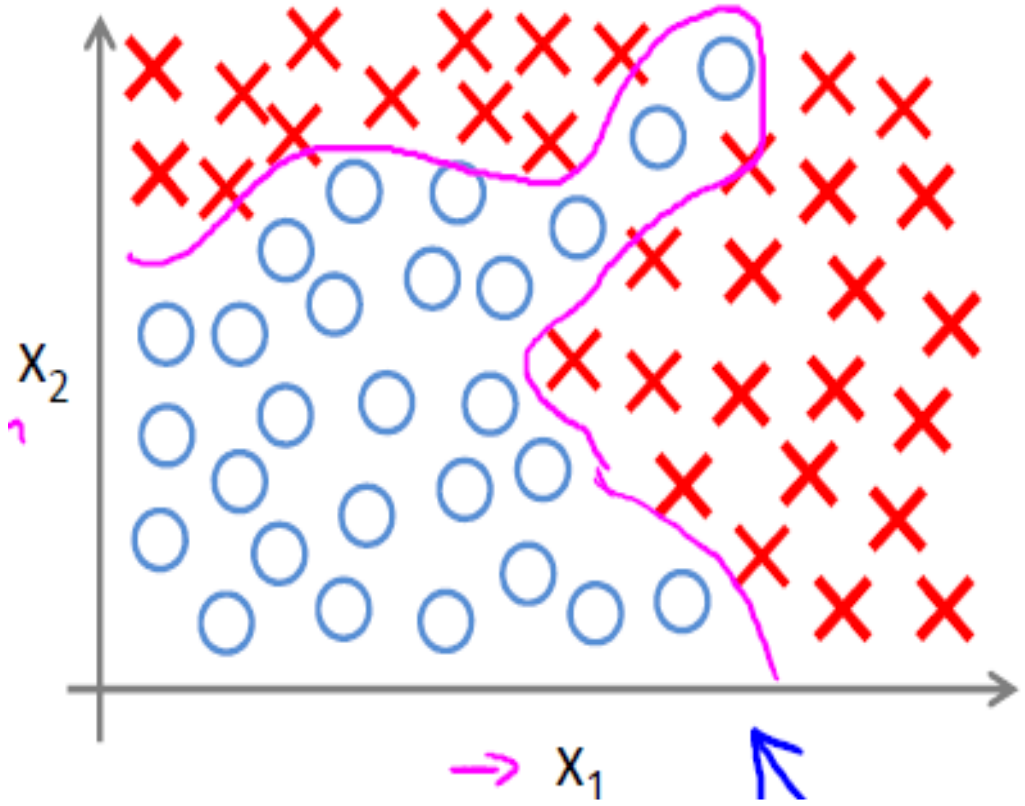
Machine Learning
DSE 2254

Rohini R. Rao & Manjunath Hegde
Department of Data Science and Computer Applications
MIT, Manipal
March 2022
Slide - Set 3 – Neural Networks

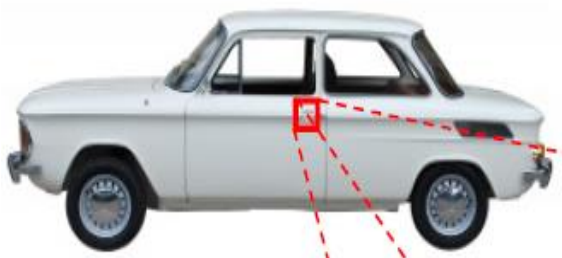
Non Linear Regression



$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

Adding Higher Order Polynomials increases time complexity

Application – Computer Vision

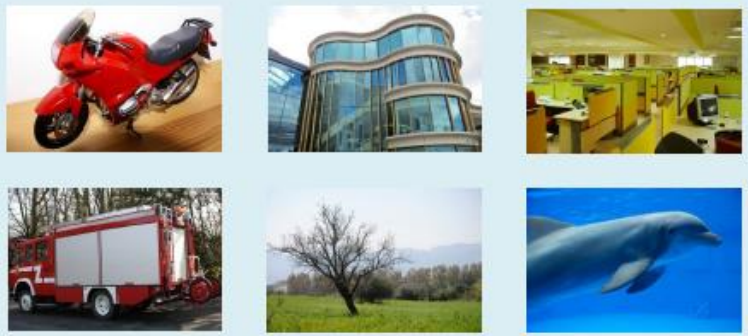


But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Application – Car Detection

Cars



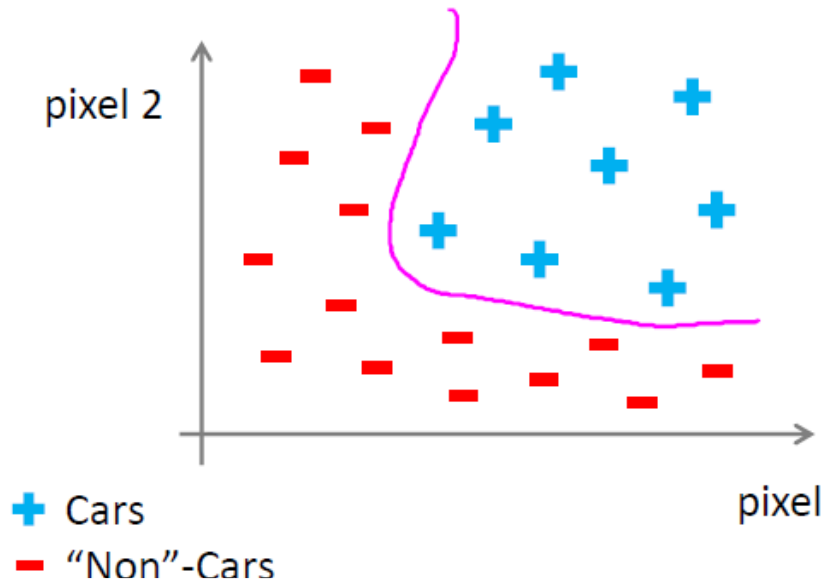
Not a car

Testing:

What is this?

Application – Car Detection

Learning Algorithm



50 x 50 pixel images→ 2500 pixels

$n = 2500$ (7500 if RGB)

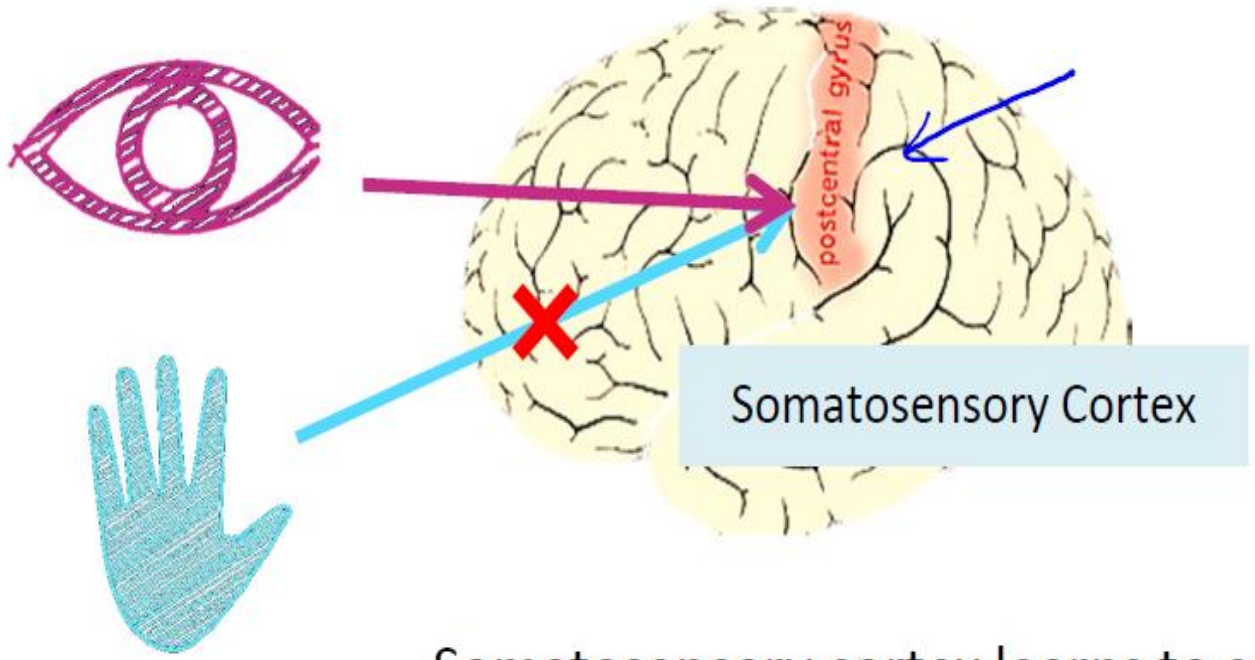
$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

