

Introduction to machine learning and linear regression models

Darko Zibar
dazi@dtu.dk

Agenda

- Applications of machine learning in communication engineering
- Various types of machine learning
- Building models from data: data-driven modelling
- Learning the model parameters
- Model evaluation
- Rosenblatt perceptron
- Learning of logical gates (AND, OR and NOR)

<https://kurser.dtu.dk/course/2024-2025/34242>

*NEW 3-WEEK COURSE: 34242 Machine learning techniques for design
and optimization of communication systems*

Course responsible:



DTU ELECTRO
Institut for Elektroteknologi og Fotonik
Ørsteds Plads
Bygning 343 , Rum 124
Email: dazi@dtu.dk
Phone: 61464688

Course Motto: *learning by doing it yourself*

Reading material

1. Simon Haykin, Neural Networks and Learning Machines

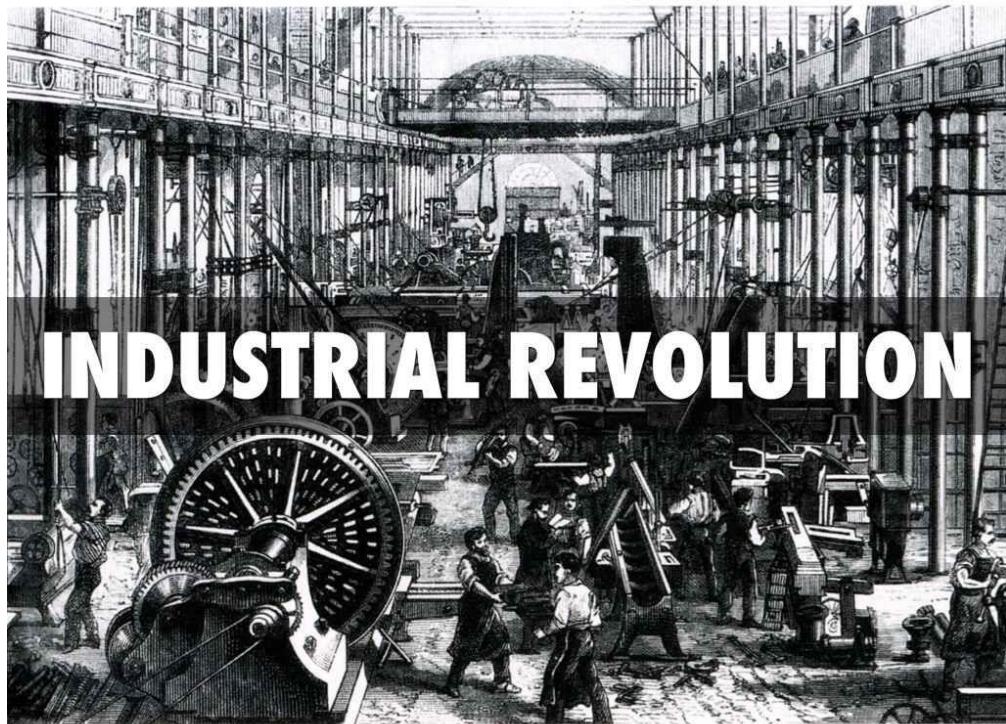
- Introduction 1,
- Chapter 1

2. Christopher M. Bishop, Pattern Recognition and Machine Learning,
Springer 2006

- Introduction (pp. 1-12)
- Chapter 3 (pp. 137-144)

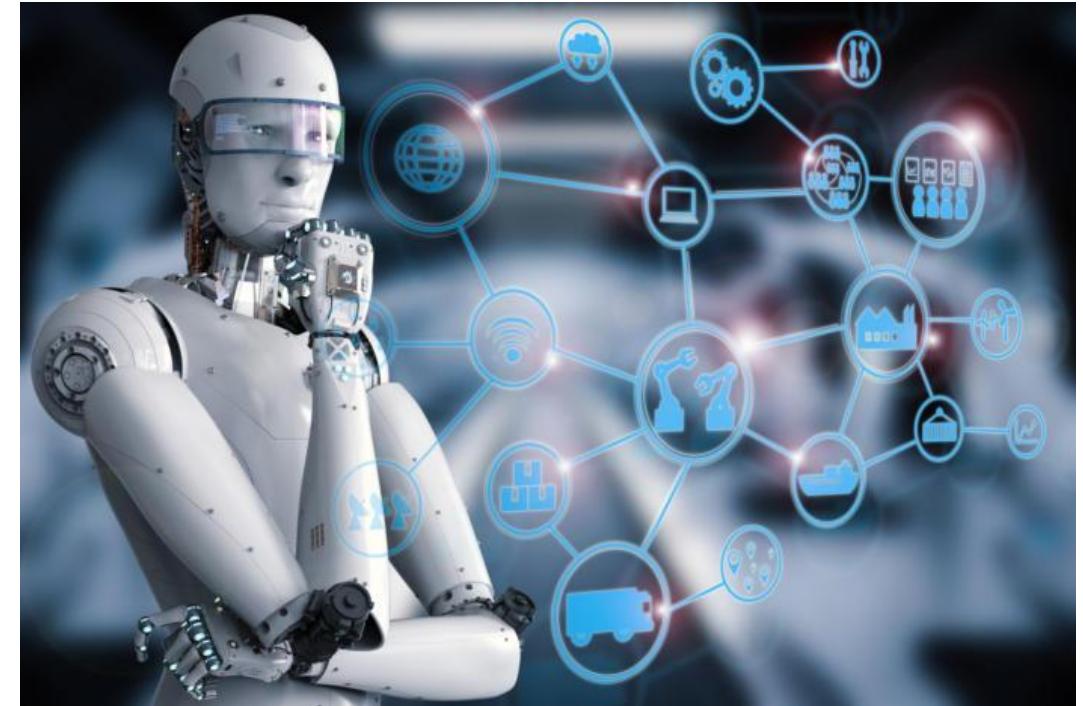
<http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf>

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



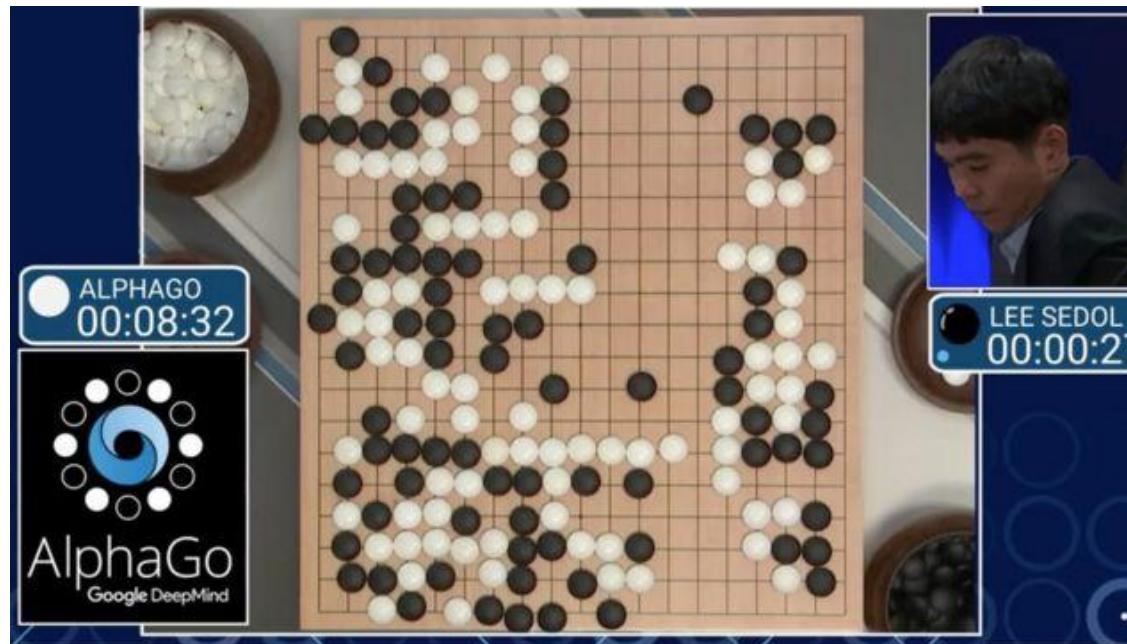
1820-1840: Industrial revolution:

*The development of machine tools
and rise of the factory system*



2007-: AI revolution:

*The development of intelligent
machines and self-learning systems*



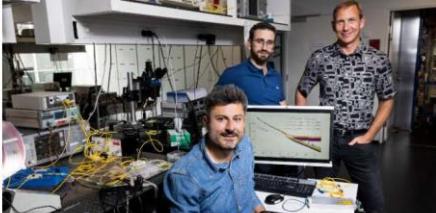
2016: AI beats the world champion of Go

Home / Physics / Optics & Photonics ☆ 📄 🖨

⌚ JANUARY 3, 2022

Minimizing laser phase noise with machine learning

by Henrik Bendix, Technical University of Denmark



Credit: Technical University of Denmark

Ultra-precise lasers can be used for optical atomic clocks, quantum computers, power cable monitoring, and much more. But all lasers make noise, which researchers from DTU Fotonik want to minimize using machine learning.



DeepMind
@DeepMind

Today in @nature, with @EPFL, the first deep reinforcement learning system that can keep nuclear fusion plasma stable inside its tokamaks, opening new avenues to advance nuclear fusion research.

Paper: dpm.dl.fai/fusion-paper

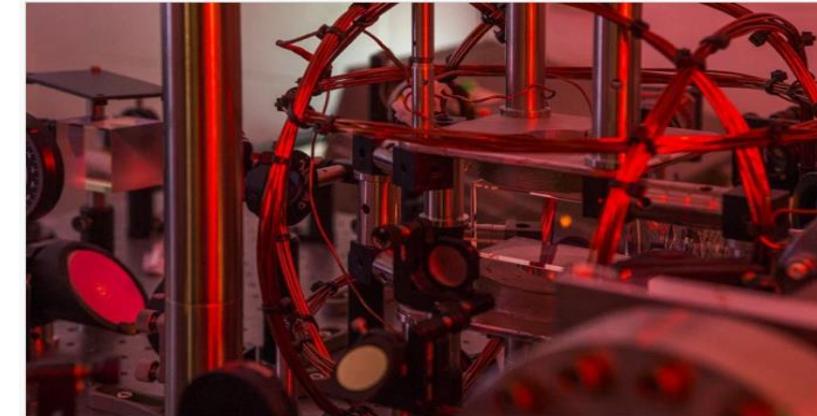


4:11 AM · Feb 17, 2022 · Twitter Web App

420 Retweets 125 Quote Tweets 1,666 Likes

AI learns and recreates Nobel-winning experiment

Posted May 16, 2016 by Devin Coldewey

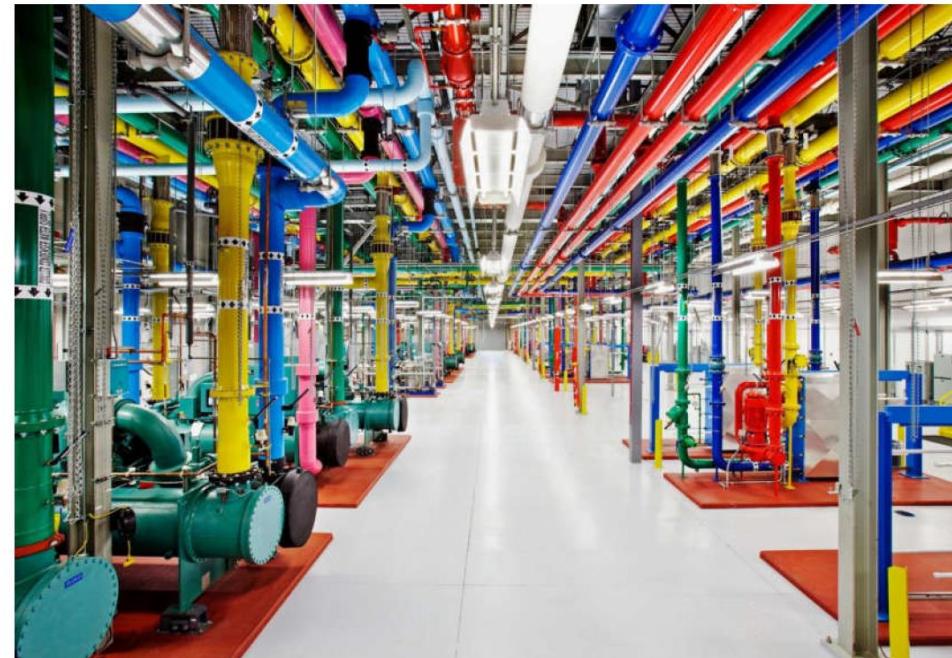


Australian physicists, perhaps searching for a way to shorten the work week, have created an AI that can run and even improve a complex physics experiment with little oversight. The research could eventually allow human scientists to focus on high-level problems and research design, leaving the nuts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

[DeepMind](#) > [Blog](#) > [DeepMind AI Reduces Google Data Centre Cooling Bill by 40%](#)



Learning to reduce power consumption



Google Data Centre

Goal: optimize data center PUE:

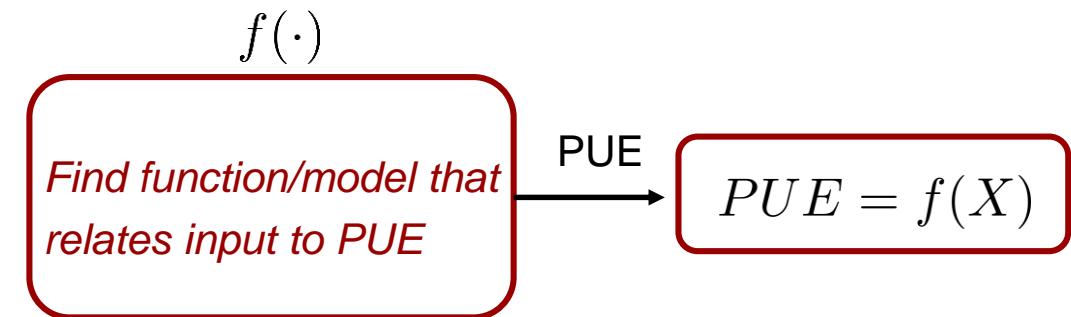
$$\text{PUE} = \frac{\text{Total energy usage}}{\text{IT load usage}}$$

PUE: power usage effectiveness

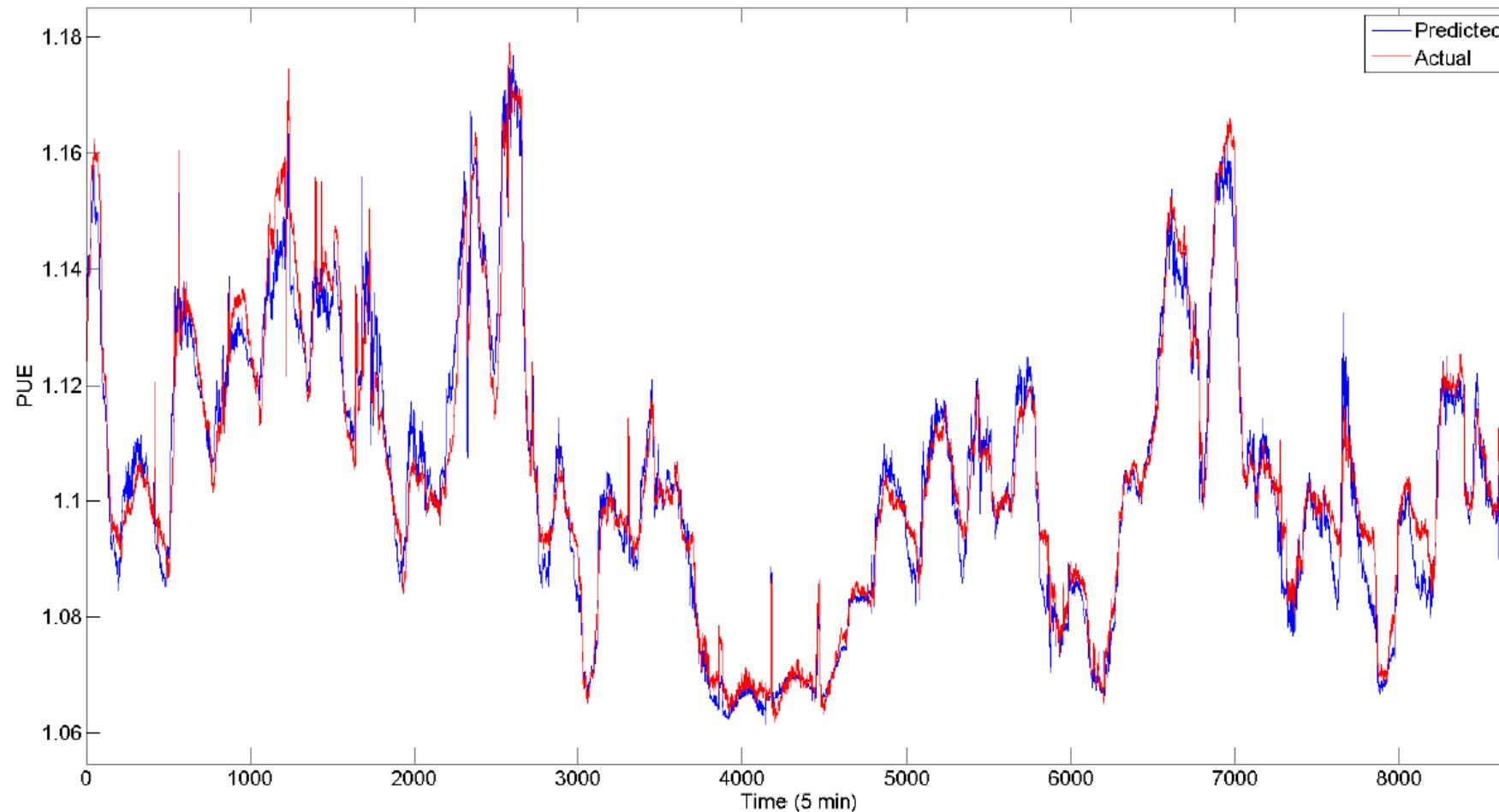
Why use machine learning?

