

The background of the slide features a complex, glowing neural network. It consists of numerous interconnected nodes, some of which are large and brightly lit with a yellow-orange glow, while others are smaller and dimmer. The nodes are connected by a web of thin, light-colored lines that crisscross the dark background. The overall effect is one of dynamic energy and connectivity.

Neural Networks

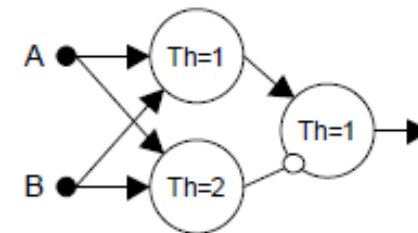
(Incl. Perceptrons)

CS 6316 – Machine Learning
Fall 2017

A brief history of ANNs

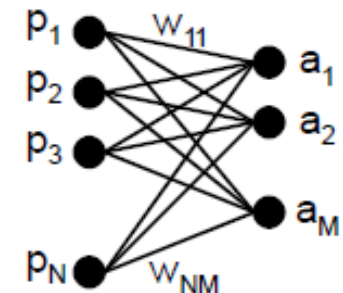
McCulloch and Pitts, 1943

- The modern era of ANNs starts in the 1940's, when Warren McCulloch (a psychiatrist and neuroanatomist) and Walter Pitts (a mathematician) explored the computational capabilities of networks made of very **simple neurons**
- A McCulloch-Pitts network fires if the sum of the excitatory inputs exceeds the threshold, as long as it does not receive an inhibitory input
- Using a network of such neurons, they showed that it was possible to construct any logical function



Hebb, 1949

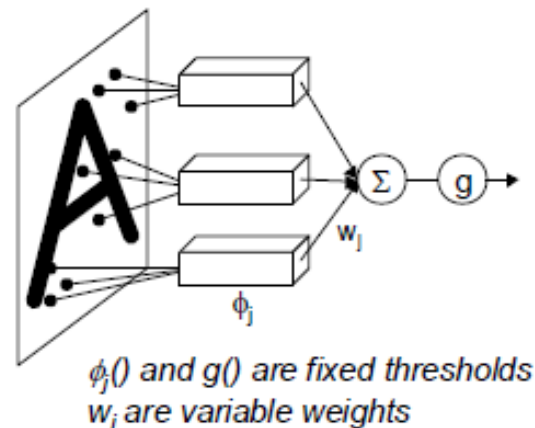
- In his book “The organization of Behavior”, Donald Hebb introduced his postulate of learning (a.k.a. Hebbian learning), which states that **the effectiveness of a variable synapse between two neurons is increased by the repeated activation of one neuron by the other across that synapse**
- The Hebbian rule has a strong similarity to the biological process in which a **neural pathway is strengthened each time it is used**



$$w_{ij}^{new} = w_{ij}^{old} + \alpha p_i a_j$$

Rosenblatt, 1958

- Frank Rosenblatt introduced the **perceptron, the simplest form of neural network**
- The perceptron is a single neuron with adjustable synaptic weights and a threshold activation function
- Rosenblatt's original perceptron consisted of three layers (sensory, association and response)
- Only one layer had variable weights, so his original perceptron is actually similar to a single neuron
- Rosenblatt also developed an error-correction rule to adapt these weights (the perceptron learning rule)
- **He also proved that if the (two) classes were linearly separable, the algorithm would converge to a solution (the perceptron convergence theorem)**



Widrow and Hoff, 1960

- Bernard Widrow and Ted Hoff introduced the LMS algorithm and used it to train the Adaline (ADaptive Linear Neuron)
- The Adaline was similar to the perceptron, except that it used a linear activation function instead of a threshold
- The LMS algorithm is still heavily used in adaptive signal processing

Minsky and Papert, 1969

- During the 1950s and 1960s two school of thought: Artificial Intelligence and Connectionism, competed for prestige and funding
 - The AI community was interested in the highest cognitive processes: logic, rational thought and problem solving
 - The Connectionists' principal model was the perceptron
 - The perceptron was severely limited in that it could only learn linearly separable patterns
 - Connectionists struggled to find a learning algorithm for multi-layer perceptrons that would not appear until the mid 1980s
- In their monograph "Perceptrons", Marvin Minsky and Seymour Papert (1969) mathematically proved the limitations of Rosenblatt's perceptron and conjectured that multi-layered perceptrons would suffer from the same limitations

The perceptron has shown itself worthy of study despite (and even because of!) its severe limitations. It has many features to attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. There is no reason to suppose that any of these virtues carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgement that the extension to multilayer systems is sterile.

- As a result of this book, research in ANNs was almost abandoned in the 1970s
 - Only a handful of researchers continued working on ANNs, mostly outside the US
- The 1970s saw the emergence of SOMs [van der Malsburg, 1973], [Amari, 1977], [Grossberg, 1976] and associative memories: [Kohonen, 1972], [Anderson, 1972]

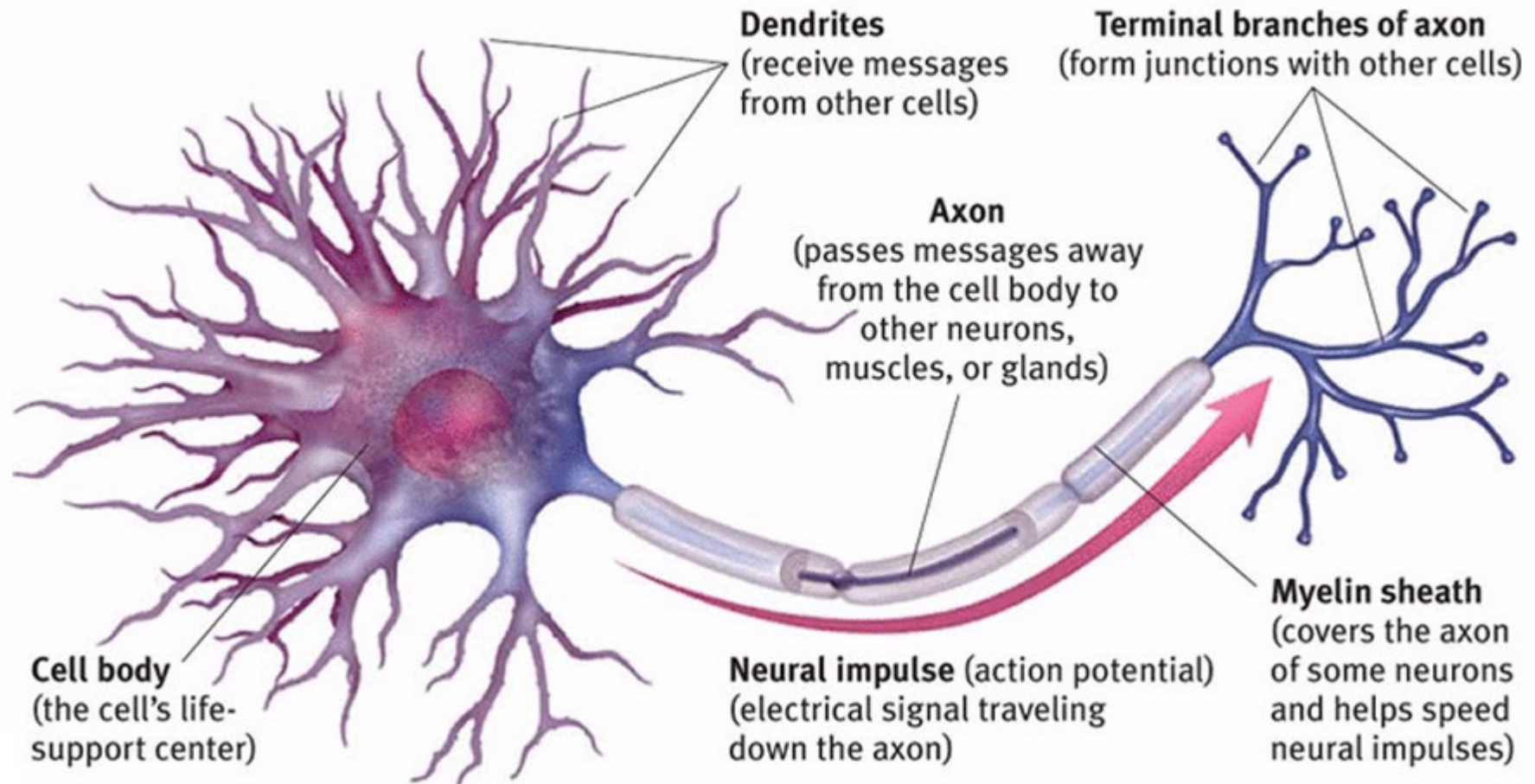
The resurgence of the early 1980s

- 1980: Steven **Grossberg**, one of the few researchers in the US that persevered despite the lack of support, establishes a new principle of self-organization called Adaptive Resonance Theory (ART) in collaboration with Gail Carpenter
- 1982: John **Hopfield** explains the operation of a certain class of recurrent ANNs (Hopfield networks) as an associative memory using statistical mechanics. Along with backprop, Hopfield's work was most responsible for the re-birth of ANNs
- 1982: **Teuvo Kohonen** presents SOMs using one- and two-dimensional lattices. Kohonen SOMs have received far more attention than van der Malsburg's work and have become the benchmark for innovations in self-organization
- 1983: **Kirkpatrick, Gelatt and Vecchi** introduced Simulated Annealing for solving combinatorial optimization problems. The concept of Simulated Annealing was later used by Ackley, Hinton and Sejnowsky (1985) to develop the Boltzmann machine (the first successful realization of multi-layered ANNs)
- 1983: **Barto, Sutton and Anderson** popularized reinforcement learning (Interestingly, it had been considered by Minsky in his 1954 PhD dissertation)
- 1984: Valentino **Braitenberg** publishes his book "Vehicles" in which he advocates for a bottom-up approach to understand complex systems: start with very elementary mechanism and build up

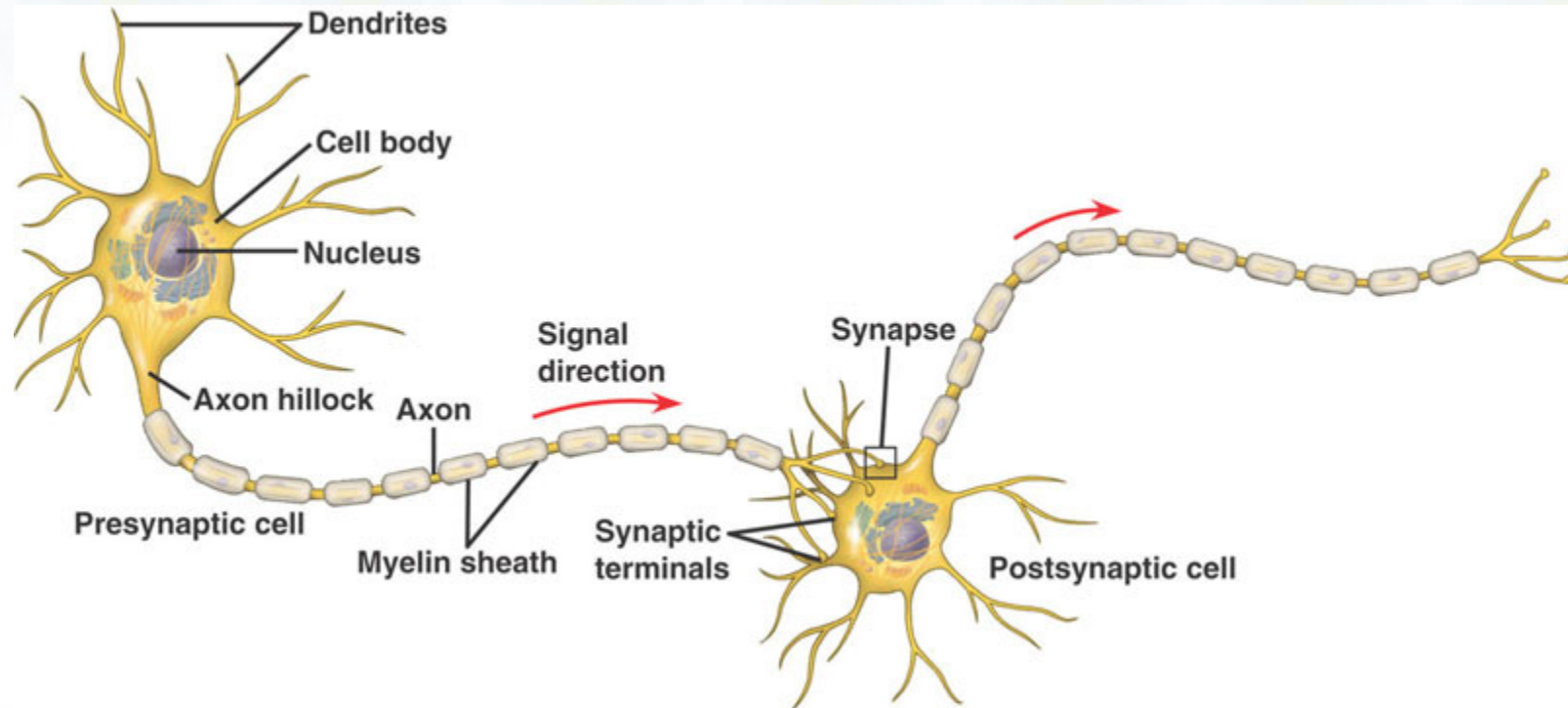
Rumelhart, Hinton and Williams, 1986

- In 1986, Rumelhart, Hinton and Williams announced the discovery of a method that allowed a network to learn to **discriminate between not linearly separable classes**. They called the method **“backward propagation of errors”**, a generalization of the LMS rule
- Backprop provided the solution to the problem that had puzzled Connectionists for two decades
 - Backprop was in reality a multiple invention: David Parker (1982, 1985) and Yann LeCun (1986) published similar discoveries
 - However, the honor of discovering backprop goes to Paul Werbos who presented these techniques in his 1974 Ph.D. dissertation at Harvard University

Neuron



Synapse





How Many Neurons In The Human Brain?

How Many Stars In The Milky Way?



Brain vs. Milky Way

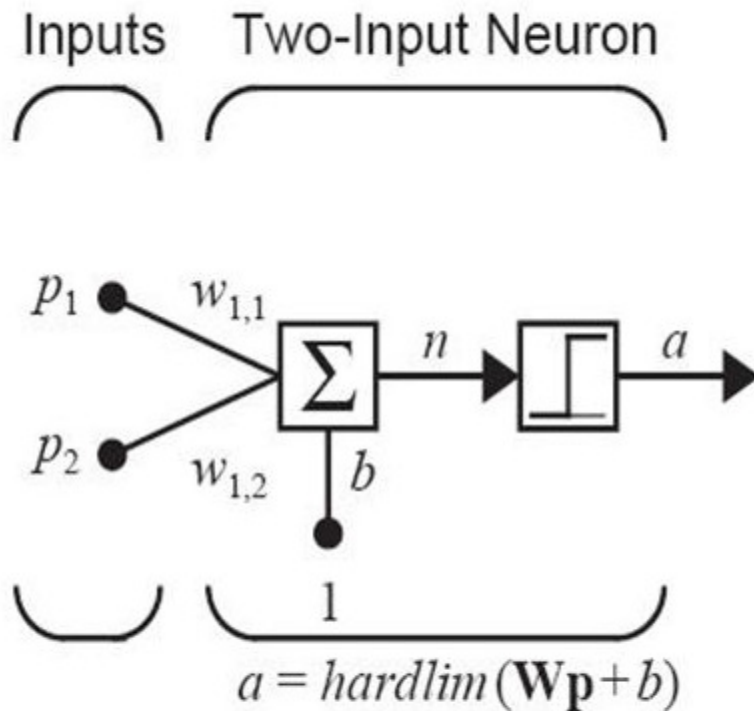
- Avg. Neurons in the Brain:
~ 86 billion neurons
- Approx. Number of Stars in the Milky Way:
~200-400 billion

Unfortunately the brain doesn't quite stack up ... !

