

# Réseaux de neurones profonds

Guillaume Bourmaud

# PLAN

I. Réseaux de neurones profonds

II. Modèles de fondation

III. « Gradient checkpointing »

IV. Apprentissage multi-GPU

# I) Réseaux de neurones profonds

# Résumé des ingrédients du « Deep Learning »

1) Grande base de données étiquetées

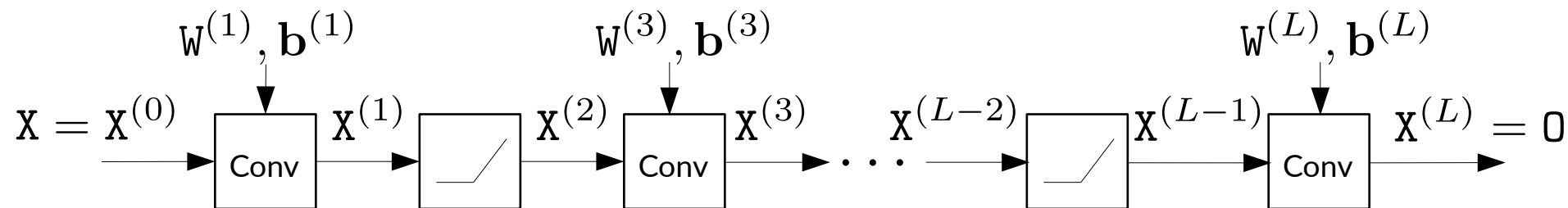
2) « **Bonne** » architecture de réseau de neurones profond

- ▶ « Perceptron » multicouche, Réseau de neurones à convolution, Transformer
- ▶ Optimisation par descente de gradient stochastique (AdamW, etc.)

3) Grande capacité de calculs en parallèle (GPUs)

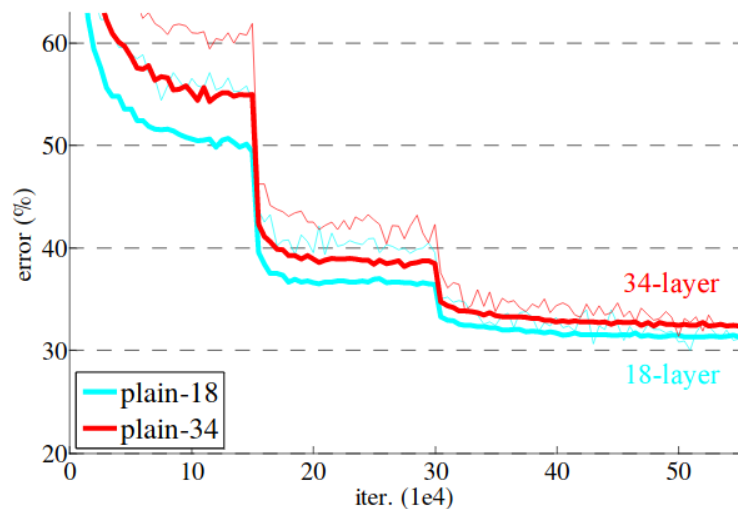
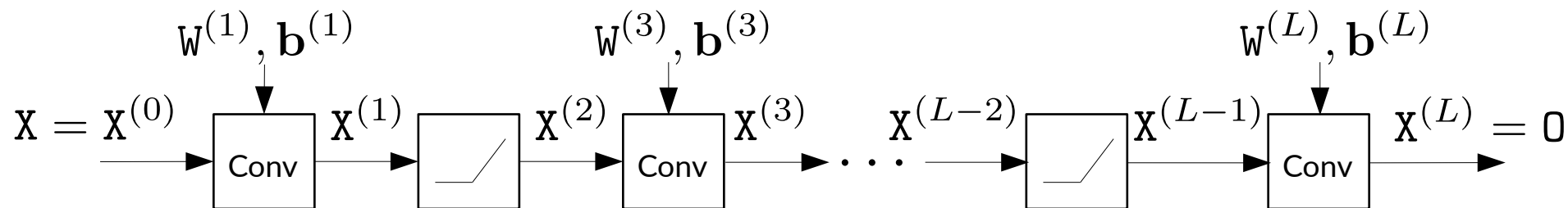
1)

## Limites du CNN « classique »



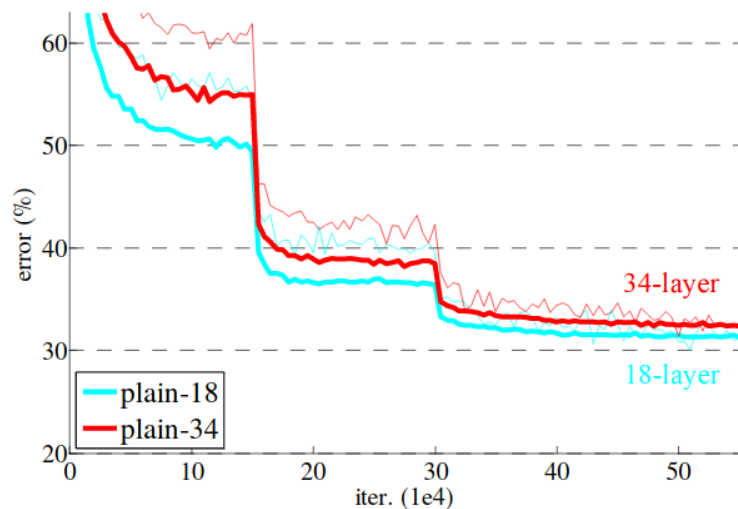
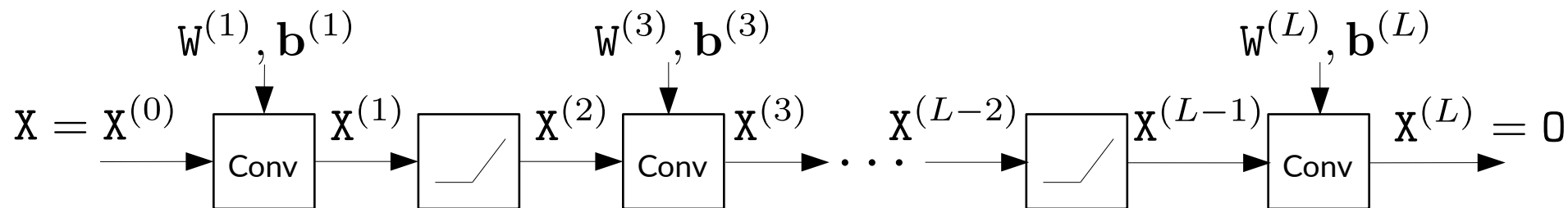
I)

## Limites du CNN « classique »



I)

## Limites du CNN « classique »



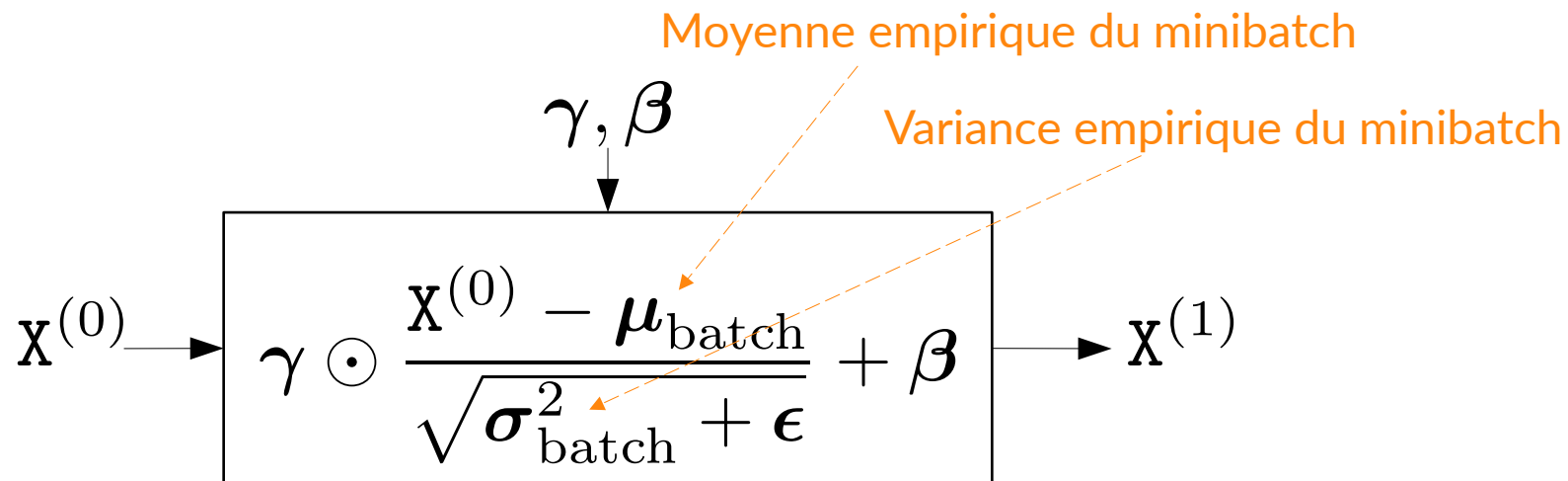
Ingrédient limitant les performances



Architecture du CNN

I)

## Ingrédient 1 : Couche de “Batch Normalization”



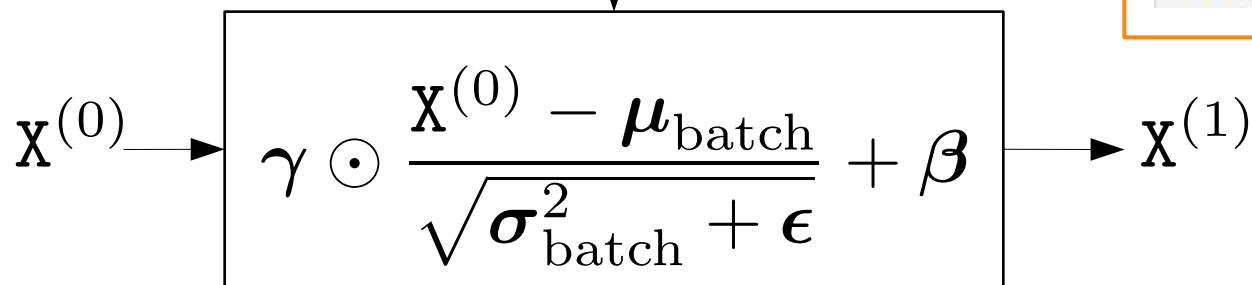
I)

