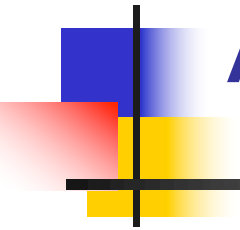
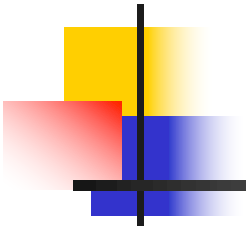


Aprendizaje de Máquina



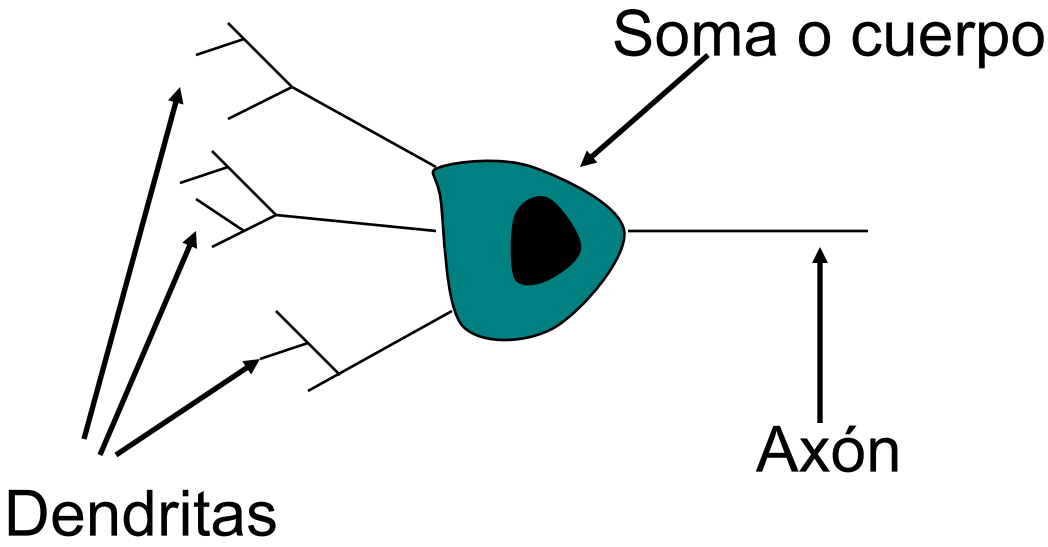
Menu

- Redes Neuronales
  - Inspiración
  - Tipo de redes
    - Acíclicas
    - Cíclicas
  - Modelo de neurona
    - Algoritmo de entrenamiento
  - Redes Acíclicas
    - Algoritmo de entrenamiento de retropopagación

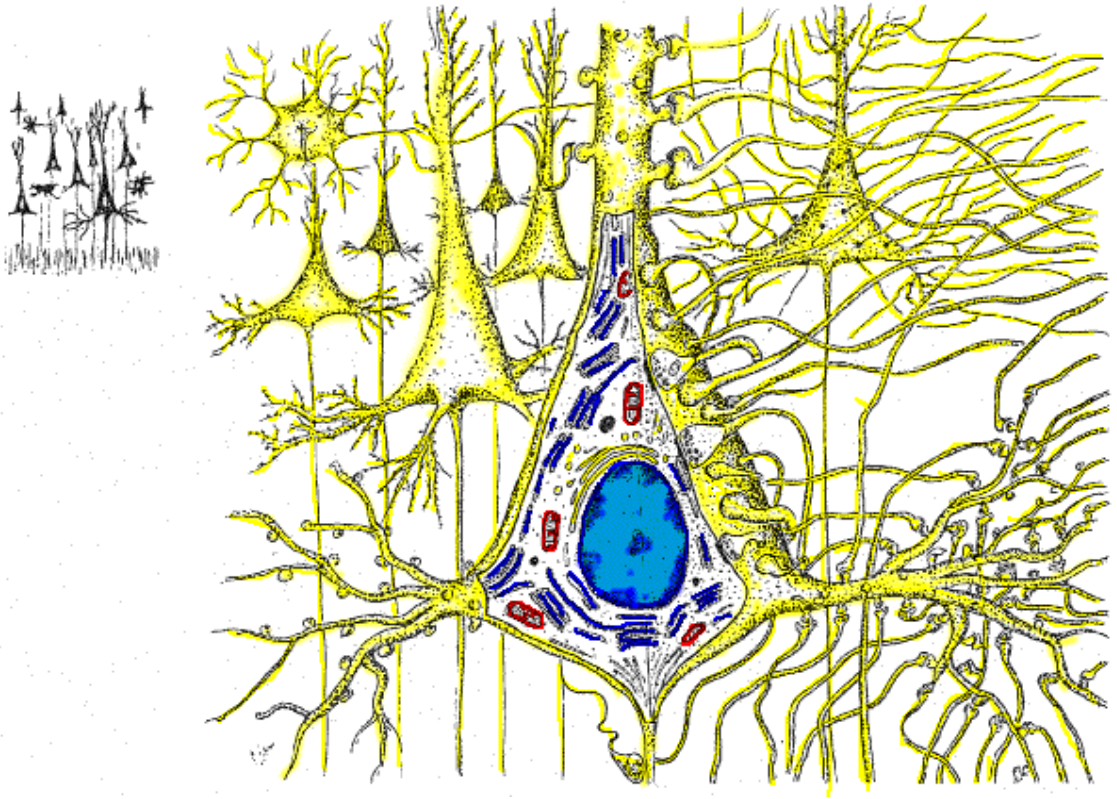
Redes Neuronales Biológicas

- Un ser humano tiene en promedio 10^{11} neuronas
- Cada una conectada con aproximadamente 10^4 otras neuronas
  - ¿Cuántas conexiones?
- Cada neurona se activa o inhibe dependiendo de los estímulos que obtiene de las otras neuronas conectadas a ella