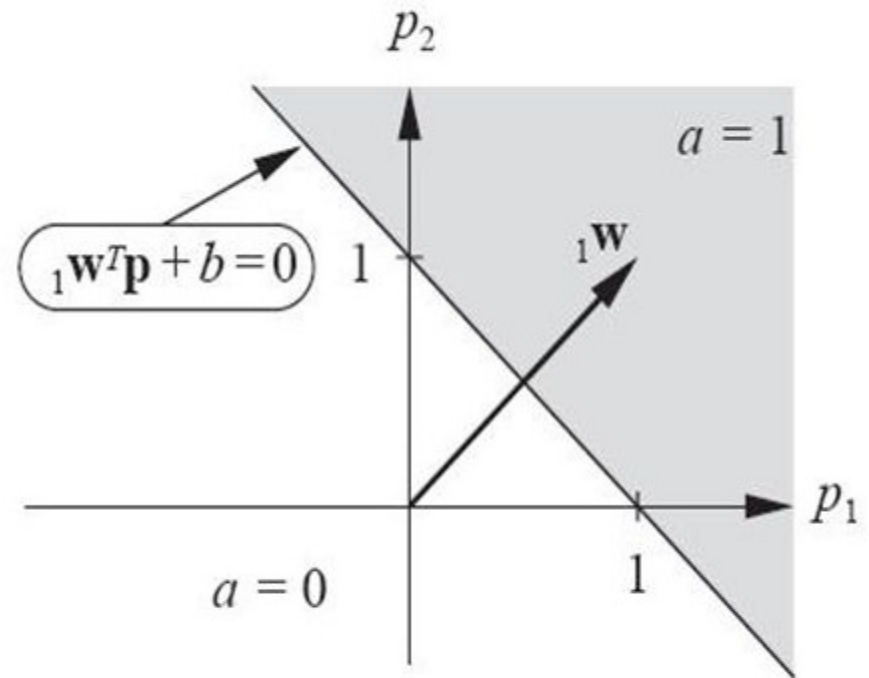
Fundamental Concept

- Neural networks are constructed and implemented to model the human brain
- Performs various tasks such as
 - Pattern matching
 - Classification
 - Optimization function
 - Approximation
 - Vector quantization
 - Data clustering
- These tasks are difficult for traditional computers

Model of Neuron

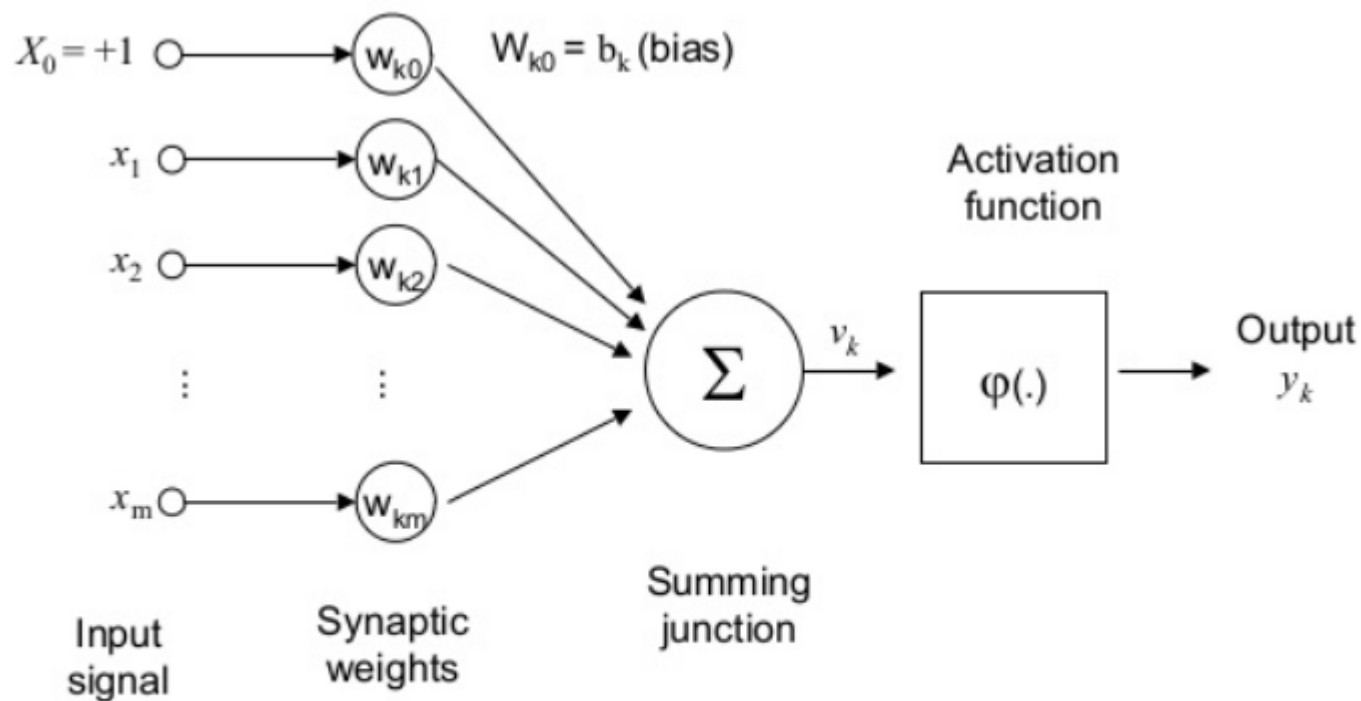


$$w_{1,1} = 1 \quad w_{1,2} = 1 \quad b = -1$$



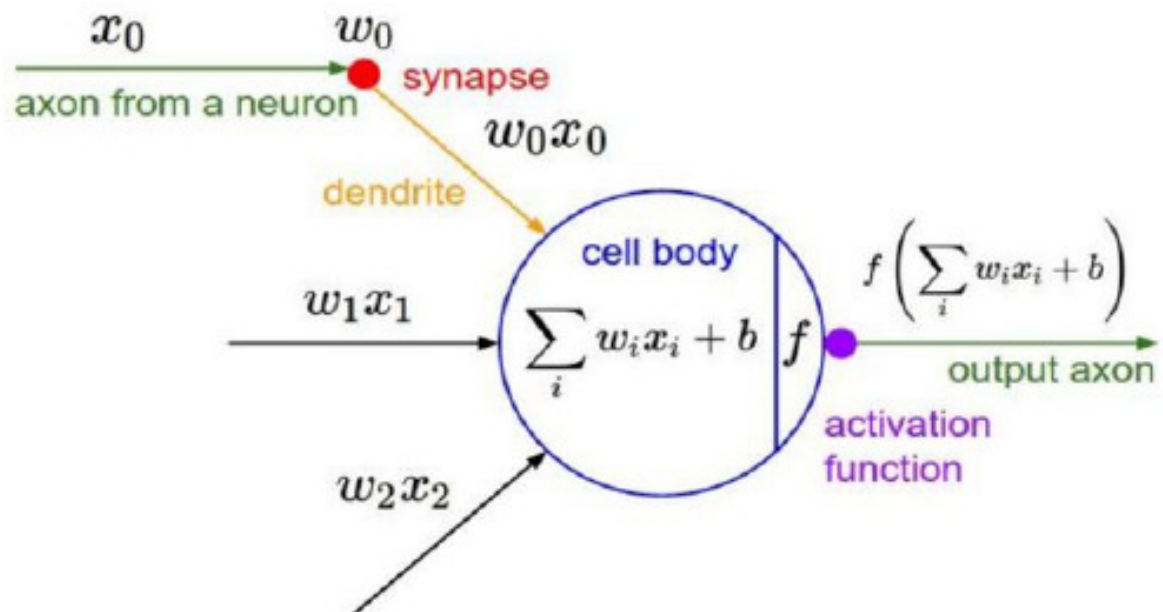
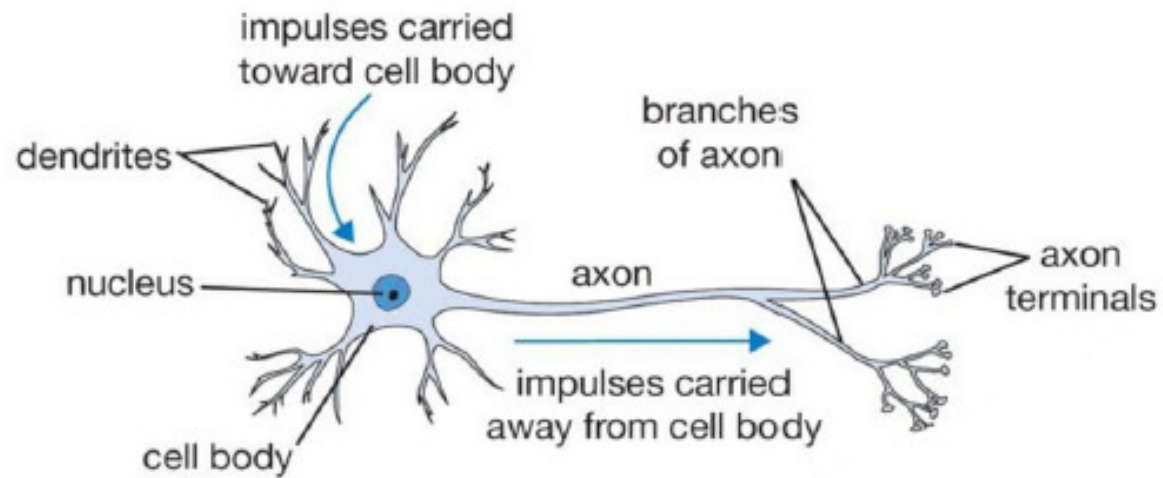
$$a = \text{hardlim}(\mathbf{w}^T \mathbf{p} + b) = \text{hardlim}(w_{1,1}p_1 + w_{1,2}p_2 + b)$$

