Aprendizaje de Máquina

ITAM

Semestre agosto-diciembre 2017

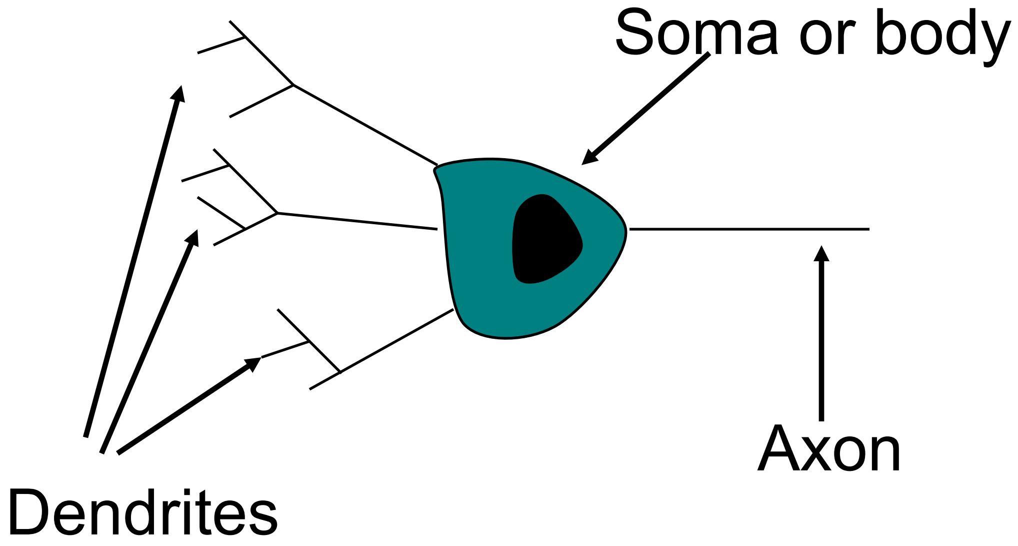
Menu

- Neural Networks
  - Inspiration
  - Types of networks
    - Recurrent or cyclic
    - Acyclic
  - A neuron model
    - Training algorithm
  - Acyclic networks (feed forward)
    - Training algorithm. Backpropagation

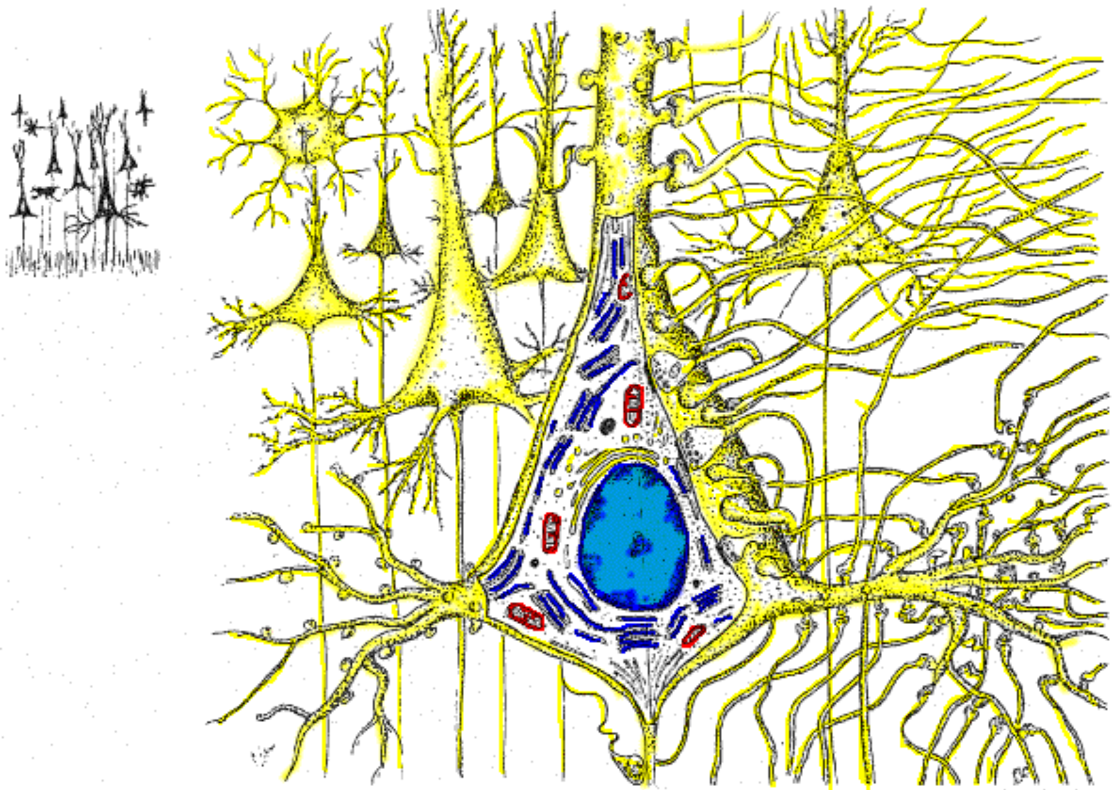
Biological Neural Networks

- A human being has 10^{11} neurons on average. Each connected to approximately 10^{4} other neurons
  - ¿Total conections?
- Each neuron is activated or inhibited depending on the inputs it receives from other neurons

