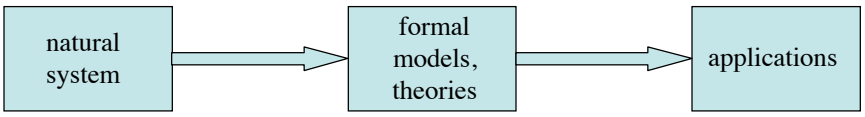
CMSC 421: Neural Computation

- definition
- synonyms
  - neural networks
  - artificial neural networks
  - neural modeling
  - connectionist models
  - parallel distributed processing
- AI perspective

Applications of Neural Networks

- pattern classification
  - virus/explosive detection, financial predictions, etc.
- image processing
  - character recognition, manufacturing inspection, etc.
- control
  - autonomous vehicles, industrial processes, etc.
- optimization
  - VLSI layout, scheduling, etc.
- bionics
  - prostheses, brain implants, etc.
- brain/cognitive models
  - memory, learning, disorders, etc.

Nature-Inspired Computation



biology, physics, etc.

interdisciplinary

computer science engineering

neural networks
genetic programming
swarm intelligence
self-replicating machines
..

Inspiration for neural networks?

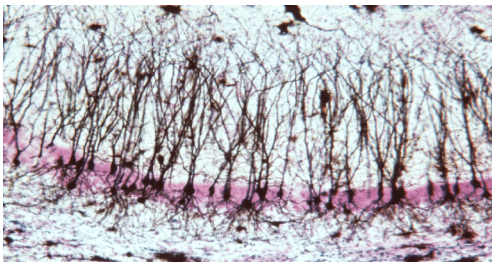
Neural Networks

nature	model/theory	applications
	random networks Hebbian learning perceptrons error backpropagation self-organizing maps …	pattern classification speech recognition image processing text-to-speech expert systems autonomous vehicles financial predictions associative memory data visualization … + brain modeling

The Brain

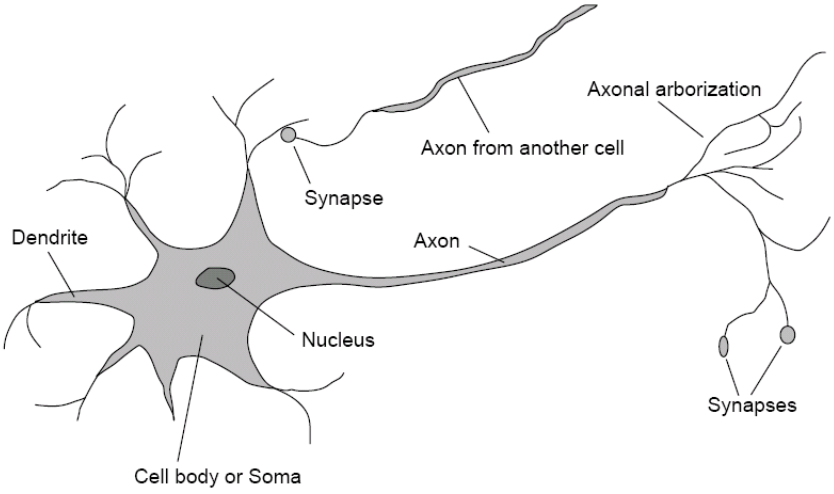
- complex
- flow of information
- what is known?

- neurons
- synapses



(Purkinje cells; Golgi + Nissl stains)

Neuron Information Processing



How Does the Brain Compute?

A familiar example …

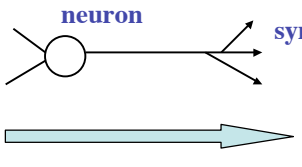
How fast is the processing?
- cycle time vs. CPU…
- signal speeds …

How does it do that?! …

- massively parallel processing 10^{11} neurons
- different computational principles

Summary: Brain as Inspiration

network of neurons
10^{11} neurons, 10^{14} synapses



spikes (pulses)

Relevance to AI:
Can a machine think?
Alan Turing and "weak AI"
Prospects for "strong AI"?