Model of Neuron

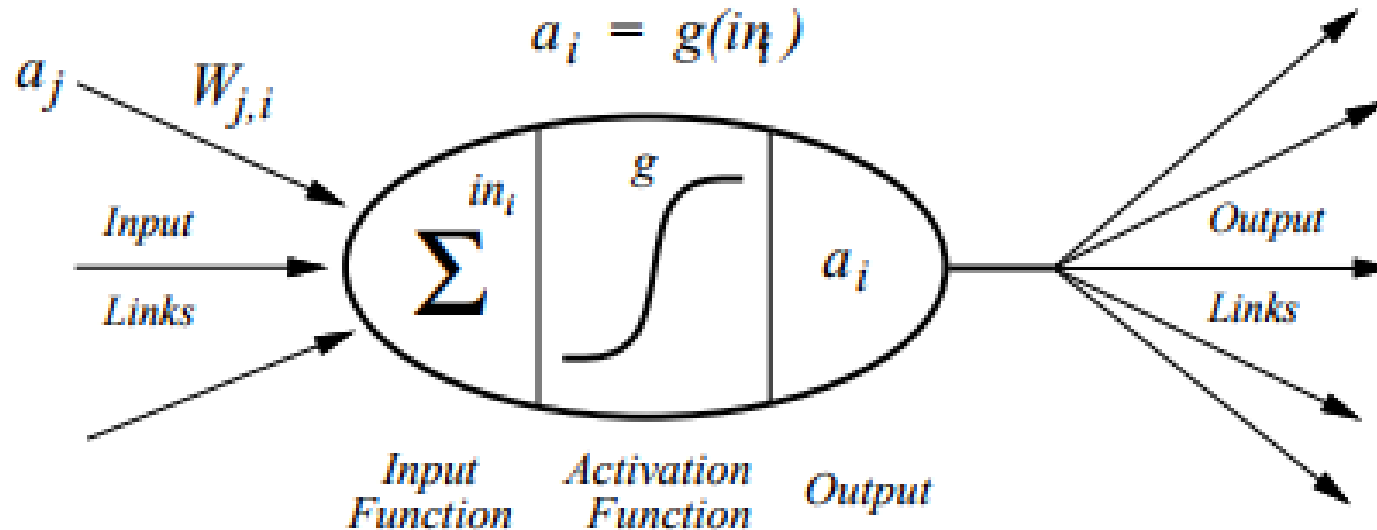


$$v_k = \sum_{j=0}^m w_{kj} x_j$$

$$y_k = \varphi(v_k)$$

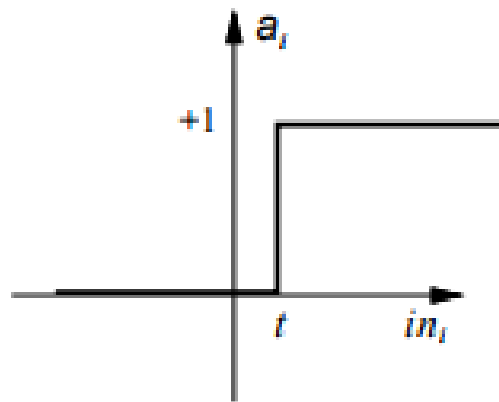


Main Processing Unit

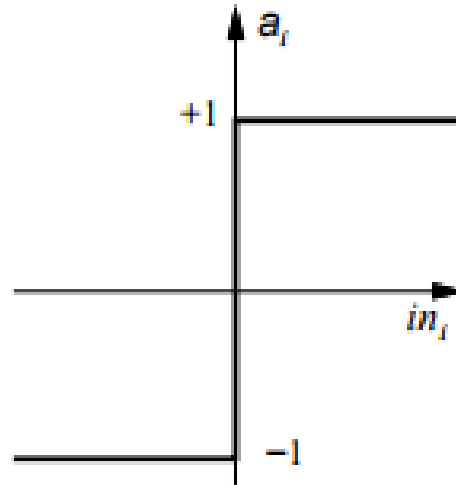


$$a_i = g\left(\sum_j W_{j,i} a_j\right)$$

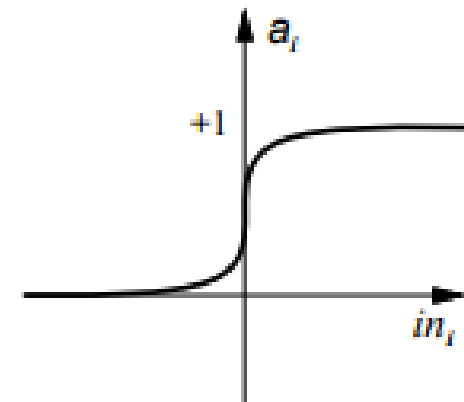
Different Threshold Functions



(a) Step function



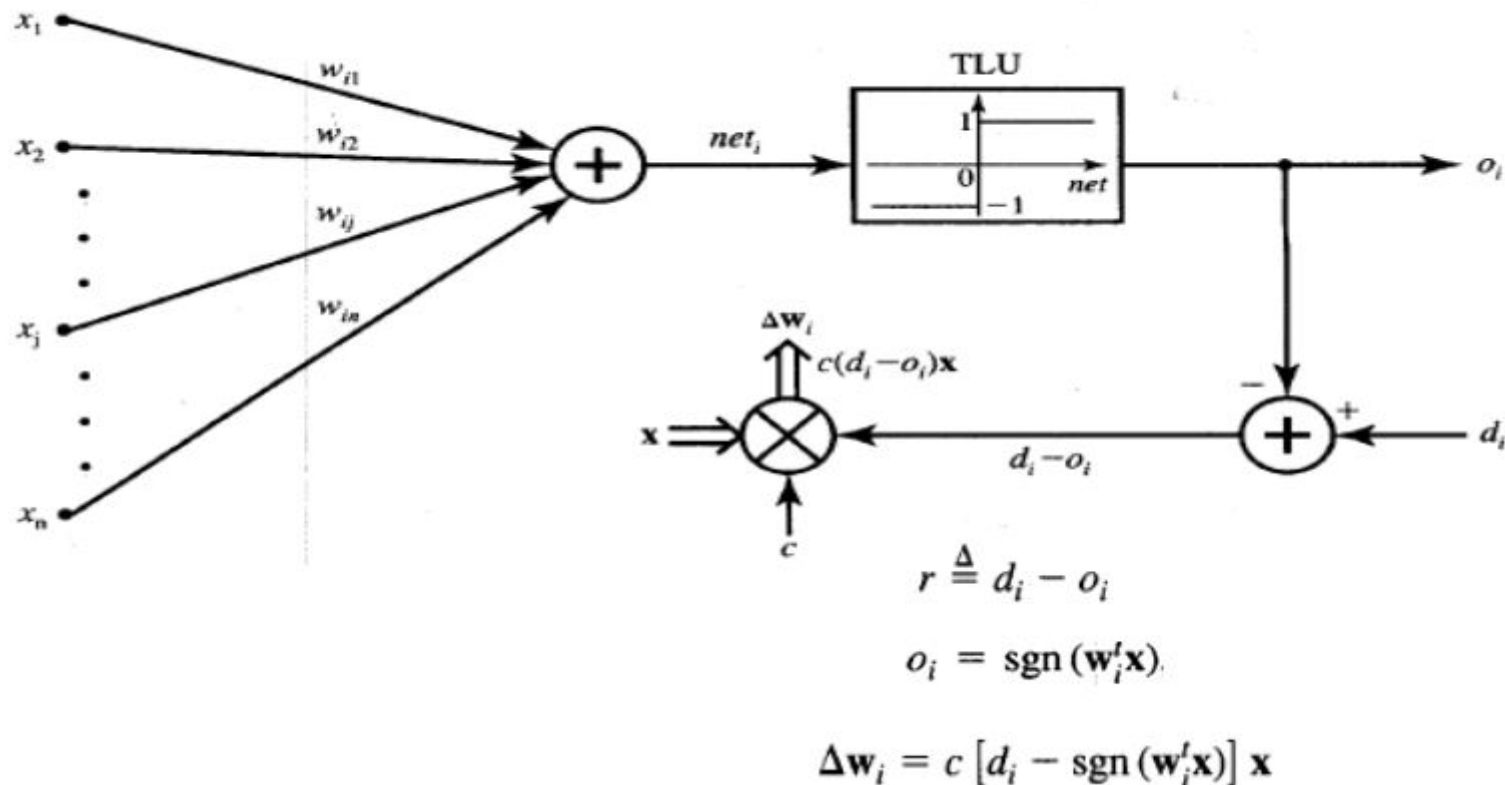
(b) Sign function



(c) Sigmoid function

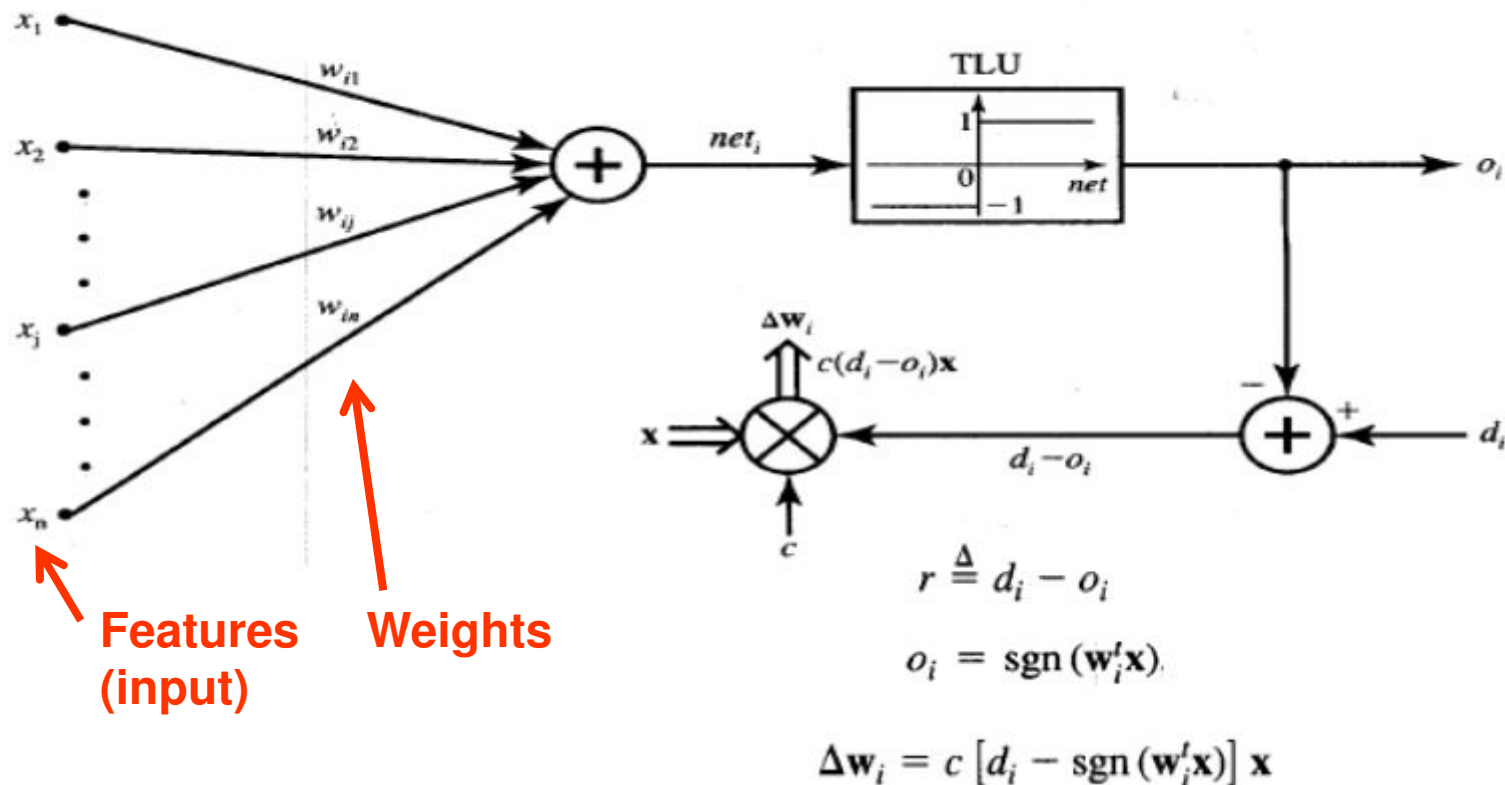
Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



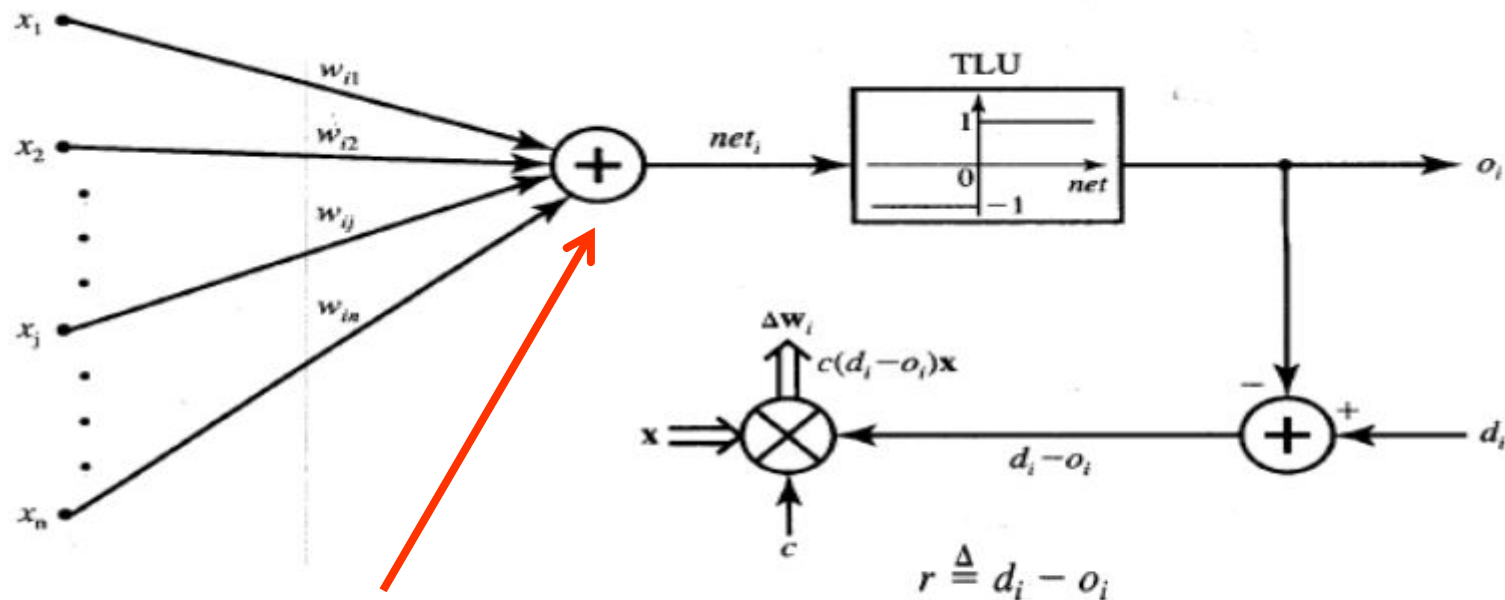
Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



Multiply the feature and the weight and then sum them all up

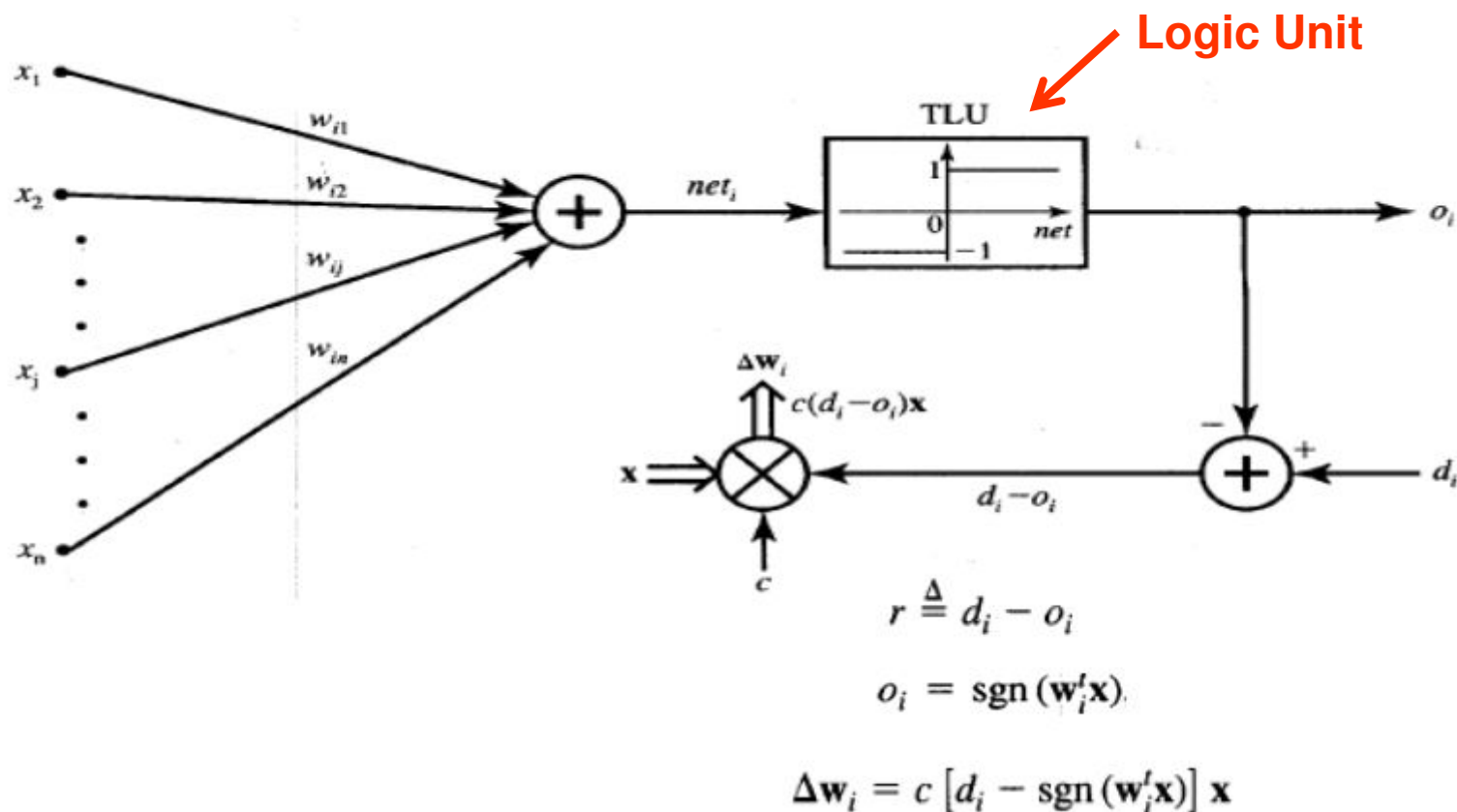
$$r \triangleq d_i - o_i$$

$$o_i = \text{sgn}(w_i'x)$$

$$\Delta w_i = c [d_i - \text{sgn}(w_i'x)] x$$

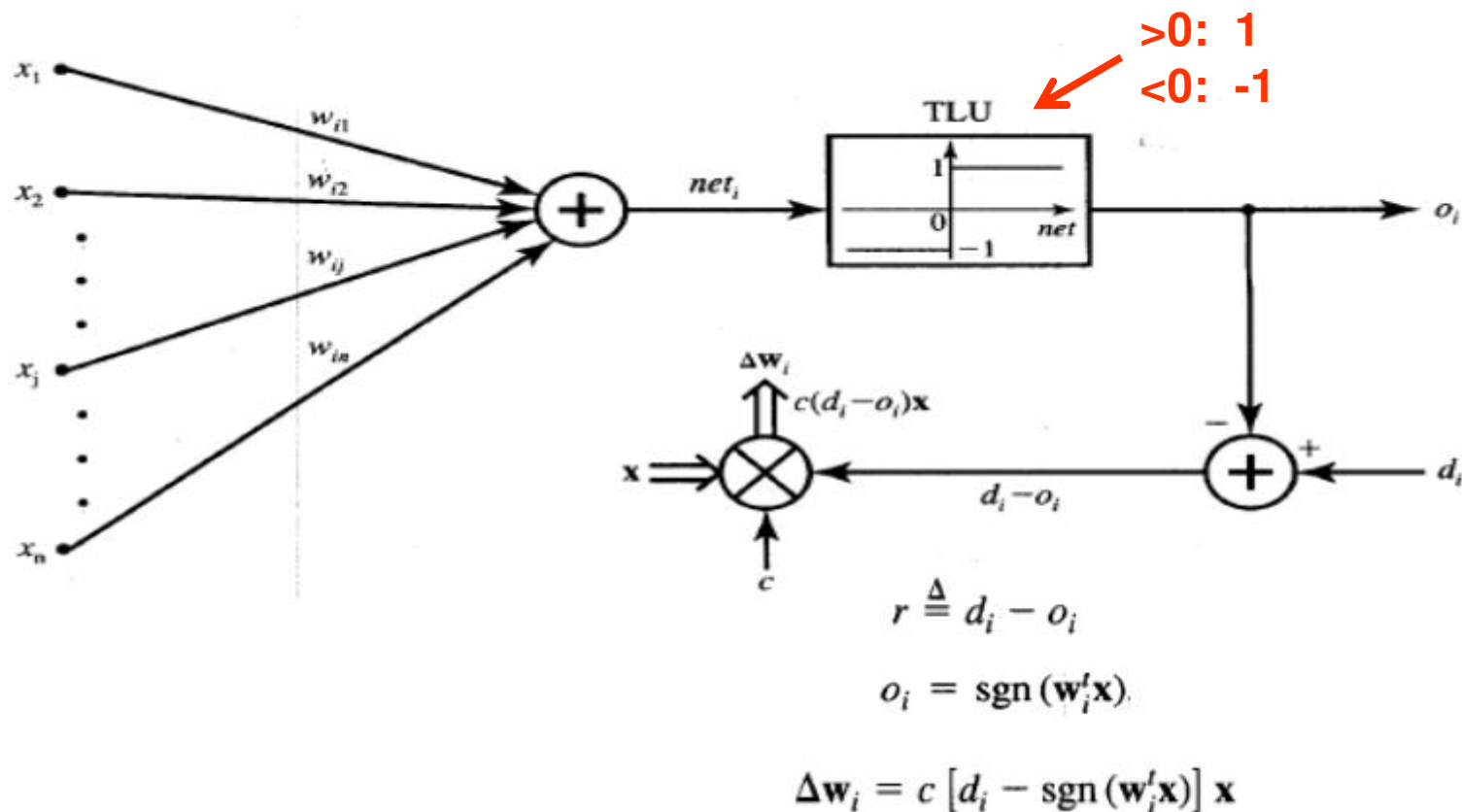
Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



