## CS 2710 Foundations of AI
## Lecture 21

# Learning

**Milos Hauskrecht**
milos@cs.pitt.edu
5329 Sennott Square



## Types of learning

- **Supervised learning**
  - Learning mapping between inputs x and desired outputs y
  - Teacher gives me y's for the learning purposes
- **Unsupervised learning**
  - Learning relations between data components
  - No specific outputs given by a teacher
- **Reinforcement learning**
  - Learning mapping between inputs x and desired outputs y
  - Critic does not give me y's but instead a signal (reinforcement) of how good my answer was
- **Other types of learning:**
  - **Concept learning, explanation-based learning, etc.**

# Supervised learning

**Data:** $D = \{d_1, d_2, .., d_n\}$ **a set of *n* examples**

$d_i = < \mathbf{x}_i, y_i >$

$\mathbf{x}_i$ is input vector, and $y$ is desired output (given by a teacher)

**Objective:** learn the mapping $f : X \rightarrow Y$

s.t. $y_i \approx f(x_i)$ for all $i = 1,.., n$

**Two types of problems:**

- **Regression:** X discrete or continuous $\rightarrow$ Y is **continuous**
- **Classification:** X discrete or continuous $\rightarrow$ Y is **discrete**



# Supervised learning examples

- **Regression:** Y is **continuous**

Debt/equity
Earnings
Future product orders

$\longrightarrow$ company stock price

- **Classification:** Y is **discrete**

```
# ######
     ##
    ##
  ######
     ###
#     ##
# ##  ##
  #######
```

$\longrightarrow$ Label “3”

Handwritten digit (array of 0,1s)

## Unsupervised learning

- **Data:** $D = \{d_1, d_2, .., d_n\}$
  $d_i = \mathbf{x}_i$ vector of values
  No target value (output) y

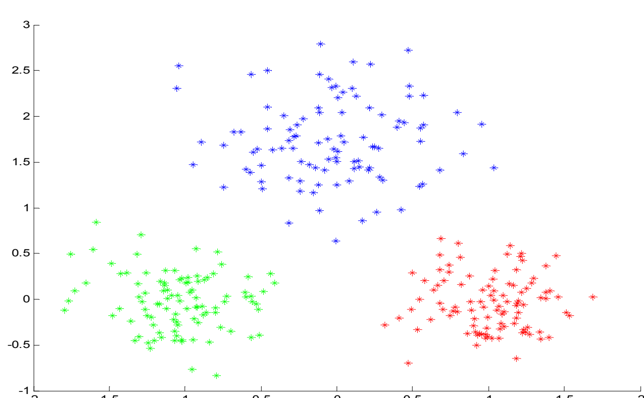
- **Objective:**
  – learn relations between samples, components of samples

**Types of problems:**
- **Clustering**
  Group together "similar" examples, e.g. patient cases
- **Density estimation**
  – Model probabilistically the population of samples, e.g. relations between the diseases, symptoms, lab tests etc.



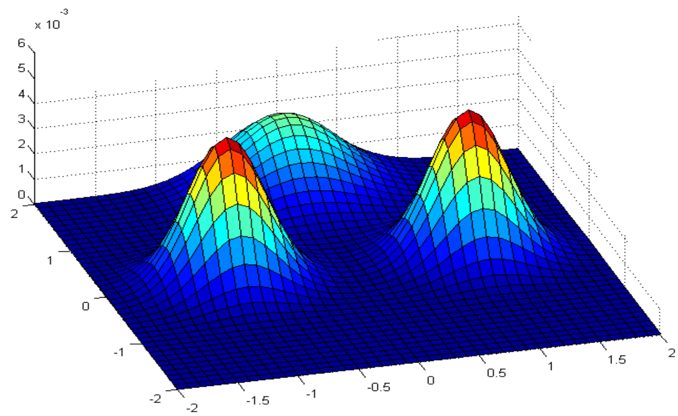
## Unsupervised learning example.

- **Density estimation.** We want to build the probability model of a population from which we draw samples $d_i = \mathbf{x}_i$



## Unsupervised learning. Density estimation

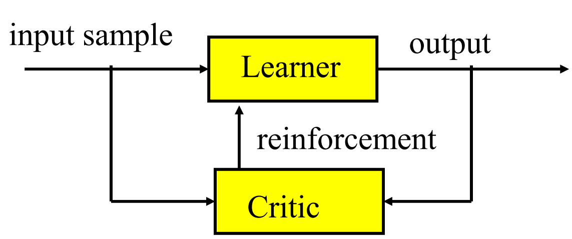
- A probability density of a point in the two dimensional space
  - Model used here: Mixture of Gaussians





