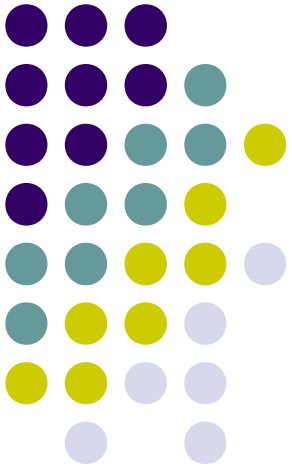
Pattern Recognition

Linear Discriminant Functions
Fisher's linear discriminant
The Perceptron

Francesco Tortorella

University of Cassino and Southern Latium

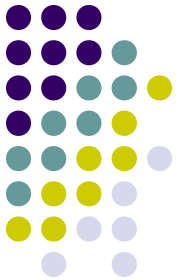
Cassino, Italy



DT-derived classifiers

- Two approaches considered till now for building a classifier, both derived from Decision Theory.
- Parametric approach:
  - MAP/Risk minimization+Gaussian densities
    - Linear Classifier
    - Quadratic Classifier
- Non parametric approach
  - MAP
    - K-NN

DT-derived classifiers



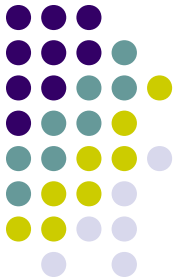
- With the linear and the quadratic classifiers we defined some discriminant functions with predefined expression.

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)+\ln P(\omega_i)$$

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^T\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i)-\frac{1}{2}\ln|\boldsymbol{\Sigma}_i|-\frac{d}{2}\ln 2\pi+\ln P(\omega_i)$$

- In both cases, we used a training set to estimate the parameters of the densities underlying the classifiers.
- Now, we use the training data to directly build the discriminant functions.

Discriminant functions

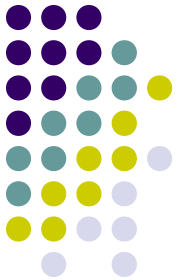


- In this case, probabilities play no role (in principle).
- Less demanding than the other approaches, but less information provided in the decision stage (no post probabilities).
- Tipically, the analytical form of the discriminant function is fixed thus the learning stage is devoted to estimate the parameters of the DF from the data.

Linear discriminant functions

- Let's consider discriminant functions linear in the components of the f.v. x (*linear discriminant functions*)
- They have some interesting properties:
  - LDF can be optimal if the underlying distributions of the data are similar
  - LDF are analitically tractable
  - LDF have low computational cost (good solution for building an initial version of classifier)
  - More complex classifiers can be built which use LDFs as components

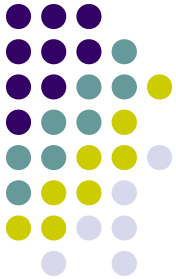
LDF in 2 class problems



- A LDF can be defined as: $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$ where $\mathbf{w}$ is the *weight vector* and w_0 is called *bias* or *threshold weight*.
- In 2 class problems the decision is taken on the basis of the sign of $f(\mathbf{x})$:

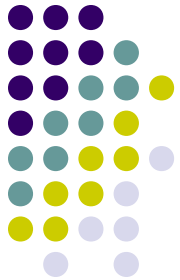
$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 \begin{matrix} >^{\omega_1} \\ \\ <_{\omega_2} \end{matrix} 0$$

LDF and decision regions



- The equation $f(\mathbf{x}) = 0$ defines the decision boundary separating the decision regions of the two classes.
- In the case of LDF, the decision boundary is an hyperplane $H = \{\mathbf{x}|f(\mathbf{x}) = 0\}$ that partitions the feature spaces into two half-spaces $R_1 = \{\mathbf{x}|f(\mathbf{x}) > 0\}$ and $R_2 = \{\mathbf{x}|f(\mathbf{x}) < 0\}$

A bit of geometry ...



- The vector $\mathbf{w}$ is normal to any vector lying on H
- The LDF $f(\mathbf{x})$ gives an algebraic measure of the distance from $\mathbf{x}$ to H
- In particular, the algebraic distance is

$$r = f(\mathbf{x})/\|\mathbf{w}\|$$

- The distance of the hyperplane from the origin is $w_0/\|\mathbf{w}\|$. If $w_0 = 0$ the hyperplane passes through the origin.

Some geometry ...

Since $\mathbf{x} = \mathbf{x}_P + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$

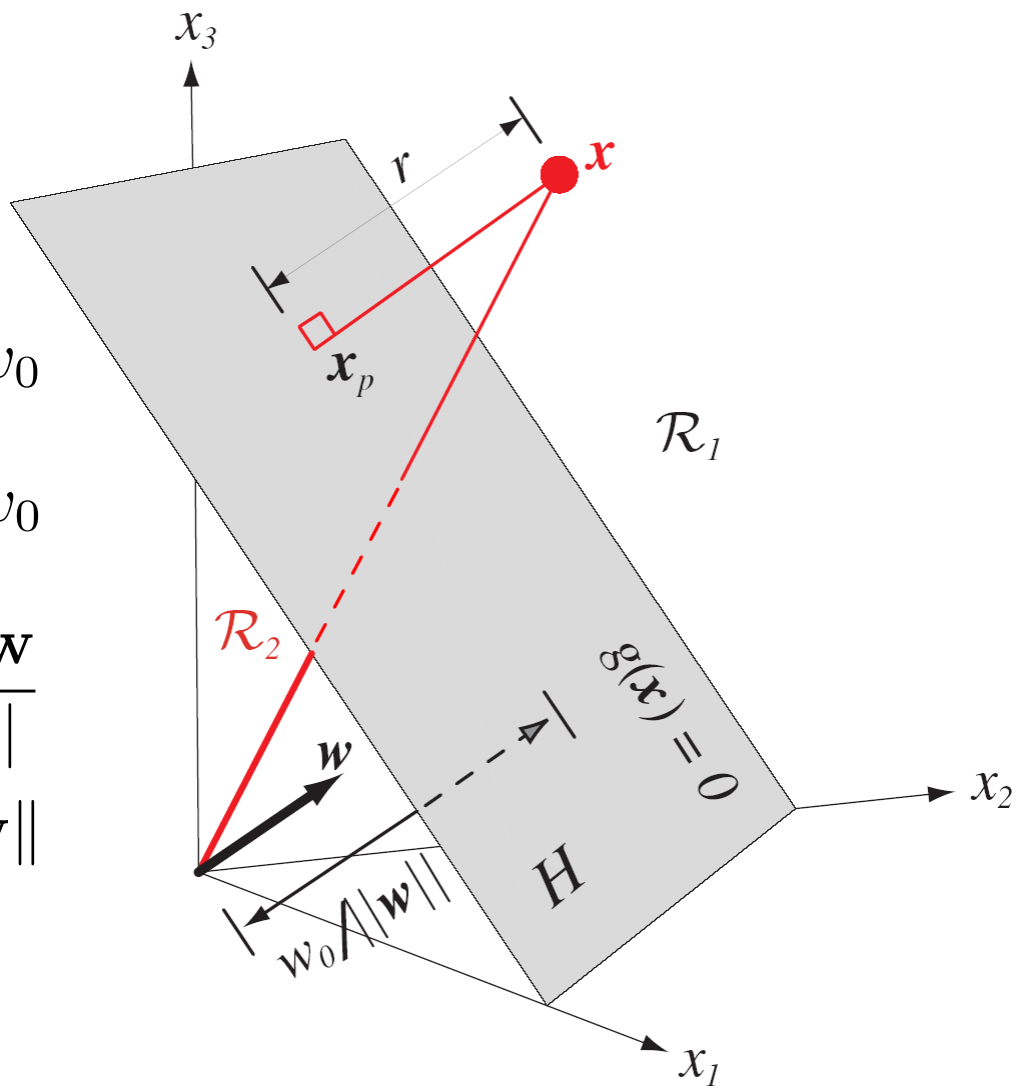
the LDF can be written:

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$$

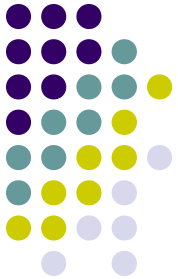
$$= \mathbf{w}^T\left(\mathbf{x}_P + r\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) + w_0$$

$$= \mathbf{w}^T\mathbf{x}_P + w_0 + r\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|}$$

$$= 0 + r\|\mathbf{w}\| = r\|\mathbf{w}\|$$



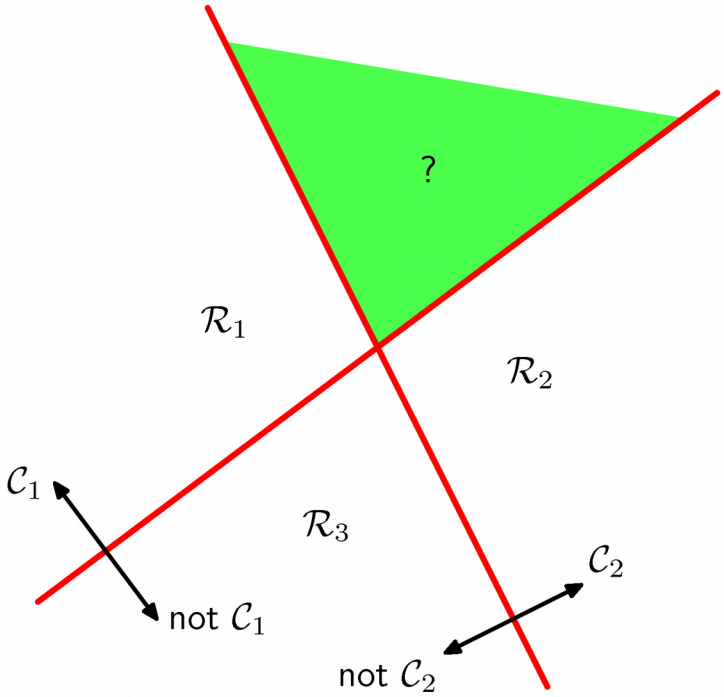
Multiple classes



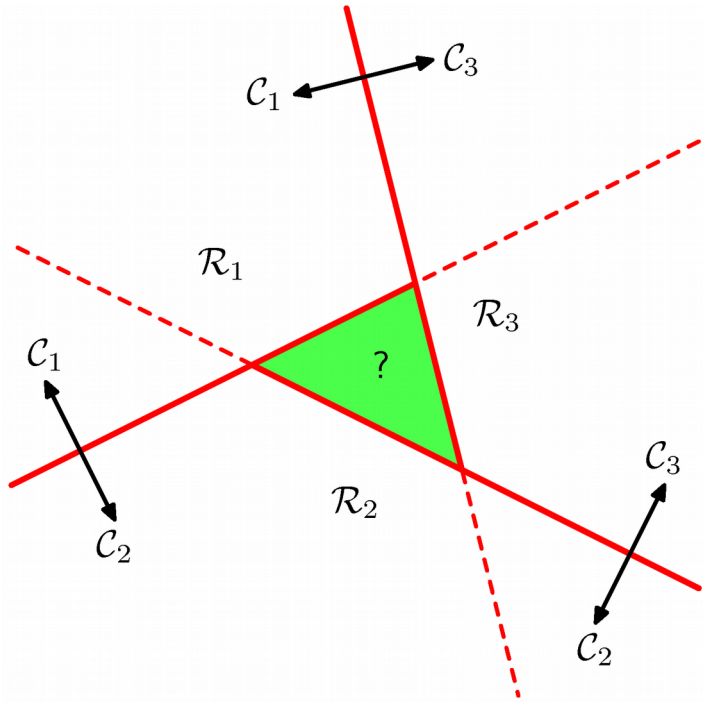
- What about the multiclass problems?
- In principle, we could use one of two approaches (K classes):
  - K classifiers *one-versus-all* (*one-versus- the rest*)
  - K(K-1) classifiers *one-versus-one*

- In both cases, problems

Multiple classes

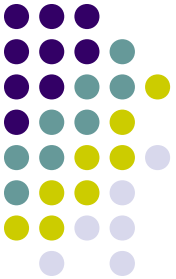


One-versus-all



One-versus-one

Multiple classes



- Solution: consider K LDF of the form

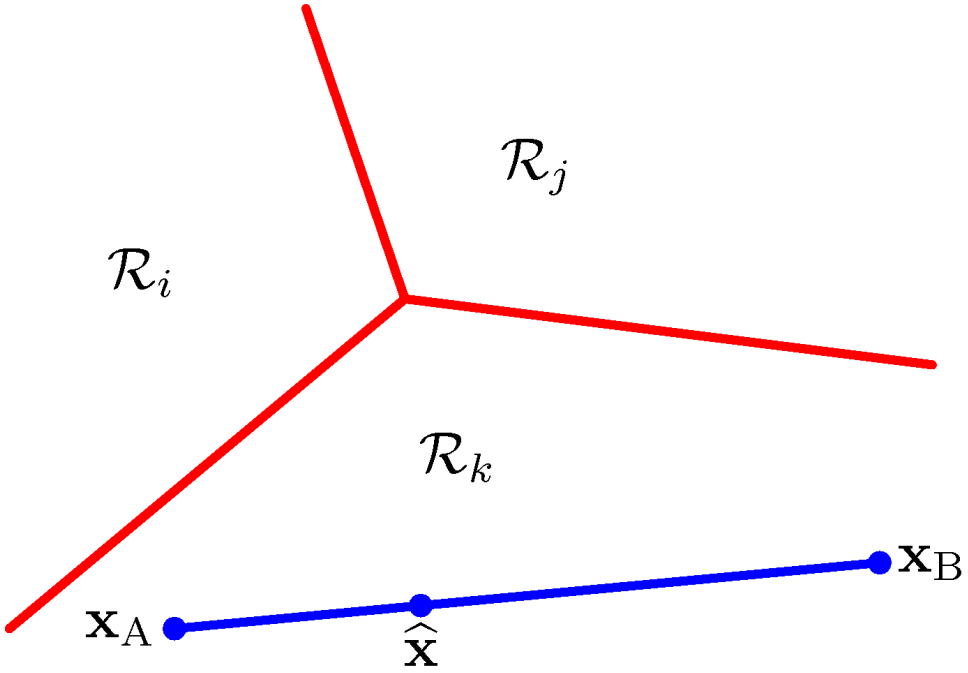
$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$

