

# Artificial Neural Networks and AI



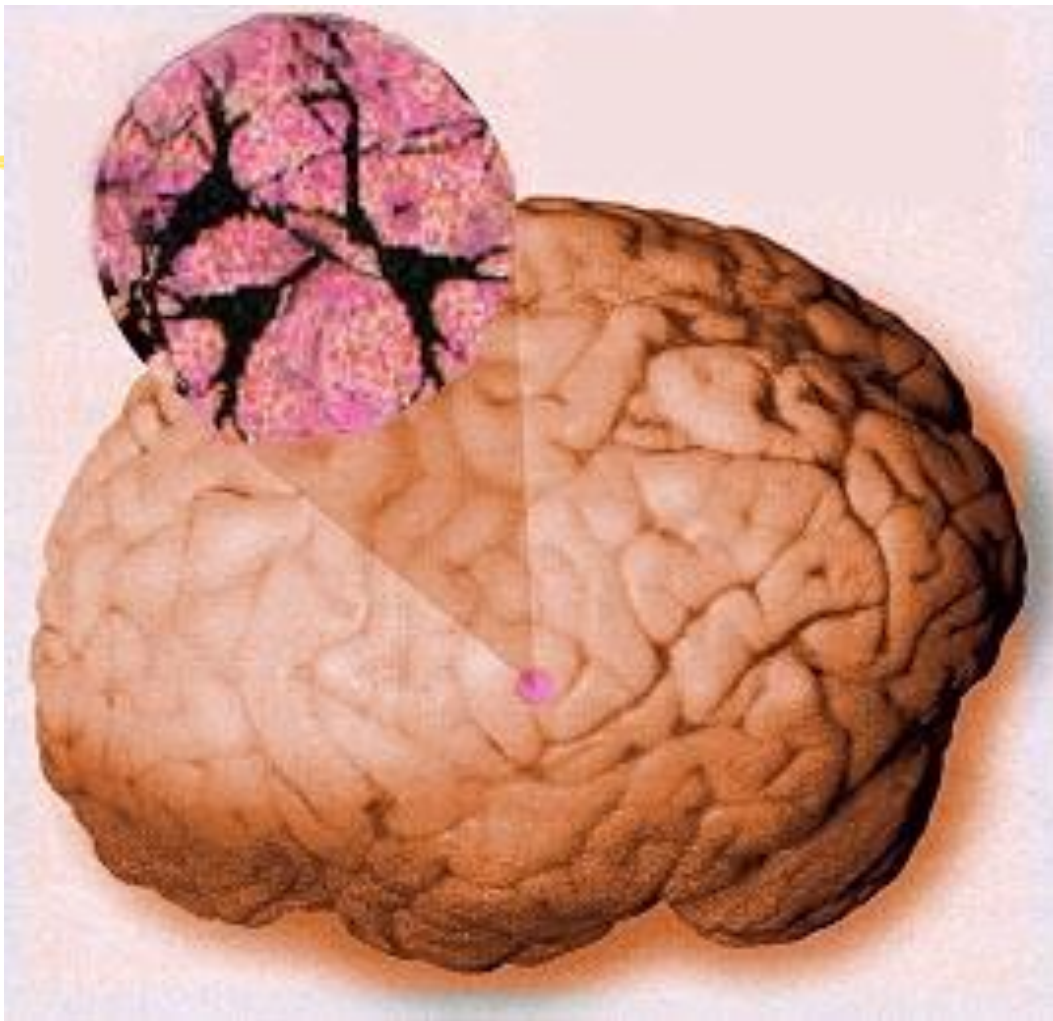
Artificial Neural Networks provide...

- A new computing paradigm
- A technique for developing trainable classifiers, memories, dimension-reducing mappings, etc
- A tool to study brain function

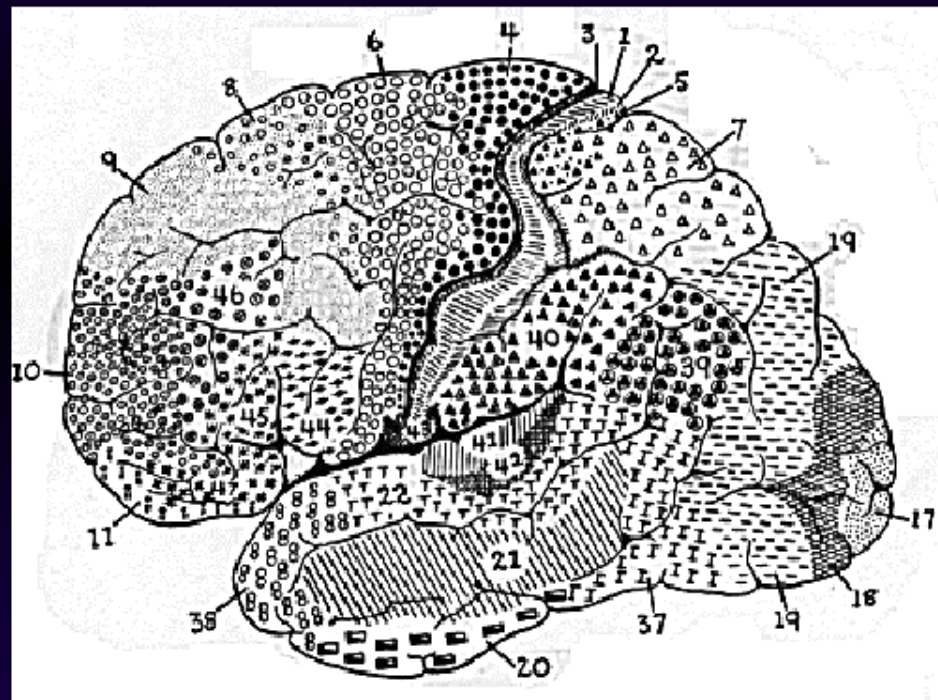
# Converging Frameworks



- **Artificial intelligence (AI):** build a “packet of intelligence” into a machine
- **Cognitive psychology:** explain human behavior by interacting processes (schemas) “in the head” but not localized in the brain
- **Brain Theory:** interactions of components of the brain -
  - computational neuroscience
  - neurologically constrained-models
- and abstracting from them as both **Artificial intelligence** and **Cognitive psychology:**
  - connectionism: networks of trainable “quasi-neurons” to provide “parallel distributed models” little constrained by neurophysiology
  - abstract (computer program or control system) information processing models



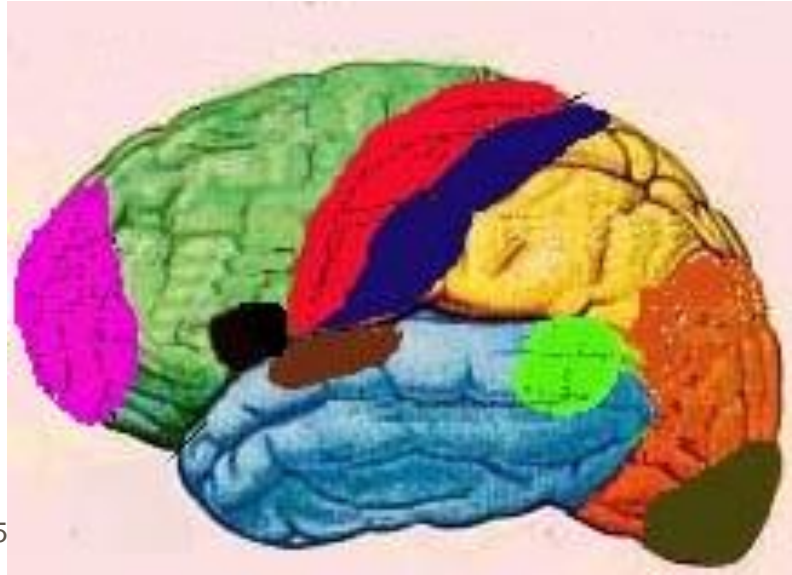
# Brodmann's cytoarchitectural map of Cortical Areas



