UPPSALA
UNIVERSITET

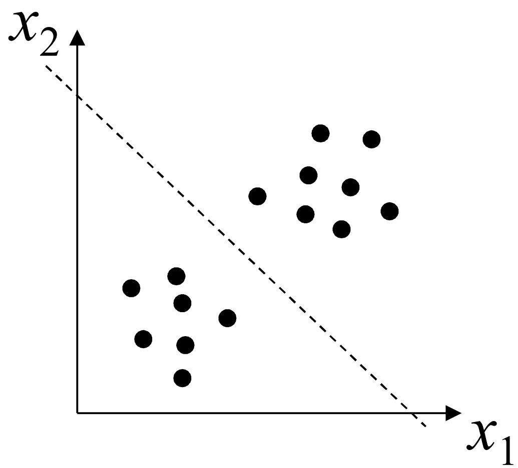
Natural Computation Methods in Machine Learning (NCML)

Lecture 3: The Perceptron

Pattern recognition

- Pattern recognition = Feature extraction + Classification
- Feature extraction: To find 'good' features
  → feature vector, $\overline{x}$
- Very sensitive to assumptions!
  - What's the most significant features to extract, to recognize someone's face?
  - prominent ≠ significant
  - Lesson's learned from a masquerade ball

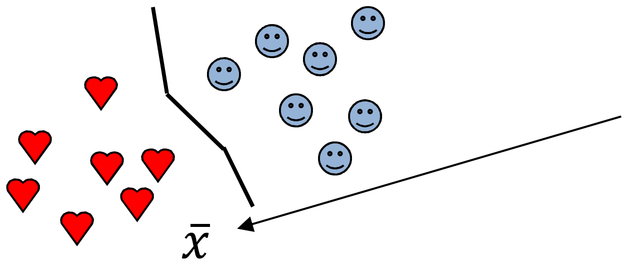
Classification



- Example: Measure two features
  - 2D feature vector, $\overline{x} = (x_1, x_2)$
- Classification: To find a *discriminant* that separates the classes, in the feature space (in this case 2D)
  - In the example, a line would suffice
  - Discriminants can take any shape
  - There is usually an infinite number of solutions (possible discriminants), even if restricted to a certain shape

Nearest neighbour classifiers

- Classify the unknown sample (vector/pattern) to the class of its closest previously classified neighbour



The new pattern, $\bar{x}$, will be classified as a ♥.

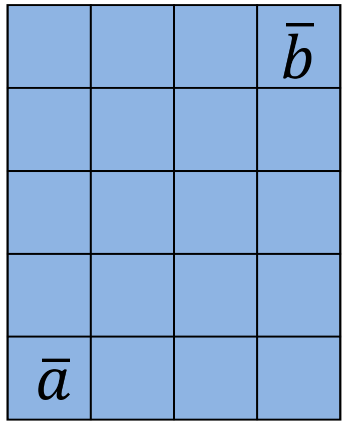
- Problem: The closest neighbour may be an outlier from the other class
- Solution: *K*-nearest-neigbour (KNN) – classify $\bar{x}$ to the most common class among its *K* closest neighbours
- *Q: No explicit mention of finding discriminants, but there must be one. What's its shape?*
- *Q: Is this supervised or unsupervised learning?*

UPPSALA UNIVERSITET

Distance measures

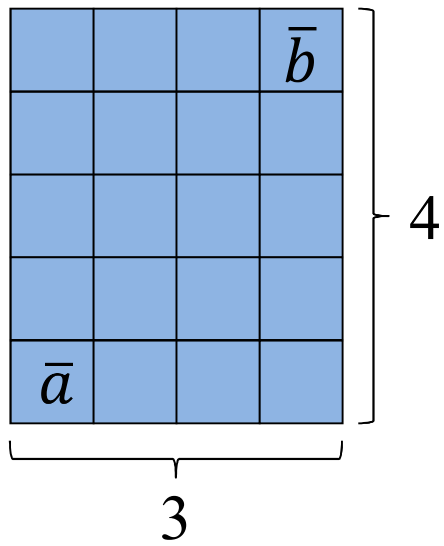
- Define distance between two vectors
  - $\bar{a} = [a_1, a_2, \dots, a_n]$ and $\bar{b} = [b_1, b_2, \dots, b_n]$
- General: the l_p norm of a vector, $\bar{x} = |\bar{a} - \bar{b}|$

