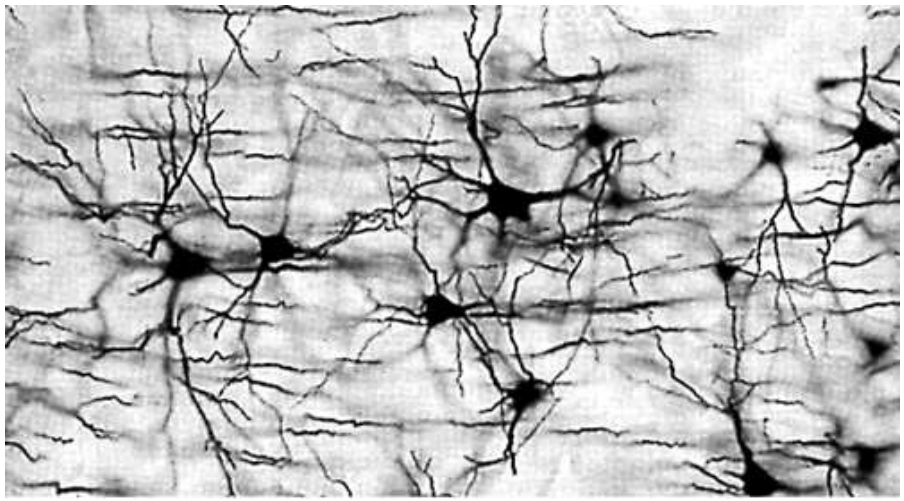
Reconocimiento estadístico de patrones: semana 9



red neuronal biológica

Plan de trabajo para la clase de hoy

1. Conceptos generales de clasificación
2. Clasificador k-vecino más cercano
3. Clasificador Bayesiano óptimo
4. Análisis discriminante lineal (LDA)
5. Clasificadores lineales y el Modelo perceptrón
6. Máquinas de soporte vectorial
7. Regresión logística
8. Redes neuronales como ejemplo de un clasificador no lineal, tipo conexionista
   (a) Definición de una red tipo feedforward para regresión
   (b) Ajuste usando el algorítmo de backpropagation para regresión
   (c) Definición y ajuste de una red para clasificación
   (d) Regularización
   (e) Redes de base radial

8. Redes Neuronales (RN)

8.1 Definición de RN para regresión

Punto de partida: ¿ Cómo definir funciones $f(x)$ en varias variables, $x = (x_1, \cdots, x_d)$?

Algunos ejemplos:

$$f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_d x_d.$$

$$f(x) = \beta_0 + \sum_i \beta_i x_i + \sum_{i,j} \beta_{i,j} x_i x_j.$$

Se puede ver lo anterior como combinar de manera particular funciones básicas tipo:

$$x \mapsto x_i \quad x \mapsto x_i x_j \quad x \mapsto x_i^2 \quad etc.$$

Veremos que un arbol define cierto tipo de función aditiva basada en una partición del espacio:

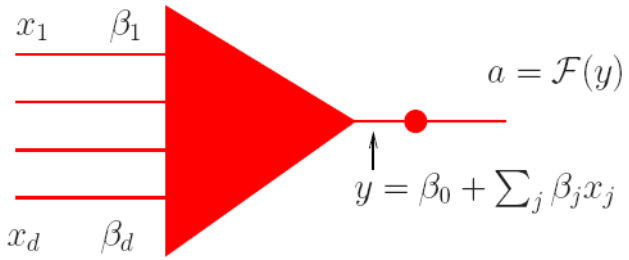
$$f(x) = \sum_i c_i I(x \in R_i).$$

IDEA: construir función compleja a través de composición de muchas funciones sencillas

La función base (= la función sencilla): **Para una función univariada $\mathcal{F}()$ dada, define:**

$$f(x) = \mathcal{F}(\beta_0 + \sum_j \beta_j x_j)$$

Visualización:



Elecciones para $\mathcal{F}()$:

$$\textbf{función idéntica: } \mathcal{F}(x) = x, \textbf{ así: } f(x) = \beta_0 + \sum_j \beta_j x_j,$$

$$\textbf{función signo: } \mathcal{F}(x) = sign(x), \textbf{ así: } f(x) = I(\beta_0 + \sum_j \beta_j x_j \geq 0),$$

$$\textbf{función logística (sigmoidal): } \mathcal{F}(x) = \frac{1}{1+\exp(-x)}, \textbf{ así: } f(x) = \frac{1}{1+\exp(-\beta_0 - \Sigma_j \beta_j x_j)},$$

Lo anterior llamamos una neurona.

Ejemplo: si x es vector binario, i.e. $x_i \in \{0, 1\}$; en caso $\mathcal{F}()$ es función signo:

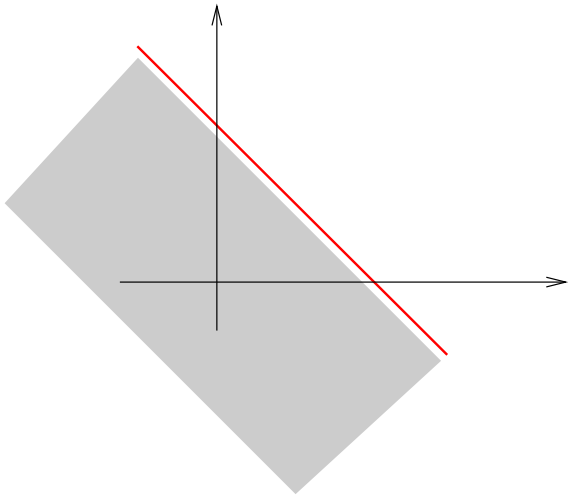
$$f(x) = I(-d + \sum x_i \geq 0) \text{ es función lógica AND}$$

$$f(x) = I(-1 + \sum x_i \geq 0) \text{ es función lógica OR}$$

Tarea: ¿ cómo construir función NOT?

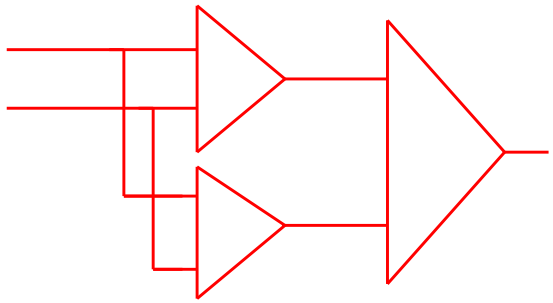
Ejemplo: si x es vector real, i.e. $x_i \in \mathcal{R}$; en caso $\mathcal{F}()$ es función signo:

f define hiperplano (= modelo perceptrón de antes)



Composición de funciones bases:

Tomamos como input de una función base las salidas de otras funciones bases.



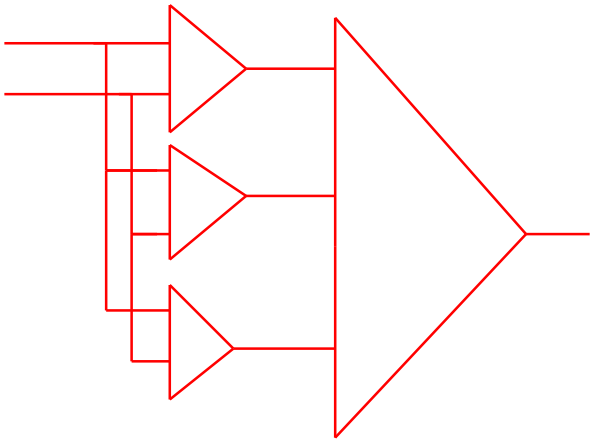
Notación: función asociada con neurona i: $f(x) = \mathcal{F}(\beta_{i,0} + \Sigma_j \beta_{i,j} x_j)$

