Michael Arbib

Learning in Neural Networks

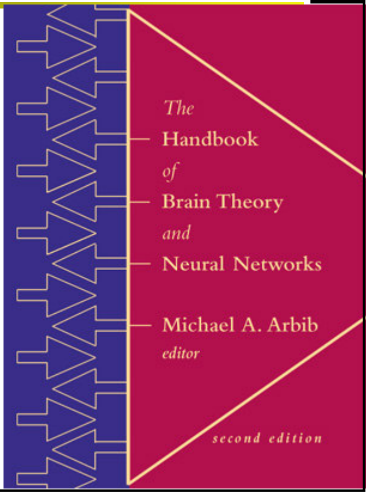
CS561: March 31, 2005

A Resource for Brain Operating Principles

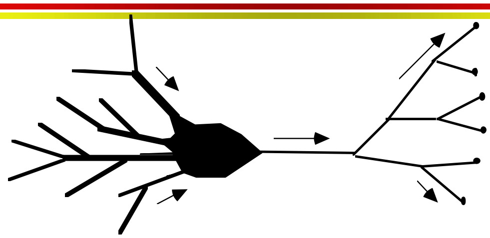
- **Grounding Models of Neurons and Networks**
- **Brain, Behavior and Cognition**
- **Psychology, Linguistics and Artificial Intelligence**
- **Biological Neurons and Networks**
- **Dynamics and Learning in Artificial Networks**
- **Sensory Systems**
- **Motor Systems**
- **Applications, Implementations and Analysis**

The Handbook is available as one of the reference works on-line at the Cognet website of The MIT Press. This can be reached from USC machines by going to USC's electronic resources site at <http://www.usc.edu/isd/elecresources>http://www.usc.edu/isd/elecresources or directly at <http://cognet.mit.edu/> http://cognet.mit.edu/library/erefs/arbib

The MIT Press, 2003



The "basic" biological neuron



Dendrites Soma Axon with branches and synaptic terminals

The soma and dendrites act as the input surface; the axon carries the outputs.

The tips of the branches of the axon form synapses upon other neurons or upon effectors (though synapses may occur along the branches of an axon as well as the ends). The arrows indicate the direction of "typical" information flow from inputs to outputs.

From Mind to Neural Networks

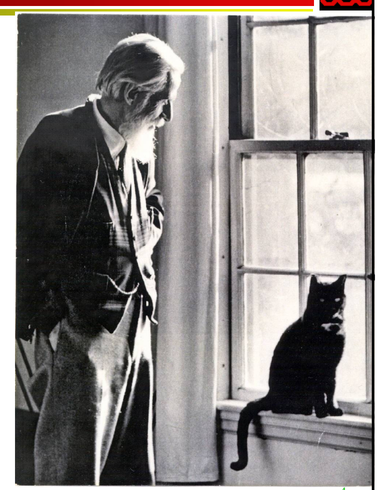
Warren McCulloch

"What is a man that he may know a number, and a number that a man may know it?"

A philosophy-driven approach:

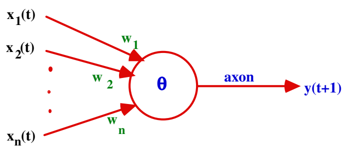
Inspired by Kant and Leibnitz, seeking to map logic onto neural function

Mathematics: Mathematical logic (propositional logic; Turing's theory of computability)



Warren McCulloch and Walter Pitts (1943)

A McCulloch-Pitts neuron operates on a discrete time-scale, t = 0,1,2,3, ... with time tick equal to one refractory period



At each time step, an input or output is

on or *off* — 1 or 0, respectively.

Each connection or synapse from the output of one neuron to the input of another, has an attached weight.

Excitatory and Inhibitory Synapses

We call a synapse

excitatory if $w_i > 0$, and

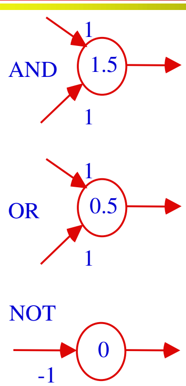
inhibitory if $w_i < 0$.

