Machine Learning

Perceptron

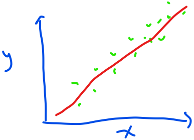
Edgar F. Roman-Rangel.
edgar.roman@itam.mx

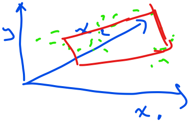
Digital Systems Department.
Instituto Tecnológico Autónomo de México, ITAM.

May 7th, 2021.

Outline

Linear regression





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

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

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} ; \quad X = \begin{bmatrix} x^1_1 & x^1_2 & \cdots & x^1_N \\ x^2_1 & x^2_2 & \cdots & x^2_N \\ \vdots & & & \\ x^M_1 & x^M_2 & \cdots & x^M_N \end{bmatrix}$$

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

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

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

Logistic regression

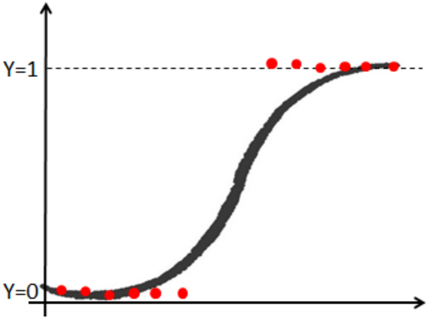
Similarly, we could fit a logistic function to perform binary classification: true vs false (0 vs 1).

$$z = \mathbf{w}^T \mathbf{x},$$
$$y = \sigma(z),$$

where,

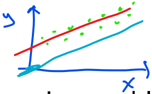
$$\sigma(z) = \frac{1}{1 + \exp^{-z}},$$

is the sigmoid function.



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

Linear perceptron

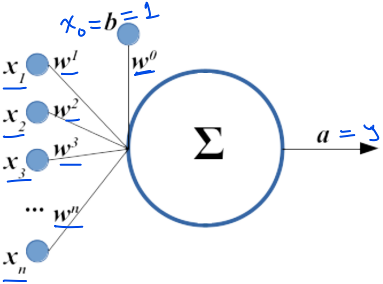


$y = wx + b$ ↳ shift vertical
↳ pendiente

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

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

$$
\begin{aligned}
y &= \mathbf{w}^T \mathbf{x}, \\
&= \sum_{n=0}^{N} \omega_n x_n, \\
&= \sum_{n=1}^{N} \omega_n x_n + \omega_0 \overset{=b}{x_0},
\end{aligned}
$$

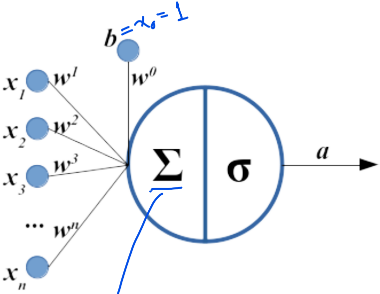


where, $\omega_0 = b$ and $x_0 = 1$.

Perceptron

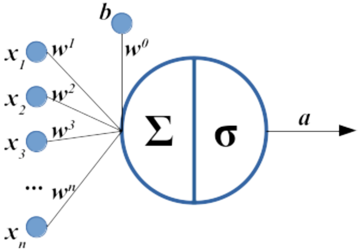
Let's use the sigmoid activation function.

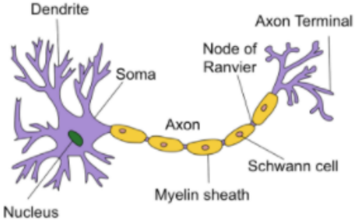
$$s = \underline{\mathbf{w}^T\mathbf{x}}, \rightarrow \text{combinación lineal}$$
$$a = \underline{\sigma}(s).$$



acumulador: combinación lineal

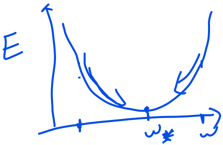
Artificial neuron





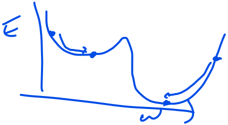
Outline

Weights estimation



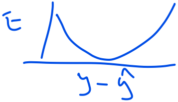
To estimate values for $\{w_i\}$ we use an iterative minization approach termed *Gradient Descent* (GD).

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



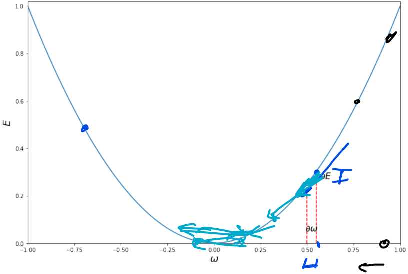
Gradient descent (GD)

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



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

Remember: relation between derivative, tangent, and direction.

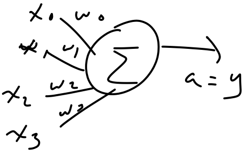


$$\frac{\partial E}{\partial \omega_i} = \lim_{h \to \infty} \frac{f(\omega_i + h) - f(\omega_i)}{h}.$$

And we move in the opposite direction of the derivative,

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

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,

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

Therefore,

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

GD example, II

Consider now a non-linar 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,

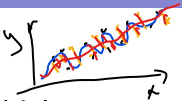
$$\frac{\partial E}{\partial \omega_n} = 0.5\,\frac{\partial (y - \hat{y})^2}{\partial \omega_n},$$
$$= -(y - \hat{y})\sigma(s)(1 - \sigma(s))x_n.$$

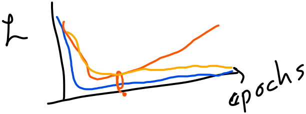
$$\frac{\partial s}{\partial \omega_n} = \frac{\partial \sigma(s)}{\partial \omega_n}$$

Therefore,

$$\omega_n = \omega_n + \eta(y - \hat{y})\sigma(s)(1 - \sigma(s))x_n.$$

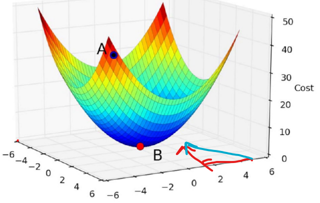
GD multivariado

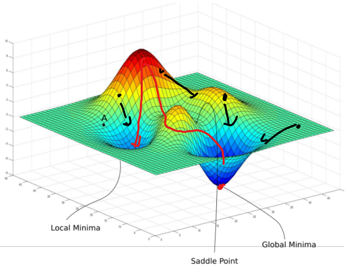




We can use it for multiple parameters.

- We always must move in the direction of the stepest 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).

Q&A

Thank you!

`edgar.roman@itam.mx`