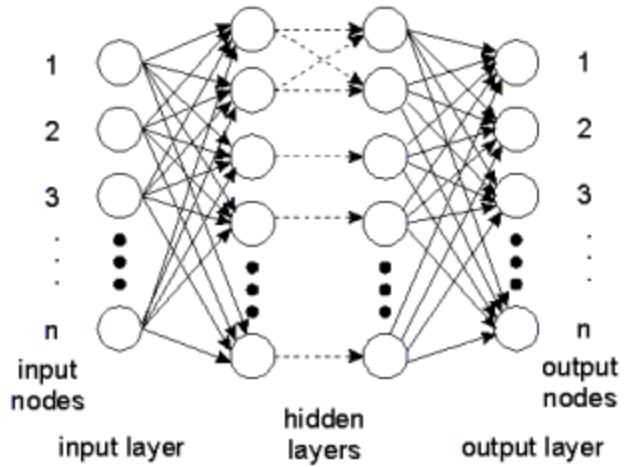
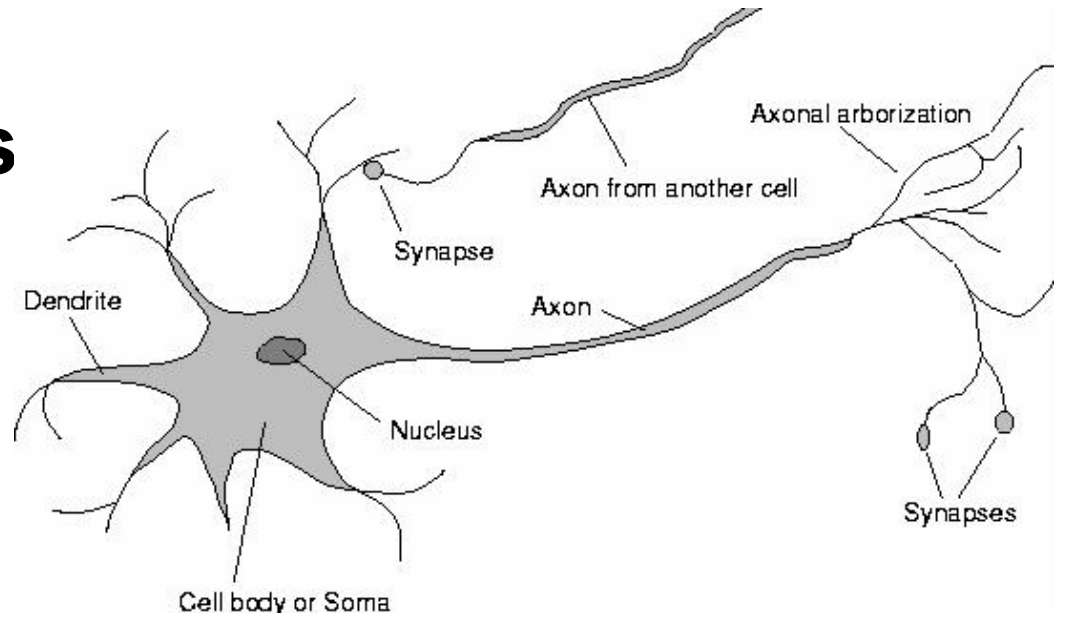


