

# Numerical features

- ▶ Throughout this lecture we assume that all features are numerical, i.e. feature vectors belong to  $\mathbb{R}^n$ .
- ▶ Most non-numerical features can be conveniently transformed to numerical ones.

**For example:**

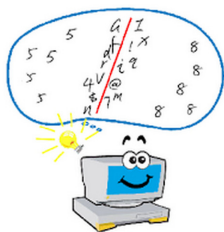
- blue* *red* ▶ Colors  $\{blue, red, yellow\}$  can be represented by  $\{0, 1, 2\}$  (or  $\{-1, 0, 1\}, \dots$ )
- (1, 0, 0), (0, 1, 0),*  
*(0, 0, 1)*  
*yellow*
- ▶ A black-and-white picture of  $x \times y$  pixels can be encoded as a vector of  $xy$  numbers that capture the shades of gray of the pixels.

# Basic Problems

We consider two basic problems:

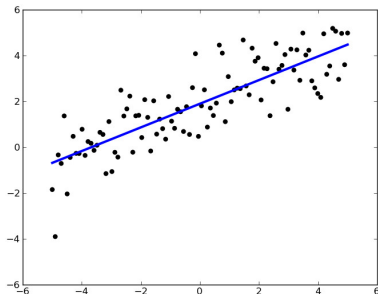
- ▶ (Binary) classification

**Our goal:** Classify inputs into two categories.



- ▶ Function approximation (regression)

**Our goal:** Find a (hypothesized) functional dependency in data.



# Binary classification in $\mathbb{R}^n$

- ▶ Assume
  - ▶ a set of instances  $X \subseteq \mathbb{R}^n$ ,
  - ▶ an *unknown* categorization function  $c : X \rightarrow \{0, 1\}$ .
- ▶ **Our goal:**
  - ▶ Given a set  $D$  of training examples of the form  $(\vec{x}, c(\vec{x}))$  where  $\vec{x} \in X$ ,
  - ▶ construct a hypothesized categorization function  $h \in \mathcal{H}$  that is consistent with  $c$  on the training examples, i.e.,
$$h(\vec{x}) = c(\vec{x}) \text{ for all training examples } (\vec{x}, c(\vec{x})) \in D$$

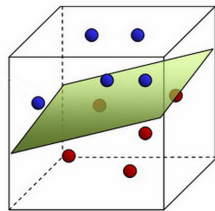
## Comments:

- ▶ In practice, we often do not strictly demand  $h(\vec{x}) = c(\vec{x})$  for all training examples  $(\vec{x}, c(\vec{x})) \in D$  (often it is impossible)
- ▶ We are more interested in good **generalization**, that is how well  $h$  classifies new instances that do not belong to  $D$ .
  - ▶ Recall that we usually evaluate accuracy of the resulting hypothesized function  $h$  on a test set.

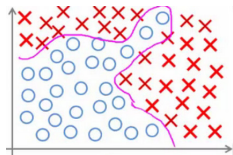
# Hypothesis Spaces

We consider two kinds of hypothesis spaces:

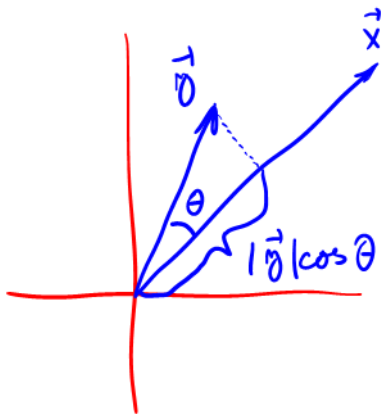
- ▶ Linear (affine) classifiers (this lecture)



- ▶ Classifiers based on combinations of linear and sigmoidal functions (classical neural networks) (next lecture)



## Length and Scalar Product of Vectors

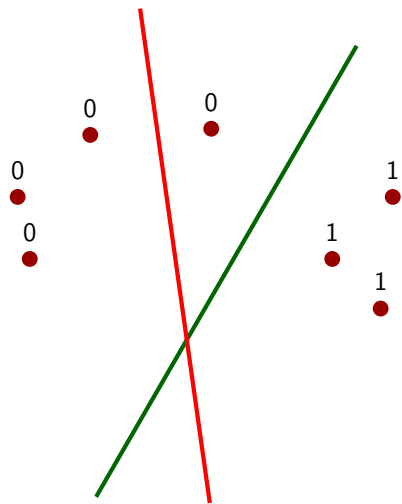


- ▶ We consider vectors  $\vec{x} = (x_1, \dots, x_m) \in \mathbb{R}^m$ .
- ▶ Typically, we use Euclidean metric on vectors:  $|\vec{x}| = \sqrt{\sum_{i=1}^m x_i^2}$   
The distance between two vectors (points)  $\vec{x}, \vec{y}$  is  $|\vec{x} - \vec{y}|$ .
- ▶ We use the *scalar product*  $\vec{x} \cdot \vec{y}$  of vectors  $\vec{x} = (x_1, \dots, x_m)$  and  $\vec{y} = (y_1, \dots, y_m)$  defined by

