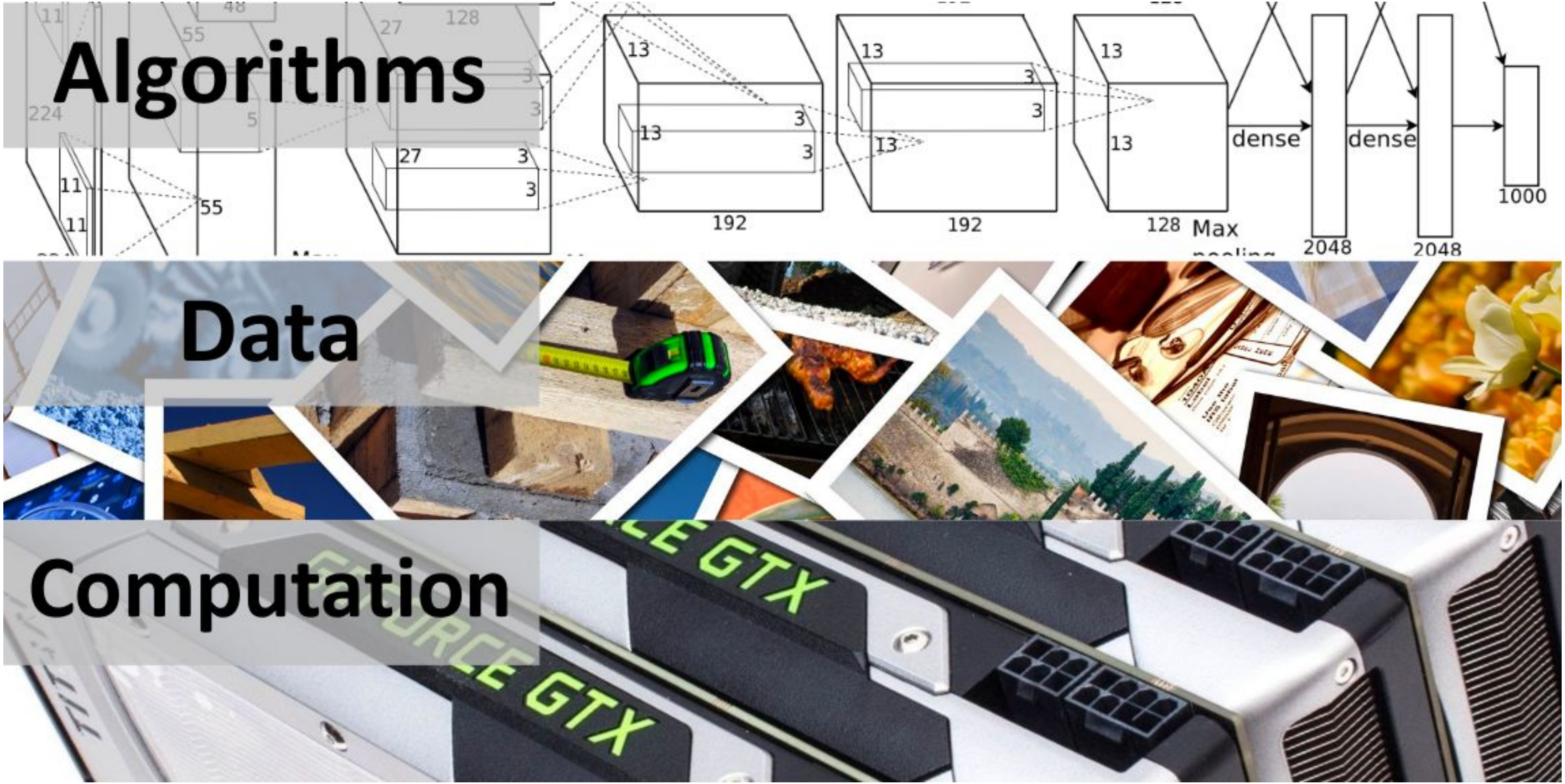
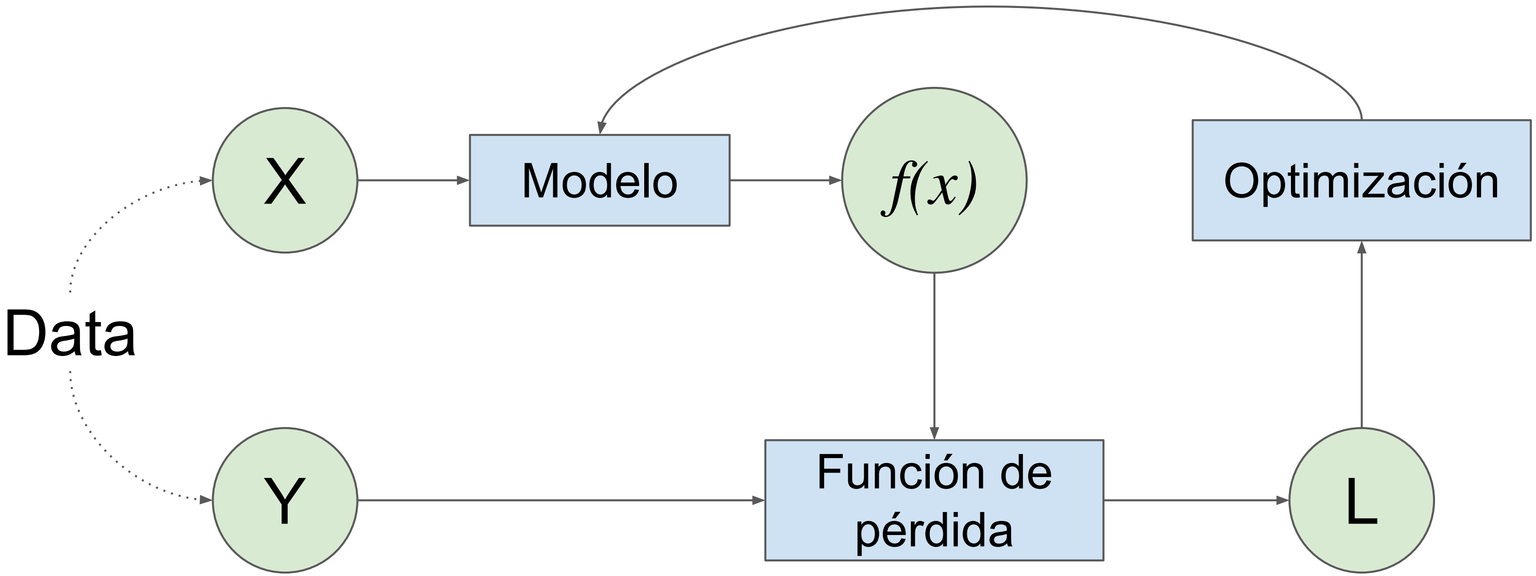
Técnicas Avanzadas de Data Mining y Sistemas Inteligentes

Maestría en Informática
Escuela de Posgrado
Pontificia Universidad Católica del Perú

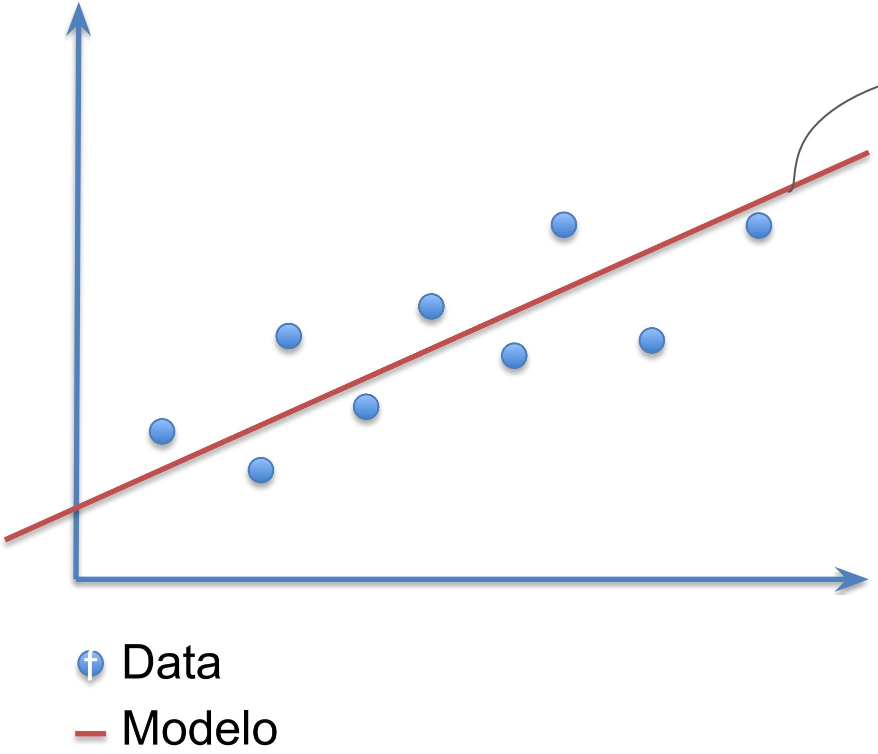
2018-2

Renato Hermoza - renato.hermoza@pucp.edu.pe


Algorithms
Data
Computation
13
192
128
Max
dense
dense
2048
2048
1000
GTX
GTX


X
Modelo
f(x)
Optimización
Data
Y
Función de
pérdida
L

Regresión lineal simple



$$f(x) = ax + b$$

Donde:

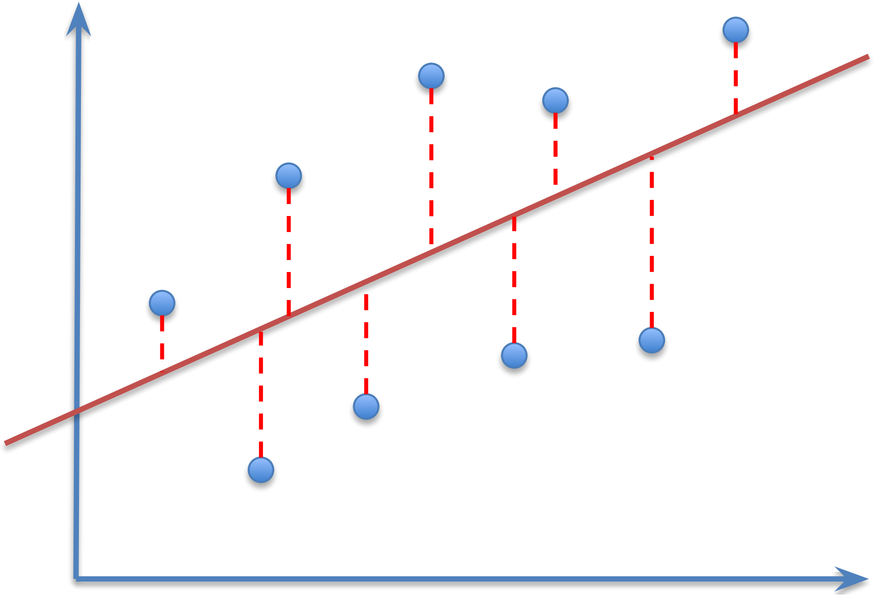
a = pendiente

b = término de intercepción (bias)

$$\begin{array}{ccccc} \boldsymbol{x} & & \boldsymbol{w} & & \\ \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 2 & 1 \\ 1 & 1 \end{bmatrix} & \times & \begin{bmatrix} 2 \\ 1 \end{bmatrix} & = & \begin{bmatrix} 3 \\ 1 \\ 5 \\ 3 \end{bmatrix} \\ \color{blue}{4}\text{x}\color{red}{2} & & \color{red}{2}\text{x}\color{green}{1} & & \color{blue}{4}\text{x}\color{green}{1} \end{array}$$

http://matrixmultiplication.xyz

Función de pérdida: MSE (Mean squared error)



Mean squared error:

$$L = \frac{1}{2m}\sum_{i=1}^{m}(f(x_i) - y_i)^2$$

Optimización: Gradient descent

$$f(x_i) = ax_i + b$$

$$L = \frac{1}{2m}\sum_{i=1}^{m}(f(x_i) - y_i)^2$$

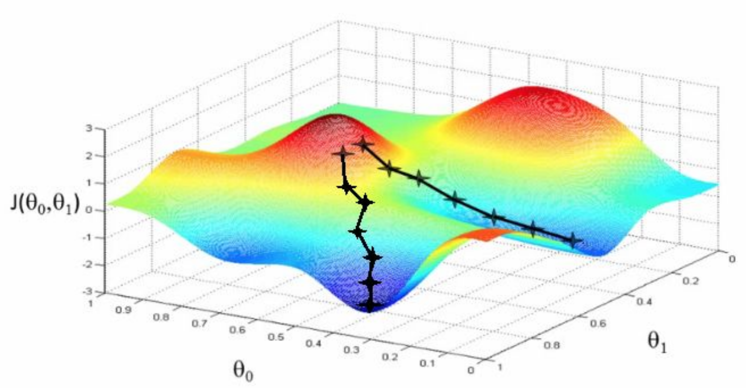
$$\frac{dL}{da} = \sum_{i=1}^{m}(f(x_i) - y_i)(x_i)$$

