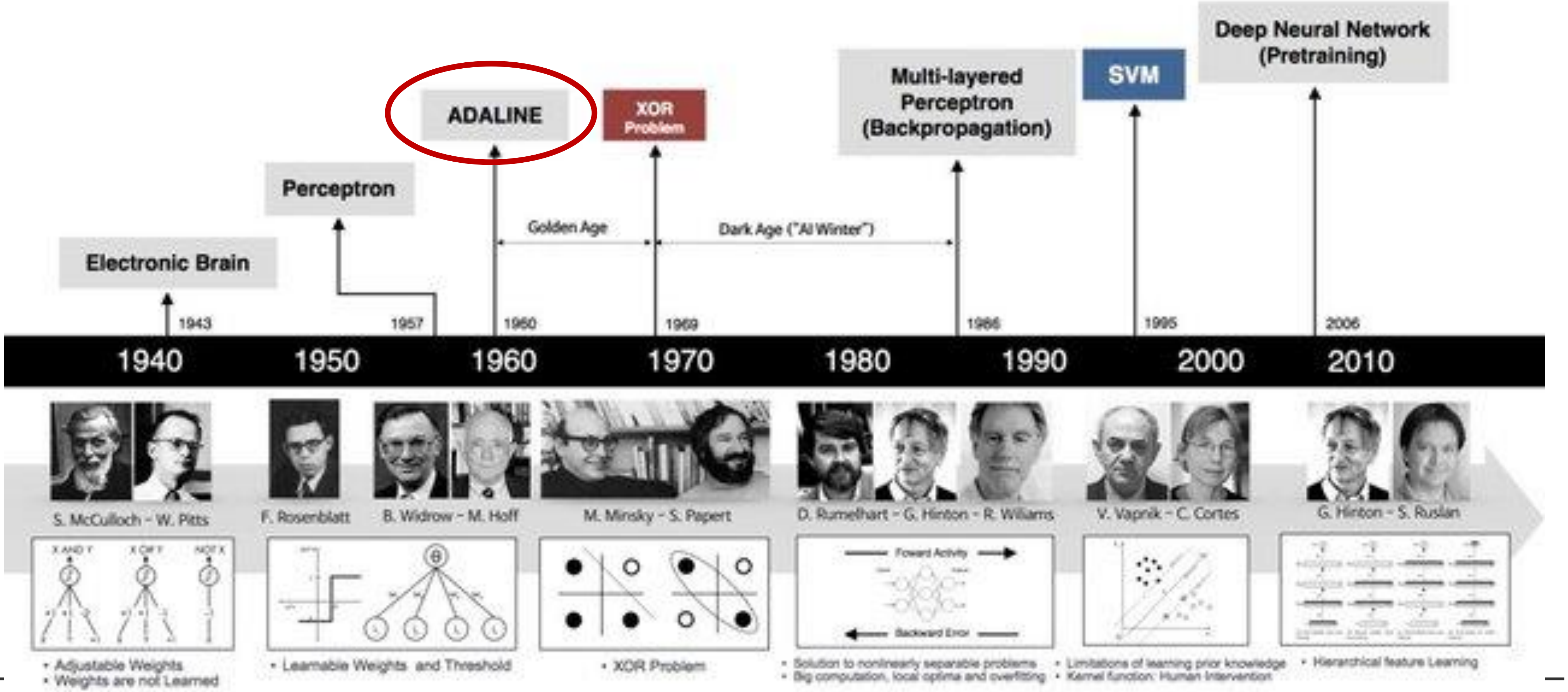
Redes Neuronales



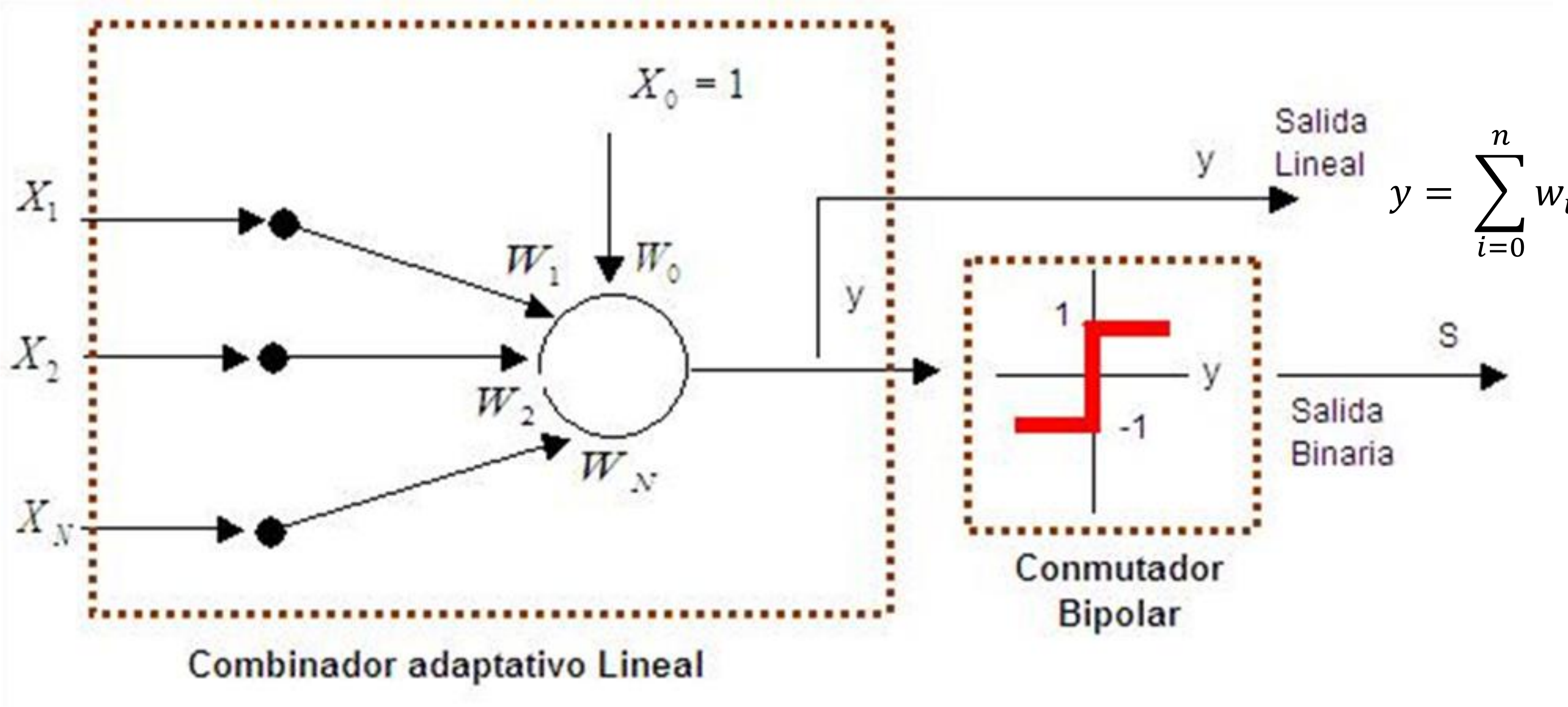
Red ADALINE (**ADA**ptative **LIN**ear **E**lement)

- Red neuronal formada por una sola neurona desarrollada por Widrow en 1960 al mismo tiempo que Rosenblatt trabajaba en el modelo del Perceptrón.

- Adapta los pesos de las conexiones **teniendo en cuenta el error** cometido al responder con respecto al valor esperado.

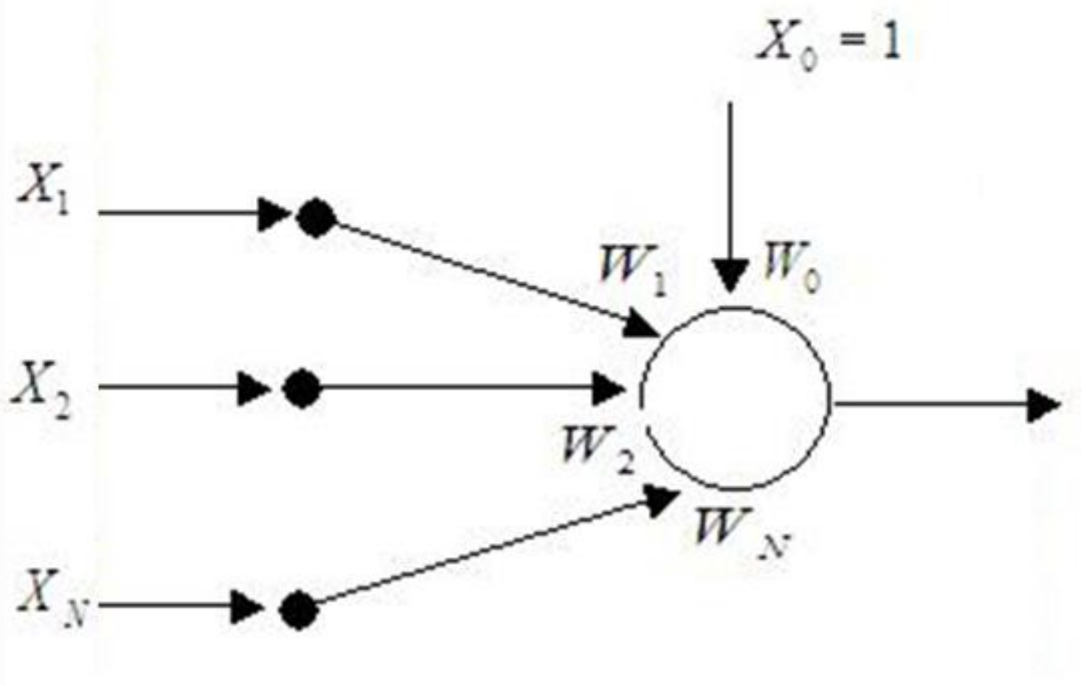
- Utiliza el **algoritmo LMS** (Least Mean Square) o **regla delta** para determinar los pesos de los arcos a fin de minimizar el error cuadrático medio.

Red ADALINE



$$y = \sum_{i=0}^{n} w_i x_i$$

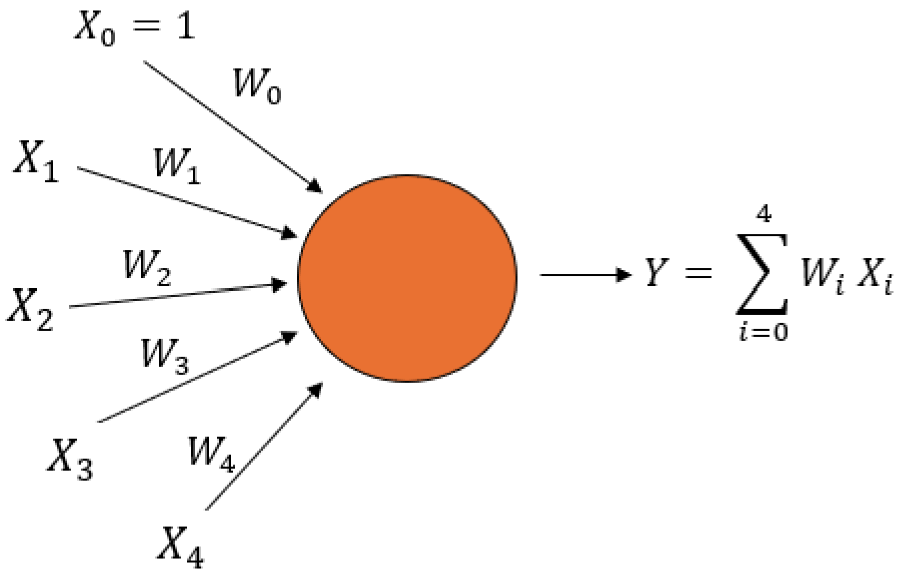
Combinador Adaptativo Lineal



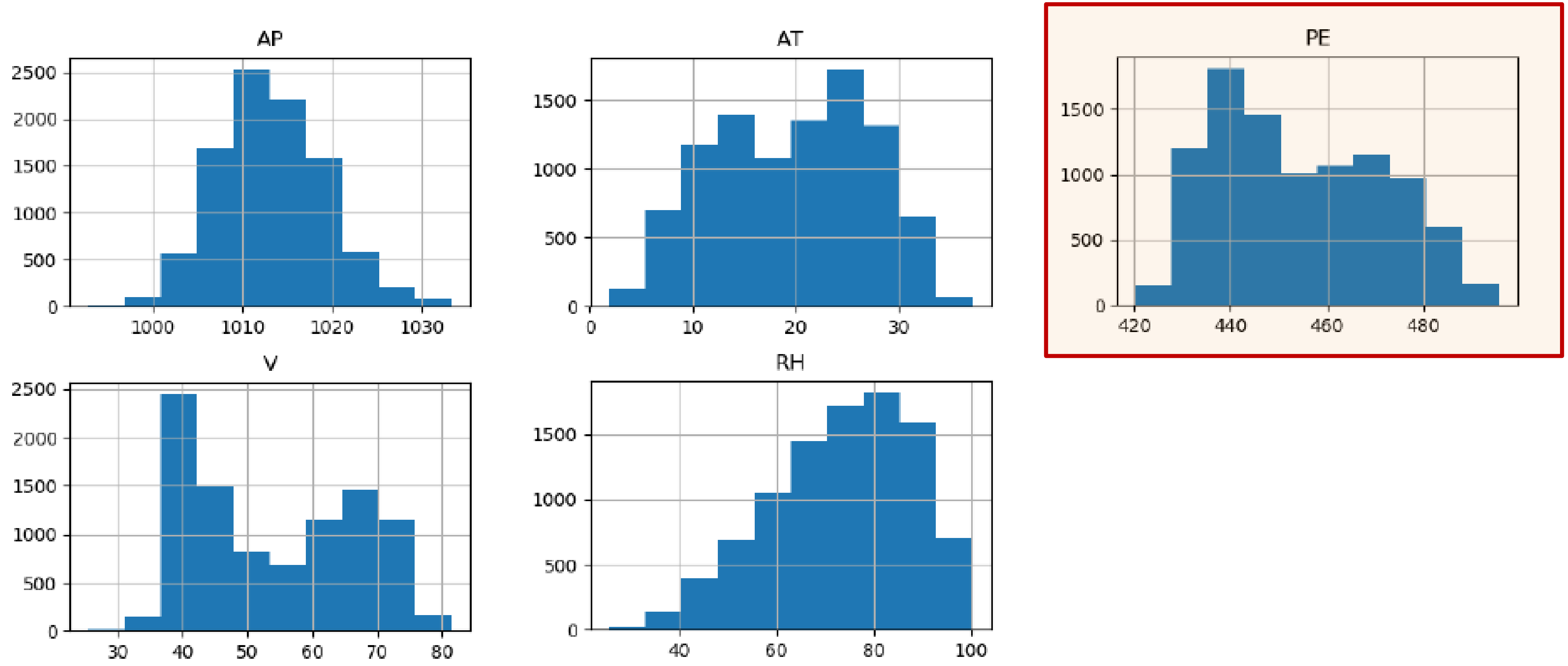
$$y = \sum_{i=0}^{n} w_i x_i$$

Ejercicio: Central de energía eléctrica

- El archivo **CCPP.csv** contiene 9568 datos de una Central de Ciclo Combinado recolectados entre 2006 y 2011.
- Objetivo: Predecir la producción neta de energía eléctrica por hora (PE) de la planta
- Las características relevadas son:
  - Temperatura ambiente (AT)
  - Presión ambiente (AP)
  - Humedad relativa (RH)
  - Vacío de escape (V)



Ejercicio: Central de energía eléctrica



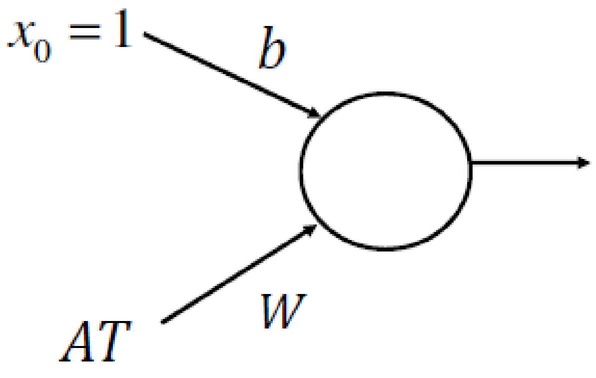
Ejercicio: Central de energía eléctrica

- Matriz de Correlación

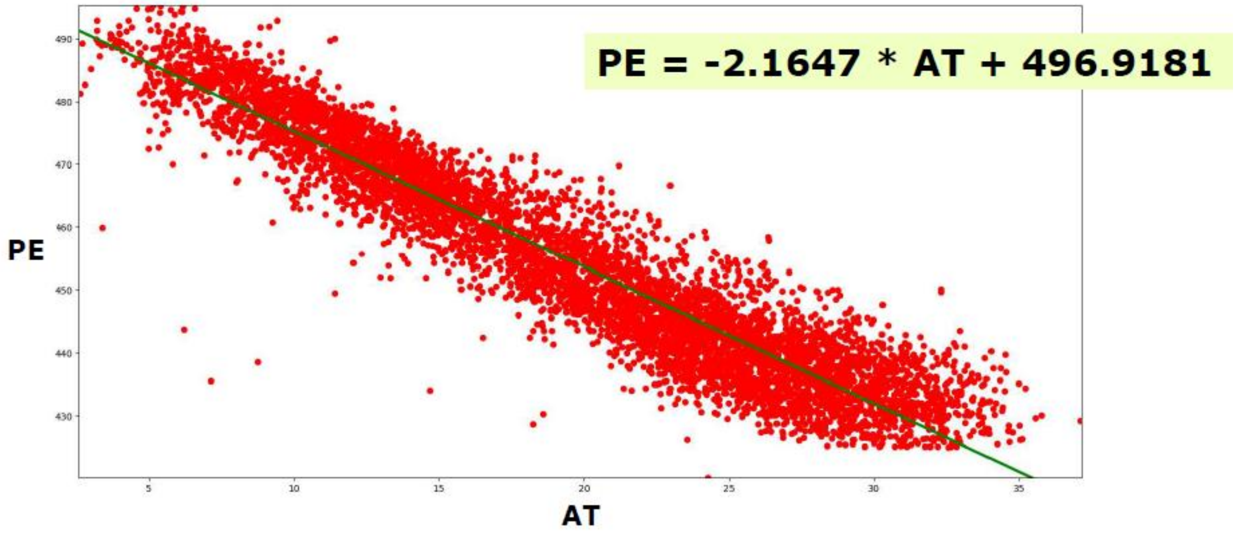
Index	AT	V	AP	RH	PE
AT	1	0.844107	-0.507549	-0.542535	-0.948128
V	0.844107	1	-0.413502	-0.312187	-0.86978
AP	-0.507549	-0.413502	1	0.0995743	0.518429
RH	-0.542535	-0.312187	0.0995743	1	0.389794
PE	-0.948128	-0.86978	0.518429	0.389794	1

