Artificial Neural Network

Dinesh K. Vishwakarma, Ph.D.

ASSOCIATE PROFESSOR, DEPARTMENT OF INFORMATION TECHNOLOGY

DELHI TECHNOLOGICAL UNIVERSITY, DELHI.

Webpage: http://www.dtu.ac.in/Web/Departments/InformationTechnology/faculty/dkvishwakarma.php

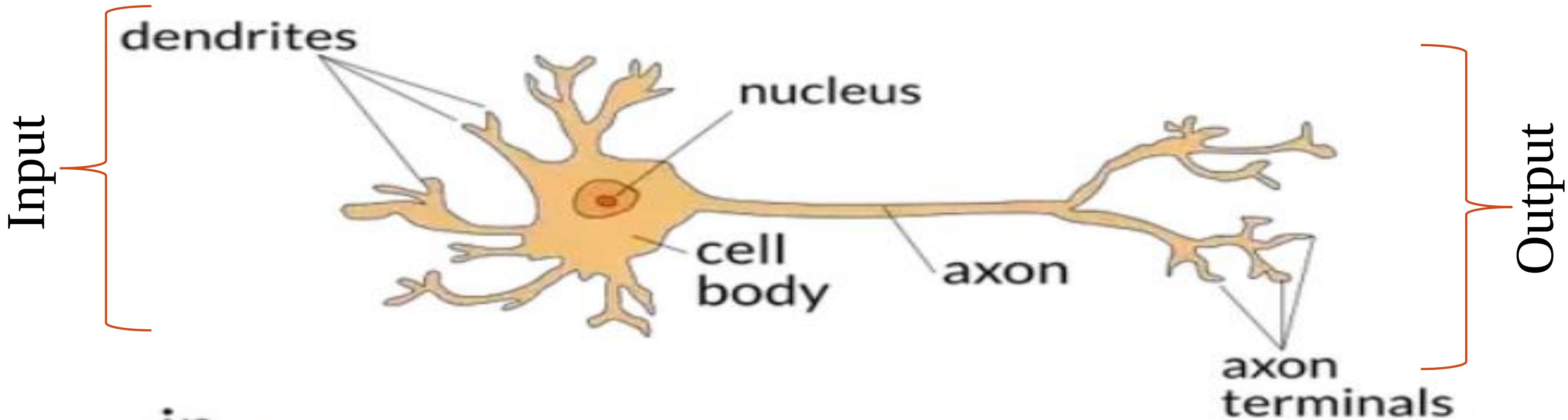
Introduction

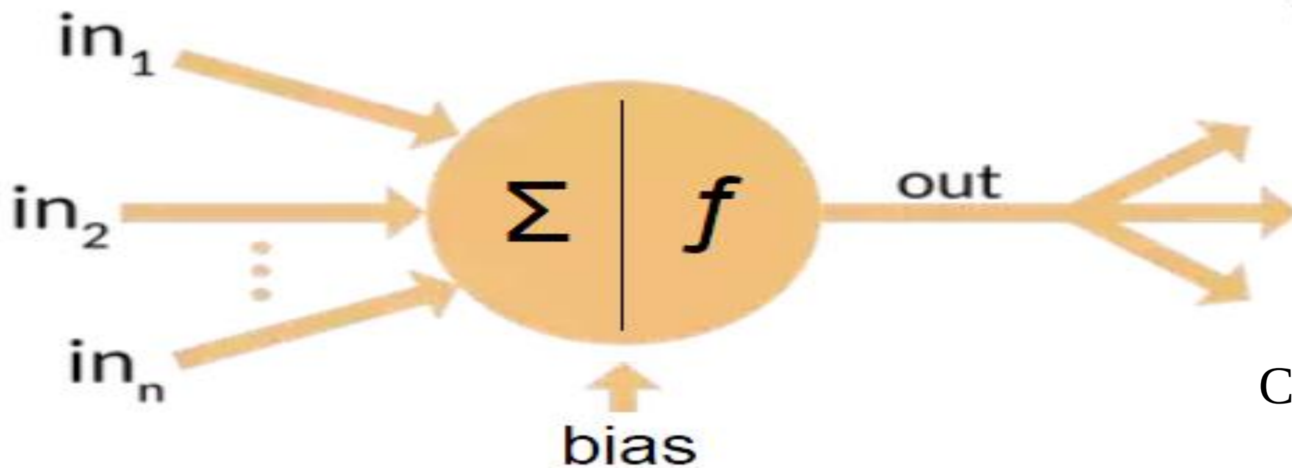
- Artificial neural networks (ANNs) provide a practical method for learning
  - ✓ real-valued functions
  - ✓ discrete-valued functions
  - ✓ vector-valued functions
- Robust to errors in training data
- Successfully applied to such problems as
  - ✓ interpreting visual scenes
  - ✓ speech recognition
  - ✓ learning robot control strategies

Introduction…

- **ANN learning well-suit to problems which the training data corresponds to noisy, complex data (inputs from cameras or microphones)**
- Can also be used for problems with symbolic representations
- **Most appropriate for problems where**
  - ✓ Instances have many attribute-value pairs
  - ✓ Target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes
  - ✓ Training examples may contain errors
  - ✓ Long training times are acceptable
  - ✓ Fast evaluation of the learned target function may be required
  - ✓ The ability for humans to understand the learned target function is not important.

Human Brain Processing





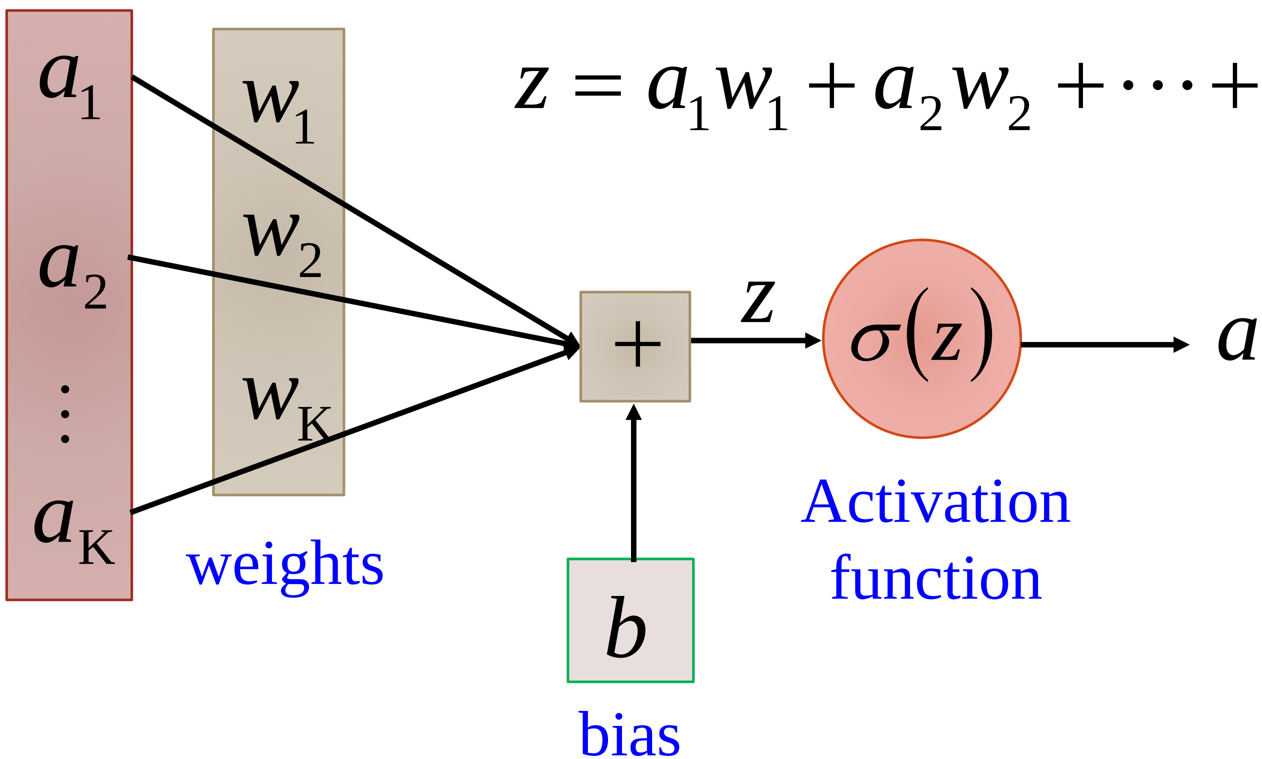
Dendrites: **Input**
Cell body: **Processor**
Synaptic: **Link**
Axon: **Output**

The human brain is made up of billions of simple processing units – neurons.

Neuron

$$f: R^K \rightarrow R$$

$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$



Neuron…

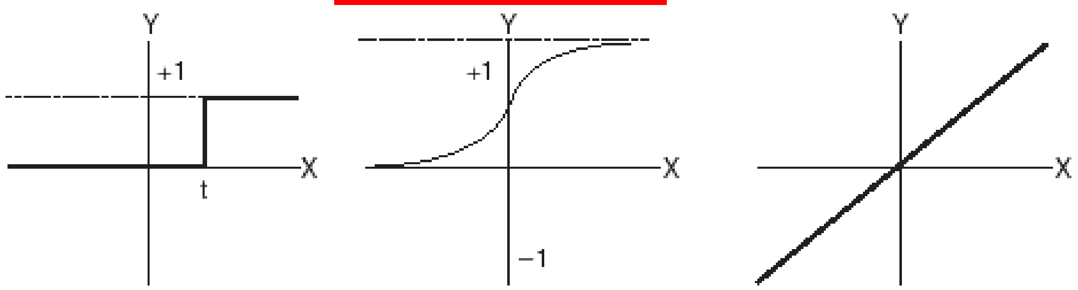
- Artificial neurons are based on biological neurons.
- Each neuron in the network receives one or more inputs.
- An activation function is applied to the inputs, which determines the output of the neuron – the activation level.

Activation functions

Activation Function works

$$X = \sum_{i=1}^{n} w_i x_i$$

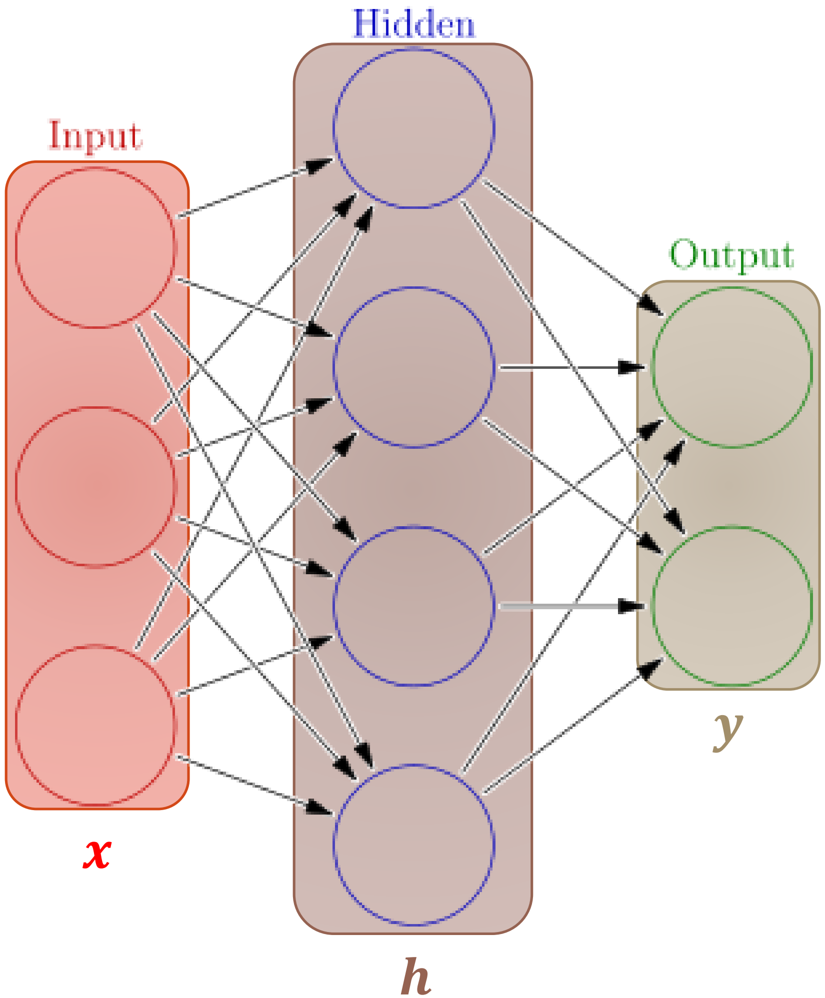
$$Y = \begin{cases} +1 & for\ X > t \\ 0 & for\ X \le t \end{cases}$$



(a) Step funtion (b) Sigmoid funtion (c) Linear funtion

6

Neural Network



Weights

$$h = \sigma(\mathbf{W}_1 x + b_1)$$

$$y = \sigma(W_2 h + b_2)$$

Activation functions

How do we train?

