmente separables



Linealmente no separables

Ejercicio: AND en Scikit-Learn

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

```
import numpy as np
from sklearn.linear_model import Perceptron
clf = Perceptron()

X = np.array([[0., 0.], [0., 1.], [1., 0.], [1., 1.]])
y = [0, 0, 0, 1]
clf.fit(X, y)
```

Ejercicio: AND en Scikit-Learn

```
print('w_1 y w_2: {}'.format(clf.coef_))

print('Predicciones: {}'.format(clf.predict(X)))
print('Accuracy: {}'.format(clf.score(X, y)))
```

Para visualizar la región de decisión:

```
pl.figure(figsize = (10, 6))
plot_decision_region(X, gen_pred_fun(clf))
plot_data(X, y, size=100)
```

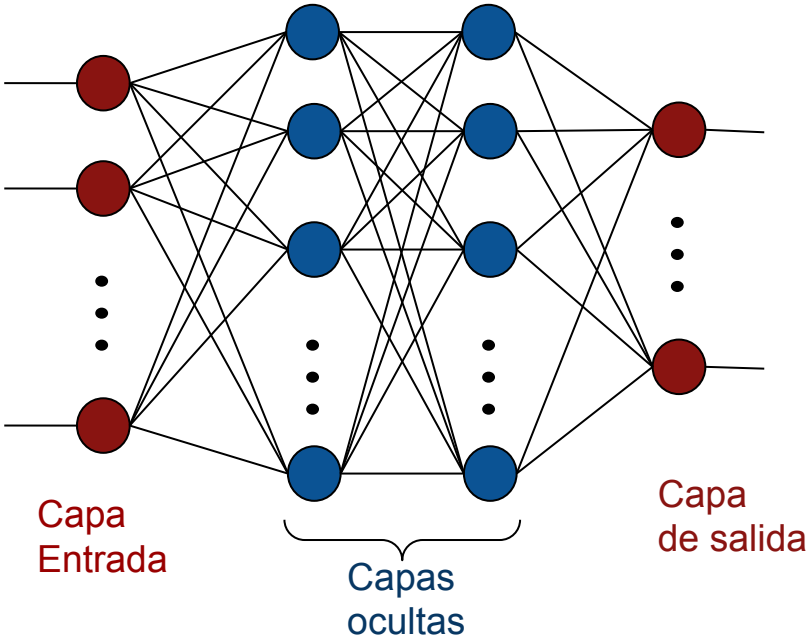
Ejercicio 2: XOR en Scikit-Learn

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

```
import numpy as np
from sklearn.linear_model import Perceptron
clf = Perceptron()

X = np.array([[0., 0.], [0., 1.], [1., 0.], [1., 1.]])
y = [0, 1, 1, 0]
clf.fit(X, y)
```

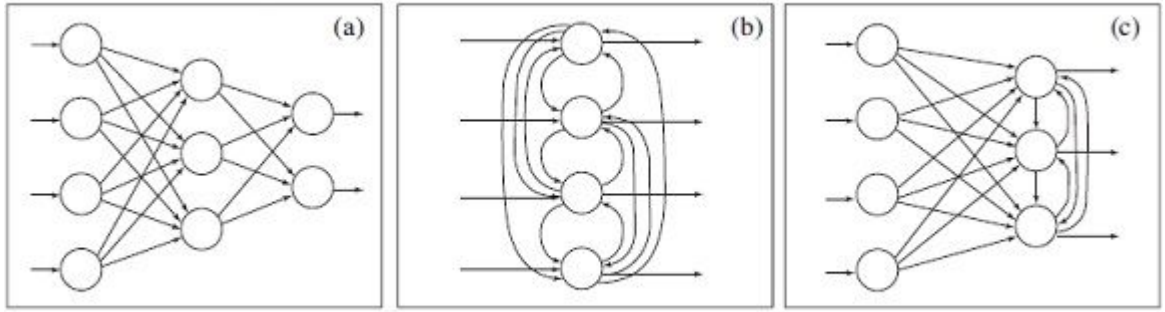
Redes Neuronales Multicapa



- **Capa de entrada:** almacena la información de entrada (preproceso)
- **Capas ocultas:** encargadas de extraer, procesar y memorizar la información

Redes Neuronales Multicapa

Figura 5. Tipos de conexiones en una red neuronal



Fuente: Hilera (1994).

