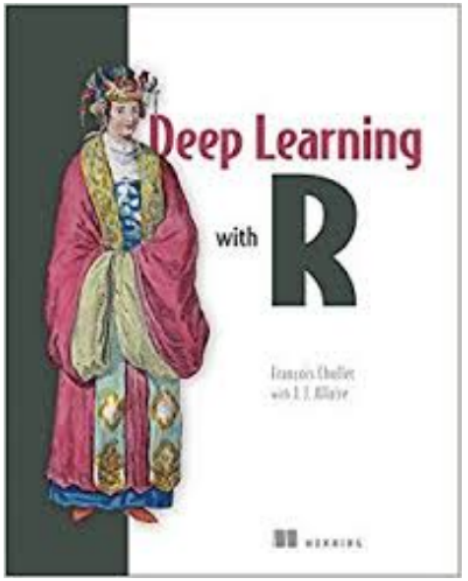
Inteligencia Artificial con R
Introducción al Deep Learning

Dr. Andrés Farall

Referencias

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

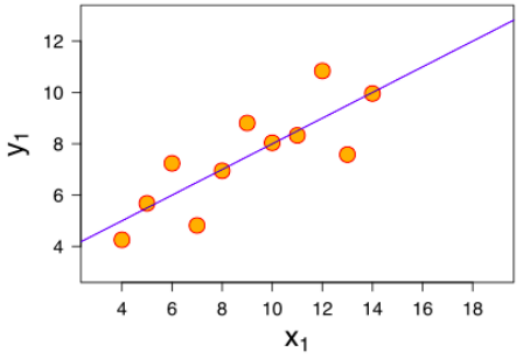


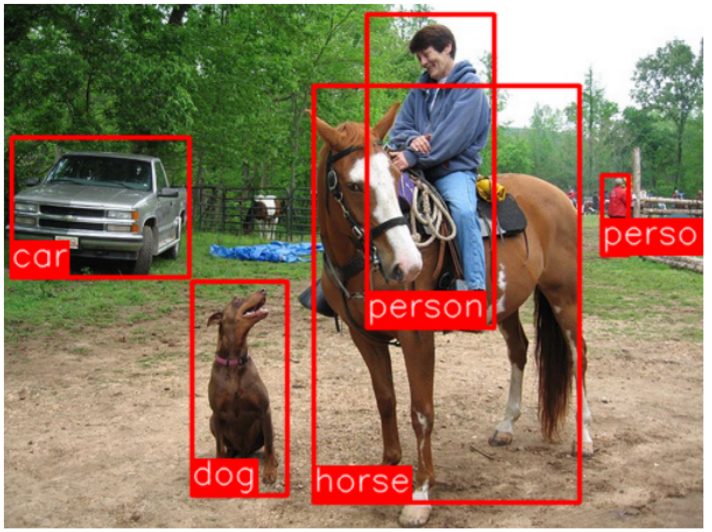
Inteligencia Artificial

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.



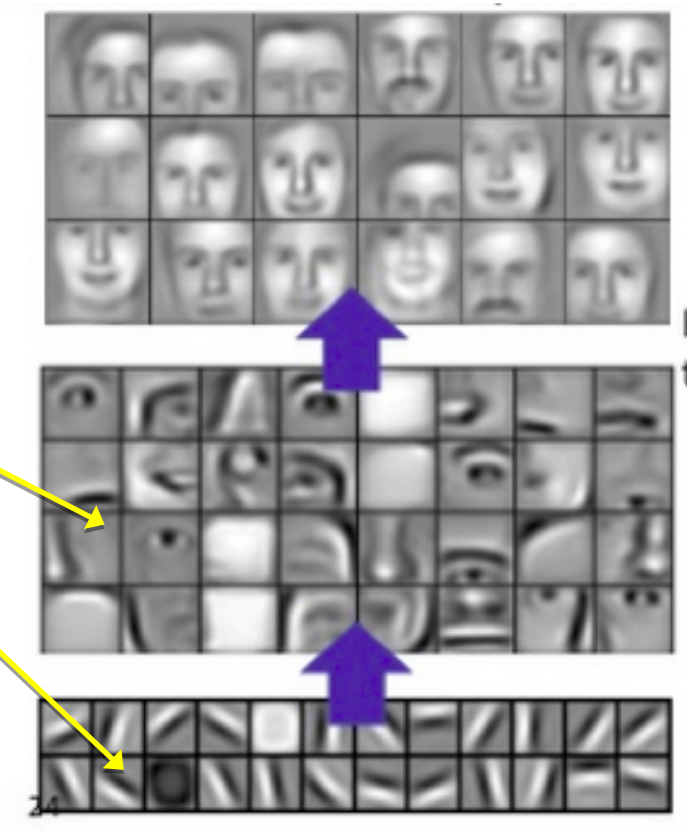


Deep Learning

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

Varios niveles de abstracción

Los niveles de abstracción se aprenden simultaneamente

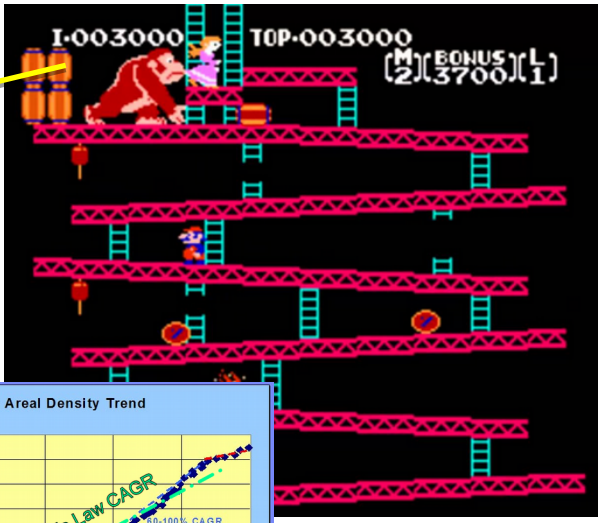


Aplicaciones de Deep Learning

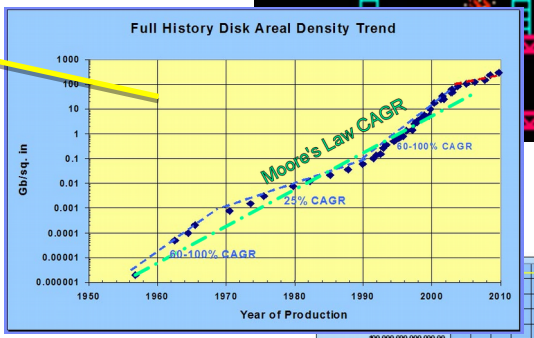
- Regresión (Single - Multiple Output)
- Clasificación (Single - Multiple Class)
- Reducción de Dimensión – Compresión
- Clasificación de Imágenes
- Reconociemiento del Habla
- OCR manuscrito
- Modelos Generativos (Síntesis de datos, Arte)
- Inferencia Estadística ?????

Las Claves del Éxito ?

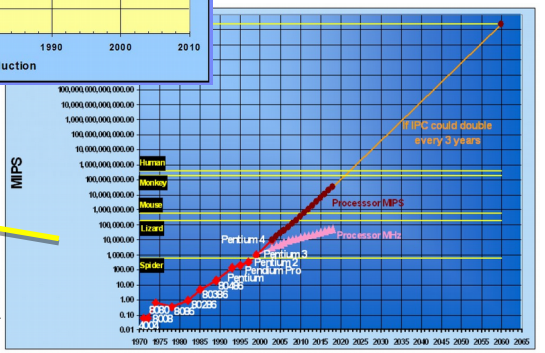
- Disponibilidad de GPU's



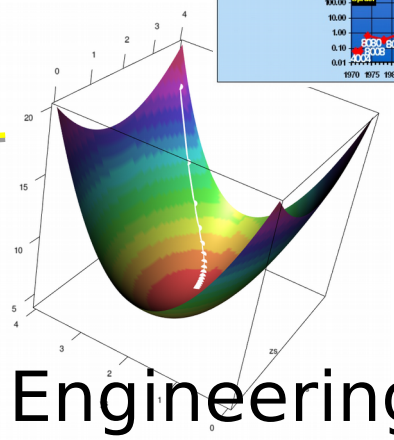
- Abundancia de Datos



- Aumento en la velocidad de procesaminto



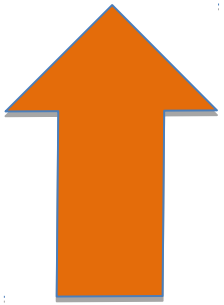
- Mejoras en los métodos de Optimización



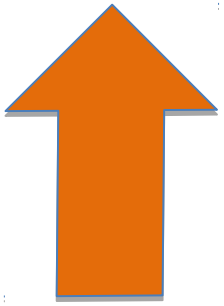
- Remueve/automatiza el Feature Engineering
- Re-utilización (transfer learning) y Versatilidad

Las Tecnologías a Usar

- R: Entorno de Data Science



- KERAS: Software OpenSource para ANN MLP



- TensorFlow: Software OpenSource para la progrmación 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, Convolucionales
  - 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

PPR

- 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 supervisada puede verse como una versión simplificada de una Red Neuronal.

Basado en Material de M.E. Szretter

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:

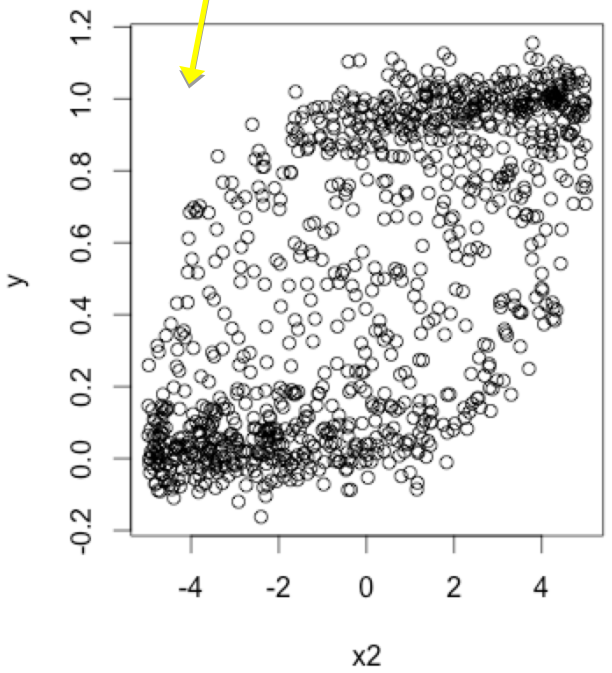
$$f(\mathbf{X}) = \sum_{m=1}^{M} g_m\left(\mathbf{w}_m^T \mathbf{X}\right)$$

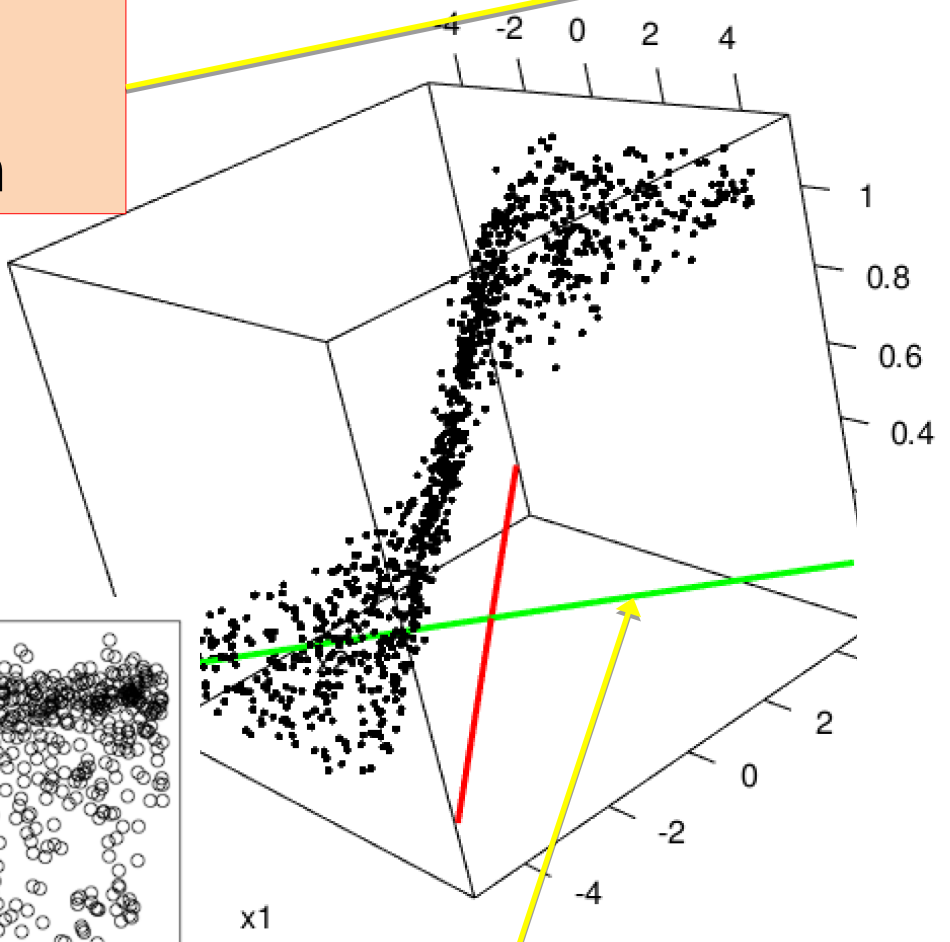
Función "ridge"

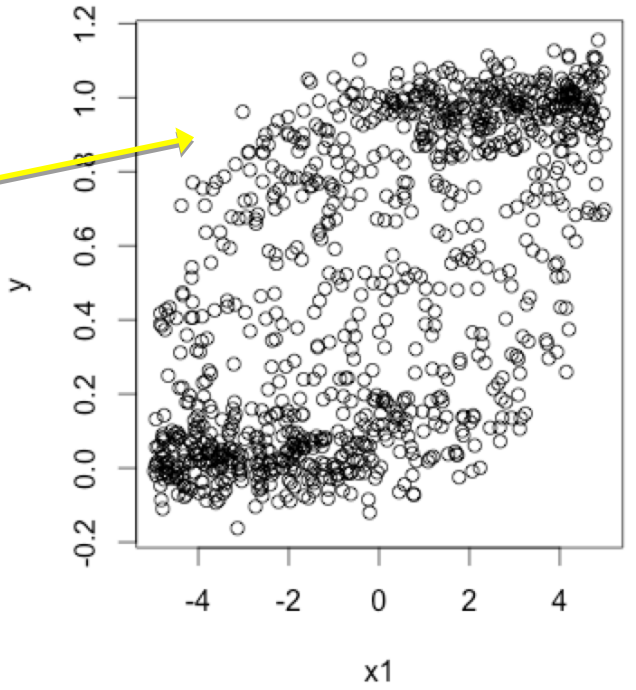
- donde $\mathbf{w}_m$ $(m = 1, \ldots, M)$ son vectores unitarios p-dimensionales de parámetros desconocidos
- y g_m son funciones no especificadas a estimar con los datos

La Intuición

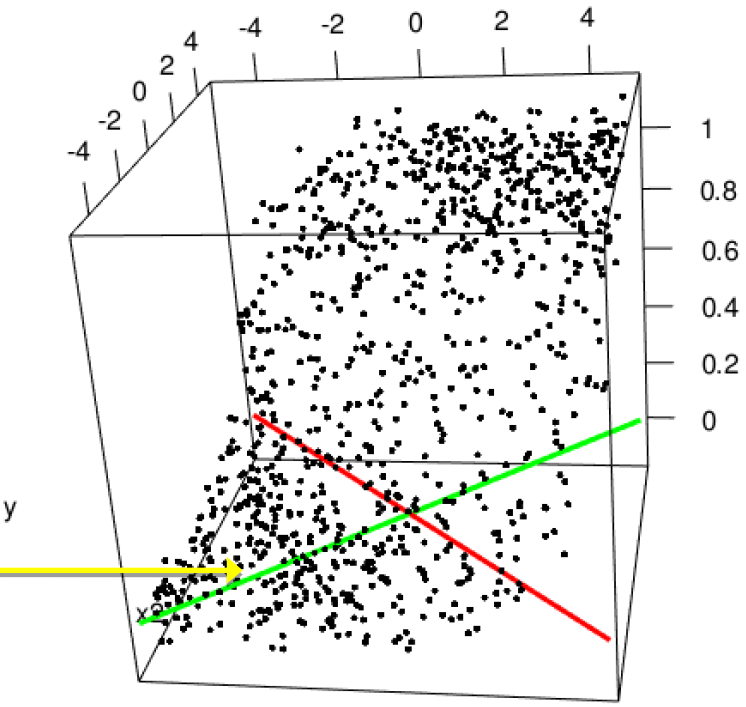
Escasa relación univariada





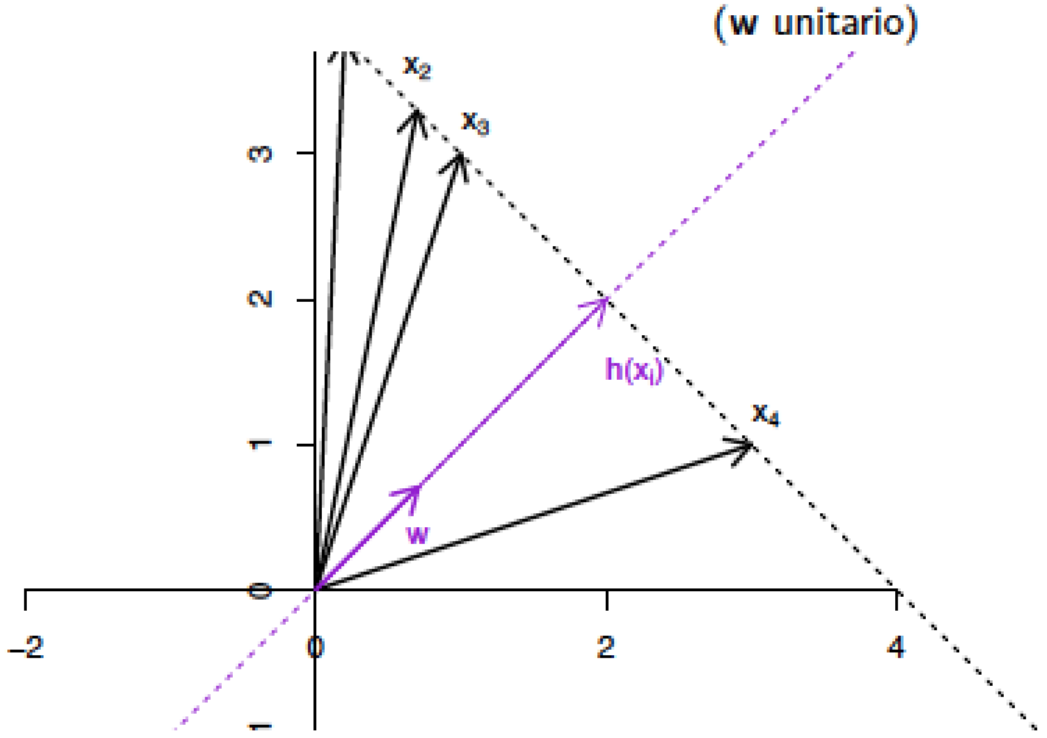


Dirección a ser hallada



Busqueda de la Dirección

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



Buscamos $\mathbf{w_m}$ para que el modelo ajuste bien: Projection Pursuit

La función $g_m\left(\mathbf{w}_m^T\mathbf{X}\right)$ 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 $(\mathbf{X}_i, Y_i)$, $(i = 1, \ldots, n)$, ¿cómo ajustamos? Buscamos minimizar la función

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

Estimación

Asumimos **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 **w**. Resulta equivalente a un problema de regresión de mínimos cuadrados pesados

Los pasos para actualizar a g y **w** se iteran hasta converger

Ajuste de PPR: caso $M > 1$

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

$$R_i = Y_i - \sum_{j=1}^{m-1} \widehat{g}_j \left(\widehat{\mathbf{w}}_j^T \mathbf{X} \right)$$