4 + 2 = 6 neurons (not counting inputs)

[3 x 4] + [4 x 2] = 20 weights

4 + 2 = 6 biases

26 learnable **parameters**

Training Perceptron

- Learning involves choosing values for the weights
- The perceptron is trained as follows:
  - ✓ First, inputs are given random weights (usually between –0.5 and 0.5).
  - ✓ An item of training data is presented. If the perceptron mis-classifies it, the weights are modified according to the following:
    - where t is the target output for the training example, o is the output generated by the perceptron and a is the learning rate, between 0 and 1 (usually small such as 0.1)
- Cycle through training examples until successfully classify all examples
  - ✓ Each cycle known as an **epoch**

Backpropagation

- Multilayer neural networks learn in the same way as perceptrons.
- However, there are many more weights, and it is important to assign credit (or blame) correctly when changing weights.
- *E* sums the errors over all of the network output units

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

Backpropagation Algorithm

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers.
- Until termination condition is met, Do
  - ✓ For each <x,t> in training examples, Do.
  - ✓ *Propagate the input forward through the network:*

1. Input the instance x to the network and compute the output o_u of every unit u in the network.
2. *Propagate the errors backward through the network:*
3. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

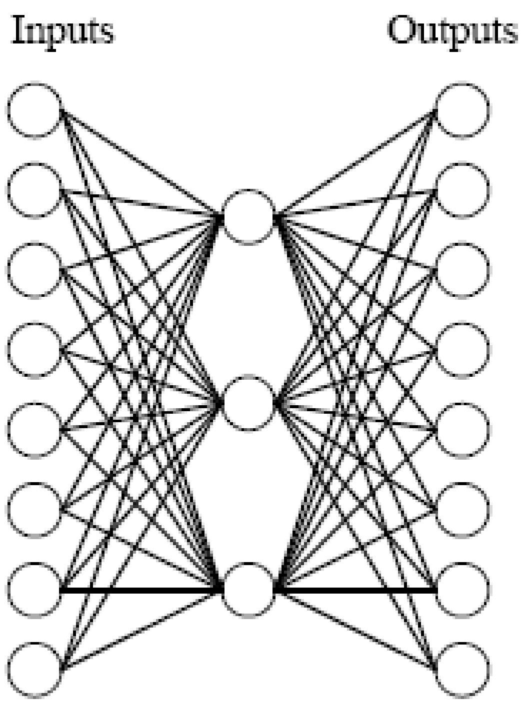
4. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{kh}\delta_k$$

5. Update each network weight w_{ji} where

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji} \qquad \Delta w_{ji} = \alpha\delta_j x_{ji}$$

Hidden Layer representation



Target Function:

Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

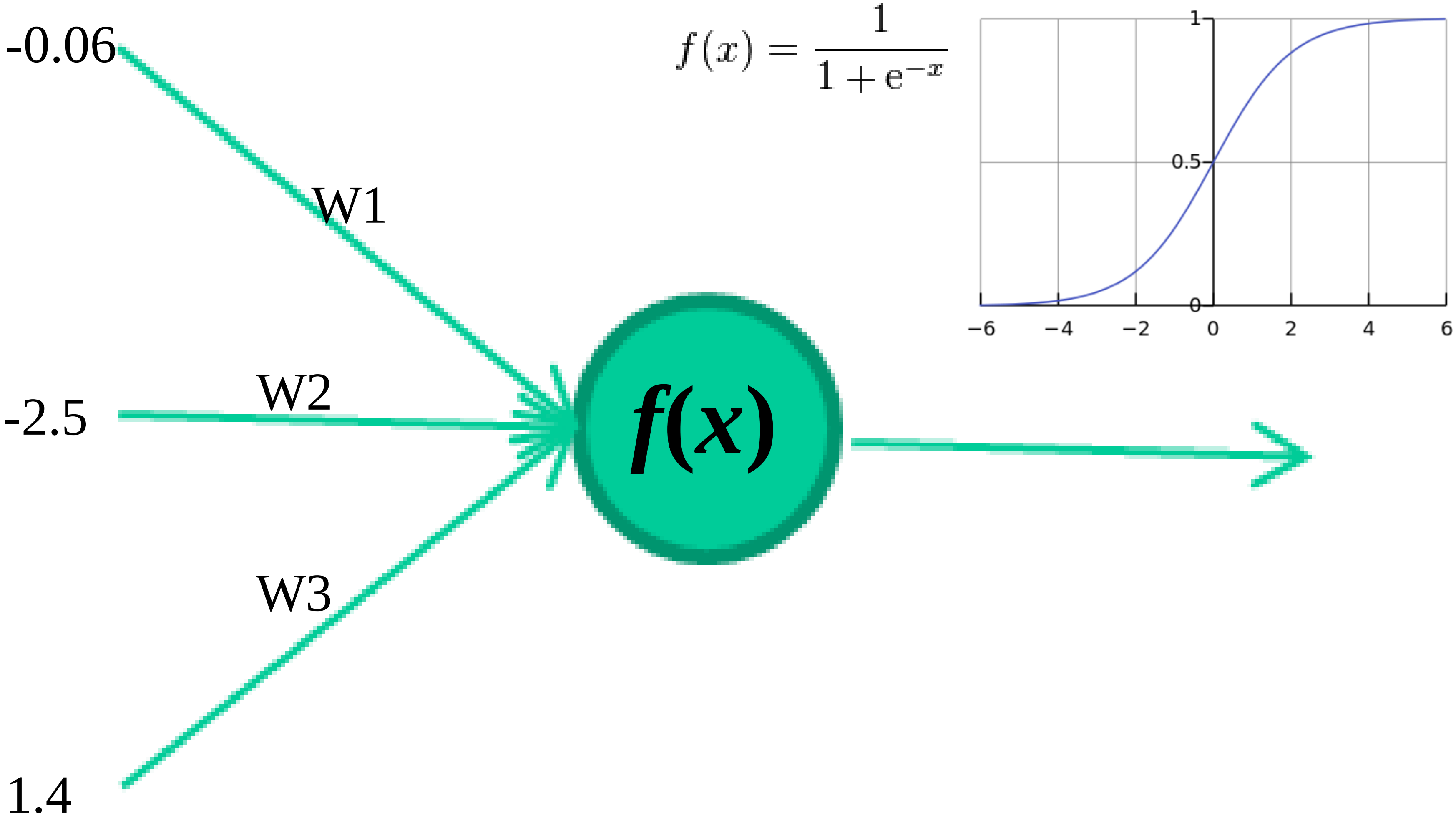
Can this be learned?

Yes

Input		Hidden Values		Output
10000000	→	.89 .04 .08	→	10000000
01000000	→	.15 .99 .99	→	01000000
00100000	→	.01 .97 .27	→	00100000
00010000	→	.99 .97 .71	→	00010000
00001000	→	.03 .05 .02	→	00001000
00000100	→	.01 .11 .88	→	00000100
00000010	→	.80 .01 .98	→	00000010
00000001	→	.60 .94 .01	→	00000001

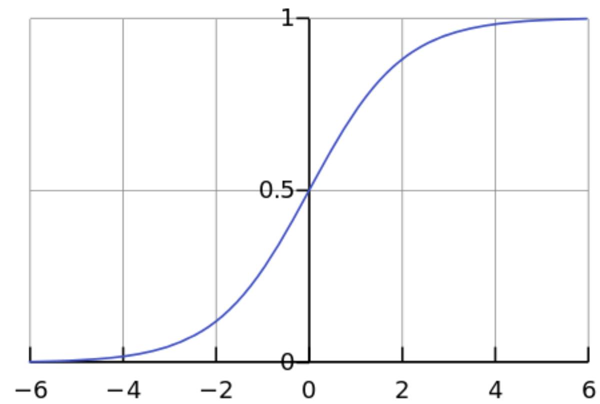
Example 1 of NN

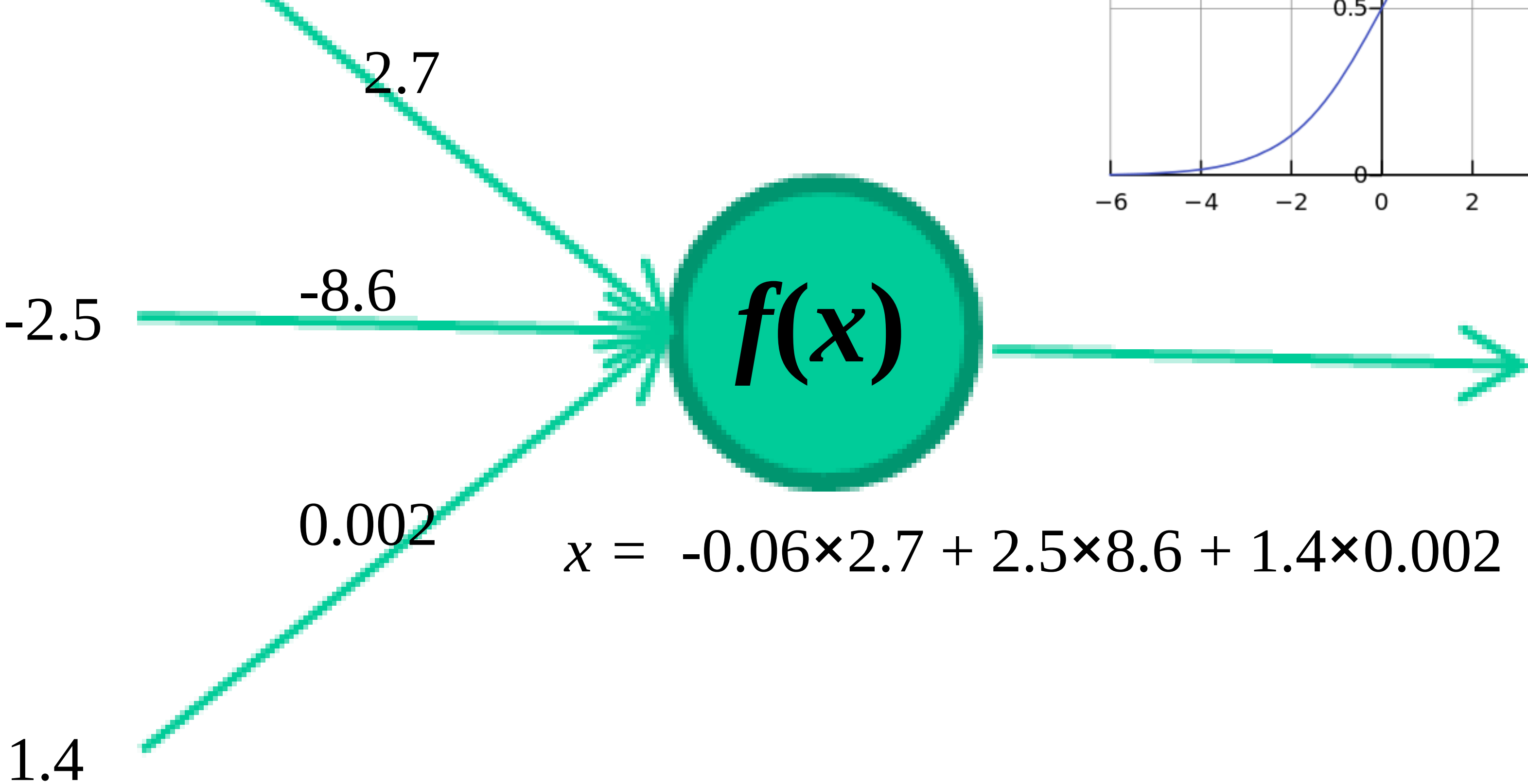
$$f(x) = \frac{1}{1 + e^{-x}}$$



Example 1 of NN…

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



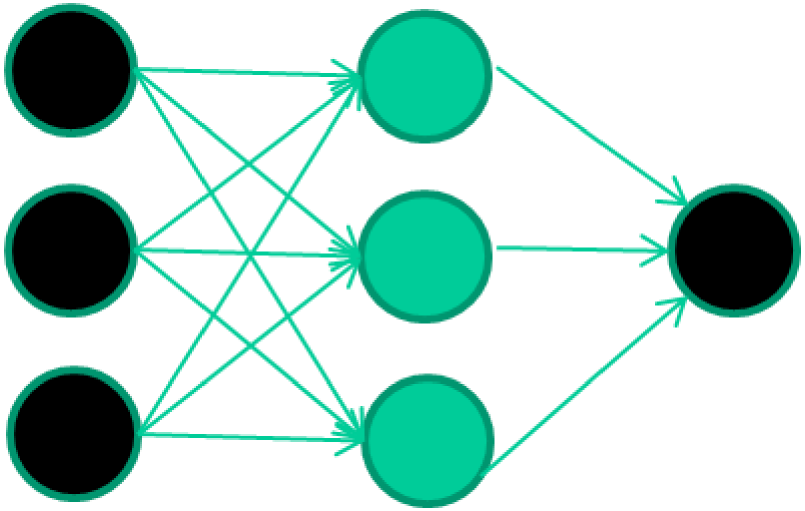


x = -0.06×2.7 + 2.5×8.6 + 1.4×0.002 = 21.34

Example 1 of NN…

A dataset

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

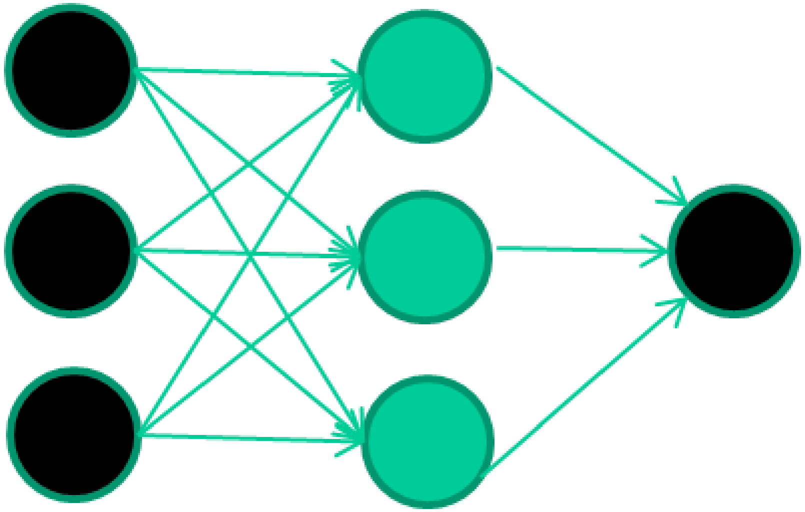


Example 1 of NN…

Training the neural network

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

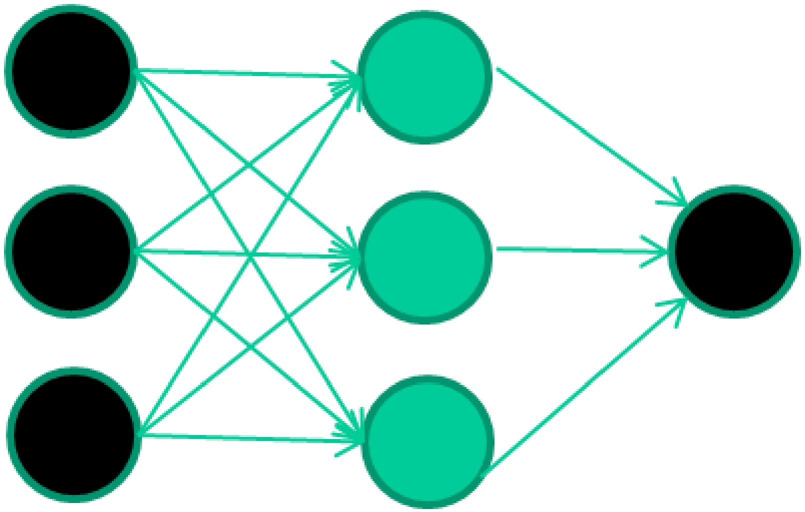


Example 1 of NN…

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Initialise with random weights

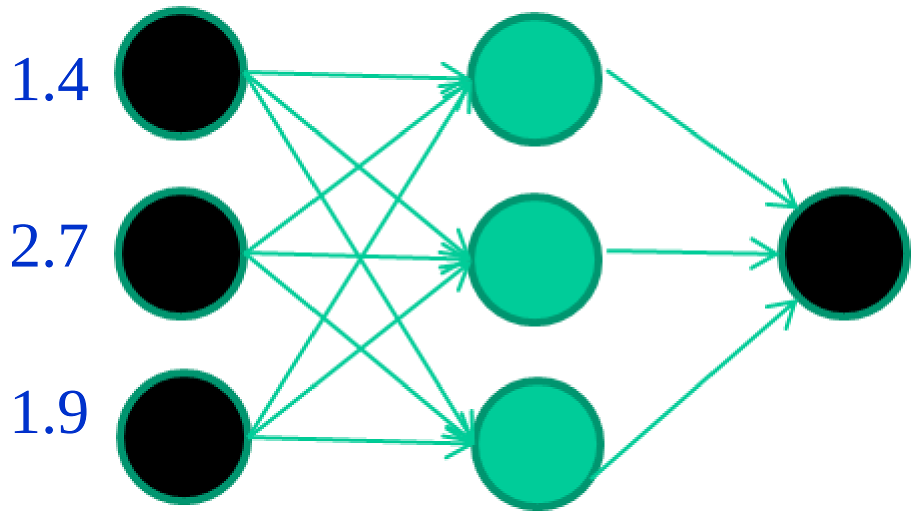


Example 1 of NN…

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Present a training pattern

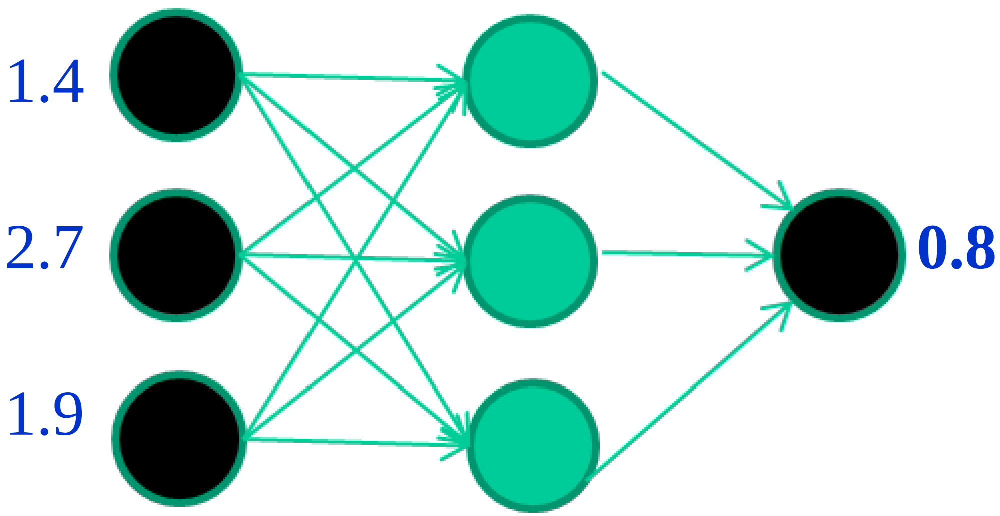


Example 1 of NN...