The Computer vs. The Brain

• information access	global	local
• control	centralized	decentralized
• processing method	sequential	massively parallel
• how programmed	programmed	self-organizing
• adaptability	minimally	prominently

Our immediate focus: supervised/inductive learning

History of Neural Networks

1945-1955: pre-computing
1955-1970: classical period
1970-1985: dark ages
1985-1995: renaissance
1995-today: modern era

perceptrons

error back-propagation

Neural Computation

- basics
- feedforward networks
  - perceptrons
  - error backpropagation
- recurrent networks

Neural Network Basics

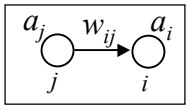
neural network = network + activation rule + learning rule

Neural Networks

1. network graph

node/neuron activation level a_i

connection/synapse weight w_{ij}



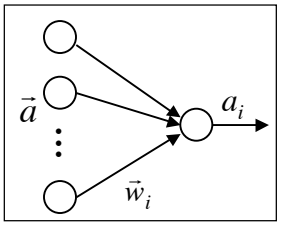
excitatory: $w_{ij} > 0$

inhibitory: $w_{ij} < 0$

feedforward vs. recurrent networks

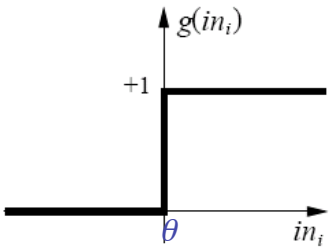
2. activation rule $in_i = \sum_j w_{ij} a_j$

$a_i = g(in_i)$



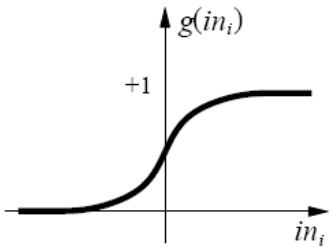
• "executing" a neural network

Choices for Activation Function



LTU (step)

$a_i = \text{step}_\theta(in_i)$



logistic (sigmoid)

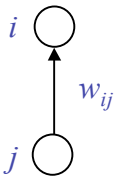
$a_i = \sigma(in_i)$

local computations → emergent behavior

others: sign, tanh, linear, radial basis, …

3. Learning Rule

weight changes as function of local activity

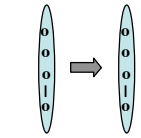


$$\Delta w_{ij} = f(a_j, a_i, in_i, w_{ij}, \ldots)$$

Neural Computation

- basics
- → feedforward networks
  - perceptrons
  - error backpropagation
- recurrent networks

Single Layer Networks



inputs outputs

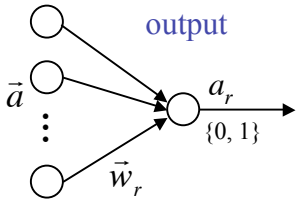
number of layers?

- supervised learning:
  LMS Rule Perceptron Learning Rule
- derived using gradient descent
- associative memory, pattern classification

Elementary Perceptron

LTU = linear threshold unit

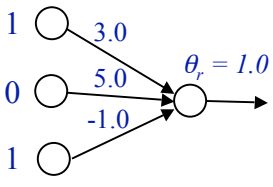
input layer



r = response

$a_r = \text{step}(in_r)$

Example …

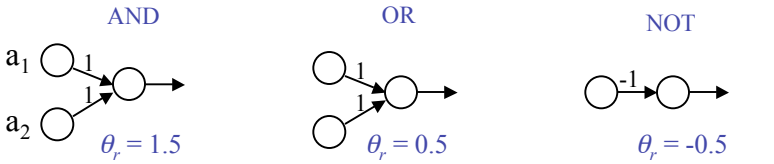


$in_r = 2.0$

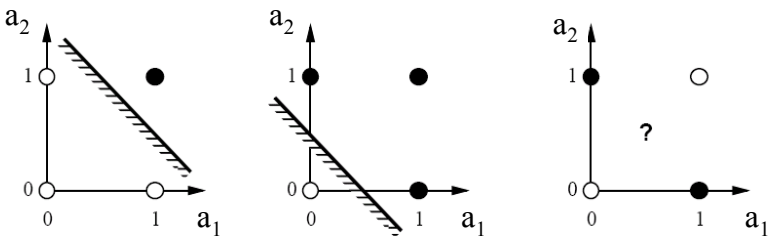
$a_r = 1$

Perceptrons as Logic Gates

Threshold needed to produce …



Linear separability:



Perceptron Learning Rule

If the target output for unit r is t_r $\quad w_{ri} = w_{ri} + \eta(t_r - a_r)a_i$

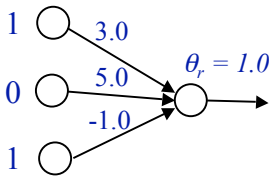
$\eta > 0 \qquad \delta_r = t_r - a_r$

Equivalent to the intuitive rules:

If output is correct:	don't change the weights
If output is low (a_r=0, t_r=1):	increment weights for inputs = 1
If output is high (a_r=1, t_r=0):	decrement weights for inputs = 1
Must also adjust threshold:	$\theta_r = \theta_r - \eta(t_r - a_r)$

(or, equivalently, assume there is a weight w_{r0} for an extra input unit that has a_0=-1: bias node)

Example of Perceptron Learning



$$w_{ri} = w_{ri} + \eta(t_r - a_r)a_i$$
$$\theta_r = \theta_r - \eta(t_r - a_r)$$

Suppose $\eta = 0.1$ and $t_r = 0$ …

Perceptron Learning Algorithm

- repeatedly iterate through examples adjusting weights using perceptron learning rule until all outputs correct
  - initialize the weights randomly or to all zero
  - until outputs for all training examples are correct
    - for each training example do
      - compute the current output a_r
      - compare it to the target t_r and update weights
- each pass through the training data is an epoch
- when will the algorithm terminate?

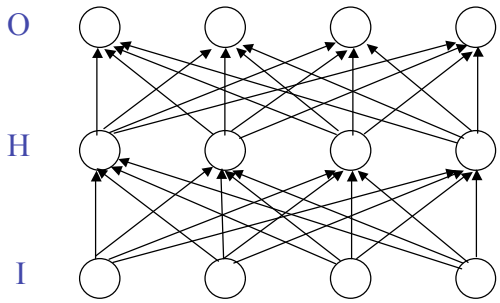
Perceptron Properties

- Perceptron Convergence Theorem: If there are a set of weights that are consistent with the training data (i.e., the data is linearly separable), the perceptron learning algorithm will converge on a solution.
- Perceptrons can only represent linear threshold functions and can therefore only learn functions that linearly separate the data, i.e., the positive and negative examples are separable by a hyperplane in n-dimensional space.
- Unfortunately, some functions (like xor) cannot be represented by a LTU.

Error Backpropagation

- widely used neural network learning method
- seminal version about 1960 (Rosenblatt)
- multiple versions since
- basic contemporary version popularized ≈ 1985
- uses multi-layer feedforward network

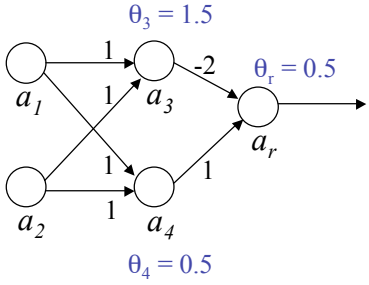
Uses Layered Feedforward Network



Representation Power of Multi-Layer Networks

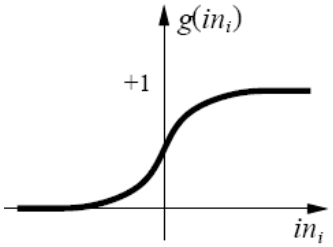
Theorem: any boolean function of N inputs can be represented by a network with one layer of hidden units.

