

Connectionist Theory

October 10, 2003

1. Intro

- (1) Connectionist networks
 - a. Neural models attempt to faithfully model all the biological details of neurons
 - b. Connectionist networks ignore all but the most basic general properties of neurons, hopefully, the most important for neural *computation*
 - i. Necessary so they can be mathematically analyzed
 - ii. This analytic understanding essential for enabling the design of networks sufficiently powerful to compute the complex cognitive functions of higher cognition
- (2) Components of Connectionist Theory
 - a. Representation (*next topic*)
 - ◆ Interface between the network and the problem domain (the domain being cognized)
 - b. Processing (*previous topic*)
 - ◆ Activation spread from input to output, with constant connections
 - c. Learning (*today's topic*)
 - ◆ Modification of connections with experience in the problem domain
- (3) Local vs. Distributed representation (*next topic*)
 - a. Local
 - ◆ each unit's activity can be separately interpreted
 - ◆ one unit \Leftrightarrow one concept from the problem domain
 - ◇ e.g., one unit \Leftrightarrow one word
 - b. Distributed
 - i. Definition
 - ◆ one unit's activity \Rightarrow many concepts
 - ◆ one concept \Rightarrow many unit's activity
 - ii. Fully distributed representation
 - ◆ activity of a single unit is uninterpretable
 - ◆ only the entire pattern of activity of a whole set of units can be interpreted
 - iii. Microfeatural representation
 - ◆ activity of a single unit interpretable as a 'microfeature', a lower-level property than a concept from the problem domain
 - ◆ e.g., NetTalk:
 - ◇ output is a speech sound, a phonetic segment like [i] the vowel sound in 'we'
 - ◇ in a local representation, there would be one unit for [i]
 - ◇ in NetTalk, [i] is represented by a pattern of activity over all the output units
 - ◇ each individual unit can be interpreted as a 'microfeature', that is,
 - ◇ a property such as 'hi frequency', 'front obstruction', 'vowel', ...
 - ◇ if the sound [i] has one of these properties, the corresponding unit is active; otherwise it is inactive
- (4) Activation vectors
 - a. A 'pattern of activity' is a list of activation values, (0.9, 0.8, -0.2, ...)
 - b. This defines the *activation vector* of the network: **a**

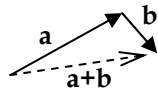
2. Vectors

(5) Notations

- a. Variable: in type, boldface: **a**; handwritten, with an over-arrow: \vec{a} or \bar{a}
- b. List of numbers (*components*): (0.9, 0.8, -0.2, ...); if n numbers, an “ n -dimensional” vector
- c. Arrow: in n -dimensional space: tail at origin, tip at $x = 0.9, y = 0.8, z = -0.2, \dots$
- d. Graph: for successive points on the x axis: 1, 2, 3, ... plot the values: 0.9, 0.8, -0.2, ...
- e. Graphic: a sequence of circles with areas: 0.9, 0.8, 0.2, ... and some convention for showing \pm sign (e.g., white circles for positive, black circles for negative)

(6) Most important vector operation: Addition

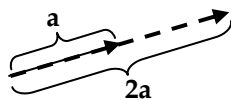
- a. Variable: **a + b**
- b. List of numbers: add corresponding components
 $(0.9, 0.8, -0.2, \dots) + (0.2, -0.1, -0.3, \dots) = (1.1, 0.7, -0.5, \dots)$
- c. Arrow: Take “step **a**” followed by “step **b**” (put the tail of one arrow at the tip of the other)



- d. Graph: superimpose the “waves” (at each point, add the heights of **a** and **b** above the x axis)
- e. Graphic: no natural correspondent of addition (“add the areas”)

(7) Second primitive vector operation: Scalar multiplication

- a. Variable: **2a** – the *scalar product* of the number (“scalar”) 2 and the vector **a**
- b. List of numbers: multiply each component in **a** by 2
 $2(0.9, 0.8, -0.2, \dots) = (1.8, 1.6, -0.4, \dots)$
- c. Arrow: Multiply the length of **a** by 2 (keeping the direction unchanged) – change the ‘scale’