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Feed it through to get output

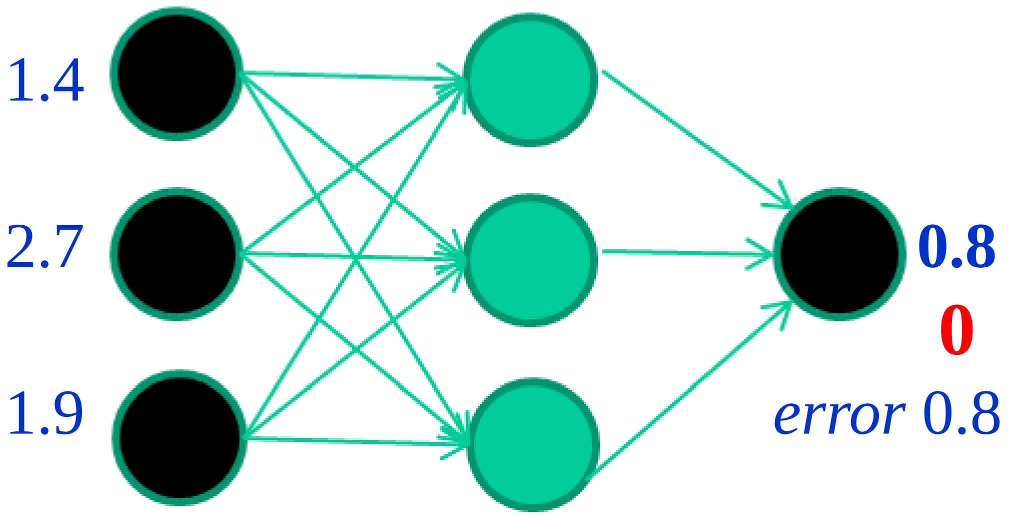


Example 1 of NN…

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Compare with target output

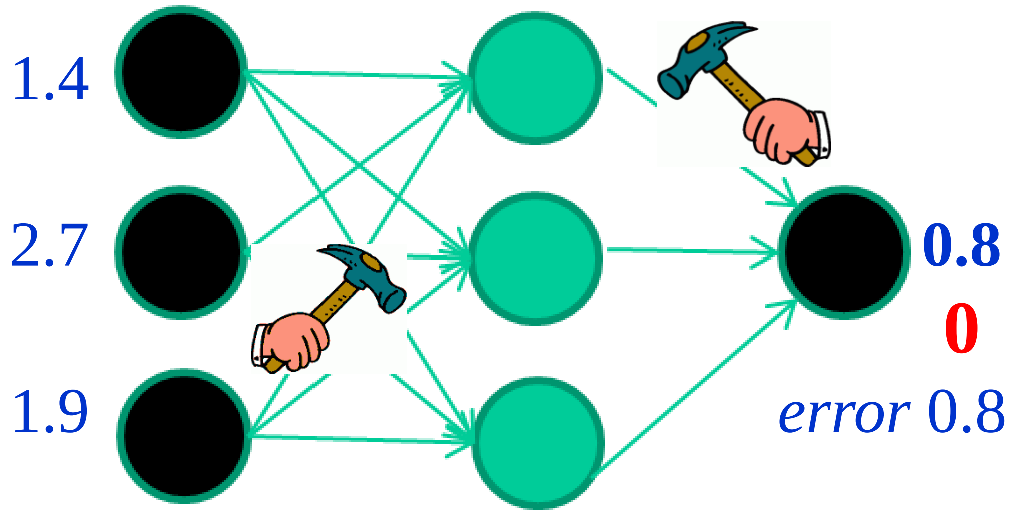


Example 1 of NN…

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Adjust weights based on error

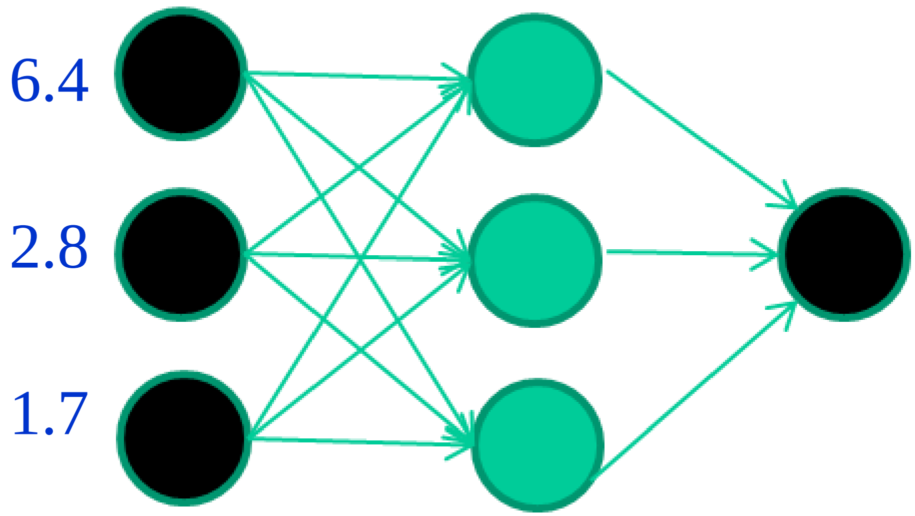


Example 1 of NN…

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Present a training pattern



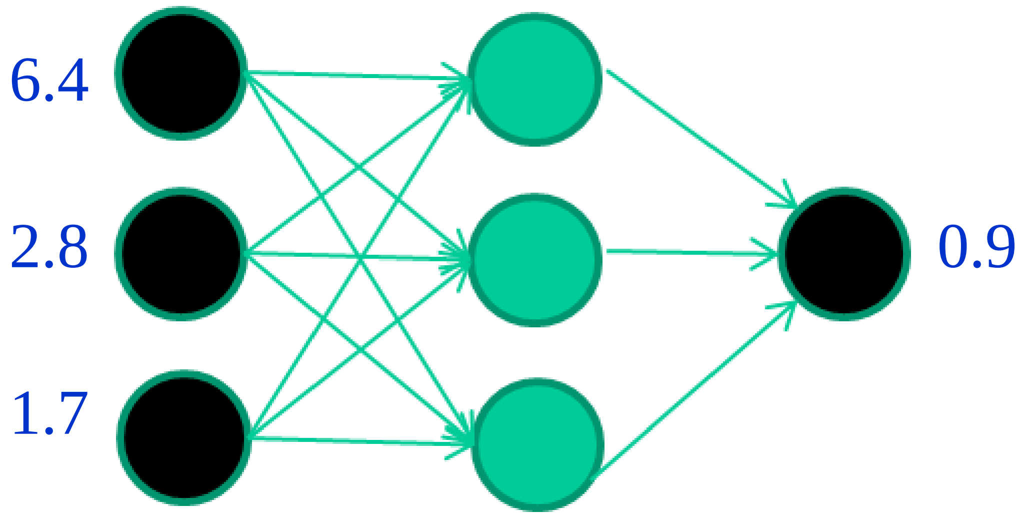
Example 1 of NN...

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

Feed it through to get output

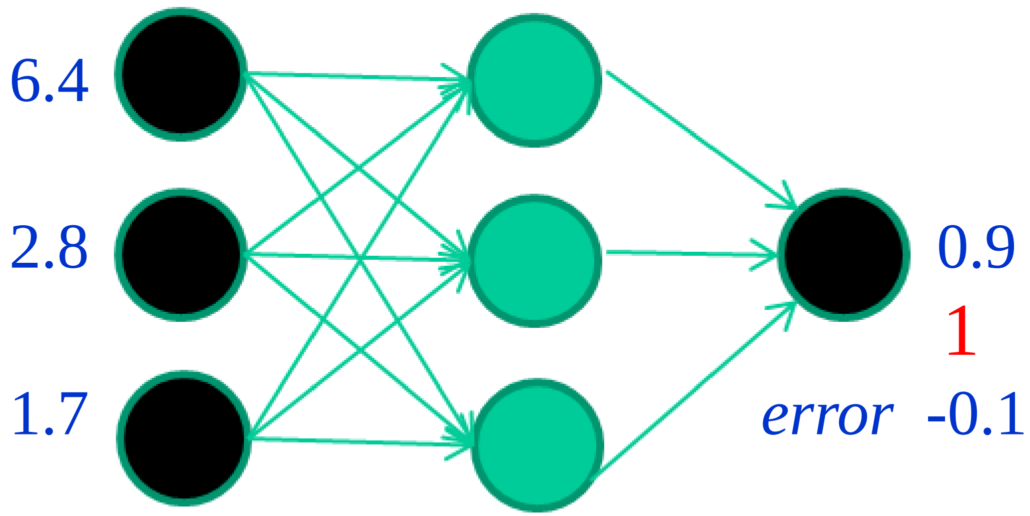


Example 1 of NN…

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Compare with target output

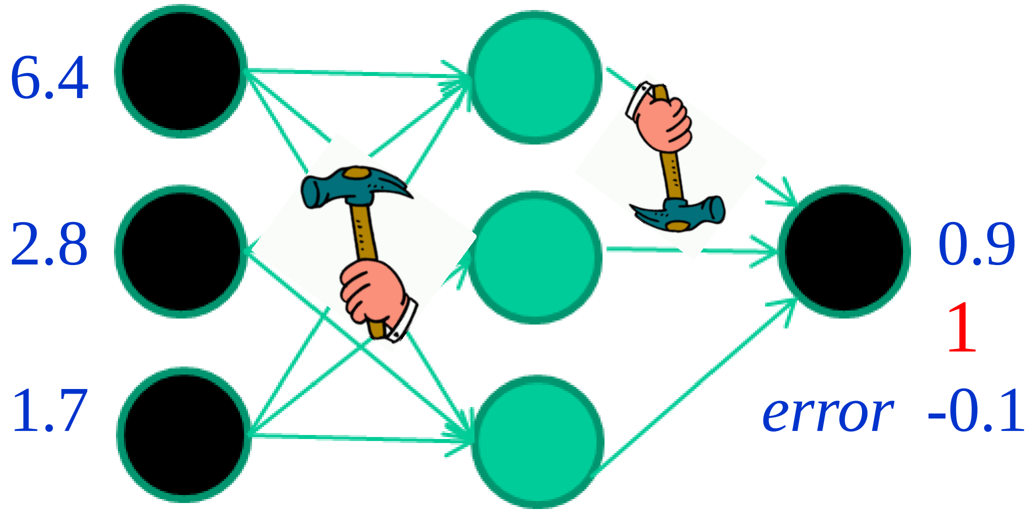


Example 1 of NN…

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Adjust weights based on error

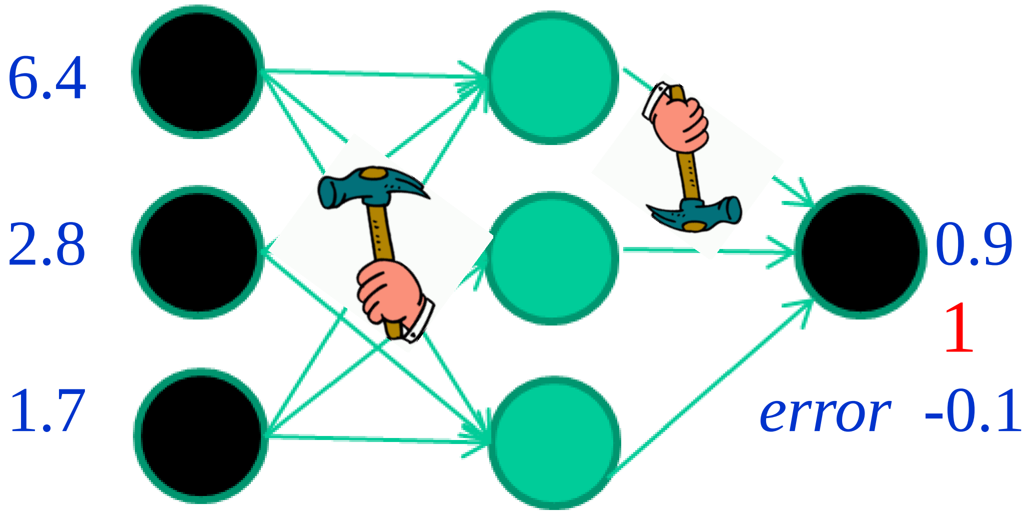


Example 1 of NN…

Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

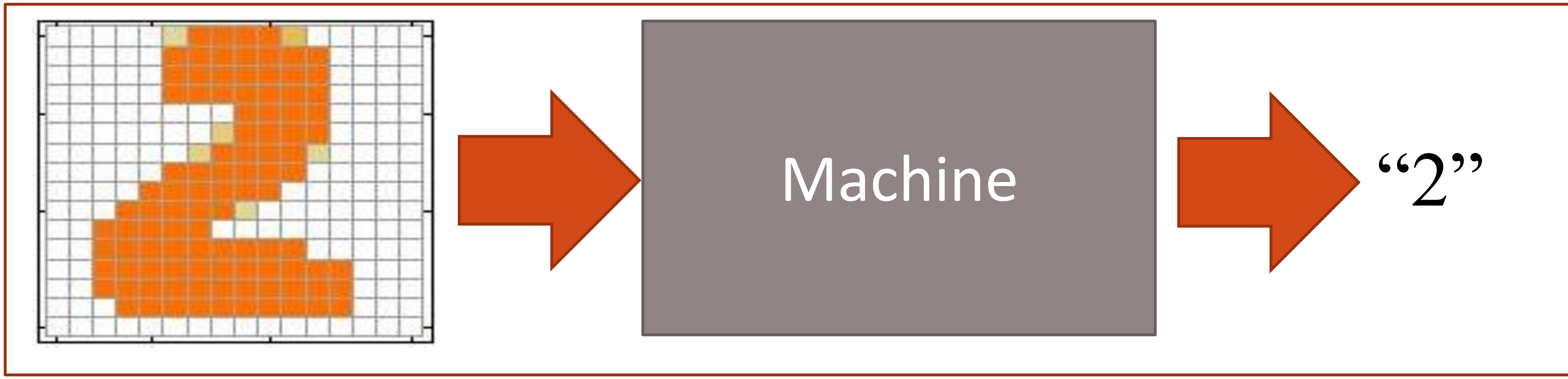
And so on ….

