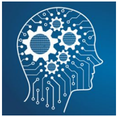
# Optimization for machine learning – Part 2 gradient calculation by backpropagation

Rodolphe Le Riche[1], Didier Rullière[2]

[1] CNRS LIMOS, Mines Saint-Etienne, UCA, France
[2] Mines Saint-Etienne, CNRS LIMOS



Dec 2025
Majeure Science des données (UP3) et Master Maths en Action
Mines Saint-Etienne

# Course program I

## **Part I.** Non stochastic optimization for ML (Rodolphe)

- Introduction
  Objectives – Optimization problem formulation – Examples of optimization usages – Basic mathematical concepts for optimization
- Steepest descent algorithm
  Fixed step steepest descent algorithm – Line search – convergence
- Improved gradient based searches
  Search directions for acceleration – Making it more global: restarts – A word about constraints – Towards ML: regularized quadratic function

# Course program II

## **Part II.** Stochastic optimization for ML (Didier)

- Stochastic Approximation SA, Stochastic Gradient Descent SGD.
- Robbins-Monro

## **Part III.** Non stochastic optimization for ML (cont., Rodolphe)

- Gradient calculation by backpropagation
- Application to neural network

## **Part IV.** Stochastic optimization for ML (cont., Didier)

- Unknown gradient. Neural Applications and Batches.
- Kiefer-Wolfowitz – Applications to ML

## Future lectures about optimization

# Course program III

- Global optimization
  Using metamodels – EGO – CMAES – Simulated Annealing... ← (UP4, upcoming)

# Reading material

Bibliographical references for this part of the class

- [Mallat, 2019] : an hindsight full video (in French) about backpropagation.
- [Bishop, 2006] : a reference book for machine learning with some pages on optimization (level end of undergraduate / bac+3)

# Application to neural network

Topics related to neural networks discussed so far:

$x$ , the weights and biases of the NN

$$\min_{x \in [LB,UB] \subset \mathbb{R}^D} f_\lambda(x) = f(x) + \lambda \|x\|_1$$

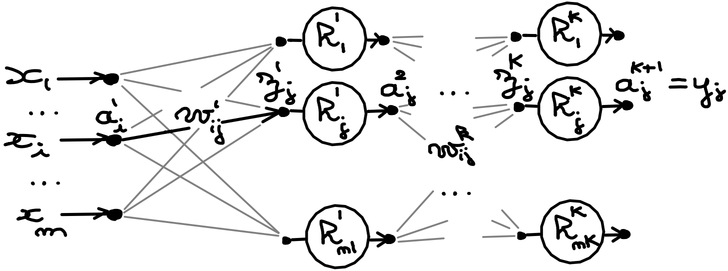
where, for classification,

$$f(x) = -\sum_{i=1}^{N} \{ t^i \log(y(e^i;x)) + (1-t^i) \log(1-y(e^i;x)) \}$$

and for regression,

$$f(x) = 1/2 \sum_{i=1}^{N} (t^i - y(e^i;x))^2$$

# Feedforward neural network: main relations



Mind the change of notation ☹

- $K$ layers, $k = 1, \ldots, K$
- Layer $k$, in matrix notation, $z^k \in \mathbb{R}^{n_k}$:

$$\begin{aligned} z^k &= {W^k}^\top a^k \\ a^{k+1} &= h^k(z^k) \end{aligned} \qquad (1)$$

- data : $(x^i, y^i)$ , $i = 1, \ldots, N$

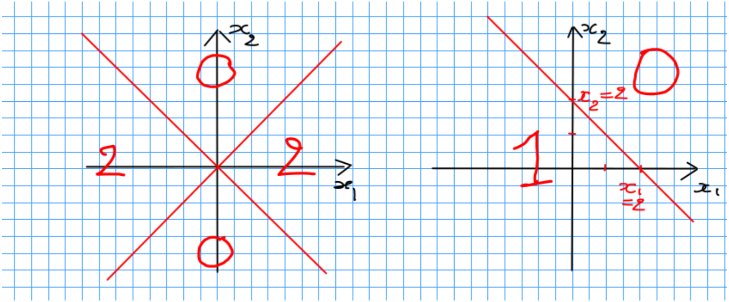
omitting data point index $i$:

