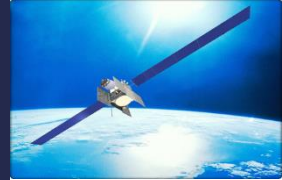


Introduction to Neural Networks

Dr. Fayyaz ul Amir Afsar Minhas

fayyazafsar@gmail.com

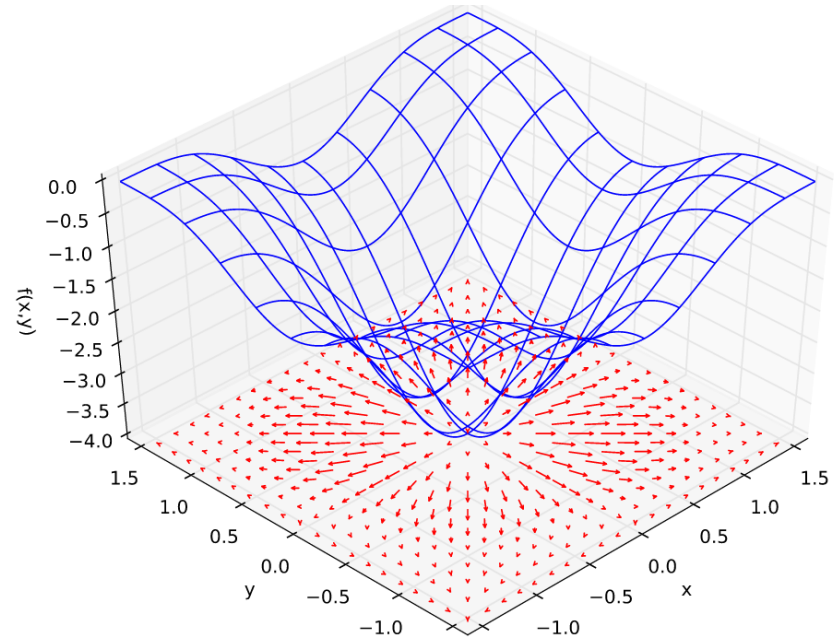
**Department of Computer and Information Sciences
Pakistan Institute of Engineering and Applied Sciences
(PIEAS)
P.O. Nilore, Islamabad.**

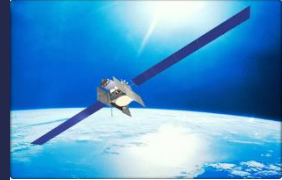
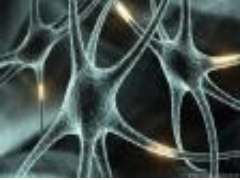


Basics

- **Gradient**
 - Generalization of the slope to multidimensional functions

- $\nabla f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$



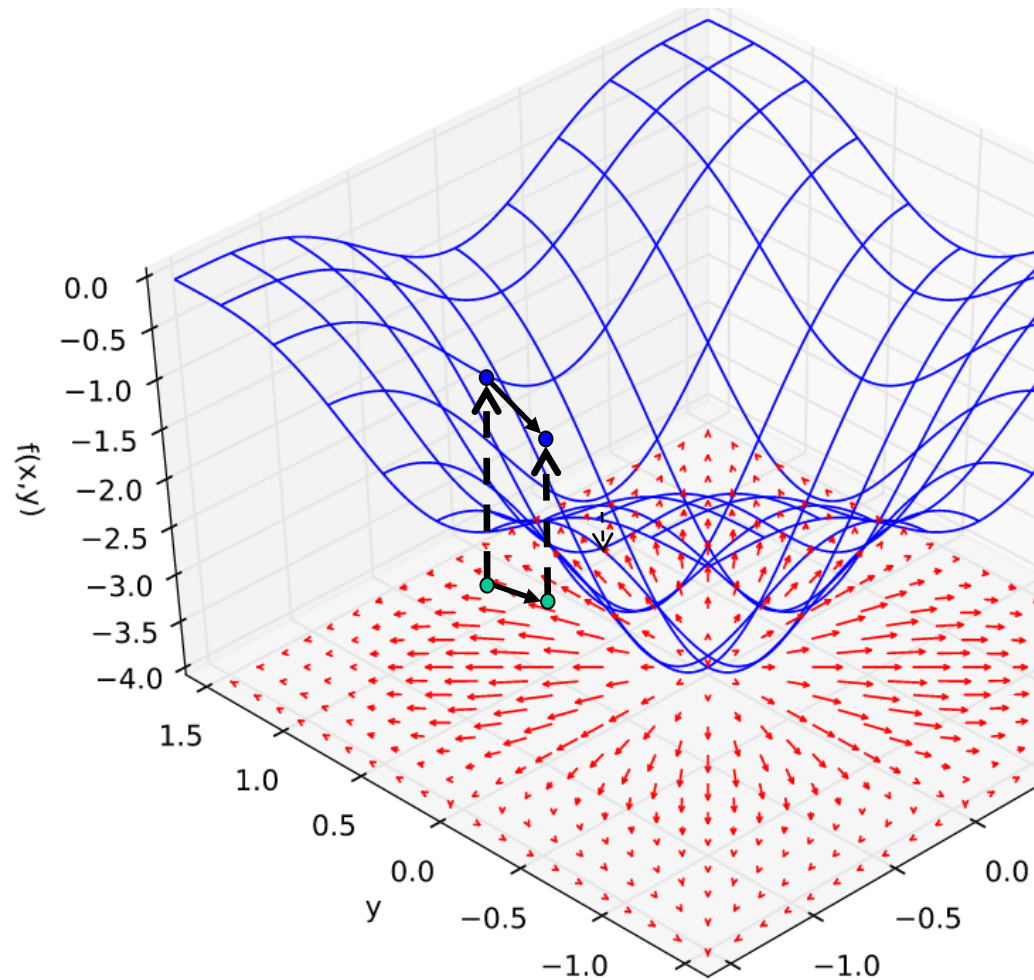


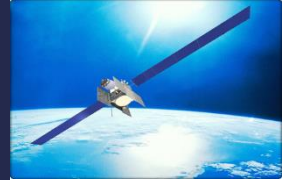
Basics

- **Gradient Descent**
 - A method for optimization
 - To find the minima of a function, take a step in the direction opposite to the gradient

$$x_i = x_{i-1} + \alpha \nabla f(x)$$

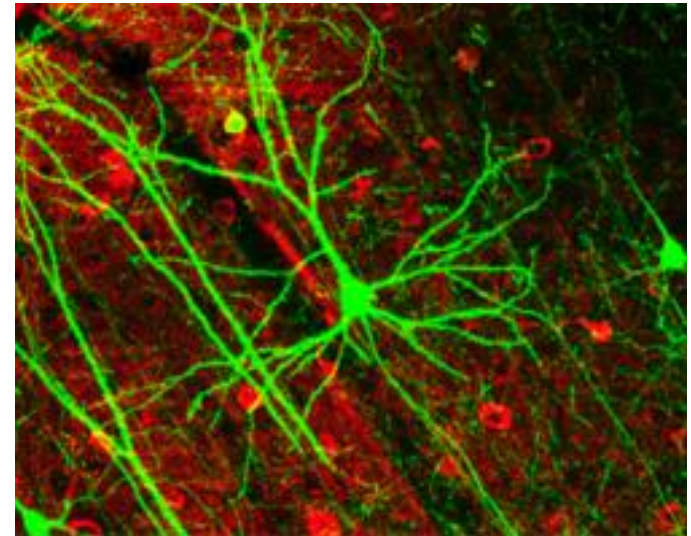
- **Local minimas?**

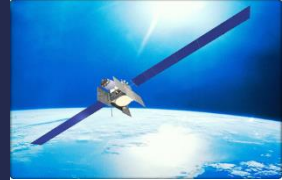




The Human Brain: Neurons and Nerve Cells

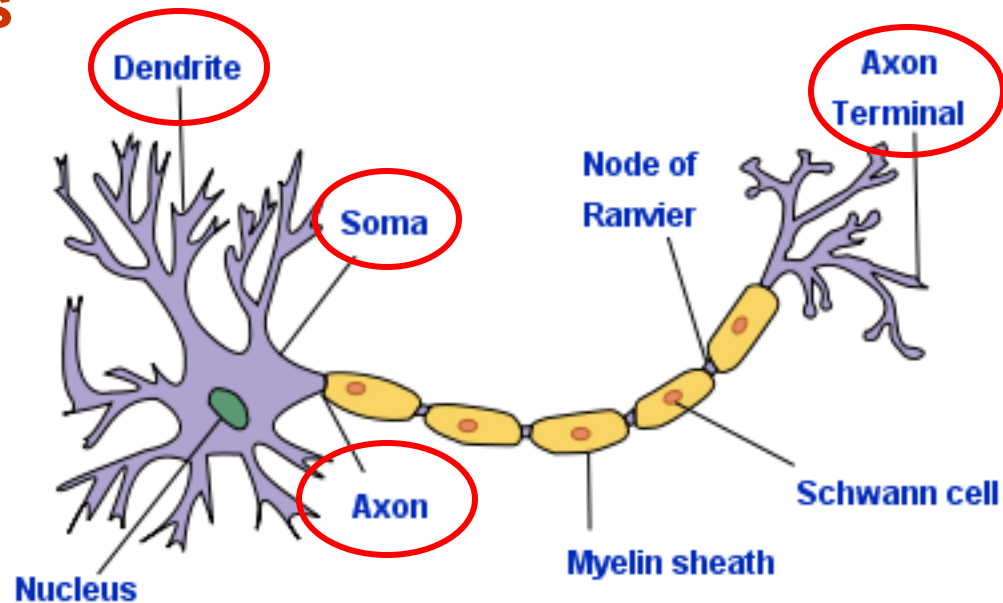
- **Most complex organ in the human body**
- Contains some **10^{10} neurons**, which are capable of electrical and chemical communication with tens of thousands of other nerve cells
- Nerve cells in turn rely on some quadrillion (**10^{15}**) **synaptic connections** for their communications.

