Farielberry Lab

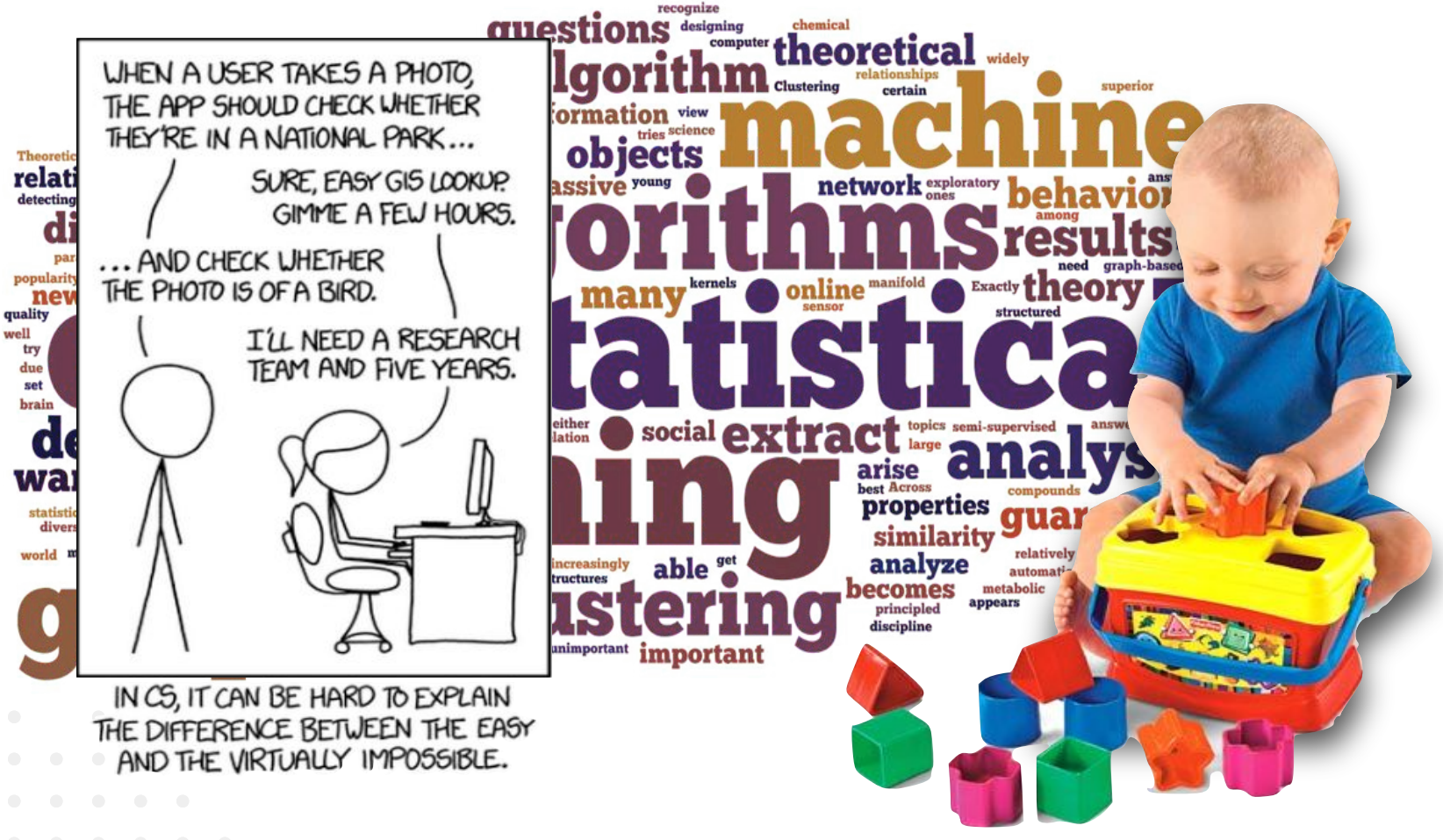
Deep Learning for Genomic Prediction

FACULTAD DE INGENIERÍA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Machine / Deep Learning 101

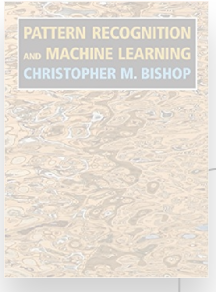
Machine Learning



Machine Learning

"El estudio de cómo las máquinas pueden, observando el ambiente, aprender a distinguir patrones de interés de un fondo y real decisiones razonables sob categorías de los mismos.

El Aprendizaje Automático es una eotos, betos, es a el en etse las.



"Descubrir automáticamente regularidades en los datos usando algoritmos en computadora y a partir de esas regularidades tomar acciones."

medida de desempeño

aprendizaje: mejorar con la experiencia en una tarea

datos aplicación

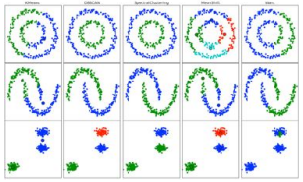
ML categories of problems

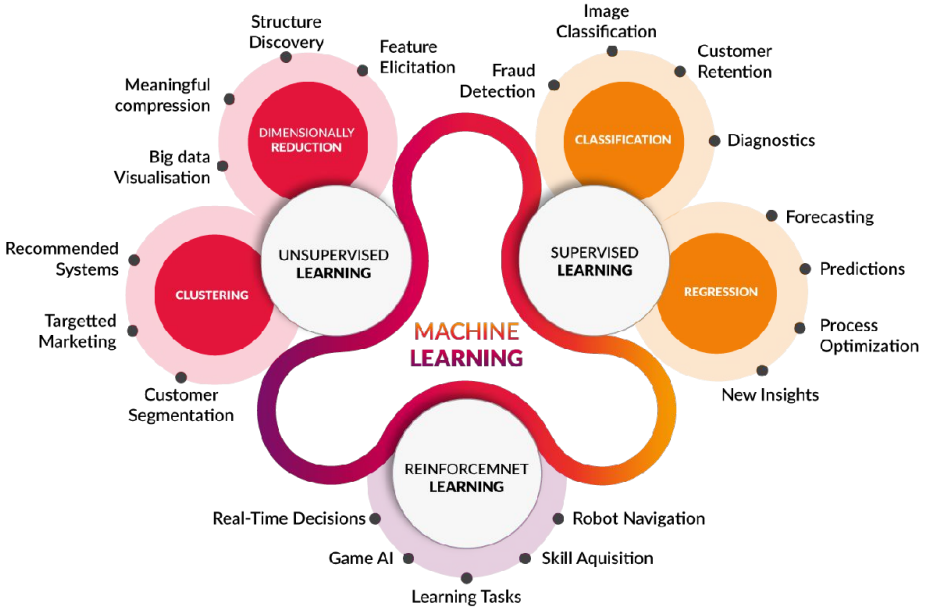
semi-supervisado
se conocen las etiquetas (clases) de algunos datos

no supervisado
no se conocen las etiquetas (clases) de los datos, se organizan a partir de las características.

identificación de patrones o estructura / evaluación indirecta o cualitativa / organiza/agrupa clustering / paramétrico o no / # clases conocida o no

k-means, Fuzzy C-means, Hierarchical Clustering, Spectral Clustering, Gaussian Mixtures, Hidden Markov Model, Neural Networks, (Generalized/Robust) PCA, Isomap, MDS, Diffusion Maps, ...





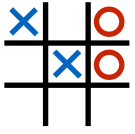
supervisado
datos de entrenamiento con etiquetas (clases) o valores de salida correctos para predecir datos nuevos

clasificación y regresión / aprendizaje explícito / evaluación directa / predicción de clase o valor / paramétrico o no

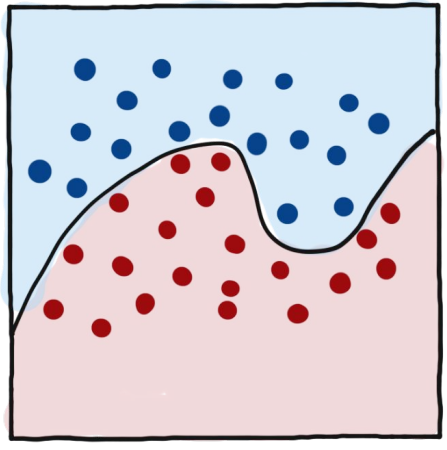
Nearest Neighbor, Support Vector Machines (SVM), Decision Trees, Random Forest, Discriminant Analysis, Naive Bayes, Neural Networks, Linear Regression, SVR, ...

reinforcement learning
realimentación del resultado de la tarea: recompensa o penalización

aproximación a IA / definir estrategias ante eventos / maximizar recompensa



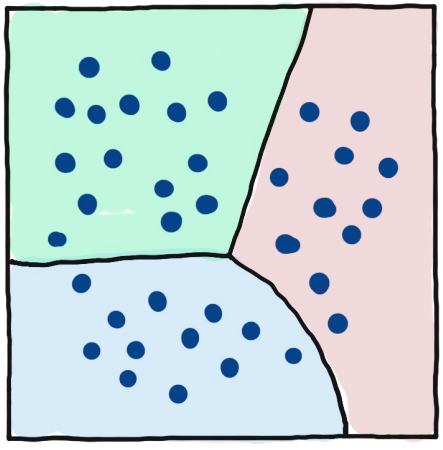
ML categories of problems



Classification

Supervised

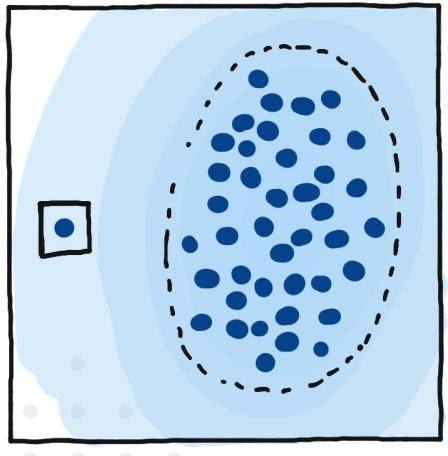
Diagnostics, classification, spam detection, OCR, ...



