

Idea de las Redes Neuronales Artificiales

**Un Clip del programa
The Today Show, 1994**

¿Qué es Deep Learning?

¿Qué es Deep Learning?



¿Qué es Deep Learning?



SanDisk Ultra 256GB MicroSDXC UHS-I Card with Adapter (SDSQUNI-256G-GN6MA).

by SanDisk

\$149⁹⁹ \$199.99 Prime

Wednesday, Mar 22

Shipping on eligible



21,224

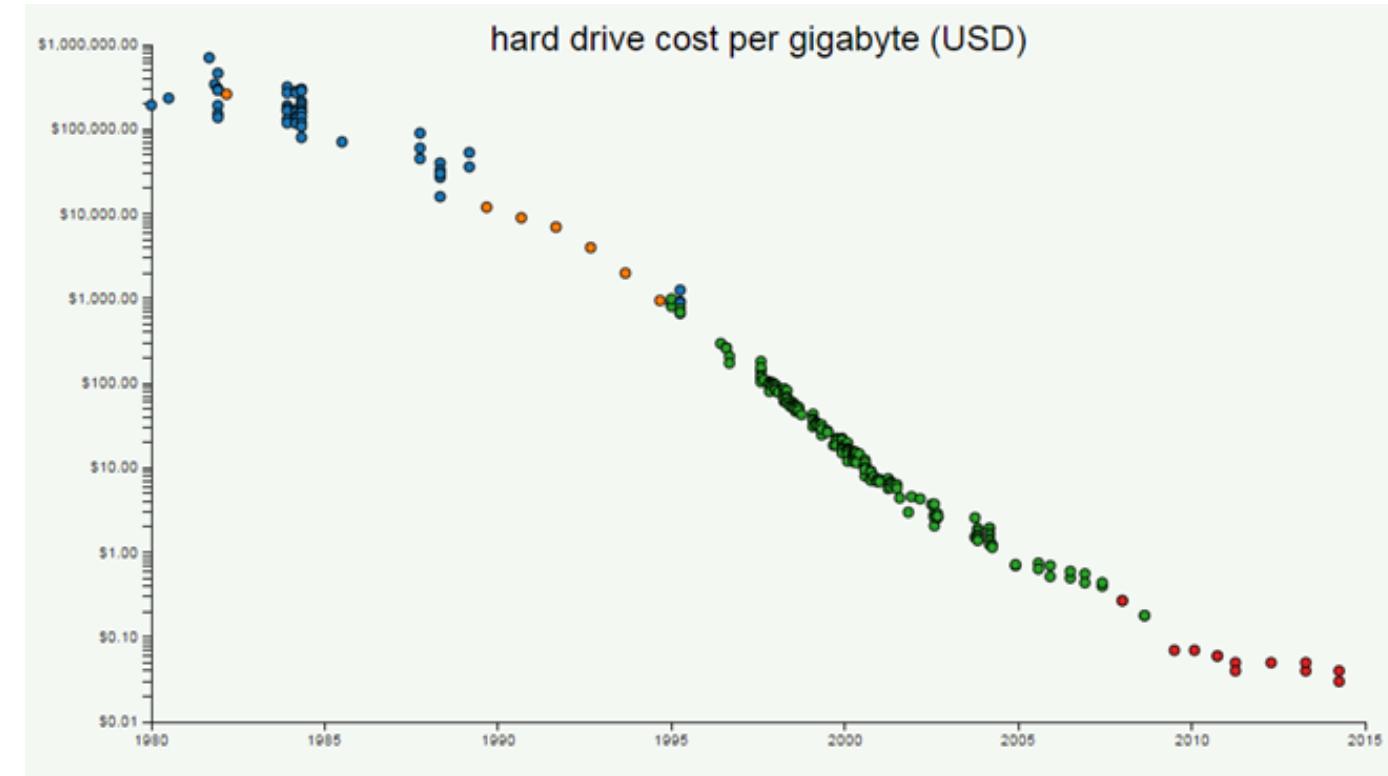
Product Features
256GB capacity
breakthrough

25,600x

Choices
(10 new offers)

\$147.99

¿Qué es Deep Learning?



Source: mkomo.com

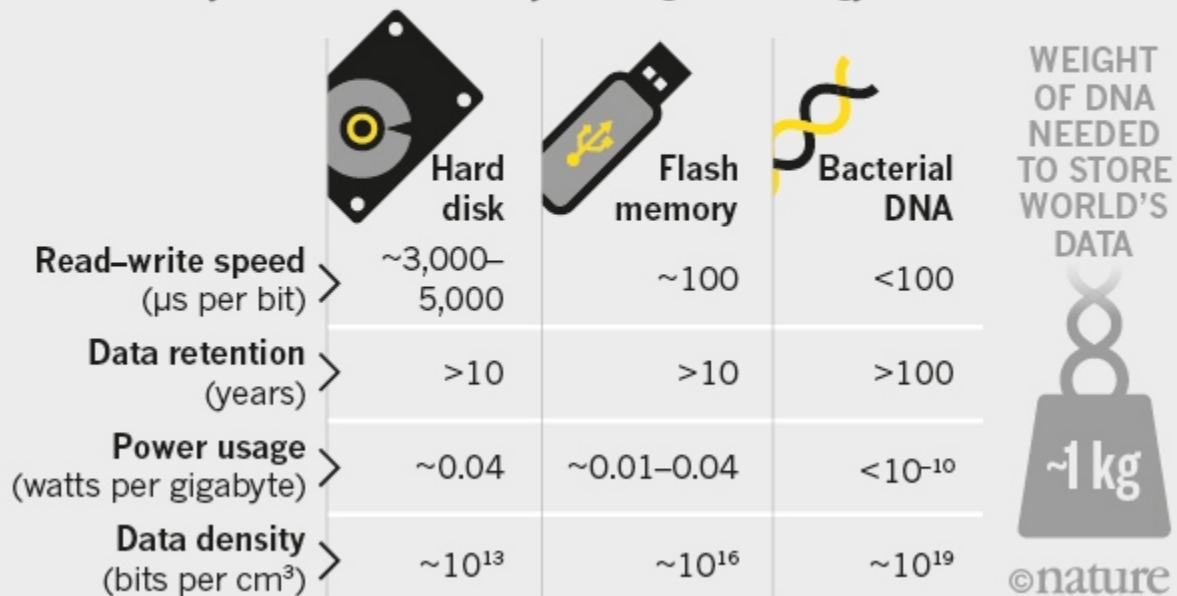
¿Qué es Deep Learning?



¿Qué es Deep Learning?

STORAGE LIMITS

Estimates based on bacterial genetics suggest that digital DNA could one day rival or exceed today's storage technology.



Source: nature.com

¿Qué es Deep Learning?

1 The accelerating pace of change ...



2 ... and exponential growth in computing power ...

Computer technology, shown here climbing dramatically by powers of 10, is now progressing more each hour than it did in its entire first 90 years

COMPUTER RANKINGS

By calculations per second per \$1,000



Analytical engine
Never fully built, Charles Babbage's invention was designed to solve computational and logical problems



Colossus

The electronic computer, with 1,500 vacuum tubes, helped the British crack German codes during WW II



UNIVAC I

The first commercially marketed computer, used to tabulate the U.S. Census, occupied 943 cu. ft.



Apple II

At a price of \$1,298, the compact machine was one of the first massively popular personal computers



Power Mac G4

The first personal computer to deliver more than 1 billion floating-point operations per second

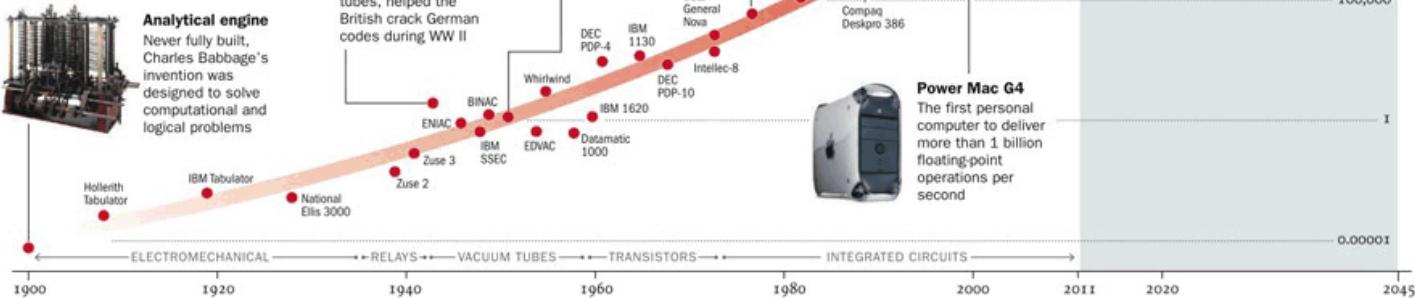
3 ... will lead to the Singularity

Surpasses brainpower equivalent to that of all human brains combined

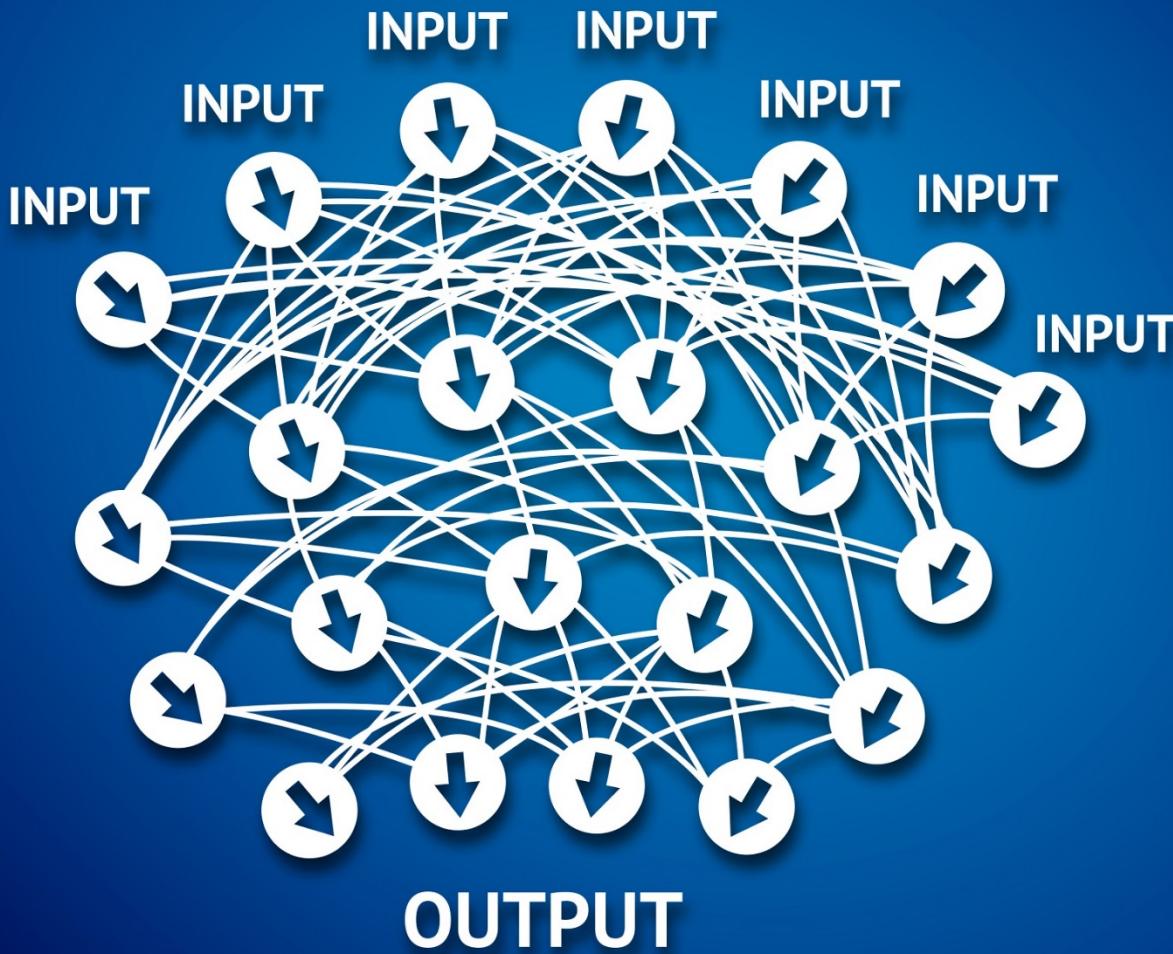
Surpasses brainpower of human in 2023



Surpasses brainpower of mouse in 2015



Source: Time Magazine

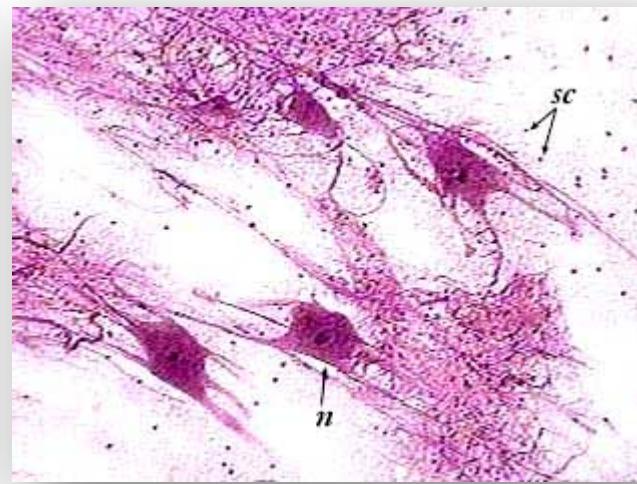
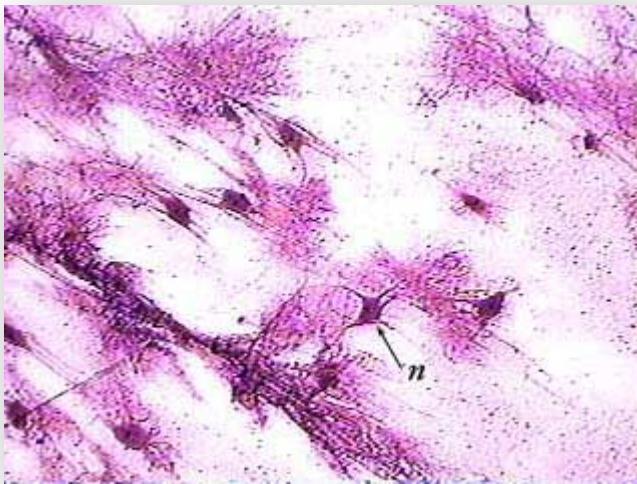


¿Qué es Deep Learning?

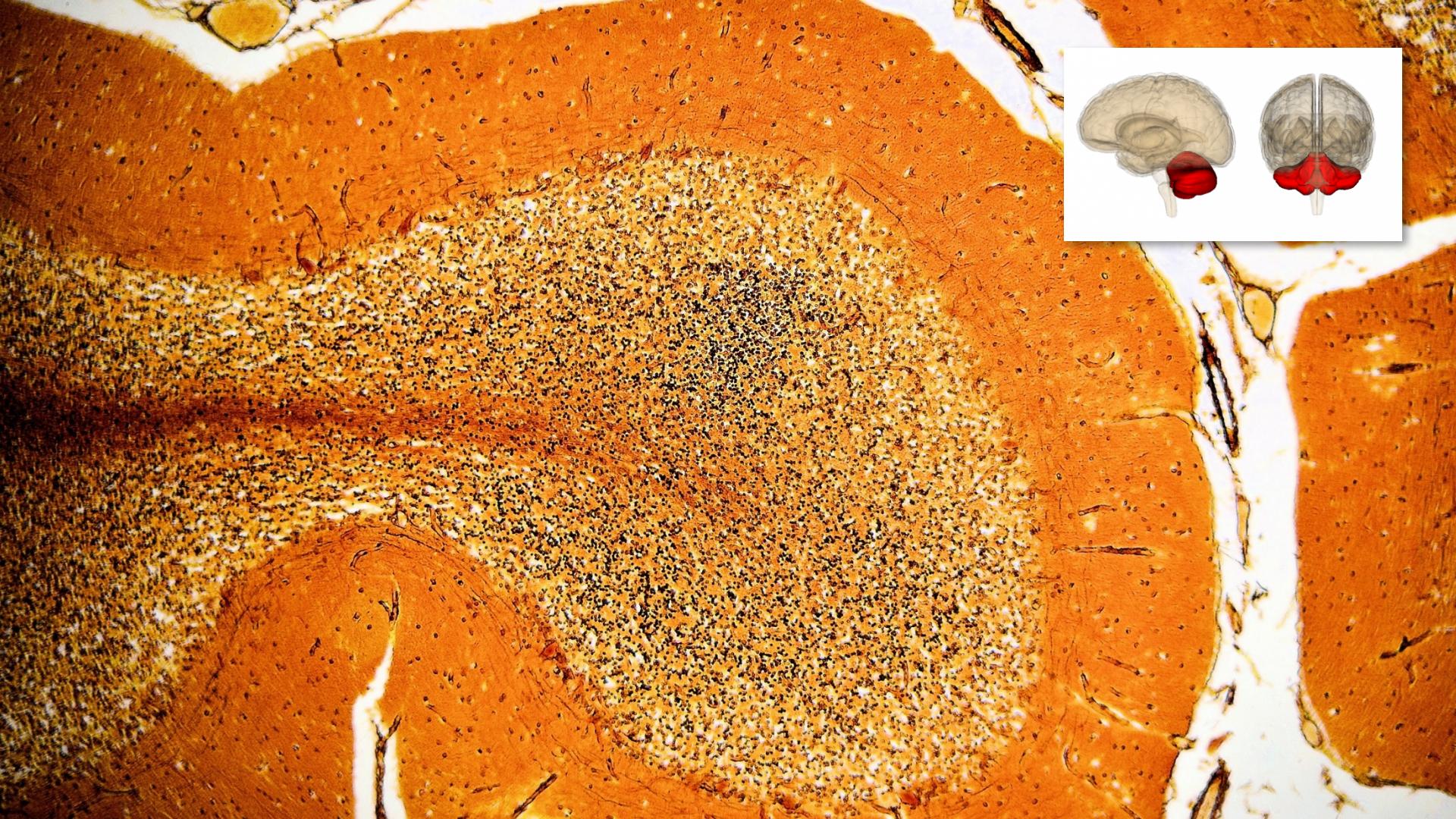
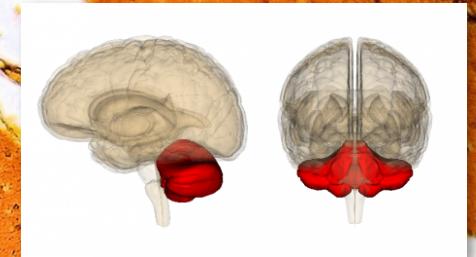


Geoffrey Hinton

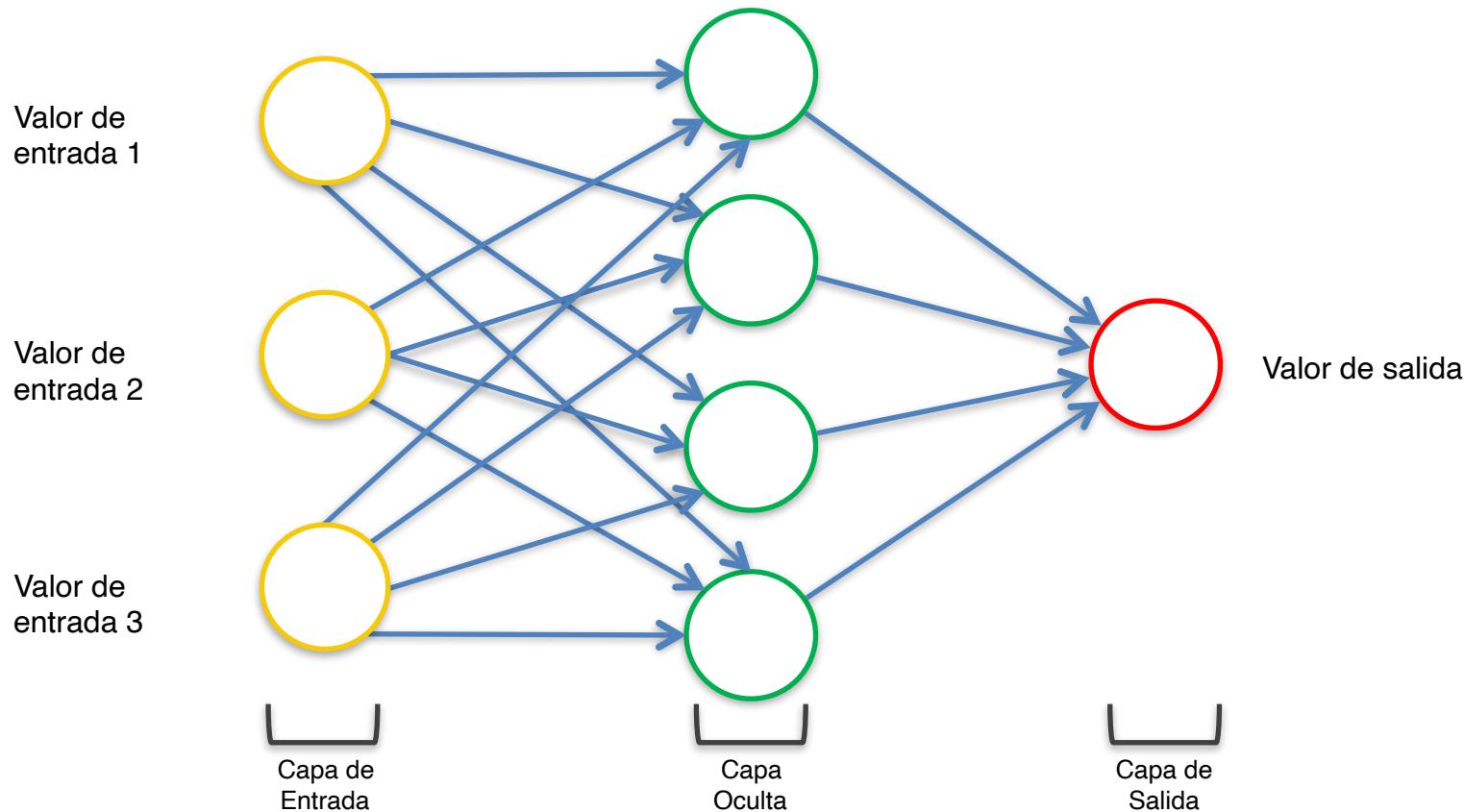
¿Qué es Deep Learning?



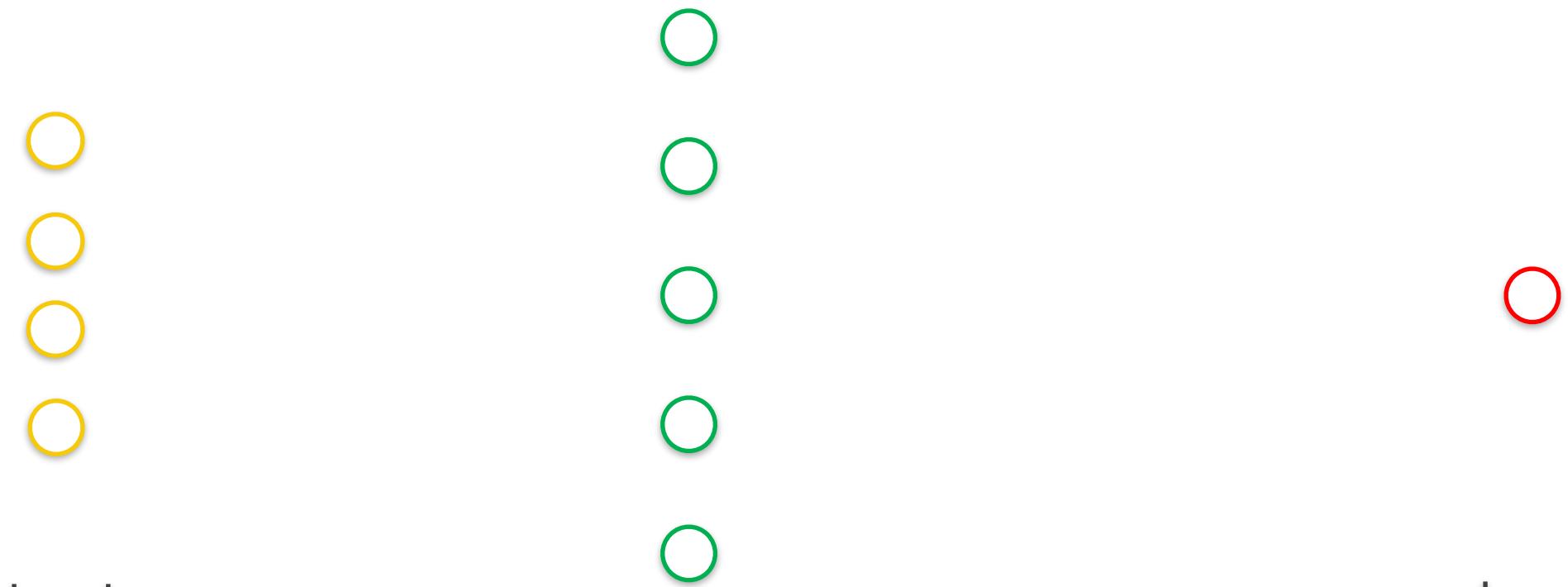
ImEdad Source: www.austincc.edu



¿Qué es Deep Learning?



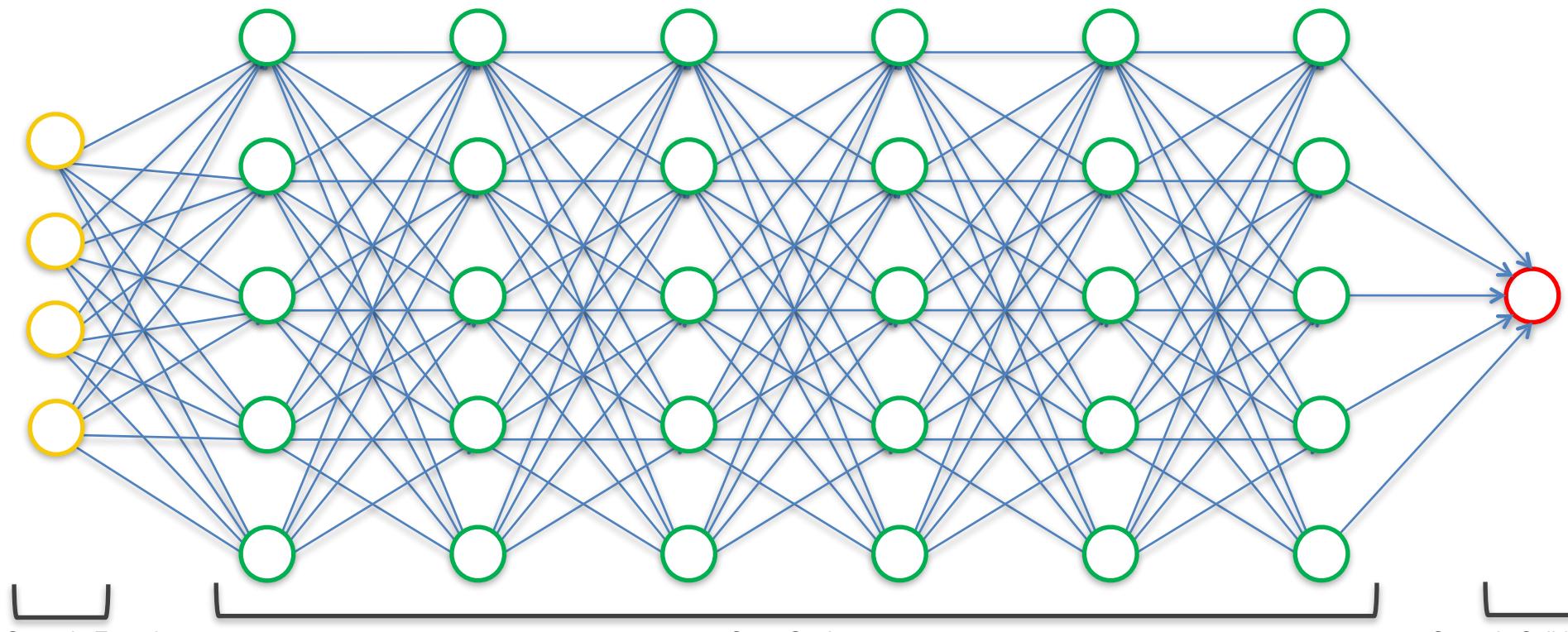
¿Qué es Deep Learning?



Capa de Entrada

Capa de Salida

¿Qué es Deep Learning?



Capa de Entrada

Capa Ocultas

Capa de Salida

Supervisado

vs

No Supervisado

Supervisado vs No Supervisado

Supervisado	Redes Neuronales Artificiales	Usadas para Regresión & Clasificación
	Redes Neuronales Convolucionales	Usadas en Visión por Computador
	Redes Neuronales Recurrentes	Usadas para Análisis de Series Temporales
No Supervisado	Mapas Autorganizado	Usadas para Selección de Factores
	Máquinas Deep Boltzmann	Usadas en Sistemas de Recomendación
	AutoEncoders	Usadas en Sistemas de Recomendación

Plan de Ataque

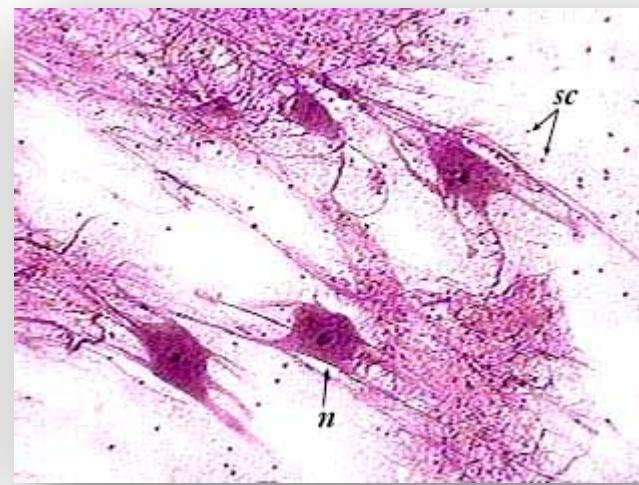
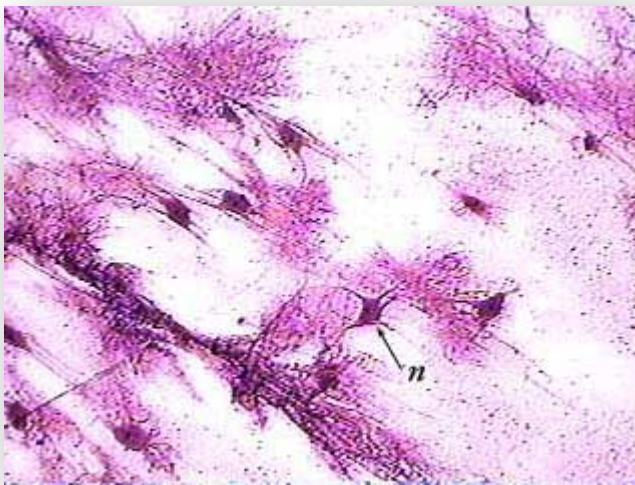
Plan de Ataque

Qué vamos a aprender en la sección

- La Neurona
- La Función de Activación
- ¿Cómo funcionan las Redes Neuronales? (ejemplo)
- ¿Cómo aprenden las Redes Neuronales?
- Gradiente Descente
- Gradiente Descente Estocástico
- Propagación hacia atrás