0-255

Quadratic features ($x_i \times x_j$): ≈3 million features

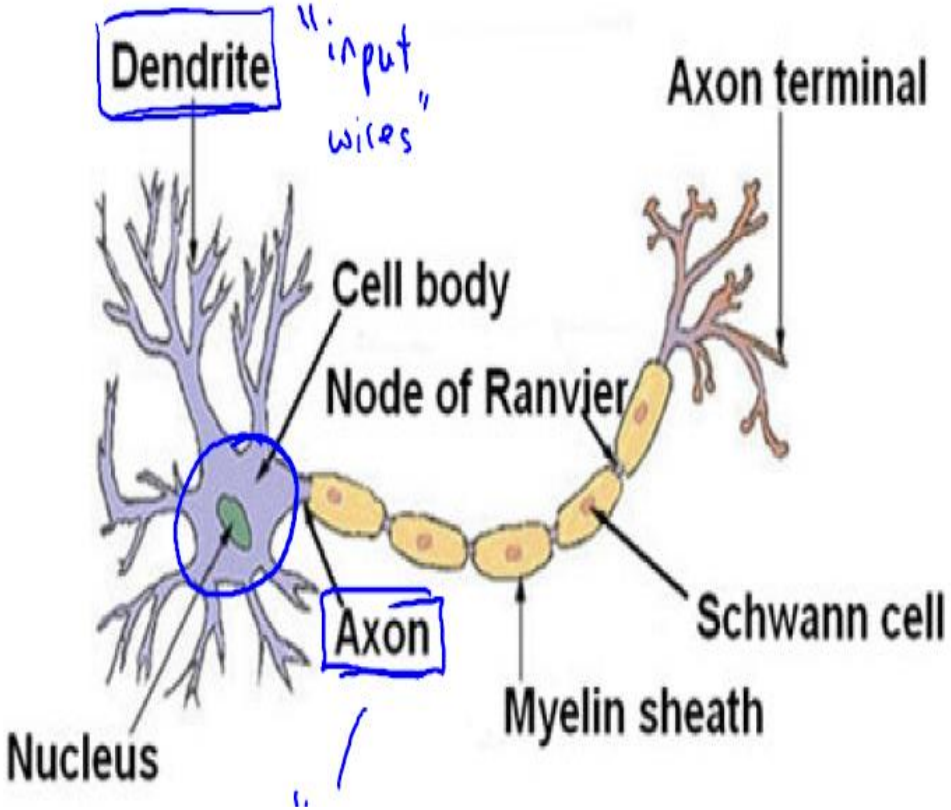
Neural Network

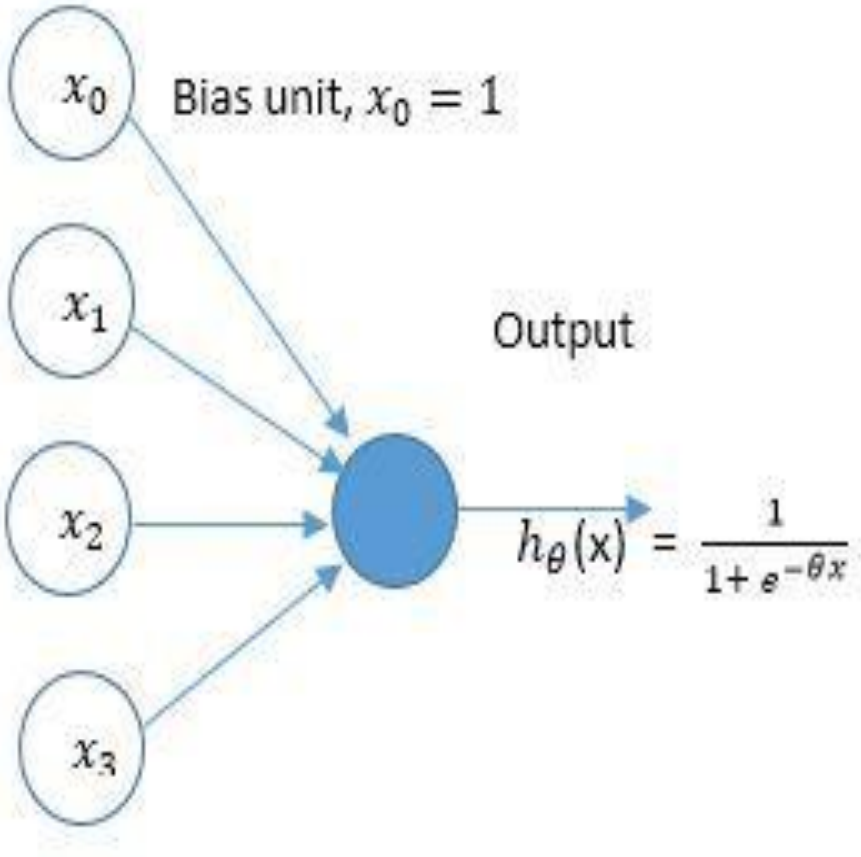
The "one learning algorithm" hypothesis



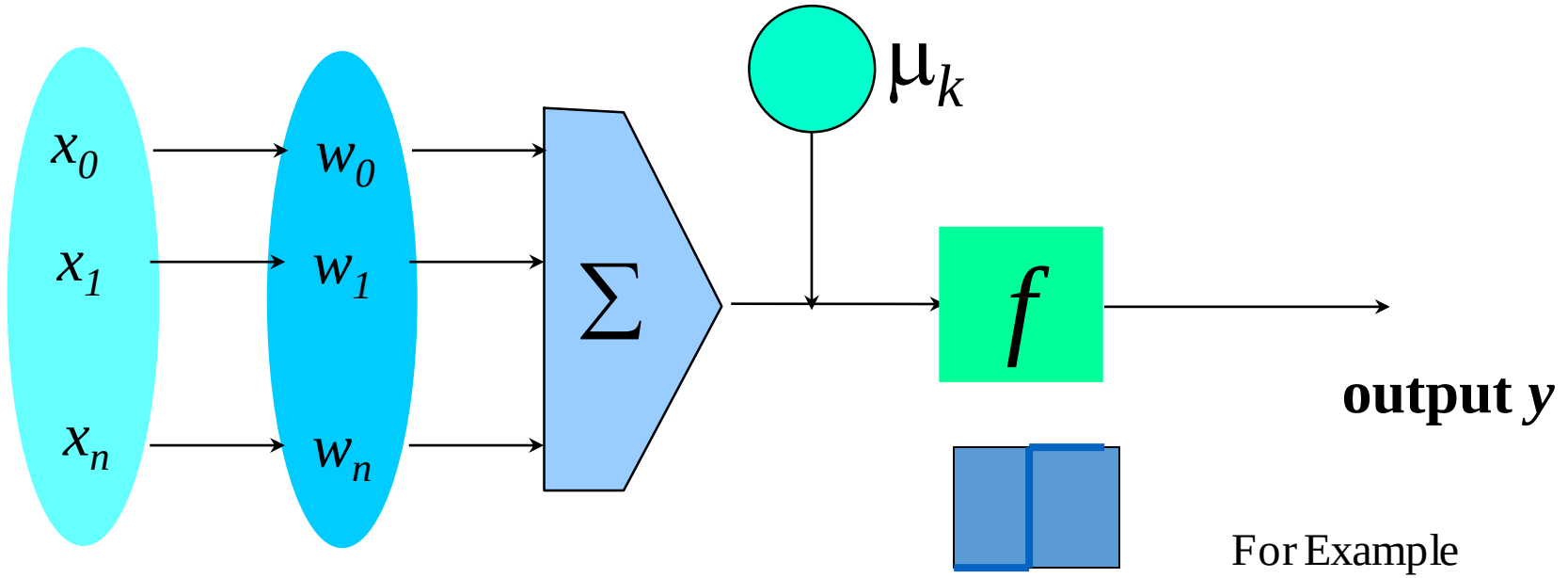
Somatosensory cortex learns to see

Neuron is a computational , logistic unit





A Neuron (= a perceptron)



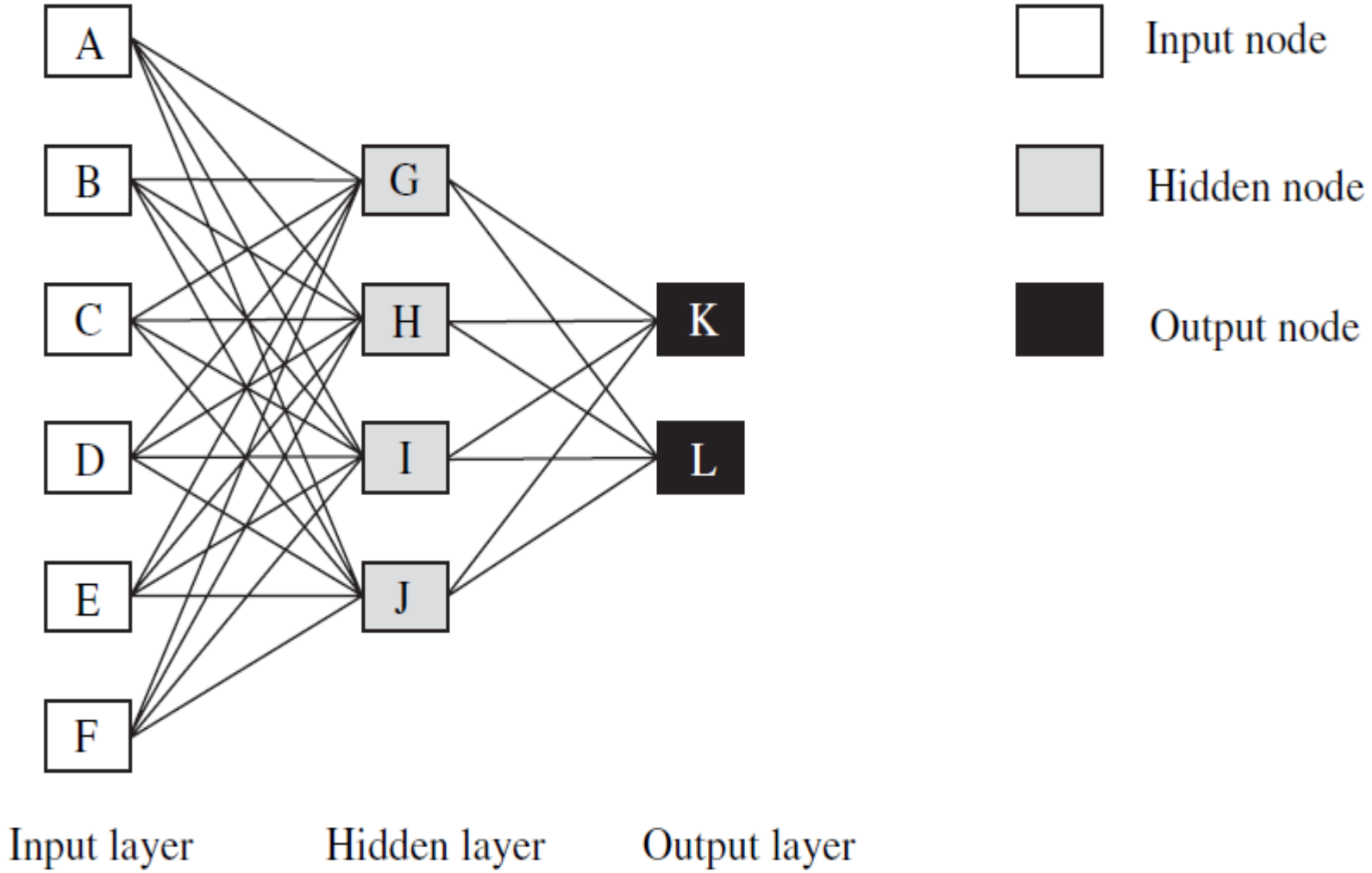
Input vector x | **weight vector w** | **weighted sum** | **Activation function**

For Example

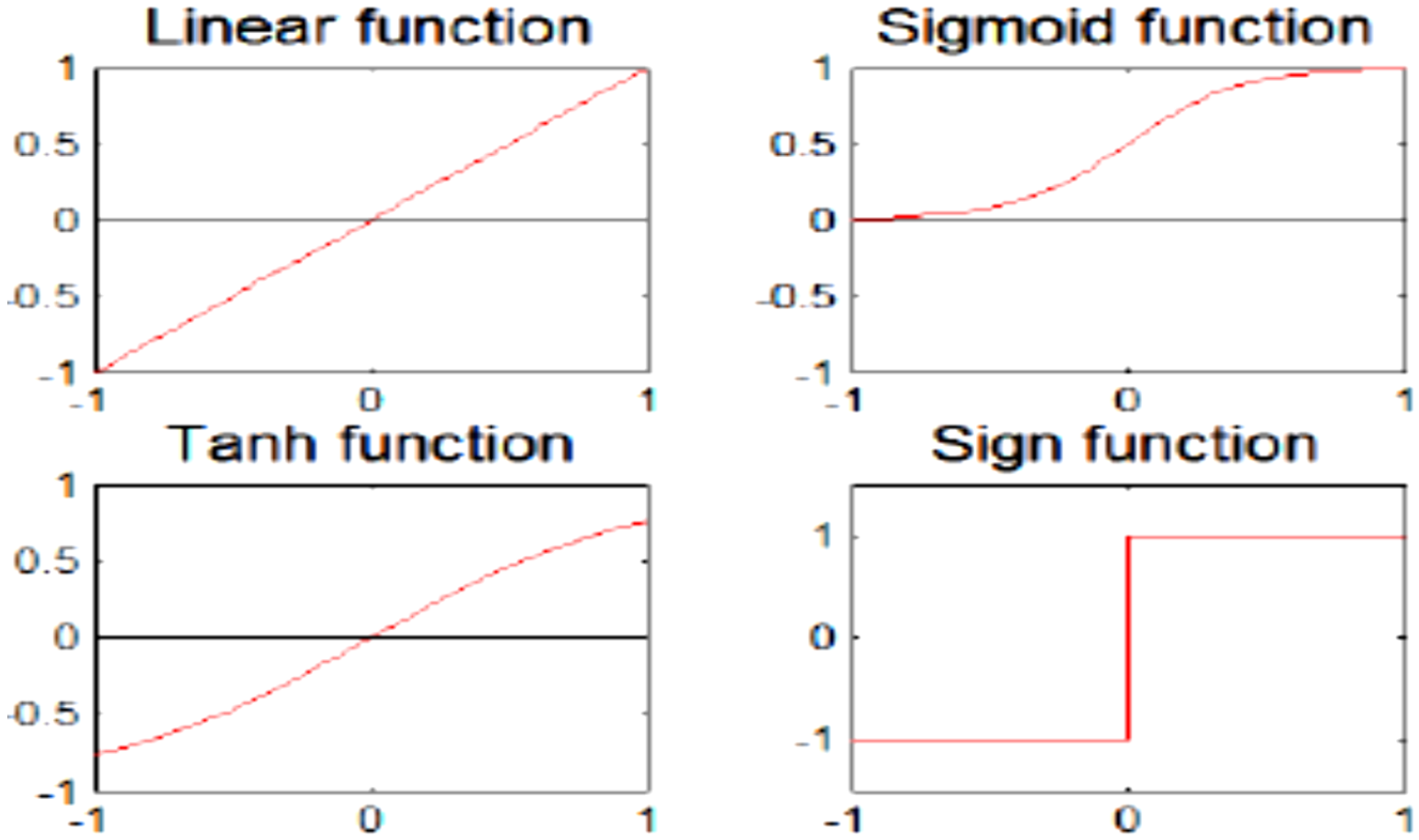
$$y = \text{sign}(\sum_{i=0}^{n} w_i x_i + \mu_k)$$