Lateral View

# Major Functional Areas

- **Primary motor:** voluntary movement
- **Primary somatosensory:** tactile, pain, pressure, position, temp., mvt.
- **Motor association:** coordination of complex movements
- **Sensory association:** processing of multisensory information
- **Prefrontal:** planning, emotion, judgement
- **Speech center (Broca's area):** speech production and articulation
- **Wernicke's area:** comprehension of speech
- **Auditory:** hearing
- **Auditory association:** complex auditory processing
- **Visual:** low-level vision
- **Visual association:** higher-level vision





Felleman &amp; Van Essen, 1991

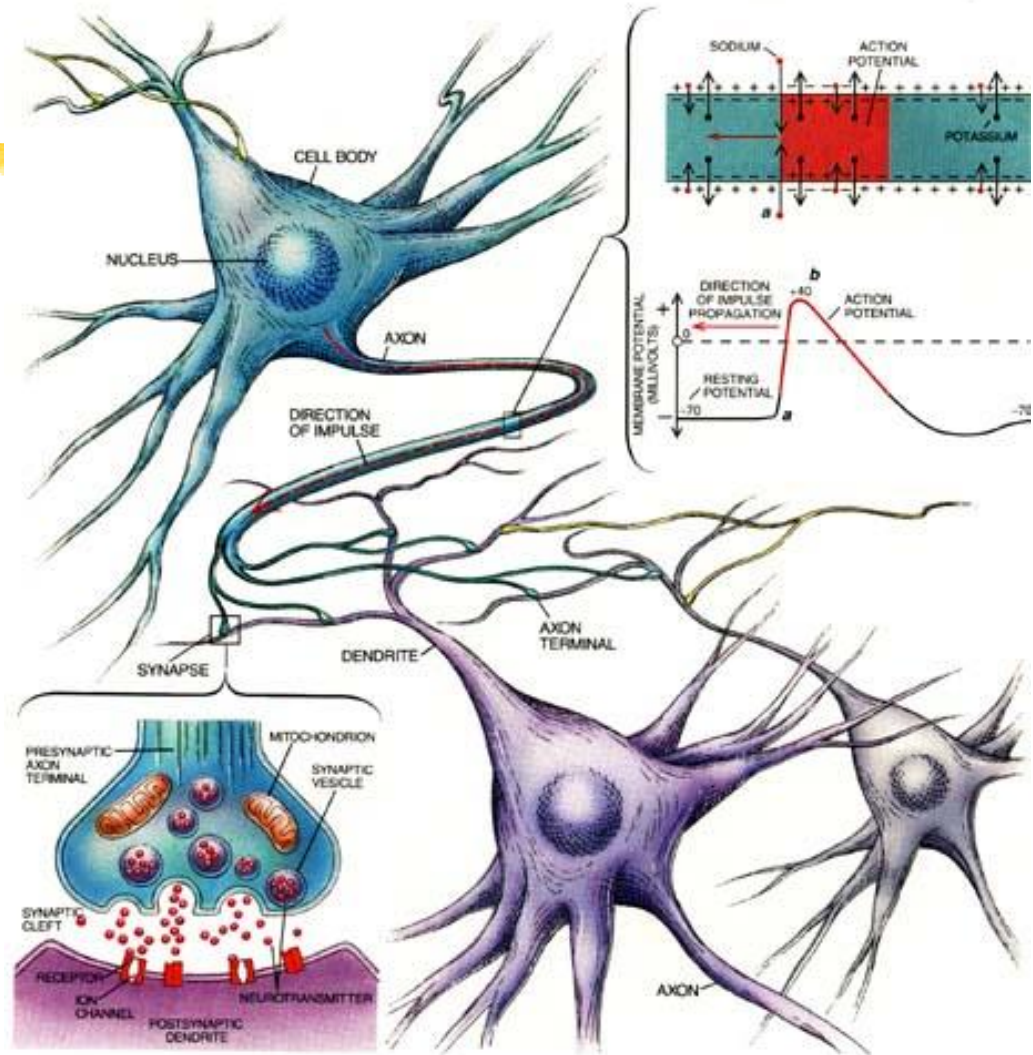


# Remember?

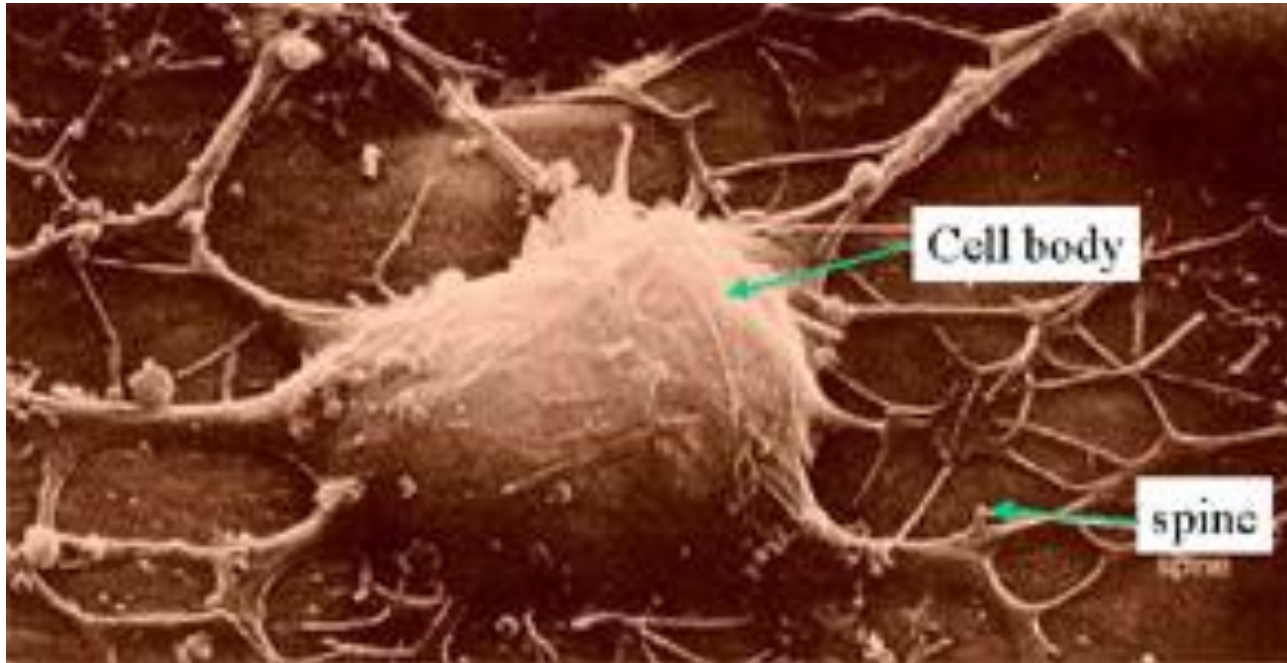
## Neurons & synapses

### Key terms:

Axon  
Dendrites  
Synapses  
Soma (cell body)