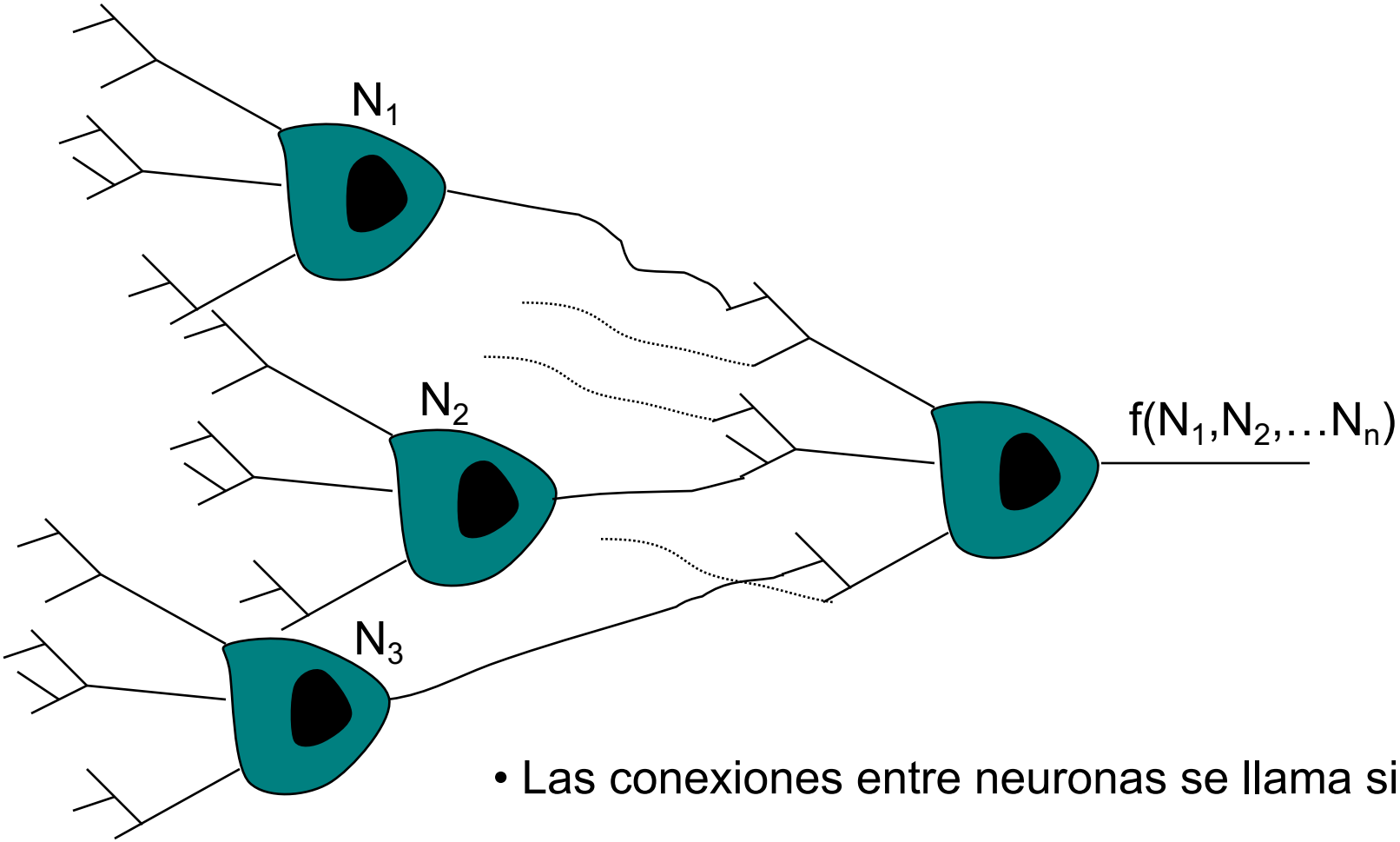
Redes Neuronales Biológicas



Redes Neuronales Biológicas

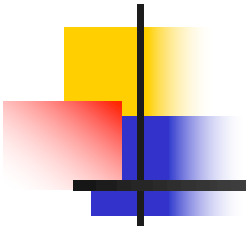


Redes Neuronales Biológicas

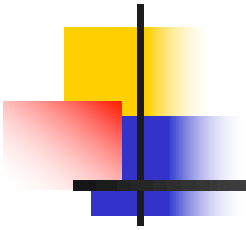


- Las conexiones entre neuronas se llama sinapsis

Redes Neuronales Biológicas



- Las neuronas más rápidas pueden cambiar de señal una vez cada 10^{-3} segundos.
- Un transistor puede hacerlo cada 10^{-10}
- Sin embargo los seres humanos puede tomar decisiones sumamente complejas sumamente rápido
  - ¿Porqué?



Redes Neuronales

- Las redes neuronales artificiales están inspiradas en las redes biológicas, pero son una simplificación de éstas. En particular:
  - Todas las neuronas son iguales (o hay poca variedad en el tipo de neuronas)
  - La operación de una neurona individual es idealizada
  - El número y el tipo de conexiones entre neuronas es mucho más simple que en las redes biológicas

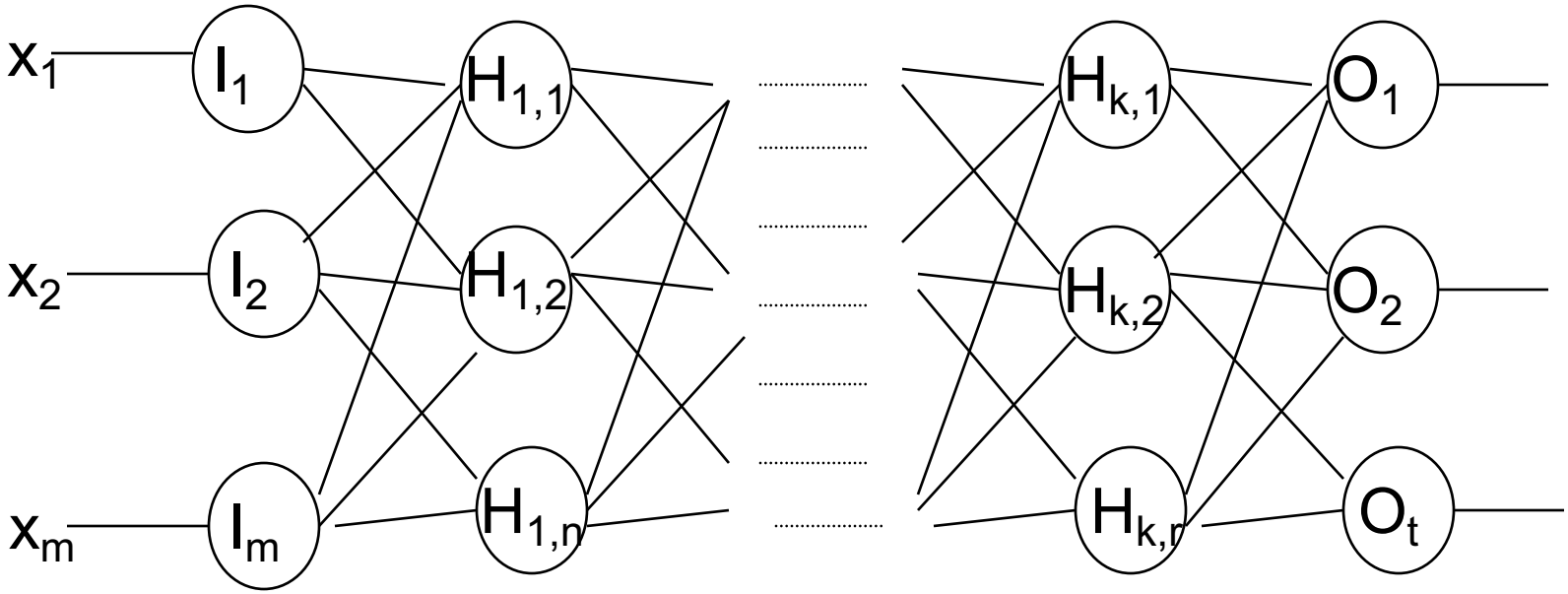
Redes Neuronales Artificiales

- Se utilizan para
  - Aprendizaje supervisado
  - Aprendizaje no supervisado
- Pueden aproximar (aprender) funciones objetivo continuas y discretas
  - Clasificación
  - Regresión
- Útiles para interpretar datos reales obtenidos a través de sensores
  - Manejo automático de coches
  - Reconocimiento de lenguaje escrito, imágenes, voz,….

Redes Neuronales
Topologías

- Tipos de redes
  - Acíclicas
    - Para cualquier neurona en un red acíclica, no existe un camino de su salida a sus entradas
    - Vamos a estudiar las redes feed-forward
  - Recurrentes (cíclicas)
    - En estas redes, cualquier neurona puede estar conectada con cualquier otra. En particular, puede existir un camino de la salida de una neurona a sus entradas
  - Mixtas

Topologías
Red Acíclica "feed forward"

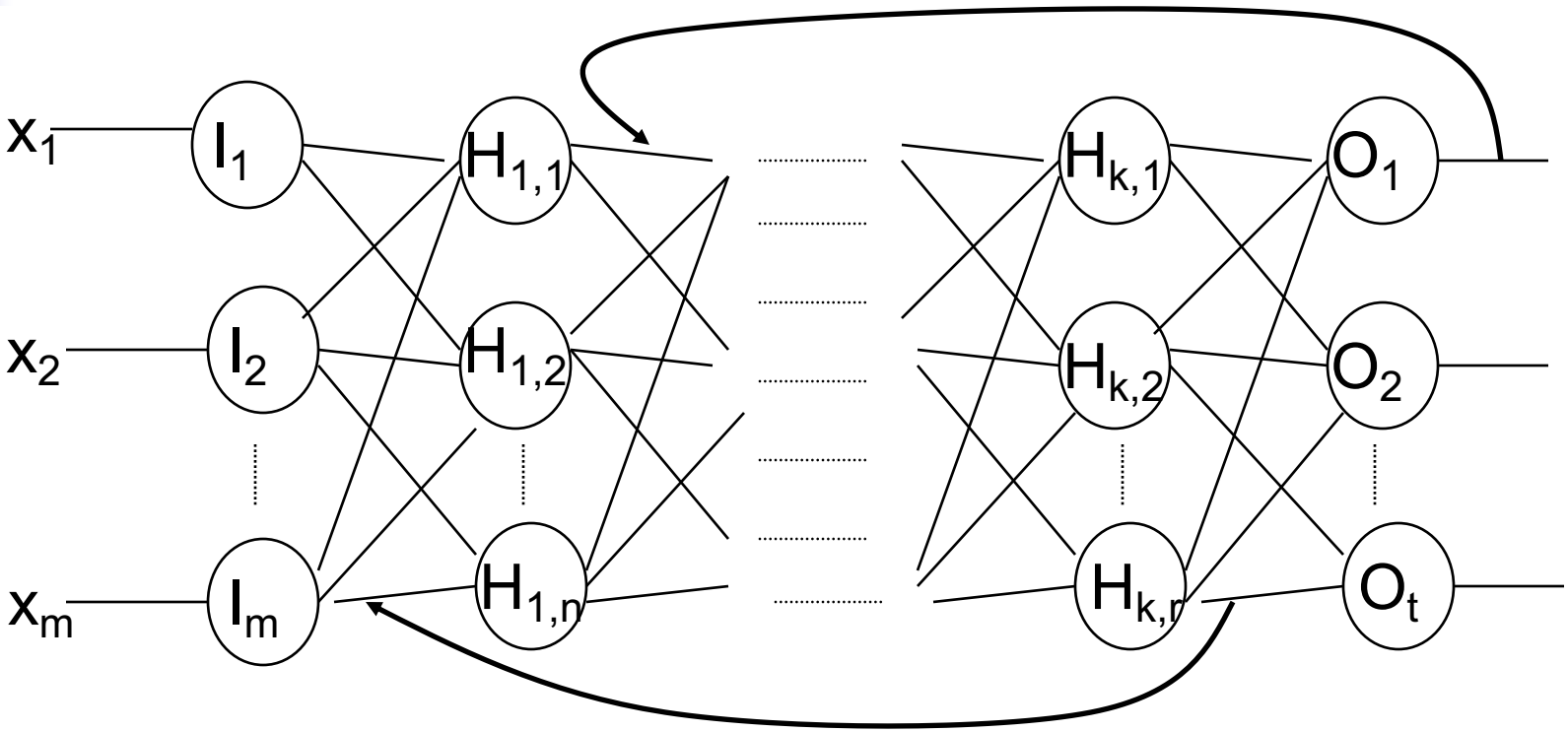


- Hay tres tipos de nivel. El nivel de entrada **I**, los niveles intermedios o escondidos **H** y el nivel de salida **O**. Cada nivel puede tener diferente número de neuronas
- Las neuronas de un nivel sólo se conectan a neuronas del nivel siguiente
- Normalmente todas las neuronas de un nivel se conectan a todas las neuronas del nivel siguiente, pero no necesariamente