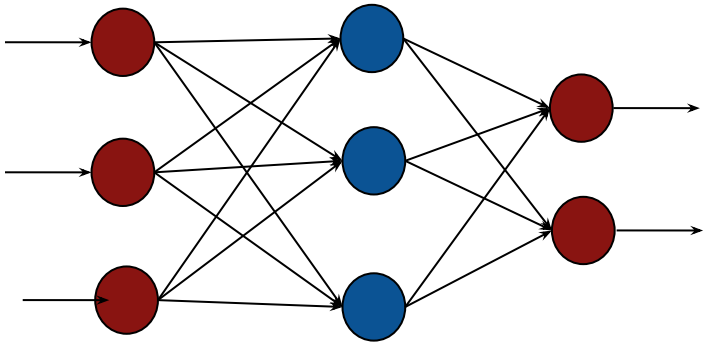
Tipos de redes. En función de cómo se **interconectan** unas capas con otras:

- ***Redes recurrentes*** (feed-back)
- **Redes no recurrentes** o ***redes en cascada*** (feed-forward).

Redes no recurentes

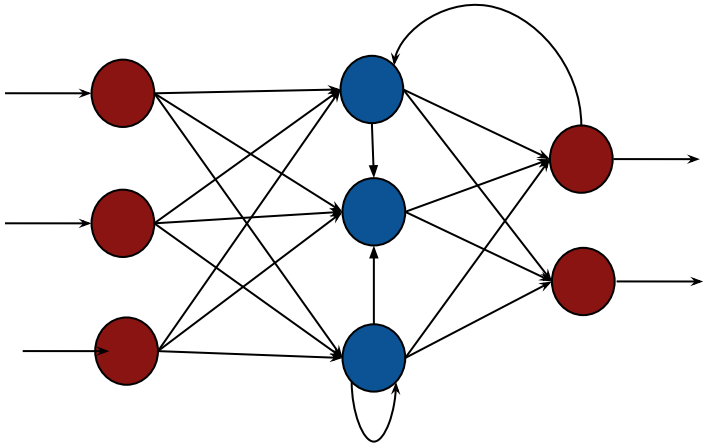
En cascada - feed back

La información fluye unidireccionalmente de una capa a otra (desde la capa de entrada a las capas ocultas y de éstas a la capa de salida), y no se admiten conexiones intracapa.



Redes Recurrentes - feed forward

La información puede volver a lugares por los que ya había pasado, formando **bucles**, y se admiten las **conexiones intracapa** (laterales), incluso de una unidad consigo misma.



Proceso de Aprendizaje

- El aprendizaje consiste:
  - en la **presentación de patrones** a la red,
  - y la subsiguiente **modificación de los pesos** de las conexiones siguiendo alguna regla de aprendizaje que trata de **optimizar su respuesta,**
- generalmente mediante la minimización del error o la optimización de alguna "función de energía".

Proceso de Aprendizaje

- **Aprendizaje Supervisado:** consiste en la presentación de patrones de entrada junto a los patrones de salida deseados (targets) para cada patrón de entrada
- **Aprendizaje No Supervisado:** No se le presentan a la red los patrones de salida deseados, se le deja seguir alguna **regla de auto-organización**.
- **Aprendizaje Reforzado:** El supervisor se limita a indicar si la salida ofrecida por la red es correcta o incorrecta, pero no indica cual respuesta debe dar.

Regla de Aprendizaje

- Una característica esencial de la red es la **regla de aprendizaje** usada, que indica **cómo se modifican los pesos de las conexiones en función de los datos usados en la entrada**, es decir, de la historia de aprendizaje de la red.
- Por ejemplo, entre los algoritmos de aprendizaje supervisado, la **regla delta generalizada**, modifica los pesos realizando en cada ciclo de aprendizaje un **incremento a los pesos proporcional a la tasa de variación del error respecto al peso**, en sentido negativo.

