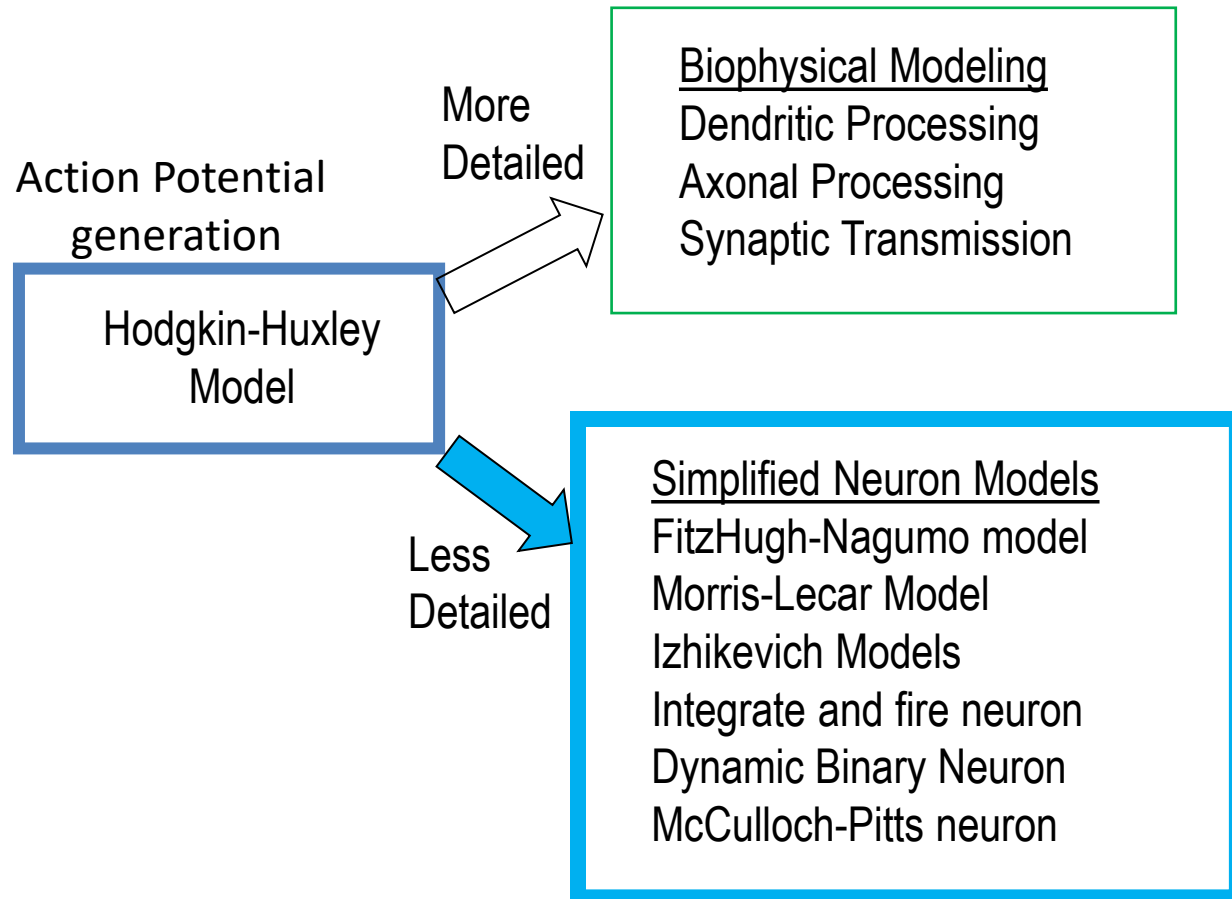
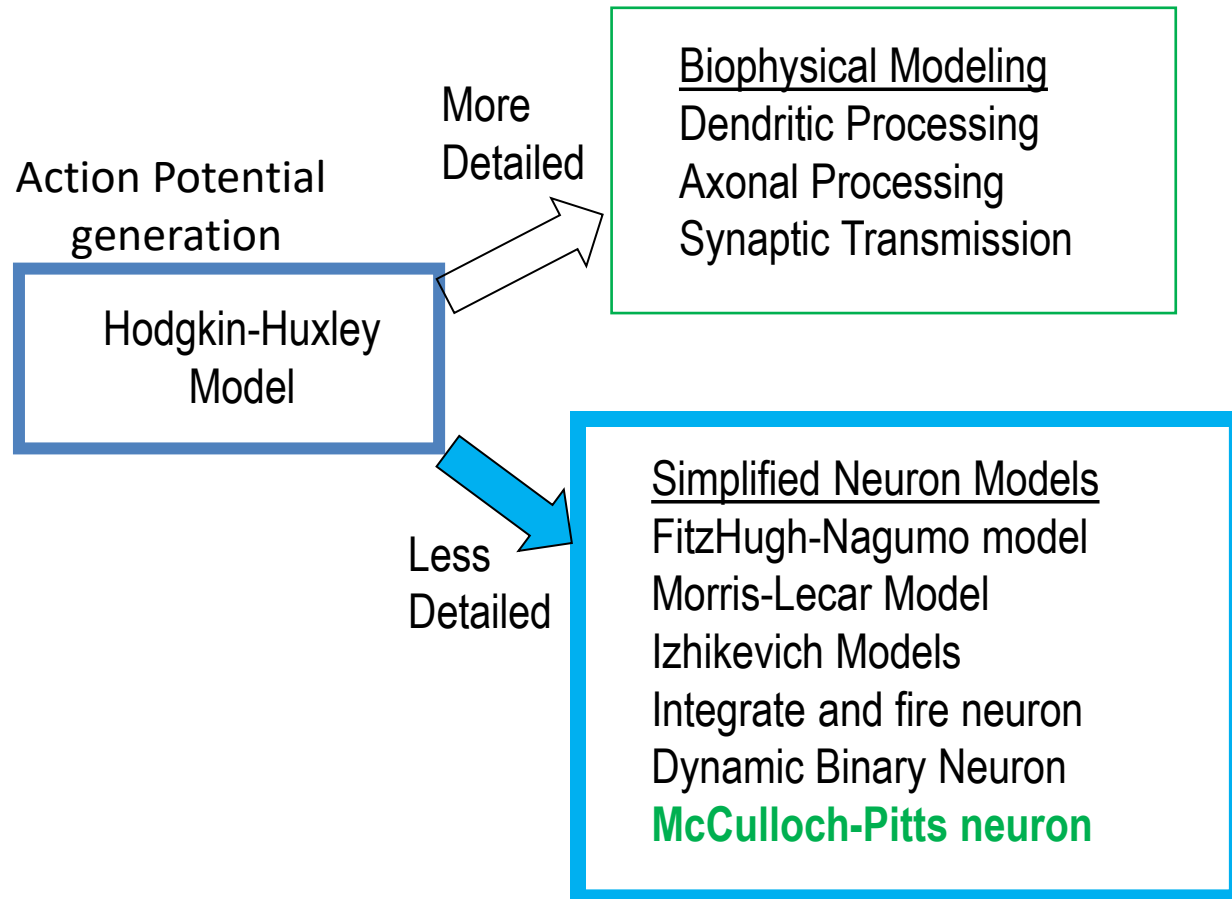


Perceptrons and Multi layered Perceptrons

Single Neuron Modeling



Single Neuron Modeling

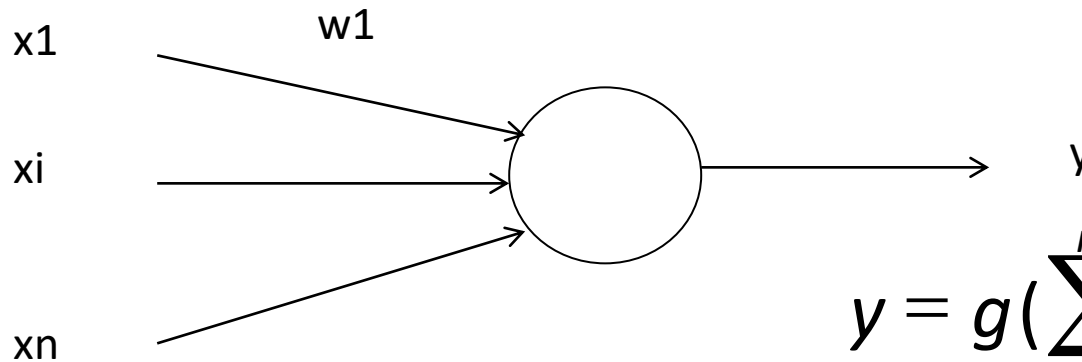


Perceptrons

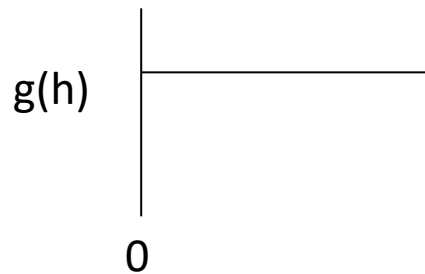
MP neuron

- McCulloch-Pitts neuron model (1943) takes inputs from many neurons and produces a single output.
- If the net effect of the external inputs is greater than a threshold, the neuron goes into excited state (1), else it remains in its resting state (0).

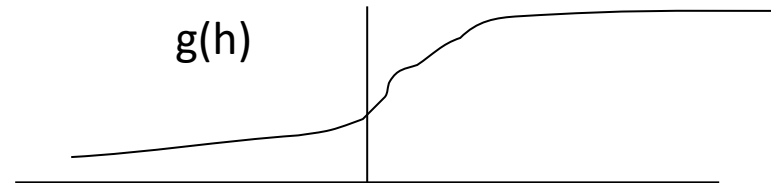
Basic Neuron Model



$$y = g\left(\sum_{i=1}^n w_i x_i - b\right)$$



or



MP neuron model

- McCulloch and Pitts thought that brain works like a computer and neurons are like logic gates
- Since the MP neurons are binary units it seemed worthwhile to check if the basic logical operations can be performed by these neurons.
- McCulloch and Pitts quickly showed that the MP neuron can implement the basic logic gates AND, OR and NOT simply by proper choice of the weights:

A Theory of how the Brain doesn't work

- McCulloch-Pitts neurons can implement logic gates.

- OR Gate: $y = g(x_1 + x_2 - 0.5)$ table:

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

- AND Gate: $y = g(x_1 + x_2 - 1.5)$ table:

X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

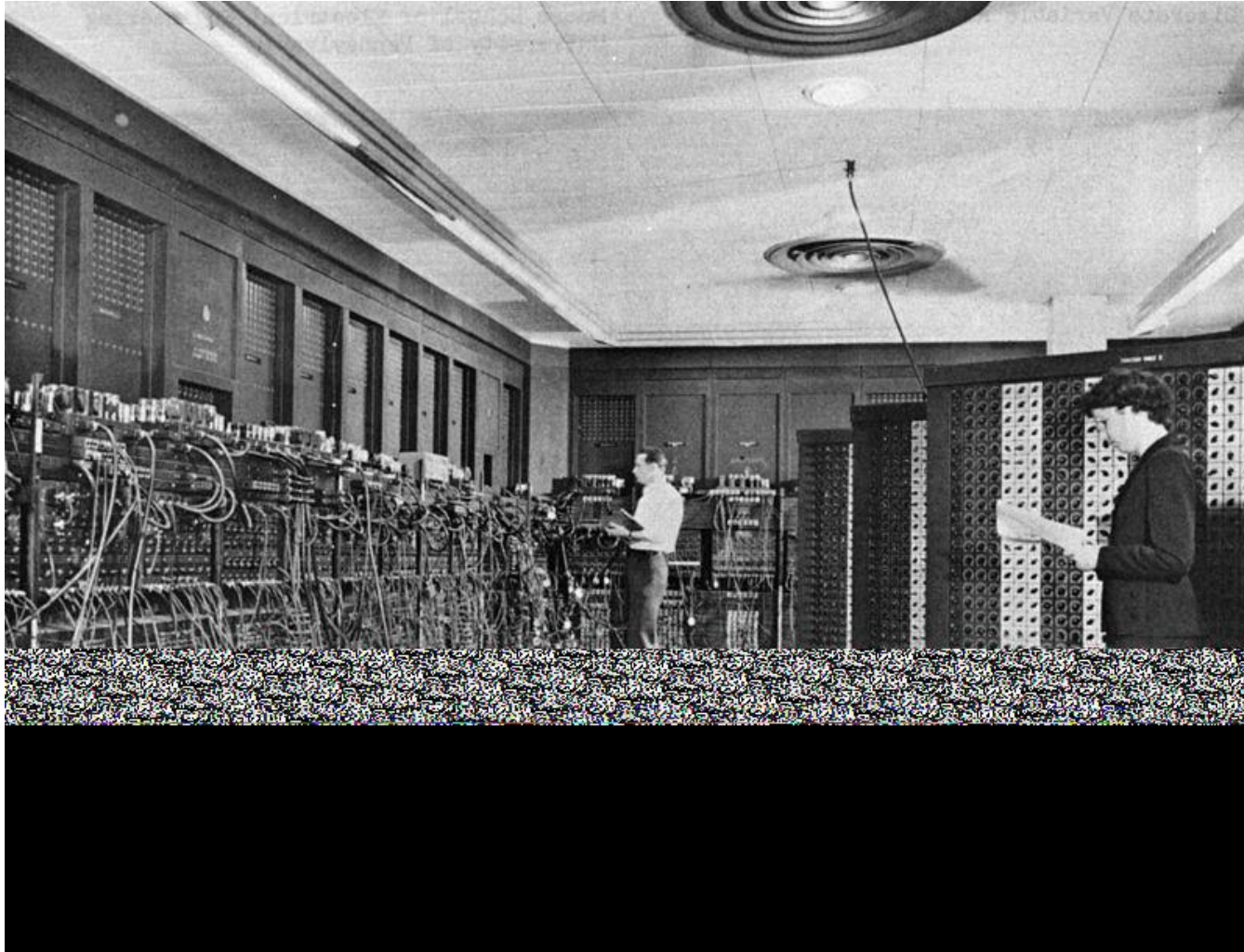
- NOT Gate: $y = g(-x + 0.5)$ table:

X	Y
0	1
1	0

- Hence Brain is a logical machine. (Wrong!)

ENIAC (1945)


(Electronic Numerical Integrator and Computer)



- The idea of considering neurons as logic gates and the brain itself as a large Boolean circuit *does not* satisfy other important requirements of a good theory of the brain.