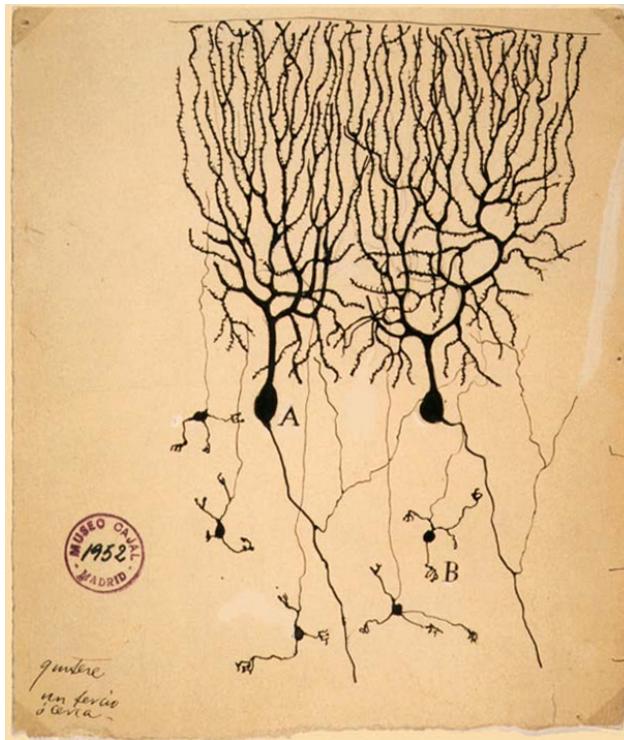
La Neurona

La Neurona



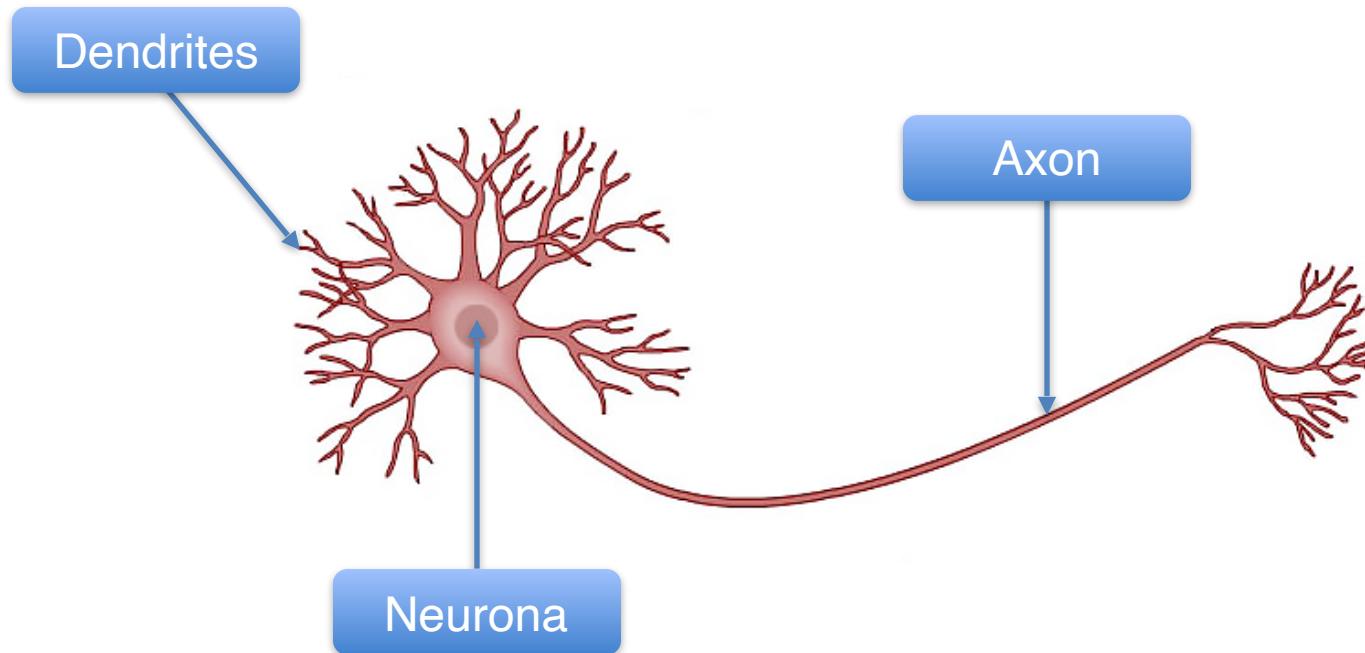
ImEdad Source: www.austincc.edu

La Neurona



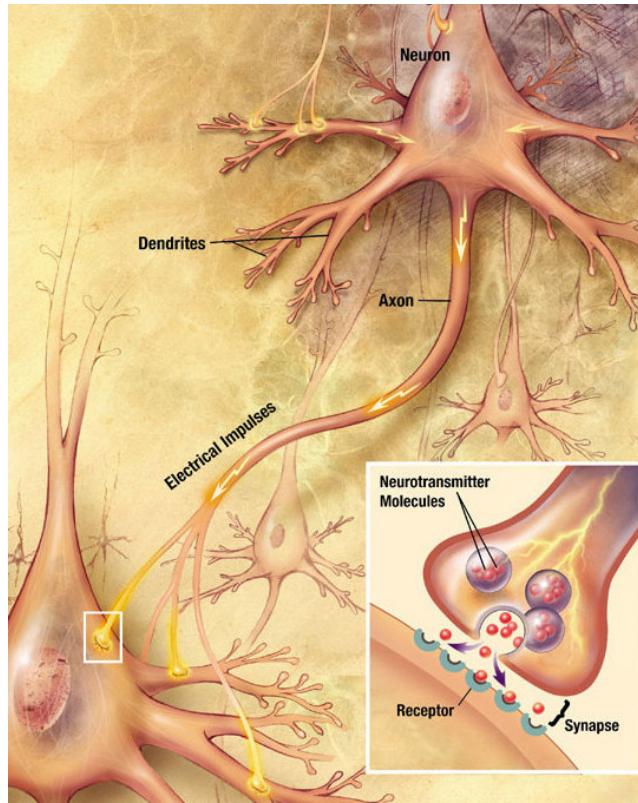
ImEdad Source: Wikipedia

La Neurona



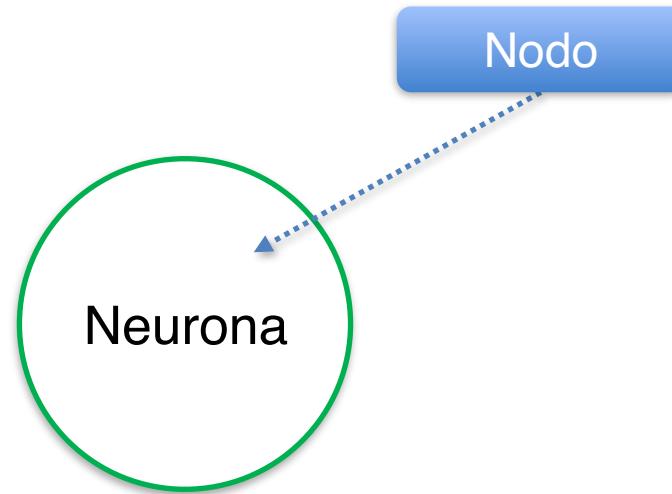
ImEdad Source: Wikipedia

La Neurona

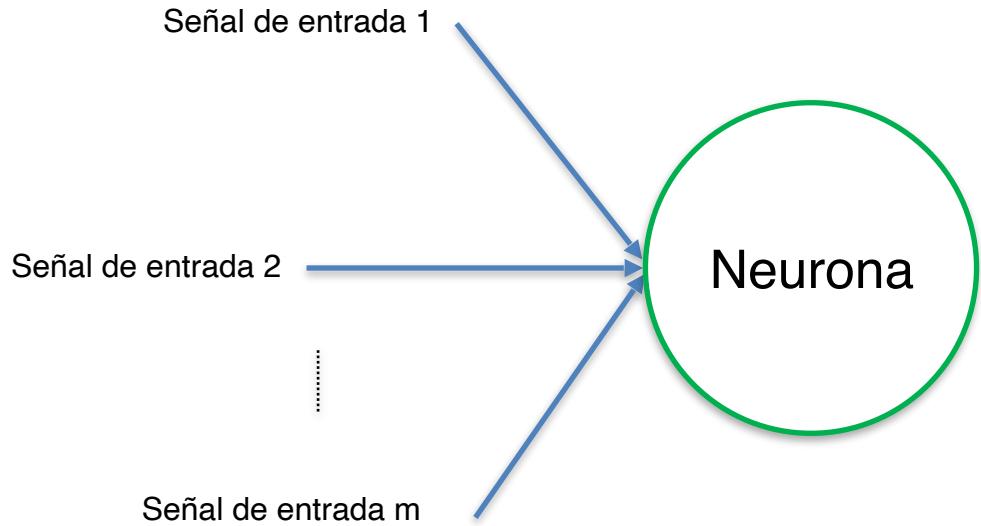


ImEdad Source: Wikipedia

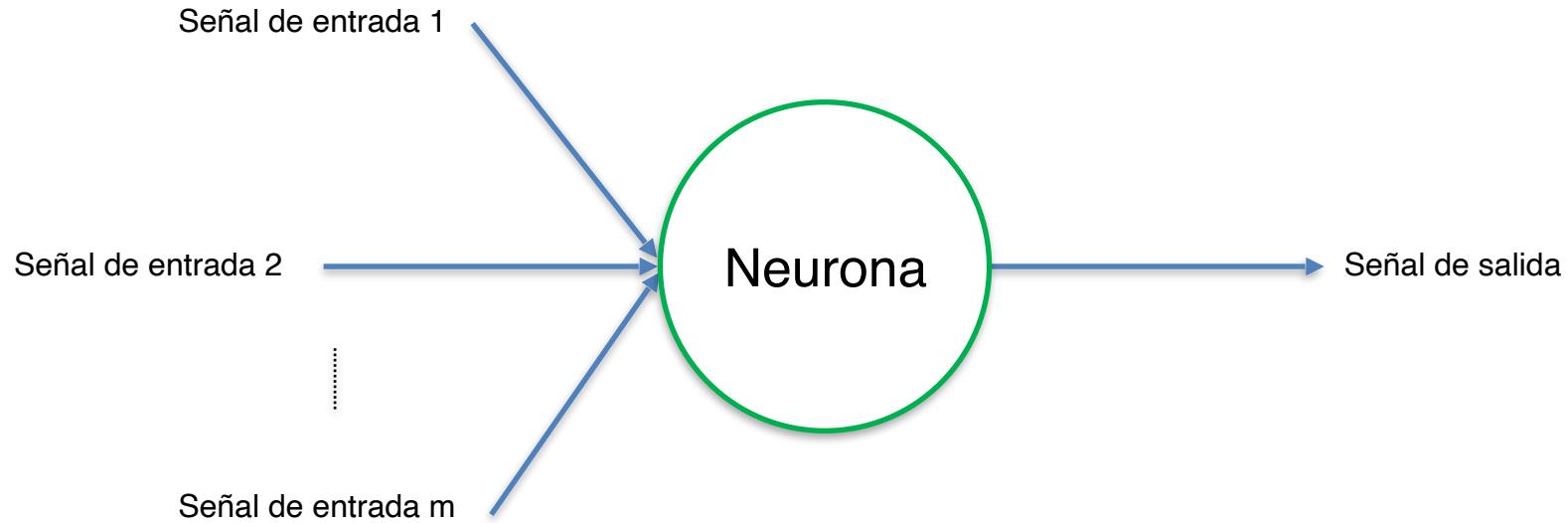
La Neurona



La Neurona



La Neurona



La Neurona

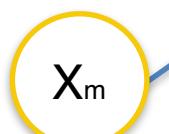
Valor de
entrada 1



Valor de
entrada 2



Valor de
entrada m

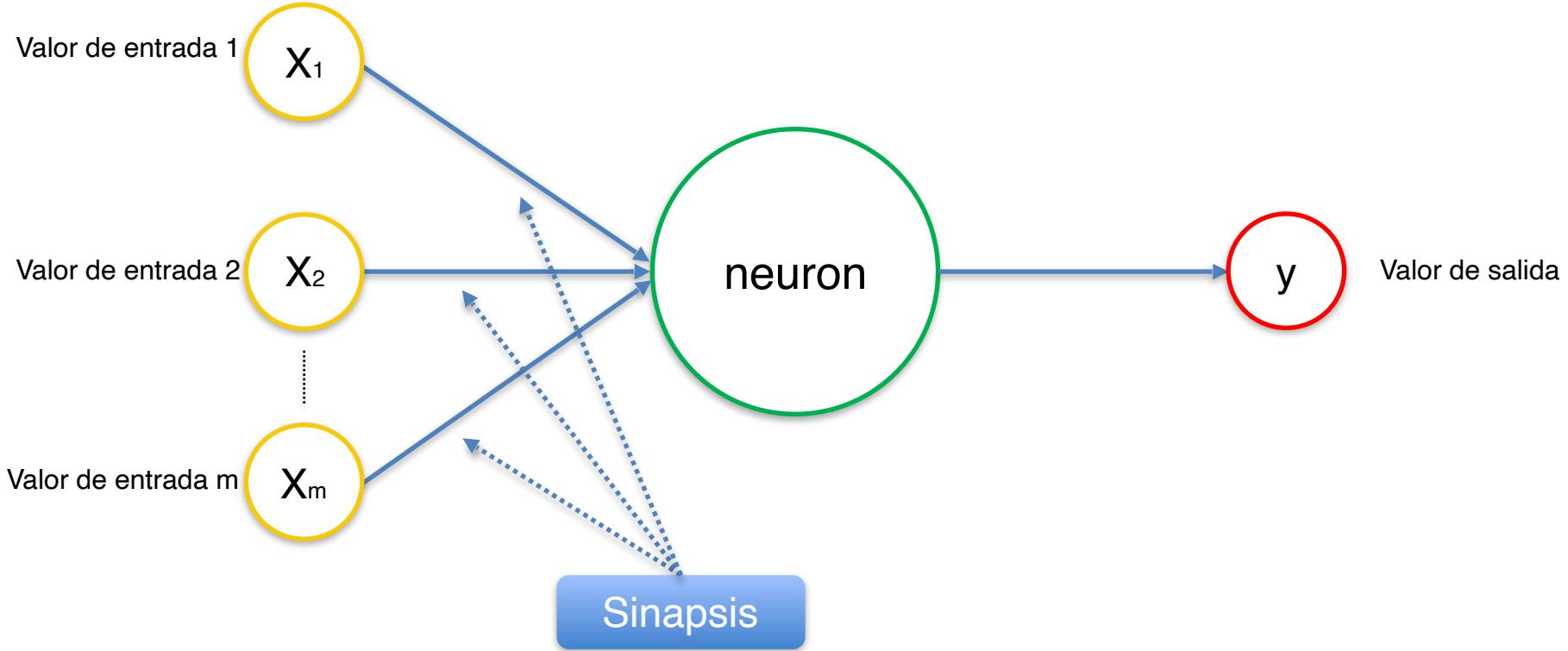


neuron

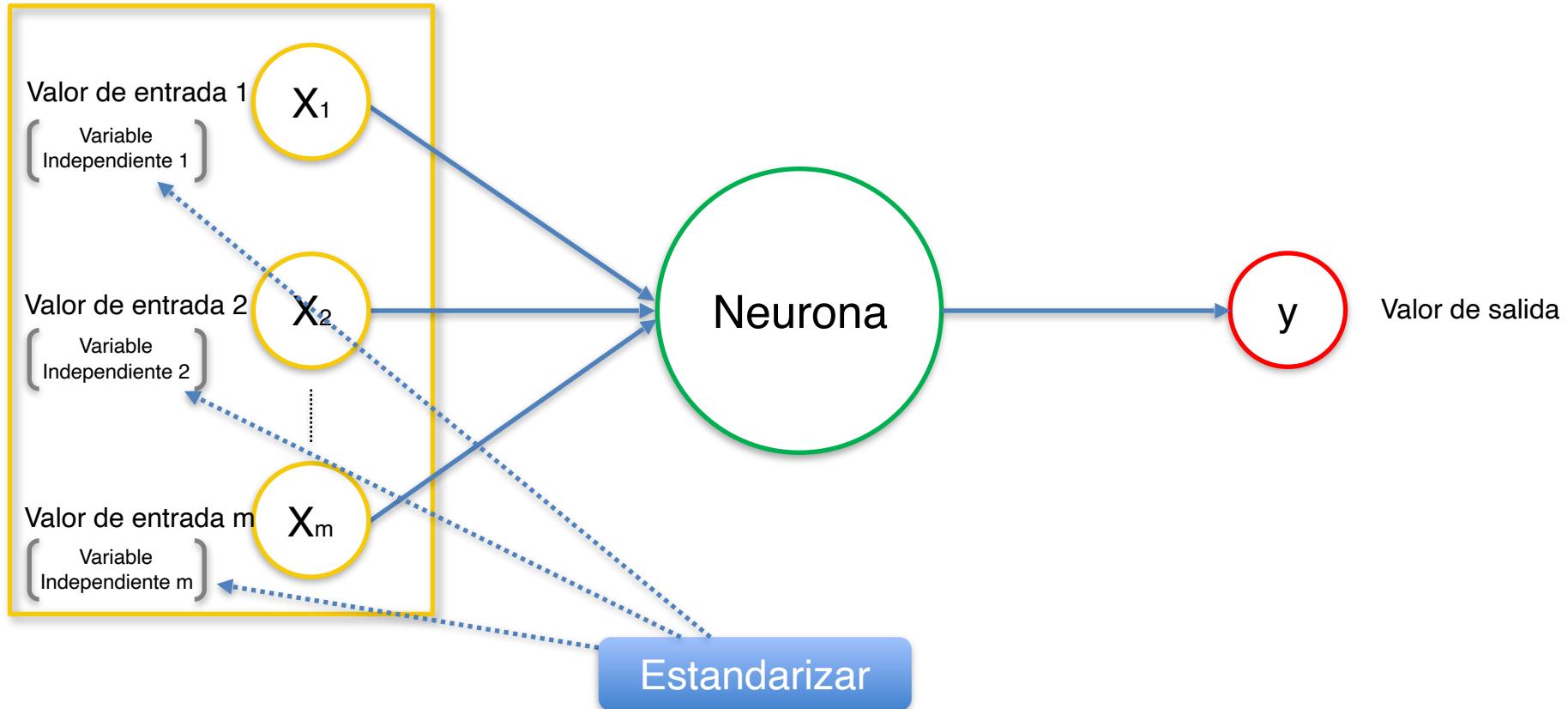
Señal de salida

Sinapsis

La Neurona



La Neurona



La Neurona

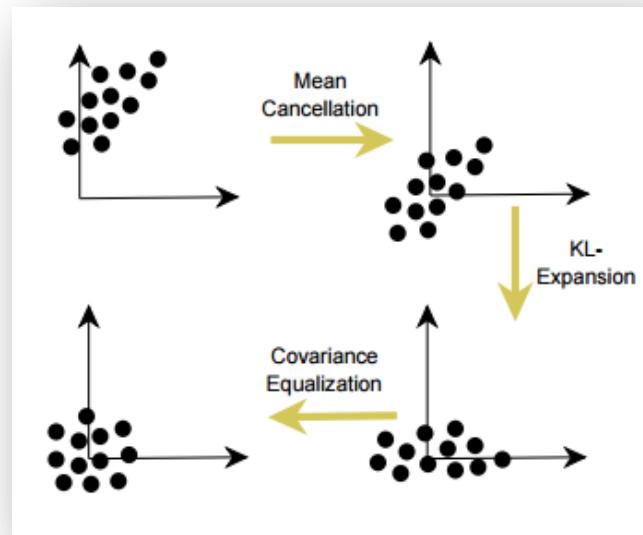
Lectura adicional

Efficient BackProp

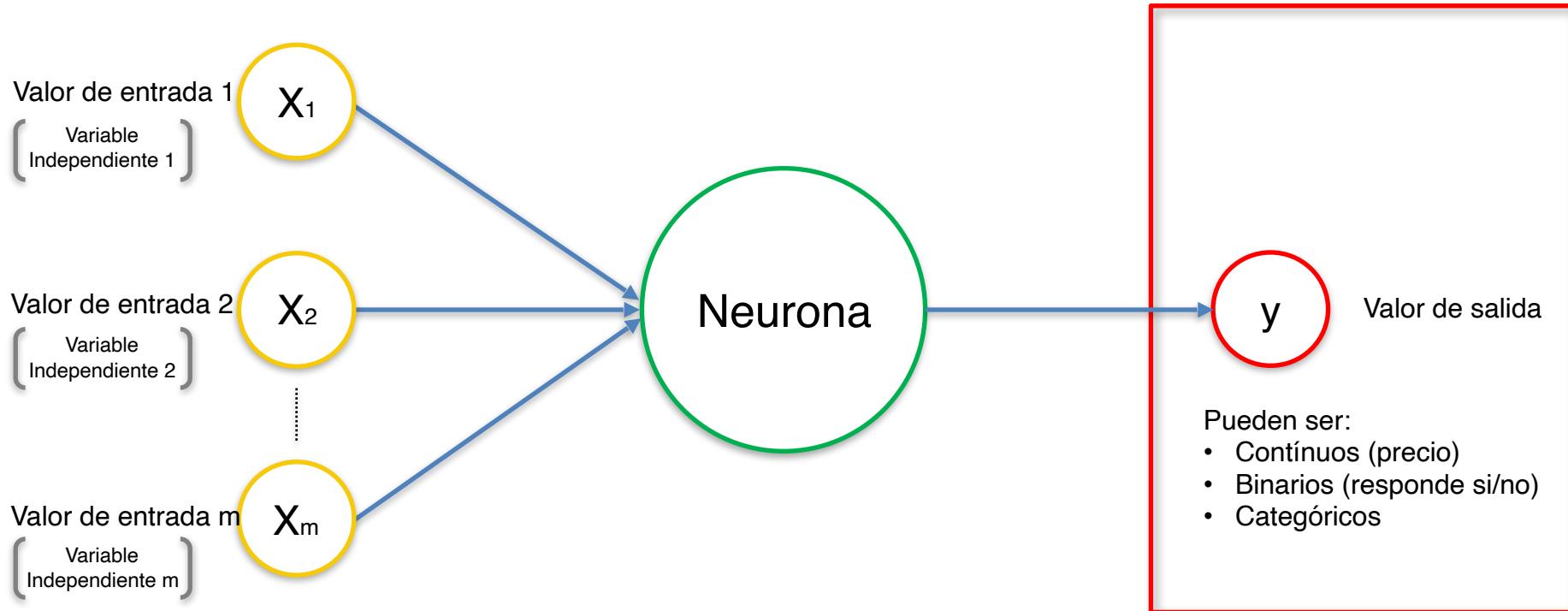
By Yann LeCun et al. (1998)

Link:

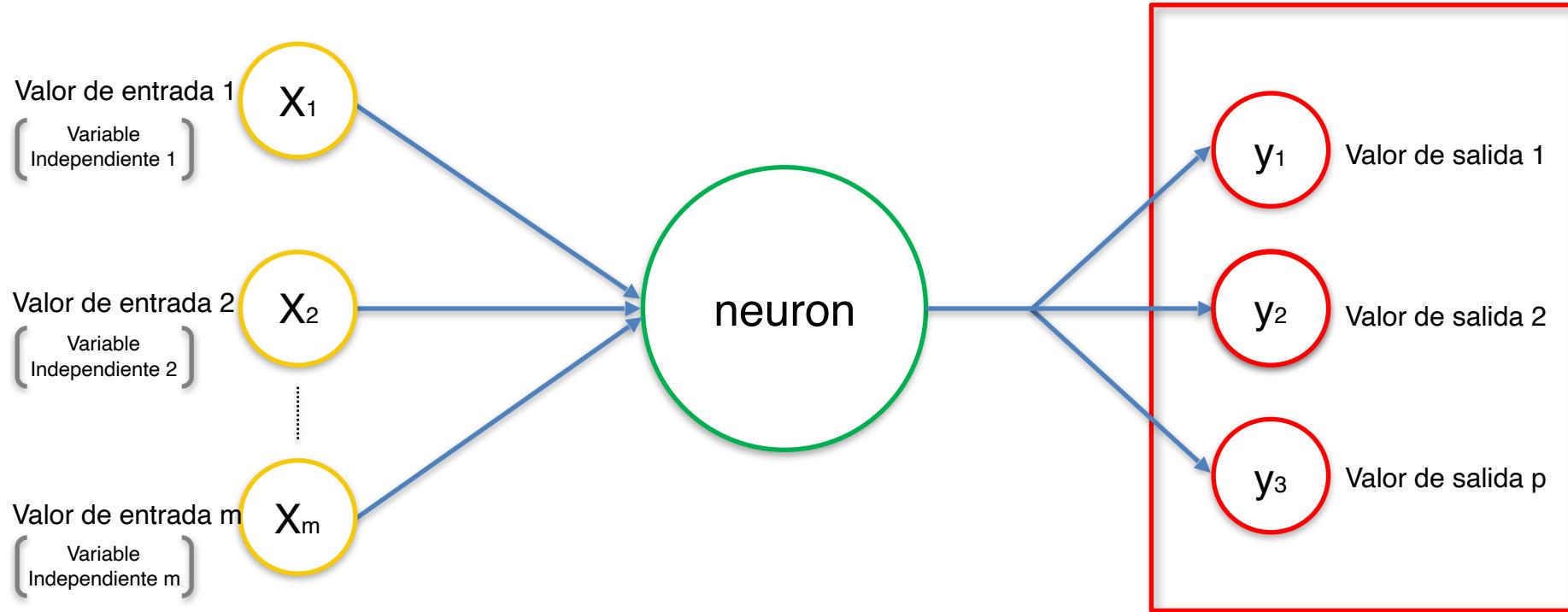
<http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf>



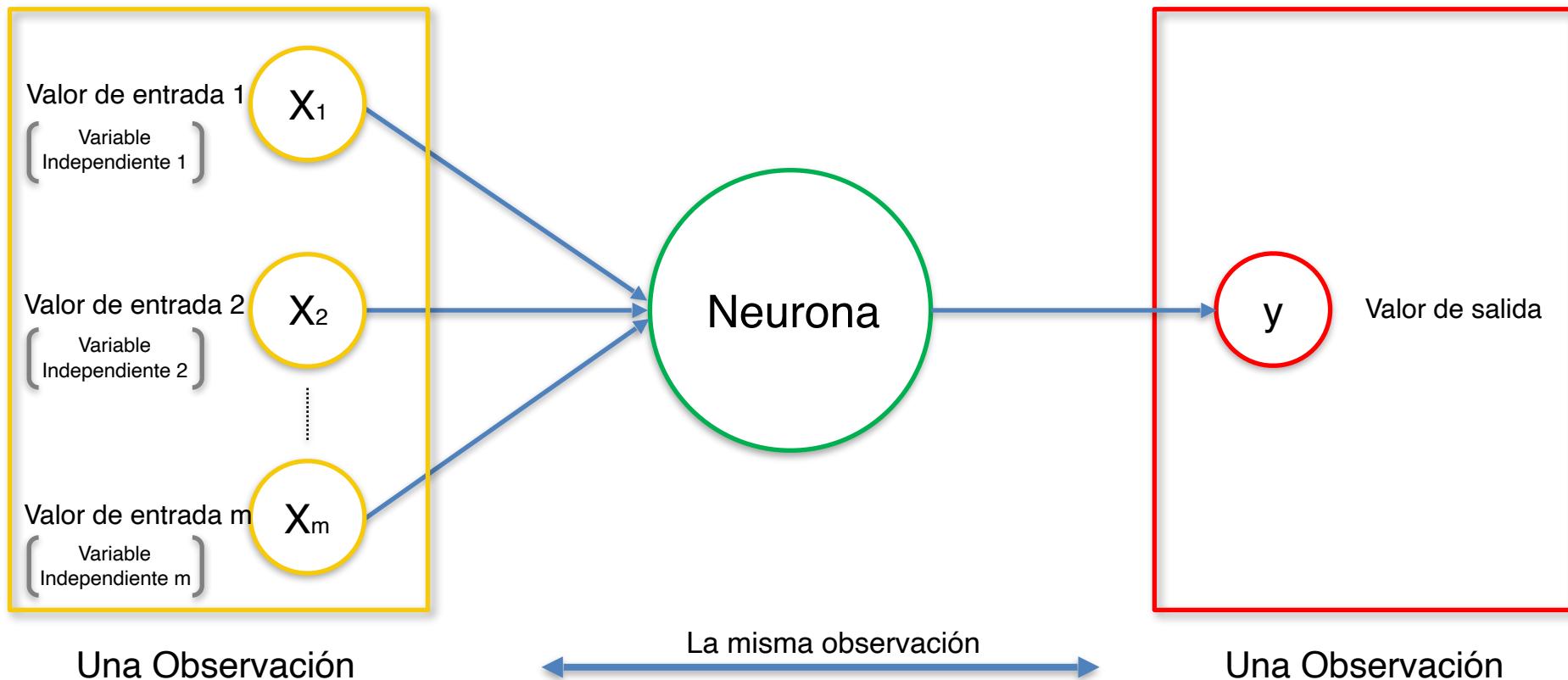
La Neurona



La Neurona



La Neurona

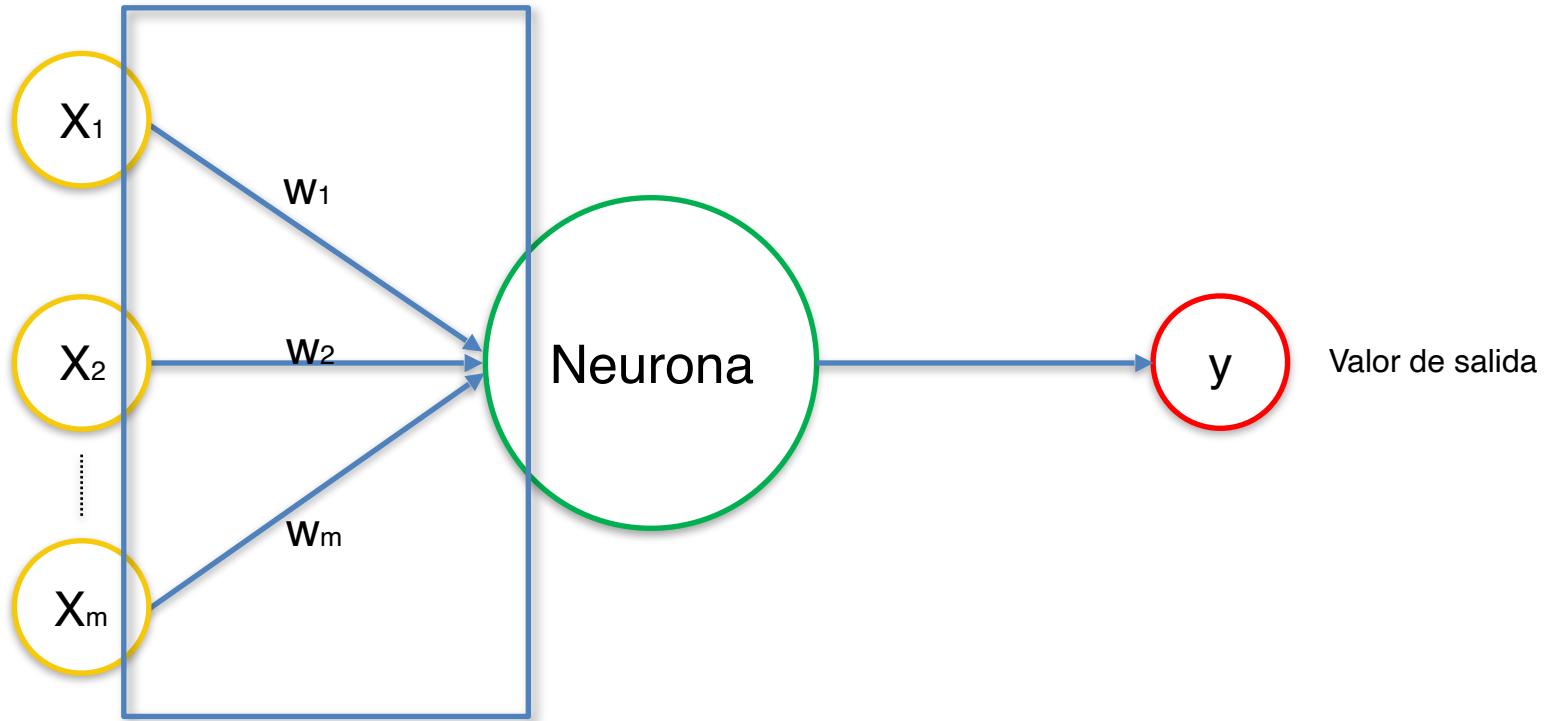


La Neurona

Valor de
entrada 1

Valor de
entrada 2

Valor de
entrada m



La Neurona

Valor de
entrada 1



w_1

Valor de
entrada 2

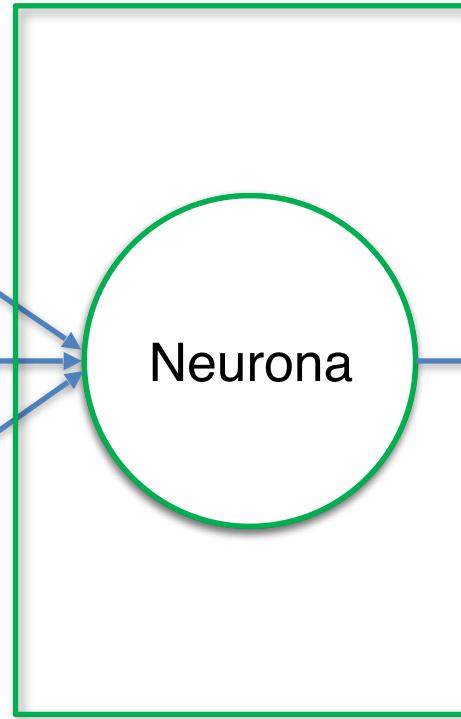


w_2

Valor de
entrada m



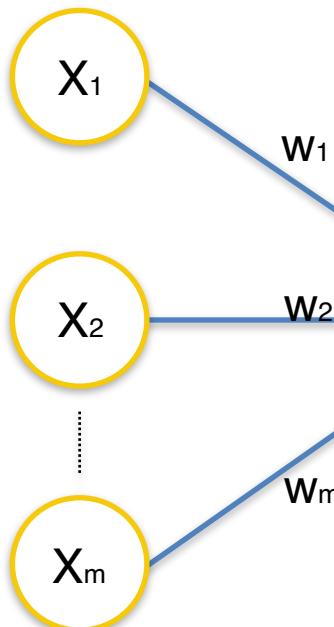
w_m



Valor de salida

La Neurona

Valor de
entrada 1

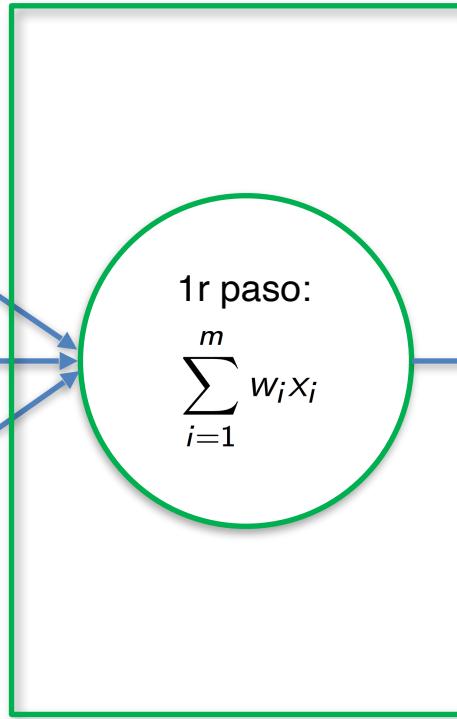


1r paso:

$$\sum_{i=1}^m w_i x_i$$

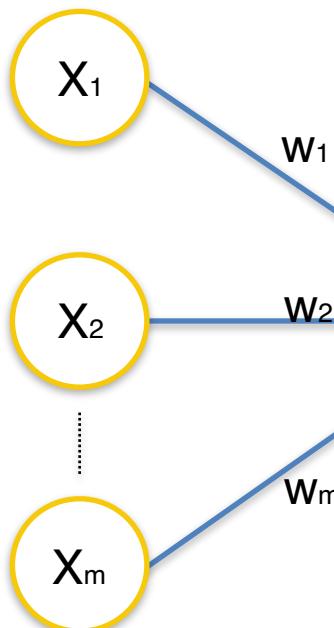
Valor de salida

y



La Neurona

Valor de
entrada 1

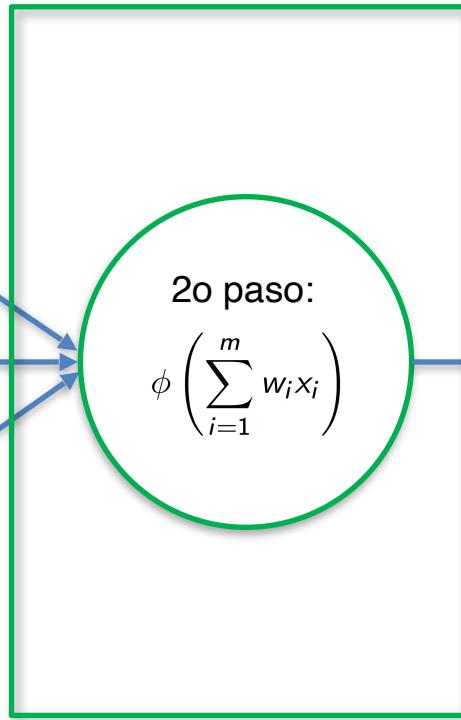


2o paso:

$$\phi \left(\sum_{i=1}^m w_i x_i \right)$$

Valor de salida

y



La Neurona

Valor de
entrada 1



w_1

Valor de
entrada 2



w_2

Valor de
entrada m



w_m

2o paso:
 $\phi \left(\sum_{i=1}^m w_i x_i \right)$

3r paso



Valor de salida

La Función de Activación

La Función de Activación

Valor de
entrada 1

X_1

W_1

Valor de
entrada 2

X_2

W_2

Valor de
entrada m

X_m

W_m

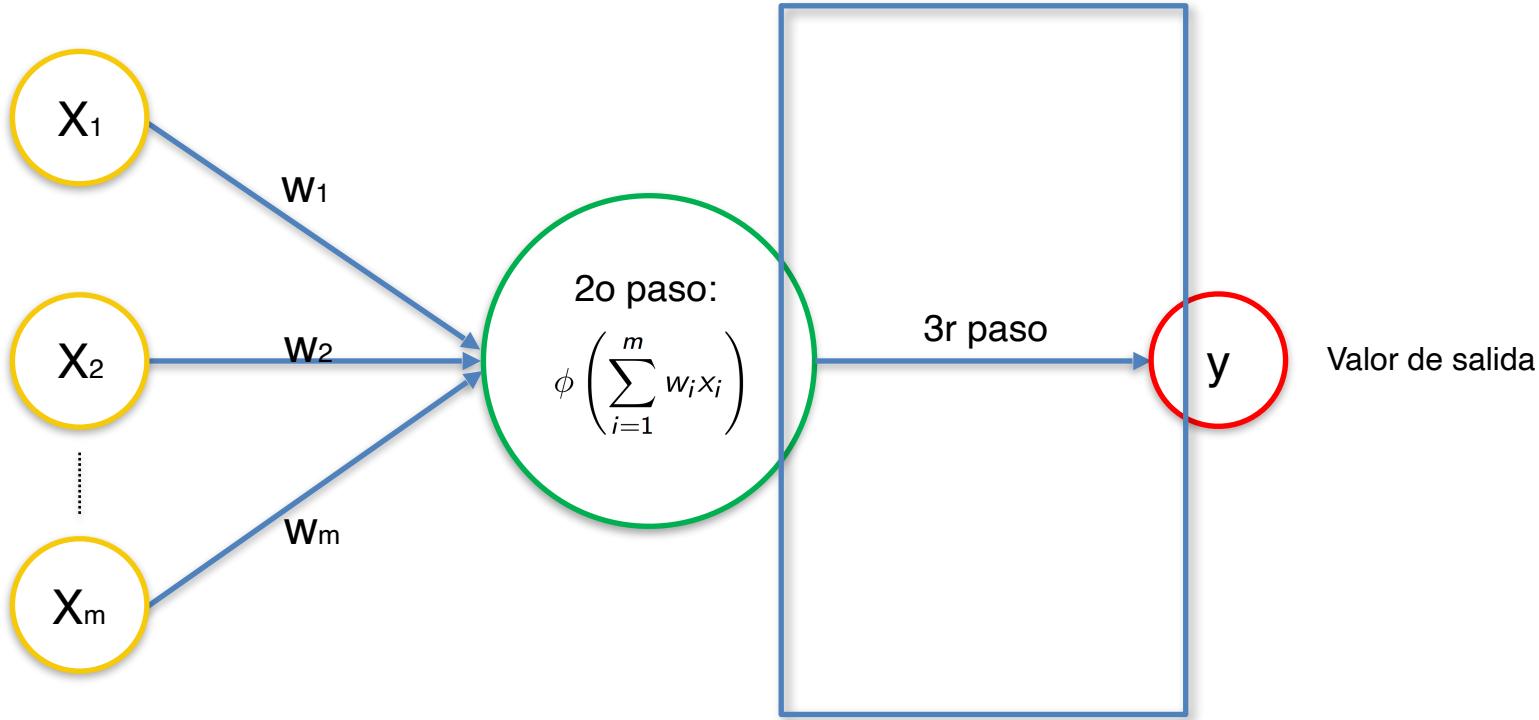
2o paso:

$$\phi \left(\sum_{i=1}^m w_i x_i \right)$$

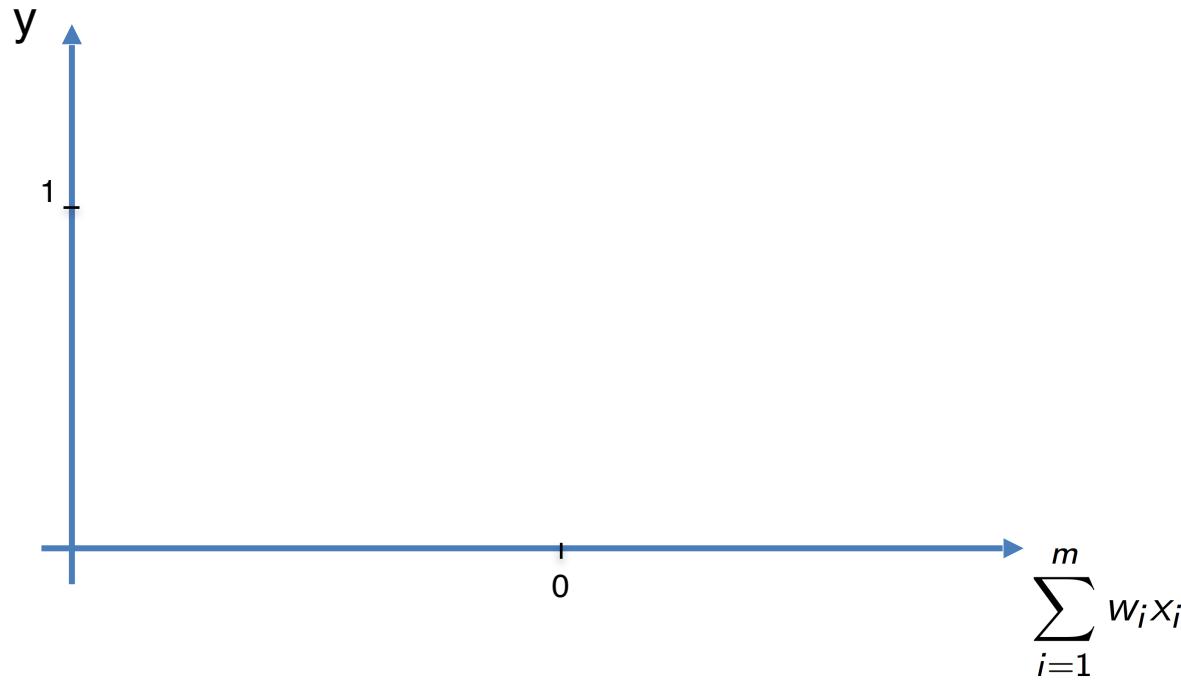
3r paso

y

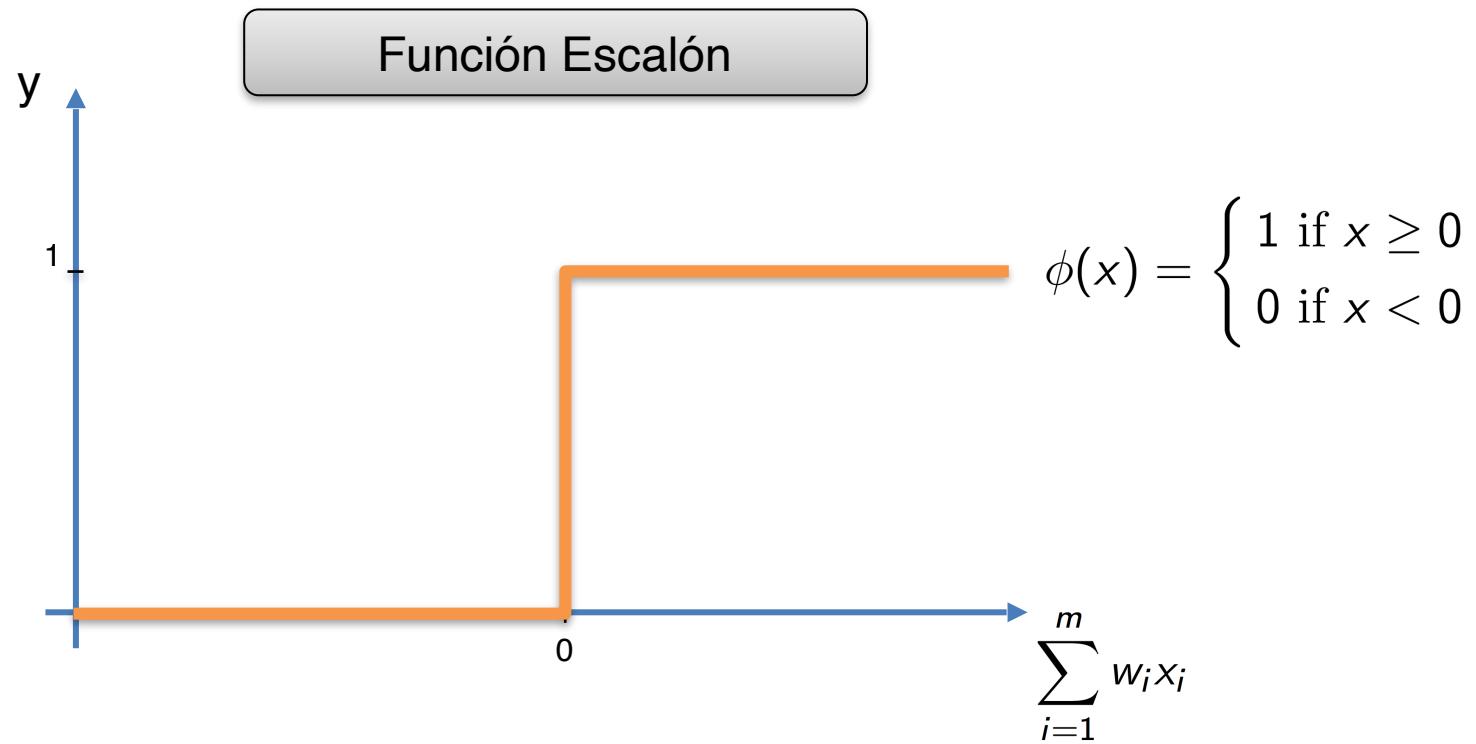
Valor de salida



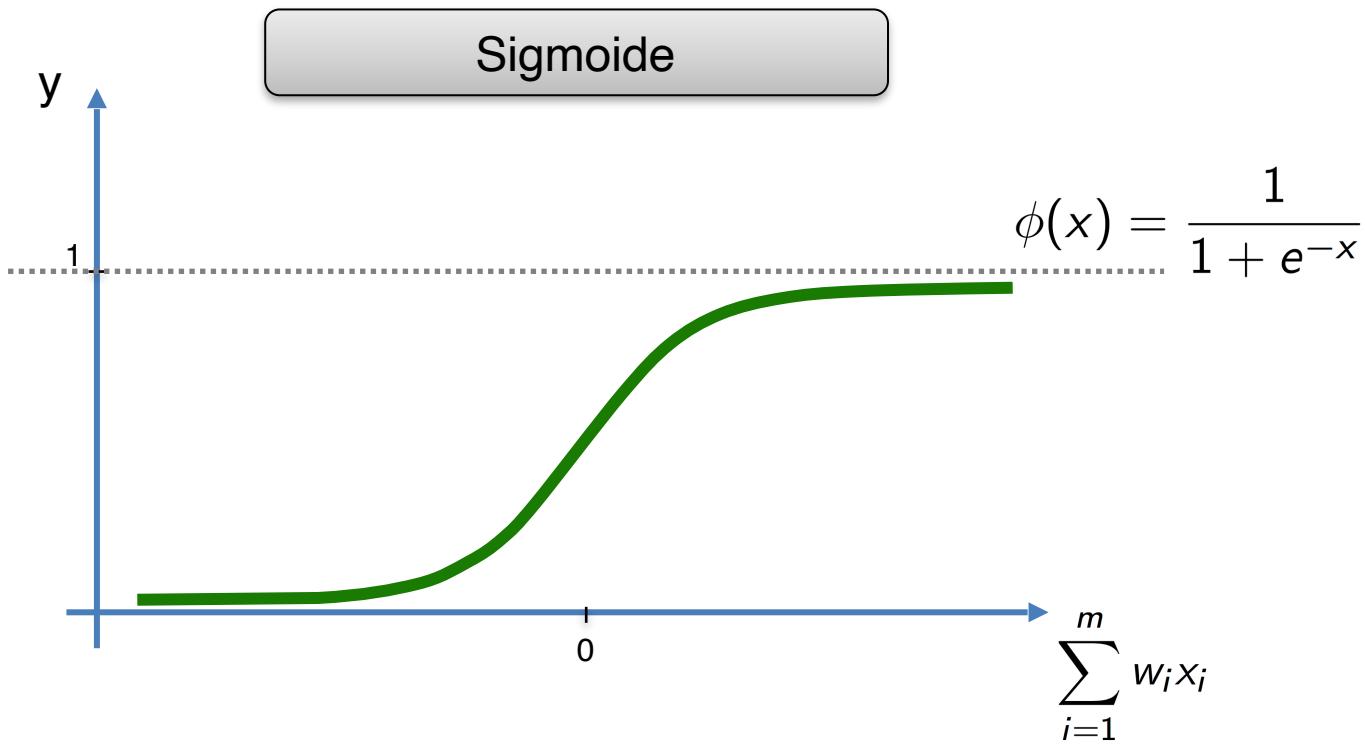
La Función de Activación



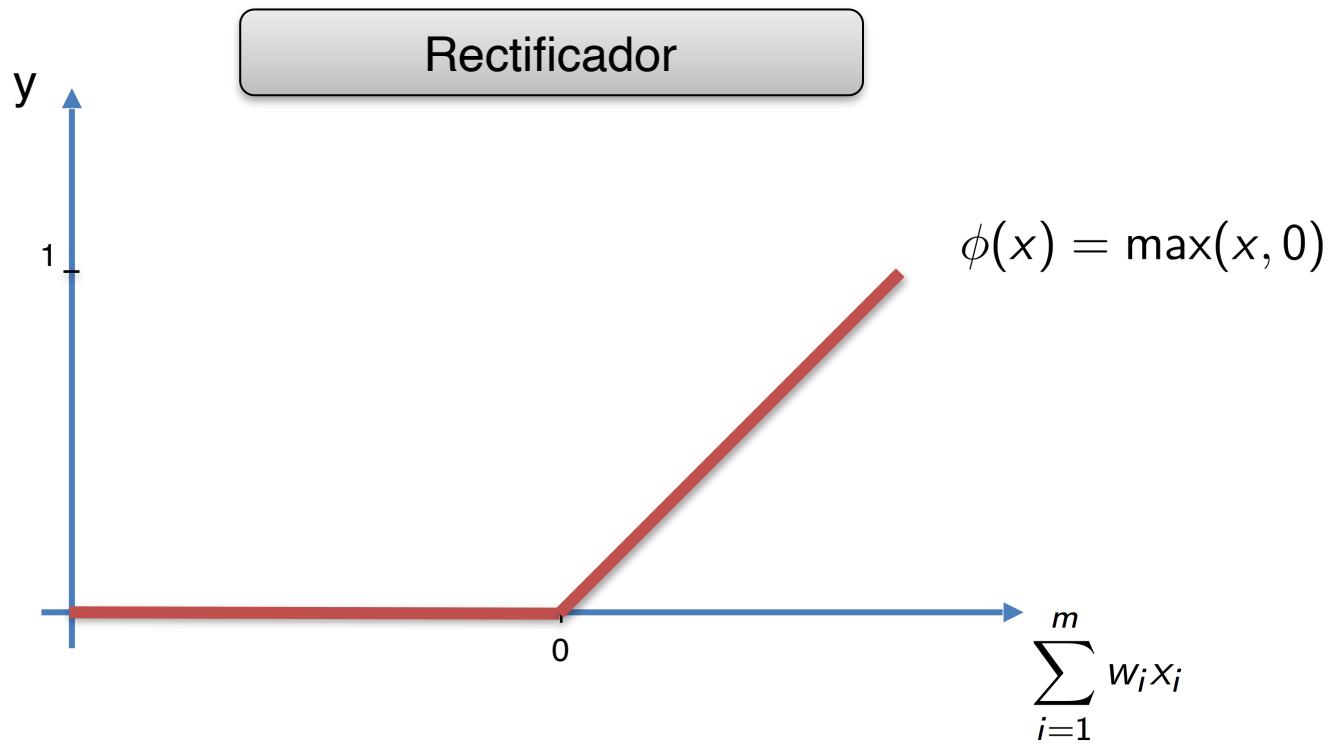
La Función de Activación



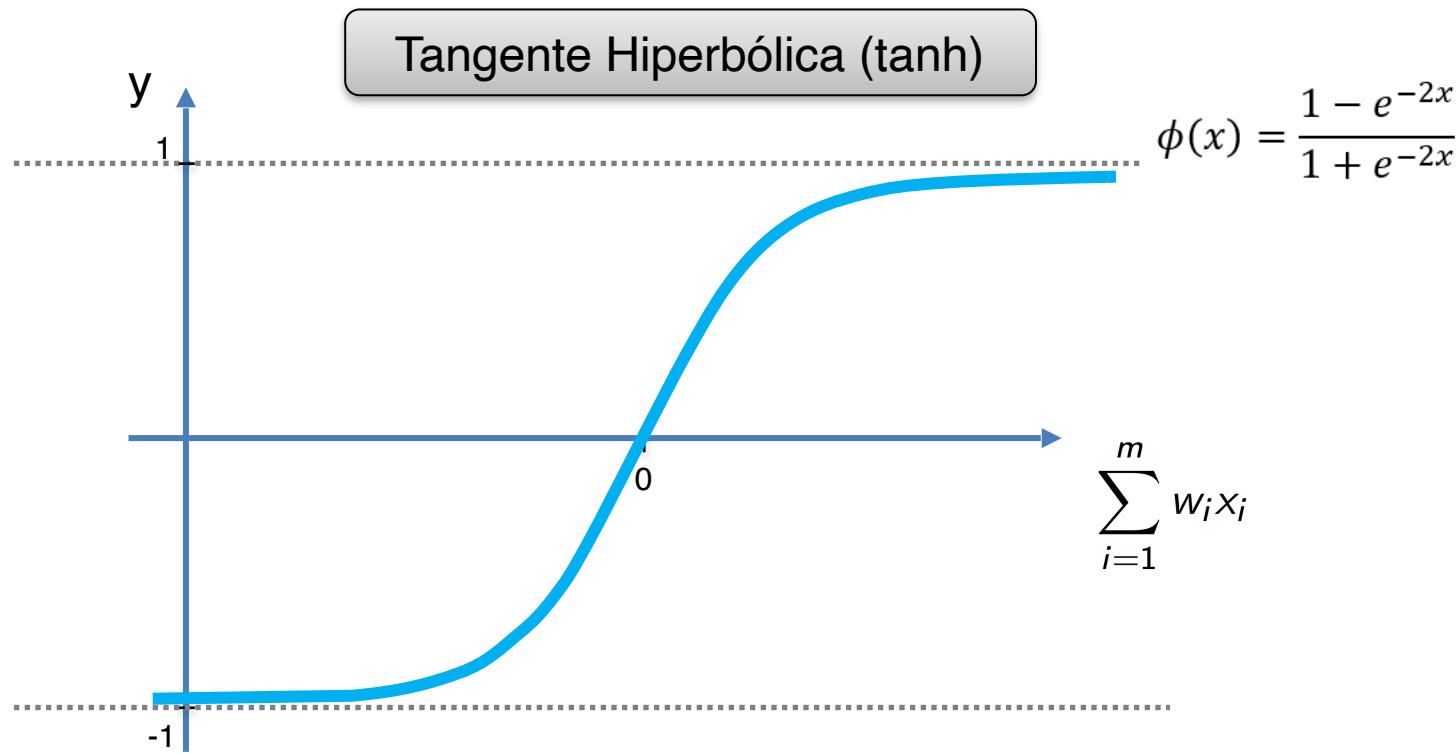
La Función de Activación



La Función de Activación



La Función de Activación

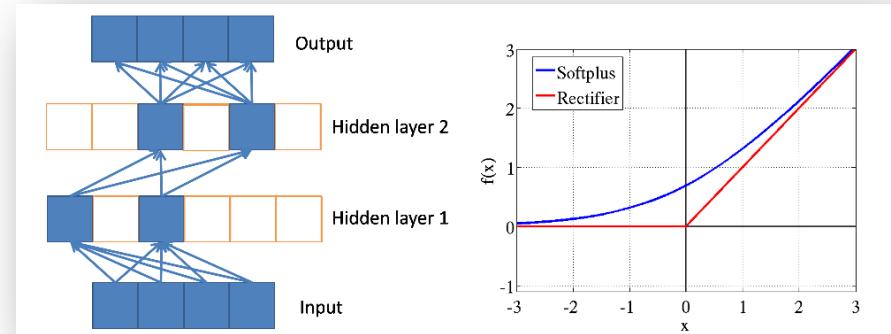


La Función de Activación

Lecturas Adicionales:

*Deep sparse Rectificador
neural networks*

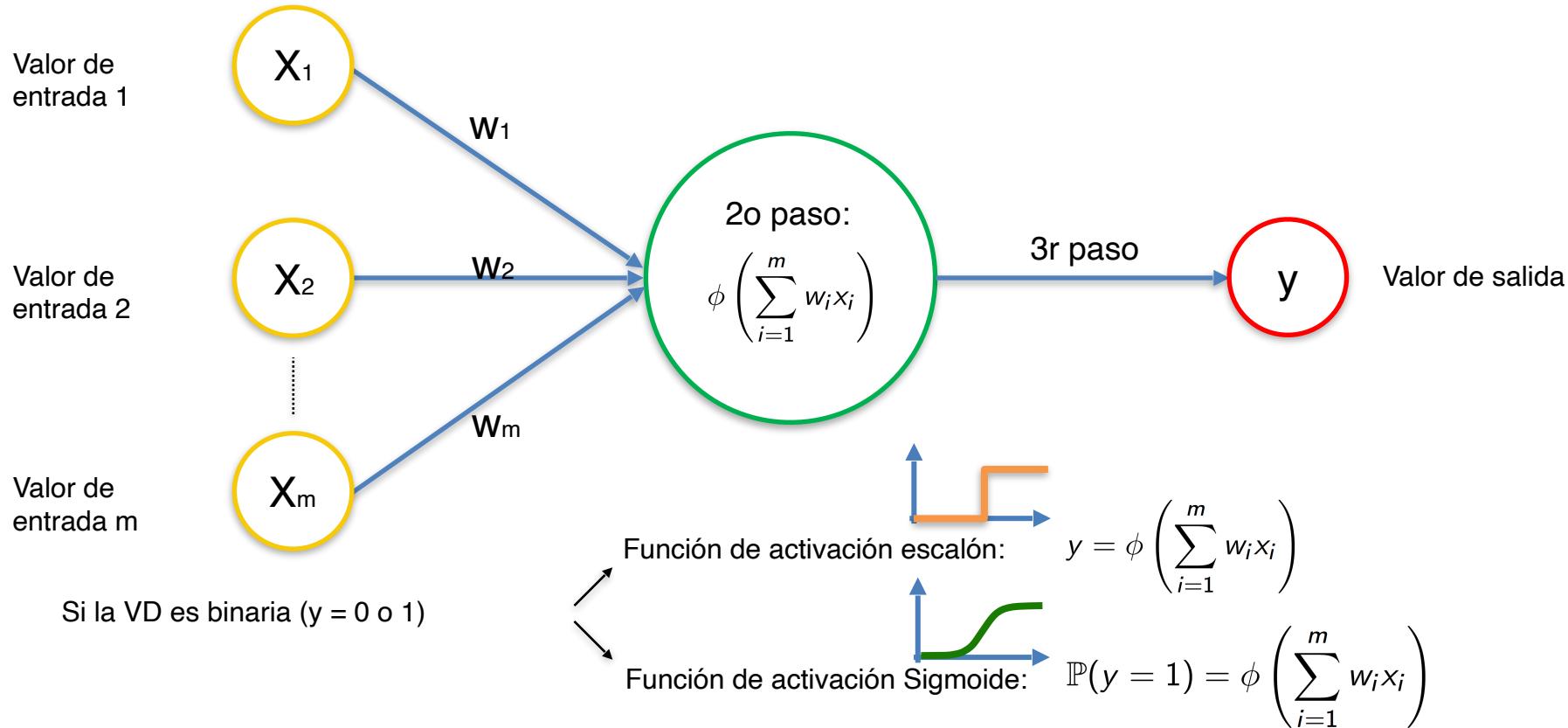
By Xavier Glorot et al. (2011)



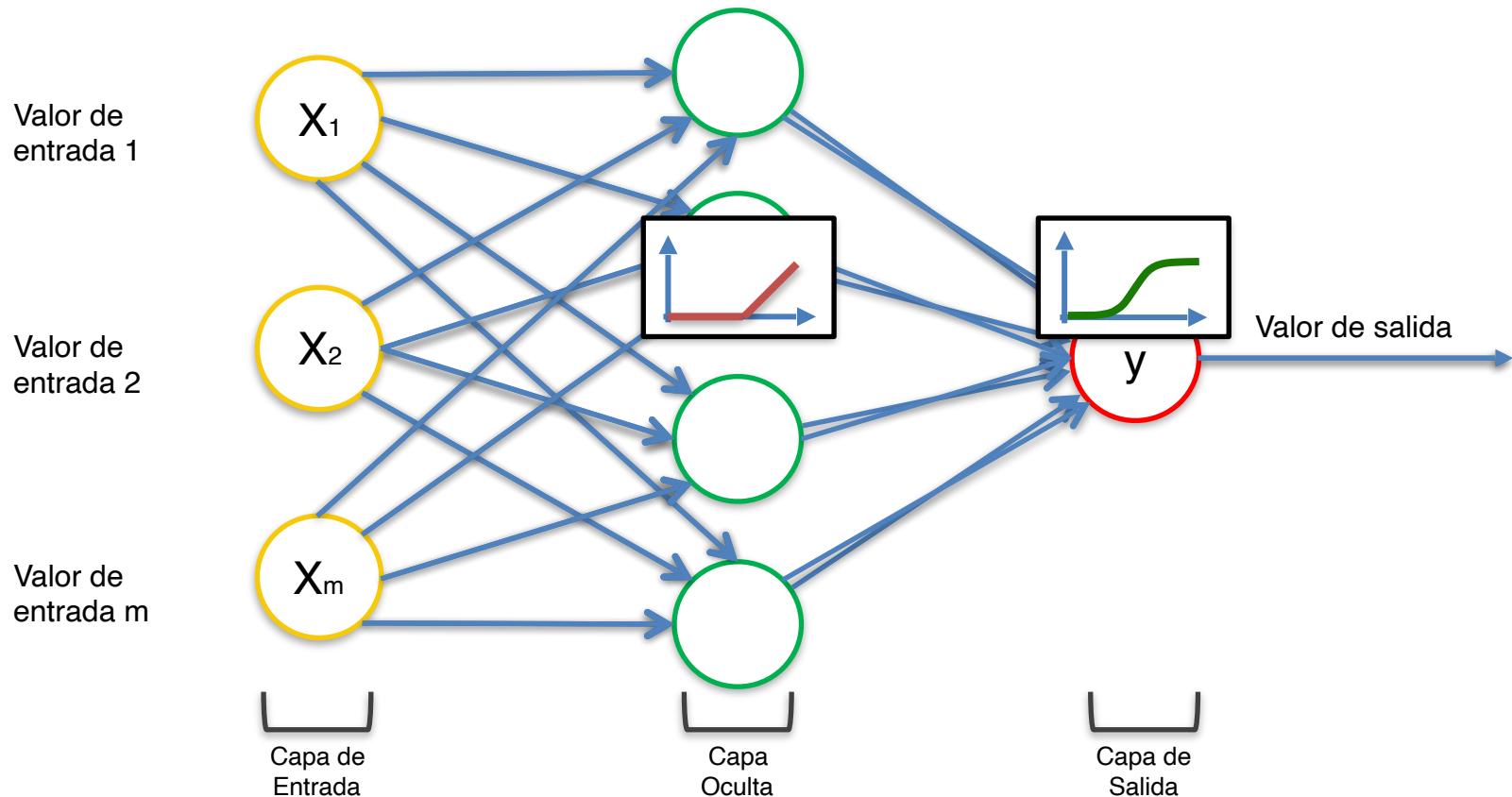
Link:

<http://jmlr.org/proceedings/papers/v15/glorot11a/glorot11a.pdf>

La Función de Activación



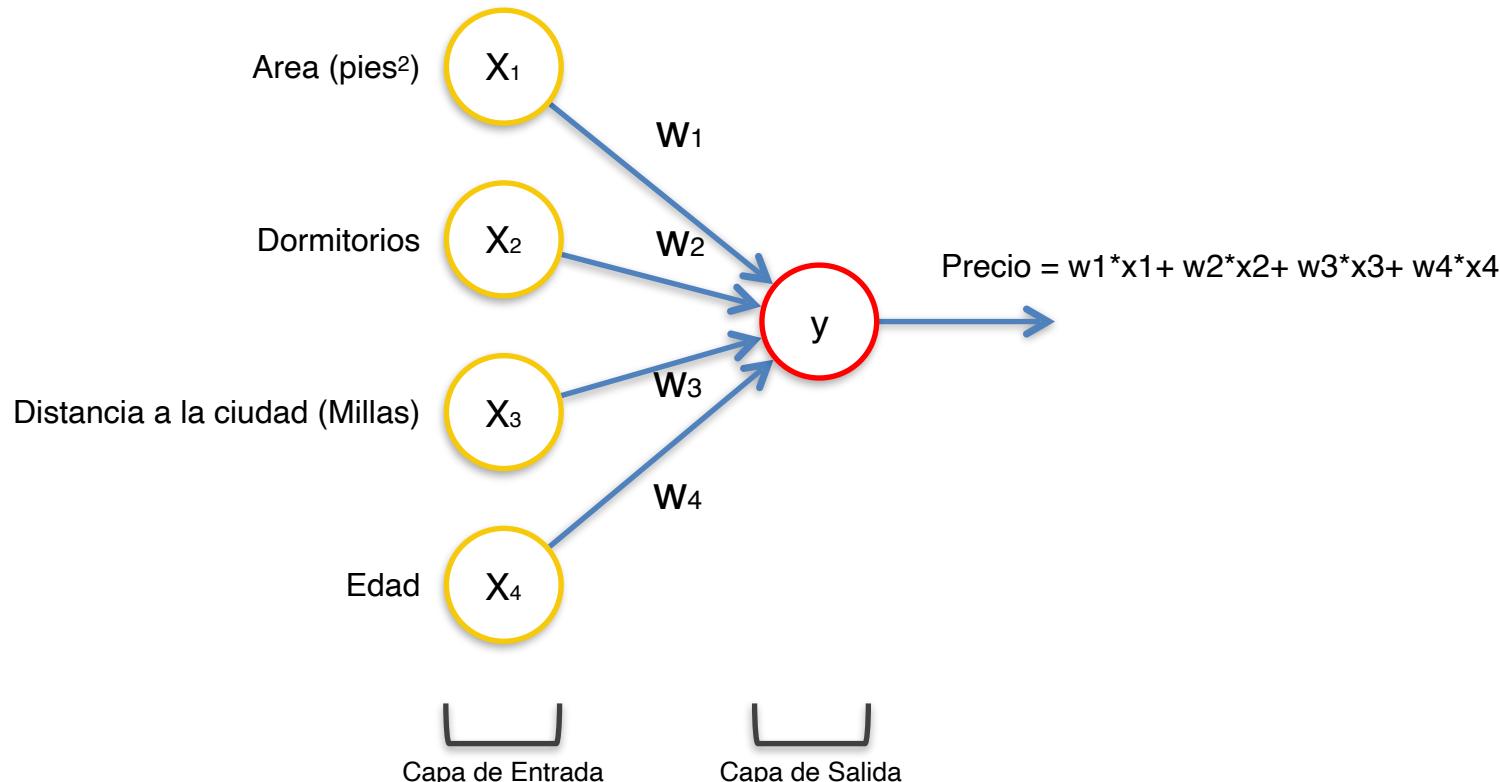
La Función de Activación



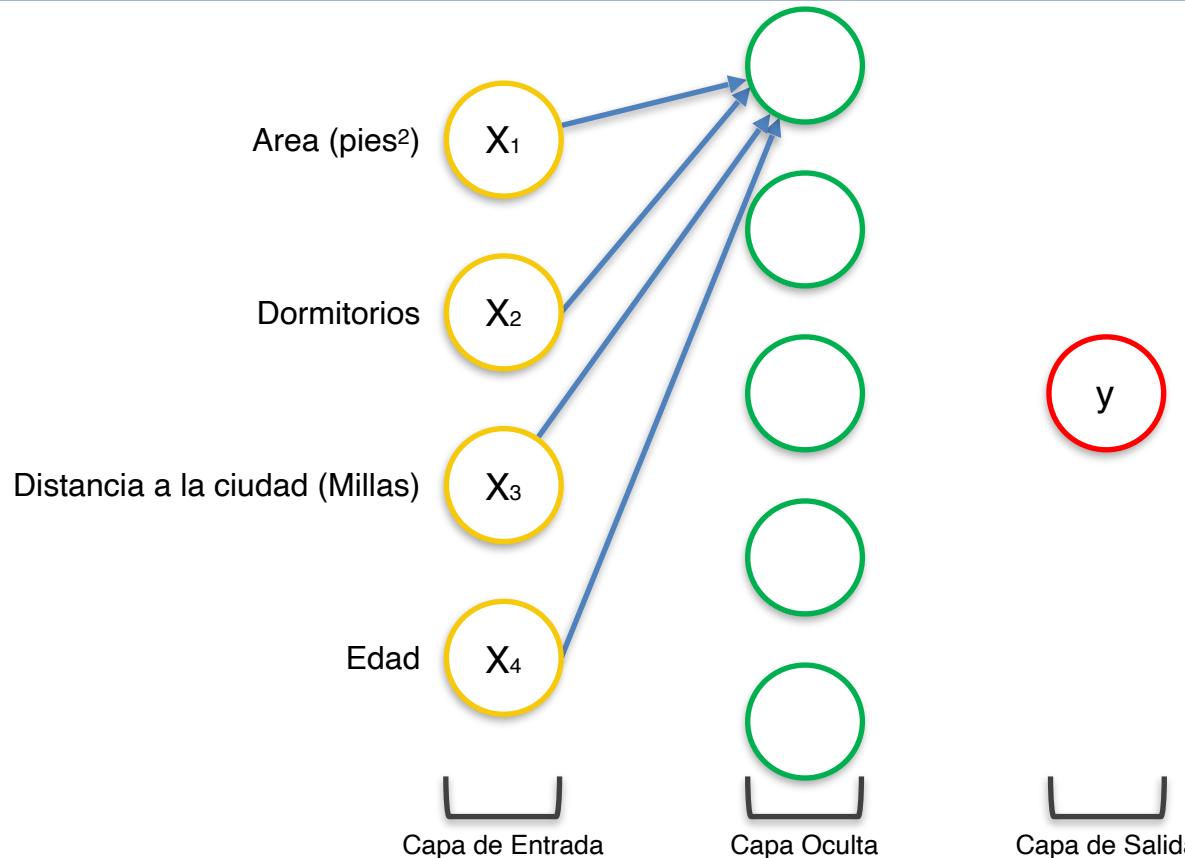
¿Cómo funcionan las Redes Neuronales?



