

SAISON
23/24



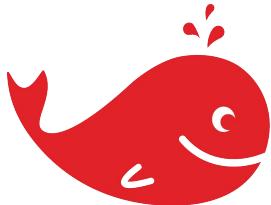
Formation
Introduction au
Deep Learning
Séquence n°5

“Opening the black box !”



FIDLE





FIDLE

Formation

Introduction au Deep Learning

Questions and answers :

<https://fidle.cnrs.fr/q2a>

Accompanied by :

IA Support (dream) Team of IDRIS

Directed by :

Agathe, Baptiste et Yanis - UGA/DAPI

Thibaut, Kamel - IDRIS



Formation

Introduction au Deep Learning



<https://fidle.cnrs.fr/listeinfo>

Fidle information list

Agoria

<http://fidle.cnrs.fr/agoria>

AI exchange list





Formation

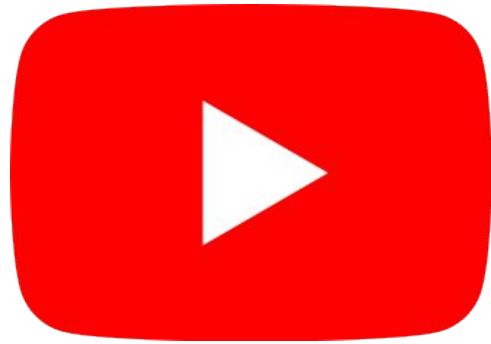
Introduction au Deep Learning



<https://listes.services.cnrs.fr/wws/info/devlog>
List of ESR* « Software developers » group

<https://listes.math.cnrs.fr/wws/info/calcul>
List of ESR* « Calcul » group

(*) ESR is Enseignement Supérieur et Recherche, french universities and public academic research organizations



YouTube

Abonnez-vous !



<https://fidle.cnrs.fr/youtube>

<https://www.youtube.com/@CNRS-FIDLE>



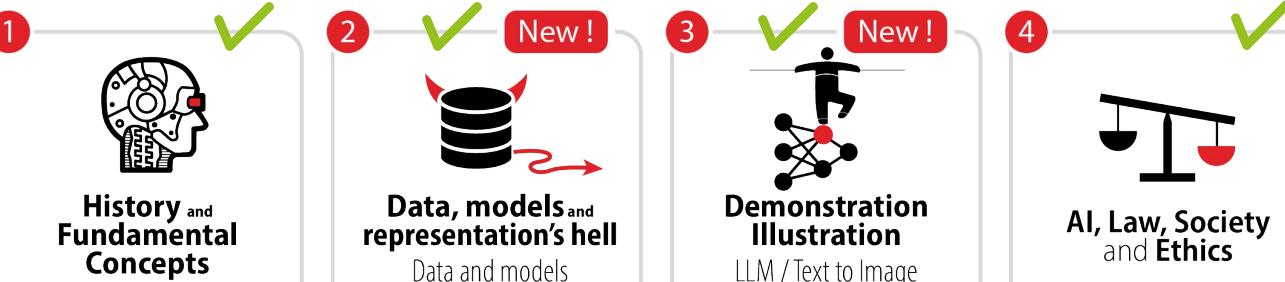
Le magazine de actualité en IA

Vendredi 19 janvier, 10h00

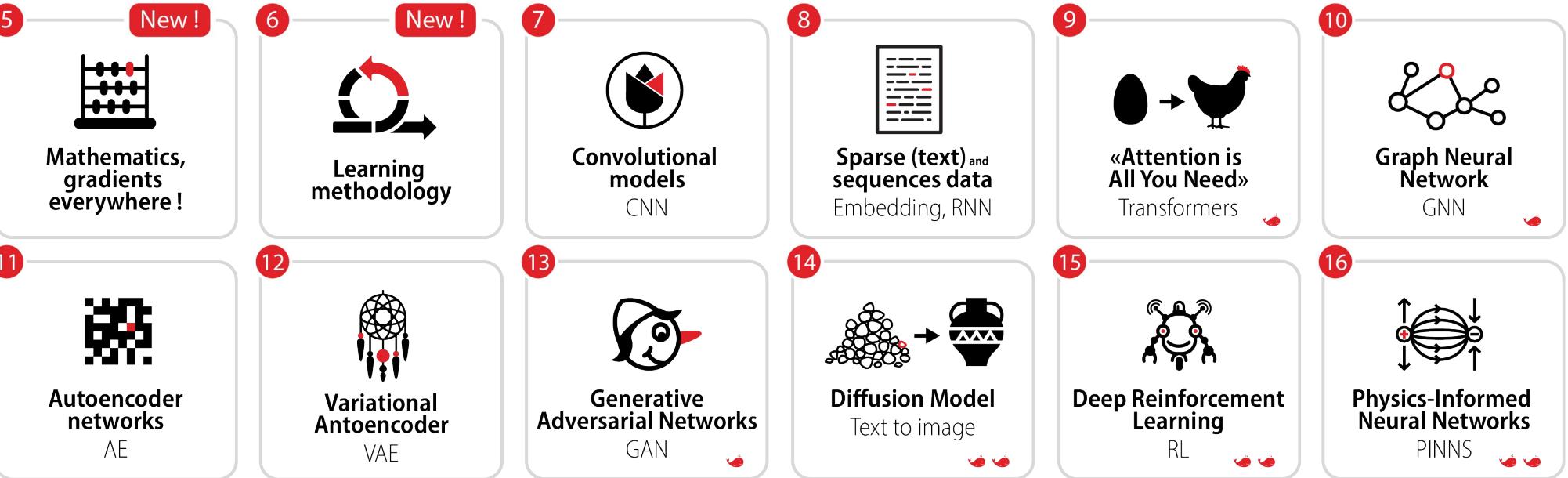
<http://www.idris.fr/panoramia.html>

<https://www.youtube.com/@IDRISCNRS>

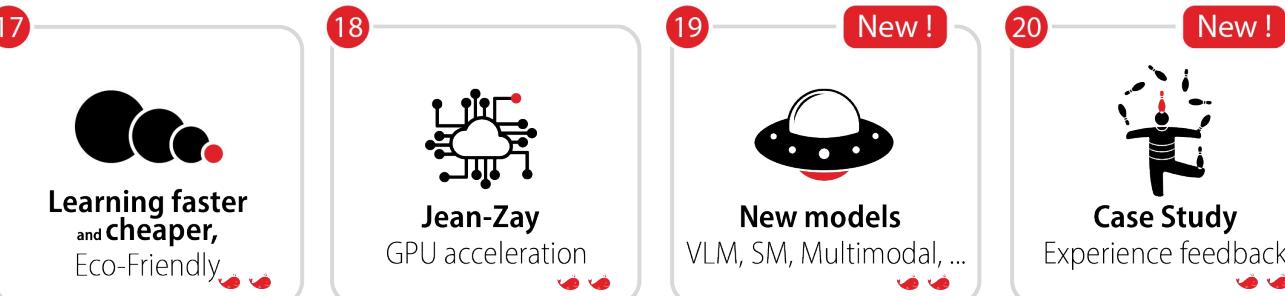
Bases, Concepts et Enjeux



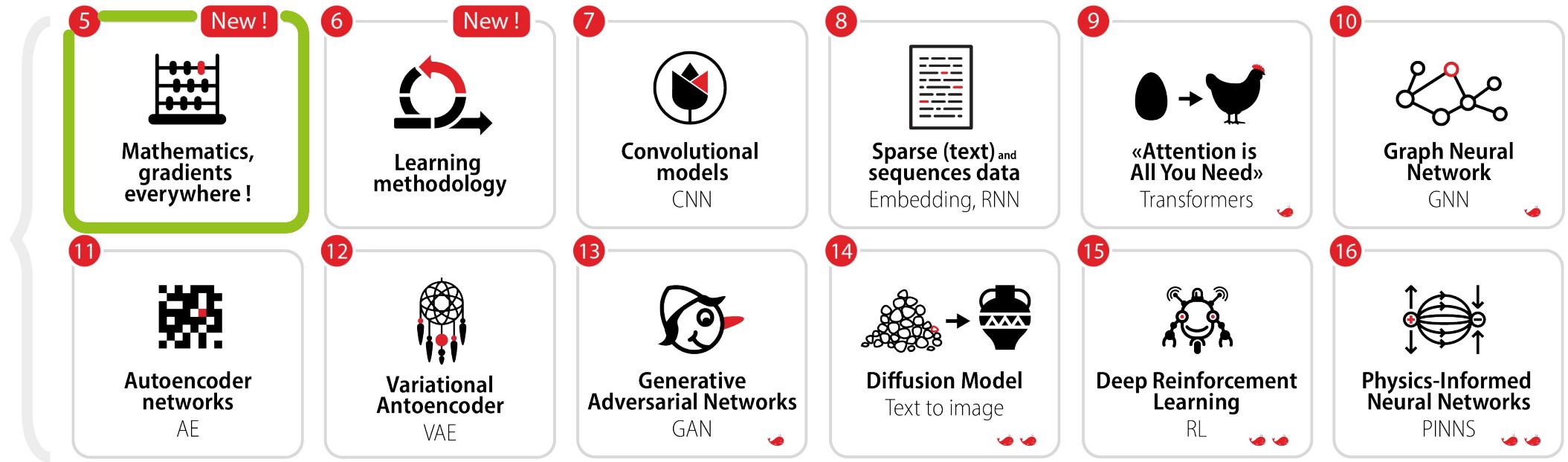
L'IA comme un outil,



Acteur de l'IA



Second part :



Objectif → utiliser l'IA comme un outil !



Introduction



Laurent Risser
(IR CNRS Maths)



Canal du midi à Toulouse



*Institut de Mathématiques de
Toulouse (IMT)*



*3IA Artificial and Natural Intelligence
Toulouse Institute (ANITI)*



Introduction

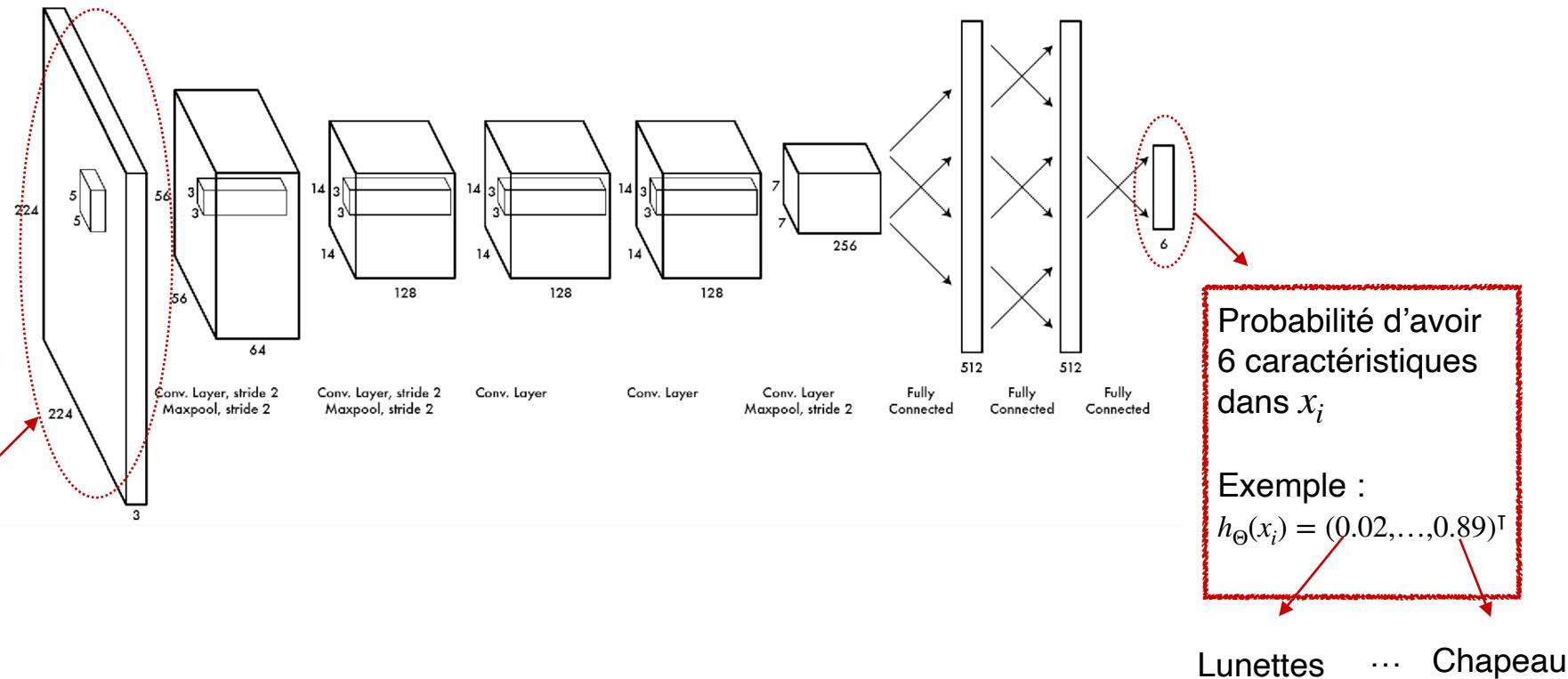


Image du réseau : [Redmon and Angelova, IEEE ICRA 2015]
Figure : Jeu de données CelebA

Paramètres du réseau appris à l'aide de descente de gradient stochastique avec des mini-batches de taille 50.

- Comment le réseau transforme couche après couche une image en probabilités ?
- Que-ce qu'une descente de gradient ?
- Comment calculer le gradient des paramètres d'un réseau de neurones (back-propagation) ?
- Stochastique... c'est quoi et quel est l'intérêt ?

Introduction

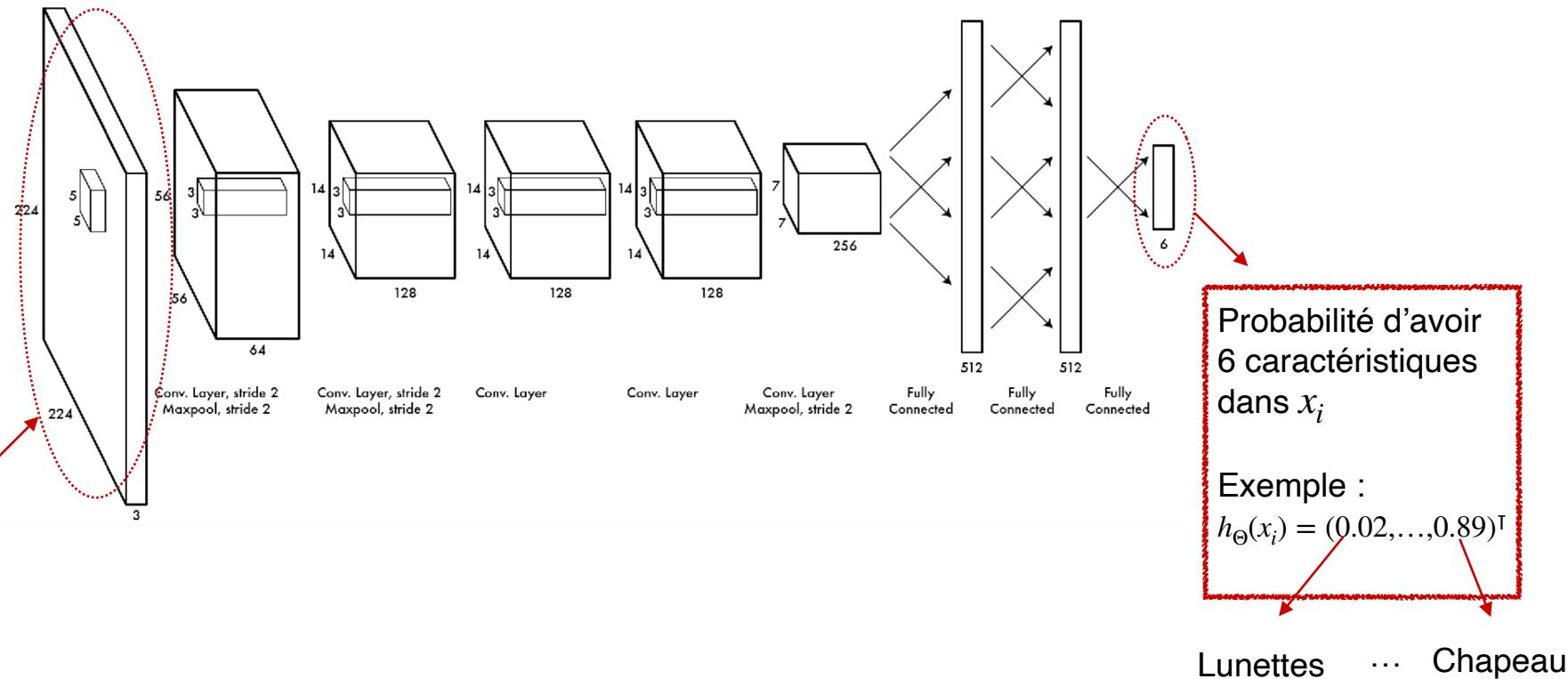


Image du réseau : [Redmon and Angelova, IEEE ICRA 2015]
Figure : Jeu de données CelebA

Paramètres du réseau appris à l'aide de descente de gradient stochastique avec des mini-batches de taille 50.

- Comment le réseau transforme couche après couche une image en probabilités ?
- Que ce qu'une descente de gradient ?
- Comment calculer le gradient des paramètres d'un réseau de neurones (back-propagation) ?
- Stochastique... c'est quoi et quel est l'intérêt ?

Introduction

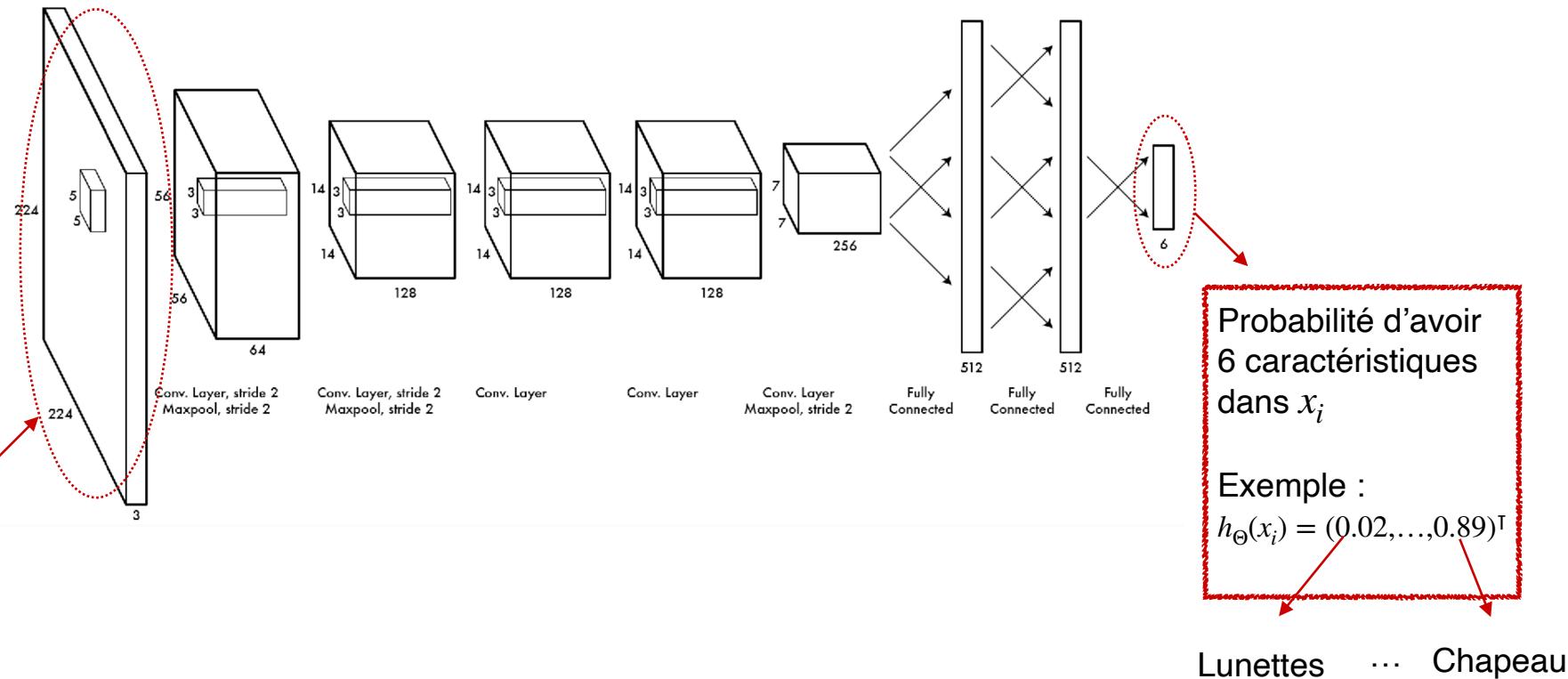


Image du réseau : [Redmon and Angelova, IEEE ICRA 2015]
Figure : Jeu de données CelebA

Paramètres du réseau appris à l'aide de descente de gradient stochastique avec des mini-batches de taille 50.

- Comment le réseau transforme couche après couche une image en probabilités ?
- Que-ce qu'une descente de gradient ?
- Comment calculer le gradient des paramètres d'un réseau de neurones (back-propagation) ?
- Stochastique... c'est quoi et quel est l'intérêt ?

Introduction

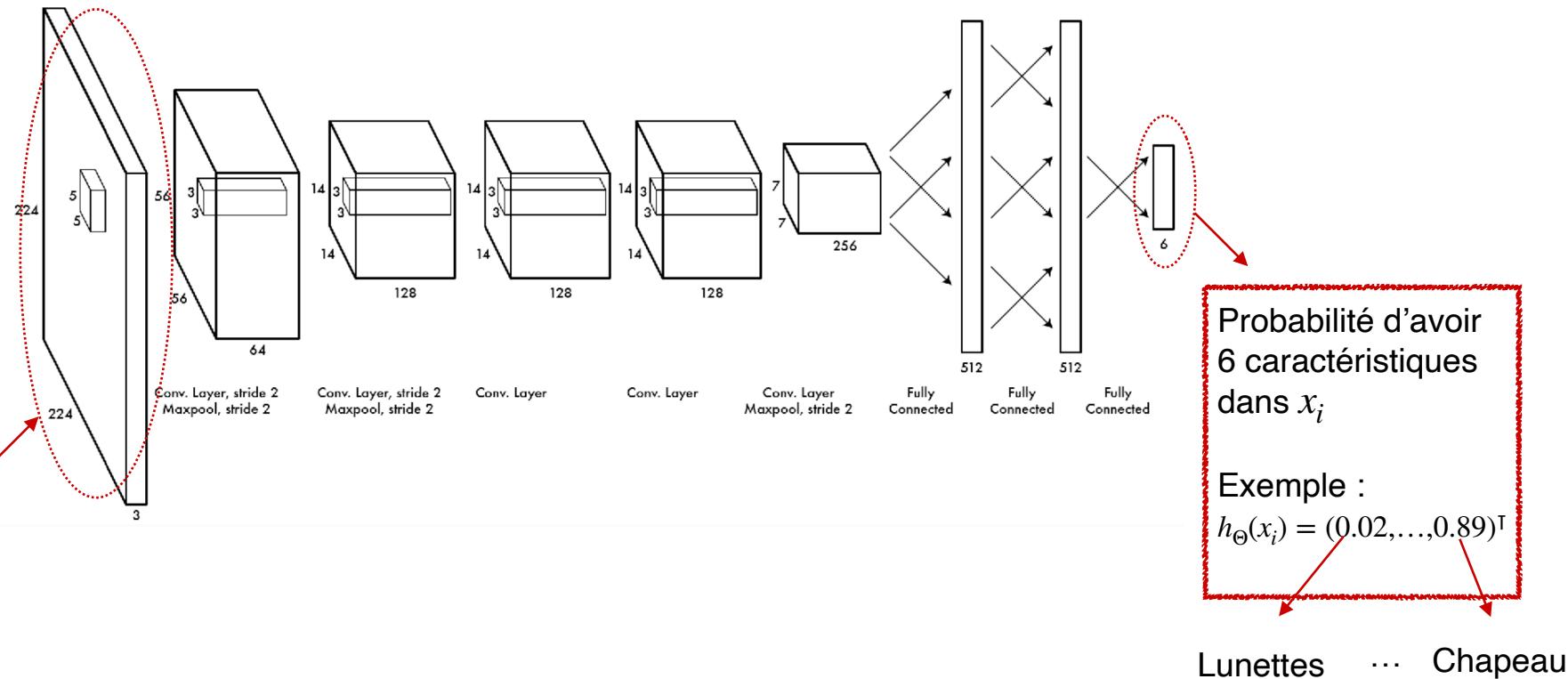
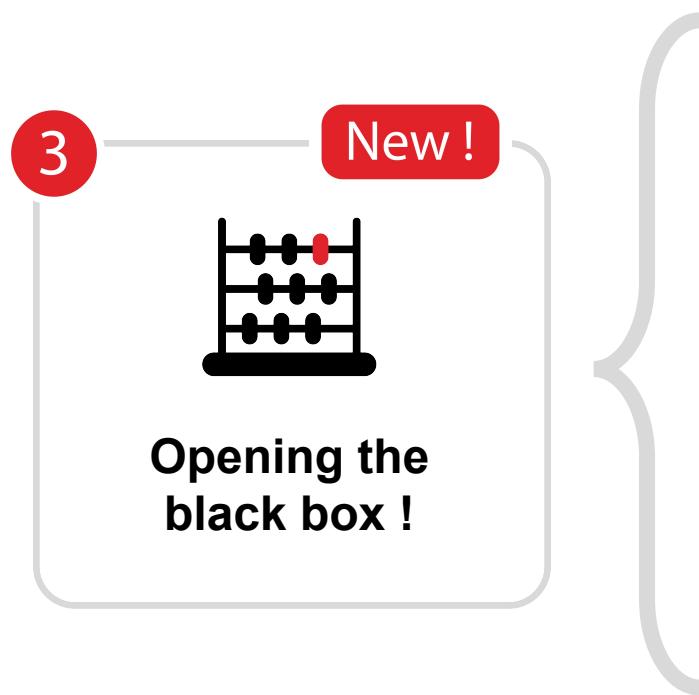


Image du réseau : [Redmon and Angelova, IEEE ICRA 2015]
Figure : Jeu de données CelebA

Paramètres du réseau appris à l'aide de descente de gradient **stochastique** avec des mini-batches de taille 50.

- Comment le réseau transforme couche après couche une image en probabilités ?
- Que-ce qu'une descente de gradient ?
- Comment calculer le gradient des paramètres d'un réseau de neurones (back-propagation) ?
- Stochastique... c'est quoi et quel est l'intérêt ?



1

Transformation de la
représentation des **données**

2

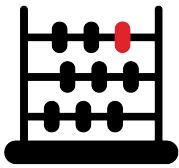
Descente de **gradient**

3

Apprentissage des **paramètres**
d'un réseau de neurones



3



Opening the
black box !

New !

1

Transformation de la
représentation des données

2

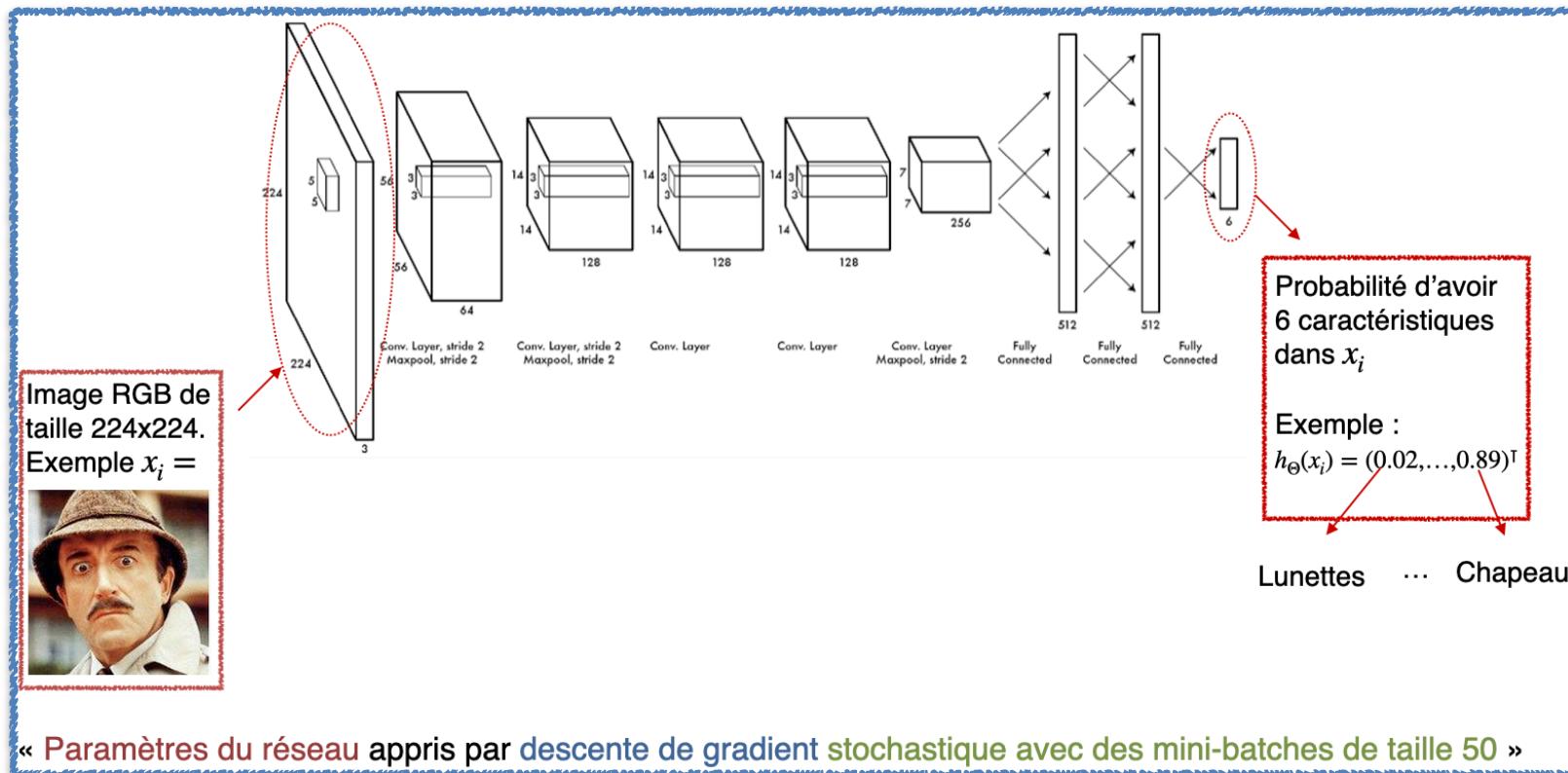
Descente de **gradient**

3

Apprentissage des **paramètres**
d'un réseau de neurones



Partie 1 : Transformation de la représentation des données



- Partie 1 : Comment le réseau transforme couche après couche une image en probabilités ?
- Que ce qu'une descente de gradient ?
- Comment calculer le gradient des paramètres d'un réseau de neurones (back-propagation) ?
- Stochastique... c'est quoi et quel est l'intérêt ?

