Chapter 2. Multilayer Perceptron

Neural Networks

2023/2024

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

Departamento de Sistemas Informáticos y Computación

Index

1 Introduction ▷ 3

2 Multilayer Perceptron ▷ 7

3 Error Back-propagation algorithm ▷ 22

4 Some issues about BackProp ▷ 41

5 A brief introduction to deep neural networks ▷ 55

6 Bibliography ▷ 66

Index

- 1 *Introduction* ▷ 3
- 2 Multilayer Perceptron ▷ 7
- 3 Error Back-propagation algorithm ▷ 22
- 4 Some issues about BackProp ▷ 41
- 5 A brief introduction to deep neural networks ▷ 55
- 6 Bibliography ▷ 66

Introduction

- From Perceptron to Multi-Layer Perceptron (MLP)
- From linear separability to complex decision boundaries (generalized linear discriminant functions)
  - Add layers to the original Perceptron architecture
  - Activation units to overcome the linear limitations
- Back-propagation:
  - Non-convex optimization
  - Initialization
  - Local-minima

Linear Discriminant Functions

- Graphical representation of a LDF. Given an input $\mathbf{x} \in \mathbb{R}^D$

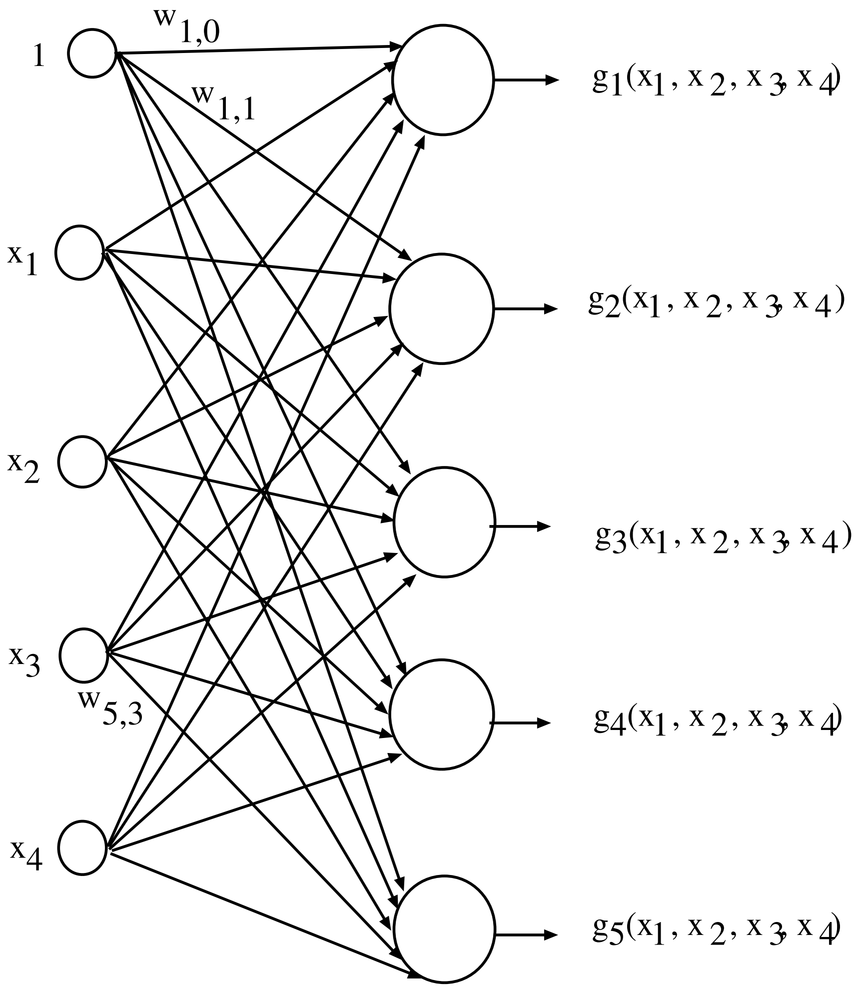
$$\hat{c} = G(\mathbf{x}) \equiv \underset{1 \leq c \leq C}{\text{argmax}} \quad g_c(\mathbf{x})$$

$$g_c(\mathbf{x}) = \mathbf{w}_c^t \mathbf{x} + w_{c0}$$

$$\mathbf{g}(\mathbf{x}) = \mathbf{W}\mathbf{x}$$

$\mathbf{W}$ is a matrix of dimension $C \times (D+1)$ (including the thresholds w_{c0}) and $\mathbf{x}$ is of dimension $D+1$ ($x_0 = 1$)

  - $E = \mathbb{R}^4$,
  - $\mathbf{w}_c \in \mathbb{R}^4$, $w_{c0} \in \mathbb{R}$ for $1 \leq c \leq 5$
  - Classes$=\{1, 2, 3, 4, 5\}$



Generalized Linear Discriminant Functions

- Graphical representation of a GLDF. Given an input $\mathbf{x} \in \mathbb{R}^D$ and a function $\Phi : \mathbb{R}^D \to \mathbb{R}^{D'}$

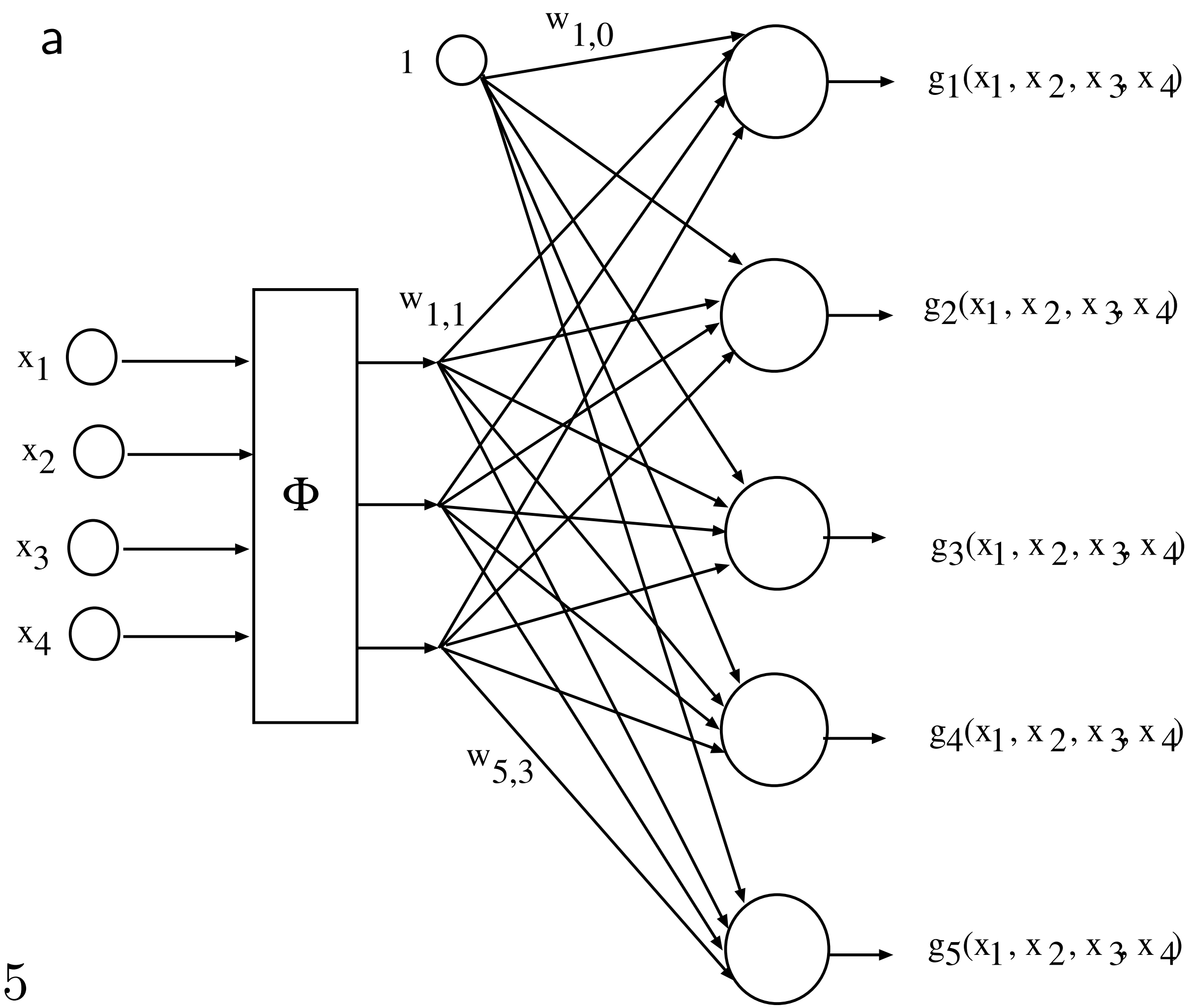
$$\hat{c} = G(\mathbf{x}) \equiv \operatorname*{argmax}_{1 \leq c \leq C} \; g_c(\mathbf{x})$$

$$g_c(\mathbf{x}) = \mathbf{w}_c^t \Phi(\mathbf{x}) + w_{c0}$$

$$\mathbf{g}(\mathbf{x}) = \mathbf{W}\Phi(\mathbf{x})$$

In this case, $\mathbf{W}$ is a matrix of dimension $C \times (D' + 1)$.

  - $E = \mathbb{R}^4$,
  - $\Phi : E \to \mathbb{R}^3$
  - Classes=$\{1, 2, 3, 4, 5\}$
  - $\mathbf{w}_c \in \mathbb{R}^3$, $w_{c0} \in \mathbb{R}$ for $1 \leq c \leq 5$



Index

1 Introduction ▷ 3

○ 2 *Multilayer Perceptron* ▷ 7

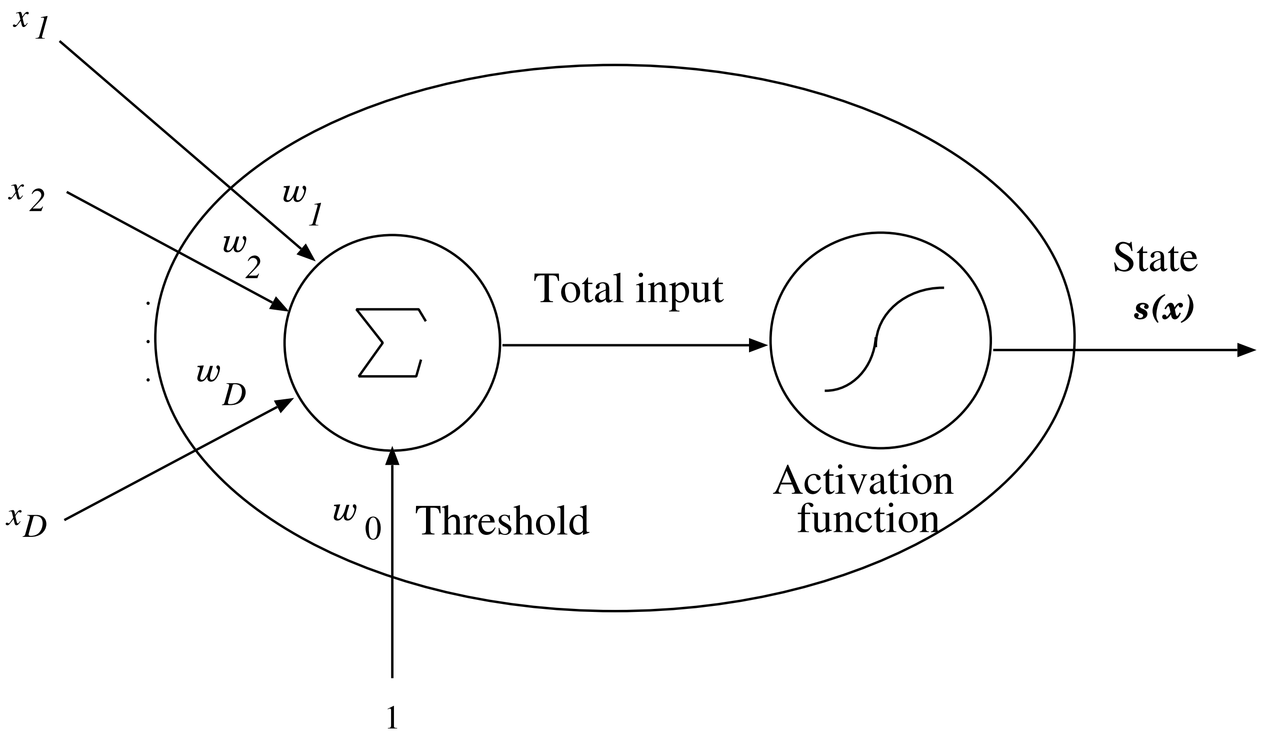
3 Error Back-propagation algorithm ▷ 22

4 Some issues about BackProp ▷ 41

5 A brief introduction to deep neural networks ▷ 55

6 Bibliography ▷ 66

Logistic linear discriminant functions



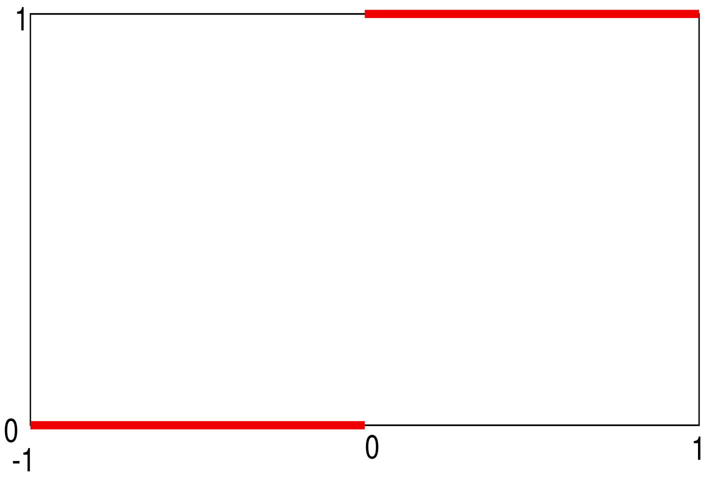
Compact notation: from $\mathbf{x} = (x_1, ..., x_D)$ to $\mathbf{x} = (1, x_1, \cdots, x_D)$, and from $\mathbf{w} = (w_1, \cdots, w_D)$ to $\mathbf{w} = (w_0, w_1, \cdots, w_D)$:

$$s(\mathbf{x}) \ = \ f(\sum_{k=1}^{D} w_k x_k + w_0) \ = \ f(\mathbf{w}^t \mathbf{x}),$$ where f is an activation function.

