

Lecture 23:

Artificial neural networks

- Broad field that has developed over the past 20 to 30 years
- Confluence of statistical mechanics, applied math, biology and computers
- Original motivation: mathematical modeling of neurological networks
- Practical applications: pattern recognition (e.g. applied to speech, handwriting, underwriting)
 - used by USPS to read handwritten zip codes
 - Can be very fast, particularly when implemented in special purpose hardware

Biological neurons

Schematic structure of a neuron
(Cichocki and Unbehauen reproduced
from *Principles of Neurocomputing for
Science & Engineering* by Ham and
Kostanic)

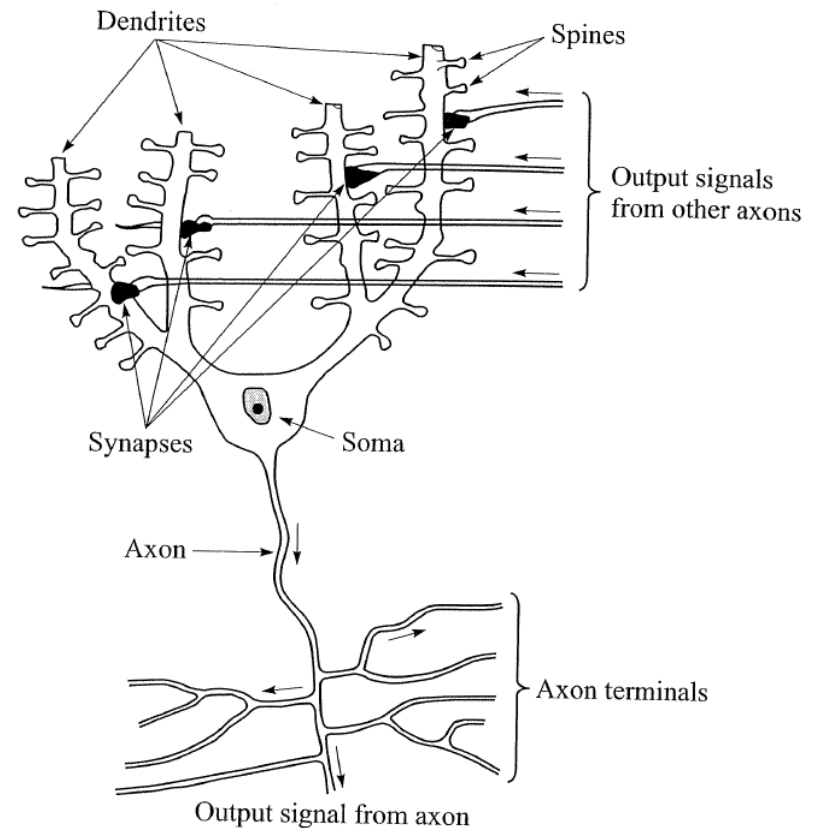
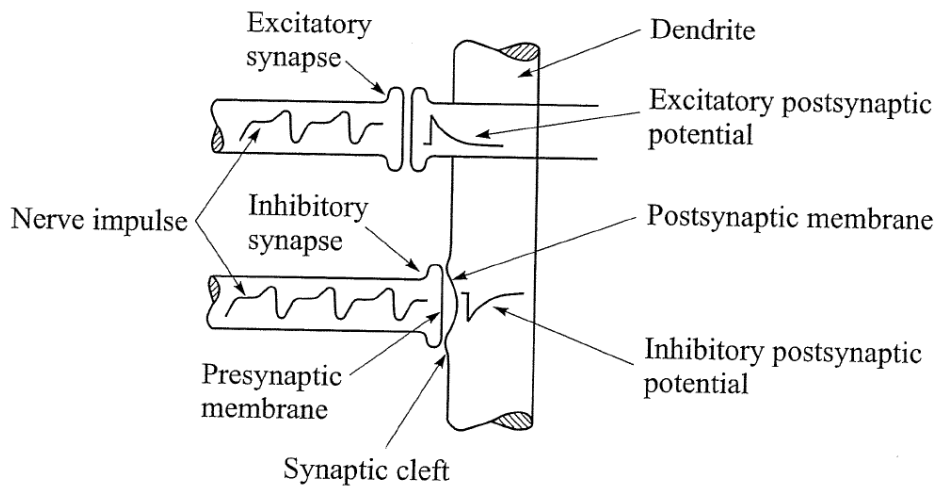
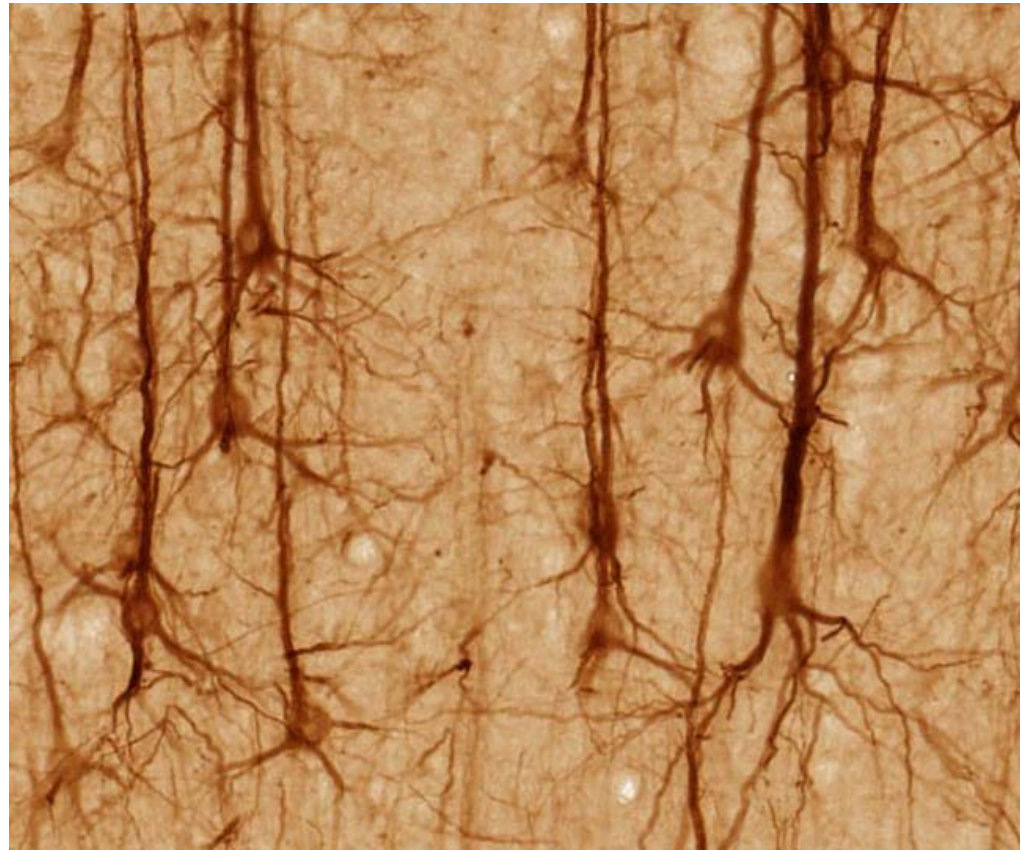