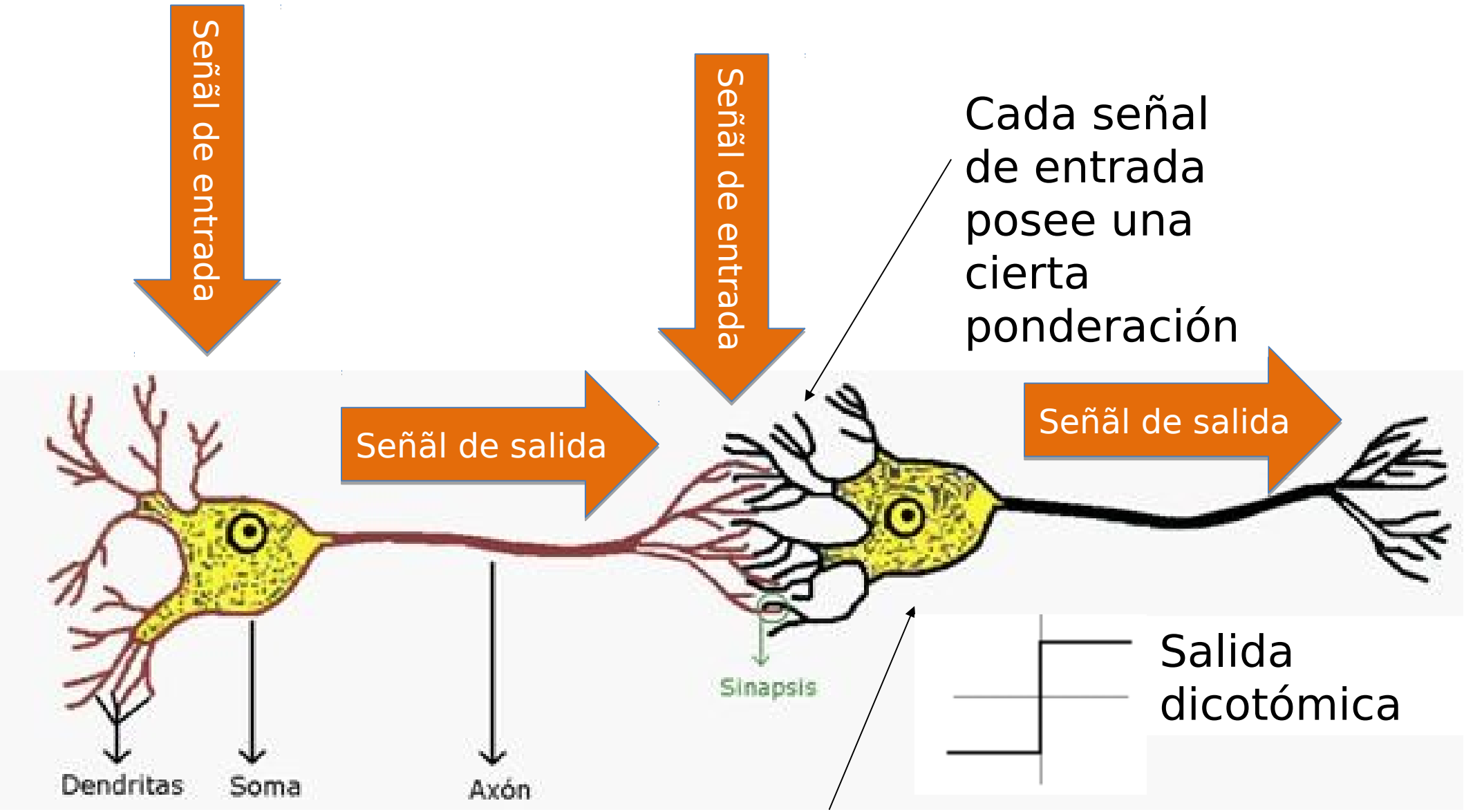
los residuos de esta aproximación. Le aplicamos el algoritmo anterior a los pares $(\mathbf{X}_i, R_i)$, para obtener $\widehat{g}_m$ y $\widehat{\mathbf{w}}_m$. El procedimiento se itera hasta que la mejora que se obtiene de la función objetivo es muy pequeña.
Muchos detalles de implementación:

- ¿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

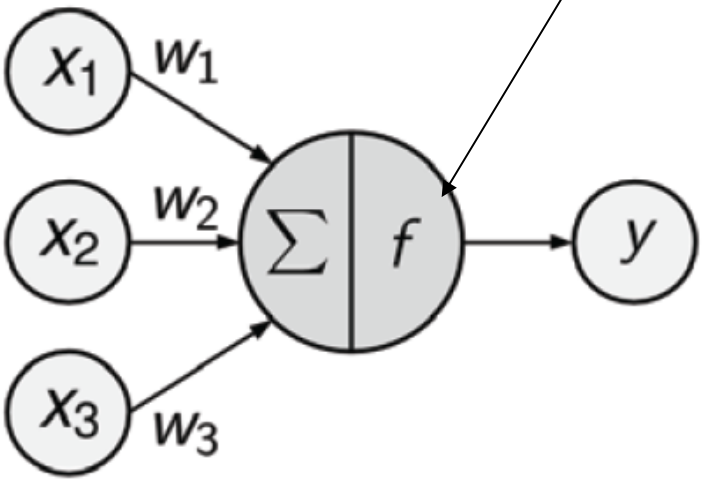


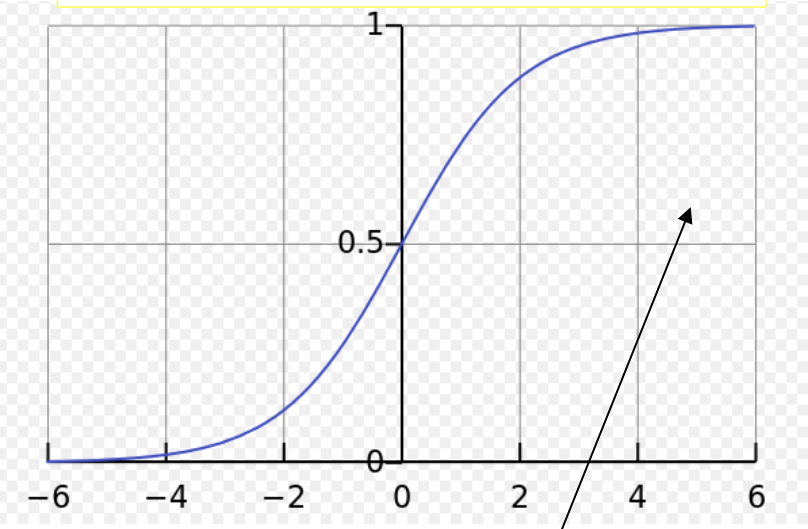
La neurona “suma” las señales ponderadas y “gatilla” una señal de salida

Elementos Básicos de una Red

Función de activación

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





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

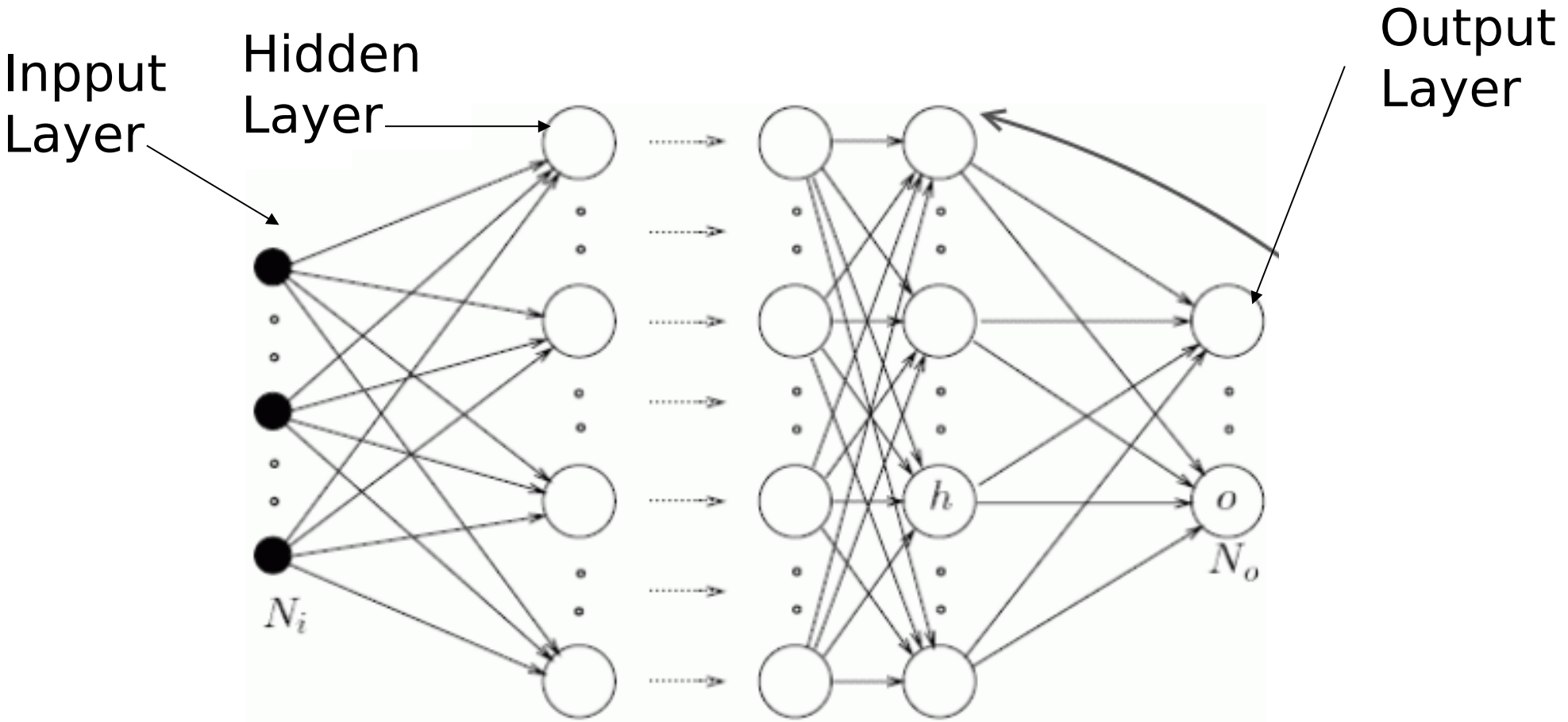
Función Sigmoidea o Logística

Tangente Hiperbólica

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

Topologías de Redes Neuronales

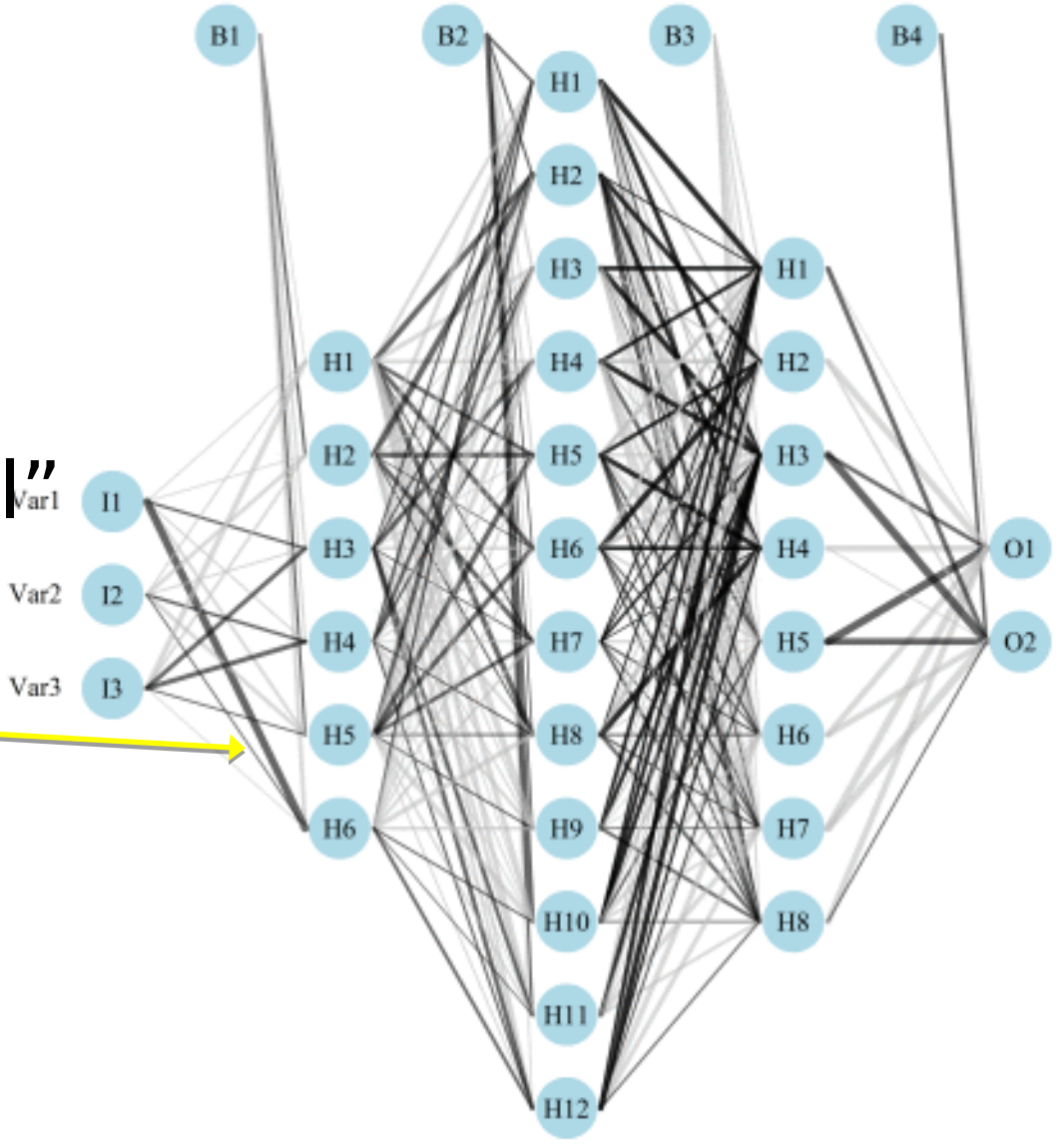
- La complejidad de una red depende de:
  - La cantidad de capas de nueronas
  - La cantidad de neuronas por capa
  - La dirección de las señales



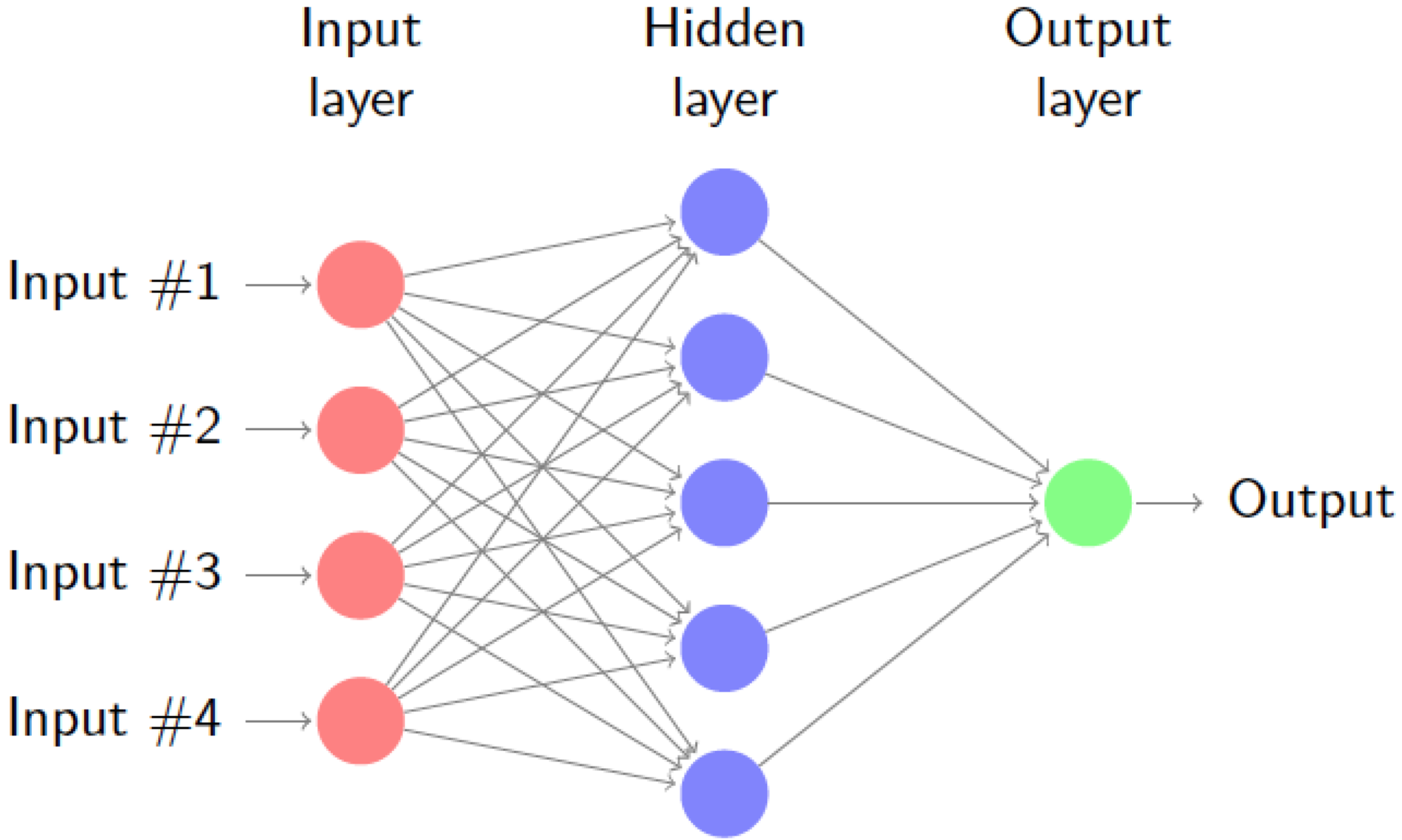
El Multilayer Perceptron

- 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 calibraods

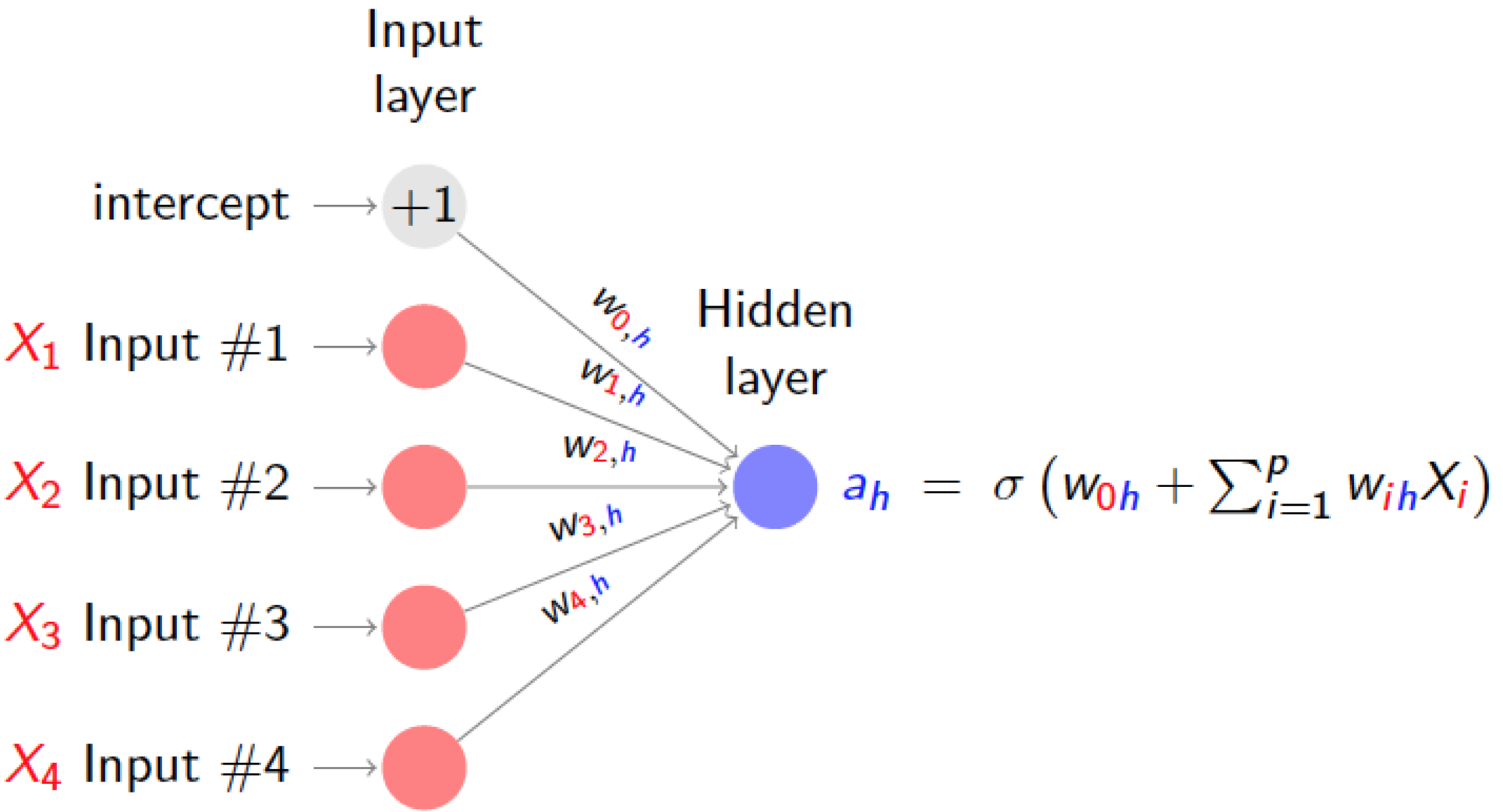


Redes Neuronales: Ejemplo



4 variables predictoras o inputs 5 unidades ocultas

Ejemplo de Red Neuronal



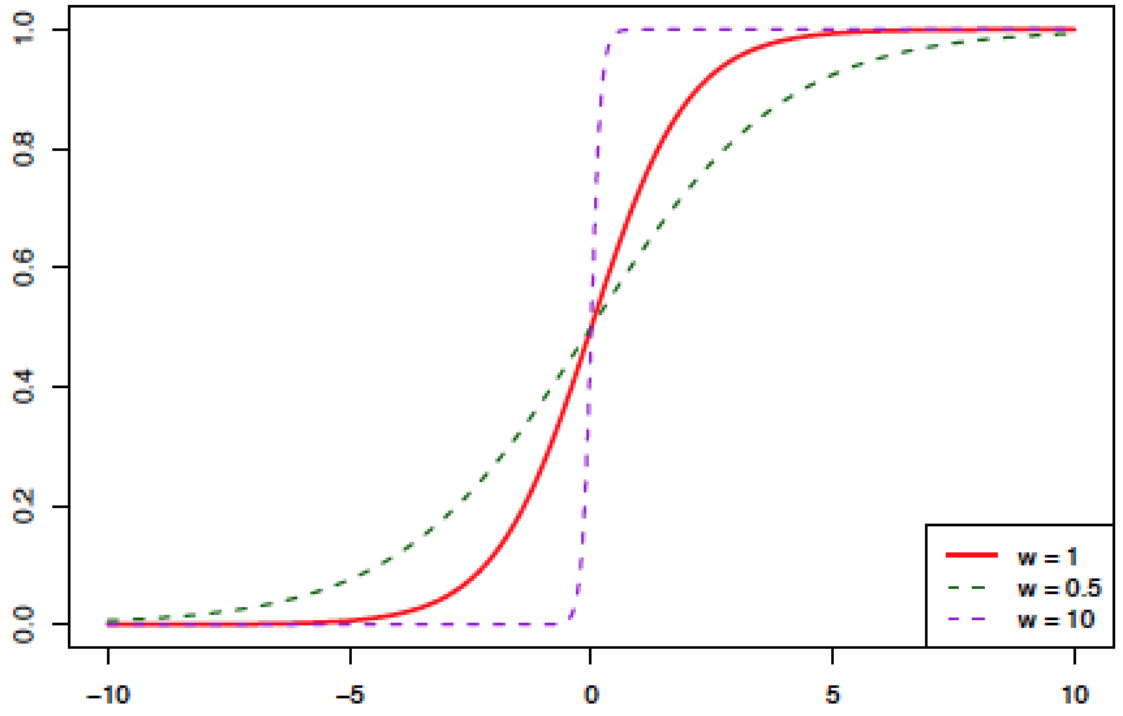
Red Neuronal

- 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
- σ es una función no lineal, usualmente la sigmoidea