Clustering

Unsupervised

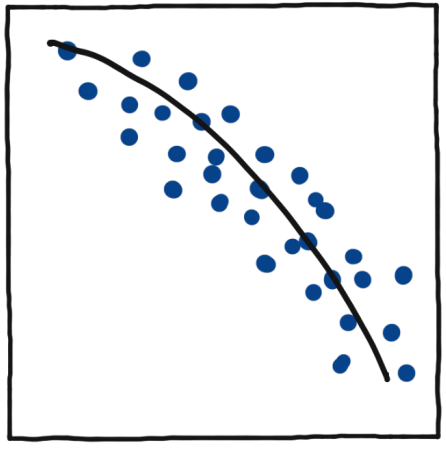
Recommender systems, segmentation, personalized advertising, data analysis, ...



Anomaly detection

Supervised

Fraud detection, failures, impostors, ...

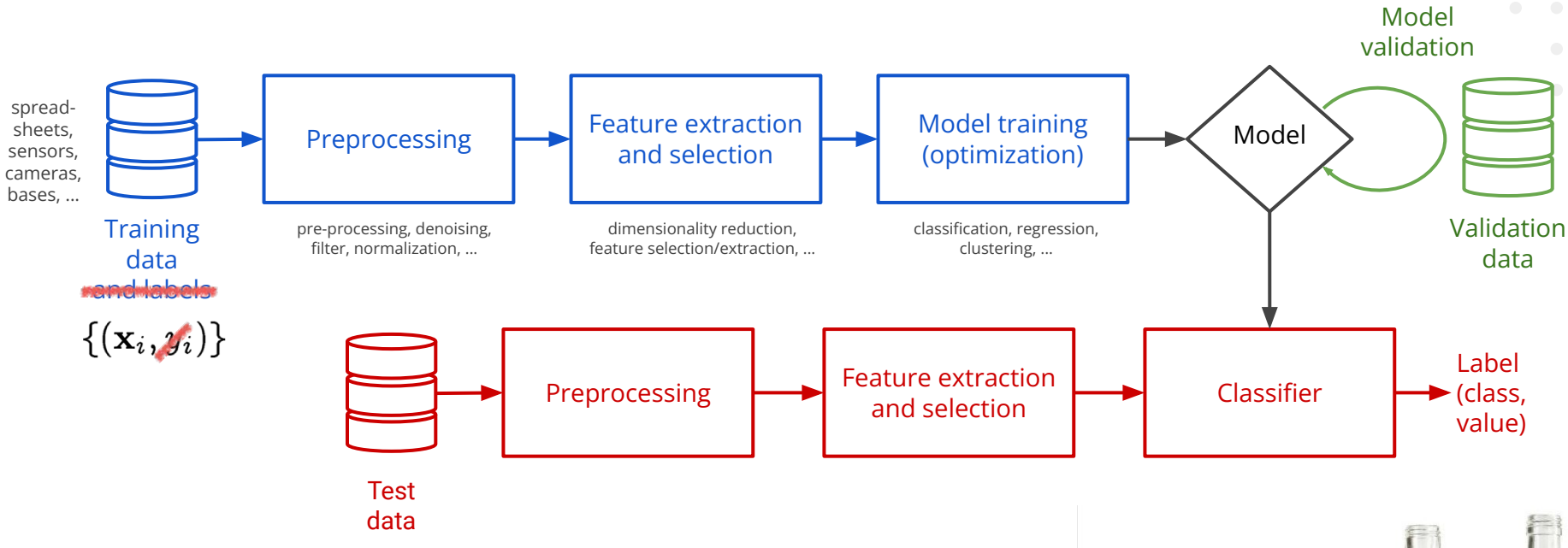


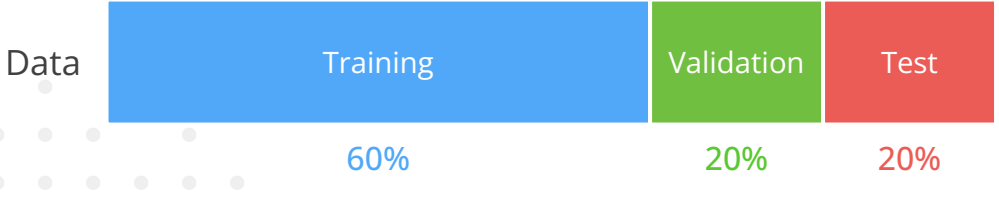
Regression

Supervised

Variable prediction, process optimization, ...

Machine Learning diagram

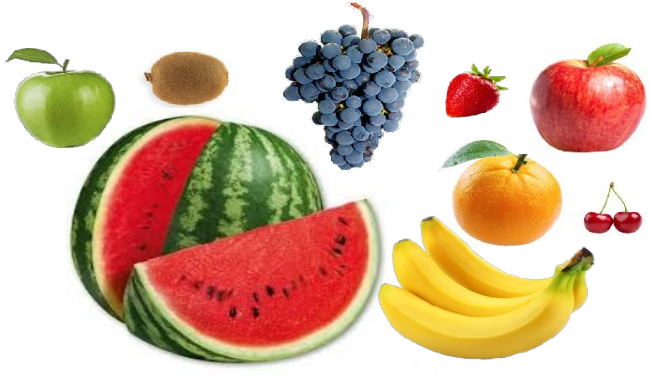




Feature selection and extraction

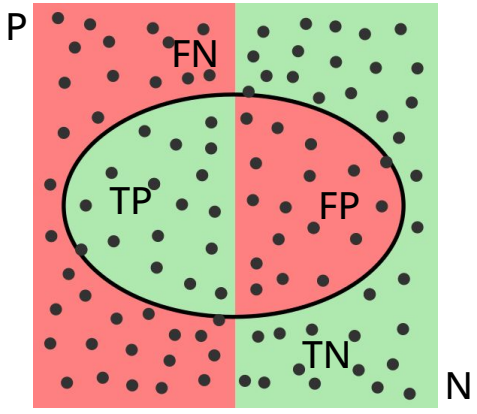
- Extract information that may allow discrimination.
- Eliminate redundant or irrelevant information.
- Reduce the dimensionality of the problem.
- Extraction: create new features combining the original features. Determine a set of lower dimensionality in the original space.
- Selection: select the features (original or transformed in the extraction process) with the highest discrimination power.
- Designed by subject matter experts or discovered for their discriminating power.

Botella de *vidrio* de altura h_b, ancho de base w_b, altura pico h_n y ancho de pico w_n



Features?

Performance metrics (classification)



success	correct rejection	false alarm (error I)	mistake (error II)
TP: true positives	TN: true negatives	FP: false positives	FN: false negatives
		login	disease

Accuracy: how close is it to the actual values?

Sensitivity/Recall: how many patients are correctly detected?

Specificity: how many healthy ones are not selected?

Precision: how many of those selected are sick?

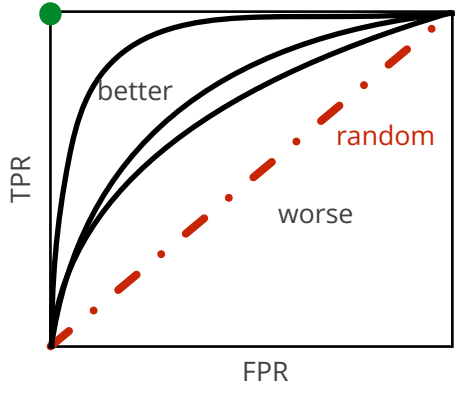
False Positive Rate: how many healthy people are selected?

F1 score: harmonic mean between TPR and PPV.

