

Prácticas de Aprendizaje Automático

Trabajo 1: Búsqueda Iterativa de Óptimos y Regresión Lineal

Pablo Mesejo y Francisco Baldán

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD
DE GRANADA



Recordatorio normas (1). Informe.

.zip = Código (.py) + Informe (.pdf)

- Presentar un **informe escrito** con las valoraciones y **decisiones** adoptadas en cada apartado
 - No es solo hacer algo → hay que argumentar el por qué
- Incluir en el informe los **gráficos** generados.
- Incluir una **valoración/discusión de los resultados obtenidos**.
- El informe debe presentarse en PDF
- Si no hay informe → se considera que el trabajo no se ha presentado

Recordatorio normas (2). Código.

- **Único script Python.**
 - Los distintos ejercicios van en apartados comentados dentro del fichero
- **Todos los resultados numéricos o gráficas serán mostrados por pantalla**, parando la ejecución después de cada apartado.
 - No escribir nada en el disco
- El path que se use en la lectura de cualquier fichero auxiliar de datos debe ser siempre **"datos/nombre_fichero"**.
 - Crear directorio llamado "datos" dentro del directorio donde se desarrolla y se ejecuta la práctica

Recordatorio normas (3). Código.

- El código **debe ejecutarse de principio a fin sin errores.**
- No es válido usar opciones en las entradas.
 - **Fijar al comienzo los parámetros por defecto** que considere óptimos.
- El código debe estar obligatoriamente **comentado** explicando lo que realizan los distintos apartados
 - **Id comentando el código** que hagáis: sirve para que entendáis mejor lo que habéis hecho, y facilita mi trabajo a la hora de corregir los ejercicios.
- Entregar **solo el código fuente, nunca los datos.**

Recordatorio normas (y 4)

.zip = Código (.py) + Informe (.pdf)

Subir el zip a PRADO, a la actividad creada para ello.

Fecha de entrega: 5 de Abril

Template

- Podéis partir, si queréis, del template que hemos preparado y del que disponéis en PRADO

```
# -*- coding: utf-8 -*-
"""
TRABAJO 1.
Nombre Estudiante:
"""

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(1)

print('EJERCICIO SOBRE LA BUSQUEDA ITERATIVA DE OPTIMOS\n')
print('Ejercicio 1\n')

def E(u,v):
    return #function

#Derivada parcial de E con respecto a u
def dEu(u,v):
    return #Derivada parcial de E con respecto a u

#Derivada parcial de E con respecto a v
def dEv(u,v):
    return #Derivada parcial de E con respecto a v

#Gradiente de E
def gradE(u,v):
    return np.array([dEu(u,v), dEv(u,v)])

def gradient_descent(?):
    #
    # gradiente descendente
    #
    return w, iterations

eta = 0.01
maxIter = 10000000000
error2get = 1e-14
initial_point = np.array([1.0,1.0])
w, it = gradient_descent(?)

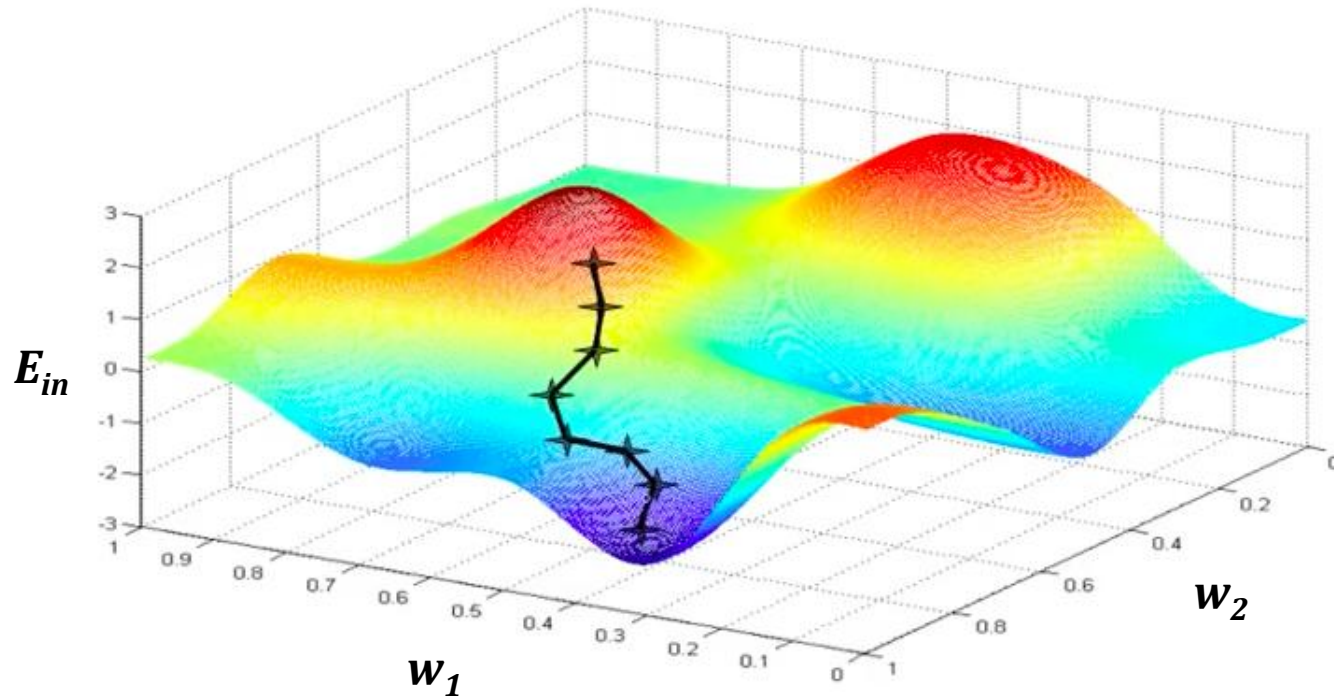
print('Numero de iteraciones: ', it)
print('Coordenadas obtenidas: (', w[0], ', ', w[1],')')
```

