# Introduction to Parallel Distributed Processing models

Deon T. Benton

003-Hebb and Delta rules

# A note about activation functions

- We briefly discussed a variety (threshold, linear, sigmoid, tanh, etc) but zoomed in on sigmoid
- Often to simplify the math it is useful to think about threshold functions where activations are -1 if input is negative and +1 if input is positive
- Can view this as indicating whether a unit is *more* or *less* active than its tonic firing rate.
- Other times it is mathematically useful to link about *linear* activation functions (activation = net input).
- These have less clear neural interpretations but are a useful stepping-stone for understanding learning algorithms.

# A long-standing view of learning/memory

- **William James, 1890:** Memory arises from association and generalization
  - *Association:* Two things that regularly occur together (“contiguity”) become associated in memory; encountering one brings the other to mind.
  - e.g.: image of German Shepard + word “dog”
  - Generalization: Same association will be promoted by new stimuli when they are “similar” to one of the associates
  - e.g.: image of a Labrador evokes word “dog” b/c Labrador is similar to German Shepard
- How might such learning occur in a neural network?

# Associative learning

- A given item (word, image, sound, motor action, feeling,etc) is represented as a pattern of activity over units
- Two items (e.g. word, picture) represented by two patterns over different units, connected via weights
- When they are “contiguous,” both patterns simultaneously active
- Weights must change so that one pattern presented alone generates the other pattern

# A mechanism: Hebbian learning

- Hebb:
  - When an axon of cell A is near enough to excite a cell B and *repeatedly and consistently takes part in firing it*, some growth process or metabolic change takes place in one or both cells such that A's efficacy, as one of the cells firing B, is increased.
- Minimal Hebb rule:
  - When there is a synapse between cell A and cell B, increment the strength of the synapse whenever A and B fire together (or in close succession).
- In math:
  - $\Delta w_{ij} = \varepsilon a_i a_j$ (outer product)
  - ...where $w_{ij}$ is is the weight from i to j, $\varepsilon$ is a constant and $a_i$ and $a_j$ are the activations of the connected units

# A note on *superpositional weight changes*

- Changes to the weight matrix are superpositional
  - changes made to a weight matrix, *w*, by the outer product of the activity between an input and output pattern pair at time *t* will be superimposed on the weight changes caused by another pair at time *t* +1.

Thus,

$$W = \begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}$$

$$(u_1 y_1) \rightarrow \begin{matrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{matrix}$$

$$(u_2 y_2) \rightarrow \begin{matrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{matrix}$$

$$\begin{matrix} 2 & 2 & 0 \\ 1 & 2 & 1 \\ 2 & 1 & 0 \end{matrix}$$

# A quick aside: The inner (or dot) product *between two vectors*

- If we have two vectors, $u$ and $v$, where

$$u = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} \text{ and } v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$$

Then their *inner* product is

$$u^T v = (u_1 u_2\, u_3) \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} = u_1 v_1 + u_2 v_2 + u_3 v_3$$

# A quick aside: The outer product *between two vectors*

- If we have two vectors, *u* and *v*, where

$u = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix}$ and $v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}$

Then their *outer* product is

$$uv^T = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} (v_1 v_2\ v_3) = \begin{matrix} u_1 v_1 & u_1 v_2 & u_1 v_3 \\ u_2 v_1 & u_2 v_1 & u_2 v_1 \\ u_3 v_1 & u_3 v_1 & u_3 v_1 \end{matrix}$$

