Neural Networks

Professor Michael C. Mozer

CSCI 3202

DICK TRACY

YOUR BROTHER WAS WORKING ON A "NEURAL-NETWORK" COMPUTER ? YOU MEAN, ARTIFICIAL INTELLIGENCE ?
AH-YOU DO KEEP UP, DON'T YOU MR. TRACY ?
1-27
"YES - A COMPUTER THAT CAN LEARN THROUGH EXPERIENCE-"

Brains vs. Computers

Tasks that are easy for brains are not easy for computers, and vice-versa.

Brains:

- recognizing faces
- retrieving information based on partial descriptions
- organizing information (the more info, the better the brain operates)

Computers:

- arithmetic
- deductive logic $(p \to q \;\&\; \neg q) \to \neg p$
- retrieving information based on arbitrary features

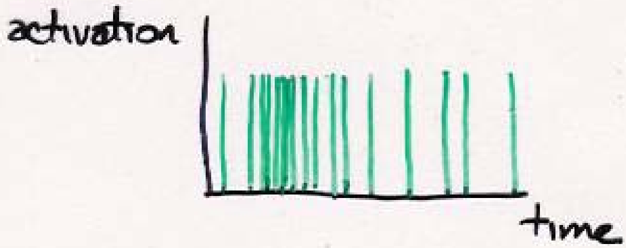
$\Rightarrow$ Brains must operate very differently from conventional computers.

How the brain operates (with apologies to neuroscientists)

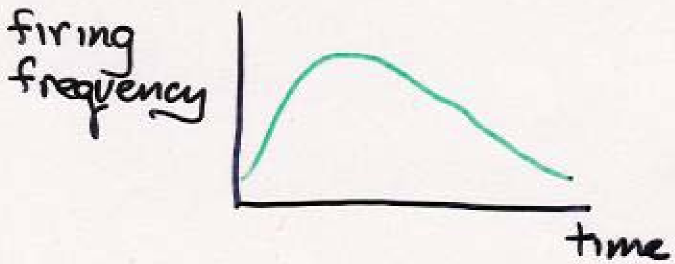
Brain is composed of neurons

Neurons convey and transform information

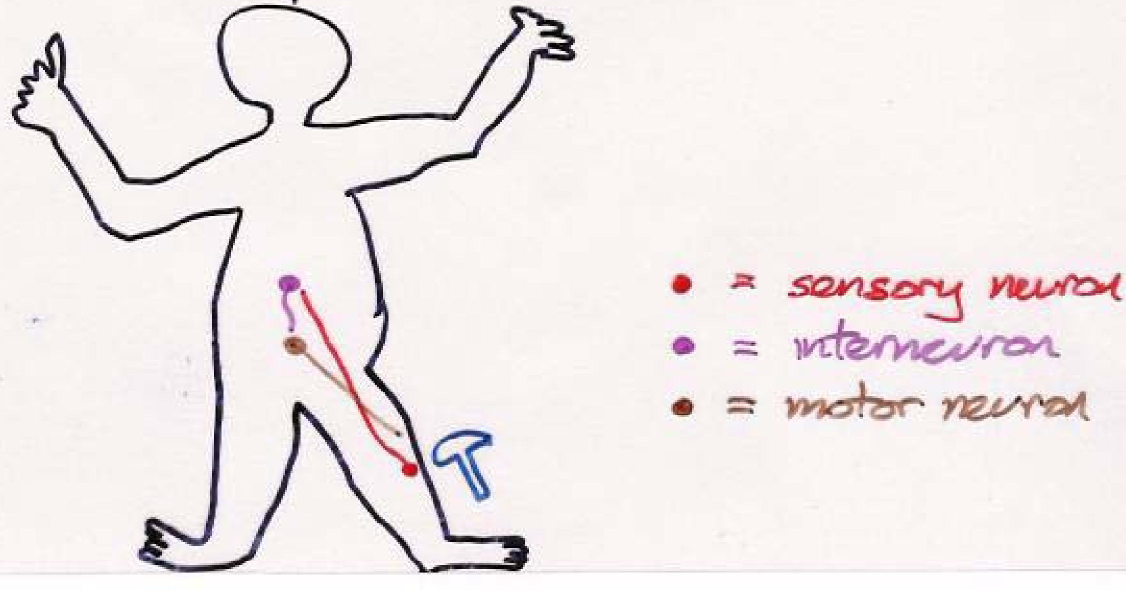
What is this information? "activation" (a scalar)



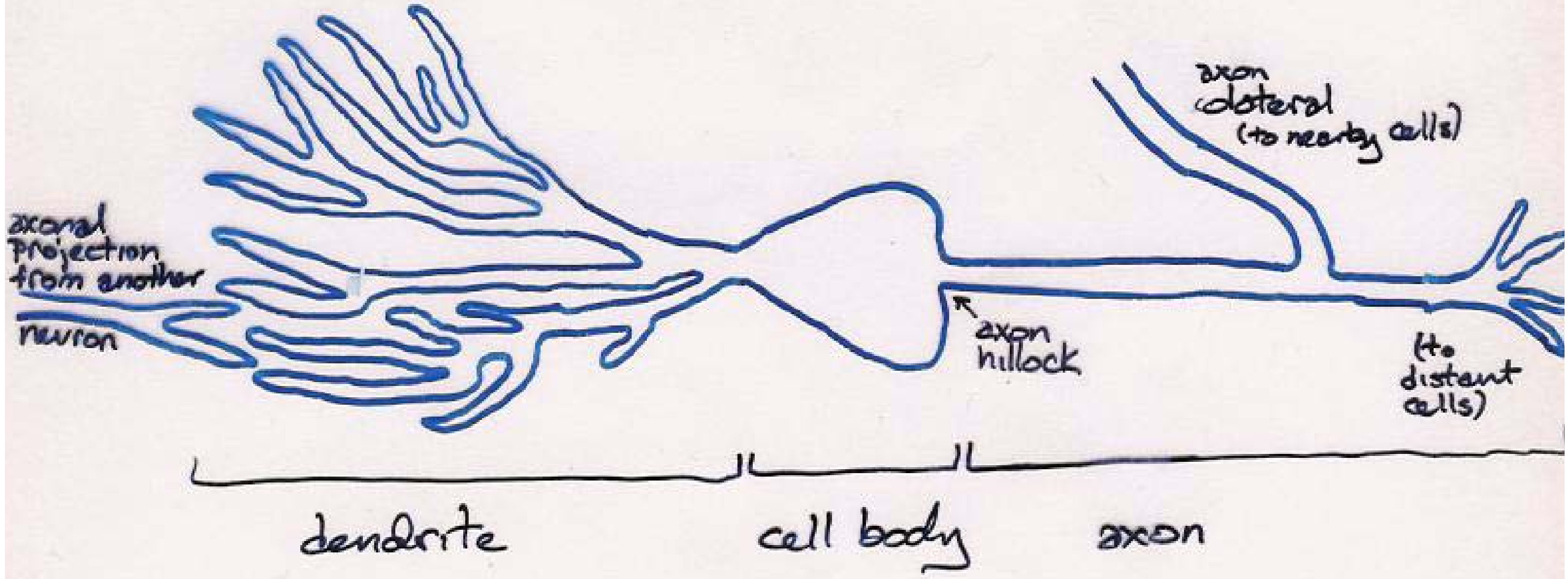
Averaging instantaneous binary-valued activity →



Cartoon sketch of simple neural circuit:



Generic (cortical) neuron



dendrite: receive signals from other neurons; integrate signals

cell body: make decision

axon: pass signal to other neurons

Gross oversimplification:

— many types of neurons

— electrical and chemical interactions

— many types of connections (e.g., dendrodendritic)

Properties of real neurons used in connectionist modeling

- Neurons are slow (10^{-3} - 10^{-2} sec propagation time)
- Large number of neurons (10^{10} - 10^{11})
- No central controller (CPU)
- Neurons receive input from a large number of other neurons (10^4 fan-in and fan-out in cortex)
- Communication via excitation and inhibition
- Statistical decision making (neurons that single-handedly turn on/off other neurons are rare)
- Learning involves modifying connection strengths (the tendency of one cell to excite/inhibit another)
- Neural hardware is dedicated to particular tasks (vs. conventional computer memory)

Conventional computers:

One very smart CPU

Lots of extremely dumb memory cells

Brains, connectionist computers:

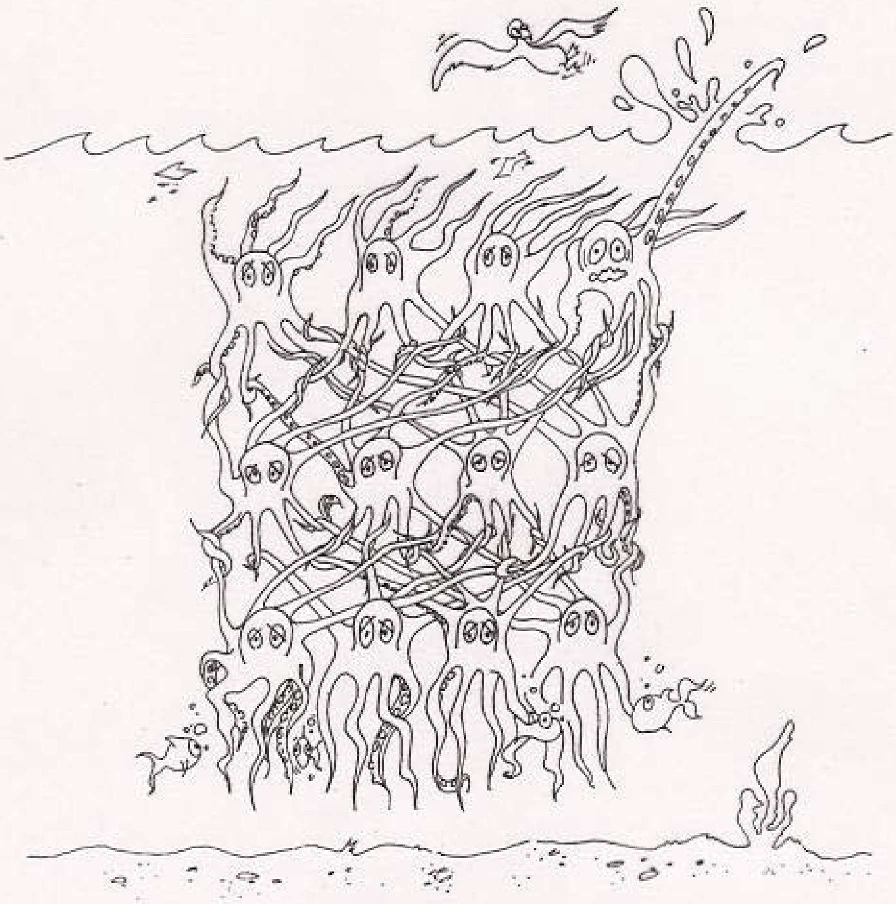
No CPU

Lots of slightly smart memory cells

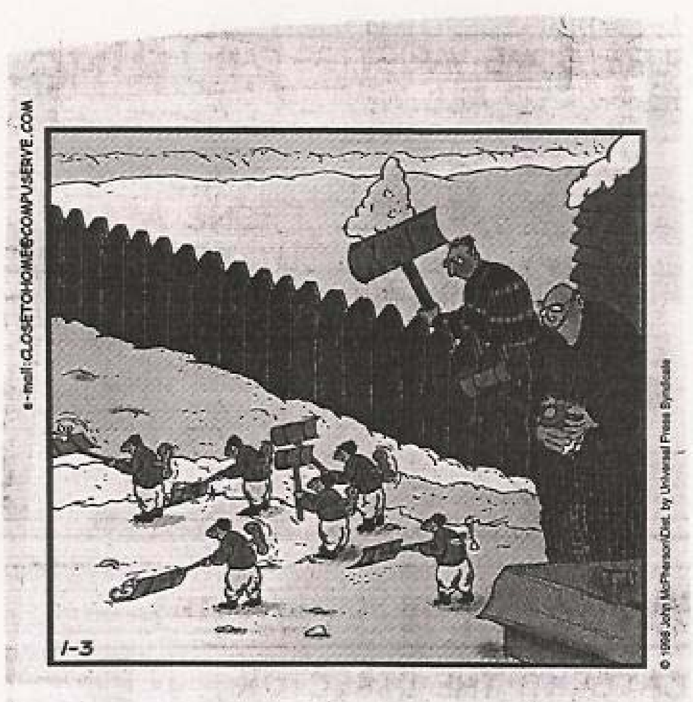
⇒ connectionism = "brain-style" computation or "neurally-inspired" computation

the neighborhood™ Jerry Van Amerongen

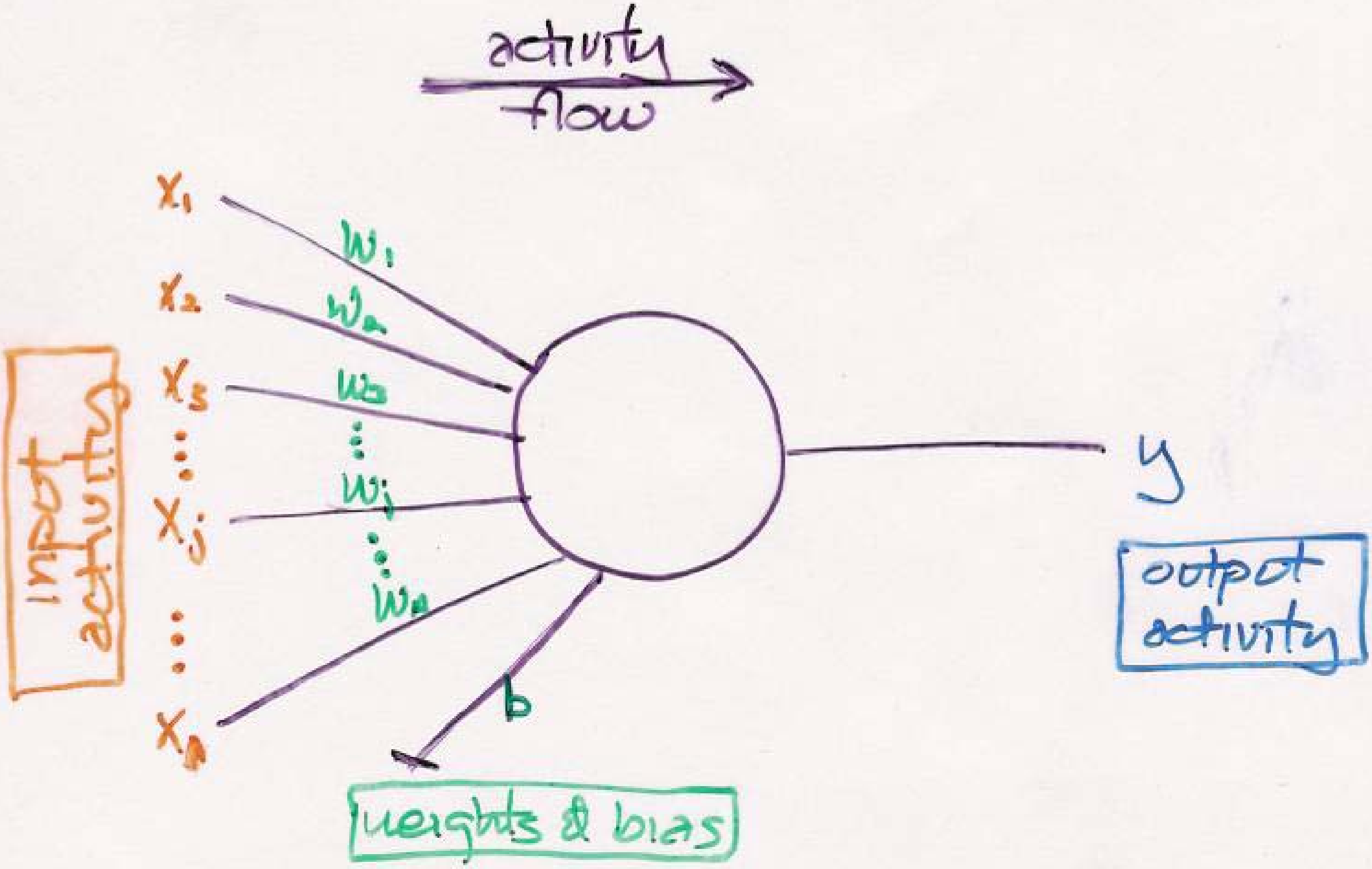
how the brain works



A computational network, as hungry octopi might design it. From *Wet Mind*.



Typical unit in a neural network



Activation function

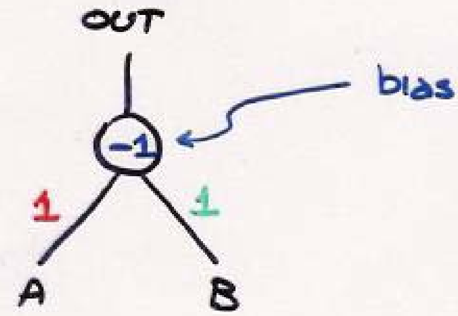
$$net_i = \sum_j w_{ij} x_j + b_i$$

$$y_i = f(net_i)$$

Examples using binary threshold unit

AND

A	B	OUT
0	0	0
0	1	0
1	0	0
1	1	1

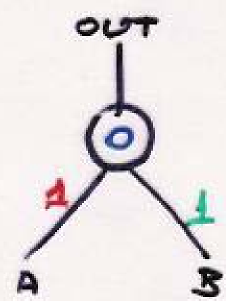


$$net = 1 \times A + 1 \times B - 1 = A + B - 1$$

$$(out = 1 \text{ if } net > 0 \text{ ; } 0 \text{ otherwise})$$

OR

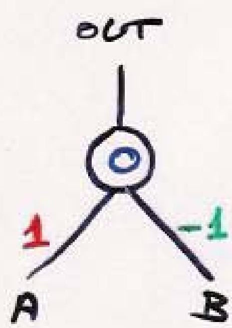
A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	1



$$net = 1 \times A + 1 \times B + 0 = A + B$$

A NOT B

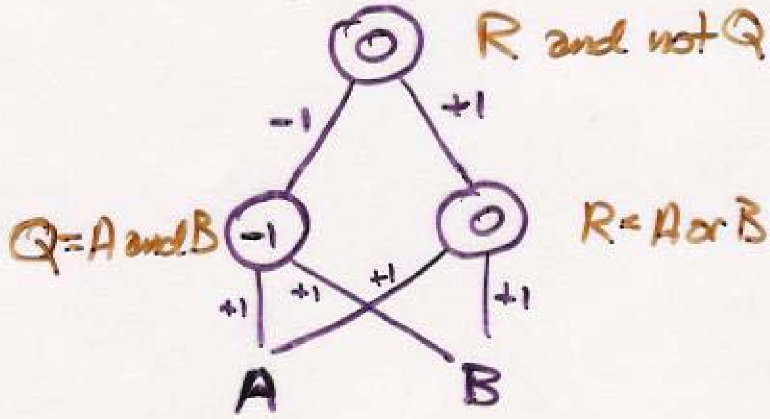
A	B	OUT
0	0	0
0	1	0
1	0	1
1	1	0



$$net = 1 \times A - 1 \times B + 0 = A - B$$

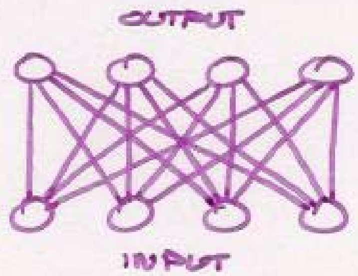
Exclusive or (XOR)

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0



Linear Associators (Anderson, Kohonen)

- Two sets of units: input and output
- Fully interconnected



- Linear activation function:

$$y_j = \sum_i w_{ji} x_i \quad \underline{\text{or}} \quad Y = WX$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_B \end{bmatrix} = \begin{bmatrix} w_{11} & \cdots & w_{1A} \\ \vdots & & \vdots \\ w_{B1} & \cdots & w_{BA} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_A \end{bmatrix}$$

- Supervised learning task

  Learn input-output mapping $X^\alpha \to D^\alpha \quad \alpha = 1 \ldots p$

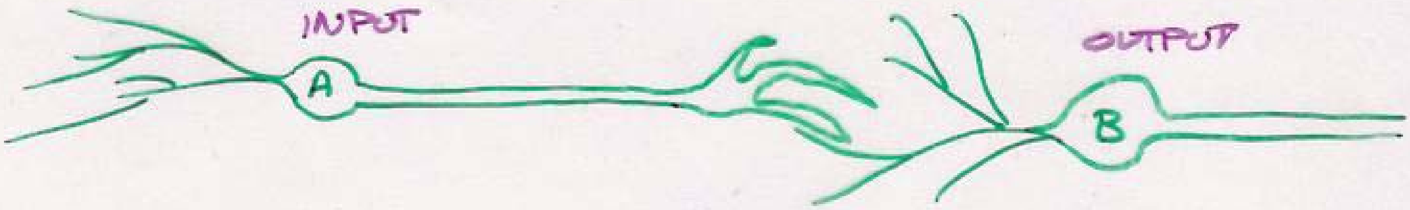
$$\begin{bmatrix} x_1^\alpha \\ x_2^\alpha \\ \vdots \\ x_A^\alpha \end{bmatrix} \quad \begin{bmatrix} d_1^\alpha \\ d_2^\alpha \\ \vdots \\ d_B^\alpha \end{bmatrix}$$

- How do we set weights so that $Y = WX^\alpha = D^\alpha$

Hebbian weight modification rule

"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth or metabolic process takes place in one or both cells such that A's efficiency as one of the cells firing B, is increased."