Neural Networks

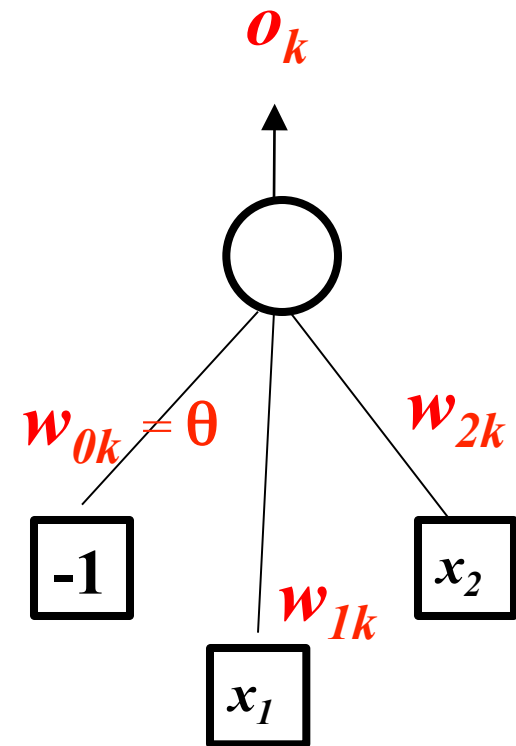


Notation: each cell j has

- a current output or *activation* level, o_j
- a target or desired output level, t_j
- links to other cells k of weight w_{jk}
- links from other cells i of weight w_{ij}
- an activation function $g()$
- a threshold or bias θ at which it fires
 - treat as constant input of -1 connected by weight θ

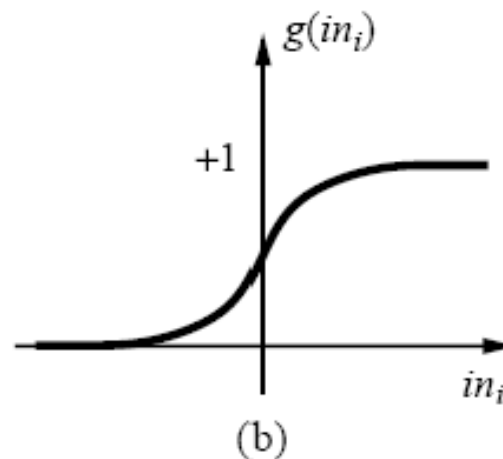
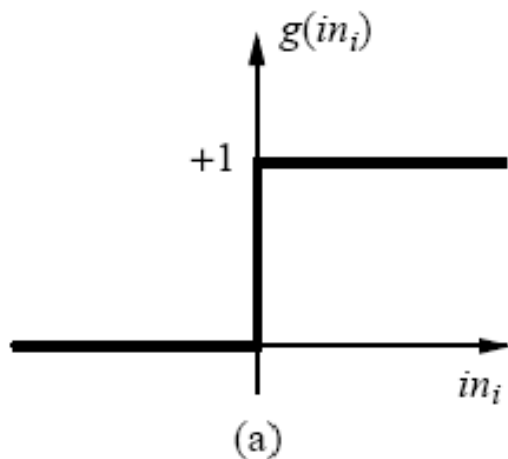
Input and Output

- each cell's output o_k is determined by:
 - $o_k = g(\text{net}_k)$
- where the net input net_k to cell k is summed:
 - $\text{net}_k = \sum_j w_{jk} o_j$