If $w_{ij} = 0$ initially, after a set of $n$ training trials on patterns (indexed by $p$) where $\triangle w_{ij} = \epsilon \ a_i a_j$,

$$w_{ij} = \epsilon \sum_{p=1}^{n} a_i^{[p]} \ a_j^{[p]}$$

notation: $a_i^{[p]}$ is unit $i$'s activation in pattern $p$

Suppose $a_i$ and $a_j$ take on values of $+1$ or $-1$

- If $a_i$ and $a_j$ are *perfectly correlated* (always the same), $a_i^{[p]} \ a_j^{[p]} = 1$, so
$$w_{ij} = \epsilon \ n$$
- If $a_i$ and $a_j$ are *perfectly anticorrelated* (always differ), $a_i^{[p]} \ a_j^{[p]} = -1$, so
$$w_{ij} = -\epsilon \ n$$
- If $a_i$ and $a_j$ are *uncorrelated* (differ as often as same)
$$w_{ij} = \epsilon \ \left(\frac{n}{2}(+1) + \frac{n}{2}(-1)\right) = 0$$
- If $a_i$ and $a_j$ are *partially correlated* (e.g., 3/4 same and 1/4 different)
$$w_{ij} = \epsilon \ n \ \left(\frac{3}{4}(+1) + \frac{1}{4}(-1)\right) = \frac{1}{2}\epsilon \ n$$
- Thus $w_{ij} \propto \mathrm{correlation}(a_i, a_j)$

# Statistical correlation

- $r_{xy} = \frac{\sum_d (x_d - \bar{x})(y_d - \bar{y})}{\sqrt{(\sum_d (x_d - \bar{x})^2)(\sum_d (y_d - \bar{y})^2)}}$
- Denominator just normalizes to [-1,1] range, so:
- $r_{xy} \propto \sum_d (x_d \ - \ \bar{x})(y_d - \bar{y})$
- In fact if activations are in [-1,1] and each unit has a mean activation of 0 across all patterns, then:
  - $a_i = (x_d - \bar{x}) \rightarrow a_i = (x_d - 0) = a_i = x_d$
  - $a_j = (y_d - \bar{y}) \rightarrow a_j = (y_d \ - \ 0) = a_j = y_d$
  - $w_{ij} \propto r_{xy} \rightarrow a_i a_j \propto x_d \, y_d$

If test pattern p′ is orthogonal to all training patterns p, $\mathrm{dp}(\mathsf{p}', \mathsf{p}) = 0$ for all p, so

$$a_j^{[\mathsf{p}']} \;=\; \epsilon \sum_{\mathsf{p}} a_j^{[\mathsf{p}]}\,\mathrm{dp}(\mathsf{p}', \mathsf{p}) \;=\; \epsilon \sum_{\mathsf{p}} a_j^{[\mathsf{p}]}\,0 \;=\; 0$$

If all training patterns are orthogonal to each other (and assuming $\epsilon = 1$), then

- If p′ is one of the training patterns (say $\mathsf{p}^*$), recall is perfect:

$$a_j^{[\mathsf{p}']} \;=\; a_j^{[\mathsf{p}^*]}\,\mathrm{dp}(\mathsf{p}^*, \mathsf{p}^*) + \sum_{p \neq p^*} a_j^{[\mathsf{p}]}\,\mathrm{dp}(\mathsf{p}', \mathsf{p}) \;=\; a_j^{[\mathsf{p}^*]} + \sum_{p \neq p^*} a_j^{[\mathsf{p}]} 0 \;=\; a_j^{[\mathsf{p}^*]}$$

- If p′ is *similar* to only one training pattern ($\mathsf{p}^*$) and orthogonal to the rest, the output is $a_j^{[\mathsf{p}^*]}$ scaled by the degree of similarity:

