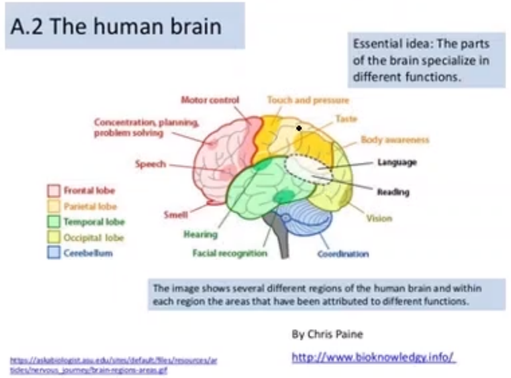
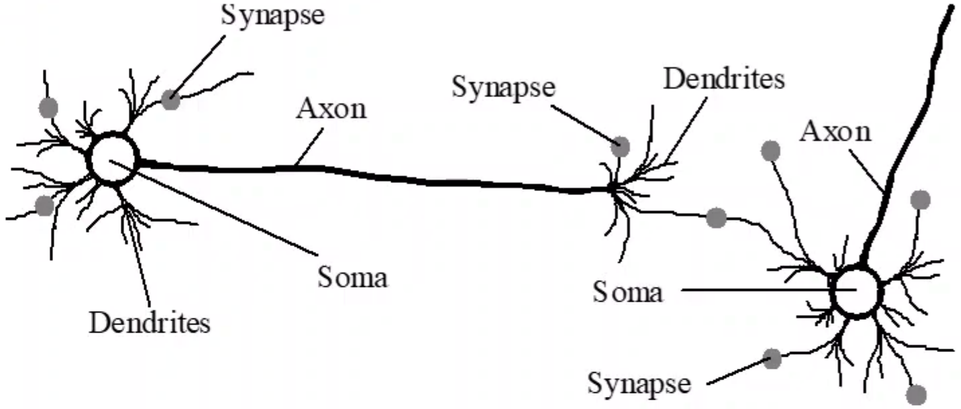
Artificial Neural Network

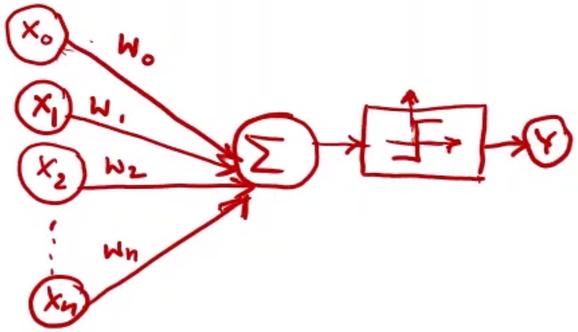
- Inspiration from how our brain works
- Neuron as a computing element
- Perceptron, the historical neuron
- Multilayer neural network
- How to accelerate learning in a multilayer neural network
- The Hopfield Network .
- Bidirectional associative memories (BAM)
- Self organizing neural networks

iration from how are brain works





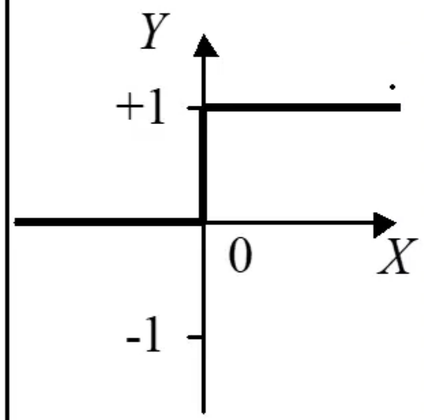
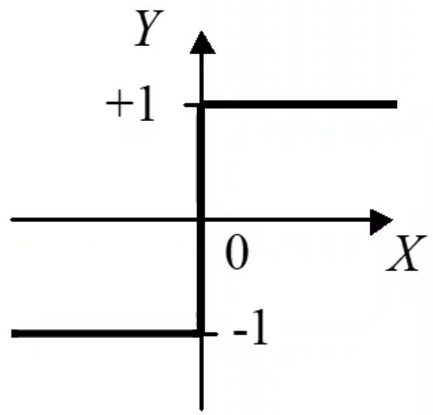
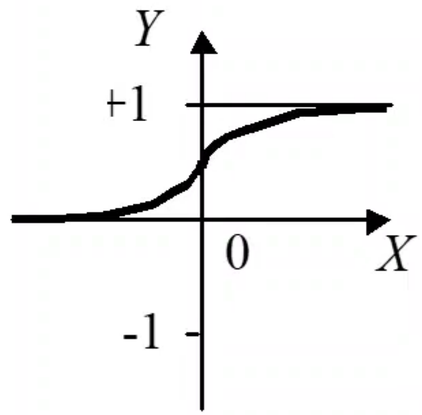
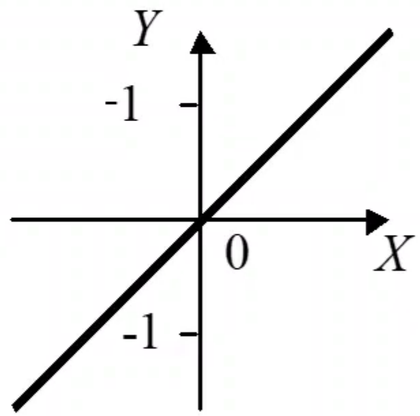
Biological Neural Network	*Artificial Neural Network*
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight



