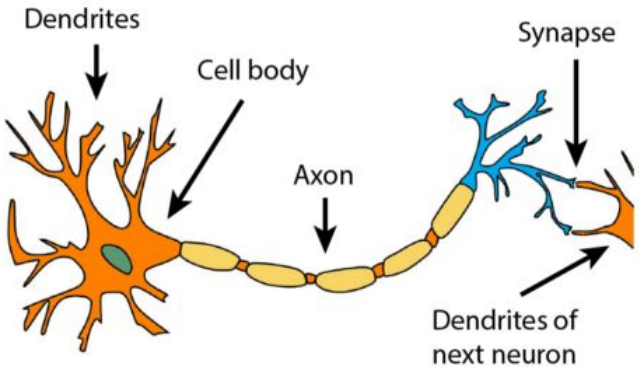
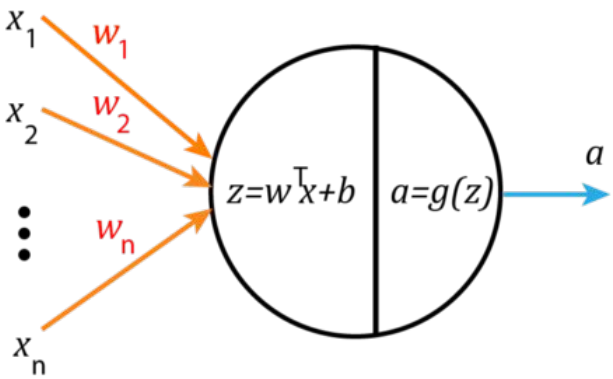
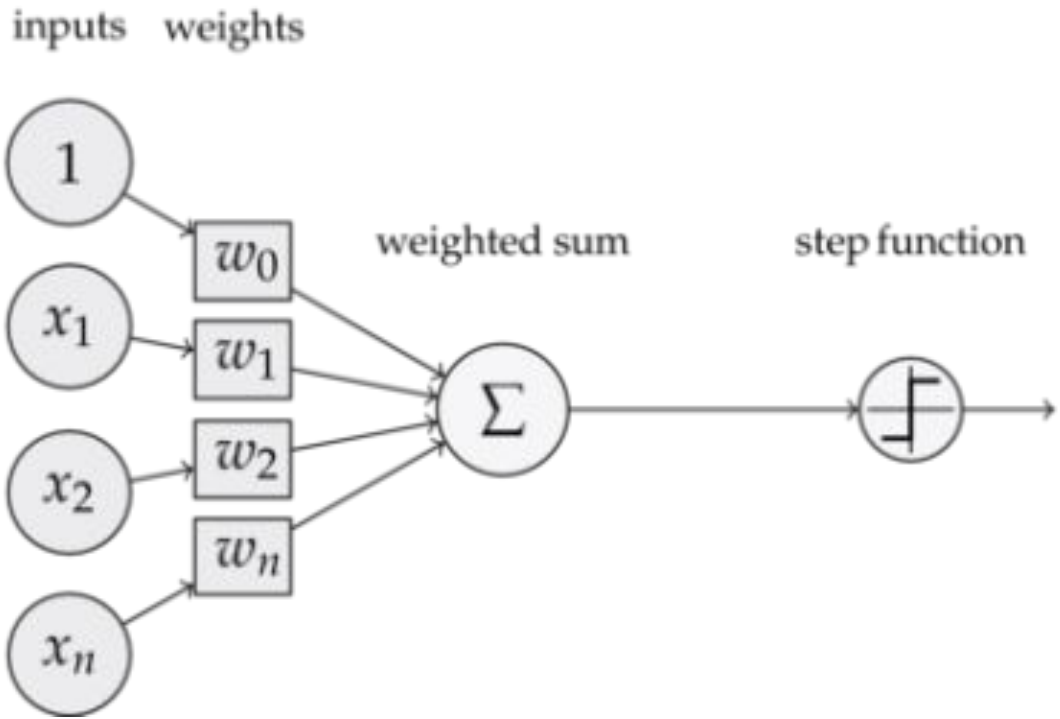
Neural Networks

ANN


Dendrites
Cell body
Axon
Synapse
Dendrites of
next neuron


x_1
x_2
x_n
w_1
w_2
w_n
$z=w^T x+b$
$a=g(z)$
a

Perceptron


inputs
weights
1
x_1
x_2
x_n
w_0
w_1
w_2
w_n
weighted sum
Σ
step function

Perceptron - Pseudo Código

Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while !*convergence* **do**
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
    end
end
//the algorithm converges when all the inputs are classified correctly

