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?

1956

1980

2017

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

$149.99 ~~$199.99~~ Prime
Wednesday, Mar 22
hipping on eligible

hoices
$147.99 (10 new offers)

21,224

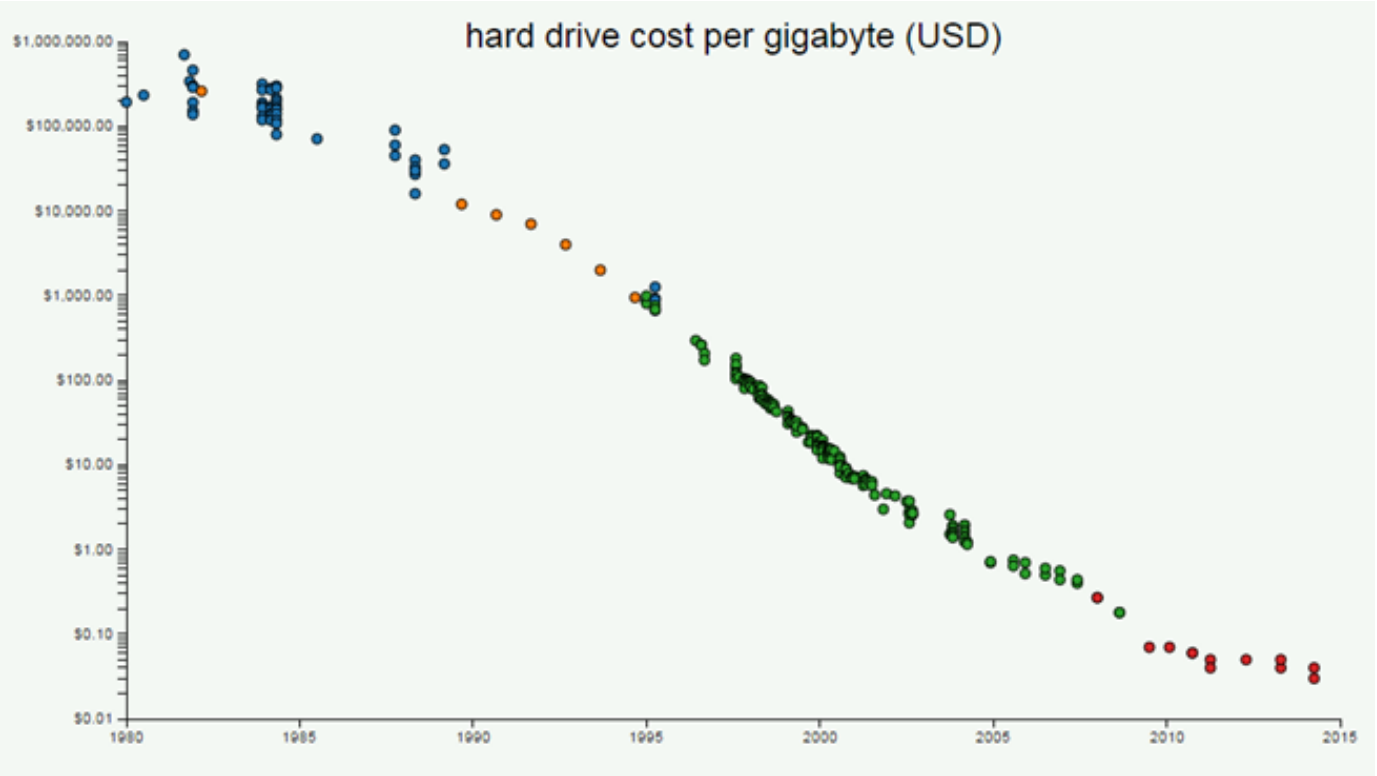
Product Features
256GB capacity
breakthrough

2x

25,600x

¿Qué es Deep Learning?

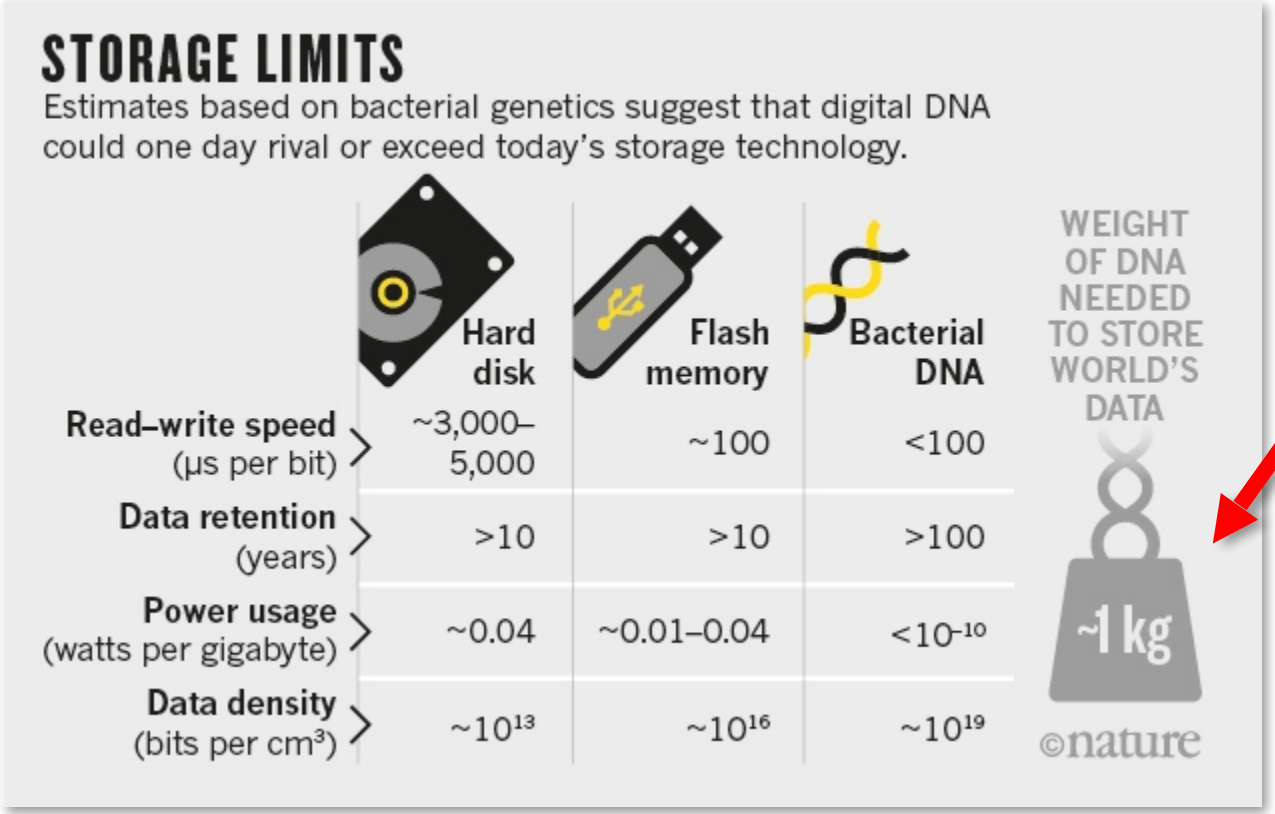
Escala Logarítmica



Source: mkomo.com

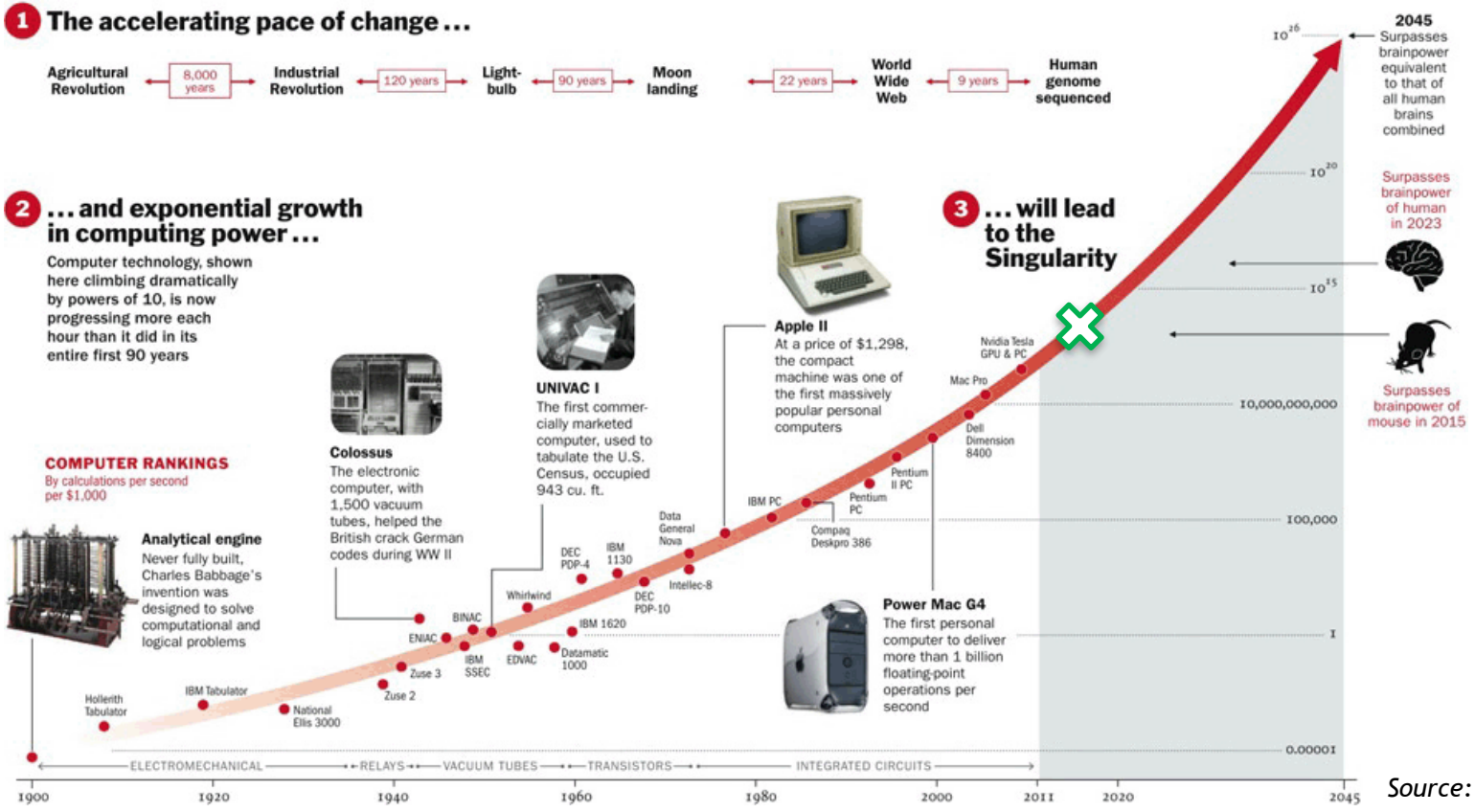
¿Qué es Deep Learning?

¿Qué es Deep Learning?

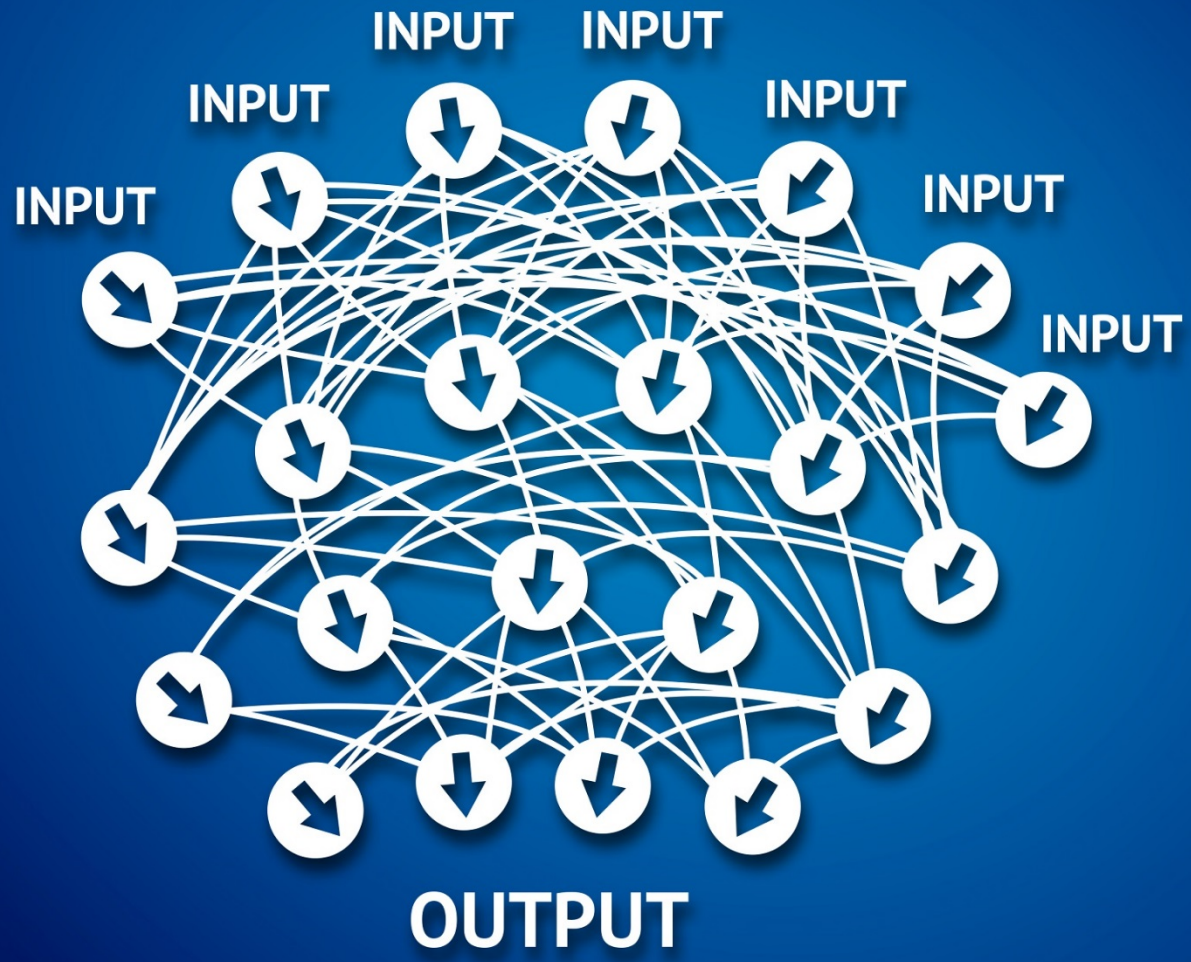


Source: nature.com

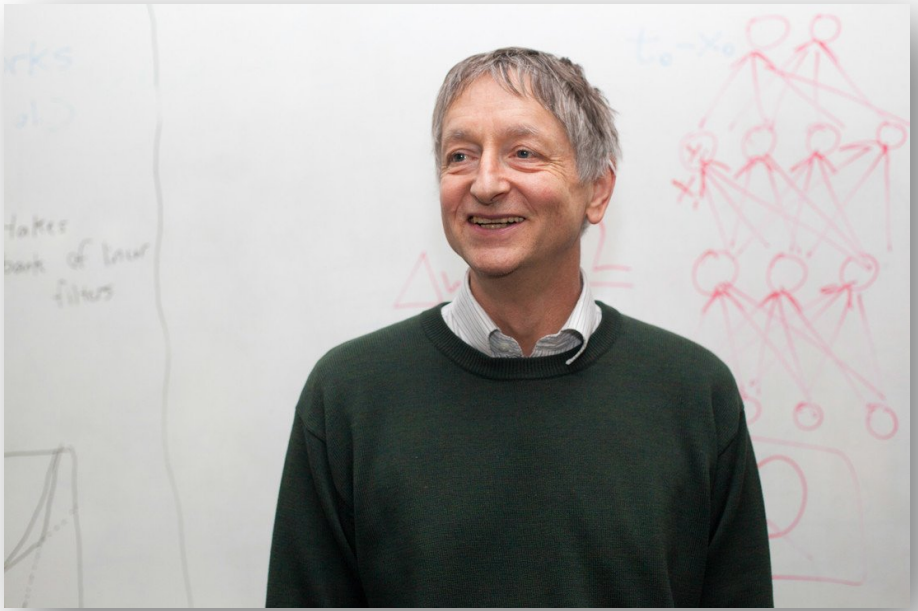
¿Qué es Deep Learning?



Source: Time Magazine

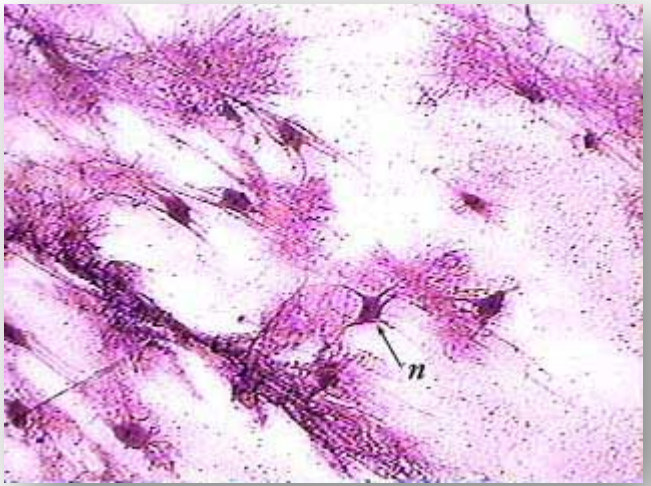

INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
INPUT
OUTPUT

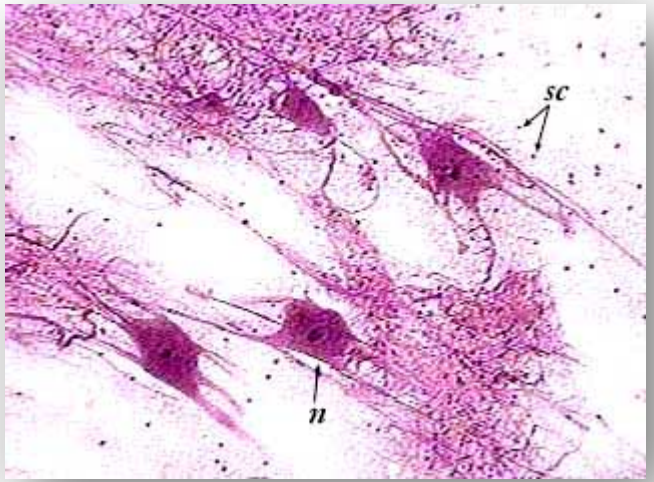
¿Qué es Deep Learning?



Geoffrey Hinton

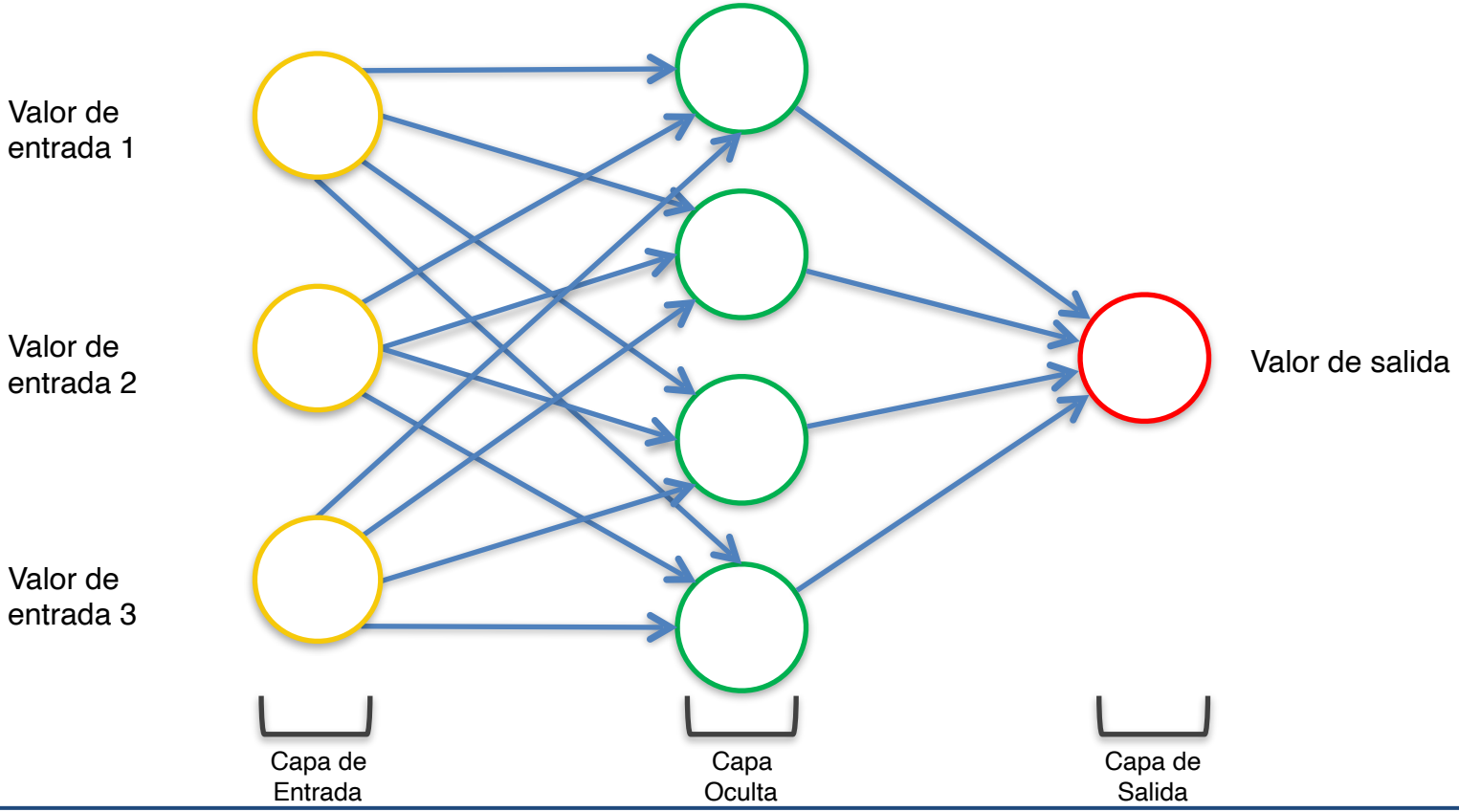
¿Qué es Deep Learning?



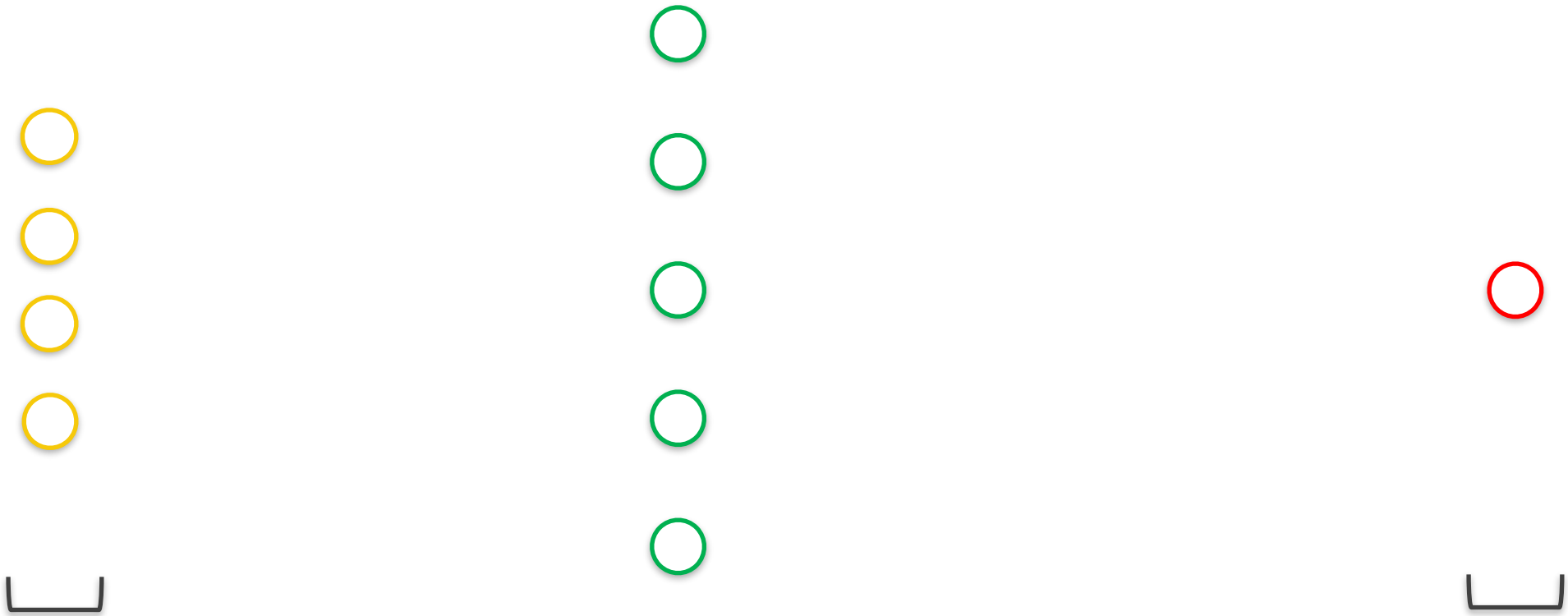


ImEdad Source: www.austincc.edu

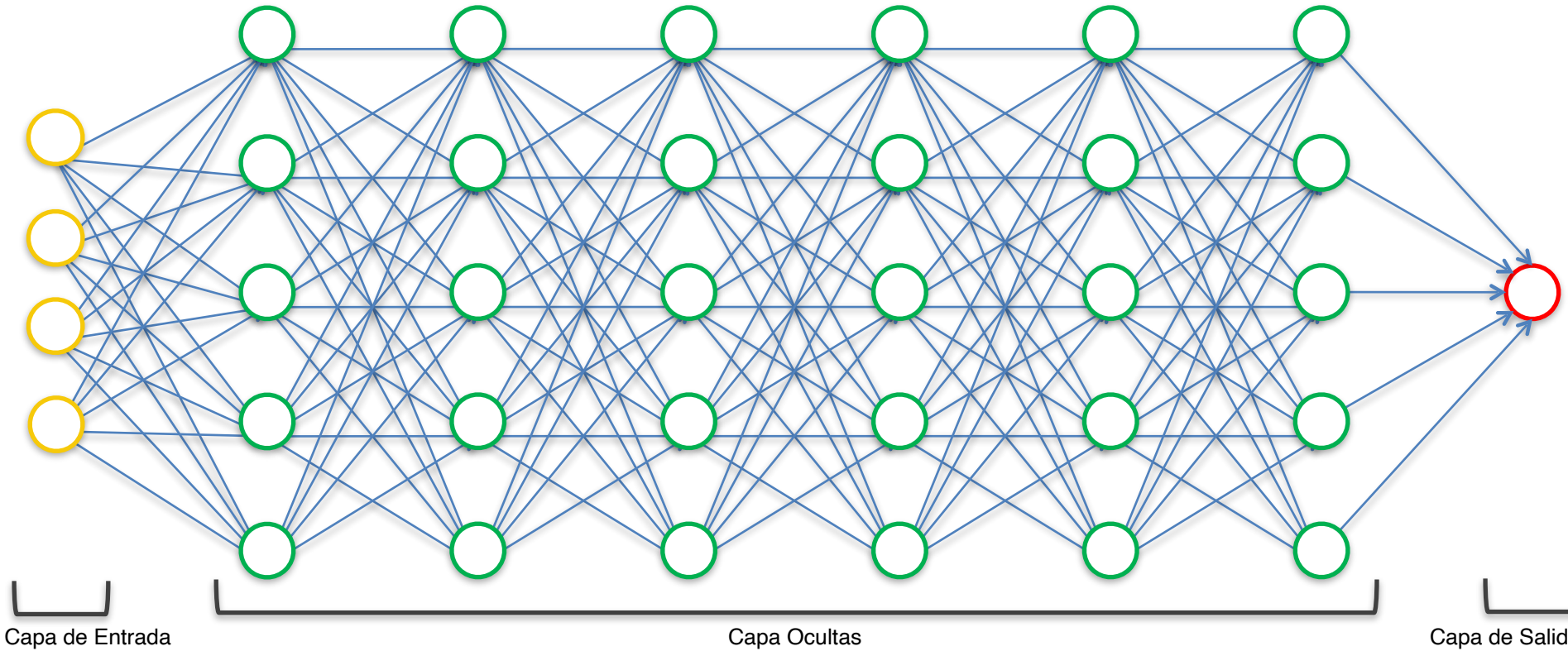
¿Qué es Deep Learning?



¿Qué es Deep Learning?



¿Qué es Deep Learning?



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 Séries 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