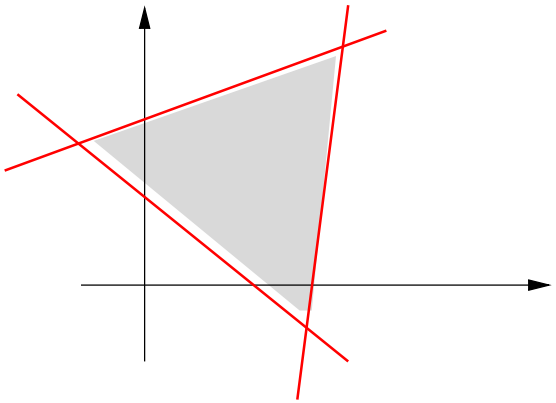
$$\mathcal{F}(\beta_{3,0} + \beta_{3,1}\mathcal{F}(\beta_{1,0} + \beta_{1,1}x_1 + \beta_{1,2}x_2) + \beta_{3,2}\mathcal{F}(\beta_{2,0} + \beta_{2,1}x_1 + \beta_{2,2}x_2))$$

Por ejemplo si x es vector real y $\mathcal{F}()$ es función sigmoide:

$$f(x) = \frac{1}{1 + \exp(-(\beta_{3,0} + \beta_{3,1}\frac{1}{1+\exp(-(\beta_{1,0}+\beta_{1,1}x_1+\beta_{1,2}x_2))} + \beta_{3,2}\frac{1}{1+\exp(-(\beta_{2,0}+\beta_{2,1}x_1+\beta_{2,2}x_2)}))}$$

Por ejemplo si x es vector binario y $\mathcal{F}()$ es función signo:

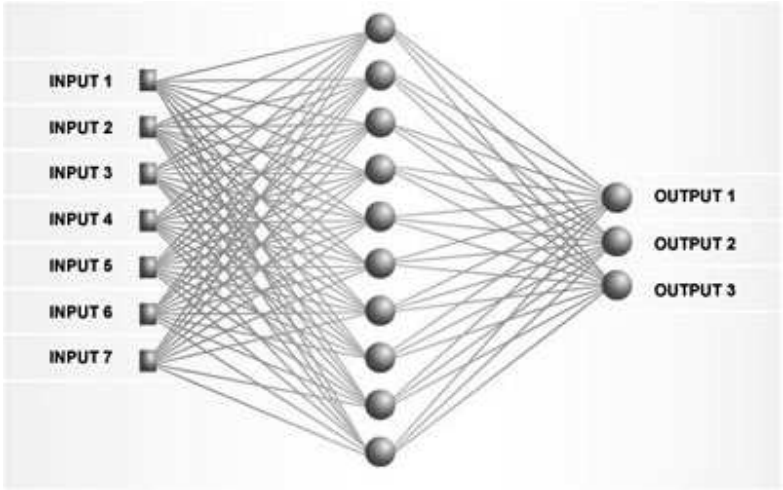




En general: ponemos las neuronas en capas

Las entradas de las neuronas de capa $c+1$ son las salidas de las neuronas de capa c.

Dos capas especiales: entradas (primera capa) y salidas (última capa).



Simplificación: define $x_0 = 1$, así $f(x) = \mathcal{F}(\Sigma_{j=0}^{d} \beta_{i,j} x_j)$. Lo anterior se llama una red tipo feedforward.

Observa: una red consiste de una estructura (topología) y los parámetros para esta estructura. La estructura óptima se busca típicamente con *trial and error*; por ejemplo por validación cruzada.

8.2 Estimación de los parámetros para regresión

Para una estructura de red dada, define β como el vector con todos los parámetros $\{\beta_{i,j}\}$.

