Neural networks for classification

Inteligencia Artificial **3º INF**

Escuela Politécnica Superior

Supervised learning: classification

Machine learning by **induction from labeled data**

$$\mathcal{D} = \{(\mathbf{x}_n, c_n)\}_{n=1}^{N}$$

$\mathbf{x} \in \mathcal{X}$: vector of **attributes**, features, independent variables, input variables. covariates…

Bias term $x_{n0} = 1$

$$\mathbf{x}_n^T = (x_{n0}\ x_{n1}\ \ x_{n2}\ \dots x_{nD})\ \ [(D+1)\text{-dimensional vector}]$$

$c \in \mathcal{C}$: (class) **label**, dependent variable, outcome, target, …

Classification: **Labels** c are **discrete** $(e.g.\ c \in \{C_1, \dots, C_K\})$

Learning algorithm: $\mathcal{L}$

w: Model parameters

$$\mathcal{L}\colon \mathcal{D} = \{(\mathbf{x}_n, c_n)\}_{n=1}^{N} \to h(\cdot\,; \mathbf{w})$$

$$h(\cdot\,;\mathbf{w})\colon\ \ \mathbf{x} \in \mathcal{X} \ \to\ h(\mathbf{x};\mathbf{w}) \in \{C_1, \dots, C_K\}$$

Binary classification

n	x_{n1}	x_{n2}	...	x_{nD}	c_n
1	2.3	0	...	10.3	C_0
2	2.5	1	...	13.1	C_1
3	2.6	0	...	-2.7	C_1
4	2.7	-1	...	-5.4	C_0
5	2.9	0	...	2.1	C_1
6	3.1	0	...	-10.9	C_0

Binary classification

0-1 encoding of the **labeled data**

$$\mathcal{D} = \{(\mathbf{x}_n, c_n)\}_{n=1}^{N} \Rightarrow \mathcal{D} = \{(\mathbf{x}_n, t_n)\}_{n=1}^{N};\ \ t_n = \begin{cases} 0 & \text{if } c_n = C_0 \\ 1 & \text{if } c_n = C_1 \end{cases}$$

$$\mathcal{L}\colon \mathcal{D} = \{(\mathbf{x}_n, c_n)\}_{n=1}^{N} \to o(\,\cdot\,;\mathbf{w})$$

$$z(\,\cdot\,;\mathbf{w})\colon \quad \mathbf{x} \in \mathcal{X} \to z(\mathbf{x};\mathbf{w}) \in \mathbb{R}$$

$$o(\mathbf{x};\mathbf{w}) = \varphi^{(o)}\big(z(\mathbf{x};\mathbf{w})\big) \in [0,1]$$

- The output $o(\mathbf{x};\mathbf{w})$ is an estimate of the class C_1 posterior

Discriminative model

$$\hat{p}(C_1|\mathbf{x},\mathbf{w}) = \varphi^{(o)}\big(z(\mathbf{x};\mathbf{w})\big)$$

Logistic regression: $\varphi^{(o)}(z) = \sigma(z) = \frac{1}{1+e^{-z}}$

Probit regression: $\varphi^{(o)}(z) = \Phi(z) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{z} e^{-\frac{y^2}{2}}dy$

Binary classification: 0-1 encoding

n	x_{n1}	x_{n2}	...	x_{nD}	t_n
1	2.3	0	...	10.3	0
2	2.5	1	...	13.1	1
3	2.6	0	...	-2.7	1
4	2.7	-1	...	-5.4	0
5	2.9	0	...	2.1	1
6	3.1	0	...	-10.9	0

From posteriors to decisions

- To output the class label, we compare the posterior of C_1 with some threshold in the interval [0,1]

$$h(\mathbf{x};\mathbf{w}) = \begin{cases} C_1 & \text{if} \;\; p(C_1|\mathbf{x},\mathbf{w}) \geq \text{threshold} \\ C_0 & \text{if} \;\; p(C_1|\mathbf{x},\mathbf{w}) < \text{threshold} \end{cases}$$

- If misclassification costs are equal:
  - Error if prediction is C_1: $p(C_0|\mathbf{x},\mathbf{w})$
  - Error if prediction is C_0: $p(C_1|\mathbf{x},\mathbf{w})$

  Predict

$$C_1 \text{ when } p(C_0|\mathbf{x},\mathbf{w}) \leq p(C_1|\mathbf{x},\mathbf{w}) \Rightarrow p(C_1|\mathbf{x},\mathbf{w}) \geq 1/2$$

$$C_0 \text{ otherwise} \Rightarrow p(C_1|\mathbf{x},\mathbf{w}) < 1/2$$

Optimal (minimal error) classification rule

Optimal threshold

Unequal classification costs

Cost of classifying an example of class C_0 as class C_1

- Cost if prediction is C_1: $p(C_0|\mathbf{x},\mathbf{w})cost_{10} + p(C_1|\mathbf{x},\mathbf{w})cost_{11}$
- Cost if prediction is C_0: $p(C_0|\mathbf{x},\mathbf{w})cost_{00} + p(C_1|\mathbf{x},\mathbf{w})cost_{01}$

C_1 prediction cost = C_0 prediction cost

Cost of correctly classifying a class C_0 example (typically, 0)

Cost of classifying an example of class C_1 as class C_0

Minimal cost classification rule:

Predict

$$C_1 \text{ when } p(C_1|\mathbf{x},\mathbf{w}) \geq \text{threshold}^*$$

$$C_0 \text{ when } p(C_1|\mathbf{x},\mathbf{w}) < \text{threshold}^*$$

Optimal threshold

E.g. If $cost_{10} - cost_{00} = 2(cost_{01} - cost_{11})$
threshold$^* = 2/3$ (it is less likely to predict C_1)

$$\text{threshold}^* = \frac{cost_{10} - cost_{00}}{cost_{10} - cost_{00} + cost_{01} - cost_{11}}$$

Learning by maximum lihelihood

Samples assumed to be iid (independent identically distributed)

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

- Likelihood of the model, given $\mathcal{D} = \{(\mathbf{x}_n, t_n)\}_{n=1}^N$

$$\mathcal{L}(\mathbf{w}) = \hat{P}\left(\{t_n\}_{n=1}^N \middle| \{\mathbf{x}_n\}_{n=1}^N, \mathbf{w}\right) = \prod_{n=1}^N \hat{p}(t_n|\mathbf{x}_n, \mathbf{w}) =$$

Factorizes because samples are assumed to be independent

$$= \prod_{n=1}^N \left(\hat{p}(C_0|\mathbf{x}_n, \mathbf{w})\right)^{1-t_n} \left(\hat{p}(C_1|\mathbf{x}_n, \mathbf{w})\right)^{t_n}$$

Estimate of class posterior

Same distribution: samples are assumed to be identically dist.

