

Logistic Regression and Gradient Descent

Machine Learning e Deep Learning

Davide Abati, Angelo Porrello

October 18th, 2018

University of Modena and Reggio Emilia

We are given a training set $\{X_i, Y_i\}_{i=1}^N$, with $X_i \in \mathbb{R}^m$ and $Y_i \in \{0, 1\}$ for each $i = 1, \dots, N$.

- N is the number of training examples;
- each example $X_i = \{x_i^{(1)}, \dots, x_i^{(m)}\}$ is a vector of m features;
- each label Y_i is either 0 or 1.

We need to learn a function that maps X to Y such that “it works well on the training set”.

We need to learn the parameters \mathbf{w} of a parametric function that maps X to Y such that “it works well on the training set”.

We need to learn the parameters \mathbf{w} of a parametric function that maps X to Y such that **some error is as low as possible on the training set.**

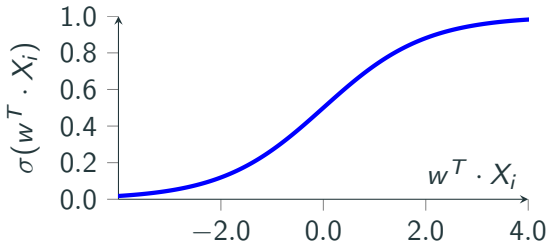
The function for classification has the following form:

$$F(X_i, w) = \sigma(w^T \cdot X_i), \quad \text{where} \quad \sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

The function for classification has the following form:

$$F(X_i, w) = \sigma(w^T \cdot X_i), \quad \text{where} \quad \sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

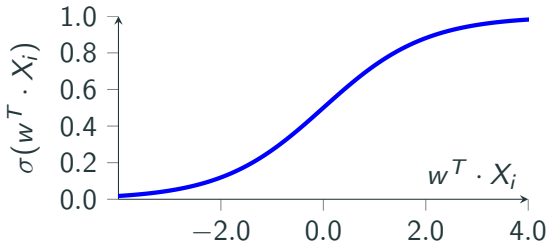
- $\sigma(x)$ is called **sigmoid function**;



The function for classification has the following form:

$$F(X_i, w) = \sigma(w^T \cdot X_i), \quad \text{where} \quad \sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

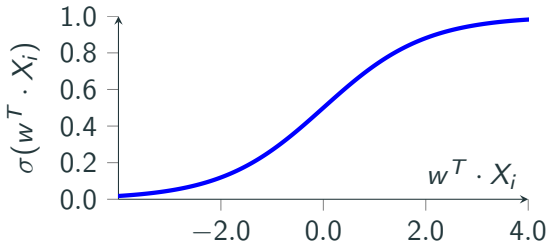
- $\sigma(x)$ is called **sigmoid function**;
- w is a vector in \mathbb{R}^m , and is called **weight vector**;



The function for classification has the following form:

$$F(X_i, w) = \sigma(w^T \cdot X_i), \quad \text{where} \quad \sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x}$$

- $\sigma(x)$ is called **sigmoid function**;
- w is a vector in \mathbb{R}^m , and is called **weight vector**;
- w is initialized randomly, but will improve as training goes.



During training, we want to **minimize** the following function:

$$\mathcal{L}(w) = -\frac{1}{N} \sum_{i=1}^N [Y_i \log(F(X_i, w)) + (1 - Y_i) \log(1 - F(X_i, w))]$$

Gradient descent is an iterative optimization algorithm for finding the minimum of a function. How? Take step proportional to the negative of the gradient of the function at the current point.

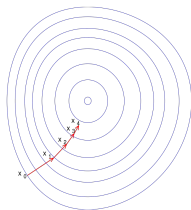


Figure 1: Gradient descent on a series of level sets

¹Credits for this slide: Andrea Palazzi https://github.com/ndrplz/deep_learning_lectures

If we consider a function $f(\boldsymbol{\theta})$, the **gradient descent update** can be expressed as:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} f(\boldsymbol{\theta}) \quad (1)$$

for each parameter θ_j .

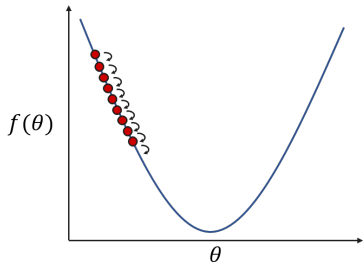
The size of the step is controlled by **learning rate** α .

¹Credits for this slide: Andrea Palazzi https://github.com/ndrplz/deep_learning_lectures

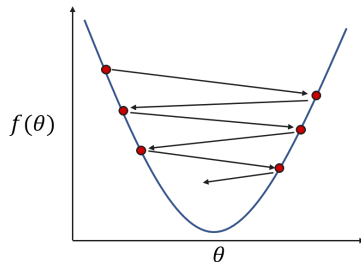
Gradient Descent for 1-d function $f(\theta)$.