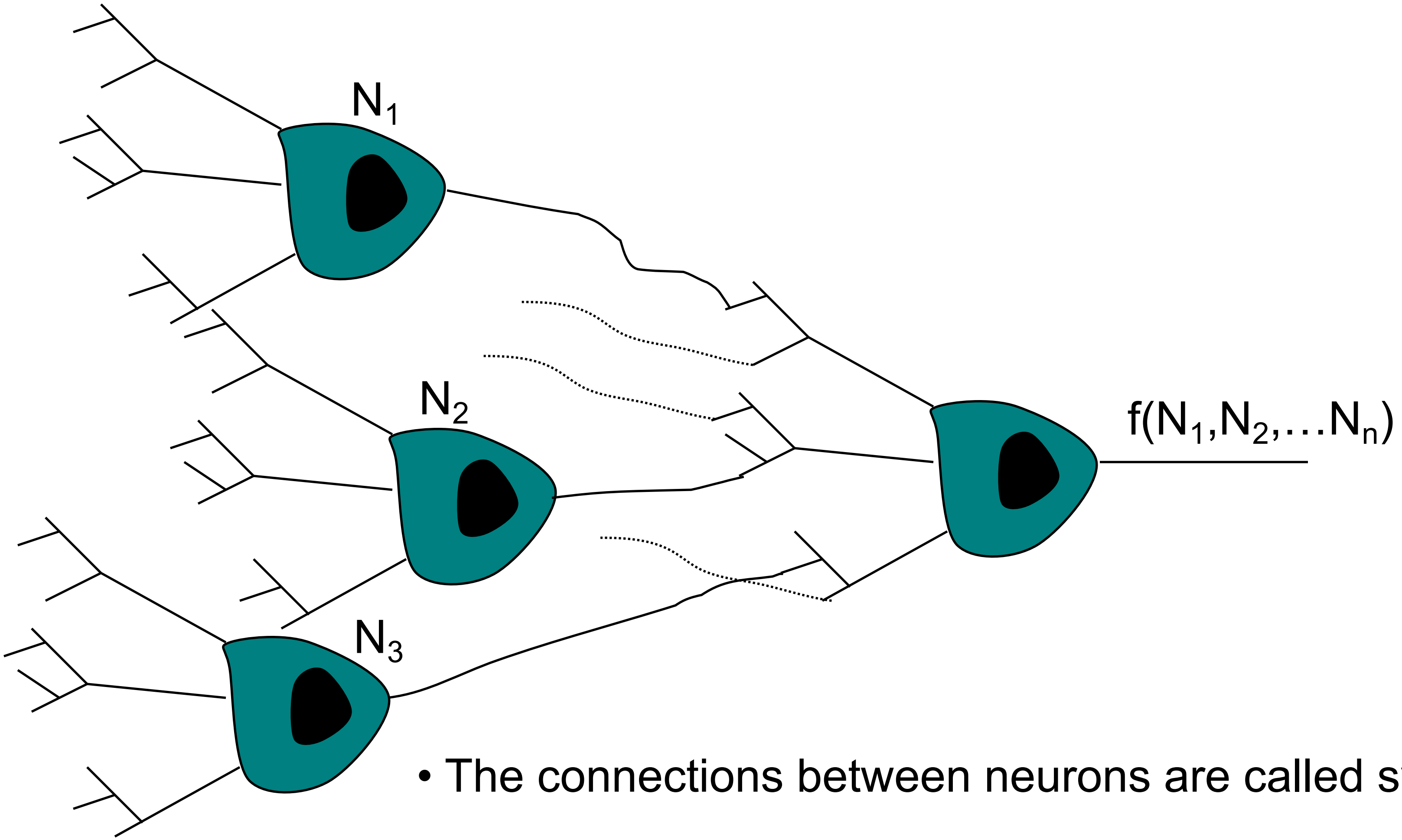
A Biological Neuron



Biological Neuron



Biological Network



- The connections between neurons are called synapses

Biological Network

- The fastest neurons can switch signal once every 10^{-3} seconds
- A transistor can do it in 10^{-11} seconds
- Nevertheless human beings can take very complex decisions very fast
  - Why?

Neural Networks

- Artificial neural nets are inspired by their biological counterpart, but have many simplifications, in particular:
  - All neurons are the same (or at least there is not much variety)
  - The operation of a neuron is idealized (neurons of the same “type” operate exactly the same)
  - The number and type of conections are much simpler

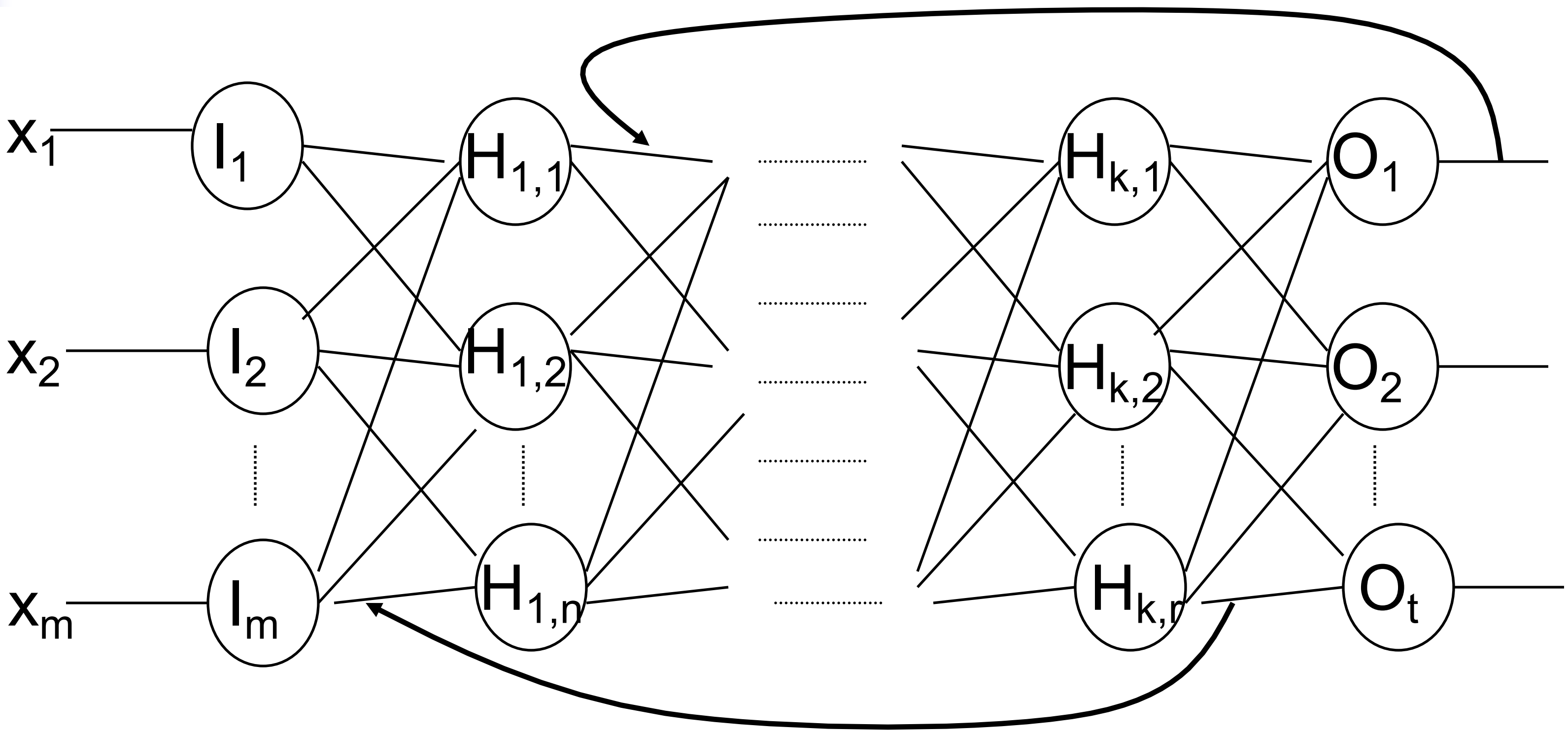
Artificial Neural Neworks

- They are used for
  - Unsupervised learning
  - Supervised learning
    - Classification problems
    - Regression problems
- They are slow to train but fast to test
- Used for:
  - Self –driving cars
  - Stock market prediction
  - Image and language recognition
  - Almost everything else

Artificial Neural Networks Topologies

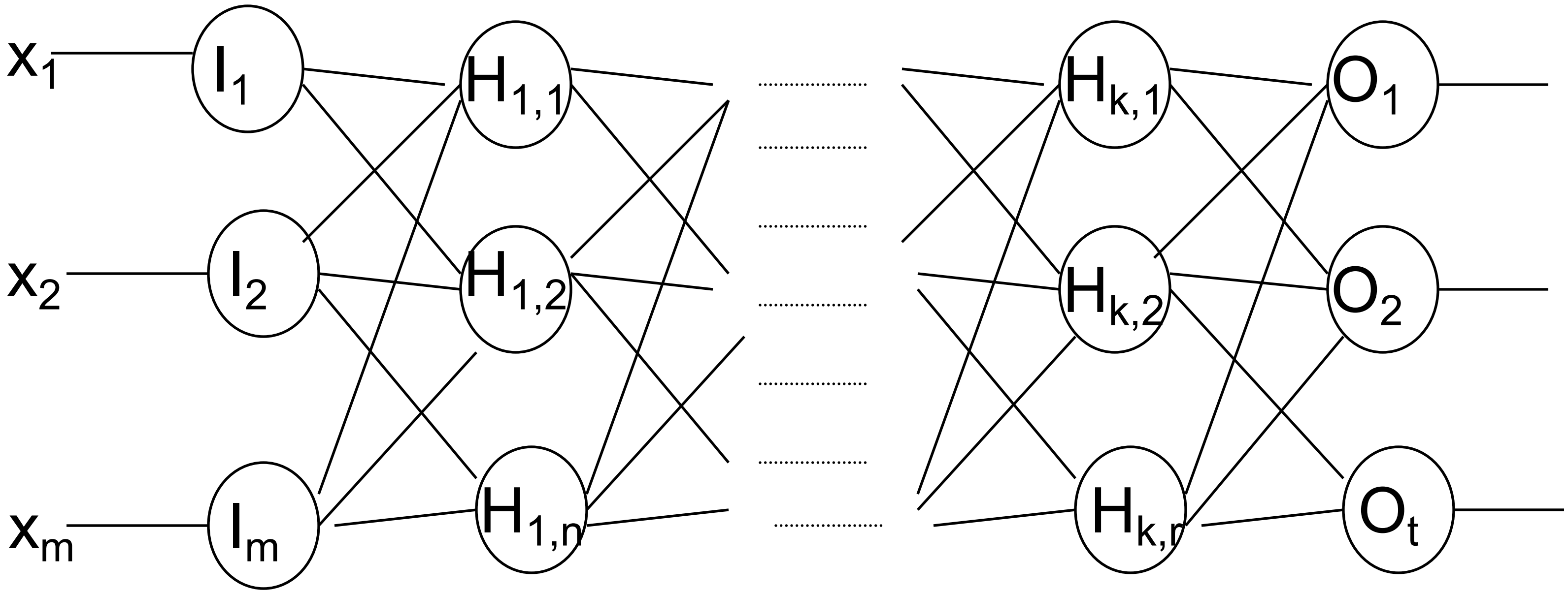
- Types of Networks
  - Acyclic
    - There is no path from the output of a neuron to its input for any neuron in the network
    - We will look at a particular acyclic topology called feed-forward
  - Recurrent
    - There are paths from the output of neuron to its input
  - Mixed
    - Acyclic and cyclic networks can be stacked together

