# Introduction to machine learning and regression models

Course 34366 Intelligent Systems
Lecture 2 – September 6, 2019

Darko Zibar
dazi@fotonik.dtu.dk

# Today's agenda

- Part I – Linear models for classification (~60 min)
  - What is machine learning?
  - Applications of machine learning
  - Types of machine learning
  - Learning the linear model
  - Learning the nonlinear model
  - What is a neural network
  - The simplest neural network (Rosenblatt perceptron)
  - Learning logical gates (AND, OR and XOR)
  - Break (~15 min)
- Part II – Problem solving session 1 (~120 min)
- Part III – Solution to exercises (~40 min)

# Material

1. Simon Haykin, Neural Networks and Learning Machines
    - Introduction 1,
    - Chapter 1

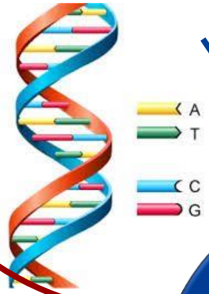
2. Christopher M. Bishop, Pattern Recognition and Machine Learning, Springer 2006
    - Introduction (pp. 1-12)
    - Chapter 3 (pp. 137-144)

http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf

https://www.academia.edu/34757446/Neural_Networks_and_Learning_Machines_3rd_Edition

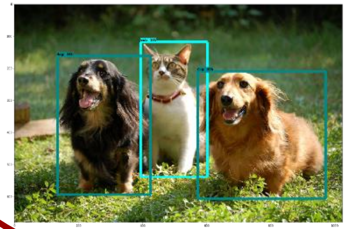
# Machine learning applications

DNA sequence classification




self-driving car

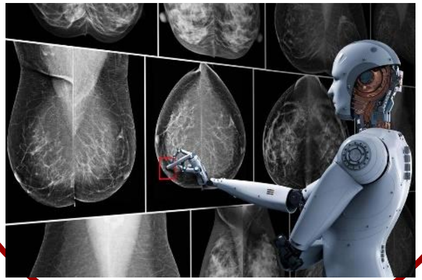
pattern and image recognition



cyber fraud detection
(anomaly detection)

recommendation systems


medical diagnosis

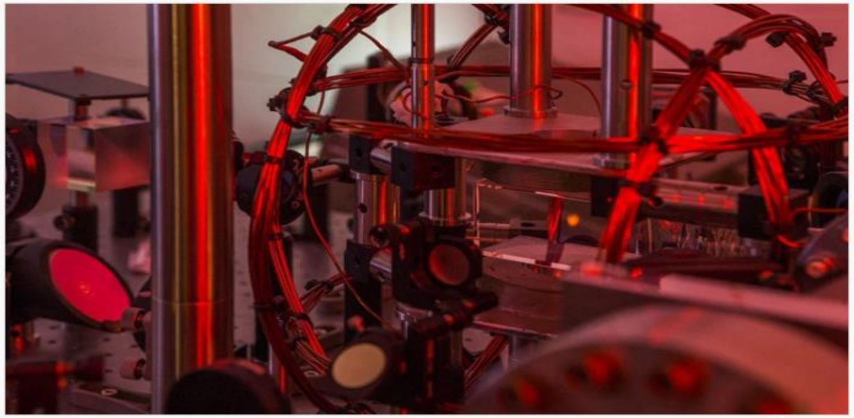


spam filtering



## AI learns and recreates Nobel-winning experiment

Posted May 16, 2016 by Devin Coldewey



Australian physicists, perhaps searching for a way to shorten the work week, have created an AI that can run and even improve a complex physics experiment with little oversight. The research could eventually allow human scientists to focus on high-level problems and research design, leaving the nuts and bolts to a robotic lab assistant.

**nature physics**

Thesis | Published: 02 December 2019

## The power of machine learning

Mark Buchanan

*Nature Physics* **15**, 1208(2019) | Cite this article

**nature photonics**

News & Views | Published: 30 November 2017

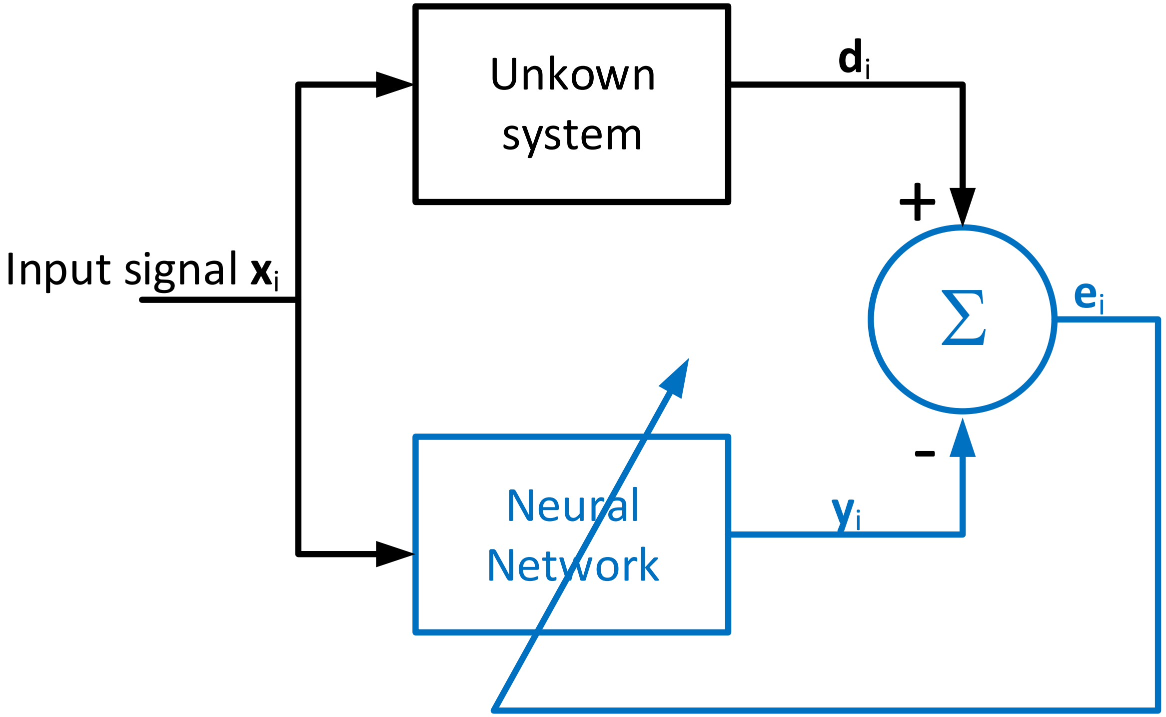
VIEW FROM... ECOC 2017

## Machine learning under the spotlight

Darko Zibar, Henk Wymeersch & Ilya Lyubomirsky

*Nature Photonics* **11**, 749–751(2017) | Cite this article

# System identification



# What is machine learning according to definitions?

"Field of study that gives computers the ability to learn without being explicitly programmed" (A. Samuel, 1959)

**Learn from data**

"A computer program is said to learn from experience **E** with respect to some task **T** and some performance measure **P**, if its performance on **T**, as measured by **P**, improves with experience **E**." (T. Mitchell, 1998)

**Learning from experience improves task's performance**

A.L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," in *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210-229, July 1959.

T.M. Mitchell, "Machine Learning," McGraw-Hill International Editions Computer Science Series) 1st Edition, 1998

# Where do we use ML?

Any ideas?

# When should we use ML?

Take 10min to discuss it with your neighbour.

**Please remember to respect social-distancing!**

When it is difficult or infeasible to develop a conventional algorithm for effectively performing the task

# What is machine learning today?

Keras
A deep learning library



TensorFlow

https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463

# Let’s go back to the beginning…

“A computer program is said to <u>learn</u> from <u>experience</u> **E** with respect to some <u>task</u> **T** and some performance measure **P**, if its performance on **T**, as measured by **P**, improves with experience **E**.” (T. Mitchell, 1998)

Example:

You want to apply machine learning to improve your spam filter

- What is T?
- What is E?
- What is P?

T.M. Mitchell, “Machine Learning,” McGraw-Hill International Editions Computer Science Series) 1st Edition, 1998

# An example of ML problem

Handwritten Digit Recognition

# Task, experience, performance…

Task: correctly recognize the hand-written digits

Experience: manually recognized digits

Performance: the machine correctly interprets the digits

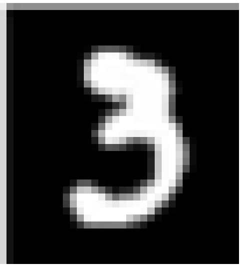
Training

Training set =

+

Associated hand-labelling

28 pixels



28 pixels

Input vectors ($\boldsymbol{x}$)

$\boldsymbol{x} = \{x_1, x_2, \ldots, x_N\}$

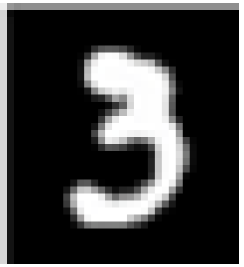
Target vector ($\boldsymbol{t}$)

$\boldsymbol{t} = \{t_1, t_2, \ldots, t_N\}$

"3"

# Training a model

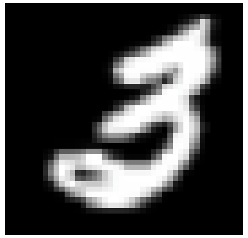
Tuning the adaptive model



$\boldsymbol{x} = \{x_1, x_2, \ldots, x_N\}$ → Adaptive model → $\boldsymbol{t} = \{t_1, t_2, \ldots, t_N\}$ “3”

Training or learning phase

# Testing a model



$\boldsymbol{x} = \{x_1, x_2, \dots, x_N\}$

Fixed parameter

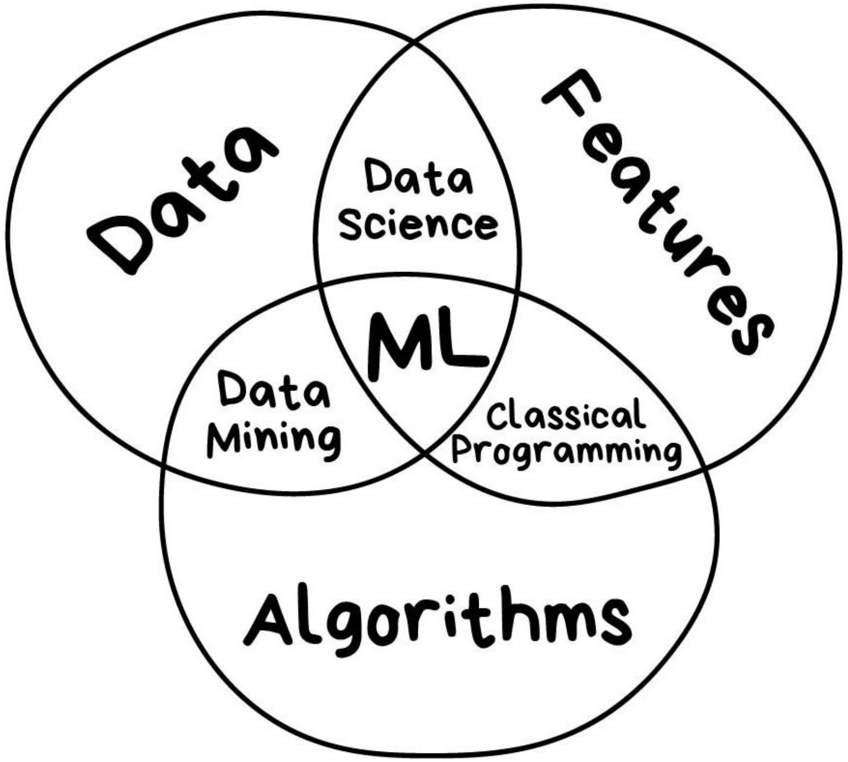
**Final model**
$y(x)$

$\hat{\boldsymbol{t}}$ ***Prediction***

Testing phase: how well does the model <u>generalize</u>?

<u>Generalization:</u> the ability to correctly categorize a new example

# The three key components of ML




- **Data → Your experience used to train your model**
- **Feature → The important parameters/variables in your data. These can be known in advance or may need to be extracted.**
- **Algorithms → The specific method you apply to solve a given problem.**

