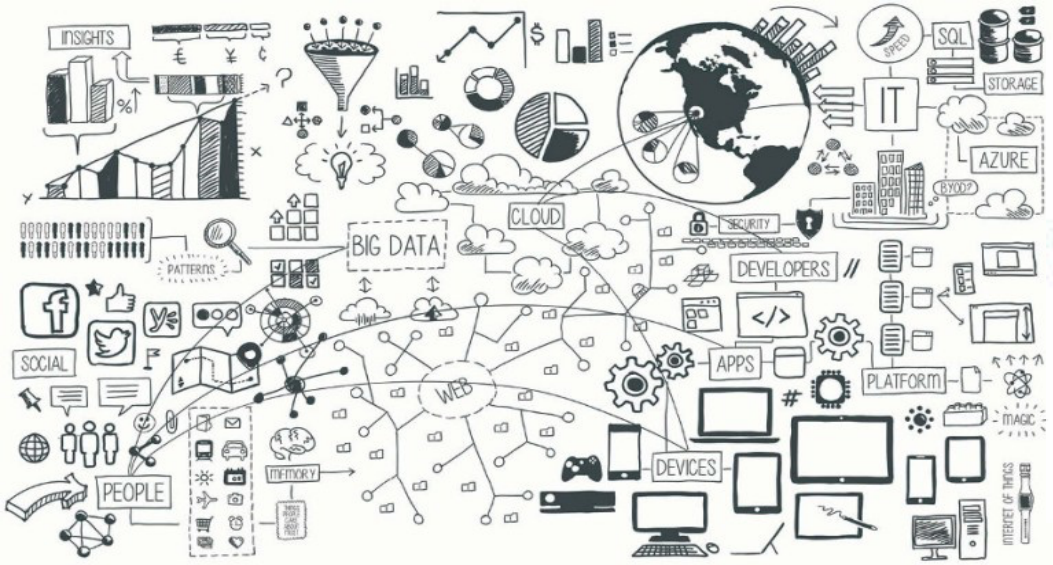
Machine Learning I (Neural Networks)