Topologies
Recurrent



- Having a loop between the output of a neuron to its input allows for the preservation of state, for memory

Topologies
Feed forward (a particular acyclic network)



- Divided in three types of layers. Input layer **I**, hidden layers **H** and output layer **O**
- A network can have more than one hidden layer
- Each layer can have a different number of neurons

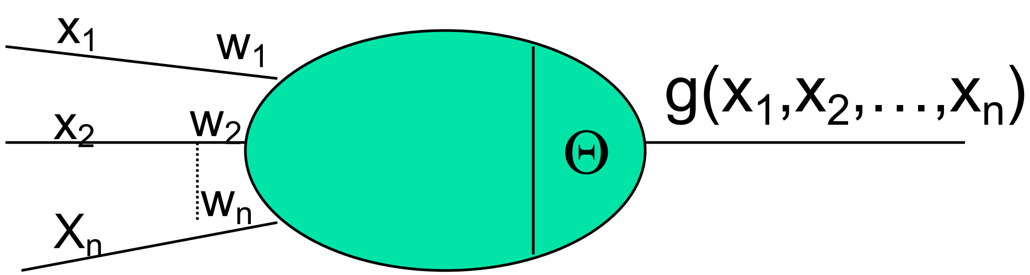
Characteristics
Feed-forward Network

- Each neuron in **I** has only one input
- The number of neurons in **I** and **O** are defined by the problem at hand
- The number of neurons and layers in **H** are free parameters
- Neurons at each layer are connected only to neurons in the next immediat layer (often to all of them)

Map

- We will see:
  - Neuron models
    - Linear, step y sigmoid
  - Topology
    - Feed-forward
  - Supervised Learning
    - Retro-propagación “Backpropagation”

Neural Networks
Neuron model



- Each neuron has an activation level that depends on its inputs
- An activated neuron transmits a signal according to its transfer funcion g :
    - Lineal
    - Step
    - Sigmoid
    - Other

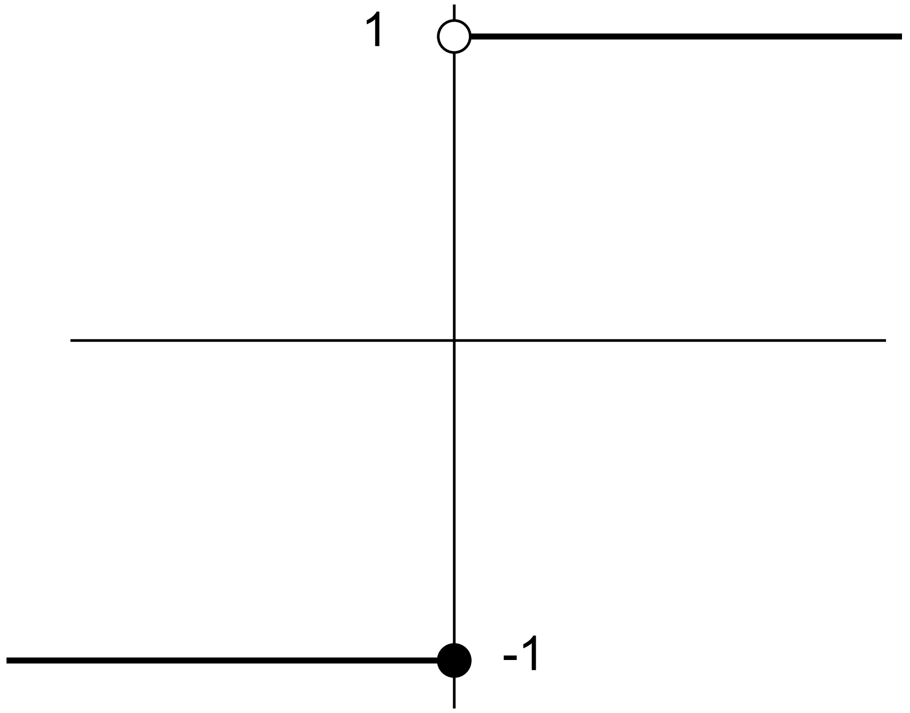
Neural Networks
Neuron model

- The activation of the neuron is a function of the weighed sum of its inputs
  - Weighed sum=$\sum w_i x_i$
  - Where the x_i are the inputs to the neuron and the w_i the weights associated with each input
  - Learning is the task of finding the right values for the weights
  - What happens if the transfer function g is simply the output of the weighed sum?

Neural Networks
Neuron model: Perceptron

- The function that represents the activation of the perceptron is
  - $g(x_1,x_2,..,x_n)= \begin{cases} 1 \text{ si } w_o+\sum_{i=1,n}w_i x_i >=0 \\ -1 \text{ otherwise} \end{cases}$
  - We can think of w_o as a threshold value since it does not depende on an input variable.
  - We could say that the perceptron fires if there is enough stimulus in the input, if the weighed sum of the inputs is greater than $-w_o$.

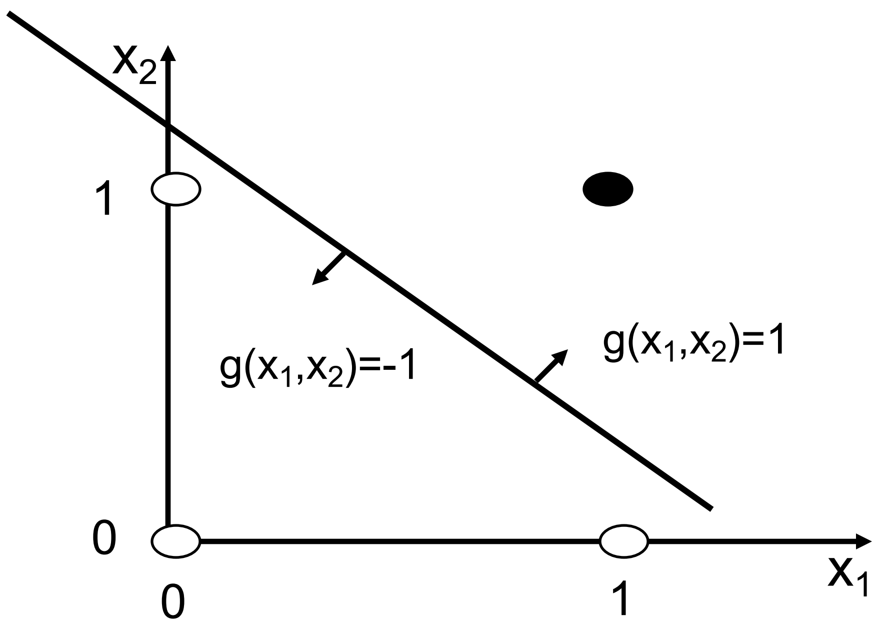
Perceptron
Transfer function: Step function



