# Introduction to nonlinear regression models and classification

Course 34242
Lecture 2

Darko Zibar
dazi@dtu.dk

# Agenda

- Repetition of linear regression models
- Basics of nonlinear regression modes
- Neural-network
- Basic idea behind classification
- Decision boundaries
- Least squares for classification
- Logistic regression

# Reading Material

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

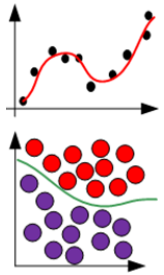
- Introduction (pp. 1-12)
- Chapter 4 (pp. 179-186, pp. 192-198, pp. 203-206)
- Chapter 5 (pp. 225 - 246)

# Machine learning

**Supervised learning**

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

- Regression
- Classification

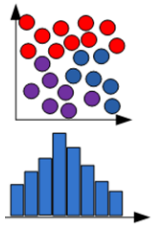


Predict outcomes

**Unsupervised learning**

Given data x learn useful properties: p(x)

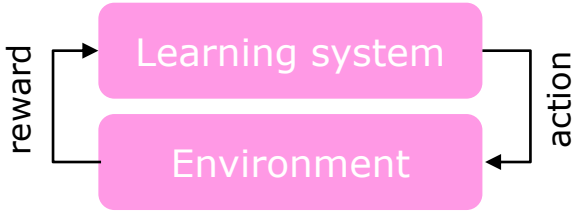
- Clustering
- Density estimation



Describe data

**Reinforcement learning**

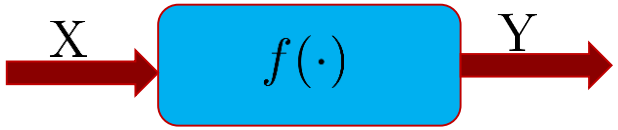
Feedback based optimization with interaction reward



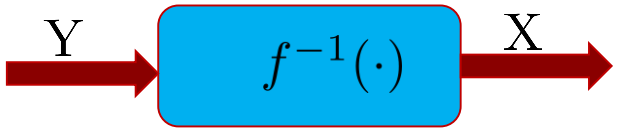

Improve

# Where does machine learning excel?

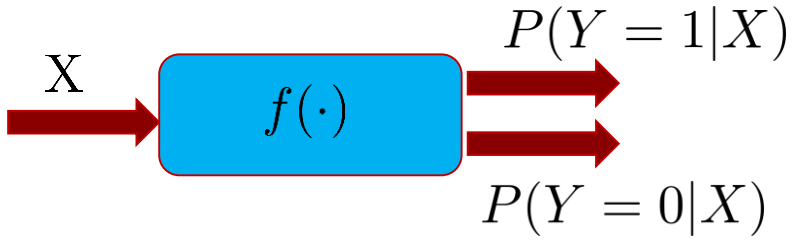
- Learning complex **direct** mappings:




- Learning complex **inverse** mappings:

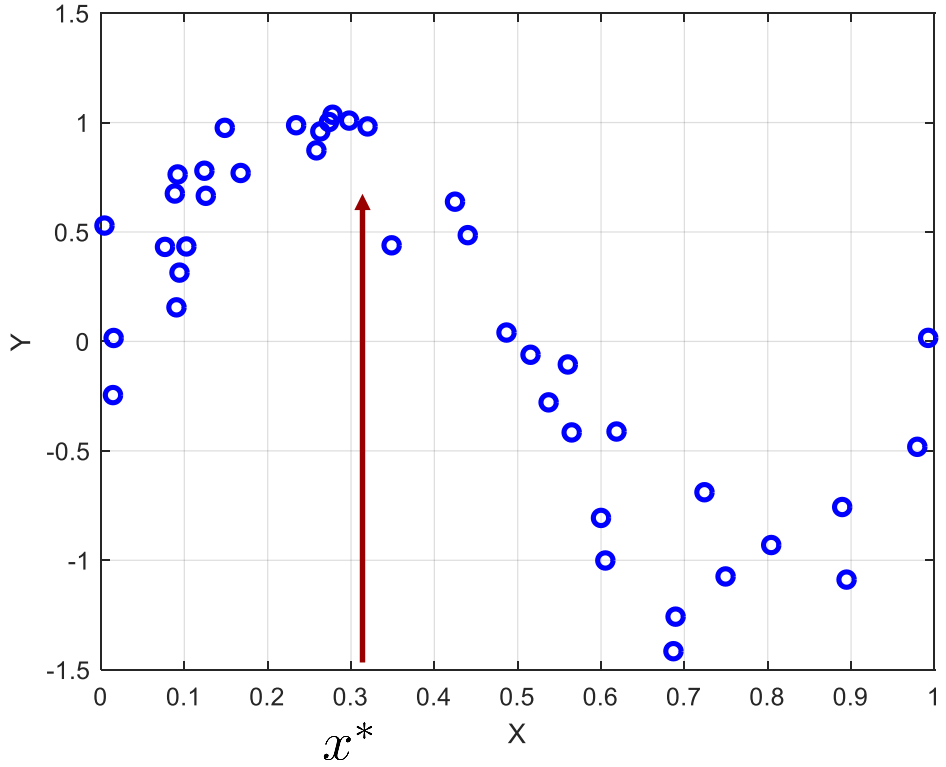



- Learning **decision rules** for complex mappings:




Learn the model, from the data, that represents $f(\cdot)$ and $f^{-1}(\cdot)$

# Learning complex functions



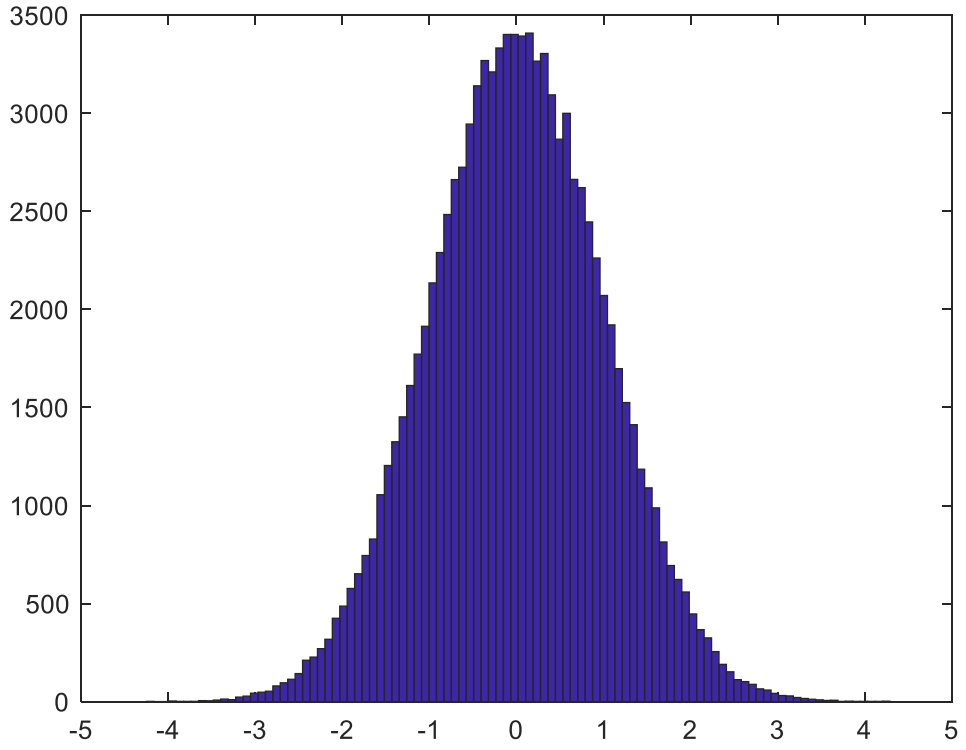

1. Given a training data-set (input and output values): $\mathcal{D} = \{x(k), y(k)\}_{k=1}^{K}$
2. For new input $x^*(k) \notin \mathcal{D}$ compute $y^*(k)$
3. Analytical expression *unknown,* must be learned from: $\mathcal{D} = \{x(k), y(k)\}_{k=1}^{K}$

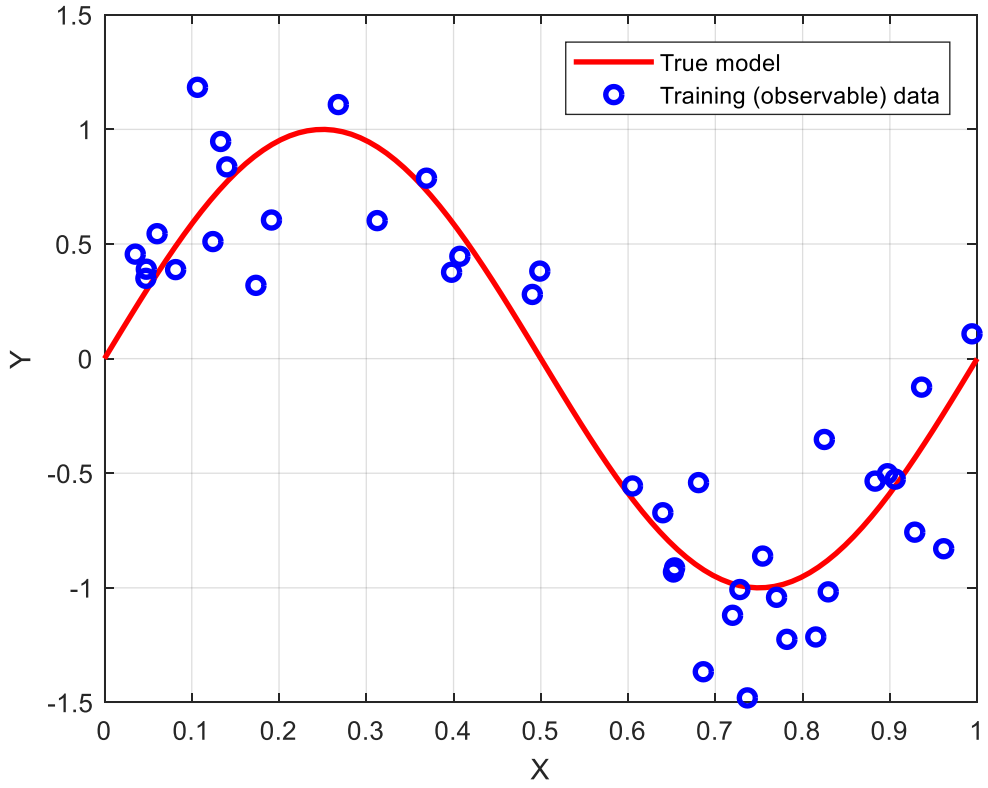
# True model and observable data

Assume that 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 :

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

Or

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

Or

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

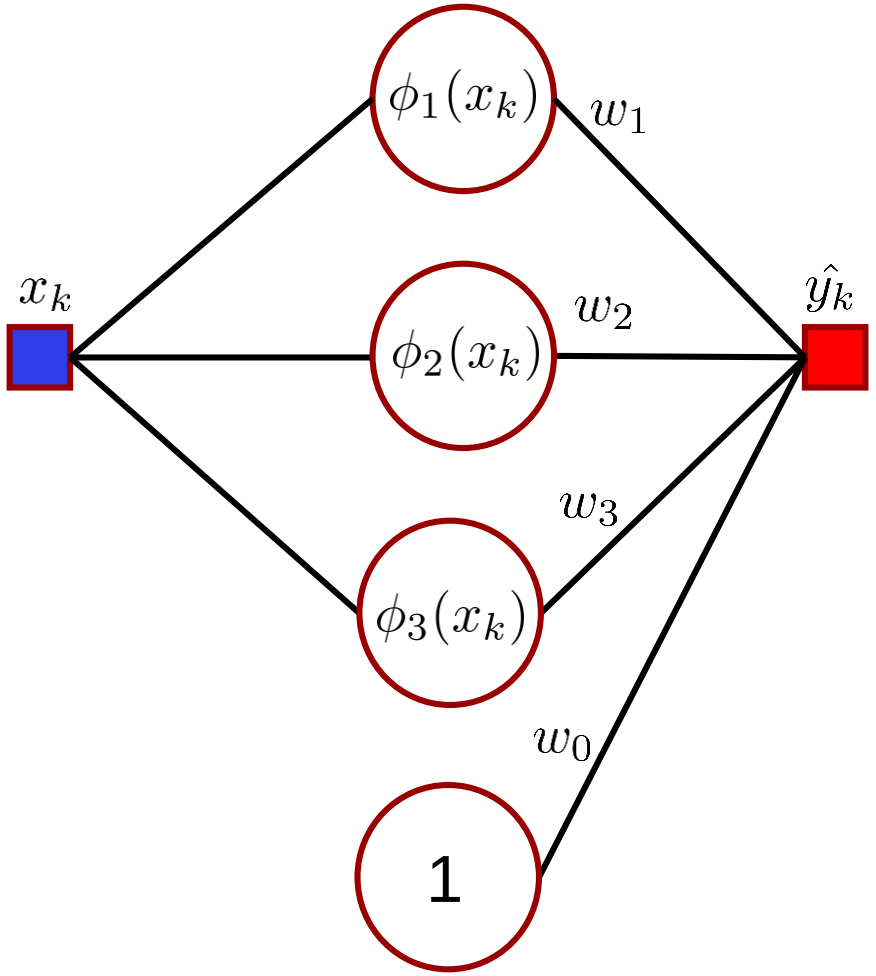
General linear model:

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

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

Objective: choose a model and then determine the weights

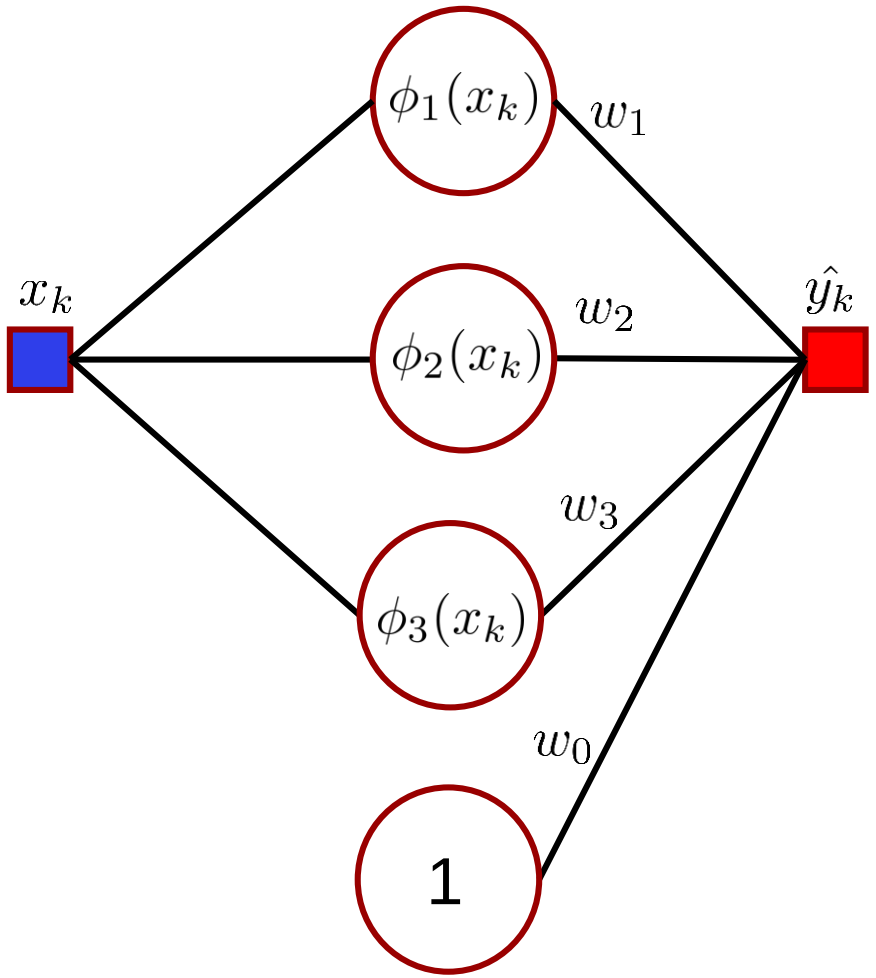
# Topology of the linear model




=

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

# 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:

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

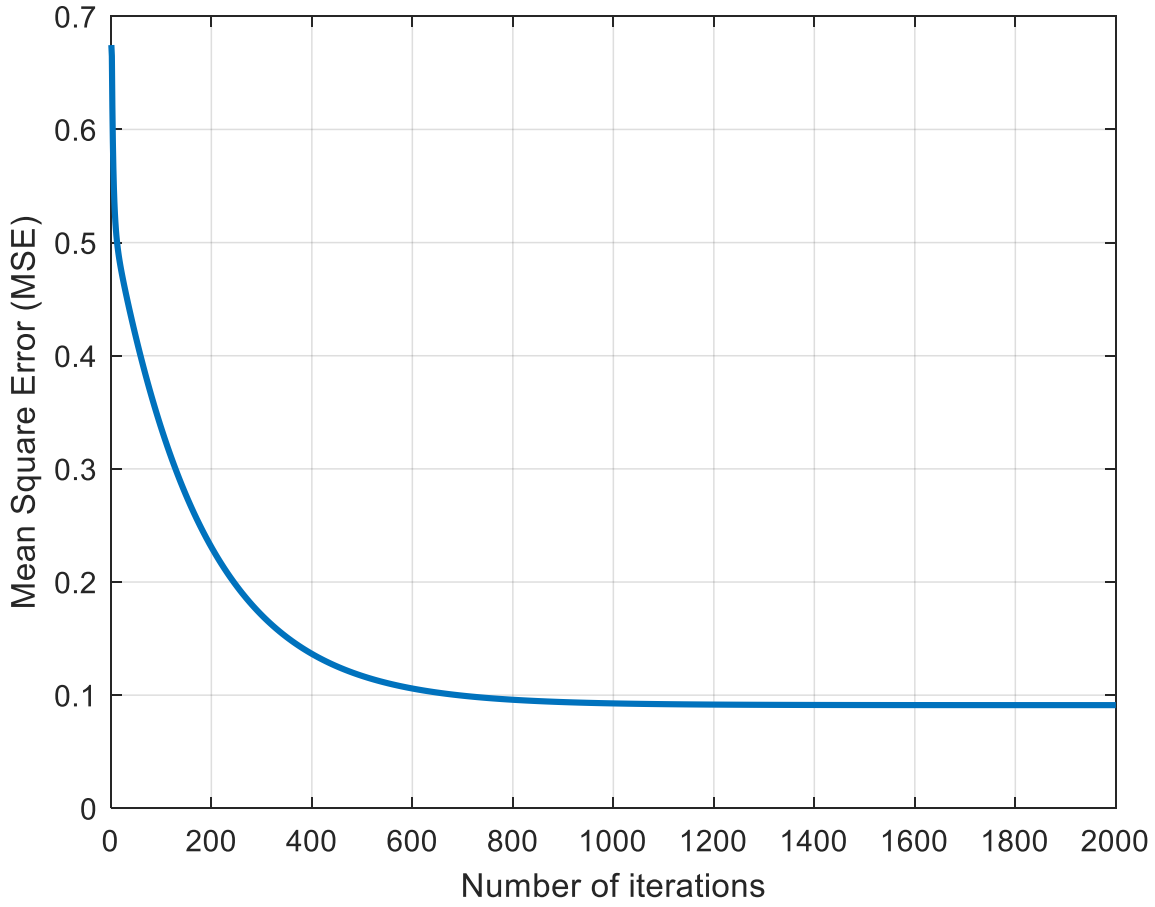
$$= \frac{1}{2}[y_k - \mathbf{w}_k^T \mathbf{\Phi}_k]^2$$

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

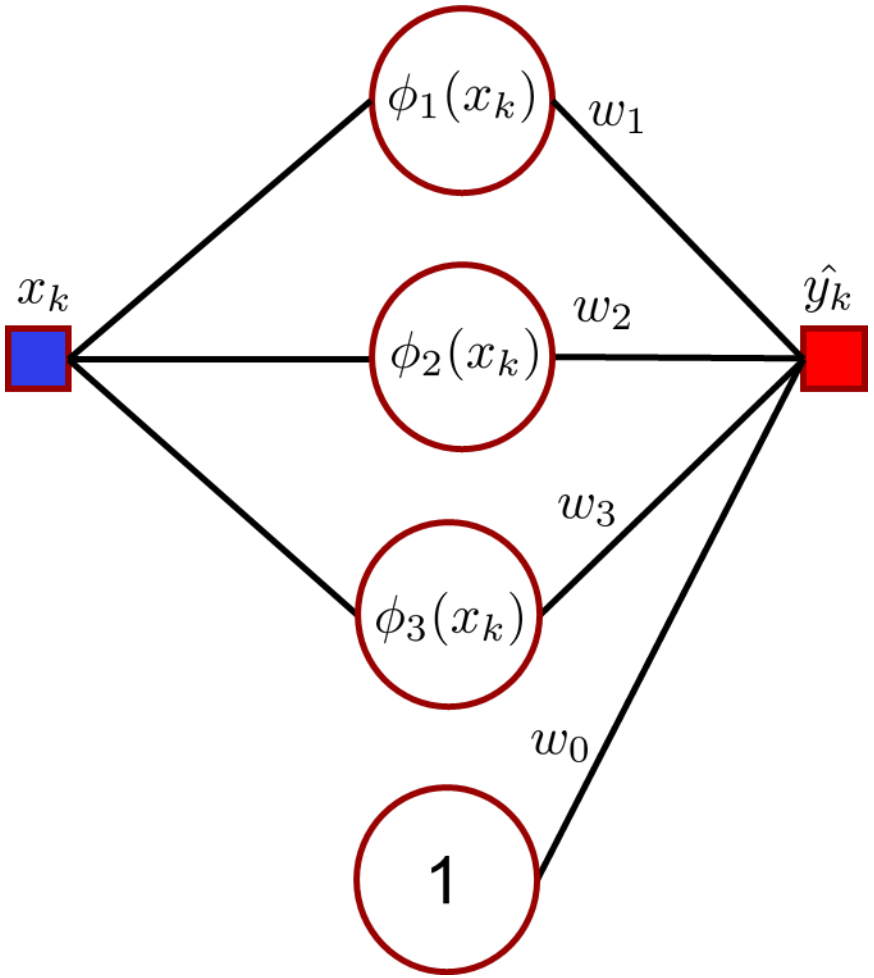
```
for i = 1 : L_iter
    for k = 1 : L_train
        e(k)
        grad = de(j)dw;
        w = w + eta*grad;
    end
    e(i) = mean(e.^2);
end
```

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

# Performance evaluation on the training data-set



# Determining the weights in one step




The output is expressed as:

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

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

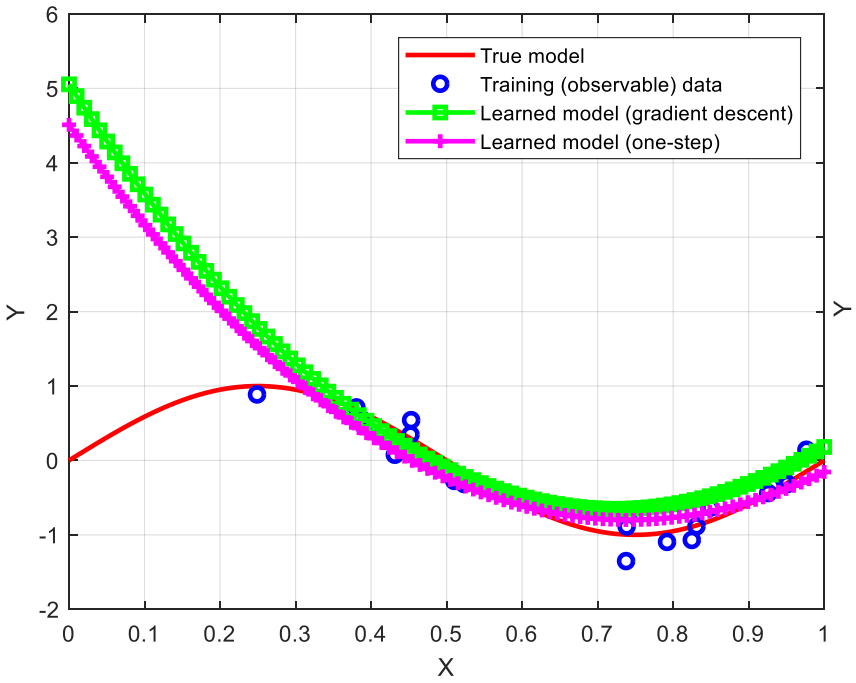
The weights are computed as Moore-Penrose pseudo inverse:

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

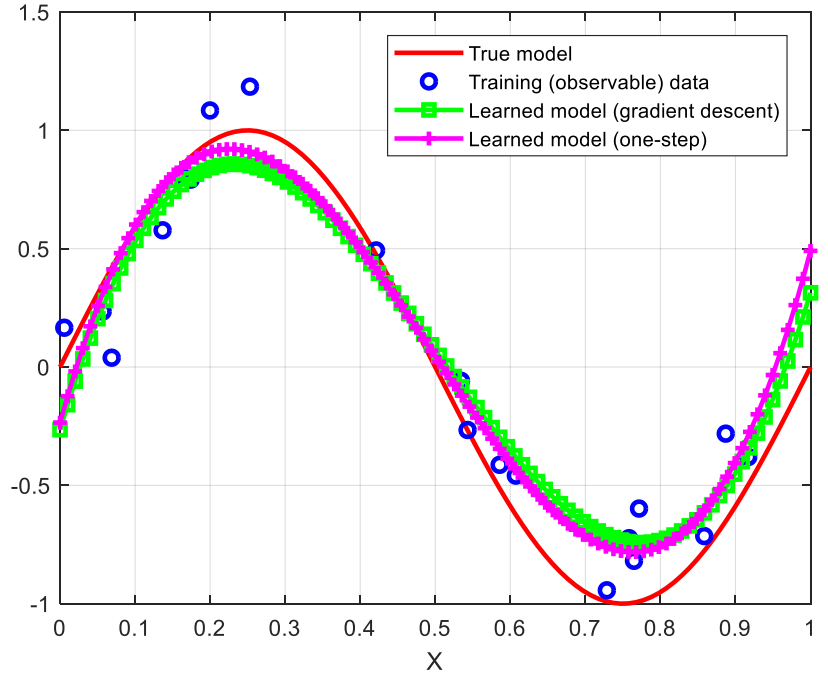
If $\mathbf{\Phi}$ is square and invertible:

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

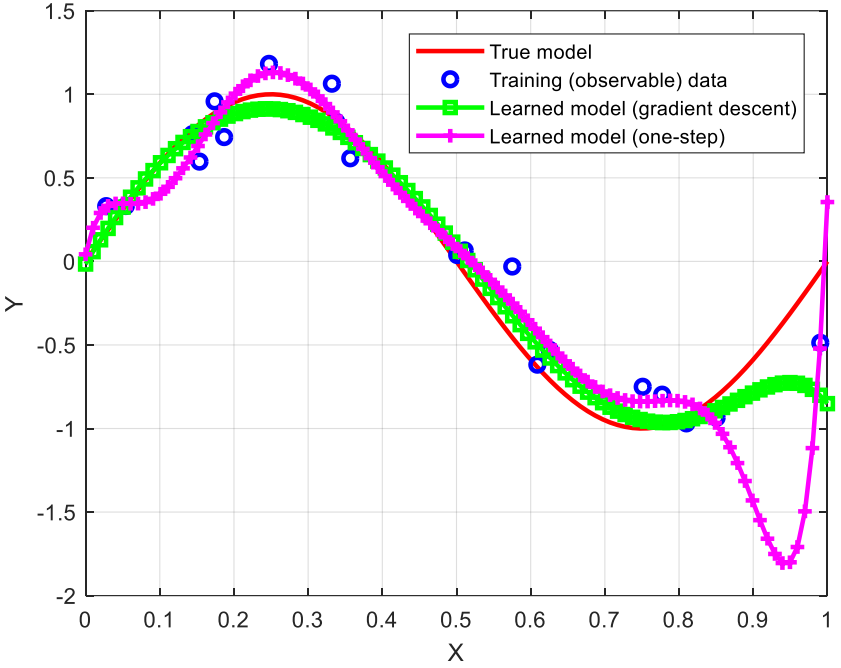
# Evaluating the learned the model




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

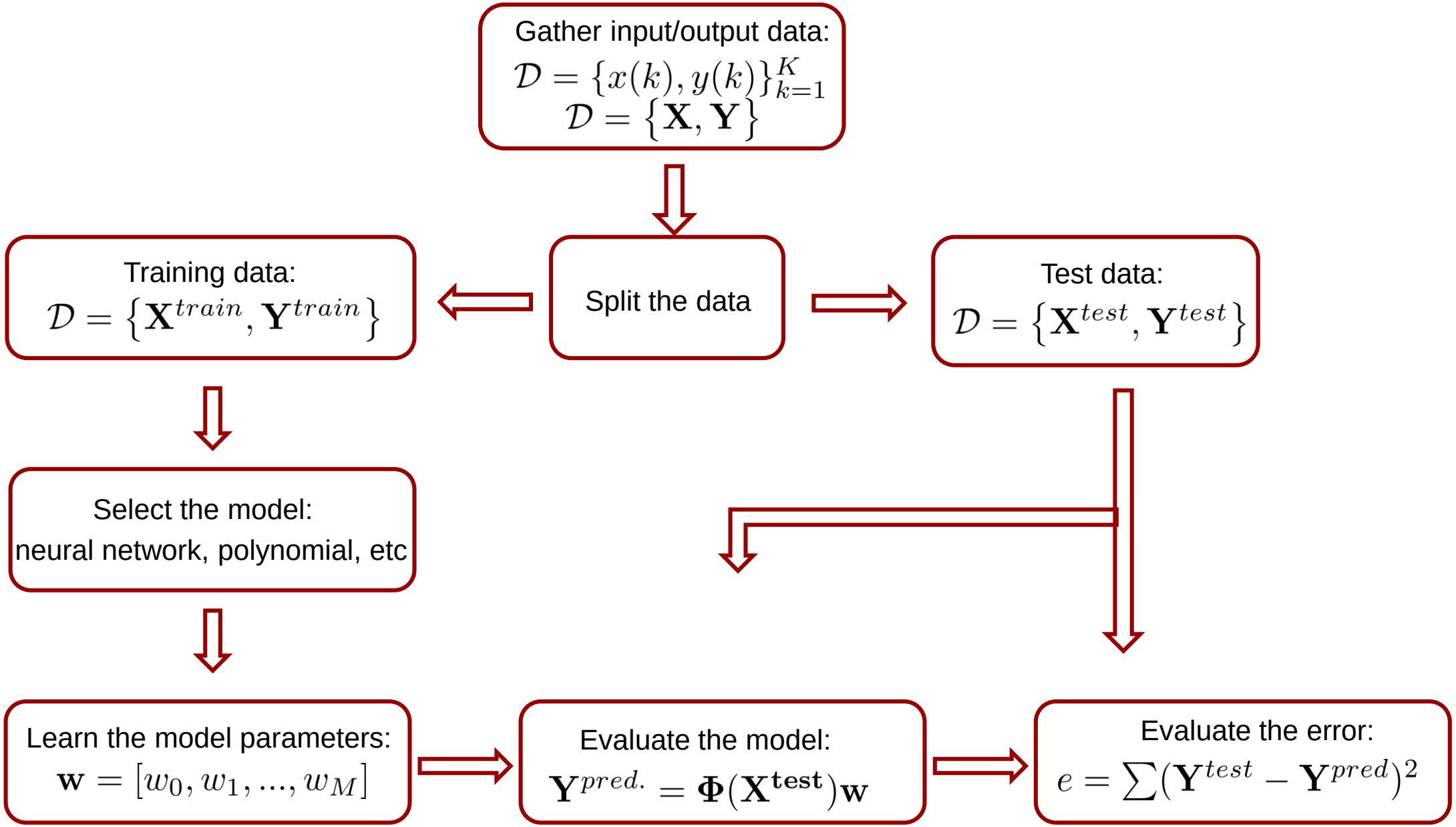



*Assumed polynomial model order M=3*
*(the right model)*

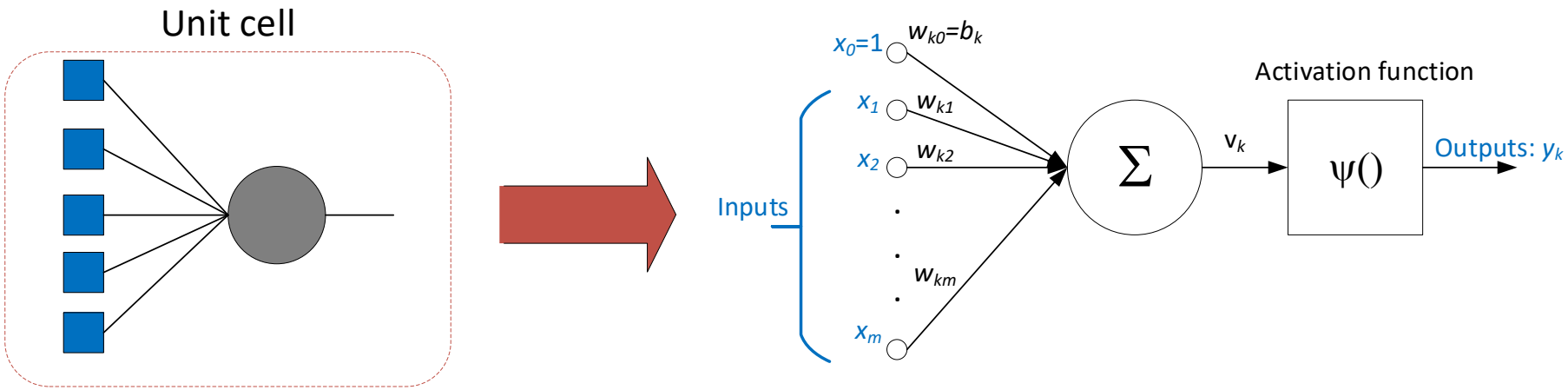



*Assumed polynomial model order M=9*
*(too complex model: overfitting)*

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


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:
$\mathbf{Y}^{pred.} = \mathbf{\Phi}(\mathbf{X^{test}})\mathbf{w}$
Evaluate the error:
$e = \sum(\mathbf{Y}^{test} - \mathbf{Y}^{pred})^2$

# General perceptron




Expressed in mathematical form:

$$y_k = \sum_{j=0}^{m} \psi(w_{kj} x_j) = \psi(\mathbf{w}^T \mathbf{x})$$

where:

$$\mathbf{w} = [b_k, w_{k1}, w_{k2}, ..., w_{km}]^T$$

$$\mathbf{x} = [1, x_1, x_2, ..., x_m]^T$$

# Examples of common basis/activation functions

Threshold function:

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

Sign function:

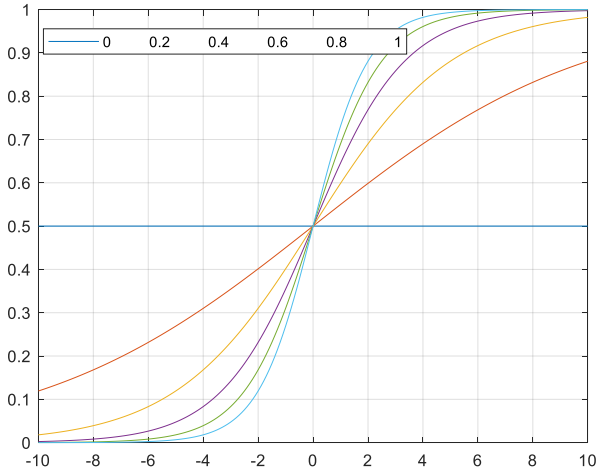
$$f(x) = \begin{cases} 1 & : x \geq 0 \\ -1 & : x < 0 \end{cases}$$

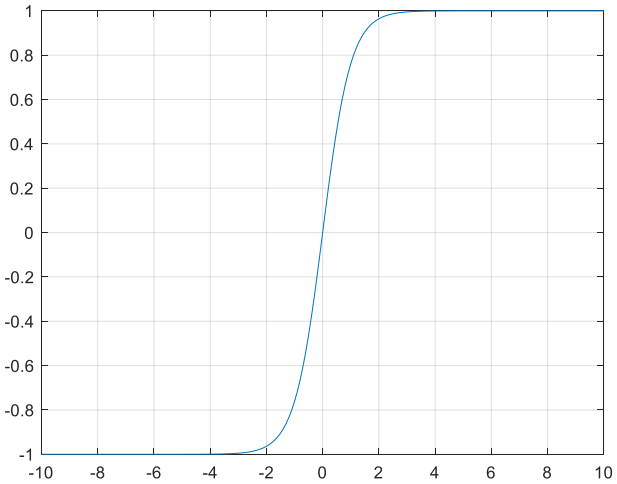
Sigmoid function:

$$\sigma(x) = \frac{1}{1+\exp(-ax)}$$

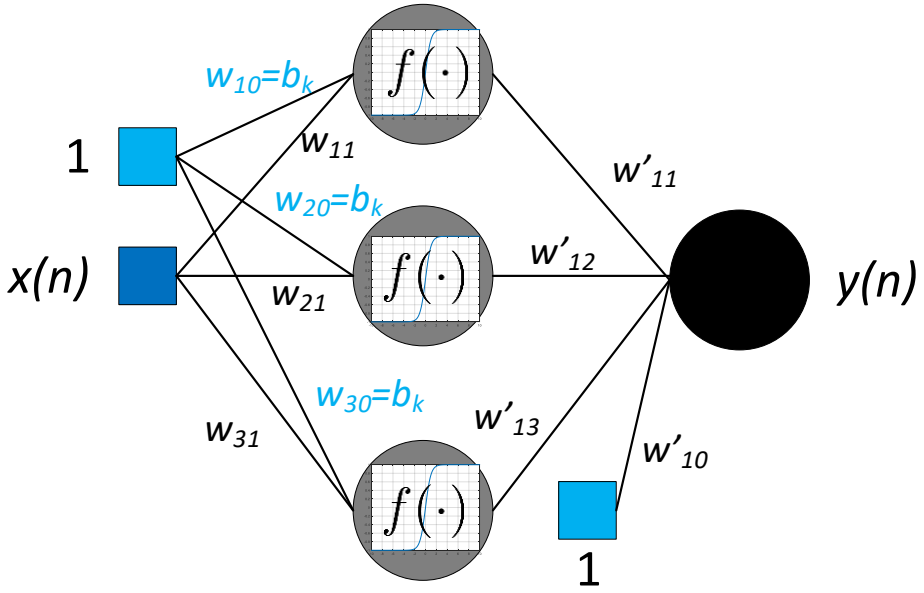
Tangents function:

$$f(x) = \tanh(x)$$



# Nonlinear regression model: (neural network)




$$y(n) = \mathbf{W}' f(\mathbf{W}[1, x(n)]^T)$$

Objective: learn the adjustable weights:

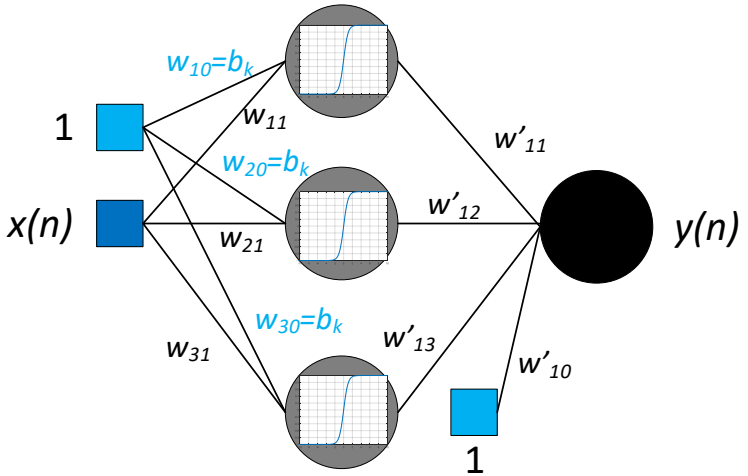
$$\mathbf{W} = \begin{bmatrix} w_{10} & w_{11} \\ w_{20} & w_{21} \\ w_{30} & w_{31} \end{bmatrix} \quad \mathbf{W}' = \begin{bmatrix} w'_{10} \\ w'_{12} \\ w'_{12} \\ w'_{13} \end{bmatrix} \quad f(x) = \tanh(x)$$

# Error definition

1. Given a data set (input and output values):

$$\mathcal{D} = \{x(n), y(n)\}_{n=1}^{L_{train}}$$

2. Propagating $x(n)$ through neural network we get $y(n)$



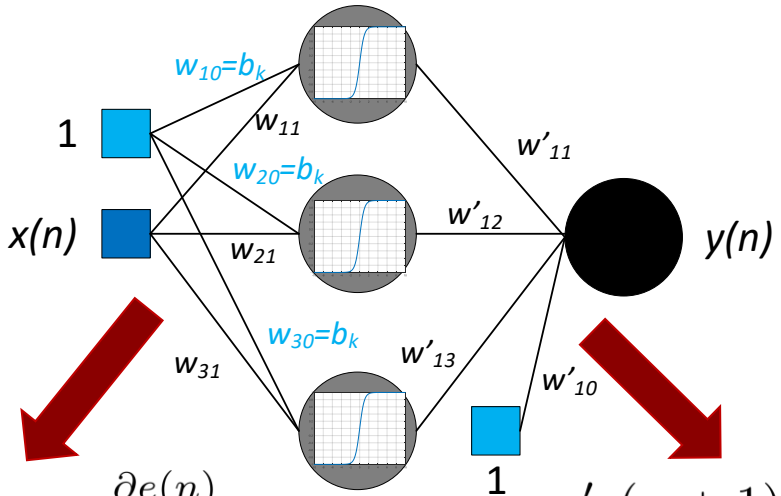

3. The error is defined as:

$$e(n) = \frac{1}{2}[d(n) - y(n)]^2$$

4. Averaging over the entire data-set:

$$E_{av} = \frac{1}{N}\sum_{n=1}^{L_{train}} e(n)$$

# Weight update rule



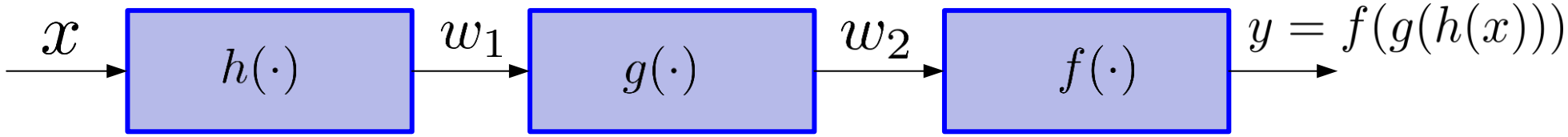

$$w_{ji}(n+1) = w_{ji}(n) - \eta \frac{\partial e(n)}{\partial w_{ji}(n)}$$

$$w'_{ij}(n+1) = w'_{ij}(n) - \eta \frac{\partial e(n)}{\partial w'_{ij}(n)}$$

On-line learning framework:

```
for k = 1 : L_iter
    randomly reshuffle data set
    for n = 1 : L_train
        w'(n+1) = w'(n) - eta*d_e_dw';
        w(n+1) =  w(n) - eta*d_e_dw;
    end
    E_epochs(k) = [w(L_train),w'(Ltrain)];
    compute MSE
end
```