We also associate a threshold θ with each neuron

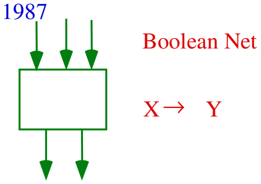
A neuron fires (i.e., has value 1 on its output line) at time t+1 if the weighted sum of inputs at t reaches or passes θ:

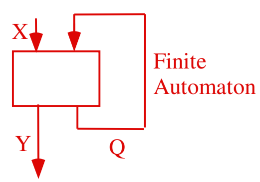
$$y(t+1) = 1 \quad \text{if and only if} \quad \Sigma\, w_i x_i(t) \geq \theta.$$

From Logical Neurons to Finite Automata



Brains, Machines, and Mathematics, 2nd Edition, 1987





Philosophically and Technologically Important

A major attack on dualism

- The "brain" of a Turing machine

A good global view of the input-output computational capacity of neural networks

Not a neuron-by-neuron account of the brain's functions:

- Logic is a culturally late activity of large neural populations, not a direct expression of neural function.

But:

An important basis for the technology of *artificial neural networks*

- with the addition of learning rules … and this is today's theme.

Samuel's 1959 Checkers Player

Programming a computer to play checkers.

Look several moves ahead: Max-min strategy

Prune the search tree

Give the computer a way to determine the value of various board positions.

Not knowing how we evaluate a board, we can at least be sure that its value depends on such things as

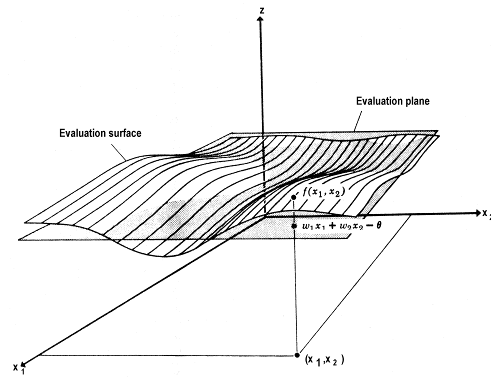
- the number of pieces each player has
- the number of kings
- balance
- mobility
- control of the center.

To these we can assign precise numbers.

Pick 16 such parameters which contribute to evaluation of the board.

Evaluation = f $(x_1, x_2, \ldots, x_{16})$ What is f?

Approximating an Evaluation Surface by a (Hyper)plane



Approximating an Evaluation Surface by a (Hyper)plane

Samuel's 1959 strategy was

in addition to cutting down the "lookahead"

to guess that

- the evaluation function was approximately linear.

… using a hyperplane approximation to the actual evaluation to play a good game:

$z = w_1x_1 + \ldots + w_{16}x_{16} - \theta$ (a linear approximation)

for some choices of the 16 weights $w_1, \ldots, w_{16}$, and θ.

In deciding which is better of two boards

the constant θ is irrelevant —

so there are only 16 numbers to find in getting the best linear approximation.

The Learning Rule

Orient an evaluation hyperplane in the 17-dimensional space:

On the basis of the current weight-setting, the computer chooses a move which appears to lead to boards of high value to the computer.

If after a while it finds that the game seems to be going badly, in that it overvalued the board it chose, then it will increase those parameters which yielded a positive contribution while reducing those that did not.

The General Theory (Lecture NN4):

Barto on *Adaptive Critics* in his HBTNN article: Reinforcement Learning

- replacing reinforcement by "expected reinforcement"

Classic Models for Adaptive Networks

The two classic learning schemes for McCulloch-Pitts formal neurons $\Sigma_i\, w_i x_i \geq \theta$

- Hebbian Learning - Amplifying Predispositions

Hebb's scheme in *The Organization of Behaviour* (1949)

- — strengthen a synapse whose activity coincides with the firing of the postsynaptic neuron
  - **[cf. Hebbian Synaptic Plasticity, HBTNN2e]**

- The Perceptron - Learning with a Teacher (Rosenblatt 1962)
  - — strengthen an active synapse if the efferent neuron fails to fire when it should have fired;
  - — weaken an active synapse if the efferent neuron fires when it should not have fired.