Timeline of Artificial Neural Network

- In 1943 Warren McCulloch and Walter Pitts proposed a very simple idea that is still the basis for most artificial neural networks.
- In 1958, Frank Rosenblatt introduced a training algorithm known as perceptron learning algorithm which could classify the linearly separable tasks:
  - **By making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron.**
  - **By selecting weights randomly , usually in the range [–0.5, 0.5], and then updating to obtain the output consistent with the training examples.**
  - **If the error, e(p), is positive, we need to increase perceptron output Y(p), but if it is negative, we need to decrease Y(p).**

Activation functions of a neuron

Step function	*Sign function*	*Sigmoid function*	*Linear function*
			
$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{if } X < 0 \end{cases}$	$Y^{sign} = \begin{cases} +1, \text{if } X \geq 0 \\ -1, \text{if } X < 0 \end{cases}$	$Y^{sigmoid} = \dfrac{1}{1+e^{-X}}$	$Y^{linear} = X$

Perceptron learning algorithm

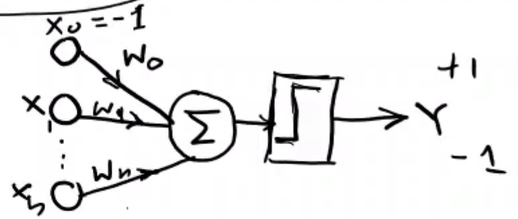
It is based on perceptron learning rule: $W_i(P+1) = W_i(P) + \alpha X_i (Y_d(P) - Y(P))$

learning rate: 0.01, 0.1

$= W_i(P) + \Delta W_i(P)$

$\Delta W_i(P) = \alpha . X_i(P) . e(P)$

$e(P) =$.



$e_p = Y_d(P) - Y(P)$

steps: 1. Initialisation: Select initial weights W_i

2. Activation: $Y(P) = \text{step}\left[\sum_{i=1}^{n} X_i(P) . W_i(P) - W_0(P)\right]$

here, n = number of perceptron inputs.

3. Weight training: $W_i(P+1) = W_i(P) + \Delta W_i(P)$

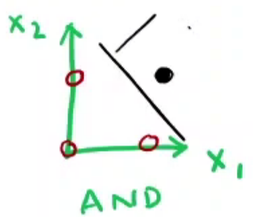
$\Delta W_i(P) = \alpha X_i(P) . e(P)$

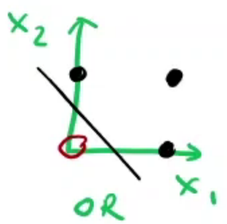
4. Iteration: increase P by 1, go back to step-2 and repeat the process until convergence.

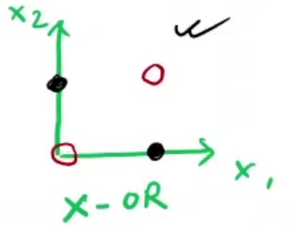
Example problems:

		AND	OR	Exclusive-OR
X_1	X_2	$X_1 \cap X_2$	$X_1 \cup X_2$	$X_1 \oplus X_2$
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