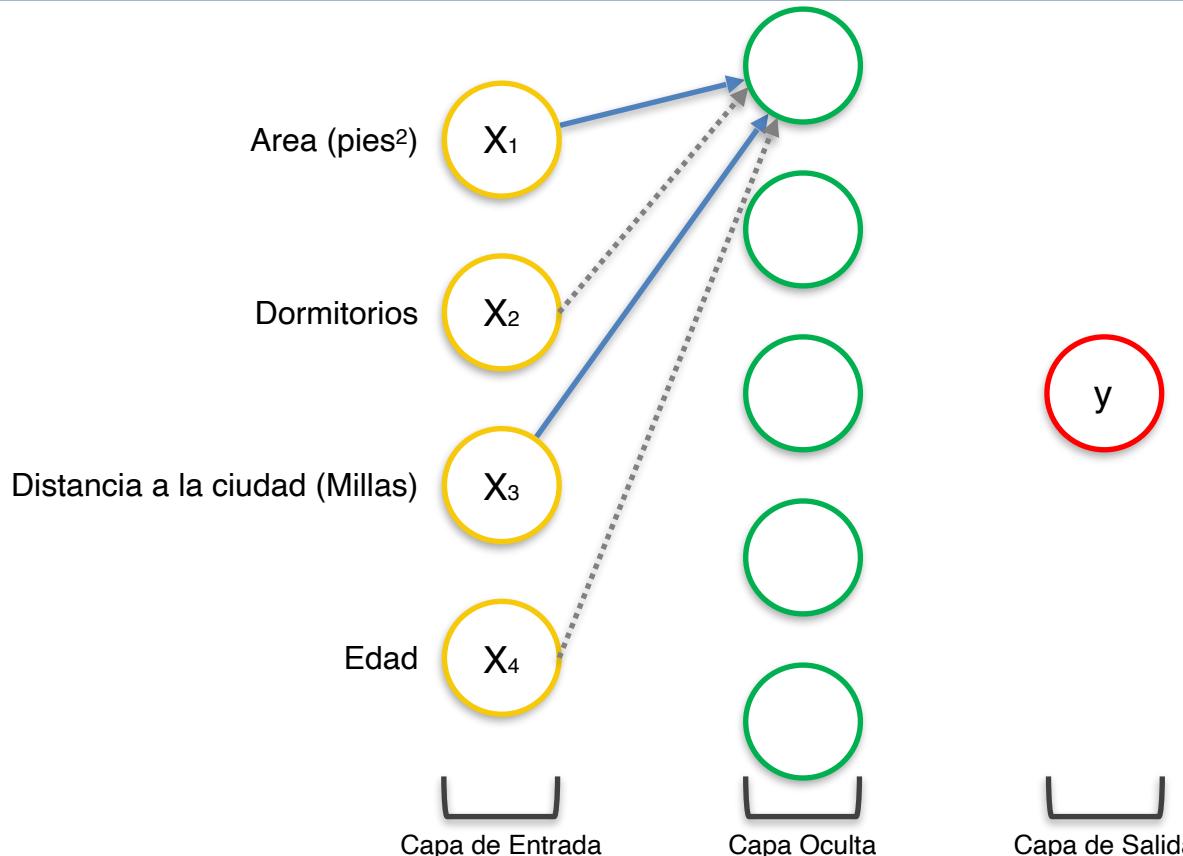
¿Cómo funcionan las Redes Neuronales?



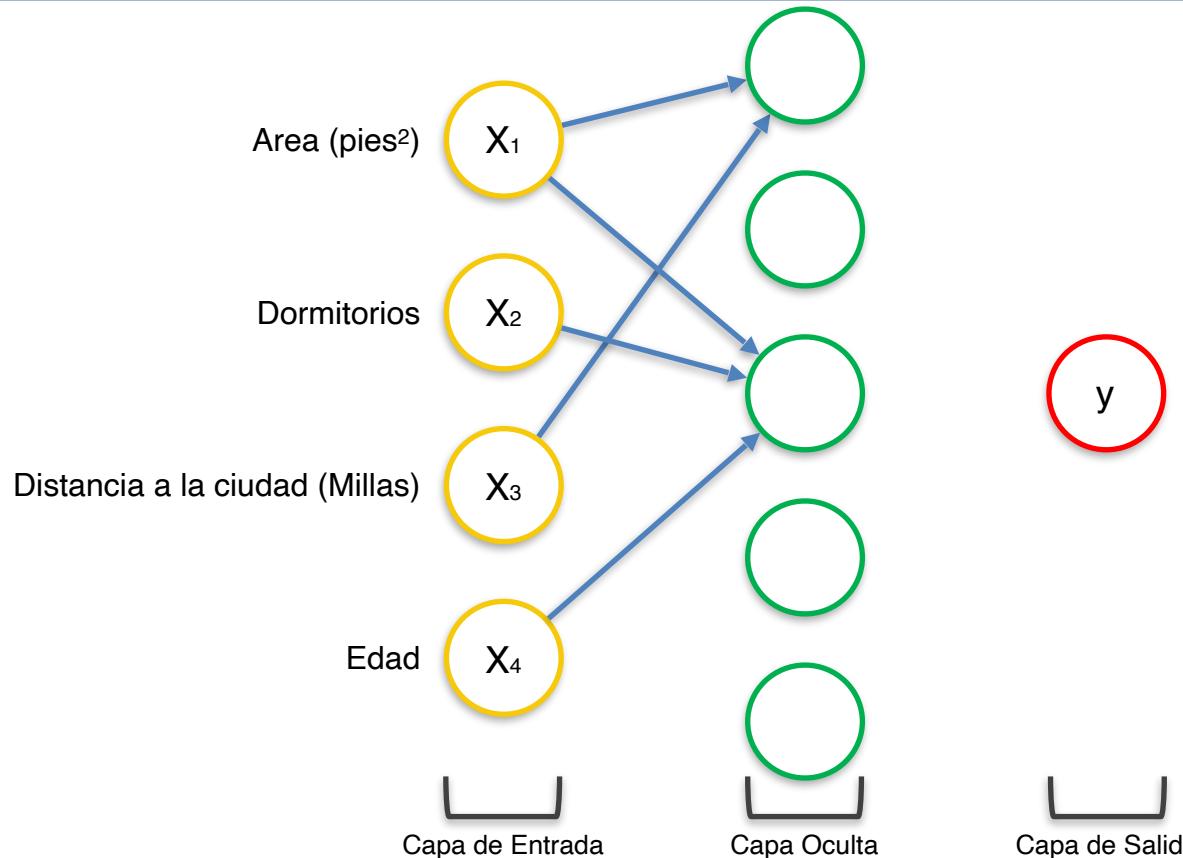
¿Cómo funcionan las Redes Neuronales?



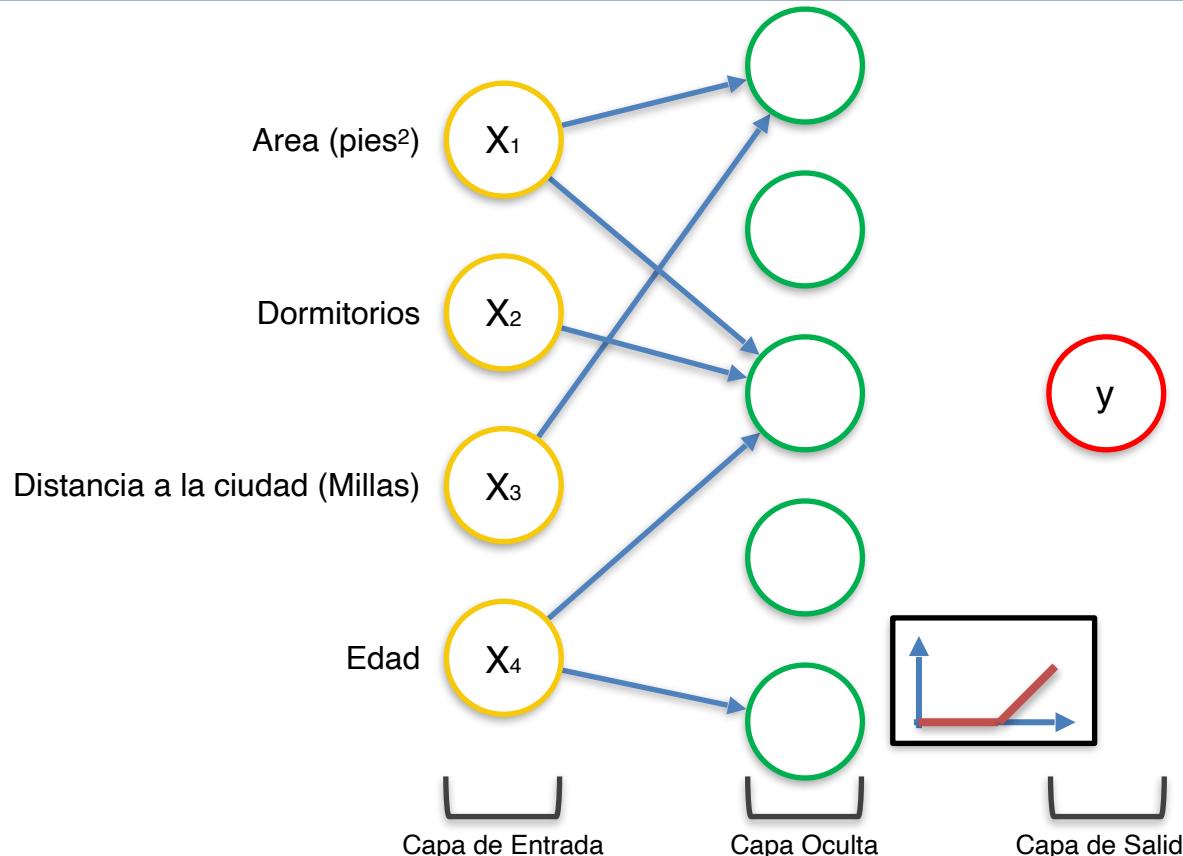
¿Cómo funcionan las Redes Neuronales?



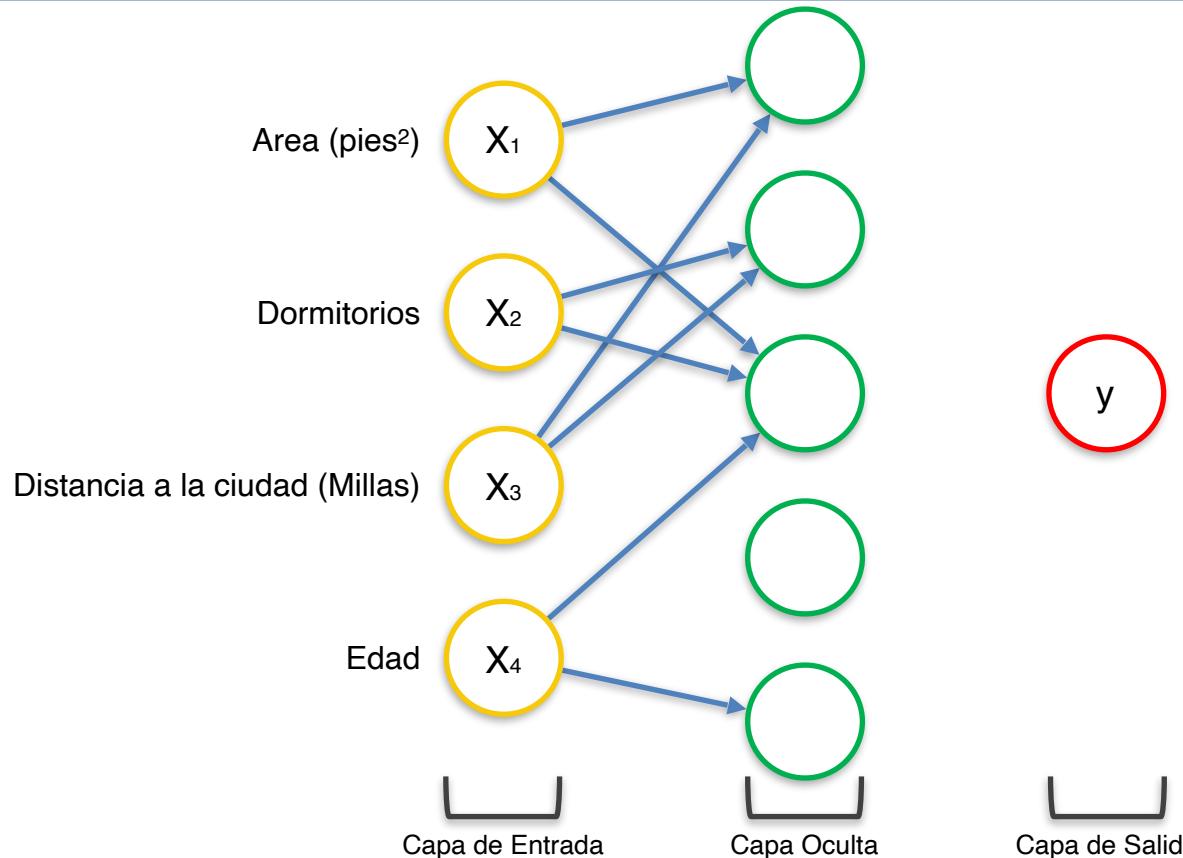
¿Cómo funcionan las Redes Neuronales?



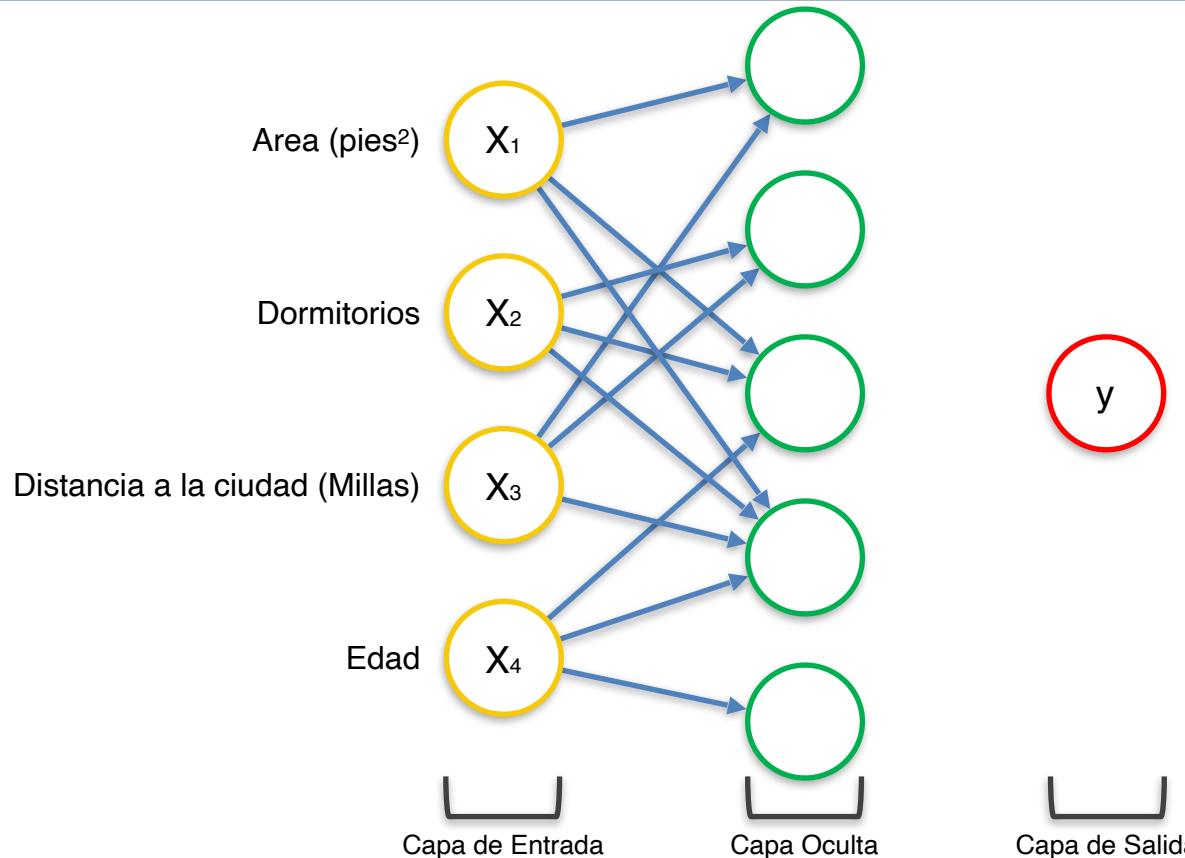
¿Cómo funcionan las Redes Neuronales?



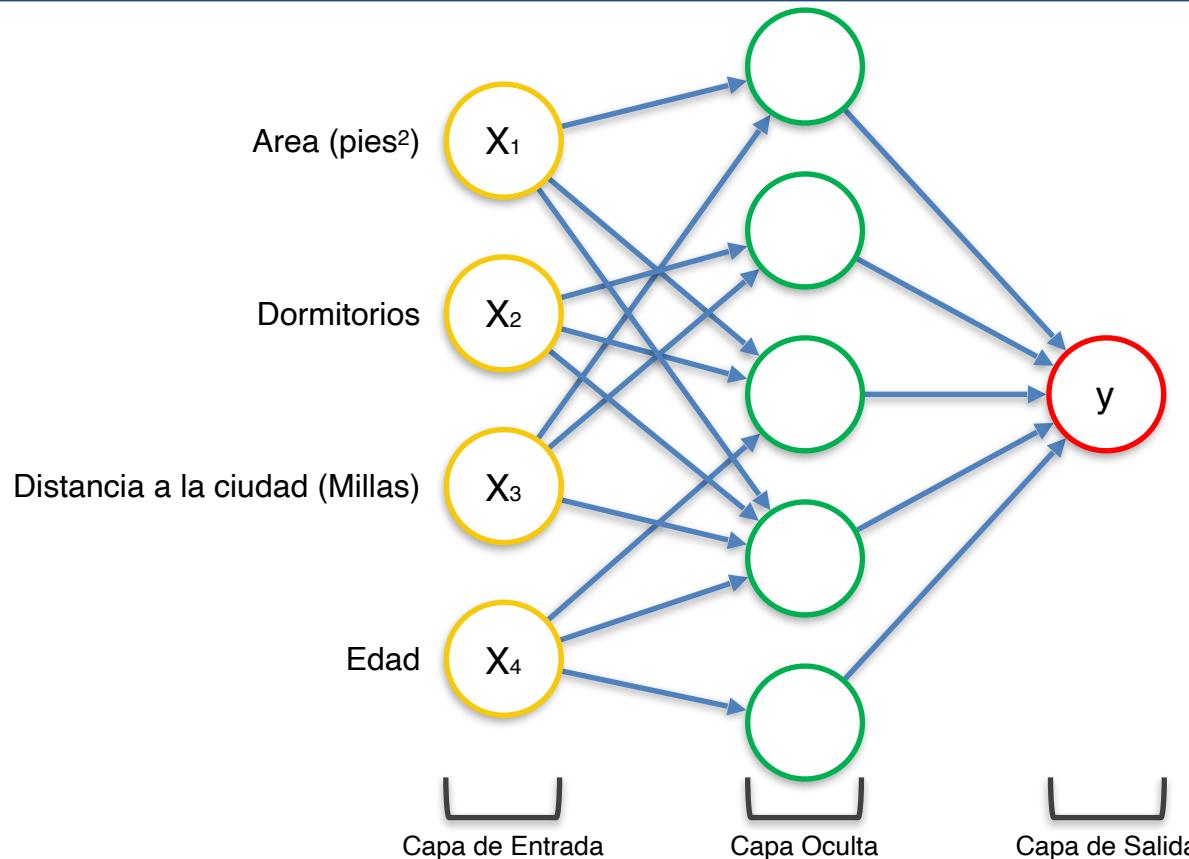
¿Cómo funcionan las Redes Neuronales?



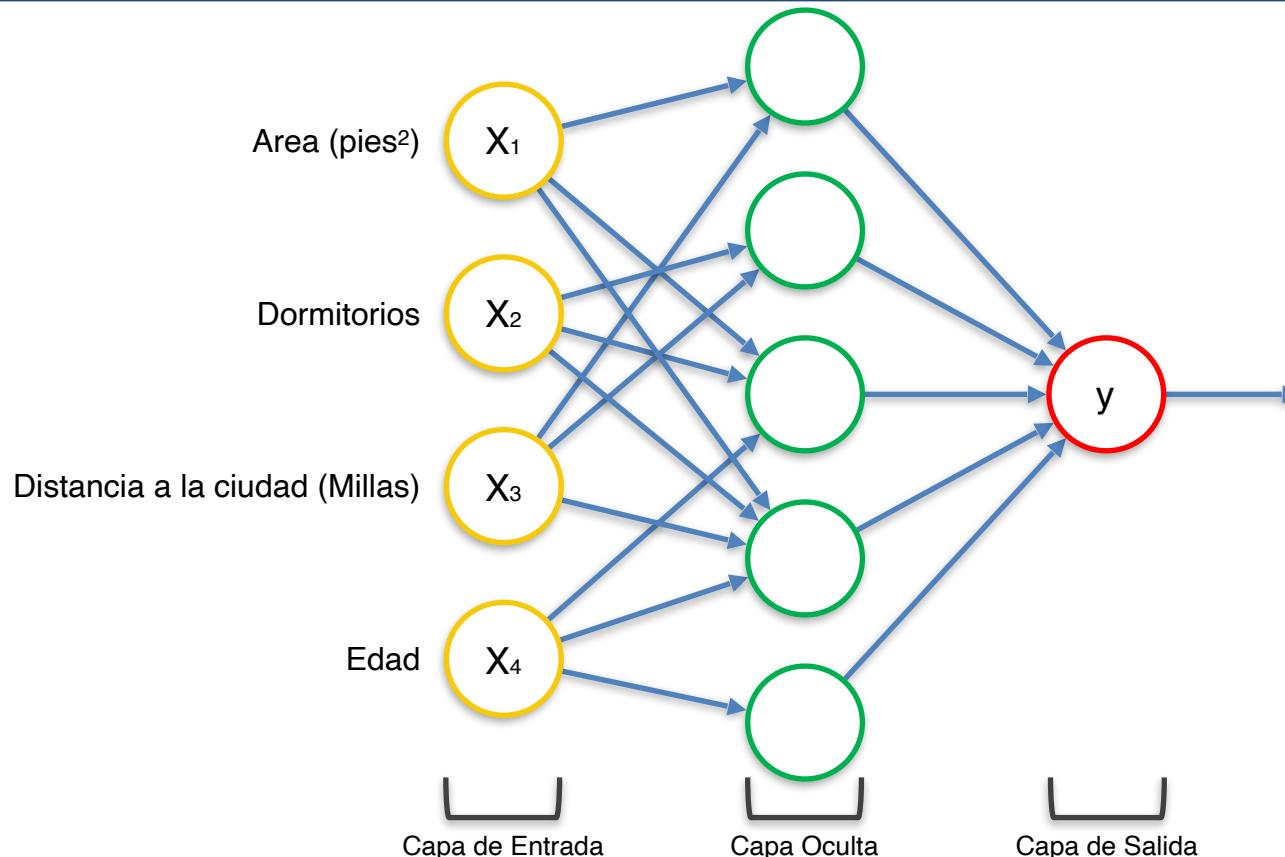
¿Cómo funcionan las Redes Neuronales?



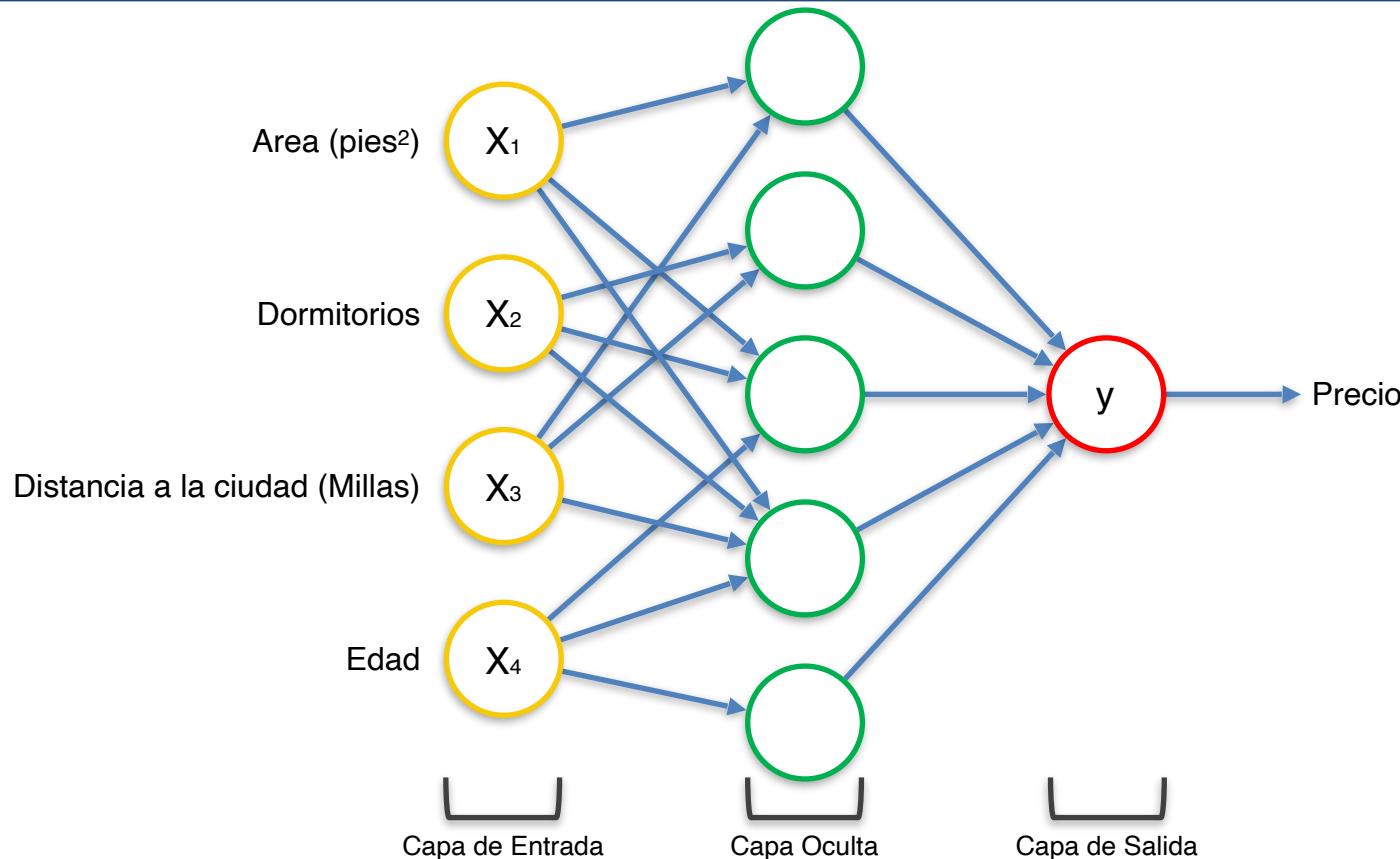
¿Cómo funcionan las Redes Neuronales?



¿Cómo funcionan las Redes Neuronales?



¿Cómo funcionan las Redes Neuronales?



¿Cómo aprenden las Redes Neuronales?

¿Cómo aprenden las Redes Neuronales?



¿Cómo aprenden las Redes Neuronales?

Valor de
entrada 1



w₁

Valor de
entrada 2



w₂

Valor de
entrada m



w_m

$$\phi \left(\sum_{i=1}^m w_i x_i \right)$$

\hat{y}

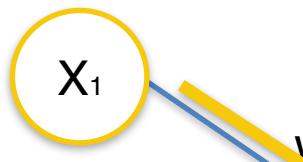
Valor de salida

y

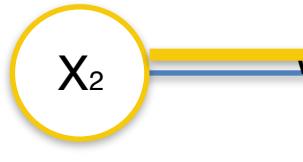
Valor Actual

¿Cómo aprenden las Redes Neuronales?

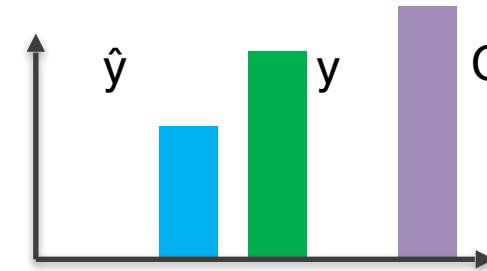
Valor de
entrada 1



Valor de
entrada 2



Valor de
entrada m



Valor de salida

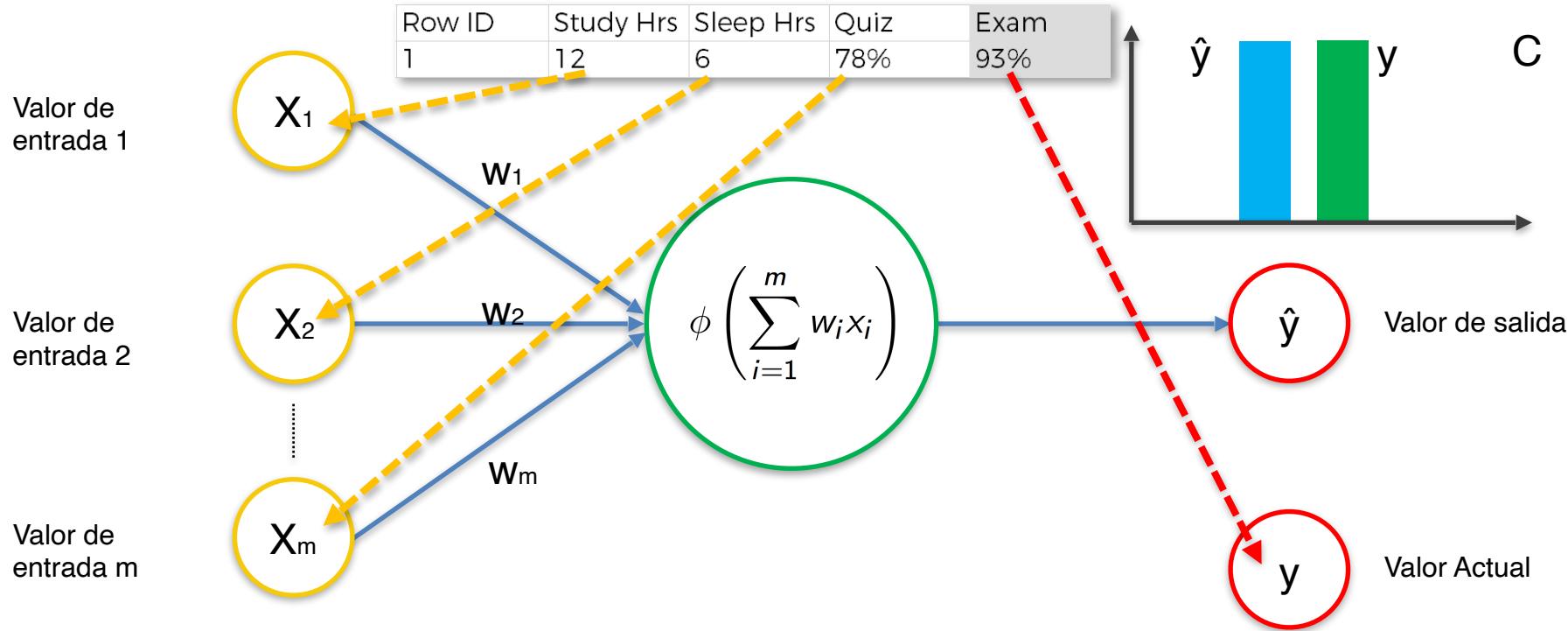
\hat{y}

y

$$C = \frac{1}{2}(\hat{y} - y)^2$$

Valor Actual

¿Cómo aprenden las Redes Neuronales?