- Log-likelihood function

$$\log \mathcal{L}(\mathbf{w}) = \sum_{n=1}^N \left((1 - t_n)\log \hat{p}(C_0|\mathbf{x}_n, \mathbf{w}) + t_n \log \hat{p}(C_1|\mathbf{x}_n, \mathbf{w})\right)$$

Same maximizer as $\mathcal{L}(\mathbf{w})$ (log is monotone)

Minimization of the cross-entropy error

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} CE(\mathbf{w})$$

Same as maximizer of $\mathcal{L}(\mathbf{w})$

- Cross-entropy error

$$\begin{aligned} CE(\mathbf{w}) &= -\log \mathcal{L}(\mathbf{w}) \\ &= -\sum_{n=1}^{N} \big((1-t_n)\log \hat{p}(C_0|\mathbf{x}_n,\mathbf{w}) + t_n \log \hat{p}(C_1|\mathbf{x}_n,\mathbf{w})\big) \\ &= -\sum_{n=1}^{N} \Big((1-t_n)\log\big(1-\hat{p}(C_1|\mathbf{x}_n,\mathbf{w})\big) + t_n \log \hat{p}(C_1|\mathbf{x}_n,\mathbf{w})\Big) \end{aligned}$$

- Discriminative model: $\hat{p}(C_1|\mathbf{x},\mathbf{w}) = \varphi^{(o)}\big(z(\mathbf{x}_n;\mathbf{w})\big)$

$\hat{p}(C_1|\mathbf{x},\mathbf{w})$

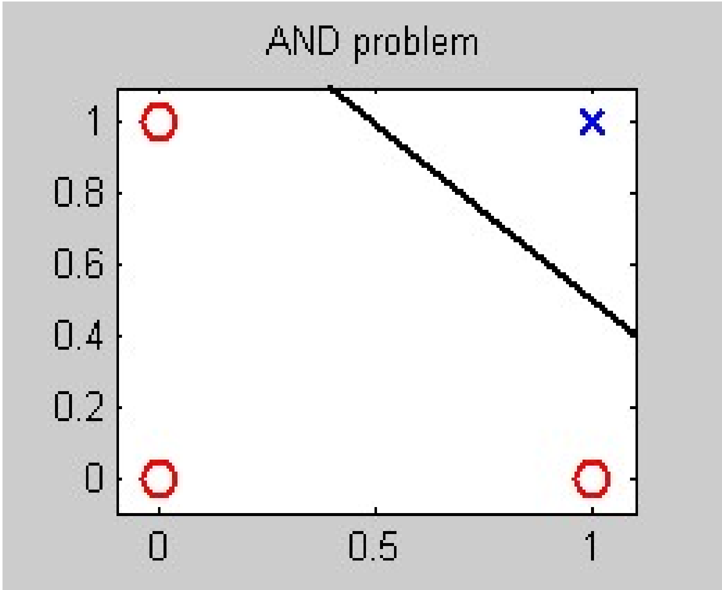
$$CE(\mathbf{w}) = -\sum_{n=1}^{N} \bigg((1-t_n)\log\Big(1-\varphi^{(o)}\big(z(\mathbf{x}_n;\mathbf{w})\big)\Big) + t_n \log\Big(\varphi^{(o)}\big(z(\mathbf{x}_n;\mathbf{w})\big)\Big)\bigg)$$

$\hat{p}(C_0|\mathbf{x},\mathbf{w})$

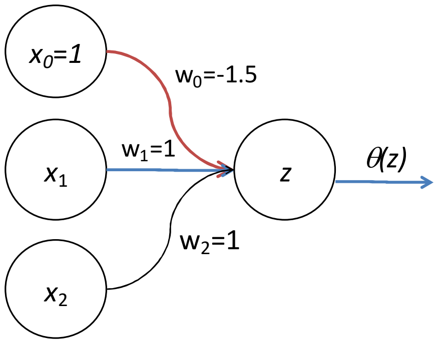
AND: A linearly separable problem

- The AND classification problem

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1



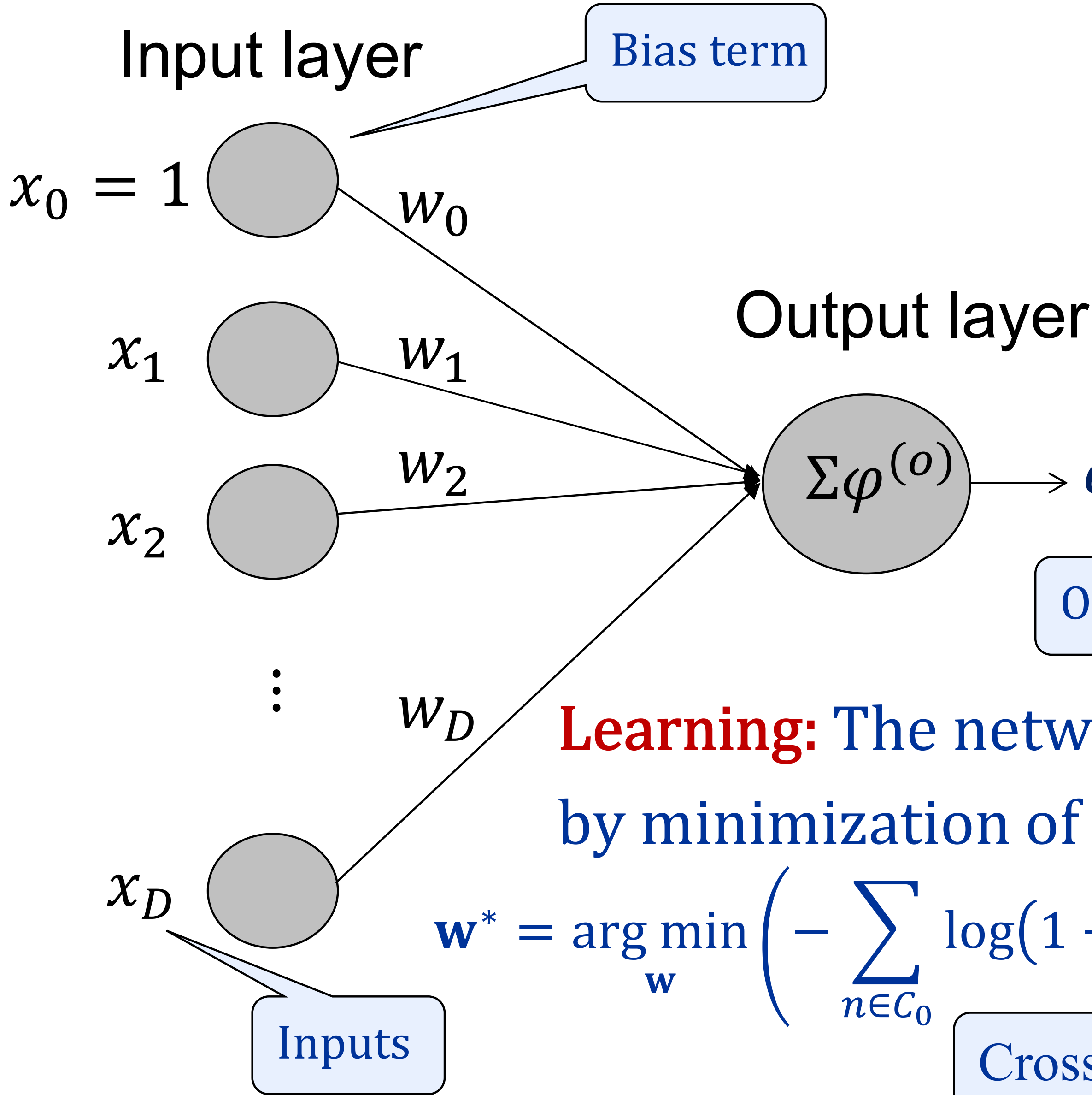
- Single-layer perceptron



x_1	x_2	$z = w_0x_0 + w_1x_1 + w_1x_2$	$h(\mathbf{x}) = \theta(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0.5 & \text{if } z = 0 \\ 0 & \text{if } z < 0 \end{cases}$
0	0	-1.5	0
0	1	-0.5	0
1	0	-0.5	0
1	1	0.5	1

Single layer perceptron

Can solve only linearly separable problems



$\mathbf{w}^{\mathrm{T}} = (w_0, w_1, \dots, w_D)$
$\mathbf{x}^{\mathrm{T}} = (x_0\ x_1 \dots x_D)$

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} = \sum_{d=0}^{D} w_d x_d$$

$$o(\mathbf{x};\mathbf{w}) = \varphi^{(o)}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right)$$

Output is an estimate of $p(C_1|\mathbf{x})$

Learning: The network weights are determined by minimization of a cost function. For instance,

$$\mathbf{w}^* = \arg\min_{\mathbf{w}}\left(-\sum_{n\in C_0}\log\left(1 - o(\mathbf{x}_n;\mathbf{w})\right) - \sum_{n\in C_1}\log\left(o(\mathbf{x}_n;\mathbf{w})\right)\right)$$