- Inputs: $a^1 = x$
- Outputs $a^{K+1} = y$

# Neuron units

| name | $h(x)$ | comment |
|---|---|---|
| linear | $h(x) = x$ | |
| rectified linear unit, relu | $= 0$ if $x < 0$ , $= x$ otherwise | nonlinear, no plateau on 1 side, non differentiable, monotonic, $\in [0, +\infty[$ |
| leaky relu | $= \varepsilon x$ if $x < 0$ , $= x$ otherwise | $\varepsilon$ small positive, avoids null gradient of relu |
| sigmoid or logistic | $= 1/(1 + \exp(-x))$ | nonlinear, plateaus on both sides, $\in ]0, 1[$ good for probabilities, monotonic |
| hyperbolic tangent, tanh | $= (\exp(x) - \exp(-x))/(\exp(x) + \exp(-x))$ | like sigmoid but with a change of sign, $\in ] -1, 1[$ |

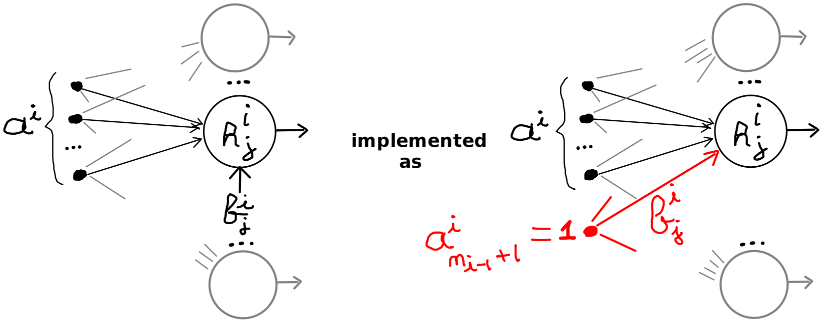
Neural nets by hand. Create 2 small networks that, roughly, separate the input space with such values of the output:




(no calculation, just thinking and drawing in 2D)

# Bias

Remove the bias from the notations by adding a fake 1 entry to each layer:




$W^i$ , $a^i$ , **becomes** $\begin{bmatrix} W^i \\ \beta^{i\,T} \end{bmatrix}$ , $\begin{bmatrix} a^i \\ 1 \end{bmatrix}$ ,

$a^{i+1} = h^i(W^{i\,T} a^i + \beta^i)$ $\qquad$ $a^{i+1} = h^i(W^{i\,T} a^i)$

# Loss function

Let's first consider regression in the least square sense.
Learning the network is done through the resolution of an optimization problem:

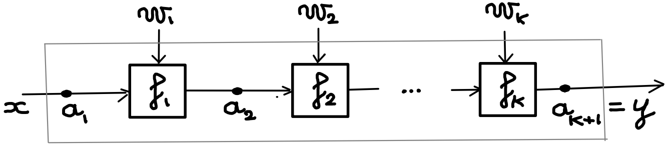
$$\min_{W^1,\ldots,W^K} L(W^1,\ldots,W^K)$$
$$\text{where } L(W^1,\ldots,W^K) = \frac{1}{2}\sum_{i=1}^{N}\|y(x^i;W^1,\ldots,W^K) - t^i\|^2 \tag{2}$$

- Statistical model parameters are written $\theta \equiv (W^1,\ldots,W^K) \equiv (W^1_{11}, W^1_{21},\ldots,W^K_{(n_{K-1}+1)n_K})$
- Optimization $\leftrightarrow$ machine learning notations: $f \leftrightarrow L$, $x \leftrightarrow \theta$.

Solve Pb (2) to learn the network : need $\partial L/\partial w^k_{ij} \Rightarrow$ backpropagation

# 1D network of functions

We present backpropagation in 2 main steps: first for any scalar (1D) functions, later for a neural net.