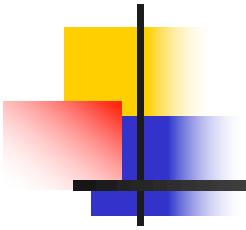
Características
Redes Feed-forward

- Cada neurona del nivel de entrada toma solamente una entrada
- El número de neuronas de entrada y de salida define el problema
- Los pesos y el número de capas ocultas y neuronas en éstas capas son parámetros libres

Topologías
Red Cíclica o Recurrente



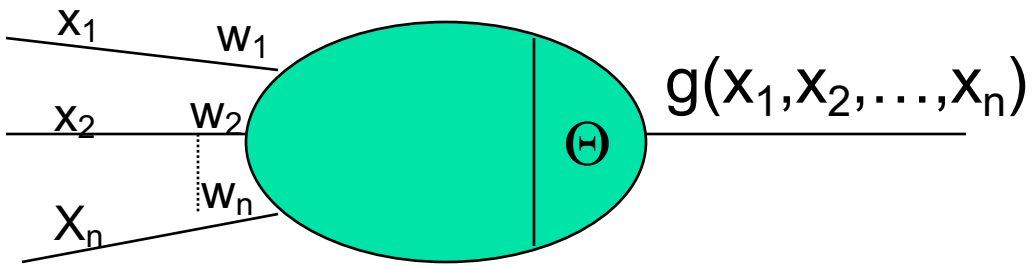
• La redes recurrentes tienen ciclos que provocan que la entrada a una neurona dependa, en parte, de la salida de la misma en otro tiempo. Esto permite que conserven un estado. Crea la posibilidad de tener memoria



Mapa

- Vamos a ver
  - Modelos de neurona
    - Perceptron lineal, escalón y sigmoidal
  - Topología de conexión
    - Feedforward
  - Aprendizaje supervisado
    - Retro-propagación “Backpropagation”

Redes Neuronales
Modelo de Neurona



- Cada neurona tiene un nivel de activación de salida que depende del valor de sus entradas
- Una neurona activada transfiere su señal de acuerdo a una funciones de transferencia g:
  - Lineal
  - Escalón
  - Sigmoidal

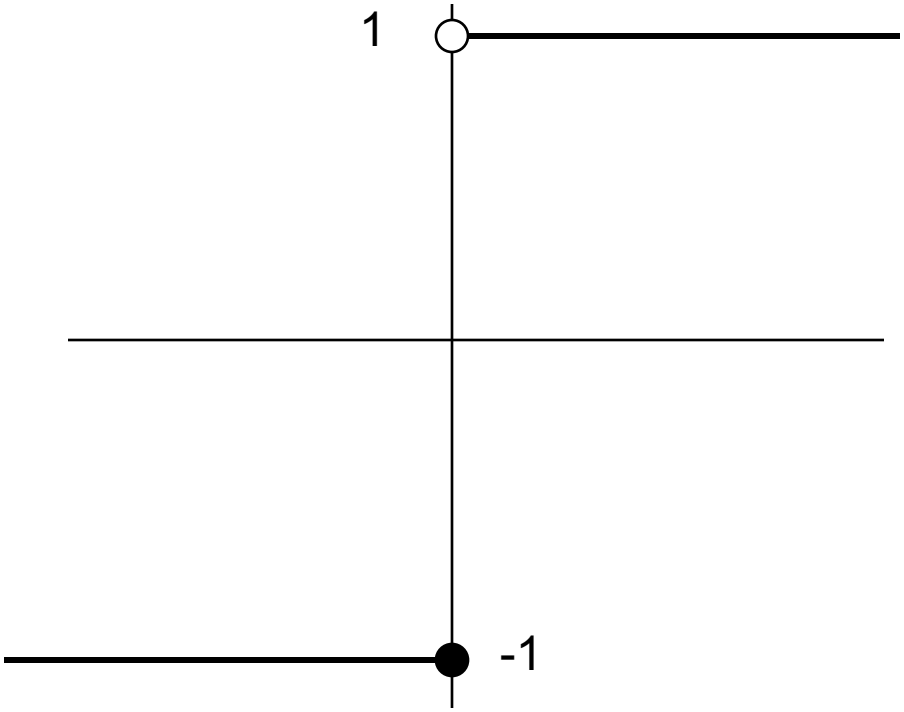
Redes Neuronales
Modelo de Neurona

- La activación de la neurona es una función de la suma ponderada de los pesos de entrada a la neurona
  - Suma ponderada=$\sum w_i x_i$
  - Donde las x_i son las entradas a la neurona y w_i son los pesos que se le dan a cada componente de la entrada
  - La tarea de aprendizaje consiste en ajustar los pesos
  - ¿Qué pasa si la función de transferencia g es simplemente el resultado de la sumatoria?

Redes Neuronales
Modelo de Neurona: Perceptrón

- La función g que representa el nivel de activación del perceptrón es
  - $g(x_1,x_2,..,x_n)= \begin{cases} 1 \text{ si } w_o+\sum_{i=1,n} w_i x_i > 0 \\ -1 \text{ de otra forma} \end{cases}$
  - Existe una variable extra w_o que no depende de ninguna entrada a la neurona. Su función es la de crear un umbral para que la neurona dispare
  - Note que g será 1 sólo si la suma ponderada de las entradas es mayor a $-w_o$.
  - Para simplificar la notación, imaginamos que siempre existe una entrada $x_0=1$ que multiplica a w_o y escribimos la sumatoria como $\sum_{i=0,n} w_i x_i$

Perceptrón
Función de Transferencia:Fn Escalón



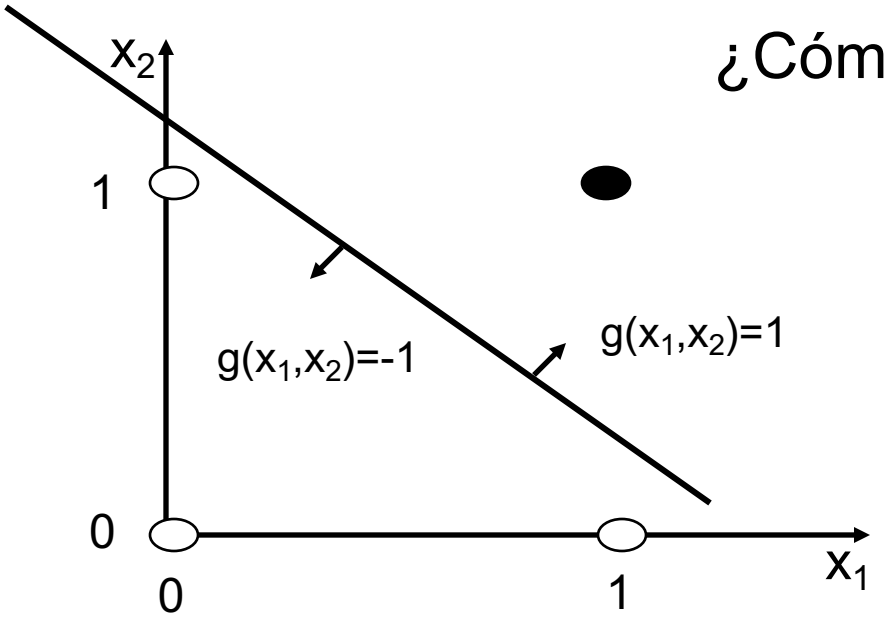
Poder de Representación Perceptrón

- Para ilustrar el poder de representación de este perceptrón graficamos la ecuación

$$\sum_{i=0,n} w_i x_i = 0$$

- Ya que cuando $\sum_{i=0,n} w_i x_i$ es mayor a cero el perceptrón clasifica el ejemplo como 1 y cuando es igual o menor a cero como -1
  - De esta manera $\sum_{i=0,n} w_i x_i = 0$ representa una barrera o línea de decisión

