Artificial Neural Networks
Part I

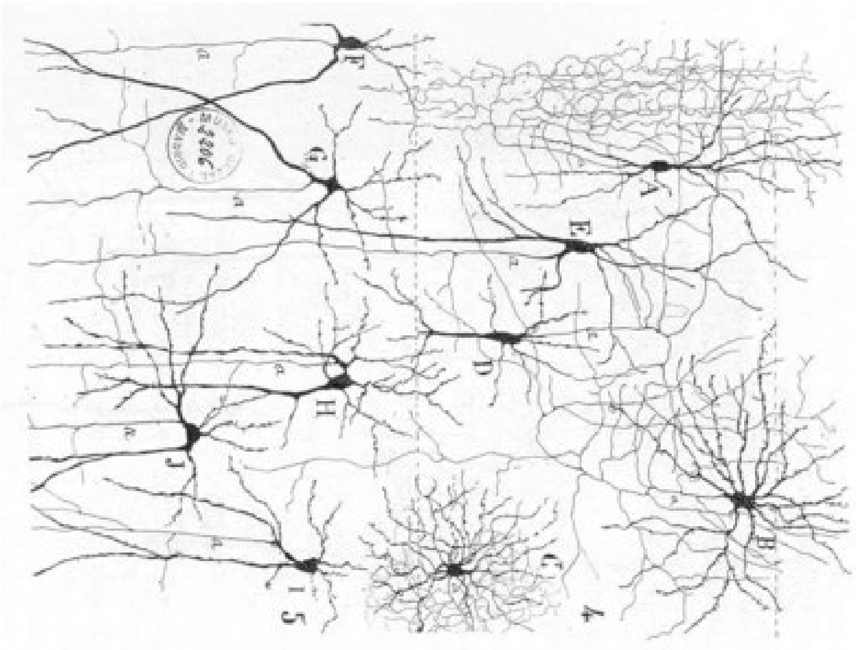
Machine Learning

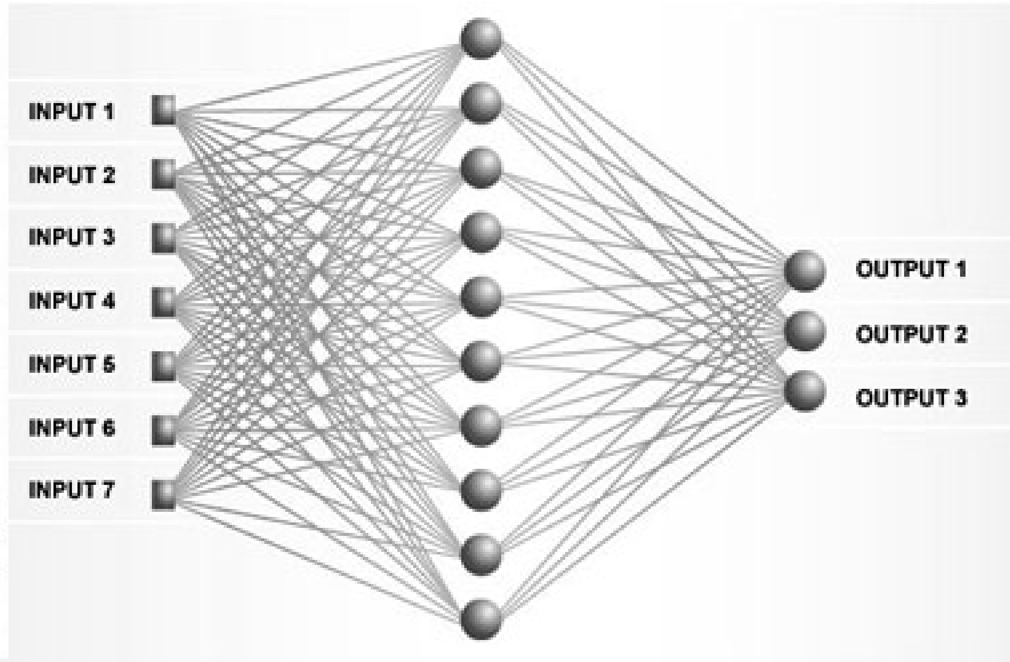
Jürgen Schmidhuber, Cesare Alippi

Michael Wand, Paulo Rauber

TAs: Róbert Csordás, Krsto Proroković,
Xingdong Zuo, Francesco Faccio, Louis Kirsch