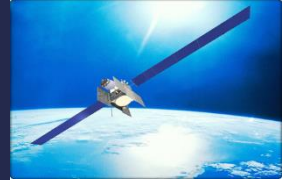




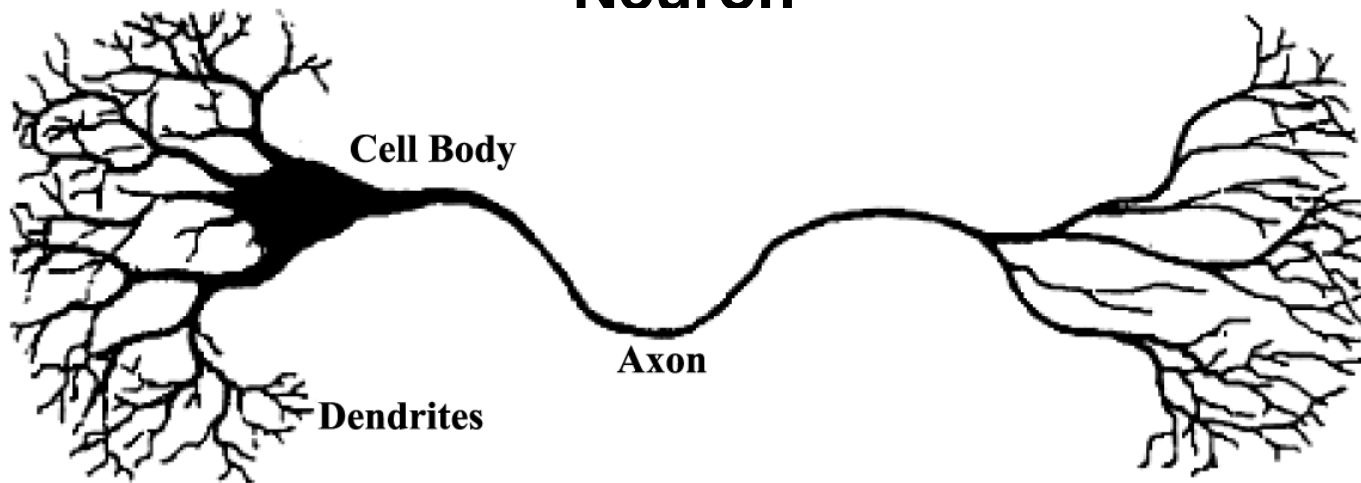
Functioning of the Biological Neuron

- **Electrically Excitable Cells**
- **Process and Transmit Information**
- **Major Parts**
 - **Soma (3-18um)**
 - Cell Body
 - **Dendrites**
 - Receive Inputs from other Neurons
 - **Axon**
 - Transmit Output to other Neurons

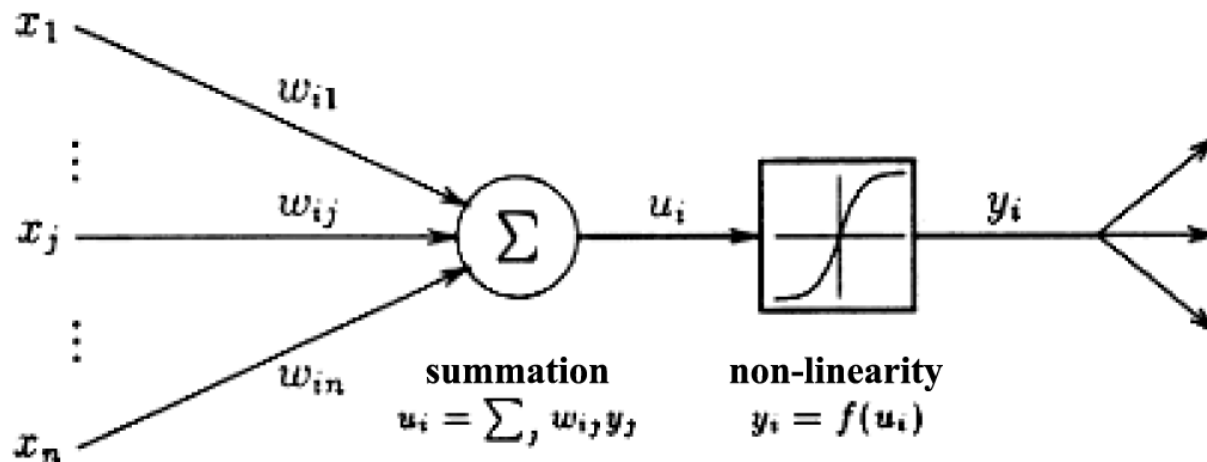




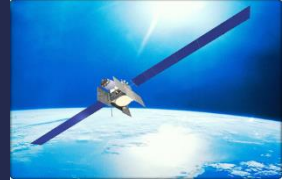
Structural Mathematical Model for the Biological Neuron



(a)



(b)



Perceptron

- Given:

- Training data and labels

$$y_{in} = w_1 X_1 + w_2 X_2 + \dots + w_n X_n + b = W^T X + b$$

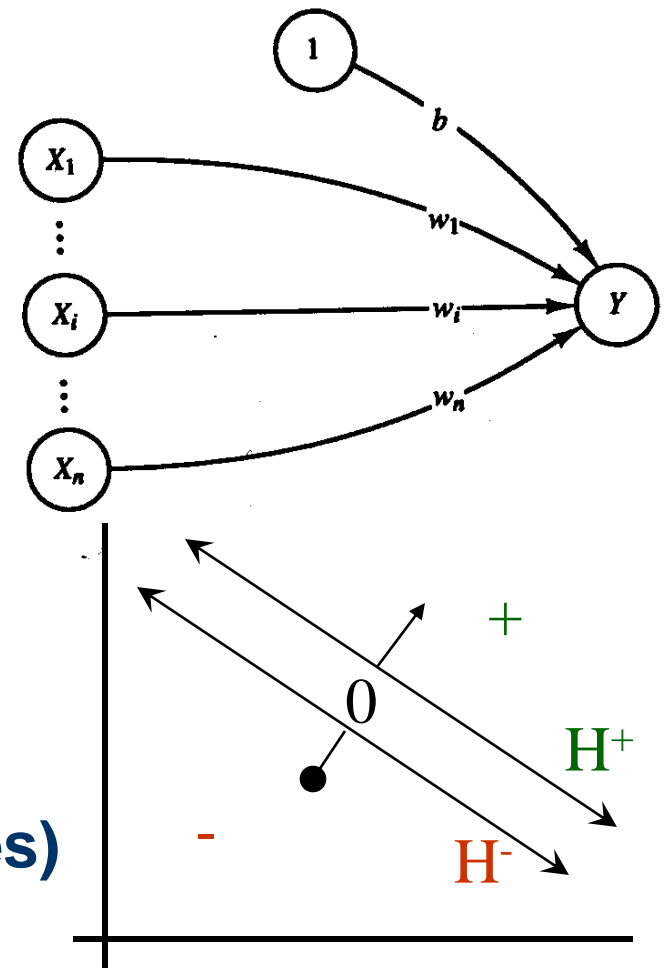
$$W = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_n \end{bmatrix}$$

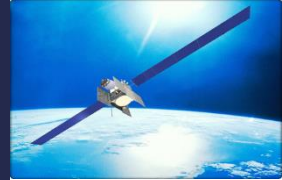
$$Y = \begin{cases} +1 & W^T X + b > \theta \\ 0 & -\theta \leq W^T X + b \leq \theta \\ -1 & W^T X + b < -\theta \end{cases}$$

- Thus there are two hyperplanes)

- $H^+ : W^T X + b = \theta$

- $H^- : W^T X + b = -\theta$





Learning Algorithm

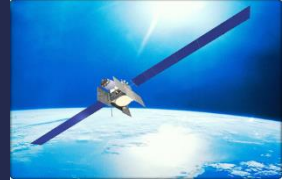
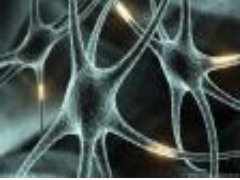
- Step 0.** Initialize weights and bias.
(For simplicity, set weights and bias to zero.)
Set learning rate α ($0 < \alpha \leq 1$).
(For simplicity, α can be set to 1.)
- Step 1.** While stopping condition is false, do Steps 2–6.
- Step 2.** For each training pair $s:t$, do Steps 3–5.
- Step 3.** Set activations of input units:
$$x_i = s_i.$$
- Step 4.** Compute response of output unit:
$$y_in = b + \sum_i x_i w_i;$$
$$y = \begin{cases} 1 & \text{if } y_in > \theta \\ 0 & \text{if } -\theta \leq y_in \leq \theta \\ -1 & \text{if } y_in < -\theta \end{cases}$$
- Step 5.** Update weights and bias if an error occurred for this pattern.
If $y \neq t$,
$$w_i(\text{new}) = w_i(\text{old}) + \alpha t x_i,$$
$$b(\text{new}) = b(\text{old}) + \alpha t$$

else
$$w_i(\text{new}) = w_i(\text{old}),$$
$$b(\text{new}) = b(\text{old}).$$
- Step 6.** Test stopping condition:
If no weights changed in Step 2, stop; else, continue.

