Let's consider an example where 3 layers are useful.

Consider problem:

Teaching C-net to distinguish horizontal from vertical lines.

**Input Array**

2 horizontal lines share no common points, but...
TRIED USING A "TEACHING" INPUT

TEACHER INPUT ALWAYS HERE WHEN HORIZONTAL LINE IS PRESENT

ANALOGOUSLY, FOR VERTICAL.

RUMELHART & ZIPSER (1985)
In the original experiment:

One cluster in 2nd layer.

The 2 units in cluster do learn to discriminate between horizontal & vertical lines — but only when the teacher line is present. (The units are just responding to the teacher lines).
So, Rumelhart & Zipser tried this:

**INPUT ARRAY**

6 x 12 elements

This structure works.
BACKPROPAGATION METHODS

USING ERROR FEEDBACK TO
TRAIN THE C-NET.

REQUIRES TEACHER TO PRESENT ERROR-CORRECTING PATTERNS.

TYPICAL STRUCTURE

LEVEL 3

LEVEL 2

INPUT LEVEL

TRESHOLDS USED IN THESE LAYERS.

C-NET 34
Using thresholds means layer 2 is a subset.

If these nodes get to fire on a given input.

Different inputs cause different subsets to fire.

Steps in a given teaching cycle.

1) Input presented on layer one. Activation propagated upwards eventually an activation pattern appears on top layer.
Top Layer = Output Layer.

Red nodes are active nodes.

This pattern of activation is compared with the desired output pattern for the given input.

So, we assume we are teaching the C-net to produce a particular output pattern for a given input.
3) For each node in the layer:

If the actual activation value of the node was different from its desired value,

then modify the weights of that node so that, given the original input, the modified weights would produce a response slightly closer to the desired response.

This must be done recursively:

something as follows:
We must modify these weights first.

Since the response of A depends on the response of all nodes in the layer below A, and these in turn depend......
THE ACTUAL RECURSIVE PROCEDURE IS MODERATELY COMPLEX. (SEE PDP Vol. 1, Chapter 8).

ON EACH ERROR CORRECTING BACK PROPAGATION PASS:

THE WEIGTS OF EACH LINK BETWEEN EACH LAYER IS EXAMINED AND PROBABLY CHANGED.

EVENTUALLY, AFTER MANY ITERATIONS, THE WEIGTS ARE TUNED TO PRODUCE THE CORRECT OUTPUT FOR EACH GIVEN INPUT.
WHEN DEGRADED (PARTIALLY DISTORTED) INPUTS ARE PRESENTED, THE C-NET CAN STILL PRODUCE MOST OF THE DESIRED OUTPUT PATTERN.

NOTE, HOWEVER, SINCE BACKPROPAGATION ALWAYS REQUIRES TEACHER — NOT ALWAYS PLAUSIBLE PSYCH. MODEL.

— BUT, IS A FASTER TEACHING METHOD THAN COMPETITIVE LEARNING.
A sophisticated threshold function used for nodes above input level.

Simple thresholds use a step function, e.g.

\[(\sum \text{input} < T) \rightarrow 0 \text{ output}\]
\[(\sum \text{input} \geq T) \rightarrow 1 \text{ output}\]

But networks using backprop use a squashed function where output varies more smoothly according to \(\sum\) of input values.
ACTIVATION FUNCTION (Sigmoid)


LET $\sigma$ be sum of input values to the given node.

THEN

$$\text{Output} = \frac{1}{1 + e^{-\sigma}}$$

$O = \text{output}$

Expressing $\frac{dO}{d\sigma}$ in terms of $O$ is unusual, but convenient.

The derivative is: $O(1-O)$

(see Winston for the derivation)
E.G. *This Squash Function*

![Graph]

**Actually** "Threshold" is not appropriate term for this function. *It is an I/O Function.*

*But people do call it a threshold function, because, for inputs above or below certain amounts, output remains just about constant.*
Why do we want this I/O threshold function to be a smooth curve?

- We want the output to be roughly proportional to input.
- In order that we can modify weights to produce an output which is closer to a desired target value.
- We want to perform gradient descent in the weight space. I.e. we simultaneously vary all the weights leading to a node...
In proportion to how much good is done by the individual changes.

A basic strategy:

We make a large change in weight $w_{ij}$ if doing so causes the output of unit $j$ to be much closer to desired output.

The output of node $j$ is $\alpha$ to:

(a) The slope of threshold function
(b) The output of each node $i$, leading to $j$
(c) The weights $w_{ij}$ for each $i$.
SUPPOSE THAT:

\[ O_i = \text{output of unit } i \]

\[ O_j = \quad \text{"" } \text{"" } j \]

\[ \beta_j = \text{degree of benefit of changing unit } j \text{'s output} \]

THAT IS, THE BENEFIT W.R.T. CAUSING \( j \)'S OUTPUT TO CONTRIBUTE TO THE DESIRED TARGET OUTPUT.

IF \( j \) is a unit in the output layer,

THEN, \( j = z \), (\( z \) is the last layer)

\[ \beta_z = d_z - O_z \]

\[ \text{desired value of unit } z \]
We want to change weight $w_{ij}$ when it helps the most.

For node $j$, if the slope of threshold function at $j$ is high (given input to $j$), then make a larger change to $w_{ij}$.

So, $\Delta w_{ij} \propto o_j(1-o_j)$

Assumed slope of the squash function

Also $\Delta w_{ij}$ should be greater when the output at unit $i$ is high.

So, $\Delta w_{ij} \propto o_i$
Suppose \( o_j = \) output of unit \( j \)
\( o_i = \) output of unit \( i \)
\( \beta_j = \) degree of benefit of changing unit \( j \).