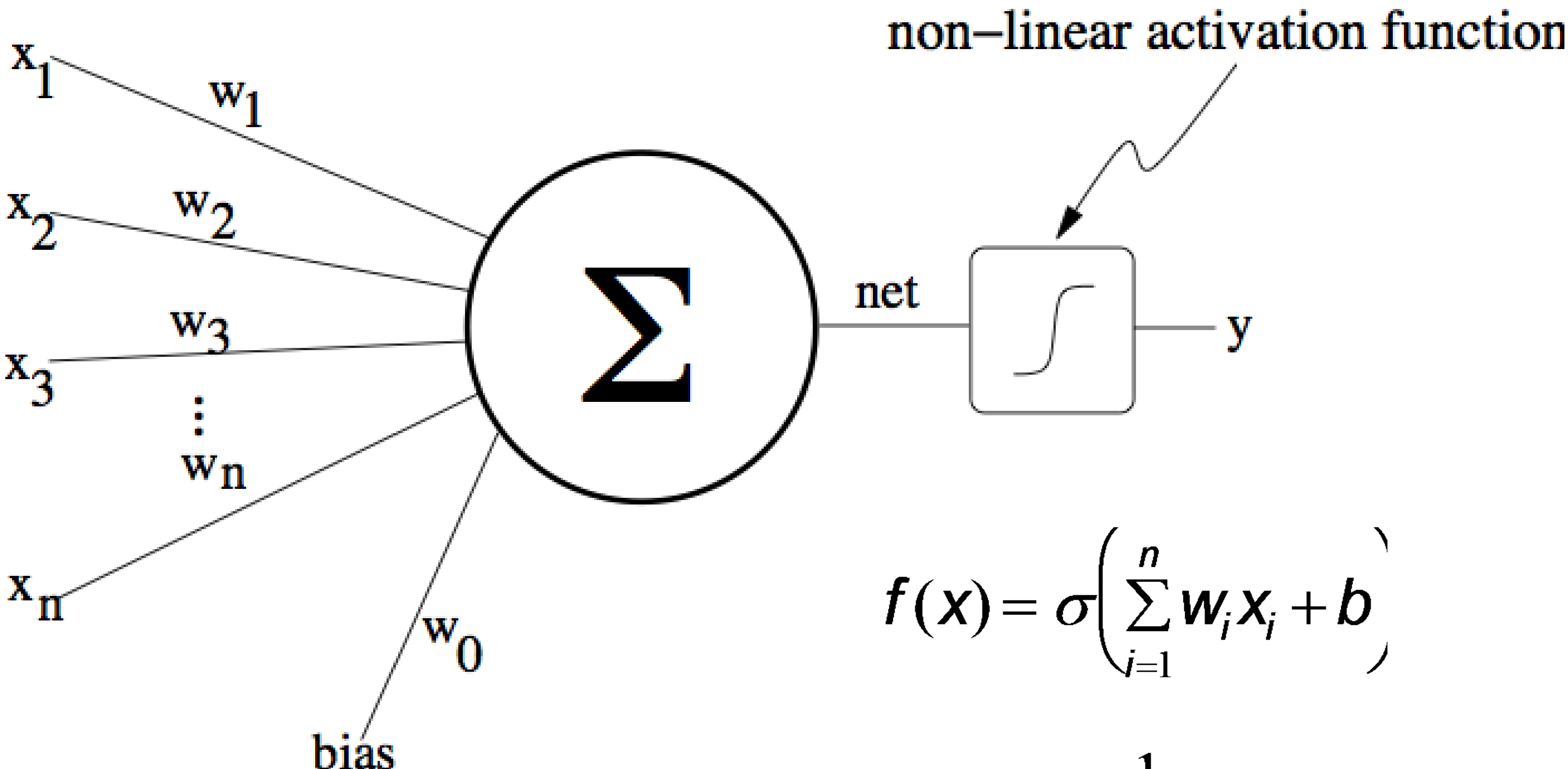
Neural Networks





- Mathematical abstraction of biological nervous systems
- Massively parallel distributed processors
- Subsymbolic (no explicit symbols maintained)
- Universal approximation: can implement arbitrary continuous mappings

Non-Linear Perceptron



$$f(x) = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

$$\sigma(net) = \frac{1}{1+e^{-net}}$$

Training Set

$$\text{example} \equiv [\text{input pattern, target}]$$

$$x^1 = \begin{bmatrix} x_1^1 & x_2^1 & x_3^1 & x_4^1 & x_5^1 \cdots x_n^1, & d^1 \end{bmatrix}$$