## Reinforcement learning

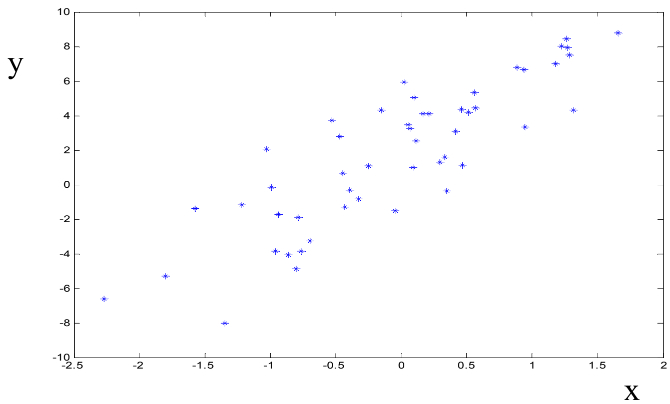
- We want to learn: $f : X \rightarrow Y$
- We see samples of $\mathbf{x}$ but not $y$
- Instead of $y$ we get a feedback (reinforcement) from a **critic** about how good our output was




- The goal is to select output that leads to the best reinforcement

# Learning

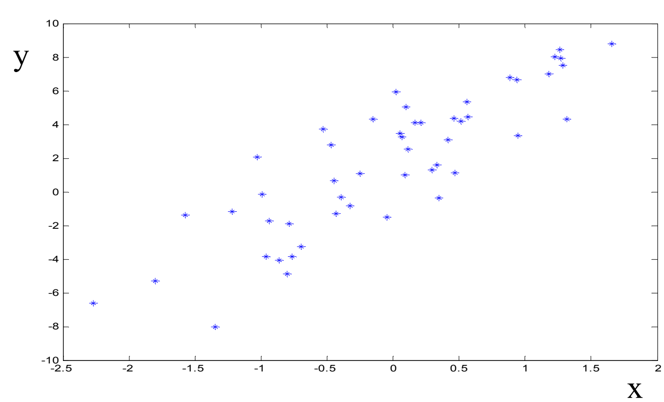
- Assume we see examples of pairs (**x** , $y$) and we want to learn the mapping $f : X \to Y$ to predict future ys for values of **x**
- We get the data what should we do?





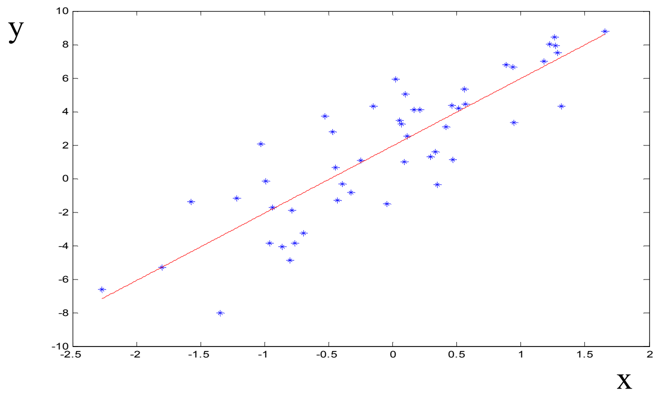

# Learning bias

- **Problem:** many possible functions $f : X \to Y$ exists for representing the mapping between **x** and y
- Which one to choose? Many examples still unseen!



# Learning bias

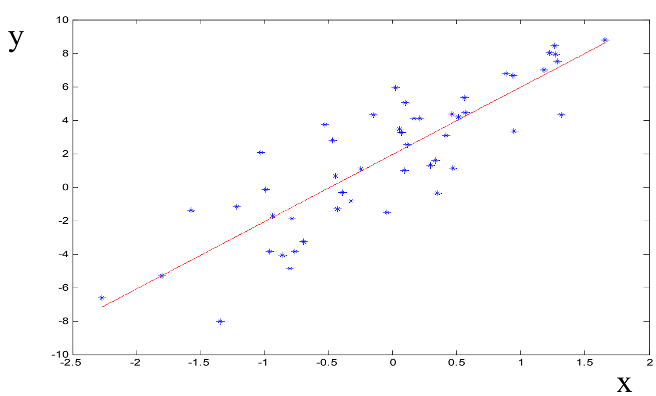
- Problem is easier when we make an assumption about the model, say, $f(x) = ax + b$
- Restriction to a linear model is an example of the learning bias