$\Sigma + i W_i - W_0 = 0$



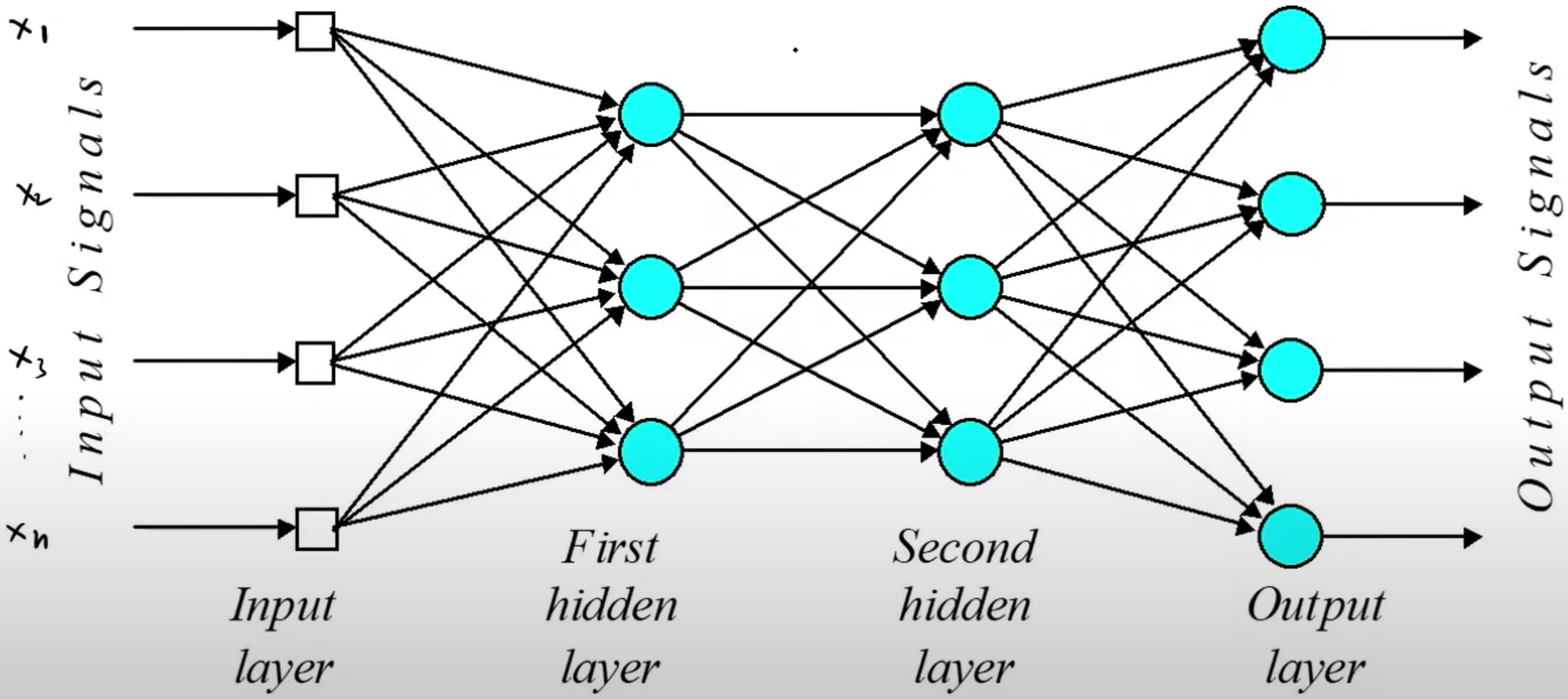




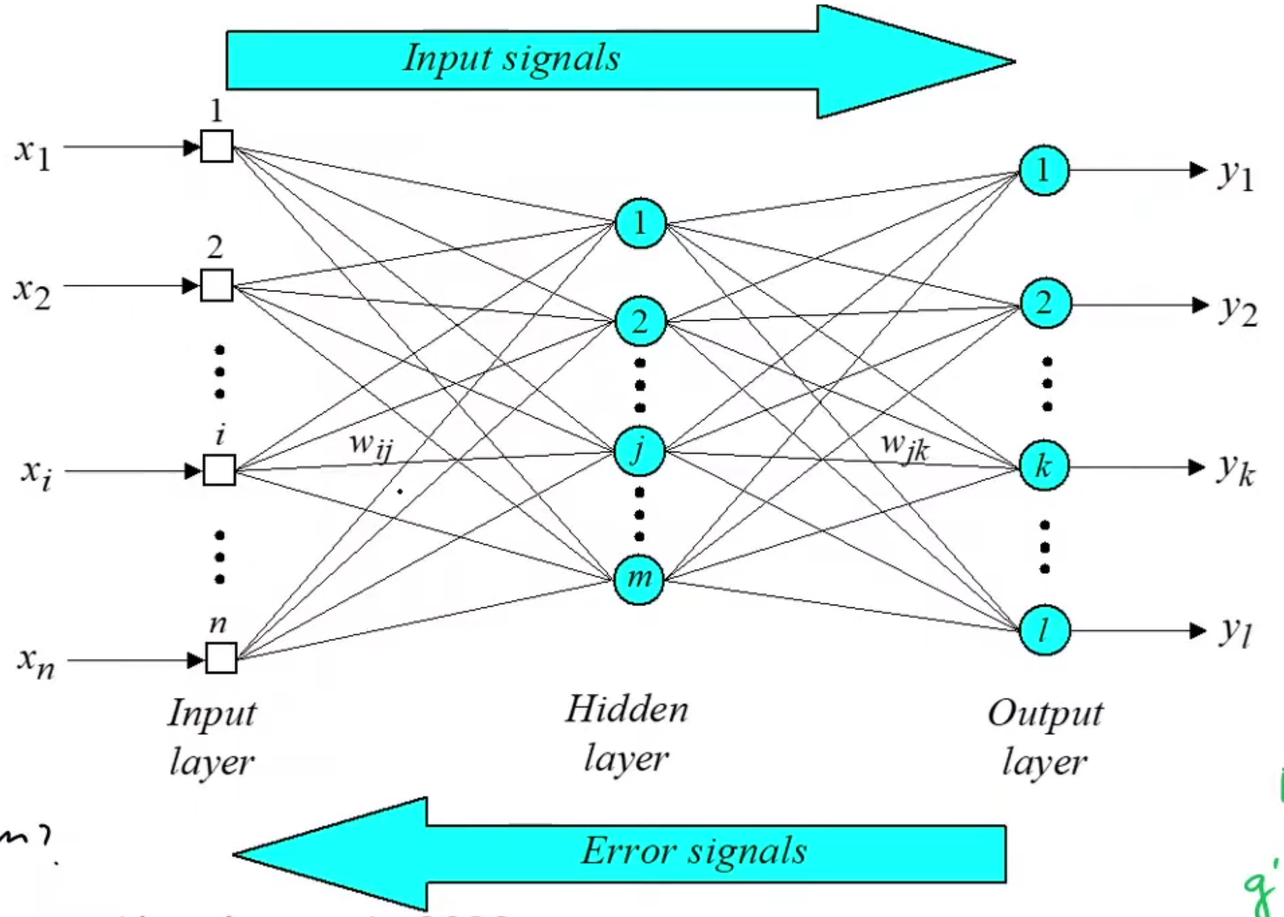
AND
NAND
NOR
OR

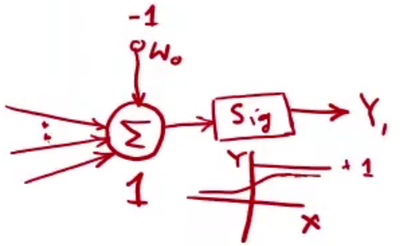
Back -propagation Algorithm

Architecture of a typical fully connected Neural Network:



What is error gradient & why do we need this?





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

$$g'(x_k^{(p)}) = \frac{d}{dx_k^{(p)}}\left(\frac{1}{1+e^{-x_k^{(p)}}}\right)$$

$$g'(z) = \frac{d}{dz}\frac{1}{1+e^{-z}} = \frac{e^{-z}}{(1+e^{-z})^2}$$

$$g'(z) = \frac{e^{-z}}{(1+e^{-z})} \cdot \frac{1}{(1+e^{-z})}$$

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

put $Y_k^{(p)} = \frac{1}{1+e^{-z}}$

$$g'(x_k^{(p)}) = Y_k(p)(1 - Y_k(p))$$

Error gradient $= \delta_k(p) = \frac{\partial Y_k(p)}{\partial (x_k^{(p)})} E_k(p)$

$Y_k(p) * (1 - Y_k(p) * E_k(p)$

Who invented Back Propagation Algorithm ?