- d. Graph: double the amplitude of the “wave” (at each point, make the graph twice as far from the x axis)
- e. Graphic: double the areas

(8) Length (magnitude) of a vector

- a. If $\mathbf{a} = (a_1, \dots, a_n)$ then its magnitude is $|\mathbf{a}|^2 = (a_1)^2 + \dots + (a_n)^2 = \sum_{i=1}^n a_i^2$
- b. $|\mathbf{ca}| = |c| |\mathbf{a}|$

(9) Relevance to connectionism

- a. “pattern of activity” ~ *direction* of activation vector
- b. *strength* of a pattern of activity ~ length of activation vector
- c. Connectionist networks can have pattern representing the word CAKE to a varying degree of ‘strength’ (which grows as the network’s confidence grows that this is indeed the correct word)
- d. Symbolic algorithms can’t have a symbol CAKE with varying strength

(10) Linear combination

- a. A linear combination of **a** and **b** is a weighted sum, e.g.,
 - ◆ $2\mathbf{a} + 3\mathbf{b}$
- b. This is a vector lying in the plane determined by **a** and **b**
- c. Every vector in this plane is a linear combination of **a** and **b** for some pair of weights ...
- d. ... as long as **a** and **b** are *linearly independent*, i.e., don’t lie on the same line

(11) Inner or 'dot' or scalar product

a. Definition

◆ If $\mathbf{a} = (a_1, \dots, a_n)$ and $\mathbf{b} = (b_1, \dots, b_n)$ then $\mathbf{a} \cdot \mathbf{b} = a_1b_1 + \dots + a_nb_n = \sum_{i=1}^n a_ib_i$

b. This is a measure of the correlation between the sequences (a_1, \dots, a_n) and (b_1, \dots, b_n)

c. Key property ('bilinearity')

◆ $(u\mathbf{a} + v\mathbf{b}) \cdot \mathbf{x} = u(\mathbf{a} \cdot \mathbf{x}) + v(\mathbf{b} \cdot \mathbf{x})$

(12) Orthogonality

a. When \mathbf{a} and \mathbf{b} are perpendicular, their dot product is zero: $\mathbf{a} \cdot \mathbf{b} = 0$

b. "Orthogonal" = perpendicular

c. Orthogonal: The series of numbers in \mathbf{a} is uncorrelated with that in \mathbf{b}

d. Orthogonal patterns are maximally dissimilar

e. $\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta$ where θ = angle between arrows \mathbf{a} and \mathbf{b}

(13) Similarity

a. Similar patterns go in similar directions

b. Angle between them is approx zero

c. $\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta = |\mathbf{a}| |\mathbf{b}| \text{sim}(\mathbf{a}, \mathbf{b})$

(14) Linear independence

a. Definition for two vectors:

◆ \mathbf{a} and \mathbf{b} are linearly independent iff there is no scalar v for which $\mathbf{a} = v\mathbf{b}$ and no scalar u for which $\mathbf{b} = u\mathbf{a}$

◆ simpler: \mathbf{a} and \mathbf{b} are linearly independent iff there are no scalars u and v such that

$$u\mathbf{a} + v\mathbf{b} = \mathbf{0}$$

b. Definition for n vectors:

◆ $\{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n\}$ are linearly independent vectors iff there is no set of scalars $\{u_1, u_2, \dots, u_n\}$ s.t.

$$u_1\mathbf{a}_1 + u_2\mathbf{a}_2 + \dots + u_n\mathbf{a}_n = \mathbf{0} \text{ or: } \sum_{i=1}^n u_i\mathbf{a}_i = \mathbf{0}$$