Perceptron - Pseudo Código

$$w_i \leftarrow w_i + \Delta w_i$$

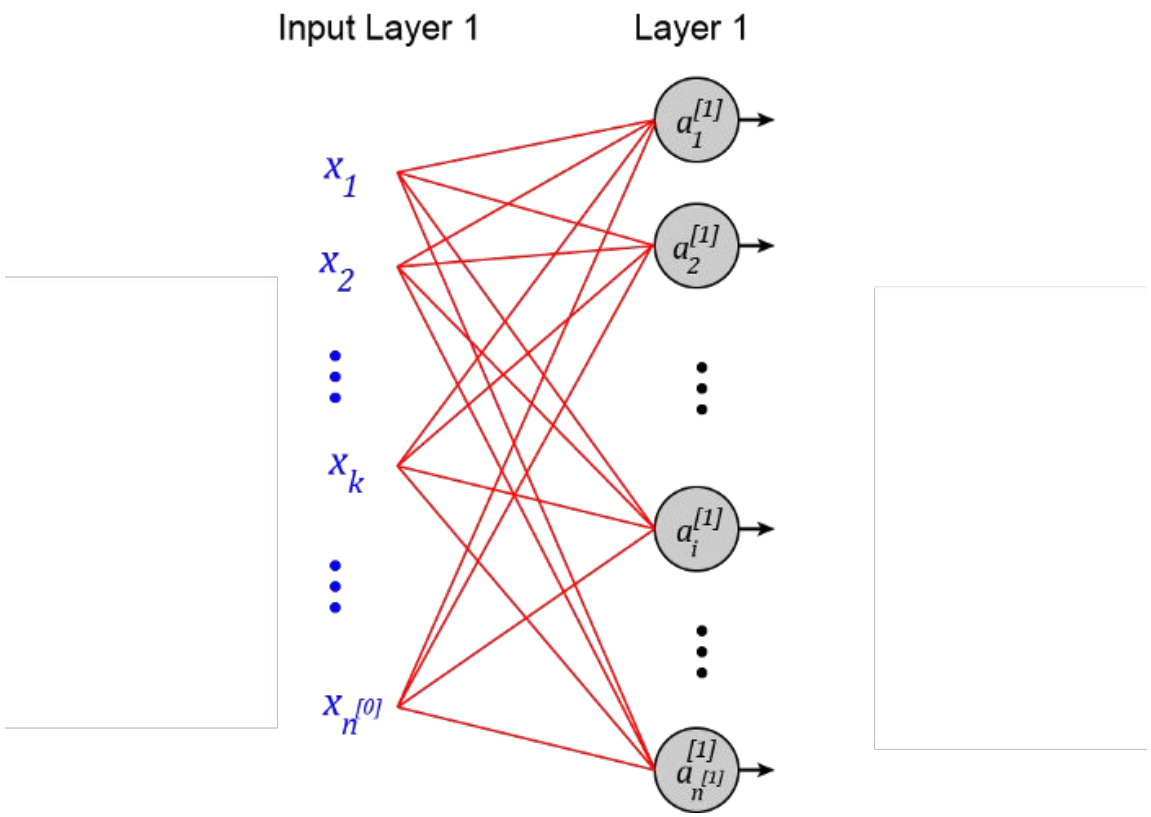
where

$$\Delta w_i = \eta(t - o)x_i$$

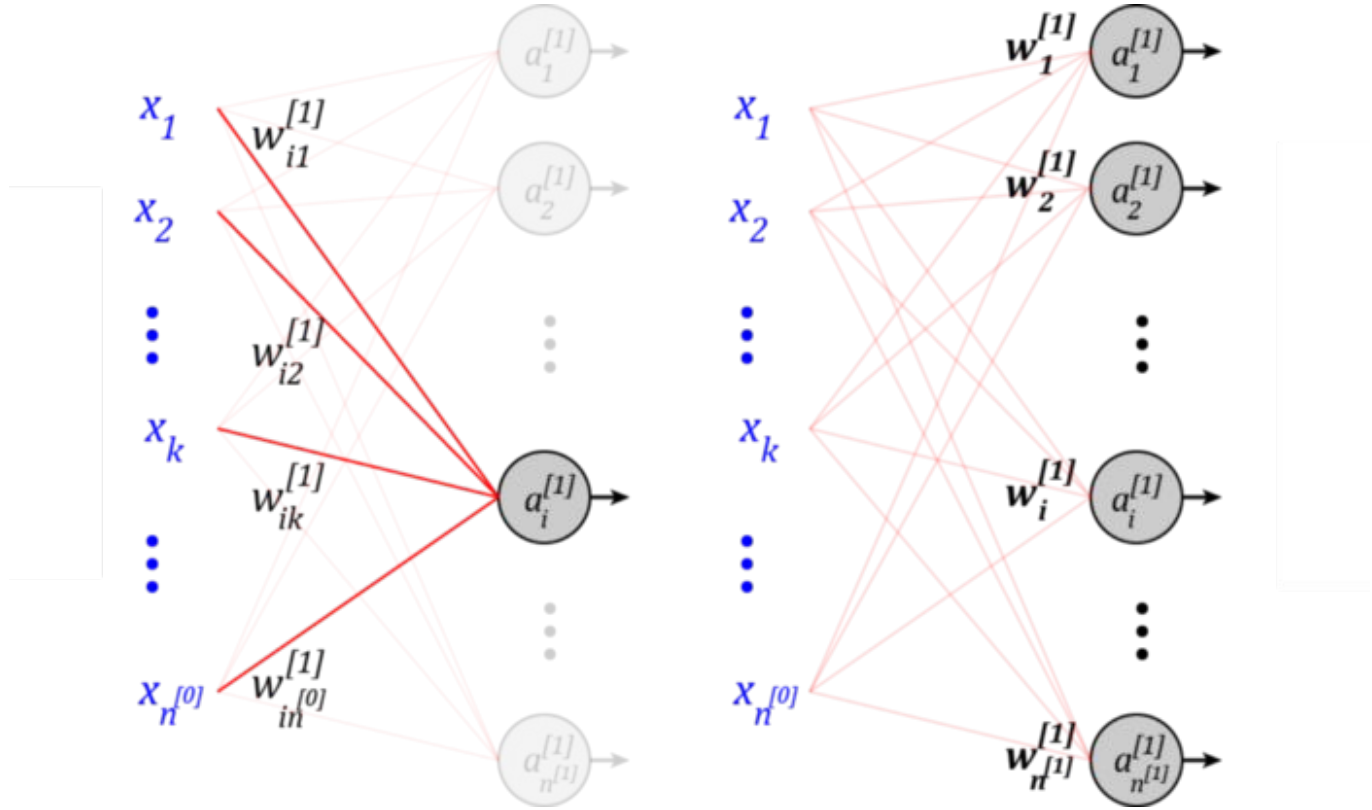
Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., 0.1) called *learning rate*

ANN



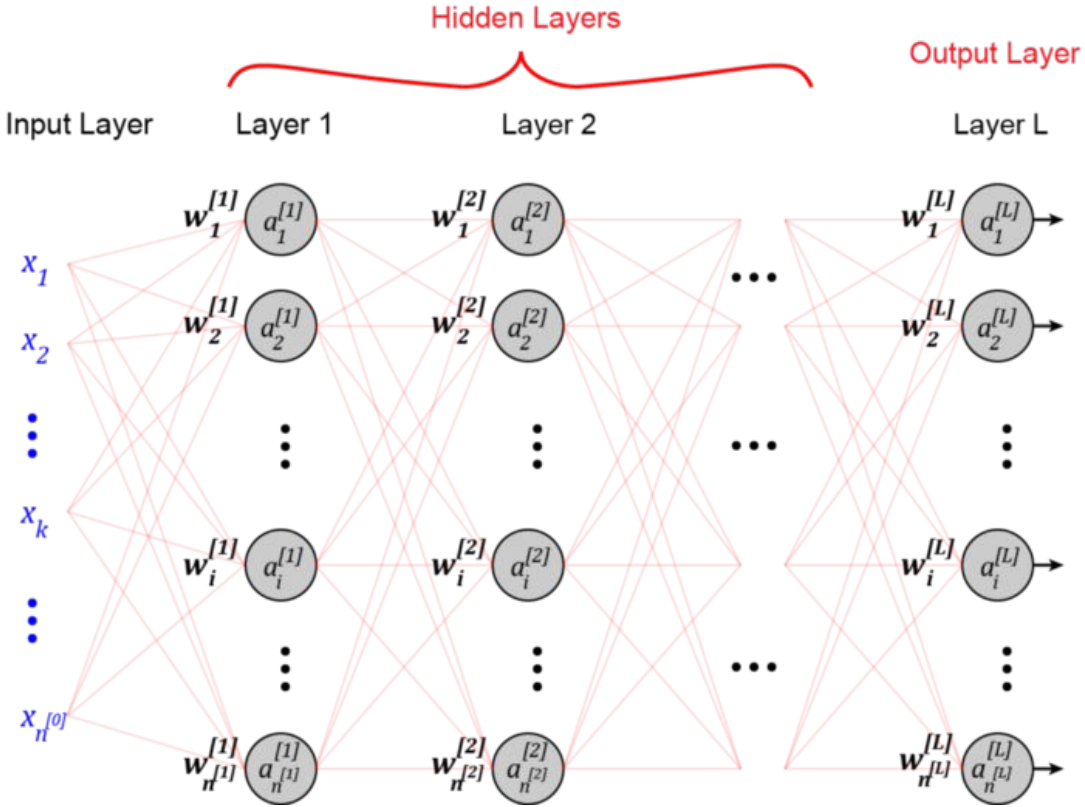
ANN


x_1
x_2
x_k
$x_{n^{[0]}}$
$w_{i1}^{[1]}$
$w_{i2}^{[1]}$
$w_{ik}^{[1]}$
$w_{in^{[0]}}^{[1]}$
$a_i^{[1]}$
$w_1^{[1]}$
$w_2^{[1]}$
$w_i^{[1]}$
$w_{n^{[1]}}^{[1]}$
$a_1^{[1]}$
$a_2^{[1]}$
$a_{n^{[1]}}^{[1]}$

ANN

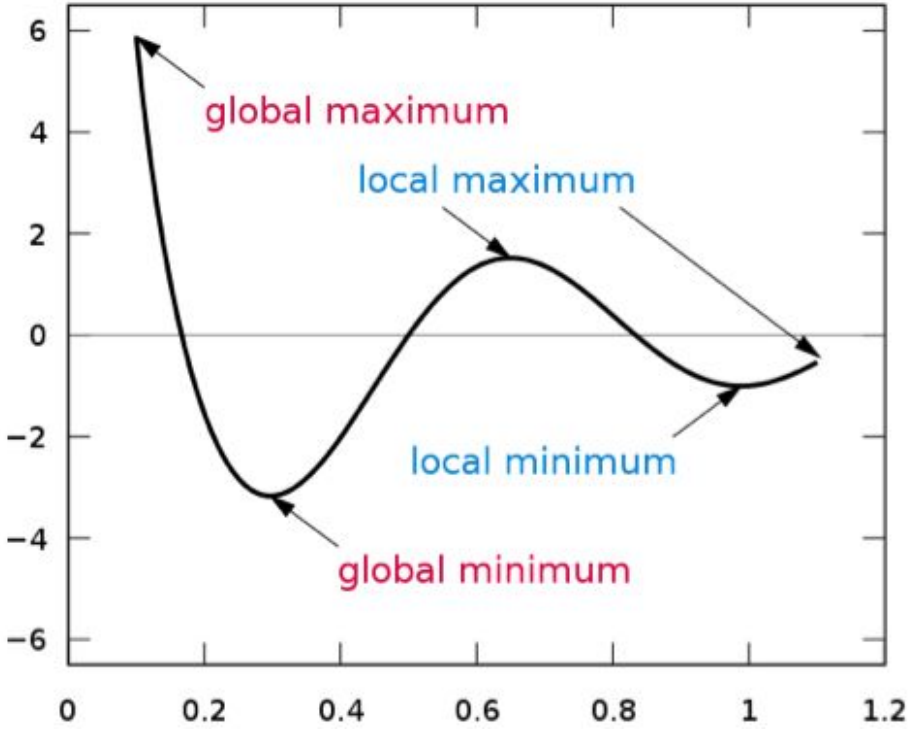
$$a_i^{[1]} = g^{[1]}\left(z_i^{[1]}\right) = g^{[1]}\left(\sum_k w_{ik}^{[1]} x_k + b_i^{[1]}\right) = g^{[1]}\left(\boldsymbol{w_i}^{[1]^T}\boldsymbol{x} + b_i^{[1]}\right) \qquad (55)$$