Confusion matrix

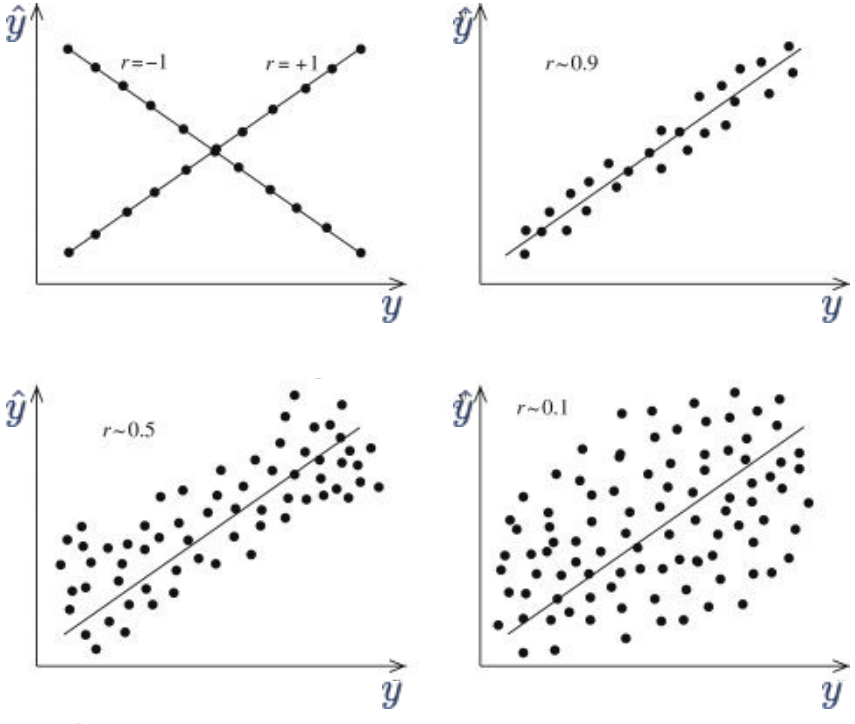
Predicted label \ Real label	enfermos	sanos
enfermos	TP	FP
sanos	FN	TN

Receiver Operating Curve (ROC)



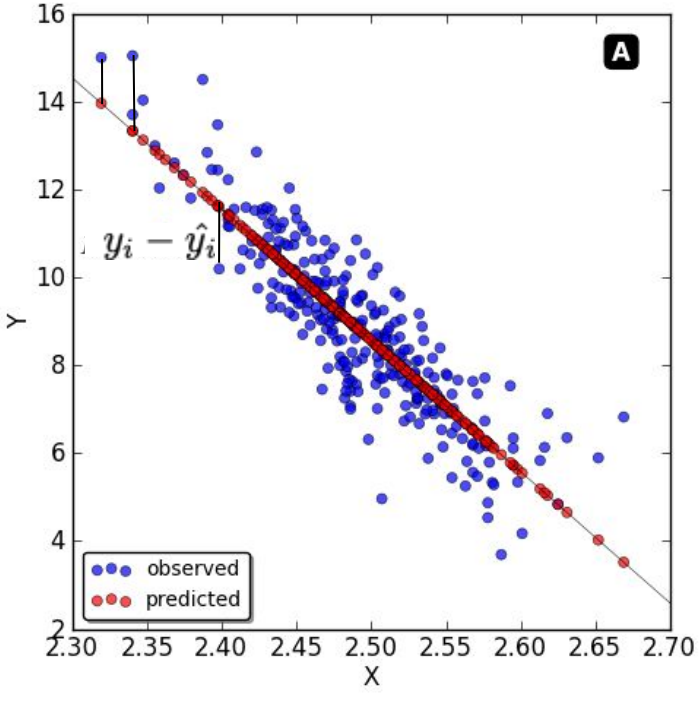
Performance metrics (regression) - cost functions $\mathcal{L}$

Pearson correlation coefficient (r)



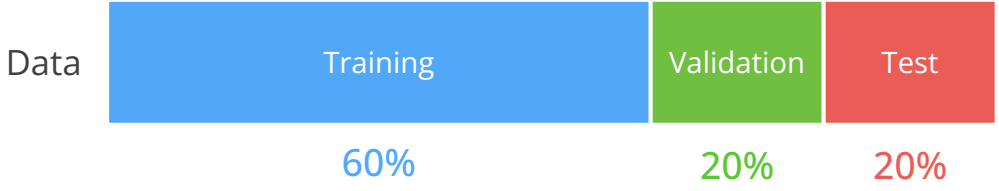
$$r_{\hat{y}y} = \frac{\sum_{i=1}^{n} \left(\hat{y}_i - \bar{\hat{y}}\right)\left(y_i - \bar{y}\right)}{\sqrt{\sum_{i=1}^{n} \left(\hat{y}_i - \bar{\hat{y}}\right)^2} \sqrt{\sum_{i=1}^{n} \left(y_i - \bar{y}\right)^2}}$$

Mean Square Error (MSE)