Aprendizaje Supervisado

El aprendizaje consiste en la modificación de los pesos de las conexiones en el sentido de reducir la discrepancia entre la salida obtenida y la deseada:

1. Aleatorizar los pesos de todas las conexiones (preferiblemente con valores pequeños)
2. Seleccionar un par de entrenamiento, es decir, un patrón de entrada y el patrón de salida deseado (target) correspondiente.

Aprendizaje Supervisado

3. Presentar el patrón de entrada y calcular la salida de la red mediante las operaciones usuales: sumatoria de las entradas ponderadas, función de activación y transferencia a la siguiente capa, hasta llegar a la capa de salida. (inicialmente obtenemos salidas aleatorias, ya que los pesos de las conexiones son aleatorios)

Aprendizaje Supervisado

4. Cálculo del error o discrepancia entre la salida obtenida y la deseada.
   - El error (**función objetivo**) se suele definir como la suma de los cuadrados de las diferencias entre las salidas reales obtenidas y las deseadas, promediado para todas las unidades de salida y todos los patrones de entrenamiento.

$$E(w) = \frac{1}{2}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

   - Si el error es menor de cierto criterio fijado de antemano, incrementar el número de ejemplos correctos; si todos los ejemplos se han clasificado correctamente, finalizar,sino continuar.

Aprendizaje Supervisado

5. Aplicar la regla de aprendizaje, es decir, ajustar los pesos de las conexiones tratando de disminuir el error, generalmente mediante el cálculo de tasas de variación o gradientes del error, **reglas de aprendizaje por gradiente descendiente**.
   - Para reducir el error habrá que modificar los pesos de las conexiones, en proporción a la tasa relativa de variación del error con respecto a la variación del peso, o sea, la derivada del error respecto al peso, EP (error respecto al peso).

Aprendizaje Supervisado

- Una forma de calcular el EP sería perturbar levemente un peso y observar como varía el error, pero no resultaría eficiente si trabajamos con muchas conexiones (este es el fundamento de algunos métodos estadísticos).

7. Volver al paso 2

Actualización de Pesos

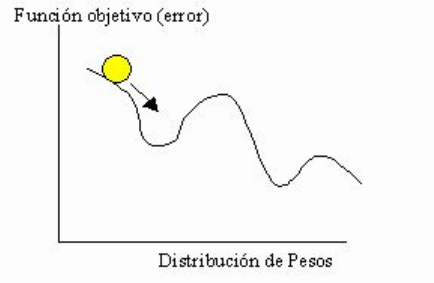
Fórmula para actualizar el peso usada en el perceptron

$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - y_j^{(k)})x_{ij}$$

λ rata de aprendizaje (parámetro)