# Electron Micrograph of a Real Neuron





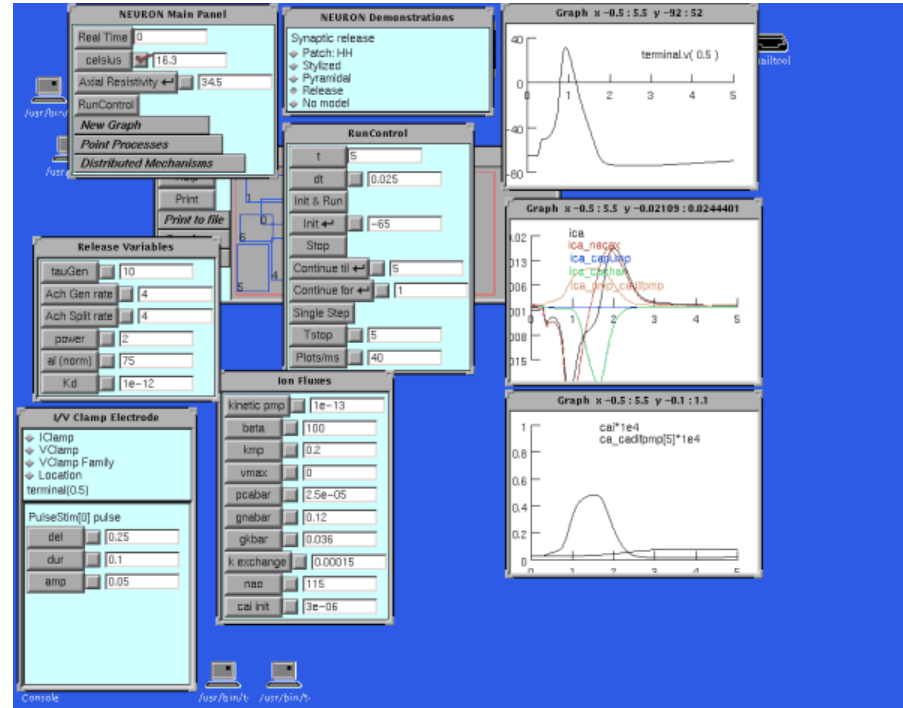
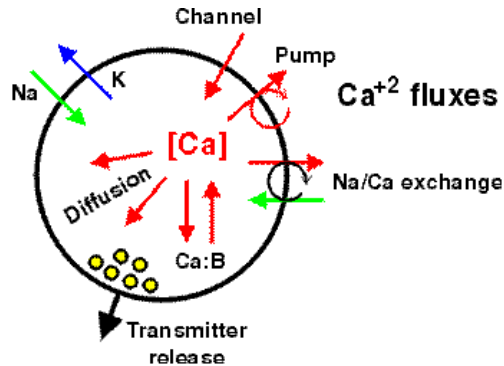
# Approaches to neural modeling



- Biologically-realistic, detailed models
  - E.g., cable equation, multi-compartment models
  - The Hodgkin-Huxley model
  - Simulators like NEURON (Yale) or GENESIS (Caltech)
- More abstract models, still keeping realism in mind
  - E.g., integrate & fire model, simple and low detail but preserves spiking behavior
- Highly abstract models, neurons as operators
  - E.g., McCulloch & Pitts model
  - Classical “neural nets” modeling

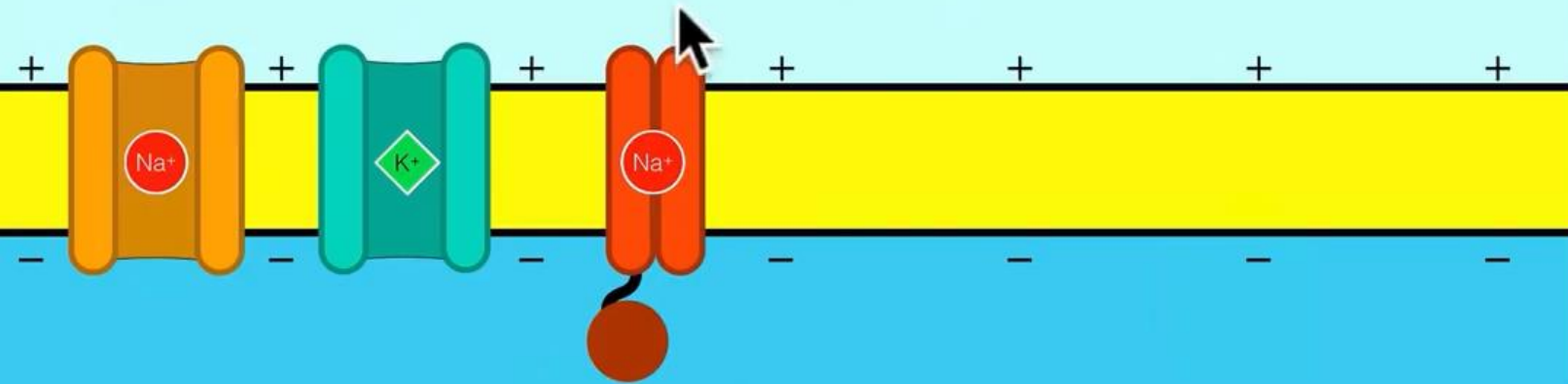
# Detailed Neural Modeling

A simulator, called "Neuron" has been developed at Yale to simulate the Hodgkin-Huxley equations, as well as other membranes/channels/etc. See <http://www.neuron.yale.edu/>