◆ $\{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n\}$ are linearly independent vectors iff no one of them is a linear combination of the others

c. If $\{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n\}$ are linearly independent then the set of all linear combination of these vectors is an n -dimensional space

d. In order for $\{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n\}$ to be linearly independent the dimension of the vectors (the number of component numbers they contain) must be at least n

e. Orthogonal vectors must be linearly independent

3. Linear Associators: Processing

(15) Simplest network: Linear associator

a. Two layers: input, output

b. Feed-forward connections only

c. Linear activation function:

i. activation of output unit j = sum of activation flowing into j from all input units i

ii. activation flowing into output unit j from input unit $i = w_{ji} a_i$

(16) The general form of the linear activation equation for a linear associator:

$$o_j = \sum_i w_{ji} i_i$$

◆ o_j is the activation value of the j th output unit

◆ i_i is the activation value of the i th input unit

(17) AS-ISNet: feed-forward simplification of Rumelhart & McClelland 1981's model for the perception of letters in words

(18) AS-ISNet

a. The particular form for the output unit for the word AS:

$$\begin{aligned}
 o_{AS} &= w_{AS,1}i_1 + w_{AS,2}i_2 + w_{AS,3}i_3 + \dots \\
 &= (w_{AS,1}, w_{AS,2}, w_{AS,3} \dots) \cdot \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ \vdots \end{pmatrix} \\
 &= \mathbf{w}_{AS} \cdot \mathbf{i} = \bar{\mathbf{w}}_{AS} \cdot \bar{\mathbf{i}}
 \end{aligned}$$

- i. \mathbf{w}_{AS} is the *weight (row) vector* of the output unit for the word AS.
- ii. \mathbf{i} is the *input (column) (activation) vector*
- iii. \cdot (or \cdot) is the *dot product (or inner product)* of the weight vector and the input vector

b. Repeating this equation for the output unit for the word IS, and stacking the two equations:

$$\begin{aligned}
 \begin{pmatrix} o_{AS} \\ o_{IS} \end{pmatrix} &= \begin{bmatrix} (w_{AS,1}, w_{AS,2}, w_{AS,3} \dots) \\ (w_{IS,1}, w_{IS,2}, w_{IS,3} \dots) \end{bmatrix} \cdot \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ \vdots \end{pmatrix} \\
 &= \begin{bmatrix} \mathbf{w}_{AS} \\ \mathbf{w}_{IS} \end{bmatrix} \cdot \mathbf{i} = \mathbf{W} \cdot \mathbf{i}
 \end{aligned}$$

- ◆ \mathbf{W} is the *weight matrix* and \cdot is the *matrix-vector product* of \mathbf{W} and \mathbf{i}
- ◆ alternatively: $\mathbf{o} = \begin{pmatrix} o_{AS} \\ o_{IS} \end{pmatrix} = \begin{pmatrix} \mathbf{w}_{AS} \cdot \mathbf{i} \\ \mathbf{w}_{IS} \cdot \mathbf{i} \end{pmatrix}$

(19) Example: With the input vector for the word AS, $\mathbf{i}^{(AS)}$, we want the output unit for IS to be zero:

$$\mathbf{w}_{IS} \cdot \mathbf{i}^{(AS)} = 0$$

This means that these two vectors must be *orthogonal* (perpendicular): $\mathbf{w}_{IS} \perp \mathbf{i}^{(AS)}$ – no similarity

(20) Interpretation of incoming weight vector \mathbf{w}_j for output unit j : for input vector \mathbf{i} , activation is:

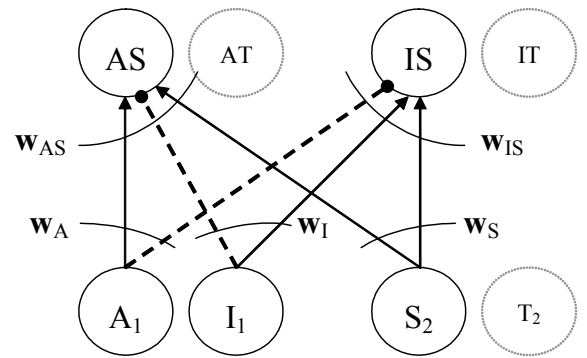
$$o_j = \mathbf{w}_j \cdot \mathbf{i} = |\mathbf{w}_j| |\mathbf{i}| \text{sim}(\mathbf{w}_j, \mathbf{i})$$

(21) The incoming weight vector of an output unit is the template it matches to the input. The closer the input pattern matches the template, the greater the output unit's activation. The output activation also increases as the strength of the input pattern, or the magnitude of the weight vector, increases.