¹Credits for this slide: Andrea Palazzi https://github.com/ndrplz/deep_learning_lectures

Choosing the the right **learning rate** α is essential to correctly proceed towards the minimum. A step *too small* could lead to an extremely *slow* convergence. If the step is *too big* the optimizer could *overshoot* the minimum or even *diverge*.



Learning Rate too small



Learning Rate too big

¹Credits for this slide: Andrea Palazzi https://github.com/ndrplz/deep_learning_lectures

Back to our problem. We need to take the derivative of this function w.r.t. w :

$$\begin{aligned}\mathcal{L}(w) &= -\frac{1}{N} \sum_{i=1}^N [Y_i \log(F(X_i, w)) + (1 - Y_i) \log(1 - F(X_i, w))] \\ &= -\frac{1}{N} \sum_{i=1}^N \left[Y_i \log \left(\frac{e^{w^T \cdot X_i}}{1 + e^{w^T \cdot X_i}} \right) + (1 - Y_i) \log \left(\frac{1}{1 + e^{w^T \cdot X_i}} \right) \right] \\ &= -\frac{1}{N} \sum_{i=1}^N \left[Y_i (w^T \cdot X_i) - Y_i \log \left(1 + e^{w^T \cdot X_i} \right) + (Y_i - 1) \log \left(1 + e^{w^T \cdot X_i} \right) \right] \\ &= -\frac{1}{N} \sum_{i=1}^N \left[Y_i (w^T \cdot X_i) - \log \left(1 + e^{w^T \cdot X_i} \right) \right]\end{aligned}$$

Back to our problem. We need to take the derivative of this function w.r.t. w :

$$\mathcal{L}(w) = -\frac{1}{N} \sum_{i=1}^N \left[Y_i (w^T \cdot X_i) - \log \left(1 + e^{w^T \cdot X_i} \right) \right]$$

$$\begin{aligned} \frac{\delta \mathcal{L}(w)}{\delta w_j} &= -\frac{1}{N} \sum_{i=1}^N \left[Y_i x_i^{(j)} - \frac{e^{w^T \cdot X_i}}{1 + e^{w^T \cdot X_i}} x_i^{(j)} \right] \\ &= -\frac{1}{N} \sum_{i=1}^N \left[Y_i - \frac{e^{w^T \cdot X_i}}{1 + e^{w^T \cdot X_i}} \right] x_i^{(j)} \\ &= -\frac{1}{N} \sum_{i=1}^N \left[Y_i - F(X_i, w) \right] x_i^{(j)} \end{aligned}$$

Back to our problem. We need to take the derivative of this function w.r.t. w :

$$\frac{\delta \mathcal{L}(w)}{\delta w} = \begin{bmatrix} \frac{\delta \mathcal{L}(w)}{\delta w_1} \\ \frac{\delta \mathcal{L}(w)}{\delta w_2} \\ \vdots \\ \frac{\delta \mathcal{L}(w)}{\delta w_m} \end{bmatrix} = \begin{bmatrix} -\frac{1}{N} \sum_{i=1}^N (Y_i - F(X_i, w)) x_i^{(1)} \\ -\frac{1}{N} \sum_{i=1}^N (Y_i - F(X_i, w)) x_i^{(2)} \\ \vdots \\ -\frac{1}{N} \sum_{i=1}^N (Y_i - F(X_i, w)) x_i^{(m)} \end{bmatrix} = -\frac{X^T \cdot (Y - F(X, w))}{N}$$

We will update the vector w accordingly:

$$w \leftarrow w - \alpha \frac{\delta \mathcal{L}(w)}{\delta w}$$

Algorithm 1 pseudocode for training

- 1: $X, Y \leftarrow \text{load_training_data}()$
 - 2: set learning rate α
 - 3: initialize w randomly
 - 4: **for** $e = 1$ to *number_of_training_steps* **do**
 - 5: compute the prediction according to the current weights $F(X, w)$
 - 6: compute the loss function $\mathcal{L}(w)$
 - 7: compute the derivative of the loss function w.r.t. weights $\frac{\delta \mathcal{L}(w)}{\delta w}$
 - 8: update the weight vector $w \leftarrow w - \alpha \frac{\delta \mathcal{L}(w)}{\delta w}$
 - 9: **end for**
-



- We want to predict if a character is alive or dead;
- (Some) of our features are:
 - male or female;
 - married or not;
 - number of deaths witnessed;
 - number of dead relatives;
 - ... and many more.