NN specific case :
$a_{i+1} = f_i(a_i, w_i) = h_i(w_i a_i) = h_i(z_i)$ where $z_i = w_i a_i$.

Need $\frac{\partial y}{\partial w_i}$ as $\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_i}$. Let's compare alternatives to do it.

# 1D network : finite differences

Say $K = 3$. The model is

$$\begin{aligned} a_1 &= x \\ a_2 &= f_1(a_1, w_1) \\ a_3 &= f_2(a_2, w_2) \\ y = a_4 &= f_3(a_3, w_3) \end{aligned}$$

where everything is scalar. Equivalently,

$$y(x, w_1, w_2, w_3) = \\ f_3(f_2(f_1(a_1, w_1), w_2), w_3)$$

1 forward propagation = K calls to $f$ (3 here)

**Finite difference**:
1 additional forward prop for each $w_i + \delta_i$ , $i = 1, \ldots, K$ $\Rightarrow$ $K^2$ extra calls to $f$ (9 in the example).

# 1D network: direct derivatives calculation

Chain rule differentiation (all derivatives evaluated at current $a^i, w^i$):

$$\frac{\partial y}{\partial w_1} = \frac{\partial f_3}{\partial a_3}\frac{\partial f_2}{\partial a_2}\frac{\partial f_1}{\partial w_1}$$
$$\frac{\partial y}{\partial w_2} = \frac{\partial f_3}{\partial a_3}\frac{\partial f_2}{\partial w_2}$$
$$\frac{\partial y}{\partial w_3} = \frac{\partial f_3}{\partial w_3}$$

Direct programming of the derivatives leads to 6 derivatives calculated.
In general, $1 + 2 + \ldots + K = K(K+1)/2 = (K^2 + K)/2$ calls to the derivatives (assume they have the same numerical cost as $f$).

# 1D network: backpropagation I

Backpropagation = partial derivatives + algorithmic recycling of the calculations. Derivatives calculated by backwards propagation of $\delta_k$.

Forward propagation:

$a_1 = x$
for $k = 1, ..., K$
$\quad a_{k+1} = f_k(a_k, w_k)$
end

Now all $a_i$'s known, calculate derivatives there

Backpropagation:

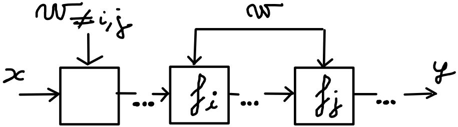
$\delta_{K+1} = 1$
for $k = K, \ldots, 2$
$\quad \delta_k = \delta_{k+1} \frac{\partial f_k}{\partial a_k} \; (= \frac{\partial y}{\partial a_k})$
$\quad \frac{\partial y}{\partial w_k} = \delta_{k+1} \frac{\partial f_k}{\partial w_k}$
end
$\partial y / \partial w_1 = \delta_2 \partial f_1 / \partial w_1$

---

5 derivatives calculated with $K = 3$.
$2(K-1)+1 = 2K-1$ derivatives in general. $\Rightarrow$ linear increase in $K$ for backprop against quadratic otherwise.

# 1D network: backpropagation II

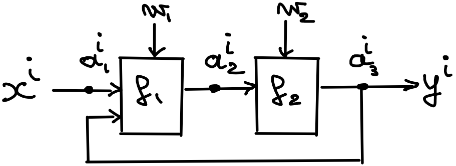
Equal weights. If the layers $i$ and $j$, $i \neq j$, have the same weights, $w_i = w_j = w$, what is the expression for $\partial y / \partial w$?



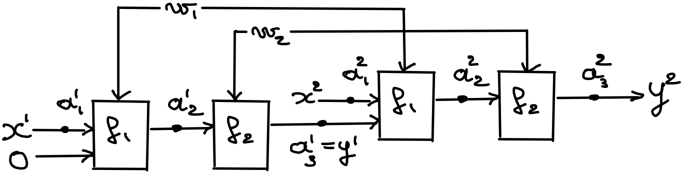

Solution: The response is given by the next slides about the backpropagation through time.