(22) Interpretation of outgoing weight vector \mathbf{w}_k for input unit k , with activation i_k : activation sent to output layer is:

$$\mathbf{a} = \mathbf{w}_k i_k$$

(23) The outgoing weight vector of an input unit is the pattern that unit 'sprays' on the output layer. The stronger the input unit (and the larger the magnitude of the weight vector), the stronger this pattern is sprayed on the output.



4. Linear Associators: Learning

(24) ASIsNet: how to determine the weights from the training examples?

(25) Hebbian learning: increase strength (by ϵ) of every connection between simultaneously active units

$$\Delta w_{jk} = \epsilon o_j i_k = \epsilon [\mathbf{o} \mathbf{i}^T]_{jk} = \epsilon [\mathbf{o} \otimes \mathbf{i}]_{jk}$$

where \otimes is the *tensor (outer vector) product*.

(26) *Tensor product* (or 'outer vector product')

a. Definition: $\mathbf{a} \otimes \mathbf{b} = \begin{matrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \\ \otimes \end{matrix} \begin{pmatrix} b_1 & b_2 & \dots & b_k \end{pmatrix} = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \dots & a_1 b_k \\ a_2 b_1 & a_2 b_2 & \dots & a_2 b_k \\ \vdots & \vdots & \ddots & \vdots \\ a_n b_1 & a_n b_2 & \dots & a_n b_k \end{bmatrix}$

b. Key property: $[\mathbf{a} \otimes \mathbf{b}] \cdot \mathbf{c} = \mathbf{a} [\mathbf{b} \cdot \mathbf{c}]$

(27) For the weight matrix as a whole:

$$\Delta \mathbf{W} = \epsilon \mathbf{o} \otimes \mathbf{i}$$

(28) For all patterns p , total Hebbian weight matrix is:

$$\mathbf{W} = \epsilon \sum_p \mathbf{o}^{(p)} \otimes \mathbf{i}^{(p)}$$

(29) Given a linear associator with weight matrix $\mathbf{W} = \epsilon \mathbf{o}^{(1)} \otimes \mathbf{i}^{(1)}$, the output for an input \mathbf{i} is the pattern $\mathbf{o}^{(1)}$ with strength $\epsilon \text{sim}(\mathbf{i}, \mathbf{i}^{(1)}) |\mathbf{i}| |\mathbf{i}^{(1)}|$

(30) Therefore, for the Hebbian weight matrix as a whole,

$$\text{output for } \mathbf{i} = \epsilon \sum_p \mathbf{o}^{(p)} \text{sim}(\mathbf{i}^{(p)}, \mathbf{i}) |\mathbf{i}| |\mathbf{i}^{(p)}|$$

(31) To get correct association, need $\text{sim}(\mathbf{i}^{(p)}, \mathbf{i}^{(p')}) = 0$ when $p \neq p'$; therefore:

(32) Hebbian learning works iff input vectors are orthogonal

(33) Convenient way of thinking: Two 'phases' or 'modes' of supervised learning:

phase⁺: the 'correct/desired/target' output pattern **is** imposed ('clamped') on the output units: \mathbf{o}^+

phase⁻: " " **is not** " ; output is computed: \mathbf{o}^-

Computed output \mathbf{o}^- : linear activation equation

(34) Linear output units: the *computed* output is \mathbf{o}^- :

$$o_j^- = \sum_k w_{jk} i_k$$

o_j^- is the activation value of the j^{th} output unit (computed by network; not an imposed target)

i_k is the activation value of the k^{th} input unit

(35) Hebbian learning uses only the *correct* or '*target*' output vector \mathbf{o}^+