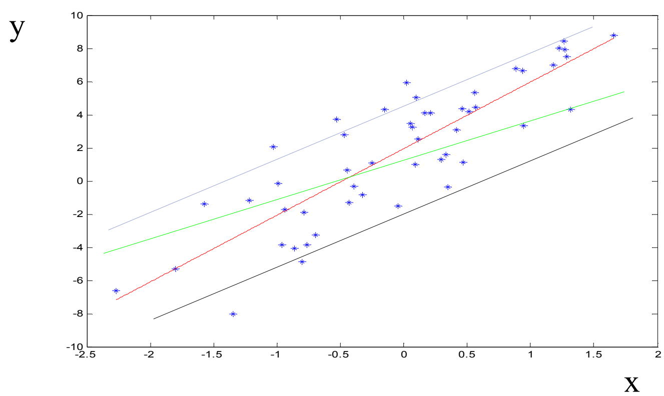

# Learning bias

- **Bias** provides the learner with some basis for choosing among possible representations of the function.
- **Forms of bias:** constraints, restrictions, model preferences
- **Important: There is no learning without a bias!**



# Learning bias

- Choosing a parametric model or a set of models is not enough
  Still too many functions $f(x) = ax + b$
  - One for every pair of parameters *a, b*






# Fitting the data to the model

We are interested in finding the **best set** of model parameters

**How is the best set defined?**

Our goal is to have the parameters that:

- reduce the misfit between the model and data
- Or, (in other words) that explain the data the best

**Error function:**

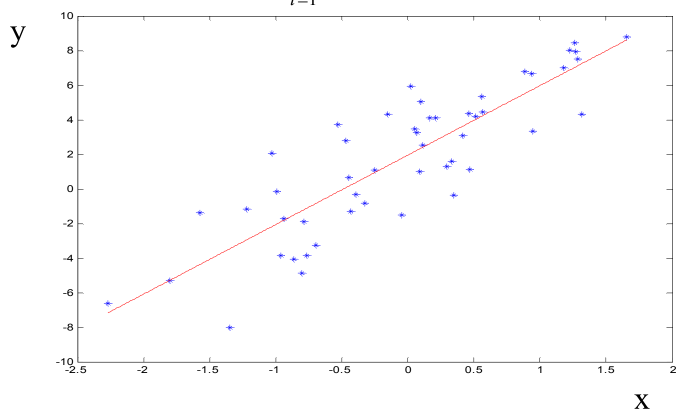
**Gives a measure of misfit between the data and the model**

- Examples of error functions:
  - Mean square error $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$
  - Misclassification error

    Average # of misclassified cases $y_i \neq f(x_i)$

## Fitting the data to the model

- **Linear regression**
  - Least squares fit with the linear model
  - minimizes $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$






## Typical learning

**Three basic steps:**

- **Select a model** or a set of models (with parameters)

  E.g. $y = ax + b$
- **Select the error function** to be optimized

  E.g. $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$
- **Find the set of parameters optimizing the error function**
  - The model and parameters with the smallest error represent the best fit of the model to the data

But there are problems one must be careful about …

# Learning

**Problem**

- We fit the model based on past experience (past examples seen)
- But ultimately we are interested in learning the mapping that performs well on the whole population of examples

**Training data:** **Data used to fit the parameters of the model**

**Training error:** $$\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

**True (generalization) error** (over the whole and not completely known population):

$$E_{(x,y)}(y - f(x))^2 \qquad \text{Expected squared error}$$

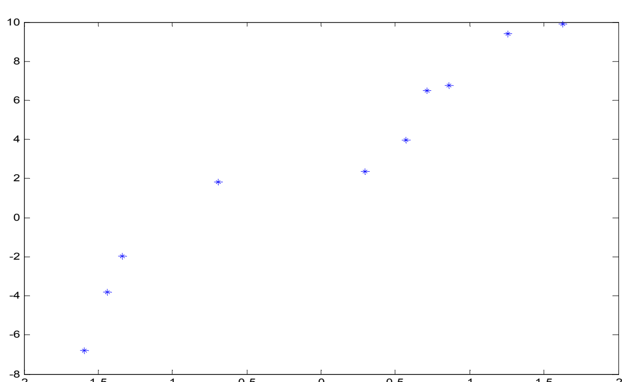
The training error tries to approximate the true error.

**But does a good training error always imply a good generalization error?**



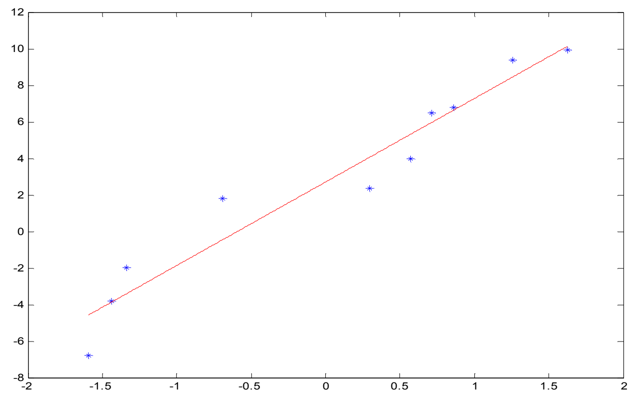
# Overfitting

- Assume we have a set of 10 points and we consider polynomial functions as our possible models



