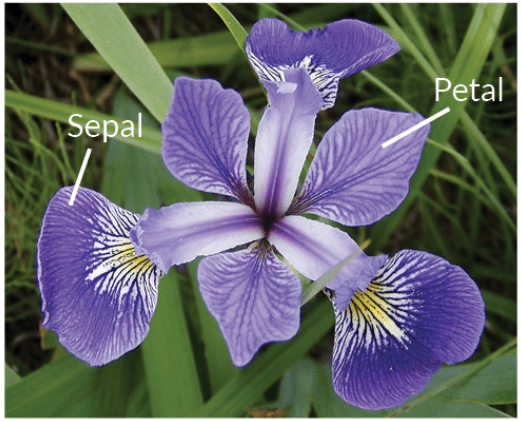
The Iris Classification Problem

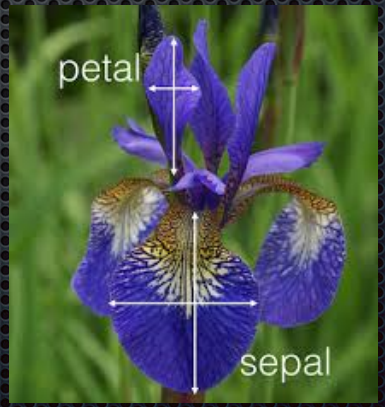


Iris Versicolor