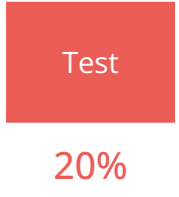


$$\mathrm{MSE}(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2$$

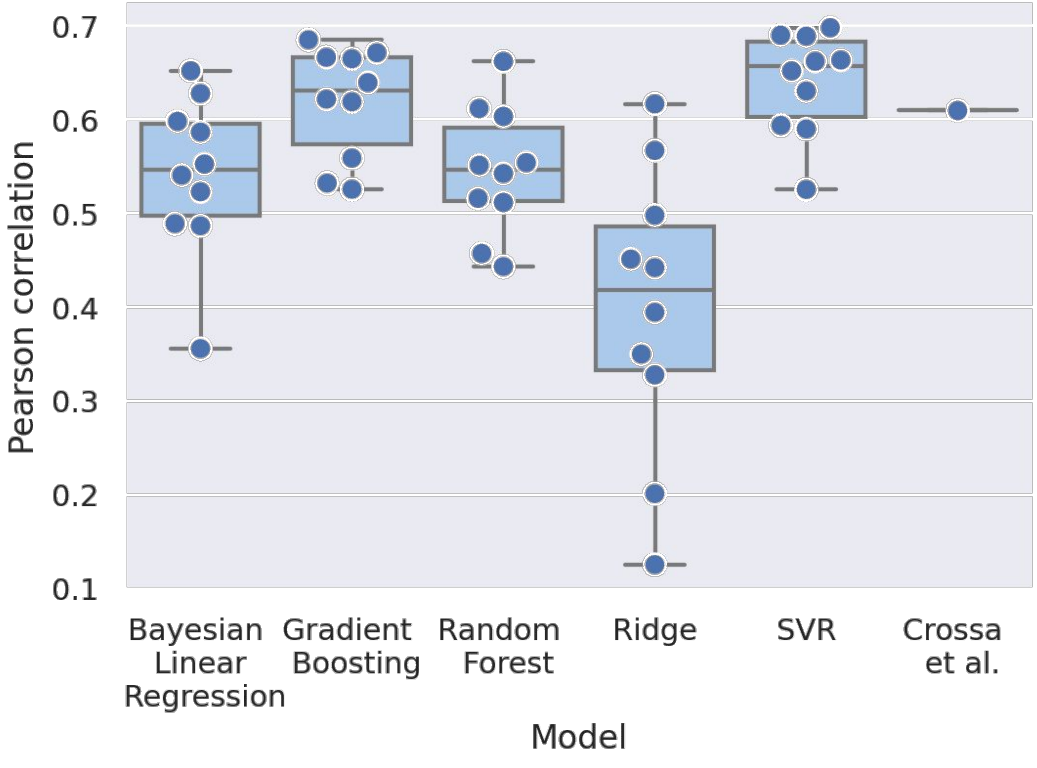
Cross-validation: k-fold



k = 4

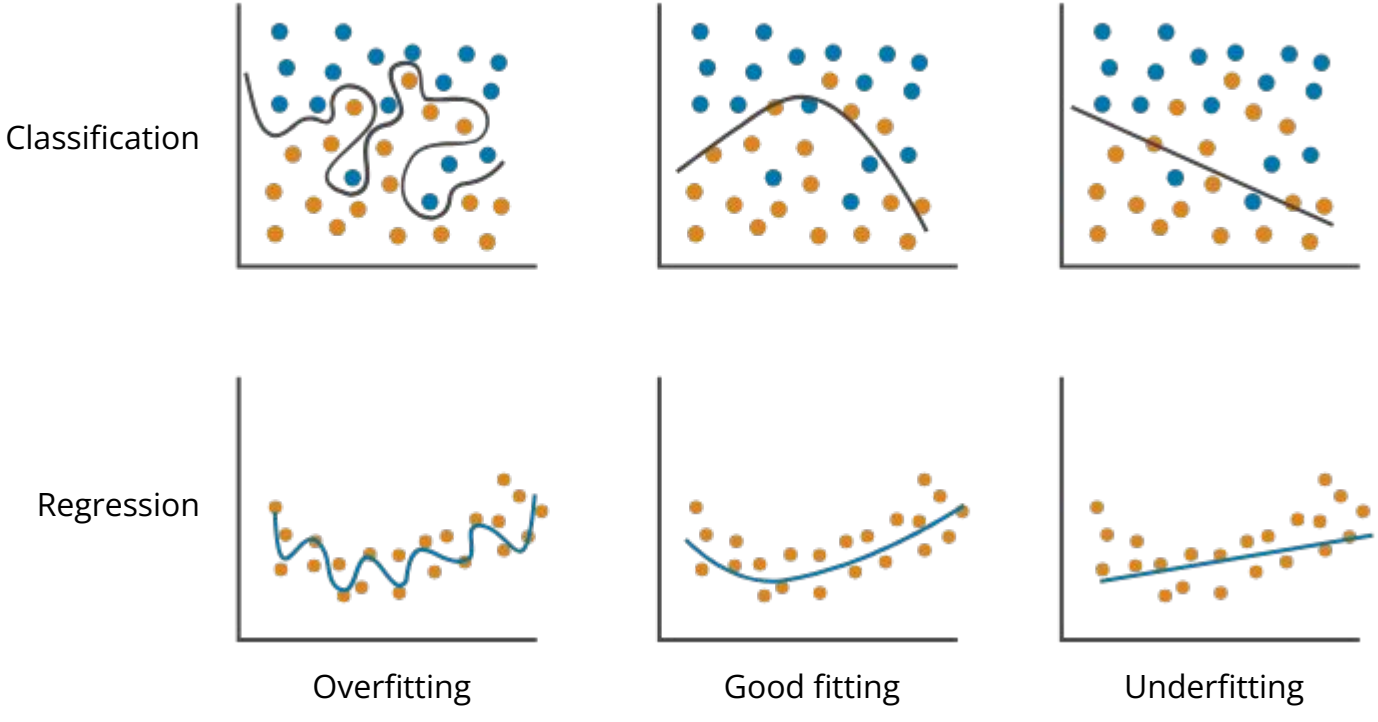


Example: 10-fold cross-validation



Wheat dataset
(n: 599, p: 1279)

Underfitting, overfitting and generalization

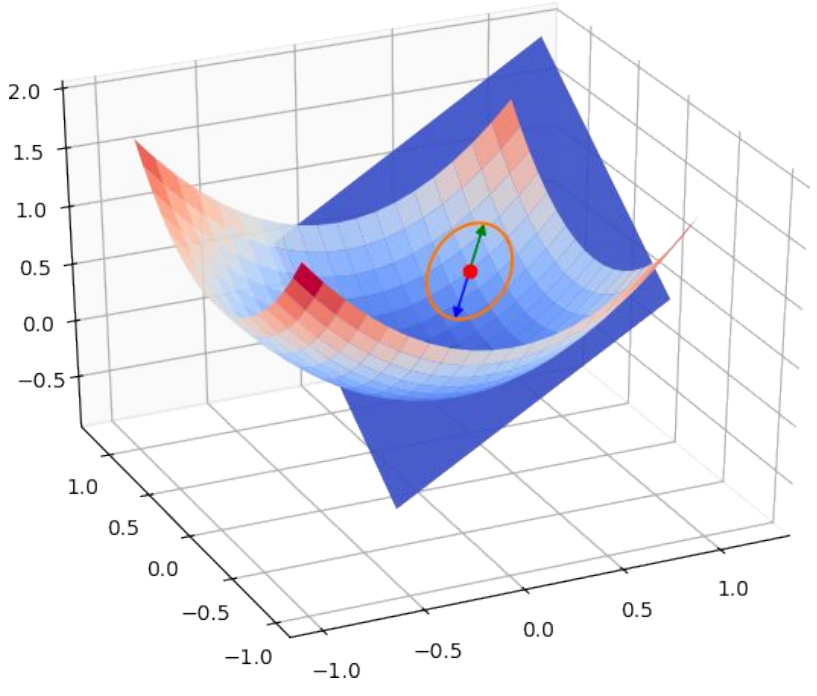


Gradient descent (GD)

- Algorithm (iterate):
  - Calculate the gradient at that point.
  - Step in the direction opposite to the gradient.

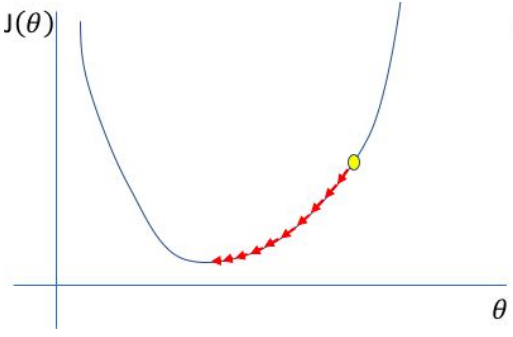
$$x[k+1] = x[k] - \eta \nabla f(x[k])$$

- Learning rate.
- Stop criteria.
- Starting point.
- Depending on the size of the data: batch GD, stochastic GD, mini-batch GD.
- Optimizers: Momentum, Adam, ...

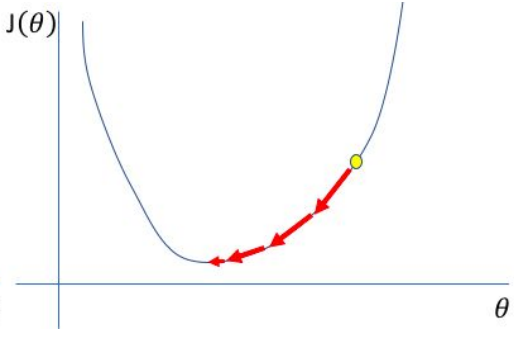


Gradient descent: learning rate

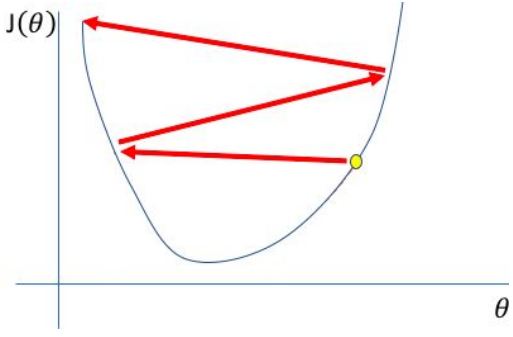
Too low



Right



Too high



https://www.i-am.ai/gradient-descent.html

Neural Networks

Perceptron

- Algorithm for *supervised* learning of *binary* classifiers.
- McCulloch-Pitts neuron (1943)
  - Rosenblat (1958)
- *Linearly separable* data.
- *Simplified model* of a biological neuron.

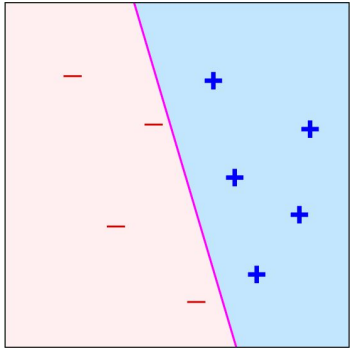
$$\mathbf{x} = (x_1, \ldots, x_d)$$

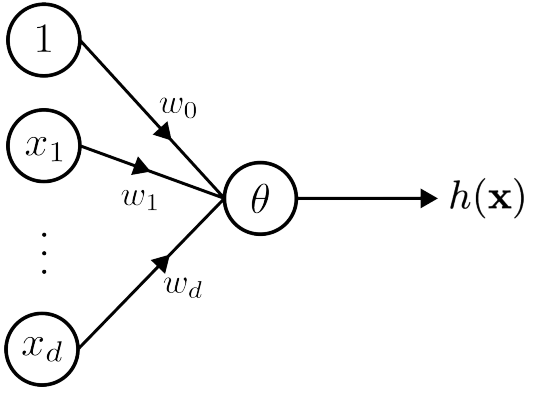
$$\mathbf{x} = [1, x_1, \cdots, x_d]^T$$

$$\mathbf{w} = [w_0, w_1, \cdots, w_d]^T$$

$$h(\mathbf{x}) = \text{sign}\left(w_0 + \sum_{i=1}^{d} w_i x_i\right)$$

$$= \text{sign}\left(\mathbf{w}^\top \mathbf{x}\right)$$

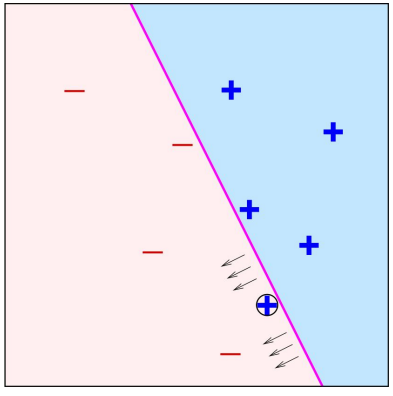




Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65.6 (1958): 386.

Perceptron

- Learning the weights' vector
  - Given a training set $y_i = \pm 1$
    $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$
  - Choose an initial (random) $\mathbf{w}_0$
  - Pick a (random) misclassified $\mathbf{x}_k$ point (error), i.e.
    $\text{sign}(\mathbf{w}^\top \mathbf{x}_k) \neq y_k$
  - Update the weights
    $\mathbf{w}(t+1) \leftarrow \mathbf{w}(t) + y_k \mathbf{x}_k$
  - Until there is no error. Guaranteed?
- Stop criteria
  - Error variation, iterations, ...