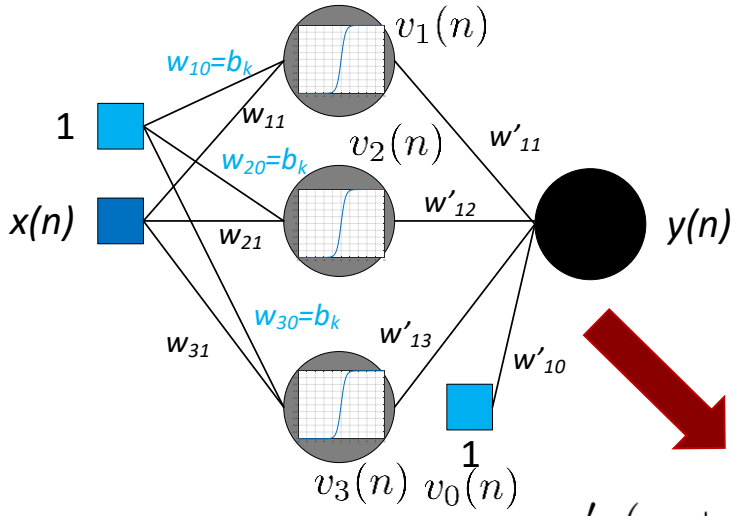
# All we need is the chain rule




$$y = f(w_2)$$
$$w_2 = g(w_1)$$
$$w_1 = h(x)$$

$$\frac{dy}{dx} = \frac{dy}{dw_2}\frac{dw_2}{dw_1}\frac{dw_1}{dx}$$

# Computing derivatives I




$$w'_{ij}(n+1) = w'_{ij}(n) - \eta \frac{\partial e(n)}{\partial w'_{ij}(n)}$$

The error defined as:

$$e(n) = \frac{1}{2}[d(n) - y(n)]^2$$

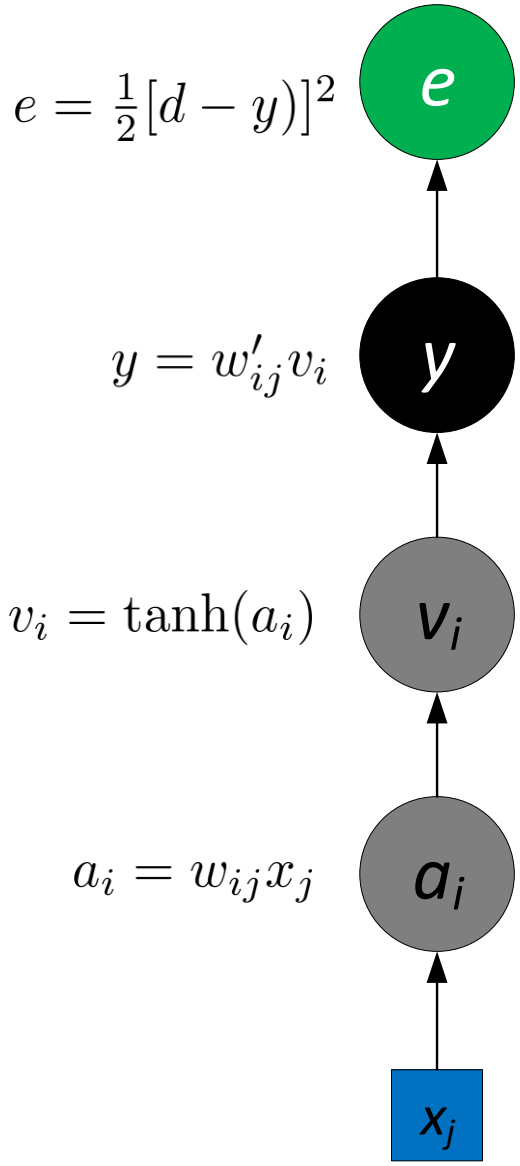
The output of the network:

$$y(n) = \sum_{j=0}^{3} w'_{1j} v_j(n)$$

The gradient then becomes:

$$\frac{\partial e(n)}{\partial w'_{1j}(n)} = [d(n) - y(n)]v_j$$

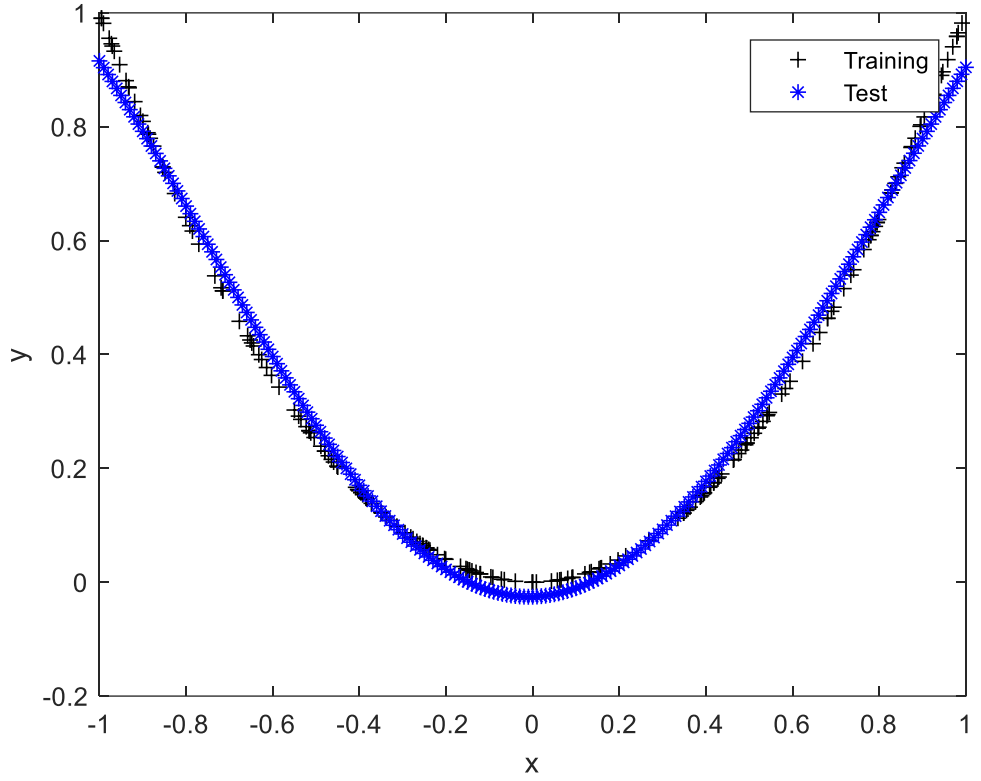
# Computing derivatives II




$$w_{ji}(n+1) = w_{ji}(n) - \eta \frac{\partial e(n)}{\partial w_{ji}(n)}$$

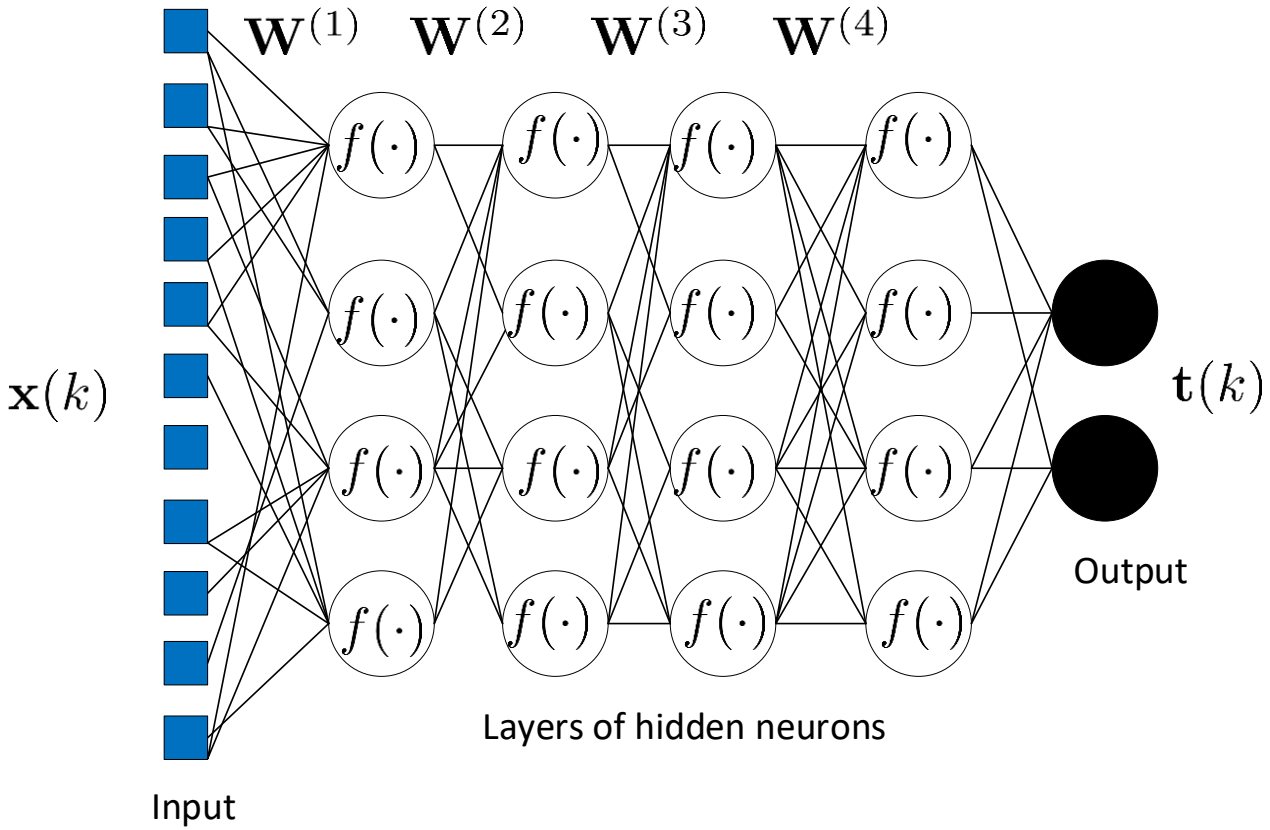
$$\frac{de}{dw_{ij}} = \frac{de}{dy}\frac{dy}{dv_i}\frac{dv_i}{da_i}\frac{da_i}{dw_{ij}}$$

# Learning parabolic function




```
%generate training data
L_train = 500;
x_train = -1 + (1+1).*rand(L,1);
y_train = x_train.^2;
%generate test data
L_test = 200;
x_test = -1 + (1+1).*rand(L_test,1);
y_test= x_test.^2;
```

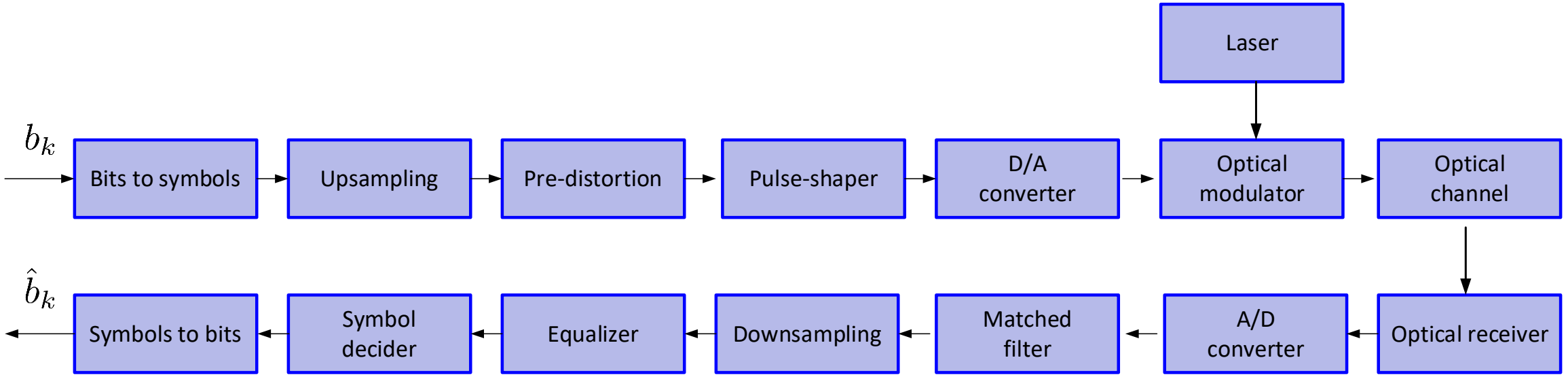
# Multi-layer neural network




-How do we determine weight matrices?
-How do we find number of hidden nodes?
-How do we find number of hidden layers?
-How do we find activations functions?
-How do we find the right topology?