$$l_p(\bar{x}) = \left(\sum_i x_i{}^p\right)^{\frac{1}{p}}$$



- Two well known special cases
  - p=2 → l_2 norm = Euclidean distance
  - p=1 → l_1 norm = city block (Manhattan) distance

l_p norm examples



$$l_p(\bar{x}) = \left(\sum_i x_i{}^p\right)^{\frac{1}{p}}$$

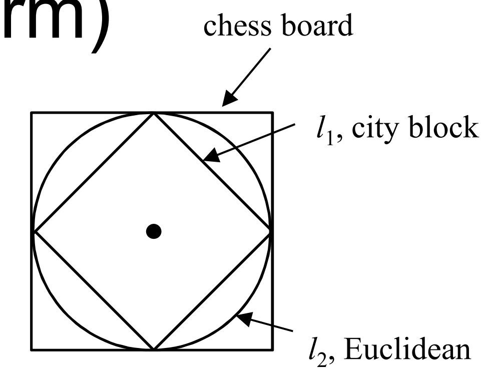
- Euclidean distance (l_2 norm)

$$d_{euc}(\bar{a},\bar{b}) = l_2(|\bar{a}-\bar{b}|) = \sqrt{\sum_i |a_i - b_i|^2} = \sqrt{3^2+4^2} = \sqrt{25} = 5$$

- City block (Manhattan) distance (l_1 norm)

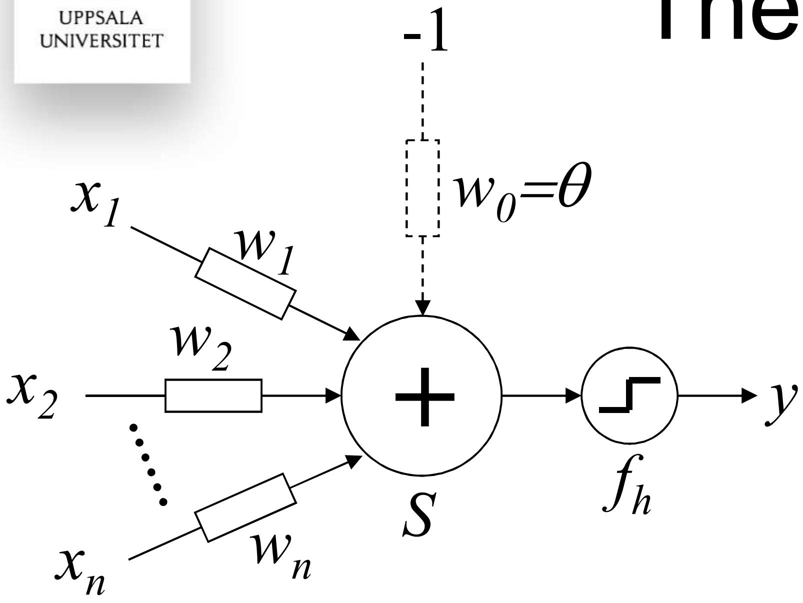
$$d_{cb}(\bar{a},\bar{b}) = l_1(|\bar{a}-\bar{b}|) = \sum_i |a_i - b_i| = 3+4 = 7$$