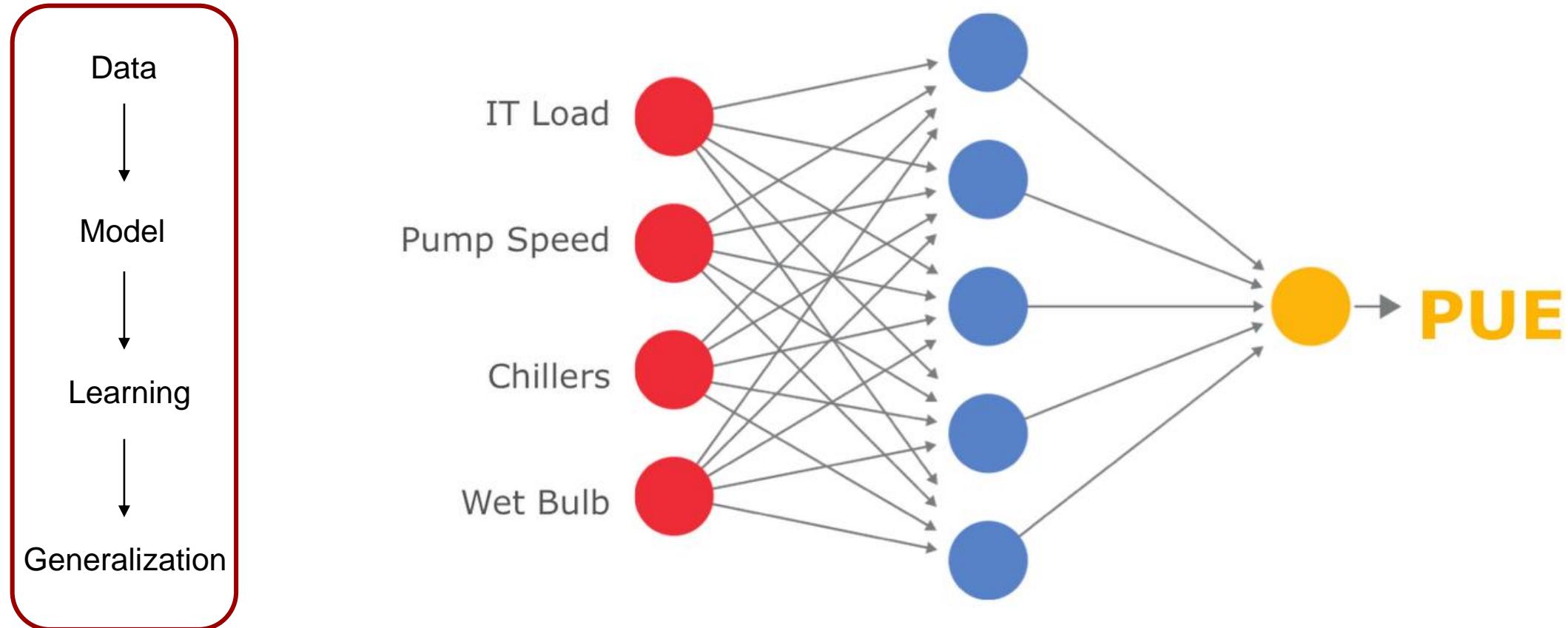
1. Total server IT load [kW]
2. Total Campus Core Network Room (CCNR) IT load [kW]
3. Total number of process water pumps (PWP) running
4. Mean PWP variable frequency drive (VFD) speed [%]
5. Total number of condenser water pumps (CWP) running
6. Mean CWP variable frequency drive (VFD) speed [%]
7. Total number of cooling towers running
8. Mean cooling tower leaving water temperature (LWT) setpoint [F]
9. Total number of chillers running
10. Total number of drycoolers running
11. Total number of chilled water injection pumps running
12. Mean chilled water injection pump setpoint temperature [F]
13. Mean heat exchanger approach temperature [F]
14. Outside air wet bulb (WB) temperature [F]
15. Outside air dry bulb (DB) temperature [F]
16. Outside air enthalpy [kJ/kg]
17. Outside air relative humidity (RH) [%]
18. Outdoor wind speed [mph]
19. Outdoor wind direction [deg]

X

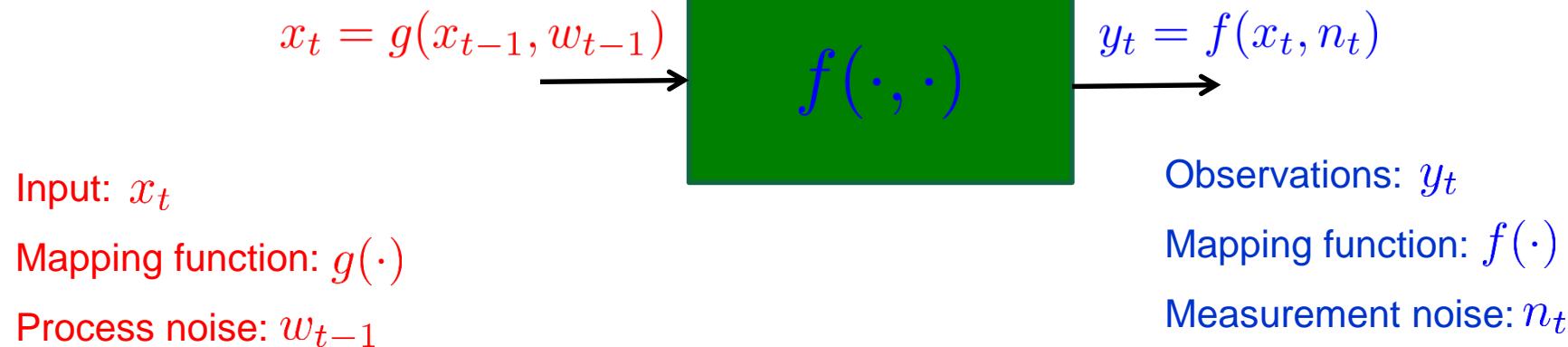
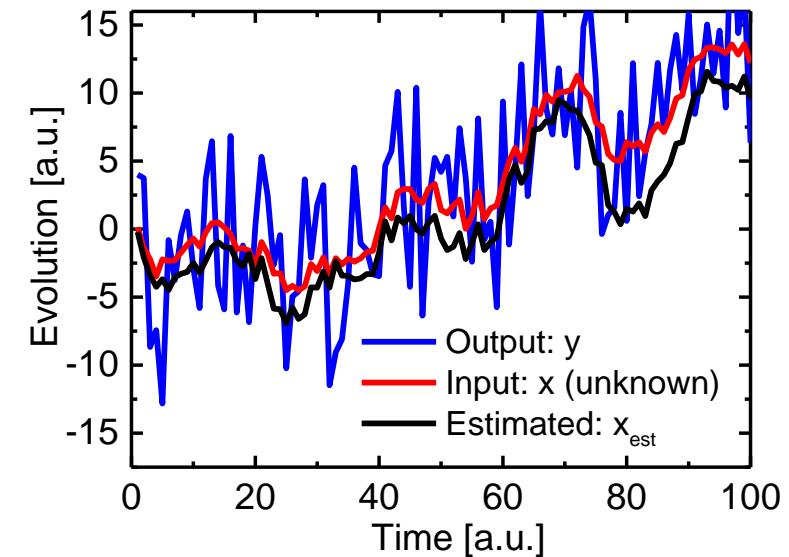
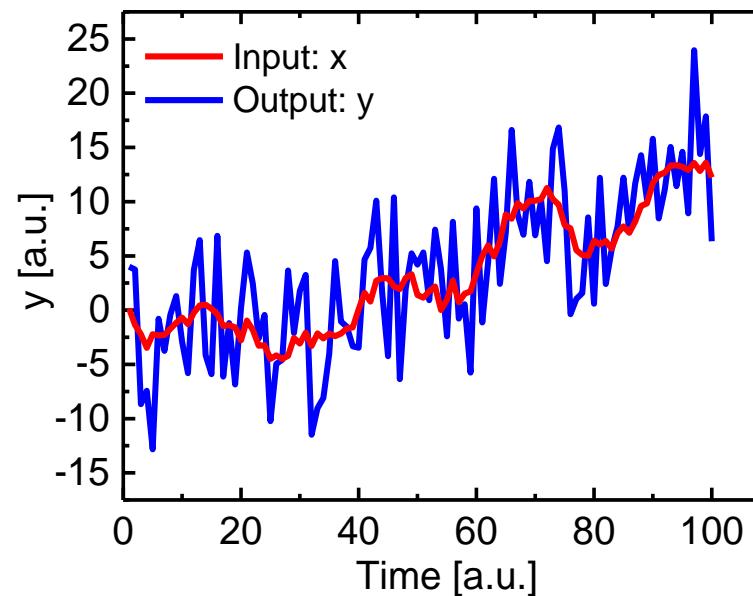
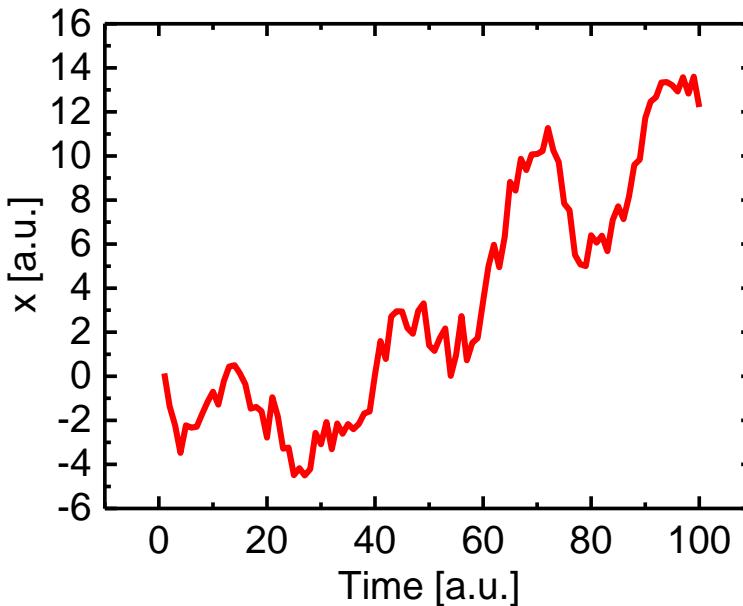


Model for predicting PUE





Recovering signals from noisy measurements



Machine learning in communication engineering

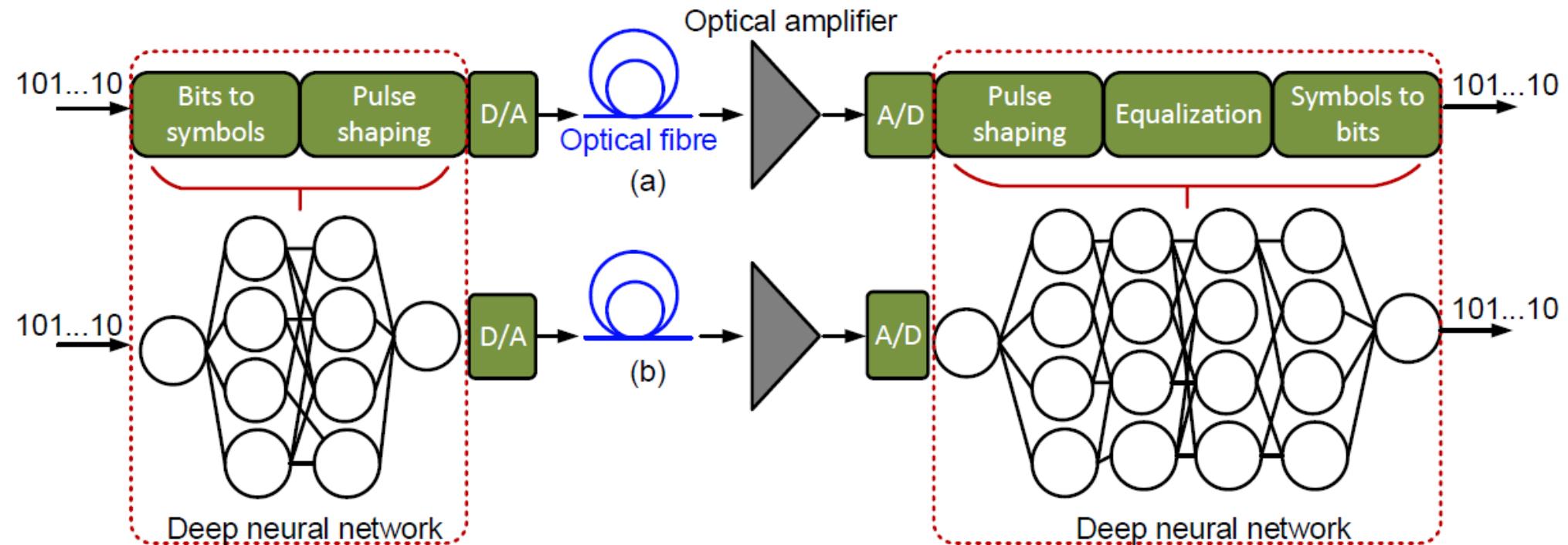
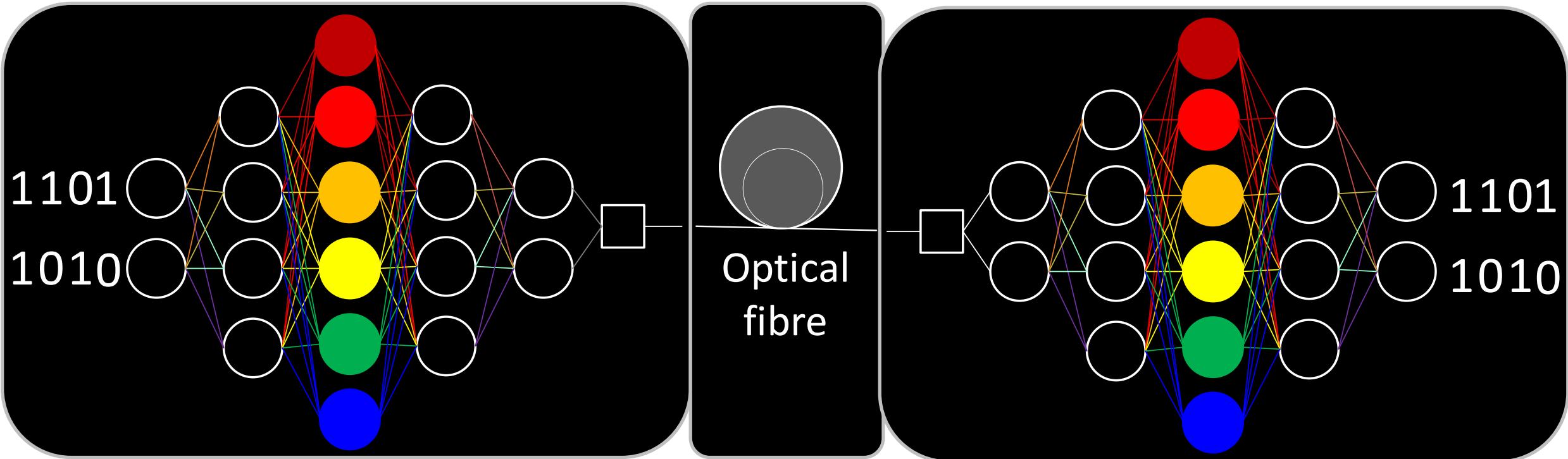


Figure 1: Fiber-optic communication systems (a) Conventional (b) **Machine learning-enabled**. D/A: digital-to-analogue-converter, A/D: analogue-to-digital converter

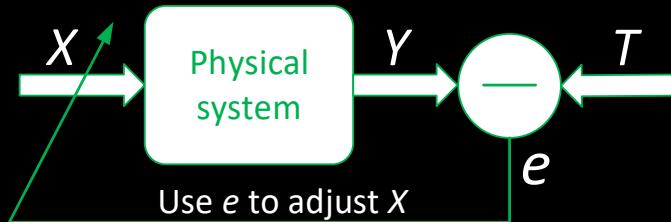
Learning to transmit and receive data over complex channels



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

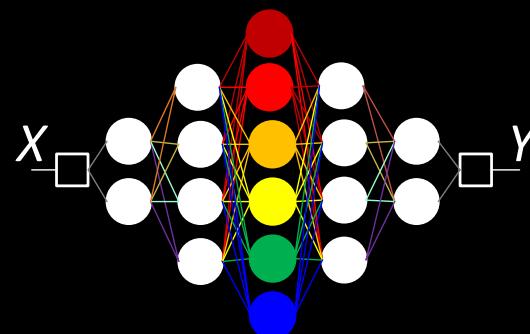
Inverse system learning

#1 Problem statement:

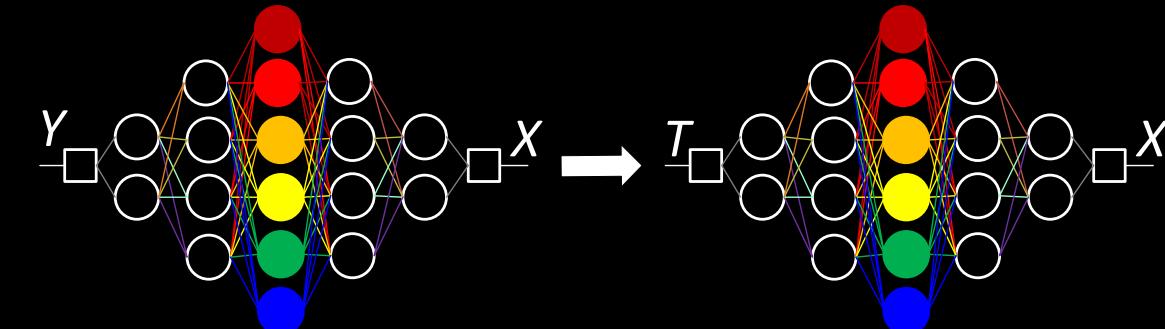


A physical system describing relation between input X and output Y is given. The objective is to determine input X that would result in a targeted output T .