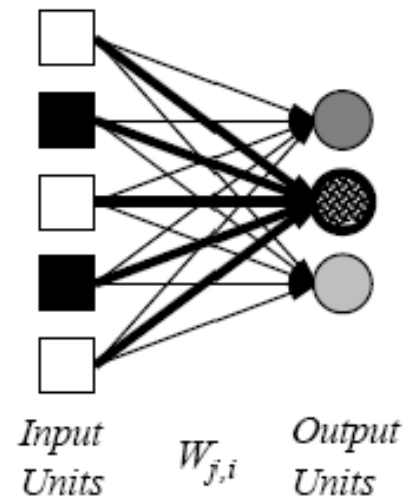


Activation Functions

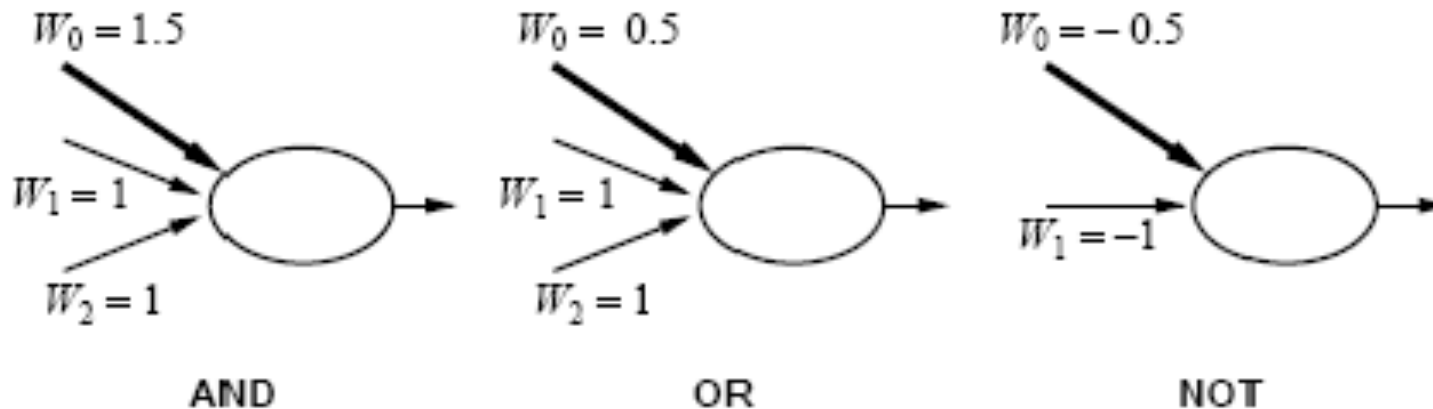
- typically non-linear, monotonic (or discontinuous at one point)
- common options:
 - $\text{step}()$, $\text{sgn}()$
 - $\text{tanh}()$
 - $\text{sigmoid}(x) = 1 / (1 + e^{-x})$



Perceptrons: Single layer feed-forward networks



- response given by: $o = \sum_j w_j x_j$



Perceptron (or “Delta Rule”) Learning

[Widrow and Hoff; Rosenblatt 1960]

- recall

$$o = \sum_j w_j x_j$$

- LMS expresses error as sum of squared errors:

$$E = \frac{1}{2} \text{Err}^2 = \sum \frac{1}{2} (t - o)^2$$

- calculate partial of Error with respect to each weight:

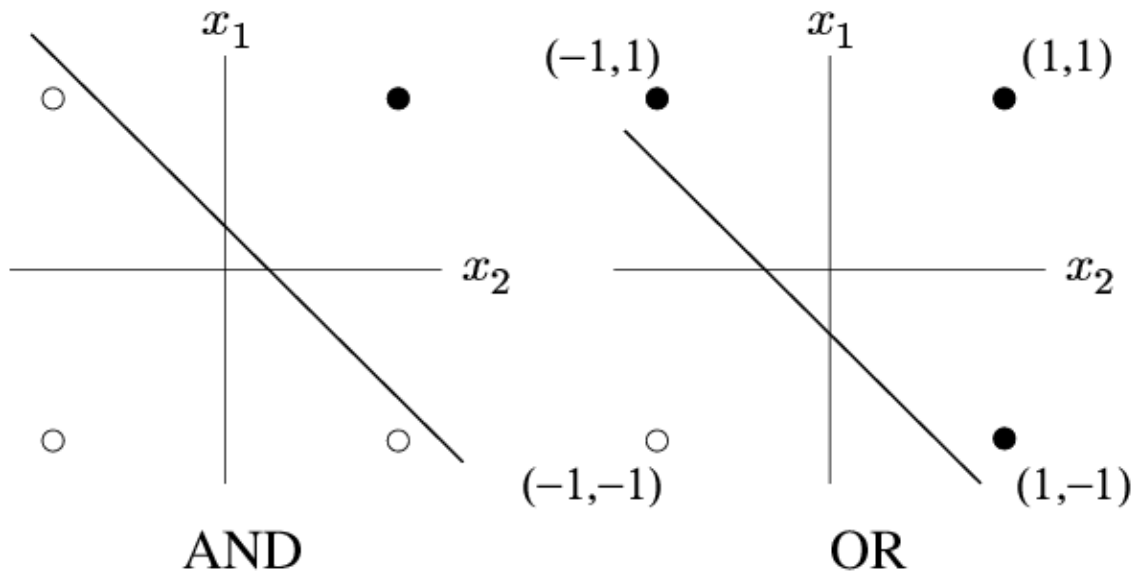
$$\frac{\partial E}{\partial w_j} = (\frac{\partial E}{\partial o})(\frac{\partial o}{\partial w_j}) = -(t - o)x_j$$