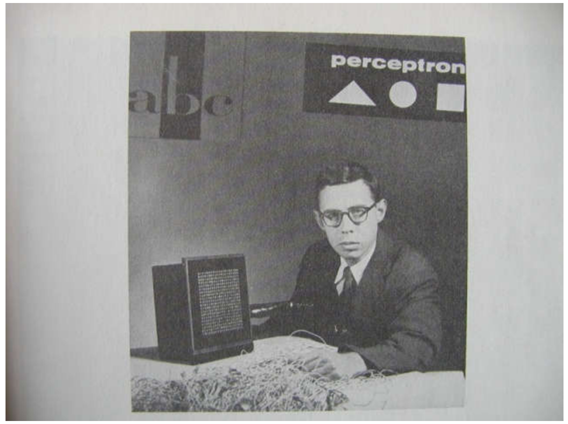
- Affects shape of equi-distant border



The (binary) Perceptron

Rosenblatt 1958





Q: Here, x_0=-1, but this would work for any value of x_0, positive or negative. Why?

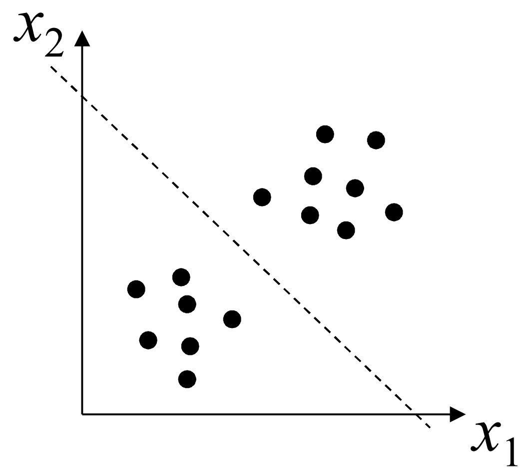
$$y = f_h(S) = \begin{cases} 1, \text{if S} > 0 \\ 0, \text{if S} \leq 0 \end{cases}$$

$$S = \sum_{i=1}^{n} w_i x_i - \theta \ = \underbrace{\sum_{i=0}^{n} w_i x_i, \text{ where } \begin{cases} x_0 = -1 \\ w_0 = \theta \end{cases}}_{\textit{Augmented vector notation}}$$

UPPSALA UNIVERSITET

Hyperplanes

- The perceptron defines a hyperplane!
- 2D (hyperplane=line) example:



$$S = \sum_{i=1}^{2} w_i x_i - \theta = w_1 x_1 + w_2 x_2 - \theta$$

- The perceptron 'flips' at S=0
  - so, set S=0 and solve for x_2

$$w_1 x_1 + w_2 x_2 - \theta = 0$$

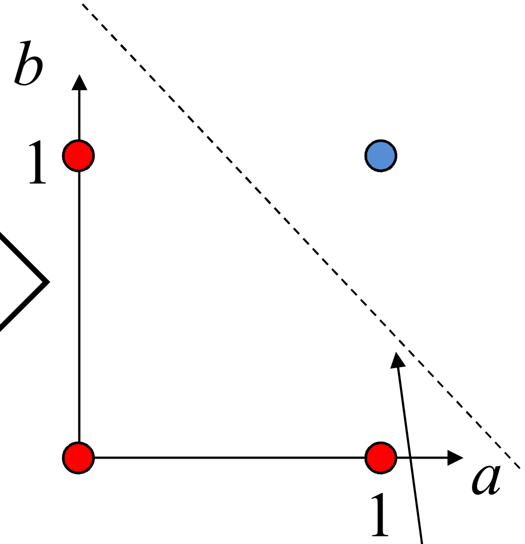
$$x_2 = \frac{\theta - w_1 x_1}{w_2} = -\frac{w_1}{w_2} x_1 + \frac{\theta}{w_2} \qquad (= kx+m \ldots a \text{ line!})$$

