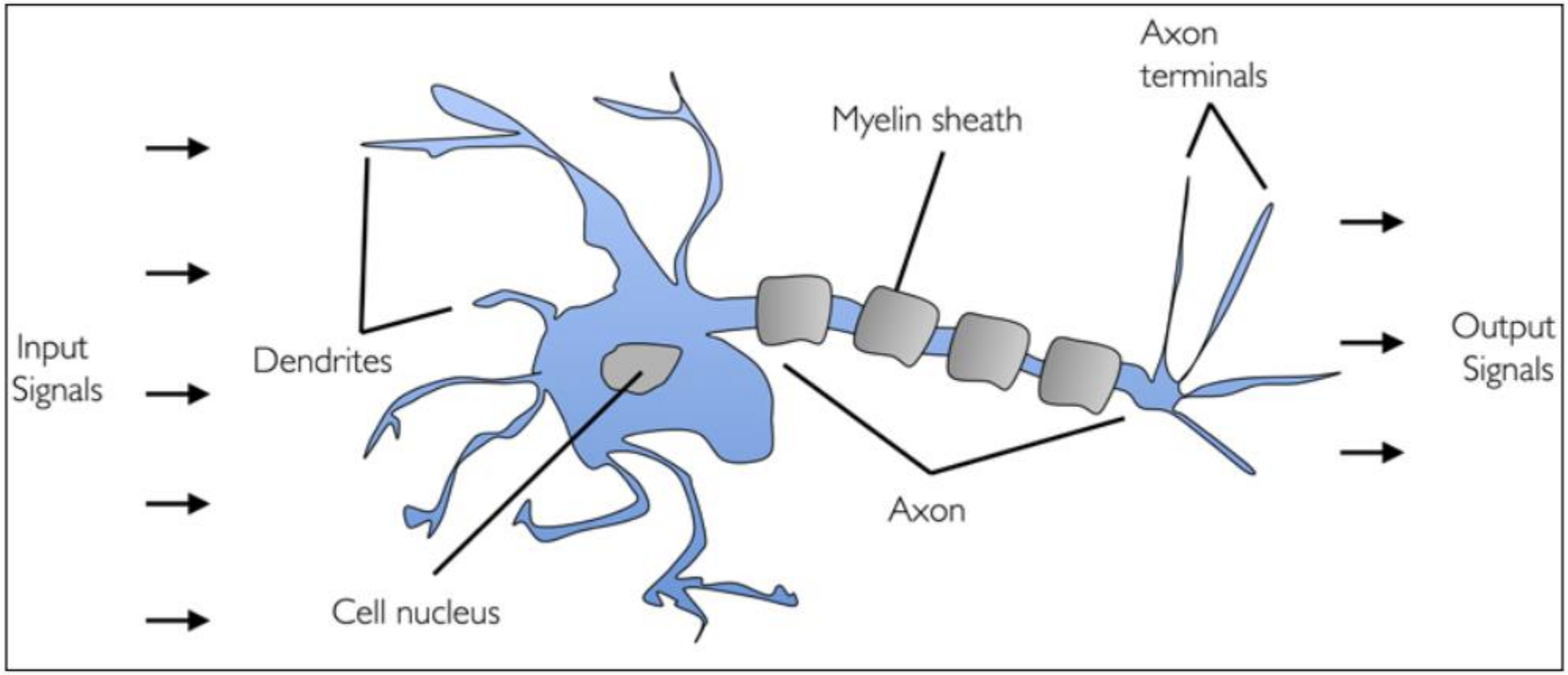
Lecture 2 - Training Simple Machine Learning Algorithms

Andre E. Lazzaretti

Introduction

- Two of the first algorithmically described machine learning algorithms for classification: the perceptron and adaptive linear neurons.
- Perceptron step by step in Python and training it to classify different flower species in the Iris dataset.
- Discussing the basics of optimization using adaptive linear neurons
- Additionally:
  - Building an intuition for machine learning algorithms
  - Using pandas, NumPy, and Matplotlib to read in, process, and visualize data.