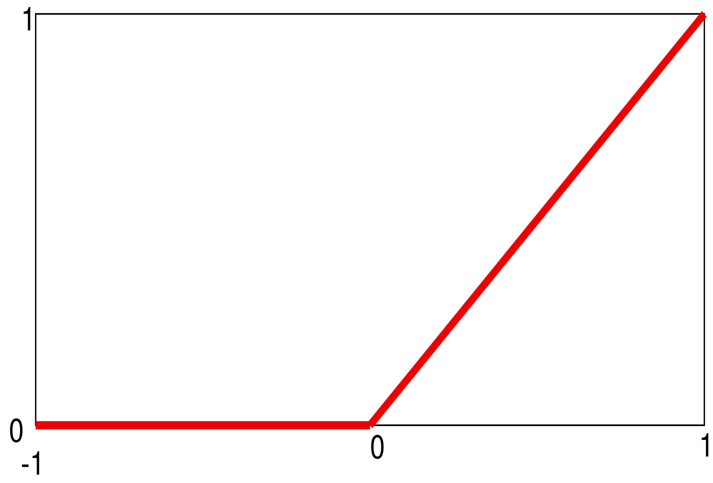
Activation functions

- *Linear*: $f_L(z) = z$, $z \in \mathbb{R}$
- *Step*: $f_E(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0 \end{cases}$, $z \in \mathbb{R}$
- *ReLU* (rectified linear unit): $f_R(z) = \max(0, z)$, $z \in \mathbb{R}$
- *PReLU* (parametric rectified linear unit): $f_{PR}(z) = \begin{cases} z & \text{if } z > 0 \\ a\ z & \text{if } z \leq 0 \end{cases}$, $z \in \mathbb{R}$
- *Sigmoid*: $f_S(z) = \frac{1}{1+\exp(-z)}$, $z \in \mathbb{R}$
- *Hyperbolic tangent*: $f_T(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$, $z \in \mathbb{R}$ $\quad (f_T(z) = 2f_S(2z) - 1)$
- *Softmax:* $f_{SM}(z_j) = \dfrac{\exp(z_j)}{\sum_{j'=1}^{M} \exp(z_{j'})}$; $\Big(f_{SM}(z_j) = f_S(z_j - \ln(\sum_{j' \neq j} \exp(z_{j'})))\Big)$, $z_1, \ldots, z_M \in \mathbb{R}$
- Others: Exponential Linear Units (ELU), Maxout, ...

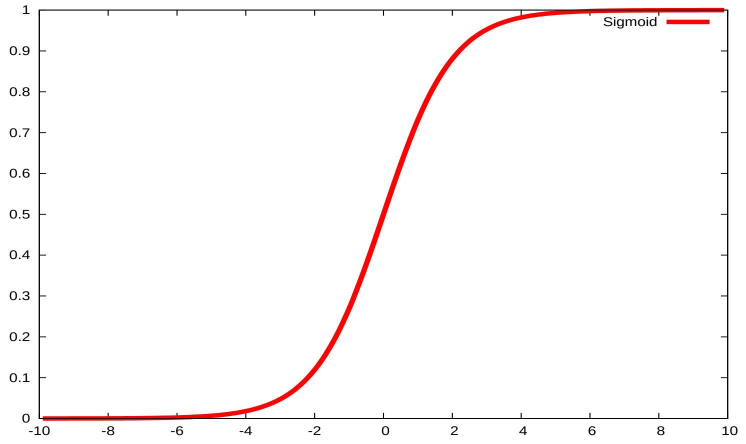
Activation functions



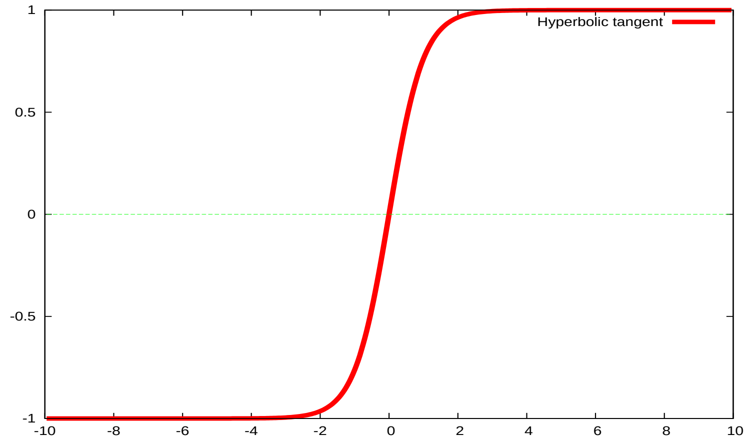
Step



ReLU

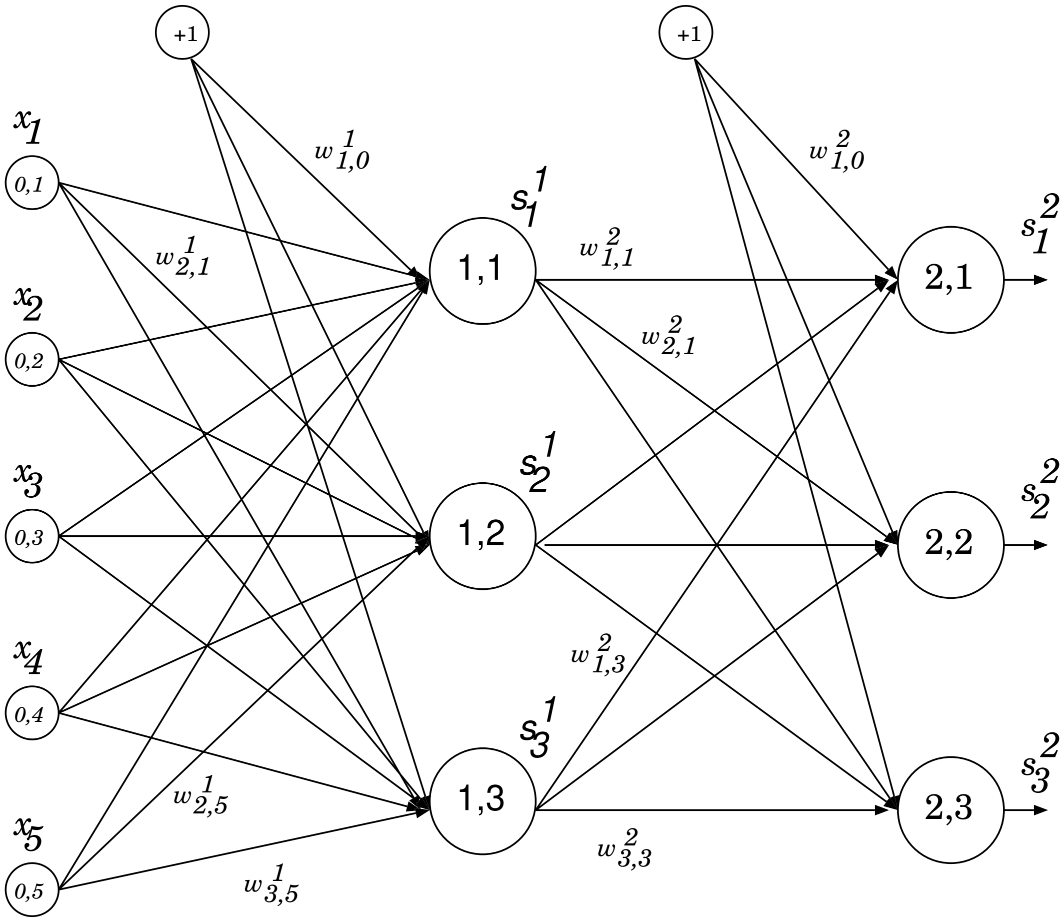


Sigmoid



Hyperbolic tangent

A two-layer perceptron



Hidden layer

$$s_j^1 = f(\textstyle\sum_i w_{j,i}^1 \; x_i), \; 1 \leq j \leq M_1$$

Output layer

$$s_j^2 = f(\textstyle\sum_i w_{j,i}^2 \; s_i^1), \; 1 \leq j \leq M_2$$

A two-layer perceptron

A two-layer perceptron consists of a combination of logistic linear discriminant functions grouped in layers and defines a set of M_2 discriminant functions:

$$g_k(\mathbf{x};\theta) \equiv s_k^2(\mathbf{x}) = f(\sum_{j=0}^{M_1} w_{k,j}^2 \; s_j^1(\mathbf{x})) = f(\sum_{j=0}^{M_1} w_{k,j}^2 \; f(\sum_{j'=0}^{M_0} w_{j,j'}^1 \; x_{j'}))$$

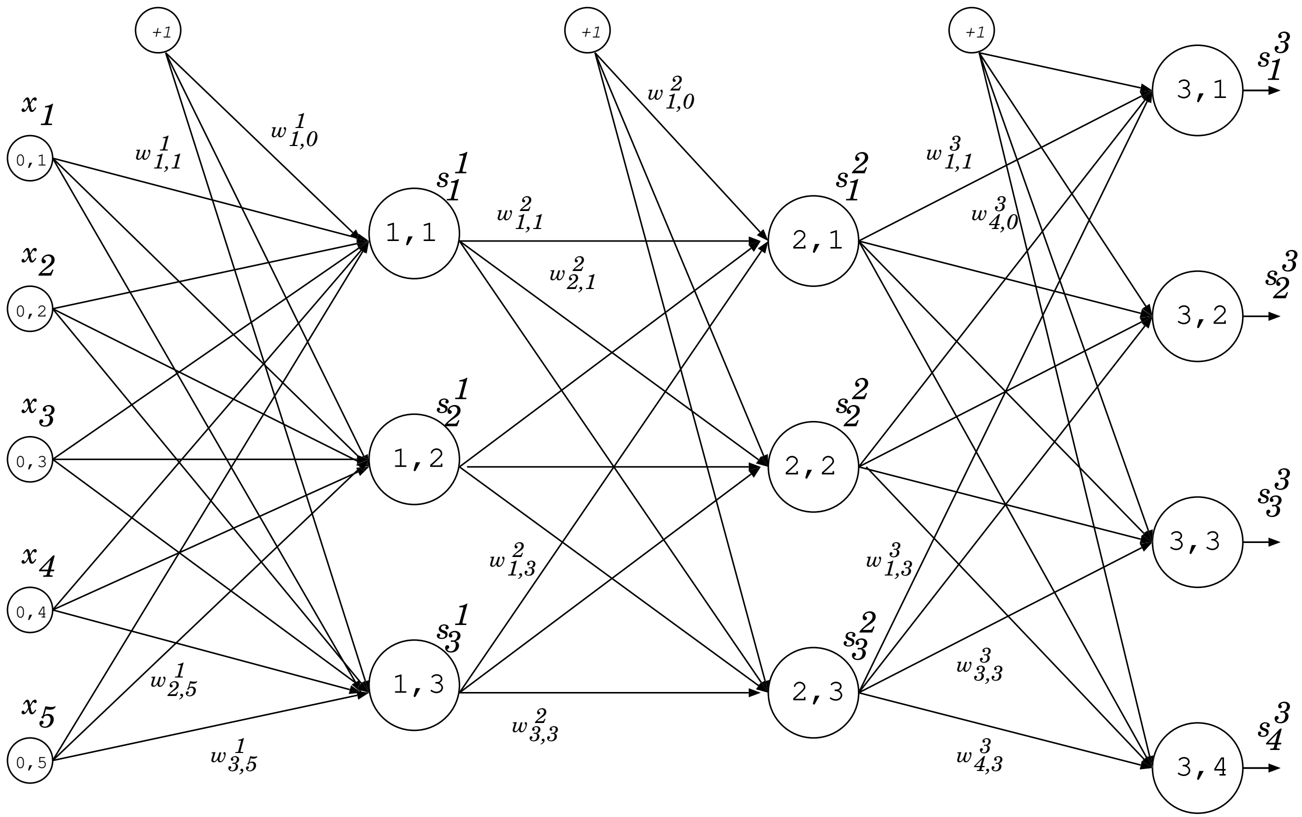
for $1 \leq k \leq M_2$, $M_0 \equiv D$. Therefore,

$$\theta \equiv \mathbf{w} = (w_{1,0}^1, \ldots, w_{M_1,M_0}^1, w_{1,0}^2, \ldots, w_{M_2,M_1}^2)$$

In compact notation: $\mathbf{g}(\mathbf{x};\theta) \equiv \mathbf{s}^2(\mathbf{x}) = \mathbf{f}(\mathbf{W}^2\mathbf{f}(\mathbf{W}^1\mathbf{x}))$

Regression problem: Let A be a training set $\{(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_N, \mathbf{t}_N)\}$, with $\mathbf{x}_n \in \mathbb{R}^{M_0}$, $\mathbf{t}_n \in \mathbb{R}^{M_2}$ search for $\mathbf{w}$ such that $\mathbf{s}^2(\mathbf{x}_n) = \mathbf{t}_n$ with $1 \leq n \leq N$.

A three-layer perceptron



Two hidden layers | An output layer

$$1 \le j \le M_1 \qquad 1 \le j \le M_2 \qquad 1 \le j \le M_3$$

$$s^1_j = f(\textstyle\sum_i w^1_{j,i}\ x_i) \qquad s^2_j = f(\textstyle\sum_i w^2_{j,i}\ s^1_i) \qquad s^3_j = f(\textstyle\sum_i w^3_{j,i}\ s^2_i)$$

A three-layer perceptron

A **three-layer perceptron** defines a set of M_3 discriminant functions:

$$g_k(\mathbf{x};\theta) \equiv s_k^3(\mathbf{x}) = f(\sum_{j=0}^{M_2} w_{k,j}^3 \; s_j^2(\mathbf{x}))$$

$$= f(\sum_{j=0}^{M_2} w_{k,j}^3 \; f(\sum_{j'=0}^{M_1} w_{j,j'}^2 \; s_{j'}^1(\mathbf{x}))) = f(\sum_{j=0}^{M_2} w_{k,j}^3 \; f(\sum_{j'=0}^{M_1} w_{j,j'}^2 \; f(\sum_{j''=0}^{M_0} w_{j',j''}^1 \; x_{j''})))$$

for $1 \leq k \leq M_3$, $M_0 \equiv D$. Therefore,

$$\theta \equiv \mathbf{w} = (w_{10}^1, \ldots, w_{M_1,M_0}^1, w_{1,0}^2, \ldots, w_{M_2,M_1}^2, w_{1,0}^3, \ldots, w_{M_3,M_2}^3)$$

In compact notation: $\mathbf{g}(\mathbf{x};\theta) \equiv \mathbf{s}^3(\mathbf{x}) = \mathbf{f}(\mathbf{W}^3\mathbf{f}(\mathbf{W}^2\mathbf{f}(\mathbf{W}^1\mathbf{x})))$

Regression problem: Let A be a training set $\{(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_N, \mathbf{t}_N)\}$, with $\mathbf{x}_n \in \mathbb{R}^{M_0}$, $\mathbf{t}_n \in \mathbb{R}^{M_3}$: search for $\mathbf{w}$ such that $\mathbf{s}^3(\mathbf{x}_n) = \mathbf{t}_n$ with $1 \leq n \leq N$.