## Detailed Neural Modeling

- <https://www.youtube.com/watch?v=HYLyhXRp298>



Leak  
channels

Voltage-gated  
channels



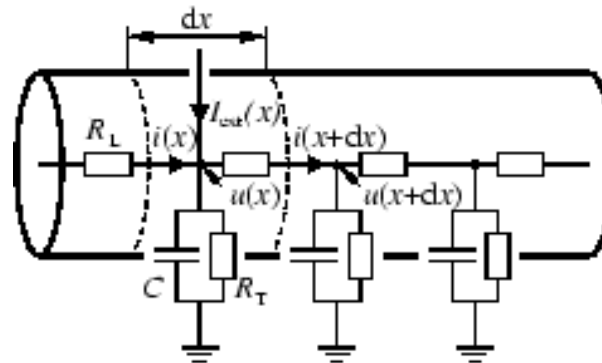
# The Cable Equation

- See

<http://diwww.epfl.ch/~gerstner/SPNM/SPNM.html>

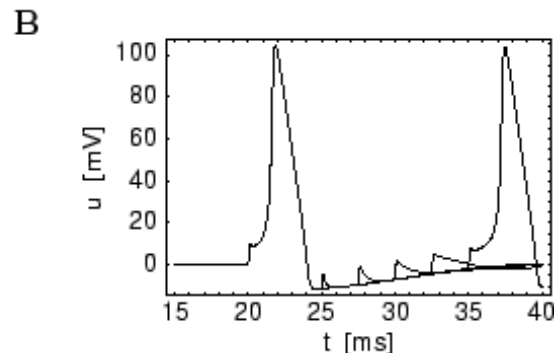
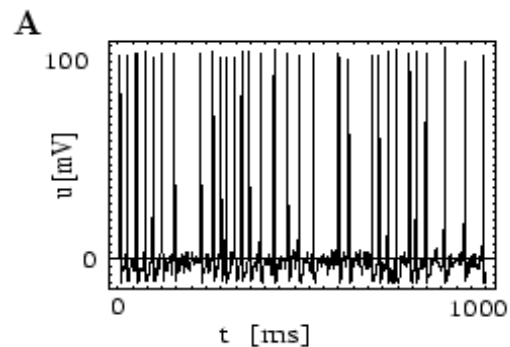
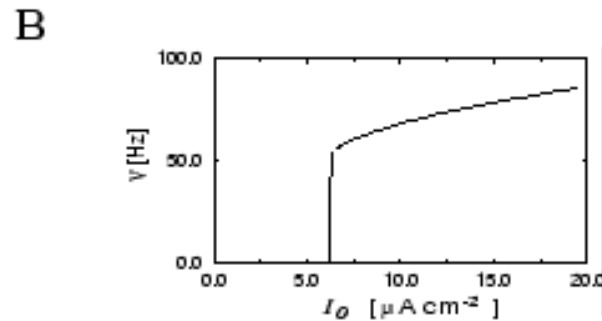
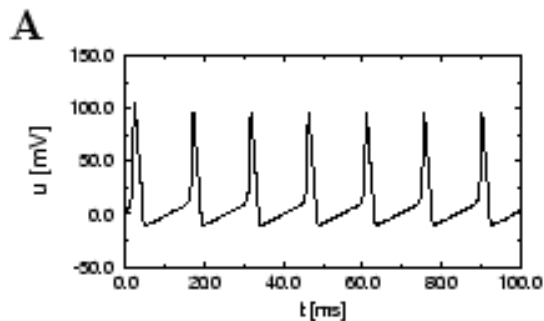
for excellent additional material (some reproduced here).

- Just a piece of passive dendrite can yield complicated differential equations which have been extensively studied by electronics in the context of the study of coaxial cables (TV antenna cable):



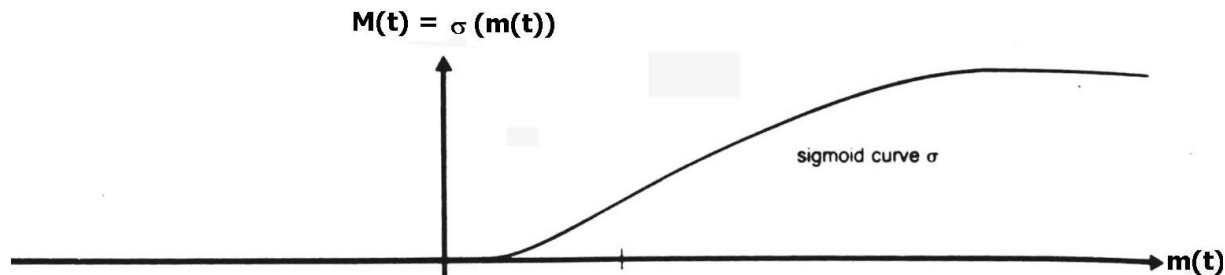
# The Hodgkin-Huxley Model

Example spike trains obtained...



# Leaky Integrator Neuron

- The simplest "realistic" neuron model is a continuous time model based on using the **firing rate** (e.g., the number of spikes traversing the axon in the most recent 20 msec.) as a continuously varying measure of the cell's activity
- The state of the neuron is described by a single variable, the **membrane potential**.
- The firing rate is approximated by a sigmoid, function of membrane potential.





# Leaky Integrator Model

$$\tau \dot{m}(t) = -m(t) + h$$

has solution  $m(t) = e^{-t/\tau} m(0) + (1 - e^{-t/\tau})h$

□  $\tau$  h for time constant  $\tau > 0$ .

- We now add synaptic inputs to get the

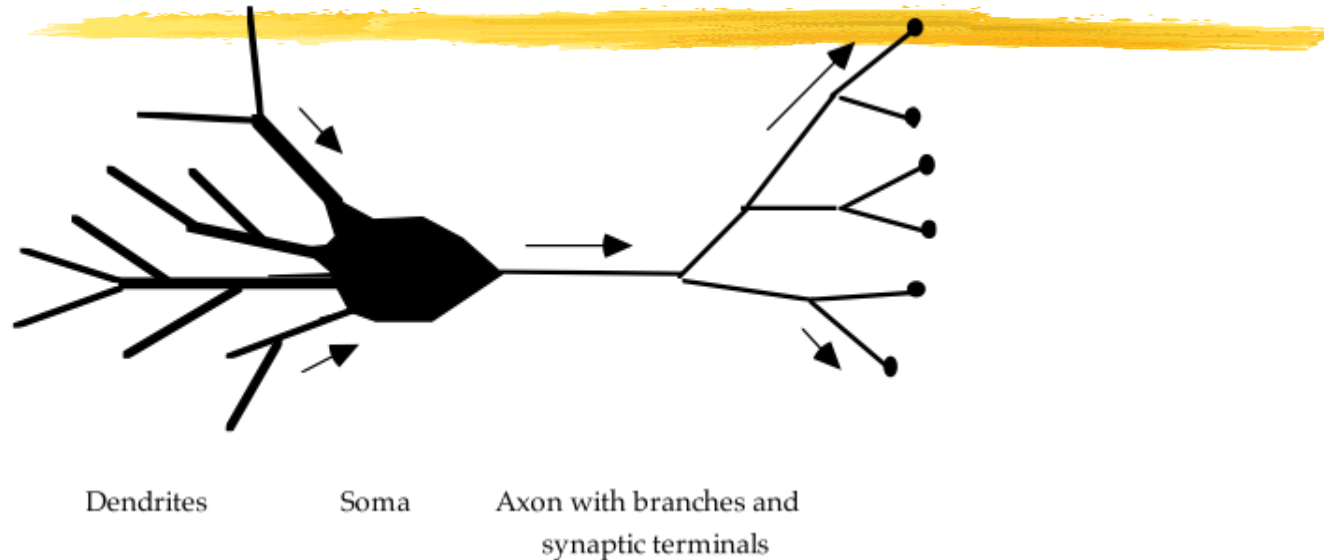
## Leaky Integrator Model:

$$\tau \dot{m}(t) = -m(t) + \sum_i w_i X_i(t) + h$$

where  $X_i(t)$  is the firing rate at the  $i^{\text{th}}$  input.

- Excitatory input ( $w_i > 0$ ) will increase  $\dot{m}(t)$
- Inhibitory input ( $w_i < 0$ ) will have the opposite effect.
- $X(t) = g(m(t))$  with  $g()$  a sigmoid relates output to membrane potential

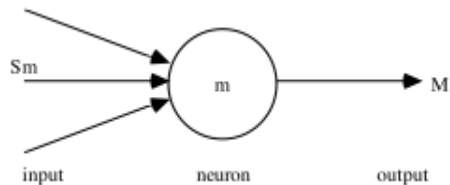
## The "basic" biological neuron



- 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.

# Vision, AI and ANNs

- 1940s: beginning of Artificial Neural Networks



McCulloch & Pitts, 1942

$$\sum w_i x_i \geq \theta$$

Perceptron learning rule (Rosenblatt, 1962)

Backpropagation

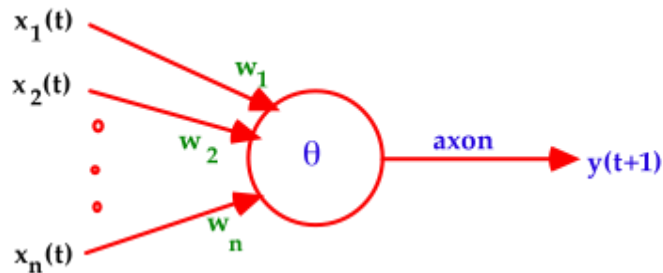
Hopfield networks (1982)

Kohonen self-organizing maps

...