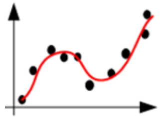
# Machine learning algorithms

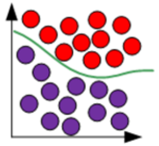
**Supervised learning**

Given data x and associated value y, learn $y = f(x)$ or $p(y|x)$

Regression

Classification





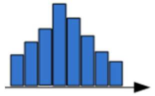
Predict outcomes

**Unsupervised learning**

Given data x learn useful properties: $p(x)$

Clustering

Density estimation



Describe data

**Reinforcement learning**

Feedback based optimization with interaction reward

reward

Learning system

Environment

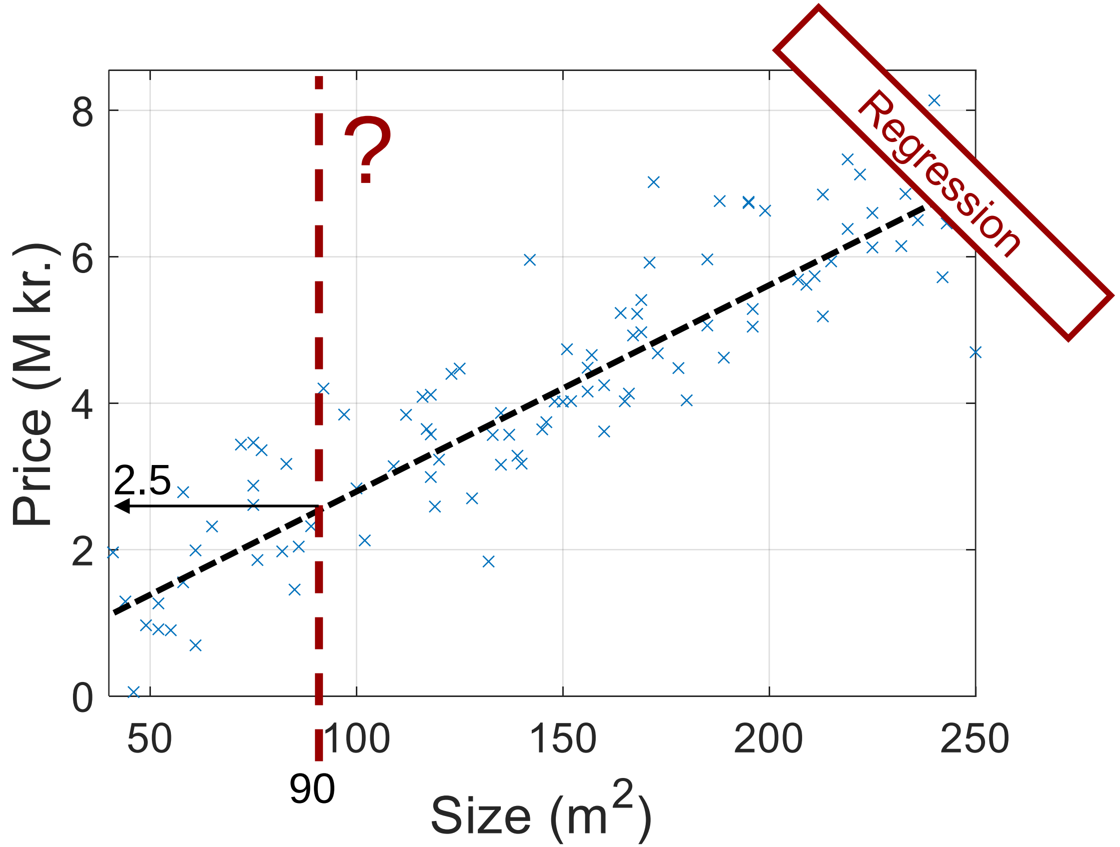
action

Improve

# Supervised learning

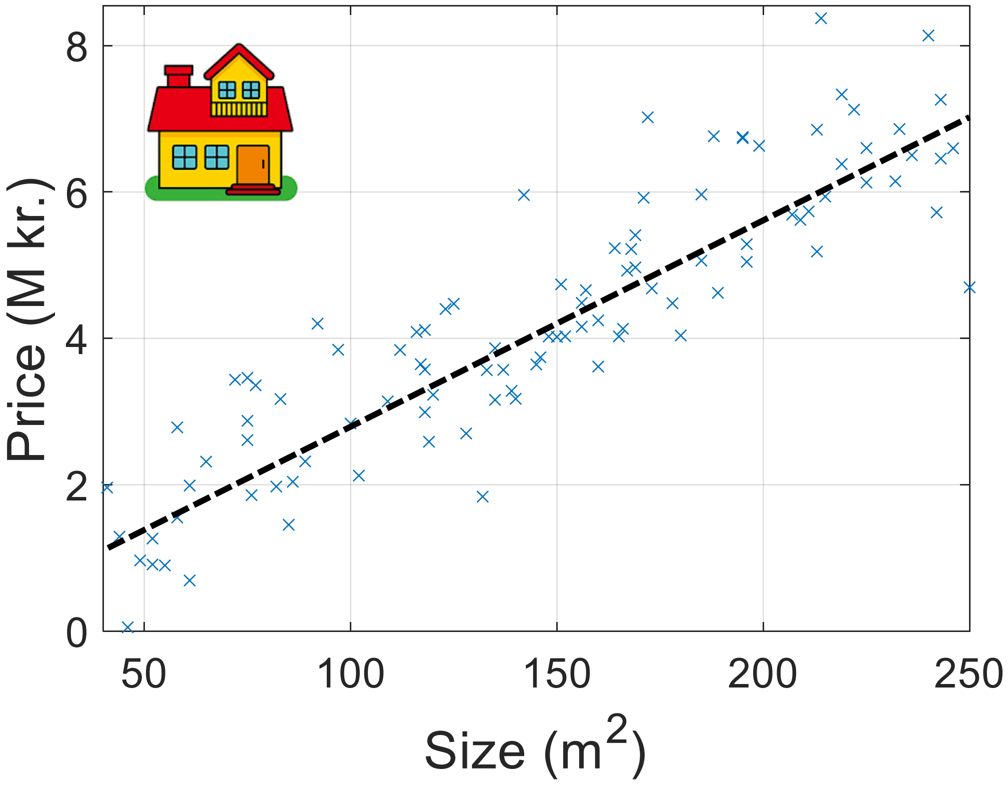
In supervised learning, the “right answers” (e.g. “labels” for training) are known.

Example: house pricing / square meter

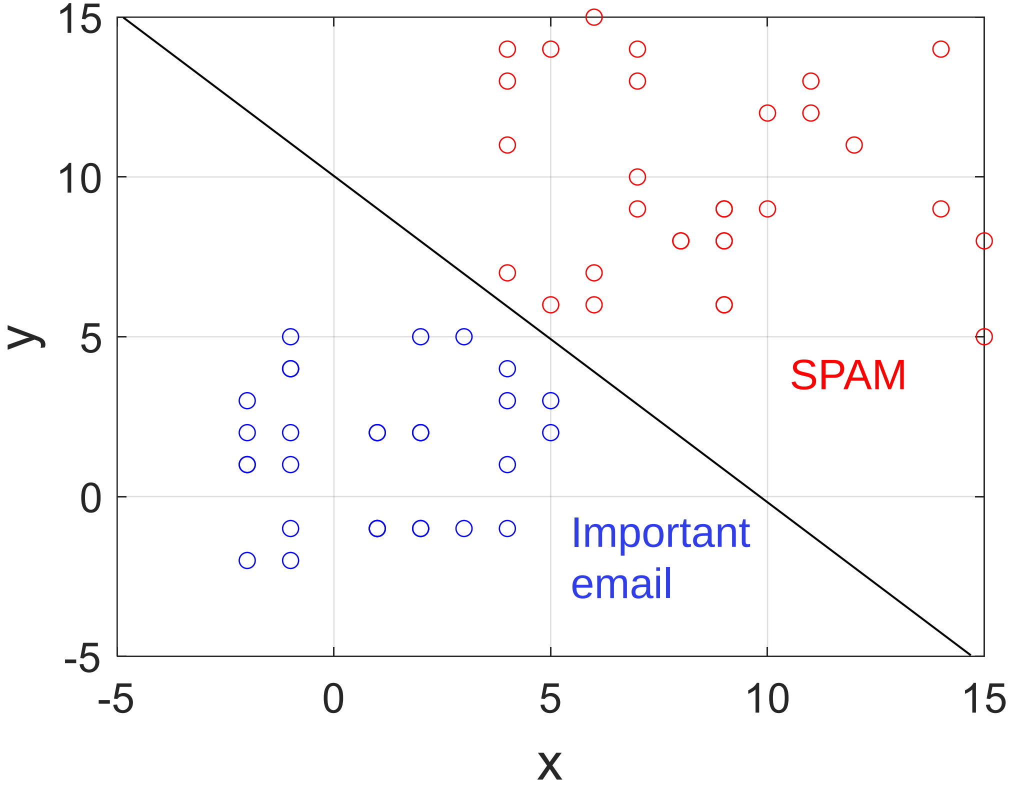


# Regression vs. classification

House pricing prediction




Spam filter

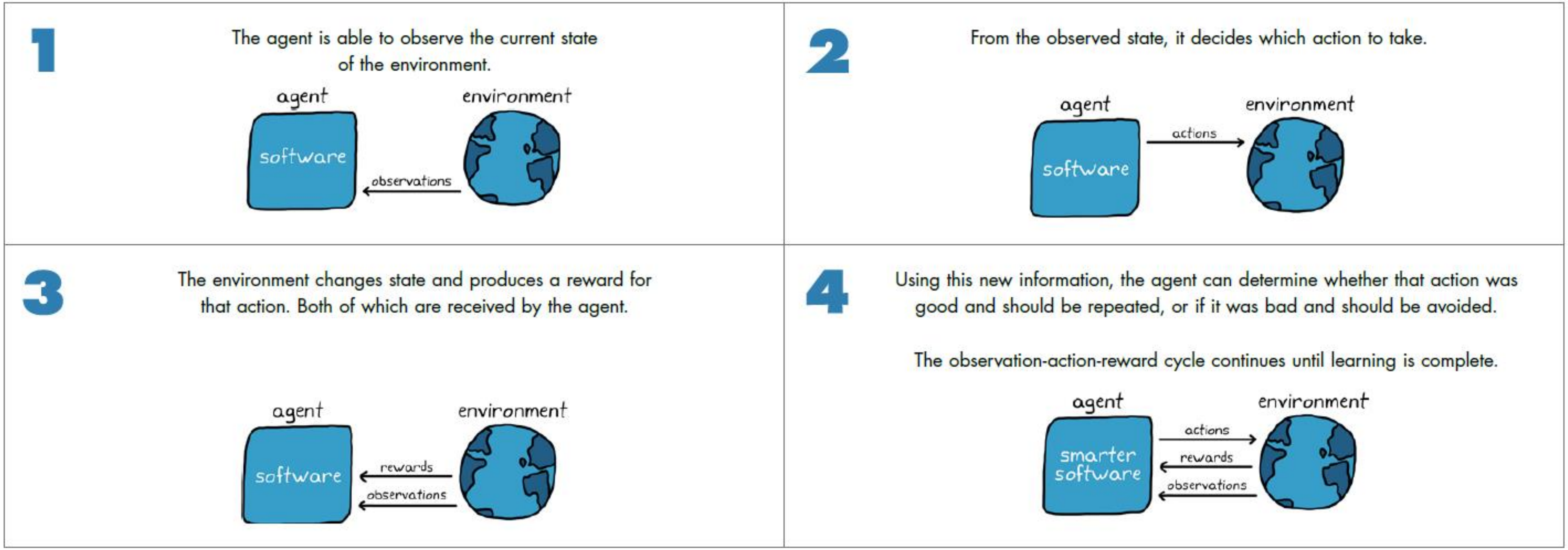


# Regression vs. classification – checking…

Regression problems predict **continuous** valued outputs

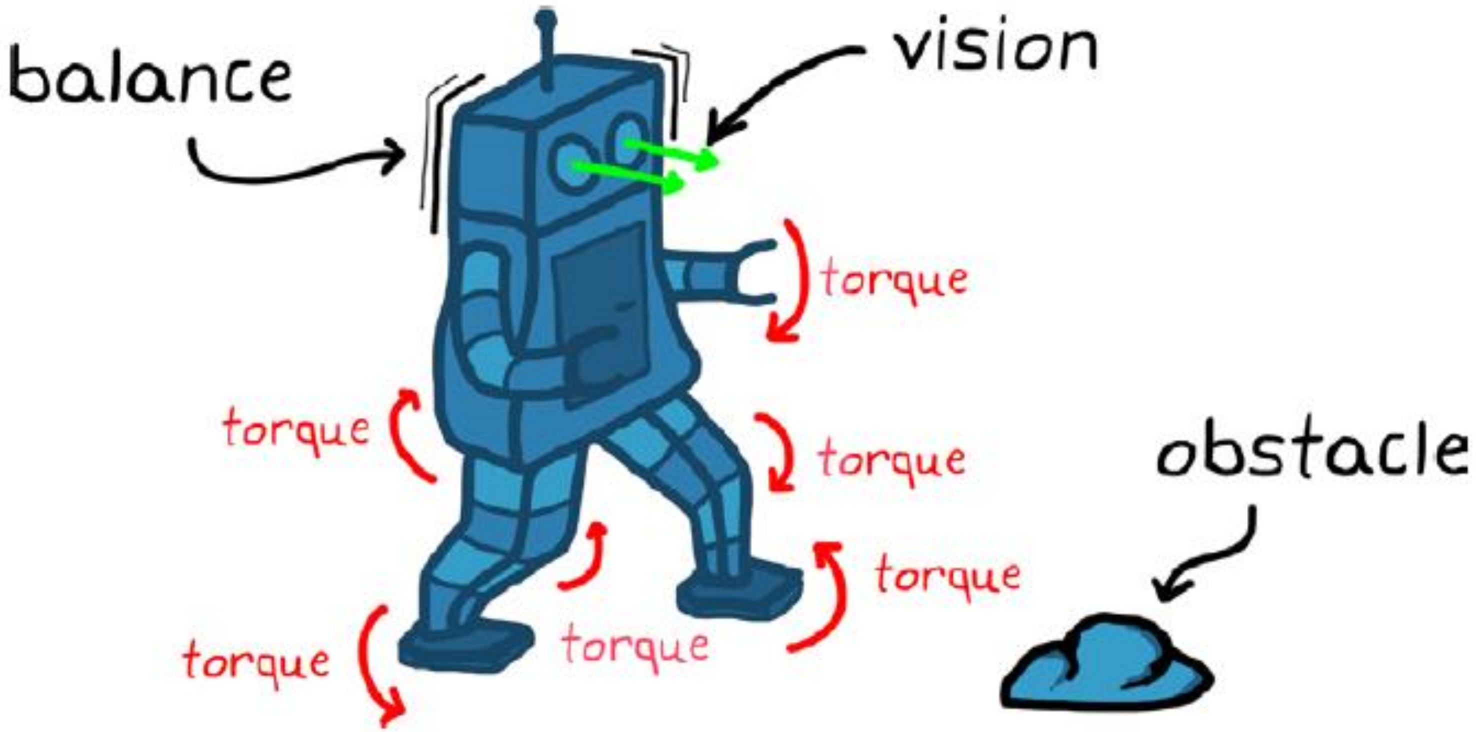
Classification problems predict **discrete** valued outputs

