



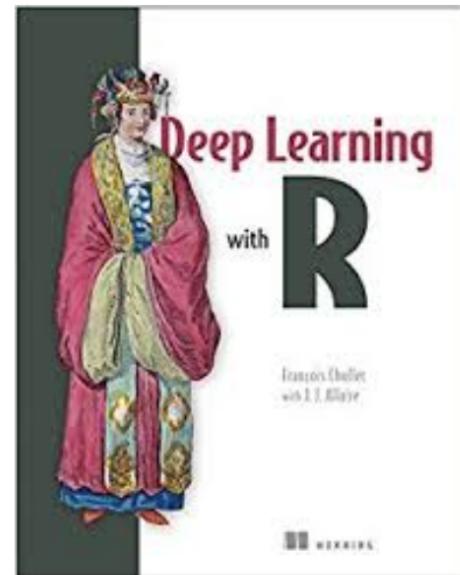
# Inteligencia Artificial con R

## Introducción al Deep Learning

Dr. Andrés Farall

# Referencias

Chollet, F., & Allaire, J. J. (2018). Deep Learning with R. Manning Publications Company.



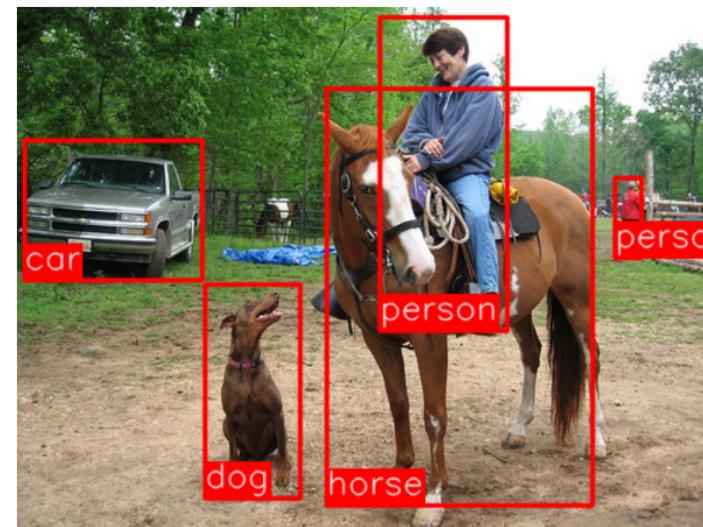
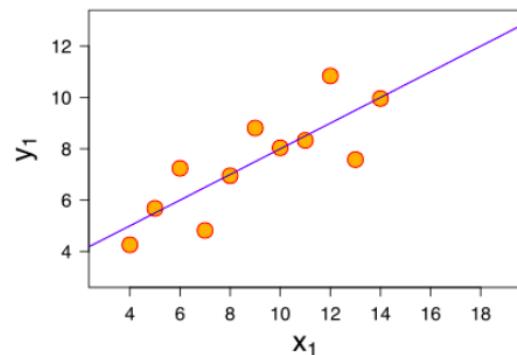
# Inteligencia Artificial

*Automatización de tareas de orden intelectual usualmente desarrolladas por personas físicas (humanos)*



# Machine Learning

Sistema informático (programa) que se **entrena** mediante el consumo de datos, para **aprender reglas** (o representaciones) con la capacidad de **generalizar** el conocimiento en futuras aplicaciones.

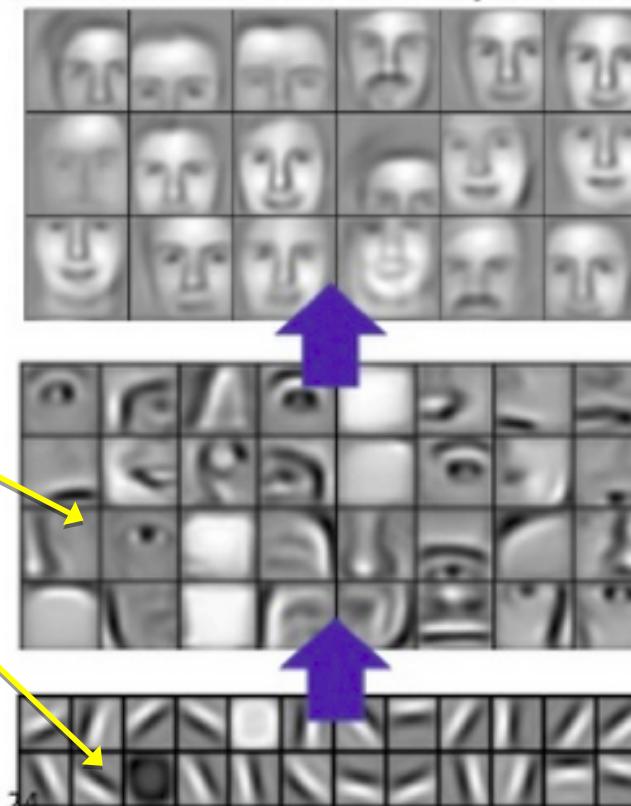


# Deep Learning

Es un subcampo del Machine Learning que aprende de los datos **representaciones útiles** mediante el apilamiento **jerarquico** y encadenado de **capas** que captan aspectos (features) del conocimiento con distintos **niveles de abstracción**.

Varios niveles de abstracción

Los niveles de abstracción se aprenden simultaneamente

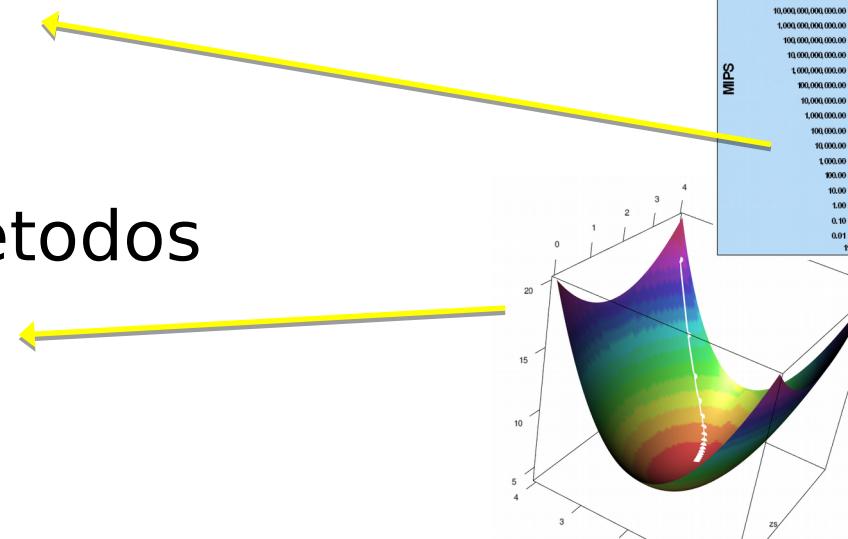
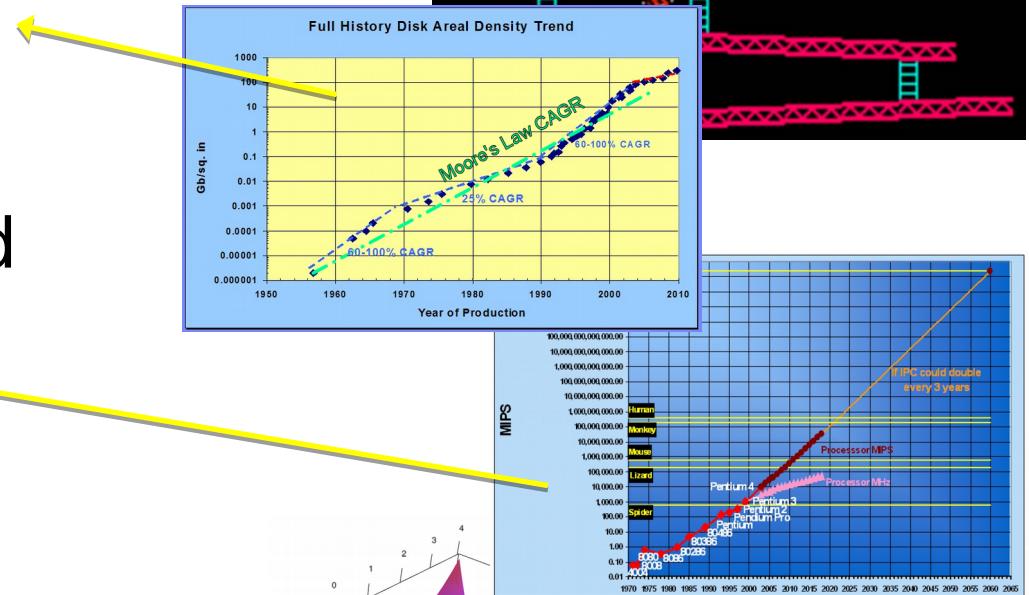
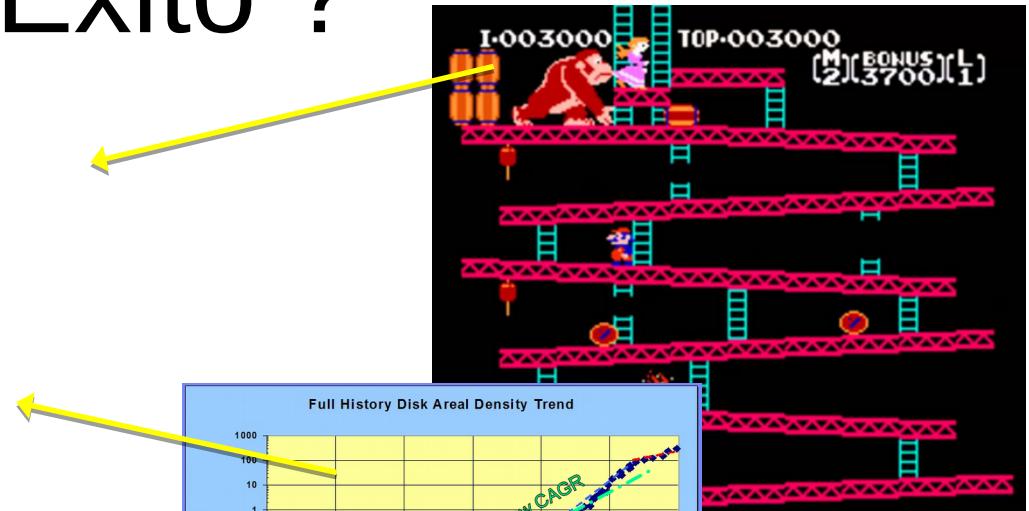


# Aplicaciones de Deep Learning

- Regresión (Single - Multiple Output)
- Clasificación (Single - Multiple Class)
- Reducción de Dimensión – Compresión
- Clasificación de **Imágenes**
- Reconocimiento del **Habla**
- OCR manuscrito
- Modelos Generativos (Síntesis de datos, Arte)
- Inferencia Estadística ????

# Las Claves del Éxito ?

- Disponibilidad de GPU's
- Abundancia de Datos
- Aumento en la velocidad de procesamiento
- Mejoras en los métodos de Optimización
- Remueve/automatiza el Feature Engineering
- Re-utilización (transfer learning) y Versatilidad

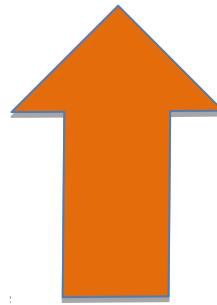


# Las Tecnologías a Usar

- **R**: Entorno de Data Science



- **KERAS**: Software OpenSource para ANN MLP



- **TensorFlow**: Software OpenSource para la programación de Grafos Computacionales

# La Interfase KERAS

- API OpenSource de muy alto nivel
- Corre tanto en CPU como en GPU
- Altísima flexibilidad en arquitectura de modelos
- Incluye capas:
  - Densas, Cnvolucionales
  - Regularización. Dropout
- Soporta múltiples back-ends:
  - TensorFlow (Google)
  - CNTK (Microsoft)
  - Theano (University of Montreal)

# La Interface TensorFlow

- Software Open Source de Computación Numérica de Grafos.
- Soporta procesamiento CPU y GPU
- Soporta trabajo Distribuido
- Tecnología subyacente a:
  - Keras
  - Tfestimators

# Arranquemos con algo de Historia Estadística

## Projection Pursuit Regression

Friedman, Jerome H., and Werner Stuetzle.  
"Projection pursuit regression." Journal of the  
American statistical Association 76.376 (1981):  
817-823.

- propuesta por Friedman y Tukey (1974)
- para regresión Friedman y Stuetzle (1981)
- usando splines Roosen y Hastie (1994)

# Projection Pursuit Regression

- Técnica supervizada que aproxima una respuesta univariada como suma de funciones continuas de **proyecciones lineales** de las covariables predictoras.
- Es un mejora/generalización sobre el modelo lineal de regresión.
- Es un **aproximador universal**.
- Como técnica supervizada puede verse como una versión simplificada de una Red Neuronal.

# El Modelo

## Idea central

extraer **combinaciones lineales** de las variables explicativas y luego modelar la respuesta como una función no lineal de estas combinaciones lineales

Modelo:

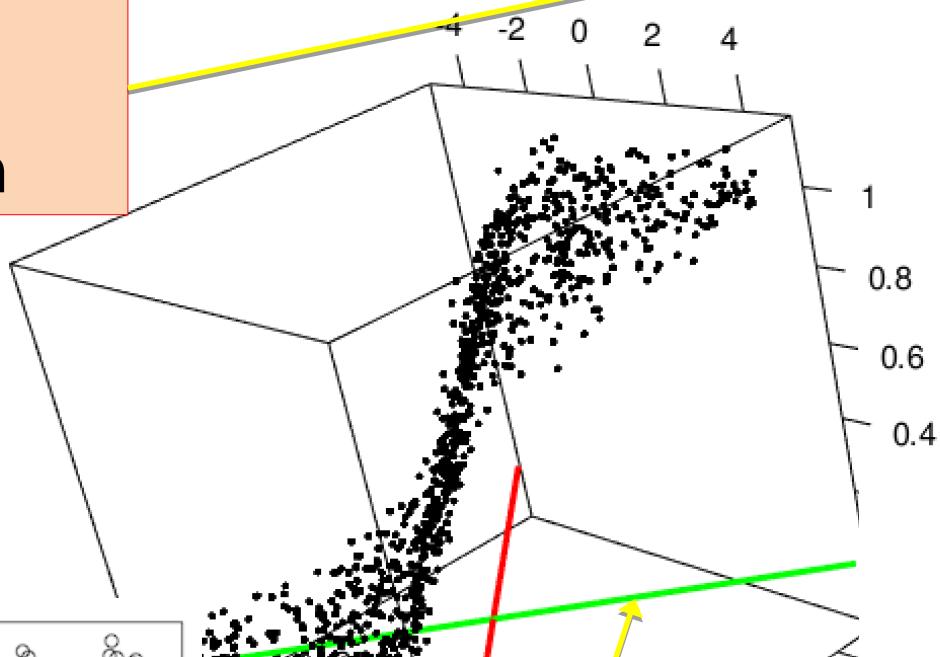
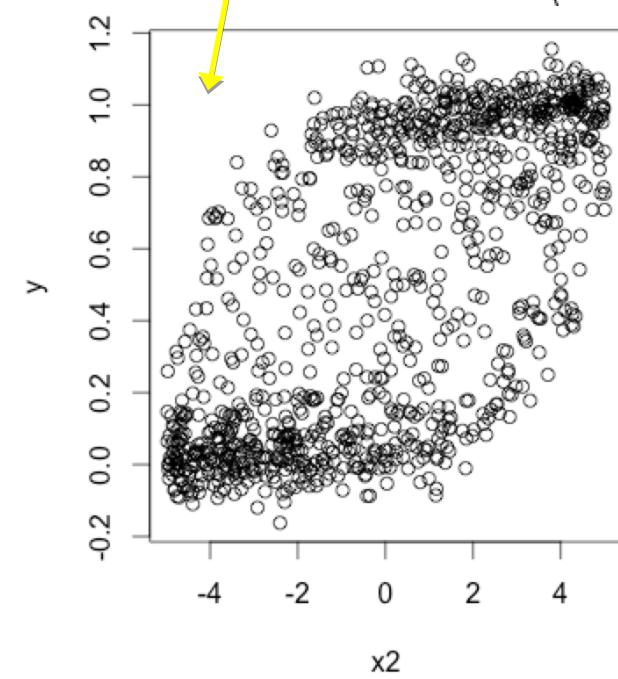
$$f(\mathbf{X}) = \sum_{m=1}^M g_m (\mathbf{w}_m^T \mathbf{X})$$

Función  
“ridge”

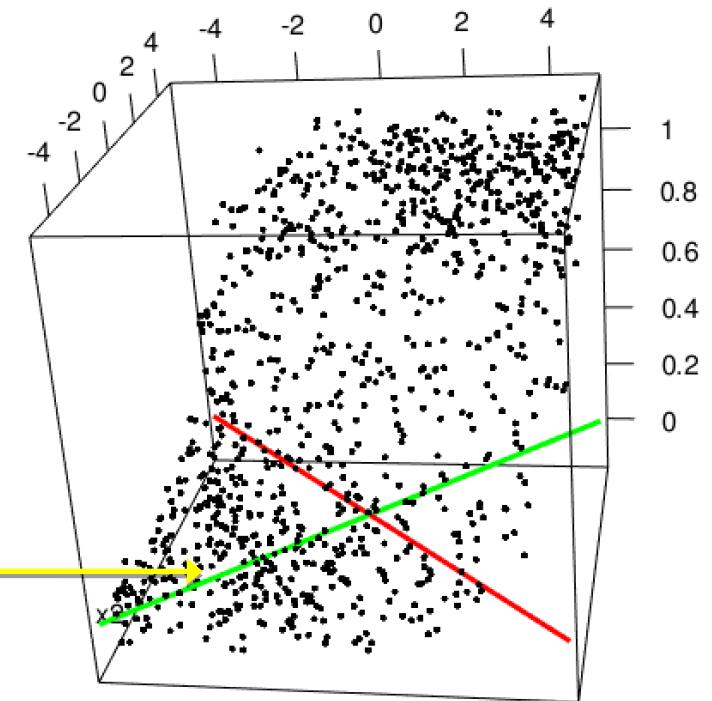
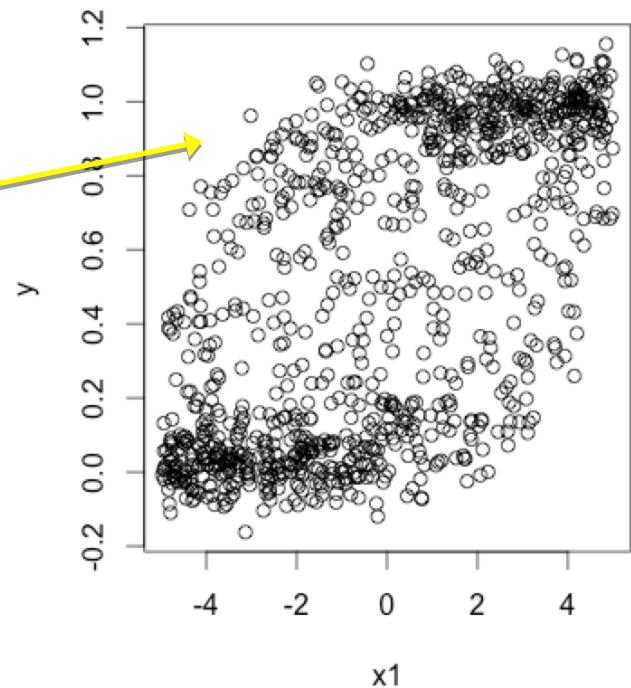
- donde  $\mathbf{w}_m$  ( $m = 1, \dots, M$ ) son vectores unitarios  $p$ -dimensionales de parámetros desconocidos
- y  $g_m$  son funciones no especificadas a estimar con los datos

# La Intuición

Escasa  
relación  
univariada

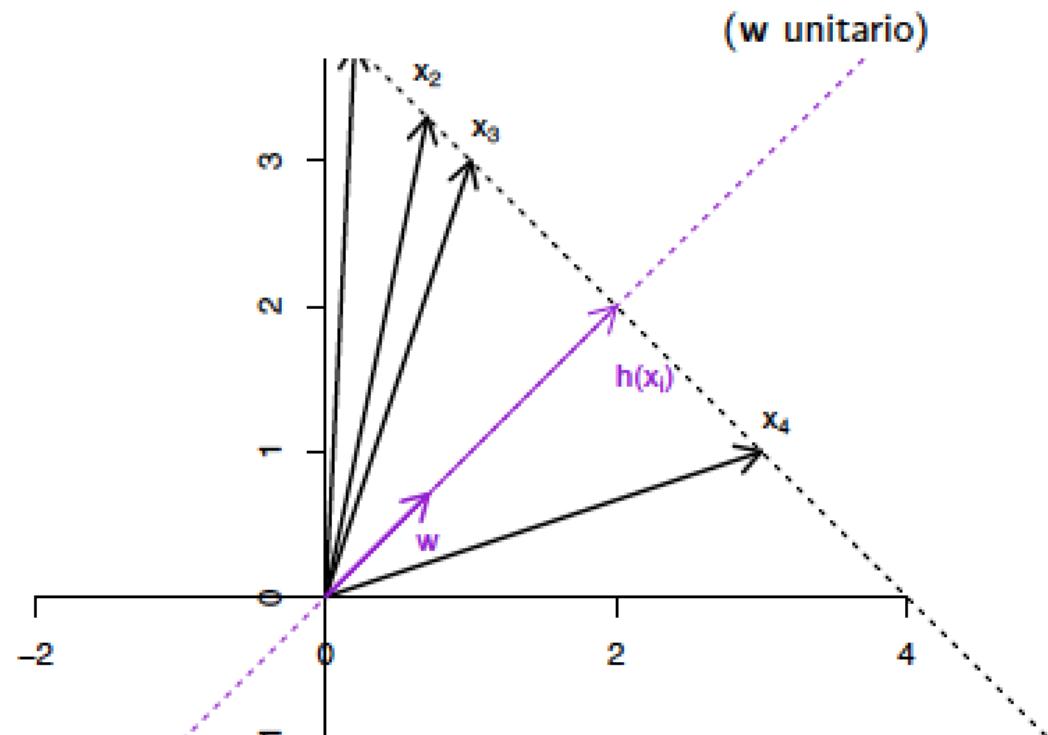


Dirección a  
ser hallada



# Busqueda de la Dirección

La función  $h : \mathbb{R}^P \rightarrow \mathbb{R}$ ,  $h(\mathbf{X}) = \mathbf{w}^T \mathbf{x}$  es la proyección del vector  $\mathbf{X}$



Buscamos  $w_m$  para que el modelo ajuste bien:  
Projection Pursuit

La función  $g_m(w_m^T \mathbf{X})$  es constante en hiperplanos (ortogonales a  $w_m$ ) y es unidimensional por lo que se sortea la maldición de la dimensionalidad

# PPR como Aproximador Universal

Porque la operación de aplicar funciones no lineales a combinaciones lineales genera una gran cantidad de clases de modelos. Por ejemplo

$$X_1 X_2 = \frac{1}{2} \left[ \left( \frac{X_1 + X_2}{\sqrt{2}} \right)^2 - \left( \frac{X_1 - X_2}{\sqrt{2}} \right)^2 \right]$$

Si  $M$  se toma suficientemente **grande**, para una elección apropiada de  $g_m$ , el modelo PPR puede ~~aproximar~~ cualquier función de  $\mathbb{R}^P$  pero...la interpretación del modelo es **difícil**

- PPR sirve para **predecir**
- Excepto, cuando  $M = 1$  (**Single Index Model**)

Dados  $(\mathbf{X}_i, Y_i)$ ,  $(i = 1, \dots, n)$ , ¿cómo ajustamos? Buscamos minimizar la función

$$\sum_{i=1}^n \underbrace{\left[ Y_i - \sum_{m=1}^M g_m(\mathbf{w}_m^T \mathbf{X}_i) \right]^2}_{\text{residuo de la } i\text{-ésima observación}}$$

## Estimación<sub>(1)</sub>

Asumimos  $\mathbf{w}$  fijo

Transformamos las observaciones:  $V_i = \mathbf{w}_m^T \mathbf{X}_i$ . Tenemos un problema de suavizado univariado  $(V_i, Y_i)_i$  observaciones en  $\mathbb{R}^2$ . Ajustamos una regresión no paramétrica: smoothing spline o promedios móviles.

