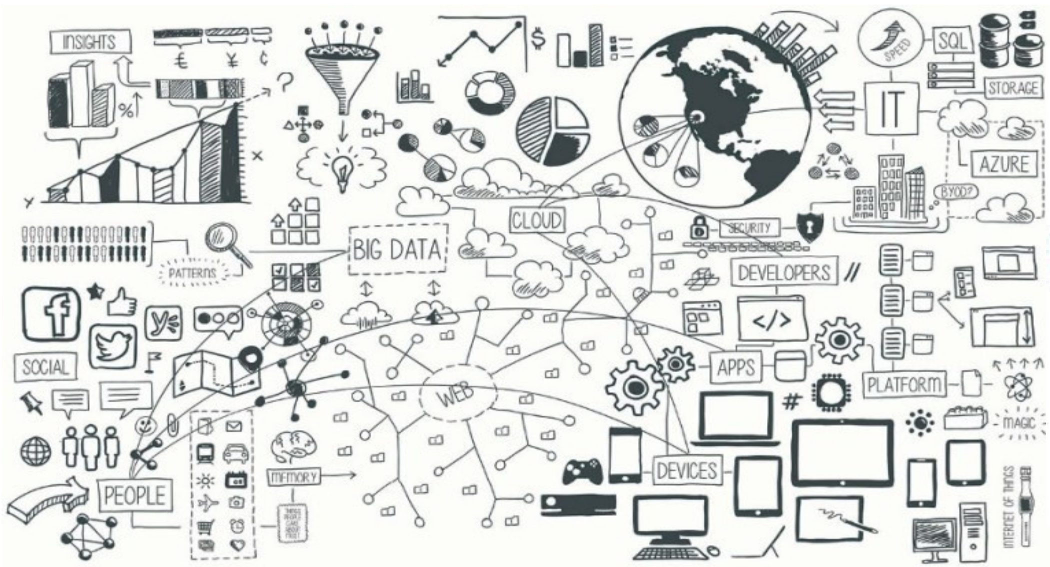
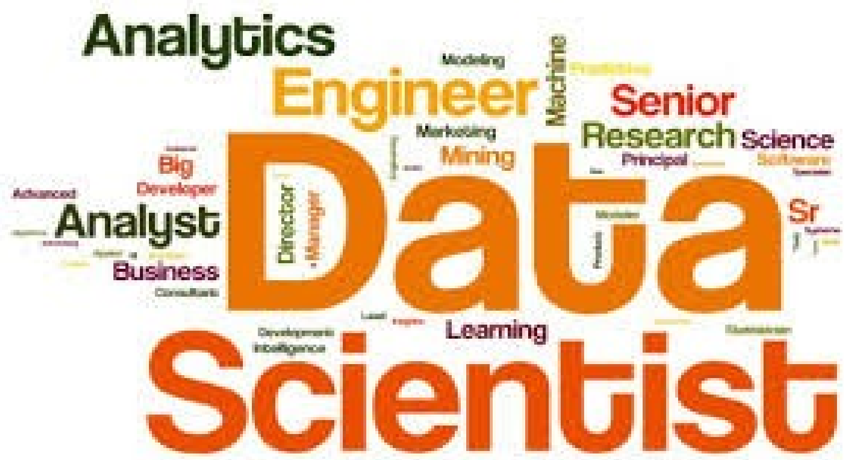
Machine Learning I (Neural Networks)

LEARNING: BACKPROPAGATION





José Manuel Gutiérrez

Javier Díez Sierra

Grupo de Meteorología

Univ. de Cantabria – CSIC
MACC / IFCA

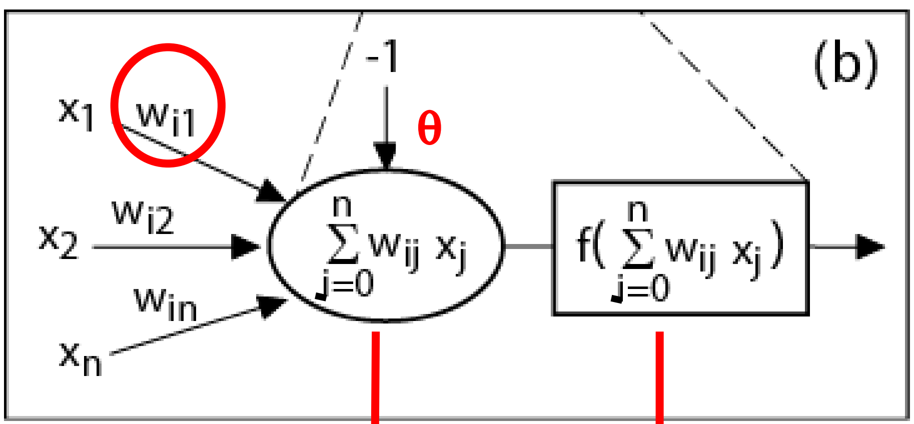
The synapses releases chemical transmitter substances, entering the dendrite, raising or lowering (**excitatory and inhibitory synapses**) the electrical potential of the cell body.