Show that, if $\mathbf{x}_k$ misclassified,

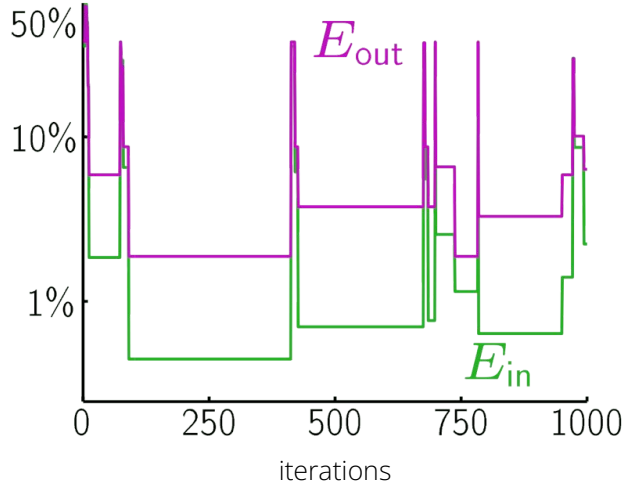
(a) $y_k \, \mathbf{w}^\top(n) \, \mathbf{x}_k < 0$

(b) $y_k \, \mathbf{w}^\top(n+1) \, \mathbf{x}_k > y_k \, \mathbf{w}^\top(n) \, \mathbf{x}_k$

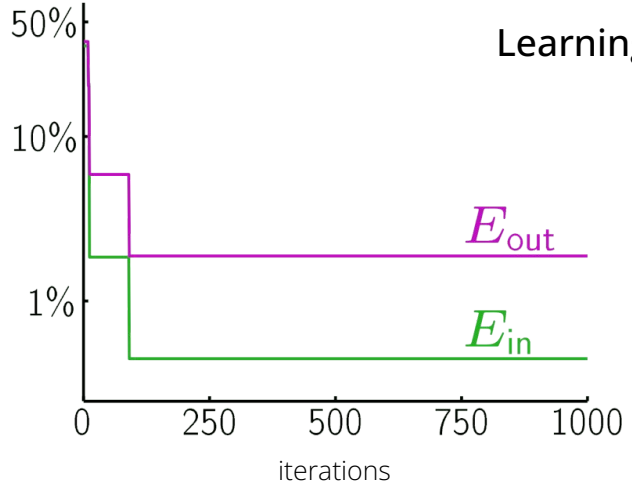
(c) the threshold moves "in the right direction".

Perceptron

PLA:



Pocket: Only update w if the error decreases.



Learning curves.

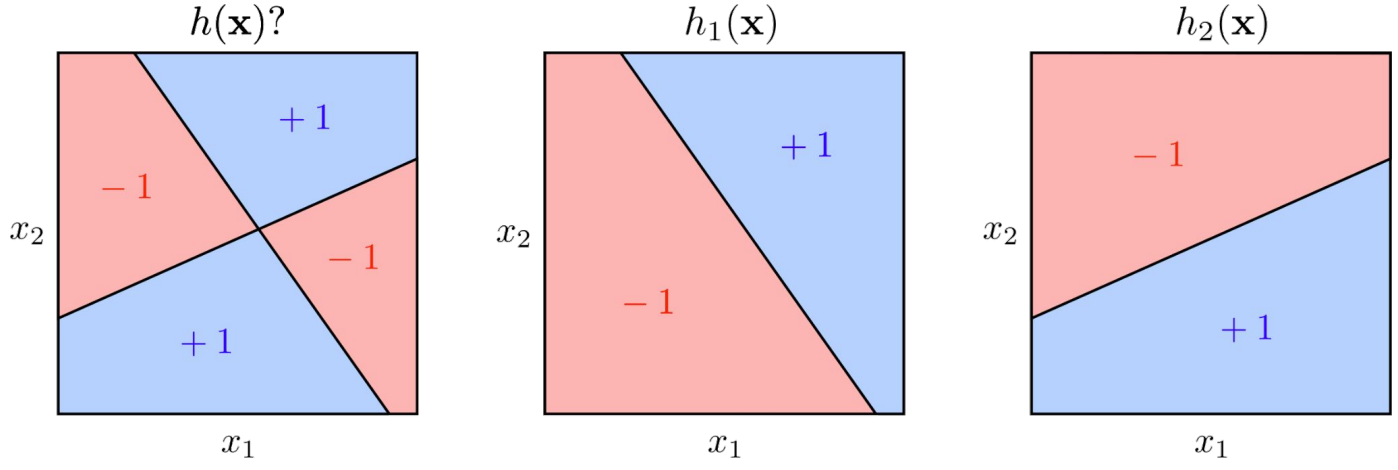
E_{in}: *in-sample* error

E_{out}: *out-of-sample* error

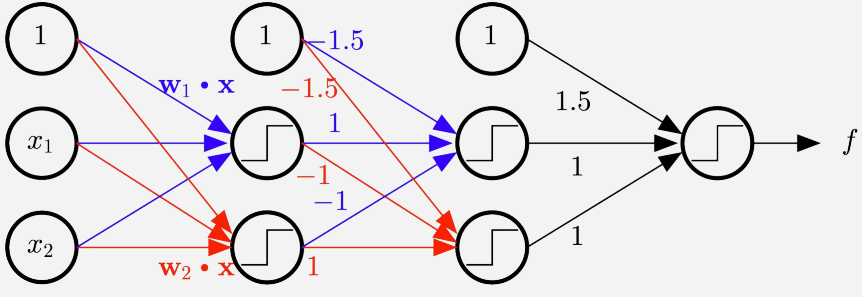
Hoeffding inequality:

$$\mathbb{P}\left[\left|E_{\text{in}} - E_{\text{out}}\right| > \epsilon\right] \leq 2Me^{-2\epsilon^2 N}$$

Perceptron

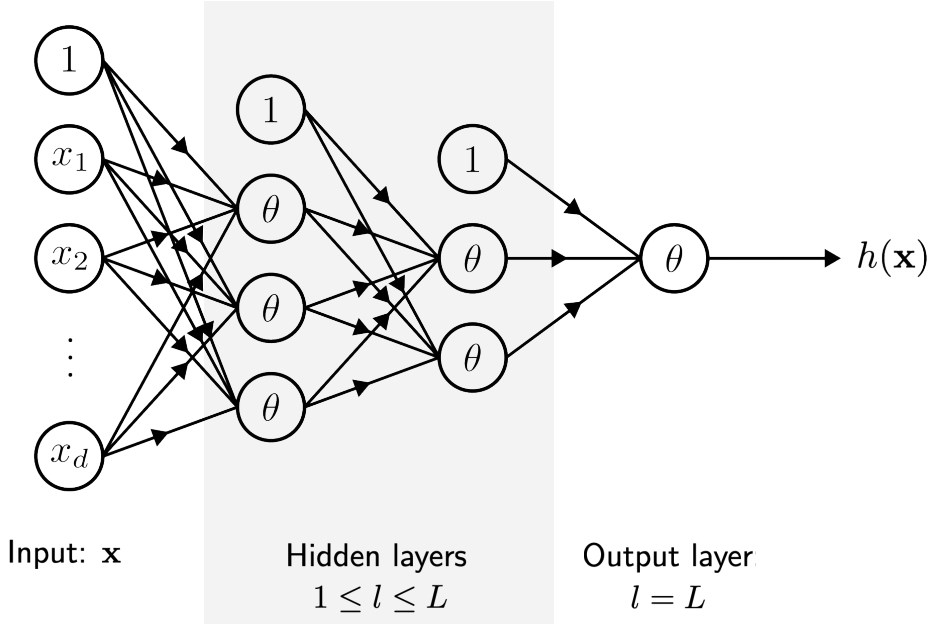


$$h(\mathbf{x}) = \text{XOR}(h_1, h_2) = h_1(\mathbf{x})\bar{h}_2(\mathbf{x}) + \bar{h}_1(\mathbf{x})h_2(\mathbf{x})$$
$$\text{OR}(x_1, x_2) = \text{sign}(x_1 + x_2 + 1.5)$$
$$\text{AND}(x_1, x_2) = \text{sign}(x_1 + x_2 - 1.5)$$

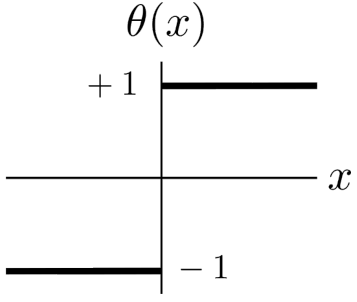


Multi-Layer Perceptron (3 layer feed-forward)

Multi-Layer Perceptron (MLP)

