# *Intelligent systems for industry, supply chain and environment*

# LESSON 16

*In case of new models.*

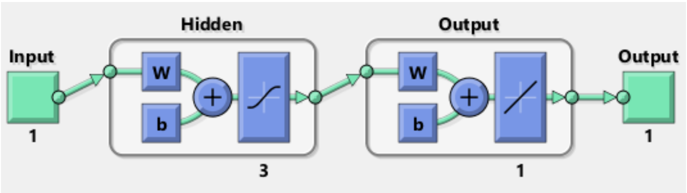
*Linear regressors and classifiers (with coding)*




# Outline

- New model?
  - What to know
  - What to do

- Creation and use (with code) of the first complete machine learning models
  - Linear regressors
  - Linear classifiers
  - First neural networks

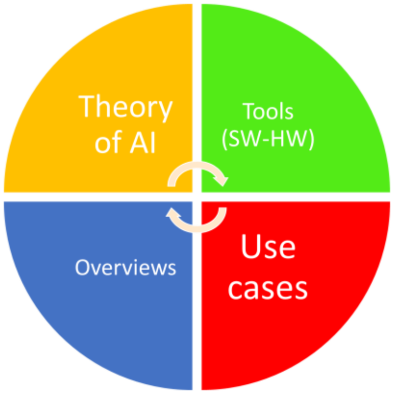



- Main points

# THEORY

# New model?

Things to know for a correct start



# *What to know*
# *in case of new models?*

- *Everyday, new models/learning techniques will be available. How to face this challenge and keep the pace?*
- *Consult documentation and examples before to waste time*
  - *Understand the topic, the type, the context!*
  - *Check on what libraries models be used*
  - *General or specific?*
  - *What the computational complexity of the model?*
    - *In train*
    - *In test/deployment*
    - *Need CUDA?*

«Let's use it, we don't need to read documentation!»

# *What to* ***know*** *in case of new models?*

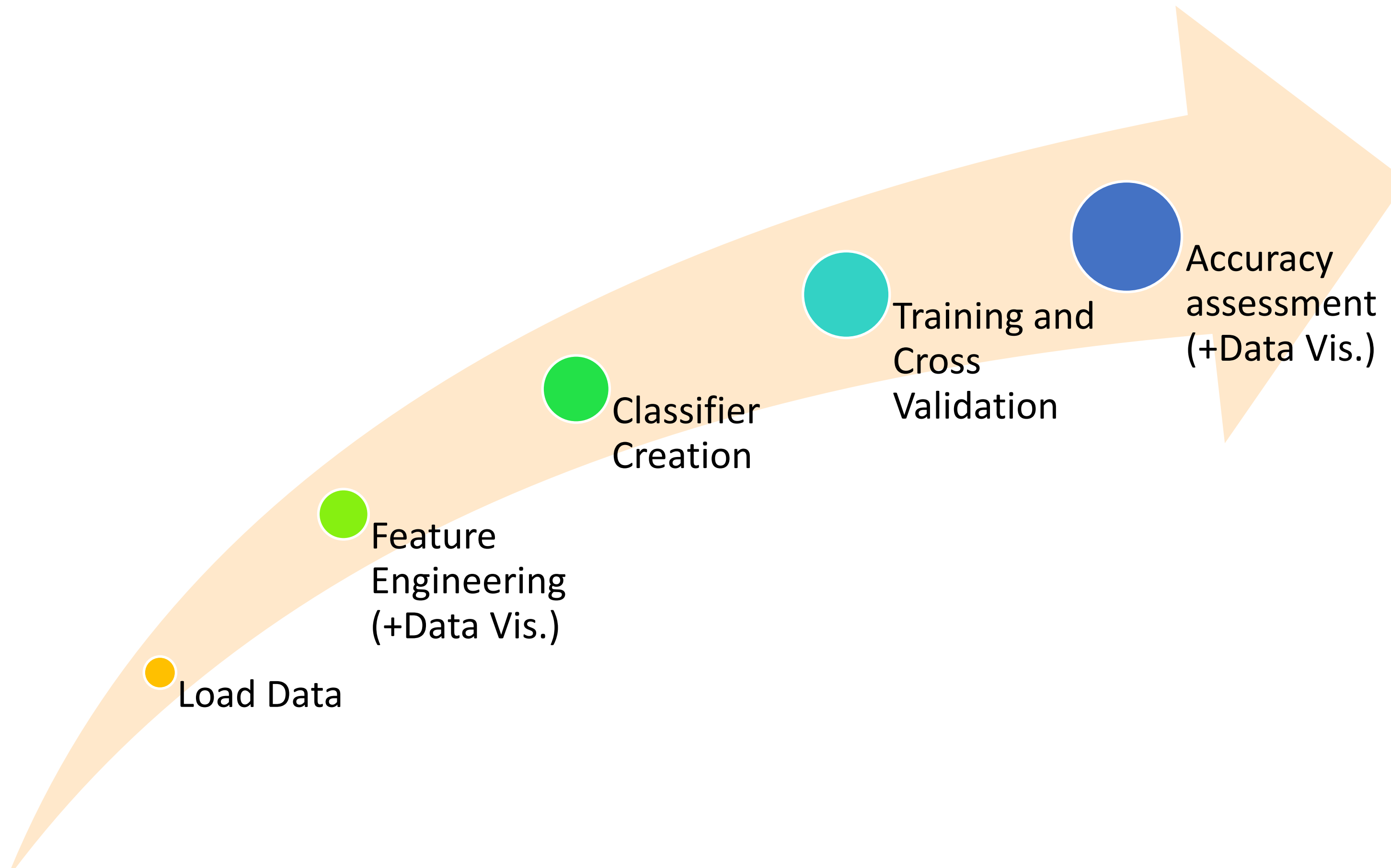
- *Consult documentation and examples before to waste time (cont…)*
  - *Memory needs*
    - *Training*
    - *Test*
  - *Examples/Dataset size*

    *1000 sample or 10M samples?*
  - *Parameters*
    - *Complex parameters to understand?*
    - *Sensitive to parameter tuning?*
  - *Can you do just a fine-tuning or need to re-train?*
  - *Portability on different libraries/environments*
    - *Is it open/GNU/free or just one single company are running it?*

# *What to do in case of new models?*

- *Try demos/examples in the documentation before to waste time, checking if it is/not suitable for your applications*
- *Try new models on a set of standard datasets and compare the accuracy and computational times*
    - *Not just the IRIS dataset..*
- *Remember to Occam's razor!*
    - *Same performance, but more complex, does it worth it?*