LEARNING: BACKPROPAGATION



José Manuel Gutiérrez

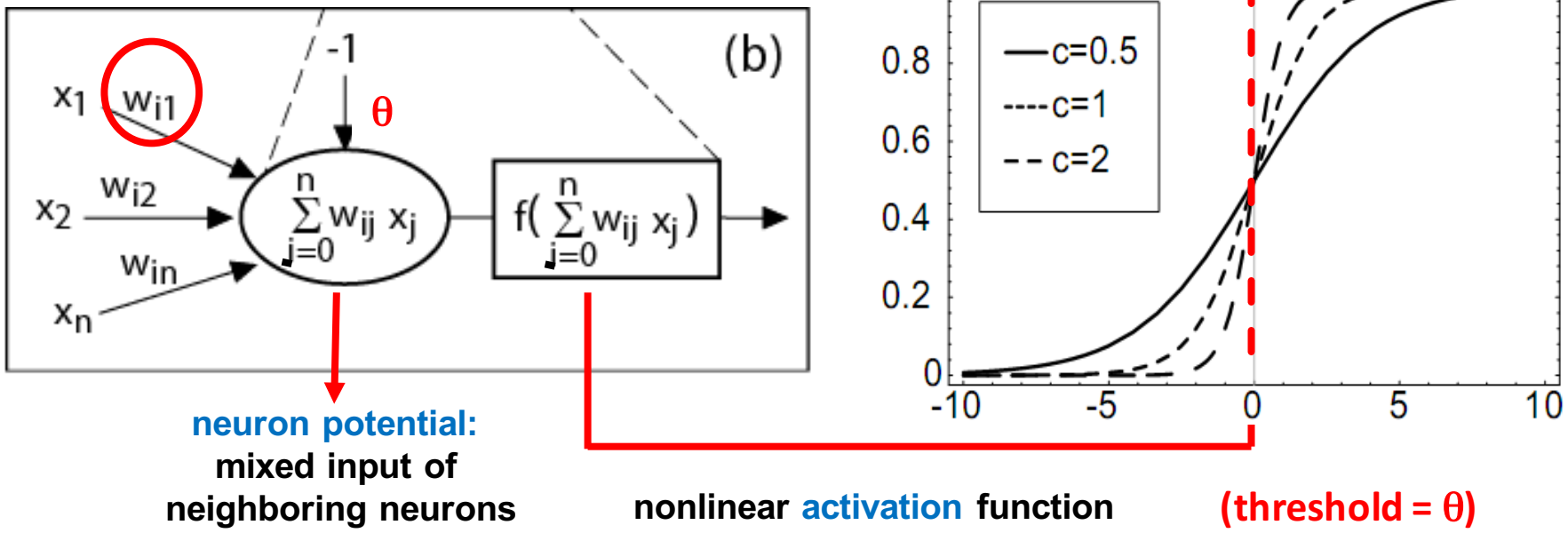
Grupo de Meteorología

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**)*.

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

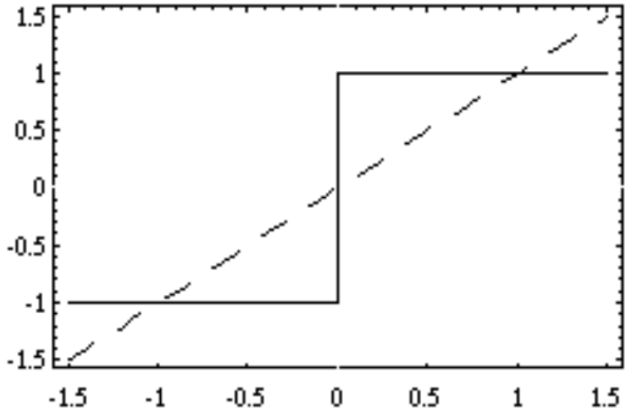
McCulloc & Pitts (1943)



- **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.

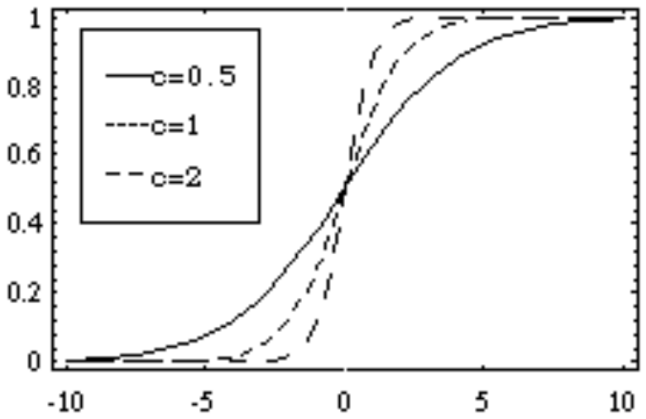
$$sgn(x) = \begin{cases} -1, \text{ si } x < 0, \\ 1, \text{ sino,} \end{cases} \quad , \quad \Theta(x) = \begin{cases} 0, \text{ si } x < 0, \\ 1, \text{ sino.} \end{cases}$$



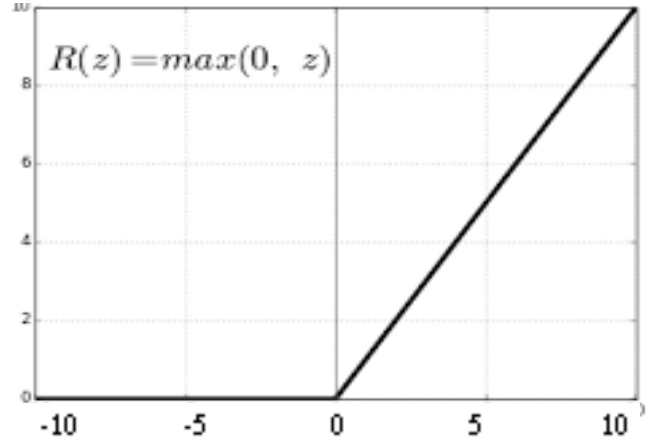
- **Funciones sigmoidales:** Funciones monótonas acotadas que dan una salida gradual no lineal.