Epoch

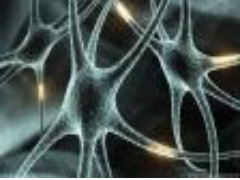
Update Occurs only when there is an error

If output is -1 and target is +1, we must increase the net input: Achieves this by increasing weight by alpha when if the input is +1 or decreasing weight if the input is -1 or not changing it when input is 0



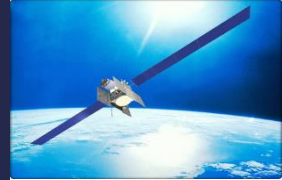
Example: AND Gate, $\theta=0.2$, $\alpha=1$

x_1	x_2	1	y_{net}	y	T	dw_1	dw_2	db	$w_1 = w_1 + dw_1$	$w_2 = w_2 + dw_2$	$b = b + db$
									0	0	0
1	1	1	0	0	1	1	1	1	1	1	1
1	0	1	2	1	-1	-1	0	-1	0	1	0
0	1	1	1	1	-1	0	-1	-1	0	0	-1
0	0	1	-1	-1	-1	0	0	-1	0	0	-1
1	1	1	-1	-1	1	1	1	1	1	1	0
1	0	1	1	1	-1	-1	0	-1	0	1	-1
0	1	1	0	0	-1	0	-1	-1	0	0	-2
0	0	1	-2	-1	-1	0	0	-1	0	0	-2
1	1	1	-2	-1	1	1	1	1	1	1	-1
1	0	1	0	0	-1	-1	0	-1	0	1	-2
0	1	1	-1	-1	-1	0	-1	-1	0	1	-2
0	0	1	-2	-1	-1	0	0	-1	0	1	-2
1	1	1	-1	-1	1	1	1	1	1	2	-1
1	0	1	0	0	-1	-1	0	-1	0	2	-2
0	1	1	0	0	-1	0	-1	-1	0	1	-3
0	0	1	-3	-1	-1	0	0	-1	0	1	-3
1	1	1	-2	-1	1	1	1	1	1	2	-2
1	0	1	-1	-1	-1	-1	0	-1	1	2	-2
0	1	1	0	0	-1	0	-1	-1	1	1	-3
0	0	1	-3	-1	-1	0	0	-1	1	1	-3

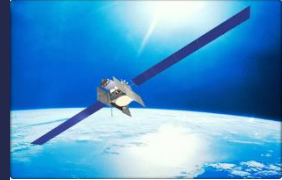
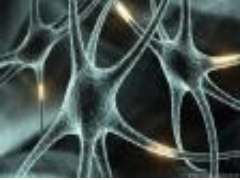


CIS529: PATTERN CLASSIFICATION & RECOGNITION

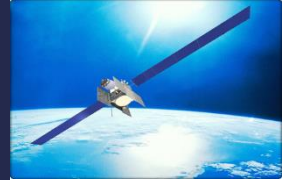
Pakistan Institute of Engineering and Applied Sciences (PIEAS).



x_1	x_2	1	y_{net}	y	T	dw_1	dw_2	db	$w_1 = w_1 + dw_1$	$w_2 = w_2 + dw_2$	$b = b + db$
									1	1	-3
1	1	1	-1	-1	1	1	1	1	2	2	-2
1	0	1	0	0	-1	-1	0	-1	1	2	-3
0	1	1	-1	-1	-1	0	-1	-1	1	2	-3
0	0	1	-3	-1	-1	0	0	-1	1	2	-3
1	1	1	0	0	1	1	1	1	2	3	-2
1	0	1	0	0	-1	-1	0	-1	1	3	-3
0	1	1	0	0	-1	0	-1	-1	1	2	-4
0	0	1	-4	-1	-1	0	0	-1	1	2	-4
1	1	1	-1	-1	1	1	1	1	2	3	-3
1	0	1	-1	-1	-1	-1	0	-1	2	3	-3
0	1	1	0	0	-1	0	-1	-1	2	2	-4
0	0	1	-4	-1	-1	0	0	-1	2	2	-4
1	1	1	0	0	1	1	1	1	3	3	-3
1	0	1	0	0	-1	-1	0	-1	2	3	-4
0	1	1	-1	-1	-1	0	-1	-1	2	3	-4
0	0	1	-4	-1	-1	0	0	-1	2	3	-4
1	1	1	1	1	1	1	1	1	2	3	-4
1	0	1	-2	-1	-1	-1	0	-1	2	3	-4
0	1	1	-1	-1	-1	0	-1	-1	2	3	-4
0	0	1	-4	-1	-1	0	0	-1	2	3	-4

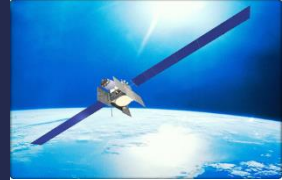


Videos!



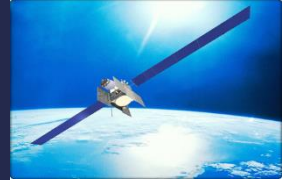
Observations on Perceptron

- Learning rate impacts the speed of learning
- Perceptron was unable to learn the XOR problem
- Perceptron learning rule convergence theorem
 - If the data is linearly separable, you can always use a perceptron algorithm to find a separating hyperplane



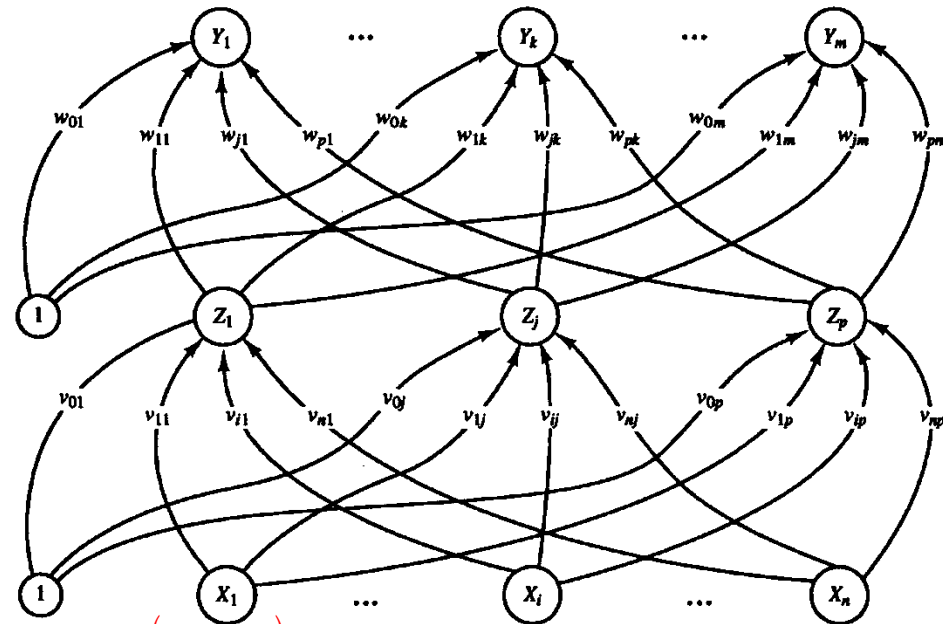
Feed Forward Backpropagation Neural Networks aka Multi-layer Perceptrons

- **Very General Nature**
 - **Applicable to a variety of practical problems**
 - NETtalk
 - Signature Classification
 - Disease Classification
 - Hand Written Character Recognition
 - Combat Outcome Predication
 - Earthquake Prediction
 - Etc...
- **Objective:**
 - **To achieve a balance between Memorization and Generalization**
 - **Memorization:** Ability to respond correctly to the input patterns used for training
 - **Generalization:** Ability to give reasonable responses to input that is similar but not identical to that used in training



Architecture

- **Consists of multiple layers**
- **Layers of units other than the input and output are called hidden units**
- **Unidirectional weight connections and biases**
- **Activation functions**
 - **Use of sigmoid functions**
 - **Nonlinear Operation: Ability to solve practical problems**
 - **Differentiable: Makes theoretical assessment easier**
 - **Derivative can be expressed in terms of functions themselves: Computational Efficiency**
 - **Activation function is the same for all neurons in the same layer**
 - **Input layer just passes on the signal without processing (linear operation)**