Hebb's scheme

Hebb (p.62, 1949):

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

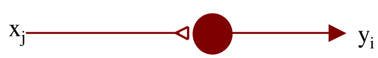
Hebb (1949) developed a multi-level model of perception and learning, in which the units of thought were encoded by cell assemblies, each defined by activity reverberating in a set of closed neural pathways.

The essence of the Hebb synapse is to increase coupling between coactive cells so that they could be linked in growing assemblies.

Hebb developed similar hypotheses at a higher hierarchical level of organization, linking cognitive events and their recall into phase sequences, temporally organized series of activations of cell assemblies.

Hebb's Rule

The simplest formalization of Hebb's rule is to increase w_{ij} by: $\Delta w_{ij} = k\, y_i\, x_j$ (1)



where synapse w_{ij} connects a presynaptic neuron with firing rate x_j to a postsynaptic neuron with firing rate y_i.

Peter Milner noted the saturation problem

von der Malsburg 1973 (modeling the development of oriented edge detectors in cat visual cortex [Hubel-Wiesel: V1 simple cells]) augmented Hebb-type synapses with

- a **normalization rule** to stop all synapses "saturating"

$$\Sigma\ w_i = \text{Constant}$$

- **lateral inhibition** to stop the first "experience" from "taking over" all "learning circuits": it prevents nearby cells from acquiring the same pattern thus enabling the set of neurons to "span the feature space"

Lateral Inhibition Between Neurons

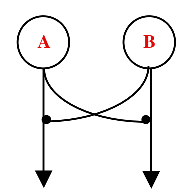
The idea is to use competition between neurons so that if one neuron becomes adept at responding to a pattern, it inhibits other neurons from doing so.

[See Competitive Learning in HBTNN]

If cell A fires better than cell B for a given orientation θ, then it fires more than B and reduces B's response further by lateral inhibition

so that A will adapt more toward θ and

B will adapt less, and the tendency will continue with each presentation of θ.



The final set of input weights to the neuron depends both

- on the initial setting of the weights, and
- on the pattern of clustering of the set of stimuli to which it is exposed.

capturing the statistics of the pattern set

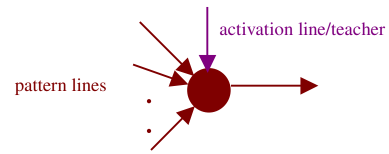
Unsupervised Hebbian Learning

tends to sharpen up a neuron's predisposition
"without a teacher,"
the neuron's firing becomes better and better correlated
with a cluster of stimulus patterns.

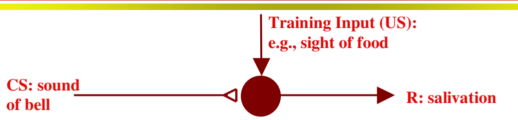
But Hebbian Learning can *be Supervised*

Supervised Hebbian Learning
is based on having an activation line separate from
the pattern lines with trainable synapses and using
the activation line to command a neuron to fire - thus
associating the firing of the neuron with those input patterns
used on the occasions when it was activated.

[This relates to the idea of associative memory.]



Supervised Hebbian learning is closely related to Pavlovian conditioning



The response of the cell being trained corresponds to the conditioned and unconditioned response (R),

the training input corresponds to the unconditioned stimulus (US), and

the trainable input corresponds to the conditioned stimulus (CS).

[Richard Thompson: US = air puff to eye; R = blink; CS = tone]

Since the US alone can fire R, while the CS alone may initially be unable to fire R,

the conjoint activity of US and CS creates the conditions for Hebb's rule to strengthen the US→R synapse,

so that eventually the CS alone is enough to elicit a response.