- Assign a sample to class C_k if $y_k(\mathbf{x}) > y_j(\mathbf{x})$ for all $j \neq k$

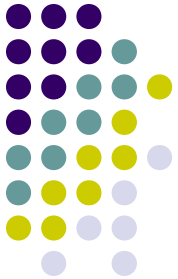
- Decision boundary between R_j and R_k the hyperplane

$$(\mathbf{w}_k - \mathbf{w}_j)^T \mathbf{x} + (w_{k0} - w_{j0}) = 0$$

Multiple classes



Finding the parameters

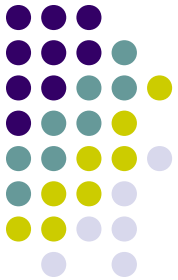


- In the case of LDF, the learning algorithm aims at finding $\mathbf{w}$ and w_0 such to optimize a particular figure of merit (e.g. error rate).
- Several possible algorithms
- We consider
  - LDA (Fisher's discriminant)
  - Perceptron

LDA

- Linear classification model in terms of dimensionality reduction
- For two classes, the input vector $\mathbf{x}$ is projected down to one dimension: $y = \mathbf{w}^T \mathbf{x}$
- The weight vector $\mathbf{w}$ is chosen so as to maximize the “class separation”

LDA



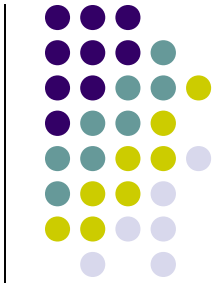
- N_1 points of class C_1;N_2 points of class C_2
- Mean vectors of the two classes:

$$\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in C_1} \mathbf{x}_n \qquad \mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in C_2} \mathbf{x}_n$$

- Maximize the difference between projected classes means

$$(m_2 - m_1) = \mathbf{w}^T (\mathbf{m}_2 - \mathbf{m}_1)$$

LDA



- The expression can be made arbitrarily large simply by increasing the magnitude of $\mathbf{w}$ thus we impose $\|\mathbf{w}\| = 1$
- Maximization problem with constraints:

$$\mathbf{w}^T (\mathbf{m}_2 - \mathbf{m}_1) \, max!$$

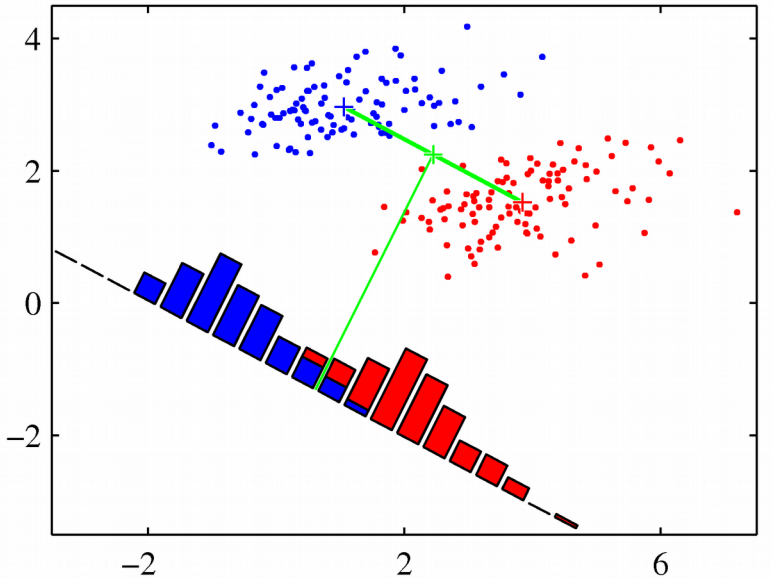
$$s.t. \| \mathbf{w}\| = 1$$

LDA

- It is possible to show that the solution is

$$\mathbf{w} \propto (\mathbf{m}_2 - \mathbf{m}_1)$$

- Maximization of the difference between means not sufficient

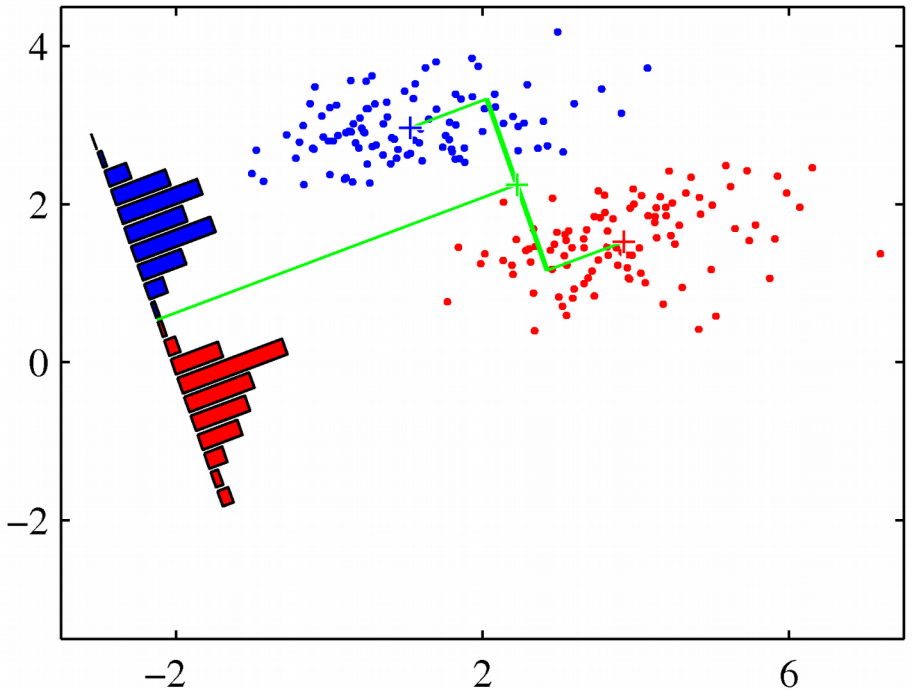


LDA

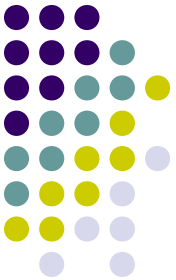
- Minimize the class overlap => minimize the variance within each class

$$\sigma_k^2 = \sum_{n \in C_k} (y_n - m_k)^2$$

$$= \sum_{n \in C_k} \left(\mathbf{w}^T \mathbf{x}_n - \mathbf{w}^T \mathbf{m}_k\right)^2$$