Asumimos  $g$  fija

Queremos minimizar (1) sólo en  $\mathbf{w}$ . Resulta equivalente a un problema de regresión de mínimos cuadrados pesados

Los pasos para actualizar a  $g$  y  $\mathbf{w}$  se iteran hasta converger

## Ajuste de PPR: caso $M > 1$

Supongamos que hemos determinado los primeros  $\hat{g}_1, \dots, \hat{g}_{m-1}$  y  $\hat{\mathbf{w}}_1, \dots, \hat{\mathbf{w}}_{m-1}$ . Sean

$$R_i = Y_i - \sum_{j=1}^{m-1} \hat{g}_j (\hat{\mathbf{w}}_j^T \mathbf{x})$$

los residuos de esta aproximación. Le aplicamos el algoritmo anterior a los pares  $(\mathbf{x}_i, R_i)$ , para obtener  $\hat{g}_m$  y  $\hat{\mathbf{w}}_m$ . El procedimiento se itera hasta que la mejora que se obtiene de la función objetivo es muy pequeña.

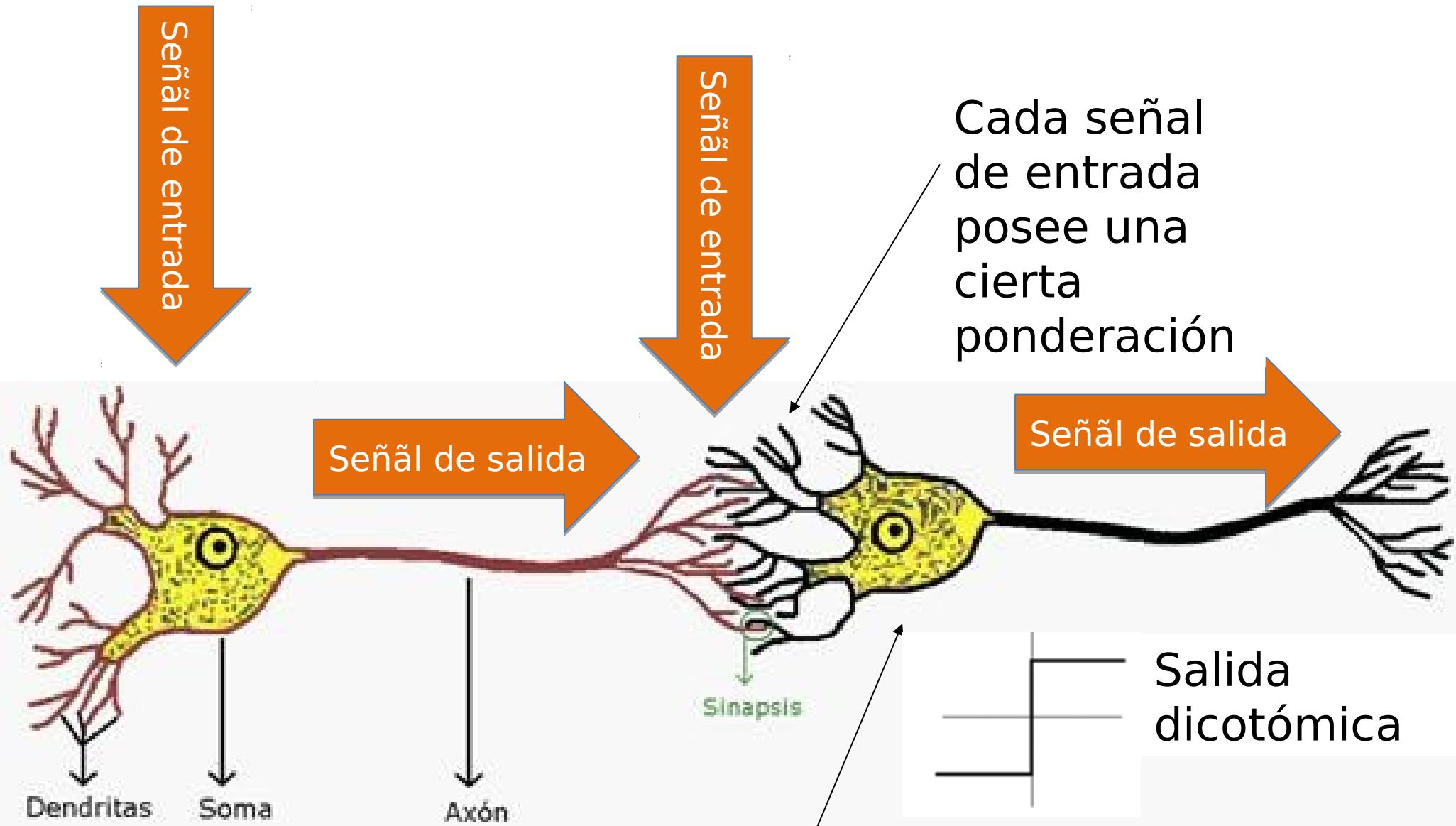
Muchos detalles de implementación:

- ¿qué técnica de suavizado se usa?
- ¿cómo se determina el parámetro de suavizado?
- estimación de  $M$ : validación cruzada o parte de la estrategia forward

# Redes Neuronales Artificiales

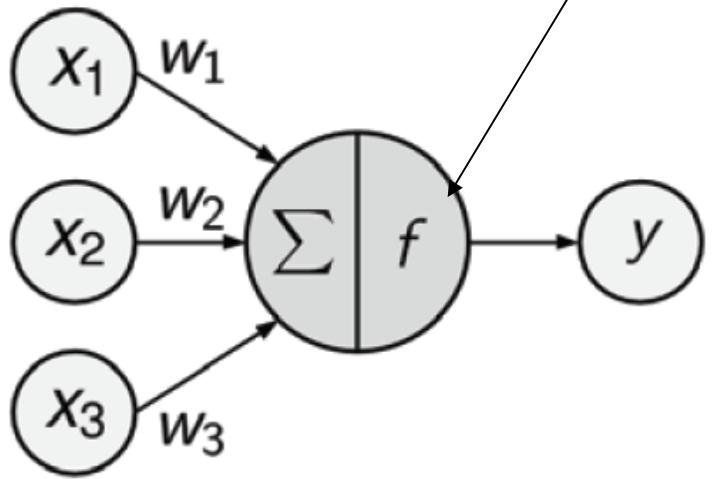
- Son métodos tipo “**Black Box**”
- Son buenos aproximadores de funciones
- Tendencia al **sobreajuste** (usar penalización)
- Sirven naturalmente tanto para **Regresión** como para **Clasificación**
- Particularmente útiles en el reconocimiento de **patrones**
- Se calibran mediante **Gradient Descent Estocástico**
- En el fondo, son modelos No Lineales encadenados
- Particularmente útiles cuando hay múltiples outputs

# La Neurona



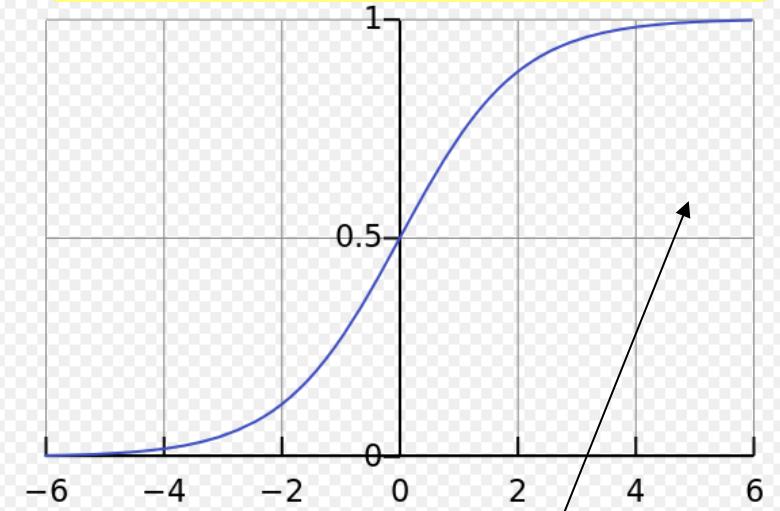
La neurona “suma” las señales ponderadas y “gatilla” una señal de salida

# Elementos Básicos de una Red



Función de  
activación

$$f(x) = \frac{1}{1 + e^{-(x)}}$$



$$f\left(\sum_{i=1}^n w_i x_i\right) = y(x)$$

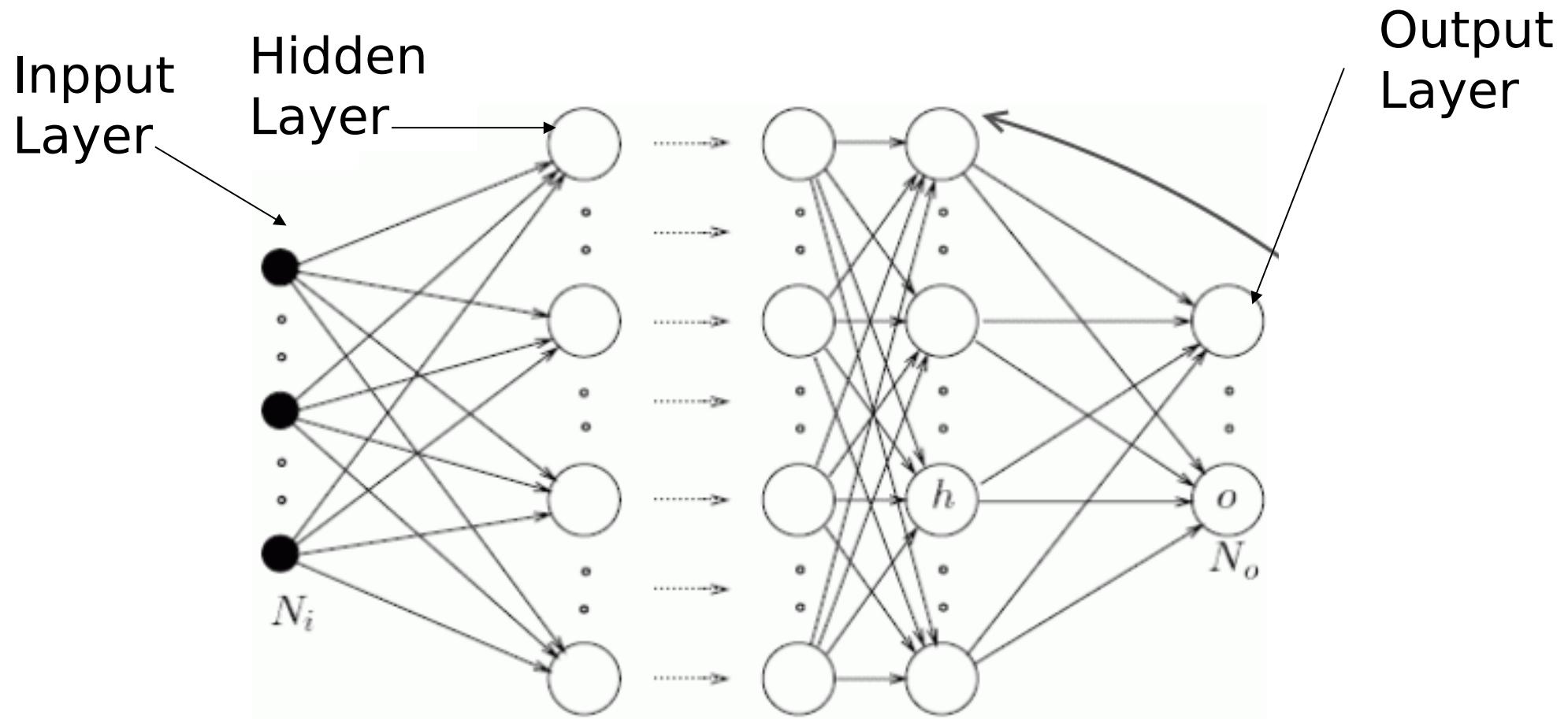
Tangente  
Hiperbólica

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Función  
Sigmoidal o  
Logística

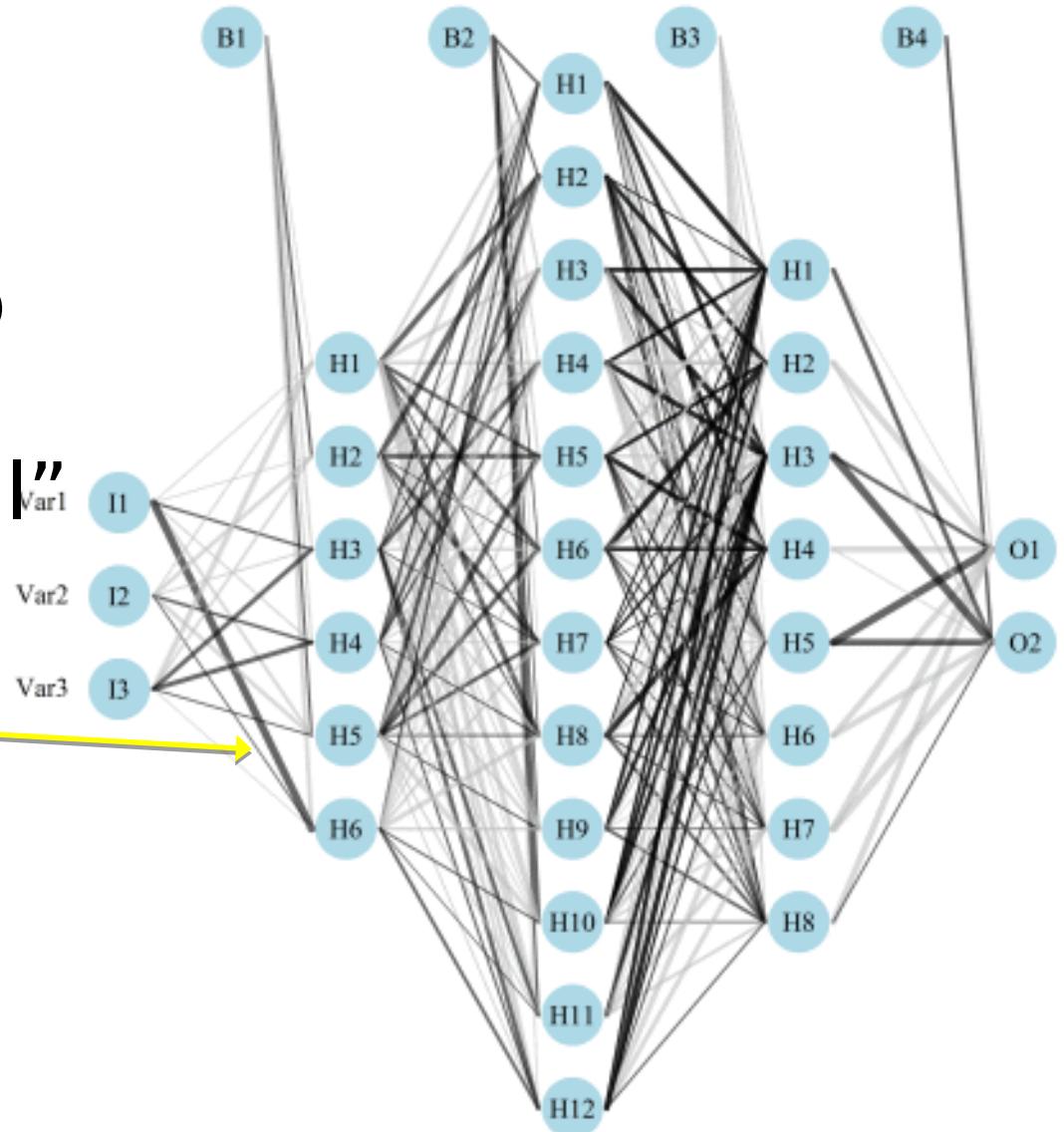
# Topologías de Redes Neuronales

- La complejidad de una red depende de:
  - La **cantidad** de capas de neuronas
  - La **cantidad** de neuronas por capa
  - La **dirección** de las señales

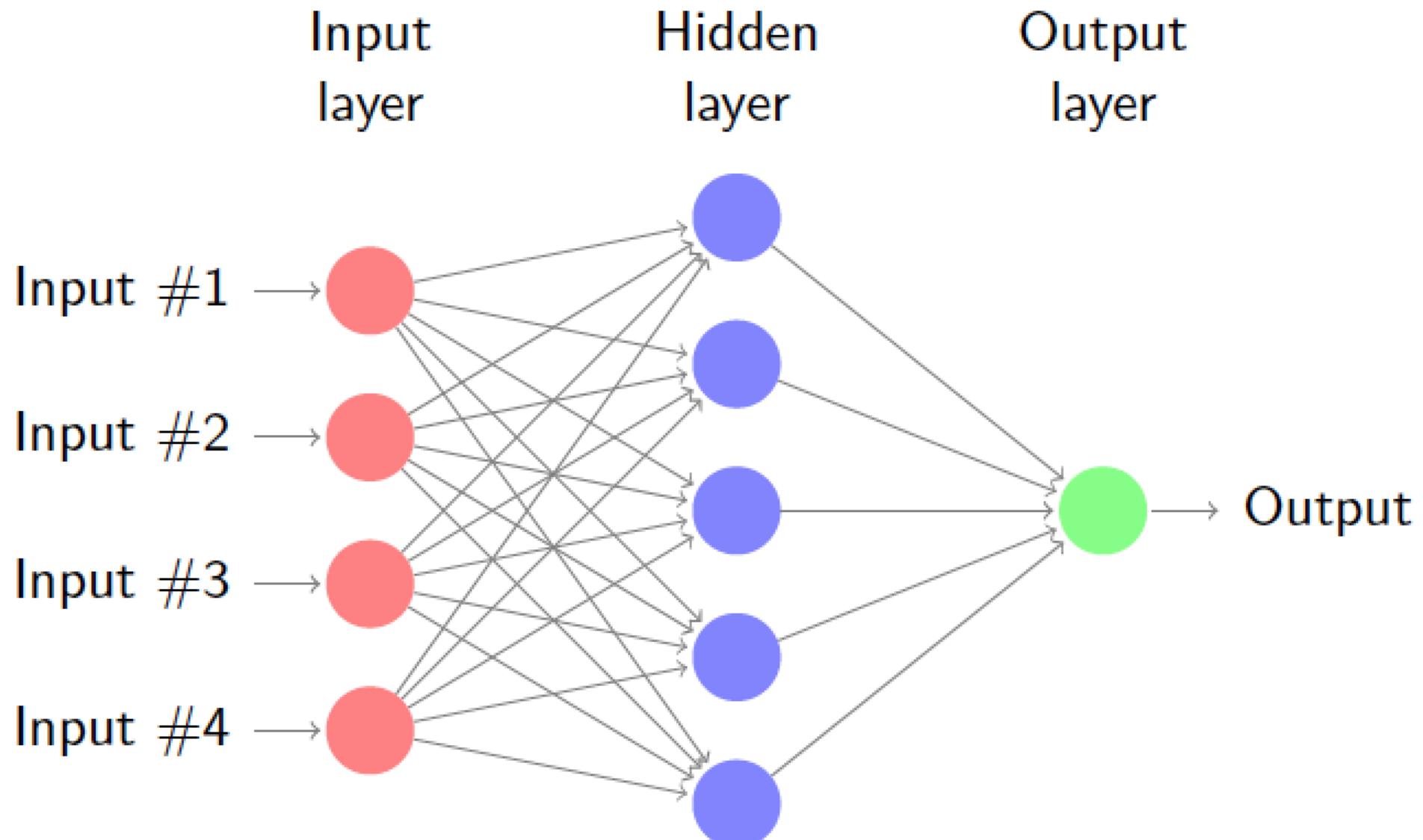


# El Multilayer Perceptron

- Es la arquitectura más difundida
  - Al menos una capa oculta
  - El modelo más sencillo (1 capa) ya genera un “aproximador universal”
  - Gran cantidad de parámetros
- a ser calibrados