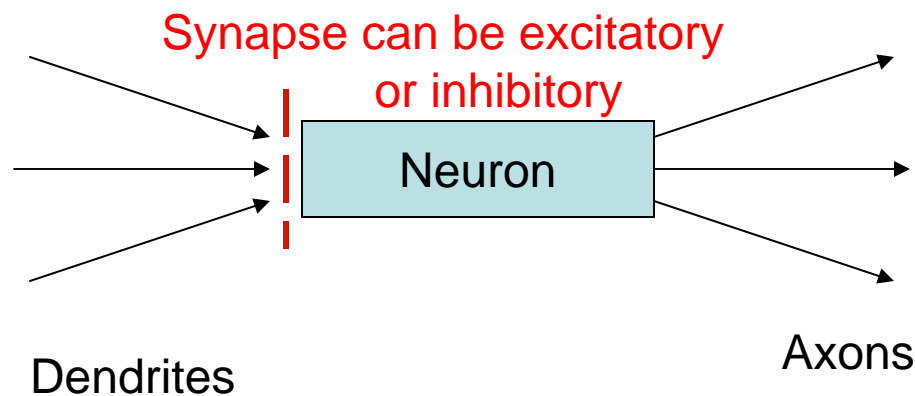
Image of primate neurons



From brain-maps.org. Human brain contains $\sim 10^{11}$ neurons

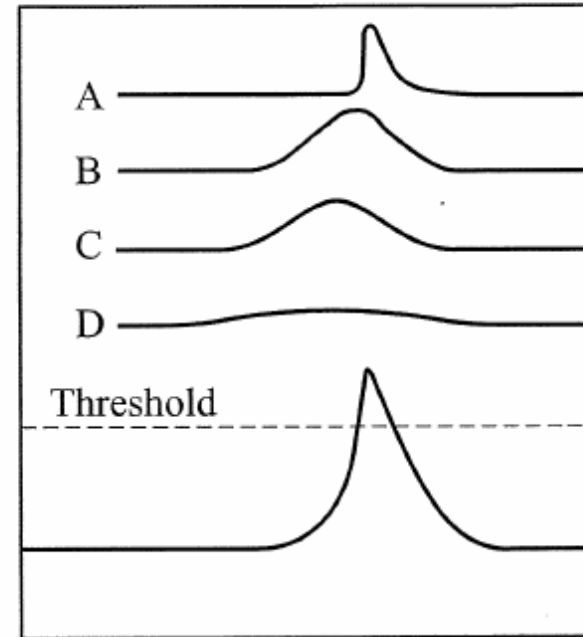
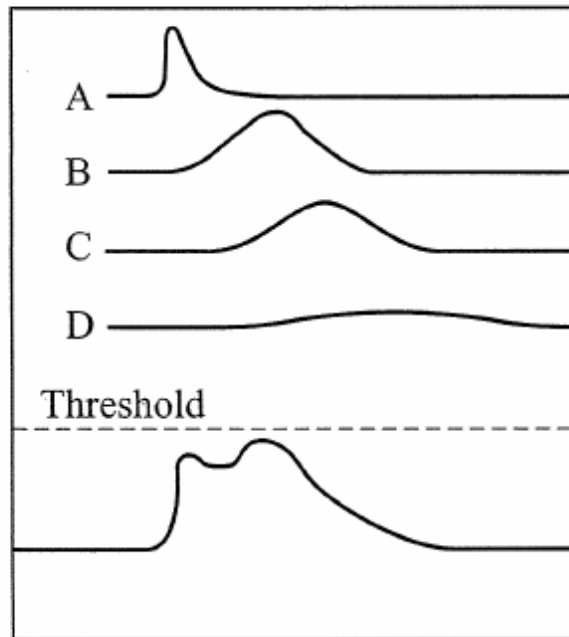
Biological neurons

- Input signals come from the axons of other neurons, which connect to dendrites (input terminals) at the synapses
- If a sufficient excitatory signal is received, the neuron fires and sends an output signal along the axons