Representational ability Perceptron

- To ilustrate the representation power of the perceptron we can plot the equiation $\sum_{i=0,n} w_i x_i = 0$
- Since when $\sum_{i=0,n} w_i x_i$ is greater that or equal to zero it classifies an input as 1 and -1 otherwise
  - $\sum_{i=0,n} w_i x_i = 0$ represents a decision bounday

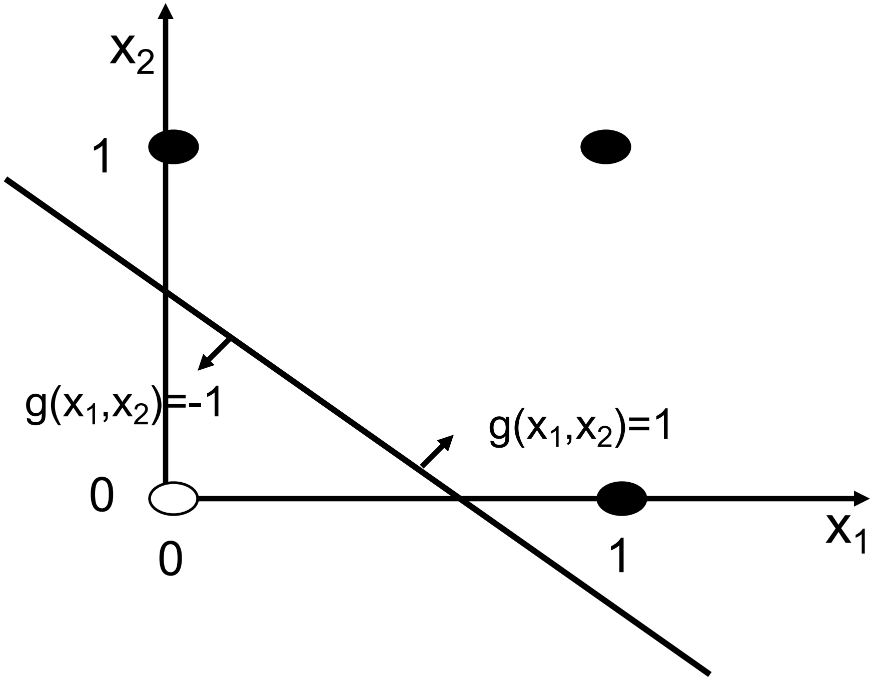
Representation Power Perceptrón



• White and black circles belong to different categories

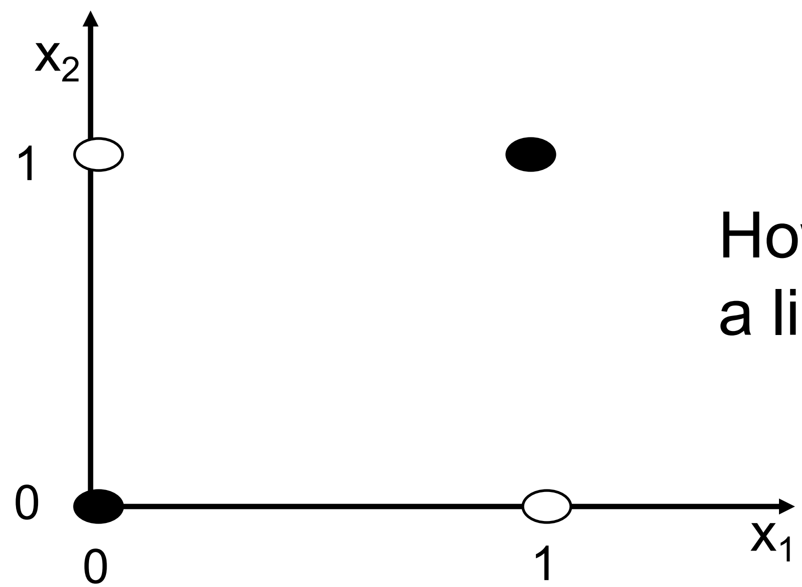
•What function is this?

Representation Power Perceptrón



- And this?

Representation Power Perceptron



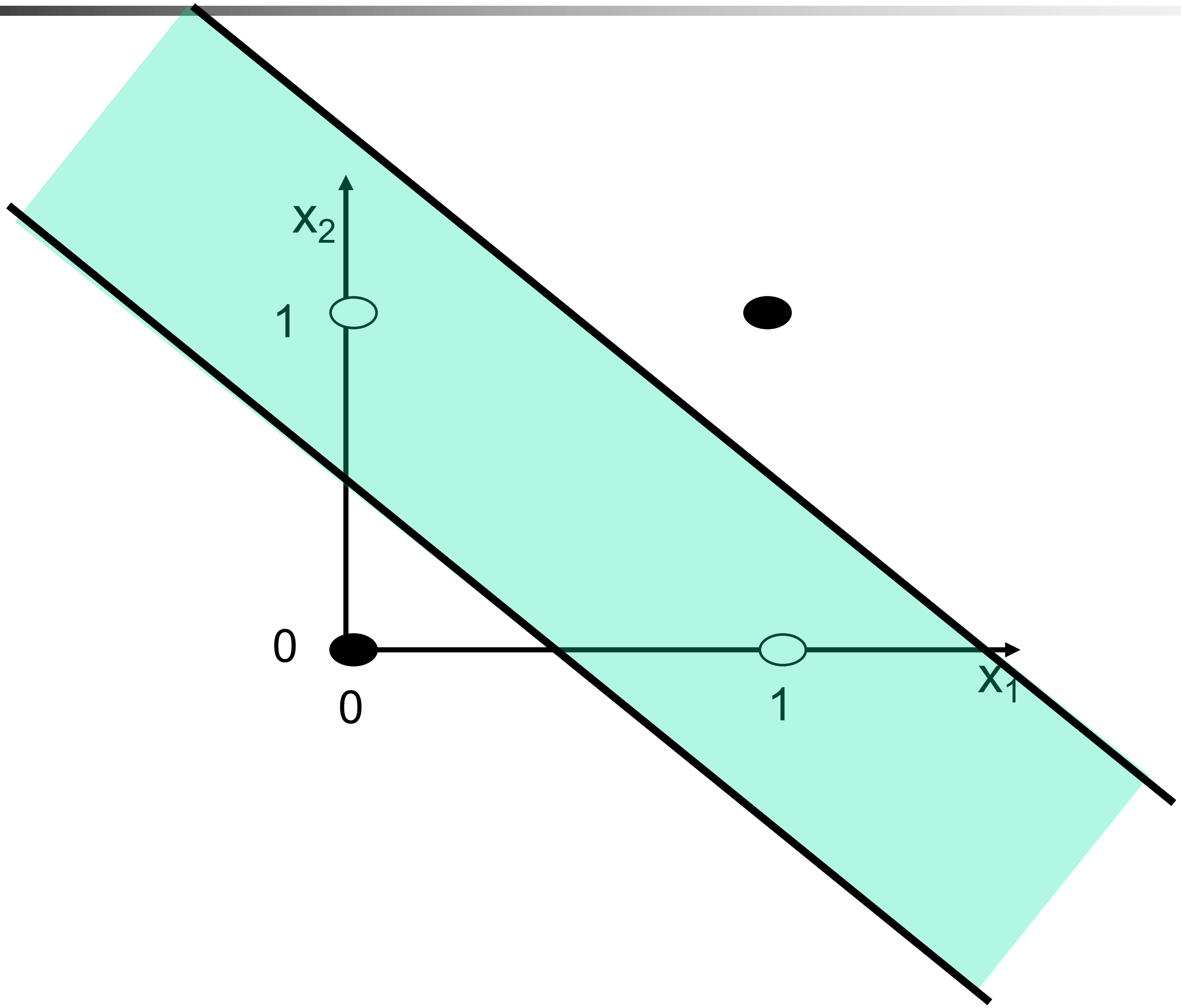
How to separate with a line?