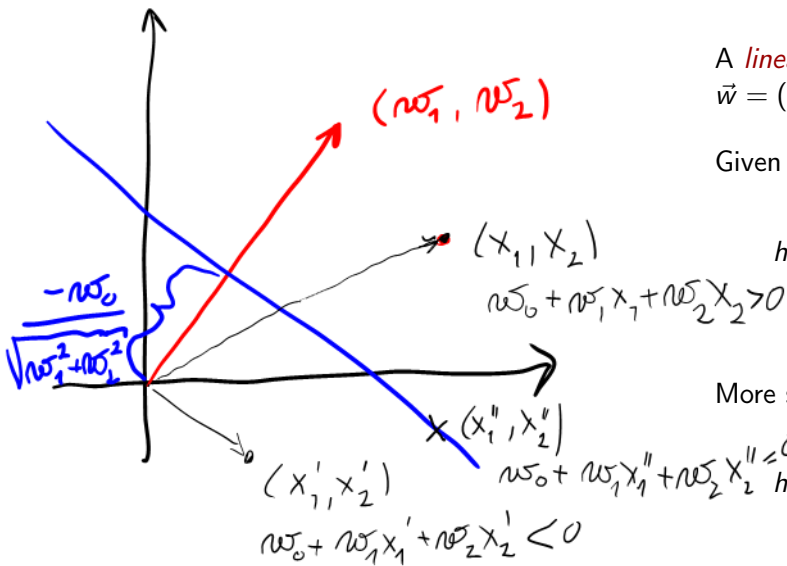
$$\vec{x} \cdot \vec{y} = \sum_{i=1}^m x_i y_i$$

- ▶ Recall that  $\vec{x} \cdot \vec{y} = |\vec{x}| |\vec{y}| \cos \theta$  where  $\theta$  is the angle between  $\vec{x}$  and  $\vec{y}$ . That is  $\vec{x} \cdot \vec{y}$  is the length of the projection of  $\vec{y}$  on  $\vec{x}$  multiplied by  $|\vec{x}|$ .
- ▶ Note that  $\vec{x} \cdot \vec{x} = |\vec{x}|^2$

## Linear classifier - example



- ▶ classification in plane using a linear classifier
- ▶ if a point is incorrectly classified, the learning algorithm turns the line (hyperplane) to improve the classification.

$\mathbb{R}^2$ 

## Linear Classifier

A *linear classifier*  $h[\vec{w}]$  is determined by a vector of *weights*  $\vec{w} = (w_0, w_1, \dots, w_n) \in \mathbb{R}^{n+1}$  as follows:

Given  $\vec{x} = (x_1, \dots, x_n) \in X \subseteq \mathbb{R}^n$ ,

$$h[\vec{w}](\vec{x}) := \begin{cases} 1 & w_0 + \sum_{i=1}^n w_i \cdot x_i \geq 0 \\ 0 & w_0 + \sum_{i=1}^n w_i \cdot x_i < 0 \end{cases}$$

More succinctly:

$$h(\vec{x}) = \text{sgn} \left( w_0 + \sum_{i=1}^n w_i \cdot x_i \right) \quad \text{where} \quad \text{sgn}(y) = \begin{cases} 1 & y \geq 0 \\ 0 & y < 0 \end{cases}$$

## Linear Classifier – Notation

Given  $\vec{x} = (x_1, \dots, x_n) \in \mathbb{R}^n$  we define an *augmented feature vector*

$$\tilde{\vec{x}} = (x_0, x_1, \dots, x_n) \quad \text{where } x_0 = 1$$

This makes the notation for the linear classifier more succinct:

$$h[\vec{w}](\vec{x}) = \text{sgn}(\vec{w} \cdot \tilde{\vec{x}})$$

$$\begin{aligned} \vec{w} \cdot \tilde{\vec{x}} &= w_0 \cdot x_0 + w_1 x_1 + \dots + w_n x_n \\ &= w_0 + \sum_{i=1}^n w_i x_i \end{aligned}$$



# Perceptron Learning

- ▶ Given a training set

$$D = \{(\vec{x}_1, c(\vec{x}_1)), (\vec{x}_2, c(\vec{x}_2)), \dots, (\vec{x}_p, c(\vec{x}_p))\}$$

Here  $\vec{x}_k = (x_{k1}, \dots, x_{kn}) \in X \subseteq \mathbb{R}^n$  and  $c(\vec{x}_k) \in \{0, 1\}$ .

**We write  $c_k$  instead of  $c(\vec{x}_k)$ .**

Note that  $\tilde{\mathbf{x}}_k = (x_{k0}, x_{k1}, \dots, x_{kn})$  where  $x_{k0} = 1$ .