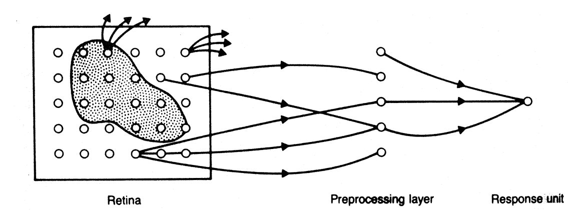
Pattern Classification by Neurons

Rosenblatt (1958) explicitly considered the problem of pattern recognition where a teacher is essential -

- for example placing b, B, b and B in the same category.

He introduced Perceptrons - neural nets that change with "experience" using an error-correction rule designed to change the weights of each response unit when it makes erroneous responses to stimuli presented to the network.

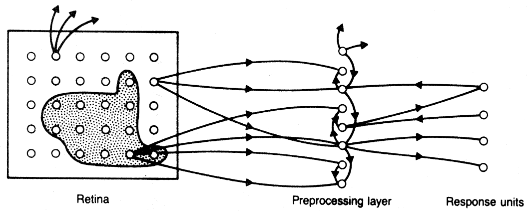
A **simple Perceptron** has no loops in the net, and only the weights to the output units can change:



Simple vs. General Perceptrons

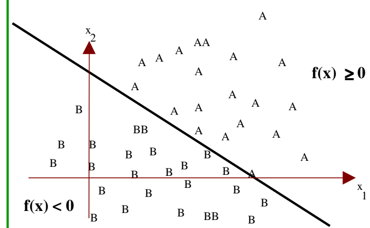
The associator units are not interconnected, and so the simple perceptron has no short-term memory.

If the units are cross-coupled - the net may then have multiple layers, and loops back from an “earlier” to a “later” layer.



Lecture NN4 will discuss back-propagation: extending perceptron techniques to loop-free multi-layer feedforward networks by “credit assignment” for “hidden Layers” between input and output.

Linear Separability



Two categories are linearly separable patterns if in fact their members can be separated by a line or (more generally) hyperplane.

A linear function of the form

$f(x) = w_1x_1 + w_2x_2 + \ldots w_dx_d + w_{d+1}$ $(w_{d+1} = -\theta)$

is a two-category pattern classifier.

$f(x) = 0 \approx w_1x_1 + w_2x_2 + \ldots w_dx_d + w_{d+1} = \theta$

gives a hyperplane as the decision surface

Training involves adjusting the coefficients $w_1, w_2, \ldots, w_d, w_{d+1}$) so that the decision surface produces an acceptable separation of the two classes.

General Training Strategy

Pick a “representative set of patterns”

Use a Training Set to adjust synaptic weights using a learning rule.

With weights fixed after training, evaluate the weights by scoring success rates on a Test Set different from the training set.

Rosenblatt (1958) provided a learning scheme with the property that

if the patterns of the training set (i.e., a set of feature vectors, each one classified with a 0 or 1) can be separated by some choice of weights and threshold,

then the scheme will eventually yield a satisfactory setting of the weights.

Perceptron Learning Rule

The best known perceptron learning rule

á strengthens an active synapse if the efferent neuron fails to fire when it should have fired, and

â weakens an active synapse if the neuron fires when it should not have:

$\Delta w_{ij} = k\,(Y_i - y_i)\,x_j$ (2)

As before, synapse w_{ij} connects a neuron with firing rate x_j to a neuron with firing rate y_i, but now

❋ Y_i is the "correct" output supplied by the "teacher."

❋ (This is similar to the Widrow-Hoff [1960] least mean squares model of adaptive control.)

The rule changes the response to x_j in the right direction:

- If the output is correct, $Y_i = y_i$ and there is no change, $\Delta w_{ij} = 0$.
- If the output is too small, then $Y_i - y_i > 0$, and the change in w_{ij} will add $\Delta w_{ij}\, x_j = k\,(Y_i - y_i)\, x_j\, x_j > 0$ to the output unit's response to $(x_1, \ldots, x_d)$.
- If the output is too large, Δw_{ij} will decrease the output unit's response.