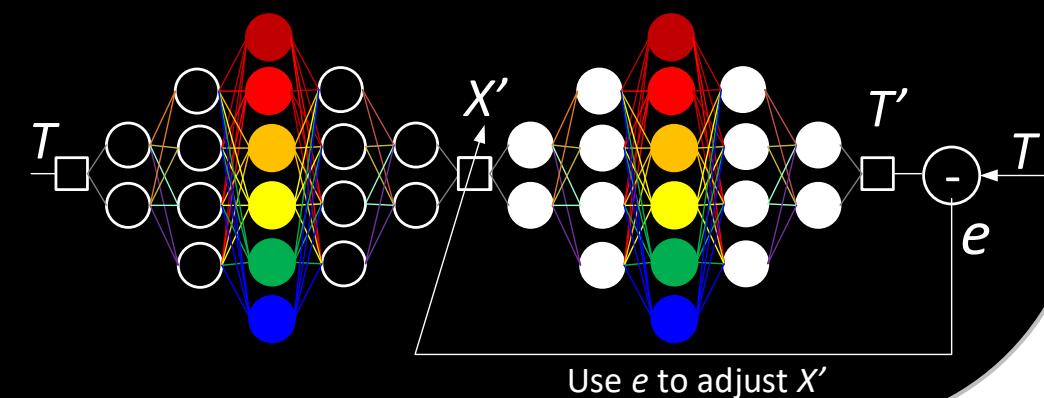
#3 Train neural network to learn *forward* mapping (from X to Y):



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



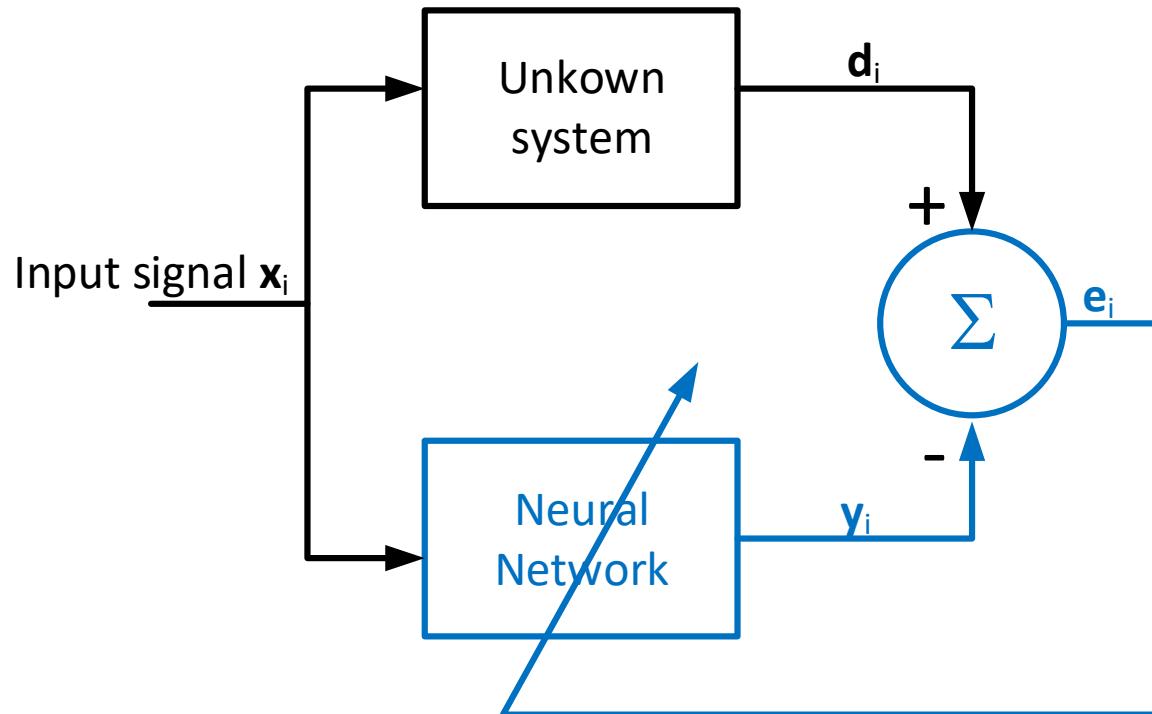
#4 Perform *final* optimization:



Use e to adjust X'

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

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

When should we use ML?

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

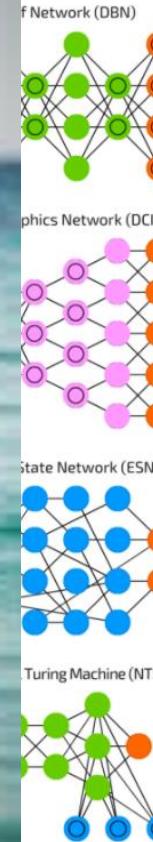
What is machine learning today?



TensorFlow



A deep learning library



Let's go back to the beginning...

“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)

Example:

You want to apply machine learning to improve your spam filter

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

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



28 pixels



28 pixels

3	4	2	1	9	5	6	2	1	8
8	9	1	2	5	0	0	6	6	4
6	7	0	1	6	3	6	3	7	0
3	7	7	9	4	6	6	1	8	2
2	9	3	4	3	9	8	7	2	5
1	5	9	8	3	6	5	7	2	3
9	3	1	9	1	5	8	0	8	4
5	6	2	6	8	5	8	8	9	9
3	7	7	0	9	4	8	5	4	3
7	9	6	4	7	0	6	9	2	3

Input vectors (x)

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



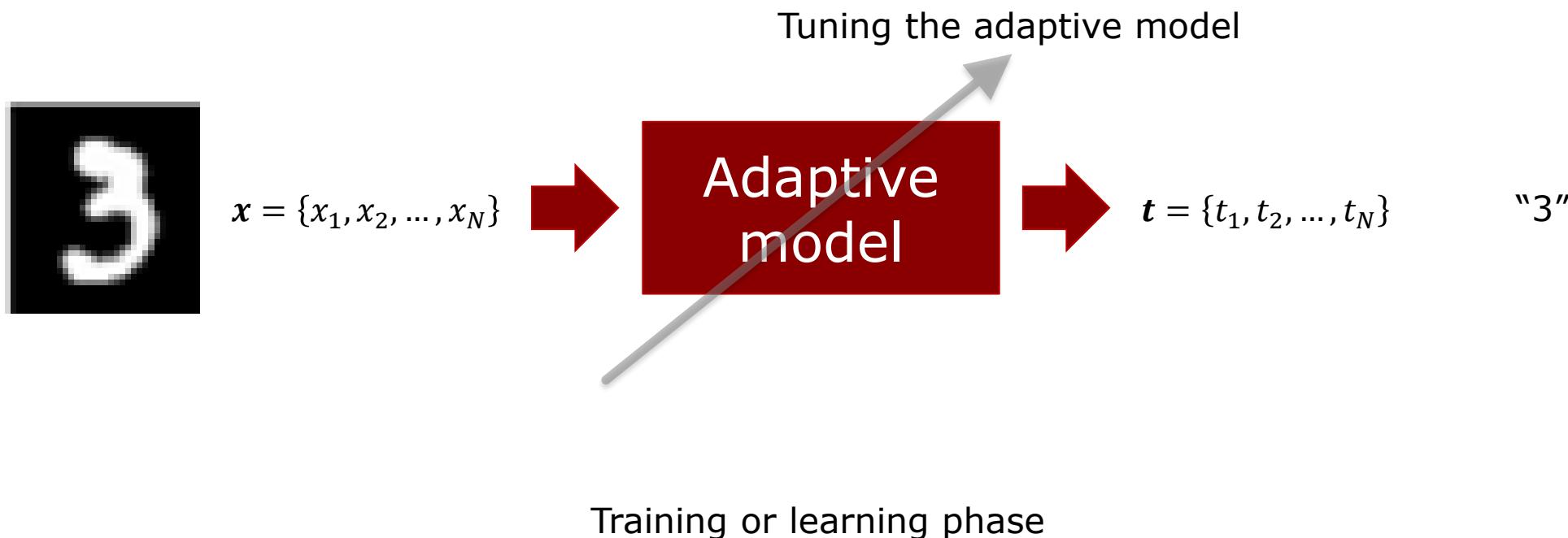
Associated
hand-labelling

Target vector (t)

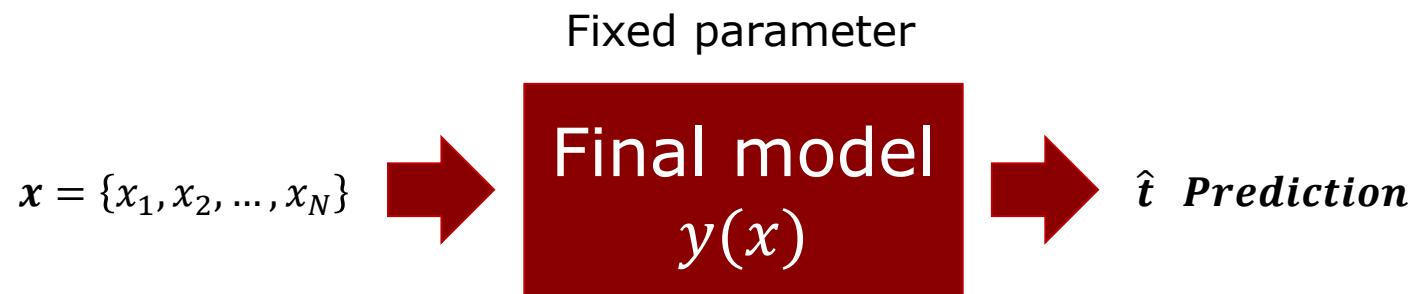
$$t = \{t_1, t_2, \dots, t_N\}$$

"3"

Training a model



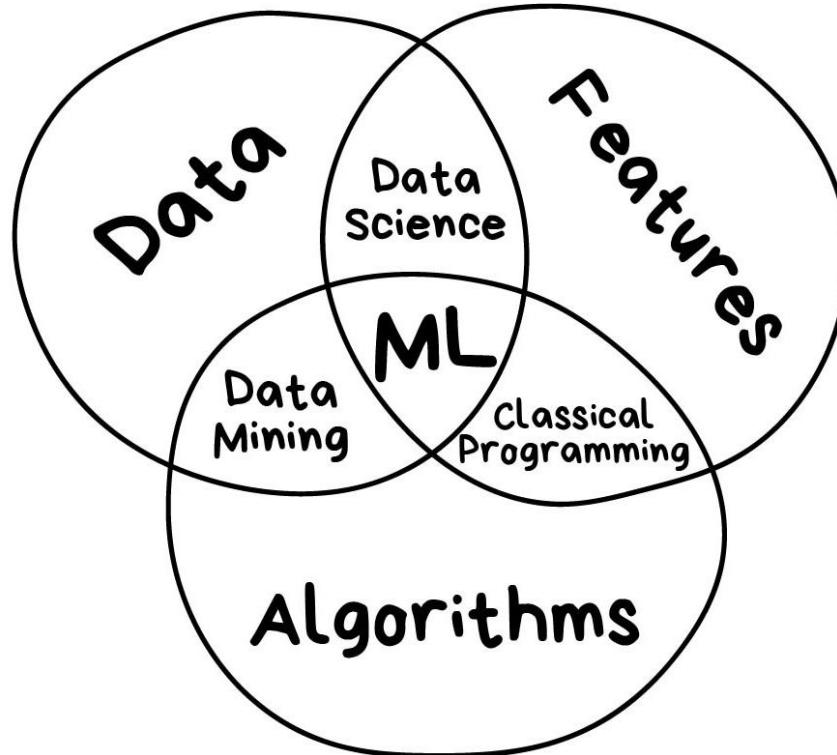
Testing a model



Testing phase: how well does the model generalize?

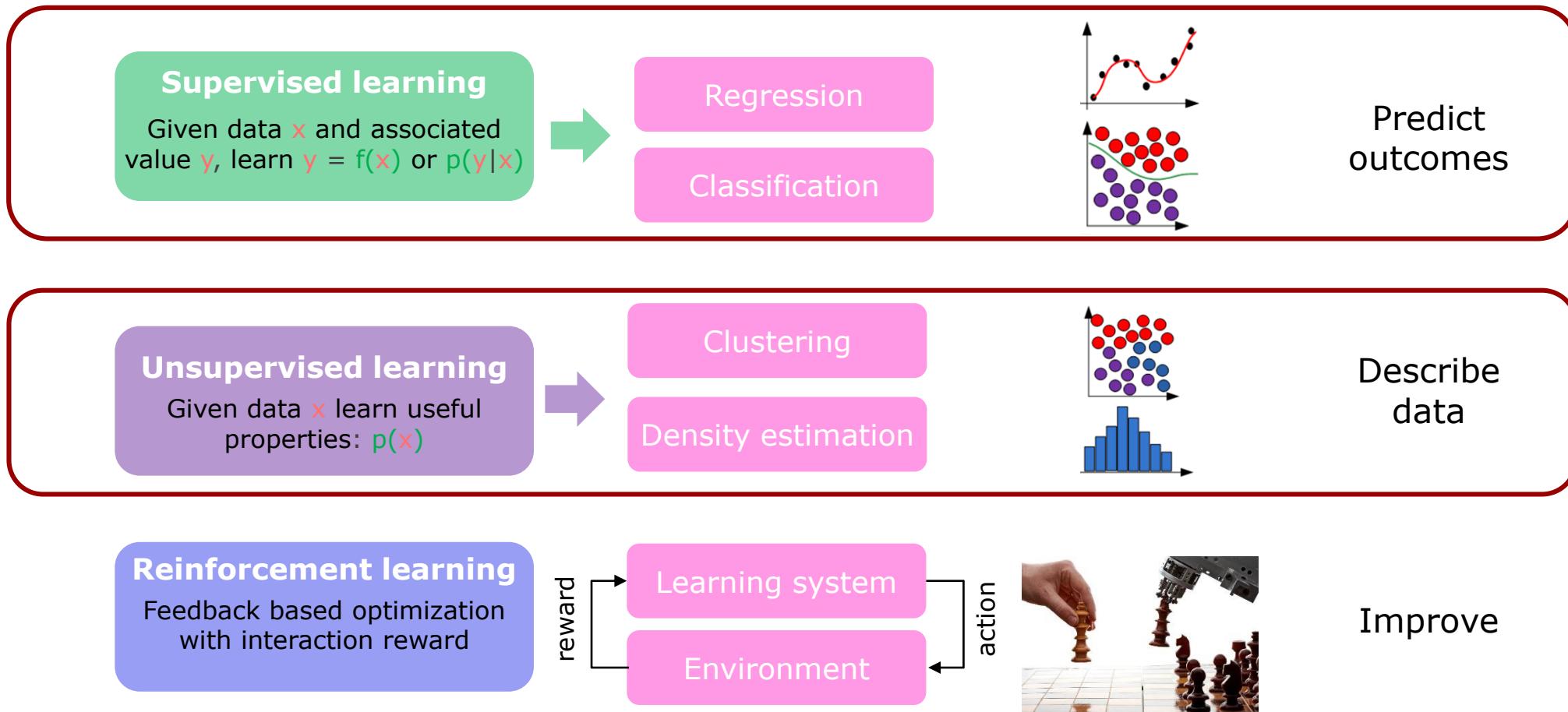
Generalization: the ability to correctly categorize a new example

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.

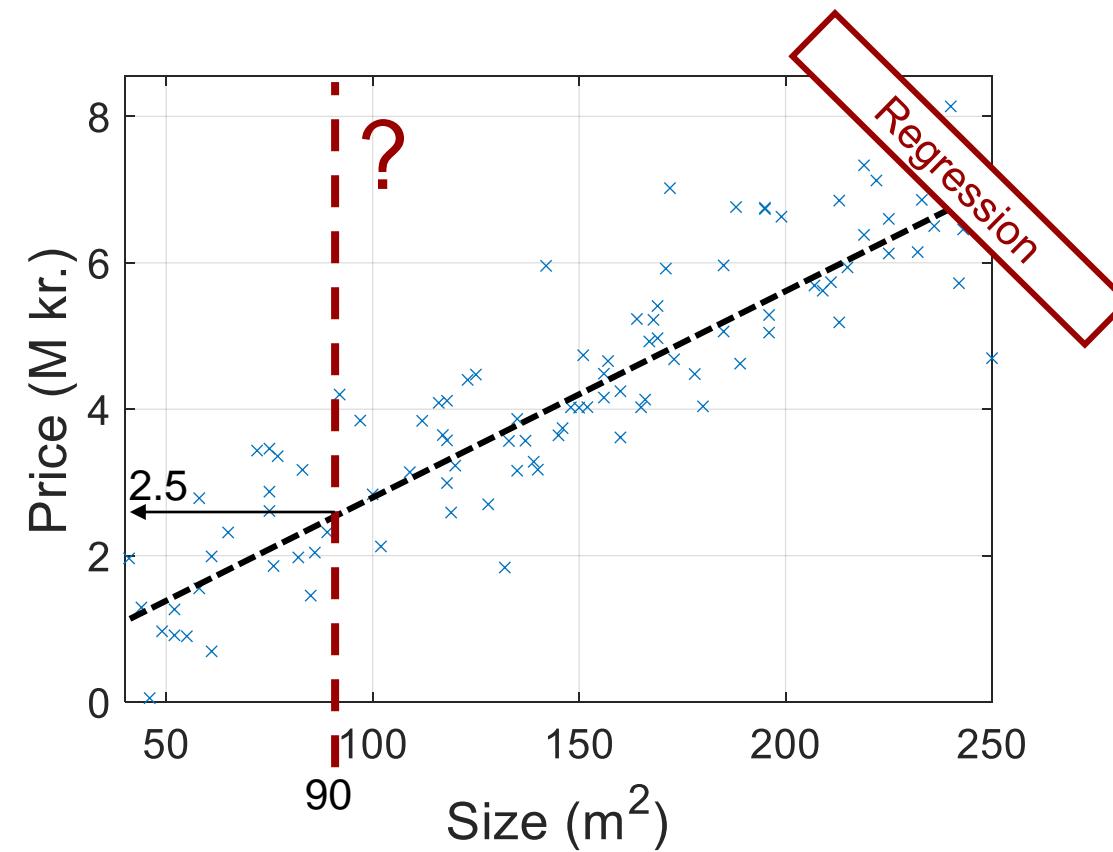
Machine learning algorithms



Supervised learning

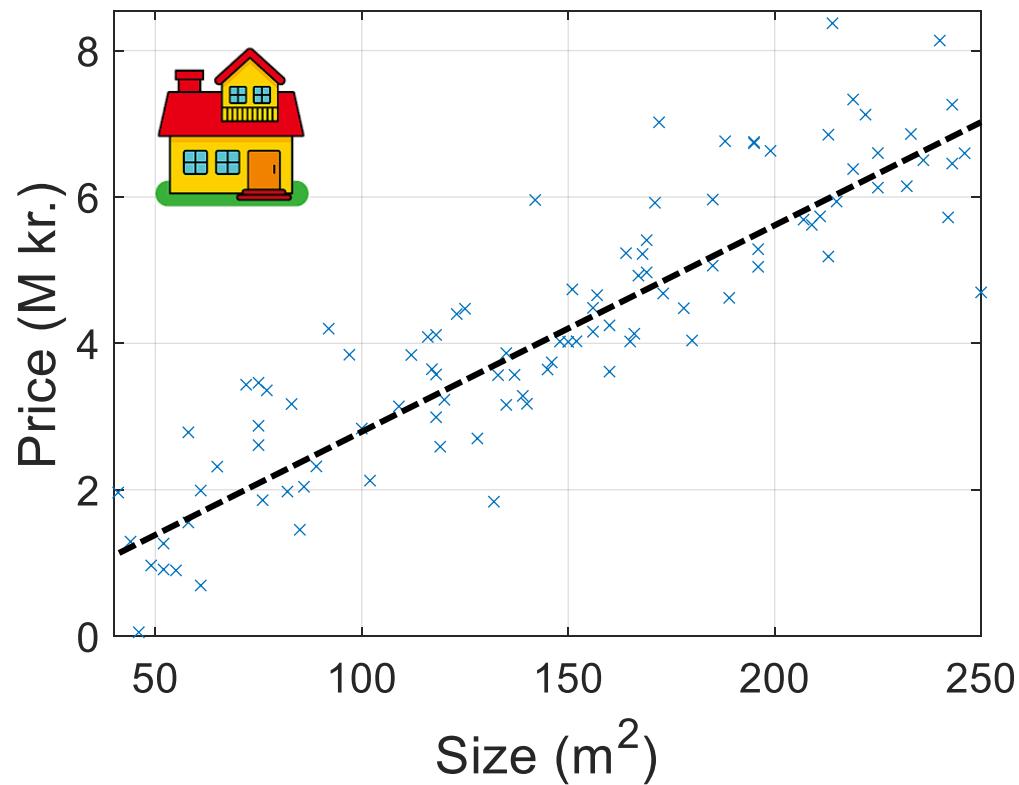
In supervised learning, the “right answers” (e.g. “labels” for training) are known.