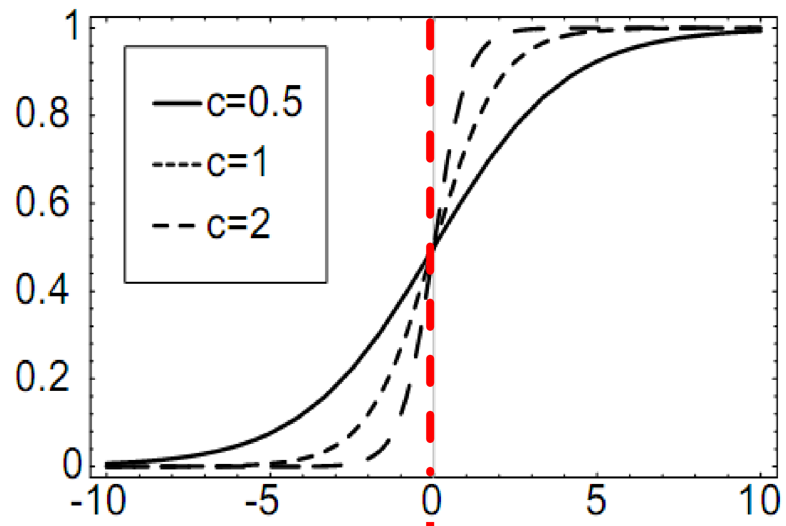
When the potential **reaches a threshold**, an electric pulse or action potential is sent down to the axon affecting other neurons *(there is a* ***nonlinear activation****).*

$$y = f(Wx), \text{ with } x_0 = -1 \text{ to account for } \theta\text{: } f(Wx - \theta).$$

McCulloc & Pitts (1943)

weights (+ or -, excitatory or inhibitory)





neuron potential:
mixed input of neighboring neurons

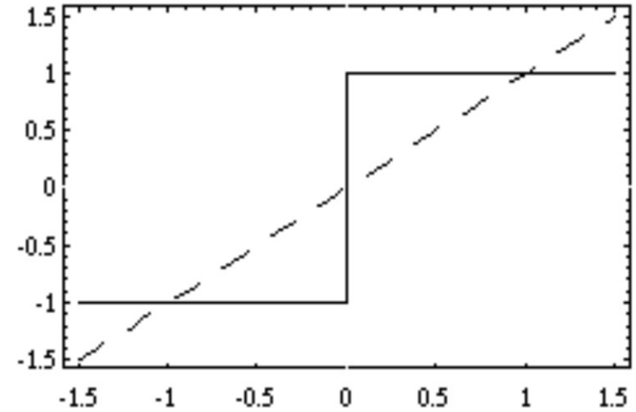
nonlinear activation function

(threshold = θ)

- **Funciones lineales:** $f(x) = x$.

- **Funciones paso:** Dan una salida binaria dependiente de si el valor de entrada está por encima o por debajo del valor umbral.

$$sgn(x) = \begin{cases} -1, & \text{si } x < 0, \\ 1, & \text{sino,} \end{cases} \quad , \quad \Theta(x) = \begin{cases} 0, & \text{si } x < 0, \\ 1, & \text{sino.} \end{cases}$$

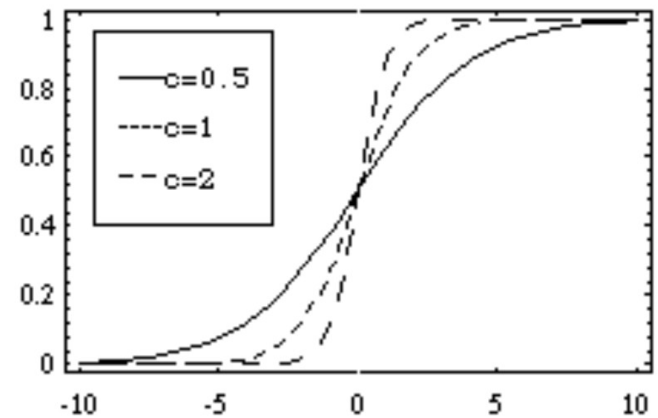


- **Funciones sigmoidales:** Funciones monótonas acotadas que dan una salida gradual no lineal.