The Brain and the Computer

• Computer


- Rigid inorganic 2D sheets matter
- Powered by DC mains
- Signals: 5volt DC 
- ✓ Extreme sensitivity to faults
- Clock cycle: $O(10^{-9})$ sec
- Centralized clock
- Power dissipation: 10^{-5} watts

✓ programmed



Brain

fault tolerant

- Soft organic 3D tissue
- Powered by ATP
- 100 mV pulsed 
- ✓ Very fault tolerant,
- 1 mS (200-300Hz)
- No centralized clock

10^{-12} watts

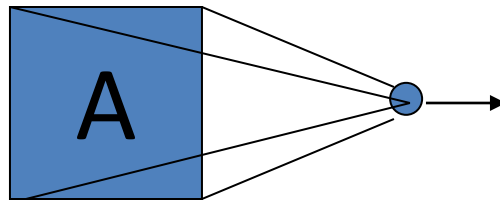
✓ learns from experience
phineas gage



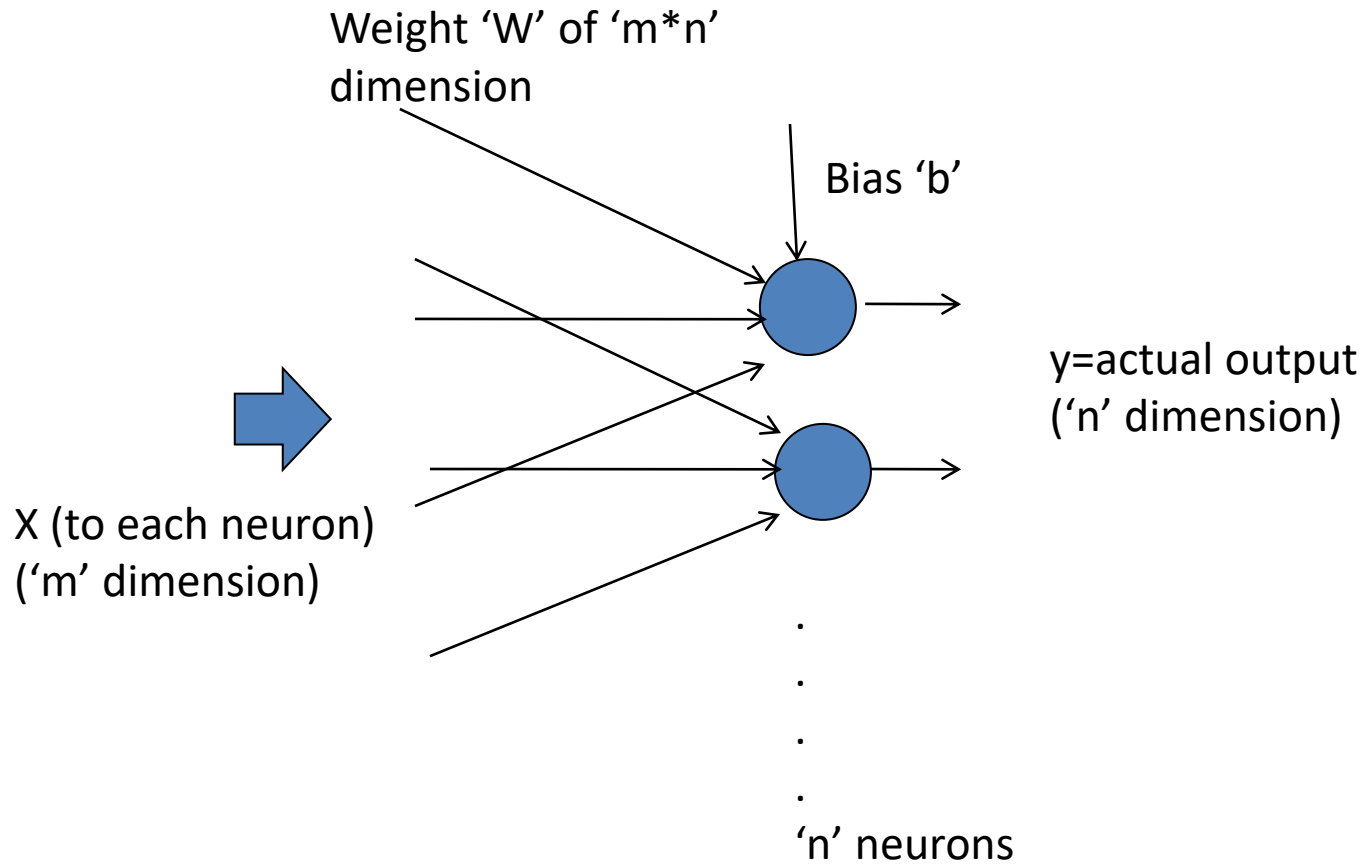
The Perceptron – A NN that learns (1960's)

Frank Rosenblatt

- Learns to recognize simple image patterns
- Uses a single layer of MP neurons



- Limitations:
 - Can classify only “linearly separable” patterns.