¿Cómo aprenden las Redes Neuronales?

Valor de
entrada 1



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%

Valor de
entrada 2



Valor de
entrada m



w_1

w_2

w_m

$$\phi \left(\sum_{i=1}^m w_i x_i \right)$$



\hat{y}

Valor de salida

y

Valor Actual

$$C = \frac{1}{2}(\hat{y} - y)^2$$

¿Cómo aprenden las Redes Neuronales?

Valor de
entrada 1



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%

Valor de
entrada 2



Valor de
entrada m

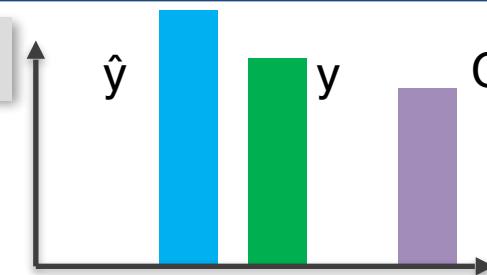


w_m

w_1

w_2

$$\phi \left(\sum_{i=1}^m w_i x_i \right)$$



\hat{y}

Valor de salida

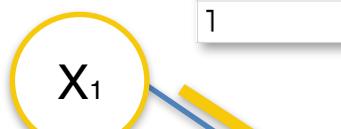
$$C = \frac{1}{2}(\hat{y} - y)^2$$

y

Valor Actual

¿Cómo aprenden las Redes Neuronales?

Valor de
entrada 1



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%

Valor de
entrada 2



Valor de
entrada m



w_1

w_2

w_m

$$\phi \left(\sum_{i=1}^m w_i x_i \right)$$



\hat{y}

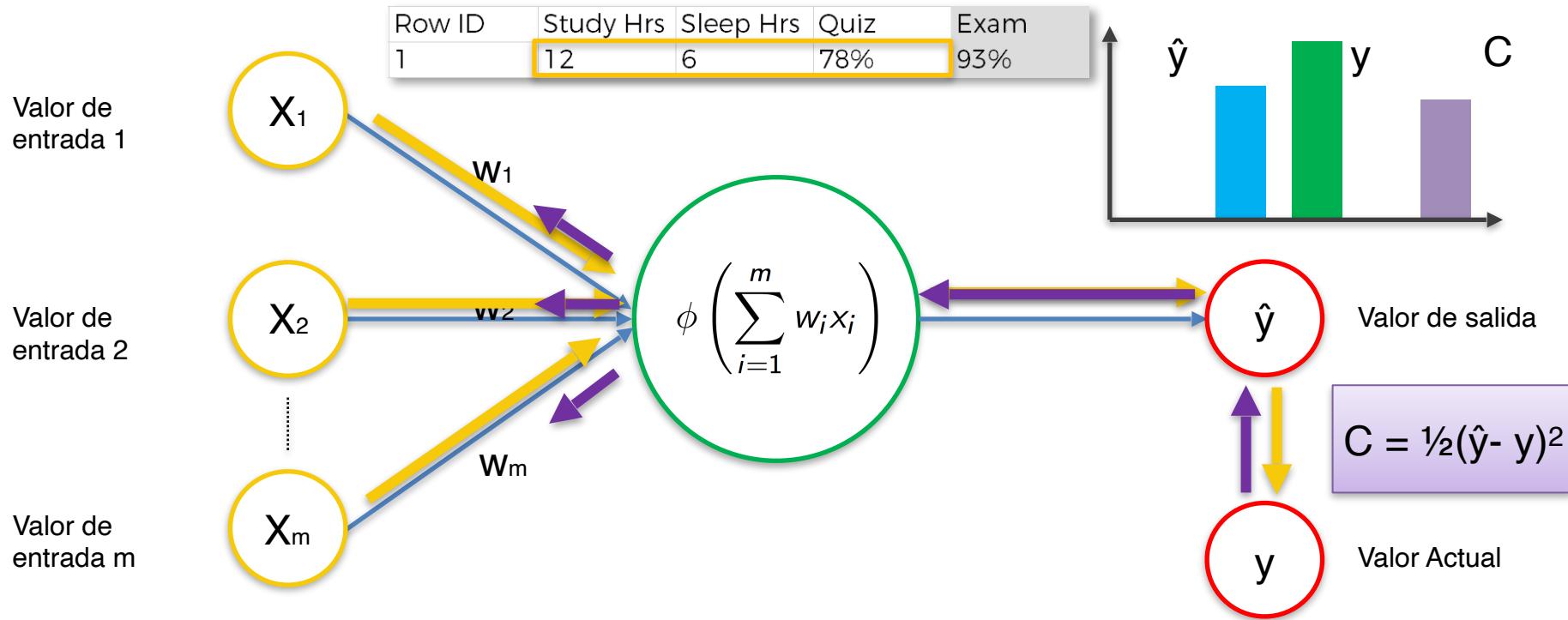
Valor de salida

$$C = \frac{1}{2}(\hat{y} - y)^2$$

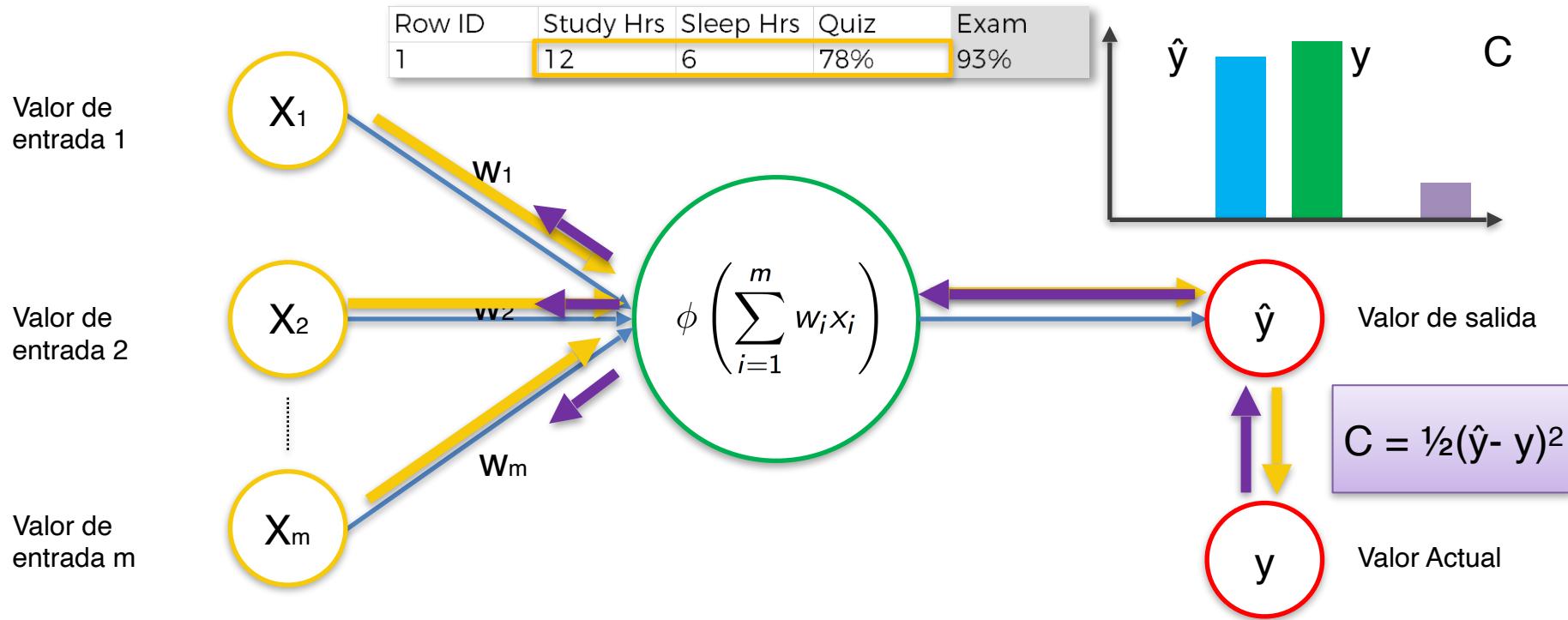
y

Valor Actual

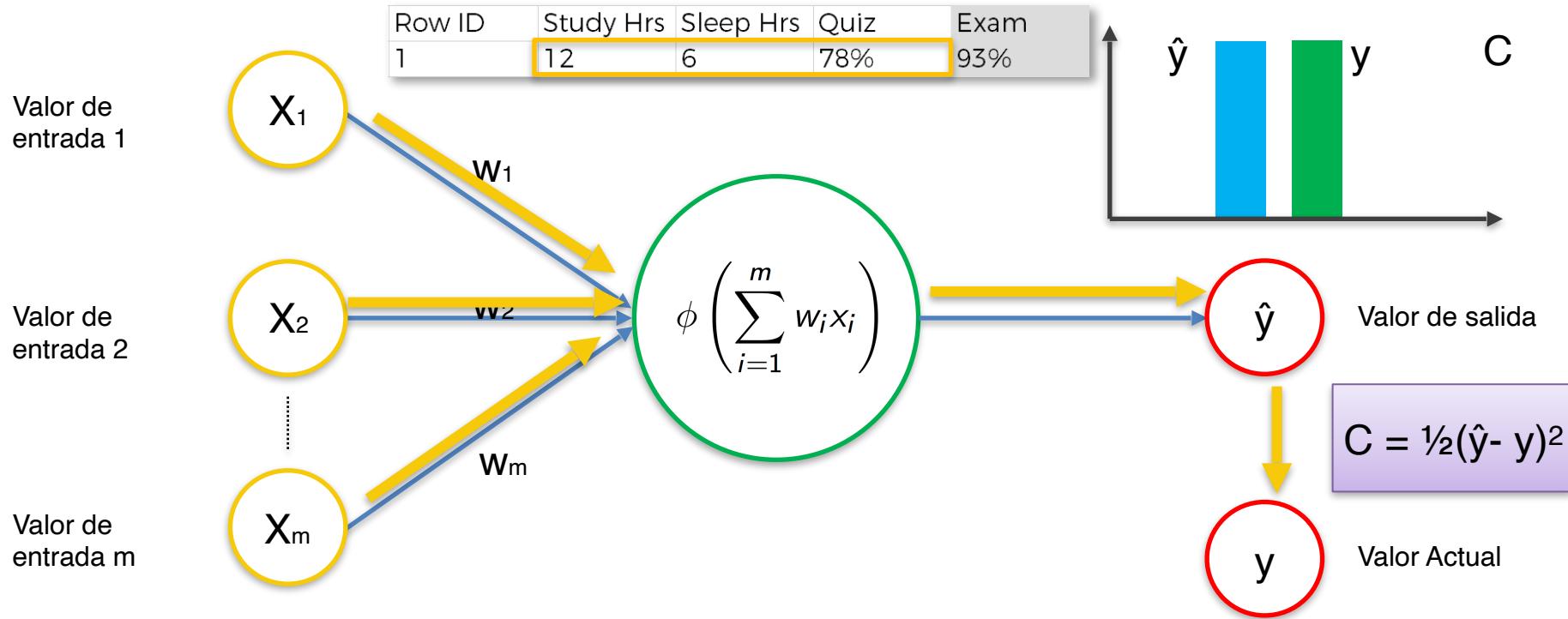
¿Cómo aprenden las Redes Neuronales?



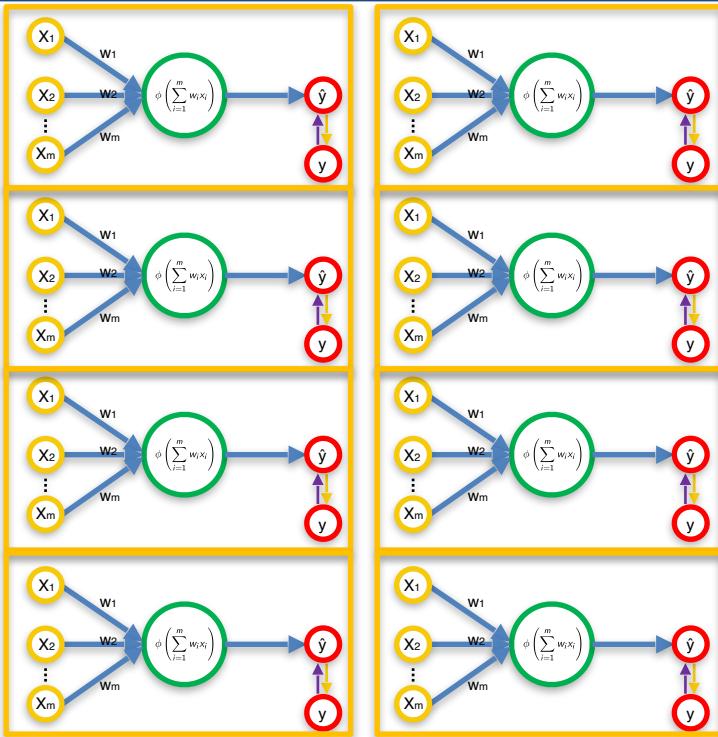
¿Cómo aprenden las Redes Neuronales?



¿Cómo aprenden las Redes Neuronales?

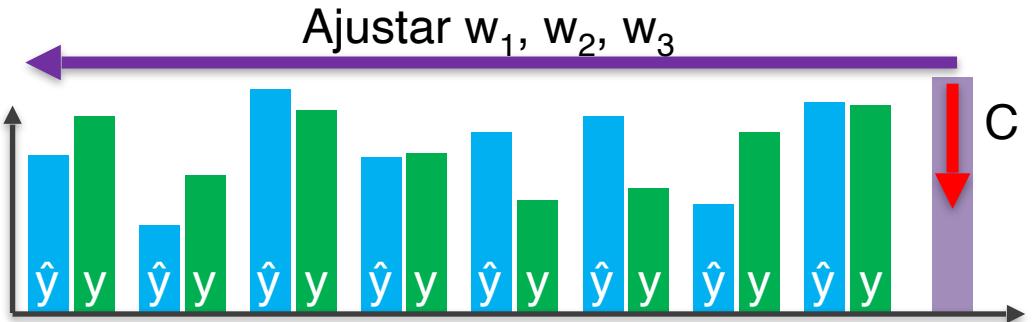


¿Cómo aprenden las Redes Neuronales?



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum \frac{1}{2}(\hat{y} - y)^2$$



¿Cómo aprenden las Redes Neuronales?

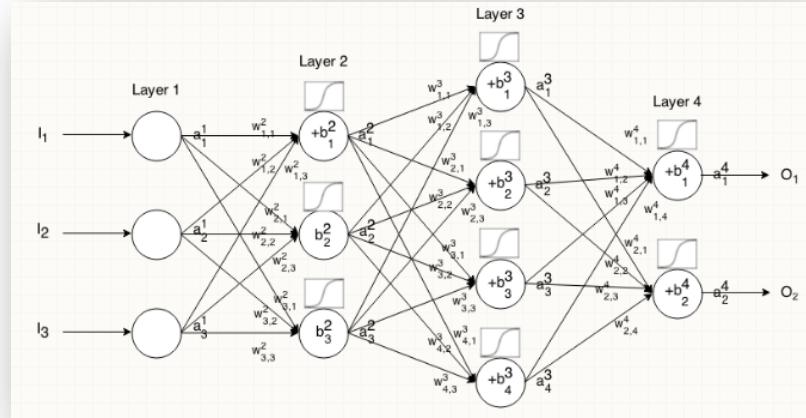
Lecturas Adicionales:

A list of cost functions used in neural networks, alongside applications

CrossValidated (2015)

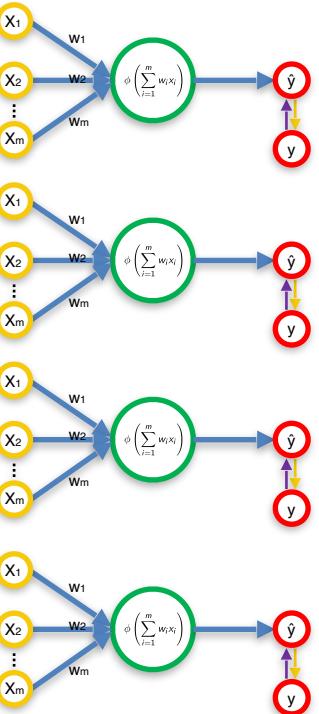
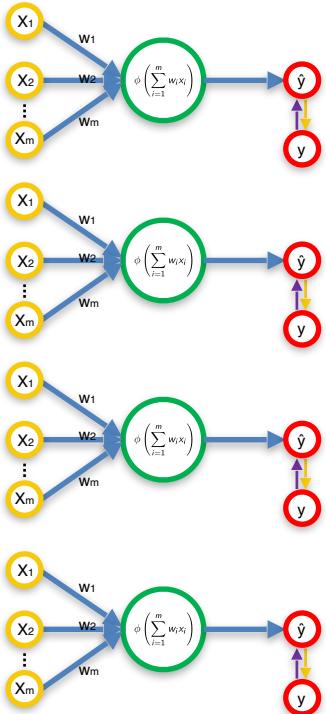
Link:

<http://stats.stackexchange.com/questions/154879/a-list-of-cost-functions-used-in-neural-networks-alongside-applications>



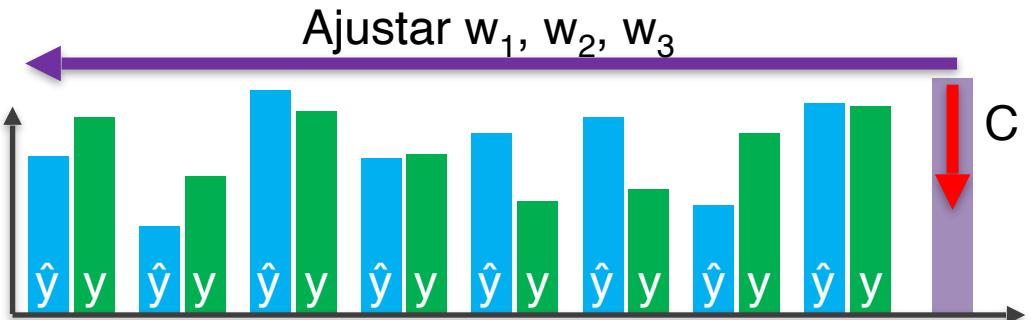
Gradiente Descendente

Gradiente Descendente

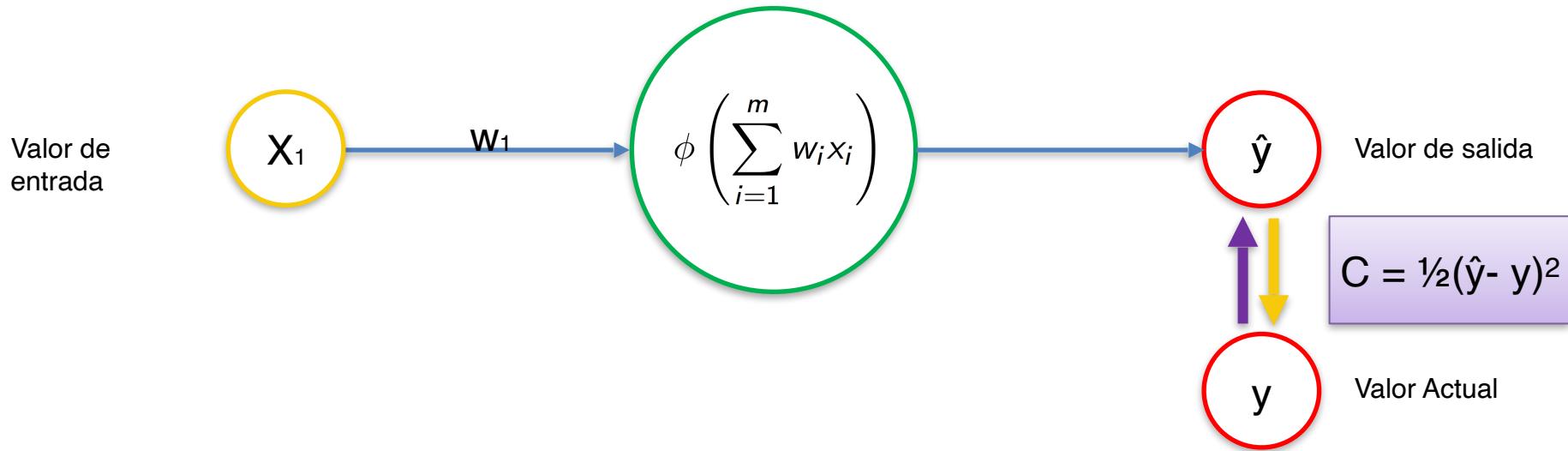


Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

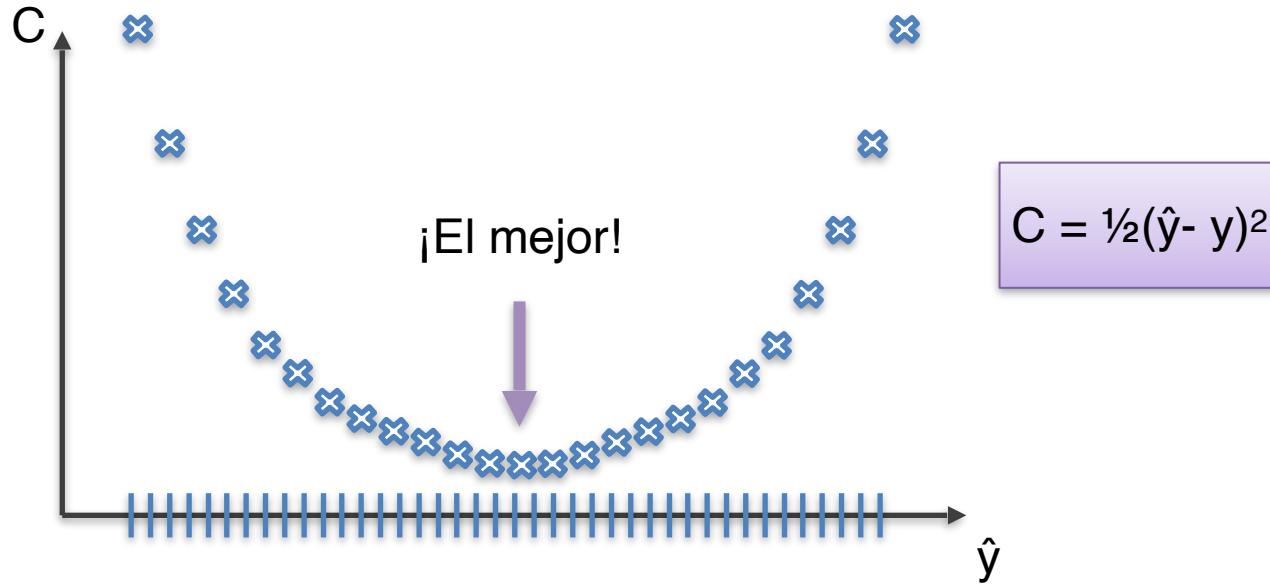
$$C = \sum \frac{1}{2}(\hat{y} - y)^2$$



Gradiente Descendente



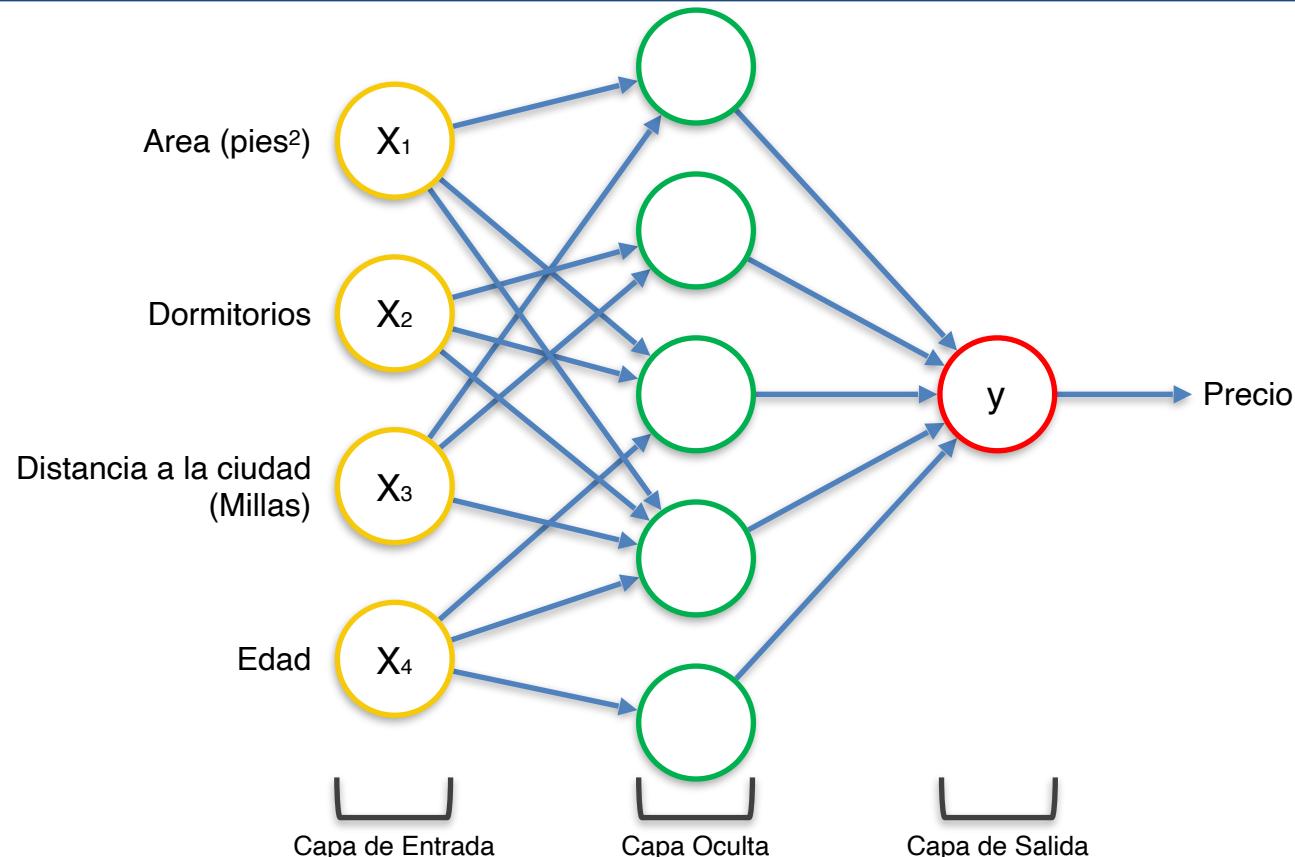
Gradiente Descendente



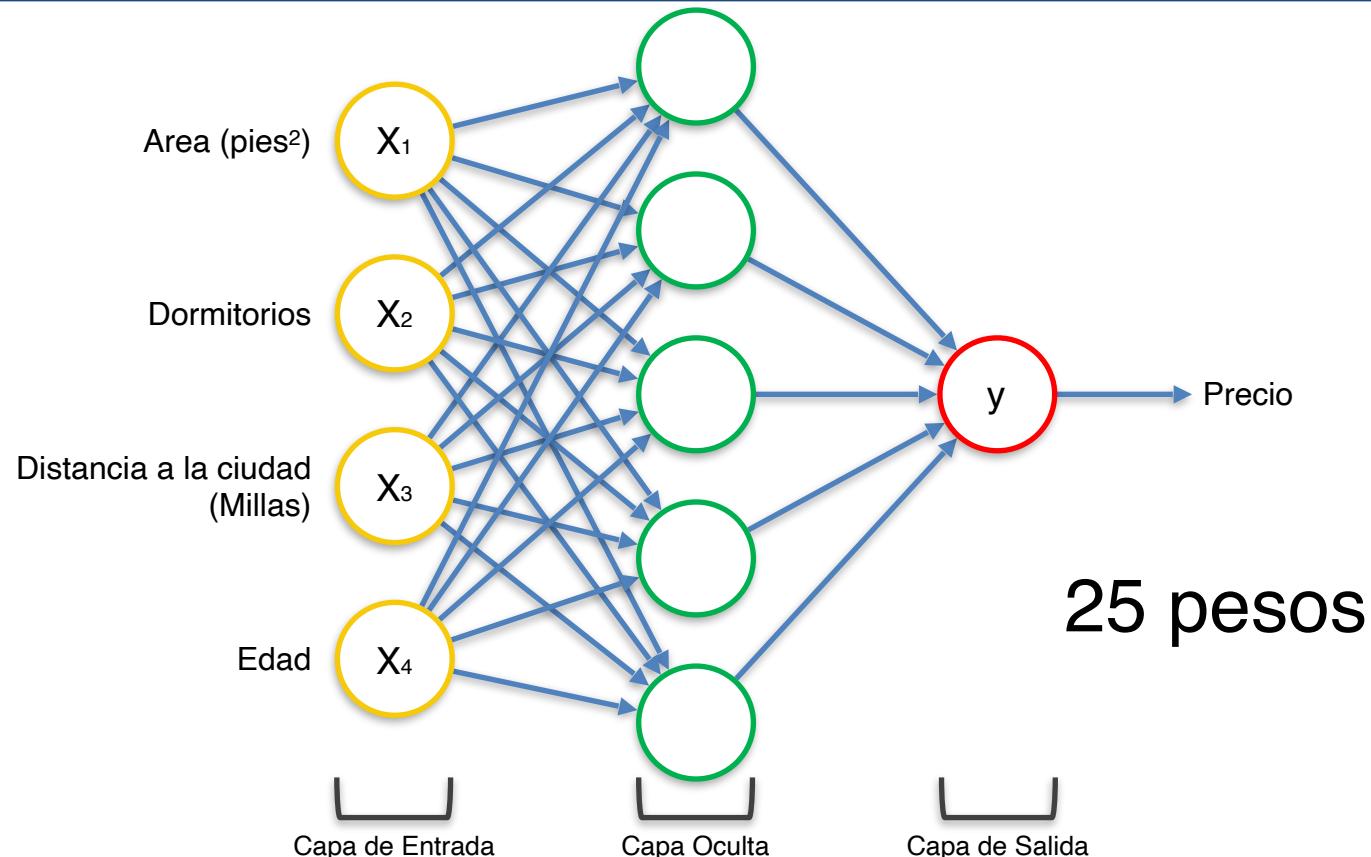
Gradiente Descendente

La Maldición de la Dimensión

Gradiente Descendente



Gradiente Descendente



Gradiente Descendente

$$1,000 \times 1,000 \times \dots \times 1,000 = 1,000^{25} = 10^{75} \text{ combinaciones}$$