a. Change of weights for a single pattern p

$$\Delta^{(p)} w_{jk} = \epsilon o_j^{+(p)} i_k^{(p)} \quad \text{or}$$

$$\Delta^{(p)} \mathbf{W} = \epsilon \mathbf{o}^{+(p)} \otimes \mathbf{i}^{(p)}$$

b. Final weights after learning all patterns

$$\mathbf{W} = \epsilon \sum_p \mathbf{o}^{(p)} \otimes \mathbf{i}^{(p)}$$

(36) Example: ASISNet, trained on only two words (“patterns”): AS and IS

a. Learned weight matrix

$$\mathbf{W} = \epsilon \mathbf{o}^{+(AS)} \otimes \mathbf{i}^{(AS)} + \epsilon \mathbf{o}^{+(IS)} \otimes \mathbf{i}^{(IS)}$$

b. Output computed from these weights, after training, for a new input \mathbf{i}

$$\begin{aligned} \mathbf{o}^{-(p)} &= \mathbf{W} \cdot \mathbf{i} \\ &= \left[\epsilon \mathbf{o}^{+(AS)} \otimes \mathbf{i}^{(AS)} + \epsilon \mathbf{o}^{+(IS)} \otimes \mathbf{i}^{(IS)} \right] \cdot \mathbf{i} \\ &= \underbrace{\left[\epsilon \mathbf{o}^{+(AS)} \otimes \mathbf{i}^{(AS)} \right] \cdot \mathbf{i}}_{\text{contribution of AS weights}} + \underbrace{\left[\epsilon \mathbf{o}^{+(IS)} \otimes \mathbf{i}^{(IS)} \right] \cdot \mathbf{i}}_{\text{contribution of IS weights}} \end{aligned}$$

c. Interpreted in terms of similarity:

$$\begin{aligned} \mathbf{o}^{-(p)} &= \left[\epsilon \mathbf{o}^{+(AS)} \otimes \mathbf{i}^{(AS)} \right] \cdot \mathbf{i} + \left[\epsilon \mathbf{o}^{+(IS)} \otimes \mathbf{i}^{(IS)} \right] \cdot \mathbf{i} \\ &= \epsilon \mathbf{o}^{+(AS)} \left[\mathbf{i}^{(AS)} \cdot \mathbf{i} \right] + \epsilon \mathbf{o}^{+(IS)} \left[\mathbf{i}^{(IS)} \cdot \mathbf{i} \right] \\ &= \epsilon \mathbf{o}^{+(AS)} \left| \mathbf{i} \right| \left| \mathbf{i}^{(AS)} \right| \text{sim}[\mathbf{i}, \mathbf{i}^{(AS)}] + \epsilon \left| \mathbf{i} \right| \left| \mathbf{i}^{(IS)} \right| \text{sim}[\mathbf{i}, \mathbf{i}^{(IS)}] \mathbf{o}^{+(IS)} \end{aligned}$$

d. Consider $\mathbf{i} = \mathbf{i}^{(p)}$ for some word $p = AS$ or IS

	$\mathbf{i}^{(p)}$	$\mathbf{i}^{(AS/IS)}$	$\left \mathbf{i}^{(p)} \right \left \mathbf{i}^{(AS/IS)} \right \text{sim}[\mathbf{i}^{(p)}, \mathbf{i}^{(AS/IS)}]$
a.	$\mathbf{i}^{(AS)} = (1 \ 0 \ 1)$	$\mathbf{i}^{(AS)} = (1 \ 0 \ 1)$	$1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 = 2$
b.	$\mathbf{i}^{(AS)} = (1 \ 0 \ 1)$	$\mathbf{i}^{(IS)} = (0 \ 1 \ 1)$	$1 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 = 1$
c.	$\mathbf{i}^{(IS)} = (0 \ 1 \ 1)$	$\mathbf{i}^{(IS)} = (0 \ 1 \ 1)$	$0 \cdot 0 + 1 \cdot 1 + 1 \cdot 1 = 2$