Cross-entropy error

Logistic regression

- Class posteriors

$$p(C_1|\mathbf{x},\mathbf{w}) = \frac{1}{1+e^{-z(\mathbf{x};\mathbf{w})}}$$

$$p(C_0|\mathbf{x},\mathbf{w}) = 1 - p(C_1|\mathbf{x},\mathbf{w}) = \frac{e^{-z(\mathbf{x};\mathbf{w})}}{1+e^{-z(\mathbf{x};\mathbf{w})}} = \frac{1}{1+e^{z(\mathbf{x};\mathbf{w})}}$$

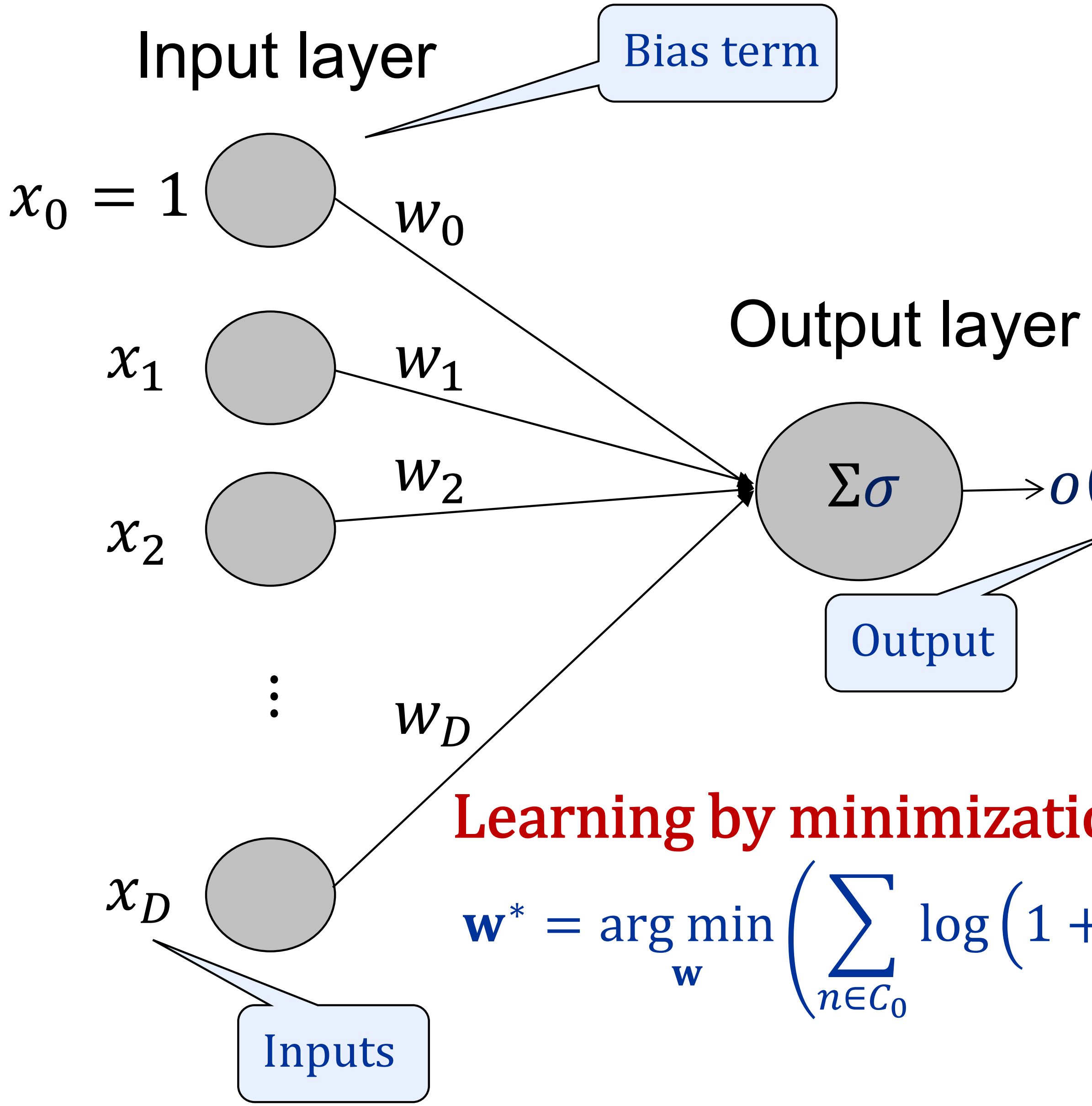
$\mathbf{w}^{\mathrm{T}} = (w_0, w_1, \dots, w_D)$
$\mathbf{x}^{\mathrm{T}} = (x_0\ x_1 \dots x_D)$

- Log-odds: $\mathrm{z}(\mathbf{x};\mathbf{w}) = \log\frac{p(C_1|\mathbf{x},\mathbf{w})}{p(C_0|\mathbf{x},\mathbf{w})}$

- Linear model for log-odds: $z(\mathbf{x};\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} = \sum_{d=0}^{D} w_d x_d$

Single-layer perceptron!

Single layer perceptron: logistic regression



$$\mathbf{w}^{\mathrm{T}} = (w_0, w_1, \ldots, w_D)$$
$$\mathbf{x}^{\mathrm{T}} = (x_0\ x_1 \ldots x_D)$$
$$\mathbf{w}^{\mathrm{T}}\mathbf{x} = \sum_{d=0}^{D} w_d x_d$$

$$o(\mathbf{x}; \mathbf{w}) = \sigma\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}\right) = \frac{1}{1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}}}$$

Learning by minimization of the cross-entropy error:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \left(\sum_{n \in C_0} \log\left(1 + e^{\mathbf{w}^{\mathrm{T}}\mathbf{x}_n}\right) + \sum_{n \in C_1} \log\left(1 + e^{-\mathbf{w}^{\mathrm{T}}\mathbf{x}_n}\right) \right)$$

Cross entropy error for logistic regression

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} CE(\mathbf{w})$$

$$\frac{\partial \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n)}{\partial \mathbf{w}} = \mathbf{x}_n \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n)\left(1 - \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n)\right)$$

- Cross-entropy error

$$CE(\mathbf{w}) = -\sum_{n=1}^{N} \left((1 - t_n) \log\left(1 - \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n)\right) + t_n \log \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n) \right)$$

- Gradient of the cross-entropy error

$$\frac{\partial}{\partial \mathbf{w}} CE(\mathbf{w}) = \sum_{n=1}^{N} \left((1 - t_n)\mathbf{x}_n \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n) - t_n \mathbf{x}_n \left(1 - \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n)\right) \right)$$

$$\delta_n = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n) - t_n$$

$$= \sum_{n=1}^{N} \left(\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n) - t_n\right) \mathbf{x}_n = \sum_{n=1}^{N} \delta_n \, \mathbf{x}_n$$

Single-layer perceptron: Batch learning

INPUT: Training instances: $\mathcal{D} = \{(\mathbf{x}_n, t_n)\}_{n=1}^{N}$

Learning parameter: $\eta > 0$

OUTPUT: $\mathbf{w}^* = \arg\min_{\mathbf{w}} CE(\mathbf{w})$

Attributes should be scaled!