Example: house pricing / square meter

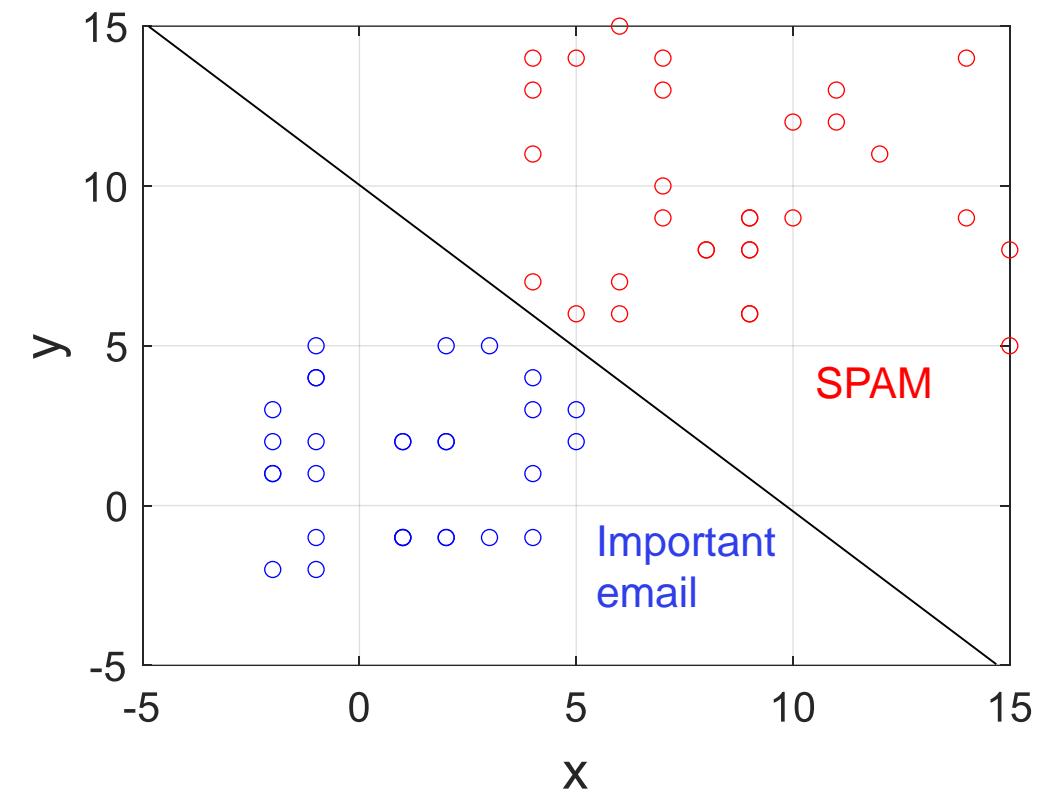


Regression vs. classification

House pricing prediction



Spam filter



Regression vs. classification – checking...

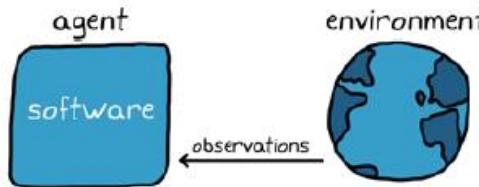
Regression problems predict **continuous** valued outputs

Classification problems predict **discrete** valued outputs

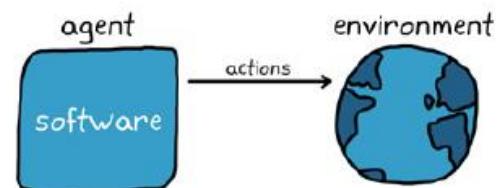
Reinforcement learning (RL)

1

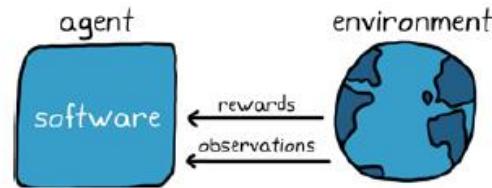
The agent is able to observe the current state of the environment.

**2**

From the observed state, it decides which action to take.

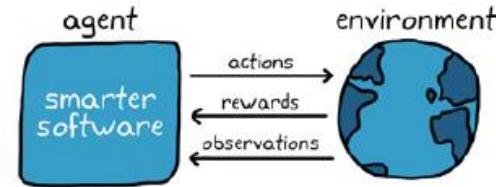
**3**

The environment changes state and produces a reward for that action. Both of which are received by the agent.

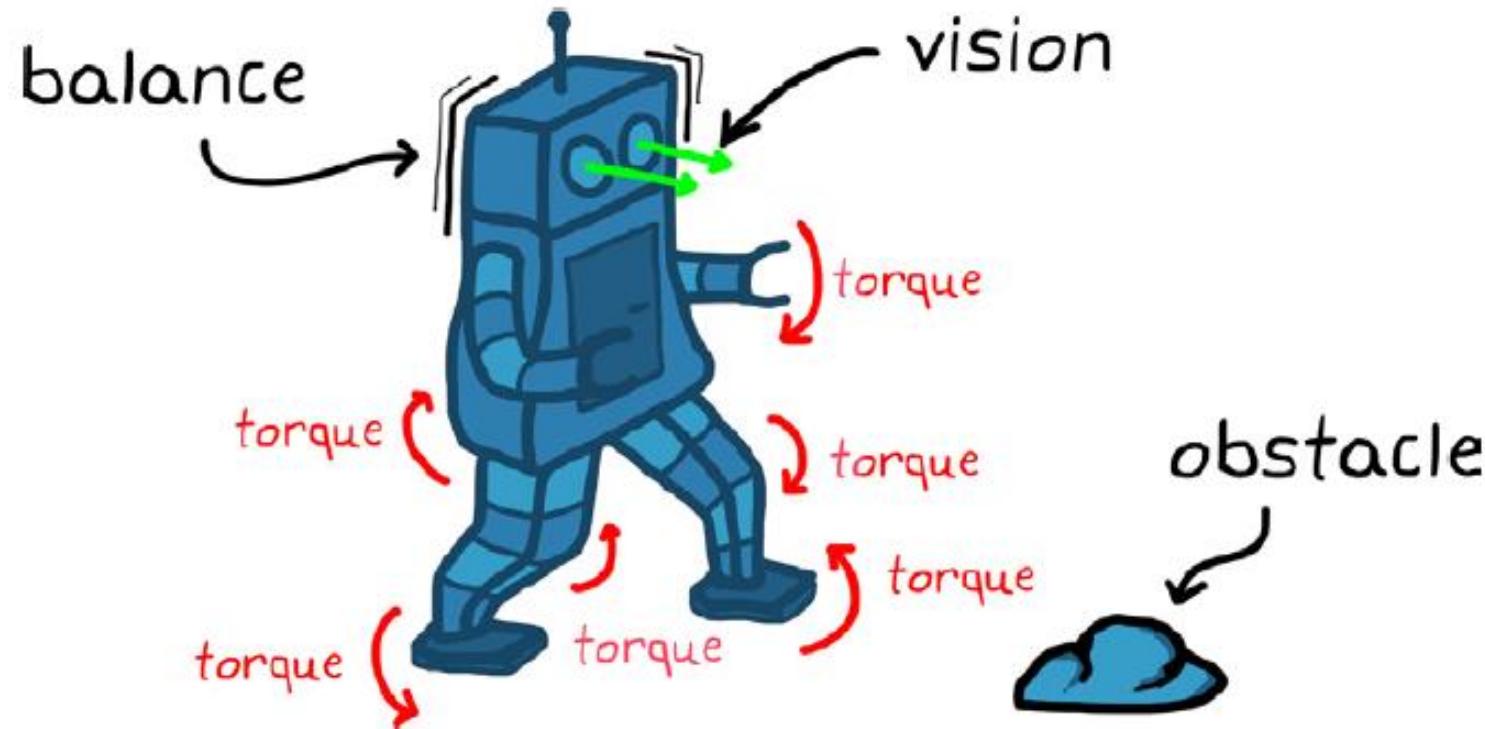
**4**

Using this new information, the agent can determine whether that action was good and should be repeated, or if it was bad and should be avoided.

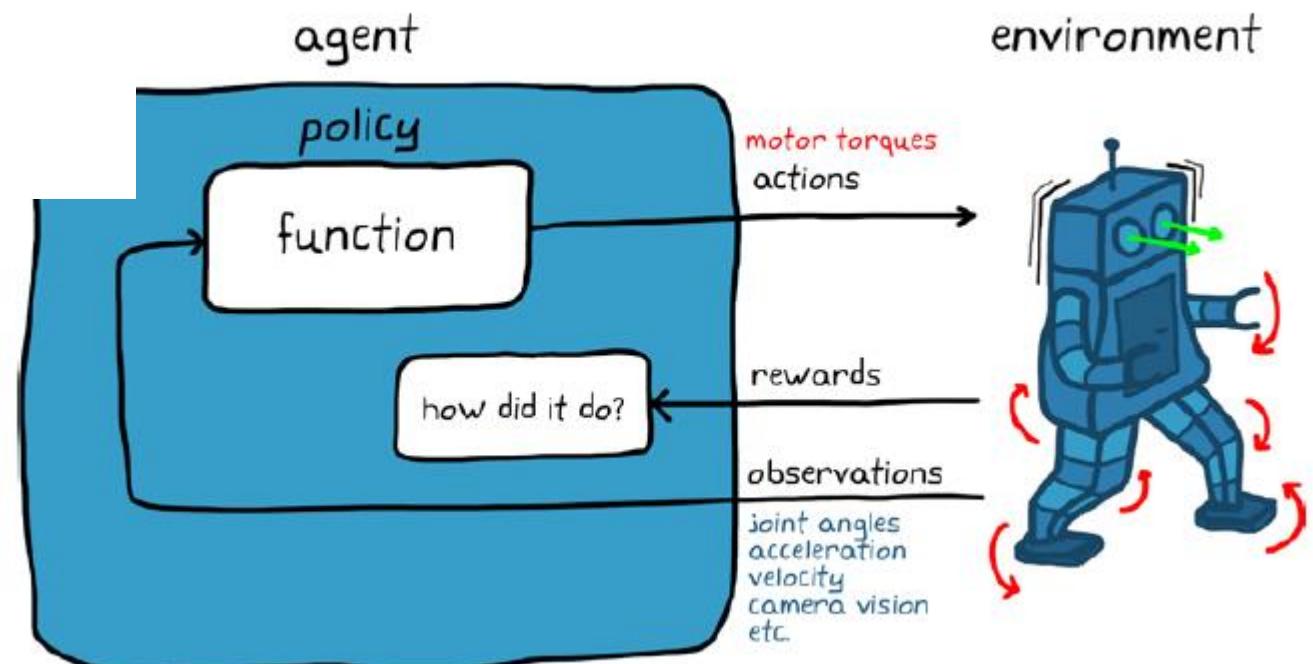
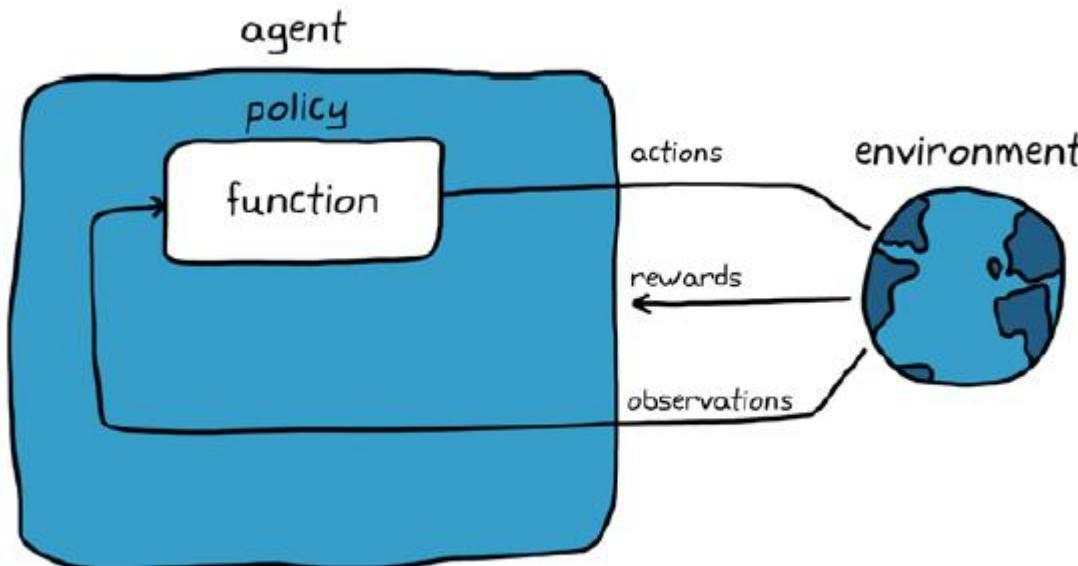
The observation-action-reward cycle continues until learning is complete.



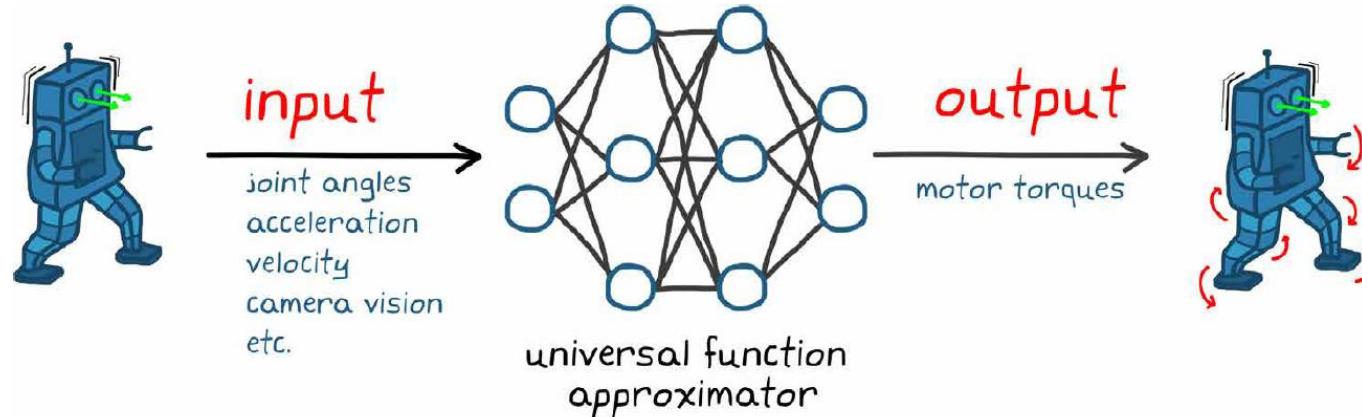
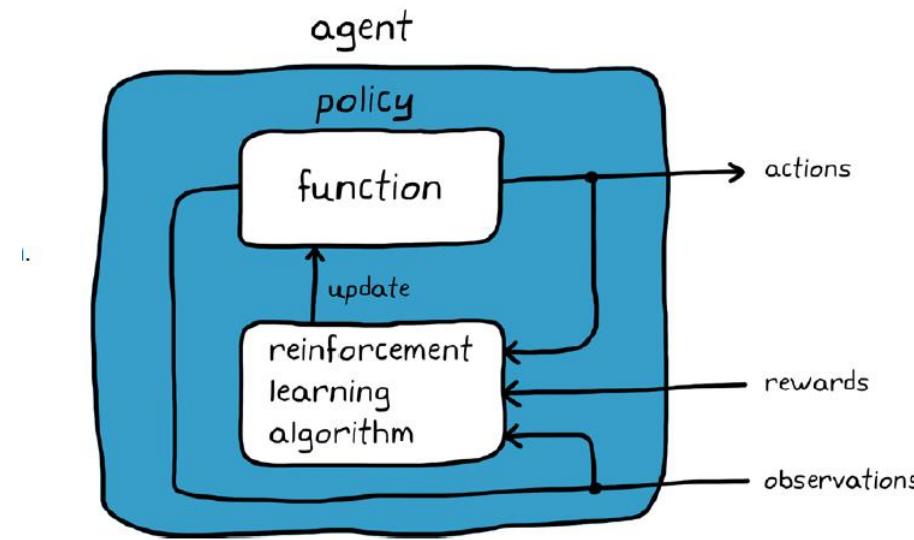
RL for controlling robot motion



RL for controlling robot motion



Actions represented by neural-network

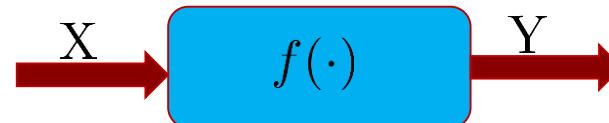


Time for a break....