# Reinforcement learning (RL)



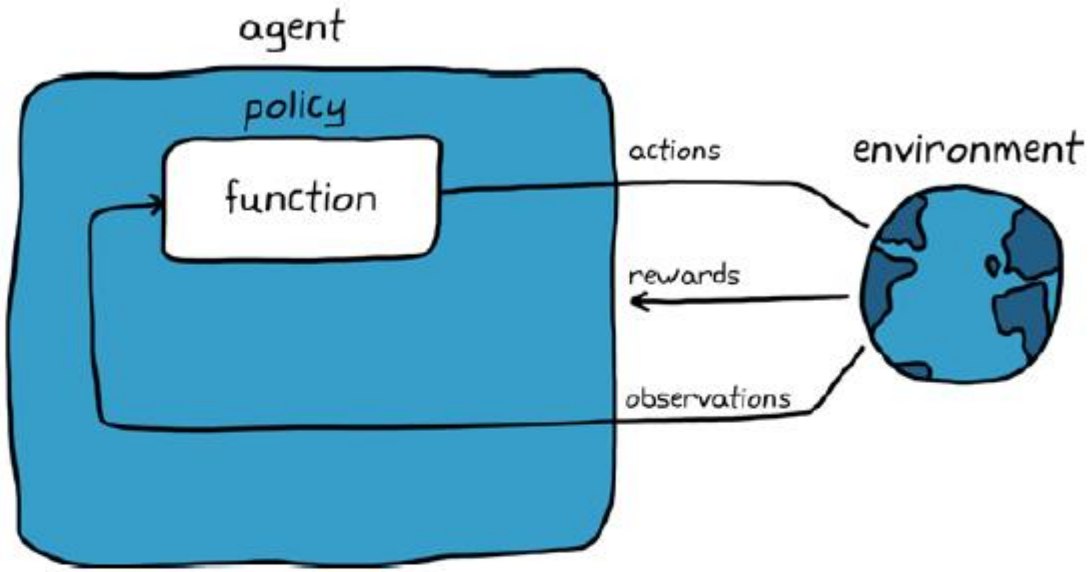

MATLAB Reinforcement learning e-book

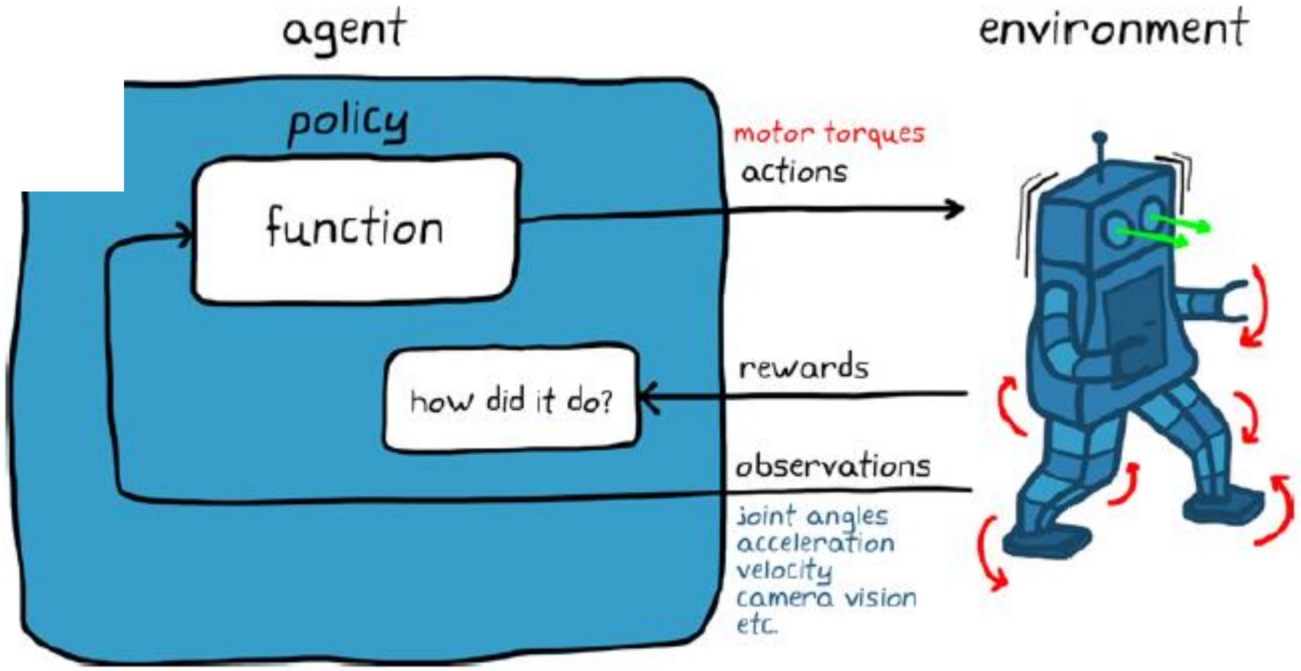
# RL for controlling robot motion




MATLAB Reinforcement learning e-book

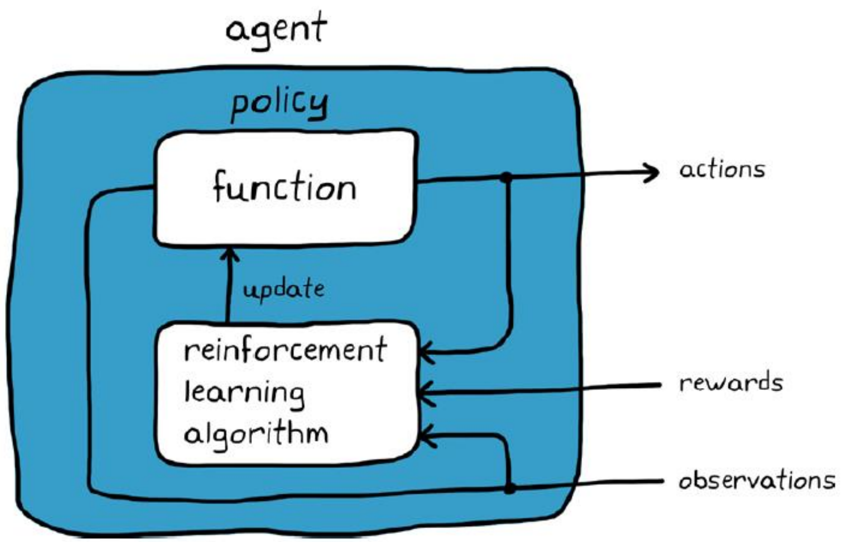
# RL for controlling robot motion

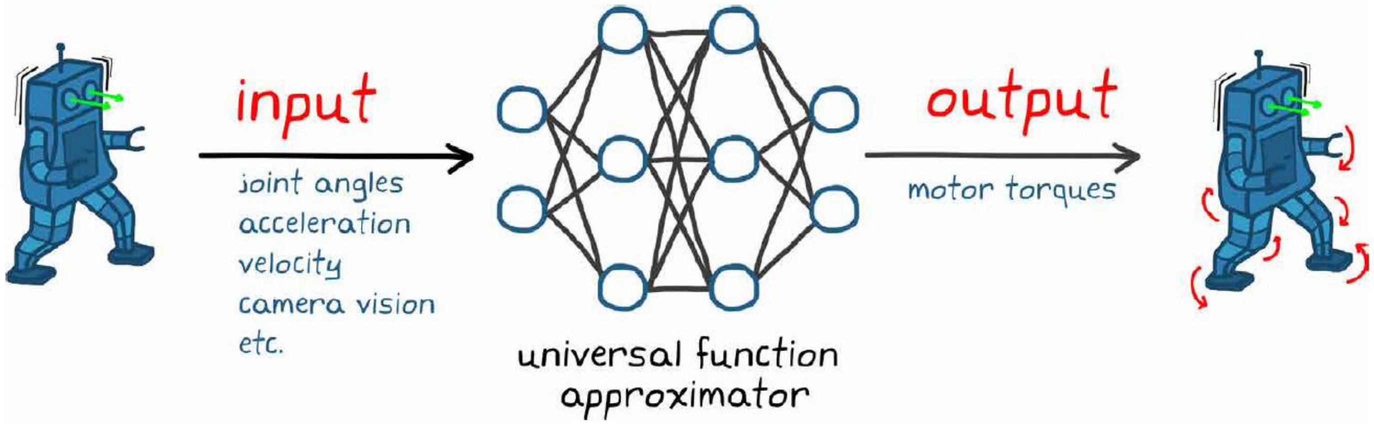



MATLAB Reinforcement learning e-book

# Actions represented by neural-network



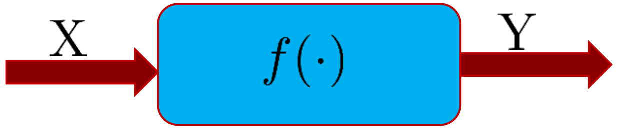




MATLAB Reinforcement learning e-book

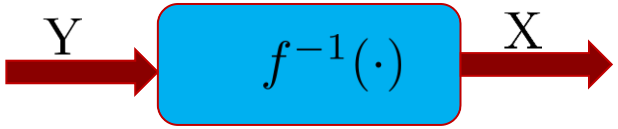
# Time for a break….

# Where does machine learning excel?

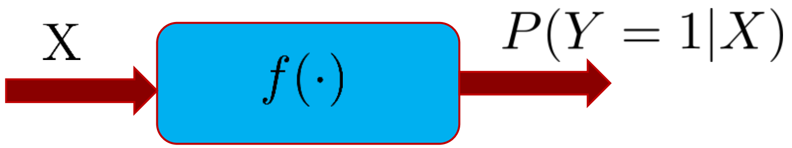
- Learning complex **direct** mappings:




- Learning complex **inverse** mappings:

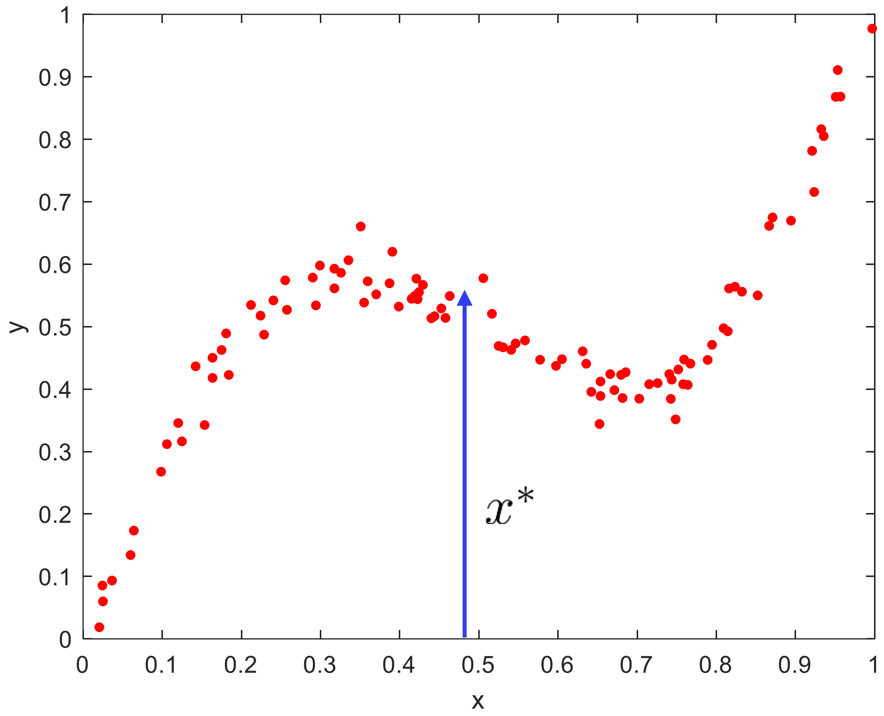



- Learning **decision rules** for complex mappings:




Use neural networks to learn $f(\cdot)$ and $f^{-1}(\cdot)$

# Learning complex functions




1. Given a training data-set (input and output values): $\mathcal{D} = \{x(k), y(k)\}_{k=1}^{K}$
2. For new input $x^*(k) \notin \mathcal{D}$ compute $y^*(k)$
3. Analytical expression *unknown,* must be learned from: $\mathcal{D} = \{x(k), y(k)\}_{k=1}^{K}$

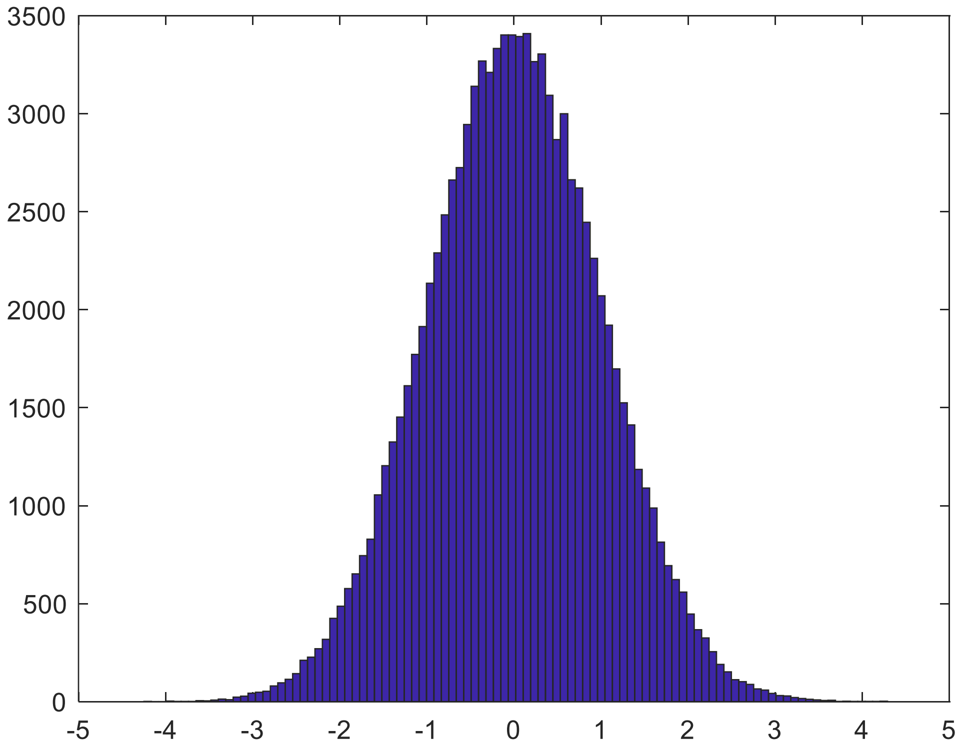
Use *machine learning* to build a model that represents the data set

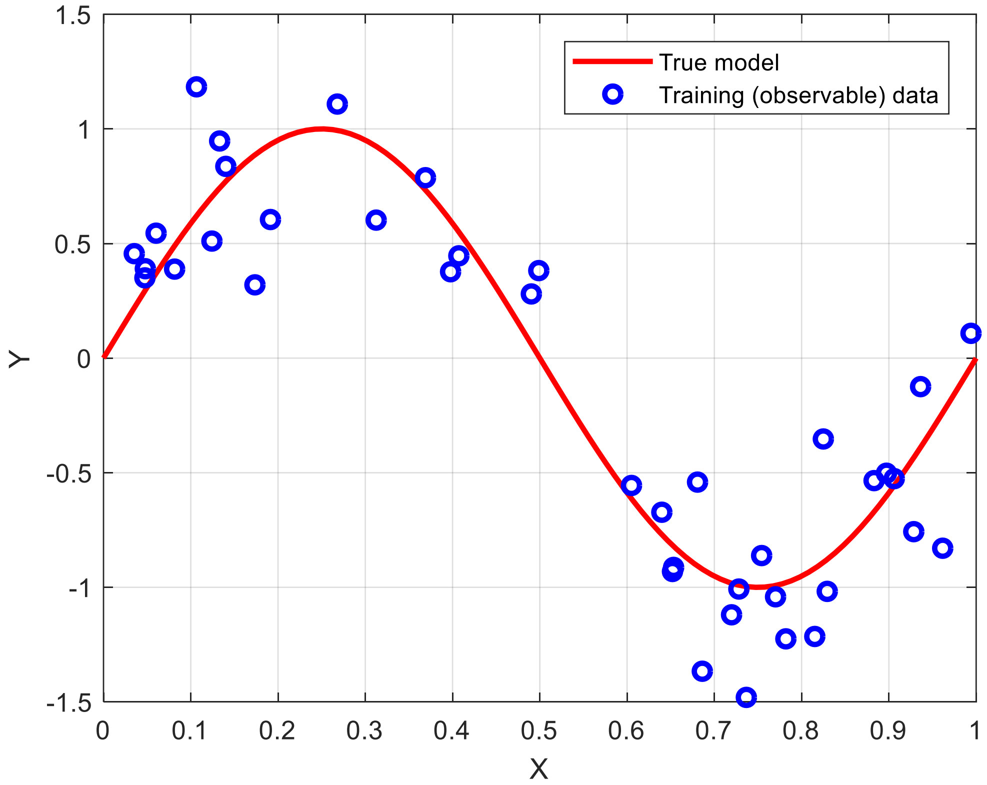
# True model and observable data

Typically, a “true” model is given by *continuous* function: $y^{true} = f(x) = \sin(2\pi x)$