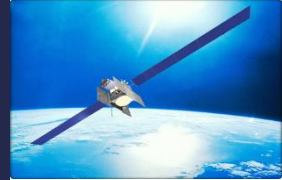


$$z_j = f(z_in_j)$$

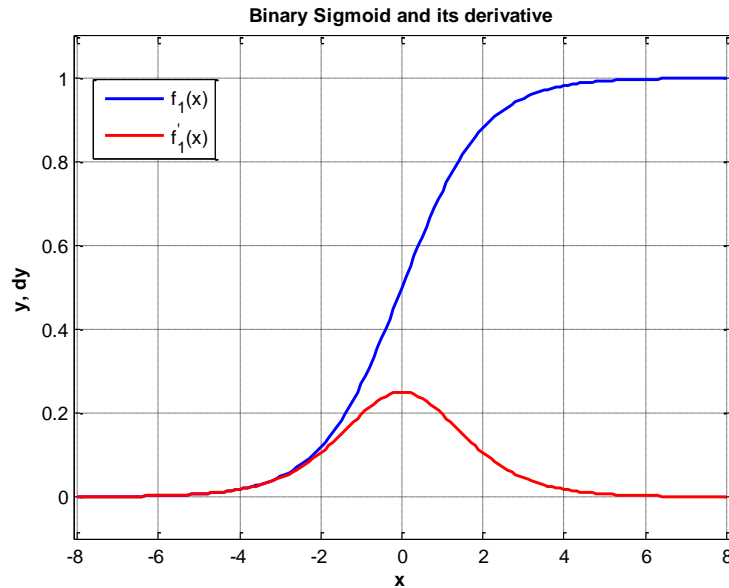
$$z_in_j = \sum_{i=0}^n x_i v_{ij}, \quad x_0 = 1, \quad j = 1 \dots p$$

$$y_k = f(y_in_k)$$

$$y_in_k = \sum_{j=0}^p z_j w_{jk}, \quad z_0 = 1, \quad k = 1 \dots m$$

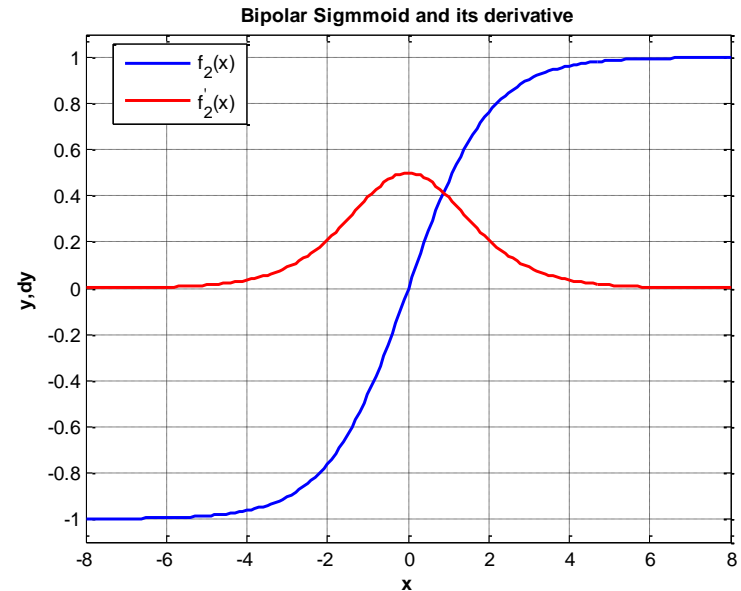


Architecture: Activation functions



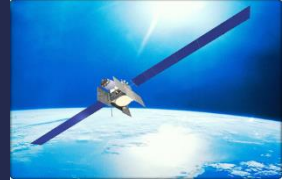
$$f_1(x) = \frac{1}{1 + \exp(-x)}$$

$$f_1'(x) = f_1(x)[1 - f_1(x)]$$



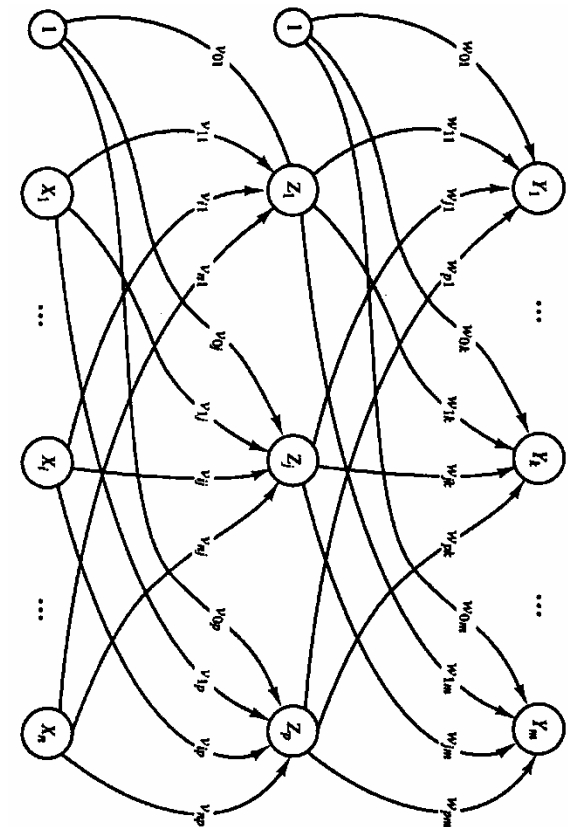
$$f_2(x) = \frac{2}{1 + \exp(-x)} - 1$$

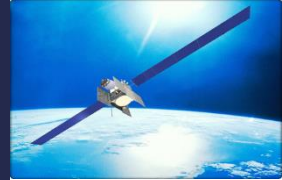
$$f_2'(x) = \frac{1}{2} [1 + f_2(x)][1 - f_2(x)]$$



Training

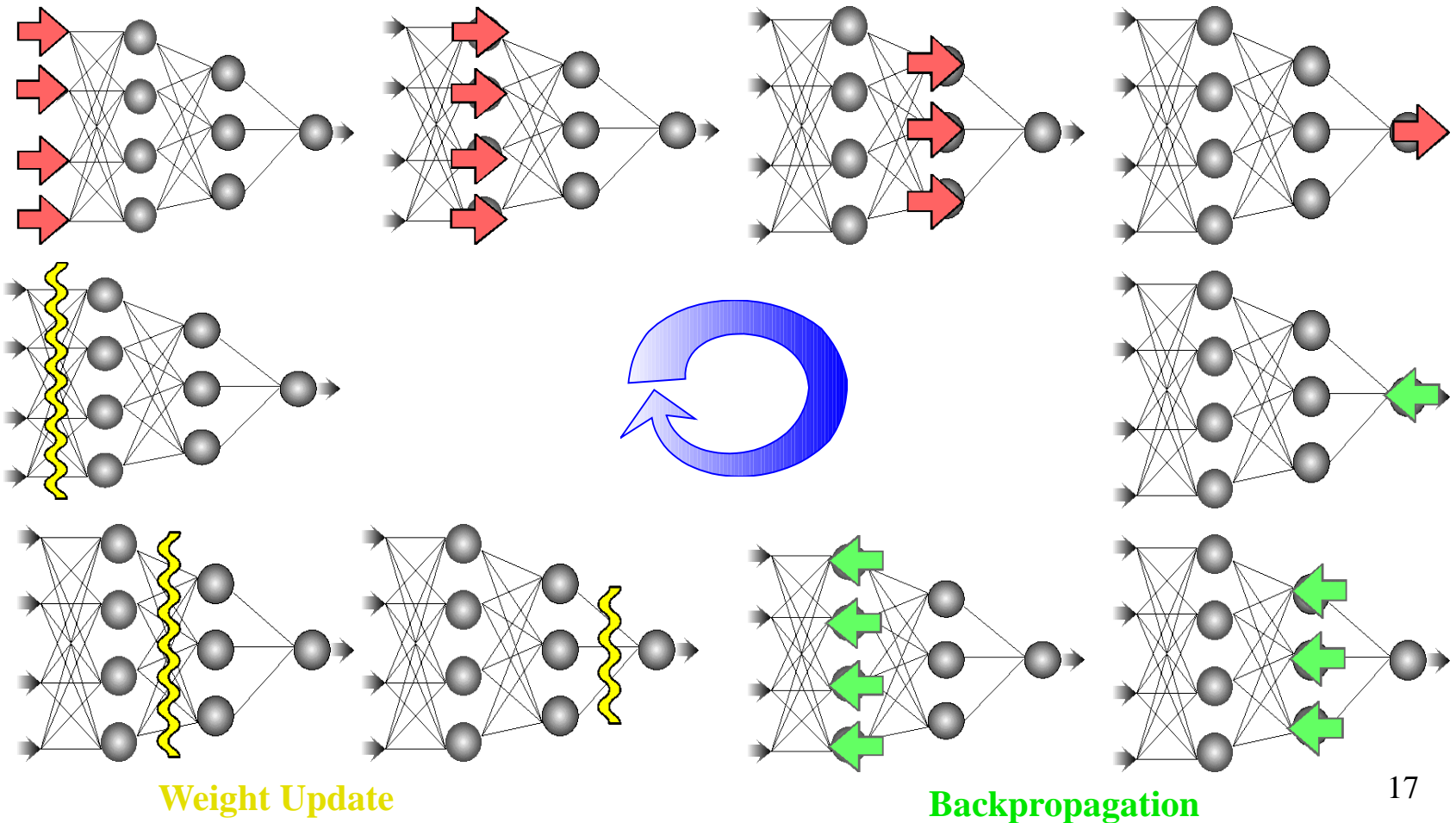
- During training we are presented with input patterns and their targets
- At the output layer we can compute the error between the targets and actual output and use it to compute weight updates through the Delta Rule
- But the Error cannot be calculated at the hidden input as their targets are not known
- Therefore we propagate the error at the output units to the hidden units to find the required weight changes (**Backpropagation**)
- 3 Stages
 - Feed-forward of the input training pattern
 - Calculation and **Backpropagation** of the associated error
 - **Weight Adjustment**
- Based on minimization of SSE (Sum of Square Errors)