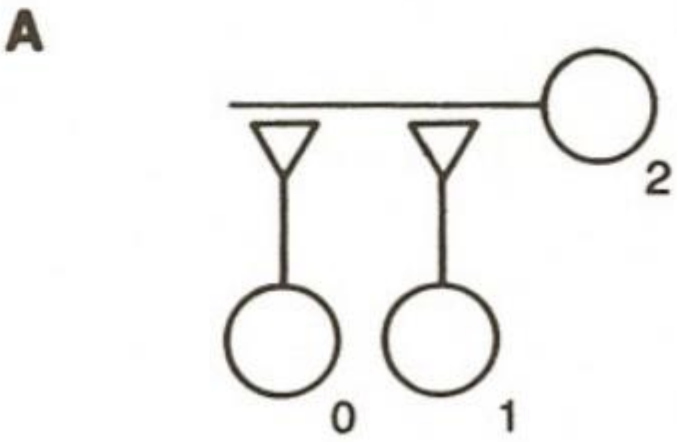
$$a_j^{[\mathsf{p}']} \;=\; a_j^{[\mathsf{p}^*]}\,\mathrm{dp}(\mathsf{p}', \mathsf{p}^*) + \sum_{p \neq p^*} a_j^{[\mathsf{p}]}\,\mathrm{dp}(\mathsf{p}', \mathsf{p}) \;=\; a_j^{[\mathsf{p}^*]}\,\mathrm{dp}(\mathsf{p}', \mathsf{p}^*)$$

In general, the output to any input pattern is a weighted combination of the outputs of all trained patterns, scaled by their similarity to the input.

- If the combination agrees with $a_j^{[\mathsf{p}']}$, this is **facilitation** (or generalization if p′ is novel)
- If the combination disagrees with $a_j^{[\mathsf{p}']}$, this is **interference** (or poor generalization)

# Houston, we have a problem!

- Unit-wise correlations are often insufficient to produce the correct output response




**B**

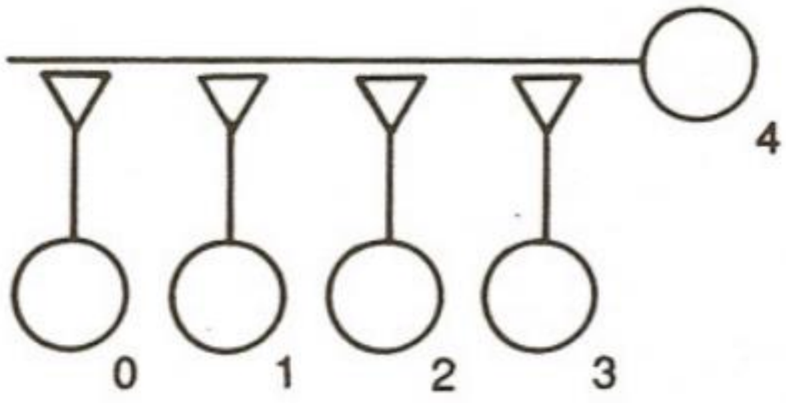
| Input | | Output |
|---|---|---|
| 0 | 1 | 2 |
| + | + | + |
| + | – | + |
| – | + | – |
| – | – | – |

Final weights: +4, 0

# Houston, we have a problem!

- Unit-wise correlations are often insufficient to produce the correct output response

C



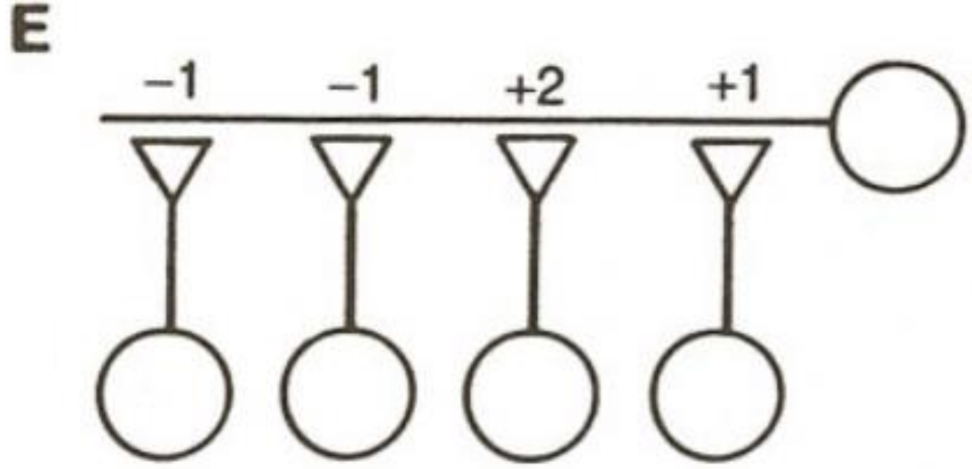

D

| Input | | | | Output |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |
| + | − | + | − | + |
| + | + | + | + | + |
| + | + | + | − | − |
| + | − | − | + | − |