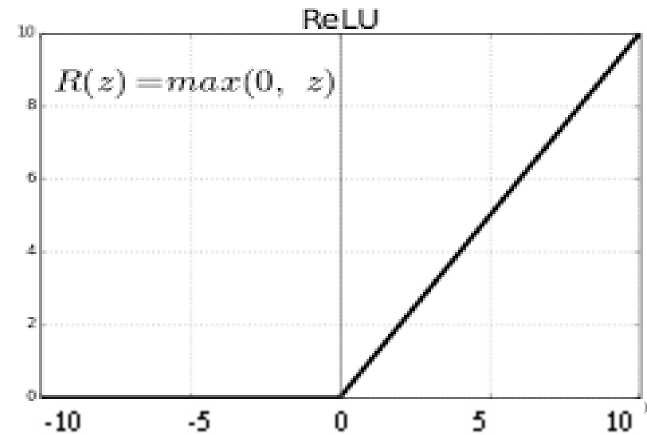
  1. La función logística de 0 a 1:

$$f_c(x) = \frac{1}{1 + e^{-c\,x}}.$$

  2. La función tangente hiperbólica de -1 a 1 $f_c(x) = tanh(c\,x)$.



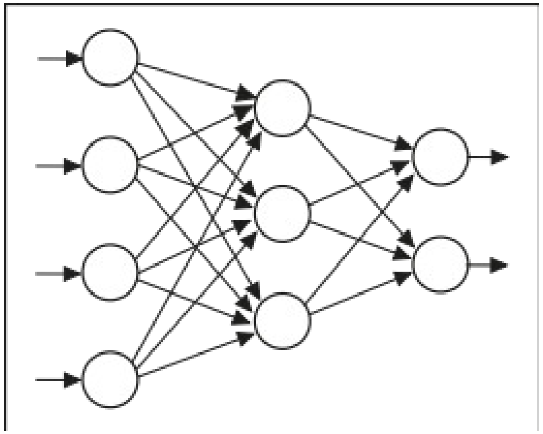
- **Rectified linear unit (ReLU):** Utilizadas para evitar el “desvanecimiento del gradiente”.



TanH	$f(x) = tanh(x) = \frac{2}{1+e^2x} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	C^∞
SoftSign	$f(x) = \frac{x}{1+\lvert x\rvert}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	C^1
SoftPlus	$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	$(0, \infty)$	C^∞
SoftExponential	$f(\alpha, x) = \begin{cases} -\frac{\ln(1-\alpha(x+\alpha))}{\alpha} \text{ for } \alpha < 0 \\ x \text{ for } \alpha = 0 \\ \frac{e^{\alpha x}-1}{\alpha} + \alpha \text{ for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1-\alpha(\alpha+x)} \text{ for } \alpha < 0 \\ e^{\alpha x} \text{ for } \alpha \geq 0 \end{cases}$	$(-\infty, \infty)$	C^∞
Sinusoid	$f(x) = \sin(x)$	$f'(x) = \cos(x)$	$[-1, 1]$	C^∞
Sinc	$f(x) = \begin{cases} 1 \text{ for } x = 0 \\ \frac{\sin(x)}{x} \text{ for } x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \text{ for } x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} \text{ for } x \neq 0 \end{cases}$	$[\approx -.217234, 1]$	C^∞
Scaled exponential linear unit (SELU)	$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$ $\lambda = 1.0507 \text{ y } \alpha = 1.67326$	$f'(\alpha, x) = \lambda \begin{cases} f(\alpha, x) + \alpha \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{cases}$	$(-\lambda\alpha, \infty)$	C^0
Rectified linear unit (ReLU)	$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \quad \text{for} \quad x < 0 \\ 1 \quad \text{for} \quad x \geq 0 \end{cases}$	$[0, \infty)$	C^0
Randomized leaky rectified linear unit (RReLU)	$f(\alpha, x) = \begin{cases} \alpha x \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0
Parametric rectified linear unit (PReLU)	$f(\alpha, x) = \begin{cases} \alpha x \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0
Logistic (a.k.a soft step)	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	(0,1)	C^∞

Supervised Problems. Input-Output pairs are provided:
(x1,y1), (x2,y2), ..., (xn,yn) and the network learns y = f(x+ε).

