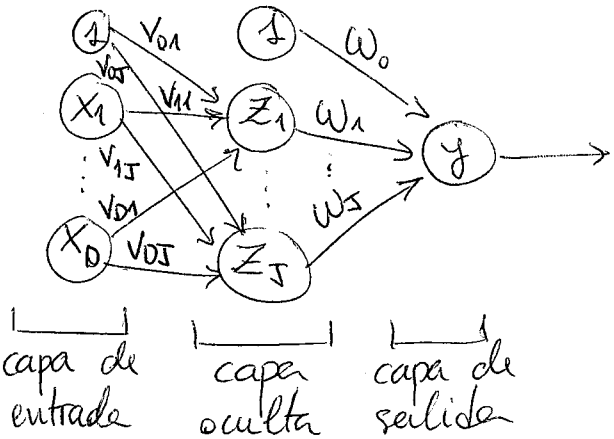
RESUMEN PARCIAL 4

REDES NEURONALES

▶ Red general de 3 capas



V_{dj} := peso entre la neurona x_d (entrada) y z_j (oculta)

w_j := peso entre neurona z_j (oculta) y la capa de salida (neurona y)

núm. de params: $\underbrace{(D+1)J}_{\text{matriz } V=(V_{dj})} + \underbrace{J+1}_{\text{vector } W=(w_j)}$

- **Forward prop:**

$$z_j = \sigma\left(\sum_{d=0}^{D} x_d V_{dj}\right)$$

vale la sigmoidal → en la última capa mejor no o combinac

$$y(x) = \sigma\left(\sum_{j=0}^{J} w_j z_j\right) = \sigma\left(\sum_{j=0}^{J} w_j \sum_{d=0}^{D} x_d V_{dj}\right)$$

- **Backward prop:**

$$w_i = w_i - \eta\,(y(x_n) - t_n)\,z_i$$

$$V_{pq} = V_{pq} - \eta\,(y(x_n) - t_n)\,w_q z_q (1 - z_q)\,x_{np}$$

- **Algoritmo:**

```
NN_model(η, nepocas):
    V, W ← inicializar pesos aleat. ∈ [-0'5, 0'5].
    for ie = 1 : nepocas:
        for n = 1 : N :
            forward_prop(x_n)
            backward_prop(x_n)
```

obs: primero se actualiza V_{pq} y después w_q.

ALGORITMOS GENÉTICOS

- Algoritmo:

```
AG_Model(función fitness f, params evolución):
    init población aleatoria inicial P
    mientras no se cumpla criterio_fin:
        S = selección_progenitores(P)
        S = recombinación(S)
        S = mutación(S)
        P = selección_supervivientes(P,S)
    return best_individuo
```

función fitness: cuantifica la calidad de una solución (individuo). Debe ser mayor cuanto mejor sea.

selección_progenitores: selección aleatoria con reemplazamiento con tantos progenitores como individuos haya en la población: normalmente se hace selección proporcional a fitness.

recombinación/cruce: se obtienen 2 descendientes del cruce de dos padres.
- → cruce 1 pto.
- → cruce 2 ptos
- → uniforme

mutación: pequeña modificación a algún individuo
- → cambiar valor de un gen binario.
- → ruido blanco (genes continuos)

selección_supervivientes: se copia la generación obtenida en la siguiente + elitismo (pequeño).

▶ Teoría de esquemas:

$O(H) :=$ número de bits definidos esquema H.

$d(H) :=$ longitud esquema $H \equiv$ distancia máx. bits def

$L :=$ longitud cadenas de soluciones.

- Cada cadena def. de longitud L es una instancia de 2^L
- Una población de N individuos contiene instancias de 2^L esquemas (todos iguales) a $N2^L$ esquemas.

$n_H(t) :=$ número instancias esquema H tiempo t.

$f_i(t) :=$ fitness individuo i en tiempo t.

$\bar{f}(t) :=$ fitness medio población tiempo t.

$\bar{f}_H(t) :=$ fitness medio esquema H tiempo t.