# Overfitting

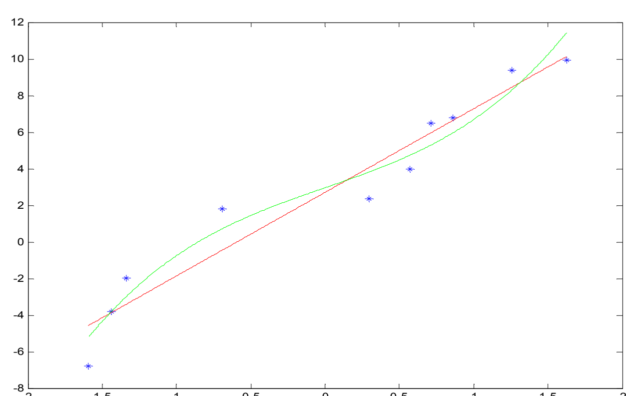
- Fitting a linear function with mean-squares error
- Error is nonzero





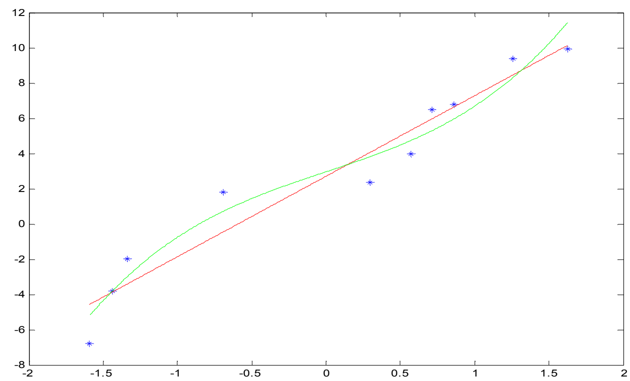

# Overfitting

- Linear vs. cubic polynomial
- Higher order polynomial leads to a better fit, smaller error



## Overfitting

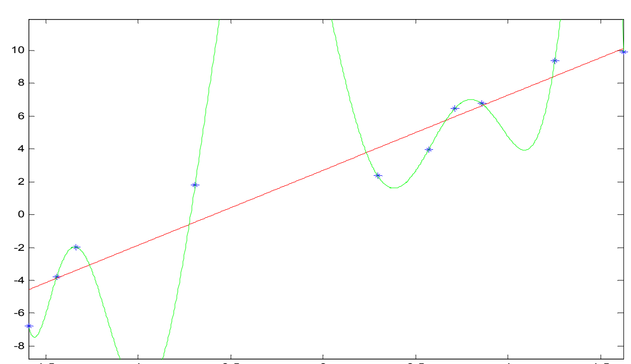
- Is it always good to minimize the error of the observed data?





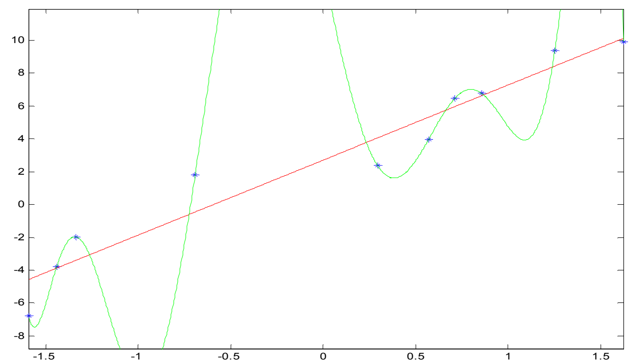

## Overfitting

- For 10 data points, degree 9 polynomial gives a perfect fit (Lagrange interpolation). Error is zero.
- Is it always good to minimize the training error?



## Overfitting

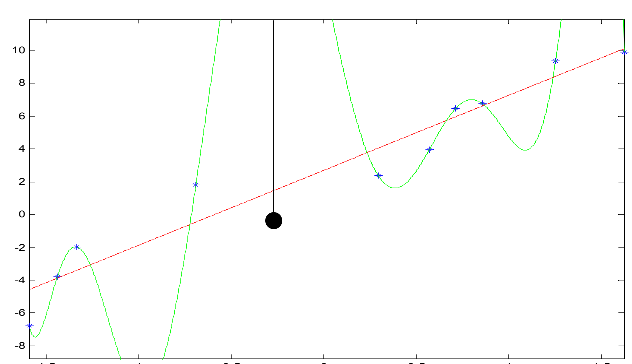
- For 10 data points, degree 9 polynomial gives a perfect fit (Lagrange interpolation). Error is zero.
- Is it always good to minimize the training error? **NO !!**
- More important: How do we perform on the unseen data?