Final weights: +4, 0, +3, 0

# Houston, we have a problem!

- Unit-wise correlations are often insufficient to produce the correct output response



# Error-correcting learning: Delta rule

Change weights so as to reduce difference between actual output ($a_j$) and **target** output (denoted $t_j$)

$$\triangle w_{ij} = \epsilon\,(t_j - a_j)\,a_i$$

- "Delta": difference between output and target
  - Also called Widrow-Hoff rule, LMS (least mean squared)
  - Related to perceptron convergence procedure (Rosenblatt)
- Similar to correlation with *error*
- Hebb rule: $\triangle w_{ij} = \epsilon\ t_j\ a_i$ (where $t_j$ is activation "clamped" on the output unit)

# Learning on orthogonal patterns (one pass): Delta = Hebb

Delta rule: $\triangle w_{ij} = \epsilon\,(t_j - a_j)\,a_i$ (assume linear units: $a_j = n_j$)

$\boxed{\text{Note: Delta = Hebb if } a_j = 0}$

For first pattern $p_1$, $w_{ij} = 0$ so $a_j^{[p_1]} = n_j^{[p_1]} = 0$, and

$$\triangle w_{ij}\,(= w_{ij}) \;=\; \epsilon\left(t_j^{[p_1]} - 0\right) \;=\; t_j^{[p_1]}\,a_i^{[p_1]}$$

$\boxed{\text{Hebb rule with target as output activation}}$

For $p_2$, $a_j^{[p_2]} \;=\; \sum_i a_i^{[p_2]} w_{ij} \;=\; \sum_i a_i^{[p_2]}\left(t_j^{[p_1]} a_i^{[p_1]}\right) \;=\; t_j^{[p_1]} \sum_i a_i^{[p_2]} a_i^{[p_1]} \underline{\sum_i a_i^{[p_2]} a_i^{[p_1]}}$ *(dot product of $p_1$ and $p_2$)*

Since $p_1$ and $p_2$ are orthogonal, $\sum_i a_i^{[p_2]} a_i^{[p_1]} = 0$, so $a_j^{[p_2]} = 0$. Thus

$$\triangle w_{ij} = t_j^{[p_2]} a_i^{[p_2]} \qquad w_{ij} = t_j^{[p_1]} a_i^{[p_1]} + t_j^{[p_2]} a_i^{[p_2]}$$

$\boxed{\text{Hebb rule again}}$

In fact, $a_j^{[p]} = 0$ for the first presentation of each training pattern p, so at the end of one sweep through all the patterns:

$$w_{ij} \;=\; \epsilon \sum_p \left(t_j^{[p]} - a_j^{[p]}\right) a_i^{[p]} \;=\; \epsilon \sum_p t_j^{[p]} a_i^{[p]}$$

This is just **Hebbian learning** using targets $t_j$ as output activations ($a_j$).