  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 $f_c(x) = tanh(c\,x)$.

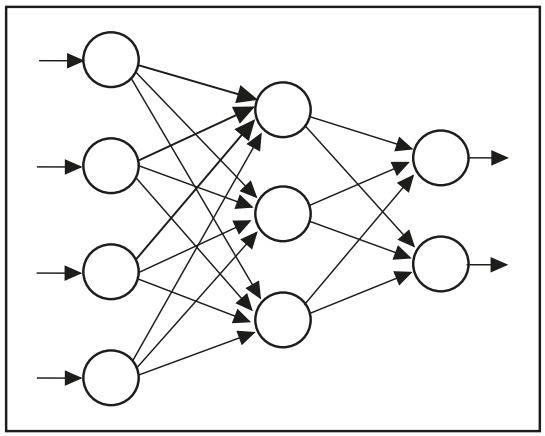


- **Rectified linear unit (ReLU):** Utilizadas para evitar el "desvanecimiento del gradiente".

TanH	$f(x) = tanh(x) = \frac{2}{1+e^2x} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	C^∞
SoftSign	$f(x) = \frac{x}{1+\|x\|}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	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)	$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$ $\lambda = 1.0507$ 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)	$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \text{ for } x < 0 \\ 1 \text{ for } x \geq 0 \end{cases}$	$[0, \infty)$	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)



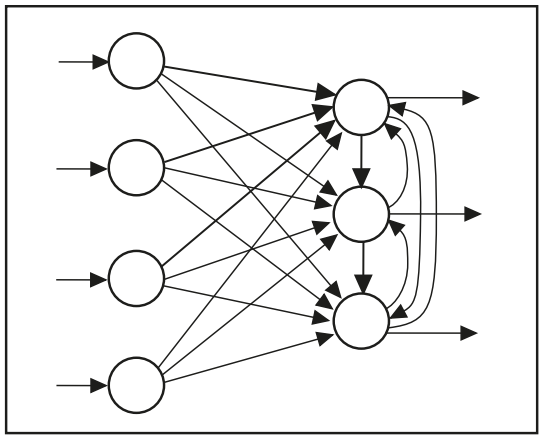
Pattern Recognition
OCR, images
Interpolation and fitting

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.