ANN

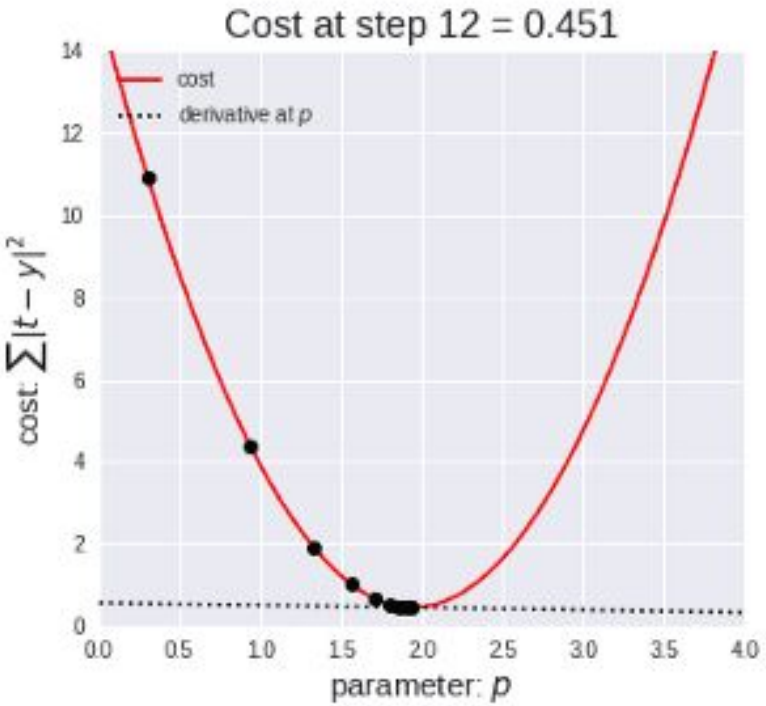

Hidden Layers
Output Layer
Input Layer
Layer 1
Layer 2
Layer L
x_1
x_2
x_k
$x_{n^{[0]}}$
$w_1^{[1]}$ $a_1^{[1]}$
$w_2^{[1]}$ $a_2^{[1]}$
$w_i^{[1]}$ $a_i^{[1]}$
$w_{n^{[1]}}^{[1]}$ $a_{n^{[1]}}^{[1]}$
$w_1^{[2]}$ $a_1^{[2]}$
$w_2^{[2]}$ $a_2^{[2]}$
$w_i^{[2]}$ $a_i^{[2]}$
$w_{n^{[2]}}^{[2]}$ $a_{n^{[2]}}^{[2]}$
$w_1^{[L]}$ $a_1^{[L]}$
$w_2^{[L]}$ $a_2^{[L]}$
$w_i^{[L]}$ $a_i^{[L]}$
$w_{n^{[L]}}^{[L]}$ $a_{n^{[L]}}^{[L]}$

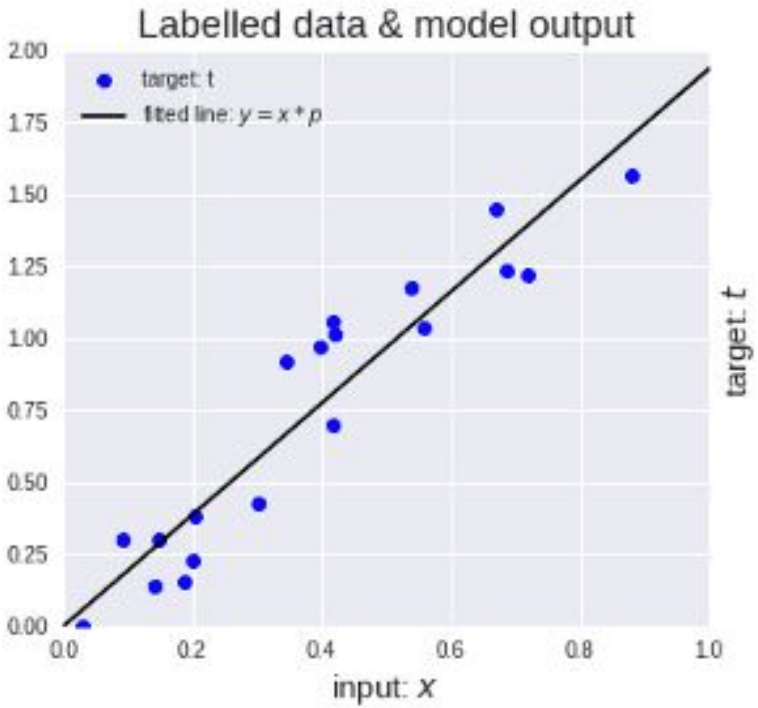
ANN - Gradient Descent

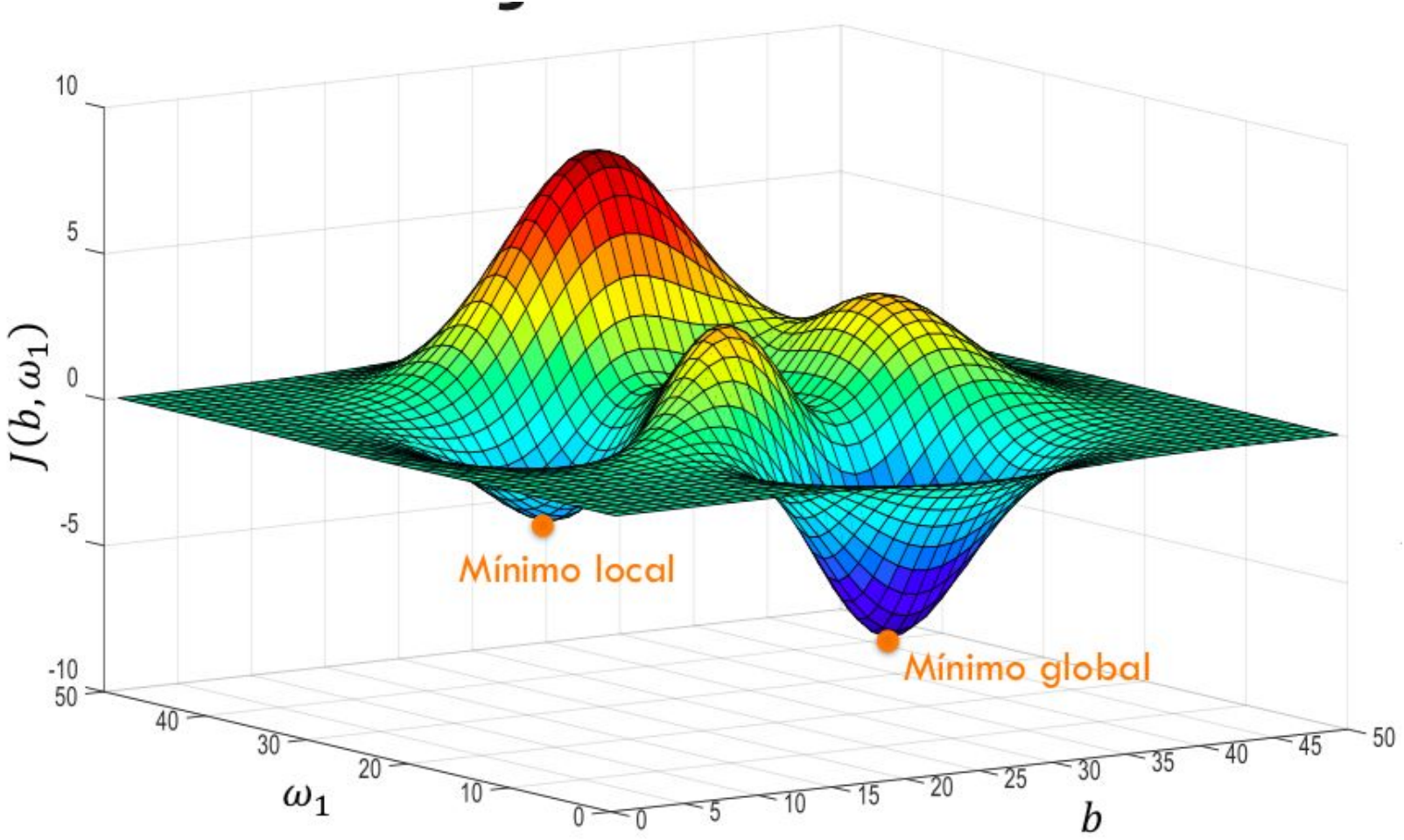
Método usado para achar os parâmetros de minimização da função custo (J)



