Neural networks

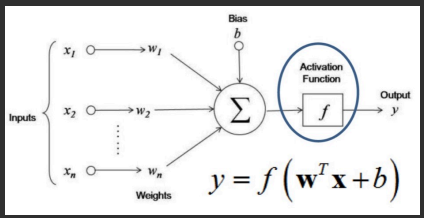
Neural Network Study (1988, AFCEA International Press, p. 60):

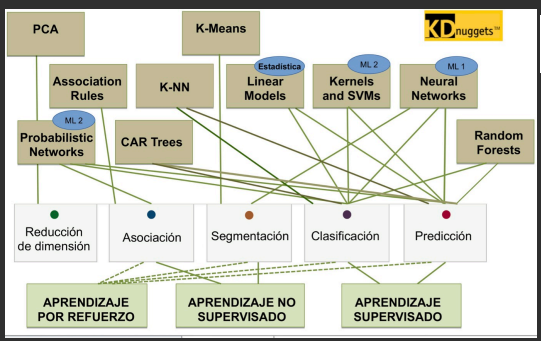
*... a neural network is a system composed of **many simple processing elements operating in parallel** whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.*

Haykin, S. (1994), Neural Networks: A Comprehen- sive Foundation, NY: Macmillan, p. 2:

A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

1. ***Knowledge is acquired through a learning process.***
2. ***Neuron weights are used to store the knowledge.***





Increasing model complexity (e.g. number of parameters) can result in **overfitting** (lack of generalization).

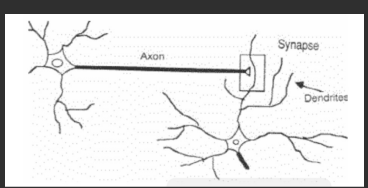
ImageNet is an image database organized according to the (nouns of the) WordNet hierarchy, in which each node of the hierarchy is depicted by an average of over five hundred images.

#synsets: 21841
#images: 14197122

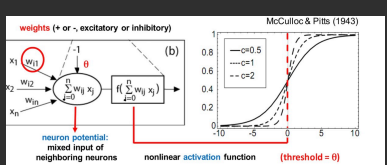
150 GB [kaggle]

Artificial Neural Networks are inspired in the structure and functioning of the **brain**, which is a collection of **interconnected neurons** (the simplest computing elements performing information processing):

- ✓Each neuron consists of a cell body, that contains a cell **nucleus**.
- ✓There are number of fibers, called **dendrites**, and a single long fiber called **axon** branching out from the cell body.
- ✓The axon connects one neuron to others (through the dendrites).
- ✓The connecting junction is called **synapse**.



- 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**)*.

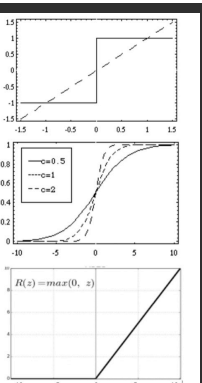


- **Funciones lineales:** $f(x) = x$.
- **Funciones paso:** Dan una salida binaria dependiente de si el valor de entrada está por encima o por debajo del valor umbral.

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

- **Funciones sigmoidales:** Funciones monótonas acotadas que dan una salida gradual no lineal.
  1. La función logística de 0 a 1:
  $$f_c(x) = \frac{1}{1 + e^{-cx}}.$$
  2. La función tangente hiperbólica de -1 a 1 $f_c(x) = tanh(c\,x)$.

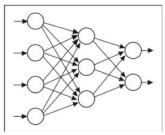
- **Rectified linear unit (ReLU):** Utilizadas para evitar el "desvanecimiento del gradiente".



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	$f(\alpha, x) = \begin{cases} -\frac{\ln(1-\alpha(x+\alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^x-1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1-\alpha(\alpha+x)} & \text{for } \alpha < 0 \\ e^{\alpha x} & \text{for } \alpha \geq 0 \end{cases}$	$(-\infty, \infty)$	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 (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
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).

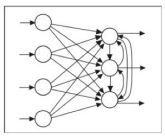
***Multilayer* Networks or Feedforward Nets.**
Several layers connected (input+hidden+output)



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.

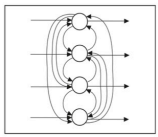
***Competitive* Networks**
Multilayer networks with lateral connections (competitive) in the last layer.



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+ε).

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



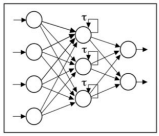
Pattern Recognition
OCR, images
Memories (robust to noise)
Prediction: Input => Input
Learning: Hegg

Autoencoders **(later)**

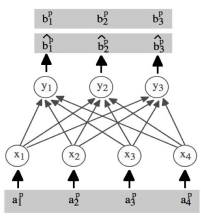
Feature extraction, compression.