1. Búsqueda iterativa de óptimos

- Implementar el algoritmo de gradiente descendente
 - Algoritmo para minimizar funciones
 - Requiere una función derivable a minimizar
 - Es un algoritmo local: empieza en un punto y va descendiendo por la pendiente más pronunciada
 - El gradiente apunta en la dirección de mayor crecimiento de la función, y su magnitud es la pendiente en dicha dirección
 - Como estamos minimizando, se emplea el signo contrario al gradiente

$$w_j := w_j - \eta \frac{\partial E_{in}(\mathbf{w})}{\partial w_j}$$

1. Búsqueda iterativa de óptimos



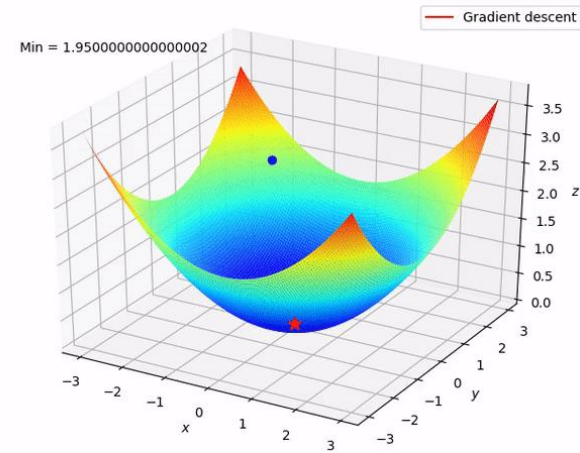
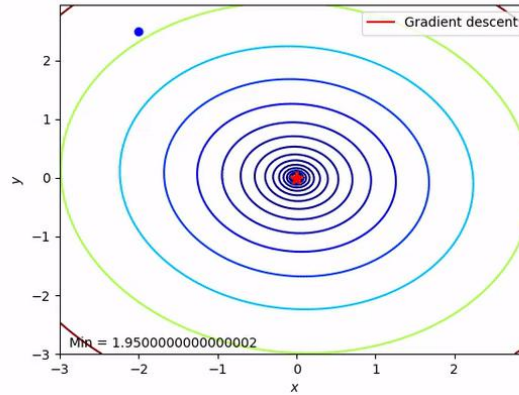
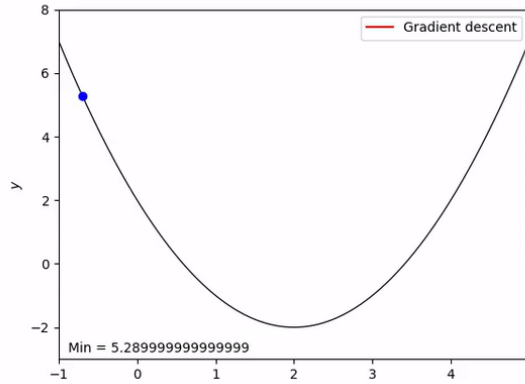
Ejemplo:
Función con dos
pesos/parámetros