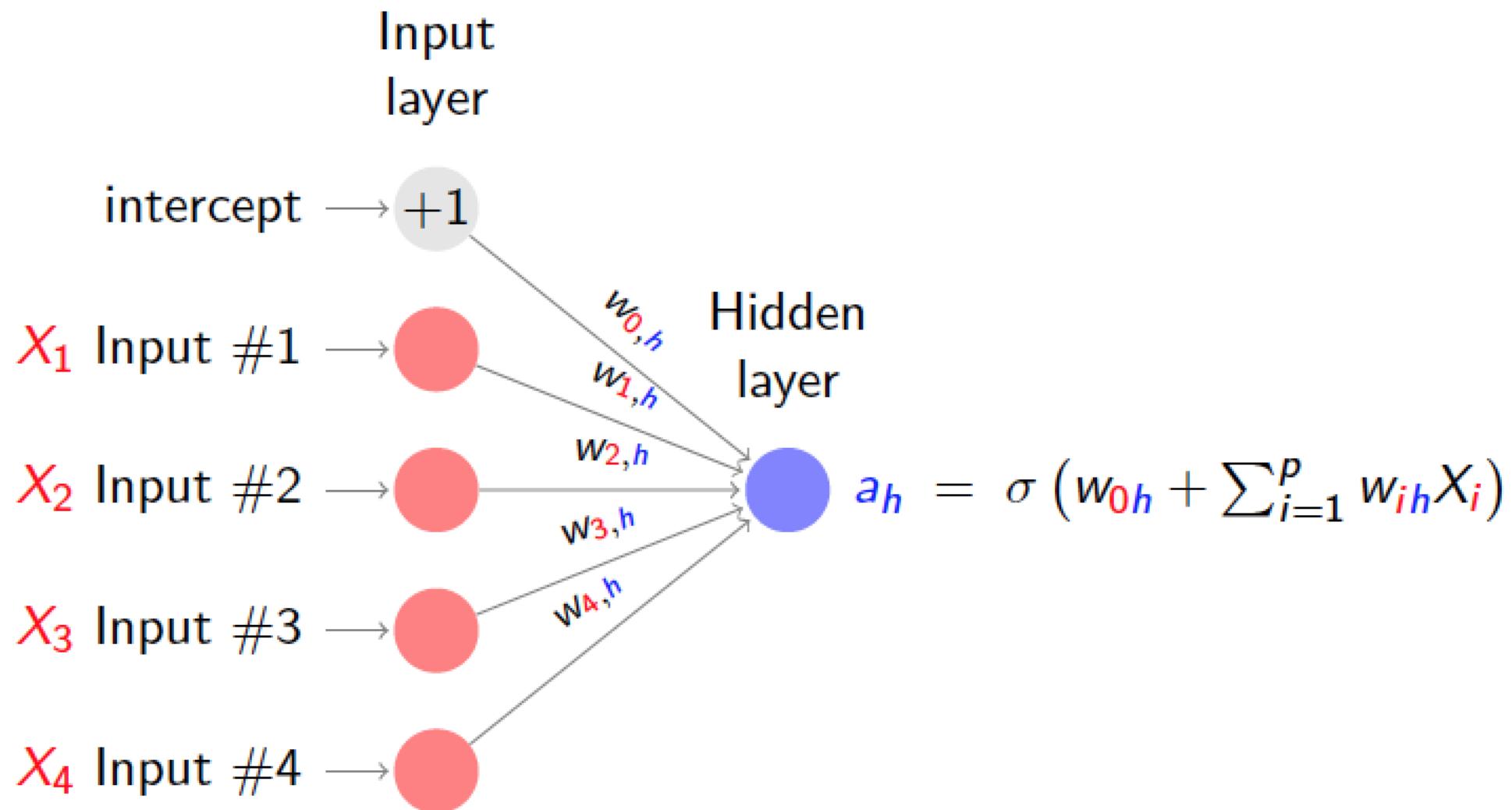


# Redes Neuronales: Ejemplo



4 variables predictoras o inputs 5 unidades ocultas

# Ejemplo de Red Neuronal



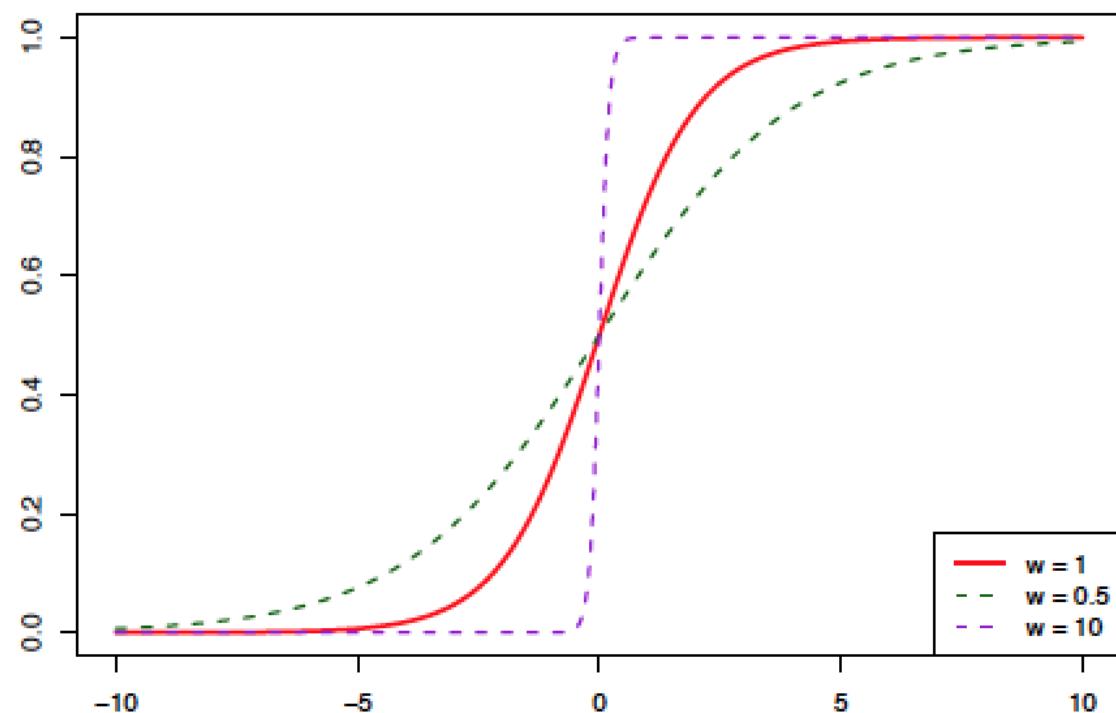
# Red Neuronal

- Nodos: unidades de memoria o neuronas
- $w_{ih}^{(l)}$ : son pesos (parámetros)  $l$  representa la layer,  $i$  indica el nodo del que sale,  $h$  el nodo al que llega
- El modelo de redes neuronales impone que cada nodo es una función no lineal de una combinación lineal de los nodos de la capa (o layer) anterior
- $\sigma$  es una función no lineal, usualmente la sigmoidea

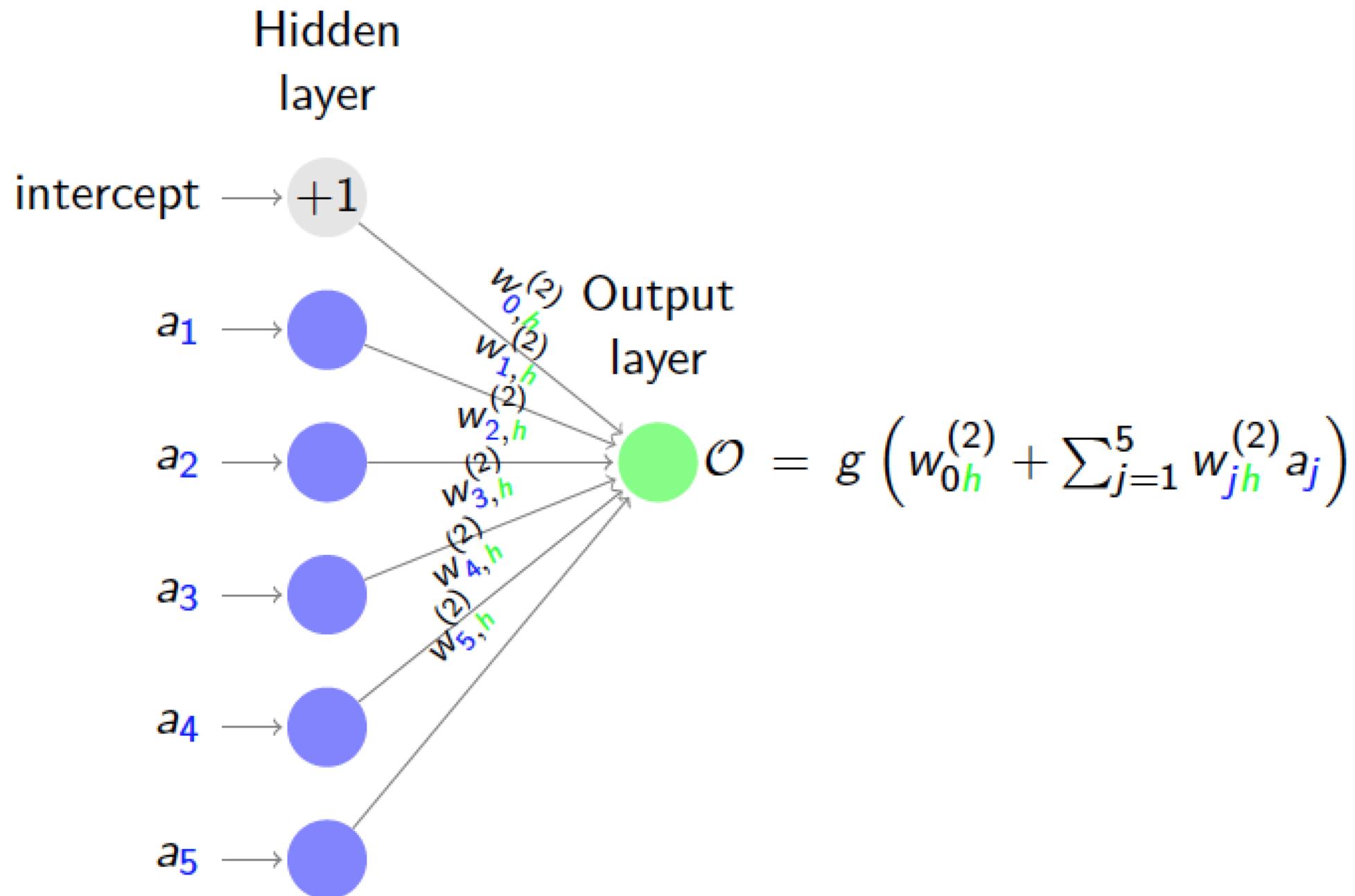
$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Graficamos

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



# Red Neuronal: output



## Red Neuronal: output

$$output = \mathcal{O} = g \left( w_{0h}^{(2)} + \sum_{j=1}^5 w_{jh}^{(2)} a_j \right) \quad (2)$$

¿Cómo se elige la función  $g$ ? Depende de la variable respuesta.

- En problemas de regresión ( $Y$  continua),  $g$  es la identidad
- Cuando la respuesta es binaria,  $g$  es la sigmoidea
- Cuando la respuesta es categórica, con  $K$  categorías,  $g$  es la identidad en cada nodo, pero el output final lleva una normalización:

$$\mathcal{O}_k = \hat{p}(\text{categoría } k \mid (z_1, \dots, z_K)) = \frac{e^{z_k}}{\sum_{h=1}^K e^{z_h}}$$

(transformación de la logística múltiple, o *función softmax*)  
donde  $z_k = w_{0k}^{(2)} + \sum_{j=1}^5 w_{jk}^{(2)} a_j$ .

# Red Neuronal: relación con otros modelos

- Sin *hidden layers*, las redes neuronales son un [modelo lineal generalizado](#)
- ([Vínculo con PPR](#)): Las redes neuronales con una sola *hidden layer* tienen la misma forma que el modelo PPR. La diferencia es que PPR usa funciones no paramétricas y RN usa sigmoideas.
- Las RN representan una versatilidad de modelos, ya que pueden variar
  - cantidad de unidades en la capa oculta
  - cantidad de capas ocultas

# Redes Neuronales: Ajuste

Todas las capas son funciones de las capas anteriores, que a su vez son funciones de las variables explicativas:

$$f(\mathbf{X}, \mathcal{W})$$

- $\mathcal{W}$  es la colección de pesos. ¿Cuántos? En el ejemplo  $(p + 1)H + (H + 1) = 5 \cdot 5 + 6 = 31$  ¡muchos!
- Si contáramos con
  - una muestra de entrenamiento:  $(\mathbf{X}_1, Y_1), \dots, (\mathbf{X}_n, Y_n)$
  - una función objetivo:  $L[Y, f(\mathbf{X}, \mathcal{W})]$
  - un algoritmo de optimizaciónpodríamos estimarla.
- Debemos penalizar la función objetivo en los pesos, para evitar el sobreajuste

# Redes Neuronales: Ajuste

Queremos hallar los pesos  $\mathcal{W}$  que resuelvan

$$\min_{\mathcal{W}} \left\{ \frac{1}{n} \sum_{i=1}^n L[Y_i, f(\mathbf{X}_i, \mathcal{W})] + \lambda J(\mathcal{W}) \right\} \quad (3)$$

En realidad  $J(\mathcal{W}) = \sum_{s=1}^S \lambda_s J_s(\mathcal{W})$ , es el término de penalización

- Se usó primero la pérdida cuadrática,

$$J(\mathcal{W}) = \frac{1}{2} \sum_{i=1}^n \sum_{l=1}^L \sum_{h=1}^{H_l} \left[ w_{ih}^{(l)} \right]^2 \text{ (weight decay penalty)}$$

(penalidad tipo Ridge)

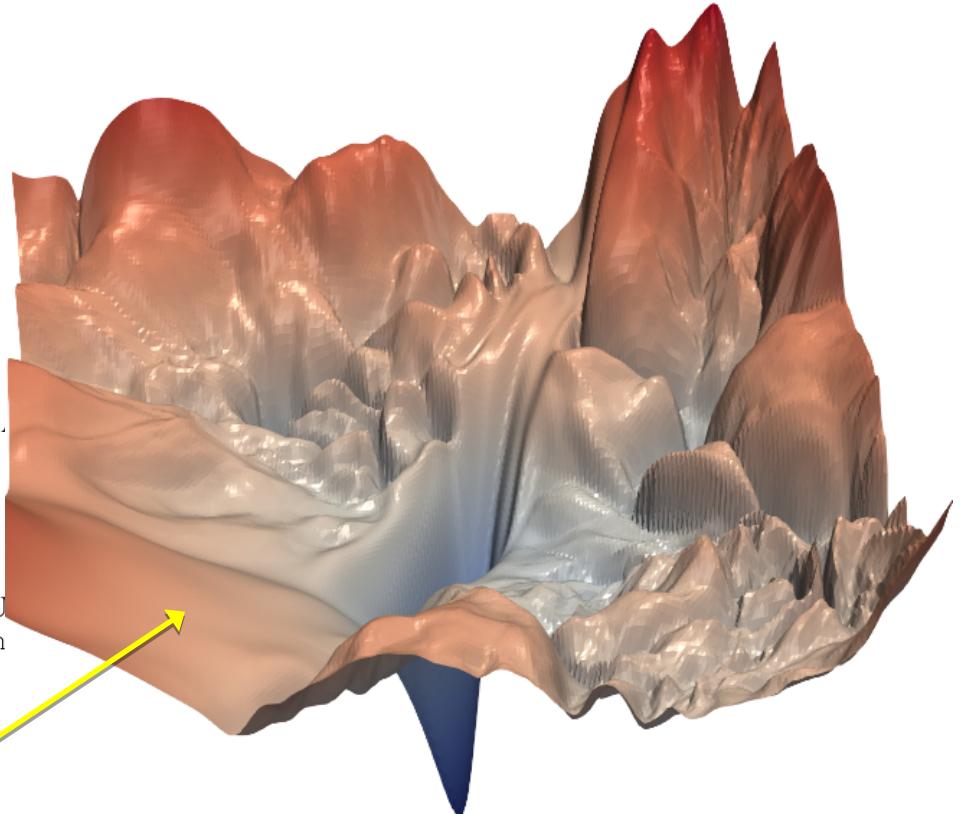
- Luego penalidades de tipo Lasso ( $l_1$ ), con el  $|\cdot|$  en lugar del cuadrado (pesos mas ralos)
- Combinaciones de ambos (*elastic net*)

# Función objetivo

## VISUALIZING THE LOSS LANDSCAPE OF NEUR...

Hao Li<sup>1</sup>, Zheng Xu<sup>1</sup>, Gavin Taylor<sup>2</sup>, Christoph Studer<sup>3</sup>, Tom Goldstein<sup>1</sup>

<sup>1</sup>University of Maryland, College Park, <sup>2</sup>United States Naval Academy, <sup>3</sup>Cornell U  
`{haoli, xuzh, tomg}@cs.umd.edu, taylor@usna.edu, studer@corn`



- Si la respuesta es **continua**,  $L$  es la pérdida cuadrática

$$L[Y_i, f(\mathbf{X}_i, \mathcal{W})] = [Y_i - f(\mathbf{X}_i, \mathcal{W})]^2$$

- Si la respuesta es **categórica**,  $L$  es la deviance binomial (multiplicada por (-1))

$$L[Y_i, f(\mathbf{X}_i, \mathcal{W})] = Y_{ik} \log(f_k(\mathbf{X}_i, \mathcal{W}))$$

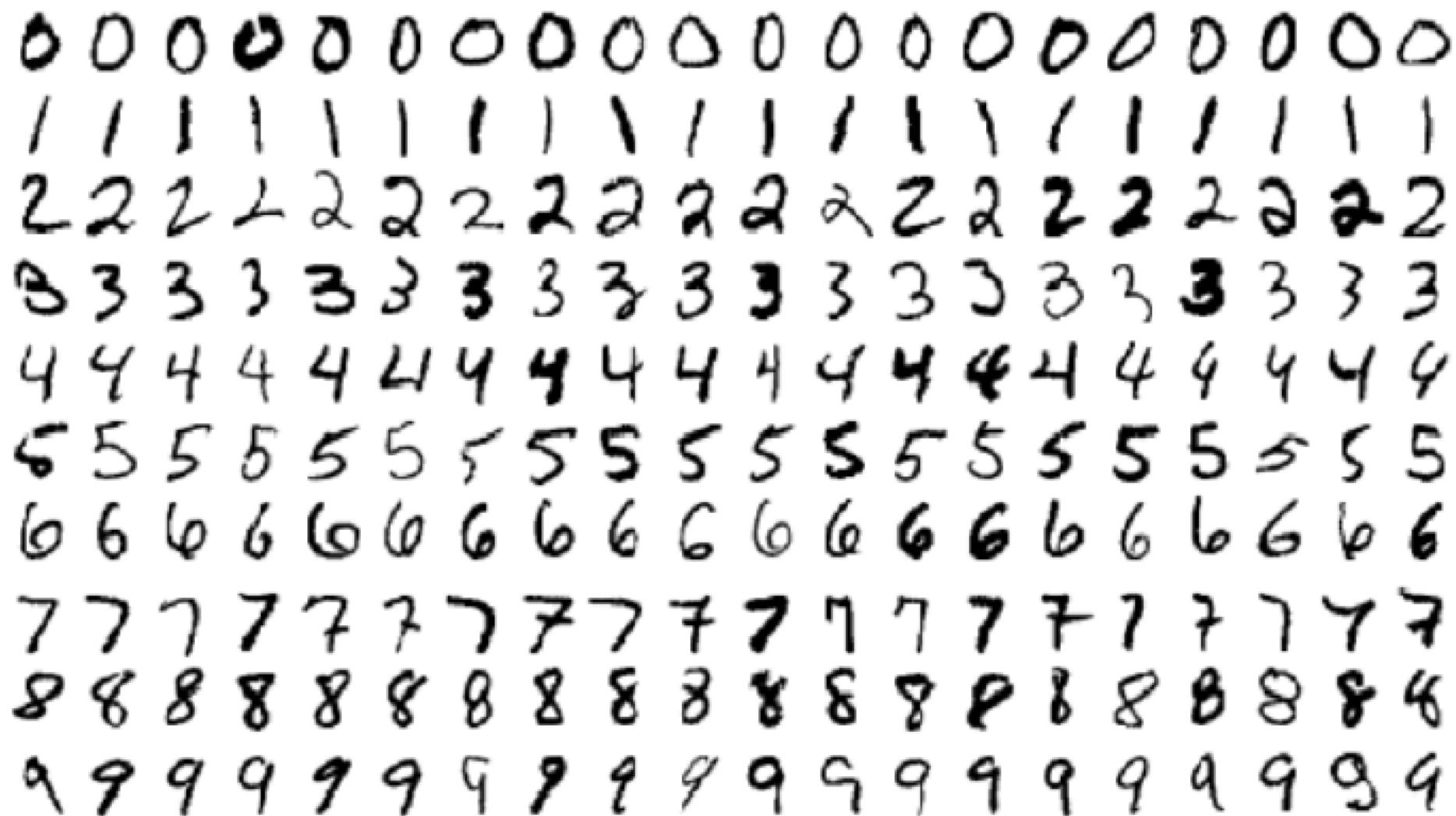
# Redes Neuronales: Algoritmo

La implementación del ajuste está lleno de detalles y mejoras

- La función objetivo es convexa en  $f$ , pero no en los pesos, no es fácil encontrar óptimos, hallaremos óptimos locales
- $f$  es una función diferenciable y  $L$  también lo es: usamos el método o búsqueda dirigida por el gradiente (*gradient descent*).
- Se inicializa en valores aleatorios
- El gradiente se calcula mediante un algoritmo que se basa en la estructura jerárquica de los pesos: *back propagation*
- El algoritmo se acelera combinándolo con selecciones aleatorias de los datos a actualizar: *Stochastic Gradient Descent*
- Los parámetros del método del gradiente son elegidos adaptivamente.

## Ejemplo: Dígitos postales (ZIP code)

Problema de clasificación: buscamos un clasificador automático de dígitos manuscritos. Base de datos de **60.000 dígitos manuscritos de entrenamiento**. Y otros **10.000 para testear**. Por ejemplo:



## Ejemplo: Dígitos postales (ZIP code)

Cada dígito está representado por una escala de grises de  $28 \times 28 = 784$  píxeles,  $(X_1, \dots, X_{784})$ . El valor que se guarda en cada píxel es un número positivo que indica la intensidad de gris presente en esa ubicación. Los 784 píxeles representan las covariables, la respuesta es un número de 0 a 9.

Presentamos la red ajustada por

- Efron, B. y Hastie T. (2016) *Computer Age Statistical Inference Algorithms, Evidence, and Data Science*. Cambridge University Press.
- Ajustado con el paquete h2o de R.
- MNIST es la base de datos curada (LeCun y Cortes, 2010) disponible públicamente antes descripta.