EJERCICIO

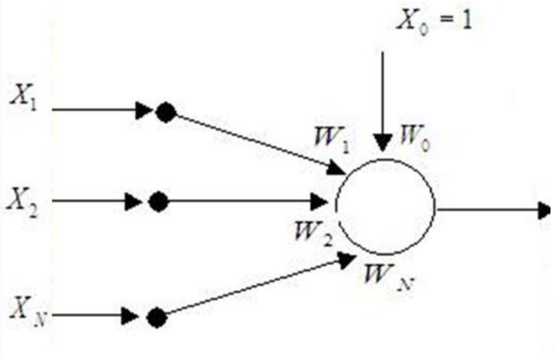
- A partir de los datos del archivo **CCPP.csv** utilice un combinador lineal para predecir la producción neta de energía eléctrica por hora (PE) de la planta a partir de la temperatura ambiente (AT)



$$PE = W * AT + b$$



Combinador Lineal



$$y = \sum_{i=0}^{n} w_i x_i$$

- Busca minimizar

$$\xi = <\varepsilon_k^2> = \frac{1}{L}\left[\sum_{k=1}^{L}\left(d_k - \sum_{i=0}^{N} x_{ik} w_i\right)^2\right]$$

donde

L : Cantidad de ejemplos

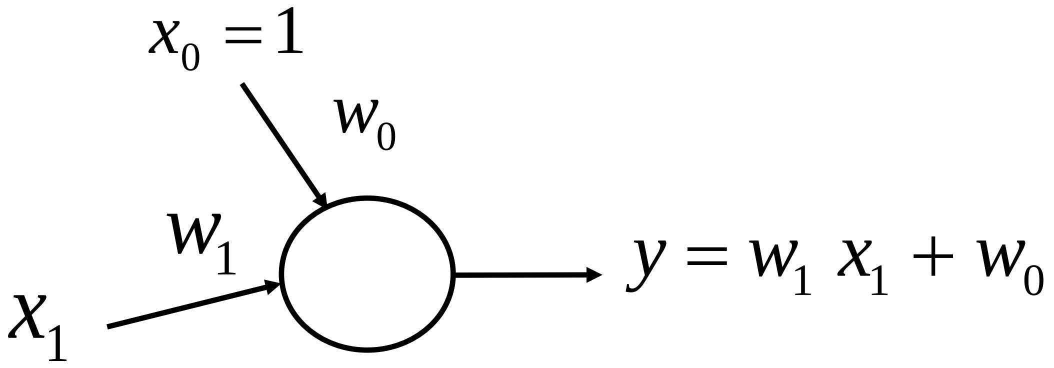
N : Cantidad de neuronas de entrada

$< >$ representa promedio

d_k : salida esperada para el ejemplo x_k

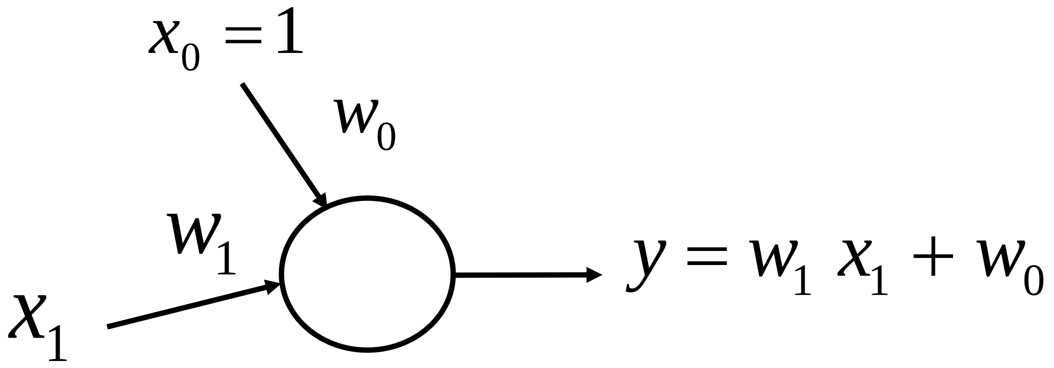
Ejemplo: Entrene un combinador lineal utilizando los siguientes ejemplos: (2,3), (1,1), (-1,-3).

- El combinador lineal será de la forma



Ejemplo: Entrene un combinador lineal utilizando los siguientes ejemplos: (2,3), (1,1), (-1,-3).

- El combinador lineal será de la forma

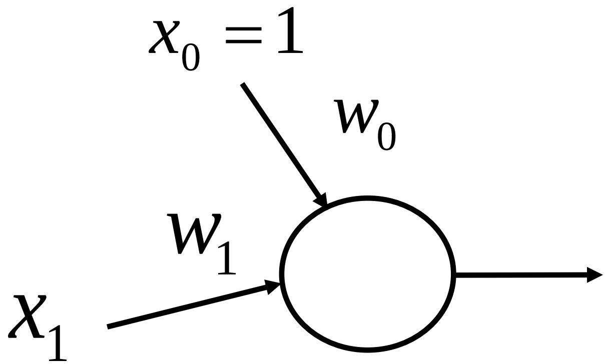


- Se busca determinar los valores de w_0 y w_1 que minimicen el error cuadrático medio

$$\xi = \frac{1}{3}\sum_{k=1}^{3}(d_k - y_k)^2$$

Ejemplo: Entrene un combinador lineal utilizando los siguientes ejemplos: (2,3), (1,1), (-1,-3).

- El combinador lineal será de la forma



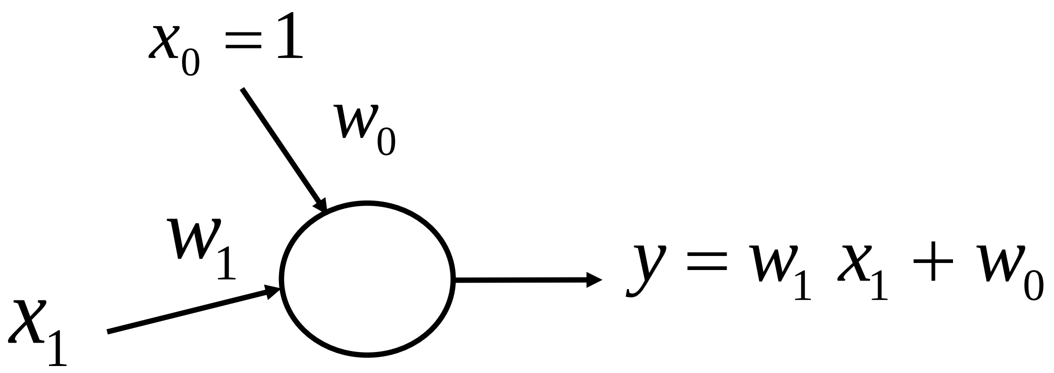
$$y = w_1\, x_1 + w_0$$

- Se busca determinar los valores de w_0 y w_1 que minimicen el error cuadrático medio

$$\xi = \frac{1}{3}\sum_{k=1}^{3}(d_k - y_k)^2 \quad = \frac{1}{3}\sum_{k=1}^{3}\left(d_k - \sum_{i=0}^{1} w_i x_{ik}\right)^2$$

Ejemplo: Entrene un combinador lineal utilizando los siguientes ejemplos: (2,3), (1,1), (-1,-3).

- El combinador lineal será de la forma



- Se busca determinar los valores de w_0 y w_1 que minimicen el error cuadrático medio

$$\xi = \frac{1}{3}\sum_{k=1}^{3}(d_k - y_k)^2 \quad = \frac{1}{3}\sum_{k=1}^{3}\left(d_k - \sum_{i=0}^{1} w_i x_{ik}\right)^2 \quad = \frac{1}{3}\sum_{k=1}^{3}(d_k - (w_1 x_k + w_o))^2$$

Ejemplo: Función de error a minimizar para los ejemplos: (2,3), (1,1), (-1,-3)

- Se busca minimizar

$$\xi = \frac{1}{3}\sum_{k=1}^{3}(d_k - y_k)^2 \qquad = \frac{1}{3}\sum_{k=1}^{3}(d_k - (w_1 x_k + w_o))^2$$

$$\xi = \frac{1}{3}\left(\left(3 - (w_1 2 + w_o)\right)^2 + \left(1 - (w_1 + w_o)\right)^2 + \left(-3 - (w_1(-1) + w_o)\right)^2\right)$$

$$\xi = \frac{1}{3}\left((3 - 2w_1 - w_0)^2 + (1 - w_1 - w_o)^2 + (-3 + w_1 - w_0)^2\right)$$

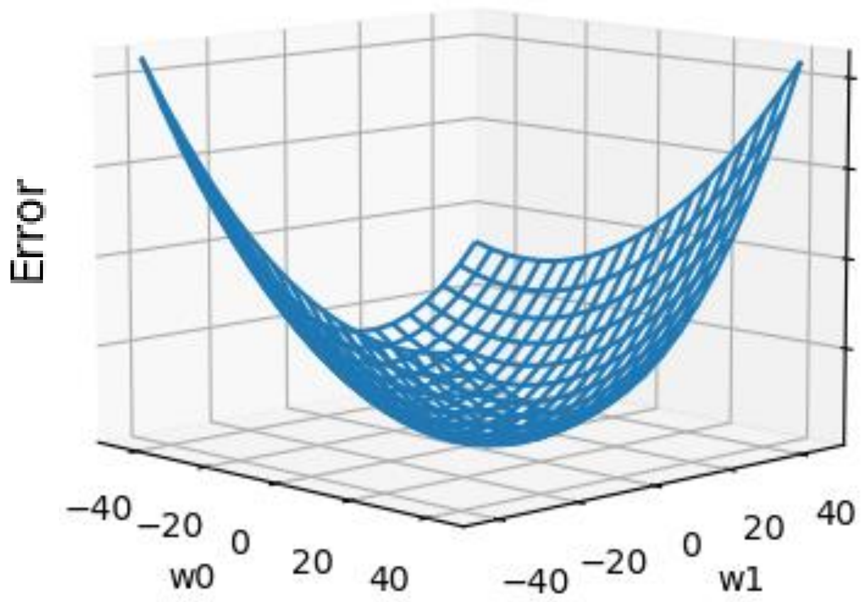
$$\xi = \frac{1}{3}(19 - 20w_1 - 2w_0 + 6w_1^2 + 4w_1 w_o + 3w_0^2)$$

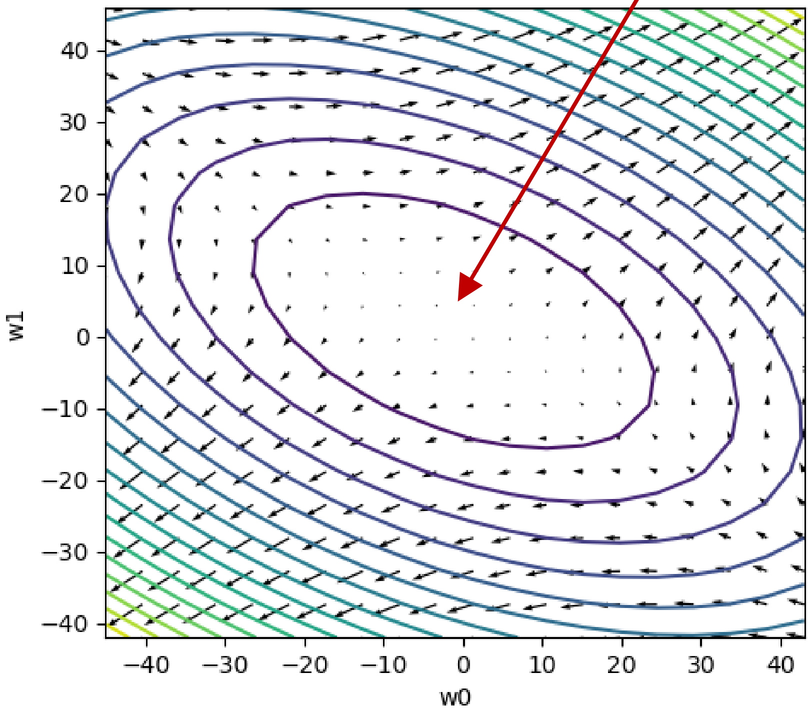
Ejemplo: Función de error a minimizar para los ejemplos: (2,3), (1,1), (-1,-3)

- Se busca minimizar la siguiente función

$$\xi = \frac{1}{3}(19 - 20w_1 - 2w_0 + 6w_1^2 + 4w_1w_0 + 3w_0^2)$$

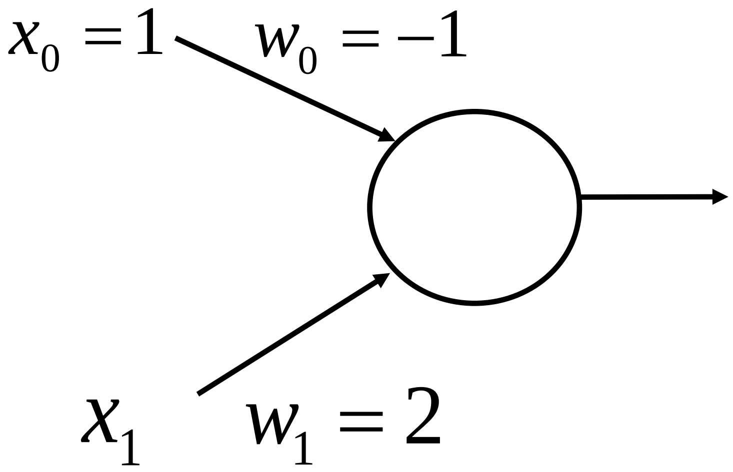
Mínimo en **w_0=-1**, **w_1=2**



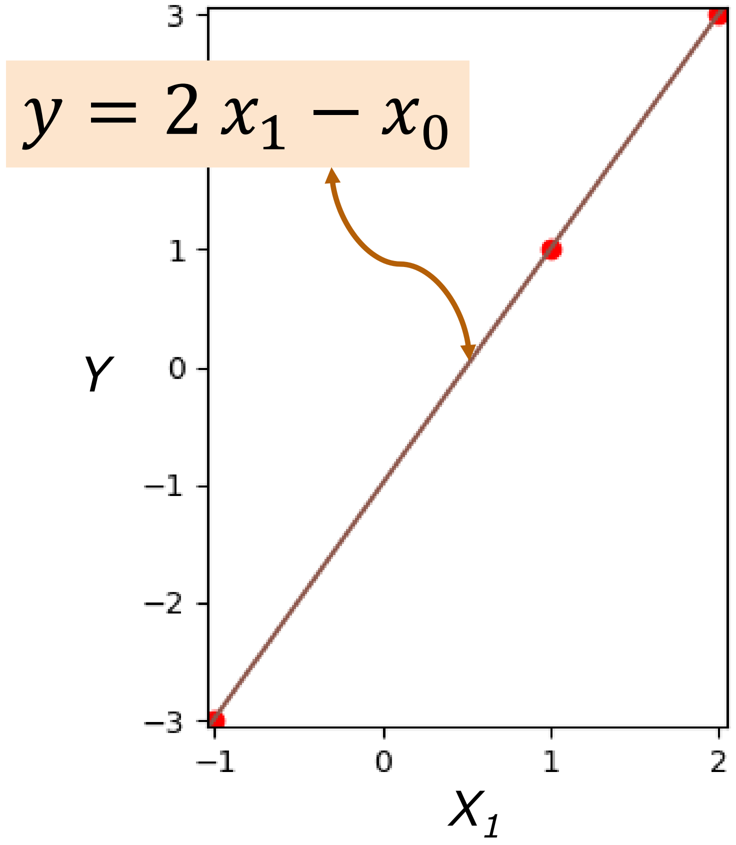


Ejemplo: Combinador lineal utilizando los siguientes ejemplos: (2,3), (1,1), (-1,-3).

- El Combinador Lineal a utilizar es el siguiente



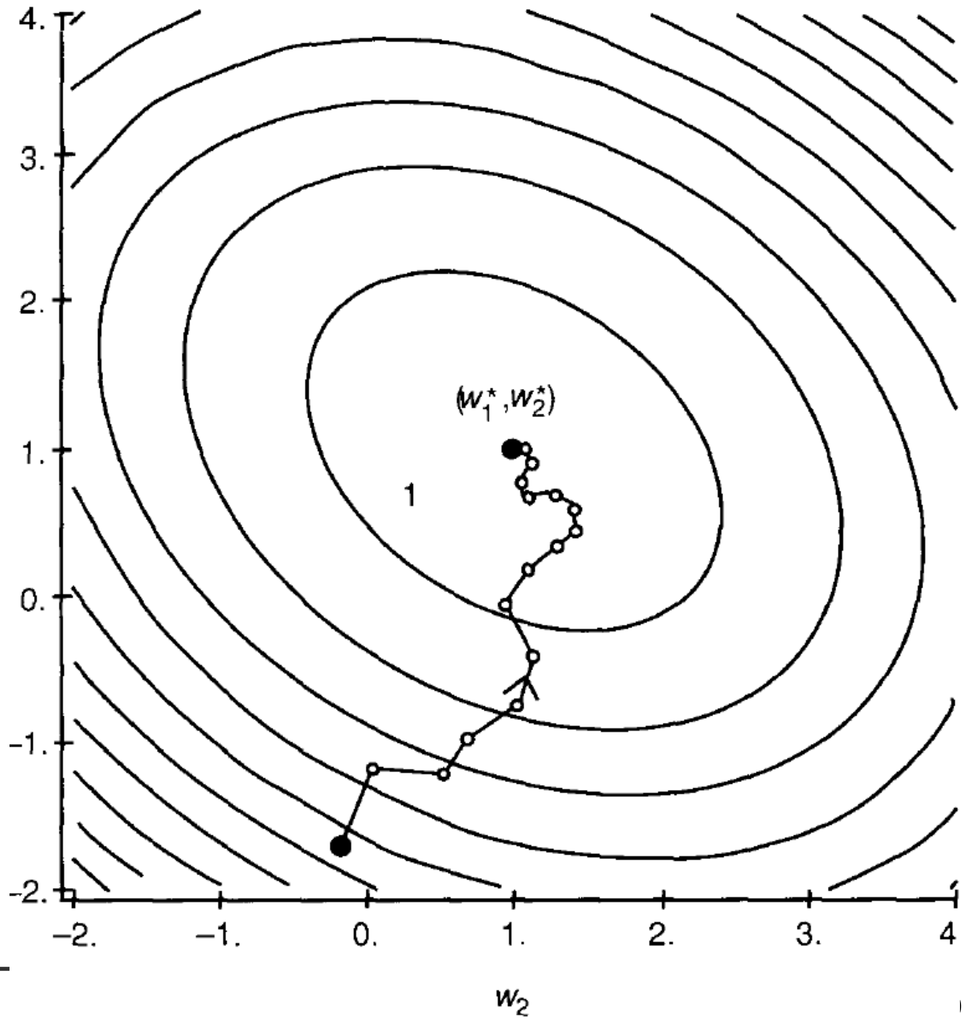
$$y = \sum_{i=0}^{1} w_i \, x_i$$



Entrenamiento del Combinador Lineal

- Se busca un método de entrenamiento que, a partir de los datos de entrada, permita calcular el vector W.