Artificial Neurons

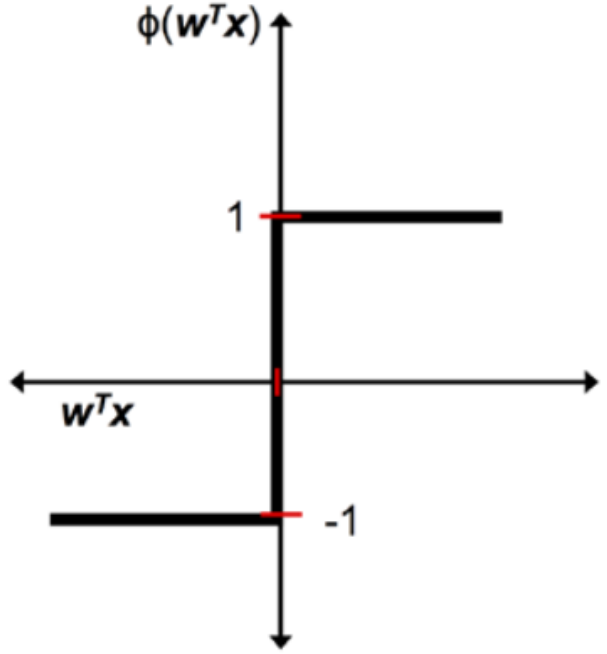


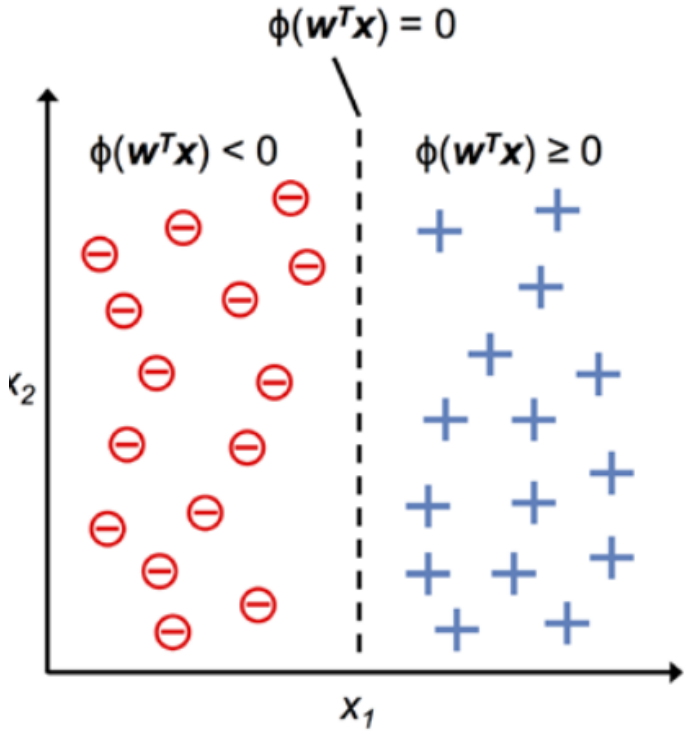
Formal Definitions

$$\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_m x_m = \sum_{j=0}^{m} \boldsymbol{x}_j \boldsymbol{w}_j = \boldsymbol{w}^T \boldsymbol{x}$$

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{otherwise} \end{cases}$$





Perceptron Learning Rule

1. Initialize the weights to 0 or small random numbers.
2. For each training sample $\boldsymbol{x}^{(i)}$:
   a. Compute the output value $\hat{y}$.
   b. Update the weights.