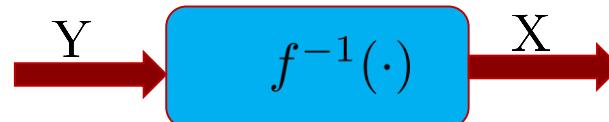


Where does machine learning excel?

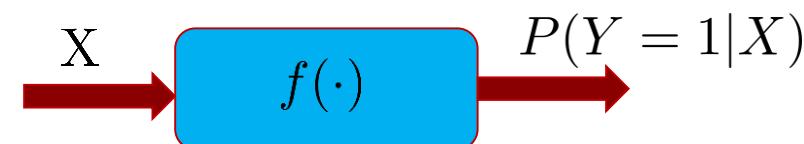
- Learning complex **direct** mappings:



- Learning complex **inverse** mappings:

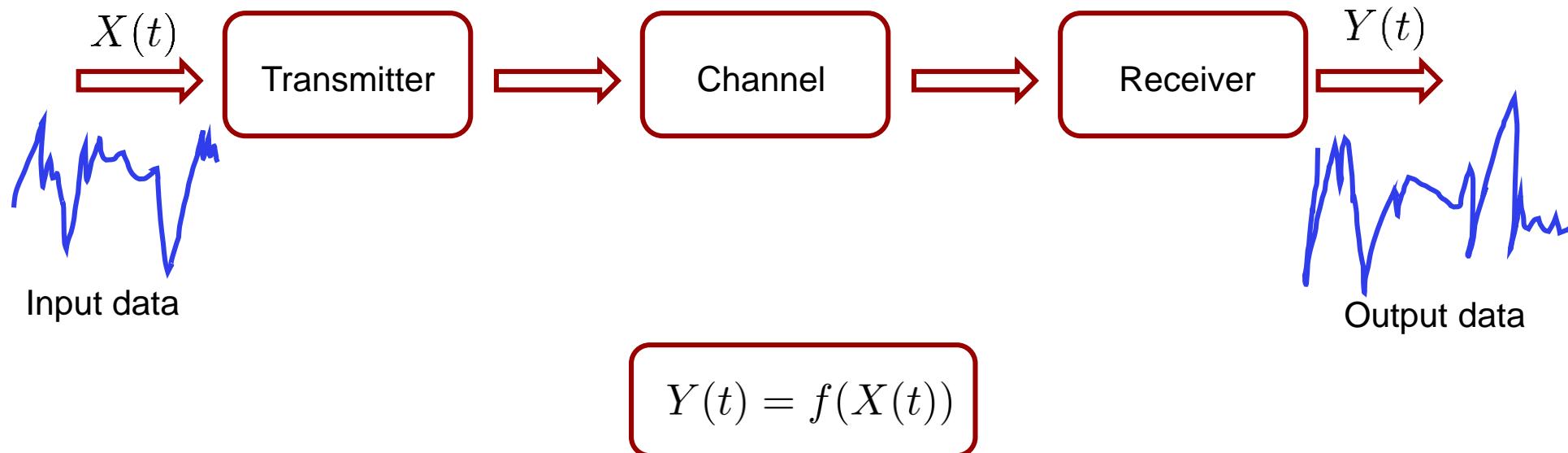


- Learning **decision rules** for complex mappings:



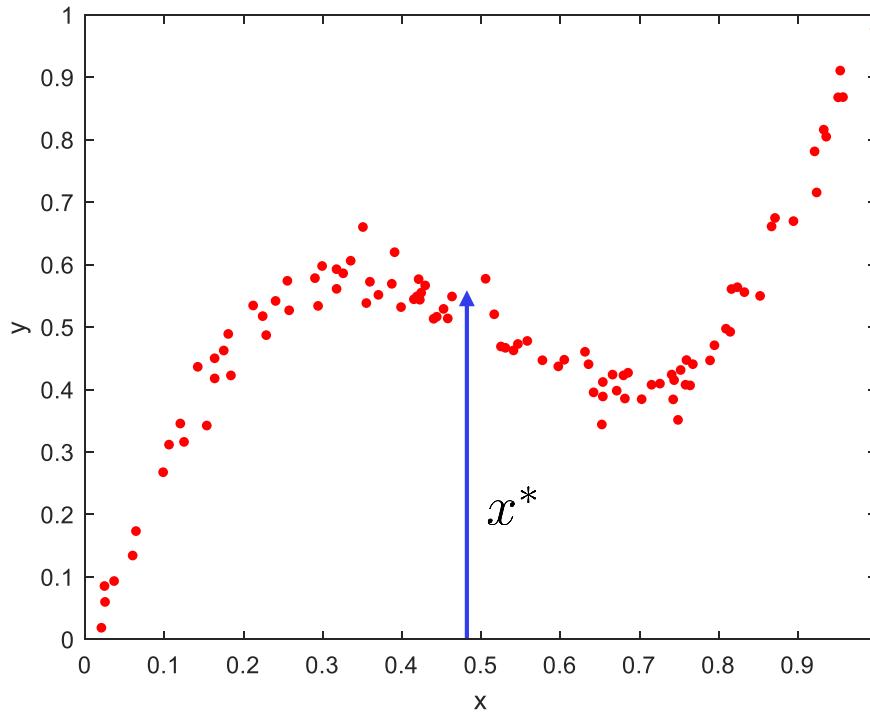
How do we learn $f(\cdot)$ and $f^{-1}(\cdot)$

Learning the communication system model



Objective: given the recordings of $X(t)$ and $Y(t)$, learn the mapping

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

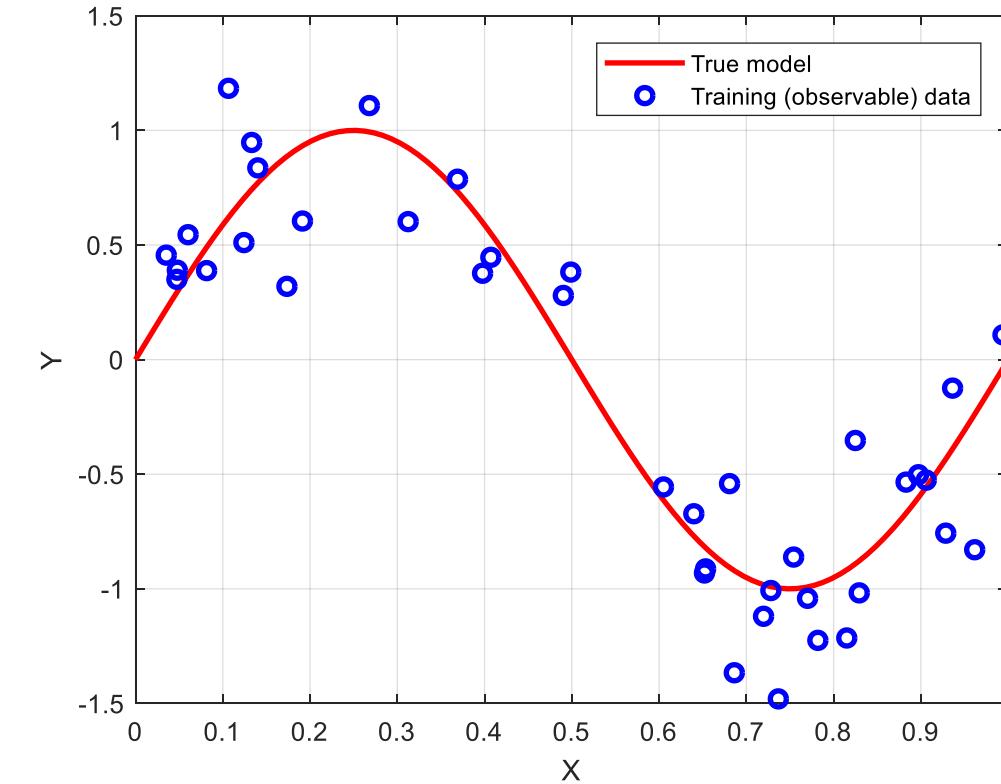
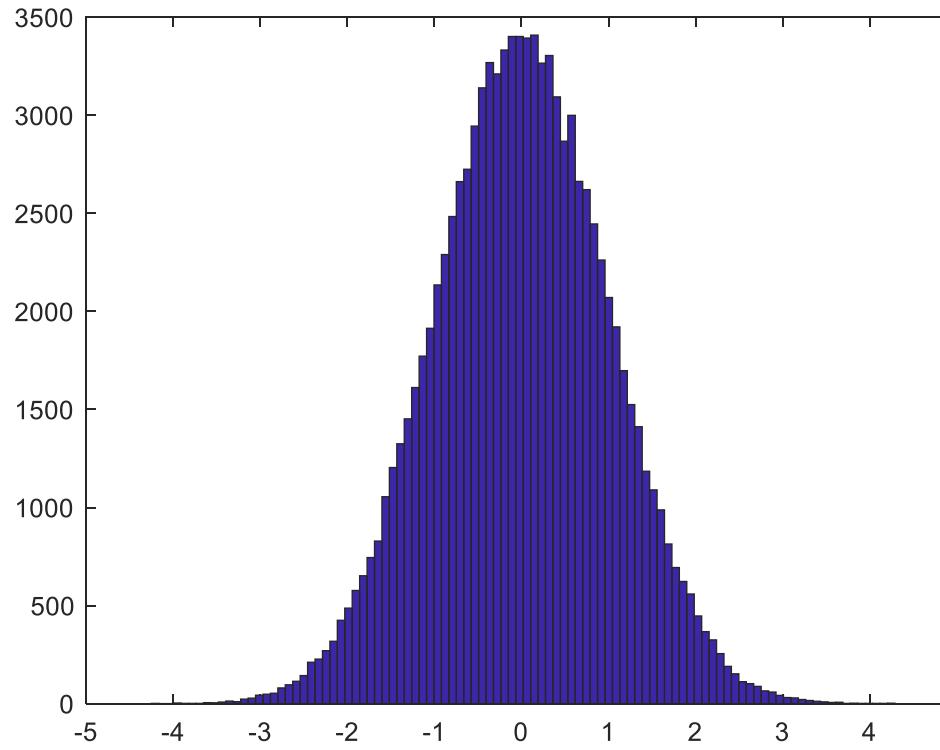
Use *machine learning* to build a model that represents the data set

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 that relates input to the output :

Or

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

Or

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

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

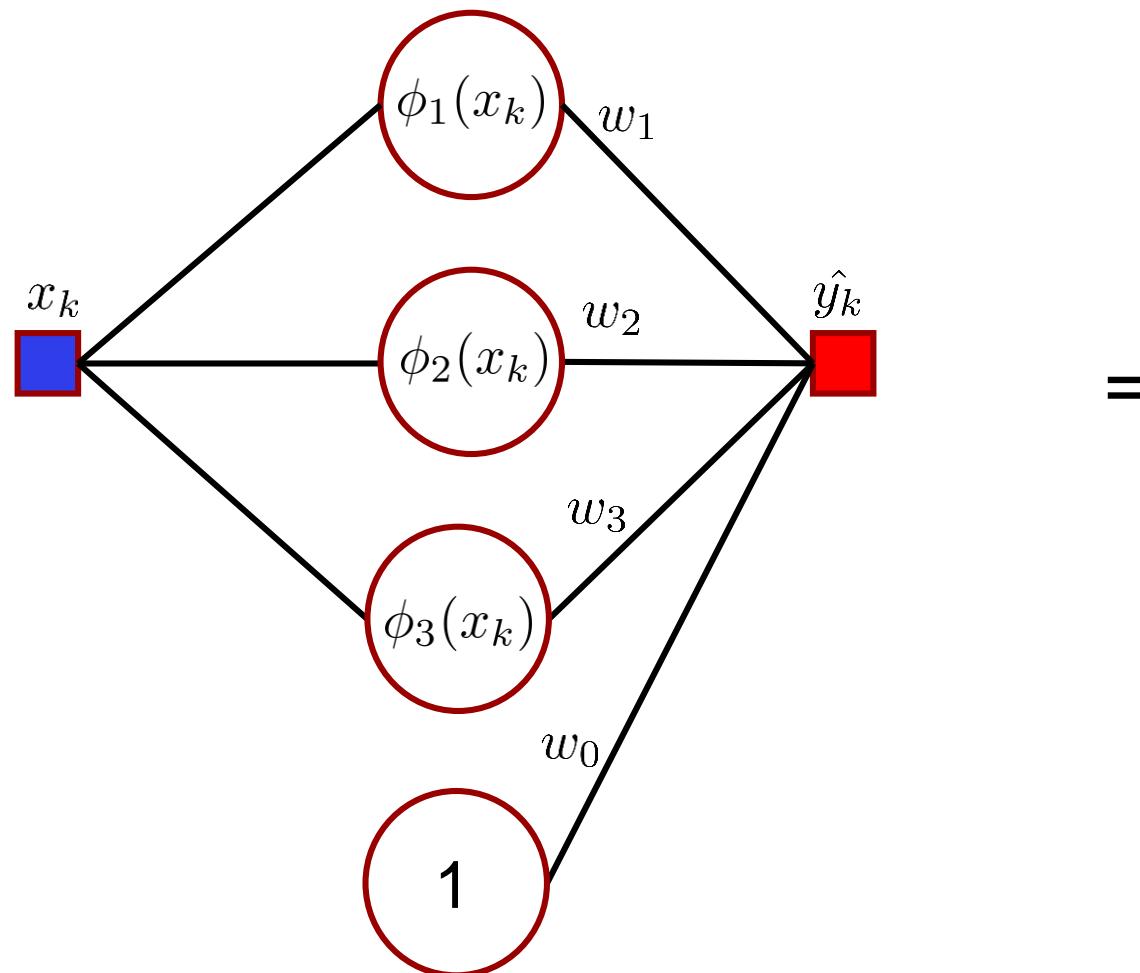
General linear model:

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

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

Procedure: choose an algorithm that determine the weights

Topology of the linear model

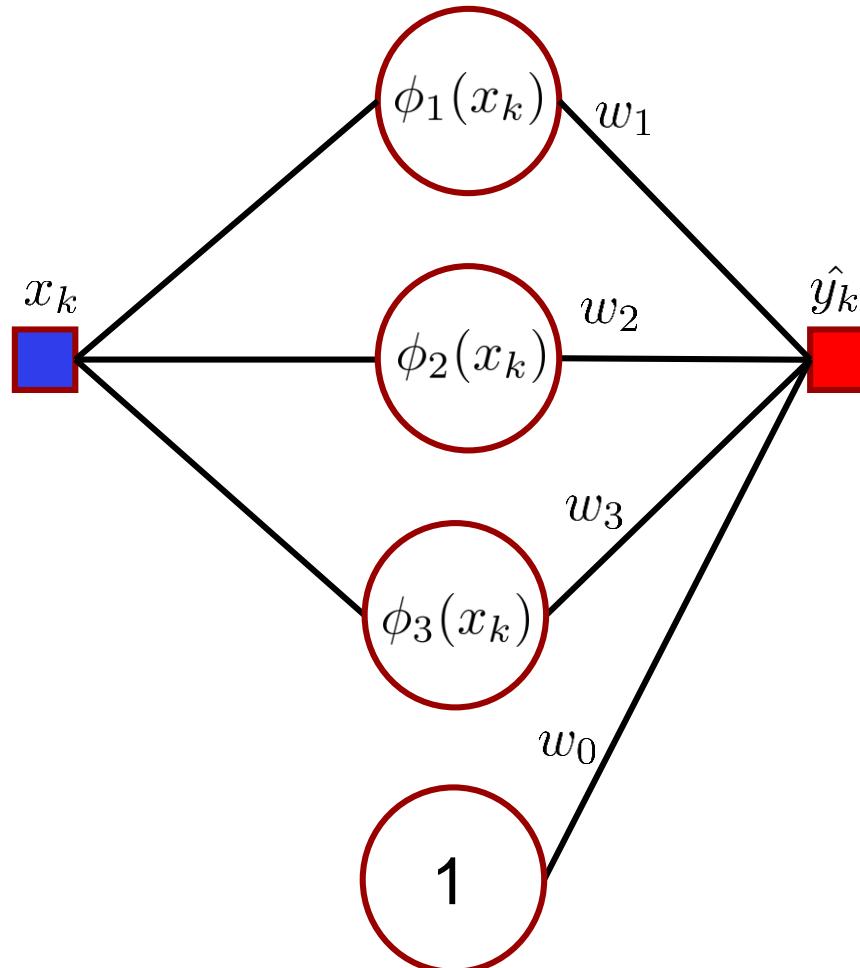


=

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

Determining the model parameters

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:

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

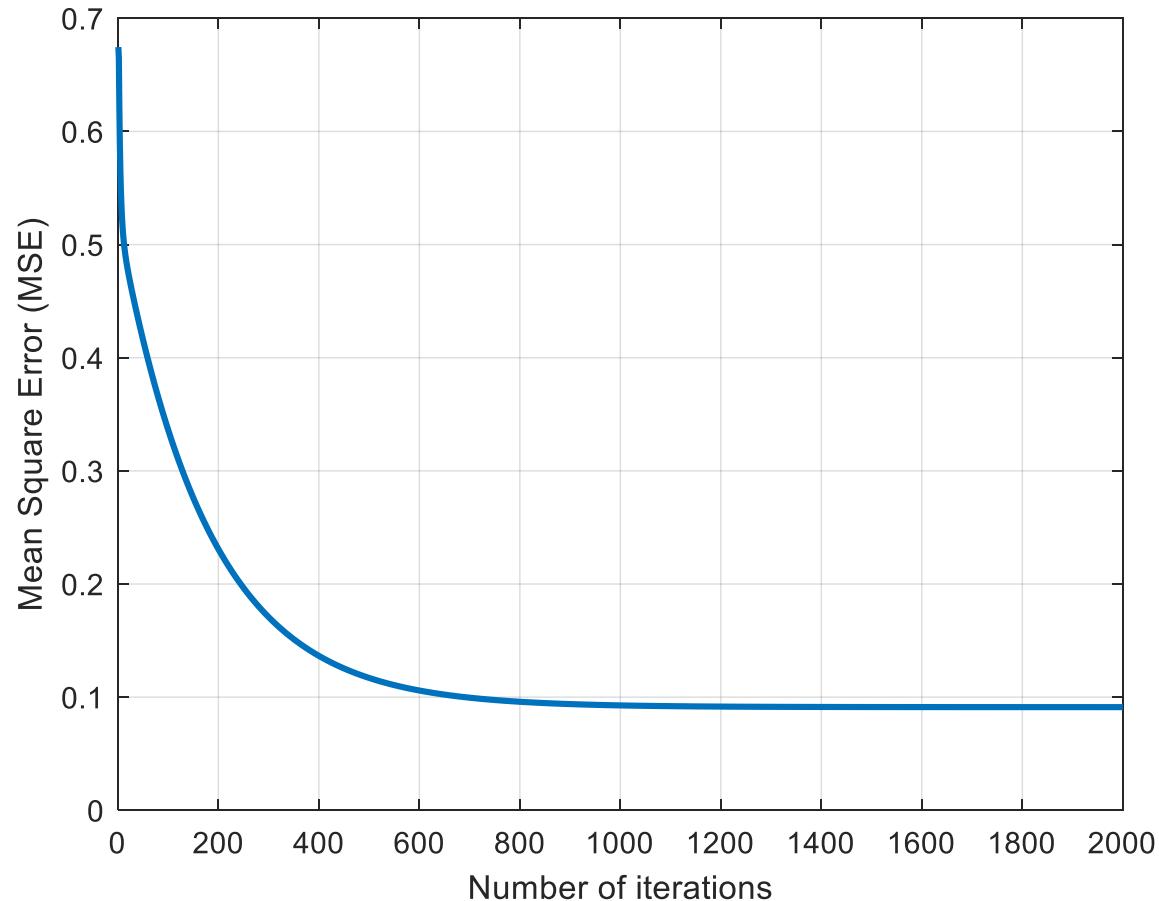
Computing the gradient: $\frac{\partial e_k}{\partial \mathbf{W}_k} = -[y_k - \mathbf{W}_k^T \Phi_k] \Phi_k$

```

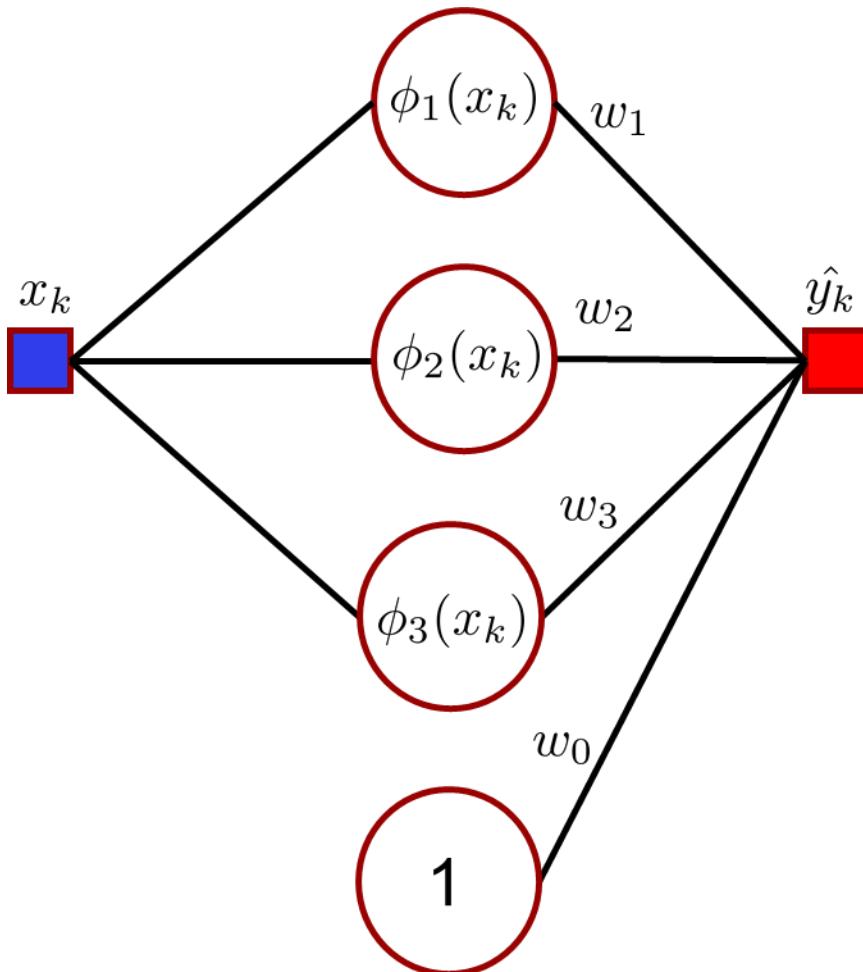
for i = 1 : L_iter
    for k = 1 : L_train
        compute e(k)
        grad = de(k) dw;
        w = w + eta*grad;
    end
    e(i) = mean(e.^2);
end
  
```

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

Performance evaluation on training data-set



Determining the weights in one step



The output is expressed as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & \phi_1(x_1) & \phi_2(x_1) & \phi_3(x_1) \\ 1 & \phi_1(x_2) & \phi_2(x_2) & \phi_3(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(x_N) & \phi_2(x_N) & \phi_3(x_N) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

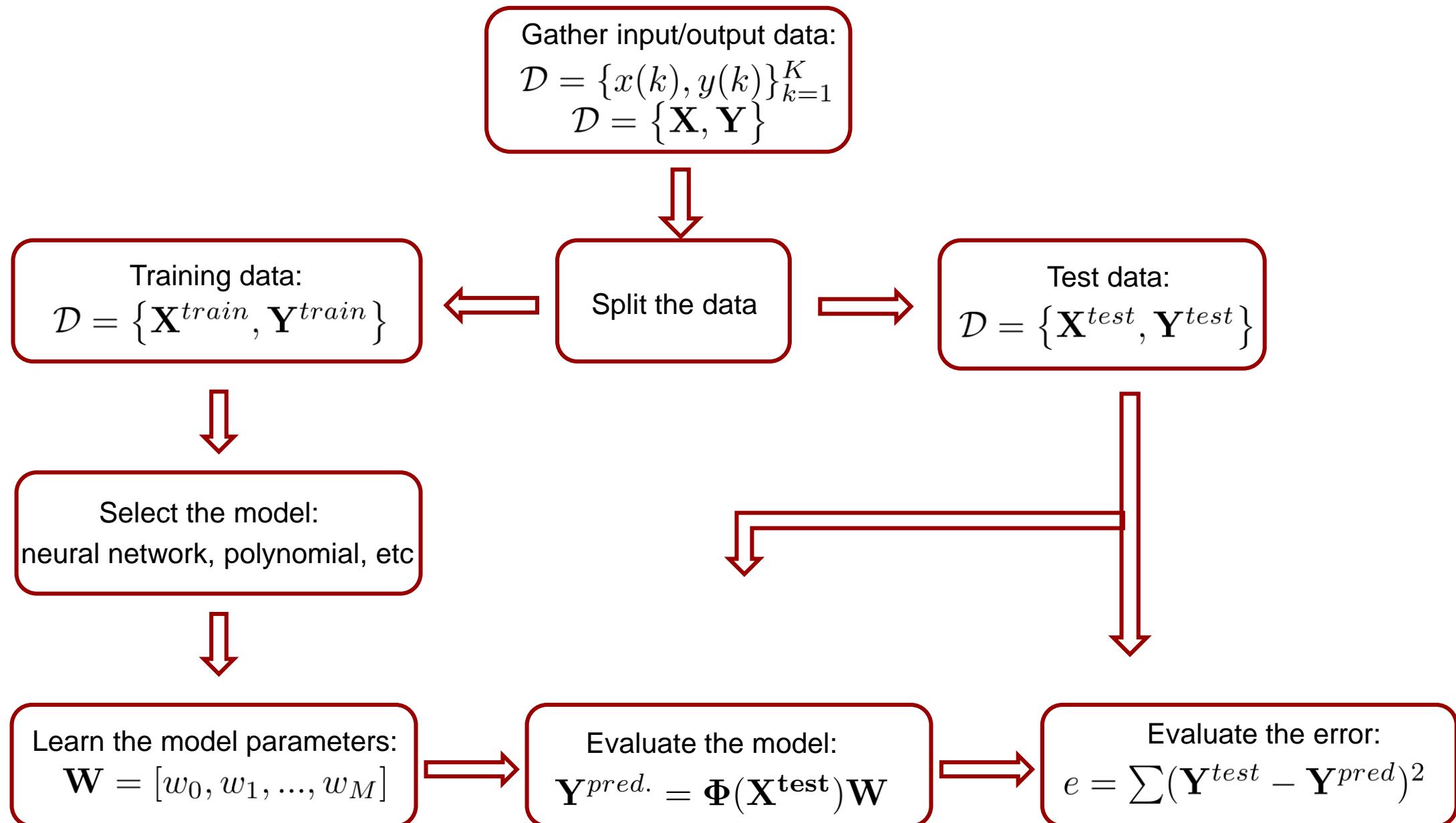
$$\mathbf{Y} = \Phi \mathbf{W}$$

The weights are computed as Moore-Penrose pseudo inverse:

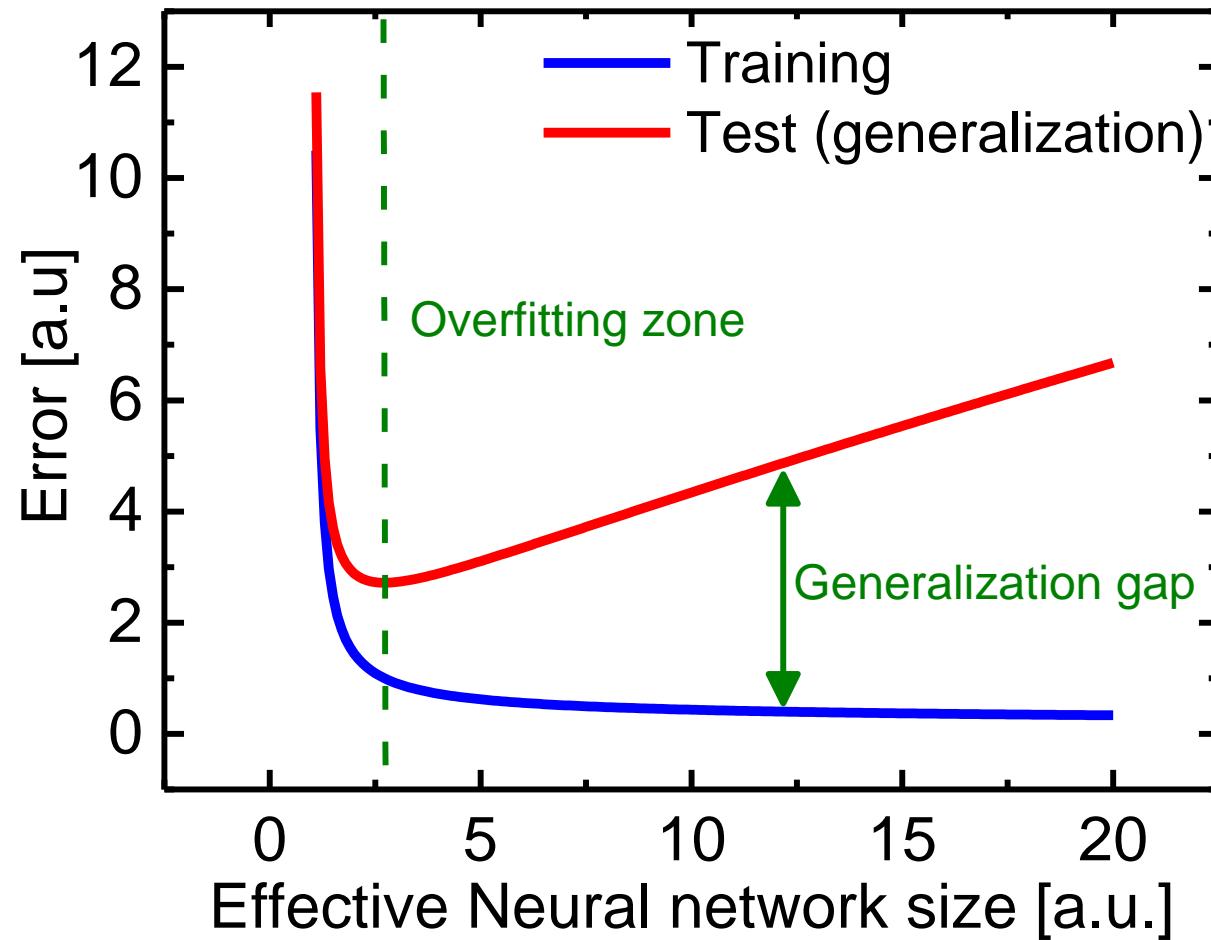
$$\mathbf{W} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{Y}$$

If Φ is square and invertible:

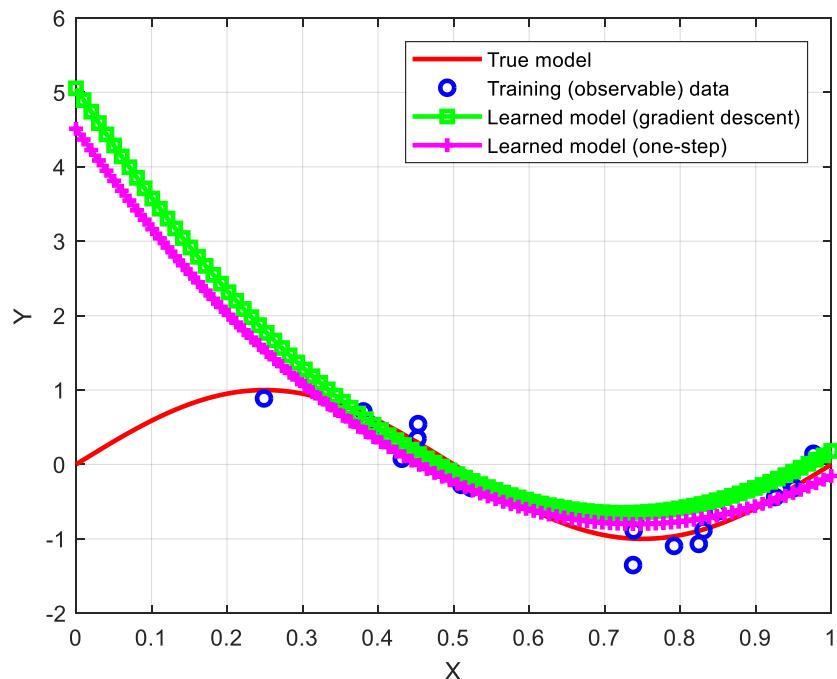
$$(\Phi^T \Phi)^{-1} \Phi^T = \Phi^{-1}$$