## Warren McCulloch and Walter Pitts (1943)

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



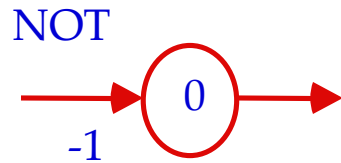
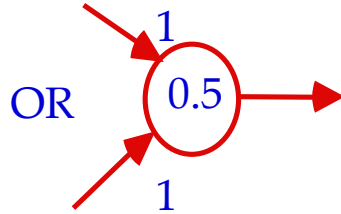
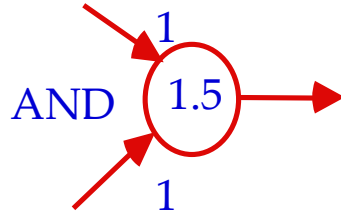
- At each time step, an input or output is  
on or off — 1 or 0, respectively.
- Each connection or synapse from the output of one neuron to the input of another, has an attached weight.

# Excitatory and Inhibitory Synapses

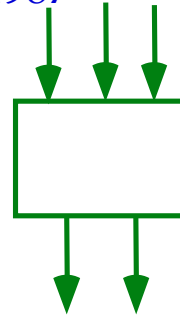
- We call a synapse  
excitatory if  $w_i > 0$ , and  
inhibitory if  $w_i < 0$ .
- We also associate a threshold  $\theta$  with each neuron
- A neuron fires (i.e., has value 1 on its output line) at time  $t+1$  if the weighted sum of inputs at  $t$  reaches or passes  $\theta$ :

$$y(t+1) = 1 \text{ if and only if } \sum w_i x_i(t) \geq \theta$$

## From Logical Neurons to Finite Automata

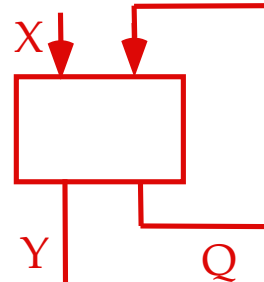


Brains, Machines, and  
Mathematics, 2nd Edition,  
1987



Boolean Net

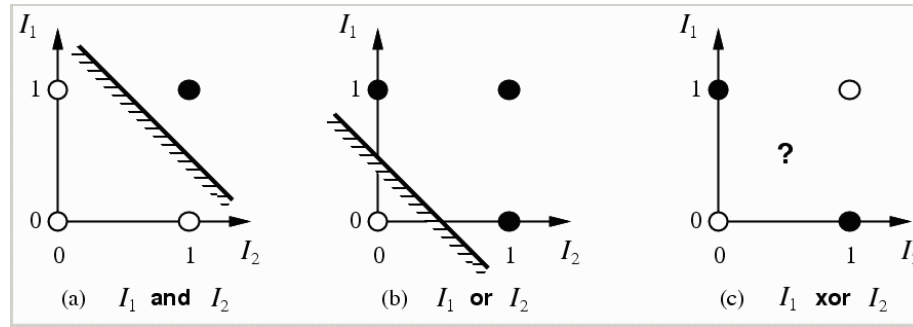
$X \rightarrow Y$



Finite  
Automaton



# limitations of a single layer perceptron and linear separability problem

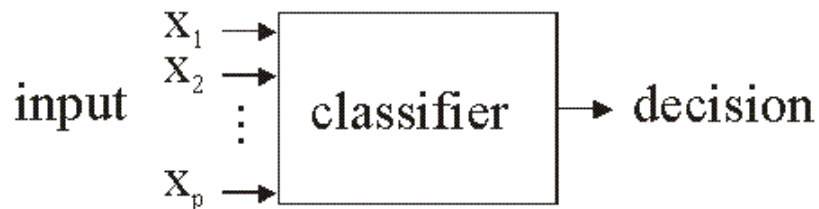


- a single layer perceptron can only represent **linearly separable functions** which follows from the inequalities that determine its firing (Minsky & Papert, 1969)
- $y(t+1) = 1$  if and only if  $\sum w_i x_i(t) \geq \theta$  else  $y(t+1) = 0$
- i.e. input space is divided in 2

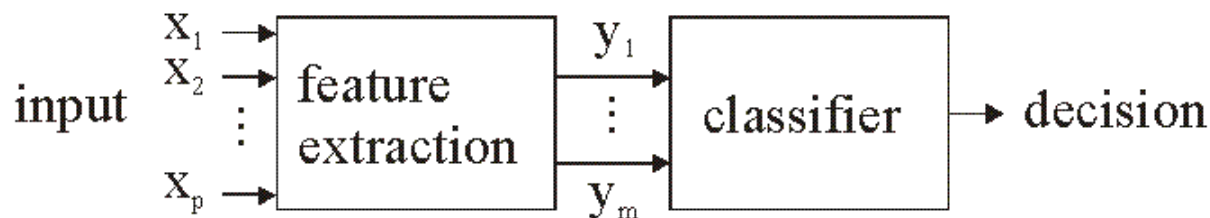
# Classifiers

- <http://www.electronicsletters.com/papers/2001/0020/paper.asp>

- 1-stage approach

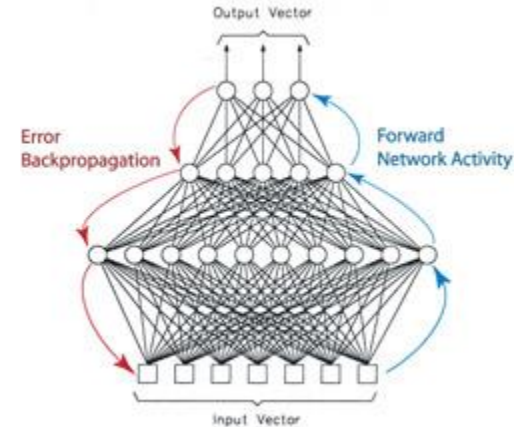
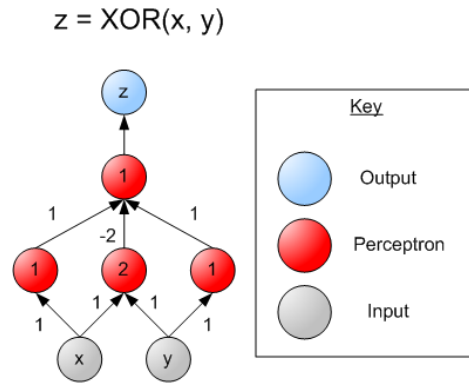