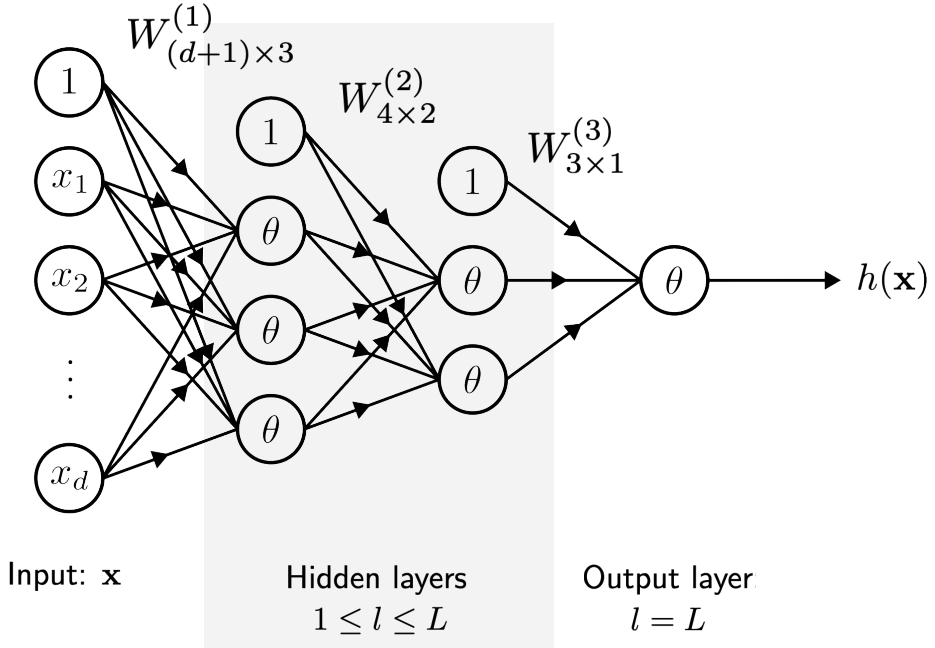


Activation Function
(not differentiable)



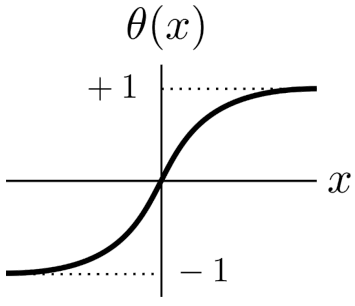
$$\theta(x) = \text{sign}(x)$$

Neural Networks



Activation Function

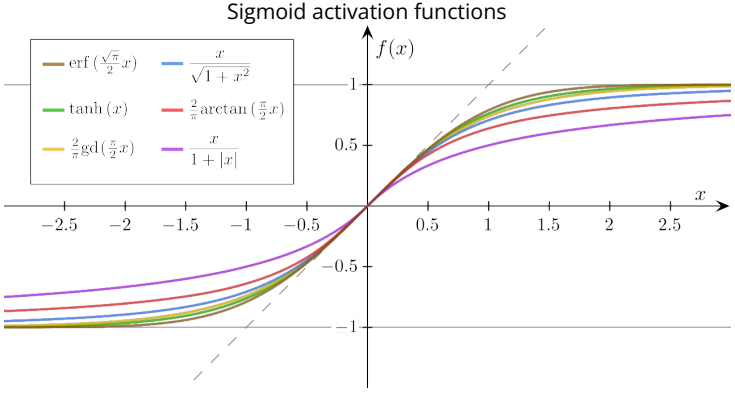
(differentiable!!!)



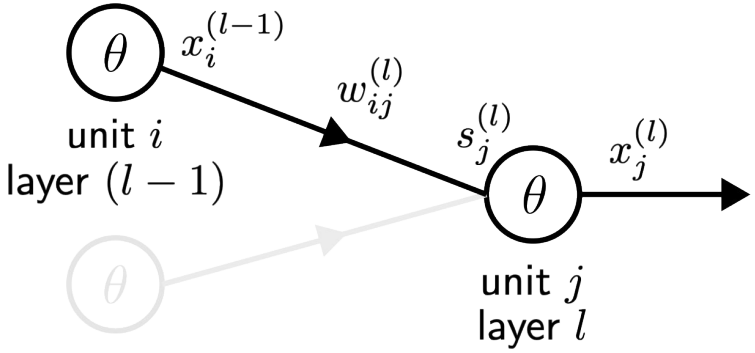
$$\theta(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Layers dimensions: $\mathbf{d} = [d, 3, 2, 1]$

$$\mathbf{d} = \left[d^{(0)}, d^{(1)}, \cdots, d^{(L)}\right]$$



Neural Networks

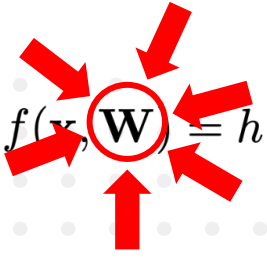


$$w_{ij}^{(l)} \quad \begin{cases} 1 \le l \le L & \text{layers} \\ 0 \le i \le d^{(l-1)} & \text{inputs} \\ 1 \le j \le d^{(l)} & \text{outputs} \end{cases}$$

Signal	Notation	Dimensions
Signal in	$\mathbf{s}^{(l)}$	$d^{(l)}$
Output	$\mathbf{x}^{(l)}$	$d^{(l)}+1$
Weights in	$\mathrm{W}^{(l)}$	$\left(d^{(l-1)}+1\right) \times d^{(l)}$

$$\mathbf{s}^{(l)} = \left(\mathrm{W}^{(l)}\right)^{\top} \mathbf{x}^{(l-1)} \qquad x_j^{(l)} = \theta\left(s_j^{(l)}\right) = \theta\left(\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}\right) \qquad \mathbf{x}^{(l)} = \left[1, \theta\left(\mathbf{s}^{(l)}\right)\right]^{\top}$$

$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{\mathrm{W}^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\theta} \mathbf{x}^{(1)} \xrightarrow{\mathrm{W}^{(2)}} \mathbf{s}^{(2)} \xrightarrow{\theta} \mathbf{x}^{(2)} \cdots \longrightarrow \mathbf{s}^{(L)} \xrightarrow{\theta} \mathbf{x}^{(L)} = h(\mathbf{x})$$



$$f(\mathbf{x}, \mathbf{W}) = h(\mathbf{x}) = \sigma\left(\left(W^{(L)}\right)^{\top} \sigma\left(\left(W^{(L-1)}\right)^{\top} \sigma\left(\cdots \sigma\left(\left(W^{(1)}\right)^{\top} \mathbf{x}^{(0)}\right)\right)\right)\right)$$

Universal approximation theorems

A feed-forward network with a single intermediate layer is sufficient to approximate, with arbitrary accuracy, any function with a finite number of discontinuities, provided that the activation functions of the hidden neurons are nonlinear.

Theorem 1. *Let σ be any continuous discriminatory function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(y_j^T x + \theta_j) \qquad (2)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\varepsilon > 0$, there is a sum, $G(x)$, of the above form, for which

$$|G(x) - f(x)| < \varepsilon \qquad \text{for all} \quad x \in I_n.$$

Cybenko, George. "Approximation by superpositions of a sigmoidal function." Mathematics of control, signals and systems 2.4 (1989): 303-314.
Hornik, Kurt, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." Neural networks 2.5 (1989): 359-366.
Hornik, Kurt. "Approximation capabilities of multilayer feedforward networks." Neural networks 4.2 (1991): 251-257.

Universal approximation theorems

5. Summary

We have demonstrated that finite superpositions of a fixed, univariate function that is *discriminatory* can uniformly approximate any continuous function of n real variables with support in the unit hypercube. Continuous sigmoidal functions of the type commonly used in real-valued neural network theory are discriminatory.

This combination of results demonstrates that any continuous function can be uniformly approximated by a continuous neural network having only one internal, hidden layer and with an arbitrary continuous sigmoidal nonlinearity (Theorem 2).

A visual proof that neural nets can compute any function
http://neuralnetworksanddeeplearning.com/chap4.html

Cybenko, George. "Approximation by superpositions of a sigmoidal function." Mathematics of control, signals and systems 2.4 (1989): 303-314.

Universal approximation theorems

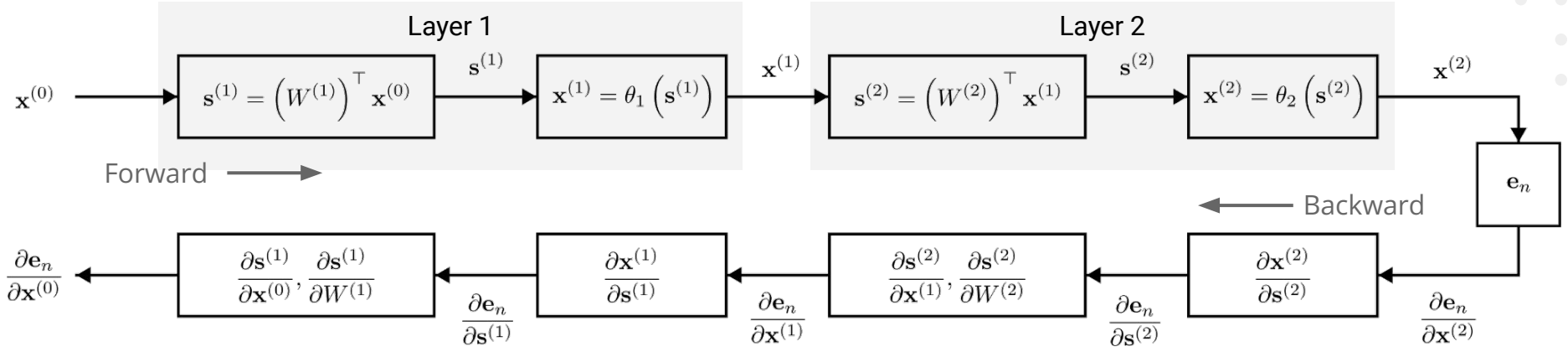
- NN can represent a wide variety of functions
- Do not provide how to construct the NN.
- Do not apply for (too) small NN[1]
  - NN will not learn.
- *Bounded depth* and *bounded width* cases.