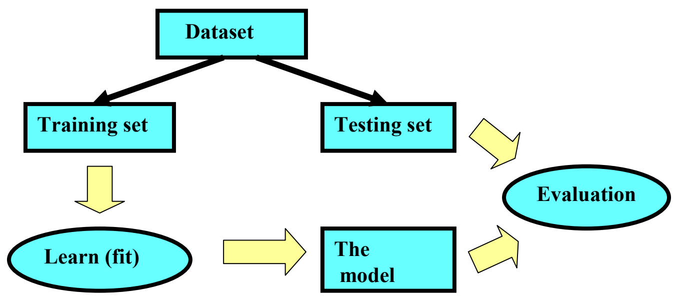

## Overfitting

- The situation when the training error is low and the generalization error is high. Causes of the phenomenon:
  - Model with more degrees of freedom (more parameters)
  - Small data size (as compared to the complexity of model)



# Evaluation framework

- We want our classifier to generalize well to future examples
- **Problem:** But we do not know all future examples !!!
- **Solution:** evaluate the classifier on **the test set** that is withheld from the learning stage






# How to evaluate the learner's performance?

- **Generalization error** is the true error for the population of examples we would like to optimize

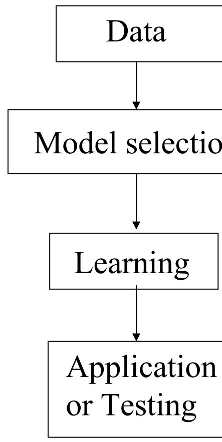
$$E_{(x,y)}(y - f(x))^2$$

  – **But it cannot be computed exactly**
- **Optimizing (mean) training error can lead to overfit, i.e.** training error may not reflect properly the generalization error

$$\frac{1}{n}\sum_{i=1,..n}(y_i - f(x_i))^2$$

- **The generalization error is more objectively estimated using a separate test data set with *m* data samples**
- **(Mean) test error** $\frac{1}{m}\sum_{j=1,..m}(y_j - f(x_j))^2$

# Design of a learning system






# Design of a learning system

**1. Data:** $D = \{d_1, d_2, .., d_n\}$

**2. Model selection:**

- **Select a model** or a set of models (with parameters)

  E.g. $y = ax + b$

- **Select the error function** to be optimized

  E.g. $\frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

**3. Learning:**

- **Find the set of parameters optimizing the error function**
  - The model and parameters with the smallest error

**4. Application:**

- **Apply the learned model**
  - E.g. predict ys for new inputs **x** using learned $f(\mathbf{x})$

# Linear regression.



## Supervised learning

**Data:** $D = \{D_1, D_2, .., D_n\}$ **a set of *n* examples**

$D_i =< \mathbf{x}_i, y_i >$

$\mathbf{x}_i = (x_{i,1}, x_{i,2}, \cdots x_{i,d})$ is an input vector of size $d$

$y_i$ is the desired output (given by a teacher)

**Objective:** learn the mapping $f : X \rightarrow Y$

s.t. $y_i \approx f(\mathbf{x}_i)$ for all $i = 1, .., n$

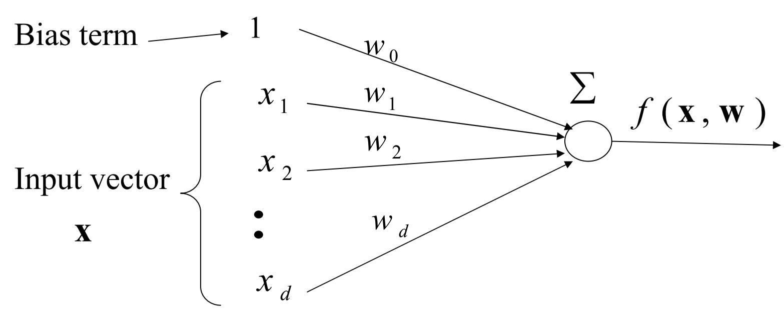
- **Regression:** Y is **continuous**
  Example: earnings, product orders $\rightarrow$ company stock price
- **Classification:** Y is **discrete**
  Example: handwritten digit in binary form $\rightarrow$ digit label

# Linear regression

- **Function** $f : X \rightarrow Y$ is a linear combination of input components

$$f(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots w_d x_d = w_0 + \sum_{j=1}^{d} w_j x_j$$