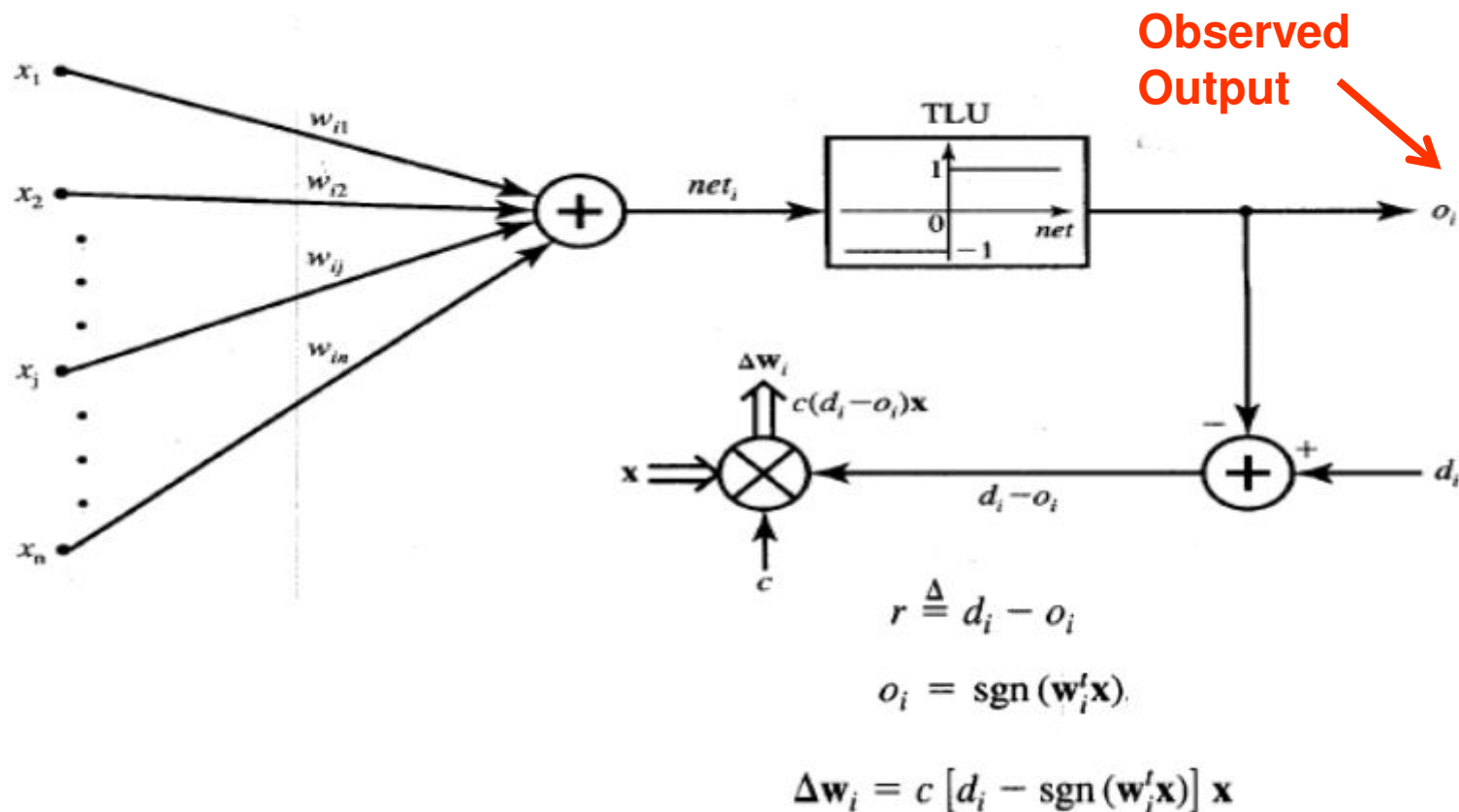
Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



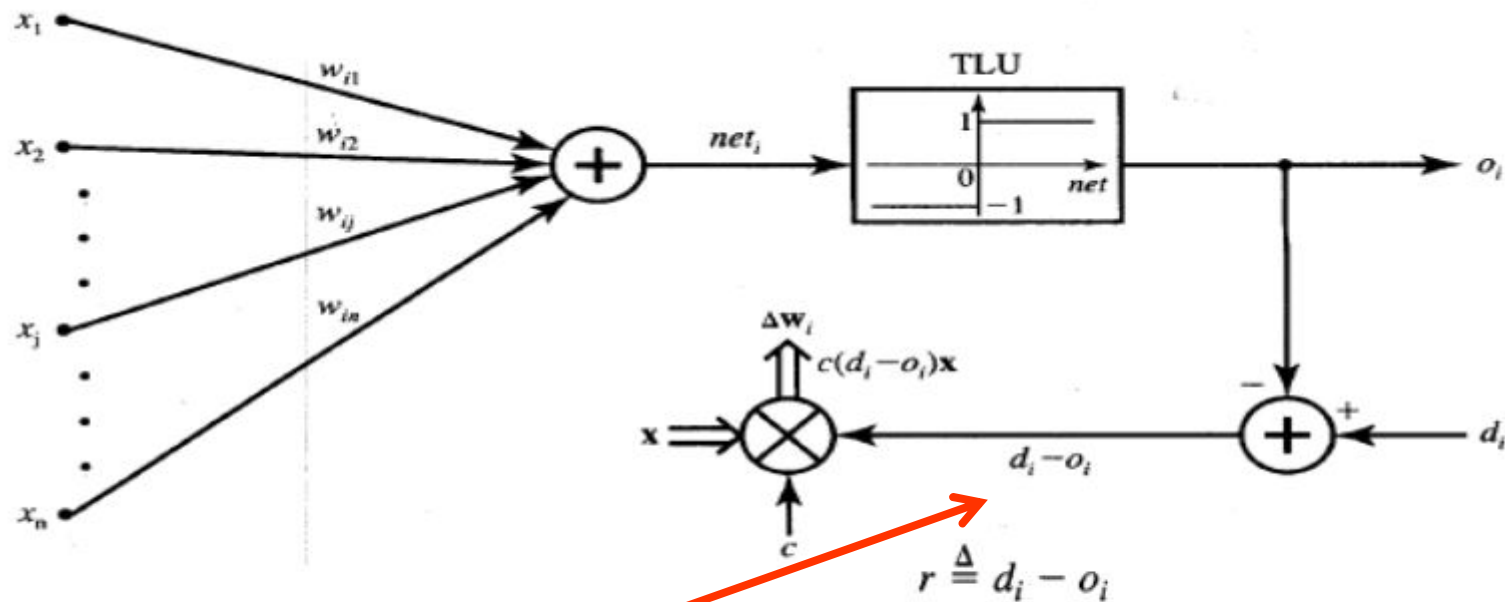
Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



Perceptron Learning “Delta” Rule

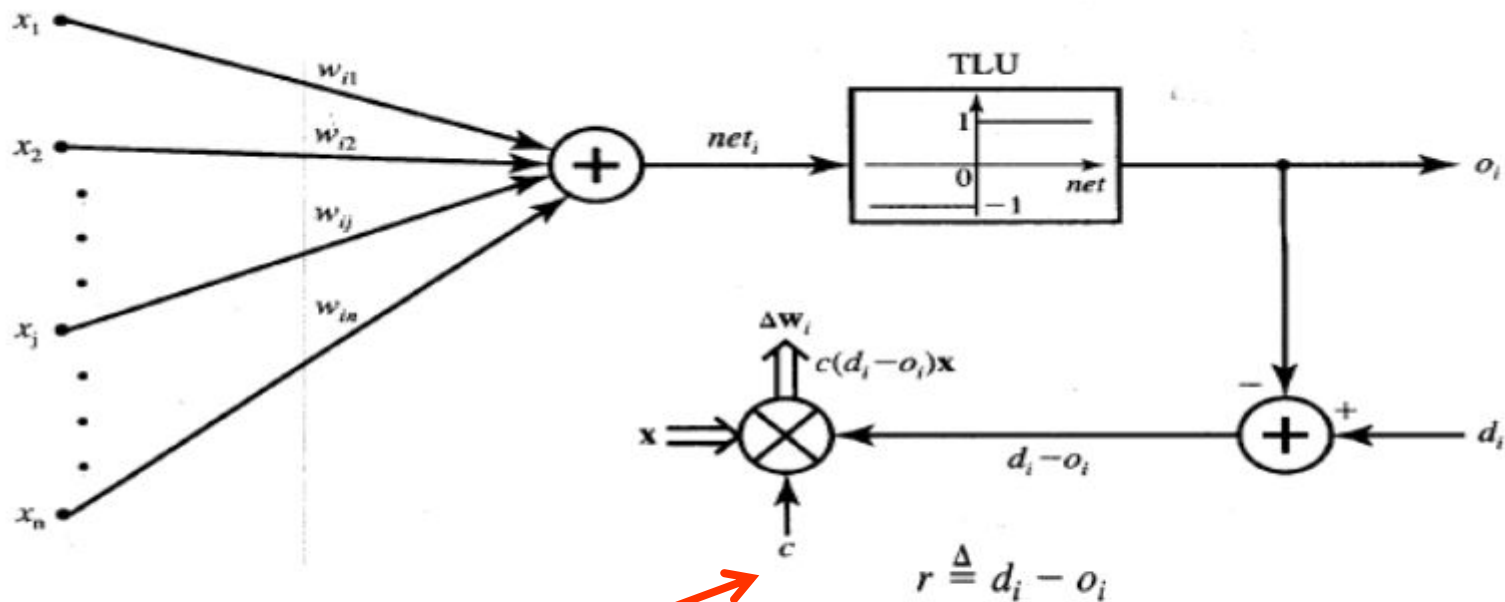
- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



If correct
(observed=actual) nothing
new to learn: no difference

Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



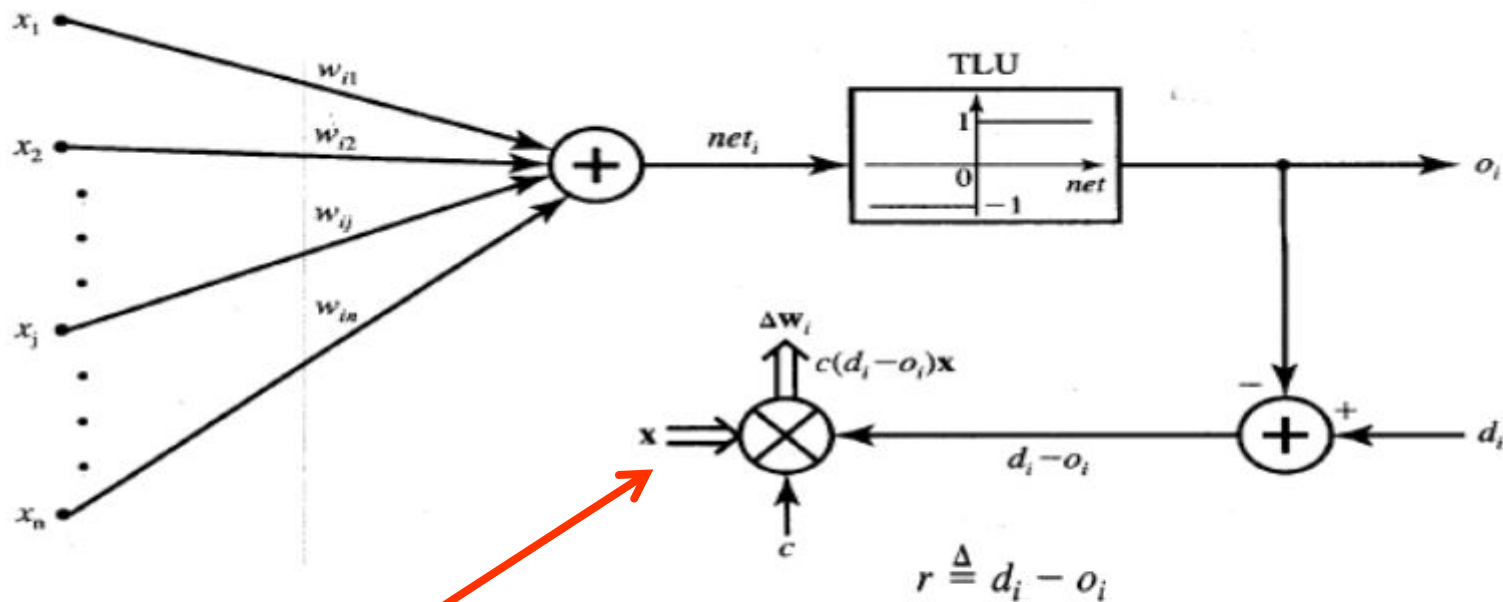
Decay rate / learning
rate

$$r \triangleq d_i - o_i$$
$$o_i = \text{sgn}(\mathbf{w}_i' \mathbf{x})$$

$$\Delta \mathbf{w}_i = c [d_i - \text{sgn}(\mathbf{w}_i' \mathbf{x})] \mathbf{x}$$

Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



**Vector of
inputs/features**

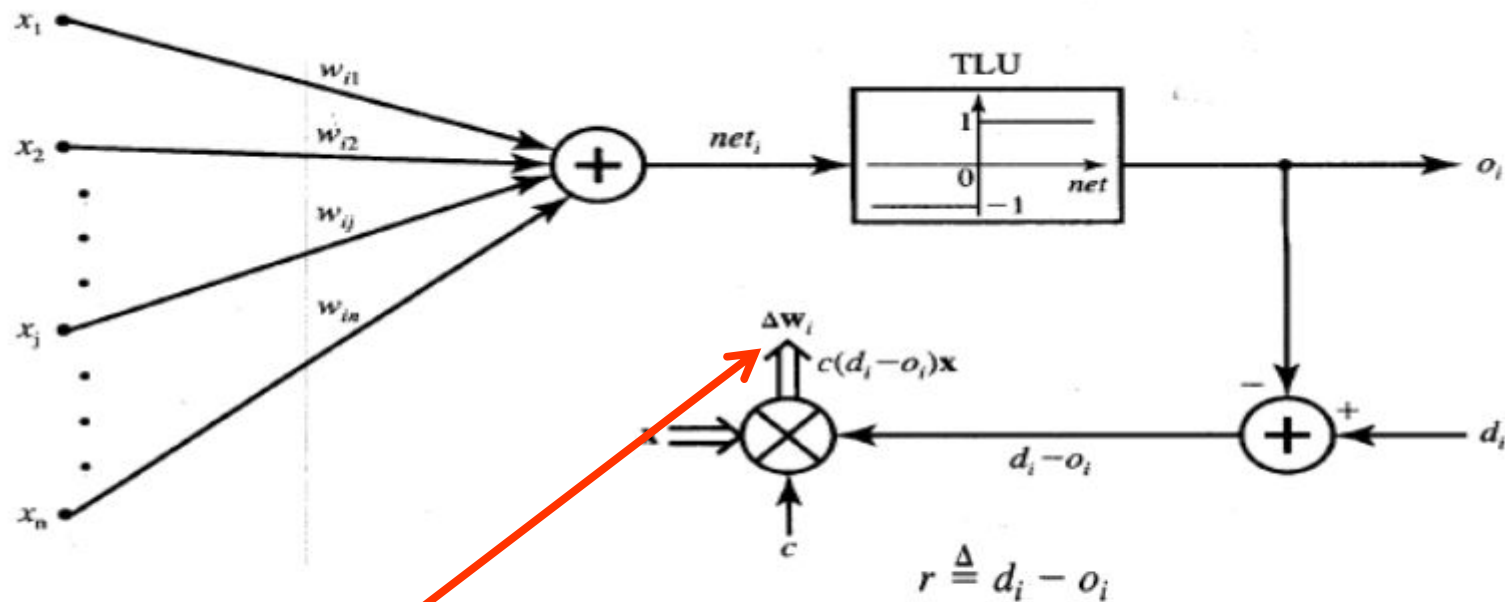
$$r \triangleq d_i - o_i$$

$$o_i = \text{sgn}(\mathbf{w}_i' \mathbf{x})$$

$$\Delta \mathbf{w}_i = c [d_i - \text{sgn}(\mathbf{w}_i' \mathbf{x})] \mathbf{x}$$

Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



Change of weight:
 $W(t+1) - W(t)$
 $t = \text{time instance / iteration}$

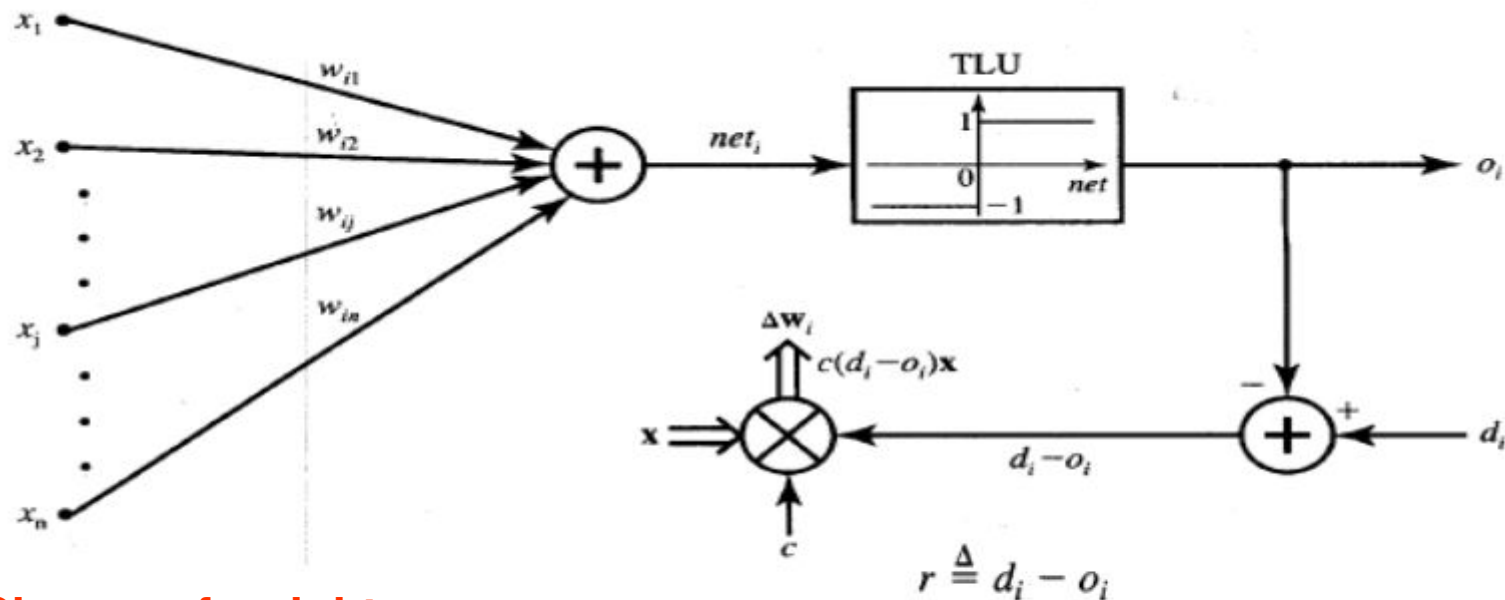
$$r \triangleq d_i - o_i$$

$$o_i = \text{sgn}(w_i'x)$$

$$\Delta w_i = c [d_i - \text{sgn}(w_i'x)] x$$

Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



Change of weight:
 $W(t+1) = W(t) + C + \text{error} + X_i$
C=learning rate
 X_i =vector

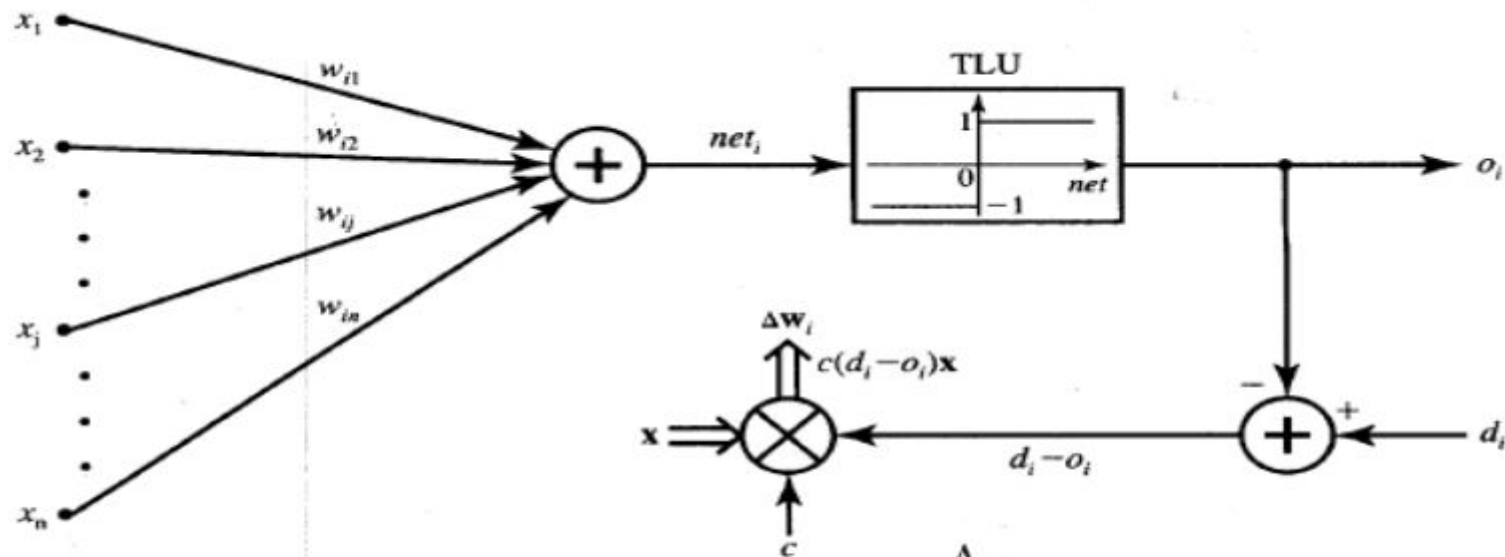
$$r \triangleq d_i - o_i$$

$$o_i = \text{sgn}(w_i'x)$$

$$\Delta w_i = c [d_i - \text{sgn}(w_i'x)] x$$

Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



Based on SIGN of the net
e.g. Error is +/- 2
Then change in weights will
be modified accordingly

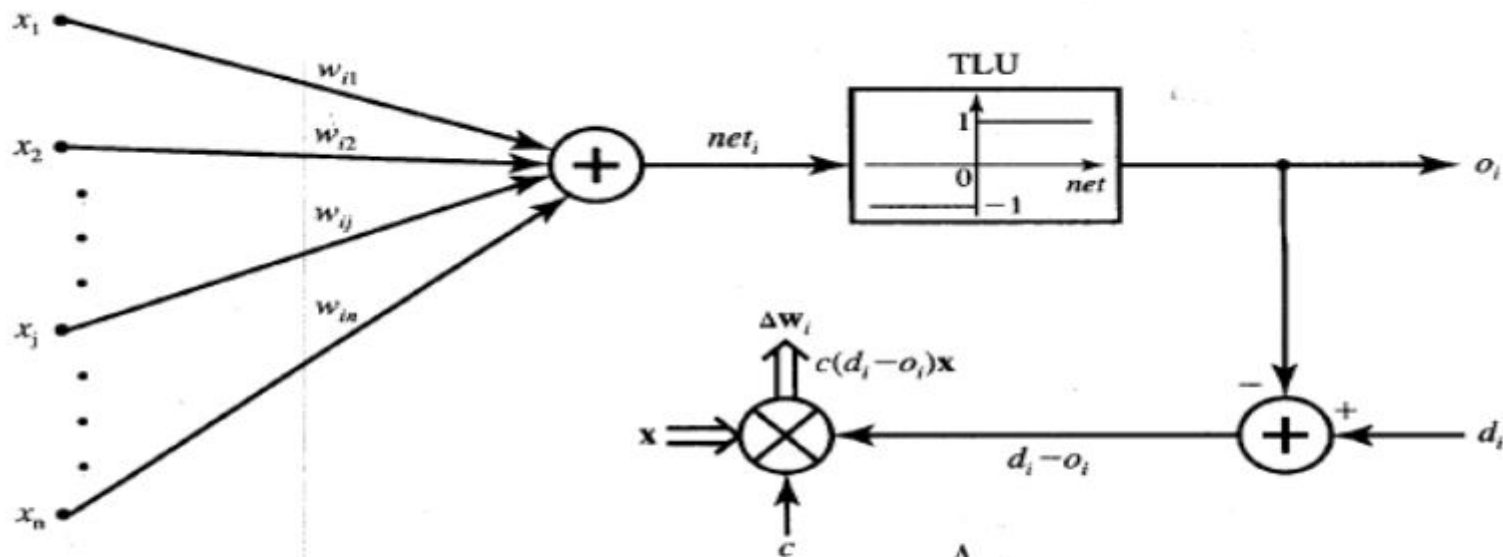
$$r \triangleq d_i - o_i$$

$$o_i = \text{sgn}(w_i'x)$$

$$\Delta w_i = c [d_i - \text{sgn}(w_i'x)] x$$

Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



If no error, sign of net matches sign of instance for that iteration then error = 0

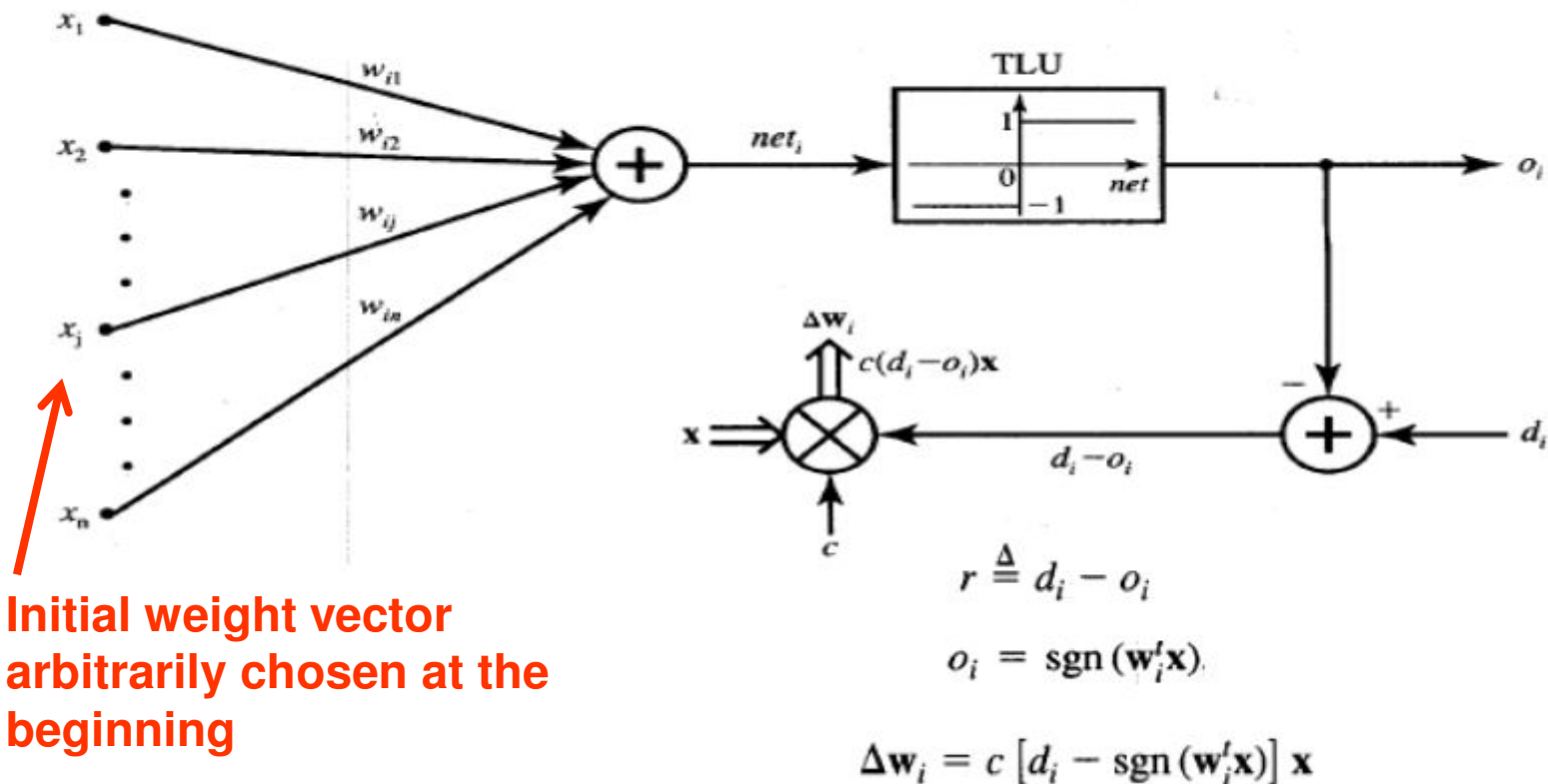
$$r \triangleq d_i - o_i$$

$$o_i = \text{sgn}(\mathbf{w}_i' \mathbf{x})$$

$$\Delta \mathbf{w}_i = c [d_i - \text{sgn}(\mathbf{w}_i' \mathbf{x})] \mathbf{x}$$

Perceptron Learning “Delta” Rule

- Learning signal is difference between the desired and actual neuron's response [Learning is supervised]



Perceptron “Delta” Rule Details

$$o_i = f(\text{net}) = wx$$

(output ‘o’) (pairwise sum of input vector ‘x’ and weight ‘w’)

- We want to *minimize* the error

$$\min E(\text{error}) = -\frac{1}{2} (d_i - o_i)^2$$

d = constant;
o depends on w

- Derivative with respect to w ...

$$\begin{aligned} \frac{\partial E}{\partial w} &= -\frac{1}{2} \cdot 2(d_i - o_i) \cdot -1 \cdot \frac{\partial o_i}{\partial w} \\ &= (d_i - o_i) \cdot x = (\text{error}_i)x \end{aligned}$$

error = difference
between actual
and output

- Change in weight w involves the learning rate ‘c’

$$\begin{aligned} \Delta w^i &= c \cdot \text{error}_i \cdot x_i = c \cdot e_i \cdot x_i \\ &= c \cdot [d_i - \underbrace{\text{sgn}(w^i x_i)}_{\text{net}}] x_i \end{aligned}$$

Instance /
exemplar “i”

net

o_i

Perceptron “Delta” Rule Details

$$\Delta w^i = c \cdot [d_i - \underbrace{\underbrace{\text{sgn}(w^i x_i)}_{\text{net}}}_{o_i}] x_i$$

Change in weights for next iteration $w(t+1)$ involves weights of previous iteration $w(t)$:

$$\Delta w = w(t + 1) - w(t)$$

$$w(t + 1) = w(t) + c \cdot e_i \cdot x_i$$

= previous weight, learning rate (c), error, strength of vector

Note: t = time instance / iteration

current iteration is $(t+1)$, previous iteration is (t)

EXAMPLE: Perceptron

Signs 1 or -1 for actual class

- $x_1 = 1$; $x_3 = 3$; $d_1 = d_3 = 1 \rightarrow \text{Class 1}$
- $x_2 = -0.5$; $x_4 = -2$; $d_2 = d_4 = -1 \rightarrow \text{Class 2}$