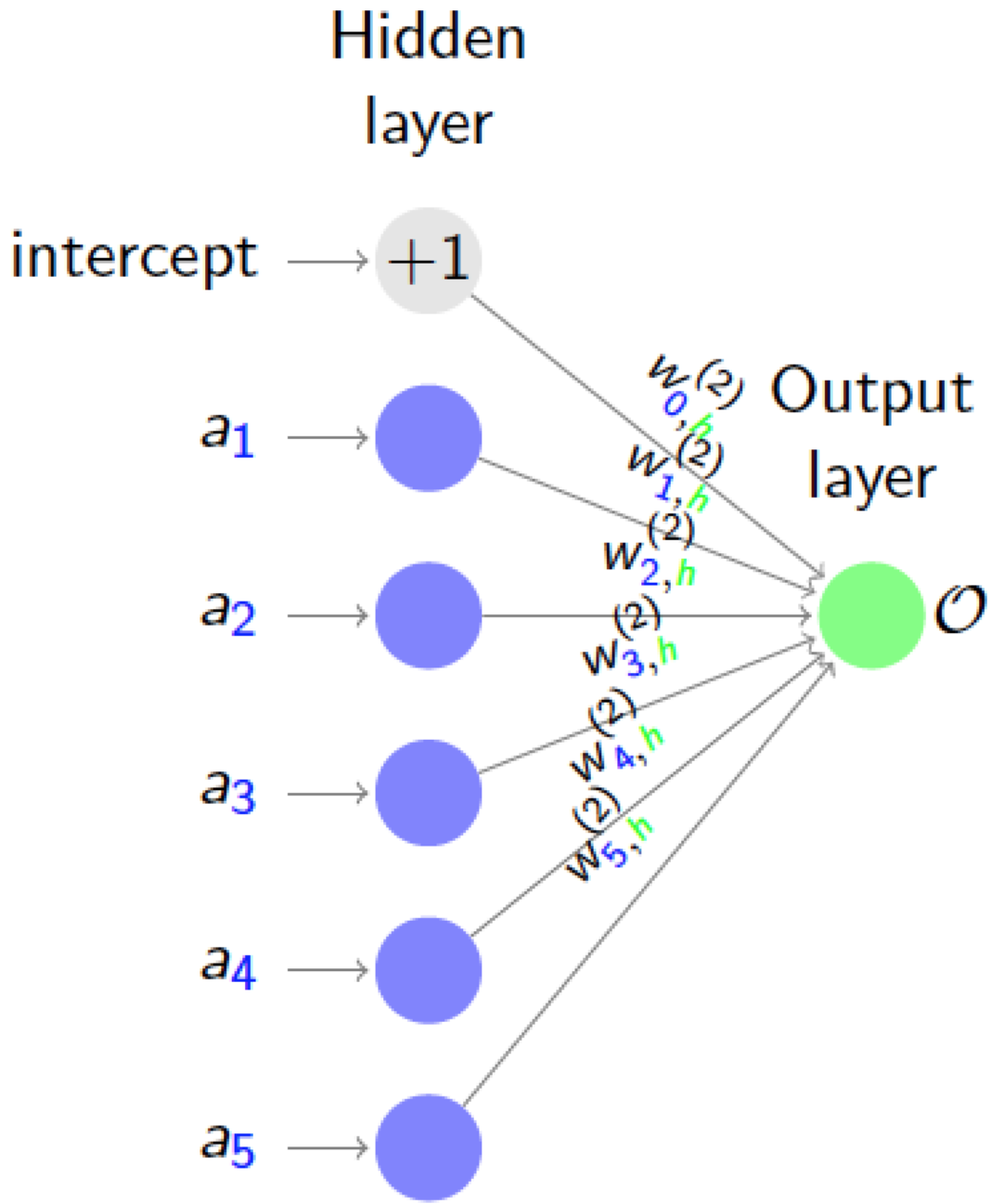
$$\sigma(t) = \frac{1}{1 + e^{-t}}$$

Graficamos

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



Red Neuronal: output



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

Red Neuronal: output

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

¿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:

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

(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: $(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)$
  - una función objetivo: $L[Y, f(\mathbf{X}, \mathcal{W})]$
  - un algoritmo de optimización

  podríamos estimarla.
- Debemos penalizar la función objetivo en los pesos, para evitar el sobreajuste

Redes Neuronales: Ajuste

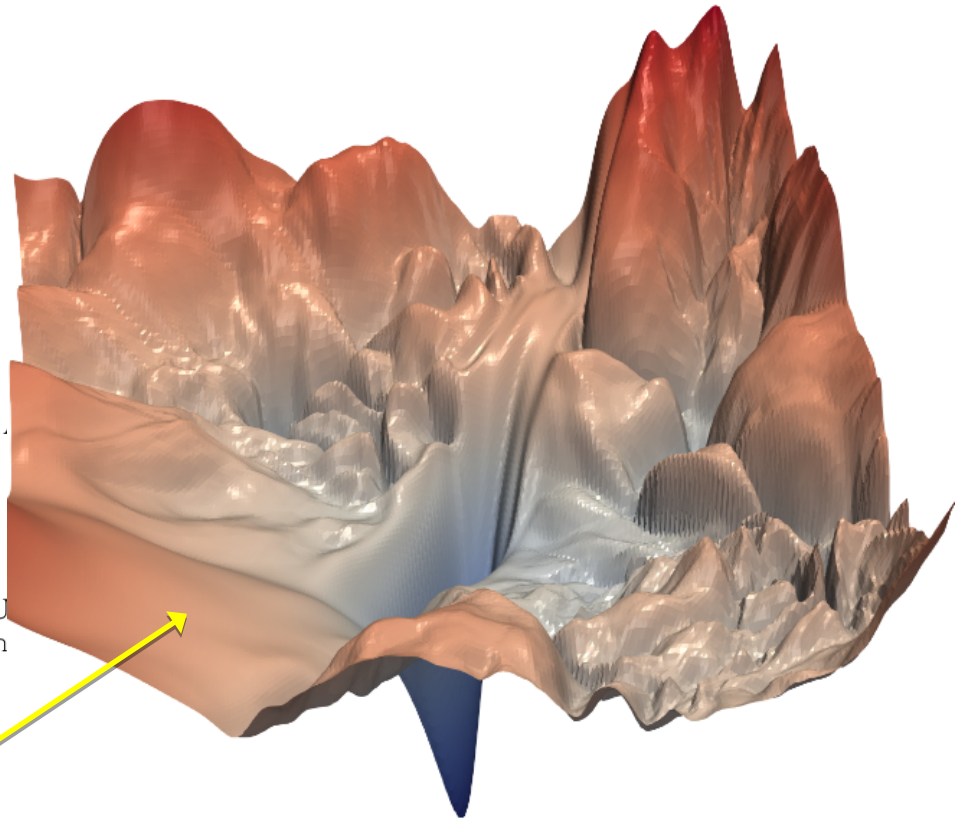
Queremos hallar los pesos $\mathcal{W}$ que resuelvan

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

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,
  $J(\mathcal{W}) = \frac{1}{2} \sum_{i=1}^{n} \sum_{l=1}^{L} \sum_{h=1}^{H_l} \left[w_{ih}^{(l)}\right]^2$ (*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[1], **Zheng Xu**[1], **Gavin Taylor**[2], **Christoph Studer**[3], **Tom Goldstein**[1]
[1]University of Maryland, College Park, [2]United States Naval Academy, [3]Cornell U
{haoli,xuzh,tomg}@cs.umd.edu, taylor@usna.edu, studer@corn

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

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

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

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

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, $(X_1, \ldots, X_{784})$. 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 `h20` 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)

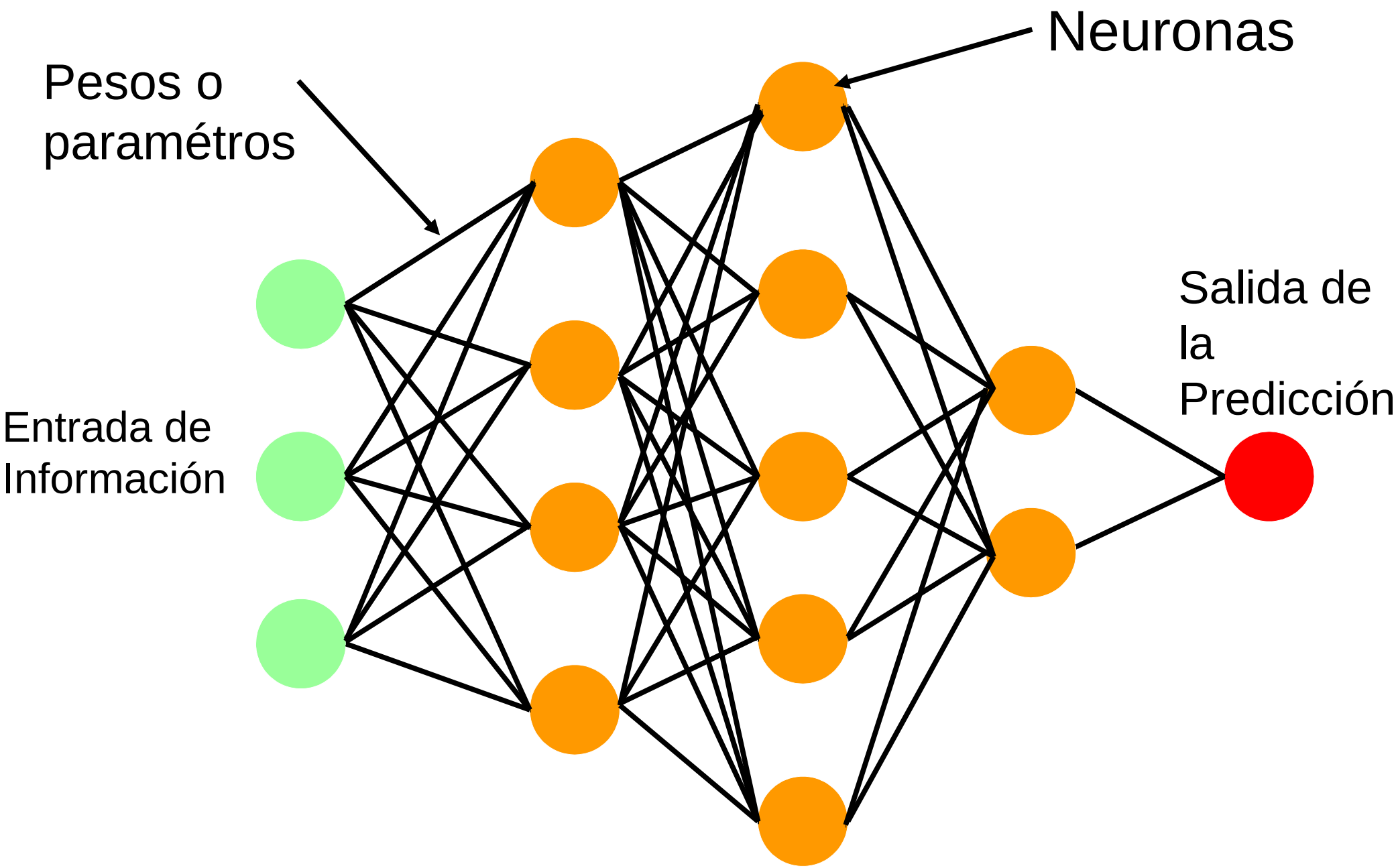
Ejemplo: Dígitos postales (ZIP code), evolución

Para los datos `MNIST`, la mejor red neuronal daba

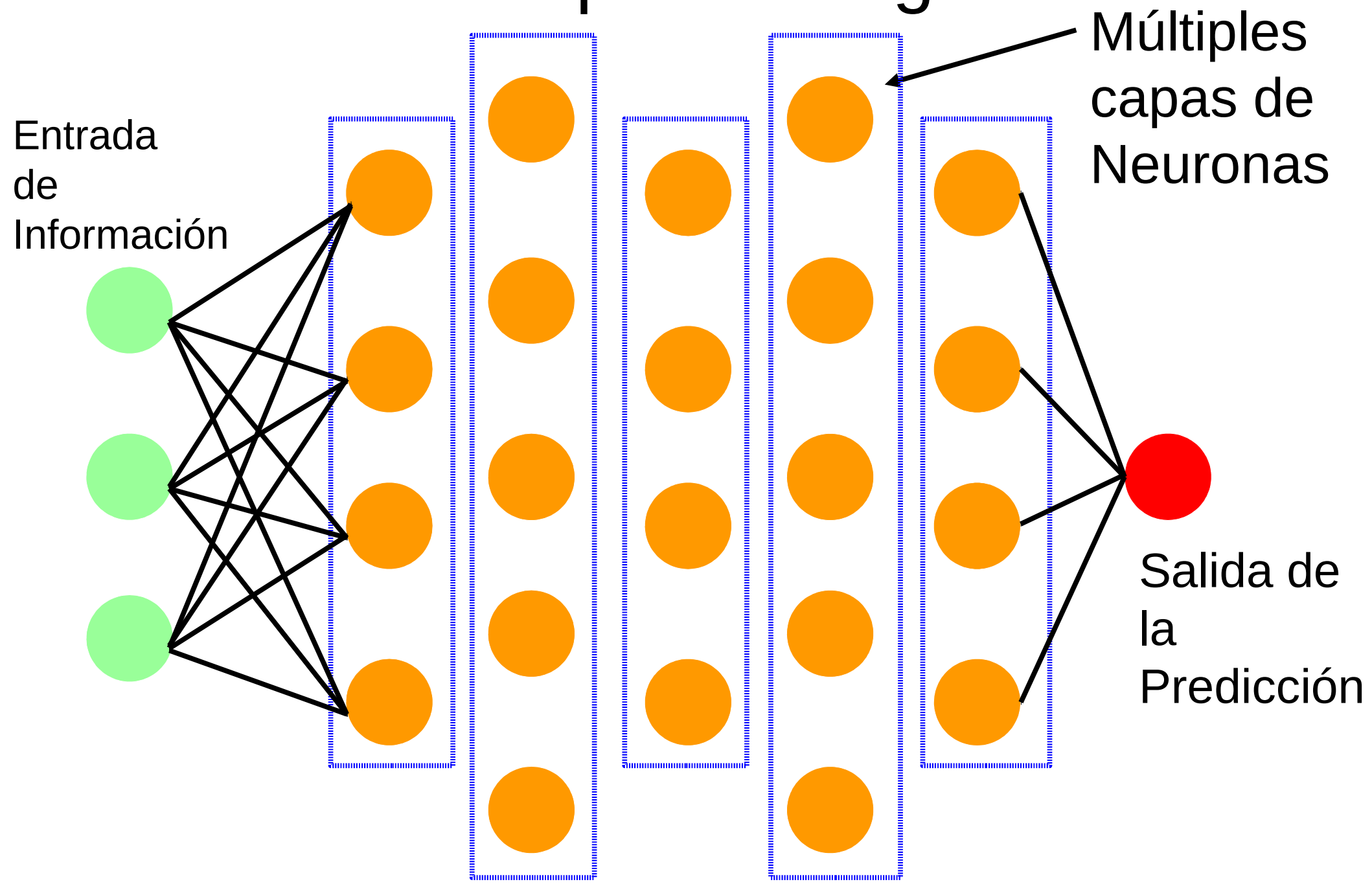
- En el 2008 un error del 1,6 %
- En el 2016 un error del 0,93 %

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.