Multilayer* *Networks or Feedforward Nets.
Several layers connected (input+hidden+output)



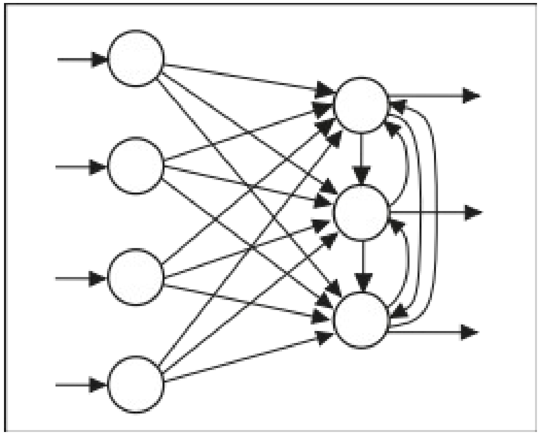
Pattern Recognition
OCR, images
Interpolation and fitting

Prediction: Input => Output

Learning: Backpropagation

Unsupervised Problems. Only input data is provided:
x1, x2, ..., xn and the network self-organizes it to provide a clustering.

Competitive* *Networks
Multilayer networks with lateral connections (competitive) in the last layer.



Segmentation

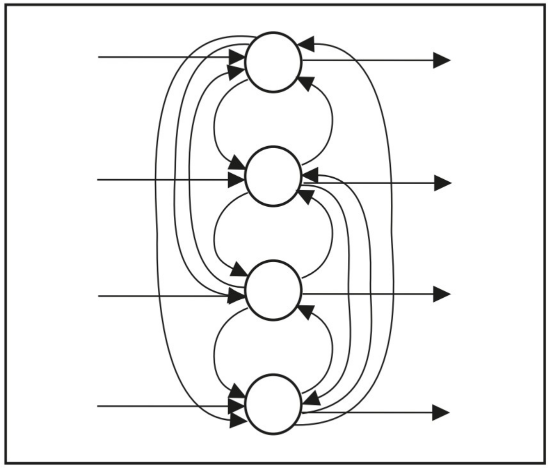
Feature extraction.

Prediction: Input => Clusters

Learning: Ad hoc
Winner-takes-all

Supervised Problems. **Input-Input pairs are provided:**
(x1,x1), (x2,x2), ..., (xn,xn) and the network learns $x = f(x+\varepsilon)$.

***Autoassociative memories* (Hopfield).**
Single layer with lateral delayed connections.



Pattern Recognition
OCR, images
Memories (robust to noise)

Prediction: Input => Input

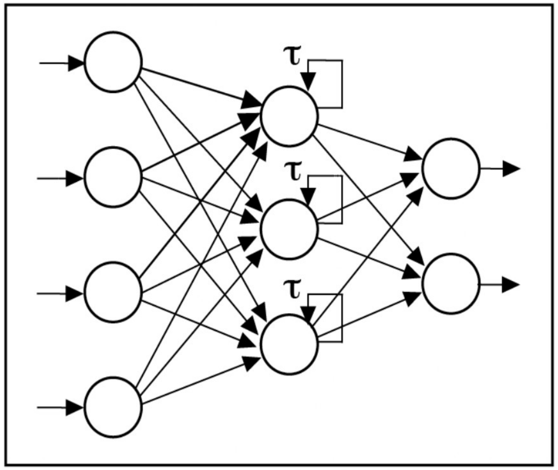
Learning: Hegg

Autoencoders (later)

Feature extraction, compression.

Supervised Problems (with memory). **Input-Output pairs are provided:**
(x1,y1), (x2,y2), ..., (xn,yn) and the network learns $y_t = f(x_{t-1,t-2,\ldots}+\varepsilon)$.

