

Error-correcting learning: Delta rule

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

$$\Delta w_{ij} = \varepsilon(t_j - a_j) a_i$$

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

Weight changes focus on *predictive differences*

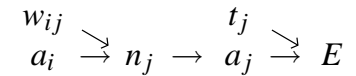
- Hebb/correlational learning depends on predictive *similarities*

Delta rule as gradient descent in error (sigmoid units)

$$n_j = \sum_i a_i w_{ij}$$

$$a_j = \frac{1}{1 + \exp(-n_j)}$$

$$\text{Error } E = \frac{1}{2} \sum_j (t_j - a_j)^2$$



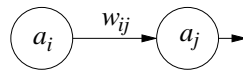
$$\text{Gradient descent: } \Delta w_{ij} = -\varepsilon \frac{\partial E}{\partial w_{ij}}$$

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial a_j} \frac{da_j}{dn_j} \frac{\partial n_j}{\partial w_{ij}} \\ &= -(t_j - a_j) a_j(1 - a_j) a_i \end{aligned}$$

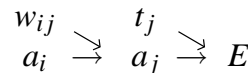
$$\Delta w_{ij} = -\varepsilon \frac{\partial E}{\partial w_{ij}} = \varepsilon(t_j - a_j) a_j(1 - a_j) a_i$$

Delta rule as gradient descent in error (linear units)

$$a_j = \sum_i a_i w_{ij}$$



$$\text{Error } E = \frac{1}{2} \sum_j (t_j - a_j)^2$$



$$\text{Gradient descent: } \Delta w_{ij} = -\varepsilon \frac{\partial E}{\partial w_{ij}}$$

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \quad (\text{Chain rule}) \\ &= -(t_j - a_j) a_i \end{aligned}$$

$$\begin{aligned} \Delta w_{ij} &= -\varepsilon \frac{\partial E}{\partial w_{ij}} = \varepsilon(t_j - a_j) a_i \\ &= \text{Delta rule} \end{aligned}$$

When does the Delta rule succeed or fail?

Delta rule is *optimal*

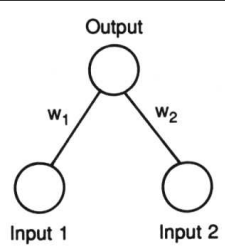
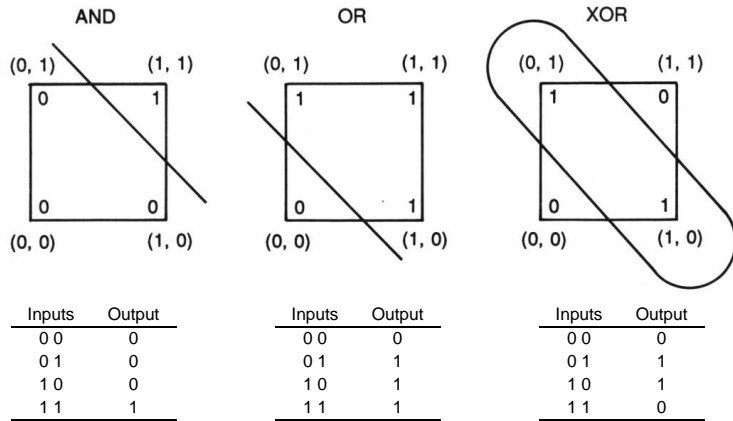
- Will find a set of weight that produces zero error *if such a set exists*