- NN width is an important hyperparameter.
- Activation function is important too (ReLU or sigmoid)
- It is known that some activation functions do not work (neither linear nor polynomial).

1. Compared with the dimensions of the input data.

Neural Networks: Training (parameters' optimization)

Simplify with two layers NN.



Minimize the error with respect to the parameters:

$$\frac{\partial \mathrm{e}(\mathbf{w})}{\partial w_{ij}^{(l)}} = \frac{\partial \mathrm{e}(\mathbf{w})}{\partial s_j^{(l)}} \frac{\partial s_j^{(l)}}{\partial w_{ij}^{(l)}}$$

$$\frac{\partial e(\mathbf{w})}{\partial s_j^{(l)}} = \delta_j^{(l)}$$

$$x_i^{(l-1)}$$

Unknown. Let's call it delta!

There's one *easy* delta to compute for $(j = 1, l = L) \Rightarrow \delta_1^{(L)}$

Repeat (mini-batch, epochs, ...) until...

Neural Networks: Backpropagation

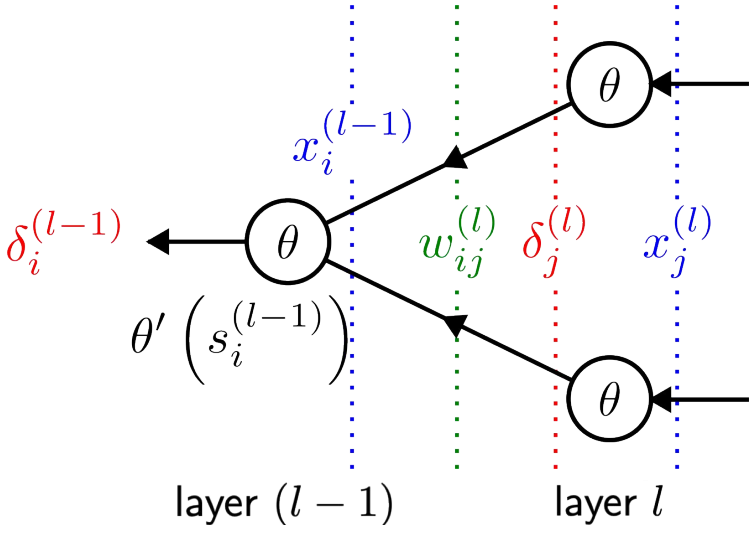
$$
\begin{aligned}
{\delta_1}^{(L)} &= \frac{\partial \mathbf{e}_n(\mathbf{w})}{\partial {s_1}^{(L)}} \\
&= \frac{\partial}{\partial {s_1}^{(L)}} \left({x_1}^{(L)} - y_n \right)^2 \quad \text{for } \ell^2 \\
&= 2 \left({x_1}^{(L)} - y_n \right) \frac{\partial {x_1}^{(L)}}{\partial {s_1}^{(L)}} \\
&= 2 \left({x_1}^{(L)} - y_n \right) \theta' \left({s_1}^{(L)} \right) \\
&= 2 \left(x_1^{(L)} - y_n \right) \left(1 - \theta^2 \left(s_1^{(L)} \right) \right)
\end{aligned}
$$

Since

$$x_1^{(L)} = \theta \left(s_1^{(L)} \right)$$

$$\theta'(s) = 1 - \theta^2(s) \quad \text{for } \tanh$$

Neural Networks: Backpropagation



$$\delta_i^{(l-1)} = \frac{\partial \mathrm{e}(\mathbf{w})}{\partial s_i^{(l-1)}}$$

$$= \sum_{j=1}^{d^{(l)}} \frac{\partial \mathrm{e}(\mathbf{w})}{\partial s_j^{(l)}} \frac{\partial s_j^{(l)}}{\partial x_i^{(l-1)}} \frac{\partial x_i^{(l-1)}}{\partial s_i^{(l-1)}}$$

$$= \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \; w_{ij}^{(l)} \; \theta'\left(s_i^{(l-1)}\right)$$

$$= \theta'\left(s_i^{(l-1)}\right) \sum_{j=1}^{d^{(l)}} \delta_j^{(l)} \; w_{ij}^{(l)}$$

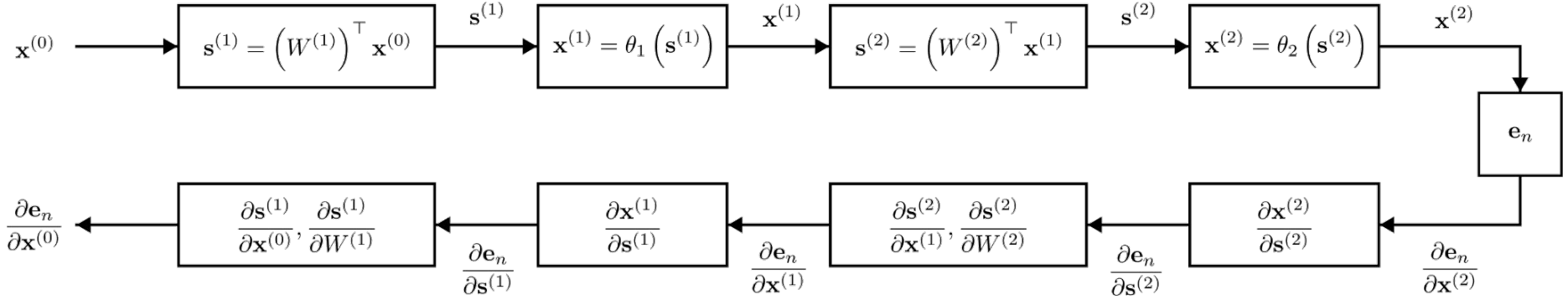
$$\delta_i^{(l-1)} = \left(1 - \left(x_i^{(l-1)}\right)^2\right) \sum_{j=1}^{d^{(l)}} w_{ij}^{(l)} \delta_j^{(l)}$$

$$\boldsymbol{\delta}^{(1)} \longleftarrow \boldsymbol{\delta}^{(2)} \ldots \longleftarrow \boldsymbol{\delta}^{(L-1)} \longleftarrow \boldsymbol{\delta}^{(L)}$$

Neural Networks: Backpropagation

Forward ⟶

$$\mathbf{x} = \mathbf{x}^{(0)} \xrightarrow{\mathbf{W}^{(1)}} \mathbf{s}^{(1)} \xrightarrow{\theta} \mathbf{x}^{(1)} \xrightarrow{\mathbf{W}^{(2)}} \mathbf{s}^{(2)} \xrightarrow{\theta} \mathbf{x}^{(2)} \cdots \longrightarrow \mathbf{s}^{(L)} \xrightarrow{\theta} \mathbf{x}^{(L)} = h(\mathbf{x})$$



$$\boldsymbol{\delta}^{(1)} \longleftarrow \boldsymbol{\delta}^{(2)} \cdots \longleftarrow \boldsymbol{\delta}^{(L-1)} \longleftarrow \boldsymbol{\delta}^{(L)}$$ ⟵ Backward

Neural Networks: Backpropagation

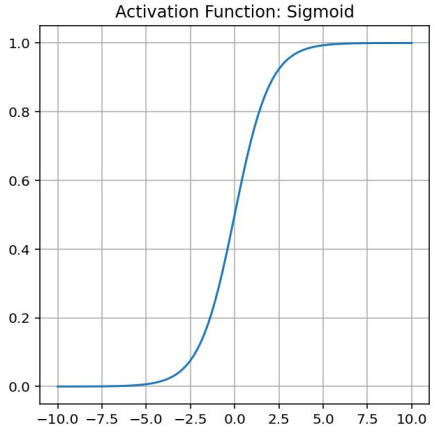
1. Initialize all weights $w_{ij}^{(l)}$ **at random**
2. For $t = 0, 1, 2, \ldots$ do
3. Pick $n \in \{1, 2, \cdots, N\}$
4. **Forward**: compute all $x_i^{(l)}$
5. **Backward**: compute all $\delta_j^{(l)}$
6. Update the weights $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta\, x_i^{(l-1)} \delta_j^{(l)}$
7. Iterate to the next step **until stop**
8. Return final weights $w_{ij}^{(l)}$

It is an efficient application of the Leibniz chain rule (1673) to such networks. It is also known as the reverse mode of automatic differentiation or reverse accumulation, due to Seppo Linnainmaa (1970). The term "back-propagating error correction" was introduced in 1962 by Frank Rosenblatt, but he did not know how to implement this, even though Henry J. Kelley had a continuous precursor of backpropagation already in 1960 in the context of control theory.

