Ciencia de Datos

2025

Introducción a la Ciencia de Datos

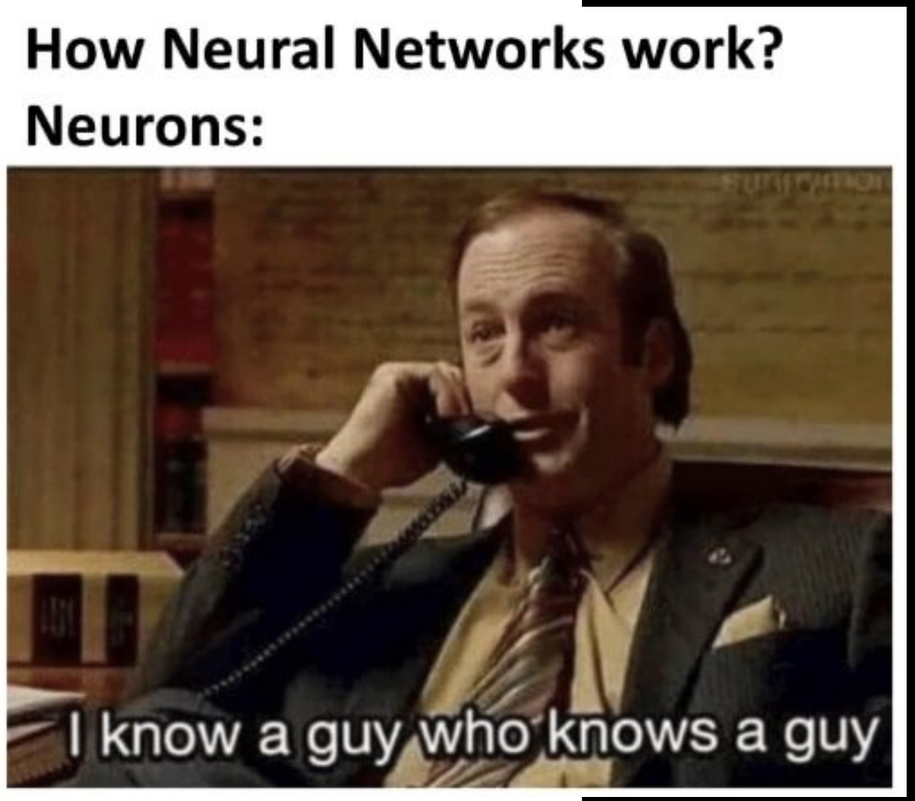
OPTATIVA - LICENCIATURA EN INFORMÁTICA
FACET-UNT

Ciencia de Datos: Fundamentos y Herramientas

CURSO DE POSTGRADO - FACET-UNT

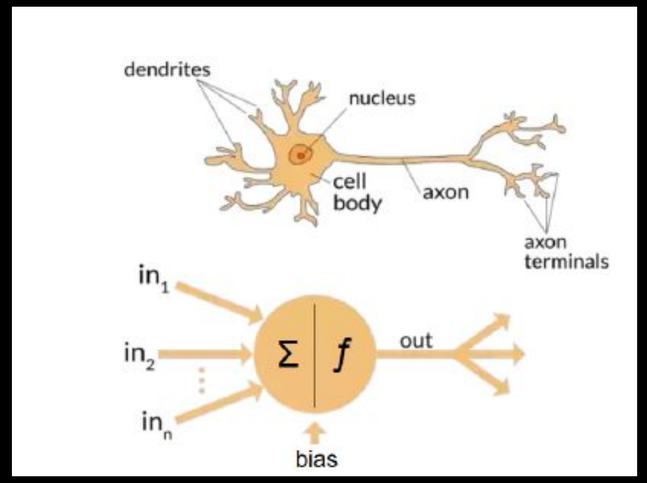
CD2023

Artificial Neural Networks
ANN
NN

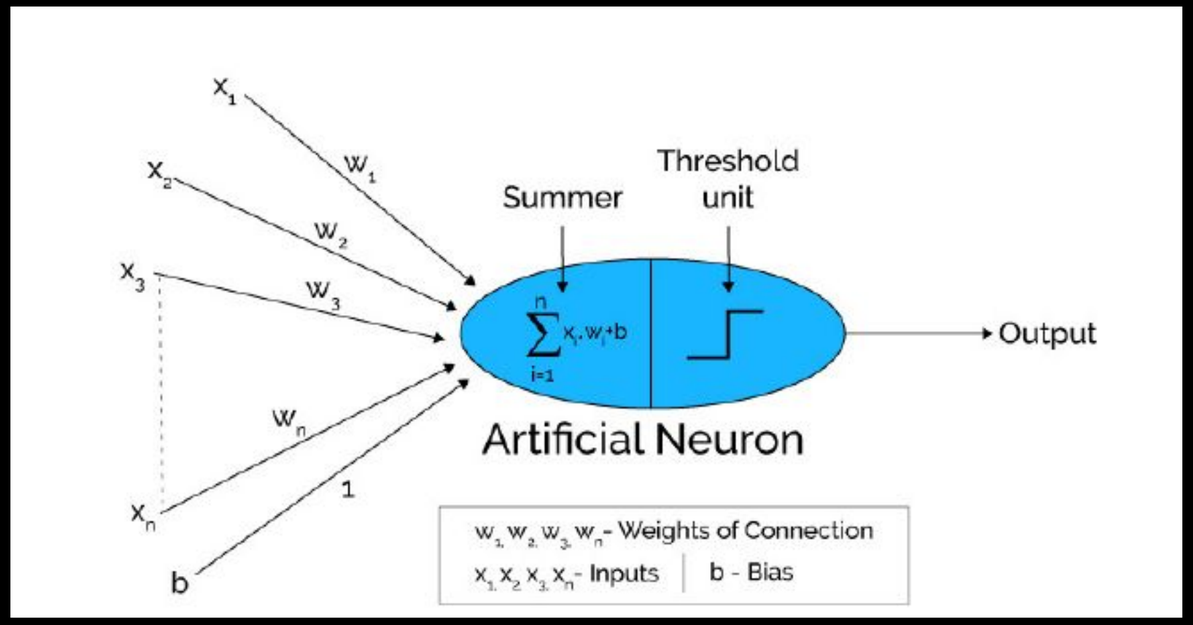


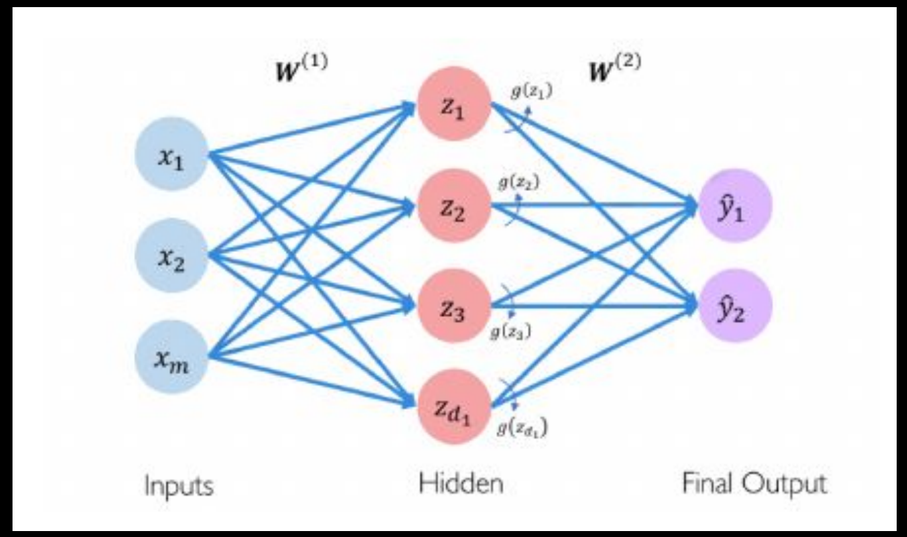
Redes Neuronales Artificiales
RNA

ANN



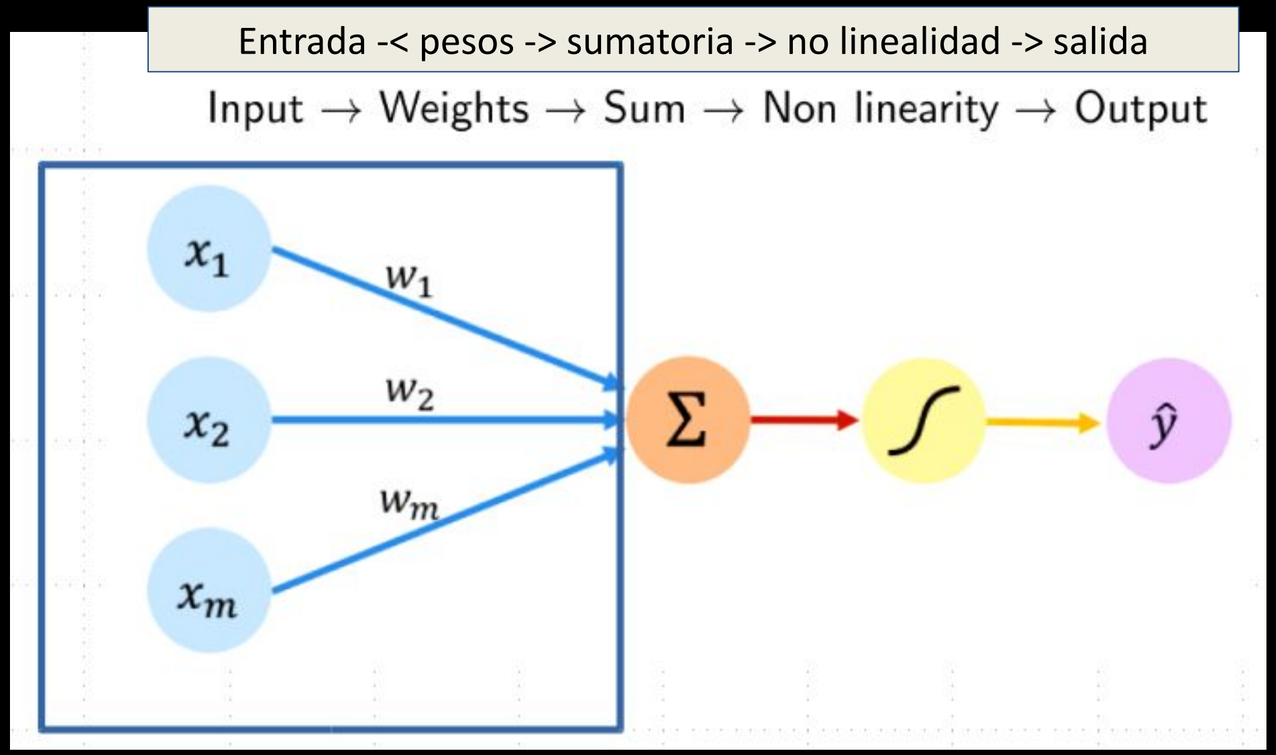
- **Data-driven modeling / modelado basado en datos > pobre capacidad de generalización**
- **Inspirado en las redes neuronales del cerebro**
- **Para resolver un gran número de problemas muy complejos: reconocimiento facial, reconocimiento de la escritura, pronostico del clima, etc.**
- **Black-box modelling (?) > basado en la composición de neuronas interconectadas > dificil entender las predicciones**