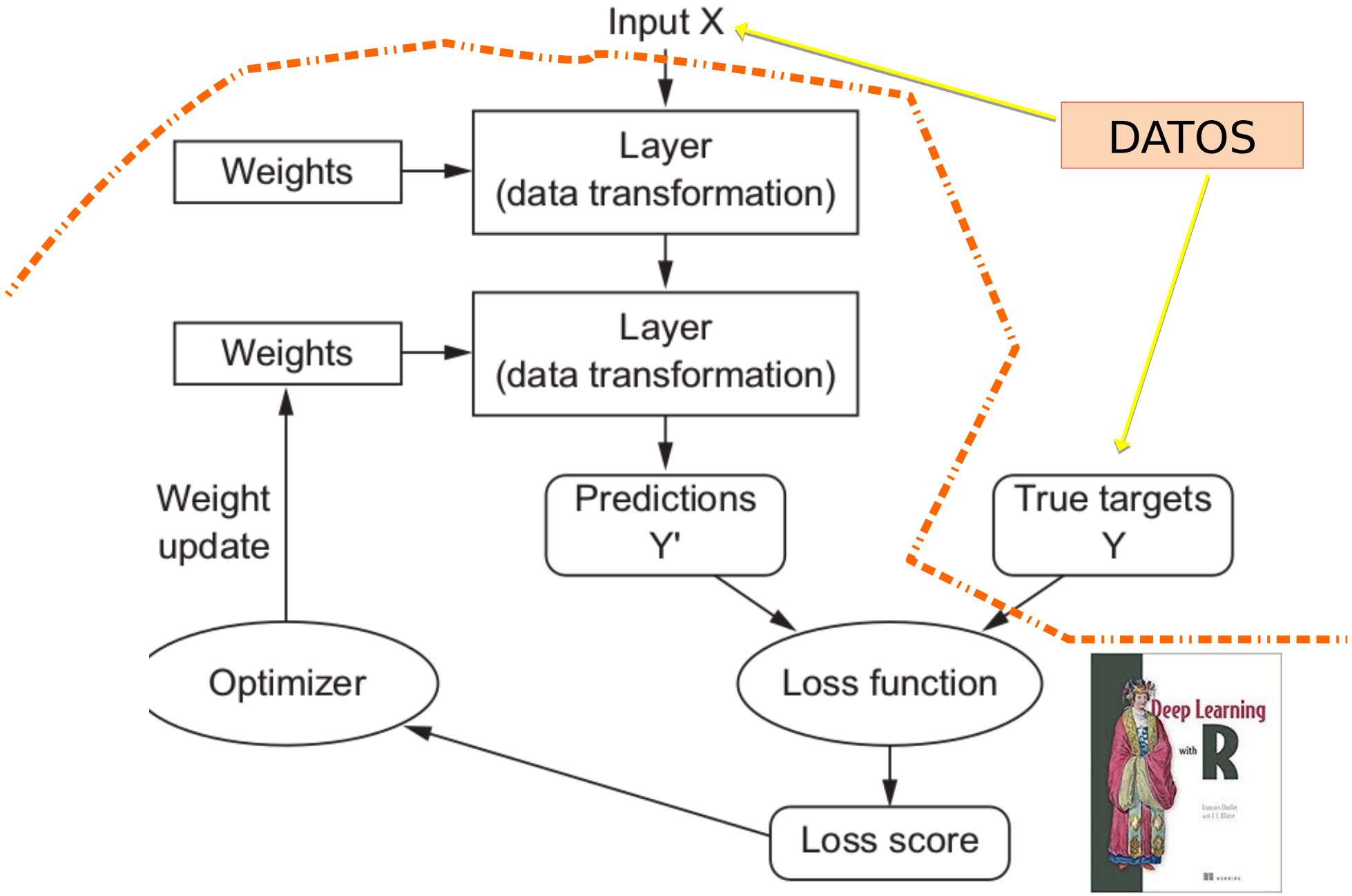
Redes Neuronales Artificiales



Deep Learning



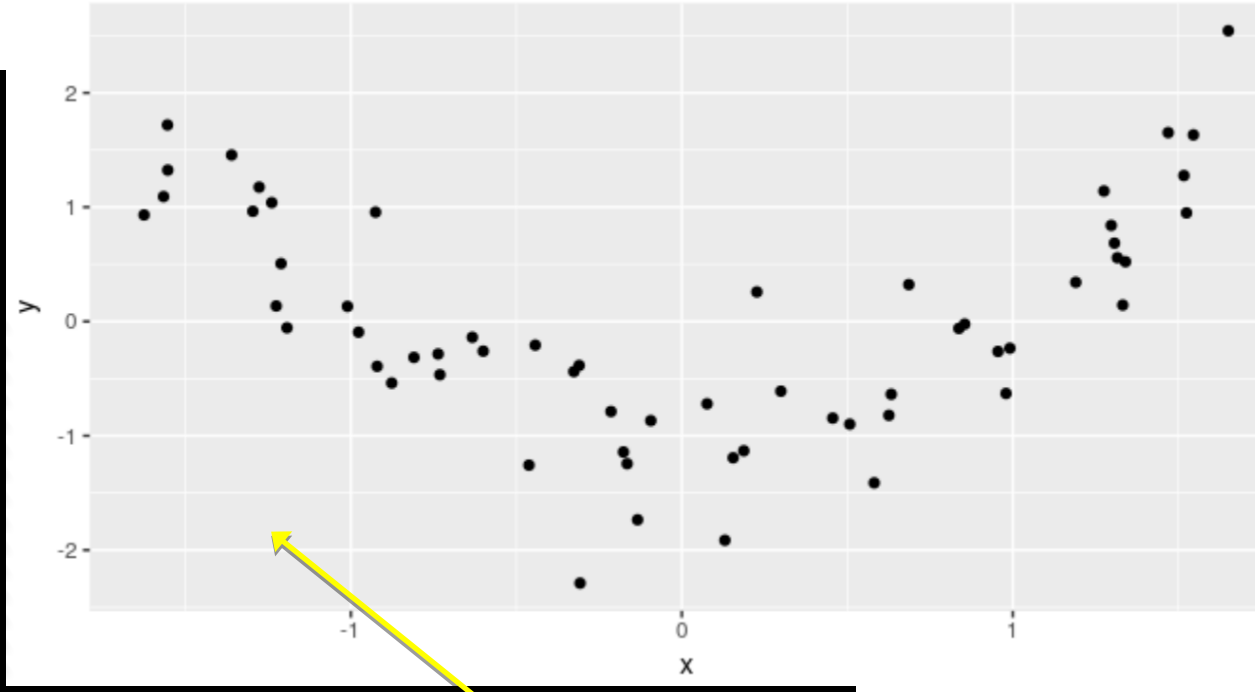
Flujo de Entrenamiento de un Modelo de Deep Learning (ANN)



Ejemplo muy Simple

Regresión Lineal

```r
# 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

Training

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

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")
```

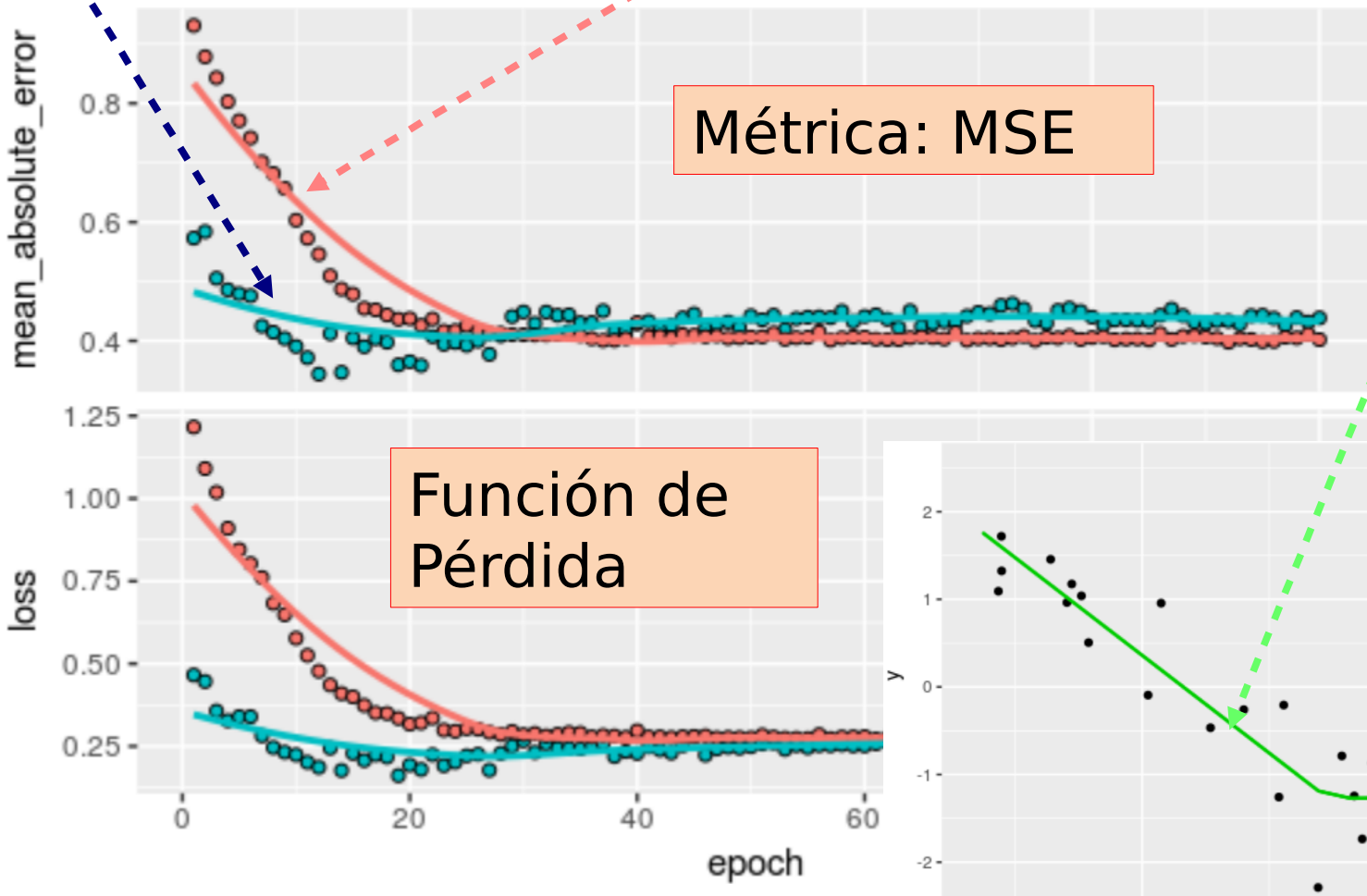
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 256)               512
_________________________________________________________________
dense_4 (Dense)              (None, 1)    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

Resultado del Ajuste

Validación

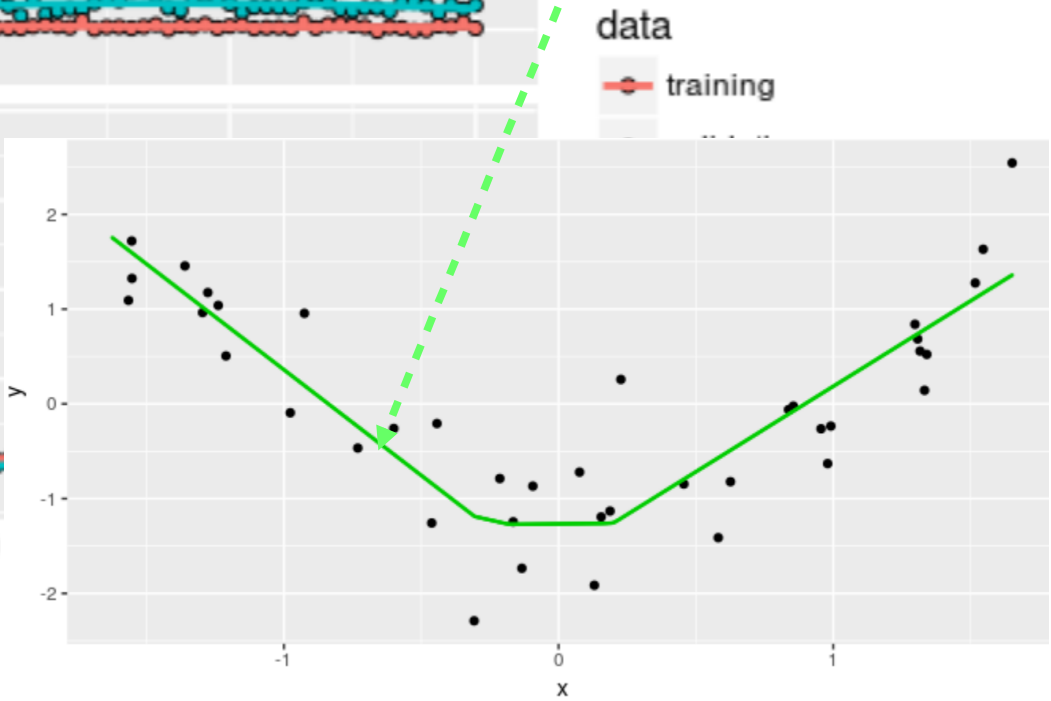
Entrenamiento

Función Predicha



Métrica: MSE

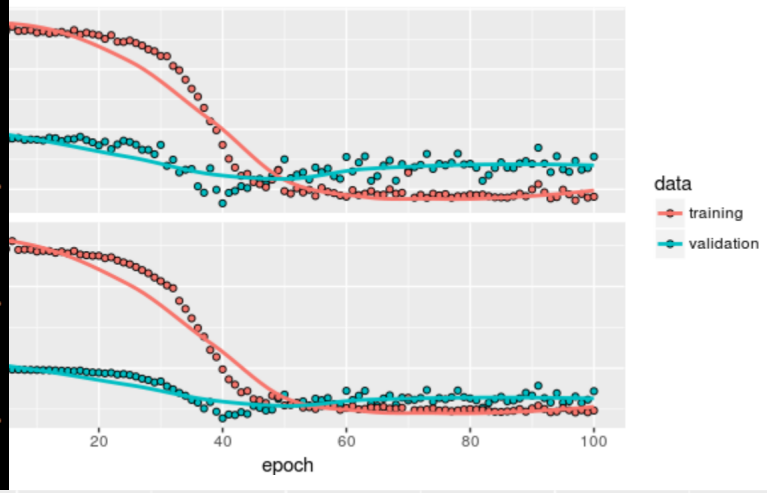
Función de Pérdida

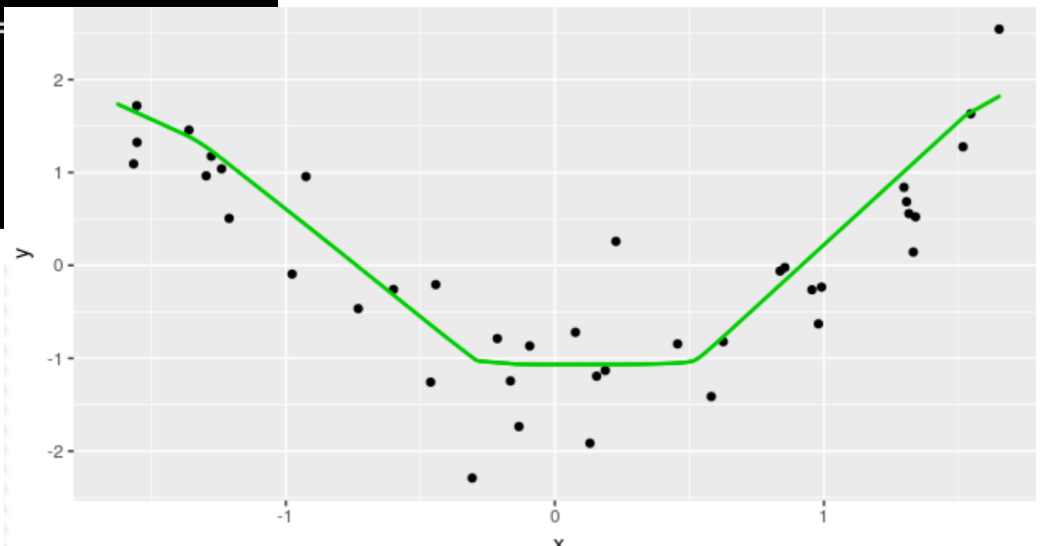


Agregando Capas Conectadas

```
modelo <- keras_model_sequential() %>%
   layer_dense(units = 256, activation = "relu" ,kernel_regular
   layer_dense(units = 128, activation = "relu",kernel_regulari
   layer_dense(units = 64, activation = "relu",,kernel_regulari
   layer_dense(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
```