Backpropagation - Wikipedia

Activation functions

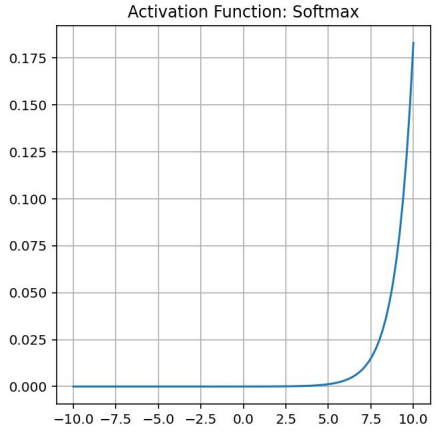
$$S(x) = \frac{1}{1 + e^{-x}}$$



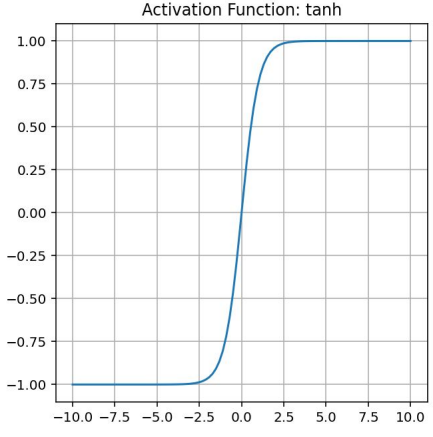
$$\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

$$i = 1, \ldots, K$$

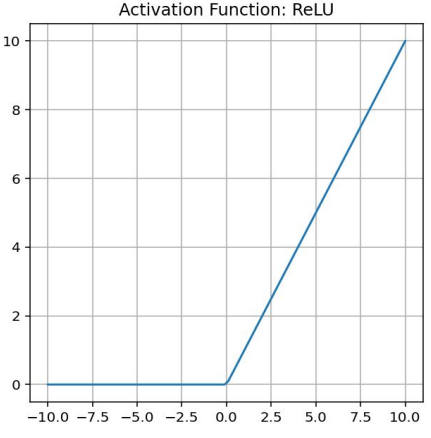
$$\mathbf{x} = (x_1, \ldots, x_K) \in \mathbb{R}^K$$



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

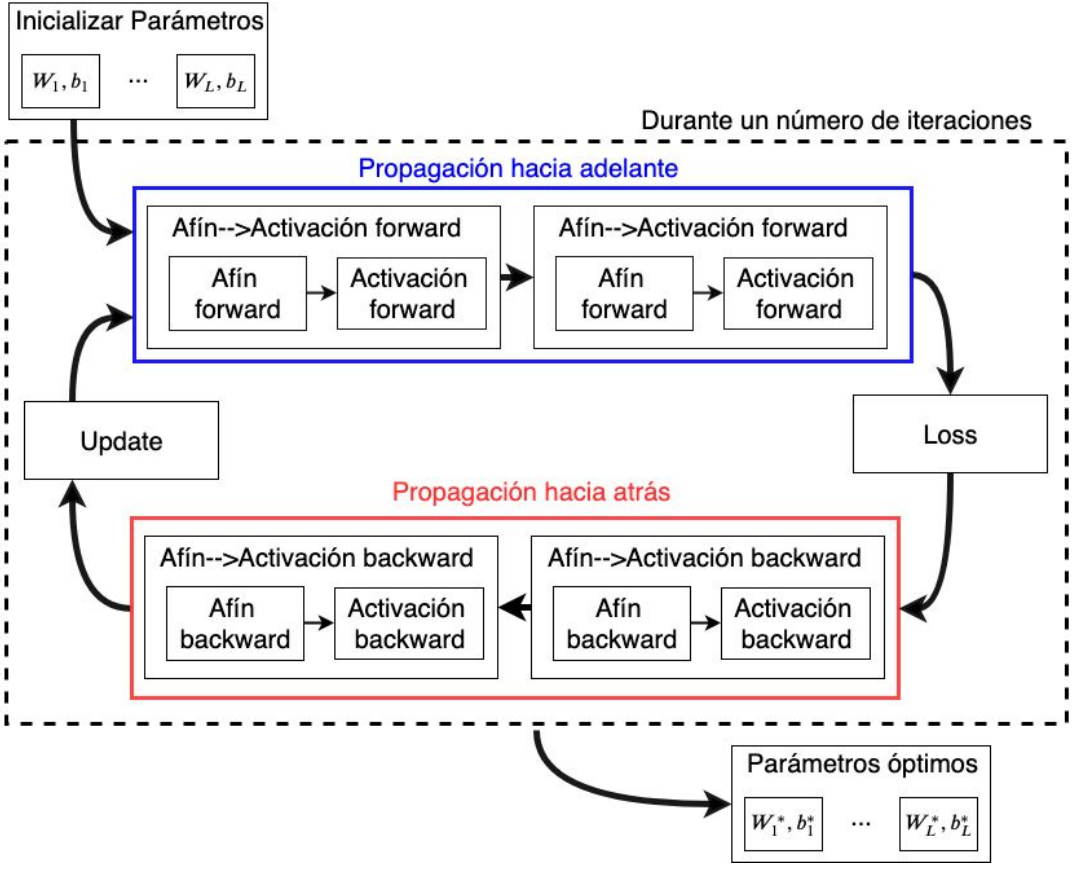


$$\mathrm{ReLU}(x) = \max\{0, x\}$$

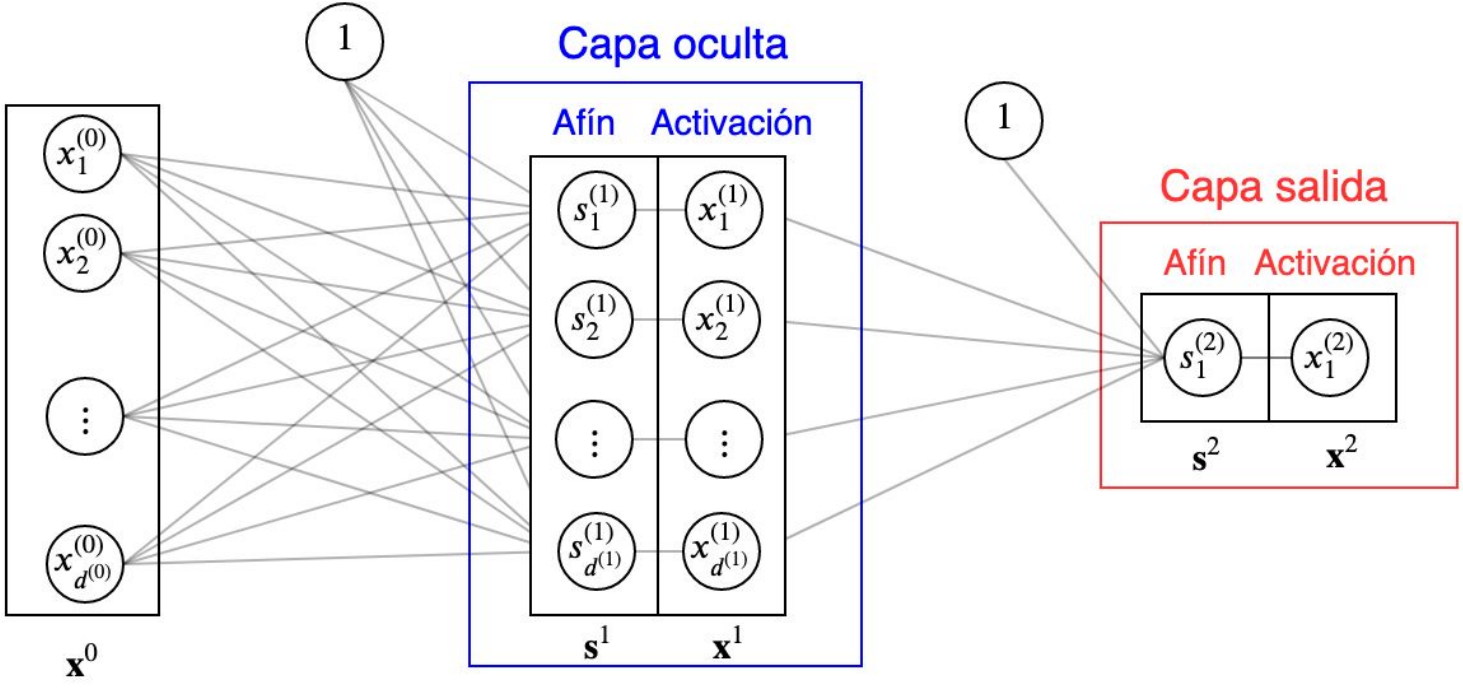


- Non-linear.
- Differentiable.
- Discriminant.

Neural Networks: 2 layers



Neural Networks: 2 layers



$$W^{(1)} \in \mathbb{R}^{\left(d^{(0)}+1\right)\times d^{(1)}}$$

$$W^{(2)} \in \mathbb{R}^{\left(d^{(1)}+1\right)\times 1}$$