MP Neuron divides the input space into two semi-infinite halves

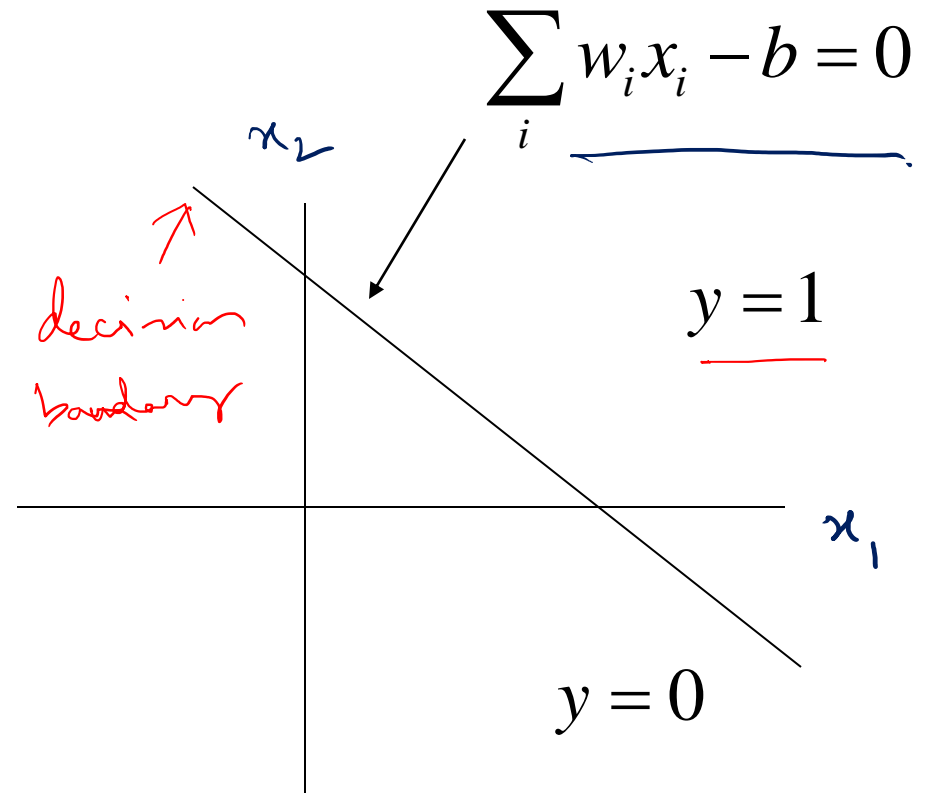
$$y = g(\sum_i w_i x_i - b)$$

$$\text{net} \equiv \sum_i w_i x_i - b$$

$$\text{net} < 0, y = 0$$

$$> 0, y = 1$$

$$x \in \mathbb{R}^n$$



AN EXAMPLE OF HOW A PERCEPTRON LEARNS

Function to be learnt: OR gate

Training
set

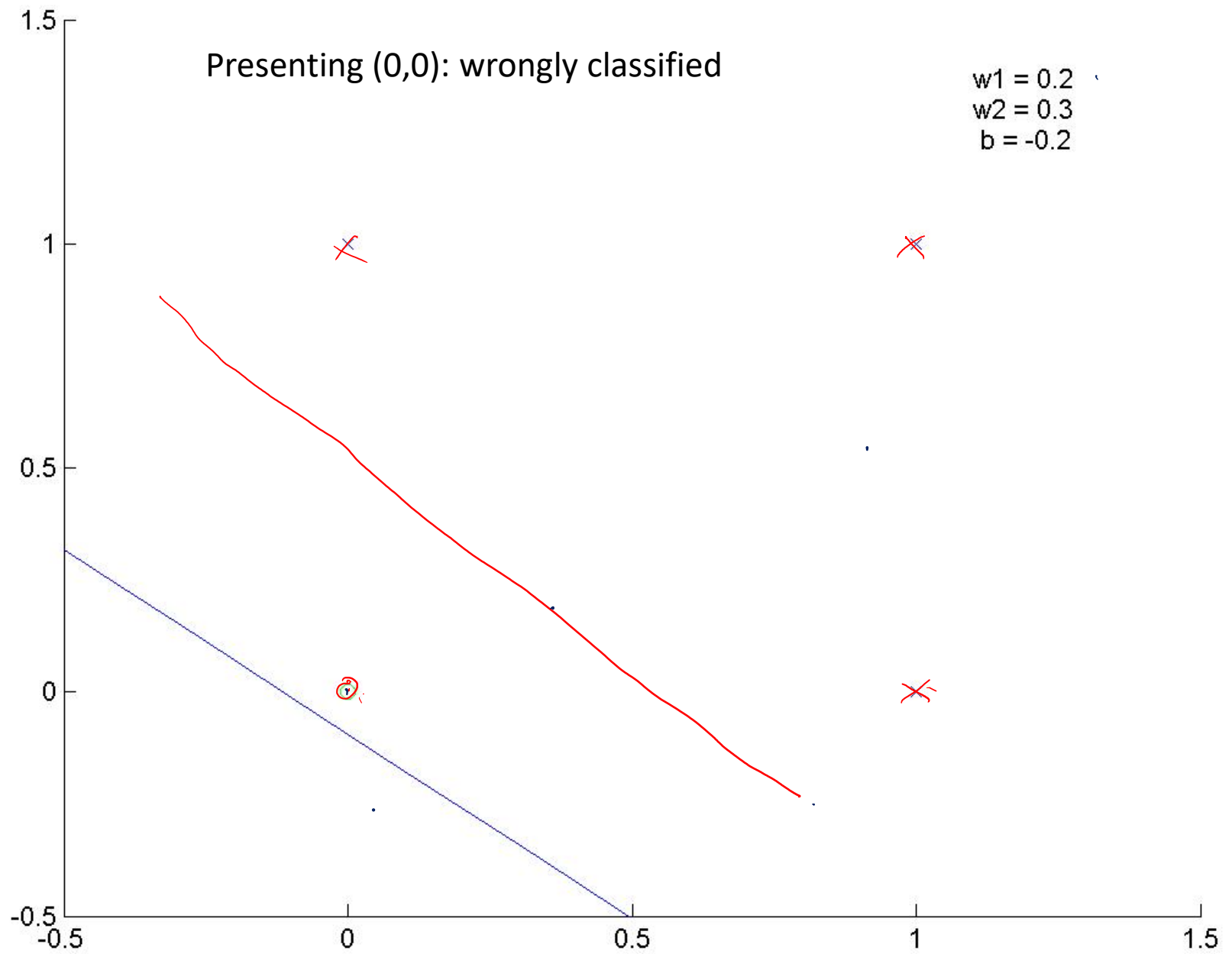
X1	X2	d
0	0	0
0	1	1
1	0	1
1	1	1

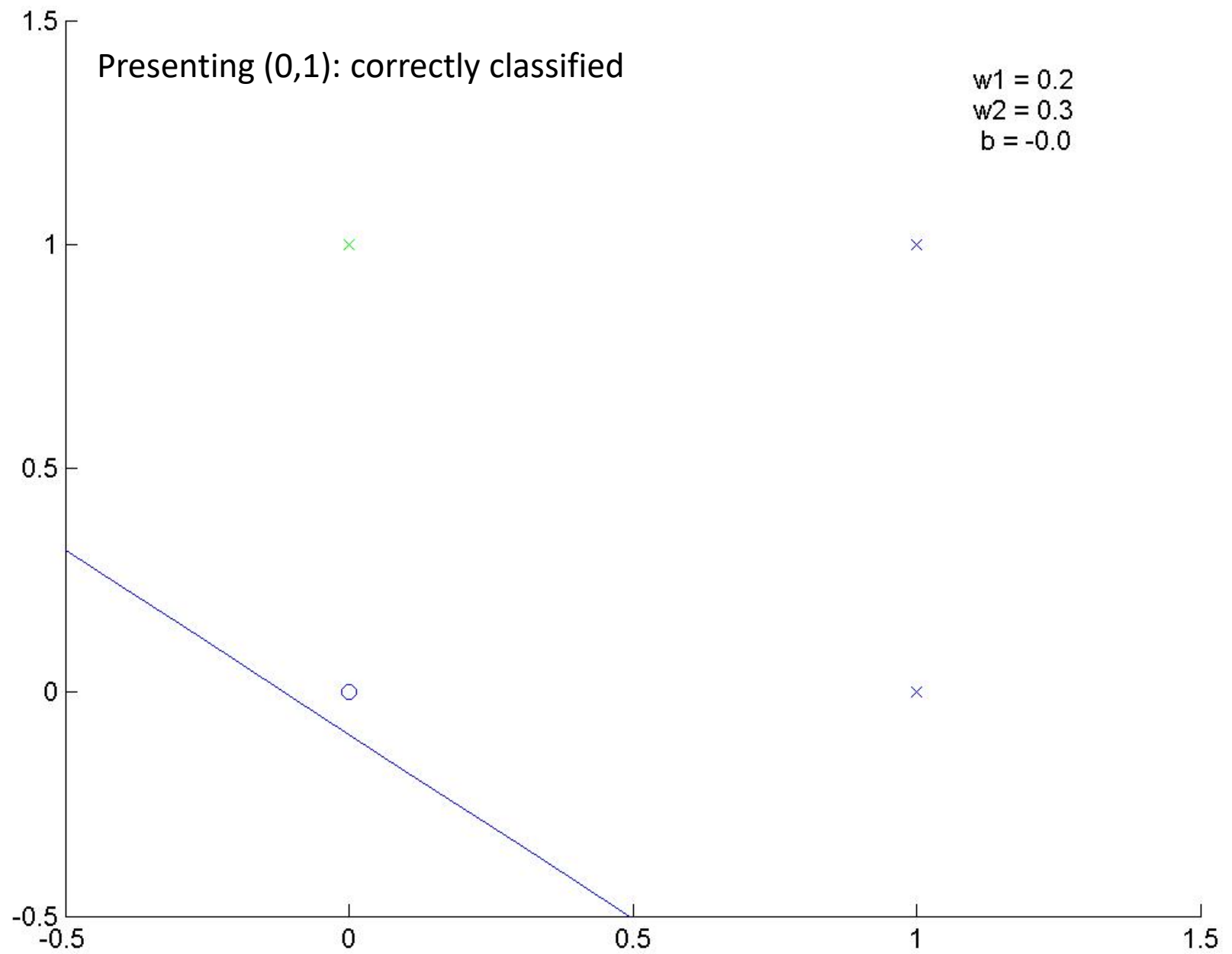
Presenting (0,0): wrongly classified

$$w_1 = 0.2$$

$$w_2 = 0.3$$

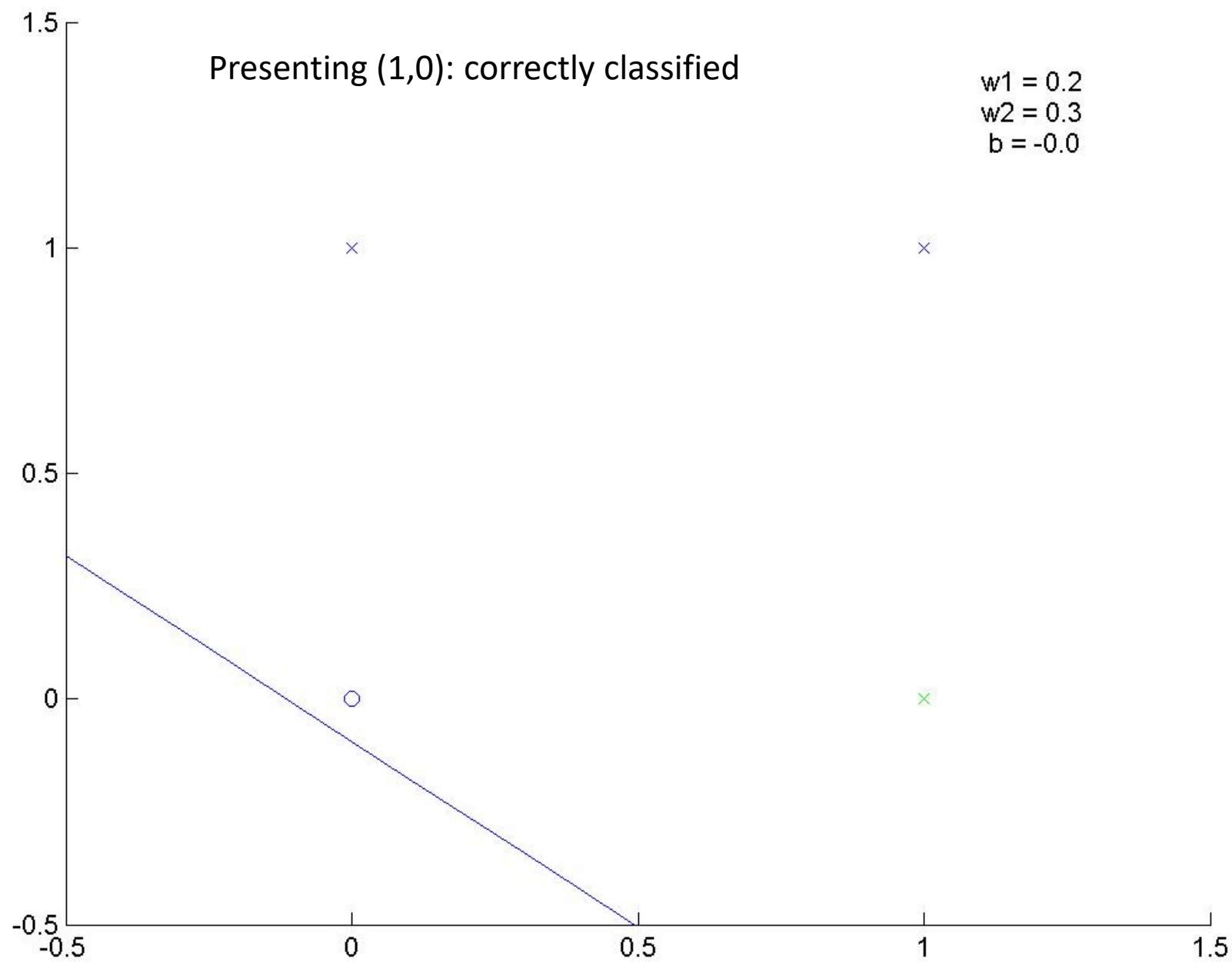
$$b = -0.2$$





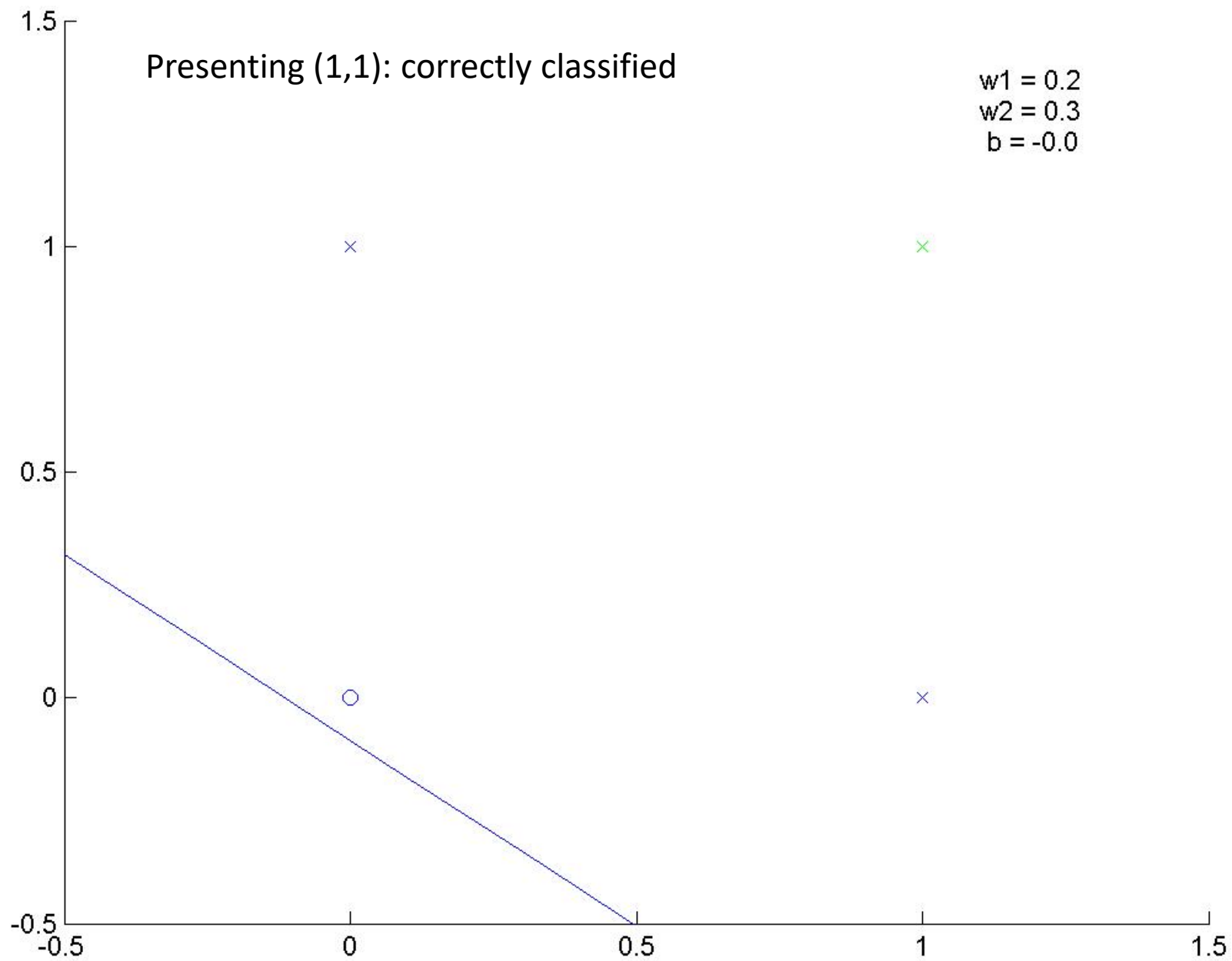
Presenting (1,0): correctly classified

$w_1 = 0.2$
 $w_2 = 0.3$
 $b = -0.0$



Presenting (1,1): correctly classified

$w_1 = 0.2$
 $w_2 = 0.3$
 $b = -0.0$



3 out of 4 correctly classified

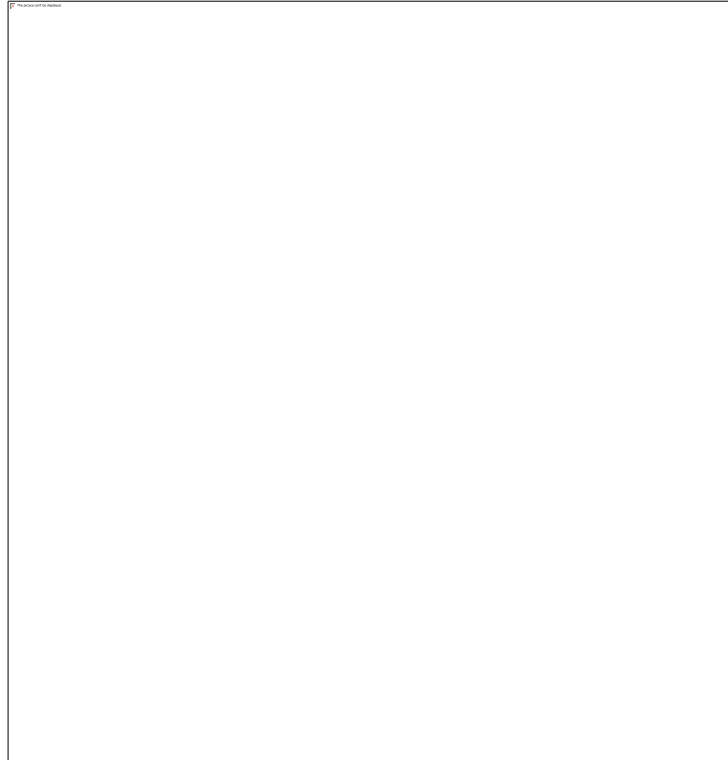
Presenting (0,0): correctly classified

Presenting (0,1): correctly classified

Presenting (1,0): correctly classified

Presenting (1,1): correctly classified

. When the training data is linearly separable, there can be an infinite number of solutions.



- In other words, a Perceptron classifies input patterns by dividing the input space into two semi-infinite regions using a hyperplane.



Perceptron Learning Rule:

- It is also called the LMS Rule or Delta Rule or Widrow-Hoff Rule.
- The steps involved in Perceptron learning are as follows:
 1. Initialization of weights: Set the initial values of the weights to 0. $\mathbf{w}(0) = 0$.
 2. Present the p^{th} training pattern, \mathbf{x} , and calculate the network output, y .

3. Using the desired output, d , for the input pattern x , adjust the weights by a small amount using the following learning rule:

$$w(t+1) = w(t) + \eta[d(t) - y(t)]x(t) \quad \text{--- ①}$$

where,

$$b(t+1) = b(t) - \eta[d(t) - y(t)]$$

$$d(n) = +1, x(t) \in C1$$

$$d(n) = -1, x(t) \in C2$$

--- ②

4. Go back to step 2 and continue until the network output error, $e = d - y$, is 0 for all patterns.

- The above training process will converge after N_{\max} iterations where,

$$\alpha = \min_{x(n) \in C1} w_0^T x(n)$$

$$\beta = \max_{x(k) \in C1} \|x(k)\|^2$$

$$N_{\max} = \beta \|w_0\|^2 / \alpha$$

- See (Haykin 1999, Chapter 3, Section 3.9) for proof of convergence.

- Range of η : $0 < \eta \leq 1$
- Averaging of past inputs leads to stable weight dynamics, which requires small η .
- Fast adaptation requires large η .

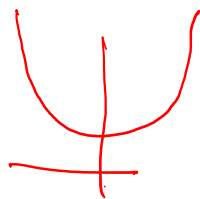
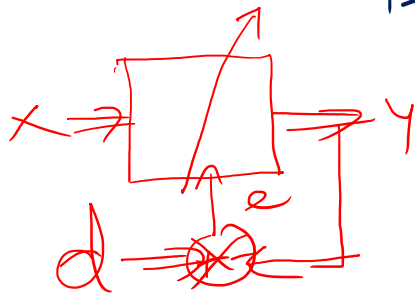
- Learning rule can also be derived from an error function:

$$\Delta w = -\eta \nabla_w E \qquad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\frac{\partial E}{\partial w_i} = -[d - y] \frac{\partial y}{\partial w_i} = -[d - y] g' x_i$$

where E denotes the squared error over all patterns. The learning rule may be derived by performing gradient descent over the error function.

$$\frac{1}{1+e^{-\lambda x}}, \lambda > 1$$



$$\Delta w = -\eta \nabla_w e$$

$$\Delta b = -\eta \frac{\partial e}{\partial b}$$

$$y = g(\underbrace{\sum w_i x_i}_{=h} - b)$$

$$d = +1 \quad x \in C_1$$

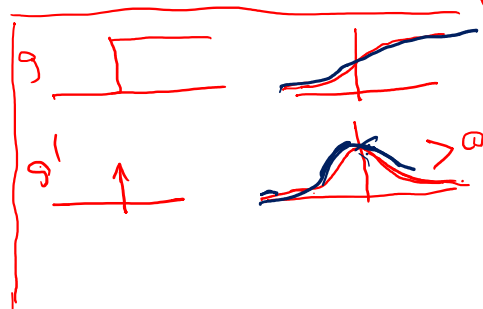
$$= 0 \quad x \in C_2$$

$$E = \frac{1}{2} \sum_p (d_p - y_p)^2$$

$$e_p = \frac{1}{2} (d_p - y_p)^2$$

$$E = \sum_p e_p$$

1



$$\frac{\partial e}{\partial w_i} = \frac{1}{2} \cdot 2 \cdot (d - y) \cdot (-1)$$

$$g'(h) x_i$$

$$= -(d - y) g'(h)$$

$$\checkmark \Delta w_i = \eta (d - y) g'(h) x_i$$

$$\frac{\partial e}{\partial b} = -\frac{1}{2} \cdot 2 (d - y) g'(h) \cdot (-1)$$

$$\checkmark \Delta b = -\eta (d - y) g'(h)$$

$$n' = \eta g'(h)$$

$$\begin{cases} \Delta w_i = n' (d - y) x_i \\ \Delta b = -n' (d - y) \end{cases}$$

- The last term in the above equation has g' , which is zero everywhere except at the origin if g is a hardlimiting nonlinearity.
- But if we take a smoother version of $g()$, which saturate at +1 and -1, like the $\tanh()$ function, the learning rule becomes,

$$\Delta w_i = \eta[d - y]g' x_i$$

- Since $g' > 0$ always for $\tanh()$ function, we can absorb it into h , considering it as a quantity that varies with x . We then have,

$$\Delta w_i = \eta'[d - y]x_i$$

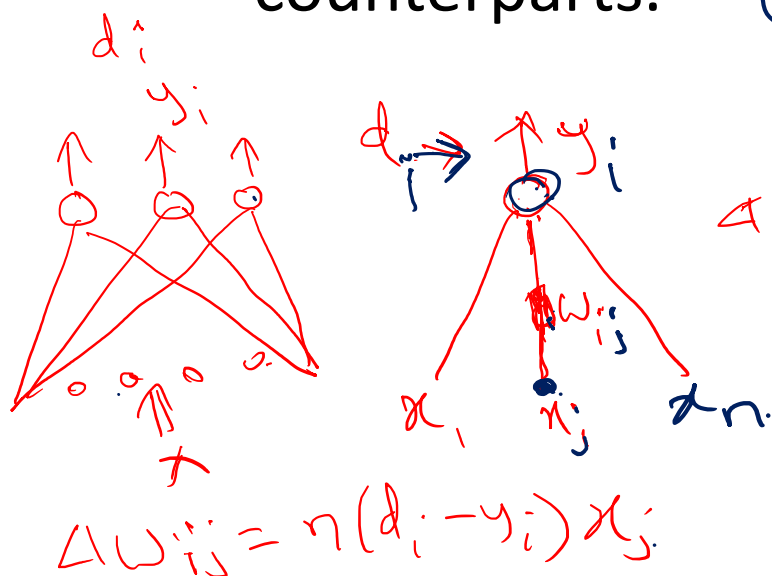
$$\Delta b = -\eta[d - y]$$

Perceptron learning
rule

- Which is identical to the Perceptron learning rule given before above.