- 1.1 couches denses
- 1.2 couches de convolutions
- 1.3 assemblage de couches
- 1.4 transformation de l'information

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

Briques de base 1 : Couche dense

Variables

↓
Observations

Eleve	Mathematiques	Physique	Informatique	LV1	LV2	Biologie	Chimie
E01	12.6	10.6	14.2	12.3	10.2	9.3	7.7
E02	13.9	12.9	17.9	12.8	8.8	12.0	10.9
E03	11.3	10.2	10.6	10.4	5.4	9.4	7.0
E04	10.8	12.2	7.7	13.2	4.8	11.1	10.6
E05	11.8	11.3	9.6	11.7	10.4	9.9	7.6
E06	11.9	11.5	13.9	14.1	8.5	9.3	10.3
E07	11.0	11.7	10.1	13.4	13.5	6.3	7.5
E08	10.4	9.2	12.8	6.8	5.1	6.7	7.3
E09	10.1	8.7	11.0	16.2	13.2	9.4	6.3
E10	10.2	7.8	13.9	10.0	3.6	9.4	8.2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Tableau de données

Question 1 : Comment mettre en avant des caractéristiques dans ces données ?

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

Briques de base 1 : Couche dense

Variables 

↓
Observations

Eleve	Mathematiques	Physique	Informatique	LV1	LV2	Biologie	Chimie
E01	12.6	10.6	14.2	12.3	10.2	9.3	7.7
E02	13.9	12.9	17.9	12.8	8.8	12.0	10.9
E03	11.3	10.2	10.6	10.4	5.4	9.4	7.0
E04	10.8	12.2	7.7	13.2	4.8	11.1	10.6
E05	11.8	11.3	9.6	11.7	10.4	9.9	7.6
E06	11.9	11.5	13.9	14.1	8.5	9.3	10.3
E07	11.0	11.7	10.1	13.4	13.5	6.3	7.5
E08	10.4	9.2	12.8	6.8	5.1	6.7	7.3
E09	10.1	8.7	11.0	16.2	13.2	9.4	6.3
E10	10.2	7.8	13.9	10.0	3.6	9.4	8.2
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Tableau de données

$$y_i = \sum_{j=1}^7 \frac{1}{7} x_i^j$$



10.9857

12.7429

9.1857

10.0571

10.3286

11.3571

10.5000

8.3286

10.7000

9.0143



Question 1 : Comment mettre en avant des caractéristiques dans ces données ?

→ Sommes pondérées des notes x_i^j de chaque étudiant i

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

Briques de base 1 : Couche dense

Somme pondérée des notes x_i^j de chaque étudiant i : $y_i = \sum_{j=1}^7 w_j x_i^j$

Moyenne en langues : $w = (0.0, 0.0, 0.0, 0.5, 0.5, 0.0, 0.0)$

Eleve	Mathematiques	Physique	Informatique	LV1	LV2	Biologie	Chimie
E01	12.6	10.6	14.2	12.3	10.2	9.3	7.7
E02	13.9	12.9	17.9	12.8	8.8	12.0	10.9
E03	11.3	10.2	10.6	10.4	5.4	9.4	7.0
E04	10.8	12.2	7.7	13.2	4.8	11.1	10.6
E05	11.8	11.3	9.6	11.7	10.4	9.9	7.6
E06	11.9	11.5	13.9	14.1	8.5	9.3	10.3
E07	11.0	11.7	10.1	13.4	13.5	6.3	7.5
E08	10.4	9.2	12.8	6.8	5.1	6.7	7.3
E09	10.1	8.7	11.0	16.2	13.2	9.4	6.3
E10	10.2	7.8	13.9	10.0	3.6	9.4	8.2

$$\begin{array}{l} \text{11.2500} \\ \text{10.8000} \\ \text{7.9000} \\ \text{9.0000} \\ \text{11.0500} \\ \text{11.3000} \\ \text{13.4500} \\ \text{5.9500} \\ \text{14.7000} \\ \text{6.8000} \end{array} \xrightarrow{y_i}$$

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

Briques de base 1 : Couche dense

Somme pondérée des notes x_i^j de chaque étudiant i : $y_i = \sum_{j=1}^7 w_j x_i^j$

Différence entre *Maths* et *Info* : $w = (1.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0, 0.0)$

Eleve	Mathematiques	Physique	Informatique	LV1	LV2	Biologie	Chimie
E01	12.6	10.6	14.2	12.3	10.2	9.3	7.7
E02	13.9	12.9	17.9	12.8	8.8	12.0	10.9
E03	11.3	10.2	10.6	10.4	5.4	9.4	7.0
E04	10.8	12.2	7.7	13.2	4.8	11.1	10.6
E05	11.8	11.3	9.6	11.7	10.4	9.9	7.6
E06	11.9	11.5	13.9	14.1	8.5	9.3	10.3
E07	11.0	11.7	10.1	13.4	13.5	6.3	7.5
E08	10.4	9.2	12.8	6.8	5.1	6.7	7.3
E09	10.1	8.7	11.0	16.2	13.2	9.4	6.3
E10	10.2	7.8	13.9	10.0	3.6	9.4	8.2

y_i	-1.6000
	-4.0000
	0.7000
	3.1000
	2.2000
	-2.0000
	0.9000
	-2.4000
	-0.9000
	-3.7000

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

Briques de base 1 : Couche dense

Somme pondérée des notes x_i^j de chaque étudiant i : $y_i = \sum_{j=1}^7 w_j x_i^j$

Moyenne dans les matières scientifiques : $w = (0.2, 0.2, 0.2, 0.0, 0.0, 0.2, 0.2)$

Eleve	Mathematiques	Physique	Informatique	LV1	LV2	Biologie	Chimie
E01	12.6	10.6	14.2	12.3	10.2	9.3	7.7
E02	13.9	12.9	17.9	12.8	8.8	12.0	10.9
E03	11.3	10.2	10.6	10.4	5.4	9.4	7.0
E04	10.8	12.2	7.7	13.2	4.8	11.1	10.6
E05	11.8	11.3	9.6	11.7	10.4	9.9	7.6
E06	11.9	11.5	13.9	14.1	8.5	9.3	10.3
E07	11.0	11.7	10.1	13.4	13.5	6.3	7.5
E08	10.4	9.2	12.8	6.8	5.1	6.7	7.3
E09	10.1	8.7	11.0	16.2	13.2	9.4	6.3
E10	10.2	7.8	13.9	10.0	3.6	9.4	8.2

$$\begin{array}{l} \text{10.8800} \\ \text{13.5200} \\ \text{9.7000} \\ \text{10.4800} \\ \text{10.0400} \\ \text{11.3800} \\ \text{9.3200} \\ \text{9.2800} \\ \text{9.1000} \\ \text{9.9000} \end{array} \xrightarrow{y_i}$$

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

Briques de base 1 : Couche dense

Somme pondérée des notes x_i^j de chaque étudiant i : $y_i = \sum_{j=1}^7 w_j x_i^j$

Moyenne dans les matières scientifiques : $w = (0.2, 0.2, 0.2, 0.0, 0.0, 0.2, 0.2)$

Eleve	Mathematiques	Physique	Informatique	LV1	LV2	Biologie	Chimie
E01	12.6	10.6	14.2	12.3	10.2	9.3	7.7
E02	13.9	12.9	17.9	12.8	8.8	12.0	10.9
E03	11.3	10.2	10.6	10.4	5.4	9.4	7.0
E04	10.8	12.2	7.7	13.2	4.8	11.1	10.6
E05	11.8	11.3	9.6	11.7	10.4	9.9	7.6
E06	11.9	11.5	13.9	14.1	8.5	9.3	10.3
E07	11.0	11.7	10.1	13.4	13.5	6.3	7.5
E08	10.4	9.2	12.8	6.8	5.1	6.7	7.3
E09	10.1	8.7	11.0	16.2	13.2	9.4	6.3
E10	10.2	7.8	13.9	10.0	3.6	9.4	8.2

$$\begin{array}{l} \text{10.8800} \\ \text{13.5200} \\ \text{9.7000} \\ \text{10.4800} \\ \text{10.0400} \\ \text{11.3800} \\ \text{9.3200} \\ \text{9.2800} \\ \text{9.1000} \\ \text{9.9000} \end{array} \quad \xrightarrow{\hspace{1cm}} \quad y_i$$

Question 2 : Comment ne garder que de l'information qui nous intéresse ?

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

Briques de base 1 : Couche dense avec biais, suivi de ReLU (Rectified Linear Unit)

Somme pondérée des notes x_i^j avec biais, suivi de ReLU : $s_i = b + \sum_{j=1}^7 w_j x_i^j$

$$y_i = \max(0, s_i)$$

s_i	\rightarrow	y_i	\rightarrow
0.88		0.88	
3.52		3.52	
-0.30		0.00	
0.48		0.48	
0.04		0.04	
1.38		1.38	
-0.68		0.00	
-0.72		0.00	
-0.90		0.00	
-0.10		0.00	

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

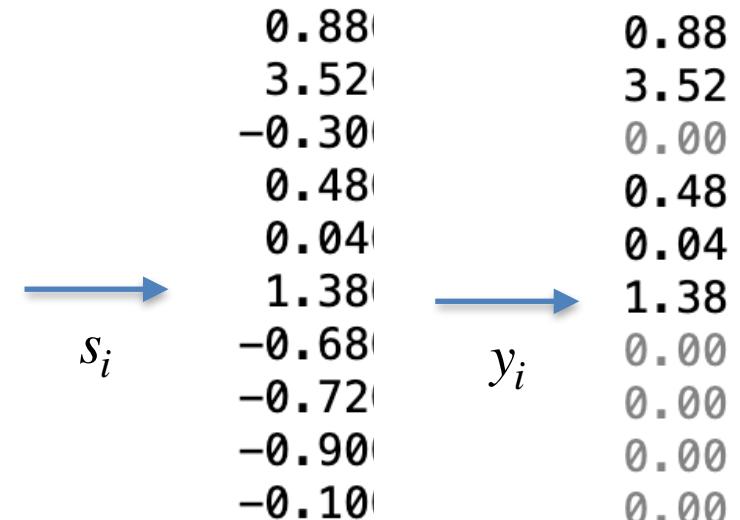
Briques de base 1 : Couche dense avec biais, suivi de ReLU (Rectified Linear Unit)

Somme pondérée des notes x_i^j avec biais, suivi de ReLU : $s_i = b + \sum_{j=1}^7 w_j x_i^j$

$$y_i = \max(0, s_i)$$

Sélection des moyennes supérieures à 10 en sciences : $w = (0.2, 0.2, 0.2, 0.0, 0.0, 0.2, 0.2)$
 $b = -10$, puis ReLU

Eleve	Mathematiques	Physique	Informatique	LV1	LV2	Biologie	Chimie
E01	12.6	10.6	14.2	12.3	10.2	9.3	7.7
E02	13.9	12.9	17.9	12.8	8.8	12.0	10.9
E03	11.3	10.2	10.6	10.4	5.4	9.4	7.0
E04	10.8	12.2	7.7	13.2	4.8	11.1	10.6
E05	11.8	11.3	9.6	11.7	10.4	9.9	7.6
E06	11.9	11.5	13.9	14.1	8.5	9.3	10.3
E07	11.0	11.7	10.1	13.4	13.5	6.3	7.5
E08	10.4	9.2	12.8	6.8	5.1	6.7	7.3
E09	10.1	8.7	11.0	16.2	13.2	9.4	6.3
E10	10.2	7.8	13.9	10.0	3.6	9.4	8.2



ReLU permet de mettre à zéro l'information inutile (en fonction des pondérations choisies)

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

Briques de base 1 : Couche dense avec biais, suivi de ReLU (Rectified Linear Unit)

Plusieurs sommes pondérées avec biais, suivi de ReLU :

- Réduction de la dimension des données
- Extraction de caractéristiques
- Sélection de l'information

$$w = (0.2, 0.2, 0.2, 0, 0, 0.2, 0.2)$$
$$b = -10, \text{ puis ReLU}$$

Eleve	Mathematiques	Physique	Informatique	LV1	LV2	Biologie	Chimie
E01	12.6	10.6	14.2	12.3	10.2	9.3	7.7
E02	13.9	12.9	17.9	12.8	8.8	12.0	10.9
E03	11.3	10.2	10.6	10.4	5.4	9.4	7.0
E04	10.8	12.2	7.7	13.2	4.8	11.1	10.6
E05	11.8	11.3	9.6	11.7	10.4	9.9	7.6
E06	11.9	11.5	13.9	14.1	8.5	9.3	10.3
E07	11.0	11.7	10.1	13.4	13.5	6.3	7.5
E08	10.4	9.2	12.8	6.8	5.1	6.7	7.3
E09	10.1	8.7	11.0	16.2	13.2	9.4	6.3
E10	10.2	7.8	13.9	10.0	3.6	9.4	8.2

$$w = (0, 0, 0, 0.5, 0.5, 0, 0)$$
$$b = -10, \text{ puis ReLU}$$

0.88 , 1.25
3.52 , 0.80
0.00 , 0.00
0.48 , 0.00
0.04 , 1.05
1.38 , 1.30
0.00 , 3.45
0.00 , 0.00
0.00 , 4.70
0.00 , 0.00

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

Briques de base 1 : Couche dense avec biais, suivi de ReLU (Rectified Linear Unit)

Plusieurs sommes pondérées avec biais, suivi de ReLU :

- Réduction de la dimension des données
- Extraction de caractéristiques
- Sélection de l'information

$$w = (0.2, 0.2, 0.2, 0, 0, 0.2, 0.2) \\ b = -10, \text{ puis ReLU}$$

Eleve	Mathematiques	Physique	Informatique	LV1	LV2	Biologie	Chimie
E01	12.6	10.6	14.2	12.3	10.2	9.3	7.7
E02	13.9	12.9	17.9	12.8	8.8	12.0	10.9
E03	11.3	10.2	10.6	10.4	5.4	9.4	7.0
E04	10.8	12.2	7.7	13.2	4.8	11.1	10.6
E05	11.8	11.3	9.6	11.7	10.4	9.9	7.6
E06	11.9	11.5	13.9	14.1	8.5	9.3	10.3
E07	11.0	11.7	10.1	13.4	13.5	6.3	7.5
E08	10.4	9.2	12.8	6.8	5.1	6.7	7.3
E09	10.1	8.7	11.0	16.2	13.2	9.4	6.3
E10	10.2	7.8	13.9	10.0	3.6	9.4	8.2

$$w = (0, 0, 0, 0.5, 0.5, 0, 0) \\ b = -10, \text{ puis ReLU}$$

0.88 , 1.25
3.52 , 0.80
0.00 , 0.00
0.48 , 0.00
0.04 , 1.05
1.38 , 1.30
0.00 , 3.45
0.00 , 0.00
0.00 , 4.70
0.00 , 0.00

Question 3 : Comment coder cela dans un langage dédié aux réseaux de neurones ?

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

Briques de base 1 : Couche dense avec biais, suivi de ReLU (Rectified Linear Unit)

```
import numpy as np
import matplotlib.pyplot as plt
import pandas

import torch
```

```
MyData=pandas.read_csv('notes.csv')
MyData.pop('Eleve')
```

```
Matieres=list(MyData.keys())
print(Matieres)
```

```
Notes=MyData.to_numpy()
print(Notes)
```

```
['Mathematiques', 'Physique', 'Informatique', 'LV1', 'LV2', 'Biologie', 'Chimie']
[[12.6 10.6 14.2 12.3 10.2 9.3 7.7]
 [13.9 12.9 17.9 12.8 8.8 12. 10.9]
 [11.3 10.2 10.6 10.4 5.4 9.4 7. ]
 [10.8 12.2 7.7 13.2 4.8 11.1 10.6]
 [11.8 11.3 9.6 11.7 10.4 9.9 7.6]
 [11.9 11.5 13.9 14.1 8.5 9.3 10.3]
 [11. 11.7 10.1 13.4 13.5 6.3 7.5]

 :
 [12.6 10.7 15.2 11.5 12.1 9.9 7.8]]
```

```
Notes_pt=torch.tensor(Notes,dtype=torch.float64)
```

Partie 1 : Transformation de la représentation des données → 1.1 : Couches denses

Briques de base 1 : Couche dense avec biais, suivi de ReLU (Rectified Linear Unit)

```
#definition de la fonction d'activation ReLU
act_func_relu=torch.nn.ReLU()

#definition de la couche dense et de ses paramètres
lin_filter=torch.nn.Linear(7, 2, bias=True)

lin_filter.weight.data = torch.tensor([[0.2,0.2,0.2,0,0,0.2,0.2],
                                       [0.0,0.0,0.0,0.5,0.5,0.,0.]],
                                       dtype=torch.float64)

lin_filter.bias.data = torch.tensor([-10,-10],dtype=torch.float64)

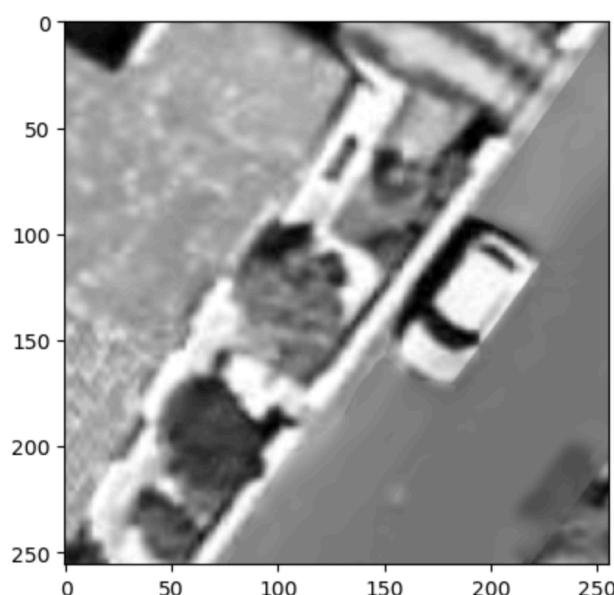
#traitement des données
filtered_data=act_func_relu(lin_filter(Notes_pt[:,::]))

#résultat
print(filtered_data)

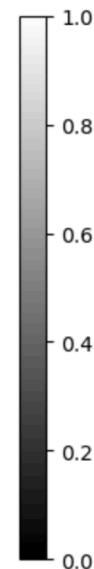
tensor([[0.8800, 1.2500],
       [3.5200, 0.8000],
       [0.0000, 0.0000],
       [0.4800, 0.0000],
       [0.0400, 1.0500],
       [1.3800, 1.3000],
       [0.0000, 3.4500],
       :
       [1.2400, 1.8000]], dtype=torch.float64, grad_fn=<ReluBackward0>)
```

Partie 1 : Transformation de la représentation des données → 1.2 : Filtres de convolutions

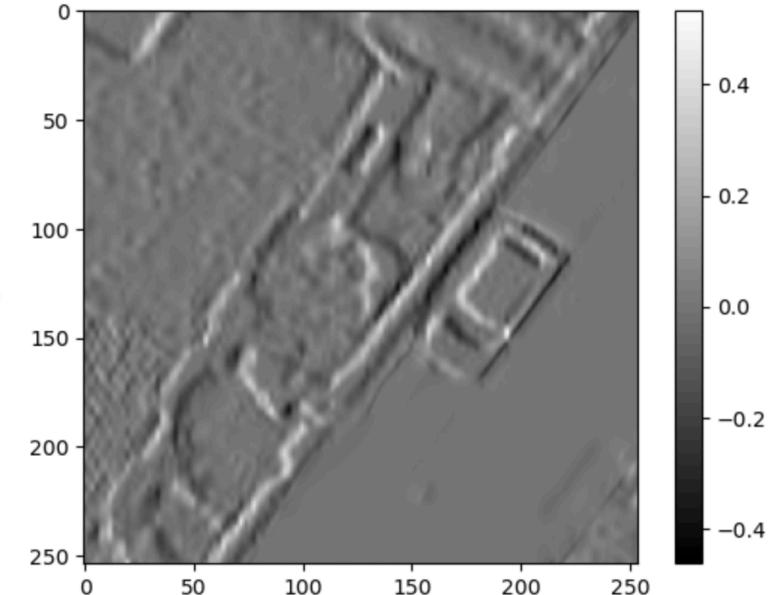
Briques de base 2 : Convolution



Im_l



$$\otimes \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} /3$$



Im_{l+1}

$$\forall (i, j) \in \Omega : Im_{l+1}[i, j] = \sum_{(m, n) \in \omega} Im_l[i + m, j + n] K[m, n]$$

Convolution



Pixels de l'image



Image filtrée



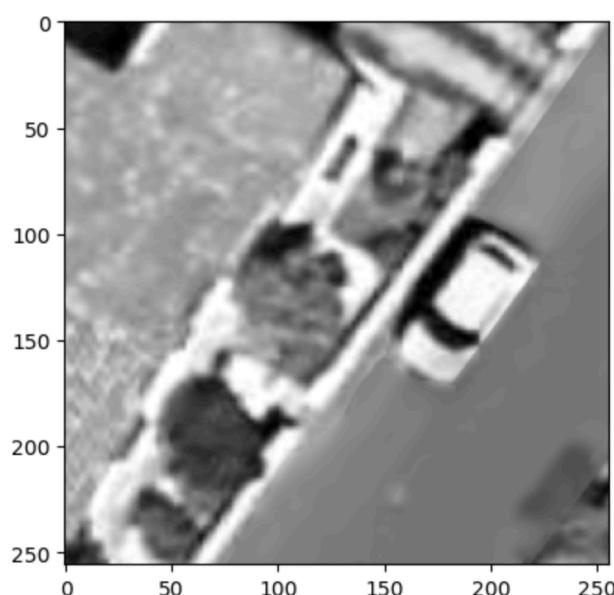
Image initiale



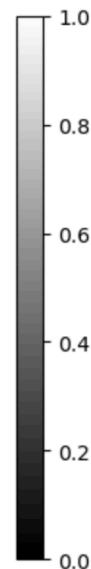
Filtre de convolution

Partie 1 : Transformation de la représentation des données → 1.2 : Filtres de convolutions

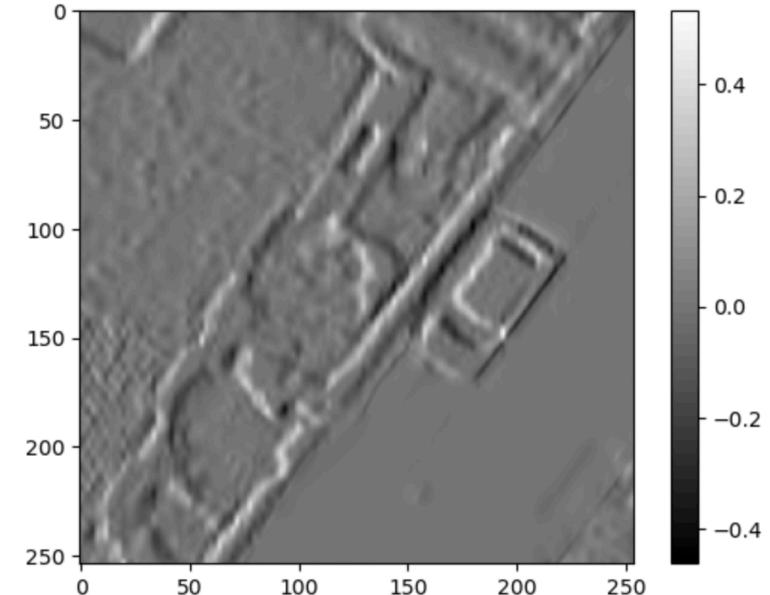
Briques de base 2 : Convolution



Im_l



$$\otimes \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} /3$$



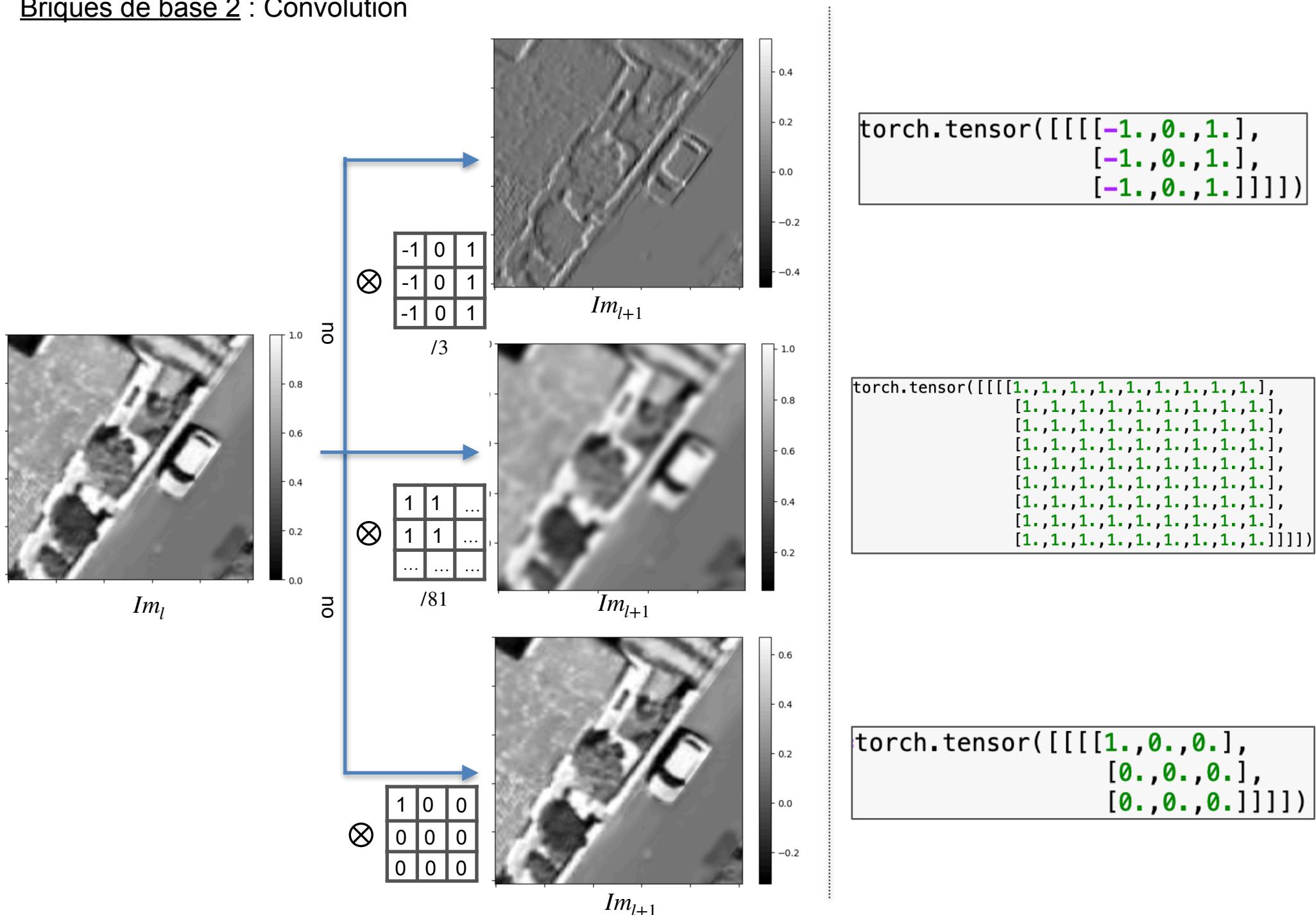
Im_{l+1}

```
conv_filter=torch.nn.Conv2d(1,1,3)
conv_filter.weight.data=torch.tensor([[[[-1.,0.,1.],
                                         [-1.,0.,1.],
                                         [-1.,0.,1.]]]])/3.
```

```
filtered_image=conv_filter(Image_GL)
```

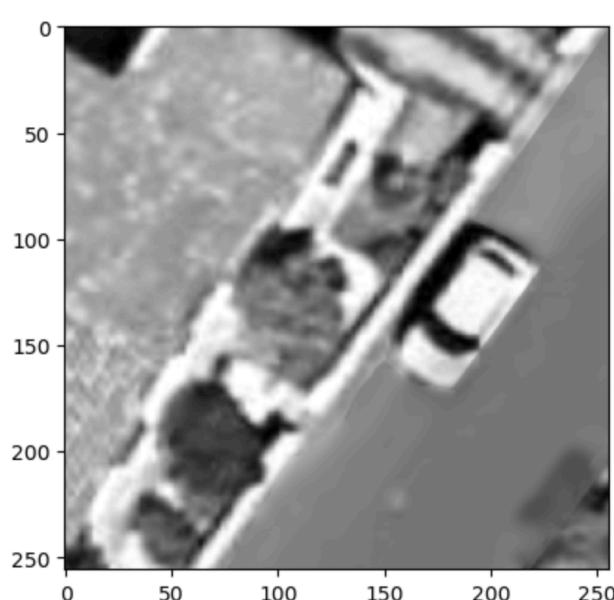
Partie 1 : Transformation de la représentation des données → 1.2 : Filtres de convolutions