Sunway TaihuLight: El Super Ordenador más potente del mundo

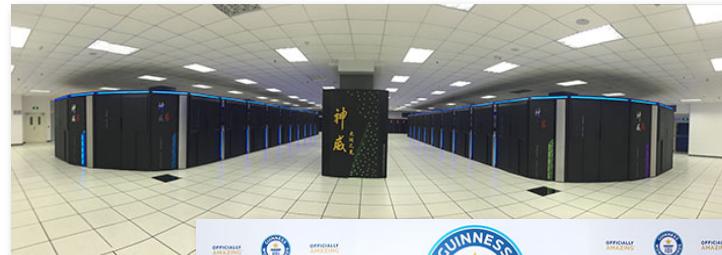
93 PFLOPS

$$93 \times 10^{15}$$

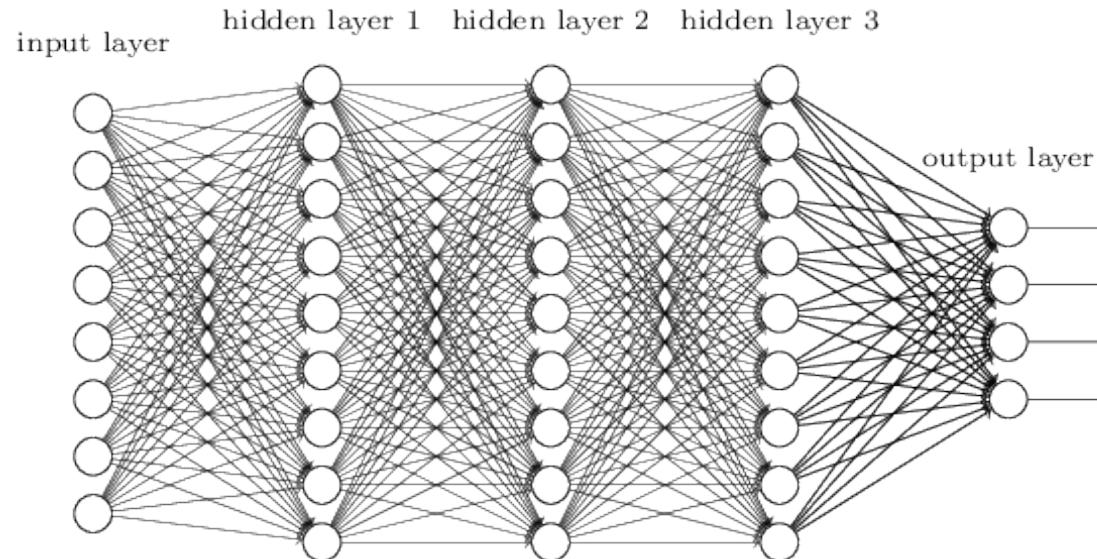
$$10^{75} / (93 \times 10^{15})$$

$$= 1.08 \times 10^{58} \text{ segundos}$$

$$= 3.42 \times 10^{50} \text{ años}$$



Gradiente Descendente

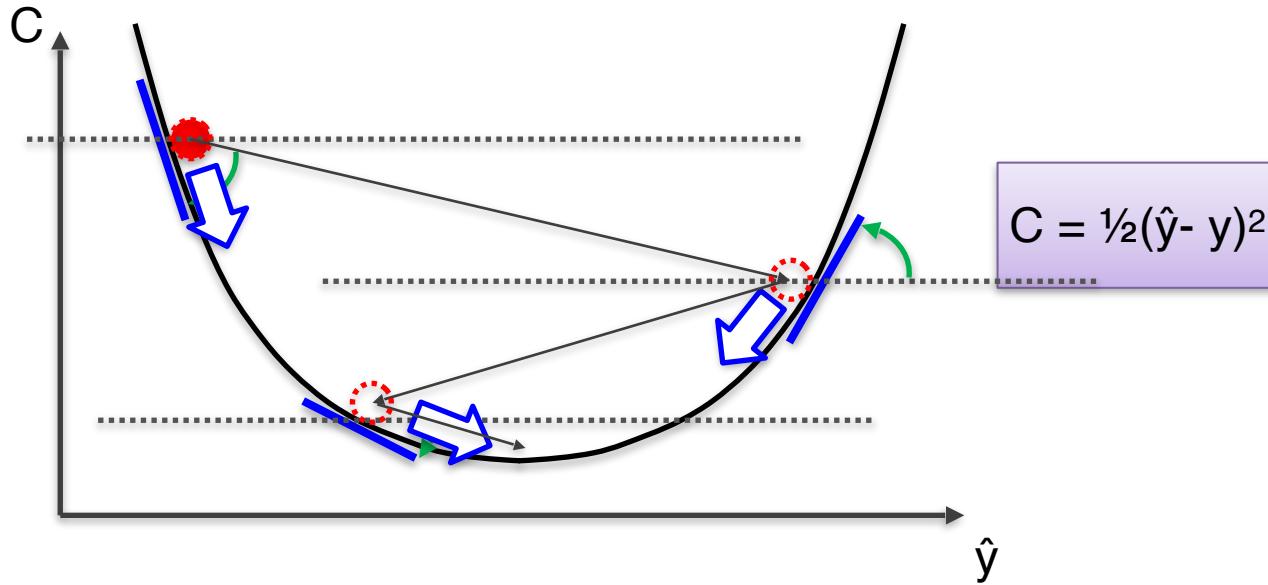


ImEdad Source: neuralnetworksanddeeplearning.com

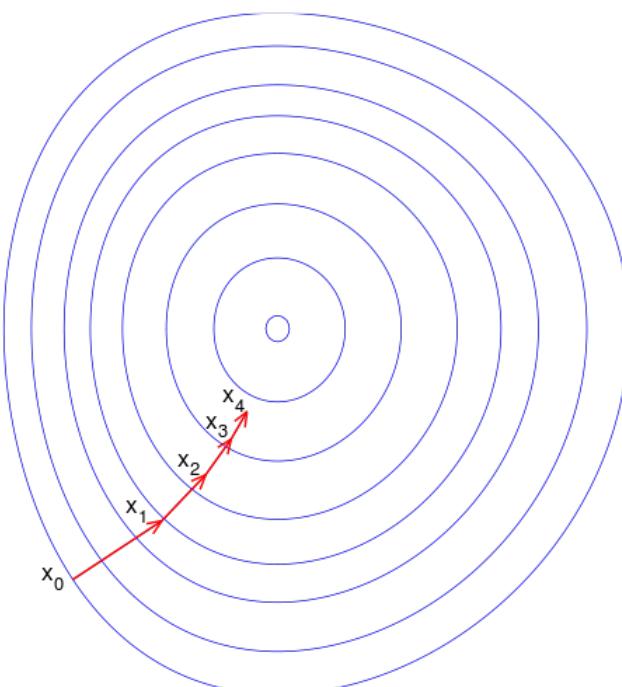
Gradiente Descendente

Gradiente Descendente

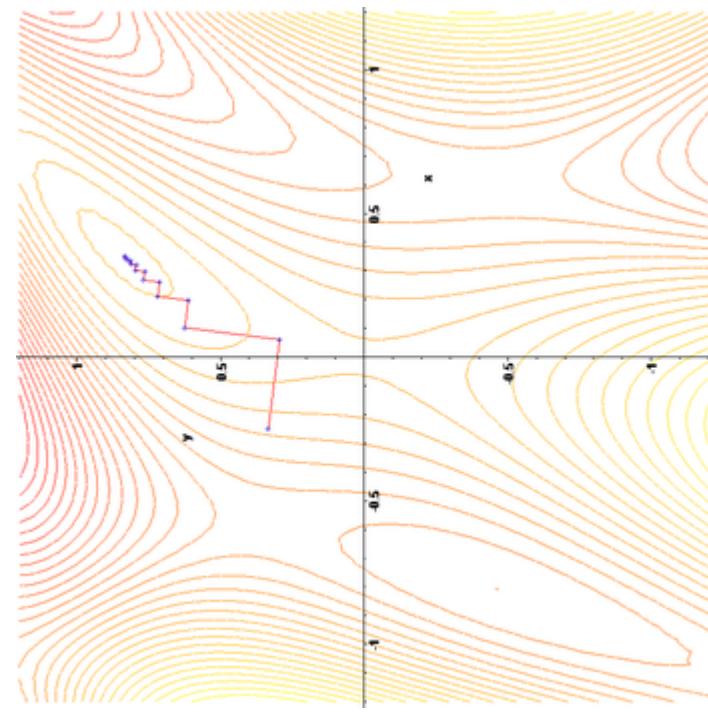
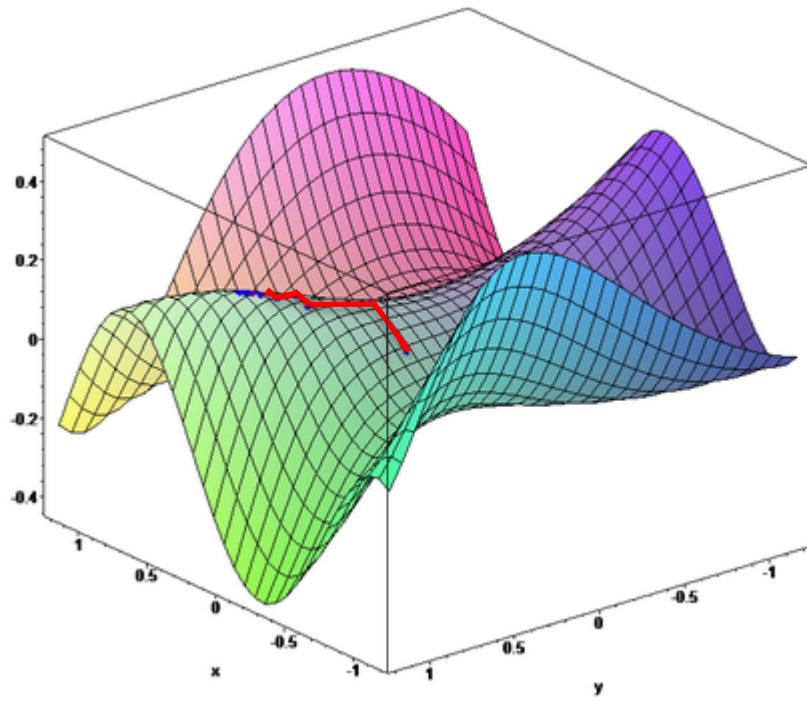
Gradiente Descendente



Gradiente Descendente

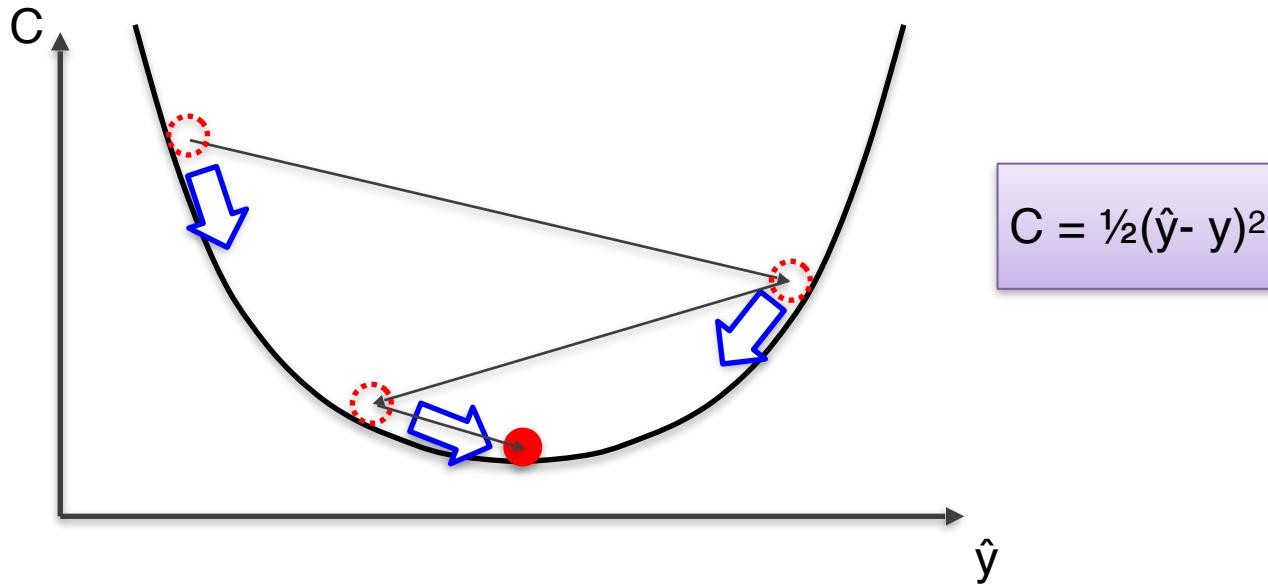


Gradiente Descendente

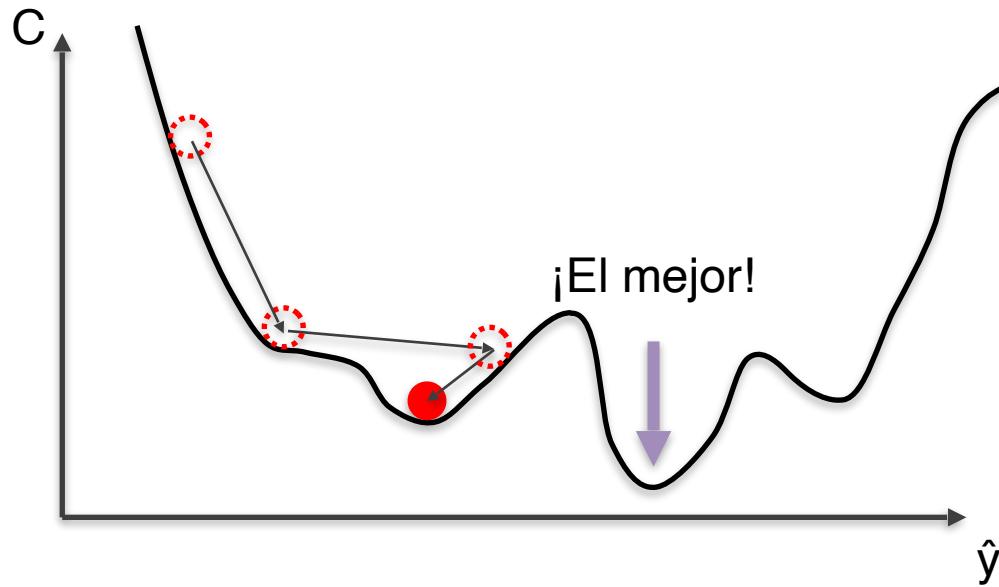


Gradiente Descendente Estocástico

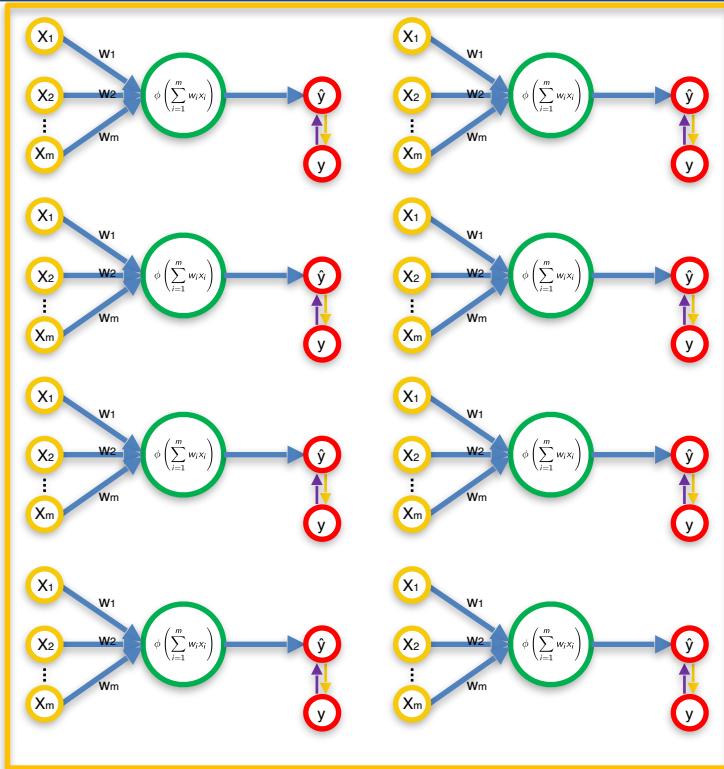
Gradiente Descendente Estocástico



Gradiente Descendente Estocástico

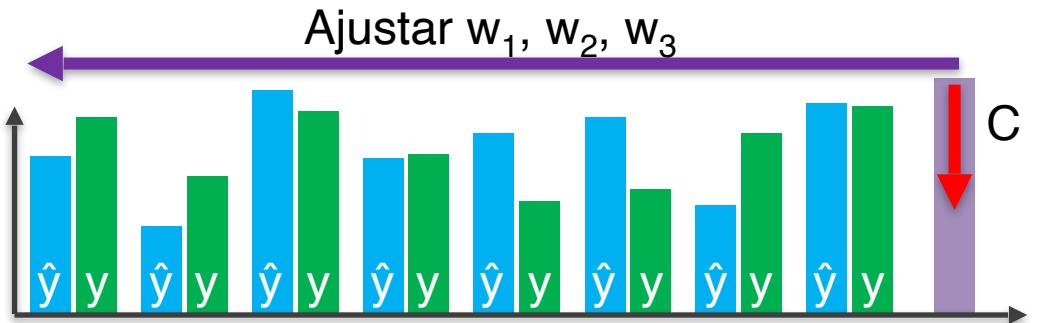


Gradiente Descendente Estocástico

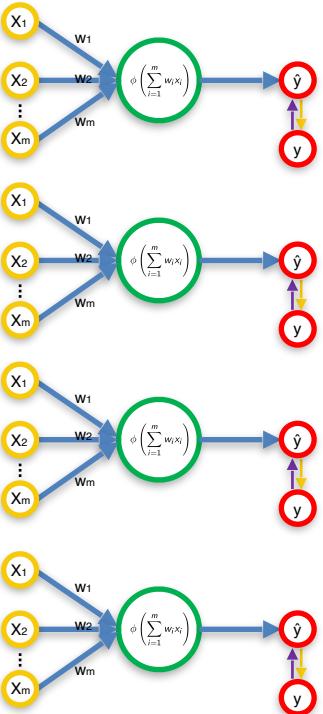
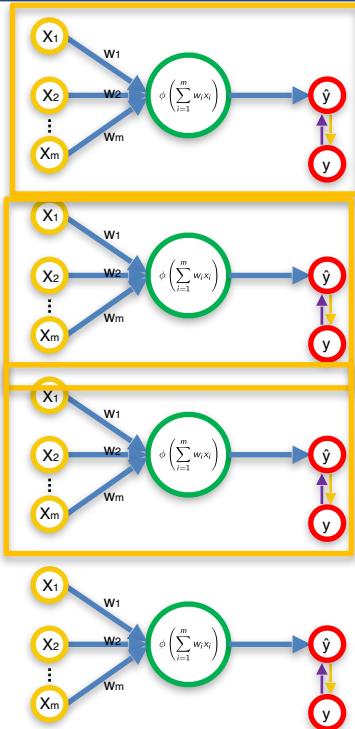


Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum \frac{1}{2}(\hat{y} - y)^2$$



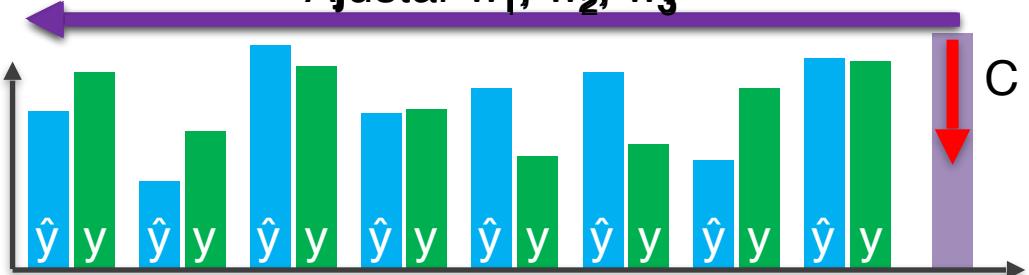
Gradiente Descendente Estocástico



Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

$$C = \sum \frac{1}{2}(\hat{y} - y)^2$$

Ajustar w_1, w_2, w_3



Gradiente Descendente Estocástico

Act w's

Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

Act w's

Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

Gradiente
Descendente
Por Bloques

Gradiente
Descendente
Estocástico

Gradiente Descendente Estocástico

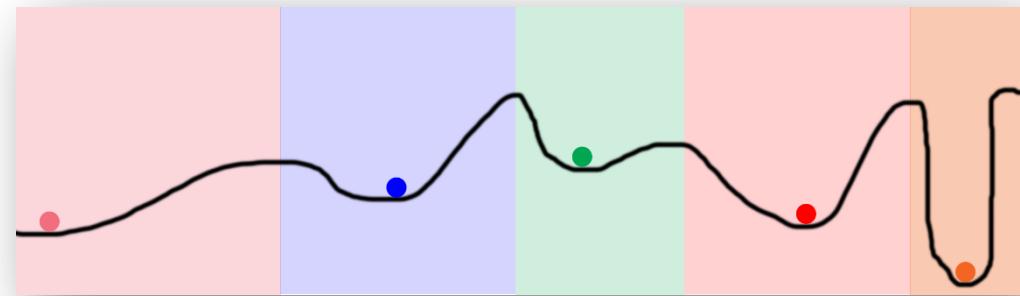
Lecturas Adicionales:

A Neural Network in 13 lines of Python (Part 2 - Gradiente Descendente)

Andrew Trask (2015)

Link:

<https://iamtrask.github.io/2015/07/27/python-network-part2/>



Gradiente Descendente Estocástico

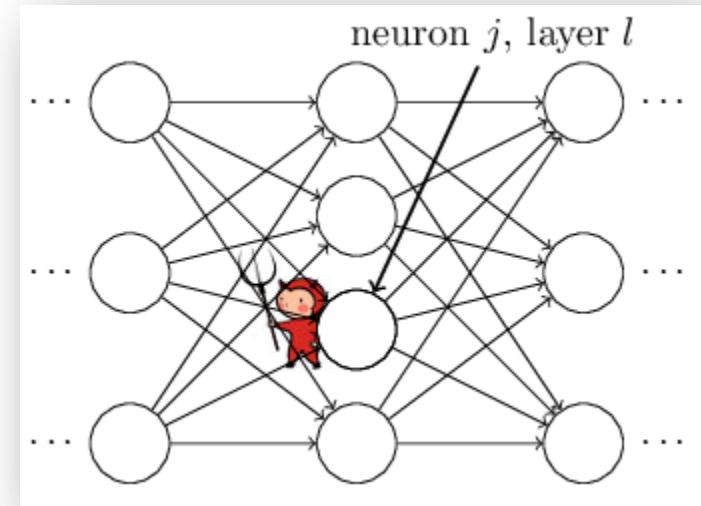
Lecturas Adicionales:

Neural Networks and Deep Learning

Michael Nielsen (2015)

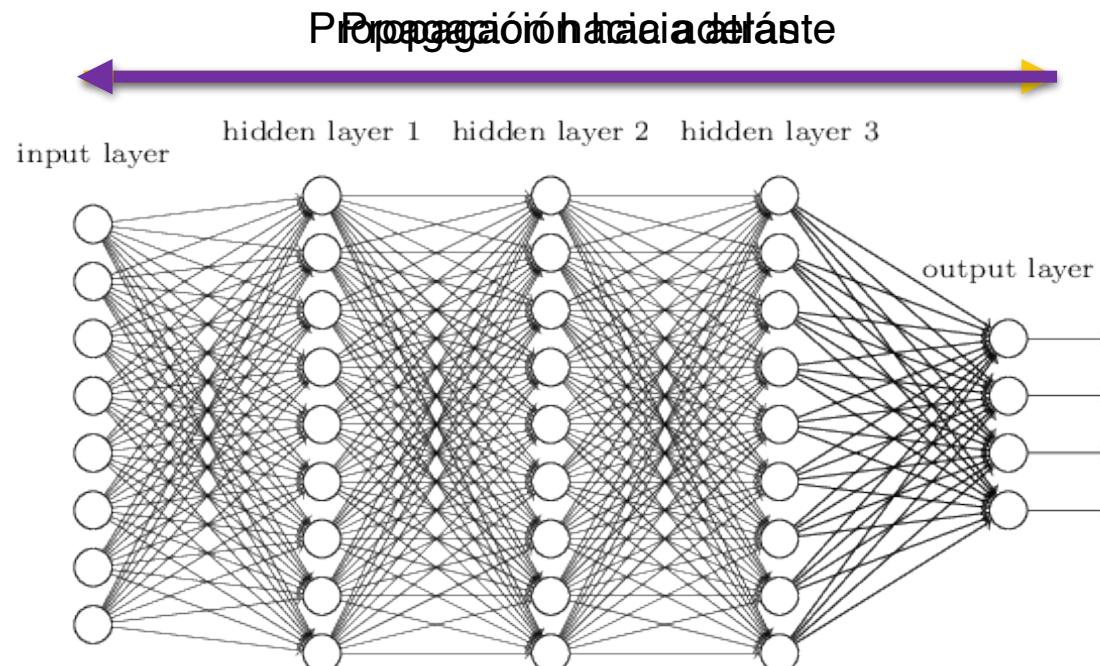
Link:

<http://neuralnetworksanddeeplearning.com/chap2.html>



Propagación hacia atrás

Gradiente Descendente



ImEdad Source: neuralnetworksanddeeplearning.com

Gradiente Descendente Estocástico

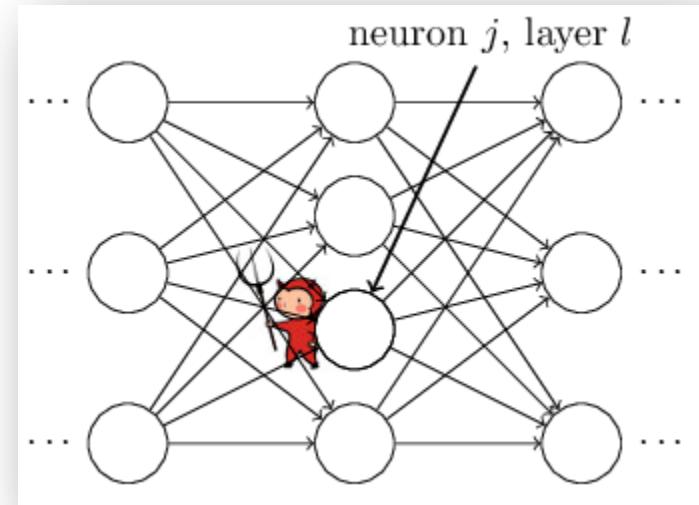
Lecturas Adicionales:

Neural Networks and Deep Learning

Michael Nielsen (2015)

Link:

<http://neuralnetworksanddeeplearning.com/chap2.html>



Entrenar la RNA con el Gradiente Descendente Estocástico

Paso 1: Inicializar los pesos aleatoriamente con valores cercanos a 0 (pero no 0).



Paso 2: Introducir la primera observación del dataset en la Capa de Entrada, cada característica es un nodo de entrada.



Paso 3: Propagación hacia adelante: de izquierda a derecha, las neuronas se activan de modo que la activación de cada una se limita por los pesos. Propaga las activaciones hasta obtener la predicción y .



Paso 4: Comparamos la predicción con el resultado real. Se mide entonces el error generado.



Paso 5: Propagación hacia atrás: de derecha a izquierda, propagando el error hacia atrás. Se actualizan los pesos según lo responsables que Sean del error. El ratio de aprendizaje gobierna cuánto deben actualizarse los pesos.



Paso 6: Se repiten los Pasos 1 a 5 y se actualizan los pesos después de cada observación (Reinforcement Learning). O:



Se repiten los Pasos 1 a 5 pero actualiza los pesos después de un conjunto de observaciones (Batch Learning).

Paso 7: Cuando todo el Conjunto de Entrenamiento ha pasado por la RNA, se completa un epoch. Hacer más epochs.

Otras Cosas

Muy bien, ¿pero dónde está

Tres formas de afrontar el problema

1. Fuerza Bruta <- La Maldición de la Dimensión
2. Perturbaciones aleatorias a los pesos pesos (~evolución) – ineficiente. “Más eficiente con un factor del número de conexiones” G.Hinton
3. Propagación hacia atrás

Video de Geoffrey Hinton's:

<https://www.youtube.com/watch?v=xfPz92B0rv8>

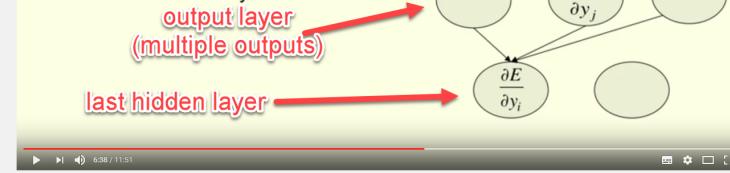
El fundamento de la propagación hacia atrás es tomar las derivadas del error de una capa y con ellas calcular las derivadas del error de la siguiente.

Por fin!!! ----->

Sin duda un algoritmo que toma un caso como dato de entrada y calcula – eficientemente – para cada peso de la red neuronal: cómo cambie el error con ese dato particular de entrada permitirá un cambio acorde a los pesos.

Sketch of the backpropagation algorithm on a single case

- First convert the discrepancy between each output and its target value into an error derivative.
- Then compute error derivatives in each hidden layer from error derivatives in the layer above.



$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j} \frac{\partial E}{\partial y_j} = y_j (1 - y_j) \frac{\partial E}{\partial y_j}$$

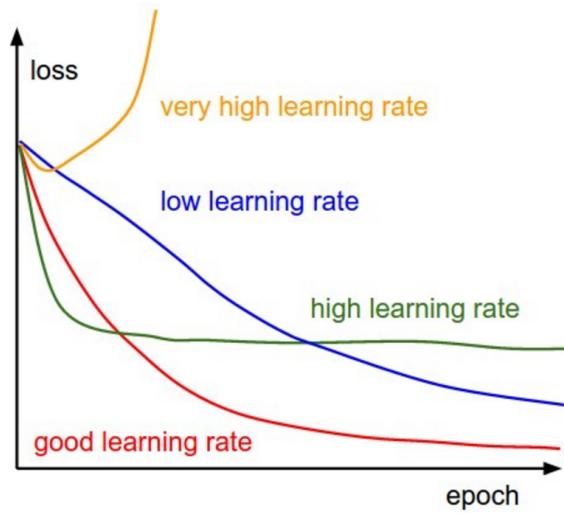
Chain rule

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

Gradient because derivative by one (of many) variables

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j}$$

Ratio de aprendizaje



Decay

Annealing the learning rate

In training deep networks, it is usually helpful to anneal the learning rate over time. Good intuition to have in mind is that with a high learning rate, the system contains too much kinetic energy and the parameter vector bounces around chaotically, unable to settle down into deeper, but narrower parts of the loss function. Knowing when to decay the learning rate can be tricky: Decay it slowly and you'll be wasting computation bouncing around chaotically with little improvement for a long time. But decay it too aggressively and the system will cool too quickly, unable to reach the best position it can. There are three common types of implementing the learning rate decay:

- **Step decay**: Reduce the learning rate by some factor every few epochs. Typical values might be reducing the learning rate by a half every 5 epochs, or by 0.1 every 20 epochs. These numbers depend heavily on the type of problem and the model. One heuristic you may see in practice is to watch the validation error while training with a fixed learning rate, and reduce the learning rate by a constant (e.g. 0.5) whenever the validation error stops improving.
- **Exponential decay**, has the mathematical form $\alpha = \alpha_0 e^{-kt}$, where α_0, k are hyperparameters and t is the iteration number (but you can also use units of epochs).
- **1/t decay** has the mathematical form $\alpha = \alpha_0 / (1 + kt)$ where α_0, k are hyperparameters and t is the iteration number.

In practice, we find that the step decay dropout is slightly preferable because the hyperparameters it involves (the fraction of decay and the step timings in units of epochs) are more interpretable than the hyperparameter k . Lastly, if you can afford the computational budget, err on the side of slower decay and train for a longer time.

Momentum

Momentum update is another approach that almost always enjoys better converge rates on deep networks. This update can be motivated from a physical perspective of the optimization problem. In particular, the loss can be interpreted as a the height of a hilly terrain (and therefore also to the potential energy since $U = mgh$ and therefore $U \propto h$). Initializing the parameters with random numbers is equivalent to setting a particle with zero initial velocity at some location. The optimization process can then be seen as equivalent to the process of simulating the parameter vector (i.e. a particle) as rolling on the landscape.

Since the force on the particle is related to the gradient of potential energy (i.e. $F = -\nabla U$), the **force** felt by the particle is precisely the (negative) **gradient** of the loss function. Moreover, $F = ma$ so the (negative) gradient is in this view proportional to the acceleration of the particle. Note that this is different from the SGD update shown above, where the gradient directly integrates the position. Instead, the physics view suggests an update in which the gradient only directly influences the velocity, which in turn has an effect on the position:

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

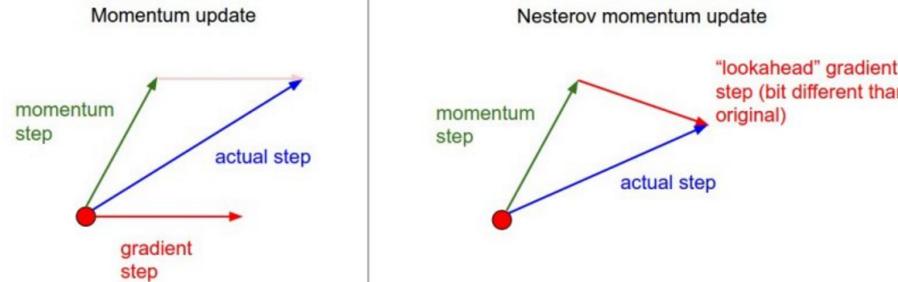
Here we see an introduction of a `v` variable that is initialized at zero, and an additional hyperparameter (`mu`). As an unfortunate misnomer, this variable is in optimization referred to as *momentum* (its typical value is about 0.9), but its physical meaning is more consistent with the coefficient of friction. Effectively, this variable damps the velocity and reduces the kinetic energy of the system, or otherwise the particle would never come to a stop at the bottom of a hill. When cross-validated, this parameter is usually set to values such as [0.5, 0.9, 0.95, 0.99]. Similar to annealing schedules for learning rates (discussed later, below), optimization can sometimes benefit a little from momentum schedules, where the momentum is increased in later stages of learning. A typical setting is to start with momentum of about 0.5 and anneal it to 0.99 or so over multiple epochs.

With Momentum update, the parameter vector will build up velocity in any direction that has consistent gradient.

Momentum

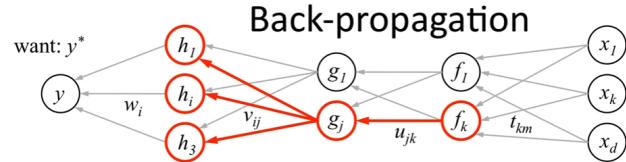
Nesterov Momentum is a slightly different version of the momentum update has recently been gaining popularity. It enjoys stronger theoretical converge guarantees for convex functions and in practice it also consistently works slightly better than standard momentum.

The core idea behind Nesterov momentum is that when the current parameter vector is at some position x , then looking at the momentum update above, we know that the momentum term alone (i.e. ignoring the second term with the gradient) is about to nudge the parameter vector by $\mu * v$. Therefore, if we are about to compute the gradient, we can treat the future approximate position $x + \mu * v$ as a "lookahead" - this is a point in the vicinity of where we are soon going to end up. Hence, it makes sense to compute the gradient at $x + \mu * v$ instead of at the "old/stale" position x .



Nesterov momentum. Instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this "looked-ahead" position.

Propagación hacia atrás



1. receive new observation $x = [x_1 \dots x_d]$ and target y^*
2. **feed forward:** for each unit g_j in each layer $1 \dots L$
compute g_j based on units f_k from previous layer: $g_j = \sigma\left(u_{j0} + \sum_k u_{jk} f_k\right)$
3. get prediction y and error $(y - y^*)$
4. **back-propagate error:** for each unit g_j in each layer $L \dots 1$

(a) compute error on g_j

$$\frac{\partial E}{\partial g_j} = \sum_i \sigma'(h_i) v_{ji} \frac{\partial E}{\partial h_i}$$

should g_j be higher or lower?
how h_i will change as g_j changes
too high or too low?

(b) for each u_{jk} that affects g_j

(i) compute error on u_{jk}

$$\frac{\partial E}{\partial u_{jk}} = \frac{\partial E}{\partial g_j} \sigma'(g_j) f_k$$

do we want g_j to be higher/lower
how g_j will change if u_{jk} is higher/lower

(ii) update the weight

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

Copyright © 2014 Victor Laverenko

<https://www.youtube.com/watch?v=An5z8lR8asY>

El video es un inicio pero todavía no entiendo como se decide h (h_1, h_2, h_3) para ajustar los pesos de la red neuronal...