LDA



- The Fisher's criterion is thus given by:

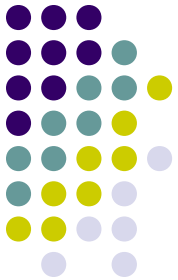
$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$

where $\mathbf{S}_B$ is the between-class covariance matrix $\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_2)(\mathbf{m}_2 - \mathbf{m}_2)^T$

and $\mathbf{S}_W$ is the total within-class cov. mat.

$$\mathbf{S}_W = \sum_{n \in C_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in C_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T$$

LDA



- The maximization of J leads to:

$$\mathbf{w} = \mathbf{S}_W^{-1} (\mathbf{m}_2 - \mathbf{m}_1)$$

- This is know as *Fisher's Linear Discriminant* (1936), although it is not a discriminant but rather a specific choice of direction for the projection of the data down to one dimension

LDA

- The projected data can be subsequently used to construct a discriminant by choosing a threshold y_0 so that

$$\mathbf{w}^T\mathbf{x} \begin{matrix} >^{\omega_1} \\ <_{\omega_2} \end{matrix} y_0$$

The Perceptron algorithm

The Perceptron of Frank Rosenblatt (1962) occupies an important place in the history of Pattern Recognition algorithms.

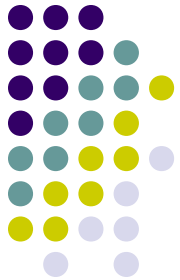
In the early 1960s he had built a special purpose HW that provided a direct implementation of perceptron learning.

Frank Rosenblatt
1928-1971

The perceptron algorithm

- It is based on a LDF $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$
- The algorithm aims at minimizing the errors made by the classifier.
- A natural choice for the error function would be the total number of misclassified samples, BUT minimizing such function is very complicated.

The perceptron algorithm Standardization of the labels

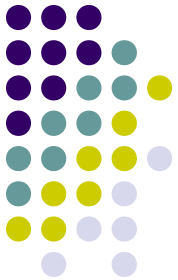


- Let us consider a sort of "standardization" of the labels by attaching to each sample $\mathbf{x}_n$ a label t_n such that:
  - $t_n = -1$ if $\mathbf{x}_n \in \omega_2$
  - $t_n = -1$ if $\mathbf{x}_n \in \omega_1$
- In this way, the sample $\mathbf{x}_n$ is correctly classified when:

$$f(\mathbf{x_n}) \cdot t_n > 0$$

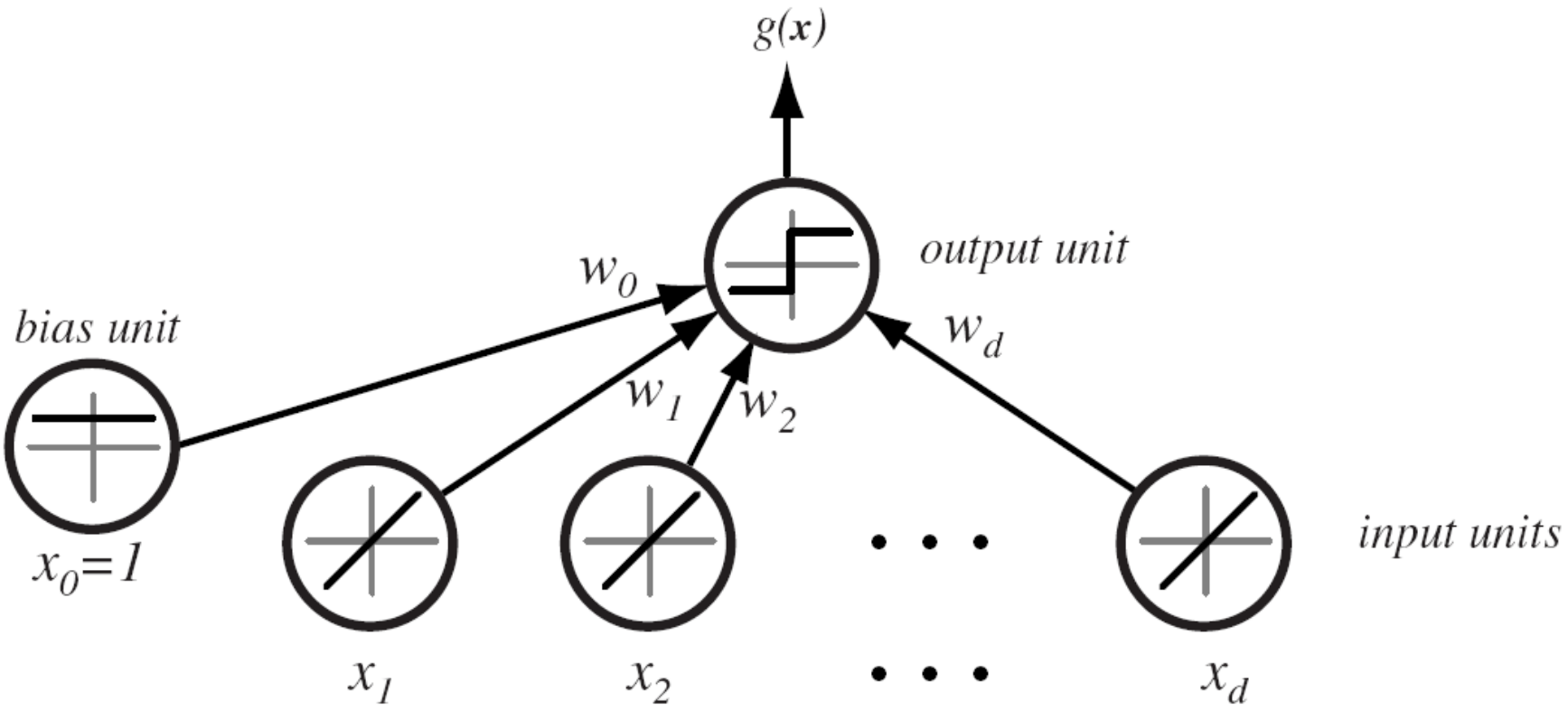
What is $f(\mathbf{x_n}) \cdot t_n$ from a geometric point of view?