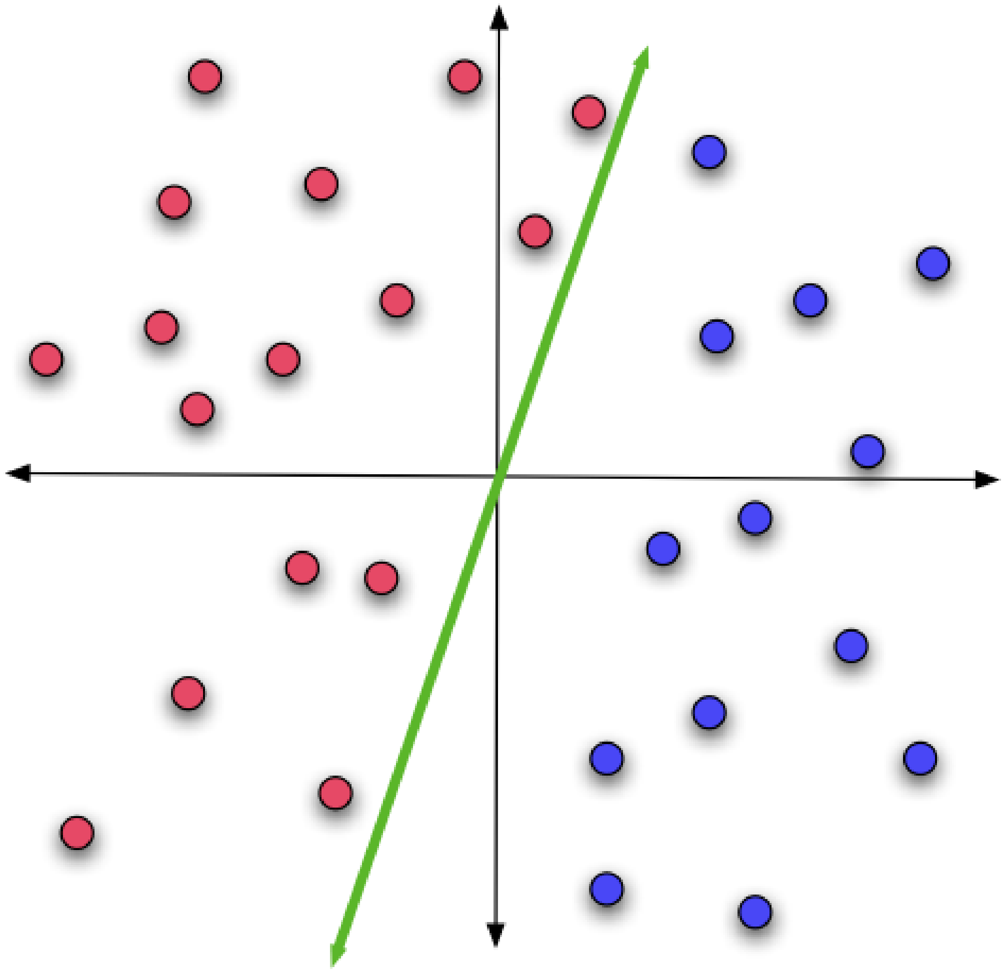
$$x^2 = \begin{bmatrix} x_1^2 & x_2^2 & x_3^2 & x_4^2 & x_5^2 \cdots x_n^2, & d^2 \end{bmatrix}$$

$$x^3 = \begin{bmatrix} x_1^3 & x_2^3 & x_3^3 & x_4^3 & x_5^3 \cdots x_n^3, & d^3 \end{bmatrix}$$

$$\vdots$$

$$x^N = \begin{bmatrix} x_1^P & x_2^P & x_3^P & x_4^P & x_5^P \cdots x_n^P, & d^P \end{bmatrix}$$