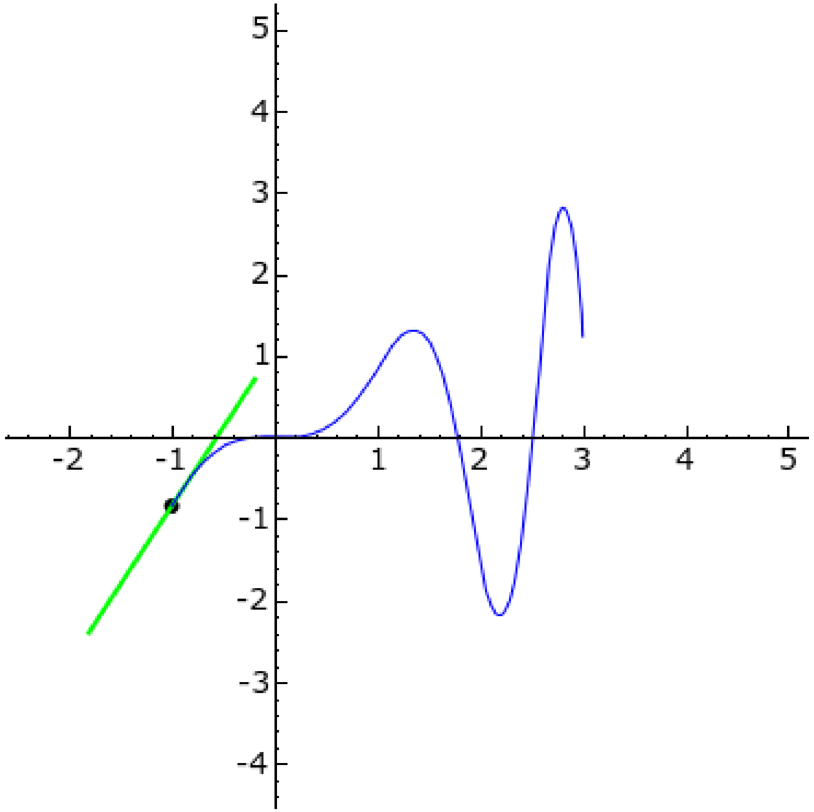
- Conceptos relacionados
  - Derivada
  - Derivada parcial
  - Vector gradiente



Derivada de una función

- Es una medida de la rapidez con la que cambia el valor de la función.

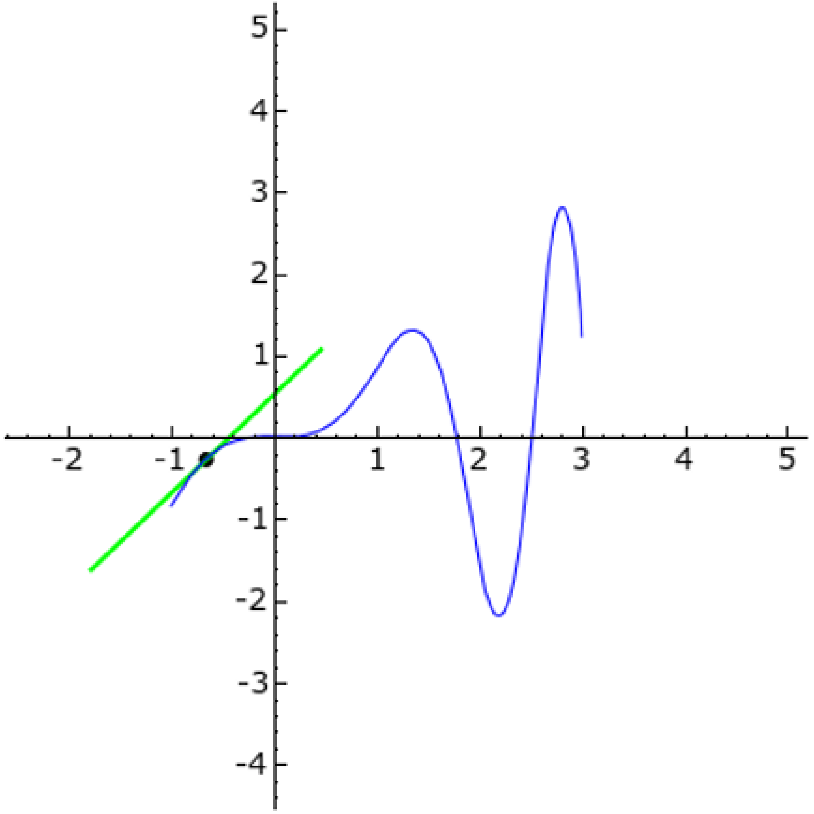
- Evaluada en un punto se corresponde con la pendiente de la recta tangente a la gráfica de la función en dicho punto.



Derivada de una función

- Es una medida de la rapidez con la que cambia el valor de la función.

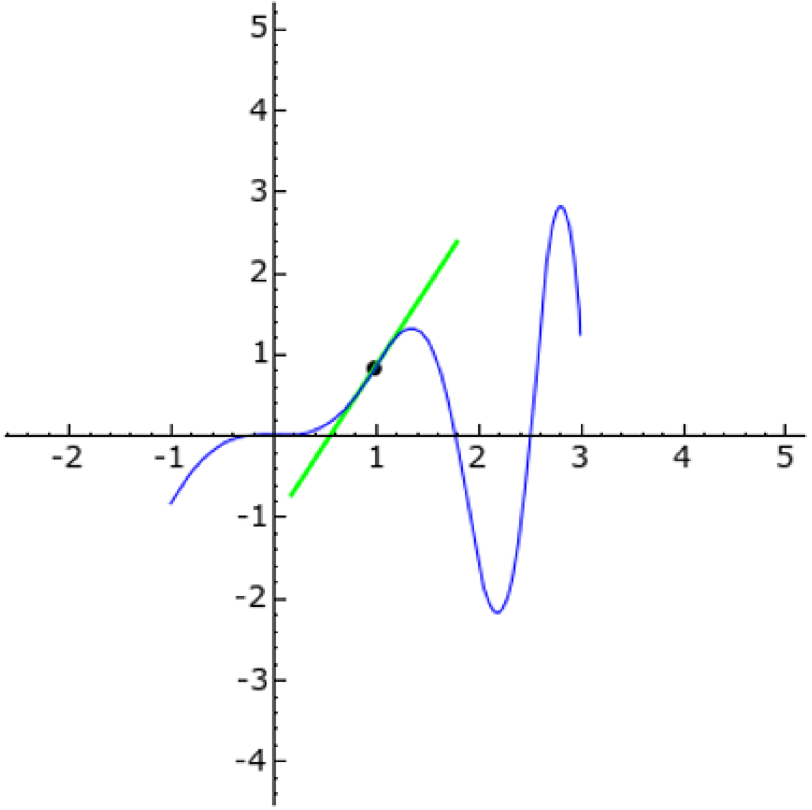
- Evaluada en un punto se corresponde con la pendiente de la recta tangente a la gráfica de la función en dicho punto.



Derivada de una función

- Es una medida de la rapidez con la que cambia el valor de la función.

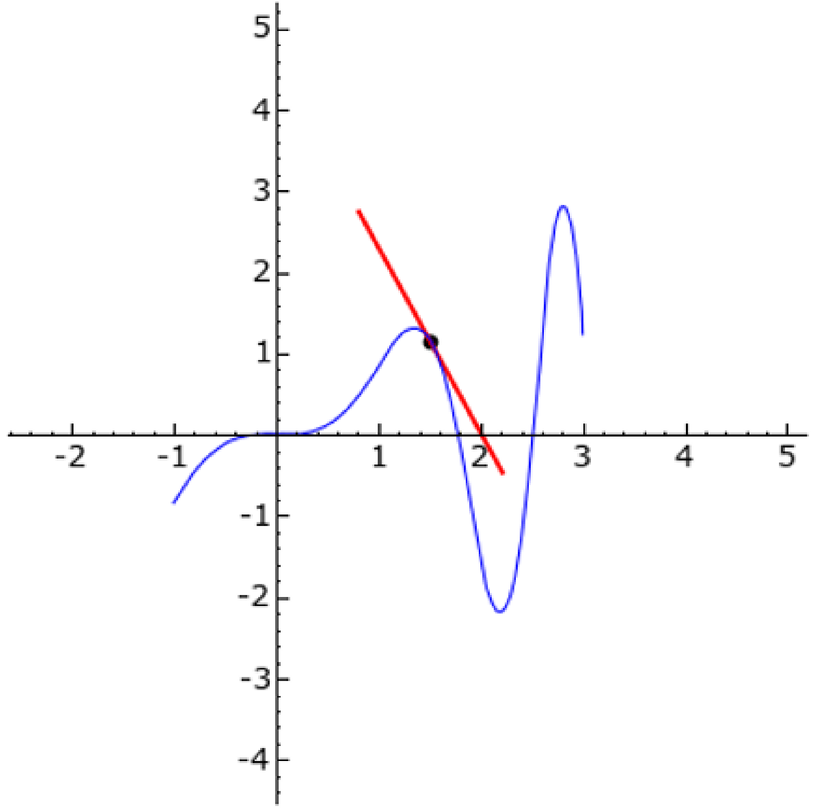
- Evaluada en un punto se corresponde con la pendiente de la recta tangente a la gráfica de la función en dicho punto.



Derivada de una función

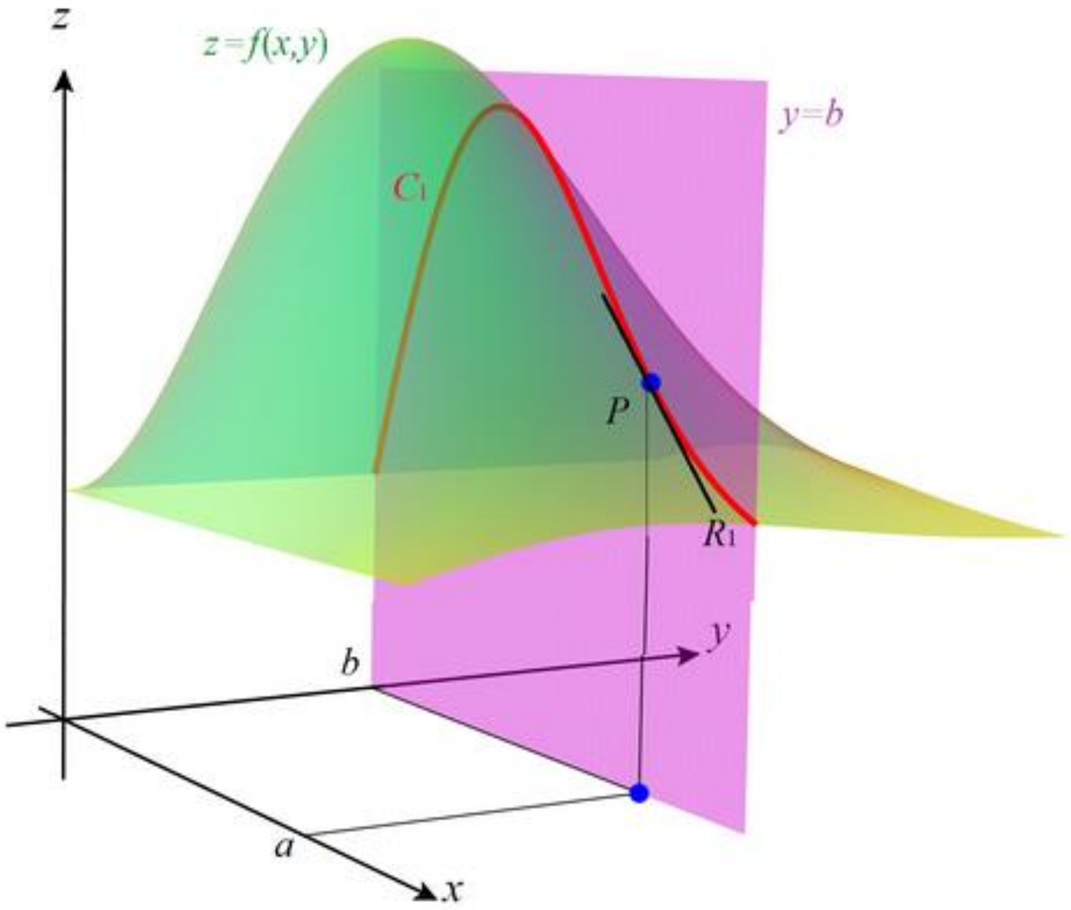
- Es una medida de la rapidez con la que cambia el valor de la función.

- Evaluada en un punto se corresponde con la pendiente de la recta tangente a la gráfica de la función en dicho punto.

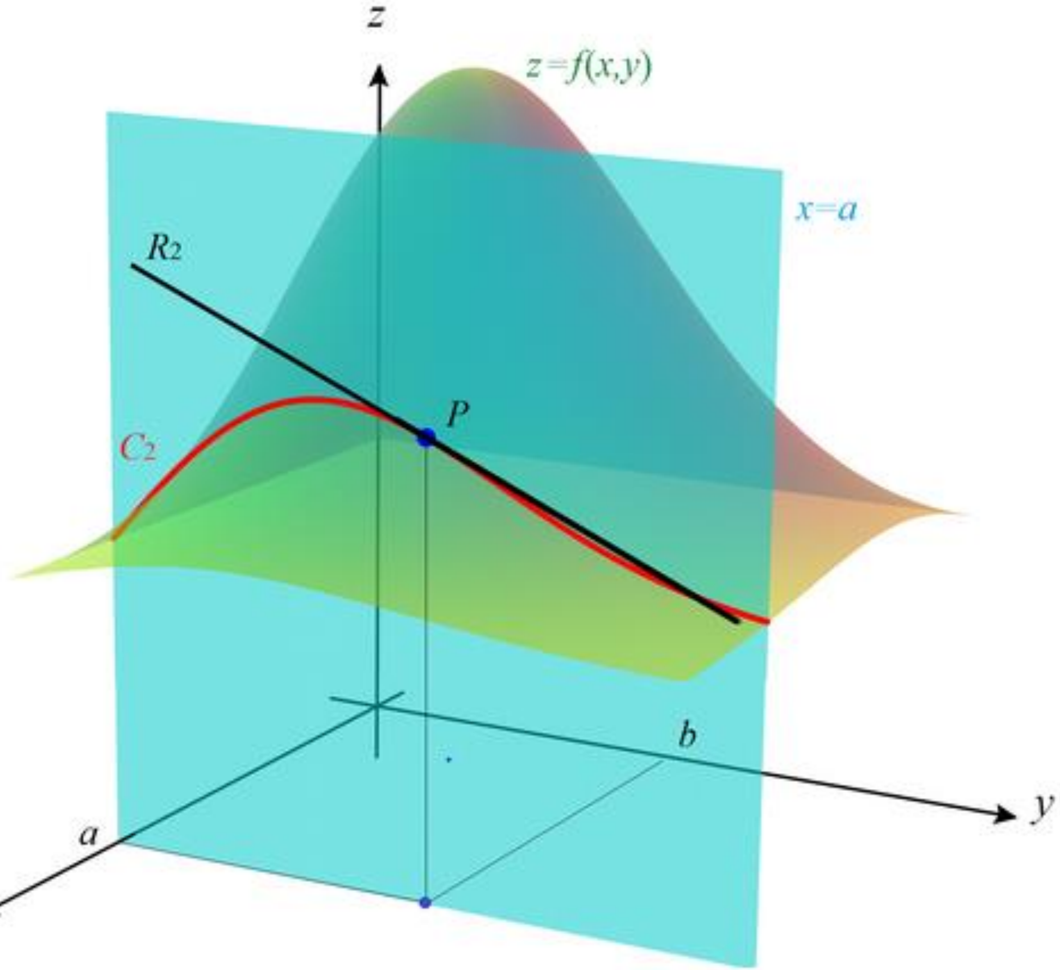


Derivadas parciales de *f(x,y)*

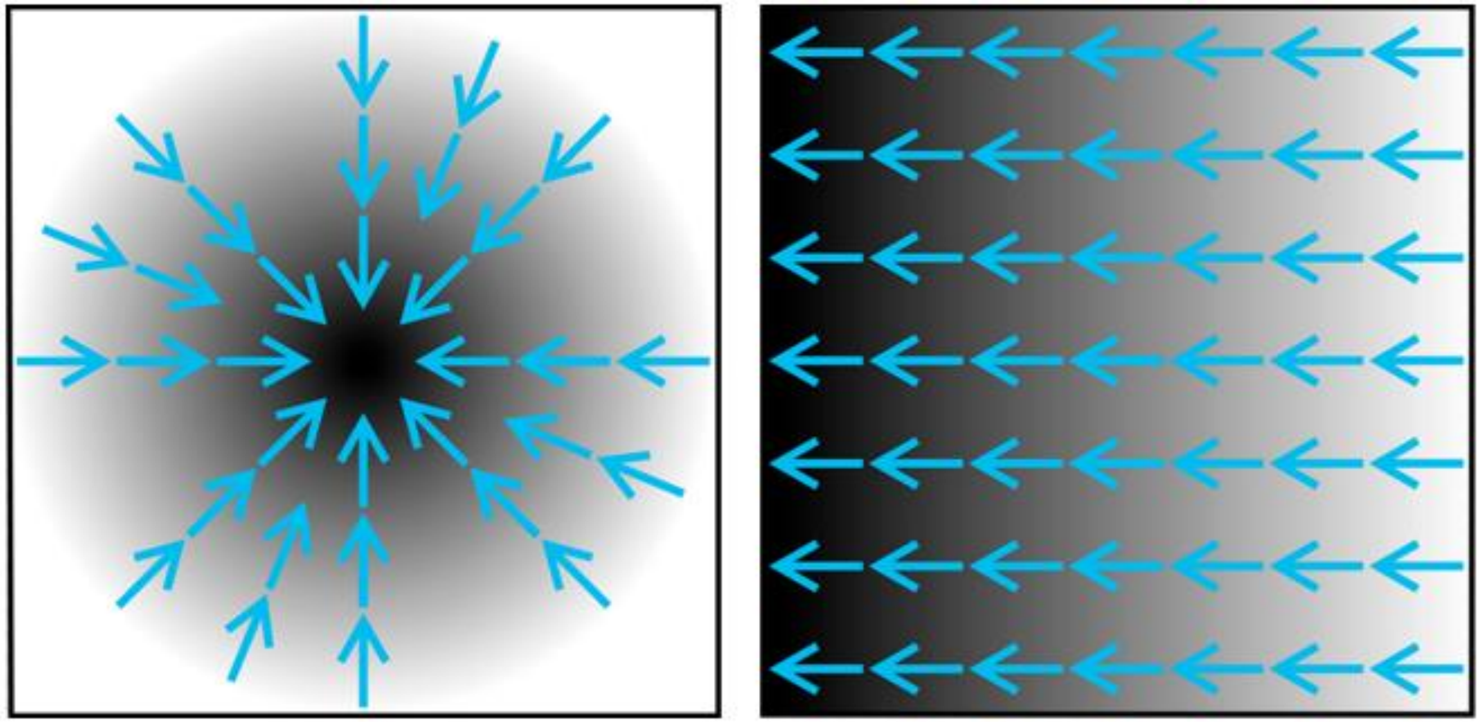
- **Con respecto a *x***



- **Con respecto a *y***



Gradiente



En las dos imágenes, los valores de la función se representan en blanco y negro. El negro representa valores más altos y su gradiente correspondiente se representa con flechas azules.