## Ejemplo: Dígitos postales (ZIP code), conclusión

Esa red, con los parámetros apropiadamente elegidos, da un error de clasificación del 0,93 % en el conjunto oficial de testeo. Random forests 2,8 % de error, modelo lineal generalizado 7,2 % Acá los 93 dígitos mal clasificados (verdadero en azul, clasificación en rojo)

42	53	60	82	58	97	89	65	72	46	72	94
49	95	71	57	83	79	87	46	93	06	37	93
4	9	1	5	8	7	8	4	9	6	3	9
23	94	53	20	37	49	61	90	91	94	24	61
2	4	5	0	3	9	6	9	1	9	2	6
53	95	61	65	32	85	35	97	12	49	60	37
3	9	1	6	3	9	5	9	1	4	0	3
91	64	50	85	72	46	13	46	03	97	27	32
9	4	5	8	3	4	1	4	0	9	2	1
87	89	61	80	94	72	70	49	53	38	38	39
2	8	1	8	4	2	7	4	3	3	3	3
89	97	71	07	95	85	05	39	85	49	72	72
2	7	1	0	9	8	0	3	8	4	7	7
72	06	97	27	47	63	56	42	50			
2	0	9	2	4	1	5	4	5			

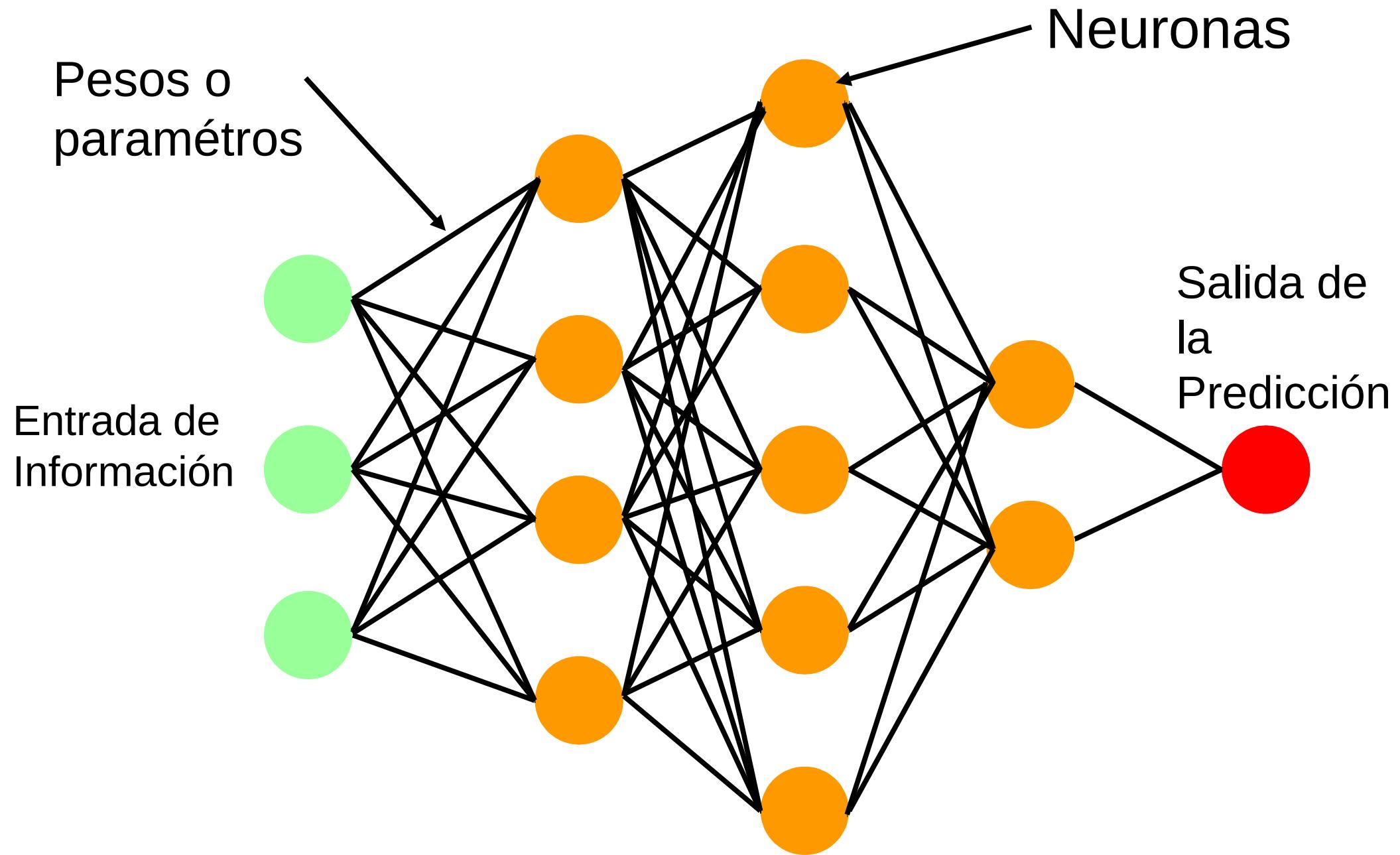
## Ejemplo: Dígitos postales (ZIP code), evolución

Para los datos MNIST, la mejor red neuronal daba

- En el 2008 un error del 1,6 %
- En el 2016 un error del 0,93 %

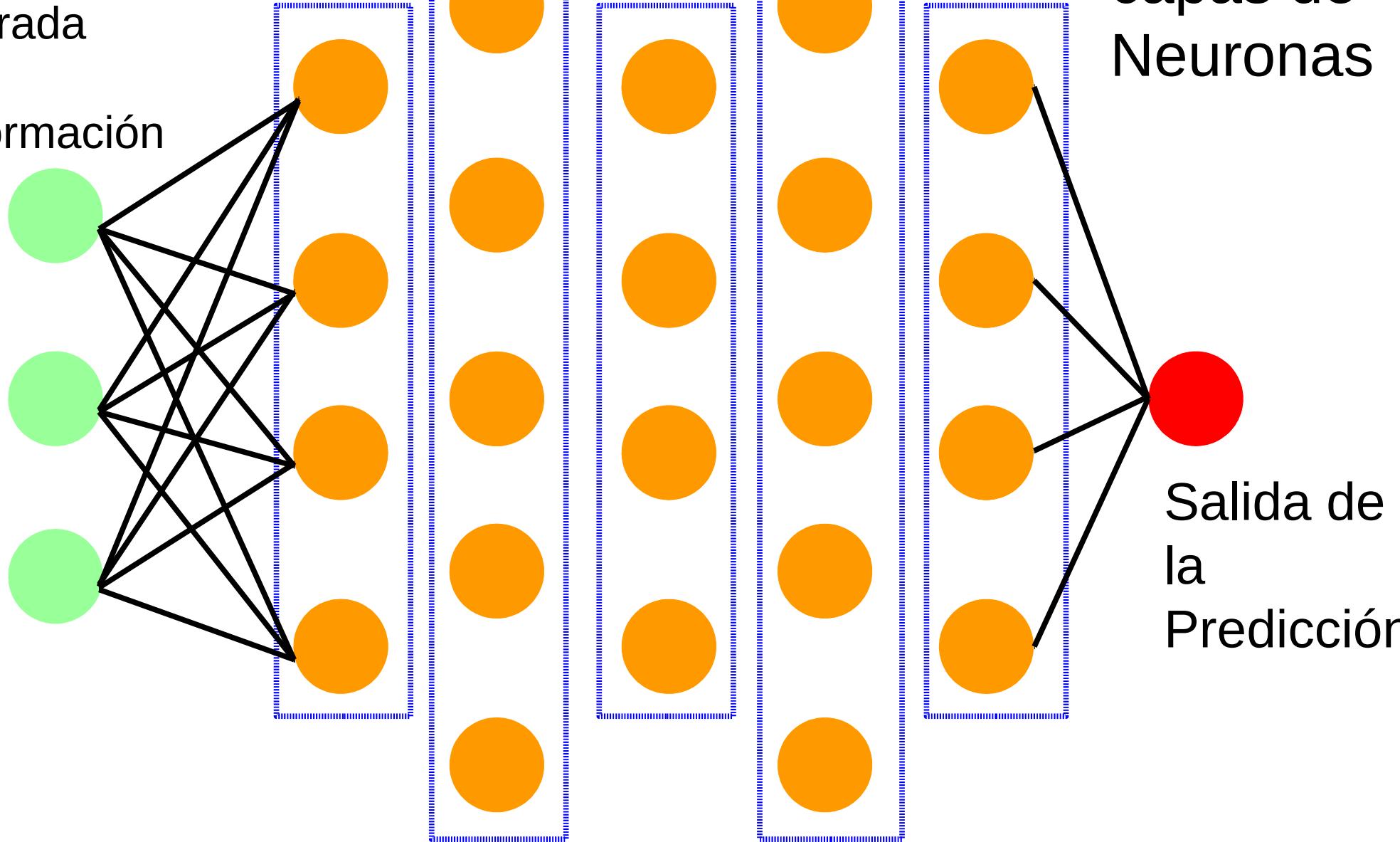
En realidad, el error estándar de la tarea de clasificación de un conjunto parecidos es mayor, ya que los datos de testeo han sido implícitamente usado por los diversos métodos para “tunear” los procedimientos.

# Redes Neuronales Artificiales



# Deep Learning

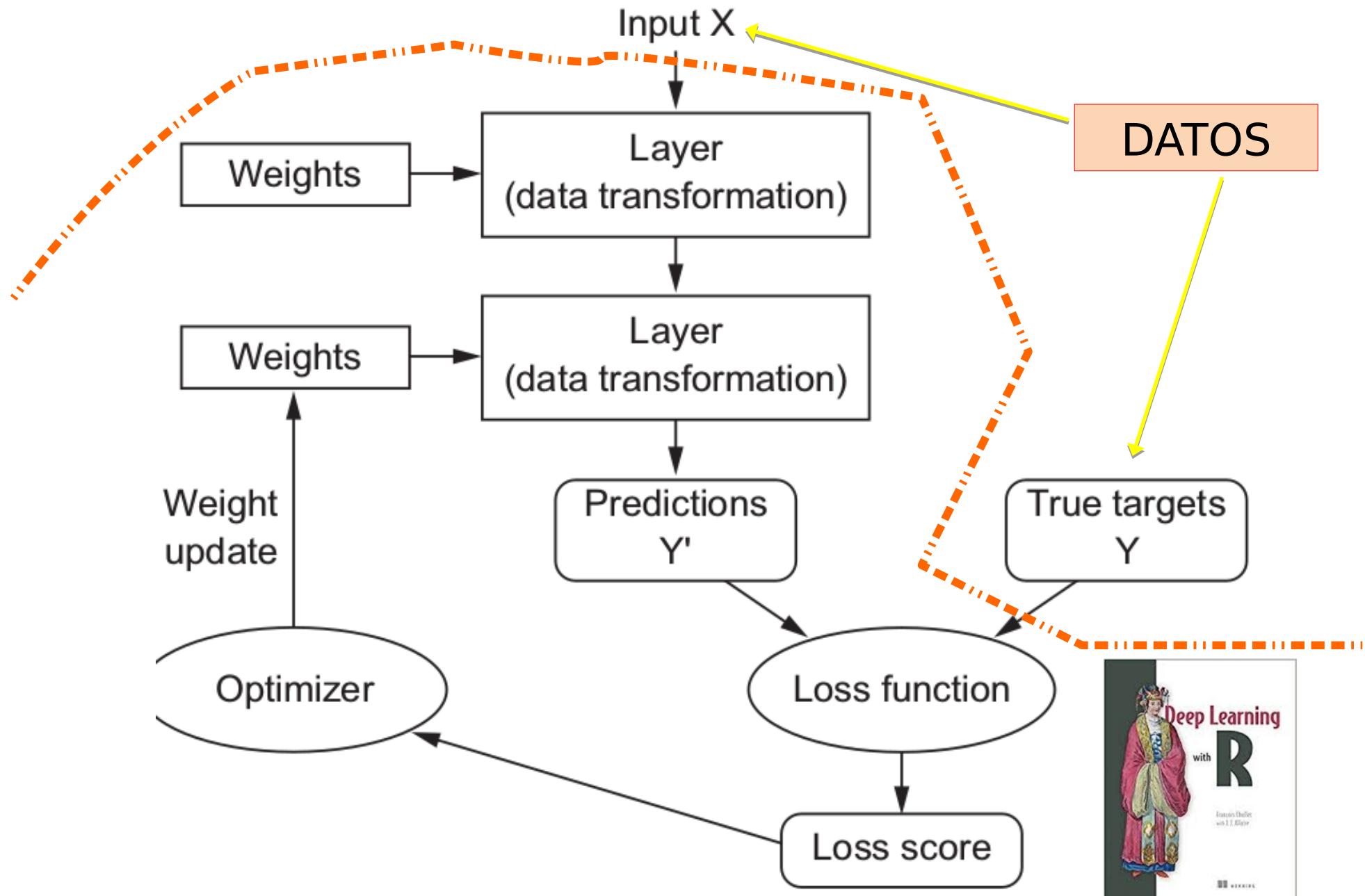
Entrada  
de  
Información



Múltiples  
capas de  
Neuronas

Salida de  
la  
Predicción

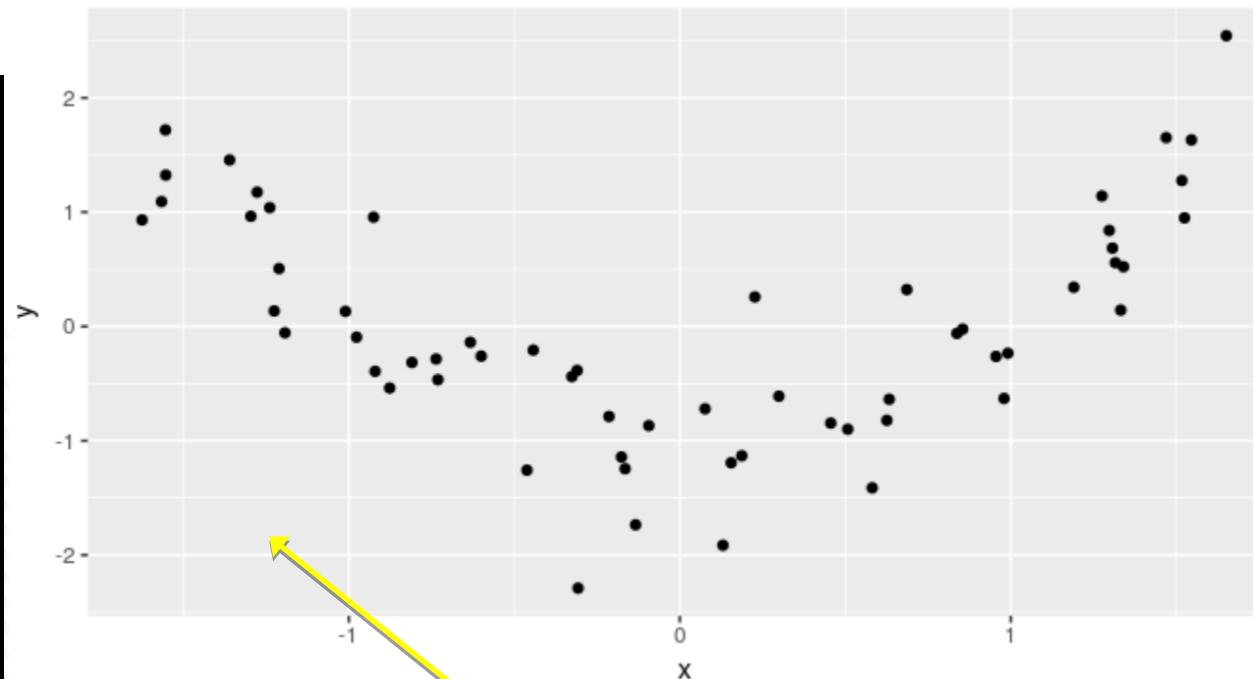
# Flujo de Entrenamiento de un Modelo de Deep Learning (ANN)



# Ejemplo muy Simple

## Regresión Lineal

```
# Data generation, a
set.seed(123)
n<-60
x<-runif(n)-0.5
ruido<-rnorm(n)/5
y<-3+4*x^2+ruido
# Scaling the dataset
x<-scale(x)
y<-scale(y)
datos<-data.frame(x=x, y=y)
# scatter plot
ggplot(datos, aes(x=x, y=y)) + geom_point()
```



# Tres Elementos Básicos de un Modelo en KERAS

Arquitectura  
del Modelo

```
modelo <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu",
              kernel_regularizer = regularizer_l1_l2(0,0),
              kernel_initializer='random_uniform',
              input_shape = c(1)) %>%
  layer_dense(units = 1, activation = "linear")
```

```
modelo %>% compile(
  optimizer = optimizer_sgd(lr=0.1, decay=0.01),
  loss = "mse",
  metrics = "mae")
```

Parámetros  
de  
Compilación

```
historia <- modelo %>%
  fit(x.train, y.train, batch_size = 16, epochs = 100,
       validation_data = list(x.test, y.test))
```

Training

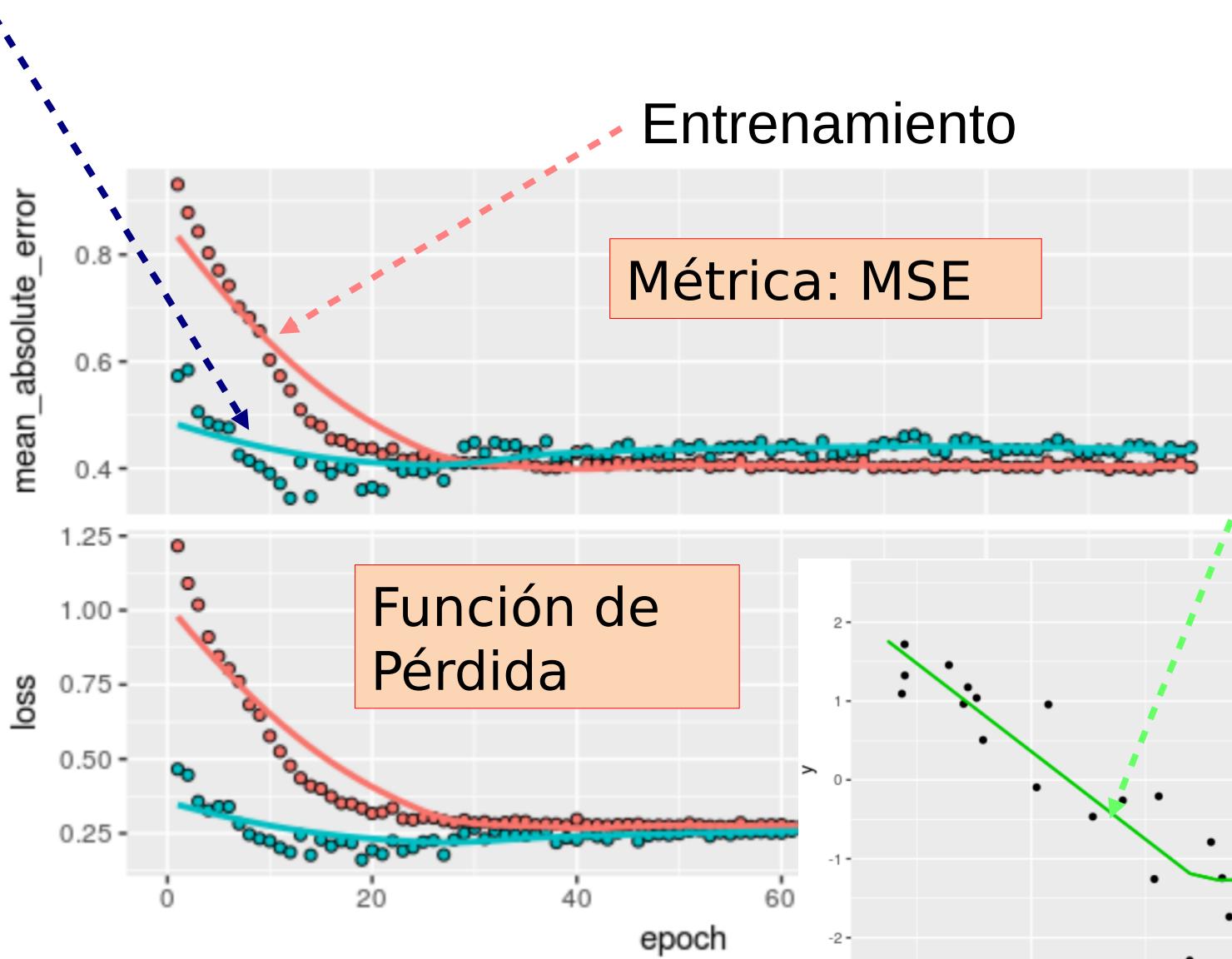
# El Modelo

```
modelo <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu",
              kernel_regularizer = regularizer_l1_l2(0,0),
              kernel_initializer='random_uniform',
              input_shape = c(1)) %>%
  layer_dense(units = 1, activation = "linear")
```

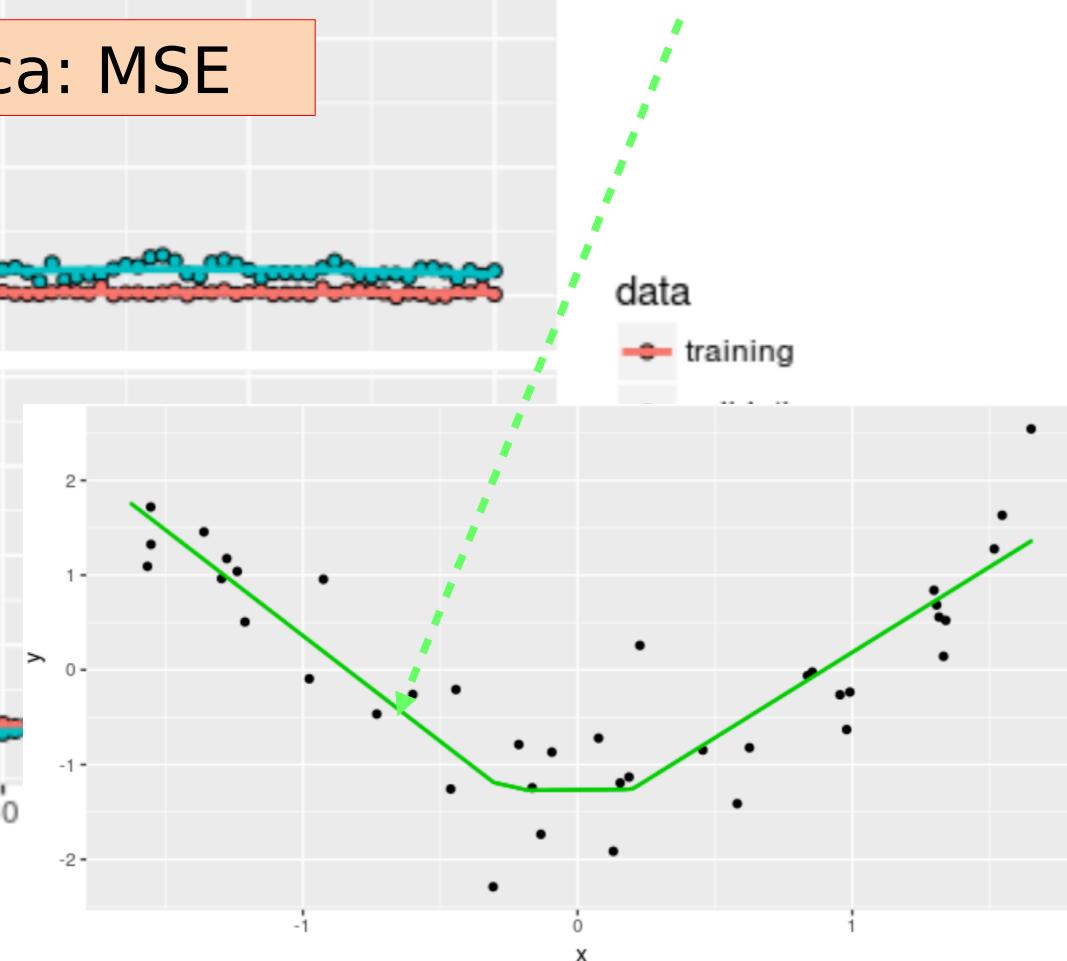
Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 256)	512
dense_4 (Dense)	(None, 1)	257
Total params:	769	
Trainable params:	769	
Non-trainable params:	0	

# Resultado del Ajuste

Validación



Función  
Predicha

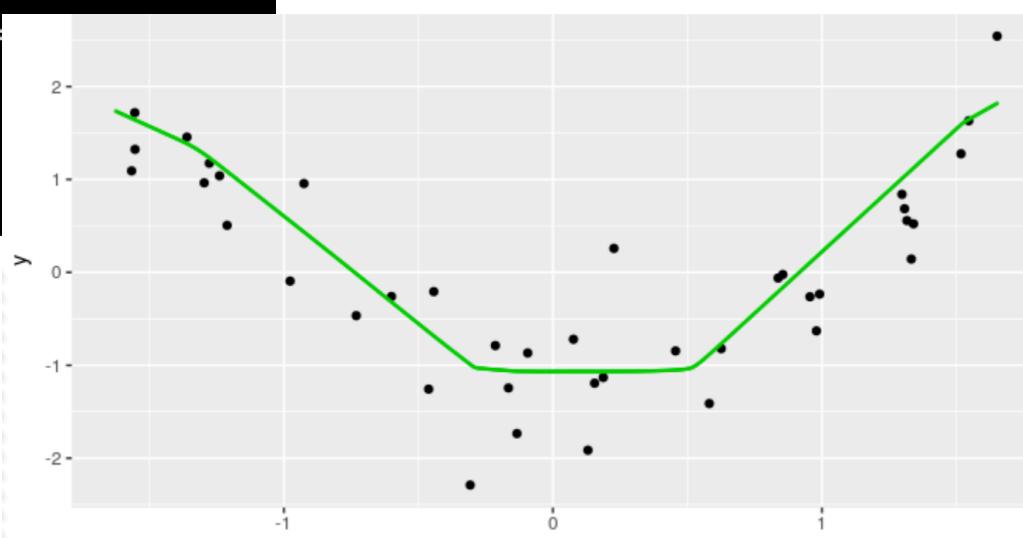
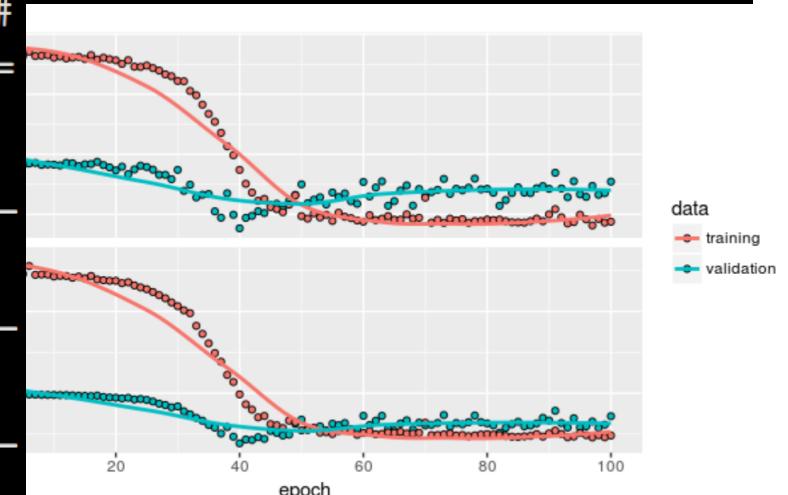