Linear Classification



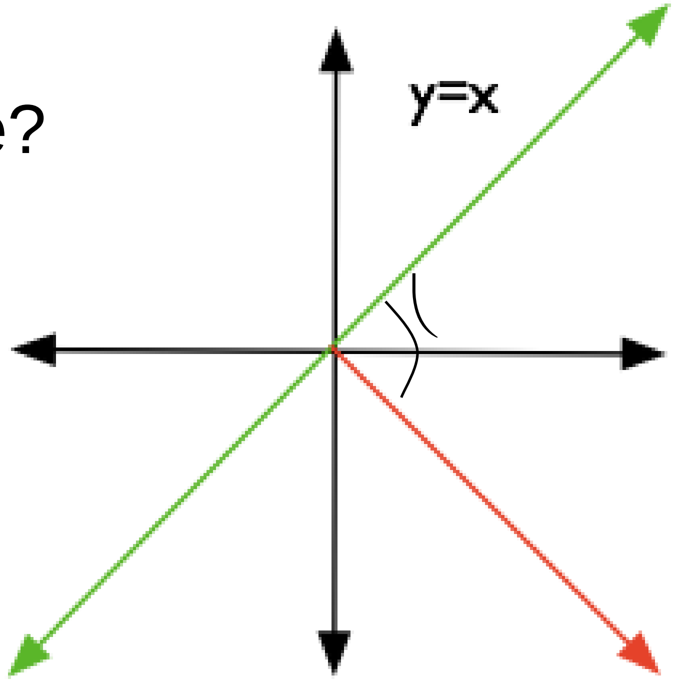
Representing Lines

- How do we represent a line?

$$y = x$$

$$0 = x - y$$

$$0 = [1, -1] \begin{bmatrix} x \\ y \end{bmatrix}$$



In general an ***hyperplane*** is defined by

$$0 = \vec{w} \cdot \vec{x}$$

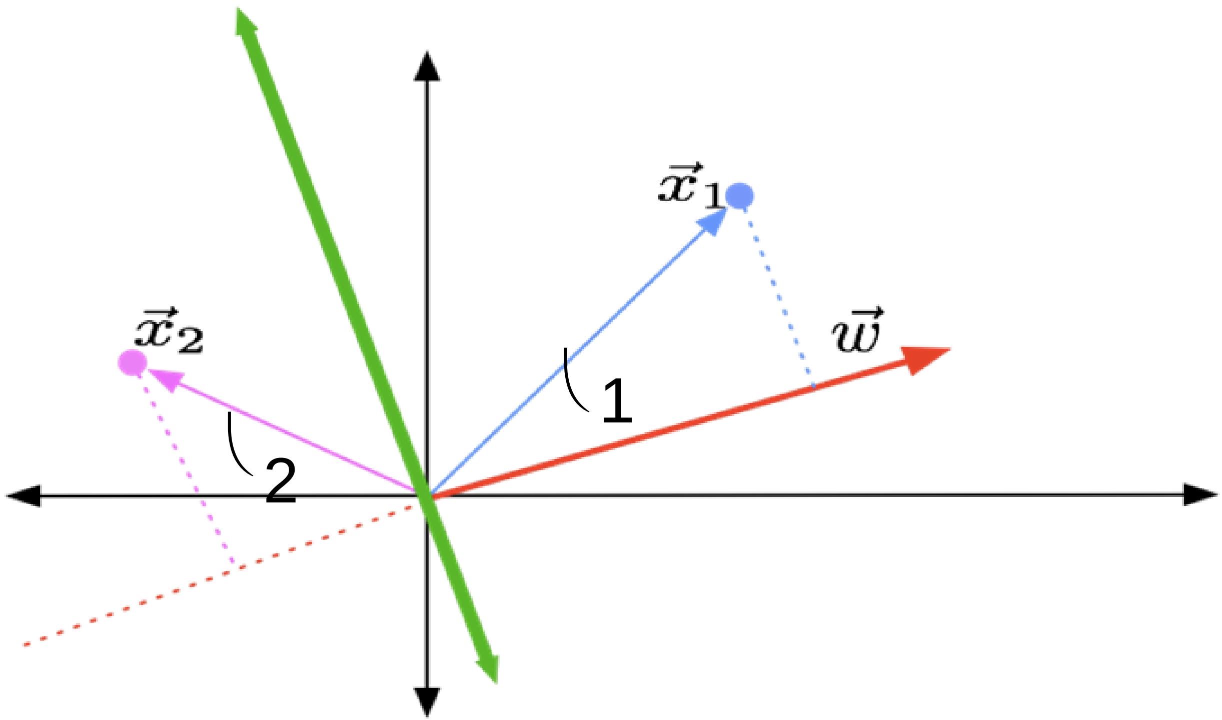
we will use Linear Algebra notation

How to implement Dot Products

- Dot product of two vectors:
  - Elements of vector 1 are multiplied with elements from vector 2, one by one
  - The results are then summed

```
result = 0
for  i  from  0  to length(vector1)
    result += vector1[i] × vector2[i]
return result
```

Dot products on vectors



$x_1 \cdot w$ is positive

$x_2 \cdot w$ is negative

Now classification is easy!

Input encoded as feature vector $\vec{x}$

Model encoded as $\vec{w}$

$$\text{Just return } y = \vec{w} \cdot \vec{x}!$$

But... how do we learn this mysterious model vector?