Note that the Delta rule is inherrently *multi-pass* ($a_j \neq 0$ on subsequent presentations)

- Weight changes caused by one pattern affect error on others

# Effects of training on response to input patterns

Calculated in terms of *changes* to activations for pattern p′ caused by training on single pattern p:

$$\begin{aligned}
\triangle a_j^{[p']} &= \sum_i a_i^{[p']} \triangle w_{ij} \\
&= \sum_i a_i^{[p']} \, \epsilon \left( t_j^{[p]} - a_j^{[p]} \right) a_i^{[p]} \\
&= \epsilon \left( t_j^{[p]} - a_j^{[p]} \right) \sum_i a_i^{[p']} a_i^{[p]} \\
&= \epsilon \left( t_j^{[p]} - a_j^{[p]} \right) \, \mathrm{dp}(p', p)
\end{aligned}$$

- If p and p′ are orthogonal, training on p will have no effect on p′
- If p and p′ are not orthogonal, training on p will affect performance on p′ (weighted by similarity) which may be good (generalization) or bad (interference)

# When does the Delta rule succeed or fail?

Delta rule is *optimal*

- Will find a set of weights that produces zero error <u>if such a set exists</u>

Need to distinguish "succeed" = zero error from "succeed" = correct binary classification

Guaranteed to succeed (<u>zero error</u>) if input patterns are **linearly independent** (LI)

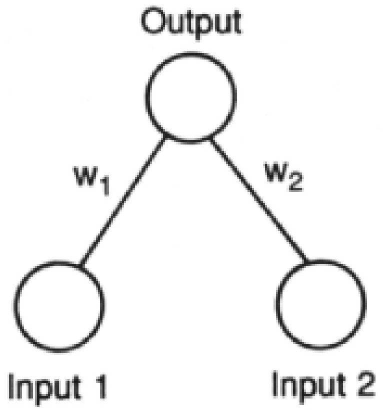
- No pattern can be created by recombining scaled versions of the others (i.e., there is *something unique* about each pattern; cf. Hebb: no similarity)
- Orthogonal patterns are linearly independent (LI is a weaker constraint)
- Linearly independent patterns can be similar as long as other aspects are unique

Succeed at binary <u>classification</u> of outputs: **Linear separability**

# Linear separability

Delta rule is guaranteed to succeed at binary classification if the task is **linearly separable**

- Weights define a plane (line for two input units) through input (state) space for which $n_j = 0$
- Must be possible to position this plane such that all patterns requiring $n_j < 0$ are on one side and all patterns requiring $n_j > 0$ are on the other side
- Property of the relationship between input and target patterns
- **AND** and **OR** are linearly separable but **XOR** is not