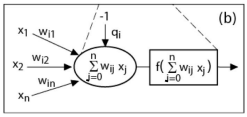
Supervised Problems (with memory). **Input-Output pairs are provided:**
(x1,y1), (x2,y2), ..., (xn,yn) and the network learns $y_t = f(x_{t-1,t-2,...}+\varepsilon)$.

***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





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

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

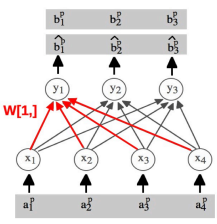
Inercia

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

Regularización

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

RSNNS



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]`

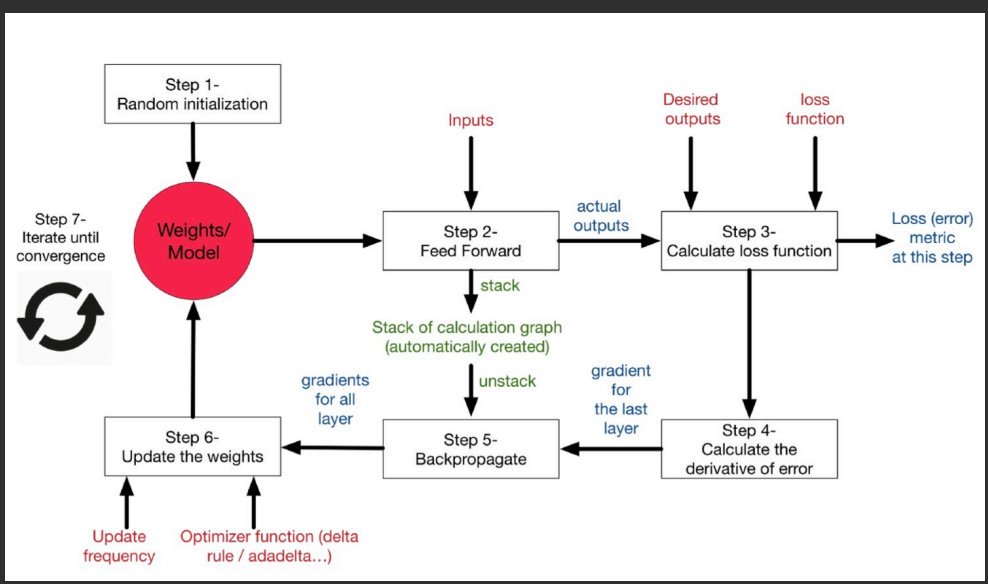
$$E(w) = \frac{1}{2} \sum_{i,p} (b_i^p - \hat{b}_i^p)^2.$$

Inercia

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

Regularización

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



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 $\partial E_x / \partial b$

$$w_t^l = w_{t-1}^l - \eta \partial E / \partial w^l$$

$$b_t^l = b_{t-1}^l - \eta \partial E / \partial b^l$$

Back Propagation

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

$$\delta^L = (a^L - y) \odot \sigma'(A^L)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(A^l)$$

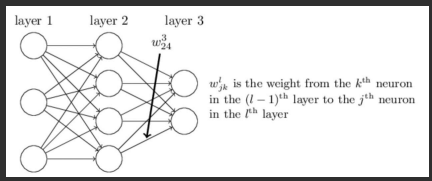
$$\partial E / \partial w^l = \delta^l a^{l-1}$$

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

Back propagation: compute $\frac{\partial C}{\partial w}$, $\frac{\partial C}{\partial b}$ where w and b are all weights and biases of the network. C is the cost function.

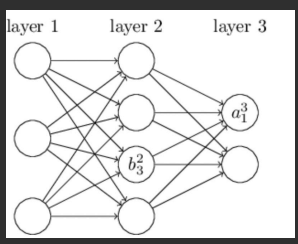
We start defining:

W^l_{jk} := weight for the connection from k^{th} neuron in $(l-1)^{th}$ layer to j^{th} neuron in l^{th} layer



b^l_j := bias j^{th} neuron of the l^{th} layer

a^l_j := activation j^{th} neuron of the l^{th} layer



N_l := number of nodes in the l^{th} layer

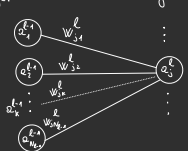
L := Number of layers

m := Number of training examples

Compute a^l_j

— Using numbers

layer $l-1$ (N_{l-1} nodes) layer l



$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

— Using matrices

$$a^l := \begin{bmatrix} a^l_1 \\ a^l_2 \\ \vdots \\ a^l_{N_l} \end{bmatrix} \quad b^l := \begin{bmatrix} b^l_1 \\ b^l_2 \\ \vdots \\ b^l_{N_l} \end{bmatrix} \quad W^l := \begin{bmatrix} w^l_{11} & w^l_{12} & \cdots & w^l_{1N_{l-1}} \\ w^l_{21} & & & \vdots \\ \vdots & & \ddots & \vdots \\ w^l_{N_l 1} & \cdots & & w^l_{N_l N_{l-1}} \end{bmatrix}$$