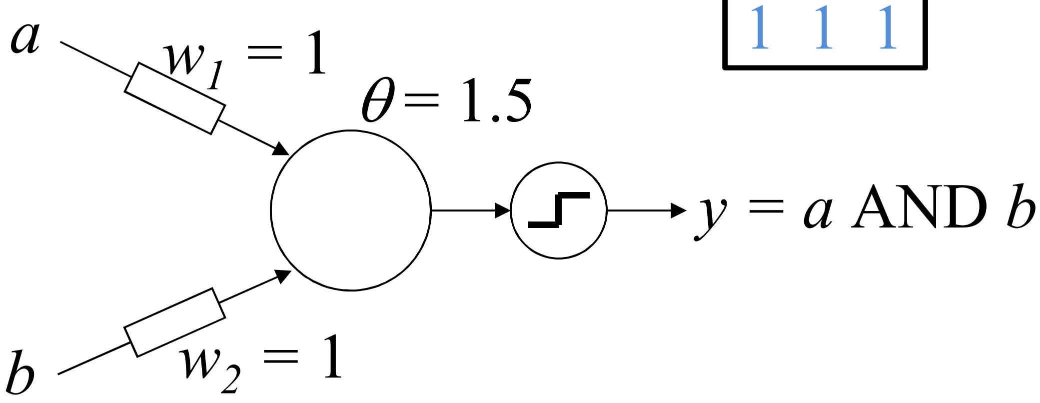
$\therefore$ The weights define the position and slope of a hyper plane!

Remember the AND gate?

AND

a	b	y
0	0	0
0	1	0
1	0	0
1	1	1





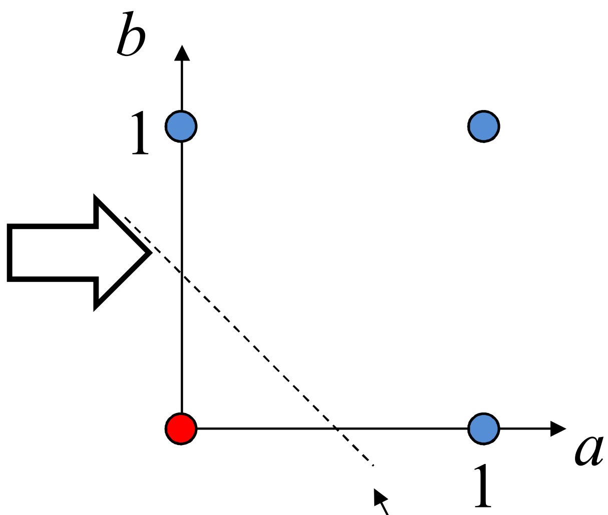
$$b = -\frac{w_1}{w_2}a + \frac{\theta}{w_2} = -\frac{1}{1}a + \frac{1.5}{1} = -a + 1.5$$

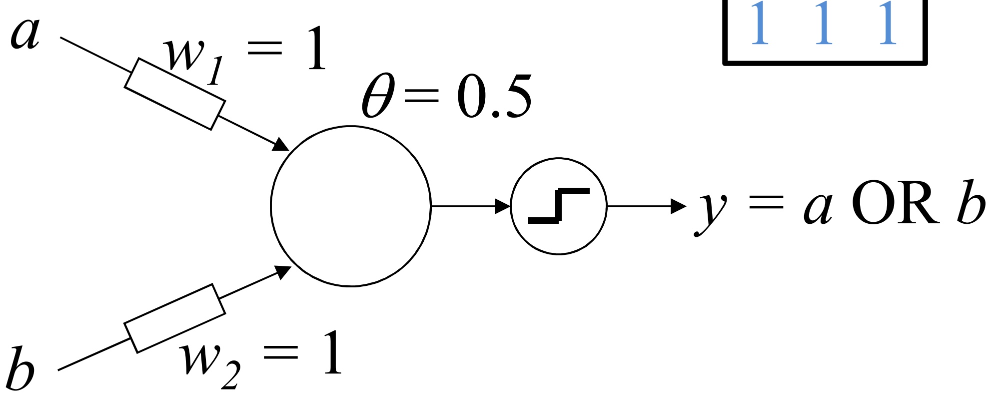
Q: What about NAND?

