Chapter 2: Training Simple Machine Learning Algorithms for Classification ... 37
Artificial neurons – a brief glimpse into the early history of machine learning ... 38
The formal definition of an artificial neuron ... 40
The perceptron learning rule ... 42
Implementing a perceptron learning algorithm in Python ... 44
An object-oriented perceptron API ... 44
Training a perceptron model on the Iris dataset ... 46
Adaptive linear neurons and the convergence of learning ... 52
Minimizing loss functions with gradient descent ... 53
Implementing Adaline in Python ... 57
Improving gradient descent through feature scaling ... 62
Large scale machine learning and stochastic gradient descent ... 64
Summary ... 69

Minimizing loss functions with gradient descent

One of the key ingredients of supervised machine learning algorithms is a defined **objective function** that is to be optimized during the learning process. This objective function is often a loss or cost function that we want to minimize. In the case of Adaline, we can define the **loss function** L to learn the model parameters as the **mean squared error** (MSE) between the **target variable** y and the **net input** z:

$$L(\mathbf{w}, b) = \frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - a^{(i)}\right)^2$$

O que é o $z^{(i)}$ e o $y^{(i)}$?

$z = w_1 x_1 + \dots + w_m x_m + b = \langle \vec{w}^T, \vec{x} \rangle + b$

$\vec{w}^T = \begin{bmatrix} w_1 \\ \dots \\ w_m \end{bmatrix}$ → valor pesos do exercício

$\vec{x} = \begin{bmatrix} x^{(i)} \\ \dots \\ x_m \end{bmatrix}$ → o i é o valor de um exemplo do conjunto de exemplos do Iris de Dados de Treino usado (um 2 valores)

→ Sai porque tem valores $\frac{\partial L}{\partial w_j}$ (vai SOMAR)

The Iris dataset, consisting of 150 examples and four features, can then be written as a 150×4 matrix, formally denoted as $\mathbf{X} \in \mathbb{R}^{150\times 4}$:

$$\begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} \\ \dots & \dots & \dots & \dots \\ x_1^{(150)} & x_2^{(150)} & x_3^{(150)} & x_4^{(150)} \end{bmatrix}$$

Notational conventions:

But most parts of this book, unless noted otherwise, we will use the superscript i to refer to the i^{th} **training example**, and the subscript j to refer to the j^{th} **dimension** of the training dataset.

We will use lowercase, bold-face letters to refer to vectors ($\mathbf{x} \in \mathbb{R}^{m\times 1}$) and uppercase, bold-face letters to refer to matrices ($\mathbf{X} \in \mathbb{R}^{n\times m}$). Accordingly, to access a scalar or matrix, we will write the letters in italics ($x^{(i)}$ or $x_j^{(i)}$, respectively).

For example, $x_1^{(150)}$ refers to the first dimension of flower example 150, the sepal length. Each row in matrix $\mathbf{X}$ represents one flower example and can be written as a four-dimensional row vector, $\mathbf{x}^{(i)} \in \mathbb{R}^{1\times 4}$:

$$\mathbf{x}^{(i)} = \begin{bmatrix} x_1^{(i)} & x_2^{(i)} & x_3^{(i)} & x_4^{(i)} \end{bmatrix}$$

And each feature dimension is a 150-dimensional column vector, $\mathbf{x}_j \in \mathbb{R}^{n\times 1}$. For example:

$$\mathbf{x}_1 = \begin{bmatrix} x_1^{(1)} \\ \dots \\ x_1^{(150)} \end{bmatrix}$$

Similarly, we can represent the **target variables** (here, class labels) as a 150-dimensional column vector

$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ \dots \\ y^{(150)} \end{bmatrix} \text{ where } y^{(i)} \in \{\text{Setosa, Versicolor, Virginica}\}$$

Minimizing loss functions with gradient descent

One of the key ingredients of supervised machine learning algorithms is a defined **objective function** that is to be optimized during the learning process. This objective function is often a loss or cost function that we want to minimize. In the case of Adaline, we can define the **loss function** L to learn the model parameters as the **mean squared error** (MSE) between the **target variable** and the **net input**:

$$L(\mathbf{w}, b) = \frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - a^{(i)}\right)^2$$

Tem de ser diferenciável!

Using gradient descent, we can now update the model parameters by taking a step in the opposite **direction** of the gradient, $\nabla L(\mathbf{w}, b)$, of our loss function, $L(\mathbf{w}, b)$:

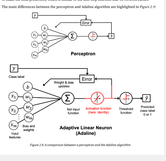
$$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}, \quad b := b + \Delta b$$

Linear Activation Function

The key difference between the Adaline rule (also known as the **Widrow-Hoff rule**) and Rosenblatt's perceptron is that the weights are updated based on a linear **activation function** rather than a unit step function like in the perceptron. In Adaline, this linear activation function, $\sigma(z)$, is simply the identity function of the net input, so that $\sigma(z) = z$.

While the linear activation function is used for learning the weights, we still use a threshold function to make the final prediction, which is similar to the unit step function that we covered earlier.