1. Randomly initialize $\mathbf{w} \sim U[-0.5,0.5]^{(D+1)}$

2. $n_{epoch} = 0$

2. While convergence criteria are not met

Similar to Rosenblatt!
$\delta_n = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n) - t_n$
prediction error

2.1 Increment epoch counter: $n_{epoch} = n_{epoch} + 1$

2.2 Calculate the network outputs: $\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n)$; $n = 1, \dots, N$

2.2 Calculate the gradient: $\frac{\partial}{\partial \mathbf{w}} CE(\mathbf{w}) = \sum_{n=1}^{N} \delta_n \, \mathbf{x}_n$

2.3 Update weights: $\mathbf{w} = \mathbf{w} - \eta \sum_{n=1}^{N} \delta_n \, \mathbf{x}_n$

Single-layer perceptron: Online learning

INPUT: Training instances: $\mathcal{D} = \{(\mathbf{x}_n, t_n)\}_{n=1}^{N}$

Learning parameter: $\eta > 0$

OUTPUT: $\mathbf{w}^* = \arg\min_{\mathbf{w}} CE(\mathbf{w})$

1. Randomly initialize $\mathbf{w} \sim U[-0.5,0.5]^{(D+1)}$
2. $n_{epoch} = 0$
3. While convergence criteria are not met

    3.1 Increment epoch counter: $n_{epoch} = n_{epoch} + 1$

    3.2 For $n = 1, \ldots, N$

    Calculate the network output: $\sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n)$

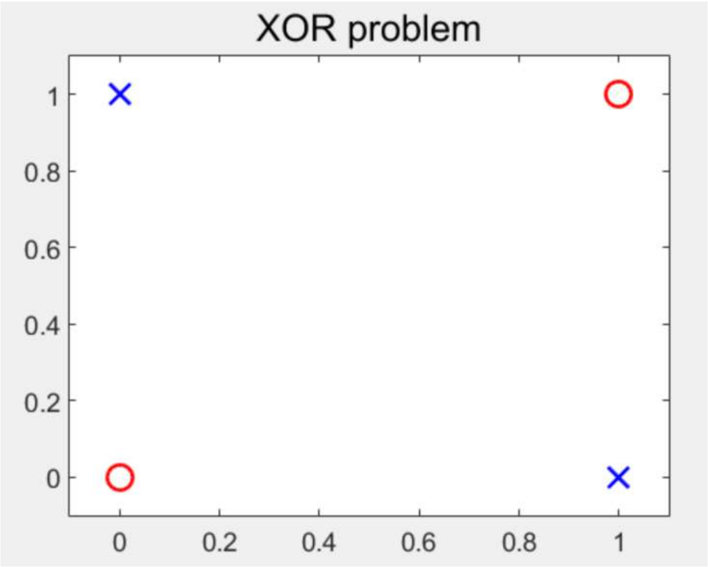
    Calculate prediction error: $\delta_n = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n) - t_n$

    Update weights: $\mathbf{w} = \mathbf{w} - \eta\delta_n\,\mathbf{x}_n$

XOR: A non-linearly separable problem

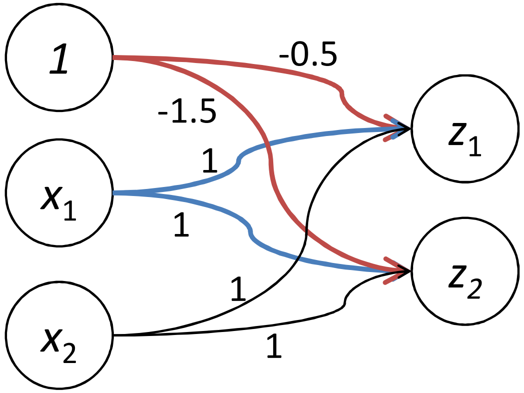
- The single-layer perceptron can only address problems that are linearly separable
- Therefore, the simple XOR problem, which is not linearly separable, cannot be solved using this learning machine.

x_1	x_2	t
0	0	1
0	1	0
1	0	0
1	1	1

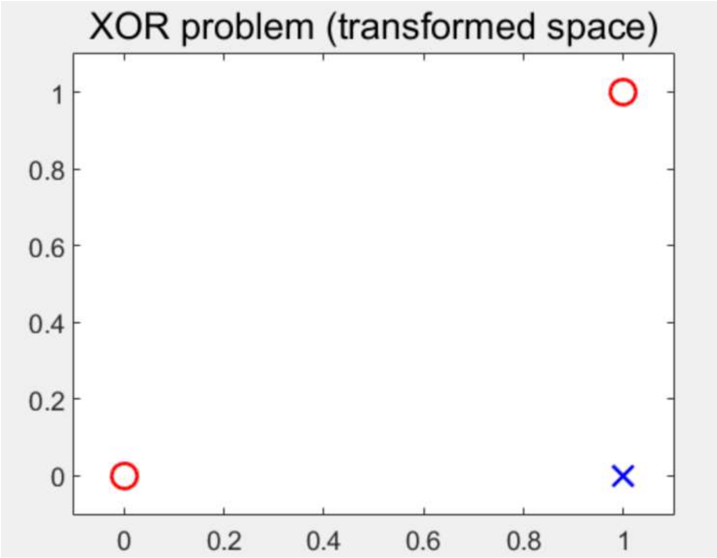


XOR: Non-linear feature construction

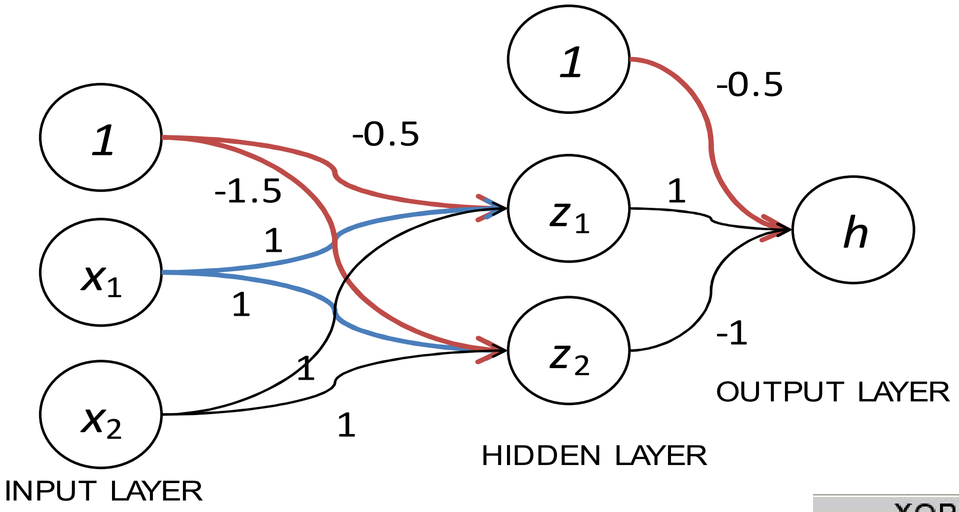
- Consider the XOR problem in a transformed feature space



x_1	x_2	z_1	z_2	t
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0



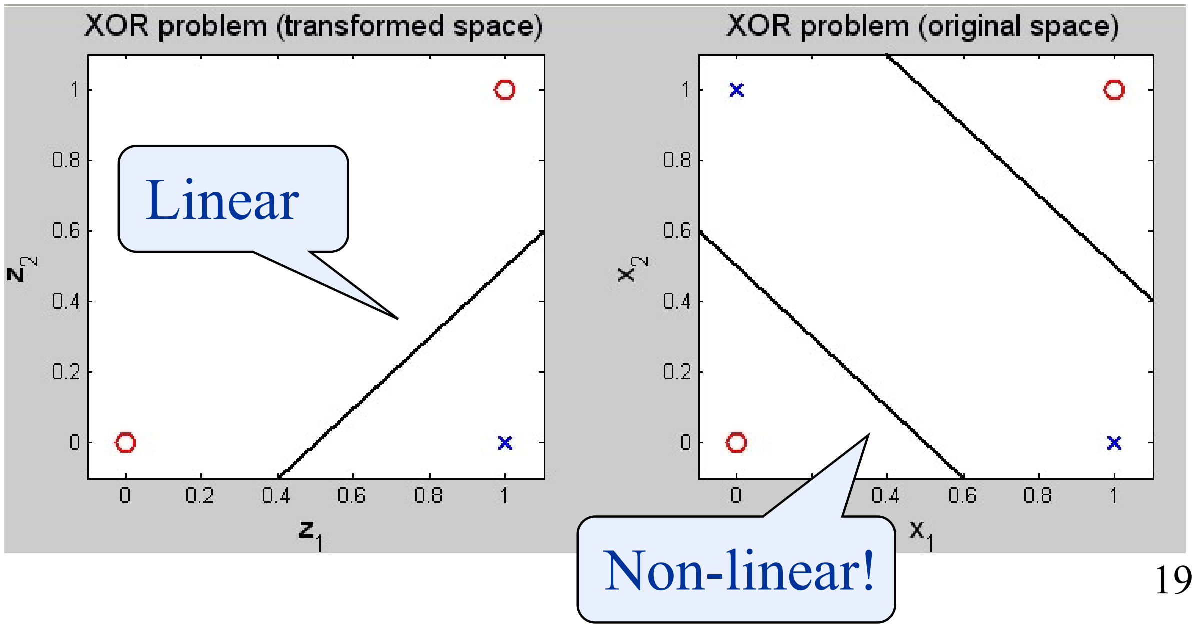
XOR: A linear model in feature space



$$\begin{cases} z_1 = \theta(x_1 + x_2 - 0.5) \\ z_2 = \theta(x_1 + x_2 - 1.5) \end{cases}$$