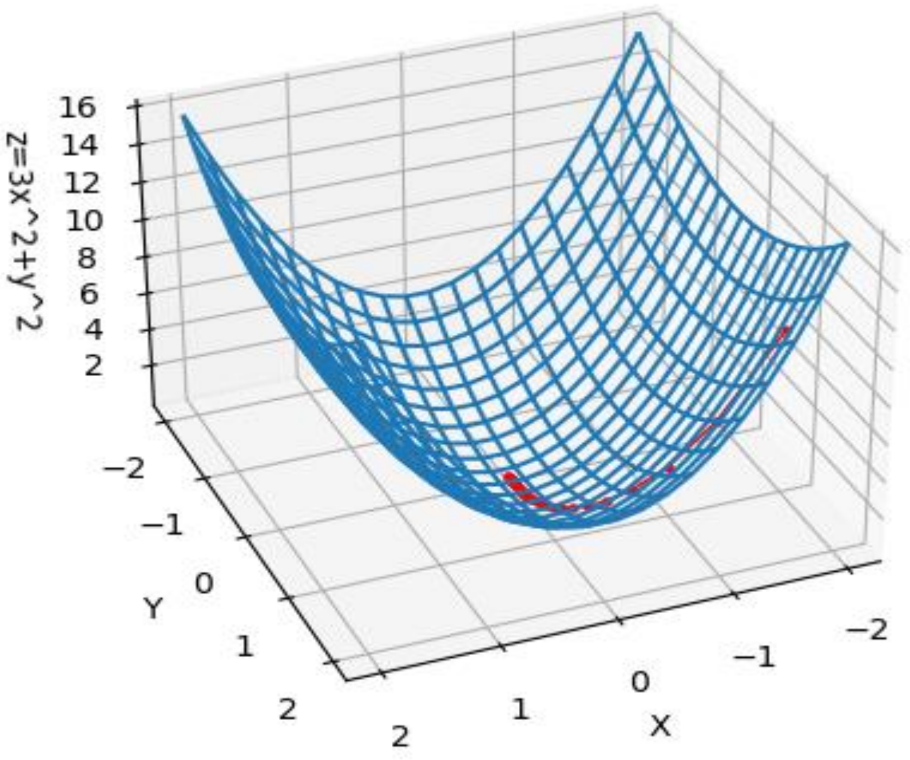
Minimización de funciones usando el gradiente

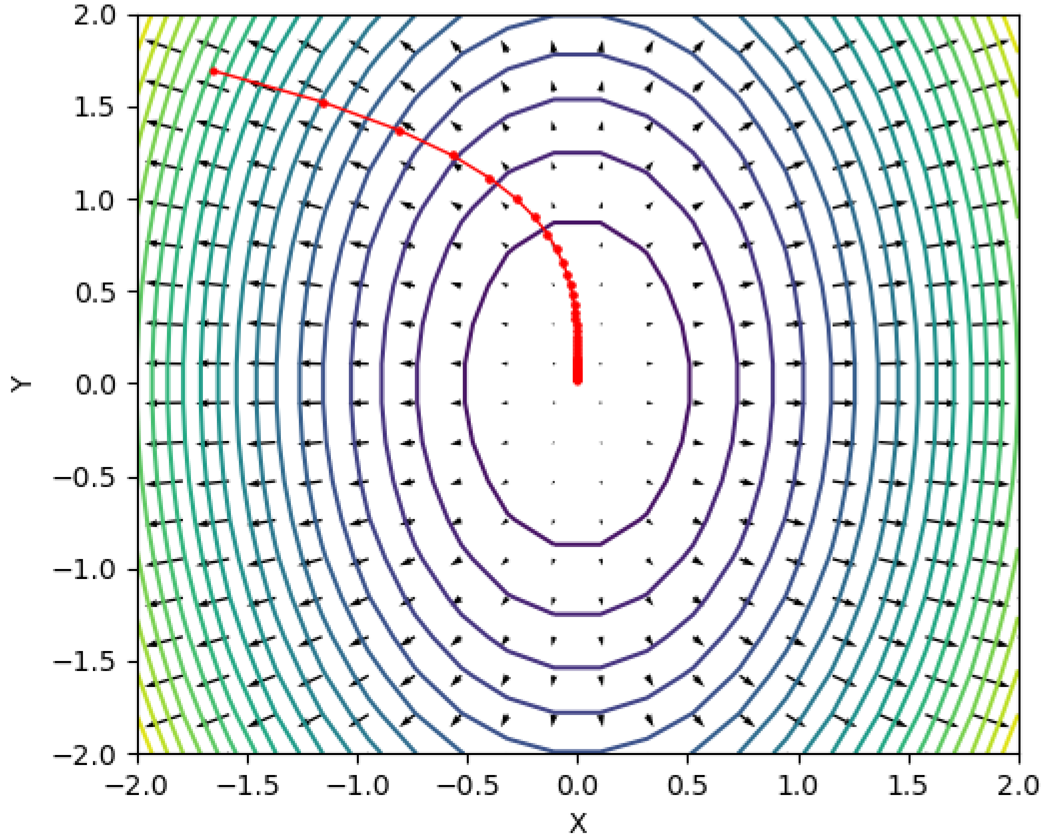
- Dada una función continua
  - Tomar un punto dentro del dominio de la función.
  - Calcular el vector gradiente de la función en ese punto.
  - Sumarle al punto anterior una fracción del gradiente negativo (para ir hacia el mínimo).
  - Repetir los dos pasos anteriores hasta que la diferencia entre evaluaciones consecutivas de la función sea inferior a una cierta cota.

Ejemplo: Minimizar **f(x,y) = 3x²+y²**

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

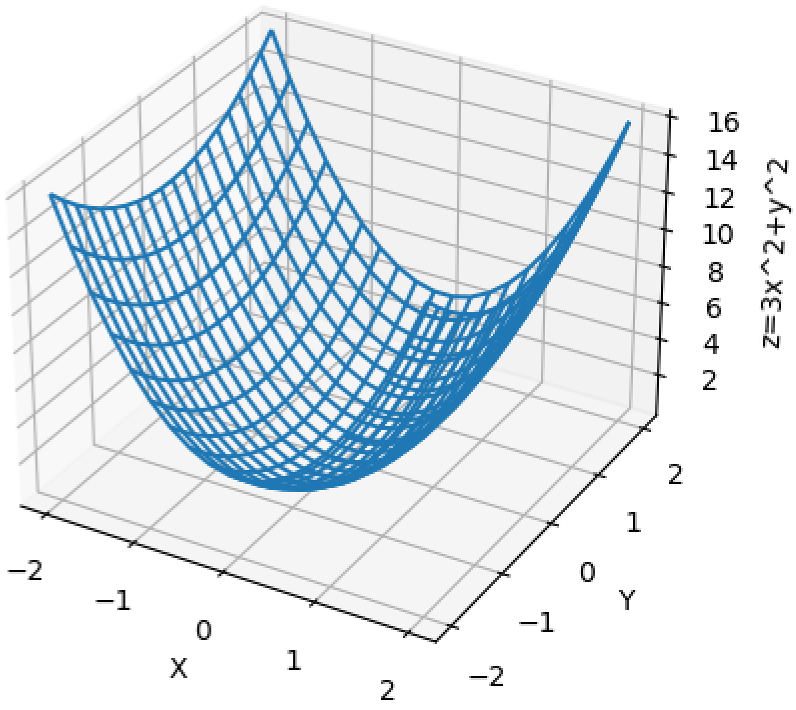
$$\nabla f = (6x, 2y)$$

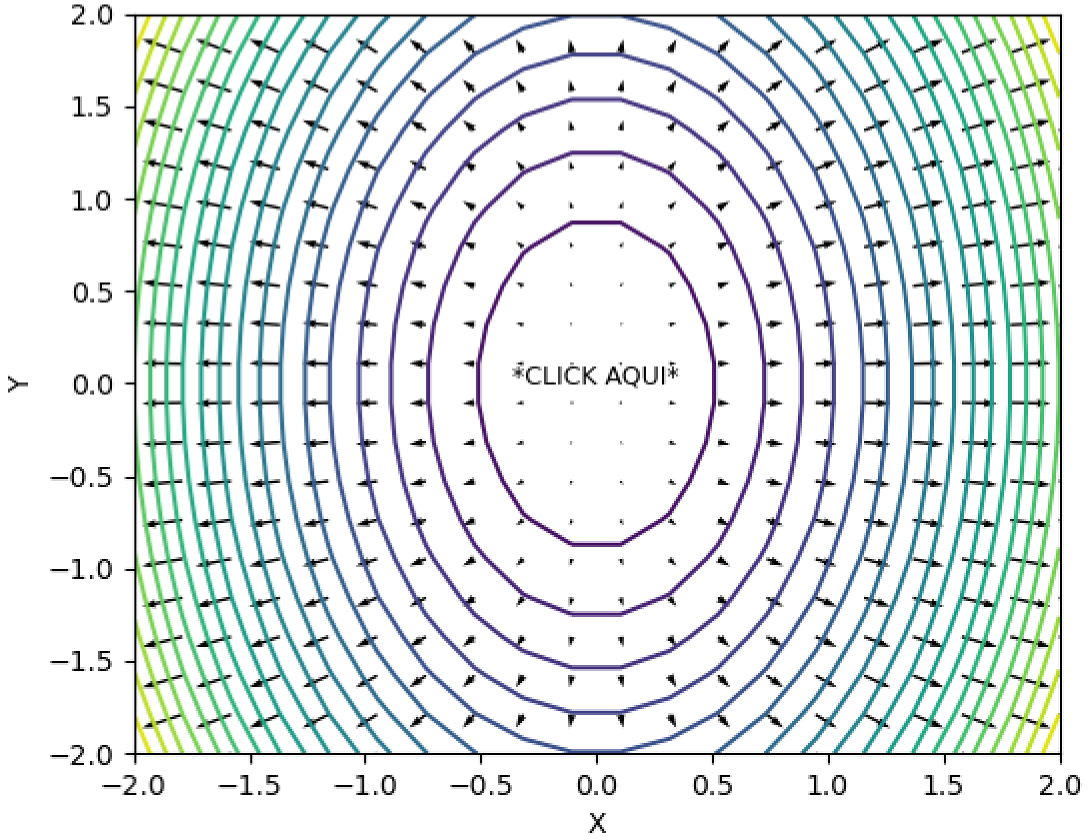




Función 1: $\mathbf{f(x,y) = 3x^2+y^2}$

- Utilice **[x, y, h] = graficoGradiente(1)** para visualizar la función y elegir el pto.inicial



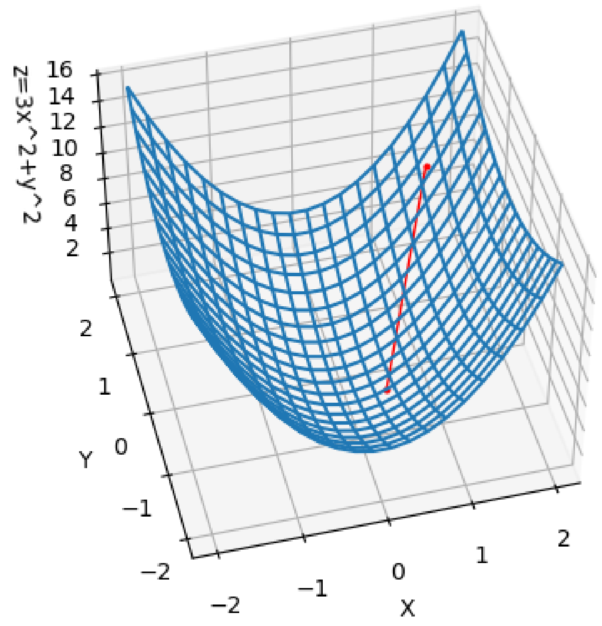


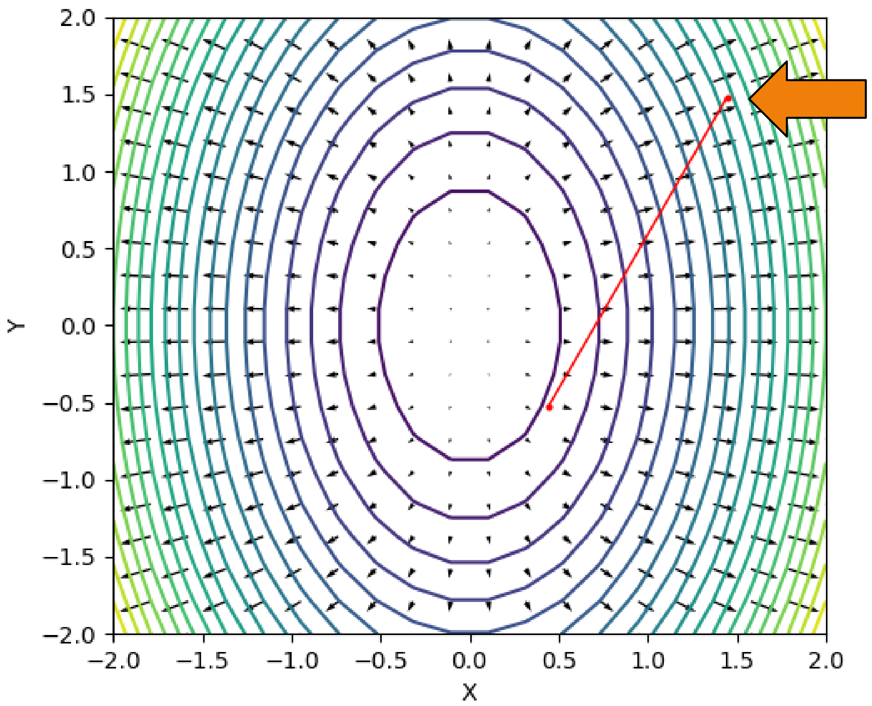
Función 1: desplazamiento en la figura

```
[x, y, h] = graficoGradiente(1)
z = 3*x**2 + y**2
PtoAnt = [x, y, z]
x = x-1
y = y-2   #--- cambiamos x e y
z = 3*x**2 + y**2;
graficarPaso(PtoAnt, [x, y, z], h)
```

gradienteFuncion_enClase.py

Función 1: $f(x,y) = 3x^2+y^2$



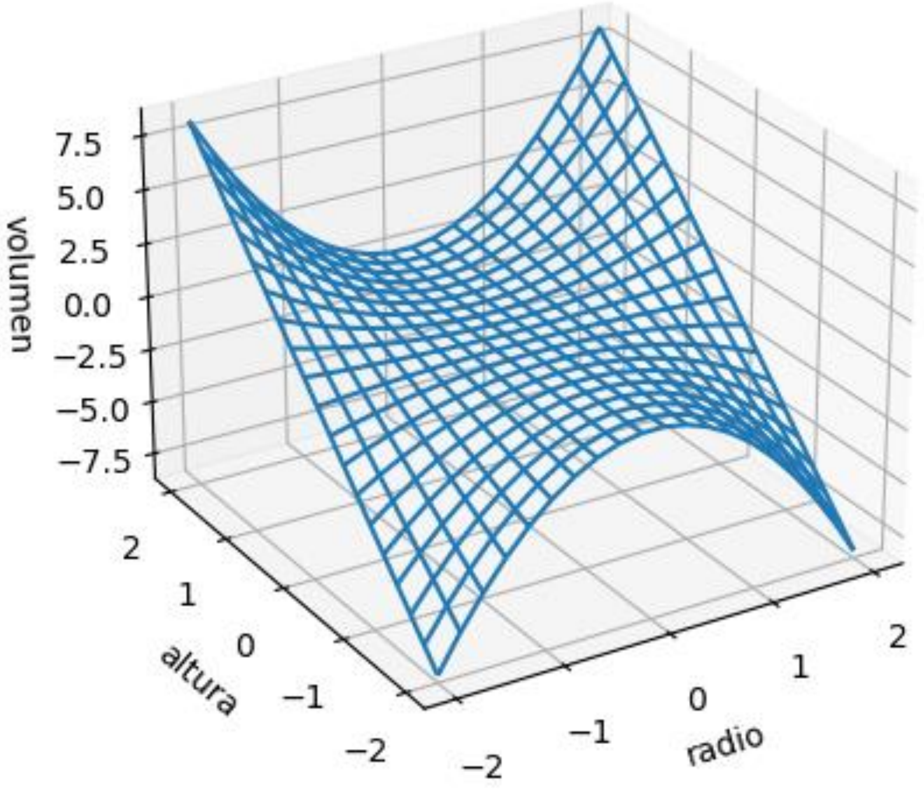


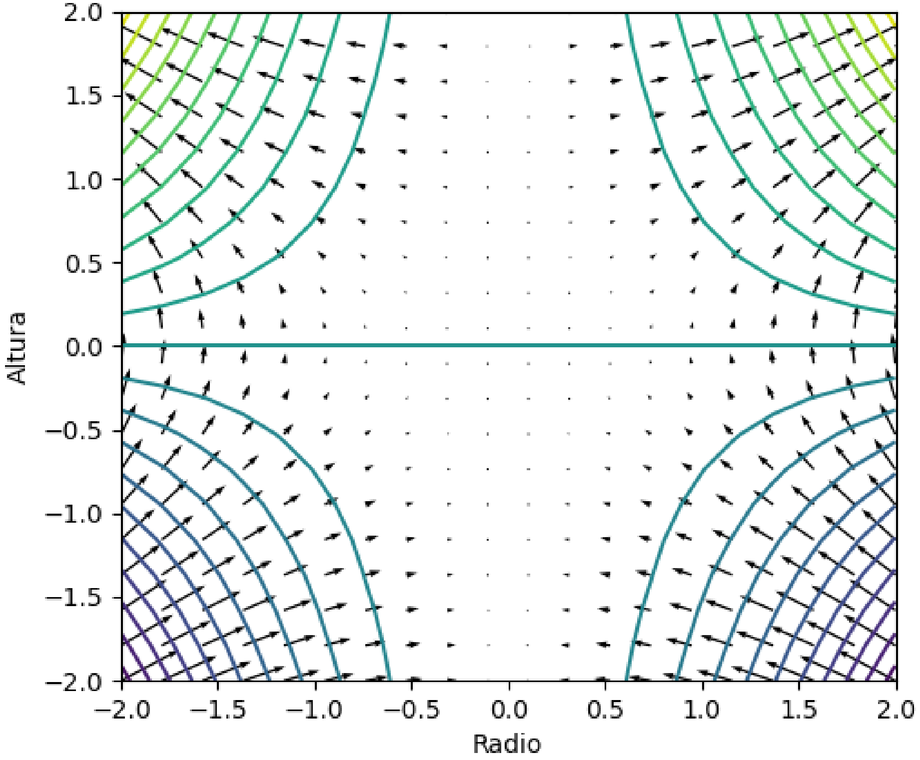
Posición inicial seleccionada

Escriba el código Python para minimizar f(x,y) usando la técnica de descenso por gradiente.