$w_0, w_1, \ldots w_k$ **- parameters (weights)**





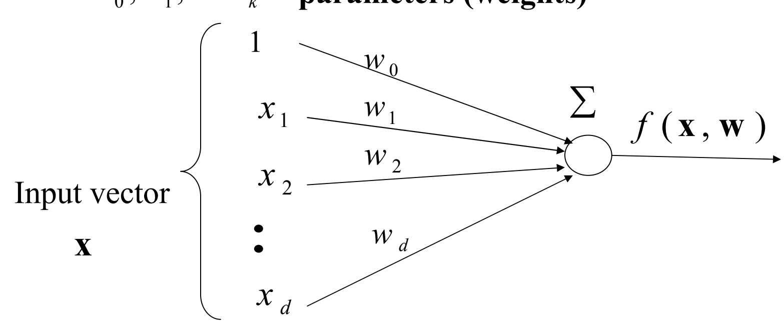

# Linear regression

- **Shorter (vector) definition of the model**
  - Include bias constant in the input vector

    $\mathbf{x} = (1, x_1, x_2, \cdots x_d)$

$$f(\mathbf{x}) = w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots w_d x_d = \mathbf{w}^T \mathbf{x}$$

$w_0, w_1, \ldots w_k$ **- parameters (weights)**



## Linear regression. Error.

- **Data:** $D_i =< \mathbf{x}_i, y_i >$
- **Function:** $\mathbf{x}_i \rightarrow f(\mathbf{x}_i)$
- We would like to have $y_i \approx f(\mathbf{x}_i)$ for all $i = 1,.., n$

- **Error function**
  - measures how much our predictions deviate from the desired answers

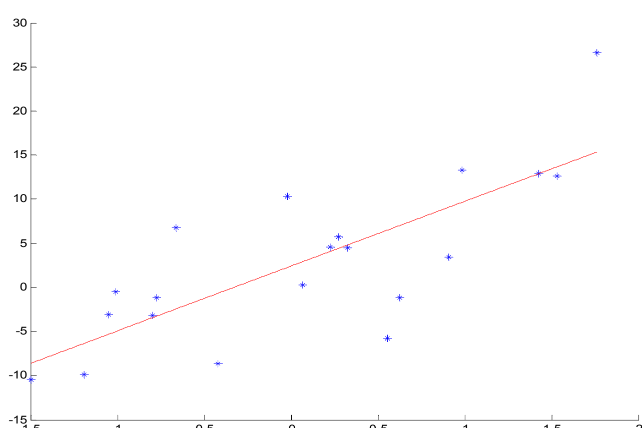
**Mean-squared error** $$J_n = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))^2$$

- **Learning:**
  **We want to find the weights minimizing the error !**



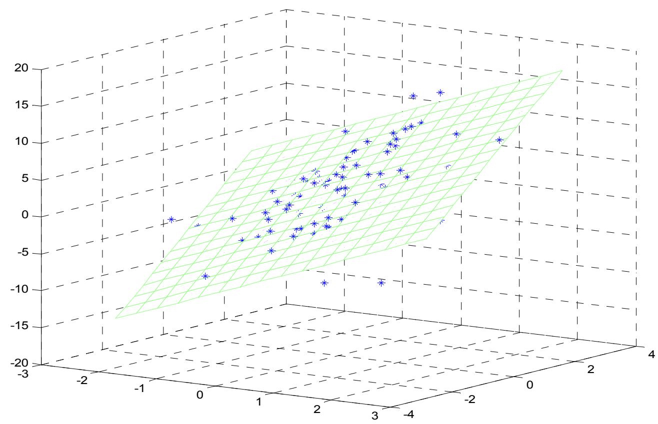
## Linear regression. Example

- 1 dimensional input $\mathbf{x} = (x_1)$



# Linear regression. Example.

- 2 dimensional input $\mathbf{x} = (x_1, x_2)$






# Linear regression. Optimization.

- We want the **weights minimizing the error**

$$J_n = \frac{1}{n} \sum_{i=1,..n} (y_i - f(\mathbf{x}_i))^2 = \frac{1}{n} \sum_{i=1,..n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

- For the optimal set of parameters, derivatives of the error with respect to each parameter must be 0

$$\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,j} = 0$$

- **Vector of derivatives:**

$$\text{grad}_{\mathbf{w}}(J_n(\mathbf{w})) = \nabla_{\mathbf{w}}(J_n(\mathbf{w})) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i) \mathbf{x}_i = \overline{\mathbf{0}}$$

## Linear regression. Optimization.

- $\mathrm{grad}_{\mathbf{w}}(J_n(\mathbf{w})) = \overline{\mathbf{0}}$ defines a set of equations in **w**

$$\frac{\partial}{\partial w_0} J_n(\mathbf{w}) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) = 0$$

$$\frac{\partial}{\partial w_1} J_n(\mathbf{w}) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,1} = 0$$

$$\ldots$$

$$\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,j} = 0$$

$$\ldots$$

$$\frac{\partial}{\partial w_d} J_n(\mathbf{w}) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,d} = 0$$