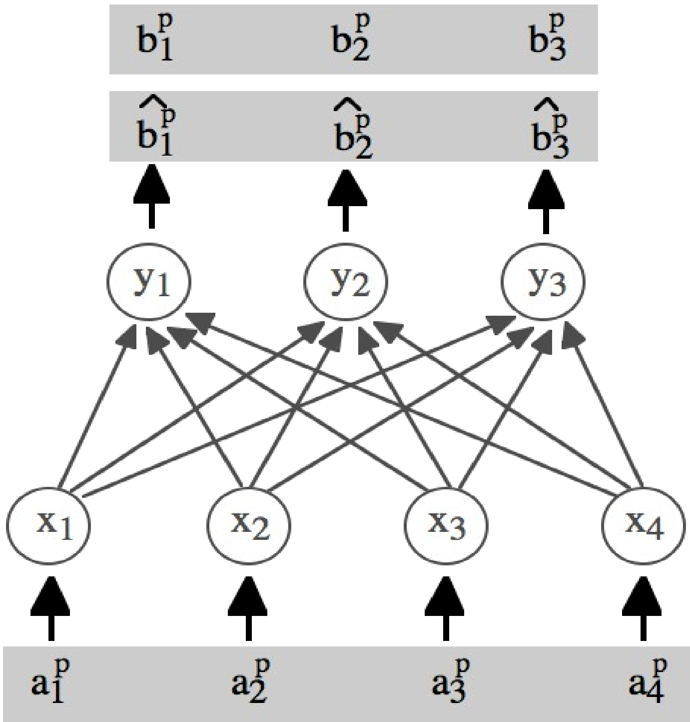
***Recurrent* Networks or Elman/Jordan nets.**
Multilayer network with hidden/output delayed lines.

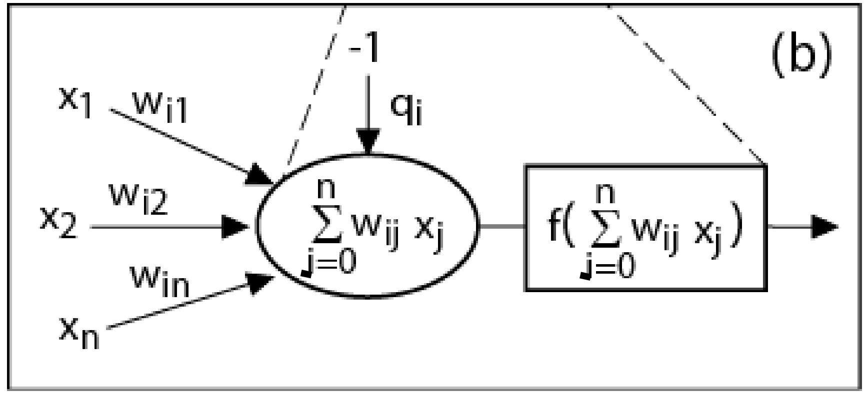


Time series analysis
Video, natural language
Interpolation and fitting

Prediction: Input => Output

Learning: Backpropagation in time





$$E(w) = \frac{1}{2}\sum_{i,p}(b_i^p - \hat{b}_i^p)^2.$$

Inicialmente se eligen valores aleatorios para los pesos.

Descenso de gradiente: Se modifican los pesos acorde la dirección del gradiente del error.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \sum_p (b_i^p - \hat{b}_i^p) f'(B_i^p) a_j^p$$

η : Tasa de aprendizaje

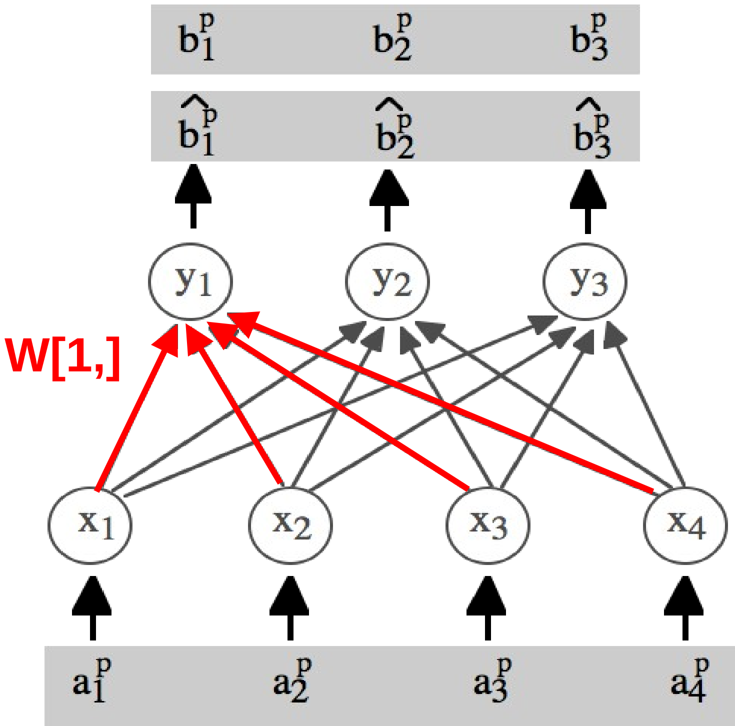
Inercia

$$\Delta w_{ij}(t+1) = -\eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(t-1)$$