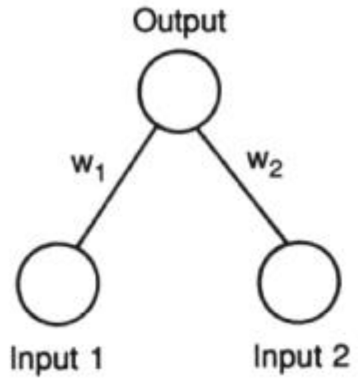



$$n_j = a_1 w_1 + a_2 w_2 + b_j \;=\; 0$$

$$a_2 = -\frac{w_1}{w_2}\, a_1 \;-\; \frac{b_j}{w_2}$$

$$(y = \quad \mathrm{a} \quad x \quad + \; \mathrm{b})$$

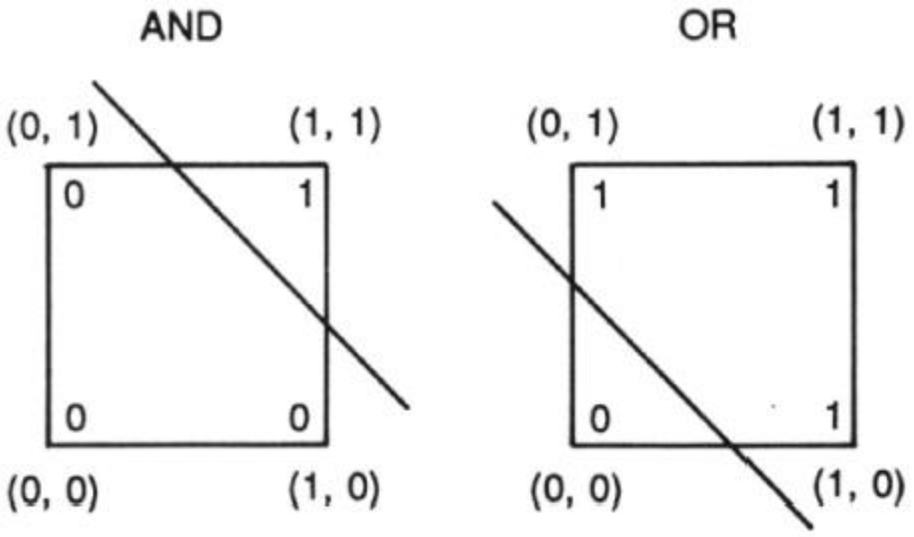
# XOR



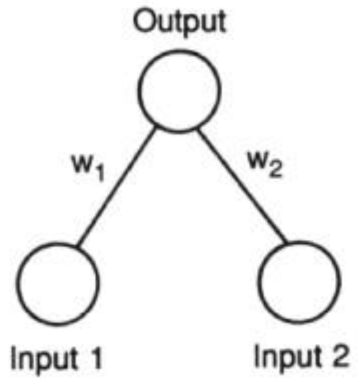

$$n_j = a_1 w_1 + a_2 w_2 + b_j \;=\; 0$$

$$a_2 = -\frac{w_1}{w_2}\; a_1 \;-\; \frac{b_j}{w_2}$$

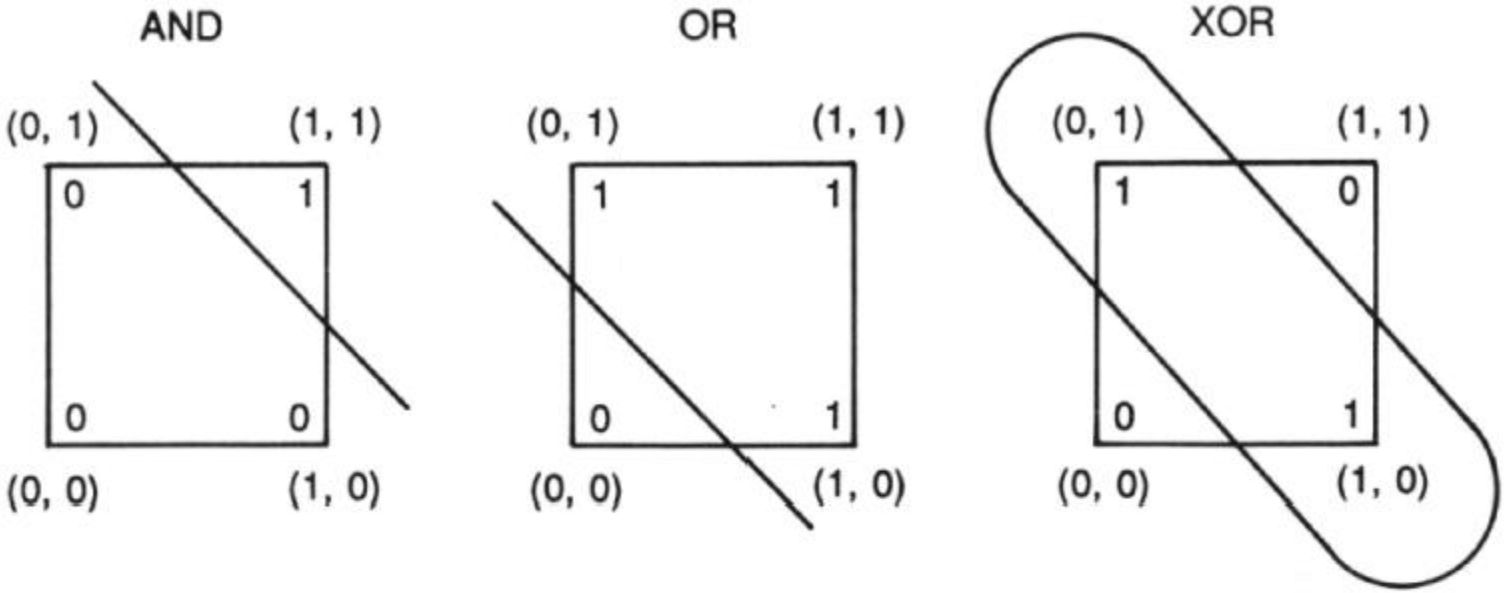
$$(y = \quad \mathrm{a} \quad x \quad + \;\mathrm{b})$$