- The *n*-dimensional input vector **x** is mapped into variable y by means of the scalar product and a nonlinear function mapping

Multi Layer Neural Network



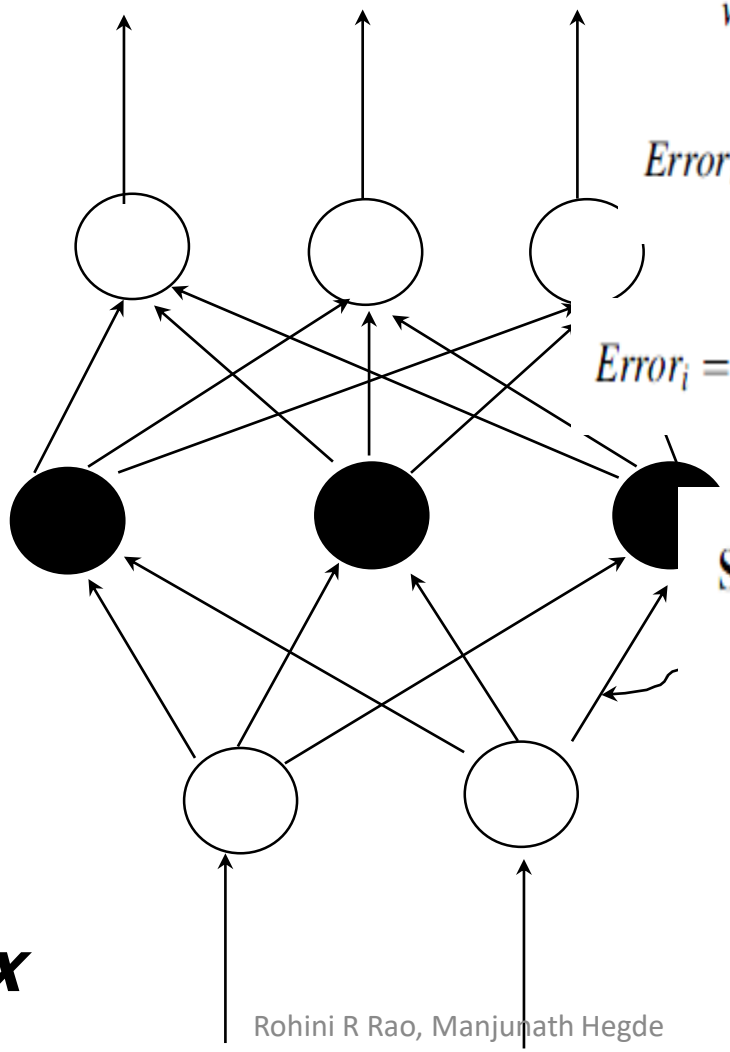
Types of Activation Function



Back propagation Algorithm

- comprises of a series of independent processors or nodes.
- These nodes are connected to other nodes and are organized into a series of layers
- Each connection has a weight associated with it.
- performs learning on a *multilayer feed-forward* neural network.
- **Feed-forward** network since none of the weights cycles back to an input unit or to a previous layer's output unit
- These modifications are made in the "backwards" direction (i.e., from the output layer) through each hidden layer down to the first hidden layer
- During the learning phase, the network learns by adjusting the weights so as to be able to predict the correct class label of the input tuples.

A Multi-Layer Feed-Forward Neural Network Back Propagation



$$w_{ij} = w_{ij} + l \times Error_j \times Output_i$$

$$Error_i = Output_i(1 - Output_i)\sum_{j=1}^{n} Error_j w_{ij}$$

$$Error_i = Output_i(1 - Output_i)(Actual_i - Output_i)$$

$$\text{Sigmoid}: \quad Output = \frac{1}{1 + e^{-Input}}$$

$$\text{Tanh}: \quad Output = \frac{e^{Input} - e^{-Input}}{e^{Input} + e^{-Input}}$$

$$Input = \sum_{j=1}^{n} I_j w_j$$

Learning technique in Neural network

- Back Propagation

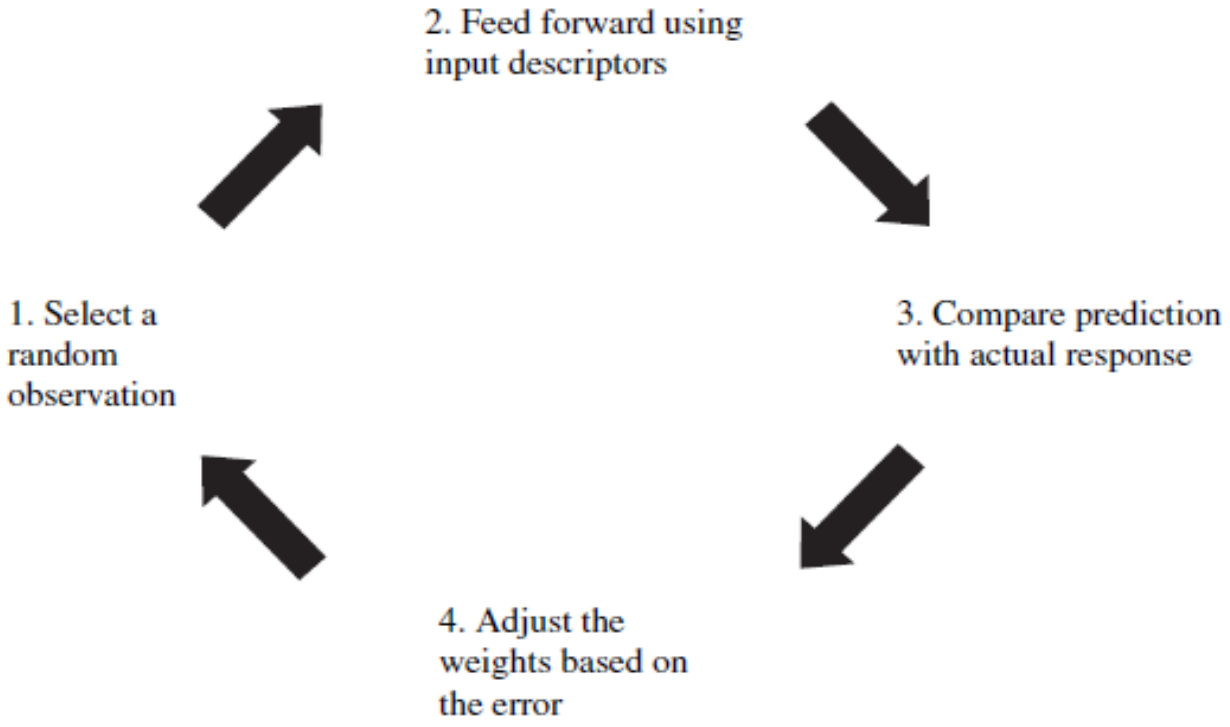
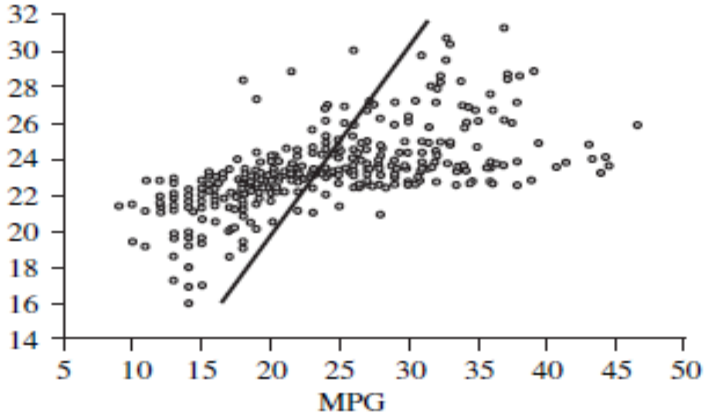