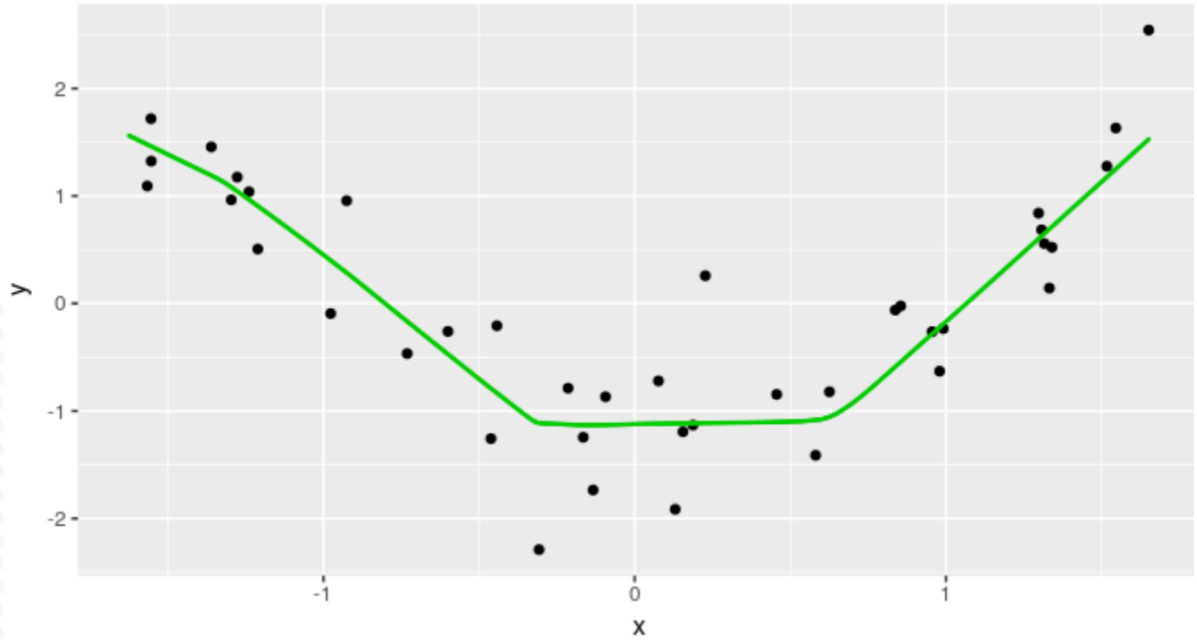




Exagerando la Complejidad del Modelo

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

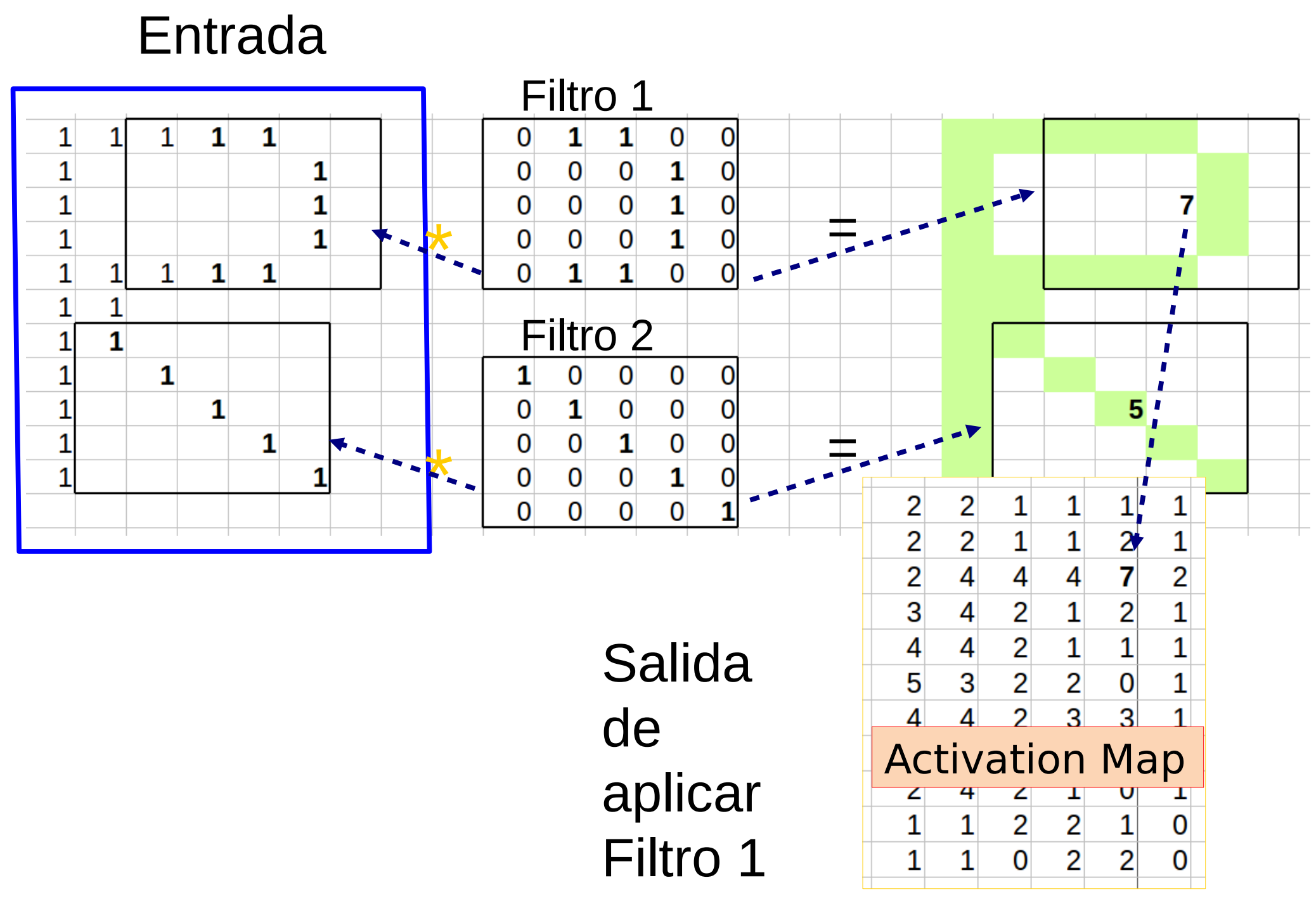


Modelos Convolucionales

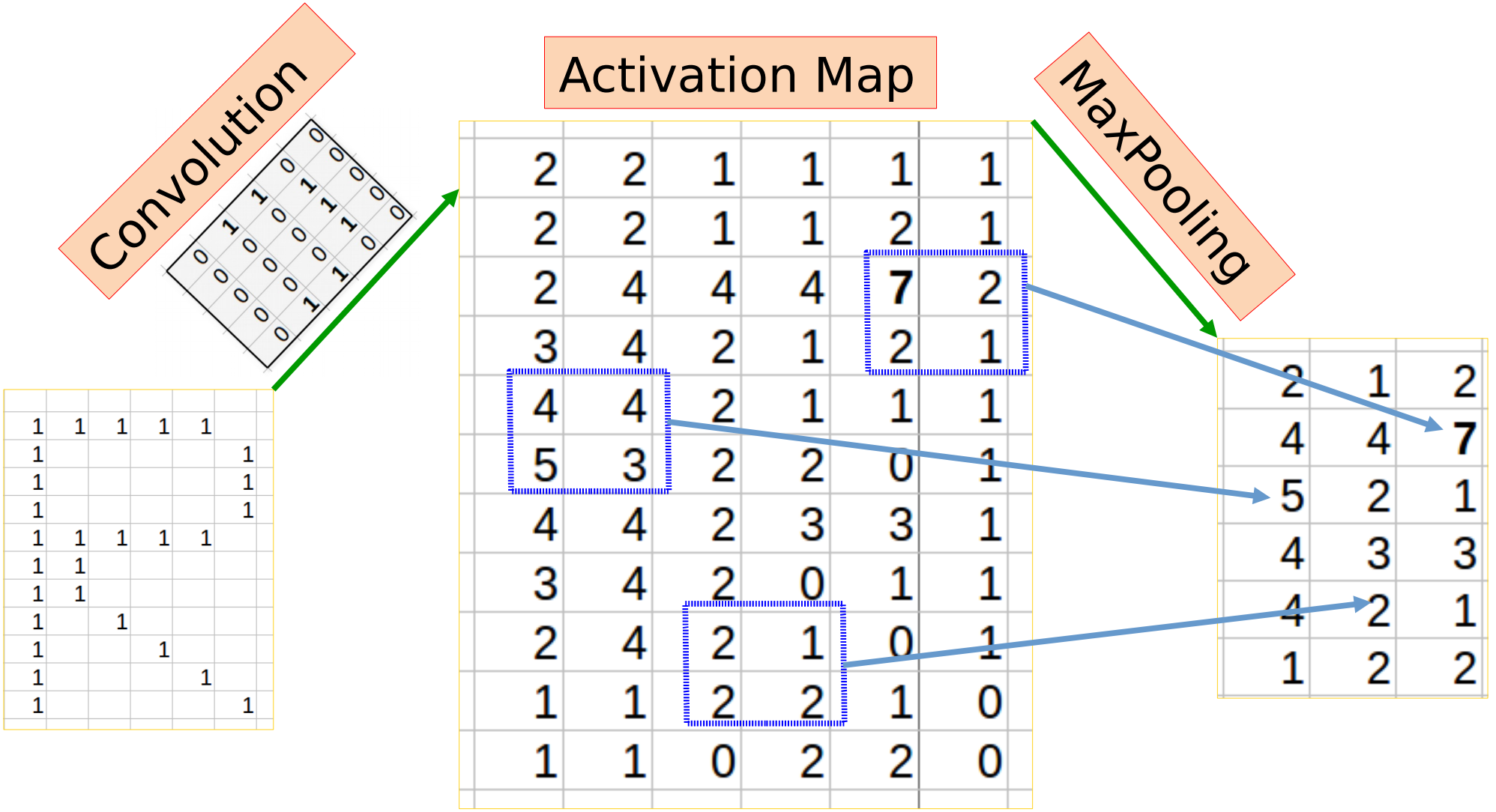
Se utilizan para modelar el comportamiento de fenómenos provenientes de procesos que ocurren en un dominio continuo (por ej.: tiempo y espacio).

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

La Operación de Convolución



La Operación de “Pooling”



CONV + POOL + … + CONV + POOL = ABSTRACCION

Ejemplo de Clasificación en Series de Tiempo

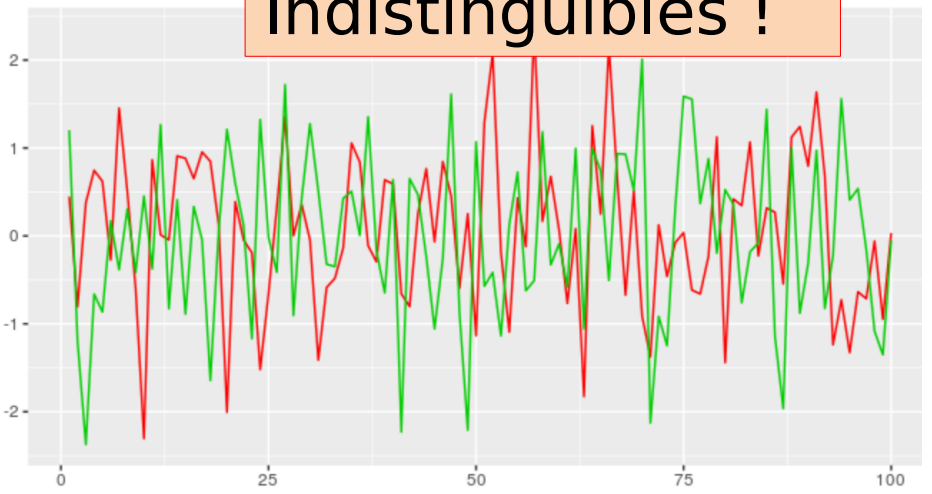
- Dos clases de series:
  - Serie 1 (200 casos, 500 obs)
  - Serie 2 (200 casos, 500 obs)

Autocor. = -1/10
y Media = + 1/30

Autocor. = +1/10
y Media = - 1/30

Ejemplo exagerado

Ejemplo real
Indistinguibles !



Definición del Modelo

Arquitectura del Modelo

```
ve                of time window
modelo <- keras_model_sequential()
modelo %>%
  layer_conv_1d(filters = 2, kernel_size = vent,
                activation = 'relu',
                input_shape = c(largo,1)) %>%
  layer_global_average_pooling_1d() %>%
  layer_dense(units = 1, activation = 'sigmoid')
```

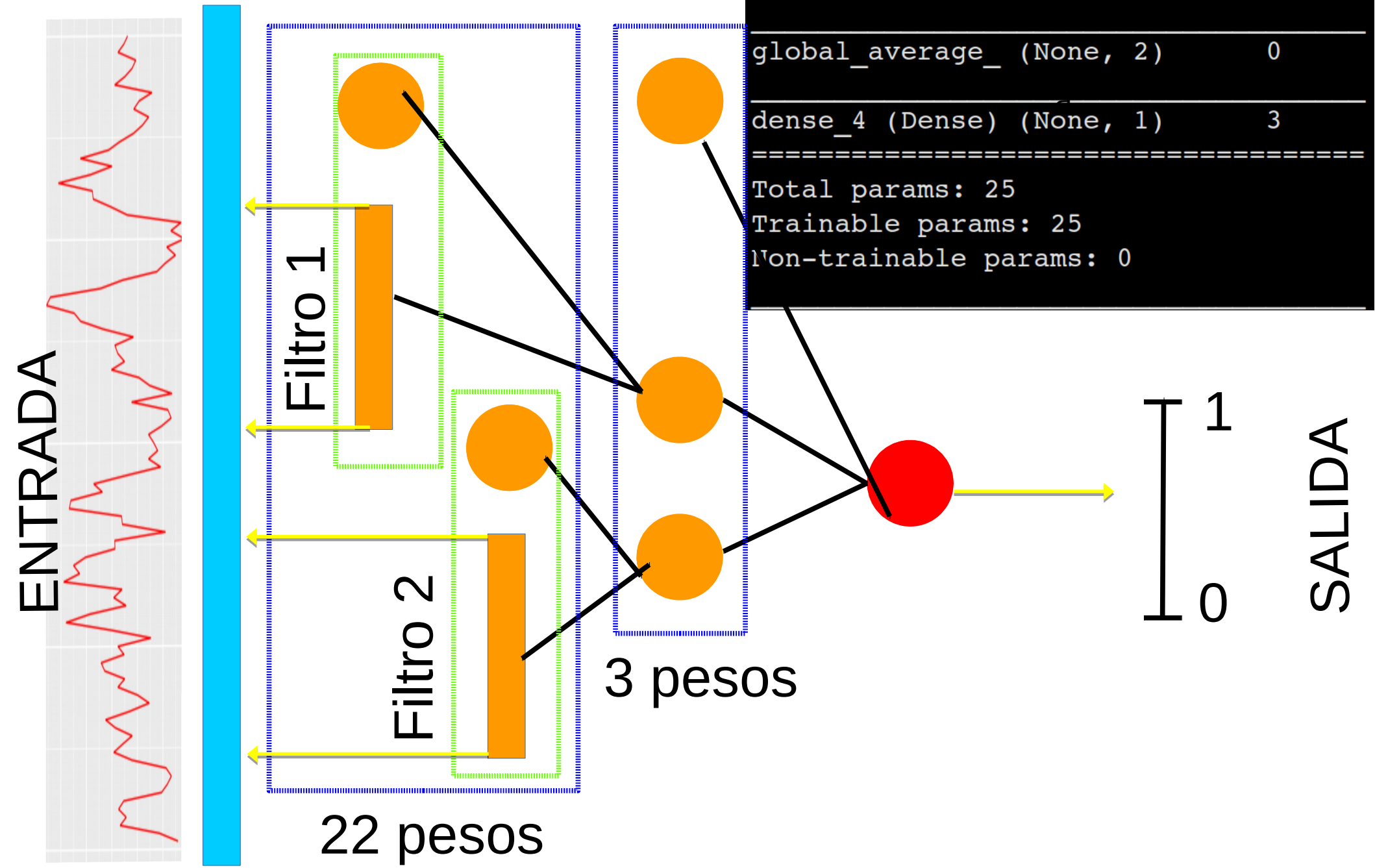
```
modelo %>%  compile(
    loss = 'binary_crossentropy',
    optimizer = 'rmsprop',
    metrics = c('accuracy'))
```

Parámetros de Compilación

```
historia <- modelo %>%
  fit(array.train, label.train, batch_size = 16,
                          epochs = 250,validation_split = 0.2)
```

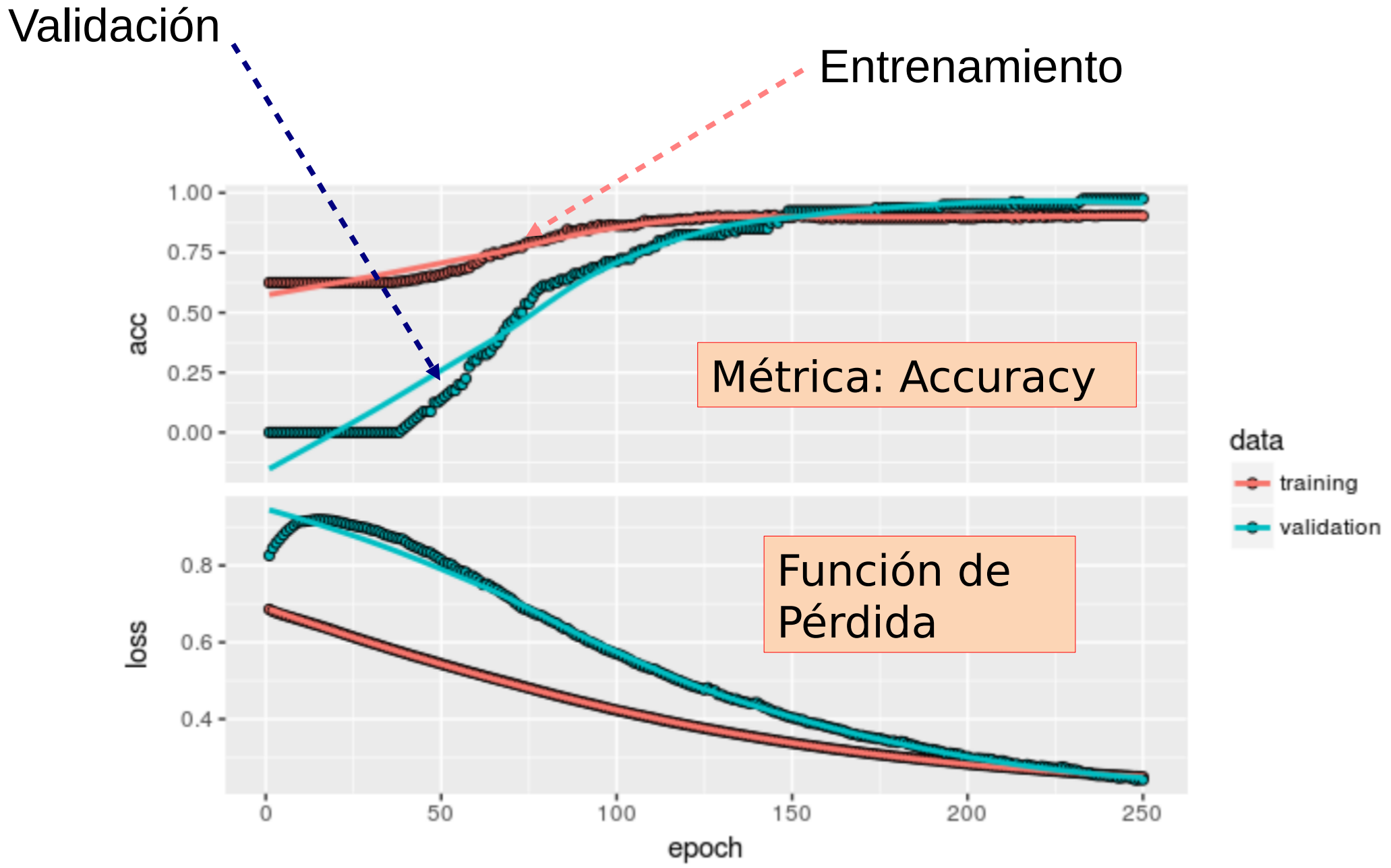
Training

El Modelo




```
_________________________________________________________________
Layer (type)                 Output Shape              Param
=================================================================
conv1d_4 (Conv1              (None, 491, 2)            22
_________________________________________________________________
global_average_              (None, 2)                 0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 3
=================================================================
Total params: 25
Trainable params: 25
Non-trainable params: 0
_________________________________________________________________
```

El Ajuste

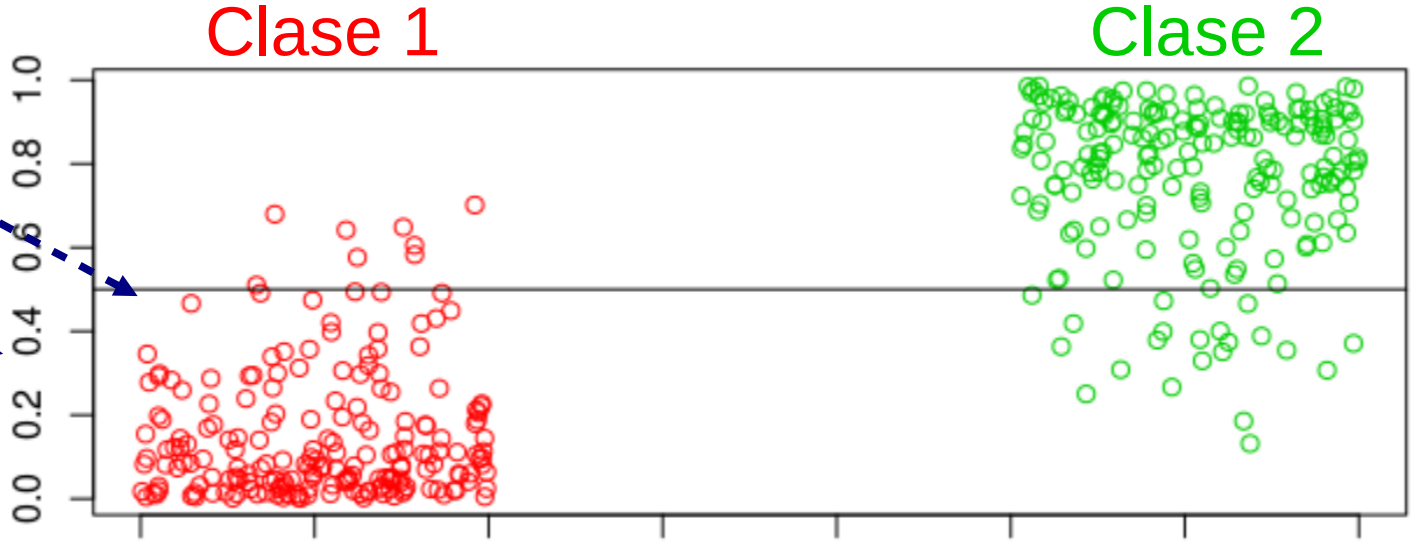

Validación
Entrenamiento
1.00
0.75
0.50
0.25
0.00
acc
Métrica: Accuracy
data
training
validation
0.8
0.6
0.4
loss
Función de Pérdida
0
50
100
150
200
250
epoch

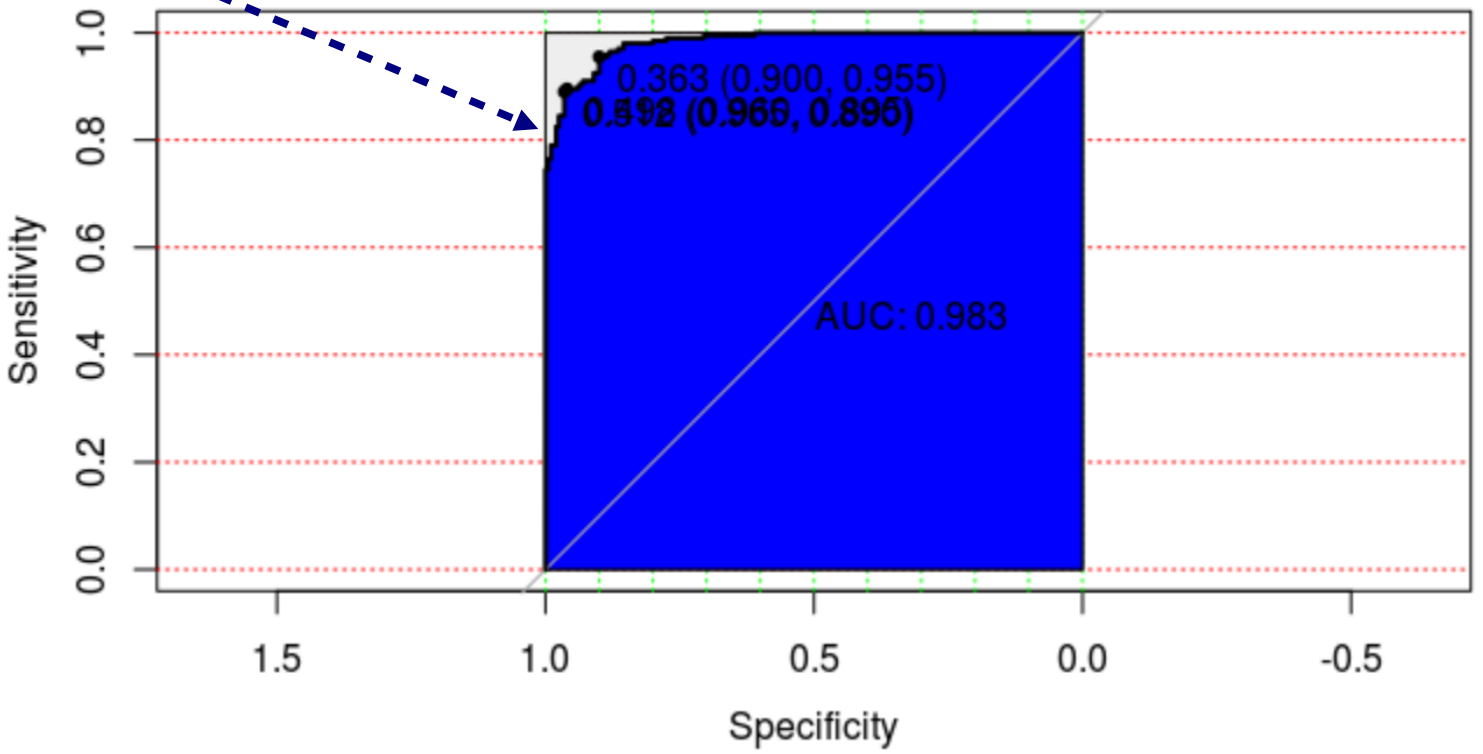
Desempeño de la Clasificación

Umbral de Clasificación

Score de Clasificación

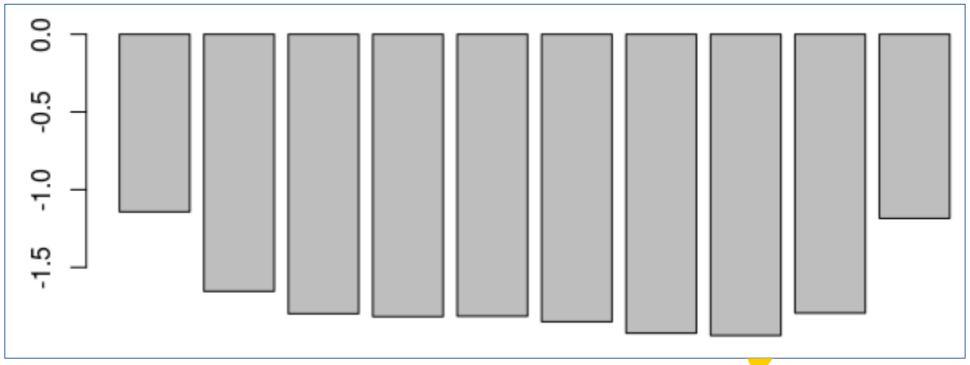
Curva ROC



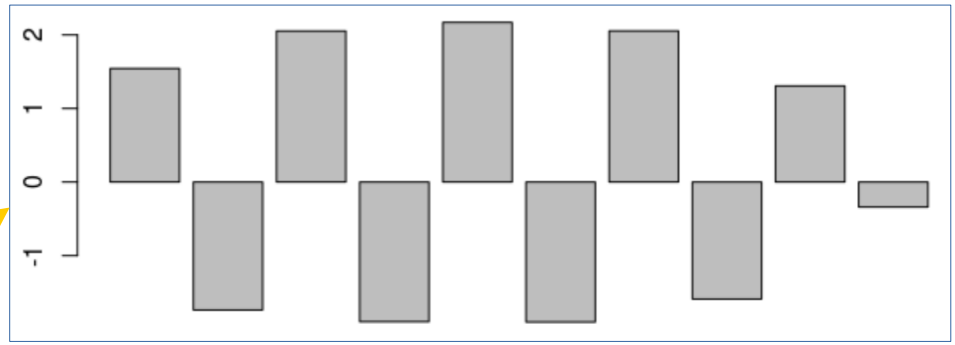


Que Aprendió la Red ?