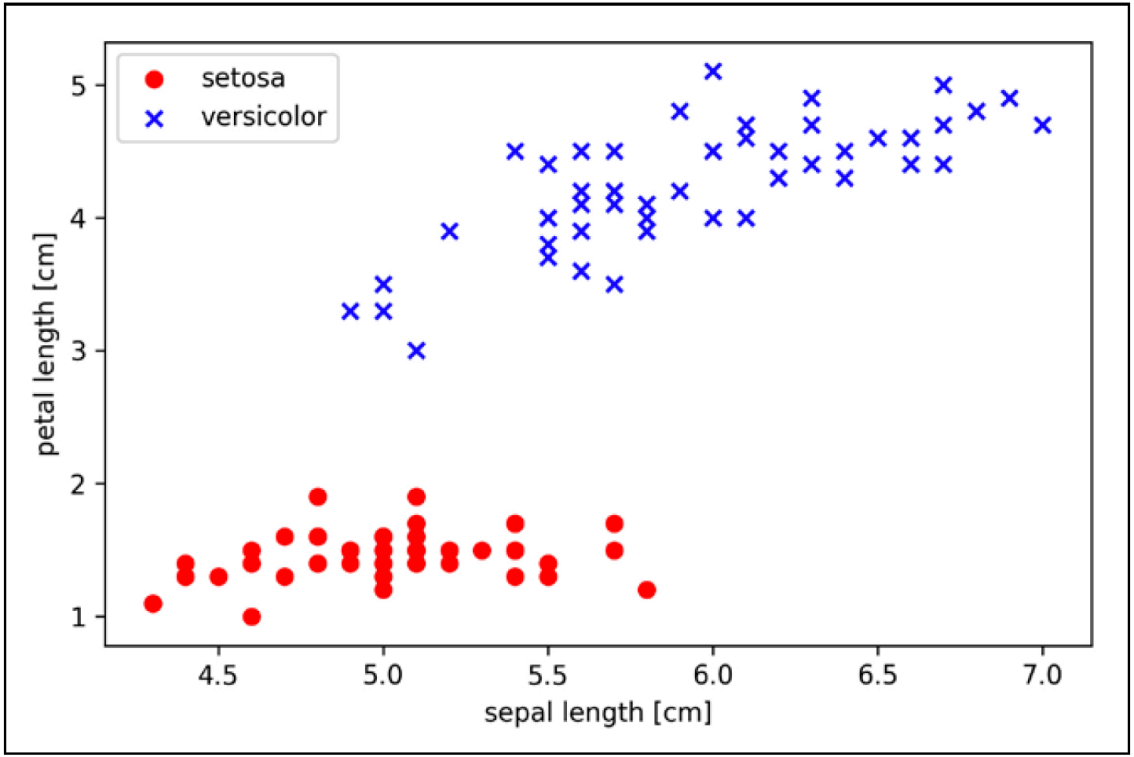
Iris Setosa

Iris Virginica

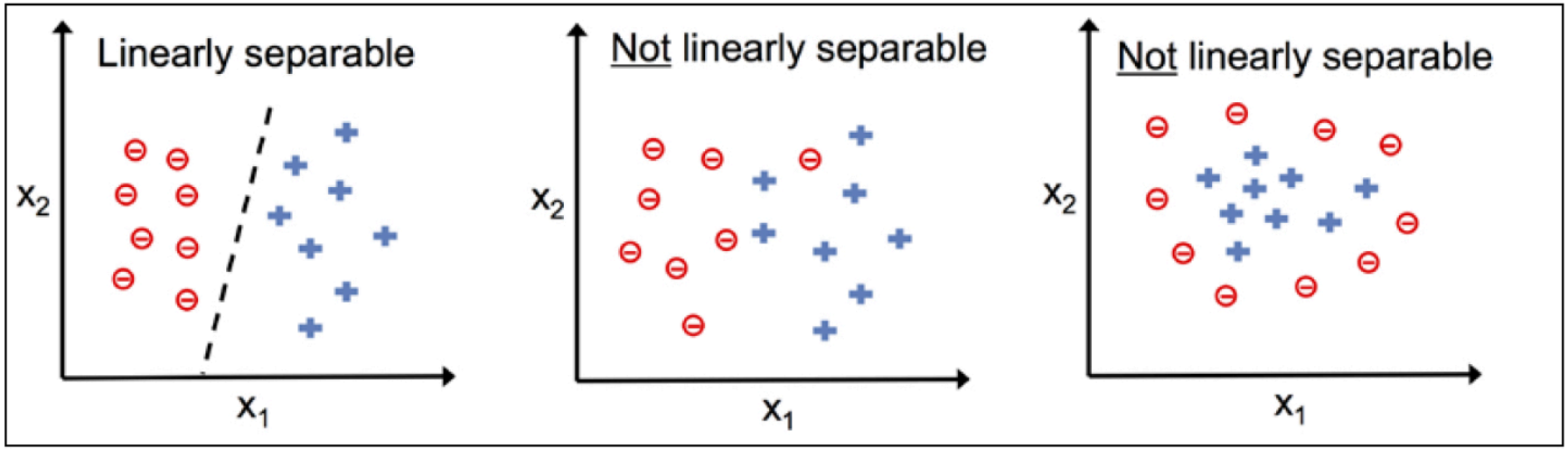
Data: petal width/length
sepal width/length