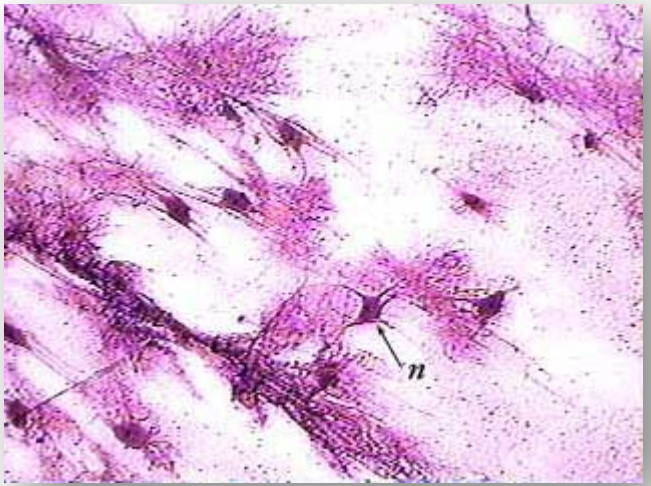
Plan de Ataque

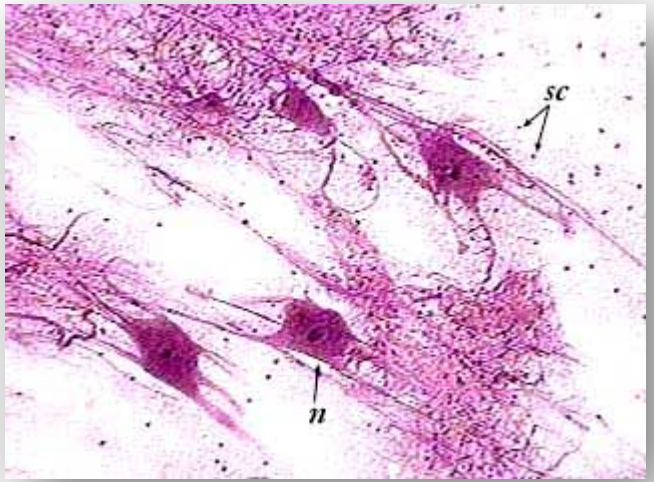
Qué vamos a aprender en la sección

- 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

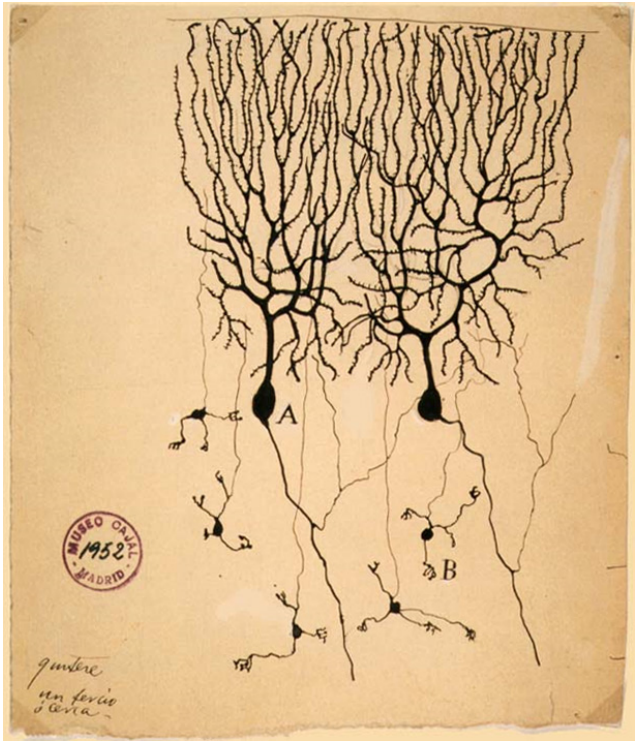
La Neurona





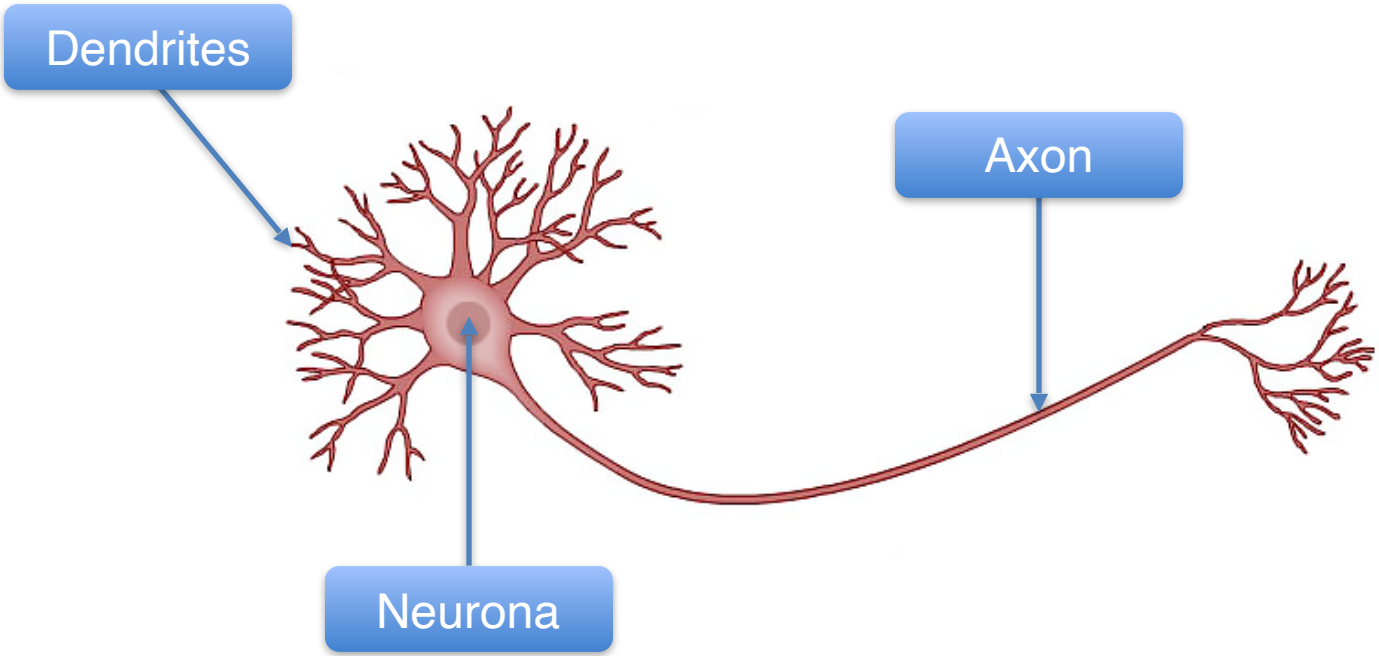
ImEdad Source: www.austincc.edu

La Neurona



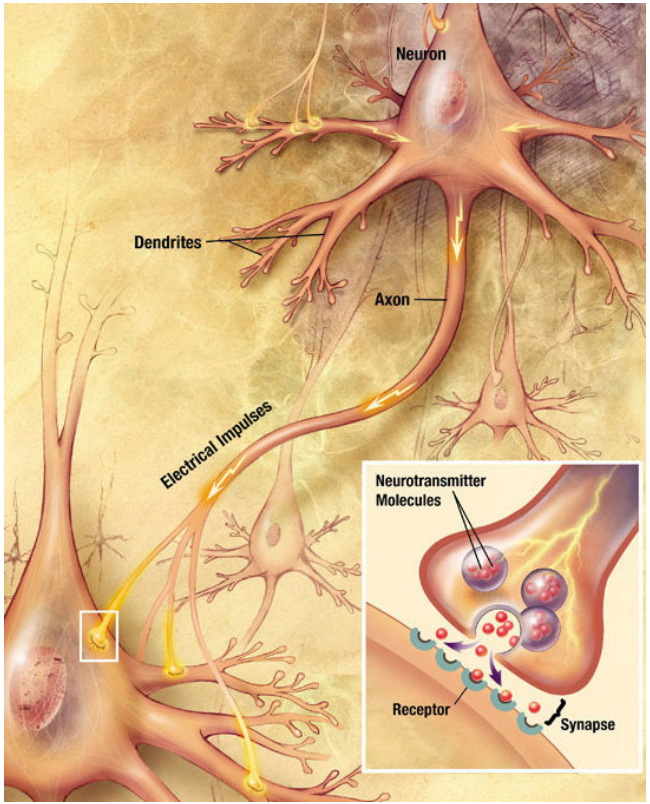
ImEdad Source: Wikipedia

La Neurona



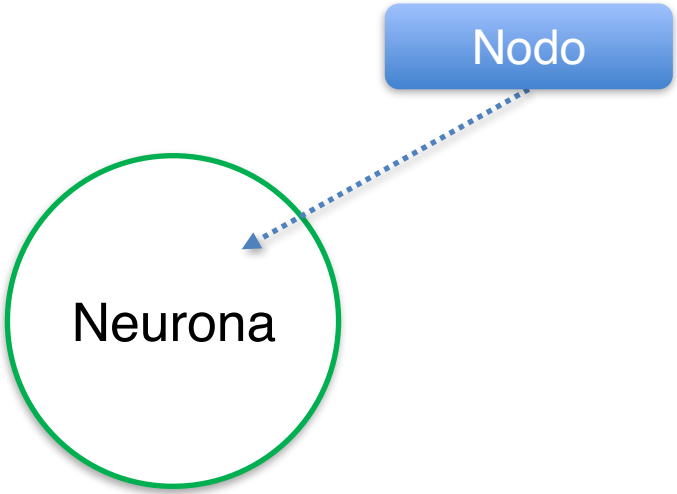
ImEdad Source: Wikipedia

La Neurona

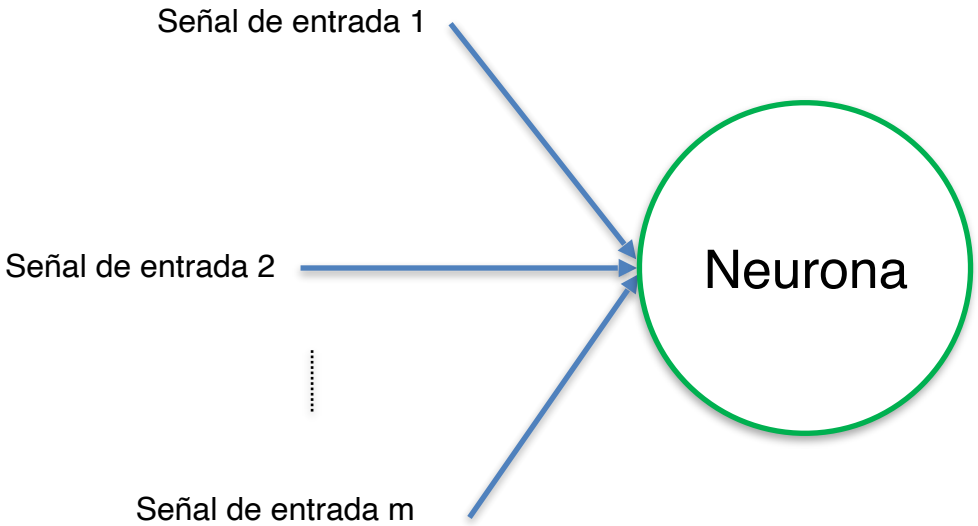


ImEdad Source: Wikipedia

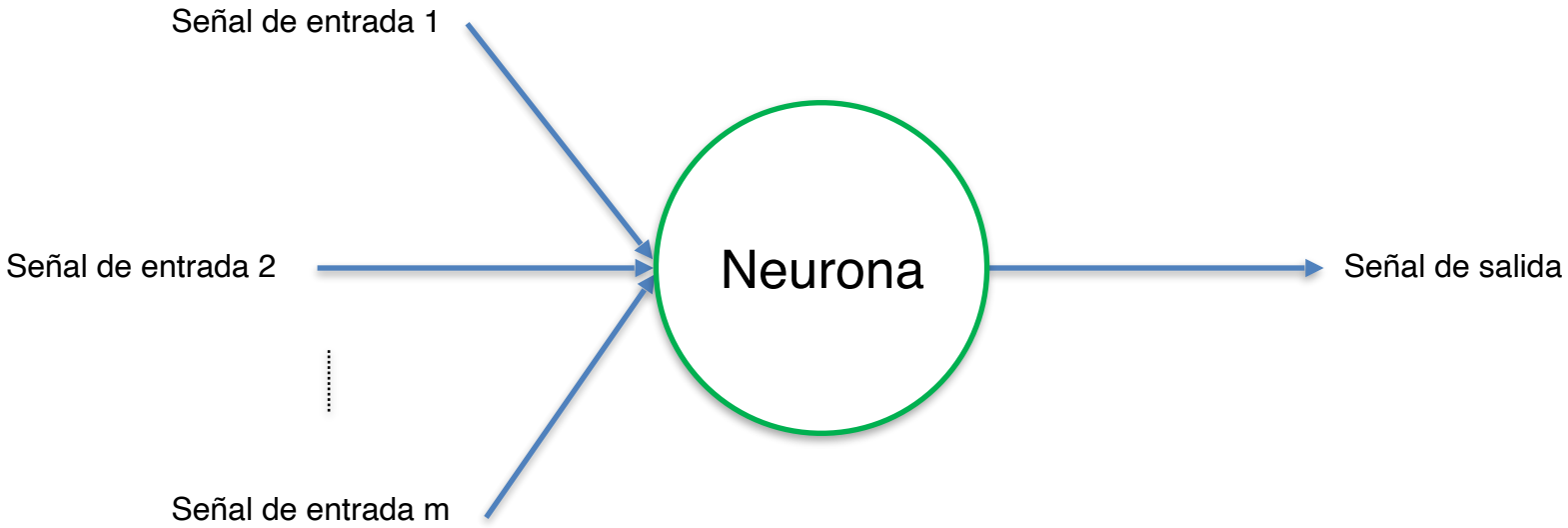
La Neurona



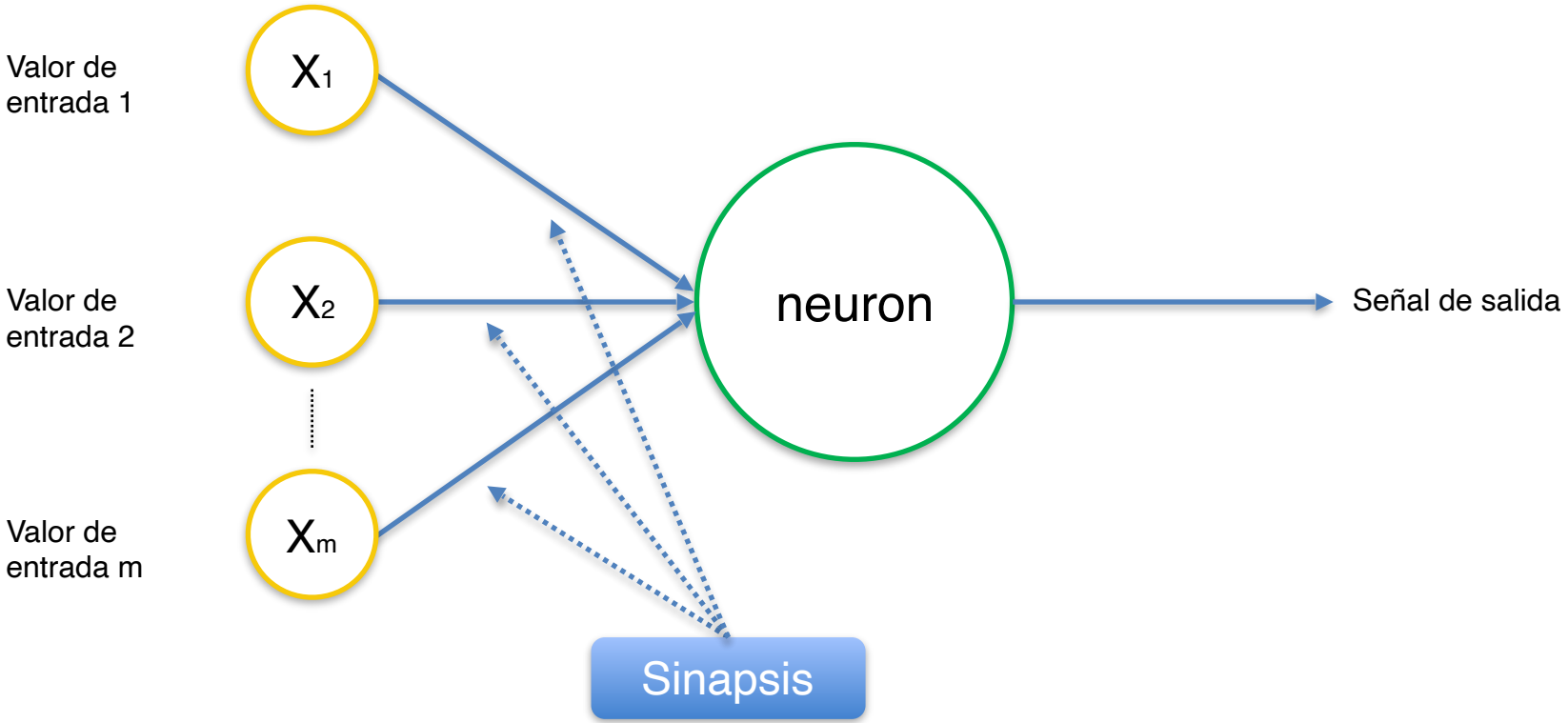
La Neurona



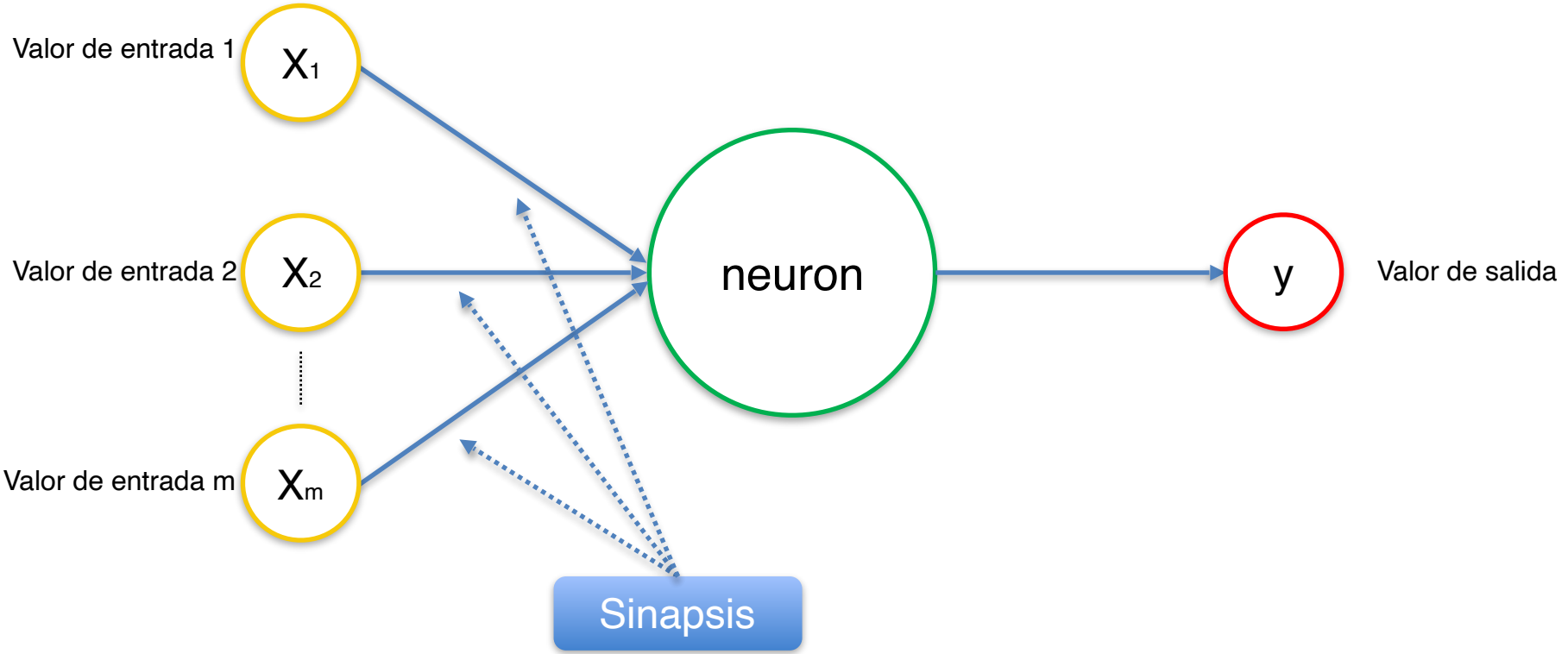
La Neurona



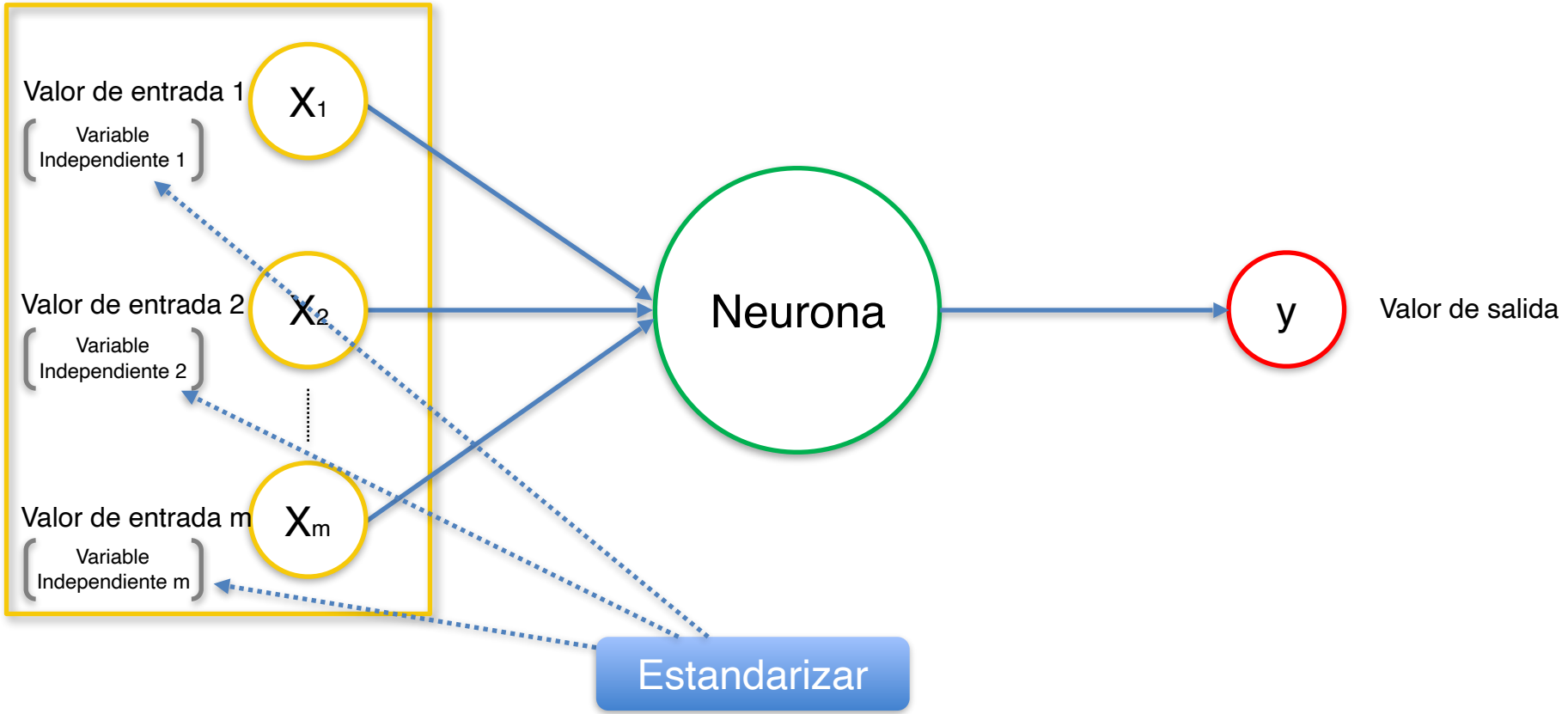
La Neurona



La Neurona



La Neurona

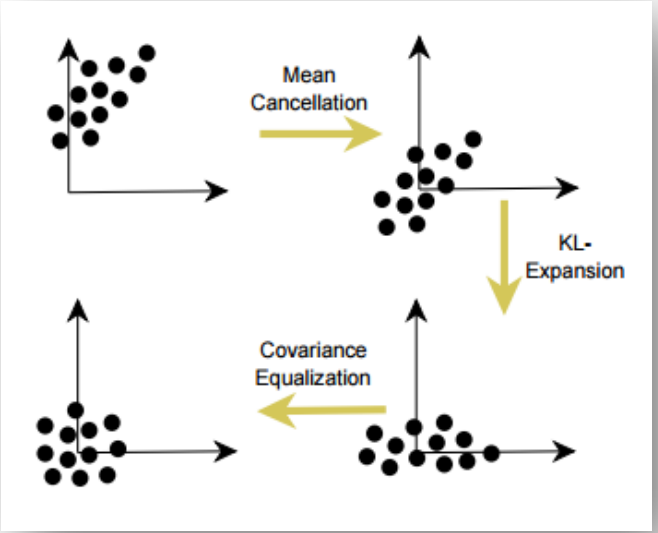


La Neurona

Lectura adicional

Efficient BackProp

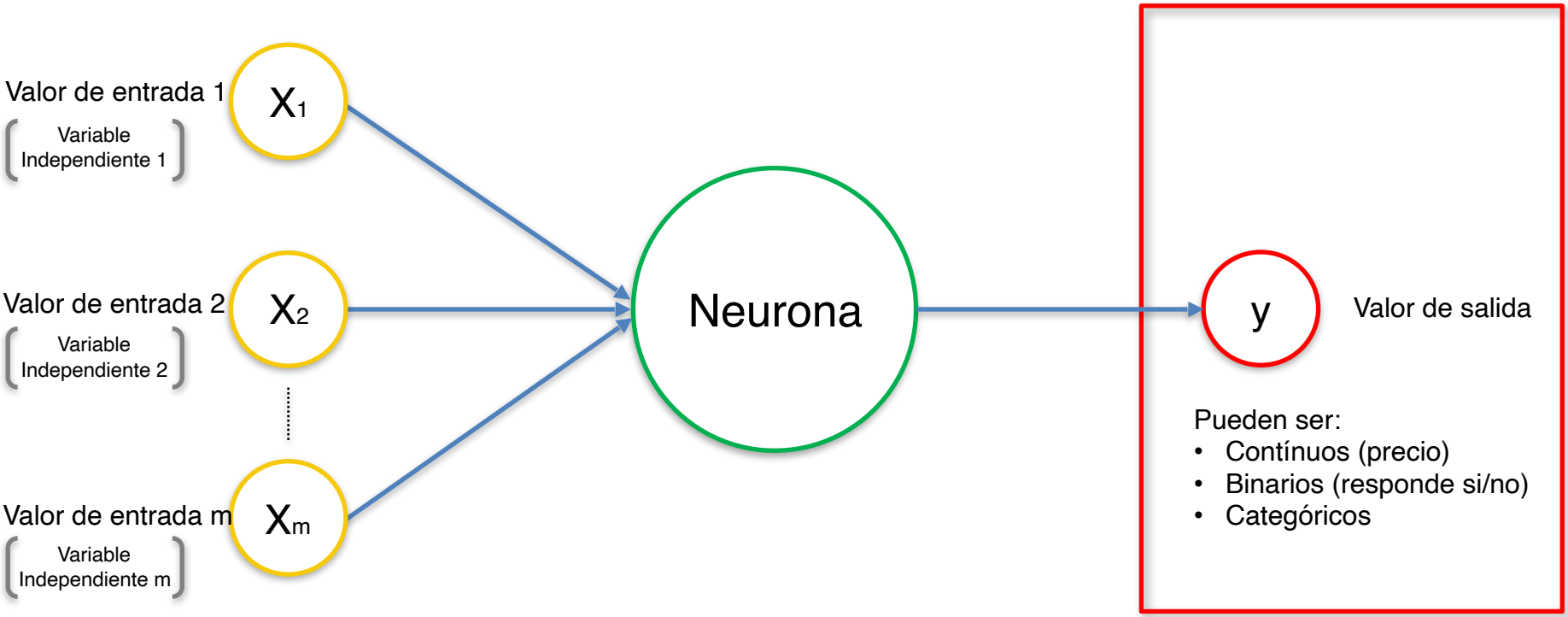
By Yann LeCun et al. (1998)



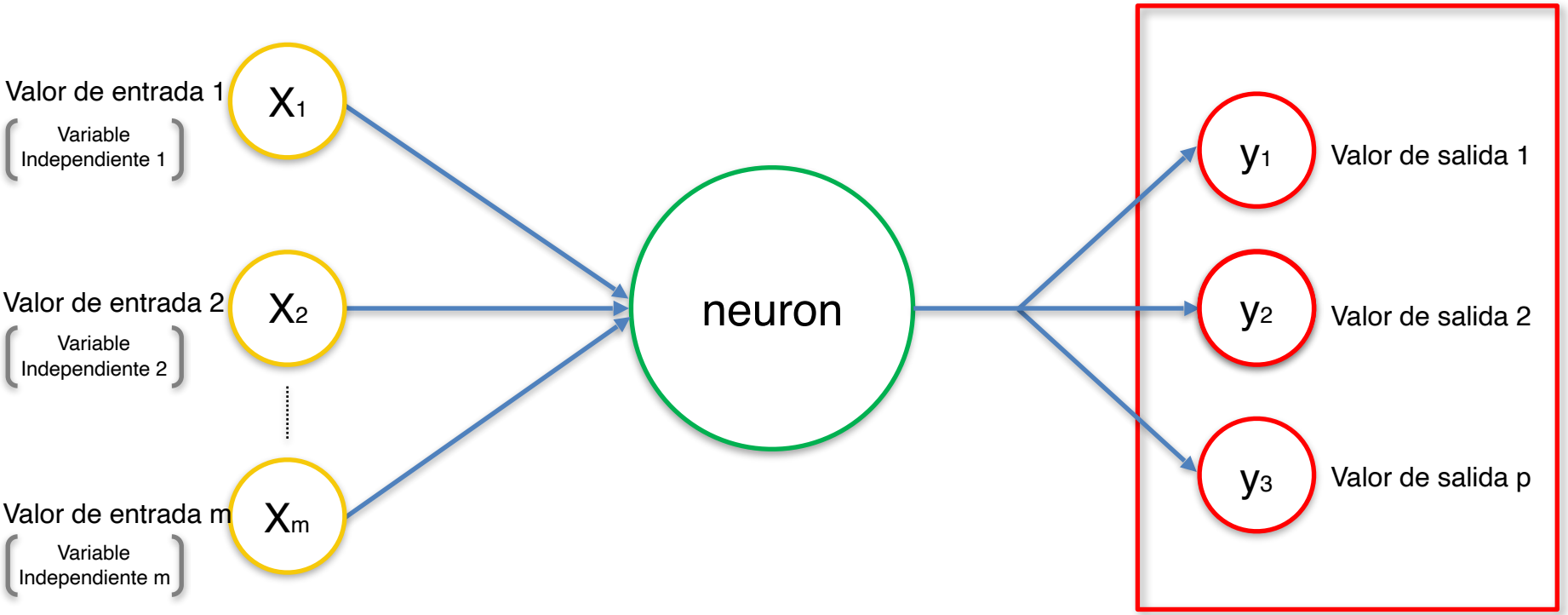
Link:

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

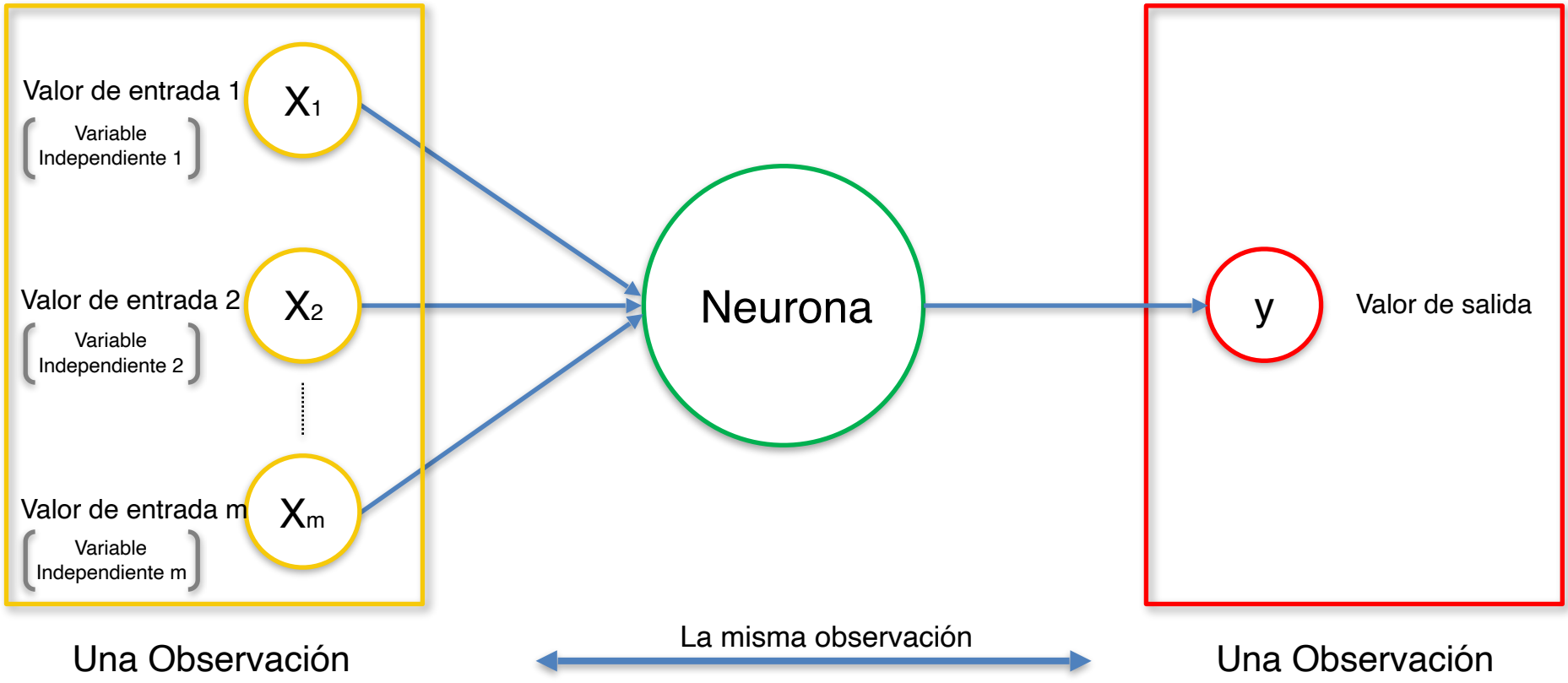
La Neurona



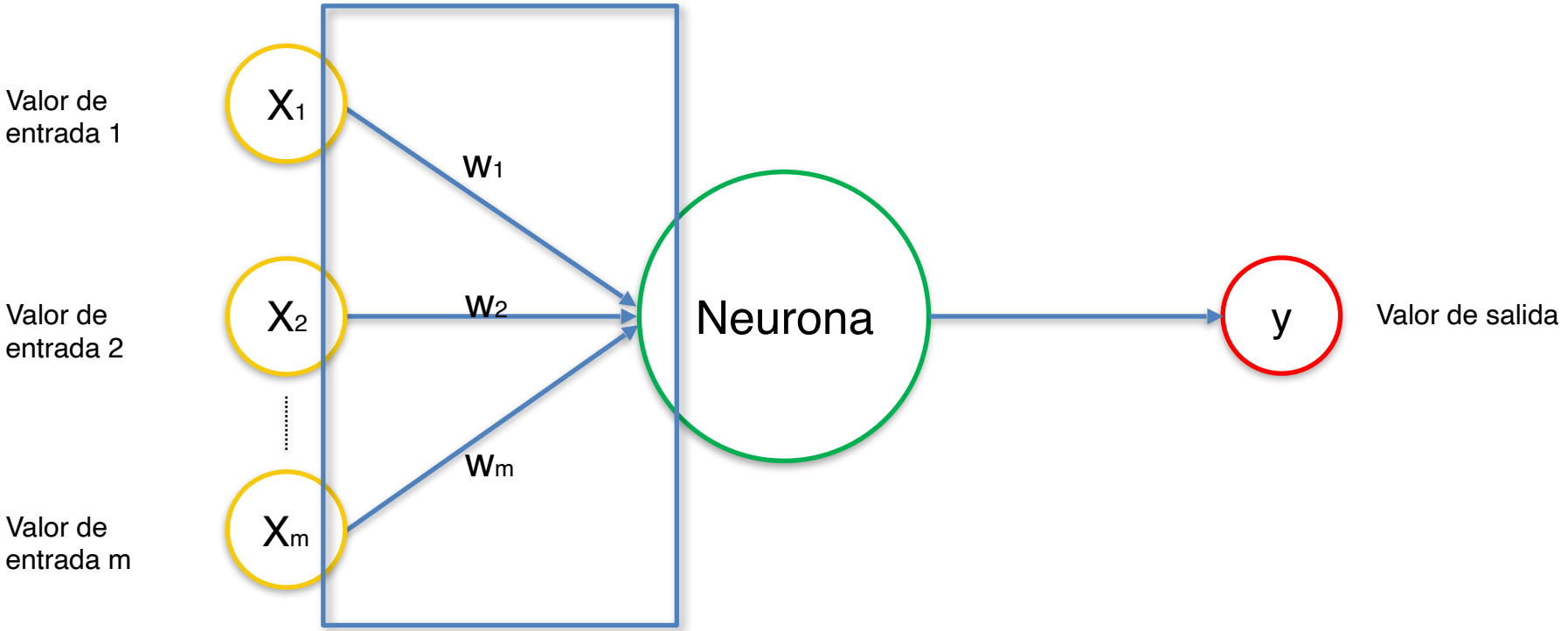
La Neurona



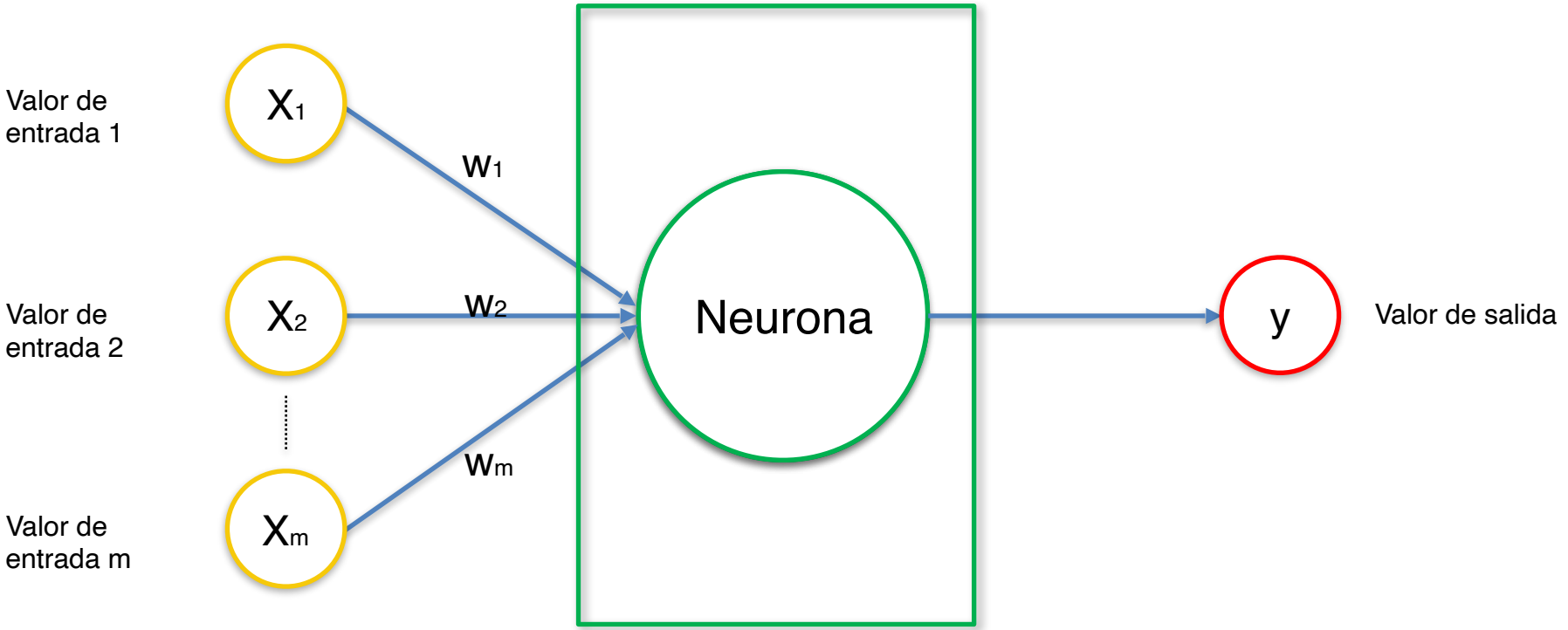
La Neurona



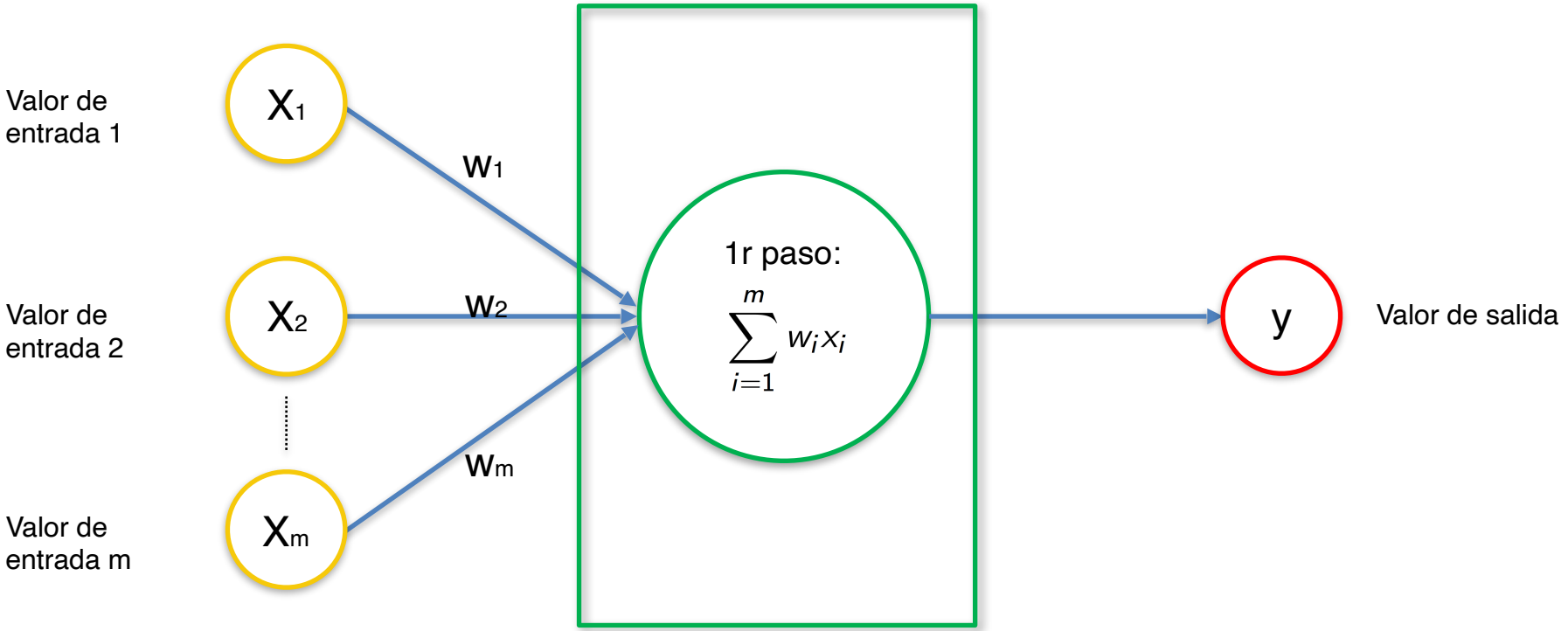
La Neurona



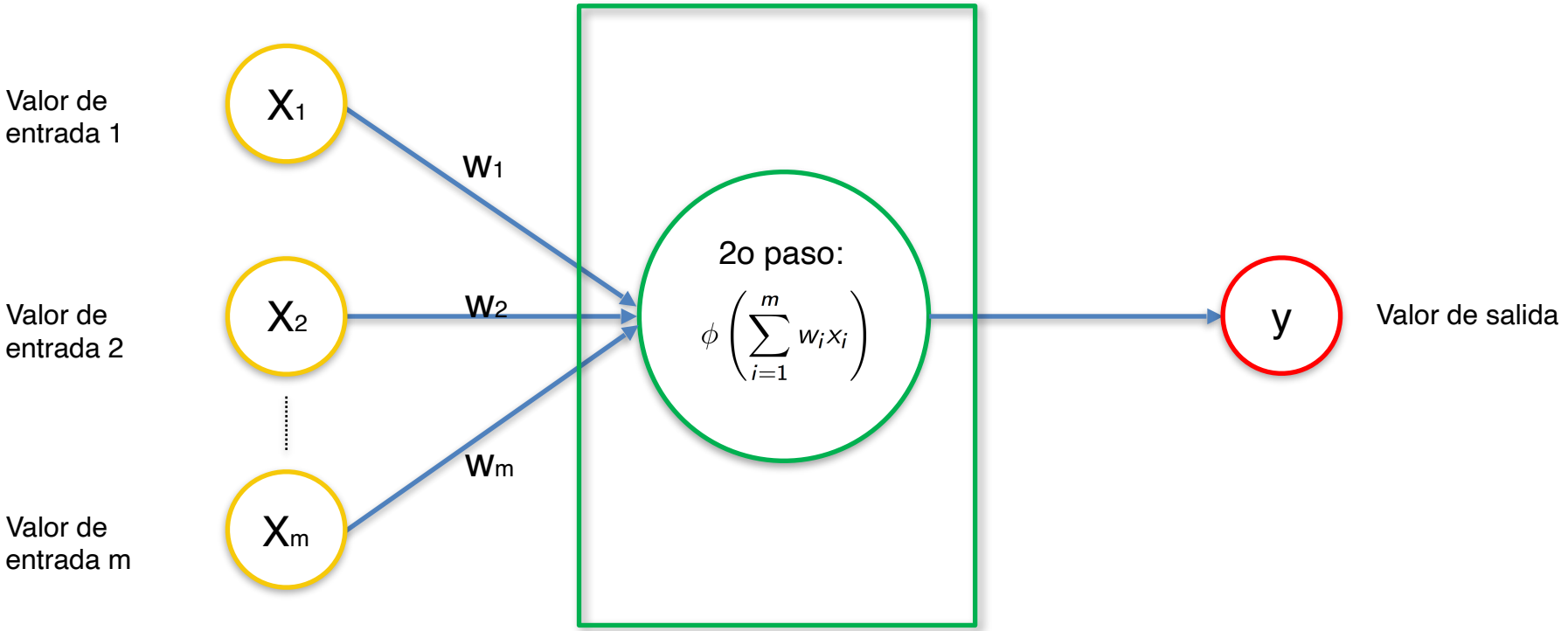
La Neurona



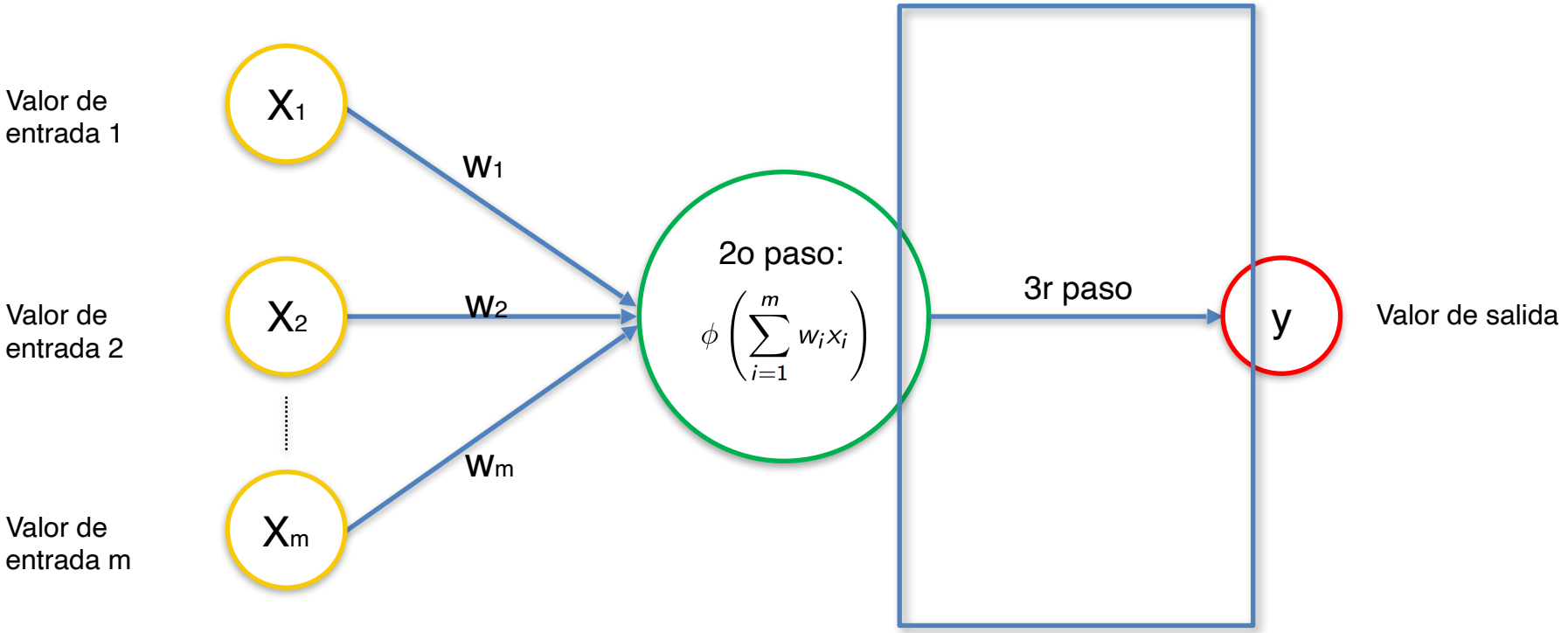
La Neurona



La Neurona

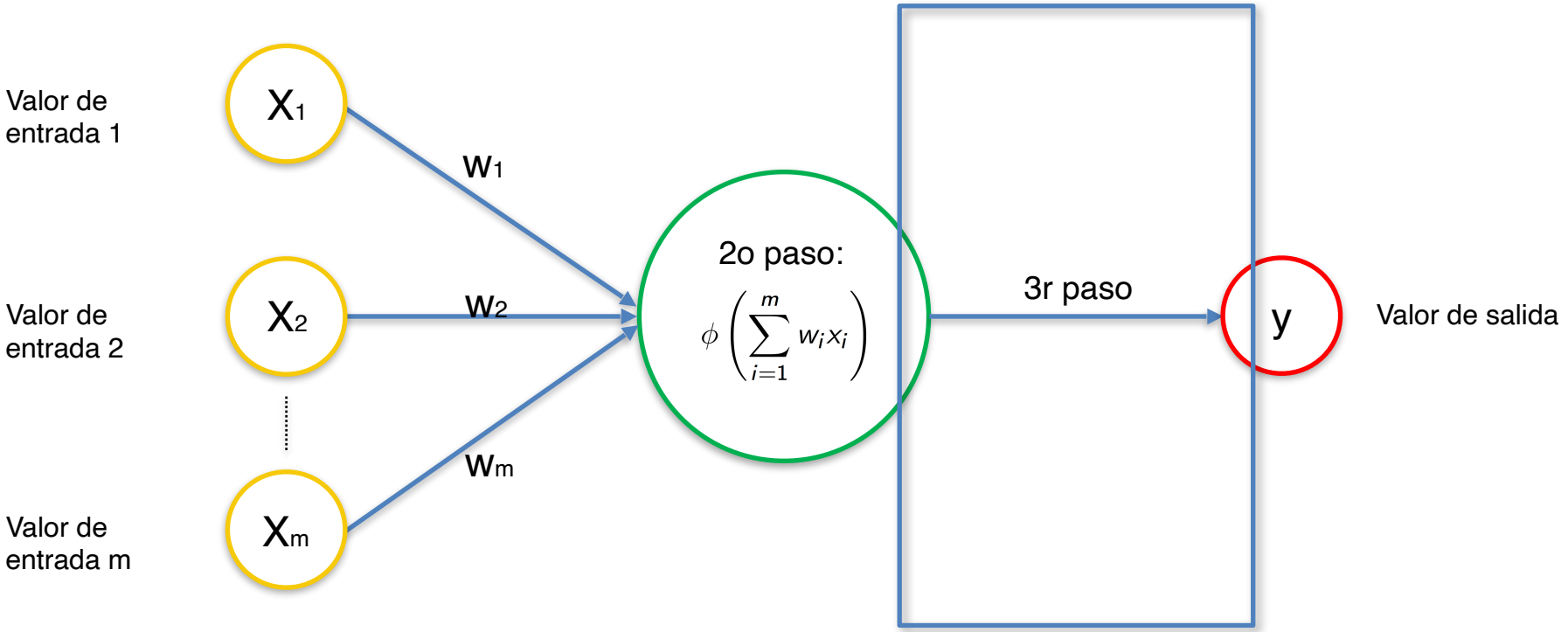


La Neurona

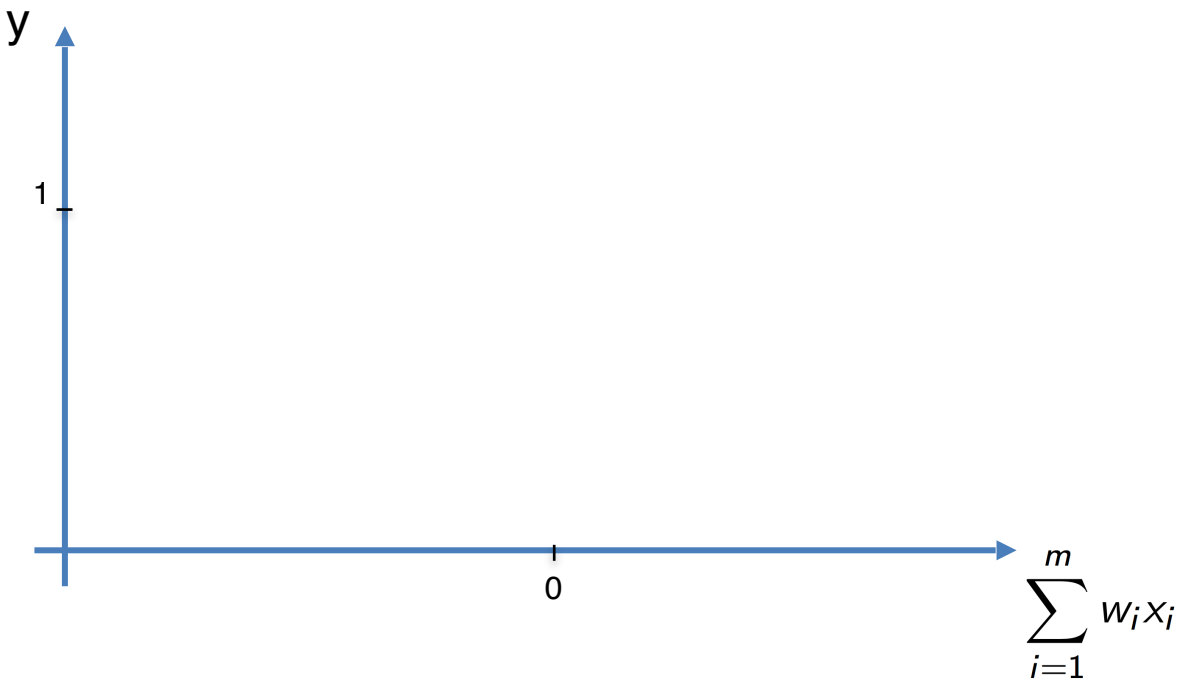


La Función de Activación

La Función de Activación

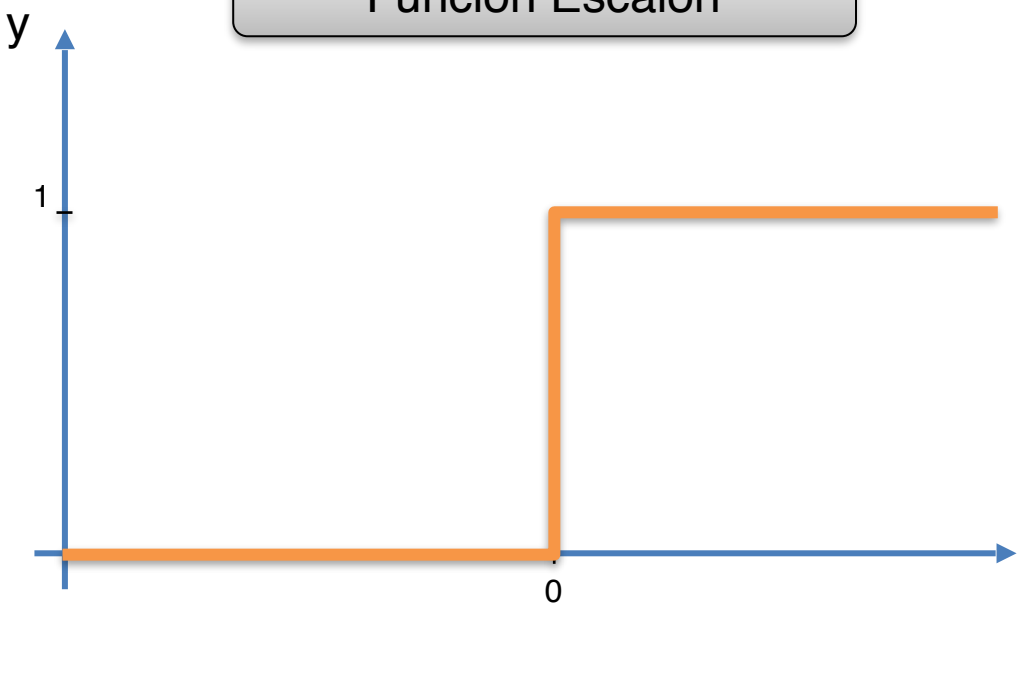


La Función de Activación



La Función de Activación

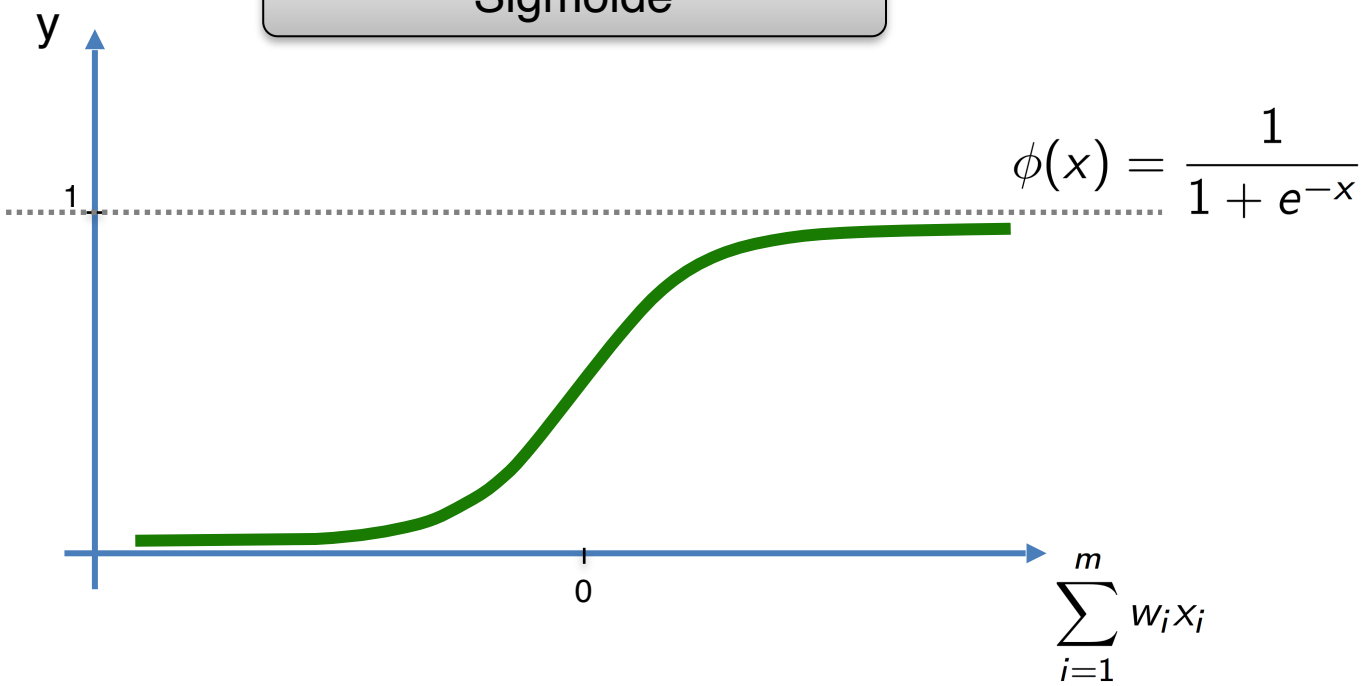
Función Escalón



$$\phi(x) = \begin{cases} 1 \text{ if } x \geq 0 \\ 0 \text{ if } x < 0 \end{cases}$$

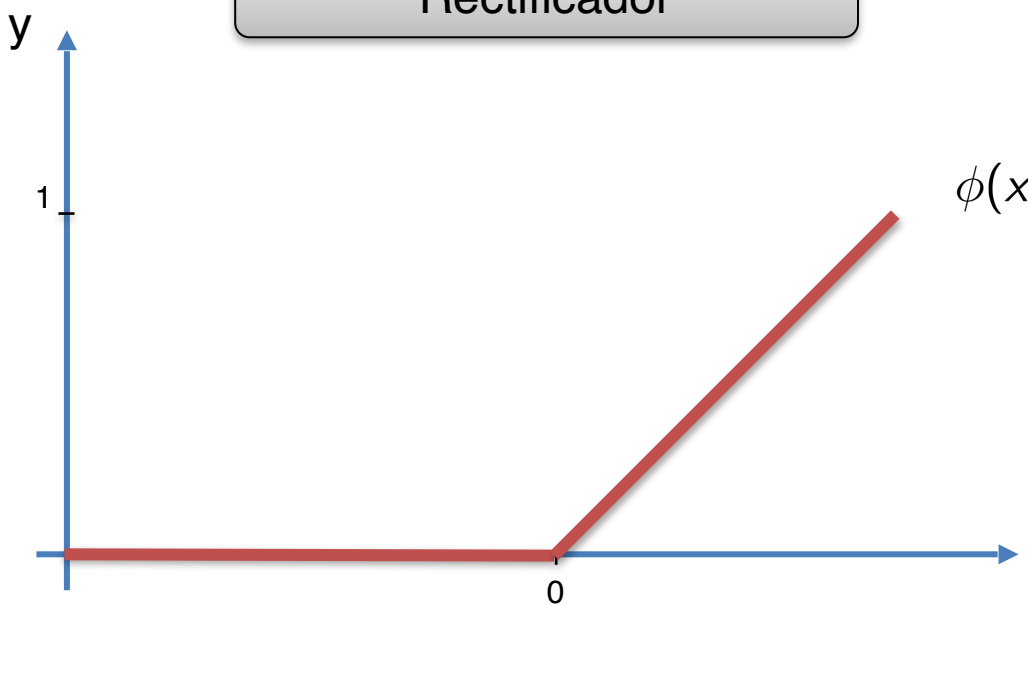
La Función de Activación

Sigmoide



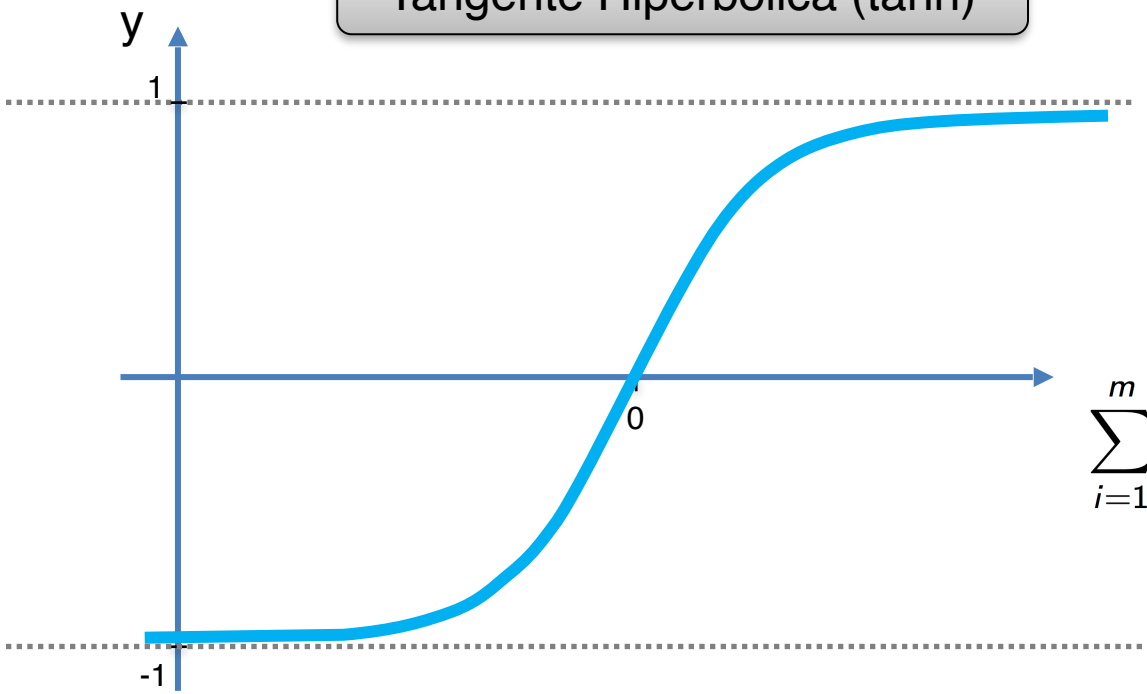
La Función de Activación

Rectificador



La Función de Activación

Tangente Hiperbólica (tanh)



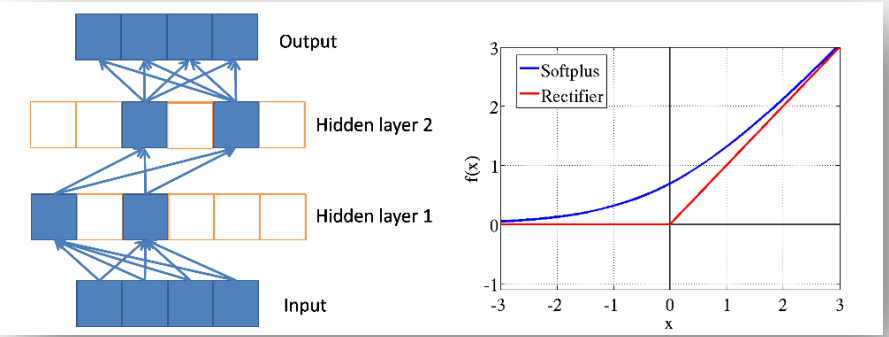
$$\phi(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

La Función de Activación

Lecturas Adicionales:

Deep sparse Rectificador neural networks

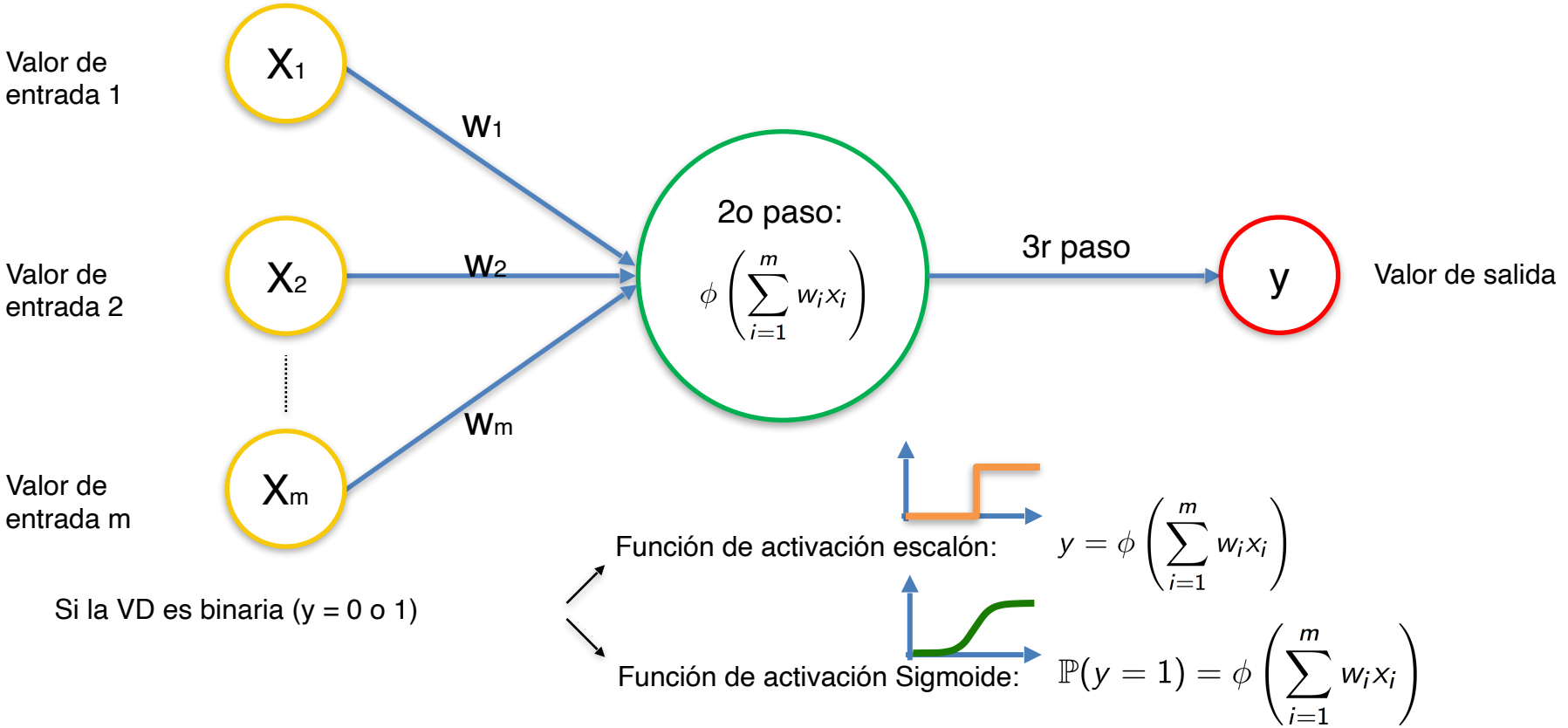
By Xavier Glorot et al. (2011)



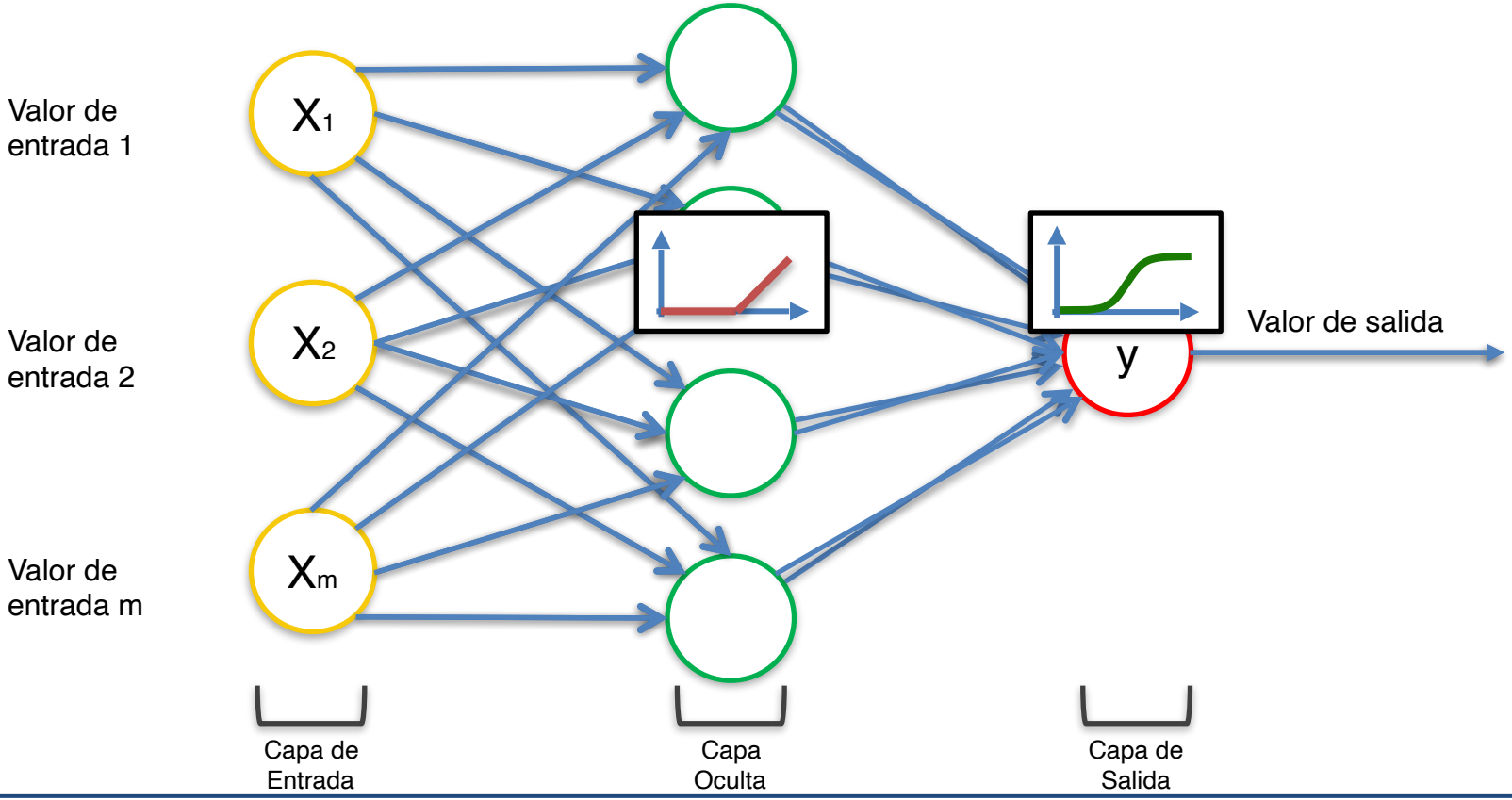
Link:

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

La Función de Activación

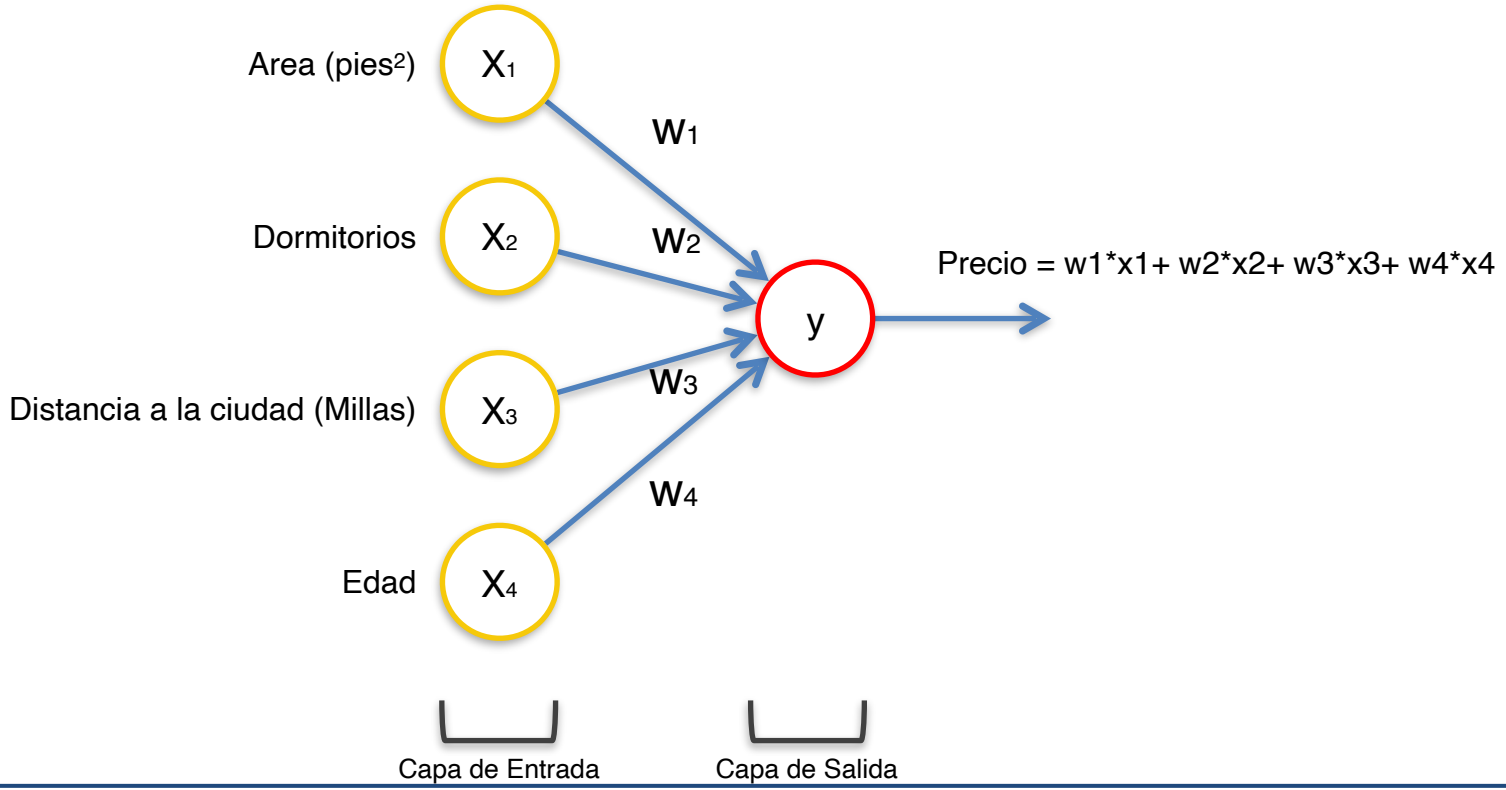


La Función de Activación

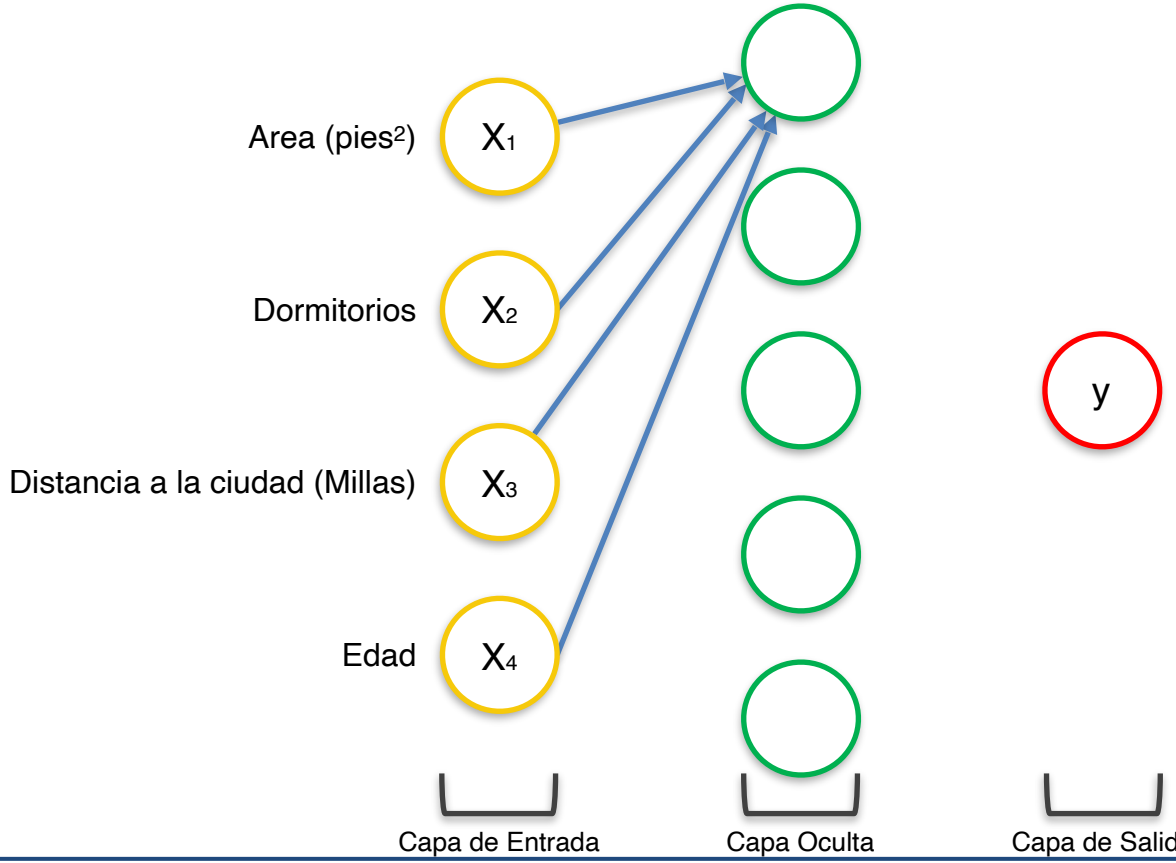


¿Cómo funcionan las Redes Neuronales?

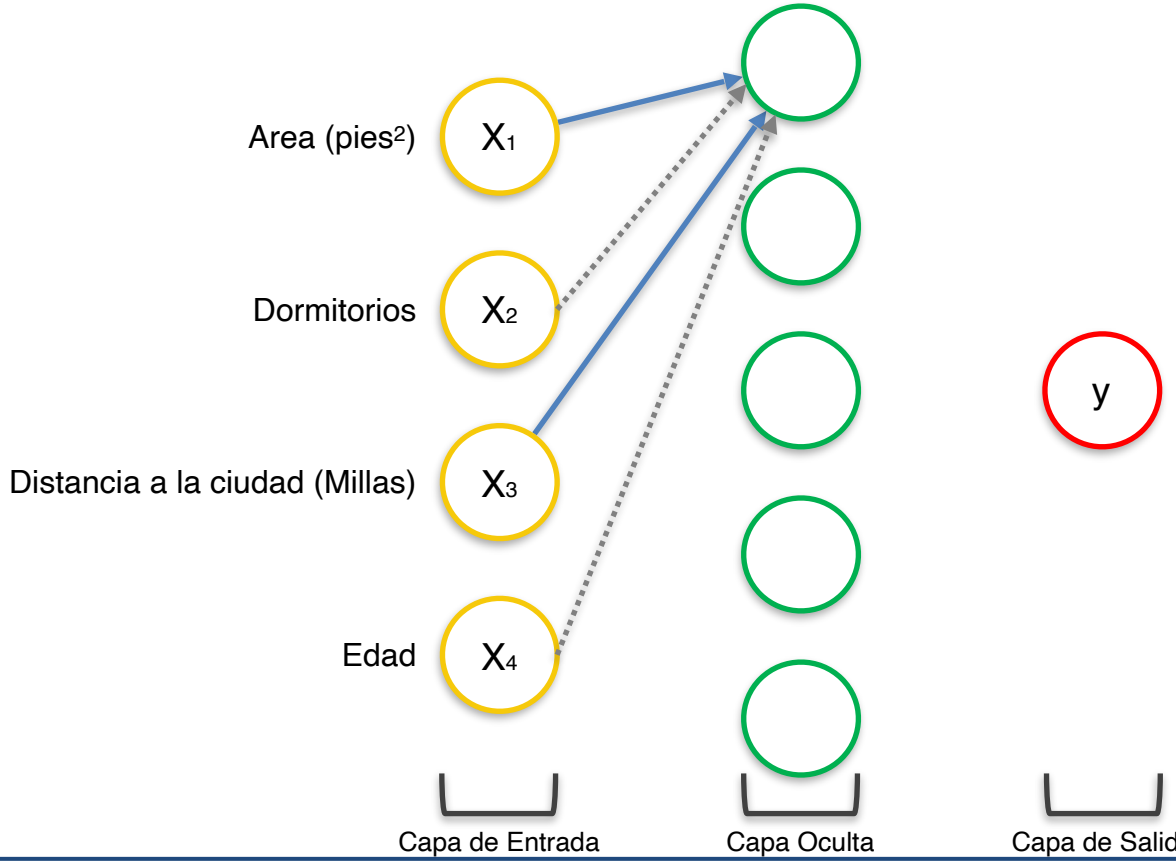
¿Cómo funcionan las Redes Neuronales?



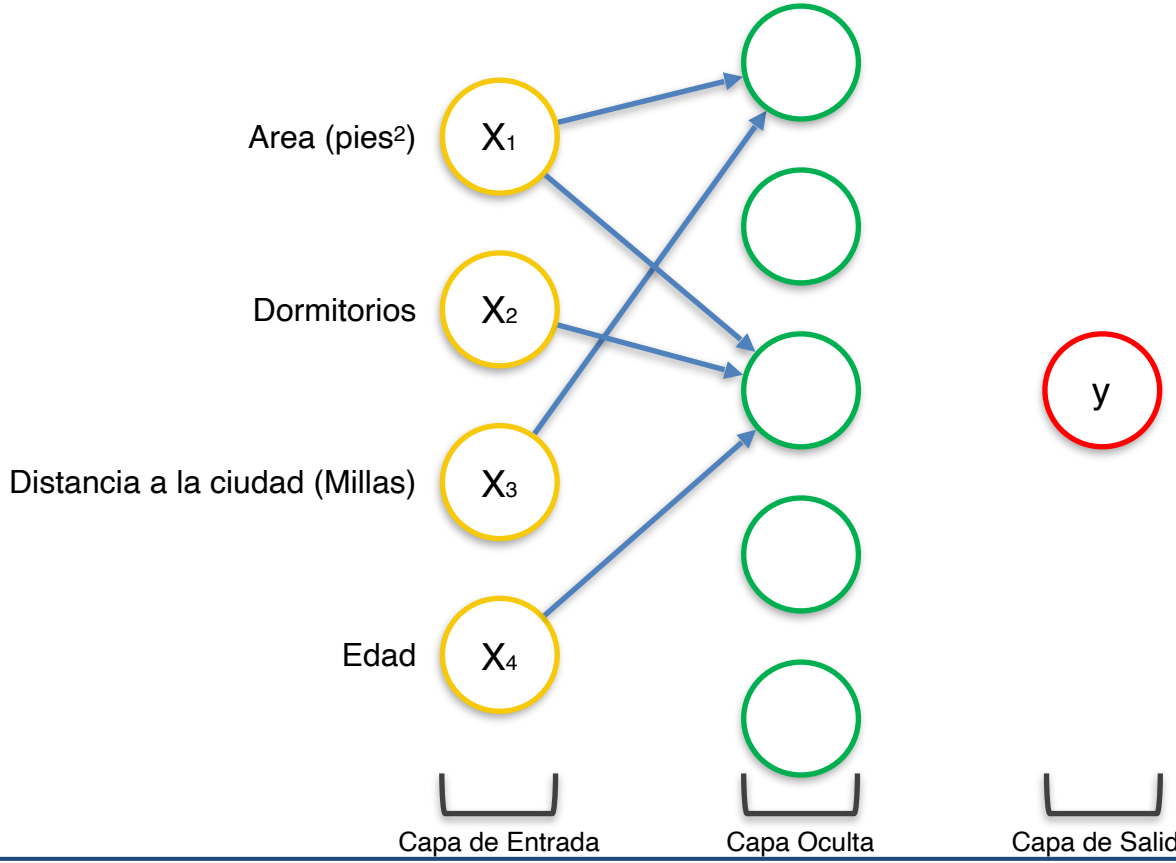
¿Cómo funcionan las Redes Neuronales?



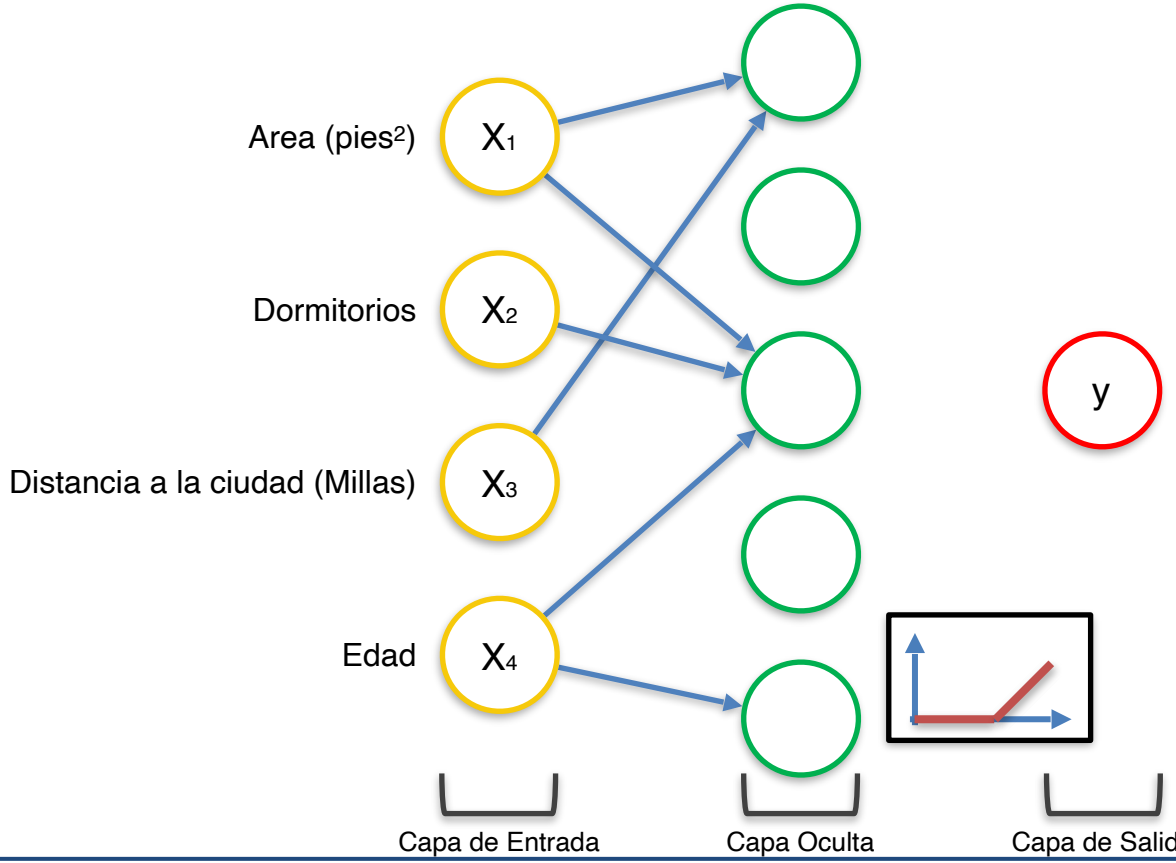
¿Cómo funcionan las Redes Neuronales?



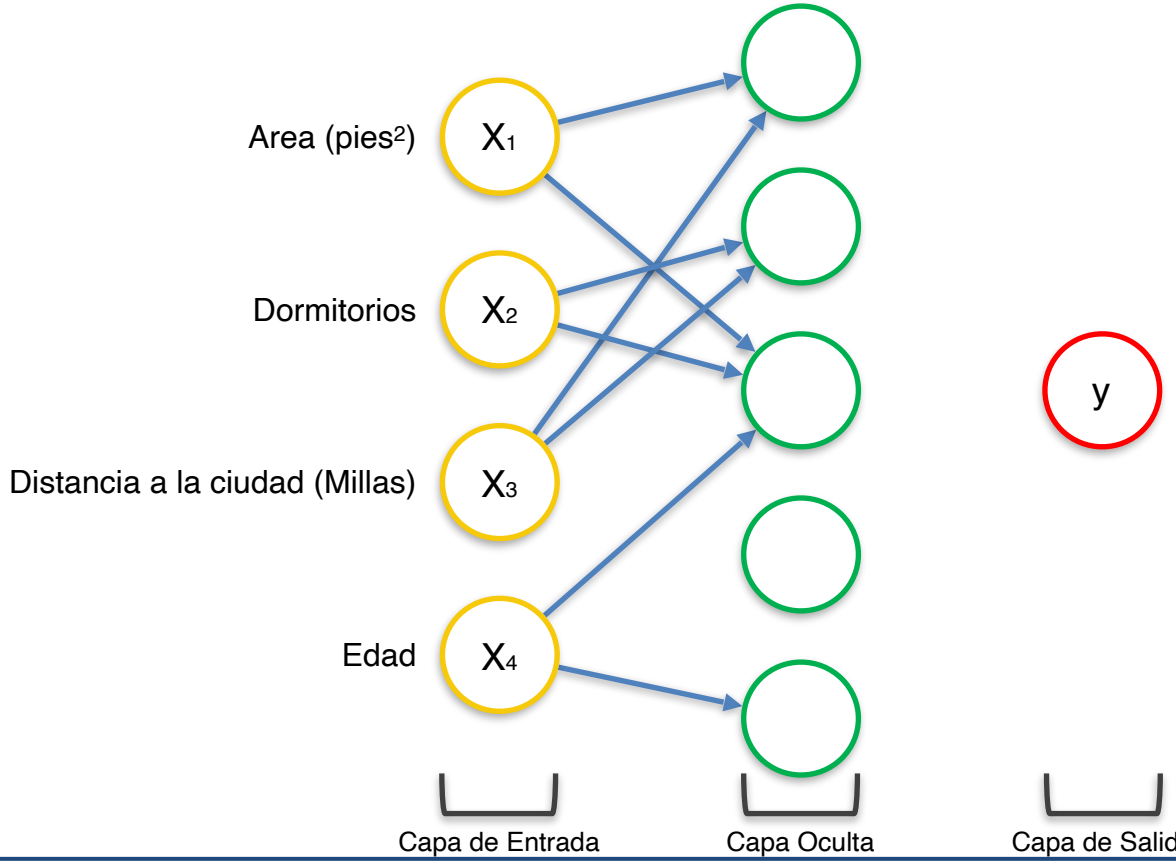
¿Cómo funcionan las Redes Neuronales?



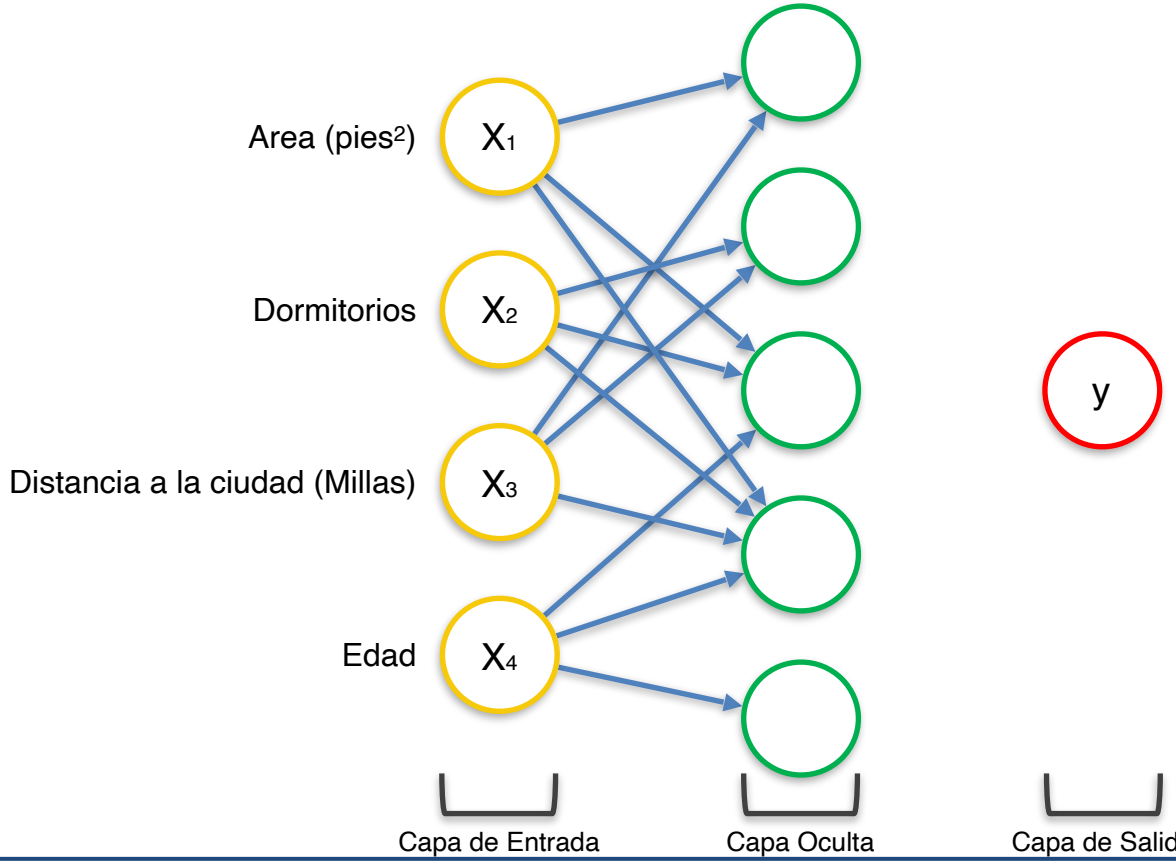
¿Cómo funcionan las Redes Neuronales?



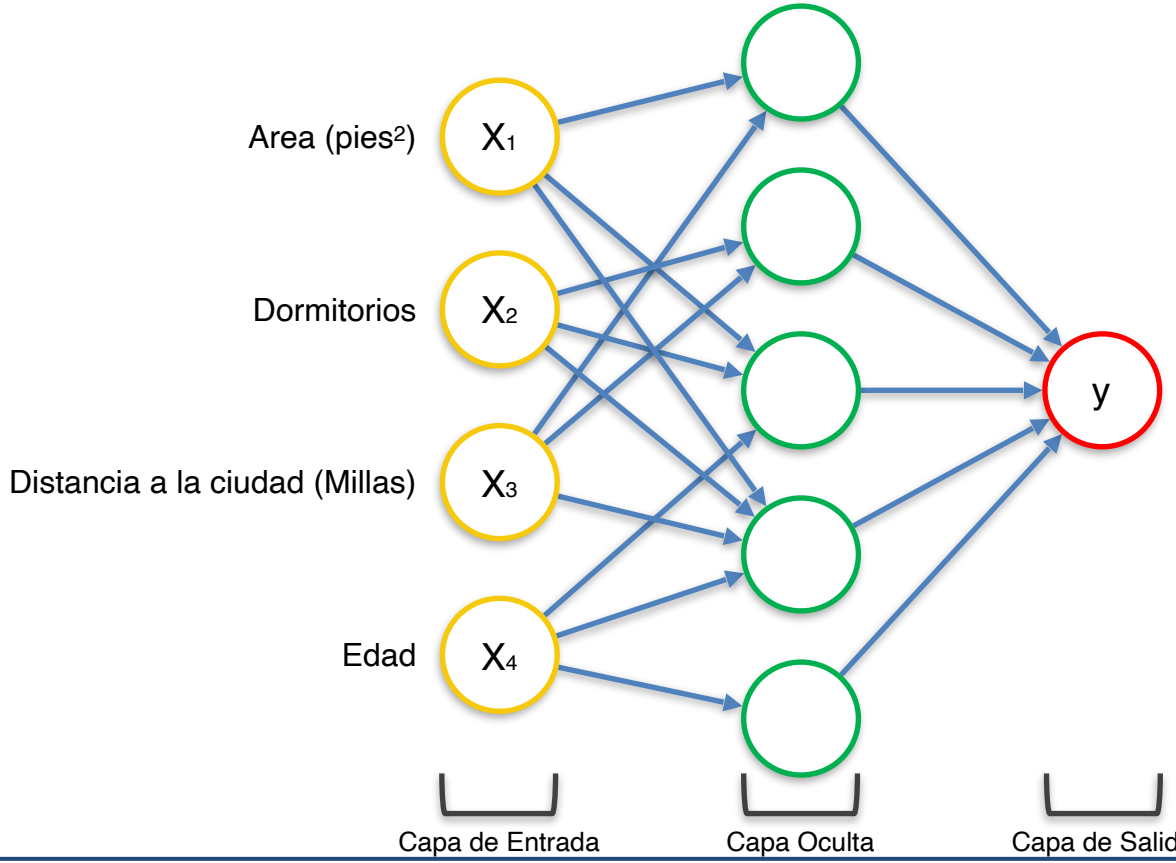
¿Cómo funcionan las Redes Neuronales?



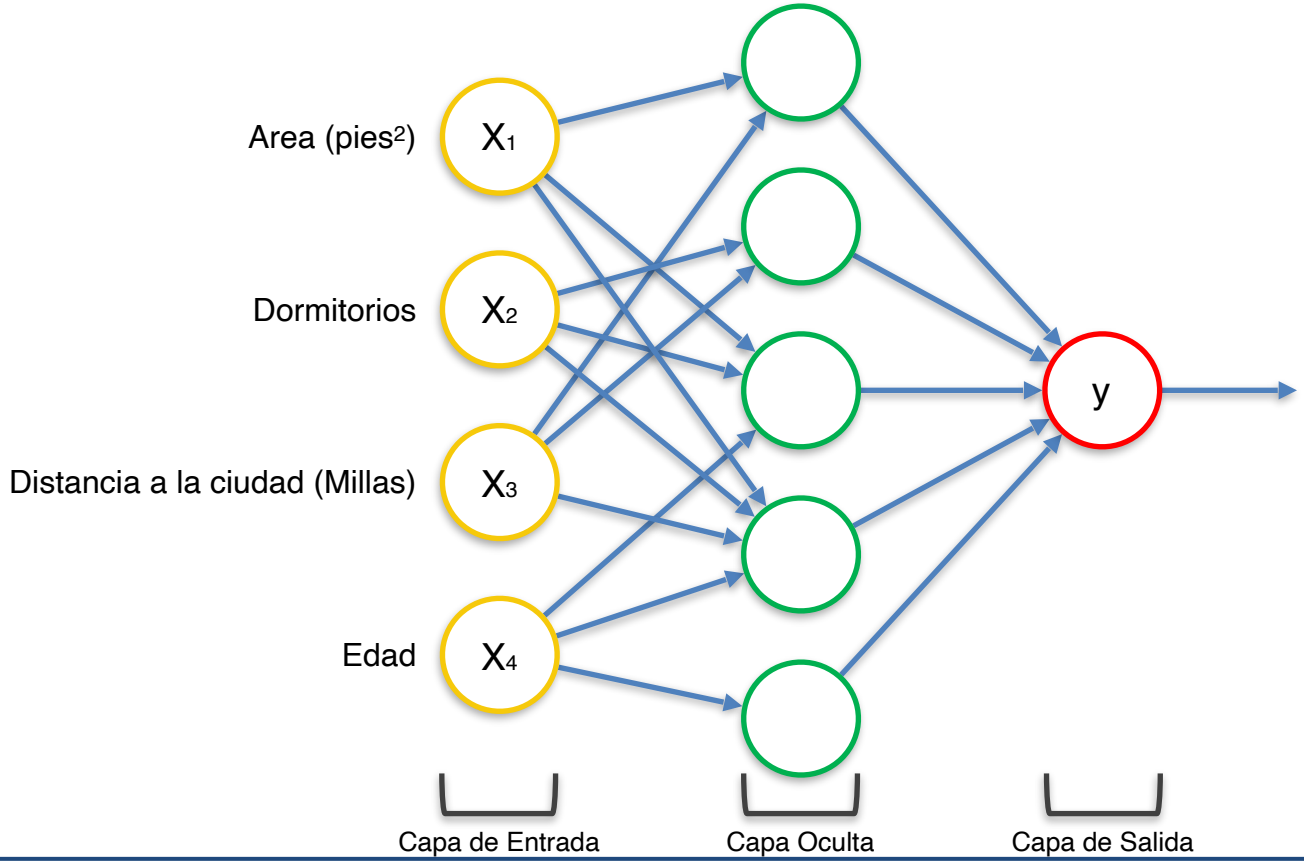
¿Cómo funcionan las Redes Neuronales?



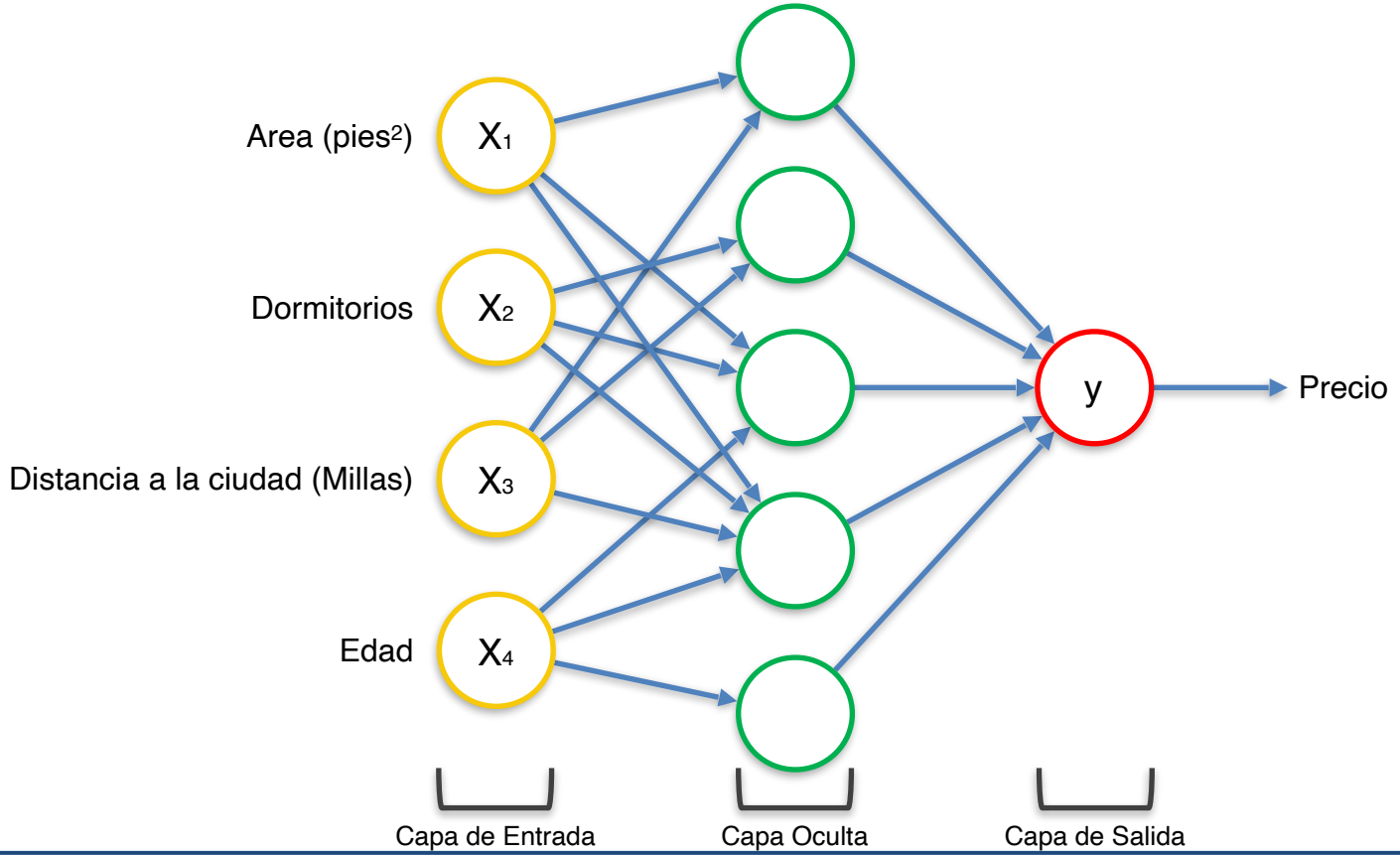
¿Cómo funcionan las Redes Neuronales?



¿Cómo funcionan las Redes Neuronales?



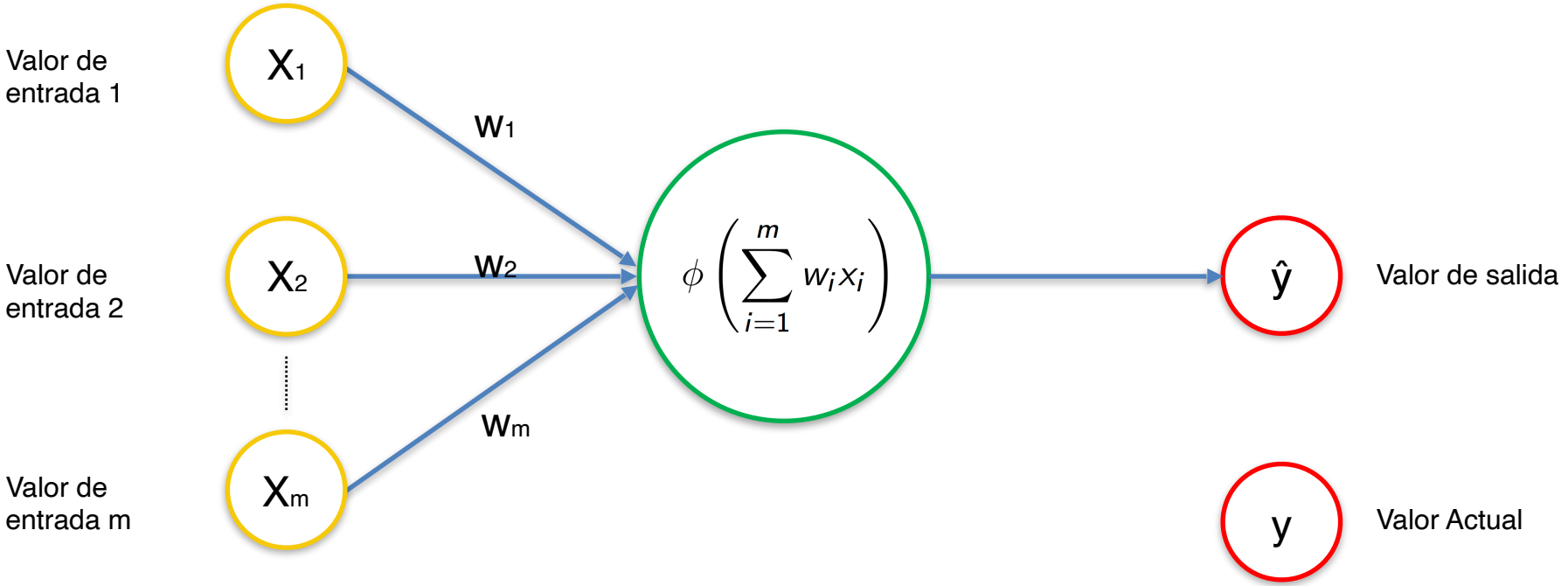
¿Cómo funcionan las Redes Neuronales?