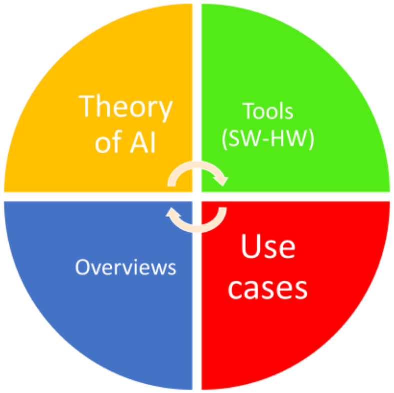
# Application of a new classifier



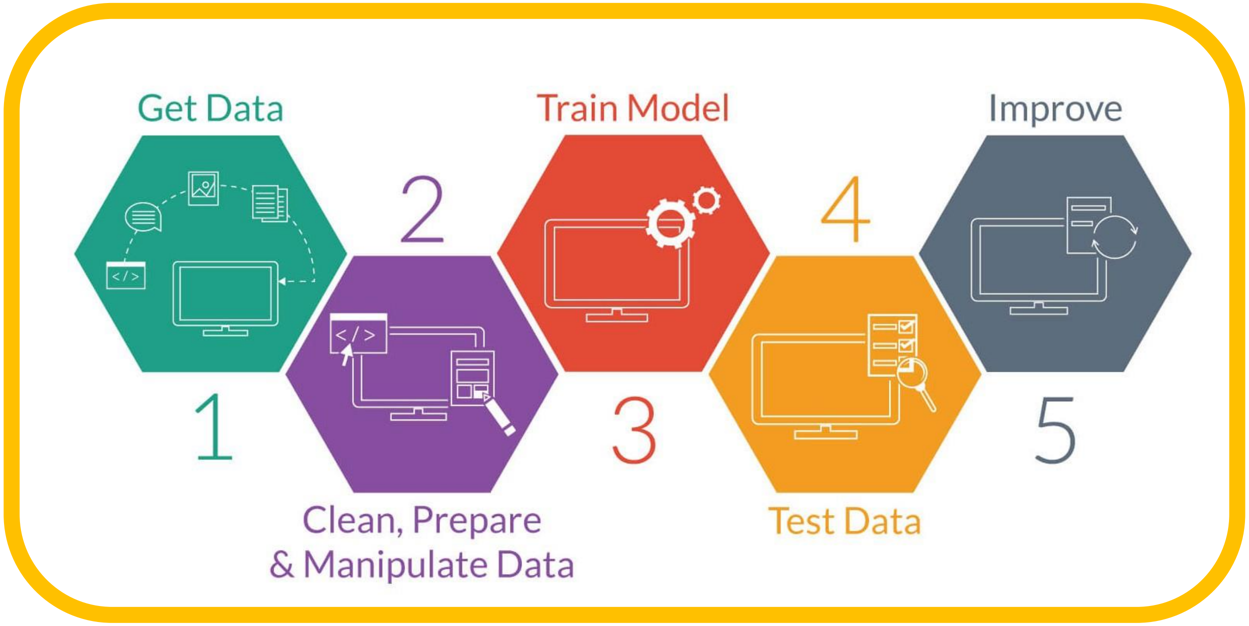
# THEORY

# A simple example of the complete workflow

The linear regressor

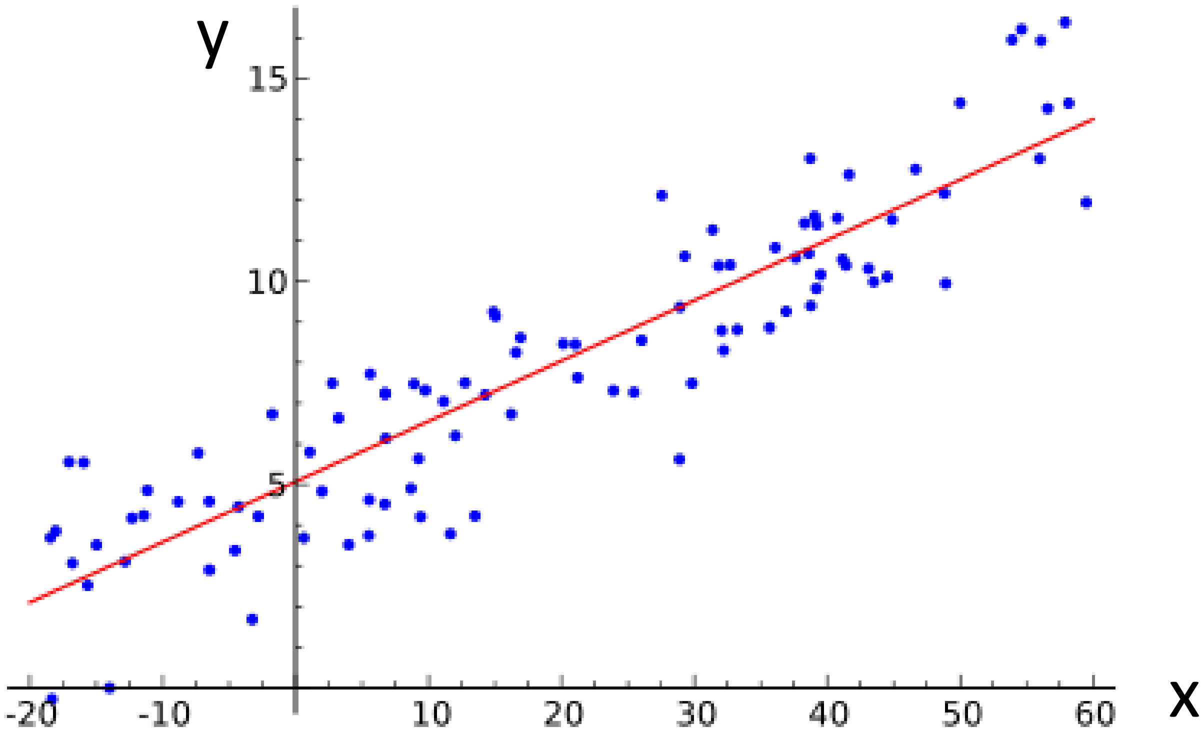


# Let’s review all the steps of the desing workflow with a simple example



# *Linear Regression*

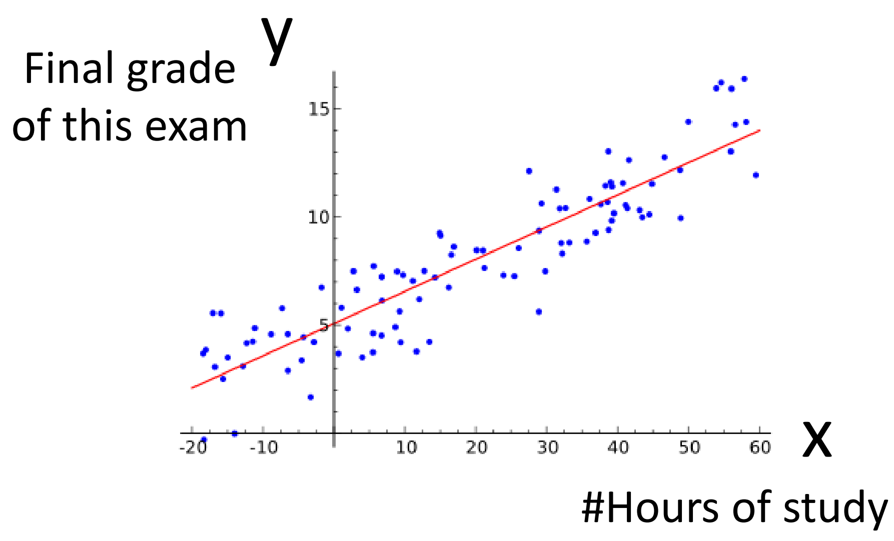
We want to find the best linear function y=f(x) to explain the data we have



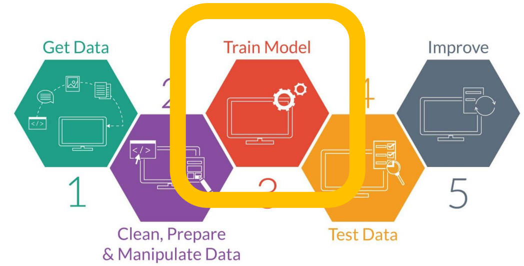
# *Linear Regression examples*

- Age, gender, and diet → height of the kid
- Type and grades of college → yearly income

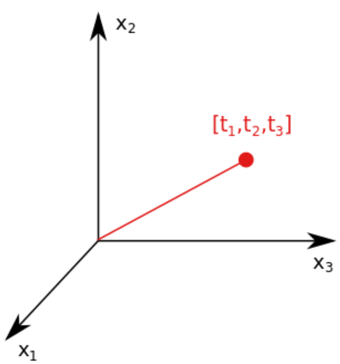
- One more..



# An interesting example…
# Extendible to N features




Representation


Vectorial form

$$\mathbf{z} = \mathbf{w} \cdot \mathbf{x} + \mathbf{b}$$

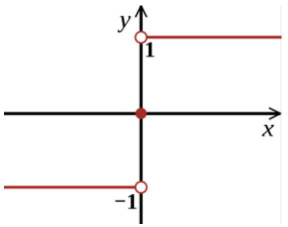
$$z = [w_1\ w_2\ w_3] \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + b$$

To completely describe the model you need to fix 4 parameters: w1, w2, w3, b.

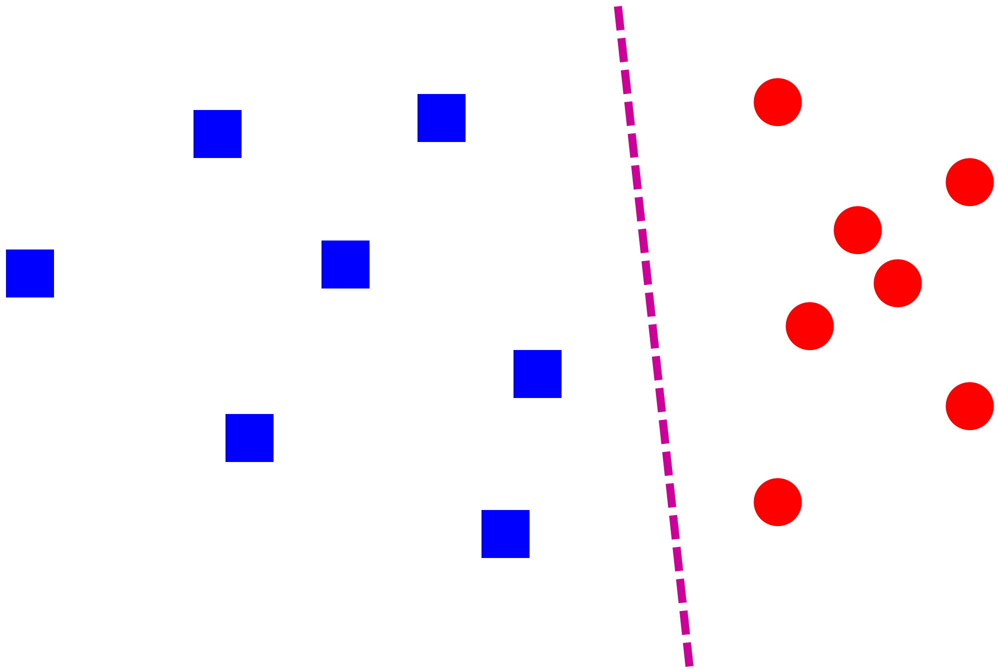
→ You need 4 data points

But you will use much more data!

# From regressor to classifiers: Linear Class.




$$\operatorname{sgn}(x) := \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0. \end{cases}$$

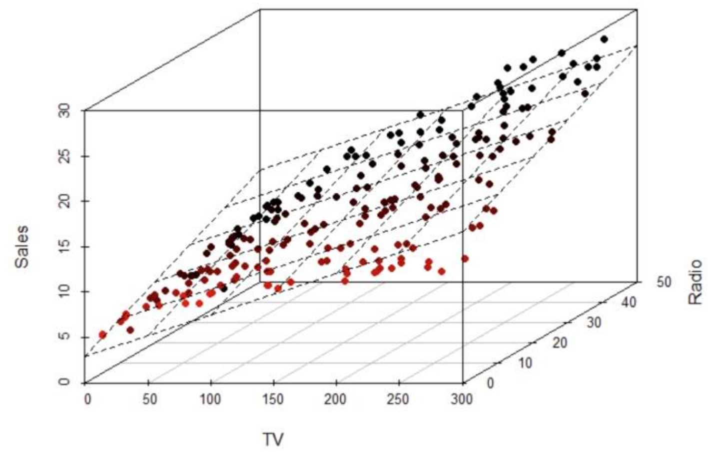


*Problem*

*Find a linear function to separate the classes:*

$$f(\mathbf{x}) = sgn(\underbrace{\mathbf{w} \cdot \mathbf{x} + b}_{\text{Linear regressor}})$$

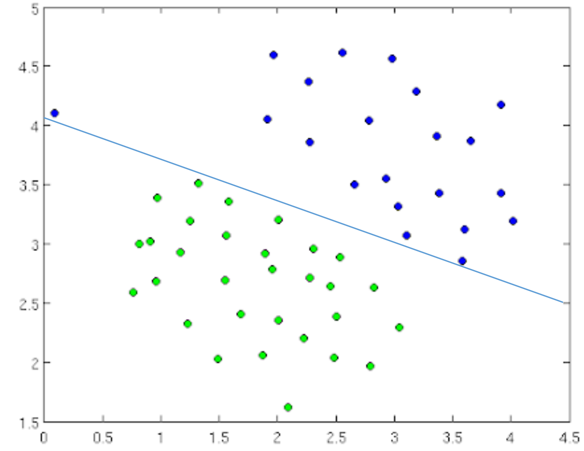
2D example




# Linear regressor

$$f1(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

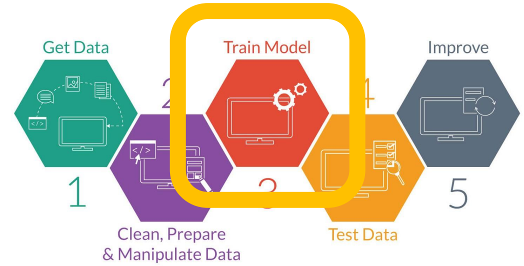
2D example



# Linear classifier:

$$f2(\mathbf{x}) = \text{sgn}(\underbrace{\mathbf{w} \cdot \mathbf{x} + b}_{\text{Linear regressor}})$$

# *Linear Regression*




**Representation**

The predicted value of y is given by:

$$\hat{y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$$

The vector of coefficients $\hat{\beta}$ is the regression model.