Perceptron

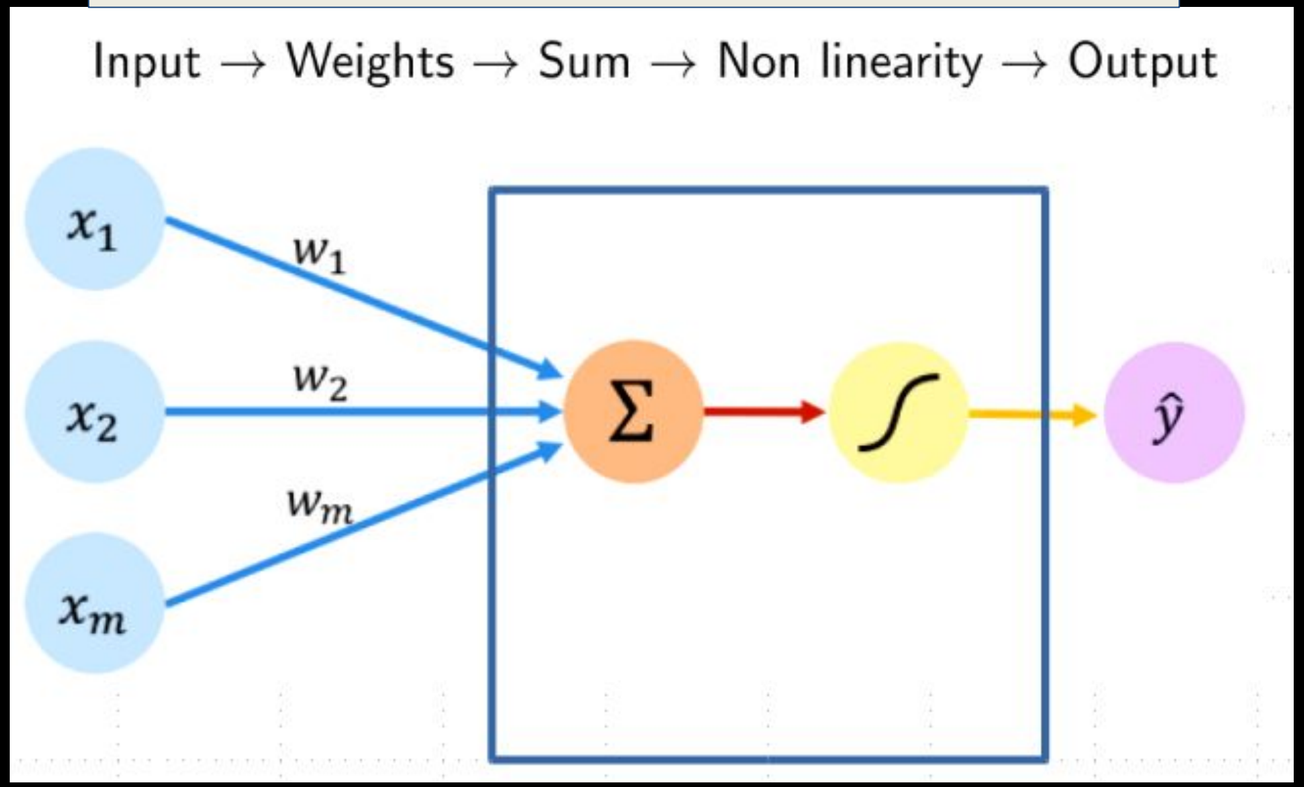
- Unidad fundamental de una red neuronal

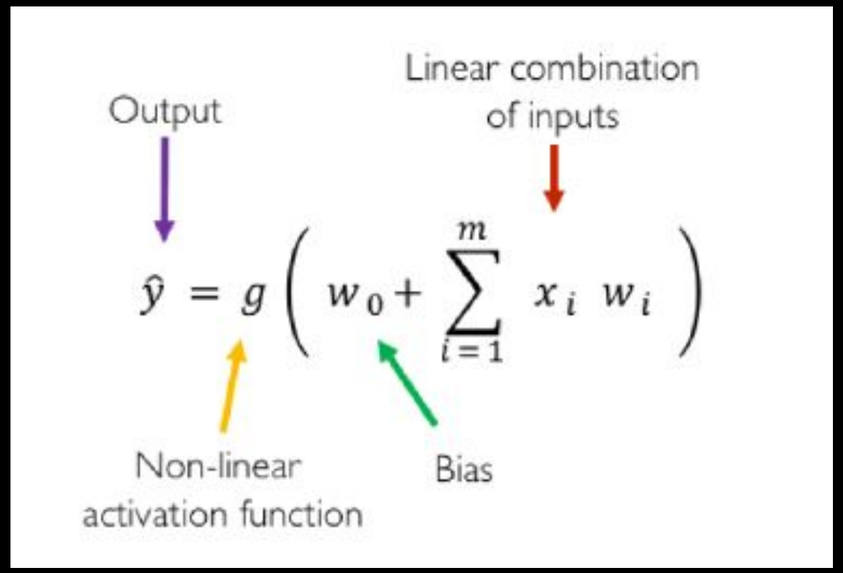


- cada entrada = una ‘feature’
- se aplica un peso(w) a cada entrada (xi)
- los pesos son inicializados y actualizados de manera aleatoria durante el entrenamiento

Perceptron

Entrada -< pesos -> sumatoria -> no linealidad -> salida

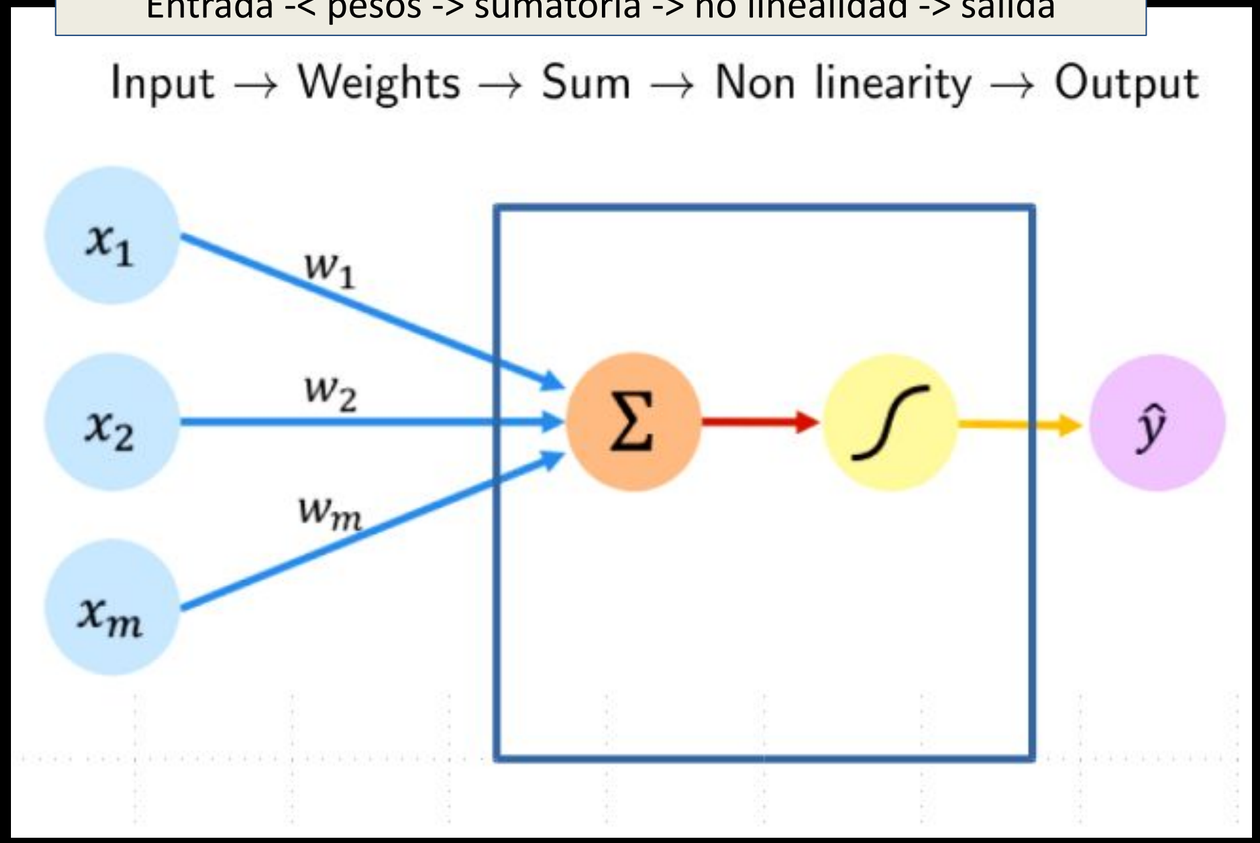




- capa oculta o ‘ hidden layer’
- Bias/sesgo: termino de ajuste (muchas veces seteado en 1)
- la función de activación es aplicada a la combinación lineal de las entradas pesadas (+ el bias)

Perceptron

Entrada -< pesos -> sumatoria -> no linealidad -> salida



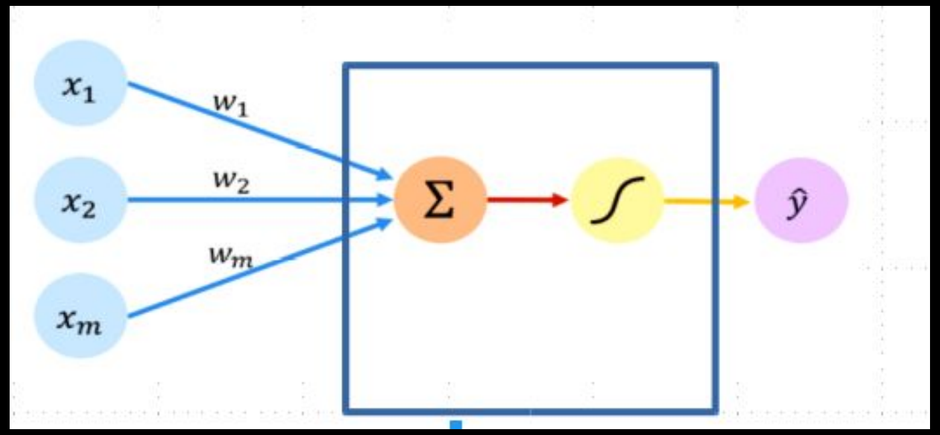
$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W}\right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

