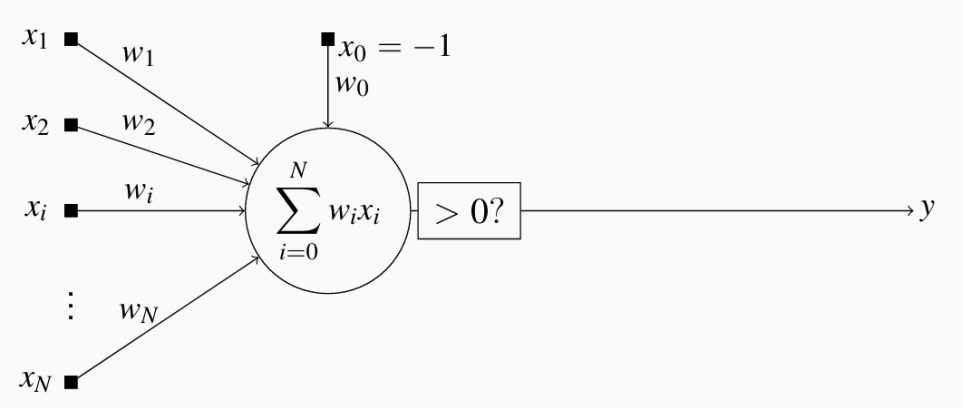
INTELIGENCIA COMPUTACIONAL

Giuliana Cagnola - 2025

INTELIGENCIA COMPUTACIONAL - versión web

TEMA 1: Perceptrón simple



Componentes:

- Entradas X
- Pesos sinápticos W
- Función de activación Φ
  - Signo
  - Lineal
  - Sigmoidea
  - Gaussiana
- Salida lineal Z=X∗W
- Salida de la función de activación Y=Φ(Z)
- Salida correcta D

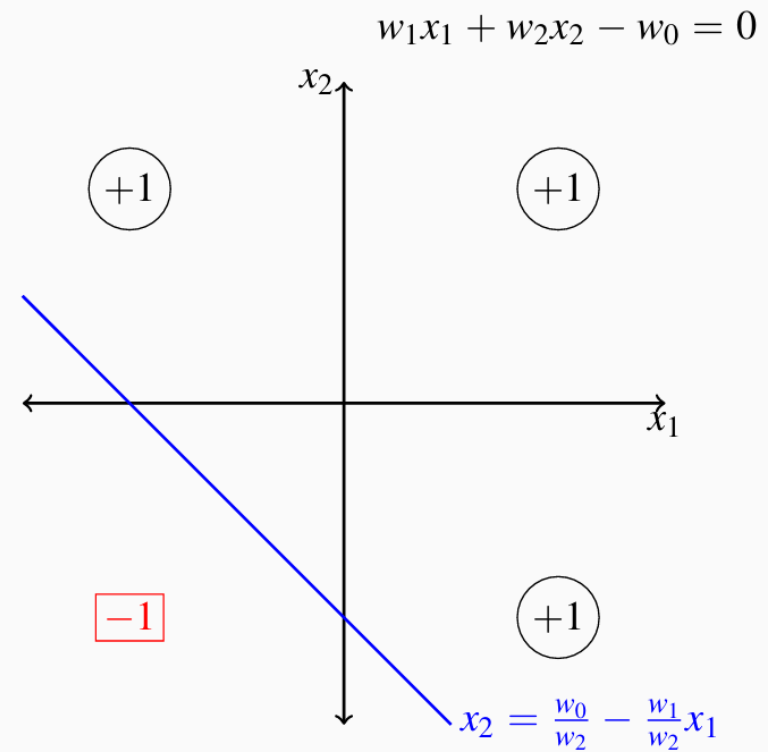
Aprendizaje:

1. Inicializar pesos al azar: W(0) = U(-0.5, 0.5)
2. Calcular salida Y(n) = Φ(W∗X(n))
3. Ajustar pesos
   a. si Y(n)=D(n) → W(n+1) = W(n)
   b. si Y=!D → W(n+1) = W(n) η*(D(n)-Y(n))
4. Repetir 2 hasta que el error sea 0

Método del gradiente: Mover los pesos en la dirección en la que se reduce el error, es decir, en la dirección opuesta al gradiente respecto de los pesos