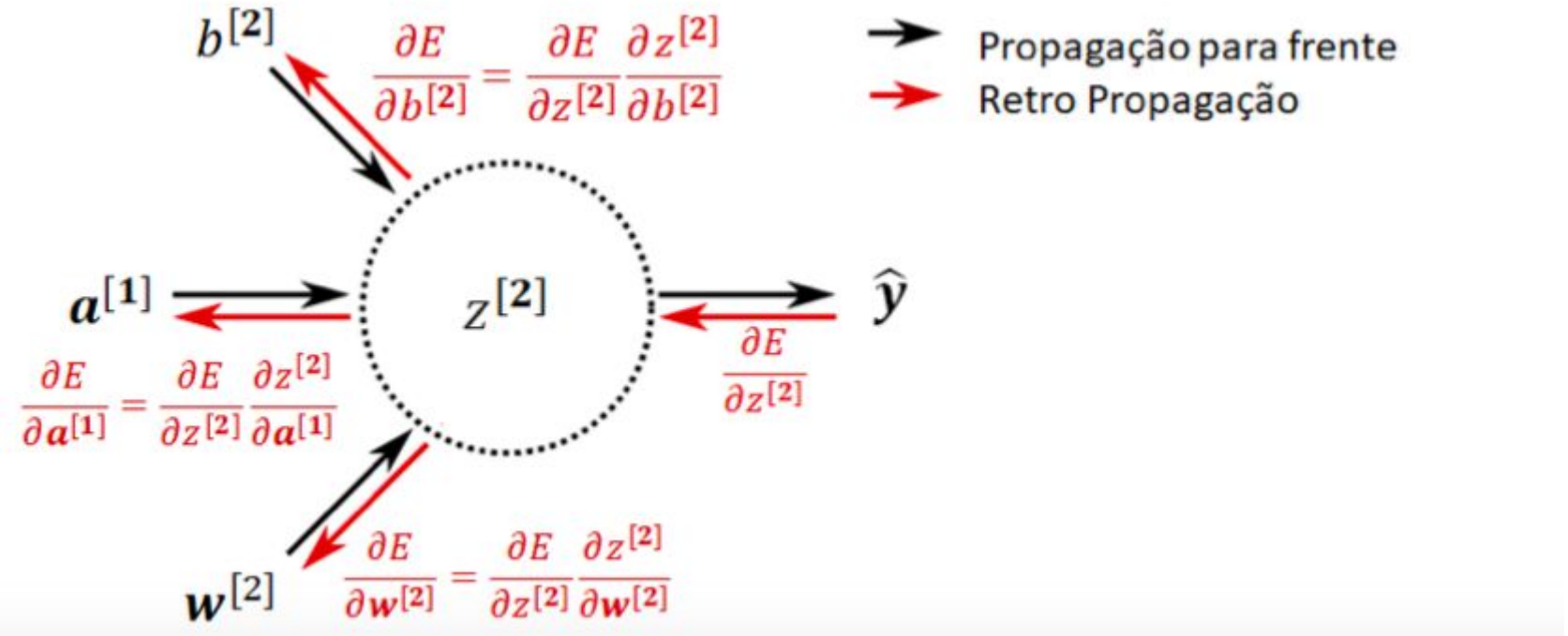
Gradient Descent


Cost at step 12 = 0.451
cost
derivative at p
cost: $\sum |t-y|^2$
parameter: p
0
2
4
6
8
10
12
14
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0

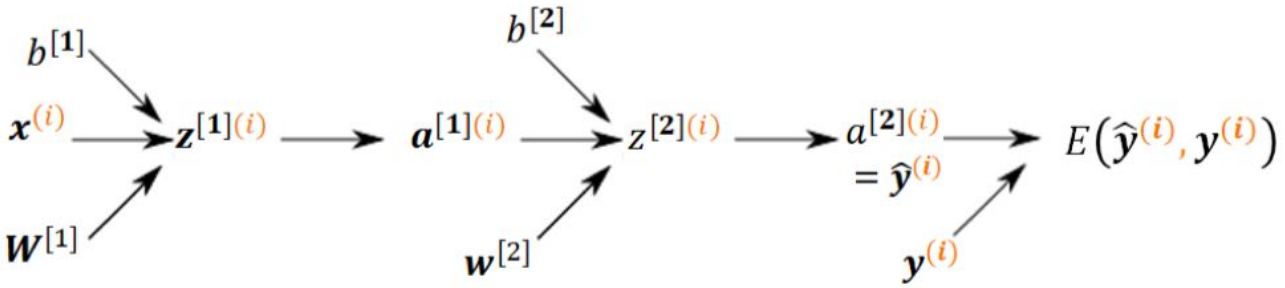

Labelled data & model output
target: t
fitted line: $y = x * p$
target: t
input: x
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
0.0
0.2
0.4
0.6
0.8
1.0


$J(b, \omega_1)$
10
5
0
-5
-10
Mínimo local
Mínimo global
50
40
30
20
10
0
ω_1
0
5
10
15
20
25
30
35
40
45
50
b

Backpropagation



Backpropagation



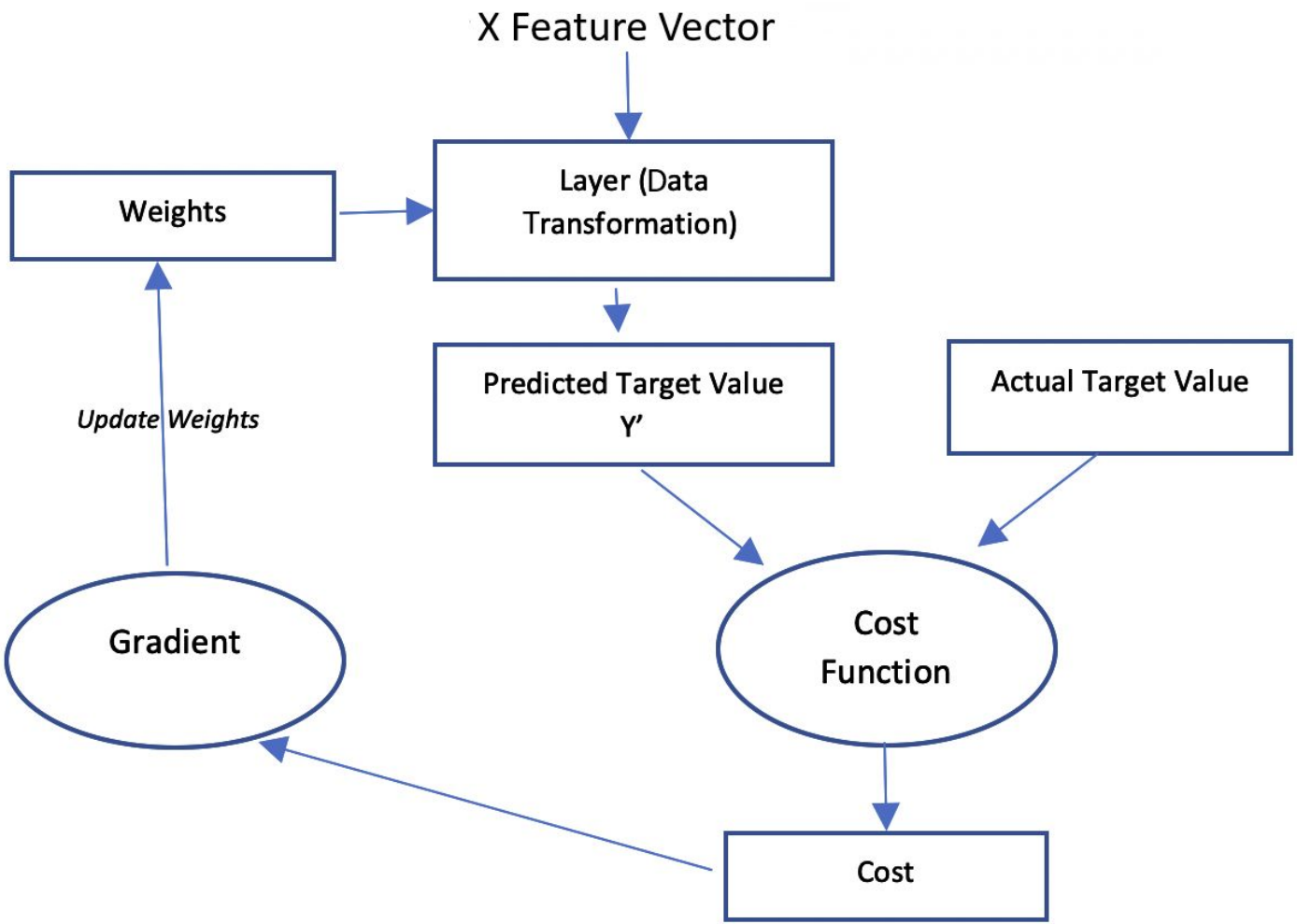
$$J(\mathbf{W}, \mathbf{b}) = \frac{1}{m} \sum_{i=1}^{m} E\left(\hat{y}^{(i)}, y^{(i)}\right)$$