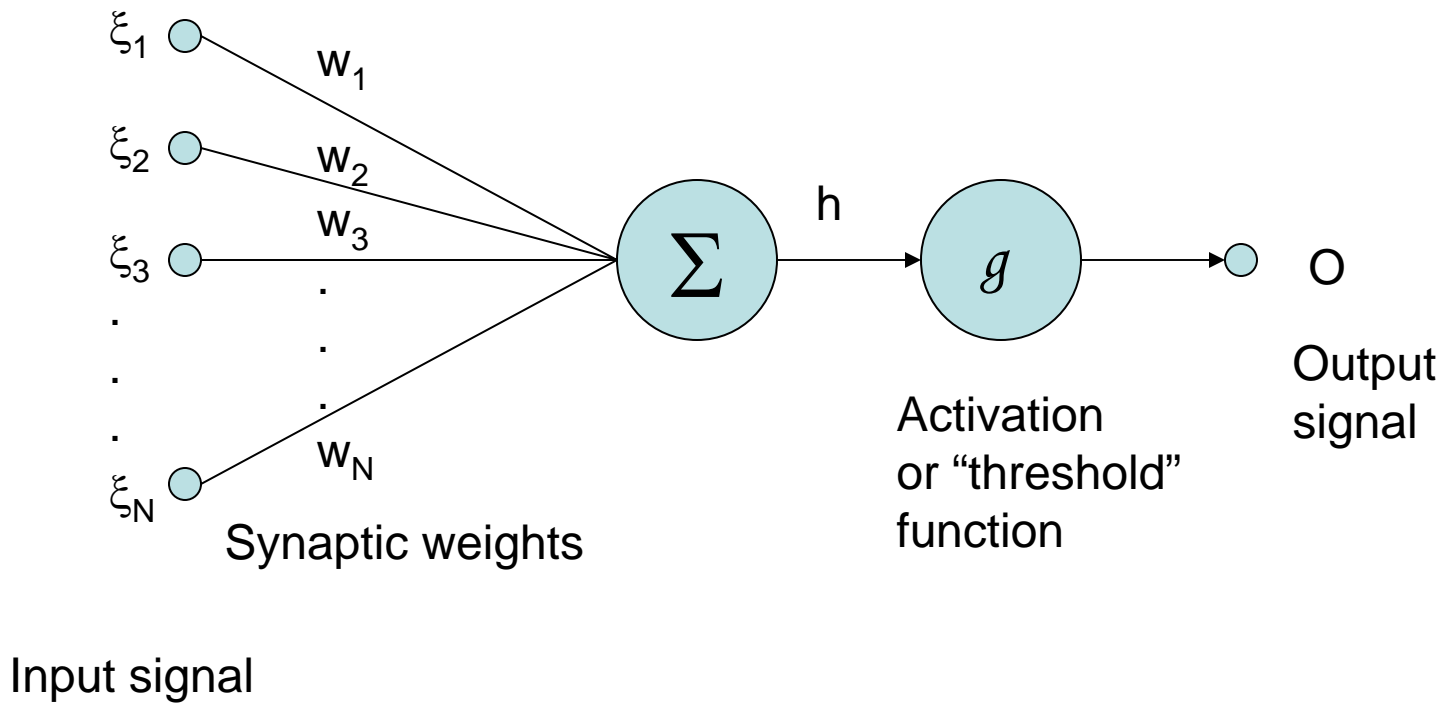
Threshold effect

- The firing of the neuron occurs when a threshold excitation is reached



Mathematical model

- Nonlinear model of an artificial neuron



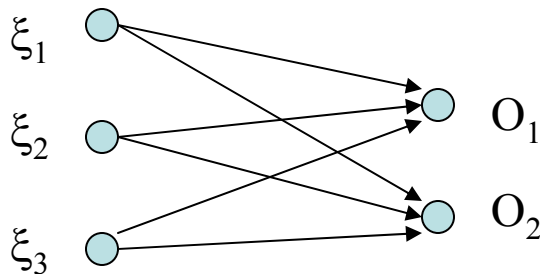
Mathematical model

- Input and output signals are normalized, typically over the range $[-1, +1]$ or $[0, +1]$
- Activation function can be
 - linear: $g(h) = h$
 - step-like: $g(h) = \text{sgn}(h)$
 - sigmoid: $g(h) = \tanh(\beta h)$ or $1/(1+e^{-2\beta h})$
- Neuron output, $y = g(\sum(w_i x_i))$

Network architecture

- Here, we confine our attention to feed-forward networks (a.k.a. “perceptrons”) → no feedback loops: transfer of information is unidirectional

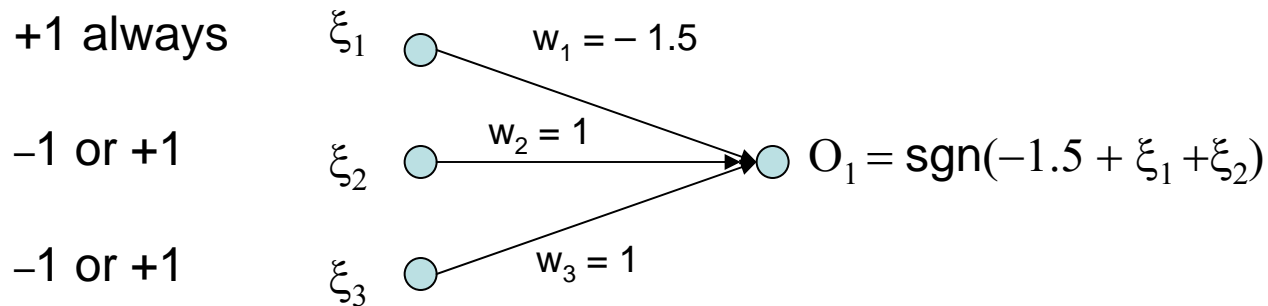
Simplest example: 1-layer perceptron with N inputs (ξ_k with $k = 1, N$) connected to M outputs (O_i with $i = 1, M$) via MN synaptic weights W_{ik}



$$O_i = g \left(\sum_{k=1, N} w_{ik} \xi_k \right)$$

Example: the logical AND function

- Represent by a 1 x 3 perceptron



(example of setting thresholds with hardwired inputs)

Training the network

- How can we teach the network to yield desired responses?

- Need a set of desired inputs and outputs:

- ξ_j^μ and ζ_j^μ , for $\mu=1, p$

- Need a measure of learning: the “cost function”, $E(\mathbf{w})$, that we seek to minimize

Matrix of synaptic weights, W_{ik}

- Finding the best set of weights is then an MN dimensional minimization problem

The cost function, $E(\mathbf{w})$

2 common choices

$$1) \quad E_{MSE}(\mathbf{w}) = \frac{1}{2} \sum_i \sum_{\mu} (\zeta_i^{\mu} - O_i^{\mu})^2 = \frac{1}{2} \sum_i \sum_{\mu} \left(\zeta_i^{\mu} - g \left(\sum_k w_{ik} \xi_k^{\mu} \right) \right)^2$$

(mean square error)

$$2) \quad E_{RE}(\mathbf{w}) = \frac{1}{2} \sum_i \sum_{\mu} \left((1 + \zeta_i^{\mu}) \ln \frac{(1 + \zeta_i^{\mu})}{(1 + O_i^{\mu})} + (1 - \zeta_i^{\mu}) \ln \frac{(1 - \zeta_i^{\mu})}{(1 - O_i^{\mu})} \right)$$

(relative entropy, for specific case of the tanh activation function)

Minimizing $E(\mathbf{w})$

- Simple training method: start with initial guess of weights and change in accord with

$$\Delta w_{ik} = -\eta \frac{\partial E}{\partial w_{ik}}$$

“Learning rate” ~ 0 to 1

(Move along direction of steepest descent)