$$w_j := w_j + \Delta w_j$$

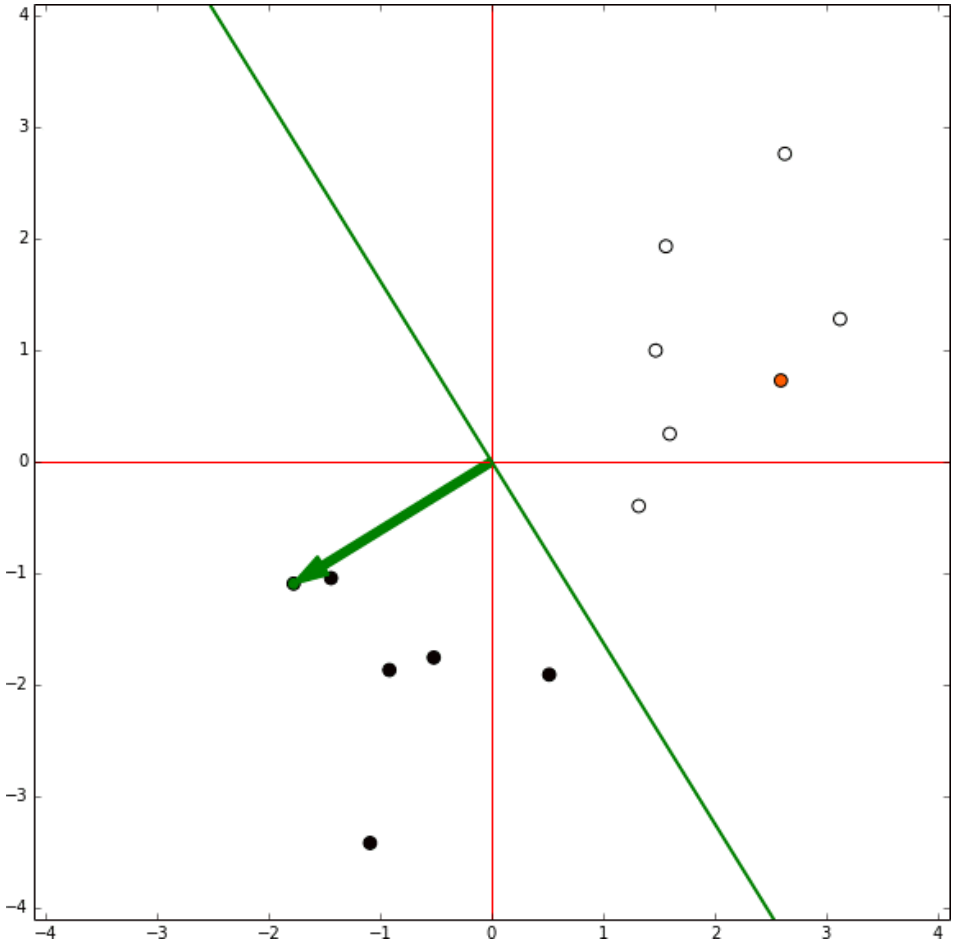
$$\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$$