- now modify weights along negative of error gradient, for some learning rate, α :

$$\Delta w_j = \alpha (t - o)x_j$$

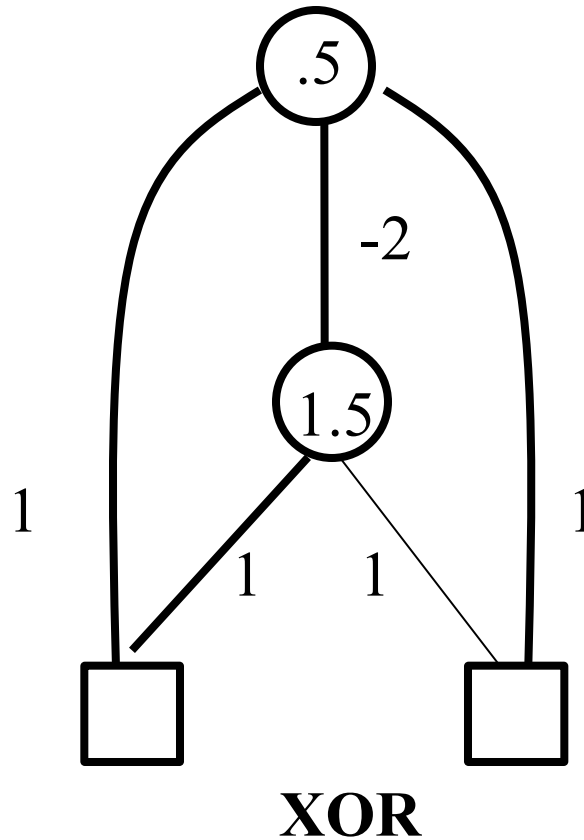
Linear non-separability problem

Minsky and Papert [1969]



Two-layer feedforward nets

- multilayer perceptron can solve XOR



Minsky and Papert's Criticism

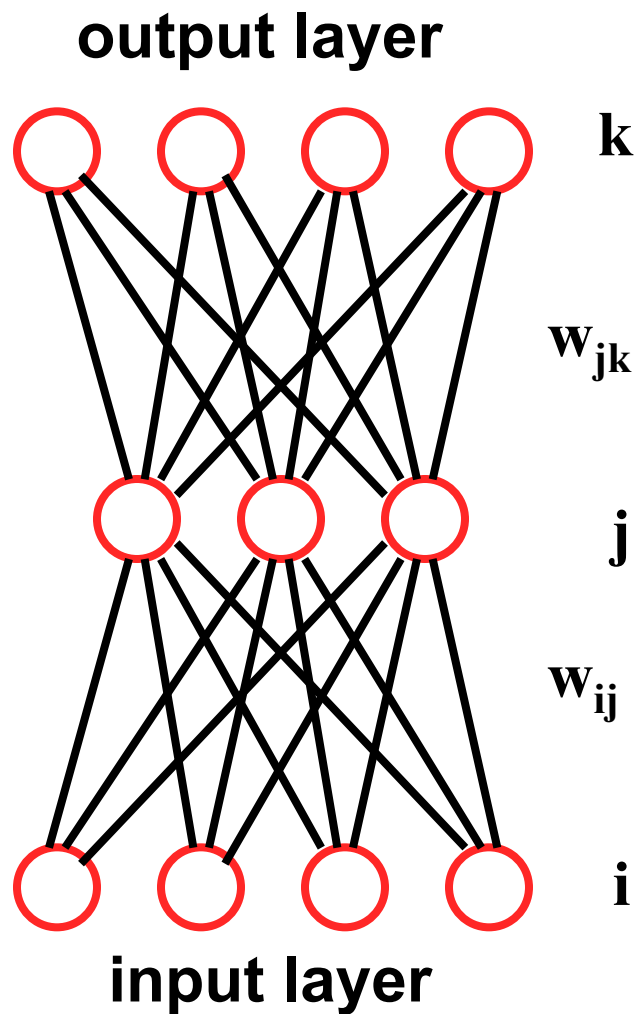
- no solution for how to *learn* weights for hidden units:
 - how to give credit (or blame) to the hidden layer units for their contribution to the error?