Multilayer perceptrons and activation functions

- Given a two-layer perceptron,

$$g_k(\mathbf{x}) \equiv s_k^2(\mathbf{x}) = f(\sum_{j=0}^{M_1} w_{k,j}^2 \; s_j^1(\mathbf{x})) = f(\sum_{j=0}^{M_1} w_{k,j}^2 \; f(\sum_{j'=0}^{M_0} w_{j,j'}^1 \; x_{j'}))$$

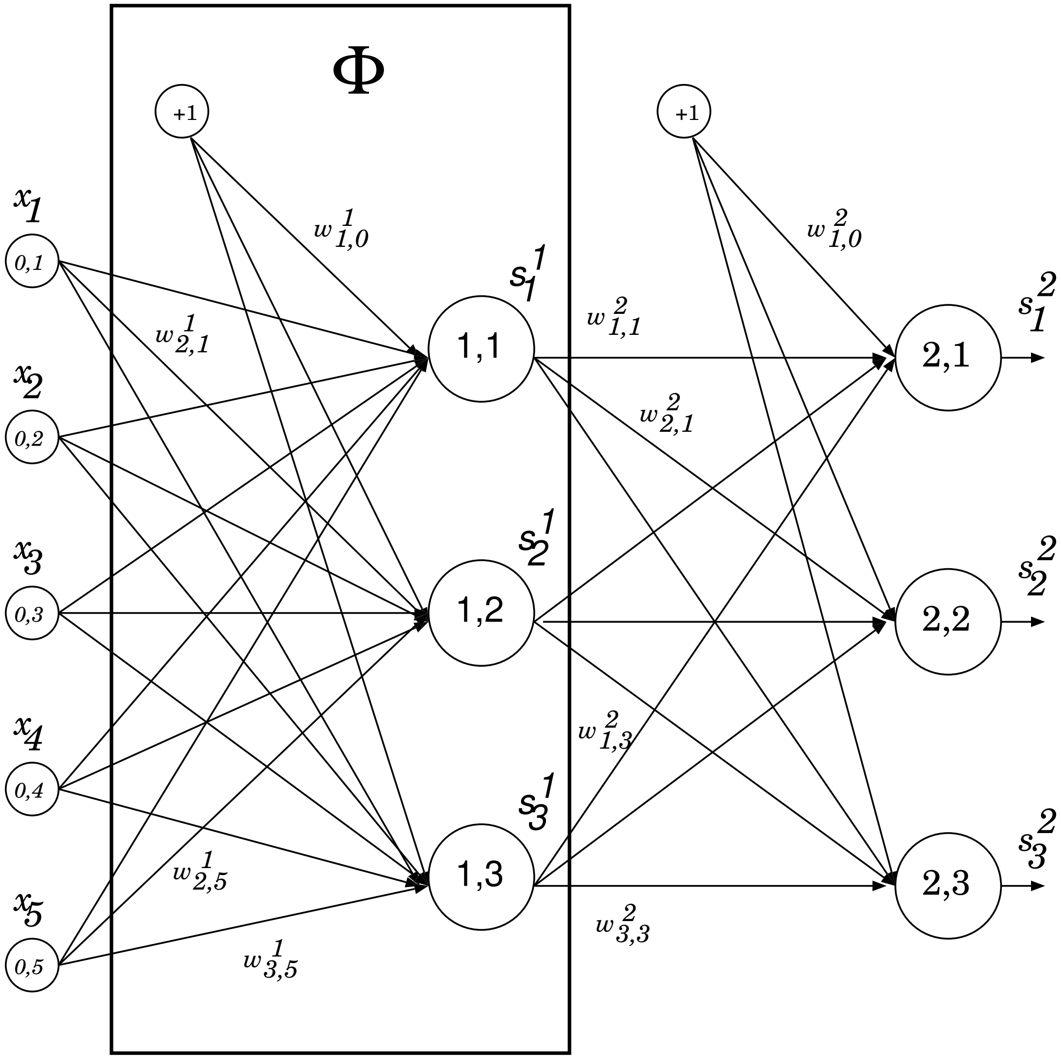
- If all the activation functions are linear, a multilayer perceptron defines **a linear discriminant function**:

$$g_k(\mathbf{x}) \equiv s_k^2(\mathbf{x}) = \sum_{j'=0}^{M_0} \left(\sum_{j=0}^{M_1} w_{k,j}^2 w_{j,j'}^1 \right) x_{j'} = \sum_{j'=0}^{M_0} w_{k,j'} \; x_{j'}$$

- If at least one activation function is not linear (and all activation functions in the output layer are linear), a multilayer perceptron defines a **generalized linear discriminant function**:

$$g_k(\mathbf{x}) \equiv s_k^2(\mathbf{x}) = \sum_{j=0}^{M_1} w_{kj}^2 \; f(\sum_{j'=0}^{M_0} w_{j,j'}^1 \; x_{j'}) = \sum_{j=0}^{M_1} w_{k,j}^2 \; \Phi_j(\mathbf{x})$$

A two-layer perceptron



The multilayer perceptron as a classifier

For a problem of C classes, the multilayer perceptron has C units and the training target vectors $\mathbf{t}_n$ have the form for $1 \leq n \leq N$:

$$t_{nc} = \begin{cases} 1\ (+1) & \text{if } \mathbf{x}_n \text{ is of the class } c \\ 0\ (-1) & \text{otherwise} \end{cases}$$

The classifier is:

$$G(\mathbf{x}) = \underset{1 \leq k \leq C}{\text{argmax}}\ g_k(\mathbf{x}; \theta) \equiv \underset{1 \leq k \leq M_2 \equiv C}{\text{argmax}}\ s_k^2(\mathbf{x})$$

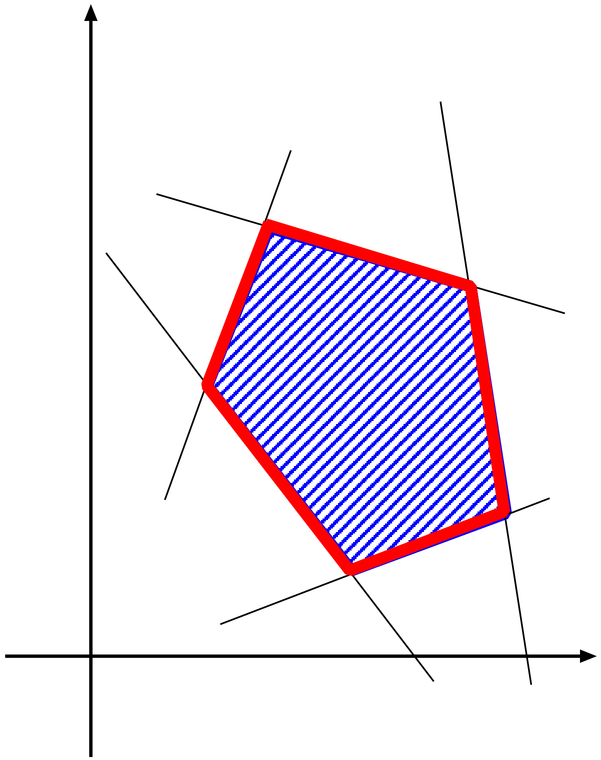
Softmax activation function is used in the output layer.

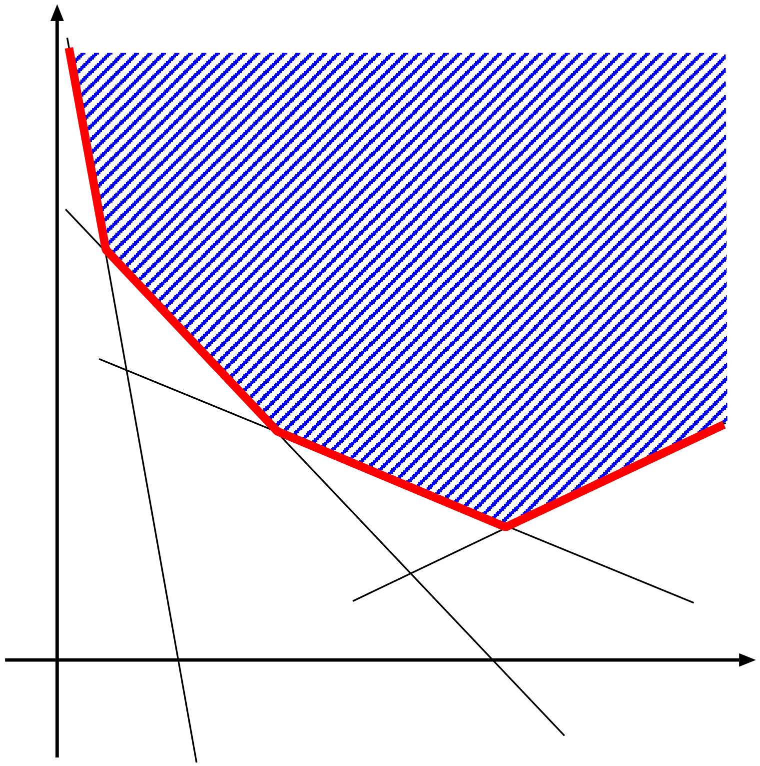
Given a training sample with N patterns, is there a multilayer perceptron that classifies correctly the training sample?

- If the training sample is linearly separable: a perceptron without hidden layers.
- A multilayer perceptron of 1 hidden layer with $N-1$ nodes and step activation functions can classify correctly the sample.
- About generalization?

Properties of the multilayer perceptron

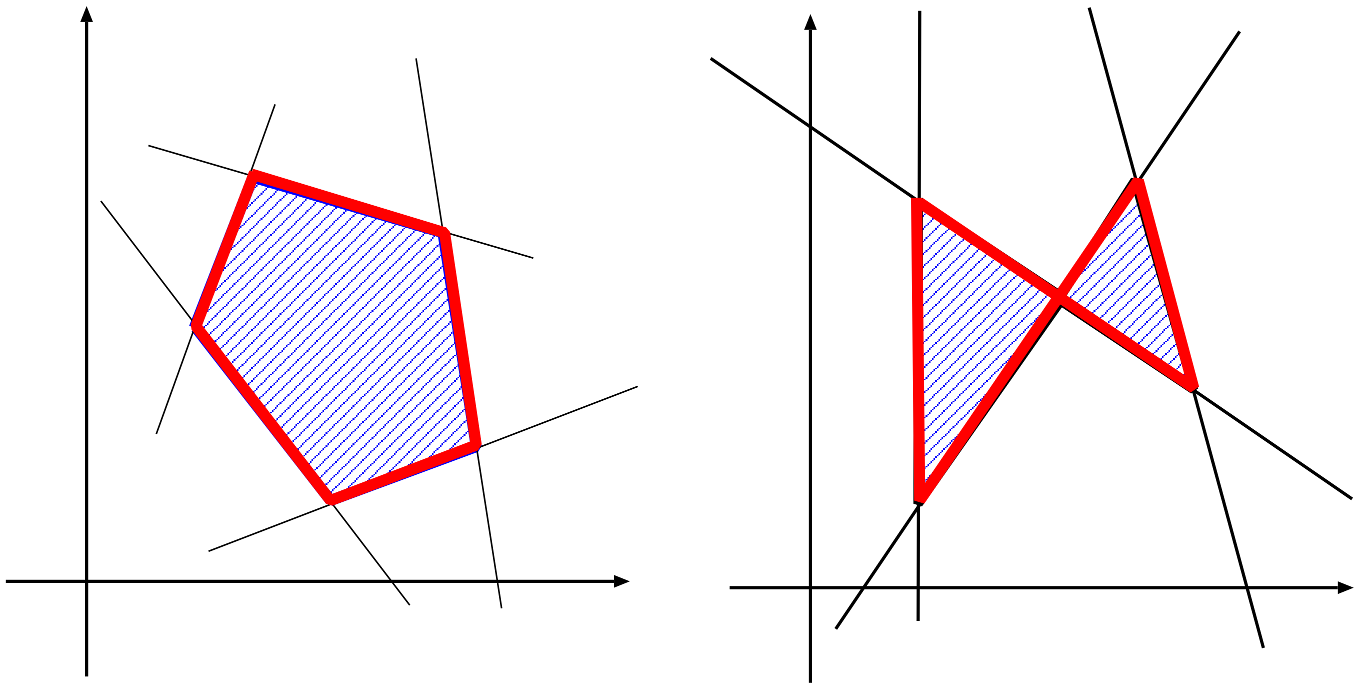
- A multilayer perceptron can implement decision borders that are linear at intervals.





Properties of the multilayer perceptron

- A multilayer perceptron with a hidden layer and step activation functions can implement convex decision borders.
- Any decision border based on hyperplanes can be implemented by means of a multilayer perceptron with two hidden layers and step activation functions.



Regression with a multilayer perceptron (function approximation)

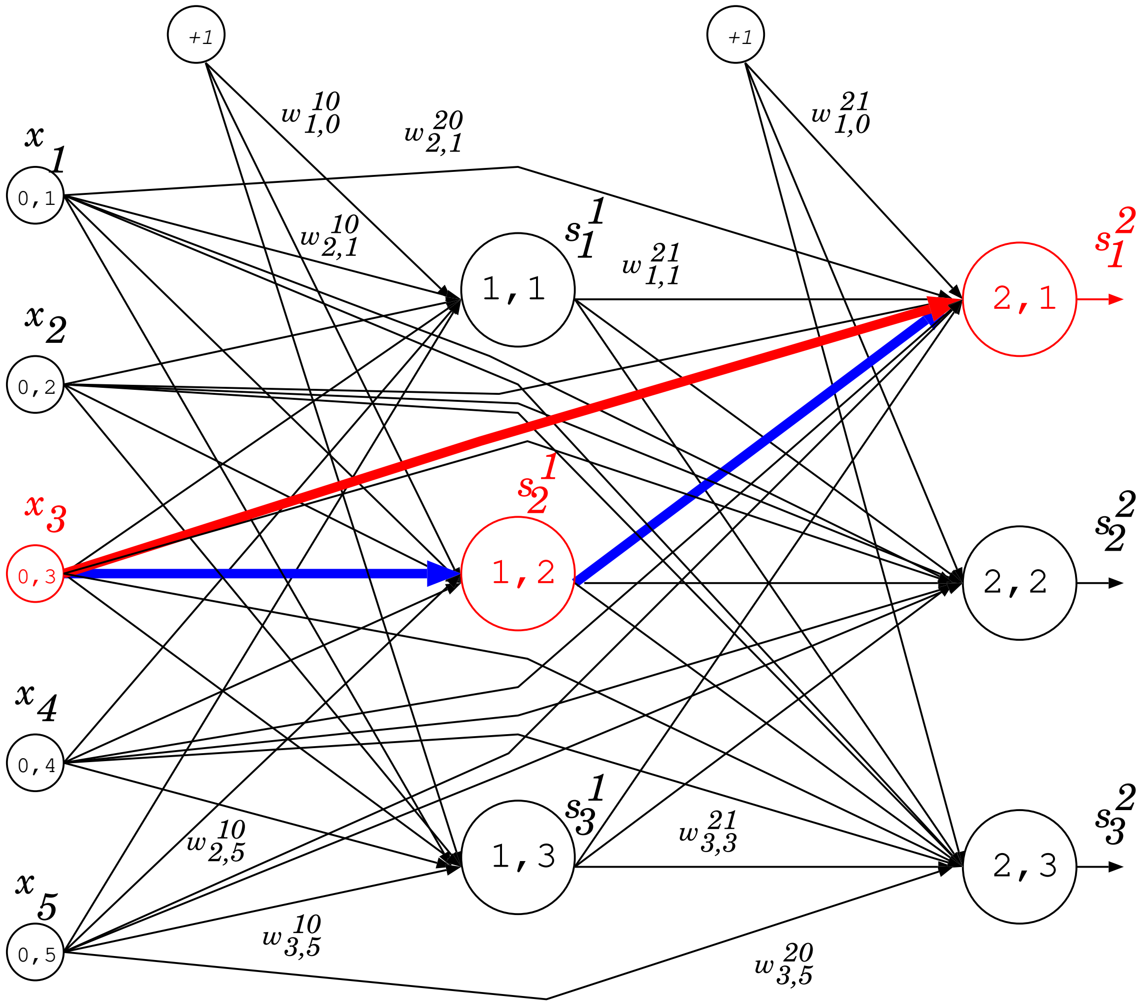
$$F(\mathbf{x}) : \mathbb{R}^{M_0} \rightarrow \mathbb{R}^{M_2} : s_k^2(\mathbf{x}) = \sum_{j=0}^{M_1} w_{k,j}^2 \ f(\sum_{j'=0}^{M_0} w_{j,j'}^1 \ x_{j'}) \text{ for } 1 \leq k \leq M_2$$

Linear activation function is usualy adopted in the output layer.