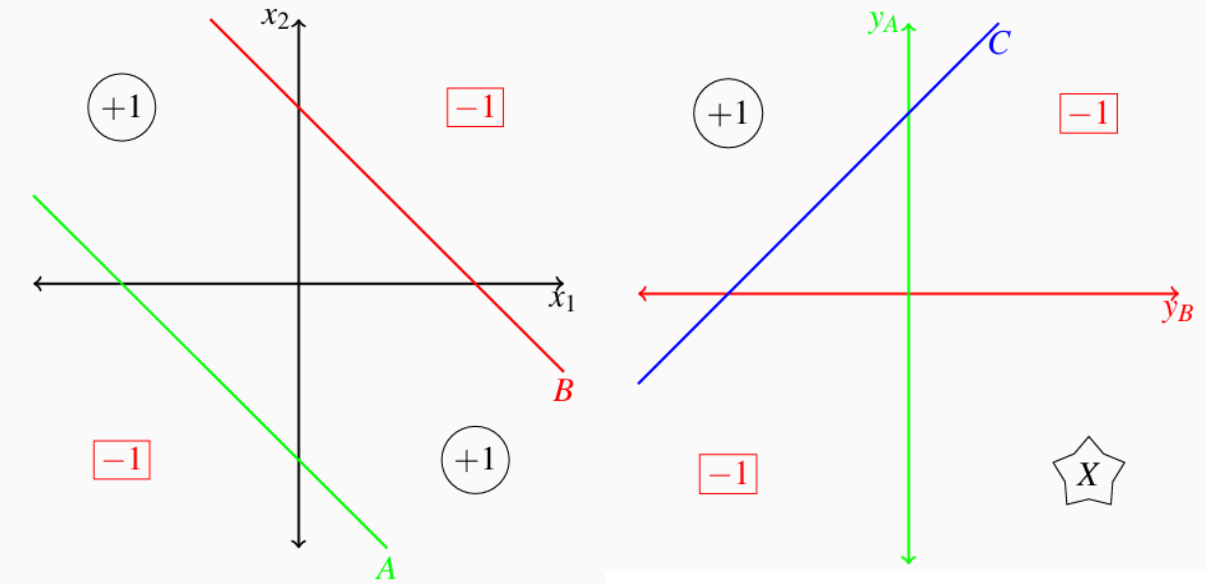
W(n+1) = W(n) + ηΛwE(W(n)): El peso en la próxima iteración W(n+1) es igual al peso actual W(n) más una tasa de aprendizaje η (escala qué tan rápido/lento se mueven los pesos) por el gradiente del error respectto de los pesos ΛwE(W(n))

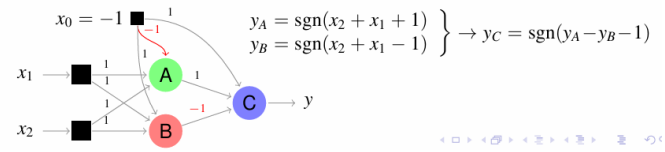
El perceptrón simple (SP) solo puede resolver problemas linealmente separables (OR, AND) separando el plano mediante una recta, el espacio mediante un plano, o el hiperespacio mediante un plano en el caso general



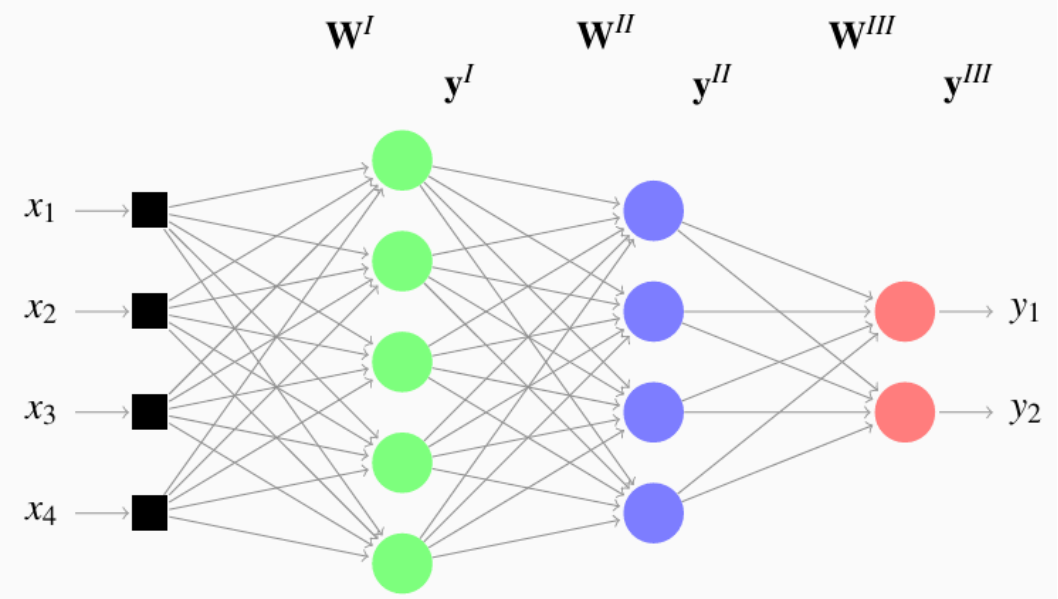
TEMA 2: Perceptrón Multicapa

Problema XOR con 3 neuronas:





Arquitectura del MLP



El perceptrón multicapa (MLP) puede resolver problemas que no son linealmente separables (por ejemplo el XOR). Se compone de entradas X, capa de entrada, k capas ocultas y capa de salida. Entre cada capa se tiene la matriz de pesos sinápticos Wk que conecta la capa k-1 con la capa k

$$\Delta w_{ji}^{(p)}(n) = \eta \left\langle \delta^{(p+1)}, \mathbf{w}_j^{(p+1)} \right\rangle (1 + y_j^{(p)}(n))(1 - y_j^{(p)}(n)) y_i^{(p-1)}(n)$$