Figure 7.31. Learning process in neural networks

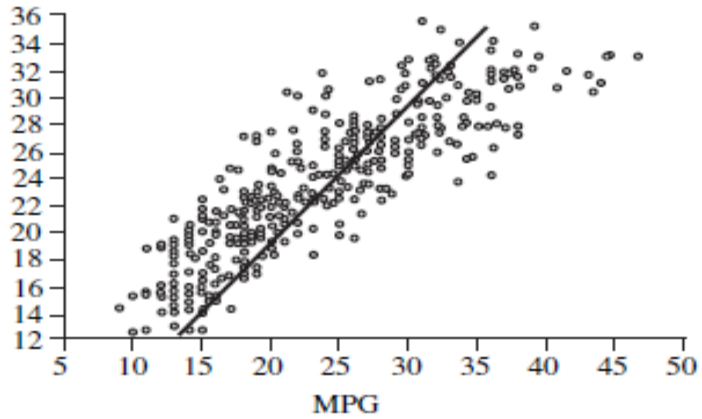
Prediction results for different cycles

5,000 cycles



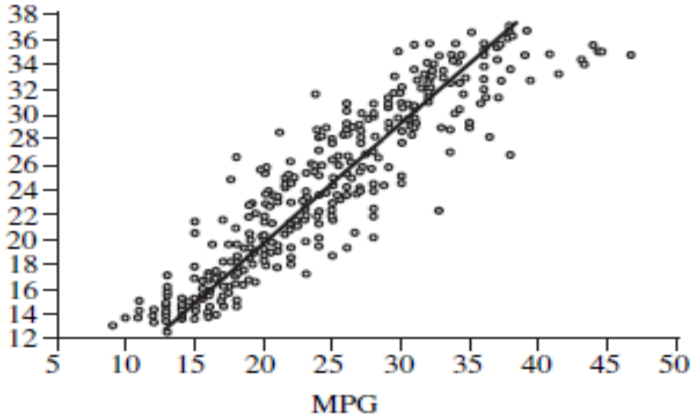
$r^2 = 0.39$

10,000 cycles



$r^2 = 0.74$

20,000 cycles

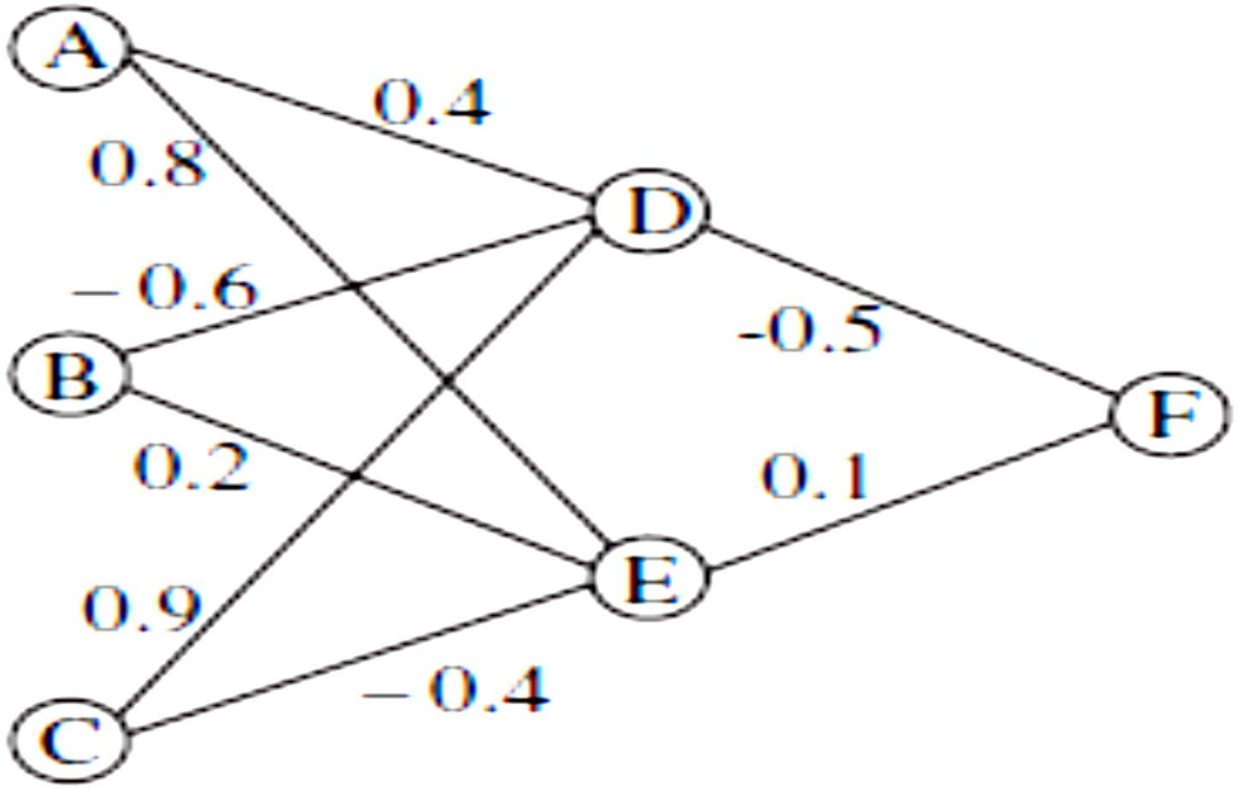


$r^2 = 0.86$

Example- Back Propagation

Step 1

	I_1	I_2	I_3	O
i	1	1	1	1
ii	0	0	0	0
iii	1	0	0	1
iv	0	0	1	0
v	0	1	1	0



Example- Back Propagation

$$Input_D = I_1 \times w_{AD} + I_2 \times w_{BD} + I_3 \times w_{CD}$$

$$Input_D = (0 \times 0.4) + (1 \times -0.6) + (1 \times 0.9) = 0.3$$

$$Input_E = I_1 \times w_{AE} + I_2 \times w_{BE} + I_3 \times w_{CE}$$

$$Input_E = (0 \times 0.8) + (1 \times 0.2) + (1 \times -0.4) = -0.2$$

$$Output_D = \frac{1}{1 + e^{-Input_D}} = \frac{1}{1 + e^{-0.3}} = 0.57$$

$$Output_E = \frac{1}{1 + e^{-Input_E}} = \frac{1}{1 + e^{0.2}} = 0.45$$

$$Input_F = Output_D \times w_{DF} + Output_E \times w_{EF}$$

$$Input_F = (0.57 \times -0.5) + (0.45 \times 0.1) = -0.24$$

$$Output_F = \frac{1}{1 + e^{-Input_F}} = \frac{1}{1 + e^{0.24}} = 0.44$$

Example – Back Propagation

$$Error_F = Output_F(1 - Output_F)(Actual_F - Output_F)$$

$$Error_F = 0.44(1 - 0.44)(0 - 0.44) = -0.108$$

$$Error_D = Output_D(1 - Output_D)(Error_F \times w_{DF})$$

$$Error_D = 0.57(1 - 0.57)(-0.108 \times -0.5) = 0.013$$

$$Error_E = Output_E(1 - Output_E)(Error_F \times w_{EF})$$

$$Error_E = 0.45(1 - 0.45)(-0.108 \times 0.1) = -0.003$$

Example – Back Propagation

$$w_{ij} = w_{ij} + l \times Error_j \times Output_i$$

$$w_{AD} = 0.4 + 0.5 \times 0.013 \times 0 = 0.4$$

$$w_{AE} = 0.8 + 0.5 \times -0.003 \times 0 = 0.8$$

$$w_{BD} = -0.6 + 0.5 \times 0.013 \times 1 = -0.594$$

$$w_{BE} = 0.2 + 0.5 \times -0.003 \times 1 = 0.199$$

$$w_{CD} = 0.9 + 0.5 \times 0.013 \times 1 = 0.907$$

$$w_{CE} = -0.4 + 0.5 \times -0.003 \times 1 = -0.402$$

$$w_{DF} = -0.5 + 0.5 \times -0.108 \times 0.57 = -0.531$$

$$w_{EF} = 0.1 + 0.5 \times -0.108 \times 0.45 = 0.076$$

Neural networks

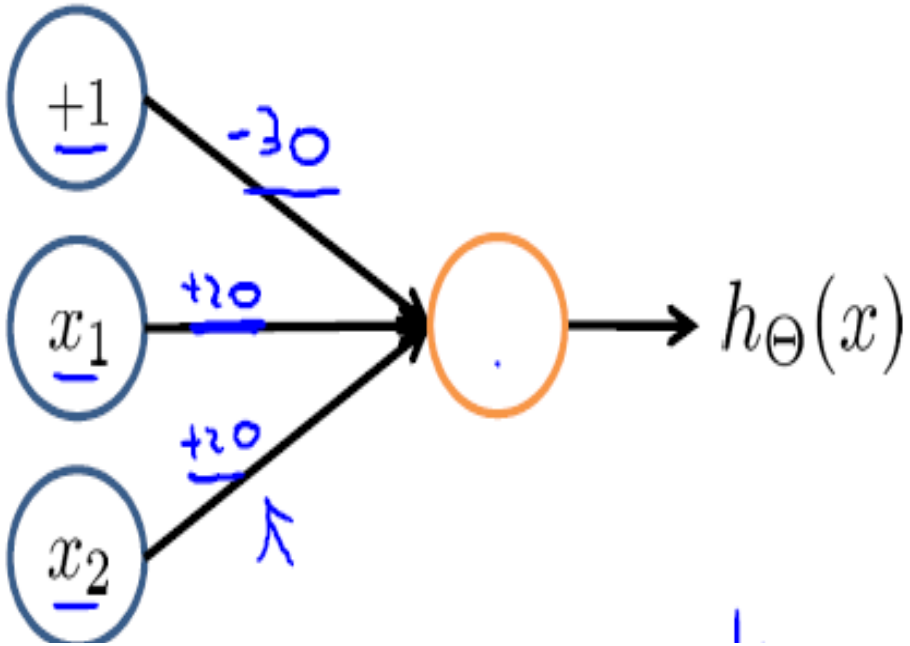
- **Domain expertise required for**
  - Hidden layers:
    - Both the number of hidden layers and the number of nodes in each hidden layer can influence the quality of the results.
    - Too few layers and/or nodes may not be adequate to sufficiently learn and too many may result in overtraining the network.
  - Number of cycles:
    - A cycle is where a training example is presented and the weights are adjusted.
    - The number of cycles or epochs should be set to ensure that the neural network does not overtrain.
  - Learning rate:
    - Prior to building a neural network, the learning rate should be set and this influences how fast the neural network learns

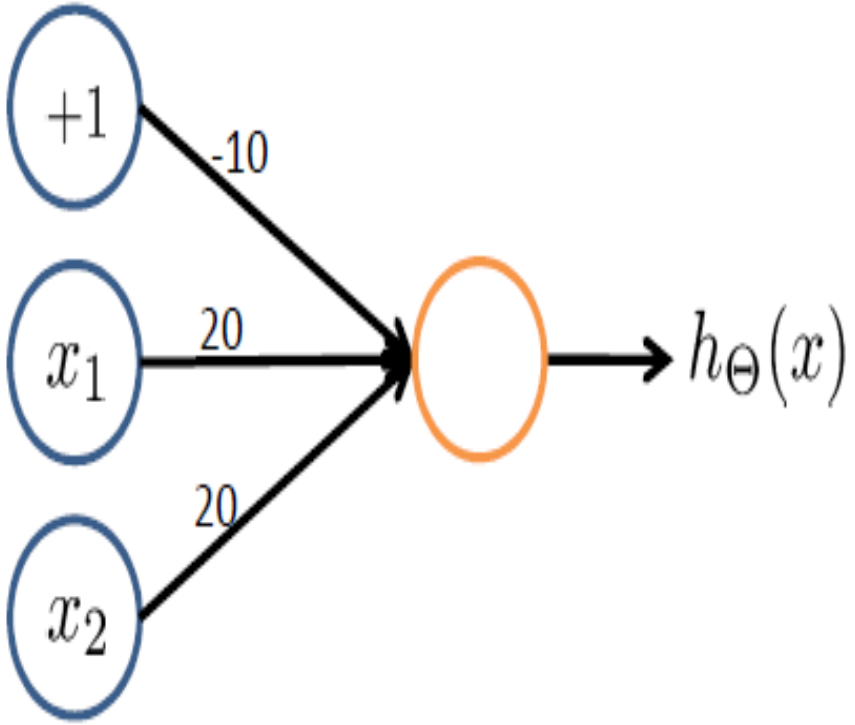
Neural networks advantages & disadvantages

- **Advantages:**
  - Linear and nonlinear models: Complex linear and nonlinear relationships can be derived using neural networks.
  - Flexible input/output: Neural networks can operate using one or more descriptors and/or response variables. They can also be used with categorical and continuous data.
  - Noise: Neural networks are less sensitive to noise than statistical regression models.
  - can be used when you may have little knowledge of the relationships between attributes and classes
  - Neural network algorithms are inherently parallel; parallelization techniques can be used to speed up the computation process.
- **The major drawbacks with neural networks are:**
  - Black box: It is not possible to explain how the results were calculated in any meaningful way.
  - Optimizing parameters: There are many parameters to be set in a neural network and optimizing the network can be challenging, especially to avoid overtraining.
- Successful on a wide array of real-world data, including handwritten character recognition, pathology and laboratory medicine, and training a computer to pronounce English text.