- Any function can be arbitrary approached by means of a multilayer perceptron of two hidden layers and step activation functions and therefore with sigmoid functions too.

- Any function can be arbitrary approached by means of a multilayer perceptron of a hidden layer and step activation functions and therefore with sigmoid functions if the number of hidden nodes is high enough.

Feed-forward networks



$$\mathbf{s}^1(\mathbf{x}) = \mathbf{f}(\mathbf{W}^{1,0}\mathbf{x}) \qquad \mathbf{s}^2(\mathbf{x}) = \mathbf{f}(\mathbf{W}^{2,1}\mathbf{s}^1(\mathbf{x}) + \mathbf{W}^{2,0}\mathbf{x})$$

Index

1 Introduction ▷ 3

2 Multilayer Perceptron ▷ 7

◦ 3 *Error Back-propagation algorithm* ▷ 22

4 Some issues about BackProp ▷ 41

5 A brief introduction to deep neural networks ▷ 55

6 Bibliography ▷ 66

Error backpropagation algorithm for the multilayer perceptron

REGRESSION PROBLEM: Given a topology of a multilayer perceptron and $A = \{(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_N, \mathbf{t}_N)\}$, with $\mathbf{x}_n \in \mathbb{R}^{N_0}$, $\mathbf{t}_n \in \mathbb{R}^{N_2}$, search for $\mathbf{w}$ such that minimizes the objective function (mean squared error):

$$E_A(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{k=1}^{M_2} \left(t_{n,k} - s_k^2(\mathbf{x}_n; \mathbf{w})\right)^2$$

that is,

$$\widehat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} E_A(\mathbf{w})$$

SOLUTION, gradient descent $(1 \leq k \leq 2, 1 \leq i \leq M_k, 0 \leq j \leq M_{k-1})$:

$$\Delta w_{i,j}^k = -\rho \, \frac{\partial E_A(\mathbf{w})}{\partial w_{i,j}^k} \qquad \left(\Delta \mathbf{W}^k = -\rho \nabla_{\mathbf{W}^k} E_A(\mathbf{w})\right)$$

Gradient descent

$$\begin{aligned} \mathbf{w}(1) &= \text{arbitrary} \\ \mathbf{w}(k+1) &= \mathbf{w}(k) - \rho_k \, \nabla J \mid_{\mathbf{w}=\mathbf{w}(k)} \end{aligned}$$

Where $\rho_k \in \mathbb{R}^{>0}$ is a *learning rate* and $\nabla J \mid_{\mathbf{w}=\mathbf{w}(k)} \equiv \left(\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \ldots, \frac{\partial J}{\partial w_D} \right) \Bigg|_{\mathbf{w}=\mathbf{w}(k)}$

1-dimension

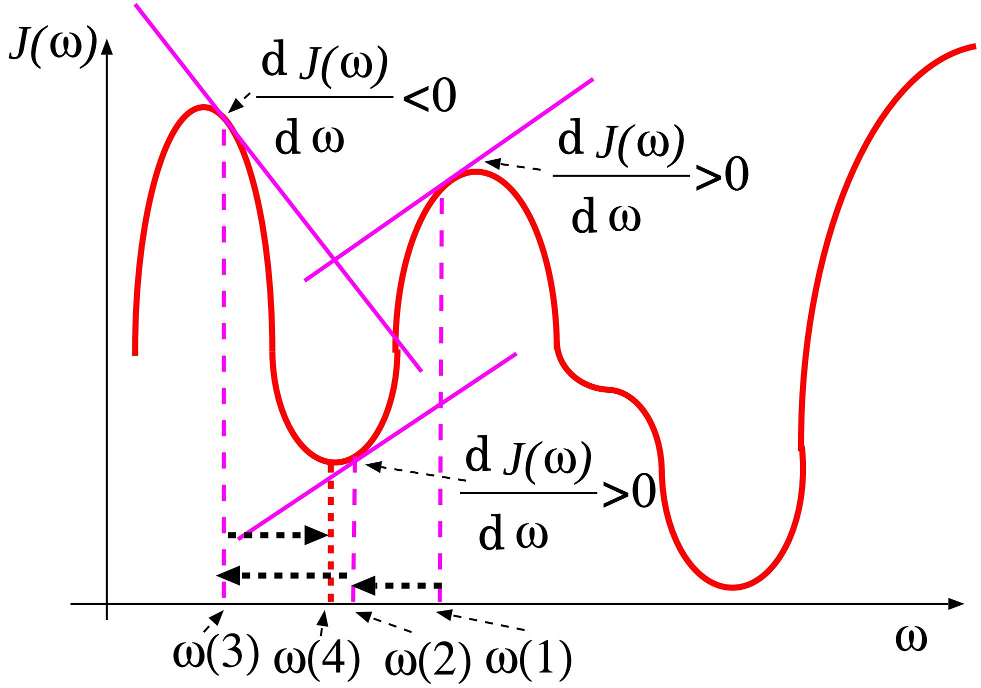
$w(1) = \text{arbitrary}$

$w(2) = w(1) - \rho \frac{dJ}{dw} \mid_{w(1)}$

$w(3) = w(2) - \rho \frac{dJ}{dw} \mid_{w(2)}$

$w(4) = w(3) - \rho \frac{dJ}{dw} \mid_{w(3)}$

$\frac{dJ}{dw} \mid_{w(4)} \quad = \quad 0$



Derivation of backpropagation algorithm (1)

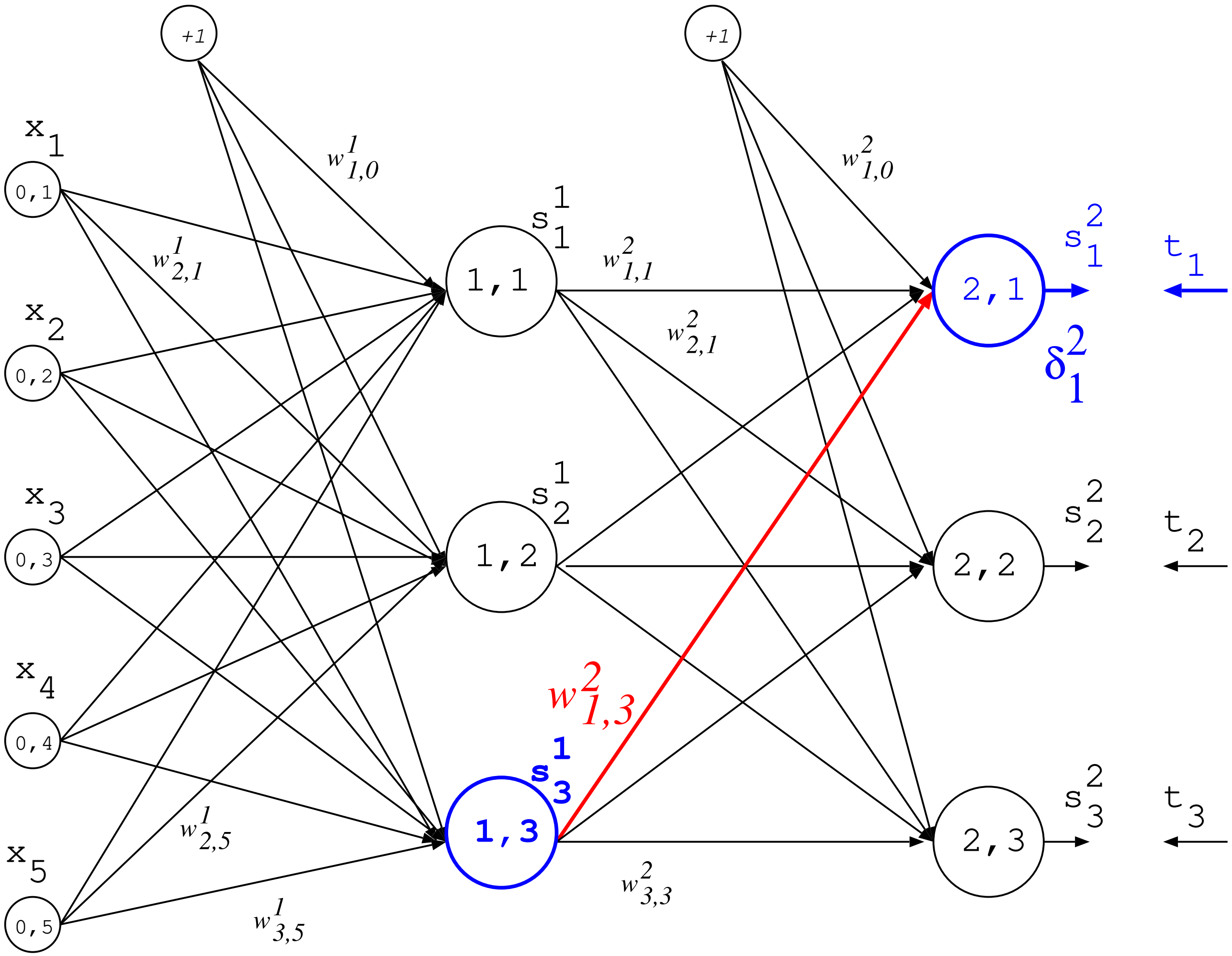
- Update weight of the output layer $w_{i,j}^2$ (if $N = 1$)

$$E_A(\mathbf{w}) = \frac{1}{2}\sum_{m=1}^{M_2}\left(t_m - s_m^2\right)^2; \quad s_m^2 = f\left(z_m^2\right); \quad z_m^2 = \left(\sum_{l=1}^{M_1} w_{m,l}^2 s_l^1\right)$$

$$\frac{\partial E_A}{\partial w_{i,j}^2} = \frac{\partial E_A}{\partial s_i^2}\frac{\partial s_i^2}{\partial z_i^2}\frac{\partial z_i^2}{\partial w_{i,j}^2} = (-1)(t_i - s_i^2)\; f'(z_i^2)\; s_j^1 = -\,\delta_i^2\; s_j^1$$

$$\boxed{\Delta w_{i,j}^2 = \rho\; \delta_i^2\; s_j^1}$$

Derivation of backpropagation algorithm (2)



$$\Delta w^2_{1\,3} \;=\; \rho\; \delta^2_1\; s^1_3 \;=\; \rho\;\; (t_1 - s^2_1)\; f'(z^2_1)\; s^1_3$$

Derivation of backpropagation algorithm (3)

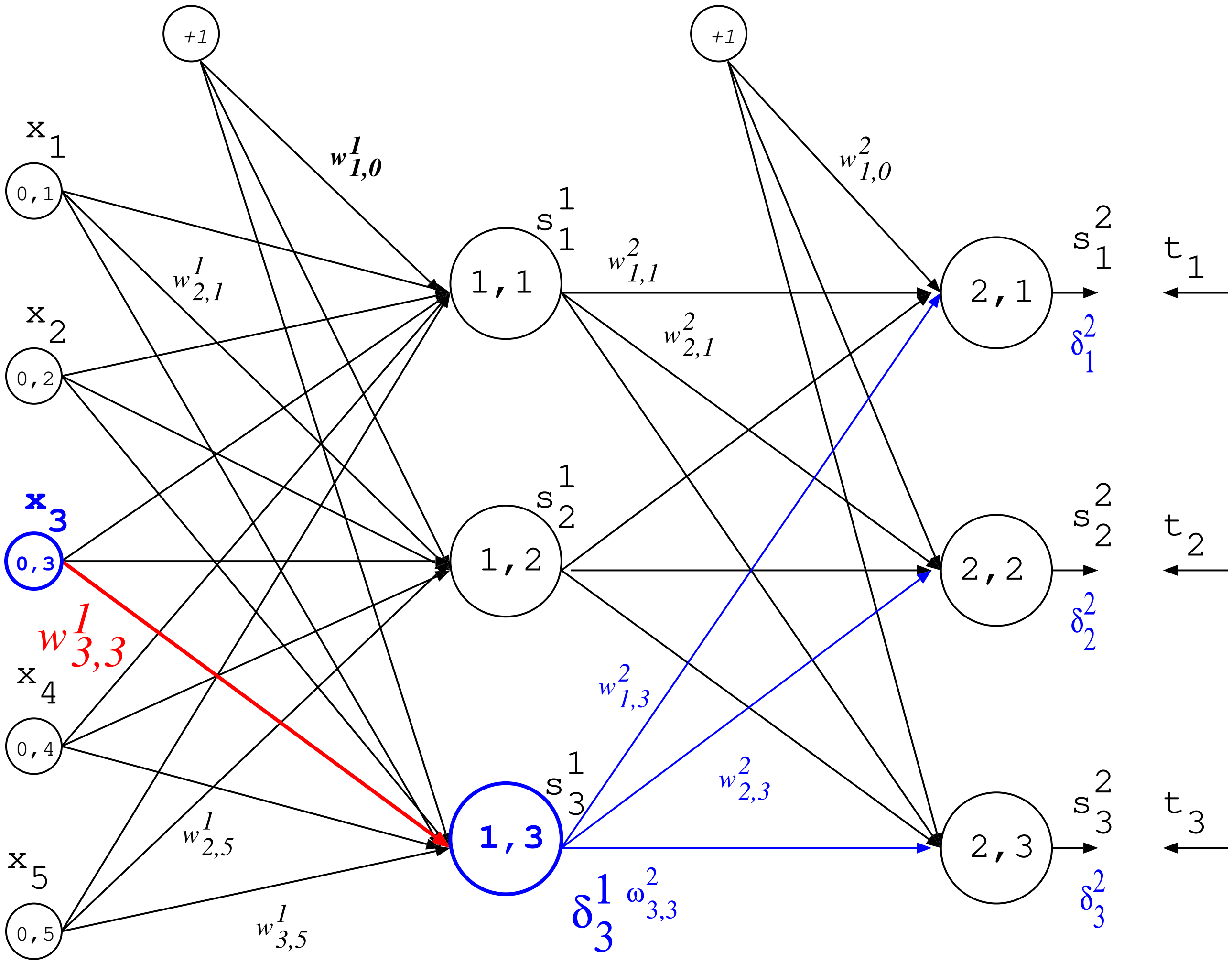
- Update the weight of the hidden layer $w^1_{i,j}$ (for $N = 1$)

$$E_A(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{M_2}\left(t_k - s^2_k\right)^2;\; s^2_k = f(z^2_k);\; z^2_k = \sum_{m=1}^{M_1} w^2_{k,m}\, s^1_m;\; s^1_m = f\left(z^1_m\right);\; z^1_m = \sum_{l=1}^{M_0} w^1_{m,l} x_l$$

$$\frac{\partial E_A}{\partial w^1_{i,j}} = \sum_{k=1}^{M_2} \frac{\partial E_A}{\partial s^2_k}\frac{\partial s^2_k}{\partial z^2_k}\frac{\partial z^2_k}{\partial s^1_i}\frac{\partial s^1_i}{\partial z^1_i}\frac{\partial z^1_i}{\partial w^1_{i,j}} = \sum_{k=1}^{M_2} -\delta^2_k\, w^2_{k,i}\, f'(z^2_i)\, x_j =$$
$$-\left(f'(z^2_i)\sum_{k=1}^{M_2}\delta^2_k\, w^2_{k,i}\right) x_j = -\,\delta^1_i\, x_j$$

$$\boxed{\Delta w^1_{i,j} = -\rho\,\frac{\partial E_A}{\partial w^1_{i,j}} = \rho\,\delta^1_i\, x_j}$$