- Minsky y Papert pointed out this limitation and set back the development of neural netwolks for years

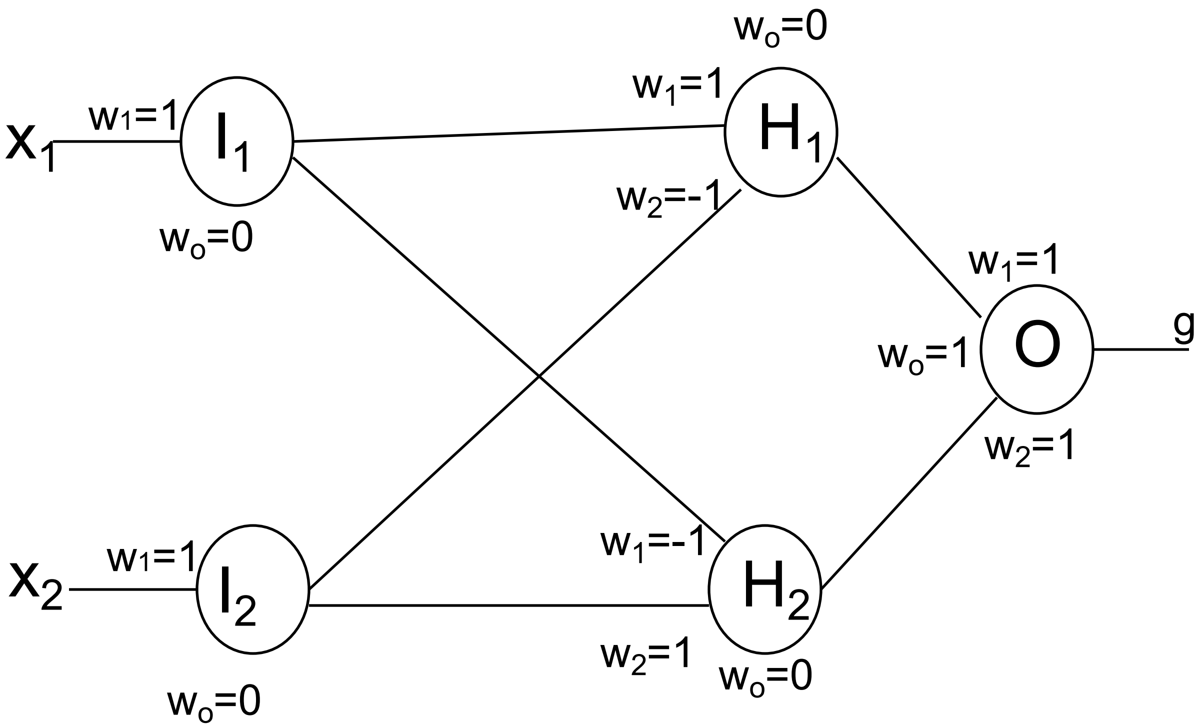
Representation Power Perceptron

- The transfer function g creates a linear decision boundary
  - Problems whose instances can be classified like this are called linearly separable
  - The perceptron can learn linearly separable functions
- XOR is not linearly separable
- The solution is to use more that 1 neuron
  - The challenge is how to connect them and how to train them

Representation Power
Perceptron



XOR (feed-forward network)



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

H_1= 1 si I_1-I_2 >0
-1 otherwise

I_1=I_2= 1 si x_i>0
-1 otherwise

O= 1 si H_1+H_2+1 >0
-1 otherwise

H_2= 1 si -I_1+I_2 >0
-1 otherwise

- Note that w_0 is 0 for all neurons except O, where its value is 1

How to train

- We wish to find an algorithm for training a feed-forward network
- But first lets revisit how to train a single neuron

Delta Rule

- Start with the function to minimize, the error
  - $E(\mathbf{w})=1/2\sum_{d\in M}(V_{ent}-g)^2$

    where V_{ent} -g is the diference between the model's value and the real value

Delta Rule

- During training we wish to modify the weights in the direction that more quickly decreases the error
- To find this we calculate the gradient
  - $\nabla E(\mathbf{w})=[\partial E/\partial w_o, \partial E/\partial w_1, \ldots, \partial E/\partial w_n]$
  - The partial derivatives of the error with respect to each weight
  - $-\nabla E(\mathbf{w})$ is the direction of que quickest decrease in error

Delta Rule

- As with the incremental regression we will take an incremental route to diminish the error and adjust the weights after evaluating each training sample
- The update rule is then:

$$w_i = w_i + \Delta w_i$$

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

- To use this rule we need to be able to compute the partial derivatives of the error

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

- Therefore

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

Delta Rule

- What happens when $g(x_1,x_2,\ldots,x_n)=\sum w_i x_i$?
  - $\Delta w_i = \eta(V_{ent}-g)\ d/dw_i\ (g)$

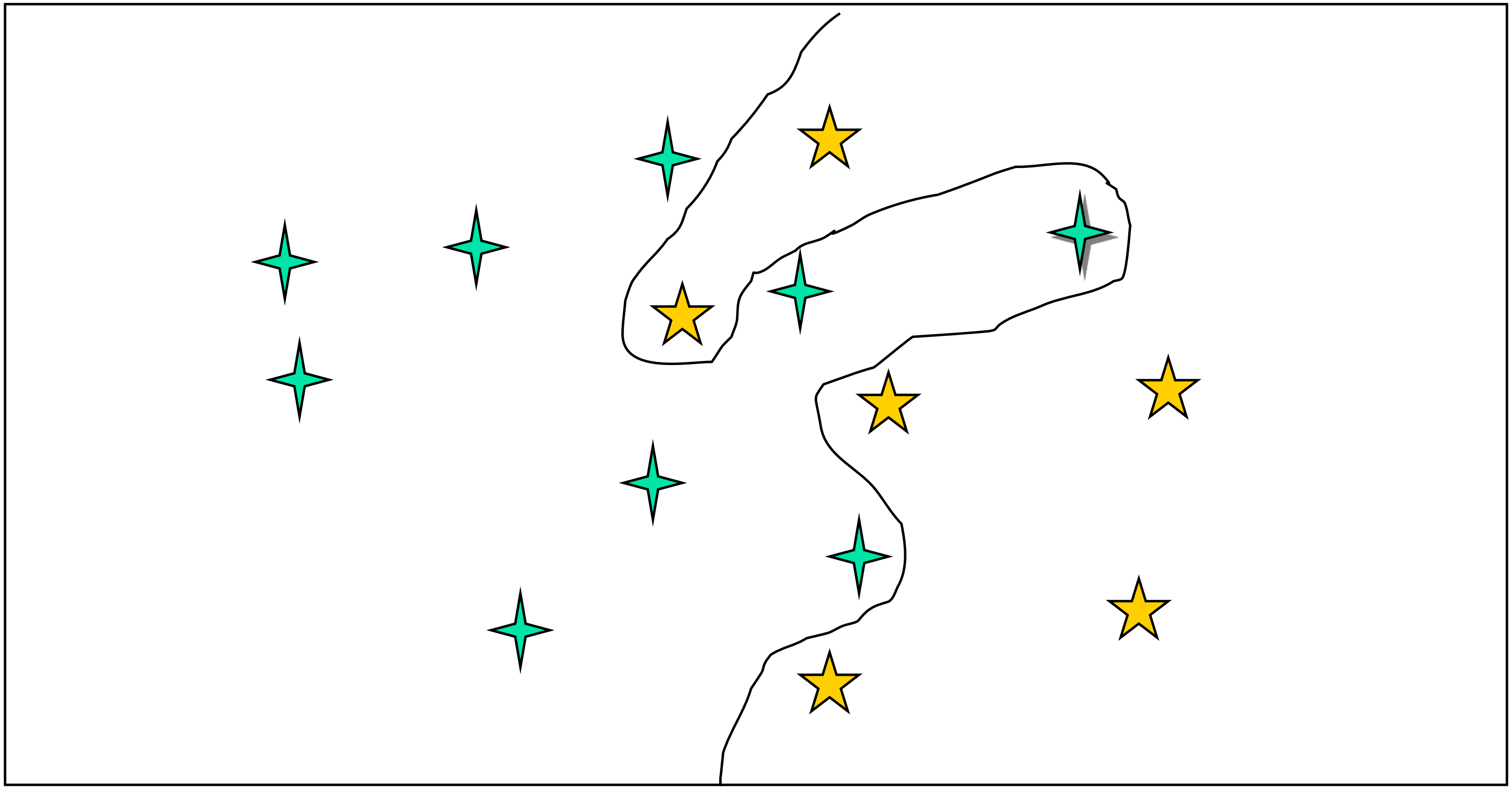
    $= \eta(V_{ent}-\sum w_i x_i)\ d/dw_i\ (\sum w_i x_i)$