## Ingrédient 1 : Couche de “Batch Normalization”

$\gamma, \beta$

En PyTorch

```
model.train()
```



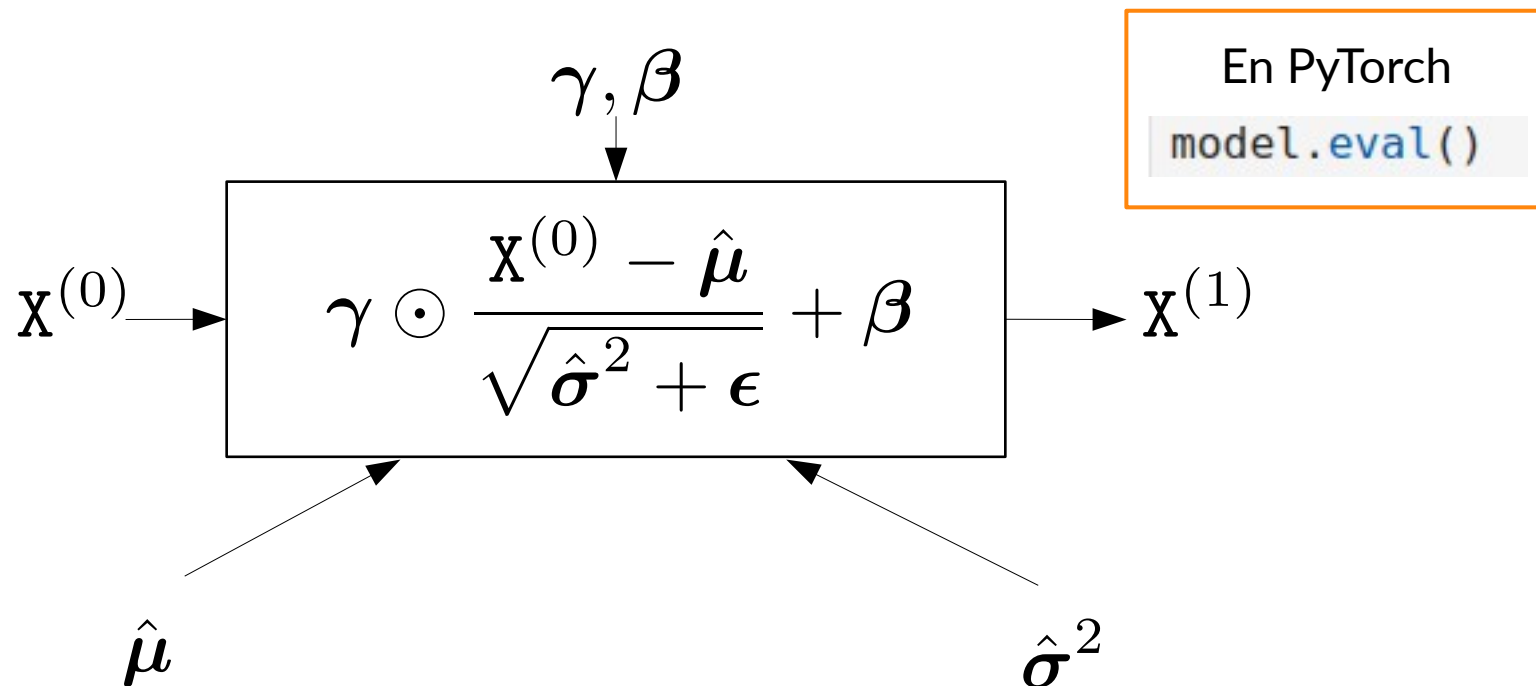
$$\hat{\mu} = 0.1 * \mu_{\text{batch}} + 0.9 * \hat{\mu}$$

$$\hat{\sigma}^2 = 0.1 * \sigma_{\text{batch}}^2 + 0.9 * \hat{\sigma}^2$$

Fonctionnement de la couche BN à l'entraînement.

I)

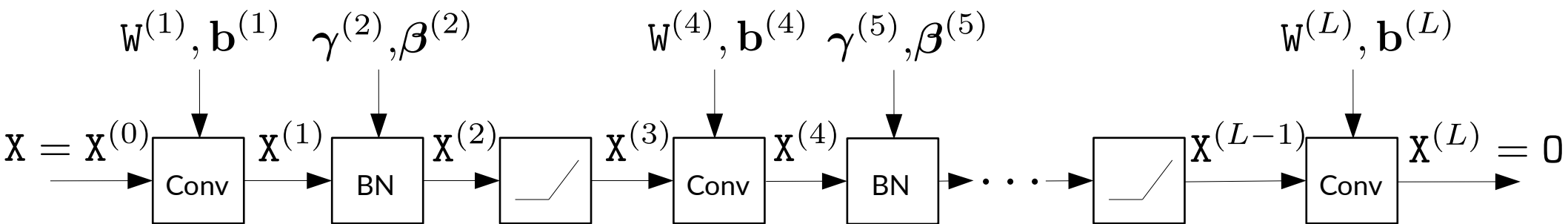
## Ingrédient 1 : Couche de “Batch Normalization”



Fonctionnement de la couche BN à l'évaluation.

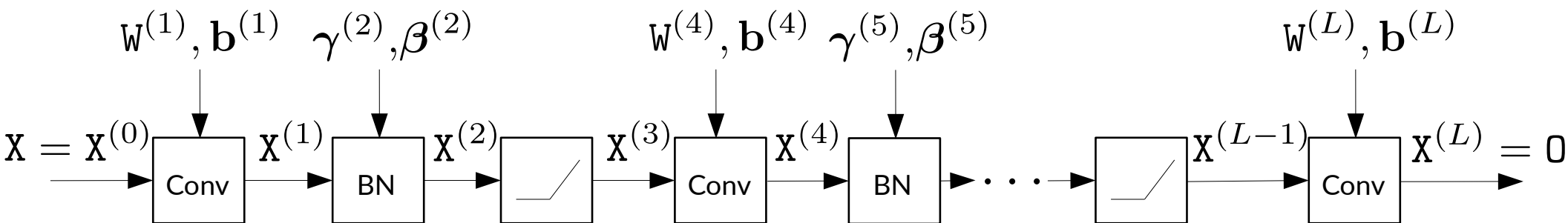
1)

# CNN + BN



1)

## CNN + BN



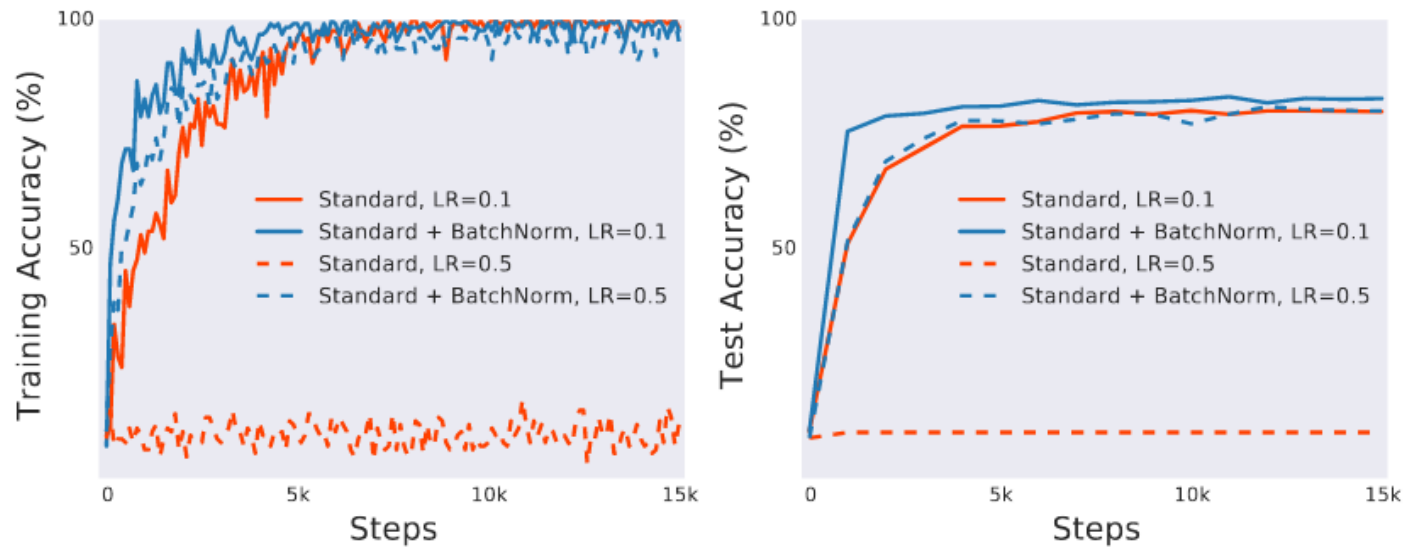
Remarque

$$\text{BN}(\text{conv}(\mathbf{x}) + b) = \gamma \frac{\text{conv}(\mathbf{x}) - \mu(\text{conv}(\mathbf{x}))}{\sigma(\text{conv}(\mathbf{x}))} + \beta$$

Le biais  $b$  de la couche de conv est « absorbé » (en mode « train ») par la BN et ne sert à rien.

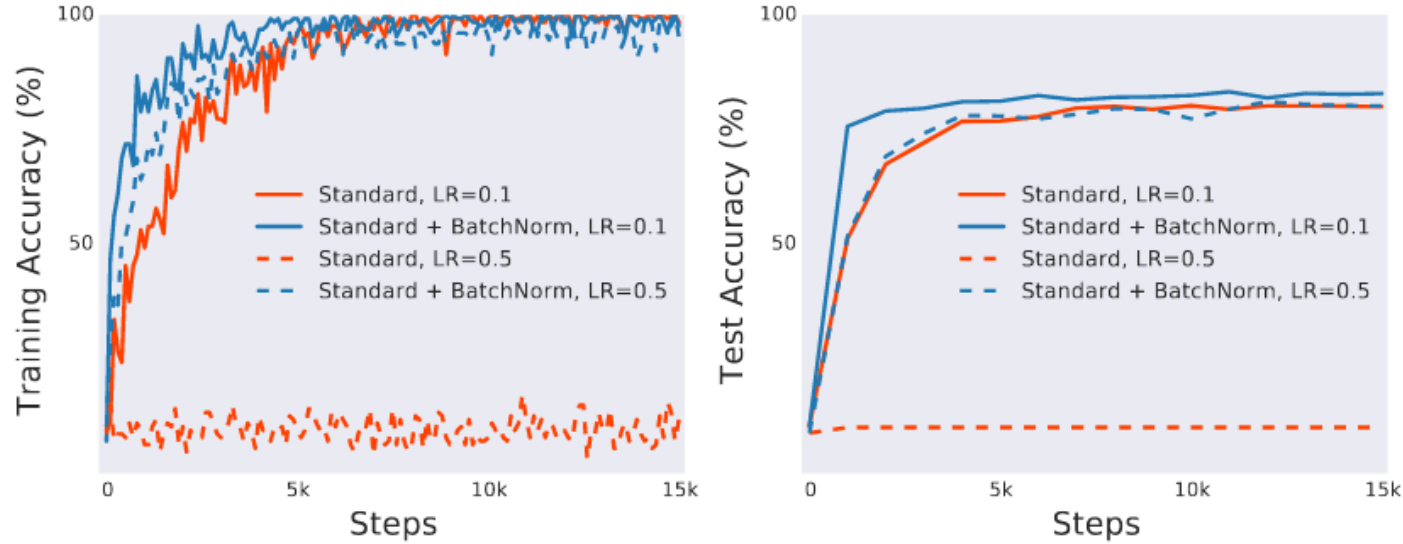
I)