Weight vector

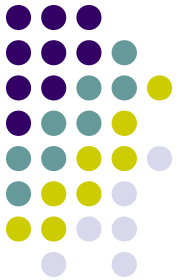


- It is possible to simplify the expression of the LDF with small changes to $\mathbf{x}$ and $\mathbf{w}$.
- If the f.v. is $\mathbf{x}^T = [x_1, x_2, \ldots, x_d]$ and the weight vector is $\mathbf{w}^T = [w_1, w_2, \ldots, w_d]$ we could assume $\mathbf{x}^T = [1, x_1, x_2, \ldots, x_d]$ and $\mathbf{w}^T = [w_0, w_1, w_2, \ldots, w_d]$
- In this way, the expression of the LDF becomes $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

The Perceptron model



The Perceptron algorithm



- For a given weight configuration, the *perceptron criterion* is given by the quantity:

$$J(\mathbf{w}) = -\sum_{n \in M} \mathbf{w}^T \mathbf{x}_n t_n$$

where M denotes the set of all the misclassified patterns:

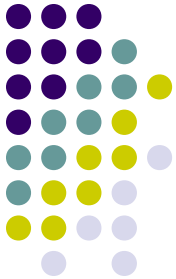
$$M = \{n | \mathbf{w}^T \mathbf{x}_n t_n < 0\}$$

and thus we have $J(\mathbf{w}) \geq 0$

The Perceptron algorithm

- The Perceptron criterion is the sum of the distances between the separating hyperplane defined by $\mathbf{w}$ and the misclassified samples.
- The goal of the learning algorithm is to find the weight vector $\mathbf{w}^*$ minimizing $J(\mathbf{w})$
- In other words, we must determin the vector $\mathbf{w}^*$ in which the gradient of J is zero: $\nabla_{\mathbf{w}} J(\mathbf{w})|_{\mathbf{w}^*} = 0$
- This equation has not a closed solution and thus we use *Gradient Descent* method, an iterative algorithm in which the weights, starting from an initial configuration, are modified proportionally to the negative of the gradient of J .

Gradient descent



- It is a general method for function minimization
- Recall that the minimum of a function $\psi(x)$ is defined by the zeros of the gradient

$$x^* = \arg\min_x \psi(x) \Rightarrow \nabla_x \psi(x)|_{x^*} = 0$$

- Only in very special cases this minimization function has a closed form solution
- In some other cases, a closed form solution may exist, but is numerically ill-posed or impractical (e.g., memory requirements)
- Gradient descent finds the minimum in an iterative fashion by moving in the direction of steepest descent

Gradient descent algorithm

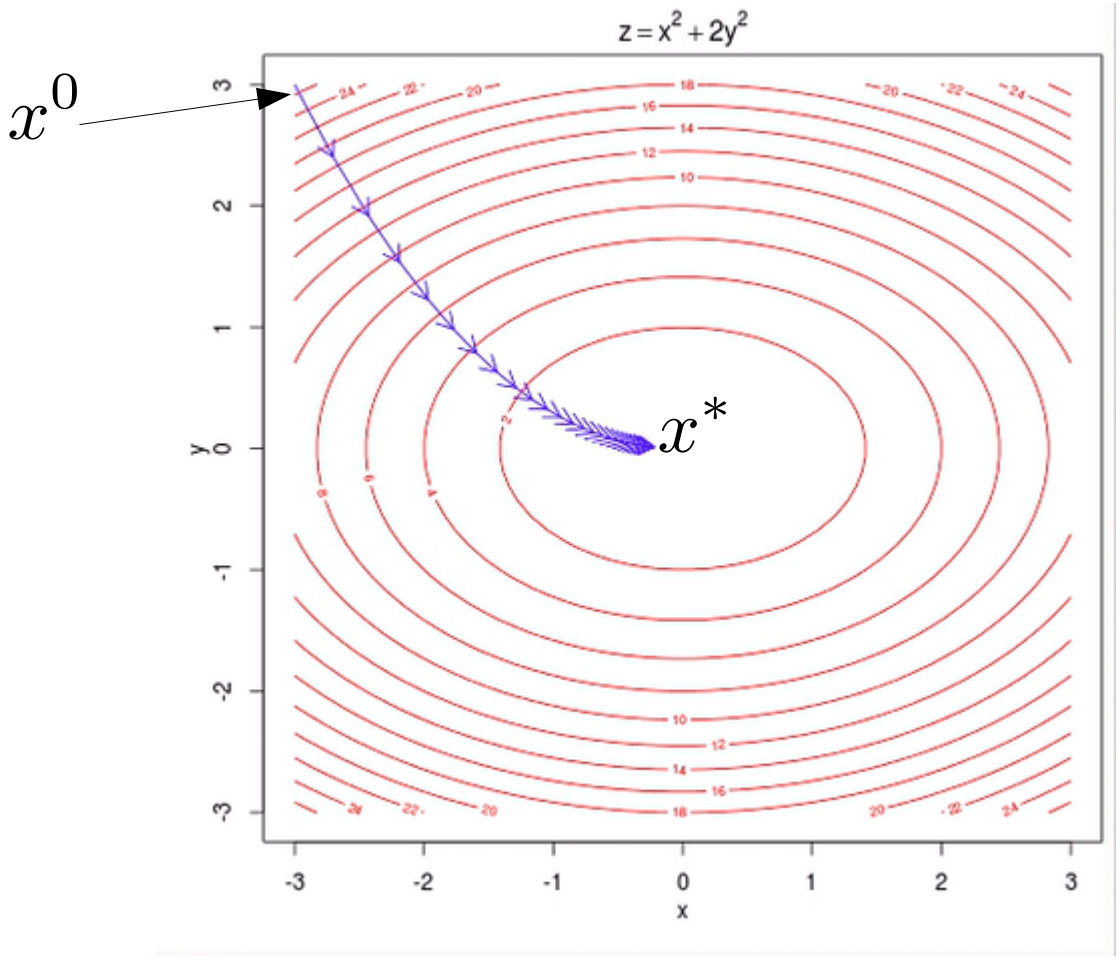
1. Start with an arbitrary solution $k = 0 \quad x^{(k)} = x^0$
2. Compute the gradient $\nabla_x \psi(x)|_{x^{(k)}}$
3. Move in the direction of steepest descent:

$$x^{(k+1)} = x^{(k)} - \eta \nabla_x \psi(x)|_{x^{(k)}}$$
$$k = k + 1$$

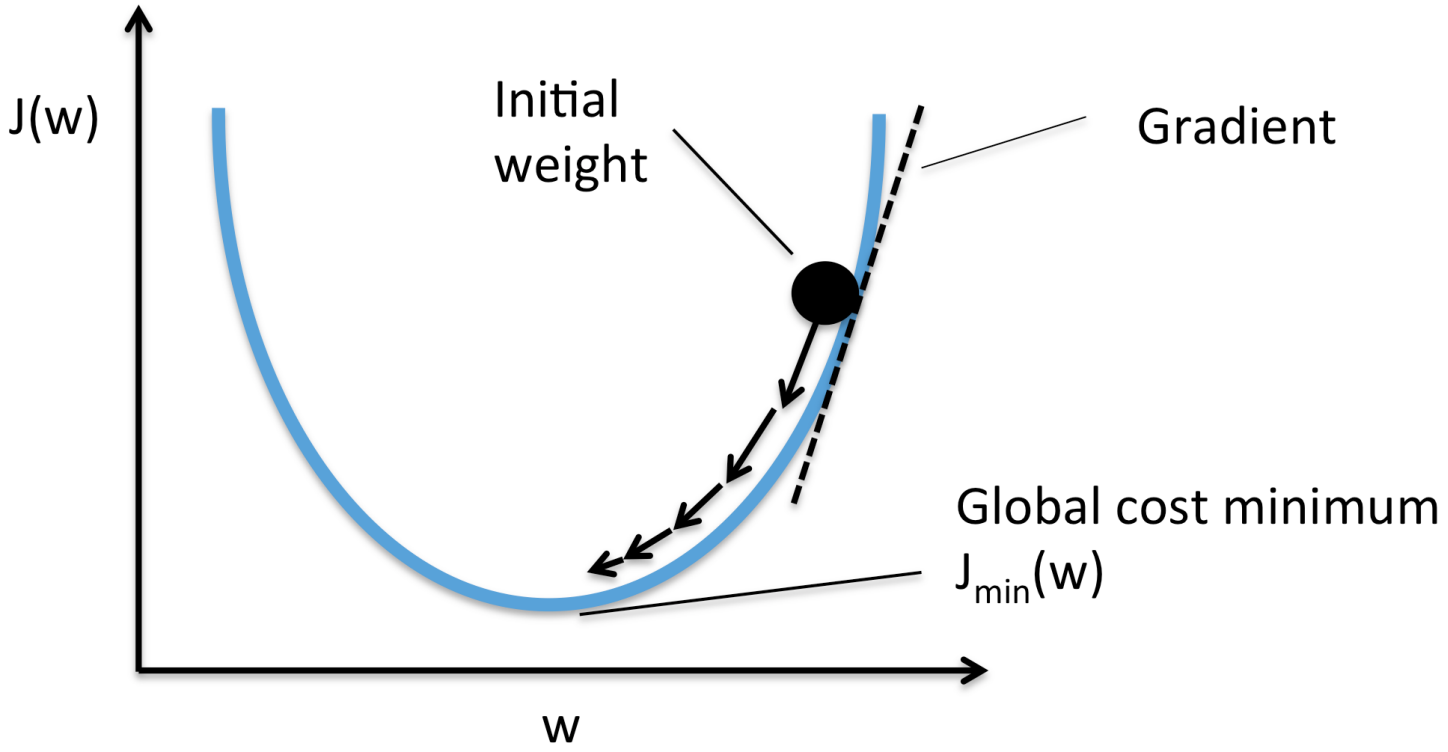
4. Go to 1 (until convergence)

η is the learning rate and defines the magnitude of the modification of $x^{(k)}$