Se busca minimizar el
error E_{in}

Partiendo de un punto
inicial

Se desciende por la
dirección de mayor
pendiente

1. Búsqueda iterativa de óptimos



Animaciones extraídas de https://jed-ai.github.io/py1_gd_animation/

1. Búsqueda iterativa de óptimos

- We want to choose \mathbf{w} so as to minimize $E_{in}(\mathbf{w})$
- Gradient Descent (GD):
 - Gradient descent is a general iterative optimization technique that reach a local optimum following the direction of the gradient vector on each point.

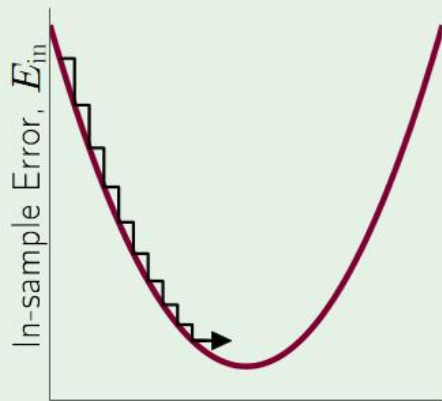
It starts on some initial value \mathbf{w} and repeatedly perform the update ,

$$w_j := w_j - \eta \frac{\partial E_{in}(\mathbf{w})}{\partial w_j} \quad (\text{GENERAL EQUATION})$$

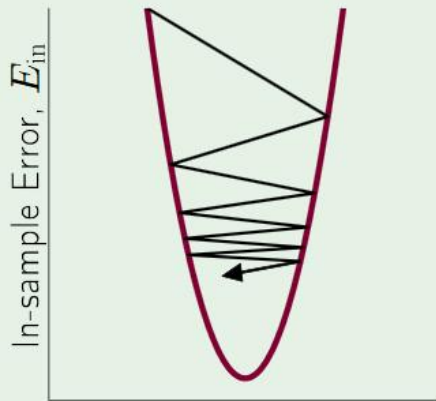
(This update is simultaneously performed for all values of $j = 0, \dots, n$). Here, η is called the learning rate.

1. Búsqueda iterativa de óptimos

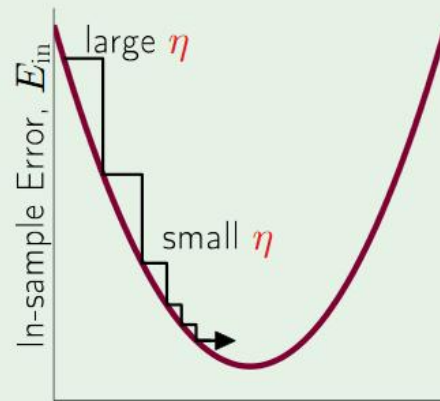
How η affects the algorithm:



η too small



η too large



variable η – just right

η should increase with the slope



1. Búsqueda iterativa de óptimos

- Implementar el algoritmo de gradiente descendente

1. Una función que implemente el gradiente descendente

```
def gradient_descent(?):
```

2. ¿Cuál es el cuerpo de la función?

$$w_j := w_j - \eta \frac{\partial E_{in}(\mathbf{w})}{\partial w_j}$$

3. ¿Qué argumentos se le pasan a la función?