Backpropagation

[Werbos 1974; Rumelhart, Hinton & Williams 1986, etc.]

- recursive method for weight adjustment in multilayer feed-forward networks
- based on generalization of the delta rule for non-linear activation functions
- assigns error value to output in hidden layers

Network Architecture



How does weight w_{jk} to output k affect error?

$$E = \frac{1}{2} \sum_k (t_k - o_k)^2$$

$$\begin{aligned} \partial E / \partial w_{jk} &= (\partial E / \partial o_k) (\partial o_k / \partial w_{jk}) \\ &= -(t_k - o_k) \partial o_k / \partial w_{jk} = -(t_k - o_k) \partial g(\text{net}_k) / \partial w_{jk} \\ &= -(t_k - o_k) g'(\text{net}_k) \partial \text{net}_k / \partial w_{jk} \\ &= -(t_k - o_k) g'(\text{net}_k) \partial / \partial w_{jk} (\sum_j w_{jk} o_j) \\ &= -(t_k - o_k) g'(\text{net}_k) o_j \end{aligned}$$

define modified error measure $\delta_k = (t_k - o_k) g'(\text{net}_k)$

$$\partial E / \partial w_{jk} = -o_j \delta_k$$

Continuing...

recall: $\delta_k = (t_k - o_k)g'(net_k)$

where $g()$ is defined as: $g(x) = 1 / (1 + e^{-x})$

and so: $g'(x) = e^{-x} / (1 + e^{-x})^2$

$$= [1 / (1 + e^{-x})] [e^{-x} / (1 + e^{-x})]$$

$$= g(x) (1 - g(x))$$

which gives: $\delta_k = (t_k - o_k)g(net_k)(1 - g(net_k))$

$$= (t_k - o_k)o_k(1 - o_k)$$

How does weight w_{ij} (to hidden unit j) affect error?