The measured (observed) data is *discrete* and corrupted by *noise*: $y_k = f(x_k) + n_k = \sin(2\pi x) + n_k$

Noise assumed to be Gaussian: $n_k \sim N(0, \sigma^2)$



# Learning the (linear ) model

Our assumptions (guesses) on the model :

$$\hat{y}(x_k, \mathbf{w}) = w_0 + w_1 x_k + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x_k^j$$

Or

$$\hat{y}(x_k, \mathbf{w}) = w_0 + w_1 \sin(2\pi x_k) + w_2 \sin(2\pi x^2) + \ldots + w_M \sin(2\pi x^M) = \sum_{j=0}^{M} w_j \sin(2\pi x_k^j)$$

Or

$$\hat{y}(x_k, \mathbf{w}) = w_0 + w_1 \sin(2\pi x_k) + w_2 \sin^2(2\pi x) + \ldots + w_M \sin^M(2\pi x) = \sum_{j=0}^{M} w_j \sin^j(2\pi x_k)$$

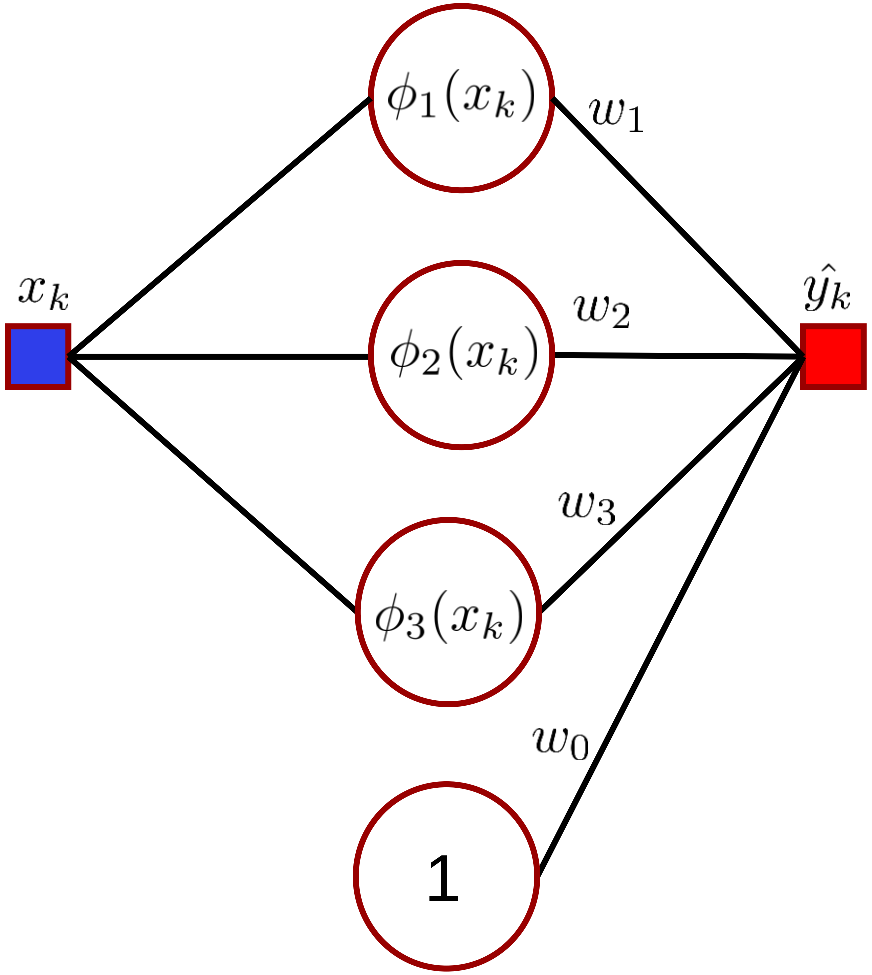
General linear model:

$$\boxed{\hat{y}(x_k, \mathbf{W}) = w_0 + \sum_{j=1}^{M} w_j \phi_j(x_k)}$$

Once we have chosen the model the task is to determine the weights: $\mathbf{W} = [w_0, w_1, \ldots, w_M]$

Procedure: choose a model then determine the weights

# Topology of the linear model

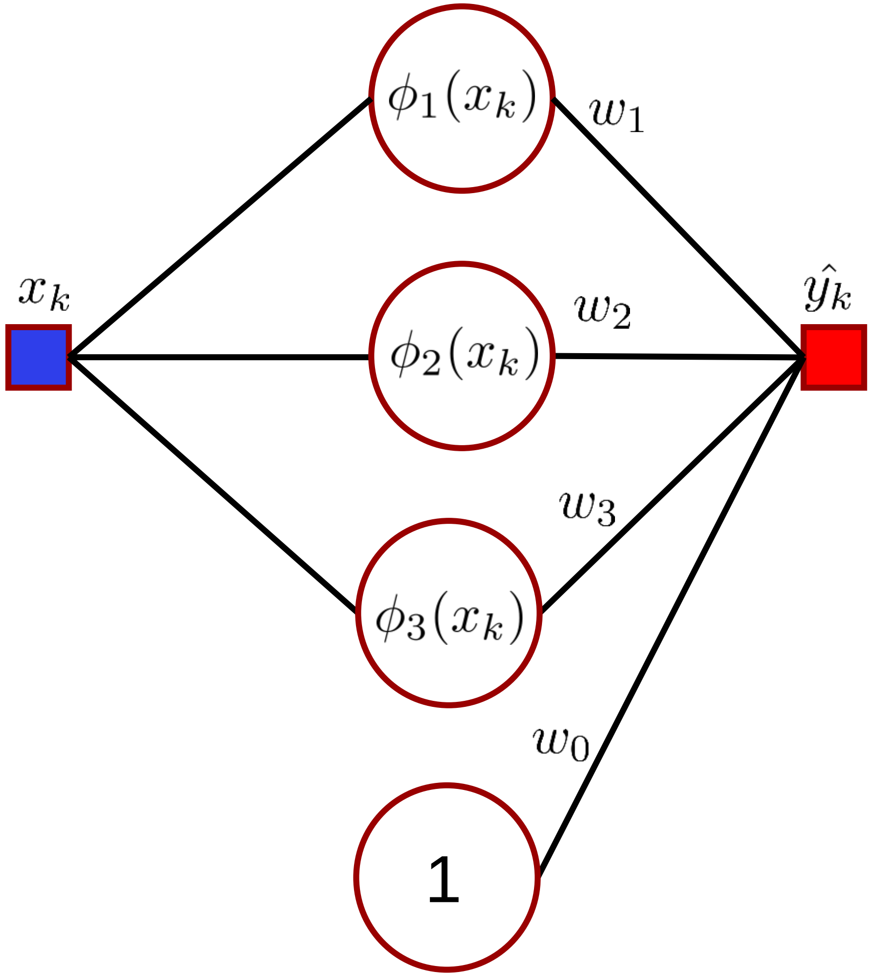



=

$$\hat{y}(x_k, \mathbf{W}) = w_0 + \sum_{j=1}^{3} w_j \phi_j(x_k)$$

# Determining the weights

Gradient descent for weights update: $\mathbf{W}_{(k+1)} = \mathbf{W}_k - \eta\nabla\frac{\partial e_k}{\partial \mathbf{W}_k}$




Defining the mean square error:

$$e_k = \frac{1}{2}[y_k - \hat{y}(x_k, \mathbf{W}_k)]^2 = \frac{1}{2}[y_k - w_0 - \sum_{j=1}^{M} w_j\phi_j(x_k)]^2$$

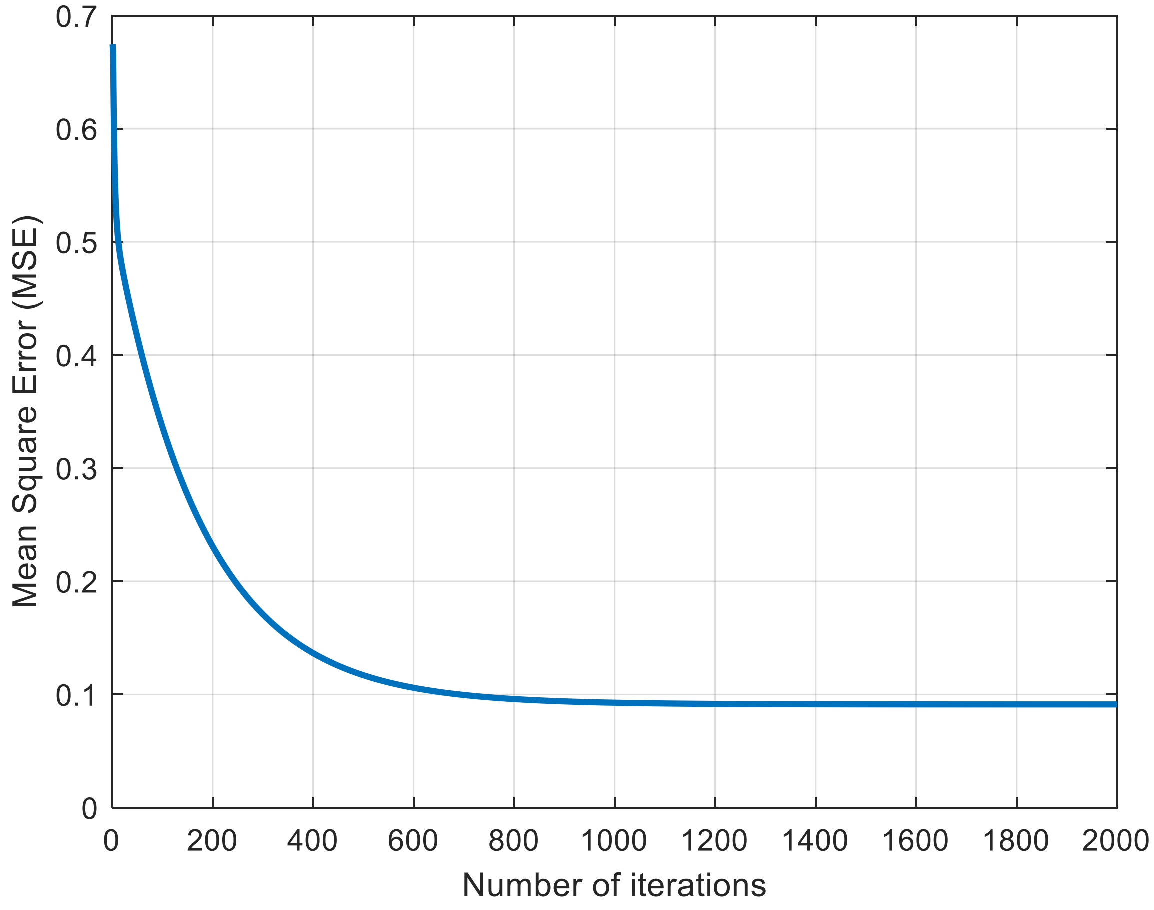
$$= \frac{1}{2}[y_k - \mathbf{W}_k^T\mathbf{\Phi}_k]^2$$

Computing the gradient: $\frac{\partial e_k}{\partial \mathbf{W}_k} = -[y_k - \mathbf{W}_k^T\mathbf{\Phi}_k]\mathbf{\Phi}_k$

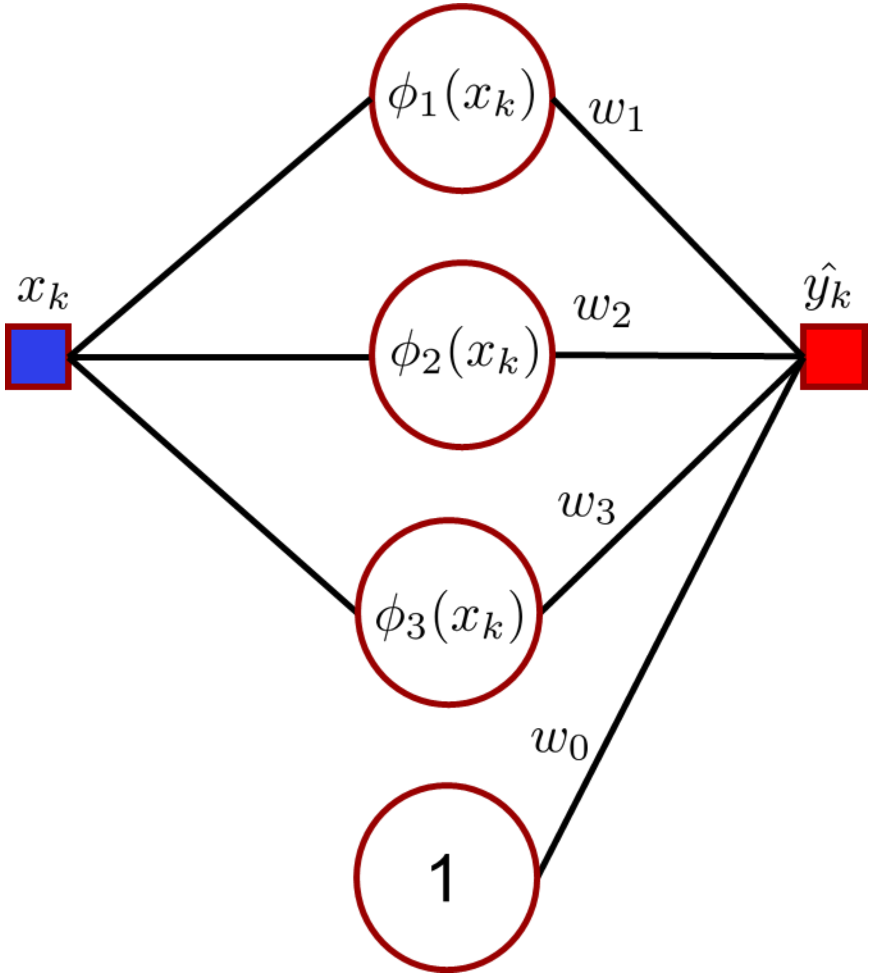
```
for i = 1 : L_iter
    for j = 1 : L_train
        grad = de(j)dw;
        W = W + eta*grad;
    end
    e(i) = mean(e.^2);
end
```

$$\mathbf{W} = [w_0, w_1, ..., w_M]^T \quad \mathbf{\Phi}_k = [1, \phi_1(x_k), ..., \phi_M(x_k)]^T$$