$$\frac{dL}{db} = \sum_{i=1}^{m}(f(x_i) - y_i)$$

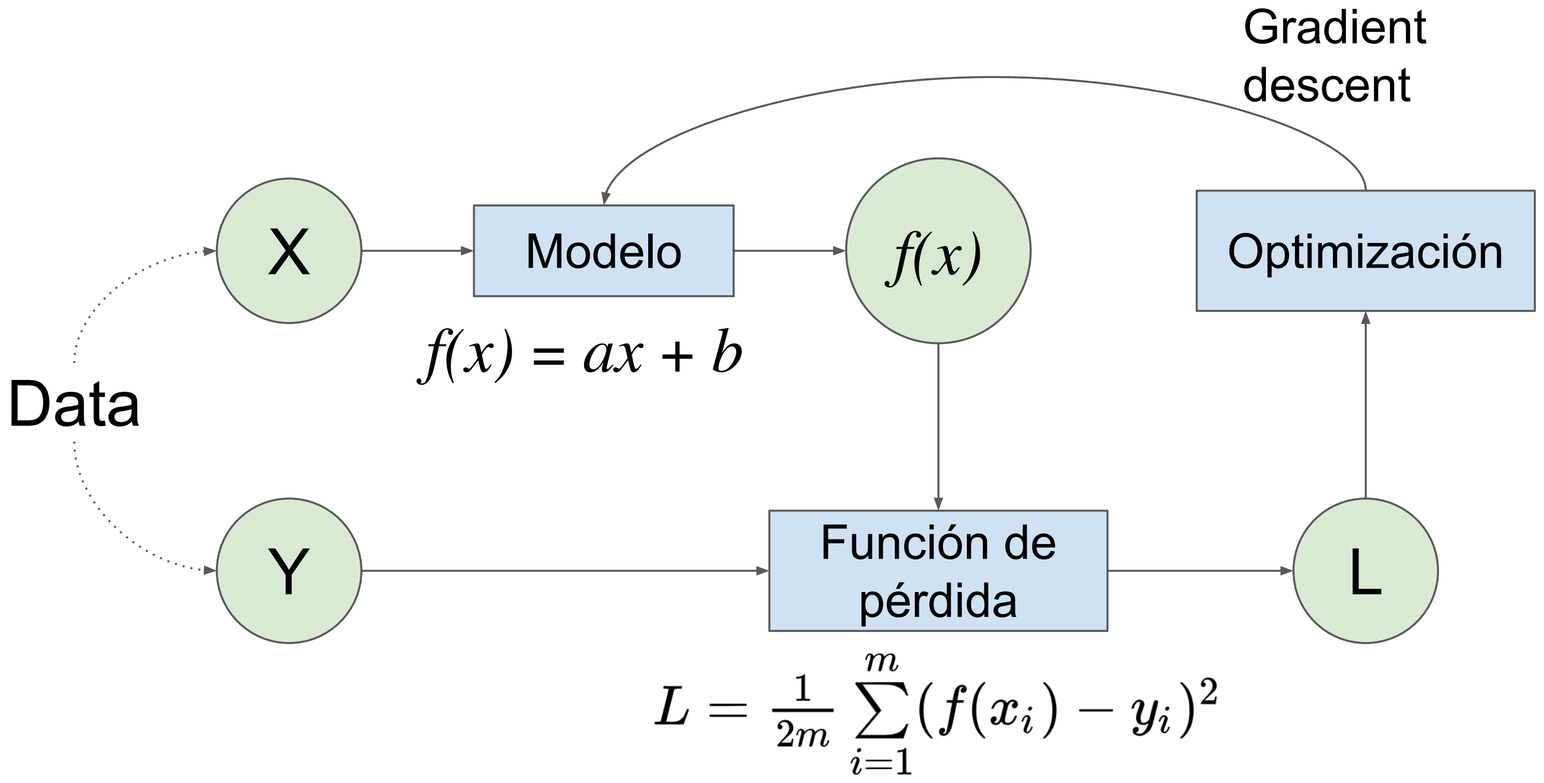
$$iterar:$$

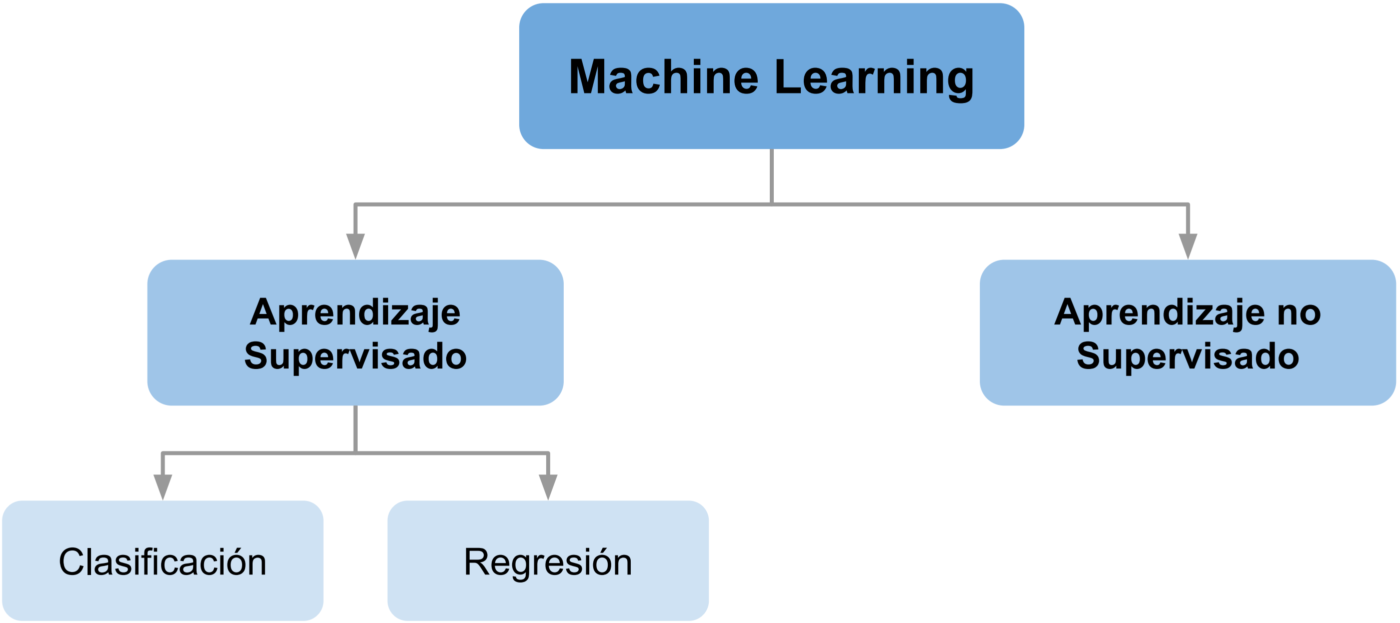
$$new\,a = a - lr * (\tfrac{dL}{da})$$

$$new\,b = b - lr * (\tfrac{dL}{db})$$

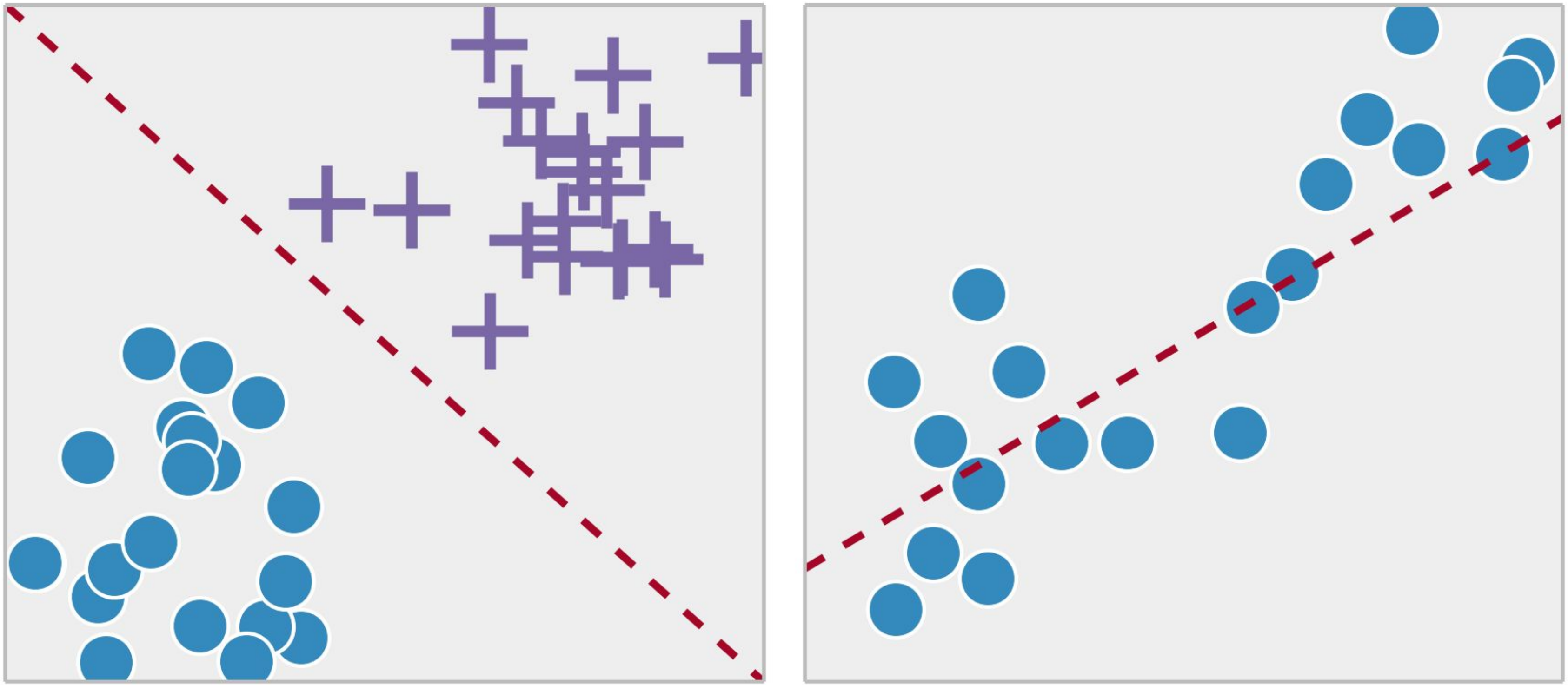


Regresión lineal simple



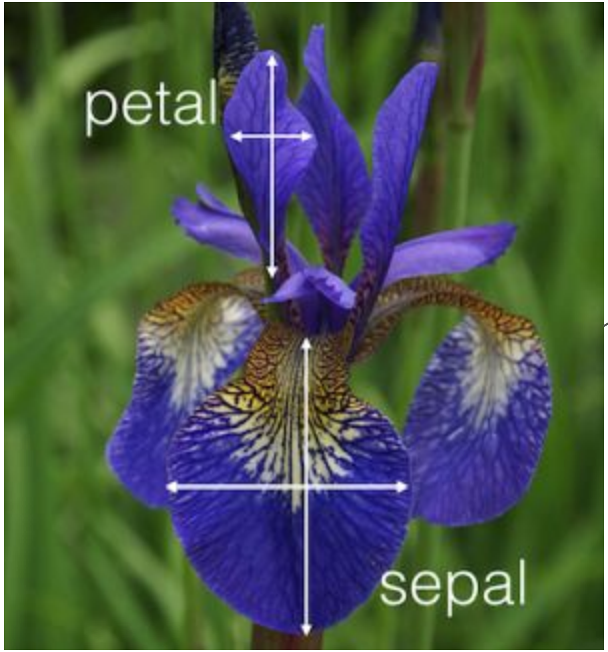

Machine Learning
Aprendizaje Supervisado
Aprendizaje no Supervisado
Clasificación
Regresión

Clasificación vs Regresión



Regresión Logística

Iris dataset



sepal_length	sepal_width	petal_length	petal_width
4.6	3.6	1.0	0.2
5.7	4.4	1.5	0.4
6.7	3.1	4.4	1.4
4.8	3.4	1.6	0.2
4.4	3.2	1.3	0.2

Target: class of iris plant

Iris dataset

Iris setosa

Iris versicolor

Iris virginica

Objetivo

sepal_length	sepal_width	petal_length	petal_width
4.6	3.6	1.0	0.2

$\xrightarrow{???}$

setosa!

~~versicolor~~

~~virginica~~

sepal_length	sepal_width	petal_length	petal_width
4.6	3.6	1.0	0.2

Variables independientes:
[4x1]

???

setosa!

3 resultados, indicando la probabilidad de cada clase:

[1, 0, 0]

Setosa Versicolor Virginica

(one hot encoding)

[3x1]

Definición del modelo

sepal_length	sepal_width	petal_length	petal_width
4.6	3.6	1.0	0.2

Variables independientes:
[4x1]

data weights

f(**x**,**W**) →

setosa!

3 resultados, indicando la probabilidad de cada clase:

[1, 0, 0]

Setosa Versicolor Virginica

(one hot encoding)

[3x1]

Definición del modelo: **Clasificador Lineal**

sepal_length	sepal_width	petal_length	petal_width
4.6	3.6	1.0	0.2

Variables independientes: **[4x1]**

data weights

$$\xrightarrow{f(\mathbf{x},\mathbf{W})}$$

$$f(x,W) = Wx$$

setosa!

3 resultados, indicando la probabilidad de cada clase:

[1, 0, 0]

Setosa Versicolor Virginica

(one hot encoding)

[3x1]

Definición del modelo: **Clasificador Lineal**

sepal_length	sepal_width	petal_length	petal_width
4.6	3.6	1.0	0.2

→ **setosa!**

$$\boxed{f(x, W)} = \boxed{W}\boxed{x}$$

4xN

3xN

?

parameters, or “weights”

Definición del modelo: **Clasificador Lineal**

sepal_length	sepal_width	petal_length	petal_width
4.6	3.6	1.0	0.2

setosa!

4xN

$$\boxed{f(x, W)} = \boxed{W}\boxed{x}$$

3xN **[3x4]**

parameters, or “weights”

Definición del modelo: **Clasificador Lineal**

sepal_length	sepal_width	petal_length	petal_width
4.6	3.6	1.0	0.2

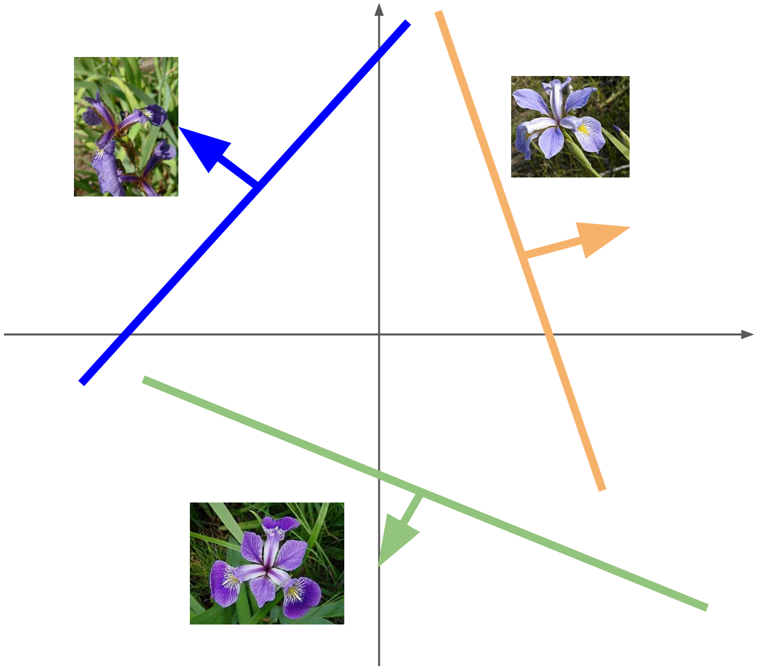
$\longrightarrow$ **setosa!**

$$\underbrace{\boxed{f(x, W)}}_{3xN} = \underbrace{\boxed{W}}_{[3x4]}\overbrace{\boxed{x}}^{4xN} \quad \underbrace{\boxed{(+b)}}_{\text{bias}}\ 3x1$$

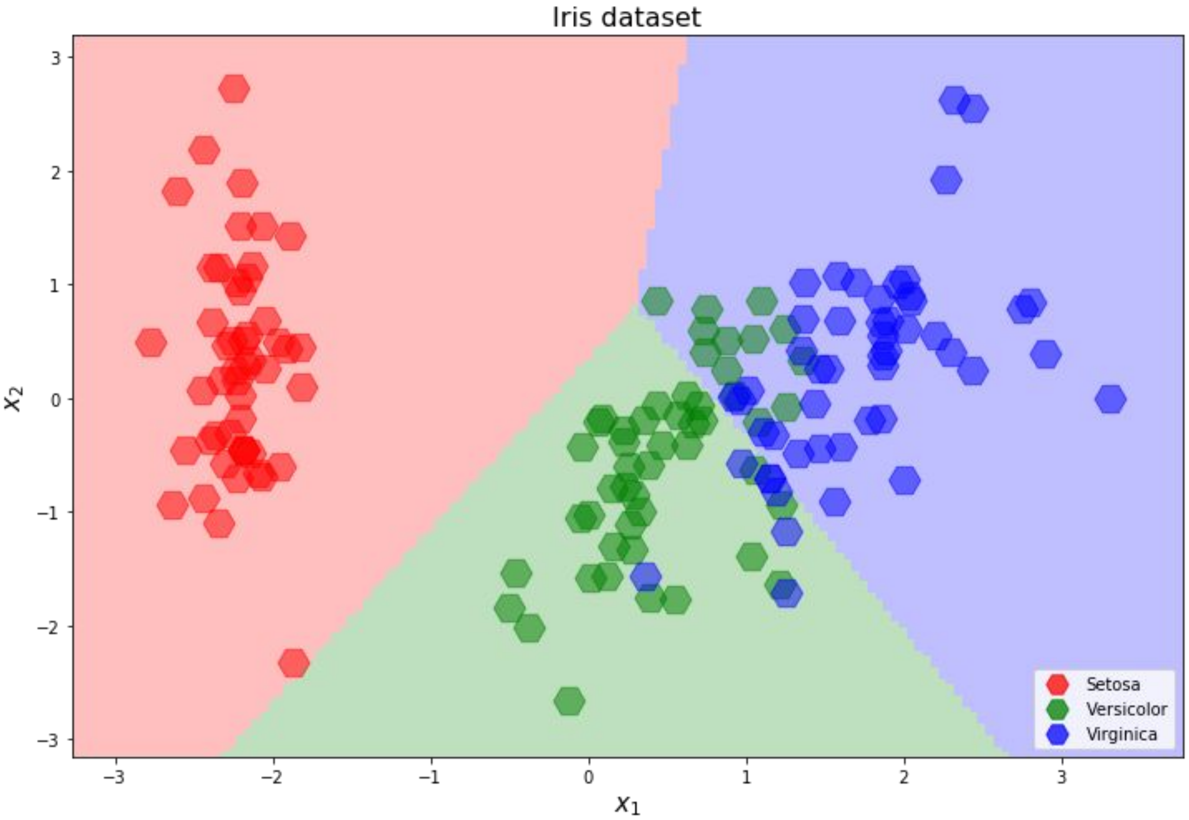
parameters, or "weights"

Interpretando el clasificador lineal

$$f(x_i) = Wx_i + b$$



Interpretando el clasificador lineal



Ej:

sepal_length	sepal_width	petal_length	petal_width
4.6	3.6	1.0	0.2

setosa!