Poder de Representación Perceptrón

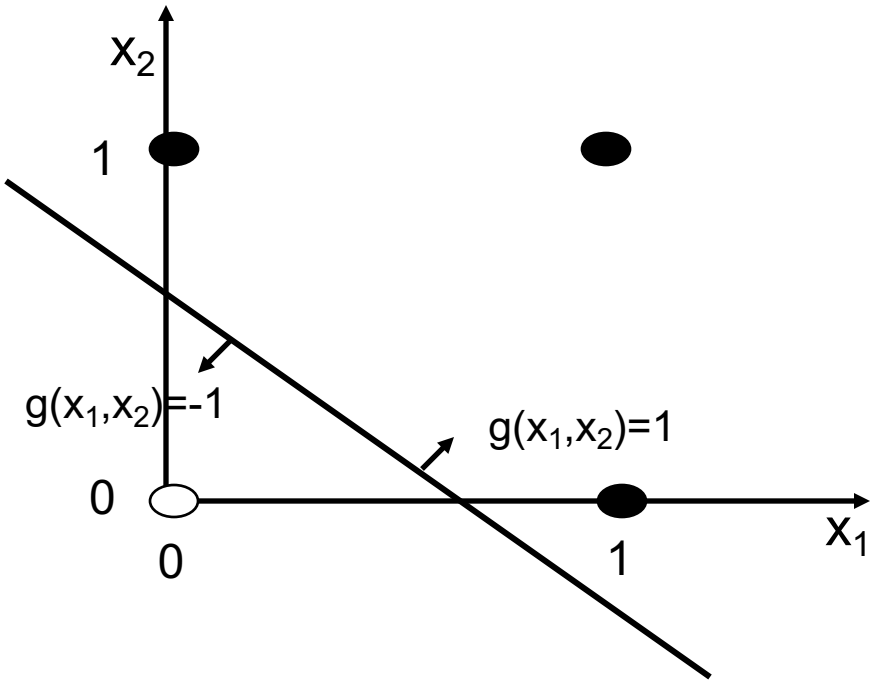


¿Cómo lo entrenamos?

•Los círculos blancos y negros pertenecen a distintas categorías.

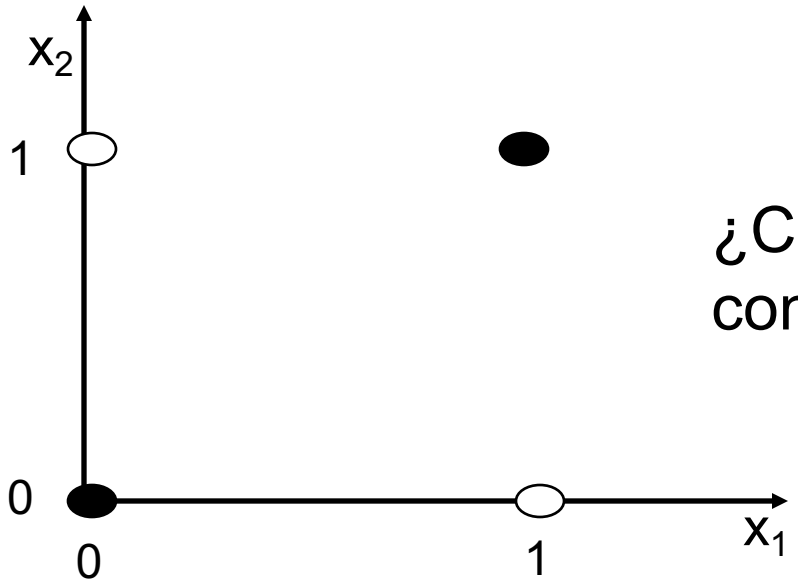
• ¿Qué función es esta?

Poder de Representación Perceptrón Lineal



- ¿Qué función es esta?

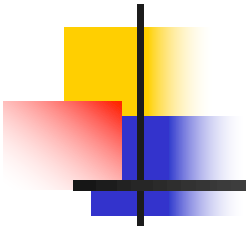
Poder de Representación Perceptrón



¿Cómo separamos con una línea?

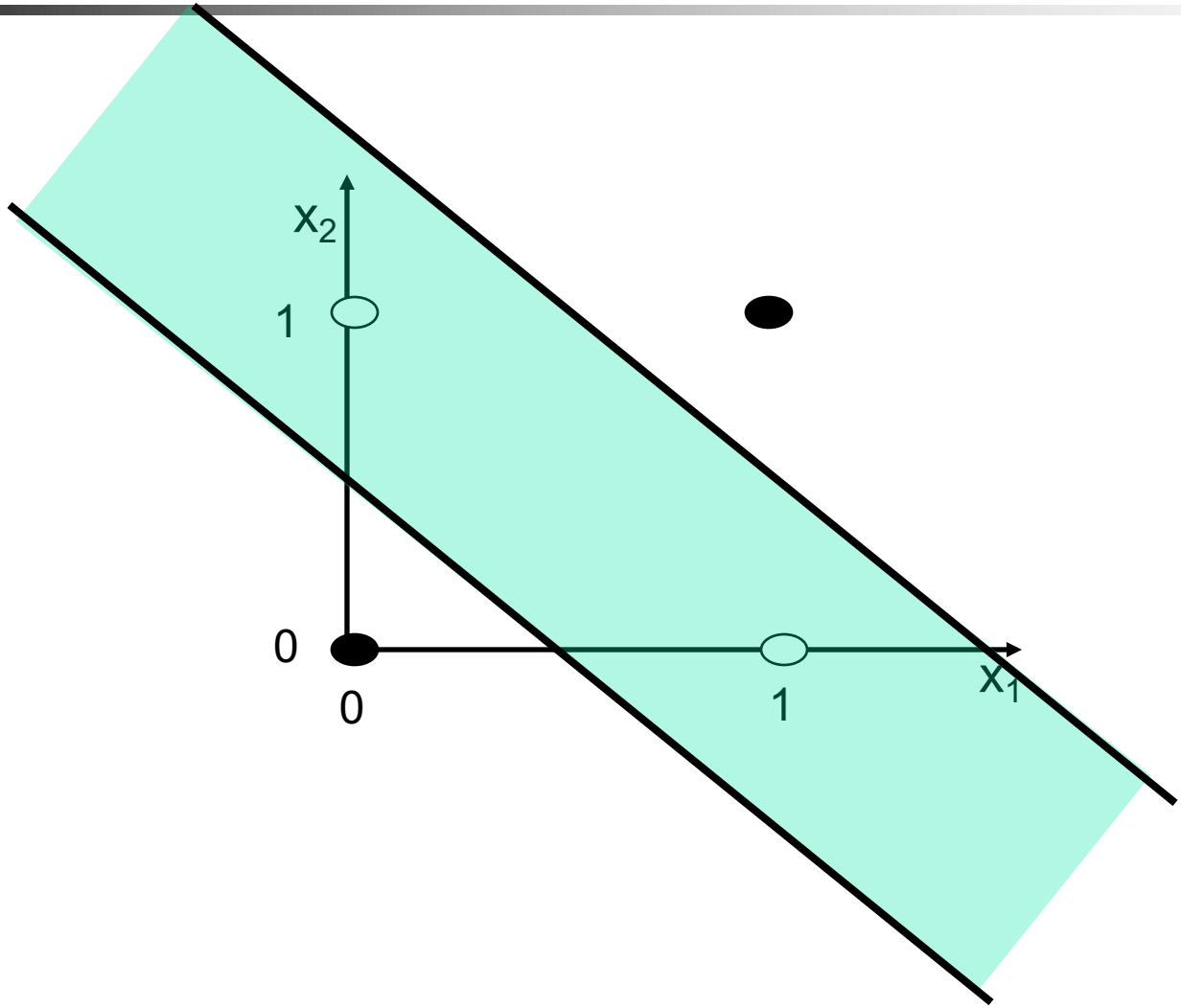
- ¿Qué función es esta?
- Minsky y Papert señalaron esta limitación y su descubrimiento significó un retraso en el desarrollo de esta tecnología