# Performance evaluation on training data-set



# Determining the weights in one step




The output is expressed as:

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \cdot \\ \hat{y}_N \end{bmatrix} = \begin{bmatrix} 1 & \phi_1(x_1) & \phi_2(x_1) & \phi_3(x_1) \\ 1 & \phi_1(x_2) & \phi_2(x_2) & \phi_3(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(x_N) & \phi_2(x_N) & \phi_3(x_N) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$$\boxed{\mathbf{Y} = \mathbf{\Phi W}}$$

The weights are computed as Moore-Penrose pseudo inverse:

$$\boxed{\mathbf{W} = \left(\mathbf{\Phi}^T \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^T \mathbf{Y}}$$

If $\mathbf{\Phi}$ is square and invertible:

$$\left(\mathbf{\Phi}^T \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^T = \mathbf{\Phi}^{-1}$$

Gather input/output data:

$$\mathcal{D} = \{x(k), y(k)\}_{k=1}^{K}$$
$$\mathcal{D} = \{\mathbf{X}, \mathbf{Y}\}$$

Split the data

Training data:

$$\mathcal{D} = \{\mathbf{X}^{train}, \mathbf{Y}^{train}\}$$

Test data:

$$\mathcal{D} = \{\mathbf{X}^{test}, \mathbf{Y}^{test}\}$$

Select the model:
neural network, polynomial, etc

Learn the model parameters:

$$\mathbf{W} = [w_0, w_1, ..., w_M]$$

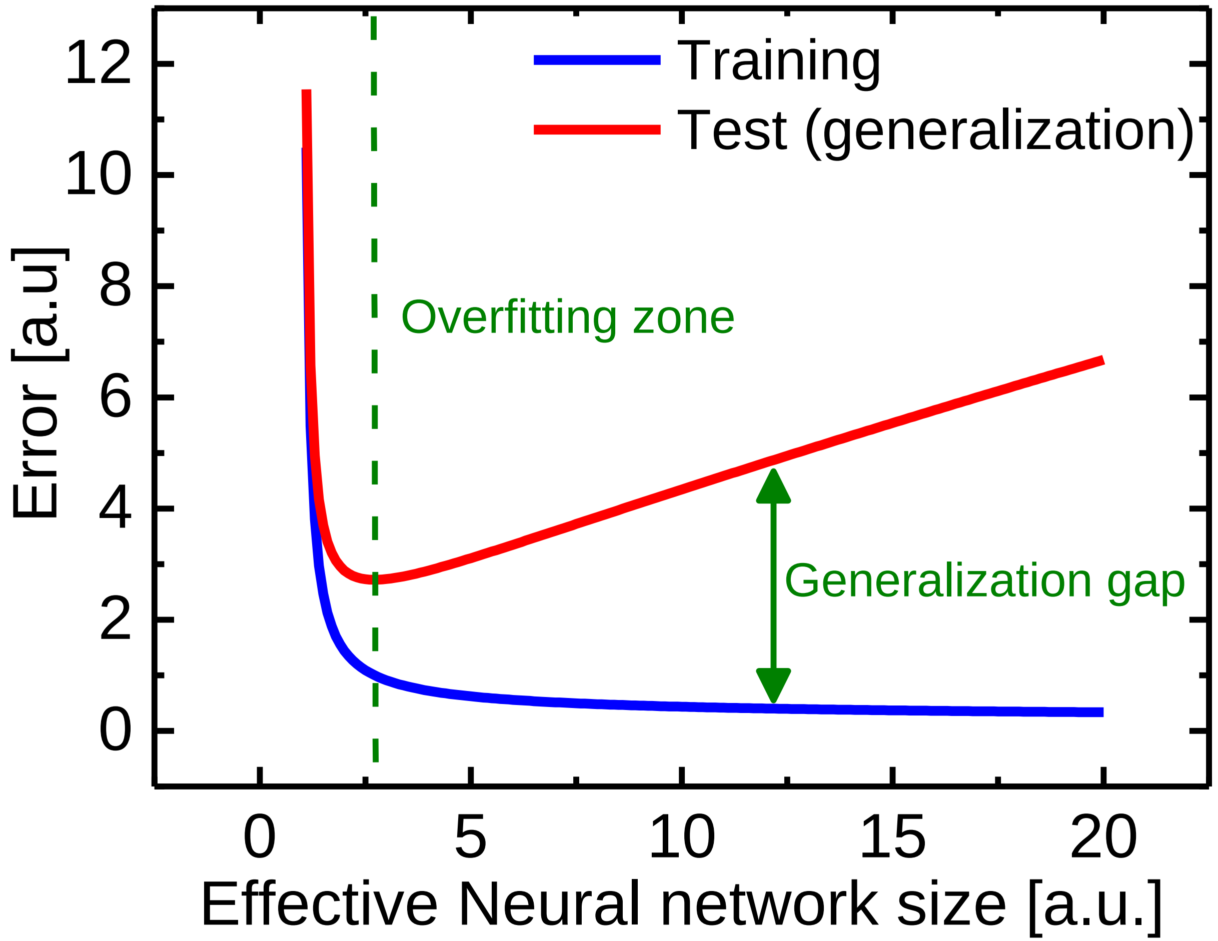
Evaluate the model:

$$\mathbf{Y}^{pred.} = \mathbf{\Phi}(\mathbf{X}^{\mathbf{test}})\mathbf{W}$$

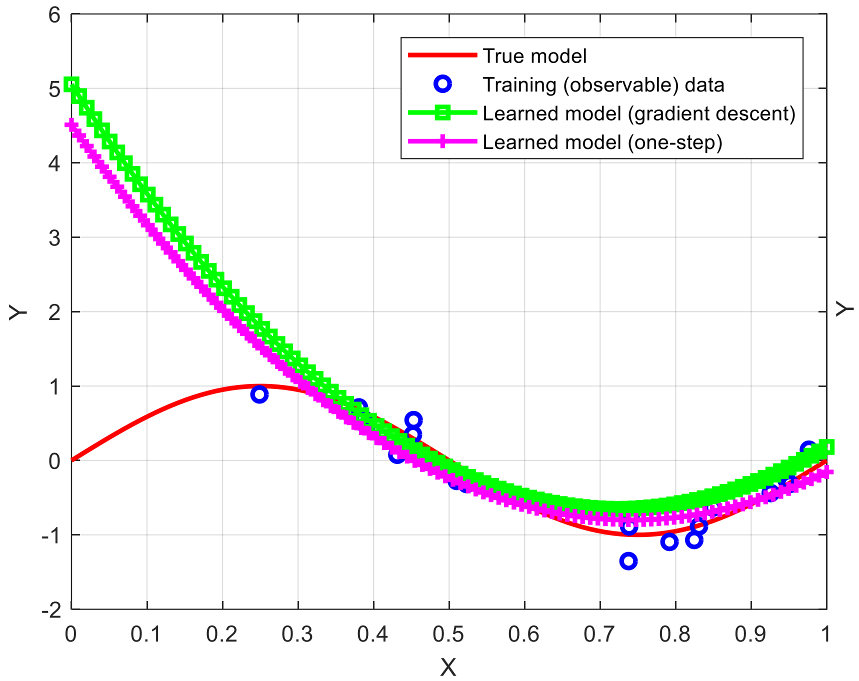
Evaluate the error:

$$e = \sum(\mathbf{Y}^{test} - \mathbf{Y}^{pred})^2$$

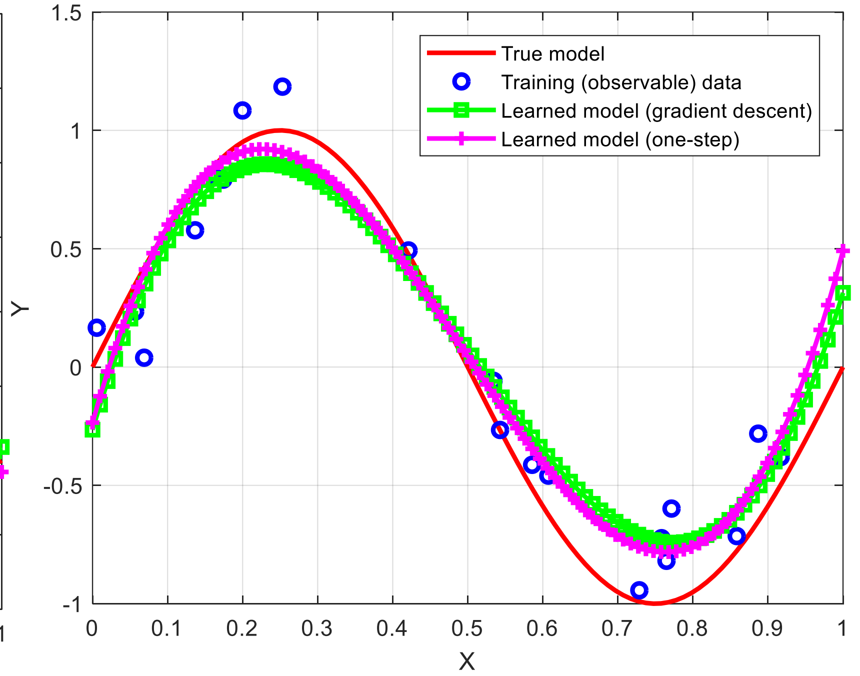
# Key performance metric: generalization



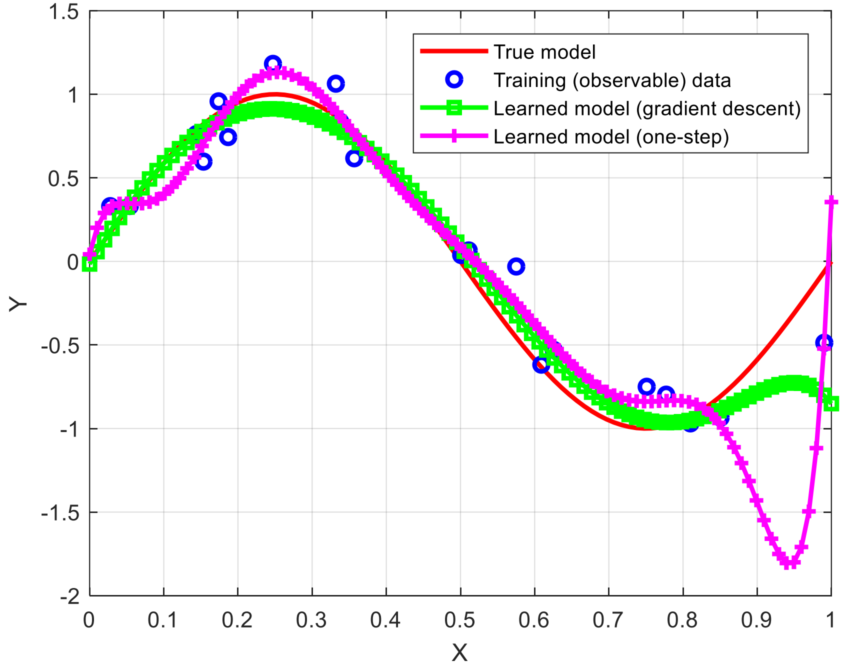
# Evaluating the learned the model




*Assumed polynomial model order M=2*
*(too simple model: underfitting)*



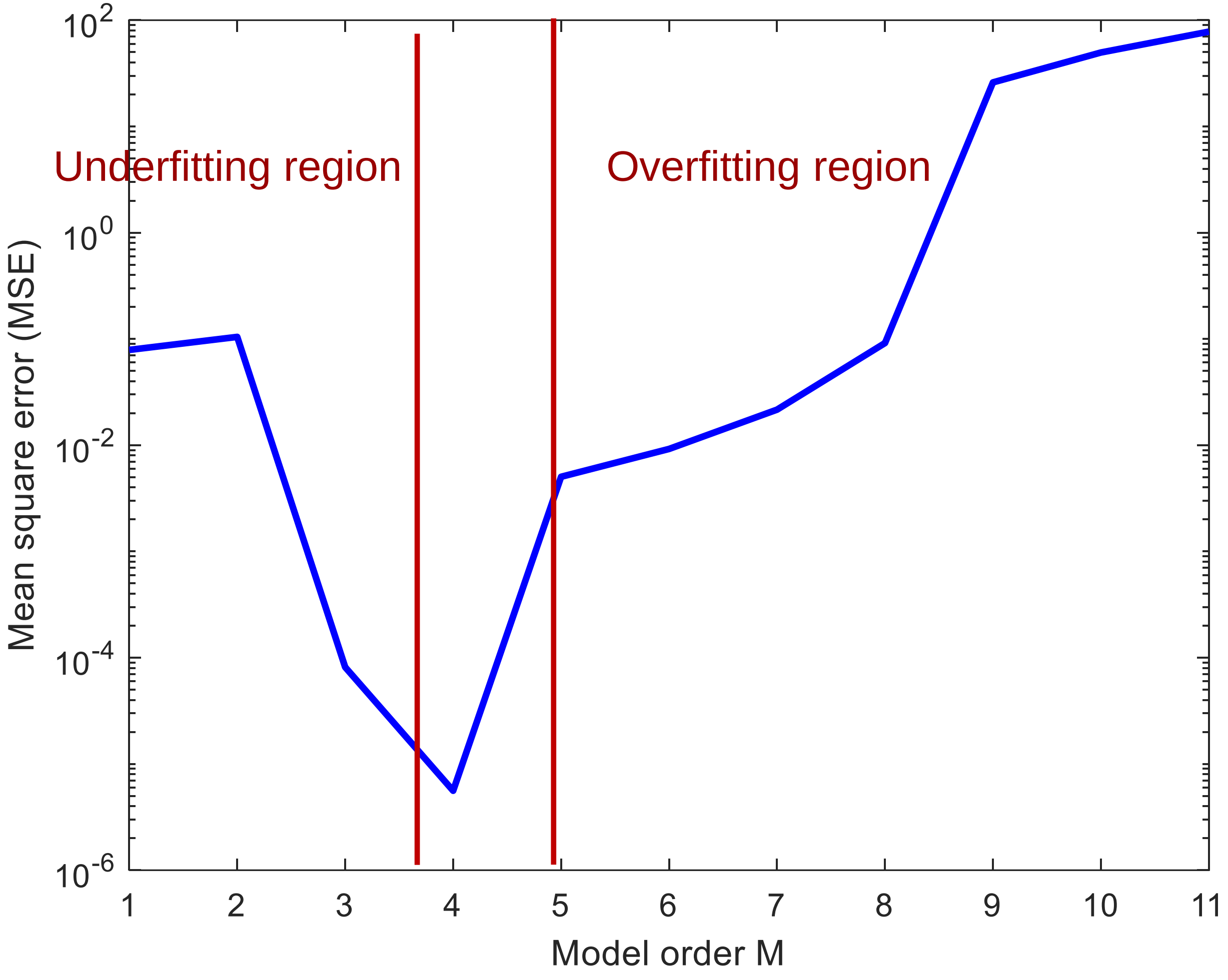

*Assumed polynomial model order M=3*
*(the right model)*




*Assumed polynomial model order M=9*
*(too complex model: overrfitting)*

Use learned weights to compute: $\boxed{\mathbf{Y} = \mathbf{\Phi W}}$ where $x \in \{0:1\}$