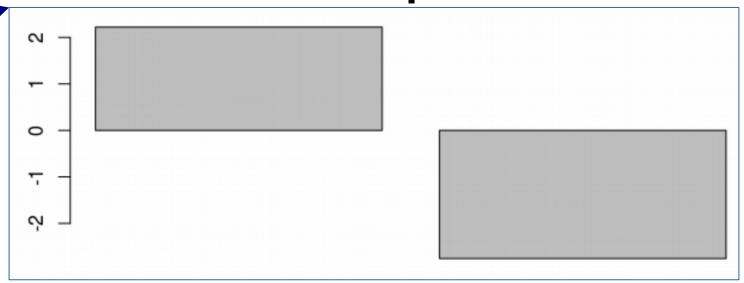
Filtro 1: Patrón de Posición

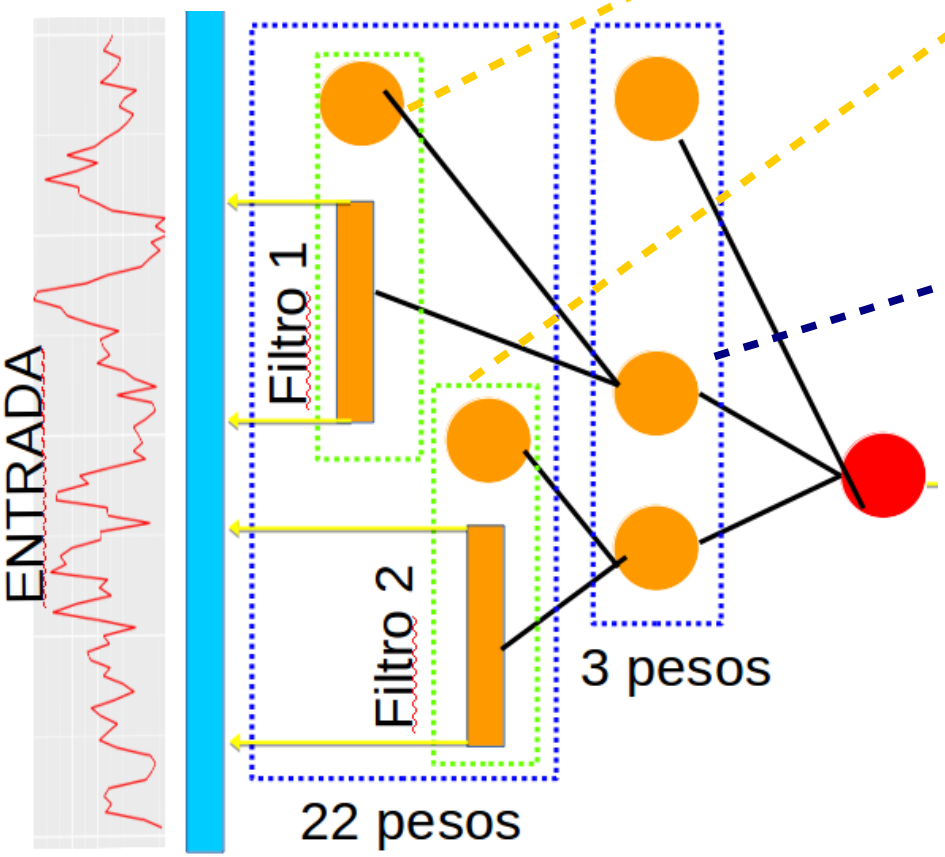


Filtro 2: Patrón de Autocorrelación



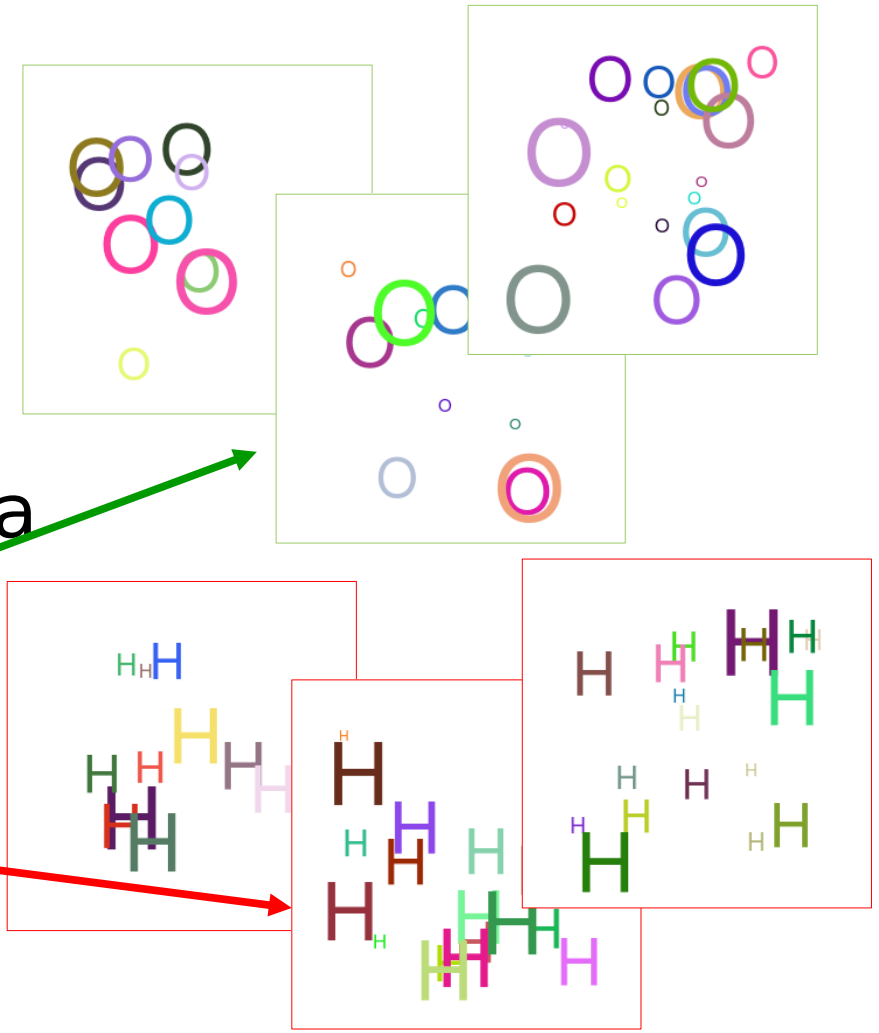
Ultima Capa





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

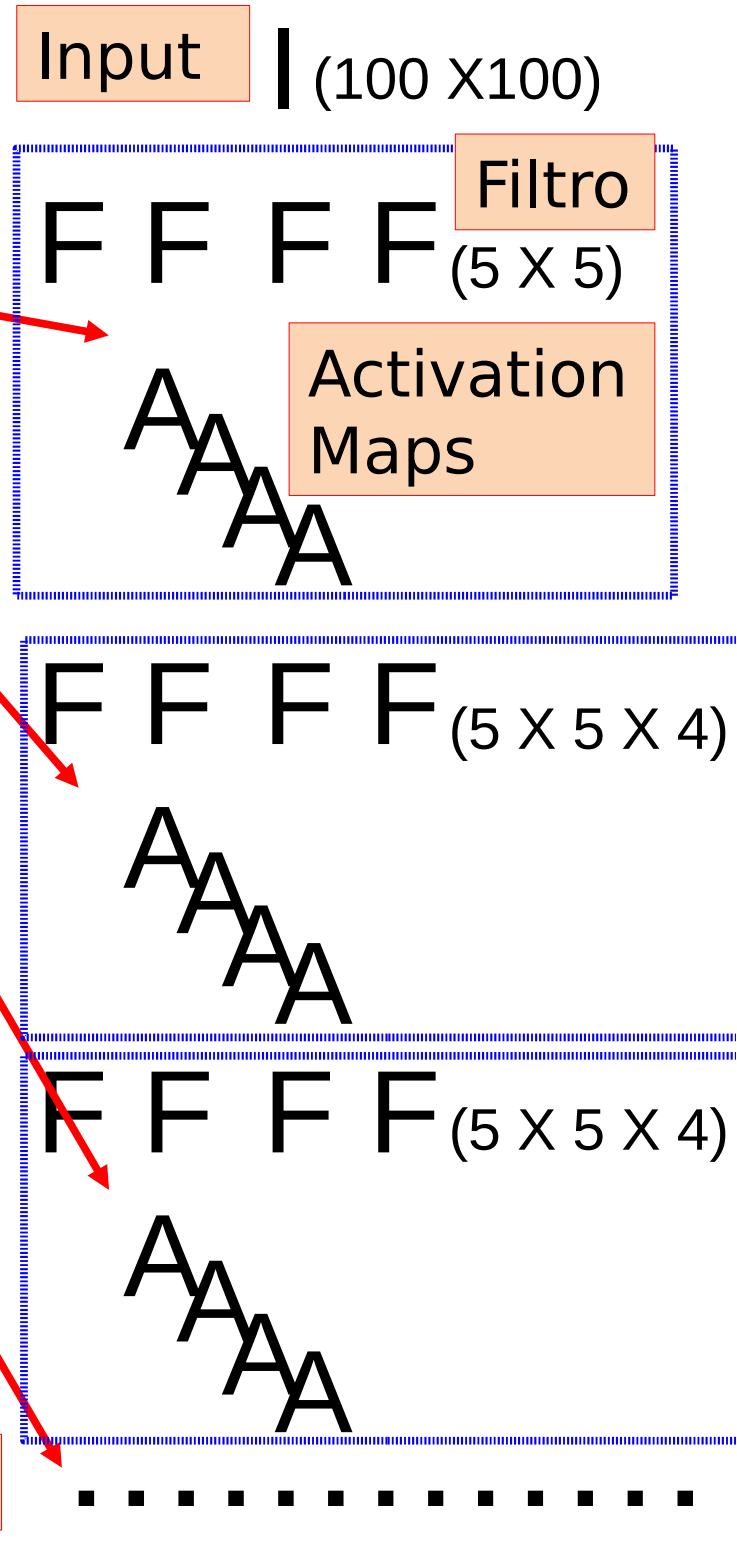
```
modelo %>%  compile(
    loss = 'binary_crossentropy',
    optimizer = RMSprop(),
    metrics = c('accuracy'))
```

Parámetros de Compilación

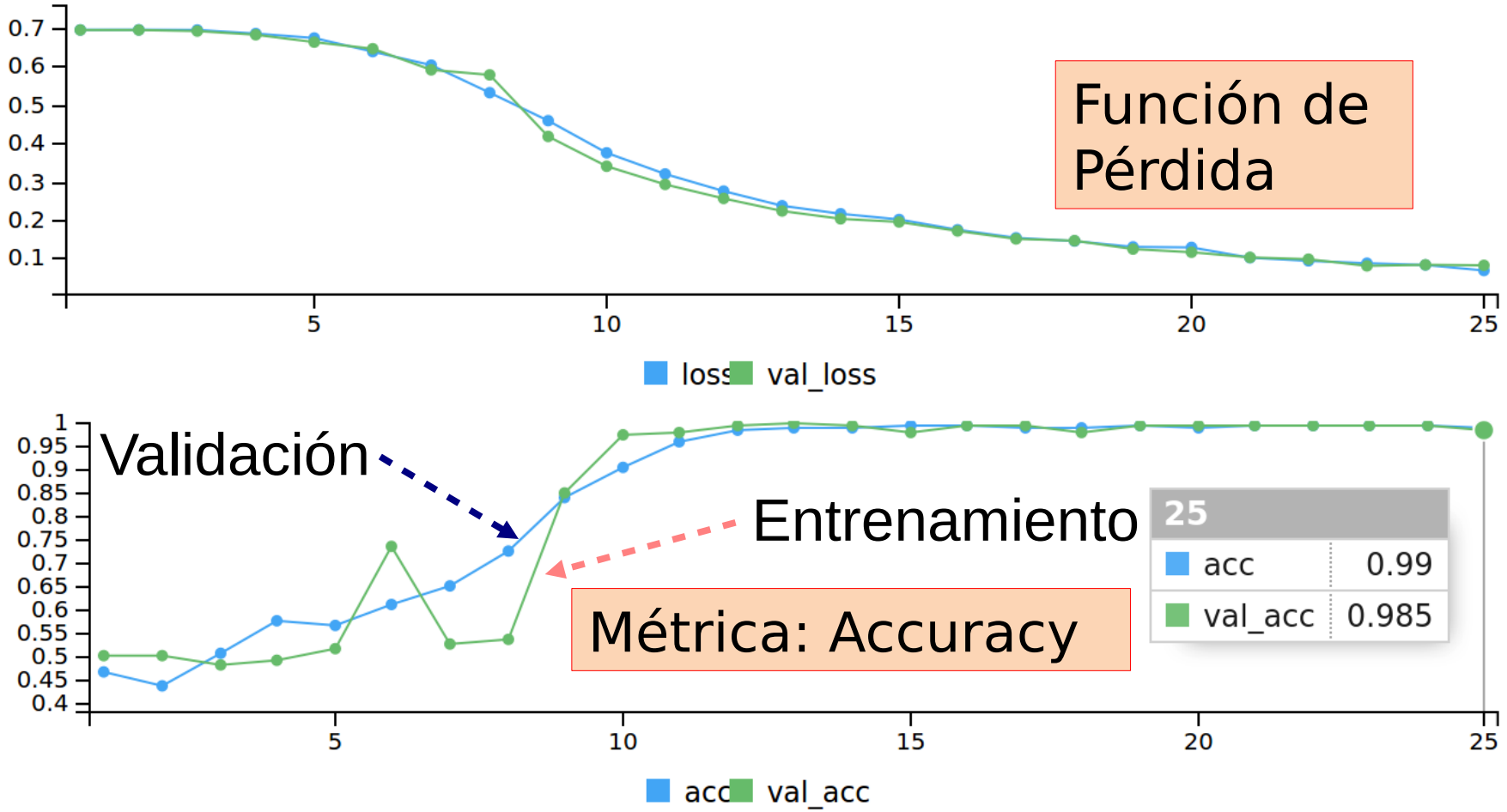
```
historia <- modelo %>%
  fit(array.train, label.train, batch_size = 1,
      epochs = 25,validation_data = list(array.test,label.test))
```

Arquitectura del Modelo

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



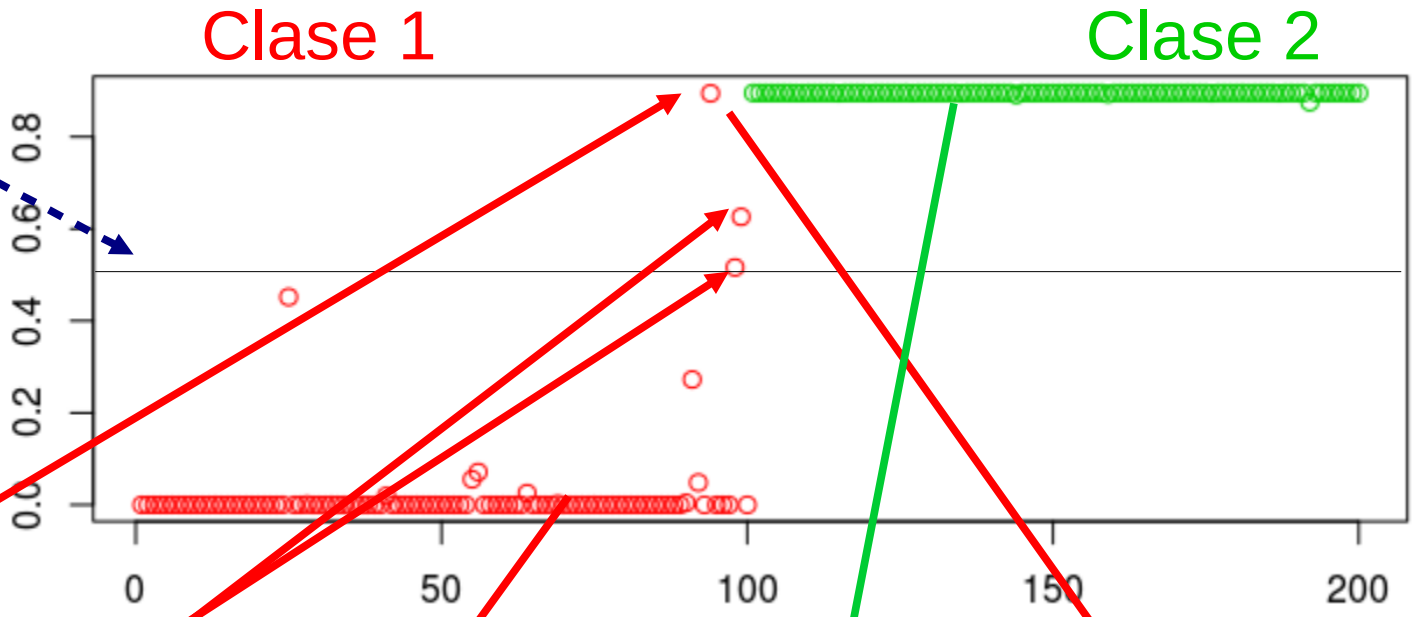
El Ajuste


0.7
0.6
0.5
0.4
0.3
0.2
0.1
5
10
15
20
25
loss
val_loss
Función de Pérdida
1
0.95
0.9
0.85
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
Validación
Entrenamiento
Métrica: Accuracy
25
acc 0.99
val_acc 0.985
acc
val_acc

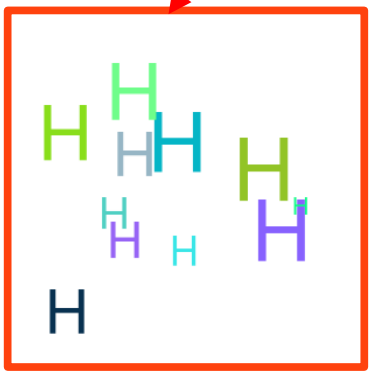
Desempeño de la Clasificación

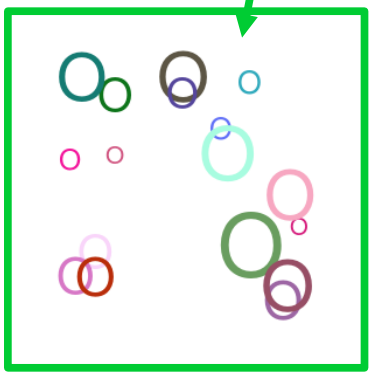
Umbral de Clasificación

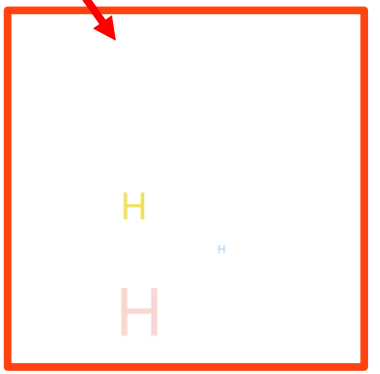
Score de Clasificación



	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

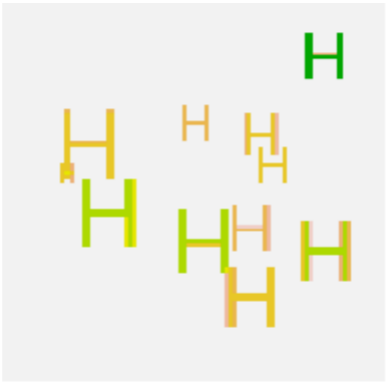
```
historia <- modelo %>%
  fit(array.train, label.train, batch_size = 1,
      epochs = 25,validation_data = list(array.test,label.test))
```

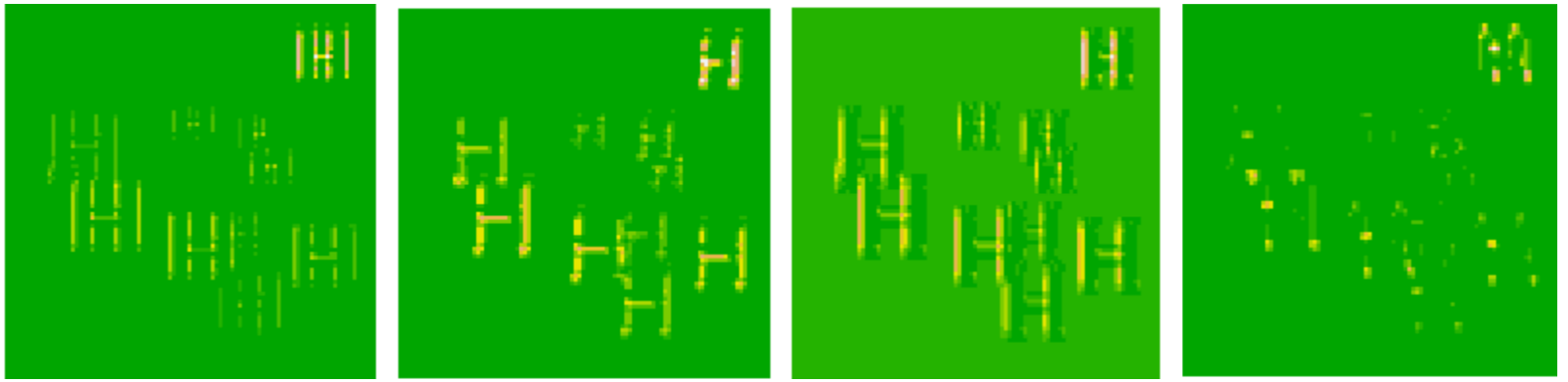
Que Aprendió la Red ?