A not-very-constructive Criticism

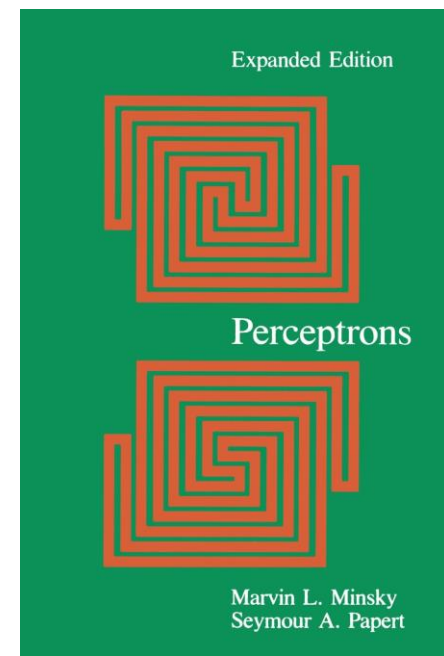
- Minsky and Papert on Perceptron:
 - A single layer perceptron is so limited in its capabilities
 - It may not be worth studying its multilayer counterparts.



$$(d_i - y_i) \rightarrow w_{ij} \leftarrow x_j$$

$$\Delta w_{ij} = \eta (d_i - y_i) x_j$$

locality principle

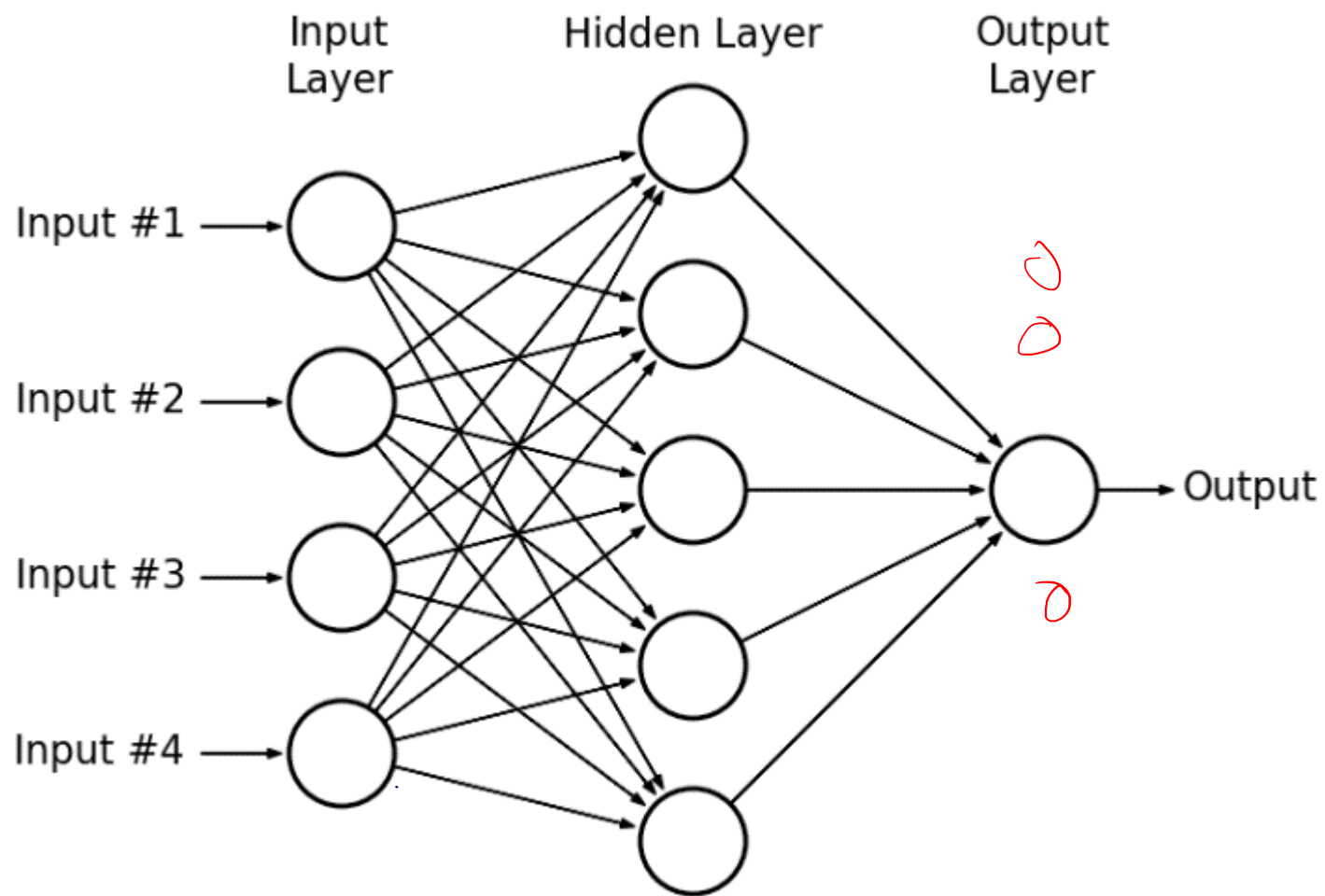


The Multi-layered Perceptron

- Improvements over Perceptron:

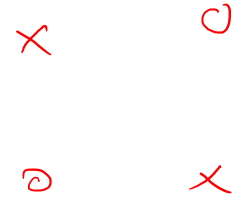
- Smooth nonlinearity – sigmoid
- 1 or more hidden layers

~~logistic~~ (1/0)
tanh (+1/-1)



The Hidden Layer

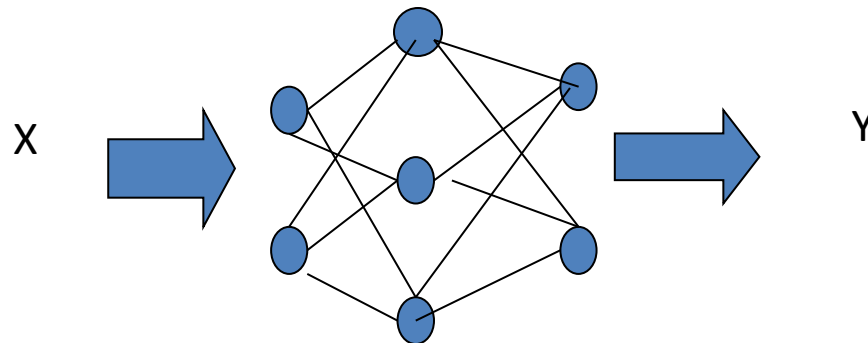
- The perceptron, which has no hidden layers, can classify only linearly separable patterns.
- The MLP, with at least 1 hidden layer can classify any linearly non-separable classes also.



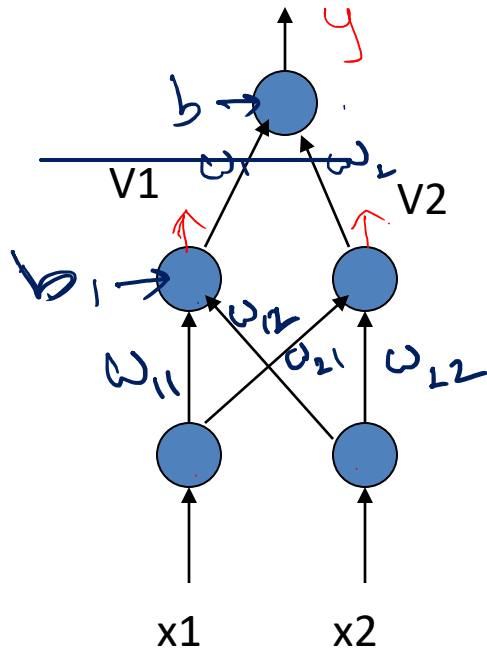
- It can easily be shown that the XOR problem which was not solvable by a Perceptron can be solved by a MLP with a single hidden layer containing two neurons.

Multi Layer Perceptron (MLP)

- Neurons are organized as layers
 - Input, output and hidden layer(s)
- Feedforward NN: Flow of info from input to output layer



MLP solves the EXOR problem



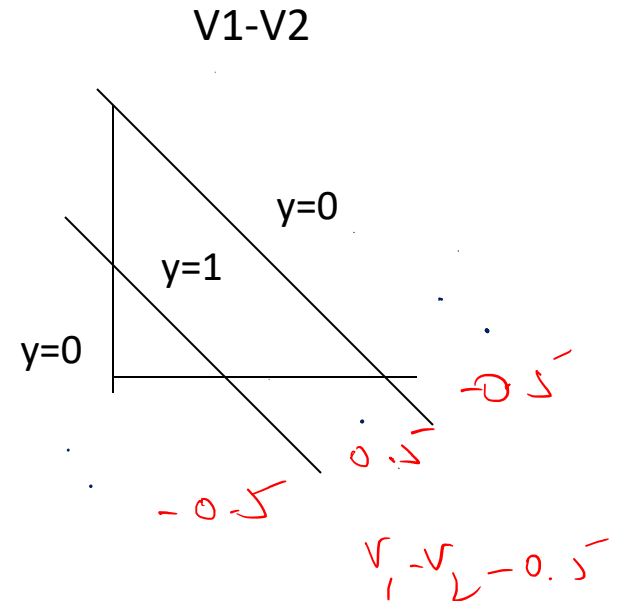
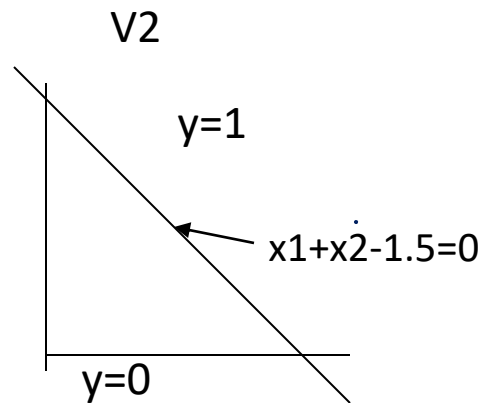
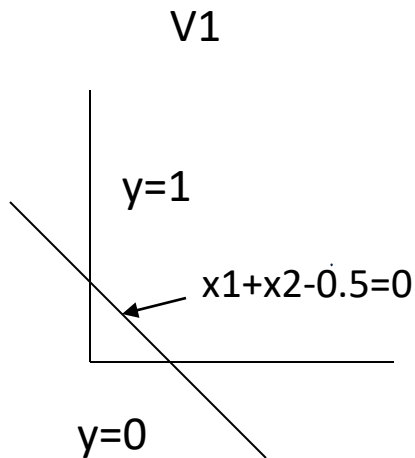
$$V_1 = g(x_1 + x_2 - 0.5)$$

$$V_2 = g(x_1 + x_2 - 1.5)$$

$$y = g(V_1 - V_2 - 0.5)$$

$g \equiv \text{Heaviside}$

MLP can model an XOR gate



$$y = g(V_1 - V_2 - 0.5)$$

XOR gate!

MLP

- An MLP can approximate any continuous multivariate function to any degree of accuracy, provided there are sufficiently many hidden neurons (Cybenko, 1988; Hornik et al, 1989). A more precise formulation is given below.
- A serious limitation disappears suddenly by adding a single hidden layer.

All-powerful MLP

- MLP is a *universal approximator*
- “An MLP with a single hidden layer can approximate any continuous I/O function with arbitrary accuracy, over a finite input domain, given enough number of nodes in the hidden layer.” (Cybenko, 1988)

Taylor series

$$f(x_0 + h) = f(x_0) + h f'(x_0) +$$

Approximation theory

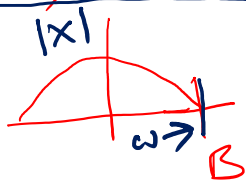
$$x(t) = \sum w_i \phi_i(t)$$

Fourier series

$$s(t) = s(t+T) \quad \forall t$$

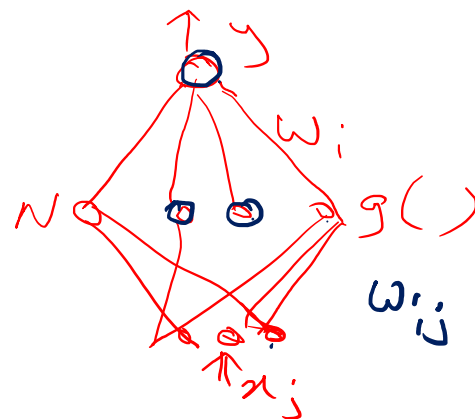
$$s(t) = a_0 + \sum_{n=1}^{\infty} C_n \cos(n\omega_0 t + \phi_n)$$