# Agregando Capas Conectadas

```
modelo <- keras_model_sequential() %>%
  layer_dense(units = 256, activation = "relu", kernel_regularizer = "l2", units = 128, activation = "relu", kernel_regularizer = "l2", units = 64, activation = "relu", kernel_regularizer = "l2", units = 1, activation = "linear")
```

Layer (type)	Output Shape	Param #
=====		
dense_5 (Dense)	(None, 256)	512
dense_6 (Dense)	(None, 128)	32896
dense_7 (Dense)	(None, 64)	8256
dense_8 (Dense)	(None, 1)	65
=====		

Total params: 41,729  
Trainable params: 41,729  
Non-trainable params: 0



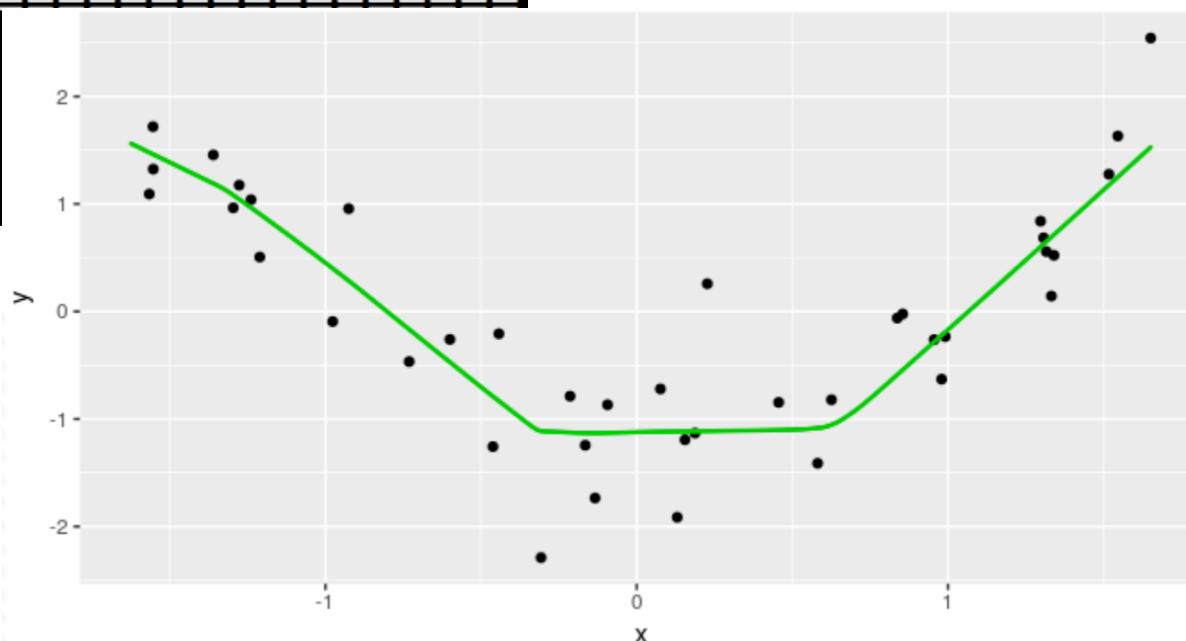
# Exagerando la Complejidad del Modelo

Layer (type)	Output Shape	Param #
=====		
dense_9 (Dense)	(None, 512)	1024
dense_10 (Dense)	(None, 512)	262656
dense_11 (Dense)	(None, 512)	262656
dense_12 (Dense)	(None, 1)	513
=====		

Total params: 526,849

Trainable params: 526,849

Non-trainable params: 0



# Modelos Convolucionales

Se utilizan para modelar el comportamiento de fenómenos provenientes de procesos que ocurren en un **dominio continuo** (por ej.: tiempo y espacio).

Aprovechan la **dependencia** existente entre los atributos (features), captando (aprendiendo) el patrón de dependencia recurrente que subyace al proceso.

# La Operación de Convolución

## Entrada

1	1	1	1	1	1
1					1
1					1
1					1
1	1	1	1	1	1
1	1				

Filtro 1

0	1	1	0	0
0	0	0	1	0
0	0	0	1	0
0	0	0	1	0
0	1	1	0	0

Filtro 2

1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

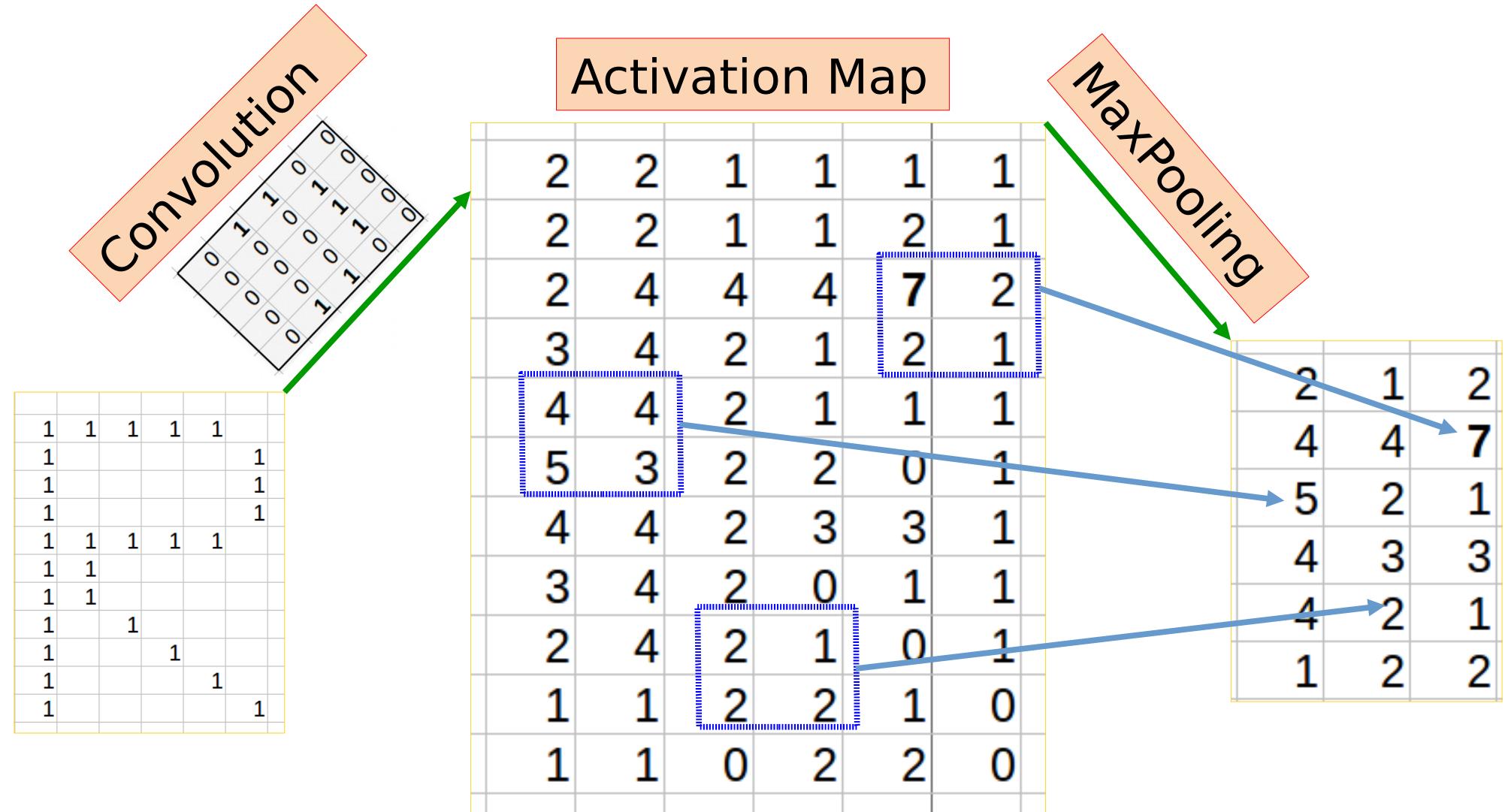
Salida  
de  
aplicar  
Filtro 1



Activation Map

2	2	1	1	1	1	1
2	2	1	1	1	2	1
2	4	4	4	4	7	2
3	4	2	1	2	1	1
4	4	2	1	1	1	1
5	3	2	2	0	1	1
4	4	2	3	3	1	1

# La Operación de “Pooling”



CONV + POOL + ... + CONV + POOL = ABSTRACCION

# Ejemplo de Clasificación en Series de Tiempo

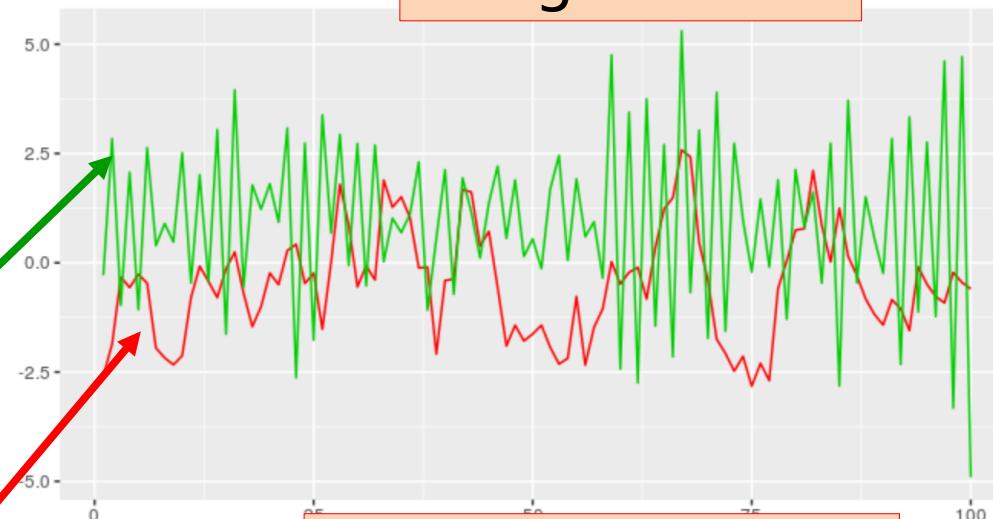
Ejemplo  
exagerado

- Dos clases de series:

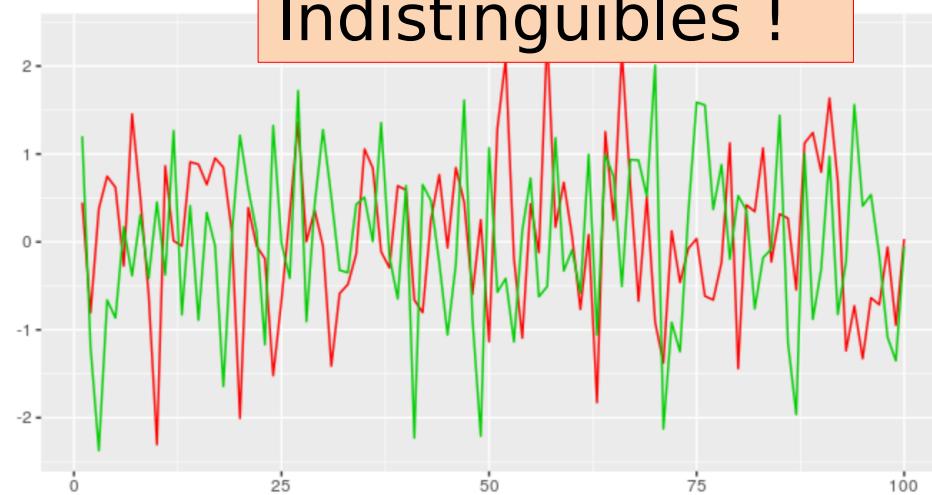
- Serie 1 (200 casos, 500 obs)
- Serie 2 (200 casos, 500 obs)

Autocor. =  $-10^{-8}$   
y Media = + 1/450

Autocor. =  $+10^{-8}$   
y Media = - 1/450



Ejemplo real  
Indistinguibles !



# Definición del Modelo

## Arquitectura del Modelo

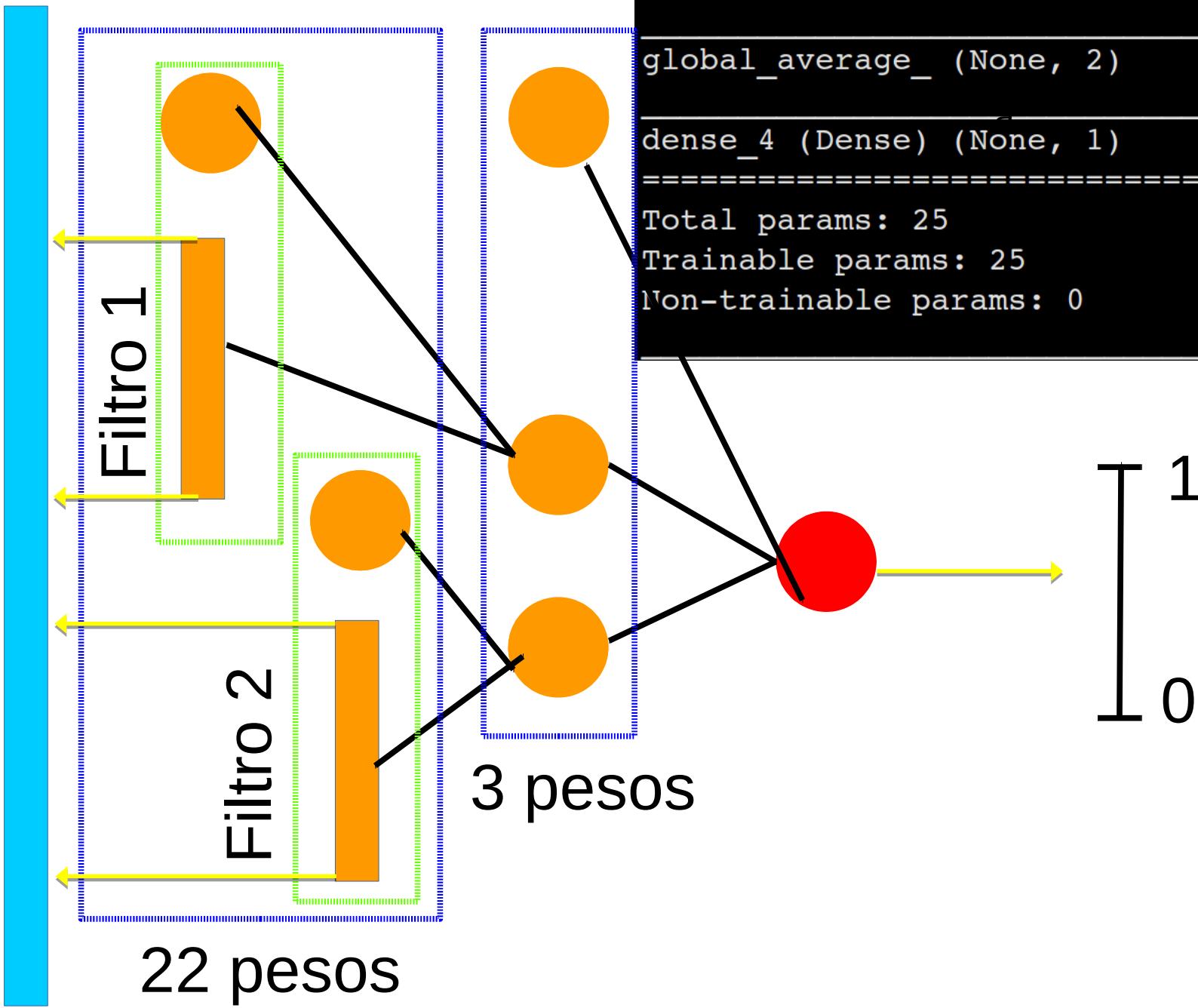
```
ventana <- 5  
modelo <- keras_model_sequential()  
modelo %>%  
  layer_conv_1d(filters = 2, kernel_size = ventana,  
                 activation = 'relu',  
                 input_shape = c(largo,1)) %>%  
  layer_global_average_pooling_1d() %>%  
  layer_dense(units = 1, activation = 'sigmoid')  
modelo %>% compile(  
  loss = 'binary_crossentropy',  
  optimizer = 'rmsprop',  
  metrics = c('accuracy'))  
historia <- modelo %>%  
  fit(array.train, label.train, batch_size = 16,  
       epochs = 250, validation_split = 0.2)
```

## Parámetros de Compilación

## Training

# El Modelo

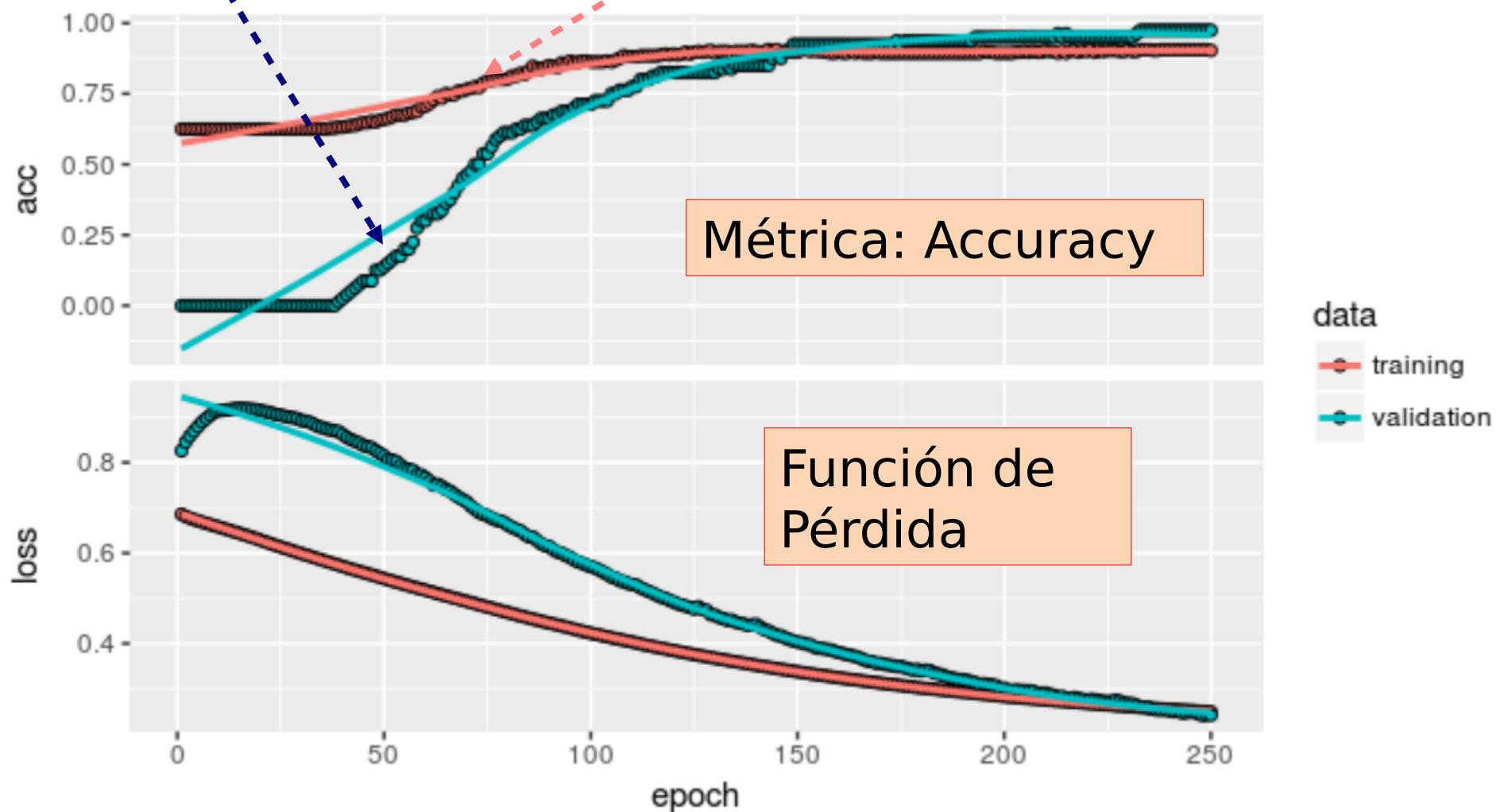
ENTRADA



# El Ajuste

Validación

Entrenamiento



# Desempeño de la Clasificación

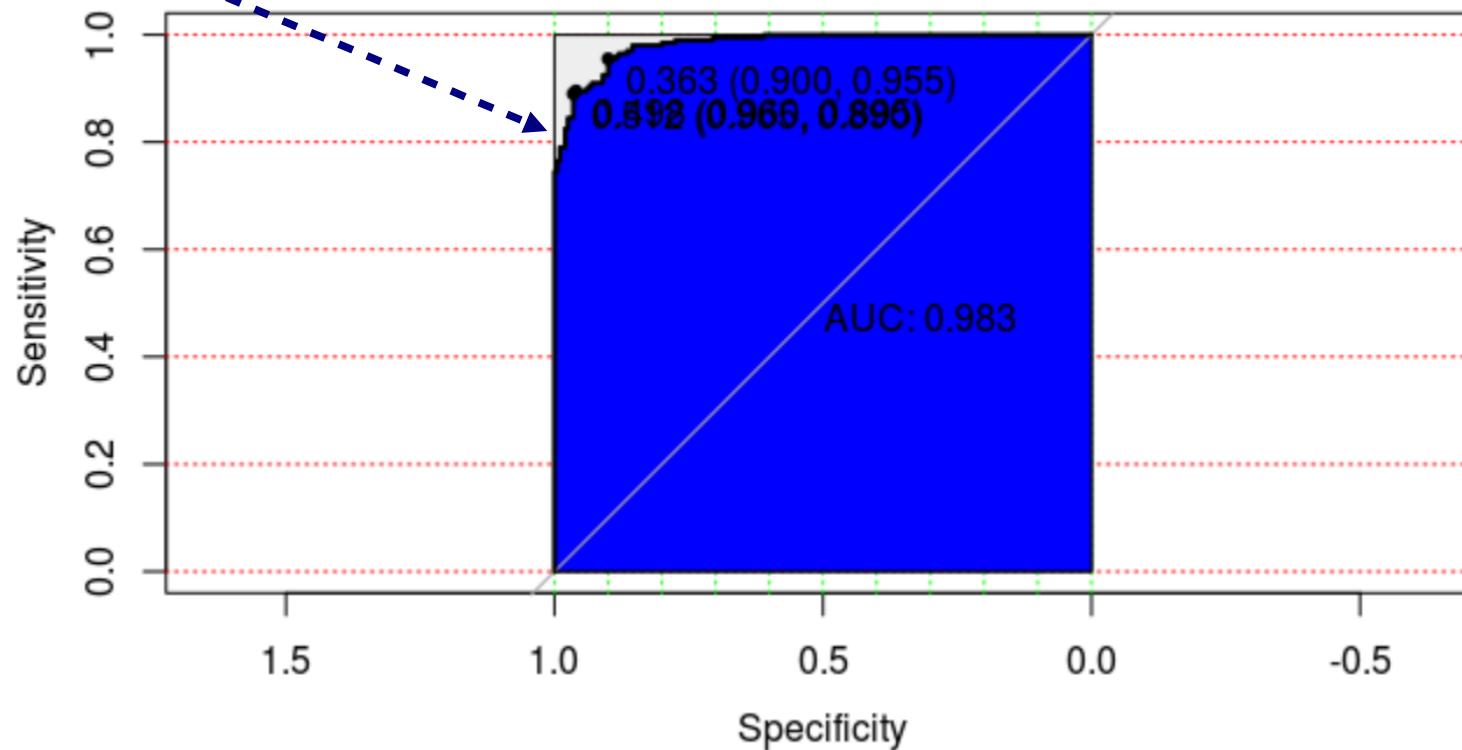
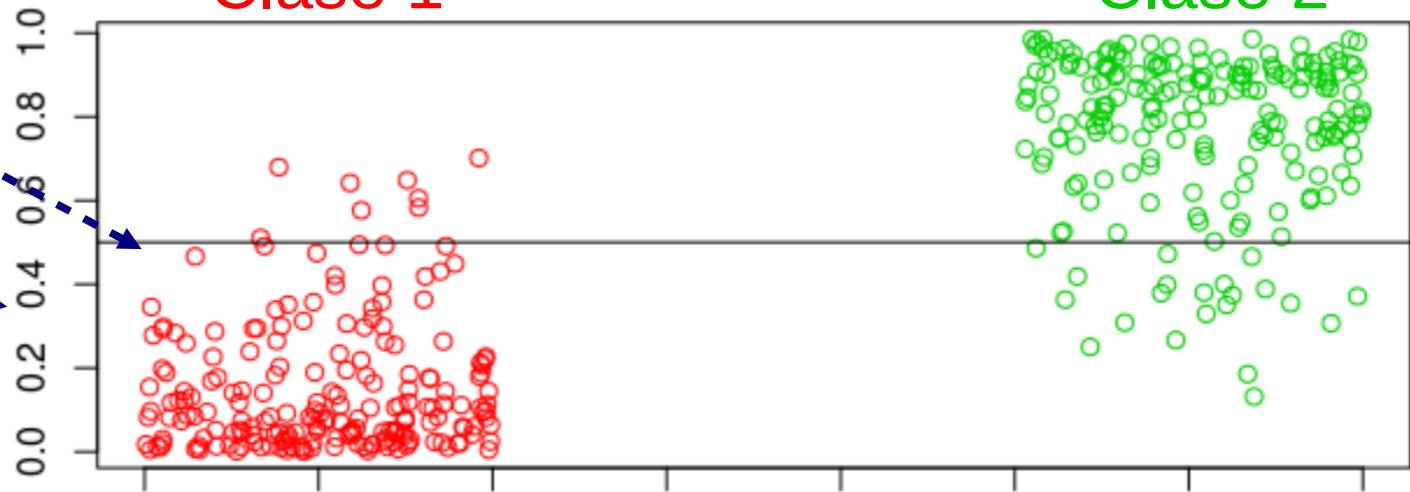
Umbral de Clasificación