Minimizing $E(\mathbf{w})$

- The derivatives are easy to compute

$$\frac{\partial E_{MSE}}{\partial w_{ik}} = - \sum_{\mu} (\zeta_i^{\mu} - o_i^{\mu}) g' \left(\sum_k w_{ik} \xi_k^{\mu} \right) \xi_k^{\mu}$$

which is conveniently written as $\frac{\partial E_{MSE}}{\partial w_{ik}} = - \sum_{\mu} \delta_i^{\mu} \xi_k^{\mu}$

with
$$\delta_i^{\mu} = (\zeta_i^{\mu} - o_i^{\mu}) g' \left(\sum_k w_{ik} \xi_k^{\mu} \right)$$

and

$$\begin{aligned} g'(h) &= 2\beta g(h)[1 - g(h)] && \text{for } g = \tanh(\beta h) \\ g'(h) &= \beta[1 - g^2(h)] && \text{for } g = 1/(1 + e^{-2\beta h}) \end{aligned}$$

Minimizing $E(\mathbf{w})$

- For the RE cost function, we find that

$$\frac{\partial E_{RE}}{\partial w_{ik}} = -\sum_{\mu} \delta_i^{\mu} \xi_k^{\mu}$$

where, for the tanh activation function,

$$\delta_i^{\mu} = (\xi_i^{\mu} - O_i^{\mu})\beta$$

Minimizing $E(\mathbf{w})$

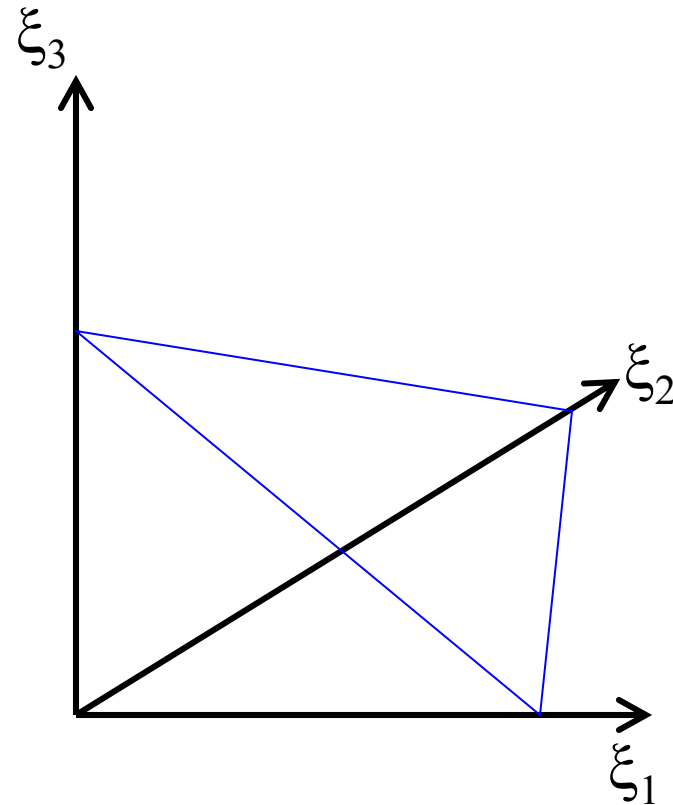
- So the procedure is to start with an initial set of starting weights and to change them iteratively according to

$$\Delta w_{ik}^{\mu} = \eta \delta_i^{\mu} \xi_k^{\mu}$$

until the cost function changes by less than some preset amount.

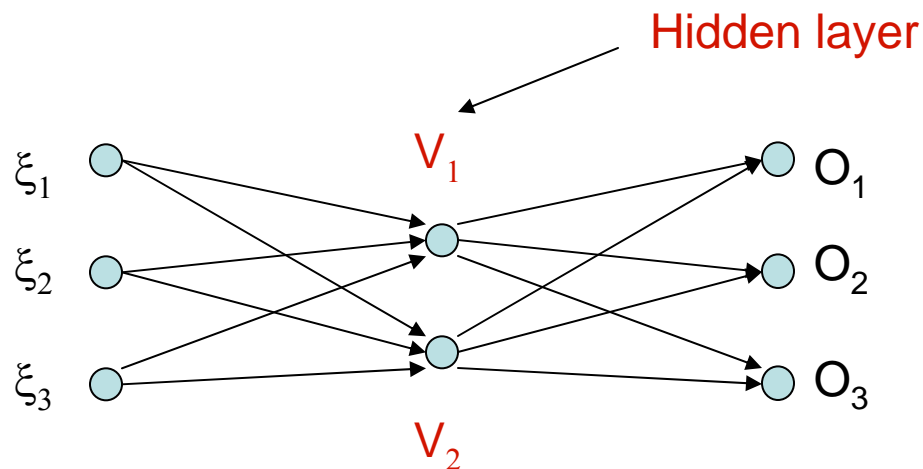
Geometric interpretation

- For each output node, there is an $N - 1$ dimensional hyperplane which separates input values yielding $O > 0$ from those with $O < 0$