$$\Delta w_0 = \eta\left(y^{(i)} - output^{(i)}\right)$$

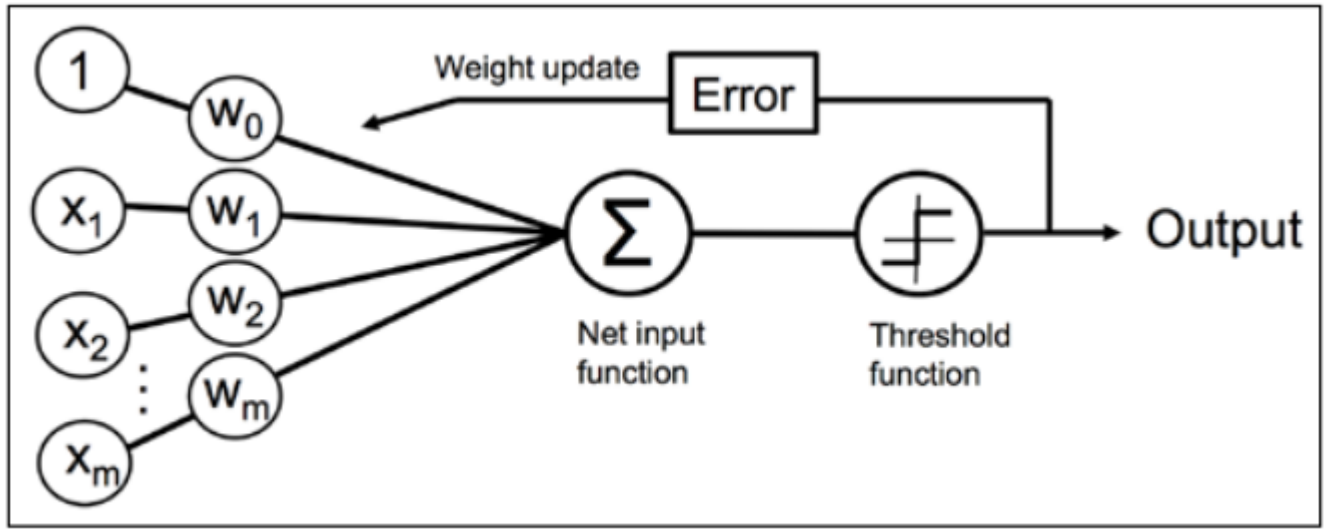
two-dimensional dataset

$$\Delta w_1 = \eta\left(y^{(i)} - output^{(i)}\right)x_1^{(i)}$$

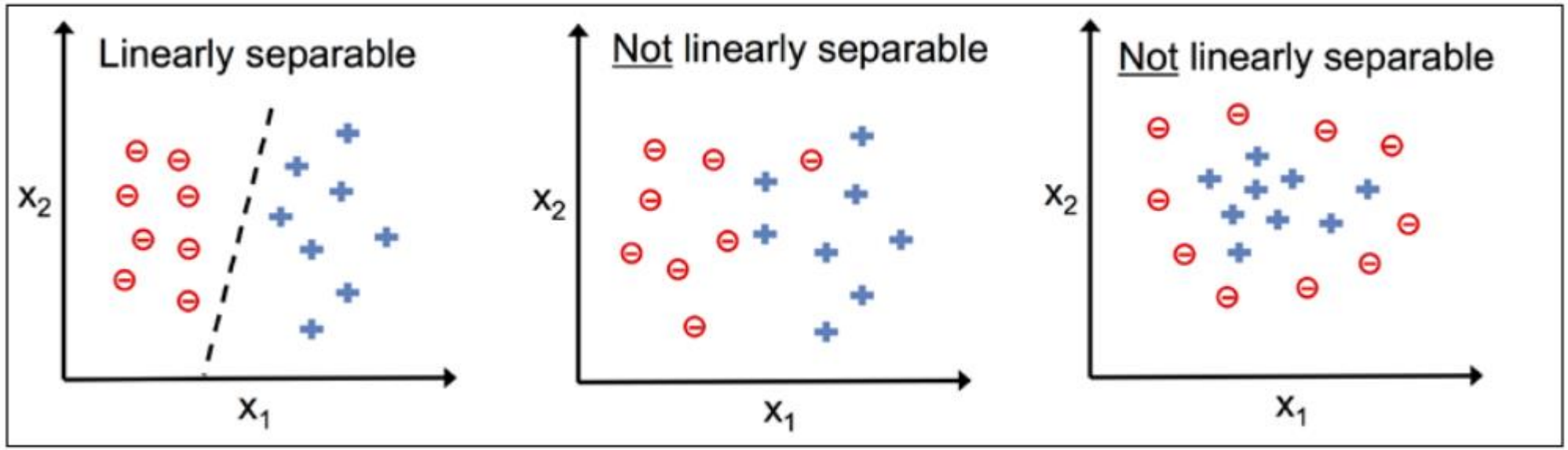
$$\Delta w_2 = \eta\left(y^{(i)} - output^{(i)}\right)x_2^{(i)}$$