Otro vídeo adicional (con fundamentos de cálculo):
<https://www.youtube.com/watch?v=mgceQli6ZKQ>

Propagación hacia a

Tres formas de afrontar el problema

1. Fuerza Bruta <- La Maldición de la Dimensión
2. Perturbaciones aleatorias a los pesos pesos (~evolución) – ineficiente. “Más eficiente con un factor del número de conexiones” G.Hinton
3. Propagación hacia atrás

Video de Geoffrey Hinton's:

<https://www.youtube.com/watch?v=xfPz92B0rv8>

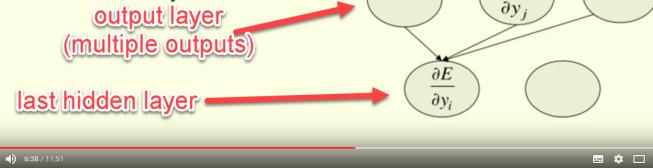
El fundamento de la propagación hacia atrás es tomar las derivadas del error de una capa y con ellas calcular las derivadas del error de la siguiente.

Por fin!!! ----->

Sin duda un algoritmo que toma un caso como dato de entrada y calcula – eficientemente – para cada peso de la red neuronal: cómo cambie el error con ese dato particular de entrada permitirá un cambio acorde a los pesos.

Sketch of the backpropagation algorithm on a single case

- First convert the discrepancy between each output and its target value into an error derivative.
- Then compute error derivatives in each hidden layer from error derivatives in the layer above.



Propagación hacia atrás (cont.)

https://www.youtube.com/watch?v=EInQoVLg_UY

~3min in

"The Propagación hacia atrás algorithm is doing what I just said [tweaking the pesos on a given input set] but it figures out for all the pesos at the same time, whether you should increase them a little bit or decrease them a little bit so that on that little batch of cases you took the answers get better"

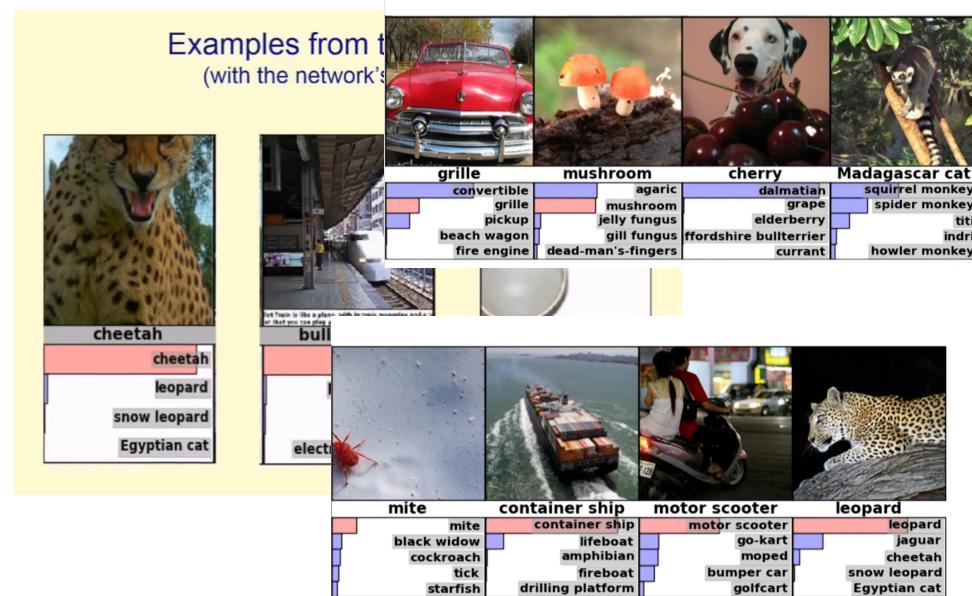
Hype in 1980s...

What was wrong in the 80s was – we didn't have enough data and we didn't have enough compute power.

1r "killer app": speech recognition. 2 students at his lab.... Now all of the speech recognition applications use some form of back propagation for training

2o: object recognition. Conventional computer vision systems in 2012 had an error of about 25%. An averEdad neural net got 16% error rate. Now the error rate is down to 4%.

Note to self: check out the 2012 ImEdadNet challenge.



Propagación hacia atrás (cont.)

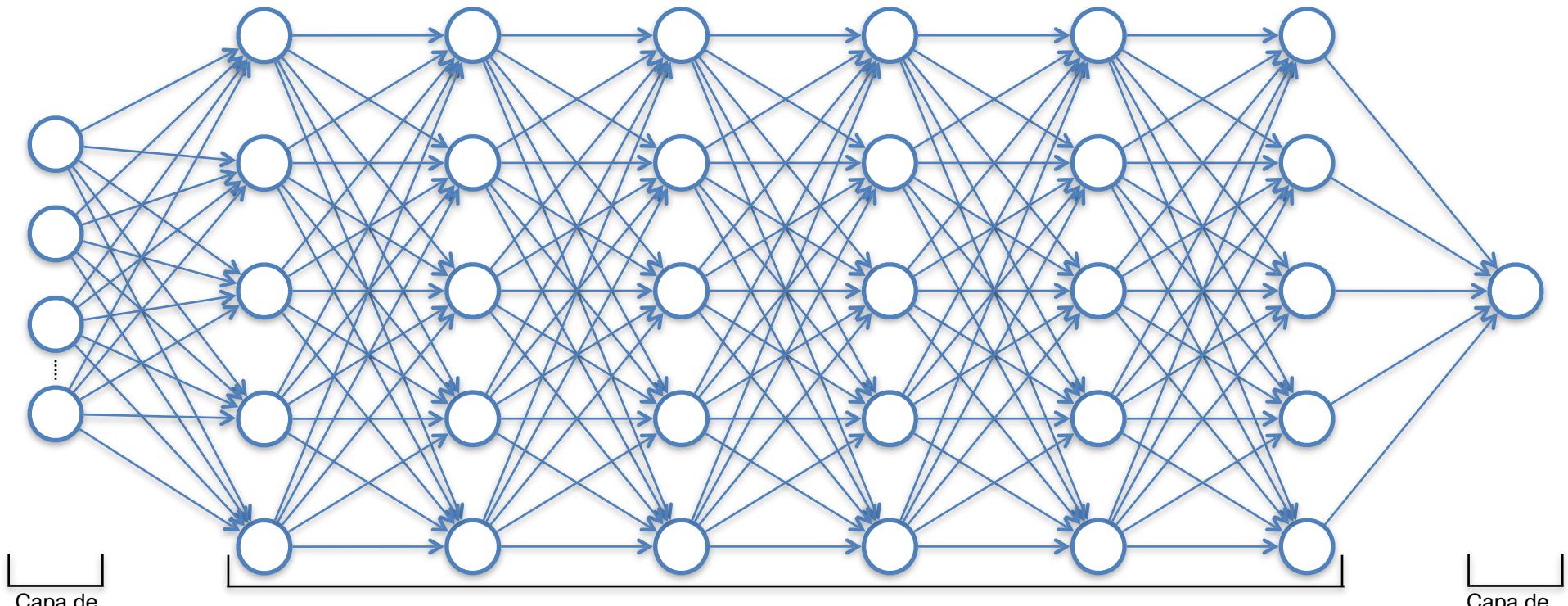
<https://www.youtube.com/watch?v=XG-dwZMc7Ng>

Aprender == cambiar los pesos (en la mente humana y en una red neuronal). Sin duda un pensamiento muy profundo y filosófico!

Vale, pero ¿dónde está el Deep Learning?

Deep Learning es aplicar los pasos 1 al 7 a todo esto...

Esto es Deep Learning



Capa de
Entrada

Capa Ocultas

Capa de
Salida

Matemáticas de la Propagación hacia atrás (avanzado)

Muy bien, ¿pero dónde está

Tres formas de afrontar el problema

1. Fuerza Bruta <- La Maldición de la Dimensión
2. Perturbaciones aleatorias a los pesos pesos (~evolución) – ineficiente. “Más eficiente con un factor del número de conexiones” G.Hinton
3. Propagación hacia atrás

Video de Geoffrey Hinton's:

<https://www.youtube.com/watch?v=xfPz92B0rv8>

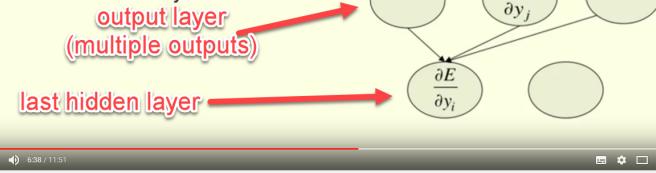
El fundamento de la propagación hacia atrás es tomar las derivadas del error de una capa y con ellas calcular las derivadas del error de la siguiente.

Por fin!!! ----->

Sin duda un algoritmo que toma un caso como dato de entrada y calcula – eficientemente – para cada peso de la red neuronal: cómo cambie el error con ese dato particular de entrada permitirá un cambio acorde a los pesos.

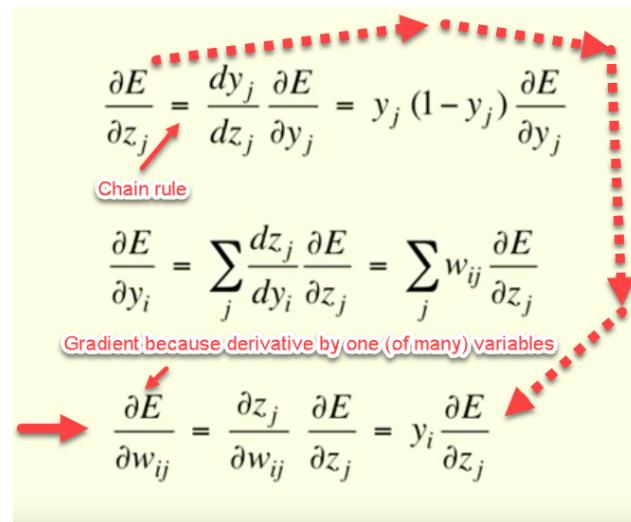
Sketch of the backpropagation algorithm on a single case

- First convert the discrepancy between each output and its target value into an error derivative.
- Then compute error derivatives in each hidden layer from error derivatives in the layer above.



$$E = \frac{1}{2} \sum_{j \in \text{output}} (t_j - y_j)^2$$

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$


$$\frac{\partial E}{\partial z_j} = \frac{dy_j}{dz_j} \frac{\partial E}{\partial y_j} = y_j (1 - y_j) \frac{\partial E}{\partial y_j}$$

Chain rule

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{dz_j}{dy_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

Gradient because derivative by one (of many) variables

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j}$$

BONUS

ANN

K-Fold cross validation

<https://www.youtube.com/watch?v=Tlgfjmp-4BA>
<https://www.youtube.com/watch?v=ADNFKiAjmWA>

Más en las transparencias

Fácil, ¿no?

Pero ¿cómo hacemos la “Media” después?

!!! Consigue maximizar la precisión de la predicción que lleva a cabo el algoritmo en global

k-fold CROSS VALIDATION

$k = 10$

20	20	20	1	20	20	20
20	20	20	20	20	20	20

Run k separate learning experiments

- pick testing set
- train
- test on testing set

Average test results from those k experiments

CROSS VALIDATION

TRAIN / TEST 10-fold C.V.

	min. training time	min. run time	max. accuracy
○	✗	○	○
○	○	✗	✗
○	○	○	○

no diff.

COMMENTS ABOUT SIZE

Mitigating over-fitting:

1. Add more data. > 10x Number of degr. Of freedom**
2. Regularization ???

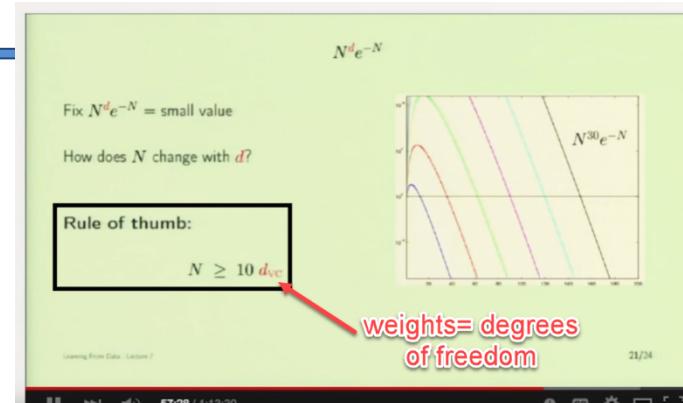
**But Geoffrey Hinton disagrees: “Statisticians (both frequentist and Bayesian) have misled us. I wanted to have hugely more parameters than training cases”

<https://www.youtube.com/watch?v=VIRCybGgHts>

Statistics and the brain

- Frequentist statisticians often tell you that you should have at least as many training examples as parameters.
 - Bayesian statisticians allow you to have less training examples than parameters, but only if you integrate over all possible settings of the parameters.
- The brain operates in a completely different regime from the one that most statisticians are familiar with.
 - It has about 10^{14} parameters and we only live for about 10^9 seconds.
 - Synapses are much cheaper than experiences so it makes sense to throw a lot of synapses at each experience.

Stanford University



(One of Welch's videos - he references somebody else's video)

But even statisticians will actually agree that more parameters is better – ensemble methods.. (same video around ~20 min in)

Also statiscitcians had to process data by hand. Now we have more power.

Also:

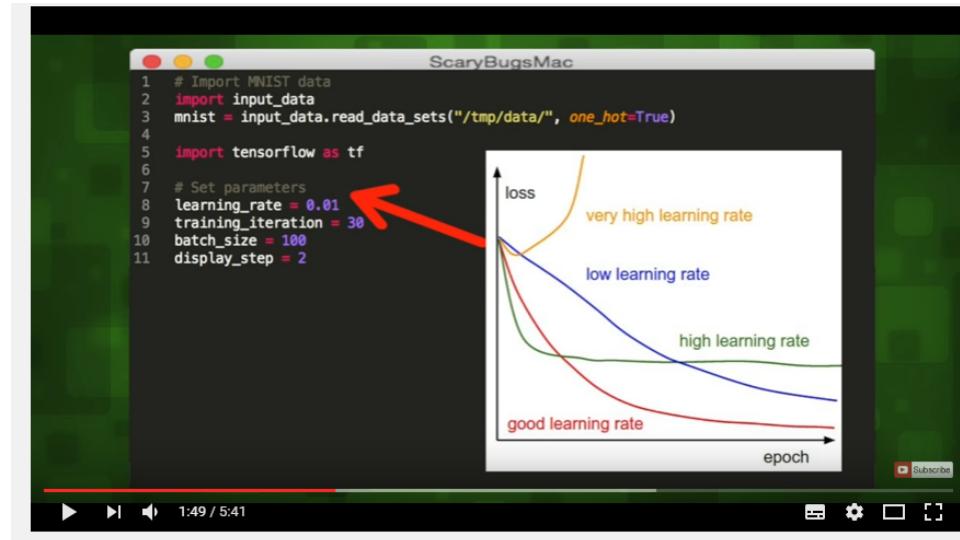
- The brain operates in a completely different regime from the one that most statisticians are familiar with.
 - It has about 10^{14} parameters and we only live for about 10^9 seconds.
 - Synapses are much cheaper than experiences so it makes sense to throw a lot of synapses at each experience.

Learning Rate

~1m40s

<https://www.youtube.com/watch?v=2FmcHiLCwTU>

Also see Hadelin's slide about this



Dropout training

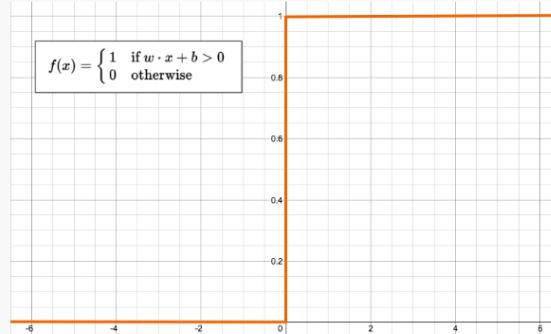
<https://visualstudiomagazine.com/articles/2014/05/01/neural-network-dropout-training.aspx>

Note – a good example of how it all works
is here:

<https://appliedgo.net/perceptron/>

The perceptron

The most basic form of an activation function is a simple binary function that has only two possible results.



Despite looking so simple, the function has a quite elaborate name: The **Heaviside Step function**. This function returns 1 if the input is positive or zero, and 0 for any negative input. A neuron whose activation function is a function like this is called a *perceptron*.



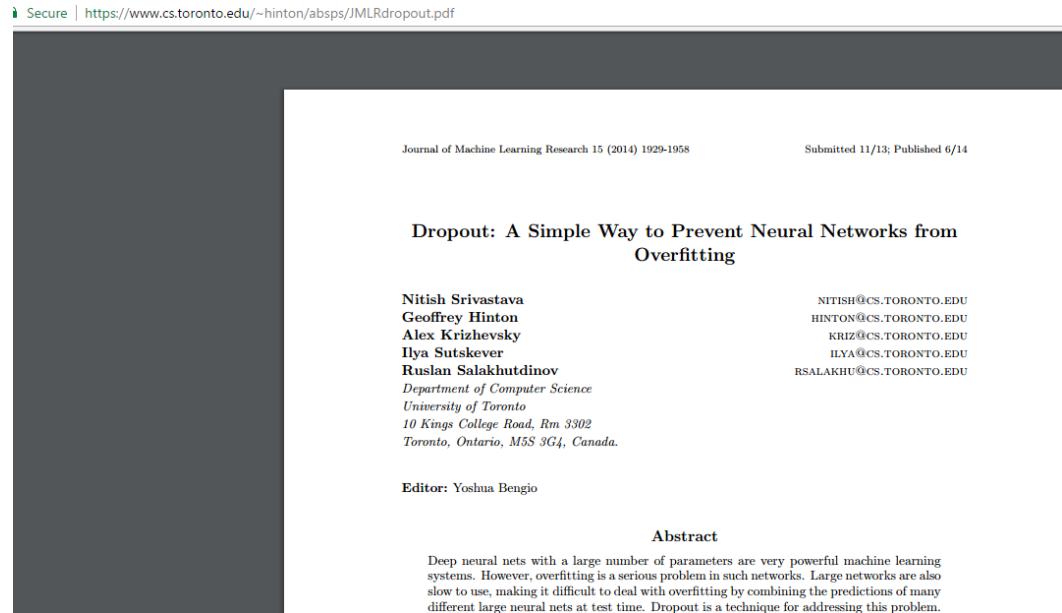
Dropout Intuition

Dropout

Dropout

~ Hinton's paper on dropout:

<https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf>



A screenshot of a web browser displaying the paper "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". The URL in the address bar is <https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf>. The page header shows "Secure" and the URL. The main content area has a large black rectangular redaction box covering most of the left side. On the right, the title "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" is centered above the authors' names: Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Below their names is their affiliation: Department of Computer Science, University of Toronto, 10 Kings College Road, Rm 3302, Toronto, Ontario, M5S 3G4, Canada. To the right of the authors' names are their email addresses: NITISH@CS.TORONTO.EDU, HINTON@CS.TORONTO.EDU, KRIZ@CS.TORONTO.EDU, ILYA@CS.TORONTO.EDU, and RSALAKHUI@CS.TORONTO.EDU. At the bottom left of the content area, it says "Editor: Yoshua Bengio". Below the abstract, there is a section titled "Abstract" with a paragraph of text.

Secure | <https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf>

Journal of Machine Learning Research 15 (2014) 1929-1958
Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

Nitish Srivastava
Geoffrey Hinton
Alex Krizhevsky
Ilya Sutskever
Ruslan Salakhutdinov
*Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto, Ontario, M5S 3G4, Canada.*

NITISH@CS.TORONTO.EDU
HINTON@CS.TORONTO.EDU
KRIZ@CS.TORONTO.EDU
ILYA@CS.TORONTO.EDU
RSALAKHUI@CS.TORONTO.EDU

Editor: Yoshua Bengio

Abstract
Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Large networks are also slow to use, making it difficult to deal with overfitting by combining the predictions of many different large neural nets at test time. Dropout is a technique for addressing this problem.

Idea del k-Fold Cross-Validation

k-Fold Cross Validation

k-Fold Cross Validation se usa para proporcionar una evaluación relevante a la eficacia de nuestro modelo

The Bias-Variance Tradeoff

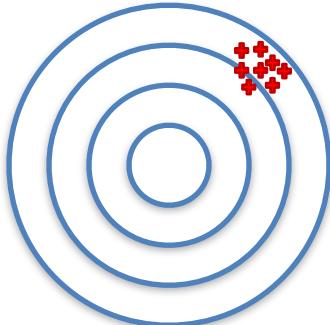
Low Bias: when your model predictions are close to the real values.

High Bias: when your model predictions are far from the real values.

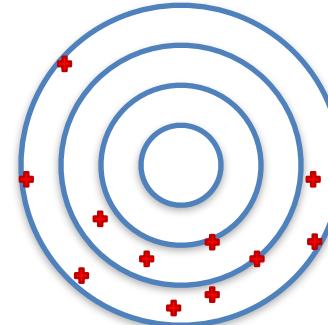
Low Variance: when you run your model several times, the different predictions won't vary much.

High Variance: when you run your model several times, the different predictions will vary a lot.