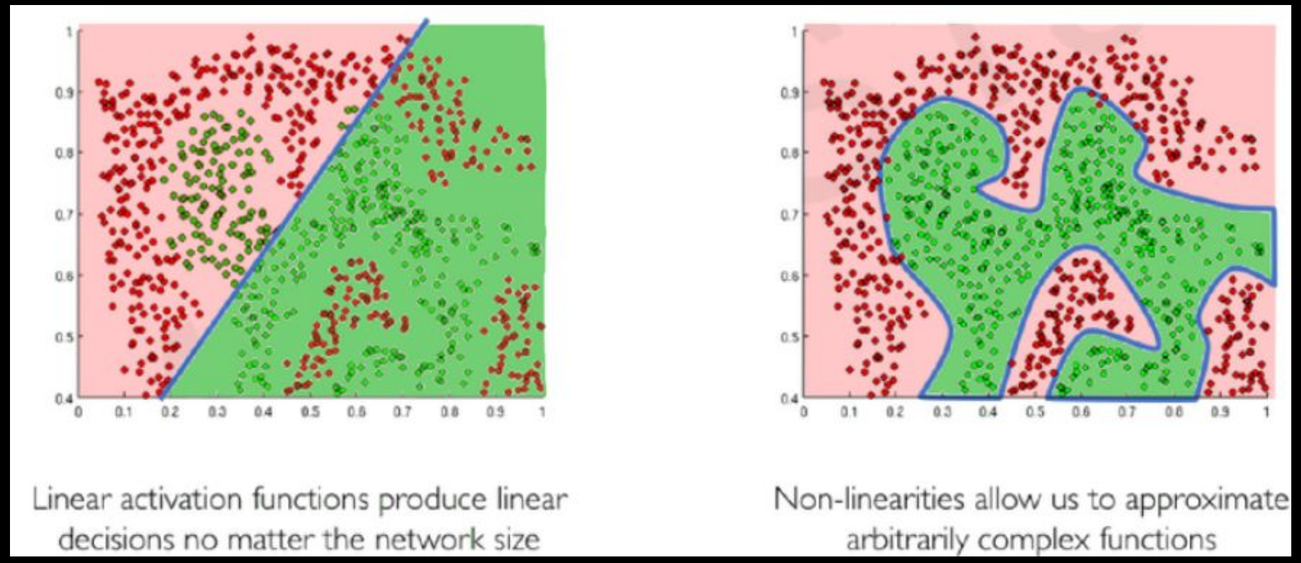
Funcion de activacion

- Provee de la capacidad de agregar no-linealidad al modelo

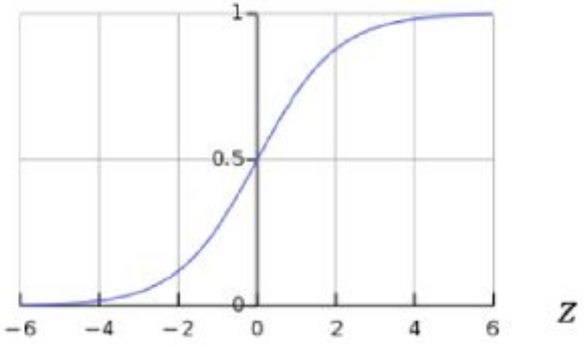


$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

- Una de las más usadas: funcion sigmoide que produce valores entre 0 y 1



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

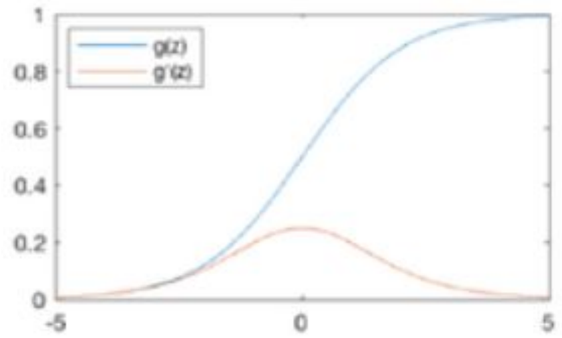


Funcion de activacion

- vamos a ver que no solo la función es importante sino también su derivada
- Es una configuración (hiperparametro) que se debe seleccionar ante del entrenamiento.

$$\hat{y} = g\left(w_0 + \boldsymbol{X}^T \boldsymbol{W} \right)$$

Sigmoid Function

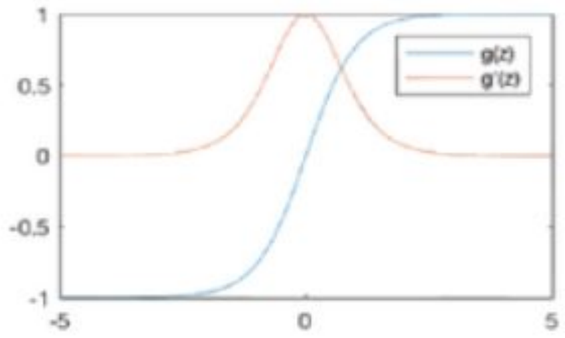


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

`tf.math.sigmoid(z)`

Hyperbolic Tangent

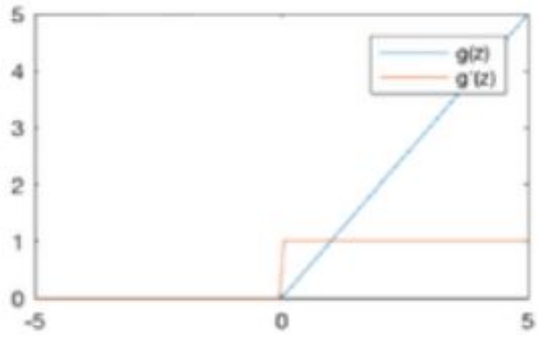


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

`tf.math.tanh(z)`

Rectified Linear Unit (ReLU)



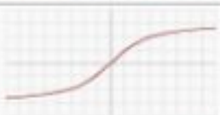
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

`tf.nn.relu(z)`

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

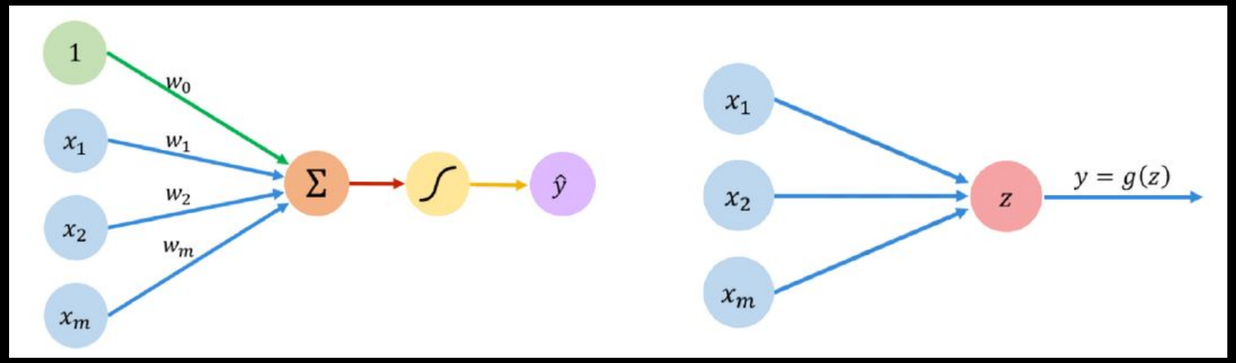
- Elegir una funcion de activacion acorde al problema y a los datos es crucial para que el modelo tenga una buena performance

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

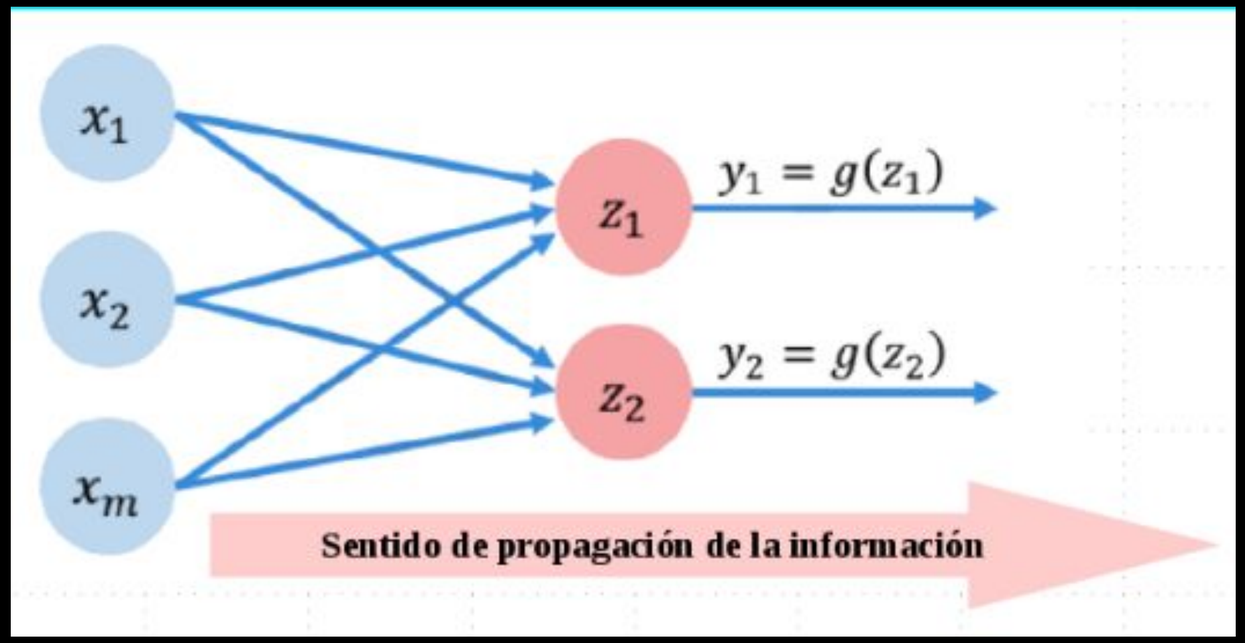
Feed Forward (FF)