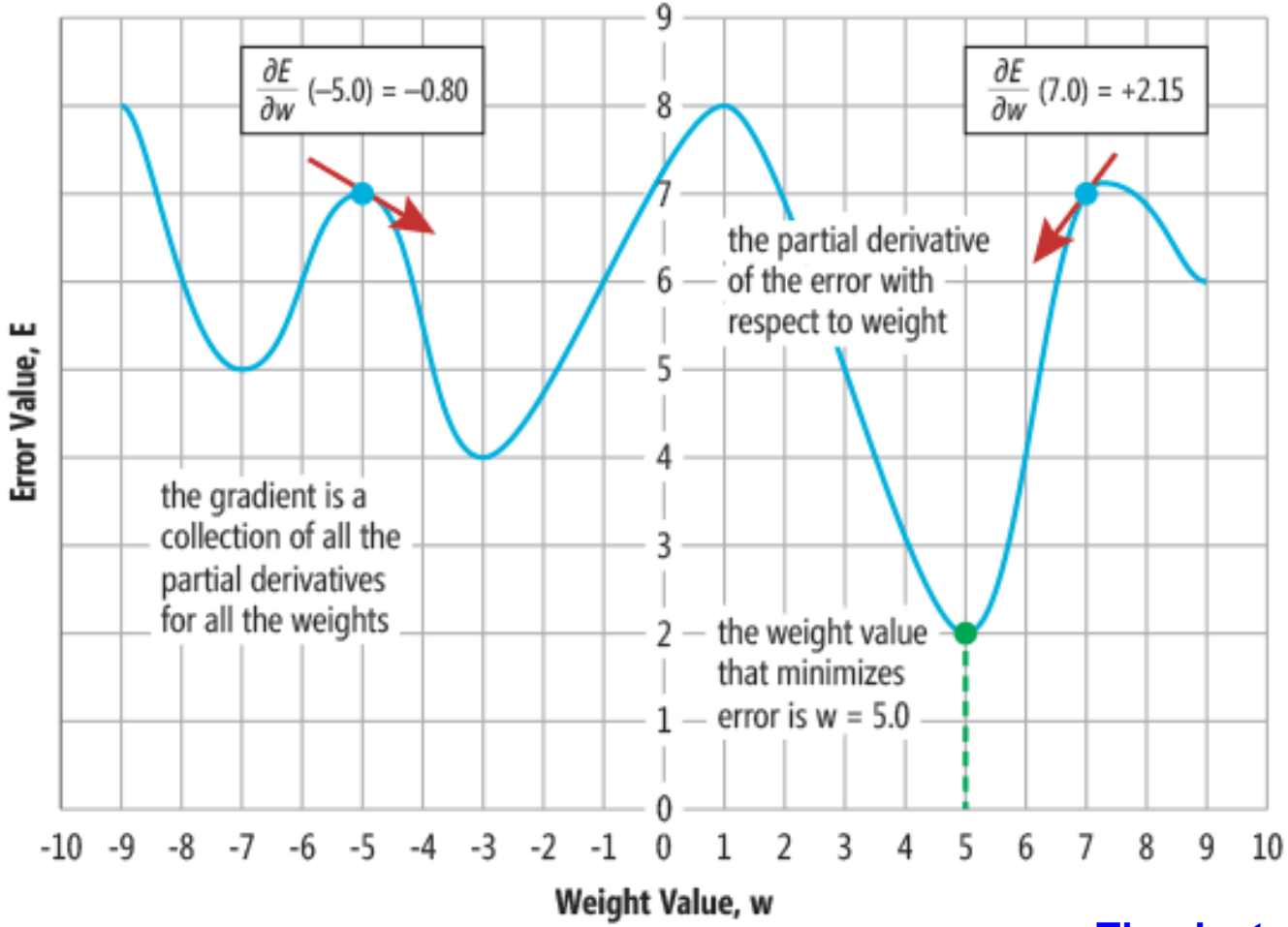
Gradient descent algorithm



Gradient descent algorithm



Gradient descent algorithm

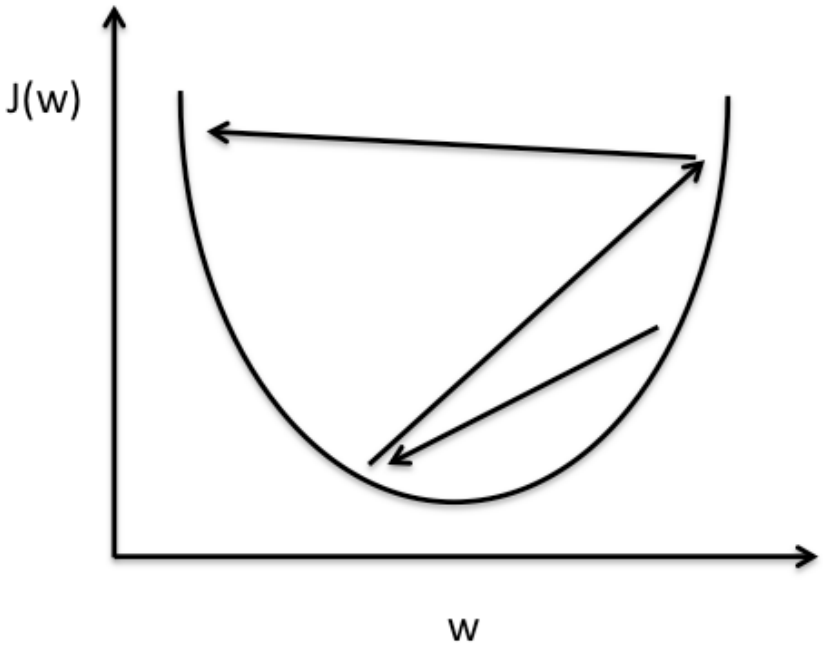


Thanks to James McCaffrey

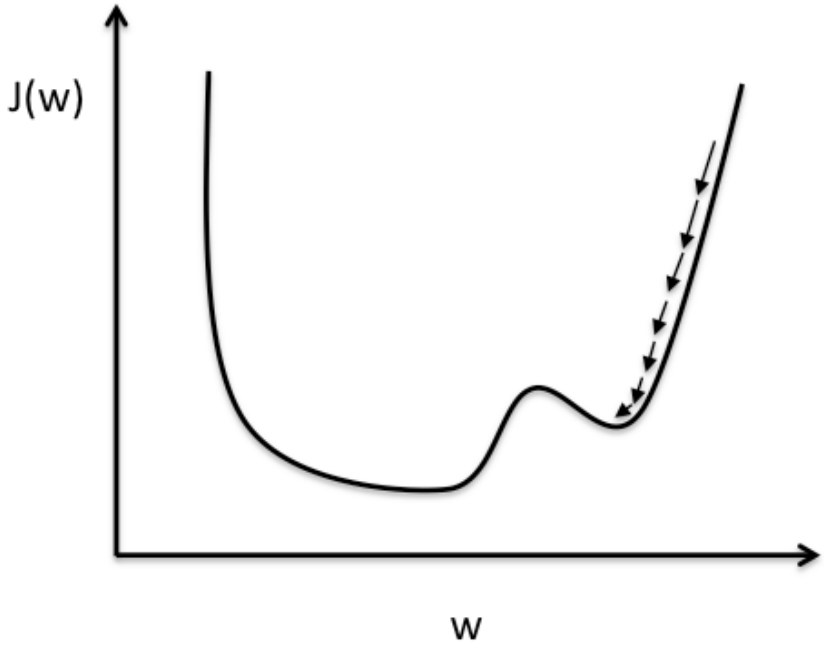
Gradient descent algorithm

- The choice of the value for the learning rate is quite difficult (black magic)
- Too large or too small values can be equally detrimental for the learning process.
  - Too large → Overshooting → no convergence
  - Too small → trapped in a local minimum

Gradient descent algorithm

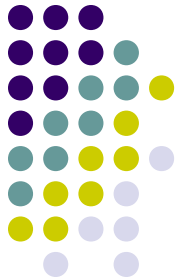


Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

Gradient descent algorithm for the Perceptron



1. Start with an arbitrary solution $k = 0 \quad \mathbf{w}^{(k)} = rand$
2. Evaluate $J(\mathbf{w}^{(k)}) = -\sum_{n \in M} \mathbf{w}^{(k)T} \mathbf{x}_n t_n$
3. Move in the direction of steepest descent:

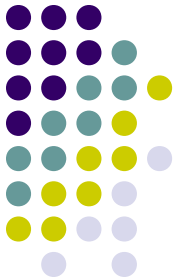
$$w_i{}^{(k+1)} = w_i{}^{(k)} - \eta \nabla_{w_i} J(\mathbf{w})|_{\mathbf{w}^{(k)}} = w_i{}^{(k)} - \eta \left(\sum_{n \in M} x_i^{(n)} t_n \right)$$

4. $k = k + 1$ Go to 2 (until convergence)

The Perceptron algorithm

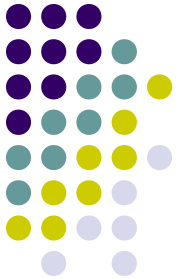
```
iterations=0
repeat
   iterations++
   J=0
   for j=1,#Ts
      evaluates f(x_j)
      if f(x_j)·t_j<0 then
         J=J-f(x_j)·t_j
         for i=0,d
            w_i=w_i+ηx_j^i t_j
         end for
      end if
   end for
until stop(J,iterations)
```