- ▶ A weight vector  $\vec{w} \in \mathbb{R}^{n+1}$  is **consistent with  $D$**  if

$$h[\vec{w}](\vec{x}_k) = \text{sgn}(\vec{w} \cdot \tilde{\mathbf{x}}_k) = c_k \quad \text{for all } k = 1, \dots, p$$

$D$  is **linearly separable** if there is a vector  $\vec{w} \in \mathbb{R}^{n+1}$  which is consistent with  $D$ .

- ▶ Our goal is to find a consistent  $\vec{w}$  assuming that  $D$  is linearly separable.

$\mathbb{R}^{m+1}$  $c_k = 0$ 

## Perceptron – Learning Algorithm

### Online learning algorithm:

Idea: Cyclically go through the training examples in  $D$  and adapt weights. Whenever an example is incorrectly classified, turn the hyperplane so that the example is closer to its correct half-space.

Compute a sequence of weight vectors  $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \dots$

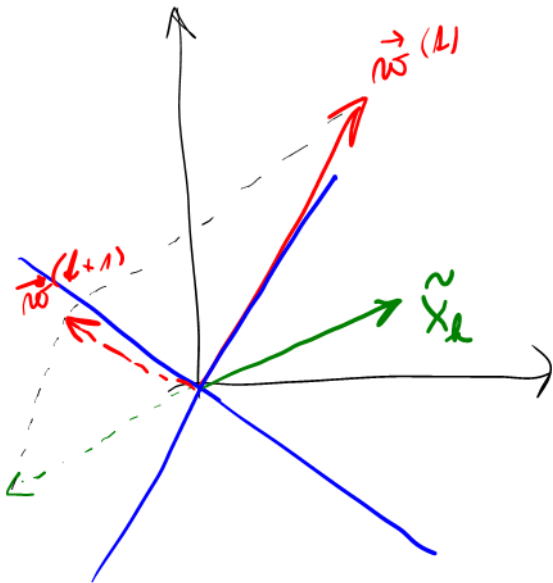
- ▶  $\vec{w}^{(0)}$  is randomly initialized close to  $\vec{0} = (0, \dots, 0)$
- ▶ In  $(t+1)$ -th step,  $\vec{w}^{(t+1)}$  is computed as follows:

$$\begin{aligned}\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \left( h[\vec{w}^{(t)}](\vec{x}_k) - c_k \right) \cdot \tilde{\vec{x}}_k \\ &= \vec{w}^{(t)} - \varepsilon \cdot \left( \text{sgn} \left( \vec{w}^{(t)} \cdot \tilde{\vec{x}}_k \right) - c_k \right) \cdot \tilde{\vec{x}}_k\end{aligned}$$

Here  $k = (t \bmod p) + 1$ , i.e. the examples are considered cyclically, and  $0 < \varepsilon \leq 1$  is a **learning speed**.

### Věta (Rosenblatt)

If  $D$  is linearly separable, then there is  $t^*$  such that  $\vec{w}^{(t^*)}$  is consistent with  $D$ .



## Example

Training set:

$$D = \{((2, -1), 1), ((2, 1), 1), ((1, 3), 0)\}$$

That is

$$\vec{x}_1 = (2, -1)$$

$$\vec{x}_2 = (2, 1)$$

$$\vec{x}_3 = (1, 3)$$

$$\tilde{\mathbf{x}}_1 = (\mathbf{1}, 2, -1)$$

$$\tilde{\mathbf{x}}_2 = (\mathbf{1}, 2, 1)$$

$$\tilde{\mathbf{x}}_3 = (\mathbf{1}, 1, 3)$$

$$c_1 = 1$$

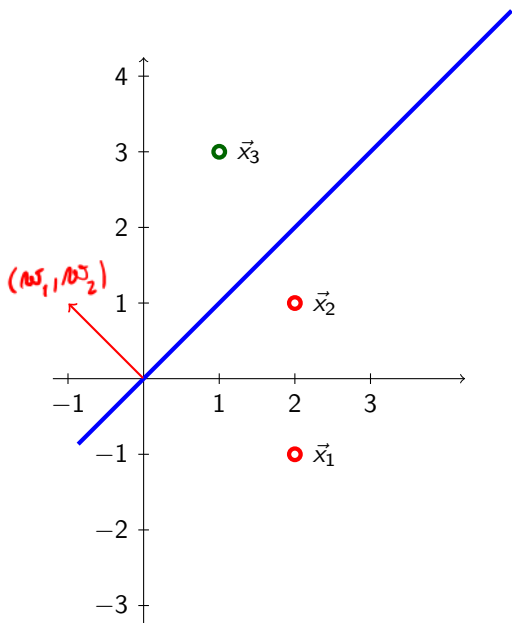
$$c_2 = 1$$

$$c_3 = 0$$

Assume that the initial vector  $\vec{w}^{(0)}$  is  $\vec{w}^{(0)} = (0, -1, 1)$ .

Consider  $\varepsilon = 1$ .