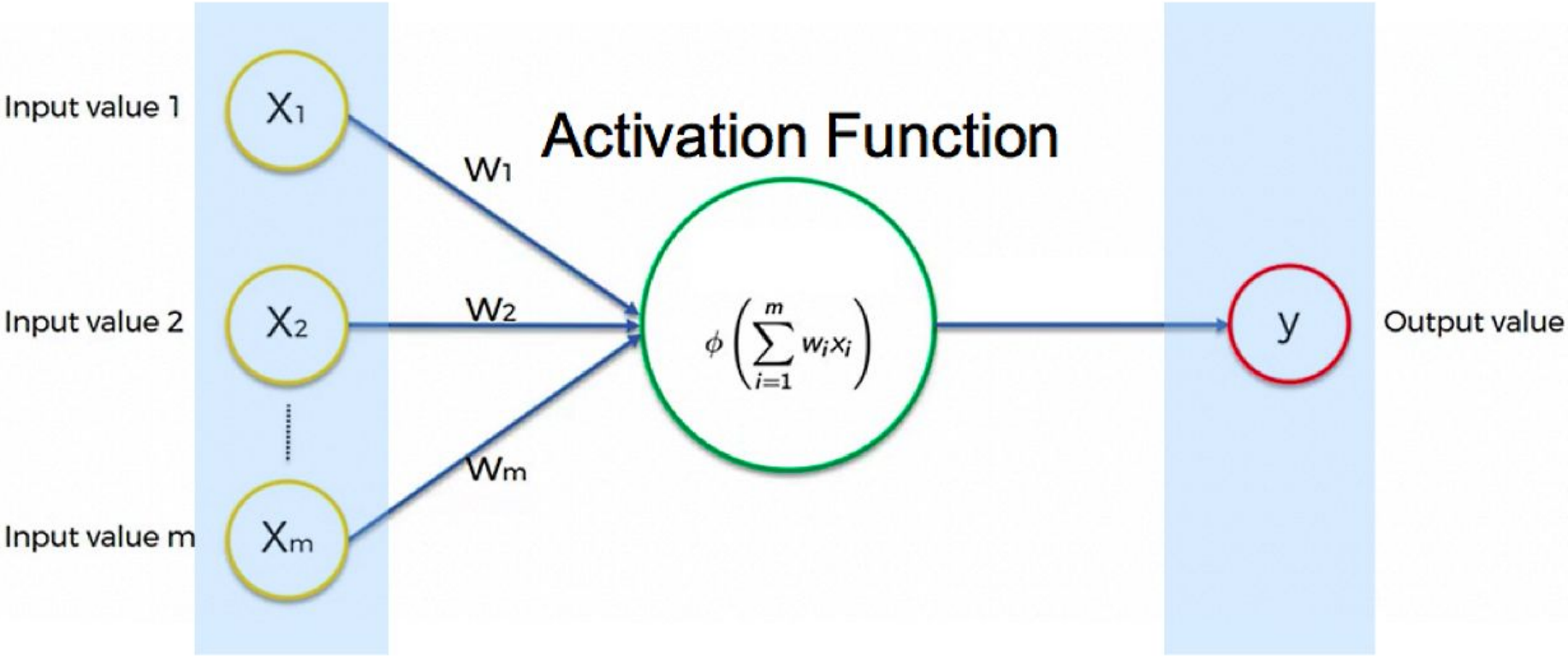
$$\frac{\partial E\left(\hat{y}^{(i)}, y^{(i)}\right)}{\partial w_{k,j}^{[l]}} = \frac{\partial E\left(\hat{y}^{(i)}, y^{(i)}\right)}{\partial a_k^{[l](i)}} \frac{\partial a_k^{[l](i)}}{\partial z_k^{[l](i)}} \frac{\partial z_k^{[l](i)}}{\partial w_{k,j}^{[l]}}$$

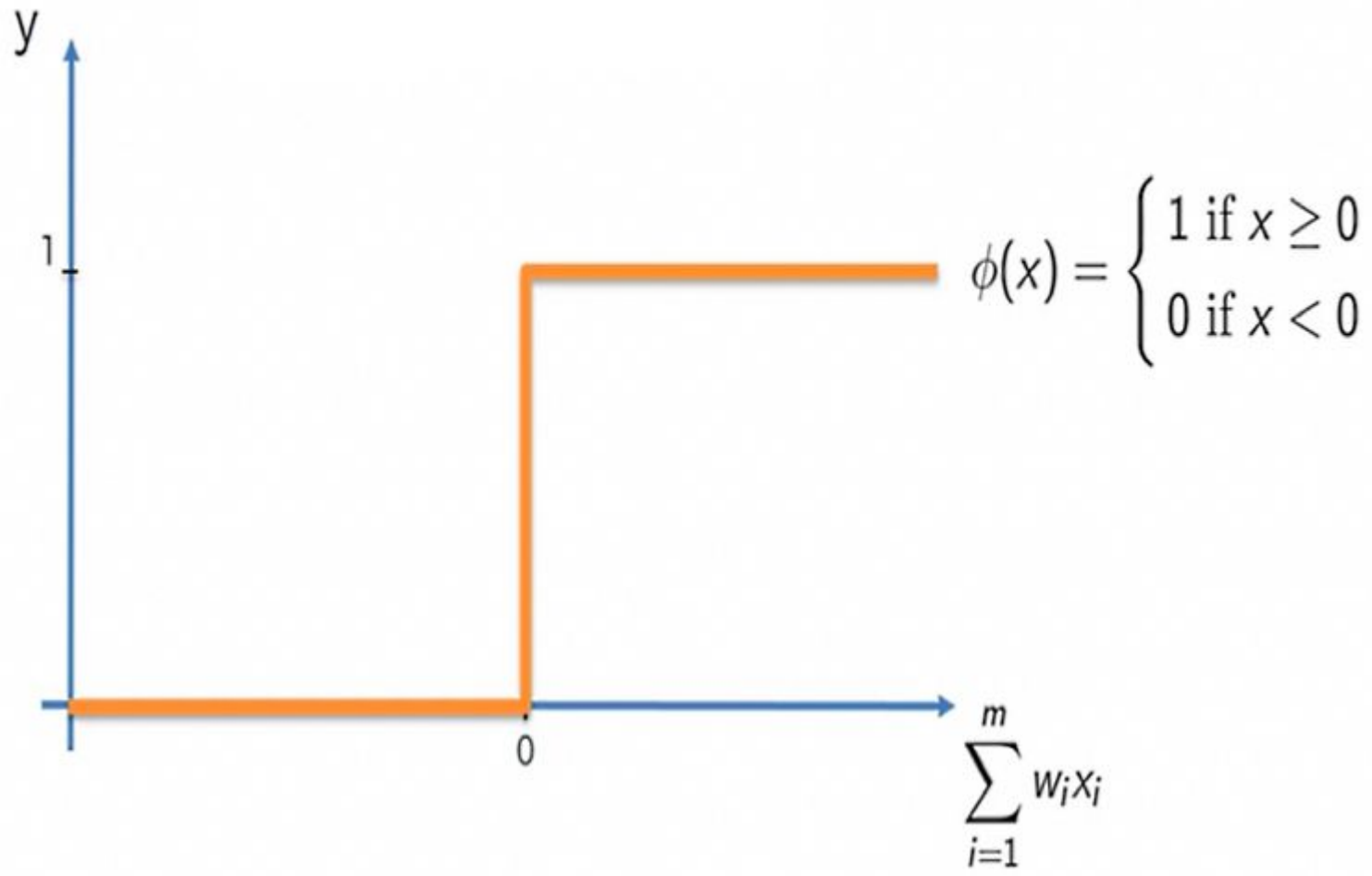
$$\frac{\partial E\left(\hat{y}^{(i)}, y^{(i)}\right)}{\partial b_k^{[l]}} = \frac{\partial L\left(\hat{y}^{(i)}, y^{(i)}\right)}{\partial a_k^{[l](i)}} \frac{\partial a_k^{[l](i)}}{\partial z_k^{[l](i)}} \frac{\partial z_k^{[l](i)}}{\partial b_k^{[l]}}$$

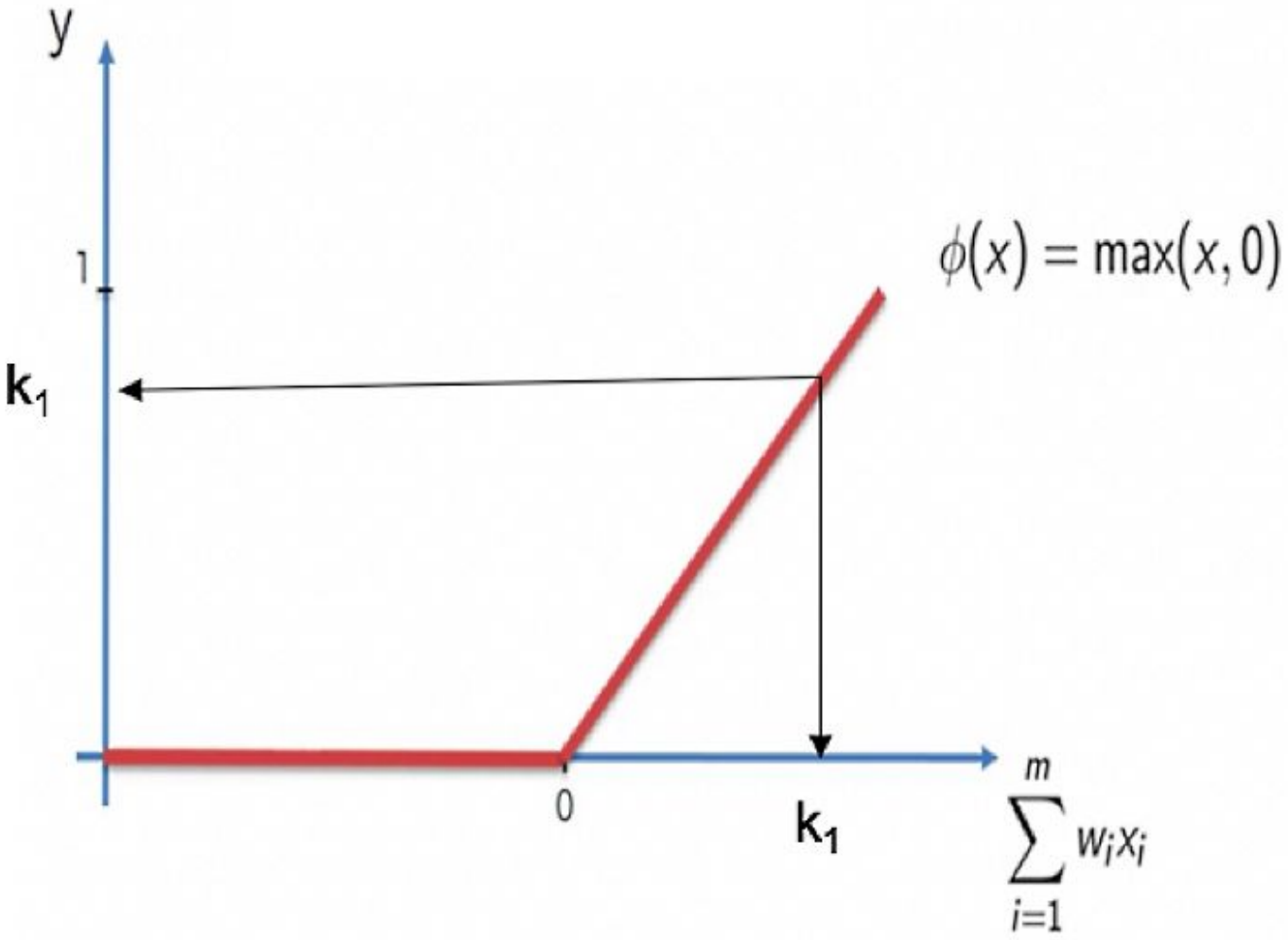
$$z_k^{[L](i)} = w_{k,j}^{[L]} a_k^{[L-1](i)} + b_k^{[L]}$$