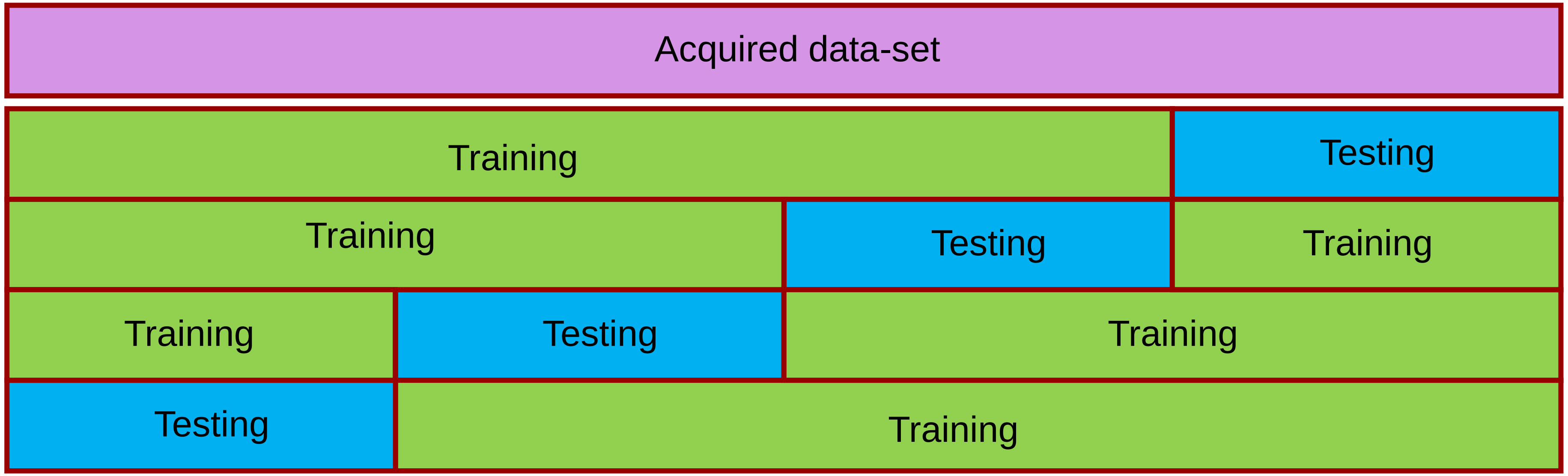
# Evaluating different models on test data



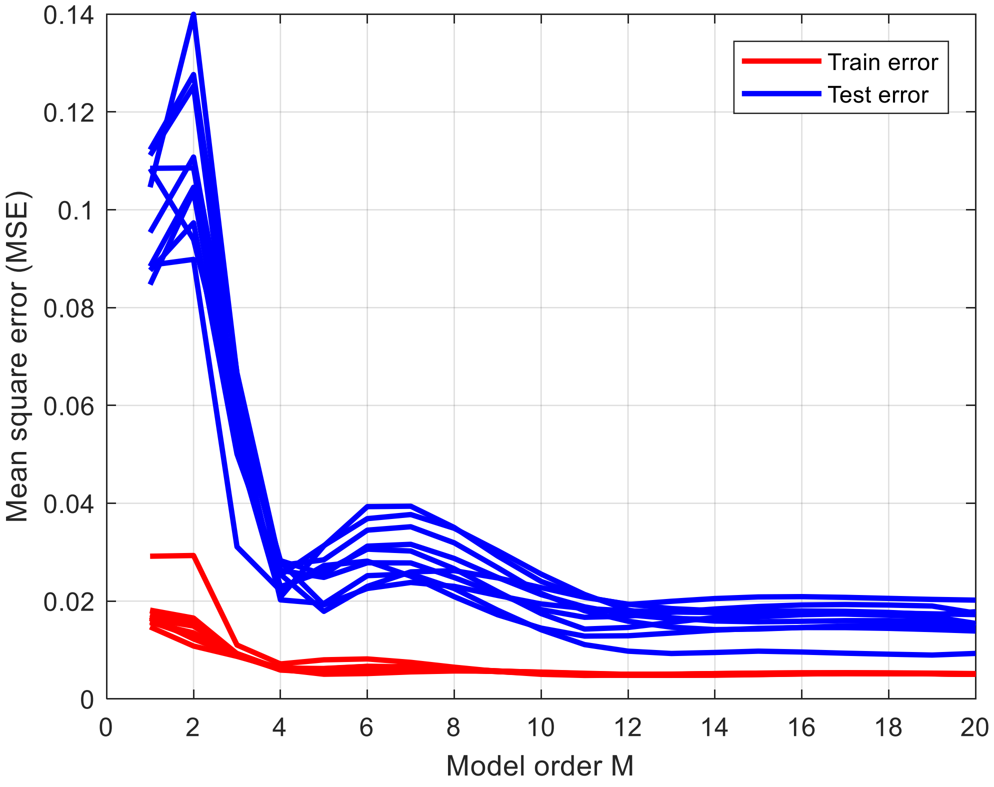

$$MSE = \frac{1}{2}\sum_{k=0}^{N}[y_k^{true} - \mathbf{\Phi}(x_k)\mathbf{W}]^2$$

# Cross-validation

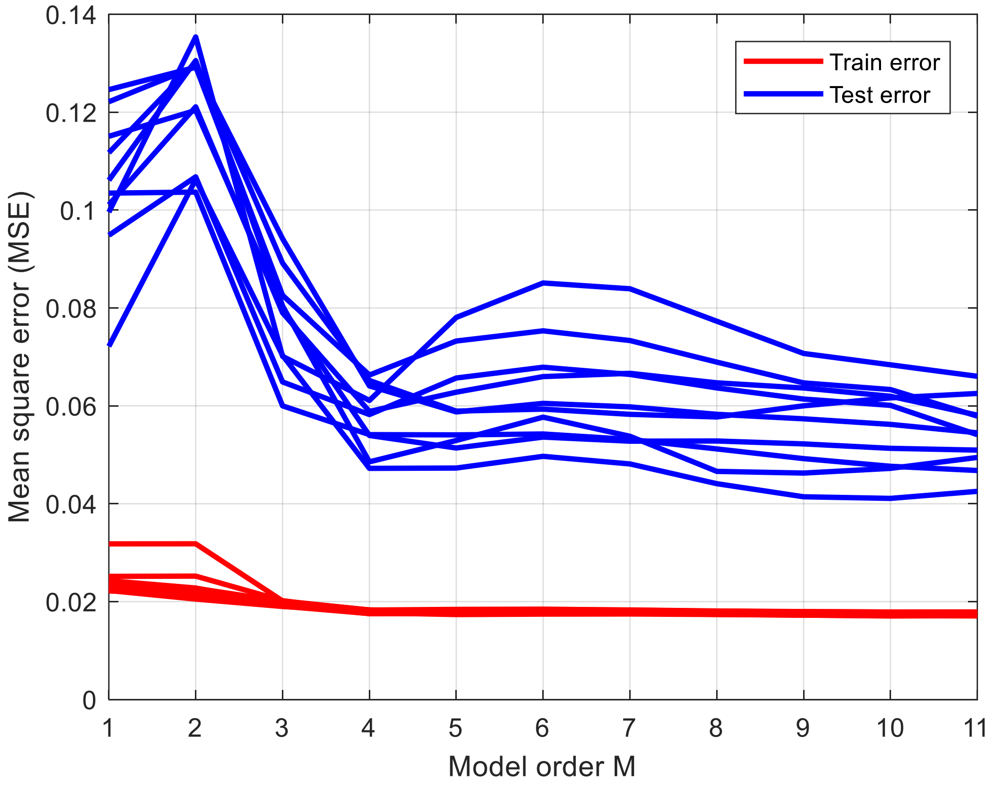
- Typically, the true mode is unknown
- We only have access to noisy observable data (data set)
- Observable data needs to be used for training and test
- Split the observable data into folds for training and test
- Perform weights estimation (learning) and testing for each fold
- Compute average test error as a function of model size



# 10-fold cross validation



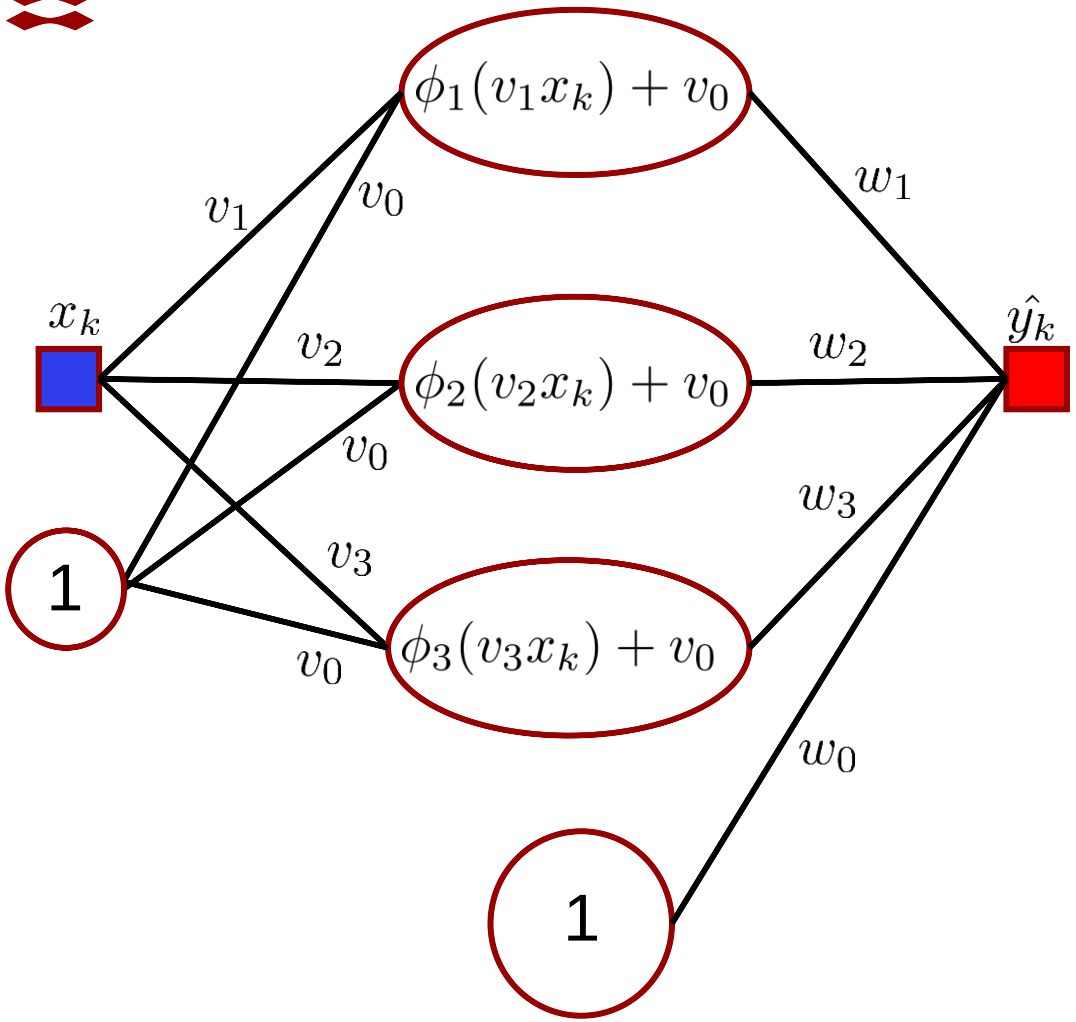

Noise variance 0.0001




Noise variance 0.09

Employ averaging over different test sets to find the model order

# Nonlinear model (neural-network)




$$= \boxed{\hat{y}(x_k, \mathbf{w}) = w_0 + \sum_{j=1}^{3} w_j \phi_j(v_j x_k + v_0)}$$

Weights that need to be learned:

$$\mathbf{W} = [w_0, w_1, ..., w_3]^T$$
$$\mathbf{V} = [v_0, v_1, ..., v_3]^T$$

The problem of determining the weights becomes nonlinear:

$$\boxed{\mathbf{Y} = \mathbf{\Phi}(\mathbf{V})\mathbf{W}}$$

We can no longer perform matrix inversion to find the weights!