# 1D backpropagation through time I

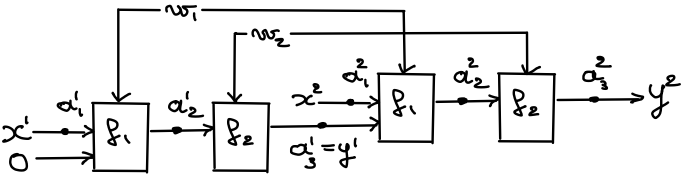
Data is ordered by time, $\left[(x^1, y^1), (x^2, y^2), \ldots, (x^T, y^T)\right]$, and memory is modeled by backwards links. Example (time$=i$):




Unfold the network and apply backpropagation (here $T = 2$):



# 1D backpropagation through time II




Partial derivation can be applied as before, accounting for the equality in weights at different times:

$$\frac{\partial y^1}{\partial w_2} = \frac{\partial f_2(a_2^1, w_2)}{\partial w_2} \quad , \quad \frac{\partial y^1}{\partial w_1} = \frac{\partial f_2(a_2^1, w_2)}{\partial a_2} \frac{\partial f_1(0, x^1, w_1)}{\partial w_1}$$

$$\frac{\partial y^2}{\partial w_2} = \frac{\partial f_2(a_2^2, w_2)}{\partial w_2} + \frac{\partial f_2(a_2^2, w_2)}{\partial a_2} \frac{\partial f_1(a_3^1, x^2, w_1)}{\partial a_3} \frac{\partial f_2(a_2^1, w_2)}{\partial w_2}$$

QUIZZ **Backpropagation through time.** Write $\frac{\partial y^2}{\partial w_1}$ for the above example.

# Backpropagation for a feedforward NN: multi-dimension

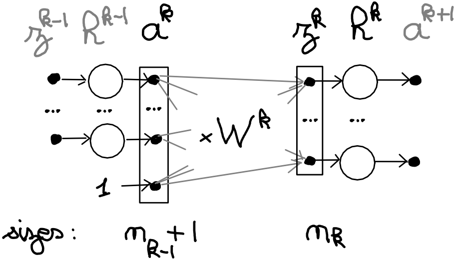
2 extensions: 1) multi-input multi-output layers 2) derivatives specific to a NN.

1) The previous 1D relations generalize to any dimension by replacing the partial derivatives by Jacobians :

$$\left[\frac{\partial f}{\partial a}\right]_{ij} = \frac{\partial f_i}{\partial a_j} \tag{3}$$

Example: $f$ and $g$ functions $\mathbb{R}^2 \to \mathbb{R}^2$, derivative of $g(f(a))$.
(to make the text simpler, the argument at which the derivatives are evaluated are not written) $\frac{\partial g}{\partial a} = \frac{\partial g}{\partial f}\frac{\partial f}{\partial a}$, replace by the $2 \times 2$ Jacobians and develop the product to get, e.g., $\left[\frac{\partial g}{\partial a}\right]_{12} = \frac{\partial g_1}{\partial a_2} = \frac{\partial g_1}{\partial f_1}\frac{\partial f_1}{\partial a_2} + \frac{\partial g_1}{\partial f_2}\frac{\partial f_2}{\partial a_2}$

# Backpropagation for a FFNN: derivatives I




$$z^k = {W^k}^\top a^k$$
$$a^{k+1} = h^k(z^k)$$

! The gradient of a scalar function is a row with Jacobian notation:
$\frac{\partial L}{\partial W^k}$ is $1 \times ((n_{k-1}+1)n_k)$
$\frac{\partial z^j}{\partial z^k}$ is $n_j \times n_k$
$\frac{\partial y}{\partial a^k}$ is $n_K \times (n_{k-1}+1)$