Slope is \( o_j(1-o_j) \) \( \rightarrow \) slope of threshold function.

Then the change to \( w_{ij} \)

\[
\Delta w_{ij} = r \cdot o_i \cdot o_j(1-o_j) \cdot \beta_j
\]

Proportionality constant

We assume this is the slope of the threshold function.

It is commonly used because easy to compute and it is the slope of a squash function.
WE HAVE SEEN THAT FOR NODES IN OUTPUT LAYER, \( z \)

\[ \beta_z = d_z - o_z \]

\( \downarrow \) desired value for unit \( z \)

FOR NODES IN HIDDEN LAYERS (INTERNAL LAYERS)

\[ \beta_j \text{ depends upon how much node } j \]
\[ \text{contributes to the input of each node, } k, \text{ in the next higher layer}. \]

\( \beta_j \) THAT DEPENDS UPON THE SLOPE OF THRESHOLD FUNCTION AT NODE \( k \),

\& UPON THE WEIGHT ON THE LINK FROM \( j \rightarrow k \),

\& HOW MUCH NODE \( k \) CONTRIBUTES, I.E., UPON \( \beta_k \).
So, \( \beta_j \) depends upon all three factors, for all nodes, \( k \), that receive input from \( j \).

So,

\[
\beta_j = \sum_k \omega_{jk} \cdot o_k (1 - o_k) \cdot \beta_k
\]

\( \triangleleft \) slope at node \( k \).

Note: When \( k = z \) (the output layer), then \( \beta_k \) is trivial to compute.

So, we first compute \( \beta_k \) for the output nodes. That allows us to compute \( \beta_j \) for the next lower layer.
BRINGING ALL THESE FORMULAS TOGETHER

- \( \beta_z = d_z - o_z \) for output layer NODES.

- \( \beta_j = \sum_k w_{jk} o_k (1 - o_k) \beta_k \) for internal NODES.

- \( \Delta w_{ij} = r \cdot o_i \cdot o_j (1 - o_j) \cdot \beta_j \)

If \( r \) is too large, then we overshoot the mark and errors are large.

If \( r \) is too small, then learning is very slow.

(typically \( \approx 1 \), good place to start)
STAGES IN BACKPROPAGATION

1) Find β_z for all these
2) Adjust weights
3) Find β_k for these nodes
4) Adjust weights between layers J & K

5) Repeat cycle backwards until input layer is reached.

\[ \text{Backprop of weight changes.} \]
WE CAN RUN BACKPROP AFTER EACH INPUT PATTERN IS PROCESSED.

OR SUM THE WEIGHT CHANGES AS EACH INPUT SET IS PROCESSED, AND CHANGE WEIGHTS AFTER ALL INPUTS ARE PROCESSED.

FIRST METHOD GIVES GREATER PRECISION, BUT MORE CPU INTENSIVE.
**BACKPROP: WINSTON'S VERSION.**

- Pick a rate parameter $r$
- Until performance is acceptable
  - For each sample input, do:
    - Compute the resulting output.
    - Compute $\beta$ for nodes in output layer using $\beta_z = d_z - O_z$
    - Compute $\beta$ for all internal nodes using $\beta_j = \sum_k w_{jk} \cdot O_k (1 - O_k) \cdot \beta_k$
      (for adjacent layers, $j$ & $k$)
    - Compute weight changes for all weights using $\Delta w_{ij} = r \cdot o_i \cdot o_j (1 - o_j) \cdot \beta_j$
- Add up the weight changes for all sample inputs and change the weights.
AS MENTIONED, BACKPROP CAN TAKE 100s or 1000s of iterations before "SATISFACTORY" results. SATISFACTORY? Each output node.

- If Goal of output NODE is "ONE", a value $> 0.9$ is CONSIDERED ALRIGHT.
- IF Goal of output NODE is "ZERO", a value $< 0.1$ is SATISFACTORY.

But, THIS VARIES. SOME EXPERIMENTERS GET MUCH BETTER RESULTS.

How?

Experimentally VARY:
- learning rate $\eta$,
- Number of NODES in each layer,
- Number of layers & overall design.
THRESHOLD FUNCTION


LET $\sigma$ be sum of input values to the given node.

THEN

\[
\text{Output} = \frac{1}{1 + e^{-\sigma}}
\]

$O = \text{output}$

Expressing $\frac{do}{d\sigma}$ in terms of $O$ is unusual, but convenient.

The derivative is: $O(1-O)$

(see Winston for the derivation)
Varying the learning rate $r$ is tedious process.

We want to find the maximum value of $r$ that does not lead to global instability in the learning process (i.e., the performance fluctuates rather much as we continue the number of iterations.)

Presently design & training of C-nets is still an art.

The mathematics of several basic designs is well understood - but there are many parameters to tune.
Tuning these parameters is a hill climbing problem. Easy to get stuck on local maxima.

The number of nodes & tunable weights in a network can greatly affect performance.

Having too many tunable weights can require more iterations, & also lead to worse performance when the network is tested with novel data.
A good heuristic for deciding how many nodes to use in hidden layer:

- The number of tunable weights in the network should be smaller than the number of distinct training samples. Otherwise, overfitting occurs. The weights attune to the training data too easily, & does not capture the underlying regularity in the training set. I.e., a lot of special cases are learned.
When a network learns a lot of special cases, it does not learn the rule involved.

So, the network performs poorly on the novel test data.

Chalmers’ (1990) Experiment.

**Training Data**
- 175 active-voice sentences
- 175 passive-voice sentences

**Test Data**
- 25 active-voice sentences
- 25 passive-voice sentences

His C-net learned the underlying rule, how to transform an "active structure" into "passive".