XOR



$a_r = a_4 \wedge \sim a_3$
$= \text{or}(a_1,a_2) \wedge \sim\text{and}(a_1,a_2)$

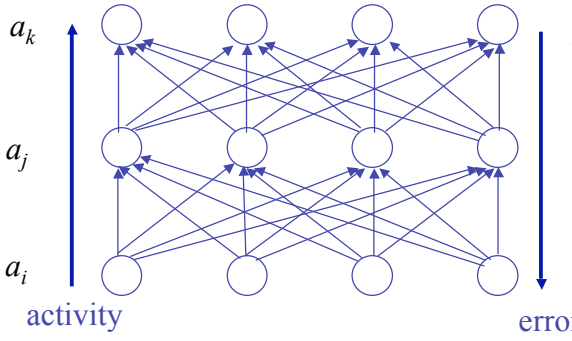
Activation Function



logistic (sigmoid)

$$a_r = \sigma(in_r)$$

Error Backpropagation Learning



$$\delta_k = (t_k - a_k) a_k (1 - a_k)$$

$$\Delta w_{kj} = \eta \delta_k a_j$$

$$\delta_j = \left(\sum_k w_{kj} \delta_k \right) a_j (1 - a_j)$$

$$\Delta w_{ji} = \eta \delta_j a_i$$

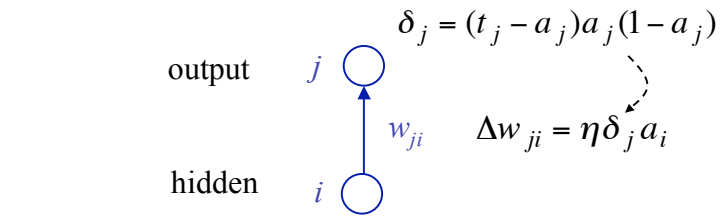
Recall: Perceptron Learning Rule

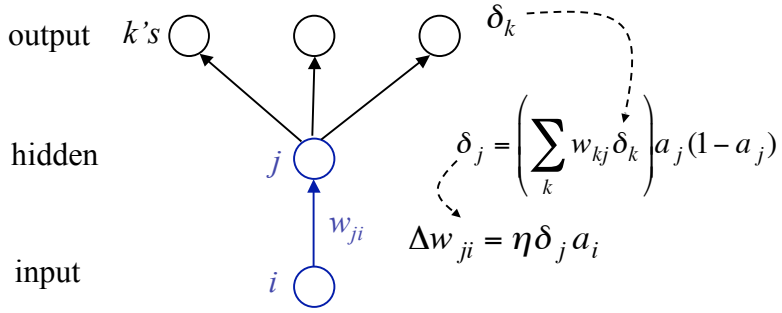
$$w_{ji} = w_{ji} + \eta \underbrace{(t_j - a_j)}_{\delta_j} a_i$$

Rewritten:

$$\boxed{\Delta w_{ji} = \eta \delta_j a_i}$$

EBP Learning Rule





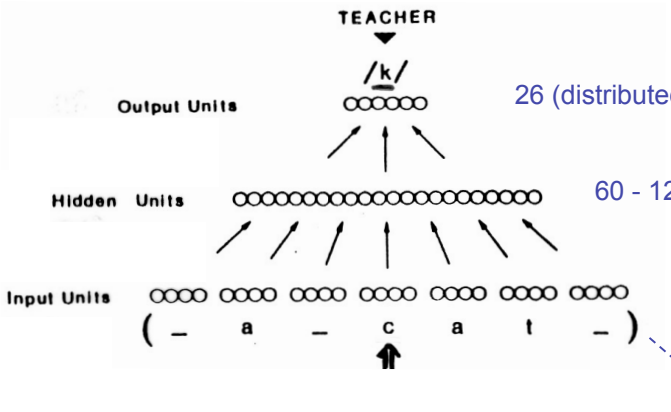
Error Backpropagation

- repeatedly found to be effective in practice
- however, not guaranteed to find solution
- why? hill climbing can get stuck in local minima
- most widely used neural network method