$$a_k^{[L](i)} = g\left(z_k^{[L](i)}\right) = \hat{y}^{(i)}$$

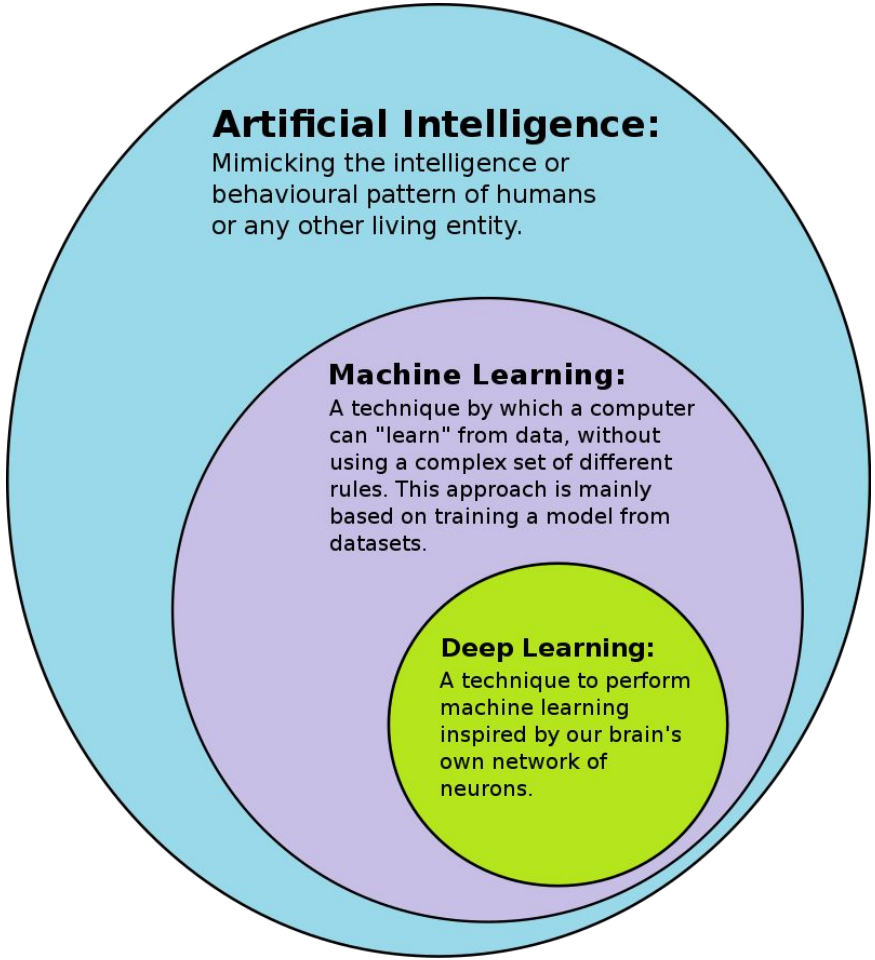

X Feature Vector
Weights
Layer (Data Transformation)
Predicted Target Value Y'
Actual Target Value
Update Weights
Gradient
Cost Function
Cost


Input value 1
Input value 2
Input value m
X1
X2
Xm
W1
W2
Wm
Activation Function
$\phi\left(\sum_{i=1}^{m} w_i x_i\right)$
y
Output value

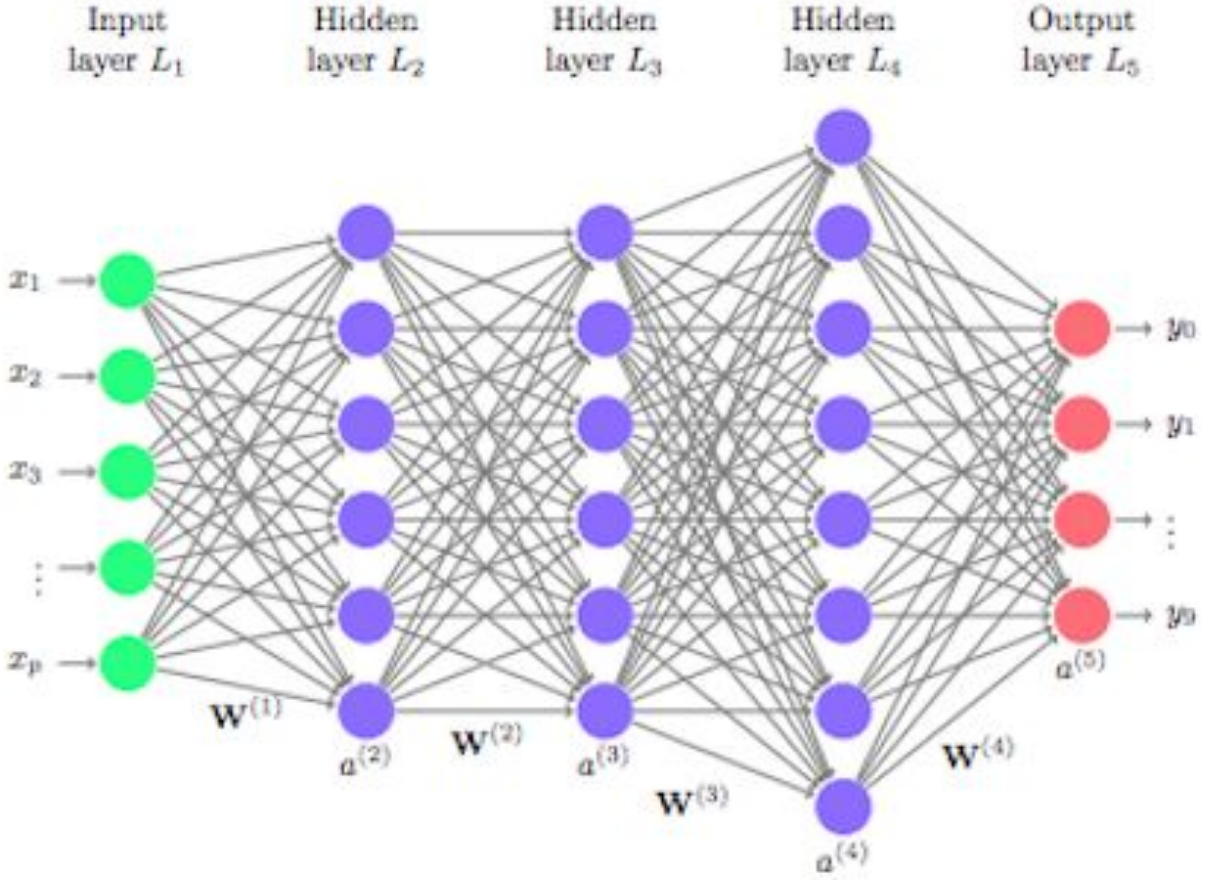

y
1
0
$\sum_{i=1}^{m} w_i x_i$
$\phi(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$