Dados los datos $\{in^d, out^d\}$ (antes decíamos $\{x_i, y_i\}$);

una manera para buscar $f_\beta()$ es

$$\min_\beta E(\beta; \{in^d, out^d\}) := \min_\beta \frac{1}{2} \sum_d ||out^d - f_\beta(in^d)||^2$$

Tenemos que recurrir a métodos iterativos de optimización; por ejemplo método del gradiente:

Elige β, η, ϵ

$convergido$<- FALSE

while (!$convergido$)

{$\beta nuevo$ <- $\beta - \eta \nabla_\beta E(\beta; \{in^d, out^d\})$

if ($||\beta - \beta nuevo|| < \epsilon$)) $convergido$<- TRUE

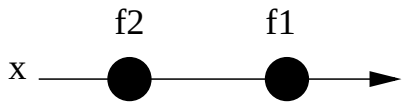
else β<-$\beta nuevo$ }

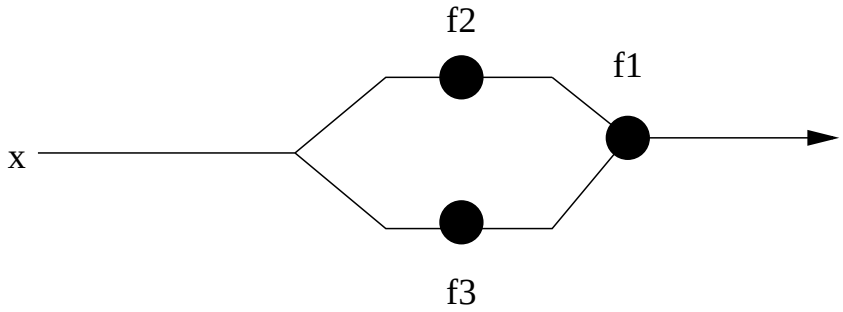
¿ Cómo calculamos $\nabla_\beta E(\beta; \{in^d, out^d\})$?

Recuérdanse: regla de la cadena

1D: Si $f(x) = f_1(f_2(x))$,

$$\frac{df(x)}{dx} = \frac{df_1(y)}{dy}\frac{df_2(x)}{dx} \text{ con } y = f_2(x).$$





2D: Si $f(x) = f_1(f_2(x), f_3(x))$,

$$\frac{df(x)}{dx} = \frac{\delta f_1(y, f_3(x))}{\delta y}\frac{df_2(x)}{dx} + \frac{\delta f_1(f_2(x), z)}{\delta z}\frac{df_3(x)}{dx} \text{ con } y = f_2(x), z = f_3(x).$$

El problema es: ¿ cómo calculamos $\nabla_\beta E(\beta; \{in^d, out^d\})$?

$$E(\beta; \{in^d, out^d\}) = \sum_d \overbrace{\frac{1}{2}||out^d - f_\beta(in^d)||^2}^{E^d} .$$

Es suficiente calcular:

$$\frac{\delta E^d}{\delta \beta_{i,j}} = \overbrace{\frac{\delta E^d}{\delta y_i}}^{\delta_i^d} \frac{\delta y_i}{\delta \beta_{i,j}}.$$

Para calcular δ_i^d usamos una recursión de atrás hacia adelante en la red:

- **Si la neurona i pertenece a la última capa ($f_\beta(in^d)_i = \mathcal{F}(y_i^d)$):**

$$\delta_i^d = -(out^d - f_\beta(in^d))\acute{\mathcal{F}}(y_i^d)$$

- **Si no:**

$$\delta_i^d = \frac{\delta E^d}{\delta y_i} = \sum_{\text{hijos } h \text{ de } i} \frac{\delta E^d}{\delta y_h}\frac{\delta y_h}{\delta y_i} \qquad \text{con } \frac{\delta y_h}{\delta y_i} = \frac{\delta y_h}{\delta a_i}\frac{\delta a_i}{\delta y_i}.$$

Entonces:

$$\delta_i^d = \sum_{\text{hijos } h \text{ de } i} \delta_h^d \beta_{h,i} \acute{\mathcal{F}}(y_i^d)$$