## Example: Separating by $\vec{w}^{(0)}$



Denoting  $\vec{w}^{(0)} = (w_0, w_1, w_2) = (0, -1, 1)$  the blue separating line is given by  $w_0 + w_1x_1 + w_2x_2 = 0$ .

The red vector normal to the blue line is  $(w_1, w_2)$ .

The points on the side of  $(w_1, w_2)$  are assigned 1 by the classifier, the others zero. (In this case  $\vec{x}_3$  is assigned one and  $\vec{x}_1, \vec{x}_2$  are assigned zero, all of this is inconsistent with  $c_1 = 1, c_2 = 1, c_3 = 0$ .)

## Example: $\vec{w}^{(1)}$

We have

$$\vec{w}^{(0)} \cdot \tilde{\mathbf{x}}_1 = (0, -1, 1) \cdot (1, 2, -1) = 0 - 2 - 1 = -3$$

thus

$$\text{sgn} \left( \vec{w}^{(0)} \cdot \tilde{\mathbf{x}}_1 \right) = 0$$

and thus

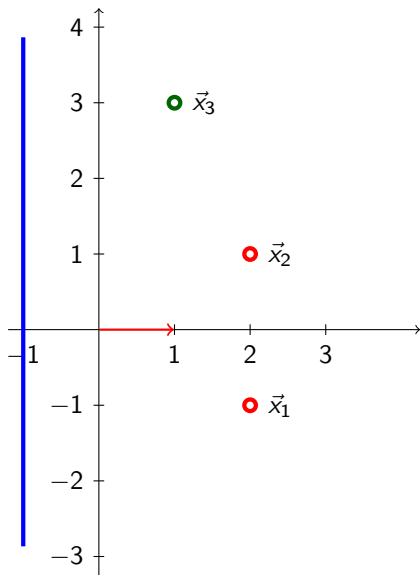
$$\text{sgn} \left( \vec{w}^{(0)} \cdot \tilde{\mathbf{x}}_1 \right) - c_1 = 0 - 1 = -1$$

(This means that  $\tilde{\mathbf{x}}_1$  is not well classified, and  $\vec{w}^{(0)}$  is not consistent with  $D$ .)

Hence,

$$\begin{aligned} \vec{w}^{(1)} &= \vec{w}^{(0)} - \left( \text{sgn} \left( \vec{w}^{(0)} \cdot \tilde{\mathbf{x}}_1 \right) - c_1 \right) \cdot \tilde{\mathbf{x}}_1 \\ &= \vec{w}^{(0)} + \tilde{\mathbf{x}}_1 \\ &= (0, -1, 1) + (1, 2, -1) \\ &= (1, 1, 0) \end{aligned}$$

# Example



## Example: Separating by $\vec{w}^{(1)}$

We have

$$\vec{w}^{(1)} \cdot \tilde{\mathbf{x}}_2 = (1, 1, 0) \cdot (1, 2, 1) = 1 + 2 = 3$$

thus

$$\text{sgn} \left( \vec{w}^{(1)} \cdot \tilde{\mathbf{x}}_2 \right) = 1$$

and thus

$$\text{sgn} \left( \vec{w}^{(1)} \cdot \tilde{\mathbf{x}}_2 \right) - c_2 = 1 - 1 = 0$$

(This means that  $\tilde{\mathbf{x}}_2$  is currently well classified by  $\vec{w}^{(1)}$ . However, as we will see,  $\tilde{\mathbf{x}}_3$  is not well classified.)

Hence,

$$\vec{w}^{(2)} = \vec{w}^{(1)} = (1, 1, 0)$$

## Example: $\vec{w}^{(3)}$

We have

$$\vec{w}^{(2)} \cdot \tilde{\mathbf{x}}_3 = (1, 1, 0) \cdot (1, 1, 3) = 1 + 1 = 2$$

thus

$$\text{sgn} \left( \vec{w}^{(2)} \cdot \tilde{\mathbf{x}}_3 \right) = 1$$

and thus

$$\text{sgn} \left( \vec{w}^{(2)} \cdot \tilde{\mathbf{x}}_3 \right) - c_3 = 1 - 0 = 1$$

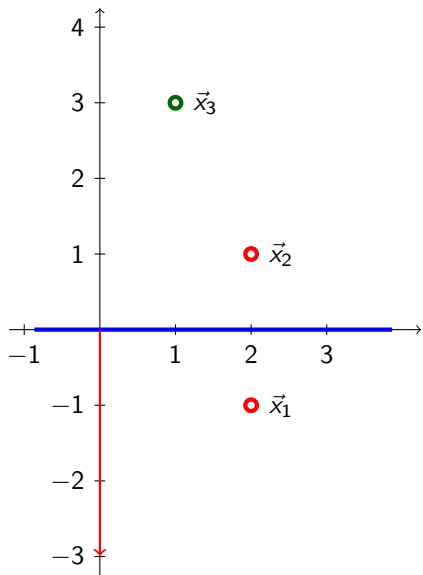
(This means that  $\tilde{\mathbf{x}}_3$  is not well classified, and  $\vec{w}^{(2)}$  is not consistent with  $D$ .)