Briques de base 2 : Convolution



Partie 1 : Transformation de la représentation des données → 1.2 : Filtres de convolutions

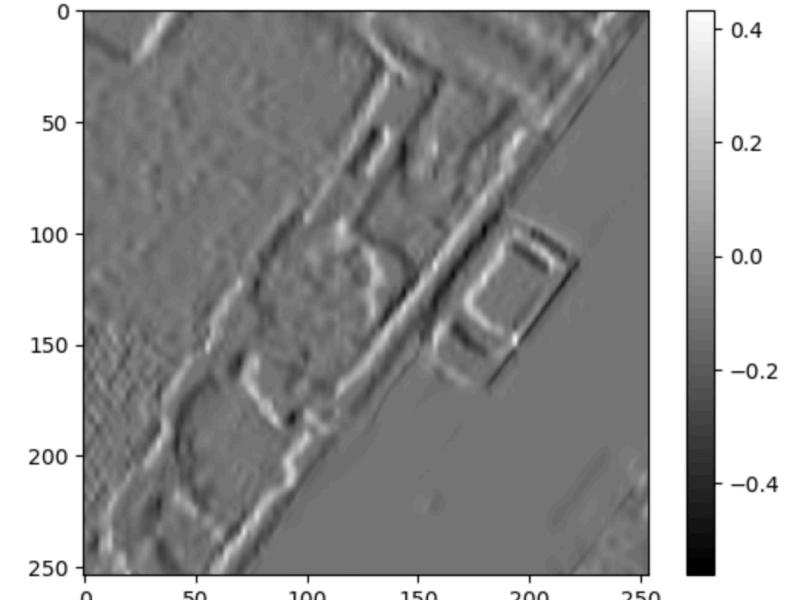
Briques de base 2 : Convolution et biais



Im_l



$$\otimes \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} /3 - 0.1$$



Im_{l+1}

$$\forall (i, j) \in \Omega : Im_{l+1}[i, j] = b + \sum_{(m, n) \in \omega} Im_l[i + m, j + n]K[m, n]$$

Pixels de l'image

Image filtrée

Biais

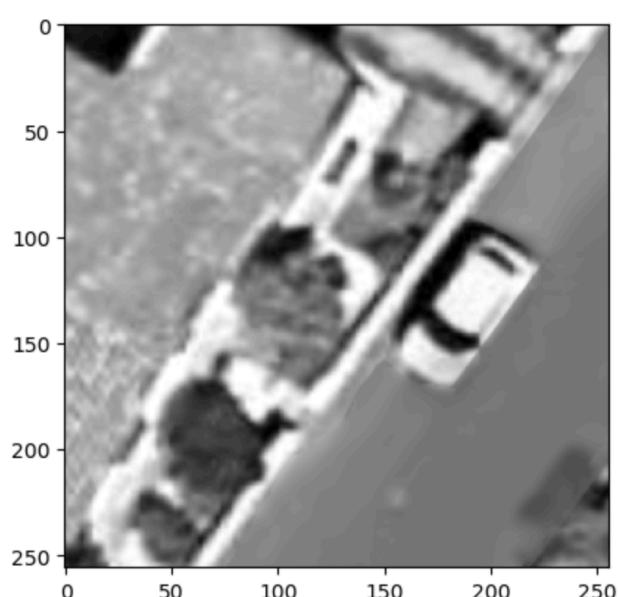
Image initiale

Filtre de convolution

Convolution + biais

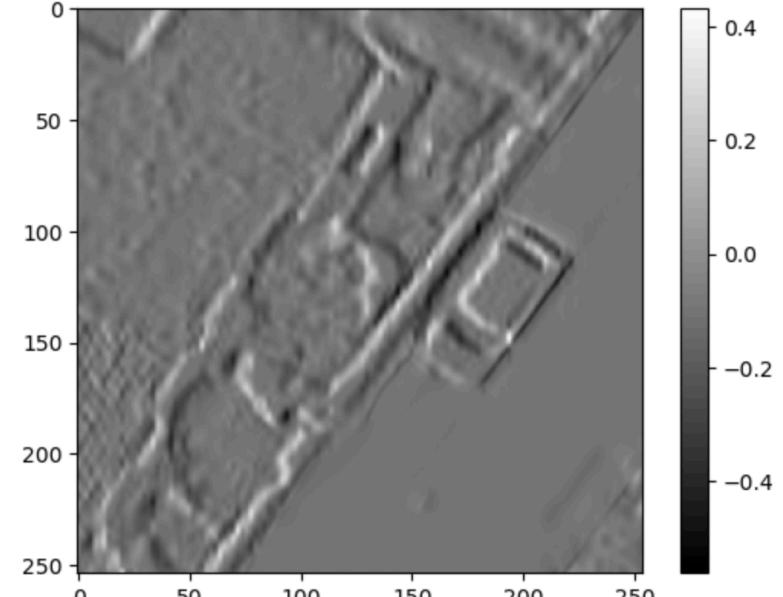
Partie 1 : Transformation de la représentation des données → 1.2 : Filtres de convolutions

Briques de base 2 : Convolution et biais



Im_l

$$\otimes \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} /3 - 0.1$$



Im_{l+1}

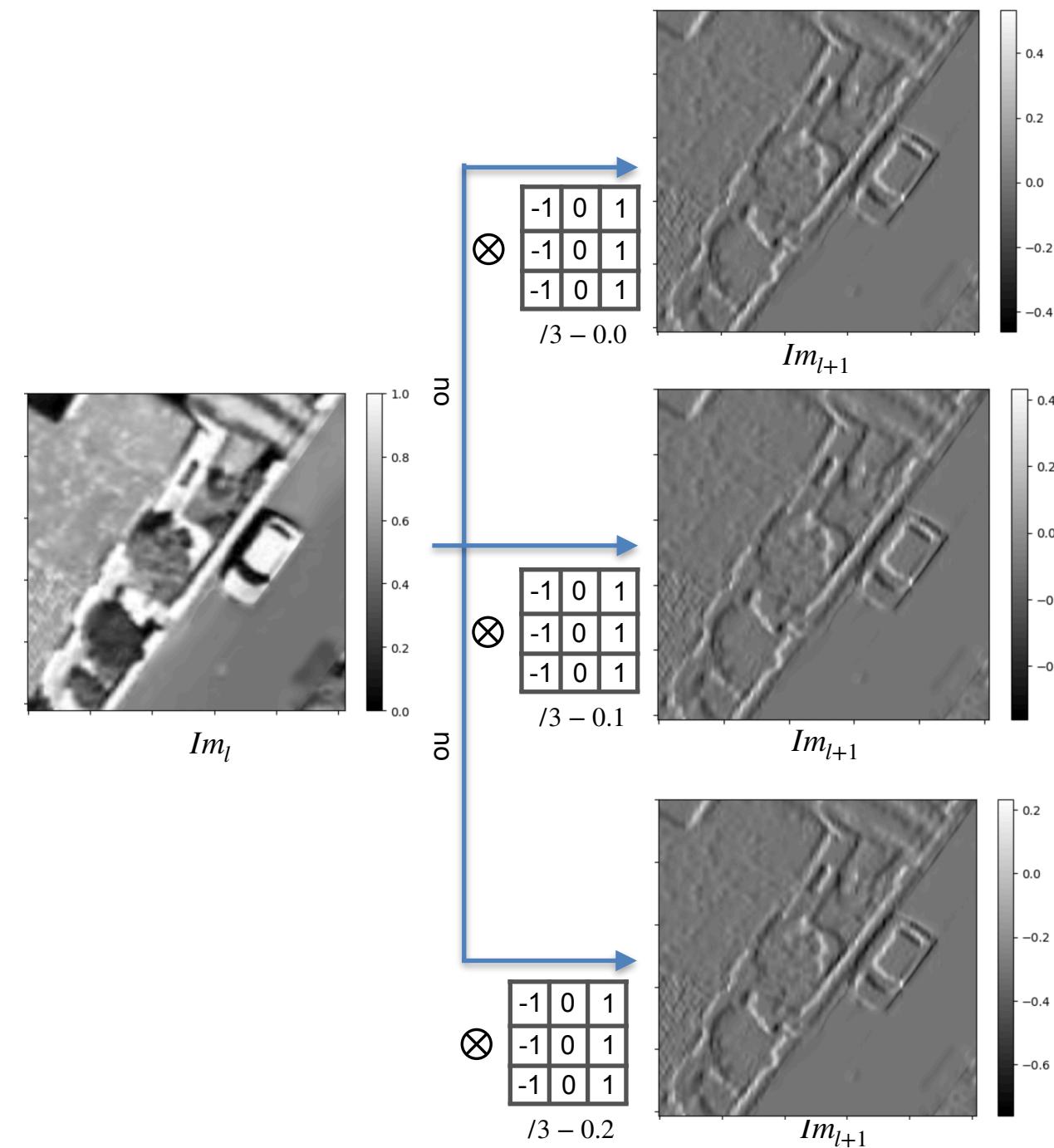
```
conv_filter=torch.nn.Conv2d(1,1,3,bias=True)
conv_filter.weight.data=torch.tensor([[[[-1.,0.,1.],
                                         [-1.,0.,1.],
                                         [-1.,0.,1.]]]])/3.

conv_filter.bias.data=torch.tensor([-0.1])

filtered_image=conv_filter(Image_GL)
```

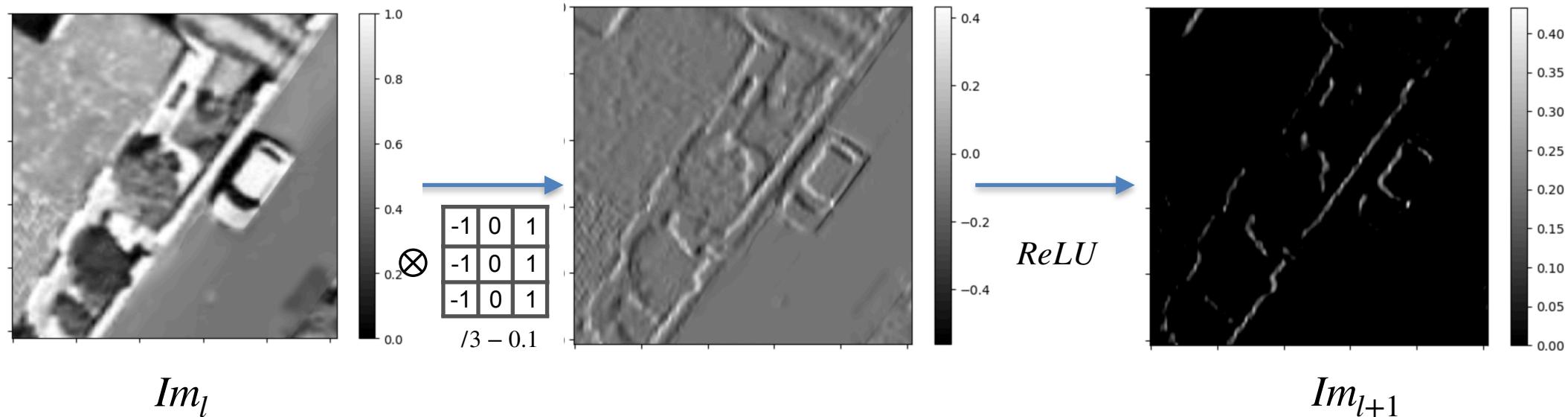
Partie 1 : Transformation de la représentation des données → 1.2 : Filtres de convolutions

Briques de base 2 : Convolution et biais



Partie 1 : Transformation de la représentation des données → 1.2 : Filtres de convolutions

Briques de base 2 : Convolution et biais, suivi de ReLu (Rectified Linear Unit)



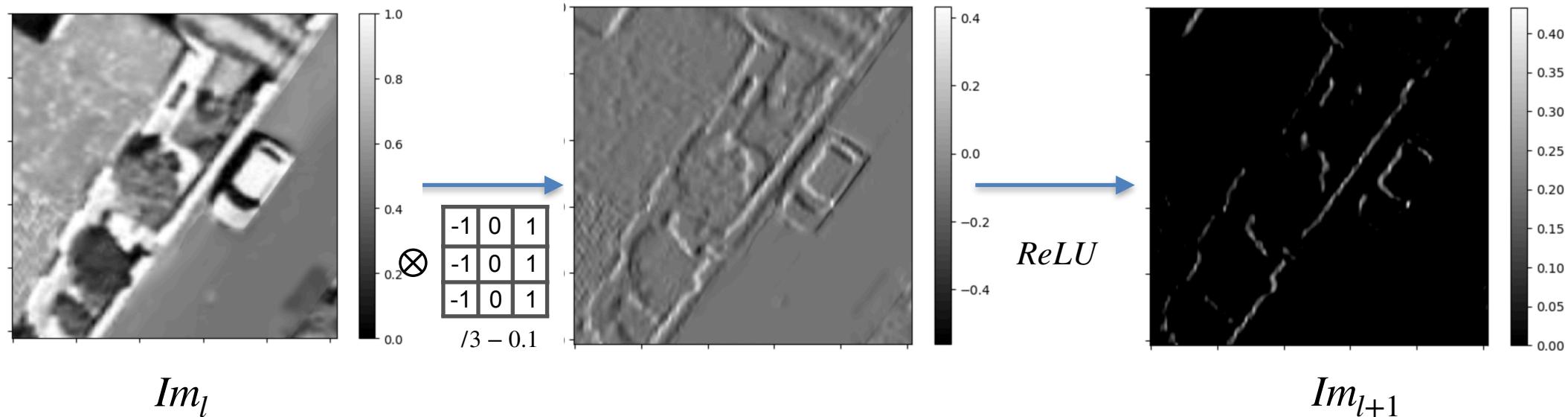
$$\forall (i, j) \in \Omega : Im_{l+1}[i, j] = \max \left(0, b + \sum_{(m,n) \in \omega} Im_l[i + m, j + n] K[m, n] \right)$$

Valeur avant ReLU

Convolution + biais, suivi de ReLU

Partie 1 : Transformation de la représentation des données → 1.2 : Filtres de convolutions

Briques de base 2 : Convolution et biais, suivi de ReLu (Rectified Linear Unit)



```
act_func_relu=torch.nn.ReLU()

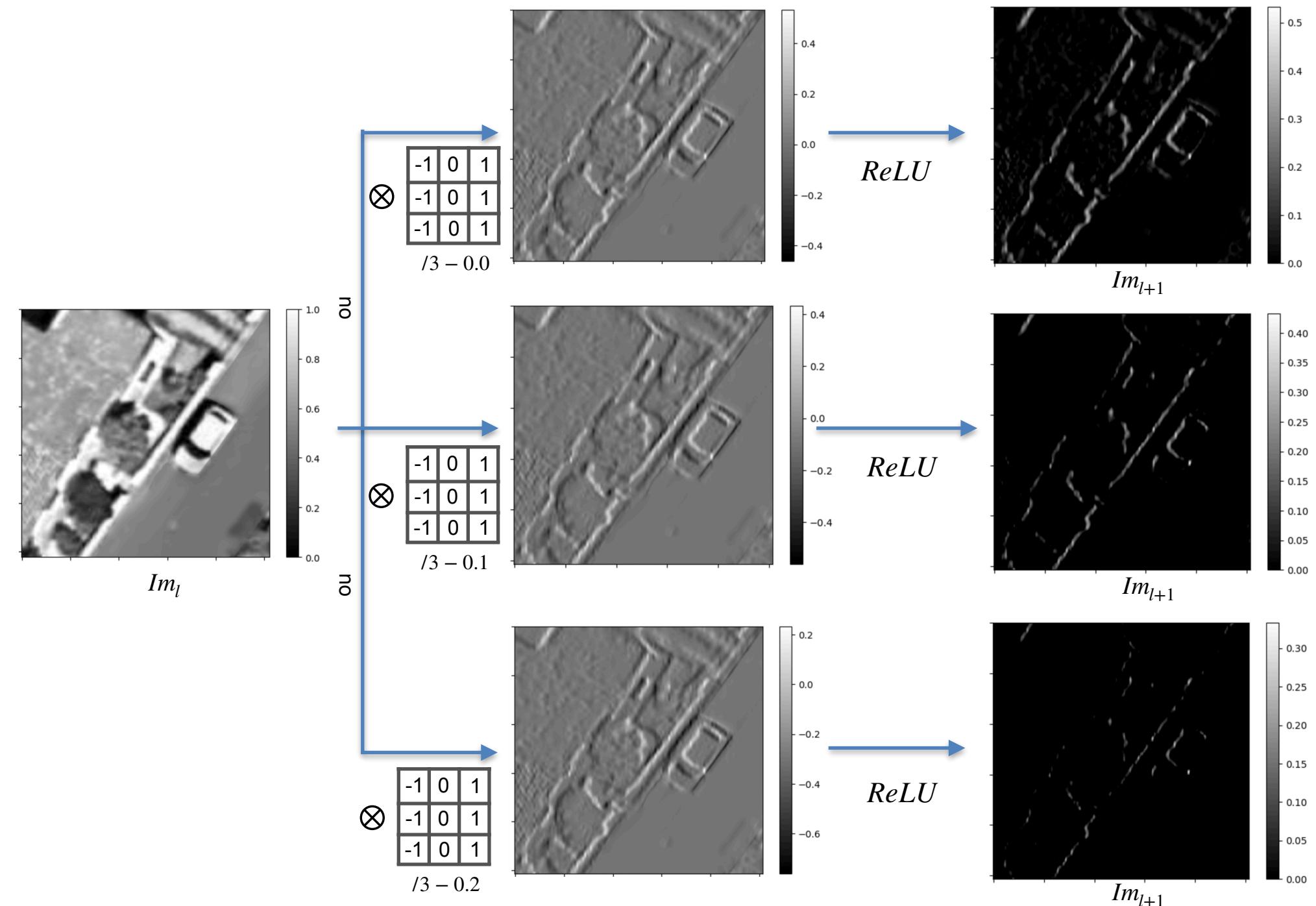
conv_filter=torch.nn.Conv2d(1,1,3,bias=True)
conv_filter.weight.data=torch.tensor([[[[-1.,0.,1.],
                                         [-1.,0.,1.],
                                         [-1.,0.,1.]]]])/3.

conv_filter.bias.data=torch.tensor([-0.1])

filtered_image=act_func_relu(conv_filter(Image_GL))
```

Partie 1 : Transformation de la représentation des données → 1.2 : Filtres de convolutions

Briques de base 2 : Convolution et biais, suivi de ReLu (Rectified Linear Unit)



Partie 1 : Transformation de la représentation des données → 1.3 : Assemblage de couches

Assemblage de briques de base : Architecture de réseau de neurones

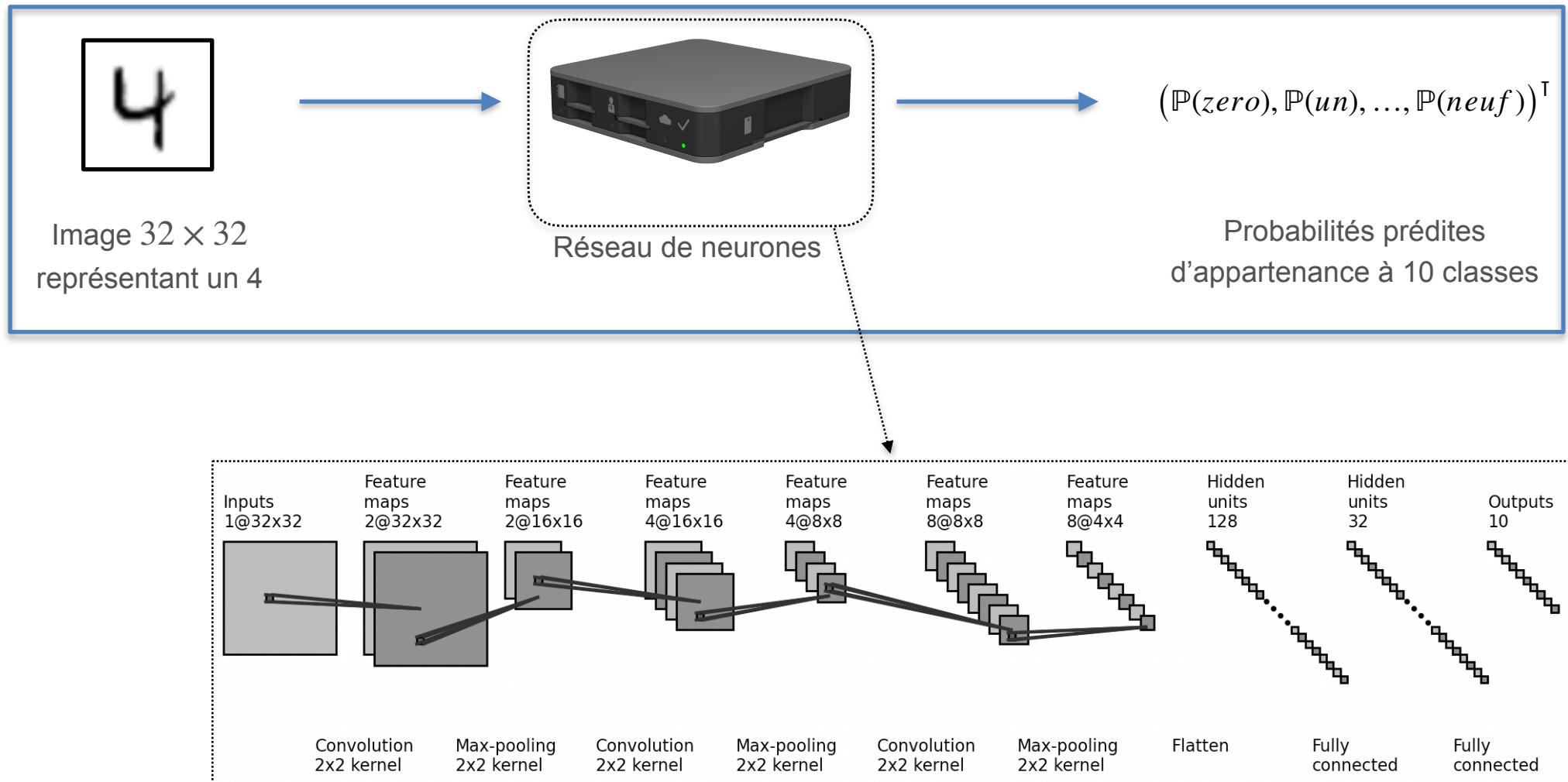
Exemple : Données MNIST [LeCun et al. 1998] pour la détection de chiffres manuscrits



Partie 1 : Transformation de la représentation des données → 1.3 : Assemblage de couches

Assemblage de briques de base : Architecture de réseau de neurones

Exemple : Données MNIST [LeCun et al. 1998] pour la détection de chiffres manuscrits



Partie 1 : Transformation de la représentation des données → 1.3 : Assemblage de couches

```
class basicCNN(nn.Module):

    def __init__(self):
        super(basicCNN, self).__init__()

        #Convolution/ReLU/MaxPooling layers definition
        self.conv1 = nn.Conv2d(1, 2, kernel_size=2, stride=1, padding=1) #1 to 2 channels
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2) #32x32 to 16x16 pixels
        self.conv2 = nn.Conv2d(2, 4, kernel_size=2, stride=1, padding=1) #2 to 4 channels
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2) #16x16 to 8x8 pixels
        self.conv3 = nn.Conv2d(4, 8, kernel_size=2, stride=1, padding=1) #4 to 8 channels
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2) #8x8 to 4x4 pixels

        #Dense layers definition
        self.fc1 = nn.Linear(8 * 4 * 4, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = F.relu(self.conv3(x))
        x = self.pool3(x)
        x = x.view(-1, 8*4*4) #flatten the data
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

    return(x)

basicCNN_model = basicCNN()
```

Partie 1 : Transformation de la représentation des données → 1.3 : Assemblage de couches

```
class basicCNN(nn.Module):

    def __init__(self):
        super(basicCNN, self).__init__()

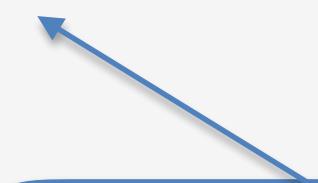
        #Convolution/ReLU/MaxPooling layers definition
        self.conv1 = nn.Conv2d(1, 2, kernel_size=2, stride=1, padding=1) #1 to 2 channels
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2) #32x32 to 16x16 pixels
        self.conv2 = nn.Conv2d(2, 4, kernel_size=2, stride=1, padding=1) #2 to 4 channels
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2) #16x16 to 8x8 pixels
        self.conv3 = nn.Conv2d(4, 8, kernel_size=2, stride=1, padding=1) #4 to 8 channels
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2) #8x8 to 4x4 pixels

        #Dense layers definition
        self.fc1 = nn.Linear(8 * 4 * 4, 32)
        self.fc2 = nn.Linear(32, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)
        x = F.relu(self.conv2(x))
        x = self.pool2(x)
        x = F.relu(self.conv3(x))
        x = self.pool3(x)
        x = x.view(-1, 8*4*4) #flatten the data
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

    return(x)

basicCNN_model = basicCNN()
```



Apprentissage des filtres de convolutions, des biais et des poids à l'aide de données annotée (partie 3 et séquence sur les CNN)

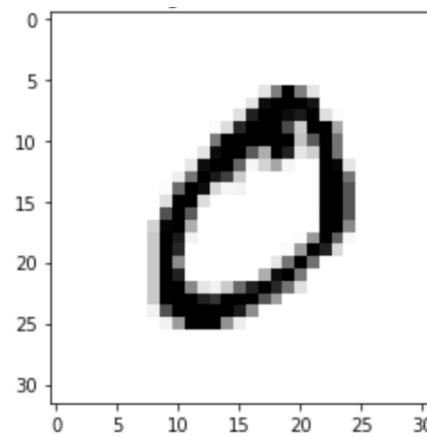
Partie 1 : Transformation de la représentation des données → 1.3 : Assemblage de couches

Une fois les paramètres du réseau appris, on peut tester :

```
pred = basicCNN_model(X)  
print(pred)
```

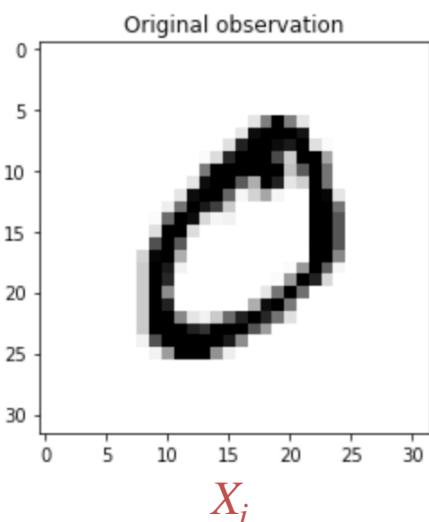
```
tensor([[0.8959, 0.0009, 0.0023, 0.0039, 0.0014, 0.0013, 0.0054, 0.0014, 0.0385,  
0.0488]])
```

avec :

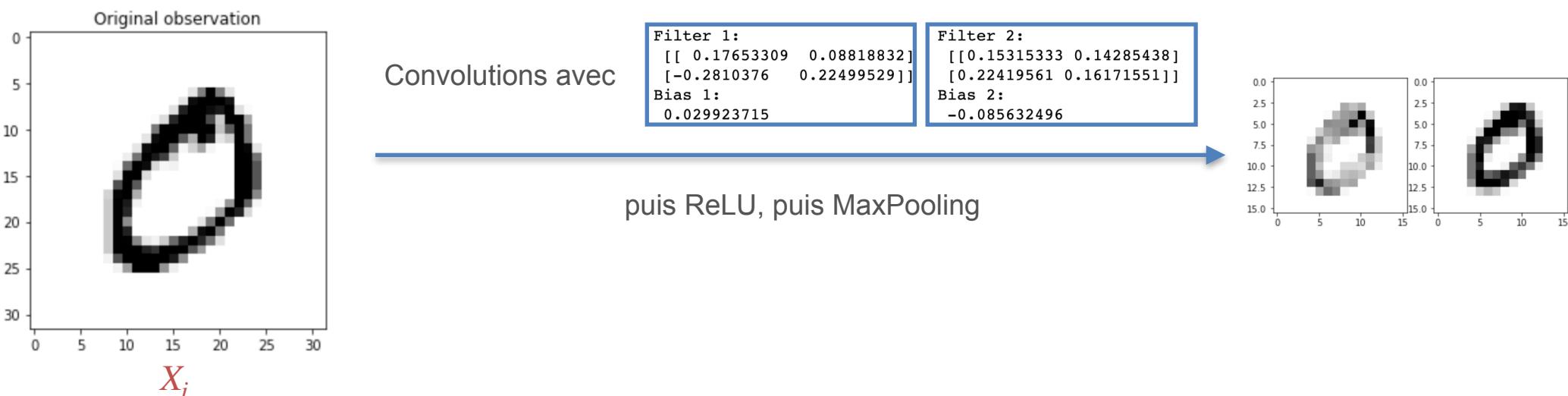


Comment le réseau de neurone modifie l'image couche après couche ?