# XOR




$$n_j = a_1 w_1 + a_2 w_2 + b_j \; = \; 0$$
$$a_2 = -\frac{w_1}{w_2} \; a_1 \; - \frac{b_j}{w_2}$$
$$(y = \quad \mathrm{a} \quad x \quad + \; \mathrm{b})$$

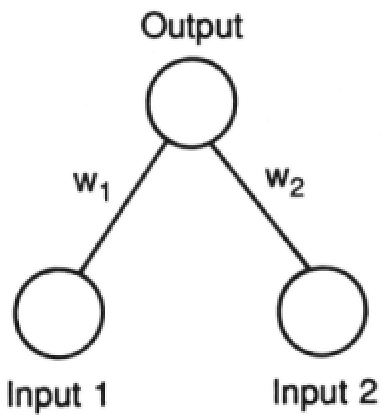


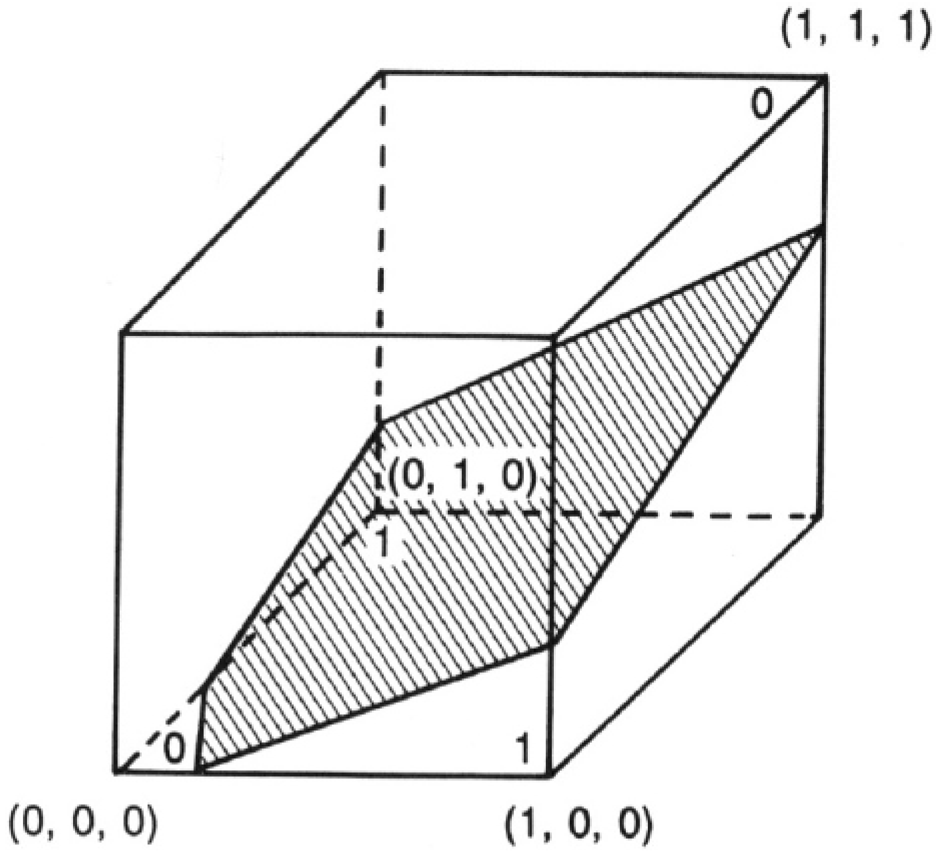
# XOR with extra dimension

XOR task can be converted to one that is linearly separable by adding a new "input"