JUST A NOTE about computation:

If you add a term in the inputs $X_0 = 1$, the formula becomes a simple matrix product (a compact notation): $\hat{\boldsymbol{y}} = \mathbf{X}\,\widehat{\boldsymbol{\beta}}$

The algebra to get the solution is more easy to deal with.

# About notation...

Elements

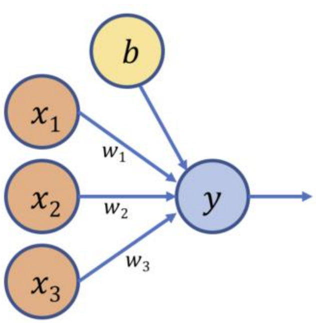
$$\hat{y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j$$

Vectorial form

$$z = \mathbf{w} \cdot \mathbf{x} + b$$

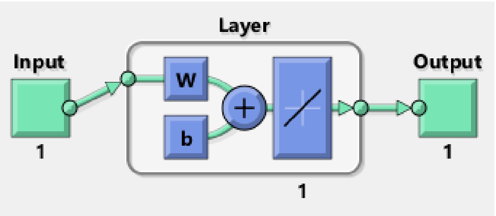
«**weights**»
Since they multiply the information coming from the inputs

«**bias**»
Something always present, not modulated from the input




3 inputs

That is (also) the linear output of a neuron



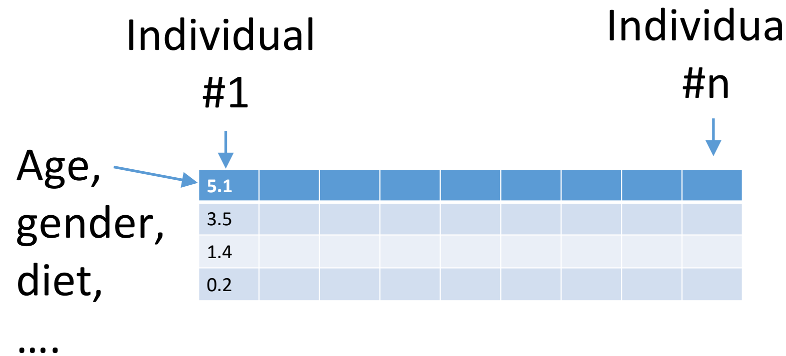

1 inputs

# *Linear Regression, #Features>1*

$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$

We can write all of the input samples fro the training in a single matrix **X**:

$$\mathbf{X} = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix}$$



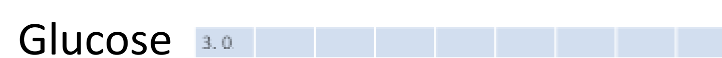

n = # samples (**distinct observations)**

m = # features

Vector **Y** does not contain classes here!!

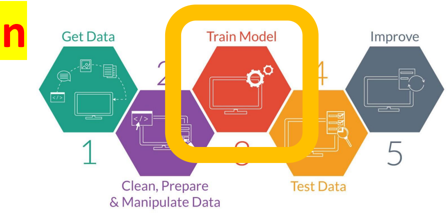
We are creating a regressor →we will have <u>numbers</u> in **Y**

$$\mathbf{y} = (y_1, \ y_2, \dots, \ y_n)$$



# Residual Sum-of-Squares

Optimization




To determine the model parameters $\hat{\beta}$ from some data, we minimize the Residual Sum of Squares which is

$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - \beta x_i)^2$$

---

Appendix (non in the exam) using the «compact notation» :

or symbolically $\mathrm{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$. To minimize it, take the derivative w.r.t. $\beta$ which gives:

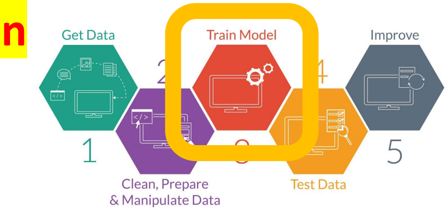
$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0$$

And if $\mathbf{X}^T\mathbf{X}$ is non-singular, the unique solution is:

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

# Iterative Regression Solutions

**Optimization**




Appendix (not in the exam):

- The exact method requires us to invert a matrix $\left(\mathbf{X}^T\mathbf{X}\right)^{-1}$ This will often be **too big**.

- There are many gradient-based methods which reduce the RSS error by taking the **derivative wrt** $\beta$

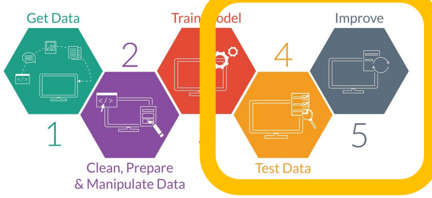
$$\mathrm{RSS}(\beta) = \sum_{i=1}^{N} (y_i - \beta x_i)^2$$

which was

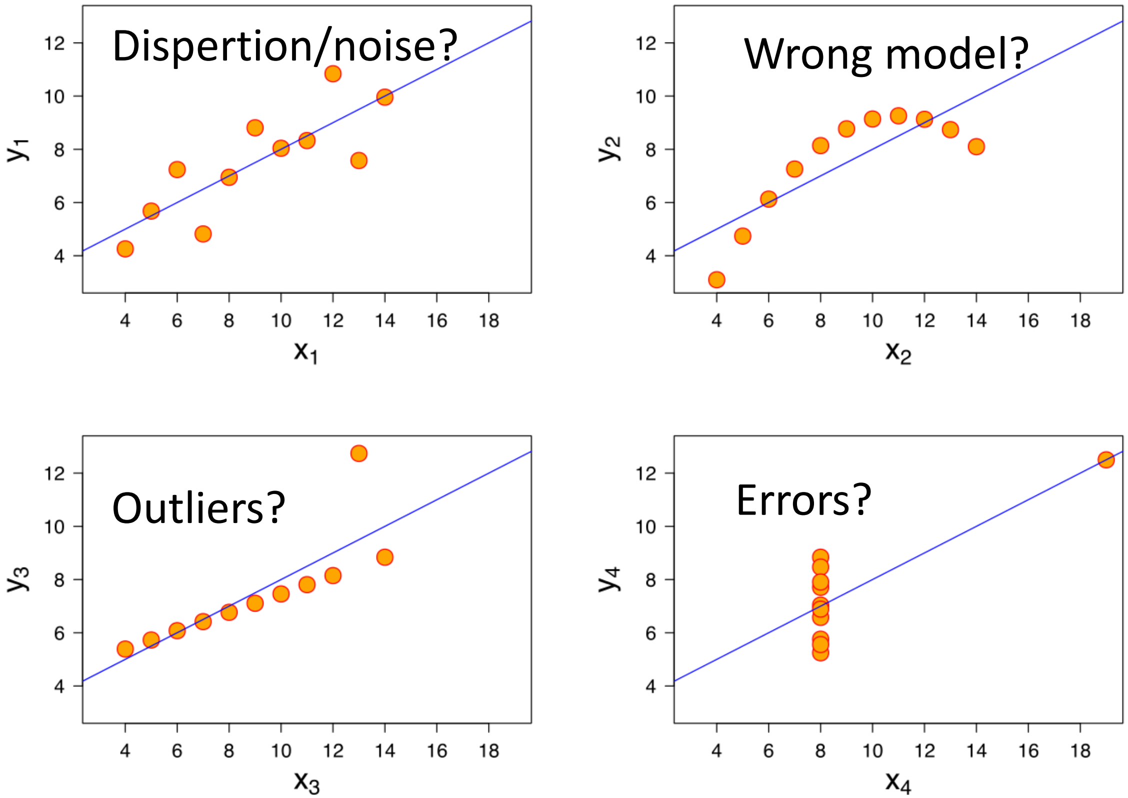
$$\nabla = \mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta)$$

Important note!
Many neural method (even in deep learning) are using gradient-based methods.
Il the gradient → 0 the method "gets stuck"

# *Is the regression good?* *$R^2$-value*




- We can **always** fit a linear model to any dataset, but how do we know if there is a **real linear relationship**?