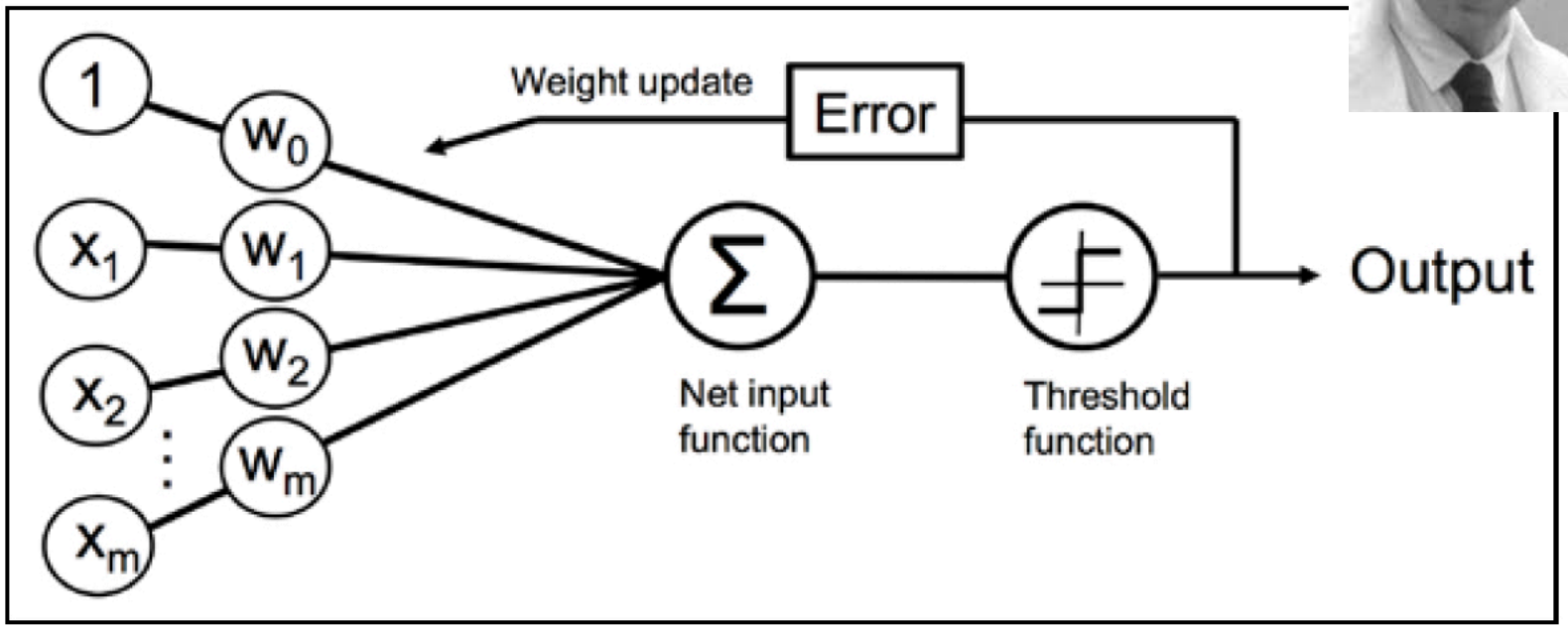
Classification problem



Linearly vs Not linearly separable



Perceptron (Rosenblatt, 1957)



A. Perceptron (Rosenblatt, 1957)

Firing of neuron is binary classification task. Use step function as activation function $\phi(z)$, i.e.

$$\phi(z) = \begin{cases} 1 & z \geq \theta \\ -1 & z < \theta \end{cases}$$

where

$$z = \sum_{j=1}^{m} w_j x_j$$

$x_1, \ldots, x_m$: input data

$w_1, \ldots, w_m$: weights

and suppose there are N data samples $(i = 1, \ldots, N)$

Define $\begin{cases} w_0 = -\theta \\ x_0 = 1 \end{cases}$ (bias)

then $z = \sum_{j=0}^{m} w_j x_j = -\theta + \tilde{z}$

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

Perceptron algorithm:

- initialise weights randomly $\in (0, 1)$
- for each training sample $(x_1^i, \ldots x_m^i)$ with true output $y^i \in \{-1, 1\}$

1. compute: $\hat{y}^i = \phi(z^i)$
2. update: $\hat{w}_j = w_j + \Delta w_j$

$$\Delta w_j = \eta \left(y^i - \hat{y}^i \right) x_j^i \qquad j = 1, \ldots, m$$

η : learning rate

Note : all weights updated simultaneously for each i

Cases : 1. $\hat{y}^i - y^i = 0$ correct prediction of output

$\longrightarrow \Delta w_j = 0$

2. $\hat{y}^i - y^i \neq 0$ incorrect

a. $\hat{y}^i = 1 \;,\; y^i = -1$

$\longrightarrow \Delta w_j = -2\eta x_j^i \longrightarrow$ move w_j (−)

b. $\hat{y}^i = -1 \;,\; y^i = 1$

$\longrightarrow \Delta w_j = 2\eta x_j^i \longrightarrow$ move w_j (+)

2b. assume $\eta = 1$ & $x_j^i = 1/2$

(sample misclassified with $\hat{y}^i = -1$ instead of 1.)