— D.O. Hebb, 1949



Simplest interpretation of Hebb rule:

1. Set all weights in network to zero initially
2. $\Delta w_{ji} = x_i^\alpha d_j^\alpha$ or $\Delta W = D^\alpha X^{\alpha T}$

$\boxed{\text{OUTER PRODUCT RULE}}$

$$\begin{bmatrix} \Delta w_{11} & \cdots & \Delta w_{1A} \\ \vdots & & \vdots \\ \Delta w_{B1} & \cdots & \Delta w_{BA} \end{bmatrix} = \begin{bmatrix} d_1^\alpha \\ \vdots \\ d_B^\alpha \end{bmatrix} \begin{bmatrix} x_1^\alpha & \cdots & x_A^\alpha \end{bmatrix}$$

After presentation of p patterns, $W = \sum_{\alpha=1}^{p} D^\alpha X^{\alpha T}$

or $w_{ji} = \sum d_j^\alpha x_i^\alpha$

Relation between Hebb rule and correlations:

correlation between x_i and d_j $= \dfrac{\sum_{\alpha=1}^{p} (x_i^\alpha - \bar{x}_i)(d_j^\alpha - \bar{d}_j)}{\sqrt{\sum (x_i^\alpha - \bar{x}_i)^2 \sum (d_j^\alpha - \bar{d}_j)^2}}$

$\boxed{\text{IF } \bar{x}_i \text{ and } \bar{d}_j = 0} \rightarrow$

$= \dfrac{\sum x_i^\alpha d_j^\alpha}{\sqrt{\sum x_i^{\alpha 2} \sum d_j^{\alpha 2}}} \sim w_{ji}$

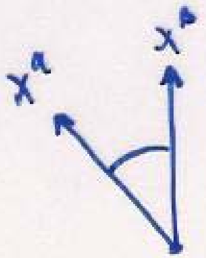
Analyzing Retrieval

Suppose the input patterns are orthonormal

I.e., normalized such that $\|x^\alpha\| = [x^{\alpha T} \cdot x^\alpha]^{1/2} = [\Sigma x_i^{\alpha 2}]^{1/2} = 1$

and x^α is orthogonal to x^β, $\alpha \neq \beta$: $x^{\alpha T} \cdot x^\beta = \Sigma x_i^\alpha x_i^\beta = 0$

For normalized vectors, dot product is a measure of similarity:



$$x^{\alpha T} \cdot x^\beta = \frac{x^{\alpha T} \cdot x^\beta}{\|x^\alpha\| \|x^\beta\|} = \cos(\text{angle between vectors})$$

$\cos(0) = 1$
$\cos(90) = 0$

Given the input to a stored association, x^α, what will system retrieve?

$$Y = W x^\alpha = \left(\sum_{\beta=1}^{P} D^\beta x^{\beta T} \right) x^\alpha$$

$$= D^1 \underline{x^{1T} x^\alpha} + D^2 \underline{x^{2T} x^\alpha} + \ldots + D^P \underline{x^{PT} x^\alpha}$$

But these terms are all zero for $\beta \neq \alpha$

$$= D^\alpha \underline{x^{\alpha T} x^\alpha}$$

$$= D^\alpha$$

If input vector has A elements, at most A associations can be stored in this manner

Interference with nonorthogonal inputs

Suppose two associations are stored, $X^1 - D^1$, $X^2 - D^2$ but X^1 isn't orthogonal to X^2 e.g., $X^{1T}X^2 = .2$

(angle is $\cos^{-1}(.2) = 78.5°$)

What will system retrieve given input X^1?

$$Y = WX^\alpha = (D^1X^{1T} + D^2X^{2T})X^1$$

$$= D^1X^{1T}X^1 + D^2X^{2T}X^1$$

$$= D^1 \quad + .2D^2$$

Interference of association α on association β is related to similarity of X^α and X^β (i.e., angle between them)

Generalization

Suppose two associations are stored, $X^1 - D^1$ and $X^2 - D^2$, and system is probed with another input, X^α.

$$Y = WX^\alpha = (D^1X^{1T} + D^2X^{2T})X^\alpha$$

$$= [X^{1T}X^\alpha]D^1 + [X^{2T}X^\alpha]D^2$$

System produces D^β with magnitude (strength) proportional to the similarity of X^α and X^β.

Suppose two associations are stored, $X^1 - D^1$ and $X^2 - D^2$, and inputs are orthogonal, and system is probed with input $X^\alpha = .5X^1 + .5X^2$

$$Y = WX^\alpha = (D^1X^{1T} + D^2X^{2T})(.5X^1 + .5X^2) = .5(D^1X^{1T}X^1 + \cancel{D^1X^{1T}X^2} + \cancel{D^2X^{2T}X^1} + D^2X^{2T}X^2)$$

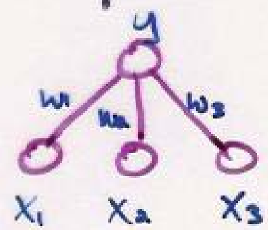
$$= .5D^1 + .5D^2$$

LINEAR INTERPOLATION

LMS weight modification rule

Can we do better than Hebb rule?

What would optimal set of weights be?



$$y = \Sigma\, x_i w_i$$

$$x_1^1 w_1 + x_2^1 w_2 + x_3^1 w_3 = d^1$$
$$x_1^2 w_1 + x_2^2 w_2 + x_3^2 w_3 = d^2$$
$$\vdots$$
$$x_1^P w_1 + x_2^P w_2 + x_3^P w_3 = d^P$$

Find weights that satisfy a system of linear equations

More general case (many output units):

$$W X^1 = D^1$$
$$W X^2 = D^2$$
$$\vdots$$
$$W X^P = D^P$$

Or:

$$W \bar{X} = \bar{D}$$

$$\bar{X} = (X^1\ X^2\ X^3\ \cdots\ X^P) = \begin{bmatrix} x_1^1 & \cdots & x_1^P \\ x_2^1 & & x_2^P \\ \vdots & & \vdots \\ x_A^1 & \cdots & x_A^P \end{bmatrix}$$

$$\bar{D} = (D^1\ D^2\ D^3\ \cdots\ D^P)$$

(Only need be concerned with a single output unit)

What if there is no set of weights that make all equations true?

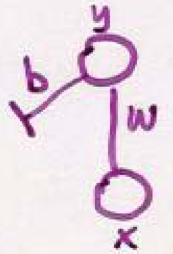
E.g., when there are more associations to be learned (equations) than weights (unknown parameters)

Answer: Find "least mean squares" (LMS) solution — set of weights that minimizes

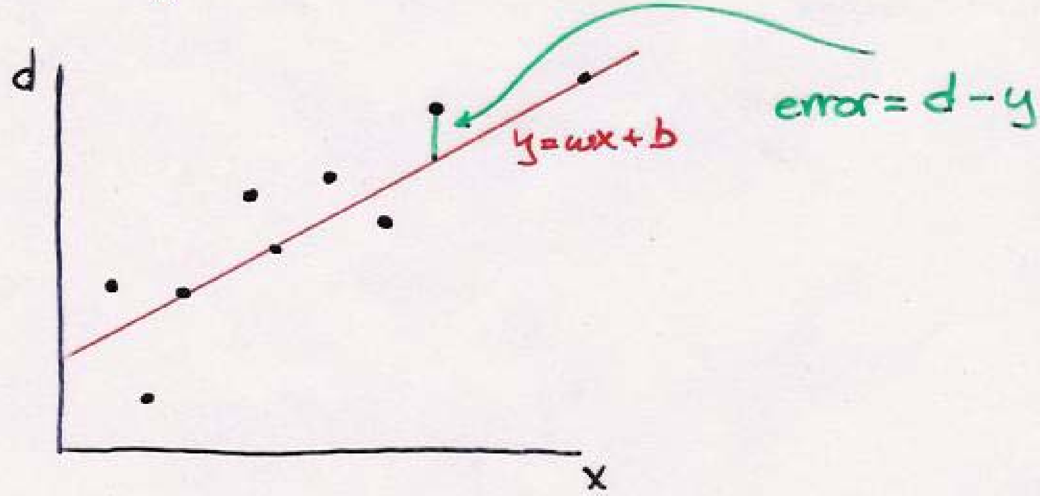
$$E = \text{error} = \sum_{\alpha=1}^{P} \sum_{i=1}^{B} (d_i^\alpha - y_i^\alpha)^2$$

This is just <u>linear regression</u> — finding the set of coefficients that allow you to best predict one variable (d_j) from some other variables ($x_1, \ldots, x_A$)

For example, consider a network with one input and one output.



$$y = wx + b$$



each point corresponds to a training pair (an association)
w = slope of regression line
b = intercept of regression line

If input vectors span the input space (i.e., every possible input vector can be expressed as a weighted sum of the x^α), LMS solution is

$$W = \bar{D}\,\bar{X}^T (\bar{X}\,\bar{X}^T)^{-1}$$

where $\bar{X} = (x^1\ x^2 \cdots x^P)$, $\bar{D} = (D^1\ D^2 \cdots D^P)$

Yuch – a network couldn't possibly compute this messy function (for one thing, it requires all input-output pairs to be available simultaneously)

Another way of looking at situation:

Optimization problem: How should we adjust weights to decrease error?