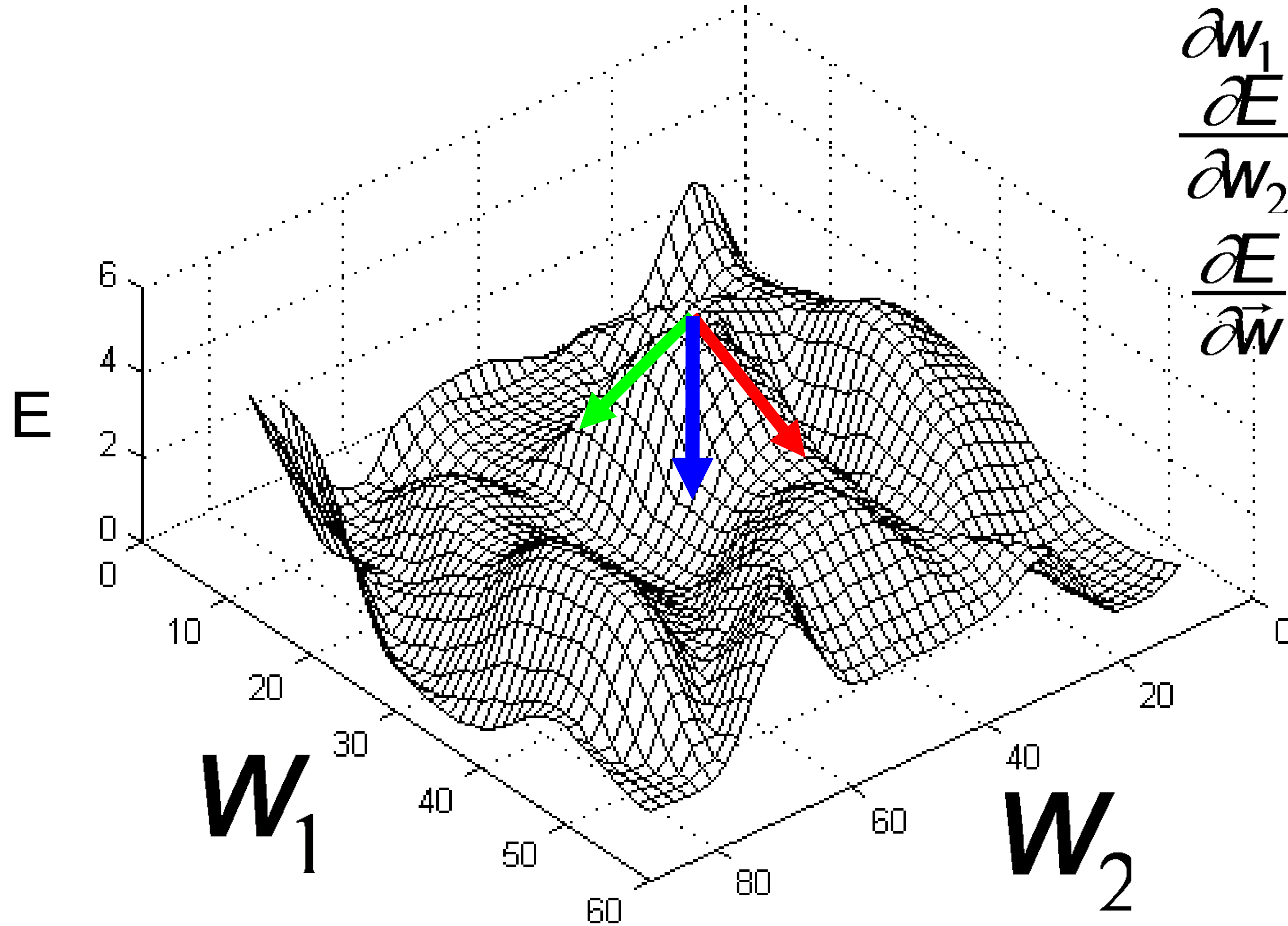
Delta Rule

- Gradient descent for linear neuron
- Calculate squared error between output and target

$$E = \frac{1}{2}(d - net)^2 = \frac{1}{2}(d - \sum_{i=1}^{n} w_i x_i)^2$$

Its value is determined by the weights w_I

Error surface



$$\frac{\partial E}{\partial w_1}$$

$$\frac{\partial E}{\partial w_2}$$

$$\frac{\partial E}{\partial \vec{w}}$$

Delta Rule

- Gradient descent for linear neuron
- Calculate squared error between output and target

$$E = \frac{1}{2}(d - net)^2 = \frac{1}{2}(d - \sum_{i=1}^{n} w_i x_i)^2$$

Its value is determined by the weights w_l

- Obtain the partial derivative of error with respect to each weight

$$\frac{\partial E}{\partial w_i} = (net - d)\frac{\partial net}{\partial w_i} = (net - d)x_i$$