Poder de Representación Perceptrón

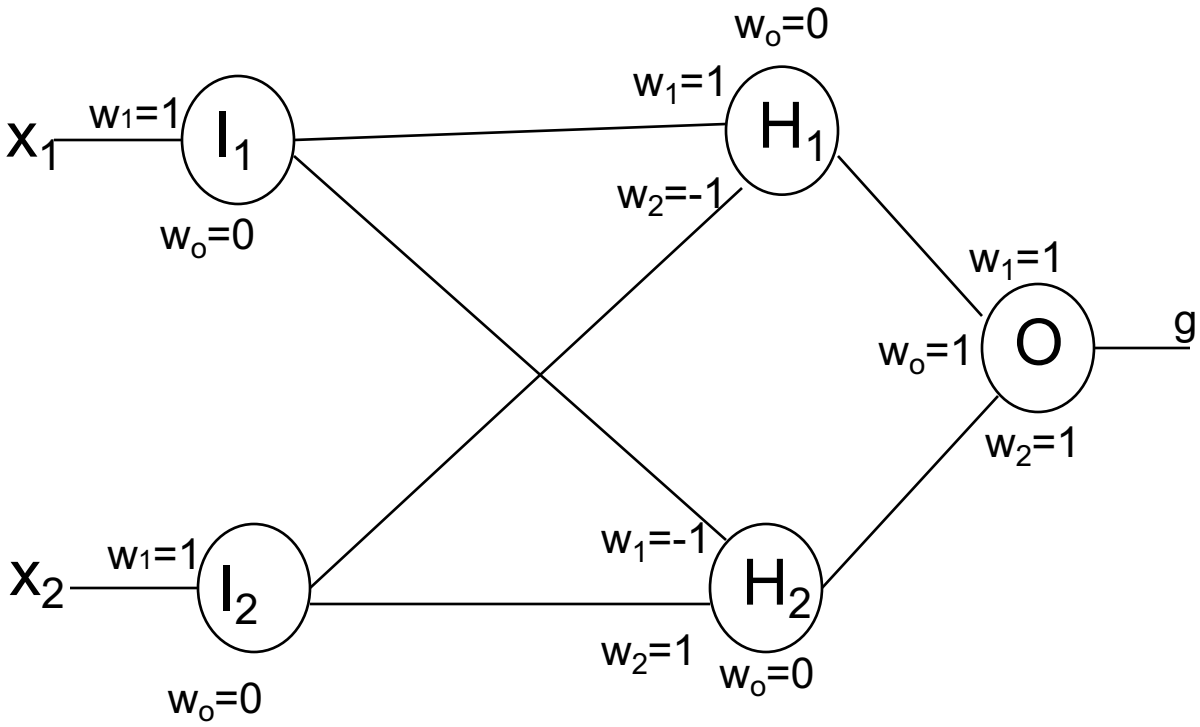


- La función g crea una barrera de decisión lineal
  - Los ejemplos que se pueden clasificar así se llaman linealmente separables
  - Este modelo de perceptrón sirve para aproximar (aprender) funciones linealmente separables
- El XOR no es linealmente separable
- La solución es usar más de 1 neurona

Poder de Representación Perceptrón



XOR


$w_o=0$
$w_1=1$
x_1
$w_1=1$
I_1
H_1
$w_2=-1$
$w_o=0$
$w_1=1$
$w_o=1$
O
g
$w_2=1$
x_2
$w_1=1$
I_2
$w_1=-1$
H_2
$w_2=1$
$w_o=0$
$w_o=0$

XOR

x1	x2	I1	I2	H1	H2	O
0	0	-1	-1	-1	-1	-1
0	1	-1	1	-1	1	1
1	0	1	-1	1	-1	1
1	1	1	1	-1	-1	-1

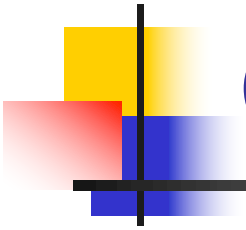
$I_1 = I_2 = 1$ si $x_i > 0$
-1 de otro modo

$H_1 = 1$ si $I_1 - I_2 > 0$
-1 de otro modo

$H_2 = 1$ si $-I_1 + I_2 > 0$
-1 de otro modo

$O = 1$ si $H_1 + H_2 + 1 > 0$
-1 de otro modo

- Note que w_0 es 0 para todas las neuronas excepto O, en donde vale 1



Como Entrenar una Red

- Queremos encontrar un algoritmo para entrenar una red no-recurrente “feed-forward”
- Primero vamos a ver una manera más general de entrenar una neurona (más general que la que vimos para el perceptrón)

La Regla Delta

- Comenzamos con la definición del error que queremos minimizar
  - $E(\mathbf{w})=1/2\sum_{d\in M}(V_{ent}-g)^2$

    donde V_{ent}-g es la diferencia entre el valor de entrenamiento y el valor que da el perceptrón
  - Tomamos la sumatoria del cuadrado de las diferencias para todos los ejemplos de entrenamiento y la dividimos entre dos
  - Es parecido al error cuadrático medio, pero en lugar de dividir entre el número de ejemplos de entrenamiento, dividimos entre dos

La Regla Delta

- Queremos modificar los pesos en la dirección que produce la disminución más rápida en el error
- Para encontrar cuál es, calculamos el gradiente del error
  - $\nabla E(\mathbf{w})=[\partial E/\partial w_o, \partial E/\partial w_1,\ldots, \partial E/\partial w_n]$
  - La derivada parciales del error con respecto a cada peso
  - $-\nabla E(\mathbf{w})$ da la dirección en que el error disminuye más rápido

La Regla Delta

- Como con el regresor lineal, vamos a tomar una ruta incremental para disminuir el error, por lo que ajustaremos los pesos con cada ejemplo de entrenamiento
- La regla para ajustar los pesos es entonces

$$w_i = w_i + \Delta w_i$$

donde $\Delta w_i = -\eta \, \partial E / \partial w_i$

- Para poder usar esta regla necesitamos calcular las derivadas parciales del error

$$\partial E / \partial w_i = \partial / \partial w_i \, 1/2 (V_{ent} - g)^2$$
$$= (V_{ent} - g) \, \partial / \partial w_i \, (V_{ent} - g)$$
$$= (V_{ent} - g)(- \partial / \partial w_i (g))$$

- Por lo tanto

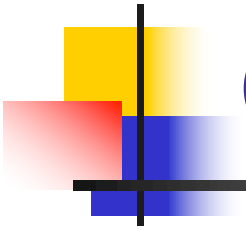
$$\Delta w_i = \eta (V_{ent} - g) \, \partial / \partial w_i \, (g)$$

La Regla Delta

- ¿Qué pasa cuando g es lo que usamos en el regresor lineal? i.e., $g(x_1,x_2,\ldots,x_n)=\sum w_i x_i$
  - $\Delta w_i = \eta(V_{ent}-g)\, d/dw_i\,(g)$

    $= \eta(V_{ent}- \sum w_i x_i)\, d/dw_i\,(\sum w_i x_i)$

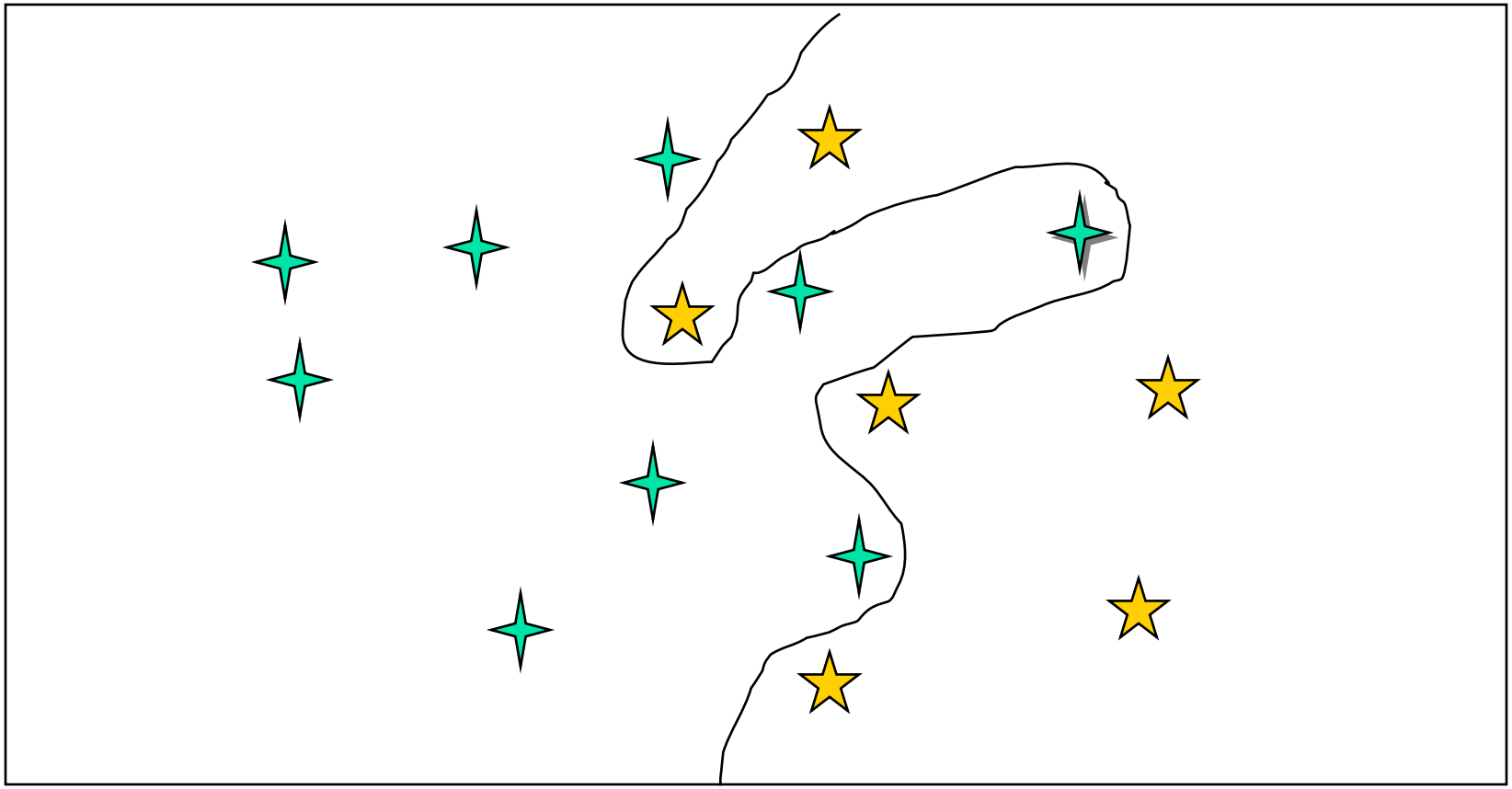
    $= \eta(V_{ent}- \sum w_i x_i)x_i$
- Esta es la regla LMS que usamos!
  - Se le conoce también como la regla Delta, la regla Adeline, o la regla Widrow-Hoff