Simple AND and Simple OR

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1 \text{ AND } x_2$

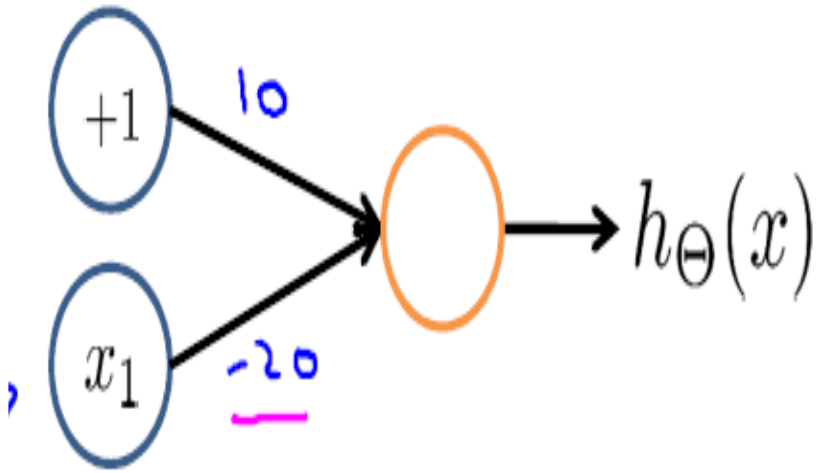




Negation

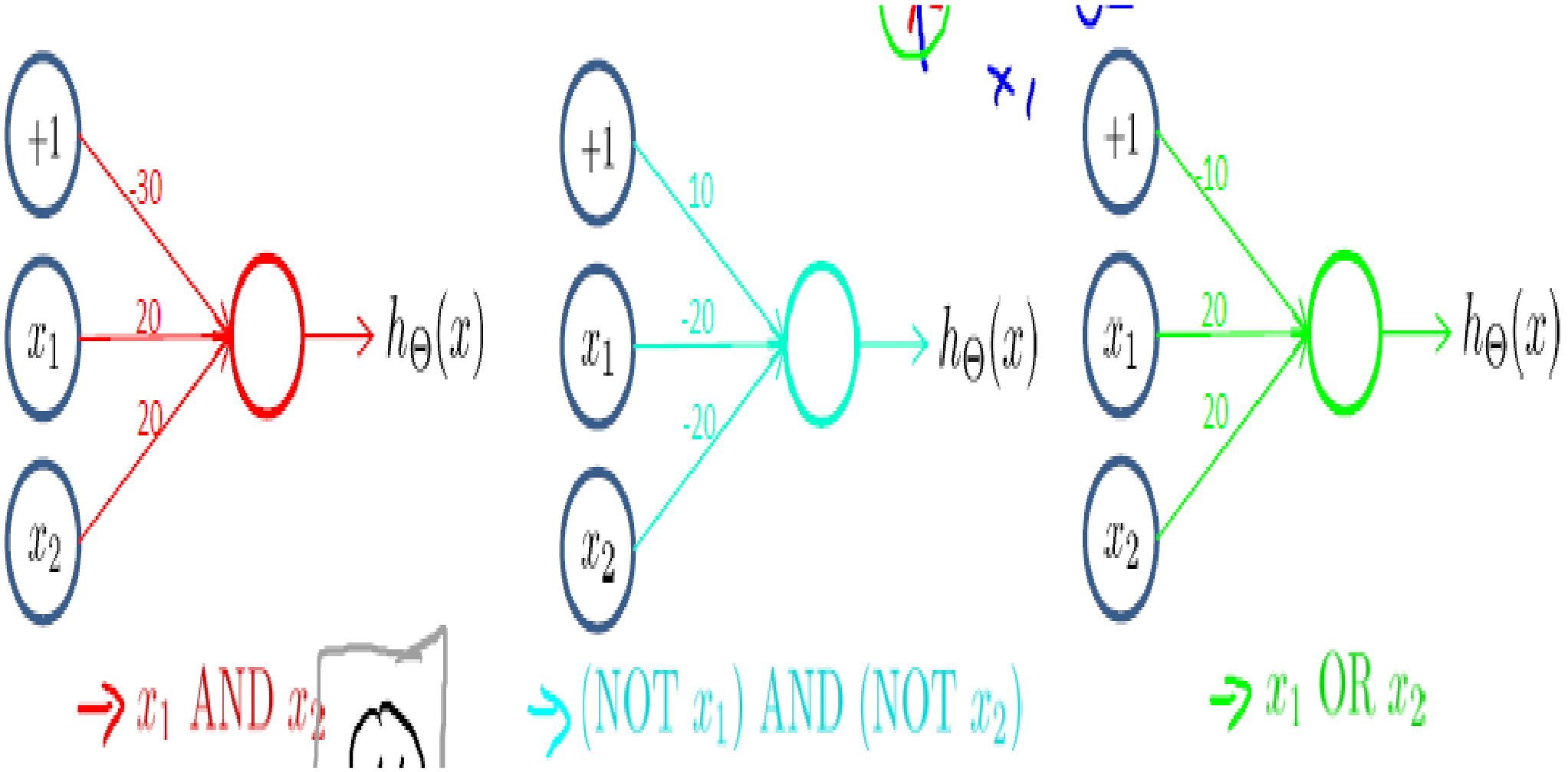
Negation:

NOT x_1

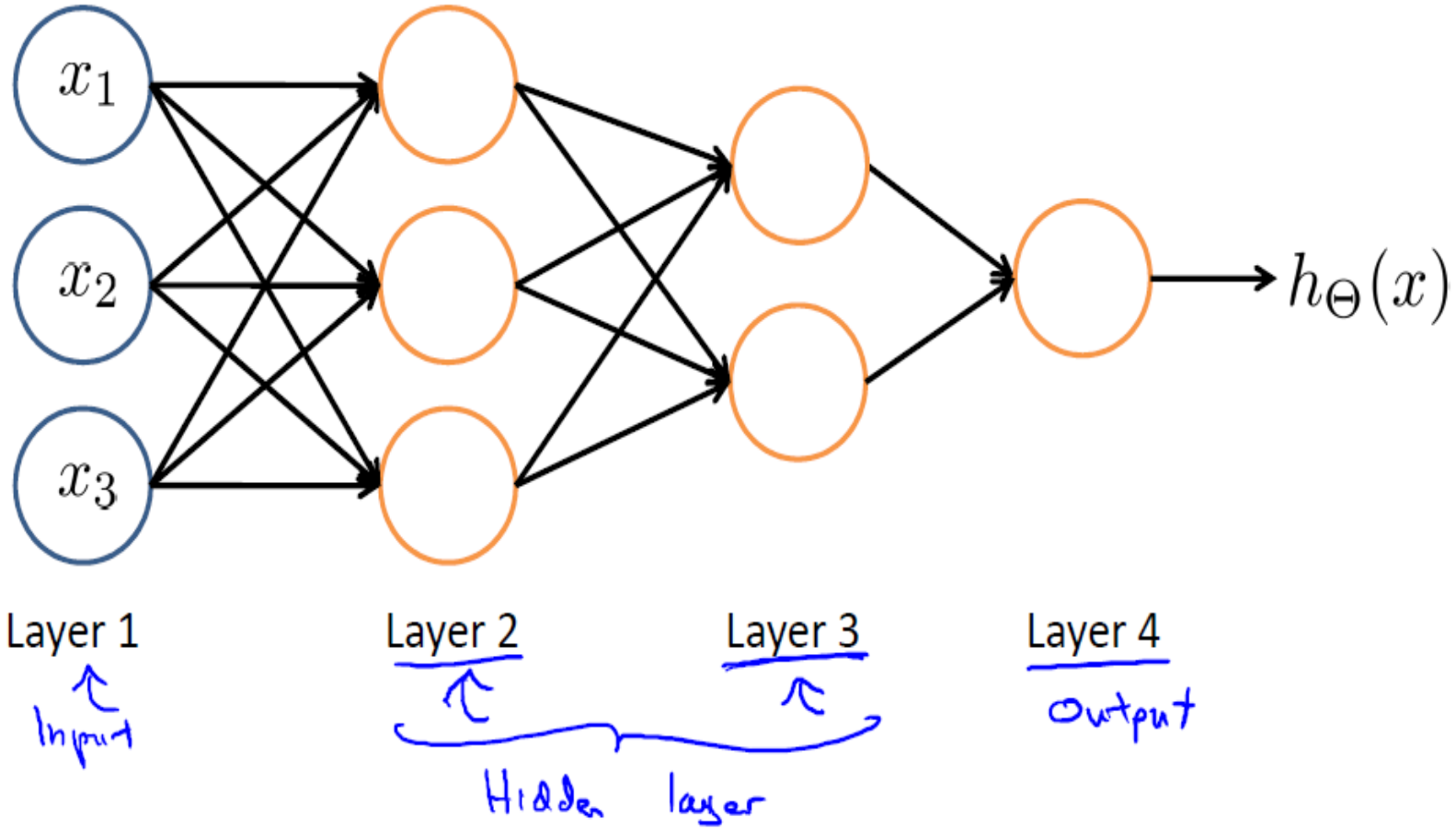


x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

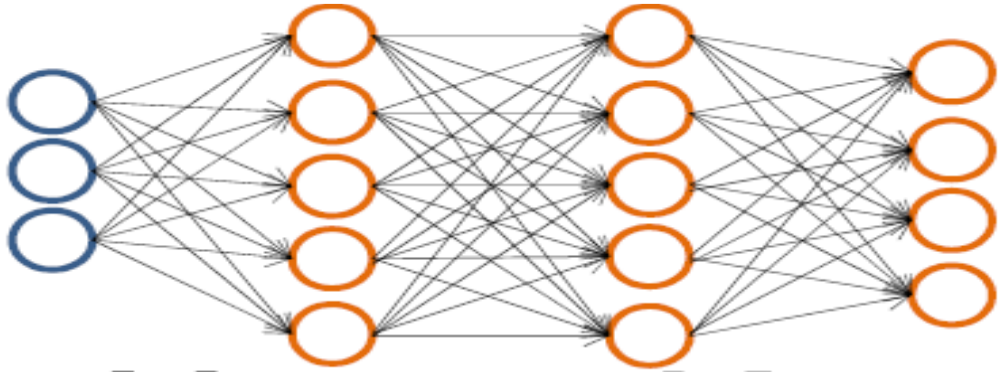
X1 XNOR x2


+1
-30
x_1
20
20
x_2
$h_\Theta(x)$
x_1 AND x_2
+1
10
x_1
-20
-20
x_2
$h_\Theta(x)$
(NOT x_1) AND (NOT x_2)
+1
-10
x_1
20
20
x_2
$h_\Theta(x)$
x_1 OR x_2

Neural Network learns its own features



Multiple output units – One vs. All



$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

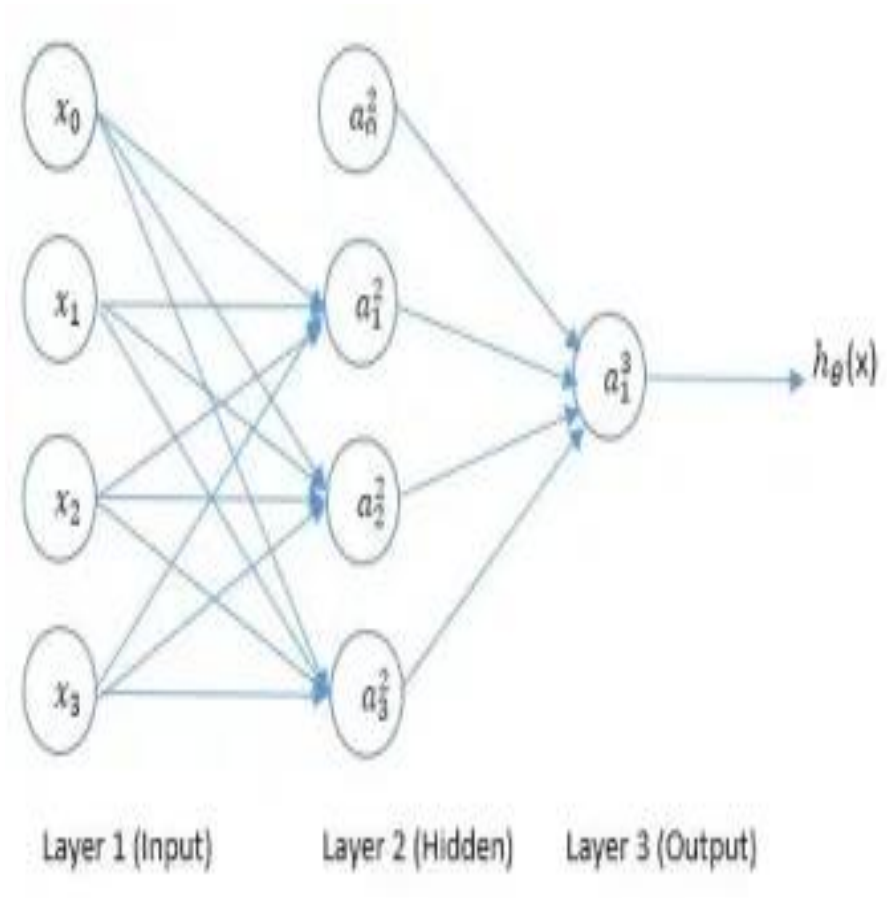
- $y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