Función 2: Volumen del cono

$$V(r,h)=\frac{r^2 h\pi}{3}$$

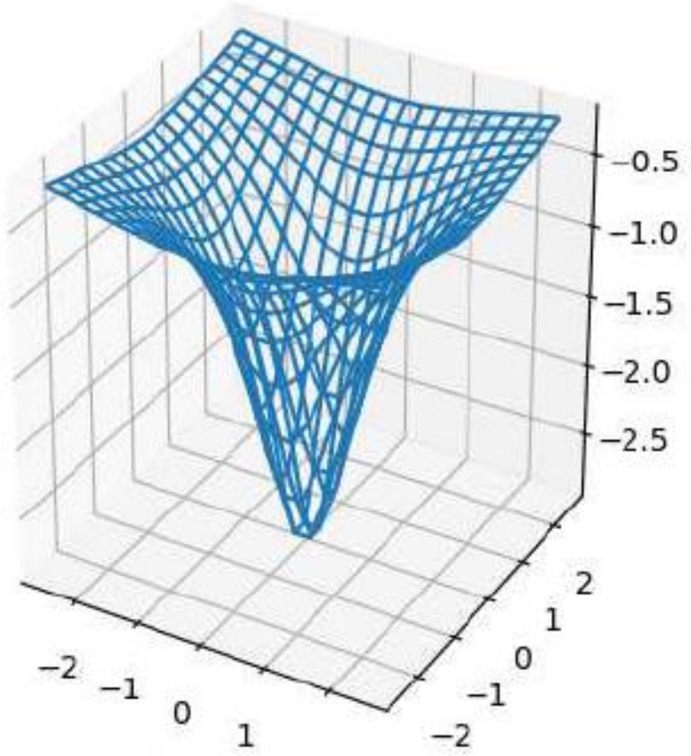




Función 3

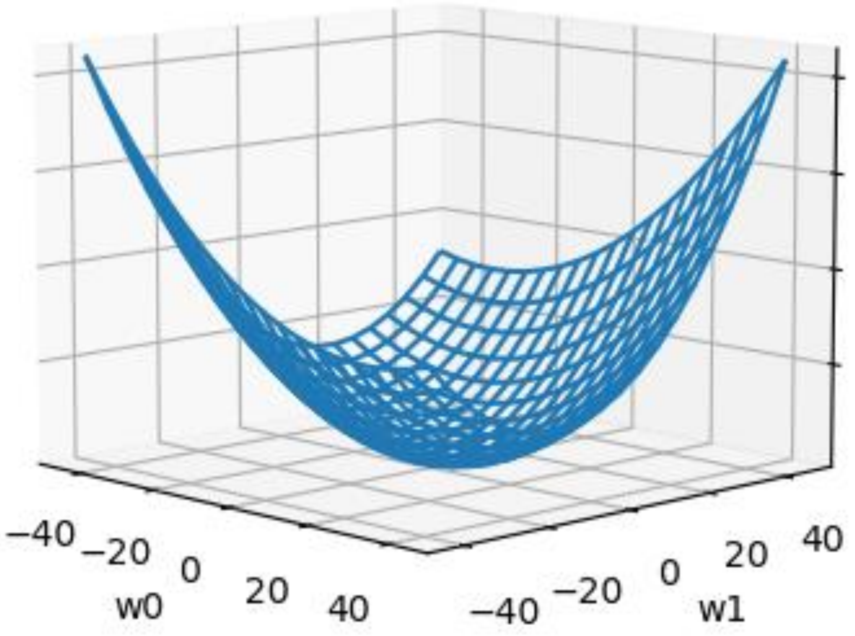
- Utilice la técnica de descenso por gradiente para hallar el mínimo de la función

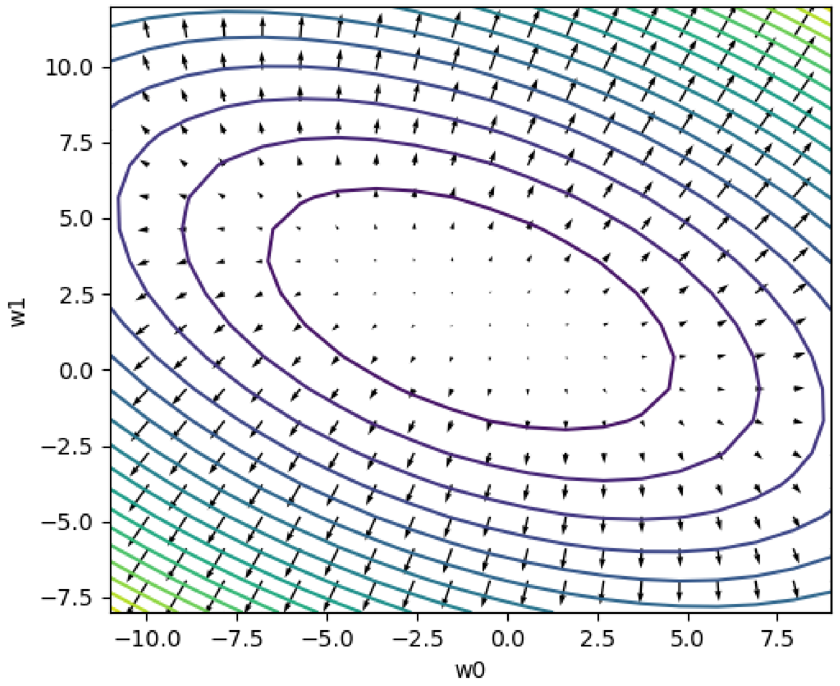
$$f(x, y) = \frac{-3}{x^2 + y^2 + 1}$$



Función 4: Error cuadrático medio

$$\xi = \frac{1}{3}\left(\left(3 - 2w_1 - w_0\right)^2 + \left(1 - w_1 - w_o\right)^2 + \left(-3 + w_1 - w_0\right)^2\right)$$

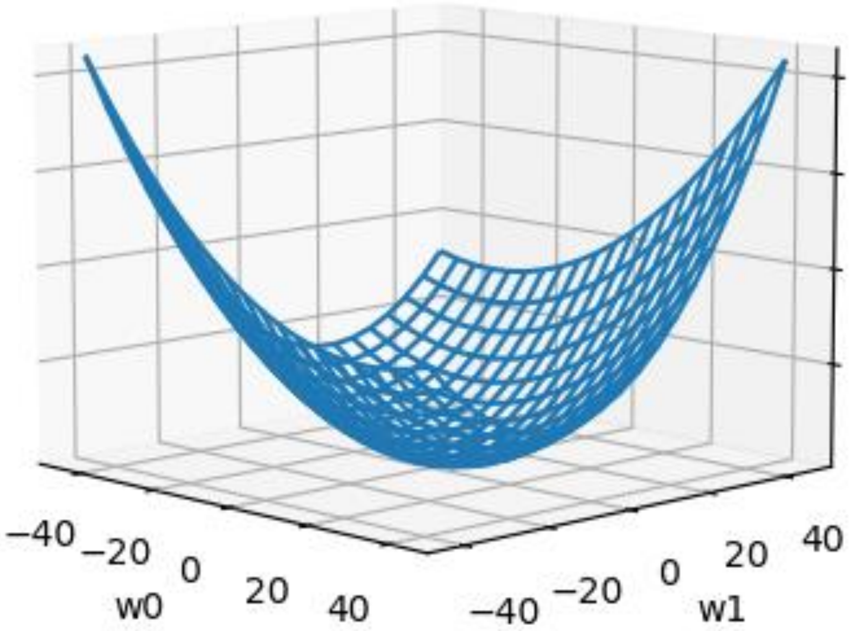




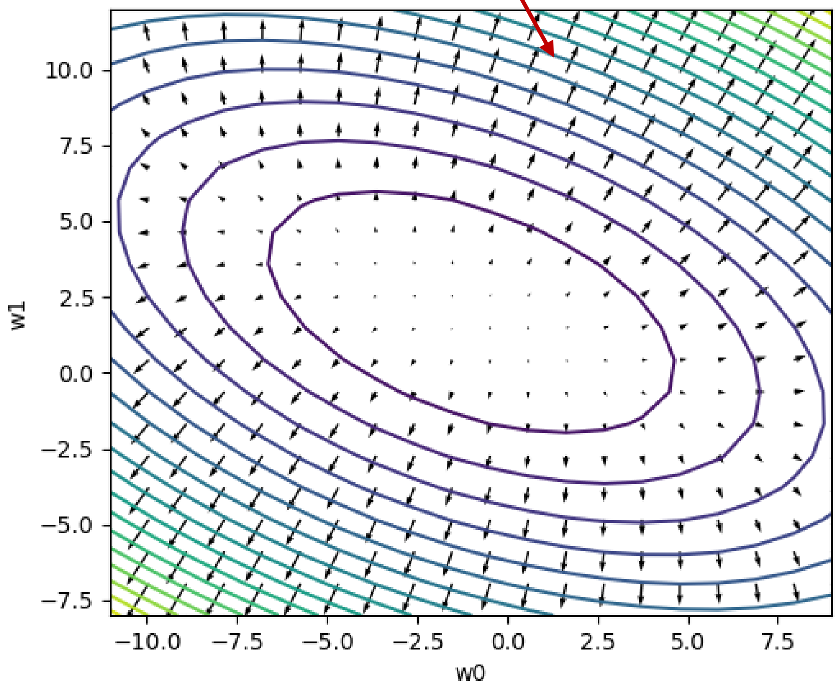
Función 4: Error cuadrático medio

$$\nabla \xi = \left(\frac{\partial \xi}{\partial w_0}, \frac{\partial \xi}{\partial w_1} \right)$$

$$\xi = \frac{1}{3}\left(19 - 20w_1 - 2w_0 + 6w_1^2 + 4w_1 w_0 + 3w_0^2\right)$$



Vector numérico que resulta de evaluar las derivadas parciales en un punto dado de la función



gradienteFuncion4_ErrorCuadratico.py

Gradiente

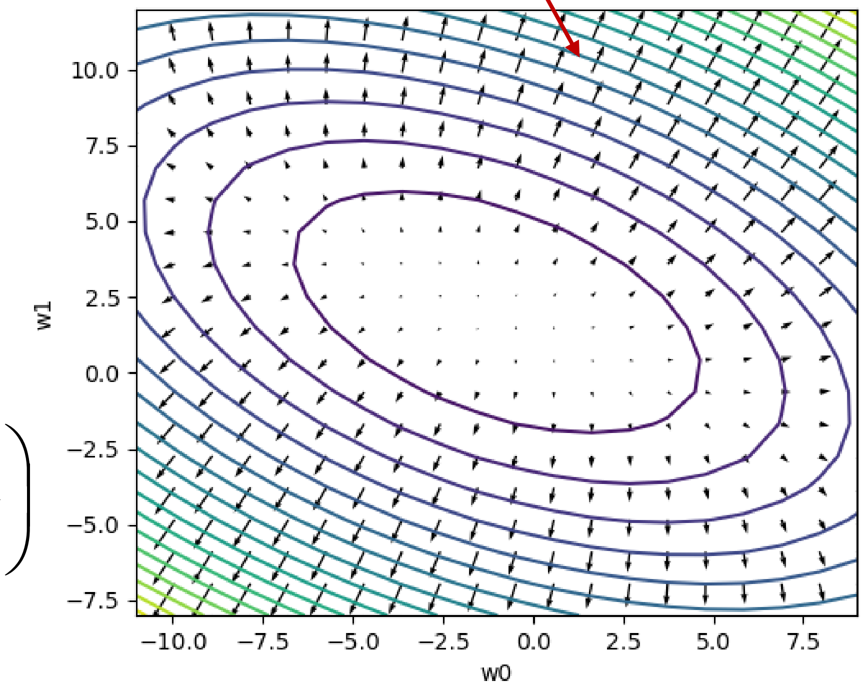
$$\xi = \frac{1}{3}\left(19 - 20w_1 - 2w_0 + 6w_1^2 + 4w_1w_0 + 3w_0^2\right)$$

$$\frac{\partial \xi}{\partial w_0} = \frac{1}{3}\left(-2 + 4w_1 + 6w_0\right)$$

$$\frac{\partial \xi}{\partial w_1} = \frac{1}{3}\left(-20 + 12w_1 + 4w_0\right)$$

$$\nabla \xi = \left(\frac{\partial \xi}{\partial w_0}, \frac{\partial \xi}{\partial w_1}\right) = \left(\frac{-2 + 4w_1 + 6w_0}{3}, \frac{-20 + 12w_1 + 4w_0}{3}\right)$$

Vector numérico que resulta de evaluar las derivadas parciales en un punto dado de la función

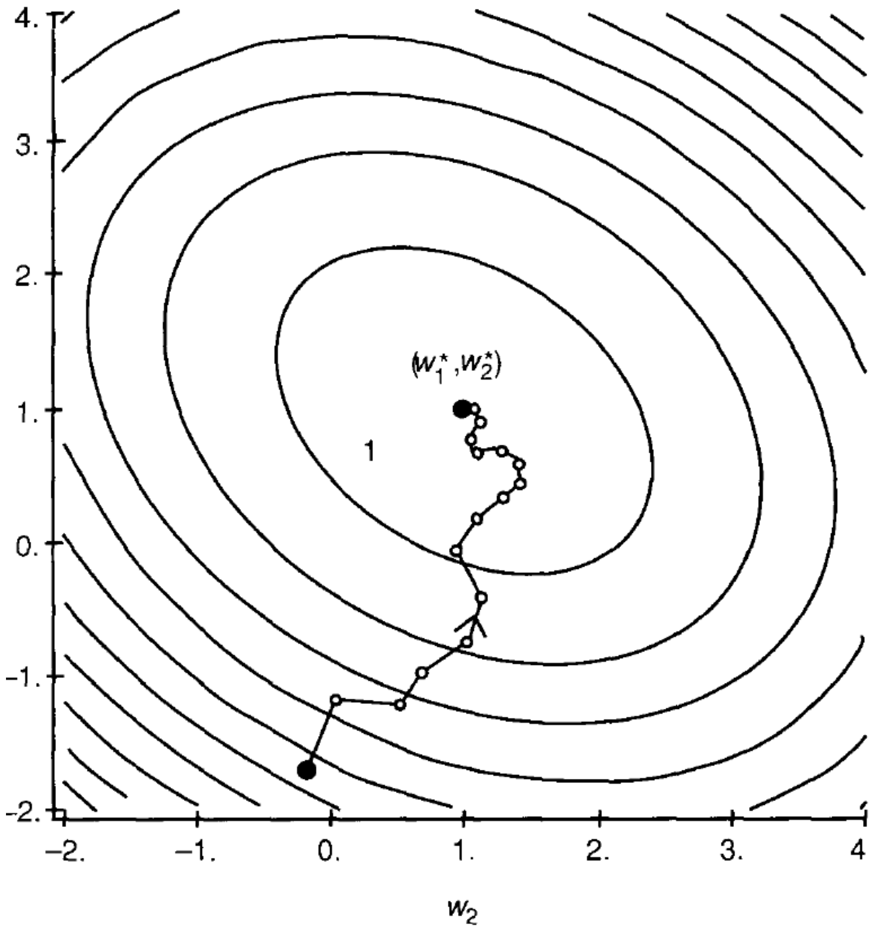


Técnica del descenso del gradiente estocástico

$$w_{t+1} = w_t - \alpha \nabla\xi(w_t)$$

- se utiliza

$$\xi =< \varepsilon_k^2 > \approx \varepsilon_k^2 = \left(d_k - \sum_{i=0}^{N} x_{ik} w_i \right)^2$$



Técnica del descenso del gradiente estocástico

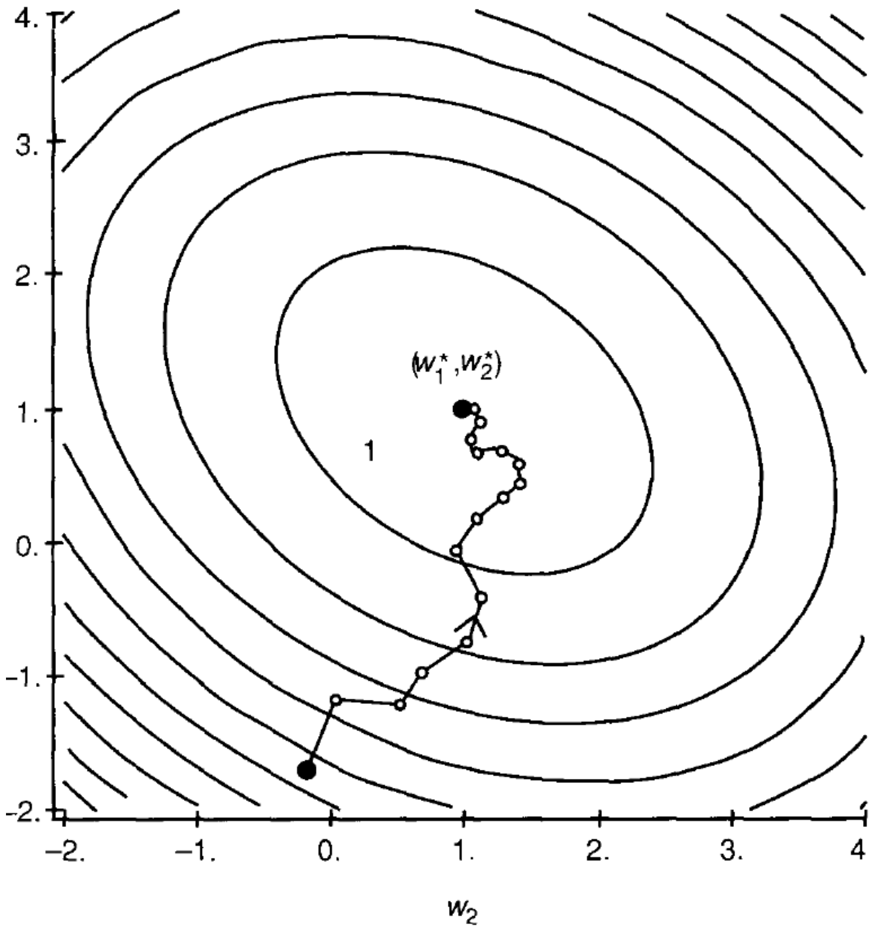
$$w_{t+1} = w_t - \alpha \nabla \xi(w_t)$$

- se utiliza

$$\xi = < \varepsilon_k^2 > \approx \varepsilon_k^2 = \left(d_k - \sum_{i=0}^{N} x_{ik} w_i \right)^2$$

- Veamos que

$$\nabla \varepsilon_k^2 = -2\, \varepsilon_k\, x_k$$



Gradiente del error en el ejemplo x_k

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = \left[\frac{\partial (d_k - y_k)^2}{\partial w_0}; \quad \ldots; \quad \frac{\partial (d_k - y_k)^2}{\partial w_n} \right]$$

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = \left[-2(d_k - y_k)\frac{\partial y_k}{\partial w_0}; \quad \ldots; \quad -2(d_k - y_k)\frac{\partial y_k}{\partial w_n} \right]$$

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = -2e_k \left[\frac{\partial y_k}{\partial w_0}; \quad \ldots; \quad \frac{\partial y_k}{\partial w_n} \right]$$

$$\nabla \varepsilon_k^2(t) = \frac{\partial \varepsilon_k^2}{\partial w} = -2e_k \left[x_{0k}, \quad x_{1k}, \quad \ldots, \quad x_{nk} \right] = -2e_k x_k$$

Entrenamiento del combinador lineal

- Para cada vector de entrada
  - Aplicar el vector de entrada, X_k
  - Calcular el gradiente utilizando

$$\nabla < \varepsilon_k^2 > \approx \nabla \varepsilon_k^2 = -2\varepsilon_k \, x_k = -2(d_k - y_k)x_k$$