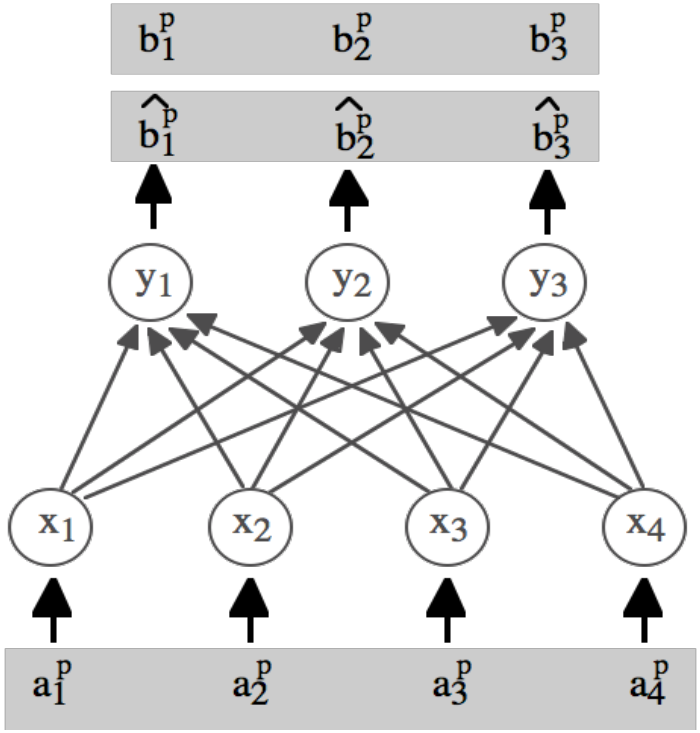


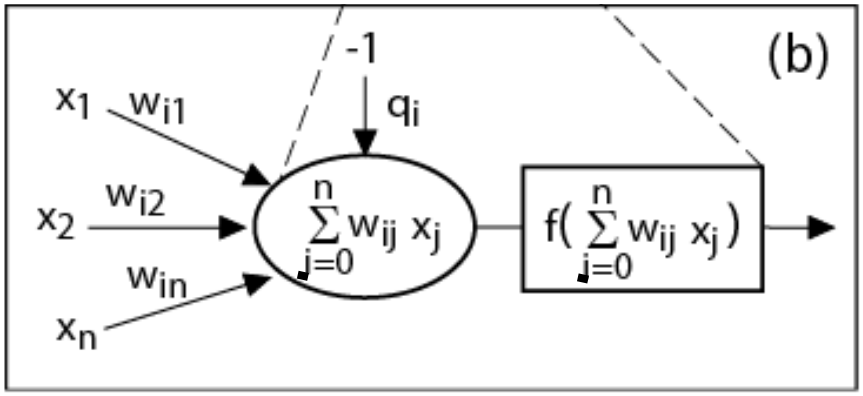
Segmentation

Feature extraction.

Prediction: Input => Clusters

Learning: Ad hoc
Winner-takes-all





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

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

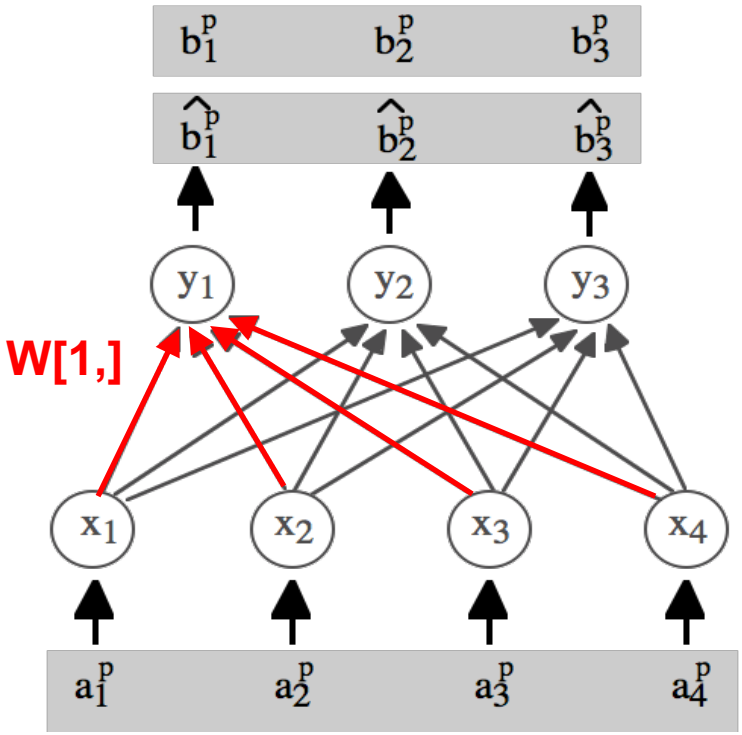
Inercia

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

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] <- runif(…)`
2. Se asigna la tasa de aprendizaje:
   `eta=0.1`
3. For i in 1:epochs
   `W[i,j] <- W[i,j] + eta*Wdelta[i,j]`