¿Cómo aprenden las Redes Neuronales?

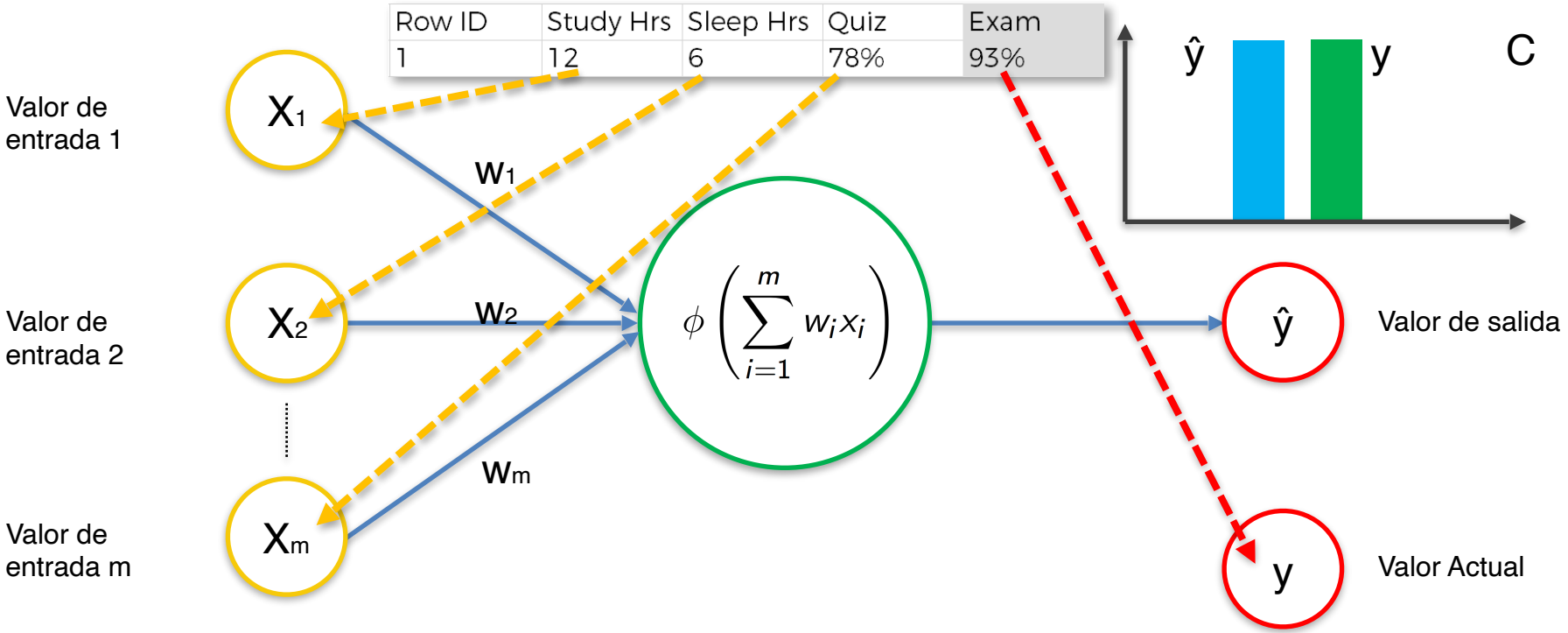
¿Cómo aprenden las Redes Neuronales?

¿Cómo aprenden las Redes Neuronales?

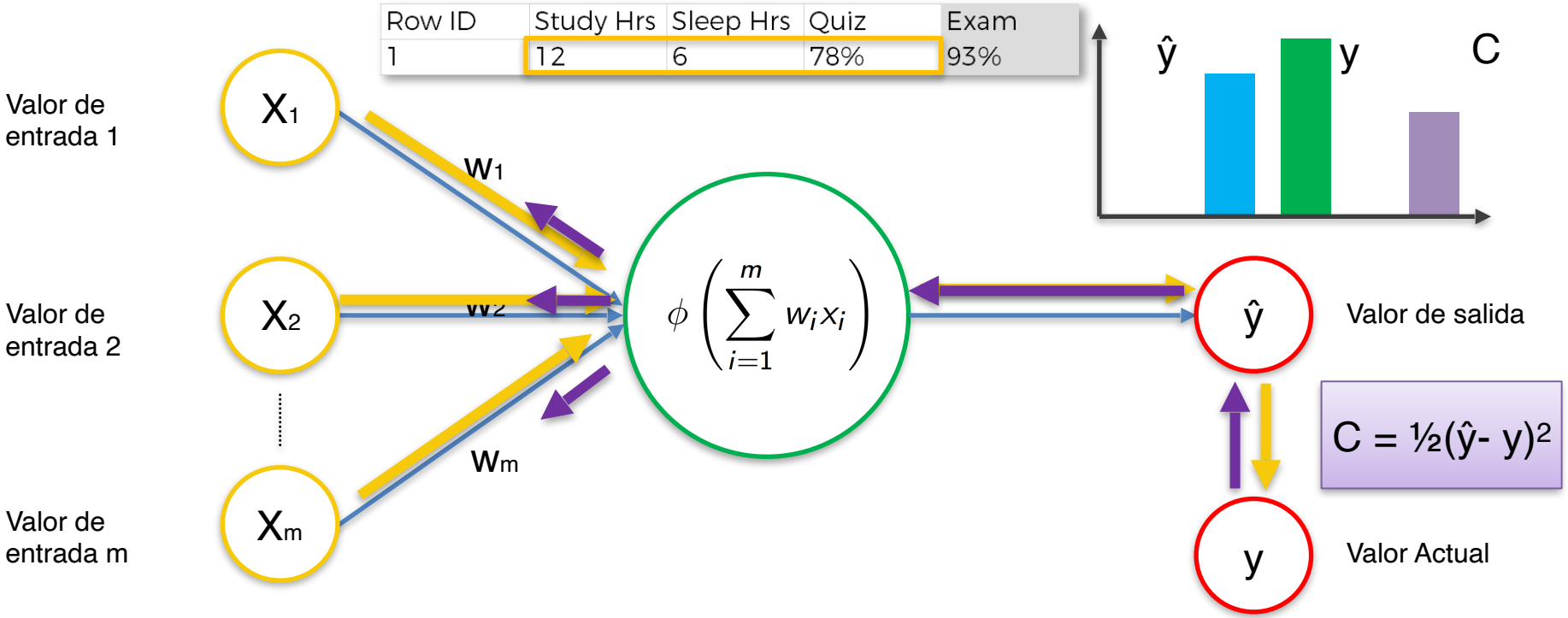


¿Cómo aprenden las Redes Neuronales?

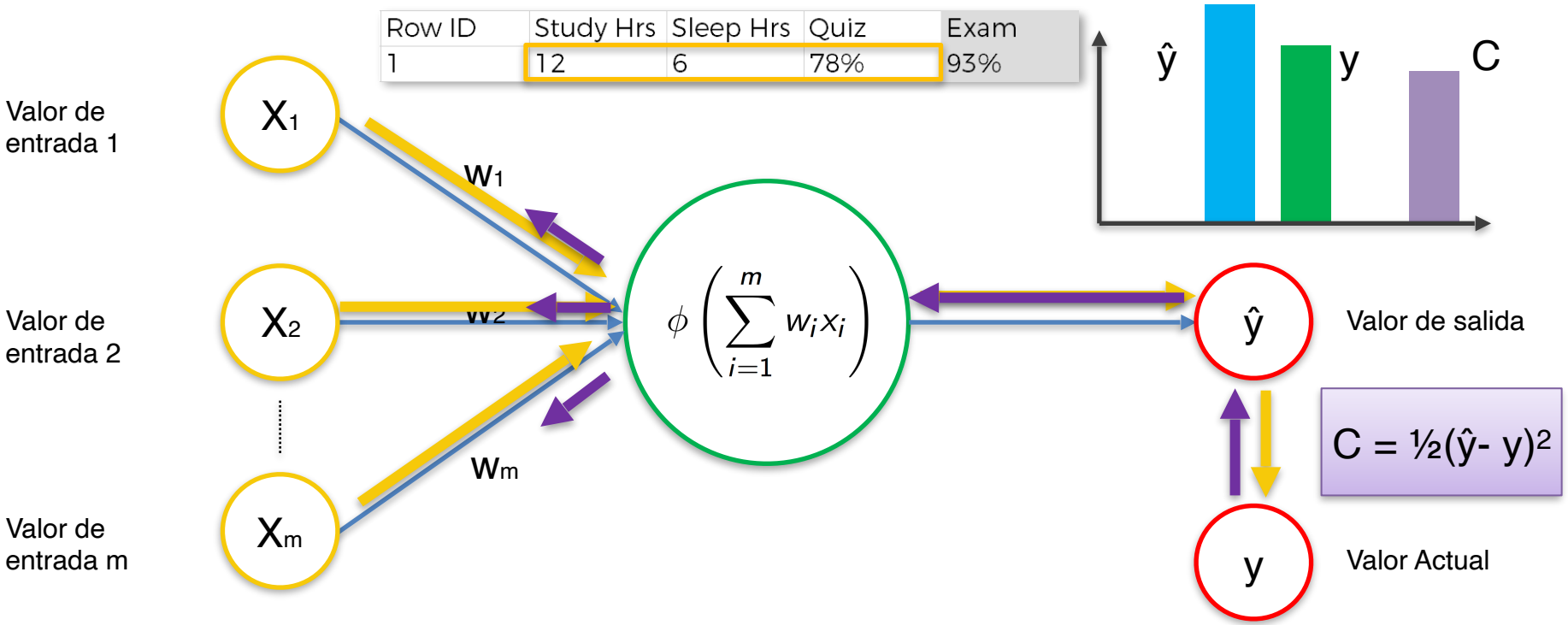
¿Cómo aprenden las Redes Neuronales?



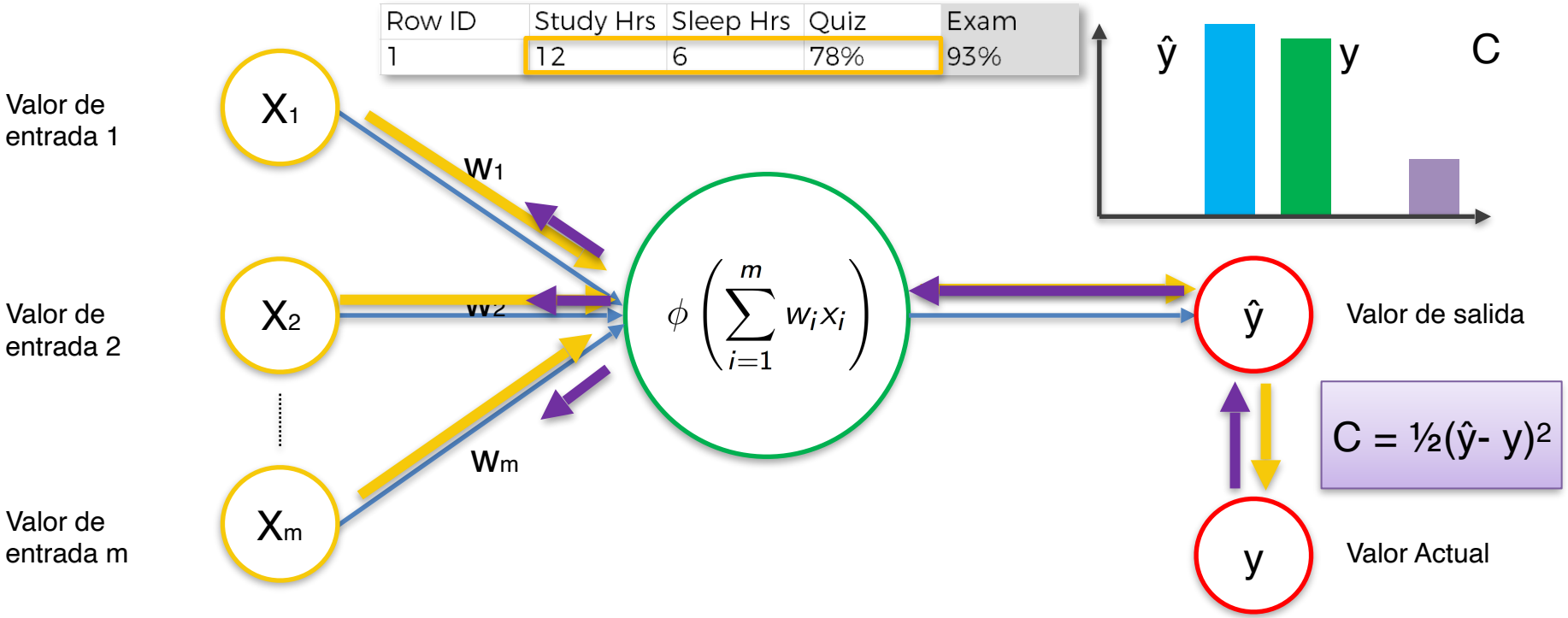
¿Cómo aprenden las Redes Neuronales?



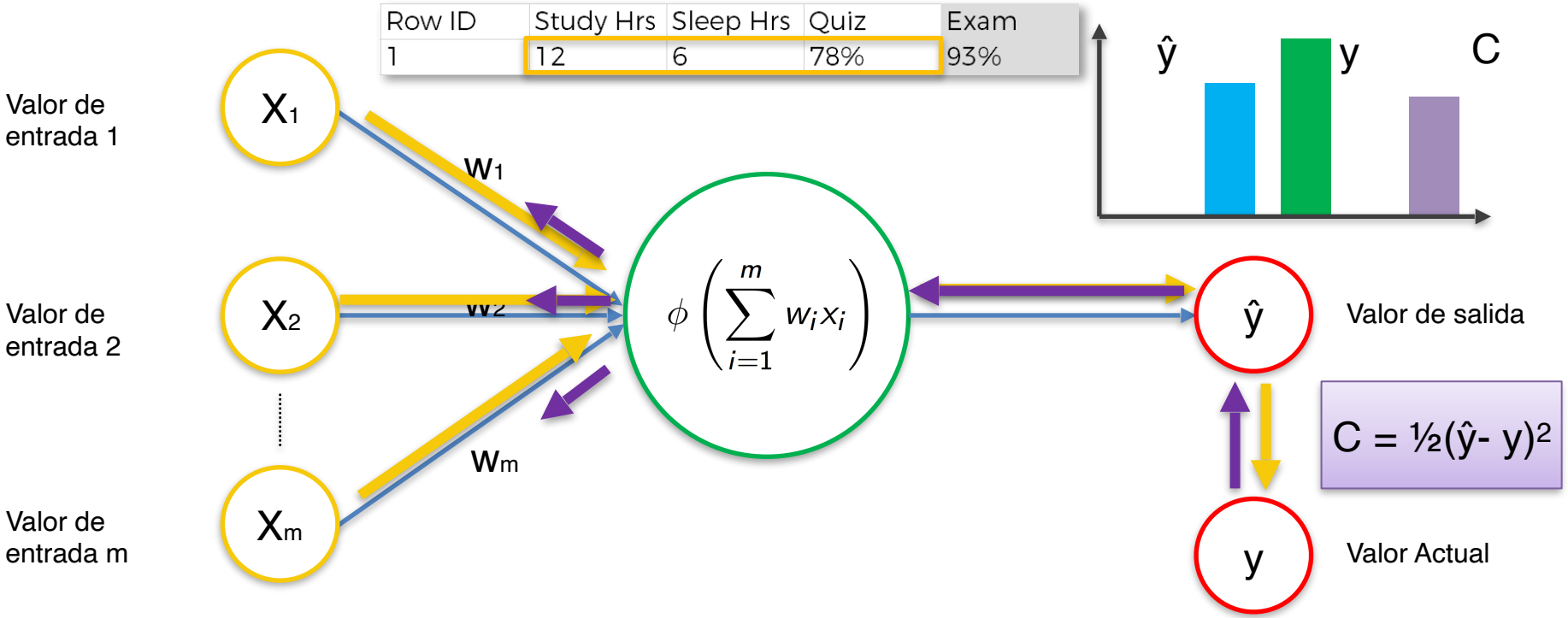
¿Cómo aprenden las Redes Neuronales?



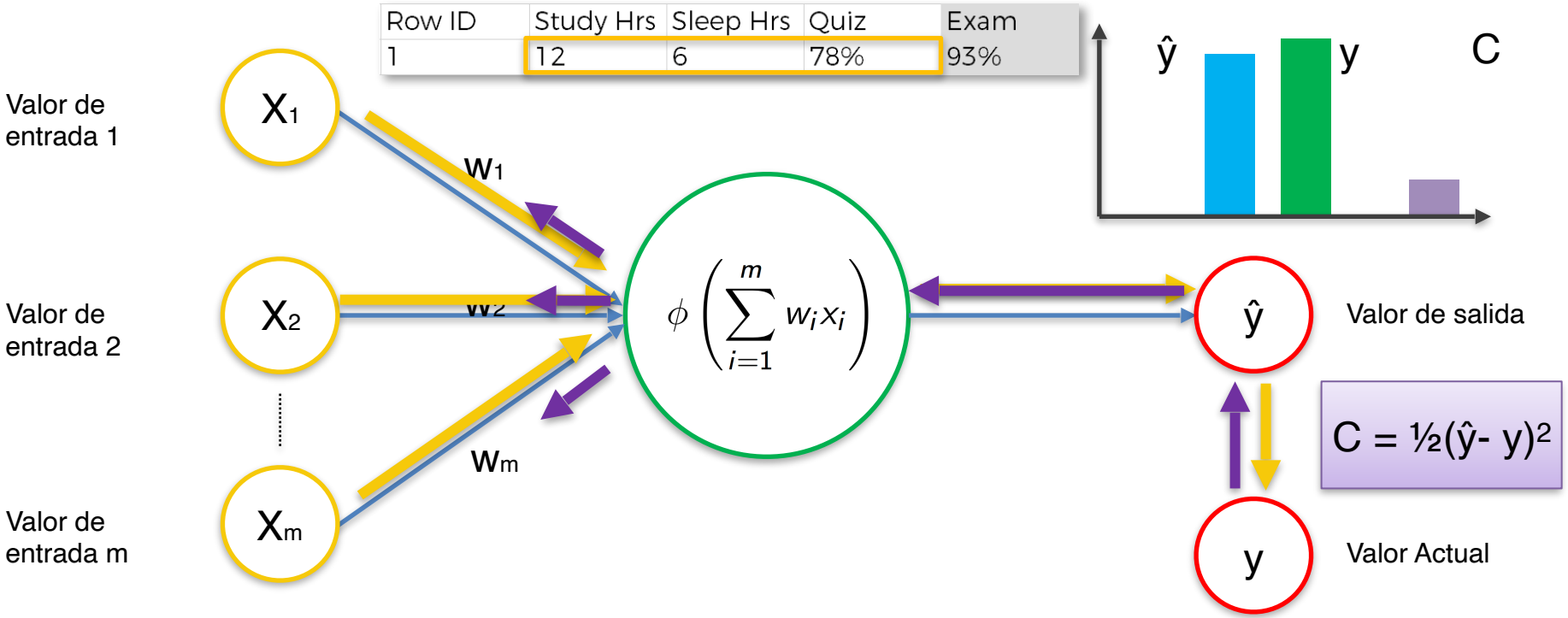
¿Cómo aprenden las Redes Neuronales?



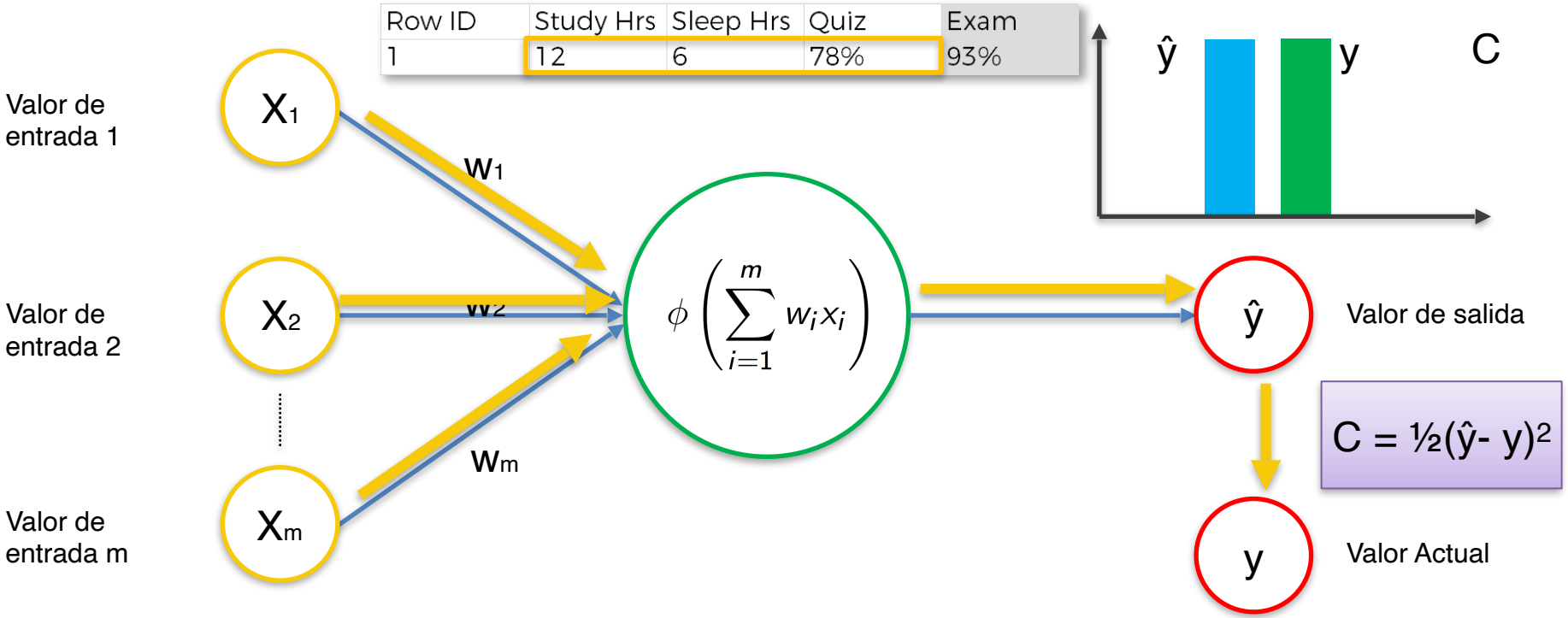
¿Cómo aprenden las Redes Neuronales?



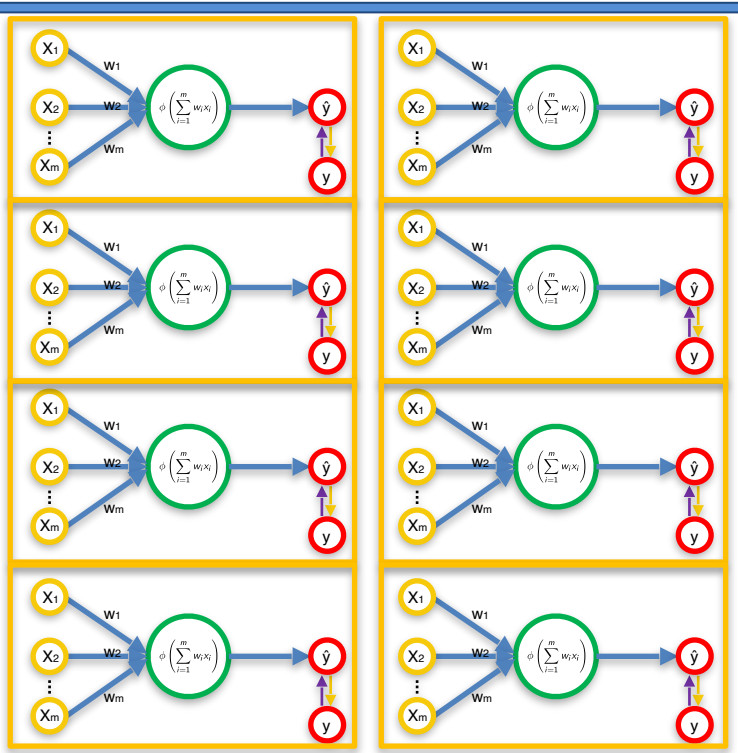
¿Cómo aprenden las Redes Neuronales?



¿Cómo aprenden las Redes Neuronales?

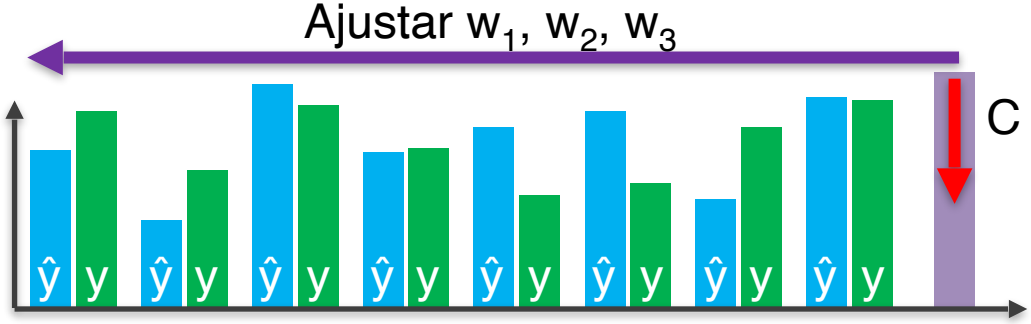


¿Cómo aprenden las Redes Neuronales?



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%

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



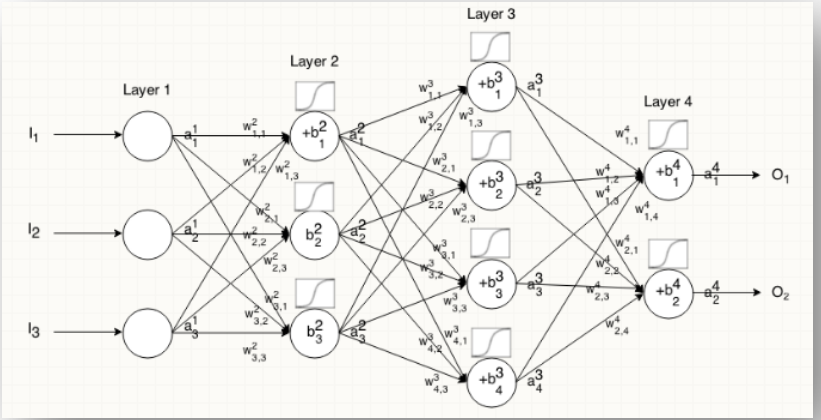
¿Cómo aprenden las Redes Neuronales?

Lecturas Adicionales:

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

CrossValidated (2015)

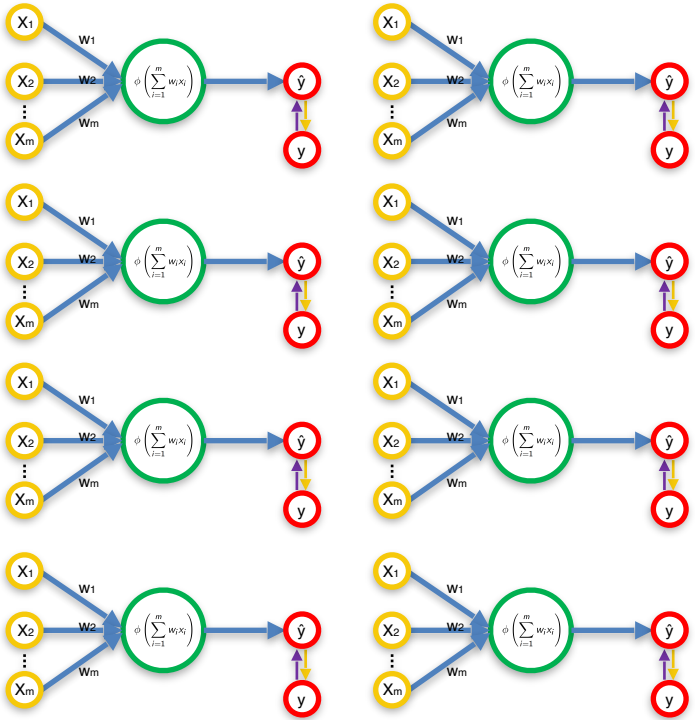
Link:



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

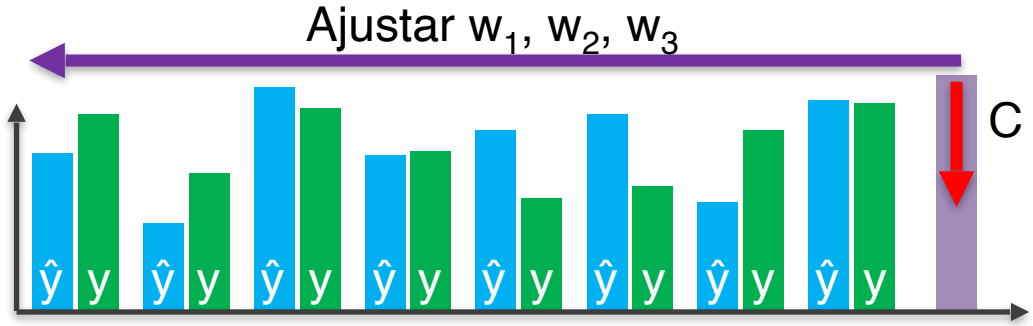
Gradiente Descendente

Gradiente Descendente

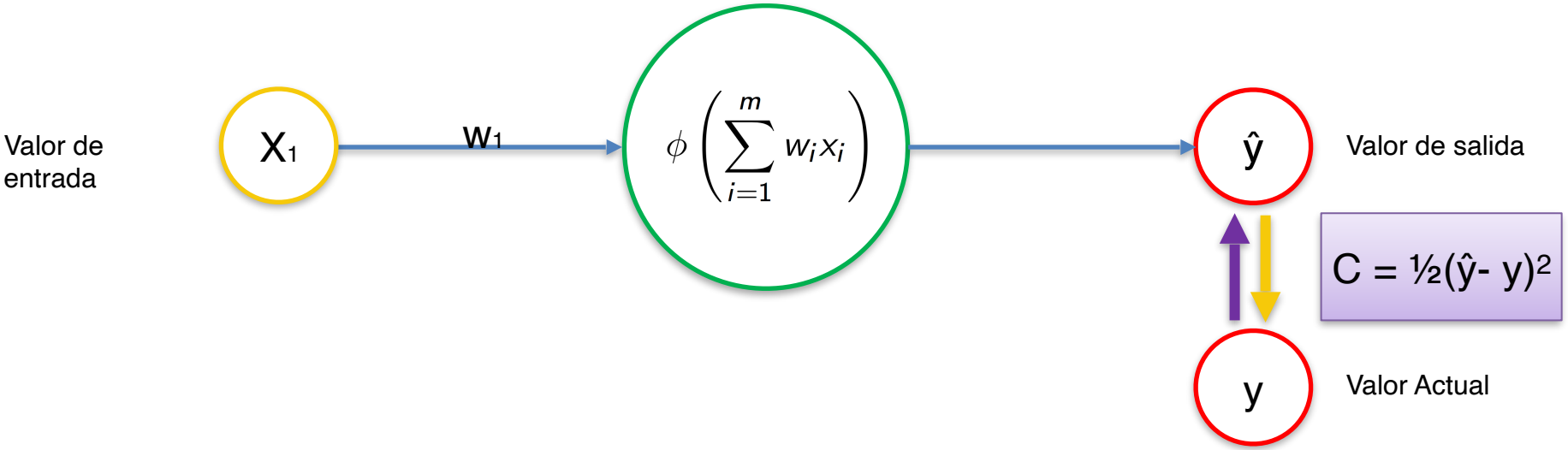


Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

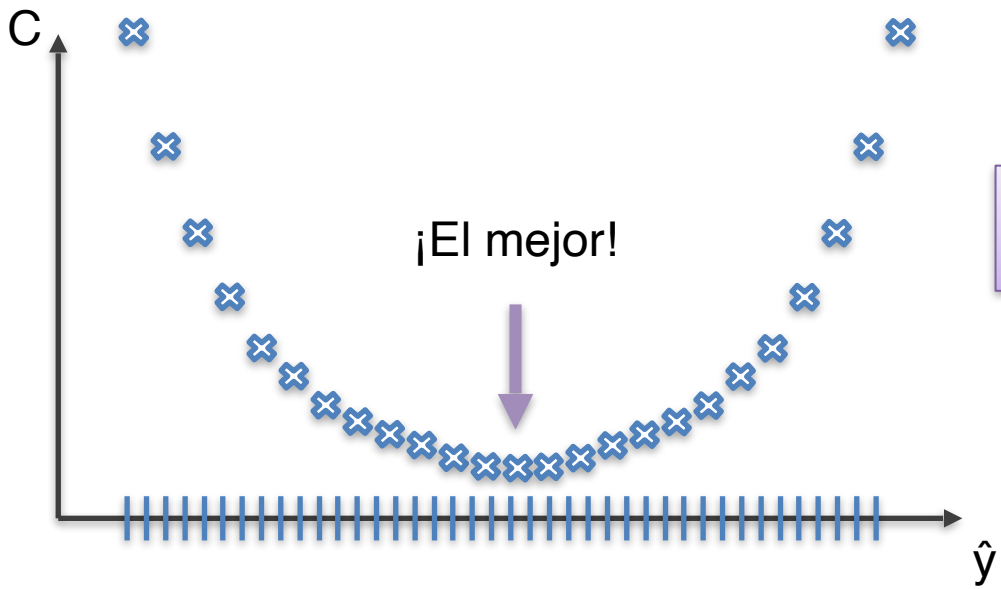
$$C = \sum \tfrac{1}{2}(\hat{y} - y)^2$$