## CNN + BN (suite)



I)

## CNN + BN (suite)



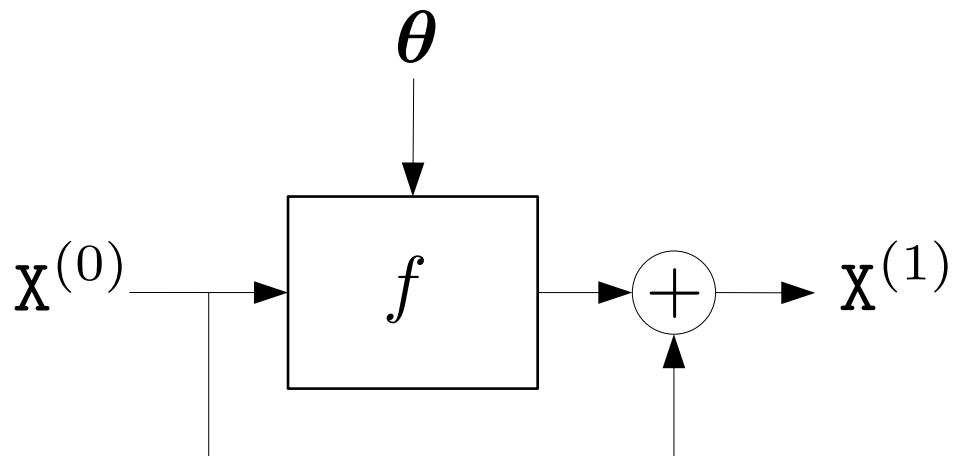
Rend le problème d'optimisation plus « lisse » :

→ Initialisation des paramètres moins critique

→ Possibilité d'utilisation d'un plus grand pas d'apprentissage → accélération de l'entraînement

I)

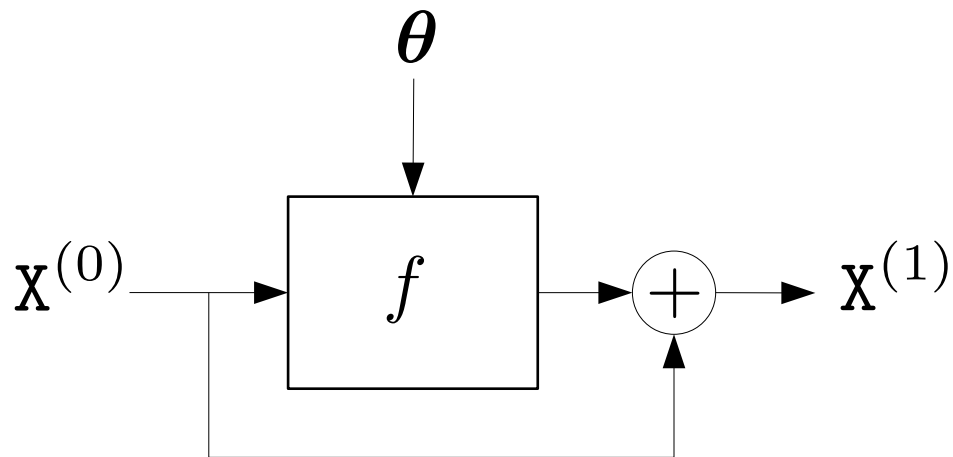
## Ingrédient 2 : Connexion résiduelle



$$x^{(1)} = x^{(0)} + f(x^{(0)}; \theta)$$

I)

## Ingrédient 2 : Connexion résiduelle



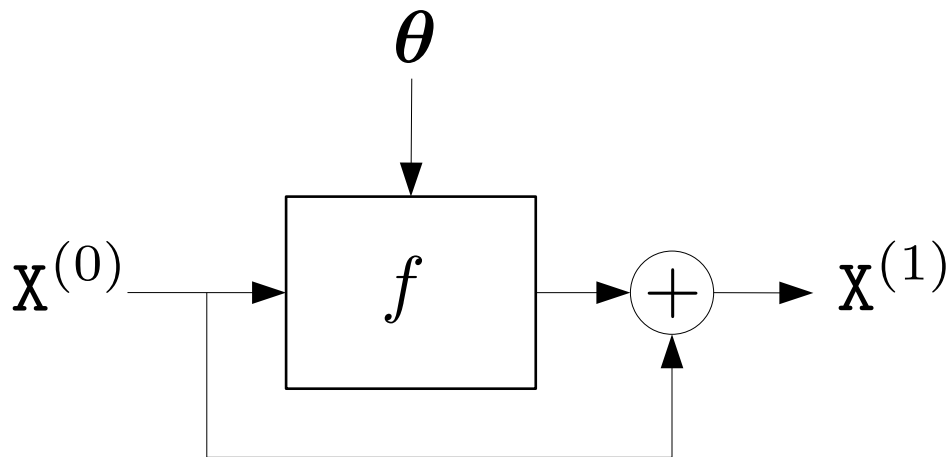
$$x^{(1)} = x^{(0)} + f(x^{(0)}; \theta)$$

Rend la fonction plus « linéaire » :

→ Réduit sa « capacité » → augmentation du nombre de couches pour un même résultat

I)

## Ingrédient 2 : Connexion résiduelle



$$X^{(1)} = X^{(0)} + f(X^{(0)}; \theta)$$

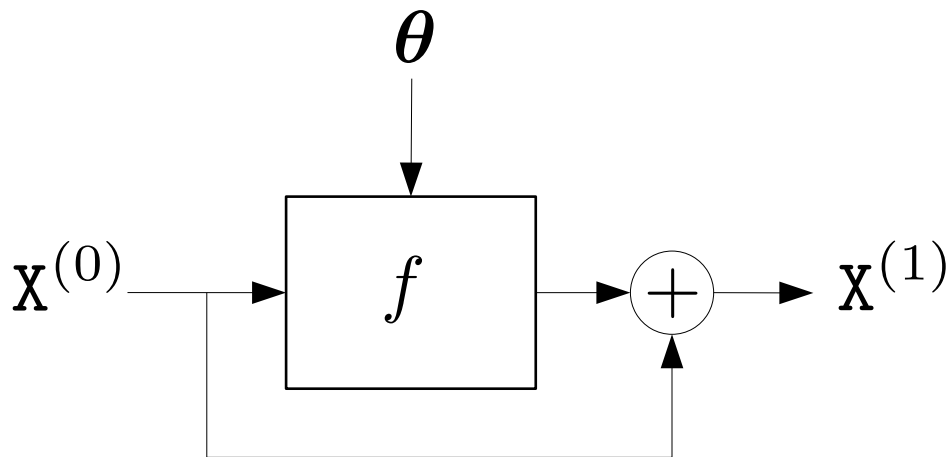
$$\frac{\partial X^{(1)}}{\partial X^{(0)}} = \mathbf{I} + \frac{\partial f(X^{(0)}; \theta)}{\partial X^{(0)}}$$

Rend la fonction plus « linéaire » :

→ Réduit sa « capacité » → augmentation du nombre de couches pour un même résultat

I)

## Ingrédient 2 : Connexion résiduelle



Rend la fonction plus « linéaire » :

→ Réduit sa « capacité » → augmentation du nombre de couches pour un même résultat

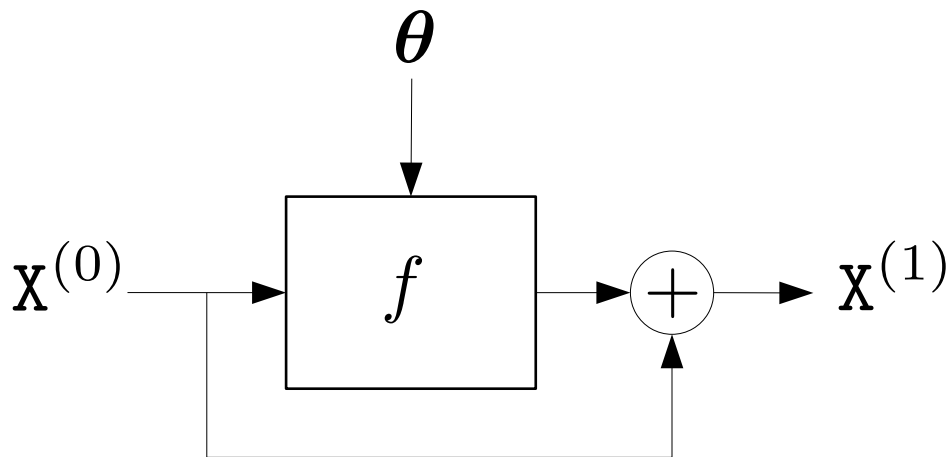
$$X^{(1)} = X^{(0)} + f(X^{(0)}; \theta)$$

$$\frac{\partial X^{(1)}}{\partial X^{(0)}} = I + \frac{\partial f(X^{(0)}; \theta)}{\partial X^{(0)}}$$

Même si ce gradient « s'évanouit » (vaut presque zéro), il reste l'identité grâce à la connexion résiduelle

I)