1. Búsqueda iterativa de óptimos

- Recomendaciones
 - Imprimid el valor de la función en cada punto del descenso de gradiente → **verificad que los valores van disminuyendo**
 - Si algo no va bien, y no dais encontrado el error, **revisad las derivadas** (probablemente no estén bien calculadas)

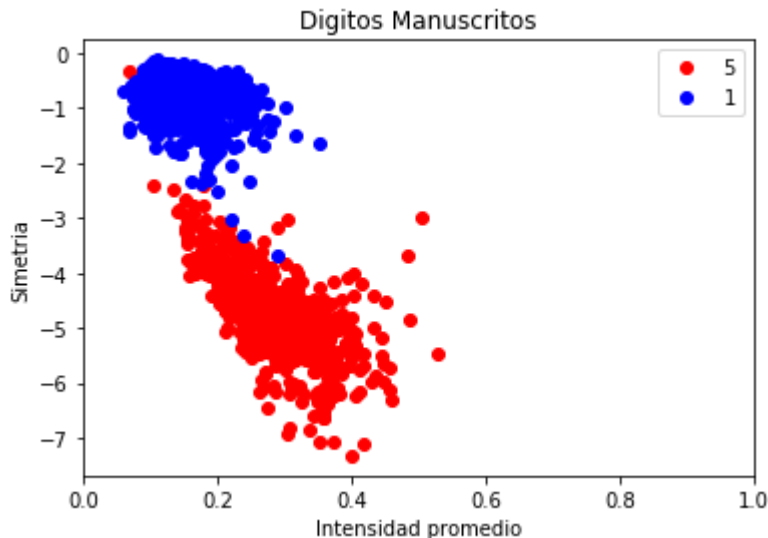
1. Búsqueda iterativa de óptimos

- Limitaciones del gradiente descendente
 - Necesidad de función derivable
 - Importancia del punto inicial (es una búsqueda local)
 - Un mínimo local de una función convexa es un mínimo global
 - Importancia del learning rate
 - Demasiado grande → podríamos no converger
 - Demasiado pequeño → llevaría demasiado tiempo

🗨️ 2. Ejercicio sobre Regresión Lineal

- En el template tenéis una función para leer los datos: `def readData(file_x, file_y)`
- La idea es usar regresión lineal para clasificación de dígitos

$$y = w_0 + w_1x_1 + w_2x_2$$



2. Ejercicio sobre Regresión Lineal

- Al final, todo consiste en estimar los \mathbf{w} 's
 - Gradient Descent
 - Stochastic Gradient Descent
 - Pseudoinversa (*one-step learning*)
 - BONUS: Método de Newton

Gradient Descent vs Stochastic Gradient Descent

Gradient Descent

It starts on some initial value \mathbf{w} and repeatedly perform the update ,

$$w_j := w_j - \eta \frac{\partial E_{in}(\mathbf{w})}{\partial w_j} \quad (\text{GENERAL EQUATION})$$

(This update is simultaneously performed for all values of $j = 0, \dots, n$). Here, η is called the learning rate.

$$\frac{\partial E_{in}(\mathbf{w})}{\partial w_j} = \frac{\partial}{\partial w_j} \frac{1}{N} \sum_{n=1}^N \text{función del error cu} = \frac{2}{N} \sum_{n=1}^N x_{nj} (\mathbf{w}^T \mathbf{x}_n - y_n) = \frac{2}{N} \sum_{n=1}^N x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

Each point (\mathbf{x}_n, y_n) contributes to the update by an amount proportional to its prediction error

In this case all points are used to compute the gradient: **BATCH GRADIENT DESCENT**