- 2-stage approach



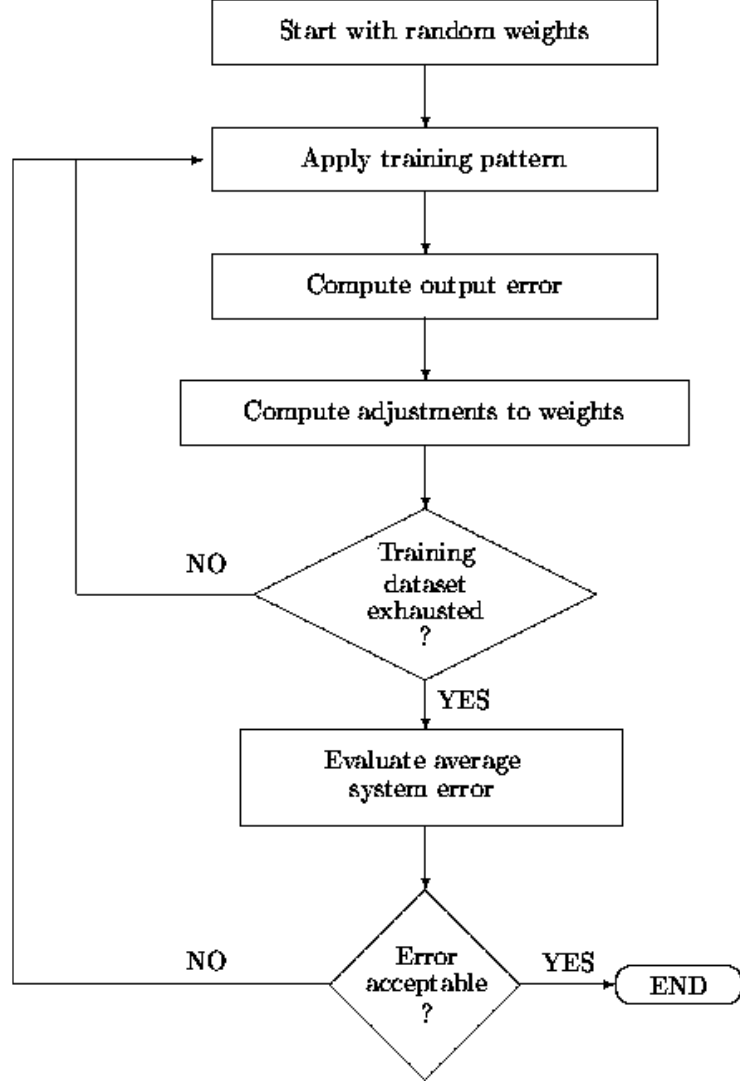
# multi-layer perceptrons

- single layer perceptrons can only represent linear decision surfaces
- multi-layer perceptrons can represent non-linear decision surfaces.

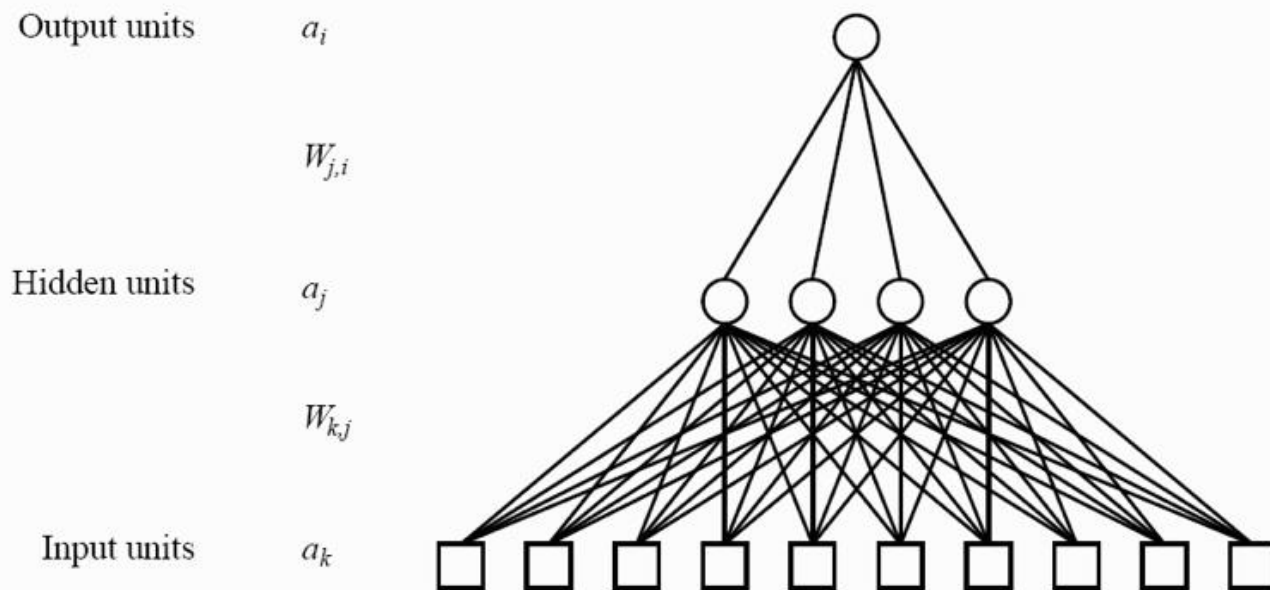


# training a multilayer network

<http://ams.egeo.sai.jrc.it/eurostat/Lot16-SUPCOM95/node7.html>



# training a multilayer network using Backpropogation



$$W_{ji} = W_{ji} + \alpha * a_j * \delta_i$$

$$\delta_i = (T_i - O_i) * g'(in_i) \quad \text{For output nodes}$$

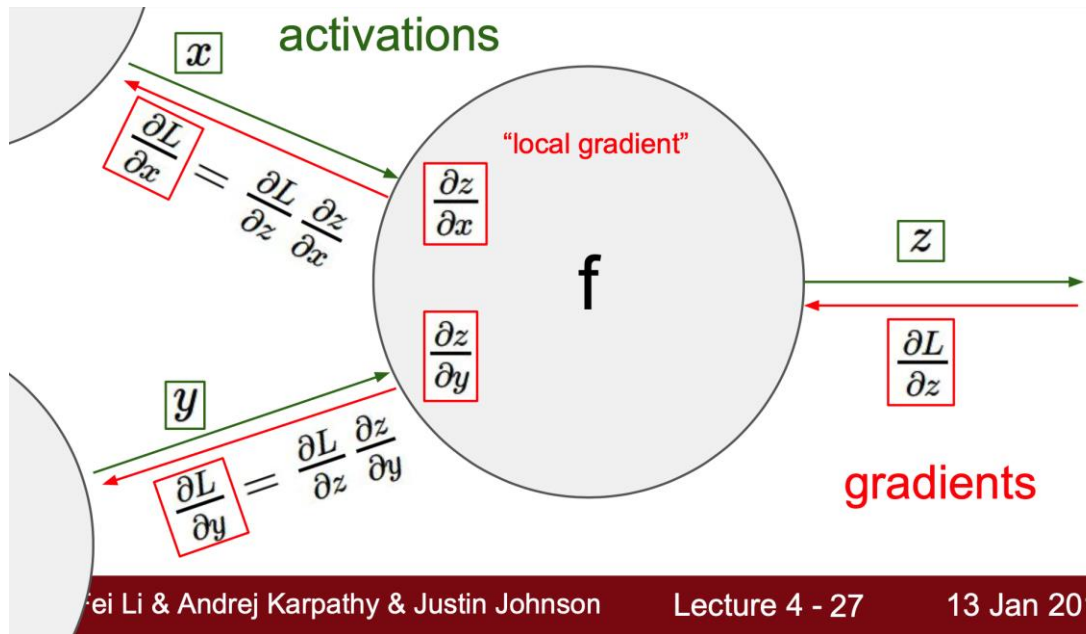
$$W_{kj} = W_{kj} + \alpha * a_k * \delta_j$$

$$\delta_j = g'(in_j) \sum W_{ji} \delta_i \quad \text{For hidden nodes}$$

# training a multilayer network using Backpropogation

Key idea of backprop:

Use chain rule to split output gradients into several input gradients

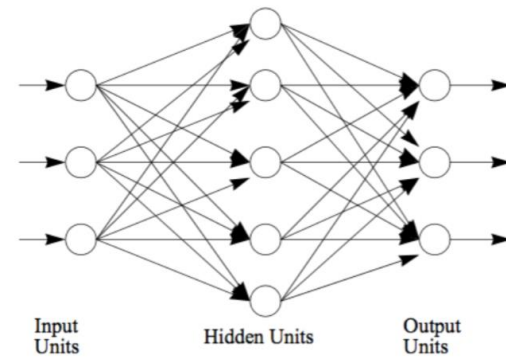




# Neural Networks

## (The back-propagation math)

### (ALFE chapter 4)



The output  $o_j$  of a unit  $j$  in a network depends on the unit's mapping function  $f_j$ , its inputs  $o_{i_1}, o_{i_2}, \dots, o_{i_m}$  (these are outputs of units that feed into unit  $j$ ), and the weights associated with each input  $w_{i_1j}, w_{i_2j}, \dots, w_{i_mj}$ . If we define the net input to the unit  $j$  to be

$$I_j = \sum_i o_i w_{ij}$$

then the output of unit  $j$  is given as

$$o_j = f_j(I_j)$$



The objective of a learning algorithm, such as back-propagation, is to change the weights in a network to reduce the difference between the the actual output of the network and the desired output provided by the training examples.

# Back-Propagation (1/2)

Let  $t_j$  be the desired output of unit  $j$  and  $o_j$  the actual output of unit  $j$ , we define the difference to be

$$E = \frac{1}{2}(t_j - o_j)^2$$

Using the gradient descent method, how much a weight  $w_{ij}$  needs to be changed can be computed as

$$w_{ij} = w_{ij} - \eta \nabla E \quad (4.2)$$

where  $\eta$  is a step size. The factor  $\nabla E$  can be computed as follows:

$$\begin{aligned} \nabla E &= \frac{\partial E}{\partial w_{ij}} = \left( \frac{\partial E}{\partial o_j} \right) \left( \frac{\partial o_j}{\partial w_{ij}} \right) \\ &= \left( \frac{\partial E}{\partial o_j} \right) \left( \frac{\partial o_j}{\partial I_j} \right) \left( \frac{\partial I_j}{\partial w_{ij}} \right) \\ &= \frac{\partial E}{\partial o_j} f'_j(I_j) o_i \end{aligned}$$

When the unit  $j$  is an output unit, then according to the definition of  $E$ , we have  $\frac{\partial E}{\partial o_j} = -(t_j - o_j)$ , thus

$$\nabla E = -f'(I_j) o_i (t_j - o_j) \quad (4.3)$$

## Back-Propagation (2/2)

When the unit  $j$  is not an output unit, then the error of  $o_j$  depends on the units  $k$  that  $j$  outputs to. Thus, we can rewrite  $\frac{\partial E}{\partial o_j}$  as

$$\frac{\partial E}{\partial o_j} = \sum_k \left( \frac{\partial E}{\partial I_k} \right) \left( \frac{\partial I_k}{\partial o_j} \right) = \sum_k \frac{\partial E}{\partial I_k} w_{jk}$$

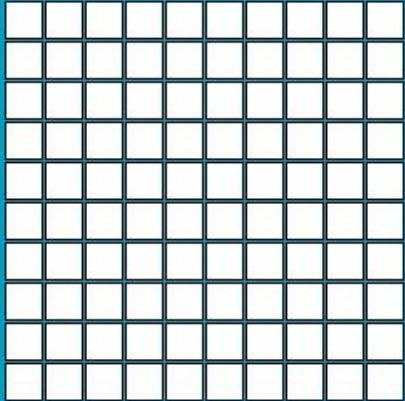
where unit  $k$  is the unit from  $j$  through  $w_{jk}$  and  $I_k$  is net input of the unit  $k$ . Thus, we have

$$\nabla E = f'(I_j) o_i \sum_k \frac{\partial E}{\partial I_k} w_{jk} \quad (4.4)$$

Using Equations 4.2, 4.3, and 4.4 we can update every weight in the network. These three equations represent the method of back-propagation.

## Demo

### Backpropagation Network



**I/O Module**

Clear Grid

Load Patt...

Save As

Add Noise

Recognize

**Network Parameters (# of neurons)**

Hidden Layer ...

Hidden Layer ...

**Training Set**

☐ A ☐ F ☐ tmp1

☐ B ☐ G ☐ tmp2

☐ C ☐ H

☐ D

☐ E

**Learning Paramet...**

Iteration...

Learn Rat...

Moment...

Error :

Train

**Status**

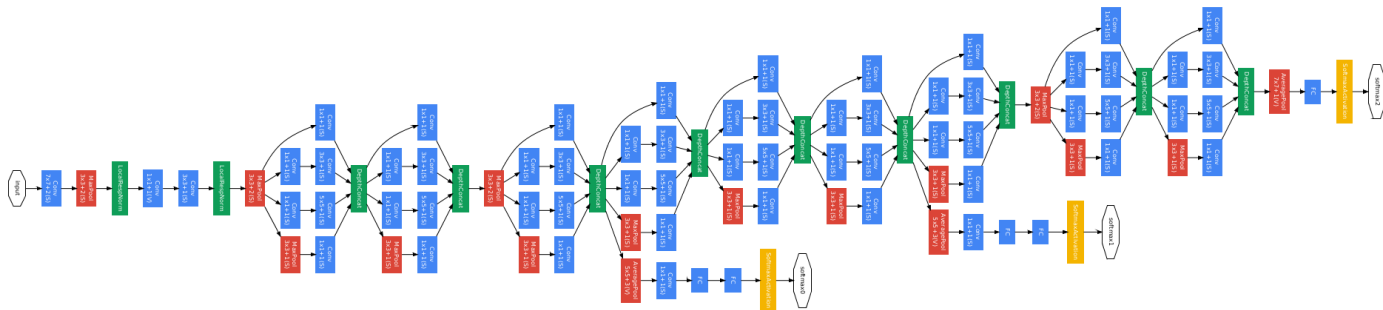
## Increasing the Realism of Neuron Models



- The McCulloch-Pitts neuron of 1943 is important as a basis for
  - logical analysis of the neurally computable, and
  - current design of some neural devices (especially when augmented by **learning rules** to adjust synaptic weights).
- However, it is no longer considered a useful model for making contact with neurophysiological data concerning real neurons.

## Deep neural networks

- Very deep layered networks with hundreds or thousands of layers, and hundreds of millions of parameters.
- Trained on very large datasets.
- More about these in session 20.





## Hopfield Networks



- A paper by John Hopfield in 1982 was the catalyst in attracting the attention of many physicists to "Neural Networks".
- In a network of McCulloch-Pitts neurons whose output is 1 iff  $\sum_j w_{ij} s_j \geq \theta_i$  and is otherwise 0, neurons are updated synchronously: every neuron processes its inputs at each time step to determine a new output.

# Hopfield Networks



- A Hopfield net (Hopfield 1982) is a net of such units subject to the **asynchronous rule for updating one neuron at a time**:

"Pick a unit  $i$  at random.

If  $\sum_j w_{ij} s_j \geq \theta_i$ , turn it on.

Otherwise turn it off."

- Moreover, Hopfield assumes **symmetric weights**:

$$w_{ij} = w_{ji}$$

## “Energy” of a Neural Network



- Hopfield defined the “energy”:

$$E = - \frac{1}{2} \sum_{ij} s_i s_j w_{ij} + \sum_i s_i \theta_i$$

- If we pick unit  $i$  and the firing rule (previous slide) does not change its  $s_i$ , it will not change  $E$ .