## Ingrédient 2 : Connexion résiduelle



$$X^{(1)} = X^{(0)} + f(X^{(0)}; \theta)$$

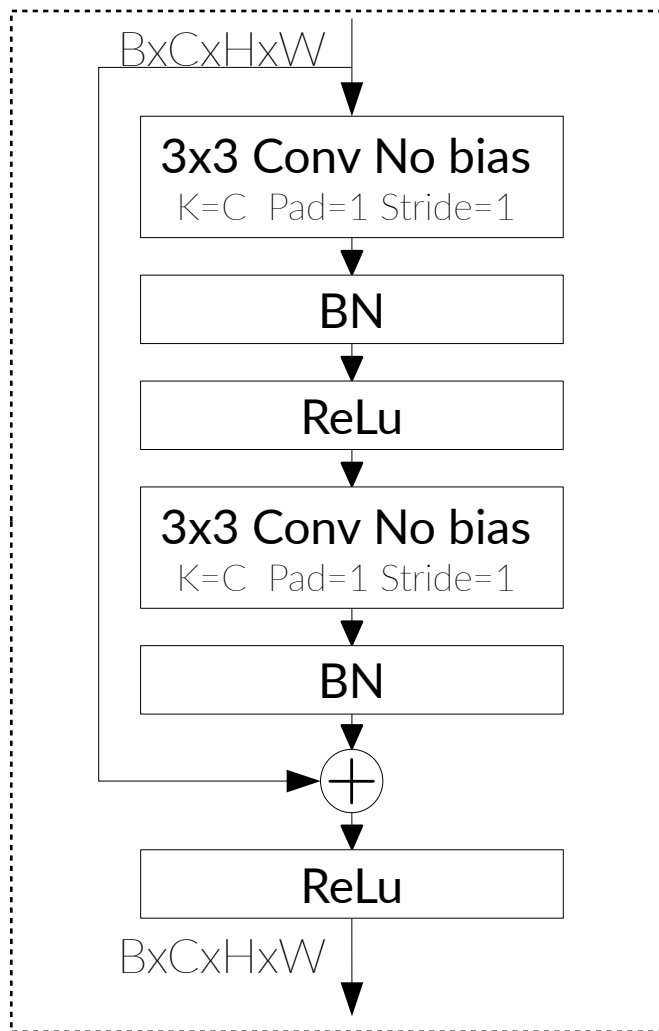
$$\frac{\partial X^{(1)}}{\partial X^{(0)}} = \mathbf{I} + \frac{\partial f(X^{(0)}; \theta)}{\partial X^{(0)}}$$

Rend la fonction plus « linéaire » :

→ Réduit sa « capacité » → augmentation du nombre de couches pour un même résultat

→ Facilite la propagation du gradient → plus de couches conduit à de meilleurs résultats (en théorie)

1)



## ResBlock A

Transforme le tenseur d'entrée

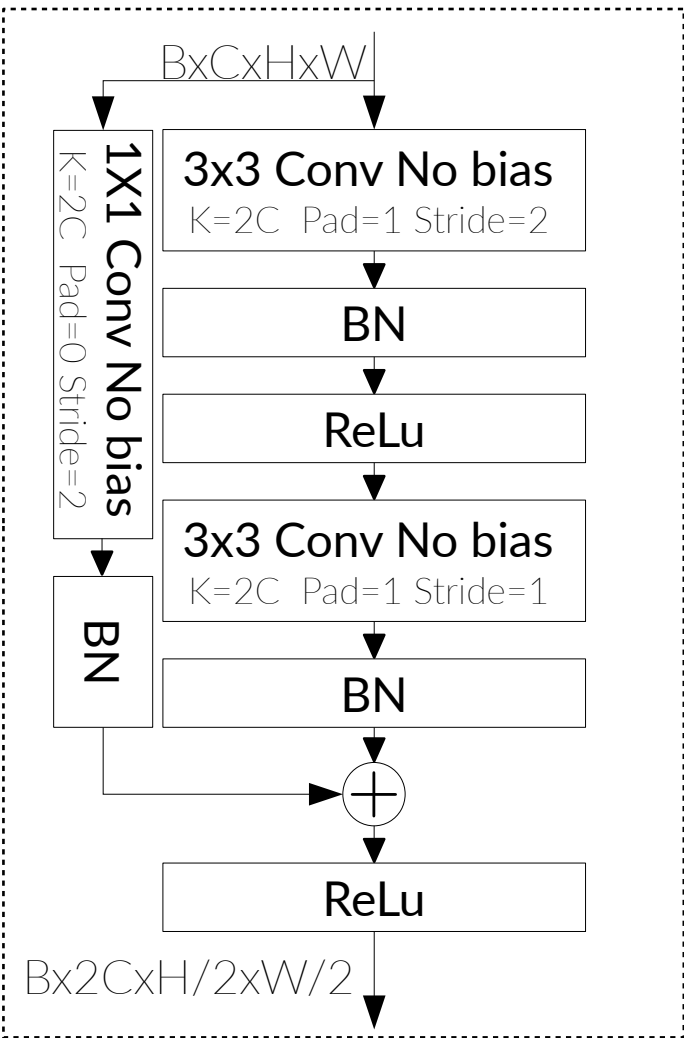
- en préservant la résolution
- en préservant le nombre de canaux

I)

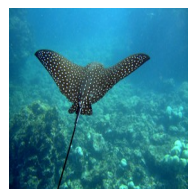
# ResBlock B

Transforme le tenseur d'entrée

- en divisant la résolution par 2
- en augmentant le nombre de canaux par 2



I)

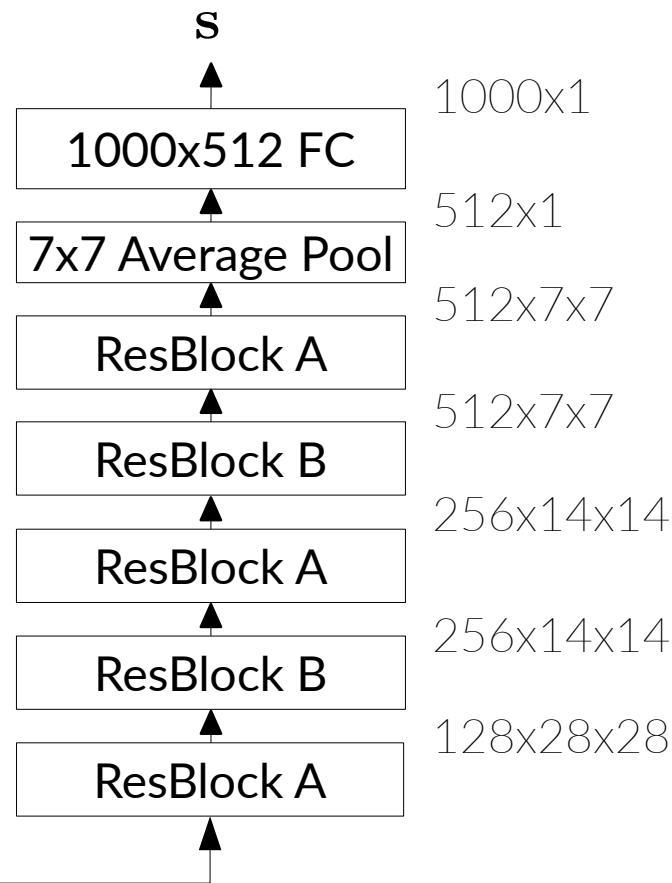
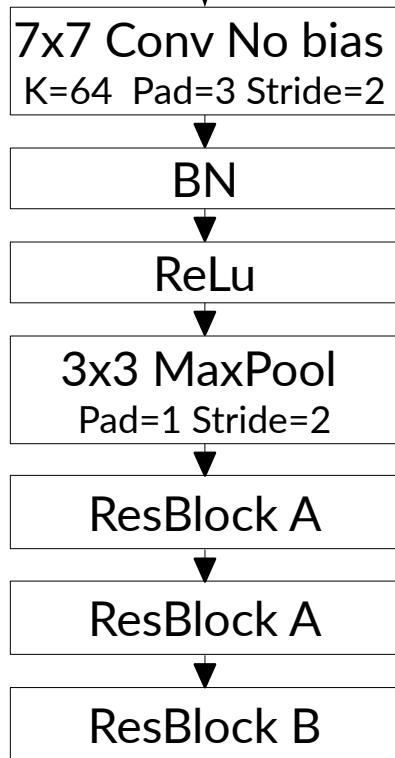


# ResNet 18

$$\arg \max(s) = 748 \doteq \text{"raie"}$$

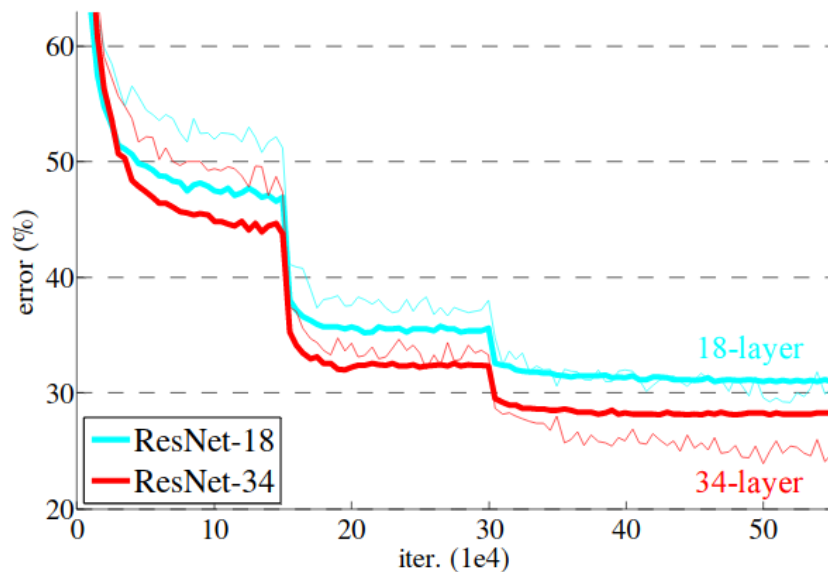
Passage  
en grande  
dimension

3x224x224  
↓  
64x112x112  
64x112x112  
64x112x112  
64x56x56  
64x56x56  
64x56x56  
128x28x28



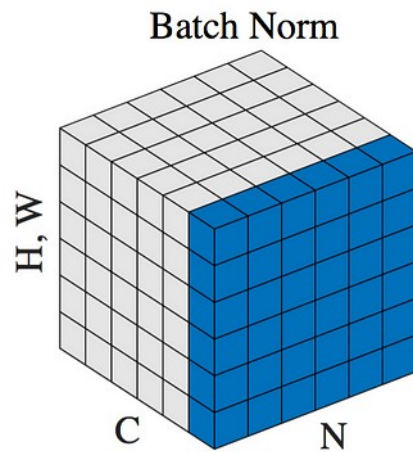
I)

# ResNet 18 < ResNet 34



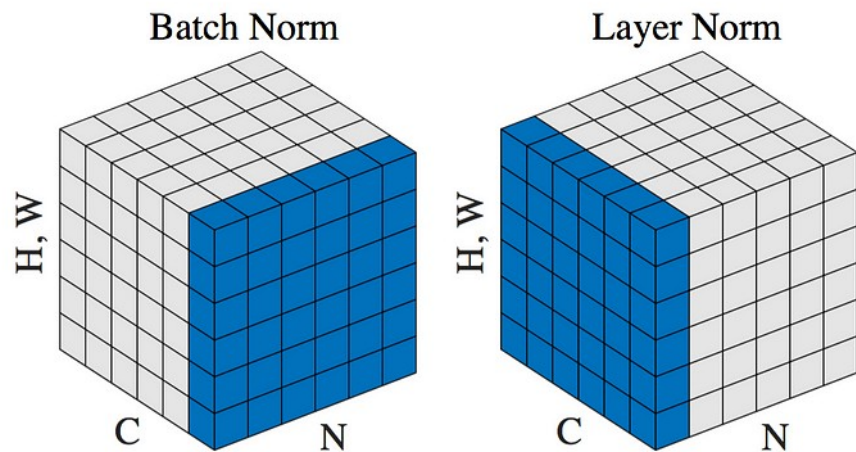
I)