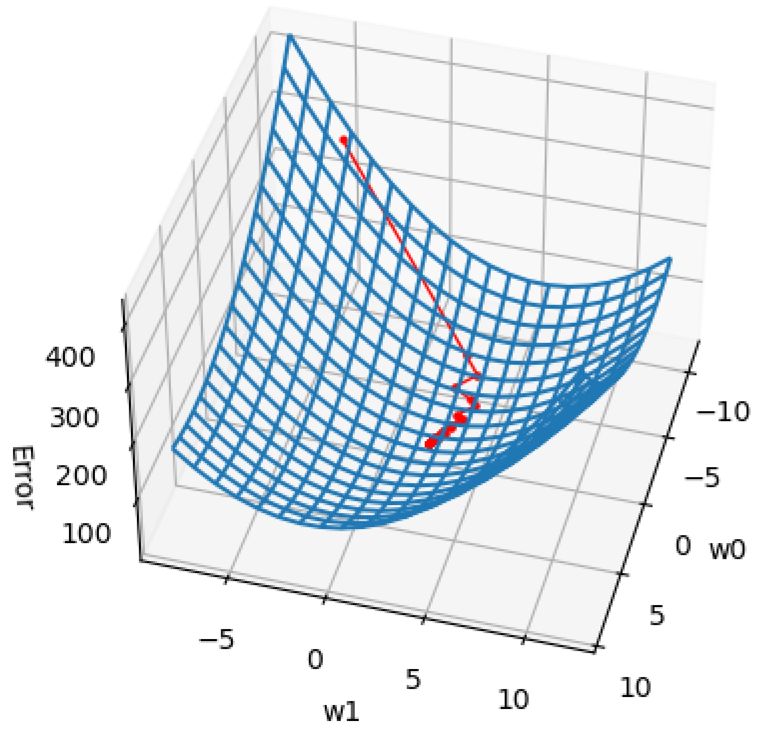
  - Actualizar el vector de pesos

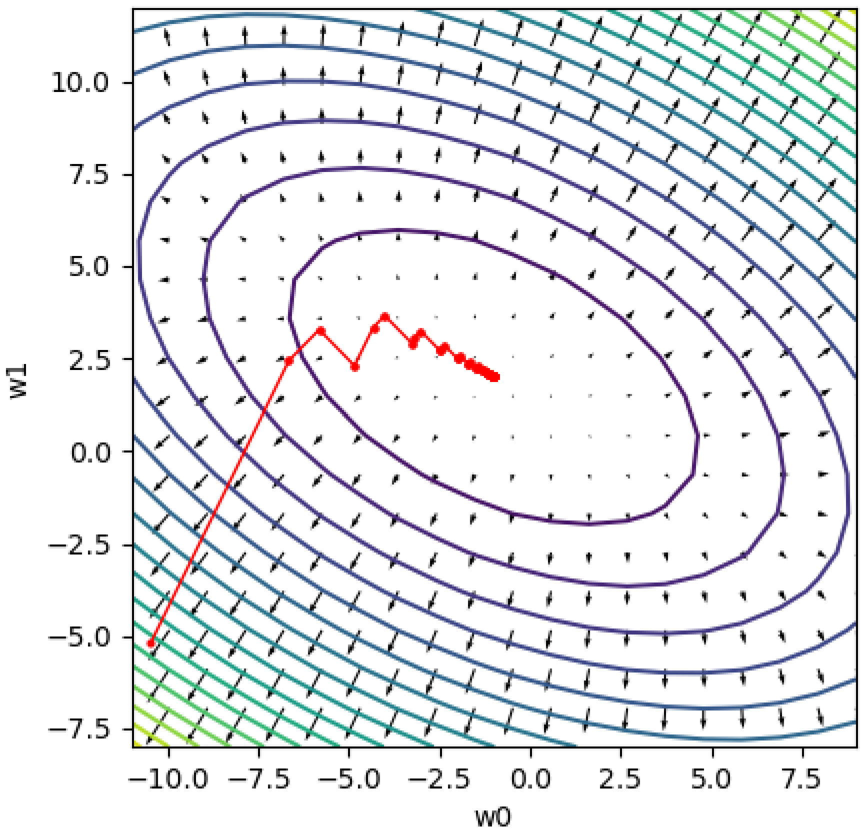
$$w_{t+1} = w_t + 2\,\alpha(d_k - y_k)x_k$$

- Repetir todo hasta que el error sea aceptable

Minimización de la función de error usando el descenso de gradiente

gradienteFuncion4_CombinadorLineal.py





Ejecutar con distintos valores de alfa (velocidad de aprendizaje)

```
import numpy as np

X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]

[w0, w1] = np.random.uniform(-50, 50,size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
    print ("ite= %d   w0= %8.5f   w1=%8.5f   E=%.8f" % (ite,w0,w1,E))
```

Los pesos iniciales son aleatorios

```python
import numpy as np

X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]

[w0, w1] = np.random.uniform(-50, 50,size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06
```

Parámetros del entrenamiento

```python
ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
    print ("ite= %d   w0= %8.5f   w1=%8.5f   E=%.8f" % (ite,w0,w1,E))
```

```python
import numpy as np

X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]

[w0, w1] = np.random.uniform(-50, 50,size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
    print ("ite= %d   w0= %8.5f   w1=%8.5f   E=%.8f" % (ite,w0,w1,E))
```

Termina o bien porque realizó la máxima cantidad de intentos o porque el valor absoluto de la diferencia entre dos valores consecutivos de la función es inferior a cierta cota

```python
import numpy as np

X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]

[w0, w1] = np.random.uniform(-50, 50,size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
    print ("ite= %d   w0= %8.5f   w1=%8.5f   E=%.8f" % (ite,w0,w1,E))
```

Calculamos la salida del combinador lineal y evaluamos el error cometido

```python
import numpy as np

X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]

[w0, w1] = np.random.uniform(-50, 50,size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
    print ("ite= %d   w0= %8.5f   w1=%8.5f   E=%.8f" % (ite,w0,w1,E))
```

Calculamos el vector gradiente

```
import numpy as np

X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]

[w0, w1] = np.random.uniform(-50, 50,size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
    print ("ite= %d   w0= %8.5f   w1=%8.5f   E=%.8f" % (ite,w0,w1,E))
```

Actualizamos los pesos en la dirección del gradiente negativo

```python
import numpy as np

X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]

[w0, w1] = np.random.uniform(-50, 50,size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
    print ("ite= %d   w0= %8.5f   w1=%8.5f   E=%.8f" % (ite,w0,w1,E))
```

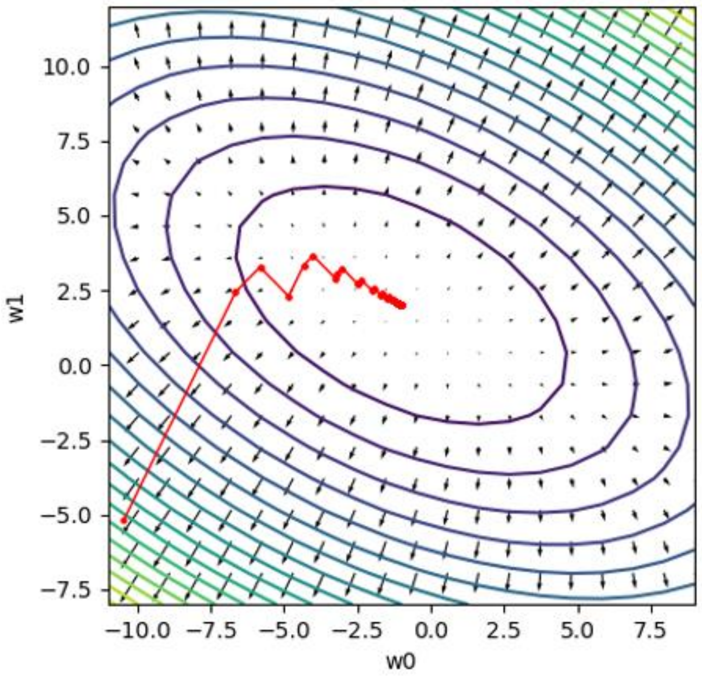
Acumulamos el cuadrado de los errores cometidos

```python
import numpy as np

X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]

[w0, w1] = np.random.uniform(-50, 50,size=2)

alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06

ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0

    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida

        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]

        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1

        sumaError = sumaError + Error**2

    E = sumaError / len(X)
    ite = ite + 1
    print ("ite= %d   w0= %8.5f   w1=%8.5f   E=%.8f" % (ite,w0,w1,E))
```

Dividimos por la cantidad de ejemplos para obtener el ECM

```python
import numpy as np
X = [2, 1, -1] # (2,3), (1,1), (-1,-3)
Y = [3, 1, -3]
[w0, w1] = np.random.uniform(-50, 50,size=2)
alfa = 0.1
MAX_ITE = 5000
COTA = 10e-06
ite = 0
E_ant = 0
E = 1
while ((ite<MAX_ITE) and (np.abs(E_ant - E) > COTA)):
    E_ant=E
    sumaError = 0
    for p in range(len(X)):
        salida = w1 * X[p] + w0
        Error = Y[p]-salida
        grad_w0 = -2*Error
        grad_w1 = -2*Error*X[p]
        w0 = w0 - alfa * grad_w0
        w1 = w1 - alfa * grad_w1
        sumaError = sumaError + Error**2
    E = sumaError / len(X)
    ite = ite + 1
    print ("ite= %d   w0= %8.5f   w1=%8.5f   E=%.8f" % (ite,w0,w1,E))
```

gradienteFuncion4_CombinadorLineal.py



Note que el vector W se actualiza con cada ejemplo que ingresa a la red

ClassNeuronaLineal.py

```
nl = NeuronaLineal(alpha=0.01, n_iter=50, cotaE=10E-07,
                   random_state=None, draw=0, title=['X1','X2'])
```

- Parámetros de entrada
  - **alpha**: valor en el intervalo (0, 1] que representa la velocidad de aprendizaje.
  - **n_iter**: máxima cantidad de iteraciones a realizar.
  - **cotaE**: termina si la diferencia entre dos errores consecutivos es menor que este valor.
  - **random_state**: None si los pesos se inicializan en forma aleatoria, un valor entero para fijar la semilla
  - **draw**: valor distinto de 0 si se desea ver el gráfico y 0 si no. Sólo si es 2D.
  - **title**: lista con los nombres de los ejes para el gráfico. Se usa sólo si **draw** no es cero.

ClassNeuronaLineal.py

nl = NeuronaLineal(alpha=0.01, n_iter=50)

nl.fit(X, T)

- Parámetros de entrada
  - **X** : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.
  - **T** : arreglo de N elementos siendo N la cantidad de ejemplos
- Retorna
  - **w_** : arreglo de M elementos siendo M la cantidad de atributos de entrada
  - **b_** : valor numérico continuo correspondiente al bias.
  - **errors_**: errores cometidos en cada iteración.

ClassNeuronaLineal.py

Y = nl.predict(X)

- Parámetros de entrada
  - **X** : arreglo de NxM donde N es la cantidad de ejemplos y M la cantidad de atributos.

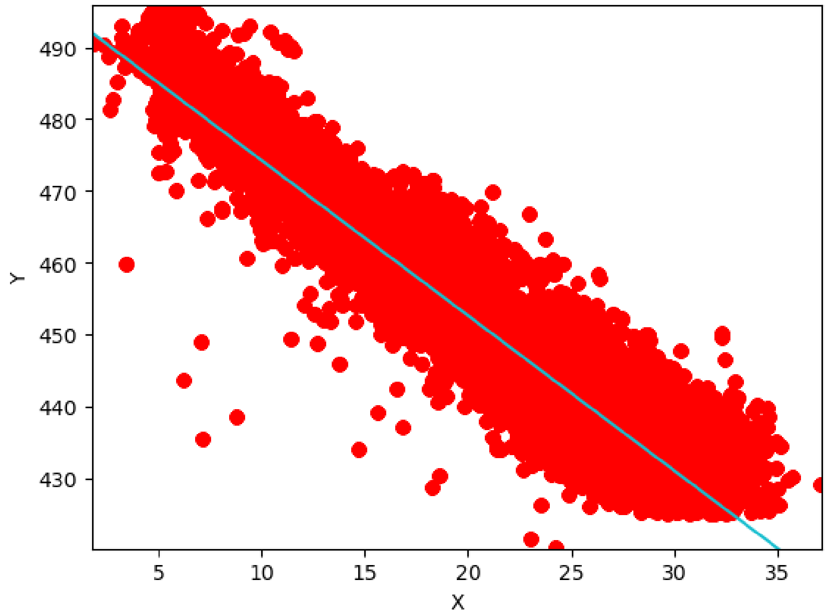
- Retorna: un arreglo con el resultado de aplicar el combinador lineal entrenado previamente con fit() a la matriz de ejemplos X.
  - **Y** : arreglo de N elementos siendo N la cantidad de ejemplos

CombLineal_PtosNOalineados.ipynb

```
import numpy as np
from ClassNeuronaLineal import NeuronaLineal

Ptos = np.array([[1,1], [3,2], [5,5], [2,3], [4,3]])
X = Ptos[:,0].reshape(-1,1)
Y = Ptos[:,1]

alfa = 0.01
MAX_ITE = 500
Cota = 10e-06

nl = NeuronaLineal(alpha=0.01, n_iter=30, cotaE=10e-06, draw=1, title=['X','Y'])
nl.fit(X, Y)

#-- gráfico con la evolución del error ---
plt.plot(range(1, len(nl.errors_) + 1), nl.errors_, marker='o')
plt.xlabel('Iteraciones')
plt.ylabel('Cantidad de actualizaciones')
plt.show()
```

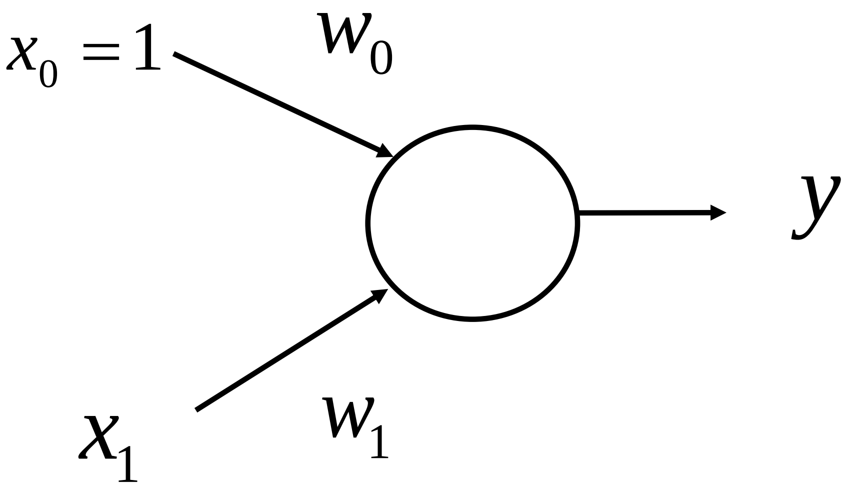
Comb_LINEAL_CCPP_RN.ipynb

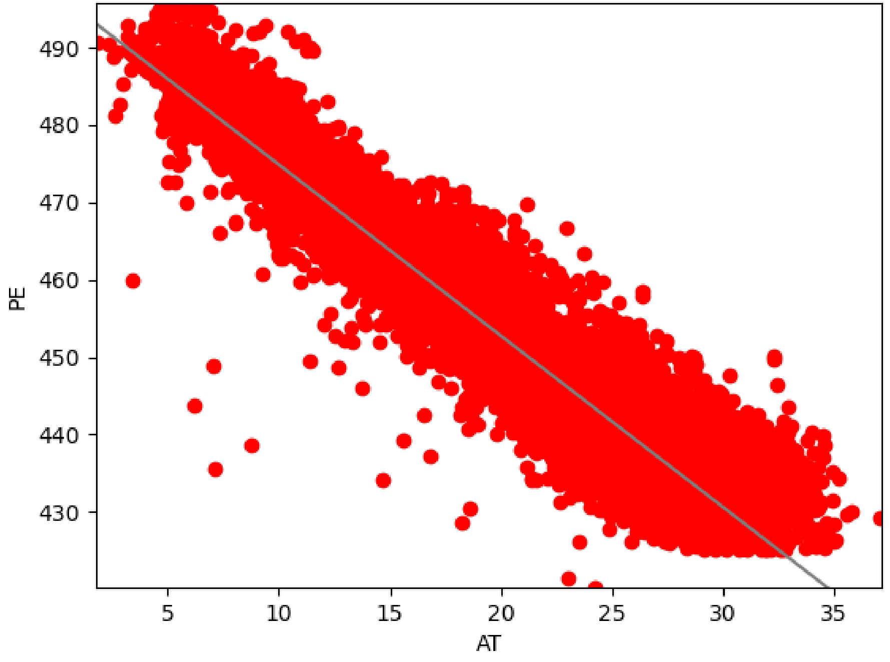
```
import numpy as np
from ClassNeuronaLineal import NeuronaLineal

datos= pd.read_csv(DATOS_DIR+'CCPP.csv')

T = np.array(datos['PE'])   # vector fila
X = np.array(datos['AT'])
X = X.reshape(-1,1)         # vector columna

alfa = 0.01
MAX_ITE = 500
Cota = 10e-06

nl = NeuronaLineal(alpha=0.0001, n_iter=50, cotaE=10e-06, draw=1,title=['X','Y'])
nl.fit(X, T)
```



Producción de energía (CCPP.csv)

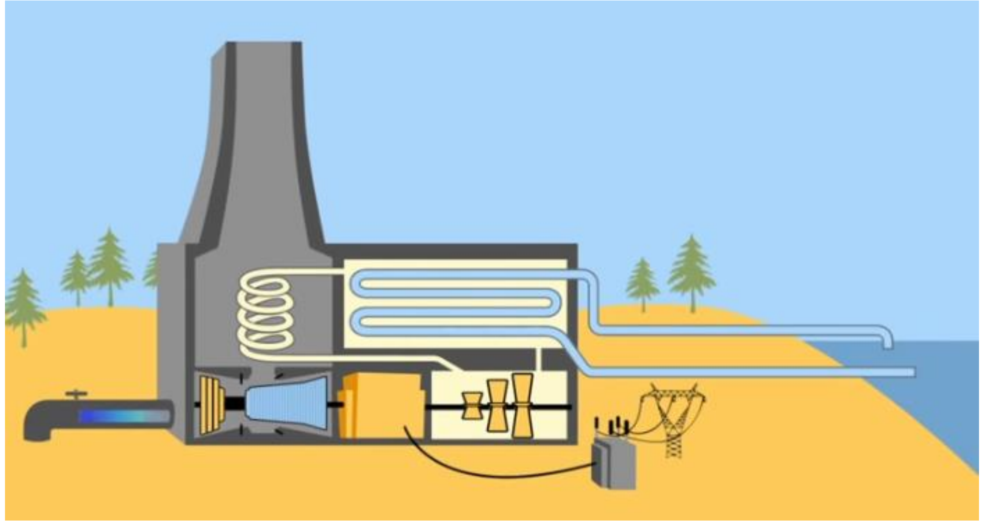
- Modifique el código anterior para predecir la producción de energía (atributo PE) a partir del valor de temperatura (atributo AT)