Augmented pattern vectors $\langle p_1, p_2, p_3, p_4 \rangle$

$$x_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad x_2 = \begin{pmatrix} -0.5 \\ 1 \end{pmatrix} \quad x_3 = \begin{pmatrix} 3 \\ 1 \end{pmatrix} \quad x_4 = \begin{pmatrix} -2 \\ 1 \end{pmatrix}$$

Assume initial weights vector: $w^1 = (-2.5 \ 1.75)$

Learning rate: $c = 1$

*Arbitrarily
chosen*

Error:

$$e_i = \pm 2 \rightarrow \Delta w^i = \pm 2x_i$$
$$e_i = 0 \rightarrow \Delta w^i = 0 \text{ (no change)}$$

EXAMPLE: Perceptron

- $x_1 = 1$; $x_3 = 3$; $d_1 = d_3 = 1 \rightarrow$ Class 1
- $x_2 = -0.5$; $x_4 = -2$; $d_2 = d_4 = -1 \rightarrow$ Class 2

initial weight vector ↘

$$p_1 net = [-2.5 \quad 1.75] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -0.75$$

x_1 ↖

sgn(net) = -1 = $o_i \neq d_i = 1 \rightarrow e_i = 2$

Reminder:
 $(-2.5)(1)$
+
 $(1.75)(1)$

$$w^2 = w^1 + 2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -2.5 \\ 1.75 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 3.75 \end{bmatrix}$$

Recall: $w(t+1) = w(t) + c \cdot e_i \cdot x_i$

EXAMPLE: Perceptron

- $x_1 = 1$; $x_3 = 3$; $d_1 = d_3 = 1$ → Class 1
- $x_2 = -0.5$; $x_4 = -2$; $d_2 = d_4 = -1$ → Class 2

$$p_2 net =$$

EXAMPLE: Perceptron

- $x_1 = 1$; $x_3 = 3$; $d_1 = d_3 = 1$ → Class 1
- $x_2 = -0.5$; $x_4 = -2$; $d_2 = d_4 = -1$ → Class 2

$$p_3 net =$$

EXAMPLE: Perceptron

- $x_1 = 1$; $x_3 = 3$; $d_1 = d_3 = 1$ → Class 1
- $x_2 = -0.5$; $x_4 = -2$; $d_2 = d_4 = -1$ → Class 2

$$p_4 net =$$

EXAMPLE: Perceptron

We reached the end of one cycle

Need to keep going until weights do not change or
Minimum threshold is reached

So ...

Start a new cycle $\langle p_1, p_2, p_3, p_4 \rangle$

EXAMPLE: Perceptron

- $x_1 = 1$; $x_3 = 3$; $d_1 = d_3 = 1$ → Class 1
- $x_2 = -0.5$; $x_4 = -2$; $d_2 = d_4 = -1$ → Class 2
- Start new cycle $\langle p_1, p_2, p_3, p_4 \rangle$

using weight vector w^5 :

$$p_1 net = [4.5 \quad -0.25] \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 4.25$$

$\swarrow x_1$

$$\text{sgn}(\text{net}) = 1 = o_i = d_i = 1 \rightarrow e_i = 0$$

$$w^6 = w^5$$

EXAMPLE: Perceptron

- $x_1 = 1$; $x_3 = 3$; $d_1 = d_3 = 1 \rightarrow \text{Class 1}$
- $x_2 = -0.5$; $x_4 = -2$; $d_2 = d_4 = -1 \rightarrow \text{Class 2}$
- Start new cycle $\langle p_1, p_2, p_3, p_4 \rangle$

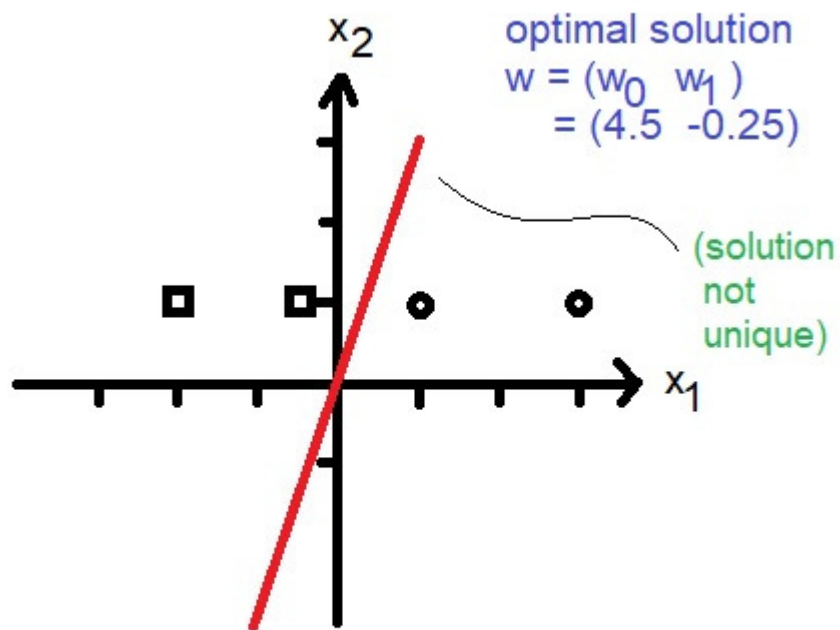
$$p_2 \text{ net} = -2.5; \text{sgn}(\text{net}) = -1 = o_i = d_i = -1 \rightarrow e_i = 0$$
$$w^7 = w^6$$

$$p_3 \text{ net} = 13.25; \text{sgn}(\text{net}) = 1 = o_i = d_i = 1 \rightarrow e_i = 0$$
$$w^8 = w^7$$

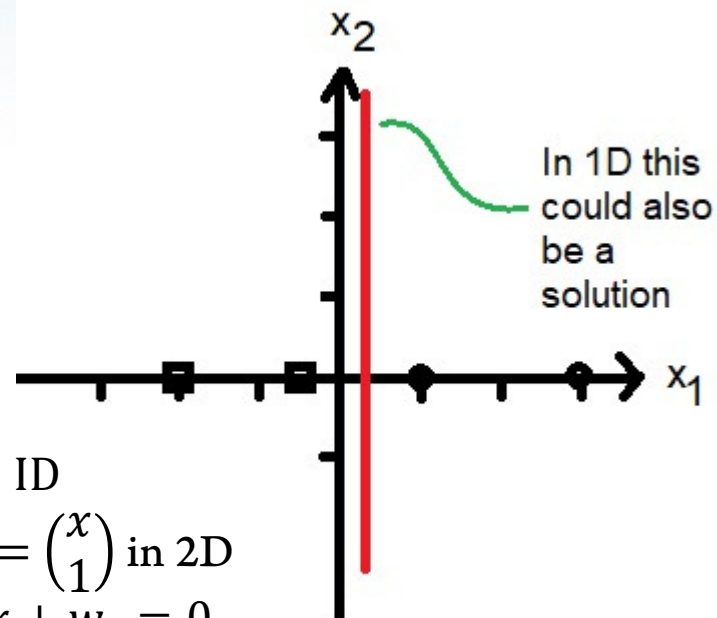
$$p_4 \text{ net} = -9.25; \text{sgn}(\text{net}) = -1 = o_i = d_i = -1 \rightarrow e_i = 0$$
$$w^9 = w^8$$

NO CHANGES IN THIS CYCLE! SOLUTION: $w^9 = [4.5 \ -0.25]^t$


EXAMPLE: Perceptron



Linear Decision Boundary (in 2D)
 $4.5x_1 - 0.25x_2 = 0$



If $x \in \text{ID}$
 $\rightarrow y = \begin{pmatrix} x \\ 1 \end{pmatrix}$ in 2D
 $w_0x + w_1 = 0$
 $x + \frac{w_1}{w_0} = 0$
 $x = \frac{0.25}{4.5} = \frac{1}{18}$



In-Class Activity!

Your turn to practice a Perceptron example

IN-CLASS ACTIVITY: PERCEPTRON

- $x_1 = 1$; $x_3 = 2.5$; $x_5 = 4$; $d_1 = d_3 = d_5 = 1$ → Class 1
- $x_2 = -0.75$; $x_4 = -1$; $d_2 = d_4 = -1$ → Class 2

Augmented pattern vectors $\langle p_1, p_2, p_3, p_4, p_5 \rangle$

$$x_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad x_2 = \begin{pmatrix} -0.75 \\ 1 \end{pmatrix} \quad x_3 = \begin{pmatrix} 2.5 \\ 1 \end{pmatrix} \quad x_4 = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \quad x_5 = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$$

Assume initial weights vector: $w^1 = ($  $)$

Learning rate: $c = 1$

Error:

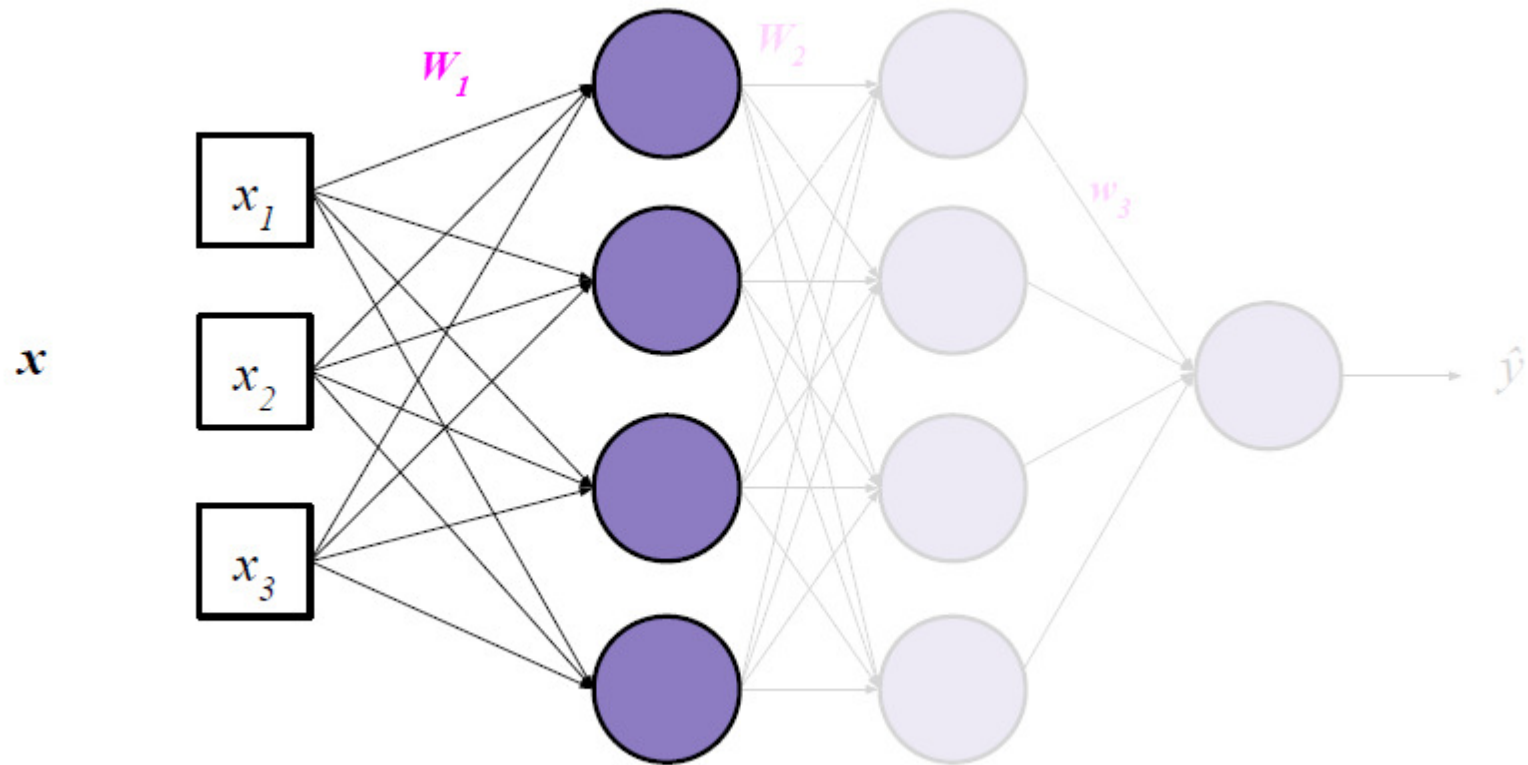
$$e_i = \pm 2 \rightarrow \Delta w^i = \pm 2x_i$$
$$e_i = 0 \rightarrow \Delta w^i = 0 \text{ (no change)}$$

*Arbitrarily
Chosen – come up
with your own*

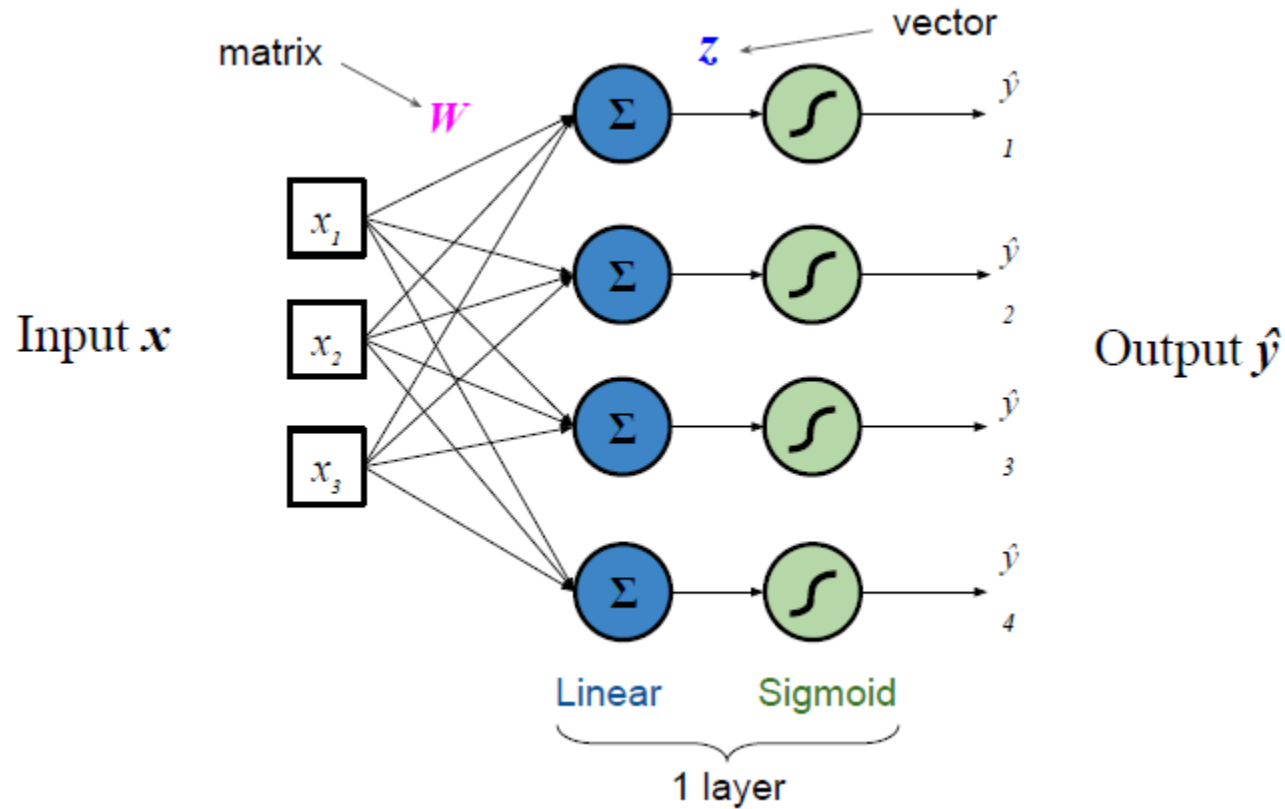
IN-CLASS ACTIVITY: PERCEPTRON

- **Work *individually*** on this assignment
 - However you may discuss with your neighbor
- **Submit on Collab** under assignments for this Perceptron In-Class activity
- Everybody must submit by **end of class today**
(or latest by **8:00pm** tonight)