Gradiente Descendente



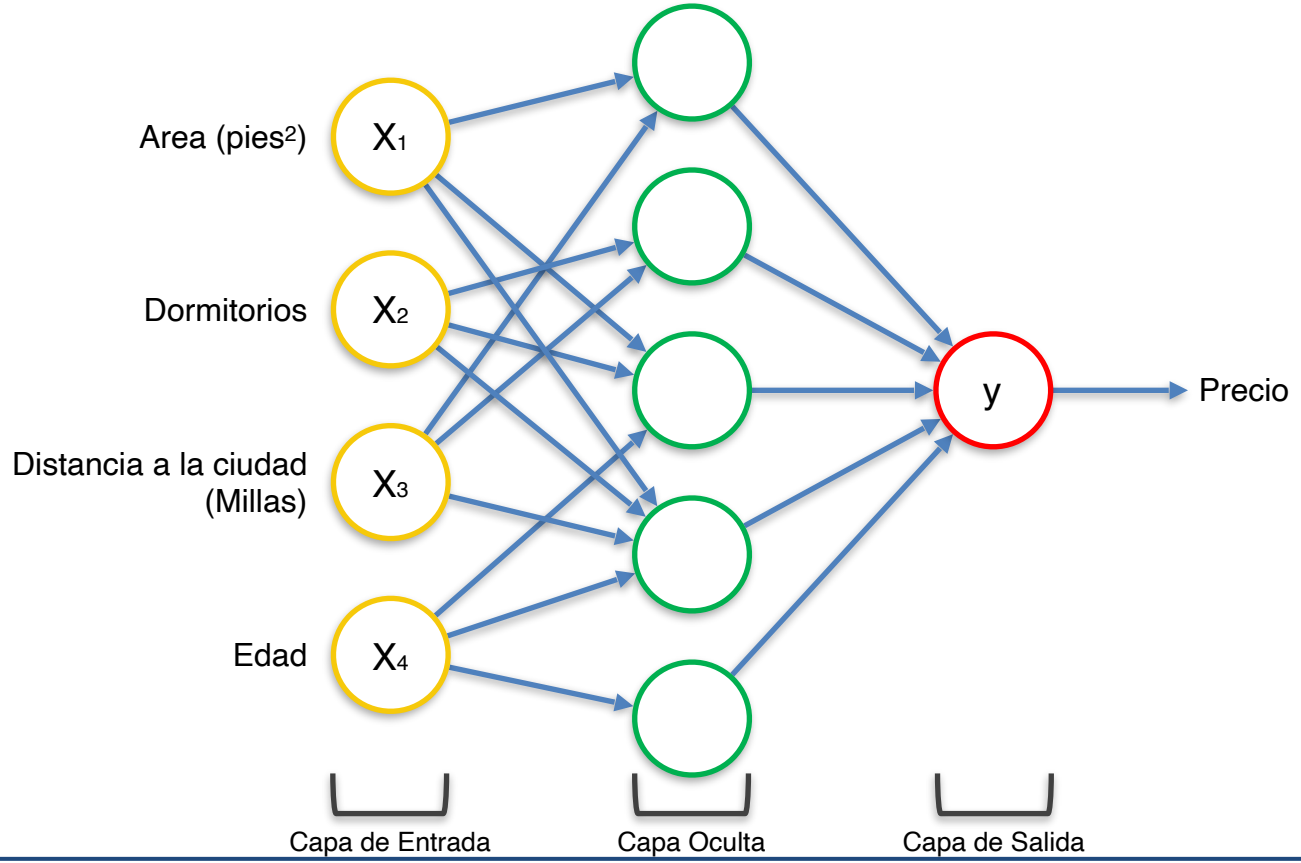
Gradiente Descendente



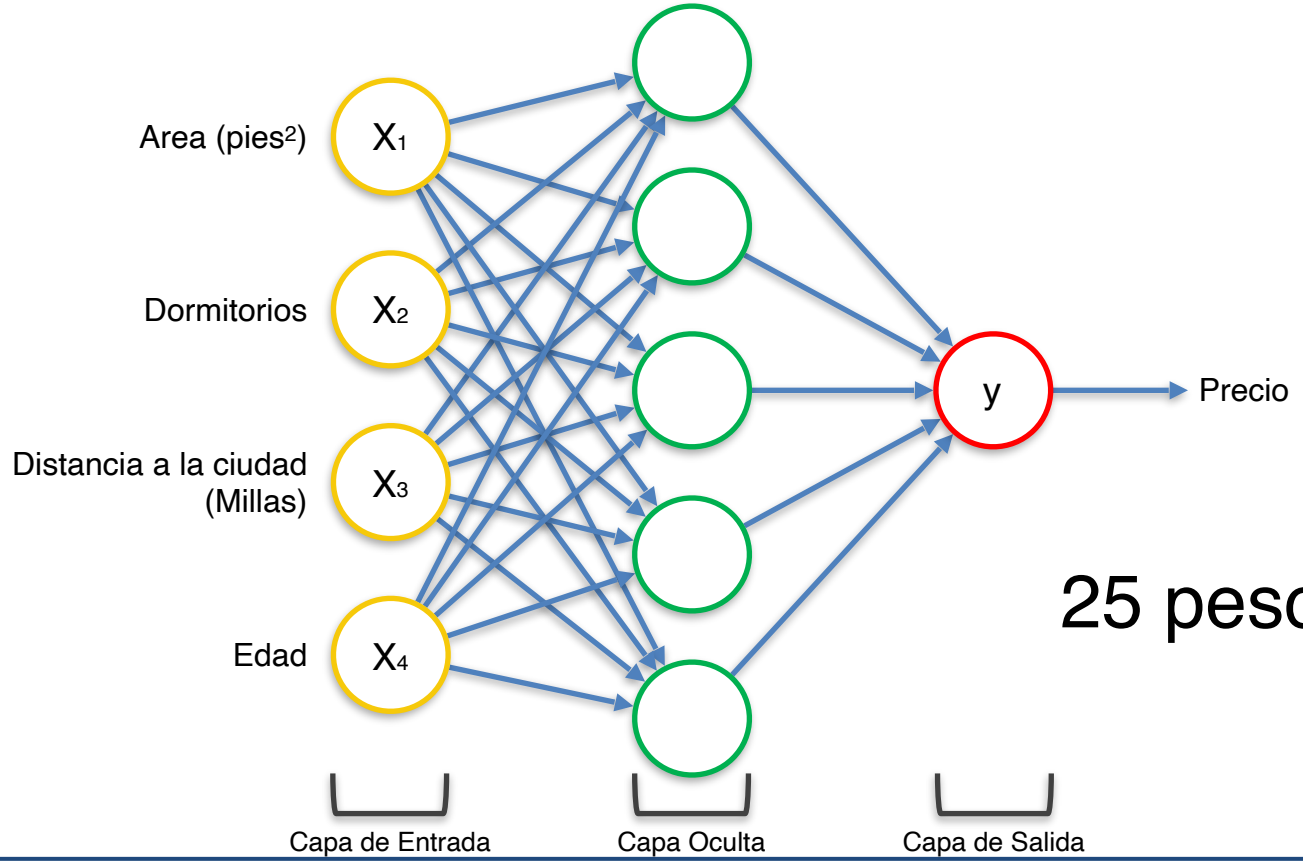
$$C = \frac{1}{2}(\hat{y} - y)^2$$

La Maldición de la Dimensión

Gradiente Descendente



Gradiente Descendente



Gradiente Descendente

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

Sunway TaihuLight: El Super Ordenador más potente del mundo

93 PFLOPS

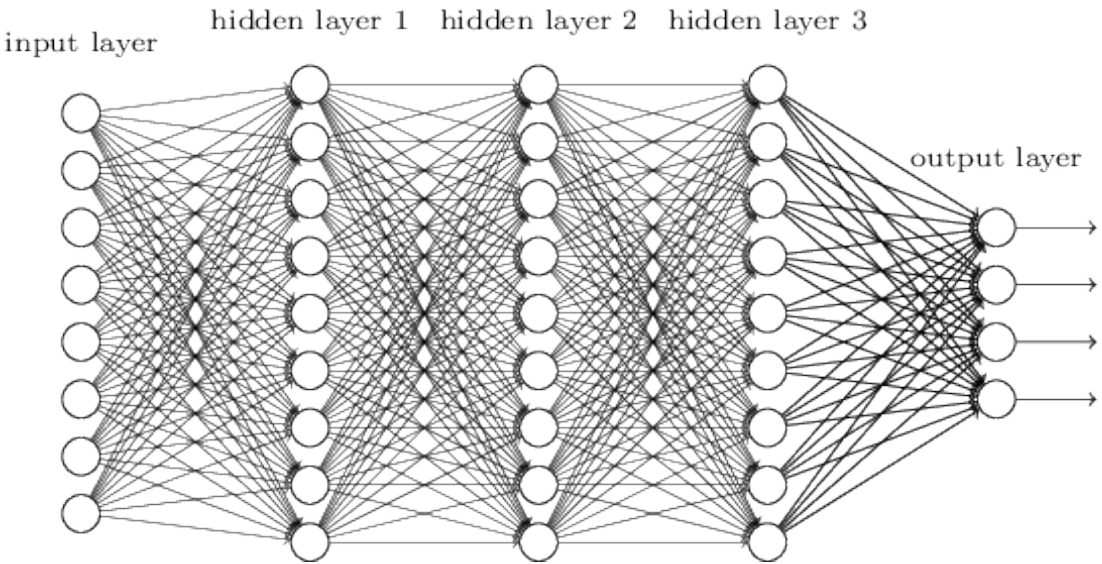
93×10^{15}

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

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

$= 3.42 \times 10^{50}$ años

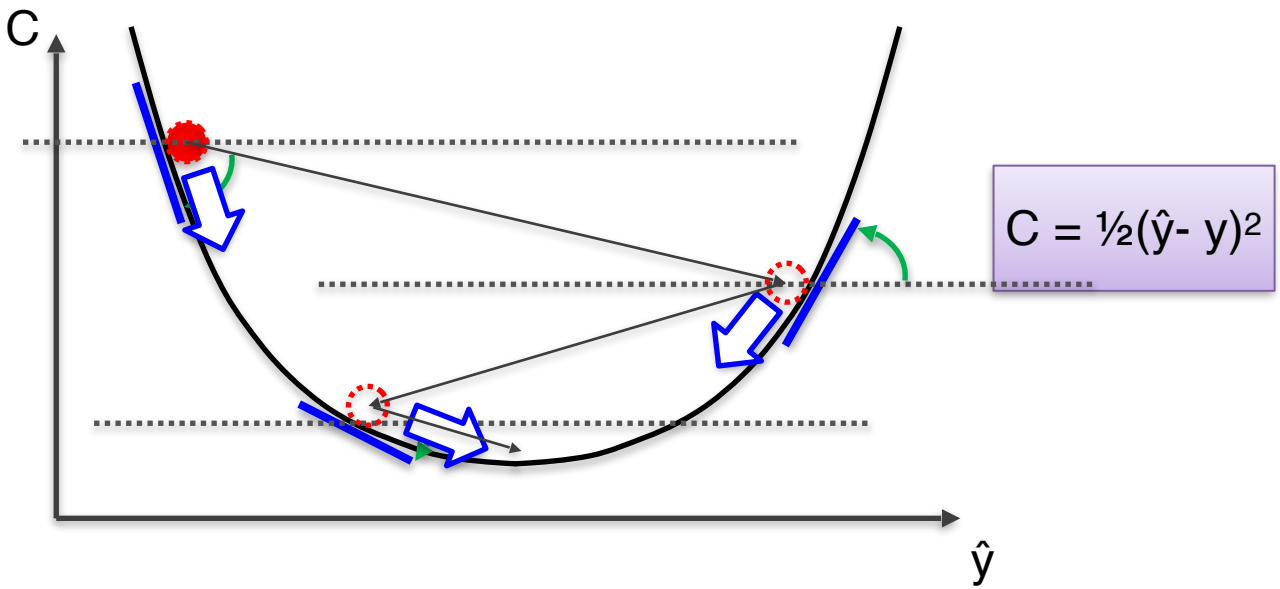
Gradiente Descendente



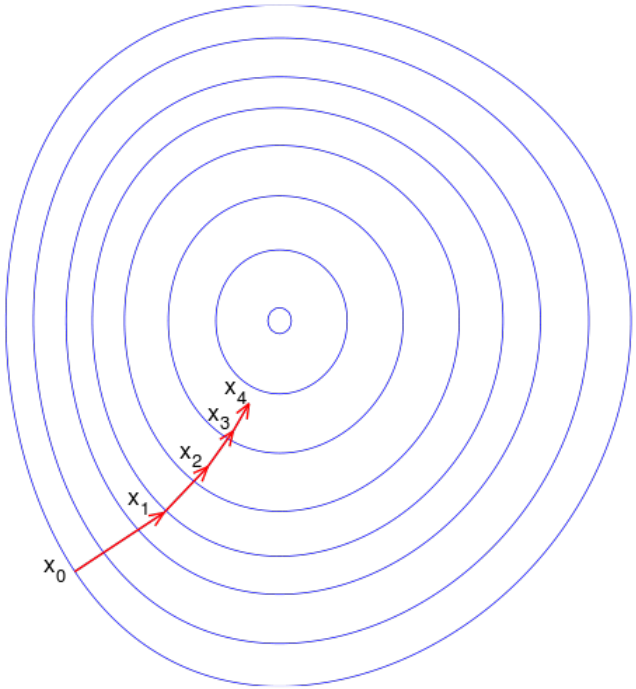
ImEdad Source: neuralnetworksanddeeplearning.com

Gradiente Descendente

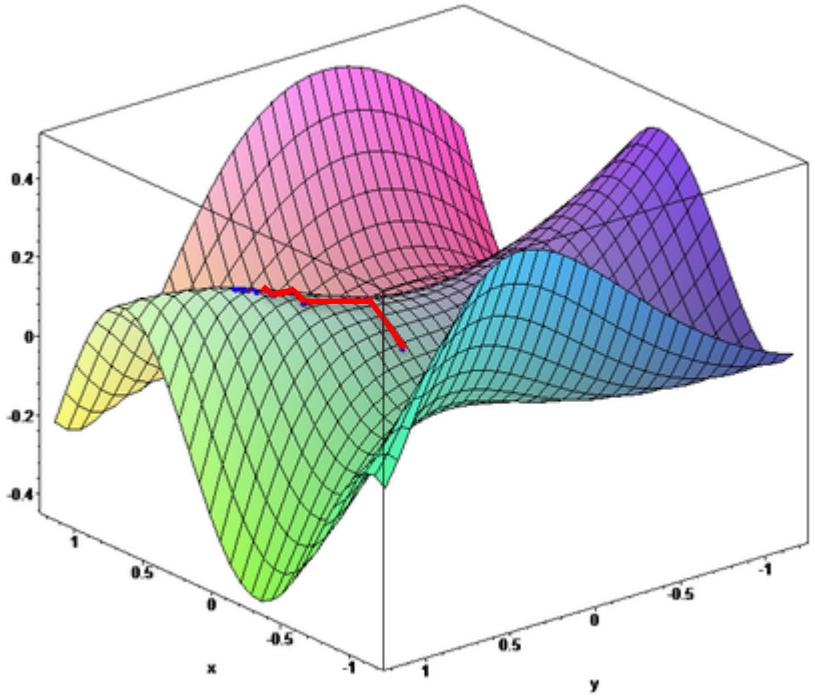
Gradiente Descendente

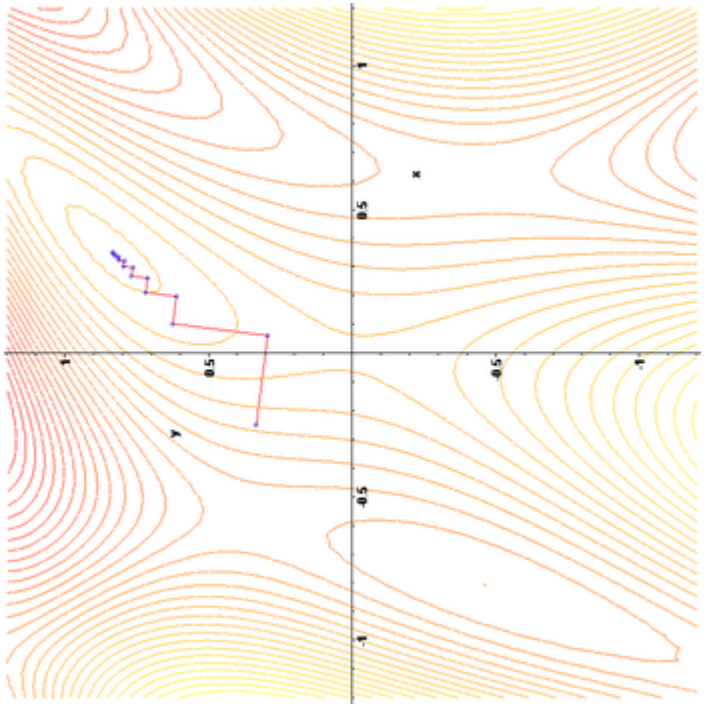


Gradiente Descendente



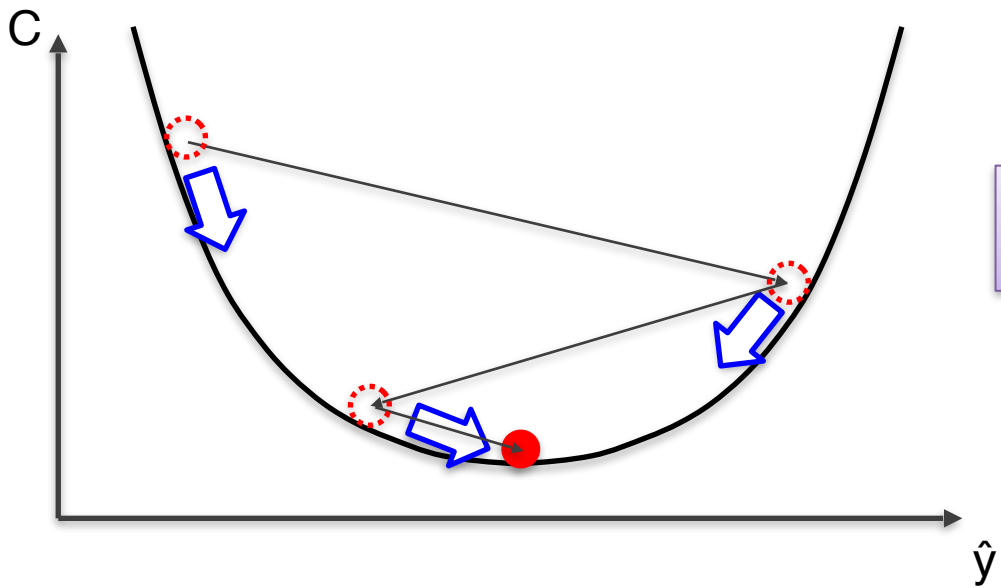
Gradiente Descendente





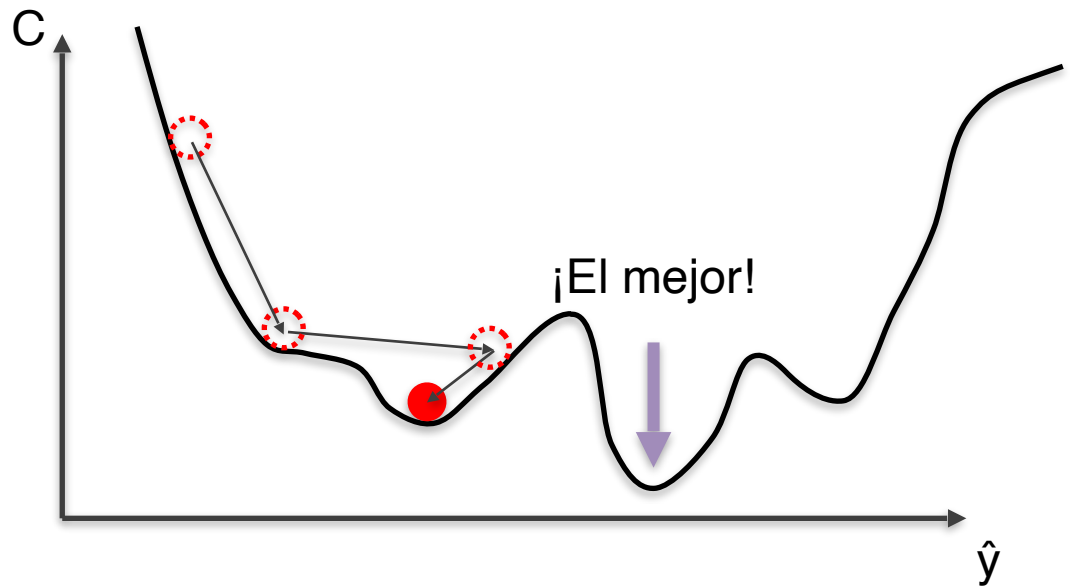
Gradiente Descendente Estocástico

Gradiente Descendente Estocástico

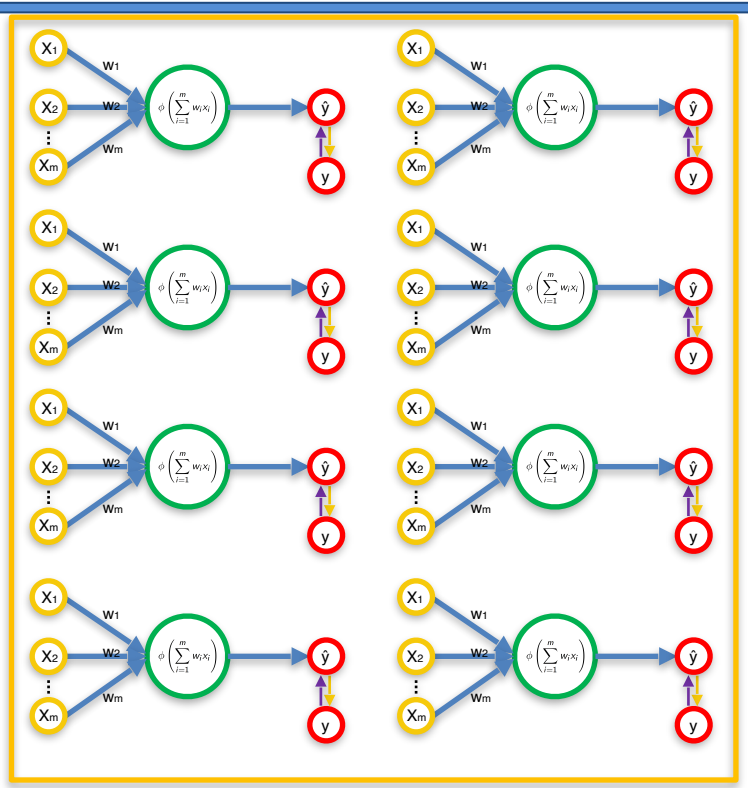


$$C = \frac{1}{2}(\hat{y} - y)^2$$

Gradiente Descendente Estocástico

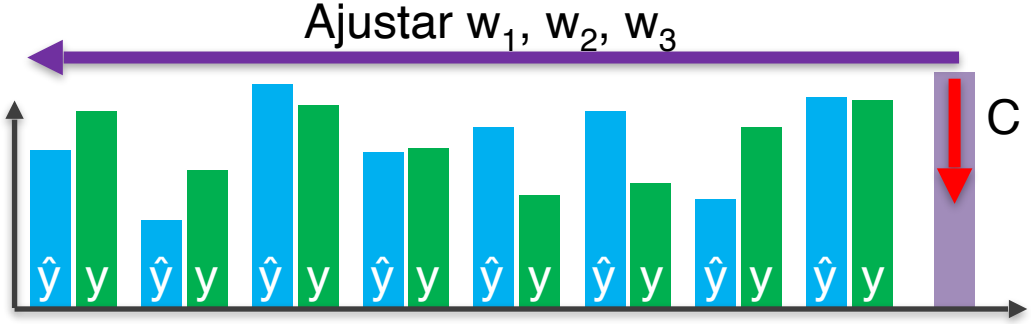


Gradiente Descendente Estocástico

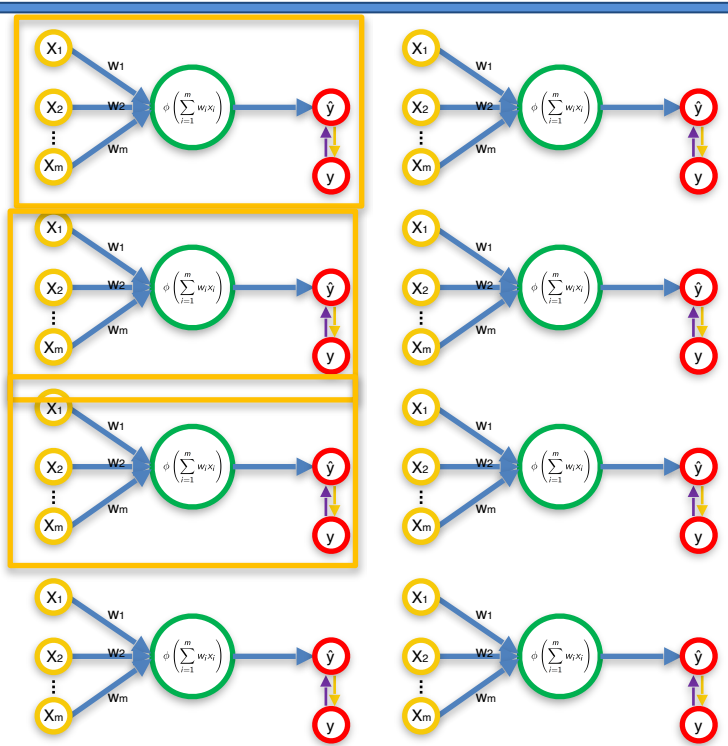


Row ID	Study Hrs	Sleep Hrs	Quiz	Exam
1	12	6	78%	93%
2	22	6.5	24%	68%
3	115	4	100%	95%
4	31	9	67%	75%
5	0	10	58%	51%
6	5	8	78%	60%
7	92	6	82%	89%
8	57	8	91%	97%

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

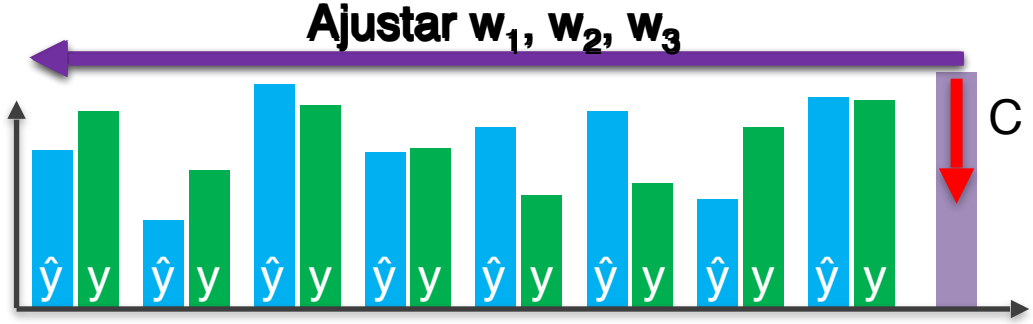


Gradiente Descendente Estocástico



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%

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



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%

Act w's

Gradiente Descendente Por Bloques

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

Gradiente Descendente Estocástico

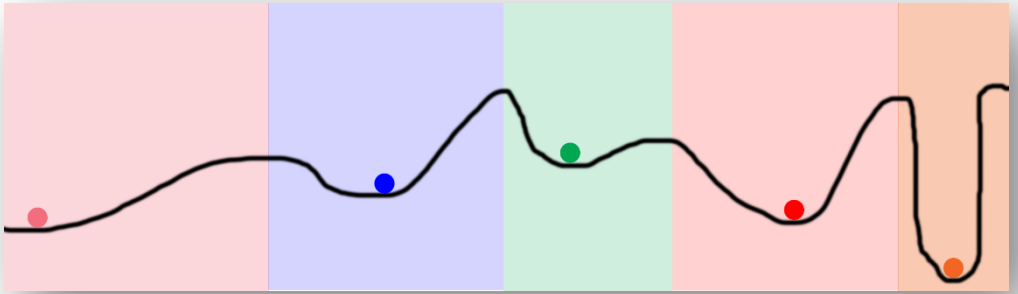
Gradiente Descendente Estocástico

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/

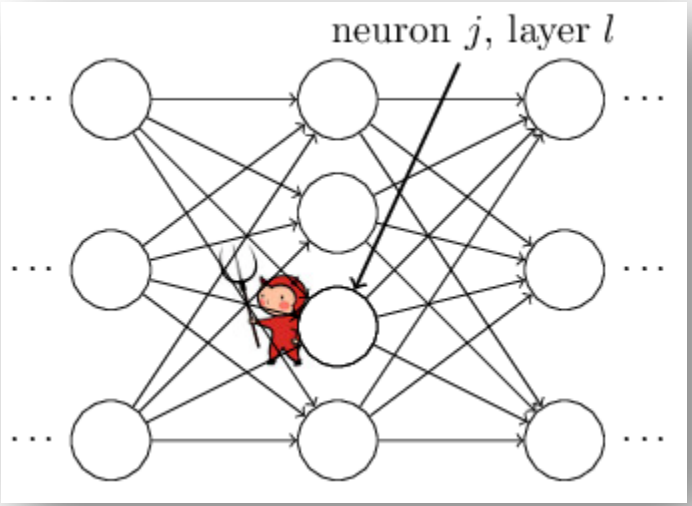
Gradiente Descendente Estocástico

Lecturas Adicionales:

Neural Networks and Deep Learning

Michael Nielsen (2015)

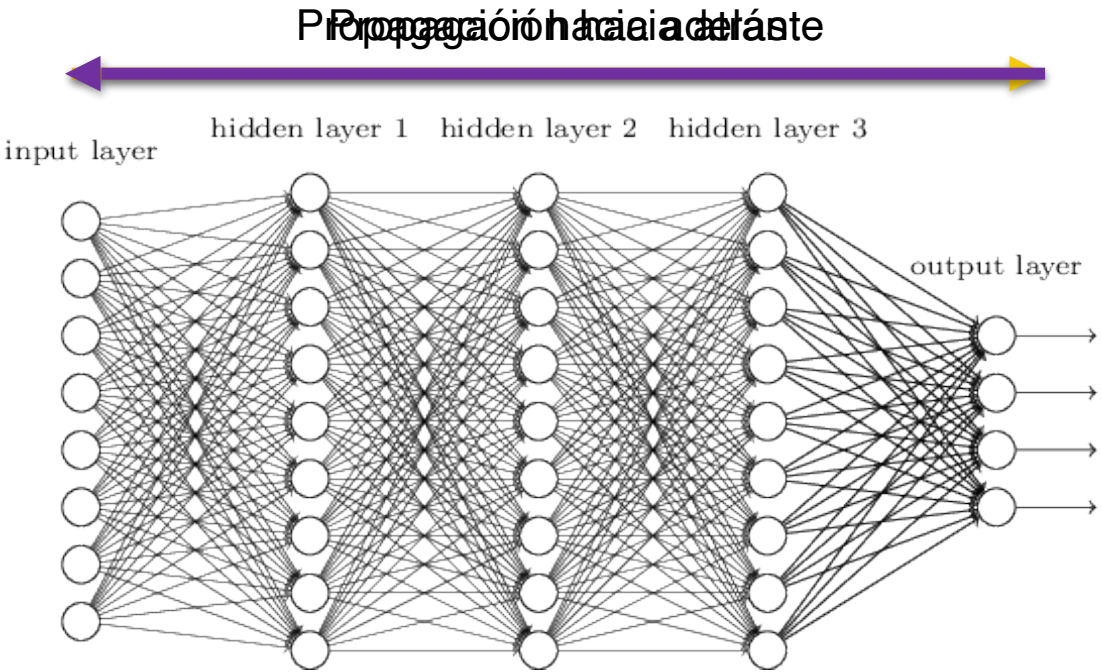
Link:



http://neuralnetworksanddeeplearning.com/chap2.html

Propagación hacia atrás

Gradiente Descendente



ImEdad Source: neuralnetworksanddeeplearning.com

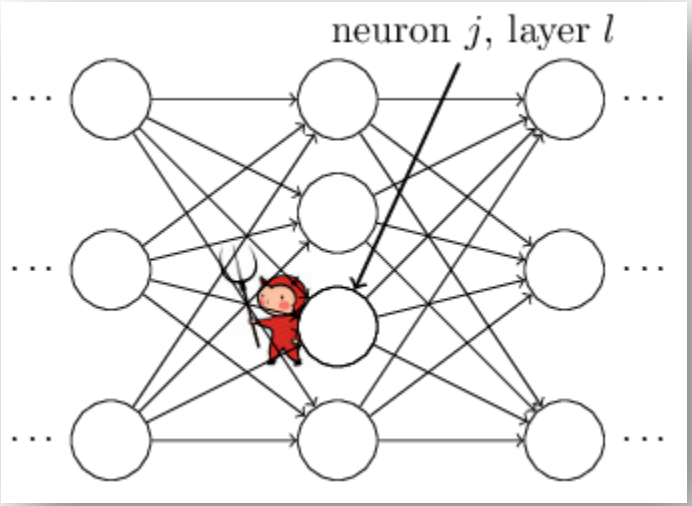
Gradiente Descendente Estocástico

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.

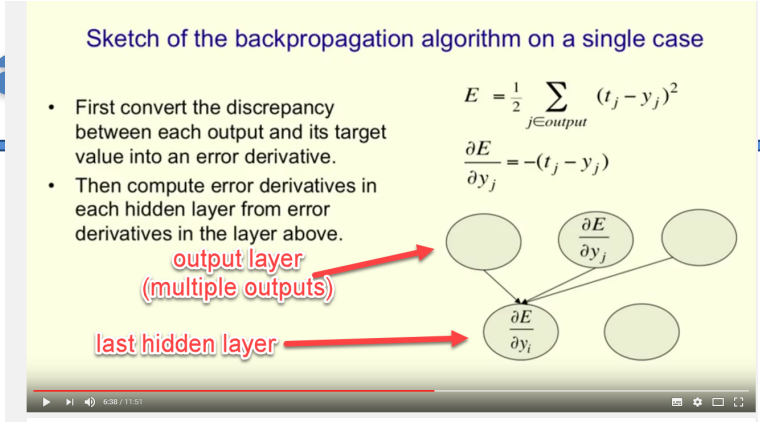
Paso 5: Propagación hacia atrás: de derecha a izquierda, propagando el error hacia atrás. Se actualizan los pesos según lo responsables que Sean del error. El ratio de aprendizaje gobierna cuánto deben actualizarse los pesos.

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 est



Tres formas de afrontar el problema

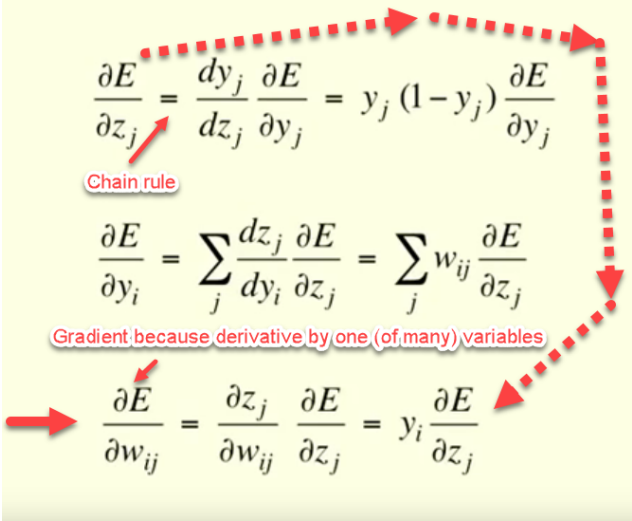
1. **Fuerza Bruta <- La Maldición de la Dimensión**
2. **Perturbaciones aleatorias a los pesos pesos (~evolución) – ineficiente. "Más eficiente con un factor del número de conexiones" G.Hinton**
3. **Propagación hacia atrás**

Video de Geoffrey Hinton's:
https://www.youtube.com/watch?v=xfPz92B0rv8

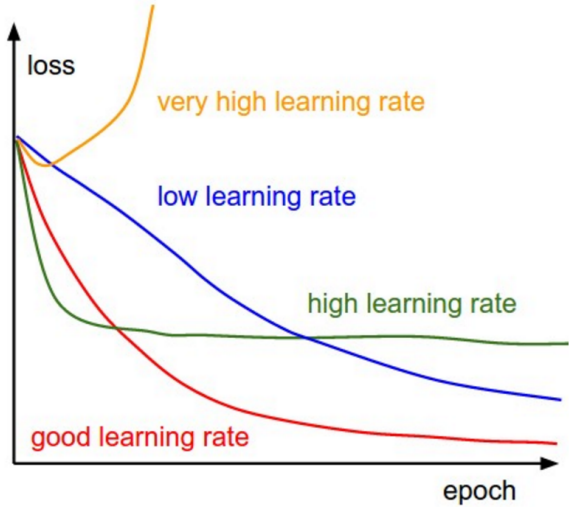
El fundamento de la propagación hacia atrás es tomar las derivadas del error de una capa y con ellas calcular las derivadas del error de la siguiente.