1970 by Finnish master student Seppo Linnainmaa. In 2020, we are celebrating BP's half-century anniversary!

Back -propagation Algorithm

Weight corrections in the output layer:

$$\Delta W_{jk}(p) = \alpha \cdot Y_j(p) \cdot \delta_k(p)$$

learning rate α = 0.01

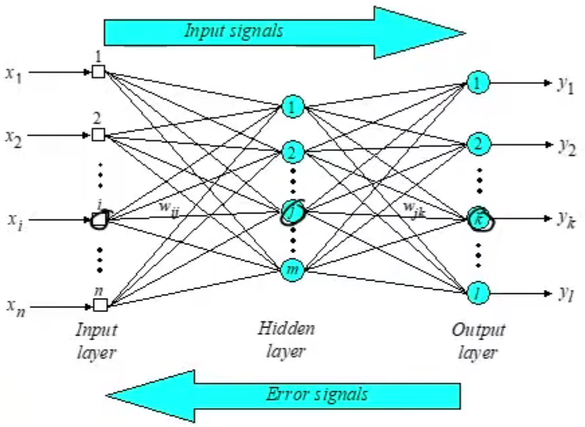
$$= \alpha \cdot Y_j(p) \cdot Y_k(p)\,(1 - Y_k(p)) \cdot E_k(p)$$

where, $Y_j(p) = \dfrac{1}{1 + e^{-X_j(p)}}$; $X_j(p) = \sum_{i=1}^{n} x_i(p)\, w_{ij} - w_{0j}$

Weight corrections in the hidden layer:

$$\Delta W_{ij} = \alpha \cdot x_i(p) \cdot \delta_j(p)$$

$$= \alpha \cdot x_i(p) \cdot Y_j(p)\,(1 - Y_j(p)) * \sum_{k=1}^{\ell} \delta_k(p) \cdot W_{jk}(p)$$



Back -propagation Algorithm

Step-1, Initializations:

- Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range.

Step-2, Activation:

- Activate the back-propagation neural network by applying inputs $x_1(p), x_2(p),\ldots, x_n(p)$ and desired outputs $y_{d,1}(p), y_{d,2}(p),\ldots, y_{d,n}(p)$.