Strategy: Compute $\dfrac{\partial E}{\partial w_{ji}}$

IF +, increasing $w_{ji} \Rightarrow$ increased E
IF −, decreasing $w_{ji} \Rightarrow$ increased E

gradient = slope of error surface along w_{ji} axis

$$\Delta w_{ji} = -\epsilon \frac{\partial E}{\partial w_{ji}}$$

ϵ : "learning rate" (step size)

Computing $\partial E / \partial w_{ji}$

$$E = \sum_{\alpha=1}^{P} \sum_{k=1}^{B} (d_k^\alpha - y_k^\alpha)^2$$

$$E = \sum_\alpha \sum_k \left(d_k^\alpha - \sum_\ell w_{k\ell}\, x_\ell^\alpha\right)^2$$

$$\frac{\partial E}{\partial w_{ji}} = \sum_\alpha \sum_k 2 \cdot \left(d_k^\alpha - \sum_\ell w_{k\ell}\, x_\ell^\alpha\right) \underbrace{\frac{\partial}{\partial w_{ji}}\left[d_k^\alpha - \sum_\ell w_{k\ell}\, x_\ell^\alpha\right]}$$

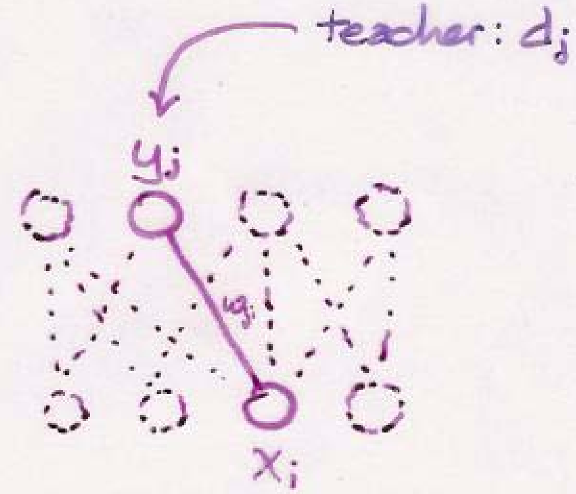
$= 0$ if $j \neq k$

$= -x_i^\alpha$ otherwise

$$\frac{\partial E}{\partial w_{ji}} = \sum_\alpha 2 \cdot \left(d_j^\alpha - \sum w_{j\ell}\, x_\ell^\alpha\right) \cdot -x_i^\alpha$$

$$= -2 \sum_\alpha (d_j^\alpha - y_j^\alpha)\, x_i^\alpha$$

$\Rightarrow$



$$\Delta w_{ji} = -\epsilon \frac{\partial E}{\partial w_{ji}}$$

$$= \epsilon \sum_\alpha (d_j^\alpha - y_j^\alpha)\, x_i^\alpha$$

$$\boxed{\begin{array}{l}\text{For each pattern, } \alpha, \\ \Delta w_{ji} = \epsilon (d_j^\alpha - y_j^\alpha)\, x_i^\alpha\end{array}}$$

ONLY LOCAL INFO IS REQUIRED!

Δb_j is like Δw_{ji}, assuming input is always 1:

$$\Delta b_j = \epsilon (d_j^\alpha - y_j^\alpha)$$

Shape of error surface

Suppose we have 1 input, 1 output unit (no bias)

$$E = \sum_{\alpha=1}^{P} \sum_{k=1}^{3} (d_k^\alpha - y_k^\alpha)^2$$

$$= \sum_\alpha (d^\alpha - w x^\alpha)^2$$

$$= \sum (d^{\alpha 2} - 2 x^\alpha d^\alpha w + x^{\alpha 2} w^2)$$

$$= \left[\sum x^{\alpha 2}\right] w^2 - \left[\sum 2 x^\alpha d^\alpha\right] w + \left[\sum d^{\alpha 2}\right]$$

∴ Error surface is quadratic in w (i.e., highest exponent of w is 2)

This is true no matter how many weights are in network

Only one minimum in quadratic surfaces ("hyperparabolic")

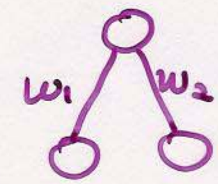
slope of error surface = dE/dw

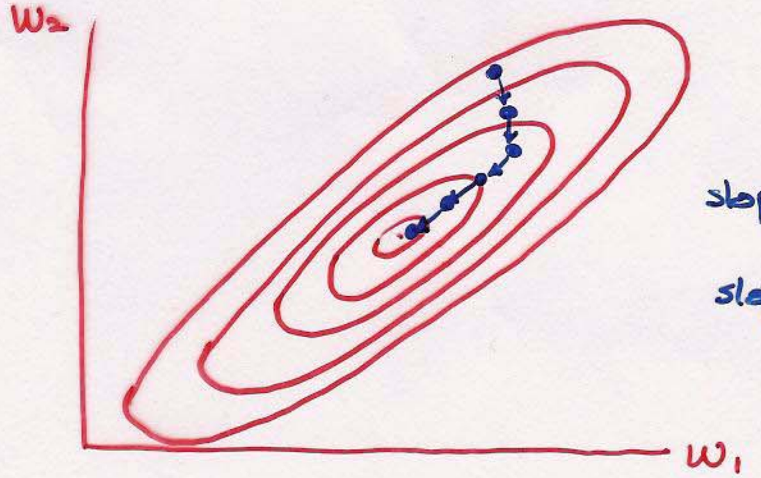
Weight update procedure:

(1) start off at some random point in weight space (•)

(2) modify weights so as to move downhill in error

Suppose we have 2 inputs, 1 output unit





slope of error surface along w_1 axis $= \partial E/\partial w_1$

slope of error surface along w_2 axis $= \partial E/\partial w_2$

Error surface is a bowl.
Slices through $w_1 - w_2$ plane are ellipses
Slices through $E - w_1$ or $E - w_2$ plane are parabolas

What determines shape of bowl?

$$E = \sum_\alpha \left(d^\alpha - [w_1 x_1^\alpha + w_2 x_2^\alpha]\right)^2 \qquad \text{(let's drop } \alpha\text{'s !)}$$

$$= \sum_\alpha (d - w_1x_1 - w_2x_2)(d - w_1x_1 - w_2x_2)$$

$$= \sum_\alpha (d^2 - 2dx_1w_1 - 2dx_2w_2 + x_1^2w_1^2 + x_2^2w_2^2 + 2x_1x_2w_1w_2)$$

$$= \cancel{\Sigma d^2} - 2\cancel{(\Sigma dx_1)}w_1 - 2\cancel{(\Sigma dx_2)}w_2 + (\Sigma x_1^2)w_1^2 + (\Sigma x_2^2)w_2^2 + 2(\Sigma x_1x_2)w_1w_2$$

not important   curvature   orientation

Curvature (elongation) along w_1-axis determined by $\sum_\alpha x_1^2$
(or, average value of x_1^2 across patterns)

Curvature (elongation) along w_2-axis determined by $\sum_\alpha x_2^2$
(or, average value of x_2^2 across patterns)

Orientation (rotation) of bowl determined by Σx_1x_2 (or, second-order moment $E[x_1x_2]$)

$\Sigma x_1 x_2$ small $\Sigma x_1 x_2$ large

Batch vs. On-Line Training

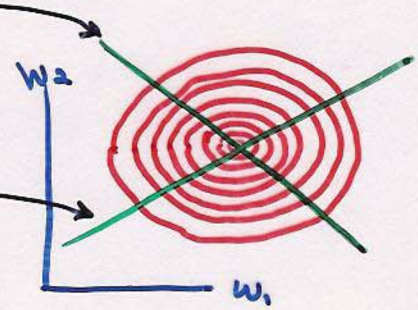
Suppose we have 2 inputs, one output, and two associations to learn, $x^1 - d^1$, $x^2 - d^2$.

Weights suitable for first association satisfy the constraint

$$x_1^1 w_1 + x_2^1 w_2 = d^1$$

And for the second association,

$$x_1^2 w_1 + x_2^2 w_2 = d^2$$



Two ways of updating weights:

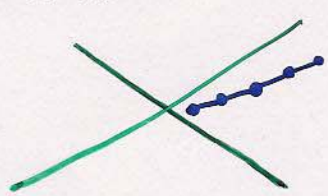
(1) sweep through all patterns and then modify weights

i.e., $\Delta w_{ji} = \sum_{\alpha=1}^{p} \epsilon (d_j^\alpha - y_j^\alpha) x_i^\alpha$

(2) modify weights after each pattern, α, has been presented

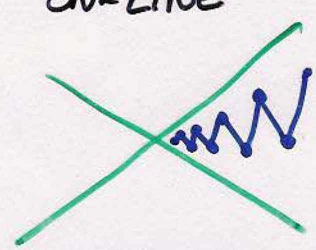
i.e., $\Delta w_{ji} = \epsilon (d_j^\alpha - y_j^\alpha) x_i^\alpha$

(1) "BATCH"



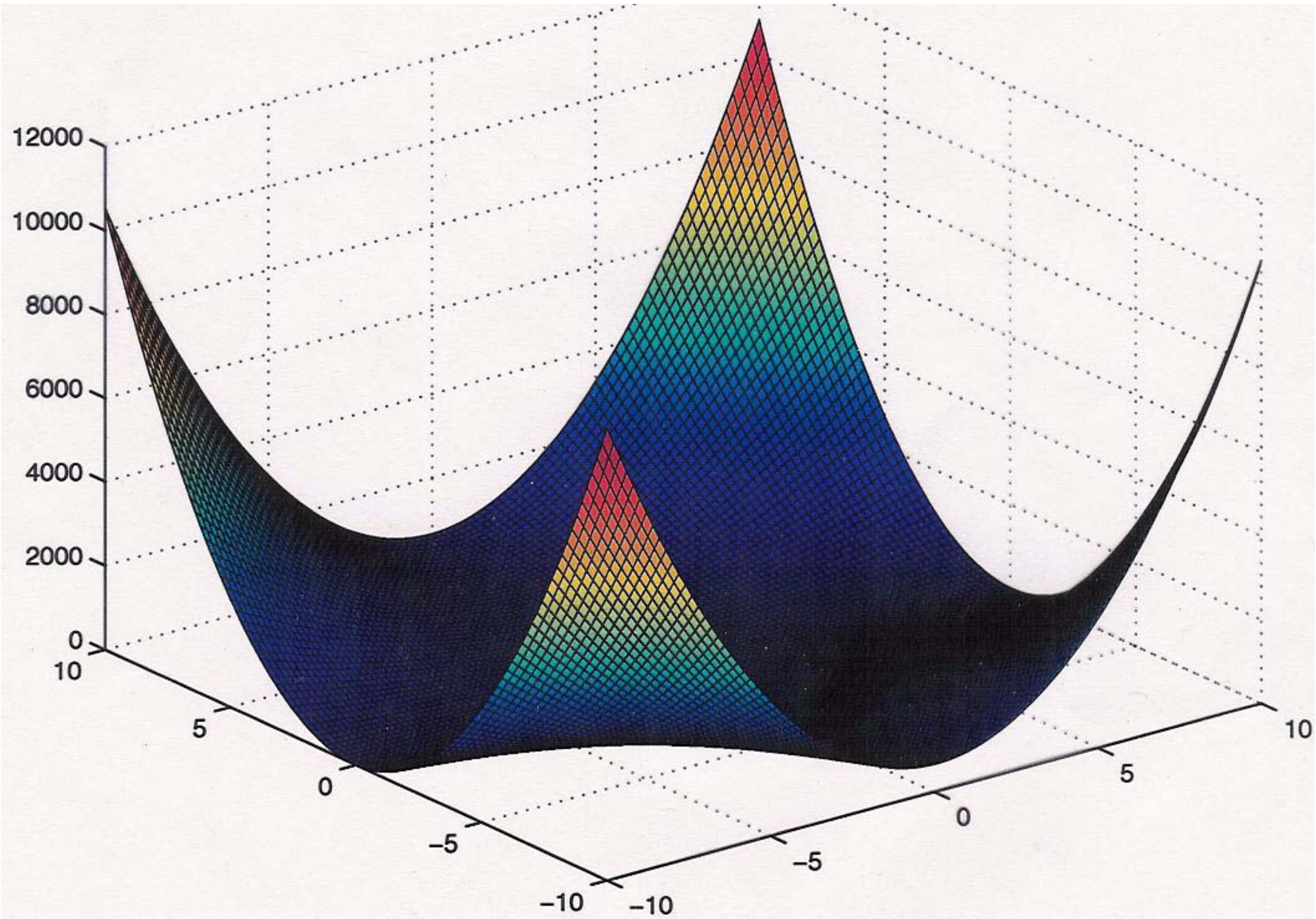
true gradient descent
i.e., $\Delta w \sim \partial E / \partial w$

(2) "ON-LINE"



noisy or "stochastic" descent

On-line can be faster if associations are similar!