Partie 1 : Transformation de la représentation des données → 1.4 : Transformation de l'information



Partie 1 : Transformation de la représentation des données → 1.4 : Transformation de l'information



Convolution et ReLU:

Couche Canal

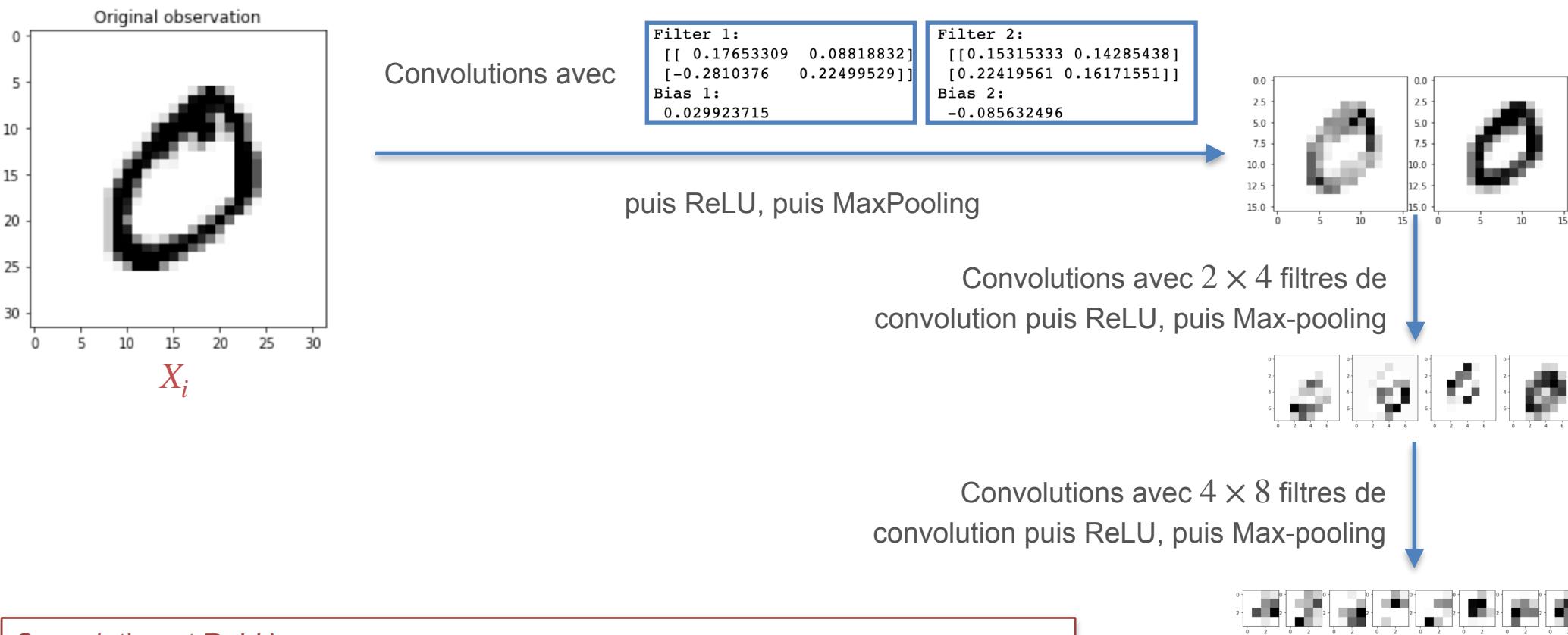
Biais du canal c de la couche 2

Filtre du canal c de la couche 2

$$I_i^{2,c}(p_x, p_y) = b_c^2 + \sum_{k=0}^1 \sum_{l=0}^1 X_i(p_x + k, p_y + l) w_c^2(k, l)$$

$$I_i^{2,c}(p_x, p_y) = \max(0, I_i^{2,c}(p_x, p_y))$$

Partie 1 : Transformation de la représentation des données → 1.4 : Transformation de l'information



Convolution et ReLU:

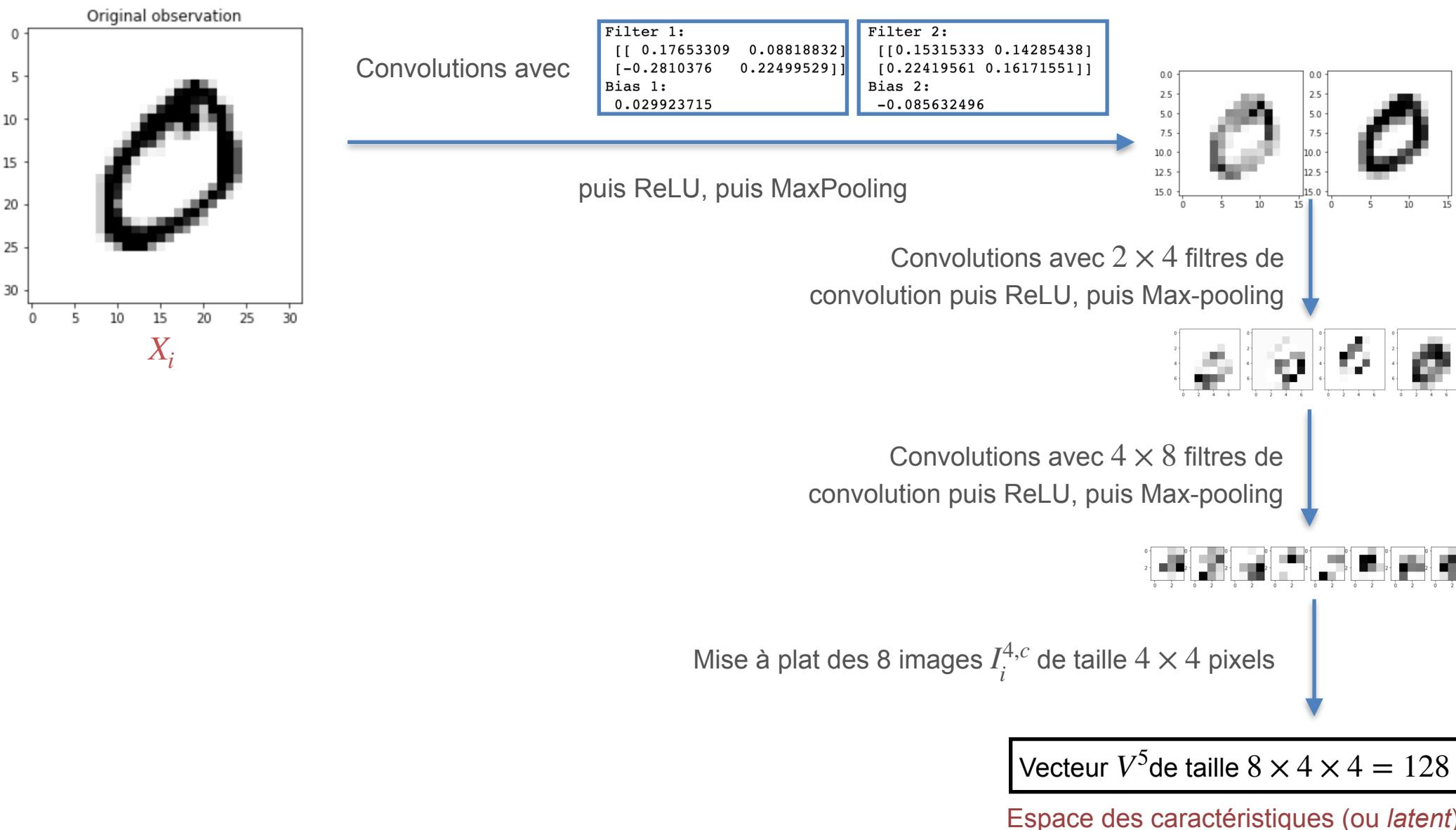
Somme sur les canaux de la couche d'entrée

Filtre du canal c' de la couche 2 au canal c de la couche 3

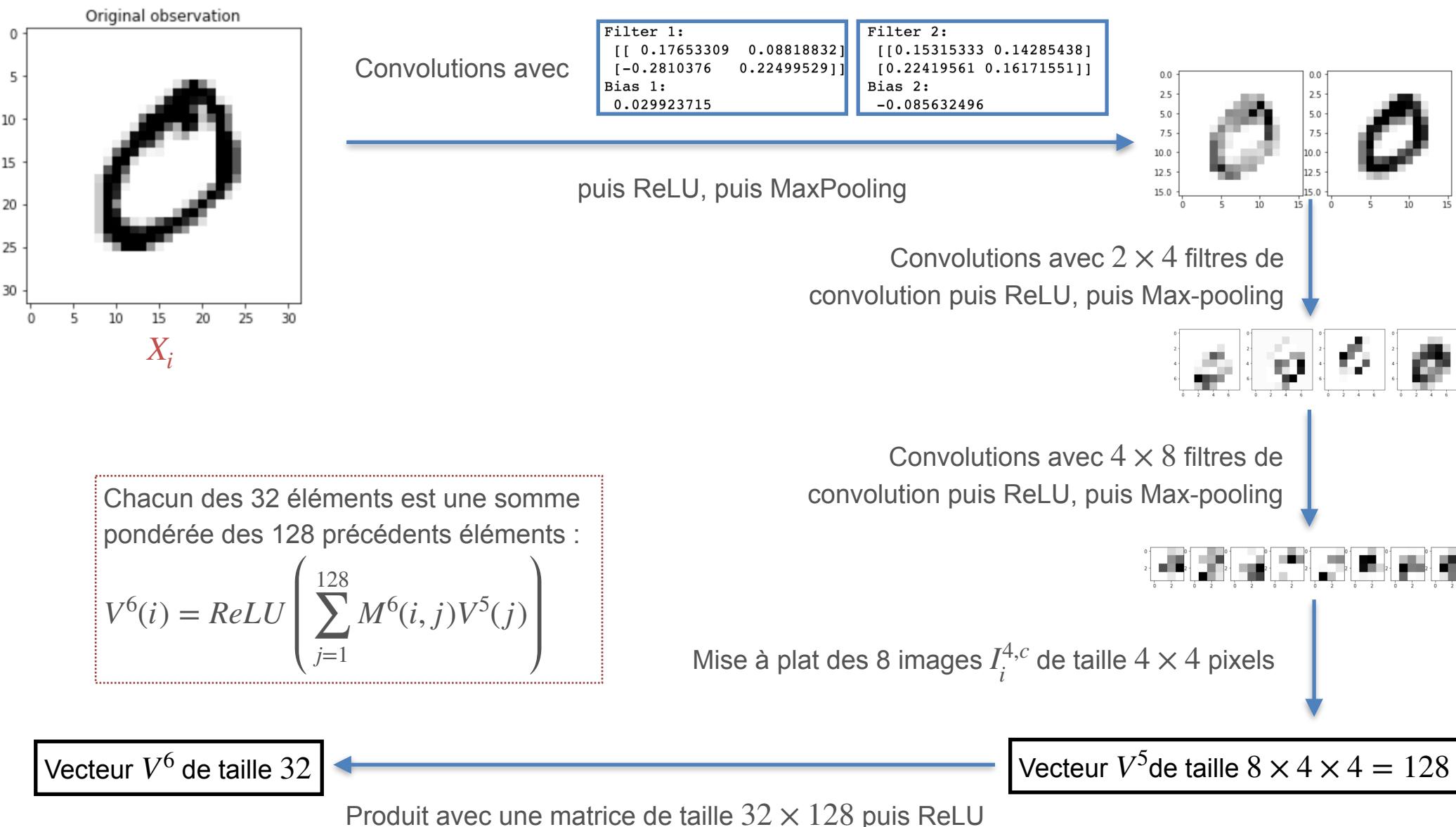
$$I_i^{3,c}(p_x, p_y) = b_c^3 + \sum_{c'=1}^2 \left(\sum_{k=0}^1 \sum_{l=0}^1 I_i^{2,c'}(p_x + k, p_y + l) w_{c,c'}^3(k, l) \right)$$

$$I_i^{3,c}(p_x, p_y) = \max(0, I_i^{3,c}(p_x, p_y))$$

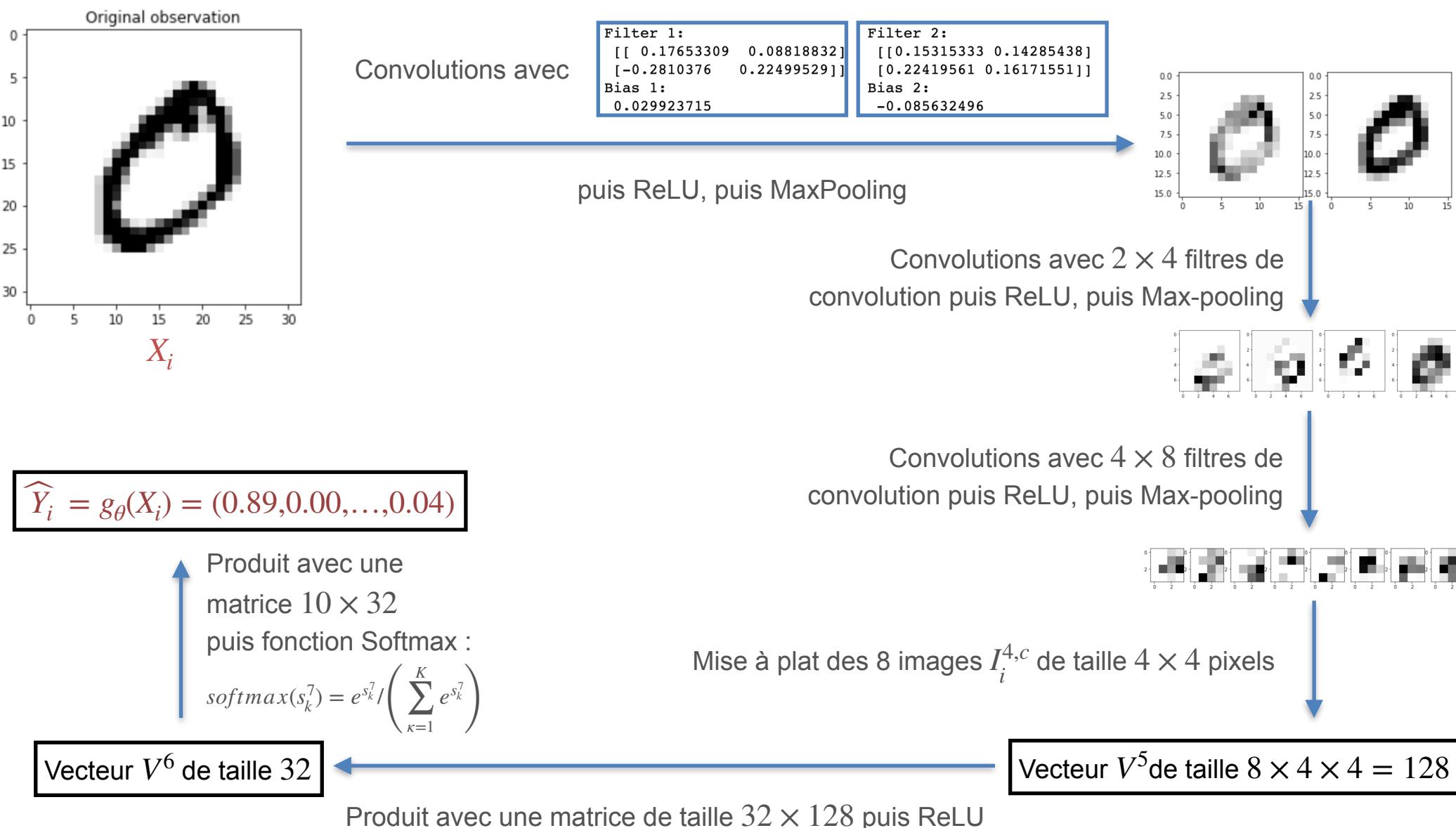
Partie 1 : Transformation de la représentation des données → 1.4 : Transformation de l'information



Partie 1 : Transformation de la représentation des données → 1.4 : Transformation de l'information

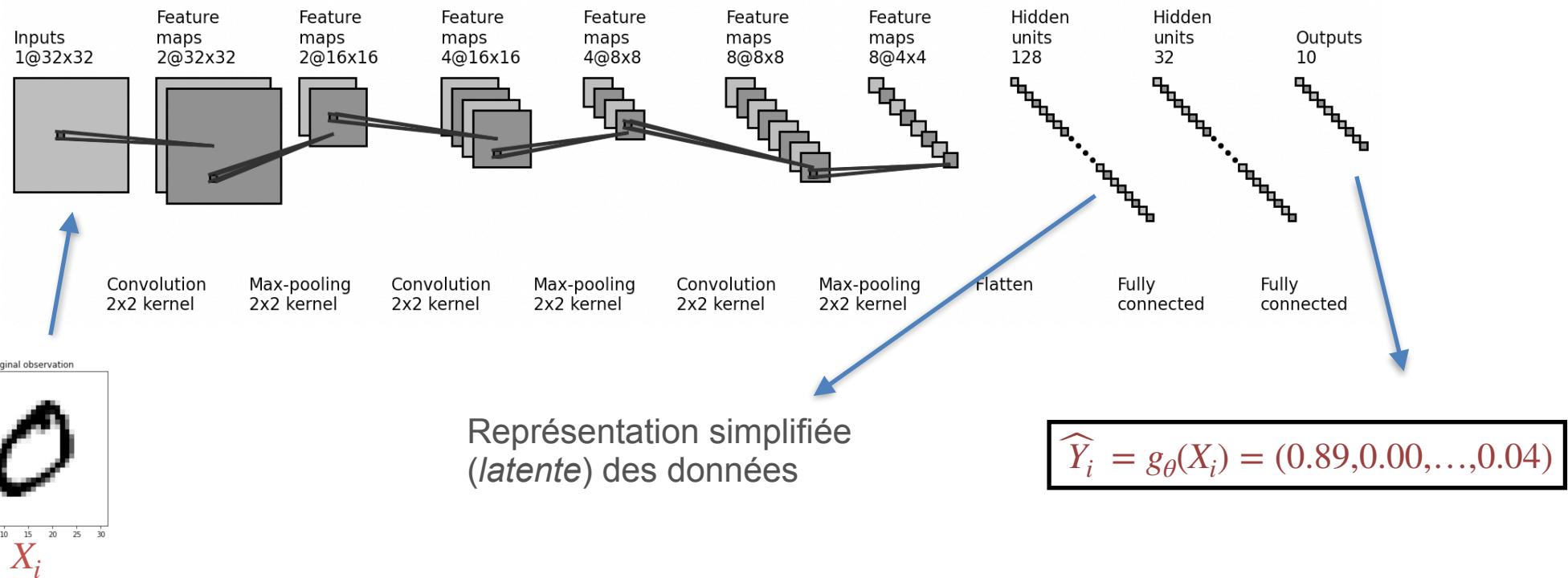


Partie 1 : Transformation de la représentation des données → 1.4 : Transformation de l'information



→ Environ 96% de bonnes prédictions sur 10000 images ici (peut être de $\approx 99\%$ avec d'autres CNNs).

Partie 1 : Transformation de la représentation des données → 1.4 : Transformation de l'information

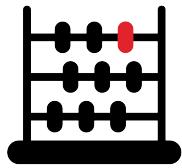


Apprentissage du CNN : Optimisation des **[filtres de convolution]**, **[biais]** et **[matrices des dernières couches]** afin qu'ils conduisent aux meilleures prédictions possibles sur un jeu d'apprentissage.

- Les filtres ressortent des propriétés pertinentes dans l'image
- Les différents canaux mélangés peuvent créer des représentations avancées de l'information
- A l'aide des biais, les fonctions ReLU permettent de supprimer l'information inutile
- Le max-pooling sélectionne dans chaque région de l'image l'information la plus pertinente.



3



Opening the
black box !

New !

1

Transformation de la
représentation des données

2

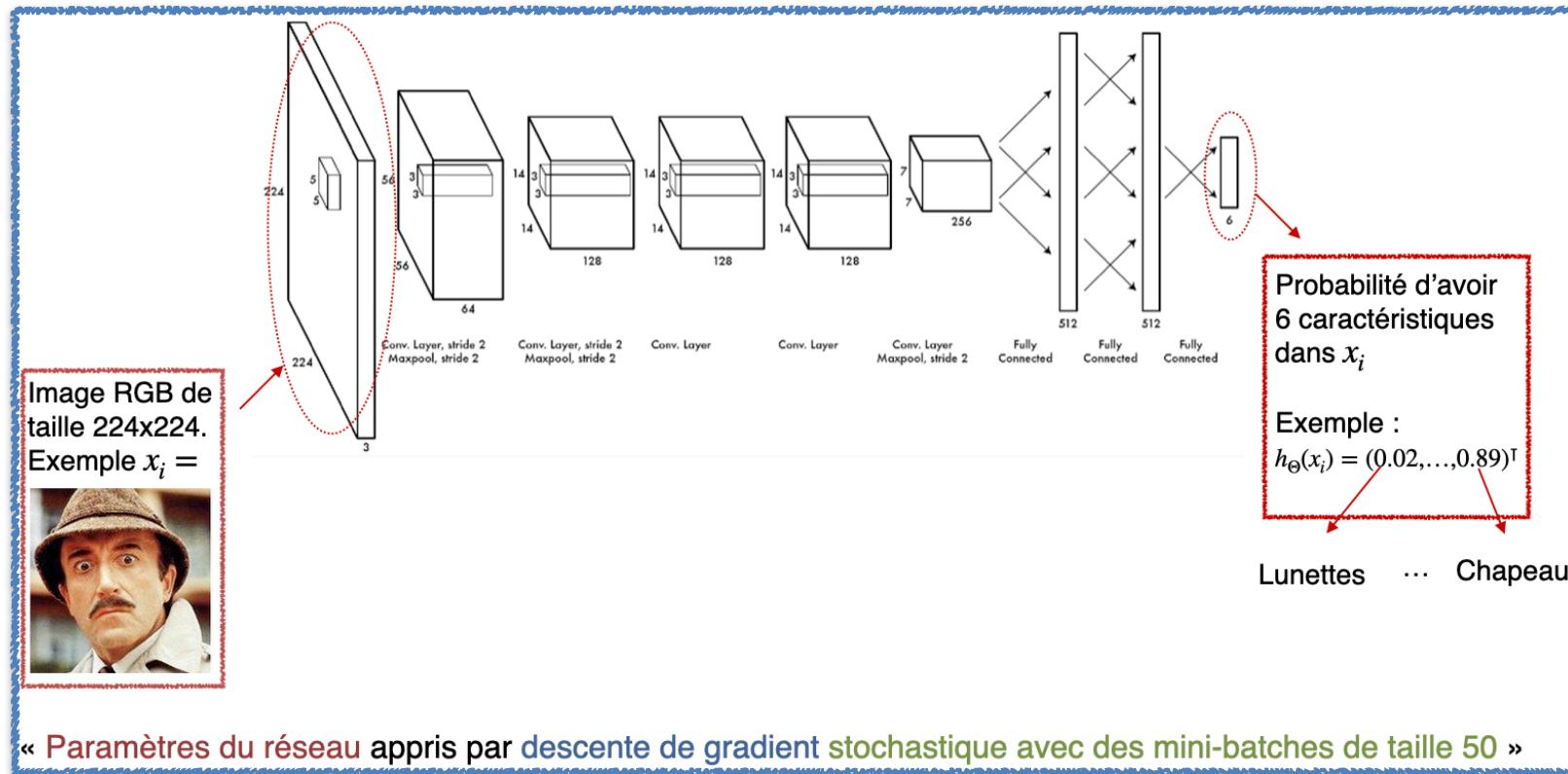
Descente de **gradient**

3

Apprentissage des paramètres
d'un réseau de neurones



Partie 2 : Descente de gradient



- Comment le réseau transforme couche après couche une image en probabilités ?
- Partie 2 : Que-ce qu'une descente de gradient ?
- Comment calculer le gradient des paramètres d'un réseau de neurones (back-propagation) ?
- Stochastique... c'est quoi et quel est l'intérêt ?

Partie 2 : Descente de gradient — Exemple introductif

On connaît les notes de n élèves au cours de l'année scolaire 2022-2023 ainsi que leur notes à un concours de fin d'année.

On aimerait prédire les notes au concours des élèves de la promotion 2023-2024 en fonction de leurs notes au cours de l'année.

	Maths	Info	Français	Concours
Elève 1	12	15	09	14
Elève 2	05	09	12	07
...
Elève i	x_i^1	x_i^2	x_i^3	y_i
...
Elève n	10	12	15	11

Nouvel élève

	Maths	Info	Français	Concours
Nouvel élève	13	14	11	?

Partie 2 : Descente de gradient — Exemple introductif

On connaît les notes de n élèves au cours de l'année scolaire 2022-2023 ainsi que leur notes à un concours de fin d'année.

On aimerait prédire les notes au concours des élèves de la promotion 2023-2024 en fonction de leurs notes au cours de l'année.

	Maths	Info	Français	Concours
Elève 1	12	15	09	14
Elève 2	05	09	12	07
...
Elève i	x_i^1	x_i^2	x_i^3	y_i
...
Elève n	10	12	15	11

Nouvel élève

	Maths	Info	Français	Concours
Nouvel élève	13	14	11	?

Posons les notations :

- Notes de l'élève $i \in \{1, \dots, n\}$ durant l'année 2022-2023 : $x_i = (x_i^1, x_i^2, \dots, x_i^p) \in \mathbb{R}^p$
- Notes de l'élève i au concours 2022-2023 : $y_i \in \mathbb{R}$
- Prédiction de y_{new} pour l'élève new en fonction des notes x_{new} : $\widehat{y_{new}} = h_\Theta(x_{new})$

Prédiction de y_{new}

Fonction $\mathbb{R}^p \rightarrow \mathbb{R}$
Paramètres Θ appris avec les $(x_i, y_i)_{i=1, \dots, n}$

Partie 2 : Descente de gradient - Quel modèle choisir pour h_θ ?

- Utilisons un modèle très simple : La **régression linéaire** !

$$\widehat{y}_i = h_\Theta(x_i) = w_0 + \sum_{j=1}^p w_j x_i^j \quad \text{dont les paramètres sont } \Theta = \{w_0, w_1, \dots, w_p\}$$

- Prédiction de note au concours pour un élève ayant les notes x_{new} au cours de l'année ?

Maths	Info	Français	Concours
13	14	11	?

Prenons $w_0 = 0.$, $w_1 = 0.33$, $w_2 = 0.33$ et $w_3 = 0.33$ \rightarrow Alors $\widehat{y}_{new} = 12.54$

Partie 2 : Descente de gradient - Critère pour apprendre les paramètres θ ?

Les meilleurs paramètres $\hat{\Theta}$ minimisent un **risque empirique** sur les $(x_i, y_i)_{i=1, \dots, n}$, par exemple :

$$\begin{aligned}\hat{\Theta} &= \arg \min_{\Theta=\{w_0, \dots, w_p\}} \frac{1}{n} \sum_{i=1}^n \text{loss}(h_{\Theta}(x_i), y_i) \\ &= \arg \min_{\Theta=\{w_0, \dots, w_p\}} \frac{1}{n} \sum_{i=1}^n (h_{\Theta}(x_i) - y_i)^2 \\ &= \arg \min_{\Theta=\{w_0, \dots, w_p\}} \frac{1}{n} \sum_{i=1}^n \left(w_0 + \sum_{j=1}^p w_j x_i^j - y_i \right)^2\end{aligned}$$

Risque empirique $R_{\Theta}((x_i, y_i)_{i=1, \dots, n})$

Partie 2 : Descente de gradient - Critère pour apprendre les paramètres θ ?

Les meilleurs paramètres $\hat{\Theta}$ minimisent un **risque empirique** sur les $(x_i, y_i)_{i=1, \dots, n}$, par exemple :

$$\Theta : \quad w_0 = 0. \quad w_1 = 0.33 \quad w_2 = 0.33 \quad w_3 = 0.33$$

	Maths	Info	Français	Concours	Prediction
Eleve 1	12	15	09	14	11.8
Eleve 2	05	09	12	07	8.6
Eleve 3	13	12	13	12	12.6
...
Eleve n	10	12	15	11	12.2

$$h_{\Theta}(x_i) = w_0 + \sum_{j=1}^p w_j x_i^j$$

$$\frac{1}{n} \sum_{i=1}^n (h_{\Theta}(x_i) - y_i)^2$$

$$\text{Risque empirique} = 2.24$$

Partie 2 : Descente de gradient - Critère pour apprendre les paramètres θ ?

Les meilleurs paramètres $\hat{\Theta}$ minimisent un **risque empirique** sur les $(x_i, y_i)_{i=1, \dots, n}$, par exemple :

$$\Theta : \quad w_0 = 0. \quad w_1 = 0.40 \quad w_2 = 0.40 \quad w_3 = 0.20$$

	Maths	Info	Français	Concours	Prediction
Eleve 1	12	15	09	14	12.6
Eleve 2	05	09	12	07	8.0
Eleve 3	13	12	13	12	12.6
...
Eleve n	10	12	15	11	11.8

$$h_{\Theta}(x_i) = w_0 + \sum_{j=1}^p w_j x_i^j$$

$$\frac{1}{n} \sum_{i=1}^n (h_{\Theta}(x_i) - y_i)^2$$

$$\text{Risque empirique} = 0.99$$

Partie 2 : Descente de gradient - Critère pour apprendre les paramètres θ ?

Les meilleurs paramètres $\hat{\Theta}$ minimisent un **risque empirique** sur les $(x_i, y_i)_{i=1, \dots, n}$, par exemple :

$$\Theta : w_0 = 0. \quad w_1 = 0.50 \quad w_2 = 0.50 \quad w_3 = 0.00$$

	Maths	Info	Français
Eleve 1	12	15	09
Eleve 2	05	09	12
Eleve 3	13	12	13
...
Eleve n	10	12	15

	Concours
Eleve 1	14
Eleve 2	07
Eleve 3	12
...	...
Eleve n	11

	Prediction
Eleve 1	13.5
Eleve 2	7.0
Eleve 3	12.5
...	...
Eleve n	11.0

$$h_{\Theta}(x_i) = w_0 + \sum_{j=1}^p w_j x_i^j$$

$$\frac{1}{n} \sum_{i=1}^n (h_{\Theta}(x_i) - y_i)^2$$

$$\text{Risque empirique} = 0.13$$

BIEN... mais ce serait mieux de manière automatique !!!