(a) Calculate the actual outputs of the neurons in the hidden layer:

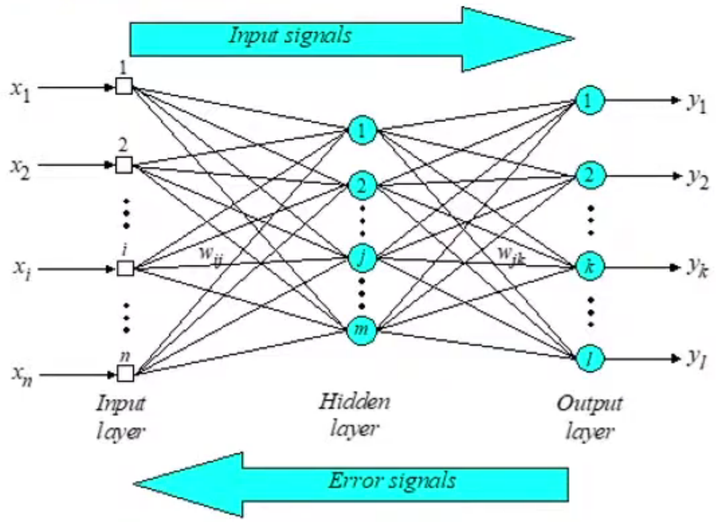
$$Y_j = \text{sigmoid}\left[\sum_{i=1}^{n} X_i(p) * W_{ij}(p) - W_{oj}\right],$$

where n is the number of inputs of neuron j in the hidden layer, and *sigmoid* is the *sigmoid* activation function

(b) Calculate the actual outputs of the neurons in the output layer:

$$Y_k(p) = \text{sigmoid}\left[\sum_{j=1}^{m} X_{jk}(p) * W_{jk}(p) - W_{ok}\right]$$

where m is the number of inputs of neuron k in the output layer.



Back-propagation Algorithm

Step-3, Weight Training

- Update the weights in the back-propagation network propagating backward the errors associated with output neurons.

(*a*) Calculate the error gradient for the neurons in the output layer:

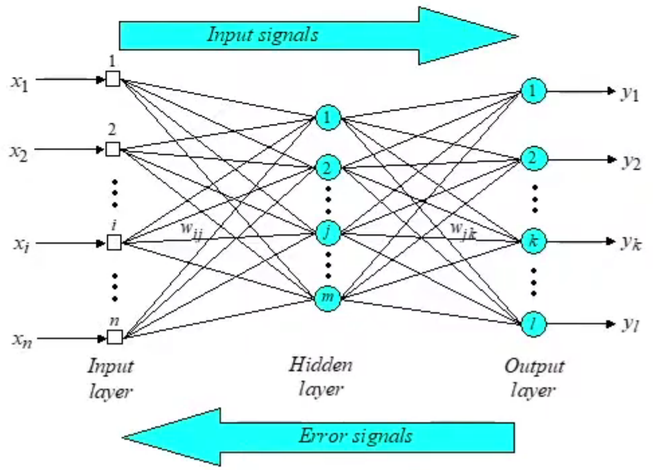
$$\delta_k = Y_k(P) * [1 - Y_k(P)] * E_k(P) \quad \text{where} \quad E_k(P) = Y_{d,k}(P) - Y_k(P)$$

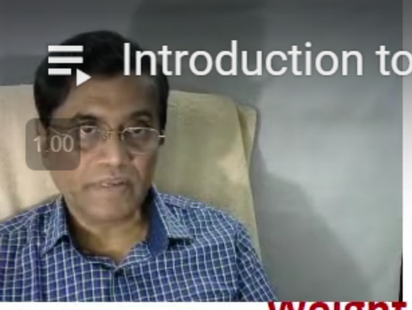
Calculate the weight corrections:

$$\Delta W_{jk}(P) = \alpha * Y_j(P) * \delta_k(P)$$

Update the weights at the output neurons:

$$W_{jk}(P+1) = W_{jk}(P) + \Delta W_{jk}(P)$$





Back-propagation Algorithm

Weight training contd.

(b) Calculate the error gradient for the neurons in the hidden layer:

$$\delta_j(p) = Y_j(p) * [1 - Y_j(p)] * \sum_{k=1}^{l} \delta_k(p) * w_{jk}(p)$$

Calculate the weight corrections:

$$\Delta w_{ij}(p) = \alpha * x_j(p) * \delta_j(p)$$

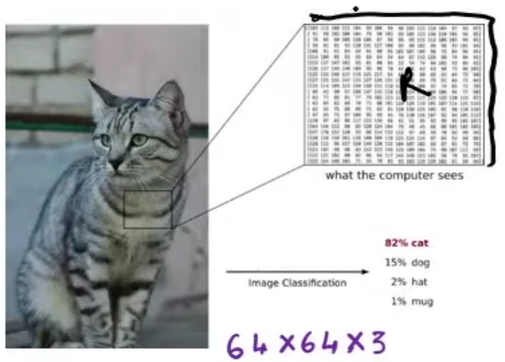
update the weights at the hidden neurons:

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

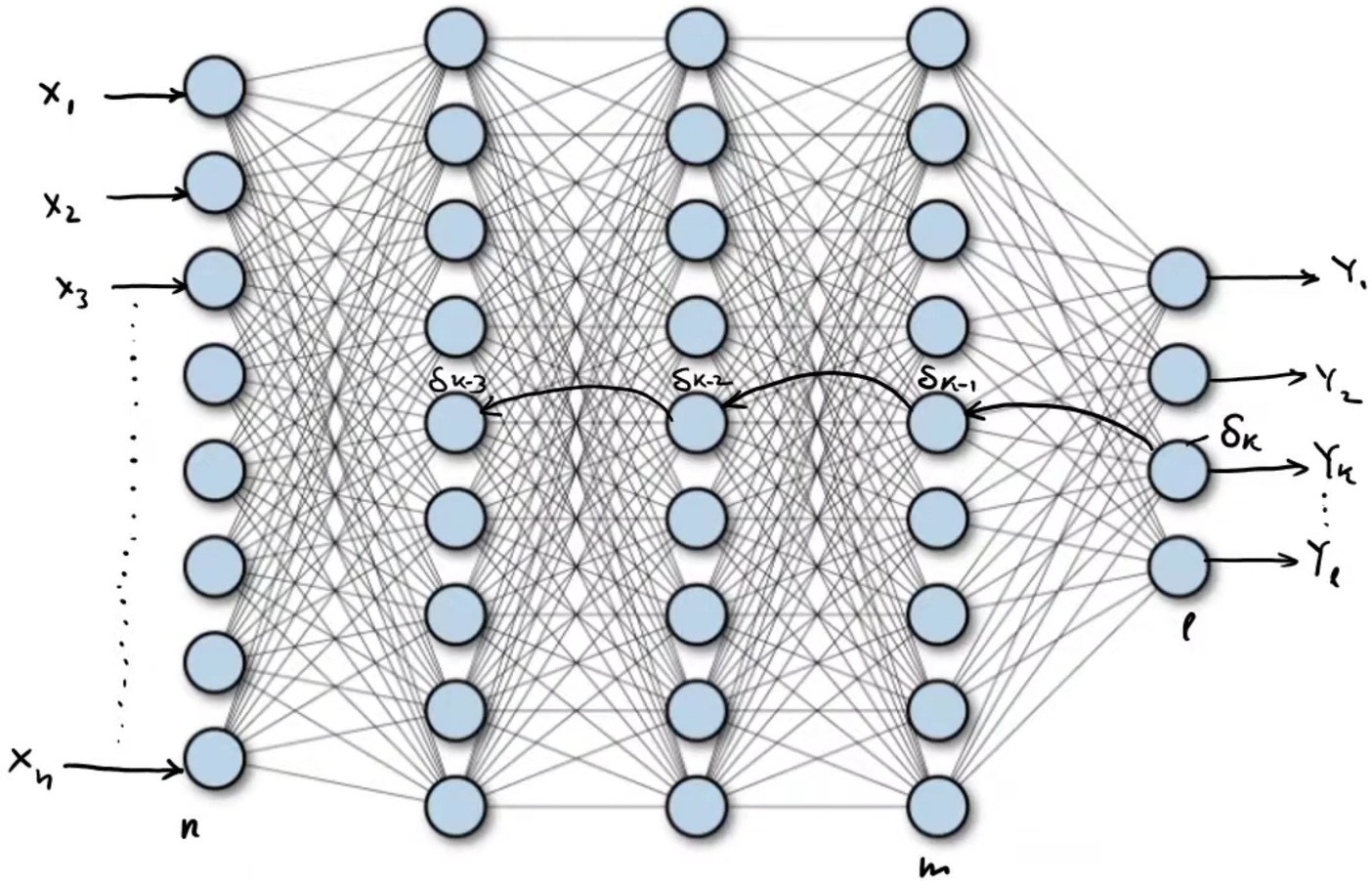
Step-4, Iteration

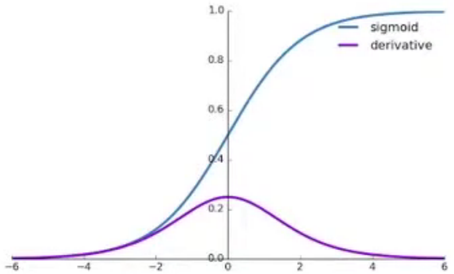
Increase iteration p by 1 and go back to Step-2 and repeat the process until the selected error criterion is satisfied