Ejemplo - Central de energía eléctrica

- El archivo **CCPP.csv** contiene 9568 datos de una Central de Ciclo Combinado recolectados entre 2006 y 2011.
- Objetivo: Predecir la producción neta de energía eléctrica por hora (PE) de la planta
- Las características relevadas son:
  - Temperatura ambiente (AT)
  - Presión ambiente (AP)
  - Humedad relativa (RH)
  - Vacío de escape (V)



(ver cómo funciona la central)

Ejemplo - Central de energía eléctrica

- Matriz de Correlación

Index	AT	V	AP	RH	PE
AT	1	0.844107	-0.507549	-0.542535	-0.948128
V	0.844107	1	-0.413502	-0.312187	-0.86978
AP	-0.507549	-0.413502	1	0.0995743	0.518429
RH	-0.542535	-0.312187	0.0995743	1	0.389794
PE	-0.948128	-0.86978	0.518429	0.389794	1

Predicción de la Producción de Energía (PE)

X

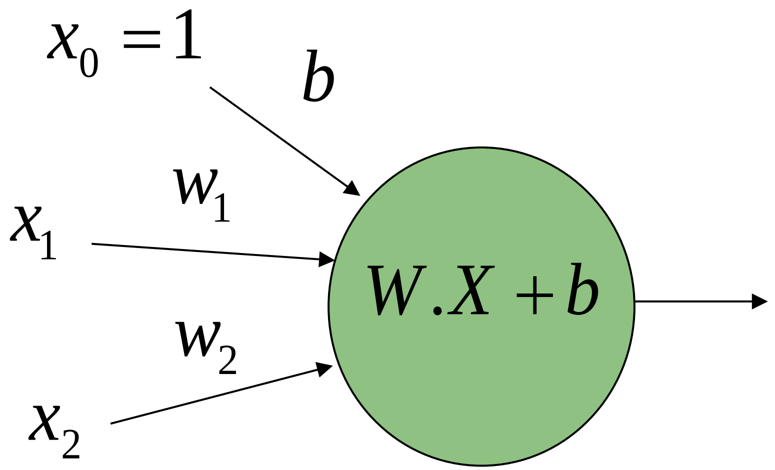
```
[[14.96, 41.76],
 [25.18, 62.96],
 [ 5.11, 39.4 ],
 ...,
 [31.32, 74.33],
 [24.48, 69.45],
 [21.6 , 62.52]]
```

AT **V**

Y

```
[463.26,
 444.37,
 488.56,
    ...,
 429.57,
 435.74,
 453.28]
```

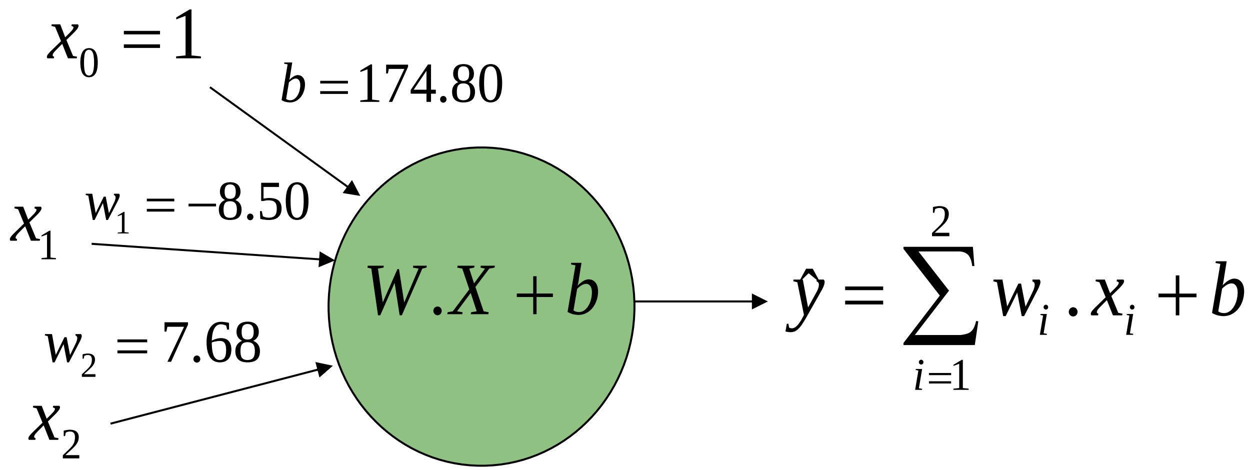
PE



$$\hat{y} = \sum_{i=1}^{2} w_i . x_i + b$$

Predicción de la Producción de Energía (PE)

```
X                              Y
[[14.96, 41.76],               [463.26,
 [25.18, 62.96],                444.37,
 [ 5.11, 39.4 ],                488.56,
 ...,                           ...,
 [31.32, 74.33],                429.57,
 [24.48, 69.45],                435.74,
 [21.6 , 62.52]]                453.28]
```



$$\underbrace{(-8.50 \quad 7.68)}_{\textbf{W}} * \begin{pmatrix} 14.96 \\ 41.73 \end{pmatrix} + \underbrace{174.80}_{\textbf{b}}$$

$$\hat{y} = 368.49$$

$$y = 463.26$$

Error cometido en este ejemplo $\longrightarrow$ $Error = (y - \hat{y}) = 94.76$

Descenso de gradiente estocástico

- Se busca minimizar el ECM para el ejemplo actual

$$E = (y - \hat{y})^2 = \left(y - (w_1.x_1 + w_2.x_2 + b)\right)^2$$

- Debemos calcular el gradiente para saber cómo modificar los pesos

$$\frac{\partial E}{\partial w_1} = -2(y - \hat{y})\frac{\partial(w_1.x_1 + w_2.x_2 + b)}{\partial w_1} = -2(y - \hat{y})x_1$$

$$\frac{\partial E}{\partial w_2} = -2(y - \hat{y})\frac{\partial(w_1.x_1 + w_2.x_2 + b)}{\partial w_2} = -2(y - \hat{y})x_2$$

$$\frac{\partial E}{\partial b} = -2(y - \hat{y})\frac{\partial(w_1.x_1 + w_2.x_2 + b)}{\partial b} = -2(y - \hat{y})$$

- La forma gral. es

$$\boldsymbol{\frac{\partial E}{w_i} = -2(y - \hat{y})x_i}$$

Descenso de gradiente estocástico

- Se busca minimizar el ECM para el ejemplo actual

$$E \ = \ (y - \hat{y})^2 = \left(y - (w_1 . x_1 + w_2 . x_2 + b)\right)^2$$

- Debemos calcular el gradiente para saber cómo modificar los pesos
- Luego

- La forma gral. es

$$\frac{\partial \boldsymbol{E}}{\boldsymbol{w_i}} = -\boldsymbol{2}(\boldsymbol{y} - \widehat{\boldsymbol{y}})\boldsymbol{x_i}$$

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w_1}$$

$$w_1 = w_1 + 2\alpha . (y - \hat{y}) . x_1 = -8.50 + 2\alpha * 94.76 * 14.96 = -8.47$$

Descenso de gradiente estocástico

- Se busca minimizar el ECM para el ejemplo actual

$$E = (y - \hat{y})^2 = \left(y - (w_1 . x_1 + w_2 . x_2 + b)\right)^2$$

- Debemos calcular el gradiente para saber cómo modificar los pesos
- Luego

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w_1} \qquad w_2 = w_2 - \alpha \frac{\partial E}{\partial w_2} \qquad b = b - \alpha \frac{\partial E}{\partial b}$$

$$w_1 = -8.47 \qquad w_2 = 7.76 \qquad b = 174.80$$

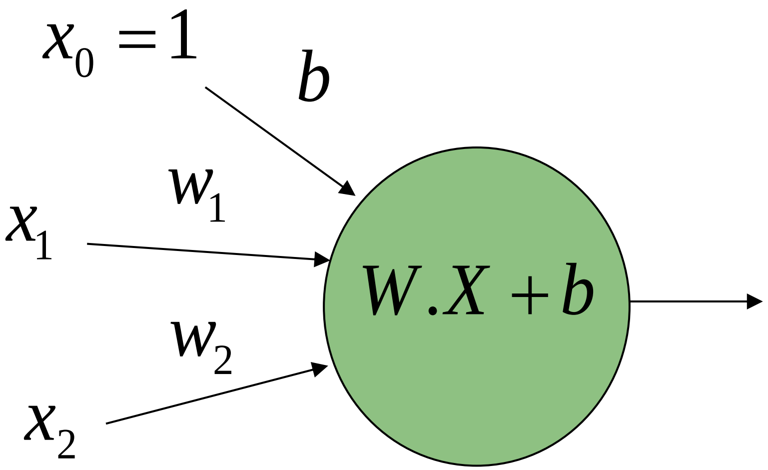
Predicción de la Producción de Energía (PE)

X

```
[[14.96, 41.76],
 [25.18, 62.96],
 [ 5.11, 39.4 ],
 ...,
 [31.32, 74.33],
 [24.48, 69.45],
 [21.6 , 62.52]]
```

Y

```
[463.26,
 444.37,
 488.56,
 ...,
 429.57,
 435.74,
 453.28]
```



$$\hat{y} = \sum_{i=1}^{2} w_i . x_i + b$$

$$\underbrace{(-8.50 \quad 7.68)}_{\mathbf{W}} * \begin{pmatrix} 14.96 \\ 41.73 \end{pmatrix} + \underbrace{174.80}_{\mathbf{b}}$$

$$\hat{y} = 368.49$$

$$y = 463.26$$

Error cometido en este ejemplo $\longrightarrow$ $Error = (y - \hat{y}) = 94.76$

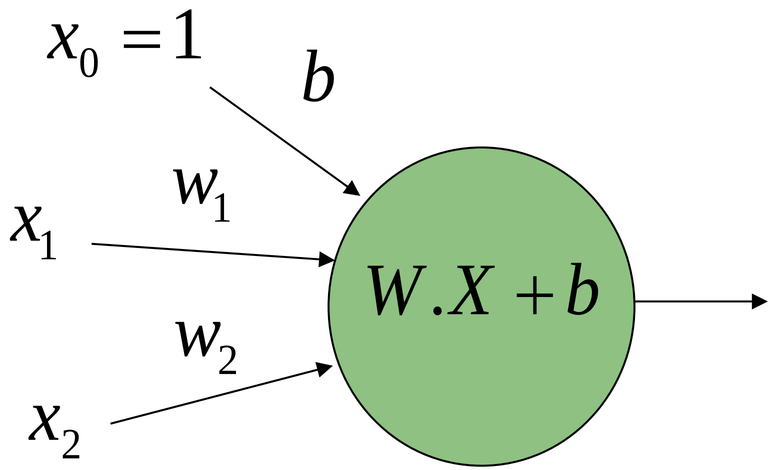
Predicción de la Producción de Energía (PE)

X

```
[[14.96, 41.76],
 [25.18, 62.96],
 [ 5.11, 39.4 ],
 ...,
 [31.32, 74.33],
 [24.48, 69.45],
 [21.6 , 62.52]]
```

Y

```
[463.26,
 444.37,
 488.56,
 ...,
 429.57,
 435.74,
 453.28]
```



$$\hat{y} = \sum_{i=1}^{2} w_i . x_i + b$$

$$\underbrace{(-8.47 \quad 7.76)}_{\mathbf{W}} * \begin{pmatrix} 14.96 \\ 41.73 \end{pmatrix} + \underbrace{174.80}_{\mathbf{b}}$$

$$\hat{y} = 372.23$$

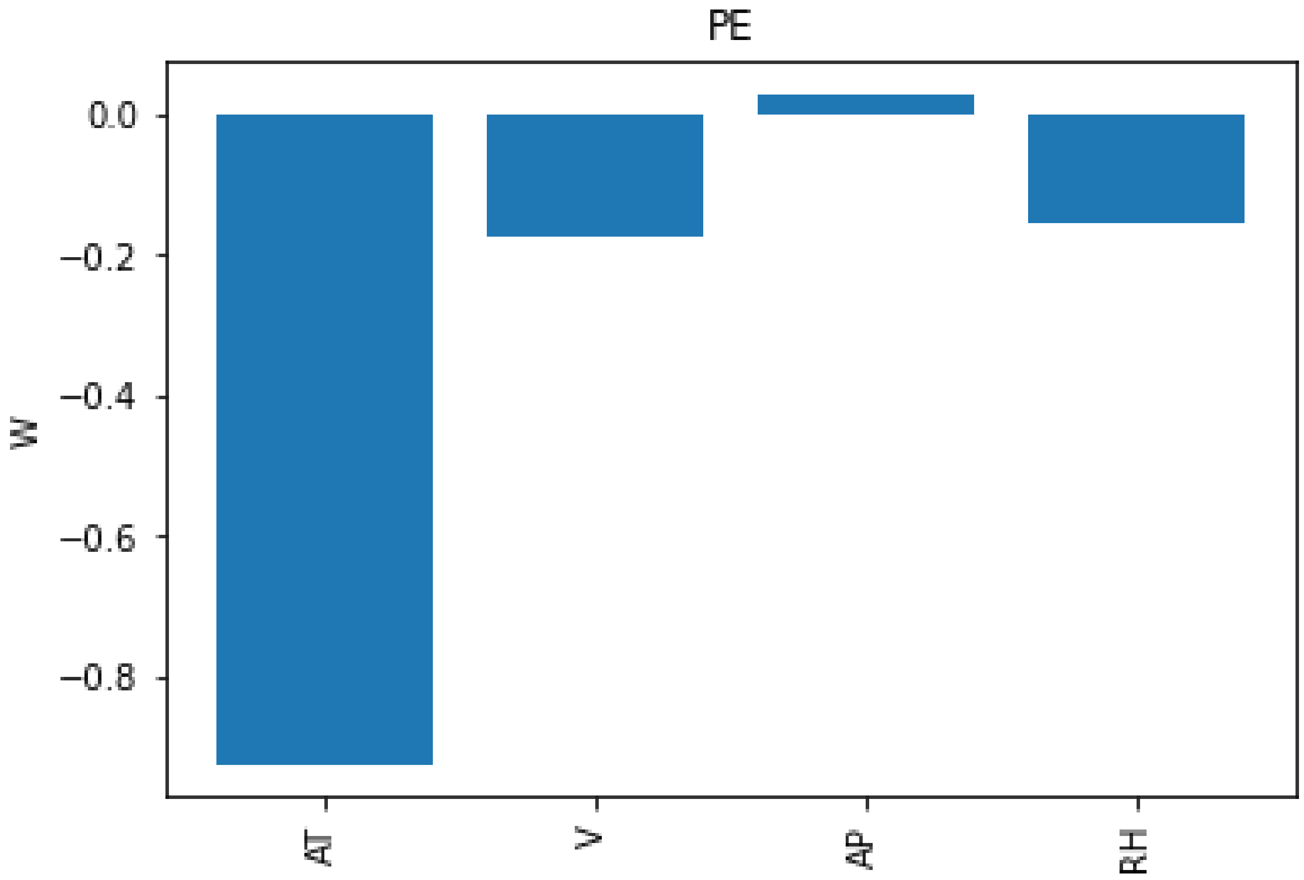
$$y = 463.26$$

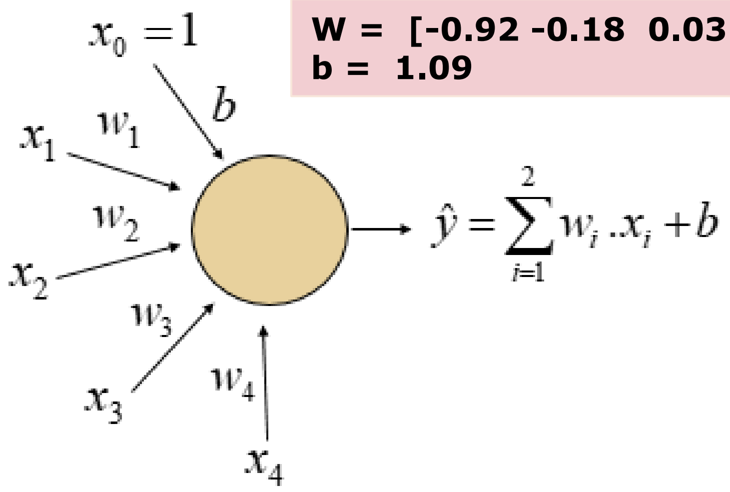
Error cometido en este ejemplo $\longrightarrow$ $Error = (y - \hat{y}) = 91.03$

Predicción de la Producción de Energía (PE)

(Atributos normalizados linealmente entre 0 y 1)

- PE = -0.92 * AT -0.18 * V + 0.03 * AP - 0.15 * RH +





W = [-0.92 -0.18 0.03 -0.15]
b = 1.09

Comb_LINEAL_CCPPx10_RN.ipynb

Comb_LINEAL_HOUSING_RN.ipynb

California Housing prices

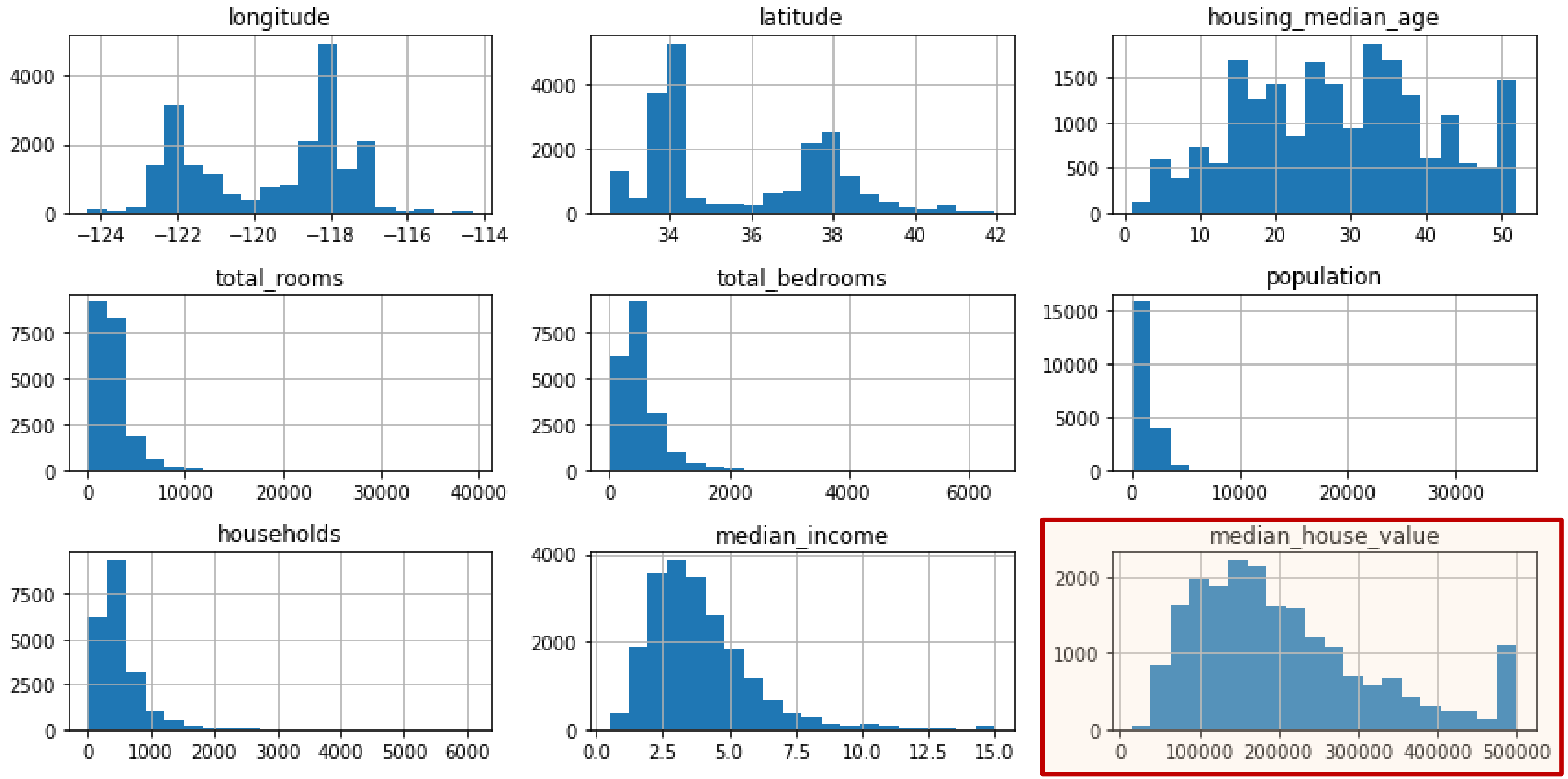
- Los datos se refieren a las casas encontradas en un distrito determinado de California y algunas estadísticas resumidas sobre ellas basadas en los datos del censo de 1990

- Objetivo: Predecir el precio de una casa.