- Perceptron simple
  (n inputs → 1 output)

$$z = w_0 + \sum_{j=1}^{m} x_j \, w_j$$



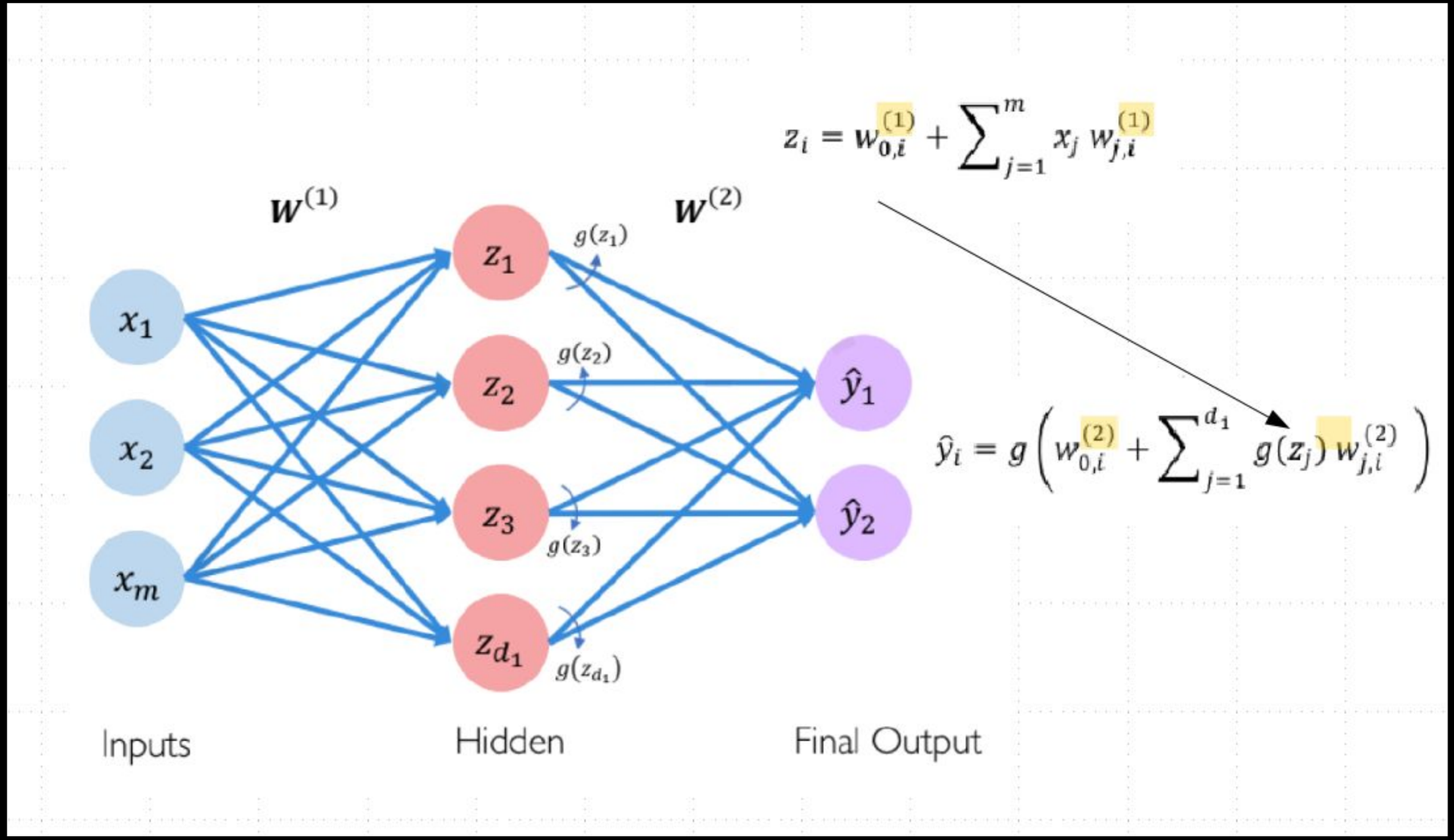
- Neural Network > compuesta por multiples perceptrones (n inputs → m salida)



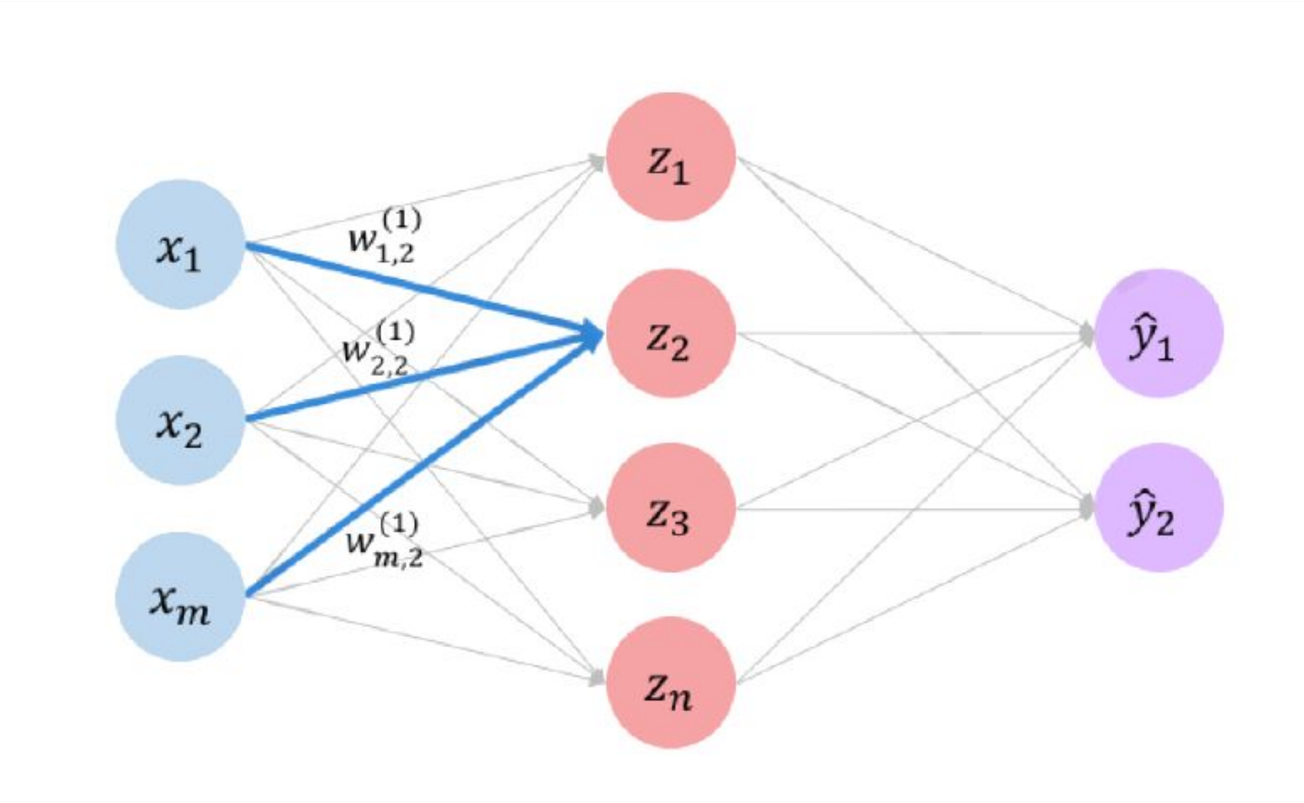
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

CAPA DENSA: cada entrada está conectada a todas las salidas

1- Red neuronal de una capa



1- Red neuronal de una capa

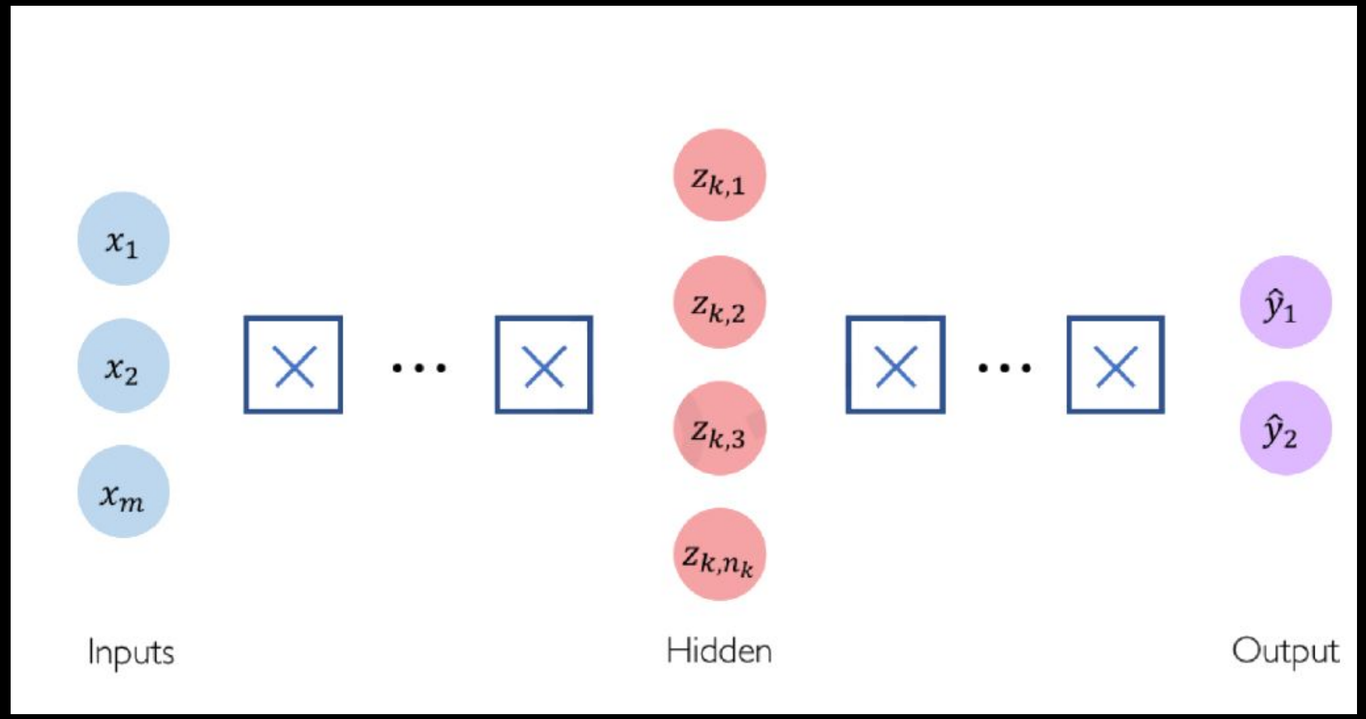


$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

M-layers Neural Network (red neuronal multi-capa)

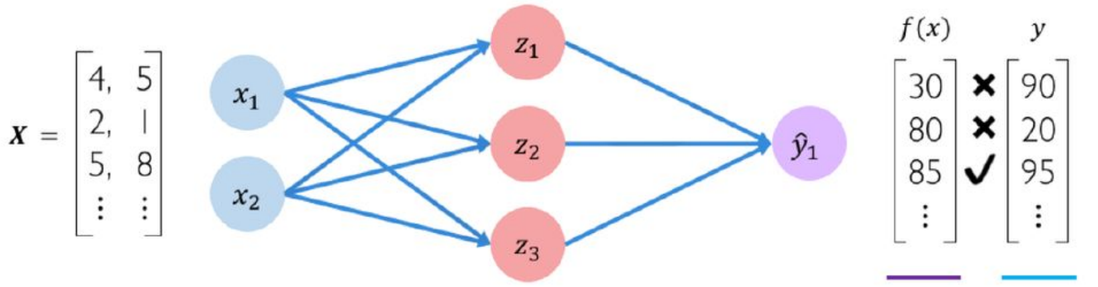
- Deep Neural Network (red neuronal profunda)



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Loss Function / Funcion de perdida

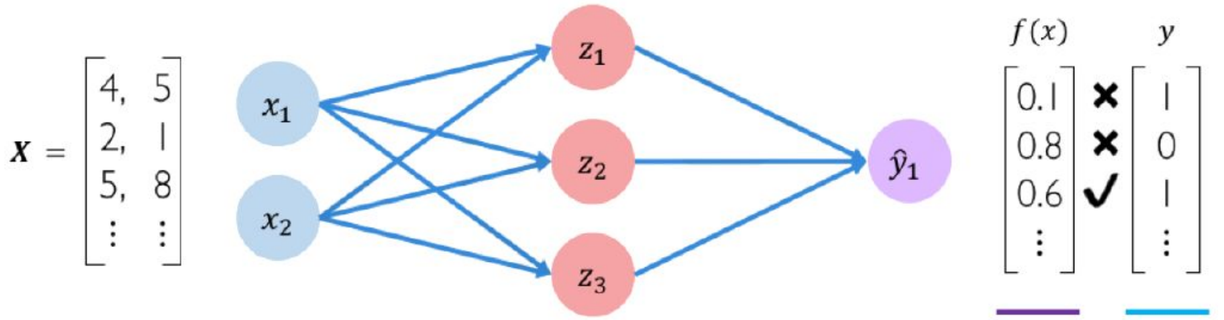
Mean squared error loss can be used with regression models that output continuous real numbers



$$J(W) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; W\right)\right)^2$$

Actual Predicted

Cross entropy loss can be used with models that output a probability between 0 and 1



$$J(W) = \frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log\left(f\left(x^{(i)}; W\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - f\left(x^{(i)}; W\right)\right)$$