12000
10000
8000
6000
4000
2000
0
10
5
0
−5
−10
−10
−5
0
5
10

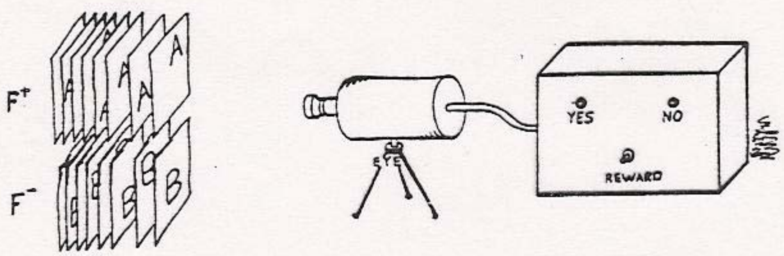
Perceptrons

History

1962, Rosenblatt, "Principles of Neurodynamics"

— defined perceptrons

— amazing theorem: Perceptron can learn to do anything it is possible to program it to do.

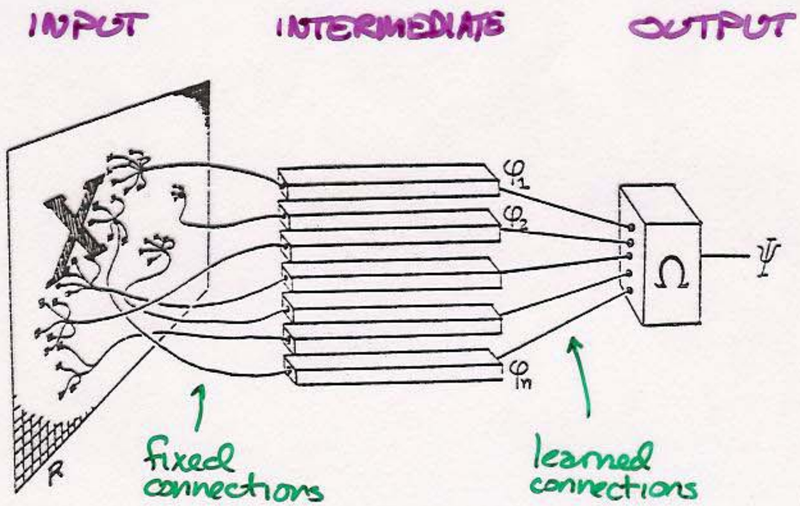


1969, Minsky and Papert, "Perceptrons: An introduction to Computational Geometry"

— Computational complexity analysis
How does learning time scale with difficulty of problem?
How much information does each weight need to represent?
Are there classes of functions that cannot be computed by perceptrons of a certain architecture?

— Limits on perceptron-like architectures

Perceptron architecture



Intermediate units compute some binary function, φ_i, of the inp

Output unit computes some binary function, Ψ, of the intermed units

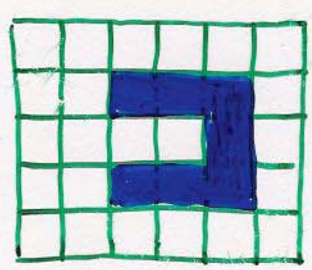
$$\Psi = \begin{cases} 1 & \text{if } \sum w_i \varphi_i > \theta \\ 0 & \text{otherwise} \end{cases}$$

"IS WEIGHTED SUM OF INPUTS ABOVE THRESHOLD?"

Binary threshold unit:
$$y = \begin{cases} 1 & \text{if } \sum w_i x_i + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Definition: Perceptron is a device capable of computing all predicates linear in some given set $\{\varphi_1, \varphi_2, \ldots \varphi_n\}$ of partial predicates

E.g., convexity

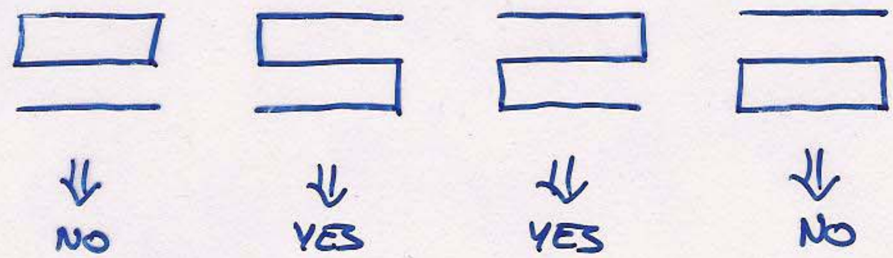


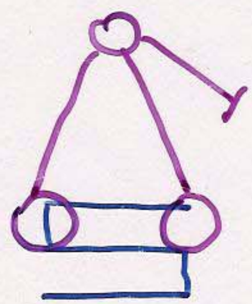
"Order" of perceptron: maximum number of input points that each intermediate unit must examine

Order of Ψ_{CONVEX} is 3

Where Perceptrons Fail

- Order- and diameter-limited perceptrons cannot compute $\psi_{\text{CONNECTEDNESS}}$





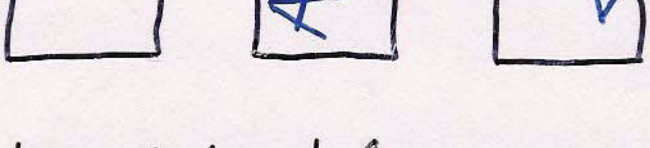
Same as XOR problem:

x_1	x_2	d
0	0	0
0	1	1
1	0	1
1	1	0

$0 \cdot w_1 + 0 \cdot w_2 + b <$
$0 \cdot w_1 + 1 \cdot w_2 + b$
$1 \cdot w_1 + 0 \cdot w_2 + b$
$1 \cdot w_1 + 1 \cdot w_2 + b$

- In general, Perceptrons can solve low-order problems (i.e., function depends only on combinations of a few inpu features), but higher-order problems result in a combinatoric explosion (need 2^A intermediate units)

- A high order problem: translation-invariant object recognitio



Requires an intermediate unit for every possible shape in every possible positio

Limitations of One-Layer Learning

With linear units and Hebb rule, input patterns must be orthogonal $\Rightarrow$ at most A (# elements in input vector) arbitrary associations can be learned

With linear units and LMS rule, input patterns must be linearly independent $\Rightarrow$ at most A arbitrary associatio[n]
can be learned

For both LMS and Hebb, more than A associations can [be] learned if one association is a linear combination of the others.

E.g.,

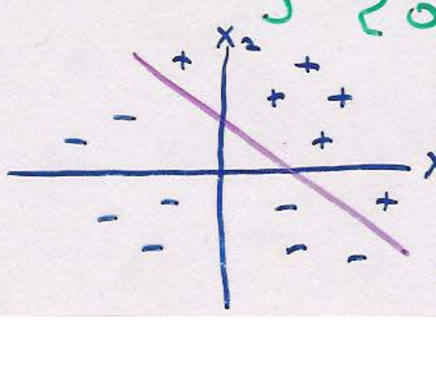
	x_1	x_2	d
assoc 1	.4	.6	1
assoc 2	.6	.4	-1
assoc 3	1	1	0

note: $\underline{x}^1 + \underline{x}^2 = \underline{x}^3$
$d^1 + d^2 = d^3$

With binary-threshold units and binary-valued desired outputs, LMS rule can still be applied (identical to the Perceptron learning rule in this case), but mappings must be "linearly separable".

E.g., network w/ 2 inputs, 1 output

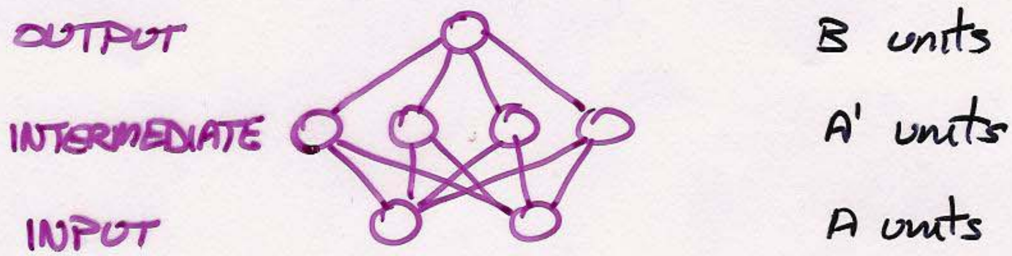
$$y = \begin{cases} 1 & \text{if } w_1x_1 + w_2x_2 + b > 0 \\ 0 & \text{otherwise} \end{cases}$$



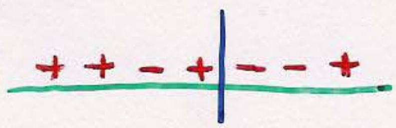
$+ : d = 1$
$- : d = 0$

$\leftarrow$ net $= w_1x_1 + w_2x_2 + b = 0$

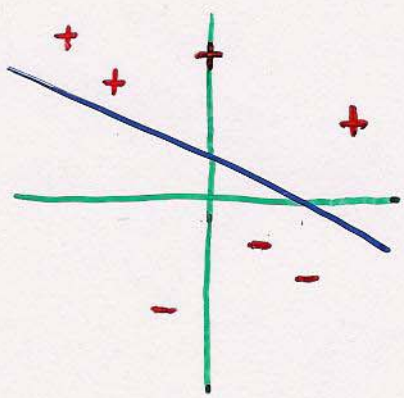
Adding a layer of "hidden" or "intermediate units"



Projecting input vector from an A-dimensional space to an A'-dimensional space

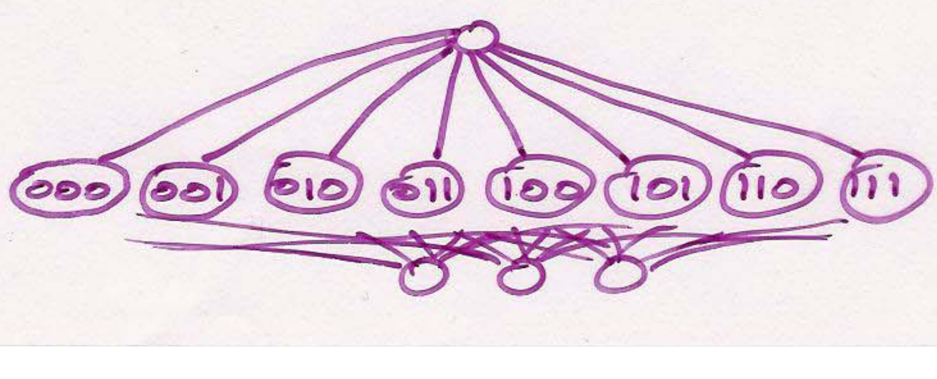
E.g., 

⇒ classification error



⇒ no error

With 2^A intermediate units, any binary mapping can be learned



Exponential number of hidden units is bad

— large network
— poor generalization

With domain knowledge we could pick an appropriate hidden representation.

e.g., perceptron scheme

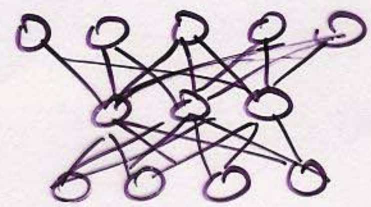
Alternative

Learn hidden representation

Problem

Where does training signal come from?
Teacher specifies target outputs, not target hiddens.