Algoritmo de backpropagation:

REPITE para cada dato d:

1. **calcula $f_\beta(in^d)$;**

2. **Para cada capa c de neuronas (de atrás hacia adelante):**

   para cada neurona i de capa c:

   - **Si la neurona i pertenece a la última capa:**

$$\delta_i^d = -(out^d - f_\beta(in^d))\acute{\mathcal{F}}(y_i^d)$$

   - **Si no:**

$$\delta_i^d = \sum_{\text{hijos } h \text{ de } i} \delta_h^d \beta_{h,i} \acute{\mathcal{F}}(y_i^d)$$

   Calcula:

$$\frac{\delta E^d}{\delta \beta_{i,j}} = \delta_i^d \ \frac{\delta y_i}{\delta \beta_{i,j}}.$$

3. **Calcula: $\beta nuevo_{i,j}$ <- $\beta_{i,j} - \eta \frac{\delta E^d}{\delta \beta_{i,j}}$**

Ejemplo en R

Las redes en R son de dos capas pero las codifican en tres númeritos: $dx - size - da$
El usuario determina el número (=size) de neuronas en la capa oculta; el input y output determina automáticamente dx y dy.
Por default, $\mathcal{F}$ es la sigmoide.

```
library(nnet)
d<-data.frame(x,y)
names(d)<-c("X","Y")
n<-nnet(Y~X,size=4,data=d)
pn<-predict(n,d)
var(x-pn[,1])
summary(n)
```

nnet(formula, data, weights, ..., subset, na.action, contrasts = NULL)
nnet(x, y, weights, size, Wts, mask, linout = FALSE, entropy = FALSE, softmax = FALSE, censored = FALSE, skip = FALSE, rang = 0.7, decay = 0, maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000, abstol = 1.0e-4, reltol = 1.0e-8, ...)

```
insertar demonn0.R
```

8.3 Definición y estimación de una RN para clasificación

Adaptamos lo anterior para clasificación.

Supongamos que los datos son de la forma$\{(in^d, out^d)\}$ con:

$$out^d = (1,0), \text{ si } d \text{ es de categoría 0}, \quad out^d = (0,1), \text{ si } d \text{ es de categoría 1}.$$

Mapeamos las salidas $\{a_i\}$ a $\{p_i\}$, con

$$p_i = \frac{exp(a_i)}{\sum_j exp(a_j)}, \text{ formando una distribución de probabilidad.}$$

Cambiamos la función de costo en por ejemplo menos la logverosimilitud:

$$EC = \sum_d \overbrace{\sum_i out_i^d \log p(in^d)_i}^{EC^d}.$$

Para poder aplicar el método del gradiente necesitamos poder calcular:

$$\frac{dEC^d}{dp_i}, \frac{dp_i}{da_j}, \frac{da_k}{d\beta_{i,j}}$$

En general: para clasificación en k categorías, usamos la codificación de out^d en vectores de longitud k de tipo: $(0, \cdots, 0, 1, 0, \cdots, 0)$.

En $\mathcal{R}$, se indica con `softmax=T` que se anade esta capa adicional. `insertar demonn.R`

8.4 Regularización

El algoritmo de ajuste no siempre converge, ni es siempre estable.

$\Rightarrow$ incluir un término de regularización en la función de costo; por ejemplo:

$$E(\beta; \{in^d, out^d\}) + \lambda ||\beta||^2, \text{ con } \lambda > 0$$

El gradiente es

$$\nabla E(\beta; \{in^d, out^d\}) + 2\lambda\beta.$$

Para tratar todas las variables de la misma manera: hay que estandarizarlas.

¿Cuál es el efecto de aumentar λ ?

Otro problema: el método del gradiente solamente converge a un óptimo local.

$\Rightarrow$ arrancar varias veces con otra inicialización.