Partie 2 : Descente de gradient - Minimisation du risque en fonction de θ

- Nous allons utiliser le **gradient du risque empirique** par rapport aux paramètres du modèle :

$$R_{\Theta}((x_i, y_i)_{i=1, \dots, n}) = \frac{1}{n} \sum_{i=1}^n (h_{\Theta}(x_i) - y_i)^2 = \frac{1}{n} \sum_{i=1}^n (w_0 + \sum_{j=1}^p w_j x_i^j - y_i)^2$$

$$\nabla R_{\Theta}(\dots) = \left(\frac{\partial R_{\Theta}(\dots)}{\partial w_0}, \frac{\partial R_{\Theta}(\dots)}{\partial w_1}, \dots, \frac{\partial R_{\Theta}(\dots)}{\partial w_p} \right)^T = \frac{1}{n} \sum_{i=1}^n 2 (h_{\Theta}(x_i) - y_i) (1, x_i^1, \dots, x_i^p)^T$$

↑
Gradient de R_{Θ}

Partie 2 : Descente de gradient - Minimisation du risque en fonction de θ

- Nous allons utiliser le **gradient du risque empirique** par rapport aux paramètres du modèle :

$$R_{\Theta}((x_i, y_i)_{i=1, \dots, n}) = \frac{1}{n} \sum_{i=1}^n (h_{\Theta}(x_i) - y_i)^2 = \frac{1}{n} \sum_{i=1}^n (w_0 + \sum_{j=1}^p w_j x_i^j - y_i)^2$$

$$\nabla R_{\Theta}(\dots) = \left(\frac{\partial R_{\Theta}(\dots)}{\partial w_0}, \frac{\partial R_{\Theta}(\dots)}{\partial w_1}, \dots, \frac{\partial R_{\Theta}(\dots)}{\partial w_p} \right)^T = \frac{1}{n} \sum_{i=1}^n 2 (h_{\Theta}(x_i) - y_i) (1, x_i^1, \dots, x_i^p)^T$$

Gradient de R_{Θ}

- Descent de gradient :

$$\Theta_{it1} = (0.0, 0.33, 0.33, 0.33)^T$$



$$R_{\Theta_{it1}}(\dots) = 2.24$$

$$\nabla R_{\Theta_{it1}}(\dots) = (0.81, 2.93, 7.23, 12.03)^T$$

$$\Theta_{it2} = \Theta_{it1} - \lambda \nabla \Theta_{it1}(\dots) = (-0.00, 0.33, 0.32, 0.32)^T$$



Partie 2 : Descente de gradient - Minimisation du risque en fonction de θ

- Nous allons utiliser le **gradient du risque empirique** par rapport aux paramètres du modèle :

$$R_{\Theta}((x_i, y_i)_{i=1, \dots, n}) = \frac{1}{n} \sum_{i=1}^n (h_{\Theta}(x_i) - y_i)^2 = \frac{1}{n} \sum_{i=1}^n (w_0 + \sum_{j=1}^p w_j x_i^j - y_i)^2$$

$$\nabla R_{\Theta}(\dots) = \left(\frac{\partial R_{\Theta}(\dots)}{\partial w_0}, \frac{\partial R_{\Theta}(\dots)}{\partial w_1}, \dots, \frac{\partial R_{\Theta}(\dots)}{\partial w_p} \right)^T = \frac{1}{n} \sum_{i=1}^n 2 (h_{\Theta}(x_i) - y_i) (1, x_i^1, \dots, x_i^p)^T$$

↑
Gradient de R_{Θ}

- Descent de gradient :

$$\Theta_{it1} = (0.0, 0.33, 0.33, 0.33)^T$$



$$R_{\Theta_{it1}}(\dots) = 2.24$$

$$\nabla R_{\Theta_{it1}}(\dots) = (0.81, 2.93, 7.23, 12.03)^T$$

$$\Theta_{it2} = \Theta_{it1} - \lambda \nabla \Theta_{it1}(\dots) = (-0.00, 0.33, 0.32, 0.32)^T$$



$$R_{\Theta_{it2}}(\dots) = 2.09$$

$$\nabla R_{\Theta_{it2}}(\dots) = (0.28, -2.47, 1.23, 5.10)^T$$

$$\Theta_{it3} = \Theta_{it2} - \lambda \nabla \Theta_{it2}(\dots) = (-0.00, 0.33, 0.32, 0.31)^T$$



Partie 2 : Descente de gradient - Minimisation du risque en fonction de θ

- Nous allons utiliser le **gradient du risque empirique** par rapport aux paramètres du modèle :

$$R_{\Theta}((x_i, y_i)_{i=1, \dots, n}) = \frac{1}{n} \sum_{i=1}^n (h_{\Theta}(x_i) - y_i)^2 = \frac{1}{n} \sum_{i=1}^n (w_0 + \sum_{j=1}^p w_j x_i^j - y_i)^2$$

$$\nabla R_{\Theta}(\dots) = \left(\frac{\partial R_{\Theta}(\dots)}{\partial w_0}, \frac{\partial R_{\Theta}(\dots)}{\partial w_1}, \dots, \frac{\partial R_{\Theta}(\dots)}{\partial w_p} \right)^T = \frac{1}{n} \sum_{i=1}^n 2 (h_{\Theta}(x_i) - y_i) (1, x_i^1, \dots, x_i^p)^T$$

↑
Gradient de R_{Θ}

- Descent de gradient :

$$\Theta_{it1} = (0.0, 0.33, 0.33, 0.33)^T$$



$$R_{\Theta_{it1}}(\dots) = 2.24$$

$$\nabla R_{\Theta_{it1}}(\dots) = (0.81, 2.93, 7.23, 12.03)^T$$

$$\Theta_{it2} = \Theta_{it1} - \lambda \nabla \Theta_{it1}(\dots) = (-0.00, 0.33, 0.32, 0.32)^T$$



$$R_{\Theta_{it2}}(\dots) = 2.09$$

$$\nabla R_{\Theta_{it2}}(\dots) = (0.28, -2.47, 1.23, 5.10)^T$$

$$\Theta_{it3} = \Theta_{it2} - \lambda \nabla \Theta_{it2}(\dots) = (-0.00, 0.33, 0.32, 0.31)^T$$



$$R_{\Theta_{it3}}(\dots) = 2.04$$

$$\nabla R_{\Theta_{it3}}(\dots) = (0.17, -3.48, 0.05, 3.71)^T$$

⋮

$$\Theta_{it1000} = \Theta_{it999} - \lambda \nabla \Theta_{it999}(\dots) = (-0.00, 0.48, 0.53, -0.01)^T$$



$$R_{\Theta_{it3}}(\dots) = 0.08$$

$$\nabla R_{\Theta_{it3}}(\dots) = (-0.01, 0.09, -0.12, 0.04)^T$$

Partie 2 : Descente de gradient - Minimisation du risque en fonction de θ

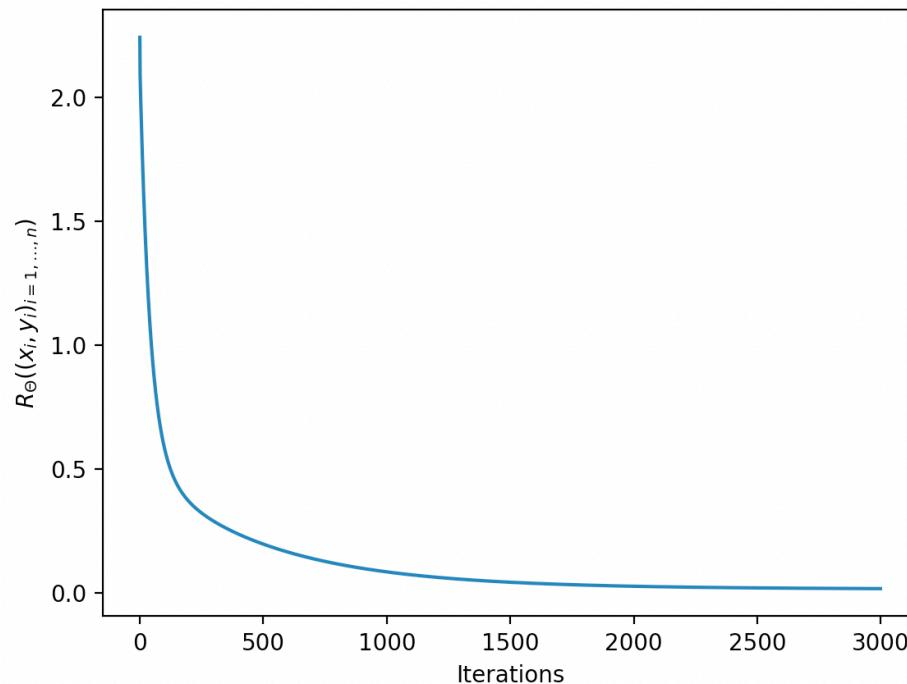
- Nous allons utiliser le **gradient du risque empirique** par rapport aux paramètres du modèle :

$$R_{\Theta}((x_i, y_i)_{i=1, \dots, n}) = \frac{1}{n} \sum_{i=1}^n (h_{\Theta}(x_i) - y_i)^2 = \frac{1}{n} \sum_{i=1}^n (w_0 + \sum_{j=1}^p w_j x_i^j - y_i)^2$$

$$\nabla R_{\Theta}(\dots) = \left(\frac{\partial R_{\Theta}(\dots)}{\partial w_0}, \frac{\partial R_{\Theta}(\dots)}{\partial w_1}, \dots, \frac{\partial R_{\Theta}(\dots)}{\partial w_p} \right)^T = \frac{1}{n} \sum_{i=1}^n 2 (h_{\Theta}(x_i) - y_i) (1, x_i^1, \dots, x_i^p)^T$$

Gradient de R_{Θ}

- Descent de gradient :



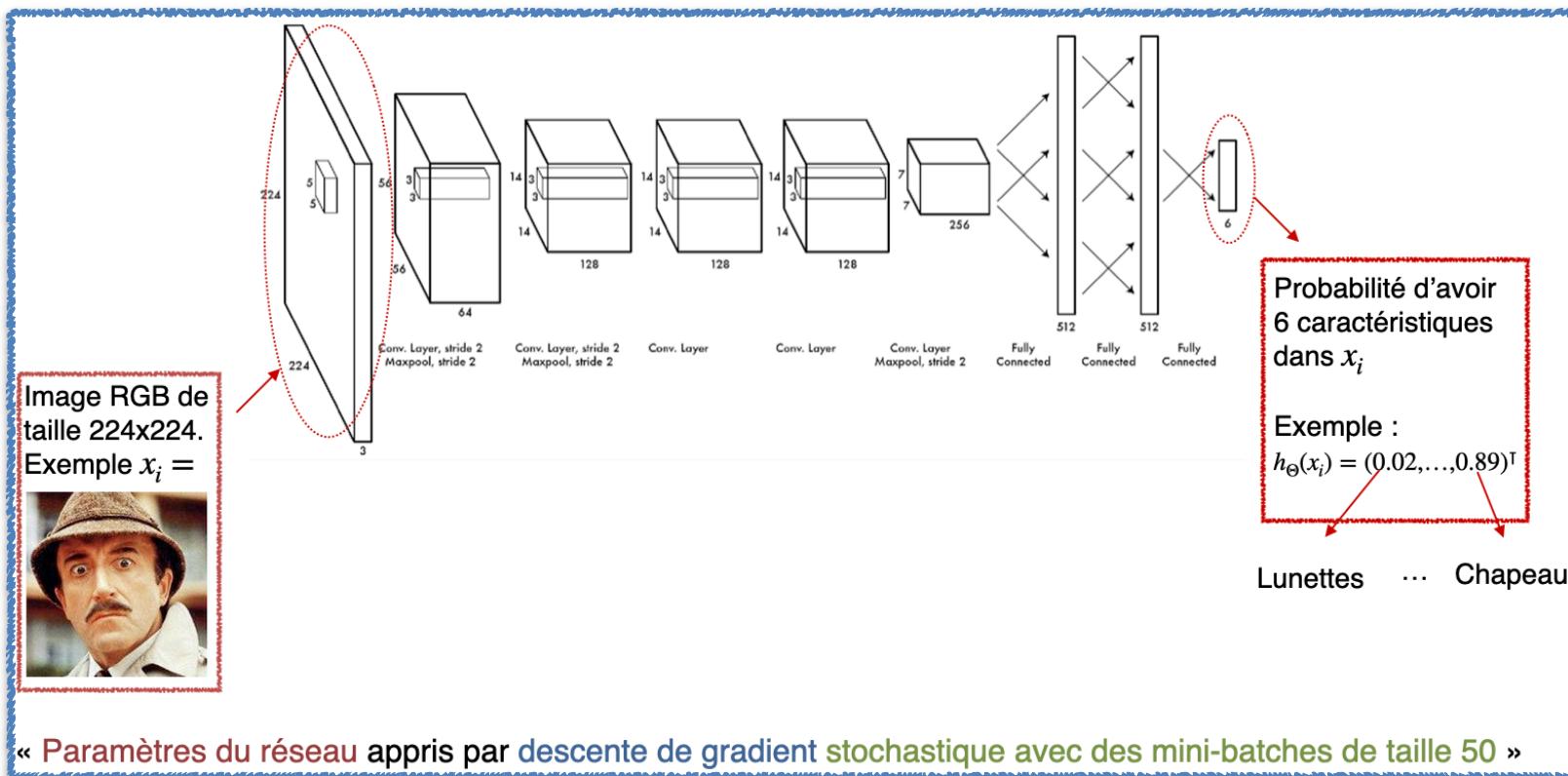
On a convergé vers une paramétrisation qui marche très bien **sur le jeu d'apprentissage !**

En pratique on fera de la validation croisée pour valider le modèle... mais ce n'est pas le point aujourd'hui.





Partie 3 : Apprentissage des paramètres d'un réseau de neurones



- Comment le réseau transforme couche après couche une image en probabilités ?
- Que ce qu'une descente de gradient ?
- Partie 3 : Comment calculer le gradient des paramètres d'un réseau de neurones (back-propagation) ?
- Stochastique... c'est quoi et quel est l'intérêt ?

- 3.1 Détail des calculs d'une prédiction (cas du perceptron multi-couches)
- 3.2 Calcul de l'erreur de prédiction
- 3.3 Détail des calculs du gradient de l'erreur (cas du perceptron multi-couches)
- 3.4 Descente de gradient classique et stochastique
- 3.5 Extension à des réseaux plus avancés

Partie 3 : Apprentissage des paramètres d'un réseau de neurones

Exemple d'utilisation de réseaux de neurones : Détection de caractéristiques dans une image



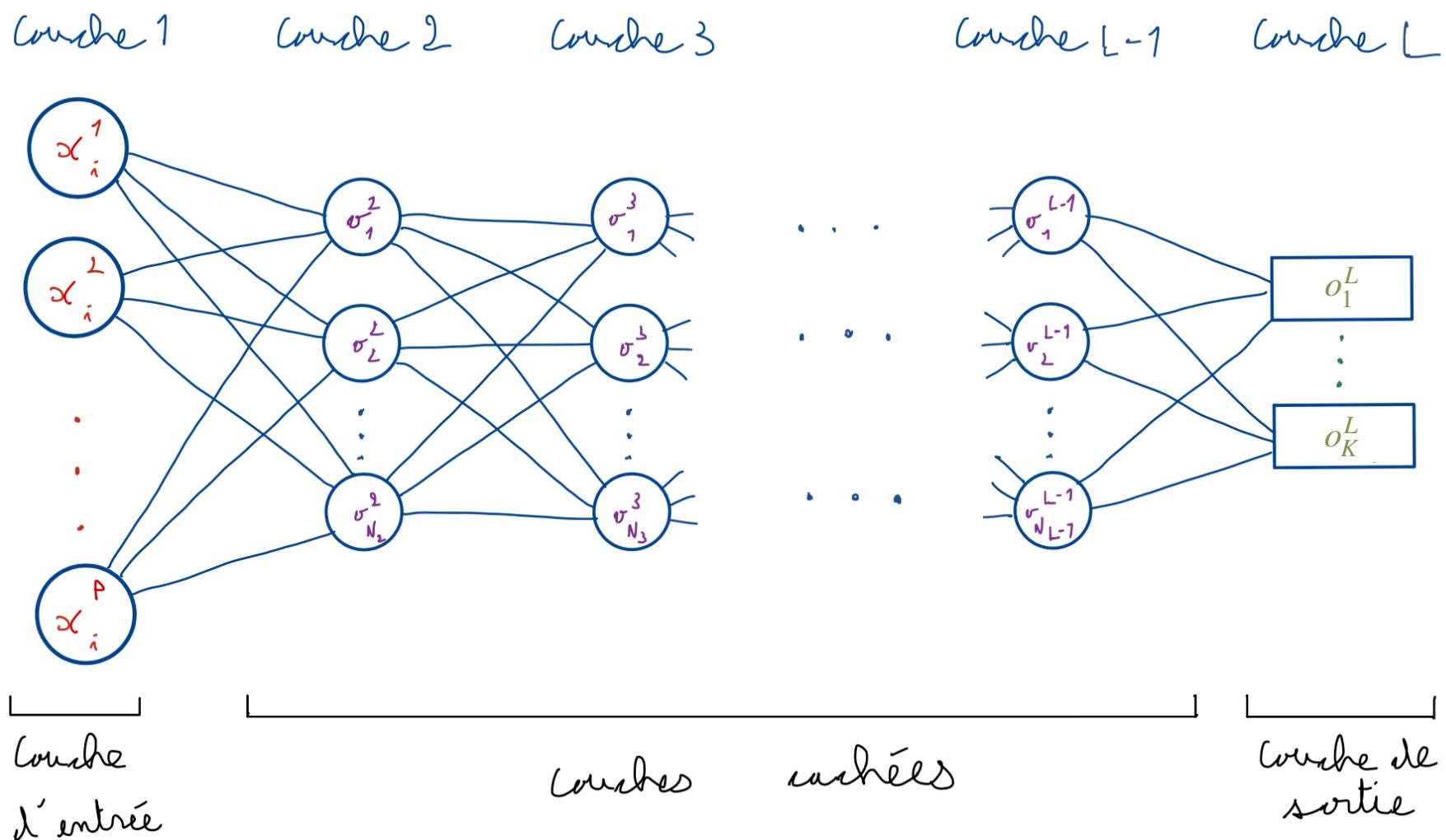
Une **entrée** $x_i = (x_i^1, x_i^2, \dots, x_i^p)$ est une image.

Une **sortie** $y_i = (y_i^1, y_i^2, \dots, y_i^K)$ est le fait d'observer K caractéristiques dans l'image.

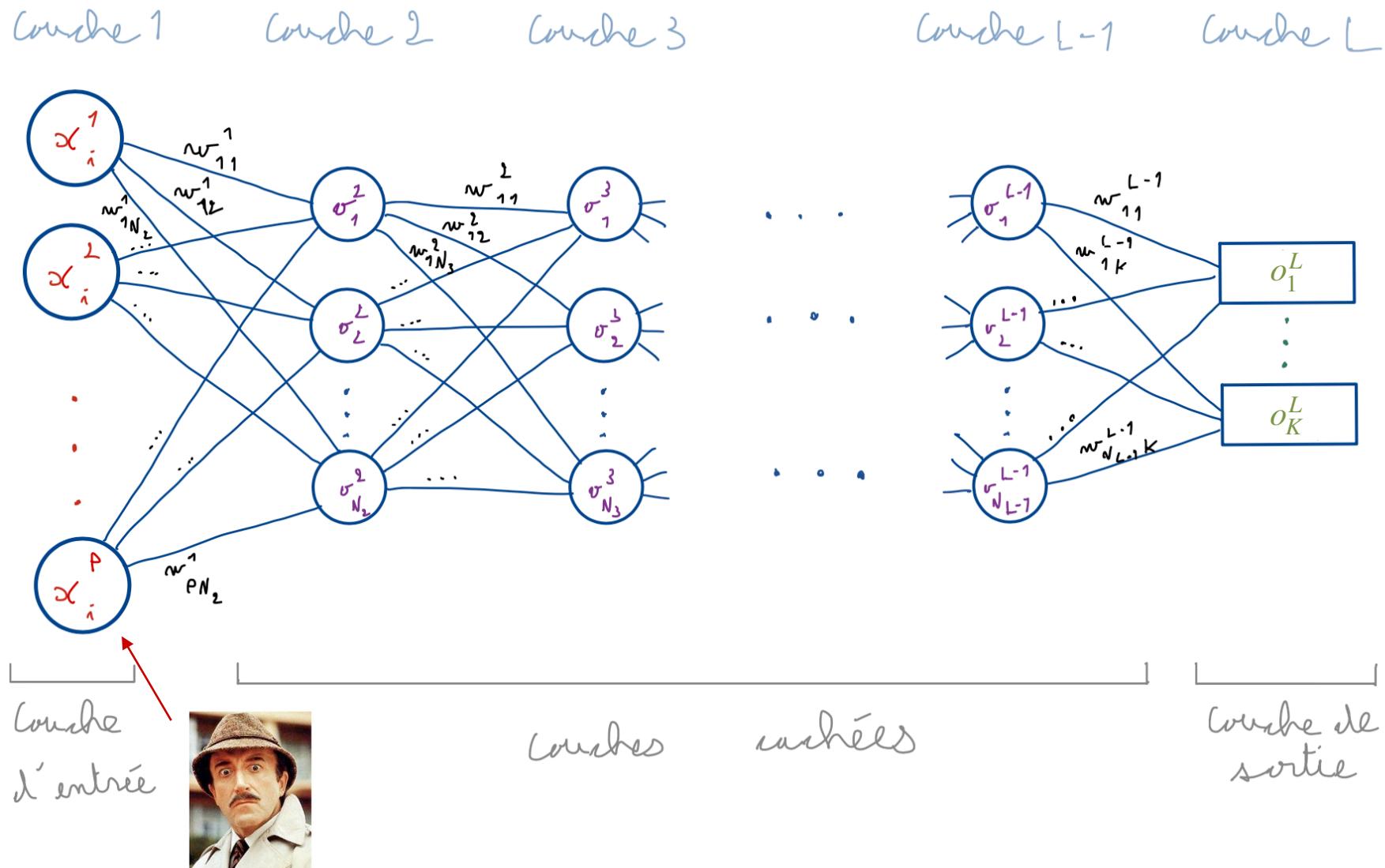
- $y_i^1 = 1$ si la personne a des lunettes et $y_i^1 = 0$ sinon ;
- $y_i^2 = 1$ si la personne sourit et $y_i^2 = 0$ sinon ;
- ...

Jeu de données *CelebA*

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.1 : Prédiction



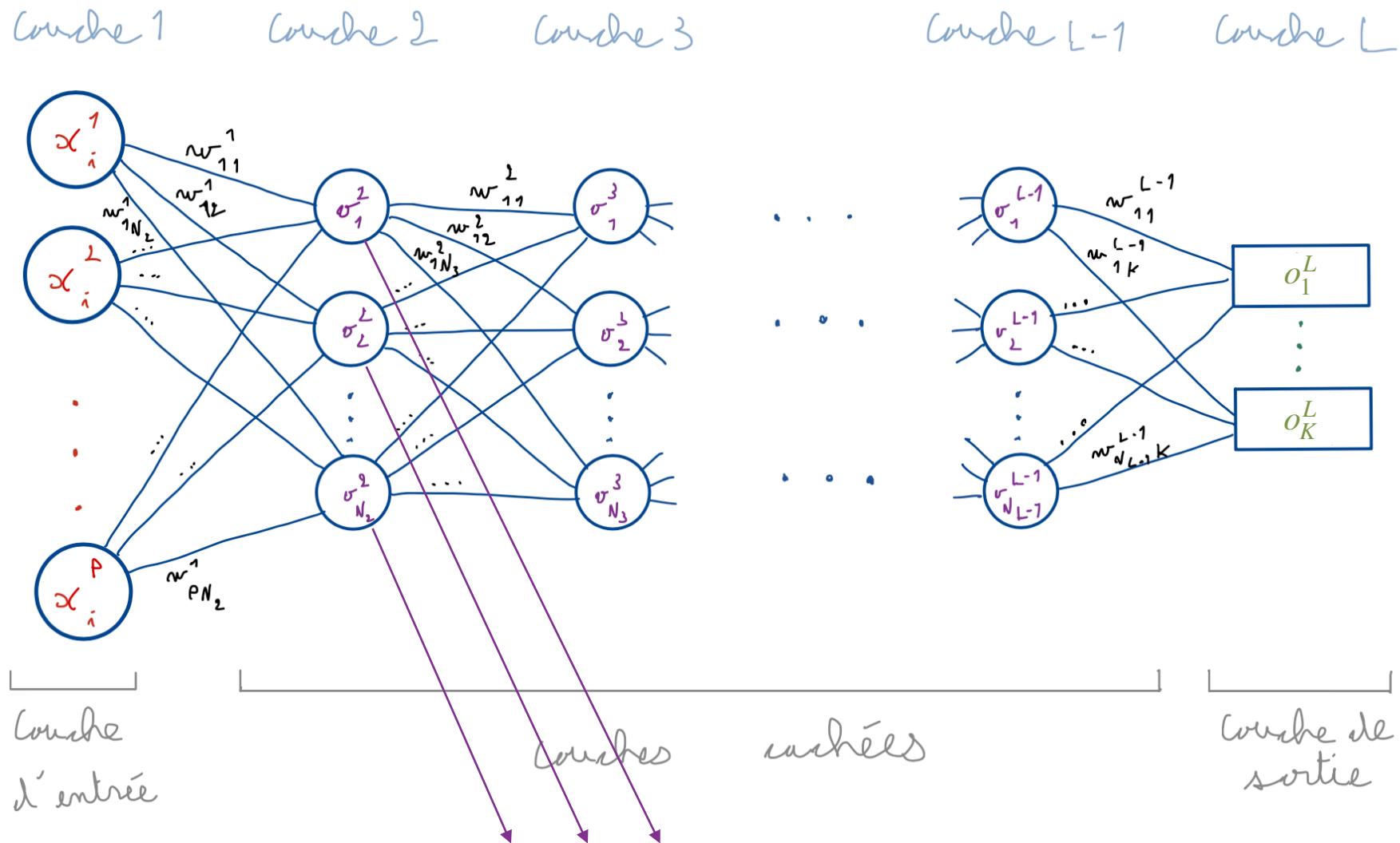
Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.1 : Prédiction



$\forall k = 1, \dots, p :$

$$o_k^1 = x_i^k$$

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.1 : Prédiction

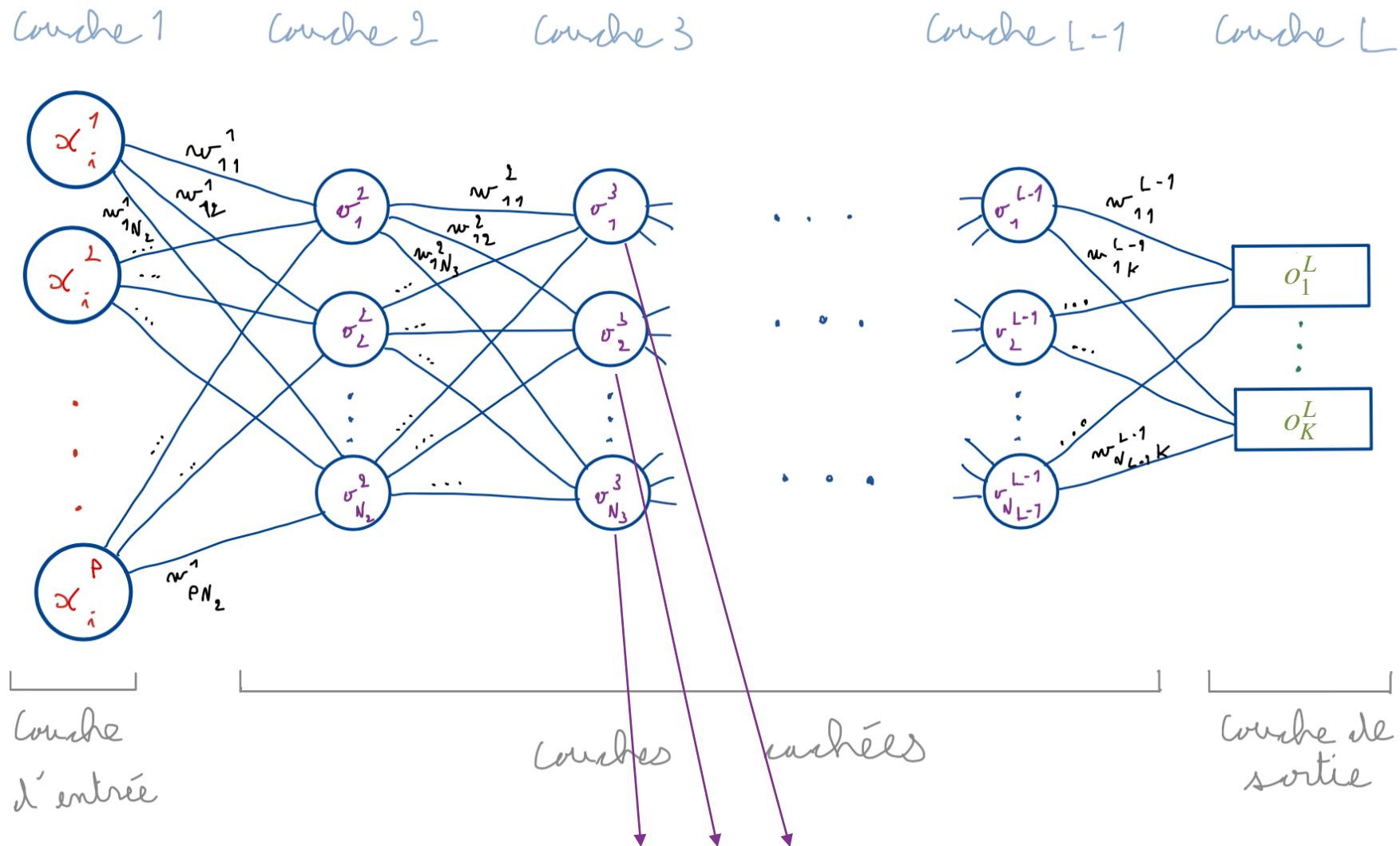


$$\forall k = 1, \dots, p : \\ o_k^1 = x_i^k$$

$$\begin{cases} s_k^{l+1} = \sum_{p \in \{1, \dots, N_l\}} w_{pk}^l o_p^l \\ o_k^{l+1} = \varphi(s_k^{l+1}) \end{cases}$$