Error Backpropagation Applications

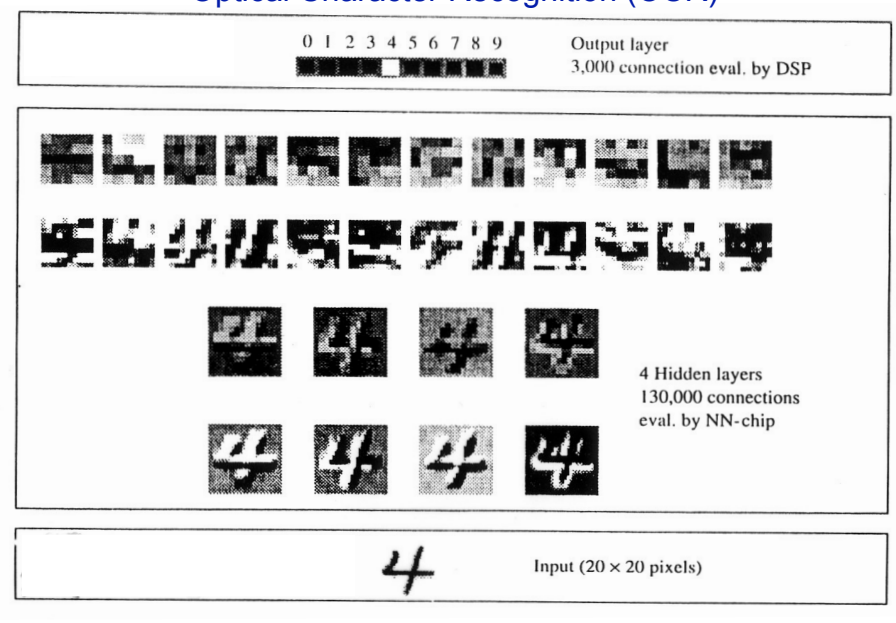
NETtalk
OCR
ALVINN
medical diagnosis
plasma confinement
neuroscience models
cognitive models
…

NETtalk



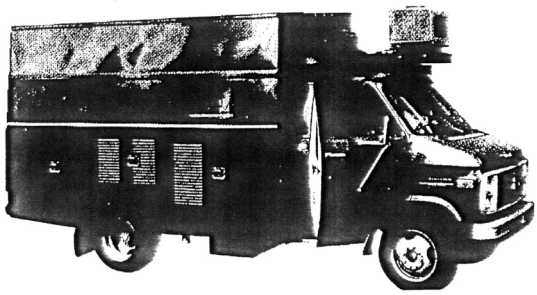
- typically > 18,000 weights
- training data: phonetic transcription of speech
- after 50 epochs: 95% correct on training data, 80% correct on test data

Optical Character Recognition (OCR)



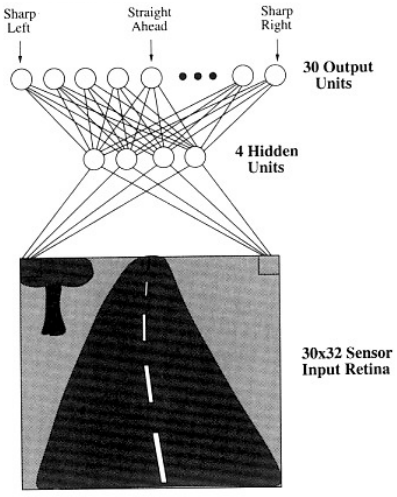
ALVINN

(Autonomous Land Vehicle In a Neural Network)

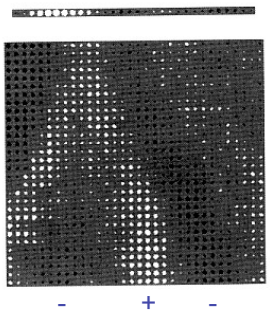


- neural net trained to drive a van along roads viewed through a TV camera
- speeds > 50 mph up to 20 miles (15 images/sec.)

ALVINN's Neural Network



Typical weights from a hidden node:



≈ 4000 weights/biases