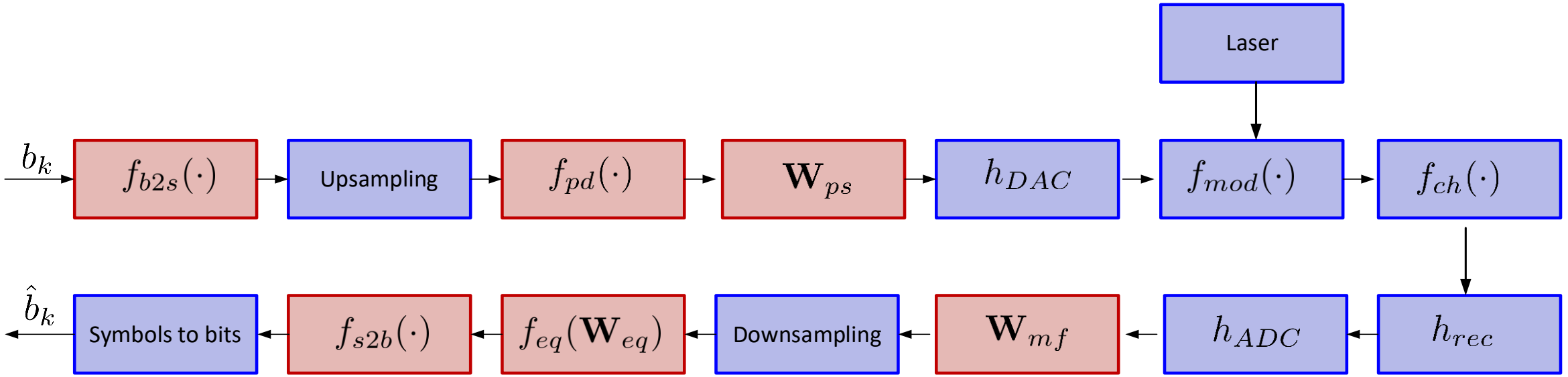
Making multi-layer networks work is complex

# Fiber-optic communication systems




Machine learning can be used to perform global optimization in the presence of system impairments

# Degrees of freedom for the optimization

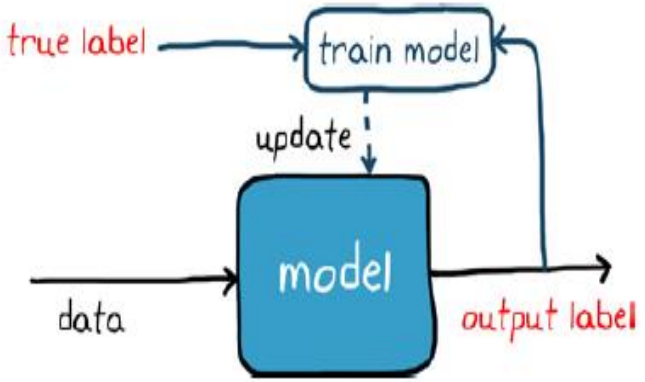


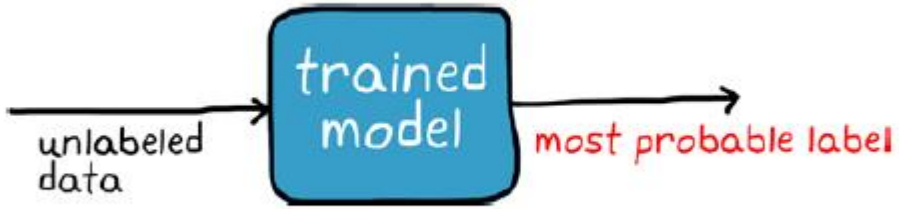

End-to-end learning allows for joint optimization of the transmitter and receiver side DSP blocks

# Supervised learning classification

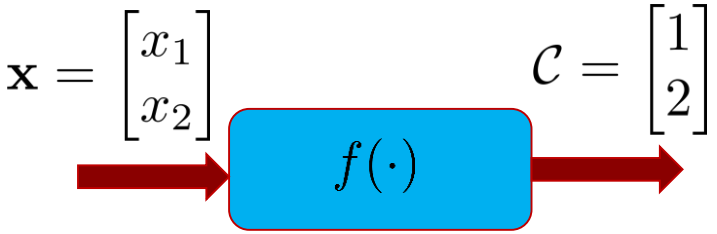
animal dataset (labeled)

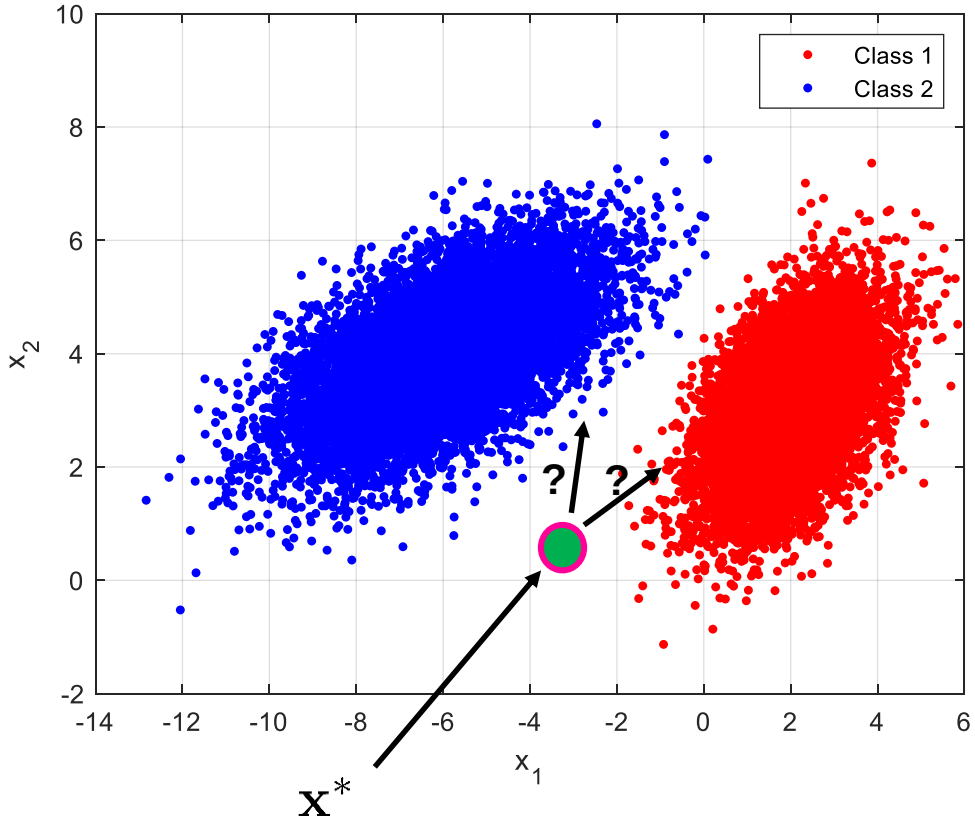
| species | weight | height | num. of legs | communal living | domesticatable | .. |
|---|---|---|---|---|---|---|
| rat | 1.3 | 1.1 | 4 | yes | yes | .. |
| robin | 1.2 | 0.8 | 4 | no | no | .. |
| elephant | 98.5 | 12.2 | 4 | yes | no | .. |
| rabbit | 2.5 | 2.1 | 4 | yes | yes | .. |
| spider | 0.1 | 0.2 | 8 | no | no | .. |
| .. | .. | .. | .. | .. | .. | .. |



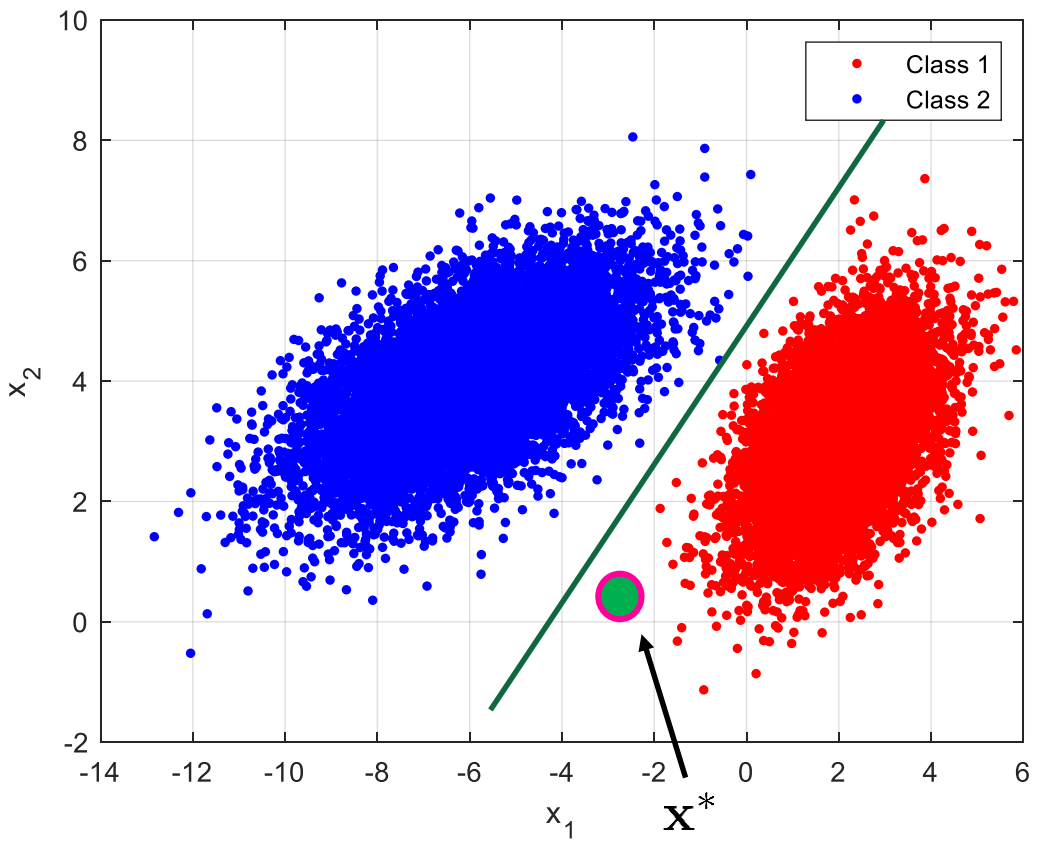
# Classification problem

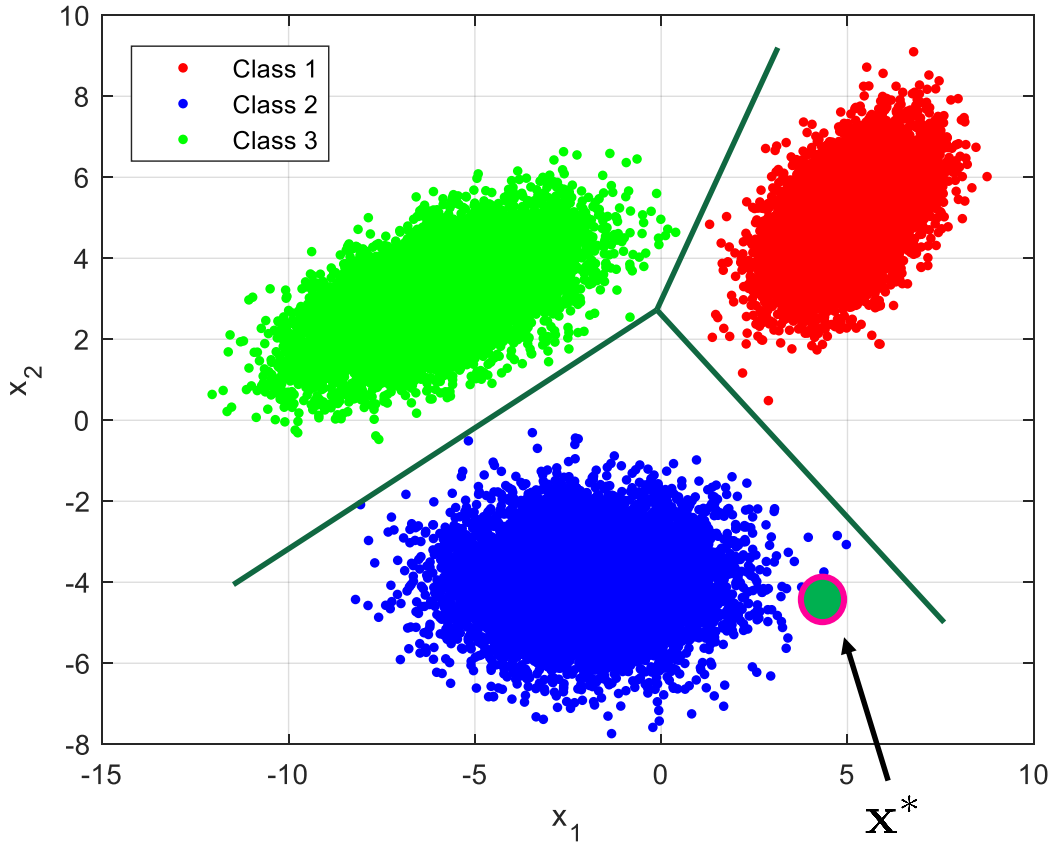



1. Given a training data-set (input and output class): $\mathcal{D} = \{\mathbf{x}(k), \mathcal{C}(k)\}_{k=1}^{K}$
2. For new input $\mathbf{x}^*(k) \notin \mathcal{D}$ predict the corresponding class
3. Decision boundary must be learned from: $\mathcal{D} = \{\mathbf{x}(k), \mathcal{C}(k)\}_{k=1}^{K}$