# Random weight initialization trick

$$\begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \cdot \\ \hat{y}_N \end{bmatrix} = \begin{bmatrix} 1 & \phi_1(v_1 x_1 + v_0) & \phi_2(v_2 x_1 + v_0) & \phi_3(v_3 x_1 + v_0) \\ 1 & \phi_1(v_1 x_2 + v_0) & \phi_2(v_2 x_2 + v_0) & \phi_3(v_3 x_2 + v_0) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(v_1 x_N + v_0) & \phi_2(v_2 x_N + v_0) & \phi_3(v_3 x_N + v_0) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Assign random weights to $\mathbf{V} = [v_0, v_1, v_2, v_3]$ by sampling from a normal distribution with variance:

$$\mathbf{V} \sim \mathcal{N}(0, \sigma \mathbf{I})$$

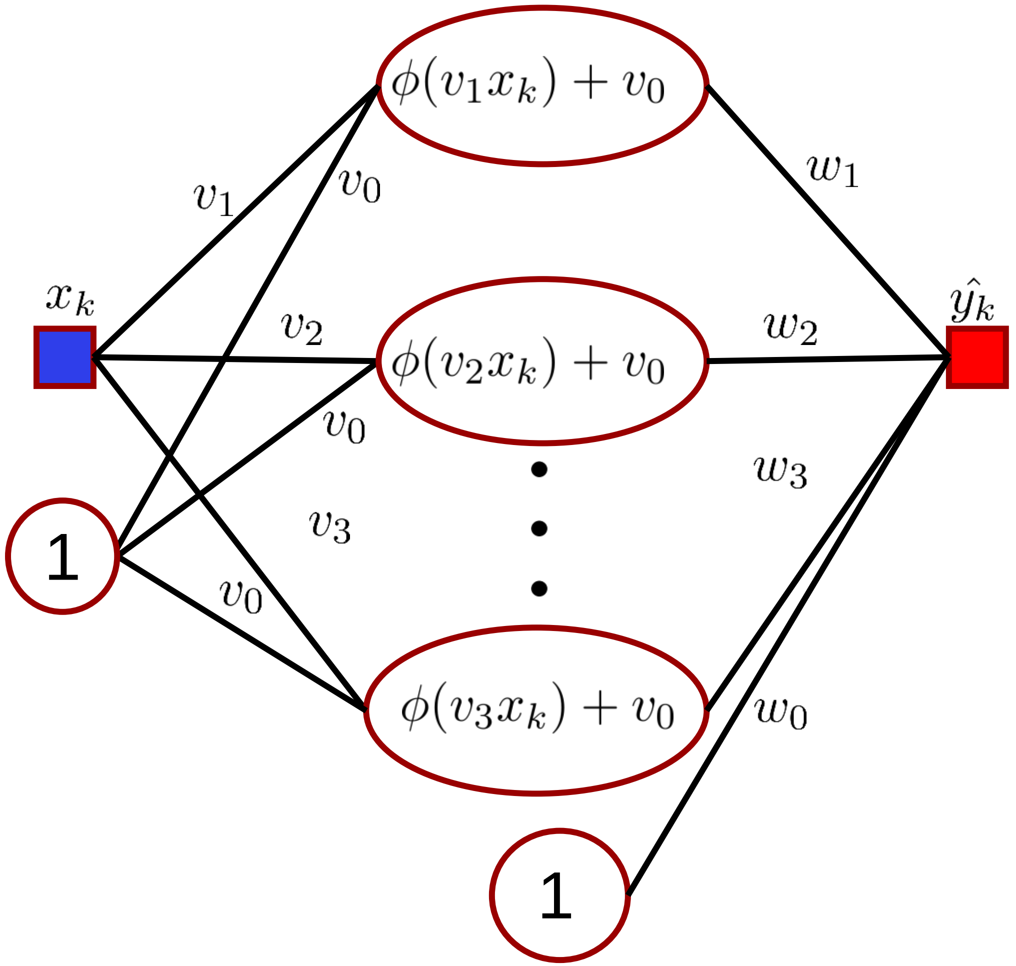
$\mathbf{I} : 3 \times 3$ identity matrix

$\sigma$ : standard deviation of the weights (tuning parameter)

Finally, we use Moore-Penrose pseudo inverse to compute the outer weights:

$$\boxed{\mathbf{W} = \left(\mathbf{\Phi}^T \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^T \mathbf{Y}}$$

# Typical one-layer neural network architecture




Common approach is to use single type basis function and very their number

# Examples of common basis/activation functions

Threshold function:

$$\psi(v_k) = \begin{cases} 1 & : v_k \geq 0 \\ 0 & : v_k < 0 \end{cases}$$

Sign function:

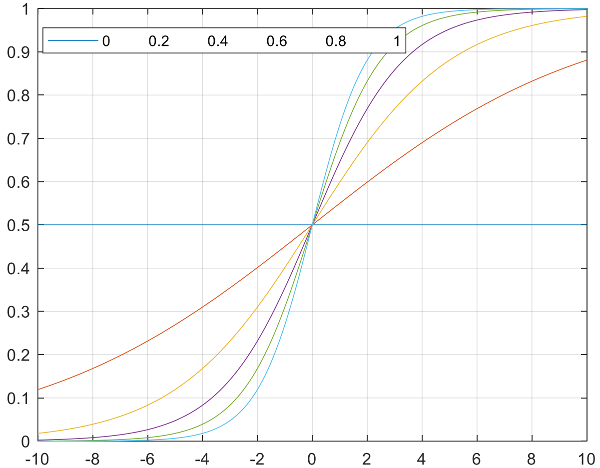
$$\psi(v_k) = \begin{cases} 1 & : v_k \geq 0 \\ -1 & : v_k < 0 \end{cases}$$

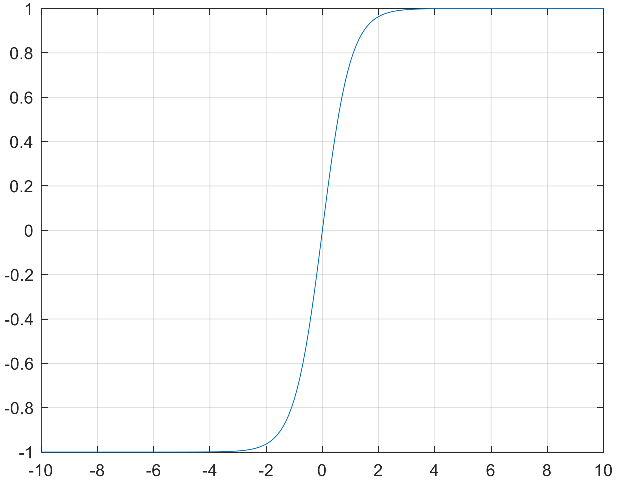
Sigmoid function:

$$\psi(v_k) = \frac{1}{1+\exp(-av_k)}$$

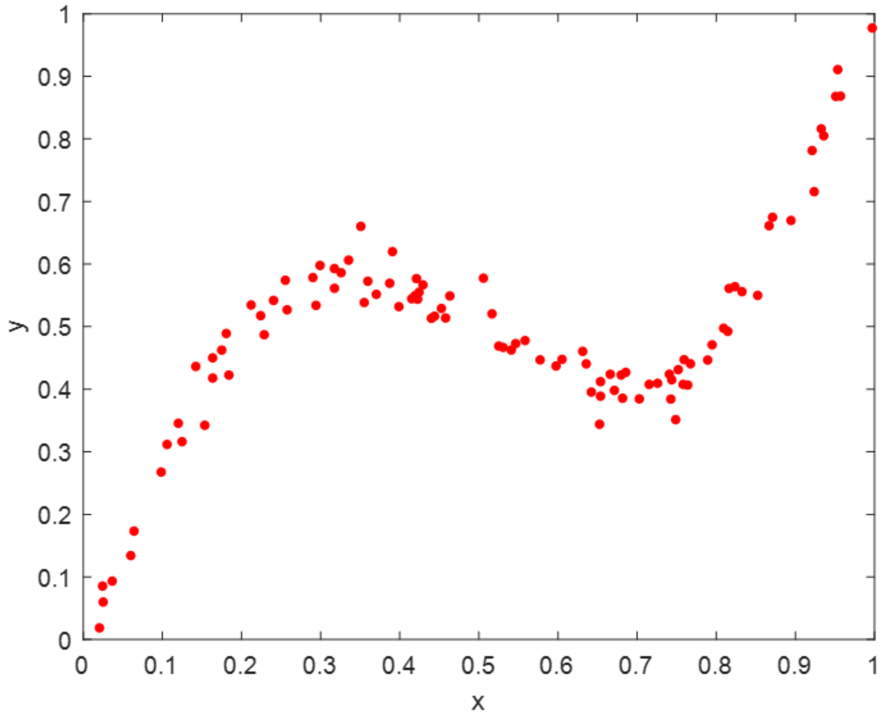
Tangents function:

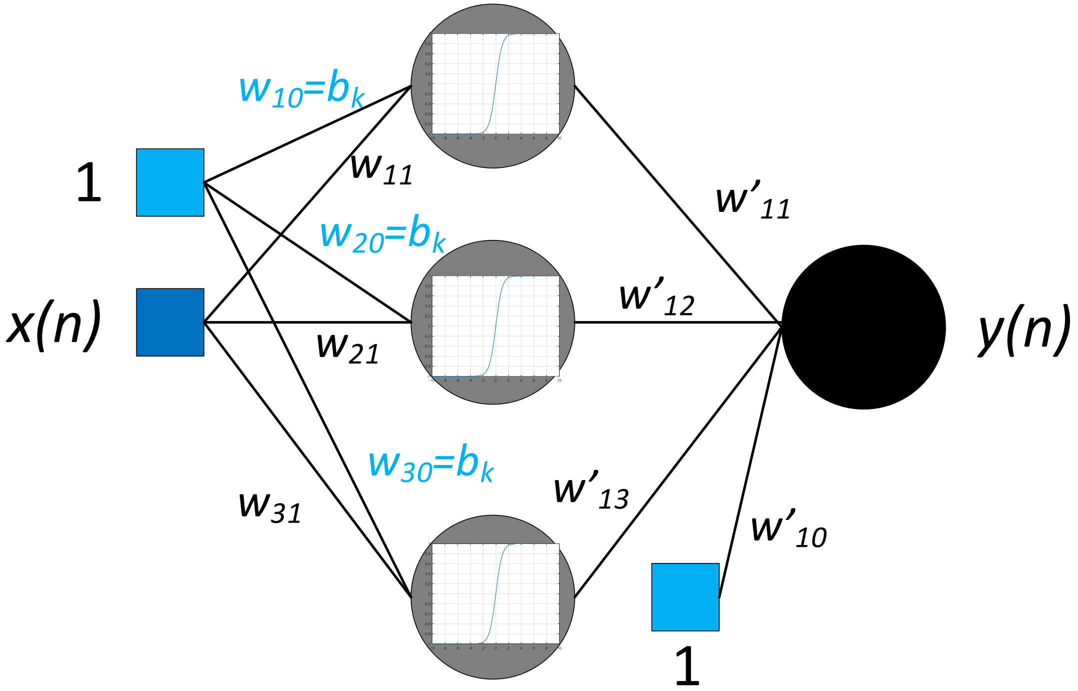
$$\psi(v_k) = \tanh(v_k)$$



# Learn the mapping using neural-network



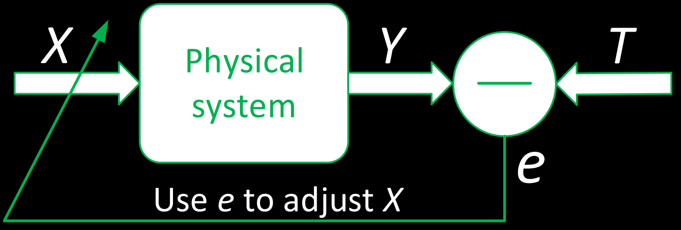




Objective: learn the coefficients *(central problem in machine learning):*

$$\mathbf{W} = \begin{bmatrix} w_{10} & w_{11} & w'_{10} \\ w_{20} & w_{21} & w'_{11} \\ w_{30} & w_{31} & w'_{12} \\ - & - & w'_{13} \end{bmatrix}$$

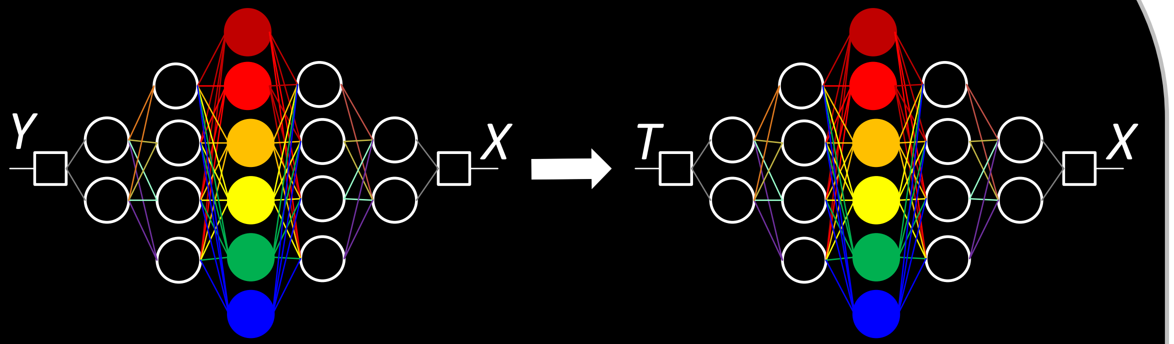
# Inverse system learning

#1 Problem statement:

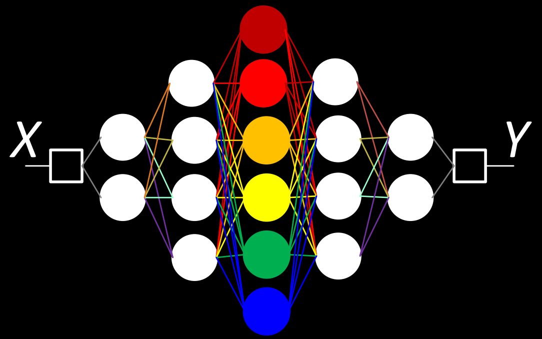



A physical system describing relation between input $X$ and output $Y$ is given. The objective is to determine input $X$ that would result in a targeted output $T$.

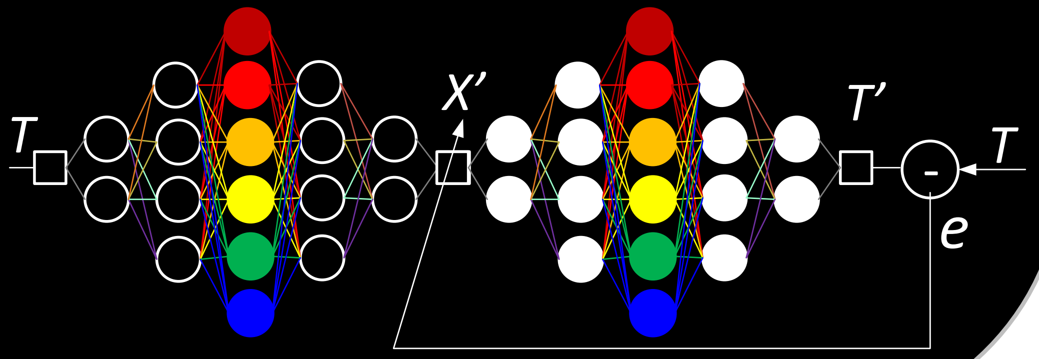
#2 Train neural network to learn *inverse* mapping (from $X$ to $Y$):




#3 Train neural network to learn *forward* mapping (from $X$ to $Y$):