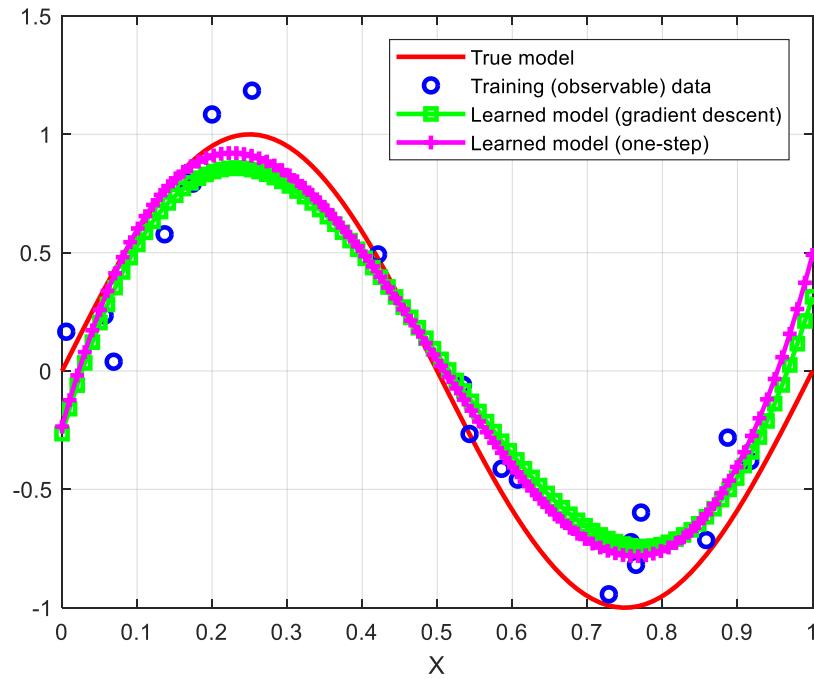
Key performance metric: generalization



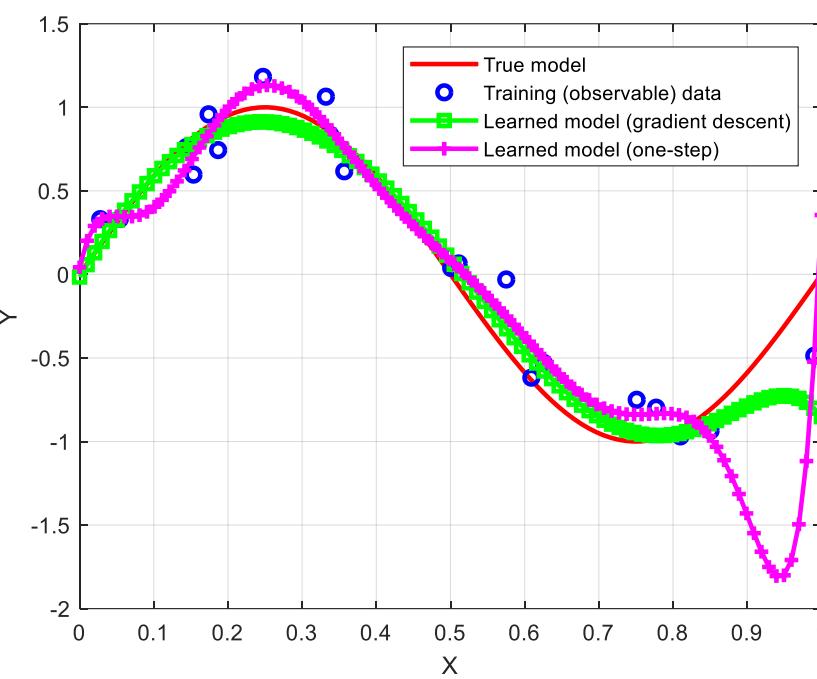
Evaluating the learned the model



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



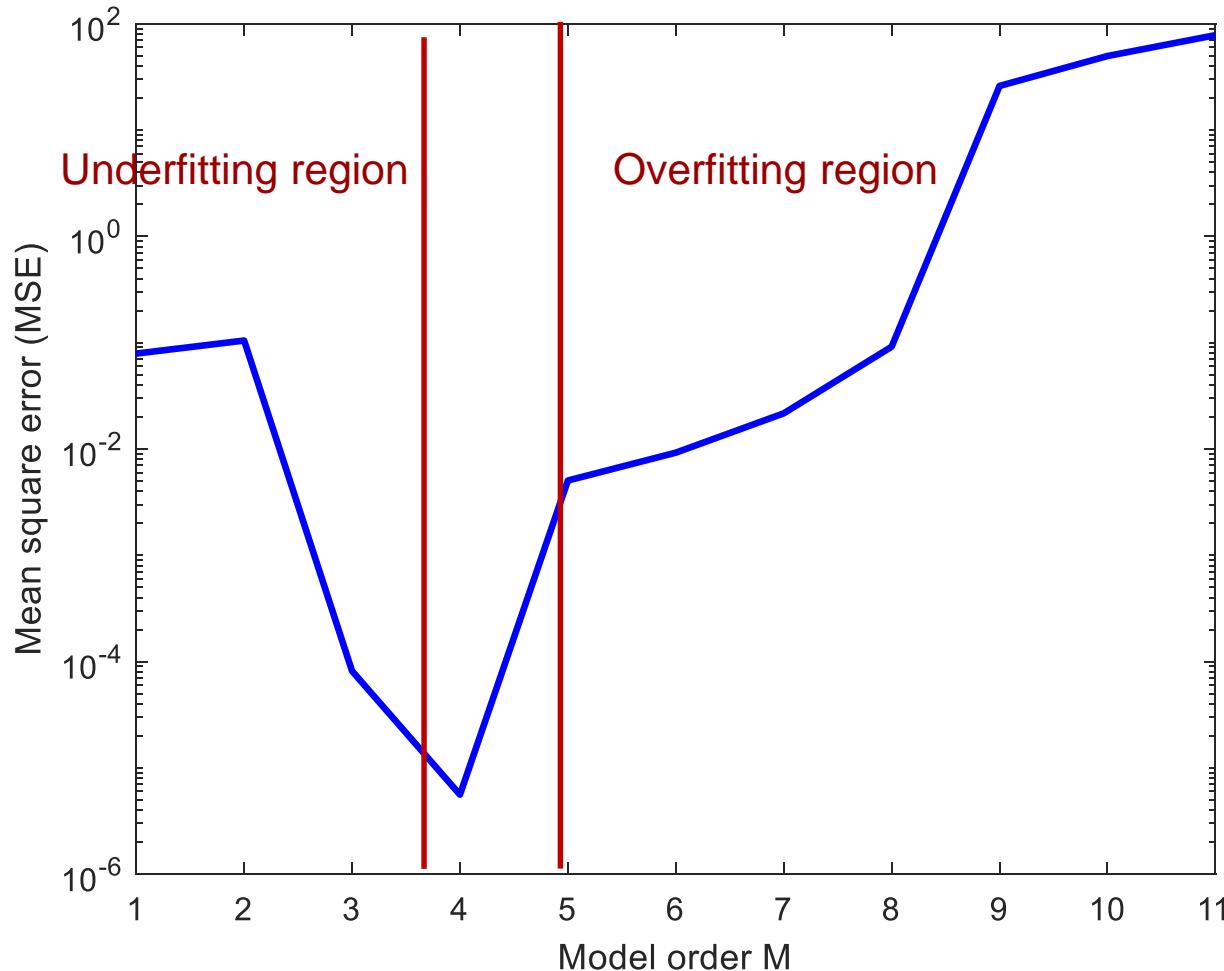
Assumed polynomial model order $M=3$
(the right model)



Assumed polynomial model order $M=9$
(too complex model: overfitting)

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

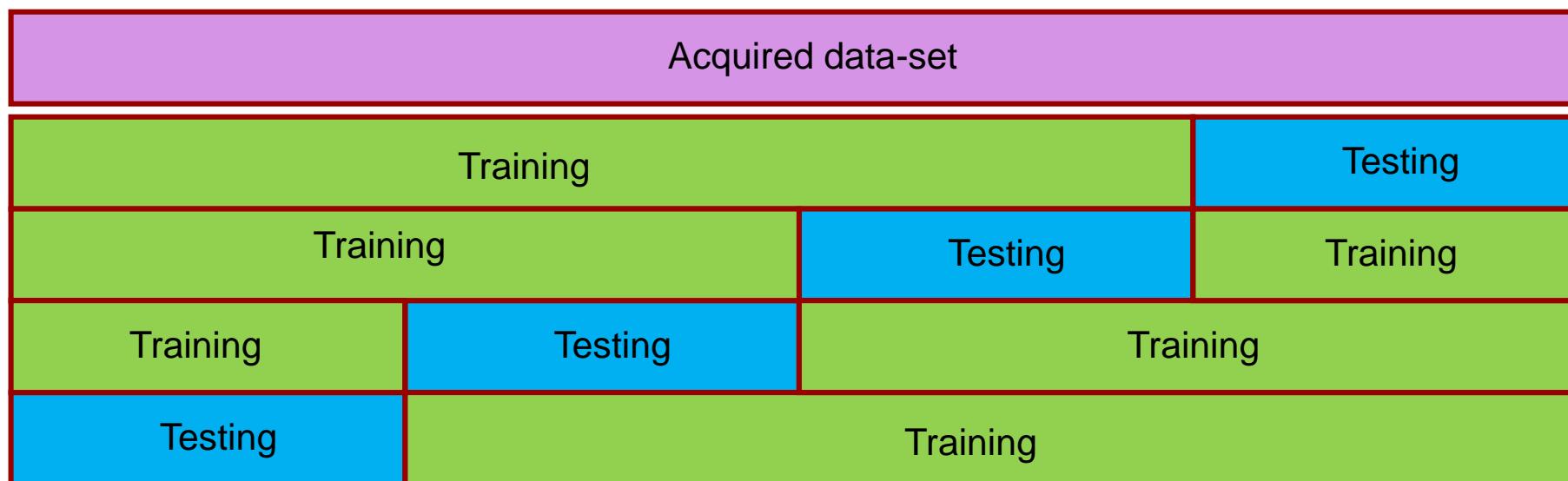
Evaluating different model orders



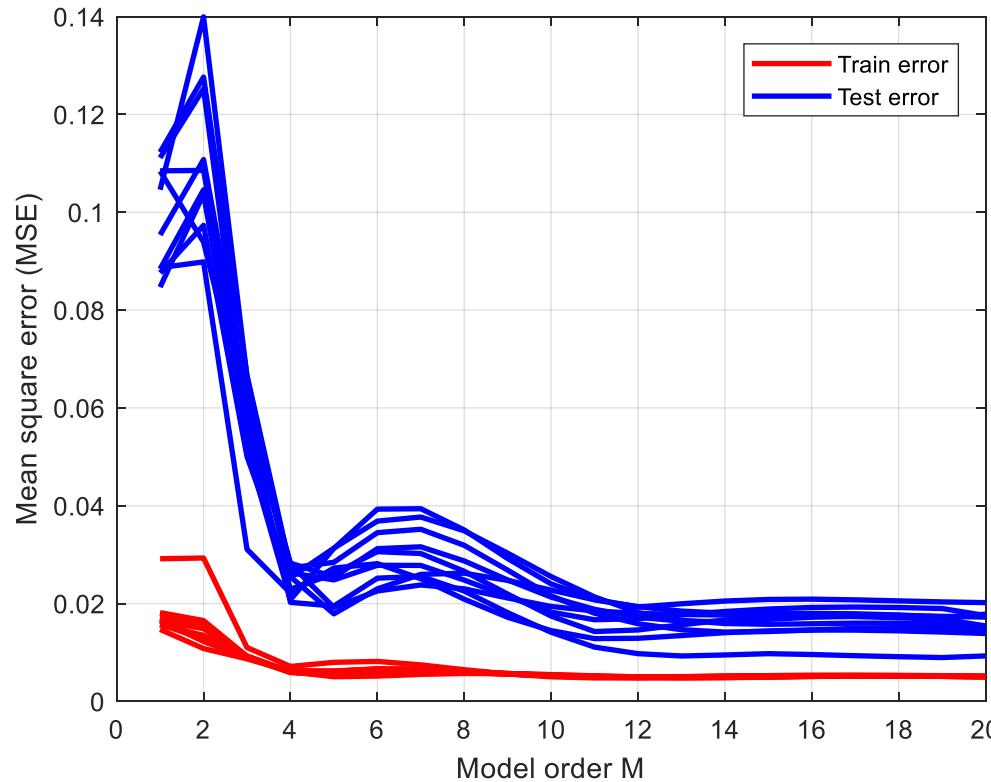
$$MSE = \frac{1}{2} \sum_{k=0}^N [y_k^{true} - \Phi(x_k)\mathbf{W}]^2$$

Cross-validation

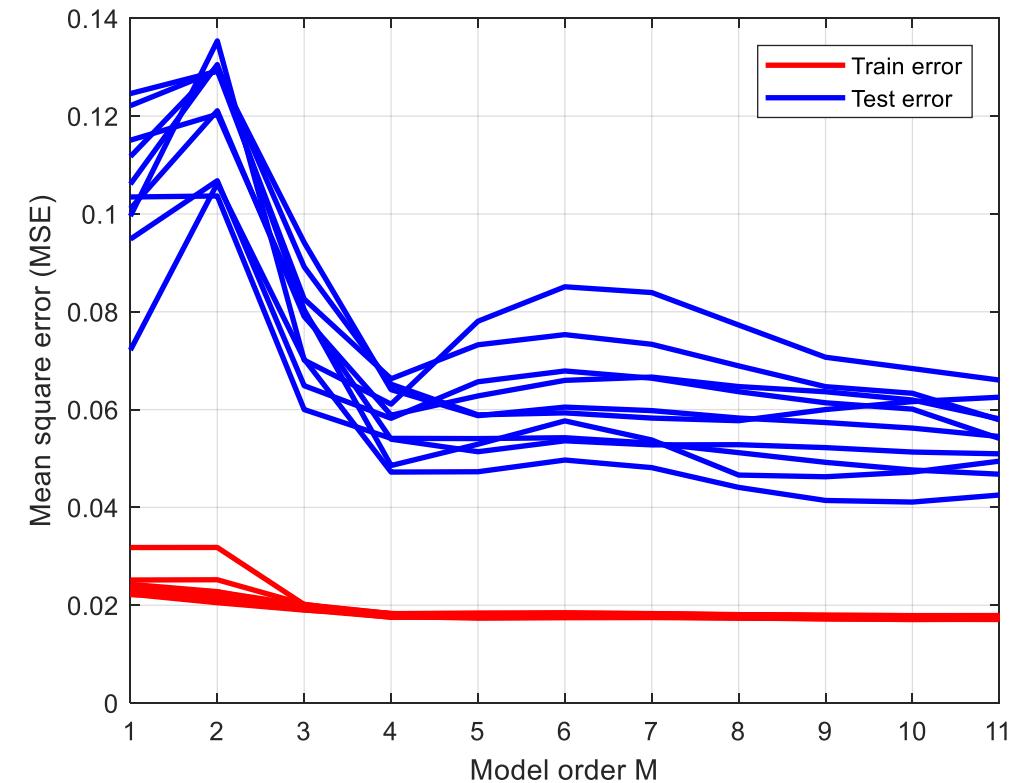
- Typically, the true mode is unknown
- We only have access to noisy observable data (data set)
- Observable data needs to be used for training and test
- Split the observable data into folds for training and test
- Perform weights estimation (learning) and testing for each fold
- Compute average test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



10-fold cross validation



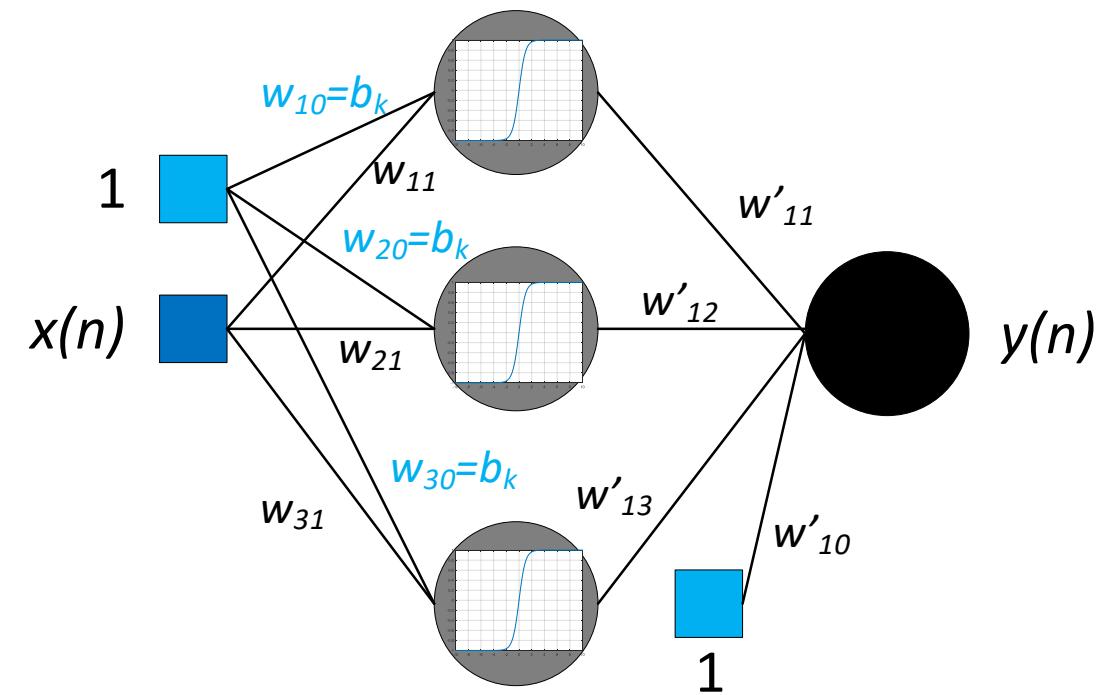
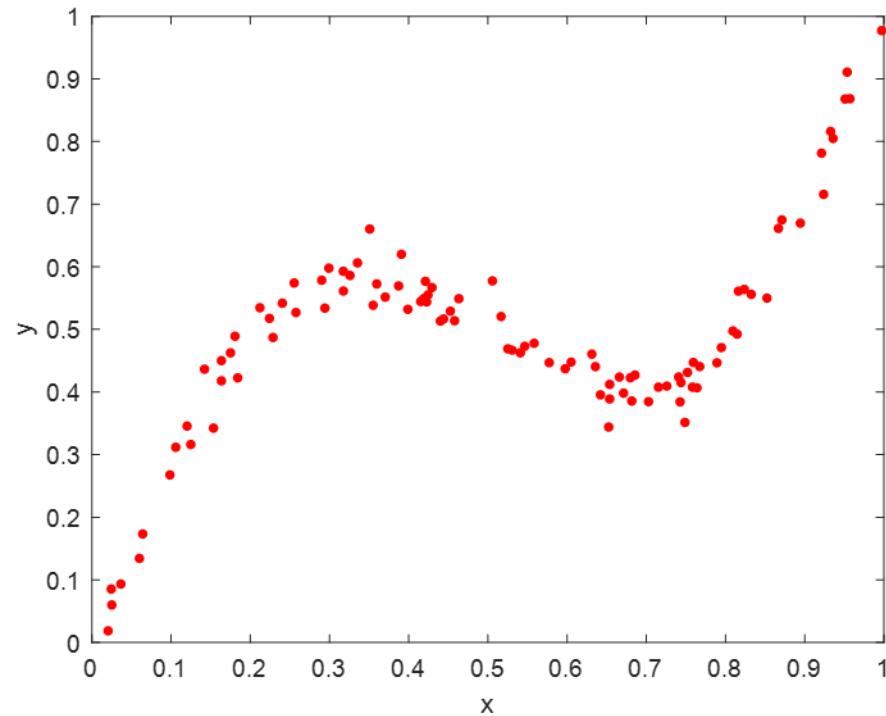
Noise variance 0.0001