Actual Predicted Actual Predicted

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y} - \mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y} - \mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y} - \sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y} - \sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \|\sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)}\|$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
log^2	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2^2 + \|\mathbf{y}\|_2^2 - \sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log \frac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2 \|\mathbf{y}\|_2}$

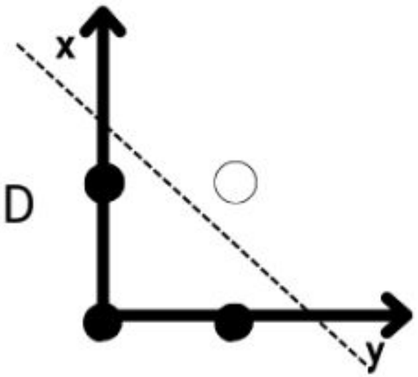
- https://arxiv.org/pdf/1702.05659.pdfs

Entrenemos un perceptron

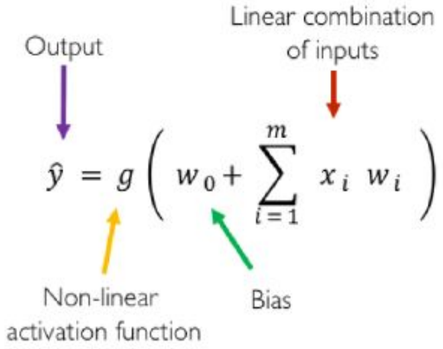
X1	X2	Y
F	F	0
F	T	0
T	F	0
T	T	1

test set

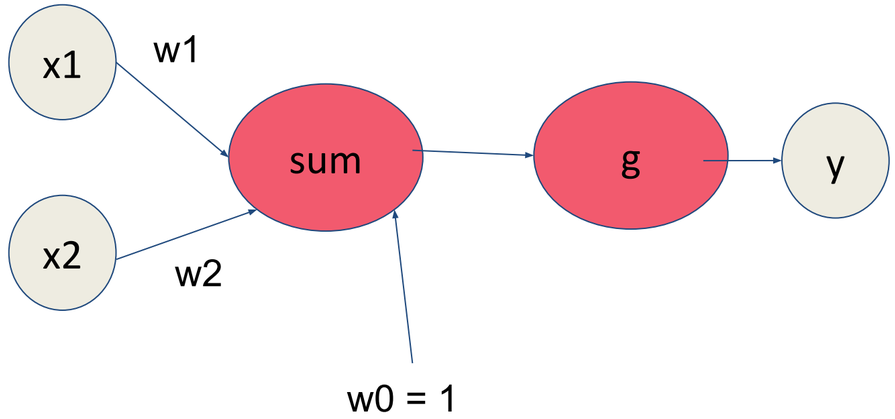
AND



Recordemos



Nuestra arquitectura sería :



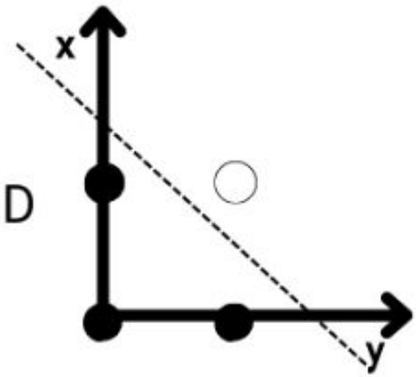
Cuanto vale g ?

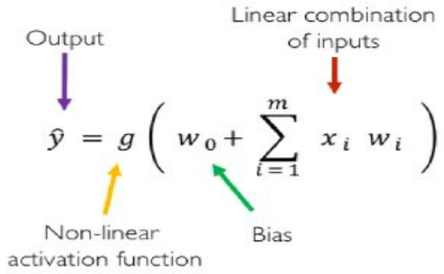
Supongamos que g devuelve 1 si es >1.5 y devuelve 0 en caso contrario

Entrenemos un perceptron

X1	X2	Y
F	F	0
F	T	0
T	F	0
T	T	1

test set





Comienza el entrenamiento It =1

se obtienen de manera aleatoria los valores de los pesos.

Supongamos w1=0.9 y w2=0.9

Entonces.

y1 = g(0,9 *0 +0,9 *0 +1) = g(1) = 1 pero y(0,0) = 0

e =abs(0-0)=0

y2 = g(0.9*1+0.9*0+1)=g(1.9) = 1 pero y(1,0) = 0

e =abs (1.9-0)=1.9

y3 = g(0.9*1+0.9*1+1) = g(2.8) = 1 y ademas y(1,1) = 1

e =0

Hasta aquí el modelo generado tiene la siguiente forma:

yaprox = g(0.9 x1 + 0.9 x2 +1)

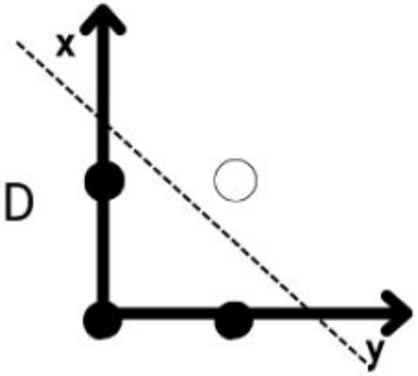
Como seria la prediccion del test set?

y = g(0.9*1 + 0.9*0+1) =g(1.9) = 1 pero deberia ser 0!!!!

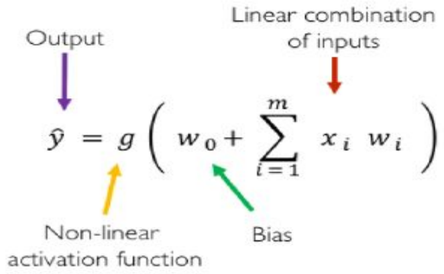
Entrenemos un perceptron

X1	X2	Y
F	F	0
F	T	0
T	F	0
T	T	1

AND



test set



Seguimos entrenando, It = 2

se obtienen de manera aleatoria los valores de los pesos.

Supongamos w1=0.4 y w2=0.4

Entonces:

y1 = g(0,4**0 +0,4 *0 +1) = g(1) = 1 pero y(0,0) = 0

e =abs(0-0)=0

y2 = g(0.4*1+0.4*0+1)=g(1.4) = 0 pero y(1,0) = 0

e =abs (0-0)=0

y3 = g(0.9*1+0.9*1+1) = g(2.8) = 1 y ademas y(1,1) = 1

e =0

Hasta aquí el modelo generado tiene la siguiente forma:

yaprox = g(0.4 x1 + 0.4 x2 +1)

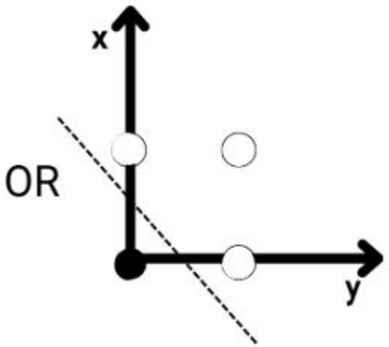
Como seria la prediccion del test set?

y = g(0.4*0+0.4*1+1) = g(1.4) = 0 y el resultado debía ser o!!! :)

Entrenemos un perceptron

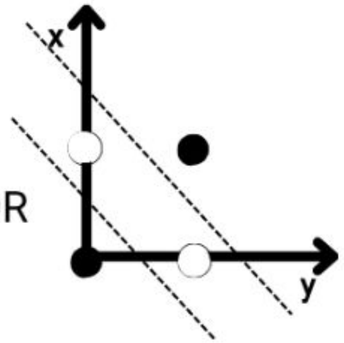
X1	X2	Y
F	F	0
F	T	1
T	F	1
T	T	1

OR



X1	X2	Y
F	F	0
F	T	1
T	F	1
T	T	0

XOR



pensar estos dos casos!

Training (optimización de la función de pérdida)

- encontrar los valores de los pesos para que la función de pérdida (loss) o costo sea la minima posible
- Problema de optimización

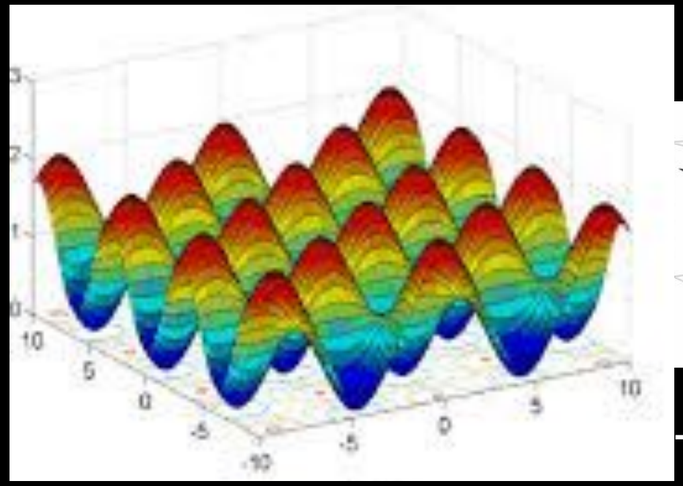
$$W^* = \underset{W}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

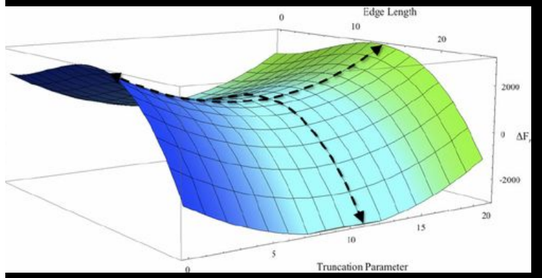
$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