## Solving linear regression

$$\frac{\partial}{\partial w_j} J_n(\mathbf{w}) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - w_0 x_{i,0} - w_1 x_{i,1} - \ldots - w_d x_{i,d}) x_{i,j} = 0$$

By rearranging the terms we get a **system of linear equations** with *d+1* unknowns

$$\boxed{\mathbf{A}\mathbf{w} = \mathbf{b}}$$

$$w_0 \sum_{i=1}^{n} x_{i,0} 1 + w_1 \sum_{i=1}^{n} x_{i,1} 1 + \ldots + w_j \sum_{i=1}^{n} x_{i,j} 1 + \ldots + w_d \sum_{i=1}^{n} x_{i,d} 1 = \sum_{i=1}^{n} y_i 1$$

$$w_0 \sum_{i=1}^{n} x_{i,0} x_{i,1} + w_1 \sum_{i=1}^{n} x_{i,1} x_{i,1} + \ldots + w_j \sum_{i=1}^{n} x_{i,j} x_{i,1} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} x_{i,1} = \sum_{i=1}^{n} y_i x_{i,1}$$

$$\bullet\bullet\bullet$$

$$w_0 \sum_{i=1}^{n} x_{i,0} x_{i,j} + w_1 \sum_{i=1}^{n} x_{i,1} x_{i,j} + \ldots + w_j \sum_{i=1}^{n} x_{i,j} x_{i,j} + \ldots + w_d \sum_{i=1}^{n} x_{i,d} x_{i,j} = \sum_{i=1}^{n} y_i x_{i,j}$$

$$\bullet\bullet\bullet$$

## Solving linear regression

- The optimal set of weights satisfies:

$$\nabla_{\mathbf{w}}(J_n(\mathbf{w})) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i = \bar{\mathbf{0}}$$

Leads to a **system of linear equations (SLE)** with $d+1$ unknowns of the form

$$\mathbf{A}\mathbf{w} = \mathbf{b}$$

$$w_0\sum_{i=1}^{n} x_{i,0}x_{i,j} + w_1\sum_{i=1}^{n} x_{i,1}x_{i,j} + \ldots + w_j\sum_{i=1}^{n} x_{i,j}x_{i,j} + \ldots + w_d\sum_{i=1}^{n} x_{i,d}x_{i,j} = \sum_{i=1}^{n} y_i x_{i,j}$$

**Solution to SLE: ?**



## Solving linear regression

- The optimal set of weights satisfies:

$$\nabla_{\mathbf{w}}(J_n(\mathbf{w})) = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \mathbf{w}^T\mathbf{x}_i)\mathbf{x}_i = \bar{\mathbf{0}}$$

Leads to a **system of linear equations (SLE)** with $d+1$ unknowns of the form

$$\mathbf{A}\mathbf{w} = \mathbf{b}$$

$$w_0\sum_{i=1}^{n} x_{i,0}x_{i,j} + w_1\sum_{i=1}^{n} x_{i,1}x_{i,j} + \ldots + w_j\sum_{i=1}^{n} x_{i,j}x_{i,j} + \ldots + w_d\sum_{i=1}^{n} x_{i,d}x_{i,j} = \sum_{i=1}^{n} y_i x_{i,j}$$

**Solution to SLE:**

$$\boxed{\mathbf{w} = \mathbf{A}^{-1}\mathbf{b}}$$

- matrix inversion

# Gradient descent solution

**Goal:** the weight optimization in the linear regression model

$$J_n = Error(\mathbf{w}) = \frac{1}{n}\sum_{i=1,..n}(y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

An alternative to SLE solution:

- **Gradient descent**

  **Idea:**
  - Adjust weights in the direction that improves the Error
  - The gradient tells us what is the right direction

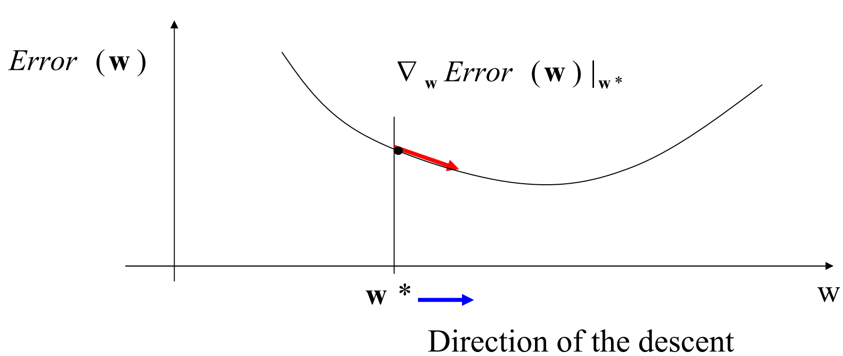
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