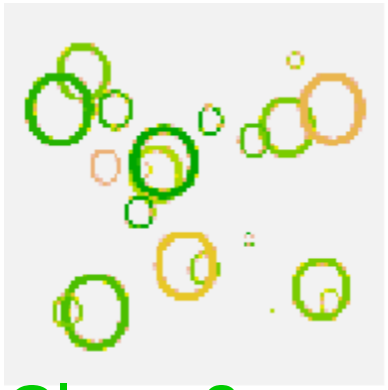
Mapas de Activación de la Primare Capa

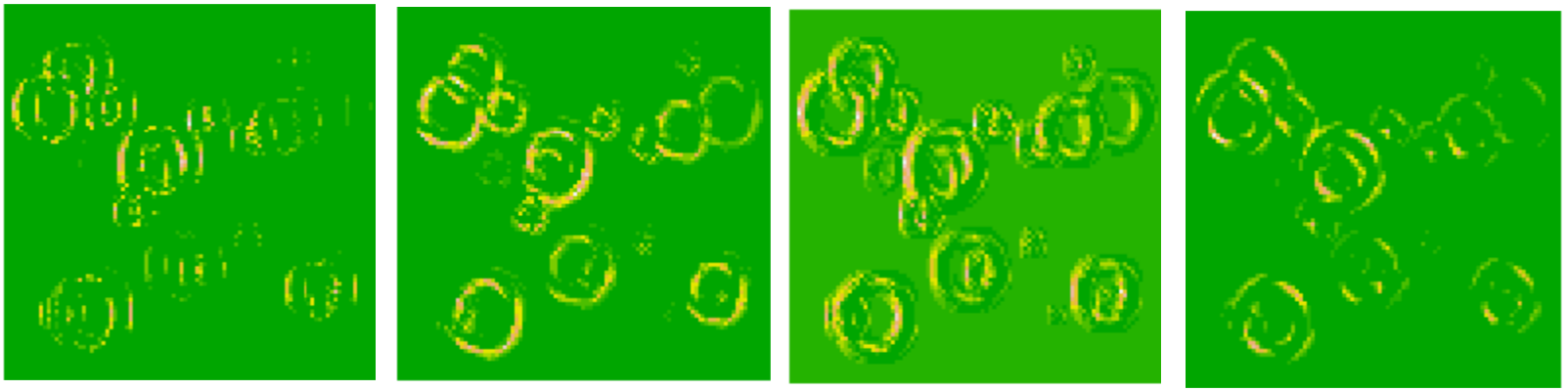
Filtro 1 | Filtro 2 | Filtro 3 | Filtro 4

Clase 1





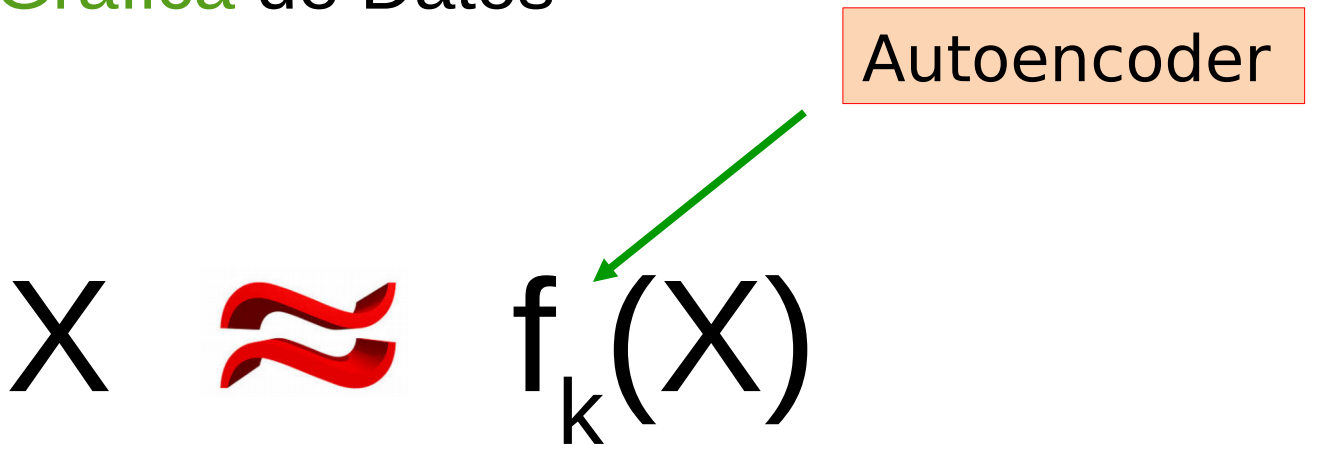




Clase 2

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



Inteligencia Artificial para la Detección de Fraudes en Seguros

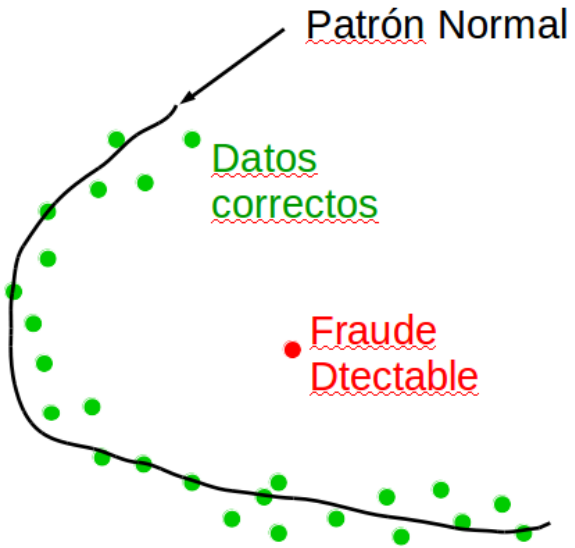
Una Aplicación para el Seguro Automotor

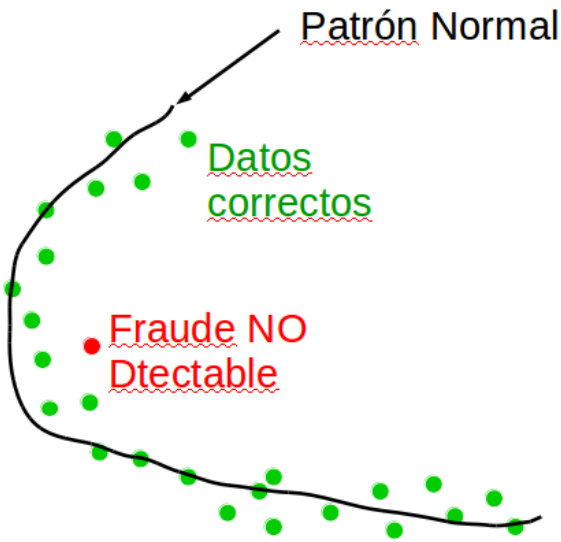
¿ Que es un Fraude en Seguros ?

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

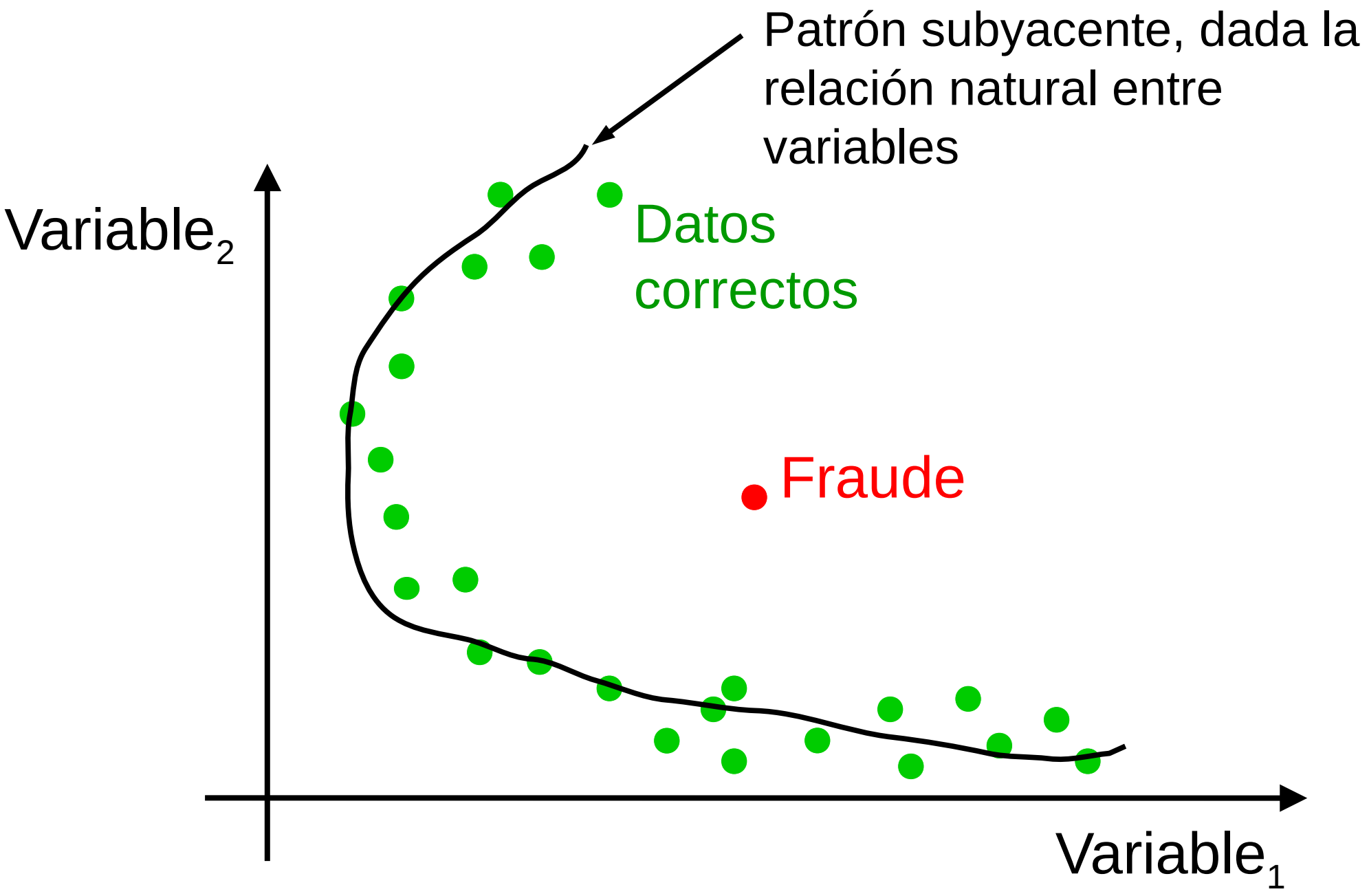
Dato SE ALEJA del patrón normal

Dato NO SE ALEJA del patrón normal

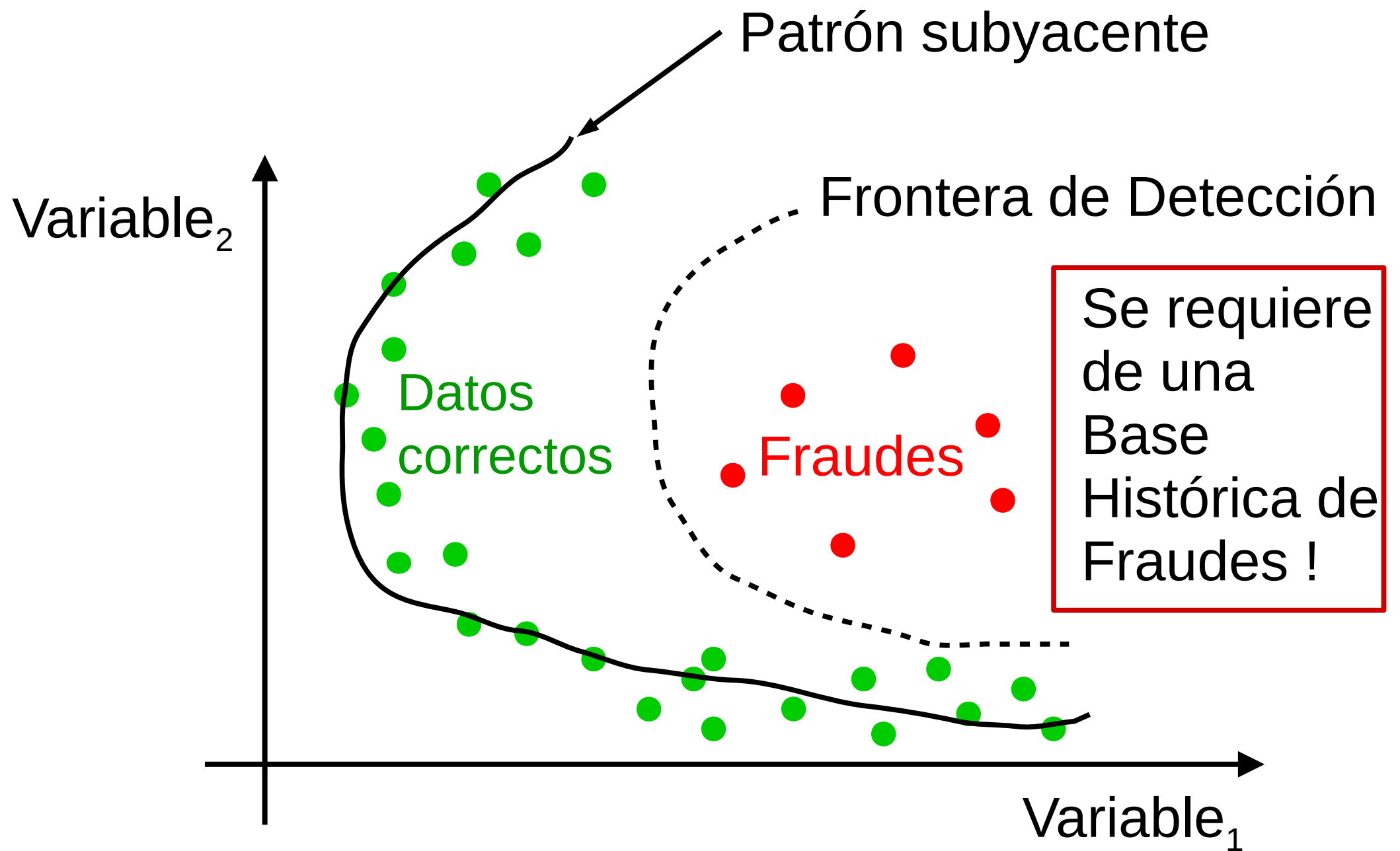




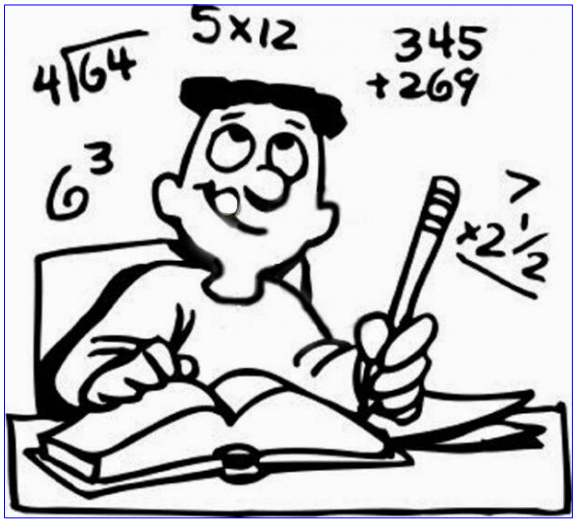
¿ Como se Vería un Fraude ?



Enfoque Supervisado para la Detección de Fraudes



¿ Como se Trabajaba Antiguamente ?