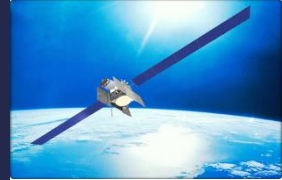




Backpropagation training cycle

Feed forward





Proof for the Learning Rule

$$E = .5 \sum_k [t_k - y_k]^2.$$

By use of the chain rule, we have

$$\begin{aligned} \frac{\partial E}{\partial w_{JK}} &= \frac{\partial}{\partial w_{JK}} .5 \sum_k [t_k - y_k]^2 \\ &= \frac{\partial}{\partial w_{JK}} .5 [t_K - f(y_{in_K})]^2 \\ &= -[t_K - y_K] \frac{\partial}{\partial w_{JK}} f(y_{in_K}) \\ &= -[t_K - y_K] f'(y_{in_K}) \frac{\partial}{\partial w_{JK}} (y_{in_K}) \\ &= -[t_K - y_K] f'(y_{in_K}) z_j. \end{aligned}$$

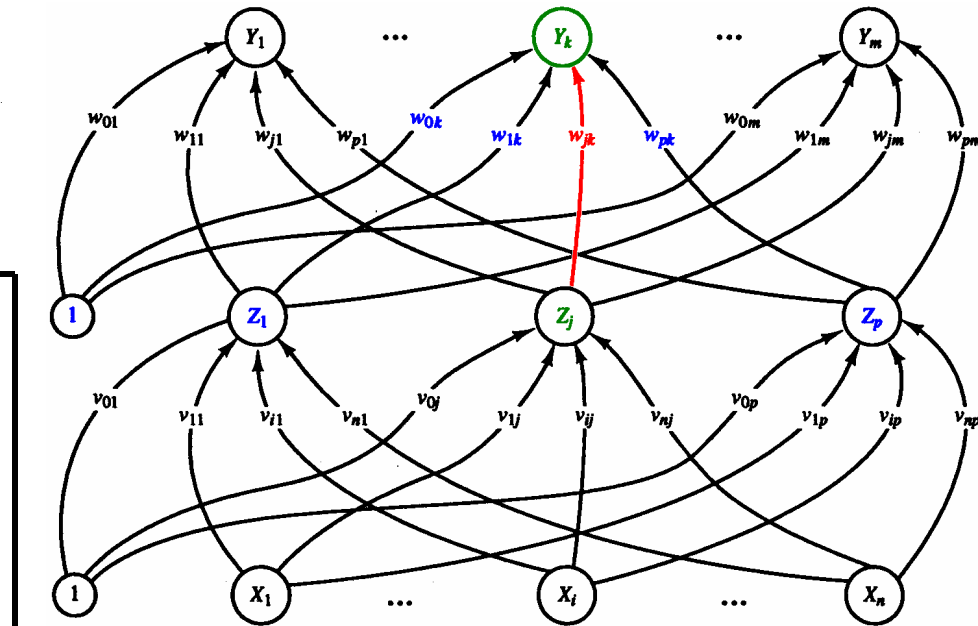
It is convenient to define δ_K :

$$\delta_K = [t_K - y_K] f'(y_{in_K}).$$

$$\Delta w_{jk} = -\alpha \frac{\partial E}{\partial w_{jk}}$$

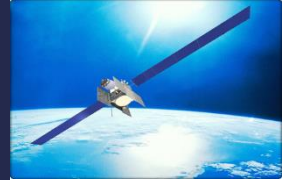
$$= \alpha [t_k - y_k] f'(y_{in_k}) z_j$$

$$= \alpha \delta_k z_j;$$



Change in w_{jk} affects only Y_k

Use of Gradient Descent Minimization



Proof for the Learning Rule...

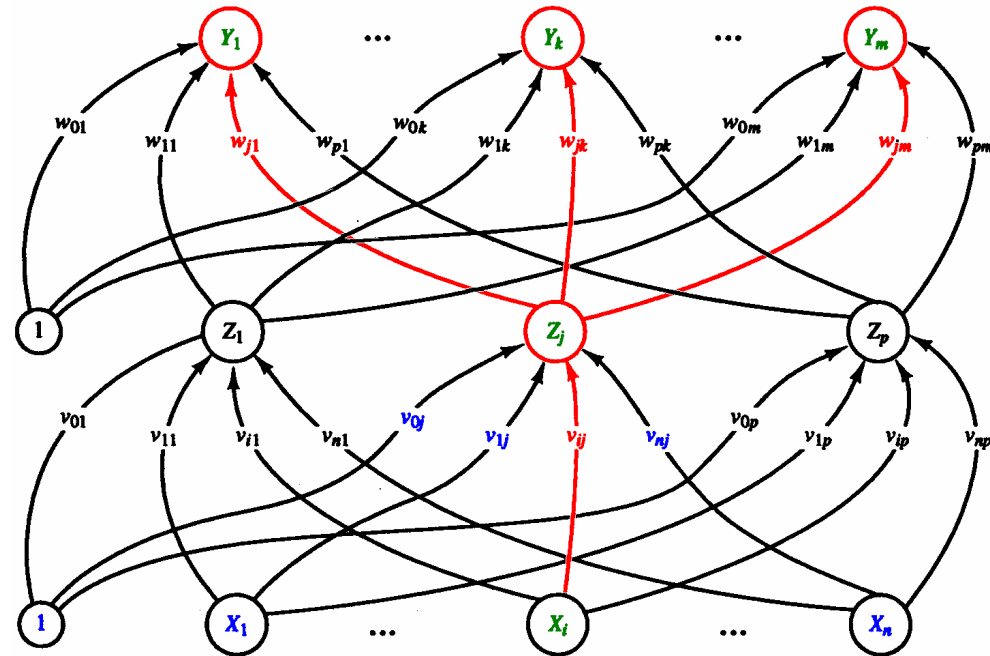
For weights on connections to the hidden unit Z_j :

$$\begin{aligned}\frac{\partial E}{\partial v_{IJ}} &= -\sum_k [t_k - y_k] \frac{\partial}{\partial v_{IJ}} y_k \\ &= -\sum_k [t_k - y_k] f'(y_{in_k}) \frac{\partial}{\partial v_{IJ}} y_{in_k} \\ &= -\sum_k \delta_k \frac{\partial}{\partial v_{IJ}} y_{in_k} \\ &= -\sum_k \delta_k w_{jk} \frac{\partial}{\partial v_{IJ}} z_j \\ &= -\sum_k \delta_k w_{jk} f'(z_{in_j}) [x_i].\end{aligned}$$

Define:

$$\delta_j = \sum_k \delta_k w_{jk} f'(z_{in_j})$$

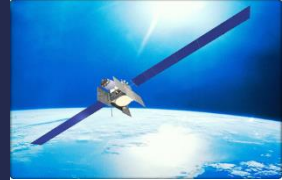
$$\begin{aligned}\Delta v_{ij} &= -\alpha \frac{\partial E}{\partial v_{ij}} \\ &= \alpha f'(z_{in_j}) x_i \sum_k \delta_k w_{jk}, \\ &= \alpha \delta_j x_i.\end{aligned}$$



Change in v_{ij} affects all $Y_{1..m}$

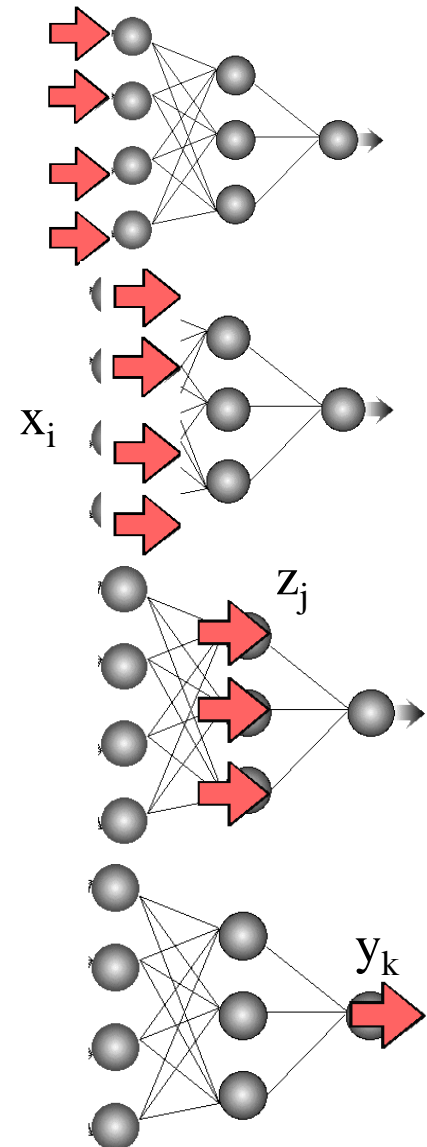
Change in v_{ij} affects only z_j

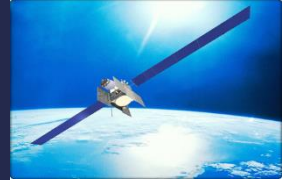
Use of Gradient Descent Minimization



Training Algorithm

- Step 0.** Initialize weights.
(Set to small random values).
- Step 1.** While stopping condition is false, do Steps 2–9.
- Step 2.** For each training pair, do Steps 3–8.
- Feedforward:**
- Step 3.** Each input unit ($X_i, i = 1, \dots, n$) receives input signal x_i and broadcasts this signal to all units in the layer above (the hidden units).
- Step 4.** Each hidden unit ($Z_j, j = 1, \dots, p$) sums its weighted input signals,
- $$z_in_j = v_{0j} + \sum_{i=1}^n x_i v_{ij},$$
- applies its activation function to compute its output signal,
- $$z_j = f(z_in_j),$$
- and sends this signal to all units in the layer above (output units).
- Step 5.** Each output unit ($Y_k, k = 1, \dots, m$) sums its weighted input signals,
- $$y_in_k = w_{0k} + \sum_{j=1}^p z_j w_{jk}$$
- and applies its activation function to compute its output signal,
- $$y_k = f(y_in_k).$$





Training Algorithm...