Guaranteed to succeed if input patterns are **linearly independent**

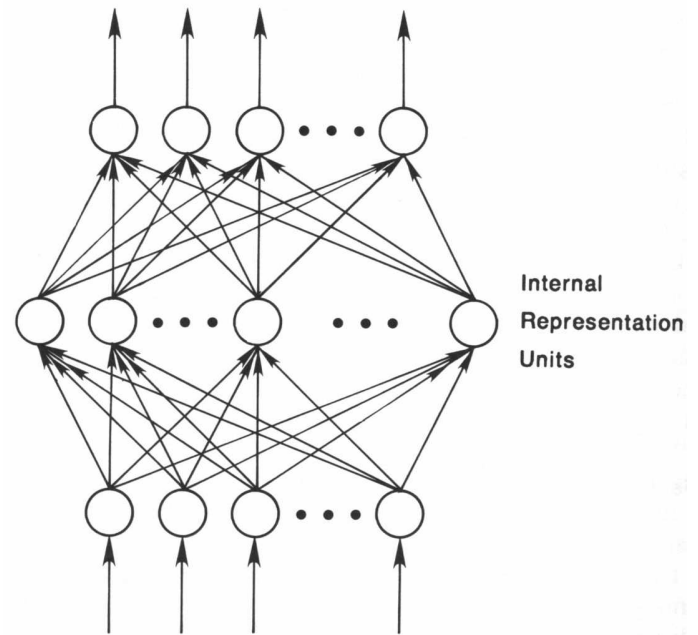
- No pattern can be created by recombining the others
- i.e., there is *something unique* about each pattern (cf. Hebb rule: no similarity (orthogonal))

Succeed at binary classification of outputs: **Linear separability**

- Weights define a plane (line for two input units) through activation (state) space
- Must be possible to position a 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
- **AND** and **OR** are linearly separable but **XOR** is not

A**B**

Output Patterns

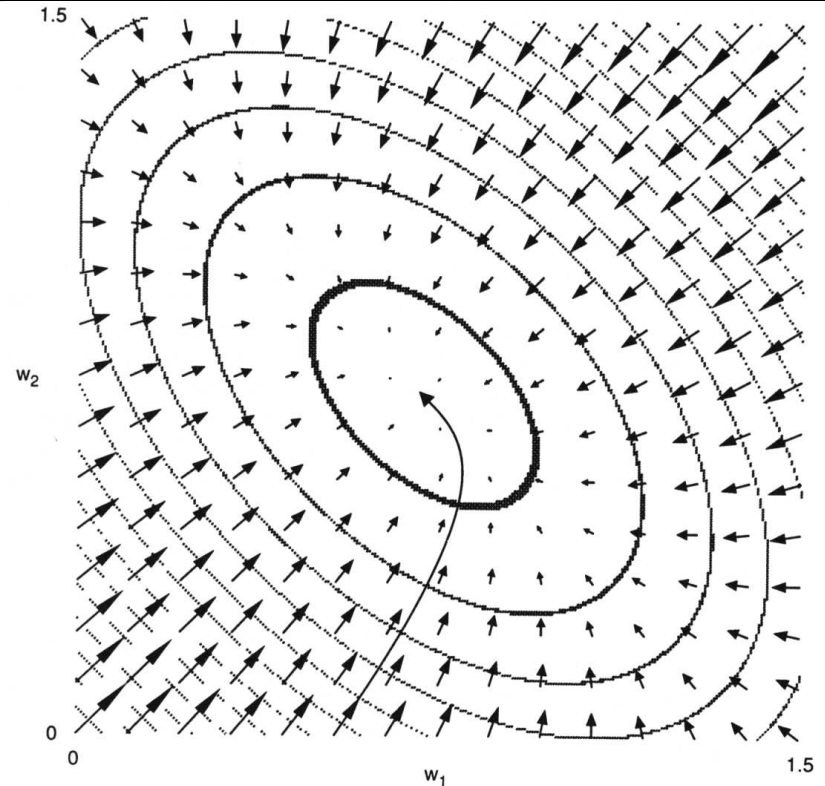
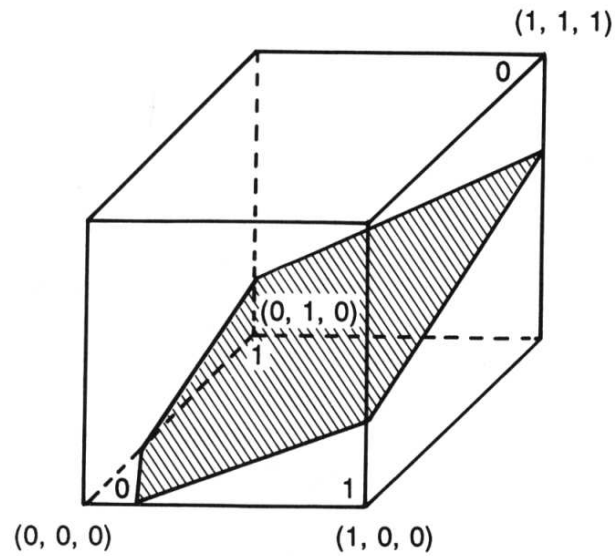
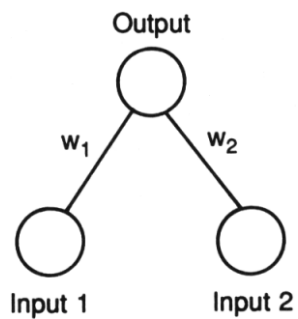