$$h = \theta(z_1 - z_2 - 0.5)$$

x_1	x_2	z_1	z_2	h
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0



Multi-layer perceptron

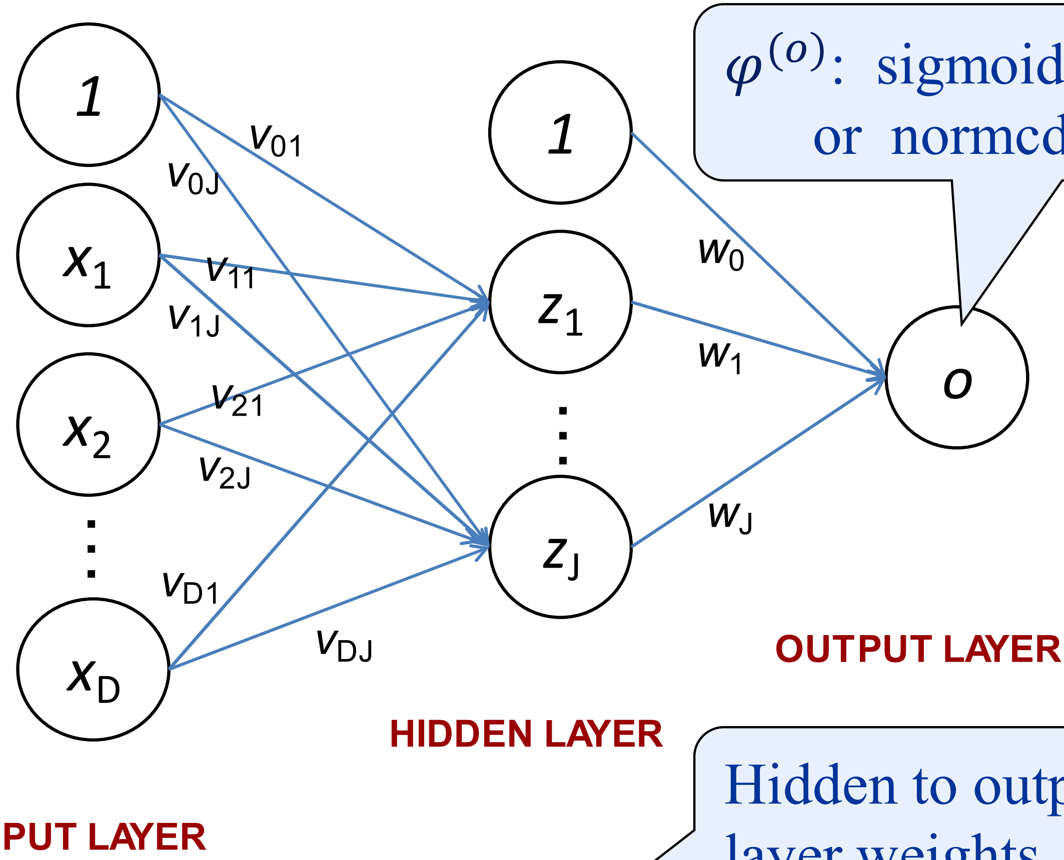
$$\mathbf{x} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_D \end{pmatrix}$$

Attribute (input) vector

input to hidden layer weights

$$\mathbf{V} = (\mathbf{v}_1 \;\; \mathbf{v}_2 \ldots \mathbf{v}_J)$$

$$\mathbf{v}_j^{\mathrm{T}} = (\mathrm{v}_{0j}\; \mathrm{v}_{1j} \ldots \mathrm{v}_{Dj});$$



$\varphi^{(o)}$: sigmoidal or normcdf

$$\mathbf{z} = \begin{pmatrix} 1 \\ z_1 \\ \vdots \\ z_J \end{pmatrix};$$

Local field

Hidden to output layer weights

$$z_j = \phi_j^{(H)}\left(\mathbf{v}_j^{\mathrm{T}}\mathbf{x}\right); \qquad j = 1, 2, \ldots, J; \qquad \mathbf{w}^{\mathrm{T}} = (\mathrm{w}_0\; \mathrm{w}_1 \ldots \mathrm{w}_D);$$

Activation function

$$o(\mathbf{x}; \mathbf{w}) = \varphi^{(o)}\left(\mathbf{w}^{\mathrm{T}}\mathbf{z}\right) = \varphi^{(o)}\left(\sum_{j=0}^{J} w_j\, \phi_j^{(H)}\left(\mathbf{v}_j^{\mathrm{T}}\mathbf{x}\right)\right)$$

Sigmoid (logit) activation function

Also: transfer function

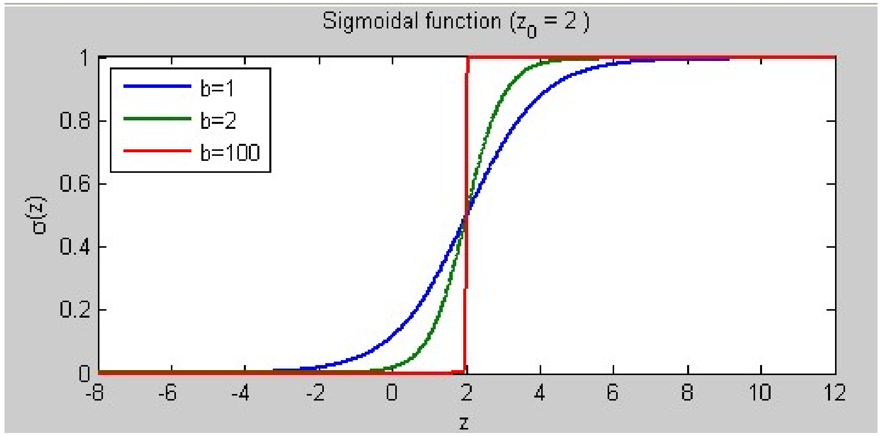
$$\sigma(z) = \frac{1}{1+e^{-z}};$$

- $\sigma(-\infty) = 0; \quad \sigma(0) = \frac{1}{2}; \sigma(+\infty) = 1;$
- Monotonically increasing: $z_2 > z_1 \Rightarrow \sigma(z_2) > \sigma(z_1)$
- Symmetry: $\sigma(-z) = 1 - \sigma(z)$
- Derivative: $\sigma'(z) = \frac{d\sigma(z)}{dz} = \sigma(z)\big(1-\sigma(z)\big)$

$$\sigma(z; z_0, b) = \frac{1}{1+e^{-b(z-z_0)}}$$

$\frac{1}{b}$: scale

z_0: center



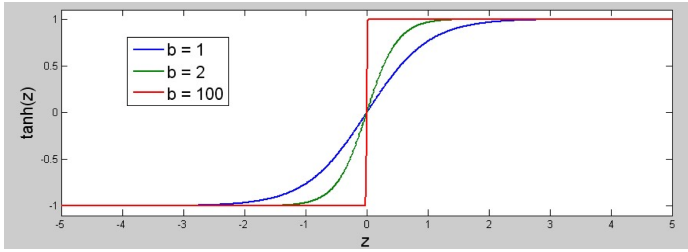
$$\lim_{b\to\infty} \sigma(z; z_0, b) = \begin{cases} 0 & \text{if } \ z < z_0 \\ \frac{1}{2} & \text{if } \ z = z_0 \\ 1 & \text{if } \ z > z_0 \end{cases}$$