e. E.g., if $p = AS$,

$$\begin{aligned} \mathbf{o}^{-(AS)} &= \epsilon \text{similarity}[\mathbf{i}^{(AS)}, \mathbf{i}^{(AS)}] \mathbf{o}^{+(AS)} + \epsilon \text{similarity}[\mathbf{i}^{(AS)}, \mathbf{i}^{(IS)}] \mathbf{o}^{+(IS)} \\ &= \epsilon \cdot 2 \cdot \mathbf{o}^{+(AS)} + \epsilon \cdot 1 \cdot \mathbf{o}^{+(IS)} \end{aligned}$$

f. Should be:

$$\mathbf{o}^{-(AS)} = 1 \cdot \mathbf{o}^{+(AS)} + 0 \cdot \mathbf{o}^{+(IS)}$$

g. Diagnosis: When learning AS, Hebbian learning fails to take into consideration IS, which is similar to AS, so it intrudes into the output for AS

(37) Two approaches to solving this problem (we’ll use both in the end)

a. Non-linearity: Use *threshold* units instead of a *linear* units in output

b. Error-correction: When changing the weights for a new input/output pair $(\mathbf{i}, \mathbf{o}^+)$, direct the change not to the correct output itself, but to the *difference* between what you want (\mathbf{o}^+) and what you’ve already got (\mathbf{o}^-) : *correct the error* that is made by the current weights

5. Error-correcting learning

(38) Instead of the Hebbian learning equation

$$\Delta^{(p)} \mathbf{W} = \varepsilon \mathbf{o}^{+(p)} \otimes \mathbf{i}^{(p)} \quad \text{or: } \Delta^{(p)} w_{ji} = \varepsilon o_j^{+(p)} i_i^{(p)}$$

we should have the *delta* (δ) rule

$$\Delta^{(p)} \mathbf{W} = \varepsilon \Delta \mathbf{o}^{(p)} \otimes \mathbf{i}^{(p)} \quad \text{or: } \Delta^{(p)} w_{ji} = \varepsilon \Delta o_j^{(p)} i_i^{(p)}$$

where $\Delta \mathbf{o}^{(p)} = \mathbf{o}^{+(p)} - \mathbf{o}^{- (p)}$ or: $\Delta o_j^{(p)} = o_j^{+(p)} - o_j^{- (p)}$

(39) Two views of δ -rule, an *error-correcting* learning algorithm:

a. One phase version: given input pattern p , output unit j computes the difference

$$\Delta o_j = [\text{target output value provided by the 'supervisor', } o_j^{+(p)}] - [\text{output value that would be computed by current weights, } o_j^{- (p)}]$$

and then uses this difference (or *error*) to compute the weight change: $\Delta^{(p)} w_{jk} = \varepsilon \Delta o_j^{(p)} i_k^{(p)}$

b. Two phase version: given input pattern p ,

phase⁺: target output value $o_j^{+(p)}$ is clamped, used for Hebbian learning:

$$\Delta^{+(p)} w_{jk} = \varepsilon o_j^{+(p)} i_k^{(p)}$$

phase⁻: output unit j is then unclamped, and, still with input pattern $\mathbf{i}^{(p)}$, output values $o_j^{- (p)}$ computed with current weights, and used for *anti* Hebbian learning:

$$\Delta^{- (p)} w_{jk} = -\varepsilon o_j^{- (p)} i_k^{(p)}$$

(40) Result of δ -rule learning, 'when possible':

$$\mathbf{W}^{\delta\text{-rule}} = \varepsilon \sum_p \mathbf{o}^{+(p)} \otimes \mathbf{w}^{\perp (p)} \quad \text{want: } \mathbf{O}^+ = \mathbf{W} \mathbf{I}; \quad \text{solution: } \mathbf{W} = \mathbf{O}^+ \mathbf{I}^{-1} \text{ if } \exists \mathbf{I}^{-1}$$

where $\mathbf{w}^{\perp (p)}$ is defined by: $\mathbf{O}^+ \equiv$ matrix with columns $\{\mathbf{o}^{+(p)}\}$; $\mathbf{I} \equiv$ matrix with columns $\{\mathbf{i}^{(p)}\}$

$$\mathbf{w}^{\perp (p)} \bullet \mathbf{i}^{(p')} = 0 \quad \text{for } p' \neq p, \text{ and } \mathbf{w}^{\perp (p)} \bullet \mathbf{i}^{(p)} = 1$$