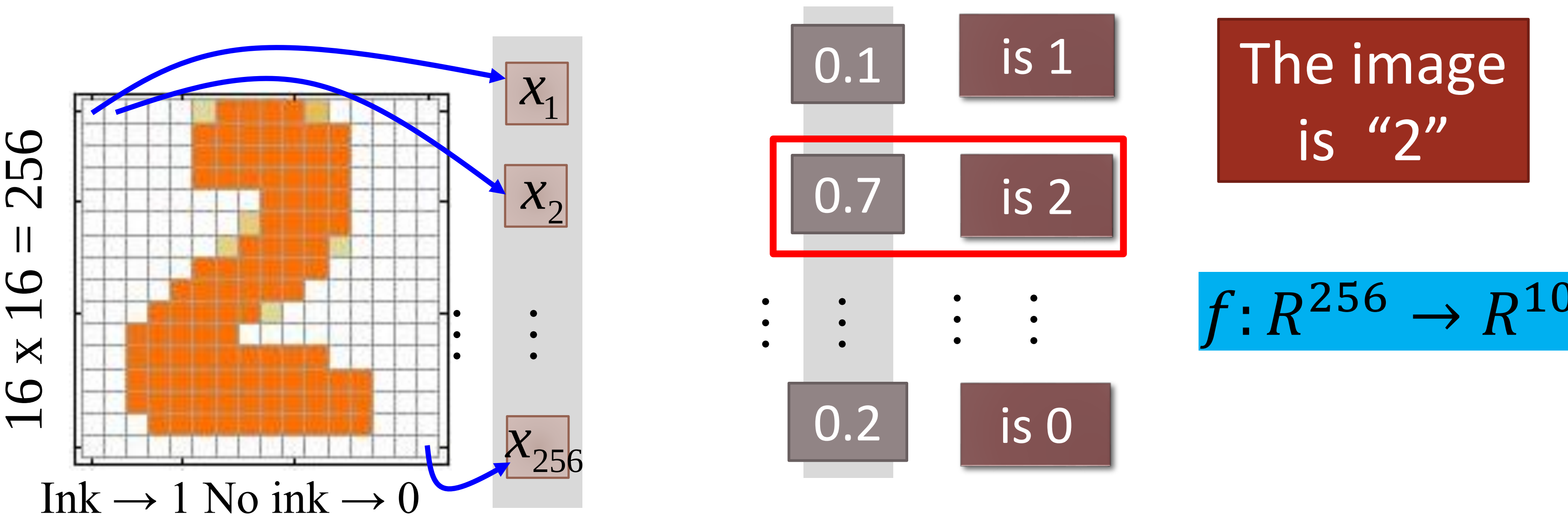


Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

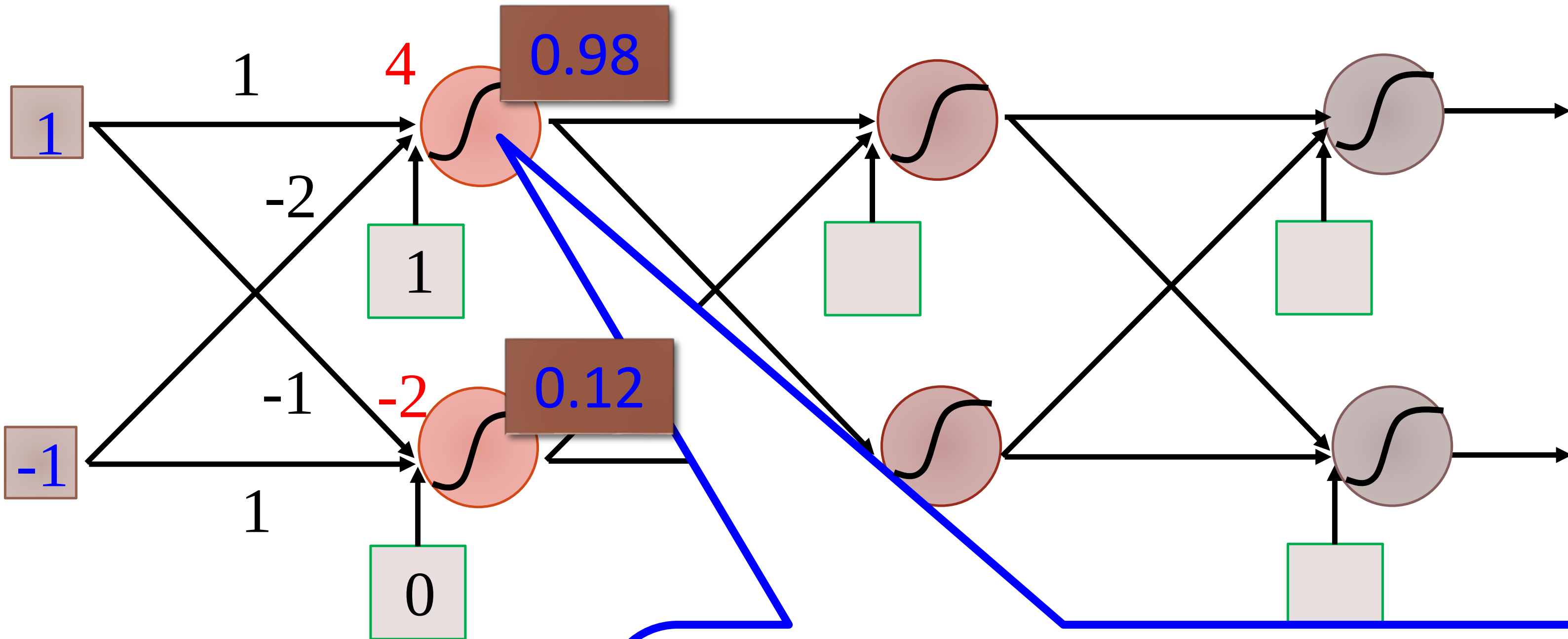
Algorithms for weight adjustment are designed to make changes that will reduce the error