- Change weights in the opposite direction of $\partial E / \partial w_i$:

$$\begin{array}{ll} (net - d)x_i & \text{increases error} \\ -(net - d)x_i & \text{decreases error} \end{array}$$

$$\Delta w_i = \eta(d - \sum_{i=0}^{n} w_i x_i)x_i = \eta(d - net)x_i$$

Generalization and testing

- Training set: the data you train on
- Testing set: the data you test generalization with

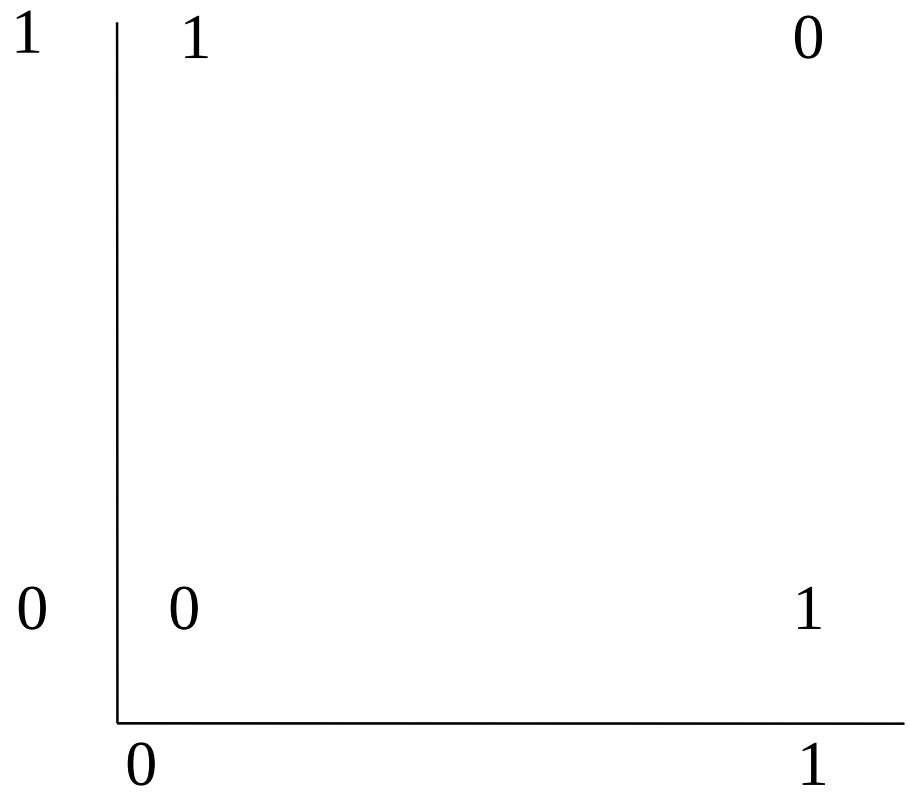
  Typically, remove a random subset from training set and use as test set

- Classification error: number of data points misclassified
- Regression error: error on training...
- Generalization error: ...and test set

Problem: some functions are not linearly separable!