Input Patterns

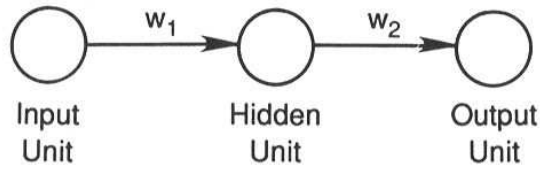
XOR can be made linearly separable by adding a new "input"

- Corresponds to a third dimension in state space
- Intermediate ("hidden") units can learn what new value is necessary

Inputs	Output
00	0
01	1
10	1
11	0



1-1-1 problem



WEIGHTS AND BIASES OF THE SOLUTIONS FOR A 1:1:1 NETWORK

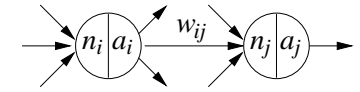
Minima	w_1	w_2	$bias_1$	$bias_2$
Global	-8	-8	+4	+4
Global	+8	+8	-4	-4
Global	-8	-8	0	0
Local	+8	+0.73	0	0

Generalized Delta Rule ("back-propagation")

$$n_j = \sum_i a_i w_{ij}$$

$$a_j = \frac{1}{1 + \exp(-n_j)}$$

$$\text{Error } E = \frac{1}{2} \sum_j (t_j - a_j)^2$$



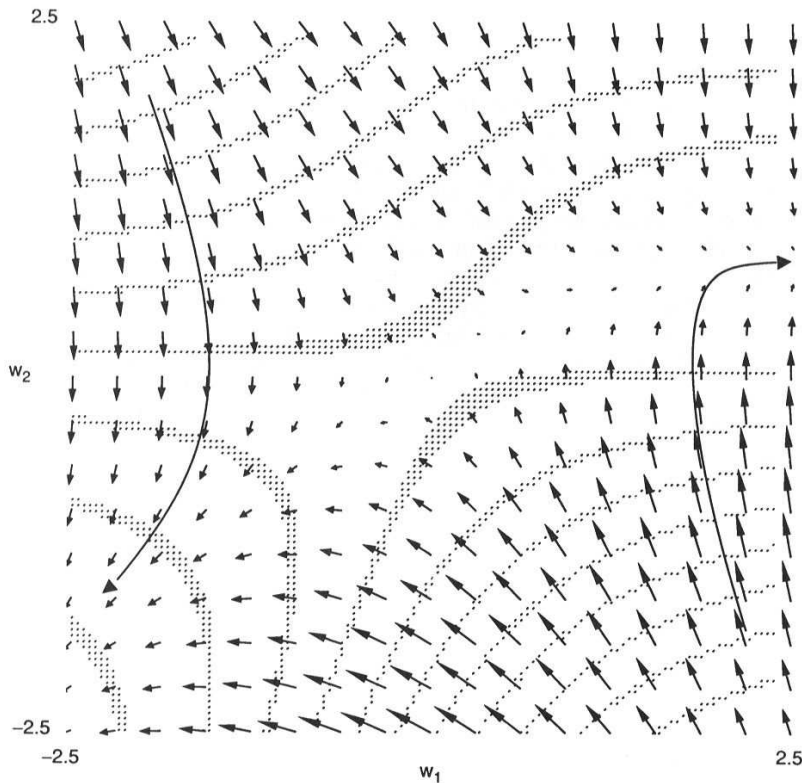
$$n_i \rightarrow a_i \xrightarrow{w_{ij}} n_j \rightarrow a_j \xrightarrow{t_j} E$$