Score de Clasificación

Curva ROC

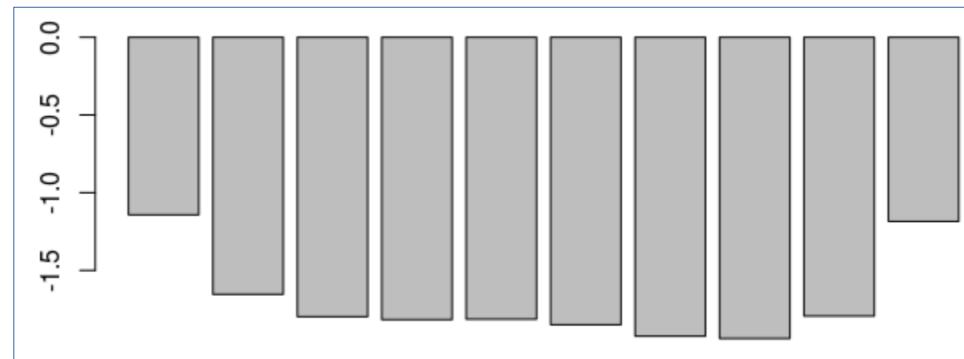
Clase 1

Clase 2

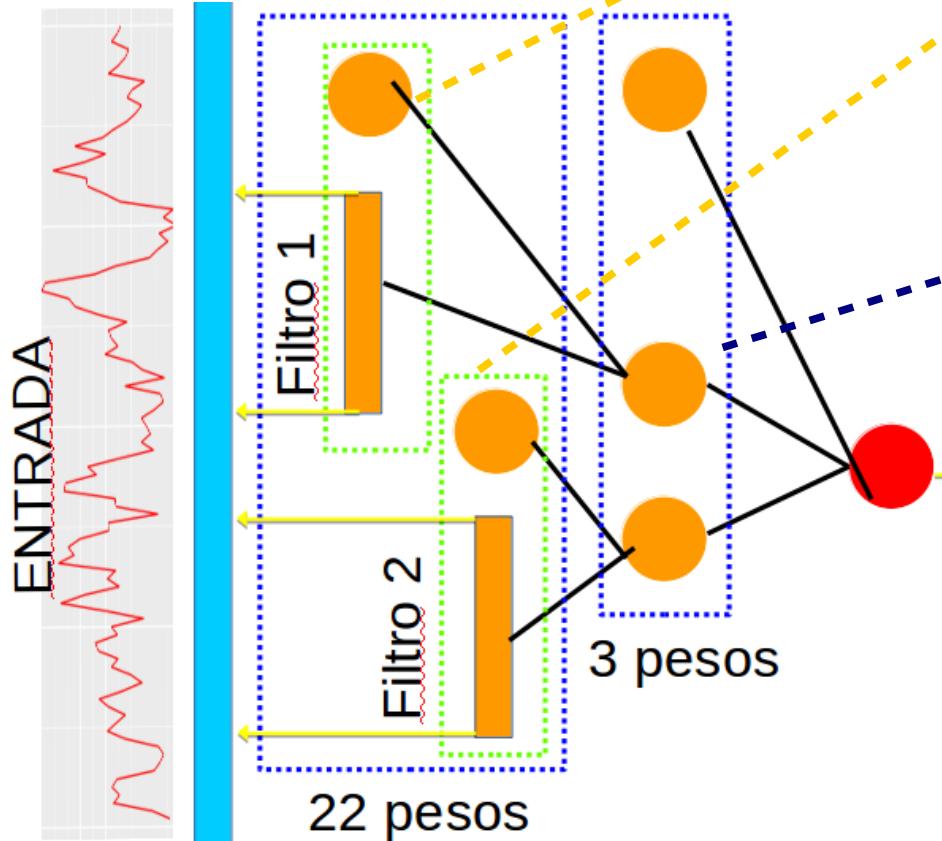
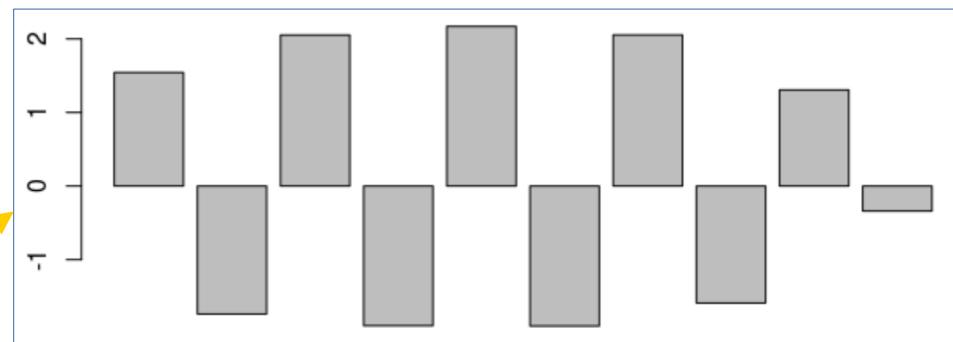


# Que Aprendió la Red ?

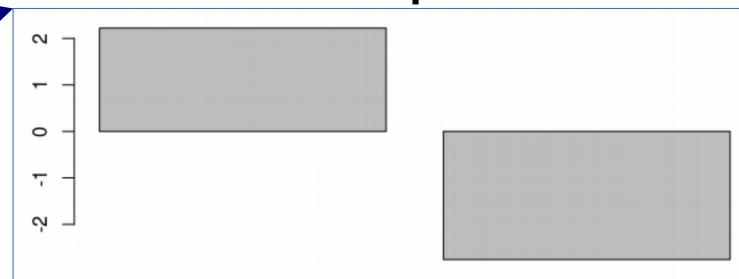
## Filtro 1: Patrón de Posición



## Filtro 2: Patrón de Autocorrelación

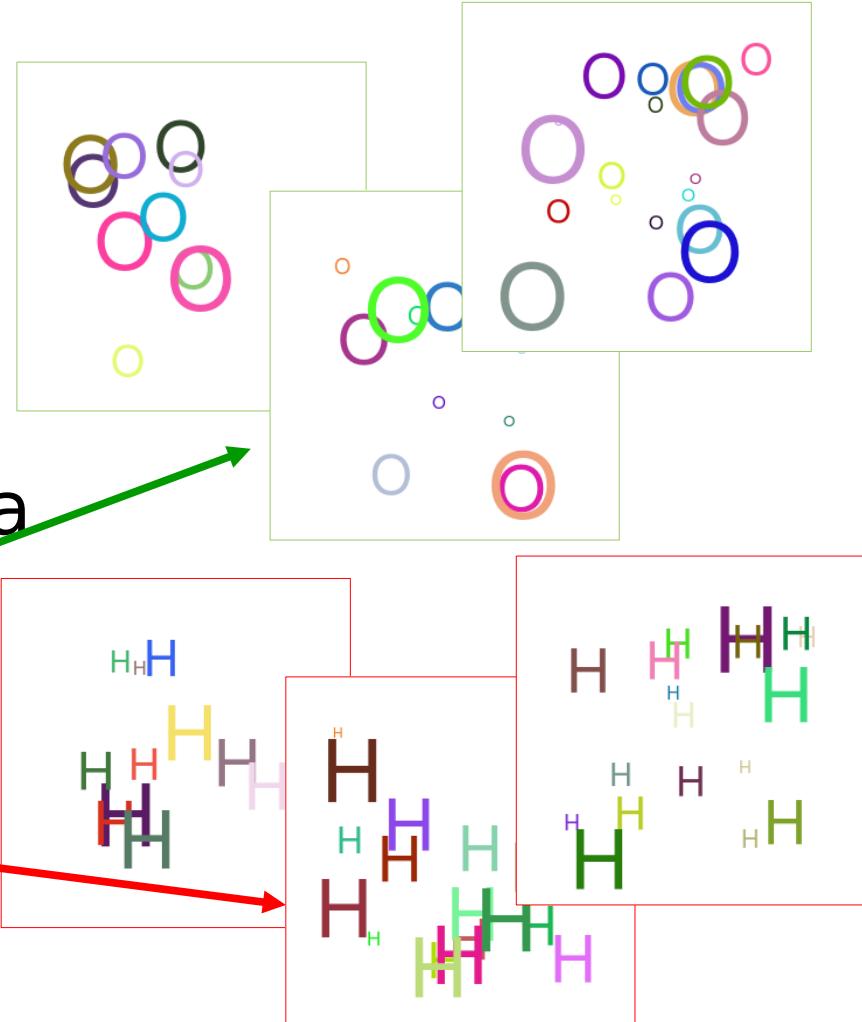


## Ultima Capa



# Ejemplo de Clasificación de Imágenes

- Imágenes generadas:
  - Posición aleatoria
  - Tamaño aleatorio
  - Cantidad de letras aleatoria
- Dos clases de imágenes:
  - Clase 1 (100 casos)
  - Clase 2 (100 casos)



# Definición del Modelo

```
modelo %>%
  layer_conv_2d(filters = filtros, kernel_size = c(vent,vent),
                 activation = 'relu',
                 input_shape = c(resol,resol,1)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = filtros, kernel_size = c(vent,vent),
                 activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = filtros, kernel_size = c(vent,vent),
                 activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_flatten() %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 1, activation = 'sigmoid')
```

Arquitectura  
del Modelo

```
modelo %>% compile(
  loss = 'binary_crossentropy',
  optimizer = RMSprop(),
  metrics = c('accuracy'))
```

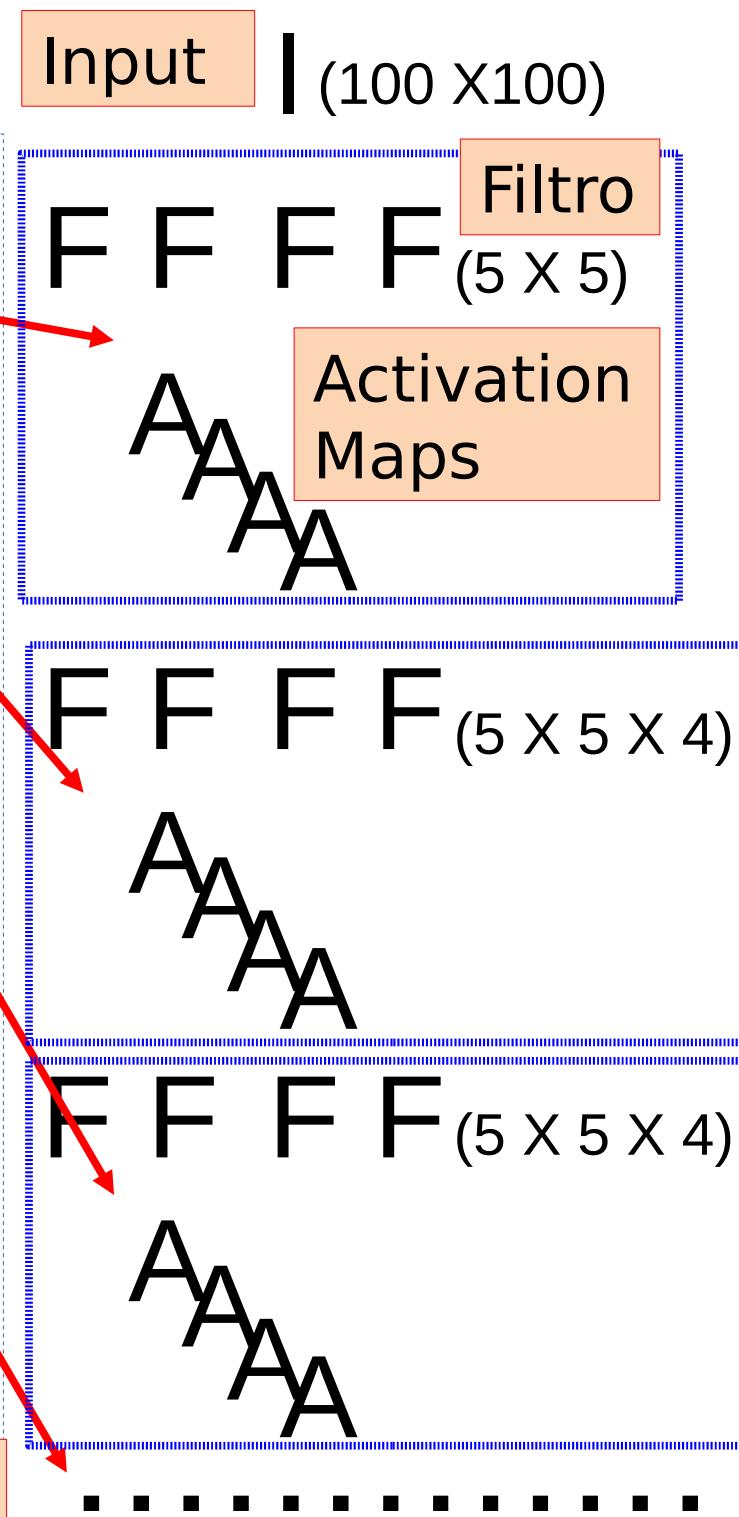
Parámetros  
de  
Compilación

```
historia <- modelo %>%
  fit(array.train, label.train, batch_size = 1,
      epochs = 25, validation_data = list(array.test, label.test))
```

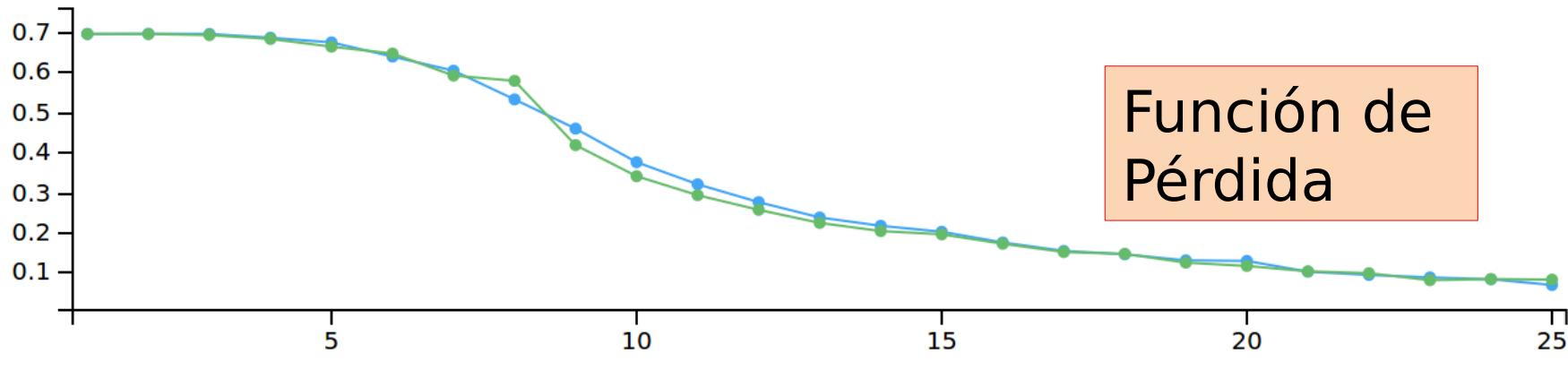
# Arquitectura del Modelo

Layer (type)	Output Shape	Param #
conv2d_77 (Conv2D)	(None, 96, 96, 4)	104
max_pooling2d_77 (MaxPooling2D)	(None, 48, 48, 4)	0
conv2d_78 (Conv2D)	(None, 44, 44, 4)	404
max_pooling2d_78 (MaxPooling2D)	(None, 22, 22, 4)	0
conv2d_79 (Conv2D)	(None, 18, 18, 4)	404
max_pooling2d_79 (MaxPooling2D)	(None, 9, 9, 4)	0
flatten_40 (Flatten)	(None, 324)	0
dropout_40 (Dropout)	(None, 324)	0
dense_38 (Dense)	(None, 1)	325
Total params: 1,237		
Trainable params: 1,237		

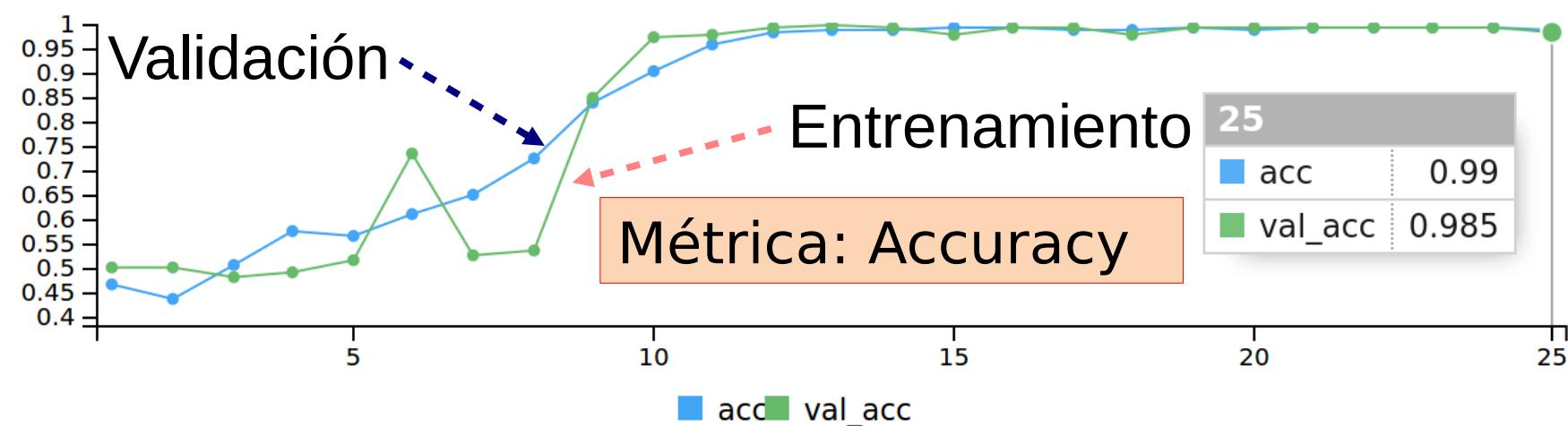
Flatten



# El Ajuste



Función de  
Pérdida



Validación

Entrenamiento

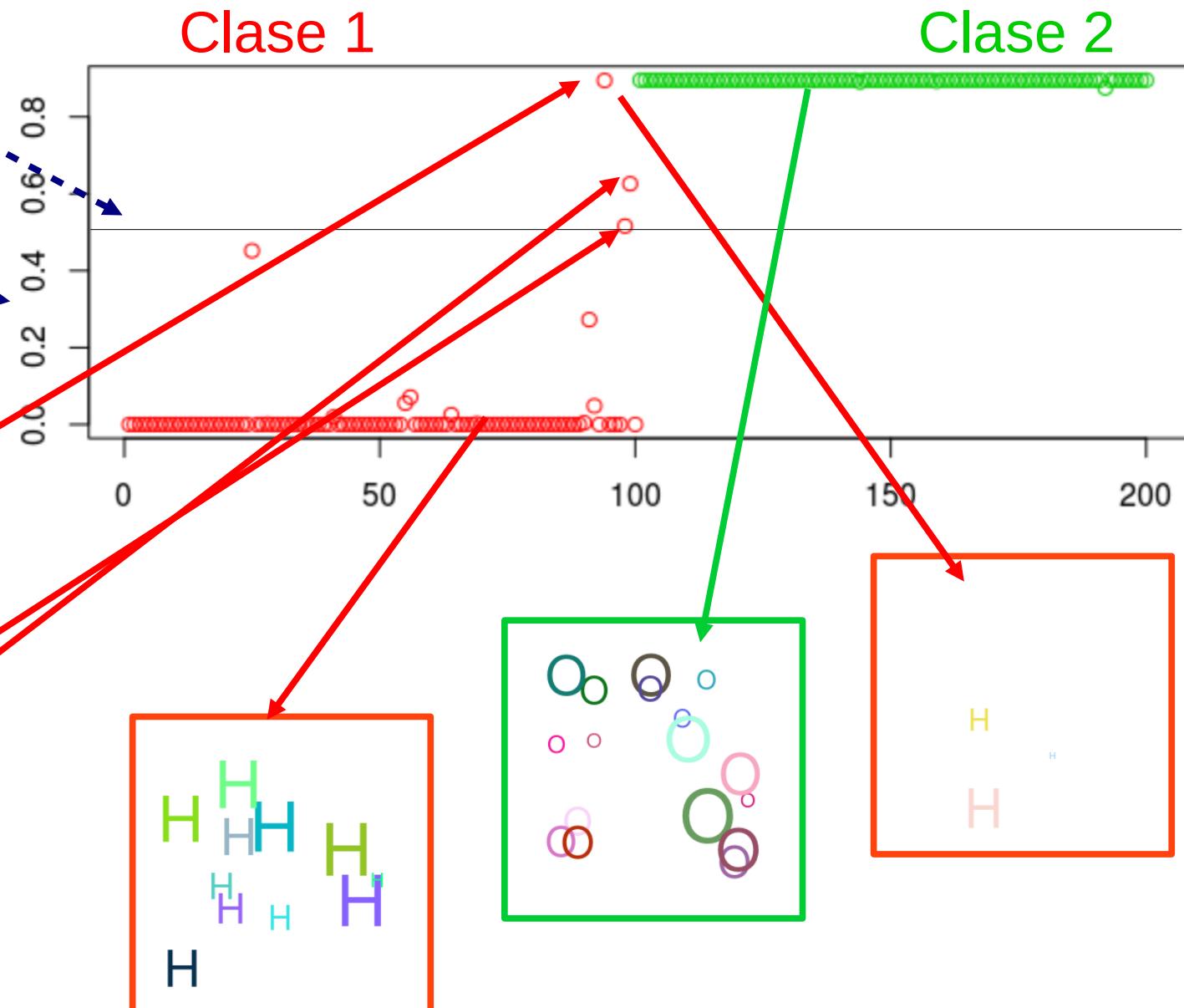
Métrica: Accuracy

# Desempeño de la Clasificación

Umbral de Clasificación

Score de Clasificación

	label.test
0.0484	0
0.0000	0
0.8947	0
0.0000	0
0.0000	0
0.0000	0
0.5162	0
0.6261	0
0.0000	0
0.8955	1
0.8955	1