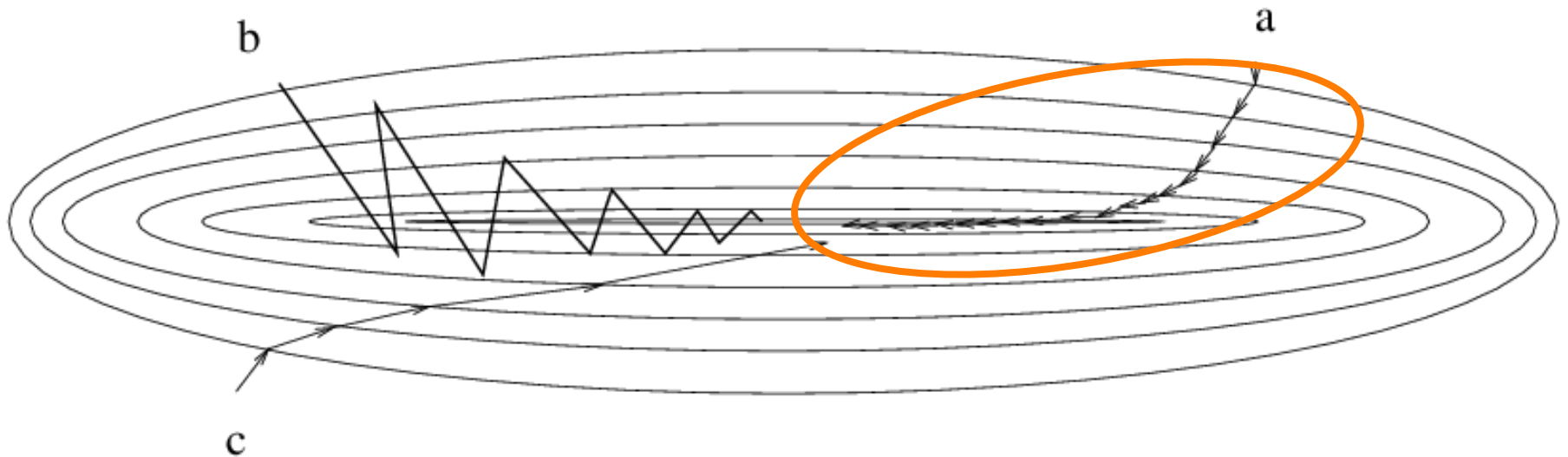
- compute recursively via Chain Rule

$$\begin{aligned}\frac{\partial E}{\partial w_{ij}} &= \sum_k \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial net_k} \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} \\ &= \sum_k -(t_k - o_k) o_k (1 - o_k) w_{jk} o_j (1 - o_j) o_i \\ &= \sum_k -\delta_k w_{jk} o_j (1 - o_j) o_i \\ &= -o_i o_j (1 - o_j) \sum_k \delta_k w_{jk}\end{aligned}$$

Backprop Training Issues

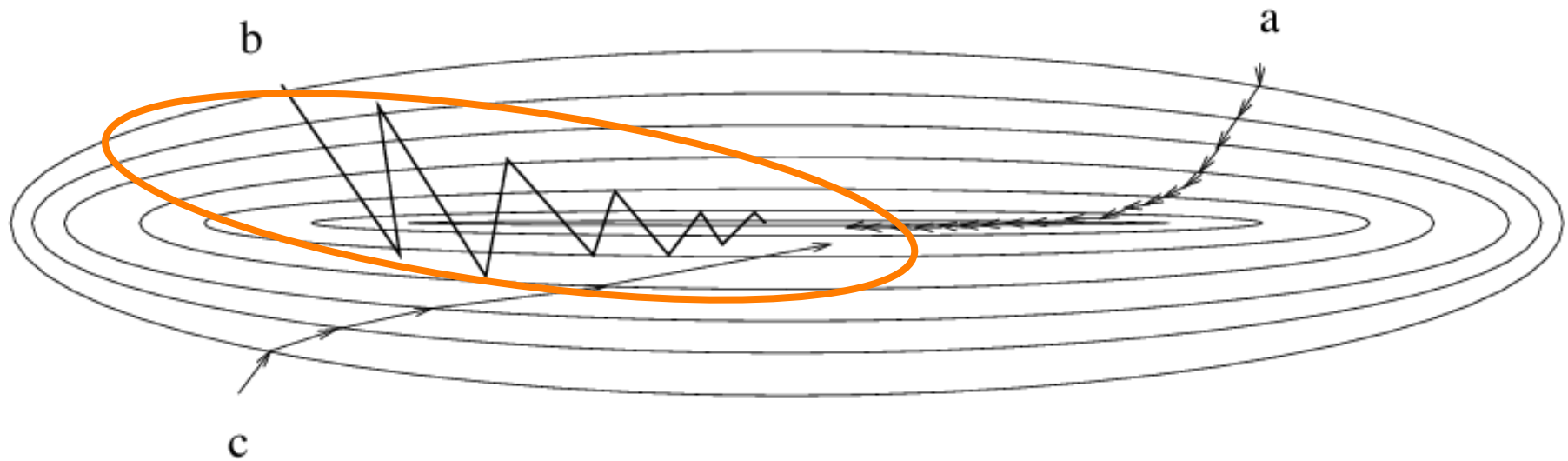
- sigmoid is asymptotic at 0 and 1
 - $g(\infty) = 1, g(-\infty) = 0$
 - $net_k = \sum_j w_{jk} o_j$
 - since o_j is determined for us, we can only vary w_{jk} ; hence, leading to infinite weights