Multi-layer networks

- Interest in feed-forward networks was limited until it was realized that 2-layer networks could describe any continuous function of the inputs



and a three-layer network can describe any function of the inputs

Multi-layer networks

- Obviously, if the activation function is linear, the two-layer network is equivalent to a one-layer network with synaptic weights $q_{ik} = \sum_j w_{jk} W_{ij}$
- But for the sigmoid or step-function (sgn) activation function, interesting new behavior can result

Training multilayer perceptrons

- As before we minimize the cost function, e.g.

$$E_{MSE}(\mathbf{W}, \mathbf{w}) = \frac{1}{2} \sum_i \sum_{\mu} (\zeta_i^{\mu} - O_i^{\mu})^2$$

but now we have to vary two sets of weights.

1) The derivatives w.r.t. W_{ik} are

$$\frac{\partial E_{MSE}}{\partial W_{ik}} = - \sum_{\mu} (\zeta_i^{\mu} - O_i^{\mu}) g' \left(\sum_k W_{ik} V_k^{\mu} \right) V_k^{\mu} = - \sum_{\mu} \delta_i^{\mu} V_k^{\mu}$$

where

$$\delta_i^{\mu} \equiv (\zeta_i^{\mu} - O_i^{\mu}) g'(H_i^{\mu}) \quad \text{and} \quad H_k^{\mu} \equiv \sum_k W_{ik} V_k^{\mu}$$

Training multilayer perceptrons

2) The derivatives w.r.t. w_{ik} are computed using the chain rule

$$\begin{aligned}\frac{\partial E_{MSE}}{\partial w_{ik}} &= \sum_{\mu} \frac{\partial E}{\partial V_j^{\mu}} \frac{\partial V_j^{\mu}}{\partial w_{ik}} \\ &= -\sum_i \sum_{\mu} (\xi_i^{\mu} - O_i^{\mu}) g'(H_i^{\mu}) W_{ij} g'(h_j^{\mu}) \xi_j^{\mu}\end{aligned}$$

For each layer, we update the weights by moving along the direction of steepest descent:

$$\Delta W_{ik} = -\eta \frac{\partial E}{\partial W_{ik}} \qquad \Delta w_{ik} = -\eta \frac{\partial E}{\partial w_{ik}}$$

Training multilayer perceptrons

- So the training procedure is
 1. Initialize all weights to random values
 2. Propagate input signal **forwards** through network to compute the intermediate and output signals
 3. Compute the cost function and its derivatives w.r.t. each weight, starting with the final layer and working **backwards**
 4. Update all weights
 5. Return to 2 (or stop if the convergence criterion is met)

Improvements in minimization

- As we know from previous lectures, the method of steepest descent can be very slow. We can use conjugate gradient or variable metric methods
- Alternatively, we can add a “momentum term” so that we include some of the previous step

$$\Delta w_{ik}|_{\text{new}} = -\eta \frac{\partial E}{\partial w_{ik}} + \alpha \Delta w_{ik}|_{\text{previous}}$$

where α is the momentum parameter (typically ~ 0.9 and must be between 0 and 1)

This can smooth the approach to the minimum. The best algorithms vary α and η as the minimum E is approached

Applications of ANN

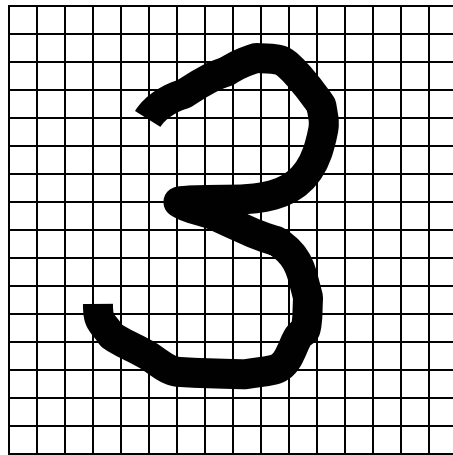
- **Underwriting**
 - Input: information about borrower/insured
 - Output: loan/insurance outcome
 - Training set: previous experience
- **Speech and handwriting recognition**
- **Financial predictions**
 - Attempts to “beat” the stock market not successful:
consistent with the “efficient market” hypothesis
- **Forecasting: weather, solar flares**
- **Diagnosis/classification: medical, astronomical**

Flexible networks

- While the number of input and output nodes is generally fixed by the problem, the number of hidden layers and nodes within them can be varied to reduce the cost function
- 3-layer perceptrons are always sufficient, although the use of more layers may reduce the required number of nodes