Como Entrenar una Red

- Queremos poder clasificar eficientemente conjuntos no lineales
  - El problema de usar la g que usamos con el regresor lineal es que es lineal y una red feedforward es una combinación lineal de neuronas, por lo tanto, toda la red sería lineal
- El algoritmo para entrenar redes neuronales multi-capas que vamos a ver necesita que g sea diferenciable
  - El problema del perceptrón con la función g de escalón es que es discontinua y por lo tanto no diferenciable

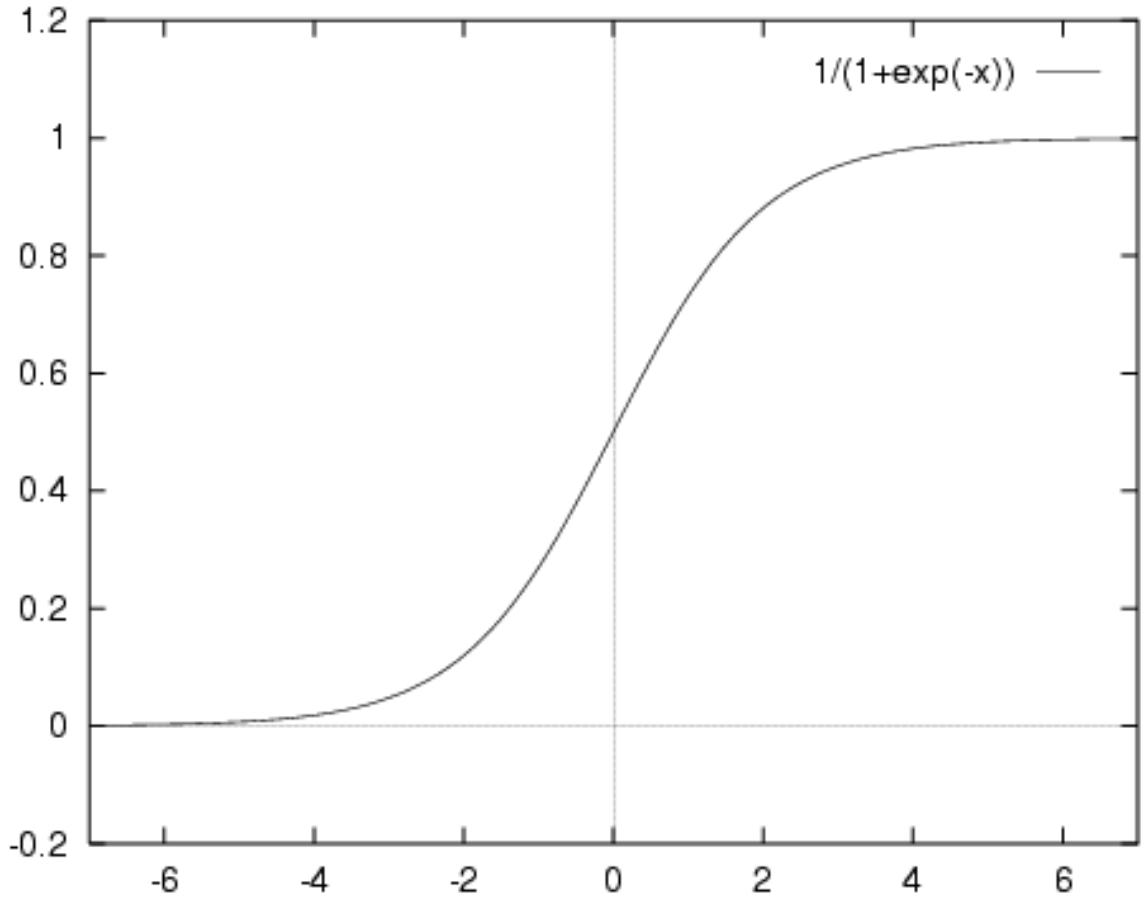
Puntos No Linealmente Separables



Representación No-lineal

- La solución a nuestros requerimientos es usar una función g que sea no-lineal y diferenciable
- Una alternativa es usar la función sigmoide σ
- $g(x_1,x_2,\ldots,x_n)= \sigma(\sum_{i=0,n} w_i x_i) =(1+e^{(-\sum w_i x_i)})^{-1}$

La Función Sigmoide



Regla Delta (no-lineal) Entrenar una Neurona

- La derivada parcial, g´, con respecto a w_i

  $\partial/\partial w_i = (1+e^{(-\Sigma w_i x_i)})^{-1}(1-(1+e^{(-\Sigma w_i x_i)})^{-1})x_i$

  $= \sigma(1-\sigma)\, x_i$

- Con esto, el algoritmo para entrenar una neurona se convierte en:

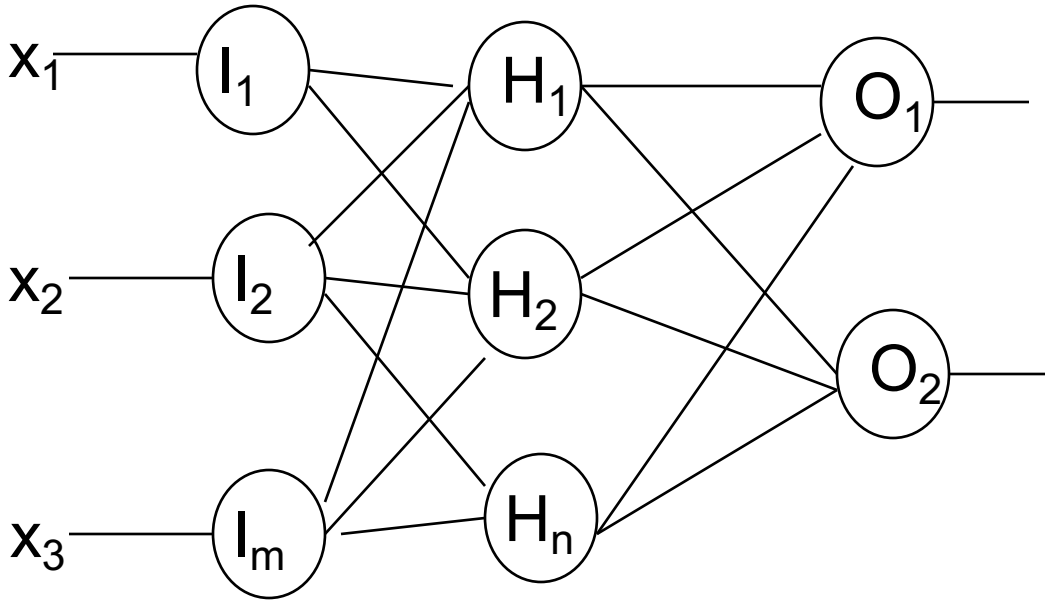
  $w_i = w_i + \Delta w_i$

  donde $\Delta w_i = \eta(V_{ent}-\sigma)\, \sigma(1-\sigma)\, x_i$

- ¿Qué cambió?
  - La función de transferencia y por lo tanto la manera en que cambia el peso con respecto al error

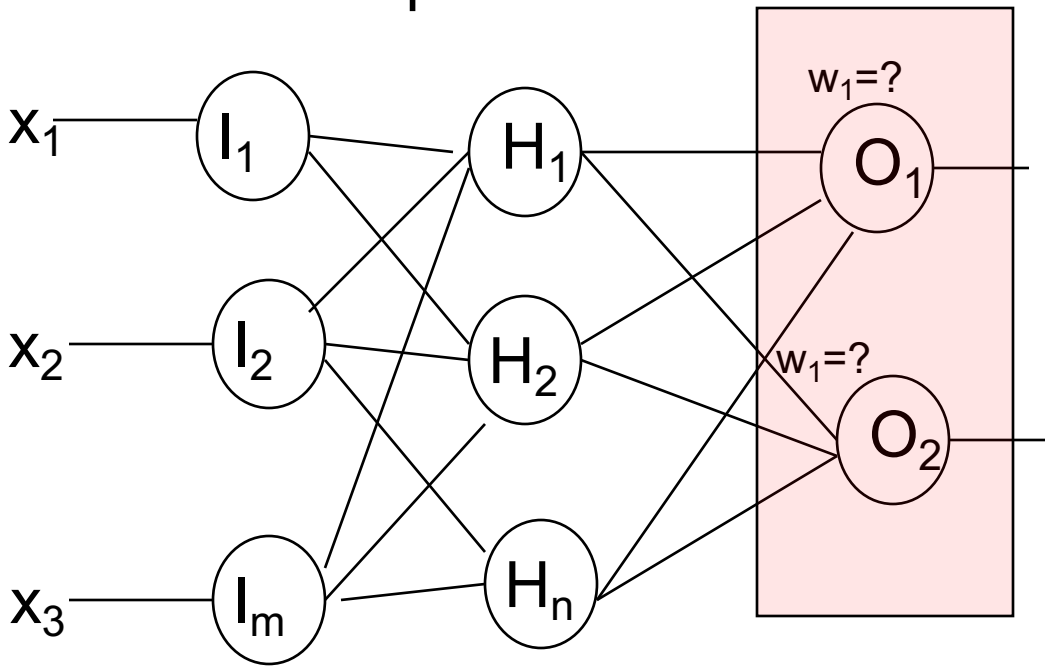
Como Entrenar una Red (feed forward)

- ¿Pero cómo entreno toda una red?