Sampling theorem



S.F. $> 2B$ $\phi(x) = \frac{\sin x}{x}$

$$s(t) = \sum w_i \phi_i(x)$$



$$y = \sum_i w_i g(\sum_j w_{ij} x_j - b_i)$$

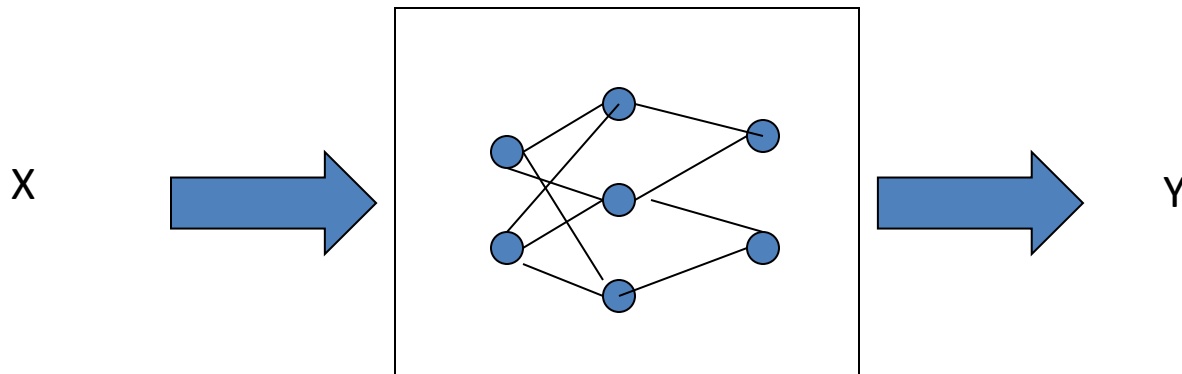
$$= \sum_{i=1}^N w_i g(w_i \cdot x - b_i)$$

$$x \rightarrow y$$

$$\in \mathbb{R}^n \quad \in \mathbb{R}$$

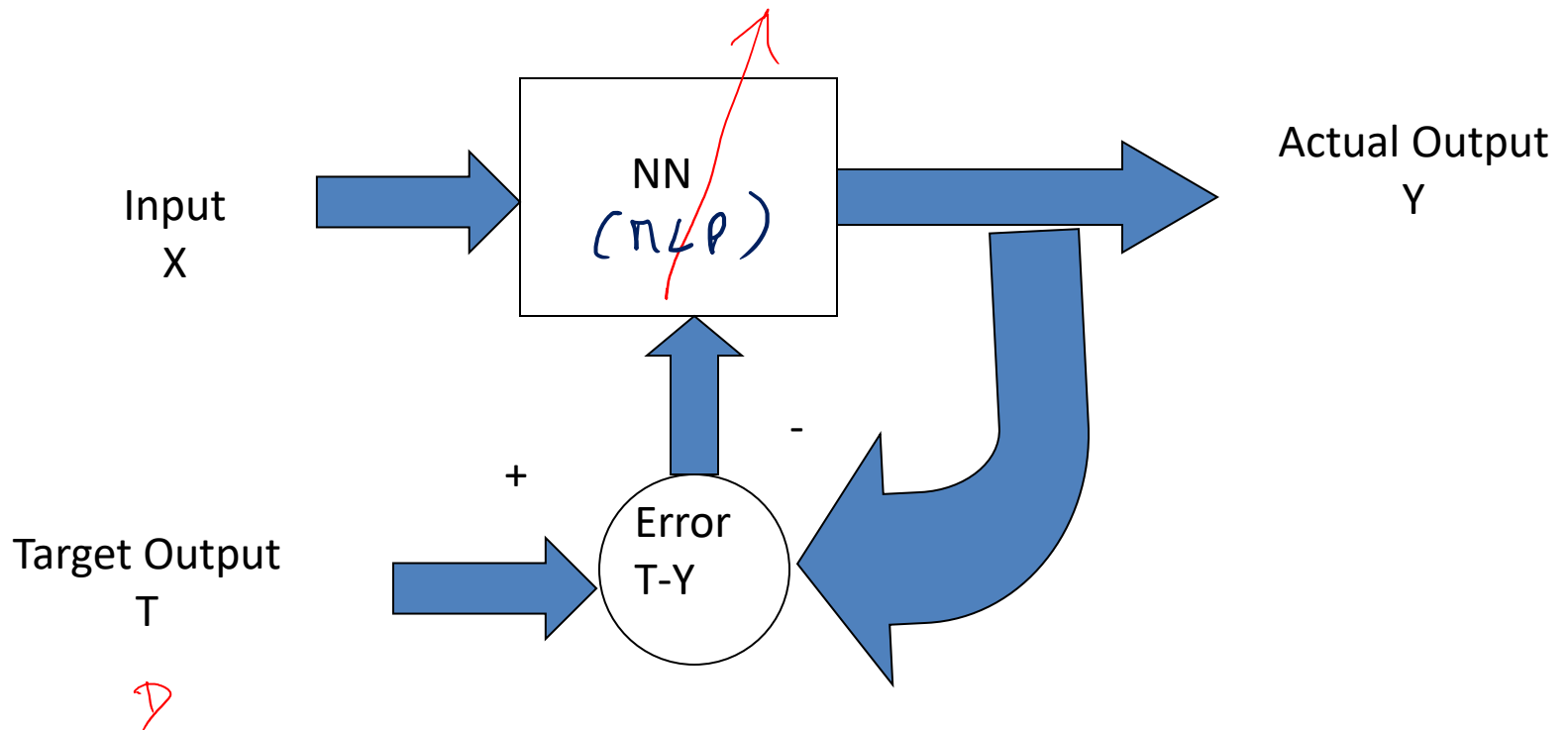
NN as an I/O Machine

- NN is a parametric model that can model arbitrary I/O relationships



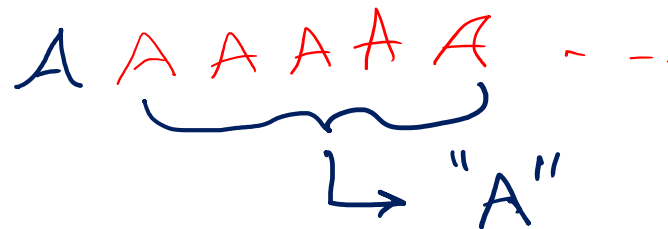
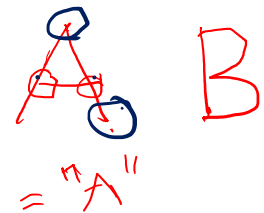
Learns from Experience

- NN can learn the unknown I/O relationship from sample data



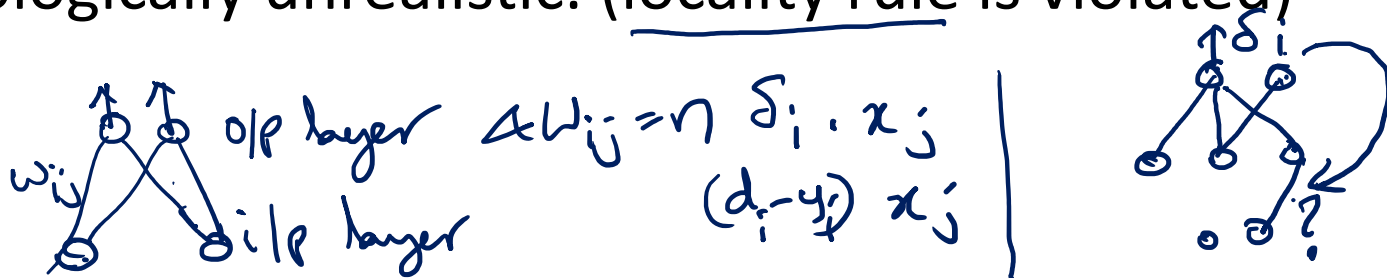
NN vs. Algorithmic Approach

- Traditional or Algorithmic Approach
 - Study and find out the steps to produce a final output from a given input
- NN Approach:
 - Learn from Experience



Training the hidden layer

- Not obvious how to train the hidden layer parameters.
- The error term is meaningful only to the weights connected to the output layer – credit assignment problem.
- In a large network with many layers, this implies that information is exchanged over distant elements of the network though they are not directly connected.
- Such an algorithm may be mathematically valid, but is biologically unrealistic. (locality rule is violated)



The Backpropagation Algorithm

- **History:**
- First described by Paul Werbos (1974) in his PhD thesis at MIT.
- Rediscovered by Rumelhart, McClelland and Williams (1986)
- Also discovered by Parker (1985) and LeCun (1985) in the same year.
- As in Perceptron, this training algorithm involves 2 passes:
 - The forward pass – outputs of various layers are computed
 - The backward pass – weight corrections are computed

Back Propagation Algorithm

- Consider a simple 3-layer network with a single neuron in each layer.

Total output error over all patterns: $E = \sum_p E_p$

Squared Output error for the p'th pattern: $E_p = \frac{1}{2} \sum_i e_i^2$

Output error for the p'th pattern: $e_i = d_i - y_i$

Network output: $y_i = g(h_i^s)$

Net input to the output layer: $h_i^s = \sum_j w_{ij}^s V_j - \theta_i^s$

Output of the hidden layer: $V_j^f = g(h_j^f)$

Net input of the hidden layer: $h_j^f = \sum_k w_{jk}^f x_k - \theta_j^f$

$$E = \sum_p E_p$$

$$E_{\text{PBH}} = \frac{1}{2} \sum_i^{n_0} (d_i - y_i)^2$$

$$\Delta w_{ij}^s = ?$$

$$\Delta w_{jk}^f = ?$$

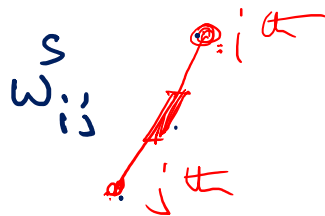
C.D

$$\Delta w = -\eta \frac{\partial E}{\partial w}$$

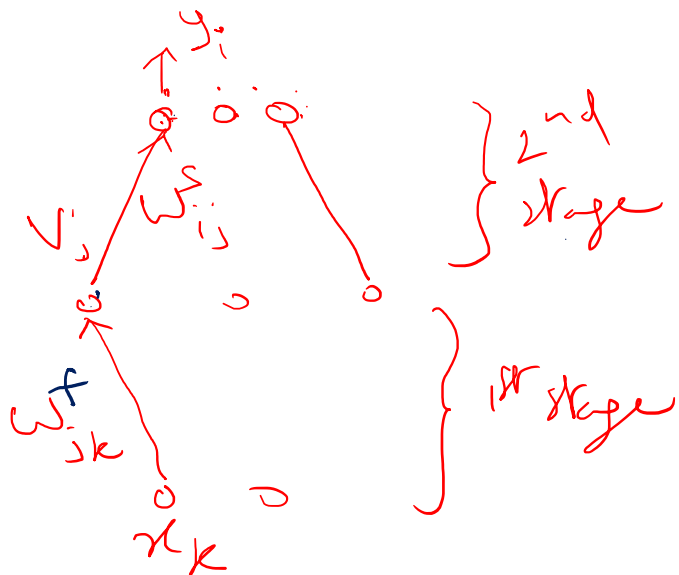
$$\Delta w_{ij}^s = -\eta \frac{\partial E_p}{\partial w_{ij}^s}$$

$$\frac{\partial E_p}{\partial w_{ij}^s} = \frac{1}{2} \cdot 2 \cdot (d_i - y_i) (-1) g'(h_i^s) V_j$$

$$\Delta w_{ij}^s = \eta (d_i - y_i) g'(h_i^s) V_j \quad \checkmark$$



locality principle

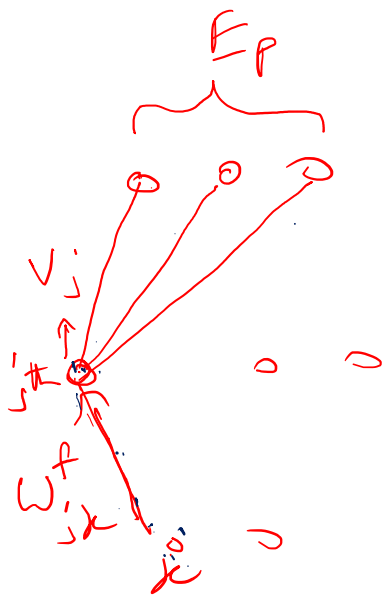


$$y_i = g(h_i^s)$$

$$h_i^s = \sum_j w_{ij}^s v_j$$

$$v_j = g(h_j^f)$$

$$h_j^f = \sum_k w_{jk}^f x_k$$



$$\Delta w_{jk}^f = -\eta \frac{\partial E_P}{\partial w_{jk}^f}$$

$$\frac{\partial E_P}{\partial w_{jk}^f} = \frac{1}{2} \cdot (2) \sum_i (d_i - y_i) (-1) g'(h_i^s) w_{ij}^s \frac{\partial V_j}{\partial w_{jk}^f}$$

$$\left(\frac{\partial V_j}{\partial w_{jk}^f} = g'(h_j^f) x_k \right) \rightarrow$$

$$= \sum_i (d_i - y_i) (-1) g'(h_i^s) w_{ij}^s g'(h_j^f) x_k$$

violates locality principle

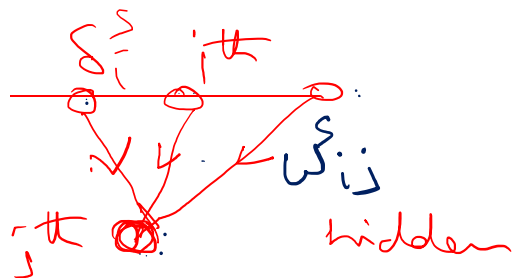
$$\underline{w_{jk}^f} \rightarrow V_j \rightarrow y_i \rightarrow E_P$$

Define δ 's (error terms)

$$\boxed{\Delta w_{ij}^s = \eta \delta_i^s y_j} \quad \checkmark$$

① ~~output layer~~

$$\delta_i^s = (d_i - y_i) g'(h_i^s) \quad \checkmark \quad \Delta w_{jk}^f = \eta \sum_i (d_i - y_i) g'(h_i^s)$$



$$w_{ij}^s g'(h_j^f) x_k$$

$$= \eta \left(\sum_i \delta_i^s w_{ij}^s \right) g'(h_j^f) x_k$$

$$\delta_j^f = \left(\sum_i \delta_i^s w_{ij}^s \right) g'(h_j^f)$$

$$\boxed{= \eta \delta_j^f x_k} \quad \checkmark$$



BPA

- Update rule for the weights using gradient descent:

$$\begin{aligned}\Delta w_{ij}^s &= -\eta \frac{\partial E_p}{\partial w_{ij}^s}; & \Delta \theta_i^s &= -\eta \frac{\partial E_p}{\partial \theta_i^s} \\ \Delta w_{jk}^f &= -\eta \frac{\partial E_p}{\partial w_{jk}^f}; & \Delta \theta_j^f &= -\eta \frac{\partial E_p}{\partial \theta_j^f}\end{aligned}$$