## Différentes couches de normalisation



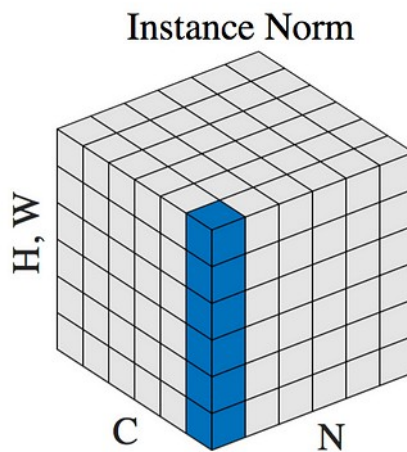
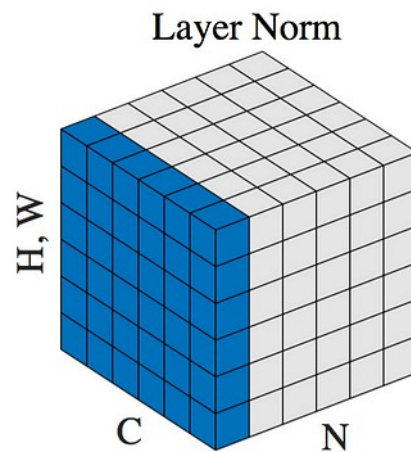
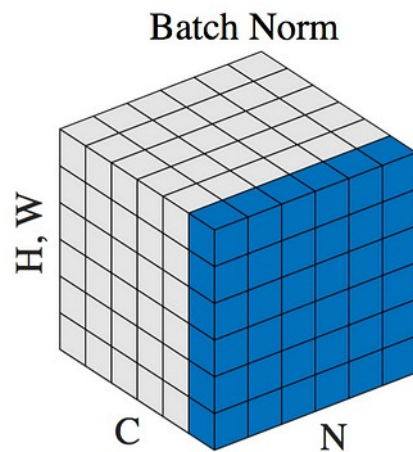
I)

## Différentes couches de normalisation



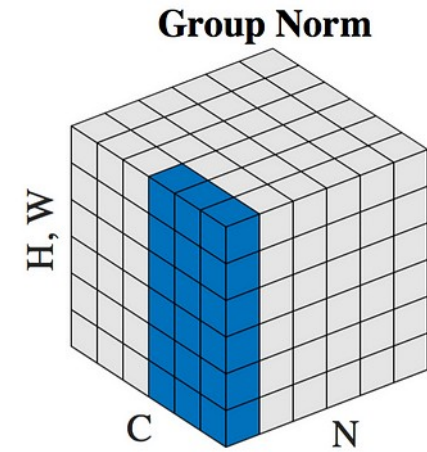
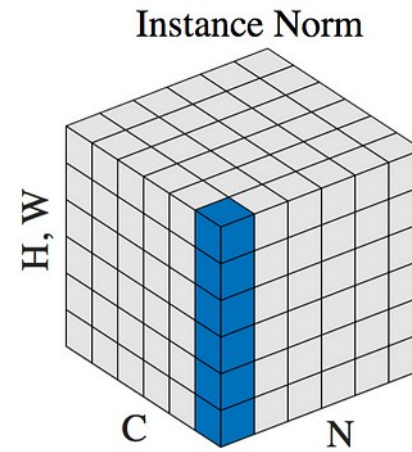
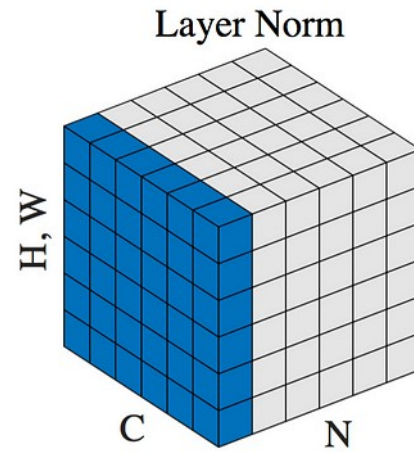
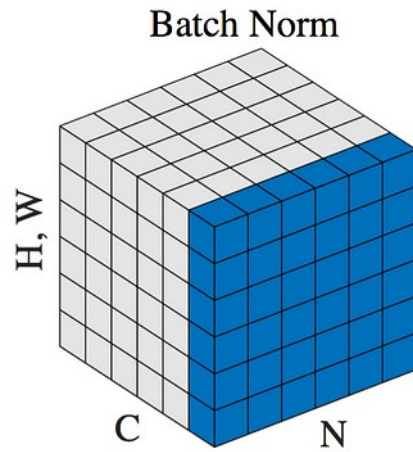
I)

## Différentes couches de normalisation



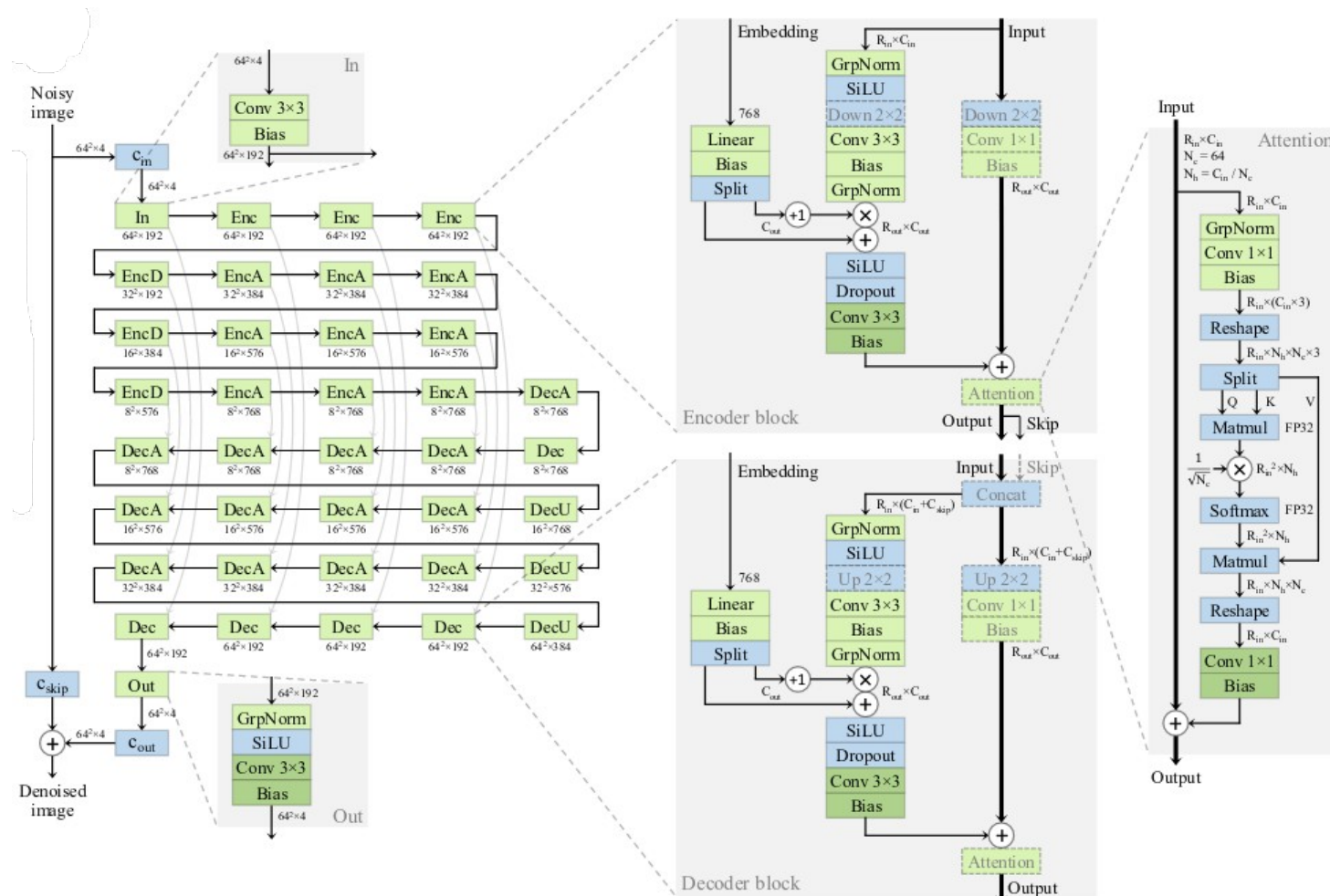
I)