y
1
k_1
$\phi(x) = \max(x, 0)$
0
k_1
$\sum_{i=1}^{m} w_i x_i$

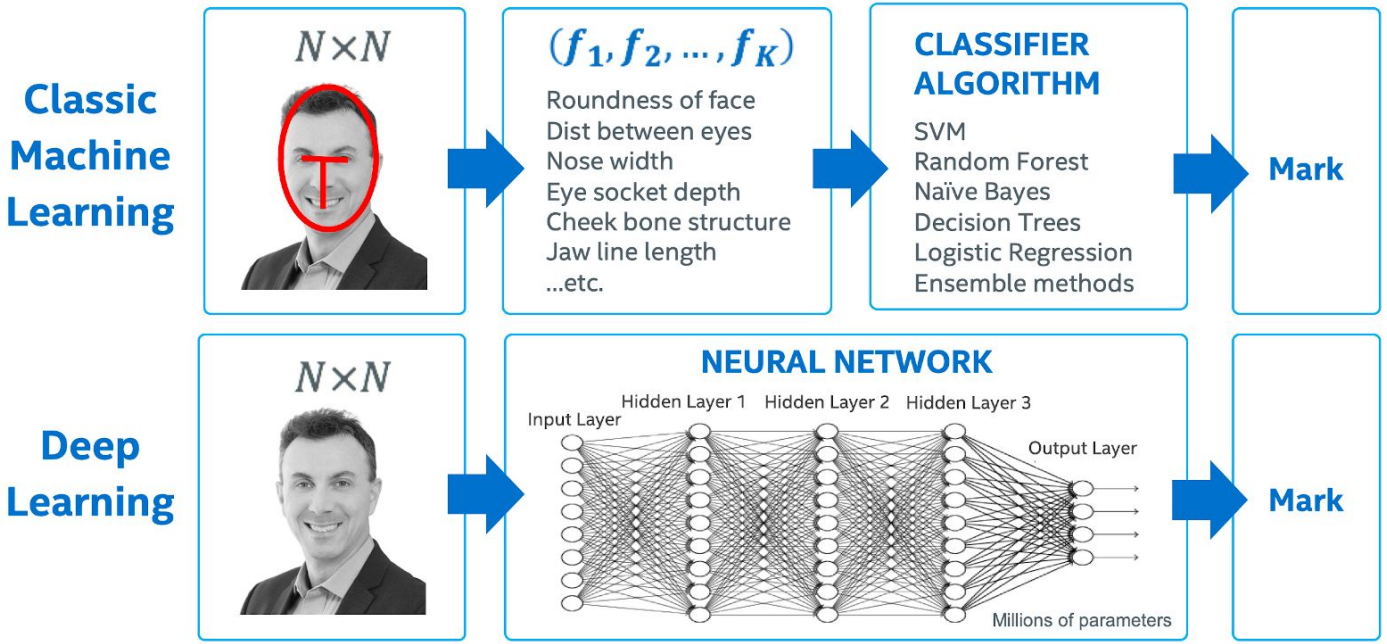
Deep-Learning



Deep-Learning



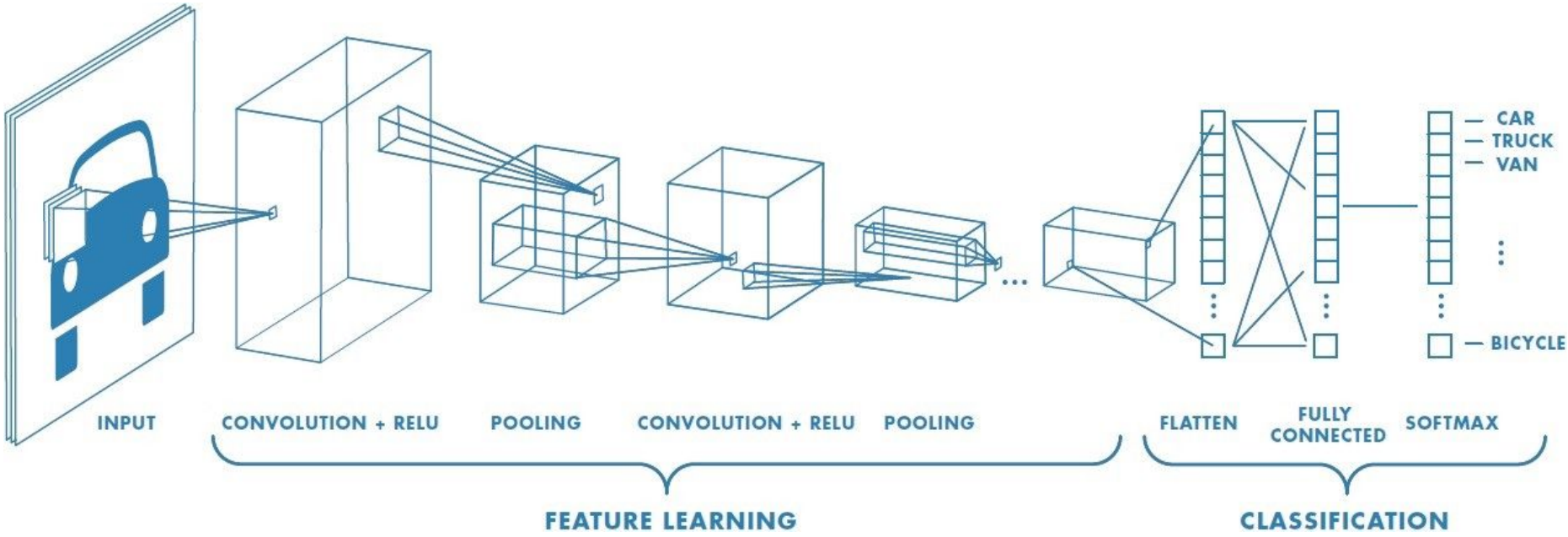
Deep-Learning



Deep-Learning

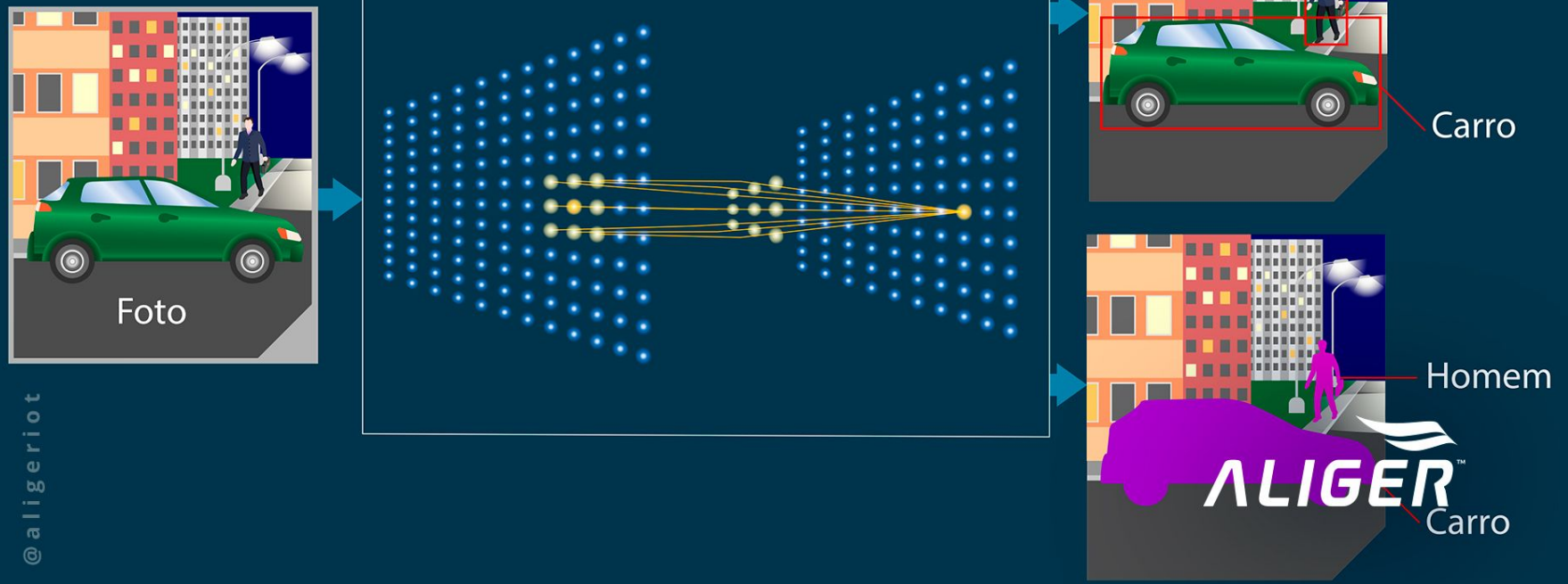
Redes Neurais Recorrentes

Redes Convolucionais

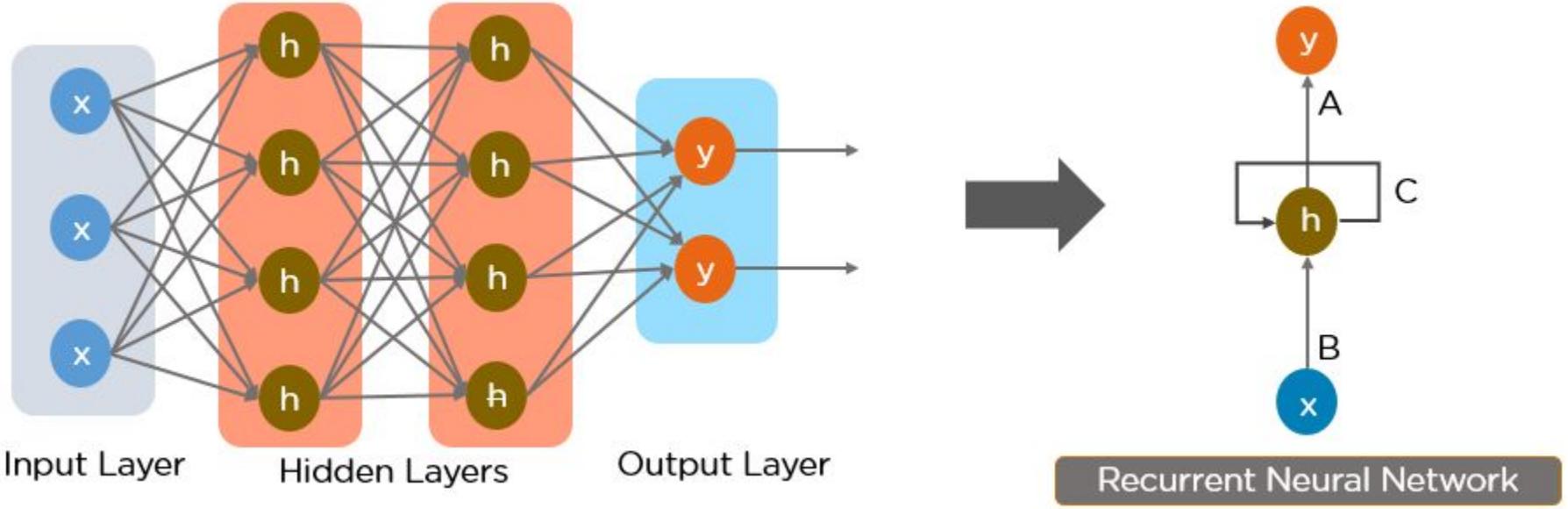

CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION

Detecção de objeto e segntação de instância

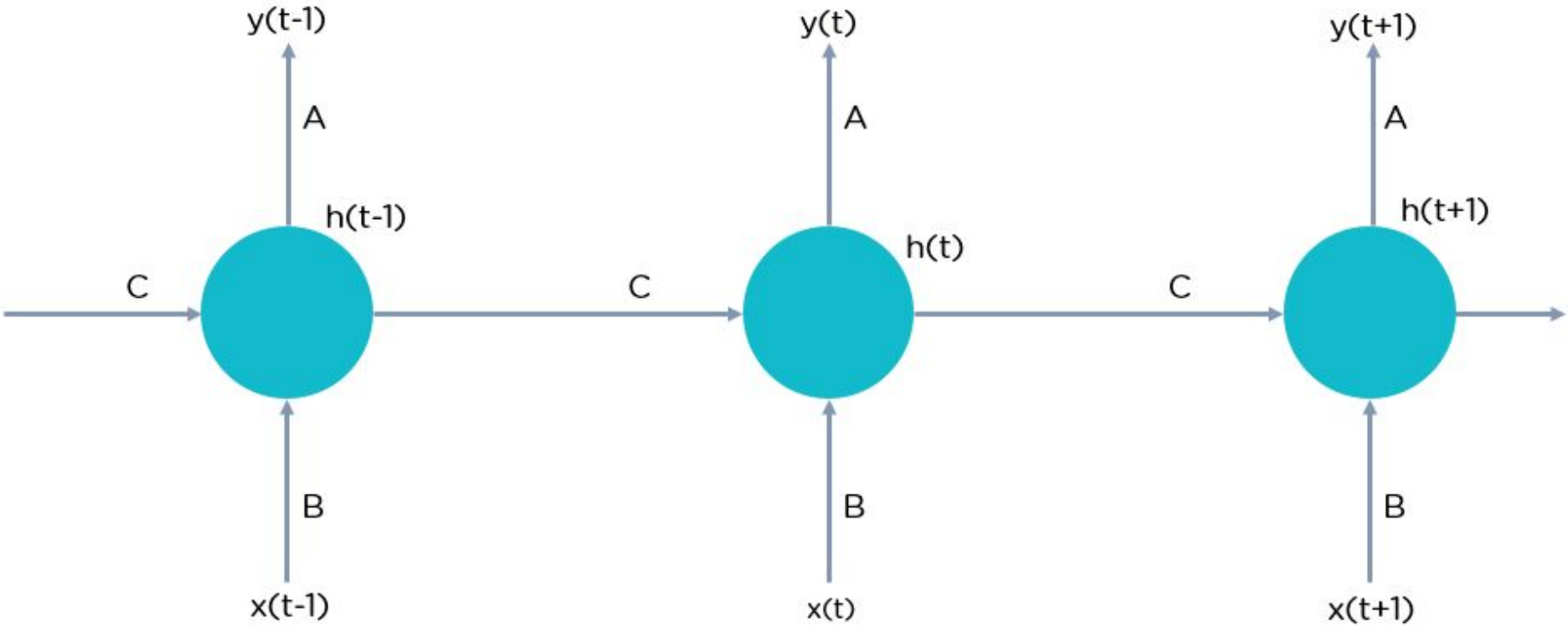