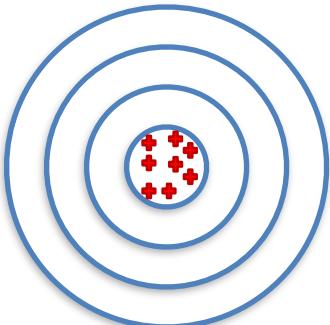
The Bias-Variance Tradeoff



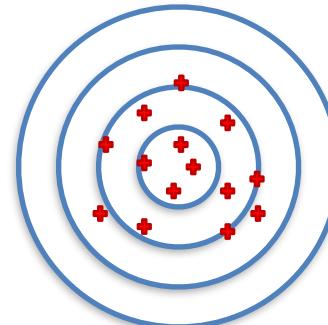
High Bias Low Variance



High Bias High Variance



Low Bias Low Variance



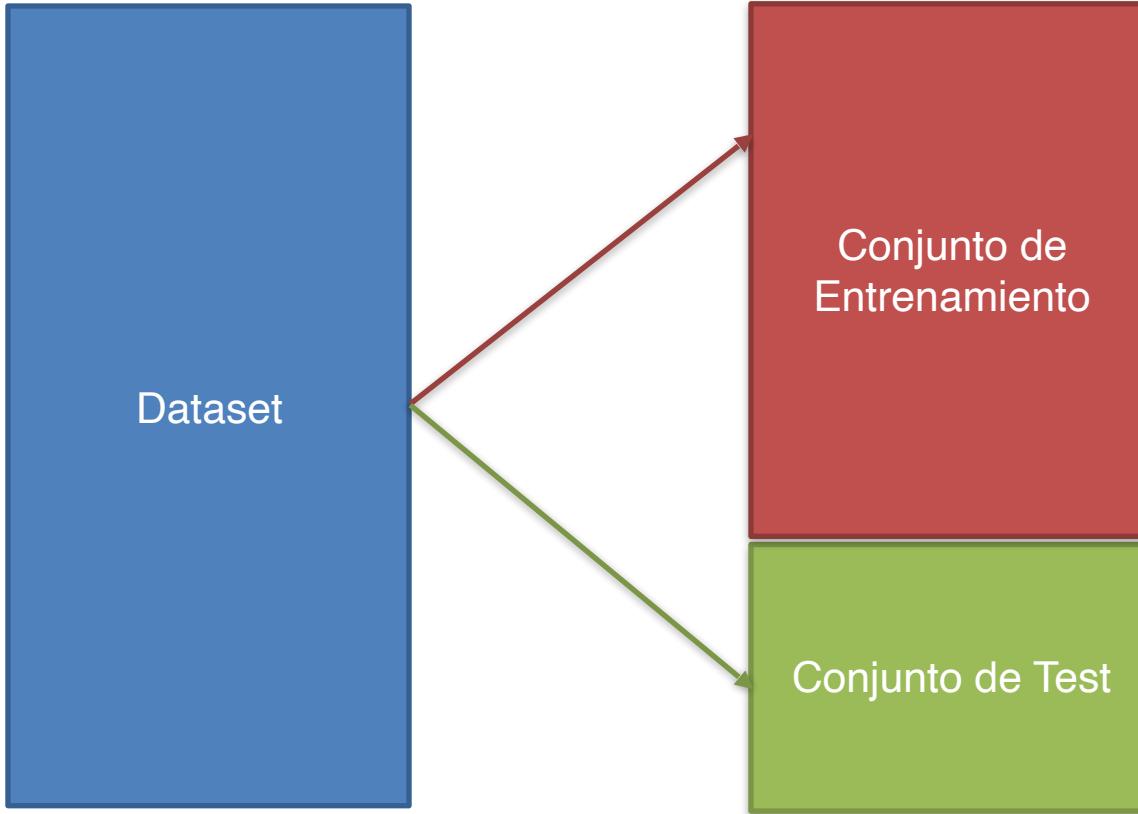
Low Bias High Variance

k-Fold Cross Validation

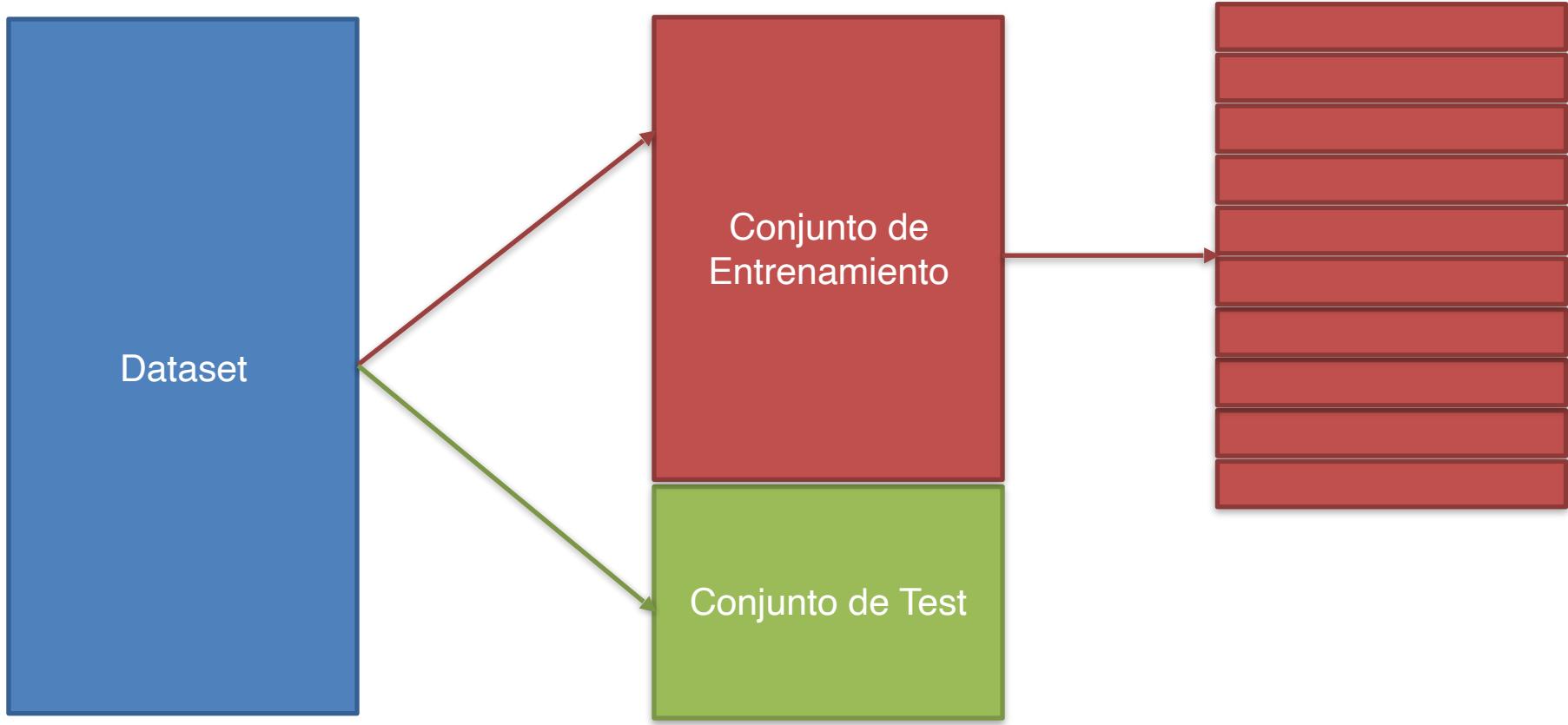


Dataset

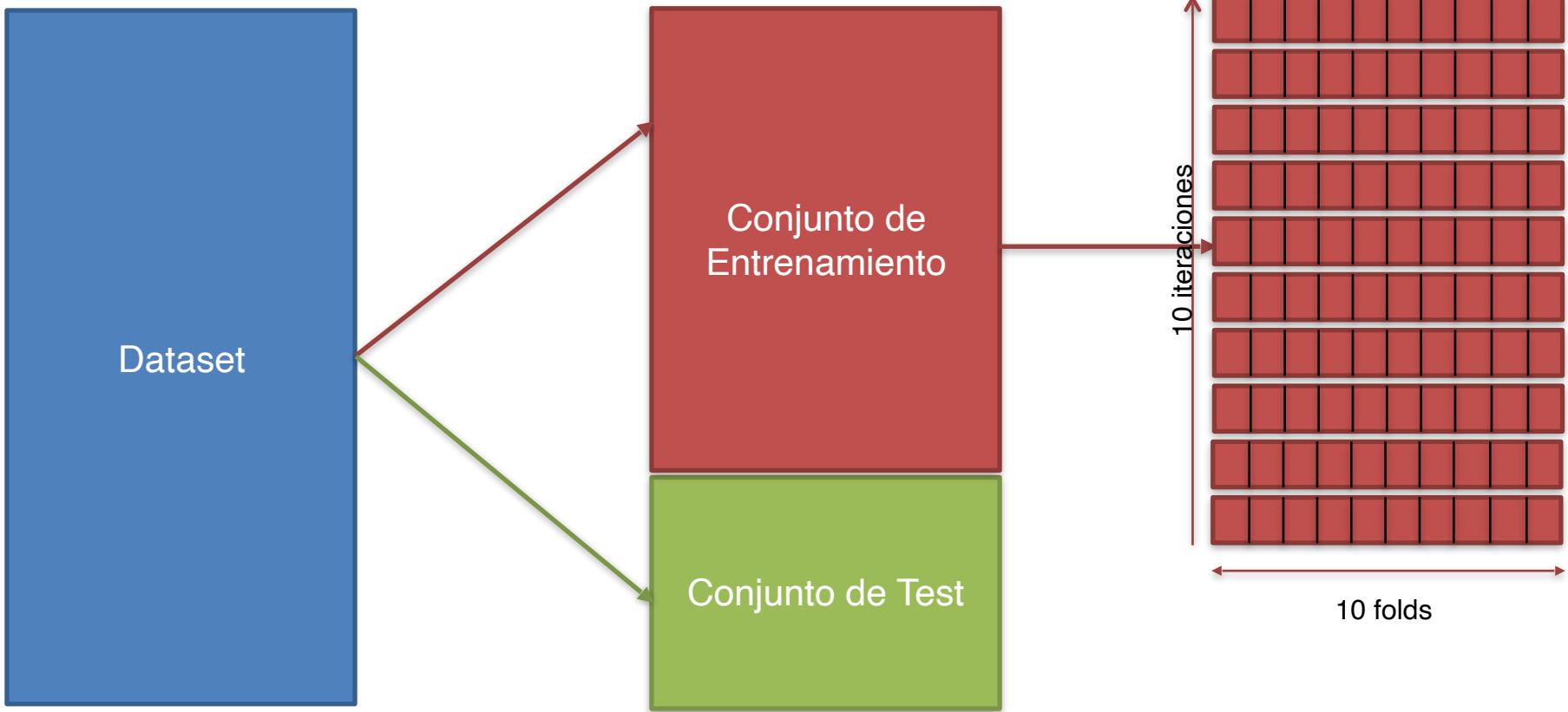
k-Fold Cross Validation



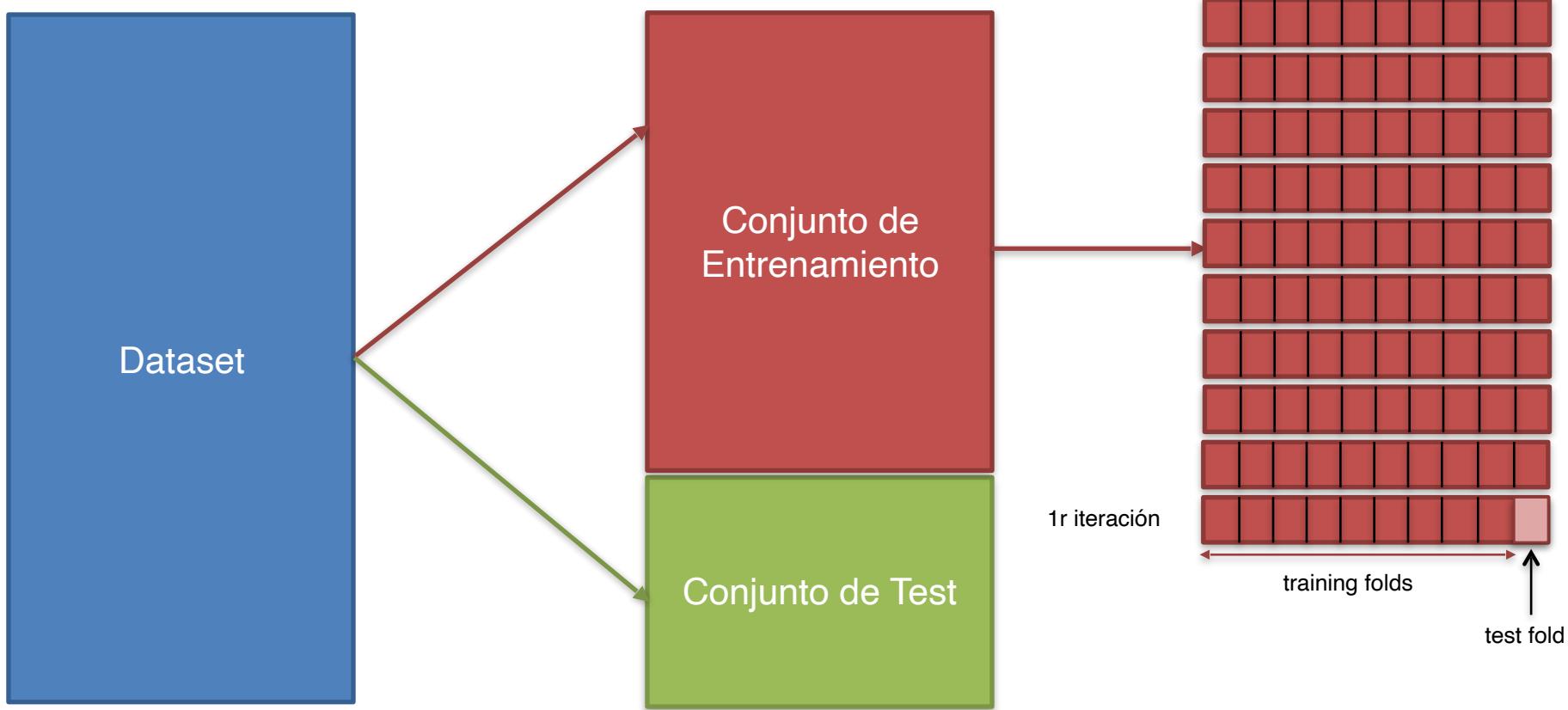
k-Fold Cross Validation



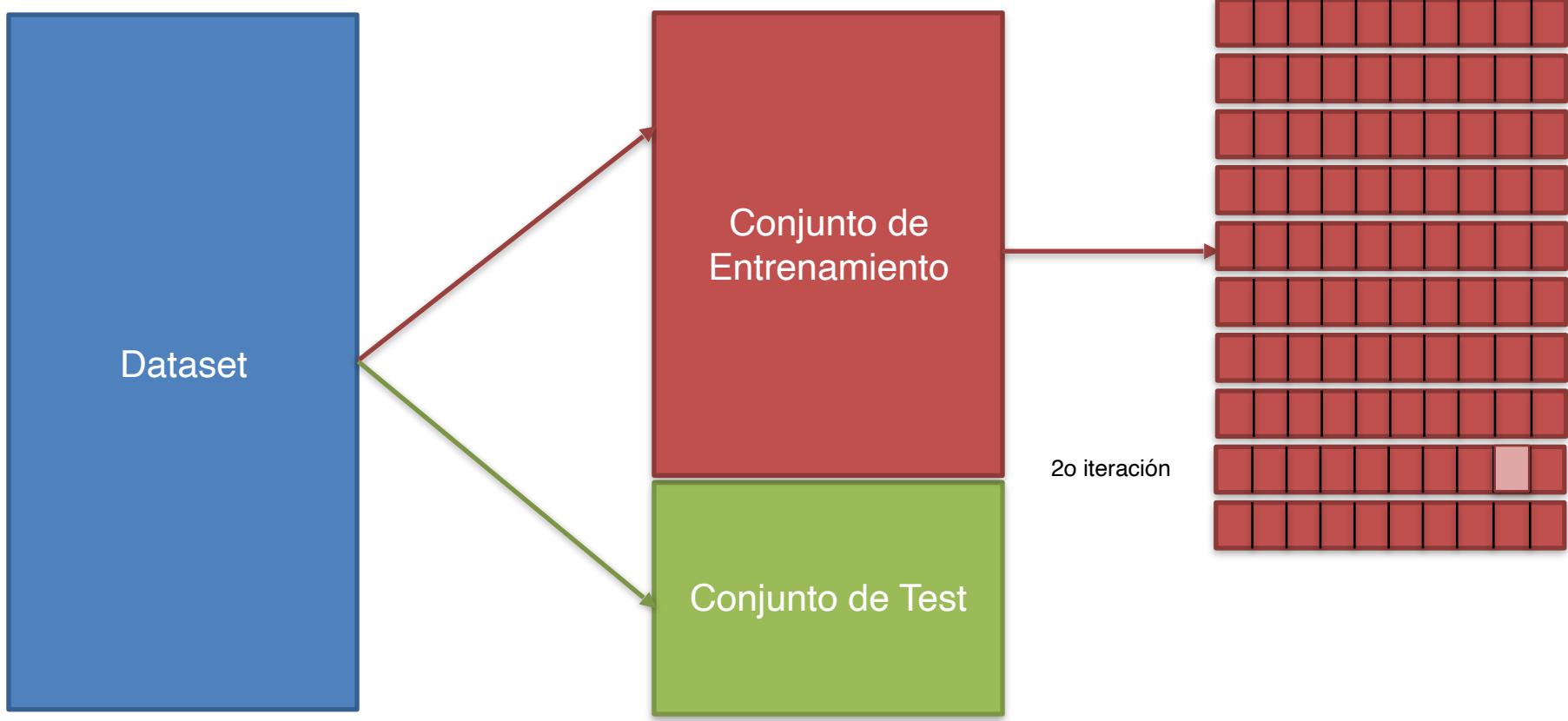
k-Fold Cross Validation



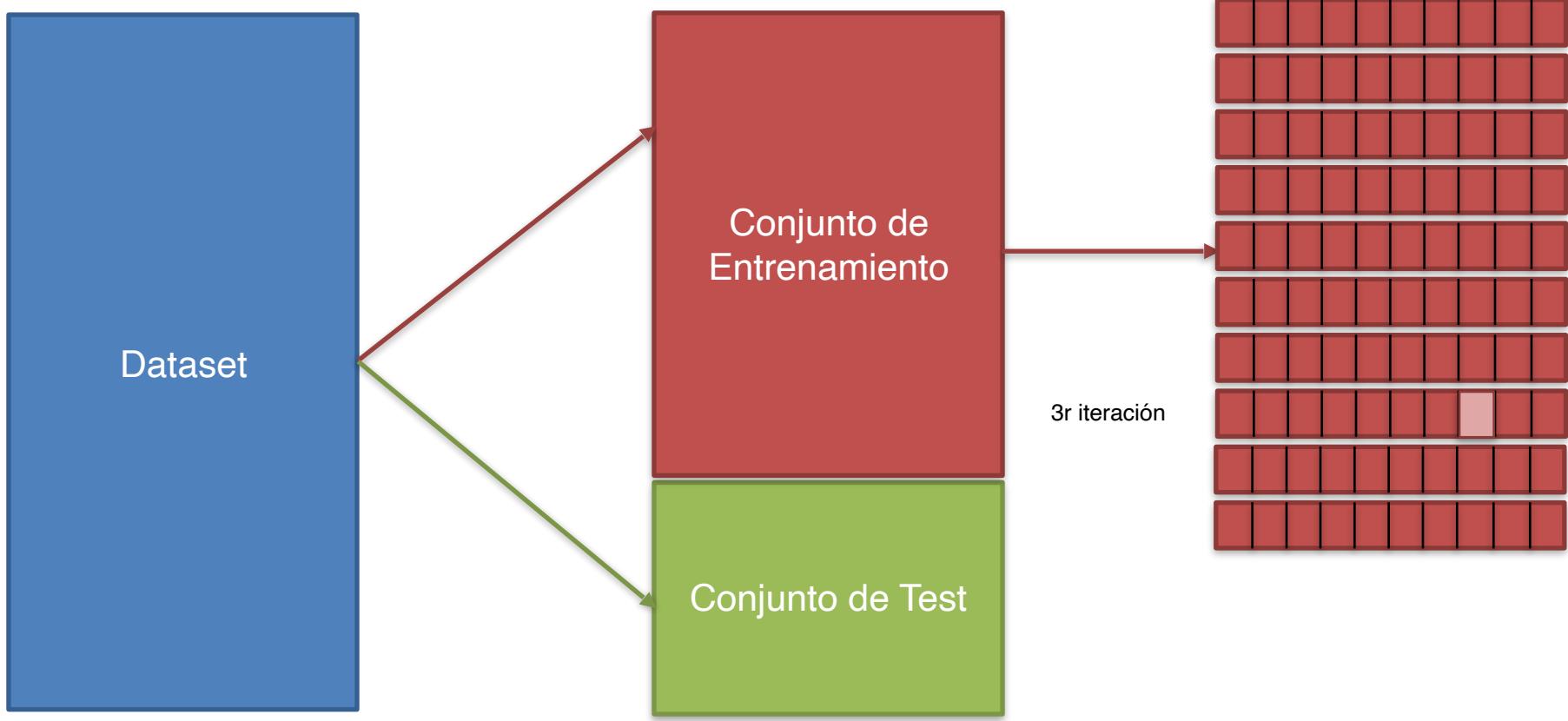
k-Fold Cross Validation



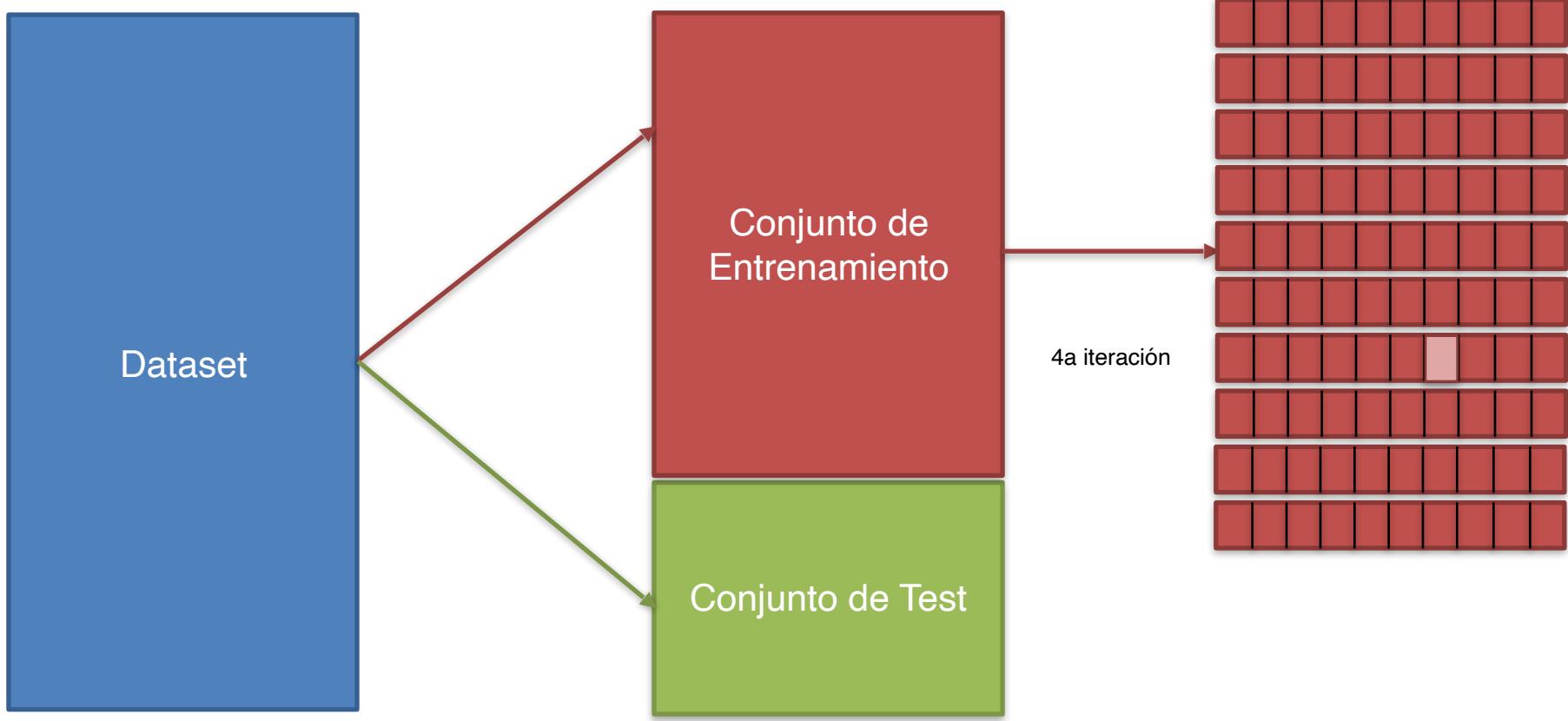
k-Fold Cross Validation



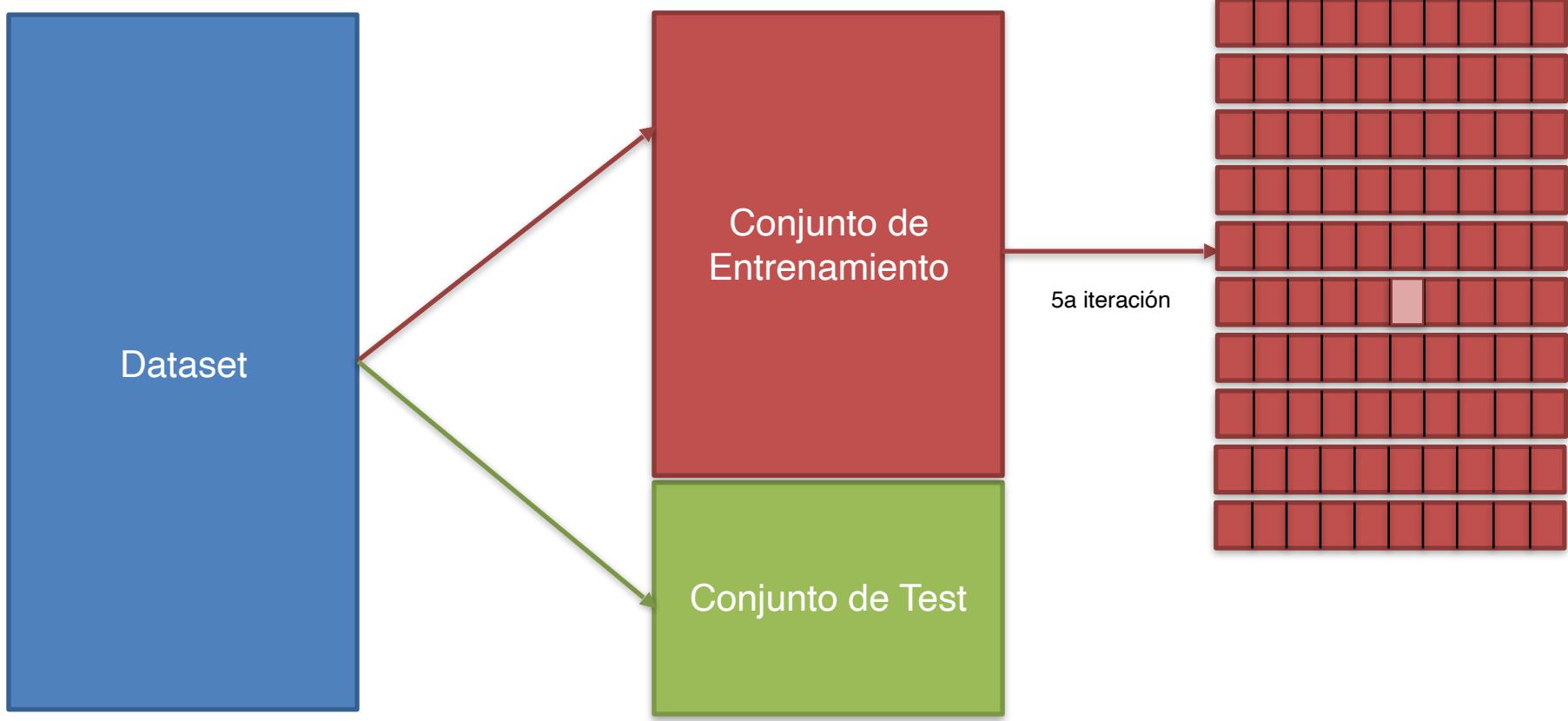
k-Fold Cross Validation



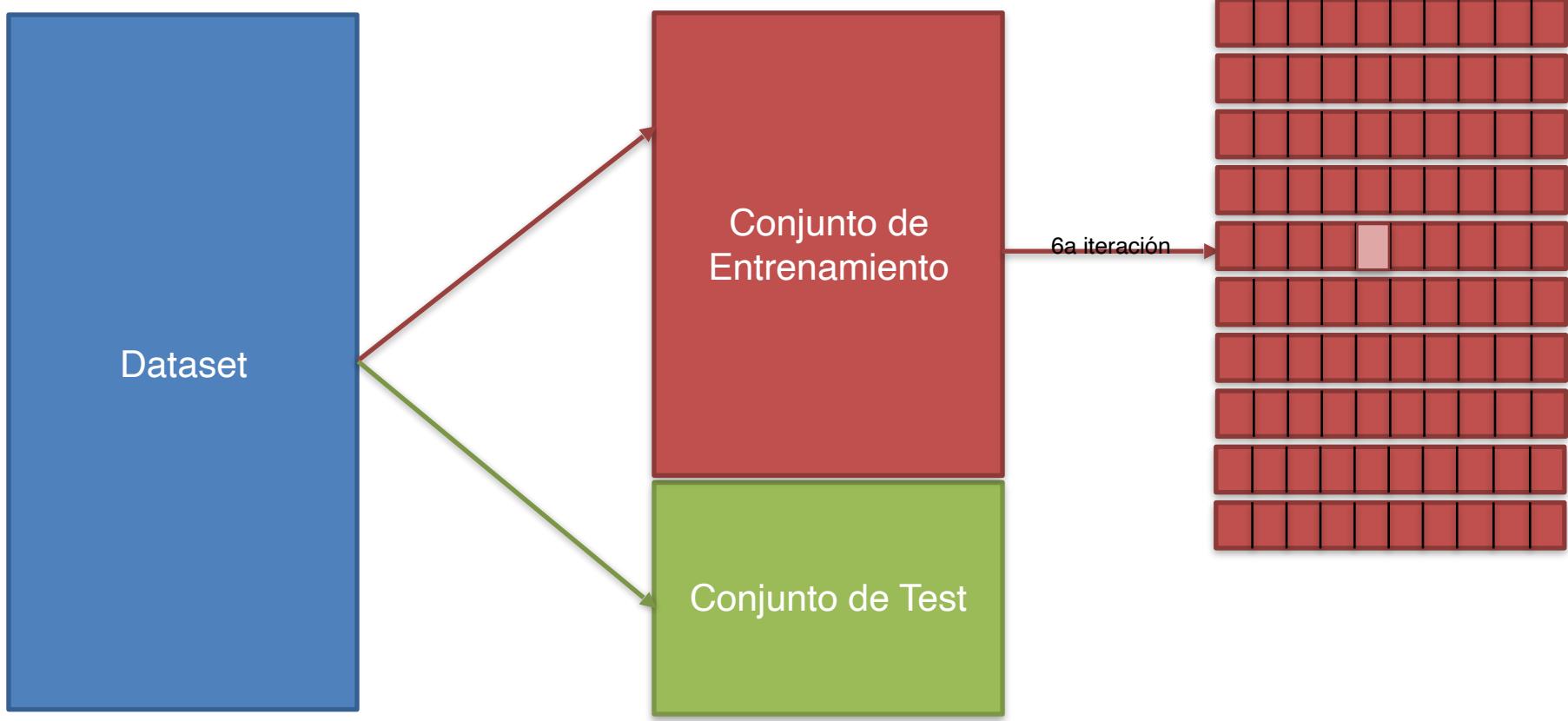
k-Fold Cross Validation



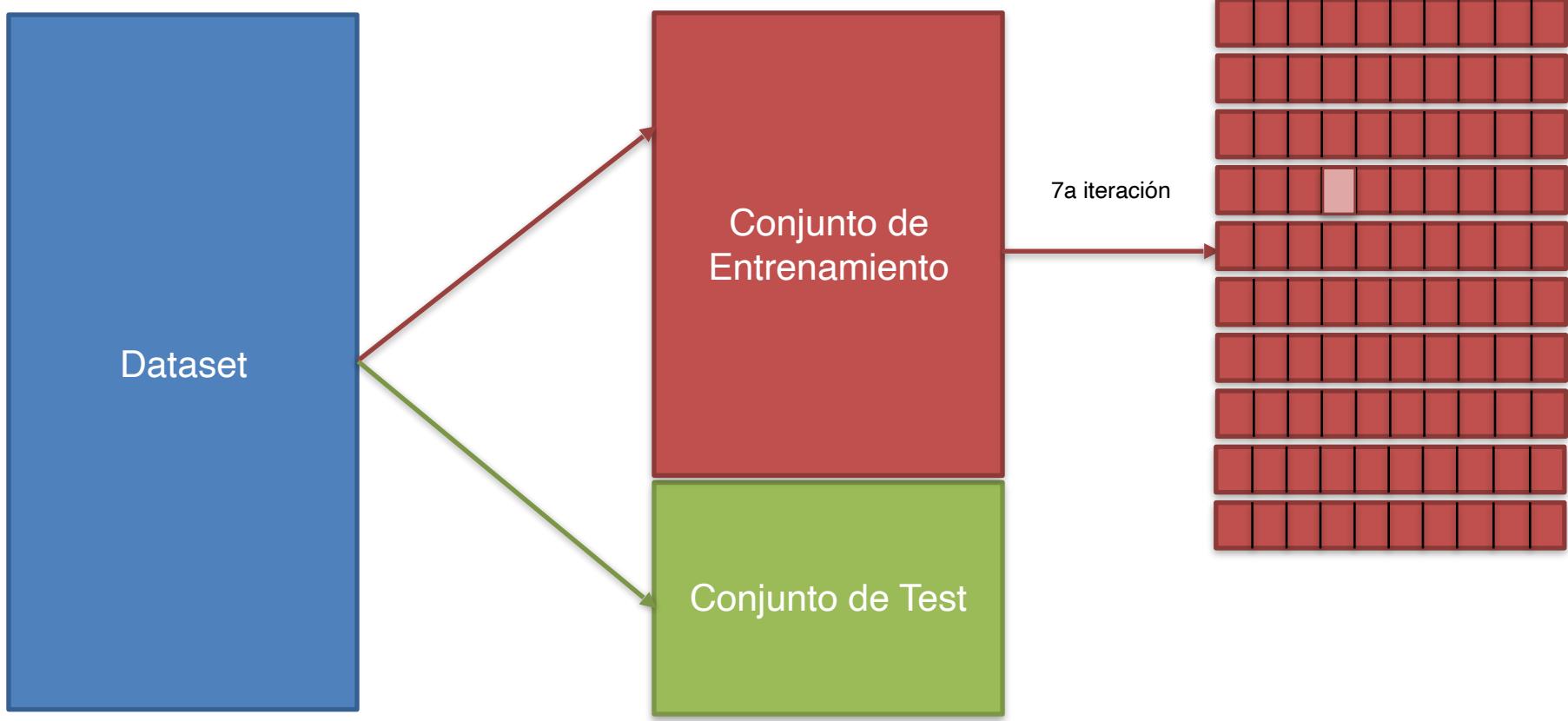
k-Fold Cross Validation



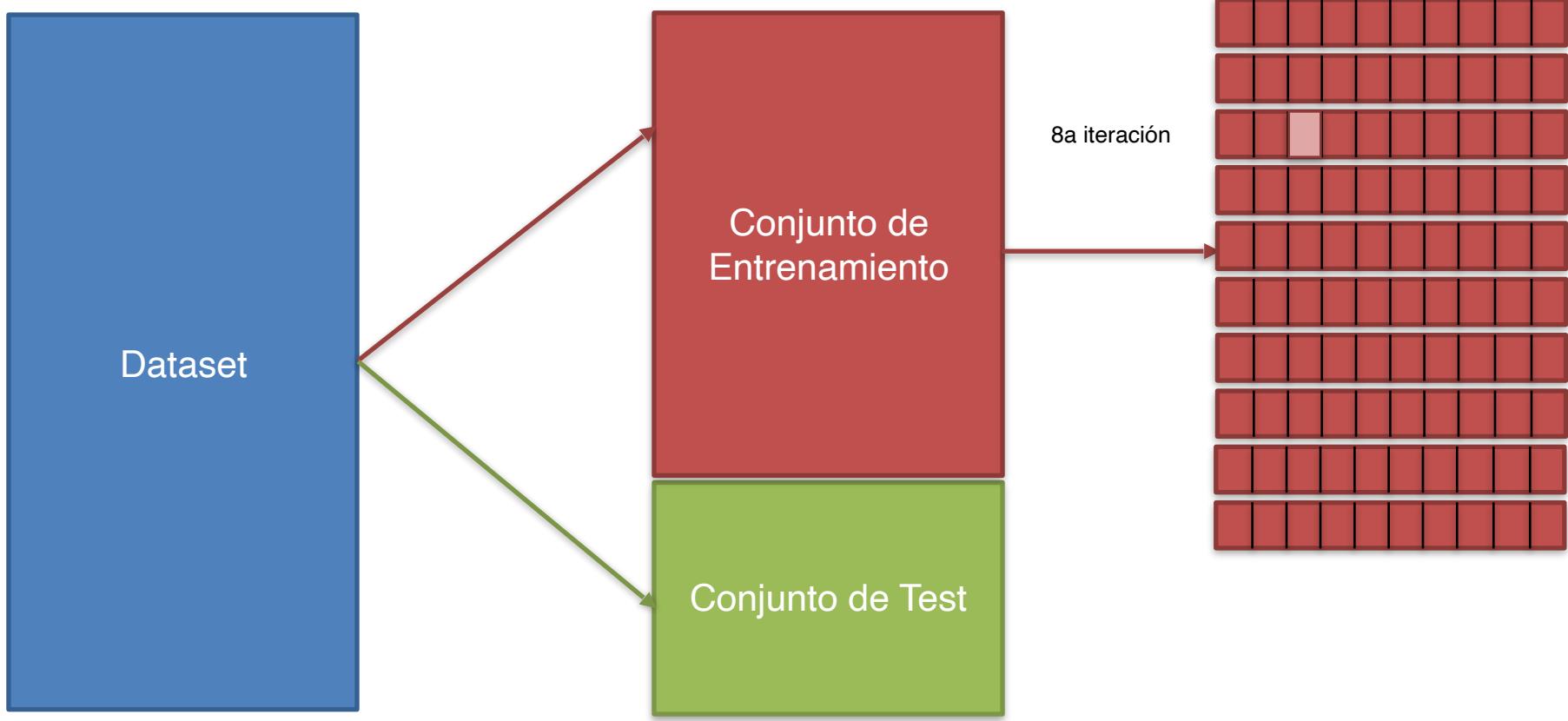
k-Fold Cross Validation



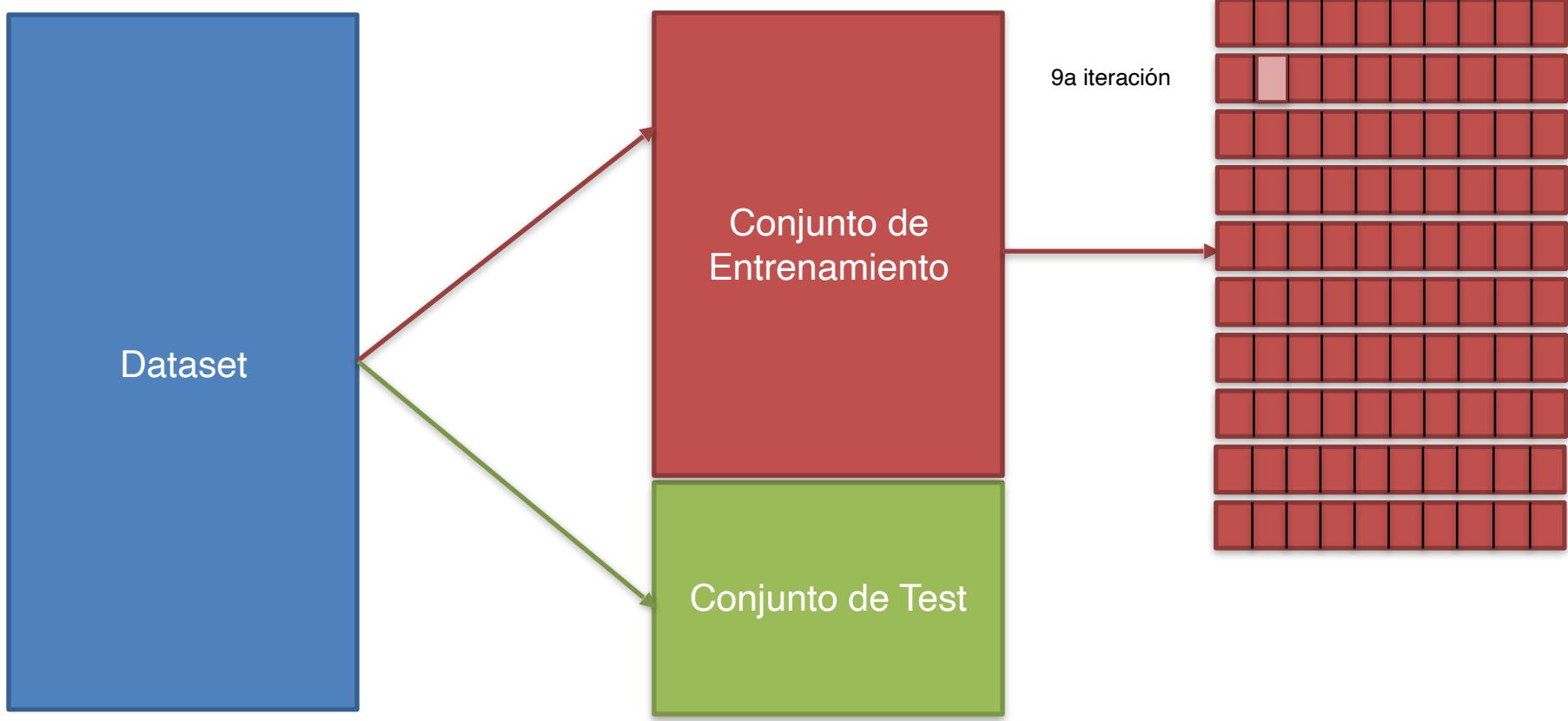
k-Fold Cross Validation



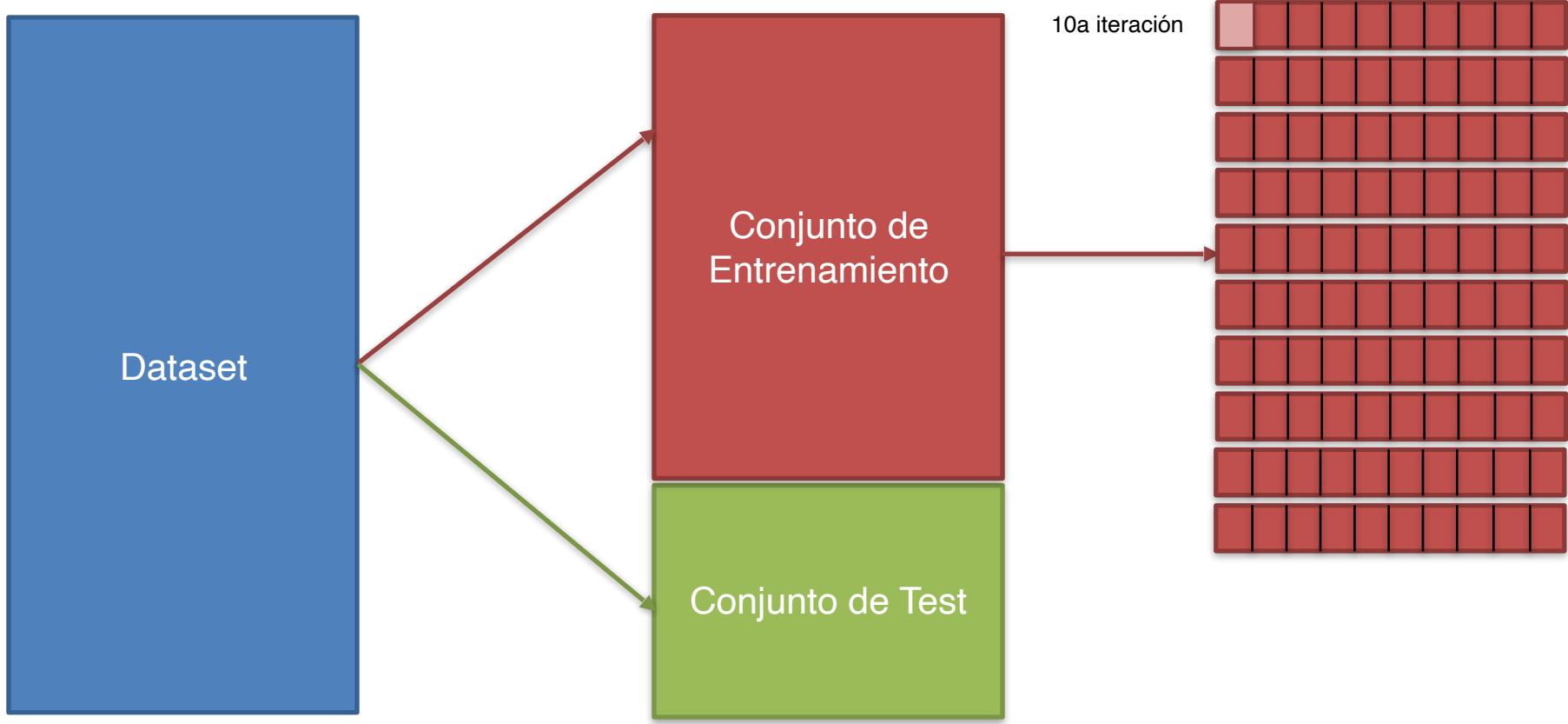
k-Fold Cross Validation



k-Fold Cross Validation



k-Fold Cross Validation



Idea de la Búsqueda en Parrilla

Búsqueda en Parrilla

Grid Search es una técnica para encontrar los mejores hiperparámetros para nuestro modelo