    $= \eta(V_{ent}-\sum w_i x_i)x_i$
- This is the LMS that we used before!
  - Also known as Adeline or Widrow-Hoff

How to train

- We want to be able to classify sets that are not linearly separable
  - The problem is that using the lineal g in a feed forward network will still yield a linear boundary. The feed forward network is a linear combination of its neurons
- The algorithm to train multi-level networks that we are going to use requires g to be differentiable
  - The problem is that the perceptron, the step function, is discontinuous and thus not differentiable
  - But it is non-linear

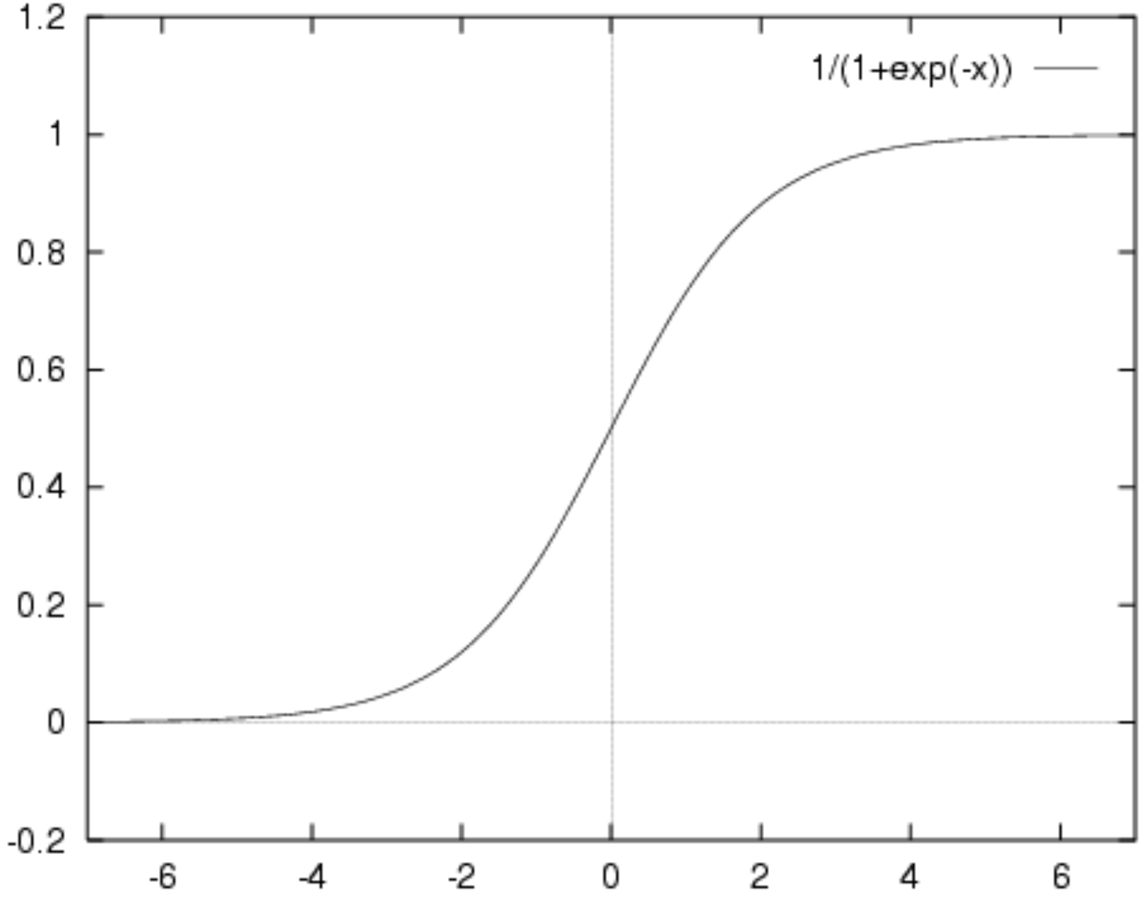
A non-linear boundary



Non-linear Representation

- The solution is to use a g that is non-linear but differentiable
- One possibility is to use the sigmoid function σ
- $g(x_1,x_2,\ldots,x_n)= \sigma(\sum_{i=0,n}w_i x_i) =(1+e^{(-\sum w_i x_i)})^{-1}$
- Performing regression with this transfer function is called logistic regression (the sigmoid is also called the logistic function)

Sigmoid



Delta Rule(non-linear) One neuron

- The partial derivative, g´, with respect to w_i

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

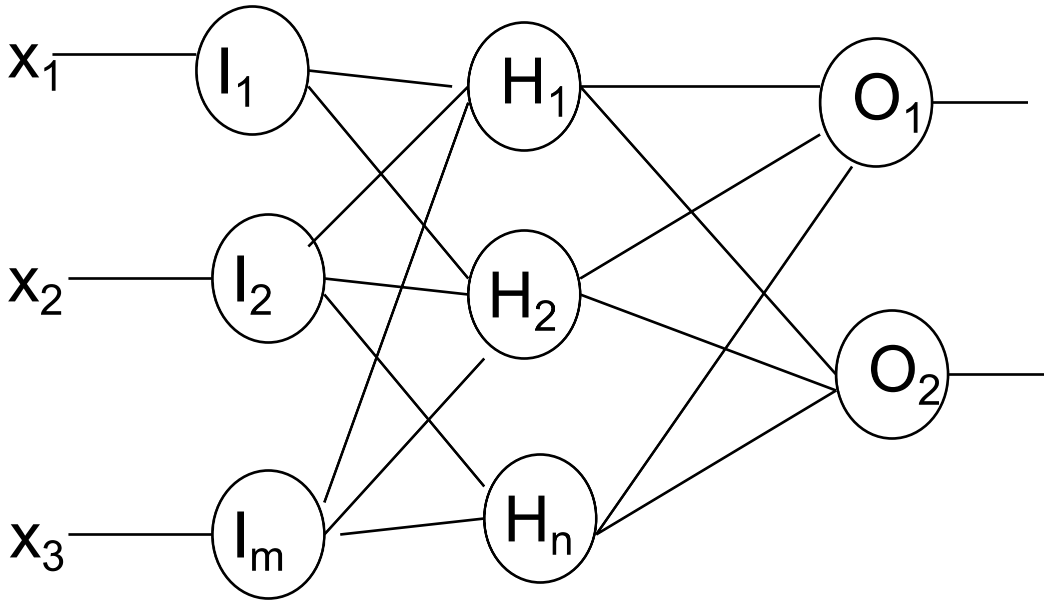
  $= \sigma(1-\sigma)\, x_i$
- The algorithm for a single neuron is then:

  $w_i=w_i + \Delta w_i$

  where $\Delta w_i = \eta(V_{ent}-\sigma)\, \sigma(1- \sigma)\, x_i$
- What changed?
  - The transfer function g and thus how the weights change with respect from to the error

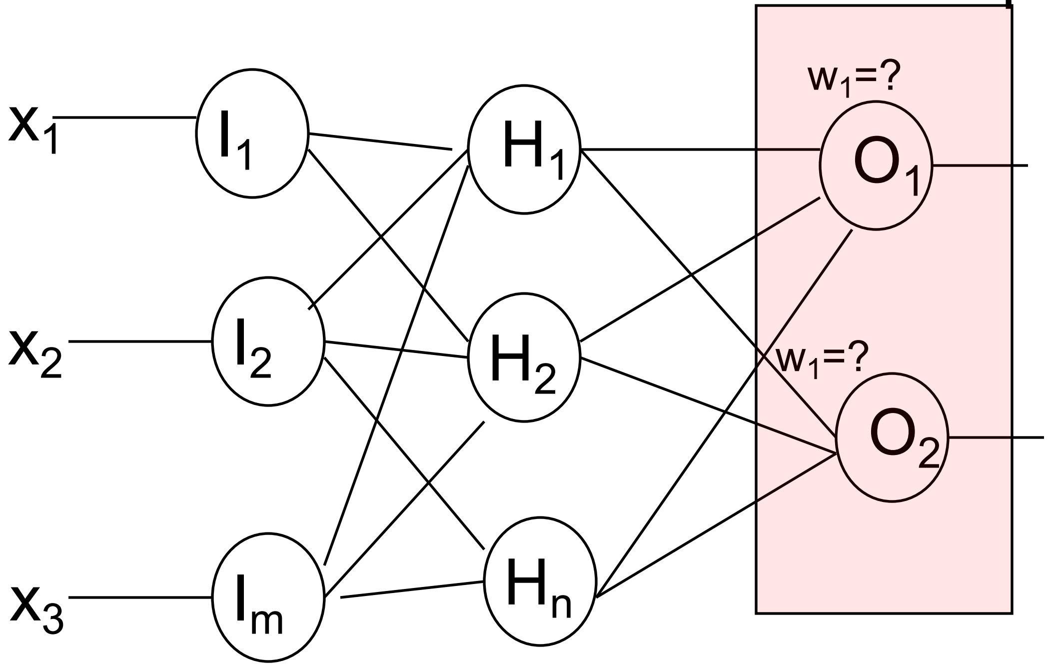
How to train (a feed forward network)

- So how do I train it?



How to train

- We start with the neurons at the output layer



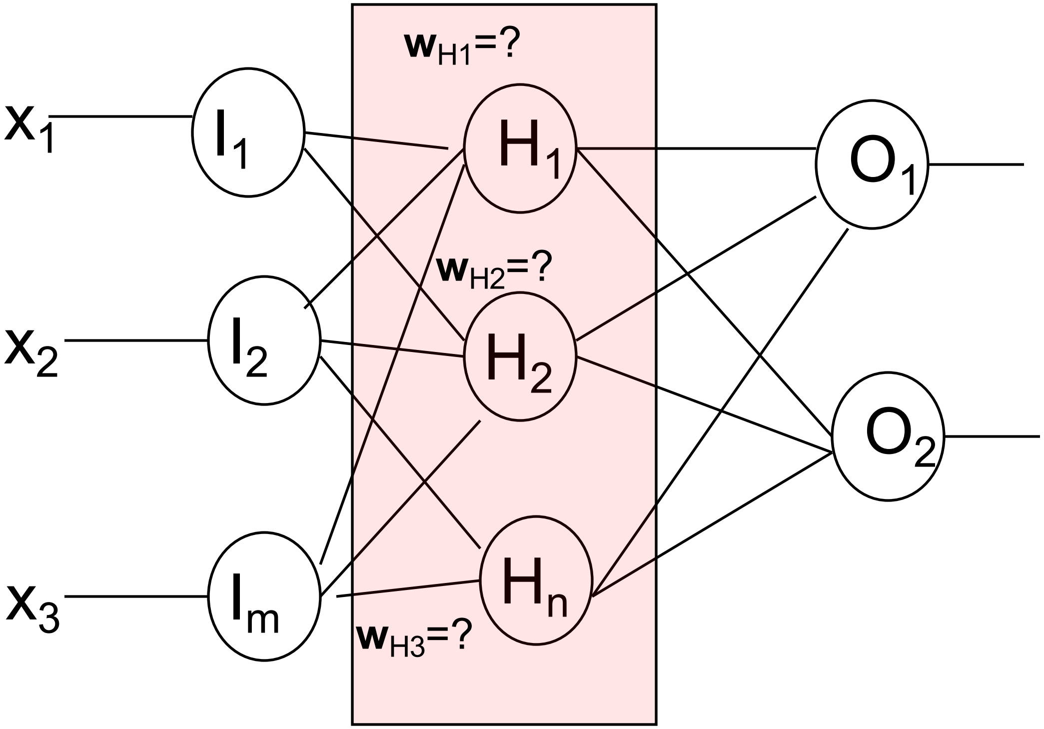
Credit Assignment

- Output neurons
  - The derivative of the error wrt the its input for each output neuron in O is

    $E_s = (V_{ent,s} - \sigma_s)\, \sigma_s (1 - \sigma_s)$
  - Its easy since we have V_{ent} , the values that we want to learn for the current example
  - The weights update according to :

    $w_i \leftarrow w_i + \eta (V_{ent,s} - \sigma_s)\, \sigma_s (1 - \sigma_s)\, x_i$
  - Like we've always done, only that here the x_i's refer to the outputs of the previous layer; the x_i's are the inputs to the neurons in question

How to Train

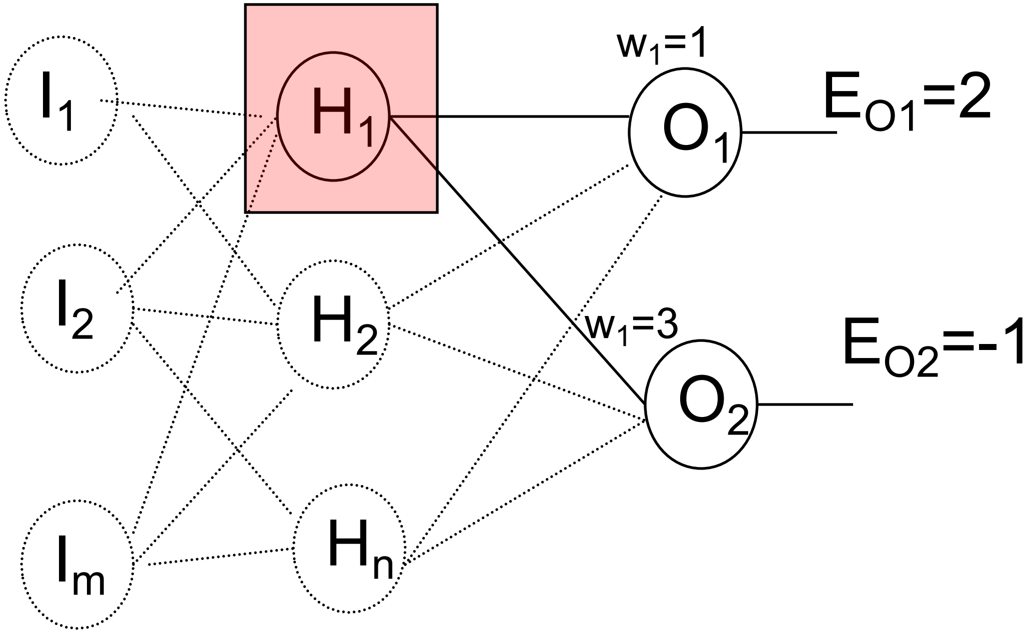


- But how to adjust the weights w_i for the neurons in H?
- What is missing as compared to the O layer?