Rede Neural Convolucional



@aligeriot



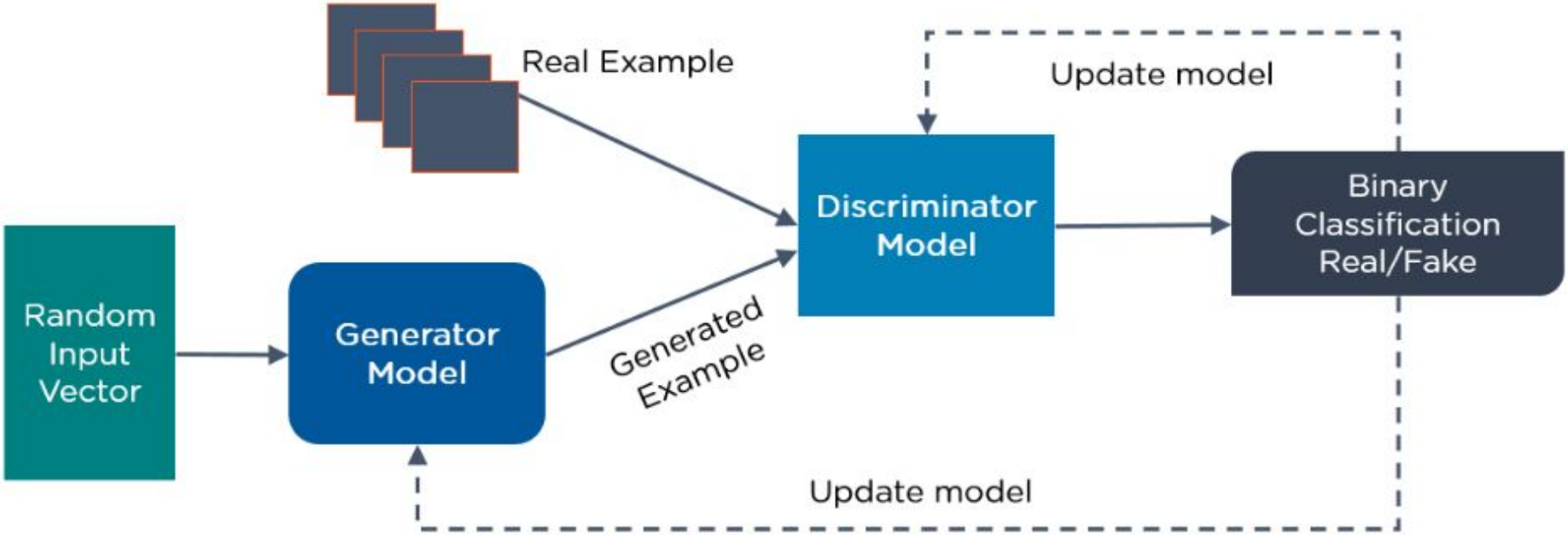
RNN -Redes Neurais Recorrentes



$h(t) = f_c(h(t-1), x(t))$

h(t) = new state
f_c = function with parameter c
h(t-1) = old state
x(t) = input vector at time step t

Generative Adversarial Networks (GANs)




source
destination
Coarse styles copied

1 ; 2 ; 3; 4 ; 5; 6; 7

0.1	0.3	….
0.2	0.4	...
0.3	0.7	
0.4	0.8	
0.5	0.1	
0.6	0.2	
0.7	0.6	….

N x M * M x D =

1*0.1 + 2*0.2 + …. + 7*0.7 =

1*0.3 + 2*0.4 + 3*0.7 + …. + 7*06 =