$(x^{(i)}, y^{(i)})$

=

Neural Network



a_i^j *activation of unit* i *in layer* j

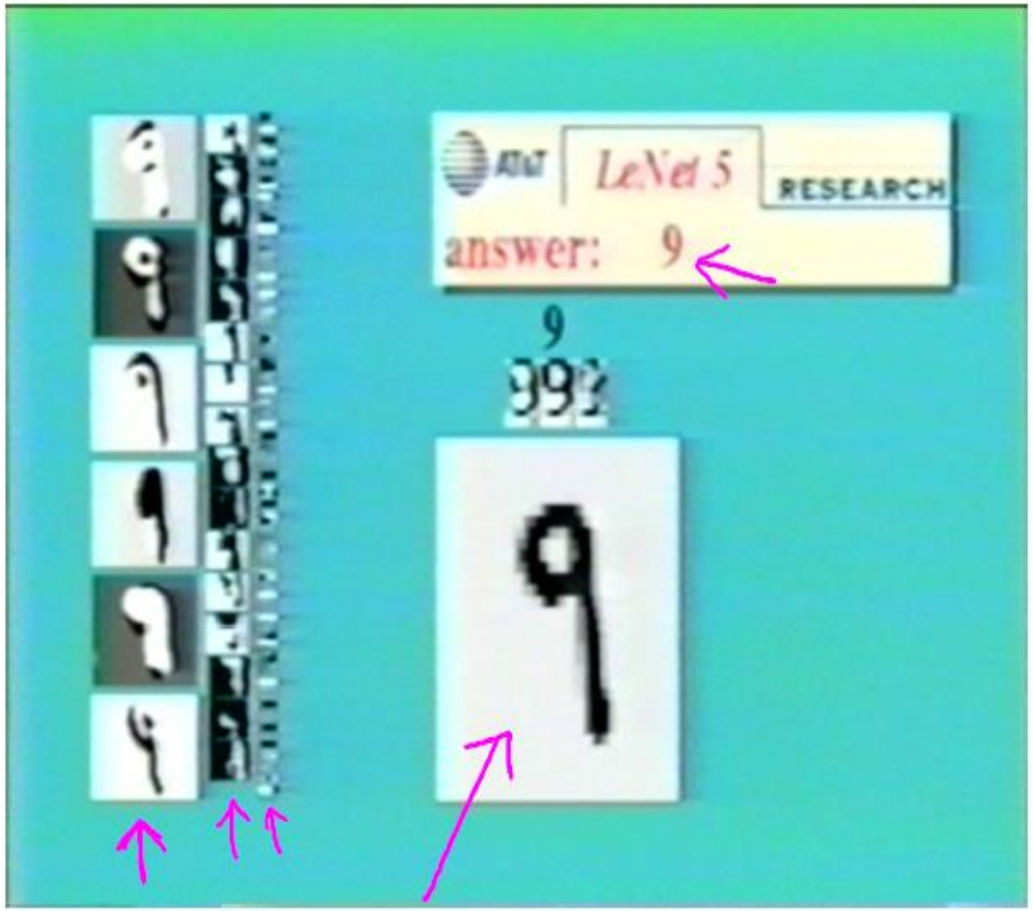
$\theta^j = matrix\ of\ weights\ controlling\ function\ mapping\ from\ layer\ j\ to\ layer\ j+1$

$$a_1^2 = g(\theta_{10}^1 x_0 + \theta_{11}^1 x_1 + \theta_{12}^1 x_2 + \theta_{13}^1 x_3)$$

$$a_2^2 = g(\theta_{20}^1 x_0 + \theta_{21}^1 x_1 + \theta_{22}^1 x_2 + \theta_{23}^1 x_3)$$

$$a_3^2 = g(\theta_{30}^1 x_0 + \theta_{31}^1 x_1 + \theta_{32}^1 x_2 + \theta_{33}^1 x_3)$$

Hand written Digit Classification



Cost Function

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

Neural network:

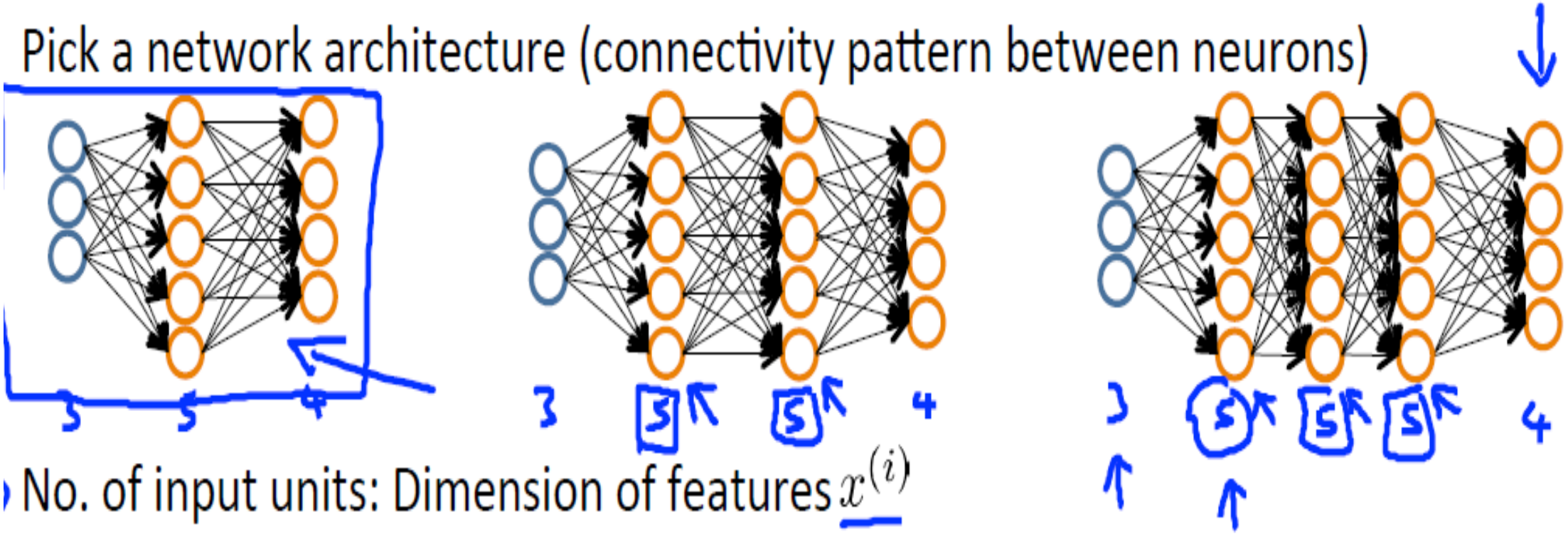
$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log((h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



- No. of input units: Dimension of features $x^{(i)}$
- No. output units: Number of classes
- Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

Training a neural network

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

```
for i = 1:m
```

→ Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$

(Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).

5. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

Training a neural network

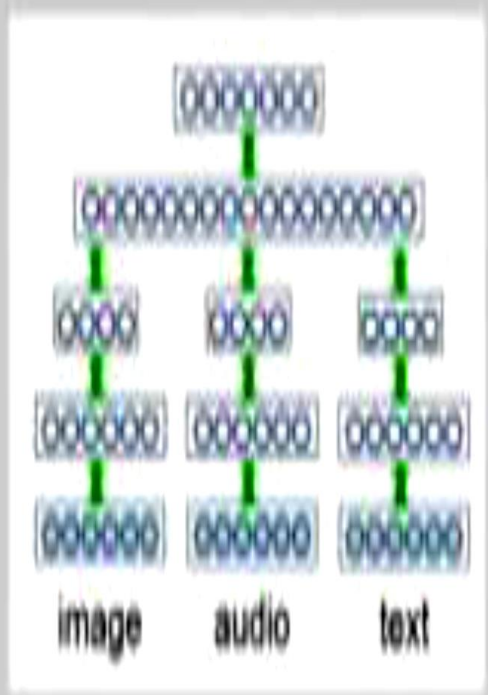
5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.

   Then disable gradient checking code.

   Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ

Deep Learning

- Breakthrough results in
  - Image classification
  - Speech Recognition
  - Machine Translation
  - Multi-modal learning



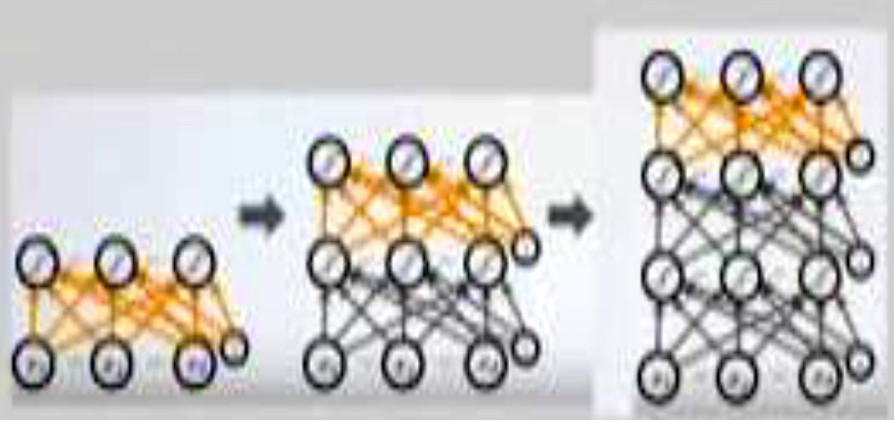
- Problem: training networks with many hidden layers doesn't work very well
- Local minima, very slow training if initialize with zero weights.
- Diffusion of gradient.