Regularización

$$E(w) = \sum_{p=1}^{r} (y_p - \hat{y}_p)^2 + \lambda \sum_{i,j} w_{ij}^2$$

RSNNS



$$E(w) = \frac{1}{2}\sum_{i,p}(b_i^p - \hat{b}_i^p)^2.$$

Descenso de gradiente: Se modifican los pesos acorde la dirección del gradiente del error.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \sum_p (b_i^p - \hat{b}_i^p) f'(B_i^p) a_j^p$$

η : Tasa de aprendizaje

ALGORITMO:

1. Se inicializan aleatoriamente los pesos:
   ```
   W[i,j] = np.random.uniform(…)
   ```
2. Se asigna la tasa de aprendizaje:
   ```
   eta=0.1
   ```
3. For i in range(epochs):
   ```
   W[i,j] = W[i,j] - eta*w_delta[i,j]
   ```

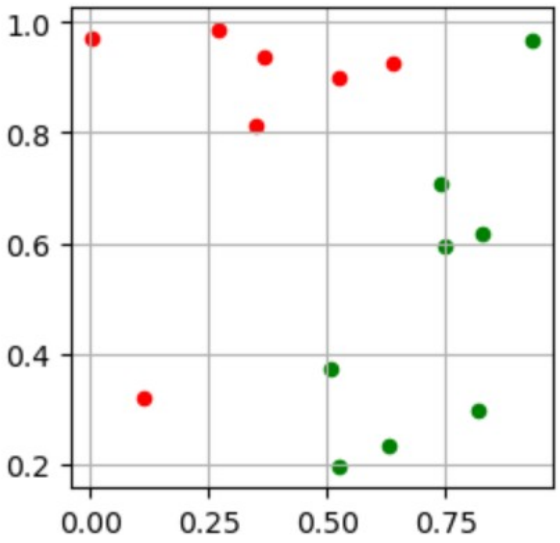
Inercia

$$\Delta w_{ij}(t+1) = -\eta \frac{\partial E}{\partial w_{ij}} + \alpha \Delta w_{ij}(t-1)$$

Regularización

$$E(w) = \sum_{p=1}^{r} (y_p - \hat{y}_p)^2 + \lambda \sum_{i,j} w_{ij}^2$$

```
lin = pd.read_csv("lineal.csv", names=['x1', 'x2', 'y'])

ceros = np.where(lin['y']==0)[0];
unos = np.where(lin['y']==1)[0]
plt.figure(figsize = [3, 3])
plt.scatter(lin['x1'][ceros], lin['x2'][ceros], color = 'r', s=20)
plt.scatter(lin['x1'][unos], lin['x2'][unos], color = 'g', s=20)


a = lin[['x1', 'x2']].T.values
np.shape(a)
# (2,  15)

b = lin['y'].values.reshape(1, len(lin['y'].values))
np.shape(b)
# (1, 15)

# Producto escalar con np.dot()
np.shape(np.dot(b.T, b))
# (15, 15)

#Incluir el bias (opcional):
# a = np.vstack([a, np.ones([1, a.shape[1]])])
```



```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def d_sigmoid(x):
    return sigmoid(x)*(1-sigmoid(x))
```