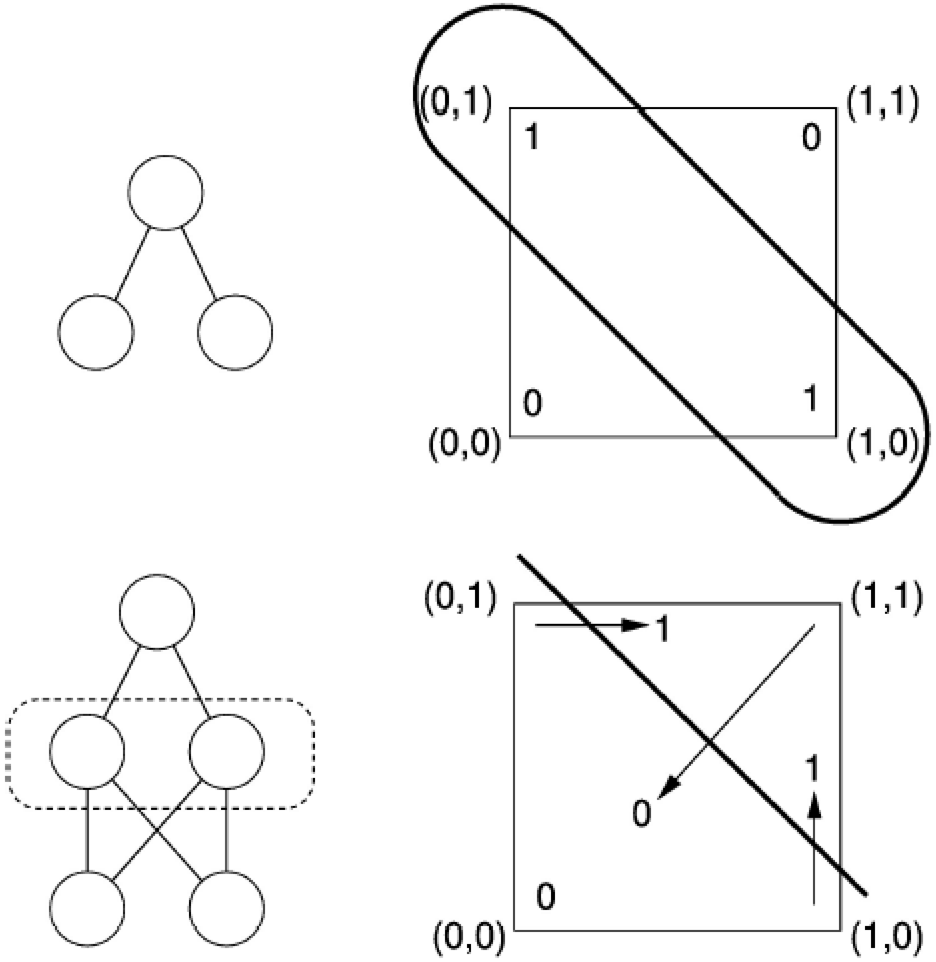
- Corresponds to a third dimension in state space
- Task is no longer XOR

| Inputs | | | Output |
|---|---|---|---|
| 0 | 0 | **0** | 0 |
| 0 | 1 | **0** | 1 |
| 1 | 0 | **0** | 1 |
| 1 | 1 | **1** | 0 |



# XOR with intermediate ("hidden") units




- Intermediate units can re-represent input patterns as new patterns with **altered similarities**
- Targets which are not linearly separable in the input space can be linearly separable in the intermediate representational space
- Intermediate units are called "hidden" because their activations are not determined directly by the training environment (inputs and targets)