Backpropagation of error:

Step 6. Each output unit ($Y_k, k = 1, \dots, m$) receives a target pattern corresponding to the input training pattern, computes its error information term,

$$\delta_k = (t_k - y_k)f'(y_{in_k}),$$

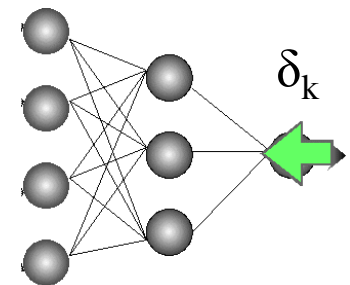
calculates its weight correction term (used to update w_{jk} later),

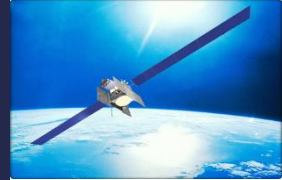
$$\Delta w_{jk} = \alpha \delta_k z_j,$$

calculates its bias correction term (used to update w_{0k} later),

$$\Delta w_{0k} = \alpha \delta_k,$$

and sends δ_k to units in the layer below.





Training Algorithm...

Step 7. Each hidden unit ($Z_j, j = 1, \dots, p$) sums its delta inputs (from units in the layer above),

$$\delta_in_j = \sum_{k=1}^m \delta_k w_{jk},$$

multiplies by the derivative of its activation function to calculate its error information term,

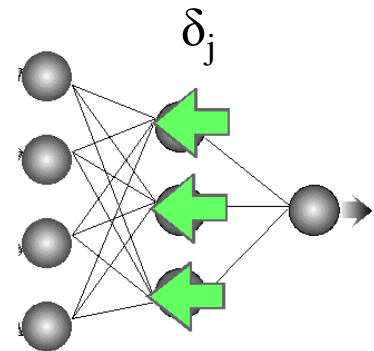
$$\delta_j = \delta_in_j f'(z_in_j),$$

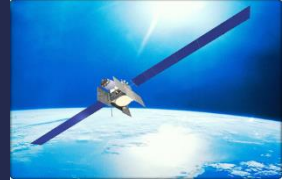
calculates its weight correction term (used to update v_{ij} later),

$$\Delta v_{ij} = \alpha \delta_j x_i,$$

and calculates its bias correction term (used to update v_{0j} later),

$$\Delta v_{0j} = \alpha \delta_j.$$





Training Algorithm...

Update weights and biases:

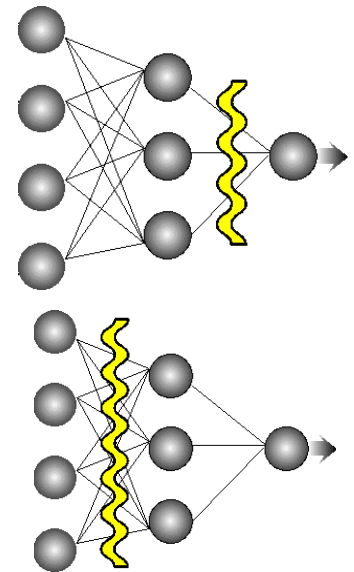
Step 8. Each output unit ($Y_k, k = 1, \dots, m$) updates its bias and weights ($j = 0, \dots, p$):

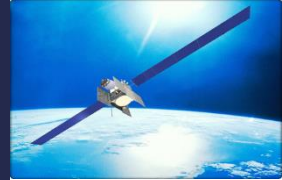
$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}.$$

Each hidden unit ($Z_j, j = 1, \dots, p$) updates its bias and weights ($i = 0, \dots, n$):

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}.$$

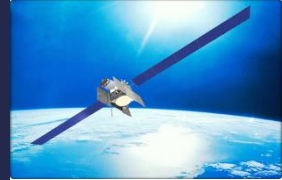
Step 9. Test stopping condition.





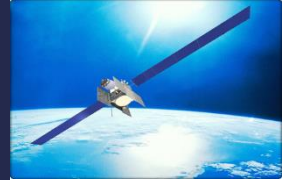
Effect of Learning Rate

- **Controls the change in synaptic weights**
- **The smaller the learning rate the smoother the trajectory in the weight space**
 - **Slower rate of learning**
- **If learning rate is made too large (for speedy convergence) the network may become unstable (oscillatory)**



Stopping Criterion

- **The Backpropagation (BP) algorithm cannot be shown to converge**
 - No well defined criterion for stopping its operation
- **Criterion Used**
 - BP is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small threshold
 - Ideally Gradient is zero at the minima
 - **Drawbacks**
 - For successful trials, learning times may be large
 - Calculation of the gradient is required
- BP is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small
 - May result in premature termination of the learning process



Application Procedure

- **Involves only the feed-forward phase (fast!)**

Step 0. Initialize weights (from training algorithm).

Step 1. For each input vector, do Steps 2–4.

Step 2. For $i = 1, \dots, n$: set activation of input unit

$$x_i;$$

Step 3. For $j = 1, \dots, p$:

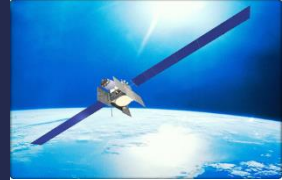
$$z_in_j = v_{0j} + \sum_{i=1}^n x_i v_{ij};$$

$$z_j = f(z_in_j).$$

Step 4. For $k = 1, \dots, m$:

$$y_in_k = w_{0k} + \sum_{j=1}^p z_j w_{jk};$$

$$y_k = f(y_in_k).$$



Solution of the XOR Problem

■ Initial weights

$$v_{01} = -0.8$$

$$v_{11} = 0.5$$

$$v_{12} = 0.9$$

$$v_{02} = +0.1$$

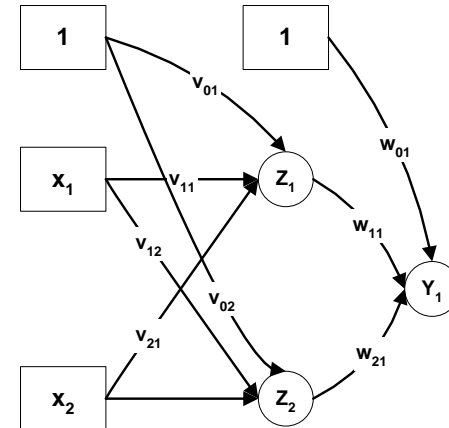
$$v_{21} = 0.4$$

$$v_{22} = 1$$

$$w_{01} = -0.3$$

$$w_{11} = -1.2$$

$$w_{21} = 1.1$$

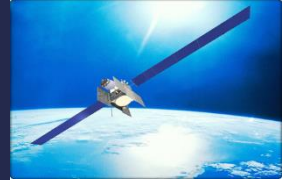


$$z_j = f(z_in_j)$$

$$z_in_j = \sum_{i=0}^2 x_i v_{ij}, \quad x_0 = 1, \quad j = 1 \dots 2$$

$$y_k = f(y_in_k)$$

$$y_in_k = \sum_{j=0}^2 z_j w_{jk}, \quad z_0 = 1, \quad k = 1$$



Solution of the XOR Problem...

- We consider the input pattern as (1,1) with target = 0 and $\alpha=0.1$
- Feed Forward

$$z_1 = \text{sigmoid}(V_1^T X) = 1 / \left[1 + e^{-(1 \cdot 0.5 + 1 \cdot 0.4 - 1 \cdot 0.8)} \right] = 0.5250$$

$$z_2 = \text{sigmoid}(V_2^T X) = 1 / \left[1 + e^{-(1 \cdot 0.9 + 1 \cdot 1.0 + 1 \cdot 0.1)} \right] = 0.8808$$

$$y_1 = \text{sigmoid}(W^T Z) = 1 / \left[1 + e^{-(0.5250 \cdot 1.2 + 0.8808 \cdot 1.1 - 1 \cdot 0.3)} \right] = 0.5097$$

- Backpropagation

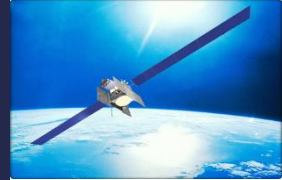
$$e = t - y_1 = 0 - 0.5097 = -0.5097$$

$$\delta_{k=1} = y_1 (1 - y_1) e = 0.5097 \cdot (1 - 0.5097) \cdot (-0.5097) = -0.1274$$

$$\Delta w_{21} = \alpha \cdot z_2 \cdot \delta_{k=1} = 0.1 \cdot 0.8808 \cdot (-0.1274) = -0.0112$$

$$\Delta w_{11} = \alpha \cdot z_1 \cdot \delta_{k=1} = 0.1 \cdot 0.5250 \cdot (-0.1274) = -0.0067$$

$$\Delta w_{01} = \alpha \cdot (1) \cdot \delta_{k=1} = 0.1 \cdot (+1) \cdot (-0.1274) = -0.0127$$



Solution of the XOR Problem...