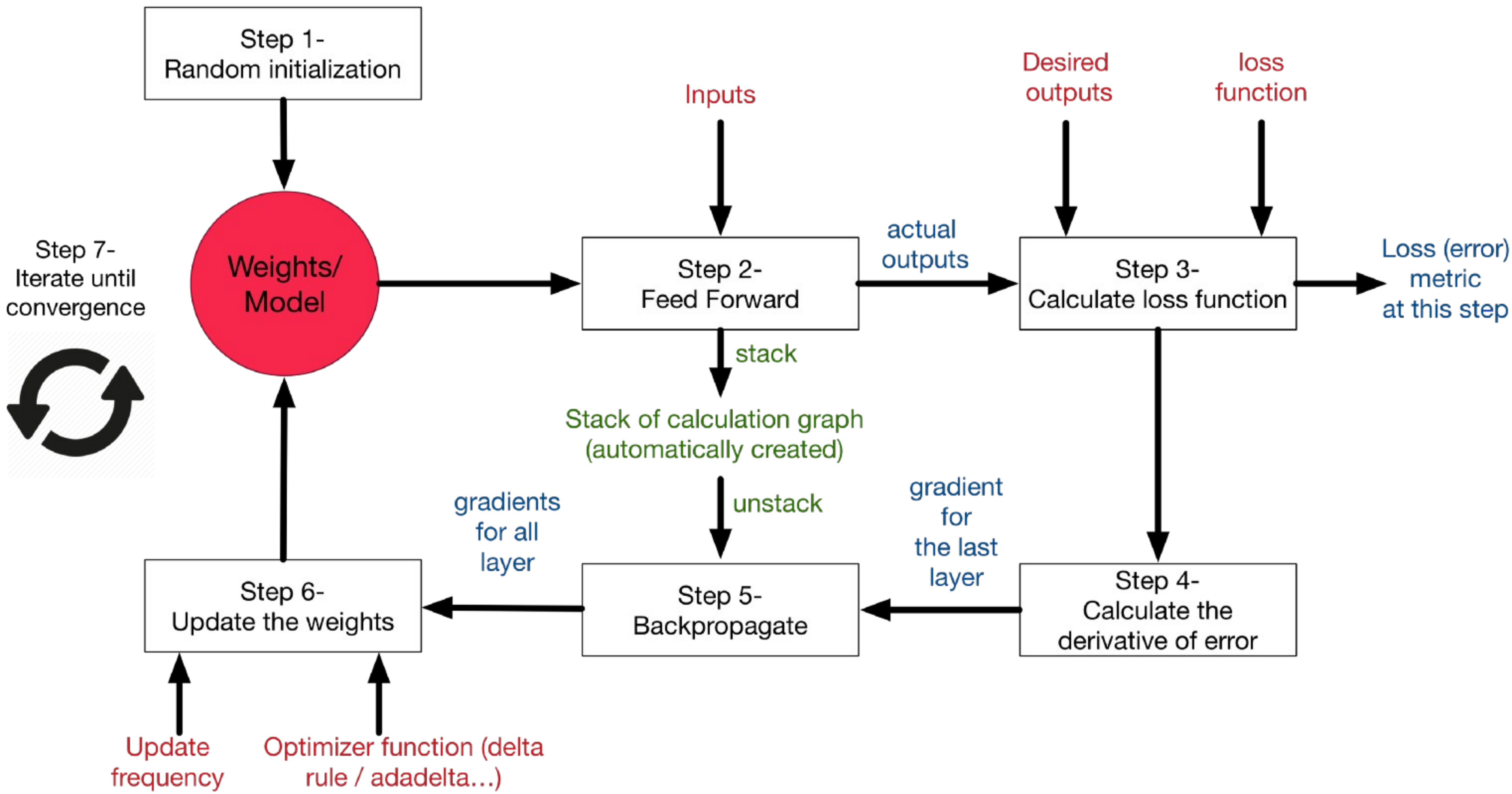
$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \sum_p (b_i^p - \hat{b}_i^p) f'(B_i^p) a_j^p$$

```
neurons = [np.shape(a)[0], np.shape(b)[0]]
weights = np.random.uniform(low=-1, high=1, size=neurons)
bias = np.random.uniform(low=-1, high=1, size = (neurons[-1], 1))

B_out = np.dot(weights.T, a) + bias
b_out = sigmoid(B_out)
np.shape(b_out)
# (1,  15)

d_error = b_out - b # derivada función de perdida
aux_delta = d_error*d_sigmoid(B_out)
w_delta = np.dot(a, aux_delta.T)
b_delta = aux_delta.sum(axis = 1, keepdims=True)
np.shape(w_delta)
# (2,  1)
np.shape(b_delta)
# (1,  1)

weights = weights - eta * w_delta
bias = bias - eta *  b_delta
```