Como Entrenar una Red

- Comenzamos por las neuronas de salida



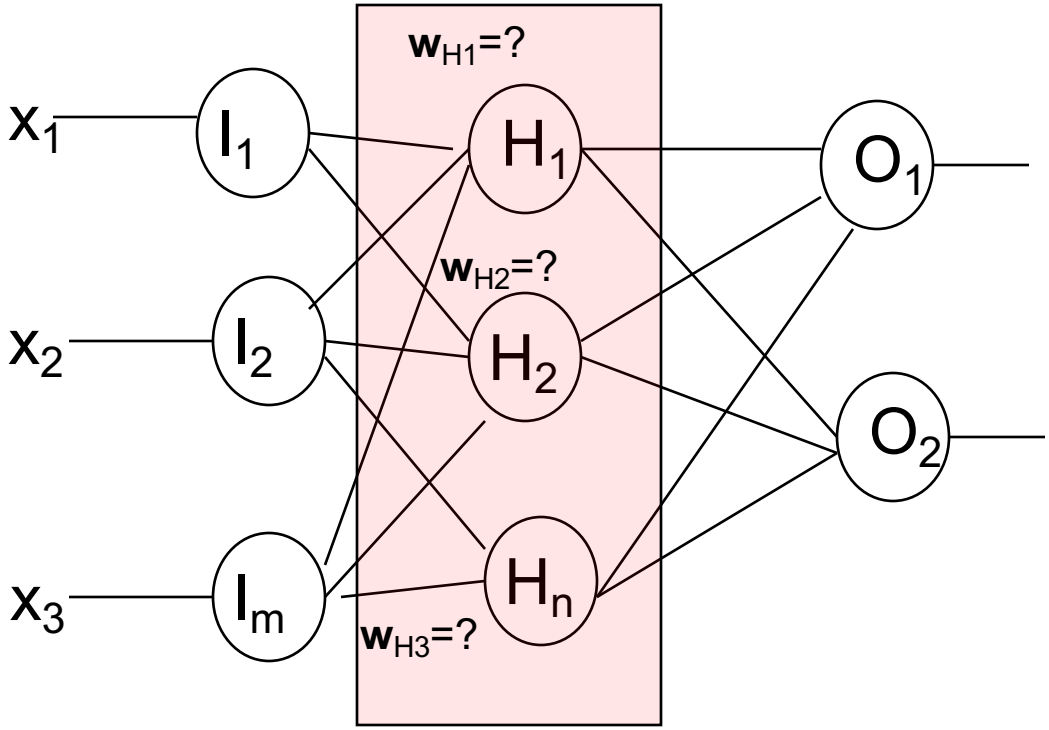
Asignación de Crédito

- Neuronas de salida
  - El error para cada neurona de salida O es

    $E_s = (V_{ent,s} - \sigma_s)\, \sigma_s(1 - \sigma_s)$
  - Es fácil pues tenemos V_{ent}, es el valor que queremos que la red aprenda para esa neurona, para la instancia de entrenamiento actual
  - Los pesos se actualizan conforme a :

    $w_i \leftarrow w_i + \eta(V_{ent,s} - \sigma_s)\, \sigma_s(1 - \sigma_s)\, x_i$
  - Como siempre, sólo que aquí las xi´s se refieren a las salidas de las neuronas del nivel anterior; las x_i´s son las entradas a las neuronas en cuestión

Como Entrenar una Red

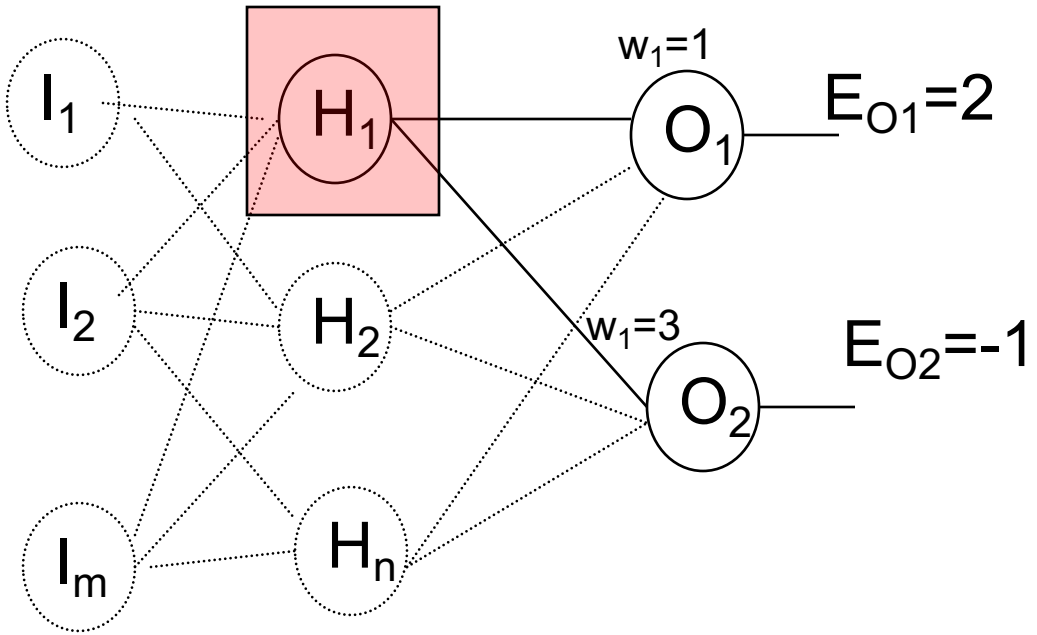


- ¿Cómo ajustar los pesos, w_i, para las neuronas H?
- ¿Cuál es el error?
- ¿Cómo creen?

Asignación de Crédito

- Neuronas intermedias
  - Tenemos el problema de asignación de crédito pues su retroalimentación es indirecta
  - Para las neuronas intermedias, H, no tenemos el término $(V_{ent,H}-\sigma_H)\ \sigma_H(1-\sigma_H)$ para calcular el error
  - ¿Qué hacer?
  - Sumamos los errores de las neuronas de la etapa siguiente (las de salida en el ejemplo) a las que esta conectada H. Ponderando cada error con el peso con el que H se conecta a cada neurona
  - Por ejemplo:

Asignación de Crédito Neurona Intermedia



Suma de contribuciones para H_1=1(2)+3(-1)=-1

Asignación de Crédito

- El error para la neurona intermedia H es entonces

$$E_H = \sigma_H(1- \sigma_H) \sum w_{Oi,H} E_{Oi}$$

donde la suma es sobre los neuronas a las que H tiene conexión y $w_{Oi,H}$ es el peso de la conexión entre la neurona H y la neurona Oi

- El peso w_i de la neurona H se actualiza conforme a:

$$w_i \text{ <---} w_i + \eta \, (\sigma_H(1- \sigma_H) \sum w_{Oi,H} E_{Oi}) \, x_i$$

donde las x_i´s son las entradas a la neurona (las salidas de la etapa anterior)

Algoritmo de Retropropagación "Backpropagation"

1.- Para cada instancia **X (X es un vector)**
- Calcular resultado con la entrada **X**

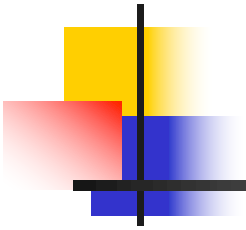
2.- Propagar los errores de la salida hacia "atrás"
- Para cada neurona de salida O calcular
  - $E_O=\sigma_O(1-\sigma_O)(V_{ent,O}-\sigma_O)$
- Para cada neurona intermedia h
  - $E_h=\sigma_h(1-\sigma_h)\sum w_{kh}E_k$, donde k son las neuronas de la etapa siguiente
- Para cada neurona j y peso w_{ji}

  $w_{ji} \leftarrow\!\!- w_{ji} + \eta E_j x_{ji}$

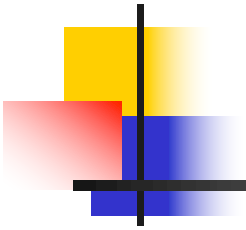
donde x_{ji} es la entrada a la neurona j de la neurona i y w_{ji} es el peso de esa conexión

3.- Repetir hasta lograr el error deseado o por un número predeterminado de iteraciones