Derivation of backpropagation algorithm (4)



$$\Delta w^1_{3\,3} \;=\; \rho\;\delta^1_3\;x_3 \;=\; \rho\;(\sum_r \delta^2_r w^2_{r,3}) f'(z^1_3)\;x_3$$

BackProp for N training samples

- Updating the weights of the output layer: $(1 \leq i \leq M_2,\ 0 \leq j \leq M_1)$:

$$\Delta w_{ij}^2 = -\rho \, \frac{\partial E_A(\mathbf{w})}{\partial w_{ij}^2} = \frac{\rho}{N} \sum_{n=1}^{N} \delta_i^2(\boldsymbol{x}_n) \; s_j^1(\boldsymbol{x}_n)$$

$$\delta_i^2(\boldsymbol{x}_n) = \big(t_{ni} - s_i^2(\boldsymbol{x}_n)\big) \;\; f'(z_i^2(\boldsymbol{x}_n)) \;\; \text{with} \;\; z_i^2(\boldsymbol{x}_n)) = \sum_{j=0}^{M_1} w_{ij}^2 s_j^1(\boldsymbol{x}_n)$$

- Updating the weights of the hidden layer: $\big(1 \leq i \leq M_1,\ 0 \leq j \leq M_0\big)$:

$$\Delta w_{ij}^1 = -\rho \, \frac{\partial E_A(\mathbf{w})}{\partial w_{ij}^1} = \frac{\rho}{N} \sum_{n=1}^{N} \delta_i^1(\boldsymbol{x}_n) \; x_{nj}$$

$$\delta_i^1(\boldsymbol{x}_n) = \Big(\sum_{r=1}^{M_2} \delta_r^2(\boldsymbol{x}_n) \; w_{ri}^2\Big) \; f'(z_i^1(\boldsymbol{x}_n)) \;\; \text{with} \;\; z_i^1(\boldsymbol{x}_n) = \sum_{j=0}^{M_0} w_{ij}^1 x_{nj}$$

BackProp for N training samples for a three-layer perceptron

- Updating the weights of the output layer: $(1 \leq i \leq M_3,\ 0 \leq j \leq M_2)$

$$\Delta w_{ij}^3 = -\rho \, \frac{\partial E_A(\mathbf{w})}{\partial w_{ij}^3} = \frac{\rho}{N} \sum_{n=1}^{N} \delta_i^3(\boldsymbol{x}_n) \; s_j^2(\boldsymbol{x}_n) \qquad \delta_i^3(\boldsymbol{x}_n) = \Big(t_{ni} - s_i^3(\boldsymbol{x}_n)\Big) \; f'(z_i^3(\boldsymbol{x}_n))$$

- Updating the weights of the second hidden layer: $\Big(\ 1 \leq i \leq M_2,\ 0 \leq j \leq M_1\Big)$

$$\Delta w_{ij}^2 = -\rho \, \frac{\partial E_A(\mathbf{w})}{\partial w_{ij}^2} = \frac{\rho}{N} \sum_{n=1}^{N} \delta_i^2(\boldsymbol{x}_n) \; s_j^1(\boldsymbol{x}_n) \qquad \delta_i^2(\boldsymbol{x}_n) = \Big(\sum_{r=1}^{M_3} \delta_r^3(\boldsymbol{x}_n) \; w_{ri}^3\Big) \; f'(z_i^2(\boldsymbol{x}_n))$$

- Updating the weights of the first hidden layer: $\Big(\ 1 \leq i \leq M_1,\ 0 \leq j \leq M_0\Big)$

$$\Delta w_{ij}^1 = -\rho \, \frac{\partial E_A(\mathbf{w})}{\partial w_{ij}^1} = \frac{\rho}{N} \sum_{n=1}^{N} \delta_i^1(\boldsymbol{x}_n) \; x_{nj} \qquad \delta_i^1(\boldsymbol{x}_n) = \Big(\sum_{r=1}^{M_2} \delta_r^2(\boldsymbol{x}_n) \; w_{ri}^2\Big) \; f'(z_i^1(\boldsymbol{x}_n))$$

BackProp algorithm

Input: Topology, Initial weights w_{ij}^{l}, $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, learning rate ρ, Convergence conditions, N training samples A

Output: Weights of connections that minimize the mean squared error of A

While no convergence

For $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, initialize $\Delta w_{ij}^{l} = 0$

For each training sample $(\boldsymbol{x}, \boldsymbol{t}) \in A$

From the input to the output layers $(l = 0, \ldots, L)$:

For $1 \leq i \leq M_l$ if $l = 0$ then $s_i^0 = x_i$ else compute z_i^l y $s_i^l = f(z_i^l)$

From the output to the input layers $(l = L, \ldots, 1)$,

For each node $(1 \leq i \leq M_l)$

Compute $\delta_i^l = \begin{cases} f'(z_i^l)\ (t_{ni} - s_i^L) & \text{if } l == L \\ f'(z_i^l)\ \left(\sum_r \delta_r^{l+1}\ w_{ri}^{l+1}\right) & \text{otherwise} \end{cases}$

For each weight w_{ij}^l $(0 \leq j \leq M_{l-1})$ compute: $\Delta w_{ij}^l = \Delta w_{ij}^l + \rho\ \delta_i^l\ s_j^{l-1}$

For $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, update the weights: $w_{ij}^l = w_{ij}^l + \frac{1}{N}\ \Delta w_{ij}^l$

Computational cost for each iteration: $O(N\, D)$, $N = |A|$, $D =$ number of weights

Incremental BackProp algorithm

Input: Topology, initial weights w_{ij}^l, $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, learning rate ρ, convergence conditions, N training samples A

Salidas: Weights that minimize the mean squared error of A

While no convergence

For each training sample $(\boldsymbol{x}, \boldsymbol{t}) \in A$ (in random order)

From the input to the output layers $(l = 0, \ldots, L)$:

For $1 \leq i \leq M_l$ if $l = 0$ then $s_i^0 = x_i$ else compute z_i^l y $s_i^l = f(z_i^l)$

From the output to the input layer $(l = L, \ldots, 1)$,

For each node $(1 \leq i \leq M_l)$

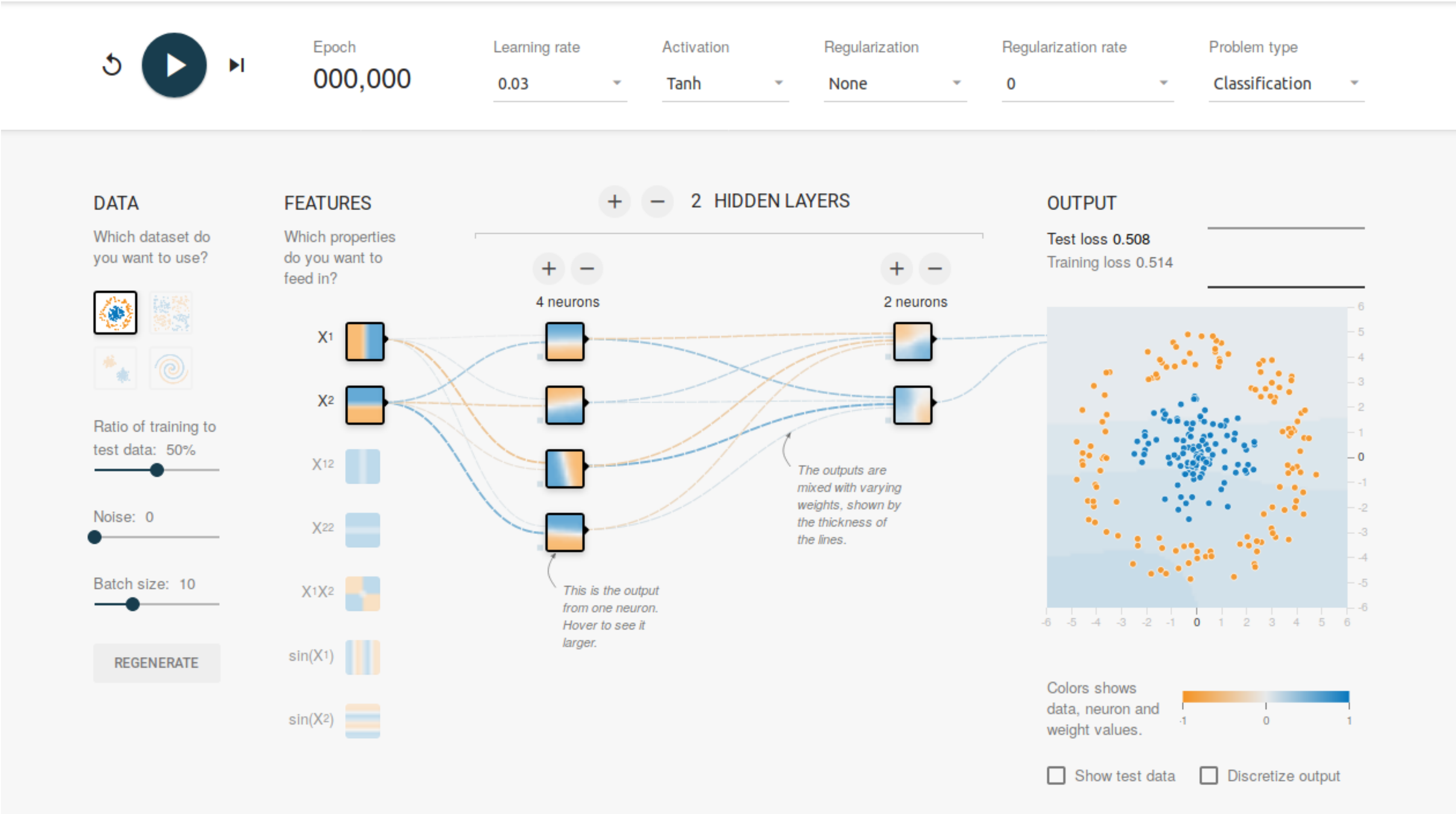
$$\text{Compute } \delta_i^l = \begin{cases} f'(z_i^l)\ (t_{ni} - s_i^L) & \text{if } l == L \\ f'(z_i^l)\ (\sum_r \delta_r^{l+1}\ w_{ri}^{l+1}) & \text{otherwise} \end{cases}$$

For each weight w_{ij}^l $(0 \leq j \leq M_{l-1})$ compute: $\Delta w_{ij}^l = \rho\ \delta_i^l\ s_j^{l-1}$

For $1 \leq l \leq L$, $1 \leq i \leq M_l$, $0 \leq j \leq M_{l-1}$, update weights: $w_{ij}^l = w_{ij}^l + \frac{1}{N}\ \Delta w_{ij}^l$

Computational cost for each iteration *mientras*: $O(N\ D)$, $N = |A|$, $D =$ number of weights

A demo of BackProp



http://playground.tensorflow.org/

A particular case: The Widrow-Hoff algorithm (Adaline)

- Given a training set $A = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$, with $\mathbf{x}_n \in \mathbb{R}^{D+1}$, $t_n \in \mathbb{R}$, search for $\mathbf{w} \in \mathbb{R}^{D+1}$: $\mathbf{w}^t\mathbf{x}_n = t_n$ (or $\mathbf{w}^t\mathbf{x}_n \approx t_n$) $1 \leq n \leq N$ ($\mathbf{x}_n = (1, x_{n_1}, x_{n_2} \ldots, x_{n_D})$ and $\mathbf{w} = (w_0, w_1, \ldots, w_D)$)

- Minimize the Widrow-Hoff function: $$J_A(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\mathbf{w}^t\mathbf{x}_n - t_n\right)^2$$

- Solution by gradient descent:

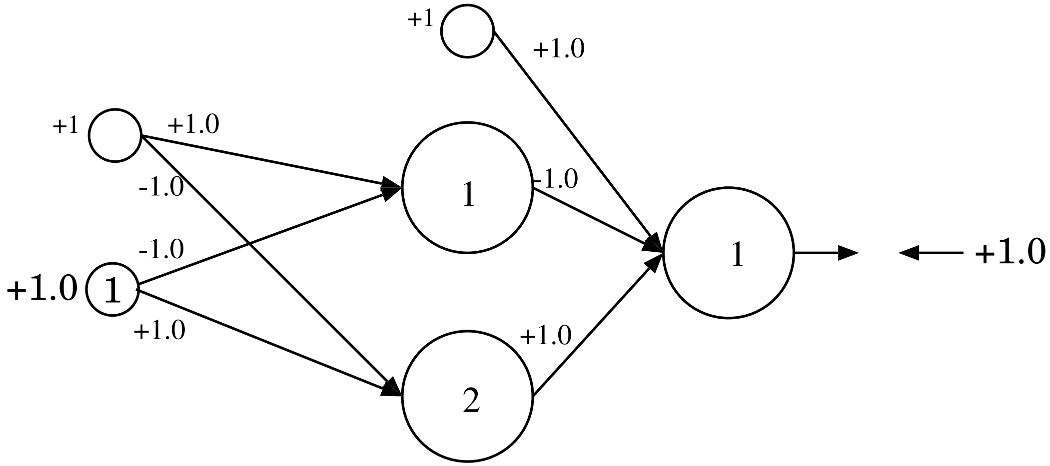
$$\begin{aligned} \mathbf{w}(1) &= \text{arbitrary} \\ \mathbf{w}(l+1) &= \mathbf{w}(l) - \rho_l \sum_{n=1}^{N}(\mathbf{w}(l)^t\, \mathbf{x}_n - t_n)\, \mathbf{x}_n \end{aligned}$$

- Correction sample by sample or on-line:

$$\begin{aligned} \mathbf{w}(1) &= \text{arbitrary} \\ \mathbf{w}(l+1) &= \mathbf{w}(l) + \rho_l \left(t(l) - \mathbf{w}(l)^t\mathbf{x}(l)\right)\, \mathbf{x}(l) \end{aligned}$$

Exercise

- The multiplayer perceptron of the figure is used for a regression problem with the hyperbolic tangent function as the activation function for all the nodes and a learning rate of $\rho = 0.5$.