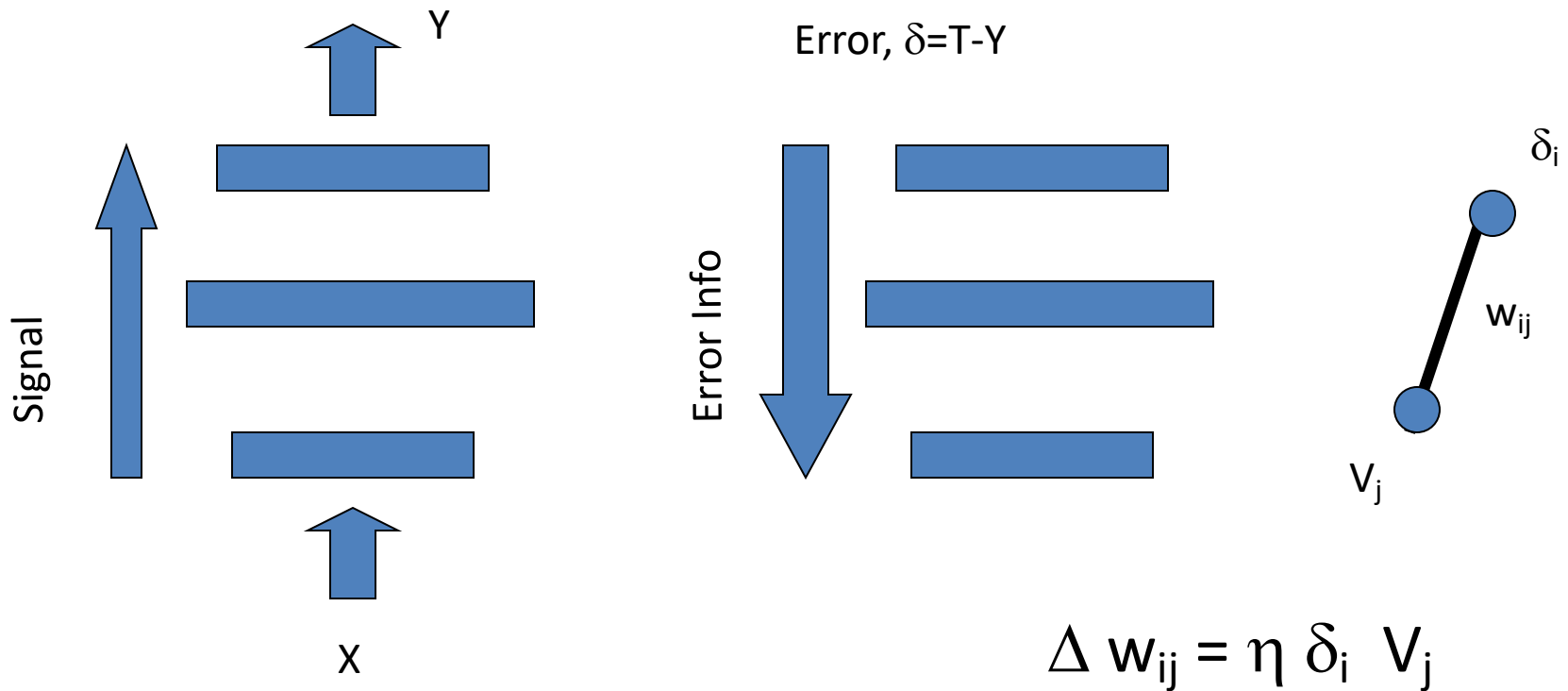
Updating w_{ij}^s :

$$\begin{aligned}\frac{\partial E_p}{\partial w_{ij}^s} &= \frac{\partial E_p}{\partial e_i} \frac{\partial e_i}{\partial y_i} \frac{\partial y_i}{\partial h_i^s} \frac{\partial h_i^s}{\partial w_{ij}^s} \\ &= e(-1)g'(h_i^s)V_j\end{aligned}$$

- **weight correction = (learning rate) * (local δ from 'top') * (activation from 'bottom')**

Training MLP

- **Backpropagation**



Training and Testing a MLP

Traning:

- Randomly initialize weights.
- Train network using backprop eqns.
- Stop training when error is sufficiently low and freeze the weights.

Testing

- Start using the network.

Merits of MLP

Merits of MLP trained by BP:

- A general solution to a large class of problems.
- With sufficient number of hidden layer nodes, MLP can approximate arbitrary target functions.
- Backprop applies for arbitrary number of layers, partial connectivity (no loops).
- Training is local both in time and space – parallel implementation made easy.
- Hidden units act as “feature detectors.”
- Good when no model is available

Demerits of MLP by BPA

Problems with MLP trained by BP:

- Blackbox approach
- Limits of generalization not clear
- Hard to incorporate prior knowledge of the model into the network
- slow training
- local minima

Development of an NN Application

- The Stages
 - Feasibility study
 - Data collection
 - Designing/Training the Network
 - Testing the Network
 - Deployment...

Feasibility Study

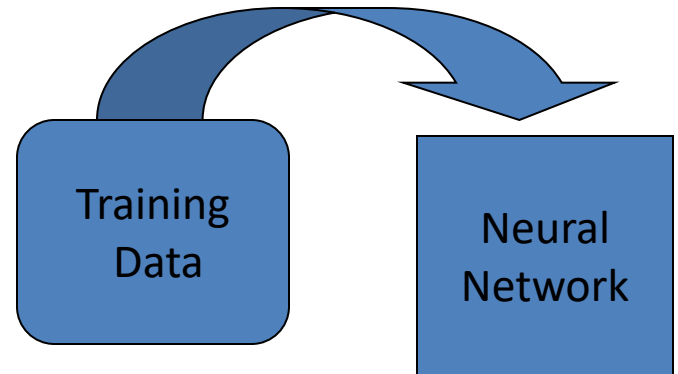
- Do you really need a NN?
- Is it an I/O problem?
- What are the inputs and the outputs?
- Are inputs and outputs uniquely related?
- Do I have enough domain knowledge?
- Do I have enough Data?

Domain Knowledge

- Can never overemphasize the value of Domain Knowledge
- Domain Knowledge is used to -
 - Breaking up the problem
 - Map problem onto NN(s)
 - Data collection
 - Data preprocessing

Data

- Knowledge in Data ==> Knowledge in NN
- Data must be-
 - Abundant
 - real data
 - simulated data
 - Comprehensive
 - include all possible cases
 - Reliable
 - Right **representation** is the key



Data Preprocessing

- Representation
 - Ex: 1) odd/even discr., 2) linkage problem
 - Remove irrelevant or redundant information
 - Normalization, scaling,...
 - Reduce input dimension
 - Principal component analysis
 - Feature extraction

Designing/Training the Network

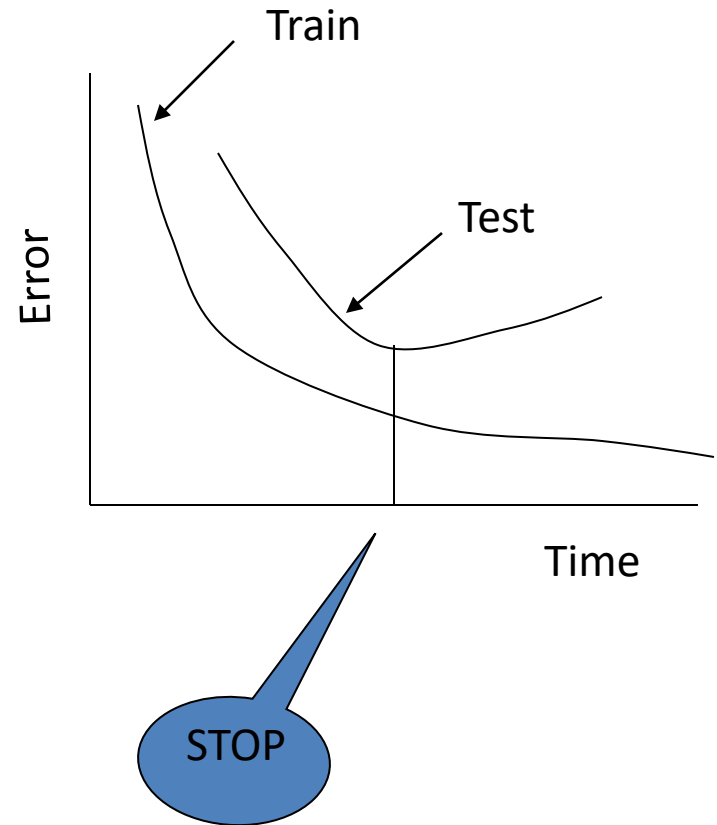
- Architecture
- Initialization
- Training

Architecture

- Feedforward Network: MLP, RBF Network, Hybrid,...?
- MLP:
 - 1 hidden layer?
 - 2 hidden layers?
 - hidden layers > 2 ?
 - Customized architecture?

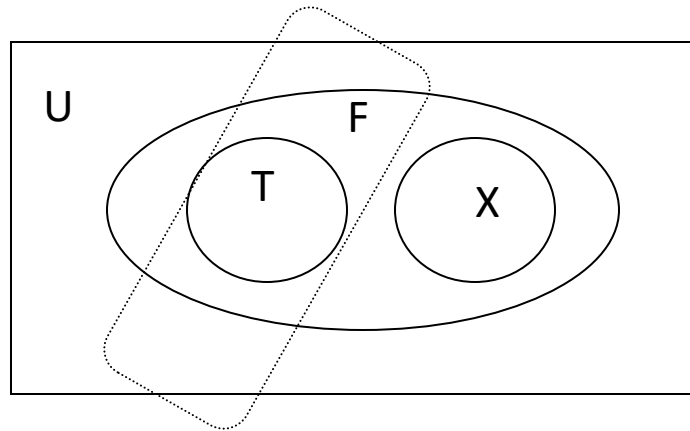
Training & Pruning

- Training procedure
 - Backpropagation
 - adaptive learning rate
 - momentum,...
 - Higher-order methods
- When to stop training?



Generalization & Testing

- The problem of generalization



- Cross-validation

Caution: Specify explicitly the conditions under which the network was trained

Pruning

- weight decay
- Optimal Brain Damage (OBD)

APPLICATIONS

Past tense learning

Ref: Rumelhart and McClelland, On learning
the past tenses of English verbs, Parallel
Distributed Processing, Vol. II, 1986

Three stages of past tense learning is seen in children.

Stage 1:- Only a small number of words used correctly in past tense

- High frequency words, majority are irregular
- Children tend to get the past tense quickly.
- Typical examples: Came, got, gave, looked, needed, took, went.

regular: → ed ending

Stage 2:-

Children use much larger number of words

Many verbs are used with correct past tense forms

Majority are regular

Eg. Wiped, pulled

Children now incorrectly apply regular past tense endings for words which they used in stage 1

Eg: come comed or camed

Stage 3:-

Regular or irregular forms exist.

Children have acquired the use of correct irregular form of past tense

But continue to apply the regular form to new words they learn.

Training

10- high frequency verbs (8 irregular + 2 regular): Eg: Came, get, give,, take, go, have, feel, look, live

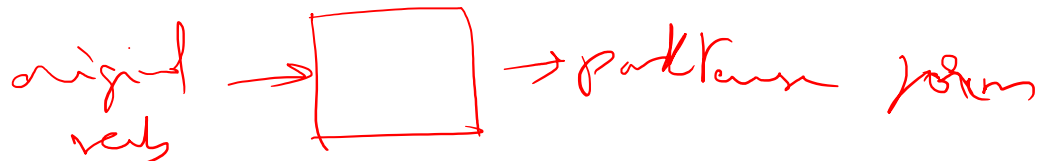
410- medium frequency verbs, 334 regular, 76 irregular

86-low frequency words, 72 regular, 14 irregular

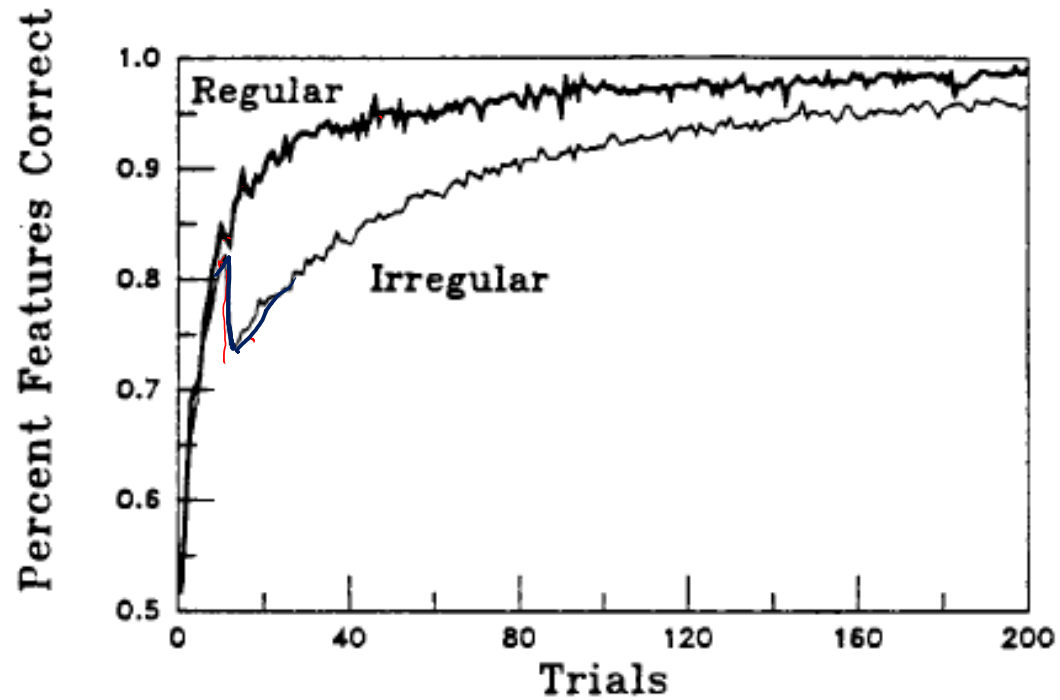
Stage 1:- 10 epochs of high frequency verbs. (Enough for good performance)

Stage 2:- 410 medium frequency verbs were added to 10 verbs (trained for 190 more epochs after phase 2). In this stage errors suddenly start creeping in. Most of the errors are due to regularization of irregular verbs

Stage 3:- 86 low frequency verbs are tested without training. Beyond state 2, the performance over regulars and irregulars is nearly the same, each touching 100%.