Gradient descent: $\Delta w_{ij} = -\epsilon \frac{\partial E}{\partial w_{ij}}$

$$\frac{\partial E}{\partial n_j} = \frac{\partial E}{\partial a_j} \frac{da_j}{dn_j} = -(t_j - a_j) a_j (1 - a_j)$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial n_j} \frac{\partial n_j}{\partial w_{ij}} = \frac{\partial E}{\partial n_j} a_i$$

$$\frac{\partial E}{\partial a_i} = \sum_j \frac{\partial E}{\partial n_j} \frac{\partial n_j}{\partial a_i} = \sum_j \frac{\partial E}{\partial n_j} w_{ij}$$

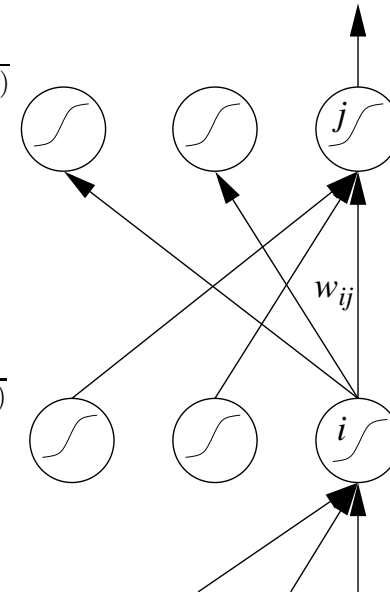


Back-propagation

$$a_j = \frac{1}{1 + \exp(-n_j)}$$

$$n_j = \sum_i a_i w_{ij}$$

$$a_i = \frac{1}{1 + \exp(-n_i)}$$



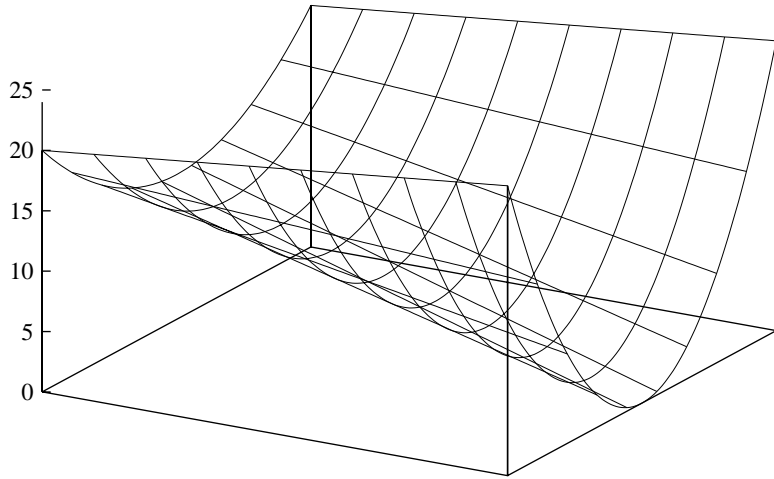
$$\frac{\partial E}{\partial a_j} = a_j - t_j$$

$$\frac{\partial E}{\partial n_j} = \frac{\partial E}{\partial a_j} a_j (1 - a_j)$$