## **$s_i$ : 0 to 1 transition**

- If  $s_i$  initially equals 0, and  $\odot w_{ij}s_j \varepsilon \ell_i$

then  $s_i$  goes from 0 to 1 with all other  $s_j$  constant,  
and the "energy gap", or change in  $E$ , is given by

$$\begin{aligned}\otimes E &= -\frac{1}{2} \odot_j (w_{ij}s_j + w_{ji}s_j) + \ell_i \\ &= -(\odot_j w_{ij}s_j - \ell_i) \quad \text{(by symmetry)} \\ \delta &0.\end{aligned}$$

## **$s_i$ : 1 to 0 transition**



- If  $s_i$  initially equals 1, and  $\sum_j w_{ij}s_j < \theta_i$

then  $s_i$  goes from 1 to 0 with all other  $s_j$  constant

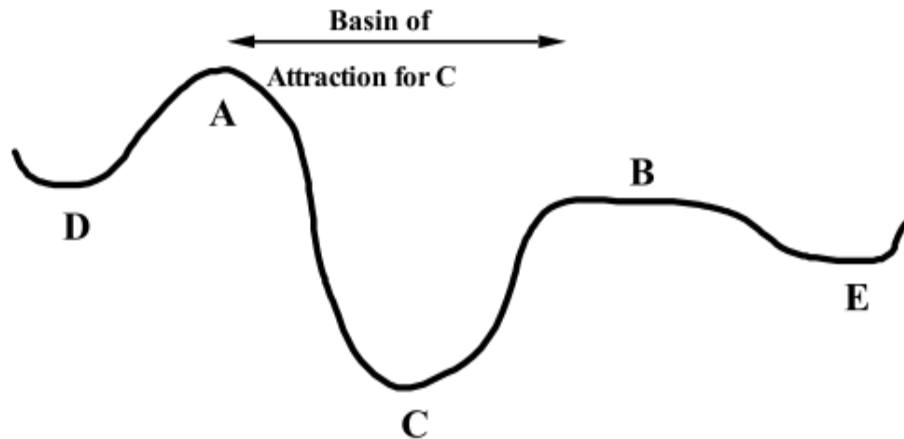
The "energy gap," or change in  $E$ , is given, for symmetric  $w_{ij}$ , by:

$$\Delta E = \sum_j w_{ij}s_j - \theta_i < 0$$

- **On every updating we have  $\Delta E \leq 0$**

## Minimizing Energy

- On every updating we have  $\frac{\partial E}{\partial \theta} \neq 0$
- Hence the dynamics of the net tends to move E toward a minimum.
- We stress that there may be different such states — they are local minima. Global minimization is not guaranteed.



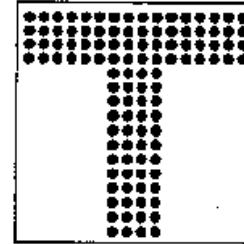
# Associative Memories

- <http://www.shef.ac.uk/psychology/gurney/notes/I5/I5.html>

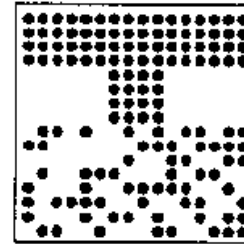
Idea: store:

So that we can recover it if presented with corrupted data such as:

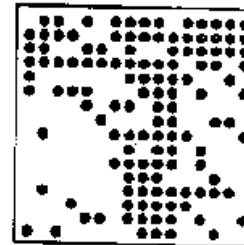
Or:



Original 'T'



half of image  
corrupted by  
noise



20% corrupted  
by noise  
(whole image)

## Hopfield Network Applet

Parameters

Grid size (X)

10

Grid size (Y)

10

Stored patterns

3

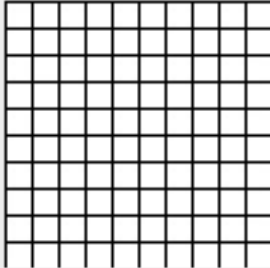
Corruption %

20

Iteration delay

25

Stored patter...



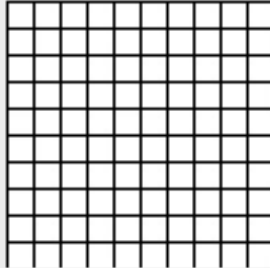
Load

Clear

<<

>>

Corrupted pattern



Alter

Clear

Go!

Stop

Hopfield Network v1.3 (c) 2001 by Kriangsiri Malasri

Pattern 1 of 3

Please send issues/bug reports to the programmer at [kmalasri@hotmail.com](mailto:kmalasri@hotmail.com) or [gte985m@prism.gatech.edu](mailto:gte985m@prism.gatech.edu).

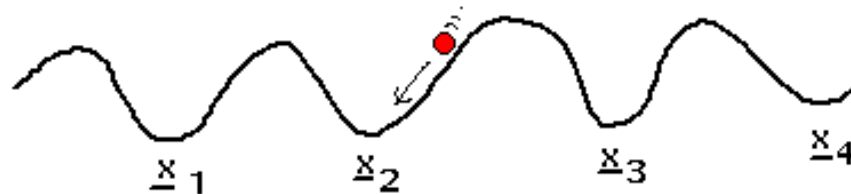
CS 561, Session 15

40



# Associative memory with Hopfield nets

- Setup a Hopfield net such that local minima correspond to the stored patterns.
- Issues:
  - because of weight symmetry, anti-patterns (binary reverse) are stored as well as the original patterns (also spurious local minima are created when many patterns are stored)
  - if one tries to store more than about **0.14\*(number of neurons)** patterns, the network exhibits unstable behavior
  - works well only if patterns are uncorrelated



$\{x_1, x_2, x_3, x_4, \dots\}$  are the 'memories' stored

# Capabilities and Limitations of Layered Networks



- Issues:
  - what can given networks do?
  - What can they learn to do?
  - How many layers required for given task?
  - How many units per layer?
  - When will a network generalize?
  - What do we mean by generalize?
  - ...

# Capabilities and Limitations of Layered Networks



- What about boolean functions?
- Single-layer perceptrons are very limited:
  - XOR problem
  - etc.
- But what about multilayer perceptrons?

We can represent any boolean function with a network with just one hidden layer.

How??

# Capabilities and Limitations of Layered Networks



To approximate a set of functions of the inputs by a layered network with continuous-valued units and sigmoidal activation function...

Cybenko, 1988: ... **at most two hidden layers** are necessary, with arbitrary accuracy attainable by adding more hidden units.

Cybenko, 1989: **one hidden layer** is enough to approximate any continuous function.

**Intuition of proof:** decompose function to be approximated into a sum of localized “bumps.” The bumps can be constructed with two hidden layers.

Similar in spirit to Fourier/wavelet decomposition. Bumps = radial basis functions.

However, in practice, deep networks are easier to train than very wide 2-layer networks.

# Optimal Network Architectures



How can we determine the number of hidden units?

- **genetic algorithms**: evaluate variations of the network, using a metric that combines its performance and its complexity. Then apply various mutations to the network (change number of hidden units) until the best one is found.
- **Pruning and weight decay**:
  - apply weight decay during training
  - eliminate connections with weight below threshold
  - re-train
- **How about eliminating units?** For example, eliminate units with total synaptic input weight smaller than threshold.

## For further information



- See

Hertz, Krogh & Palmer: Introduction to the theory of neural computation (Addison Wesley)

In particular, the end of chapters 2 and 6.