Given an input sample $x = 1$ and the corresponding target $t = +1$, calculate:

a) The outputs of all the nodes
b) The corresponding errors in the output and hidden node
c) The new weights

BackProp for classification

- Softmax is used as the activation function in the output layer.

- Training

  Problem: Given a topology of a multilayer perceptron and $A = \{(\mathbf{x}_1, \mathbf{t}_1), \ldots, (\mathbf{x}_N, \mathbf{t}_N)\}$, with $\mathbf{x}_n \in \mathbb{R}^{M_0}$, $\mathbf{t}_n \in \{0,1\}^{M_2 \equiv C}$, search for $\mathbf{w}$ such that minimizes the objective function (cross-entropy):

$$C_A(\mathbf{w}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{M_2} t_{n,k} \, \log s_k^2(\mathbf{x}_n; \mathbf{w})$$

  Solution, gradient descent: $\Delta w_{i,j}^k = -\rho \, \dfrac{\partial C_A(\mathbf{w})}{\partial w_{i,j}^k}$

- The cross-entropy leads to faster training and improved generalization (Bishop 2006)

Derivation of BackProp for classification

- For $N = 1$, $C_A(\mathbf{w}) = -\sum_{k=1}^{M_2} t_k \ \log s_k^2(\mathbf{x};\mathbf{w})$
- Solution, gradient descent: $\Delta w_{i,j}^k = -\rho \dfrac{\partial C_A(\mathbf{w})}{\partial w_{i,j}^k}$
- Update weight of the output layer $w_{i,j}^2$ (if $N = 1$) $(z_k^2 = \sum_{m=1}^{M_1} w_{k,m}^2 s_m^1)$

$$C_A(\mathbf{w}) = \sum_{m=1}^{M_2} t_k \ \log s_k^2; \quad s_m^2 \ = f\left(z_m^2\right); \quad z_m^2 \ = f\left(\sum_{l=1}^{M_1} w_{m,l}^2 s_l^1\right)$$

$$\frac{\partial C_A}{\partial w_{i,j}^2} \ = \ \frac{\partial C_A}{\partial s_i^2} \frac{\partial s_i^2}{\partial z_i^2} \frac{\partial z_i^2}{\partial w_{i,j}^2} \ = \ \frac{t_i}{s_i^2} f'(z_i^2) \ s_j^1 \ = \ -\ \delta_i^2 \ s_j^1$$

$$\boxed{\Delta w_{i,j}^2 = -\rho \frac{\partial E_A}{\partial w_{i,j}^2} = -\rho \frac{t_i}{s_i^2} f'(z_i^2) \ s_j^1 = \rho \ \delta_i^2 \ s_j^1}$$

Convergence of the error backpropagation algorithm

GENERAL THEOREM OF CONVERGENCE: *Let λ_k be the eigenvalues of the matrix $\frac{\partial^2 E_A(\mathbf{w})}{\partial \omega_i \, \partial \omega_j}$ for a given $\mathbf{w}$. If $|1 - \lambda_k \rho| < 1 \; \forall k$, when the number of iterations tends to ∞, $\mathbf{w}$ tends to a local minimum of $E_A(\mathbf{w})$*

Learning factor

- $\rho < 2/\lambda_{max}$ (Bishop, 95)
- Large $\rho \Rightarrow$ fast convergence and tendency to oscillate.
- Small $\rho \Rightarrow$ slow convergence.

Probabilistic interpretation of the output of a multilayer perceptron

1. A multilayer perceptron of L layers as a classifier in C classes

2. $A = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$ with $\mathbf{x}_n \in \mathbb{R}^d, \mathbf{t}_n \in \mathbb{R}^C \; 1 \leq n \leq N$ and $t_{n,k} = 1$ if $c(\mathbf{x}_n) = k$ and $t_{n,k} = 0$ otherwise for $1 \leq k \leq C$

3. The squared error:

$$E_A(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} \sum_{k=1}^{C} \left(t_{n,k} - s_k^L(\mathbf{x}_n)\right)^2$$

ASSUMPTIONS: A has been generated following a distribution $\Pr(\mathbf{x}, \mathbf{t})$ and A is sufficiently huge and representative of $\Pr$:

If a global minimum of the mean squared error is reached and the a-posteriori probability is implementable by means of a multilayer perceptron, then, the outputs of the multilayer perceptron implement the a-posteriori probability underlying in the training samples: $s_k^L(\mathbf{x}) = \Pr(k \mid \mathbf{x})$.

Likelihood and squared error

- Assume that: $t_{n,k} = s_k^L(\mathbf{x}_n; \mathbf{w}) + \epsilon$, where ϵ is a Gaussian noise:

$$p(t_{n,k} \mid \mathbf{x}_n; \mathbf{w}) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left(-\frac{(s_k^L(\mathbf{x}_n; \mathbf{w}) - t_{nk})^2}{2\sigma^2}\right)$$

- As $\mathbf{t}_n \in \mathbb{R}^C$, let us assume that $p(\mathbf{t}_n|\mathbf{x}_n; \mathbf{w}) = \prod_{k=1}^{C} p(t_{n,k}|\mathbf{x}_n; \mathbf{w})$

- Given a training sample $A = \{(\mathbf{x}_1, \mathbf{t}_1), ..., (\mathbf{x}_N, \mathbf{t}_N)\}$ the maximum likelihood estimation of $\mathbf{w}$ is:

$$\begin{aligned}
\operatorname*{argmax}_{\mathbf{w}} \prod_{n=1}^{N} p(\mathbf{t}_n \mid \mathbf{x}_n; \mathbf{w}) &= \operatorname*{argmin}_{\mathbf{w}} \left(-\sum_{n=1}^{N} \log p(\mathbf{t}_n \mid \mathbf{x}_n; \mathbf{w})\right) \\
&= \operatorname*{argmin}_{\mathbf{w}} \left(\sum_{n=1}^{N} \sum_{k=1}^{C} (s_k^L(\mathbf{x}_n; \mathbf{w}) - t_{n,k})^2\right)
\end{aligned}$$

In this case, the maximum likelihood estimation conveys to a mean squared error minimization.

Index

1 Introduction ▷ 3

2 Multilayer Perceptron ▷ 7

3 Error Back-propagation algorithm ▷ 22

∘ 4 *Some issues about BackProp* ▷ 41

5 A brief introduction to deep neural networks ▷ 55

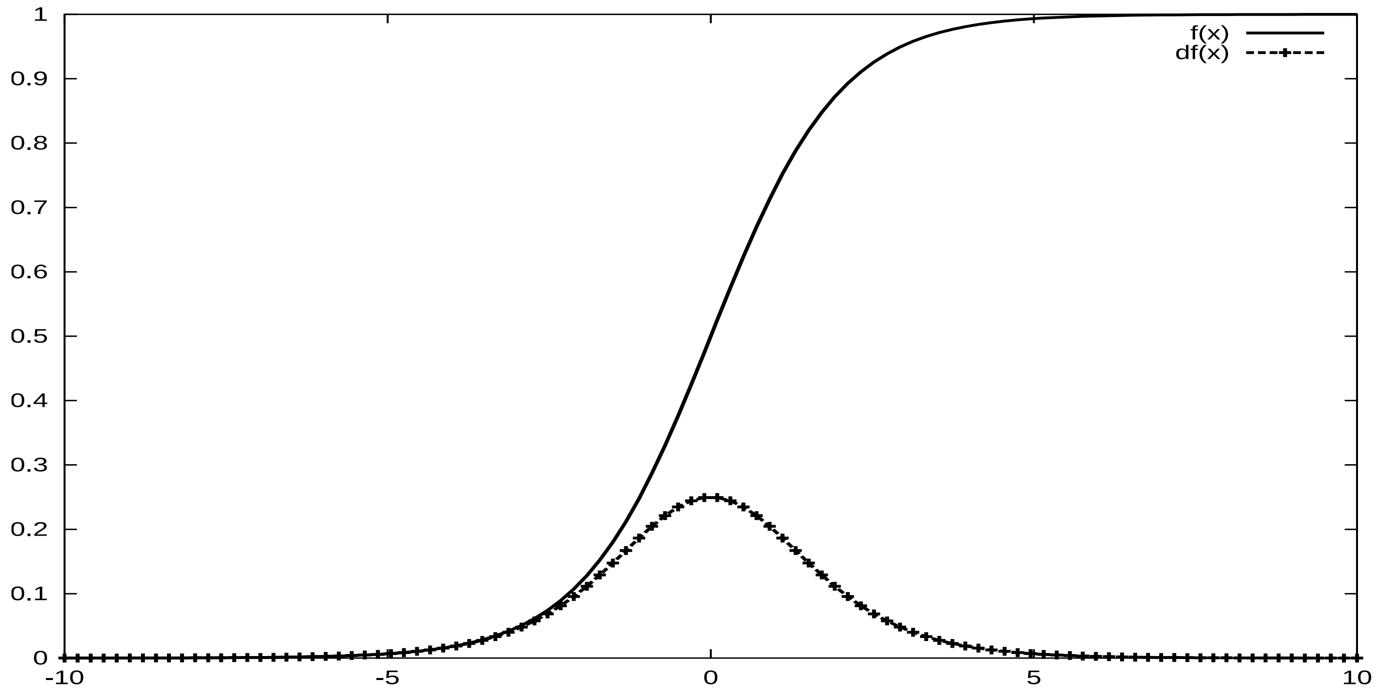
6 Bibliography ▷ 66

Derivatives of activation functions

- *Linear*: $f_L(z) = z \Rightarrow f'_L(z) = 1$, $z \in \mathbb{R}$.
- *Step*: $f_E(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0 \end{cases} \Rightarrow f'_E(z) = \begin{cases} 0 & \text{if } z > 0 \text{ or } z < 0 \\ \text{not deriv.} & z = 0 \end{cases}$
- *ReLU*: $f_R(z) = \max(0, z) \Rightarrow f'_R(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z < 0 \\ \text{not derivable} & z = 0 \end{cases}$
- *PReLU*: $f_{PR}(z) = \begin{cases} z & \text{if } z > 0 \\ a\ z & \text{if } z < 0 \end{cases} \Rightarrow f'_R(z) = \begin{cases} 1 & \text{if } z > 0 \\ a & \text{if } z < 0 \\ \text{not derivable} & z = 0 \end{cases}$
- *Sigmoid*: $f_S(z) = \frac{1}{1+\exp(-z)} \Rightarrow f'_S(z) = f_S(z)\ (1 - f_S(z))$
- *Hyperbolic tangent*: $f_T(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)} \Rightarrow f'_T(z) = 1 - (f_T(z))^2$
- *Softmax:* $f_{SM}(z_k) = \frac{\exp(z_k)}{\displaystyle\sum_{k'} \exp(z_{k'})} \Rightarrow f'_{SM}(z_k) = f_{SM}(z_k)\ (1 - f_{SM}(z_k))$

Network paralysis

$$f(x) = \frac{1}{1 + \exp(-x)} \qquad \frac{df(x)}{dx} = f(x)\,(1 - f(x))$$



PROBLEM: Vanishing and exploding gradientes

Batch and online BackProp

- Off-line or batch algorithm: 1 update of the weights by epoch.
- Mini-batch training: B a subset of A. Batch algorithm applied to B.
- Incremental algorithm: sample $\mathbf{x}(l)$ at iteration l. $|A|$ updates of the weights by epoch.
- Online algorithm: each sample $\mathbf{x}$ is used only once.

Gradient descent optimization algorithms (Ruder 2016)

- Stochastic gradient descent.
- Stochastic gradient descent with momentum.
- Adagrad (Adaptive Gradient)
- Adadelta (an extension of Adagrad)
- Adam (Adaptive Moment Estimation):
- Nesterov accelerated gradient, RMSProp, AdaMax, Nadam, ...

Gradient descent optimization algorithms (Ruder 2016) (1)

- Stochastic gradient descent: gradiente descent in the l mini-batch.

$$\Delta \mathbf{w}(l) = \rho \nabla_{\mathbf{w}} E_B(\mathbf{w}(l))$$

- Stochastic gradient descent with momentum:

$$\Delta \mathbf{w}(l) = \rho \nabla_{\mathbf{w}} E_B(\mathbf{w}(l)) + \gamma \Delta \mathbf{w}(l-1)$$

Gradient descent optimization algorithms (Ruder 2016) (2)

- Adagrad (Adaptive Gradient): ($\times \equiv$ element-wise product)

$$\mathbf{m}(l) = \mathbf{m}(l-1) + (\nabla_{\mathbf{w}} E_B(\mathbf{w}(l)) \times \nabla_{\mathbf{w}} E_B(\mathbf{w}(l)))$$

$$\mathbf{q}(l) : \forall i, \quad q_i(l) = \frac{\rho}{\sqrt{m_i(l) + \epsilon}}$$

$$\Delta\mathbf{w}(l) = \mathbf{q}(l) \times \nabla_{\mathbf{w}} E_B(\mathbf{w}(l))$$

Gradient descent opimization algorithms (Ruder 2016) (3)

- Adadelta (an extension of Adagrad);

$$\mathbf{m}(l) = \gamma_1 \mathbf{m}(l-1) + (1-\gamma_1)\left(\nabla_{\mathbf{w}} E_B(\mathbf{w}(l)) \times \nabla_{\mathbf{w}} E_B(\mathbf{w}(l))\right)$$

$$\mathbf{v}(l) = \gamma_2 \mathbf{v}(l-1) + (1-\gamma_2)(\Delta\mathbf{w}(l-1) \times \Delta\mathbf{w}(l-1))$$

$$\mathbf{q}(l) : \forall i, \quad q_i(l) = \rho \sqrt{\frac{v_i(l-1)}{m_i(l)+\epsilon}}$$

$$\Delta\mathbf{w}(l) = \mathbf{q}(l) \times \nabla_{\mathbf{w}} E_B(\mathbf{w}(l))$$

Gradient descent opimization algorithms (Ruder 2016) (4)

- Adam (Adaptive Moment Estimation):

$$\mathbf{m}(l) = \gamma_1 \mathbf{m}(l-1) + (1-\gamma_1)\left(\nabla_{\mathbf{w}} E_B(\mathbf{w}(l)) \times \nabla_{\mathbf{w}} E_B(\mathbf{w}(l))\right)$$

$$\mathbf{q}(l) : q_i(l) = \frac{1}{\sqrt{\frac{m_i(l)}{1-\gamma_1} + \epsilon}}$$

$$\mathbf{v}(l) = \frac{\rho}{1-\gamma_2}\left(\gamma_2 \mathbf{v}(l-1) + (1-\gamma_2)\nabla_{\mathbf{w}} E_B(\mathbf{w}(l))\right))$$

$$\Delta\mathbf{w}(l) = \mathbf{q}(l) \times \mathbf{v}(l)$$

Input normalization and weight initialization

- Input coding: Normalize the input range to $[0, 1]$.

$$A = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^D \Rightarrow \begin{cases} \mu_j = \frac{1}{N} \sum_{i=1}^{N} x_{i,j} \\ \\ \sigma_j^2 = \frac{1}{N-1} \sum_{i=1}^{N} (x_{i,j} - \mu_j)^2 \end{cases} \quad 1 \leq j \leq D$$

$$\forall \mathbf{x} \in \mathbb{R}^D, \mathbf{x}^N : x_j^N = \frac{x_j - \mu_j}{\sigma_j} \Rightarrow \begin{cases} \mu_j^N = 0 \\ \\ \sigma_j^N = 1 \end{cases} \quad \text{for } 1 \leq j \leq D$$

- Weight initialization: (n is the size of previous layer)

$$\left[-\frac{1}{\sqrt{n}}, +\frac{1}{\sqrt{n}} \right]$$

Regularization

- Problem: to prevent very big weights.