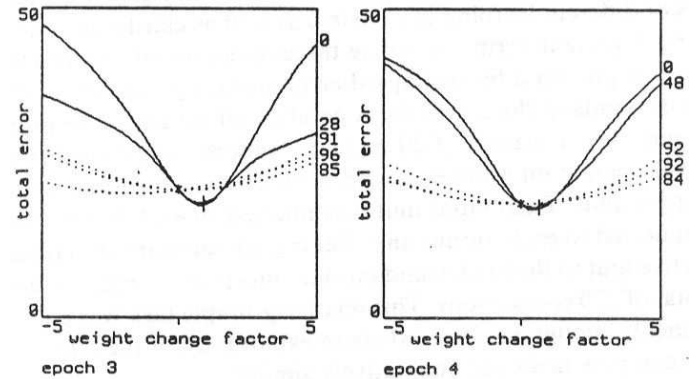
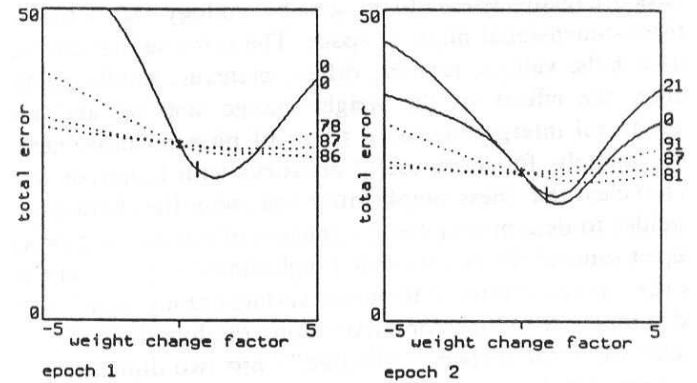
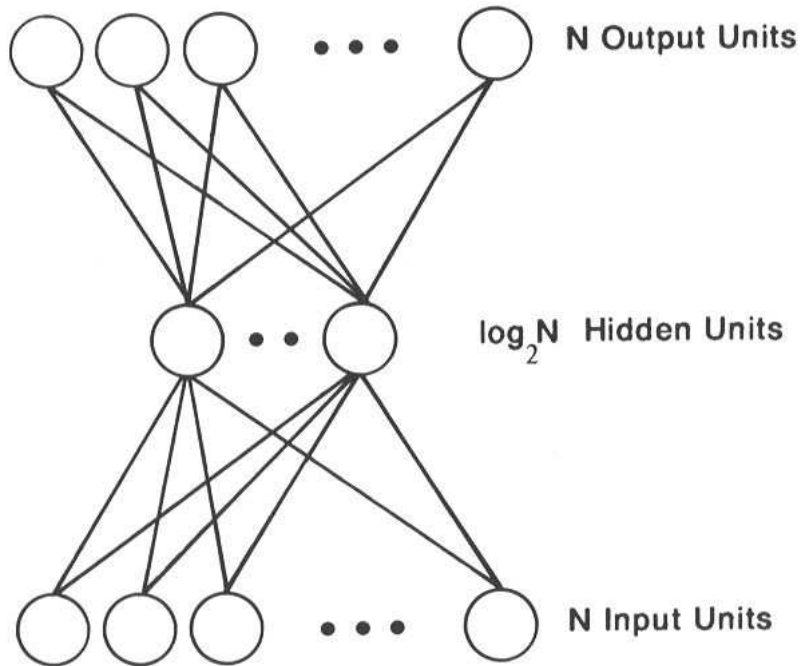
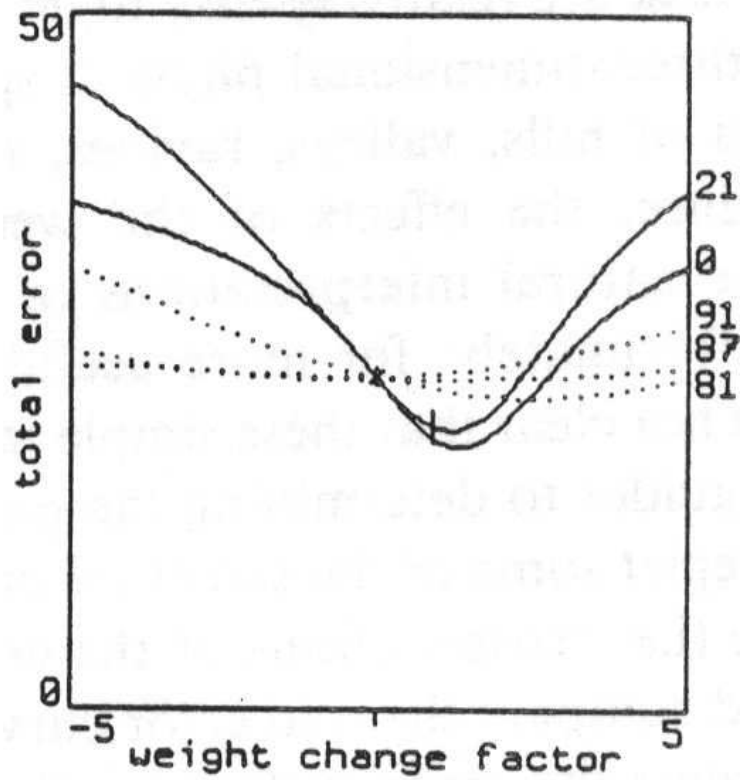
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial n_j} a_i$$