Extending LMS to handle squashing nonlinearities and hidden units



First, let's rederive LMS rule.

$$\Delta w_{ji} \sim -\frac{\partial E}{\partial w_{ji}}$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial o_j}\,\frac{\partial o_j}{\partial w_{ji}}$$

$$E = \sum_k (d_k - o_k)^2$$

$$\frac{\partial E}{\partial o_j} = -2(d_j - o_j)$$

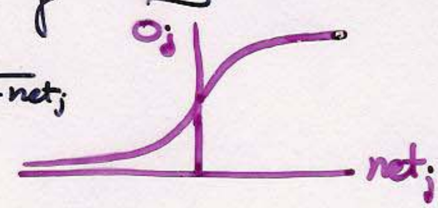
$$\sim -(d_j - o_j)\,o_i$$

$$o_j = \sum_k w_{jk} o_k$$

$$\frac{\partial o_j}{\partial w_{ji}} = o_i$$

Suppose output unit has sigmoid squashing function:

$$net_j = \sum_k w_{jk} o_k \qquad o_j = \frac{1}{1+e^{-net_j}}$$

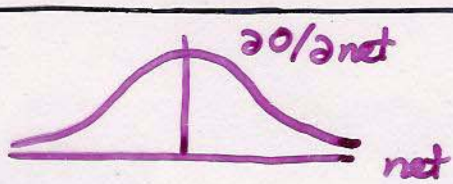


$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial o_j}\,\frac{\partial o_j}{\partial net_j}\,\frac{\partial net_j}{\partial w_{ji}}$$

$$net_j = \sum_k w_{jk} o_k$$

$$\frac{\partial net_j}{\partial w_{ji}} = o_i$$

$$\sim -(d_j - o_j)\;o_j(1-o_j)\;o_i$$

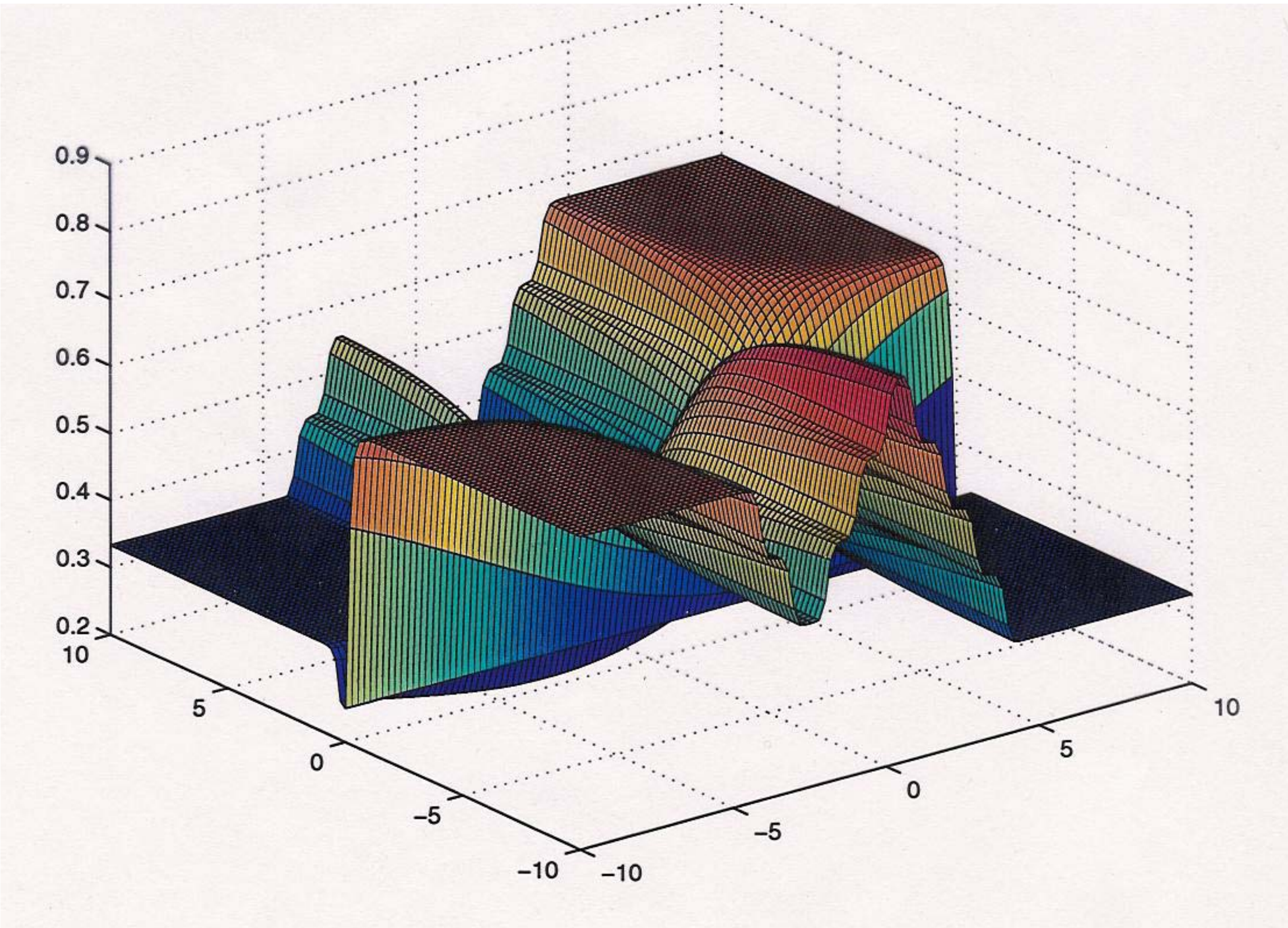


$$o_j = (1+e^{-net_j})^{-1}$$

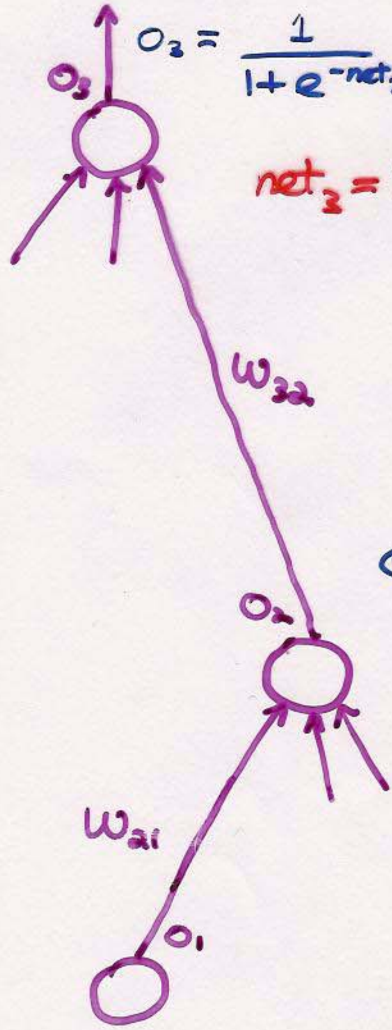
$$\frac{\partial o_j}{\partial net_j} = -1 \cdot (1+e^{-net_j})^{-2} \cdot -e^{-net_j}$$

$$= \frac{1}{1+e^{-net}} \cdot \frac{e^{-net}}{1+e^{-net}}$$

$$= \frac{1}{1+e^{-net}} \cdot \left(1 - \frac{1}{1+e^{-net}}\right) = o_j(1-o_j)$$


0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
10
5
0
−5
−10
−10
−5
0
5
10

Back Propagation (Rumelhart, Hinton, & Williams; leCun; Parker; Werbos)



$$e = (d_3 - o_3)^2 \longrightarrow \frac{\partial e}{\partial o_3} = -2(d_3 - o_3)$$

$$o_3 = \frac{1}{1+e^{-net_3}} \longrightarrow \frac{\partial o_3}{\partial net_3} = o_3(1-o_3)$$

$$net_3 = \sum_i w_{3i} o_i \longrightarrow \frac{\partial net_3}{\partial w_{32}} = o_2$$

$$\longrightarrow \frac{\partial net_3}{\partial o_2} = w_{32}$$

$$o_2 = \frac{1}{1+e^{-net_2}} \longrightarrow \frac{\partial o_2}{\partial net_2} = o_2(1-o_2)$$

$$net_2 = \sum_i w_{2i} o_i \longrightarrow \frac{\partial net_2}{\partial w_{21}} = o_1$$

$$\frac{\partial e}{\partial w_{32}} = \frac{\partial e}{\partial o_3} \frac{\partial o_3}{\partial net_3} \frac{\partial net_3}{\partial w_{32}} \sim -(d_3 - o_3)\, o_3 (1-o_3)\, o_2$$

$$\frac{\partial e}{\partial w_{21}} = \frac{\partial e}{\partial o_2} \frac{\partial o_2}{\partial net_2} \frac{\partial net_2}{\partial w_{21}} \sim (d_3 - o_3)\, o_3 (1-o_3)\, w_{32}\, o_2 (1-o_2)\, o_1$$

$$\uparrow$$

$$\frac{\partial e}{\partial o_2} = \frac{\partial e}{\partial o_3} \frac{\partial o_3}{\partial net_3} \frac{\partial net_3}{\partial o_2} = -(d_3 - o_3)\, o_3 (1-o_3)\, w_{32}$$