METODO DEL GRADIENTE DESCENDIENTE

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights

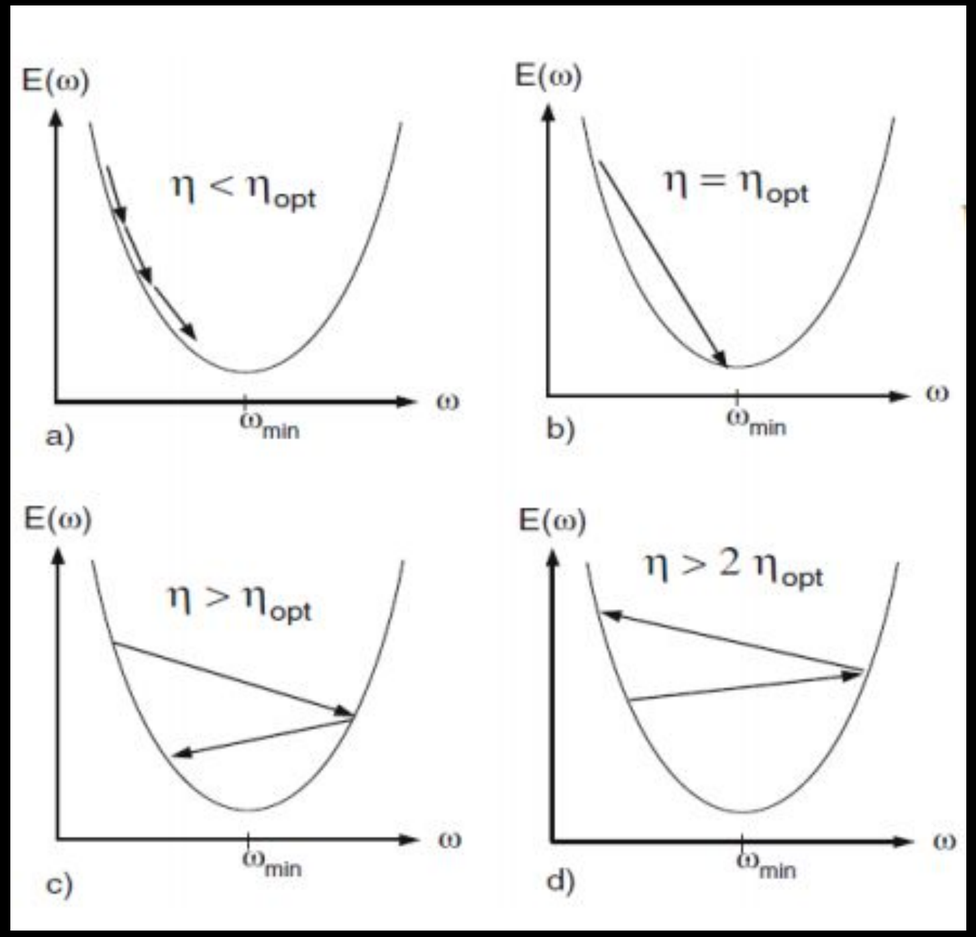




Tiene algunos problemas:

- Minimos locales minima
- Silla de montar: cuando llega a una región de gradiente cero, no puede escapar

Training (optimización de la función de pérdida)



$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

How can we set the learning rate?

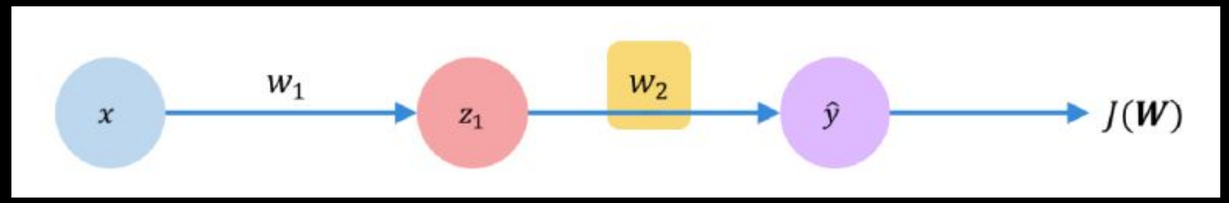
- https://ruder.io/optimizing-gradient-descent/index.html

Gradient Descent Algorithms

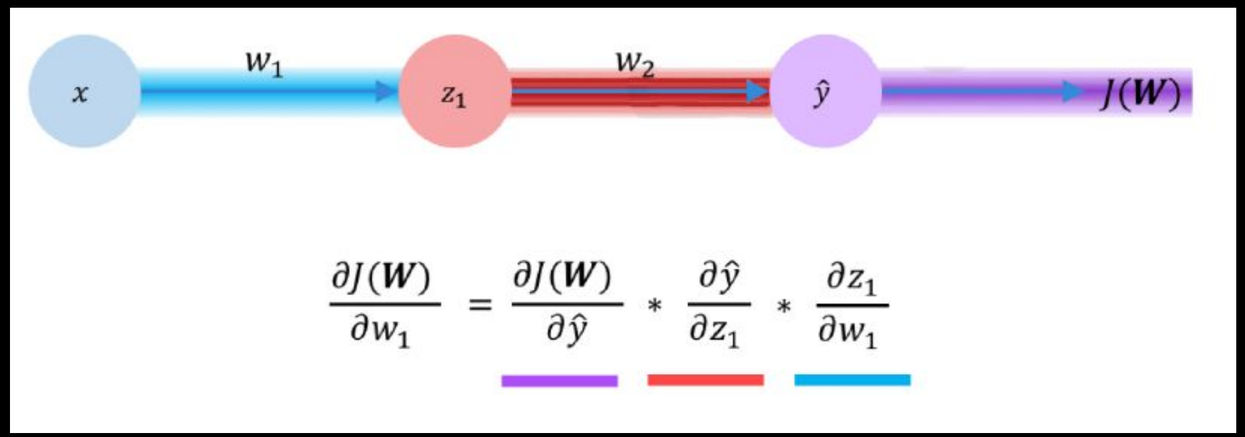
Algorithm	TF Implementation	Reference
• SGD	tf.keras.optimizers.SGD	Kiefer & Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function." 1952.
• Adam	tf.keras.optimizers.Adam	Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.
• Adadelta	tf.keras.optimizers.Adadelta	Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.
• Adagrad	tf.keras.optimizers.Adagrad	Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.
• RMSProp	tf.keras.optimizers.RMSProp	

Backpropagation

- Que tanto afectan los cambios pequeños en uno o más pesos?



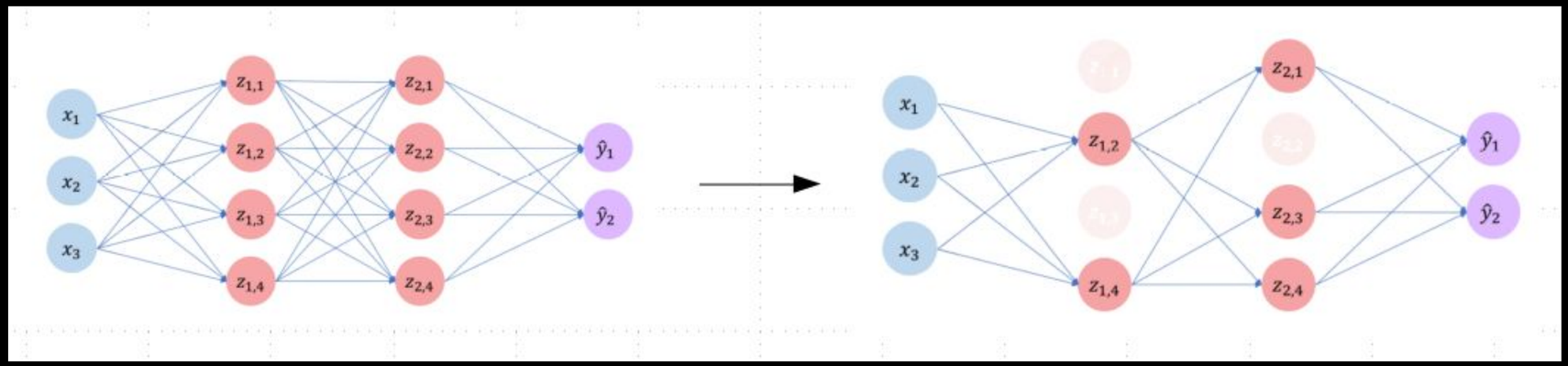
- Se aplica la regla de la cadena



- Repite para cada uno de los pesos de la red usando la forma del gradiente de la siguiente capa!

Regularizacion

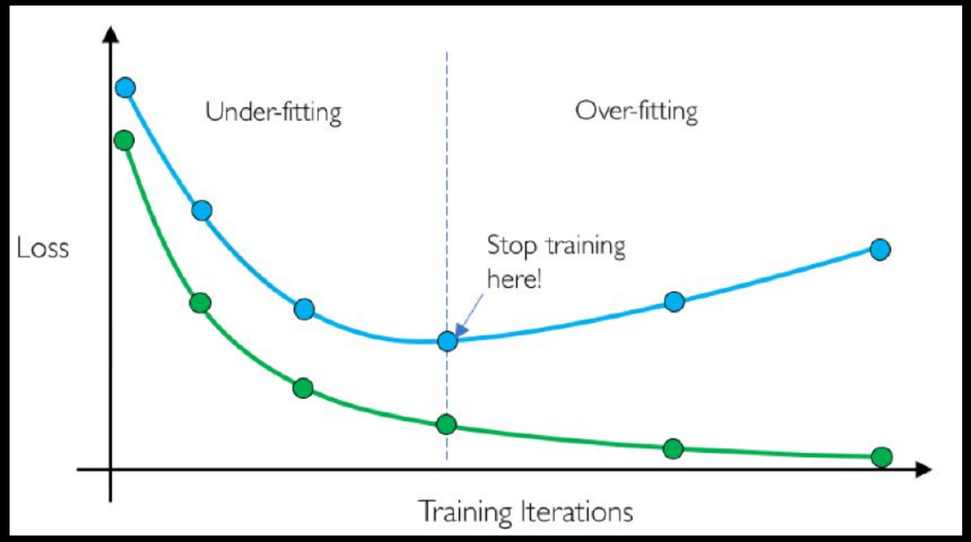
- Técnica que agrega restricciones al problema de optimizacion (evita modelos complejos)
- Busca lograr modelos mas generalesl