Hence,

$$\begin{aligned} \vec{w}^{(3)} &= \vec{w}^{(2)} - \left( \text{sgn} \left( \vec{w}^{(2)} \cdot \tilde{\mathbf{x}}_3 \right) - c_3 \right) \cdot \tilde{\mathbf{x}}_3 \\ &= \vec{w}^{(2)} - \tilde{\mathbf{x}}_3 \\ &= (1, 1, 0) - (1, 1, 3) \\ &= (0, 0, -3) \end{aligned}$$



## Example: Separating by $\vec{w}^{(3)}$



## Example: $\vec{w}^{(4)}$

We have

$$\vec{w}^{(3)} \cdot \tilde{\mathbf{x}}_1 = (0, 0, -3) \cdot (1, 2, -1) = 3$$

thus

$$\text{sgn} \left( \vec{w}^{(3)} \cdot \tilde{\mathbf{x}}_1 \right) = 1$$

and thus

$$\text{sgn} \left( \vec{w}^{(3)} \cdot \tilde{\mathbf{x}}_1 \right) - c_1 = 1 - 1 = 0$$

(This means that  $\tilde{\mathbf{x}}_1$  is currently well classified by  $\vec{w}^{(3)}$ . However, as we will see,  $\tilde{\mathbf{x}}_2$  is not.)

Hence,

$$\vec{w}^{(4)} = \vec{w}^{(3)} = (0, 0, -3)$$

## Example: $\vec{w}^{(5)}$

We have

$$\vec{w}^{(4)} \cdot \tilde{\mathbf{x}}_2 = (0, 0, -3) \cdot (1, 2, 1) = -3$$

thus

$$\text{sgn}(\vec{w}^{(4)} \cdot \tilde{\mathbf{x}}_2) = 0$$

and thus

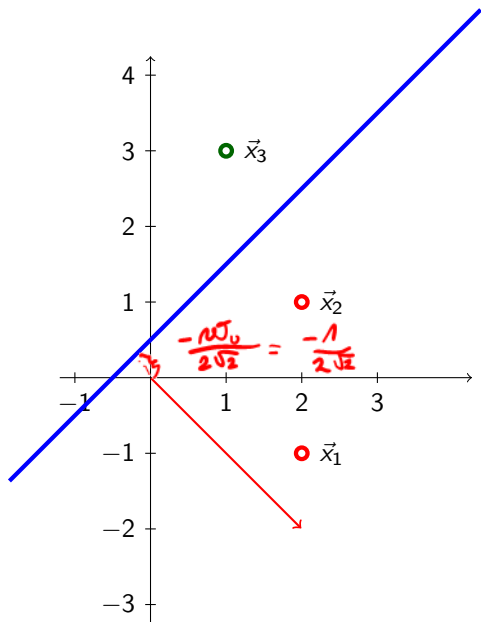
$$\text{sgn}(\vec{w}^{(4)} \cdot \tilde{\mathbf{x}}_2) - c_2 = 0 - 1 = -1$$

(This means that  $\tilde{\mathbf{x}}_2$  is not well classified, and  $\vec{w}^{(4)}$  is not consistent with  $D$ .)

Hence,

$$\begin{aligned}\vec{w}^{(5)} &= \vec{w}^{(4)} - \left( \text{sgn}(\vec{w}^{(4)} \cdot \tilde{\mathbf{x}}_2) - c_2 \right) \cdot \tilde{\mathbf{x}}_2 \\ &= \vec{w}^{(4)} + \tilde{\mathbf{x}}_2 \\ &= (0, 0, -3) + (1, 2, 1) \\ &= (1, 2, -2)\end{aligned}$$

## Example: Separating by $\vec{w}^{(5)}$



$$\vec{w} = (1, 2, -2)$$

## Example: The result

The vector  $\vec{w}^{(5)}$  is consistent with  $D$ :

$$\text{sgn} \left( \vec{w}^{(5)} \cdot \tilde{\mathbf{x}}_1 \right) = \text{sgn} \left( (1, 2, -2) \cdot (1, 2, -1) \right) = \text{sgn}(7) = 1 = c_1$$

$$\text{sgn} \left( \vec{w}^{(5)} \cdot \tilde{\mathbf{x}}_2 \right) = \text{sgn} \left( (1, 2, -2) \cdot (1, 2, 1) \right) = \text{sgn}(3) = 1 = c_2$$

$$\text{sgn} \left( \vec{w}^{(5)} \cdot \tilde{\mathbf{x}}_3 \right) = \text{sgn} \left( (1, 2, -2) \cdot (1, 1, 3) \right) = \text{sgn}(-3) = -1 = c_3$$

# Perceptron – Learning Algorithm

## Batch learning algorithm:

Compute a sequence of weight vectors  $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \dots$