Gradient Descent vs Stochastic Gradient Descent

Stochastic Gradient Descent

- An alternative is to use a **stochastic estimation** using only a part of the sample to compute the gradient, $M \ll N$ (SGD)

$$\frac{\partial E_{in}(\mathbf{w})}{\partial w_j} = \frac{2}{M} \sum_{n=1}^M x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

- Higher variability in the gradient estimation (less examples in the average)
 - Very fast of computing
 - In non-convex funtions empirical evidence of getting good local minimum
- Although an only item could be used on each iteration, a minibatch of items is the accepted rule (size: 32-128)



Gradient Descent vs Stochastic

Batch Gradient Descent

- Given the data set $(\mathbf{x}_n, y_n), n = 1, 2, \dots, N$
 - Fix $\mathbf{w}=0, \eta = \eta_0$
 - Iterate
For $j=0, \dots, K$:
 $w_j := w_j - \eta \sum_n \text{Falta el } h(\mathbf{x}_n) - y_n$ (all sample participate)
 - Until $E_{in}(\mathbf{w}) < \text{epsilon}$

Stochastic Gradient Descent

- Fix $\mathbf{w}=0, \eta = \eta_0$
- Iterate:
- Shuffle and Split the sample into a sequence of mini-batches**
- Iterate on mini-batches
For $j=0, \dots, K$:
 $w_j := w_j - \eta \sum_{n \in \text{Minibatch}} \text{Aquí tambien falta } h(\mathbf{x}_n) - y_n$ (only a mini-batch participate)
- Until $E_{in}(\mathbf{w}) < \text{epsilon}$

Gradient Descent vs Stochastic Gradient Descent

Batch Gradient Descent

Vs

Stochastic Gradient Descent

all points are used to compute the gradient

$$\frac{2}{N} \sum_{n=1}^N x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

$M \ll N$ (SGD)

$$\frac{\partial E_{in}(\mathbf{w})}{\partial w_j} = \frac{2}{M} \sum_{n=1}^M x_{nj} (\mathbf{h}(\mathbf{x}_n) - y_n)$$

2. Ejercicio sobre Regresión Lineal: Pseudoinversa

A Linear Regression Algorithm

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{2}{N} \mathbf{X}^T (\mathbf{X}\mathbf{w} - \mathbf{y}) = \mathbf{0}$$

$$\mathbf{X}^T \mathbf{X} \mathbf{w} = \mathbf{X}^T \mathbf{y}$$

$$\mathbf{w} = \mathbf{X}^\dagger \mathbf{y} \quad \text{where} \quad \mathbf{X}^\dagger = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$$

\mathbf{X}^\dagger is the 'pseudo-inverse' of \mathbf{X}

- 1: Construct the matrix \mathbf{X} and the vector \mathbf{y} from the data set $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)$ as follows

$$\underbrace{\mathbf{X} = \begin{bmatrix} -\mathbf{x}_1^T- \\ -\mathbf{x}_2^T- \\ \vdots \\ -\mathbf{x}_N^T- \end{bmatrix}}_{\text{input data matrix}}, \quad \underbrace{\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\text{target vector}}.$$

- 2: Compute the pseudo-inverse $\mathbf{X}^\dagger = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$.
- 3: Return $\mathbf{w} = \mathbf{X}^\dagger \mathbf{y}$.