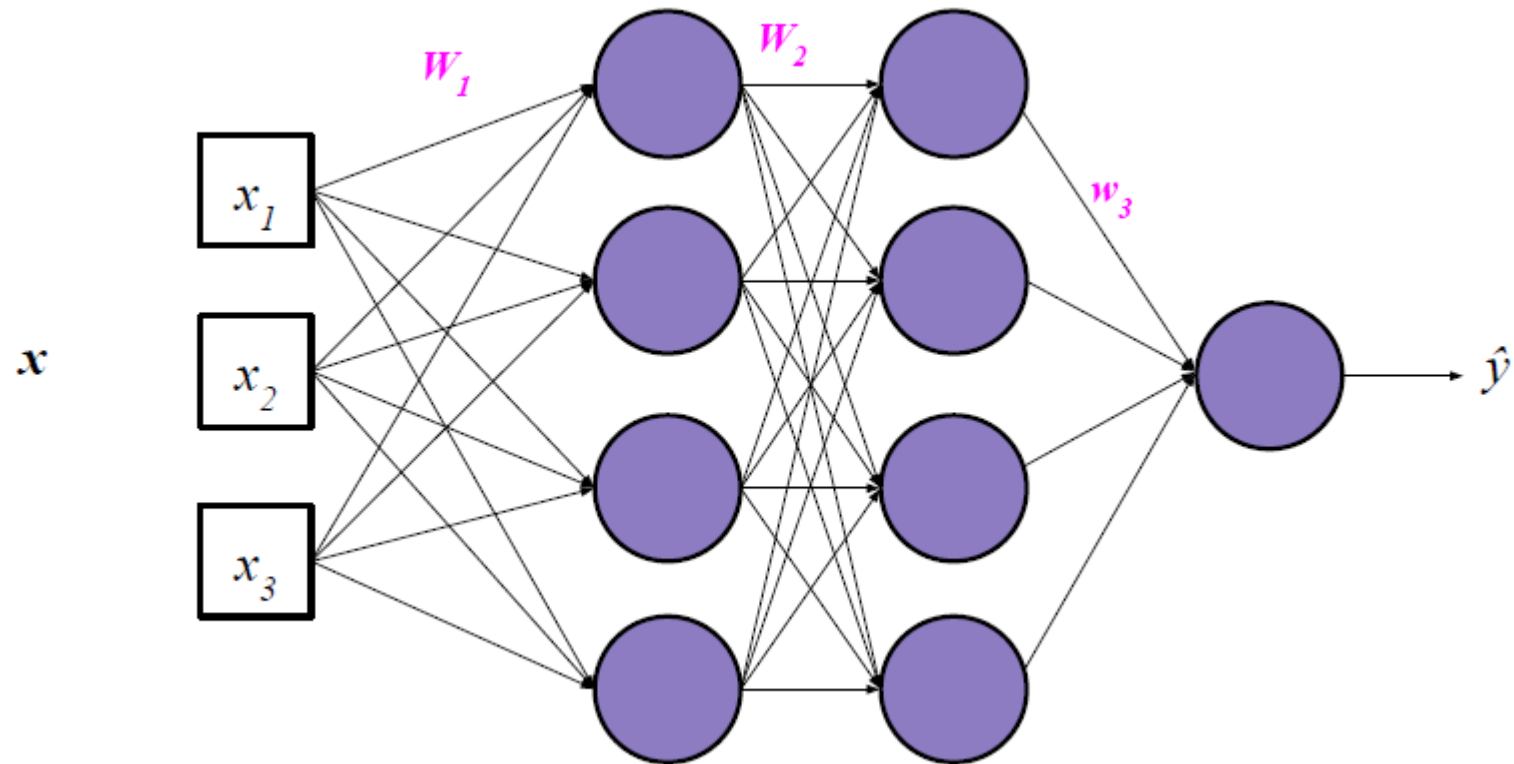
1-Layer Network



1-Layer Network (with 4 neurons)

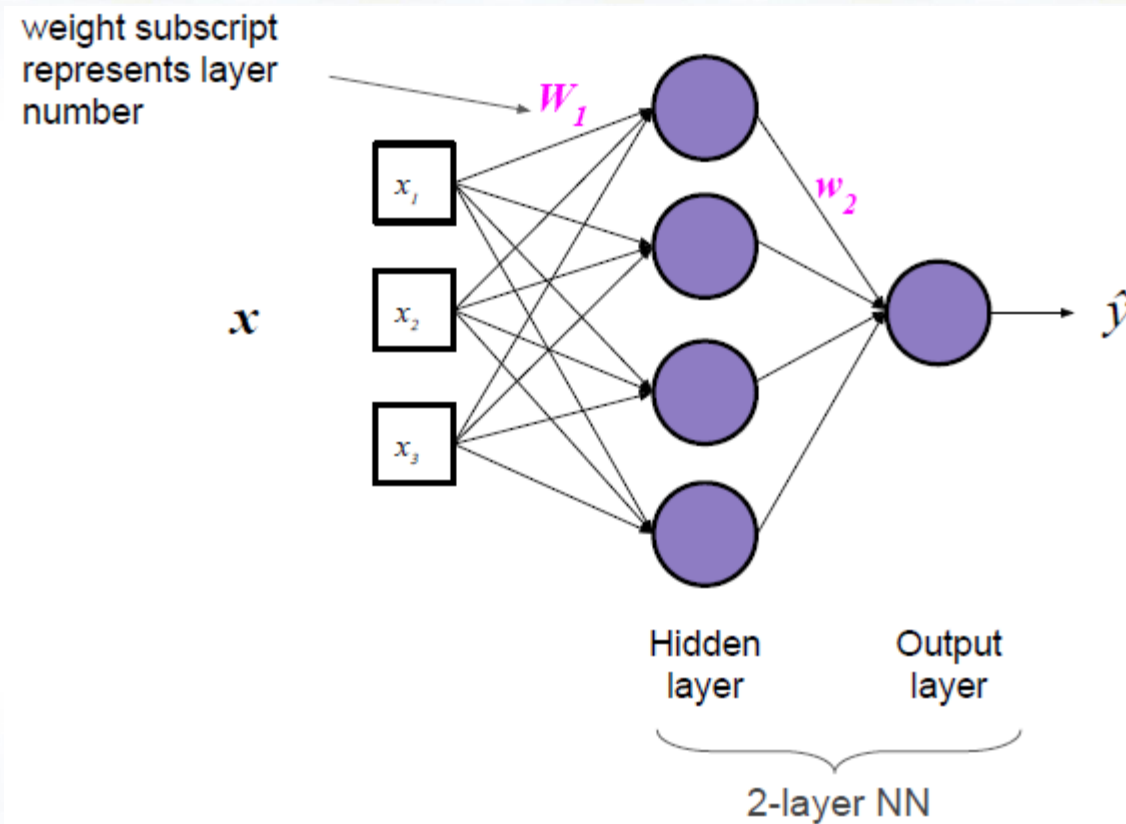


Multi-Layer Neural Network



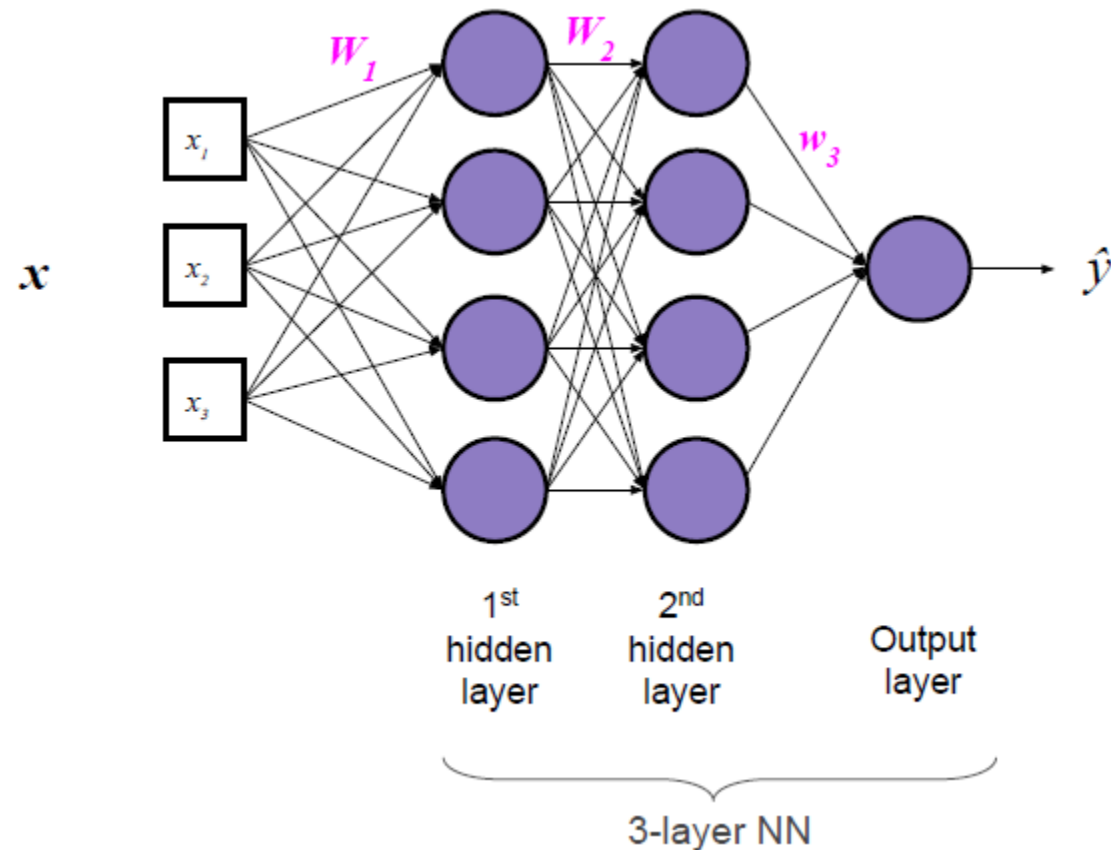
Multi-Layer Neural Network

(Multi-Layer Perceptron [MLP] Network)



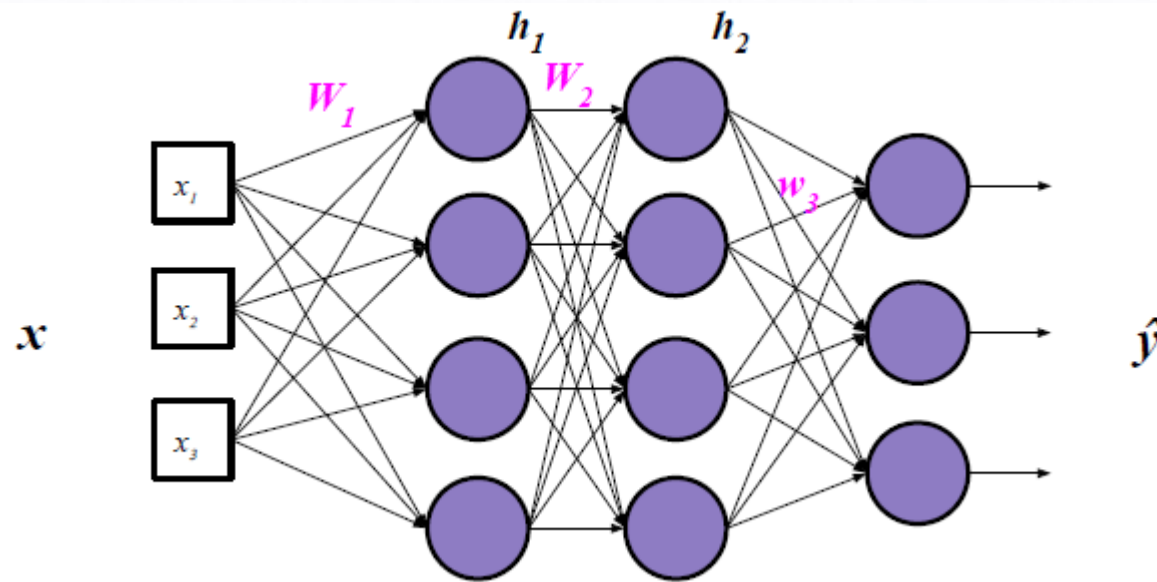
Multi-Layer Neural Network

(Multi-Layer Perceptron [MLP] Network)

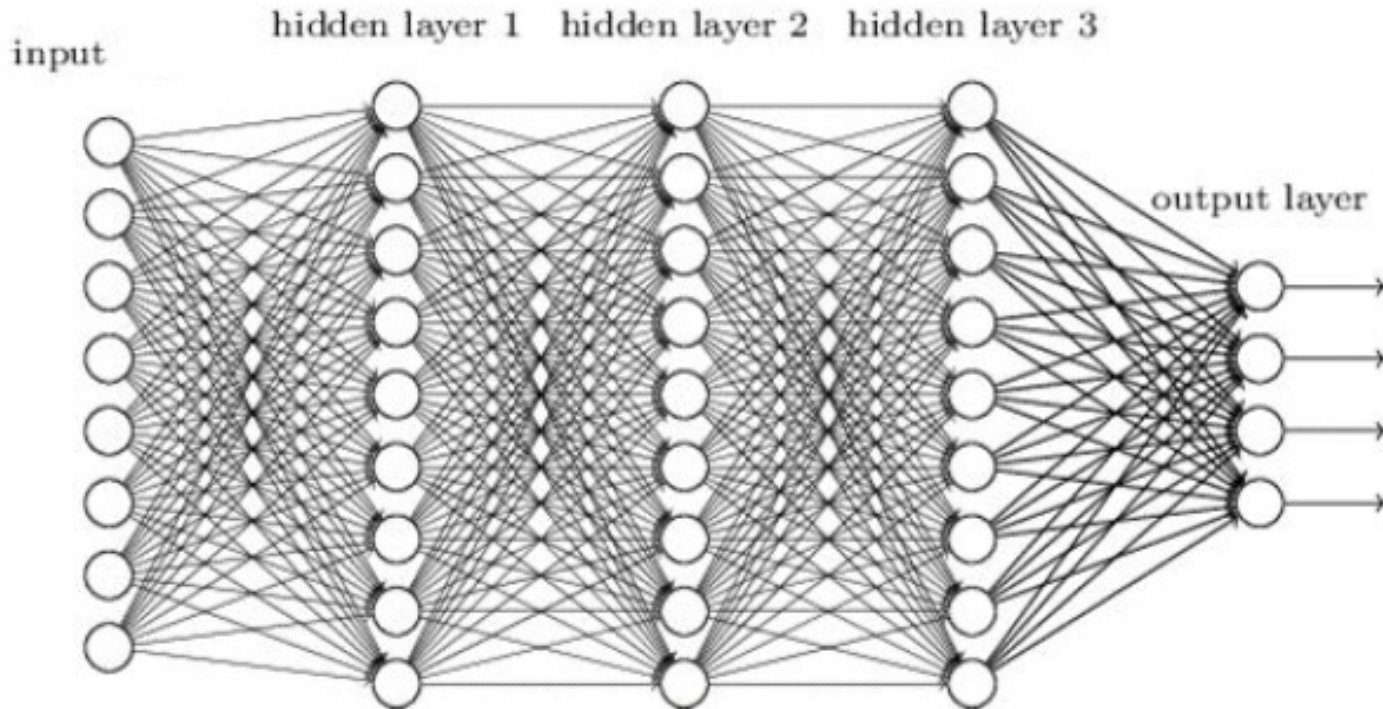


Multi-Layer Neural Network

(Multi-Class Output [MLP])