Example of Digit Recognition



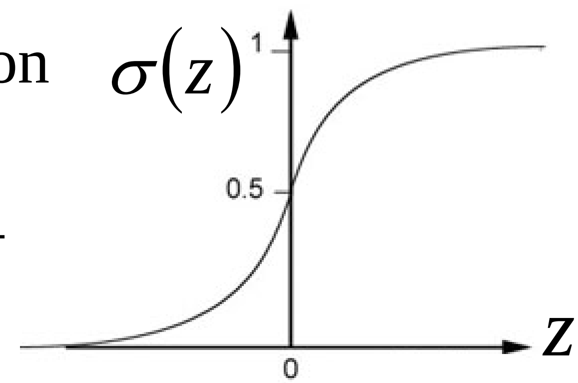


Example of Neural Network

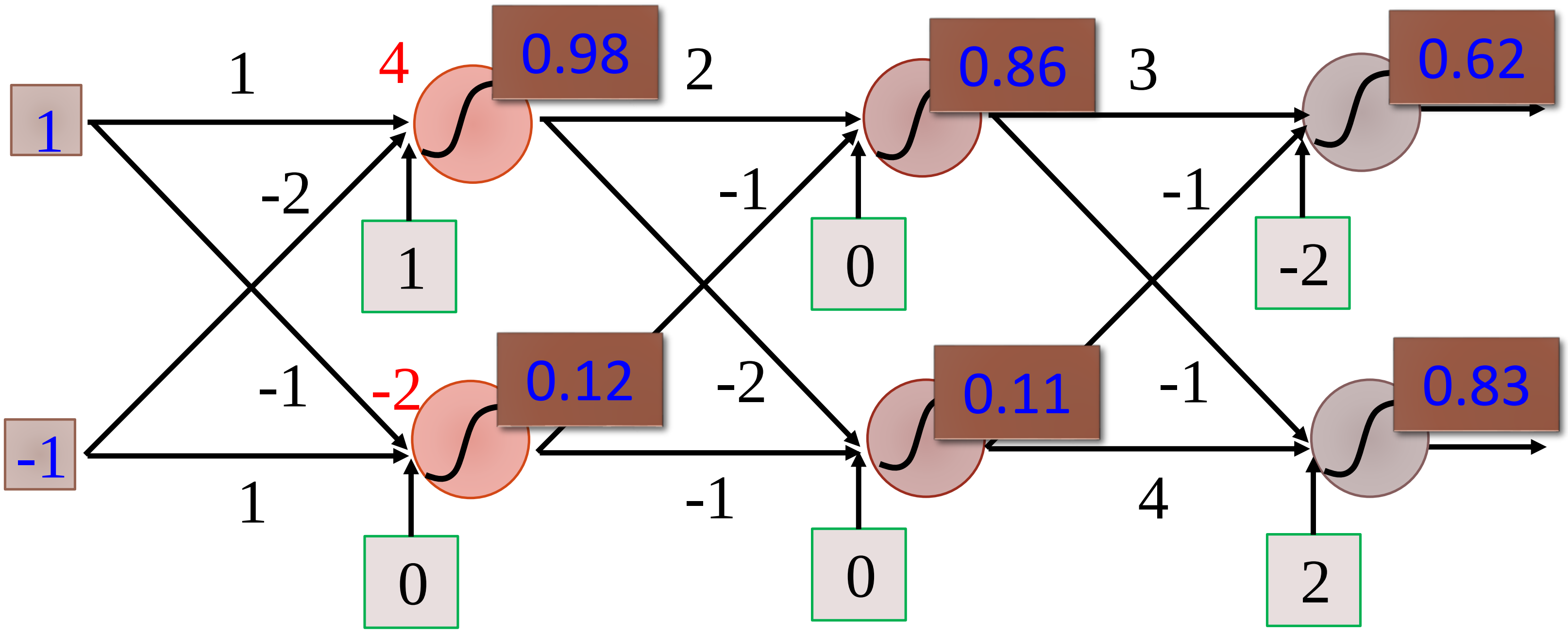


Sigmoid Function

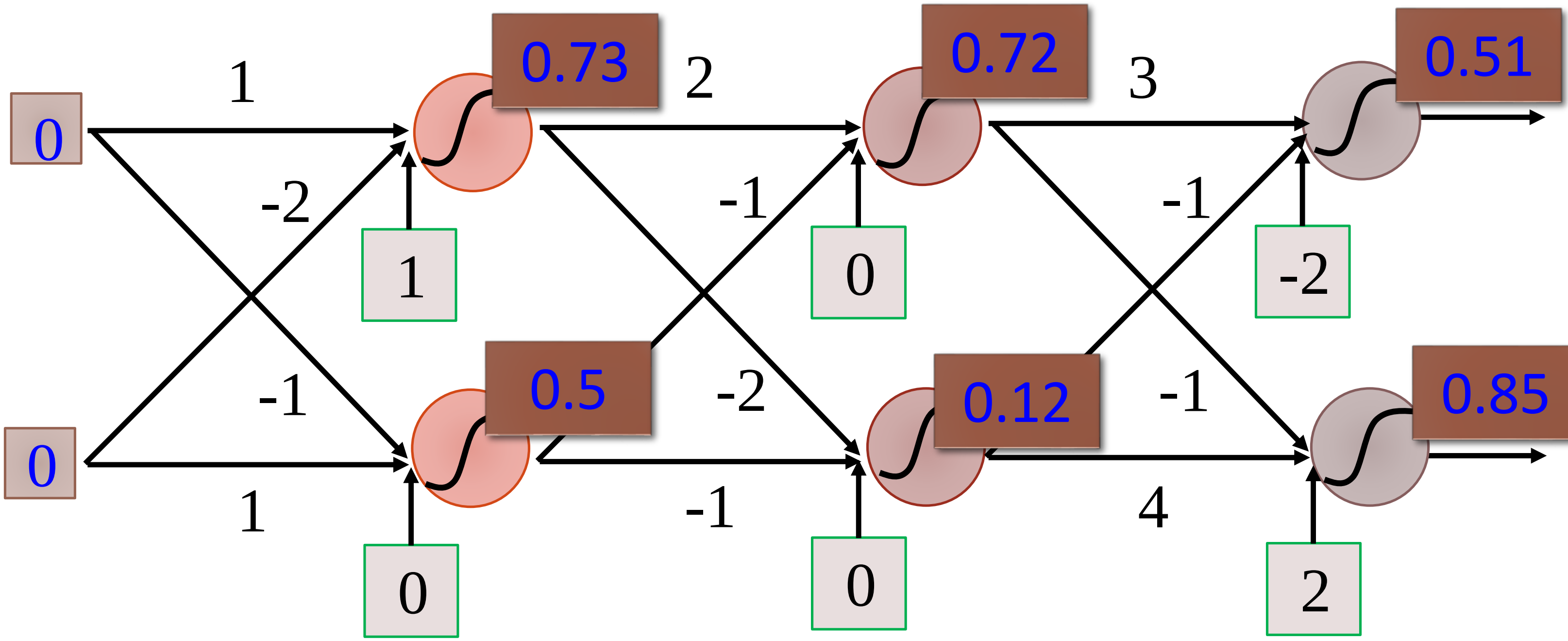
$$\sigma(z) = \frac{1}{1+e^{-z}}$$



Example of Neural Network



Example of Neural Network

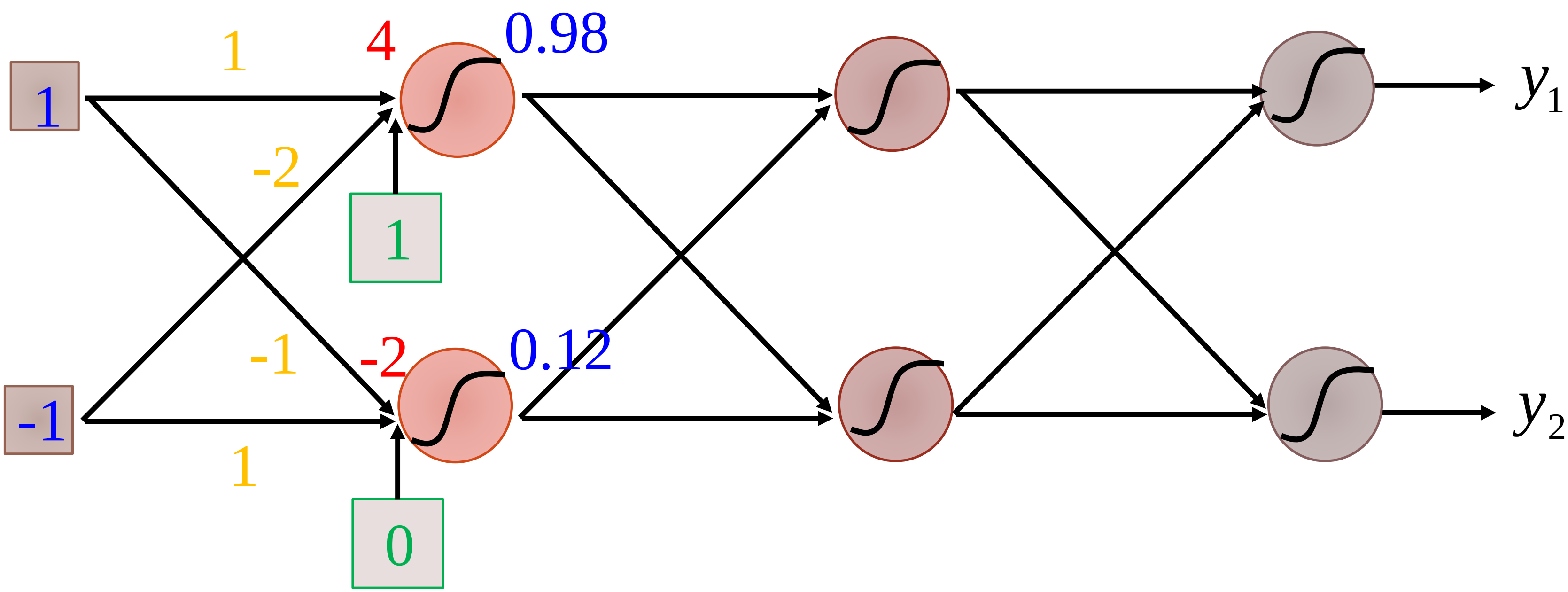


$f: R^2 \rightarrow R^2$

$$f\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix} \quad f\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 0.51 \\ 0.85 \end{bmatrix}$$

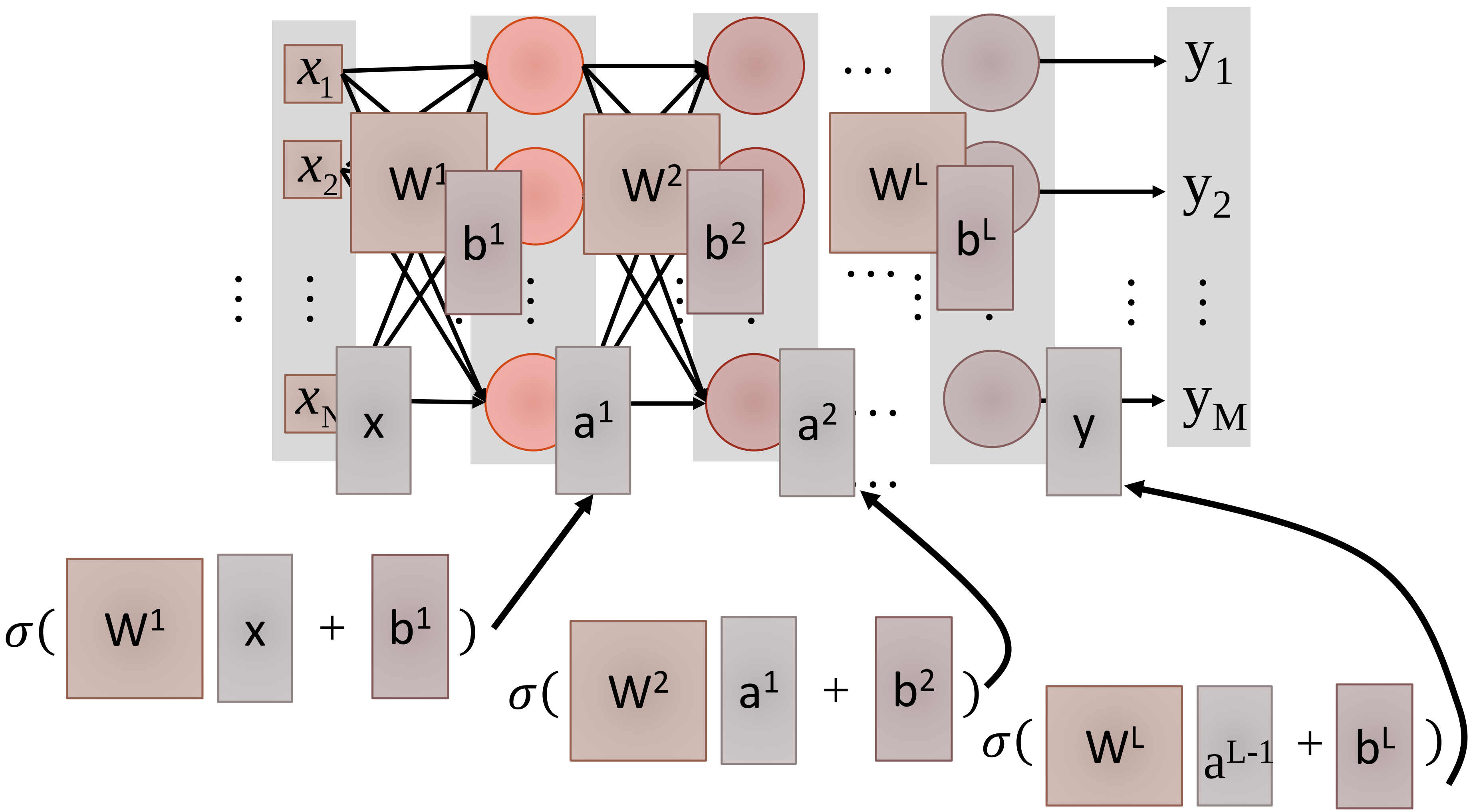
Different parameters define different function

Example of Neural Network

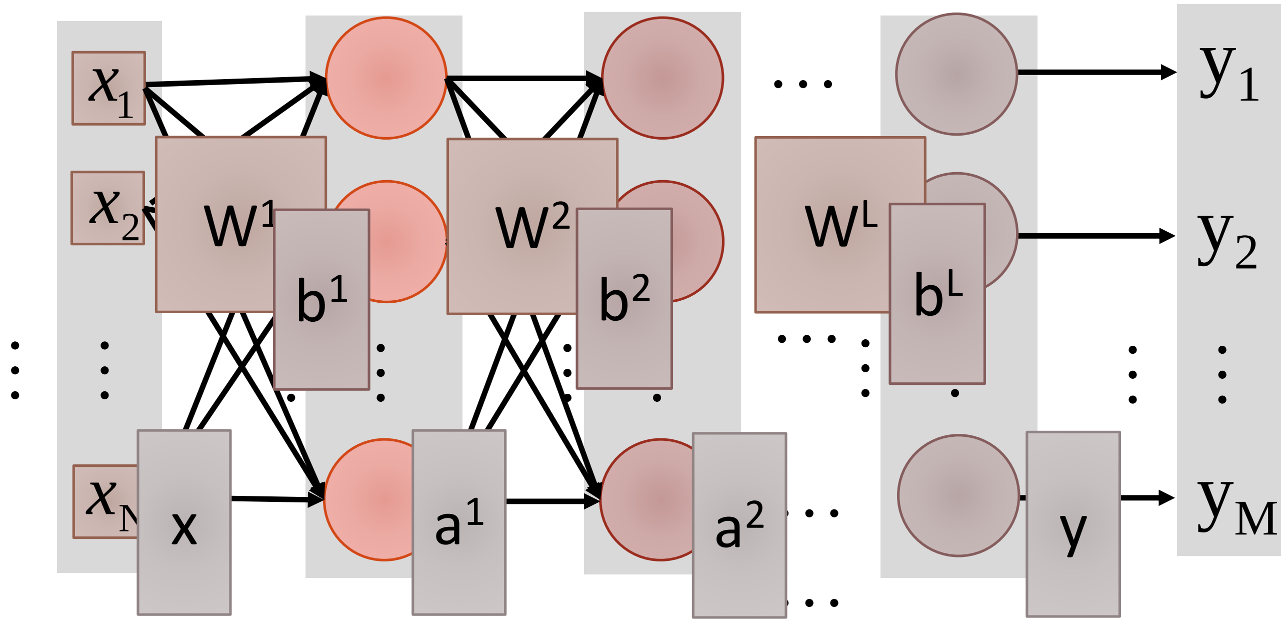


$$\sigma\Big(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}}\Big) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Example of Neural Network



Neural Network



$$y = f(x)$$

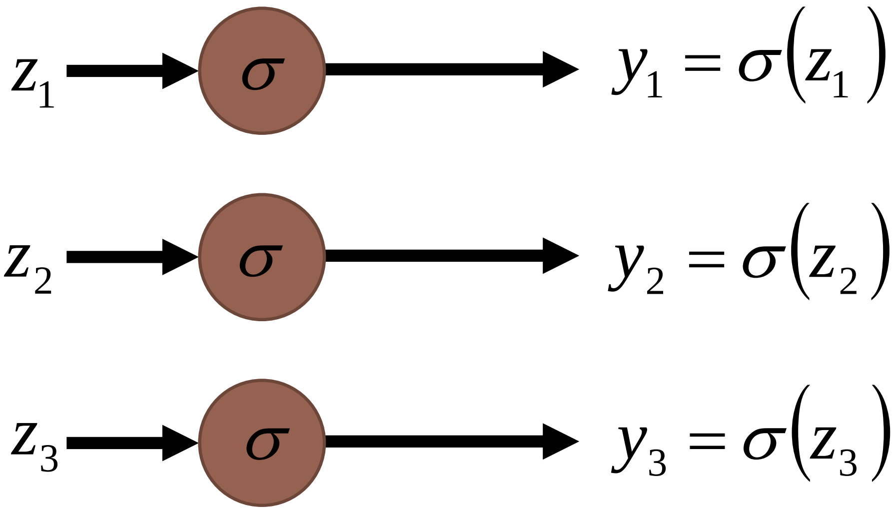
Using parallel computing techniques to speed up matrix operation

$$= \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \cdots + b^L)$$

Softmax

- Softmax layer as the output layer

<u>Ordinary Layer</u>



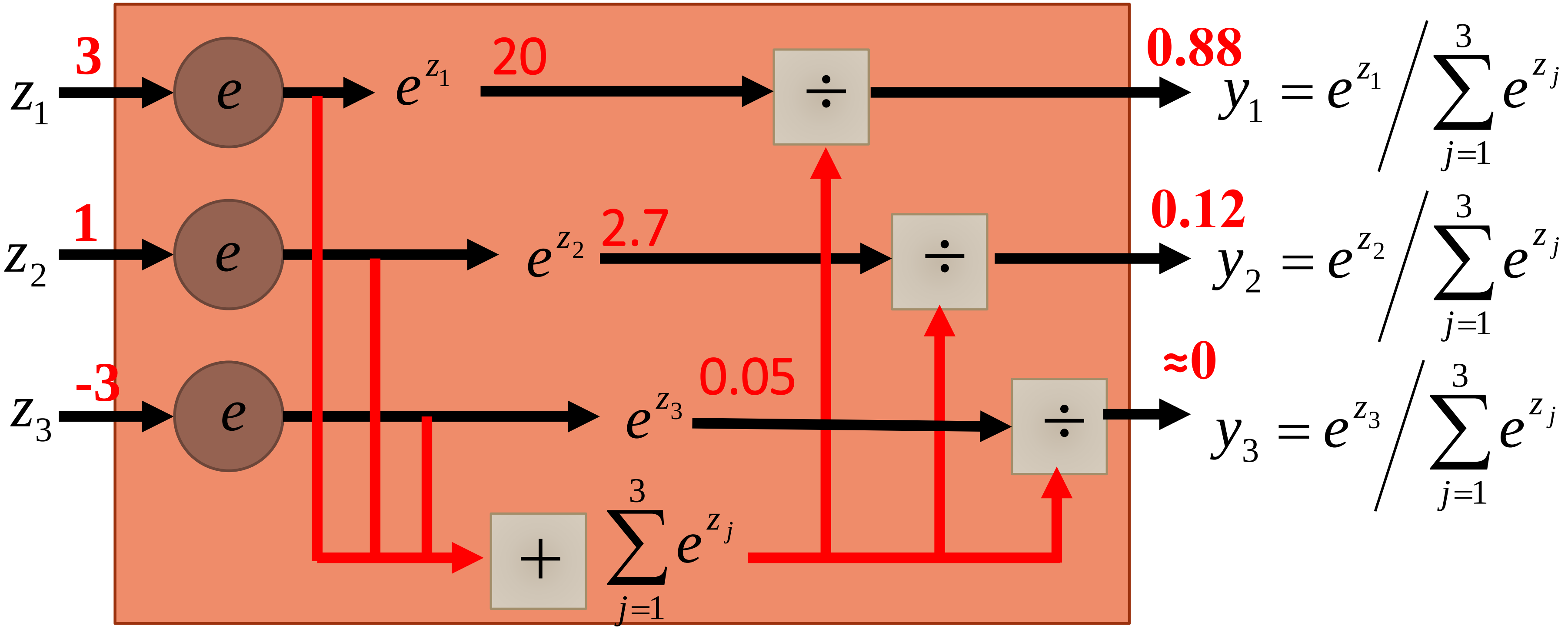
In general, the output of network can be any value.