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.



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 a_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
```

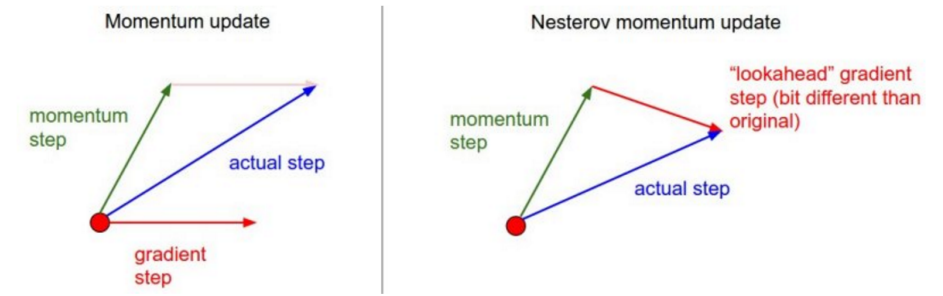
Here we see an introduction of a `v` variable that is initialized at zero, and an additional hyperparameter (`mu`). As an unfortunate misnomer, this variable is in optimization referred to as *momentum* (its typical value is about 0.9), but its physical meaning is more consistent with the coefficient 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

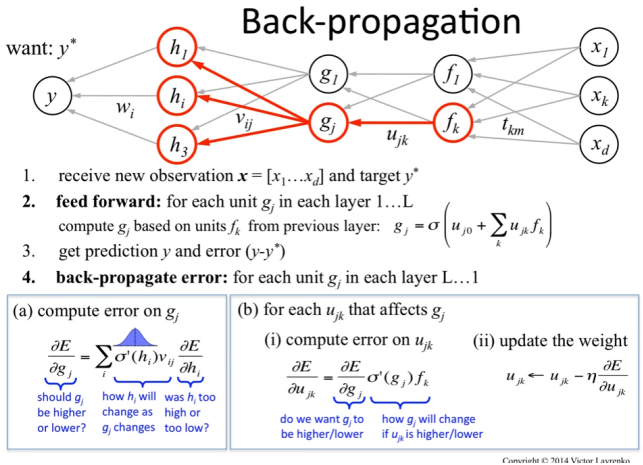
Nesterov Momentum is a slightly different version of the momentum update has recently been gaining popularity. It enjoys stronger theoretical converge guarantees for convex functions and in practice it also consistenly works slightly better than standard momentum.

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

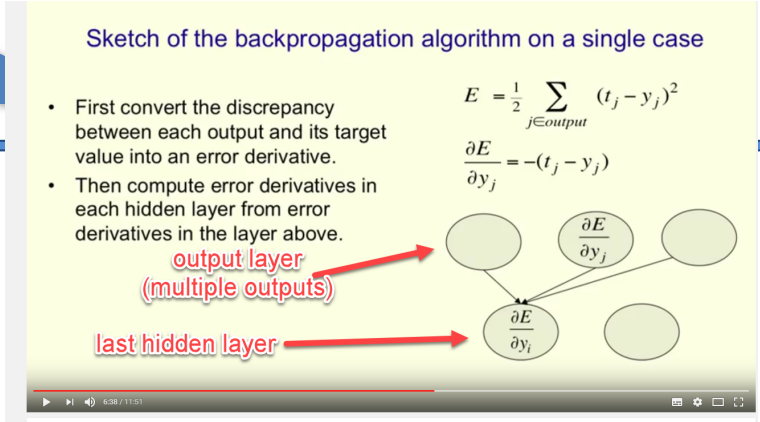


https://www.youtube.com/watch?v=An5z8lR8asY

El video es un inicio pero todavía no entiendo como se decide h (h1, h2, h3) 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.

Propagación hacia atrás (cont.)

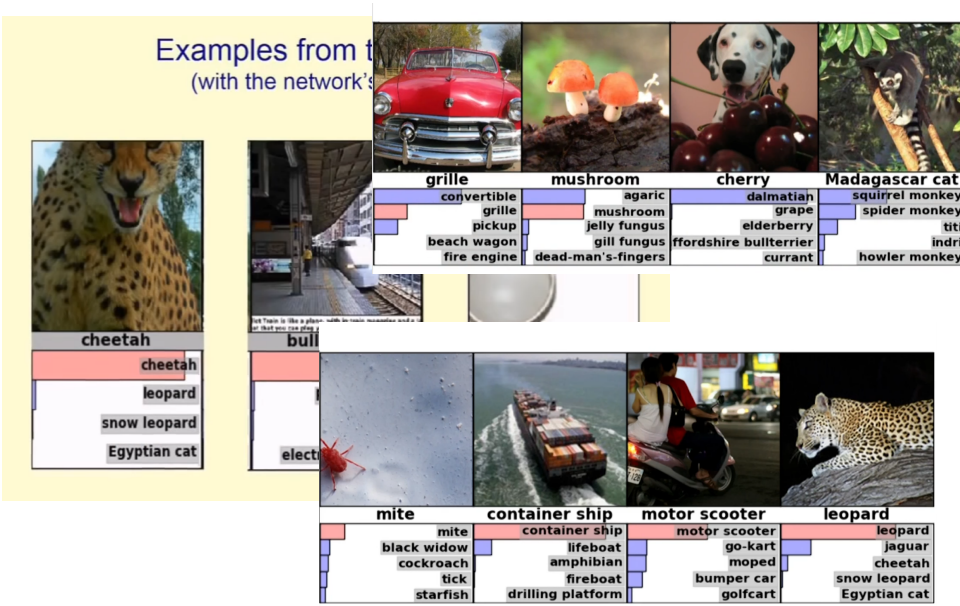
https://www.youtube.com/watch?v=EInQoVLg_UY
~3min in
“The Propagación hacia atrás algorithm is doing what I just said [tweaking the pesos on a given input set] but it *figures out for all the pesos at the same time*, whether you should increase them a little bit or decrease them a little bit so that 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.)

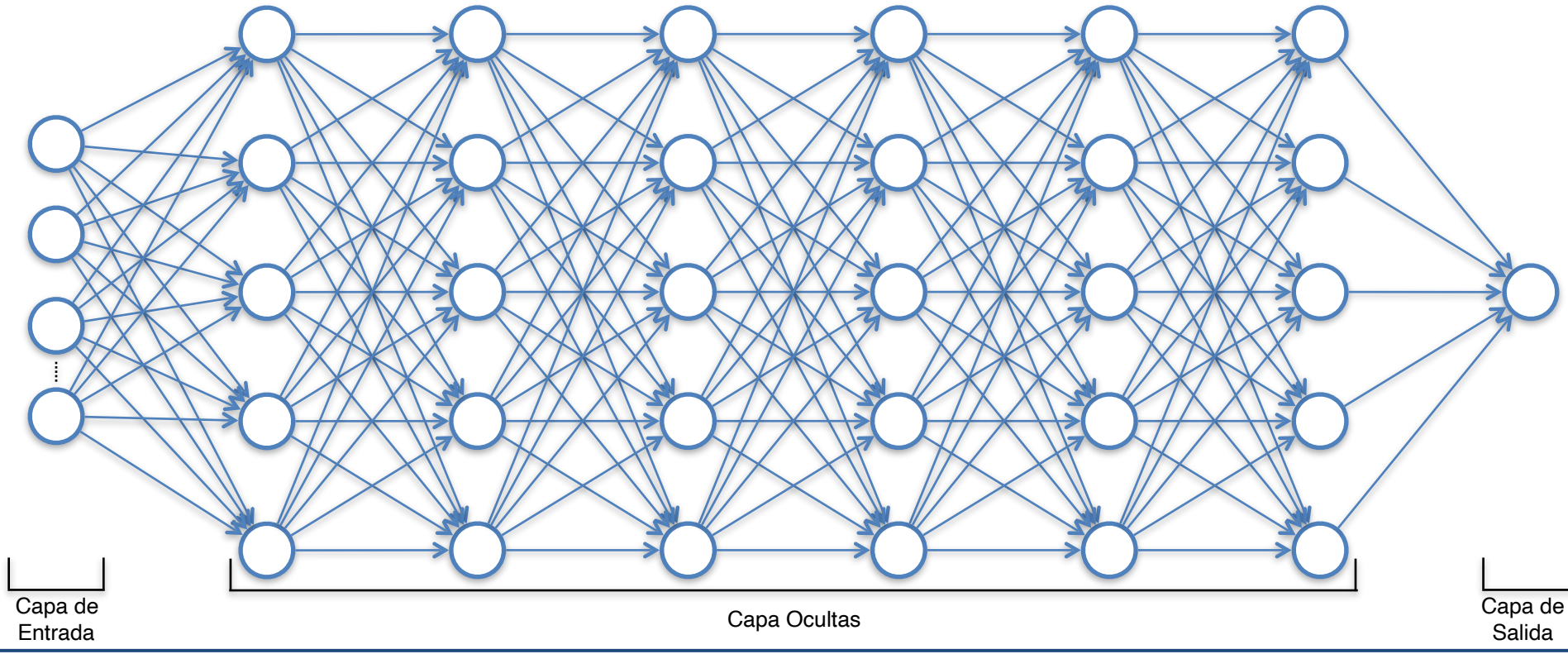
https://www.youtube.com/watch?v=XG-dwZMc7Ng

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?

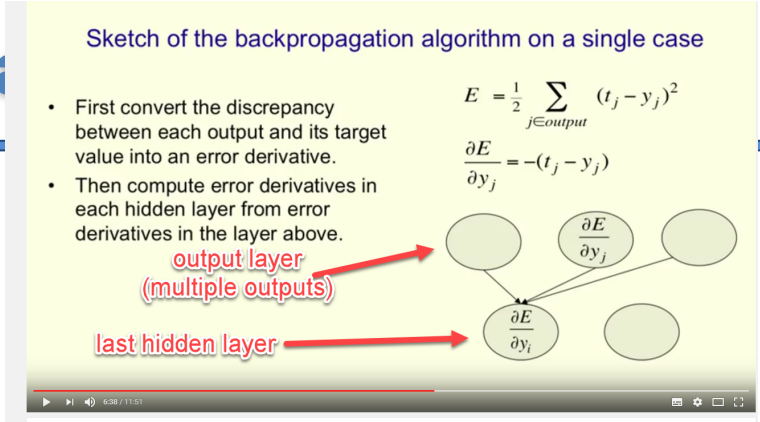
Deep Learning es aplicar los pasos 1 al 7 a todo esto…

Esto es Deep Learning



Matemáticas de la Propagación hacia atrás (avanzado)

Muy bien, ¿pero dónde est



Tres formas de afrontar el problema

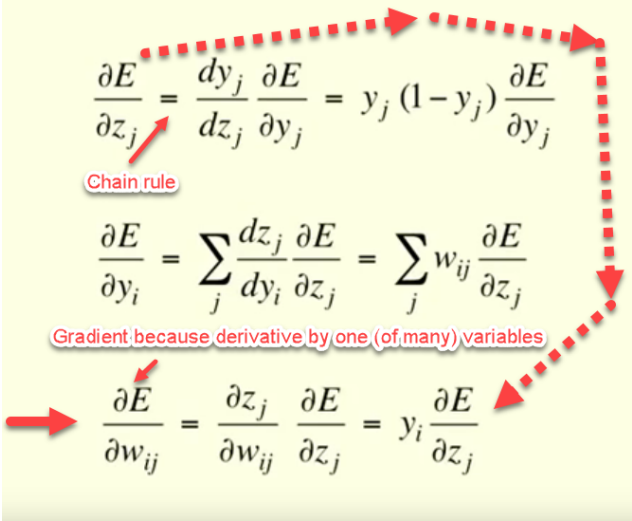
1. **Fuerza Bruta <- La Maldición de la Dimensión**
2. **Perturbaciones aleatorias a los pesos pesos (~evolución) – ineficiente. "Más eficiente con un factor del número de conexiones" G.Hinton**
3. **Propagación hacia atrás**

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.



BONUS

ANN

K-Fold cross validation

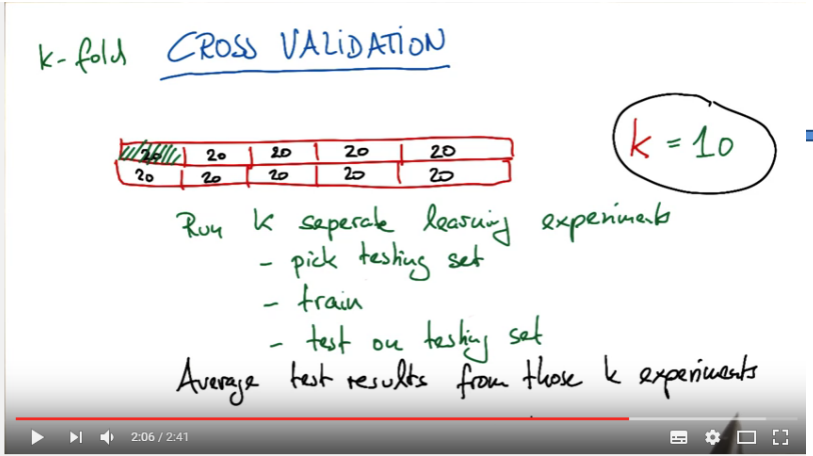
https://www.youtube.com/watch?v=TIgfjmp-4BA
https://www.youtube.com/watch?v=ADNFKiAjmWA

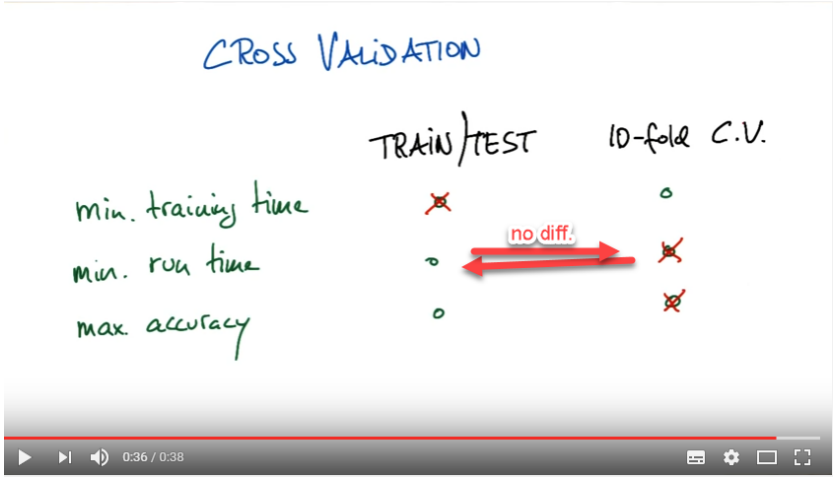
Más en las transparencias

Fácil, ¿no?