Patrón subyacente

Modelo Actuarial

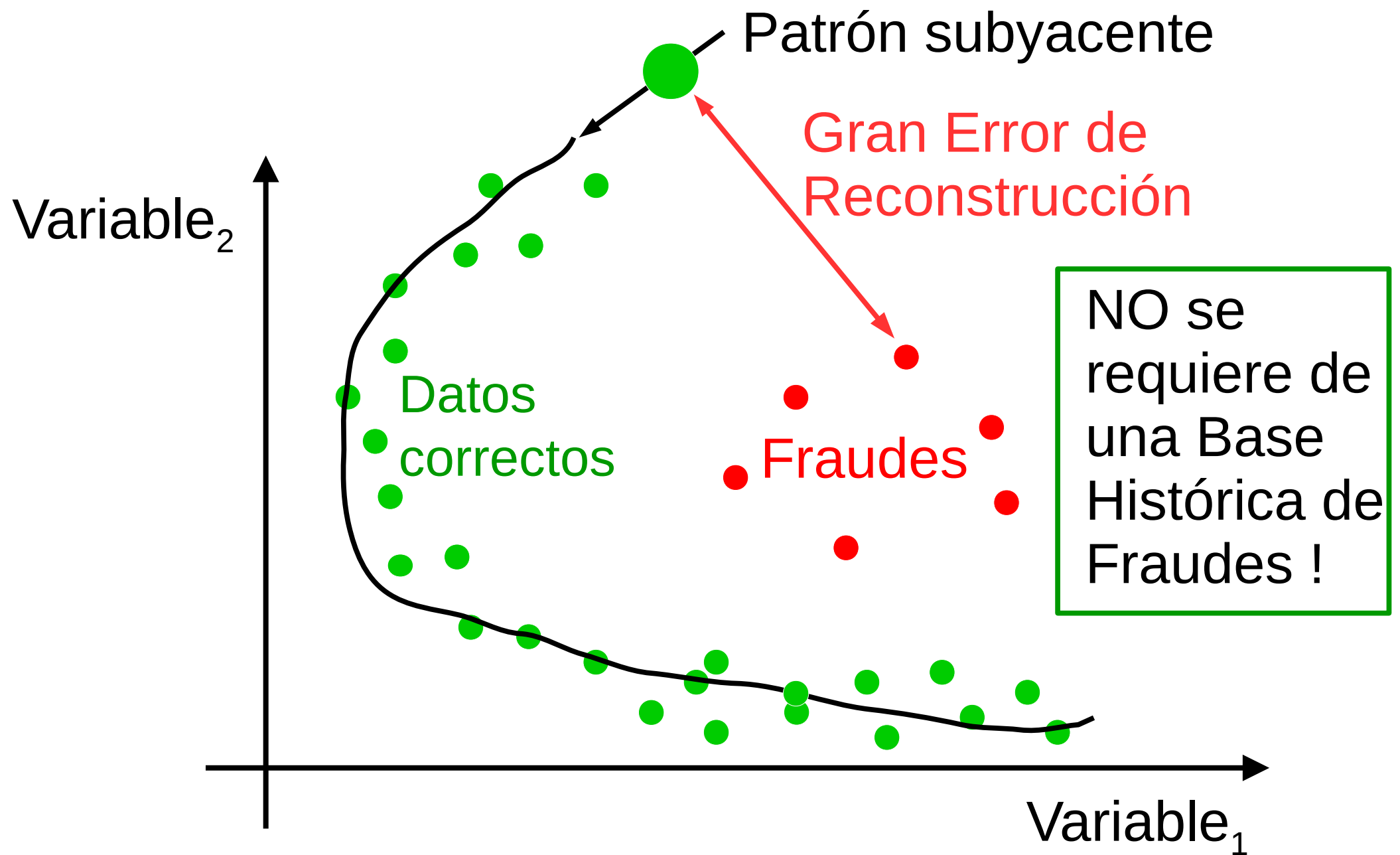
Variable$_2$

Datos correctos

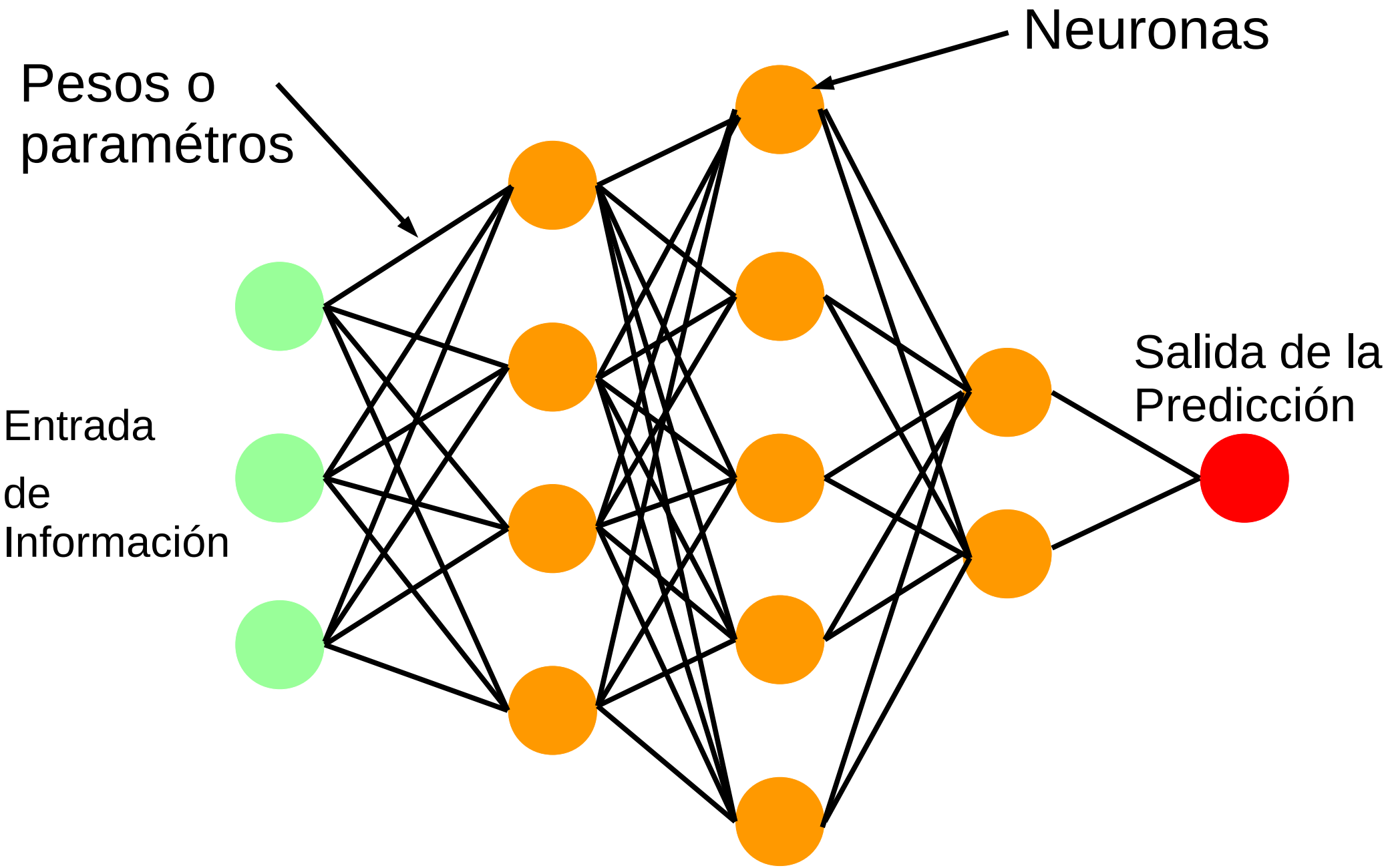
Fraudes

Variable$_1$

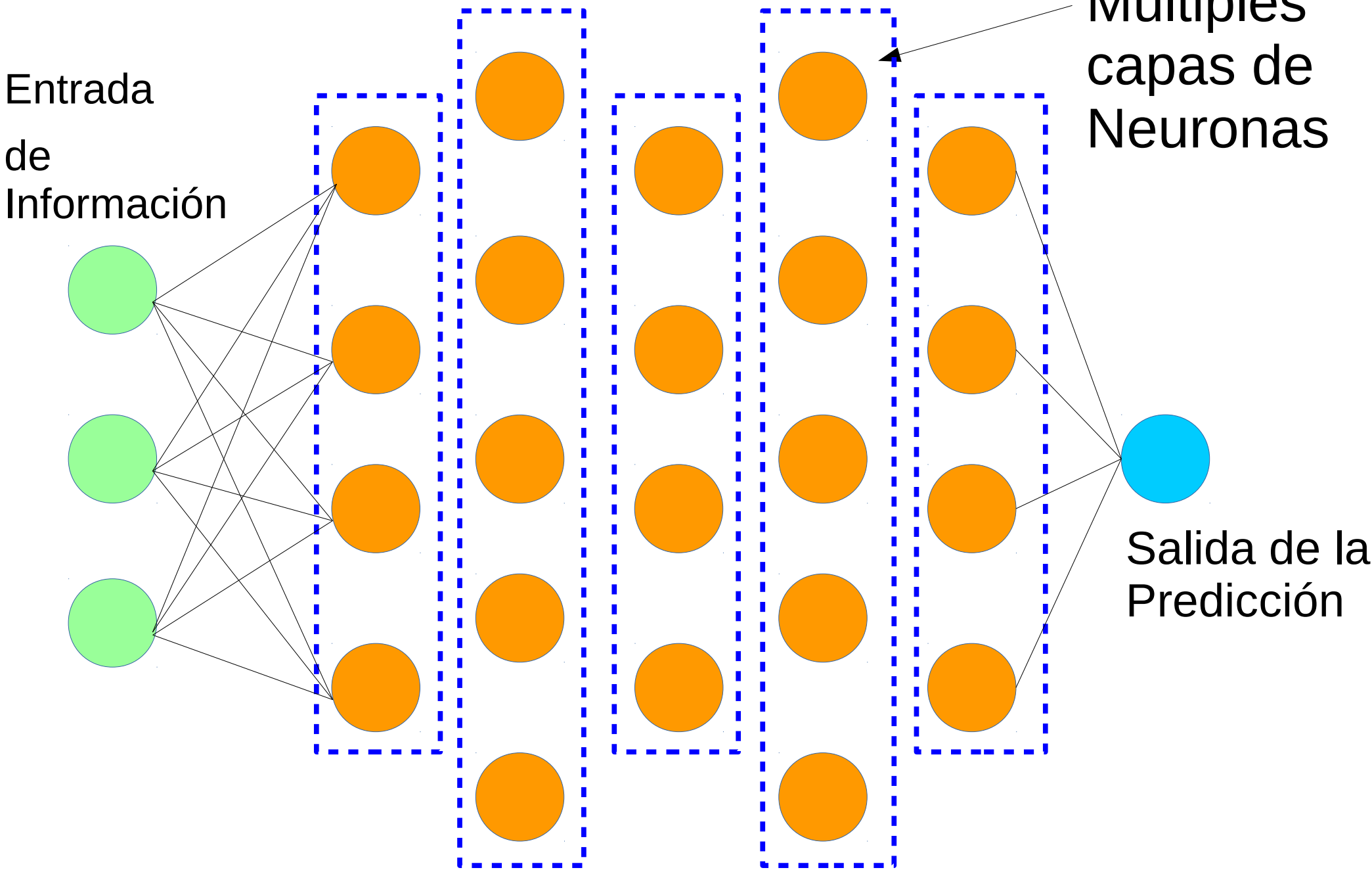
Enfoque No Supervisado



Redes Neuronales Artificiales



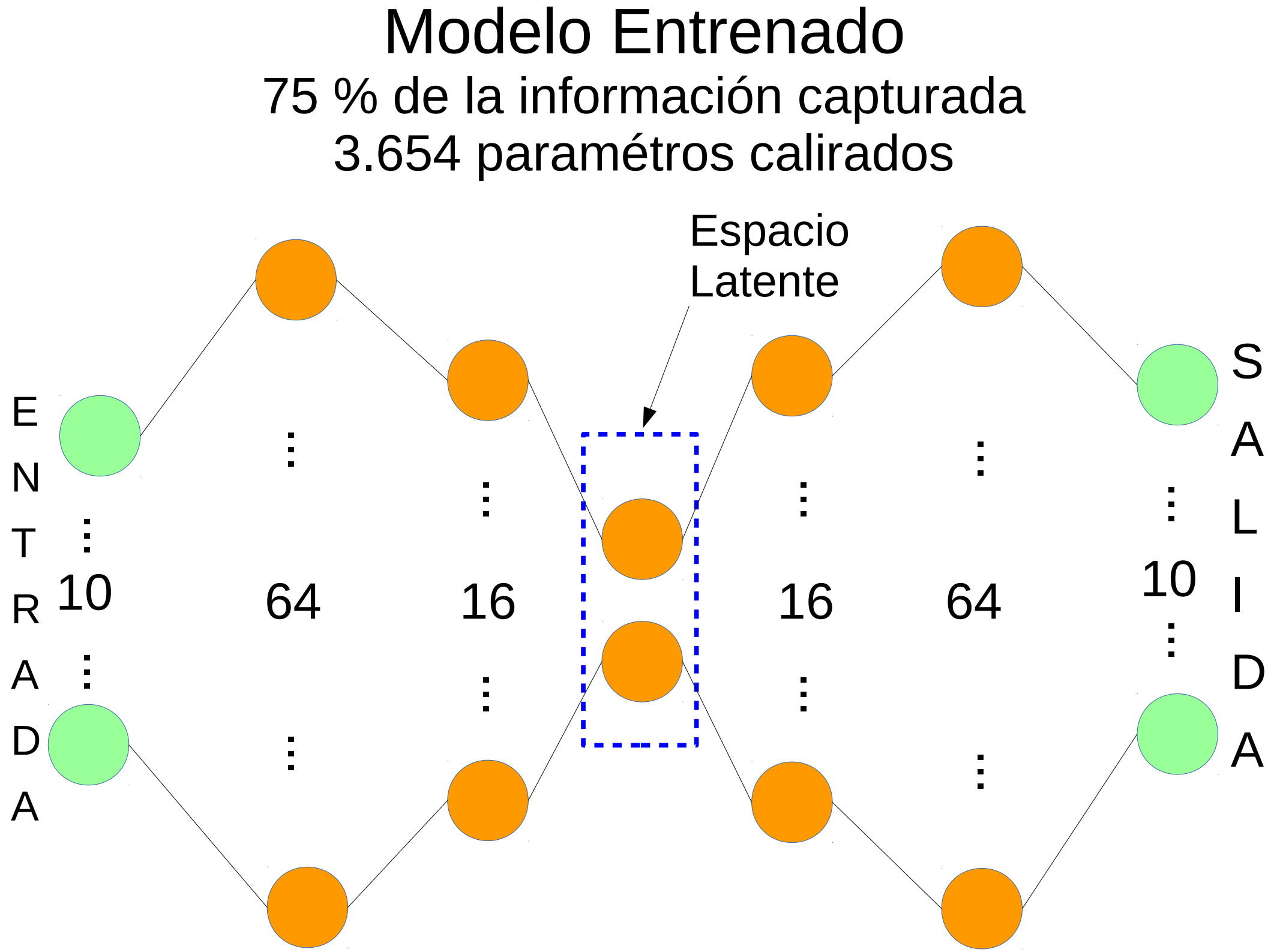
Deep Learning



Modelo Entrenado

75 % de la información capturada

3.654 paramétros calirados



Aplicación al Seguro Automotor

SUSEP 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)

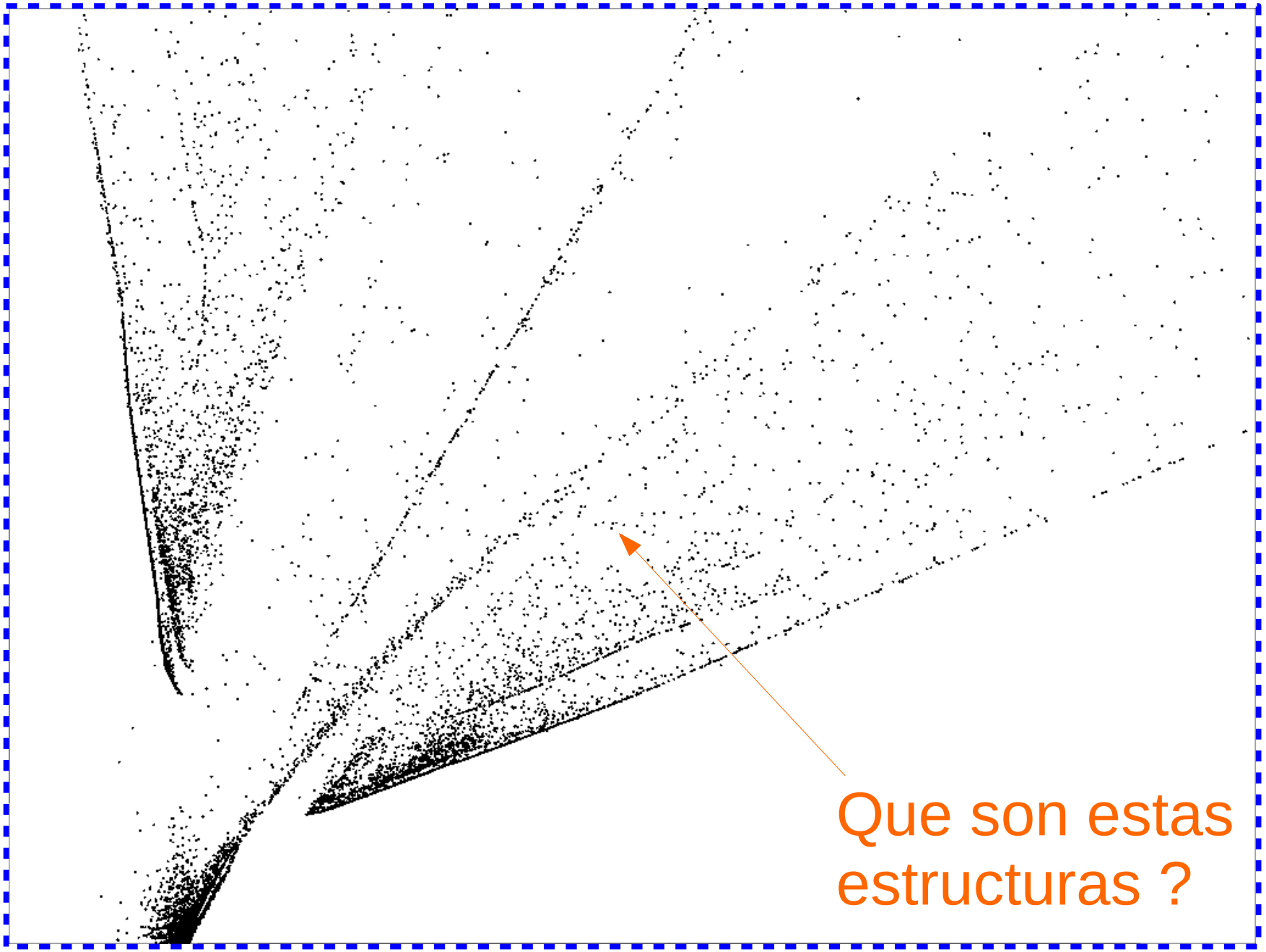
El Autoencoder en KERAS

```r
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 aproximation
genrec <- keras_model(ent, salida)
```

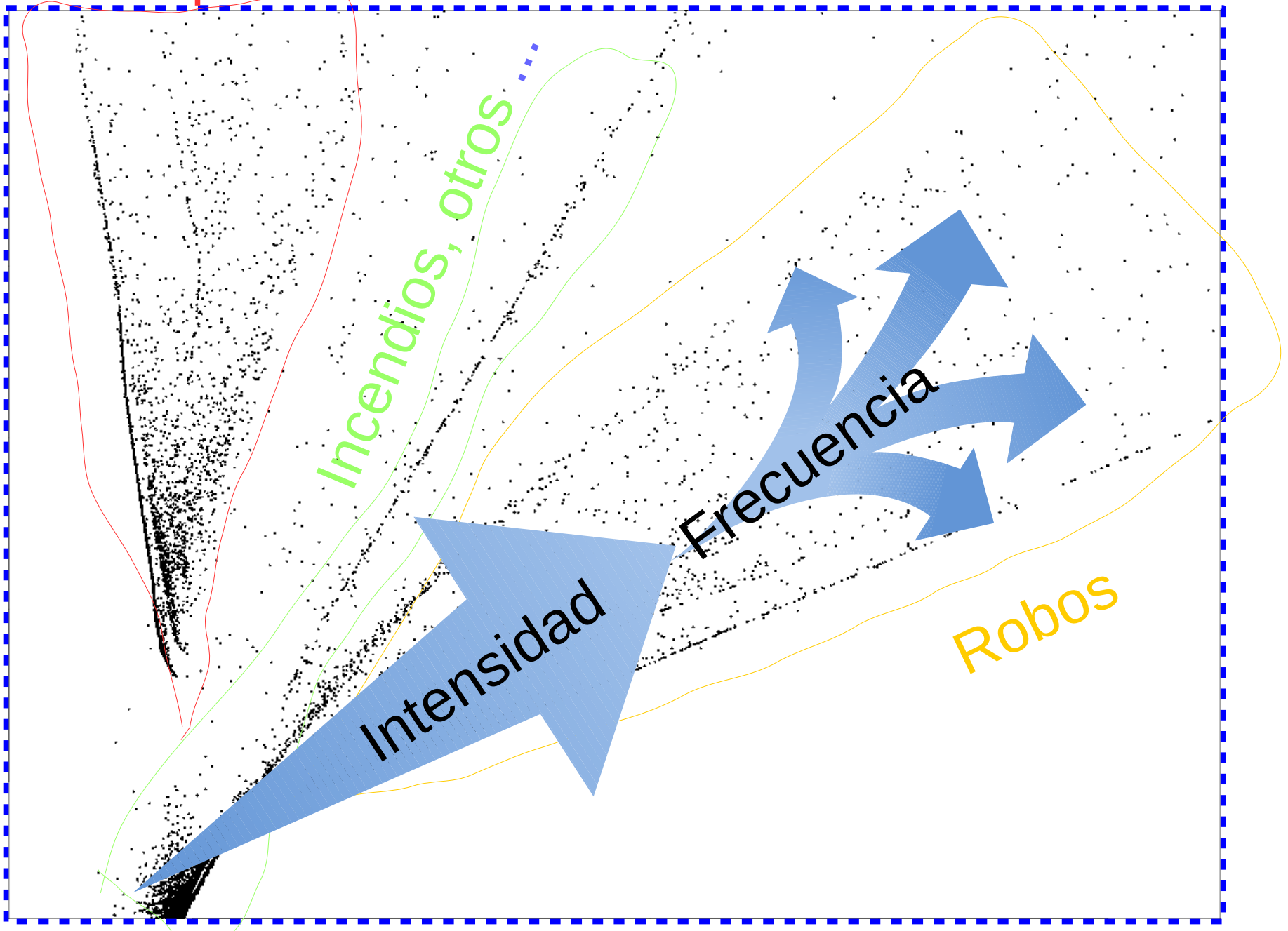
$X \approx f_k(X)$

```r
# 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),
)
```

¿ Como es el Espacio Latente de los Siniestros ?



¿ Como es el Espacio Latente de los Siniestros ?



¿ Cuáles 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

???