May not be easy to interpret

Softmax

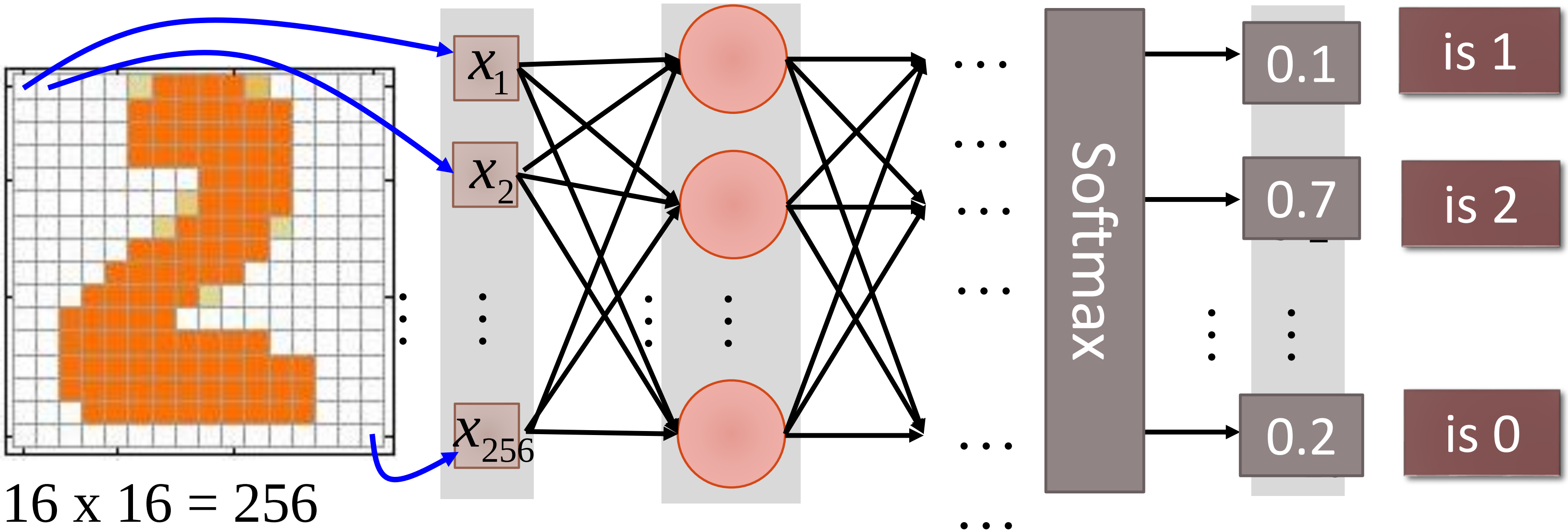
- Softmax layer as the output layer

***Probability*:**
- $1 > y_i > 0$
- $\sum_i y_i = 1$



Network Parameters

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$

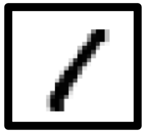


16 x 16 = 256

Ink → 1

No ink → 0

Set the network parameters such that ……

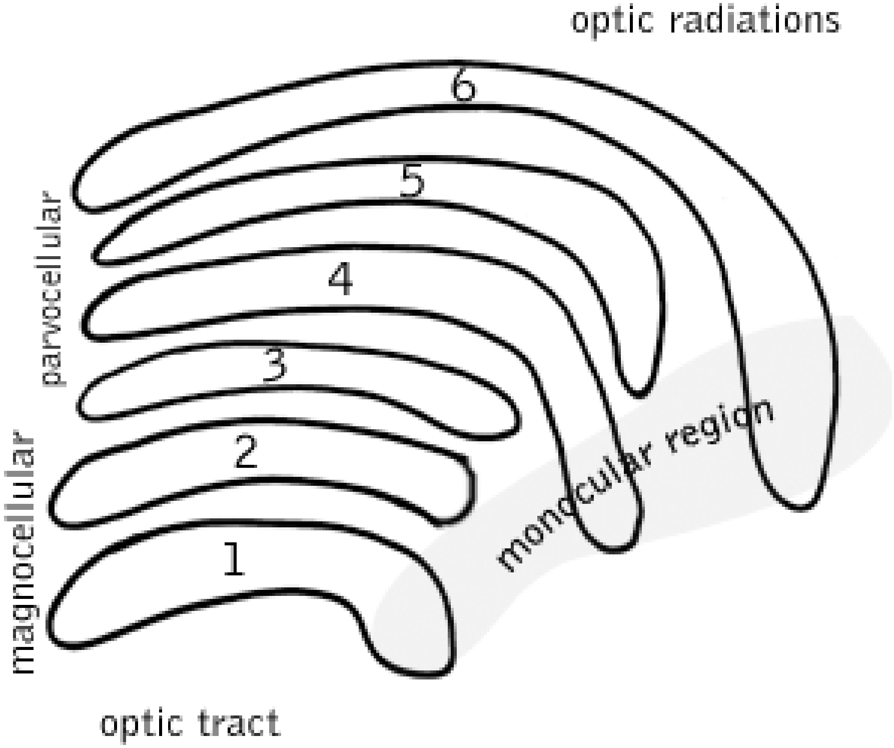


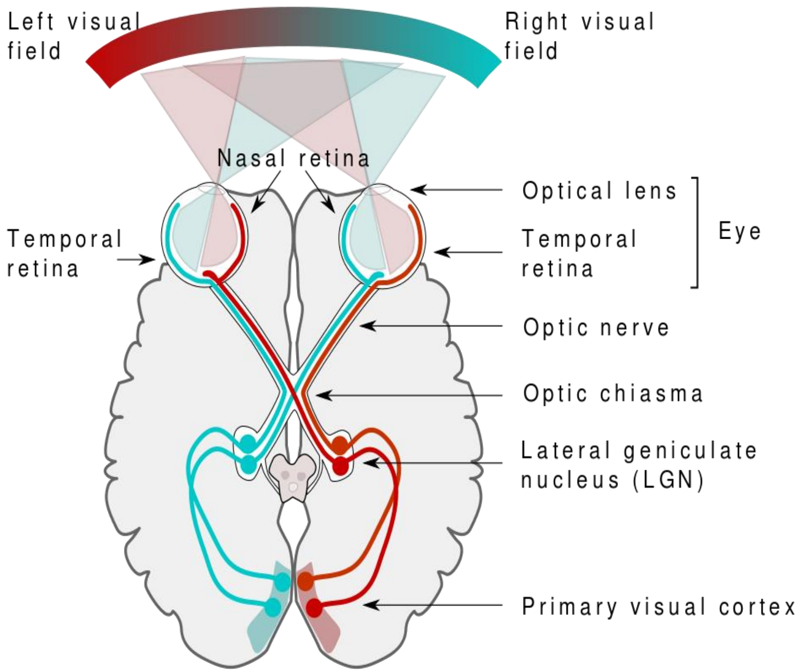
Input: → y_1 has the maximum value

Input: → y_2 has the maximum value

Visual Information Processing

- Visual information processed by our brain is multi-layered.



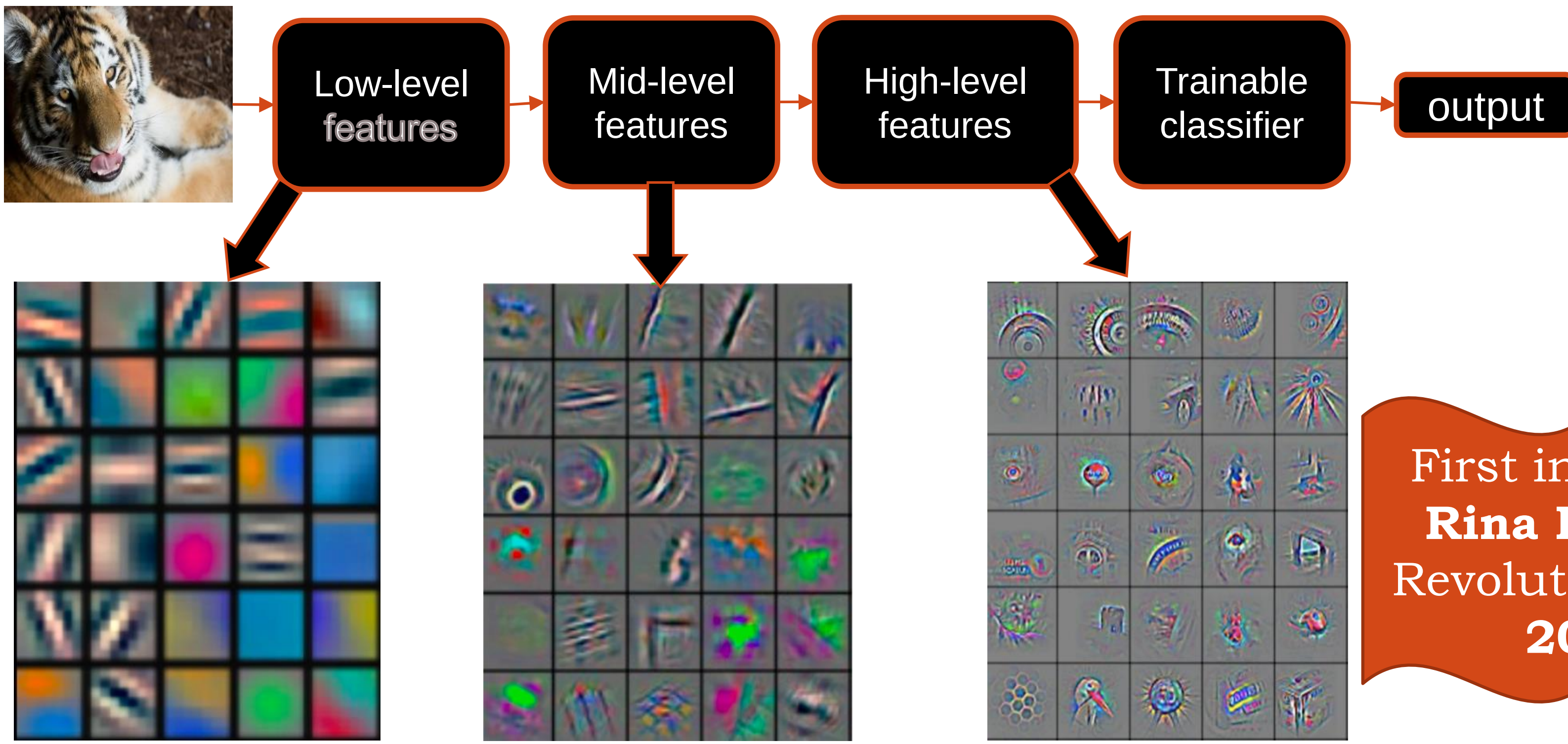


Enabling Factor of DL

- Training of deep networks was made computationally feasible by:
  - Faster CPU's
  - The move to parallel CPU architectures
  - **Advent of GPU computing**
- Neural networks are often represented as a matrix of weight vectors.
- GPU's are optimized for very fast matrix multiplication
- 2008 - Nvidia's CUDA library for GPU computing is released.

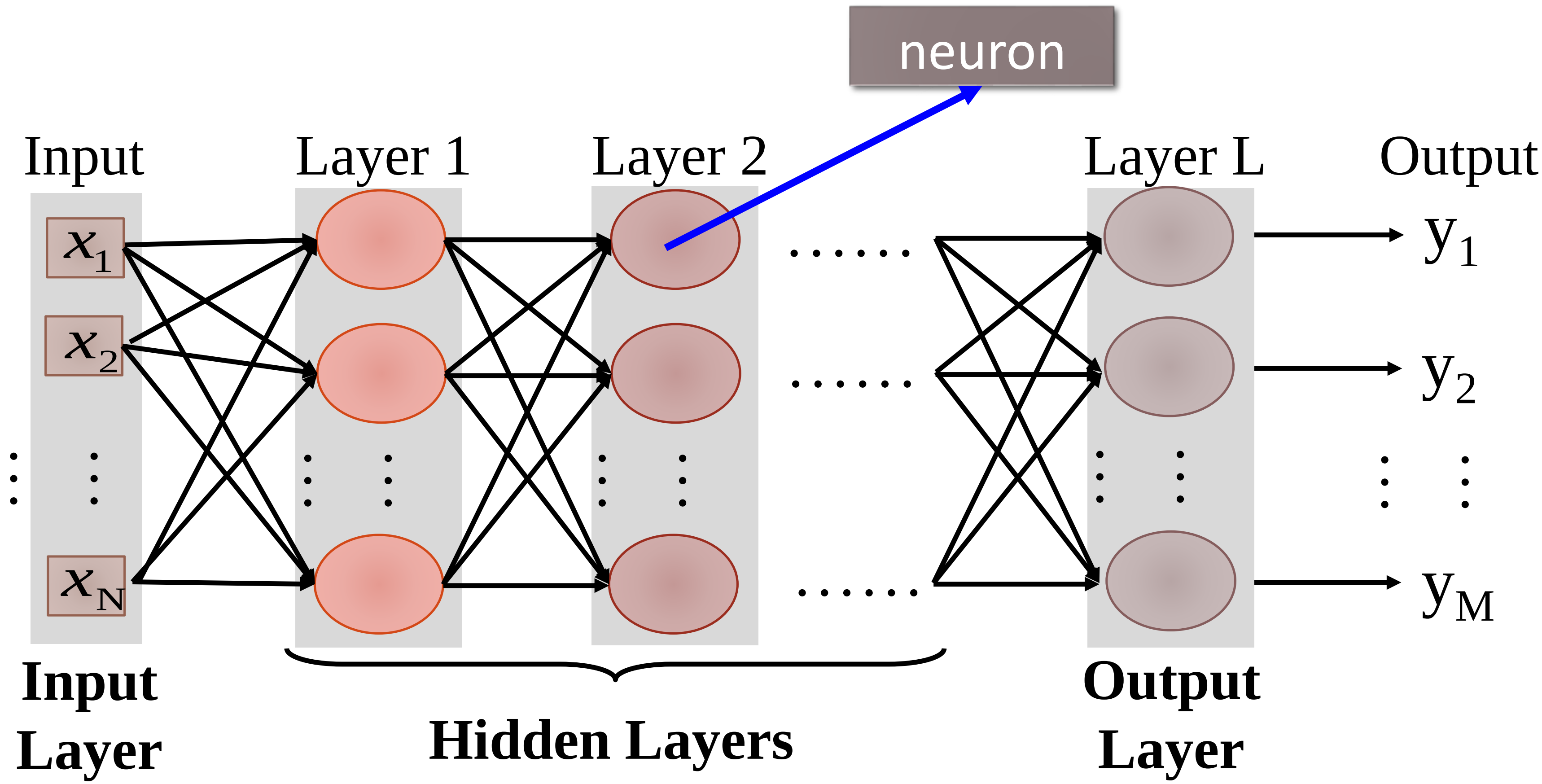
Hierarchical Learning

Inspired from visual information processing, a representation of Hierarchical Learning is developed, also know as **“Deep Learning”**



First in 1986 by **Rina Dechter**
Revolution **since 2012**

Deep Neural Network



Deep means many hidden layers

Why Deep Network?

Layer X Size	Word Error Rate (%)
1 X 2k	24.2
2 X 2k	20.4
3 X 2k	18.4
4 X 2k	17.8
5 X 2k	17.2
7 X 2k	17.1

Not surprised, more parameters, better performance

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

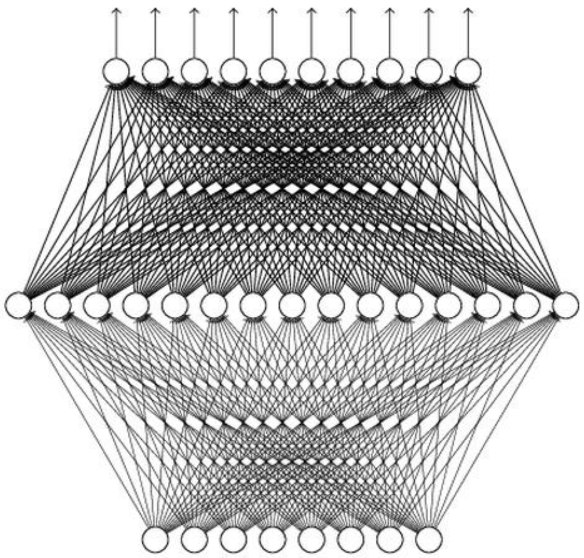
Why Deep Network?