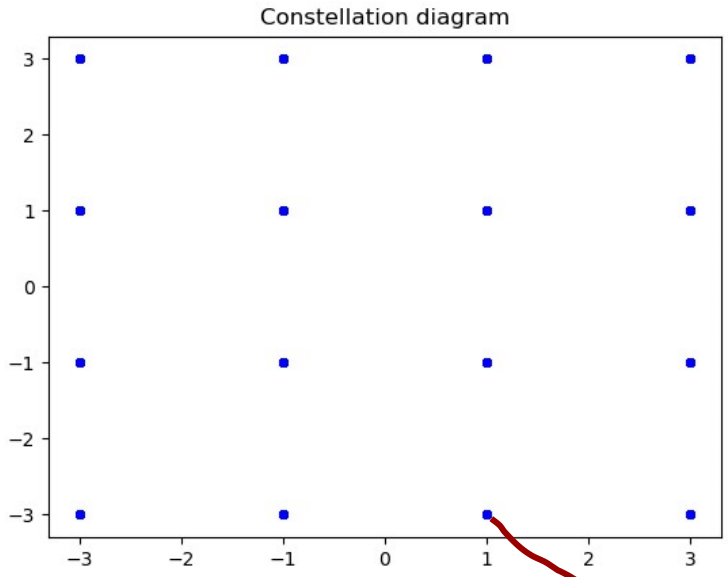
# The importance of decision boundary



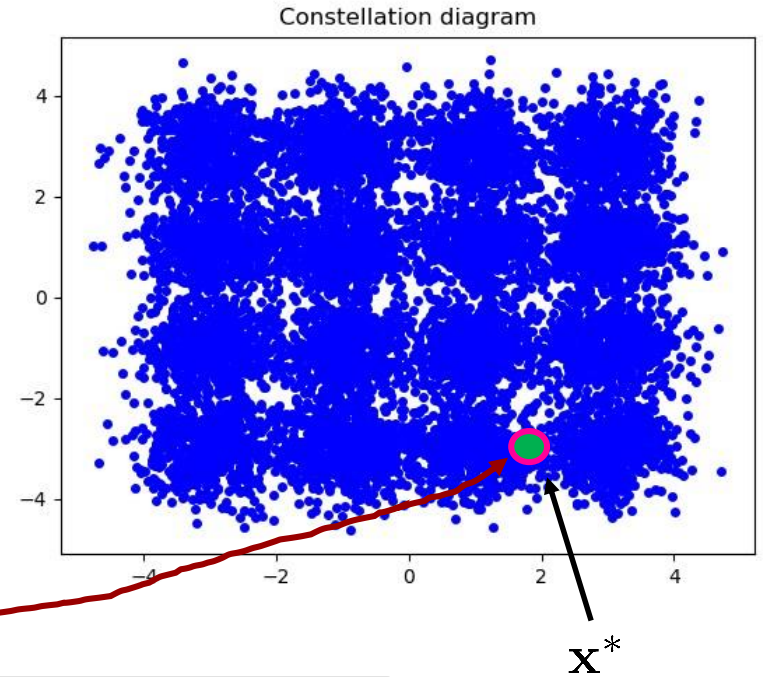




Decision boundary separates classes

# Classification in data-communication


Channel adds noise




Classify the received symbol $x^*$ to the closest transmitted symbol

# Discrete time channel model

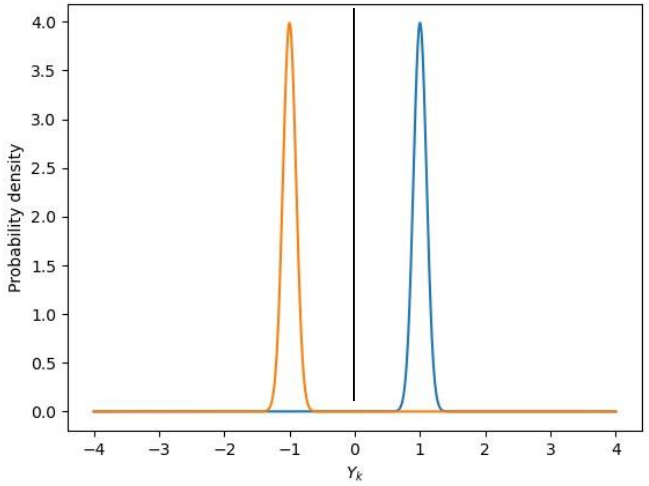
Relationship between the transmitted and the received signal through noisy channel can be modelled as:

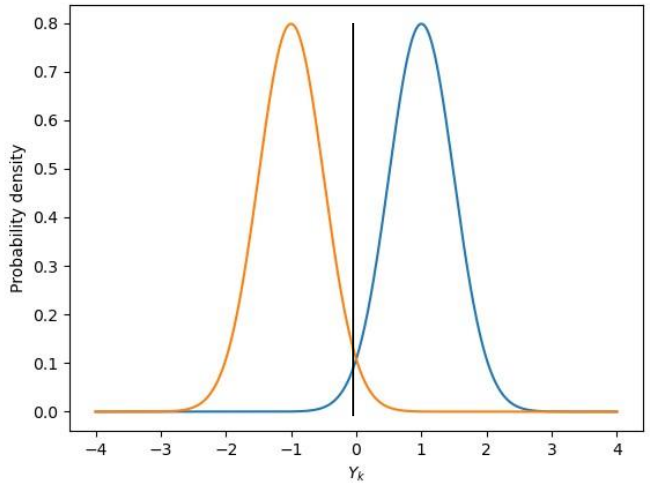
$$Y_k = X_k + N_k$$

$X_k$ : Transmitted discrete symbols that randomly take value +1 or -1

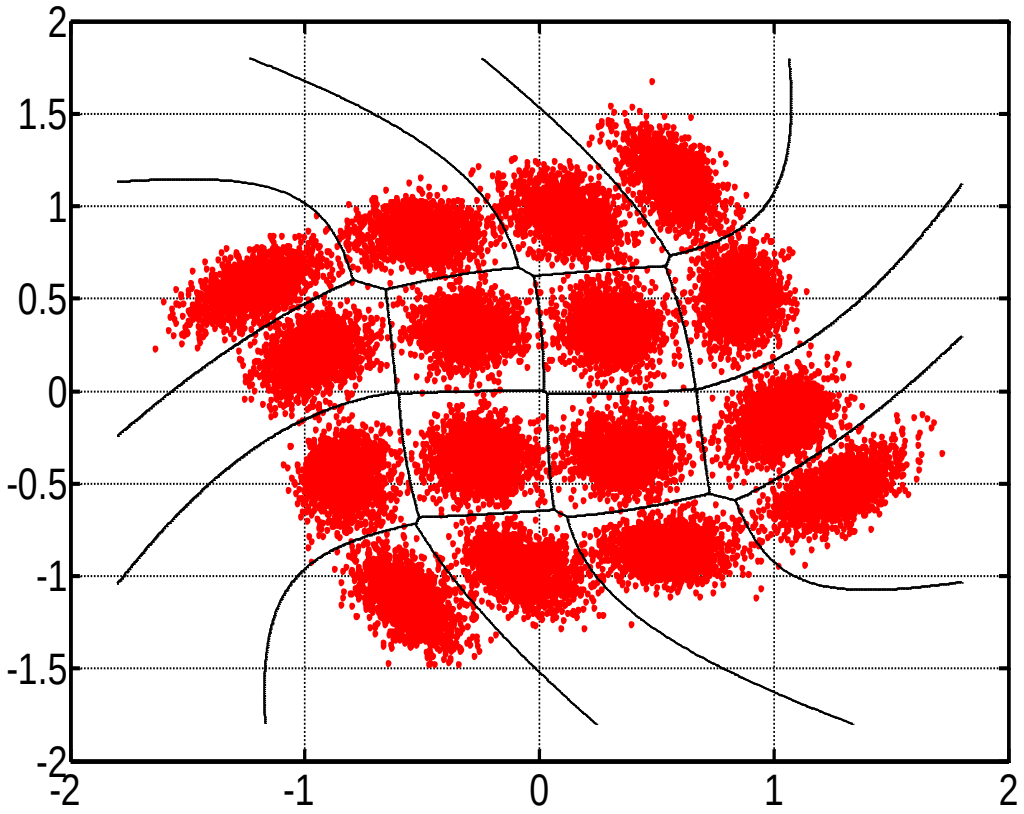
$N_k$ : Additive White Gaussian Noise, AWGN, with zero mean and variance $\sigma^2$

$Y_k$ : Received noisy symbols that have Gaussian distribution with means of +1 or -1, respectively



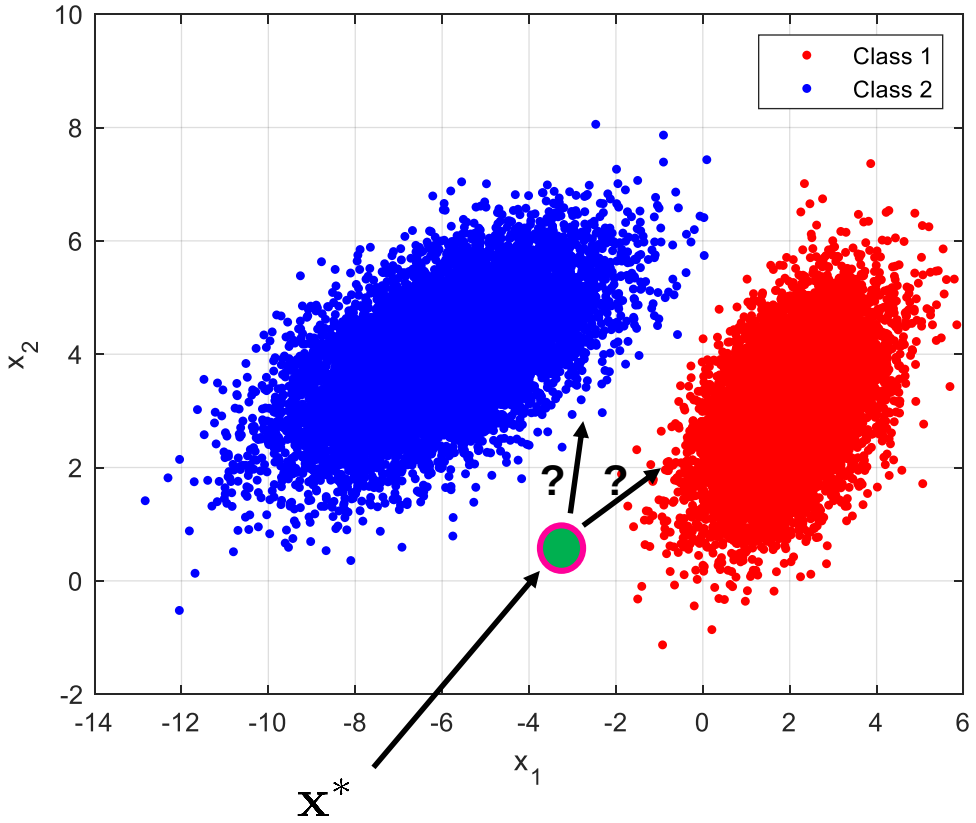



# Classification in data-communication




Nonlinear decision boundaries separates classes

# Discriminant function




Linear discriminant function:

$$y(\mathbf{x}) = w_1 x_1 + w_2 x_2 + w_0 = \mathbf{w}^T \mathbf{x} + w_0$$

Inputs: $\mathbf{x} = [x_1, x_2]^T$

Weight vector: $\mathbf{w} = [w_1, w_2]^T$

Bias: $w_0$

Decision boundary: $y(\mathbf{x}) = 0$

Decision rule:

if $y(\mathbf{x}) \geq 0$
    $\mathbf{x}$ assigned to class 1
else
    $\mathbf{x}$ assigned to class 2

The objective is to determine the weights and bias $\mathbf{w} = [w_0, w_1, w_2]$

# Building the data-set for two classes

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 1.344 | 0.344 | Class 1 |
| 1.112 | 0.241 | Class 2 |
| 2.196 | 7.100 | Class 1 |

modify to

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 1.344 | 0.344 | 01 |
| 1.112 | 0.241 | 10 |
| 2.196 | 7.100 | 01 |

# Building the data-set for three classes

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 1.344 | 0.344 | Class 1 |
| 1.112 | 0.241 | Class 2 |
| 2.196 | 7.100 | Class 1 |
| 0.811 | 1.344 | Class 3 |
| 5.124 | 2.344 | Class 2 |
| 7.196 | 6.344 | Class 3 |

modify to

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 1.344 | 0.344 | 001 |
| 1.112 | 0.241 | 010 |
| 2.196 | 7.100 | 001 |
| 0.811 | 1.344 | 100 |
| 5.124 | 2.344 | 010 |
| 7.196 | 6.344 | 100 |

# Least squares for classification I

Each class described by its own linear model:

$$y_1(\mathbf{x}) = \mathbf{w}_1^T\mathbf{x} + w_{10} = w_{11}x_1 + w_{12}x_1 + w_{10}$$
$$y_2(\mathbf{x}) = \mathbf{w}_2^T\mathbf{x} + w_{20} = w_{21}x_1 + w_{22}x_1 + w_{20}$$

Grouping it together:

$$\mathbf{y}(\mathbf{x}) = \widetilde{\mathbf{W}}^T\widetilde{\mathbf{x}}$$

Writing it out:

$$\mathbf{y}(\mathbf{x}) = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} w_{10} & w_{11} & w_{12} \\ y_{20} & w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}$$

An input $\mathbf{x}$ is assigned for the output for which $\mathbf{y}(\mathbf{x})$ is largest:

The objective is to determine weights matrix $\mathbf{W}$

# Least squares for classification II

Given the training data-set: $\mathcal{D} = \{\mathbf{x}(k), \mathbf{t}(k)\}_{k=1}^{K}$

| $x_1$ | $x_2$ | $t_1$ | $t_2$ |
|---|---|---|---|
| 1.344 | 0.344 | 0 | 1 |
| 1.112 | 0.241 | 1 | 0 |
| 2.196 | 7.100 | 1 | 0 |

Define matrices:

$$\widetilde{\mathbf{X}}^{K\times 3} = \begin{bmatrix} 1 & x_1(1) & x_2(1) \\ 1 & x_1(2) & x_2(2) \\ \vdots & \vdots & \vdots \\ 1 & x_1(K) & x_2(K) \end{bmatrix} \quad \widetilde{\mathbf{W}}^{3\times 2} = \begin{bmatrix} w_{10} & w_{20} \\ w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \quad \mathbf{T}^{K\times 2} = \begin{bmatrix} t_1(1) & t_2(1) \\ t_1(2) & t_2(2) \\ \vdots & \vdots \\ t_1(K) & t_2(K) \end{bmatrix}$$

The error between the output of the model and true classes:

$$e = \widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \mathbf{T}$$

# Least squares for classification III

The sum-of-squares error function:

$$E_D = \frac{1}{2}\mathrm{Tr}\left\{(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \widetilde{\mathbf{T}})^T(\widetilde{\mathbf{X}}\widetilde{\mathbf{W}} - \widetilde{\mathbf{T}})\right\}$$

Error is minimized for:

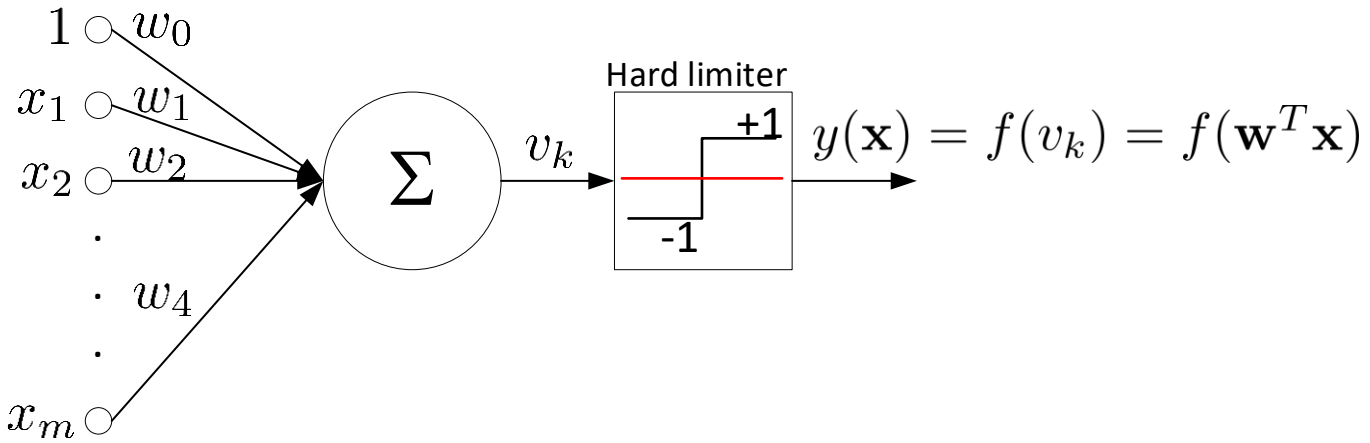
$$\frac{\partial E_D}{\widetilde{\mathbf{W}}} = 0$$

The weight matrix is then given by:

$$\widetilde{\mathbf{W}} = (\widetilde{\mathbf{X}}^T\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^T\mathbf{T}$$

# Rossenblatt perceptron (simplest neural network)

An example of a linear discriminant model:




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

$$y(\mathbf{x}) = f(v_k) = \begin{cases} 1 & : v_k \geq 0 \\ -1 & : v_k < 0 \end{cases}$$

The objective is to determine weights matrix $\mathbf{W}$

# Perceptron algorithm

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 1.344 | 0.344 | -1 |
| 1.112 | 0.241 | 1 |
| 2.196 | 7.100 | 1 |

Input: $\mathbf{X}^{K\times 3} = [\mathbf{x}(1), \mathbf{x}(2), ..., \mathbf{x}(K)]^T$ where: $\mathbf{x}(k) = [1, x_1(k), x_2(k)]$

Output (true classes): $\mathbf{T}^{K\times 1} = [t(1), t(2), ..., t(K)]^T$ where: $t(k) \in \{-1, 1\}$

Weights: $\mathbf{W}^{K\times 3} = [\mathbf{w}(1), \mathbf{w}(2), ..., \mathbf{w}(K)]^T$ where: $\mathbf{w}(k) = [w_0(k), w_1(k), w_2(k)]$

**Algorithm:**

for $k = 1 : K$

$$y(k) = \begin{cases} 1 & : \mathbf{w}(k)\mathbf{x}(k)^T \geq 0 \\ -1 & : \mathbf{w}(k)\mathbf{x}(k)^T < 0 \end{cases}$$

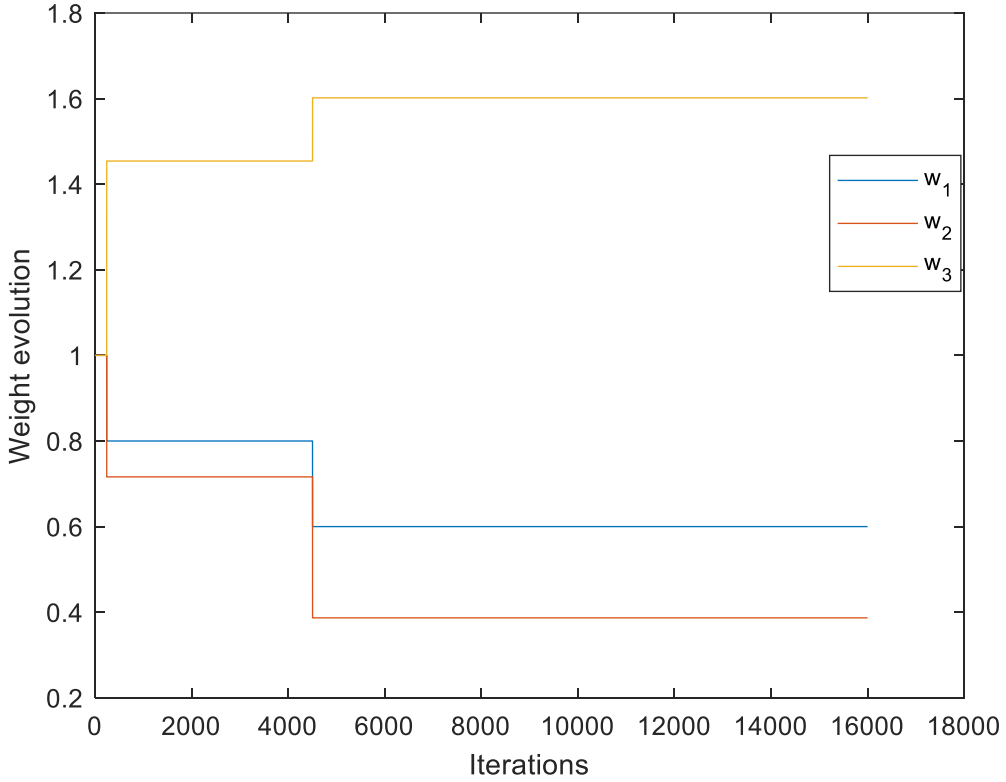
$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta[t(k) - y(k)]\mathbf{x}(k)$$

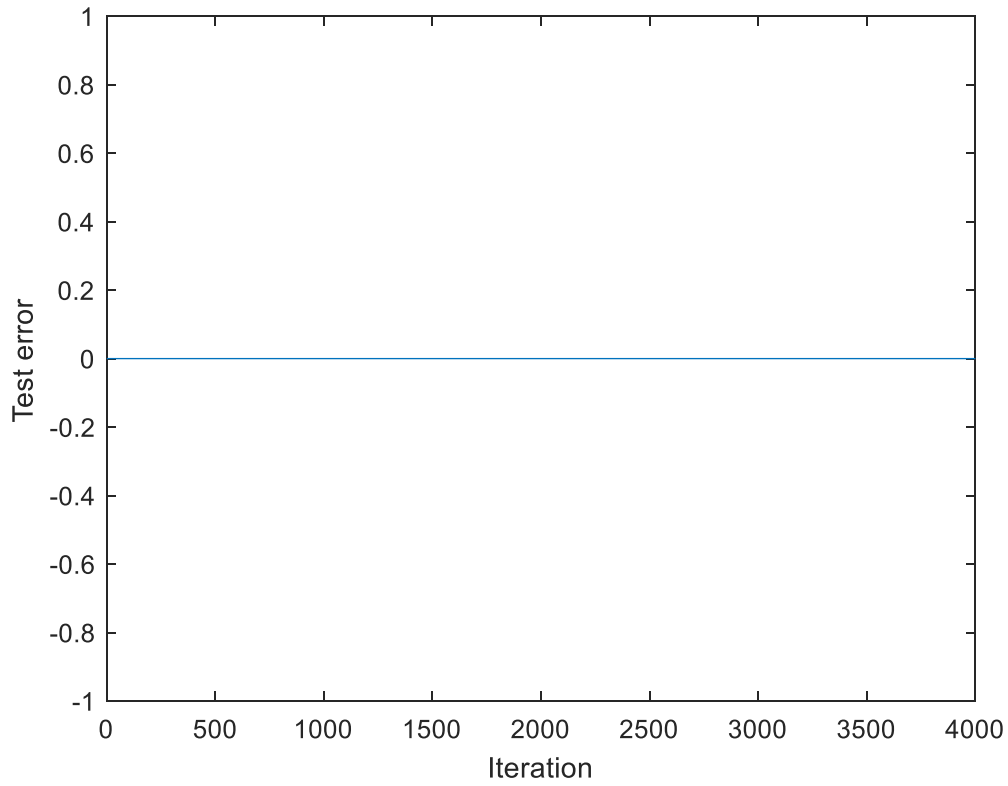
end

Learning rate coefficient: $\eta$

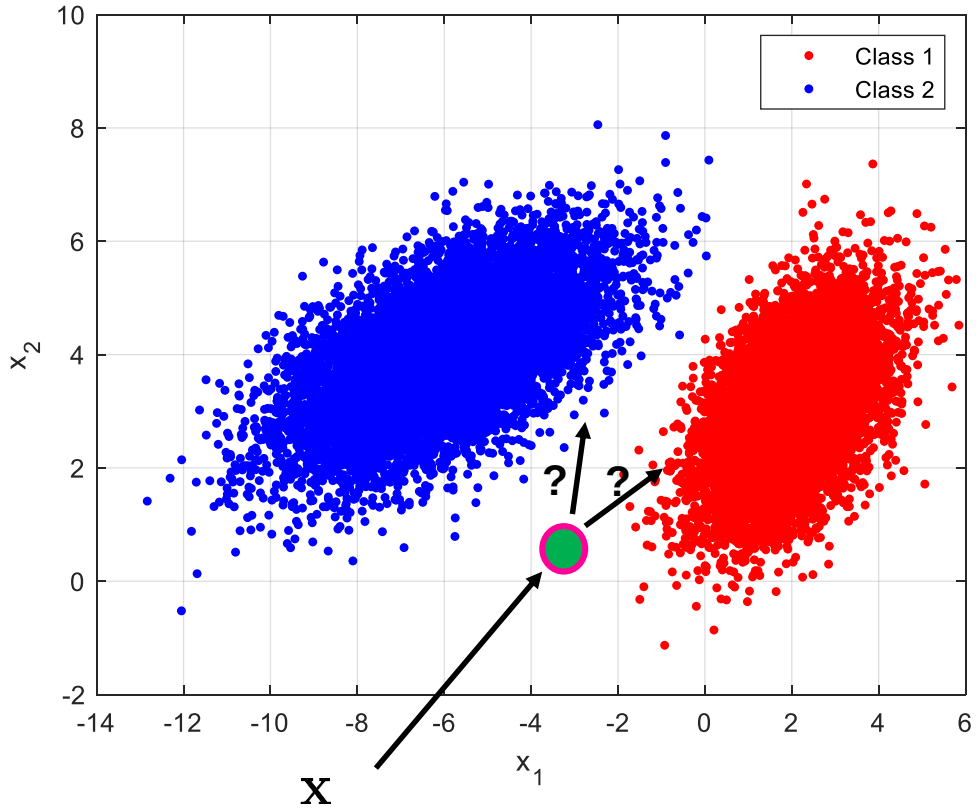
$\mathbf{w}(1) = [0, 0, 0]$

# Results of perceptron algorithm



# Probabilistic generative models




Probability of $\mathbf{x}$ belonging to Class 1, $\mathcal{C}_1$ :

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)+p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$

$$= \frac{1}{1+\exp(-a)} = \sigma(a)$$

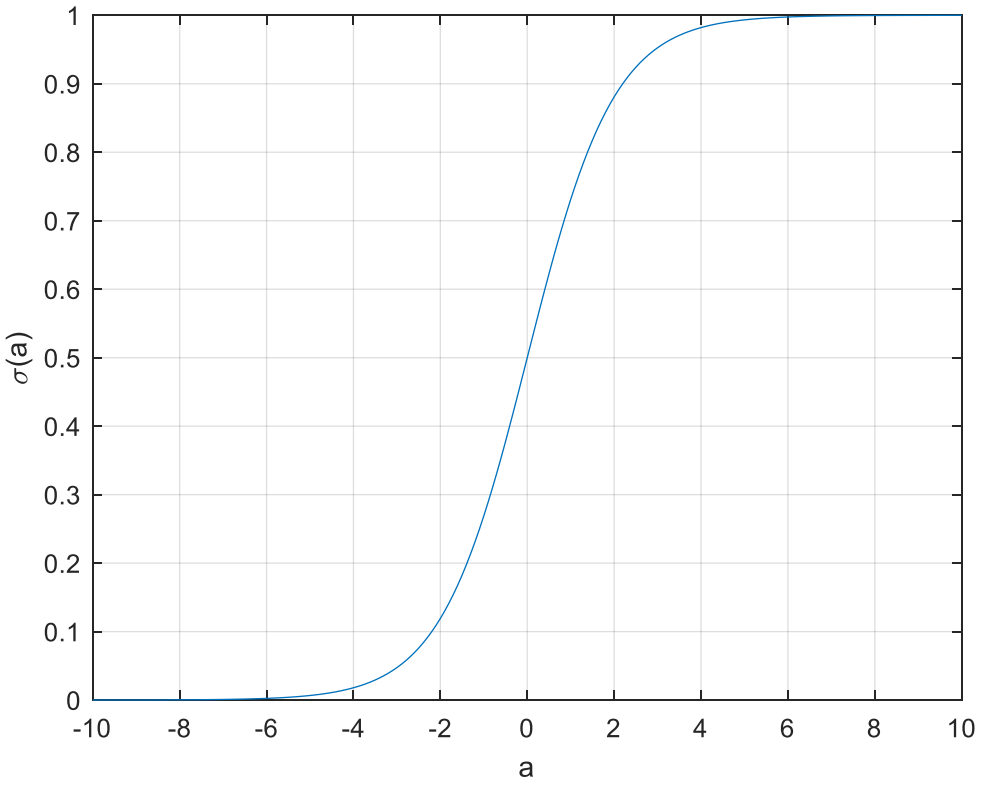
where:

$$a = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)}$$

Probability of $\mathbf{x}$ belonging to Class 2, $\mathcal{C}_2$ :

$$p(\mathcal{C}_2|\mathbf{x}) = 1 - p(\mathcal{C}_1|\mathbf{x})$$

# Logistic sigmoid



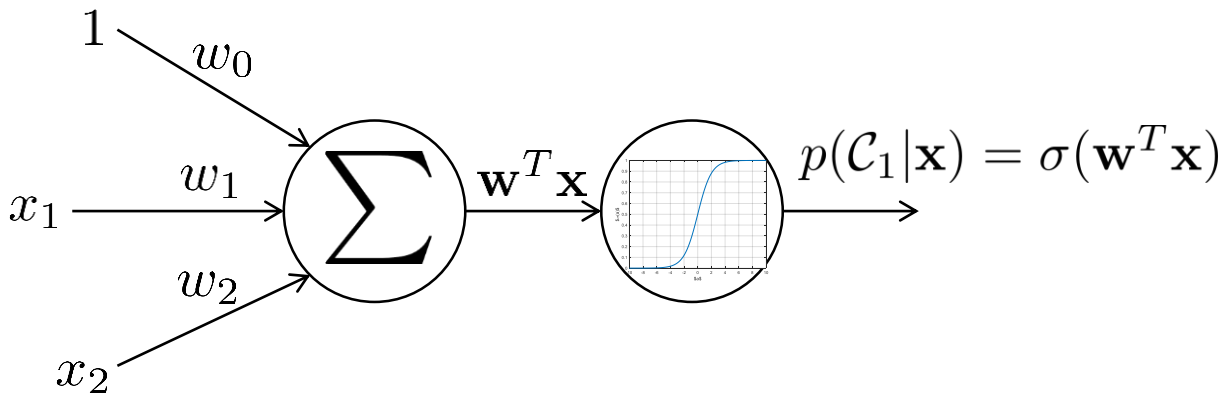

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)+p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$

$$= \frac{1}{1+\exp(-a)} = \sigma(a)$$

Logistic sigmoid functions maps the whole real axis into a finite interval [0;1]