'When possible': when such a set $\{\mathbf{w}^{\perp (p)}\}_p$ exists, i.e., when $\{\mathbf{i}^{(p)}\}_p$ are *linearly independent* vectors.

(41) Note that this weight matrix *performs correctly*; the *similarity* $[\mathbf{w}^{\perp (p)}, \mathbf{i}^{(p)}] = 0$, so there's no cross-talk.

(42) More generally, δ -rule learning finds the weight matrix $\mathbf{W}^{\delta\text{-rule}}$ that *minimizes summed-square error*:

$$\bar{o} \equiv \sum_p \|\delta^{(p)}\|^2,$$

where $\|\mathbf{v}\|^2 \equiv \sum_j v_j^2$ is the squared-length of the vector \mathbf{v} , and $\delta_j \equiv o_j^+ - o_j^-$ is the *error at j* ; δ is the *error vector*. That is:

$$\bar{o} = \sum_p \sum_j \delta_j^2 = \sum_p \sum_j (o_j^{+(p)} - o_j^{- (p)})^2$$

Indeed, at each step of learning, $\Delta \mathbf{W}$ takes a 'step in the direction of steepest descent of \bar{o} ', i.e., *performs gradient descent in error*. That is,

(L) $\Delta w_{jk} = -\varepsilon \partial \bar{o} / \partial w_{jk}$ in matrix/vector notation: $\Delta \mathbf{W} = -\varepsilon \nabla_{\mathbf{W}} \bar{o}$

The partial derivatives $\partial \bar{o} / \partial w_{jk}$ form the *gradient* $\nabla_{\mathbf{W}} \bar{o}$ of \bar{o} .

(43) Example: Two input units \rightarrow 1 linear output unit, one pattern, target value = 1

This prescription (L) can be generalized to arbitrarily complex networks!
(The calculus just gets harder.)

6. Generalized δ -rule: Back-propagation

- (44) The supervised, error-correcting δ -rule can be generalized to a vast array of more complex network architectures. We maintain the idea that *learning is gradient descent in error*:

$$\Delta \mathbf{W} = -\varepsilon \nabla_{\mathbf{W}} \bar{o}$$

and maintain the definition of error \bar{o}

$$\bar{o} \equiv \sum_p \|\delta^{(p)}\|^2 = \sum_p \sum_j \delta_j^2 = \sum_p \sum_j (o^{+(p)}_j - o^{-(p)}_j)^2$$

We just have a more complex way of computing the network's own output $\mathbf{o}^{(\cdot)}$, which makes \bar{o} more complex, which makes its gradient $\nabla_{\mathbf{W}} \bar{o}$ more complex.

- (45) The result: *the generalized δ -rule or backpropagation learning algorithm* (Rumelhart, Hinton, & Williams 1986). As for simple δ -rule:

$$\Delta^{(p)} \mathbf{W} = \varepsilon \delta^{(p)} \otimes \mathbf{i}^{(p)} \quad \text{or: } \Delta^{(p)} w_{ji} = \varepsilon \delta_j^{(p)} i_i^{(p)}$$

Compare to the simple δ -rule above:

$$\Delta^{(p)} \mathbf{W} = \varepsilon \Delta \mathbf{o}^{(p)} \otimes \mathbf{i}^{(p)} \quad \text{or: } \Delta^{(p)} w_{ji} = \varepsilon \Delta o_j^{(p)} i_i^{(p)}$$

where earlier $\Delta \mathbf{o} \equiv \mathbf{o}^+ - \mathbf{o}^- \equiv \delta$ for linear networks.