## Différentes couches de normalisation



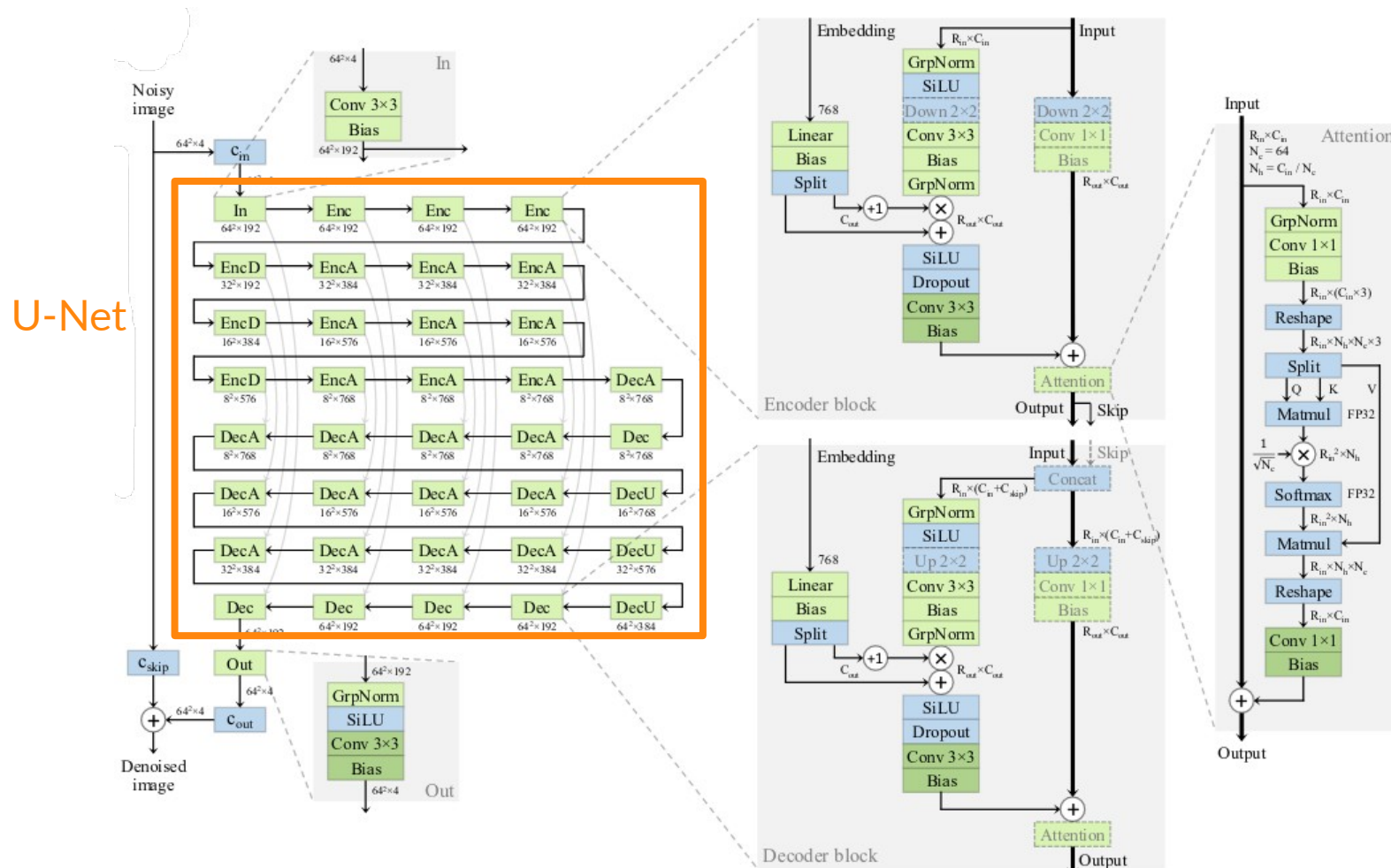
1)

# En 2024 : U-Net + ResBlock toujours là !



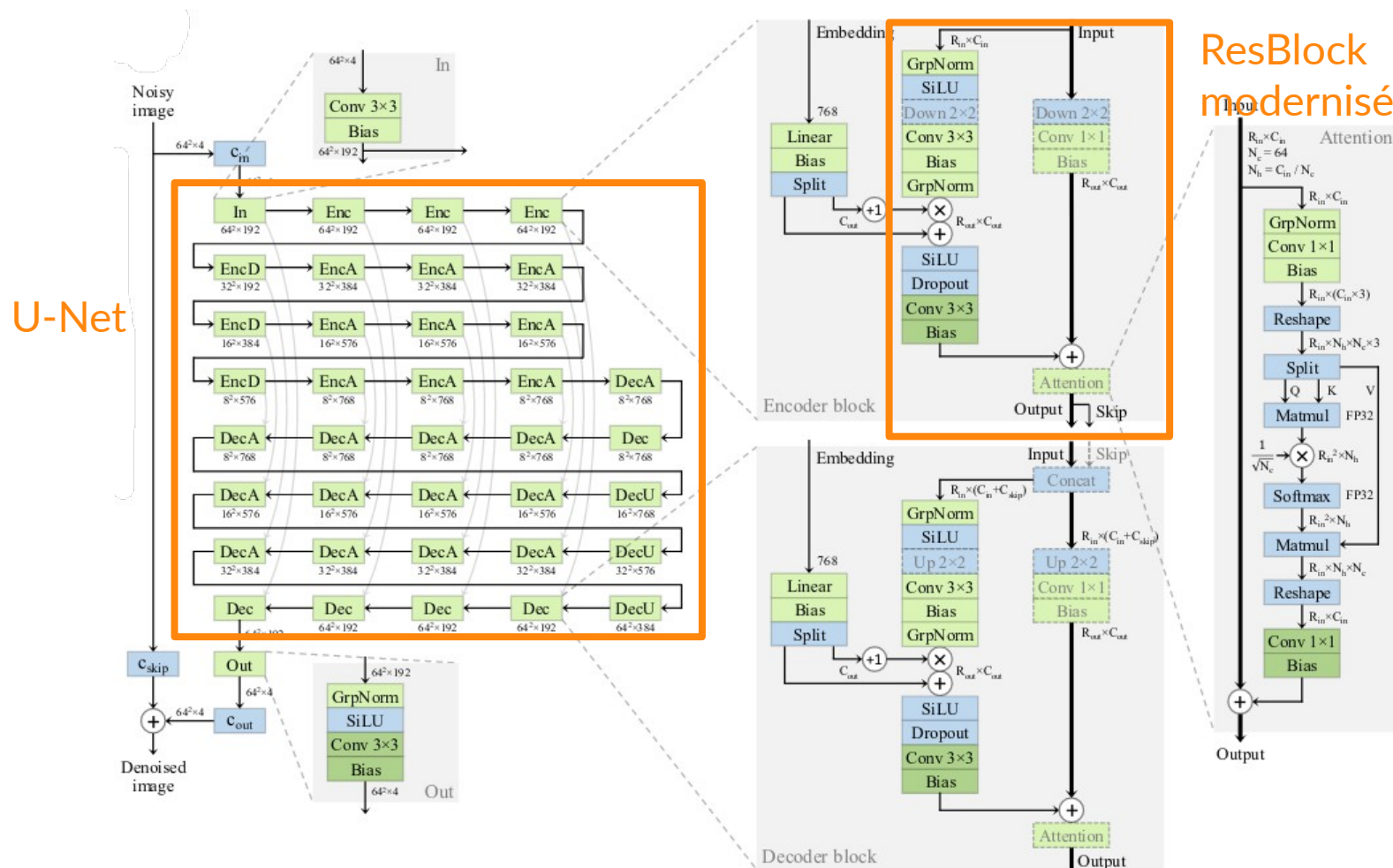
1)

# En 2024 : U-Net + ResBlock toujours là !



1)

# En 2024 : U-Net + ResBlock toujours là !



## II) Modèles de fondation

## Résumé des ingrédients du « Deep Learning »

- 1) **Grande** base de données étiquetées
- 2) « Bonne » architecture de réseau de neurones profond
  - ▶ « Perceptron » multicouche, Réseau de neurones à convolution, Transformer
  - ▶ Optimisation par descente de gradient stochastique (AdamW, etc.)
- 3) Grande capacité de calculs en parallèle (GPUs)

II)

# Comment faire avec une petite base de données étiquetées ?

Exemple : détection du frelon asiatique



Présence (1042 images)



Absence (1844 images)

**Ingrédient limitant les performances**



**Taille de la base de données étiquetées**

II)

# Comment faire avec une petite base de données étiquetées ?

Exemple : détection du frelon asiatique



Présence (1042 images)



Absence (1844 images)

**Ingrédient limitant les performances**



**Taille de la base de données étiquetées**

**Solution**

**1) Récupérer un modèle de fondation**

**2) Spécialiser ce modèle de fondation sur sa petite base de données étiquetées**

II)

## Modèle de fondation « historique »

Historiquement (~ à partir de 2012), un modèle de fondation (le terme « foundation model » n'est introduit qu'en 2021) est :

## Modèle de fondation « historique »

Historiquement (~ à partir de 2012), un modèle de fondation (le terme « foundation model » n'est introduit qu'en 2021) est :

- un CNN (e.g ResNet)

## Modèle de fondation « historique »

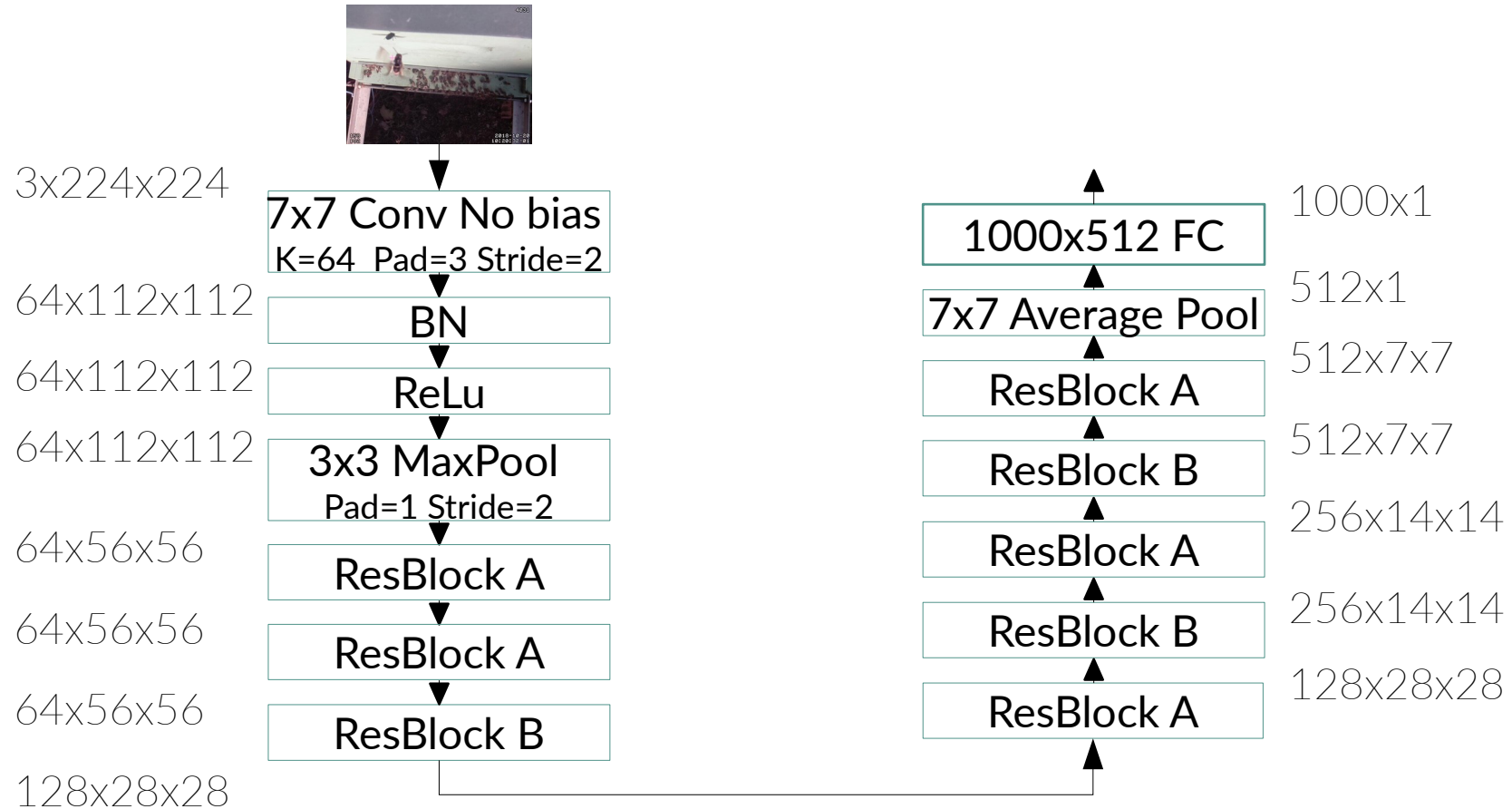
Historiquement (~ à partir de 2012), un modèle de fondation (le terme « foundation model » n'est introduit qu'en 2021) est :