The main differences between the perceptron and Adaline algorithm are highlighted in Figure 2.9:



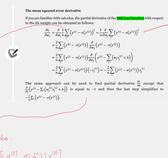


Figure 2.9: A comparison between a perceptron and the Adaline algorithm

The mean squared error derivative

If you are familiar with calculus, the partial derivative of the **loss function** with respect to the jth **weight** can be obtained as follows:

$$\begin{aligned}\frac{\partial L}{\partial w_j} &= \frac{1}{2n}\sum_i 2\left(y^{(i)} - a^{(i)}\right)\frac{\partial}{\partial w_j}\left(y^{(i)} - a^{(i)}\right) \\ &= \frac{1}{n}\sum_i \left(y^{(i)} - a^{(i)}\right)\frac{\partial}{\partial w_j}\left(y^{(i)} - \sum_j w_j x_j^{(i)}\right) \\ &= \frac{1}{n}\sum_i \left(y^{(i)} - a^{(i)}\right)\left(-x_j^{(i)}\right) = -\frac{1}{n}\sum_i \left(y^{(i)} - a^{(i)}\right)x_j^{(i)}\end{aligned}$$

The same approach can be used to find partial derivative $\frac{\partial L}{\partial b}$ except that $\frac{\partial}{\partial b}\left(y^{(i)} - \sum_j w_j x_j^{(i)}\right)$ is equal to -1 and thus the last step simplifies to $-\frac{1}{n}\sum_i\left(y^{(i)} - a^{(i)}\right)$.

A qual de nós é $a^{(i)}$?

Nós não temos $\frac{\partial L}{\partial a^{(i)}}$

$L(\vec{w}, b) = \frac{1}{2n}\sum_{i=1}^{n}\left(y^{(i)} - a^{(i)}\right)^2$

Passo 1 - Derivada Parcial

$\frac{\partial L}{\partial w_j} = ?$

Regra da Cadeia

$\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial a^{(i)}} \cdot \frac{\partial a^{(i)}}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial w_j}$

$\frac{\partial L}{\partial a^{(i)}}$ → Derivada da Função Loss

$\frac{\partial a^{(i)}}{\partial z^{(i)}}$ → Derivada da Função de Ativação

$\frac{\partial z^{(i)}}{\partial w_j}$ → Derivada da Função Linear

$\frac{\partial L}{\partial w_j} = -\frac{2}{n}\sum_i \left[\left(y^{(i)} - \sigma(z^{(i)})\right)x_j^{(i)}\right]$

$\frac{\partial L}{\partial b} = -\frac{2}{n}\sum_i \left(y^{(i)} - \sigma(z^{(i)})\right)$

$L(\vec{w}, b) = \frac{1}{2n}\sum_{i=1}^{n}\left[\left(y^{(i)} - a^{(i)}\right)^2 + \left(y^{(i)} - a^{(i)}\right)^2 + \left(y^{(i)} - a^{(i)}\right)^2\right]$

Uma Rede Neural com 3 neurónios. Cada neurónio tem a sua distribuição, mas recebem todos o mesmo x. Assim o $z = (\vec{w}^T, \vec{x}) + b$ é o mesmo

$z = w_1 x_1 + w_2 x_2 + b$

Ou seja, o w_1 aparece na função Loss $L(\vec{w}, b)$ três vezes, uma vez em cada neurónio

$L(\vec{w}, b) = \frac{1}{2n}\left[\left(y^{(1)} - a^{(1)}\right)^2 + \left(y^{(2)} - a^{(2)}\right)^2 + \left(y^{(3)} - a^{(3)}\right)^2\right]$

Aparece aqui!

Derivada Parcial → $\frac{\partial}{\partial w_1}\left[\frac{1}{2n}\left[\left(y^{(1)} - a^{(1)}\right)^2 + \left(y^{(2)} - a^{(2)}\right)^2 + \left(y^{(3)} - a^{(3)}\right)^2\right]\right]$

Lembrar Regra da Cadeia da Derivada

$g(h(x)) \rightarrow \frac{d}{dx} = \left(\frac{dg}{dh}\right)\left(\frac{dh}{dx}\right)$

The Chain Rule If g is differentiable at c and f is differentiable at $g(c)$, then the composite function $F = f \circ g$ defined by $F(x) = f(g(x))$ is differentiable at c and F' is given by the product

$$F'(c) = f'(g(c)) \cdot g'(c)$$

In Leibniz notation, if $y = f(u)$ and $u = g(x)$ are both differentiable functions, then

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Lembrando que o w_1 aparece no $z^{(1)}$, no $z^{(2)}$ e no $z^{(3)}$

Vamos passar a ter somente o $z^{(i)}$

$\frac{\partial}{\partial z}\left[\frac{1}{2n}\left(y^{(i)} - z^{(i)}\right)^2\right] =$

$\frac{\partial}{\partial z}\left[\frac{1}{2n}\left(y^{(i)2} - 2y^{(i)}z^{(i)} + z^{(i)2}\right)\right]$