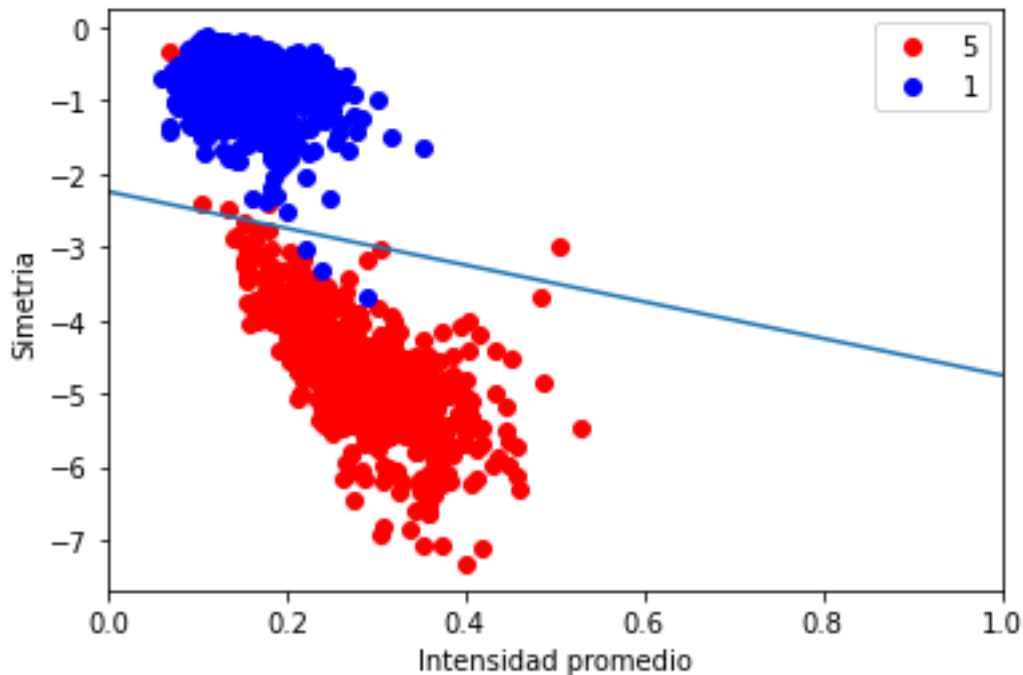
Can we always compute $(\mathbf{X}^T \mathbf{X})^{-1}$?

- Let consider the Singular Value Decomposition (SVD) : $\mathbf{X} = \mathbf{U} \mathbf{D} \mathbf{V}^T$
- $\mathbf{X}^T \mathbf{X} = \mathbf{V} \mathbf{D} \mathbf{D}^T \mathbf{V}^T$

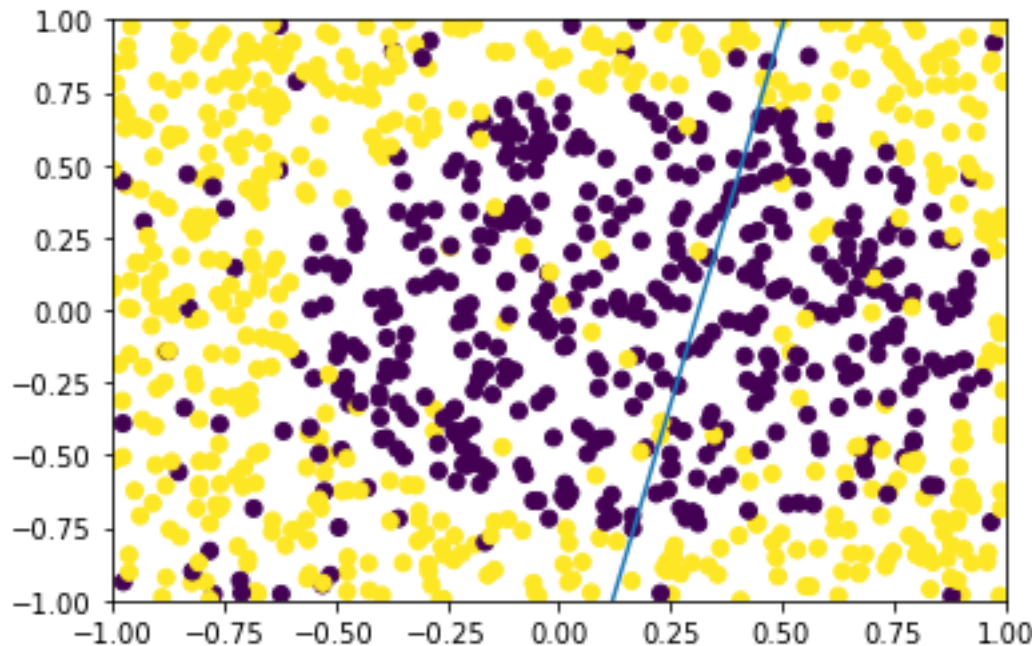


2. Ejercicio sobre Regresión Lineal

como se representa



2. Ejercicio sobre Regresión Lineal



Referencias interesantes:

<https://medium.com/@lucaspereira0612/solving-xor-with-a-single-perceptron-34539f395182>

<http://work.caltech.edu/slides/slides03.pdf> (slides 19-23)



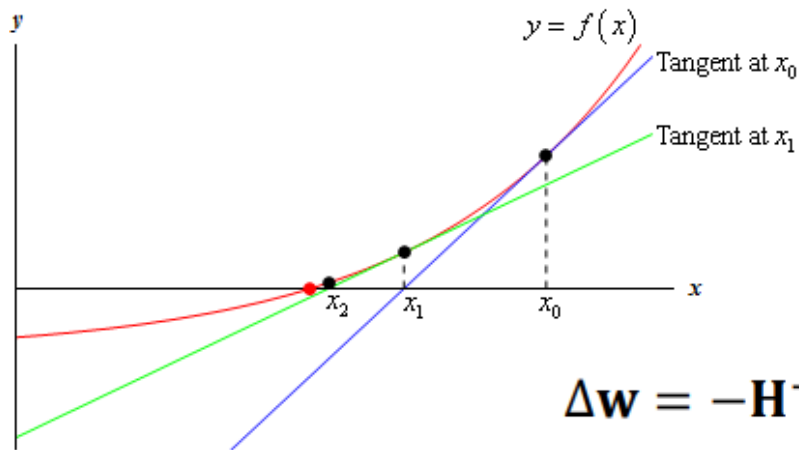
BONUS: Método de Newton

A new update rule for \mathbf{w} based on the **second order derivatives** (Hessian)

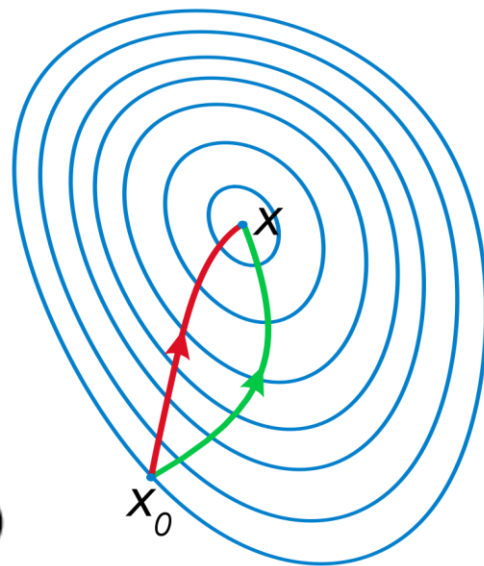
$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}.$$

$$f''(x) = \nabla^2 f(x) = H_f(x) \in \mathbb{R}^{d \times d}$$

$$x_{k+1} = x_k - \gamma [f''(x_k)]^{-1} f'(x_k).$$



$$\Delta \mathbf{w} = -\mathbf{H}^{-1} \nabla E_{\text{in}}(\mathbf{w}_0)$$



Enlaces recomendados

- Materiales de Andrew Ng sobre *linear regression* y *gradient descent* (lectures 2 and 4):
https://www.youtube.com/playlist?list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN
- Materiales de Yaser Abu-Mostafa
(<http://work.caltech.edu/lectures.html>):
 - The linear model I: <https://www.youtube.com/watch?v=FlbVs5GbBIQ>
 - The linear model II: <https://www.youtube.com/watch?v=qSTHZvN8hzs>

Prácticas de Aprendizaje Automático

Trabajo 1: Búsqueda Iterativa de Óptimos y Regresión Lineal

Pablo Mesejo y Francisco Baldán

Universidad de Granada

Departamento de Ciencias de la Computación e Inteligencia Artificial



UNIVERSIDAD
DE GRANADA