$\varphi(\cdot)$ non-linéaire → ici $\varphi(s_k^{l+1}) = \text{ReLU}(s_k^{l+1}) = \max(0, s_k^{l+1})$

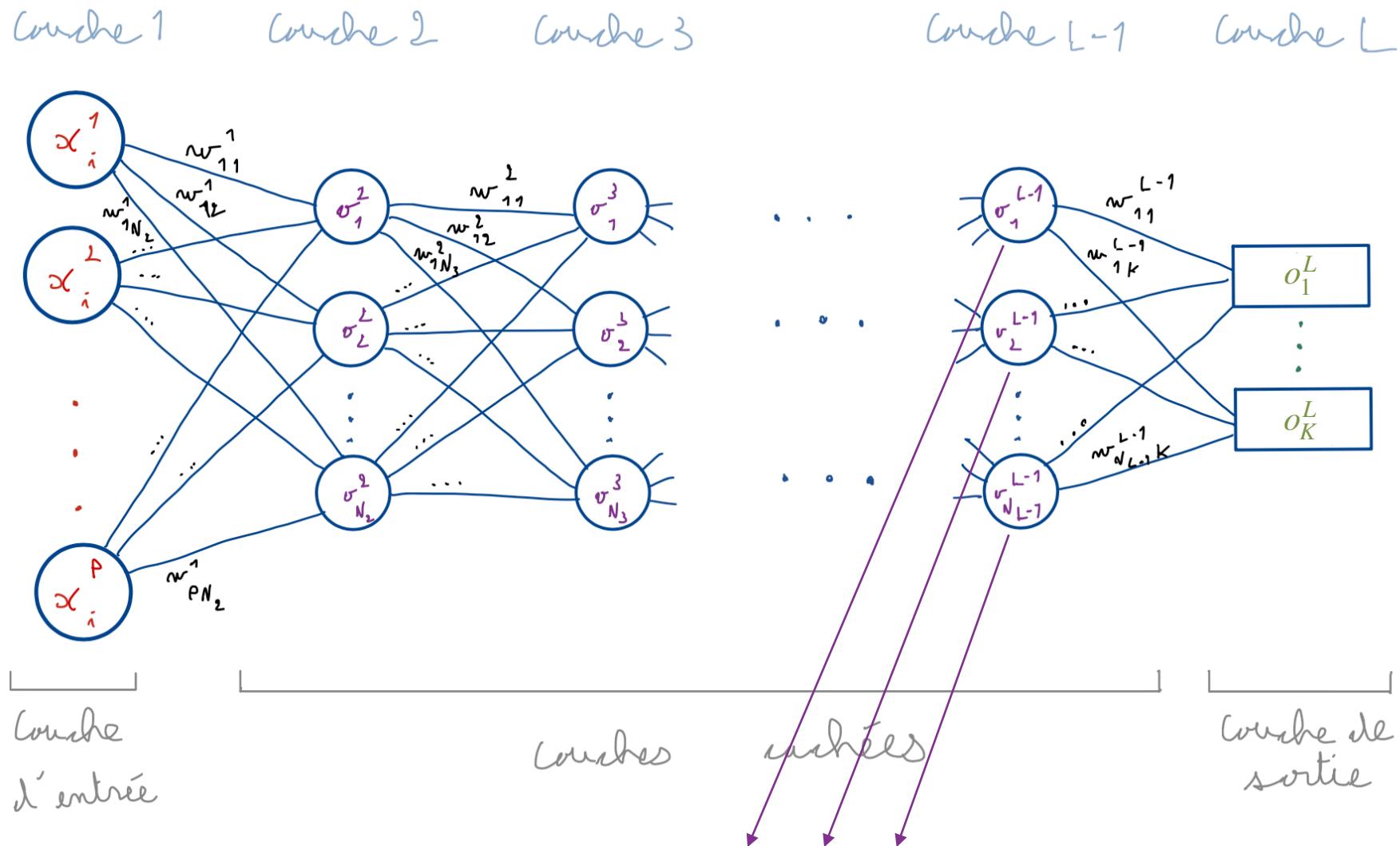
Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.1 : Prédiction



$$\begin{cases} s_k^{l+1} = \sum_{p \in \{1, \dots, N_l\}} w_{pk}^l o_p^l \\ o_k^{l+1} = \varphi(s_k^{l+1}) \end{cases}$$

$\varphi(\cdot)$ non-linéaire → ici $\varphi(s_k^{l+1}) = \text{ReLU}(s_k^{l+1}) = \max(0, s_k^{l+1})$

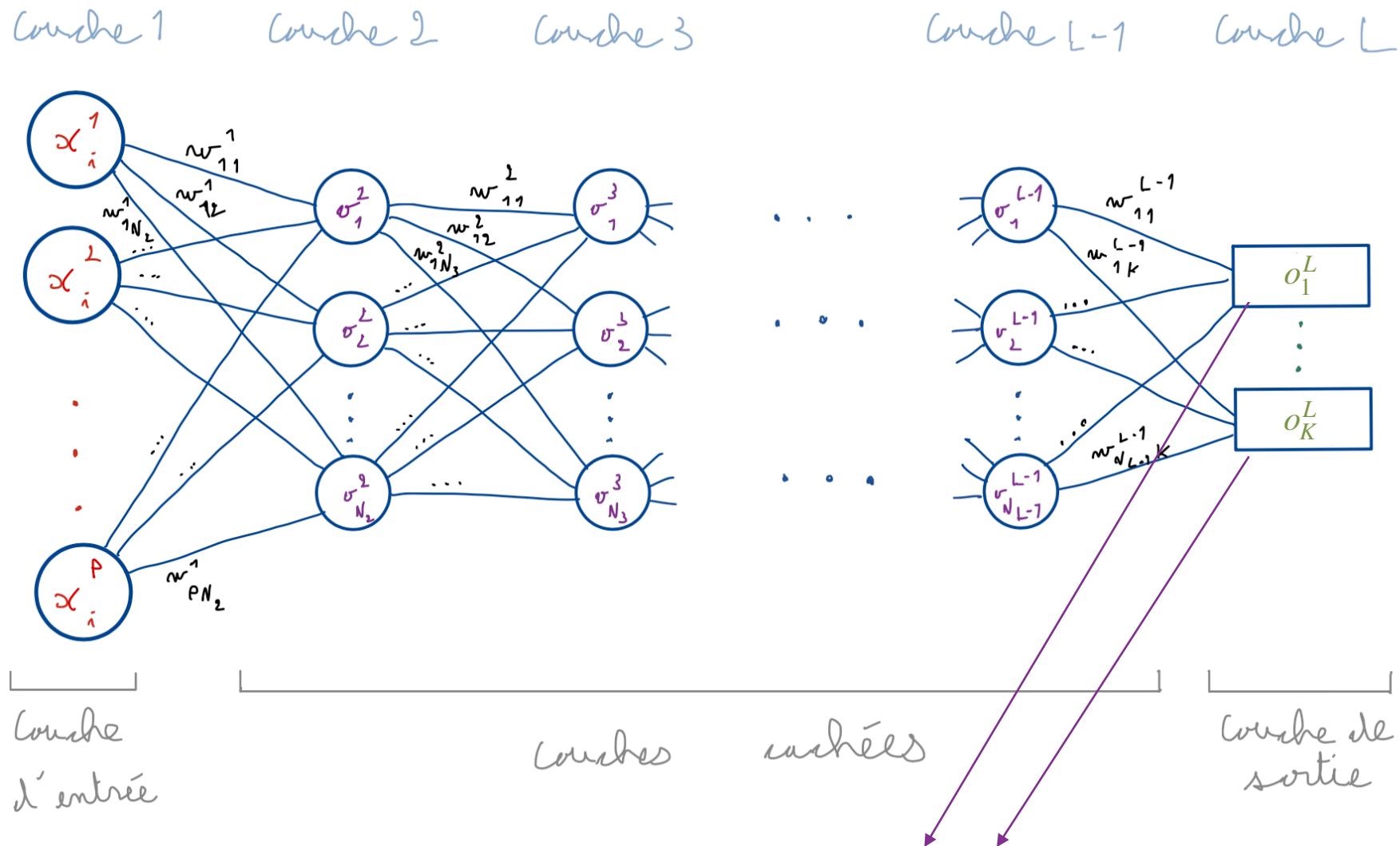
Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.1 : Prédiction



$$\begin{cases} s_k^{l+1} = \sum_{p \in \{1, \dots, N_l\}} w_{pk}^l o_p^l \\ o_k^{l+1} = \varphi(s_k^{l+1}) \end{cases}$$

$\varphi(\cdot)$ non-linéaire → ici $\varphi(s_k^{l+1}) = \text{ReLU}(s_k^{l+1}) = \max(0, s_k^{l+1})$

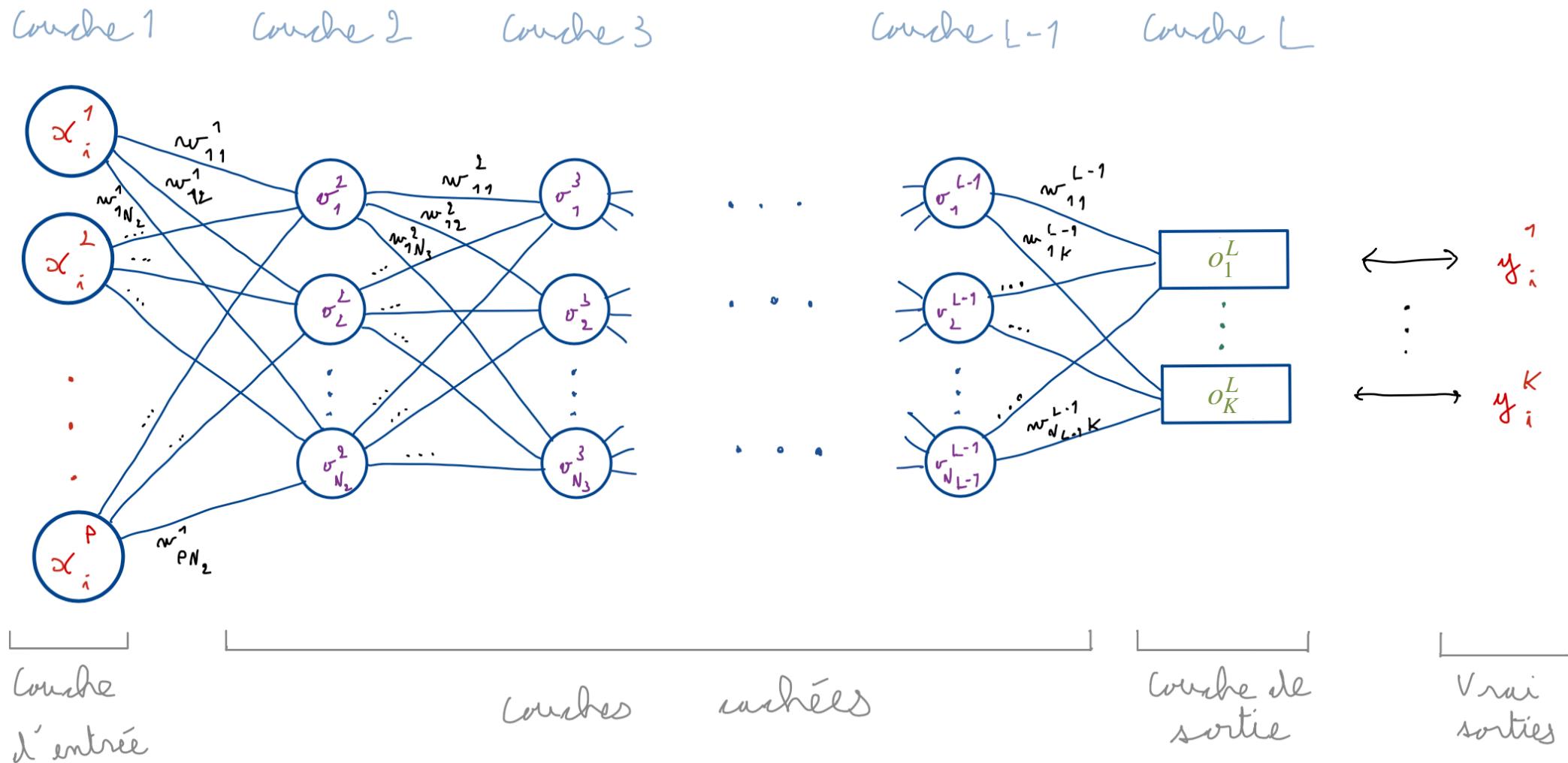
Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.1 : Prédiction



$$\begin{cases} s_k^{l+1} = \sum_{p \in \{1, \dots, N_l\}} w_{pk}^l o_p^l \\ o_k^{l+1} = \varphi(s_k^{l+1}) \end{cases}$$

$\varphi(\cdot)$ non-linéaire → ici $\varphi(s_k^{l+1}) = \text{logistic}(s_k^{l+1}) = 1/(1 + e^{-s_k^{l+1}})$

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.1 : Prédiction

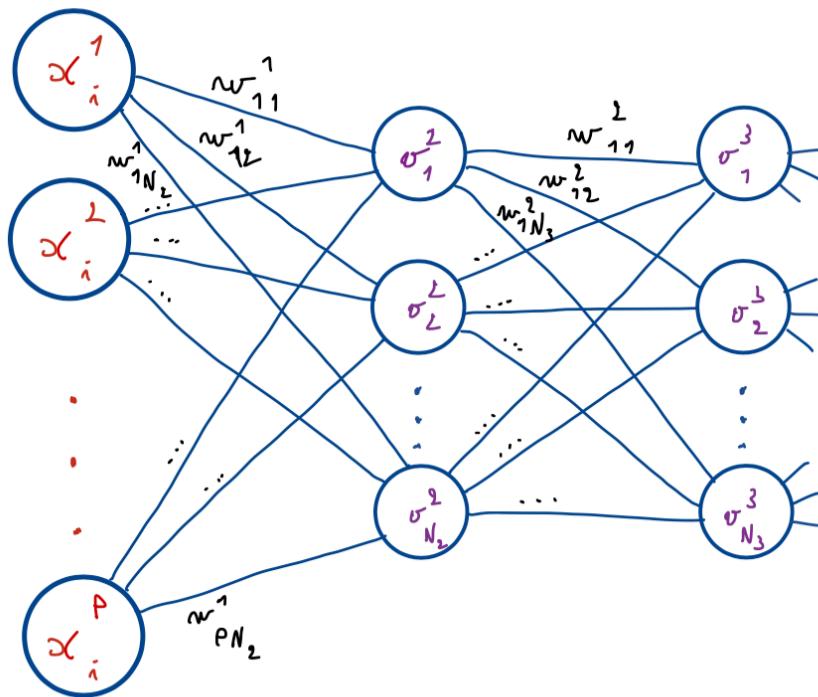


Prédiction : $h_{\Theta}(x_i) = (o_1^L, \dots, o_K^L)$

Valeurs à prédire : $y_i = (y_i^1, \dots, y_i^K)$

Remarque : les indices i ne sont pas notés dans les o_k^l pour simplifier les notations.

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.2 : Calcul de l'erreur

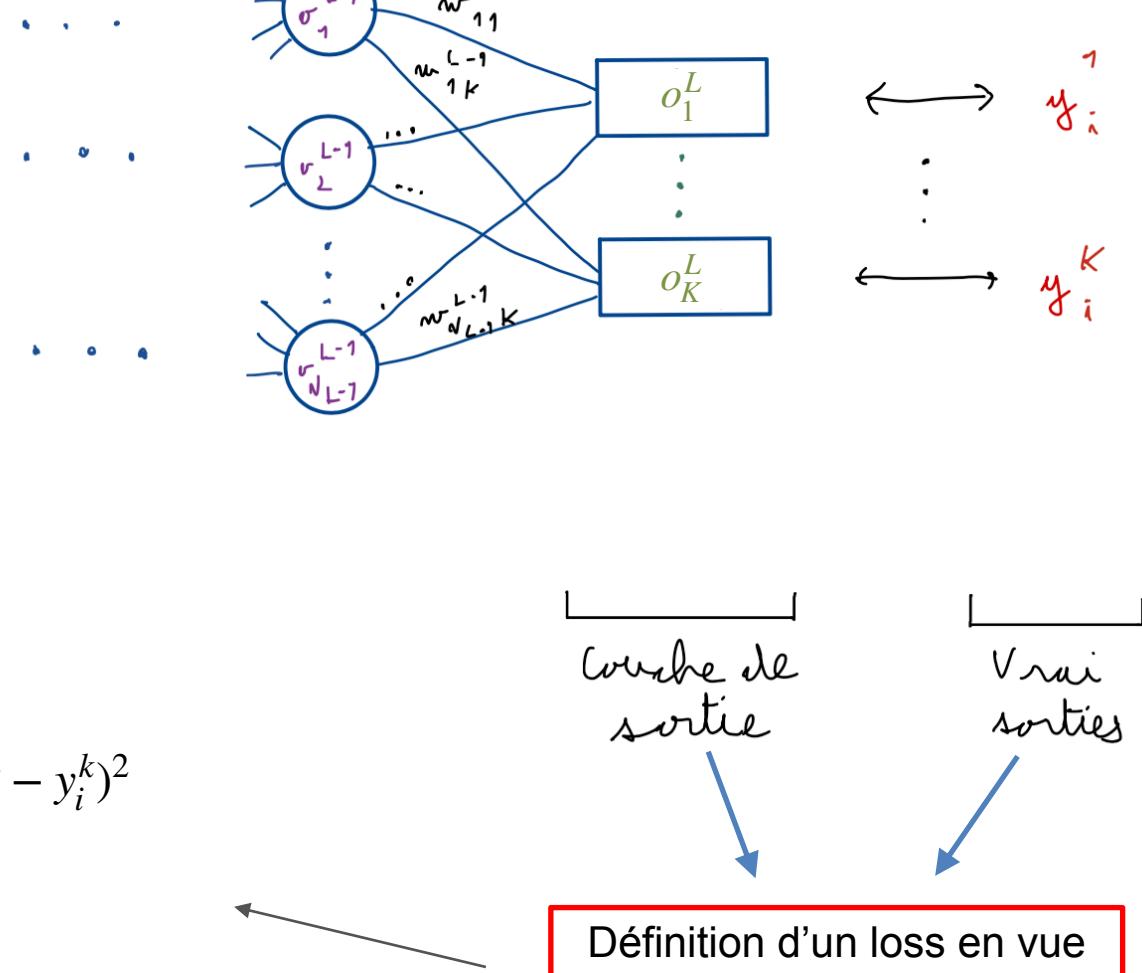


$$\text{Loss possible : } loss(h_{\Theta}(x_i), y_i) = \sum_{k=1}^K (o_k^L - y_i^k)^2$$

avec : $h_{\Theta}(x_i) = (o_1^L, \dots, o_K^L)$

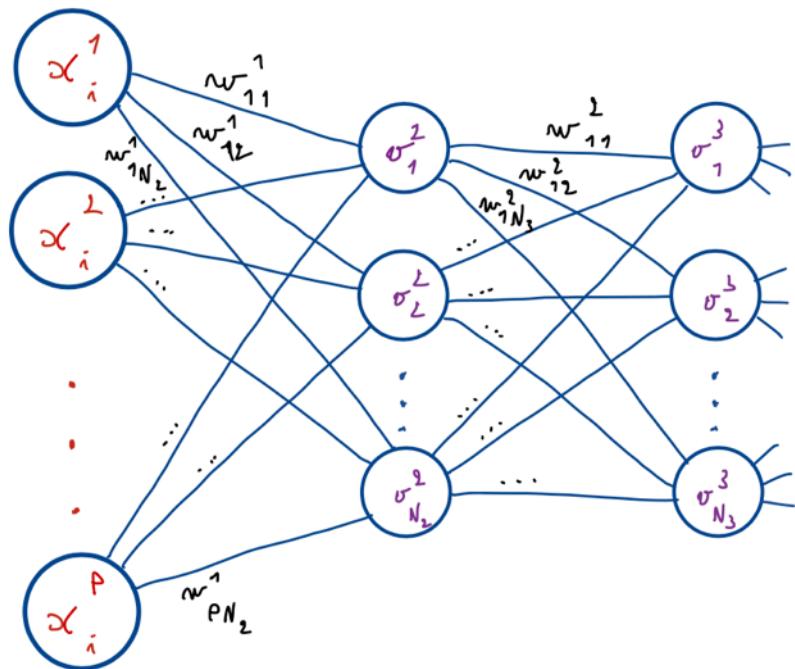
Remarque :

- Il existe d'autres losses comme la *Cross-entropy*.
 - Le loss doit être dérivable par rapport aux o_k^L

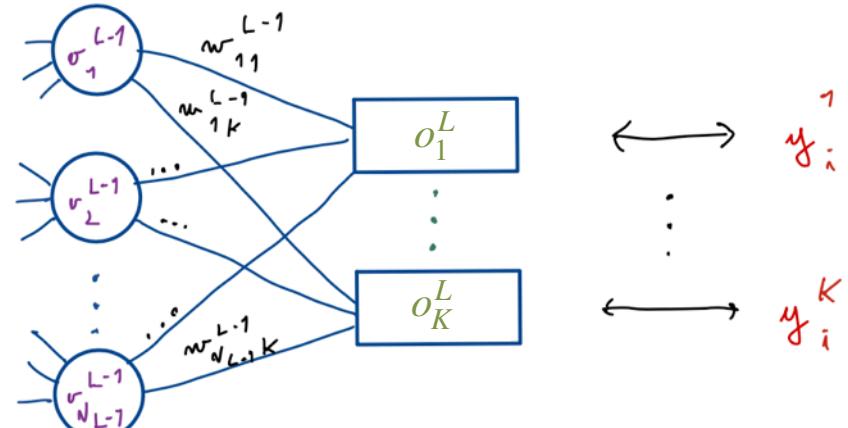


Définition d'un loss en vue
de l'apprentissage des
paramètres $\Theta = \{w_{pk}^l\}_{p,k,l}$

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.3 : Gradient de l'erreur



...
...
...



y_i^1
⋮
 y_i^K

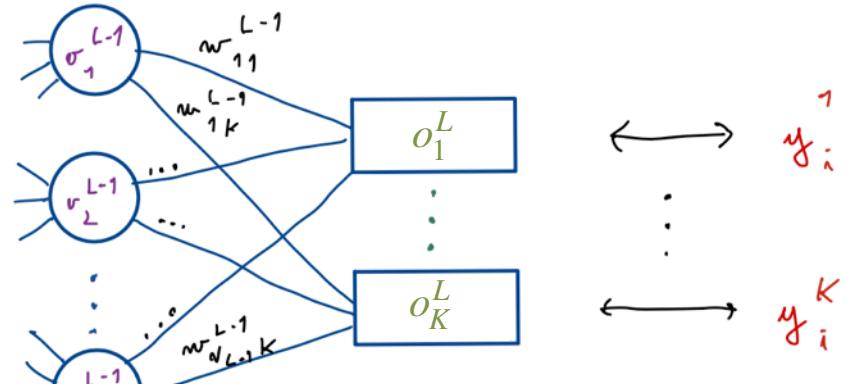
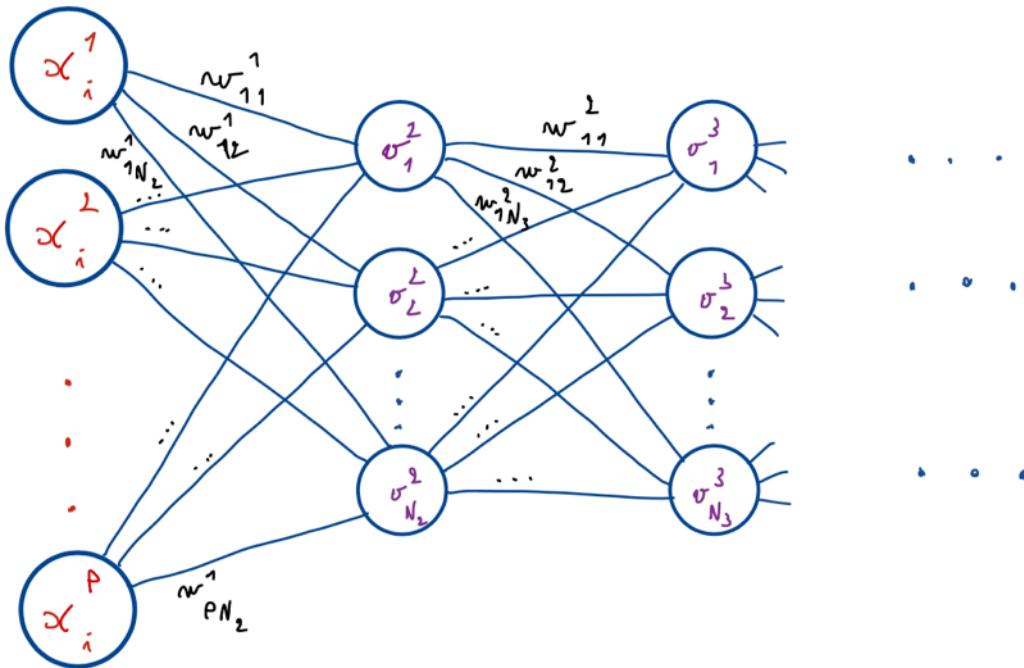
Avec : $h_\Theta(x_i) = (o_1^L, \dots, o_K^L)$
 $y_i = (y_i^1, \dots, y_i^K)$

Recherche des paramètres optimaux $\hat{\Theta}$ tels que :

$$\begin{aligned}\hat{\Theta} &= \arg \min_{\Theta=\{w_0, \dots, w_p\}} R_\Theta ((x_i, y_i)_{i=1, \dots, n}) \\ &= \arg \min_{\Theta=\{w_0, \dots, w_p\}} \frac{1}{n} \sum_{i=1}^n \text{loss}(h_\Theta(x_i), y_i)\end{aligned}$$

→ descente de gradient en grande dimension

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.3 : Gradient de l'erreur



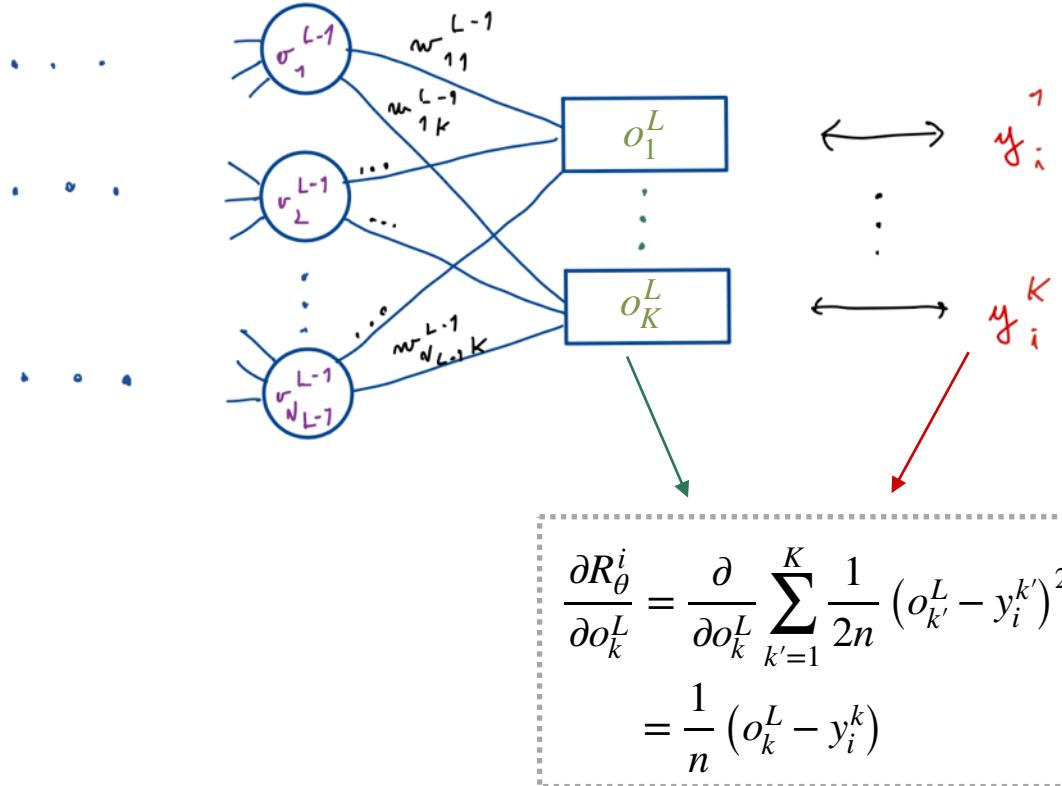
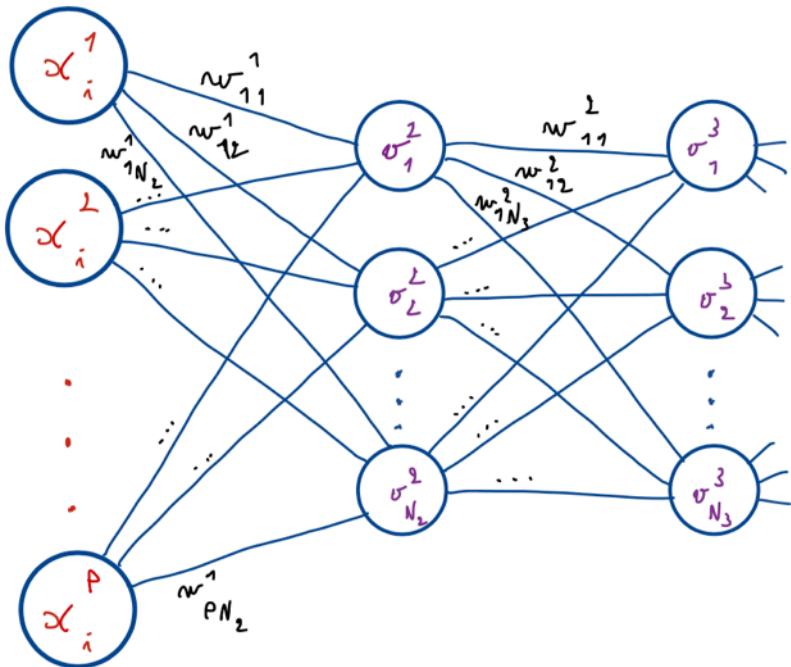
$$\nabla R_{\Theta}((x_i, y_i)_{i=1, \dots, n}) = \left(\frac{\partial R_{\Theta}(\dots)}{\partial w_{1,1}^1}, \frac{\partial R_{\Theta}(\dots)}{\partial w_{1,2}^1}, \dots, \frac{\partial R_{\Theta}(\dots)}{\partial w_{N_{L-1},K}^{N_{L-1}}} \right)^{\top} \quad (\text{Gradient du risque empirique})$$

$$= \frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K \left(\frac{\partial \text{loss}(h^k(x_i), y_i^k)}{\partial w_{1,1}^1}, \frac{\partial \text{loss}(h^k(x_i), y_i^k)}{\partial w_{1,2}^1}, \dots, \frac{\partial \text{loss}(h^k(x_i), y_i^k)}{\partial w_{N_{L-1},K}^{N_{L-1}}} \right)^{\top}$$

$$= \frac{1}{n} \sum_{i=1}^n \sum_{k=1}^K \left(\frac{\partial (h^k(x_i) - y_i^k)^2}{\partial w_{1,1}^1}, \frac{\partial (h^k(x_i) - y_i^k)^2}{\partial w_{1,2}^1}, \dots, \frac{\partial (h^k(x_i) - y_i^k)^2}{\partial w_{N_{L-1},K}^{N_{L-1}}} \right)^{\top}$$