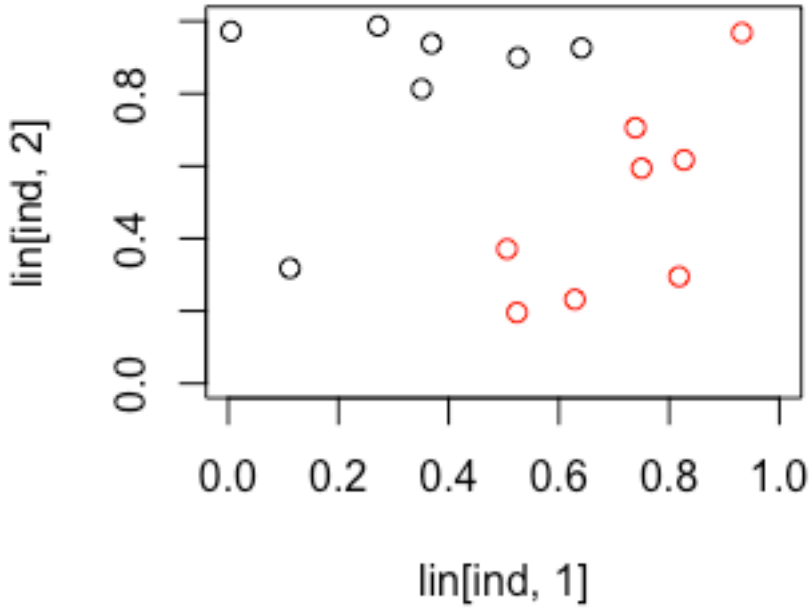
Inercia

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

Regularización

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

```
Lin <- read.csv("lineal.csv",header = F)

ind=which(lin[,3]==0)
plot(lin[ind,1],lin[ind,2],type="p",xlim=c(0,1),ylim=c(0,1))
lines(lin[-ind,1],lin[-ind,2],type="p",col="red")

a<-as.matrix(lin[,-3])
dim(a)
# [1] 15  2

b<-as.matrix(lin[,3])
dim(b)
#[1] 15  1

# Producto escalar c(1,2,3)%*%c(3,2,1)
t(b)%*%b
#8

#Incluir el bias:
a <- cbind(a,rep(1,nrow(a)))
```



```
activation <- function(z) {
       1/(1 + exp(-z))
}

neurons <- c(ncol(a),ncol(b))
W <- matrix(data = runif(prod(neurons), min = -1, max = 1),
       nrow = neurons[2], ncol = neurons[1])
```

```
        [,1]       [,2]       [,3]
[1,] -0.18798 0.11861 0.86242
```

```
bout <- activation(a %*% t(W))
```

```
> dim(yout)
#[1] 15  1
```

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

```
error <- b-bout
aux<- error*bout*(1-bout)
Wdelta <- t(aux)%*%a
```

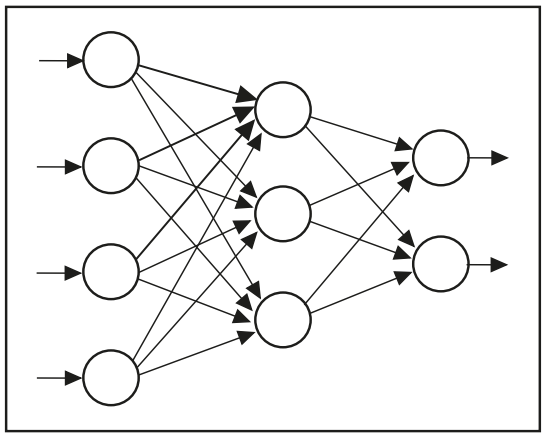
```
# [1,] -0.04360 -0.66470 -0.60538
```

```
lin<-read.csv("lineal.csv",header = F)
a<-as.matrix(lin[,-3])
b<-as.matrix(lin[,3])

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