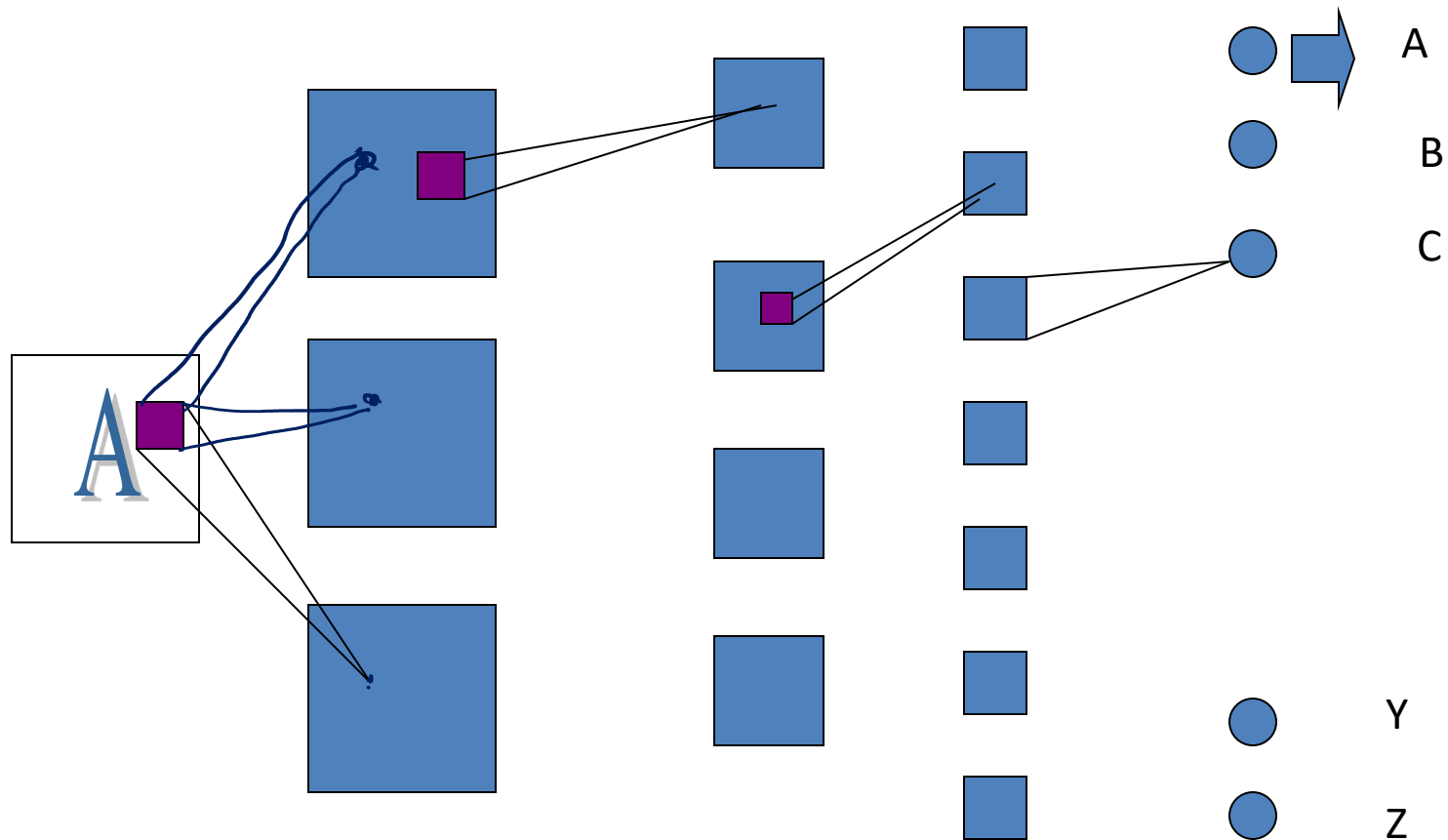
Performance



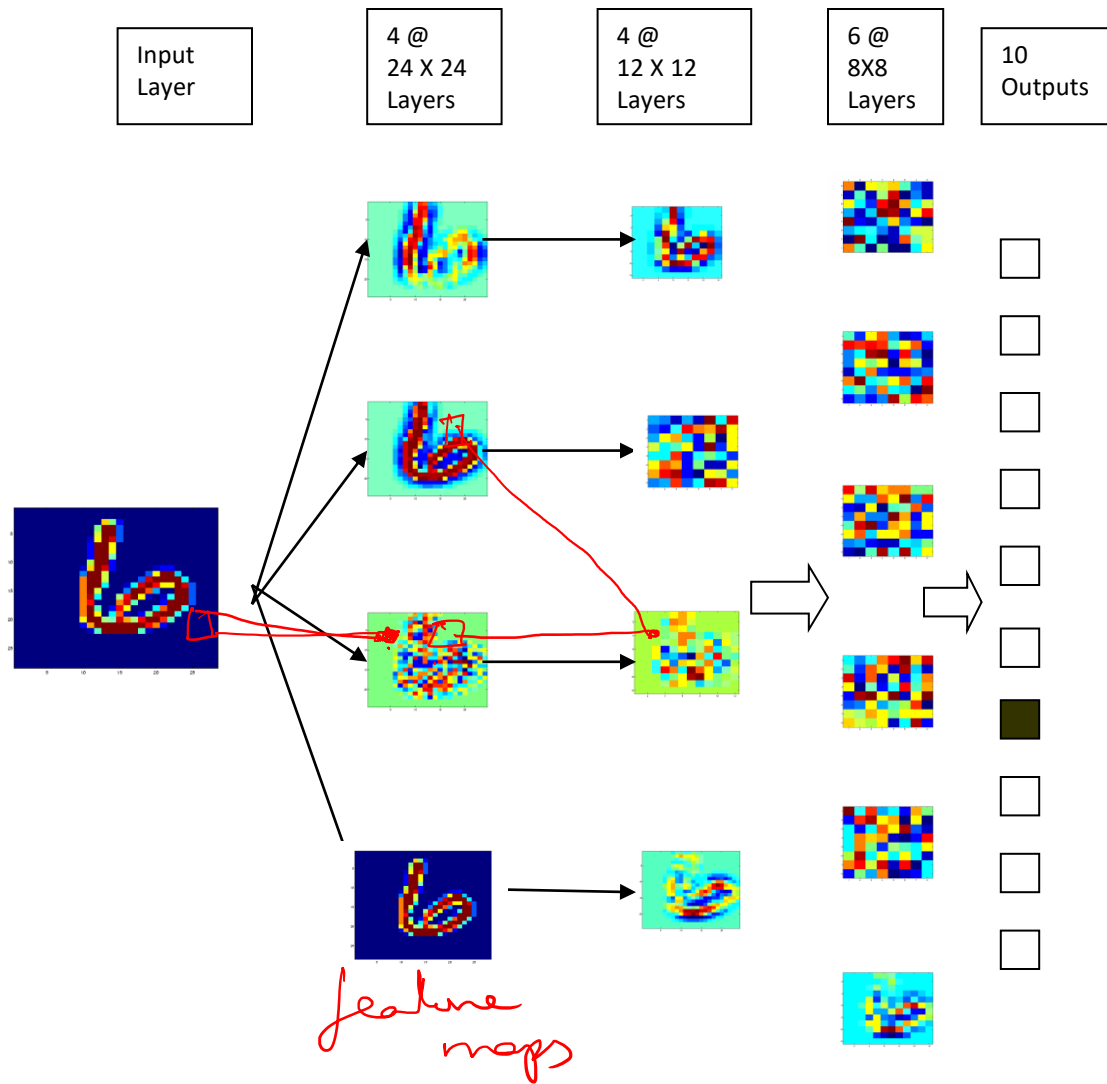
Handwritten Numeral Recognition using Convolutional Networks

- Lenet: Developed at AT & T to read zip codes on US postal envelopes
- Inspired by biological visual system
- Speed: 10 chars/sec to 1000 char/sec
- 99% accuracy on test data at a rejection rate of 12%.

Convolutional Neural Network for Character Recognition



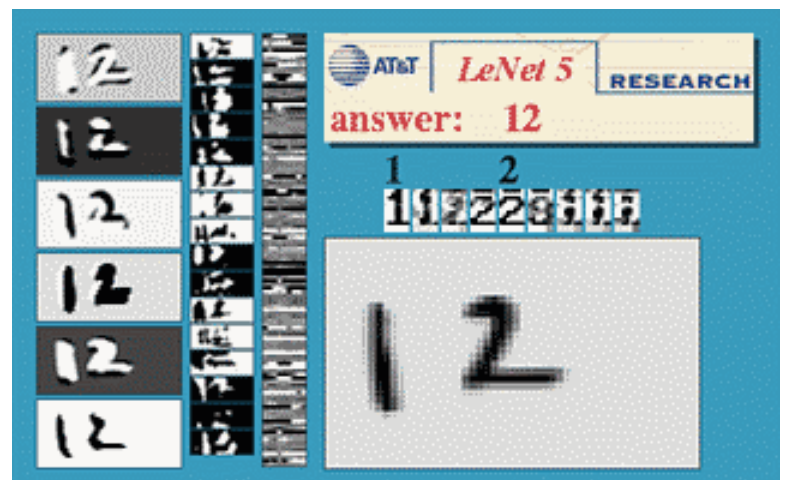
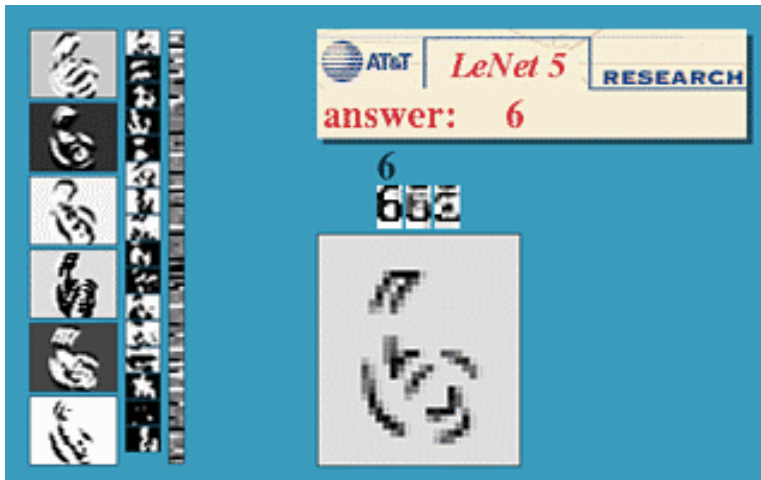
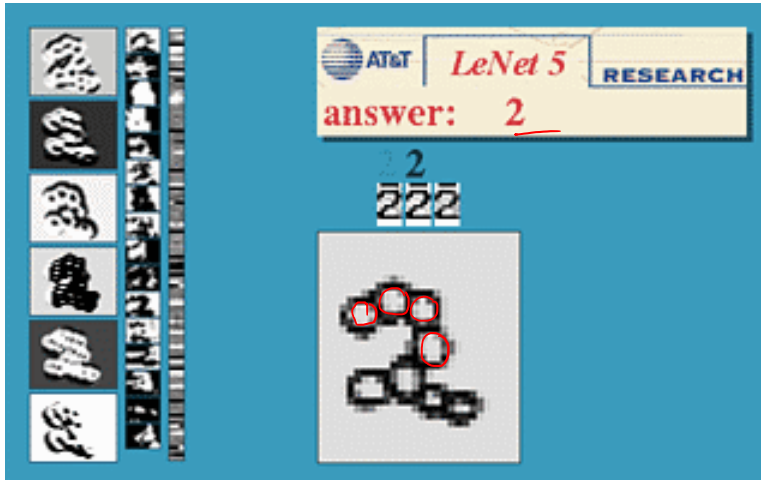
NEURAL NETWORK RECOGNIZES NUMERALS

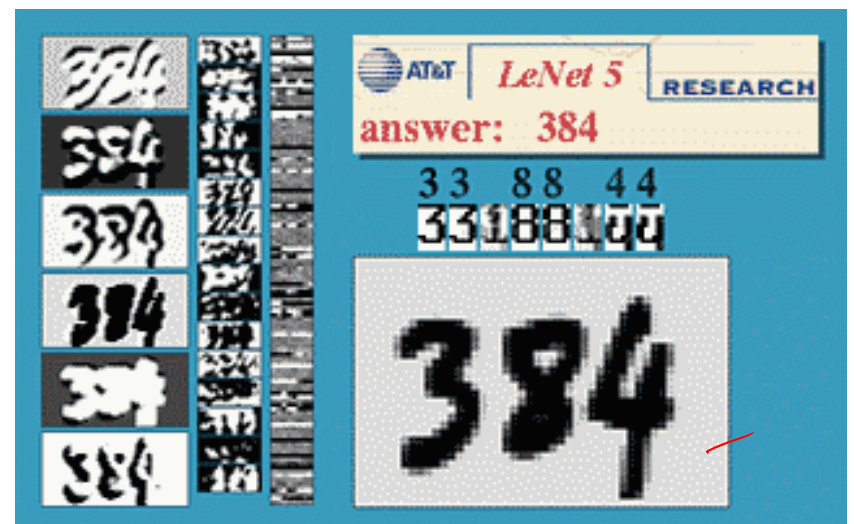
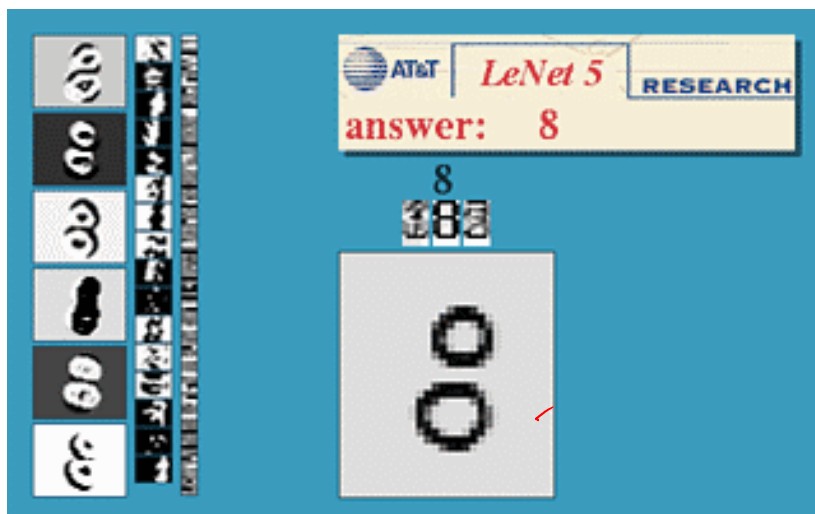
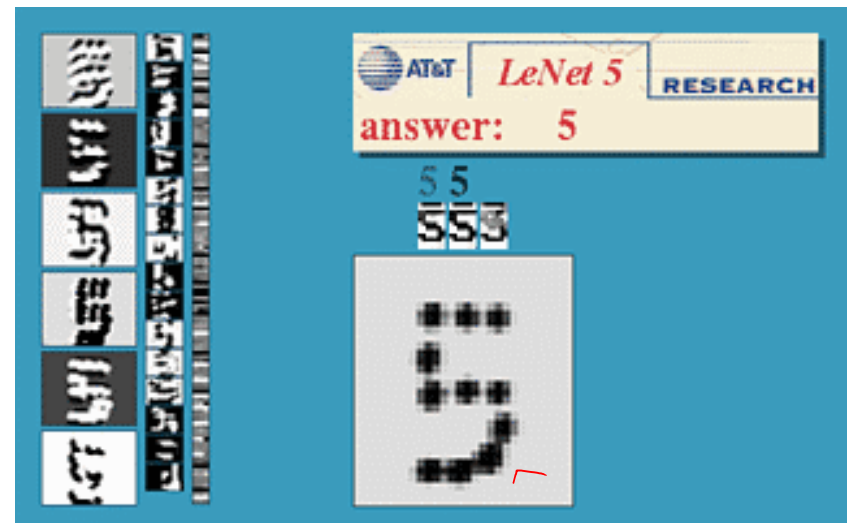


Handwritten red text: 0
1
2
3

MNIST dataset

2
4





NetTalk: A neural network that can read text aloud

Ref: Sejnowski, TJ Rosenberg, CR (1986).