Ecuación del delta de pesos para la capa p mediante retropropagación

ENTRENAMIENTO

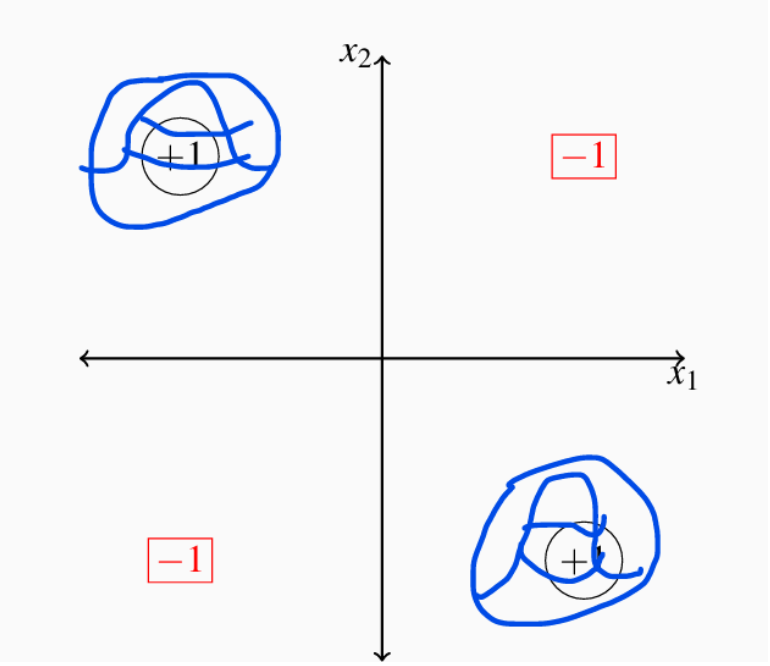
- Para cada época
  - Para cada patrón
    - Forward → e=Yd-Y

- Backward →
- Ajustar pesos → W(n+1)=W(n)+ ΛW

◦ Para cada patrón

- Forward
- Medir desempeño → ECT - Y=Yd

TEMA 3: Funciones de base radial (RBF)



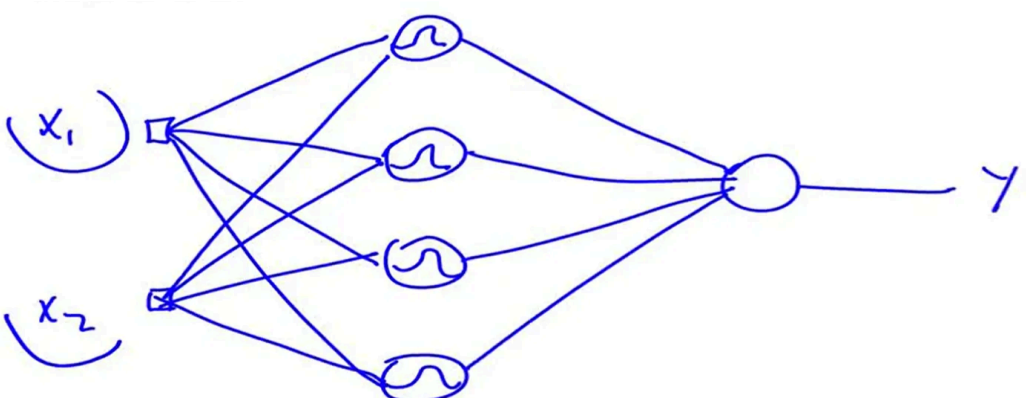
Problema XOR resuelto con regiones radiales

Modelo matemático

$$y_k(\mathbf{x}_\ell) = \sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}_\ell)$$

donde:

$$\phi_j(\mathbf{x}_\ell) = e^{-\frac{\|\mathbf{x}_\ell - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}}$$



ENTRENAMIENTO

- Parte 1: no supervisado
  - Obtener μ (centroide) y σ (desviación)
    - k medias por lote
    - k medias online
- Parte 2: supervisado
  - Obtener W → LMS

Objetivo: Encontrar k conjuntos Cj de forma que cada conjunto Cj sea lo más diferente posible de los demás y los elementos XI que pertenecen al conjunto Cj sean lo más parecidos posible entre sí, y encontrar el centroide μj de cada Cj

Ecuación de optimización: $\min \left\{ J = \sum_{j=1}^{k} \sum_{\ell \in C_j} \|\mathbf{x}_\ell - \boldsymbol{\mu}_j\|^2 \right\}$

ALGORITMO K MEDIAS - LOTE

1. Inicializar k conjuntos al azar: Cj(0)=U(XI)
2. Calcular centroides: μj=ΣXI/|Cj(n)|
3. Reasignar cada patrón a su centroide más cercano: I pertenece a Cj si y solo si |XI-μj|^<|XI-ui|^2
4. Repetir hasta que no se realicen más reasignaciones

ALGORITMO K MEDIAS - ONLINE

1. Seleccionar k patrones al azar como centroides iniciales: μj(0)=U(XI)
2. Seleccionar centroide ganador μj para cada patrón XI: j* = arg min(|XI-μj|)
3. Ajustar centroide por método del gradiente: μj(n+1)=μj + η(XI-μj(n))
4. Repetir hasta que no haya mejoras para J

$$w_{kj}(n+1) = w_{kj}(n) - \eta \left(\sum_i w_{ki}(n)\phi_i(n) - d_k(n) \right) \phi_j(n)$$