- un CNN (e.g ResNet)
- qui est **pré-entraîné** = entraîné sur ImageNet1k

Rappel : ImageNet1k = 1.2M d'images étiquetées sur 1000 classes qui représentent une grande diversité d'images **issues de notre monde**



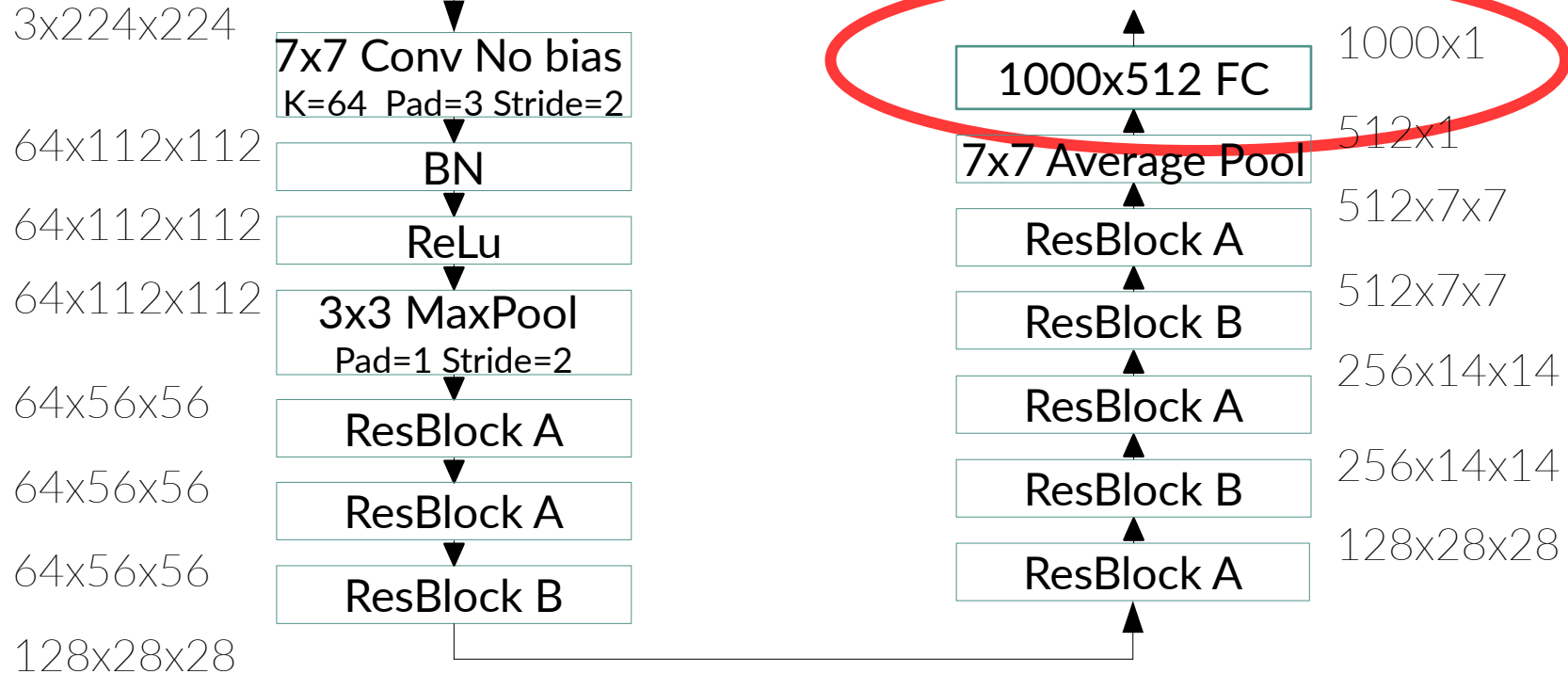
## II) Exemple de spécialisation (« fine-tuning ») d'un ResNet 18 pré-entraîné sur ImageNet1k



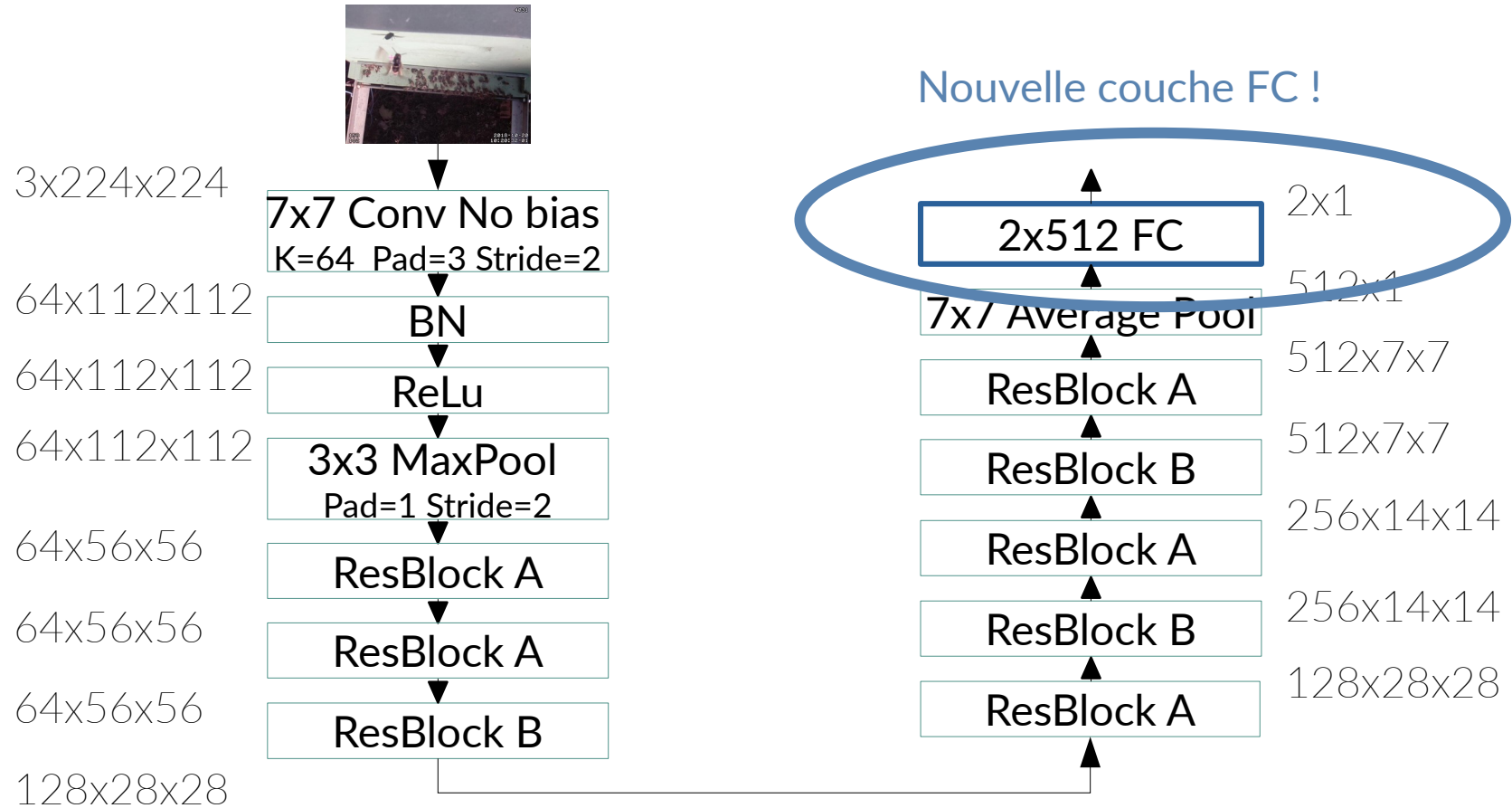
## II) Exemple de spécialisation (« fine-tuning ») d'un ResNet 18 pré-entraîné sur ImageNet1k



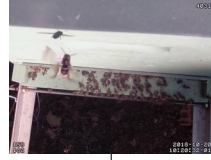
Problème : on veut prédire 2 scores (présence de frélon, absence de frelon), pas 1000 scores...



## II) Exemple de spécialisation (« fine-tuning ») d'un ResNet 18 pré-entraîné sur ImageNet1k



## II) Exemple de spécialisation (« fine-tuning ») d'un ResNet 18 pré-entraîné sur ImageNet1k



$\arg \max(s) = 2 \doteq \text{''Présence''}$

3x224x224

7x7 Conv No bias  
K=64 Pad=3 Stride=2

64x112x112

BN

64x112x112

ReLu

64x112x112

3x3 MaxPool  
Pad=1 Stride=2

64x56x56

ResBlock A

64x56x56

ResBlock A

64x56x56

ResBlock B

128x28x28

2x512 FC

2x1

7x7 Average Pool

512x1

ResBlock A

512x7x7

ResBlock B

512x7x7

ResBlock A

256x14x14

ResBlock B

256x14x14

ResBlock A

128x28x28

Paramètres du ResNet18 pré-entraîné sur ImageNet

Initialisation classique (aléatoire)

## II) Exemple de spécialisation (« fine-tuning ») d'un ResNet 18 pré-entraîné sur ImageNet1k

Lors du pré-entraînement sur ImageNet1k :

- Les images sont normalisées (comme on l'a fait en TP sur MNIST)

$$X_{\text{norm}} = \frac{X - \mu_{\text{ImageNet1k}}}{\sigma_{\text{ImageNet1k}}}$$

## II) Exemple de spécialisation (« fine-tuning ») d'un ResNet 18 pré-entraîné sur ImageNet1k

Lors du pré-entraînement sur ImageNet1k :

- Les images sont normalisées (comme on l'a fait en TP sur MNIST)

$$X_{\text{norm}} = \frac{X - \mu_{\text{ImageNet1k}}}{\sigma_{\text{ImageNet1k}}}$$

- Pour utiliser le réseau, il faut appliquer cette **même** normalisation à nos images !