# Definición del Modelo

```
modelo %>%
  layer_conv_2d(filters = filtros, kernel_size = c(vent,vent),
                 activation = 'relu',
                 input_shape = c(resol,resol,1)) %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = filtros, kernel_size = c(vent,vent),
                 activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_conv_2d(filters = filtros, kernel_size = c(vent,vent),
                 activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_flatten() %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 1, activation = 'sigmoid')
```

Arquitectura  
del Modelo

```
modelo %>% compile(
  loss = 'binary_crossentropy',
  optimizer = RMSprop(),
  metrics = c('accuracy'))
```

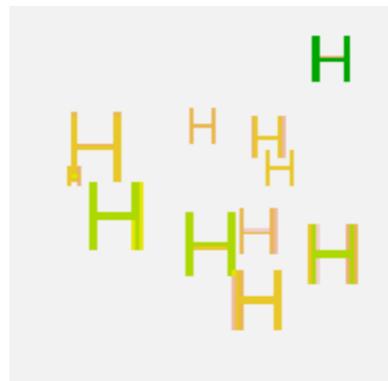
Parámetros  
de  
Compilación

```
historia <- modelo %>%
  fit(array.train, label.train, batch_size = 1,
      epochs = 25, validation_data = list(array.test, label.test))
```

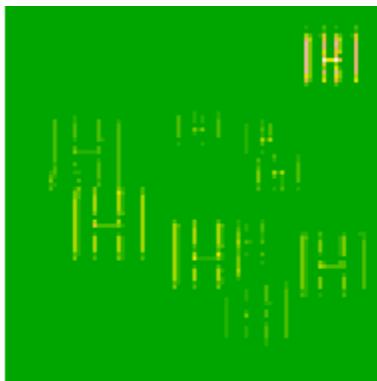
# Que Aprendió la Red ?

Mapas de Activación de la Primera Capa

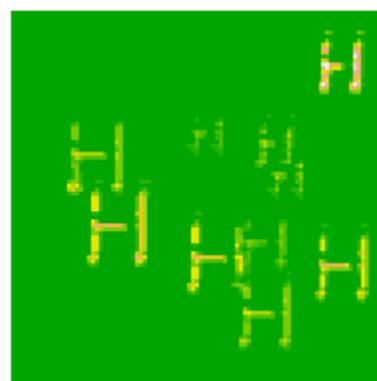
Clase 1



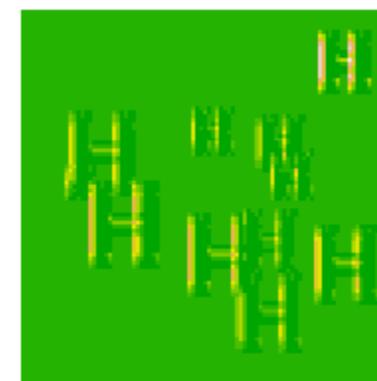
Filtro 1



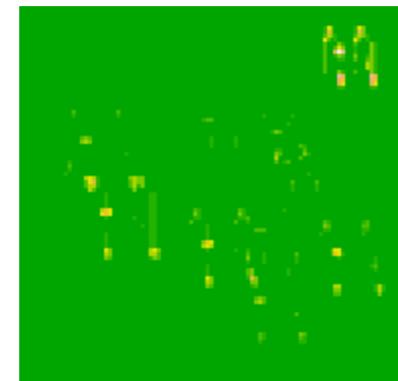
Filtro 2



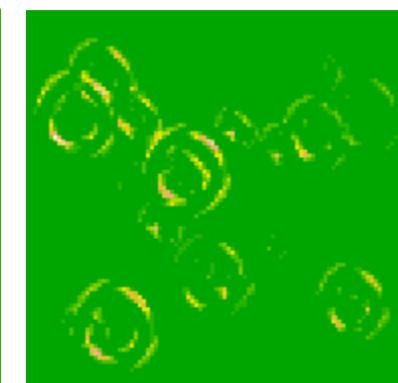
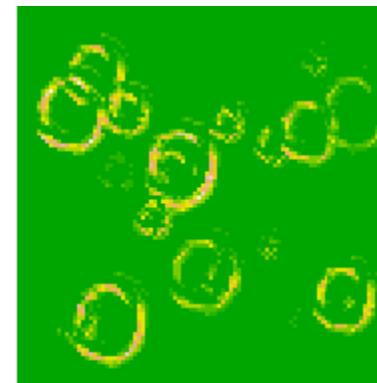
Filtro 3



Filtro 4



Clase 2



# Autoencoders

- Son modelos de **ANNs** para reducción de dimensión en datos dimensionalmente grandes.
- Son útiles para:
  - Detectar Errores (**Quality Control**)
  - Comprimir Información
  - Detección de Novedades (**Novelty Detection**)
  - Representación **Gráfica** de Datos

Autoencoder

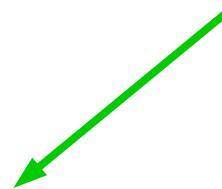
$$X \approx f_k(X)$$

# Inteligencia Artificial para la Detección de Fraudes en Seguros

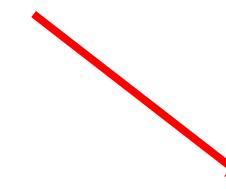
Una Aplicación para el Seguro  
Automotor

# ¿ Que es un Fraude en Seguros ?

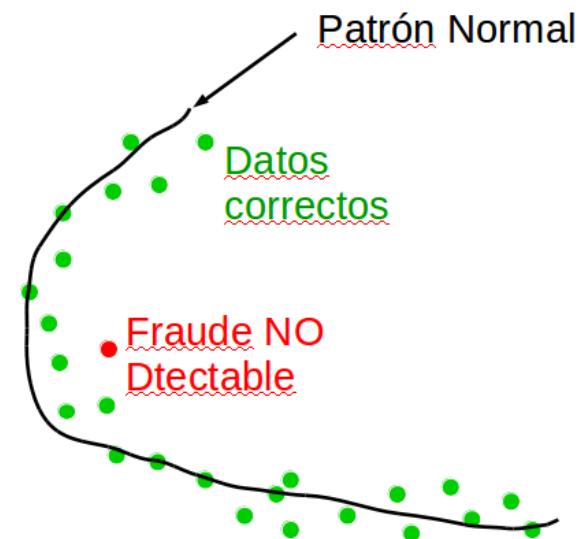
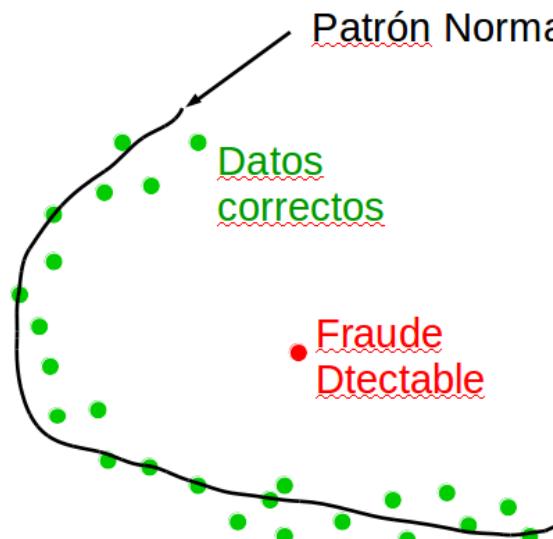
Es una **alteración del dato** que produce un perjuicio económico a la aseguradora



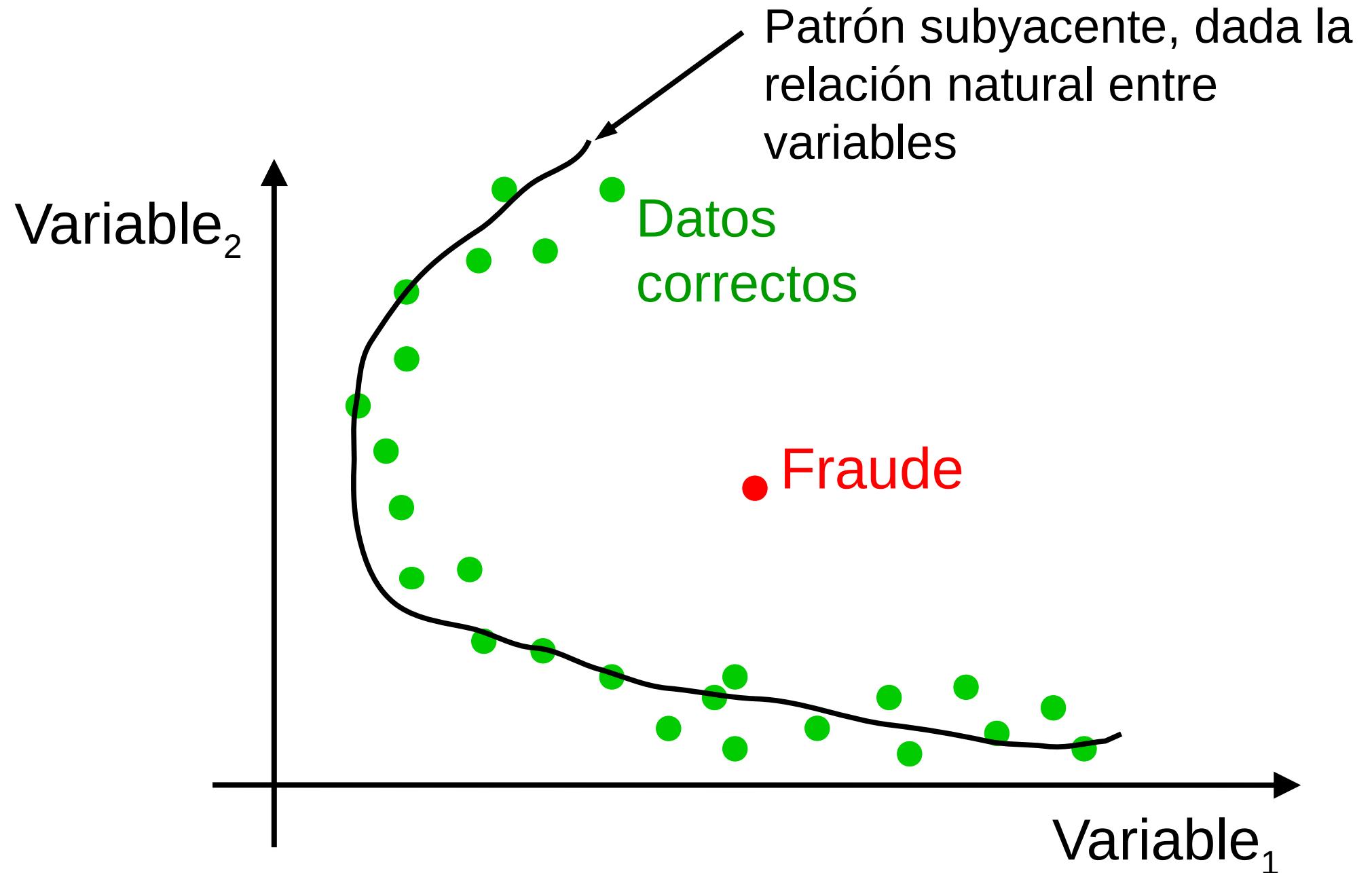
Dato **SE ALEJA** del patrón normal



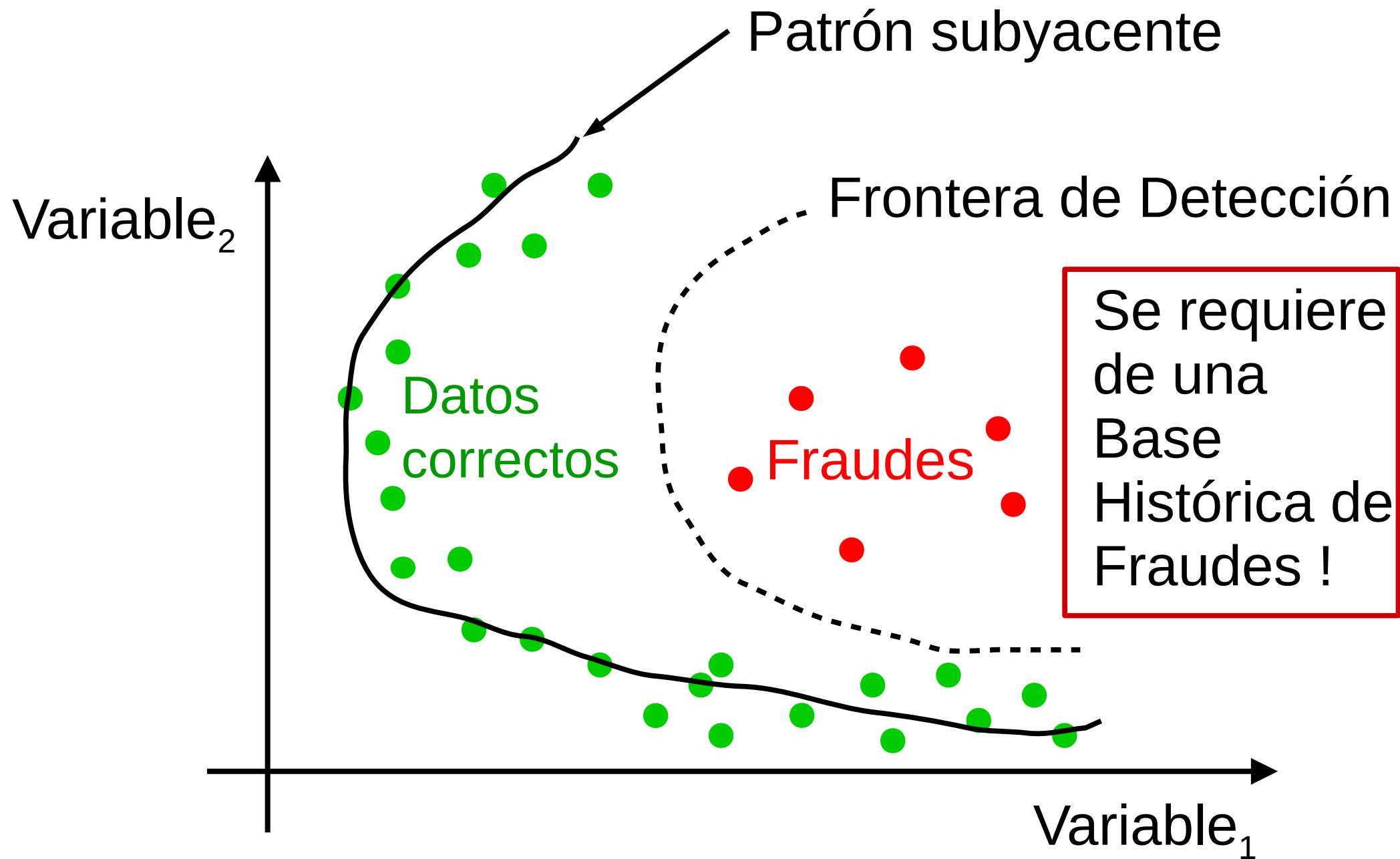
Dato **NO SE ALEJA** del patrón normal



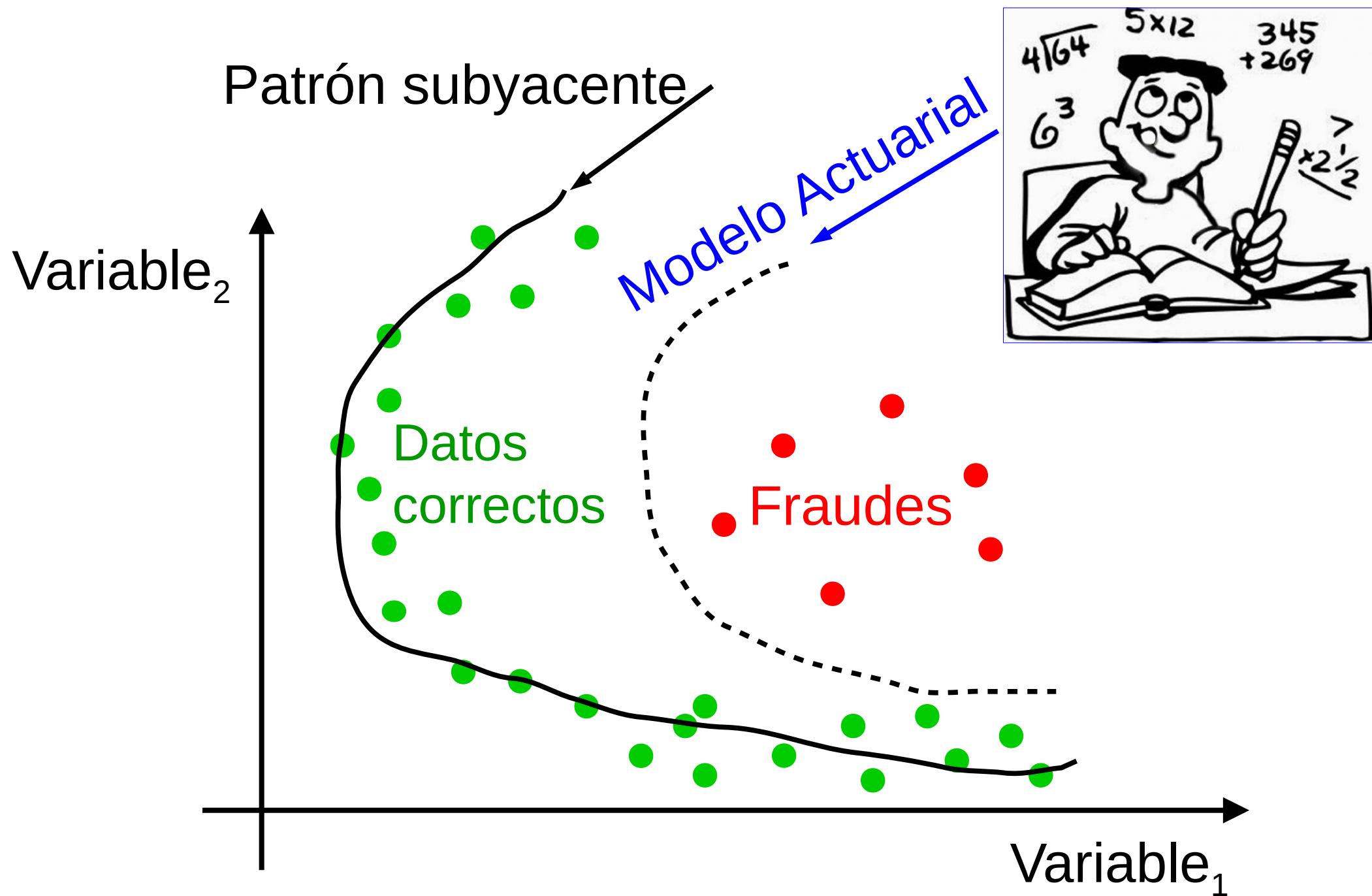
# ¿ Como se Vería un Fraude ?