- Universal Theorem

Any continuous function f

$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer

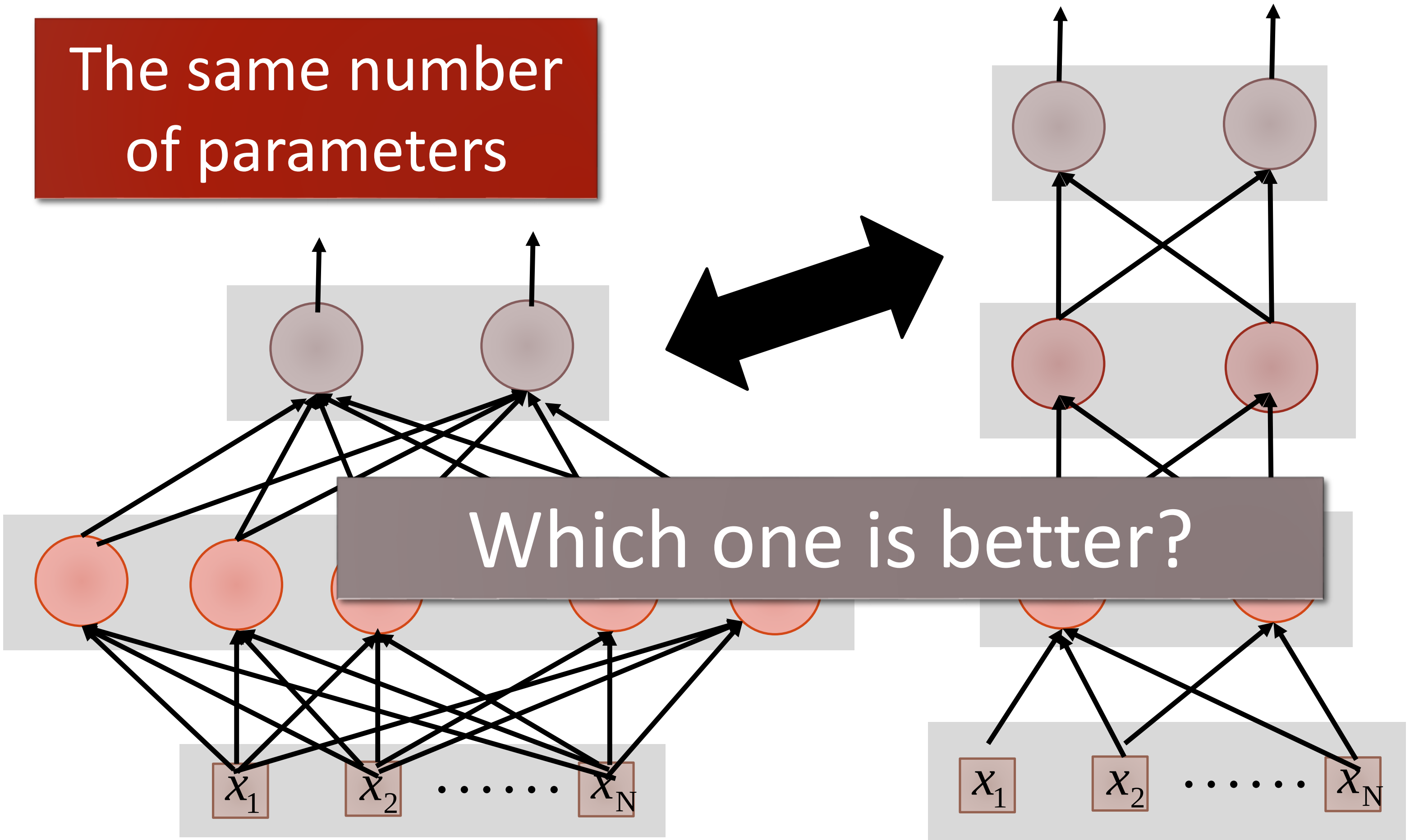


(given **enough** hidden neurons)

Why "Deep" neural network not "Fat" neural network?

Fat + Short v.s. Thin + Tall

The same number of parameters



Shallow

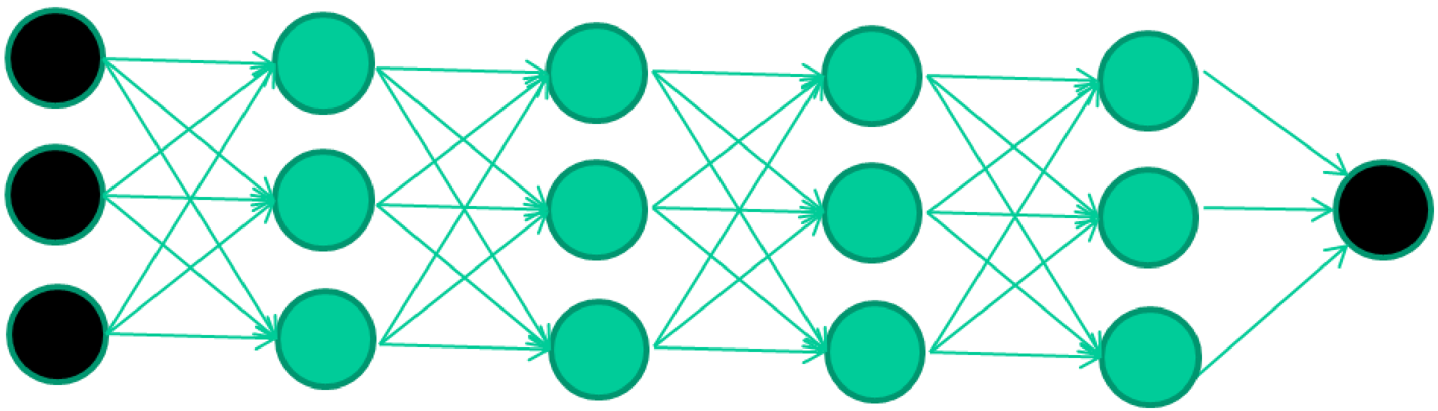
Deep

Fat + Short v.s. Thin + Tall

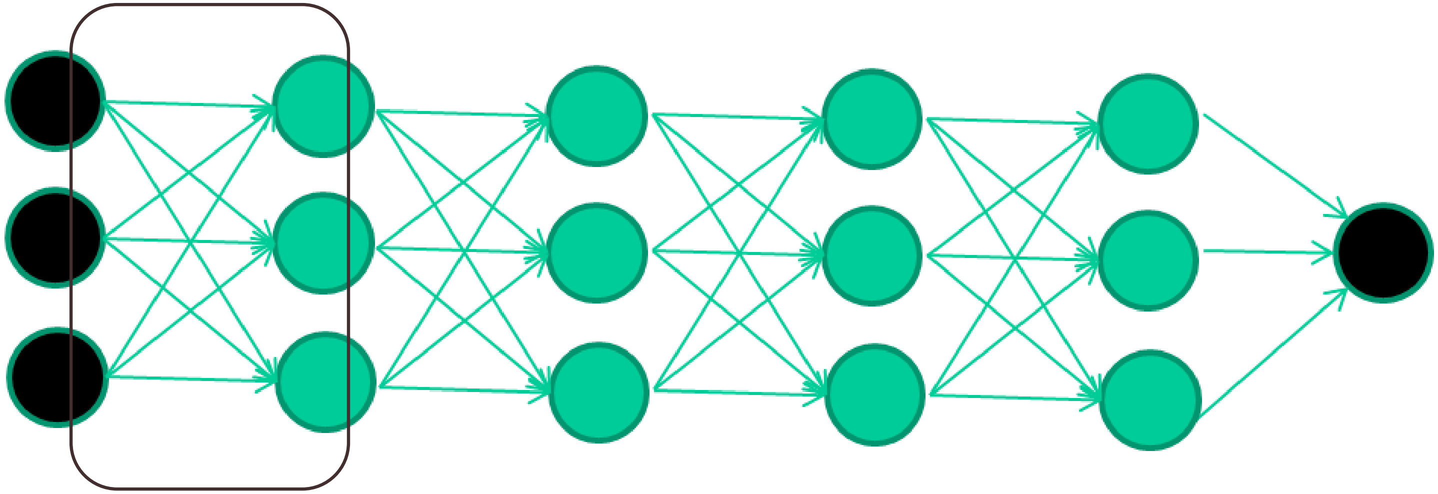
Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

Training multi-layer NNs (DNN)