The Perceptron Convergence Theorem

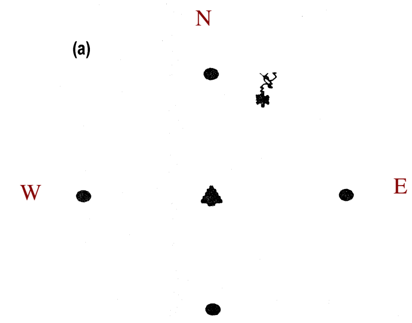
Thus, w + Δw classifies the input pattern x "more nearly correctly" than w does.

Unfortunately, in classifying x "more correctly" we run the risk of classifying another pattern "less correctly."

However, the *perceptron convergence theorem* shows that Rosenblatt's procedure does not yield an endless seesaw, but will eventually converge to a correct set of weights if one exists, albeit perhaps after many iterations through the set of trial patterns.

[See *Brains, Machines, and Mathematics*, 2nd Edition, pp. 66-69 for a proof.]

The Robot, Four Landmarks (N, E, W, S) and the Goal (the Tree "at the top of the Hill")



The "payoff function" z is a scalar quantity that increases as the robot gets closer to the goal:
Think of z as "height on a hill" so goal-seeking becomes "hill-climbing"

Hill-Climbing and Landmark Learning

"Hill-Climbing in the Fog":

At time t the robot takes a single step in direction i(t), moving from a position with payoff z(t) to one with payoff z(t+1).

If z(t+1)- z(t) > 0, then the robot's next step is in the same direction, i(t+1) = i(t), with high probability.

If z(t+1)- z(t) < 0, then i(t+1) is chosen randomly.

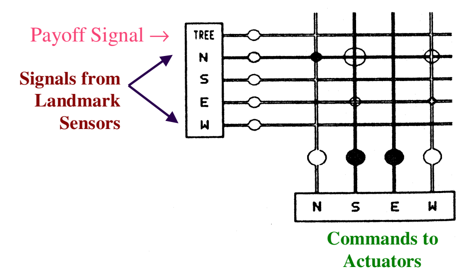
cf. bacterial chemotaxis: run-and-twiddle mechanism.

Landmark Learning

Barto and Sutton show how to equip our robot with a simple nervous system (four neurons!) which can be trained to use "olfactory cues" from the four landmarks to improve its direction-finding with experience.

The Four-Neuron Adaptive Controller

Learning Problem: Find the right synaptic weights



Note: The Payoff Signal does *not* provide explicit error signals to each actuator command.
Think of z(t)-z(t-1) as (positive or negative) reinforcement.

Hill-Climbing in Weight Space

The net can "learn" appropriate values for the weights.

We here raise hill-climbing to a more abstract level:

instead of hill-climbing in the physical space

— choose a direction again if it takes the robot uphill —

we now conduct

>>>hill-climbing in weight space<<<

At each step, the weights are adjusted in such a way as to improve the performance of the network.

The z input, with no associated weights of its own, is the "teacher" used to adjust the weights linking sensory inputs to motor outputs.

Landmark Learning

Let output $s_j(t) = \Sigma_i w_{ji}(t)x_i(t)$ (1)

Since current weights w may not yet be correct, we add a noise term, setting the output of element j at time t to

$y_j(t) = 1$ if $s(t) + NOISE_j(t) > 0$, else 0 (2)

where each $NOISE_j(t)$ is a normally distributed random variable with zero mean (each with the same variance).

The weights change according to:

$\Delta w_{ji}(t) = c[z(t)-z(t-1)]y_j(t-1)x_i(t-1)$ (3)

where c is a positive "learning rate".

Think of z(t)-z(t-1) as (positive or negative) reinforcement.

$\Delta w_{ji}(t) = c[z(t)-z(t-1)]y_j(t-1)x_i(t-1)$

w_{ji} will only change if

a j-movement takes place ($y_j(t-1)>0$) and

the "robot" is near the i-landmark ($x_i(t-1)>0$)

It will then change in the direction of z(t)-z(t-1).