Perceptron Model



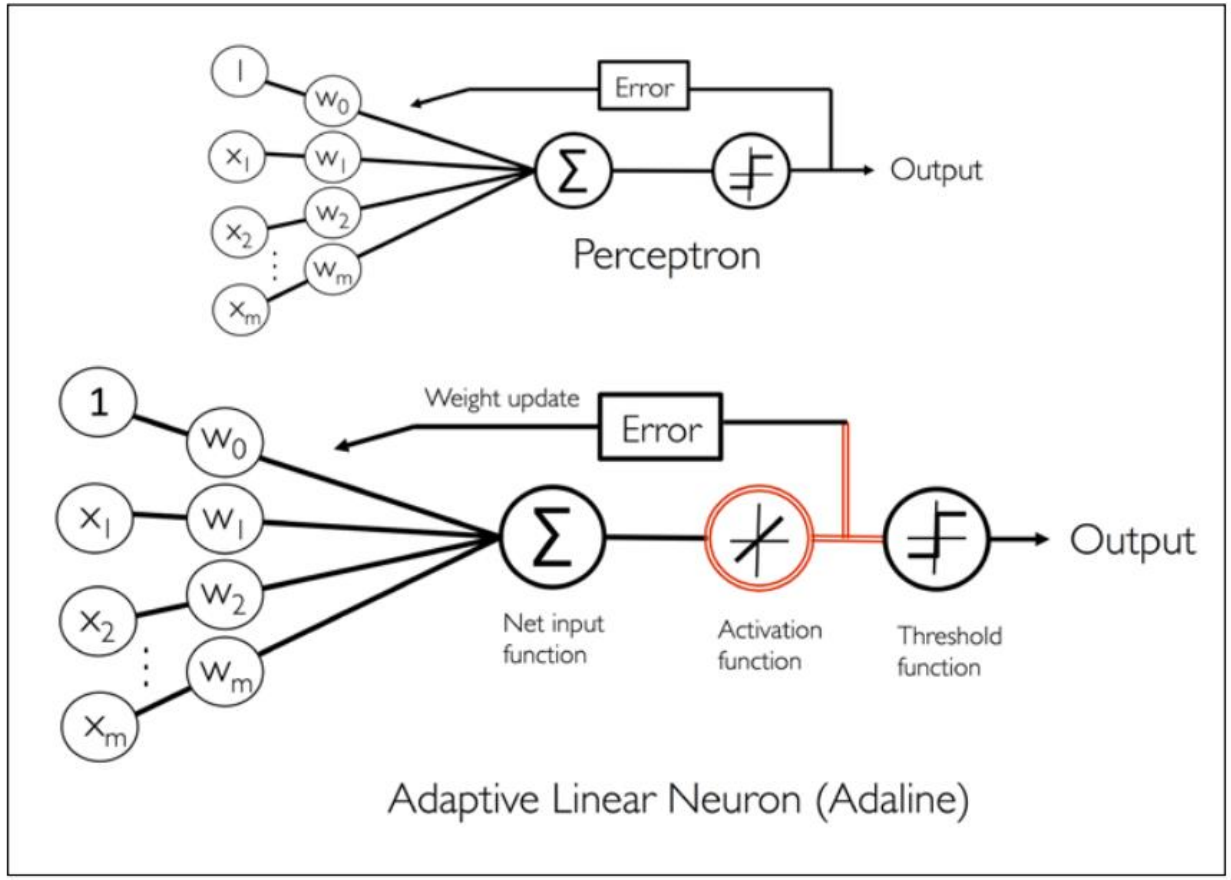
Linearly Separable



Perceptron Algorithm in Python

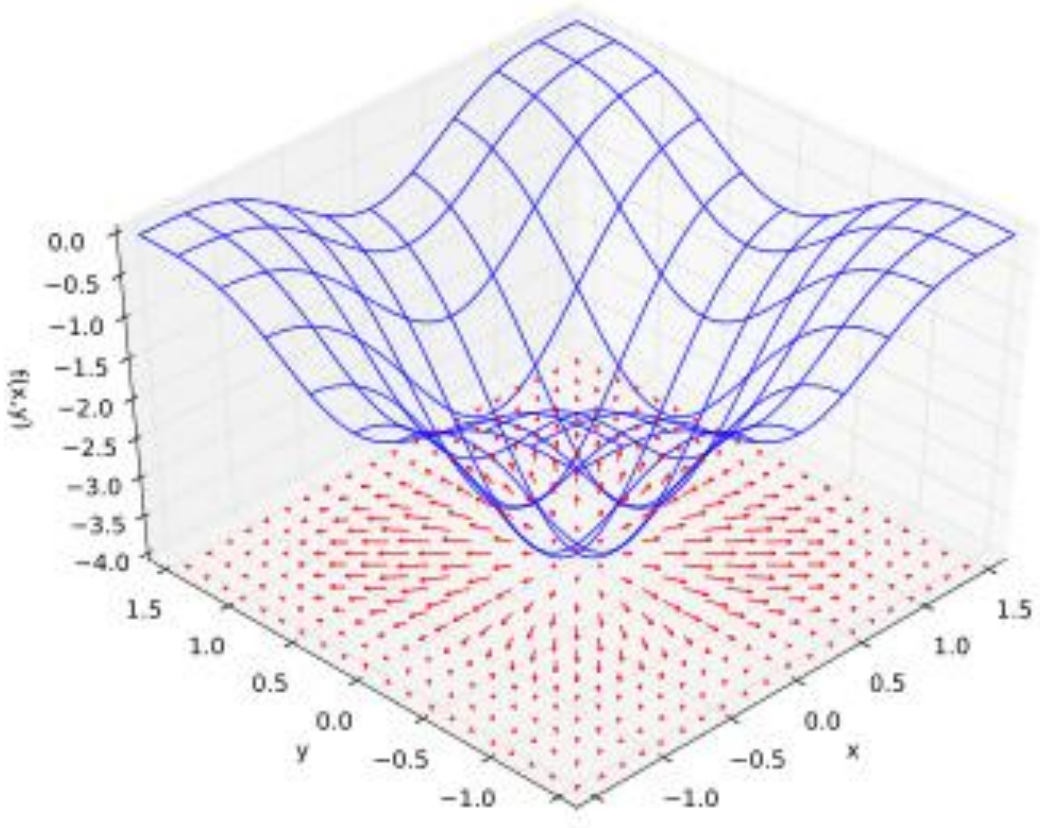
- Code for perceptron
- IRIS dataset
- Visualizing results