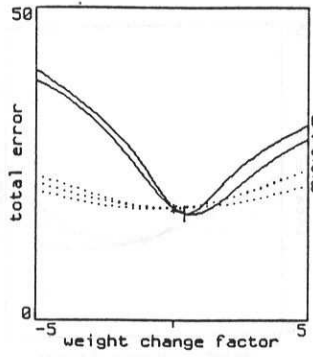
$$\frac{\partial E}{\partial a_i} = \sum_j \frac{\partial E}{\partial n_j} w_{ij}$$

Accelerating learning: Momentum descent

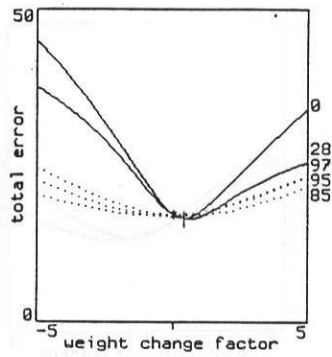


$$\Delta w_{ij}^{[t]} = -\epsilon \frac{\partial E}{\partial w_{ij}} + \alpha (\Delta w_{ij}^{[t-1]})$$

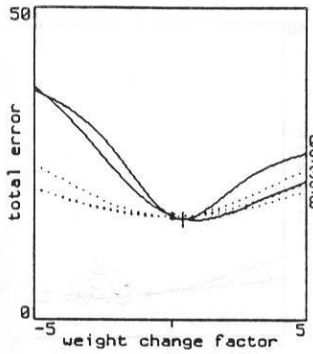




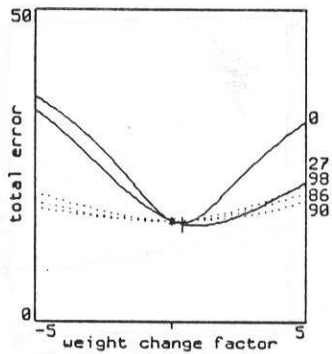
epoch 5



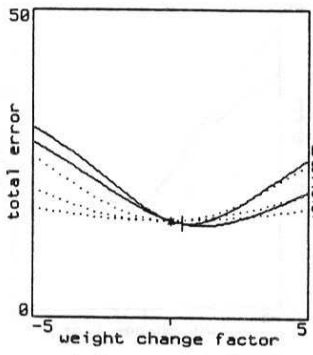
epoch 6



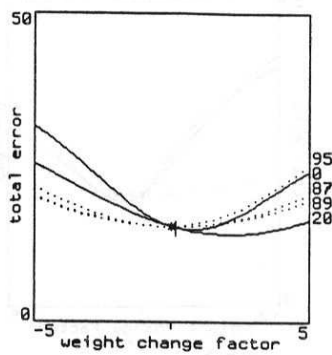
epoch 7



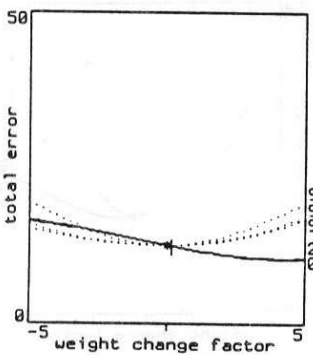
epoch 8



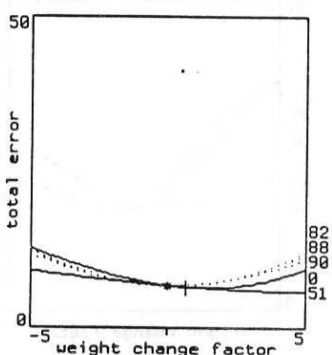