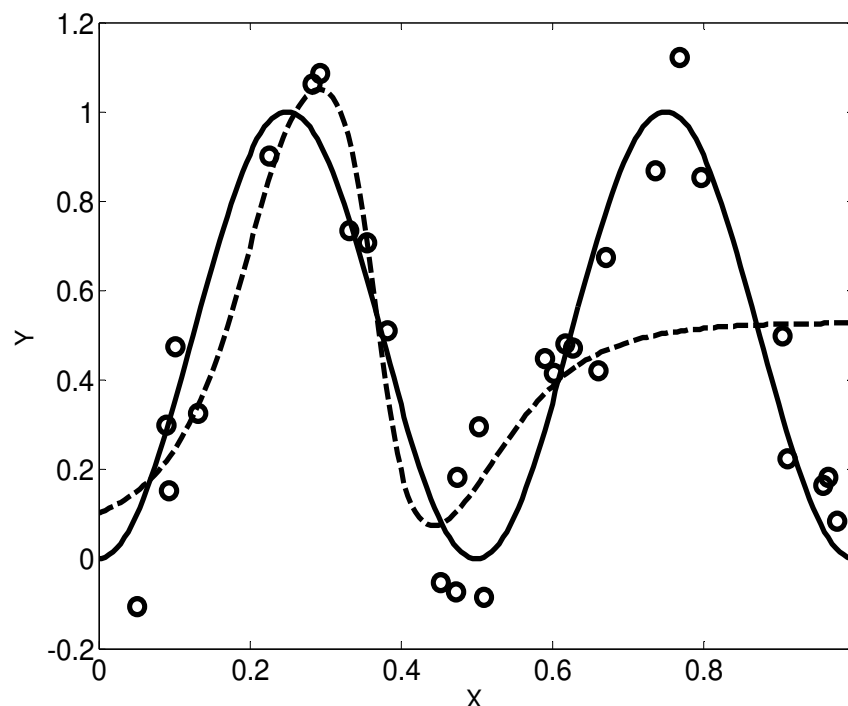
“Deep” Neural Networks (>1 Hidden Layer)



Researchers have successfully used 1000 layers to train an object classifier

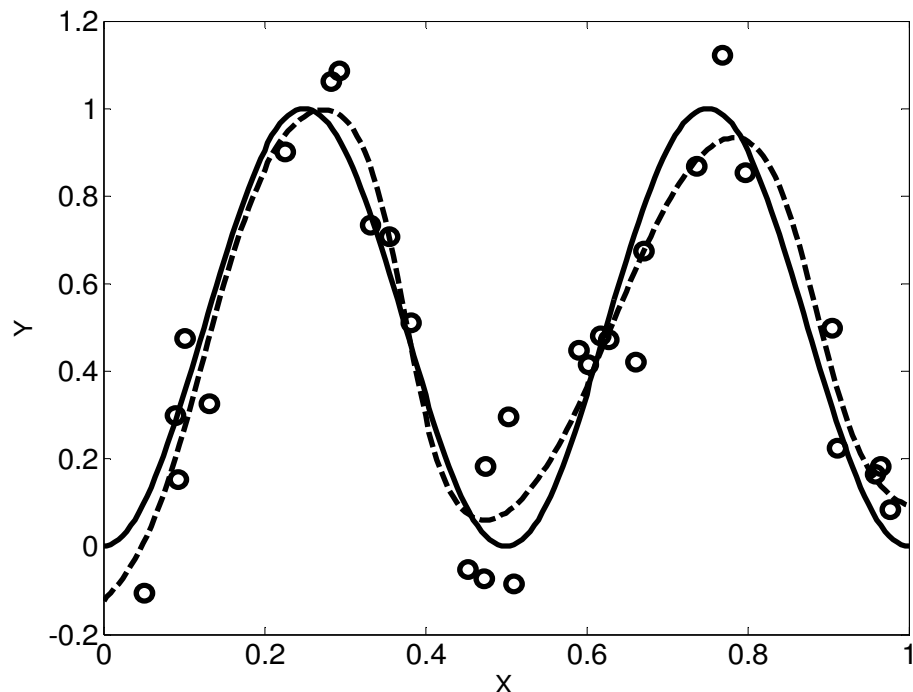
Example: Univariate Regression

- **Data set:** 30 samples generated using sine-squared target function with Gaussian noise (st. deviation 0.1).
- **MLP network**
(two hidden units)
underfitting



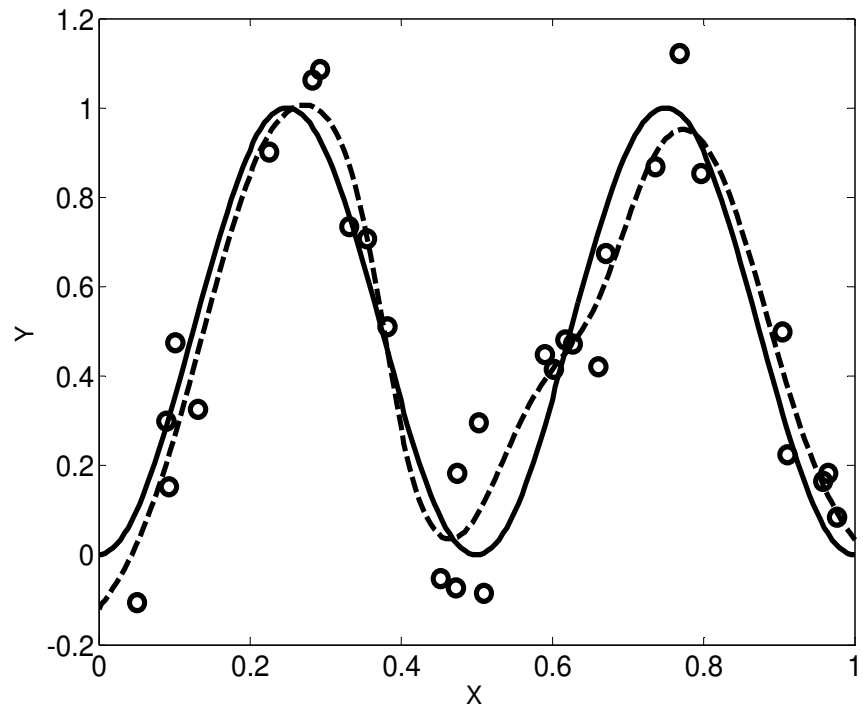
Example: Univariate Regression

- **Data set:** 30 samples generated using sine-squared target function with Gaussian noise (st. deviation 0.1).
- **MLP network**
(five hidden units)
near optimal



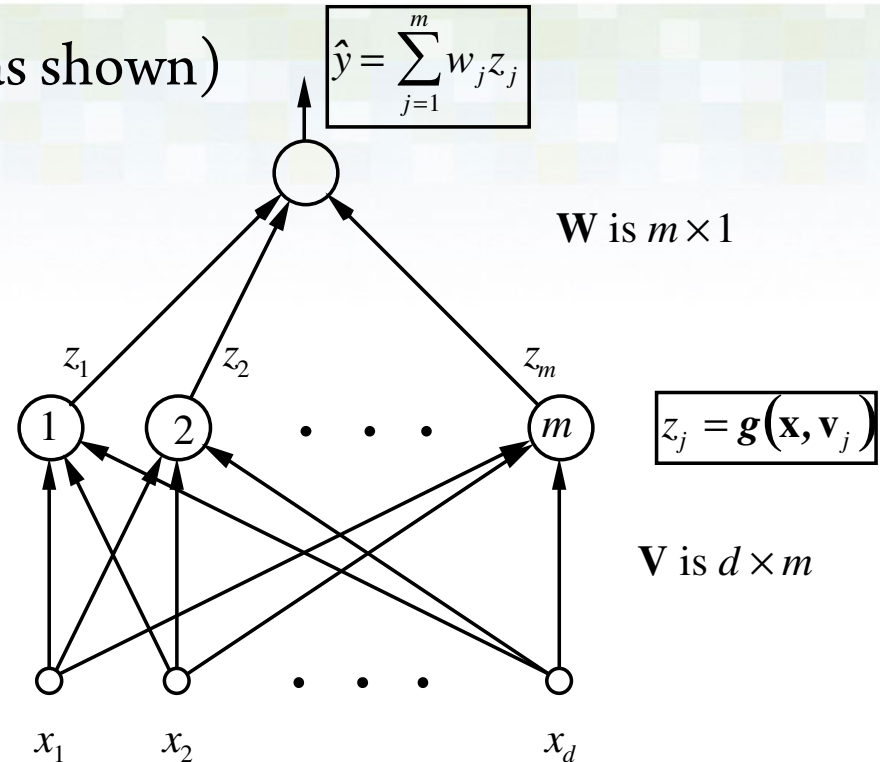
Example: Univariate Regression

- **Data set:** 30 samples generated using sine-squared target function with Gaussian noise (st. deviation 0.1).
- **MLP network**
(20 hidden units)
little overfitting



Backpropagation for classification

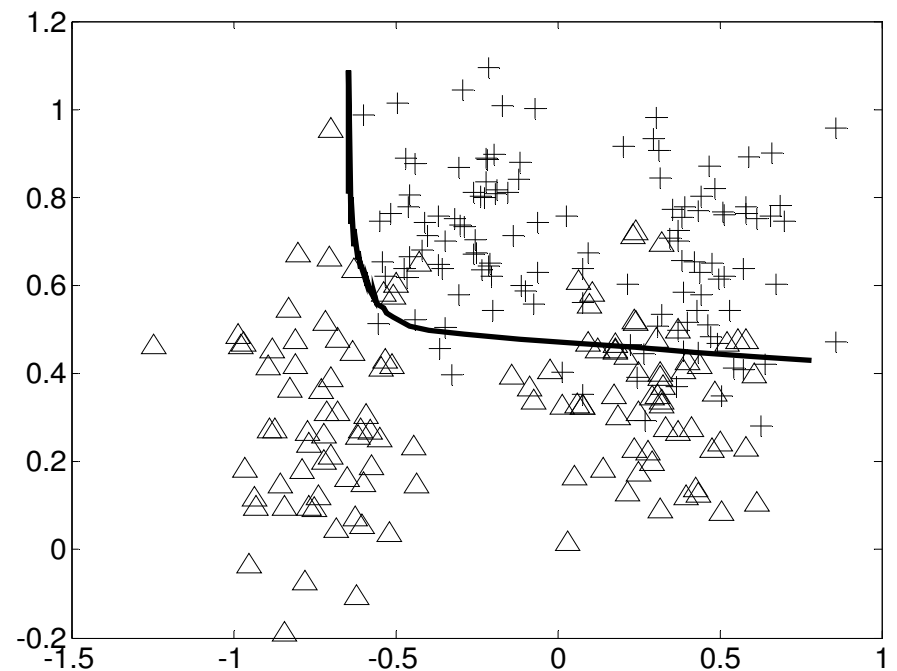
- Original MLP is for **regression** (as shown)



- For **classification**:
 - **sigmoid output unit** (\sim logistic regression using log-likelihood loss – see textbook)
 - during training, use **real-values 0/1** for class labels
 - during operation, **threshold** the output of a trained MLP classifier at **0.5** to predict class labels

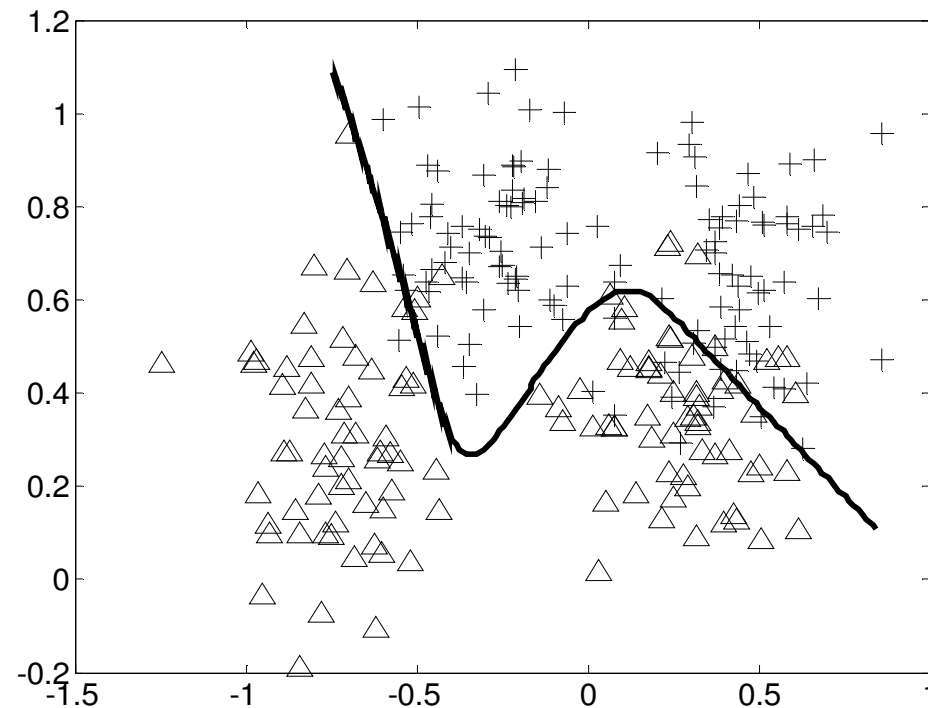
Classification Example (Ripley's Data set)

- **Data set:** 250 samples \sim mixture of gaussians, where
Class 0 data has centers $(-0.3, 0.7)$ and $(0.4, 0.7)$, and
Class 1 data has centers $(-0.7, 0.3)$ and $(0.3, 0.3)$.
- The variance of all gaussians is 0.03.
- MLP classifier
(two hidden units)
underfitting



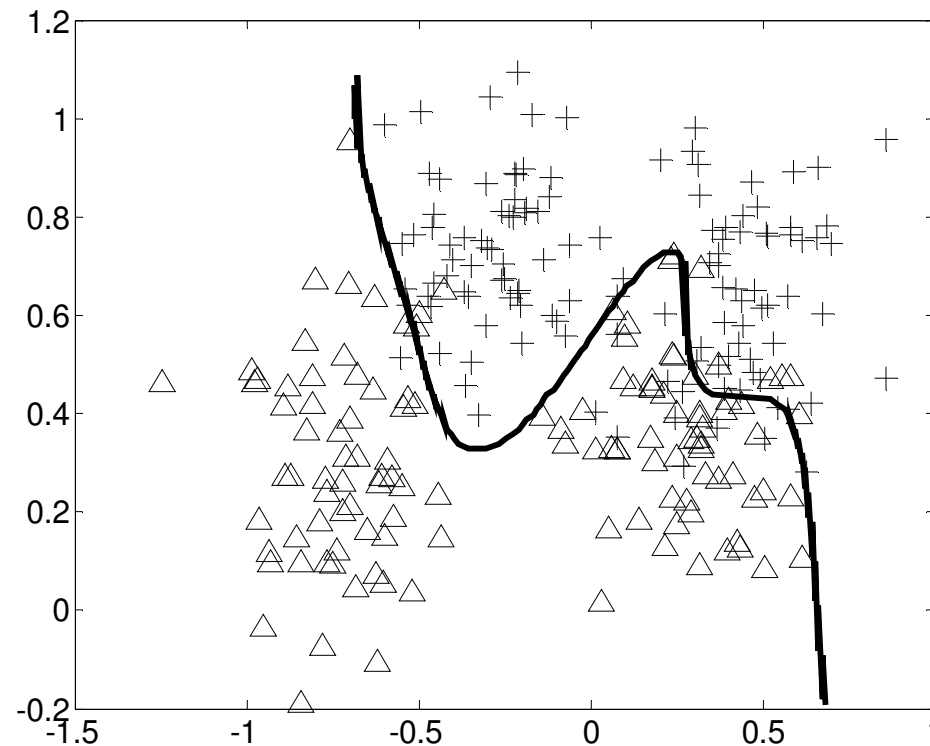
Classification Example

- MLP classifier (three hidden units)
- Near optimal solution



Classification Example

- MLP classifier (six hidden units)
- Some overfitting





References:

- ~History information thanks to R. Osuna @ TAMU ~ Pattern AnalysisSupport Vector Machines ~ A.W.Moore tutorials ~ cs.cmu.edu/~awm
- ~Perceptron and Perceptron delta rule example adapted with thanks from Dr. H. Wechsler, GMU
- ~Perceptrons and Neural Networks ~ M.Veloso ~ CMU
- ~~Support Vector Machine ~ M.Tan ~ UBC
- ~NN and Deep Learning ~ Dr.Qi