OSS
- Size of W^l is $N_l \times N_{l-1}$
- in the example above $w^l_{j1}\ w^l_{j2} \dots w^l_{jN_{l-1}}$ is the j^{th} row of the matrix W^l

$$\Longrightarrow \quad a^l = \sigma(w^l a^{l-1} + b^l)$$

We introduce

$$z^l := w^l a^{l-1} + b^l \quad , \quad z^l = \begin{bmatrix} z^l_1 \\ z^l_2 \\ \vdots \\ z^l_{N_l} \end{bmatrix}$$ where the generic $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$

so $a^l = \sigma(z^l)$

The cost function

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

, $y(x) :=$ desired output for the example x

$a^L(x) :=$ activation last layer using the example x

One assumption of C is that it can be written as average of other cost function for each sample

In fact we can also write C as:

$C = \frac{1}{n} \sum_x C_x$, where $C_x = \frac{1}{2}\|y - a^L\|^2$ is the cost function for the example x

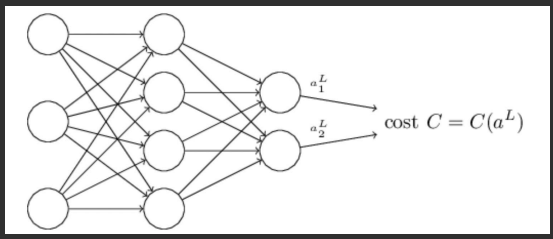
This assumption is important because backpropagation let us compute $\frac{\partial C_x}{\partial w}, \frac{\partial C_x}{\partial b}$

We recover $\frac{\partial C}{\partial w}, \frac{\partial C}{\partial b}$ by averaging over training examples.

IMPORTANT NOTION !!

We drop C_x and we use C, but we are still working on training example x

Another assumption is that C can be written in function of the outputs of the neural network



$$C = \frac{1}{2}\|y - a^L\|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

Hadamard product

$$\begin{bmatrix}1\\2\end{bmatrix} \odot \begin{bmatrix}3\\4\end{bmatrix} = \begin{bmatrix}1*3\\2*4\end{bmatrix} = \begin{bmatrix}3\\8\end{bmatrix}$$

The four Fundamental equations

— $\delta^l = \begin{bmatrix} \frac{\partial C}{\partial z_1^l} \\ \frac{\partial C}{\partial z_2^l} \\ \vdots \\ \frac{\partial C}{\partial z_{n_l}^l} \end{bmatrix}$, $\delta_j^l = \frac{\partial C}{\partial z_j^l}$ error of the j^{th} neuron in the l^{th} layer

this quantity is important because the error of each neuron propagates causing the overall cost to change

An equation for the error in the output layer, δ^L

$$\delta^L_j = \frac{\partial C}{\partial a^L_j}\sigma'(z^L_j)$$

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

where $\nabla_a C = \begin{bmatrix} \frac{\partial C}{\partial a^L_1} \\ \frac{\partial C}{\partial a^L_2} \\ \vdots \\ \frac{\partial C}{\partial a^L_n} \end{bmatrix}$

For example, using a quadratic cost $C = \frac{1}{2}(a^L - y)^2$ $\quad \delta^L = (a^L - y) \odot \sigma'(z^L)$

An equation for the error δ^l in terms of the error in the next layer, δ^{l+1}: In particular

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

where $(w^{l+1})^T$ is the transpose of the weight matrix w^{l+1} for the $(l+1)^{\text{th}}$ layer. This equation appears complicated, but each element has a nice interpretation. Suppose we know the error δ^{l+1} at the $l+1^{\text{th}}$ layer. When we apply the transpose weight matrix, $(w^{l+1})^T$, we can think intuitively of this as moving the error *backward* through the network, giving us some sort of measure of the error at the output of the l^{th} layer. We then take the Hadamard product $\odot\sigma'(z^l)$. This moves the error backward through the activation function in layer l, giving us the error δ^l in the weighted input to layer l.

An equation for the rate of change of the cost with respect to any bias in the network: In particular:

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j$$, that we already calculated

To make notation lighter:

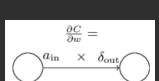
$$\frac{\partial C}{\partial b} = \delta$$, of course we evaluate δ on the same neuron as the bias b

An equation for the rate of change of the cost with respect to any weight in the network: In particular:

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$$

More intuitively

$$\frac{\partial C}{\partial w} = a_{\text{in}}\delta_{\text{out}}$$



The backpropagation algorithm

1. **Input x:** Set the corresponding activation a^1 for the input layer.
2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.
3. **Output error δ^L:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.
4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.
5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.