- ▶  $\vec{w}^{(0)}$  is randomly initialized close to  $\vec{0} = (0, \dots, 0)$
- ▶ In  $(t + 1)$ -th step,  $\vec{w}^{(t+1)}$  is computed as follows:

$$\begin{aligned}\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^p \left( h[\vec{w}^{(t)}](\vec{x}_k) - c_k \right) \cdot \tilde{\vec{x}}_k \\ &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^p \left( \text{sgn} \left( \vec{w}^{(t)} \cdot \tilde{\vec{x}}_k \right) - c_k \right) \cdot \tilde{\vec{x}}_k\end{aligned}$$

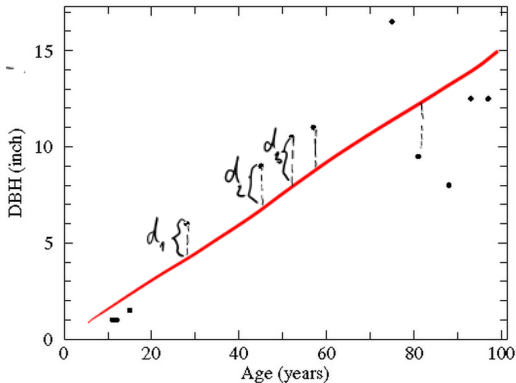
Here  $k = (t \bmod p) + 1$ , i.e. the examples are considered cyclically, and  $0 < \varepsilon \leq 1$  is a **learning speed**.

# Function Approximation – Oaks in Wisconsin

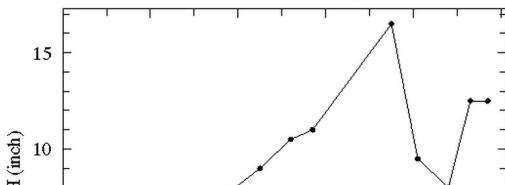
This example is from *How to Lie with Statistics* by Darrell Huff (1954)

$$E = d_1^2 + d_2^2 + d_3^2 + \dots$$

Oak Diameter vs. Age



Oak Diameter vs. Age



Age (years)	DBH (inch)
97	12.5
93	12.5
88	8.0

# Function Approximation

- ▶ Assume
  - ▶ a set  $X \subseteq \mathbb{R}^n$  of instances,
  - ▶ an *unknown* function  $f : X \rightarrow \mathbb{R}$ .
- ▶ **Our goal:**
  - ▶ Given a set  $D$  of training examples of the form  $(\vec{x}, f(\vec{x}))$  where  $\vec{x} \in X$ ,
  - ▶ construct a hypothesized function  $h \in \mathcal{H}$  such that
$$h(\vec{x}) \approx f(\vec{x})$$
for all training examples  $(\vec{x}, f(\vec{x})) \in D$ 

Here  $\approx$  means that the values are somewhat close to each other w.r.t. an appropriate *error function*  $E$ .
- ▶ In what follows we use the *least squares* defined by

$$E = \frac{1}{2} \sum_{(\vec{x}, f(\vec{x})) \in D} (f(\vec{x}) - h(\vec{x}))^2$$

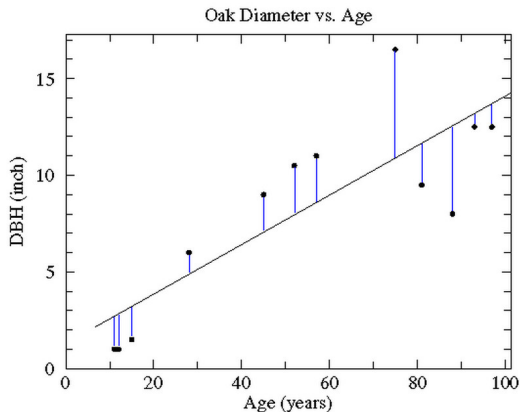
Our goal is to minimize  $E$ .

The main reason is that this function has nice mathematical properties (as opposed e.g. to  $\sum_{(\vec{x}, f(\vec{x})) \in D} |f(\vec{x}) - h(\vec{x})|$ ).



# Least Squares – Oaks in Wisconsin

Age (years)	DBH (inch)
97	12.5
93	12.5
88	8.0
81	9.5
75	16.5
57	11.0
52	10.5
45	9.0
28	6.0
15	1.5
12	1.0
11	1.0



# Linear Function Approximation

- ▶ Given a set  $D$  of training examples:

$$D = \{(\vec{x}_1, f(\vec{x}_1)), (\vec{x}_2, f(\vec{x}_2)), \dots, (\vec{x}_p, f(\vec{x}_p))\}$$

Here  $\vec{x}_k = (x_{k1} \dots, x_{kn}) \in \mathbb{R}^n$  and  $f_k(\vec{x}) \in \mathbb{R}$ .

Recall that  $\tilde{\mathbf{x}}_k = (x_{k0}, x_{k1} \dots, x_{kn})$ .