What about the OR?

OR

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1





$$b = -\frac{w_1}{w_2}a + \frac{\theta}{w_2} = -\frac{1}{1}a + \frac{0.5}{1} = -a + 0.5$$

Adjusting the line automatically

Training

- We need a number of pairs $(\overline{x}, d)$ of feature vectors $(\overline{x})$ and desired responses (d)
- For each such pair, and perceptron output $y(\overline{x})$:
  - If $y=d$, do nothing
  - If $y=0$, $d=1$: Reinforce the connections (to increase S)
  - If $y=1$, $d=0$: Weaken the connections (to decrease S)
- Reinforce/weaken = add/subtract x_i to/from w_i
  - note that this will leave the weight unchanged if $x_i=0$

    (makes sense since 0 inputs do not contribute to the error)

Perceptron Convergence Procedure

PCP, Rosenblatt 1958

w_i, $0 \le i \le n$, the weight from input i

x_i, $0 \le i \le n$, the value of input i

$x_0 = -1$ (or any other constant value)

1) Initialize the node: Set all weights, w_i, to small random values
2) Present a feature vector $\bar{x} = [x_1, x_2, \dots, x_n]$ and a desired response, d
3) Compute perceptron output: $y = f_h(S) = \begin{cases} 1, S > 0 \\ 0, S \le 0 \end{cases}$ $\quad S = \sum_{i=0}^{n} w_i x_i$
4) Adjust weights:

$$w_i(t+1) = w_i(t) + \Delta w_i$$

$$\Delta w_i = \begin{cases} 0, \text{ if } y = d \\ +x_i, \text{ if } y = 0,\ d = 1 \\ -x_i, \text{ if } y = 1,\ d = 0 \end{cases}$$

5) Repeat from 2) with a new vector

Perceptron Convergence Procedure

slightly rewritten, now with a throttle/brake (η)

w_i, $0 \leq i \leq n$, the weight from input i

x_i, $0 \leq i \leq n$, the value of input i

x_0 = -1 (or any other constant value)

1) Initialize the node: Set all weights, w_i, to small random values
2) Present a feature vector $\bar{x} = [x_1, x_2, \ldots, x_n]$ and a desired response, d
3) Compute perceptron output: $y = f_h(S) = \begin{cases} 1, S > 0 \\ 0, S \leq 0 \end{cases}$ $\quad S = \sum_{i=0}^{n} w_i x_i$
4) Adjust weights:

$$w_i(t+1) = w_i(t) + \Delta w_i$$

$$\boxed{\Delta w_i = \eta \delta x_i,} \text{ where } \delta = d - y$$

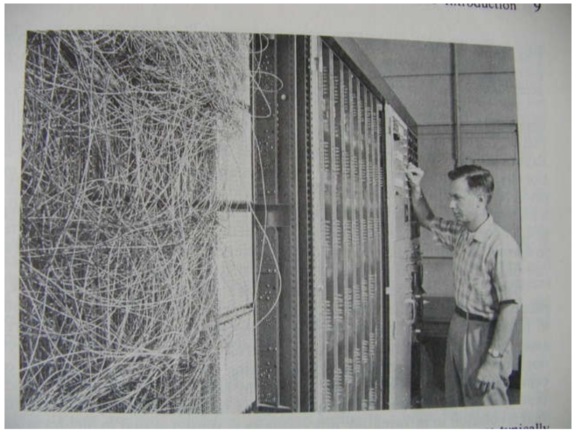
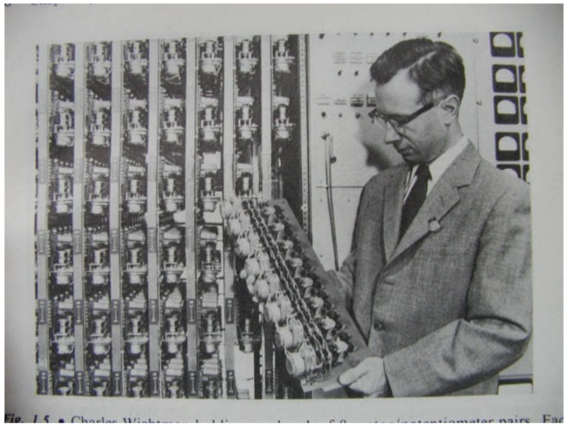
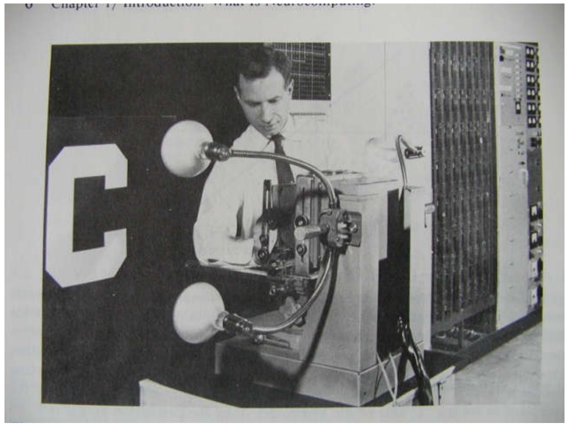
Many update rules can be written like this, only the definition of δ differs.

5) Repeat from 2) with a new vector

UPPSALA UNIVERSITET

Perceptron Convergence Theorem
PCT

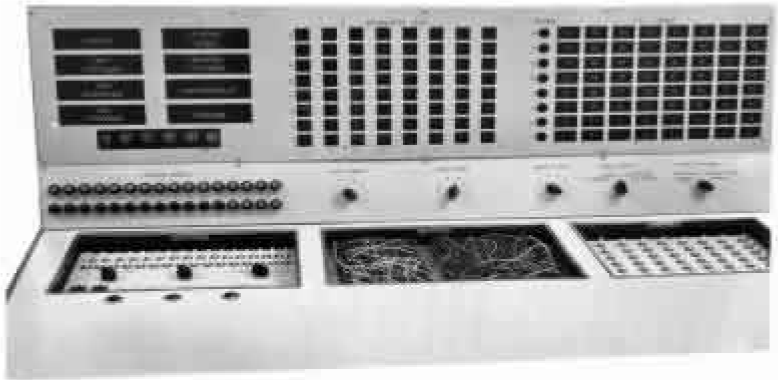
- The Perceptron Convergence Procedure converges to an optimal discriminant in a finite number of steps, if such a discriminant exists
- PCP + PCT ➔ The boom in the 1960's

Mark I (1957): One the first neurocomputers (the first based on Perceptrons)

Adam

The first European digital neurocomputer

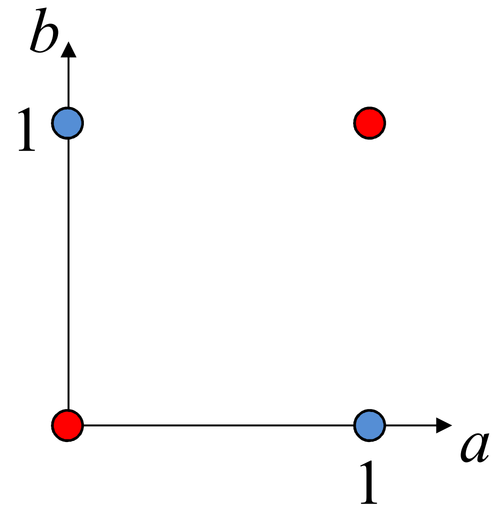


Perceptron Convergence Theorem

PCT

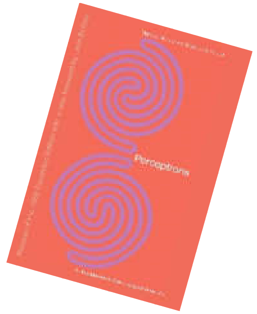
- The Perceptron Convergence Procedure converges to an optimal discriminant in a finite number of steps, ***if such a discriminant exists***
  - Problem: It seldom does! Few interesting classification problems are separable by one linear discriminant (a hyperplane)
  - Simple example: XOR