$R^2$ = 0.67 for all
Dim > 2 it is hard to visually understand the fitting quality!

| Property | Value | Accuracy |
|---|---|---|
| Mean of $x$ | 9 | exact |
| Sample variance of $x$ : $\sigma^2$ | 11 | exact |
| Mean of $y$ | 7.50 | to 2 decimal places |
| Sample variance of $y$ : $\sigma^2$ | 4.125 | ±0.003 |
| Correlation between $x$ and $y$ | 0.816 | to 3 decimal places |
| Linear regression line | $y = 3.00 + 0.500x$ | to 2 and 3 decimal places, respectively |
| Coefficient of determination of the linear regression : $R^2$ | 0.67 | to 2 decimal places |

Anscombe's quartet: four data sets that have nearly identical simple descriptive statistics

# *R2-values* (***R-squared***)

*Let $\hat{y}$ be a predicted value, and $\bar{y}$ be the sample mean. Then the R-squared statistic is*

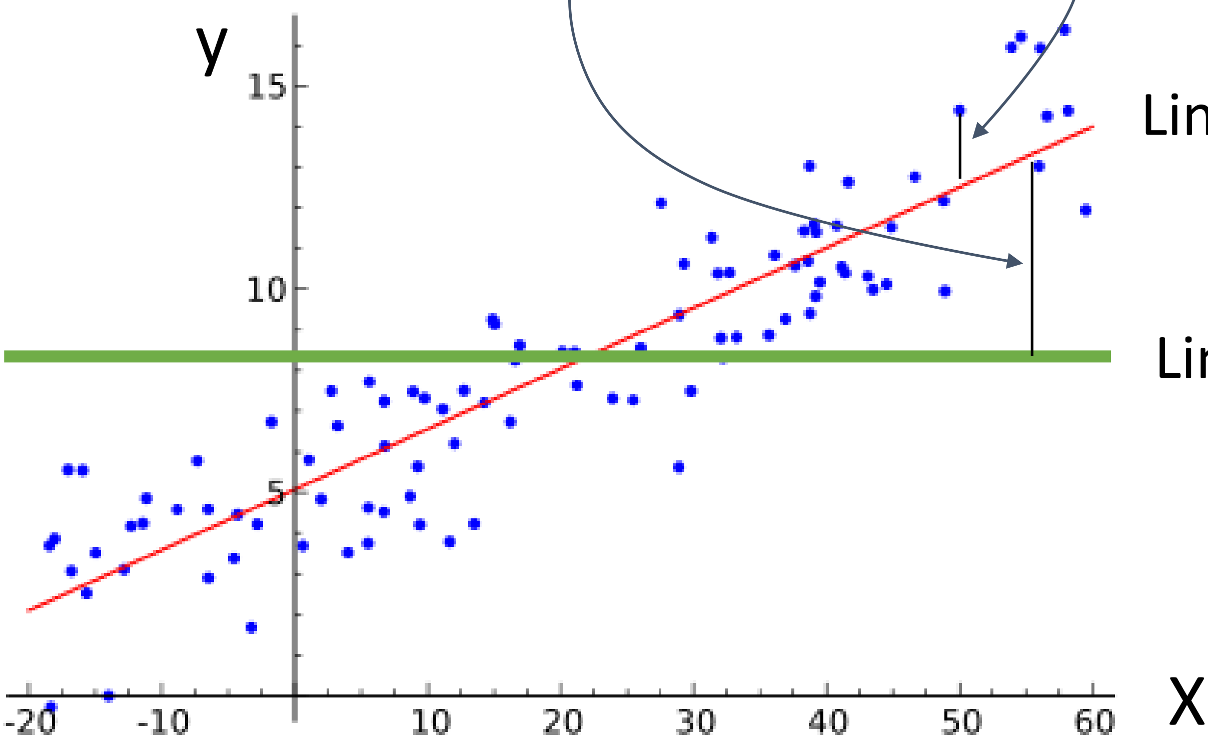
$$R^2 = 1 - \frac{\sum(y_i - \widehat{y_i})^2}{\sum(y_i - \bar{y})^2}$$

*And can be described as the fraction of the total variance not explained by the model.*

# *R-squared*

**→ 1 means good fit**

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2} = \frac{\text{sum of squares of residuals}}{\text{total sum of squares (proportional to the variance of the data)}}$$




Line of $\hat{y}$

Line of $\bar{y}$ (mean of data)

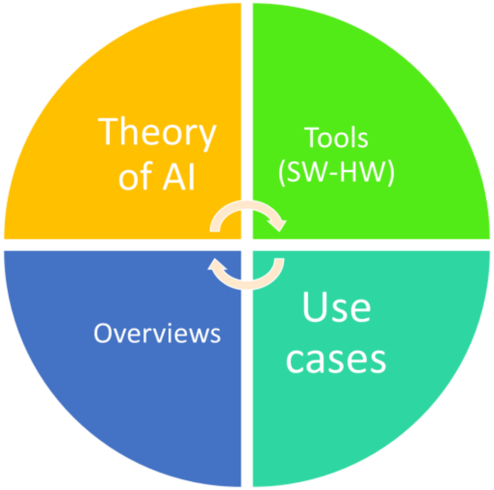
$R^2$ It must be reported!
When Dim >2
it is quite important
(you can't see…)

# Toolboxes

## Matlab and Python

Linear regressor

Liner classifiers

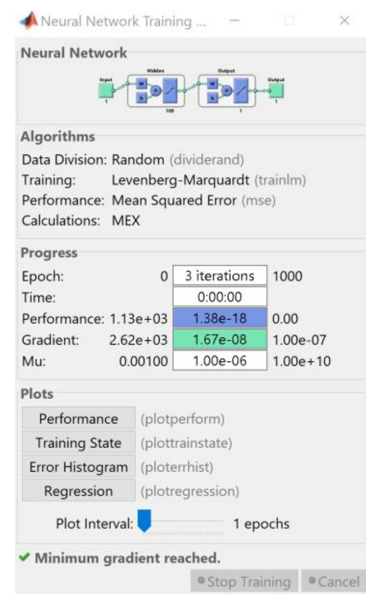


# *Matlab file*

- *1) Follow next slides*
- *2) and then lunch your scritp.*

laboratory_MATLAB_linear_and_neural.m

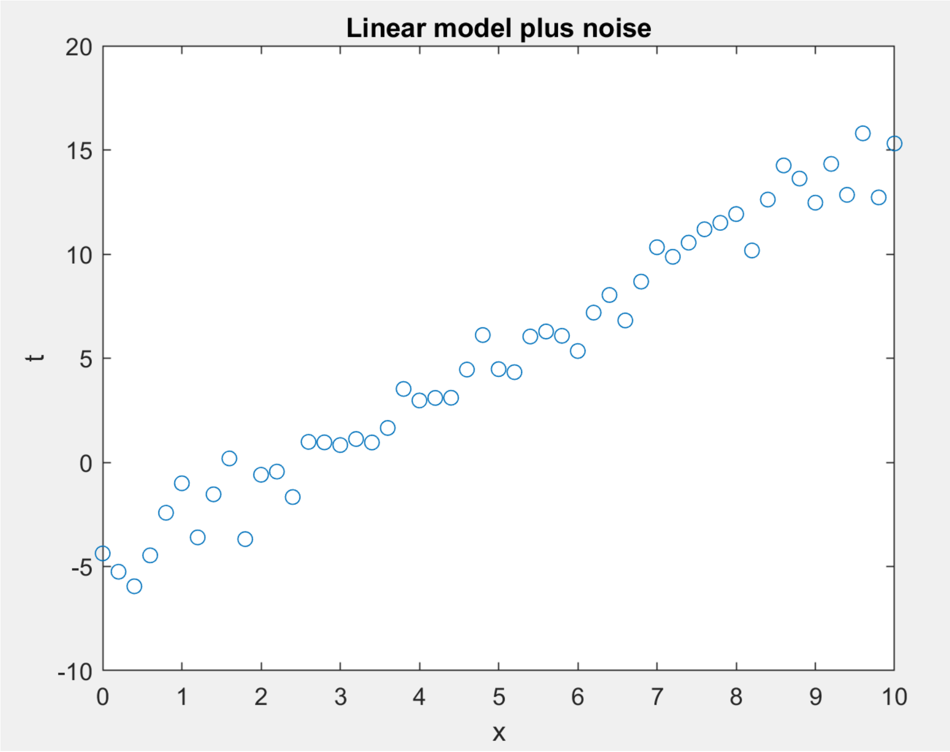
Starting from regression
we will train and test
the first neural networks



# Matlab: create and train a linear regressor

Create 51 points for the eq. $t = 2x - 5$ plus gaussian noise

```
%% creating a data with known distribution
% PROBLEM 01
x = [0:0.2:10];
noise = randn(size(x));
t = 2 * x - 5 + noise;


h1 = figure ;
plot(x,t, 'o');
title('PROBLEM 01 liner data plus noise');
xlabel('x');
ylabel('t');
```



# Create and train a linear regressor

**net1 = newlind(x,t)**

METHOD#1 (direct, see Lesson #15 from data matrix)

```
%% create the linear model and train it

if (1) % optimized solution based on the invertion of the matrix of data
    net1  = newlind(x,t);
else  % not optimized this is the general learning for non-linear networks

                                                                          dation partition

end
```

# Create and train a linear regressor

You can create the linear regressor using the initialization and learning for neural networks

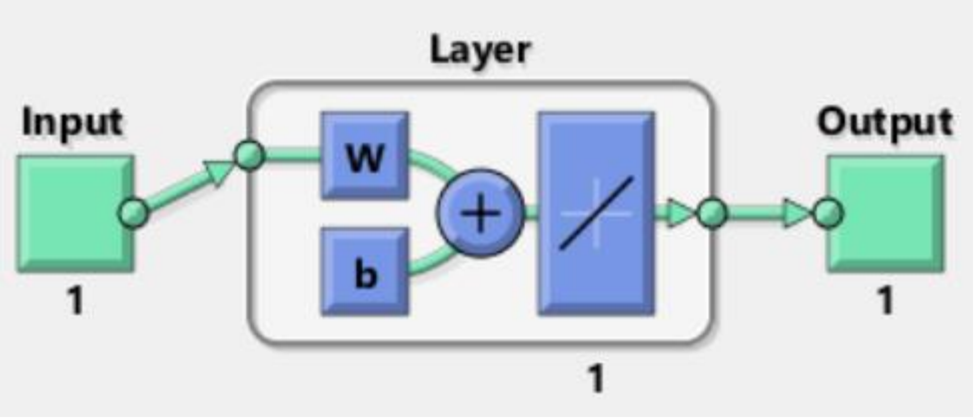
```
%% create the linear model and train it

if (1) % optimized solution based on the invertion of the matrix of data

else  % not optimized this is the general learning for non-linear networks
    % net1 = perceptron;
    net1 = linearlayer;
    net1 = configure(net1,[0],0);

    % A little help to tune the inizial points
    net1.IW{1,1} = 3.1;
    net1.b{1} = -3.1

    [net1 , tr] = train(net1,x,t);    % WARNING --> Train/Validation auto.created
    figure(h2); plotperform(tr)   % need more tuning..... :-(
end
```