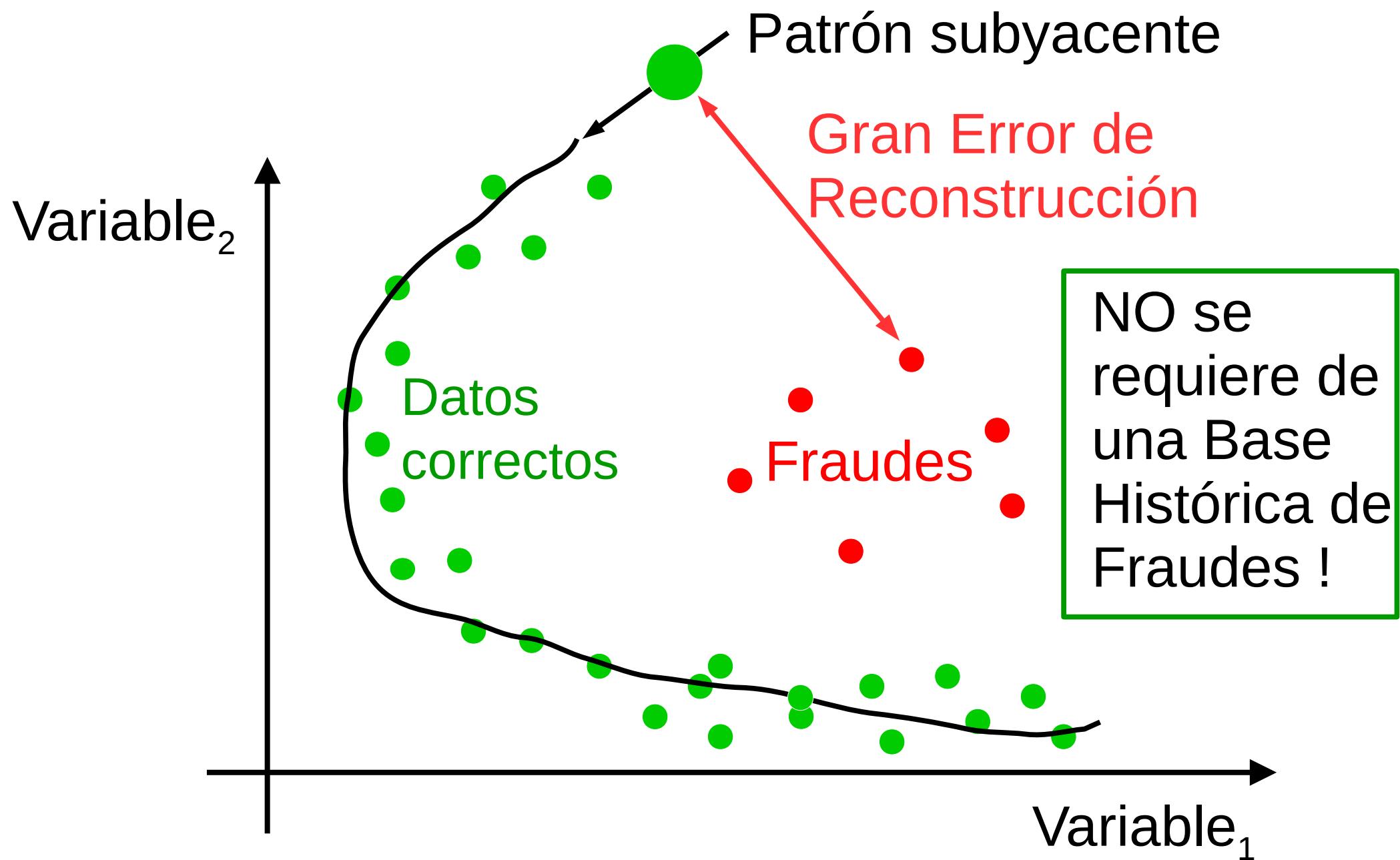
# Enfoque Supervisado para la Detección de Fraudes



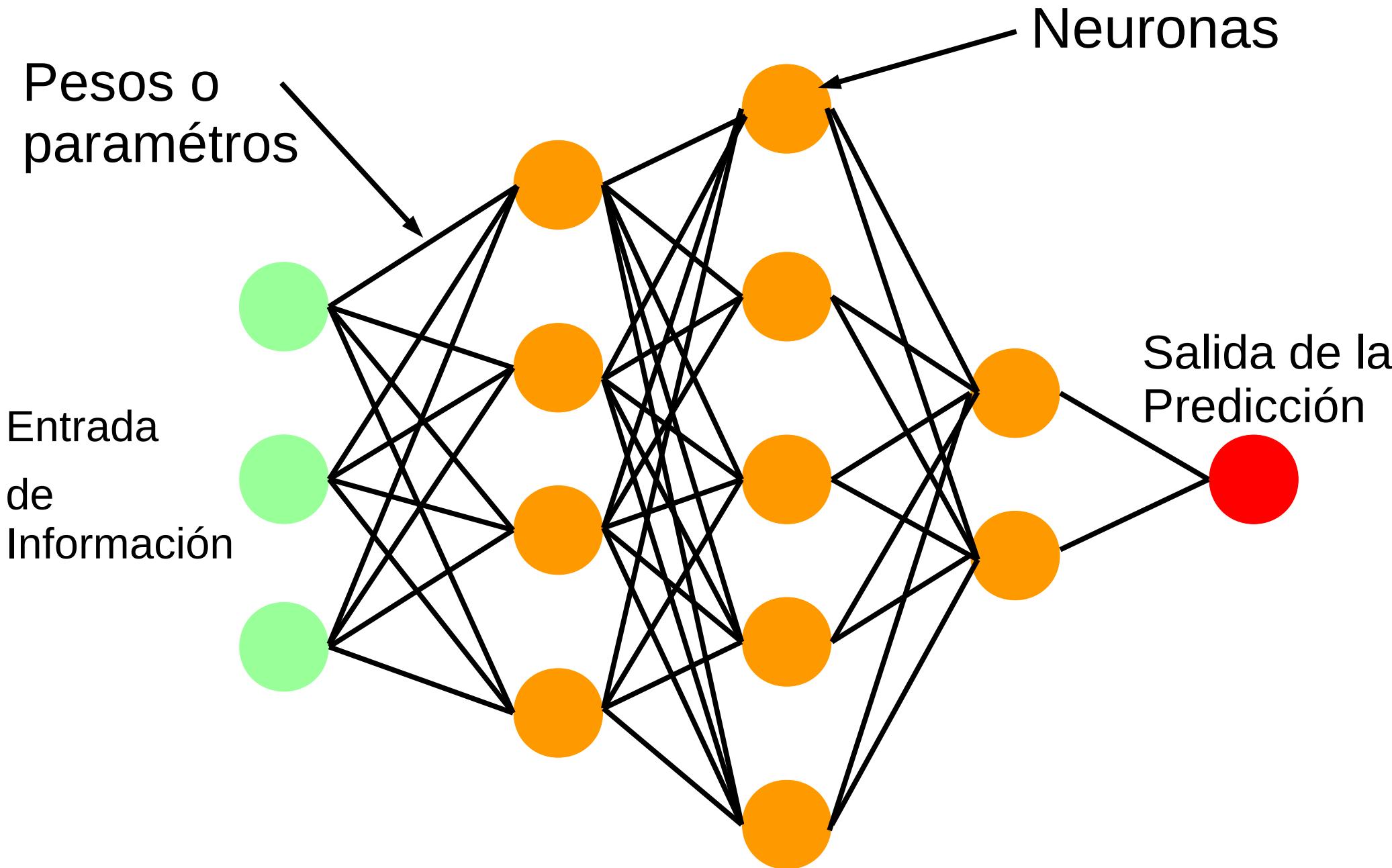
# ¿ Como se Trabajaba Antiguamente ?



# Enfoque No Supervisado

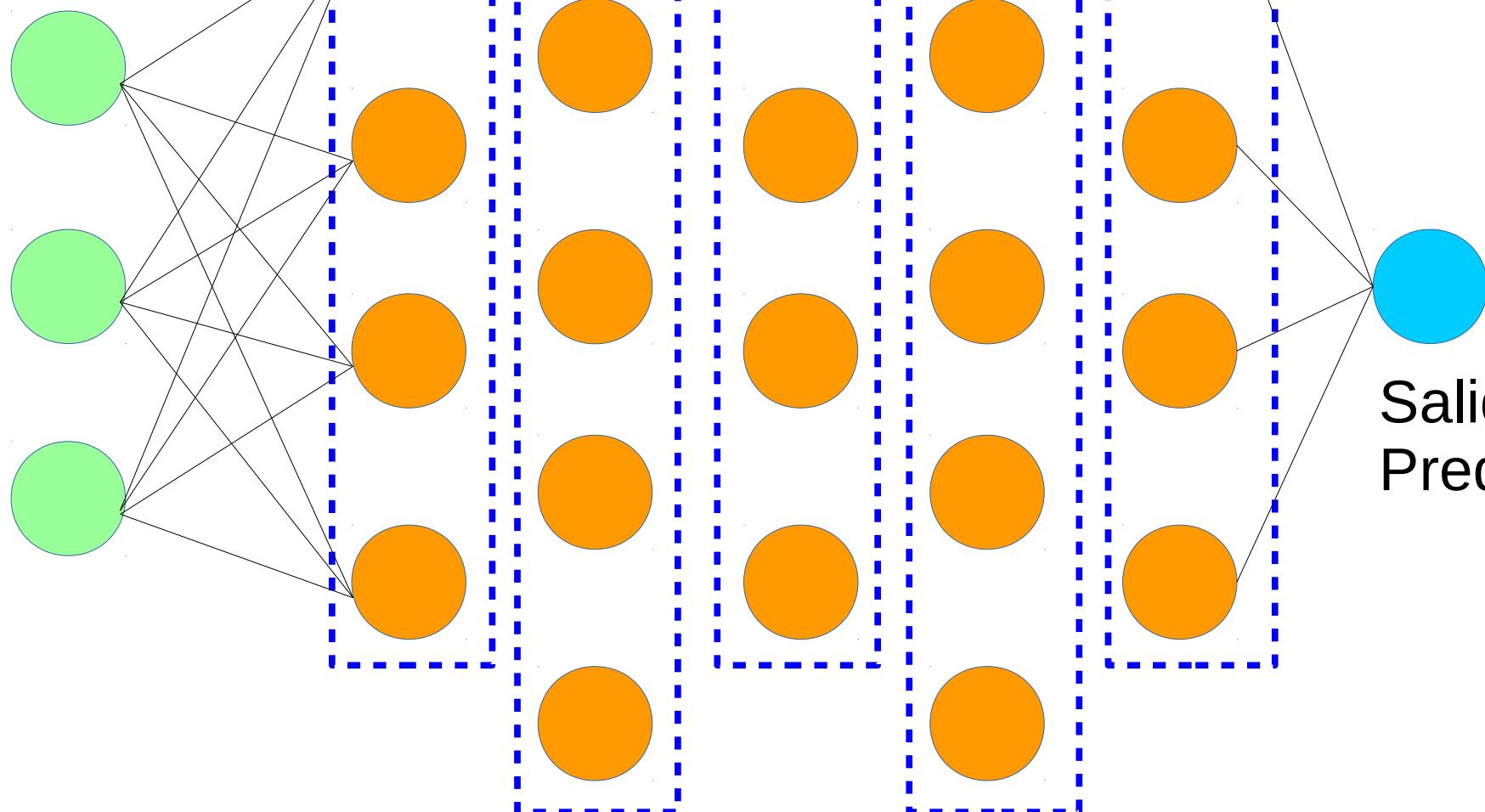


# Redes Neuronales Artificiales



# Deep Learning

Entrada  
de  
Información



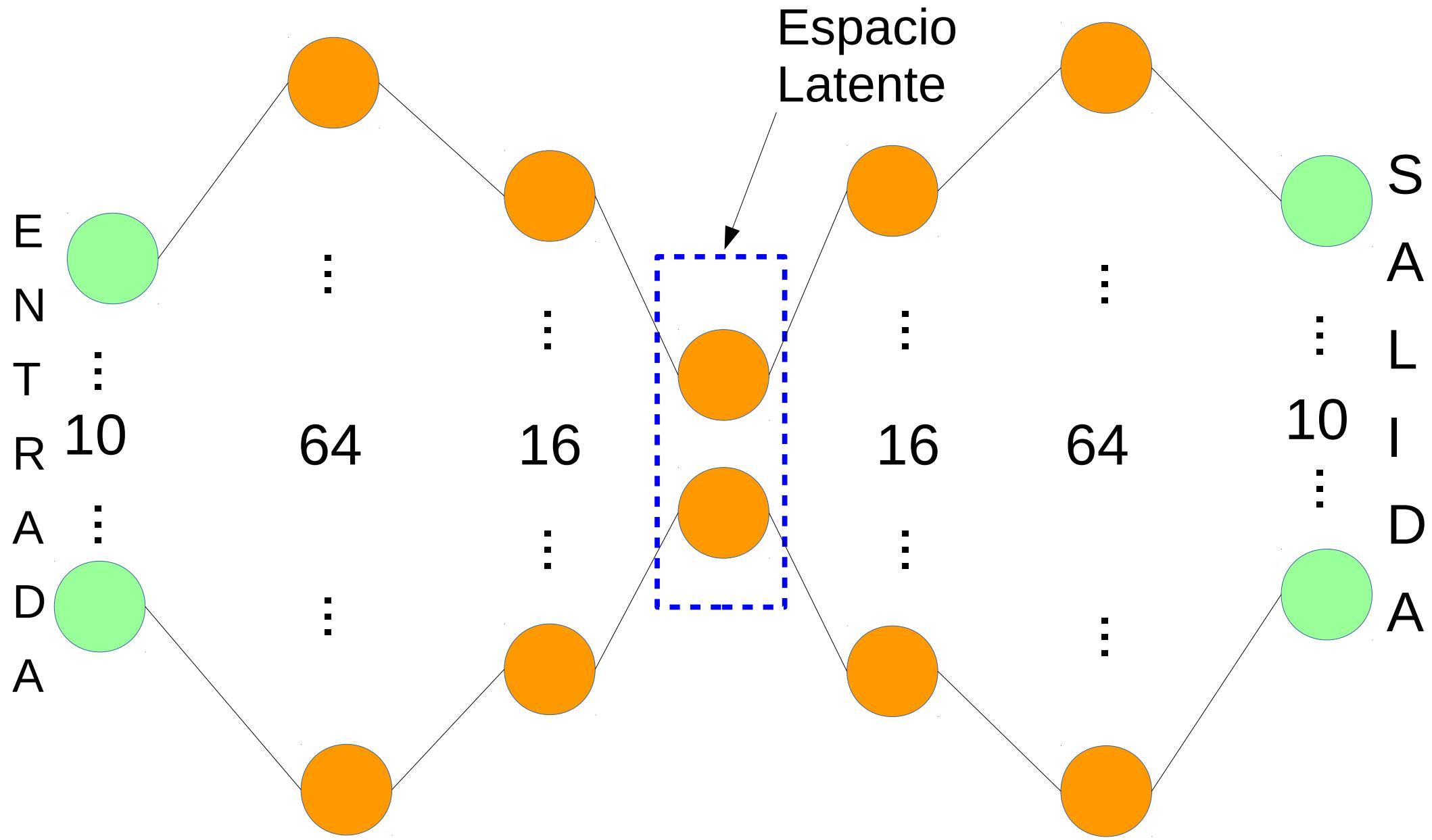
Múltiples  
capas de  
Neuronas

Salida de la  
Predicción

# Modelo Entrenado

75 % de la información capturada

3.654 parámetros calibrados



# Aplicación al Seguro Automotor



Superintendência  
de Seguros Privados

- AUTOSEG (Sistema de Estadística de Automóviles, Brasil)
- Total de producción año 2011 (con siniestros)
- Cada registro es una póliza
- Variables:
  - Características del auto
  - Características del conductor
  - Parámetros de la cobertura
  - Detalles del siniestro

10 Variables  
Predictoras de  
Fraude

Robo (Frecuencia)  
Colisión Total (Frecuencia)  
Colisión Parcial (Frecuencia)  
Incendio (Frecuencia)  
Otros (Frecuencia)  
Robo (Intensidad)  
Colisión Total (Intensidad)  
Colisión Parcial (Intensidad)  
Incendio (Intensidad)  
Otros (Intensidad)

# El Autoencoder en KERAS

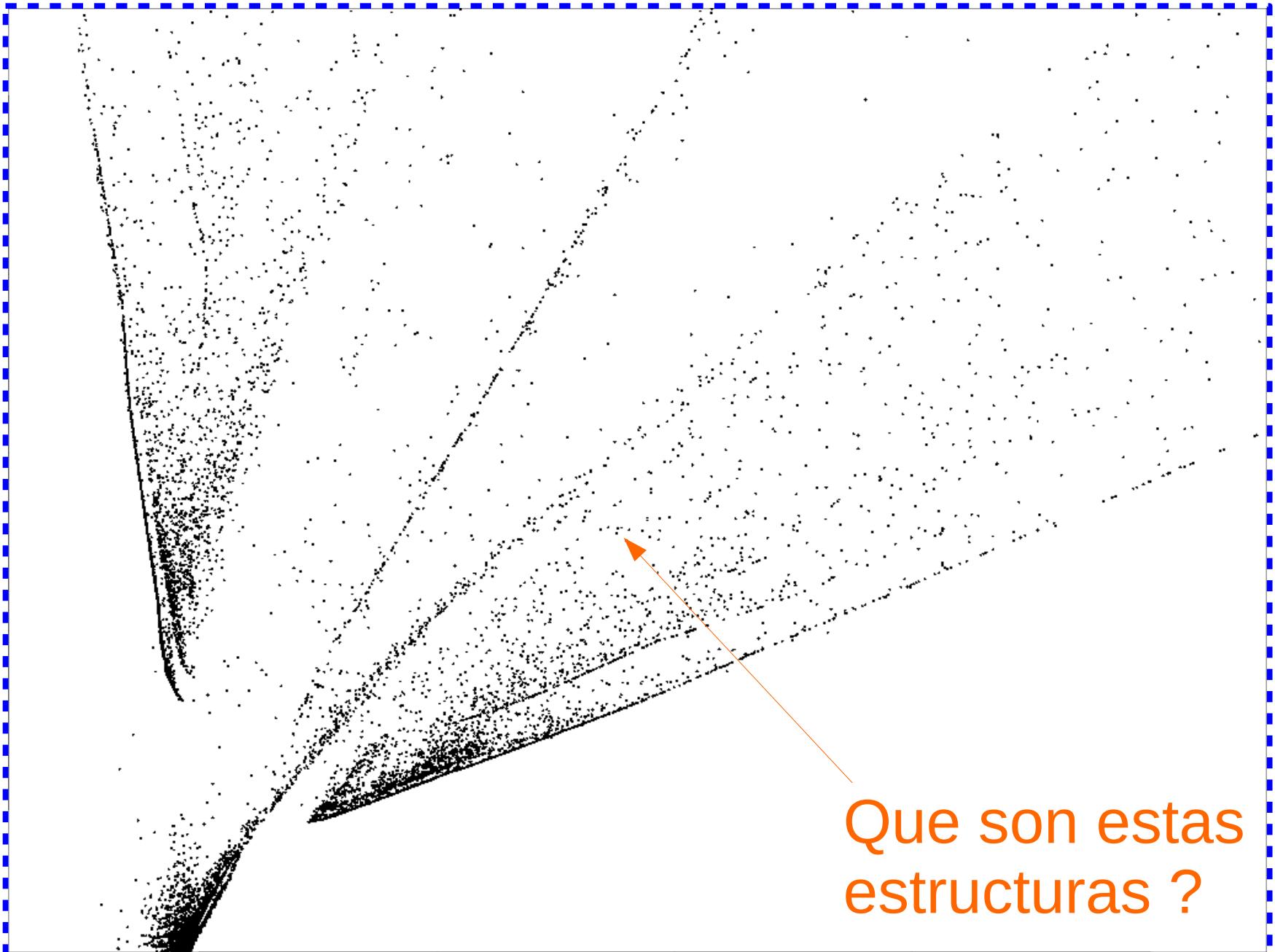
```
activacion<-"relu"
dim.orig<-ncol(x_train)
ent <- layer_input(shape = dim.orig)
inter <- layer_dense(ent, 64, activation = activacion)
inter2 <- layer_dense(inter, 16, activation = activacion)
laten <- layer_dense(inter2, 3, activation = activacion)
expan2 <- layer_dense(laten, 16, activation = activacion)
expan <- layer_dense(expan2,64, activation = activacion)
salida <- layer_dense(expan,dim.orig, activation = "linear")
# to generate latent variables
genlat <- keras_model(ent, laten)
# to generate the approximation
genrec <- keras_model(ent, salida)|
```

```
# Compilation
genrec %>% compile(
  loss = "mse",
  metrics= "mse",
  optimizer = "adam"
)
# training the model
genrec %>% fit(
  x = x_train,
  y = x_train,
  epochs = 100,
  batch_size = 500,
  validation_data = list(x_test, x_test),
)
```

$$X \approx f_k(X)$$

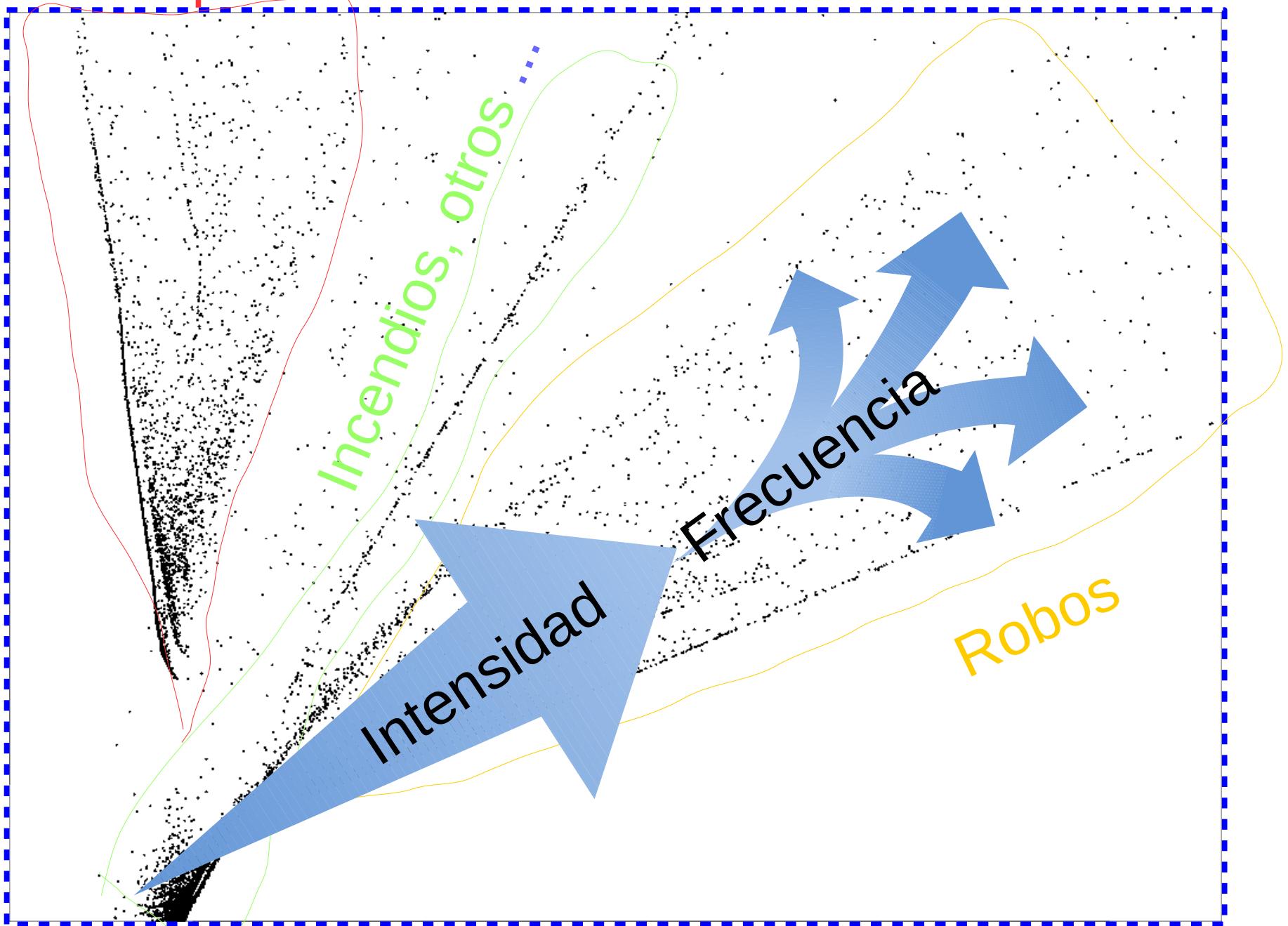


# ¿ Como es el Espacio Latente de los Siniestros ?



# ¿ Como es el Espacio Latente de los Siniestros ?

Choques



# ¿ Cuales son los Datos más Sospechosos ?

Severo caso de Inseguridad !

Muchos choques !

	A	B	C	D	E	F	G	H
1		Caso 1	Caso 2	Caso 3	Caso 4	Caso 5	Caso 6	Caso 7
2	Robo (Frecuencia)	4	90	1	20	14	0	10
3	Colisión Parcial (Frecuencia)	27	36	57	31	17	66	12
4	Colisión Total (Frecuencia)	2	0	4	5	0	5	4
5	Incendio (Frecuencia)	0	0	1	0	0	0	1
6	Otros (Frecuencia)	19	1	303	35	22	35	113
7	Robo (Intensidad)	689,624	789,914	24,938	875,853	485,012	0	232,474
8	Colisión Parcial (Intensidad)	1,363,938	286,844	136,349	128,054	110,431	242,843	42,883
9	Colisión Total (Intensidad)	613,457	0	111,529	237,248	0	165,278	85,614
10	Incendio (Intensidad)	0	0	25,780	0	0	0	2,822
11	Otros (Intensidad)	967,914	67	47,541	57,021	227,216	97,463	8,328

???