Ecuación de ajuste de pesos para la capa lineal

Gaussianas n-dimensionales

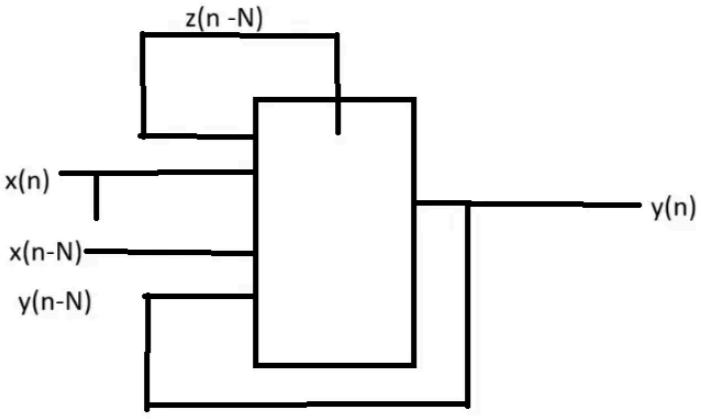
Forma general $\rightarrow \mathbf{x}, \boldsymbol{\mu}_j \in \mathbb{R}^N, \mathbf{U}_j \in \mathbb{R}^{N\times N}$:

$$\mathcal{N}(\mathbf{x}, \boldsymbol{\mu}_j, \mathbf{U}_j) = \frac{1}{(2\pi)^{N/2}|\mathbf{U}_j|^{1/2}} \cdot e^{-\frac{1}{2}\left[(\mathbf{x}-\boldsymbol{\mu}_j)^T \mathbf{U}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right]}$$

- Vector de entradas X
- Vector de centroides μj
- Matriz de covarianza Uj
- Dimensión N

TEMA 4: Redes neuronales dinámicas (DNN)

Y=f(X(n), Z(n-k), Y(n-k)) → la salida está en función de las entradas X (actuales y anteriores), y se retroalimenta con los estados anteriores Z y las salidas anteriores Y



Clasificación

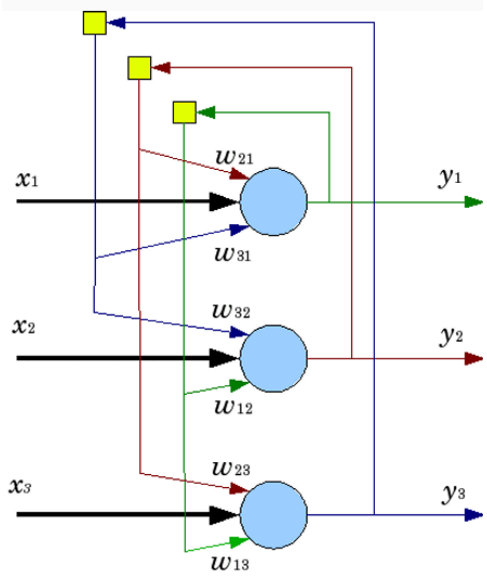
1. DNN
    a. TDNN (redes neuronales con retardos en el tiempo)
    b. RNN (redes neuronales recurrentes)

i. Totalmente recurrentes
    1. Hopfield → memorias asociativas
ii. Parcialmente recurrentes
    1. BPPT (retropropagación a través del tiempo)
    2. Elman
    3. Jordan

1.a - TDNN

- **Entrada con Ventana Temporal:** La entrada a la red no es solo el dato actual X(n), sino una ventana de datos que incluye la muestra actual más un número predefinido de muestras pasadas X(n-1), X(n-2), ..., X(n-N)
- **Compartición de Peso:** Las neuronas en una misma capa que operan sobre diferentes retardos de tiempo comparten los mismos pesos. Esto asegura que la red pueda reconocer un patrón particular sin importar dónde ocurra en la ventana de tiempo.

1.b.i.1 - Hopfield



- Igual cantidad de entradas, neuronas y salidas
- La salida de cada neurona se realimenta como entradas a todas las demás neuronas, menos a sí misma

- Son memorias asociativas (accesibles por contenido): dado un patrón de entrada, seleccionan el patrón del conjunto de memorias fundamentales más parecido
- Las neuronas tienen disparo probabilístico
- El entrenamiento es no supervisado
- Se pueden obtener estados espurios y oscilaciones

$$y_j(n) = sgn(x) = sgn\left(\sum_{i=1}^{N} w_{ji} y_i(n-1) - \theta_j\right) \cdots \begin{cases} x > 0 & +1 \\ x = 0 & y_j(n-1) \\ x < 0 & -1 \end{cases}$$

$$w_{ji} = w_{ij} \quad \forall\, i \neq j$$
$$w_{ii} = 0 \quad \forall i$$

ENTRENAMIENTO (almacenamiento)