**Our goal:** Find  $\vec{w}$  so that  $h[\vec{w}](\vec{x}) = \vec{w} \cdot \tilde{\mathbf{x}}$  approximates the function  $f$  some of whose values are given by the training set.

- ▶ **Least Squares Error Function:**

$$E(\vec{w}) = \frac{1}{2} \sum_{k=1}^p (\vec{w} \cdot \tilde{\mathbf{x}}_k - f_k)^2 = \frac{1}{2} \sum_{k=1}^p \left( \sum_{i=0}^n w_i x_{ki} - f_k \right)^2$$

## Gradient of the Error Function

Consider the **gradient** of the error function:

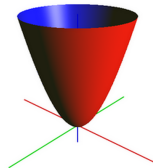
$$\nabla E(\vec{w}) = \left( \frac{\partial E}{\partial w_0}(\vec{w}), \dots, \frac{\partial E}{\partial w_n}(\vec{w}) \right) = \sum_{k=1}^p (\vec{w} \cdot \tilde{\mathbf{x}}_k - f_k) \cdot \tilde{\mathbf{x}}_k$$

What is the gradient  $\nabla E(\vec{w})$ ? It is a vector in  $\mathbb{R}^{n+1}$  which points in the direction of the steepest *ascent* of  $E$  (it's length corresponds to the steepness). Note that here the vectors  $\tilde{\mathbf{x}}_k$  are *fixed* parameters of  $E$ !

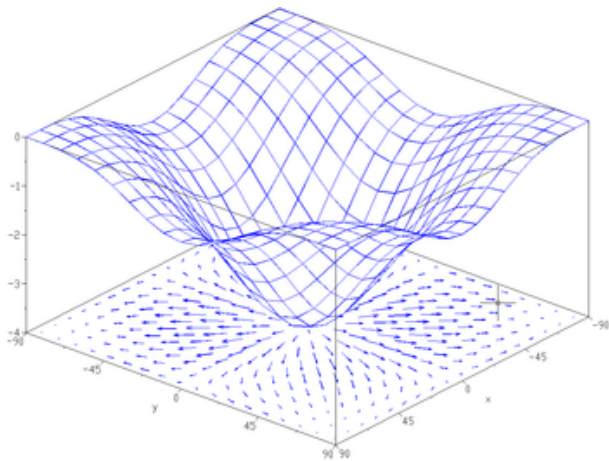
### Fakt

If  $\nabla E(\vec{w}) = \vec{0} = (0, \dots, 0)$ , then  $\vec{w}$  is a global minimum of  $E$ .

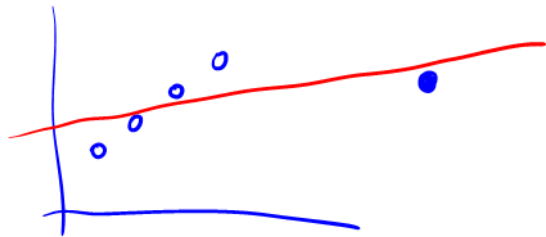
This follows from the fact that  $E$  is a convex paraboloid that has a unique extreme which is a minimum.



# Gradient – illustration



# Function Approximation – Learning



## Gradient Descent:

- ▶ Weights  $\vec{w}^{(0)}$  are initialized randomly close to  $\vec{0}$ .
- ▶ In  $(t + 1)$ -th step,  $\vec{w}^{(t+1)}$  is computed as follows:

$$\begin{aligned}\vec{w}^{(t+1)} &= \vec{w}^{(t)} - \varepsilon \cdot \nabla E(\vec{w}^{(t)}) \\ &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^p \left( \vec{w}^{(t)} \cdot \tilde{\mathbf{x}}_k - f_k \right) \cdot \tilde{\mathbf{x}}_k \\ &= \vec{w}^{(t)} - \varepsilon \cdot \sum_{k=1}^p \left( h[\vec{w}^{(t)}](\tilde{\mathbf{x}}_k) - f_k \right) \cdot \tilde{\mathbf{x}}_k\end{aligned}$$

Here  $k = (t \bmod p) + 1$  and  $0 < \varepsilon \leq 1$  is the learning speed.

Note that the algorithm is almost similar to the batch perceptron algorithm!

## Tvrzení

For sufficiently small  $\varepsilon > 0$  the sequence  $\vec{w}^{(0)}, \vec{w}^{(1)}, \vec{w}^{(2)}, \dots$  converges (component-wisely) to the global minimum of  $E$ .