# Backpropagation for a FFNN: derivatives II

We ultimately look for the derivatives of the loss

$$L(W) = \frac{1}{2}\|y(x;W) - t\|^2 \quad , \quad \frac{\partial L}{\partial W^k_{ij}} = \; ? \tag{4}$$

Calculating the derivatives w.r.t. the layers states $z^k$ (and their flow through the network) is sufficient:

$$\frac{\partial L}{\partial W^k_{ij}} = \frac{\partial L}{\partial z^k}\frac{\partial z^k}{\partial W^k_{ij}} = \frac{\partial L}{\partial z^k_j} a^k_i$$

because $\partial z^k / \partial W^k_{ij}$ is a column of 0's except at the $j$-th component where it is $= a^k_i$. Summing up in matrix notation,

$$\frac{\partial L}{\partial W^k} = a^k \frac{\partial L}{\partial z^k} \qquad \text{which is } ((n_{k-1}+1) \times n_k) \tag{5}$$

# Backpropagation for a FFNN: derivatives III

Let study $\delta^k := \partial L / \partial z^k$ . Other definitions for $\delta^k$ ($\partial L/\partial a^k$, $\partial y/\partial z^k$, $\partial y/\partial a^k$) are possible. Go backwards with partial derivatives:

$$
\begin{aligned}
\delta^k = \frac{\partial L}{\partial z^k} &= \frac{\partial L}{\partial z^{k+1}} \frac{\partial z^{k+1}}{\partial z^k} \\
&= \delta^{k+1} \frac{\partial z^{k+1}}{\partial a^{k+1}} \frac{\partial a^{k+1}}{\partial z^k} \\
&= \delta^{k+1} {W^{k+1}}^\top \begin{bmatrix} \operatorname{diag}({h^k}'(z^k)) \\ 0 \end{bmatrix}
\end{aligned} \tag{6}
$$

QUIZZ Dimensions. Check the dimensions of the matrices in Eq. (6).

# Backprop for a feedforward NN: summing up

- Given a data point $(x, t)$, do a forward propagation and save the states $a^k$, $z^k$, $y$.
- Initialization :
  $\delta^{K+1} = \partial L / \partial z^{K+1} = \partial L / \partial y = (y(x) - t)^\top$
- $$\begin{array}{l} \text{for } k = K, \ldots, 1 \\ \qquad \delta^k = \delta^{k+1} {W^{k+1}}^\top \begin{bmatrix} \text{diag}({h^k}'(z^k)) \\ 0 \end{bmatrix} \\ \qquad \partial L / \partial W^k = a^k \delta^k \\ \text{end} \end{array}$$

What about several entries, $x^i$ , $i = 1, \ldots, N$?

# Batch backpropagation

Because the loss is a sum over the data set,

$$\frac{\partial L}{\partial W^k} = \sum_{i=1}^{N} \frac{\partial L}{\partial W^k} \text{ (for each } (x^i, t^i)) \tag{7}$$

---

- Initialize $\partial L / \partial W^k = 0$ , $k = 1, K$
- for each data point $(x^i, t^i)$
  - Do a forward propagation at $x^i$ and save the states $a^k$, $z^k$, $y$.
  - $\delta^{K+1} = \partial L / \partial z^{K+1} = \partial L / \partial y = (y - t^i)^\top$
  - for $k = K, \ldots, 1$

    $$\delta^k = \delta^{k+1} W^{k+1\top} \begin{bmatrix} \operatorname{diag}(h^{k\prime}(z^k)) \\ 0 \end{bmatrix}$$

    $$\partial L / \partial W^k = \partial L / \partial W^k + a^k \delta^k$$

    end
- end , return $\partial L / \partial W^k$ , $k = 1, K$

# References I


Bishop, C. M. (2006).
Pattern recognition and machine learning.

Mallat, S. (2019).
Descente de gradient et rétropropagation du gradient.
Collège de France course, https://www.college-de-france.fr/agenda/cours/apprentissage-par-reseaux-de-neurones-profonds/descente-de-gradient-et-retro-propagation-du-gradient, in French.