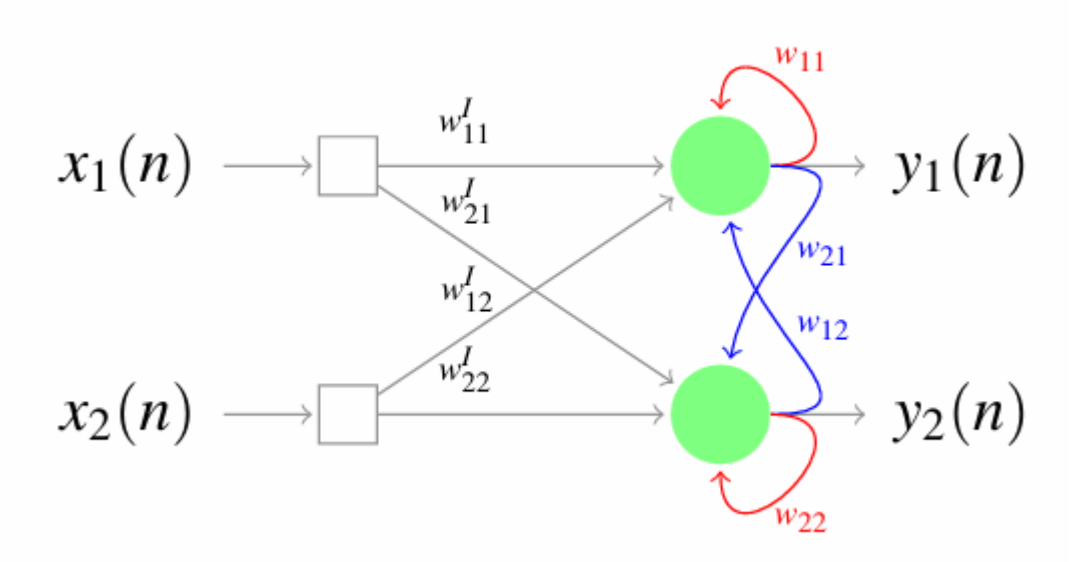
- No iterativo
- Si dos neuronas se activan en simultaneo, la conexión entre ellas se fortalece
- Wji = sum(Xkj*Xki)/N

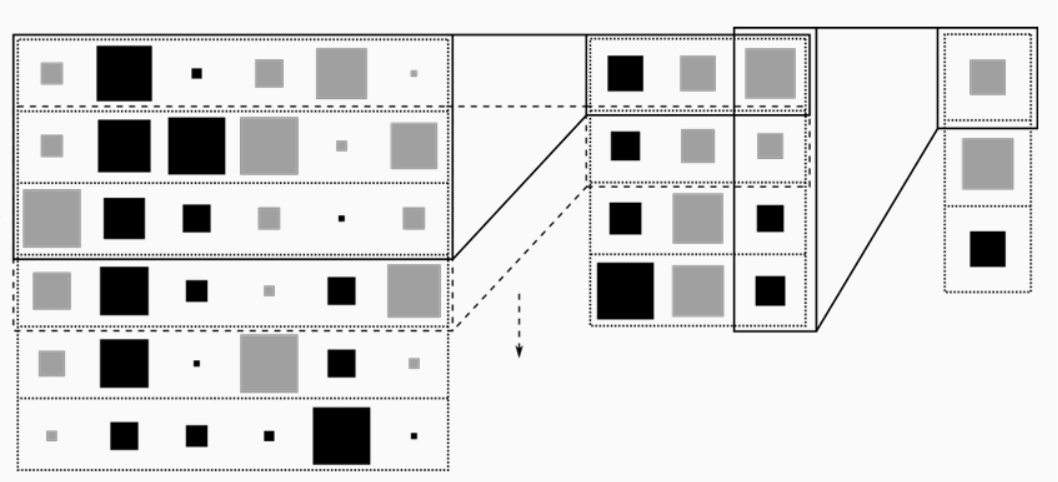
PRUEBA (recuperación)

- Iterativo

1. Y(0)=X
2. j* = rnd(N)
3. Yj*(n)=sgn(sum(Wji*Yi(n-1))
4. Repetir hasta estabilizar la salida (la red convergió a una memoria fundamental)

1.b.ii.1 - BPPT





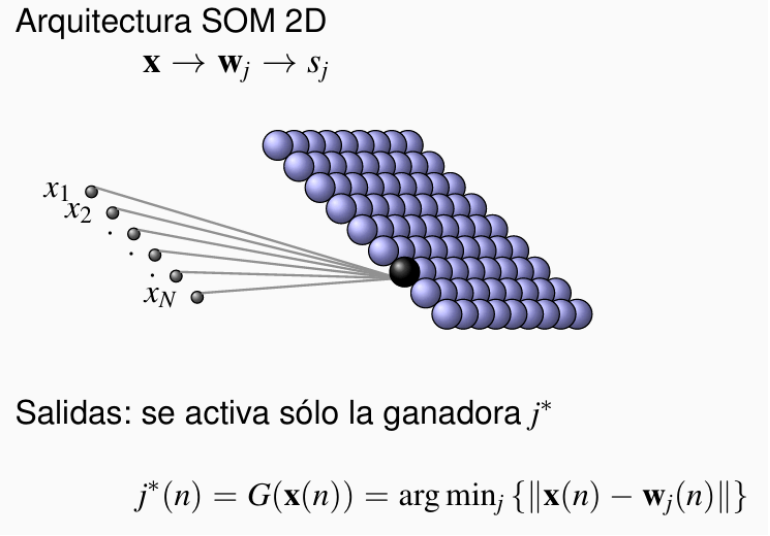
Memoria de largo, mediano y corto plazo

- Se pasa de una arquitectura totalmente recurrente a una arquitectura feed forward. La red se desenvuelve en el tiempo. Luego se aplica BP
- Tiene el problema de desvanecimiento del gradiente
- Los pesos son los mismos en cada paso de tiempo