**METHOD#2**

- inzialization as linear net
- 1 input 1 layer
- Weight inizialization
- Levemberg-Marquardt optimization
- (general, but not optimized)

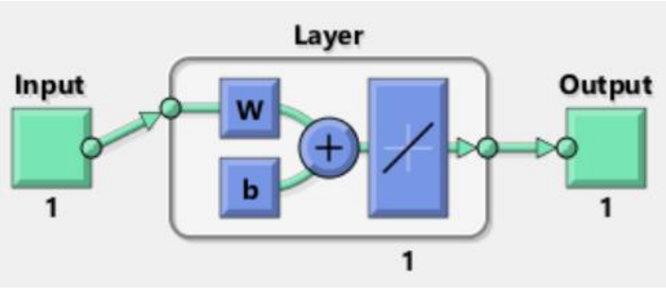
# The created linear regressor

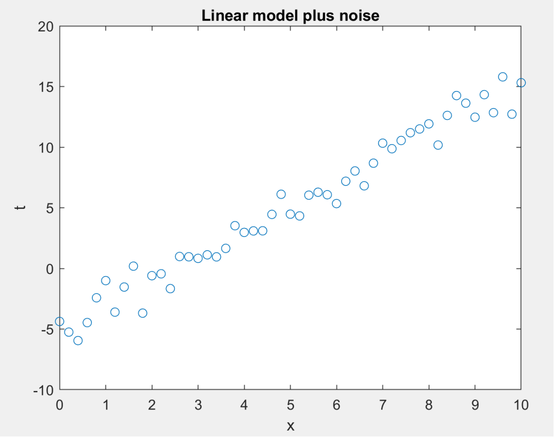
```
%% visualization of the trained linear model
view(net1)
```




Output = w input + b

# Access the parameters




```
% how to access the weight of the model
W = net1.IW{1,1} ;
b= net1.b{1} ;
fprintf('--------------------------------\n');
fprintf('trained linear model W = %f \n' , W);
fprintf('trained linear model b = %f \n' , b);
fprintf('--------------------------------\n');
```

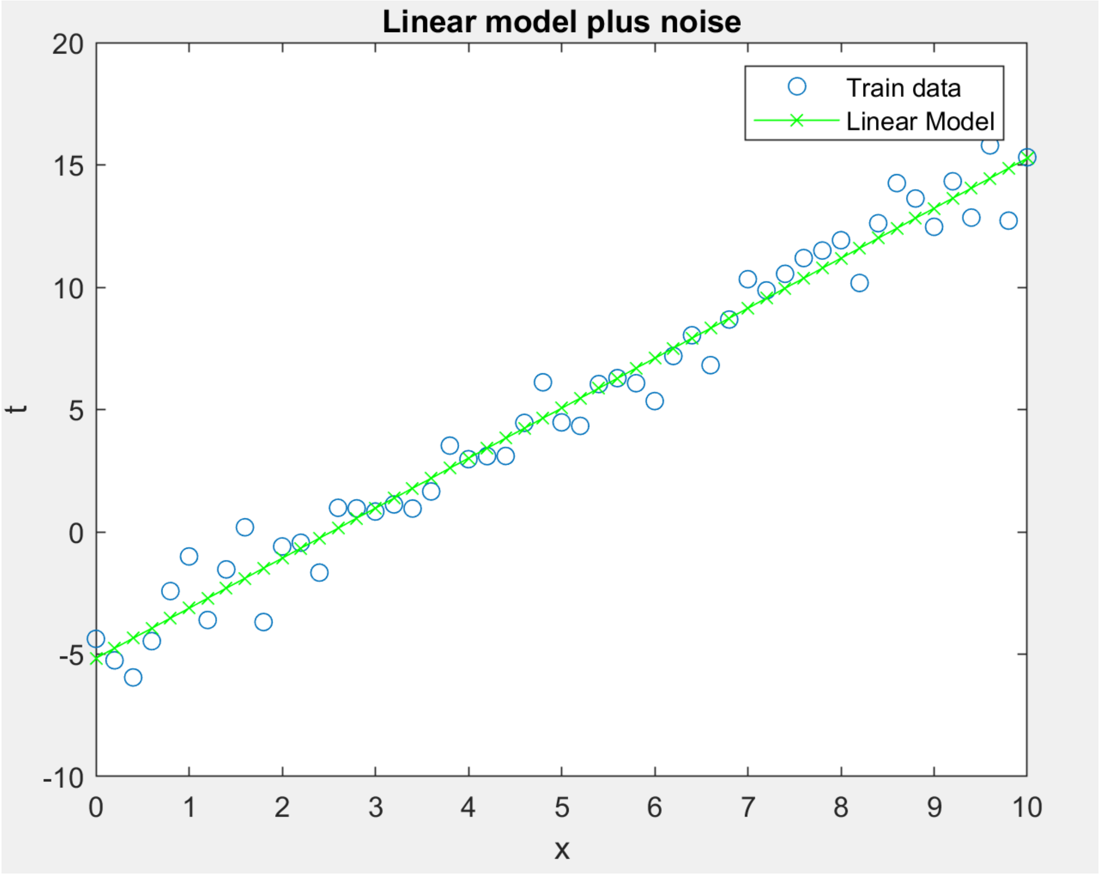
```
x = [0:0.2:10];
noise = randn(size(x));
t = 2 * x - 5 + noise;
```

```
--------------------------------
trained linear model W = 2.043733
trained linear model b = -5.165622
--------------------------------
```

# Processing inputs and plotting the results

```
h1 = figure ;
plot(x,t, 'o');
title('Linear model plus noise');
xlabel('x');
ylabel('t');
```

```
% how to check the performance
y1 = net1(x);  % simulate the inputs
perf = perform(net1,y1,t) % Error on **trainig**

% plot the output of the model
figure(h1)
hold on
plot(x, y1, 'xg-')
legend('Train data', 'Linear Model')
```




**net1 = newlind(x,t)**

**y1 = net1(x)**

# Accuracy assessment

```
%% how to check the performance
y1 = net1(x);   % simulate the inputs
perf = perform(net1,y1,t) % Error on **trainig**
net1.performFcn  % is telling you the error metrics (MSE for Lin.)
mse_check = sum((y1 - t).^2 )/ size(t,2);
mean_t = mean(t);
R2 = 1 - ( sum((t - y1).^2 )/sum((t - mean_t).^2) );
fprintf('------trained linear model------\n');
fprintf('Perf.         = %f \n' , perf );
fprintf('Type of index = %s \n' , net1.performFcn );
fprintf('MSE           = %f \n' , mse_check );
fprintf('R2            = %f \n' , R2 );
fprintf('--------------------------------\n');
```

$$MSE = \sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}$$

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$

**WARNING**
This is about TRAIN dataset
You need to re-do the measures
on a TEST dataset

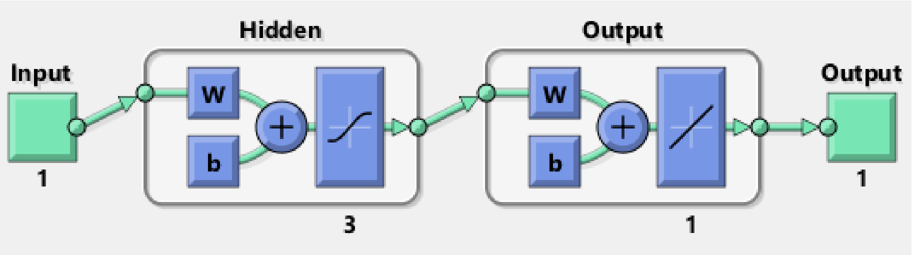
```
------trained linear model------
Perf.         = 0.836647
Type of index = mse
MSE           = 0.836647
R2            = 0.976989
--------------------------------
```