Stopping rules



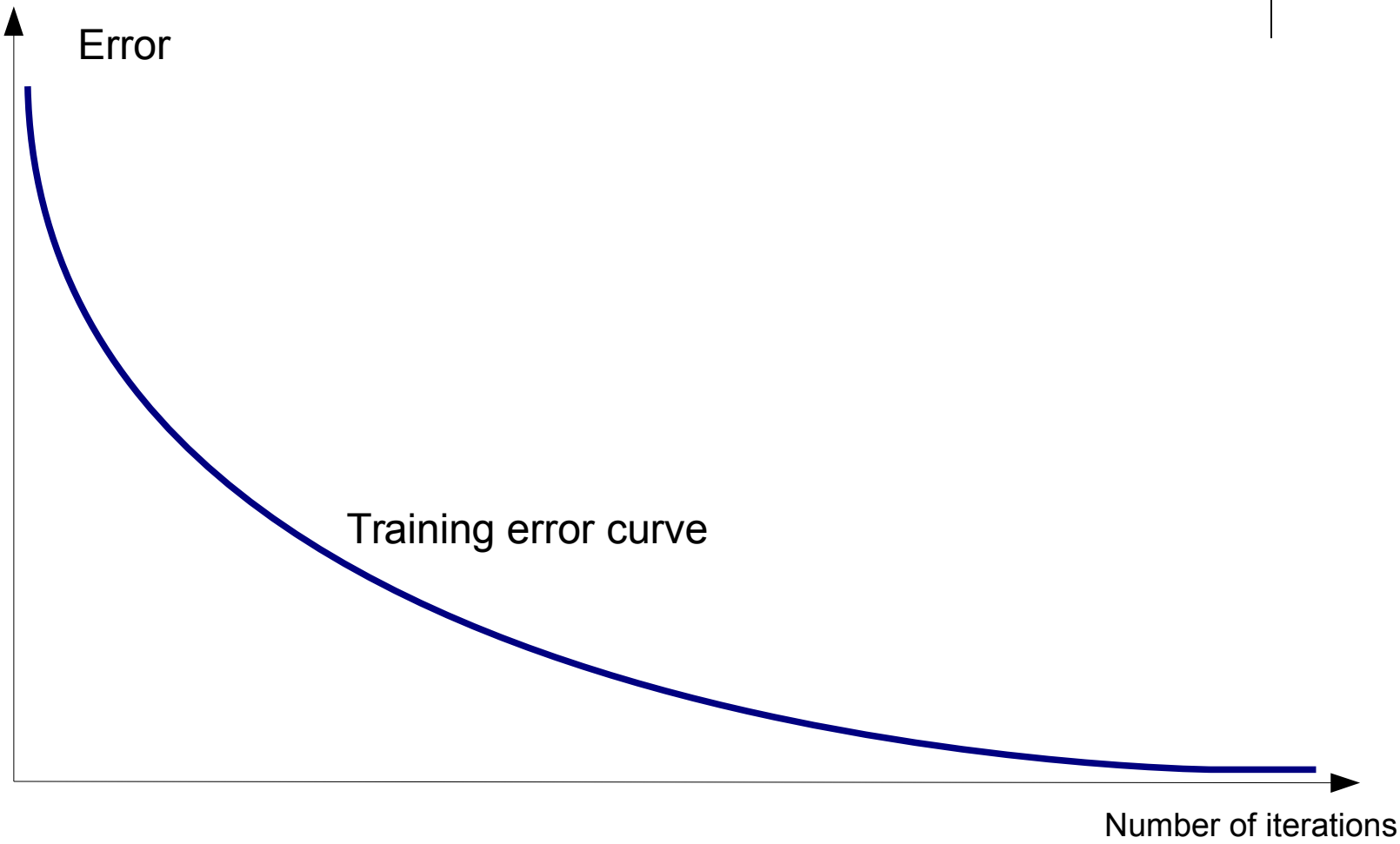
- If classes are linearly separable, the perceptron algorithm is guaranteed to converge to a valid solution
- If the two classes are not linearly separable, the perceptron algorithm will not converge
- Since no weight vector a can correctly classify every sample in a non-separable dataset, the corrections in the perceptron rule will never cease
- One ad-hoc solution to this problem is to define a stopping rule which can be defined in terms of
  - Criterion J : $J(\mathbf{w}^k) \leq J_{min}$
  - Iteration number

Overfitting and underfitting



- When to stop the learning phase?
- The central question is that our learning algorithm must perform well on previously unseen inputs → **Generalization**
- When training a machine learning model, we can compute some error measure on the training set → **Training Error**
- How the training error varies during the training process?

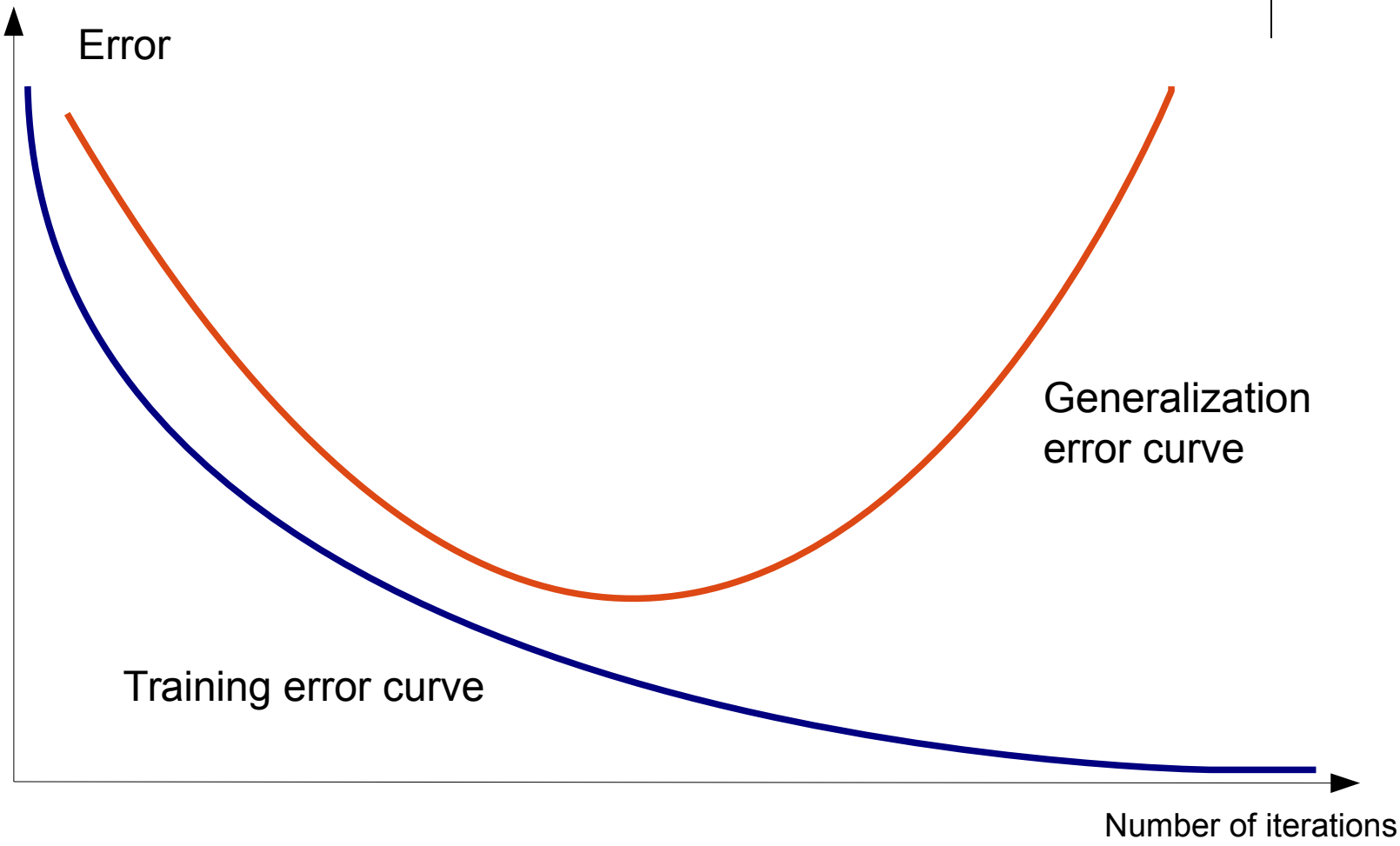
Overfitting and underfitting



Overfitting and underfitting

- We could go on until the training error is low, but we want that the **Test Error** (or **Generalization Error**) must be low as well.
- We tipically estimate the generalization error on a test set containing samples collected separately from the training set.
- What if we observe how the generalization error varies during the training phase?

Overfitting and underfitting



Overfitting and underfitting

- We can observe two different problems.
- **Underfitting**: when the model is not able to obtain a sufficently low error on the training set
- **Overfitting**: when the gap between the training error and the test error is too large

Overfitting and underfitting