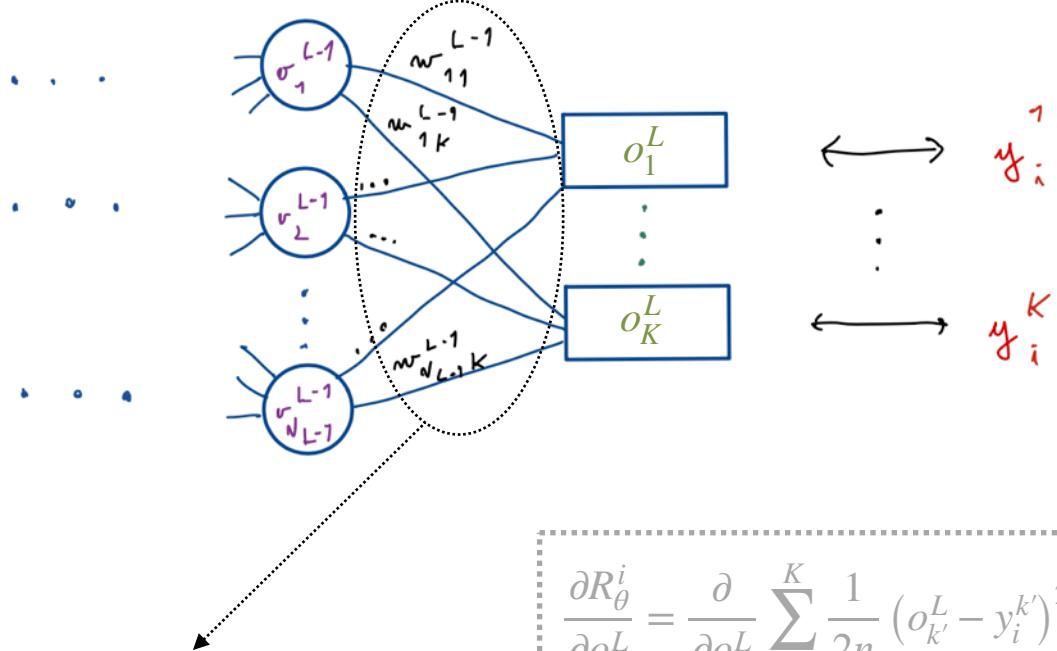
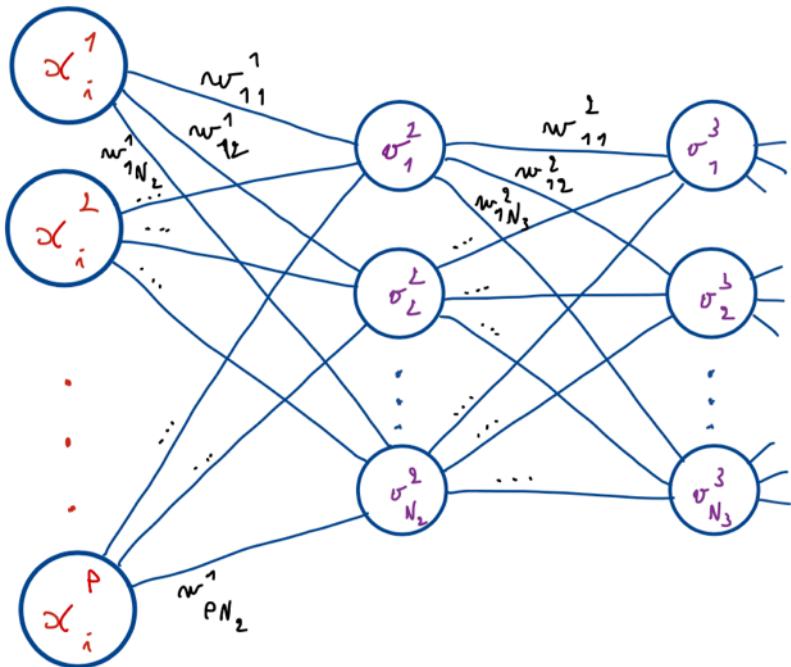
Calcul analytique possible mais les $h^k(x_i)$ sont issus de fonctions composées couches après couches
→ **dérivation avec le théorème des fonctions composées (*chain rule*) !**

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.3 : Gradient de l'erreur



Notation : $R_\theta^i = loss(h(x_i), y_i) = \frac{1}{2} \sum_{k'=1}^L (o_{k'}^L - y_i^{k'})^2$

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.3 : Gradient de l'erreur



$$\frac{\partial R_{\Theta}^i}{\partial w_{pk}^l} = \frac{\partial R_{\Theta}^i}{\partial o_k^{l+1}} \frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} \frac{\partial s_k^{l+1}}{\partial w_{pk}^l}$$

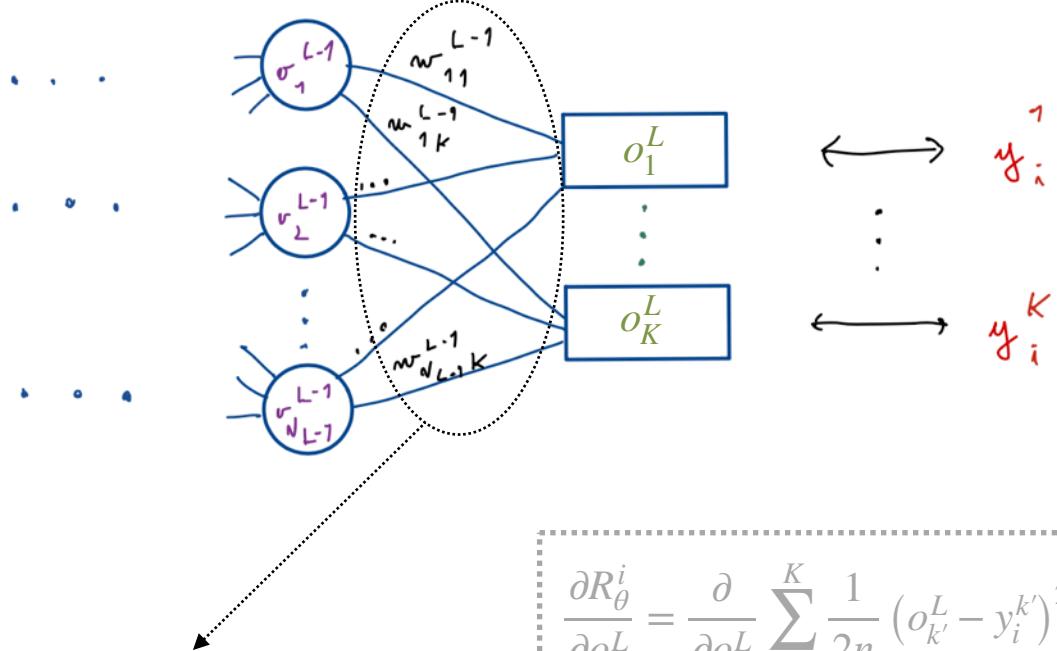
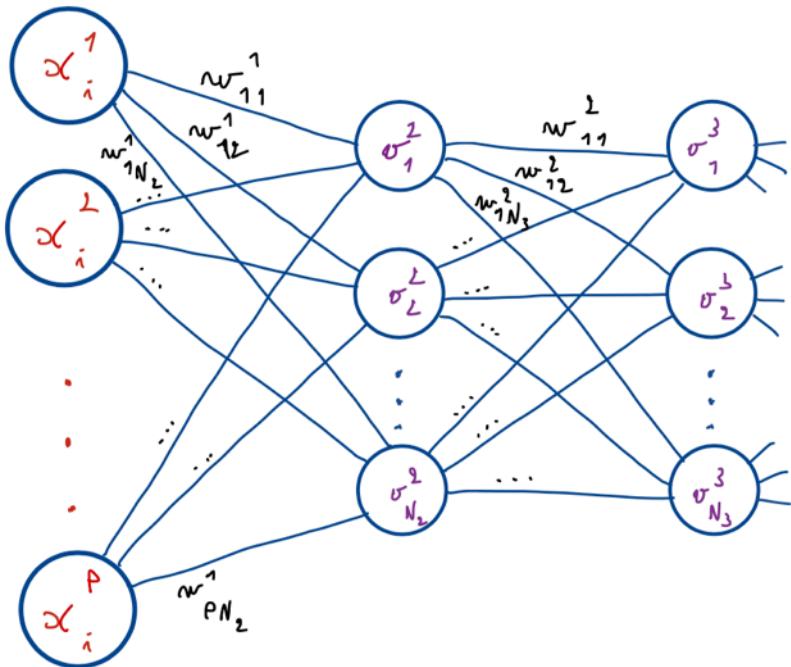
$$l = L - 1$$

$$\frac{\partial R_\theta^i}{\partial o_k^L} = \frac{\partial}{\partial o_k^L} \sum_{k'=1}^K \frac{1}{2n} (o_{k'}^L - y_i^{k'})^2$$

$$= \frac{1}{n} (o_k^L - y_i^k)$$

Usage de la « chaine rule »

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.3 : Gradient de l'erreur



$$\frac{\partial R_{\Theta}^i}{\partial w_{pk}^l} = \frac{\partial R_{\Theta}^i}{\partial o_k^{l+1}} \frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} \frac{\partial s_k^{l+1}}{\partial w_{pk}^l}$$

$$l = L - 1$$

$$\frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} = \frac{\partial}{\partial s_k^{l+1}} \varphi(s_k^{l+1}) \text{ [connu]}$$

$$l = L - 1$$

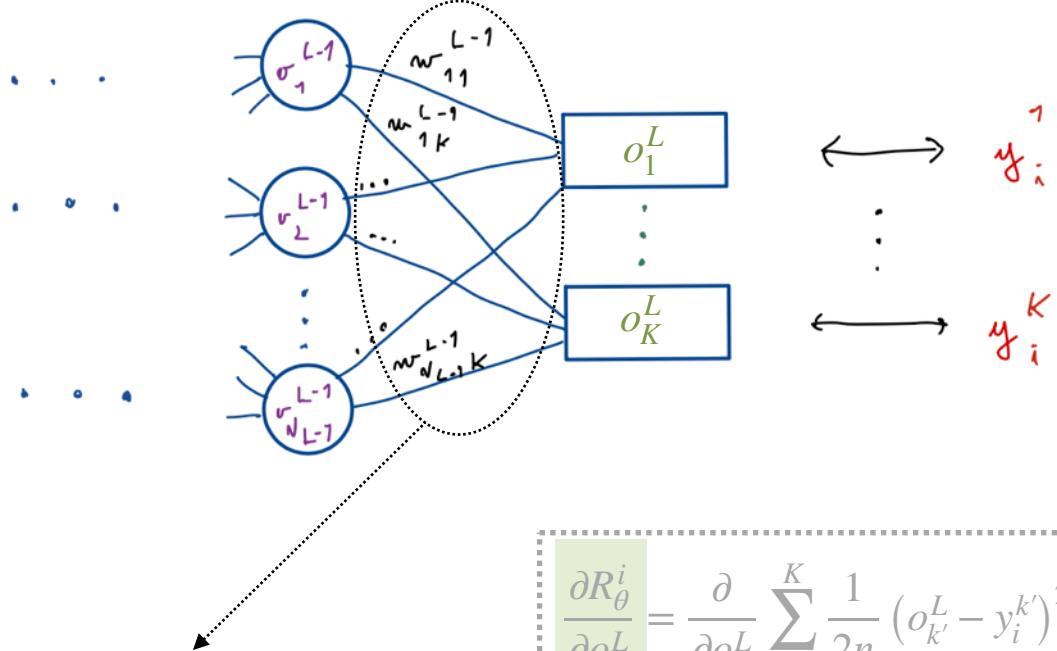
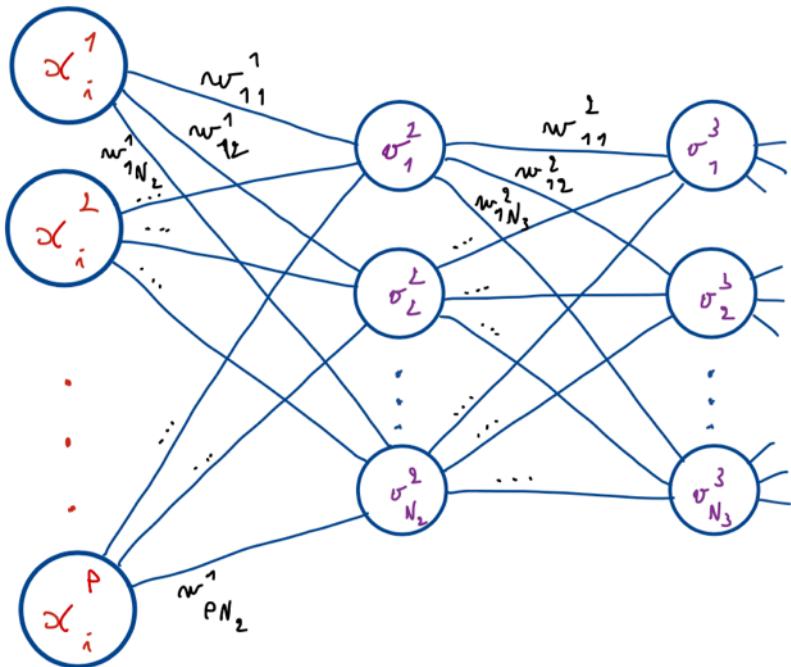
$$\frac{\partial R_\theta^i}{\partial o_k^L} = \frac{\partial}{\partial o_k^L} \sum_{k'=1}^K \frac{1}{2n} (o_{k'}^L - y_i^{k'})^2$$

$$= \frac{1}{n} (o_k^L - y_i^k)$$

$$\begin{aligned}\frac{\partial s_k^{l+1}}{\partial w_{pk}^l} &= \frac{\partial}{\partial w_{pk}^l} \sum_{p' \in \{1, \dots, N_l\}} w_{p'k}^l o_{p'}^l \\ &= \frac{\partial}{\partial w_{pk}^l} w_{pk}^l o_p^l \\ &= o_p^l\end{aligned}$$

$$l = L - 1$$

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.3 : Gradient de l'erreur



$$\frac{\partial R_{\Theta}^i}{\partial w_{pk}^l} = \frac{\partial R_{\Theta}^i}{\partial o_k^{l+1}} \frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} \frac{\partial s_k^{l+1}}{\partial w_{pk}^l}$$

$$l = L - 1$$

$$\frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} = \frac{\partial}{\partial s_k^{l+1}} \varphi(s_k^{l+1}) \text{ [connu]}$$

$$l = L - 1$$

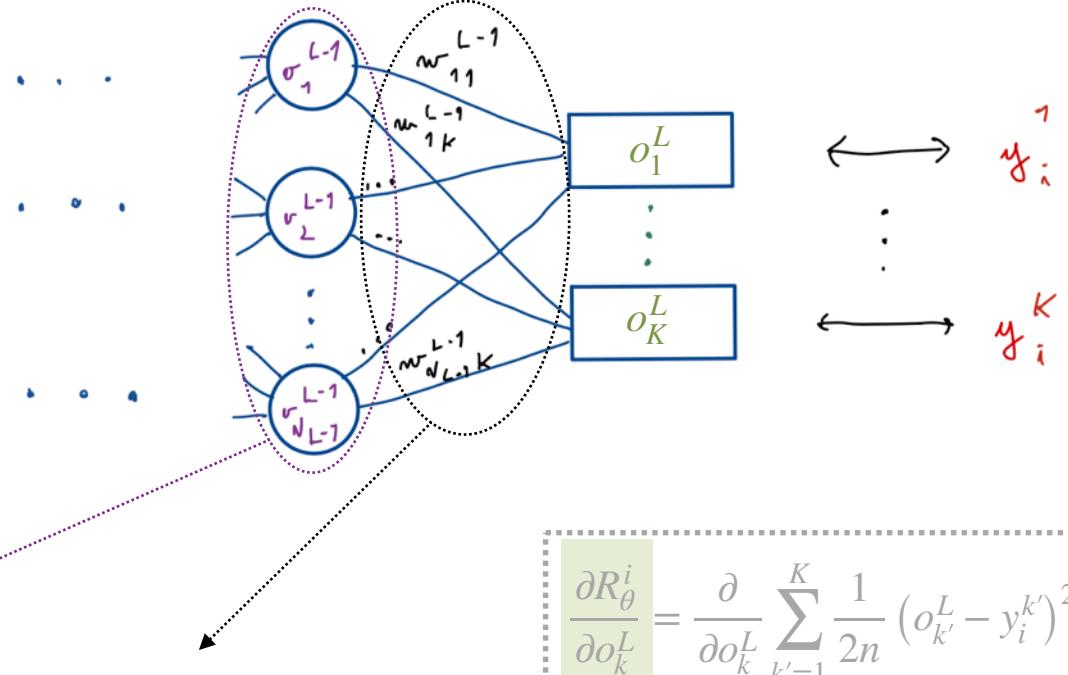
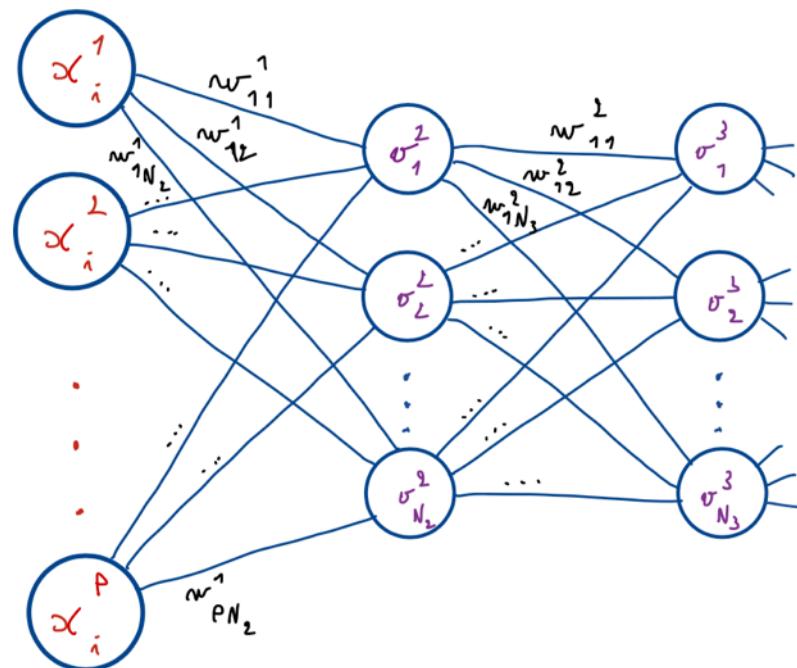
$$\frac{\partial R_\theta^i}{\partial o_k^L} = \frac{\partial}{\partial o_k^L} \sum_{k'=1}^K \frac{1}{2n} (o_{k'}^L - y_i^{k'})^2$$

$$= \frac{1}{n} (o_k^L - y_i^k)$$

$$\begin{aligned}\frac{\partial s_k^{l+1}}{\partial w_{pk}^l} &= \frac{\partial}{\partial w_{pk}^l} \sum_{p' \in \{1, \dots, N_l\}} w_{p'k}^l o_{p'}^l \\ &= \frac{\partial}{\partial w_{pk}^l} w_{pk}^l o_p^l \\ &= o_p^l\end{aligned}$$

$$l = L - 1$$

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.3 : Gradient de l'erreur



$$\frac{\partial R_\Theta^i}{\partial o_k^l} = \sum_{p=1}^{N_{l+1}} \underbrace{\frac{\partial R_\Theta^i}{\partial o_k^{l+1}}}_{\text{Connu à la couche } l+1} \underbrace{\frac{\partial o_k^{l+1}}{\partial s_k^{l+1}}}_{\text{Derivee de } \varphi} \underbrace{\frac{\partial s_k^{l+1}}{\partial o_k^l}}_{=w_{pk}^l}$$

$$\frac{\partial R_\Theta^i}{\partial w_{pk}^l} = \frac{\partial R_\Theta^i}{\partial o_k^{l+1}} \frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} \frac{\partial s_k^{l+1}}{\partial w_{pk}^l}$$

$$\begin{aligned} \frac{\partial R_\Theta^i}{\partial o_k^L} &= \frac{\partial}{\partial o_k^L} \sum_{k'=1}^K \frac{1}{2n} (o_{k'}^L - y_i^{k'})^2 \\ &= \frac{1}{n} (o_k^L - y_i^k) \end{aligned}$$

$l = L - 1$

Usage de la « chaîne rule »

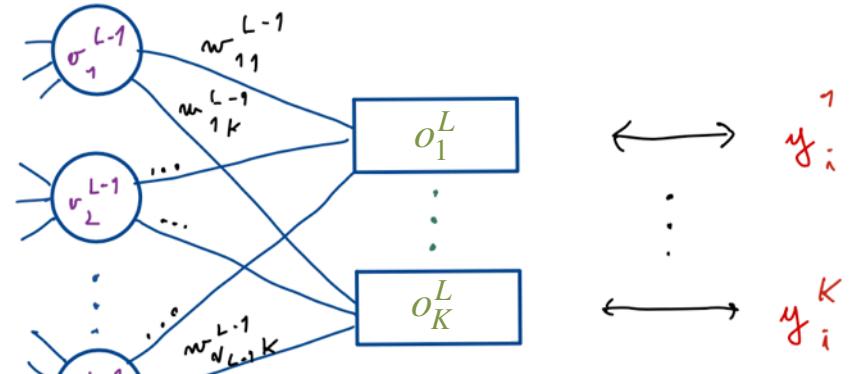
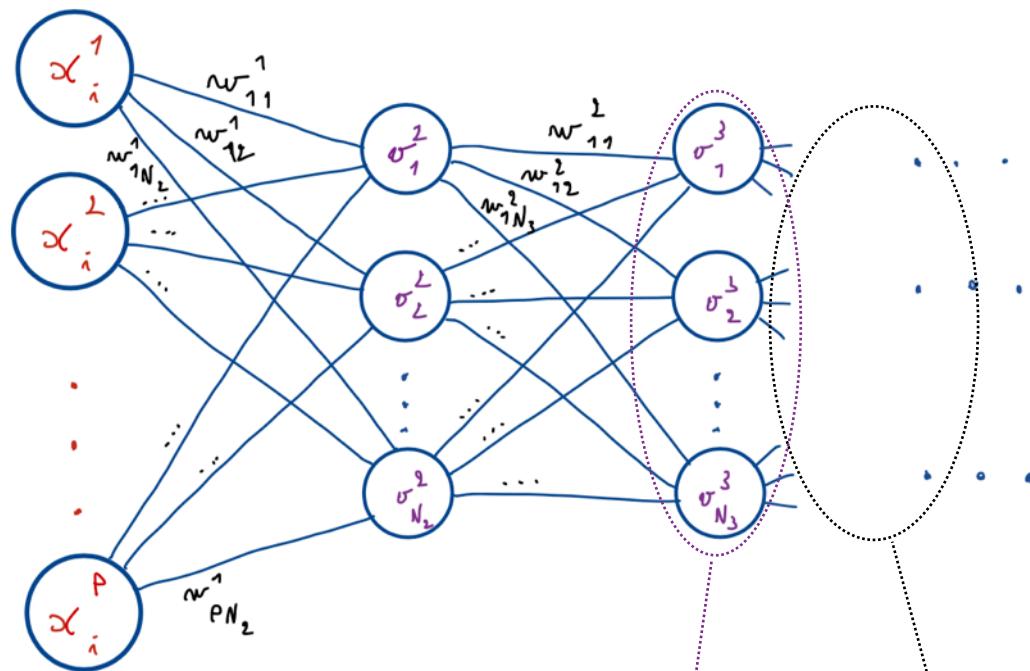
$$\frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} = \frac{\partial}{\partial s_k^{l+1}} \varphi(s_k^{l+1}) \quad [\text{connu}]$$

$l = L - 1$

$$\begin{aligned} \frac{\partial s_k^{l+1}}{\partial w_{pk}^l} &= \frac{\partial}{\partial w_{pk}^l} \sum_{p' \in \{1, \dots, N_l\}} w_{p'k}^l o_{p'}^l \\ &= \frac{\partial}{\partial w_{pk}^l} w_{pk}^l o_p^l \\ &= o_p^l \end{aligned}$$

$l = L - 1$

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.3 : Gradient de l'erreur



$$\frac{\partial R_\Theta^i}{\partial o_k^l} = \sum_{p=1}^{N_{l+1}} \underbrace{\frac{\partial R_\Theta^i}{\partial o_k^{l+1}}}_{\text{Connu à la couche } l+1} \underbrace{\frac{\partial o_k^{l+1}}{\partial s_k^{l+1}}}_{\text{Derivee de } \varphi} \underbrace{\frac{\partial s_k^{l+1}}{\partial o_k^l}}_{=w_{pk}^l}$$

$$\frac{\partial R_\Theta^i}{\partial w_{pk}^l} = \frac{\partial R_\Theta^i}{\partial o_k^{l+1}} \frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} \frac{\partial s_k^{l+1}}{\partial w_{pk}^l}$$

$l = 3$

Usage de la « chaîne rule »

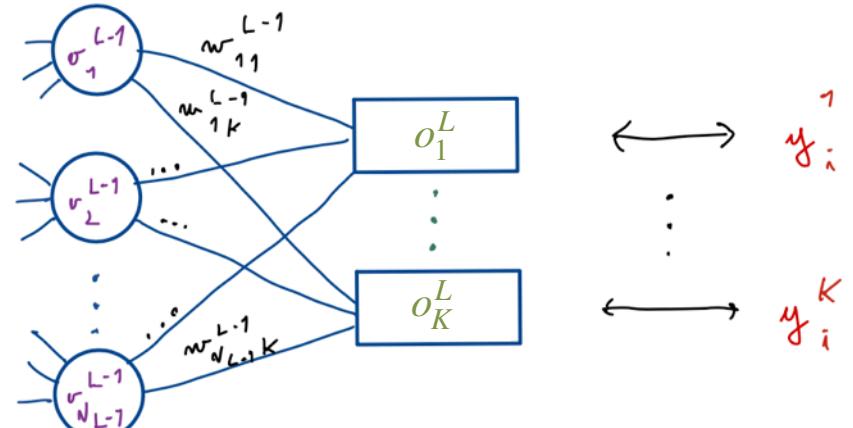
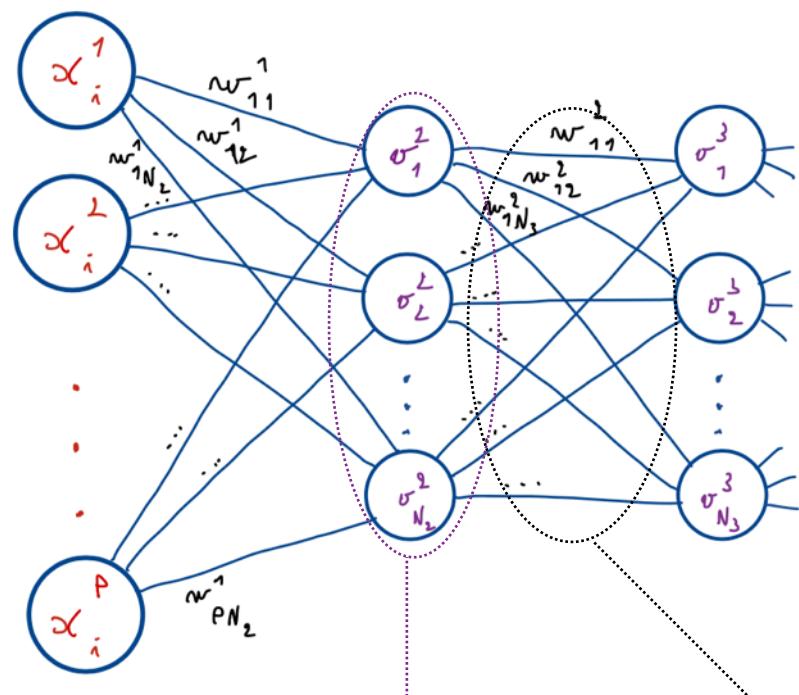
$$\frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} = \frac{\partial}{\partial s_k^{l+1}} \varphi(s_k^{l+1}) \quad [\text{connu}]$$