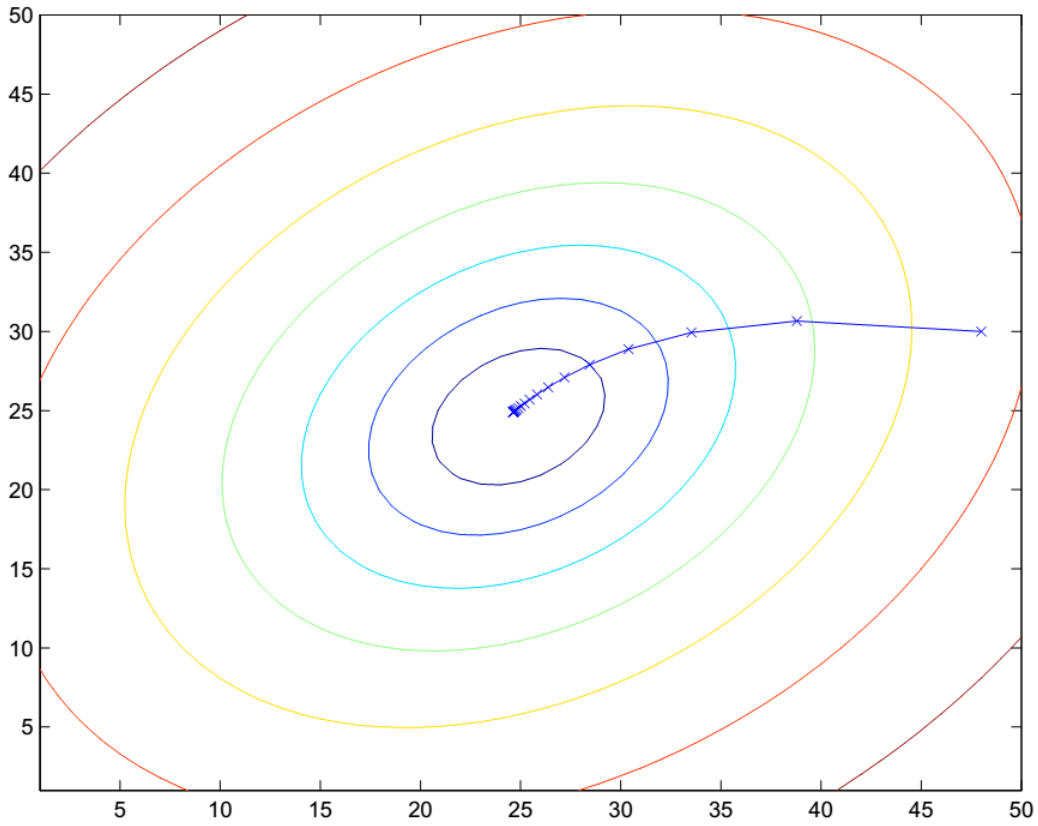
ADAptive LInear Neurons (ADALINE)

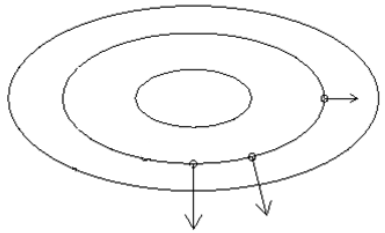


$$J(\boldsymbol{w}) = \frac{1}{2}\sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right)\right)^2$$