Credit Assignment

- Neurons in H
  - We have a credit assignment problem since the feedback to H is indirect
  - For H we don't have $(V_{ent,H}-\sigma_H)$ in order to compute the error
  - What to do?
  - Intuition: add the errors of the next layer (O in this example) and approportion the error according to the connecting weights.
  - Por ejemplo:

Credit Assignment Intermediate Neurons



Sum of contributions for $H_1=1(2)+3(-1)=-1$

Credit Assignment

- The derivative of the error for an intermediate neuron H wrt its input

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

  where the sum is over all the neurons to which H is connected and $w_{Oj,H}$ is the weight of the connection between H and the O_i neuron at the next layer
- The weight w_i of H is updated according to:

  $$w_i \leftarrow\!\!-\!\!-\!\!- w_i + \eta \sum w_{Oj,H} E_{Oi}\ \sigma_H(1-\sigma_H)\ x_i$$

  where the x_i's are its inputs coming from the previous layer

Training Algorithm Backpropagation

1.- For each training instance **X**

- Compute the network´s output for input **X**

2.- Propagate the output errors back

- For each neuron O in the output layer calculate
  - $E_O = \sigma_O(1 - \sigma_O)(V_{ent,O} - \sigma_O)$
- For each intermediate neuron h in the hidden layers
  - $E_h = \sigma_h(1 - \sigma_h) \sum w_{kh} E_k$, where k are the neurons in the next immediate layer
- For each j and weight w_{ji}

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

  where x_{ji} is the input to neuron j from neuron i and w_{ji} is the weight of this connection

3.- Repeat until a desired error is achieved or there is no significant decrease in error or for a predetermiend number of iterations (also known as epochs)

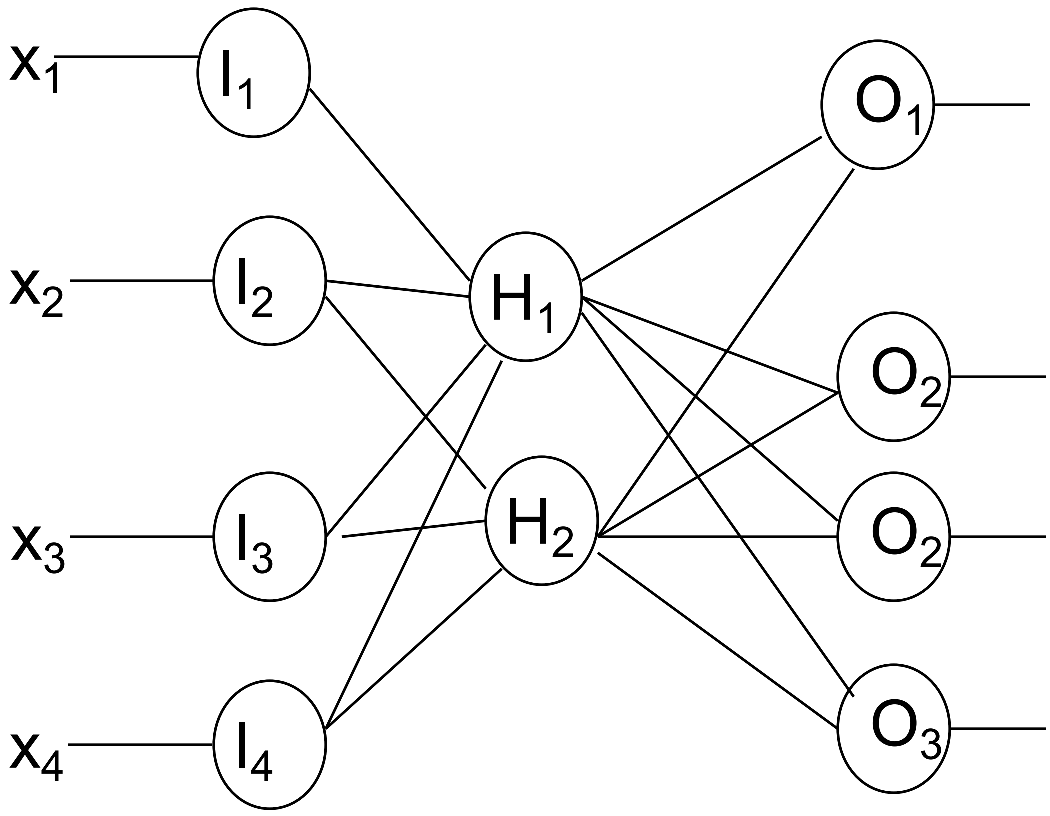
Batch and Mini-batch Gradient Descent

- Instead of computing the error for each example, propagating the errors back and adjusting the weights
- Compute the average error for a larger subset, propagate it back and adjust the weights

Internal Representation

- An interesting property of ANN is their ability to find useful representation of the data in its hidden layers
- They can find patterns that are not explicit in the input data but which are useful for gerneralizing
- Likewise; if the input data contains uncorrelated attribute, i.e., attributes that do not contribute to the learning task, we hope their importance will see itself reduced in the intermediate layers

Internal Representation Example (auto-encoder)



Internal Representation Example (auto-encoder)

Input		Hidden		Output
1000		.00 .14		1000
0100		.09 .93		0100
0010		.98 .99		0010
0001		.96 .06		0001

• If we round to the nearest integer we get a binary code. The network finds this efficient representation

* Example generated with Neural Applet Version 4.3.0 (www.cs.ubc.ca)

Some heuristics and recommendations

- How many training examples?
  - At least 10 times the number of weights un the network
- At most two hidden layers (not any more!!!)
- Watchout for overfitting
  - Use validation set
    - Determine number hidden layers and neurons
    - Determine when to stop training (epochs) (“early stopping”)
    - Regularization and dropout

Some heuristics and recommendations

- Experiment with other transfer functions
  - Tanh is similar to the sigmoid but with values between -1 y 1
  - ReLu which takes the max between 0 y $w^T\mathbf{X}$
- Use different loss functions
  - Cross Entropy

Exercises

1. Train a logistic regression for the AND and XOR problems using Tensorflow
2. Train a ANN for the XOR problem
   - Create a visualization that enables us to view the decision boudaries
3. Train a ANN that identifies points inside a circle (generate the data yourself)
   - Change the number of neurons in the intermediate layer
   - Optional: Plot the error vs the model complexity

Tools

- Tensorflow , Teano, Caffe
- Keras
- Matlab
- R (RSNNS, neuralnet)
- Python: PyBrain, pyfann…
- Many resources on the web
  - Neural Applet
    - http://www.aispace.org/

Extra Sides

Tensorflow

- What is it?
  - A machine learning package mainly used for neuran networks
  - Efficient and scalable
  - CPU, GPU distributed computation
- How to use?
  - Declare variables and operations to perform. This defines a graph in which nodes are operations and edges dataflows. The graph allows for distributed computation
  - Every interacion is handeled through the session manager
    - Call the previousy defined operations
    - The session Is responsible for managing the computing resources

Tensorflow

- Ejercicios
  - Perceptron
  - 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**
- Agreagar 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)