Gradient Descent Too Slow



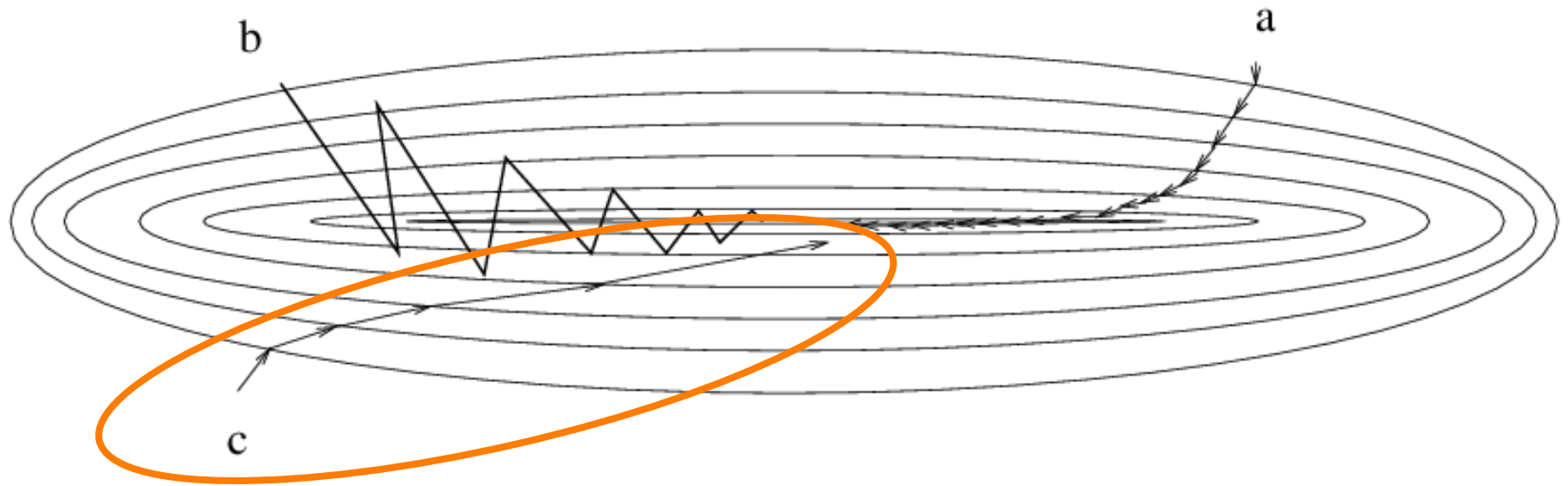
- gradient descent needs infinitesimal steps
=> this could take forever

Gradient Descent Too Fast



- in practice, use a larger step size -- *learning rate*: α
=> too large a value leads to oscillations in error space

Gradient Descent just right



- use a momentum factor γ
 - e.g., for output unit: $\Delta w(t + 1) = \alpha o_j \delta_k + \gamma \Delta w(t)$

Local Minima

- non-linear functions
- error surface is full of hills and valleys
- gradient descent can get trapped
- solutions?

Overlearning

- don't want network to be overtrained
- partition the data examples into three sets:

Data Set	Used for	Analogy with
Training	Learning; determines the weights	Example problems solved in class
Validation	Tuning; e.g., choose number of hidden units in the network or when to stop training	Questions on final exam
Test set	Assess performance (generalization) of the selected network	Problems solved later in professional life

When should we use NN's?