$\frac{\partial}{\partial z} = \frac{-1}{n}y^{(i)} + \frac{1}{n}z^{(i)}$

E agora vamos fazer $\frac{\partial z^{(i)}}{\partial w_1}$

$\frac{\partial}{\partial w_1}\left(w_1 x_1 + w_2 x_2 + b\right) = x_1$

Logo portanto tudo $\frac{\partial}{\partial w_1}\left[\frac{1}{2n}\left[\left(y^{(1)} - a^{(1)}\right)^2 + \left(y^{(2)} - a^{(2)}\right)^2 + \left(y^{(3)} - a^{(3)}\right)^2\right]\right] = \left(-\frac{1}{n}y^{(1)} + \frac{1}{n}z^{(1)}\right)(x_1^{(1)}) = -\frac{1}{n}y^{(1)}x_1^{(1)} + \frac{1}{n}z^{(1)}x_1^{(1)}$

$L(\vec{w}, b) = \frac{1}{2n}\left[\left(y^{(1)} - a^{(1)}\right)^2 + \left(y^{(2)} - a^{(2)}\right)^2 + \left(y^{(3)} - a^{(3)}\right)^2\right]$

$\frac{\partial}{\partial w_1}\left[\frac{1}{2n}\left[\left(y^{(1)} - z^{(1)}\right)^2 + \left(y^{(2)} - z^{(2)}\right)^2 + \left(y^{(3)} - z^{(3)}\right)^2\right]\right]$

$\frac{\partial}{\partial w_1}\left[\frac{1}{2n}\left(y^{(1)} - z^{(1)}\right)^2 + \frac{1}{2n}\left(y^{(2)} - z^{(2)}\right)^2 + \frac{1}{2n}\left(y^{(3)} - z^{(3)}\right)^2\right]$

$\frac{\partial}{\partial w_1}\left(\frac{1}{2n}\left(y^{(1)} - z^{(1)}\right)^2\right) + \frac{\partial}{\partial w_1}\left(\frac{1}{2n}\left(y^{(2)} - z^{(2)}\right)^2\right) + \frac{\partial}{\partial w_1}\left(\frac{1}{2n}\left(y^{(3)} - z^{(3)}\right)^2\right)$

$\left[-\frac{1}{n}y_1^{(1)}x_1^{(1)} + \frac{1}{n}z^{(1)}x_1^{(1)}\right] + \left[-\frac{1}{n}y_2^{(2)}x_1^{(2)} + \frac{1}{n}z^{(2)}x_1^{(2)}\right] + \left[-\frac{1}{n}y_3^{(3)}x_1^{(3)} + \frac{1}{n}z^{(3)}x_1^{(3)}\right]$

Função Loss

$\frac{\partial L}{\partial w_1}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{3}\left(z^{(i)}x_1^{(i)} - y_1^{(i)}x_1^{(i)}\right)$

$\frac{\partial L}{\partial b}(\vec{w}) = \left(\frac{\partial L}{\partial z}\right)\left(\frac{\partial z}{\partial b}\right) = \frac{1}{n}\sum_{i=1}^{3}\left(z^{(i)} - y_1^{(i)}\right)$

$L(\vec{w}, b) = \frac{1}{2n}\left[\left(y^{(1)} - a^{(1)}\right)^2 + \left(y^{(2)} - a^{(2)}\right)^2 + \left(y^{(3)} - a^{(3)}\right)^2\right]$ Lembrando que $\frac{\partial z}{\partial b} = 1$

Aparece aqui!

COMPARANDO OS RESULTADOS

Meu Resultado

$\frac{\partial L}{\partial w_j} = \frac{1}{n}\sum_{i=1}^{3}\left(z^{(i)}x_j^{(i)} - y^{(i)}x_j^{(i)}\right)$

$\frac{\partial L}{\partial b} = \frac{1}{n}\sum_{i=1}^{3}\left(z^{(i)} - y^{(i)}\right)$

Raschka

Passo-a-passo:

$\frac{\partial L}{\partial w_j} = -\frac{2}{n}\sum_i\left[\left(y^{(i)} - \sigma(z^{(i)})\right)x_j^{(i)}\right]$

$\frac{\partial L}{\partial b} = -\frac{2}{n}\sum_i\left(y^{(i)} - \sigma(z^{(i)})\right)$

Tem que lembrar que o gradiente da Função Loss $L(\vec{w}, b)$ aponta pr a direção de maior crescimento da função!

Como o objetivo é minimizar a função Loss o algoritmo iterativo de minimização tem que andar na direção oposta ao do gradiente!

Although the Adaline learning rule looks identical to the perceptron rule, we should note that $\sigma(z^{(i)})$ with $z^{(i)} = \mathbf{w}^T\mathbf{x}^{(i)} + b$ is a real number and not an integer class label. Furthermore, the weight update is calculated based on all examples in the training dataset (instead of updating the parameters incrementally after each training example), which is why this approach is also referred to as **batch gradient descent**. To be more explicit and avoid confusion when talking about related concepts later in this chapter and this book, we will refer to this process as **full batch gradient descent**.