$\alpha > 0$ - a **learning rate** (scales the gradient changes)



# Gradient descent method

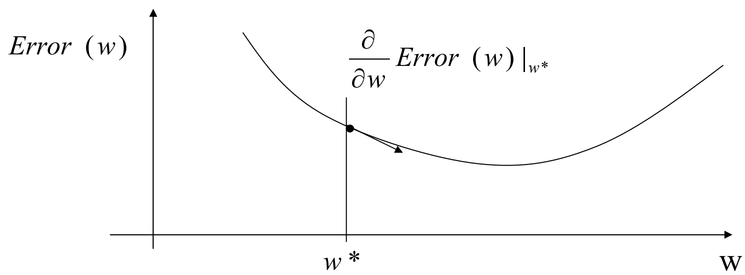
- Descend using the gradient information




- Change the value of **w** according to the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

# Gradient descent method




- New value of the parameter

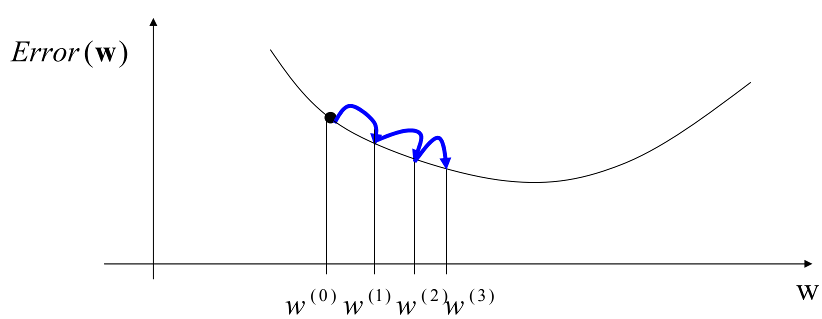
$$w_j \leftarrow w_j^* - \alpha \frac{\partial}{\partial w_j} Error(w)|_{w^*} \qquad \text{For all j}$$

$\alpha > 0$ - a learning rate (scales the gradient changes)



# Gradient descent method

- Iteratively converge to the optimum of the Error function



# Online gradient algorithm

- The error function is defined for the whole dataset $D$

$$J_n = Error(\mathbf{w}) = \frac{1}{n}\sum_{i=1,..n}(y_i - f(\mathbf{x}_i,\mathbf{w}))^2$$

- **error for a sample** $D_i = <\mathbf{x}_i, y_i>$

$$J_{\text{online}} = Error_i(\mathbf{w}) = \frac{1}{2}(y_i - f(\mathbf{x}_i,\mathbf{w}))^2$$

- **Online gradient method: changes weights after every sample**

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} Error_i(\mathbf{w})$$

- **vector form:**

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} Error_i(\mathbf{w})$$

$\alpha > 0$ - Learning rate that depends on the number of updates



# Online gradient method

Linear model $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

On-line error $J_{online} = Error_i(\mathbf{w}) = \frac{1}{2}(y_i - f(\mathbf{x}_i,\mathbf{w}))^2$

**On-line algorithm:** generates a sequence of online updates

**(i)-th update step with :** $D_i = <\mathbf{x}_i, y_i>$

**j-th weight:**

$$w_j^{(i)} \leftarrow w_j^{(i-1)} - \alpha(i)\frac{\partial Error_i(\mathbf{w})}{\partial w_j}\Big|_{\mathbf{w}^{(i-1)}}$$

$$\boxed{w_j^{(i)} \leftarrow w_j^{(i-1)} + \alpha(i)(y_i - f(\mathbf{x}_i, \mathbf{w}^{(i-1)}))x_{i,j}}$$

**Annealed learning rate:** $\alpha(i) \approx \frac{1}{i}$

- Gradually rescales changes in weights

# Online regression algorithm

**Online-linear-regression** (*D, number of iterations*)

  **Initialize** weights $\mathbf{w} = (w_0, w_1, w_2 \ldots w_d)$

  **for** $i$=1:1*: number of iterations*

    **do**   **select** a data point $D_i = (\mathbf{x}_i, y_i)$ from $D$

      **set** $\alpha = 1/i$

      **update** weight vector

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha(y_i - f(\mathbf{x}_i, \mathbf{w}))\mathbf{x}_i$$

  **end for**

  **return** weights $\mathbf{w}$

- **Advantages:** very easy to implement, continuous data streams



# On-line learning. Example