Heavyside step function

Other activation functions

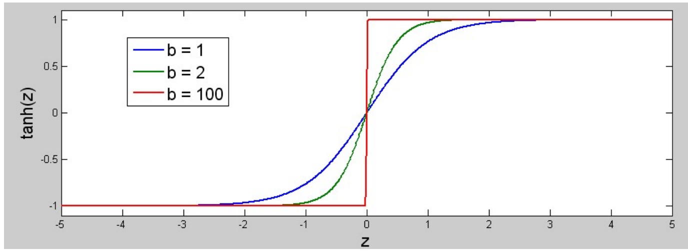
In regression output layer

- **Linear**: $\varphi(z) = z$

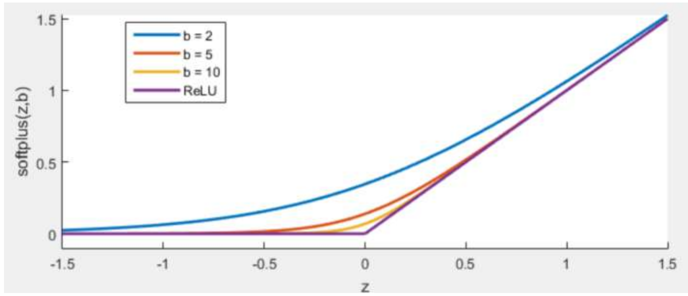


- **Hyperbolic tangent**: $\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

normcdf



- **Probit**: $\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-\frac{y^2}{2}} dy$



- **softplus**: $(z; z_0, b) = \frac{log\left(1+e^{-b(z-z_0)}\right)}{b}$

$$\lim_{b \to \infty} \text{softplus}(z; 0, b) = \text{ReLU}(z)$$

- **Rectified linear unit**: $\text{ReLU}(z) = \max(0, z)$

E.g. $a = 0.01$

- **Leaky rectified linear unit**: $\text{Leaky ReLU}(z) = \begin{cases} az & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}; \quad 0 < a \ll 1$

Avoids zero gradient if $z < 0$

Multi-layer perceptron: Online learning

INPUT: Training instances: $\mathcal{D} = \{(\mathbf{x}_n, t_n)\}_{n=1}^{N}$

Learning parameter: $\eta > 0$

OUTPUT: $\mathbf{V}^*, \mathbf{w}^* = \arg\min_{\mathbf{V},\mathbf{w}} CE(\mathbf{V}, \mathbf{w})$

Only one hidden layer.
Sigmoid activations.

1. Randomly initialize $\mathbf{V}, \mathbf{w} \sim U[-0.5, 0.5]^{(D+1)}$
2. $n_{epoch} = 0$
3. While convergence criteria are not met

    3.1 Increment epoch counter: $n_{epoch} = n_{epoch} + 1$

    3.2 For $n = 1, \ldots, N$

$z_{nj} = \sigma\left(\mathbf{v}_j^{\mathrm{T}} \mathbf{x}_n\right), \quad j = 1, 2, \ldots, J;$ **# forward propagation**

$o_n = \sigma\left(\mathbf{w}^{\mathrm{T}} \mathbf{z}_n\right)$ # network output

$\delta_n = o_n - t_n;$ # prediction error

$\mathbf{w} = \mathbf{w} - \eta \delta_n \mathbf{z}_n$ # weight update

$\Delta_{nj} = z_{nj}(1 - z_{nj})\delta_n, \quad j = 1, 2, \ldots, J;$ **# error backpropagation**

$\mathbf{v}_j = \mathbf{v}_j - \eta \Delta_{nj} \mathbf{x}_n$ # weight update

Universal approximation property

THEOREM: "A feed-forward network with a **single hidden layer** containing a **sufficiently large number of hidden neurons** can uniformly **approximate any continuous function** on compact subsets of $\mathbb{R}^D$, under mild conditions on the activation function"

- GOOD NEWS:

  A neural network can be used to make optimal predictions!

- NOT SO GOOD NEWS:

  How does one find such a network?

  - How many neurons do we need?
  - How easy is it to learn the network parameters (weights)?

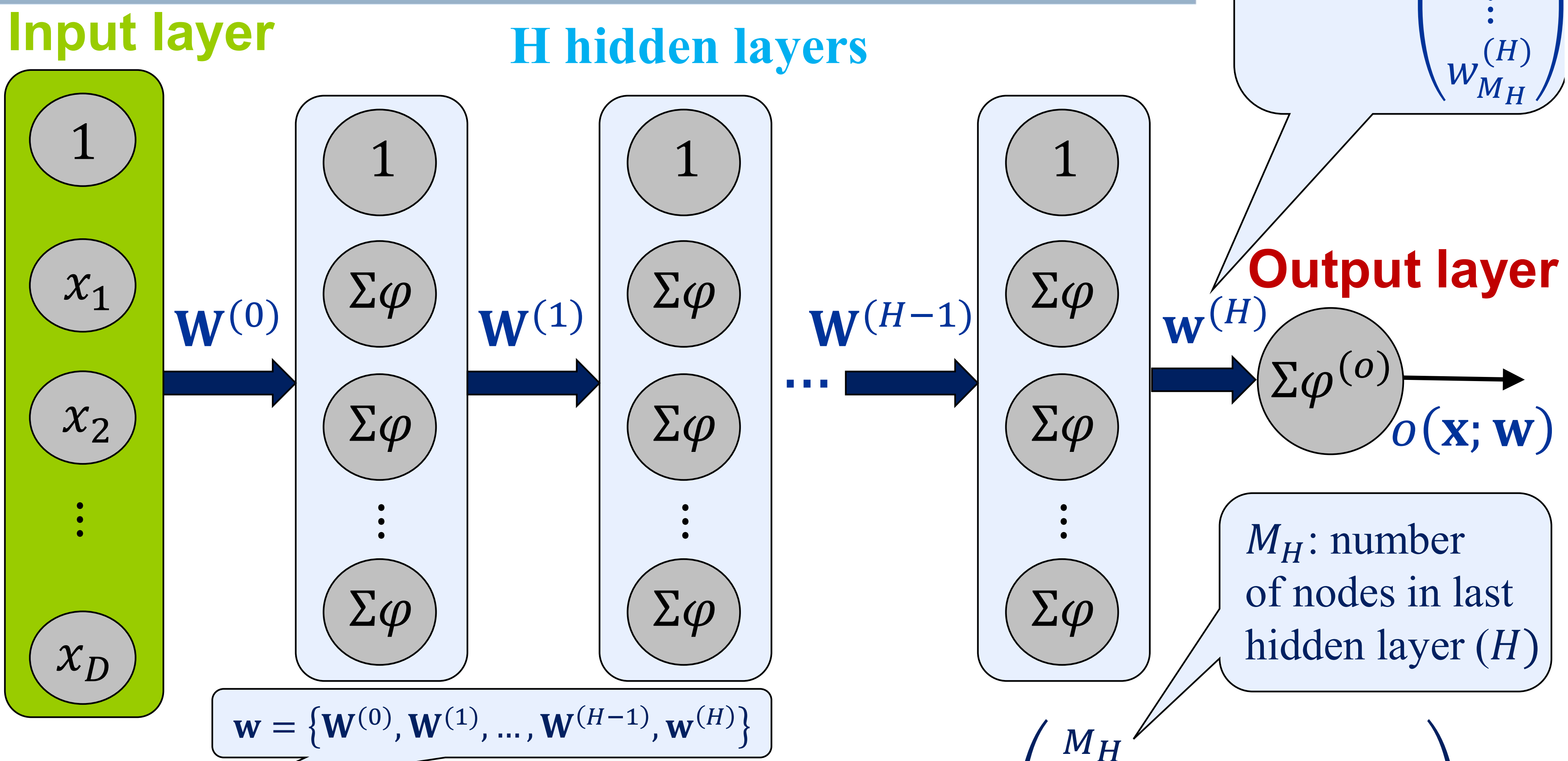
Use CV

Having more than one layer can help (deep networks)

K. Hornik, M. Stinchcombe and H. White, "Multi-layer Feedforward Networks are Universal Approximators," Neural Networks, Vol. 2, pp. 359-366 (1989).