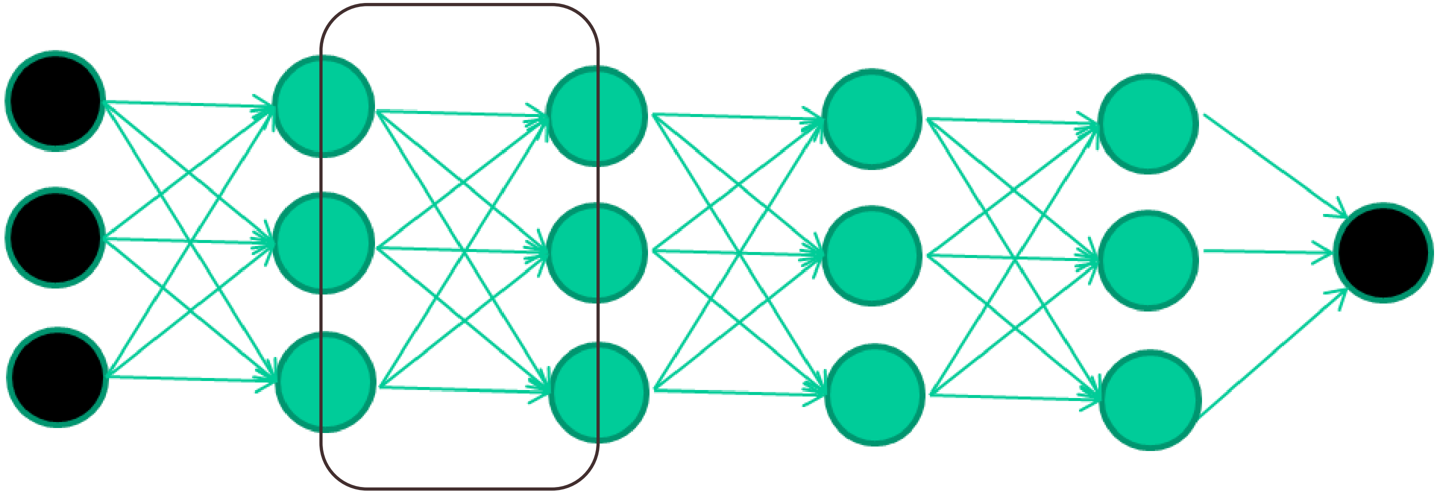


Training multi-layer NNs



Train **this** layer first

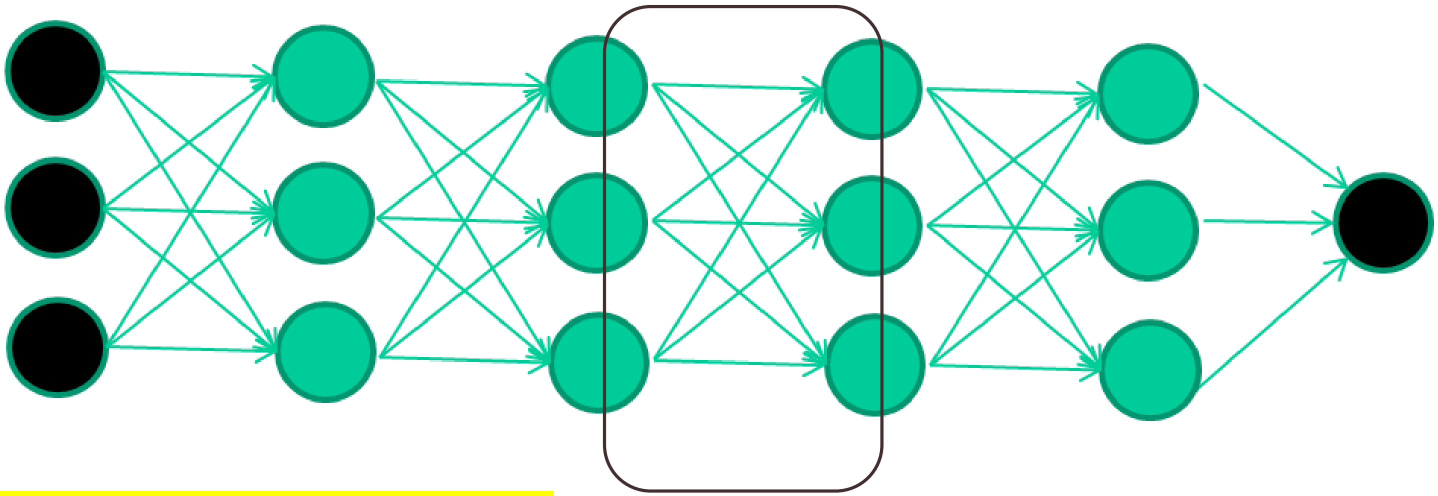
Training multi-layer NNs



Train **this** layer first

then **this** layer

Training multi-layer NNs

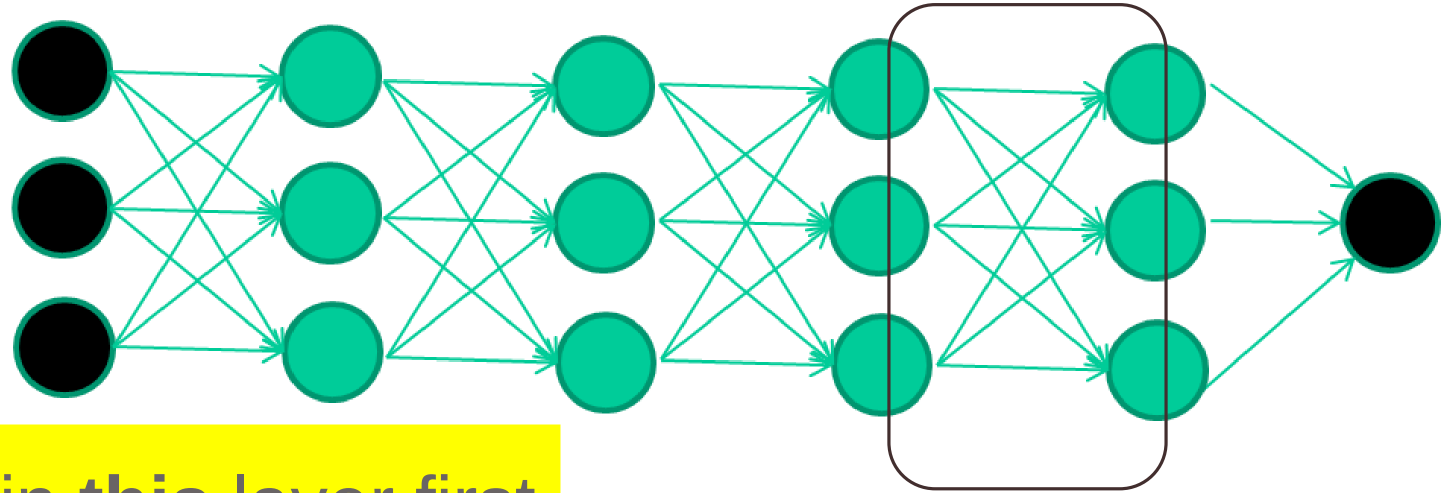


Train **this** layer first

then **this** layer

then **this** layer

Training multi-layer NNs



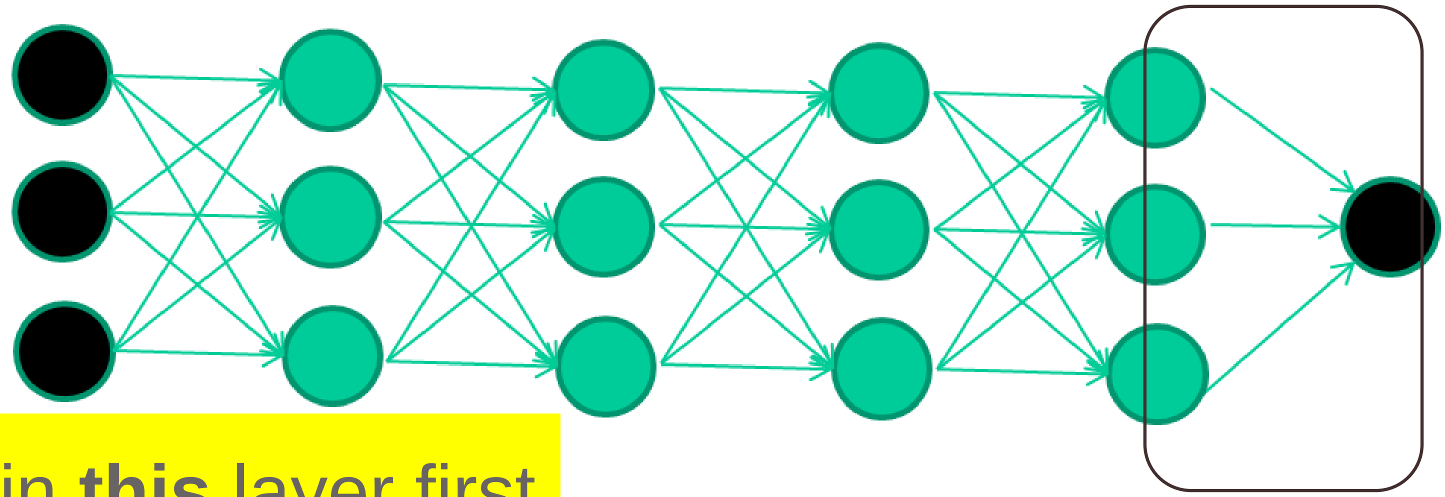
Train **this** layer first

then **this** layer

then **this** layer

then **this** layer

Training multi-layer NNs



Train **this** layer first

then **this** layer

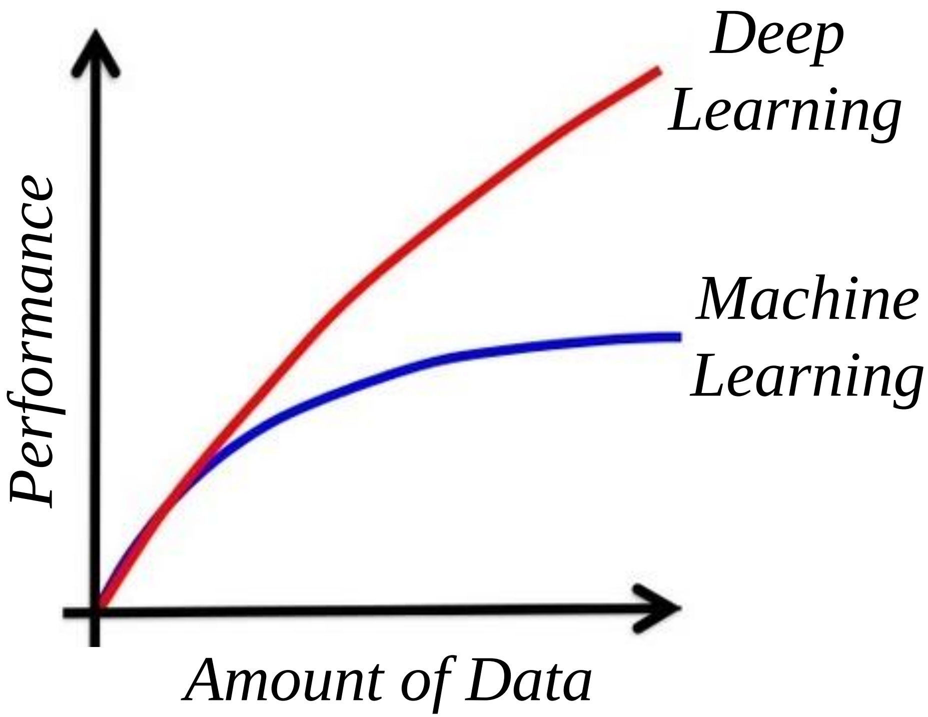
then **this** layer

then **this** layer

finally **this** layer

When to use Deep Learning?

- Data size is large
- High end infrastructure
- Lack of domain understanding
- Complex problem such as image classification, speech recognition etc.



Fuel of deep learning is the big data by Andrew Ng

Limitations of Deep Learning

- Very slow to train
- Models are very complex, with lot of parameters to optimize:
  - ✓ Initialization of weights
  - ✓ Layer-wise training algorithm
  - ✓ Neural architecture
    - Number of layers
    - Size of layers
    - Type – regular, pooling, max pooling, soft max
  - ✓ Fine-tuning of weights using back propagation

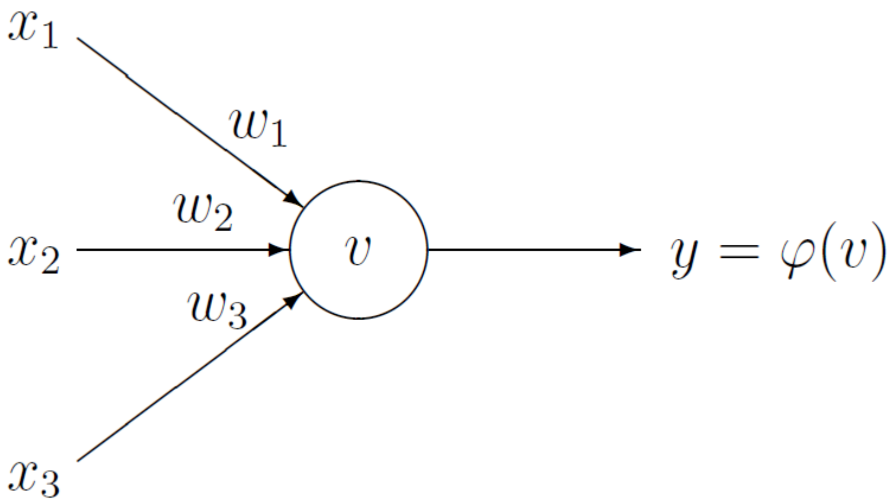
Thank you!

dinesh@dtu.ac.in

Problems on Neural Networks

Problem 1

- Consider a artificial Neurons, which has three inputs nodes x = (x1, x2, x3) that receive only binary signals (either 0 or 1). How many different input patterns this node can receive? What if the node had four inputs? Five? Can you give a formula that computes the number of binary input patterns for a given number of inputs?



Solutions

- There are three inputs, the number of combinations of 0 and 1 is 8.

x_1	0	1	0	1	0	1	0	1
x_2	0	0	1	1	0	0	1	1
x_3	0	0	0	0	1	1	1	1

- If there are four inputs nodes then number of combinations is 16.

x_1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
x_2	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
x_3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
x_4	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

- If there are n-input nodes then the number combinations will be 2^n.

Problem 2

- Consider a artificial neurons have three inputs, the weights corresponding to the these inputs have (2, -4, 1), the activation function is unit step. Determine the output for following input values.

Pattern	P_1	P_2	P_3	P_4
x_1	1	0	1	1
x_2	0	1	0	1
x_3	0	1	1	1

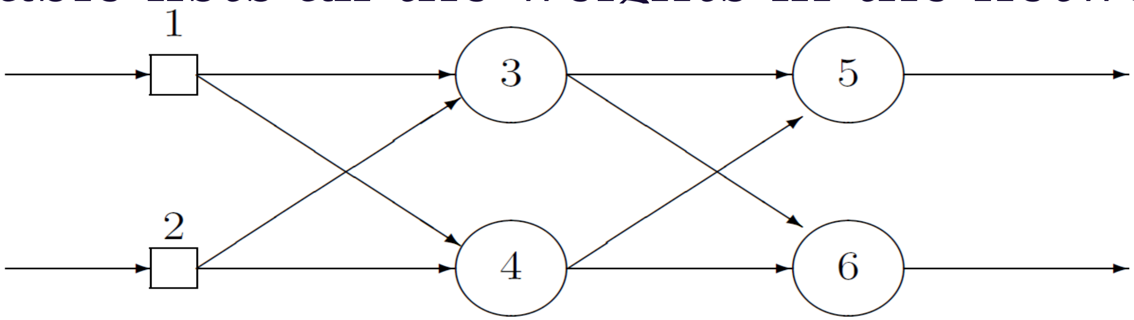
Solutions

- To find the output for each patterns
  - ✓ First calculate the weighted sum $\sum_i w_i x_i = w_1 \cdot x_1 + w_2 \cdot x_2 + w_3 \cdot x_3$
  - ✓ Apply the activation function i.e. unit step $\varphi(v) = \begin{cases} 1 & for \; v \geq 0 \\ 0 & otherwise \end{cases}$
  - ✓ The calculations for each input pattern are:

$$
\begin{aligned}
P_1: &\quad v = 2\cdot 1 - 4\cdot 0 + 1\cdot 0 = 2\,, &\quad (2 > 0)\,, &\quad y = \varphi(2) = 1 \\
P_2: &\quad v = 2\cdot 0 - 4\cdot 1 + 1\cdot 1 = -3\,, &\quad (-3 < 0)\,, &\quad y = \varphi(-3) = 0 \\
P_3: &\quad v = 2\cdot 1 - 4\cdot 0 + 1\cdot 1 = 3\,, &\quad (3 > 0)\,, &\quad y = \varphi(3) = 1 \\
P_4: &\quad v = 2\cdot 1 - 4\cdot 1 + 1\cdot 1 = -1\,, &\quad (-1 < 0)\,, &\quad y = \varphi(-1) = 0
\end{aligned}
$$

Problem 3

- Consider a feed forward neural network with one hidden layer. A weight on connection between nodes i and j is denoted by w_{ij} , such as w_{13} is the weight on the connection between nodes 1 and 3. The following table lists all the weights in the network.



$w_{13} = -2$	$w_{35} = 1$
$w_{23} = 3$	$w_{45} = -1$
$w_{14} = 4$	$w_{36} = -1$
$w_{24} = -1$	$w_{46} = 1$

- Node 3, 4, 5 and 6 uses unit step activation function. Compute the output of the n/w for following inputs.

Pattern:	P_1	P_2	P_3	P_4
Node 1:	0	1	0	1
Node 2:	0	0	1	1

Solutions

- In order to find the output of the network it is necessary to calculate weighted sums of hidden nodes 3 and 4:

$$v_3 = w_{13}x_1 + w_{23}x_2 \ , \quad v_4 = w_{14}x_1 + w_{24}x_2$$

- Then find the outputs from hidden nodes using activation function. $y_3 = \varphi(v_3) \ , \quad y_4 = \varphi(v_4)$
- Use the outputs of the hidden nodes y3 and y4 as the input values to the output layer (nodes 5 and 6), and find weighted sums of output nodes 5 and 6:

$$v_5 = w_{35}y_3 + w_{45}y_4 \ , \quad v_6 = w_{36}y_3 + w_{46}y_4$$

- Finally, compute the outputs from nodes 5 and 6 using

$$y_5 = \varphi(v_5) \ , \quad y_6 = \varphi(v_6)$$

Solutions

P_1: *Input pattern* $(0, 0)$

$$\begin{aligned} v_3 &= -2 \cdot 0 + 3 \cdot 0 = 0, & y_3 &= \varphi(0) = 1 \\ v_4 &= 4 \cdot 0 - 1 \cdot 0 = 0, & y_4 &= \varphi(0) = 1 \\ v_5 &= 1 \cdot 1 - 1 \cdot 1 = 0, & y_5 &= \varphi(0) = 1 \\ v_6 &= -1 \cdot 1 + 1 \cdot 1 = 0, & y_6 &= \varphi(0) = 1 \end{aligned}$$

The output of the network is $(1, 1)$.

P_2: *Input pattern* $(1, 0)$

$$\begin{aligned} v_3 &= -2 \cdot 1 + 3 \cdot 0 = -2, & y_3 &= \varphi(-2) = 0 \\ v_4 &= 4 \cdot 1 - 1 \cdot 0 = 4, & y_4 &= \varphi(4) = 1 \\ v_5 &= 1 \cdot 0 - 1 \cdot 1 = -1, & y_5 &= \varphi(-1) = 0 \\ v_6 &= -1 \cdot 0 + 1 \cdot 1 = 1, & y_6 &= \varphi(1) = 1 \end{aligned}$$

The output of the network is $(0, 1)$.

Solutions

P_3: *Input pattern* $(0, 1)$

$$\begin{aligned} v_3 &= -2 \cdot 0 + 3 \cdot 1 = 3, & y_3 &= \varphi(3) = 1 \\ v_4 &= 4 \cdot 0 - 1 \cdot 1 = -1, & y_4 &= \varphi(-1) = 0 \\ v_5 &= 1 \cdot 1 - 1 \cdot 0 = 1, & y_5 &= \varphi(1) = 1 \\ v_6 &= -1 \cdot 1 + 1 \cdot 0 = -1, & y_6 &= \varphi(-1) = 0 \end{aligned}$$

The output of the network is $(1, 0)$.

P_4: *Input pattern* $(1, 1)$

$$\begin{aligned} v_3 &= -2 \cdot 1 + 3 \cdot 1 = 1, & y_3 &= \varphi(1) = 1 \\ v_4 &= 4 \cdot 1 - 1 \cdot 1 = 3, & y_4 &= \varphi(3) = 1 \\ v_5 &= 1 \cdot 1 - 1 \cdot 1 = 0, & y_5 &= \varphi(0) = 1 \\ v_6 &= -1 \cdot 1 + 1 \cdot 1 = 0, & y_6 &= \varphi(0) = 1 \end{aligned}$$

The output of the network is $(1, 1)$.