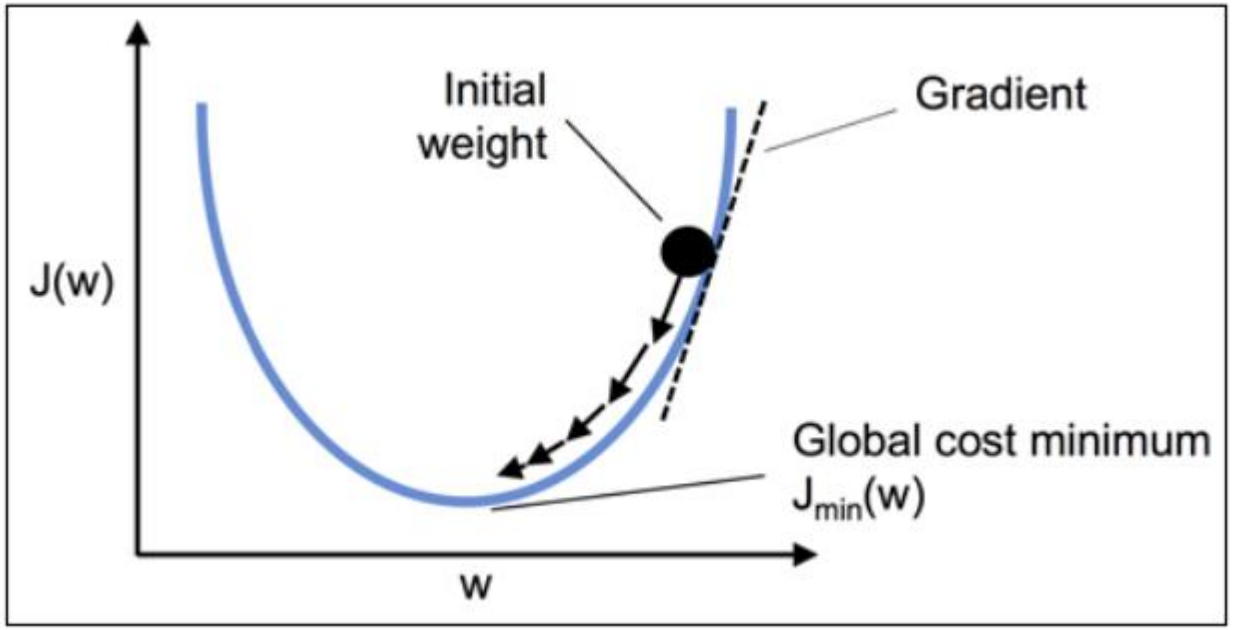
Gradient Descent







Gradient Descent



$$\boldsymbol{w} := \boldsymbol{w} + \Delta \boldsymbol{w}$$

$$\Delta \boldsymbol{w} = -\eta \nabla \mathrm{J}(\boldsymbol{w})$$

$$\frac{\partial J}{\partial w_j} = -\sum_i \left(y^{(i)} - \phi\left(z^{(i)} \right) \right) x_j^{(i)}$$

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j} = \eta \sum_i \left(y^{(i)} - \phi\left(z^{(i)} \right) \right) x_j^{(i)}$$

Proof

$$\frac{\partial J}{\partial w_j} = \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right) \right)^2$$

$$= \frac{1}{2} \frac{\partial}{\partial w_j} \sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right) \right)^2$$

$$= \frac{1}{2} \sum_i 2 \left(y^{(i)} - \phi\left(z^{(i)}\right) \right) \frac{\partial}{\partial w_j} \left(y^{(i)} - \phi\left(z^{(i)}\right) \right)$$