Example: Handwriting recognition

- Input: 16 x 16 pixel image: $N = 256$



$\xi = 1$ for pixels where
ink is present, -1
otherwise
 $\rightarrow 2^{256} \sim 10^{77}$
possible input states

- Output: 10 nodes: $O_k = 1$ if number is k and -1
otherwise

Example: Handwriting recognition

Feed forward neural net developed by Le Cun et al. with four hidden layers

Training set: 10^4 images digitized from addresses on actual US mail

28 x 28 *grayscale* pixels

Test set (not used for training): ~ 3000 additional images

4635 nodes, 98442 connections

Example: Handwriting recognition

Performance after 30 adaptation cycles:

1.1% error on training set

3.4% error on test set

Can achieve 1% error on test set if it rejects 5.7% of the characters

Example: photometric redshifts

- Use pattern recognition derive hard-to-measure parameter from observations of easy-to-measure parameter
- Sloan Digital Sky Survey provides broad-band photometric data (in 5 bands) for $\sim 10^8$ objects (mainly galaxies) and spectra for $\sim 10^6$

Spectra allow the redshift to be determined unequivocally, providing the distance and allowing the 3-D distribution to be determined

Example: photometric redshifts

- While only 1% of the objects are observed spectroscopically, “photometric redshifts” can be estimated for the other 99%
 - relative and absolute fluxes at 5 observed bands are correlated with z
- Collister and Lahav (2004, *PASP*, 116, 345) used artificial neural networks (e.g. 3-layer perceptron) to determine photometric redshifts: “ANNz” program