- Solution: add a regularization term to the goal function

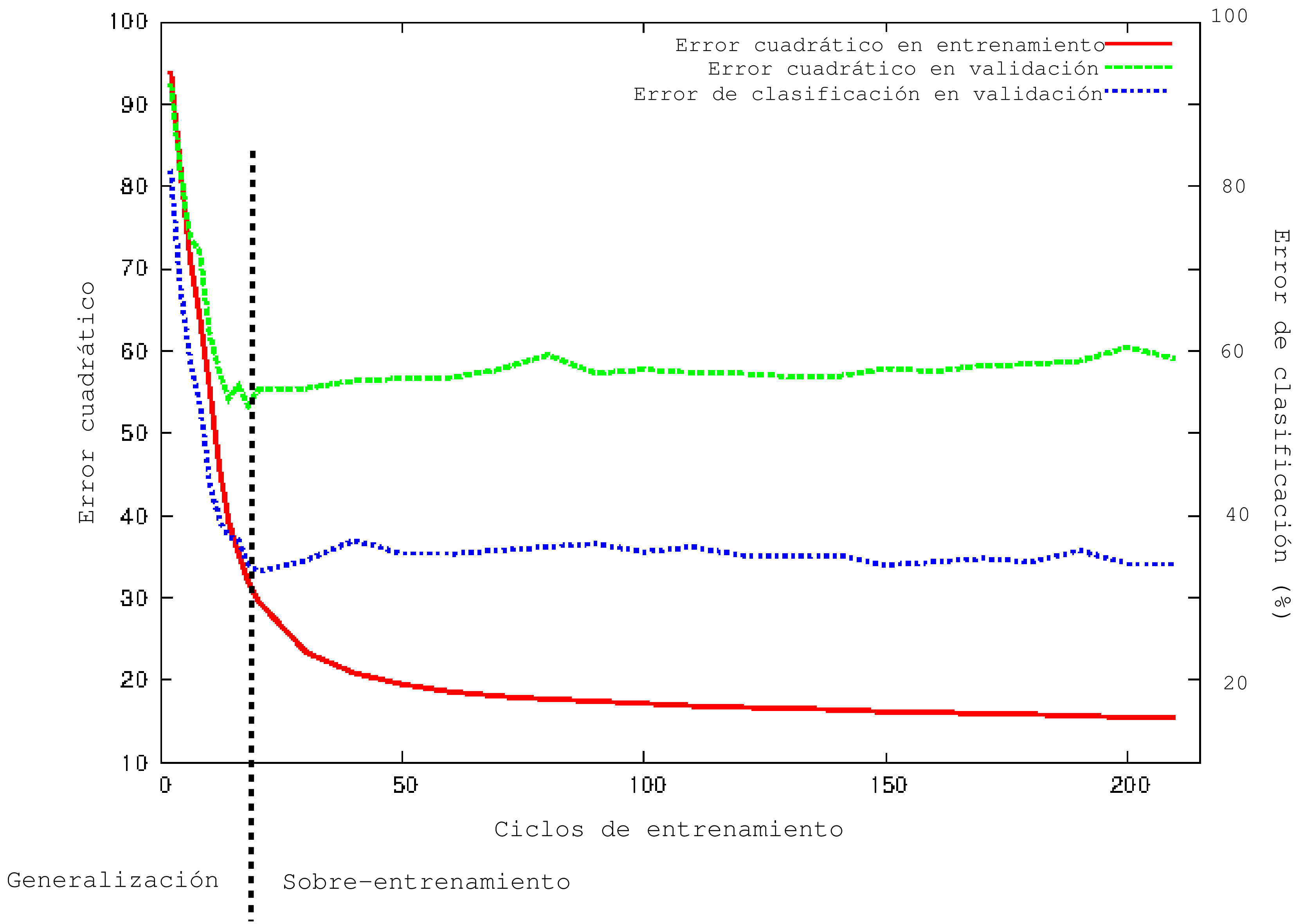
  - Regularization $L_2 : q_S(\boldsymbol{\Theta}) + \dfrac{\lambda}{2} \displaystyle\sum_{l,i,j} (\theta^l_{ij})^2$

  - Regularization $L_1 : q_S(\boldsymbol{\Theta}) + \dfrac{\lambda}{2} \displaystyle\sum_{l,i,j} \parallel \theta^l_{ij} \parallel$

Other techiques for avoiding "bad" local minimas [Koehn 2020]

- Shuffling the training data.
- Curriculum learning: From "easy" samples to "difficult" samples.
- Regularization: A new objective function to optimize, $E_A(\mathbf{w}) + \dfrac{\lambda}{2}\,||\mathbf{w}||^2$.
- Adding a Gaussian noise ϵ_k: $\Delta\omega^k_{i,j} = \rho\ \left(\delta^k_i\ s^{k-1}_j + \epsilon_l\right)$

Convergence conditions



Validation approaches

- *Resubstitution method*
  - The training set $=$ the test set
- *Hold-out method*
  - A training set and a validation set.
  - A test set for the evaluation.
- *Cross-validation*
  - Divide the training set in S parts.
  - For $i := 1$ to S
    Use $S - 1$ parts as training set (and validation) and the rest as a test set.
  - The result of the evaluation is the average of the results on the S repetitions.
- *Leave-one-out method* (Cross-validation with $S = N$)

Index

1 Introduction ▷ 3

2 Multilayer Perceptron ▷ 7

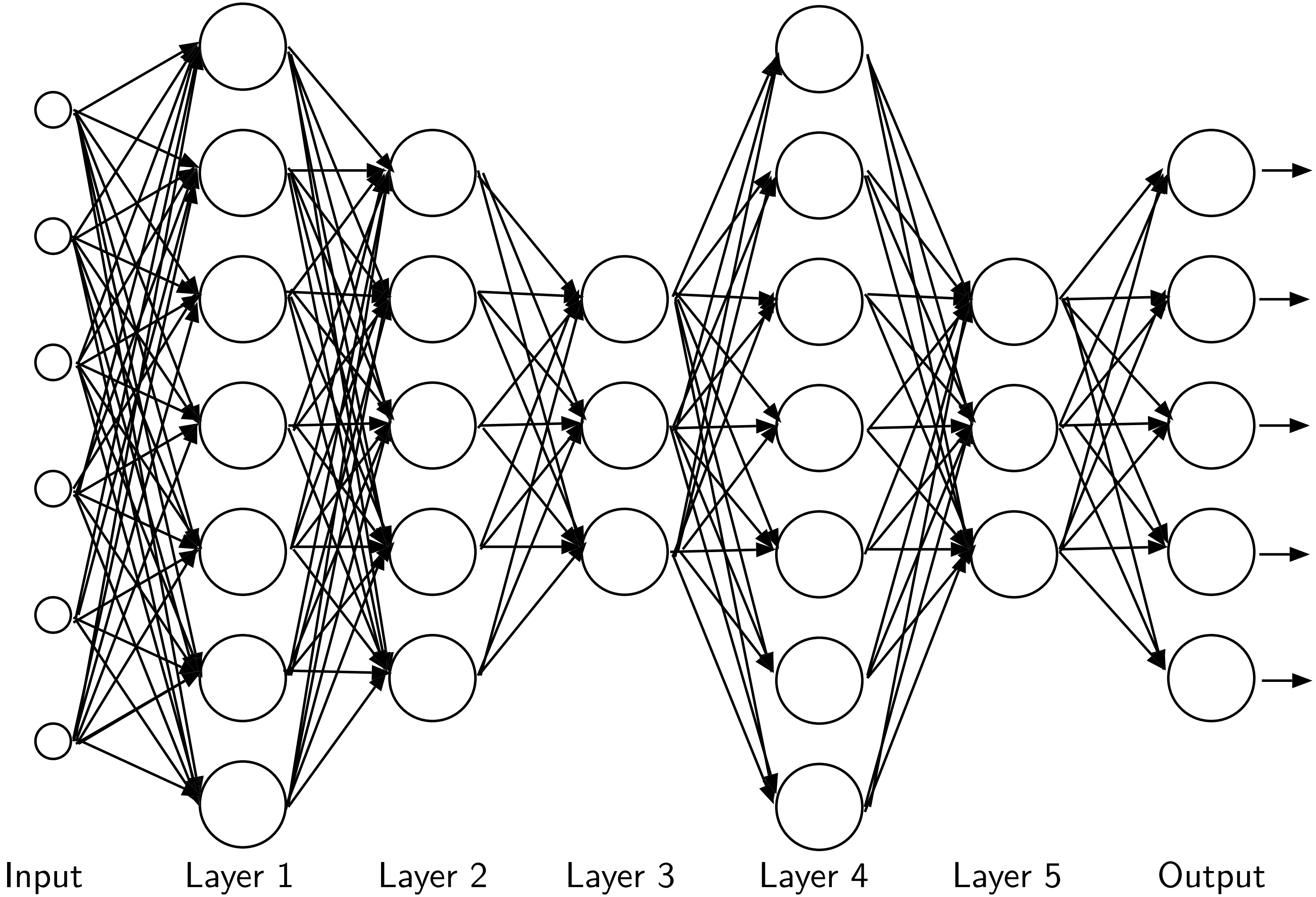
3 Error Back-propagation algorithm ▷ 22

4 Some issues about BackProp ▷ 41

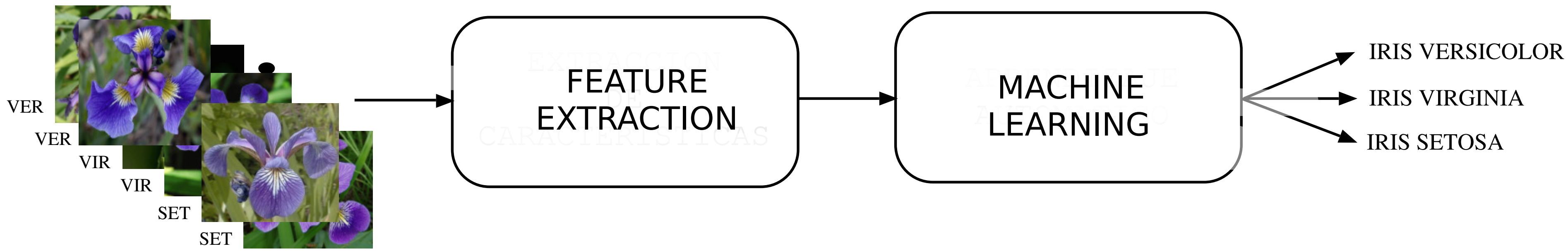
○ 5 *A brief introduction to deep neural networks* ▷ 55