$\mathbb{E}[n_H(t+1)] :=$ valor esperado de instancias esquema H tiempo $t+1$

Paso 1: selección proporcional a fitness.

$$\mathbb{E}_S[n_H(t+1)] = n_H(t) \cdot \frac{\bar{f}_H(t)}{\bar{f}(t)}$$

Paso 2: cruce en un punto

$$S_c = 1 - P_c \cdot \frac{d(H)}{L-1}$$

Paso 3: mutación bitflip

$$S_M = (1 - P_M)^{O(H)}$$

Paso 4:

$$\mathbb{E}[n_H(t+1)] = n_H(t) \frac{\bar{f}_H(t)}{\bar{f}(t)} \cdot \left(1 - P_c \frac{d(H)}{L-1}\right)(1 - P_M)^{O(H)}$$

ÁRBOLES DE DECISIÓN

▶ Observaciones:

- Inviable encontrar una solución óptima (maldición de la dimensionalidad) ⟹ debemos tener sesgos ⟹
  ⟹ construimos el árbol paso a paso, escogiendo la mejor división en cada paso (CRITERIO DE DIVISIÓN)
- El árbol crece hasta que se cumple el CRITERIO DE PARA
- El árbol se poda (CRITERIO DE PODA) para evitar 'overfitting

▶ CRITERIO DE DIVISIÓN: CRITERIO DE GINI

Debemos medir la impureza del nodo.

Impureza Gini: $$i(t) = \sum_{i=1}^{K} f_i(1-f_i)$$

f_i = fracción ejemplos clase i nodo
(K = número de clases total

Variación de impureza tras una división:

$$\Delta i(t,s) = i(t) - P_L\, i(t_L) - P_R\, i(t_R)$$

P_L → proporción de ejemplos que van al izdo.

P_R → " " " " dcho

▶ CRITERIO DE PARADA:

- todos los ejemplos clasificados o no hay nuevas divisiones p
- el nº de ejemplos de un nodo es "muy pequeño".
- la ganancia de impureza es "pequeña".
- profundidad máxima.

} pre-pod

▶ PODA: evitar "overfitting"

↓ Criterio de poda: $$R_\alpha(t) = R(t) + \alpha\, C(t)$$

$R \equiv$ error del árb con raíz en

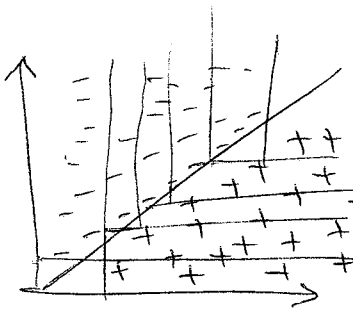
$C \equiv$ número de hojas desde

α = peso precisión-complejidad

↓
CART

- **HIPERPARÁMETROS**
  - número mínimo de ejemplos para dividir una hoja
  - aplicar o no poda
  - límite de profundidad (si la hay) o número máximo de nodos del árbol.

- **DESVENTAJAS ÁRBOLES**



  - no manejan bien interacciones complejas entre atributos.
  - Falta de poder expresivo
  - problema de replicación: mismo subárbol en partes diferentes.

- **VENTAJAS ÁRBOLES**
  - Fáciles de entender por no expertos. Se pueden convertir en reglas $\Rightarrow$ interpretables.
  - manejan atributos nominales y numéricos
  - gestionan bien atributos no informativos o redundantes.
  - pueden trabajar con missing values.
  - método no paramétrico. No hay una idea predefinida sobre el concepto de aprender.
  - pocos hiperparámetros.

CONJUNTOS DE CLASIFICADORES

Tma de Condorcet: °jurado con errores indeps y $< 50\% \Rightarrow$
$\Rightarrow$ voto por mayoría $\longrightarrow$ 100% tamaño jurado $\longrightarrow \infty$

▶ DESVENTAJAS

- más lento que un clasificador único ya que hay que crear cientos o miles de clasificadores.
- se pierde interpretabilidad de los árboles.

▶ VENTAJAS

- familia de alg. con mejor rendimiento actualmente.
- los conjuntos basados en aleatorización son alg. sin prácticamente hiperparámetros que ajustar.
- si son árboles, se crean y clasifican muy rápido.

BAGGING

Dataset $L: (x_i, y_i)$ $i = 1, \longrightarrow, N$
Ensemble size T
for t=1 to T:
    sample = BootstrapSample(L)
    h_t = TrainClf(sample)

output:

$$H(x) = \underset{j}{\operatorname{argmax}} \left(\sum_{t=1}^{T} \mathbb{1}_{\{h_t(x) = j\}} \right)$$

↑ voto por mayoría

BOOSTING

Dataset $L: (x_i, y_i)$ $i = 1, \longrightarrow, N$
Ensemble size T
asignar pesos ejemplos 1/N
for t=1 to T:
    h_t = buildClf(L, pesos)
    e_t = weightedError(L, pesos)
    if $e_t == 0$ or $e_t \geq 0'5$: brea
    dividir pesos ejemplos mal: 2e
    dividir pesos ejemplos bien: 2(1-

output

$$H(x) = \underset{j}{\operatorname{argmax}} \left(\sum_{t=1}^{T} \log\left(\frac{1-e_t}{e_t}\right) \mathbb{1}_{h_t(x) = j} \right)$$

↑ voto ponderado