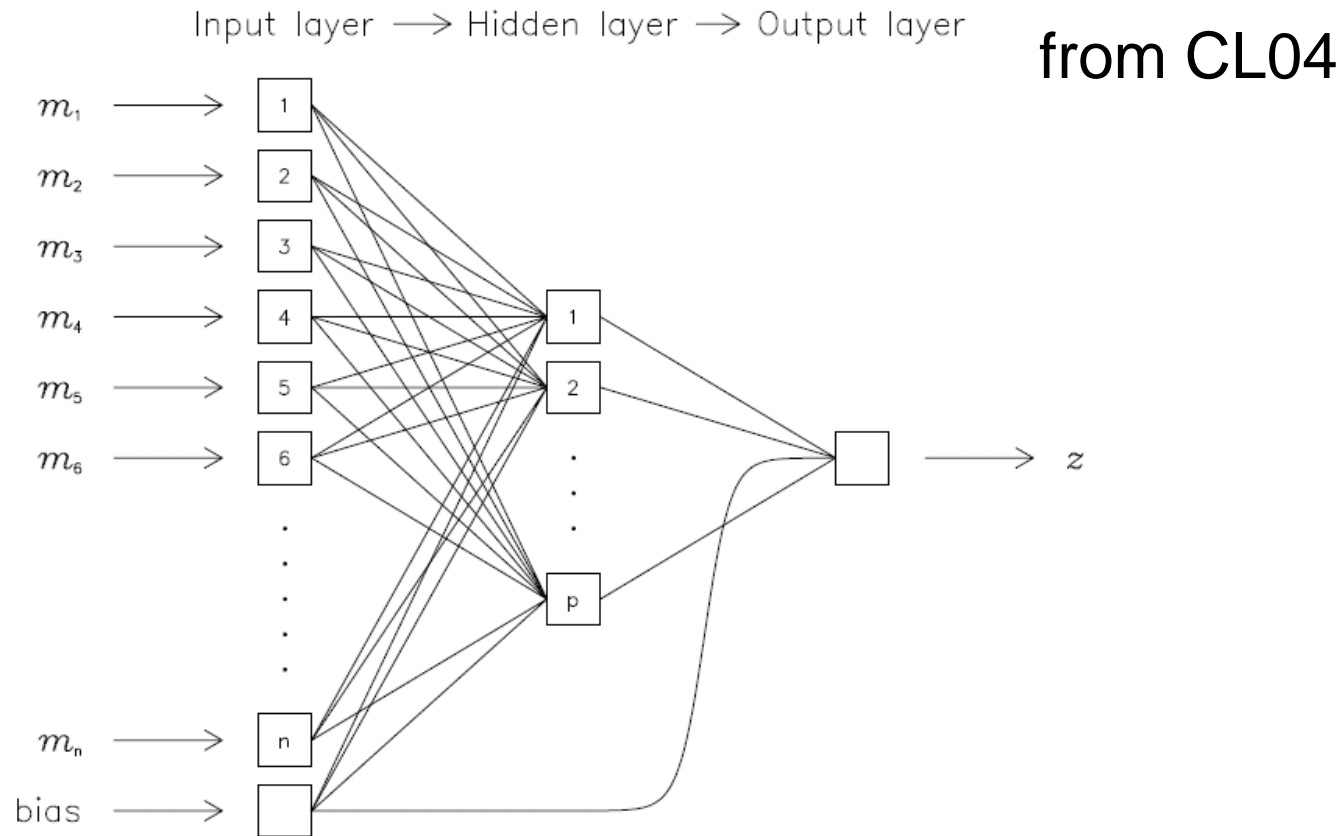


FIG. 1.— A schematic diagram of a multi-layer perceptron, as implemented by ANNz, with input nodes taking, for example, magnitudes $m_i = -2.5 \log_{10} f_i$ in various filters, a single hidden layer, and a single output node giving, for example, redshift z . The architecture is $n:p:1$ in the notation used in this paper. Each connecting line carries a weight w_{ij} . The bias node allows for an additive constant in the network function defined at each node. More complex networks can have additional hidden layers and/or outputs.

Example: photometric redshifts

- CL04 used a 3-layer perceptron with a 5:10:10:1 architecture
- Training set: 10^4 galaxies with spectroscopic redshifts
- Used “committee” of five independently trained networks
- Cost function modified to prevent blowup of weights

2.1. Network training

Given a suitable training set of galaxies for which we have both photometry, \mathbf{m} , and a spectroscopic redshift, z_{spec} , the ANN is trained by minimizing the *cost function*