NETtalk: A parallel network that learns to read
aloud, Tech. Rep. No. JHU/EECS-86/Q1

Nettalk is a system that can read English text aloud and can pronounce letters accurately based on context (Sejnowski and Rosenberg 1986).

Background:

English is not a phonetic language.

Char → Sound mapping is not unique (different depending on context).

The character 'c' is pronounced as /k/ in "cat" and as /s/ in "façade."

The letters "-ave" form a long vowel in "gave" and "brave" but not in "have."

Similarly, the letters "-ea-" are pronounced as /ii/ (long vowel) in "read" (present tense) (pronounced as "reed") and as /i/ (short vowel) as in "read" (past tense) (pronounced as "red").

Network

It uses a three layer MLP.

Input representation:

26 alphabets and three punctuation marks (comma, fullstop and blank space) are supported

Each character is presented along with a context which consists of three additional characters on either side of the character.

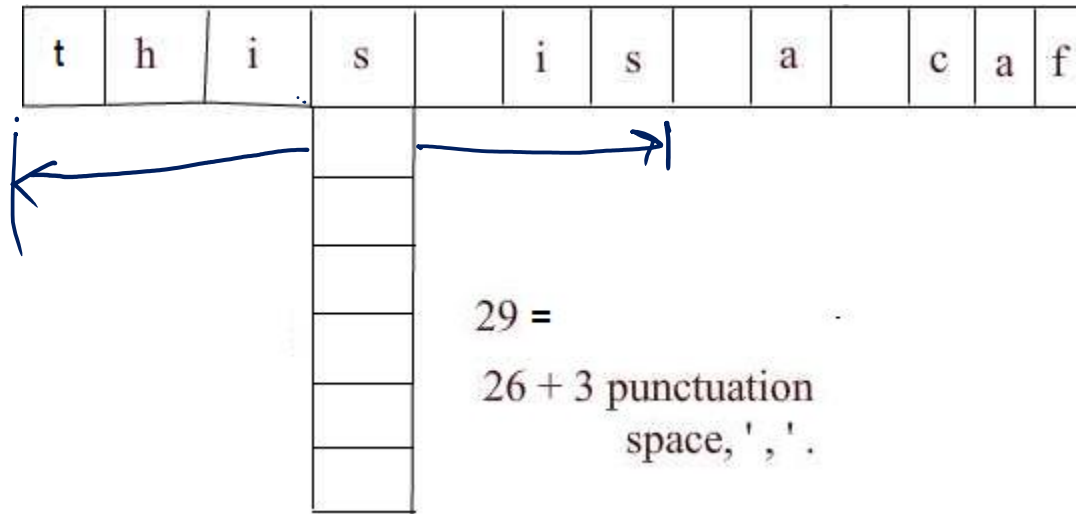
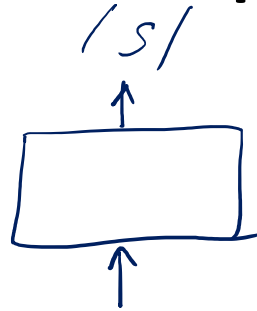
Thus text is presented as windows of 7 symbols

Hence, $(26 + 3) * 7$ input neurons

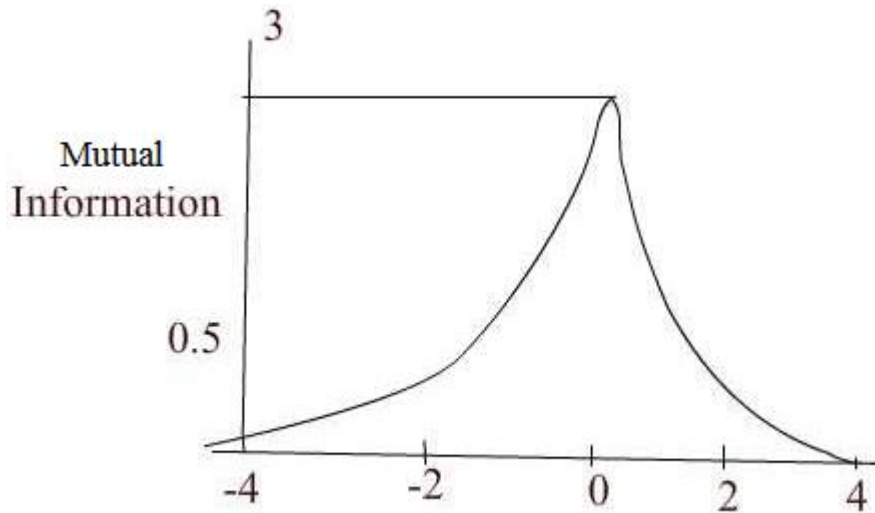
Hidden layer : 80 hidden neurons

And Output neurons: 26 (encoding phonemes)

Input representation



Network architecture



- By a window of 7 characters, mutual information from a central character falls off rapidly in both directions as shown in the above figure.
- Beyond 3 neighbors mutual information falls to less than 0.1.

Ref: Hetz et al, 1991

Training data

Phonetic transcription from informal continuous speech of a child

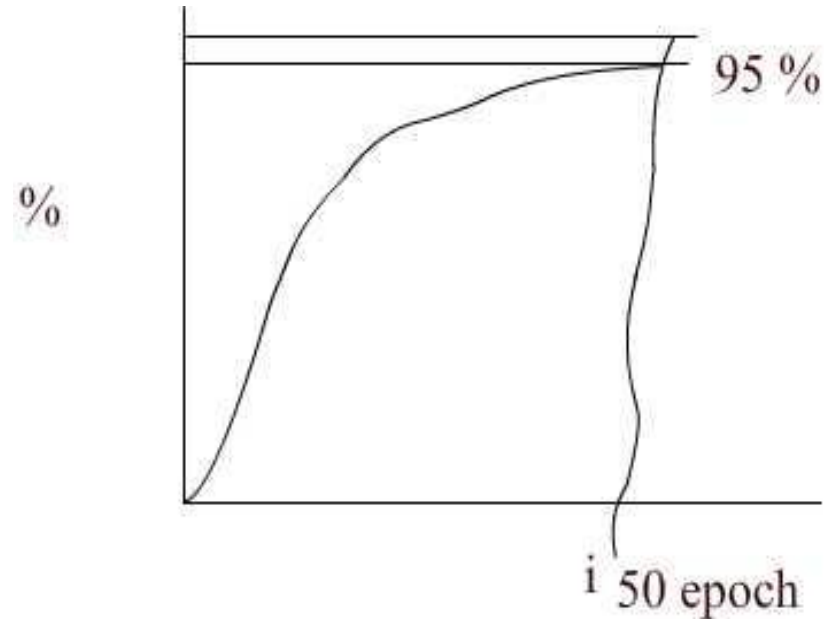
20,000 words from a pocket dictionary

Eg: phone f – o n –

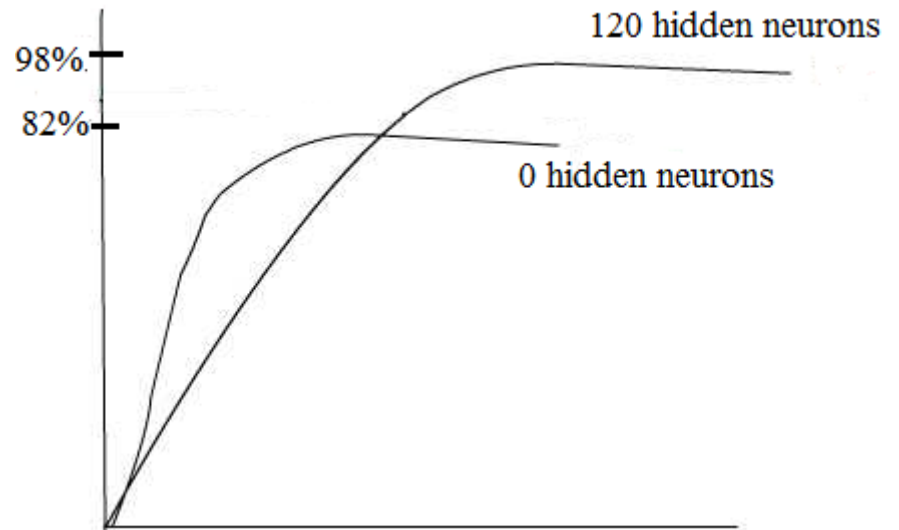
A set of 1000 words were chosen from this dictionary; these are selected from the Brown corpus of the most frequent words in English.

Learning curve

- Found in human skill learning
- Performance reaching about 95% correct after training the network with about 50,000 words.



- The presence of a hidden layer is found to be crucial to achieve this high level of performance. When a two-layer network (perceptron) was used for the same problem, performance quickly rose to 82% and saturated at that level.



Stages of learning

- One of the first features learnt by the network is distinction between vowels and consonants
- Next the network learnt to pause at word boundaries. Output resembled pseudo words with pauses between them.
- After 10 passes text was understandable

Error patterns

Errors were meaningful. e.g thesis, these

The network rarely confused between vowels & consonants.
Some errors actually were due to errors in annotation.

Test Performance

- 439 words from the regular speech. These words were not present in the training set.
- Performance was at 78%. Thus the network demonstrated that it can generalize from one set of words to another.

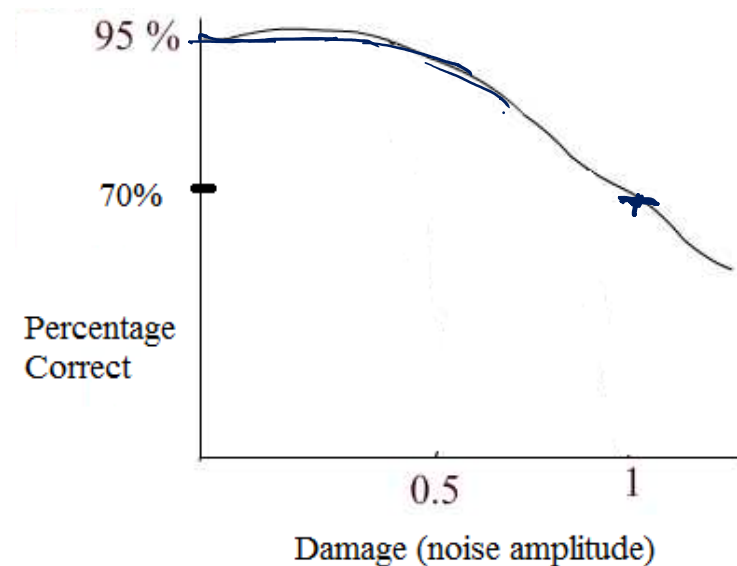
Damage testing

- In order to test the robustness of the network in face of damage, random noise, n , uniformly distributed between -0.5 and 0.5 was added to each weight in the trained network.

$$\omega_i \leftarrow \omega_i \pm \nu$$

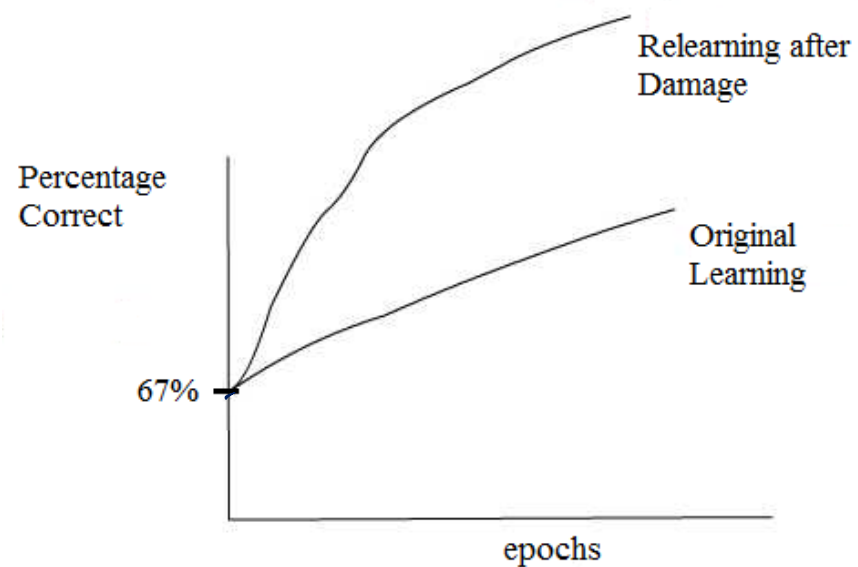
$$\nu \in [-0.5, 0.5]$$

- The damaged network showed 67% performance.

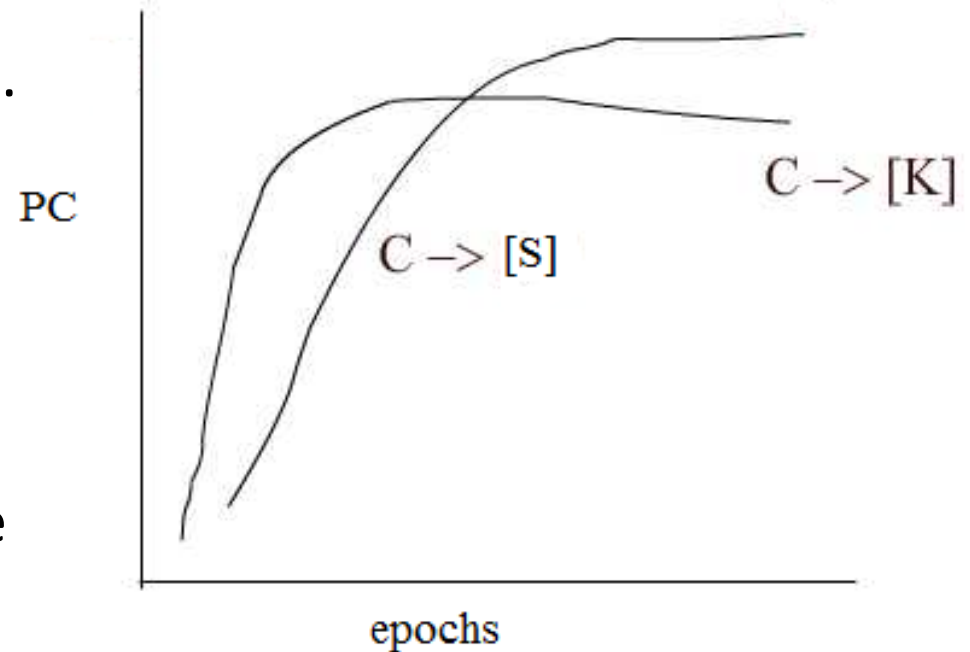


Relearning after Damage

- Although the damaged network begins with the same performance level as the original network, it relearns the task much faster than the original network.



- Specific letter-to-phoneme correspondences showed that different correspondences were learnt at different rates.
- Soft 'c' (as in 'nice') takes longer to learn than hard 'c' (as in 'cat'). This is probably because hard c occurs about twice as often as soft c in the corpus.



Interestingly, children show a similar difficulty in learning.