u1	u2	u1 XOR u2
0	0	0
0	1	1
1	0	1
1	1	0

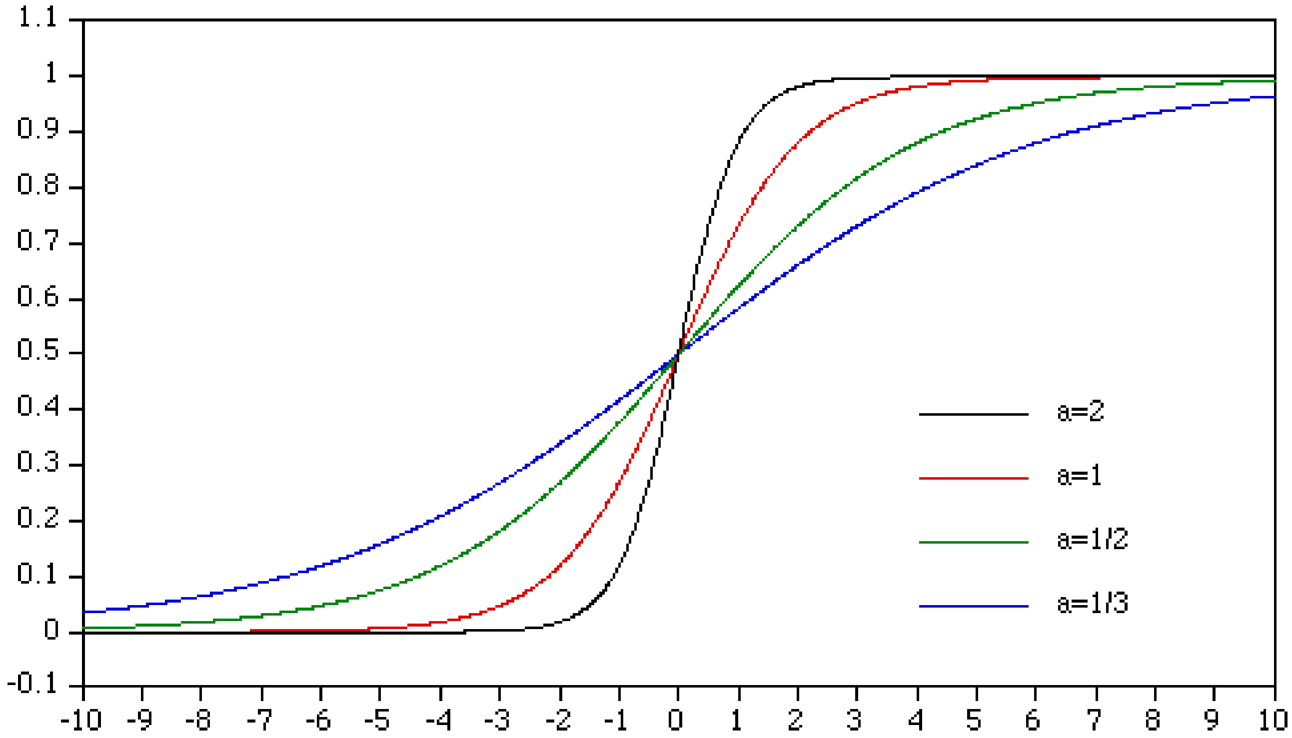
XOR function



Since XOR (a simple function) could not be separated by a line the perceptron is very limited in what kind of functions it can learn. US funding for neural networks dried up for more than a decade after Minsky and Papert book *Perceptrons (1969)*.

Non-Linear Threshold

$$\sigma(y) = \frac{1}{1+e^{-ay}}$$

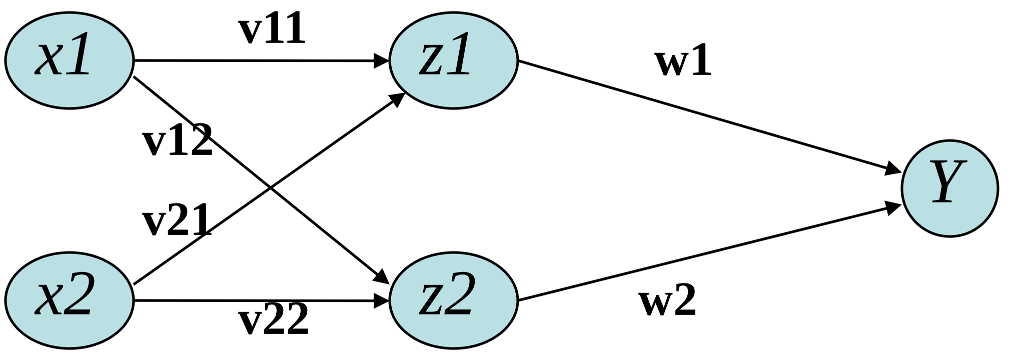


$x.w+b$ (weighted sum of inputs)

- can now benefit from combining perceptrons (neurons) into layered networks

Why hidden units must be non-linear?