- where complex mapping needs to be found for which a closed-form analytic solution is not readily available or computationally efficient

Backprop Applications

NOVICE



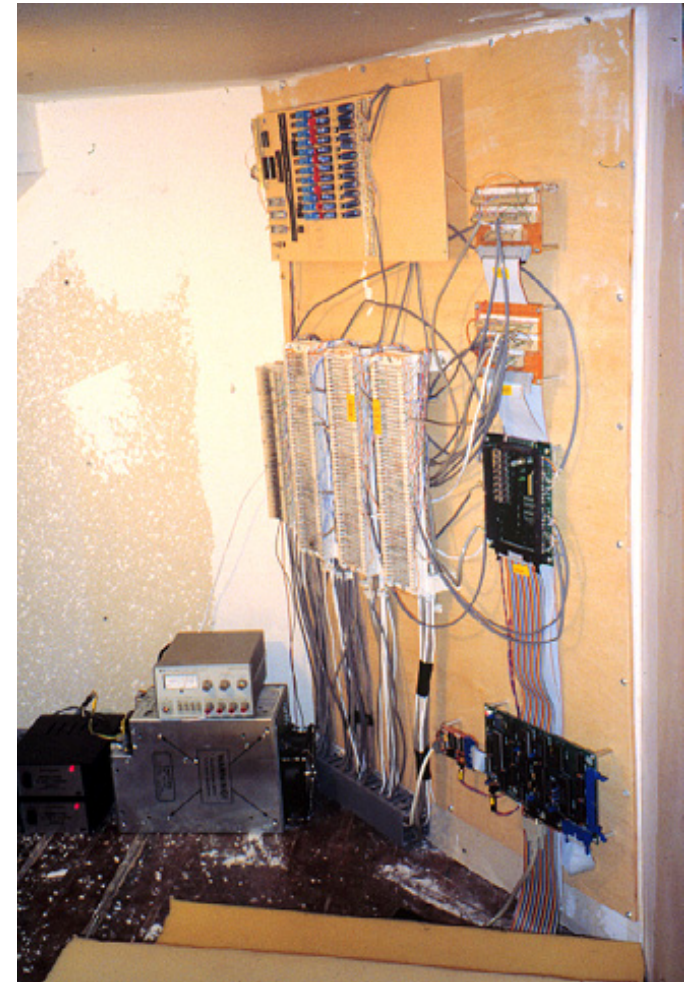
- multiple networks used for different activities (e.g. base steering, approaching, reaching)
- solves inverse perspective projection and inverse kinematic mappings
- adapts on-line to changes in configuration

Character Recognition

- OCHRE
 - Java demo
- LeNet - US Postal Code recognition
 - [asamples acomp anoise 12-4-21]

Mozer's Adaptive House

... a home that essentially **PROGRAMS ITSELF** by observing the actions of the inhabitants



ALVINN - Driving a Car

- Gaussian output array PDF
 - overcomes problem of severe non-linearities
- training “on the fly”
 - management of training data to avoid overlearning
 - artificial synthesis of unseen training data



Speech Synthesis

Traditional Approach

- 1. create *if-then* rules to encode all regularities
- 2. maintain database of exceptions to rules
- 3. build a production system to resolve conflicts
- e.g. 'c' can either be pronounced 's' as in **center**, **icy**, and **city** or a 'k' as in **cat** and **crumb**
 - if a 'c' appears before an i, e, or y
 - then pronounce it like an 's'
 - else pronounce it like a 'k'
 - exception: **celtic**

Networks approach

Learn to pronounce words through many examples

- choose an architecture
- choose a representation for the input and output
- determine a training set
 - random pick of 16,000 of 20,000 words from Webster's
- train
 - give each word in training set and indicate errors
- test
 - correct response to 90% of 4,000 remaining words

NETTalk

- building a production system to generate appropriate phonemes corresponding to written text is a very difficult problem
- non-intuitive relationships can be discovered by network whereas they might not be recognizable by human programmer

GloveTalk

- translates hand gestures to speech through an adaptive interface
- gestures are mapped continuously to ten control parameters of a parallel formant speech synthesizer
- allows the hand to act as an artificial vocal tract that produces speech in real time
- one subject trained to speak intelligibly with Glove-TalkII