epoch 9



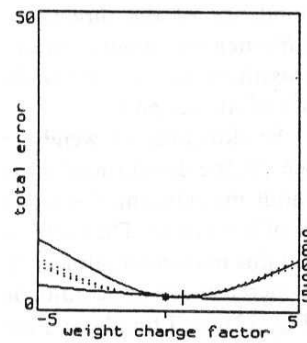
epoch 10



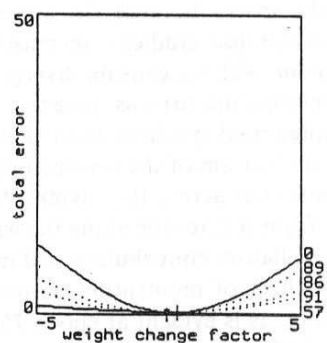
epoch 25



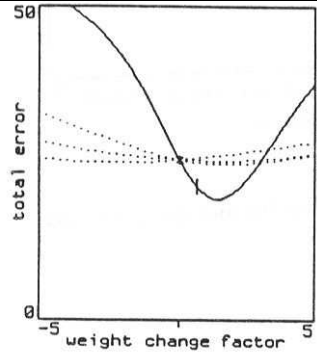
epoch 50



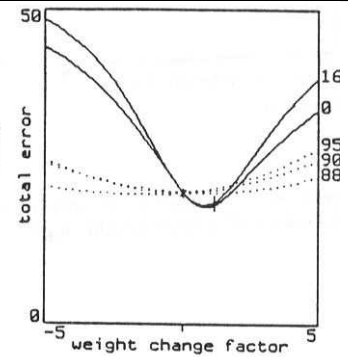
epoch 75



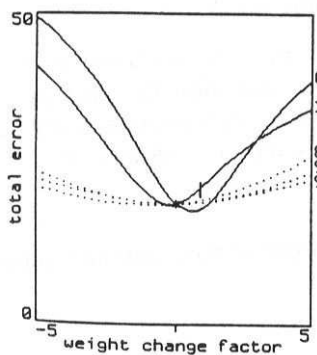
epoch 107 (solution)



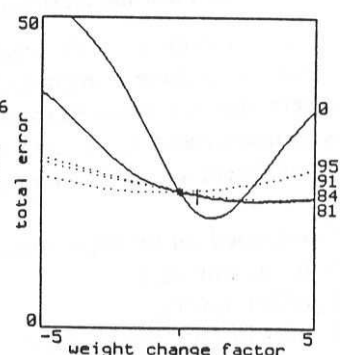
epoch 1



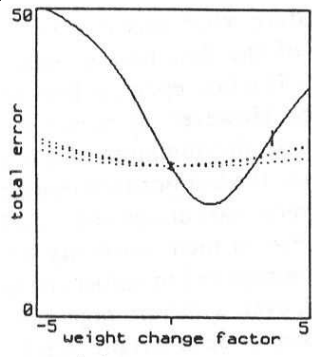
epoch 2



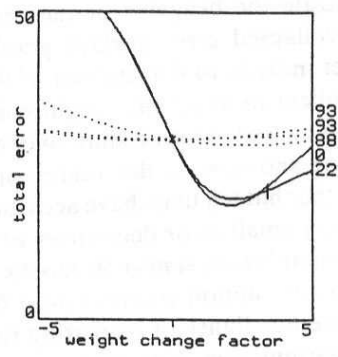
epoch 3



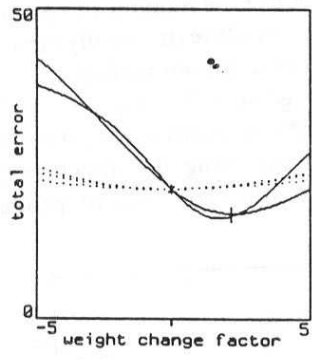
epoch 4



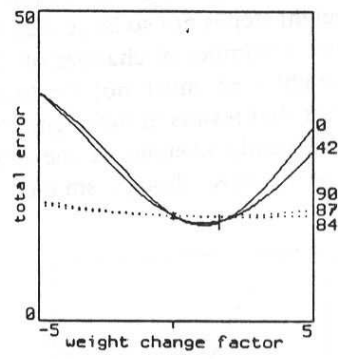
epoch 1



epoch 2



epoch 3



epoch 4