[1, 0, 0]

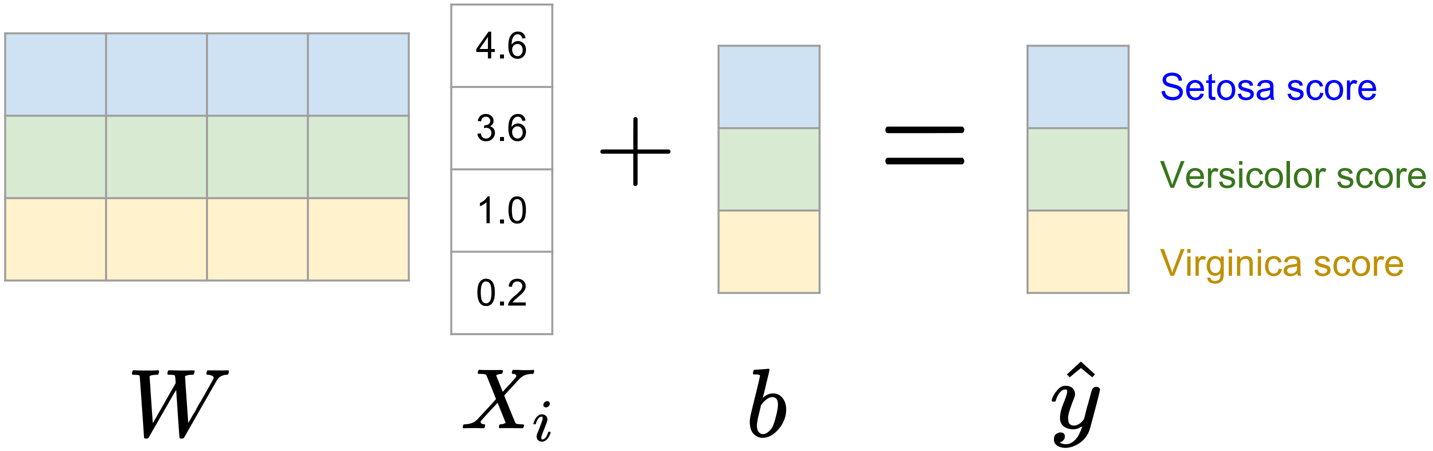
Setosa Versicolor Virginica

$$X_i = \begin{bmatrix} 4.6 \\ 3.6 \\ 1.0 \\ 0.2 \end{bmatrix} \qquad y_i = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Ej:

sepal_length	sepal_width	petal_length	petal_width
4.6	3.6	1.0	0.2

$$\hat{y} = f(x_i) = Wx_i + b$$



Ej:

sepal_length	sepal_width	petal_length	petal_width
4.6	3.6	1.0	0.2

Random numbers!

$$\hat{y} = f(x_i) = Wx_i + b$$

$$\underbrace{\begin{bmatrix} -0.4 & -0.1 & -2.1 & 1.6 \\ -1.8 & -0.8 & 0.5 & -1.2 \\ -1.1 & -0.9 & 0.6 & 2.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 4.6 \\ 3.6 \\ 1.0 \\ 0.2 \end{bmatrix}}_{X_i} + \underbrace{\begin{bmatrix} 1.8 \\ 0.4 \\ 0.1 \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} -2.18 \\ -10.5 \\ -7.14 \end{bmatrix}}_{\hat{y}} \begin{array}{l} \text{Setosa score} \\ \text{Versicolor score} \\ \text{Virginica score} \end{array}$$

¿Cómo interpretamos estos scores?

-2.18	Setosa score
-10.5	Versicolor score
-7.14	Virginica score

Softmax Classifier (Multinomial Logistic Regression)

-2.18	Setosa score
-10.5	Versicolor score
-7.14	Virginica score

$$softmax(\hat{y}_i) = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

Softmax Classifier (Multinomial Logistic Regression)

$$softmax(\hat{y_i}) = \frac{e^{\hat{y_i}}}{\sum_j e^{\hat{y_j}}}$$

Setosa score	-2.18
Versicolor score	-10.5
Virginica score	-7.14

$\hat{y_i}$

Softmax Classifier (Multinomial Logistic Regression)

$$softmax(\hat{y}_i) = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$$

	$\hat{y}_i$		$e^{\hat{y}_i}$
Setosa score	-2.18	$\xrightarrow{\text{exp}}$	0.11304
Versicolor score	-10.5		0.00003
Virginica score	-7.14		0.00079

Softmax Classifier (Multinomial Logistic Regression)

$$softmax(\hat{y}_i) = \frac{e^{\hat{y_i}}}{\sum_j e^{\hat{y_j}}}$$

	$\hat{y}_i$	exp	$e^{\hat{y_i}}$	normalize	$\frac{e^{\hat{y_i}}}{\sum_j e^{\hat{y_j}}}$
Setosa score	-2.18		0.11304		0.99280
Versicolor score	-10.5		0.00003		0.00024
Virginica score	-7.14		0.00079		0.00696

Softmax!

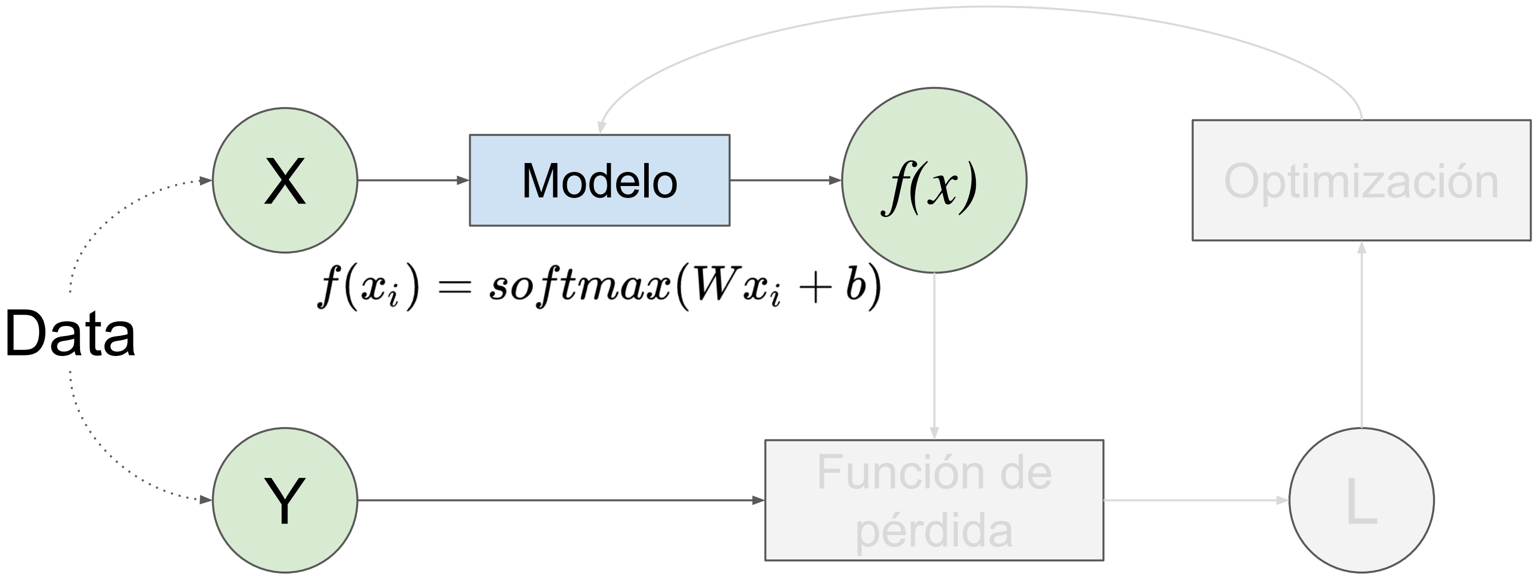
Definición final del modelo

$$f(x_i) = softmax(Wx_i + b)$$

	$\hat{y}_i$	exp $\rightarrow$ $e^{\hat{y}_i}$	normalize $\rightarrow$ $\frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}}$
Setosa score	-2.18	0.11304	0.99280
Versicolor score	-10.5	0.00003	0.00024
Virginica score	-7.14	0.00079	0.00696

Softmax!

Regresión logística



$$f(x_i) = softmax(Wx_i + b)$$

¿Cómo evaluamos el modelo?

Cross Entropy (Función de pérdida)

$$L = -\sum p(x) \log q(x)$$

$$L = -\sum_{i=1}^{m} y_i log(\hat{y}_i)$$

Cross Entropy (Función de pérdida)

$$L = -\sum_{i=1}^{m} \boxed{y_i} log\boxed{(\hat{y}_i)}$$

	y_i	$\hat{y}_i$
Setosa	1	0.99280
Versicolor	0	0.00024
Virginica	0	0.00696

Cross Entropy (Función de pérdida)

$$L = -\sum_{i=1}^{m} y_i \, log(\hat{y}_i)$$

	y_i	$log(\hat{y}_i)$
Setosa	1	-0.00723
Versicolor	0	-8.32723
Virginica	0	-4.96723

Cross Entropy (Función de pérdida)

$$L = -\sum_{i=1}^{m} \boxed{y_i} \boxed{log(\hat{y}_i)}$$

$$\begin{array}{l} \text{Setosa} \\ \text{Versicolor} \\ \text{Virginica} \end{array} \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}_{y_i} * \underbrace{\begin{bmatrix} -0.00723 \\ -8.32723 \\ -4.96723 \end{bmatrix}}_{log(\hat{y}_i)} = \begin{bmatrix} -0.00723 \\ 0 \\ 0 \end{bmatrix}$$

Cross Entropy (Función de pérdida)

$$L = -\sum_{i=1}^{m} y_i log(\hat{y}_i)$$

$$L = -\sum \begin{bmatrix} -0.00723 \\ 0 \\ 0 \end{bmatrix} = \mathbf{0.00723}$$

Review:
Entropy, Cross-entropy

Ej:
Queremos usar una antena para enviar el estado del clima.

Ej:

Queremos usar una antena para enviar el estado del clima.

					
	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
code	000	001	010	011	100
bits	3	3	3	3	3

cantidad promedio de bits usados = 3

Ej:
Queremos usar una antena para enviar el estado del clima.

					
	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x)	0.25	0.45	0.2	0.09	0.01
code	000	001	010	011	100
bits	3	3	3	3	3

Veamos las probabilidades del clima en una ciudad A

bits promedio = 3(0.25) + 3(0.45) + 3(0.2) + 3(0.09) + 3(0.01) = 3

Ej:
Queremos usar una antena para enviar el estado del clima.

					
	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x)	0.25	0.45	0.2	0.09	0.01
code	1100	0	10	1101	1110
bits	4	1	2	4	4

Usamos códigos más óptimos para las probabilidades de la ciudad A

bits promedio = 4(0.25) + 1(0.45) + 2(0.2) + 4(0.09) + 4(0.01) = 2.25

Ej:
Queremos usar una antena para enviar el estado del clima.

					
	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x)	0.25	0.45	0.2	0.09	0.01

¿Cómo obtenemos mínimo número de bits que se puede usar?

Ej:
Queremos usar una antena para enviar el estado del clima.

					
	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x)	0.25	0.45	0.2	0.09	0.01
1/p(x)	4	2.22	5	11.11	100

¿Cómo obtenemos mínimo número de bits que se puede usar?

Ej:

Queremos usar una antena para enviar el estado del clima.

					
	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x)	0.25	0.45	0.2	0.09	0.01
1/p(x)	4	2.22	5	11.11	100
$-\log_2(p)$	2	1.15	2.32	3.47	6.64

¿Cómo obtenemos mínimo número de bits que se puede usar?

$$\log_2\left(\frac{1}{p}\right) = -\log_2(p)$$

Ej:

Queremos usar una antena para enviar el estado del clima.

					
	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x)	0.25	0.45	0.2	0.09	0.01
$-\log_2(p)$	2	1.15	2.32	3.47	6.64

¿Cómo obtenemos mínimo número de bits que se puede usar?

$$\log_2\left(\frac{1}{p}\right) = -\log_2(p)$$

bits promedio = 0.25(2) +0.45(1.15) +0.2(2.32) +0.09(3.47) +0.01(6.64) = 1.86

Ej:

Queremos usar una antena para enviar el estado del clima.

	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x)	0.25	0.45	0.2	0.09	0.01
$-\log_2(p)$	2	1.15	2.32	3.47	6.64

¿Cómo obtenemos mínimo número de bits que se puede usar?

Entropy

$$-\sum p(x) \log p(x)$$

bits promedio = 0.25(2) +0.45(1.15) +0.2(2.32) +0.09(3.47) +0.01(6.64) = 1.86

Ej:

Queremos usar una antena para enviar el estado del clima.

					
	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x)	0.25	0.45	0.2	0.09	0.01
$-\log_2(p)$	2	1.15	2.32	3.47	6.64

¿Cómo obtenemos mínimo número de bits que se puede usar?

Entropy

$$-\sum p(x)\log p(x)$$

La entropía en este caso es 1.86, esta cantidad de bits representa la información que uno obtendría en promedio, al conocer el clima en la ciudad A.

¿Qué pasa si usamos la misma antena para una ciudad B?

¿Qué pasa si usamos la misma antena para una ciudad B?

					
	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x) en A	0.25	0.45	0.2	0.09	0.01
$-\log_2(p_A)$	2	1.15	2.32	3.47	6.64
p(x) en B	0.3	0.2	0.2	0.25	0.05

Si la antena usa los bits óptimos para la ciudad A.

¿Cuál sería la cantidad de bits promedio que enviaría la antena en la ciudad B?

¿Qué pasa si usamos la misma antena para una ciudad B?

					
	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x) en A	0.25	0.45	0.2	0.09	0.01
$-\log_2(p_A)$	2	1.15	2.32	3.47	6.64
p(x) en B	0.3	0.2	0.2	0.25	0.05

Si la antena usa los bits óptimos para la ciudad A.

¿Cuál sería la cantidad de bits promedio que enviaría la antena en la ciudad B?

bits promedio = 0.3(2) +0.2(1.15) +0.2(2.32) +0.25(3.47) +0.05(6.64) = 2.5

¿Qué pasa si usamos la misma antena para una ciudad B?

	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x) en A	0.25	0.45	0.2	0.09	0.01
$-\log_2(p_A)$	2	1.15	2.32	3.47	6.64
p(x) en B	0.3	0.2	0.2	0.25	0.05

Si la antena usa los bits óptimos para la ciudad A.

¿Cuál sería la cantidad de bits promedio que enviaría la antena en la ciudad B?

Cross-entropy

$$-\sum p(x) \log q(x)$$

bits promedio = 0.3(2) +0.2(1.15) +0.2(2.32) +0.25(3.47) +0.05(6.64) = 2.5

¿Qué pasa si usamos la misma antena para una ciudad B?

					
	Soleado	Nublado	Nublado parcial	Lluvia	Tormenta
p(x) en A	0.25	0.45	0.2	0.09	0.01
$-\log_2(p_A)$	2	1.15	2.32	3.47	6.64
p(x) en B	0.3	0.2	0.2	0.25	0.05

Si la antena usa los bits óptimos para la ciudad A.

¿Cuál sería la cantidad de bits promedio que enviaría la antena en la ciudad B?

Cross-entropy

$$-\sum p(x) \log q(x)$$

La entropía cruzada en este caso es 2.5, esta cantidad de bits representa la información que uno obtendría en promedio al conocer el clima en la ciudad B usando la codificación óptima para la ciudad A.