Multilayer perceptron: classification

$$\mathbf{w}^{(H)} = \begin{pmatrix} w_0^{(H)} \\ w_1^{(H)} \\ \vdots \\ w_{M_H}^{(H)} \end{pmatrix}$$

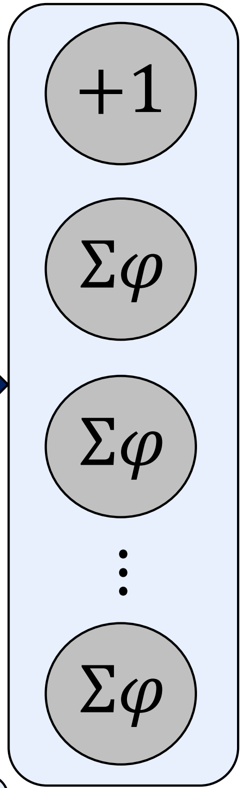


$$o(\mathbf{x};\mathbf{w}) = \varphi^{(o)}\left(\left(\mathbf{w}^{(H)}\right)^T \boldsymbol{\phi}^{(H)}(\mathbf{x})\right) = \varphi^{(o)}\left(\sum_{m=0}^{M_H} w_m^{(H)} \phi_m^{(H)}(\mathbf{x})\right)$$

Output of hidden layer h

$$\mathbf{W}^{(h-1)} = \left(\mathbf{w}_1^{(h-1)} \mathbf{w}_2^{(h-1)} \dots \mathbf{w}_{M_h}^{(h-1)}\right)$$

$$\boldsymbol{\phi}^{(h-1)}(\mathbf{x}) \xrightarrow{\mathbf{W}^{(h-1)}}$$



$$\rightarrow \boldsymbol{\phi}^{(h)}(\mathbf{x}) = \begin{pmatrix} 1 \\ \phi_1^{(h)}(\mathbf{x}) \\ \vdots \\ \phi_{M_h}^{(h)}(\mathbf{x}) \end{pmatrix}$$

$$\phi_0^{(h)} = 1$$

$$\boldsymbol{\phi}^{(0)}(\mathbf{x}) = \mathbf{x} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_D \end{pmatrix}$$

$M_0 = D$

$$\left(\mathbf{w}_m^{(h-1)}\right)^{\mathrm{T}} = \left(w_{m0}^{(h-1)}\ \ w_{m1}^{(h-1)}\ \dots w_{mM_{h-1}}^{(h-1)}\right)$$

$$\boldsymbol{\phi}^{(h-1)}(\mathbf{x}) = \begin{pmatrix} 1 \\ \phi_1^{(h-1)}(\mathbf{x}) \\ \vdots \\ \phi_{M_h}^{(h-1)}(\mathbf{x}) \end{pmatrix}$$

$$\phi_m^{(h)}(\mathbf{x}) = \varphi^{(h)}\left(\left(\mathbf{w}_m^{(h-1)}\right)^{\mathrm{T}} \boldsymbol{\phi}^{(h-1)}(\mathbf{x})\right); \qquad m = 1, \dots, M_h$$

$$M_{H+1} = 1$$

In the last layer: $\varphi^{(H+1)}(z) = \varphi^{(o)}(z)$ (logistic, probit,…)

$h = 1, \dots, H, H+1$

Learning the network weights

The network weights are determined by minimization of a cost function:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \left(\left[-\sum_{n \in C_0} \log\big(1 - o(\mathbf{x}_n; \mathbf{w})\big) - \sum_{n \in C_1} \log\big(o(\mathbf{x}_n; \mathbf{w})\big) \right] + \lambda_1 \|\mathbf{w}\|_1 + \lambda_2 \|\mathbf{w}\|_2^2 \right)$$

Cross-entropy error

$\lambda_1 > 0$ | L_1 penalty | $\lambda_2 > 0$ | L_2 penalty

Use CV to select

- **Architecture**: Number of hidden layers / Neurons in hidden layer
- **Optimization method**: Quasi-Newton, Gradient descent, stochastic gradient descent, use of momentum… Gradients needed!
- **Hyperparameters**: Magnitude of penalties, of momentum term, maximum # of iterations, …

Multi-class classification (K classes)

1-of K encoding of the **labeled data**

$$\mathcal{D} = \{(\mathbf{x}_n, c_n)\}_{n=1}^{N} \Rightarrow \mathcal{D} = \{(\mathbf{x}_n, \mathbf{t}_n)\}_{n=1}^{N}; \quad \boldsymbol{t}_n^{\mathrm{T}} = (t_{n1}\ t_{n2} \ldots t_{nK})$$

$$t_{nk} = \mathbb{I}[c_n = C_k] = \begin{cases} 0 & \text{if} \quad y_n \neq C_k \\ 1 & \text{if} \quad y_n = C_k \end{cases}; \quad k = 1, 2, \ldots, K$$

$$\mathcal{L}: \mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N} \rightarrow o(\,\cdot\,; \mathbf{w})$$

$$z(\,\cdot\,; \mathbf{w}): \quad \mathbf{x} \in \mathcal{X} \rightarrow \mathbf{z}(\mathbf{x}; \mathbf{w}) \in \mathbb{R}^K$$

$$\mathbf{o}(\mathbf{x}; \mathbf{w}) = \boldsymbol{\varphi}^{(\mathrm{o})}\big(\mathbf{z}(\mathbf{x}; \mathbf{w})\big) \in \Delta^K$$