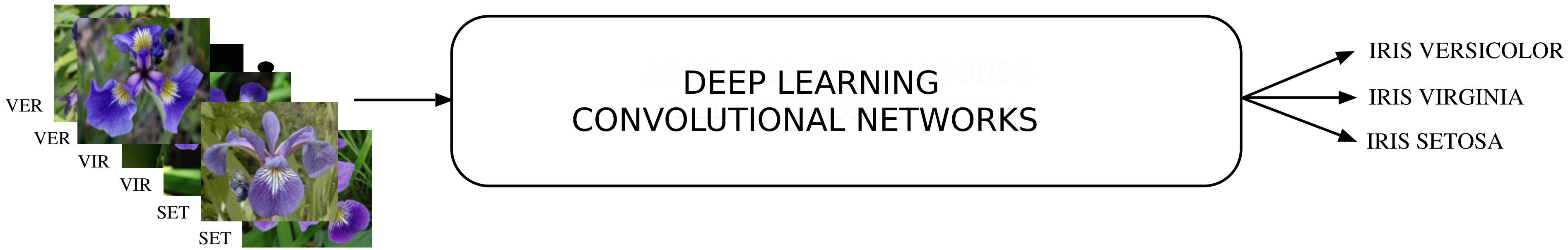
6 Bibliography ▷ 66

Deep neural network concept

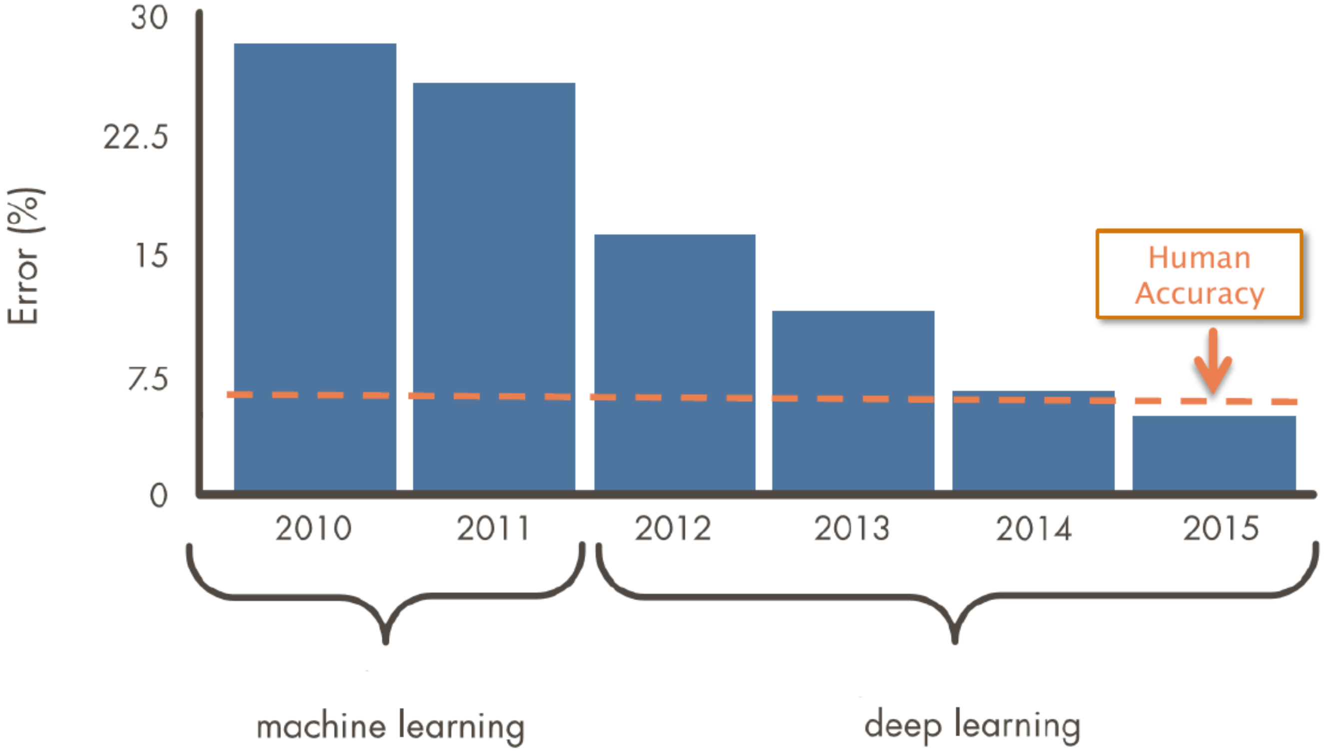


Deep learning



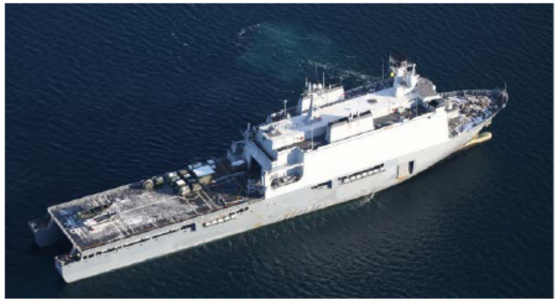


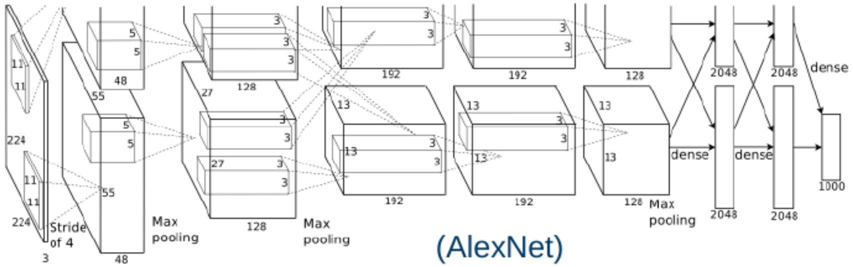
Deep learning [Daly 2017]



Source: ILSVRC Top-5 Error on ImageNet

Deep learning [Dyrdal 2019]





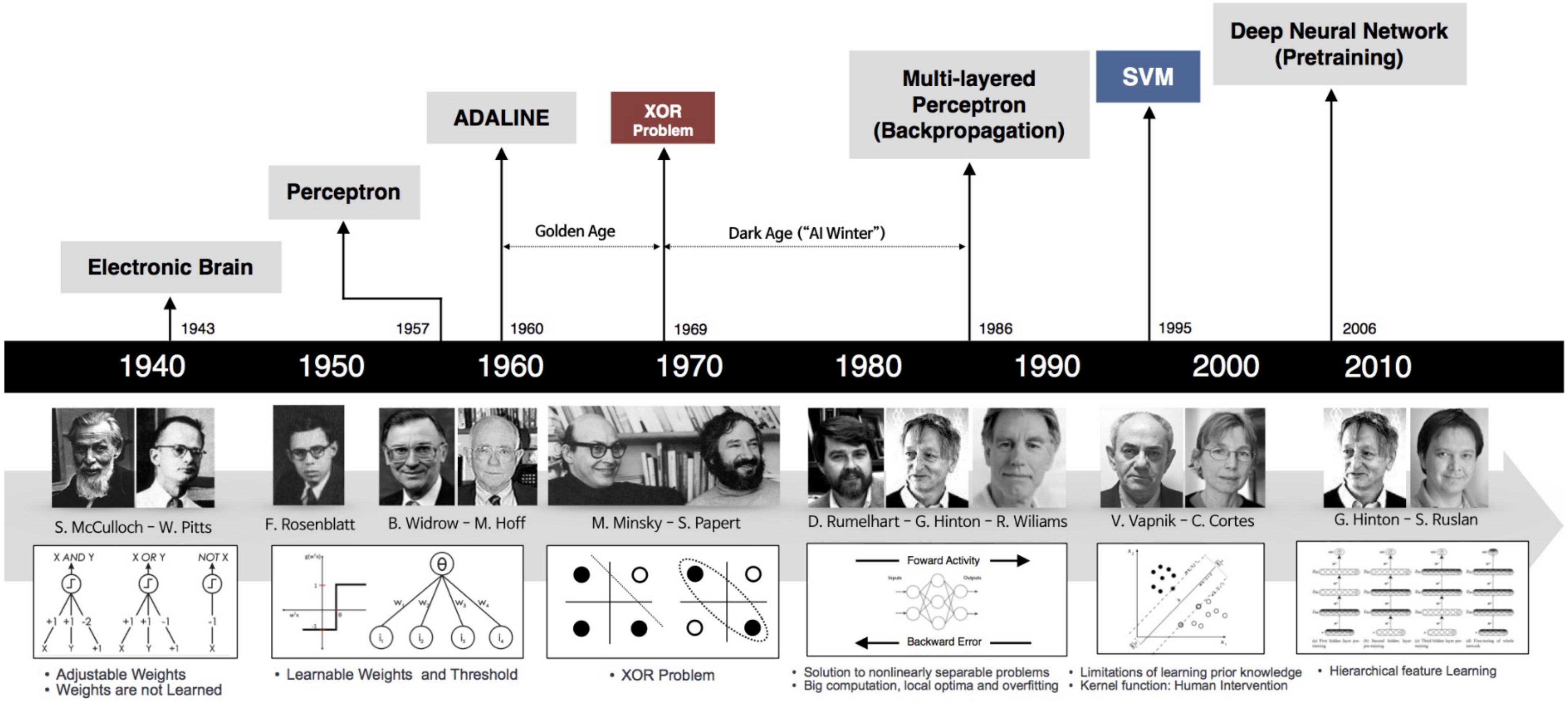
«Ship»

Millions of images

Millions of parameters

Thousands of classes

Deep learning [Serengil 2017]



Dynamic networks

- Recurrent networks:
  - Simple recurrent networks.
  - Elman network: recurrent + multiplayer perceptron,
  - Second order recurrent networks.
  - Long Short-Term Memory (LSTM)
  - Gated Recurrent Units (GRU)

- Feed-forward networks:
  - Convolutional networks.
  - Transformer (for text, images, tables, ...)
  - Pre-trained networks, usually based on Transformer: BERT (Google AI), GPT-3 (OpenAI), XML (Facebook), DALL-E 2 (OpenAI), BART (Facebook), Flamingo (DeepMind), ...
  - Prompting and few-shot (meta) learning.

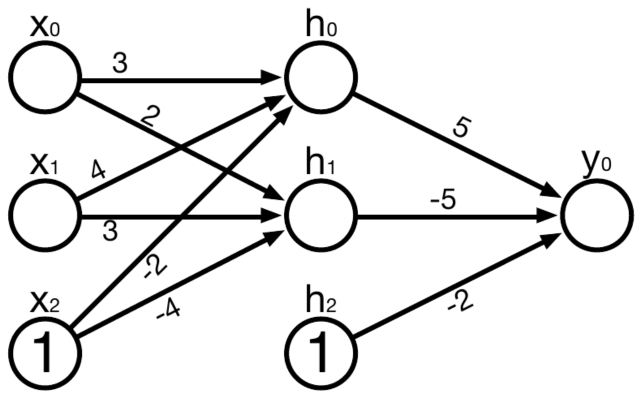
Avoiding vanishing and exploding gradientes [Koehn 2020]

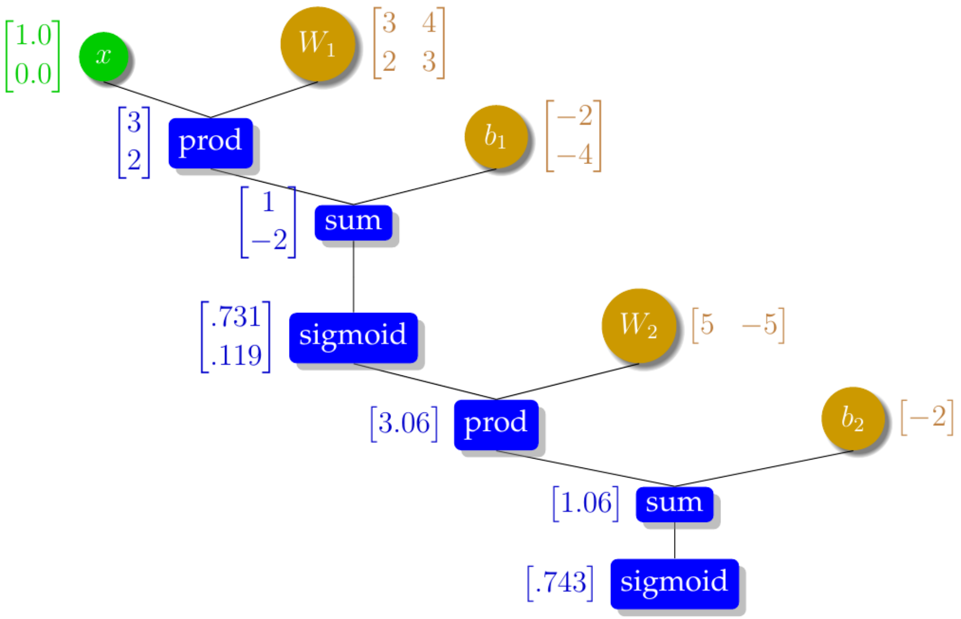
- Using activation functions such as Relu, LeakyRelu/PreLU, Maxout, ELU, ...
- Dropout: Randomly some nodes are ignored in each iteration.
- Adaptive gradient clipping.
- Residual connections: From $y = f(x)$ to $y = f(x) + x$.
- Highway networks: Residual networks+control gates.
- Batch normalization.
- Layer normalization.
- Input normalization.
- Weight initialization.
- Regularization.

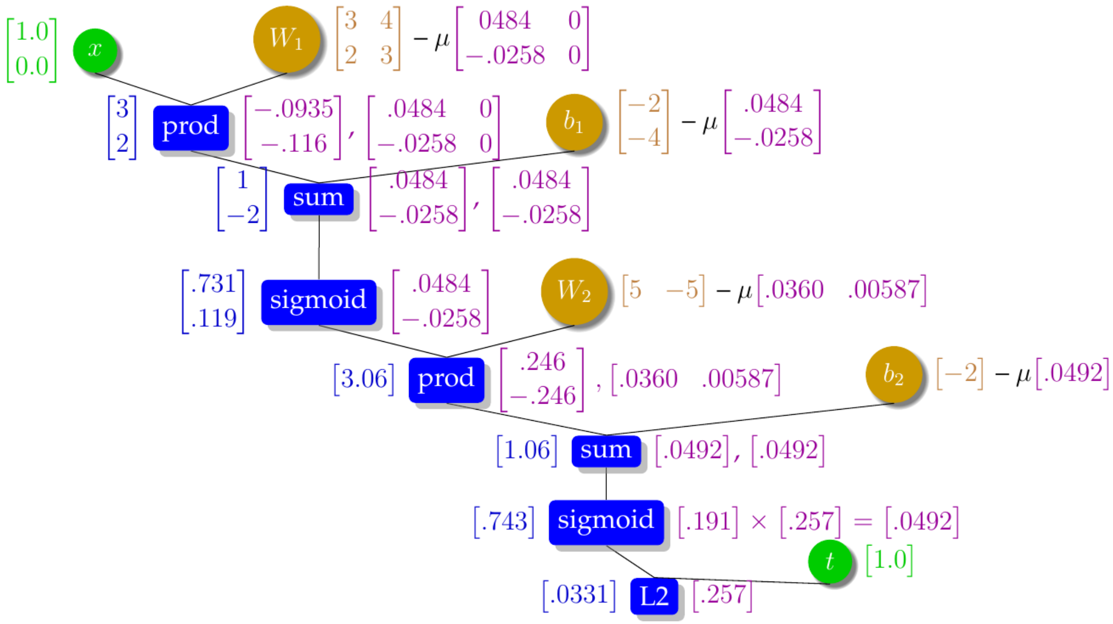
Other issues

- Training issues:
  - Data augmentation.
- Computational issues:
  - Using computational graphs implemented in TensorFlow, PyTorch, ... (Koehn 2020)
  - Use of graphics processing units (GPUs): the size of the minibatch is usual conditioned by the memory of the GPUs.

Computational graphs (Koehn 2020)







Computational graphs in PyTorch (http://www.statmt.org/nmt-book/)

```
import torch

# Data
W = torch.tensor([[3,4],[2,3]], requires_grad=True, dtype=torch.float)
b = torch.tensor([-2,-4], requires_grad=True, dtype=torch.float)
W2 = torch.tensor([5,-5], requires_grad=True, dtype=torch.float)
b2 = torch.tensor([-2], requires_grad=True, dtype=torch.float)
data = [ [ torch.tensor([0.,0.]), torch.tensor([0.]) ],
         [ torch.tensor([1.,0.]), torch.tensor([1.]) ],
         [ torch.tensor([0.,1.]), torch.tensor([1.]) ],
         [ torch.tensor([1.,1.]), torch.tensor([0.]) ] ]
mu = 0.1
```

```
for iteration in range(1000):
  # forward computation
  total_error = 0
  for item in data:
    x = item[0]
    t = item[1]
    s = W.mv(x) + b
    h = torch.nn.Sigmoid()(s)
    z = torch.dot(W2, h) + b2
    y = torch.nn.Sigmoid()(z)
    error = 1/2 * (t - y) ** 2
    total_error = total_error + error
  # backward computation
  total_error.backward()
  W.data = W - mu * W.grad.data
  b.data = b - mu * b.grad.data
  W2.data = W2 - mu * W2.grad.data
  b2.data = b2 - mu * b2.grad.data
  W.grad.data.zero_()
  b.grad.data.zero_()
  W2.grad.data.zero_()
  b2.grad.data.zero_()
  print("error: ",total_error.data/4)
```

Index

1 Introduction ▷ 3

2 Multilayer Perceptron ▷ 7

3 Error Back-propagation algorithm ▷ 22

4 Some issues about BackProp ▷ 41

5 A brief introduction to deep neural networks ▷ 55

∘ 6 *Bibliography* ▷ 66

Recommended bibliography

- Christopher M. Bishop. 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA.

- Christopher M. Bishop. 1995. Neural Networks for Pattern Recognition. Oxford University Press, Inc., New York, NY, USA.

- Ian Goodfellow, Yoshua Bengio and Aaron Courville. Deep Learning. MIT Press 2016. http://www.deeplearningbook.org

- Philipp Koehn. Neural Machine Translation. Cambridge University Press 2020