Entropy

$$-\sum p(x)\log p(x)$$

Cross-entropy

$$-\sum p(x)\log q(x)$$

Cross Entropy (Función de pérdida)

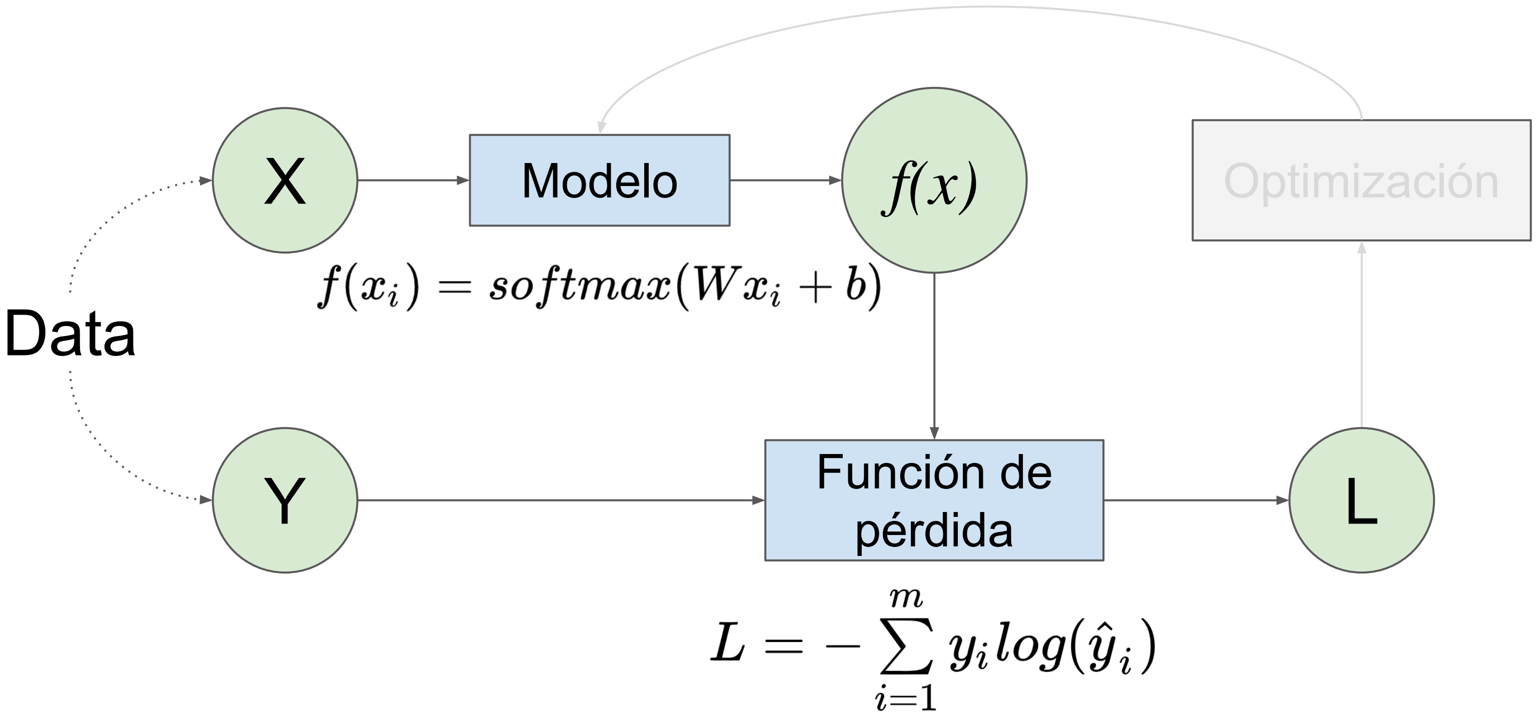
$$L = -\sum_{i=1}^{m} \boxed{y_i} log\boxed{(\hat{y}_i)} = -\sum \begin{bmatrix} -0.00723 \\ 0 \\ 0 \end{bmatrix}$$

	y_i	$\hat{y}_i$
Setosa	1	0.99280
Versicolor	0	0.00024
Virginica	0	0.00696

$$= \mathbf{0.00723}$$

Si usamos la predicción del modelo, **0.00723** representaría la información que obtendríamos al conocer la data real.

Regresión logística



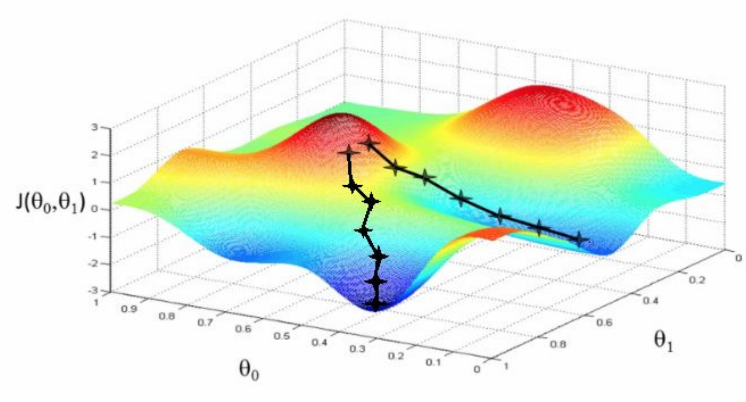
Optimización

Optimización

$$f(x_i) = softmax(Wx_i + b)$$

$$L = -\sum_{i=1}^{m} y_i log(\hat{y}_i)$$

$$\frac{dL}{dW}, \frac{dL}{db} ?$$



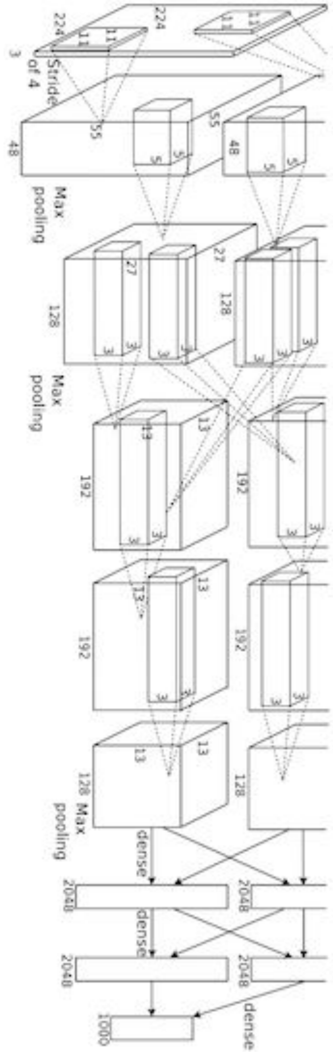
Convolutional Network
(AlexNet)

input image

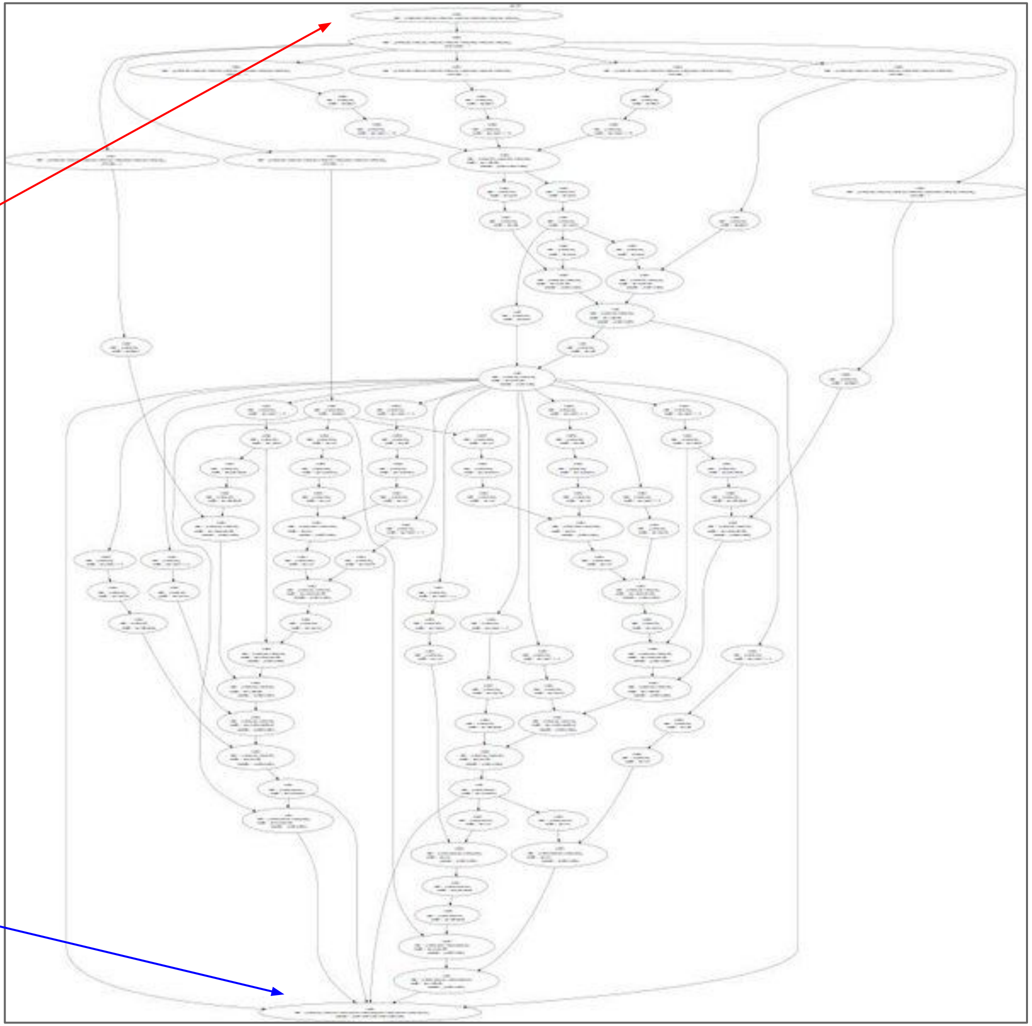
weights

loss

$$\frac{dL}{dW}?$$



Neural Turing Machine



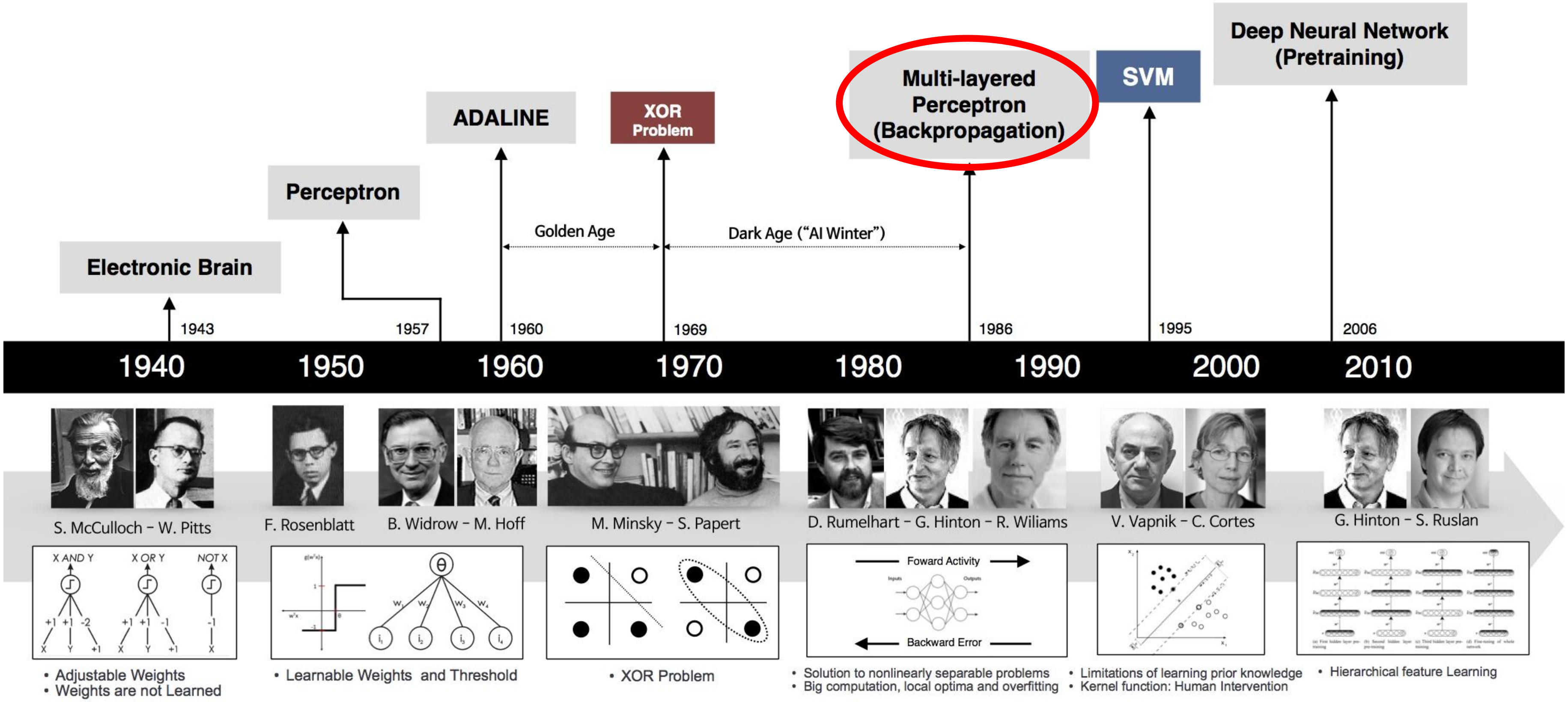
input tape

loss

$$\frac{dL}{dW}?$$

¿Cómo derivamos las gradientes?

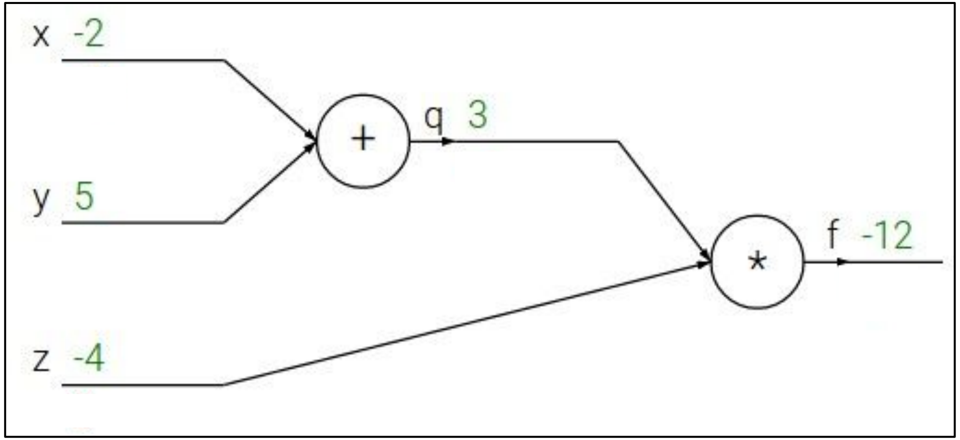
Backpropagation !



Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

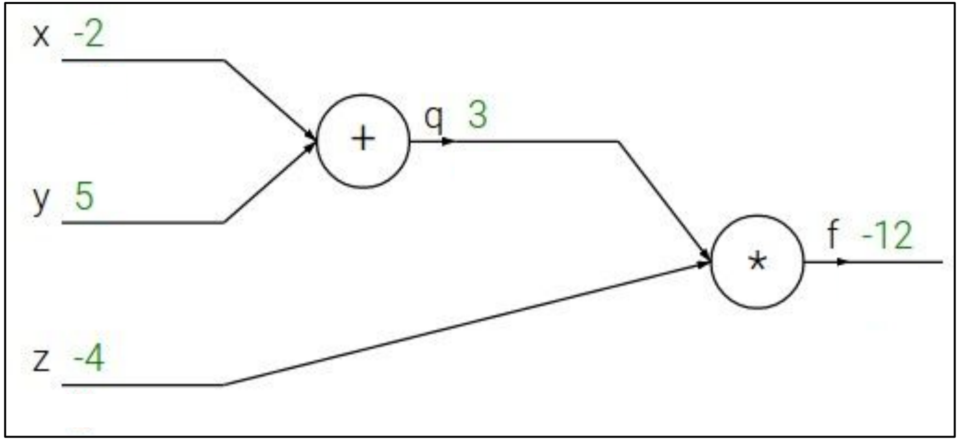


Queremos: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$



Queremos: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

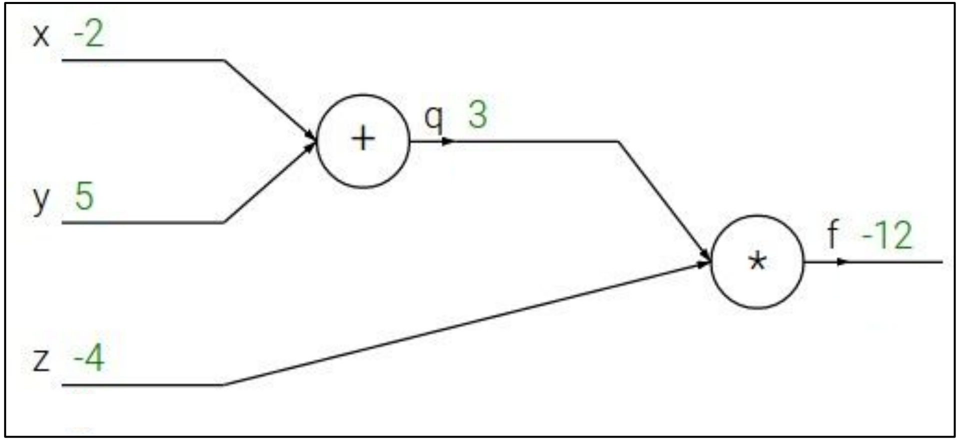
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Queremos: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



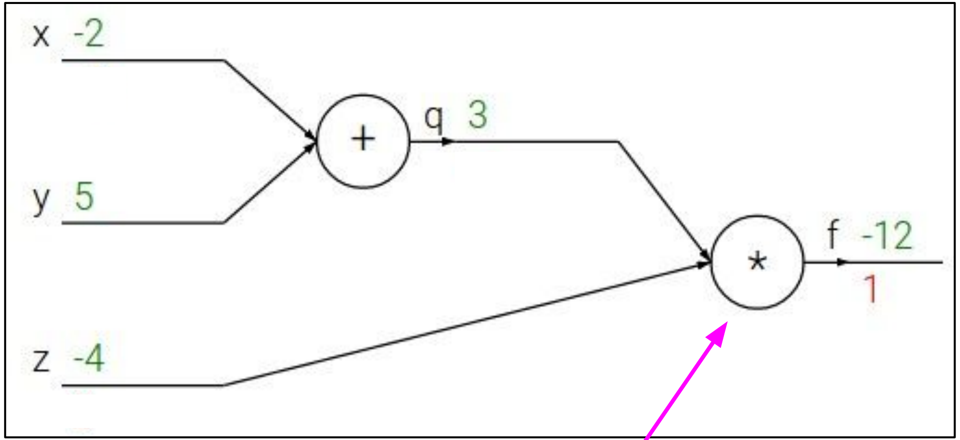
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Queremos: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{df}{df} = 1$$

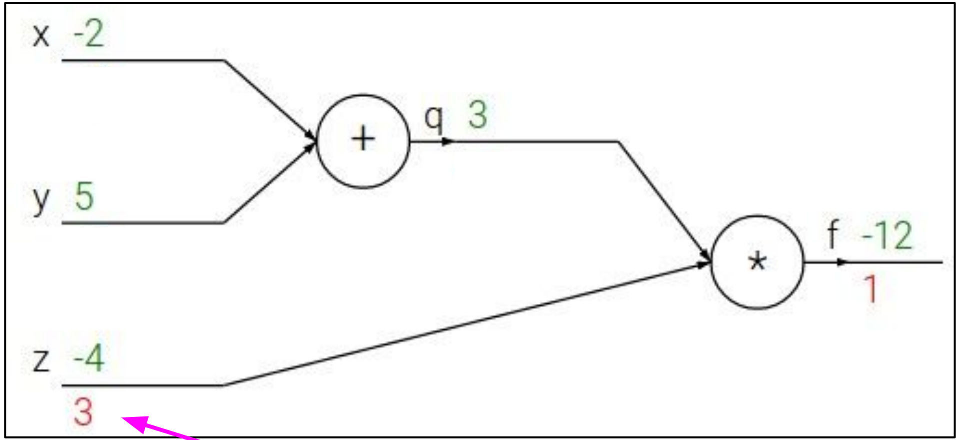
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Queremos: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{df}{dz} = q = 3$$

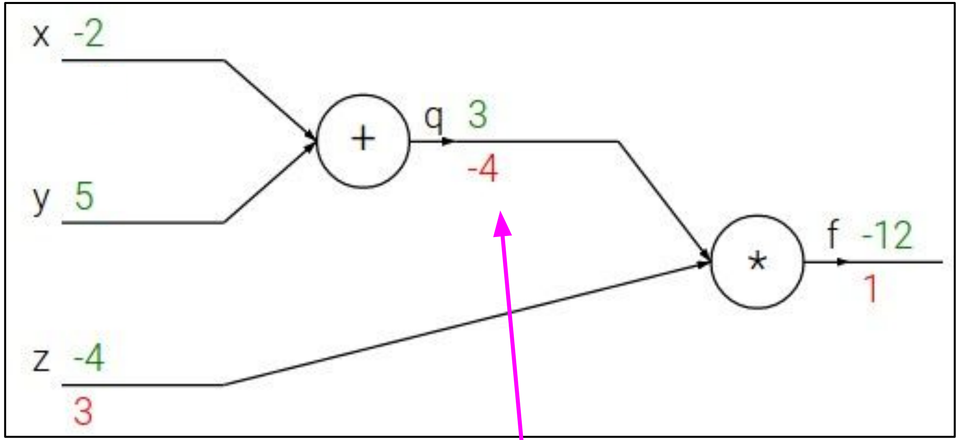
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Queremos: $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial f}{\partial z}$



$$\frac{df}{dq} = z = -4$$

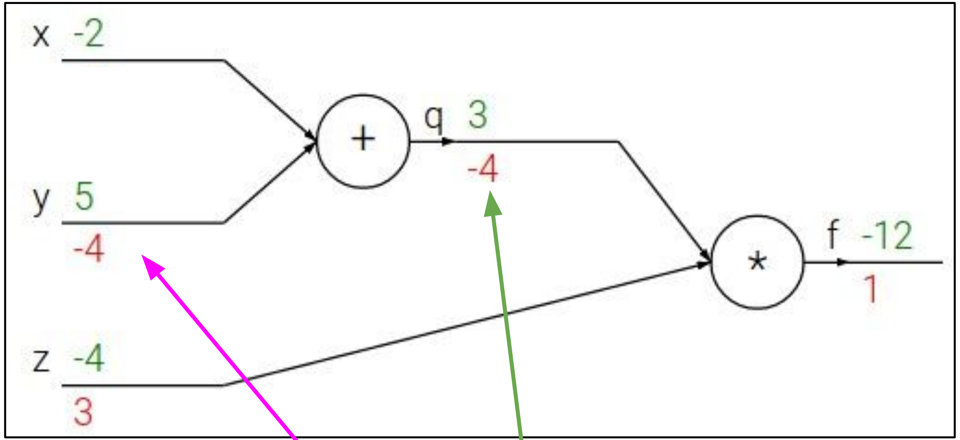
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Queremos: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

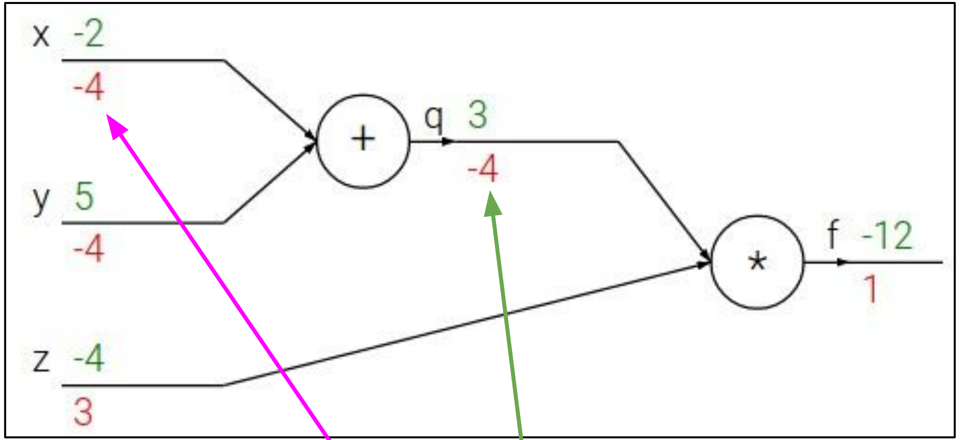


Chain rule:

$$\boxed{\frac{df}{dy} = \frac{df}{dq}\frac{dq}{dy} = -4 * 1 = -4}$$

$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4



$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

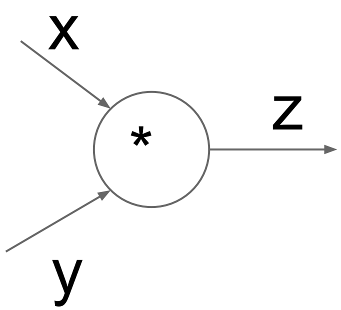
$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Queremos: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

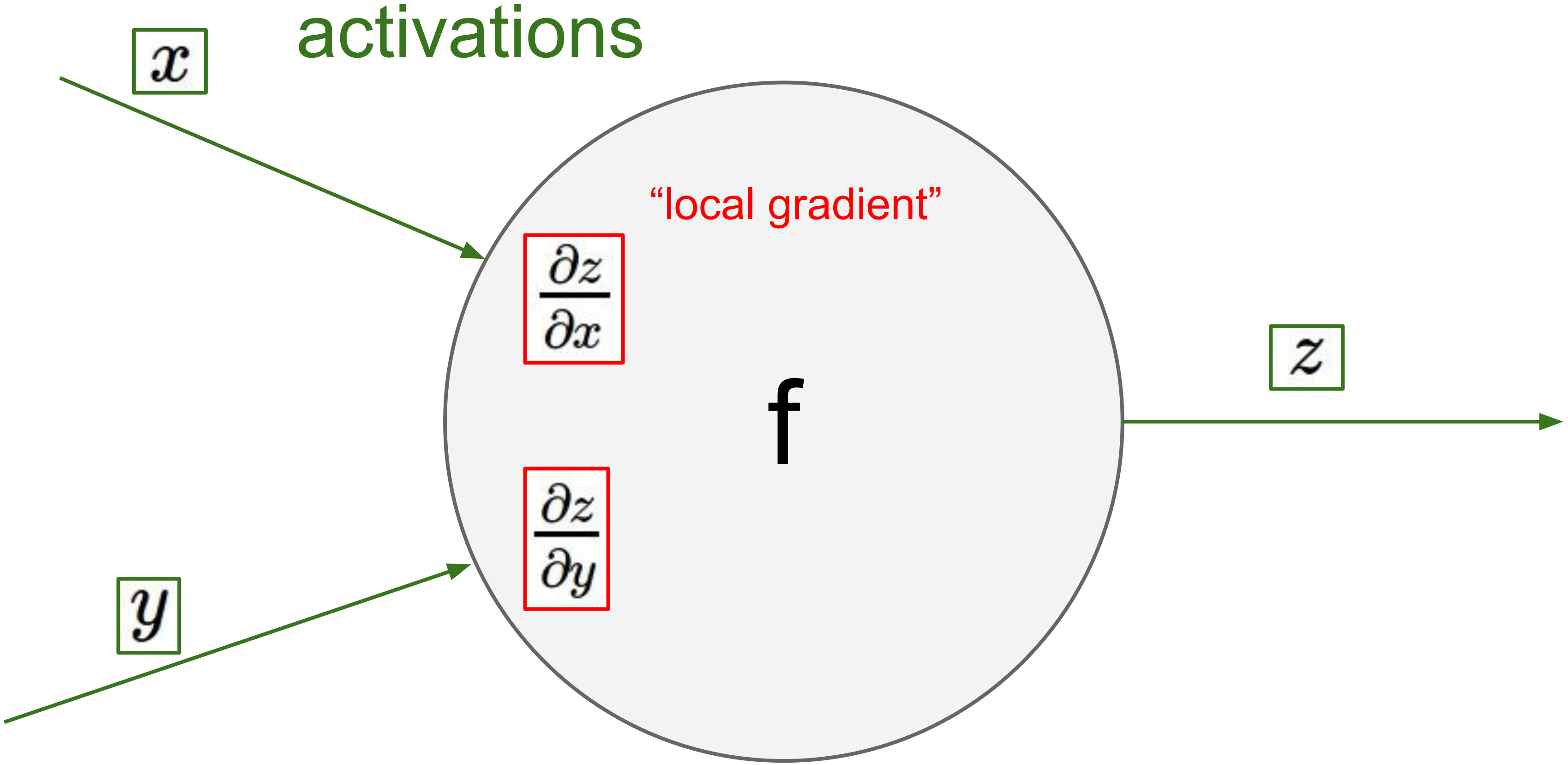
Chain rule:

$$\boxed{\frac{df}{dx} = \frac{df}{dq}\frac{dq}{dx} = -4 * 1 = -4}$$

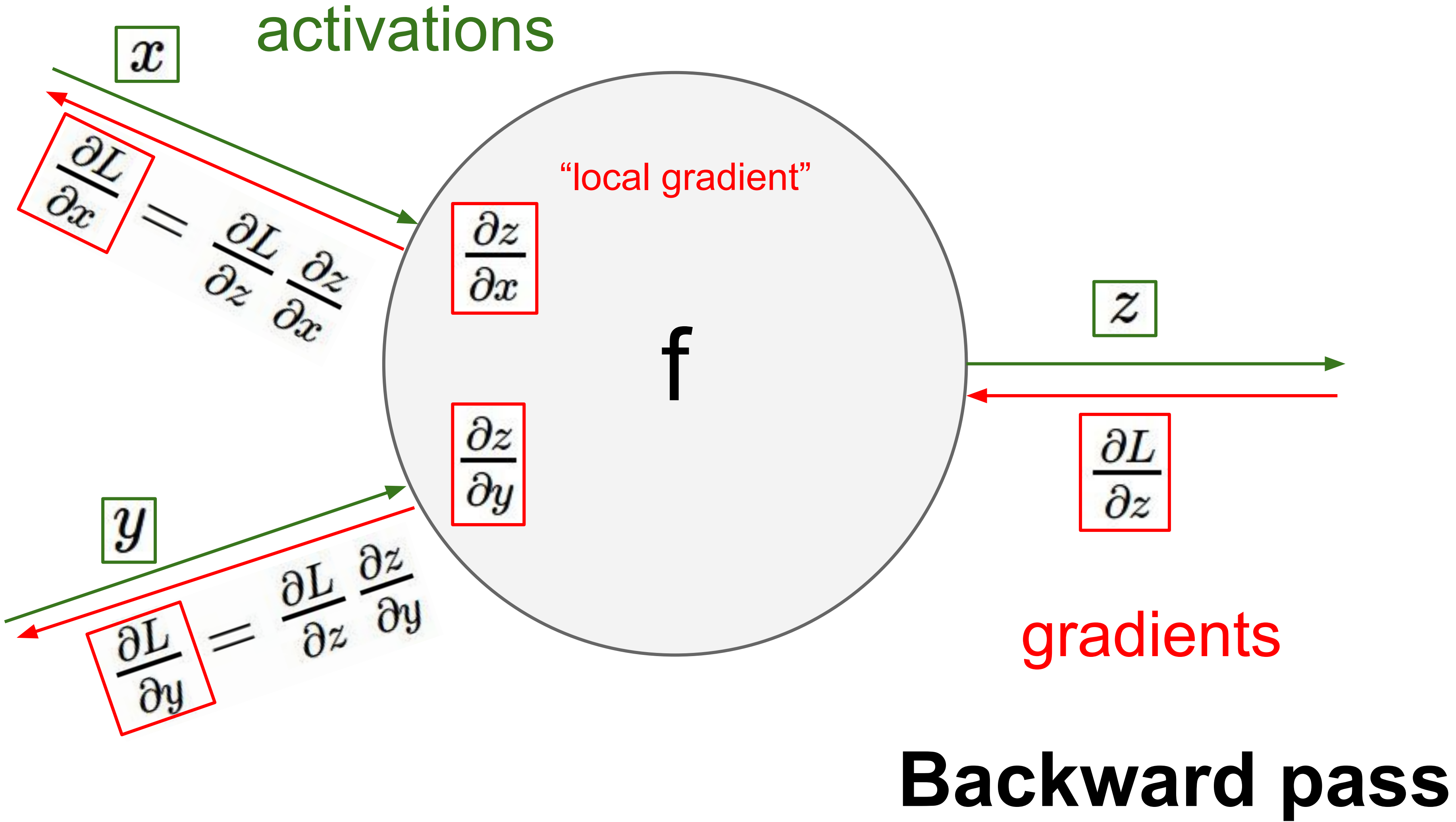
Implementation: forward/backward API




```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```



Forward pass



activations
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
z
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients
Backward pass

Example: Torch MulConstant

$$f(X) = aX$$

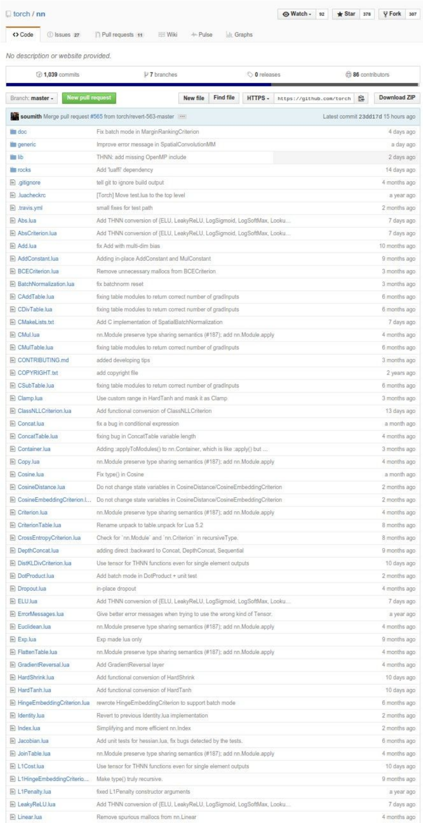
```lua
local MulConstant, parent = torch.class('nn.MulConstant', 'nn.Module')

function MulConstant:__init(constant_scalar,ip)
  parent.__init(self)
  assert(type(constant_scalar) == 'number', 'input is not scalar!')
  self.constant_scalar = constant_scalar

  -- default for inplace is false
   self.inplace = ip or false
   if (ip and type(ip) ~= 'boolean') then
      error('in-place flag must be boolean')
   end
end

function MulConstant:updateOutput(input)
  if self.inplace then
    input:mul(self.constant_scalar)
    self.output = input
  else
    self.output:resizeAs(input)
    self.output:copy(input)
    self.output:mul(self.constant_scalar)
  end
  return self.output
end