Again view z(t) as "height on a hill":

w_{ji} increases and a j-movement becomes more likely —

if z increases (the "robot" moves uphill); while

w_{ji} decreases and a j-movement becomes less likely if the robot moves downhill.

The w's are shifting in an abstract 16-dimensional space of weight-settings.

Climbing a Metahill

The weights can be evaluated globally by the extent to which they determine an uphill movement, associating with a particular vector w the sum

$$S(w) = \Sigma_x E\{[z(x+y(x,w)) - z(x)]\} \quad (4)$$

z(x) is the payoff value associated with position x

z(x+y(x,w)) is the payoff associated with the position

that is reached by taking the step y(x,w)

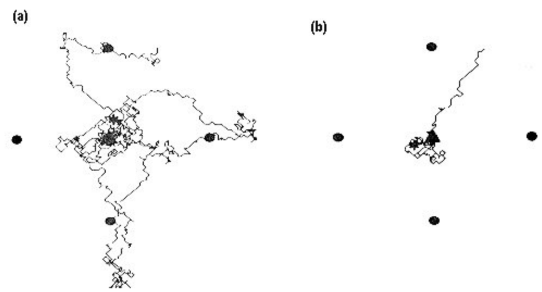
determined by (1) and (2) using the weights w, and

the expectation E averages over all the values of the noise terms in (2).

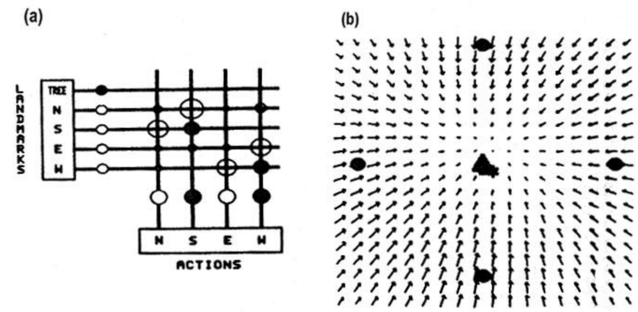
We may think of S as defining height on a "metahill."

The rule (3) tells us how to change weights in a way which is likely to increase S using just local information based on the robot's current step in physical space.

Before and After Learning



Learning a Map Implicitly



16 synaptic weights encode movement vectors at 400 locations in space

Back to the Perceptron: Training Hidden Units

In the ***simple Perceptron*** (Rosenblatt 1962) ,
only the weights to the output units can change.
This architecture **can only support linearly separable maps**.
The problem for many years was to extend the perceptron concept to multilayered networks

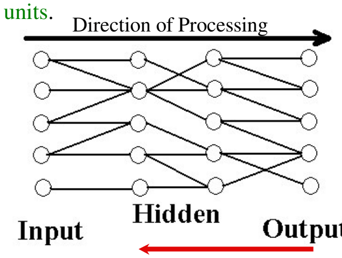
The credit assignment problem: "How does a neuron deeply embedded within a network 'know' what aspect of the outcome of an overall action was 'its fault'?"

I.e.: given an "error" which is a global measure of overall system performance, what local changes can serve to reduce global error?

Back-Propagation

Backpropagation: a method for training a loop-free network which has three types of unit:

- input units;
- hidden units carrying an internal representation;
- output units.



Direction of Training (Synapse Adjustment):
Backpropagation of Error Signals

Neurons

Each unit has both input and output taking continuous values in some range [a,b].

The response is a sigmoidal function of the weighted sum.

Thus if a unit has inputs x_k with corresponding weights w_{ik}, the output x_i is given by

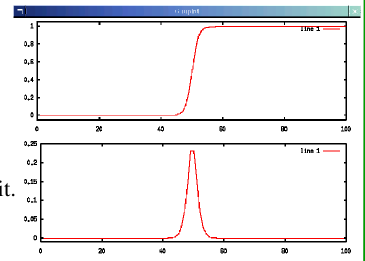
$x_i = f_i(\Sigma_k w_{ik} x_k)$

where f_i is the sigmoid function

$f_i(x) = 1/\ \{1+ \exp\ [- (x - \theta_i)]\}$

with θ_i a bias (threshold) for the unit.

tThis f is *differentiable*.