Pero ¿cómo hacemos la "Media" después?

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





ANN

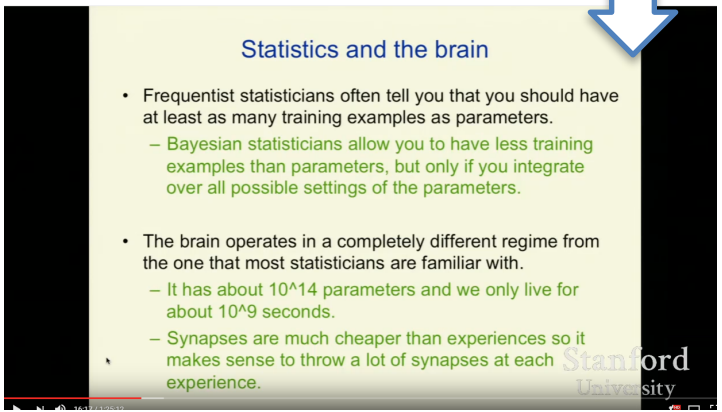
COMMENTS ABOUT SIZE

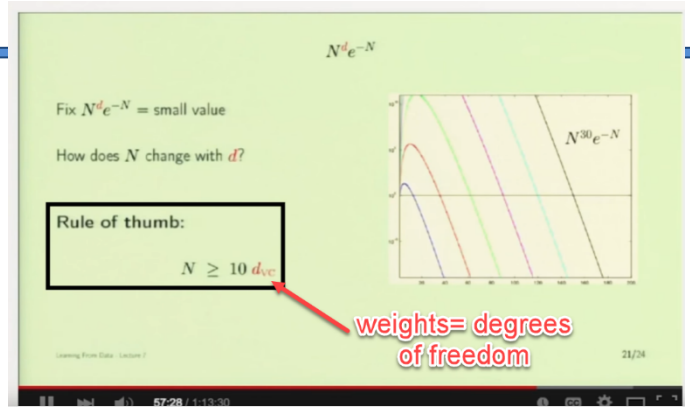
Mitigating over-fitting:

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

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

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



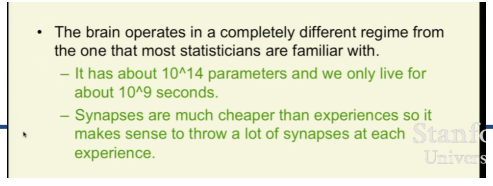


(One of Welch's videos – he references somebody else's video)

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.

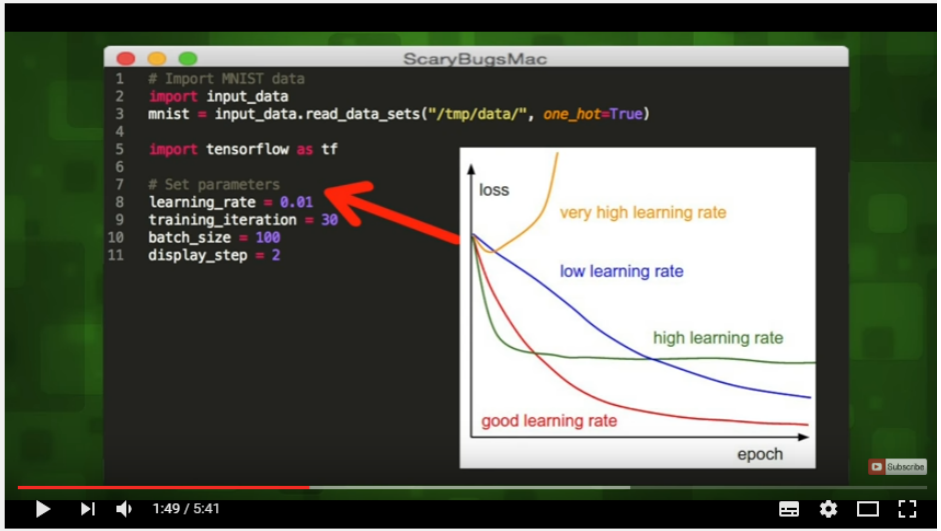
Also:



ANN

Learning Rate

~1m40s
https://www.youtube.com/watch?v=2FmcHiLCwTU
Also see Hadelin's slide abuot this



ANN

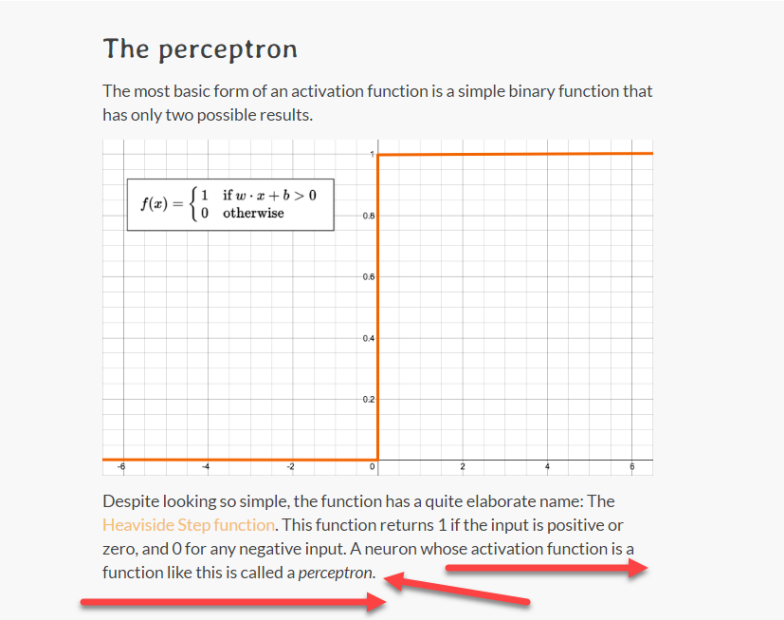
Dropout training

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

ANN

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

https://appliedgo.net/perceptron/



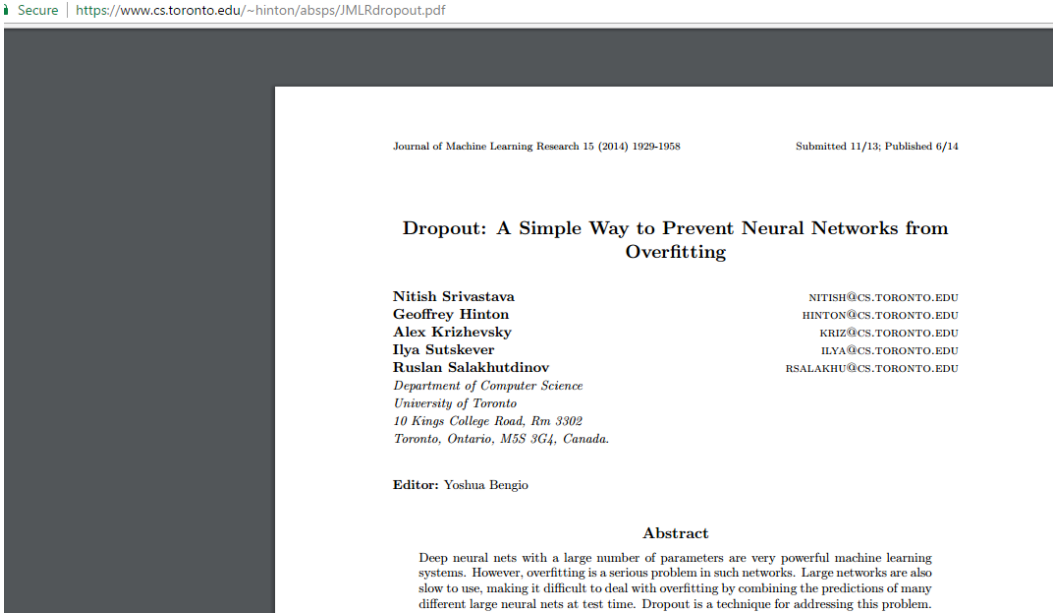
Dropout Intuition

Dropout

Dropout

~ Hinton's paper on dropout:

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



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.

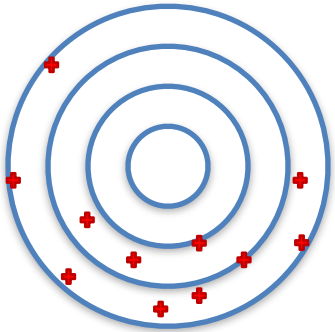
High Bias: when your model predictions are far from the real values.

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

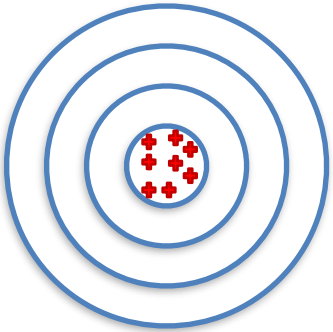
High Variance: when you run your model several times, the different predictions will vary a lot.

The Bias-Variance Tradeoff

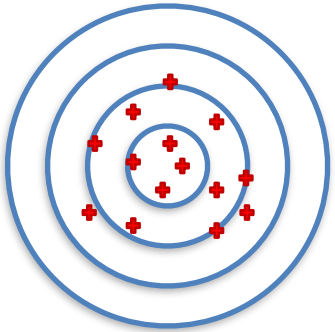
High Bias Low Variance



High Bias High Variance



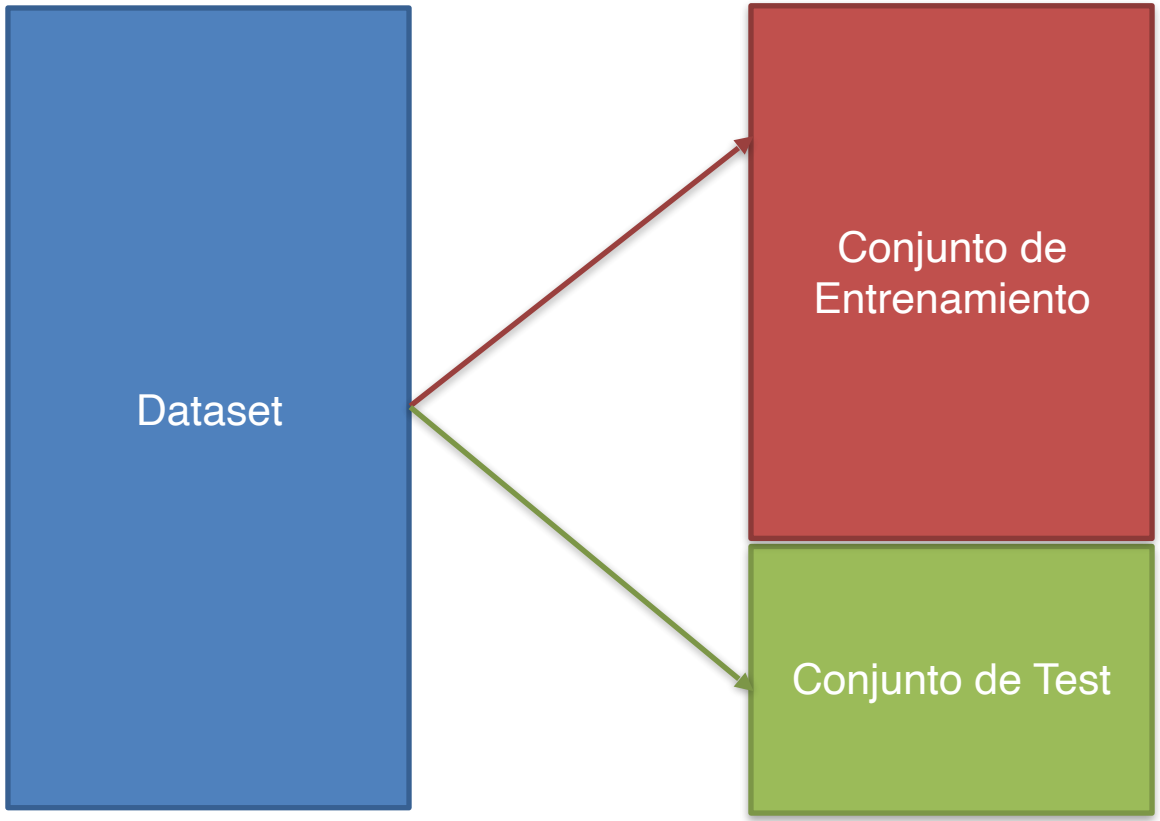
Low Bias Low Variance



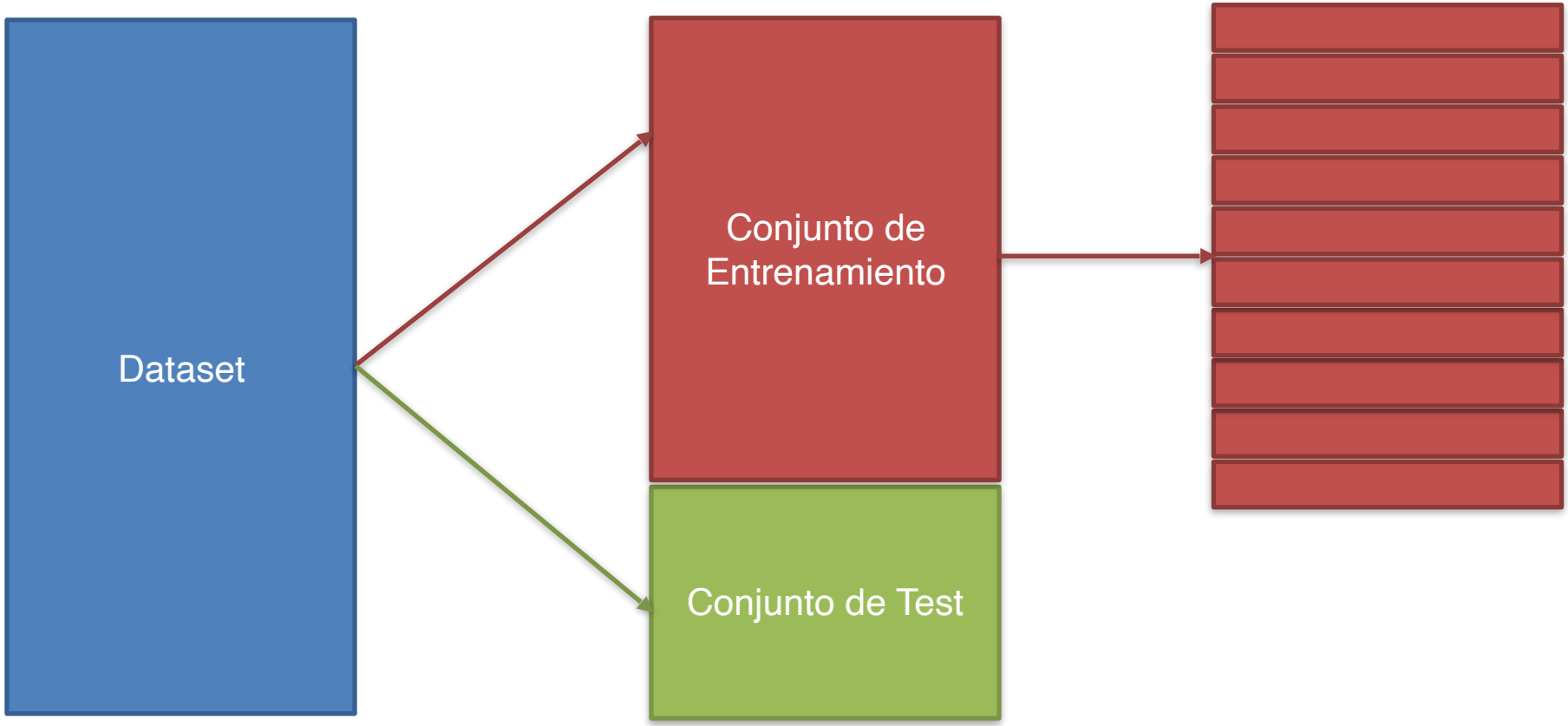
Low Bias High Variance

k-Fold Cross Validation

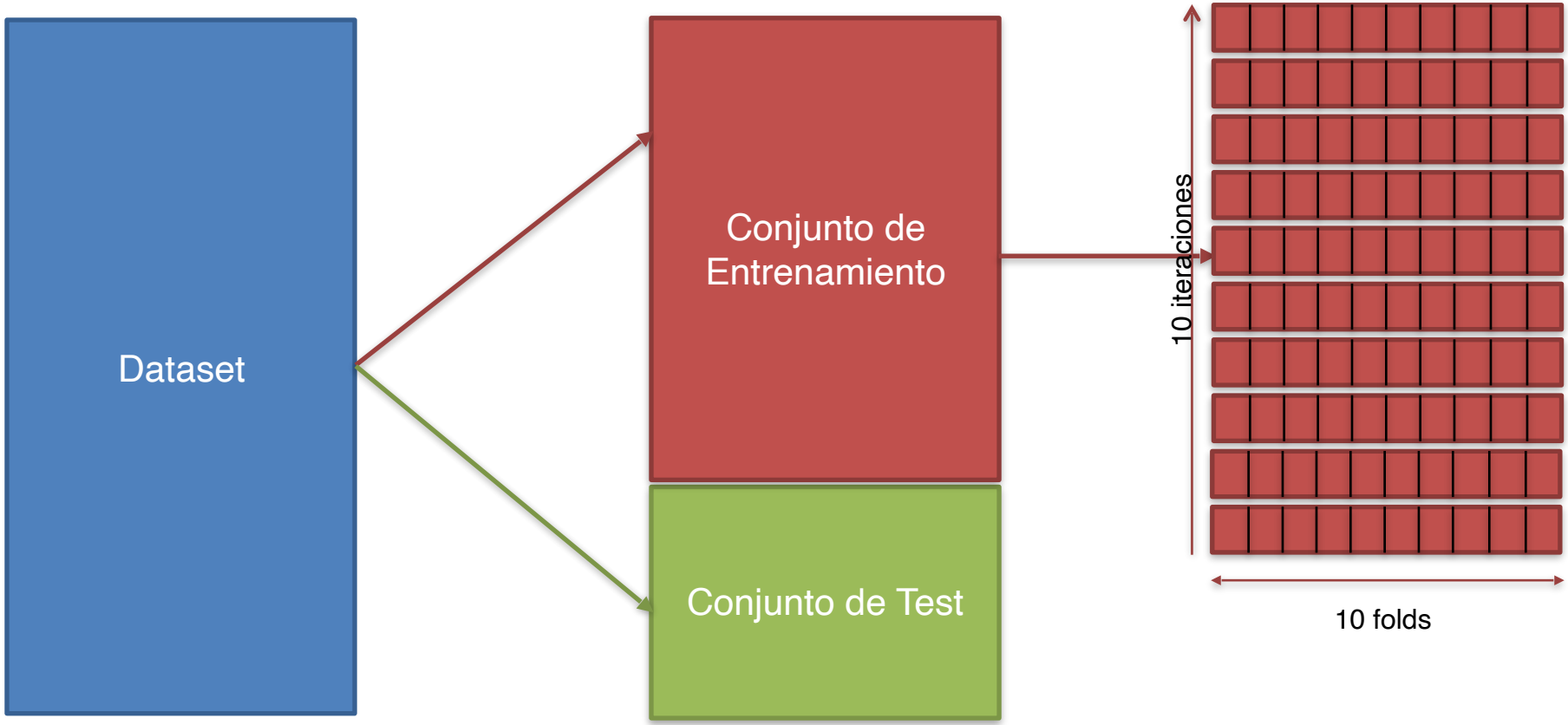
k-Fold Cross Validation



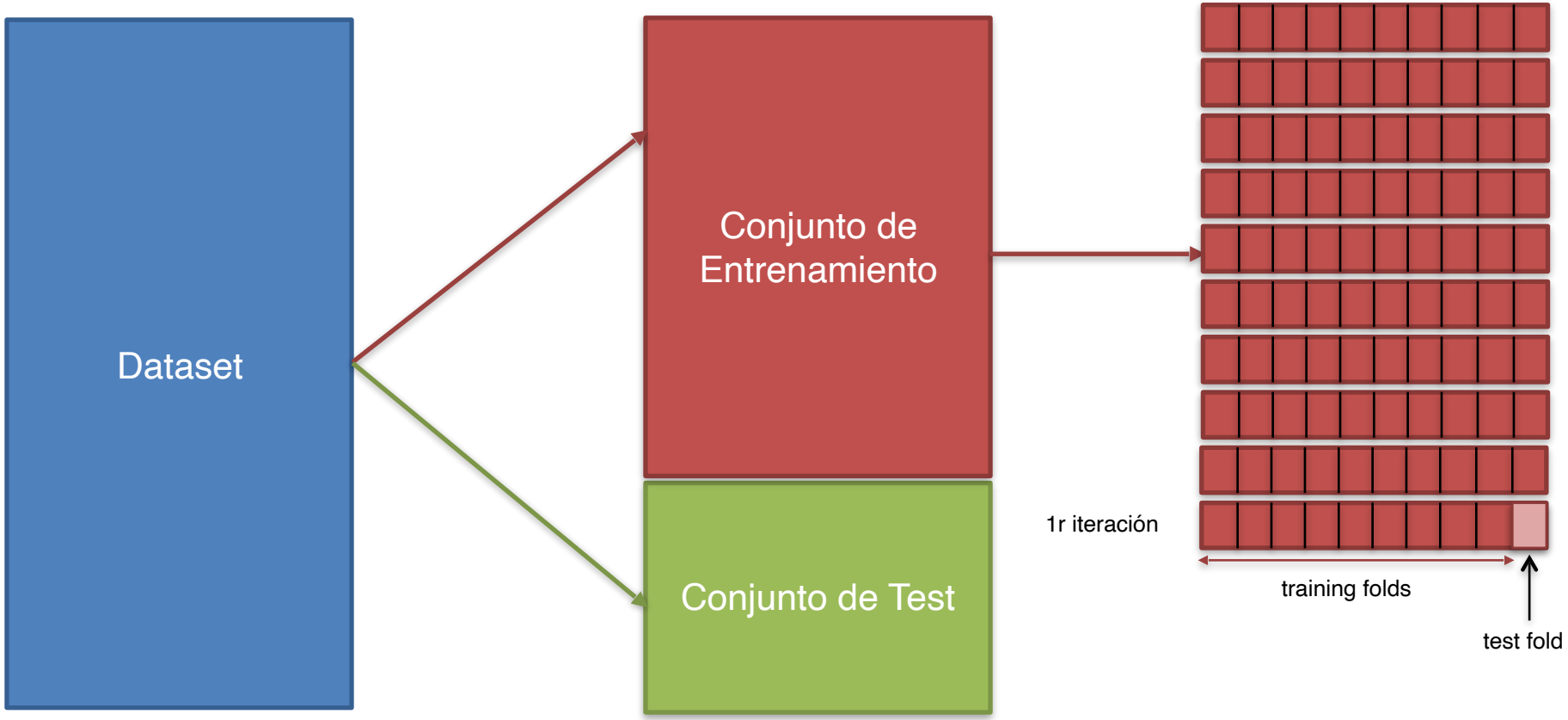
k-Fold Cross Validation



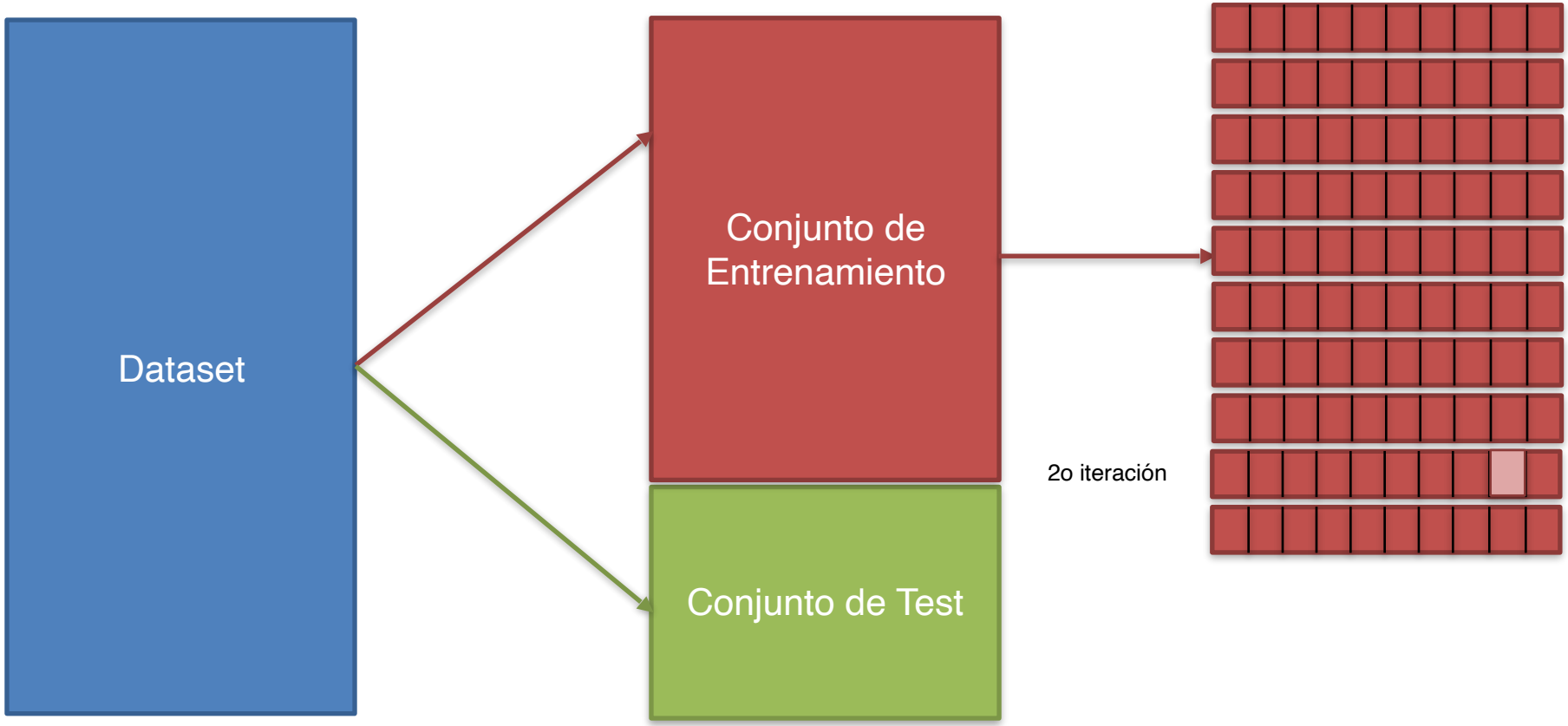
k-Fold Cross Validation



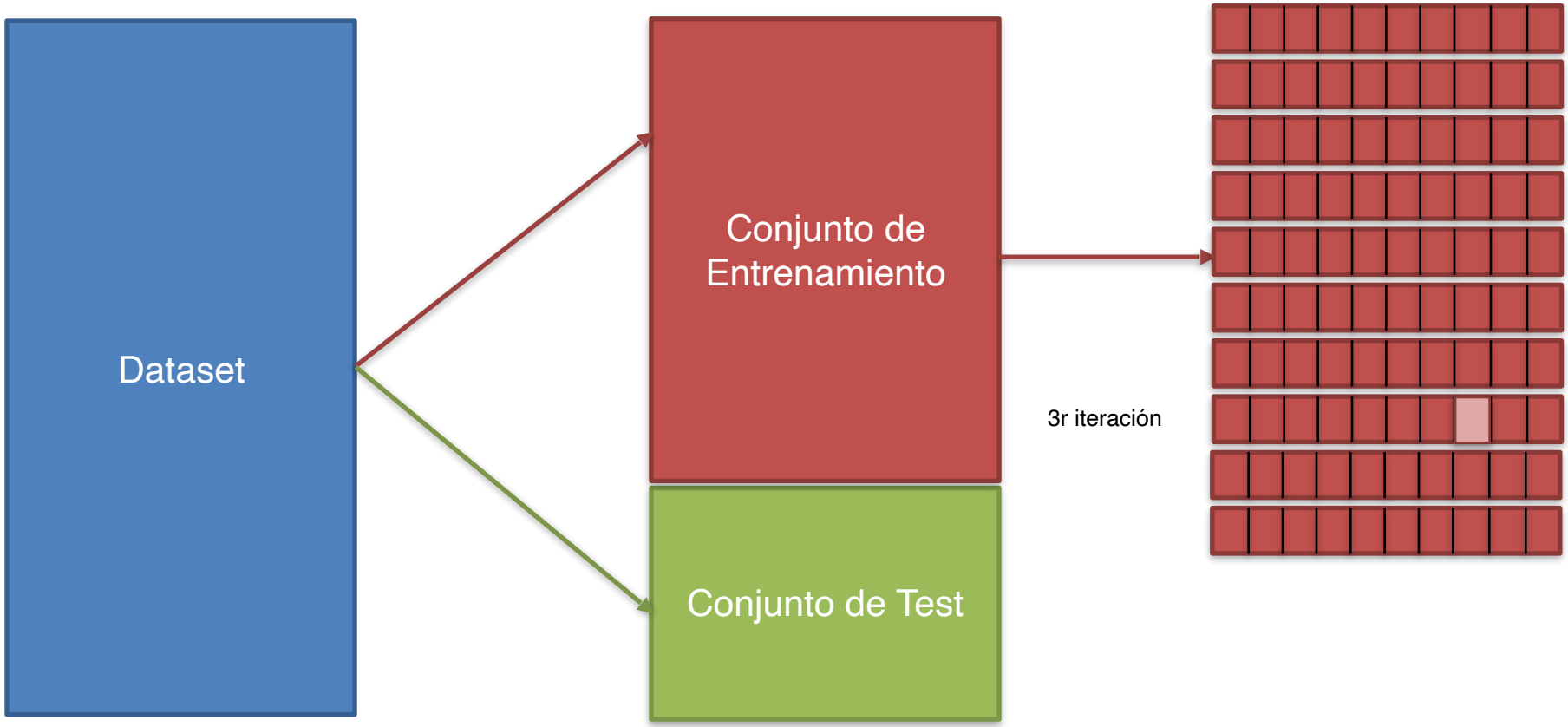
k-Fold Cross Validation



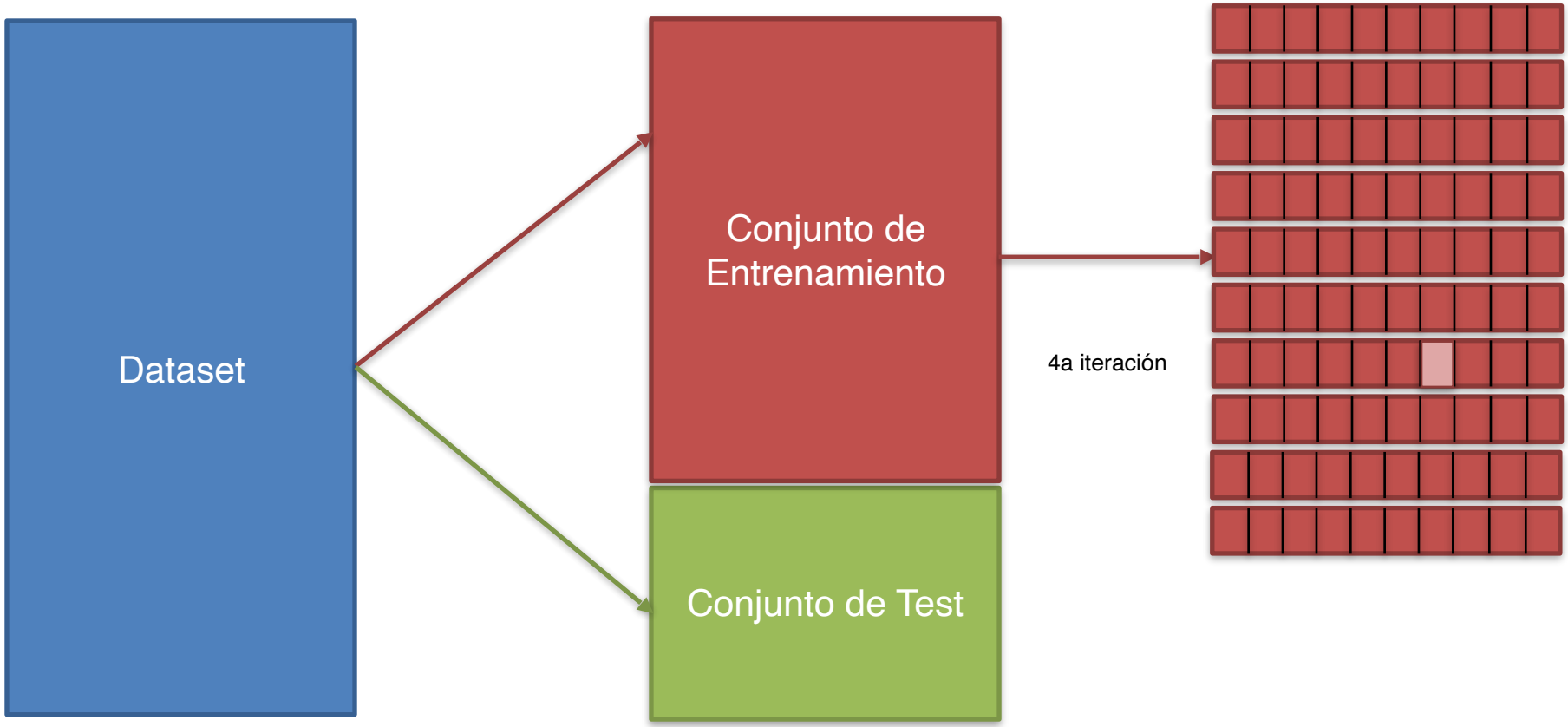
k-Fold Cross Validation



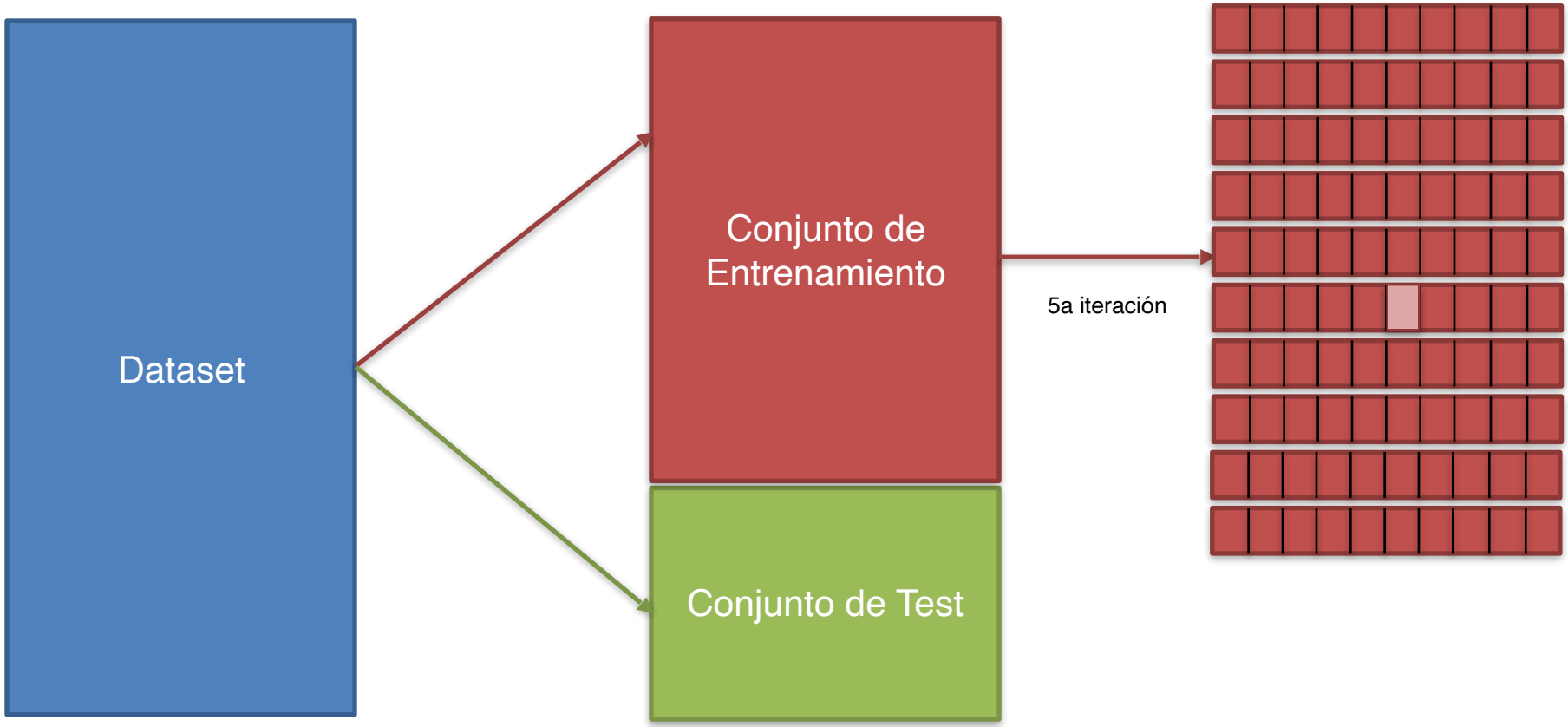
k-Fold Cross Validation



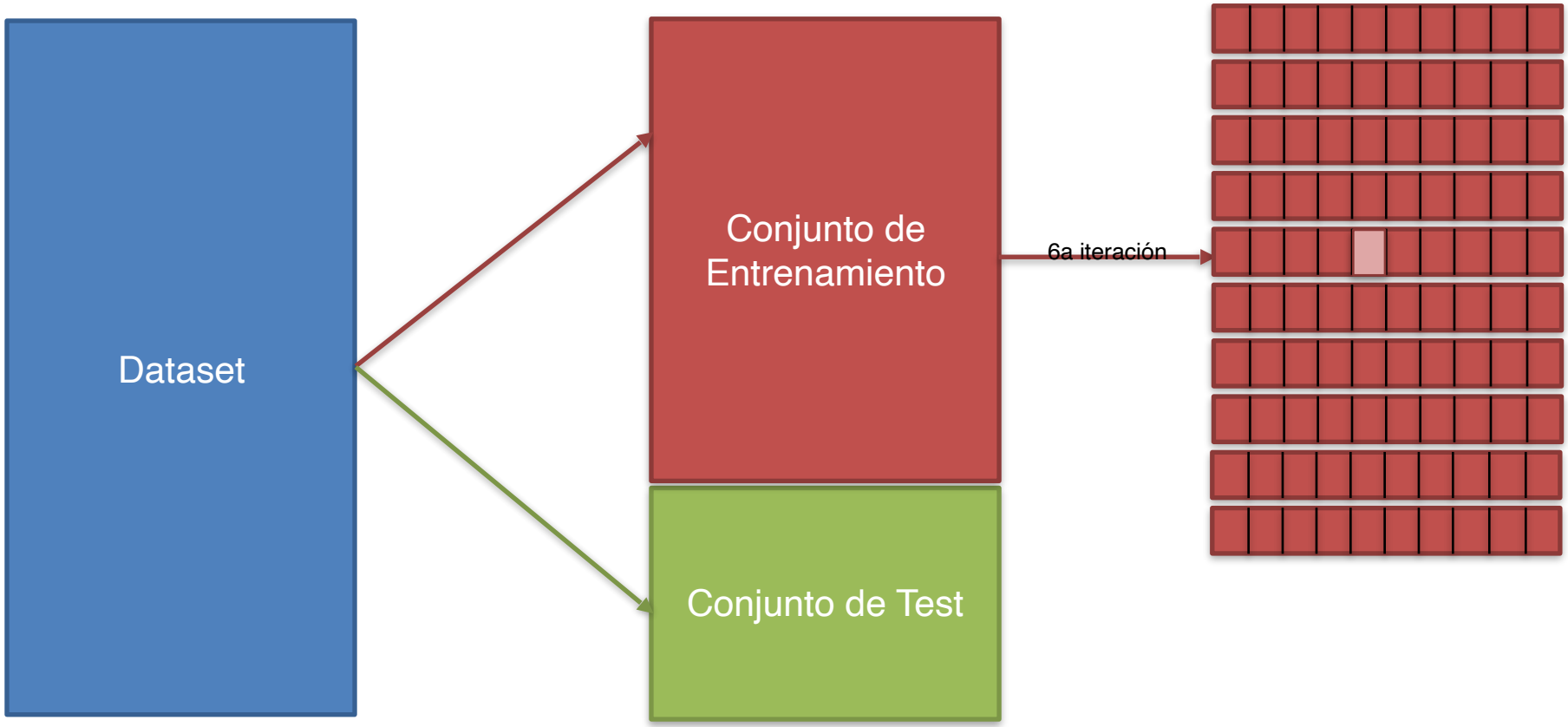
k-Fold Cross Validation



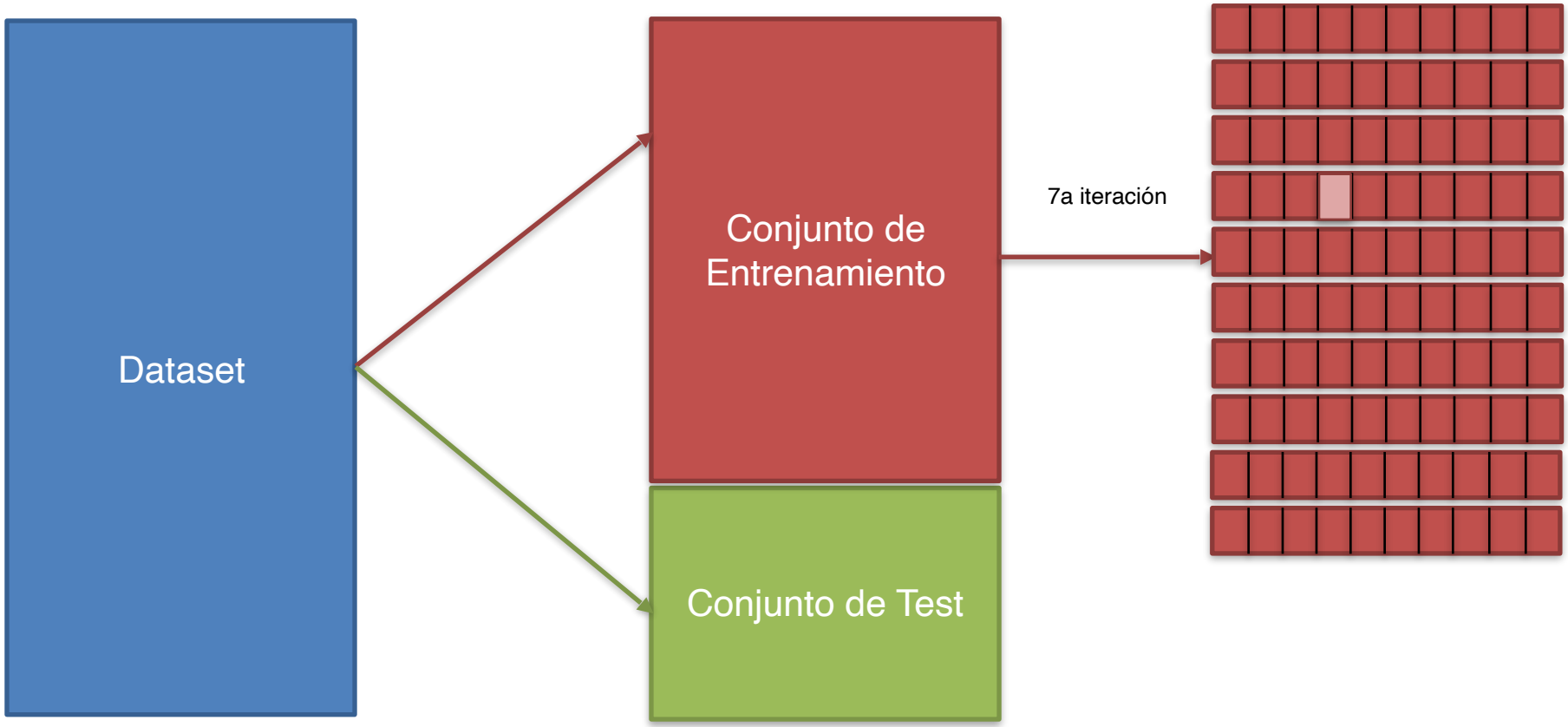
k-Fold Cross Validation



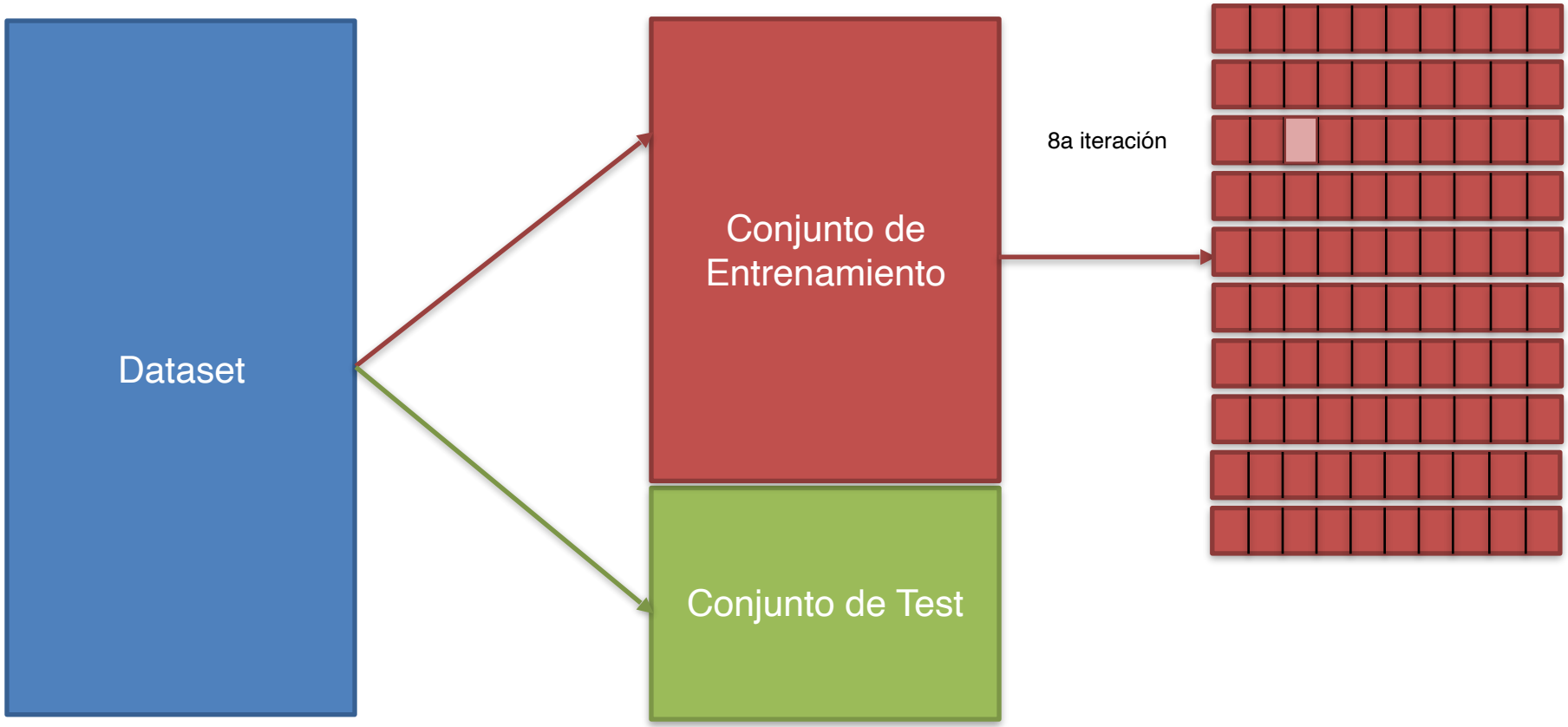
k-Fold Cross Validation



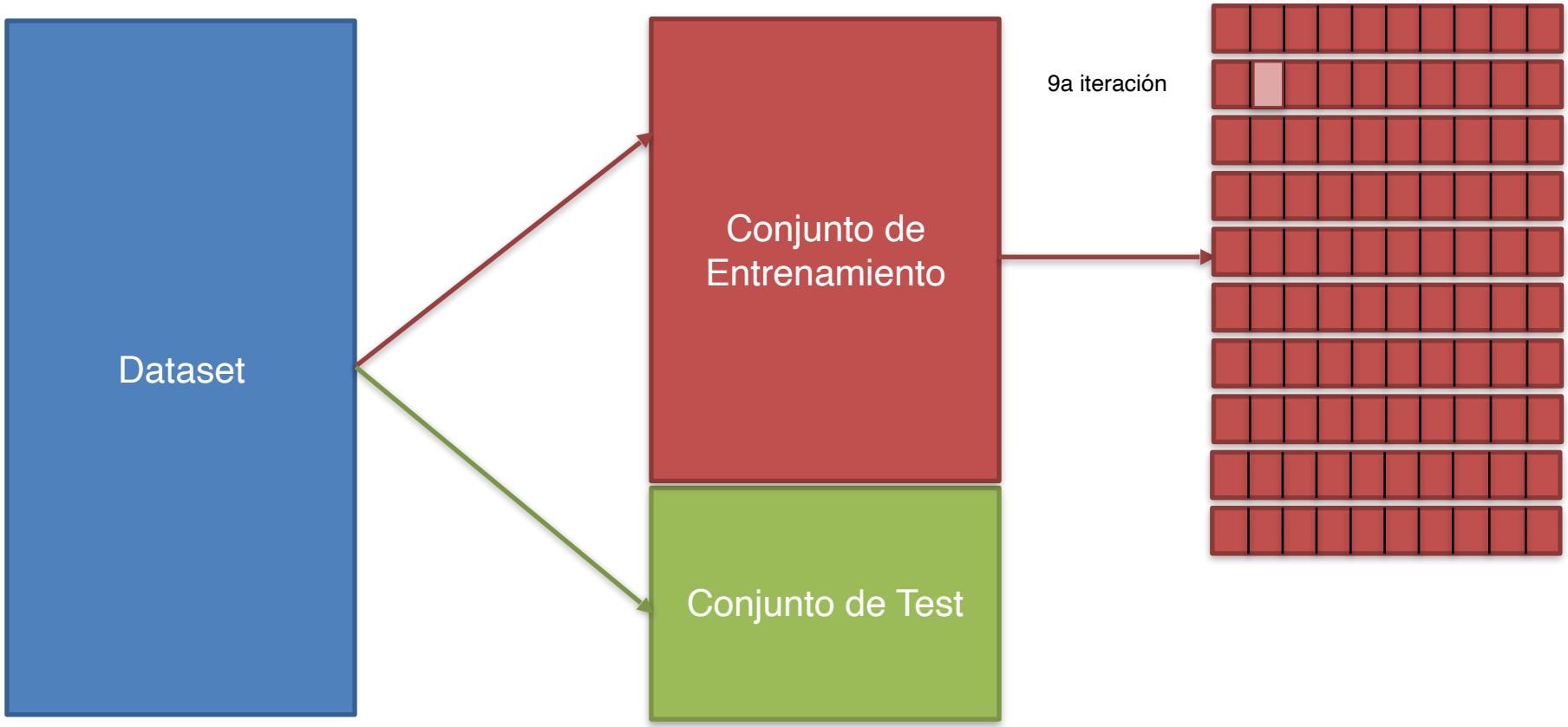
k-Fold Cross Validation



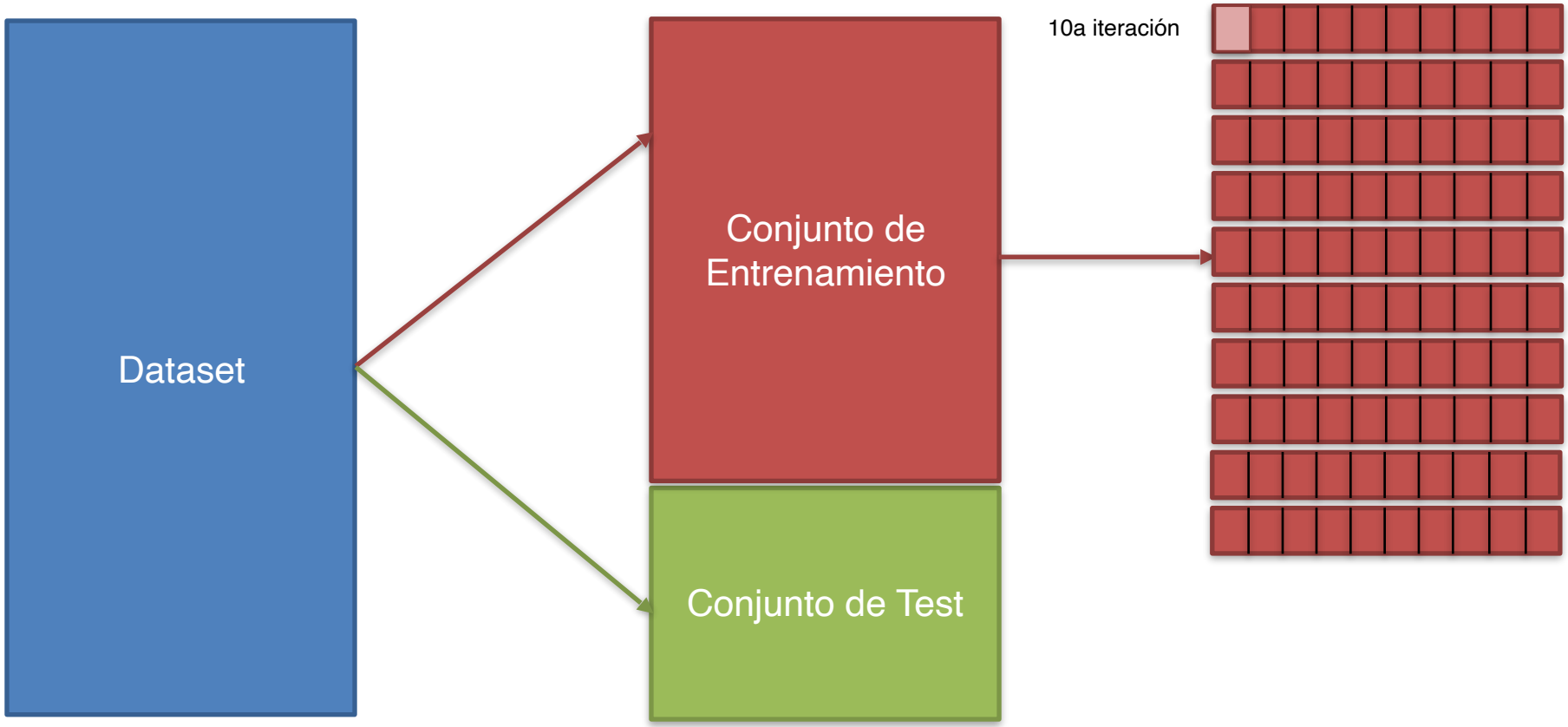
k-Fold Cross Validation



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