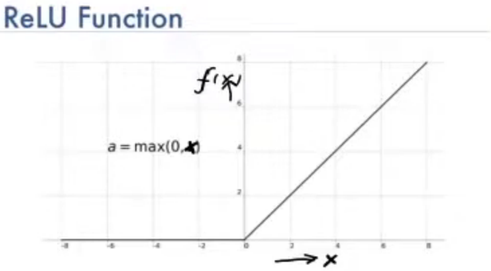
lly connected /dense neural network



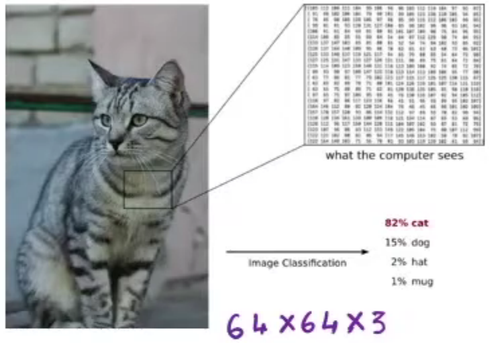
64 x 64 x 3
= 12288



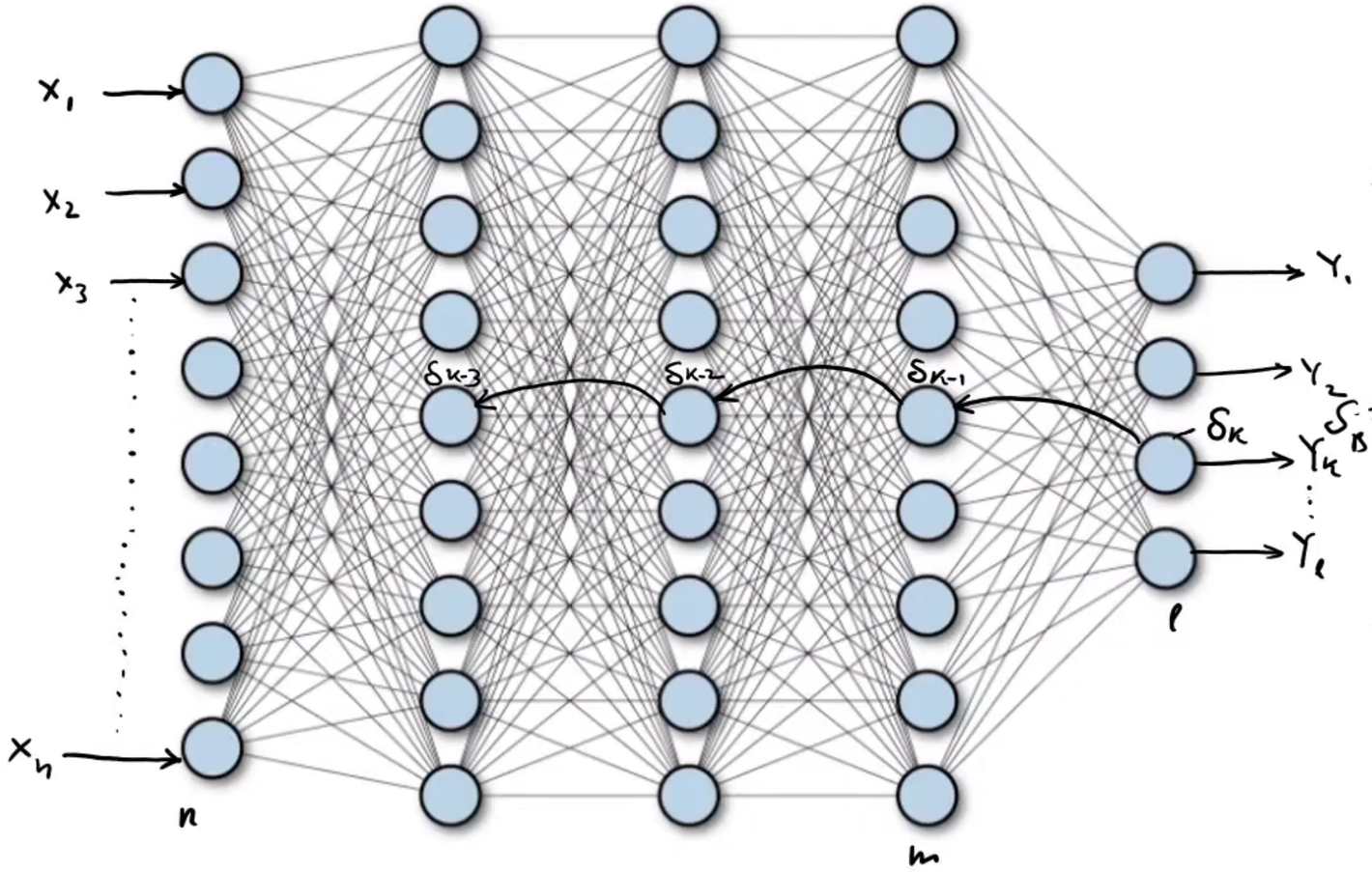


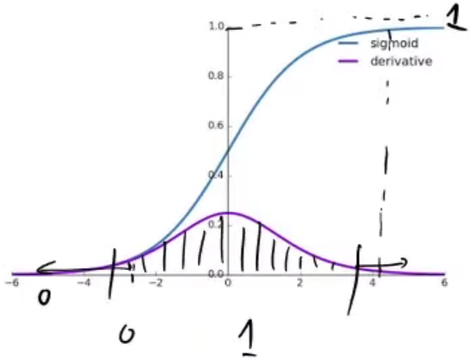


lly connected /dense neural network



64 x 64 x 3
= 12288





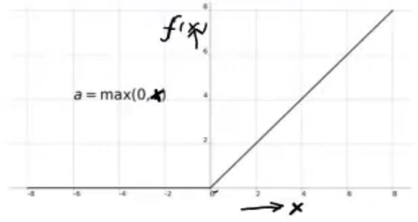
$\delta_k = \delta_{k-3} \cdot \delta_{k-2} \cdot \delta_{k-1} \cdot \delta_k$