Choose a training set $\mathcal{T}$ of pairs (p,t) each comprising an input pattern p and the corresponding desired output vector t.

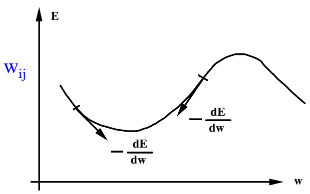
At each trial, we choose an input pattern p from $\mathcal{T}$ and consider the corresponding restricted error

$E = \Sigma_k(t_k - o_k)^2$

where k ranges over designated "output units"
with $(t_1, ...,t_n)$ the target output vector,
and $(o_1, ...,o_n)$ the observed output vector

The net has many units interconnected by weights w_{ij}. The learning rule is to change w_{ij} so as to reduce E by gradient descent.

To descend the hill, reverse the derivative.



$\Delta w_{ij} = - \partial E/\partial w_{ij} = 2\ \Sigma_k(t_k - o_k)\ \partial o_k/\partial w_{ij}$

Backpropagation

In a layered loop-free net, changing the weights w_{ij} according to the gradient descent rule may be accomplished equivalently by **back propagation**, working back from the output units. {See HBTNN I.3 for proof.}

Proposition: Consider a layered loop-free net with error measure $E = \Sigma_k(t_k - o_k)^2$, where k ranges over designated "output units," and let the weights w_{ij} be changed according to the gradient descent rule

$\Delta w_{ij} = - \partial E/\partial w_{ij} = 2\ \Sigma_k(t_k - o_k)\ \partial o_k/\partial w_{ij}$.

Then the weights may be changed inductively, working back from the output units: Δw_{ij} is proportional to $\delta_i o_j$, where:

Basis Step: $\delta_i = (t_i - o_i)f_i'$ for an output unit. [cf. Perceptron - but with added f_i' term.]

Induction Step: If i is a hidden unit, and if δ_k is known for all units which receive unit i's output then $\delta_i = (\Sigma_k\ \delta_k w_{ki})\ f_i'$, where k runs over all units which receive unit i's output. [unit i receives error propagated back from a unit k to the extent to which i affects k.]

The "error signal" δ_i propagates back layer by layer.

Backpropagation is Non-Biological

Heuristic:

The above theorem tells us how to compute Δw_{ij} for gradient descent.

It does not guarantee that the above step-size is appropriate to reach the minimum;

It does not guarantee that the minimum, if reached, is global.

The back-propagation rule defined by this proposition is thus a heuristic rule, not one guaranteed to find a global minimum.

Since it is heuristic, it may also be applied to neural nets which are loop-free, even if not strictly layered.

Non-Biological

See HBTNN articles on “Backpropagation” and “Hebbian Synaptic Plasticity”. (Optional reading.)

Critique

What such learning methods achieve:

In "many cases" (the bounds are not yet well defined)

- if we train a net N with repeated presentations of the various (x_k, y_k) from some **training set**
- then it will converge to a set of connections which enable N to compute a function f: $X \rightarrow Y$ with the property that as k runs from 1 to n, the $f(x_k)$ "correlate fairly well" with the y_k.

An end to programming? NO!!

Consider three issues:

a) complexity: Is the network complex enough to encode a solution method?

b) practicality: Can the net achieve such a solution within a feasible period of time?

c) efficacy: How do we guarantee that the generalization achieved by the machine matches our conception of a useful solution?

Programming will survive into the age of neural computing, but greatly modified

Given a complex problem, programmers will still need to

- Decompose it into subproblems
- Specify an initial structure for a separate network for each subproblem
- Place suitable constraints on (the learning process for) each network; and, finally,
- Apply debugging techniques to the resultant system.

We may expect that the initial design and constraint processes may in some cases suffice to program a complete solution to the problem without any use of learning at all.