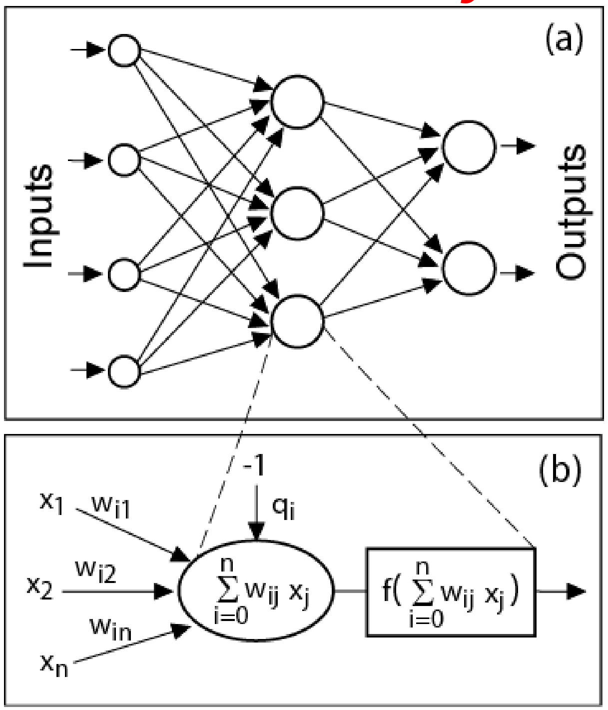
```python
lin = pd.read_csv("lineal.csv", names=['x1', 'x2', 'y'])
a = lin[['x1', 'x2']].T.values
b = lin['y'].values.reshape(1, len(lin['y'].values))

def backprop(a,b, epochs = 500, eta = 0.1)
```

```python
def backprop(a,b, epochs = 500, eta = 0.1):
        ### Inicializar matrices y listas

    for i in range(epochs):
        ### Propagar hacia delante

        ### Actualizar pesos

        ### Error output
        print(error)

    ### Return values
    return error, pesos, sesgos..
```

https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e



x : h : y



The neural activity (output) is given by a *nonlinear function*

$$y_i = f\Big(\sum_k W_{ik}\, f\big(\sum_j w_{kj} a_{pj}\big)\Big) \quad h_i$$

$$E(w) = \frac{1}{2}\sum_{p,i}\Big(b_{pi} - f\big(\sum_k W_{ik}\, f(\sum_j w_{kj} a_{pj})\big)\Big)^2$$

Gradient descent $\Delta W_{ik} = -\eta \frac{\partial E}{\partial W_{ik}}; \quad \Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}},$

Ver desarrollo en el documento
1999_BookCCGP_caps1_2.pdf
páginas 30 a 32.
[Disponible en moodle]

Illustrative video:

https://www.youtube.com/watch?v=Ilg3gGewQ5U

1. Init the neural weight with random values
2. Select the input and propagate it (estimate hidden and output)
3. Compute the error associate with the output

$$\delta_{pi} = (b_{pi} - \hat{b}_{pi}) f'(\hat{B}_{pi}) = (b_{pi} - \hat{b}_{pi})\hat{b}_{pi}(1 - \hat{b}_{pi})$$

4. Compute the error associate with the hidden neurons

$$\psi_{pk} = \sum_i \delta_{pi} W_{ki}\, f'(\hat{H}_{pk})$$

5. Compute

$$\Delta W_{ik} = \eta\, \delta_{pi} \hat{h}_{pk}, \quad \Delta w_{kj} = \eta\, \psi_{pk}\, a_{pj}$$

and update the neural weight according to these values

How the backprop algorithm works

http://neuralnetworksanddeeplearning.com/chap2.html

Feed Forward

$$A^l = w^l a^{l-1} + b^l$$

$$a^l = f(A^l)$$

$$E = \frac{1}{2} \sum_x \left\| y(x) - a^L \right\|^2$$

Gradient Descent

$\partial E_x / \partial w$ y $\partial E_x / \partial b$

$$w^l_t = w^l_{t-1} - \eta \partial E / \partial w^l$$

$$b^l_t = b^l_{t-1} - \eta \partial E / \partial b^l$$

Back Propagation

$$\partial E_x / \partial w^L = (a^L - y) \odot \sigma'(A^L) a^{L-1}$$

$$\boxed{\begin{aligned} \delta^L &= (a^L - y) \odot \sigma'(A^L) \\ \delta^l &= ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(A^l) \end{aligned}}$$

$$\partial E / \partial w^l = \delta^l a^{l-1}$$

$$\partial E / \partial b^l = \sum \delta^l$$