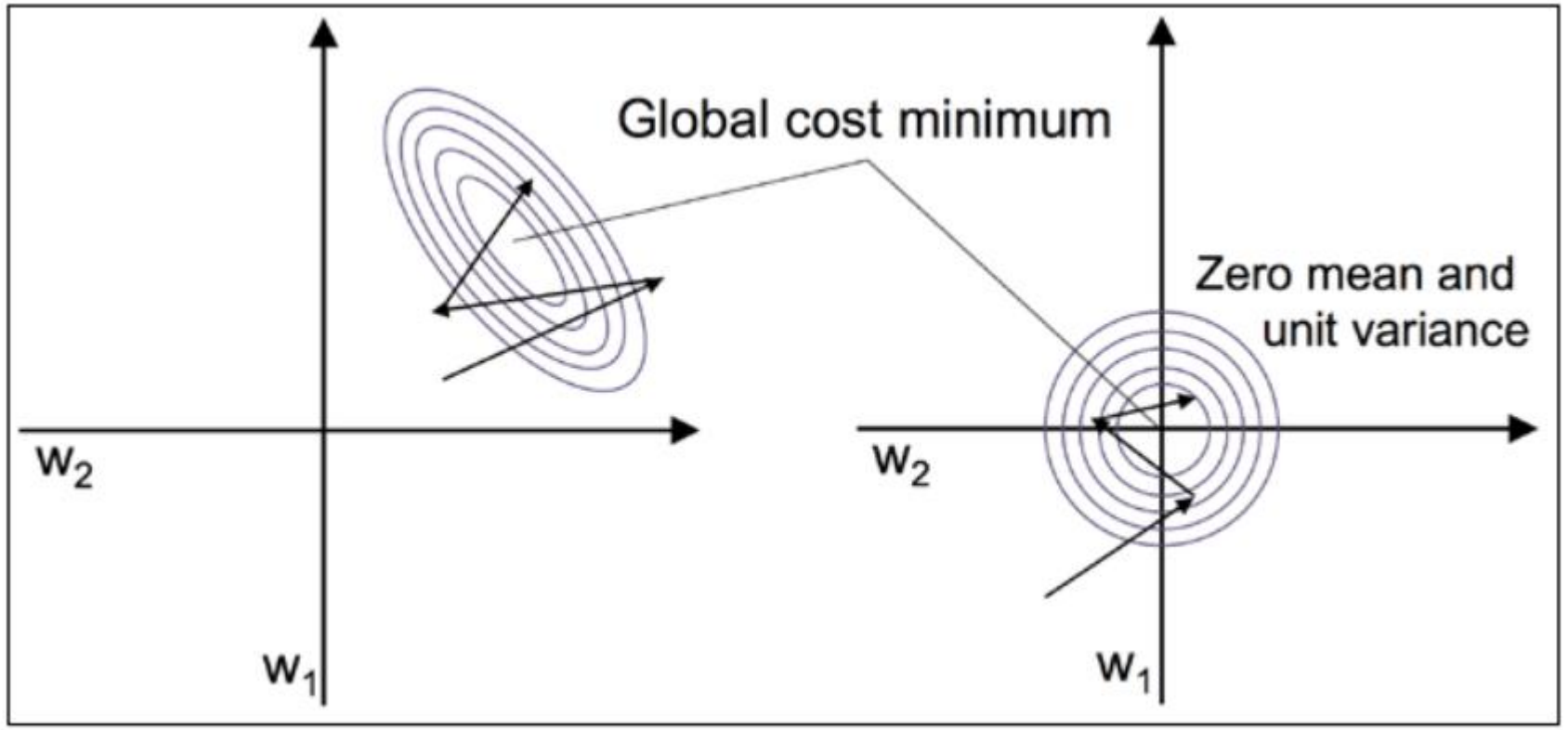
$$= \sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right) \right) \frac{\partial}{\partial w_j} \left(y^{(i)} - \sum_i \left(w_j^{(i)} x_j^{(i)} \right) \right)$$

$$= \sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right) \right) \left(-x_j^{(i)} \right)$$

$$= -\sum_i \left(y^{(i)} - \phi\left(z^{(i)}\right) \right) x_j^{(i)}$$

Feature Scaling

$$x'_j = \frac{x_j - \mu_j}{\sigma_j}$$



ADALINE in Python

- Code for ADALINE
- Feature scaling

Stochastic gradient descent

$$\Delta \boldsymbol{w} = \eta \sum_i \left(y^{(i)} - \phi\left(z^{(i)} \right) \right) \boldsymbol{x}^{(i)}$$

We update the weights incrementally for each training sample:

$$\eta \left(y^{(i)} - \phi\left(z^{(i)} \right) \right) \boldsymbol{x}^{(i)}$$

In stochastic gradient descent implementations, the fixed learning rate η is often replaced by an adaptive learning rate that decreases over time, for example:

$$\frac{c_1}{[number\ of\ iterations] + c_2}$$

Where c_1 and c_2 are constants. We shall note that stochastic gradient descent does not reach the global minimum, but an area very close to it. And using an adaptive learning rate, we can achieve further annealing to the cost minimum.

Stochastic gradient descent in Python

- Code for large-scale machine learning.