# Try a feedforward neural network!

**net2 = feedforwardnet (3)** Inizialization with 3 neurons

**net2 = train(net2,x,t);** training

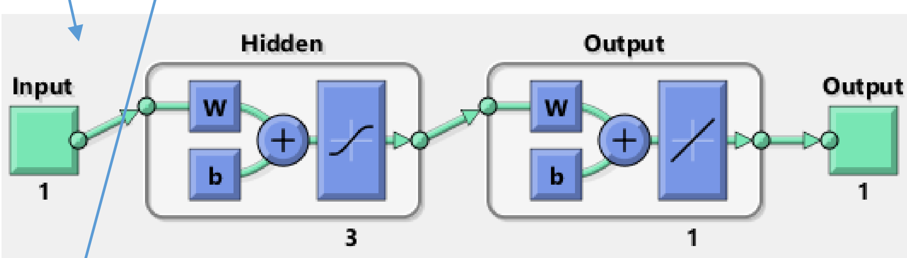
**y2 = net2(x)** running



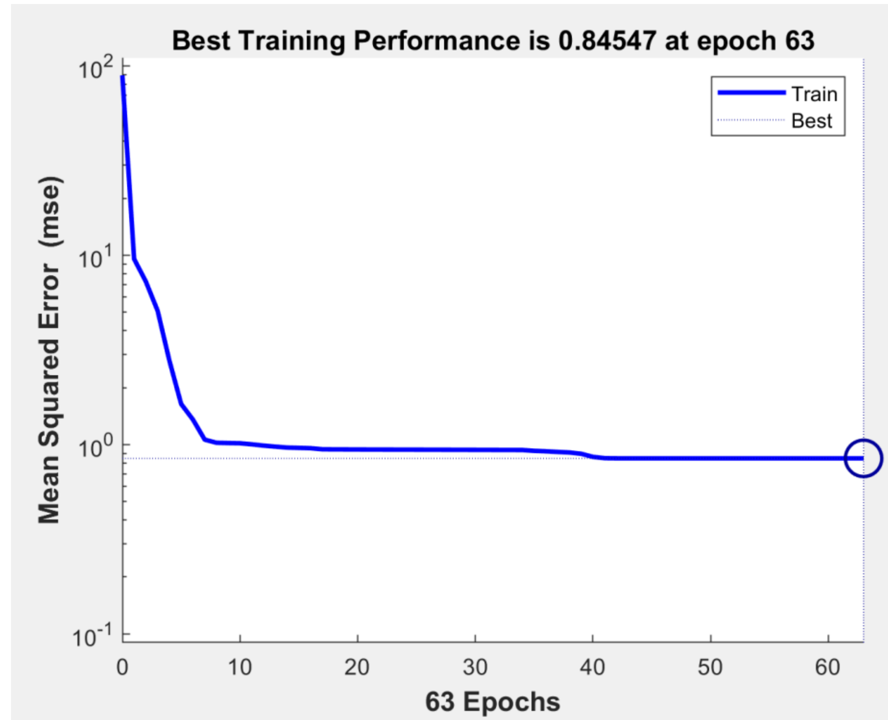
# Try a feedforward neural network!

```
%% % Let's face the OCCAM's RAZOR --> use a large network
net2 = feedforwardnet(3);

net2.divideParam.trainRatio = 1.0;
net2.divideParam.testRatio  = 0.0;
net2.divideParam.valRatio   = 0;

[net2  , tr] = train(net2,x,t);

h2 = figure;
plotperform(tr)

view(net2)
y2 = net2(x);
perf = perform(net2,y2,t) % Error on **trainig**
```

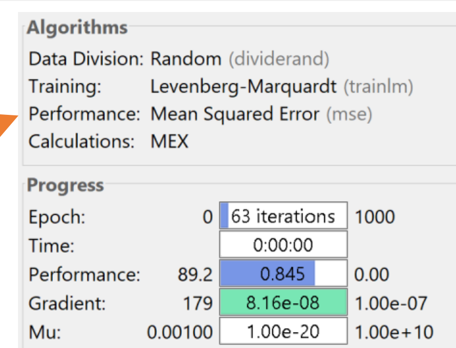
Force to use all data in the TRAIN set




```
perf =

    0.8455
```




MSE

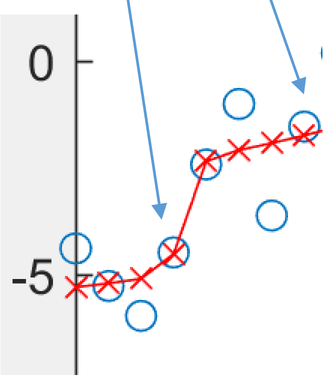


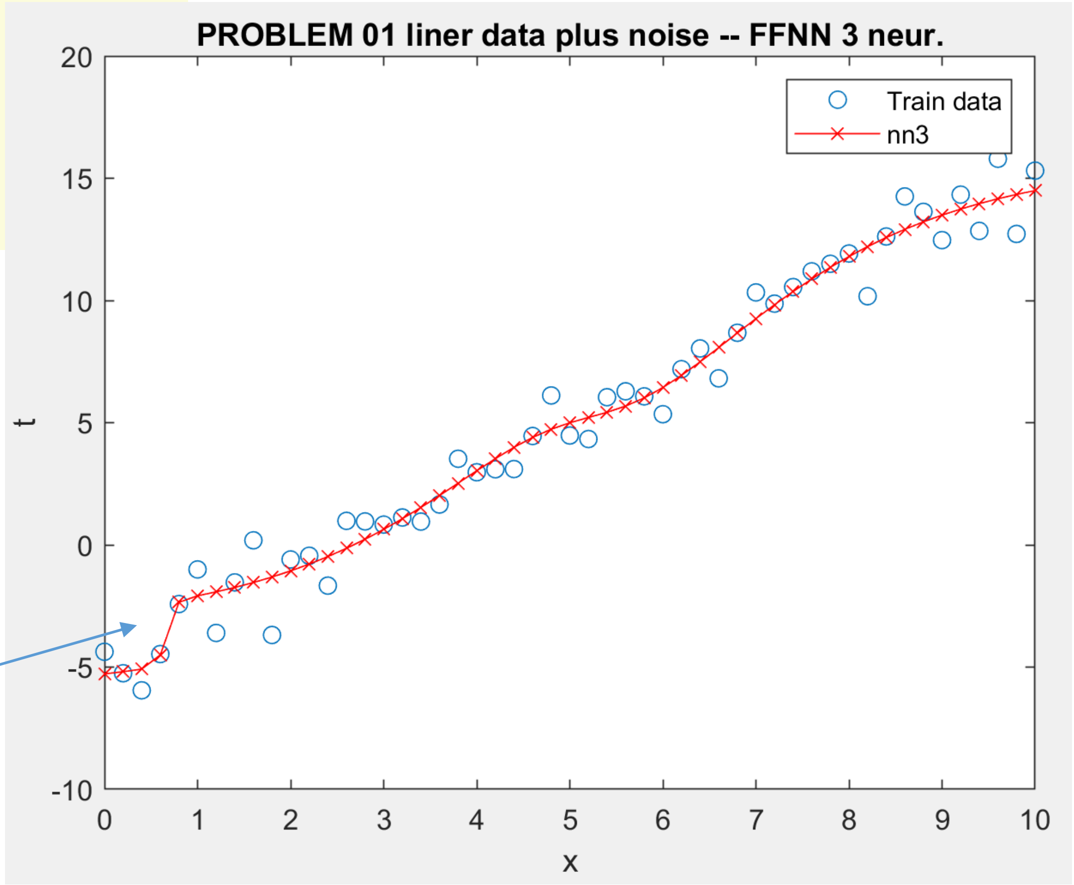
# Try a feedforward neural network!

```
h3 = figure
plot(x,t, 'o');
title('PROBLEM 01 liner data plus noise -- FFNN 3 neur.');
xlabel('x');
ylabel('t');

hold on
plot(x,y2, 'xr-');
xlabel('x');
ylabel('t');
legend('Train data', 'nn3')
```

Noise fitting ?

correct



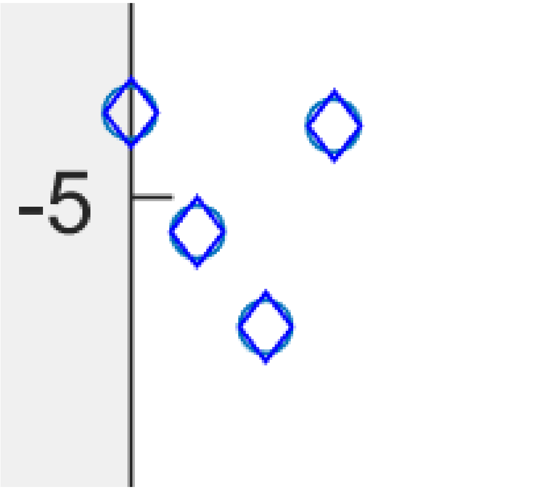



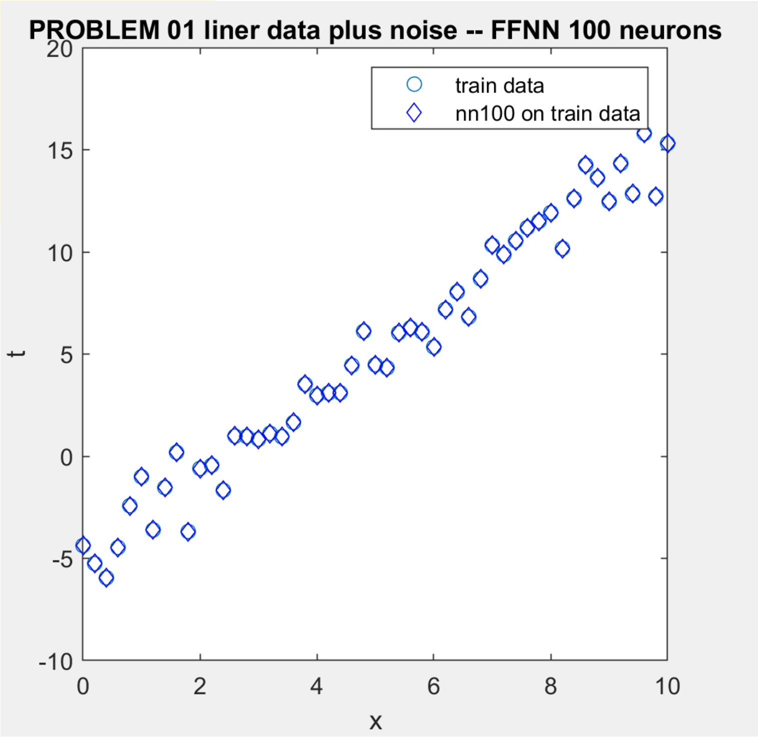
# Wrong design:
# 100 neurons with 51 data points!

```
%% real overfitting
% Let's face the OCCAM's RAZOR --> use very large network
net3 = feedforwardnet(100);
net3.divideParam.trainRatio = 1.0;
net3.divideParam.testRatio  = 0.0;
net3.divideParam.valRatio   = 0;

[net3  , tr] = train(net3,x,t);
figure(h2); plotperform(tr)
view(net3)
y3 = net3(x);
```

# Wrong design:
# 100 neurons with 51 data points!

```
h4 = figure
subplot(1,2,1)
plot(x,t, 'o');
title('PROBLEM 01 liner data plus noise -- FFNN 100 neurons');
xlabel('x');
ylabel('t');
hold on
plot(x,y3, 'db');
legend('train data', 'nn100 on train data')
```