This applies to a layer of weights *with respect to which* \mathbf{i} is the 'input' activation pattern, δ the 'output' error.

- (46) The canonical architecture: 3 layer, feed-forward network with non-linear units with the *logistic* activation function f

$$o^-_j \equiv f(\sum_k w_{jk} i_k)$$

$$a = f(i) = \frac{1}{1 + e^{-i}} \quad \Rightarrow \quad f'(i) = a(1 - a) \text{ [derivative of } f]$$

- (47) The middle layer contains '*hidden*' units: they are hidden from the 'teacher'; their values are never set by the supervisor during learning

- (48) So now, must *compute* δ , except at output units.

- (49) For the canonical architecture (for any smooth, increasing f):

$$\delta_j^{[\text{layer } 2]} = f'(a_j) \sum_k \delta_k^{[\text{layer } 3]} w_{kj} \quad \text{"backward propagation of error":}$$

chain rule for partial derivatives

$$\delta_m^{[\text{layer } 3]} \equiv f'(a_m) (o^+_m - o^-_m) \quad \text{error at the output units}$$

Here the non-linearity of the units is f (with f' its derivative):

[for linear units, $f(a) = a$, $f'(a) = 1$]



7. Optimization rules in connectionism

(50) Learning is minimization of “error”

- a. Supervised: (input, output) pairs; gradient descent in \bar{o}
 - i. Supervisor makes it possible to compute error
 - ii. Hidden units designed during learning to get from input to output
 - iii. E.g., if inputs are not linearly independent, can’t use simple δ -rule
- b. Unsupervised: input vectors only
 - i. Not all learning is supervised
 - ii. Function minimized is not truly “error”, but a measure of how ‘useful’ a hidden unit might be, e.g.,
 - ◆ maximize its information
 - ◆ maximize its independence from other hidden units
 - ◆ maximize the ability to recreate input patterns from hidden unit activation vectors

(51) Processing is maximization of Harmony

$$H(\mathbf{a}) = \sum_j \sum_k a_j w_{jk} a_k = \sum_j \sum_k o_j w_{jk} i_k \quad \text{for a two-layer feedforward net}$$

(52) The Harmony of an activation pattern measures how consistent it is with the connections.

Harmony sums contributions from all connections. Individually, each connection’s contributions:

- a. Harmony is positive if two connected units are active (1) and they have an excitatory connection – their activations are ‘consistent’, like ‘A’ and ‘AS’ units
- b. Harmony is negative if two connected units are active and they have an inhibitory connection: – their activations are ‘inconsistent’, like ‘A’ and ‘IS’ units
- c. Harmony is zero if either unit is inactive (0)

(53) Harmony-maximizing nets do *constraint satisfaction*; each connection is a constraint with which the activation vectors must be consistent

- a. Letter-perception example with obscured lines
- b. Room-schema example

(54) Global consistency

- a. Parts of an interpretation may be locally consistent, but they are not globally consistent with one another
- b. Changing a single unit’s activation lowers Harmony: must change many units simultaneously to increase Harmony
- c. This is a *local maximum* of Harmony
- d. Want the *global maximum*
- e. Deterministic networks get ‘stuck’ at local maxima
- f. Random or ‘stochastic’ networks can escape local maxima and find global maxima

(55) Boltzmann machines / Harmony Theory

- a. Global probability distribution:

$$p(\mathbf{a}) \propto e^{H(\mathbf{a})/T}$$

- b. Simulated annealing:

As $T \searrow 0$, $p(\mathbf{a}^*) \nearrow 1$ for $\mathbf{a}^* \equiv \mathbf{a}$ that maximizes $H(\mathbf{a})$

- c. Update one-at-a-time:

$$p(a_j = 1) = 1 / (1 + e^{-\Delta H_j / T})$$

$$\Delta H_j = H(a_j = 1) - H(a_j = 0) = \sum_k w_{jk} a_k$$

(56) Learning in Boltzmann machines / Harmony Theory: *NEXT TOPIC*