# Finding the Minimum in Dimension One

Assume  $n = 1$ . Then the error function  $E$  is

$$E(w_0, w_1) = \frac{1}{2} \sum_{k=1}^p (w_0 + w_1 x_k - f_k)^2$$

Minimize  $E$  w.r.t.  $w_0$  a  $w_1$ :

$$\frac{\delta E}{\delta w_0} = 0 \quad \Leftrightarrow \quad w_0 = \bar{f} - w_1 \bar{x} \quad \Leftrightarrow \quad \bar{f} = w_0 + w_1 \bar{x}$$

where  $\bar{x} = \frac{1}{p} \sum_{k=1}^p x_k$  a  $\bar{f} = \frac{1}{p} \sum_{k=1}^p f_k$

$$\frac{\delta E}{\delta w_1} = 0 \quad \Leftrightarrow \quad w_1 = \frac{\frac{1}{p} \sum_{k=1}^p (f_k - \bar{f})(x_k - \bar{x})}{\frac{1}{p} \sum_{k=1}^p (x_k - \bar{x})^2}$$

i.e.  $w_1 = \text{cov}(f, x) / \text{var}(x)$

## Finding the Minimum in Arbitrary Dimension

Let  $A$  be a matrix  $p \times (n + 1)$  ( $p$  rows,  $n + 1$  columns) whose  $k$ -th row is the vector  $\tilde{\mathbf{x}}_k$ .

Let  $\vec{f} = (f_1, \dots, f_p)^\top$  be the *column* vector formed by values of  $f$  in the training set.

Then

$$\nabla E(\vec{w}) = 0 \quad \Leftrightarrow \quad \vec{w} = (A^\top A)^{-1} A^\top \vec{f}$$

if  $(A^\top A)^{-1}$  exists

(Then  $(A^\top A)^{-1} A^\top$  is the so called Moore-Penrose pseudoinverse of  $A$ .)

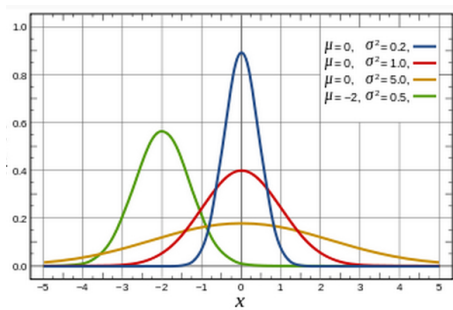
# Normal Distribution – Reminder

Distribution of continuous random variables.

Density (one dimensional, that is over  $\mathbb{R}$ ):

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(x - \mu)^2}{2\sigma^2}\right\} =: N[\mu, \sigma^2](x)$$

$\mu$  is the expected value (the mean),  $\sigma^2$  is the variance.





# Maximum Likelihood vs Least Squares (Dim 1)

**Fix a training set**  $D = \{(x_1, f_1), (x_2, f_2), \dots, (x_p, f_p)\}$

Assume that each  $f_k$  has been generated randomly by

$$f_k = (w_0 + w_1 \cdot x_k) + \epsilon_k$$

Here

- ▶  $w_0, w_1$  are **unknown numbers**
- ▶  $\epsilon_k$  are normally distributed with mean 0 and an unknown variance  $\sigma^2$

Assume that  $\epsilon_1, \dots, \epsilon_p$  were generated **independently**.

Denote by  $p(f_1, \dots, f_p \mid w_0, w_1, \sigma^2)$  the probability density according to which the values  $f_1, \dots, f_p$  were generated assuming fixed  $w_0, w_1, \sigma^2, x_1, \dots, x_p$ .

(For interested: The independence and normality imply

$$p(f_1, \dots, f_p \mid w_0, w_1, \sigma^2) = \prod_{k=1}^p N[w_0 + w_1 x_k, \sigma^2](f_k)$$

where  $N[w_0 + w_1 x_k, \sigma^2](f_k)$  is a normal distribution with the mean  $w_0 + w_1 x_k$  and the variance  $\sigma^2$ .)

# Maximum Likelihood vs Least Squares

Our goal is to find  $(w_0, w_1)$  that maximizes the likelihood that the training set  $D$  with fixed values  $f_1, \dots, f_n$  has been generated:

$$L(w_0, w_1, \sigma^2) := p(f_1, \dots, f_p \mid w_0, w_1, \sigma^2)$$

## Věta

$(w_0, w_1)$  maximizes  $L(w_0, w_1, \sigma^2)$  for arbitrary  $\sigma^2$  iff  $(w_0, w_1)$  minimizes  $E(w_0, w_1)$ , i.e. the least squares error function.

Note that the maximizing/minimizing  $(w_0, w_1)$  does not depend on  $\sigma^2$ .

Maximizing  $\sigma^2$  satisfies  $\sigma^2 = \frac{1}{p} \sum_{k=1}^p (f_k - w_0 - w_1 \cdot x_k)^2$ .

# Comments on Linear Models

- ▶ Linear models are parametric, i.e. they have a fixed form with a small number of parameters that need to be learned from data (as opposed e.g. to decision trees where the structure is not fixed in advance).
- ▶ Linear models are stable, i.e. small variations in the training data have only limited impact on the learned model. (tree models typically vary more with the training data).
- ▶ Linear models are less likely to overfit (low variance) the training data but sometimes tend to underfit (high bias).