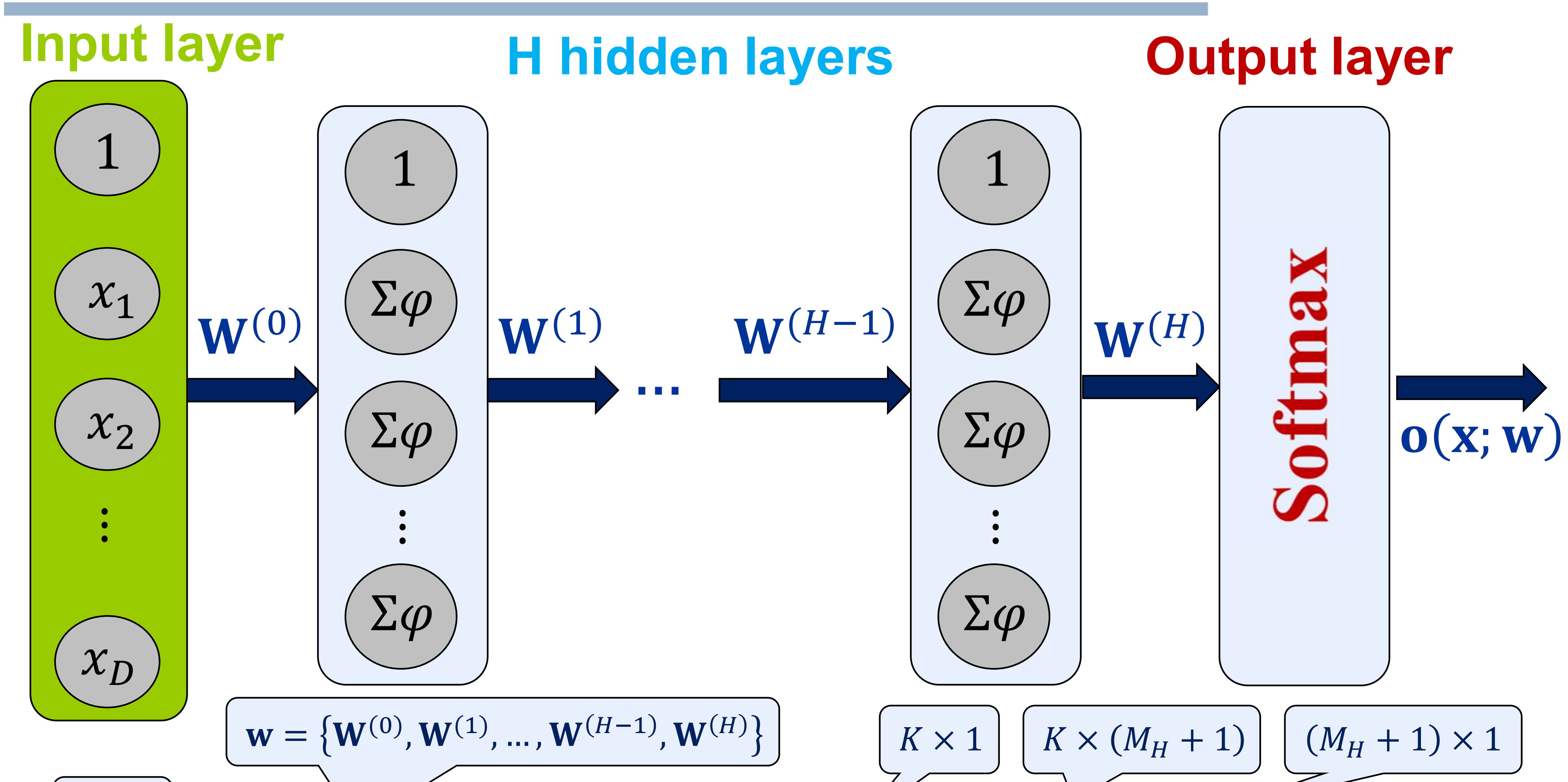
Probability simplex in K dimensions

Discriminative model $\hat{p}(C_k|\mathbf{x}, \mathbf{w}) = \varphi_K^{(o)}(\mathbf{z}(\boldsymbol{x}; \mathbf{w}))$

$$\varphi_k^{(o)}(\mathbf{z}) \geq 0 \qquad k = 1, \ldots, K$$
$$\varphi_1^{(o)}(\mathbf{z}) + \cdots + \varphi_K^{(o)}(\mathbf{z}) = 1$$

$$\left(\boldsymbol{\varphi}^{(\mathrm{o})}(\mathbf{z})\right)^{\mathrm{T}} = \left(\varphi_1^{(o)}(\mathbf{z}) \ldots \varphi_K^{(o)}(\mathbf{z})\right)$$

MLP for multiclass classification



$$\mathbf{o}(\mathbf{x};\mathbf{w}) = \boldsymbol{\varphi}^{(o)}\big(\mathbf{z}(\mathbf{x},\mathbf{w})\big) = \boldsymbol{\varphi}^{(o)}\left(\left(\mathbf{W}^{(H)}\right)^T \boldsymbol{\phi}^{(H)}(\mathbf{x})\right)$$

Softmax layer

$$\mathbf{z}(\mathbf{x};\mathbf{w}) = \begin{pmatrix} z_1(\mathbf{x},\mathbf{w}) \\ z_2(\mathbf{x},\mathbf{w}) \\ \vdots \\ z_K(\mathbf{x},\mathbf{w}) \end{pmatrix} \Rightarrow \boxed{\textbf{Softmax}} \Rightarrow \boldsymbol{\varphi}^{(\mathrm{o})}\big(\mathbf{z}(\mathbf{x};\mathbf{w})\big) = \begin{pmatrix} \varphi_1^{(o)}\big(\mathbf{z}(\mathbf{x},\mathbf{w})\big) \\ \varphi_2^{(o)}\big(\mathbf{z}(\mathbf{x},\mathbf{w})\big) \\ \vdots \\ \varphi_K^{(o)}\big(\mathbf{z}(\mathbf{x},\mathbf{w})\big) \end{pmatrix}$$

$$\hat{p}(C_k|\mathbf{x},\mathbf{w}) = \varphi_k^{(o)}\big(\mathbf{z}(\mathbf{x},\mathbf{w})\big)$$
$$k = 1, \dots, K$$

Probabilities

$$\varphi_k^{(o)}(\mathbf{z}) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} \geq 0 \qquad k = 1, \dots, K$$

$$\varphi_1^{(o)}(\mathbf{z}) + \cdots + \varphi_K^{(o)}(\mathbf{z}) = 1$$

Learning by maximum lihelihood

Samples assumed to be iid (independent identically distributed)

$$\mathbf{w}^* = \arg\max_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

- Likelihood of the model, given $\mathcal{D} = \{(\mathbf{x}_n, \mathbf{t}_n)\}_{n=1}^N$

$$\mathcal{L}(\mathbf{w}) = \hat{P}\left(\{\mathbf{t}_n\}_{n=1}^N \middle| \{\mathbf{x}_n\}_{n=1}^N, \mathbf{w}\right) = \prod_{n=1}^N \hat{p}(t_n|\mathbf{x}_n, \mathbf{w}) =$$

Factorizes because samples are assumed to be independent

$$= \prod_{n=1}^N \prod_{k=1}^K \left(\hat{p}(C_k|\mathbf{x}_n, \mathbf{w})\right)^{t_{nk}}$$

Factors equal because samples are assumed to be identically dist.

- Log-likelihood function

$t_{nk} = \mathbb{I}[c_n = C_k]$

$$\log \mathcal{L}(\mathbf{w}) = \sum_{n=1}^N \sum_{k=1}^K t_{nk} \log \hat{p}(C_k|\mathbf{x}_n, \mathbf{w})$$

Minimization of the cross-entropy error

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} CE(\mathbf{w})$$

- Cross-entropy error

$$CE(\mathbf{w}) = -\log \mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \log \hat{p}(C_k|\mathbf{x}_n, \mathbf{w})$$

- Discriminative model: $\hat{p}(C_1|\mathbf{x}, \mathbf{w}) = \varphi_k^{(o)}\big(z(\mathbf{x}_n, \mathbf{w})\big)$

Output of the kth neuron in the last layer of the MLP.

$$\widehat{CE}(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \log \varphi_k^{(o)}\big(z(\mathbf{x}_n, \mathbf{w})\big)$$

Neural networks summary

- **Advantages**
  - Excellent predictors.
  - Adaptive (online learning)
- **Disadvantages**
  - Costly training.
  - Finding appropriate architecture can be difficult
    - Number of hidden layers / nodes in each hidden layer
    - Activation function
  - Determining the hyperparameters for the optimization
    - Type of optimization
    - In SGD: learning rate, size of mini-batches, momentum term, strength of regularization terms, …
  - Difficult interpretation.

State-of-the-art: Deep neural networks

References

- Multilayer perceptron with TensorFlow
  https://playground.tensorflow.org/

- Scikit learn documentation
  - Neural networks
    http://scikit-learn.org/stable/modules/neural_networks_supervised.html

- Simon O. Haykin "Neural Networks and Learning Machines, 3rd edition", Pearson (2009)

- Ian Goodfellow and Yoshua Bengio and Aaron Courville "Deep Learning", MIT Press (2016)

- 3Blue1Brown [https://www.3blue1brown.com/]:
  - Neural networks: https://www.3blue1brown.com/lessons/neural-networks
  - Gradient descent: https://www.3blue1brown.com/lessons/gradient-descent
  - Backpropagation: https://www.3blue1brown.com/lessons/backpropagation
  - Backpropagation calculus: https://www.3blue1brown.com/lessons/backpropagation-calculus