TEMA 5: Mapas autoorganizativos (SOM)

- Autoorganización: Proceso por el cual se obtiene ordenamiento global a partir de interacciones locales
- El entrenamiento es no supervisado (mediante aprendizaje competitivo)

- Los datos que están cercanos en el espacio original se mapean a neuronas que están cercanas en el mapa (preservación topológica)



ENTRENAMIENTO

1. Inicializar pesos aleatoriamente: Wji=U(-0.5, 0.5)
2. Seleccionar neurona ganadora: G(X)=arg min_j(X(n)-Wj(n))
3. Ajustar pesos:
    a. W(n+1)=W(n)+ μ(n)((X(n)-Wj(n)) si Yi pertenece a la vecindad
    b. W(n+1)=W(n) si Yi no pertenece a la vecindad
4. Repetir hasta que no haya reajuste de pesos

Etapas del entrenamiento:

1. Ajuste global
    a. Vecindad grande
    b. Tasa de aprendizaje grande
2. Transición
    a. Disminuir linealmente la vecindad hasta 1
    b. Disminuir exponencialmente la tasa de aprendizaje
3. Ajuste local

a. Vecindad 0 (solo se ajusta la neurona ganadora)
b. Tasa de aprendizaje pequeña

SOM como clasificador:

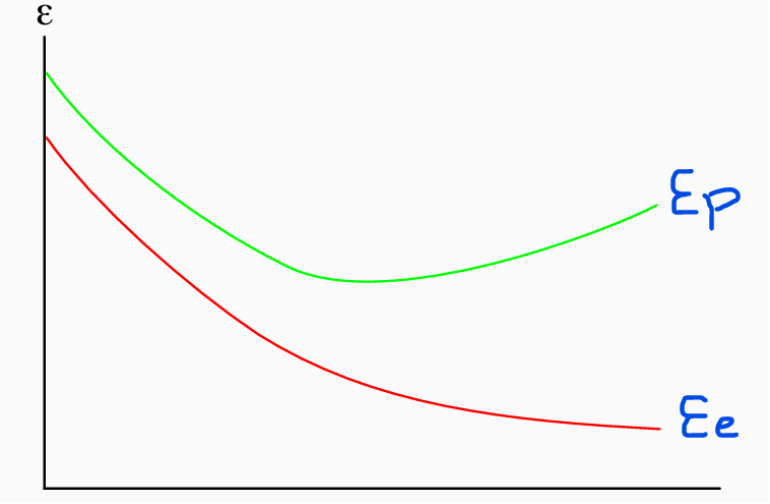
1. Entrenamiento no supervisado
2. Etiquetado de neuronas (supervisado)
3. Clasificación por mínima distancia

LVQ → Cuantización vectorial con aprendizaje

- Se tiene un conjunto de vectores prototipo con su etiqueta de clase
- Para clasificar un nuevo patrón se asigna según la mínima distancia entre el patrón de entrada y el conjunto de vectores prototipo

---

TEMA 6: Capacidad de Generalización



Error en el entrenamiento Ee y error en la prueba Ee vs cantidad de épocas

- Capacidad de generalización: El algoritmo debe tener un buen desempeño frente a datos nunca antes vistos
- Sobreentrenamiento: El modelo captura detalles específicos de los datos de entrenamiento que no son generales al conjunto de datos (ruido o señales

incorrectas)

Validación cruzada (k fold): Subdividir el dataset en k subconjuntos, entrenar con k-1 subconjuntos y usar el restante para validar. Hacerlo para diferentes divisiones, y calcular el error y la desviación promedio para obtener un valor lo menos sesgado posible

Medidas de desempeño:

- Matriz de confusión: clases reales vs clases predichas
- Accuracy: a=(tp+tf)/N → casos bien clasificados sobre el total de casos
- F1 score: F1=2tp/(2t´p+fp+ff) → combina sensibilidad y presición

---

TEMA 7: Aprendizaje Automático

K-means

- Algoritmo no supervisado para dividir al conjunto de datos en k subconjuntos, en donde cada punto pertenece a la clase de su centroide más cercano, y los centroides se ajustan iterativamente hasta estabilizarse
- Se busca minimizar la suma de las distancias cuadradas de los puntos xi a los centroides ci
- Clasificador KNN: Un dato es clasificado por la mayoría de votos entre sus vecinos

Árbol de clasificación

- Modelo de regresión que divide recursivamente el espacio de datos mediante preguntas “si ... entonces ...”
- Se elige la mejor variable y el punto de corte para dividir ortogonalmente a los datos