Hierarchical Representation

- Hierarchical Representation help represent complex functions.
- NLP: character ->word -> Chunk -> Clause -> Sentence
- Image: pixel > edge -> texton -> motif -> part -> object
- Deep Learning: learning a hierarchy of internal representations
- Learned internal representation at the hidden layers (trainable feature extractor)
- Feature learning

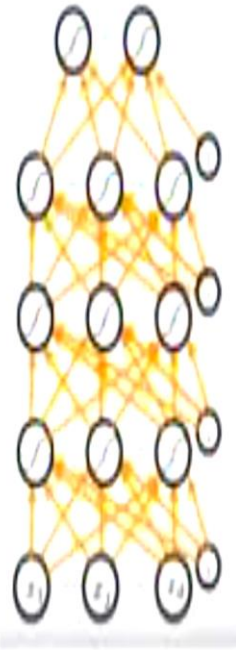
Unsupervised Pre-training

- We will use greedy, layer wise pre-training
  - Train one layer at a time
  - Fix the parameters of previous hidden layers
  - Previous layers viewed as feature extraction
- find hidden unit features that are more common in training input than in random inputs



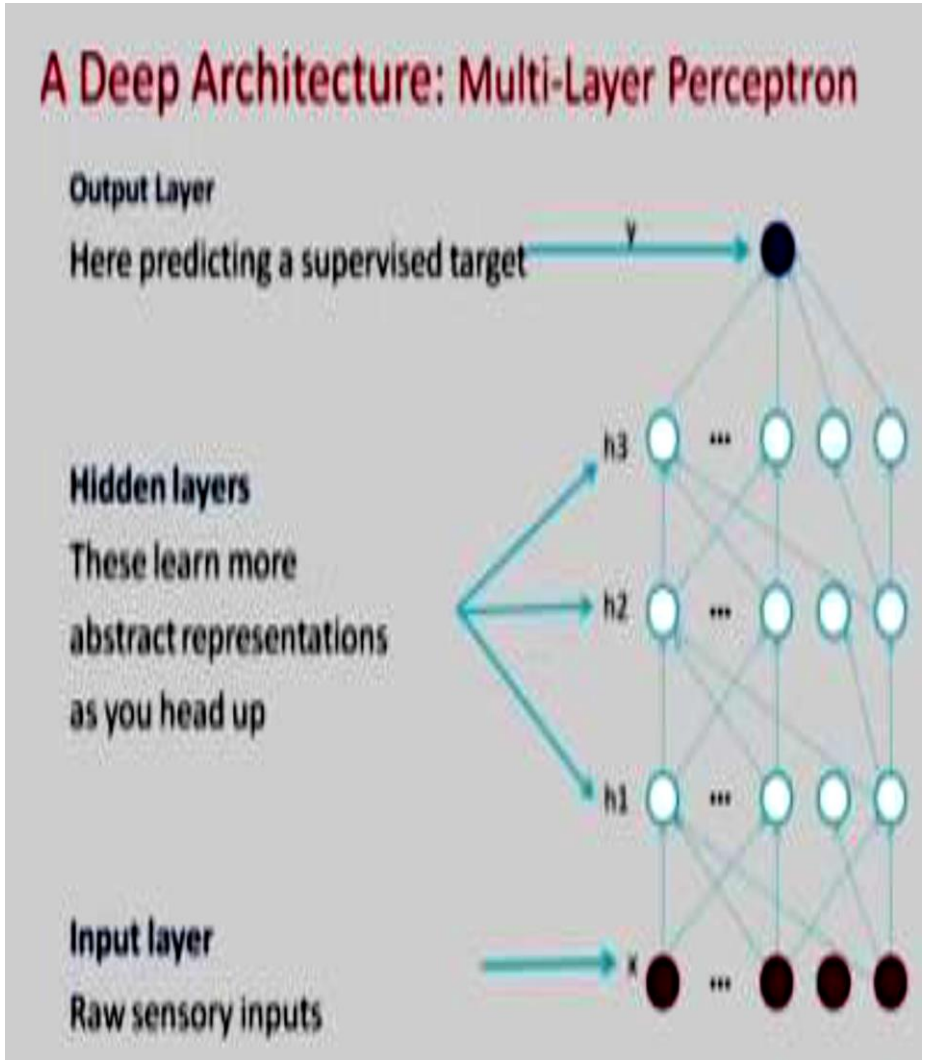
Tuning the Classifier

- After pre-training of the layers
  - Add output layer
  - Train the whole network using supervised learning (Back propagation)



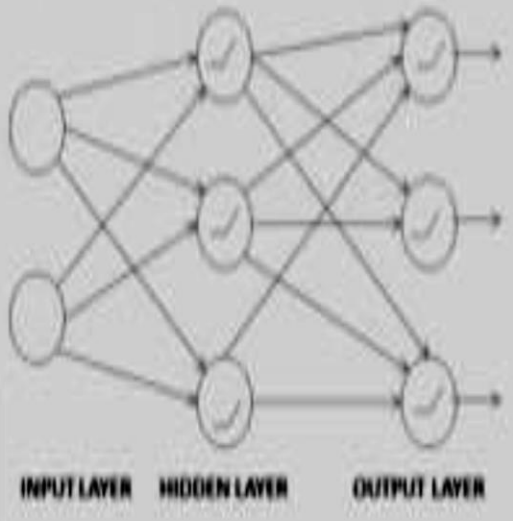
Deep Neural Network

- Feed forward NN
- Stacked Autoencoders (multilayer neural net with target output = input)
- Stacked restricted Boltzmann machine
- Convolutional Neural Network

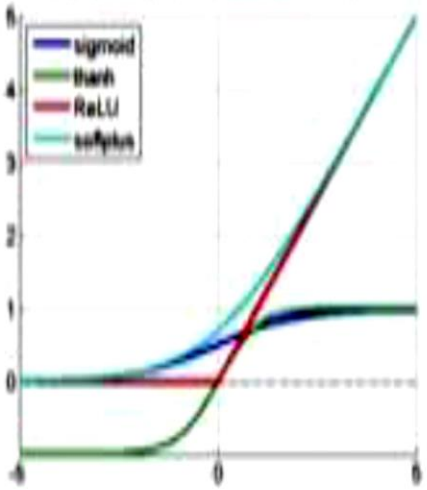


Training of neural networks

- Forward Propagation :
  - Sum inputs, produce activation
  - feed-forward



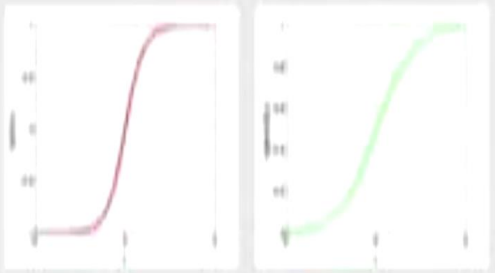
Activation Functions examples



Activation Functions

Non-linearity

- $\tanh(x)=\frac{e^x-e^{-x}}{e^x+e^{-x}}$

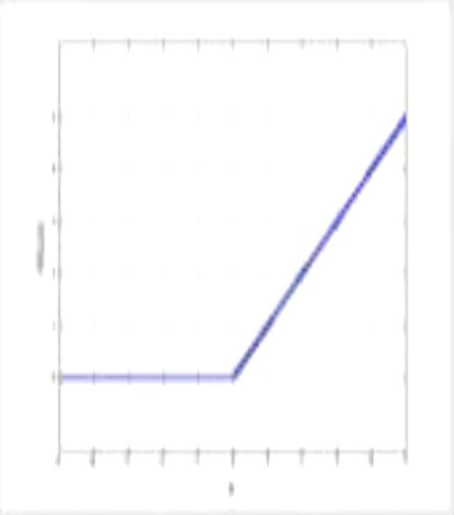


- $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$

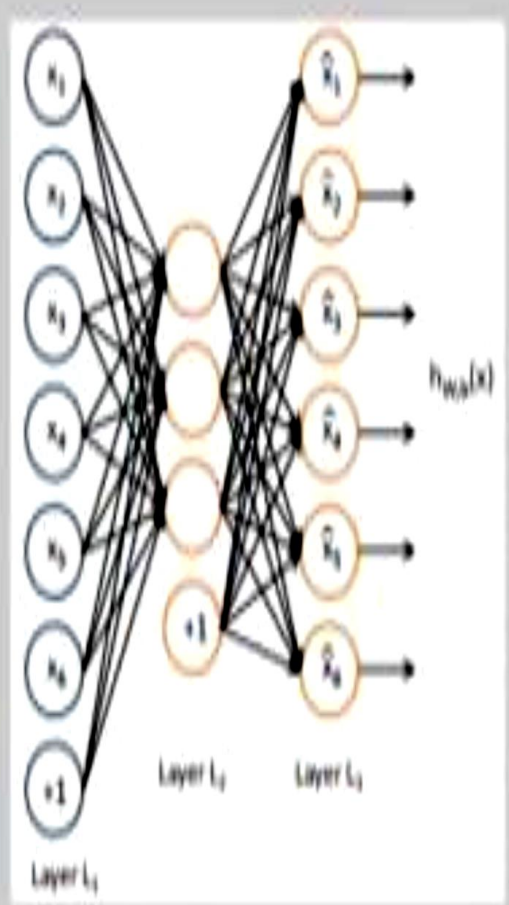
- Rectified linear
  relu(x) = max(0,x)

- Simplifies backprop
- Makes learning faster
- Make feature sparse

→ Preferred option



Autoencoder



Unlabeled training examples set

$\{x^{(1)}, x^{(2)}, x^{(3)} \ldots\}$, $x^{(i)} \in \mathbb{R}^n$

Set the target values to be equal to the inputs. $y^{(i)} = x^{(i)}$

Network is trained to output the input (learn identify function).

$$h_{w,b}(x) \approx x$$

Solution may be trivial!