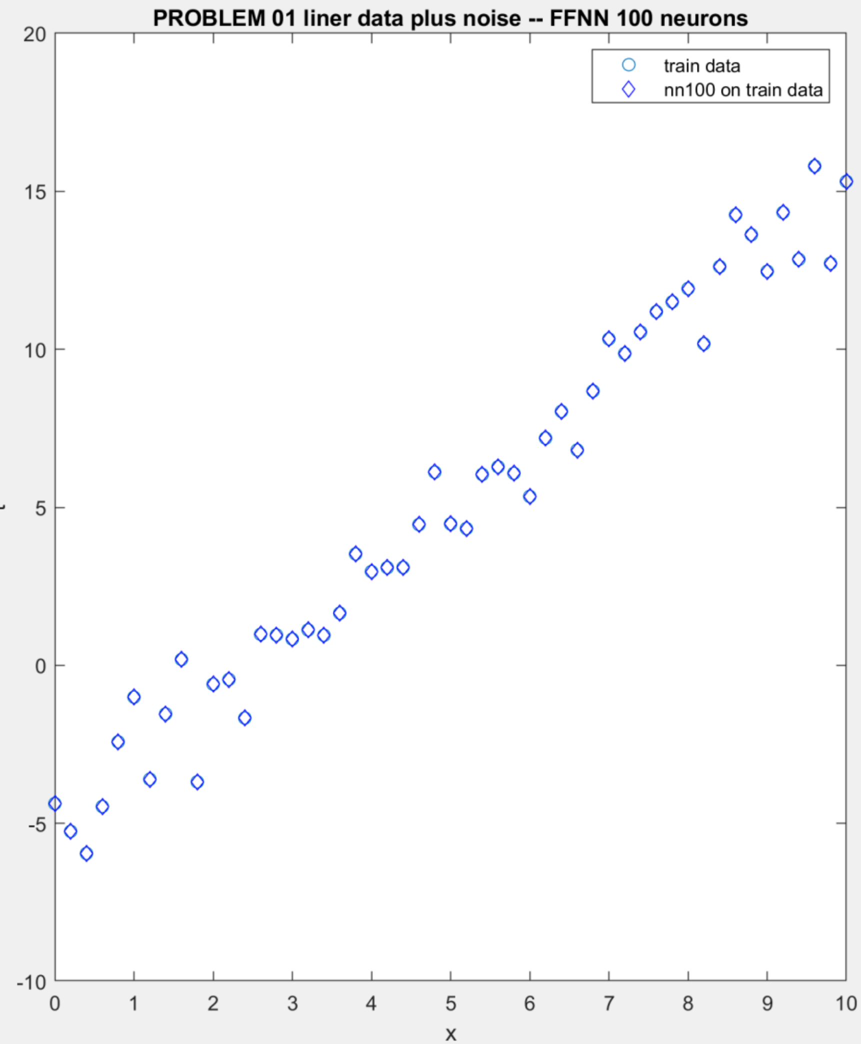



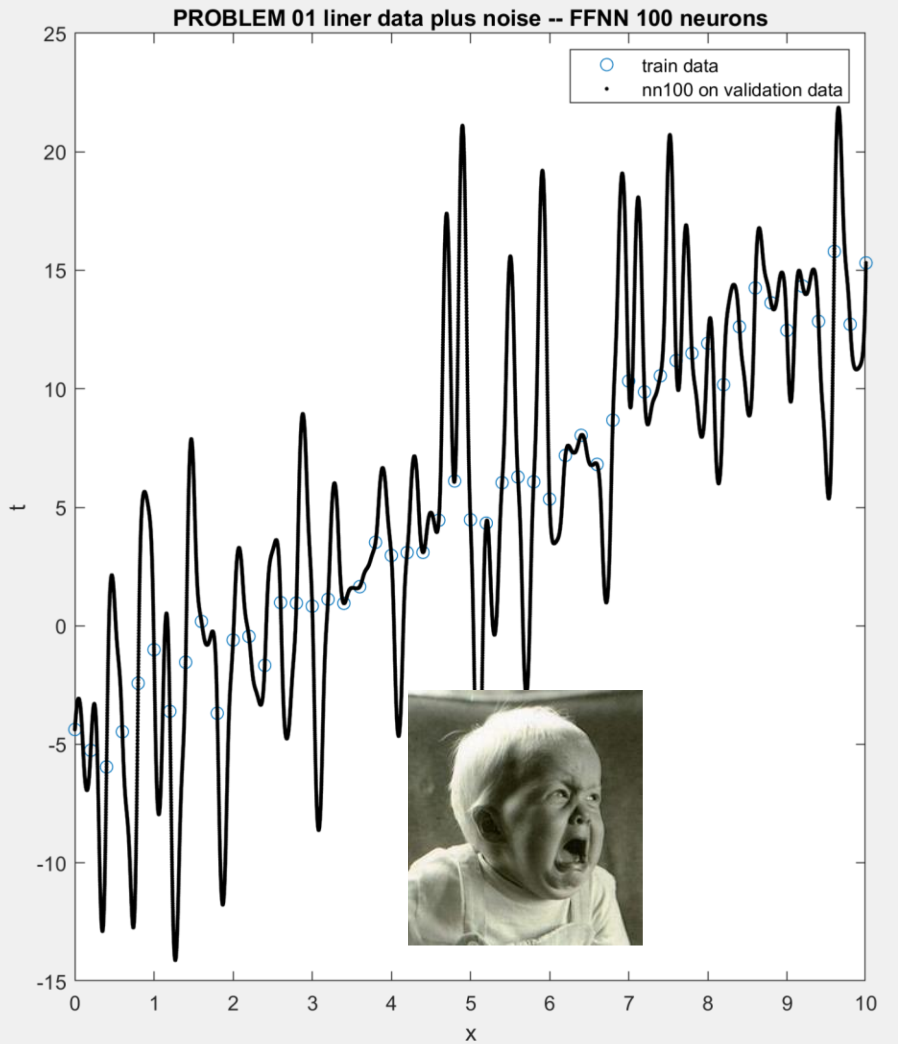
# Let’s built a test set….

Let’s inject in the net3 (100 neurons) more x points…

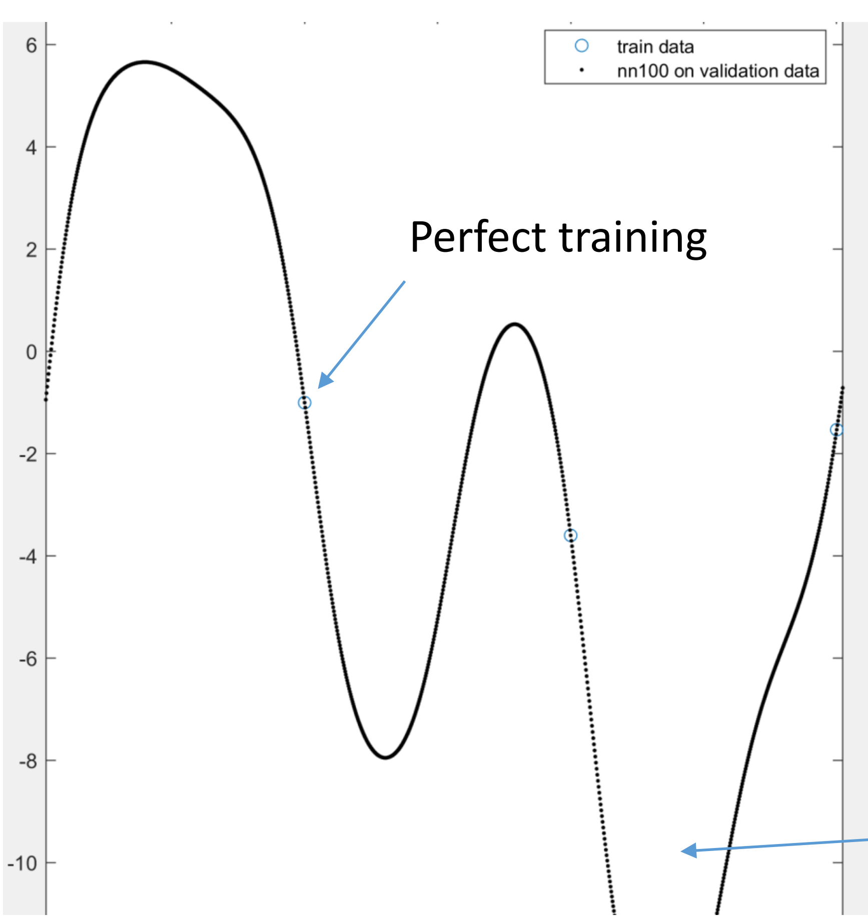
```
%% test point
x_bis = [0:0.0005:10];
y3_bis = net3(x_bis);

perf = perform(net3,y3,t) % Error on **trainig**

subplot(1,2,2)
plot(x,t, 'o');
title('PROBLEM 01 liner data plus noise -- FFNN 100 neurons');
xlabel('x');
ylabel('t');
hold on
plot(x_bis,y3_bis, '.k');
legend('train data', 'nn100 on validation data' )
```