Multi-layer net with linear hidden layers is equivalent to a single layer net



– Because z1 and z2 are linear units

$$z_1 = x_1 \cdot v_{(1,1)} + x_2 \cdot v_{(2,1)}$$

$$z_2 = x_2 \cdot v_{(1,2)} + x_2 \cdot v_{(2,2)}$$

$$y = z_1 \cdot w_1 + z_2 \cdot w_2$$

$$= x_1 \cdot u_1 + x_2 \cdot u_2$$

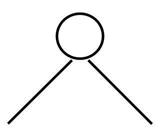
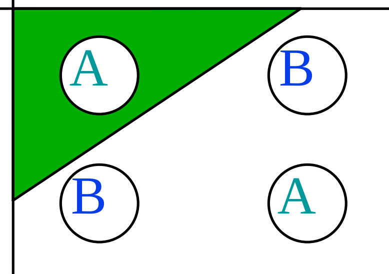
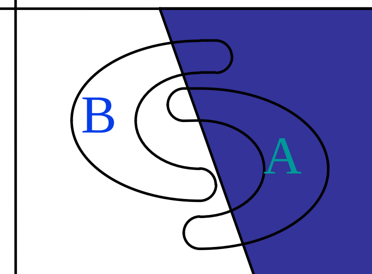
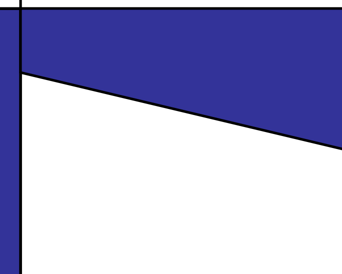
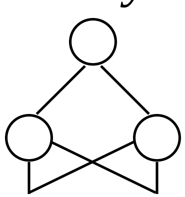
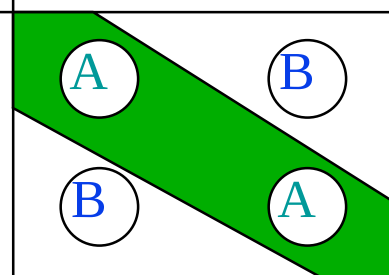
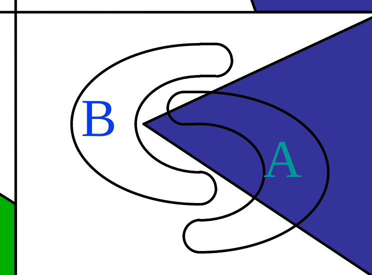
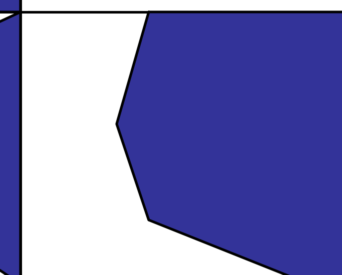
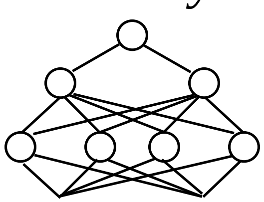
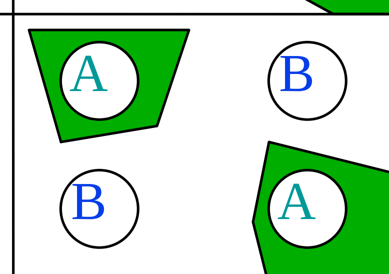
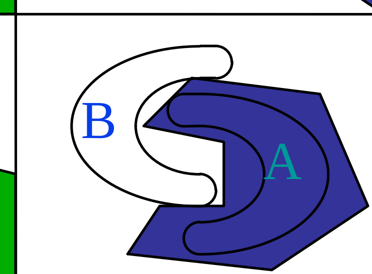
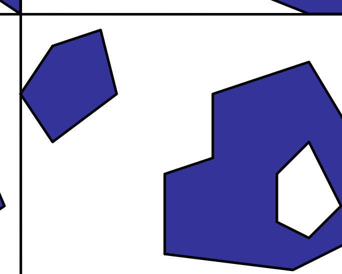
Where

$$u_1 = (v_{(1,1)} w_1 + v_{(1,2)} w_2)$$

$$u_2 = (v_{(2,1)} w_1 + v_{(2,2)} w_2)$$

therefore the output y is still a linear combination of x1 and x2.

Non linearly separable problems

Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Classes with Meshed regions*	*Most General Region Shapes*
Single-Layer 	*Half Plane Bounded By Hyperplane*			
Two-Layer 	*All piecewise continuous functions*			
Three-Layer 				

Types of supervised learning

- Regression: function approximation
- Classification: classify input data

All function approximators that can do regression can also do classification (just introduce a condition, e.g. $y > 0$)

Generalization

- Can a trained perceptron correctly classify patterns not included in the training samples?

  □Depends on the quality of training samples selected.

  □Also to some extent depends on the learning rate and initial weights

- How can we know if the learning is ok?

  □Reserve a few samples for testing.

Solving XOR with 2-layer perceptron

XOR can be composed of ‘simpler’ logical functions.

A xor B = (A or B) and not (A and B)
The last term simply removes the troublesome value.