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0



Sudden death of the first boom

- Minsky & Papert, *Perceptrons*, 1969

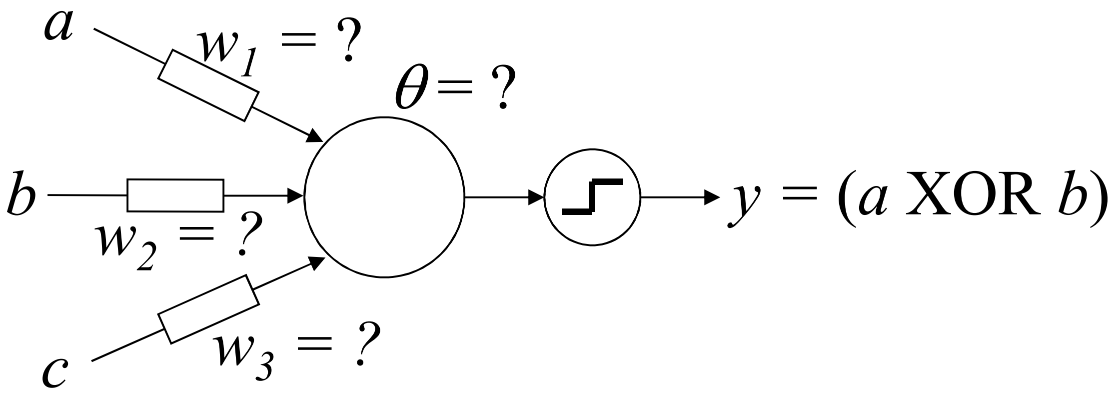


- Old news to the researchers, but a shock to the funding agencies ...
- No grants → Mass exodus to other fields
  - Bernard Widrow, for example, removed the main problem (the step function) and thus invented the adaptive filter
- Neural winter, 1969 – early 1980's

What can we do?

Solution 1: Linearize the problem

- Increase dimensionality – add more inputs
  - For example, XOR becomes linearly separable if we add an extra input c which is 'a AND b'



XOR(a,b)

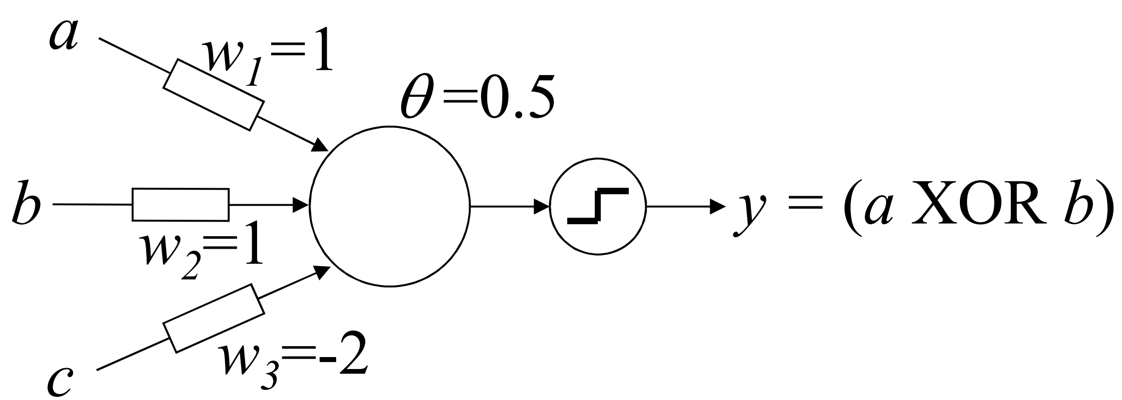
a	b	c	y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Important (and often used) general principle: Complex problems may become easier by first projecting them to a higher-dimensional space