DROP OUT

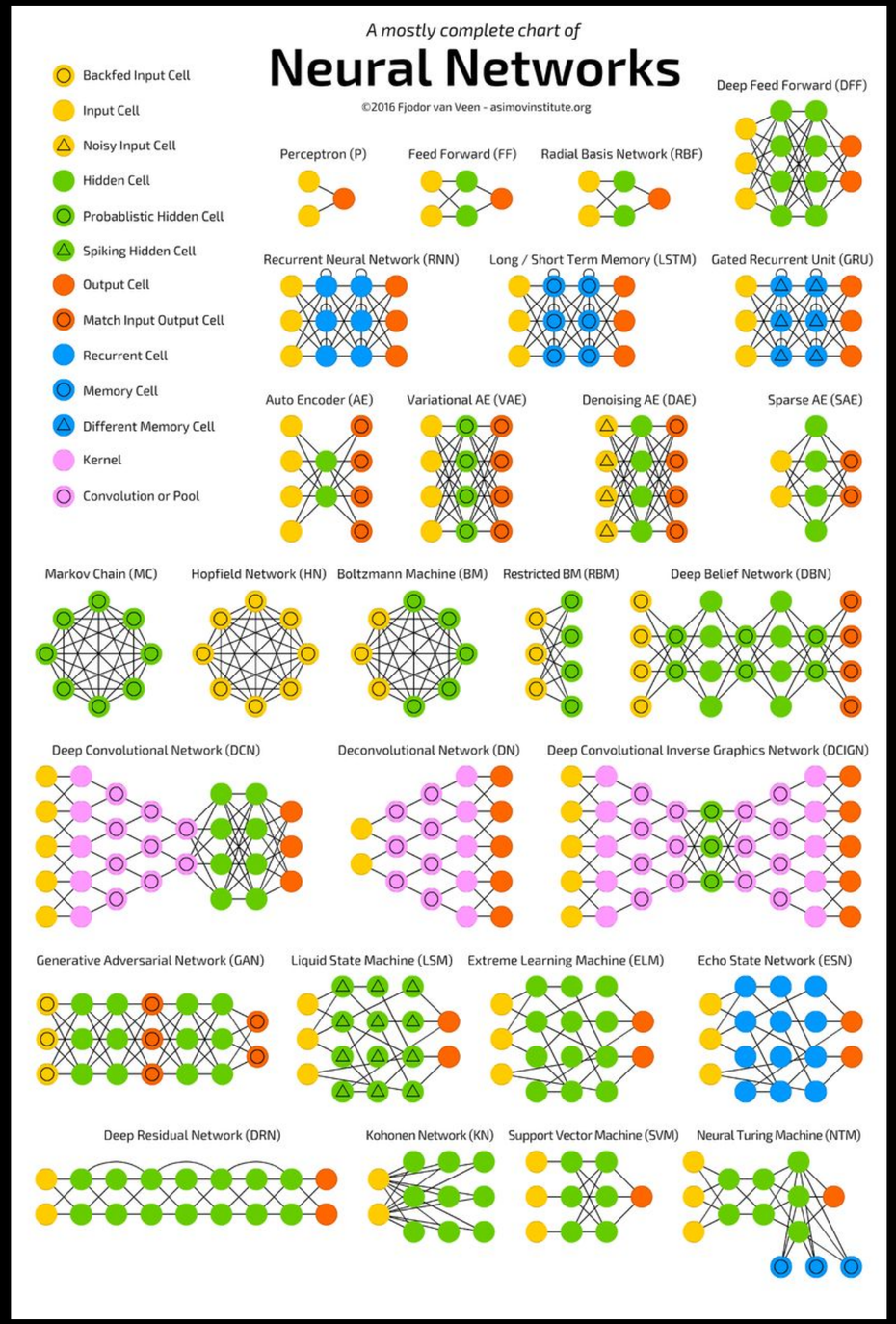
- Desactiva algunas neuronas (aleatoriamente) durante el entrenamiento
- << overfitting

blue = val
green = training



EARLY STOPPING

- Para el entrenamiento antes de llegar al overfitting


A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)


ZOO

Optimización de hiperparametros

- Elegir el set optimo de hiperparametros para un algoritmo de aprendizaje (maximiza la performance del modelo)
- Son seteados antes del proceso de aprendizaje (#neurons, #cells, loss optimization algorithms, etc)

GRID SEARCH

1 — Identificar los hiperparametros del modelo a optimizar.

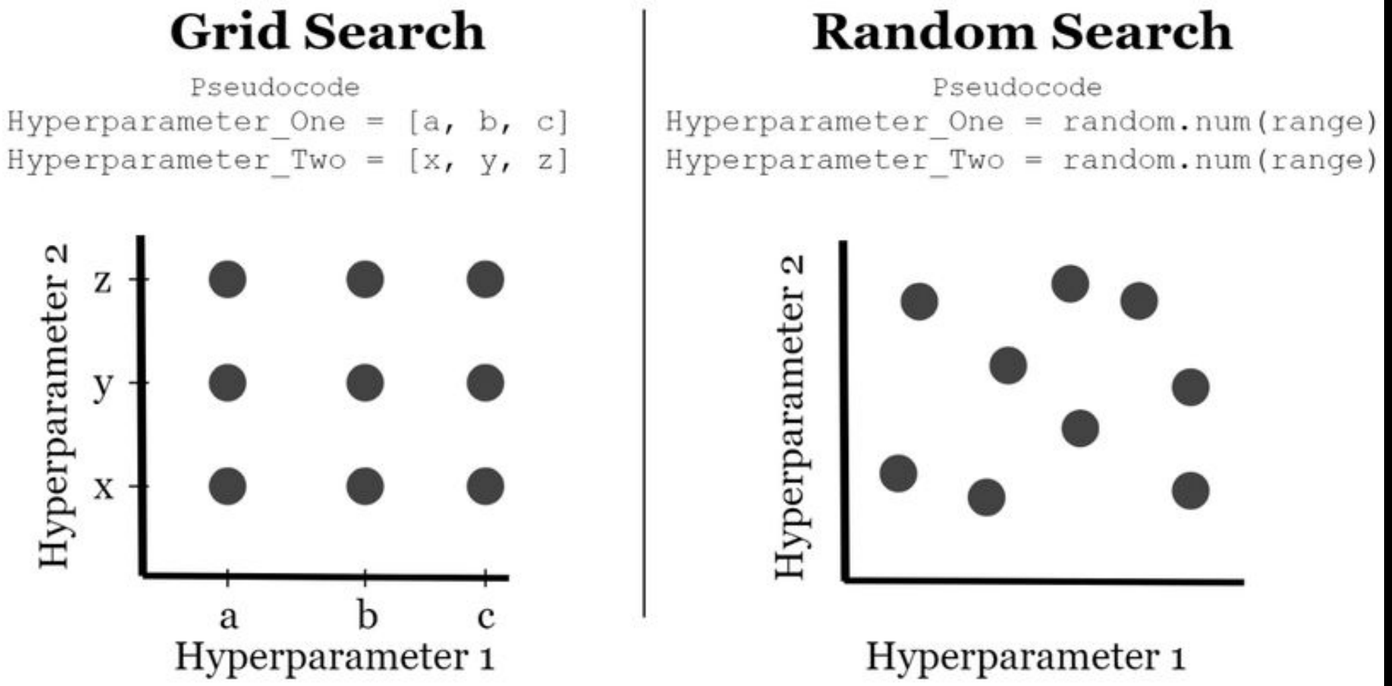
2 — Estimar el error obtenido para cada combinación en una grilla de hiperparametros.

3 — Selecciona la combinacion de hiperparametros con menor error

Optimización de hiperparametros

```
# learning_rate choices
 learning_rates = [ 0.1, 0.2, 0.3, 0.4, 0.5,
                    0.01, 0.02, 0.03, 0.04, 0.05 ]
 # iterations choices
 iterations = [ 100, 200, 300, 400, 500 ]

 parameters = []
 for i in learning_rates :
     for j in iterations :
         parameters.append( ( i, j ) )

 print("Available combinations : ",  parameters )

 # Applying linear searching in list of available combination
 # to achieved maximum accuracy on CV set

 for k in range( len( parameters ) ) :
   #  model = METHOD(..., learning_rate = parameters[k][0],iterations = parameters[k][1] )
   # ...
```

Optimización de hiperparametros



Otros métodos

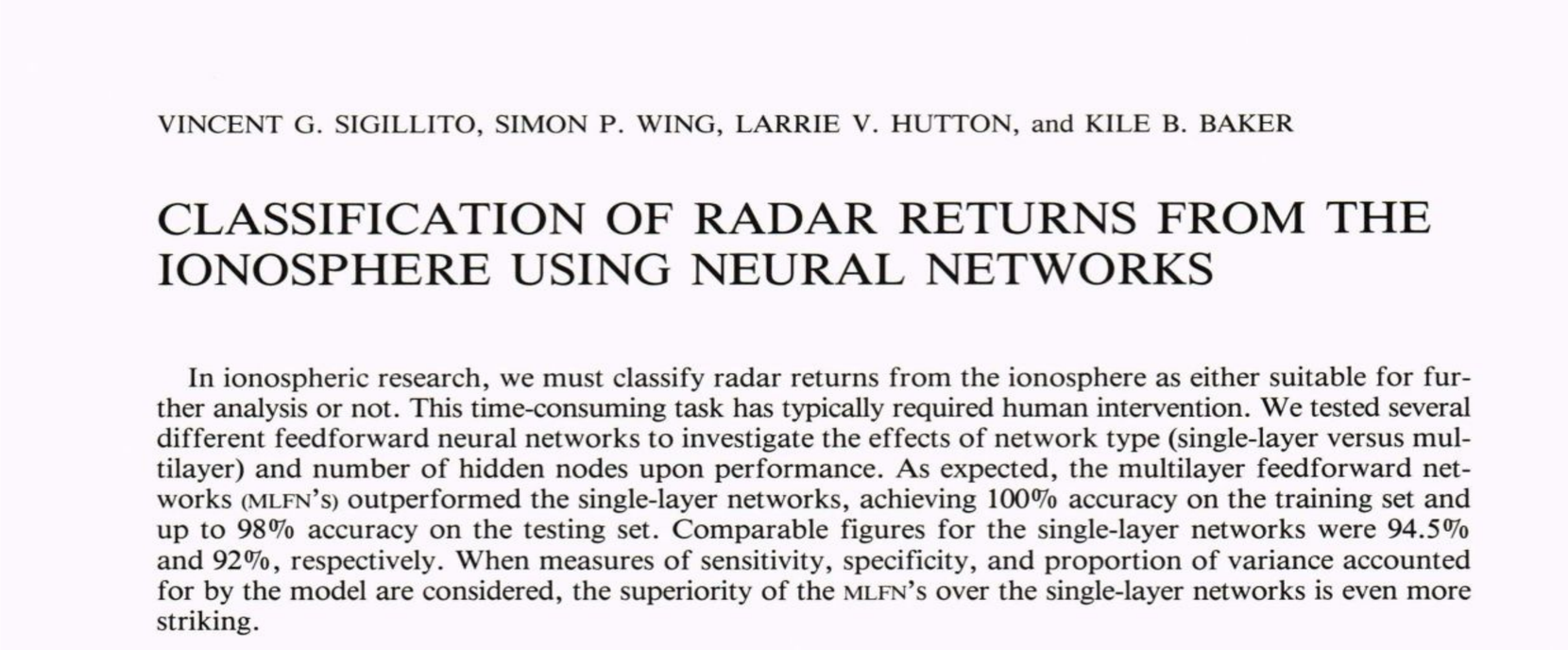
- Optimización Bayesiana
- Optimización Evolutiva

https://towardsdatascience.com/hyperparameters-tuning-from-grid-search-to-optimization-a09853e4e9b8

TP -

PROBLEMA-

cClasificar los ecos de un radar ionosférico segun dos clases (“good” or “bad”).
Hay 17 pares de números que representan 17 valores discretos correspondientes a la parte entera y compleja de la senial y que seran las entradas a la red neuronal o



VINCENT G. SIGILLITO, SIMON P. WING, LARRIE V. HUTTON, and KILE B. BAKER

CLASSIFICATION OF RADAR RETURNS FROM THE IONOSPHERE USING NEURAL NETWORKS

In ionospheric research, we must classify radar returns from the ionosphere as either suitable for further analysis or not. This time-consuming task has typically required human intervention. We tested several different feedforward neural networks to investigate the effects of network type (single-layer versus multilayer) and number of hidden nodes upon performance. As expected, the multilayer feedforward networks (MLFN's) outperformed the single-layer networks, achieving 100% accuracy on the training set and up to 98% accuracy on the testing set. Comparable figures for the single-layer networks were 94.5% and 92%, respectively. When measures of sensitivity, specificity, and proportion of variance accounted for by the model are considered, the superiority of the MLFN's over the single-layer networks is even more striking.