Batch y Mini-batch Gradient Descent



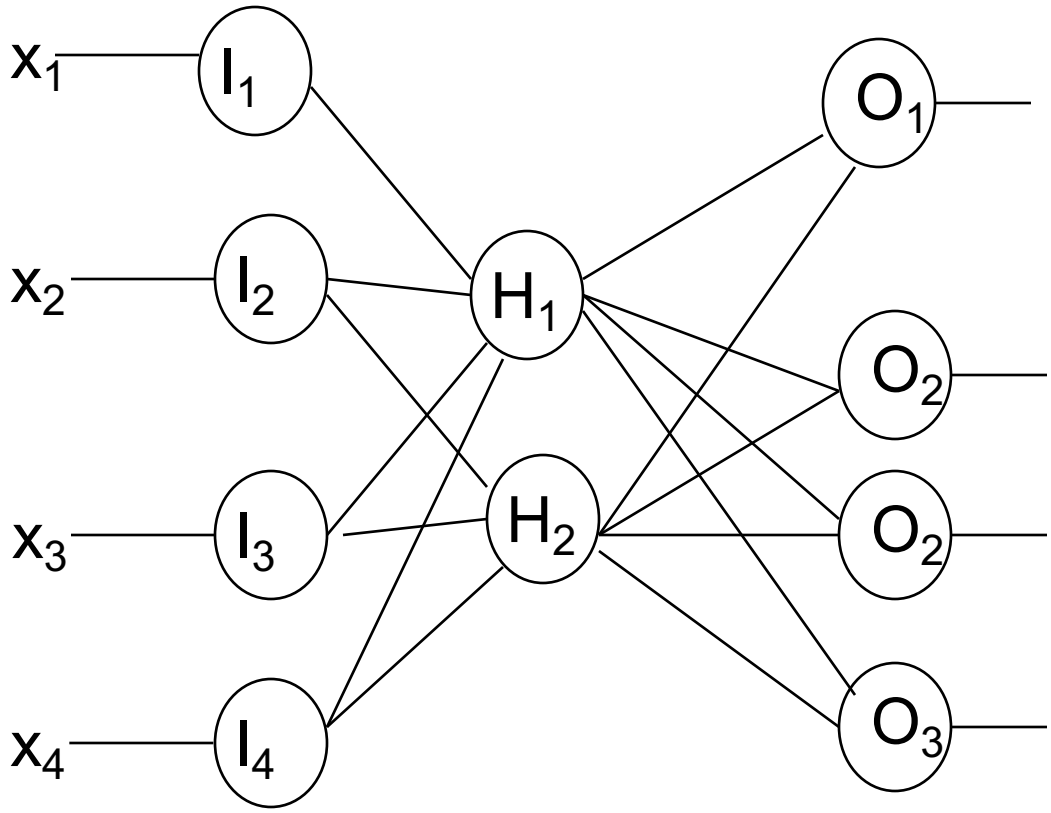
- En lugar de calcular el error para cada ejemplo y propagarlo ajustando los pesos
- Calculamos el error promedio para b ejemplos (suele ser 5 o 10) y luego propagamos ajustando pesos
- Si b=el numero de ejemplos totales entonces es Batch gradient descent, si es más chico es mini-batch



Representación Interna

- Una propiedad interesante de este tipo de RN es la habilidad de encontrar representaciones útiles en las capas intermedias.
- Pueden encontrar patrones que no están explícitamente en los datos de entrada, pero que son útiles para la labor de generalización
- De igual manera, si los datos de entrada contienen atributos aleatorios, i.e., que no contribuyen para la clasificación, esperamos que estos sean eliminados (su importancia reducida) en las capas intermedias

Representación Interna
Ejemplo



Representación Interna Ejemplo

Entrada		Intermedio		Salida
1000		.00 .14		1000
0100		.09 .93		0100
0010		.98 .99		0010
0001		.96 .06		0001

- Note que si redondeamos al entero más cercano tenemos un código binario. La red descubre esta representación eficiente

* Ejemplo generado con el Neural Applet Version 4.3.0 (www.cs.ubc.ca)

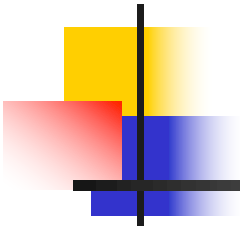
Heurísticos y recomendaciones

- ¿Cuántos datos para entrenar?
  - Al menos 10 veces el número de pesos en la red
- A lo más dos capas ocultas (a menos que….)
- Cuidado con el sobre entrenamiento
  - Usar el conjunto de validación
    - Determinar número de capas ocultas y número de neuronas
    - Determinar cuando debemos de dejar de aprender (“early stopping”)
    - Regularizacion y dropout

Heurísticos y recomendaciones

- Experimentar con otras funciones de activación
  - Tanh que se parece a la sigmoide pero va entre -1 y 1
  - ReLu que toma el máximo entre 0 y $w^T\mathbf{X}$
- Usar otras medidas de pérdida
  - Entropía cruzada (p es la distribución real y q la estimada)

$$H(p,q) = -\sum_x p(x)\,\log q(x).$$



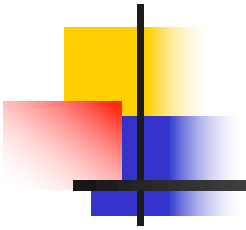
Ejercicios

1. Entrene una red neuronal para resolver el problema de XOR
   - Opcional: Haga una visualización que permita observar, aunque sea aproximadamente, la(s) frontera(s) de decisión
2. Haga una red neuronal que identifique puntos dentro de un círculo (genere usted los datos)
   - Cambie el número de neuronas de las capas intermedias

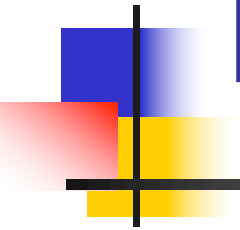
Algunas Aplicaciones

- Identificación de patrones y tendencias
- Predicción de ventas
- Predicción de precios (bolsa)
- Risk management
- Evaluación de créditos
- Medicina
  - Modelos de sistemas corporales, predicción de enfermedades. Epidemiología
- Reconocimiento de caracteres
- Reconocimiento de lenguaje hablado
- Filtrado de señales
- Control
- Compresión de imágenes
- ……

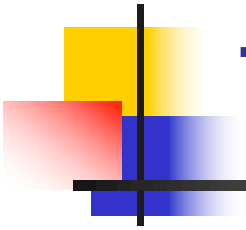
Herramientas



- Tensorflow, Teano, Caffe
- Neuroshell
- Matlab
- R (RSNNS, neuralnet)
- Python: PyBrain, pyfann…
  - Nosotros usaremos Pybrain. Instálenlo
- Varios paquetes en la red, por ejemplo
  - Neural Applet
    - http://www.aispace.org/
- Hay también mucho código que pueden usar (SNNS)

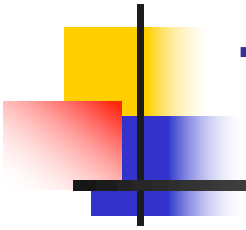


Diapositivas extras



Tensorflow

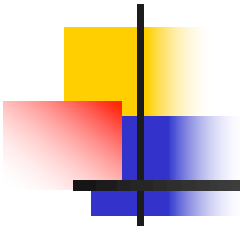
- Qué es?
  - Paquete de aprendizaje de máquina usado para redes neuronales
  - Rápido y escalable
- Cómo se usa?
  - Se definen las variables y las operaciones a realizar (el grafo en donde los nodos son operaciones y los datos)
  - Se abre una sesión
    - Responsable de los recursos para el cómputo
  - Se manda llamar a las operaciones definidas
  - Toda interección se maneja a través del manejador de la sesión



Tensorflow

- Ejercicios
  - Perceptrón
  - XOR
  - Círculo
  - Imágenes

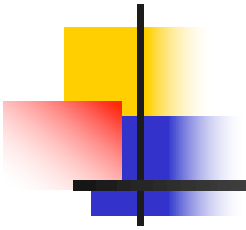
Ejercicio (en caso de no haberlo hecho antes)



- Programe un perceptrón con función de transferencia lineal en python (perceptron4Class.ipynb si no lo programaron antes)
- Entrénelo para la función *and* luego para la función *or*
- Visualice los datos y grafique la barrera de decisión
- Pueden hacerlo en equipos de dos

Ejercicio Red en Pybrain Entrenamiento

- from pybrain.tools.shortcuts import buildNetwork
- from pybrain.datasets import SupervisedDataSet
- from pybrain.supervised.trainers import BackpropTrainer
- X,Y=samples(10000) **---Construido por ustedes. Definan la codificación apropiada de los datos y cree ejemplos para entrenar**
- net = buildNetwork(9, 2, 2)—**Experimentar topologías**
- Agrea\gar a estrucutra de pybrain
  - ds = SupervisedDataSet(9, 2)
  - ds.setField('input', X)
  - ds.setField('target', Y)
- Entrenar red
  - trainer = BackpropTrainer(net, ds)
  - for i in range(5): ----**probar con número de ciclos**
- trainer.train()



Ejercicio

- Obtener una salida de la red para el vector de entrada X
  - net.activate(X)