$$E = \sum_k (z_{\text{phot}}(\mathbf{w}, \mathbf{m}_k) - z_{\text{spec},k})^2, \quad (2)$$

with respect to the weights, \mathbf{w} , where $z_{\text{phot}}(\mathbf{w}, \mathbf{m}_k)$ is the network output for the given input and weight vectors, and the sum is over the galaxies in the training set. To ensure that the weights are *regularized* (i.e. that they do not become too large), an extra quadratic cost term

$$E_w = \beta \sum_{i,j} w_{ij}^2, \quad (3)$$

is added to equation 2.

Example: photometric redshifts

- Results are quite impressive
- R.m.s. error in z is ~ 0.02 (versus 0.07 for competing method)

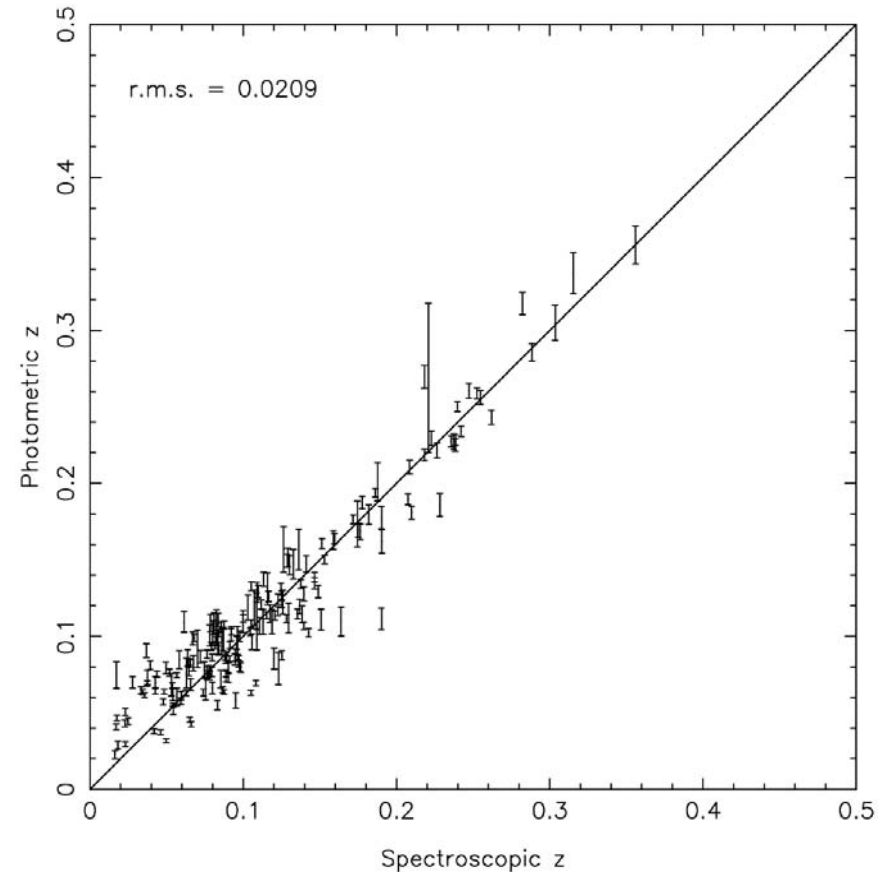


FIG. 3.— A subset of 200 galaxies randomly selected from the results of Fig. 2, and with the error bars calculated by ANNZ shown. These are a combination of contributions from photometric noise (§2.2) and network variance (§2.3).