# Logistic regression

| $x_1$ | $x_2$ | $t_1$ |
|---|---|---|
| 1.344 | 0.344 | 0 |
| 1.112 | 0.241 | 1 |
| 2.196 | 7.100 | 1 |




$$p(\mathcal{C}_1|\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \sigma([w_0, w_1, w_2][1, x_1, x_2]^T)$$
$$p(\mathcal{C}_2|\mathbf{x}) = 1 - p(\mathcal{C}_1|\mathbf{x})$$

if $p(\mathcal{C}_1|\mathbf{x}) > p(\mathcal{C}_2|\mathbf{x})$
$\mathbf{x}$ assigned to class 1
else
$\mathbf{x}$ assigned to class 2

# Weights learning rule

For a given data-set: $\{\mathbf{x}_k, t_k\}$ where $t_k \in \{0, 1\}$and $k = 1, ..., K$

$$E(\mathbf{w}) = -\sum_{k=1}^{K} \{t_k \ln y_k + (1 - t_k) \ln(1 - y_k)\}$$

where $y_k = \sigma(\mathbf{w}^T \mathbf{x}_k)$

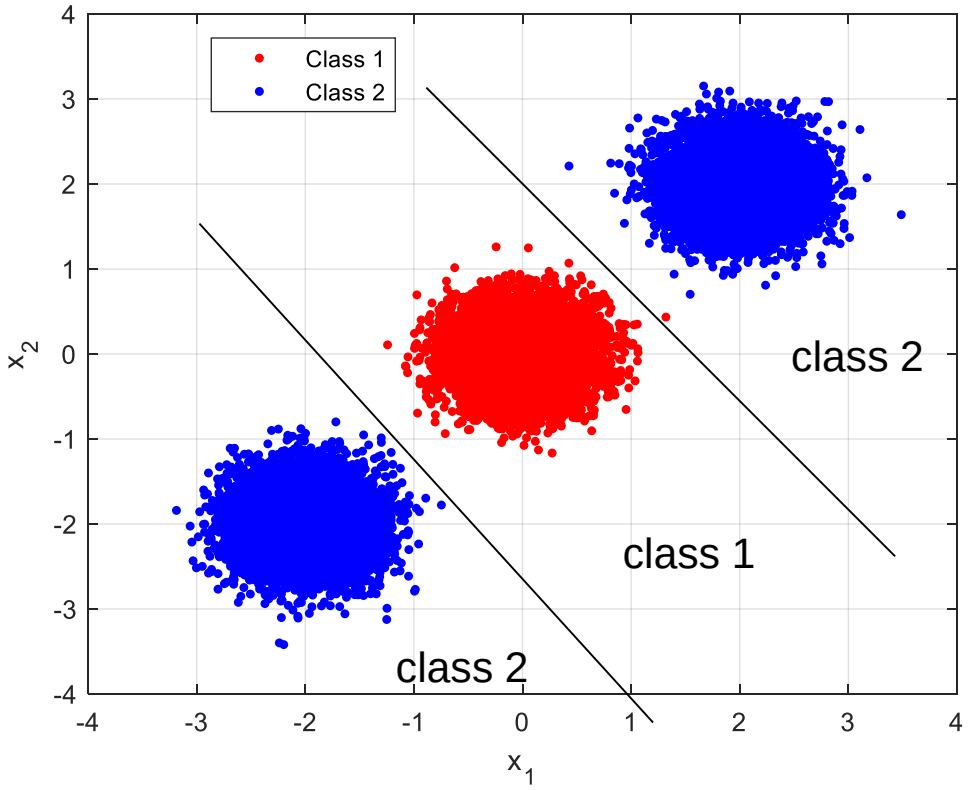
$$\mathbf{w}_{(k+1)} = \mathbf{w}_k - \eta \nabla E(\mathbf{w})$$

Taking the derivative, the update rule becomes:

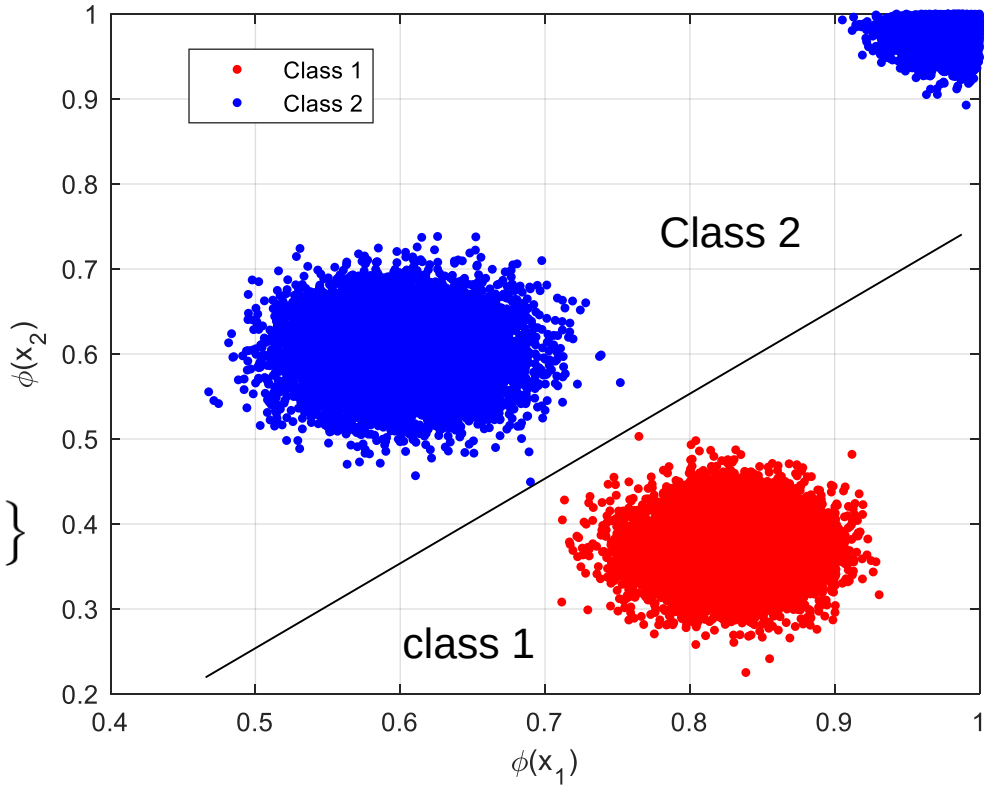
$$\mathbf{w}_{(k+1)} = \mathbf{w}_k - \eta(y_k - t_k)\mathbf{x}_k$$

```
for i = 1 : L_iter
    for k = 1 : L_batch
            G = G +grad(k)
    end
    w(i+1) = w(i) - eta*mean(G);
end
```

# Transformations using basis functions

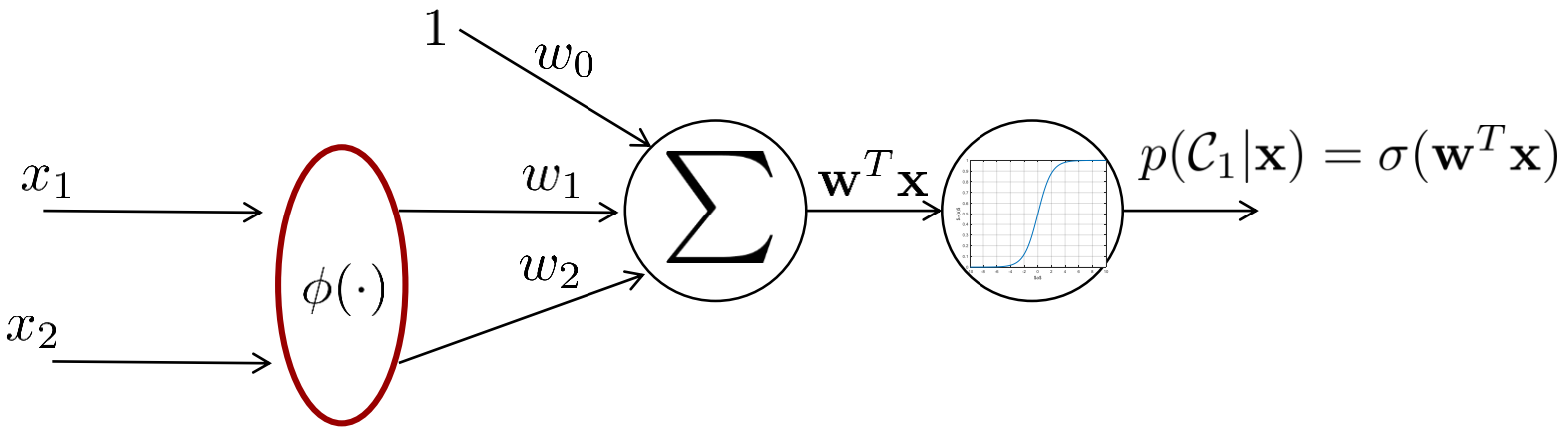



$$\phi(\mathbf{x}) = \exp\left\{ -\frac{(\mathbf{x}-\mu)^2}{2s^2} \right\}$$




Transforming inputs changes *nonlinear* decision boundary to the *linear* one

# Logistic regression with input transformations

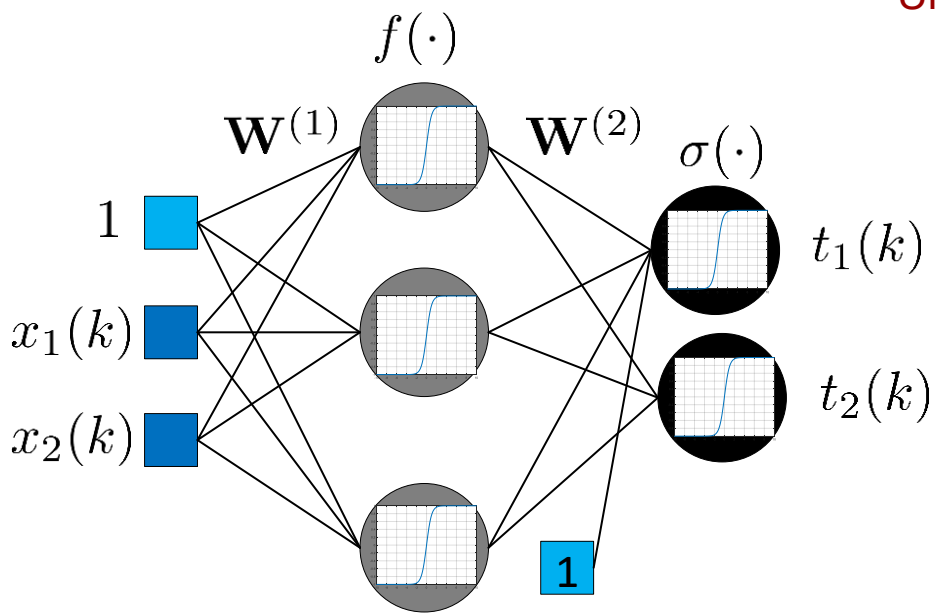



$$p(\mathcal{C}_1|\mathbf{x}) = \sigma(\mathbf{w}^T\phi(\mathbf{x}))$$
$$p(\mathcal{C}_2|\mathbf{x}) = 1 - p(\mathcal{C}_1|\mathbf{x})$$

if $p(\mathcal{C}_1|\mathbf{x}) > p(\mathcal{C}_2|\mathbf{x})$
$\mathbf{x}$ assigned to class 1
else
$\mathbf{x}$ assigned to class 2

# Nonlinear classifier – neural network

| $x_1$ | $x_2$ | $t_1$ | $t_2$ |
|---|---|---|---|
| 1.344 | 0.344 | 0 | 1 |
| 1.112 | 0.241 | 1 | 0 |
| 2.196 | 7.100 | 1 | 0 |




Unknown weight matrices that need to be determined:

$$\mathbf{W}^{(1)} = \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \end{bmatrix}$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} w_{11}^2 & w_{12}^2 & w_{13}^2 & w_{14}^2 \\ w_{21}^2 & w_{22}^2 & w_{23}^2 & w_{24}^2 \end{bmatrix}$$

$$\mathbf{t}(k) = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \sigma(\mathbf{W}^{(2)} f(\mathbf{W}^{(1)} \mathbf{x}(k)))$$

# In this lecture we have learned….

- Difference between classification and regression
- Linear and nonlinear decision boundaries
- Discriminant function
- Logistic regression
- Difference between training and test set
- Least squares for learning decision boundaries
  - Matrix inversion
  - Iterative method
- Neural networks for classification
- Transformation of nonlinear to linear decision boundaries

# Problem solving session

- Generate 3 classes by drawing samples from 2-D Gaussian distribution
- Split the data-set into training and test
- Implement the least squares method for learning weights within discriminant function
- Implement perceptron algorithm for two class classification
- Implement the least squares method and perform logistic regression
- Test you implementation by performing classification on test set