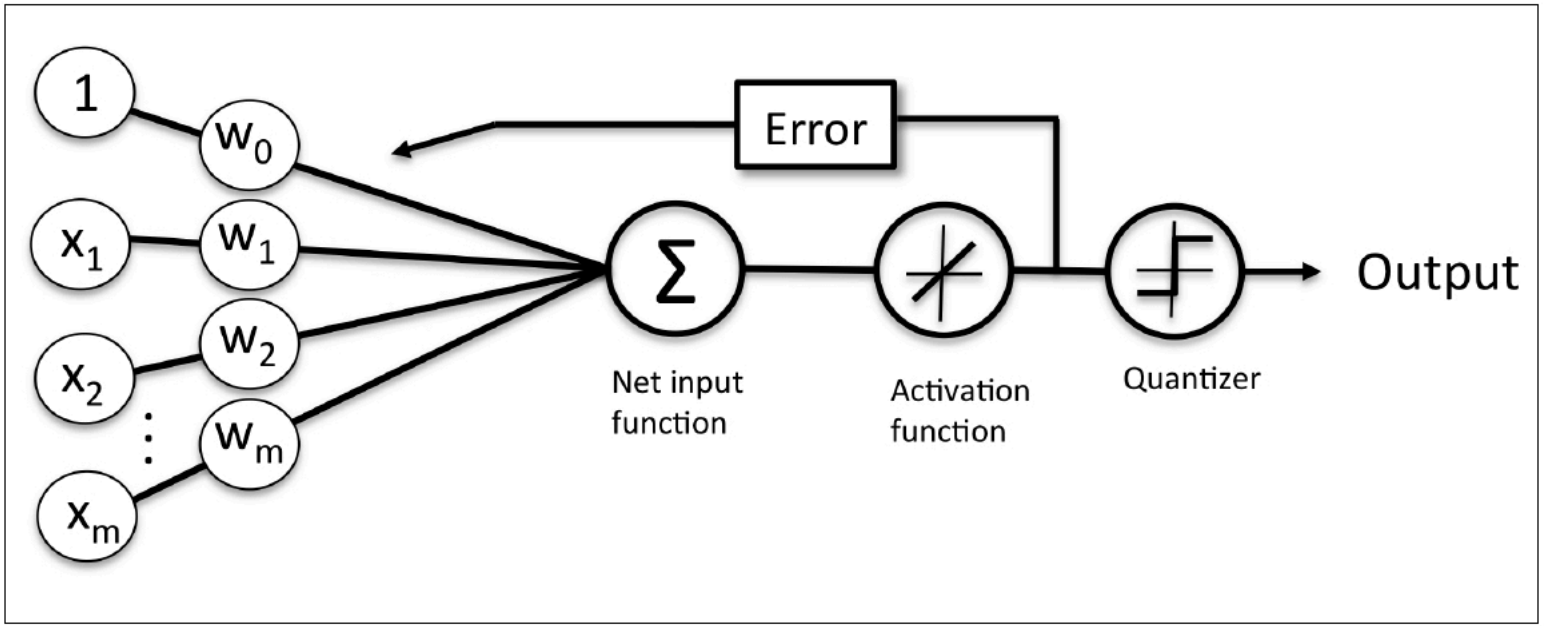
$\rightarrow \Delta w_j = 1$

pushed such that z is larger, so that it is more likely that $z^i \geq 0$ and $\hat{y}^i = \phi(z^i) = +1$

Remarks :- the output is also called a class label

- classes should be seperable by a linear decision boundary.

Adaptive Linear Neuron (AdaLiNe)



B. Adaptive linear neuron

In the perceptron, the weight were updated based on the step function. In the Adaline, the weight are updated using the linear activation function

$$\phi(z) = z$$

where $$z = \sum_{j=0}^{m} w_j x_j$$

After this, a quantizer (e.g. a step function) can be used to predict the class labels, i.e.

$$q(z) = \begin{cases} 1 & z \geq 0 \\ -1 & z < 0 \end{cases}$$

where again the bias is taken into account through w_0 and x_0.

To determine the weights, we define a cost function

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

where the sum is over the samples i.

