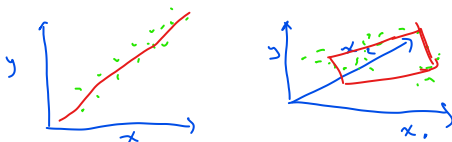


Outline

Perceptron

Gradient descent

Linear regression



Regression model that approximates y from input data \mathbf{x} , using the set of weights $\mathbf{w} = \{w_i\}$,

$$\mathbf{y} = \mathbf{X}\mathbf{w}.$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} ; \mathbf{X} = \begin{bmatrix} x_1^1 & x_1^2 & \dots & x_1^n \\ x_2^1 & x_2^2 & \dots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ x_m^1 & x_m^2 & \dots & x_m^n \end{bmatrix}$$

We could learn \mathbf{w} using the normal equation (least squares):

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}.$$

$$\mathbf{w} \approx \frac{\mathbf{X}^T \mathbf{y}}{\mathbf{X}^T \mathbf{X}} \leftarrow \text{No está definido en álgebra lineal}$$

Logistic regression

Similarly, we could fit a logistic function to perform binary classification: true vs false (0 vs 1).

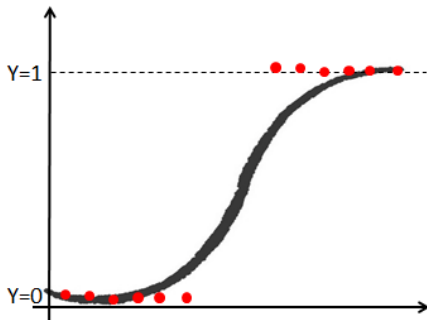
$$z = \mathbf{w}^T \mathbf{x},$$

$$y = \sigma(z),$$

where,

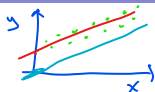
$$\sigma(z) = \frac{1}{1 + \exp^{-z}},$$

is the sigmoid function.



It actually, gives the probability of $y = 1$.

Linear perceptron



$$y = \underline{w}x + \underline{b}$$

↳ shift variable
↳ pendiente

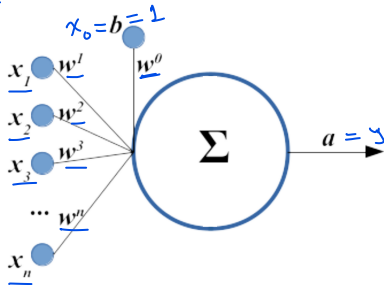
Another formulation for regression problems. $y = w_1x_1 + w_2x_2 + \dots + b$

N : # de pesos = # de variables en el vector de entrada

$$y = \mathbf{w}^T \mathbf{x},$$

$$= \sum_{n=0}^N \omega_n x_n,$$

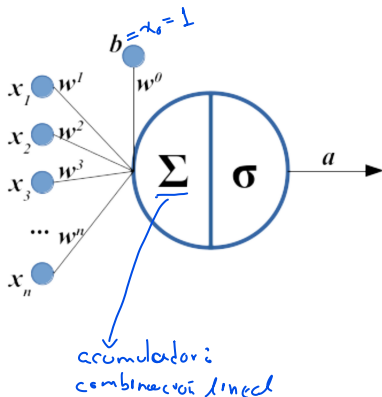
$$= \sum_{n=1}^N \omega_n x_n + \omega_0 x_0, \quad \text{where } \omega_0 = b \text{ and } x_0 = 1.$$



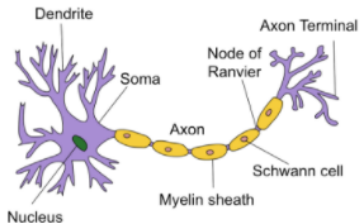
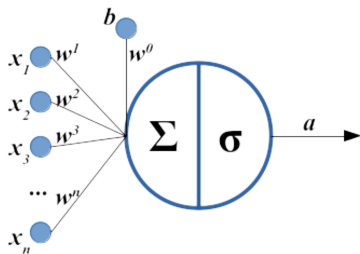
Perceptron

Let's use the sigmoid activation function.

$$s = \mathbf{w}^T \mathbf{x}_i \rightarrow \text{combination linear}$$
$$a = \sigma(s).$$



Artificial neuron

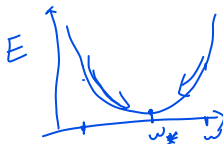


Outline

Perceptron

Gradient descent

Weights estimation



To estimate values for $\{w_i\}$ we use an iterative minization approach termed *Gradient Descent* (GD).