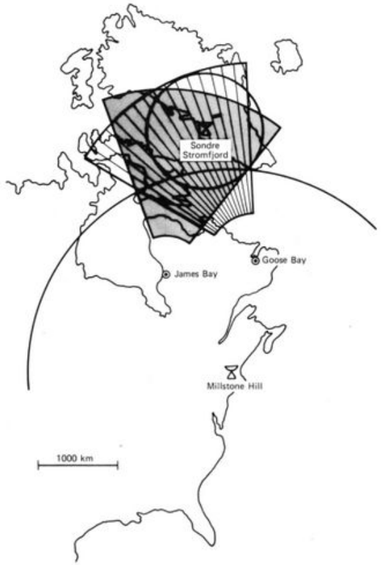
Paper source:

Sigillito, V. G., Wing, S. P., Hutton, L. V., \& Baker, K. B. (1989). Classification of radar returns from the ionosphere using neural networks. Johns Hopkins APL Technical Digest, 10, 262-266.

Data source:

Space Physics Group of the Johns Hopkins University Applied Physics Laboratory.

Dr. Z. Saavedra - FACET,
UNT



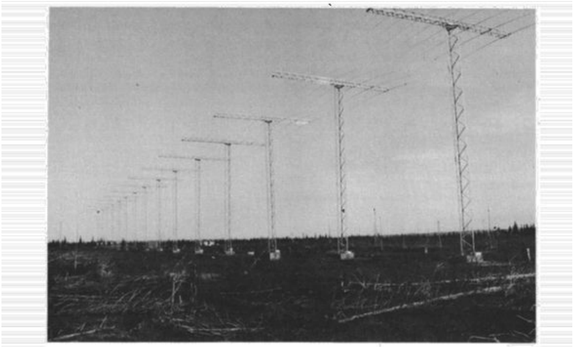
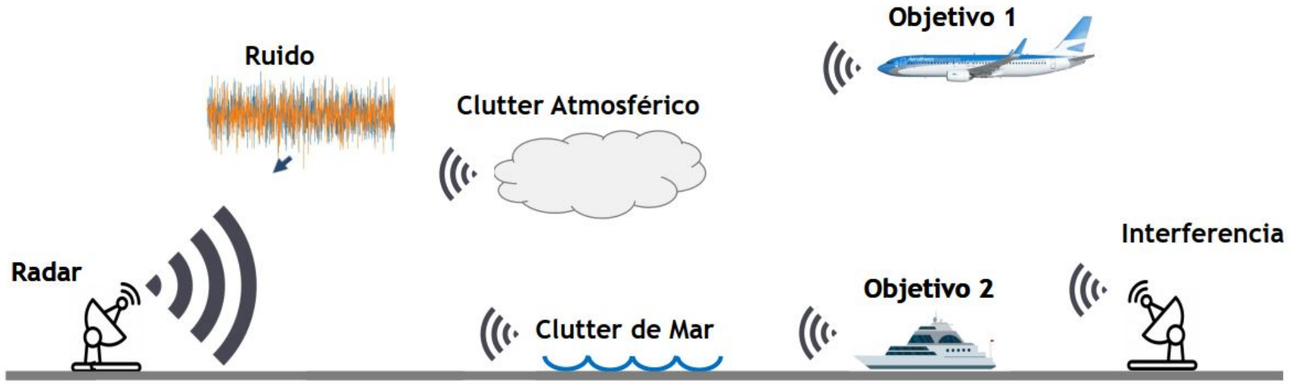
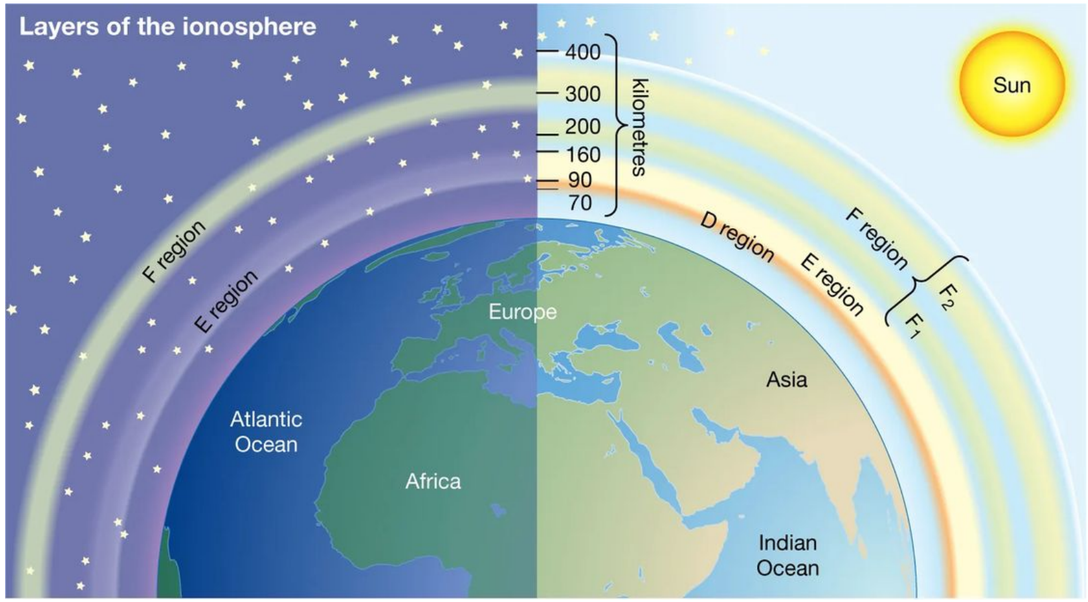


Fig. 7. View of antenna array from front, northwest corner. Shed for phasing matrix is located at the center of the array.



$$Eco(t) = Atte \cdot RCS \cdot u(t) \cdot m(t) \cdot e^{j\omega_c t \pm j\omega_D t + \varphi}$$

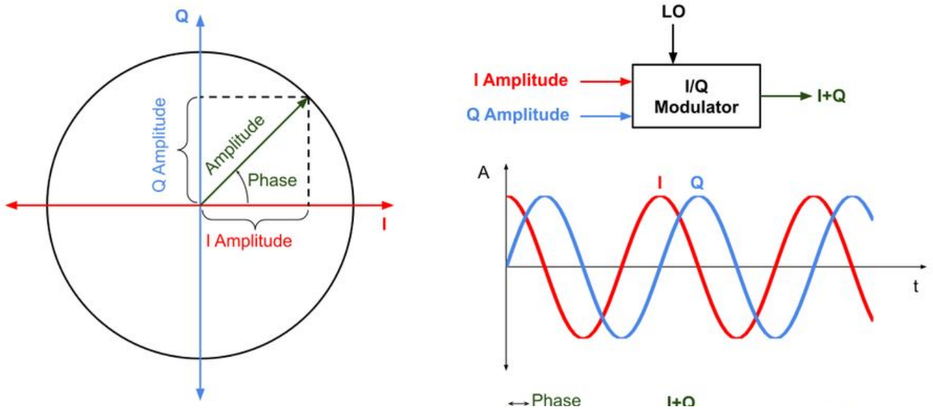
$$S_R(t) = Eco(t) + Clutter\ (t) + Ruido(t) + Interferencia(t)$$

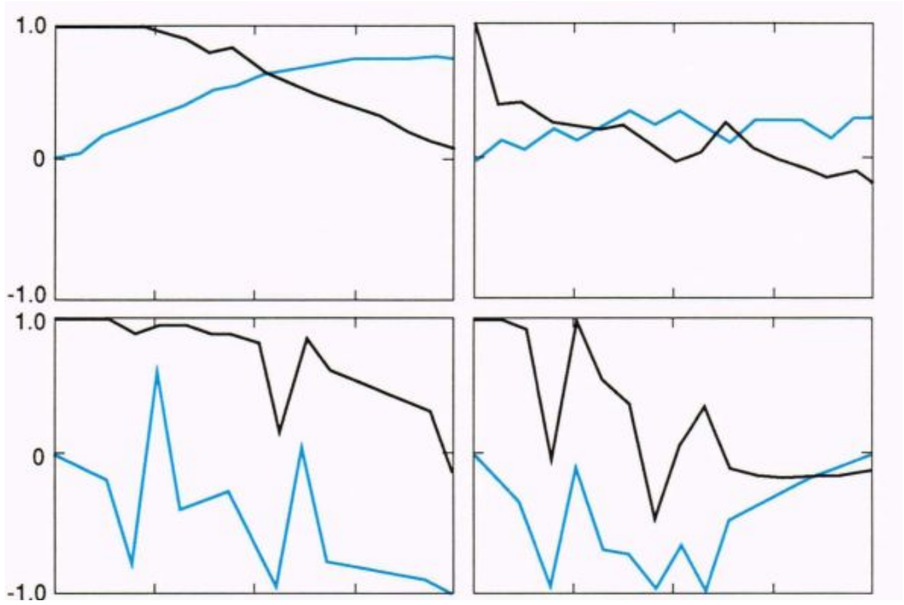


***: "target" = ionosphere**

IQ Modulation: it is a specific Phase-Modulation type, that provides us certain information in the real component of the signal and another possible coded information in the imaginary part. We can obtain both of them using demodulation techniques such as quadrature hybrids.

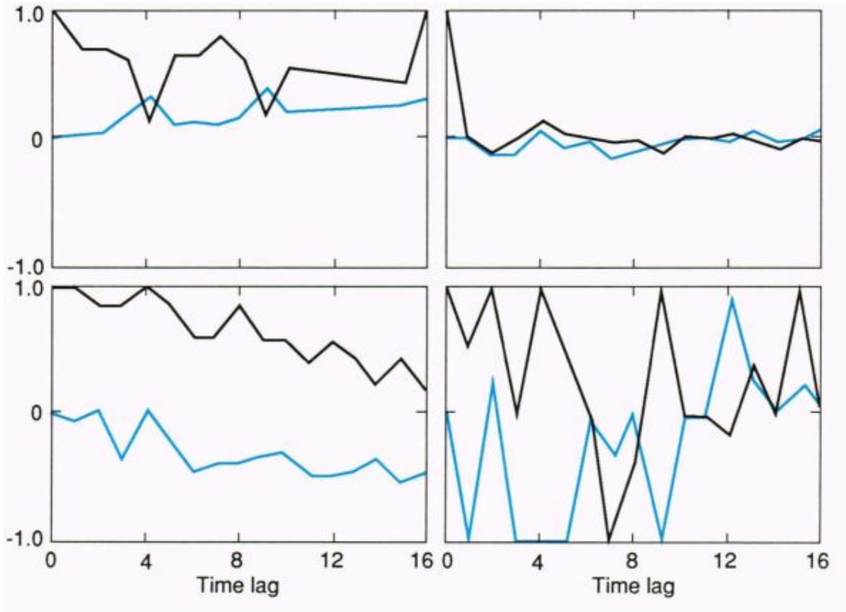
$$C(t) = A(t) + iB(t)$$





Good echo

Bad echo



Good echo

Bad echo

Auto-Correlation Function Output