$l = 3$

$$\begin{aligned} \frac{\partial s_k^{l+1}}{\partial w_{pk}^l} &= \frac{\partial}{\partial w_{pk}^l} \sum_{p' \in \{1, \dots, N_l\}} w_{p'k}^l o_{p'}^l \\ &= \frac{\partial}{\partial w_{pk}^l} w_{pk}^l o_p^l \\ &= o_p^l \end{aligned}$$

$l = 3$

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.3 : Gradient de l'erreur



$$\frac{\partial R_\Theta^i}{\partial o_k^l} = \sum_{p=1}^{N_{l+1}} \underbrace{\frac{\partial R_\Theta^i}{\partial o_k^{l+1}}}_{\text{Connu à la couche } l+1} \underbrace{\frac{\partial o_k^{l+1}}{\partial s_k^{l+1}}}_{\text{Derivée de } \varphi} \underbrace{\frac{\partial s_k^{l+1}}{\partial o_k^l}}_{=w_{pk}^l}$$

$$\frac{\partial R_\Theta^i}{\partial w_{pk}^l} = \frac{\partial R_\Theta^i}{\partial o_k^{l+1}} \frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} \frac{\partial s_k^{l+1}}{\partial w_{pk}^l}$$

$l = 2$

Usage de la « chaîne rule »

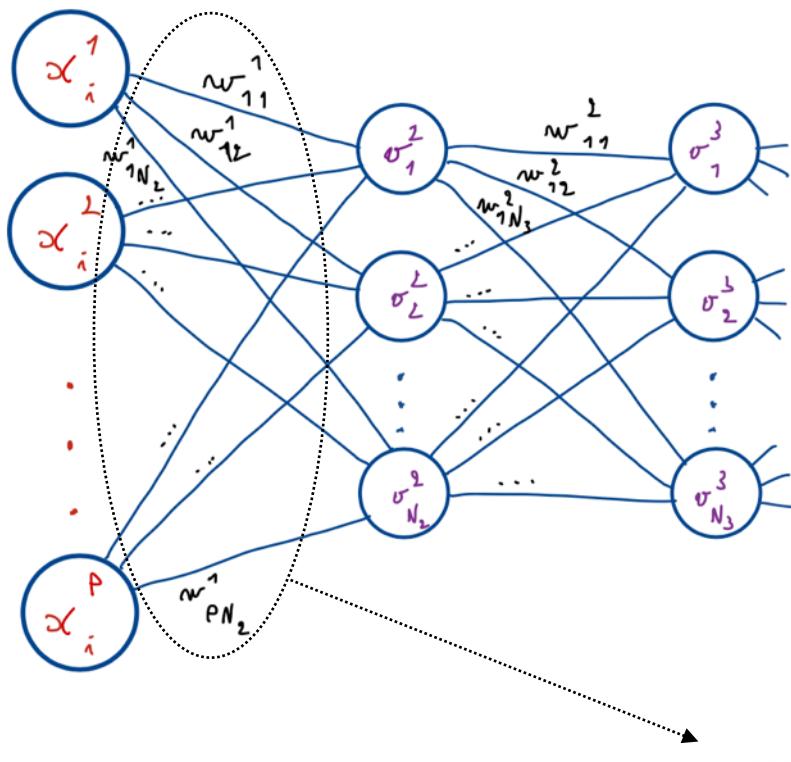
$$\frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} = \frac{\partial}{\partial s_k^{l+1}} \varphi(s_k^{l+1}) \quad [\text{connu}]$$

$l = 2$

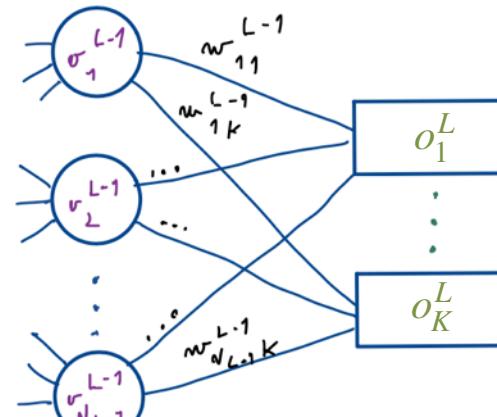
$$\begin{aligned} \frac{\partial s_k^{l+1}}{\partial w_{pk}^l} &= \frac{\partial}{\partial w_{pk}^l} \sum_{p' \in \{1, \dots, N_l\}} w_{p'k}^l o_{p'}^l \\ &= \frac{\partial}{\partial w_{pk}^l} w_{pk}^l o_p^l \\ &= o_p^l \end{aligned}$$

$l = 2$

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.3 : Gradient de l'erreur



...



y_i^1

y_i^K

$$\frac{\partial R_\Theta^i}{\partial w_{pk}^l} = \frac{\partial R_\Theta^i}{\partial o_k^{l+1}} \frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} \frac{\partial s_k^{l+1}}{\partial w_{pk}^l}$$

$l = 1$

$$\frac{\partial o_k^{l+1}}{\partial s_k^{l+1}} = \frac{\partial}{\partial s_k^{l+1}} \varphi(s_k^{l+1}) \text{ [connu]}$$

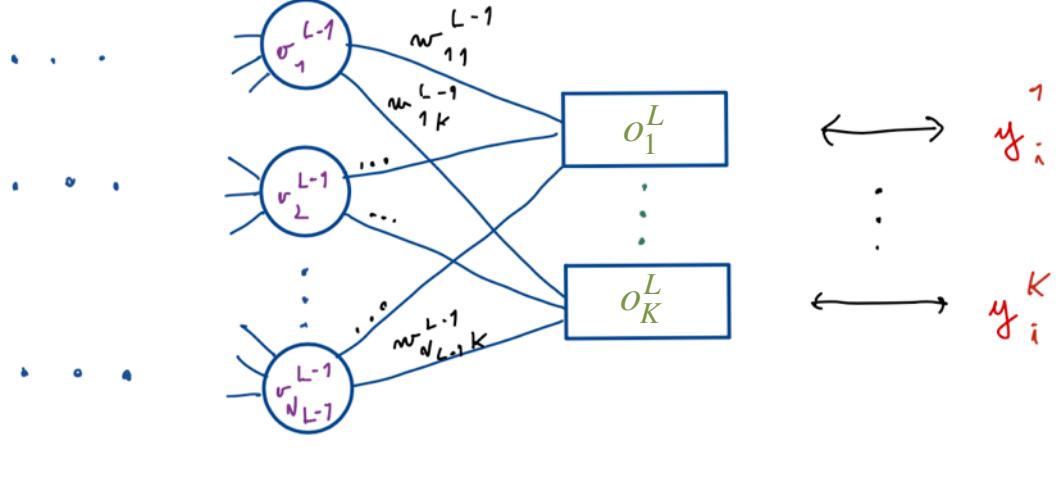
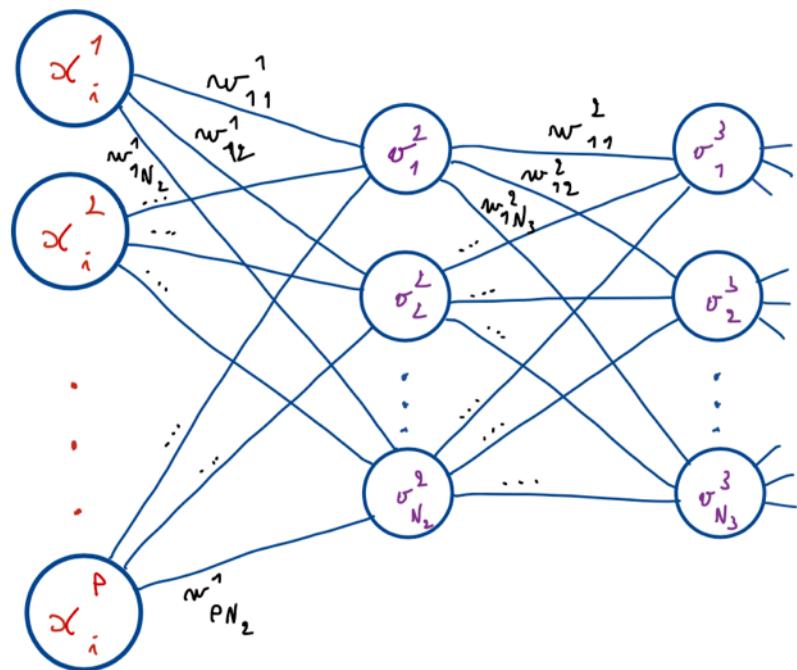
$l = 1$

$$\begin{aligned} \frac{\partial s_k^{l+1}}{\partial w_{pk}^l} &= \frac{\partial}{\partial w_{pk}^l} \sum_{p' \in \{1, \dots, N_l\}} w_{p'k}^l o_{p'}^l \\ &= \frac{\partial}{\partial w_{pk}^l} w_{pk}^l o_p^l \\ &= o_p^l \end{aligned}$$

$l = 1$

Usage de la « chaîne rule »

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.3 : Gradient de l'erreur

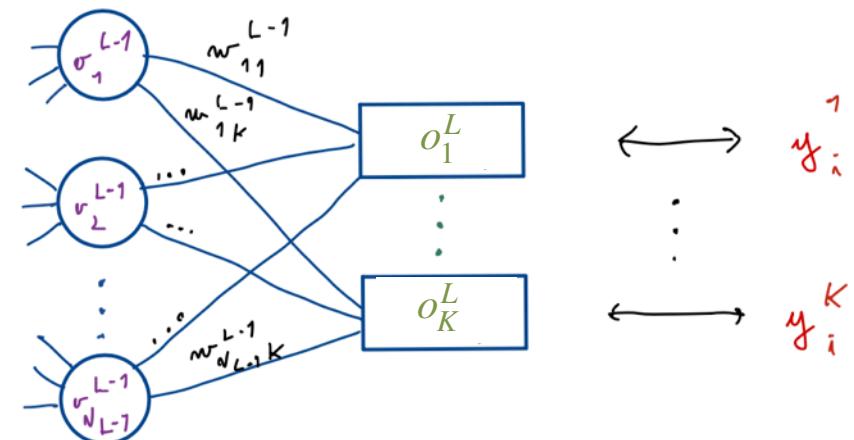
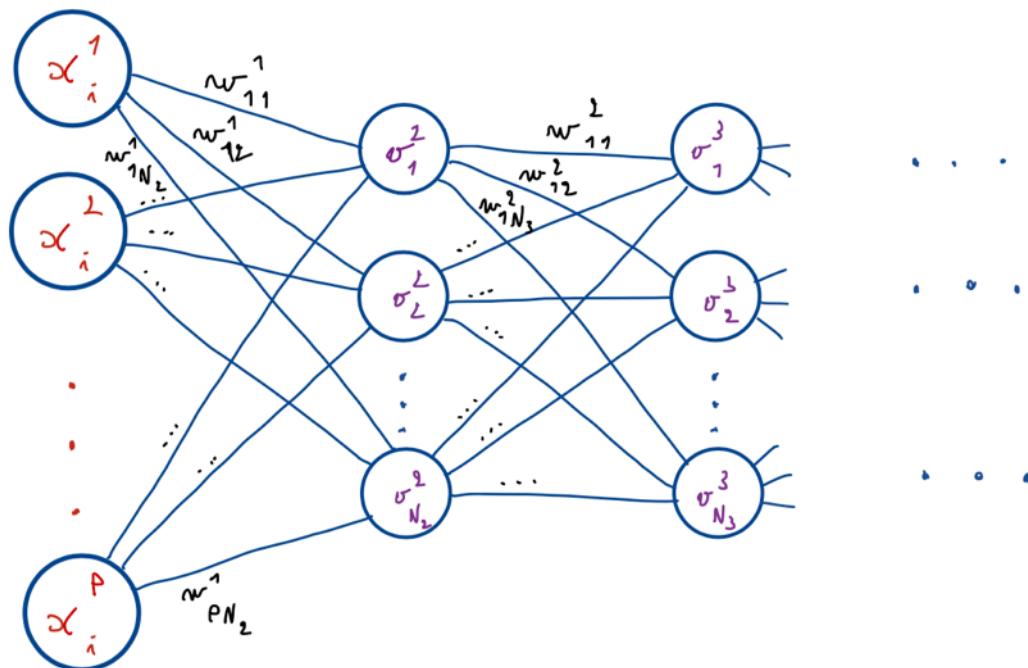


BINGO !!!

On a calculé le gradient pour une observation i :

$$\nabla R_\Theta^i ((x_i, y_i)_{i=1, \dots, n}) = \left(\frac{\partial R_\Theta^i(\dots)}{\partial w_{1,1}^1}, \frac{\partial R_\Theta^i(\dots)}{\partial w_{1,2}^1}, \dots, \frac{\partial R_\Theta^i(\dots)}{\partial w_{N_{L-1},K}^L} \right)^T$$

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.4 : Descente de gradient



Apprentissage des $\hat{\Theta} = \arg \min_{\Theta} \frac{1}{n} \sum_{i=1}^n \text{loss}(h_{\Theta}(x_i), y_i)$ par descente de gradient :

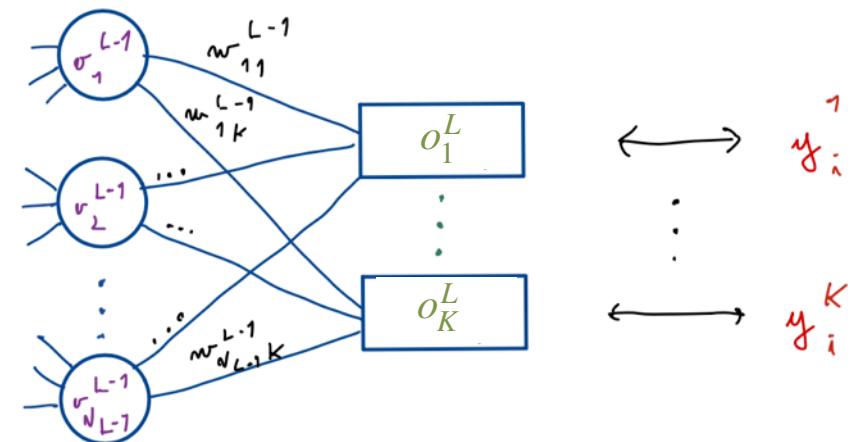
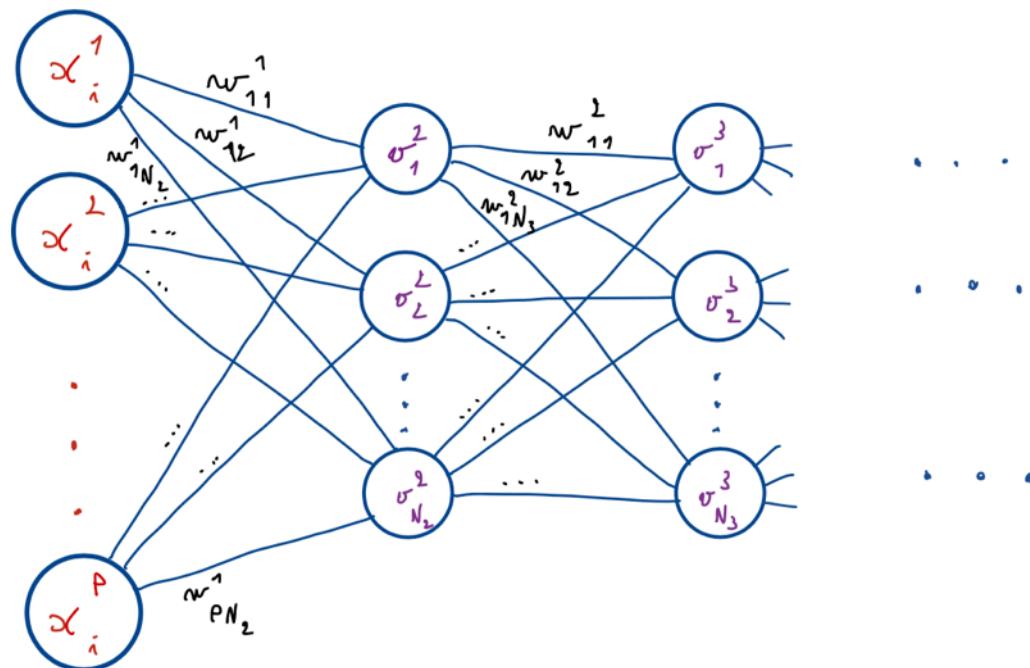
Risque empirique R_{Θ}

Tant que non convergence :

- $\forall i \in \{1, \dots, n\}$: Calcule les sorties du réseau de neurones $\hat{y}_i = h_{\Theta}(x_i)$
- $\forall i \in \{1, \dots, n\}$: Calcule $\text{loss}(h_{\Theta}(x_i), y_i)$ et $\nabla_{\Theta} \text{loss}(h_{\Theta}(x_i), y_i)$ par rétro-propagation
- Calcule $\nabla_{\Theta} R_{\Theta} = \frac{1}{n} \sum_{i=1}^n \nabla_{\Theta} \text{loss}(h_{\Theta}(x_i), y_i)$
- $\Theta \leftarrow \Theta - \lambda \nabla R_{\Theta}$

Remarque :
Utilise des informations de la phase forward

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.4 : Descente de gradient



Apprentissage des $\hat{\Theta} = \arg \min_{\Theta} \frac{1}{n} \sum_{i=1}^n \text{loss}(h_{\Theta}(x_i), y_i)$ par descente de gradient **stochastique** :

Risque empirique R_{Θ}

Tant que non convergence :

- $\forall i \in B$: Calcule les sorties du réseau de neurones $\hat{y}_i = h_{\Theta}(x_i)$
- $\forall i \in B$: Calcule $\text{loss}(h_{\Theta}(x_i), y_i)$ et $\nabla_{\Theta} \text{loss}(h_{\Theta}(x_i), y_i)$ par rétro-propagation
- Calcule $\nabla_{\Theta} R_{\Theta} \approx \frac{1}{\#B} \sum_{i \in B} \nabla_{\Theta} \text{loss}(h_{\Theta}(x_i), y_i)$
- $\Theta \leftarrow \Theta - \lambda \nabla R_{\Theta}$

Observations d'un mini-batch B :

- Changent à chaque itération
- Sans remise durant un epoch
- Plusieurs epochs sont courants
- Plus rapide
- Moins gourmand en mémoire

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.5 : ... et dans d'autres cas ?

Architecture de CNN classique (VGG) :

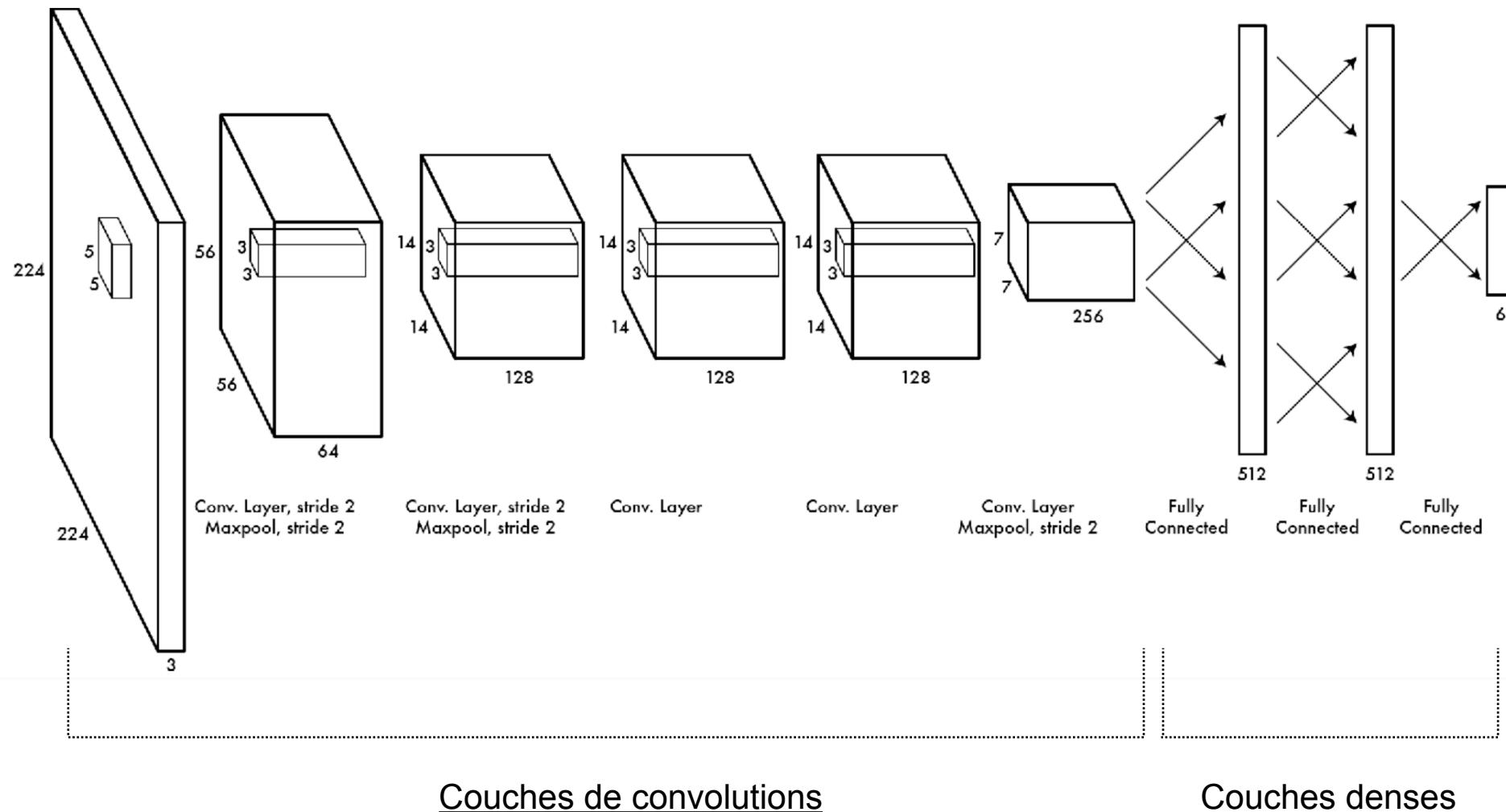
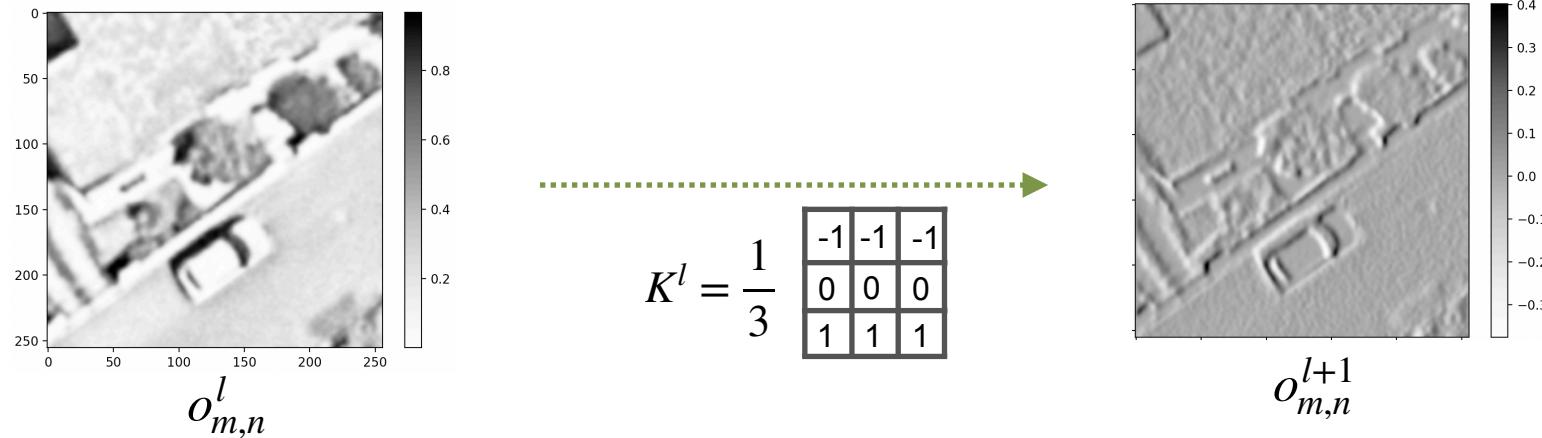


Image du réseau : [Redmon and Angelova, IEEE ICRA 2015]

Partie 3 : Apprentissage des paramètres d'un réseau de neurones → 3.5 : ... et dans d'autres cas ?

Couches de convolution : Apprentissage des poids de filtres de convolution optimaux.



Modèle forward :

$$\begin{cases} s_{f,g}^{l+1} = \sum_{v=-V}^V \sum_{w=-W}^W K_{v,w}^l o_{f+v,g+w}^l \\ o_{f,g}^{l+1} = \varphi(s_{f,g}^{l+1}) \end{cases}$$

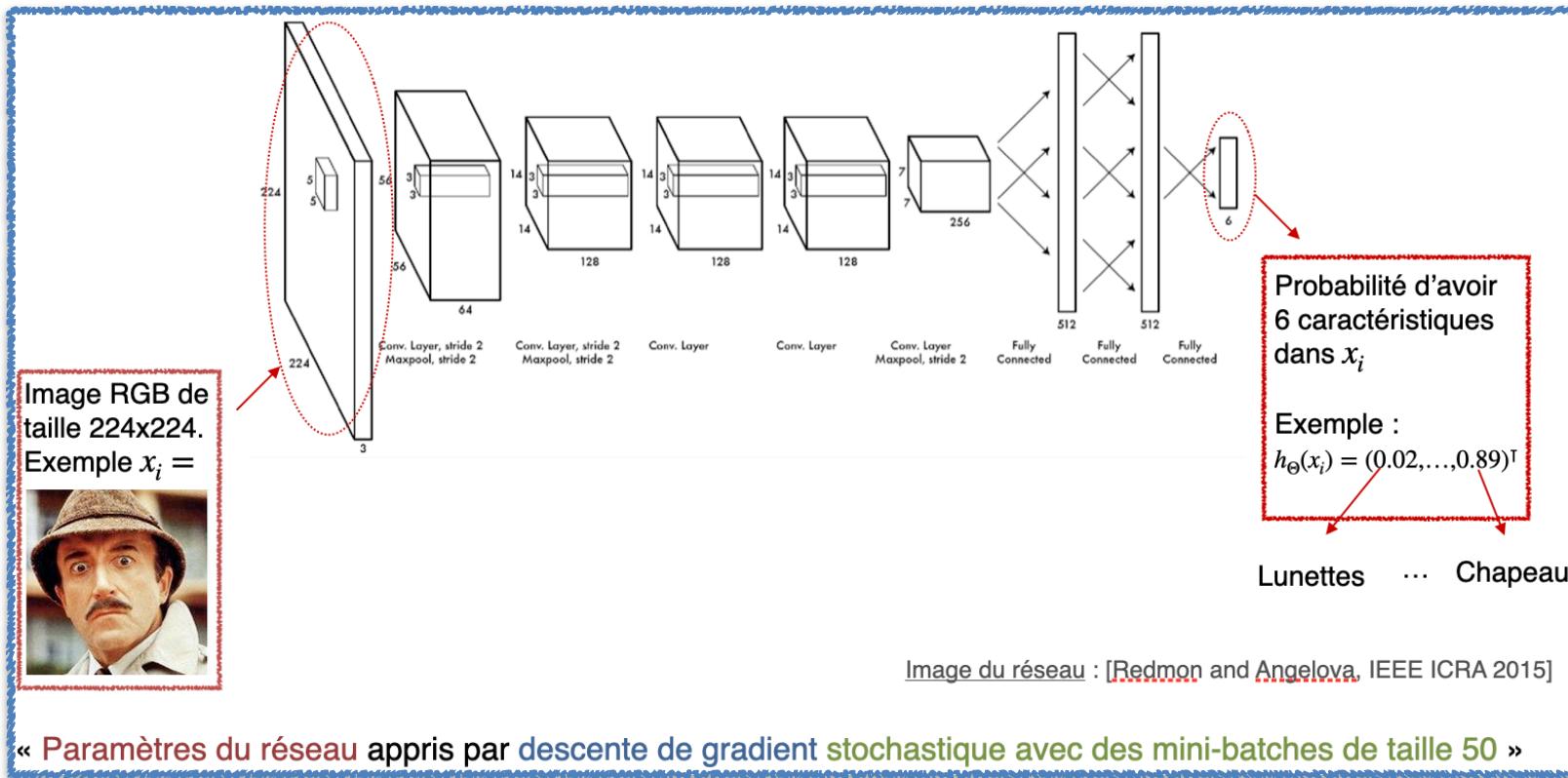
Equations backward :

$$\frac{\partial R_\theta^i}{\partial o_{m,n}^l} = \sum_{v=-V}^V \sum_{w=-W}^W \frac{\partial R_\theta^i}{\partial o_{m+v,n+w}^{l+1}} \frac{\partial o_{m+v,n+w}^{l+1}}{\partial s_{m+v,n+w}^{l+1}} \frac{\partial s_{m+v,n+w}^{l+1}}{\partial o_{m,n}^l} = \frac{\partial R_\theta^i}{\partial o_{m+v,n+w}^{l+1}} \varphi'(s_{m+v,n+w}^{l+1}) K_{-v,-w}^l$$

$$\frac{\partial R_\theta^i}{\partial K_{v,w}^l} = \sum_{m=1}^M \sum_{n=1}^N \frac{\partial R_\theta^i}{\partial o_{m,n}^{l+1}} \frac{\partial o_{m,n}^{l+1}}{\partial s_{m,n}^{l+1}} \frac{\partial s_{m,n}^{l+1}}{\partial K_{v,w}^l} = \frac{\partial R_\theta^i}{\partial o_{m,n}^{l+1}} \varphi'(s_{m,n}^{l+1}) o_{m+v,n+w}^l$$

Remarque : Se parallélise très bien sur un GPU

Conclusion



J'espère que vous y voyez plus clair !

Des questions ?

Jeudi 18 janvier 2024, 14h