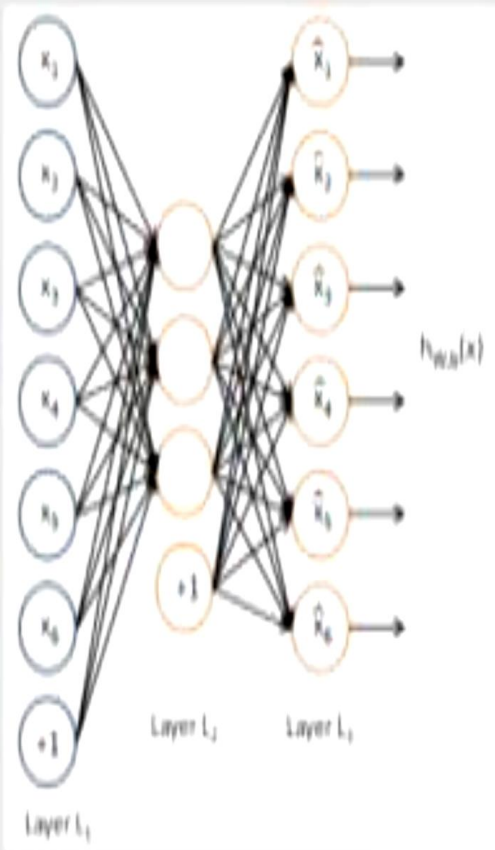
Autoencoders and sparsity

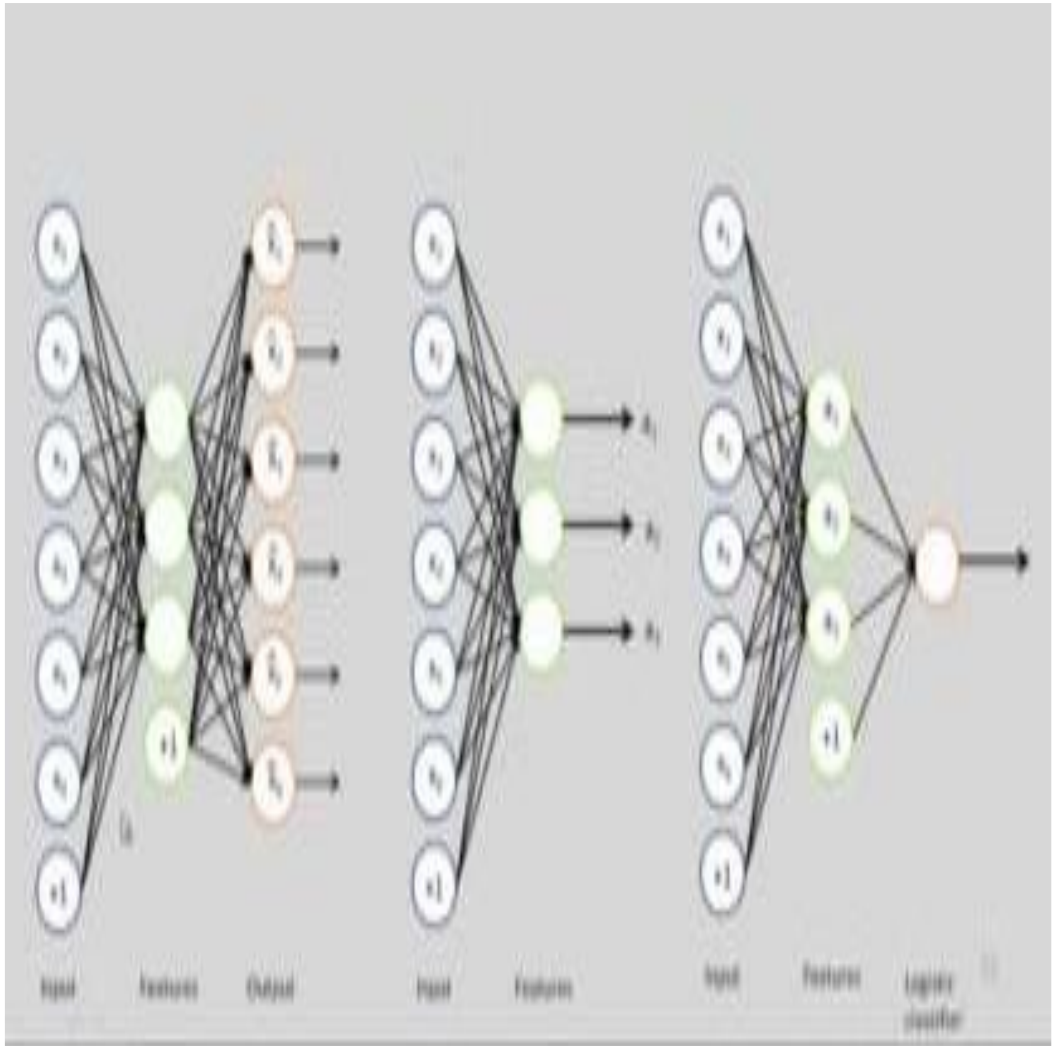
1. Place constraints on the network, like limiting the number of hidden units, to discover interesting structure about the data.
2. Impose sparsity constraint.

a neuron is "active" if its output value is close to 1

It is "inactive" if its output value is close to 0.

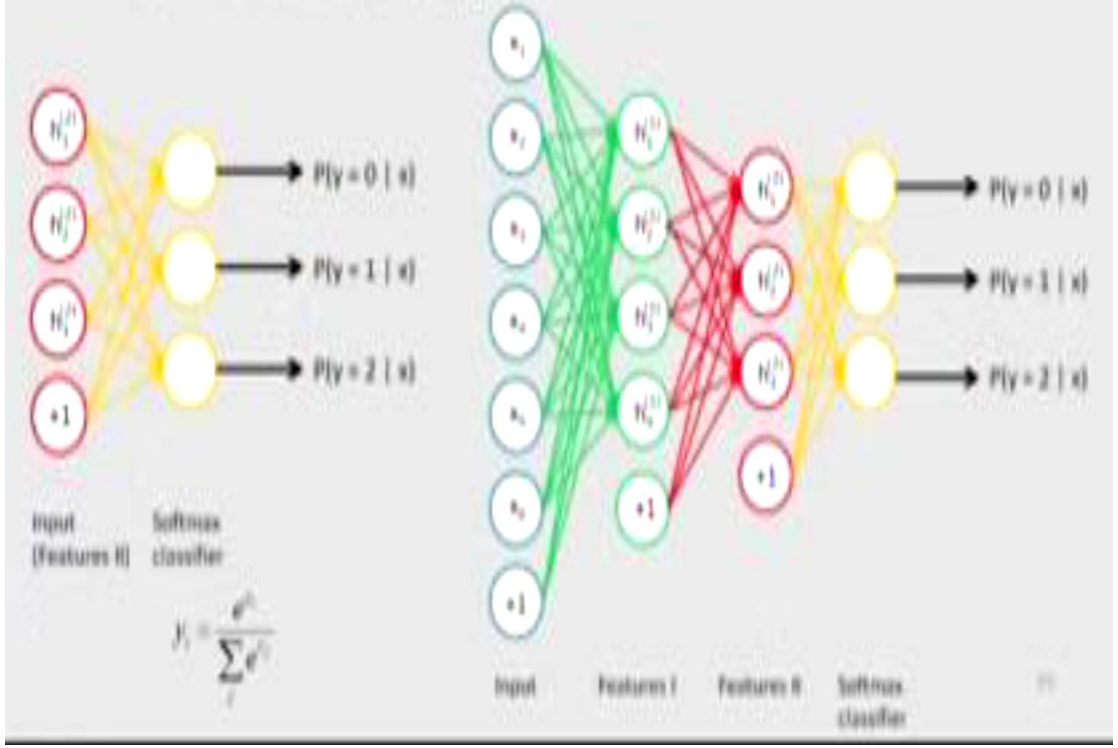
constrain the neurons to be inactive most of the time.





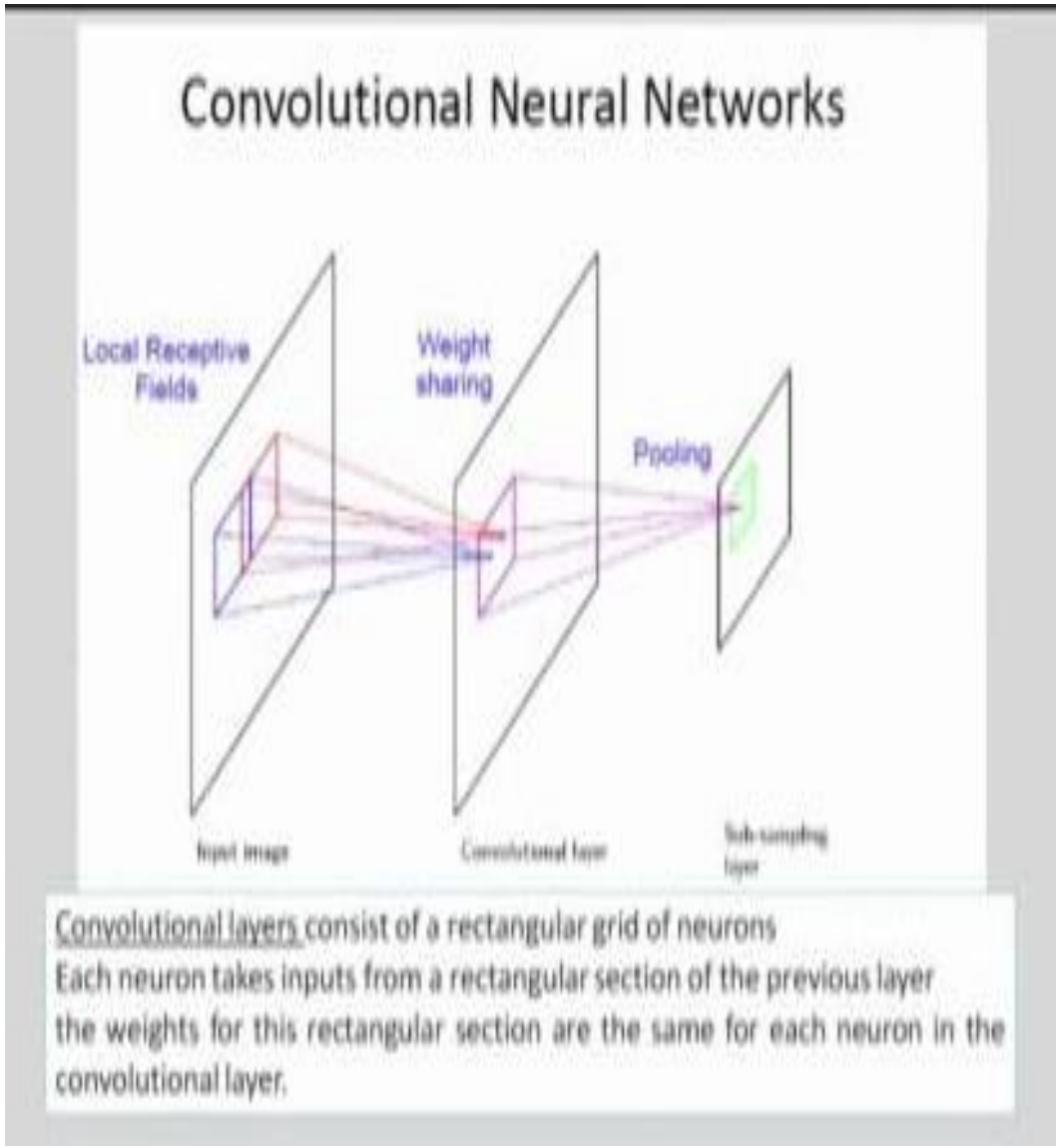
Stacked Auto-Encoders

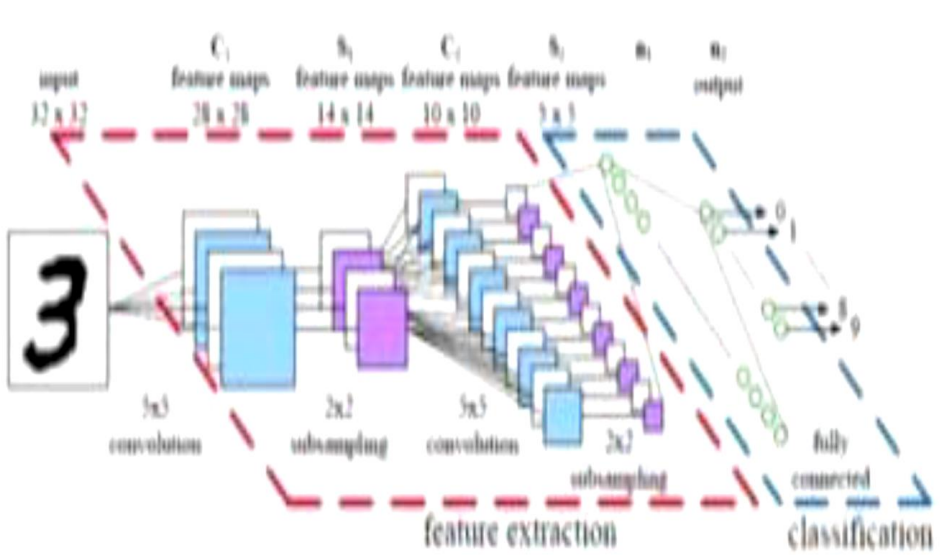
- Do supervised training on the last layer using final features
- Then do supervised training on the entire network to fine- tune all weights



Convolutional Neural netwoks

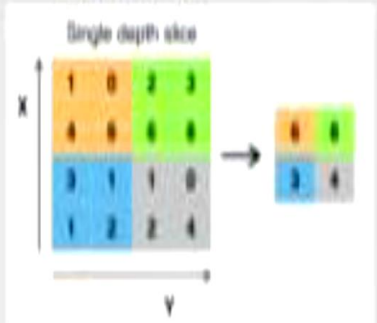
- A CNN consists of a number of convolutional and subsampling layers.
- Input to a convolutional layer is a m x m x r image where m x m is the height and width of the image and r is the number of channels, e.g. an RGB image has r=3
- Convolutional layer will have k filters (or kernels)
- size n x n x q
- n is smaller than the dimension of the image and,
- q can either be the same as the number of channels r or smaller and may vary for each kernel





Pooling: Using features obtained after Convolution for Classification

The pooling layer takes small rectangular blocks from the convolutional layer and subsamples it to produce a single output from that block : max, average, etc.



CNN properties

- CNN takes advantage of the sub-structure of the input
- Achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features.
- CNN are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.

Recurrent Neural Network (RNN)