Noise variance 0.09

Employ averaging over different test sets to find the model order

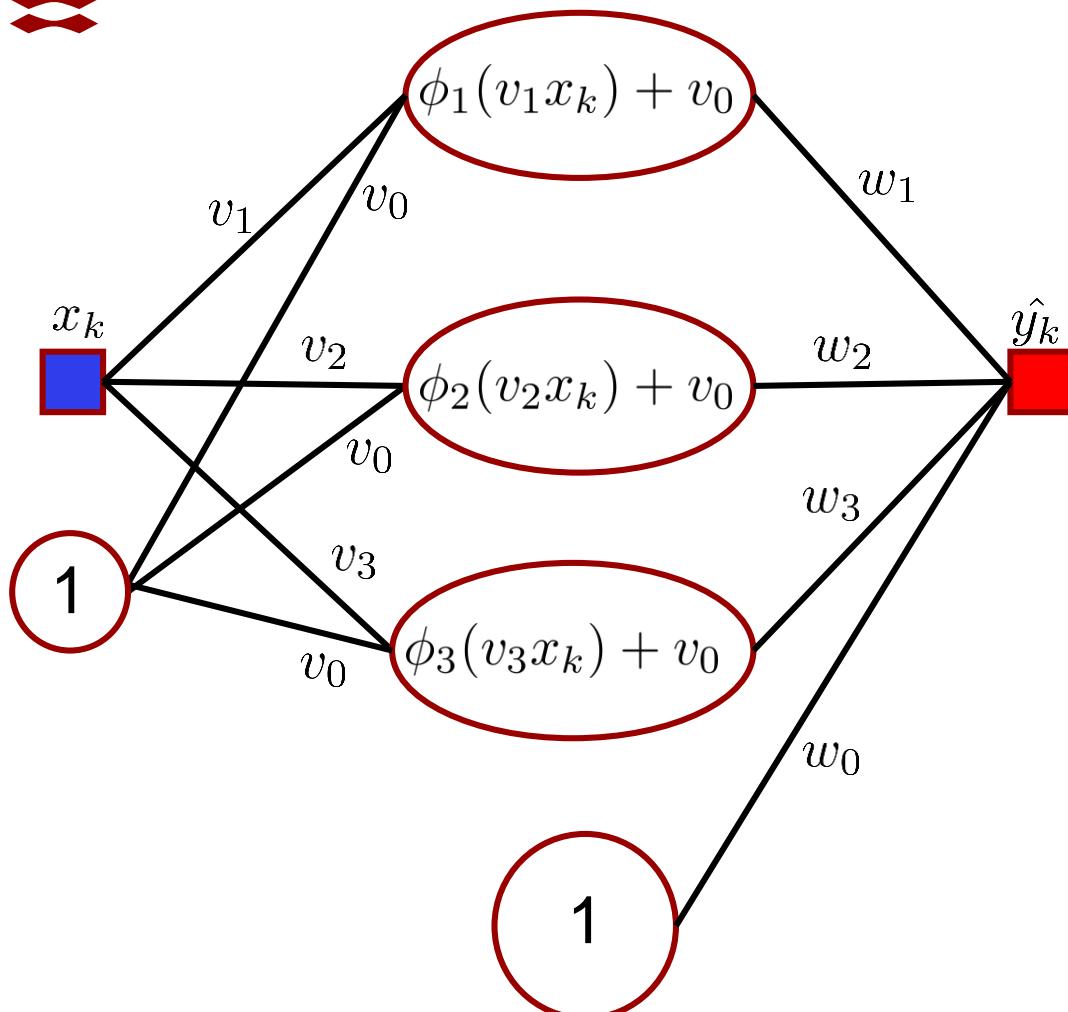
Learn the mapping using neural-network



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

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

Nonlinear model (neural-network)



=

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

Weights that need to be learned:

$$\mathbf{W} = [w_0, w_1, \dots, w_3]^T$$

$$\mathbf{V} = [v_0, v_1, \dots, v_3]^T$$

The problem of determining the weights becomes nonlinear:

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

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

Random weight initialization trick

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & \phi_1(v_1x_1 + v_0) & \phi_2(v_2x_1 + v_0) & \phi_3(v_3x_1 + v_0) \\ 1 & \phi_1(v_1x_2 + v_0) & \phi_2(v_2x_2 + v_0) & \phi_3(v_3x_2 + v_0) \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \phi_1(v_1x_N + v_0) & \phi_2(v_2x_N + v_0) & \phi_3(v_3x_N + v_0) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

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} : 3×3 identity matrix

σ : standard deviation of the weights (**tuning parameter**)

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

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

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:

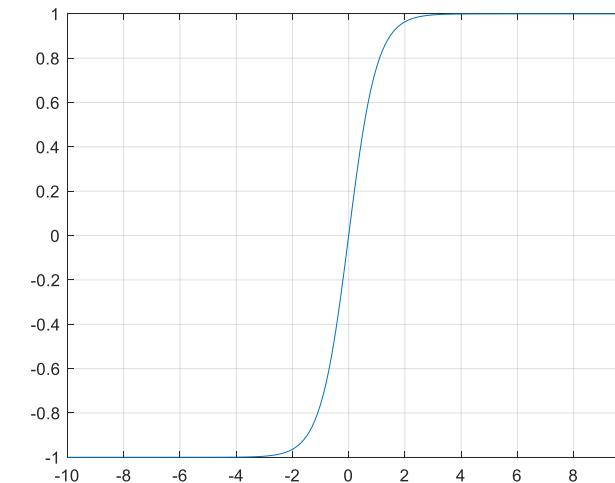
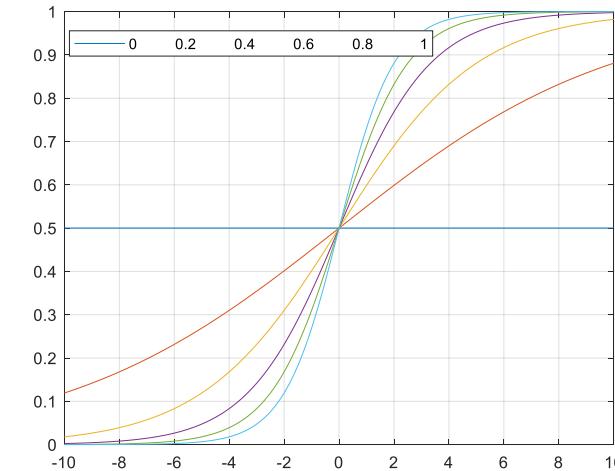
$$\psi(v_k) = \begin{cases} 1 & : v_k \geq 0 \\ -1 & : v_k < 0 \end{cases}$$

Sigmoid function:

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

Tangents function:

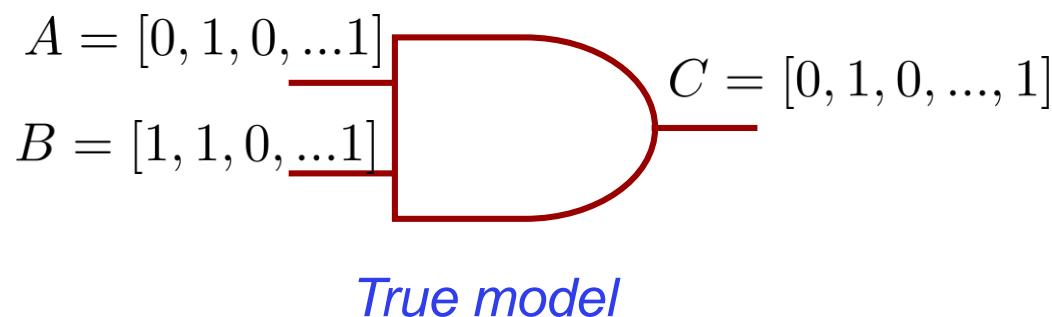
$$\psi(v_k) = \tanh(v_k)$$



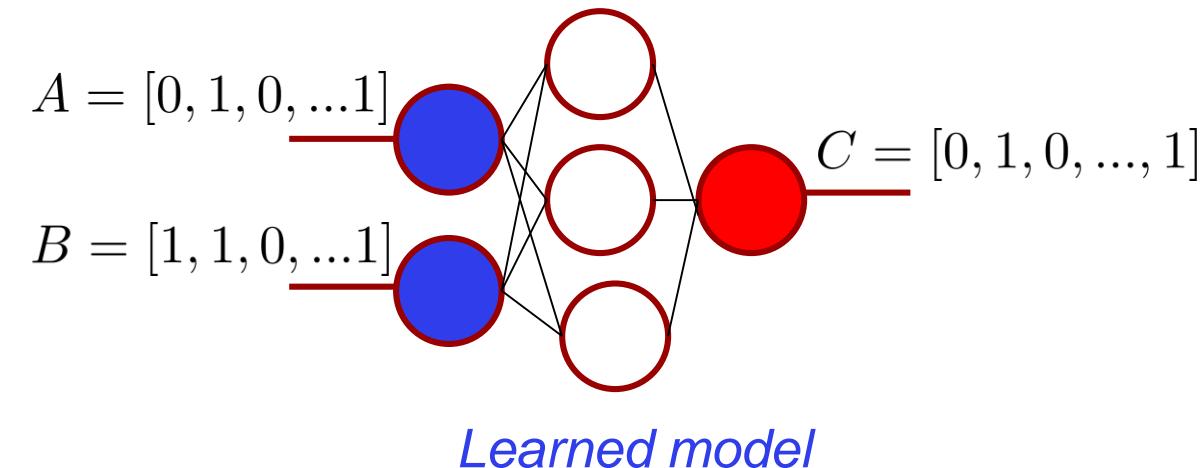
Can we learn mappings between categorical variables?

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

AND gate

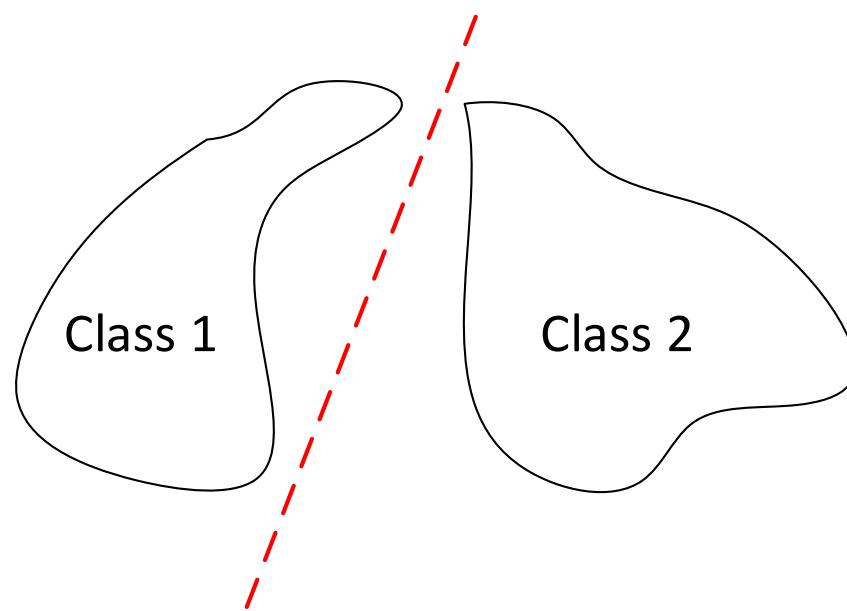


True model

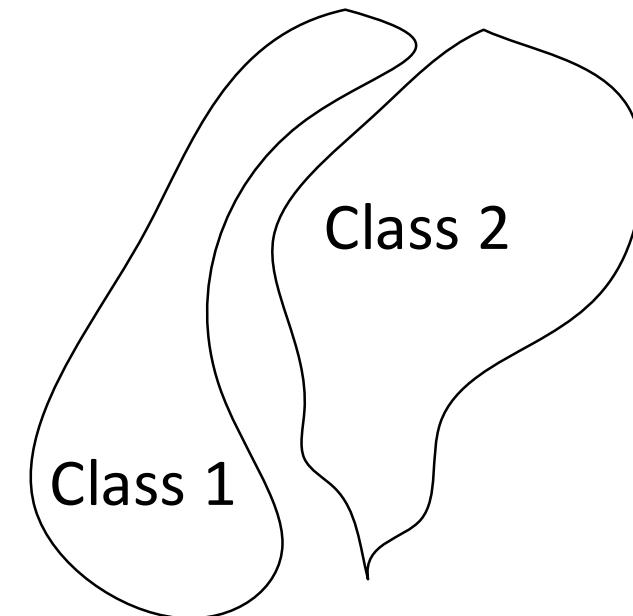


Learned model

Linearly separable classes



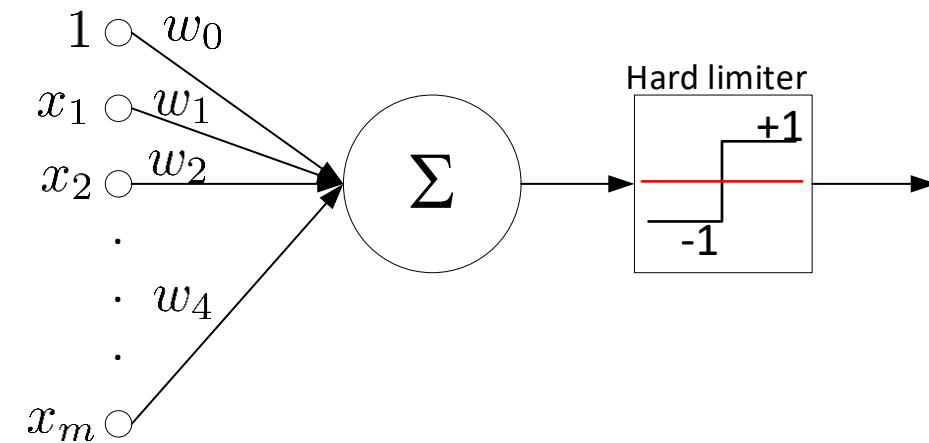
(a) Linearly **separable** classes



(b) Linearly **non-separable** classes

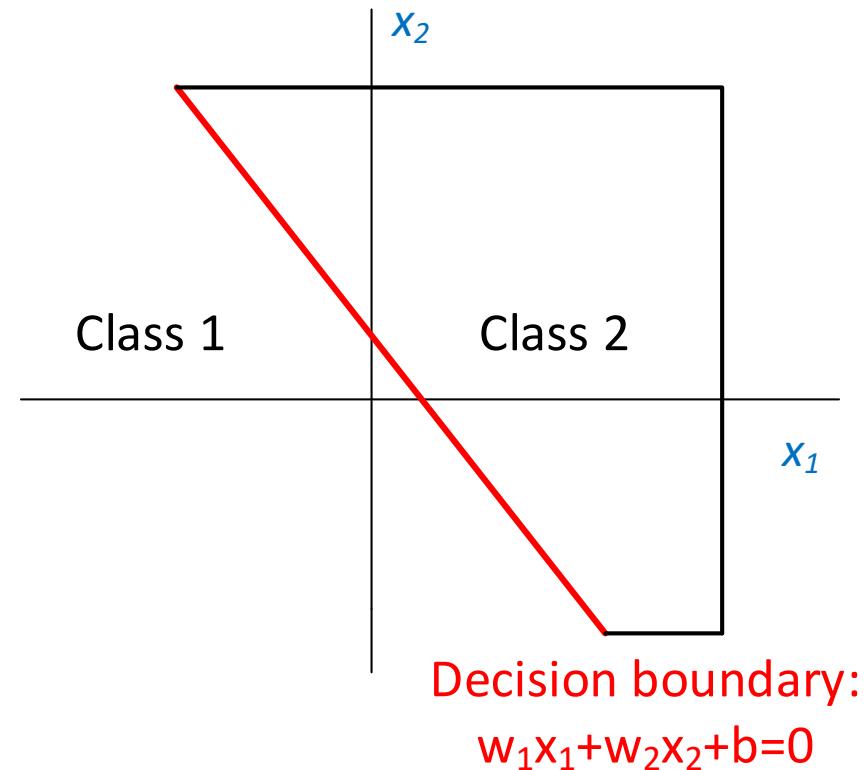
The perceptron only works for linearly separable classes

Rosenblatt perceptron (simplest neural network)



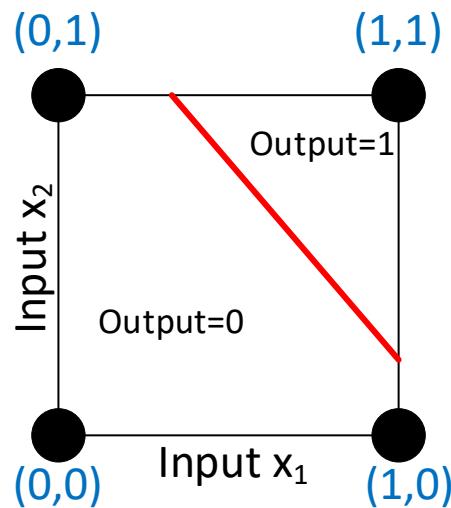
$$v_k = \sum_{j=0}^m w_j x_j = \mathbf{W}^T \mathbf{X}$$

Decision boundary

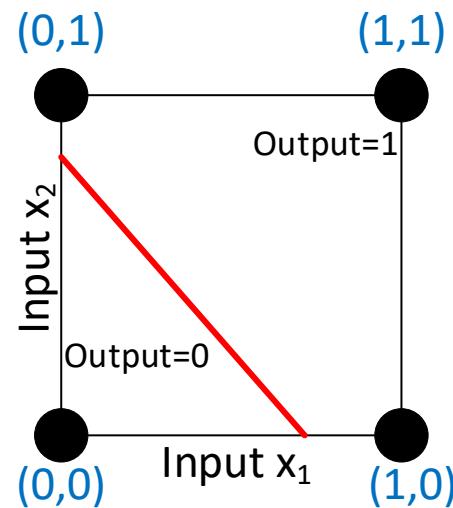


$$v_k = \sum_{j=0}^m w_j x_j = \mathbf{w}^T \mathbf{x} = 0$$

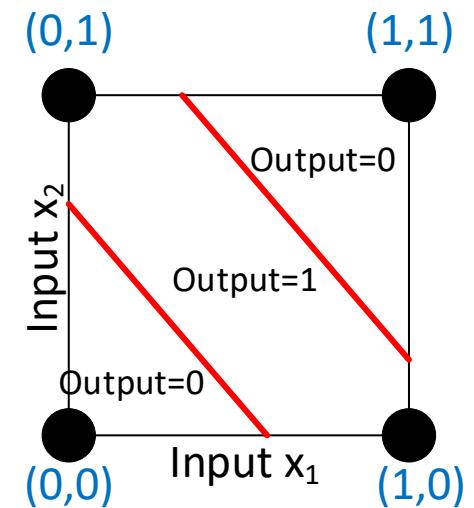
Decision Boundaries (DB)



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:

$$\mathbf{x}(n) = (m+1)\text{-by-1 input vector} \\ = [+1, x_1(n), x_2(n), \dots, x_m(n)]^T$$

$$\mathbf{w}(n) = (m+1)\text{-by-1 weight vector} \\ = [b, w_1(n), w_2(n), \dots, w_m(n)]^T$$

b = bias

$y(n)$ = actual response (quantized)

$d(n)$ = desired response

η = learning-rate parameter, a positive constant less than unity

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \dots$
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

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

where $\text{sgn}(\cdot)$ 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

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

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

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