What it boils down to

$$\Delta w_{ji} = \epsilon \, \delta_j \, o_i$$

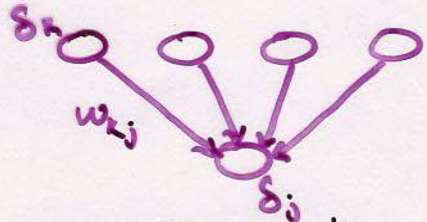
For output unit,

$$\delta_j = (d_j - o_j) \cdot o_j(1-o_j)$$

this term is simply $\frac{\partial o_j}{\partial net_j}$

For hidden unit,

$$\delta_j = \left[\sum_k \delta_k w_{kj}\right] \cdot o_j(1-o_j)$$

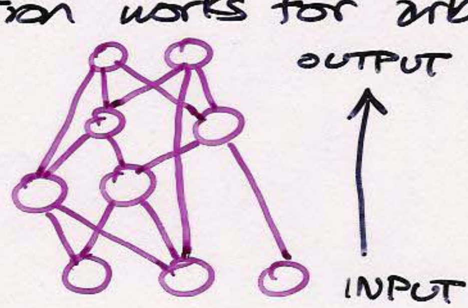


- Δw_{ji} for output* unit is the same as LMS with a nonlinear output (LMS ≡ "delta rule"; back prop ≡ "generalized delta rule")

- Two phase process: Forward (activation propagation) phase
  Backward (error propagation) phase

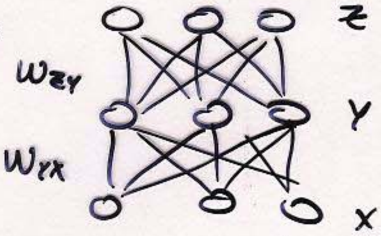
- As with LMS rule, back prop performs gradient descent in error space, i.e., finding best set of weights that minimize error. weights = <u>all</u> weights (biases) in network

- Back propagation works for arbitrarily deep <u>feedforward</u> networks



— Nonlinearities (sigmoid squashing functions) are important. Any multilayer linear network can be reduced to a single layer network

E.g.

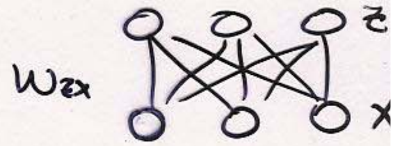


$$Z = W_{ZY} Y$$

$$Y = W_{YX} X$$

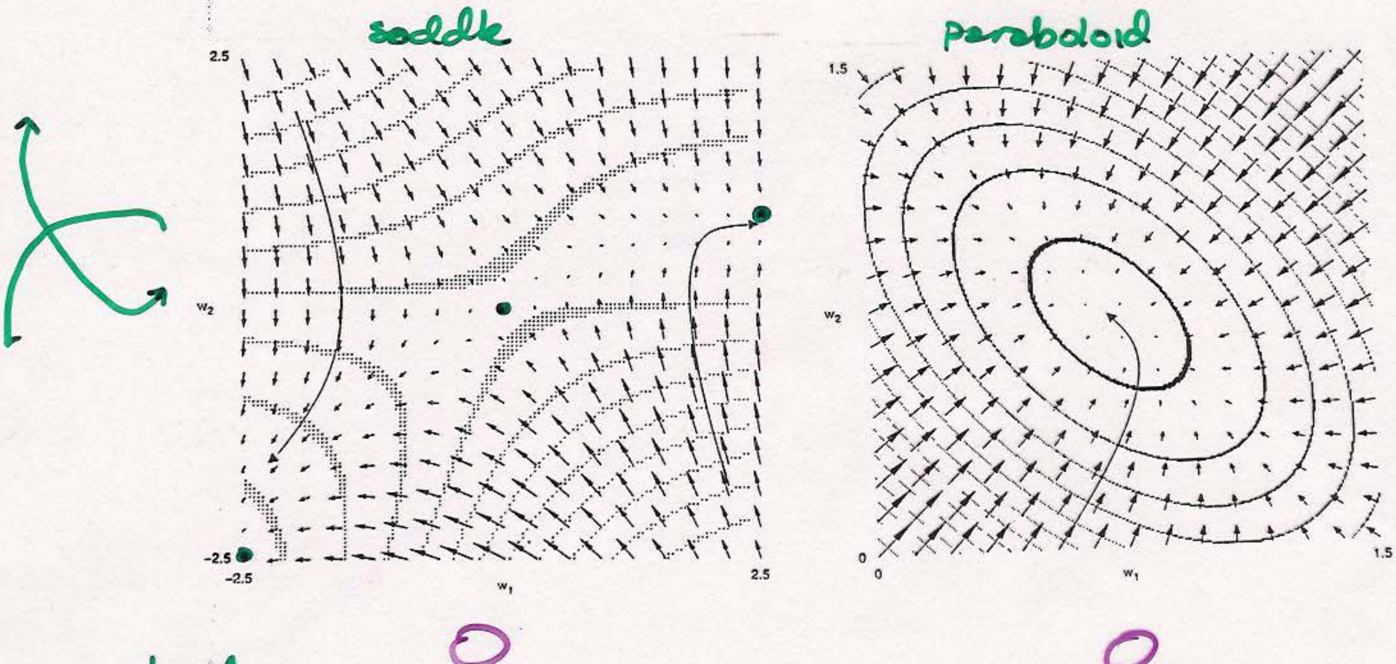
$$\Rightarrow \quad Z = \underbrace{W_{ZY} W_{YX}} X$$

$$Z = W_{ZX} X$$

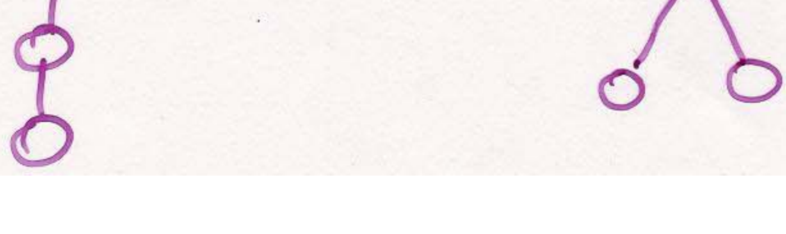


∴ A multilayer linear network is no more powerful than a single layered linear network (can't compute XOR, etc.)

— Error surface is no longer hyperparabolic (i.e., no longer one global minimum)

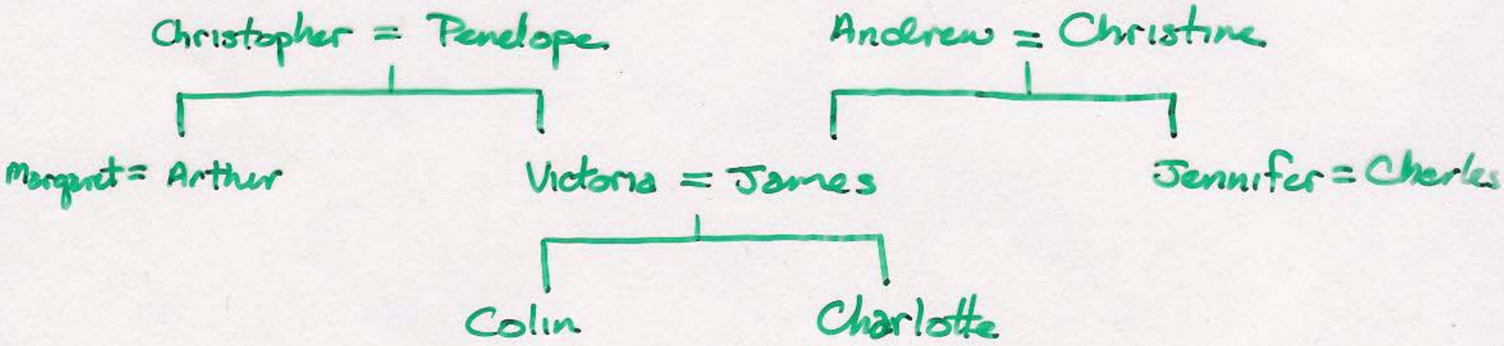


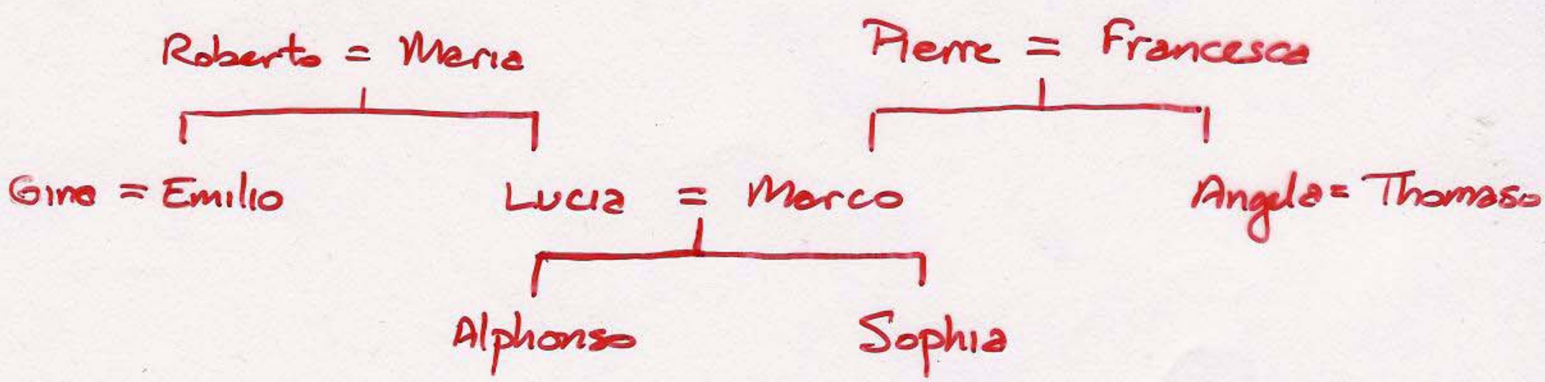
• = local minimum



Family Trees Task (HINTON, Proc. 8th Annual Conf. Cog. Sci Society, Amherst, MA, 1986, Erlbaum Assoc.)

Can we get a network to learn the information in these two family trees?