- ▶ Most complex problems have no closed-form solution.
- ▶ Iterative approaches reach fairly good approximations.
- ▶ Risk of getting trapped in local minima.



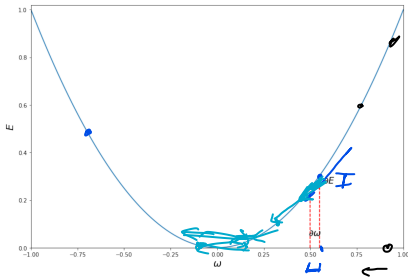
Gradient descent (GD)

$$\mathcal{L} = (y - \hat{y})^2$$



We require a *loss function*. e.g., $E = (y - \hat{y})^2$.

Remember: relation between derivative, tangent, and direction.



$$\frac{\partial E}{\partial \omega_i} = \lim_{h \rightarrow 0} \frac{f(\omega_i + h) - f(\omega_i)}{h}.$$

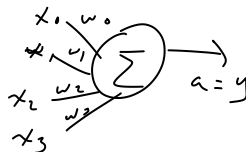
And we move in the opposite direction of the derivative,

$$\underline{\omega_i} = \underline{\omega_i} - \underline{\eta} \frac{\partial E}{\partial \omega_i}.$$

GD example, I

Consider first only a linear perceptron:

- ▶ $\hat{y} = \mathbf{w}^T \mathbf{x} = \sum_{n=0}^N \omega_n x_n$, ✓
- ▶ $E = 0.5(y - \hat{y})^2$. ✓



Then,

$$\begin{aligned}
 \frac{\partial E}{\partial \omega_n} &= 0.5 \frac{\partial (y - \hat{y})^2}{\partial \omega_n}, \\
 &= 0.5(2)(y - \hat{y}) \frac{\partial (y - \hat{y})}{\partial \omega_n}, \\
 &= (y - \hat{y}) \left[0 - \frac{\partial \sum_{n=0}^N \omega_n x_n}{\partial \omega_n} \right], \\
 &= \underline{-(y - \hat{y})x_n}. \quad \checkmark
 \end{aligned}$$

Therefore,

$$\underline{\omega_n} = \underline{\omega_n} + \underline{\eta}(y - \hat{y})\underline{x_n}.$$

GD example, II

Consider now a non-linear perceptron:

- ▶ $\hat{y} = \sigma(s)$, ✓
- ▶ $s = \mathbf{w}^T \mathbf{x} = \sum_{n=0}^N \omega_n x_n$, ✓
- ▶ $E = 0.5(y - \hat{y})^2$. ✓

The derivative of the sigmoid function is: $\sigma'(s) = \sigma(s)(1 - \sigma(s))$.

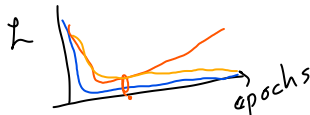
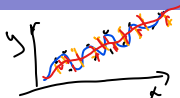
Then,

$$\begin{aligned} \frac{\partial E}{\partial \omega_n} &= \frac{\partial (y - \hat{y})^2}{\partial \omega_n}, \\ &= -(y - \hat{y}) \underbrace{\sigma(s)(1 - \sigma(s))}_{\frac{\partial \sigma(s)}{\partial \omega_n}} x_n. \end{aligned}$$

Therefore,

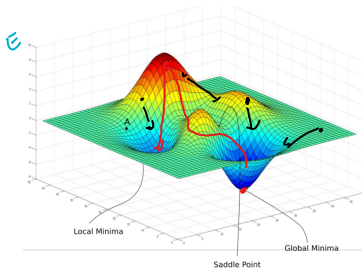
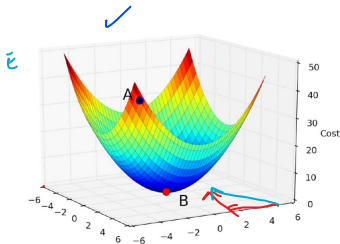
$$\omega_n = \omega_n + \underbrace{\eta(y - \hat{y})}_{\frac{\partial E}{\partial \omega_n}} \underbrace{\sigma(s)(1 - \sigma(s))}_{\frac{\partial \sigma(s)}{\partial \omega_n}} x_n.$$

GD multivariado



We can use it for multiple parameters.

- ▶ We always must move in the direction of the steepest descent, so first compute the all partial derivatives and then update.



GD procedure

1. Random initialization.
2. Forward pass.
3. Error estimation.
4. Gradient computation.
5. Backward pass (weight adjustment).