# ...and we overfitted  (a lot!)

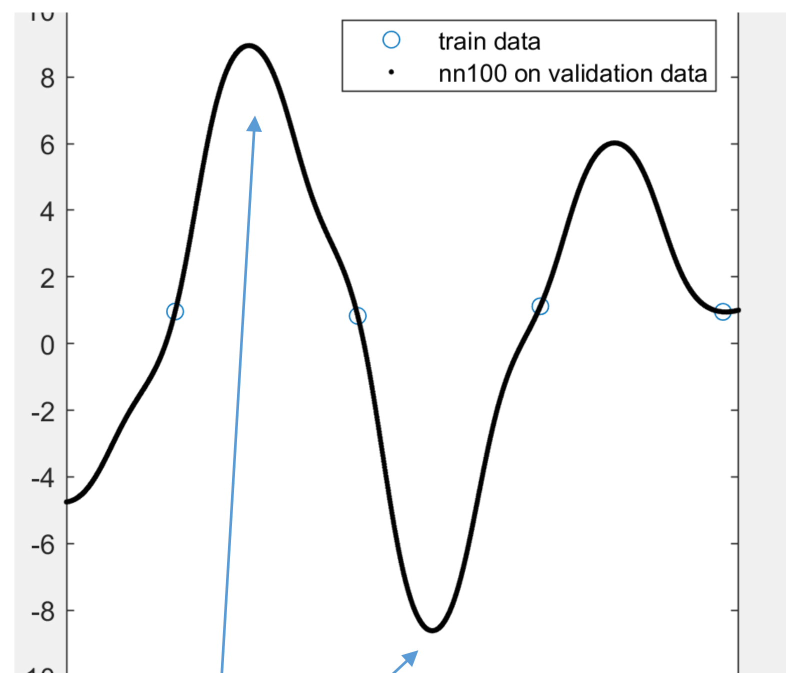


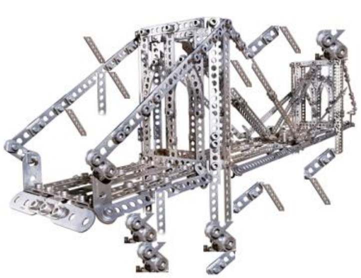
# Remember next time you will initialize a model with #DoF > #samples…




Perfect training




Very bad generalization



# Python file in Colab

- *1) Follow next slides*
- *2) and then lunch your script.*

laboratory_COLAB_linear_regressors.ipynb

Let’s review how to create a regressor, check accuracy and doing some plots

# Import the needed libraries:
# - plotting
# - numerical processing

```
import matplotlib.pyplot as plt
import numpy as np
```

NumPy

It adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

# Data preparation

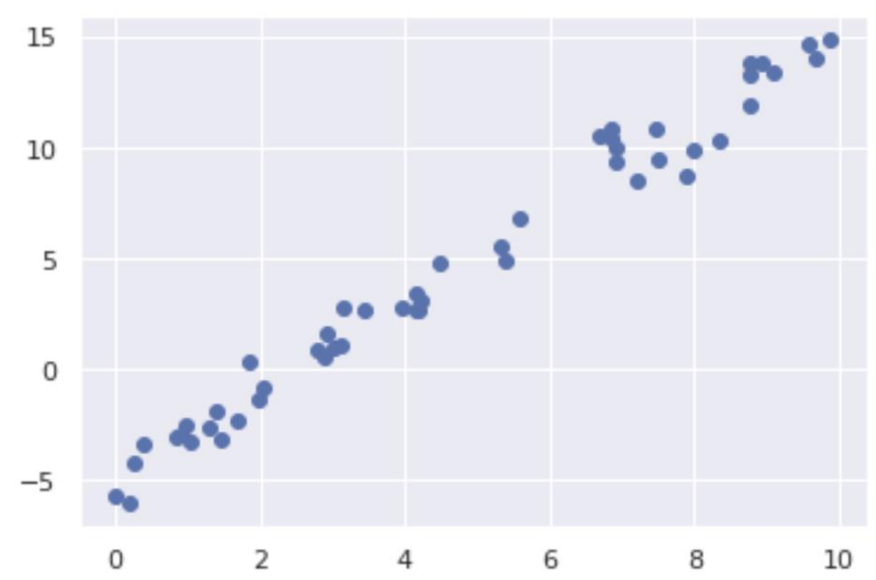
## 1D Linear Regression

We want to create a model wich is a a straight-line fit of the form

$$y = ax + b$$

Consider the following data, which is scattered about a line with a slope of 2 and an intercept of -5:

```
[35] rng = np.random.RandomState(1)
     num_points = 50
     x = 10 * rng.rand(num_points )
     t = 2 * x - 5 + rng.randn(num_points )
     plt.scatter(x, t);
```



# Creation of the linerar regressor

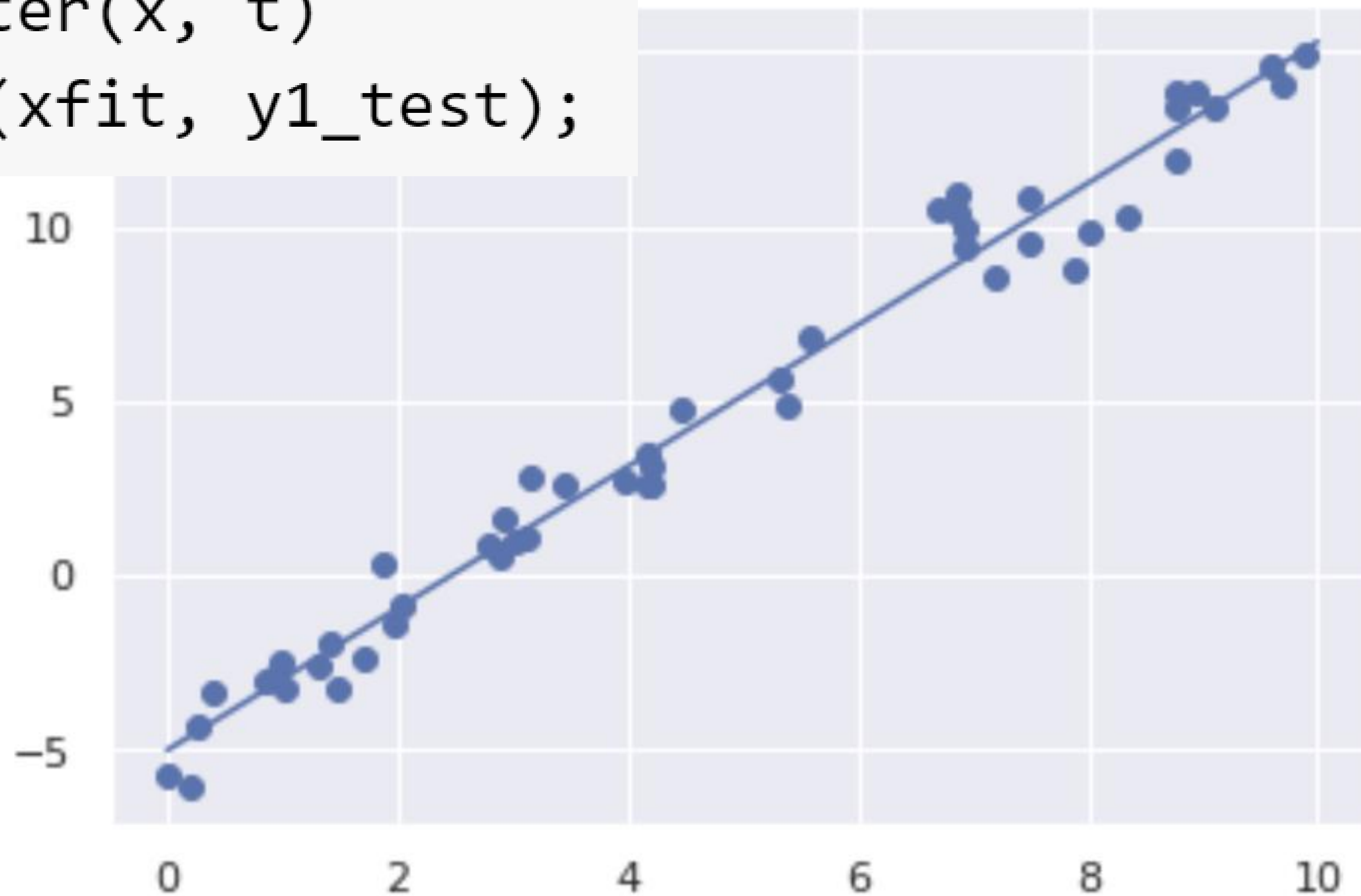
We can use Scikit-Learn's LinearRegression estimator
to fit this data and construct the best-fit line:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True)

model.fit(x[:, np.newaxis], t) # newaxis adds a dimension (needed from the function)

y1 = model.predict(x[:, np.newaxis])  # let's create the predicted values

xfit = np.linspace(0, 10, 1000)  # let's create an array to test the regressor [0 ... 10] of 1000 points
y1_test = model.predict(xfit[:, np.newaxis])  # let's create the predicted values
```

```
model = LinearRegression(fit_intercept=True)
model.fit(x[:, np.newaxis], t)
y1 = model.predict(x[:, np.newaxis]
```

# Plotting

```
plt.scatter(x, t)
plt.plot(xfit, y1_test);
```



# Get the parameters of the model

```
[37] print("Model slope:     ", model.coef_[0])
     print("Model intercept:", model.intercept_)
```

```
Model slope:      2.0272088103606953
Model intercept: -4.998577085553204
```

# Accuracy assessement

```
[38] from sklearn.metrics import mean_squared_error
     from math import sqrt

     # Mean squared error
     mse = mean_squared_error(y1, t)
     rmse = sqrt(mse)

     #R^2
     from sklearn.metrics import r2_score
     R2 = r2_score(y1, t)

     print("MSE:     ", mse  )
     print("RMSE:    ", rmse )
     print("R2:      ", R2   )
```

```
MSE:      0.8183388570266171
RMSE:     0.9046208360559783
R2:       0.9786330659856474
```

** WARNING **
You have to create a TEST daset to get the real accuracy in generalization. This is processed on the TRAIN dataset.

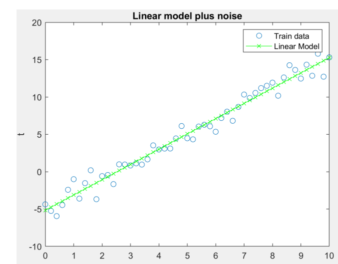
You will find different numbers w.r.t. Matlab due to the different data initialization

# Main points

- *New model? What to know? What to do?*
- *Creation and use (with code) of the first complete machine learning model*
  - *Linear regressors*
  - *First neural network*
  - *Accuracy assessment*

**net1 = newlind(x,t)**

**y1 = net1(x)**




**net2 = feedforwardnet (3)**
**net2   = train(net2,x,t);**
**y2 = net2(x)**