```
backprop <- function(y,x, epochs = 10, eta = 0.1) {
### Inicializar matrices y listas

for (j in 1:epochs) {
      ### Propagar hacia delante

      ### Actualizar pesos

      ### Error output
      print(error)
}
### Return values
return(...)
}
```

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)



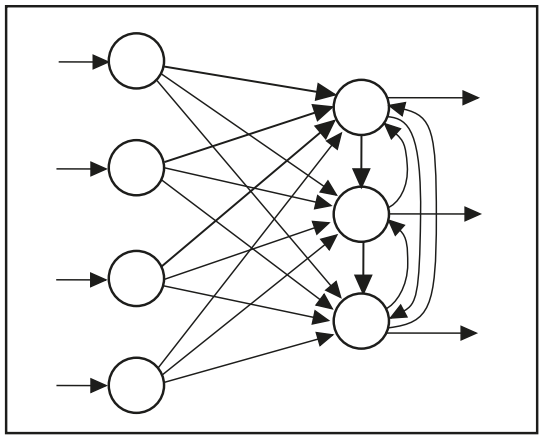
Pattern Recognition
OCR, images
Interpolation and fitting

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.



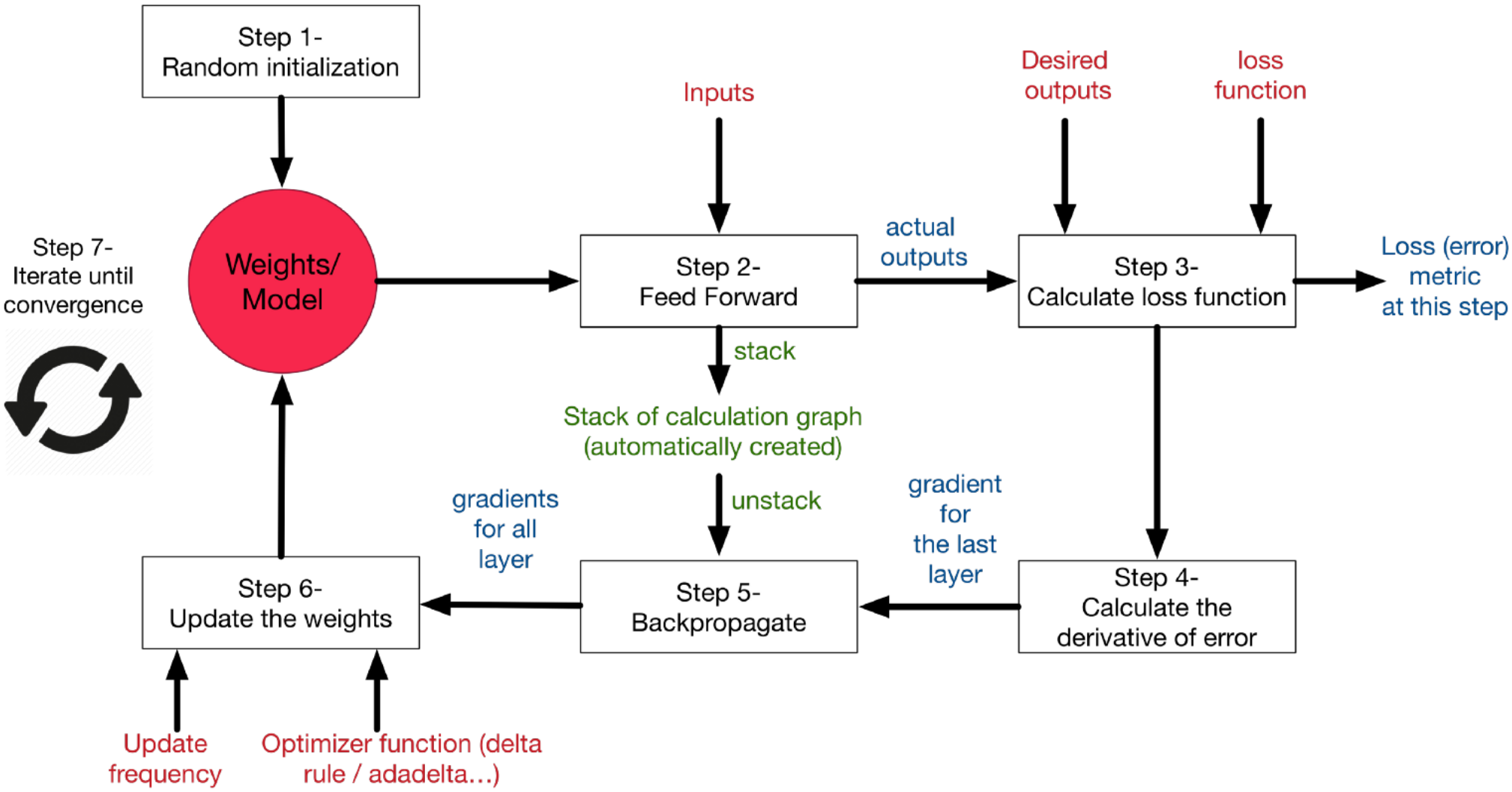
Segmentation

Feature extraction.

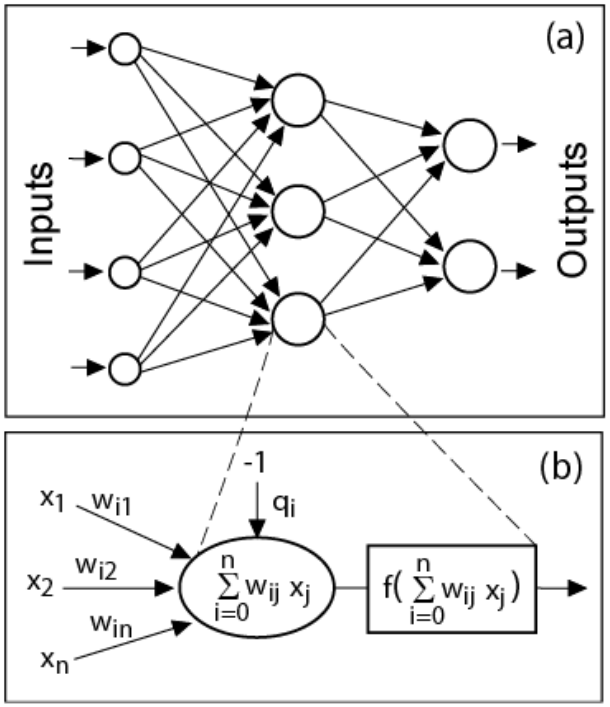
Prediction: Input => Clusters

Learning: Ad hoc
Winner-takes-all

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.

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=llg3gGewQ5U

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

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

Gradient descent

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

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

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

and update the neural weight according to these values

```
lin<-read.csv("circle.csv",header = F)
a<-as.matrix(lin[,-3])
b<-as.matrix(lin[,3])

backprop_mlp(a,b, h=5, epochs = 500, eta = 0.1)
# h: hidden neurons
```

TAREA (los BONUS 1 y 2 son opcionales):

Generalizar la función backprop anterior para que contemple la inclusión de una capa oculta. Aplicar la función al ejemplo de la clasificación circular.

Si se fija el número máximo de épocas en 1000. ¿Qué número de neuronas ocultas y qué valor de la tasa de aprendizaje (eta) es óptimo para este problema? (basta una solución aproximada).

BONUS 1- ¿Sabrías incluir un término de inercia en el método de backpropagation (ver transparencia 4)?

BONUS 2- Generalizar a un número arbitrario de capas (recomendación, definir los pesos como una lista de matrices, una para cada capa).