- Analizar los atributos antes de usarlos.

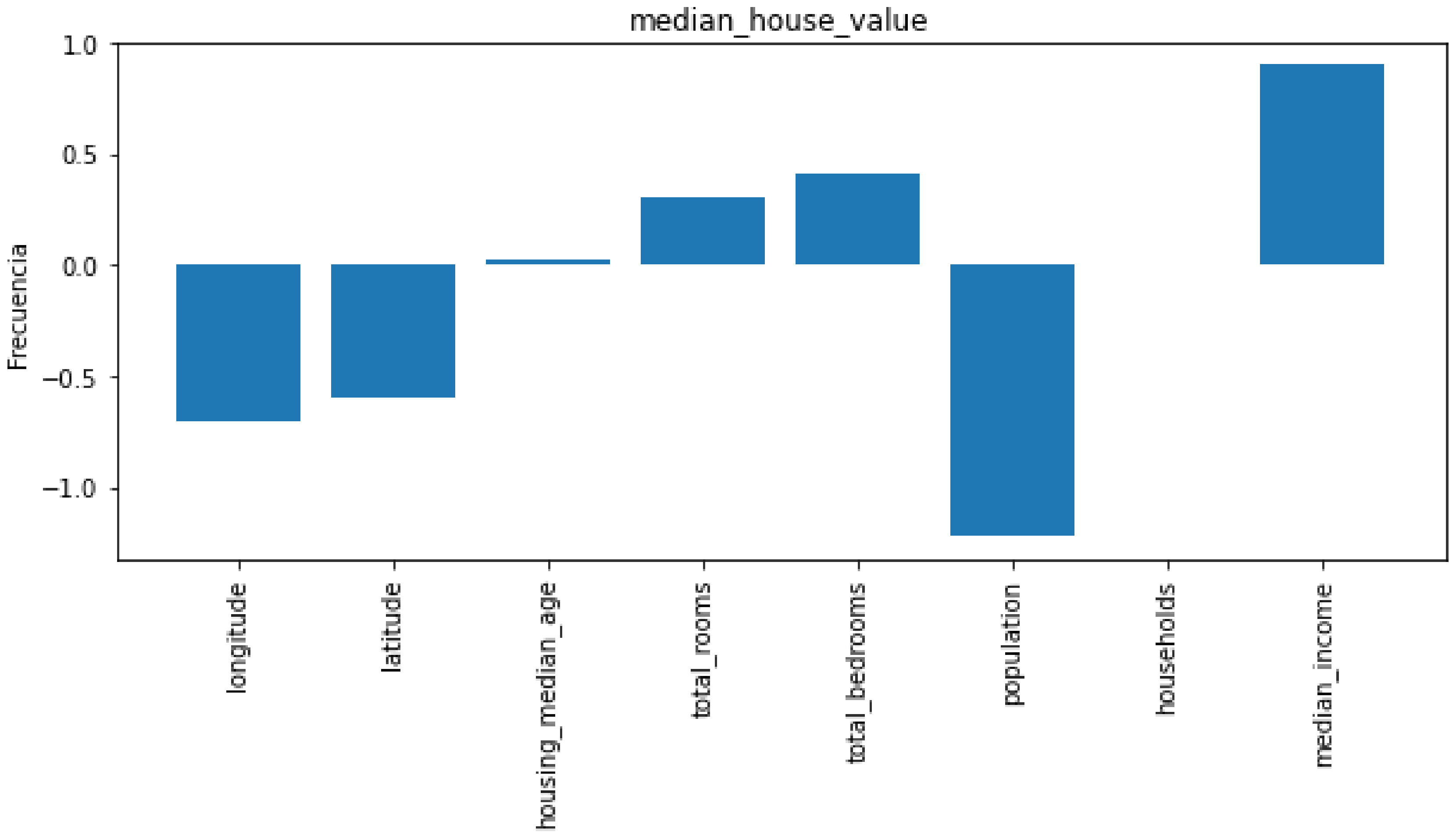
- **longitud**: Una medida de cuán al oeste se encuentra una casa; un valor más alto indica una ubicación más al oeste.
- **latitud**: Una medida de cuán al norte se encuentra una casa; un valor más alto indica una ubicación más al norte.
- **housingMedianAge**: Edad mediana de una casa dentro de una manzana; un número más bajo indica un edificio más nuevo.
- **totalRooms**: Número total de habitaciones dentro de una manzana.
- **totalBedrooms**: Número total de dormitorios dentro de una manzana.
- **población**: Número total de personas que residen dentro de una manzana.
- **households**: Número total de hogares, un grupo de personas que residen dentro de una unidad habitacional, para una manzana.
- **medianIncome**: Ingreso medio de los hogares dentro de una manzana de casas (medido en decenas de miles de dólares estadounidenses).
- **medianHouseValue**: Valor medio de la casa para los hogares dentro de una manzana (medido en dólares estadounidenses).
- **oceanProximity**: Ubicación de la casa con respecto al océano/mar.

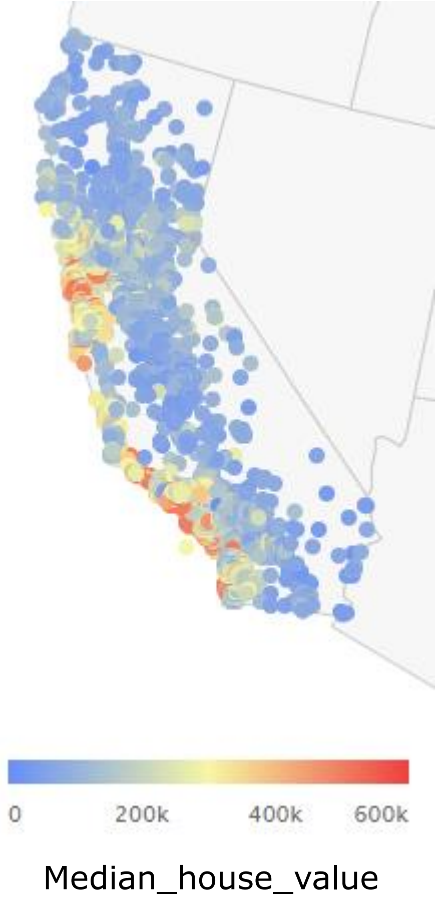
California Housing prices



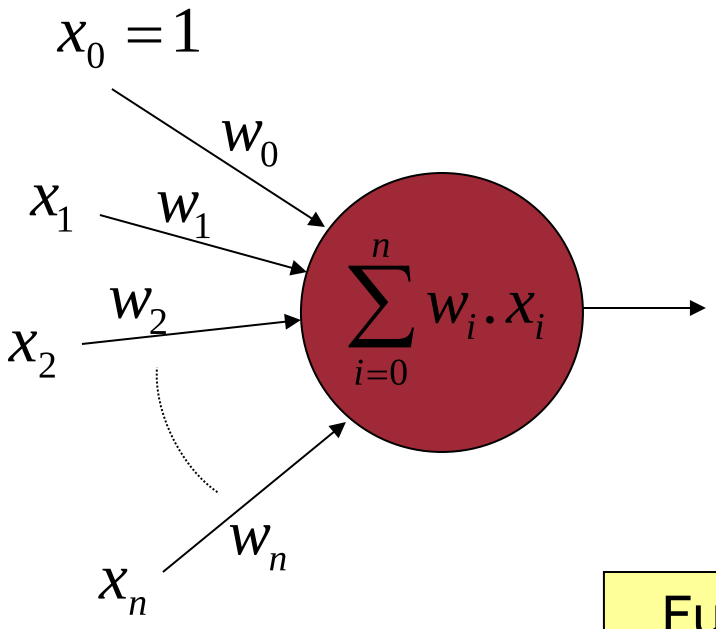
Comb_LINEAL_HOUSING_RN.ipynb

Vector W – ejemplos normalizados linealmente





Neurona General



$$y_k = f\left(\sum_{i=0}^{n} w_i . x_i\right)$$

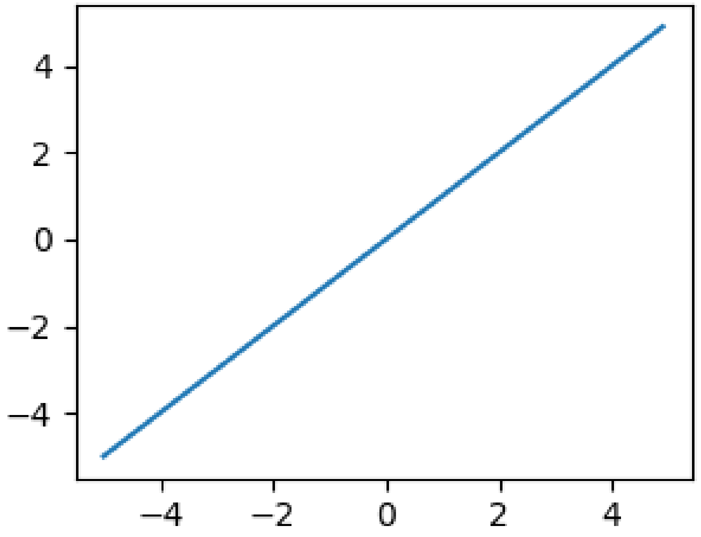
Función derivable

Función de Salida LINEAL

$$f(x) = x \qquad f'(x) = 1$$

desde Python

```
import numpy as np
import grafica as gr
from matplotlib import pyplot as plt

x = np.array(range(-50,50,1))/10.0
y = gr.evaluar('purelin', x)
plt.plot(x,y,'-')
```



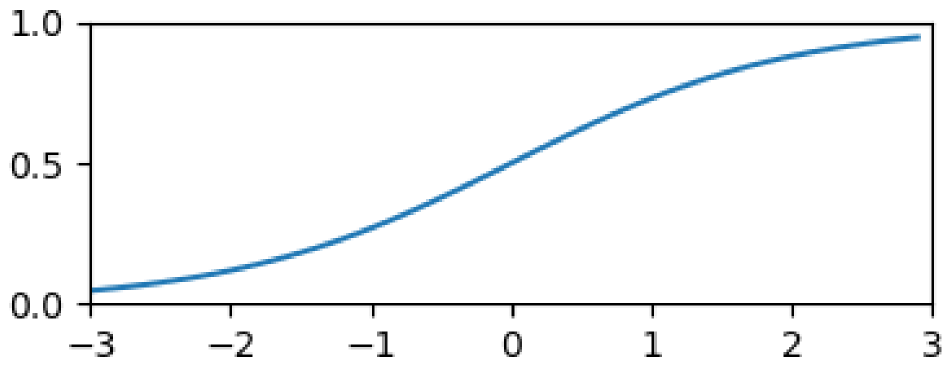
Función SIGMOIDE $\in$ (0,1)

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

$$f'(x) = f(x)*(1 - f(x))$$

desde Python

```
import numpy as np
import grafica as gr
from matplotlib import pyplot as plt

x = np.array(range(-30,30,1))/10.0
y = gr.evaluar('logsig', x)
plt.plot(x,y,'-')
plt.axis([-3, 3, 0, 1])
plt.show()
```



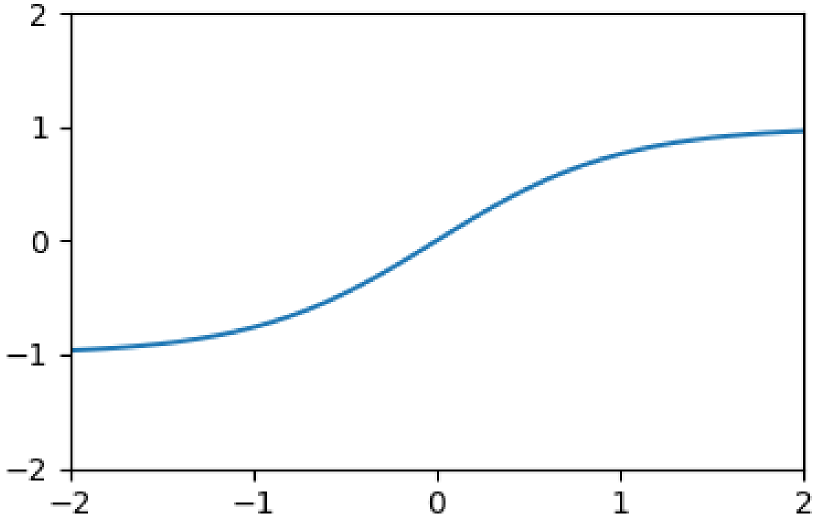
Función SIGMOIDE ∈ (-1,1)

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

$$f'(x) = 1 - f(x)^2$$

desde Python

```
import numpy as np
import grafica as gr
from matplotlib import pyplot as plt

x = np.array(range(-30,30,1))/10.0
y = gr.evaluar('tansig', x)
plt.plot(x,y,'-')
plt.axis([-2, 2, -2, 2])
plt.show()
```



Ejemplo

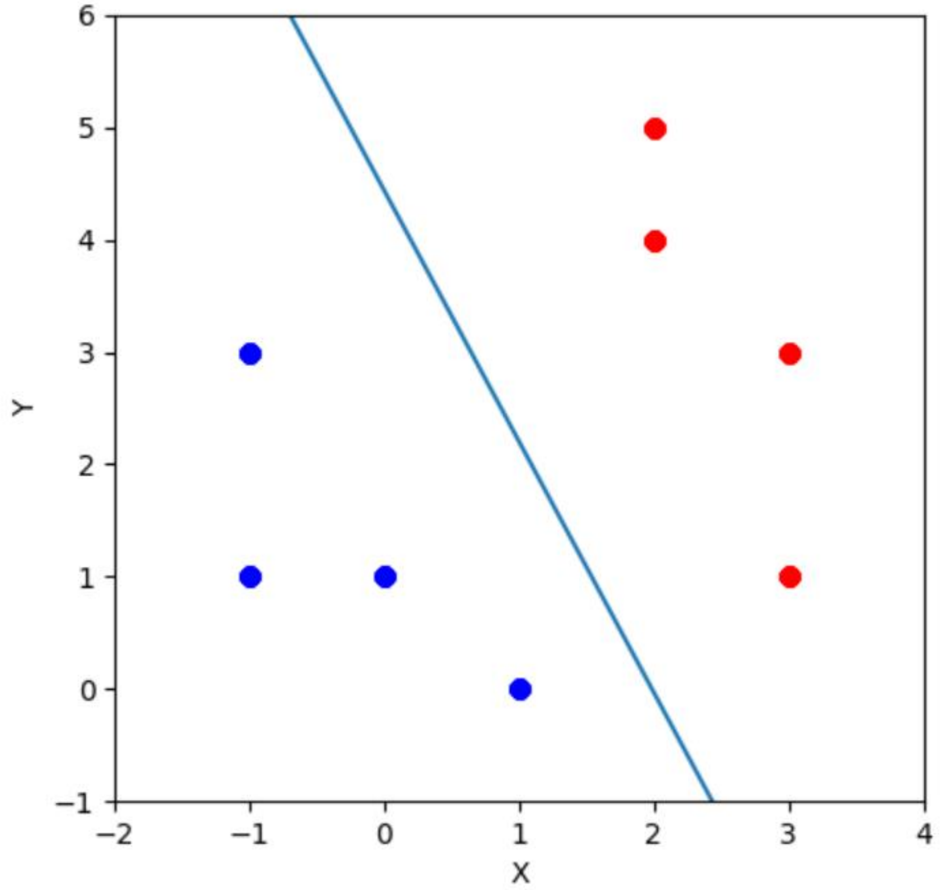
- Dados los siguientes conjuntos de puntos del plano

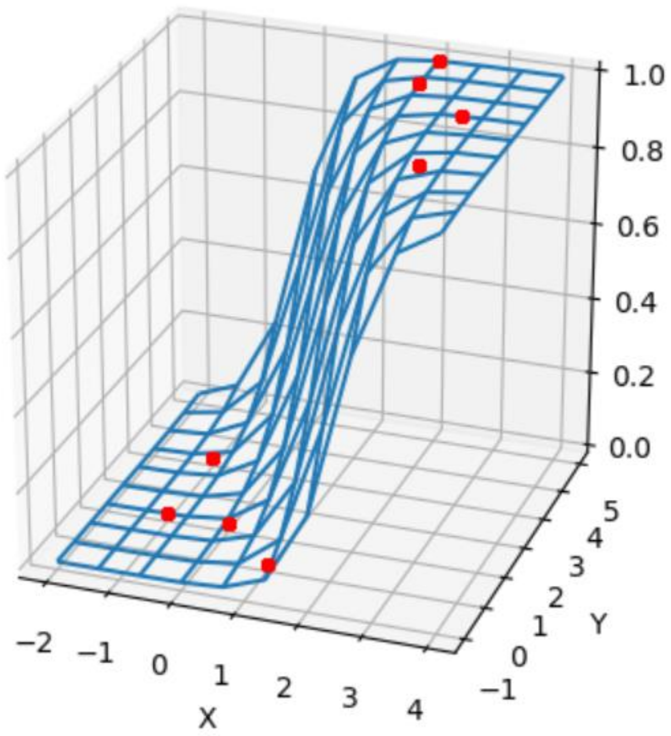
  A = {(2,2), (1,0), (0,1), (-1,1)}

  B = {(3,1), (3,3), (2,4), (2,5)}

  - Utilice una **neurona no lineal** para clasificarlos
  - Representar gráficamente la solución propuesta.

A = {(-1,3), (1,0), (0,1), (-1,1)}
B = {(3,1), (3,3), (2,4), (2,5)}





neuronaNoLineal.py