Naive Bayes

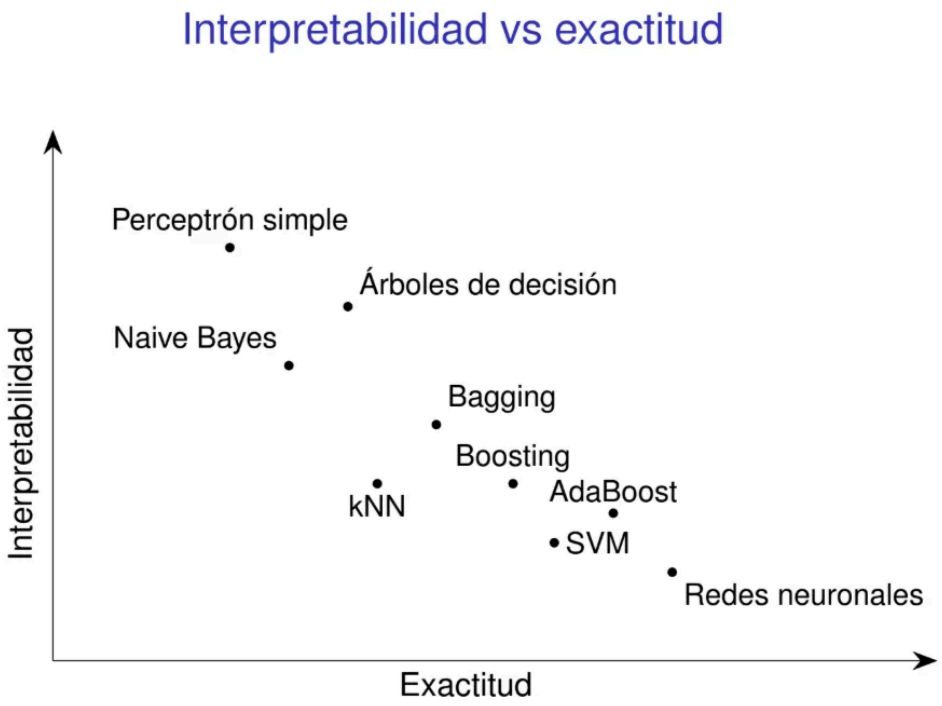
- Clasificador probabilístico basado en el teorema de bayes: $P(C/X)=P(X/C)*P(C)/P(X)$
- Asume (erróneamente) que los atributos son independientes entre sí

Máquinas de soporte vectorial

- Algoritmo supervisado que busca encontrar el hiperplano que mejor separa los datos de diferentes clases
- El margen de separación entre los puntos más cercanos entre sí de cada clase (vectores de soporte) debe ser máximo
- Si los datos no son linealmente separables se usa un kernel (función de transformación de datos a un espacio de mayor dimensión)

Ensamble de clasificadores

- Se combinan varios modelos simples para obtener un mejor rendimiento que el de cada modelo por separado
- BAGGING: Se entrenan varios modelos con diferentes subconjuntos de datos obtenidos muestras aleatorias con reemplazo (bootstrap), y se promedia (regresión) o se hace voto por mayoría (clasificación)
- STACKING: Se combinan distintos modelos y se lo usa como entrada para un modelo final (metaclasificador) que da la predicción
- BOOSTING: Entrena modelos secuencialmente, en donde el modelo actual se enfoca en los errores que cometieron los modelos anteriores (pondera más a los mejores clasificadores)
- ADABOOST: Se busca prestar más atención a los casos más difíciles (si el patrón se clasifica bien entonces pesa menos, y viceversa) y se hace voto por promedio ponderado



TEMA 8: Inteligencia Colectiva

- **FSM: {X, Y, E, D}** → Se representan mediante grafos
  - X: entradas
  - Y: salidas
  - E: estados
  - D: reglas de transición
    - deterministas
    - probabilísticas

- **CA (autómata celular): {A, T, C}**
  - A: autómatas de estados finitos
  - T: topología (triangular, cuadrada, pentagonal, etc)
  - C: conectividades

- **Agente**: Entidad (real o virtual) capaz de percibir su entorno y modificarlo a partir de un conjunto de reglas y comportamientos
- Agente inteligente → Autónoo: aprende de la experiencia (reactividad) y tiene la capacidad de modificar su comportamiento (proactividad)

- **Sistemas multiagentes**: Conjunto de agentes simples que interactúan entre si y con su entorno para generar comportamientos colectivos emergentes. Son cooperativos y tienen racionalidad social (buscan las acciones que beneficien a todos los individuos, no solo a si mismos)

utilidad esperada = f(utilidad individual) + f(utilidad social)

TEMA 9: Algoritmos Evolutivos

La evolución como un algoritmo

```
Inicializar(Población)
MejorAptitud ← Evaluar(Población)
mientras MejorAptitud < AptitudRequerida
    Progenitores ← SelecciónNatural(Población)
    Población ← ReproducciónVariación(Progenitores)
    MejorAptitud ← Evaluar(Población)
fin
```

Se crea una población de posibles soluciones

Se evalúa la mejor aptitud mediante la función de fitness

Mientras que la mejor aptitud sea menor a la aptitud requerida (criterio de parada del algoritmo): Se seleccionan los progenitores (mejores individuos) (ruleta, ventana, competencias), se reproducen los progenitores para generar los hijos y se aplican operadores de variación (mutación, cruza). Esto amplía el espacio de búsqueda y permite salir de mínimos locales.