Challenge: Geometrical explanation why this makes XOR linearly separable

What can we do?
Solution 1: Linearize the problem

- Increase dimensionality – add more inputs
  - For example, XOR becomes linearly separable if we add an extra input c which is 'a AND b'



XOR(a,b)

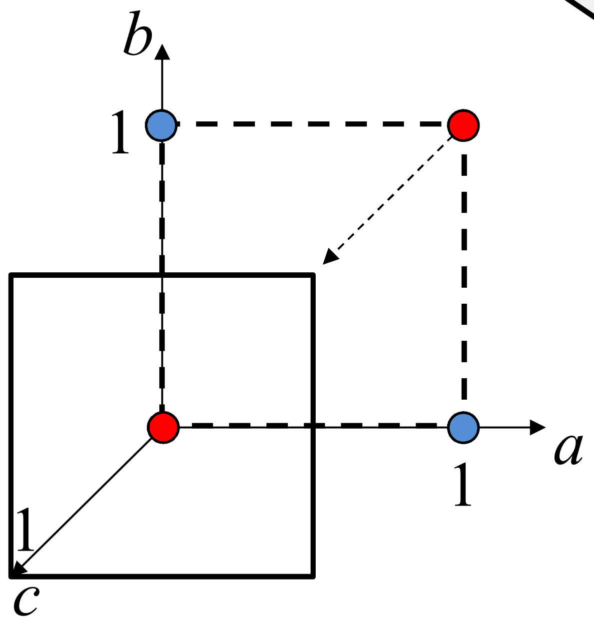
a	b	c	y
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Important (and often used) general principle: Complex problems may become easier by first projecting them to a higher-dimensional space

Challenge: Geometrical explanation why this makes XOR linearly separable

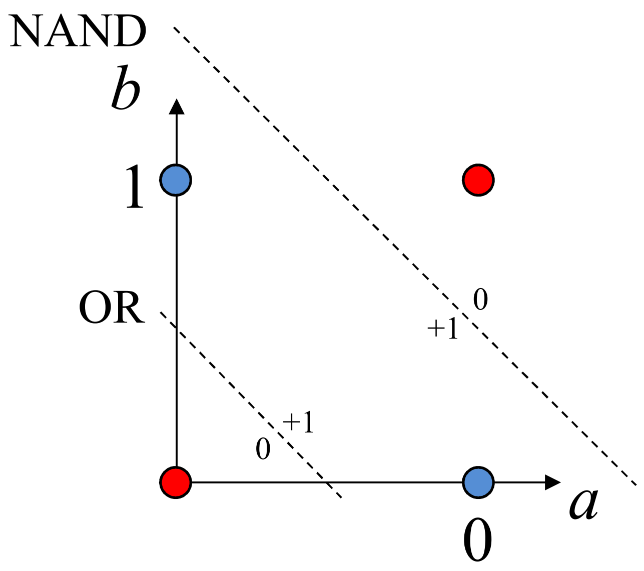
UPPSALA UNIVERSITET

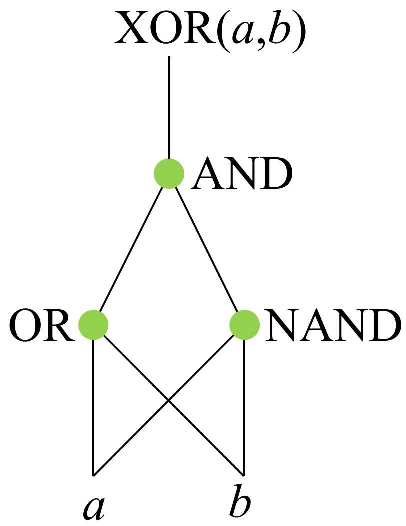
…OR 2D→3D



What can we do?

Solution 2: Combine perceptrons in layers





But how do we train such a network? (next lecture)