Knowledge can be represented as triples:

(has-mother Alphonso Lucia)
(has-wife James Victoria)
(has-son Penelope Arthur)

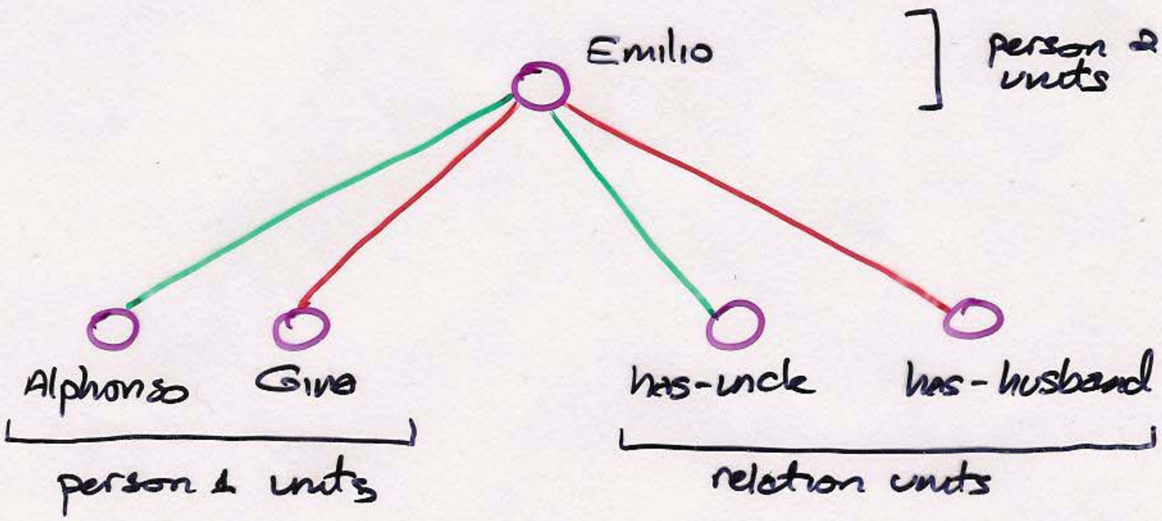
12 relationships: brother, sister, nephew, niece, uncle, aunt, husband, wife, son, daughter, father, mother

Can system learn to produce the third term of a triple when given the first two?

E.g., (has-aunt Alphonso ?) Angela!

A two layer net won't do:

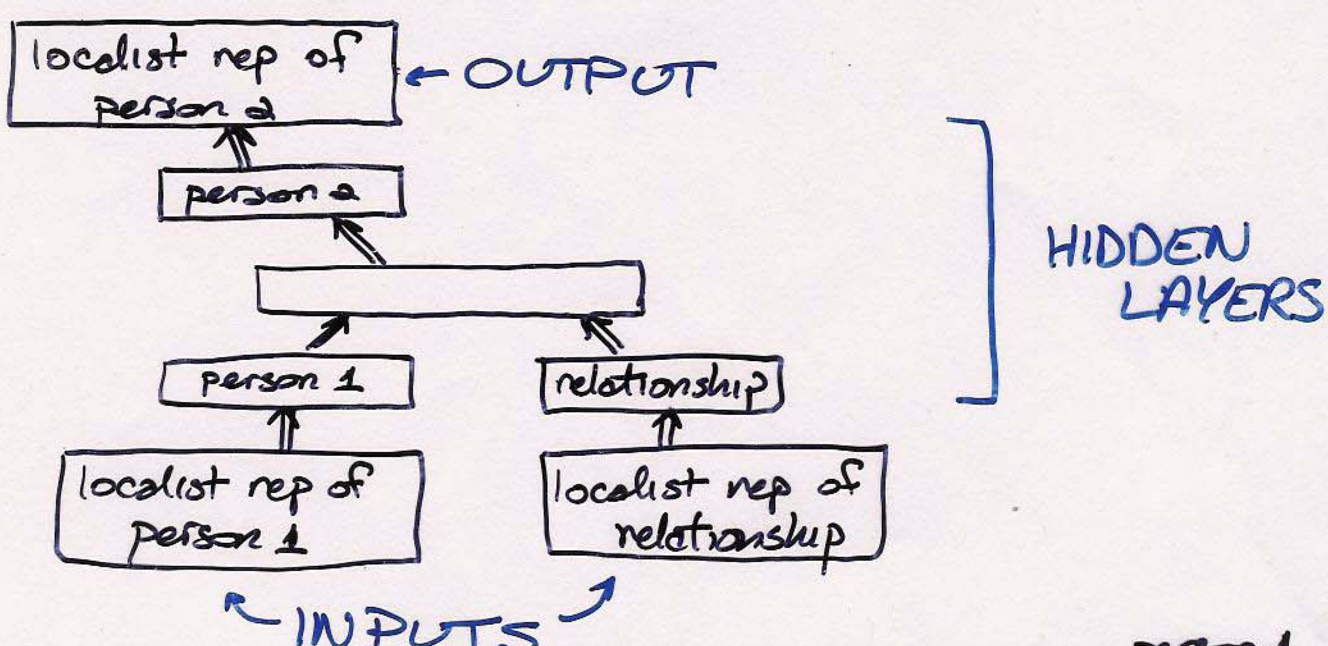
E.g., (has-uncle Alphonso Emilio)
(has-husband Gina Emilio)



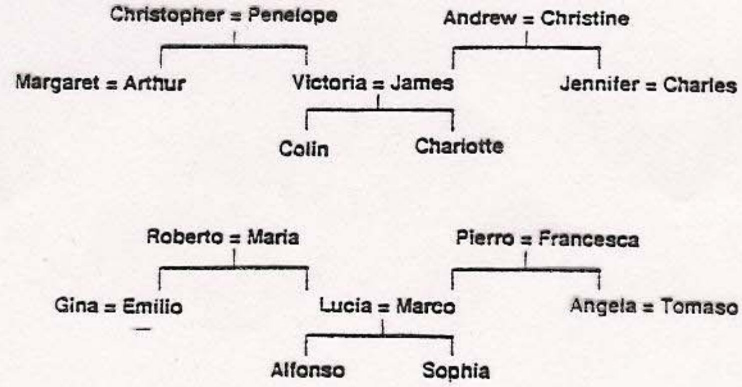
Problem:

(has-uncle Gina ?)
(has-husband Alphonso ?)

Hinton's architecture



System must learn an "internal representation" of person 1, person 2, relationship



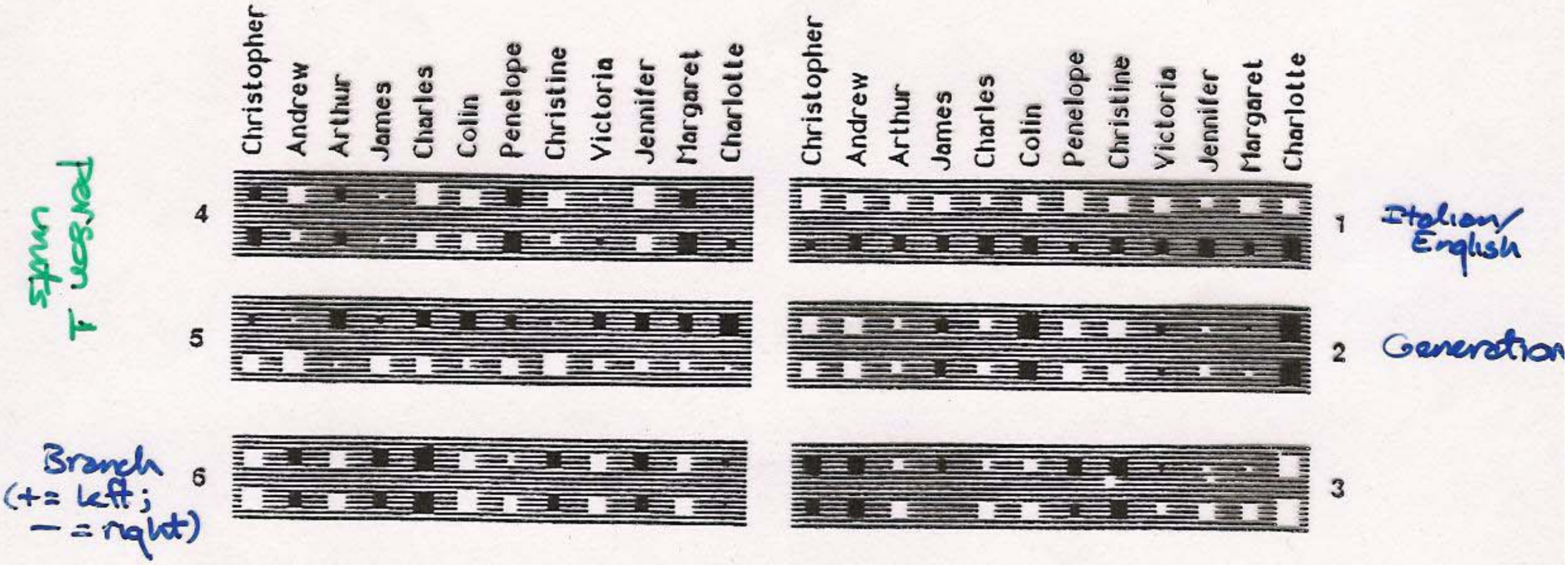


Figure 6: The weights from the 24 input units that represent people to the 6 units in the second layer that learn distributed representations of people. White rectangles stand for excitatory weights, black for inhibitory weights, and the area of the rectangle encodes the magnitude of the weight. The weights from the 12 English people are in the top row of each unit. Beneath each of these weights is the weight from the isomorphic Italian.

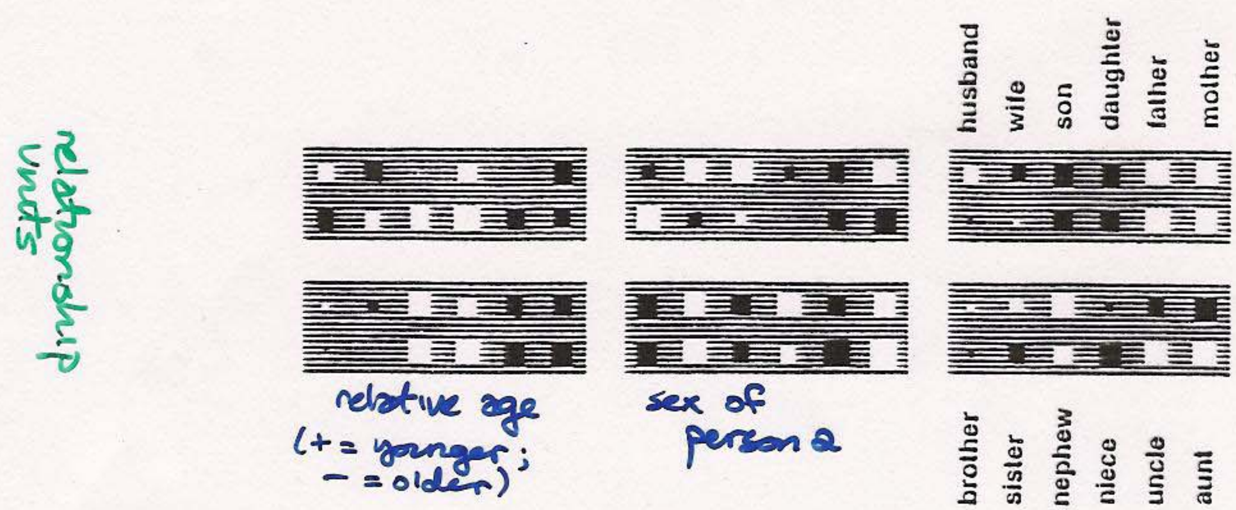


Figure 7: The weights from the 12 input units that represent relationships to the 6 units in the second layer that learn distributed representations of the relationships.