Se evalúa la nueva población mediante la función de fitness y se repite el proceso hasta alcanzar el criterio de convergencia

---

Colonias de hormigas y enjambres de partículas

COLONIA DE HORMIGAS (ACO)

1. Inicializar feromonas al azar en cada arista del grafo sigma_ij=rand(0, sigma0)
2. Ubicar n hormigas en el nodo origen
3. Repetir hasta que todas las hormigas sigan el miso camino
    a. Para cada hormiga:
        i. vaciar camino pk(t)=null
        ii. seleccionar próximo nodo según la probabilidad (*)
        iii. agregar un paso (i,j) al camino pk(t)
    b. Calcular la longitud del camino f(pk(t))
    c. Para cada conexión:
        i. evaporar feromonas: sigma(t)=(1-rho)∗sigma(t)
        ii. depositar feromonas_ sigma(t+1)=sigma(t)+ delta_sigma(t) (**)
        iii. Incrementar contador t++
4. Devolver el mejor camino

(*) pk_ij(t) = (sigma^alfa * eta*beta)/(sum(sigma^alfa * eta*beta)

(**) delta_sigma_ij(t) =

Q → uniforme

Q/f(pk(t)) → global

Q/d_ij → local

OPTIMIZACIÓN POR ENJAMBRE DE PARTÍCULAS (PSO)

- Cada partícula representa una posible solución (un punto en el espacio de búsqueda)
- Cada partícula tiene:
  - Posición x
  - Velocidad v
  - Mejor posición personal y
- Todas las partículas tienen la mejor posición global y_prima que es la mejor solución encontrada
- En cada iteración:
  - Las partículas actualizan su velocidad y posición
  - Evalúan la función objetivo
  - Ajustan sus mejores valores locales y globales
  - El enjambre tiende a converger hacia el mejor punto encontrado
- El método global tiene la mejor posición entre todas las particulas, mientras que el método global las de su vecindad

PSOG

1. Inicializar posiciones al azar: x_ki(0) = U(xi_min, xi_max)
2. Repetir hasta alcanzar criterio de convergencia:
   a. Para cada partícula k:
      i. actualizar mejor local (si corresponde): si f(xk(t))<f(yk) → yk=xk(t)
      ii. actualizar mejor global (si corresponde): si f(xk(t))<f(yprima) → yprima=xk(t)

b. Para cada partícula k:

   i. actualizar velocidad: v_ki(t+1) = v_ki(t) + c1*r1*(y-x_ki(t)) + c2*r2*(yprima-x_ki(t)) (*)

   ii. actualizar posición: x_k(t+1) = x_k(t) + v_k(t+1)

3. Devolver la mejor partícula (solución óptima)

(*)

- c1*r1*(y-x_ki(t)) → componente cognitivo
- c2*r2*(yprima-x_ki(t)) → componente social
- c1 → exploración (decreciente)
- c2 → convergencia (creciente)
- r1, r2 = U(0,1) → componentes estocásticas

TEMA 10: Lógica borrosa

Red Neuronal	Memoria Asociativa Borrosa
Aprende pesos mediante entrenamiento numérico	Almacena asociaciones difusas directamente
Usa funciones de activación	Usa operadores lógicos borrosos (min, max, etc.)
Precisa grandes conjuntos de datos	Puede trabajar con reglas lingüísticas
Resultado numérico	Resultado interpretativo y difuso

TP Final

Modelos

AGENT

FOLLOWMOUSESHEPHERD

Modelo alternativo para controlar el pastor mediante el mouse

NNSHEEPHERD

Define la arquitectura de la nn a entrenar con algoritmo genético.

Las entradas de la nn son:

- Posición del pastor
- Objetivo
- Posiciones de las ovejas más cercanas

La salida es un vector dirección que determina el movimiento del pastor

STROMBONSHEEP

Implementa el comportamiento heurístico de las ovejas mediante el modelo de strombon

- Repulsión local (de oveja a oveja)
- Repulsión global (de oveja a pastor)
- Atracción al centro de gravedad (solo cuando está siendo pastoreada)

Scripts

RUNTRAIN

Ejecuta el algoritmo genético y utiliza multiprocesing para evaluar la función de fitness

- Inicializar población al azar
- Seleccionar progenitores (ventana+elitismo)
- Variación: cruza y mutación

RUNSIM

Ejecuta la simulación (el juego)

Sim

INTERFACE

SHEEP

Define la posición, dirección y el modelo de comportamiento de cada oveja. Cada oveja tiene un booleano pastoreada para indicar si está bajo la influencia de un pastor

SHEEPHERD

Define la posición, dirección y el modelo de comportamiento del pastor. Tiene un método update que delega la lógica del movimiento al modelo de IA

WORLD

Inicializa las ovejas con el modelo de strombon y el pastor con la nn o followmouse. Maneja el bucle de tiempo (update), detecta si se cumplió el objetivo (shepherd_finished) y calcula el centro de gravedad de las ovejas (centroGravedadOvejas) para el cálculo del fitness

INTERFACE

Dibuja el mundo y muestra información en la pantalla de simulación

Training

EVALUADOR

Contiene la lógica para evaluar:

1. Decodificar el genoma binario en pesos para la nn
2. Ejecutar la simulación con esos pesos
3. Evaluar la función de fitness: 1/ticks_to_finish + 1/dist**2

Config

CONFIG.YAML

Contiene todos los parámetros del sistema