The mathematical problem to solve is to determine $\underline{w}_*$ as

$$\underline{w}_* = \min_{\underline{w}} J(\underline{w})$$

Mathematical problem

Input $\mathbf{x} = (x_0, \ldots, x_m)^T$

Weights $\mathbf{w} = (w_0, \ldots, w_m)^T$

data samples: i = 1,…,N

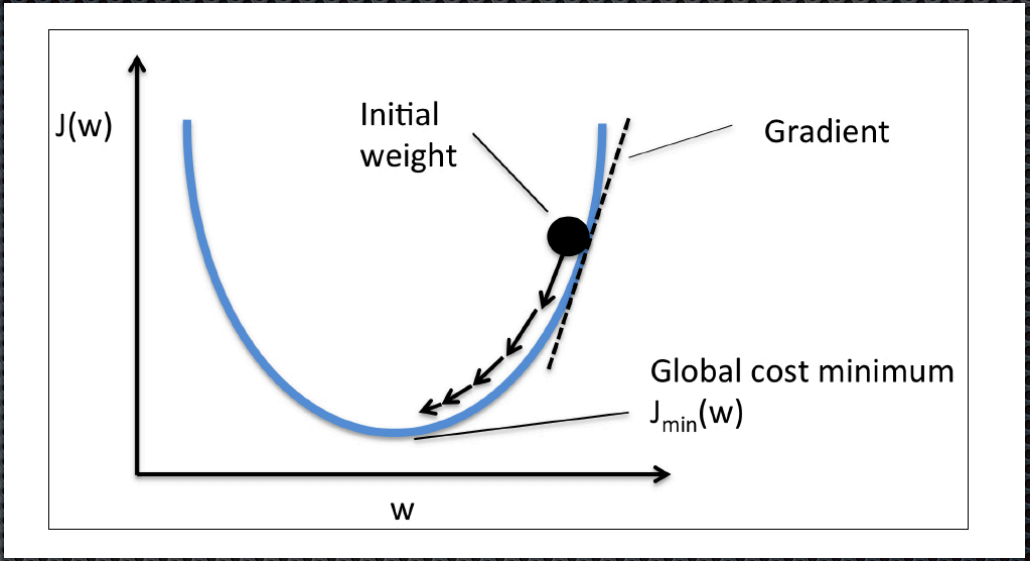
Activation: $\mathbf{z} = \mathbf{w}^T \mathbf{x}$ $\phi(\mathbf{z}) = \mathbf{z}$

Cost function: $J(\mathbf{w}) = \frac{1}{2} \sum_i (y^i - \phi(z^i))^2)$

Task: $min_{\mathbf{w}}\ J(\mathbf{w})$

Method: Gradient Descent

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \Delta\mathbf{w}_{k+1}$$



$$\Delta\mathbf{w}_{k+1} = -\eta\nabla J(\mathbf{w}_k)$$

Method 1. Gradient descend

iteration index k, $\underline{w}_0 = \underline{w}_{\text{initial}}$

then
$$\begin{cases} \underline{w}_{k+1} = \underline{w}_k + \Delta \underline{w}_{k+1} \\ \Delta \underline{w}_{k+1} = -\eta \, \nabla E(\underline{w}_k) \end{cases}$$

η: learning rate

$-\nabla E$: downgradient

Now $(\nabla E)_j$ can be computed from

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

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

use $\phi(z^i) = \sum_{j=0}^{m} x_j^i \, w_j$

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

Hence:

$$\left(\Delta \underline{w}_{k+1}\right)_j = \eta \sum_i \left(y^i - \phi(z^i) \right) x_j^i$$

Remarks: - convergence can be slow when small values of η are needed. In this case, it often helps to rescale the input data (with $\underline{x}_j = (x_j^1, \ldots, x_j^N)$)

$$\underline{x}_j^s = \frac{\underline{x}_j - \mu_j}{\sigma_j}$$

where $\mu_j = \frac{1}{N} \sum_{i=1}^{N} x_j^i$ (mean)

and $\sigma_j^2 = \frac{1}{N} \sum_{i=1}^{N} \left(x_j^i - \mu_j \right)^2$

Method 1: Stochastic gradient descend

Update of weights:

$$\left(\Delta \underline{w}_{k+1}\right)_j = \eta \left(y^i - \phi(z^i)\right) x_j^i$$

Advantages:
- cheaper
- more easily to hop over 'local' minima of Ξ.

Variants:
- use varying η
- shuffle training data for every iteration (epoch)

Exercise B2