Gradiente descendente

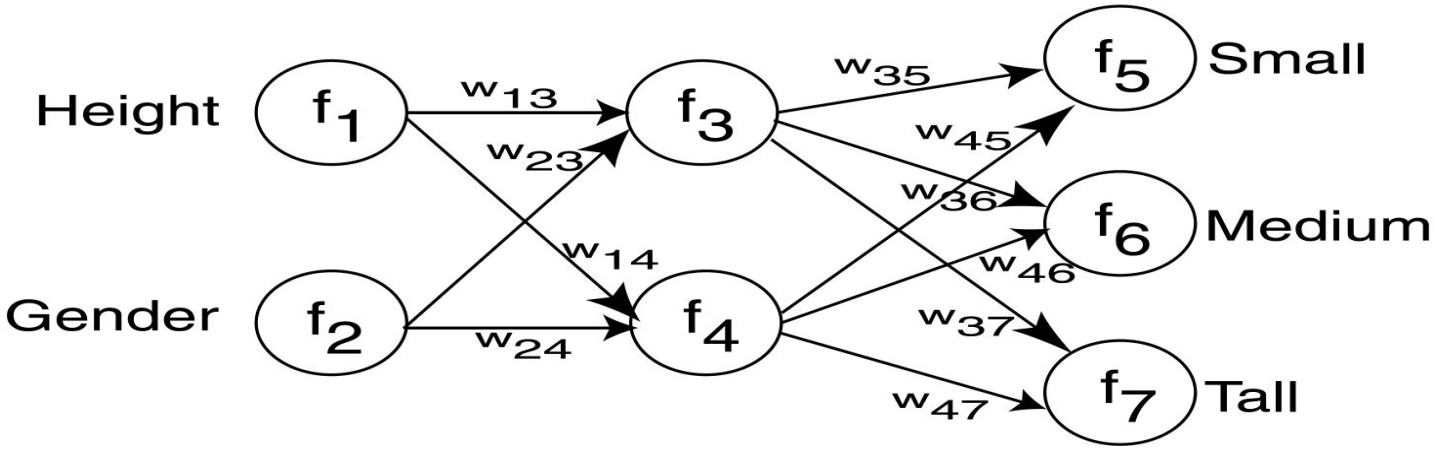
$$w_j \leftarrow w_j - \lambda \frac{\partial E(w)}{\partial w_j}$$



Retropropagación

- La retropropagación consiste en propagar el error hacia atrás, es decir, de la capa de salida hacia la capa de entrada, pasando por las capas ocultas intermedias y ajustando los pesos de las conexiones con el fin de reducir dicho error.
- Hay distintas versiones o reglas del algoritmo de retropropagación y distintas arquitecturas conexionistas a las que pueden ser aplicados.

Redes Neuronales para clasificación



Características de diseño

Determinar:

- Número de nodos en la capa de entrada
  - Un nodo de entrada por cada variable numérica o binaria
  - Variable categórica
    - Un nodo por cada valor categórico, o
    - Codificar la variable (de aridad k) usando $\log_2 k$ nodos de entrada
- Número de nodos en la capa de salida
  - Problema de dos clases, un nodo de salida
  - Problema de k clases, k nodos de salida

Características de diseño

- Topología de la red
  - Número de capas ocultas, nodos ocultos
  - Feedforward o recurrente arquitectura de red
- Los pesos y “biases” deben ser inicializados (valores aleatorios son aceptados)
- Datos con valores perdidos deben ser removidos o remplazados

Características de las redes neuronales

- Red neuronales multicapa con al menos una capa oculta son **aproximadores universales** (pueden ser usadas para aproximar cualquier función objetivo)
- Escoger topología apropiada para evitar overfitting
- Pueden haber dimensiones redundantes, ya que los pesos son aprendidos. Los pesos de dimensiones redundantes se caracterizan por ser muy pequeños

Características de las redes neuronales

- El método del gradiente descendente usado en el aprendizaje puede converger a mínimos locales. (se puede incluir un término de momento a la fórmula de actualización del peso)
- Entrenar una red neuronal es costoso , sin embargo la parte de test es rápida.

Ejercicio 3: XOR Multicapa en Scikit-Learn

```
from sklearn.neural_network import
MLPClassifier

X = np.array([[0., 0.], [0., 1.], [1., 0.],
[1., 1.]])
y = [0, 1, 1, 0]
```

Ejercicio 3: XOR Multicapa en Scikit-Learn

```
clf = MLPClassifier(solver='sgd',
learning_rate='constant', learning_rate_init=0.001,
                    activation='tanh',
max_iter=100000,tol=0.00000001,hidden_layer_sizes=(2))
clf.fit(X, y)
clf.score(X, y)
```

Ejercicio 3: XOR Multicapa en Scikit-Learn

Evolución del error de entrenamiento:

```
plt.ylabel('Mean Square Error')
plt.xlabel('Epochs')
plt.plot(clf.loss_curve_)
```

Ejercicio 3: XOR Multicapa en Scikit-Learn

Visualizando la superficie de decisión:

```
pl.figure(figsize = (10, 6))
plot_decision_region(X, gen_pred_fun(clf))
plot_data(X, y, size=100)
```