$0.2 \times 0 \cdot 1 \times 0 \cdot 3 \times 0.5$

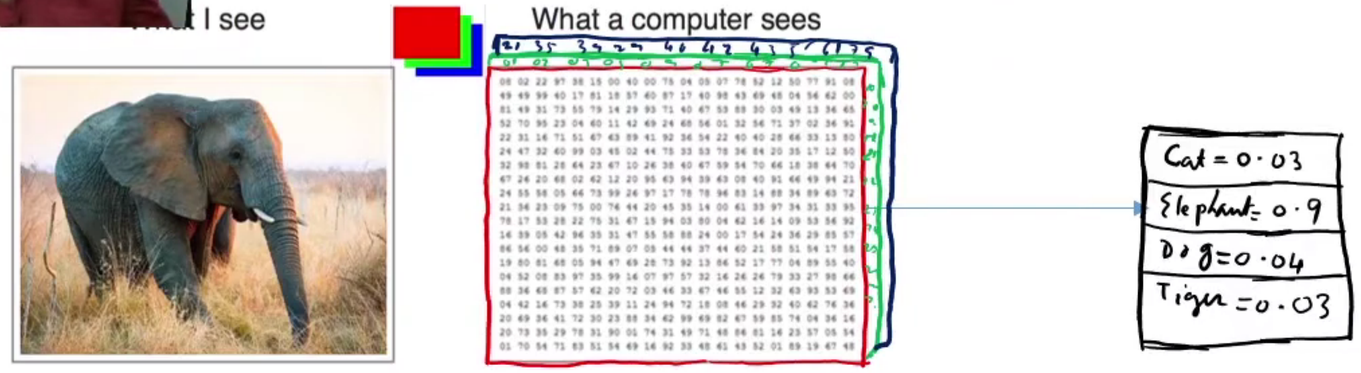
$= 0.000\,1$

Vanishing gradient Prob

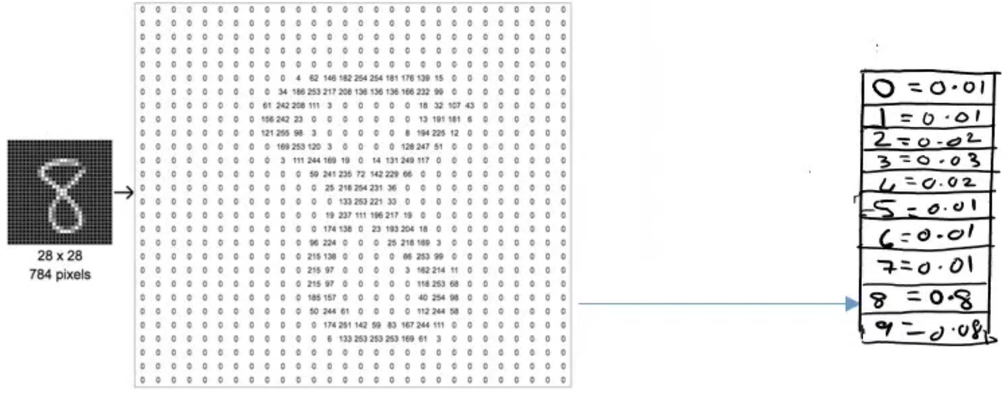
ReLU Function



What is CNN and why CNN?



- Yan Le Cun in 1980s gave the idea of CNN and deep learning.
- In 2012 Alexnet showed that time is ripe to revisit deep learning.



Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are:

$$\text{Softmax}(Y^{(i)}) = \frac{e^{Y^{(i)}}}{\sum_j e^{Y_j}}$$

Yann LeCun, one of the three researchers behind the creation of MNIST. Later on Facebook AI research upgraded it to QMNIST.

softmax is very useful function because it converts the scores to a normalized probability distribution, which can be displayed to a user

Why Deep Learning?

- Self driving car
- News Aggregation and Fraud News Detection
- Natural Language Processing including machine Translation
- Virtual Assistants
- Entertainment
- Financial Fraud Detection
- Healthcare

Why Deep Learning?

- Personalisation
- Detecting Developmental Delay in Children
- Colourisation of Black and White images
- Adding sounds to silent movies
- Demographic and Election Predictions
- Weather prediction
- Deep Dreaming