■ Backpropagation...

$$\delta_{j=1} = z_1(1 - z_1) \cdot \delta_1 \cdot w_{11} = 0.5250 \cdot (1 - 0.5250) \cdot (-0.1274) \cdot (-1.2) = 0.0381$$

$$\delta_{j=2} = z_2(1 - z_2) \cdot \delta_1 \cdot w_{21} = 0.8808 \cdot (1 - 0.8808) \cdot (-0.1274) \cdot 1.1 = -0.0147$$

$$\Delta v_{11} = \alpha \cdot x_1 \cdot \delta_{j=1} = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$

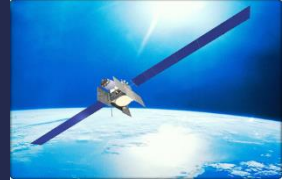
$$\Delta v_{21} = \alpha \cdot x_2 \cdot \delta_{j=1} = 0.1 \cdot 1 \cdot 0.0381 = 0.0038$$

$$\Delta v_{01} = \alpha \cdot (+1) \cdot \delta_{j=1} = 0.1 \cdot (+1) \cdot 0.0381 = +0.0038$$

$$\Delta v_{12} = \alpha \cdot x_1 \cdot \delta_{j=2} = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$

$$\Delta v_{22} = \alpha \cdot x_2 \cdot \delta_{j=2} = 0.1 \cdot 1 \cdot (-0.0147) = -0.0015$$

$$\Delta v_{02} = \alpha \cdot (+1) \cdot \delta_{j=2} = 0.1 \cdot (+1) \cdot (-0.0147) = -0.0015$$



Solution of the XOR Problem...

- **Weight Update**

$$v_{01} = -0.7962$$

$$v_{11} = 0.5038$$

$$v_{12} = 0.8985$$

$$v_{02} = +0.0985$$

$$v_{21} = 0.4038$$

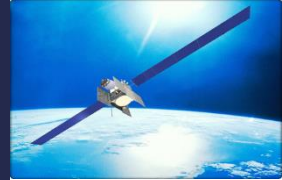
$$v_{22} = 0.9985$$

$$w_{01} = -0.3127$$

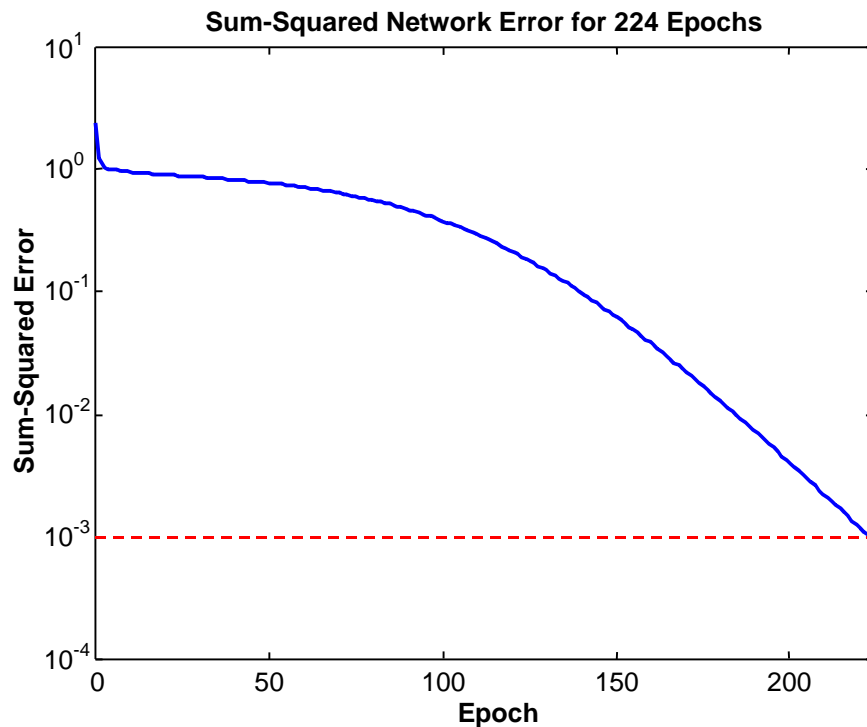
$$w_{11} = -1.2067$$

$$w_{21} = 1.0888$$

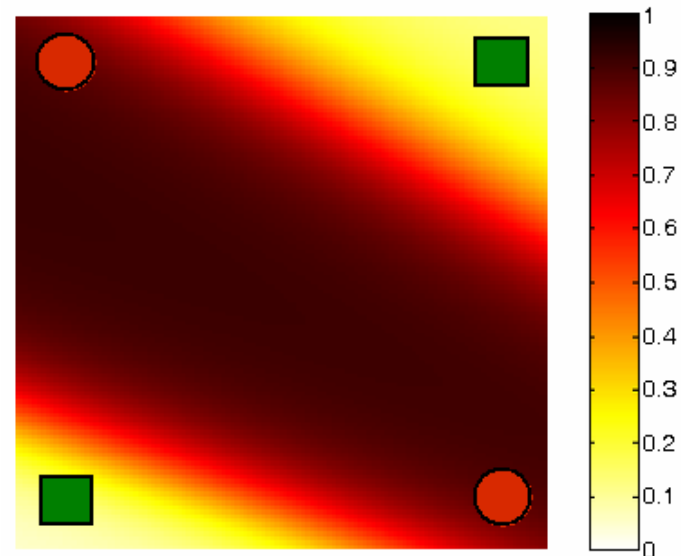
- **The training process is repeated until the sum of squared errors is less than 0.001.**



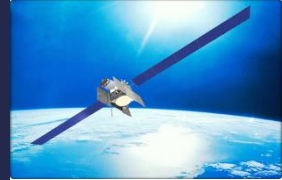
Solution of the XOR Problem...



Inputs		Desired output	Actual output	Error e	Sum of squared errors
x_1	x_2				
1	1	0	0.0155	-0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	-0.0175	

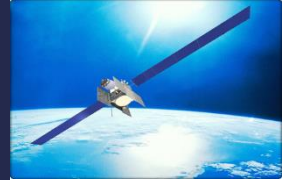


See Video!



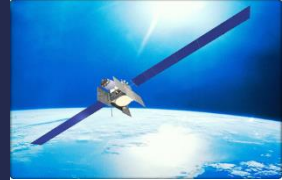
Number of hidden layers...

- **Theoretically, One hidden layer is sufficient for a Backpropagation net to approximate any continuous mapping from the input patterns to the output pattern to an arbitrary degree of accuracy**
- **However two hidden layers may make training easier in some situations**



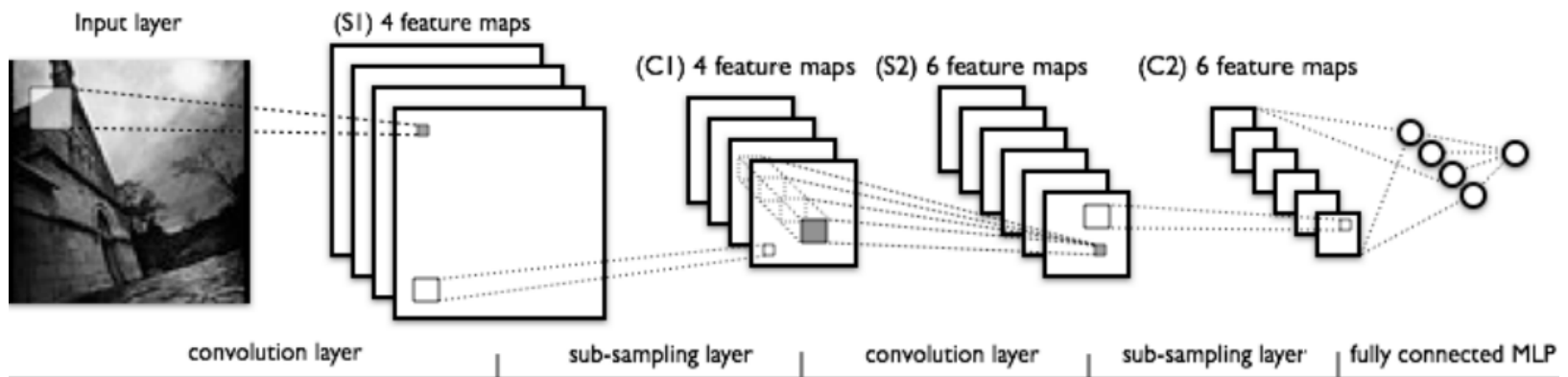
Selecting parameters

- **Architecture**
 - **Number of layers**
 - **Number of neurons in each layer**
- **Activation Function**
- **Learning rate**
- **Stopping criterion**