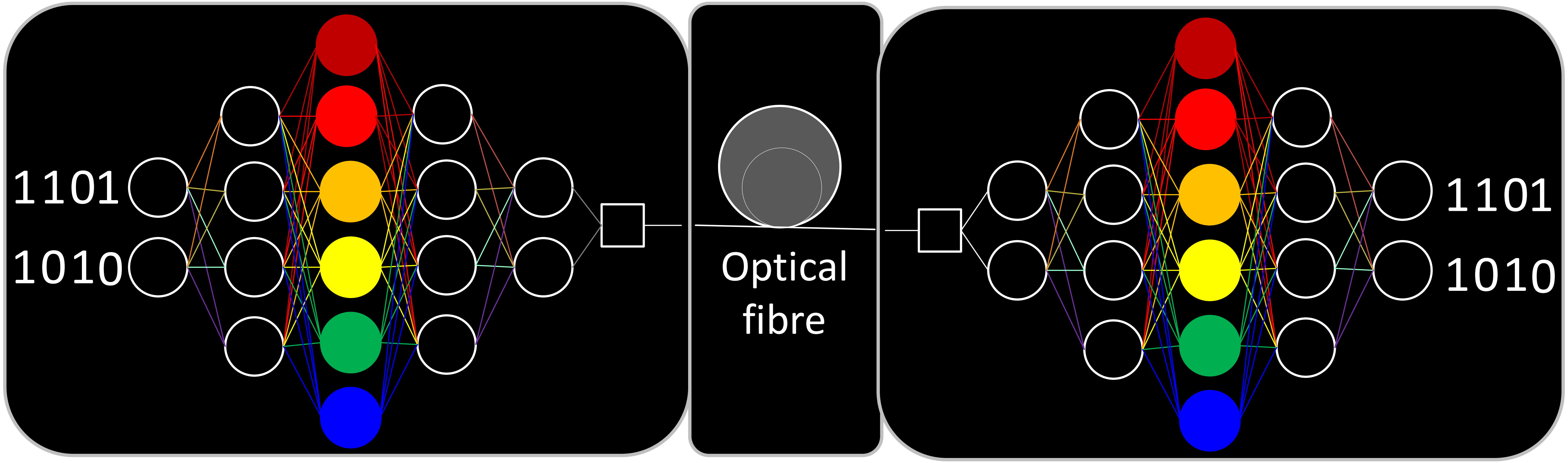

#4 Perform *final* optimization:





[1] D. Zibar et al., "Inverse system design using machine learning: the Raman amplifier case," Journal of Lightwave technology, 2019

# Learning to transmit and receive data over complex channels

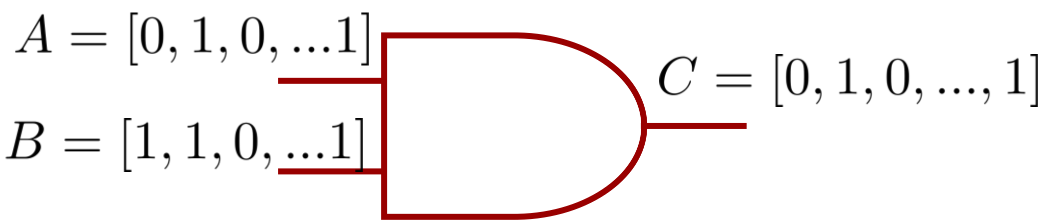



[1] R.Jones, M. Yankov, D. Zibar et al., "End-to-end learning of GMI optimized constellation shapes," ECOC 2019

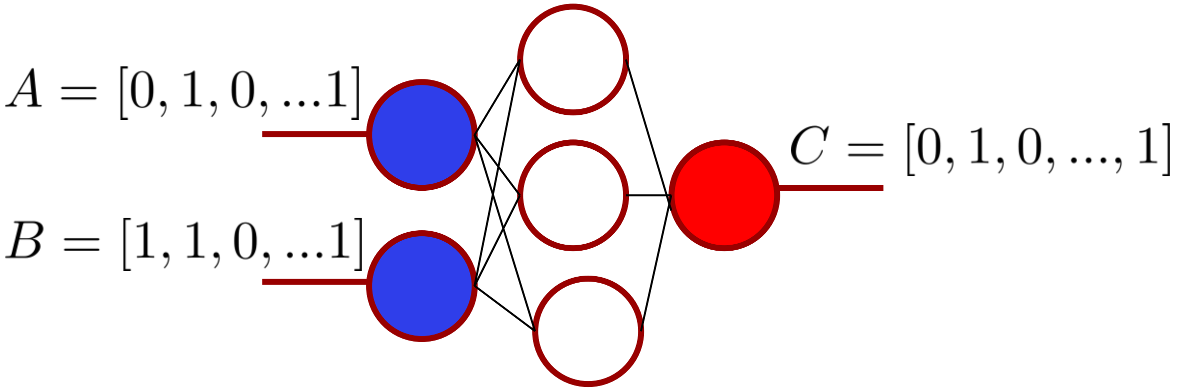
# Can we learn mappings between categorical variables?

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

AND gate

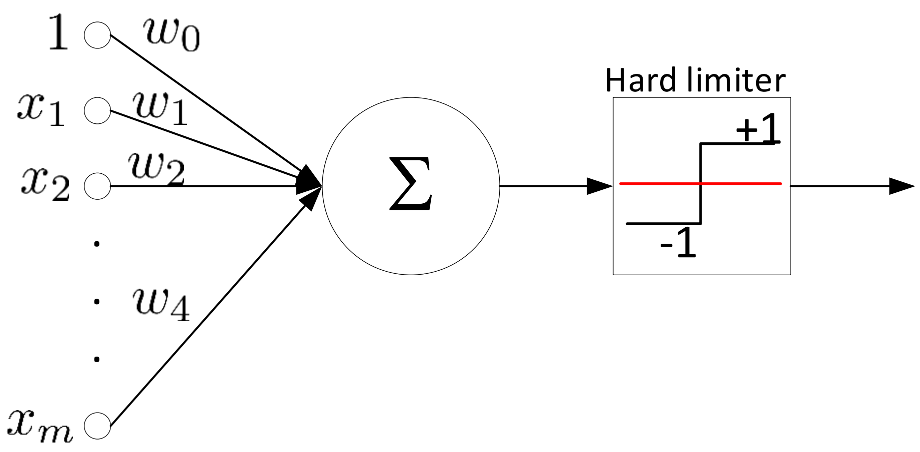



*True model*



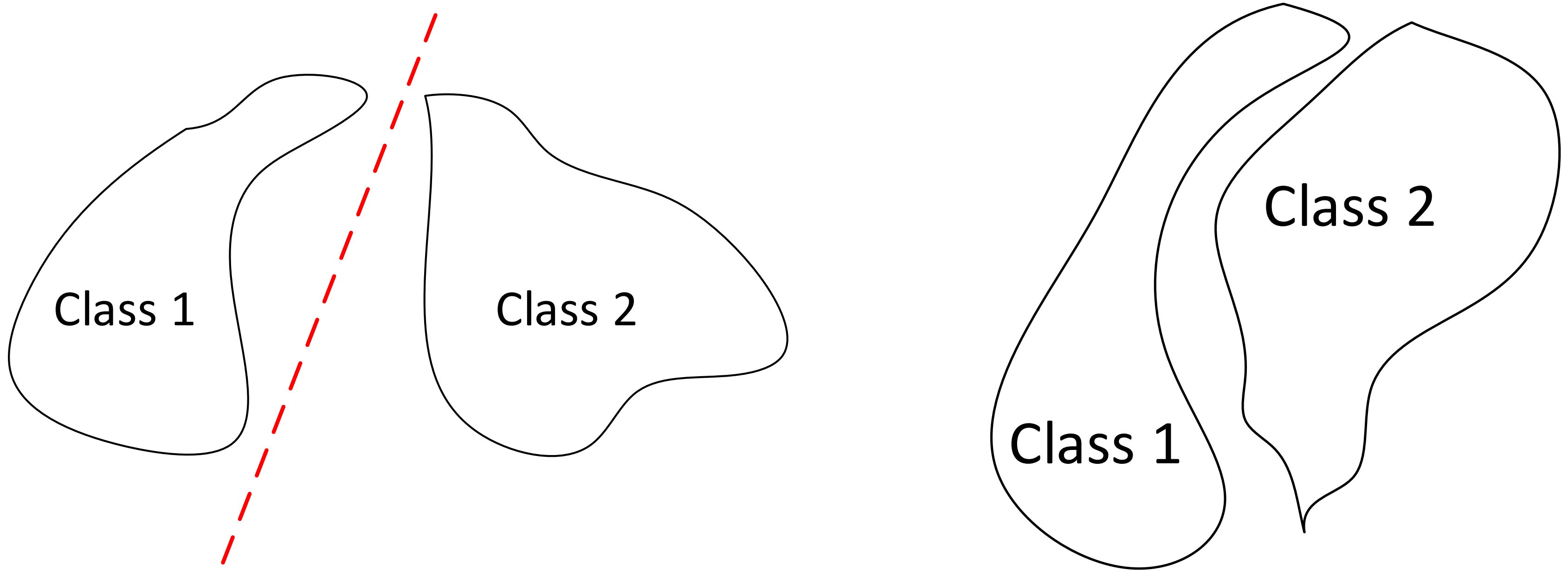

*Learned model*

# Rossenblatt perceptron (simplest neural network)




$$\boxed{v_k = \sum_{j=0}^{m} w_j x_j = \mathbf{W}^T \mathbf{X}}$$

# Linearly separable classes

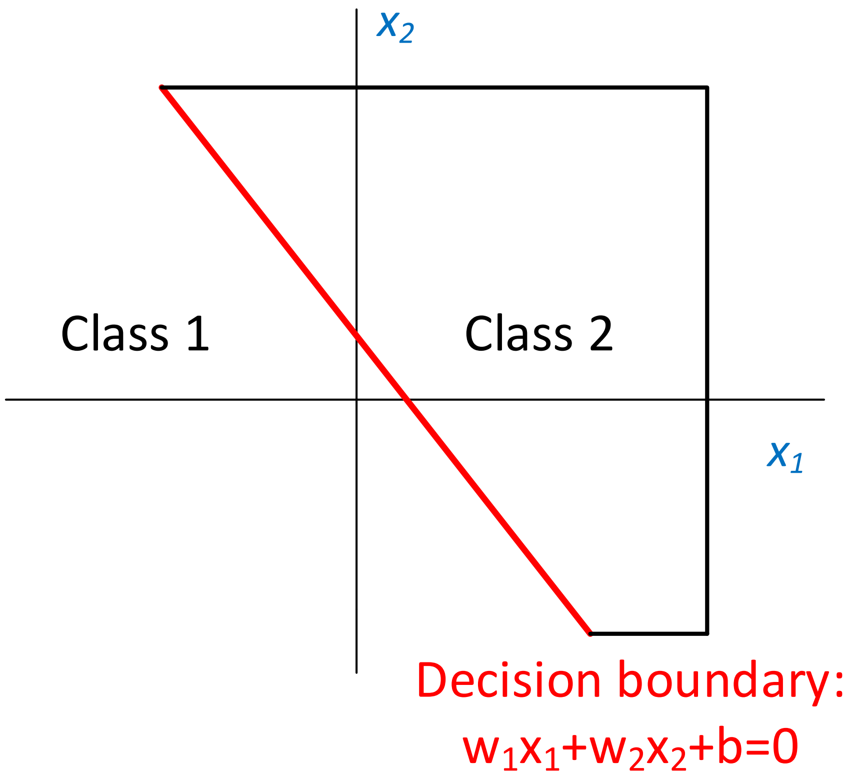



(a) Linearly separable classes

(b) Linearly non-separable classes

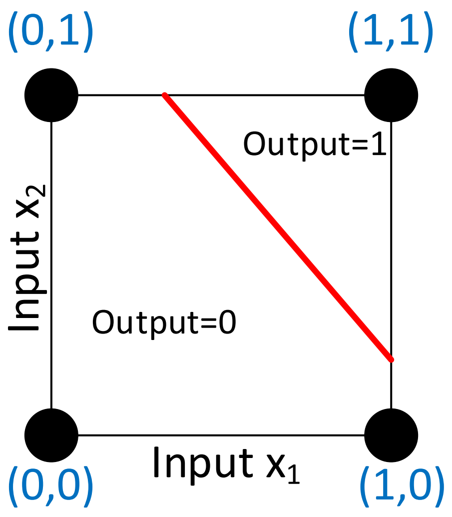
The perceptron only works for linearly separable classes

# Decision boundary



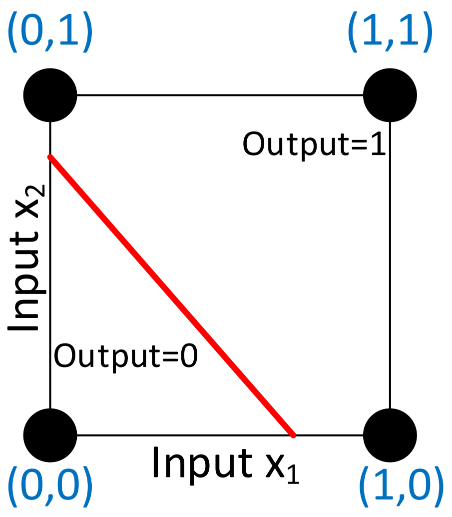

$$v_k = \sum_{j=0}^{m} w_j x_j = \mathbf{w}^T \mathbf{x} = 0$$

# Decision Boundaries (DB)



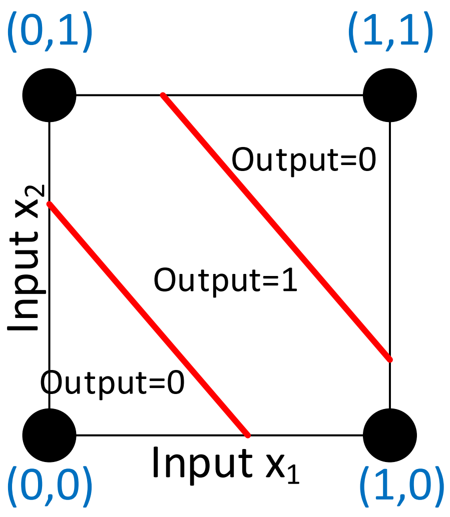

AND gate

*(Linear DB)*




OR gate

*(Linear DB)*




XOR gate

*(Nonlinear DB)*

# Perceptron learning algorithm

**TABLE 1.1** Summary of the Perceptron Convergence Algorithm

*Variables and Parameters:*

$$\begin{aligned}\mathbf{x}(n) &= (m+1)\text{-by-1 input vector}\\ &= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T\\ \mathbf{w}(n) &= (m+1)\text{-by-1 weight vector}\\ &= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T\\ b &= \text{bias}\\ y(n) &= \text{actual response (quantized)}\\ d(n) &= \text{desired response}\\ \eta &= \text{learning-rate parameter, a positive constant less than unity}\end{aligned}$$

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step $n$, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as

$$y(n) = \operatorname{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

   where $\operatorname{sgn}(\cdot)$ is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

   where

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$

5. *Continuation.* Increment time step $n$ by one and go back to step 2.

[1] Simon Haykin, Neural Networks and Learning Machines, pp. 54

# Categorical functions learning

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

AND gate

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

OR gate

| A | B | C |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Exclusive OR gate

1. *Specify the length of the training and test data set*
2. *Generate data sets by uniform sampling from one of the tables*
3. *Run the perceptron learning algorithm and learn the weights*
4. *Run the validation on the test data*

# In this lecture we have learned….

- What machine learning is
- How to build linear and nonlinear models from data
- Difference between training and test set
- How to estimate the weights for linear and nonlinear models
  - Gradient descent
  - Matrix inversion
- How to evaluate learned model
- Rosenblatt perceptron
- How to learn logical gates