function MulConstant:updateGradInput(input, gradOutput)
  if self.gradInput then
    if self.inplace then
      gradOutput:mul(self.constant_scalar)
      self.gradInput = gradOutput
      -- restore previous input value
      input:div(self.constant_scalar)
    else
      self.gradInput:resizeAs(gradOutput)
      self.gradInput:copy(gradOutput)
      self.gradInput:mul(self.constant_scalar)
    end
    return self.gradInput
  end
end
```

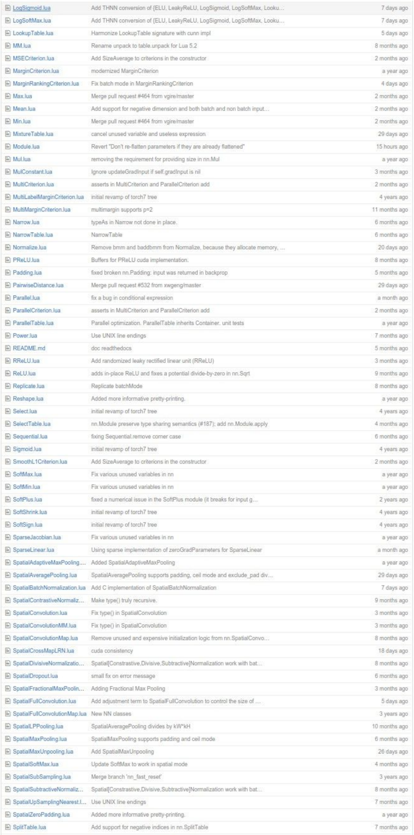
initialization

forward()

backward()

TensorFlow Layers



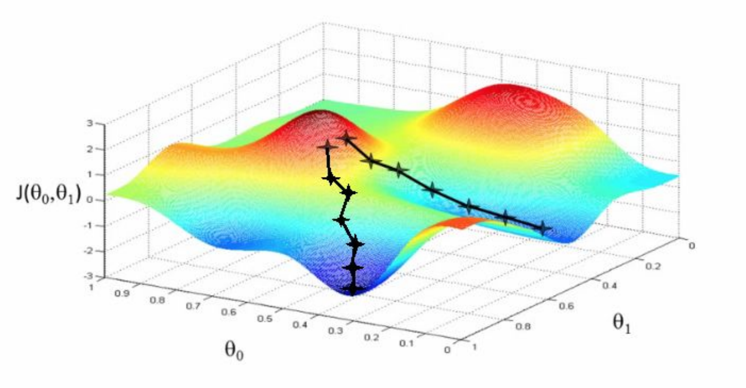


=

Optimización: Gradient descent

Iterar:

1. **Forward**: obtener la pérdida (Loss).
2. **Backprop:** calcular las gradientes.
3. **Actualizar** los parámetros del modelo.

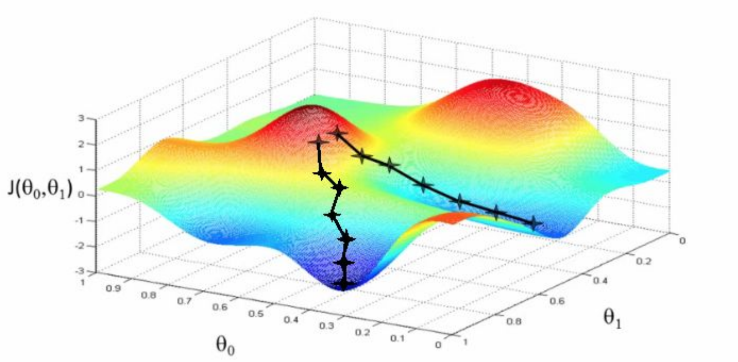


Optimización: Gradient descent

Iterar:

1. **Forward**: obtener la pérdida (Loss).
2. **Backprop:** calcular las gradientes.
3. **Actualizar** los parámetros del modelo.

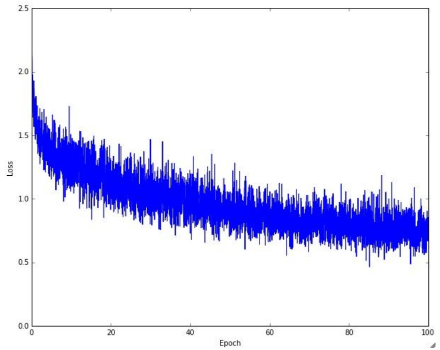
¿Y si tenemos mucha data?

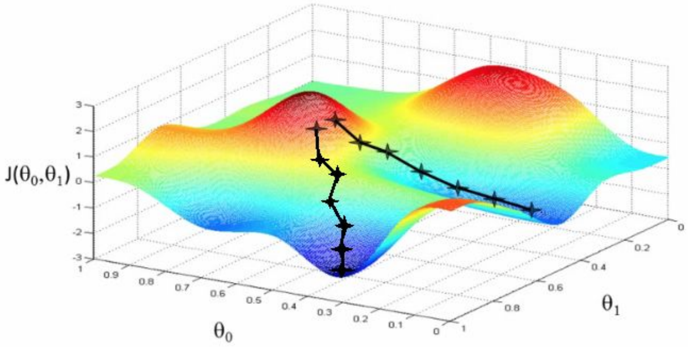


Optimización: Mini-batch Gradient descent

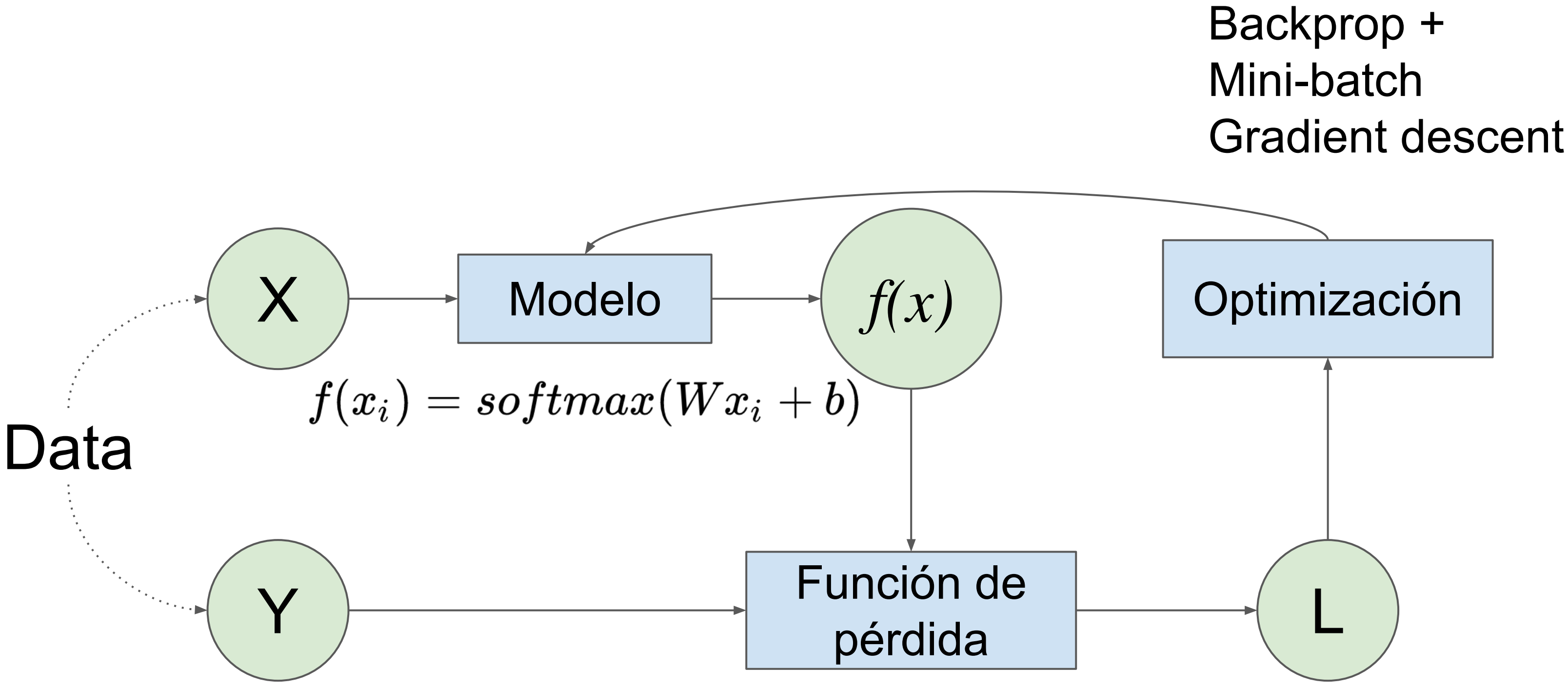
Iterar:

1. **Sample**: obtener una muestra de la data.
2. **Forward**: obtener la pérdida (Loss).
3. **Backprop:** calcular las gradientes.
4. **Actualizar** los parámetros del modelo.





Regresión logística



$$f(x_i) = softmax(Wx_i + b)$$

$$L = -\sum_{i=1}^{m} y_i log(\hat{y}_i)$$

Redes Neuronales

Redes Neuronales

Hasta ahora teníamos funciones lineales antes de aplicar softmax:

$$f(x_i) = softmax\boxed{(Wx_i + b)}$$

¿Cómo incrementamos la complejidad del modelo?

Redes Neuronales

Hasta ahora teníamos funciones lineales antes de aplicar softmax:

$$f(x_i) = softmax\boxed{(Wx_i + b)}$$

¿Cómo incrementamos la complejidad del modelo?

Podemos añadir operaciones intermedias

Redes Neuronales

$$\underbrace{\boxed{h(x_i)}}_{10xN} = \underbrace{\boxed{W_1}}_{10x4}\underbrace{\boxed{x_i}}_{4xN} + \underbrace{\boxed{b_1}}_{10x1}$$

Redes Neuronales

$$\boxed{h(x_i)} = \boxed{W_1}\boxed{x_i} + \boxed{b_1}$$

10xN 10x4 4xN 10x1

$$\boxed{f(x_i)} = softmax(\boxed{W_2}\boxed{h(x_i)} + \boxed{b_2})$$

3xN 3x10 10xN 3x1

Redes Neuronales

$$h_1(x_i) = W_1 x_i + b_1$$

$$h_2(x_i) = W_2 h_1(x_i) + b_2$$

$$\vdots$$

$$f(x_i) = softmax(W_m h_n(x_i) + b_m)$$

Redes Neuronales

$$h_1(x_i) = W_1 x_i + b_1$$

$$h_2(x_i) = W_2 h_1(x_i) + b_2$$

$$\vdots$$

$$f(x_i) = softmax(W_m h_n(x_i) + b_m)$$

Pero una serie de operaciones lineales se puede reducir a solo 1.

$$y_1 = 2x + 10$$

$$y_2 = 3y_1 - 5$$

$$y_2 = 6x + 25$$

Redes Neuronales

$$h_1(x_i) = \boxed{\sigma}(W_1 x_i + b_1)$$

$$h_2(x_i) = \boxed{\sigma}(W_2 h_1(x_i) + b_2)$$

$$\vdots$$

$$f(x_i) = softmax(W_m h_n(x_i) + b_m)$$

Pero una serie de operaciones lineales se puede reducir a solo 1.

Añadimos funciones de activación

Redes Neuronales

$$h_1(x_i) = \boxed{\sigma}(W_1 x_i + b_1)$$

$$h_2(x_i) = \boxed{\sigma}(W_2 h_1(x_i) + b_2)$$

$$\vdots$$

$$f(x_i) = \underline{softmax}(W_m h_n(x_i) + b_m)$$

Pero una serie de operaciones lineales se puede reducir a solo 1.

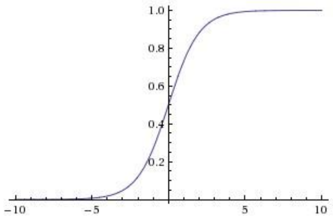
Añadimos funciones de activación

softmax también es una función de activación

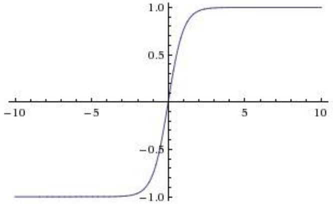
Funciones de Activación

Sigmoid

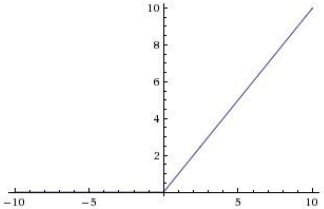
$$\sigma(x) = 1/(1 + e^{-x})$$



tanh tanh(x)

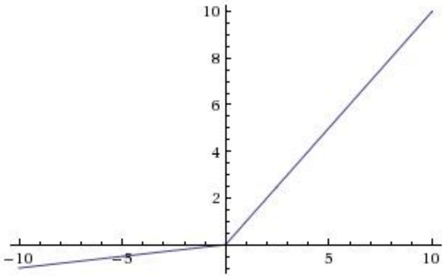


ReLU max(0,x)



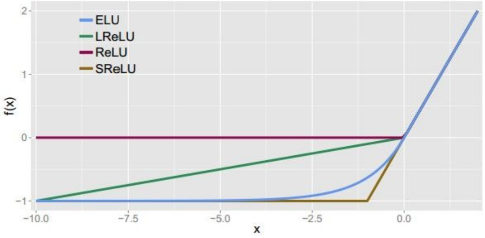
Leaky ReLU

max(0.1x, x)



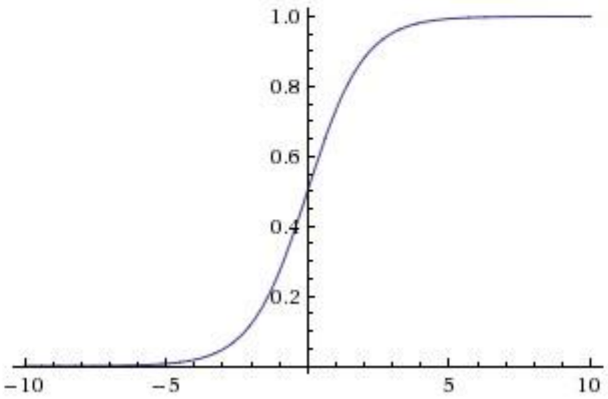
ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$



Funciones de Activación

$$\sigma(x) = 1/(1+e^{-x})$$

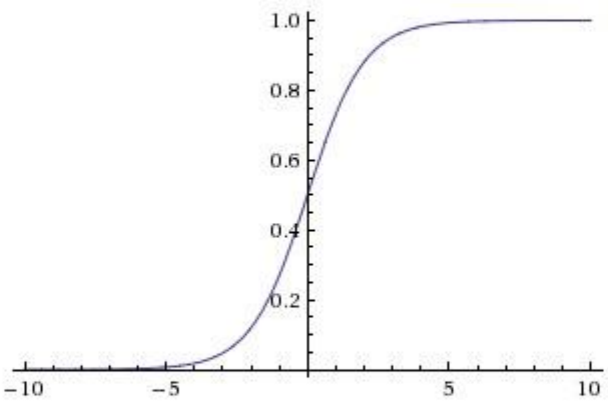


Sigmoid

- Reduce el rango a [0,1]
- Históricamente popular, por la interpretación de “activar” una neurona.

Funciones de Activación

$$\sigma(x) = 1/(1 + e^{-x})$$

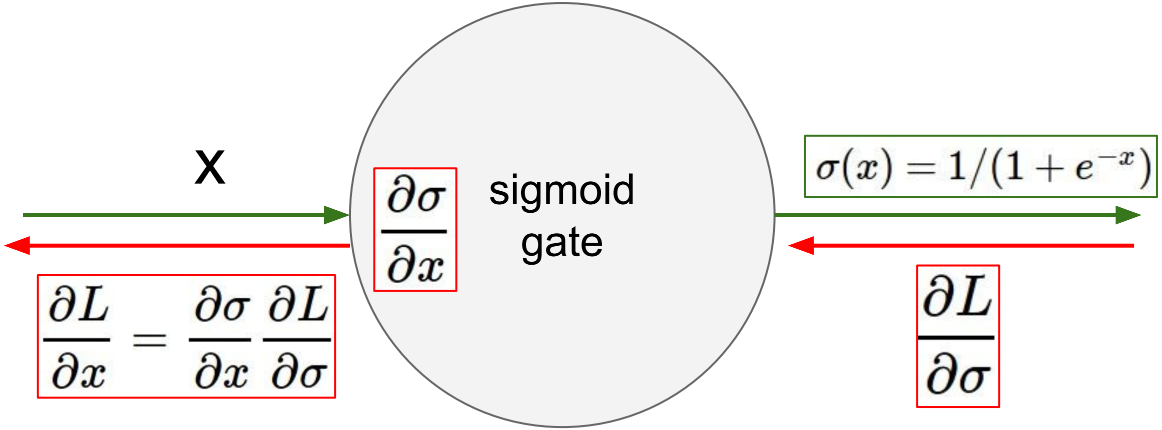


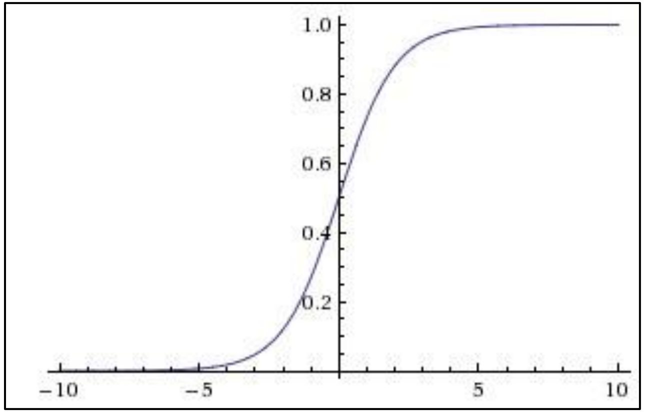
Sigmoid

- Reduce el rango a [0,1]
- Históricamente popular, por la interpretación de “activar” una neurona.

3 problemas:

1. Las neuronas saturadas “matan” las gradientes.



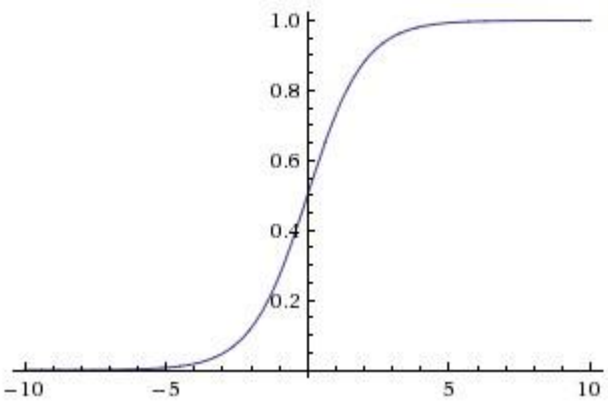


¿Qué pasa cuando x = -10?

¿Qué pasa cuando x = 10?

Funciones de Activación

$$\sigma(x) = 1/(1 + e^{-x})$$



Sigmoid

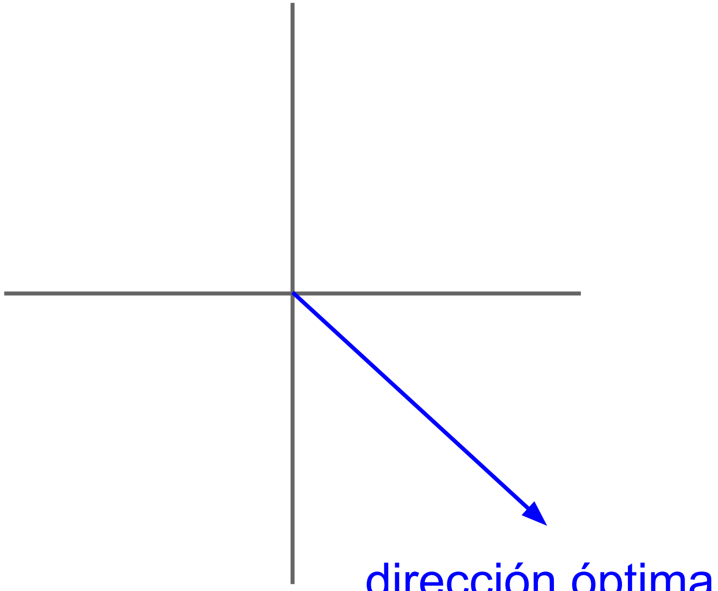
- Reduce el rango a [0,1]
- Históricamente popular, por la interpretación de “activar” una neurona.

3 problemas:

1. Las neuronas saturadas “matan” las gradientes.
2. El outputs no está centrado en cero.

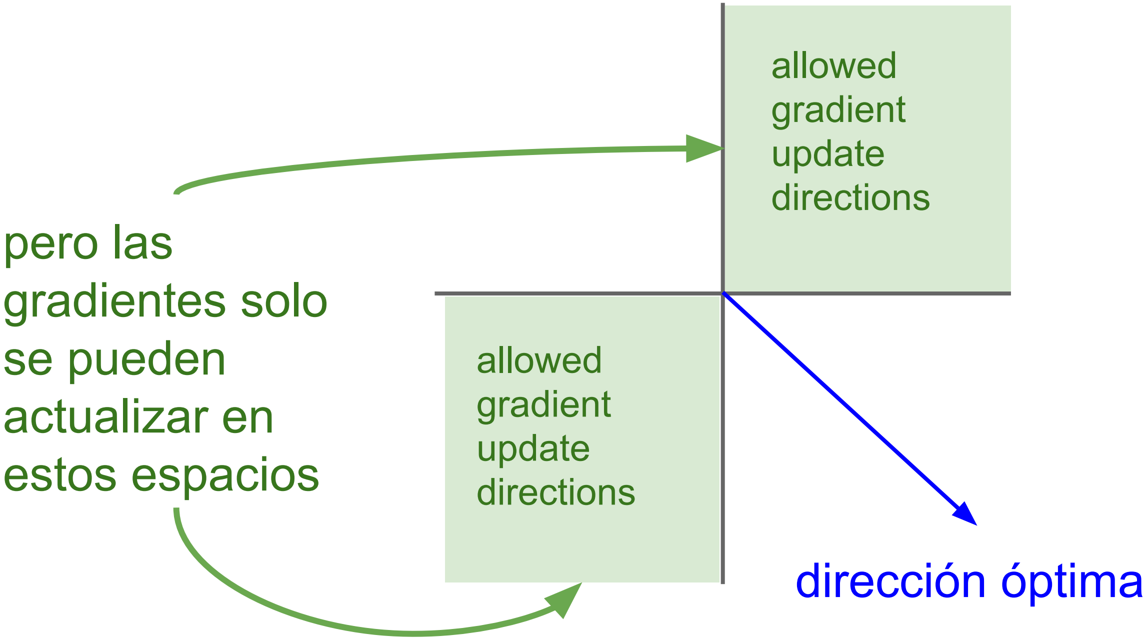
Cuando el input (x) de una neurona siempre es positivo, las gradientes van a ser todas positivas o todas negativas.

Espacio de optimización de la gradiente:



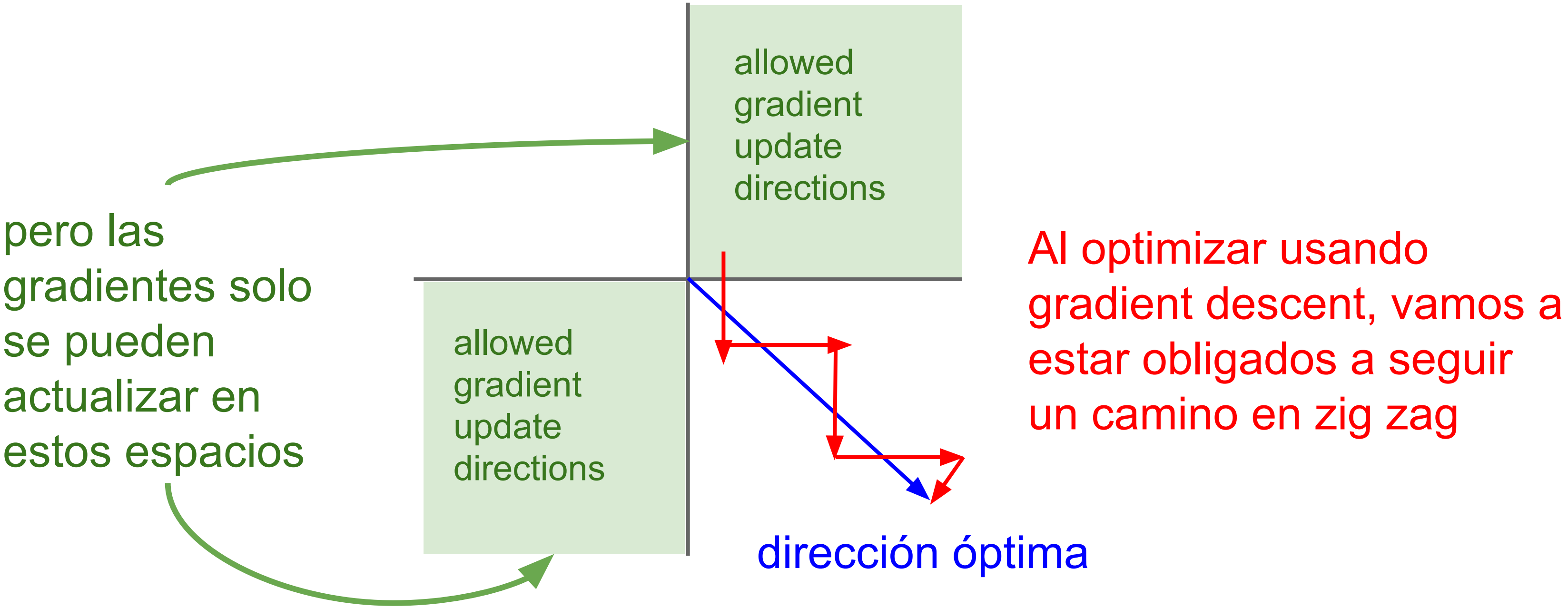
Cuando el input (x) de una neurona siempre es positivo, las gradientes van a ser todas positivas o todas negativas.

Espacio de optimización de la gradiente:



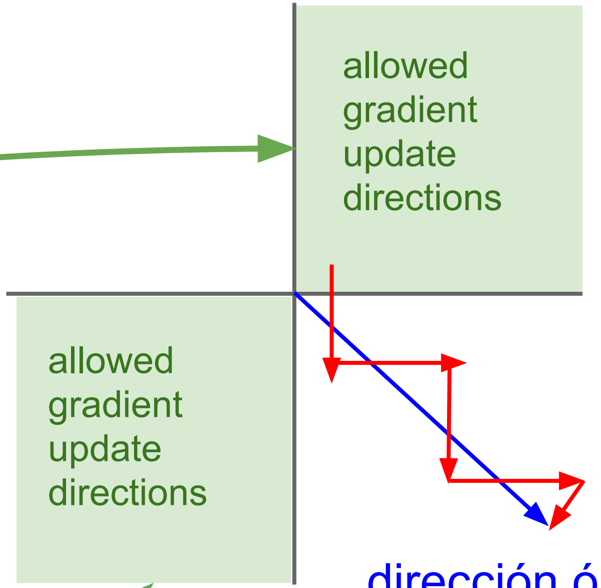
Cuando el input (x) de una neurona siempre es positivo, las gradientes van a ser todas positivas o todas negativas.

Espacio de optimización de la gradiente:



Cuando el input (x) de una neurona siempre es positivo, las gradientes van a ser todas positivas o todas negativas.

Espacio de optimización de la gradiente:



pero las gradientes solo se pueden actualizar en estos espacios

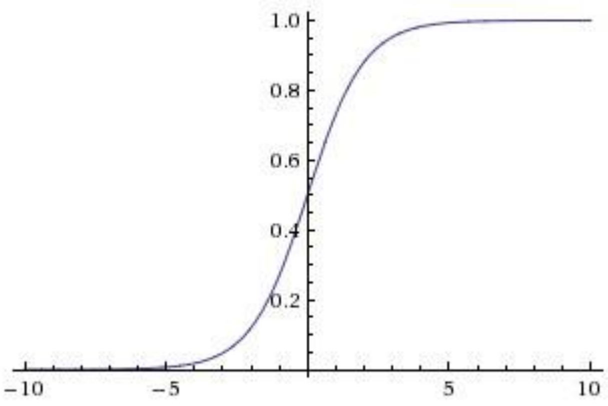
Al optimizar usando gradient descent, vamos a estar obligados a seguir un camino en zig zag

dirección óptima

(este es uno de los motivos por lo que normalizamos la data)

Funciones de Activación

$$\sigma(x) = 1/(1+e^{-x})$$



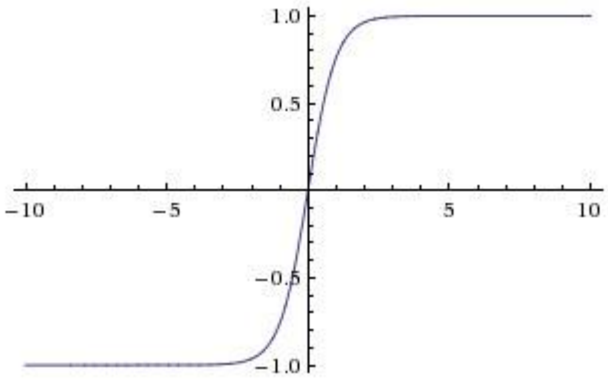
Sigmoid

- Reduce el rango a [0,1]
- Históricamente popular, por la interpretación de "activar" una neurona.

3 problemas:

1. Las neuronas saturadas "matan" las gradientes.
2. El outputs no está centrado en cero.
3. La función exponencial es pesada de calcular.

Funciones de Activación

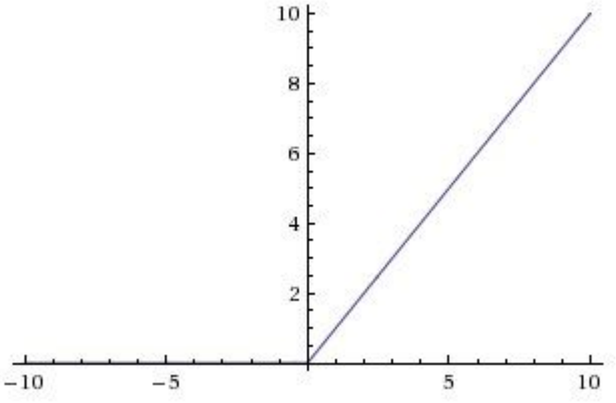


$$\tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

- Reduce el rango a [-1,1]
- Centrada en cero :D
- Neuronas saturadas :(

Funciones de Activación

relu(x) = max(0,x)



ReLU
(Rectified Linear Unit)

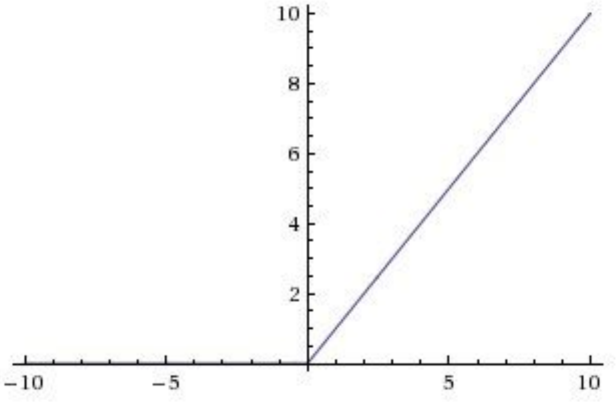
- No se satura (en los positivos).
- Computacionalmente eficiente.

- No está centrada en cero.
- Las gradientes cuando x<0 es ?

[Krizhevsky et al., 2012]

Funciones de Activación

relu(x) = max(0,x)



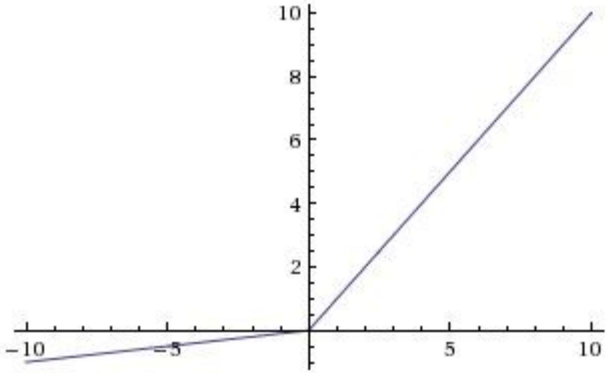
ReLU
(Rectified Linear Unit)

- No se satura (en los positivos).
- Computacionalmente eficiente.

- No está centrada en cero.
- Las gradientes cuando x<0 es 0.

[Krizhevsky et al., 2012]

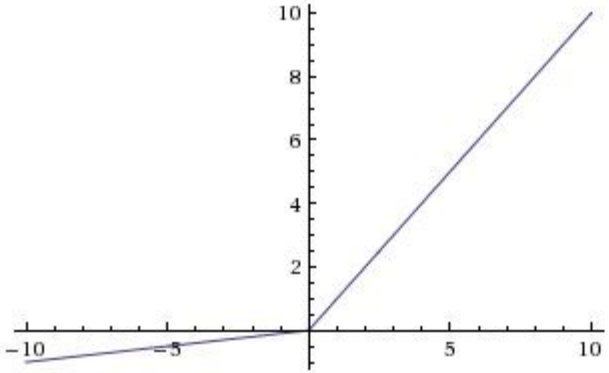
Funciones de Activación



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Funciones de Activación



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Parametric Rectifier (PReLU)

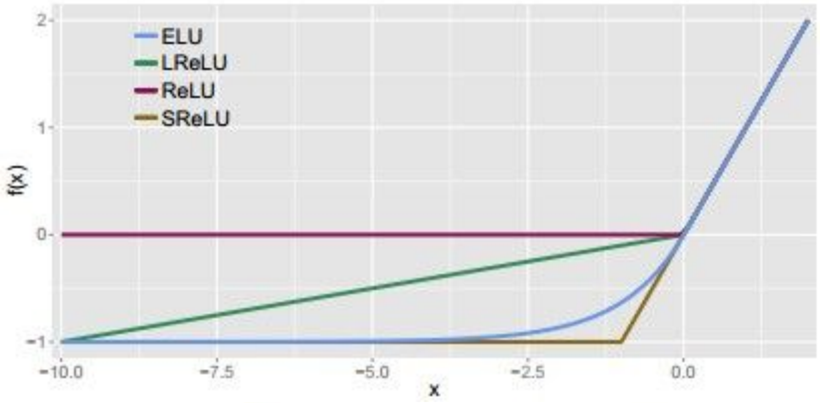
$$f(x) = \max(\alpha x, x)$$

alpha es un parámetro a optimizar

[Mass et al., 2013]
[He et al., 2015]

Funciones de Activación

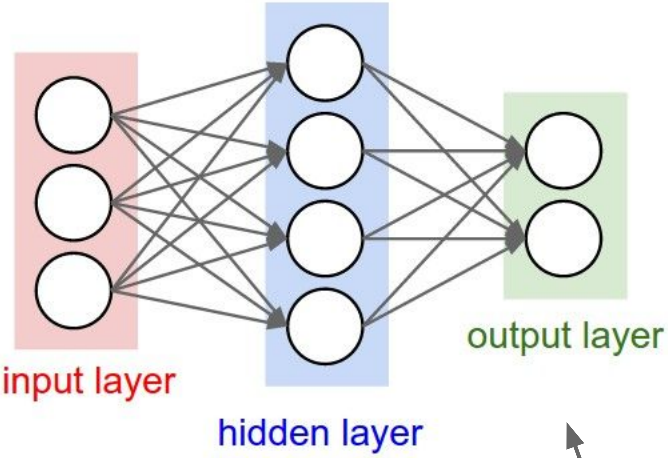
Exponential Linear Units (ELU)



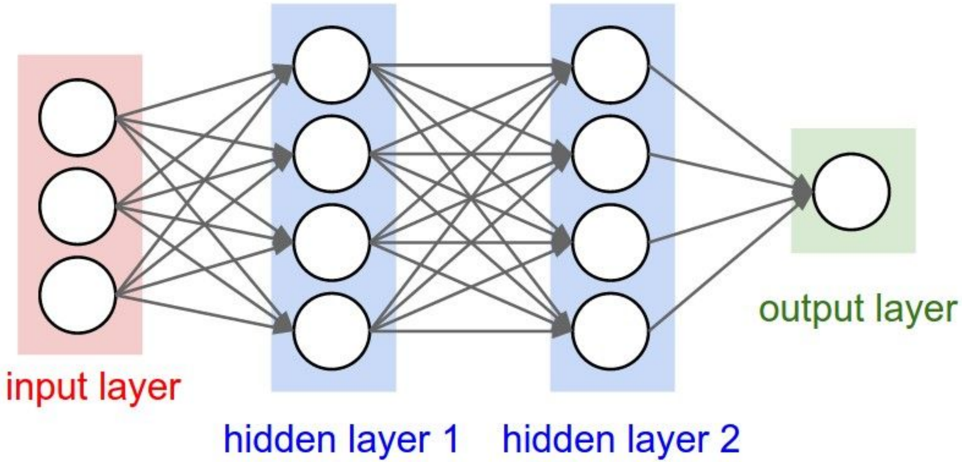
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

[Clevert et al., 2015]

Redes Neuronales



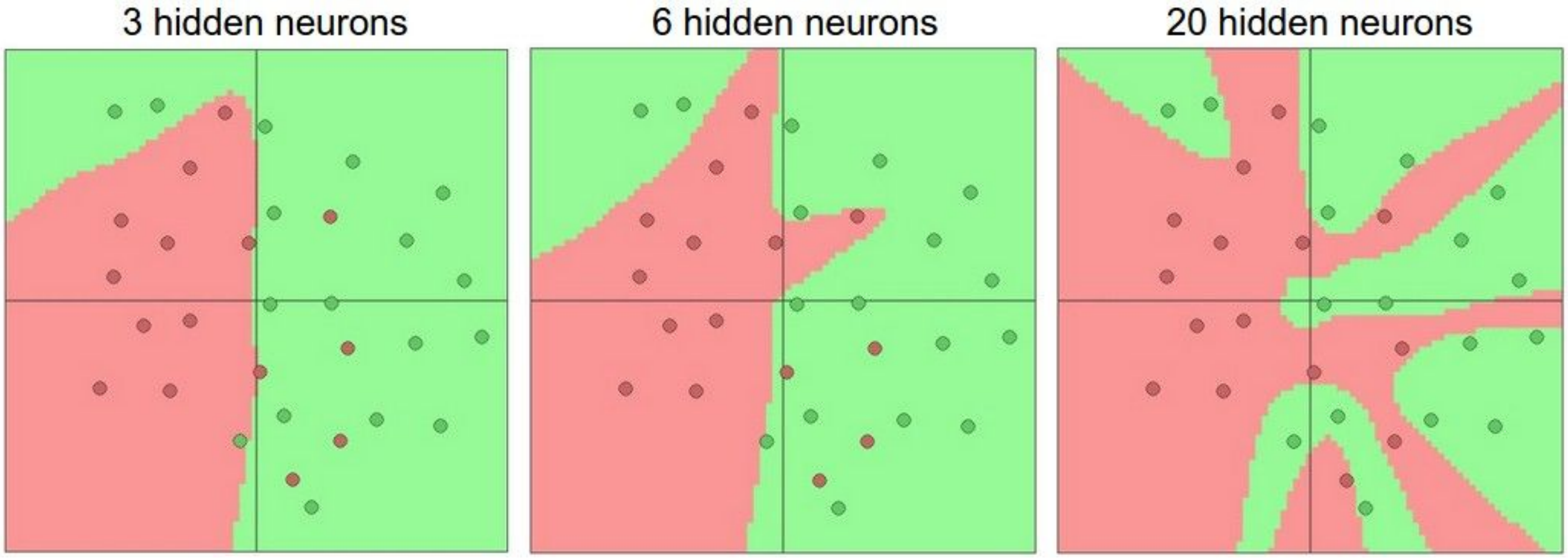
“2-layer Neural Net”, or
“1-hidden-layer Neural Net”



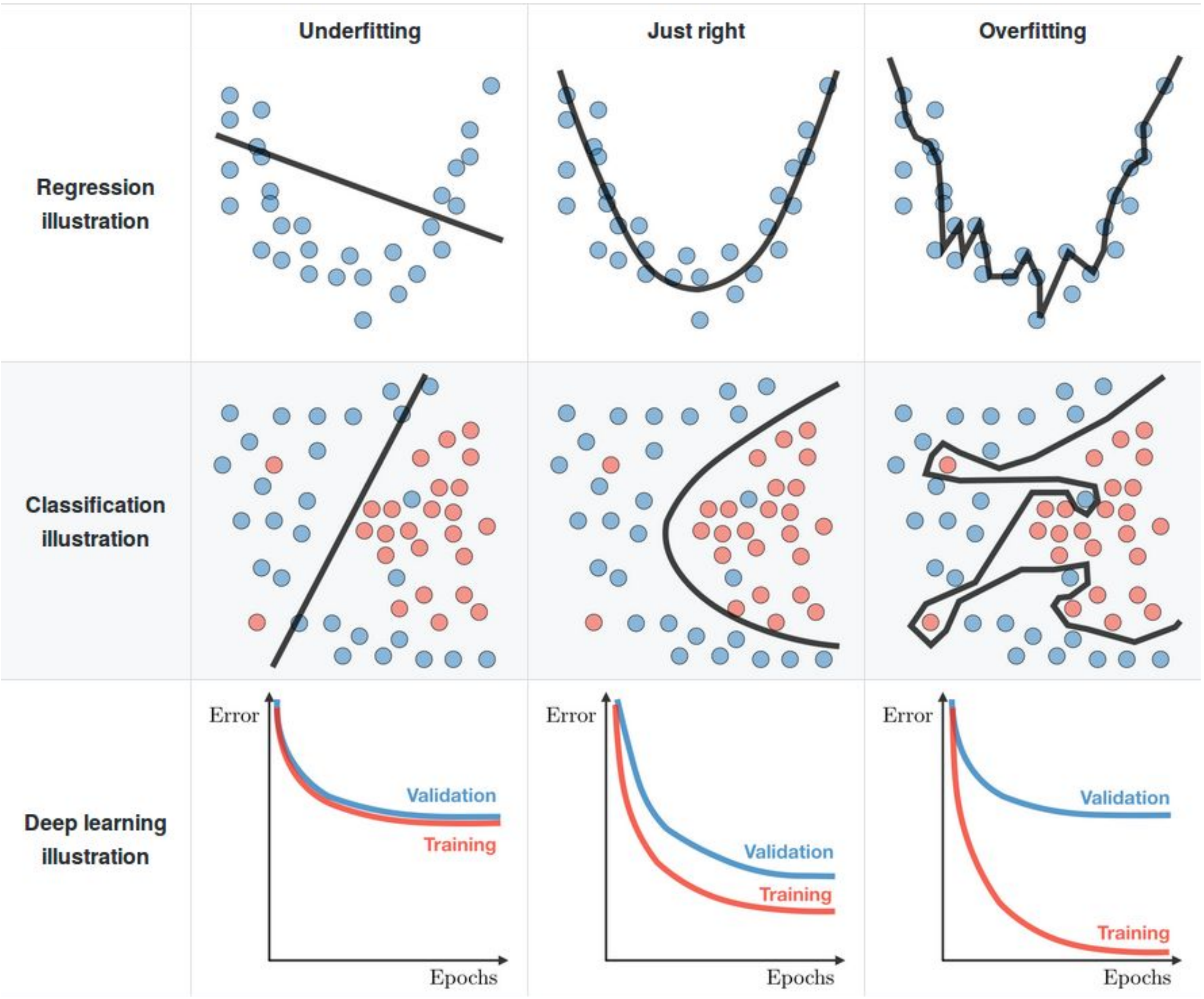
“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

“Fully-connected” layers

¿Número de capas, de neuronas?



de neuronas?.... *hiperparametros*



Underfitting
Just right
Overfitting
Regression illustration
Classification illustration
Deep learning illustration
Error
Validation
Training
Epochs
Error
Validation
Training
Epochs
Error
Validation
Training
Epochs

	Underfitting	Just right	Overfitting
Symptoms	• High training error • Training error close to test error • High bias	• Training error slightly lower than test error	• Very low training error • Training error much lower than test error • High variance
Possible remedies	• Complexify model • Add more features • Train longer		• Perform regularization • Get more data

Redes Neuronales