Applications

The LeNet-5 network:



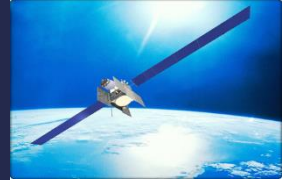
Important ideas:

Extract local features (local receptive fields) and merge them later to create global features

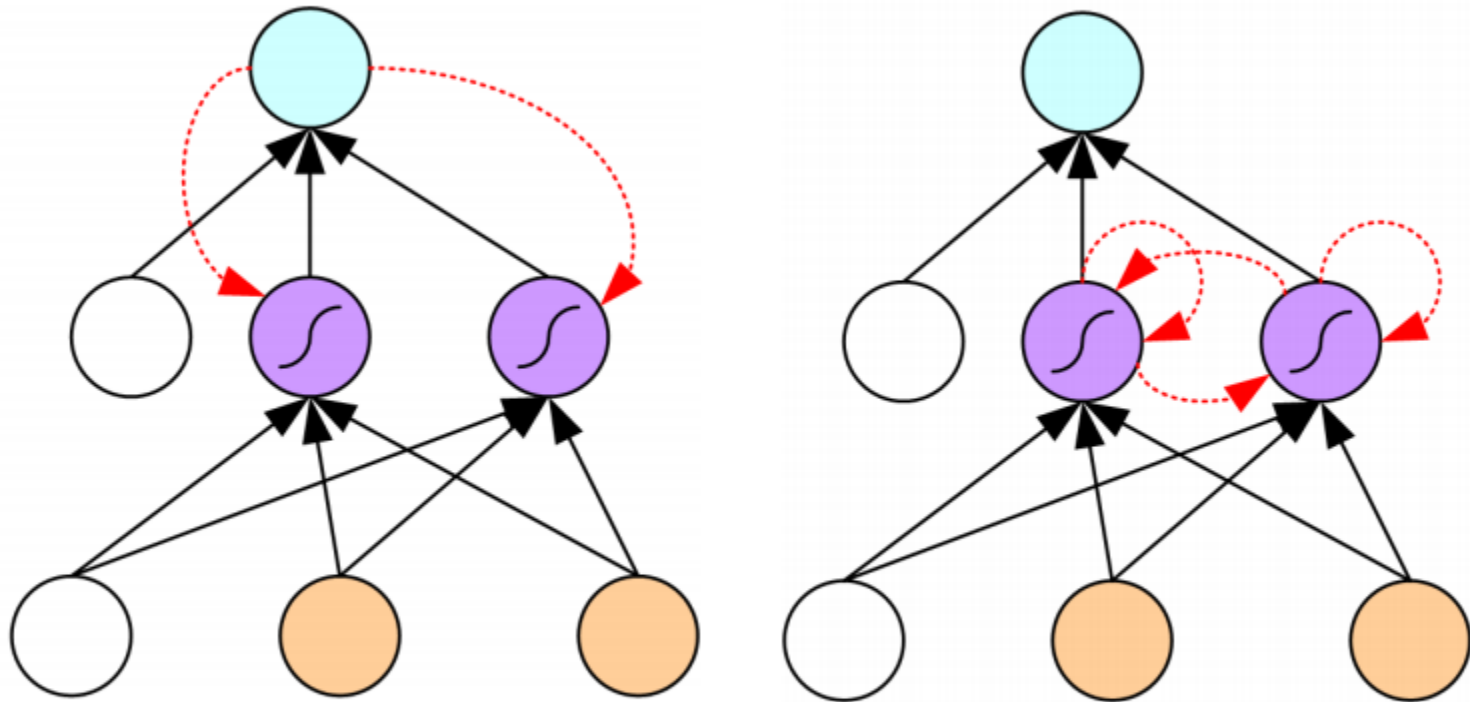
Local features that are useful in one region are likely to be useful elsewhere - weight sharing

<http://yann.lecun.com/exdb/lenet/>

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, november 1998.

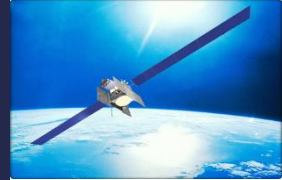


Recurrent Networks

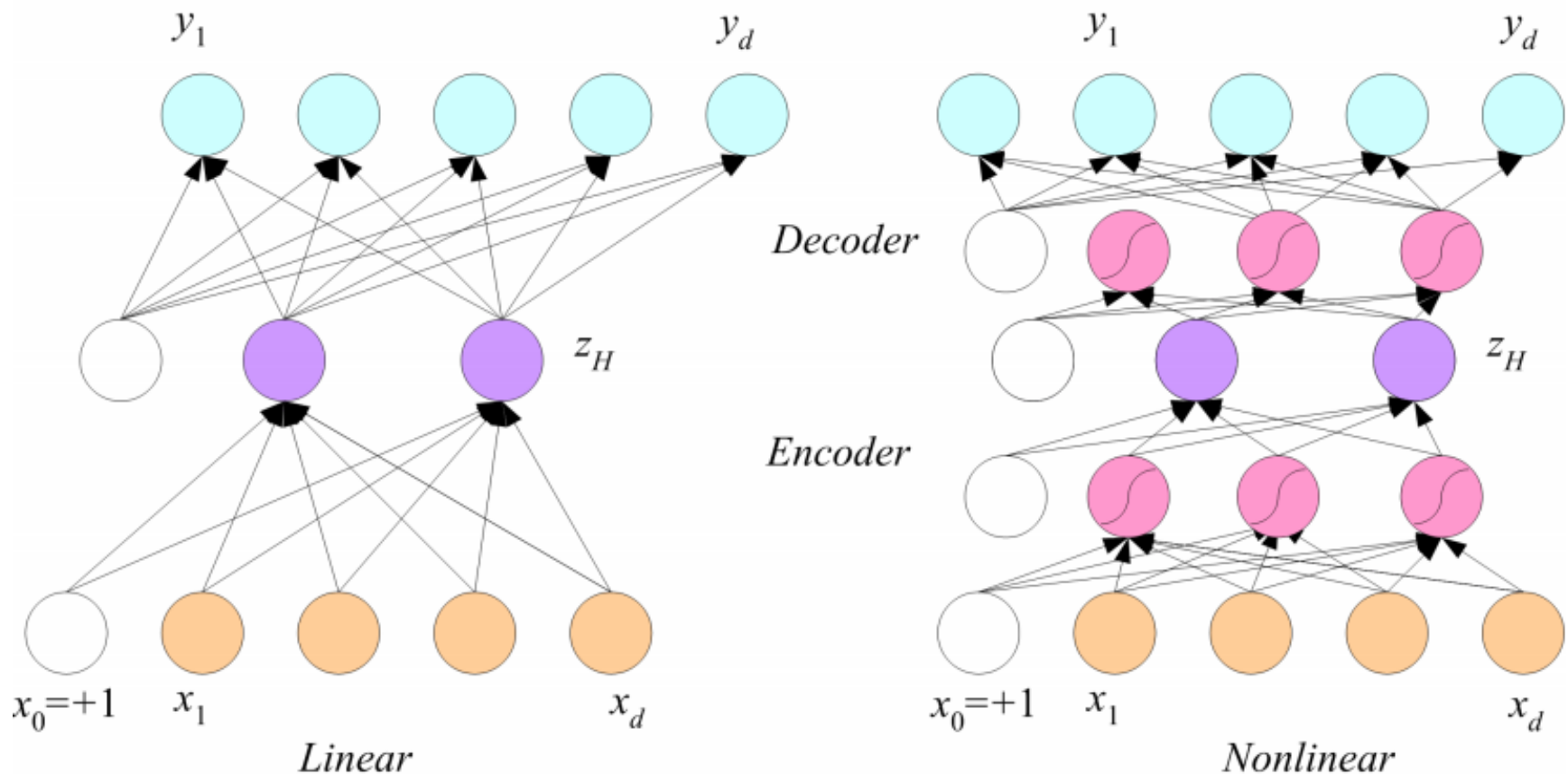


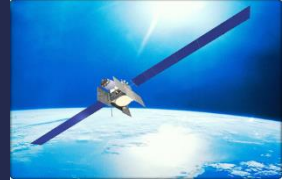
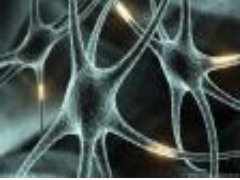
The cycles allow the network to exhibit dynamic temporal behavior

RNNs can use their internal memory to process arbitrary sequences of inputs.



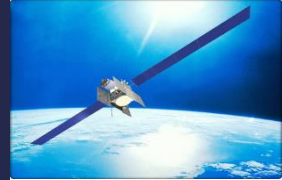
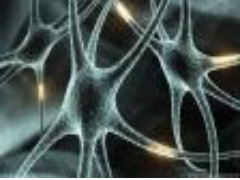
Dimensionality Reduction





Deep learning

- **NNs can be used for feature learning**



Reading

- **Fundamentals of Neural Networks (Laurene Faucett)**
 - **Perceptron: Chapter 2**
 - **MLP: Chapter 6**
- **Neural Networks a comprehensive foundation (1999)**