# II) Exemple de spécialisation (« fine-tuning ») d'un ResNet 18 pré-entraîné sur ImageNet1k

Image « normalisée »



3x224x224

7x7 Conv No bias  
K=64 Pad=3 Stride=2

64x112x112

BN

64x112x112

ReLu

64x112x112

3x3 MaxPool  
Pad=1 Stride=2

64x56x56

ResBlock A

64x56x56

ResBlock A

64x56x56

ResBlock B

128x28x28

$\arg \max(s) = 2 \doteq \text{''Présence''}$

2x512 FC

2x1

7x7 Average Pool

512x1

ResBlock A

512x7x7

ResBlock B

512x7x7

ResBlock A

256x14x14

ResBlock B

256x14x14

ResBlock A

128x28x28

Paramètres du ResNet18 pré-entraîné sur ImageNet

Initialisation classique (aléatoire)

## Modèles de fondation « modernes »

Pré-entraînement sur une **très** grande base de données.

Exemple : ~2012 ImageNet 1k =  $1.2 \times 10^6$  images étiquetées  
~2022 LAION-5B =  $5 \times 10^9$  images étiquetées

## Modèles de fondation « modernes »

Pré-entraînement sur une **très** grande base de données.

Exemple : ~2012 ImageNet 1k =  $1.2 \times 10^6$  images étiquetées

~2022 LAION-5B =  $5 \times 10^9$  images étiquetées

Rendu possible grâce à :

- l'augmentation continue de la puissance des cartes graphiques
- l'apparition des couches d'attention (« Transformer »)

## Modèles de fondation « modernes »

Pré-entraînement sur une **très** grande base de données.

Exemple : ~2012 ImageNet 1k =  $1.2 \times 10^6$  images étiquetées

~2022 LAION-5B =  $5 \times 10^9$  images étiquetées

Rendu possible grâce à :

- l'augmentation continue de la puissance des cartes graphiques
- l'apparition des couches d'attention (« Transformer »)

Apparition de modèles de fondation dans un grand nombre de domaines :

- Reconstruction 3D, Segmentation d'image, Imagerie médicale etc.

## Modèles de fondation « modernes »

Pré-entraînement sur une **très** grande base de données.

Exemple : ~2012 ImageNet 1k =  $1.2 \times 10^6$  images étiquetées

~2022 LAION-5B =  $5 \times 10^9$  images étiquetées

Rendu possible grâce à :

- l'augmentation continue de la puissance des cartes graphiques
- l'apparition des couches d'attention (« Transformer »)

Apparition de modèles de fondation dans un grand nombre de domaines :

- Reconstruction 3D, Segmentation d'image, Imagerie médicale etc.
- Et bien-sûr les LLM (« Large Language Model ») avec GPT !

## II) Exemple de modèle de fondation pour la segmentation d'images

### « Segment Anything »

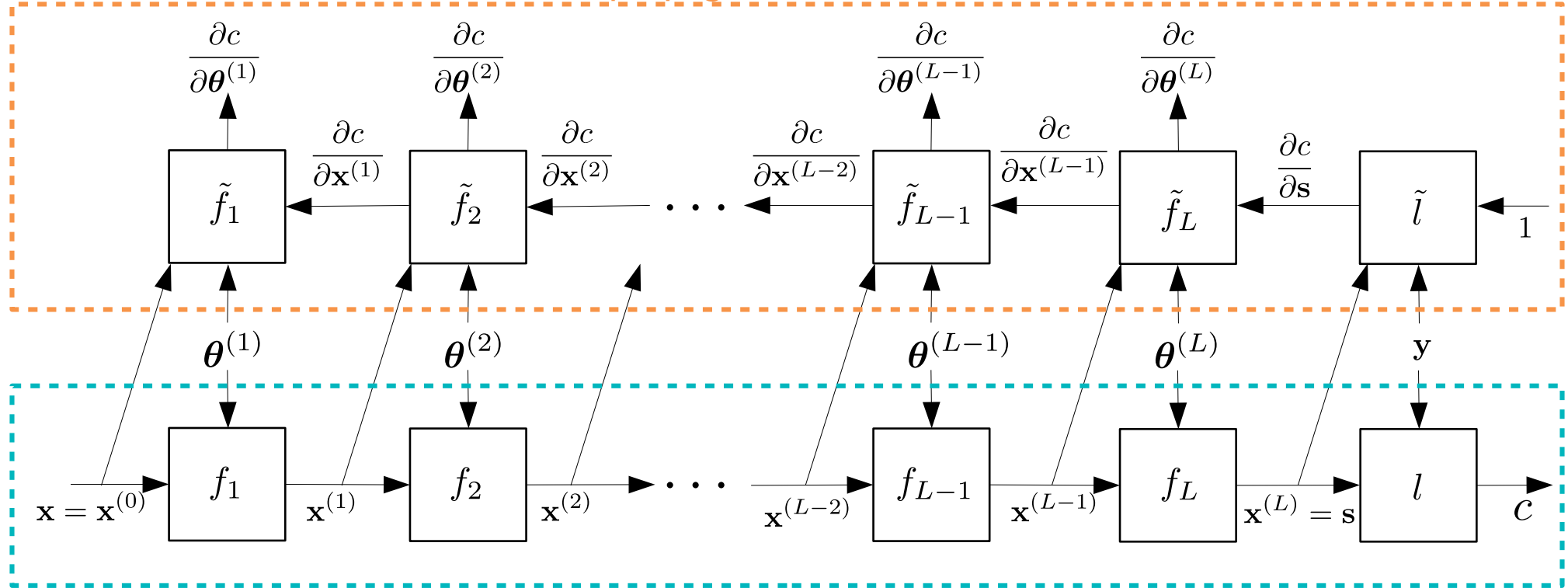


Entraînement sur « SA-1B » : une base de  $11 \times 10^6$  images avec  $1.1 \times 10^9$  masques de segmentation

### III) « Gradient checkpointing »

# Rétropropagation « classique »

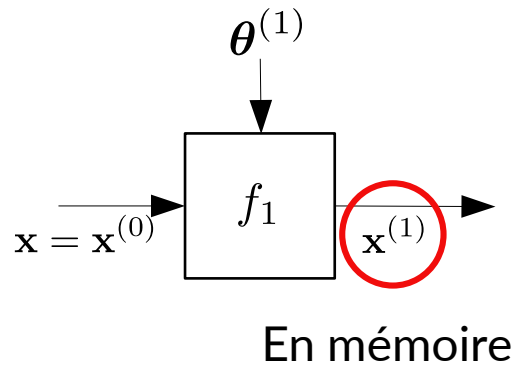
Rétropropagation (“Backward”)



Propagation avant (“Forward”)

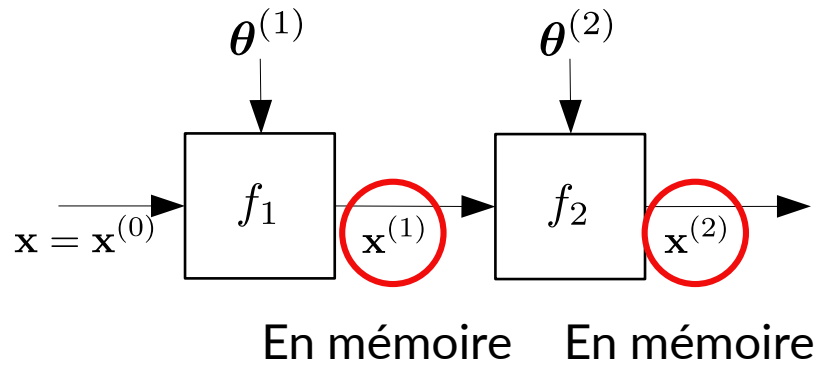
III)

## Rétropropagation « classique »



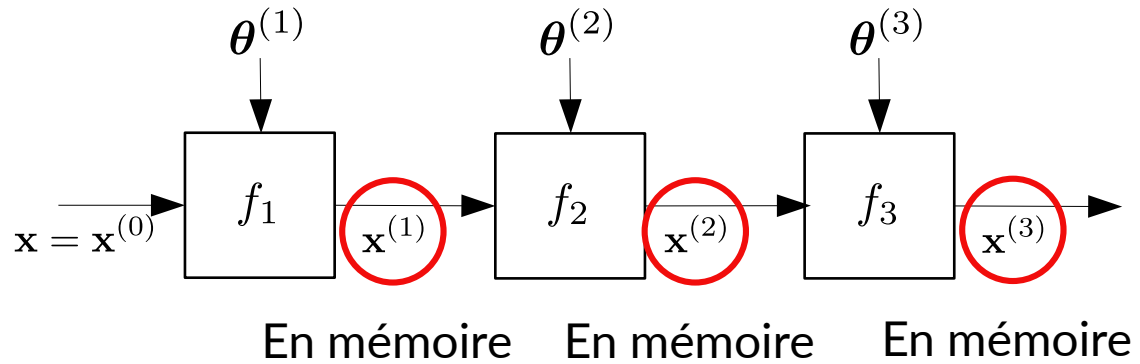
III)

## Rétropropagation « classique »



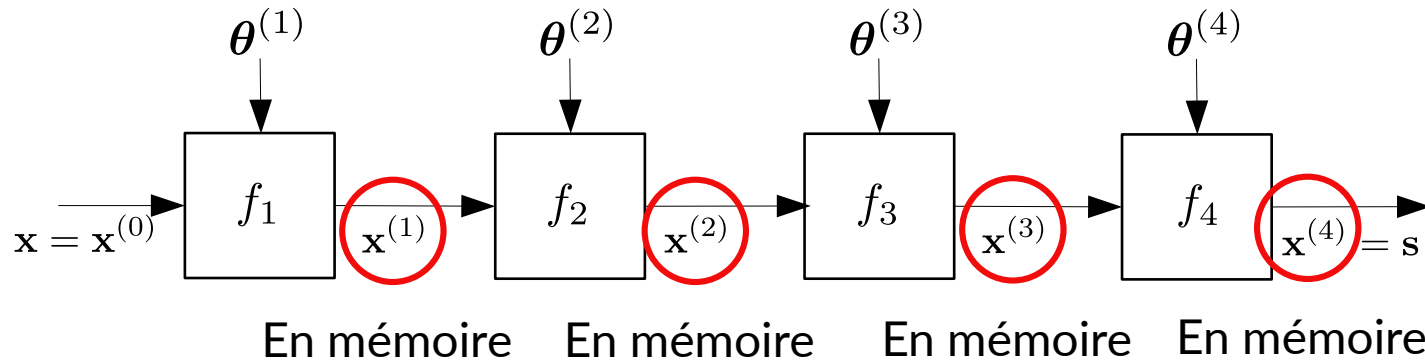
III)

## Rétropropagation « classique »



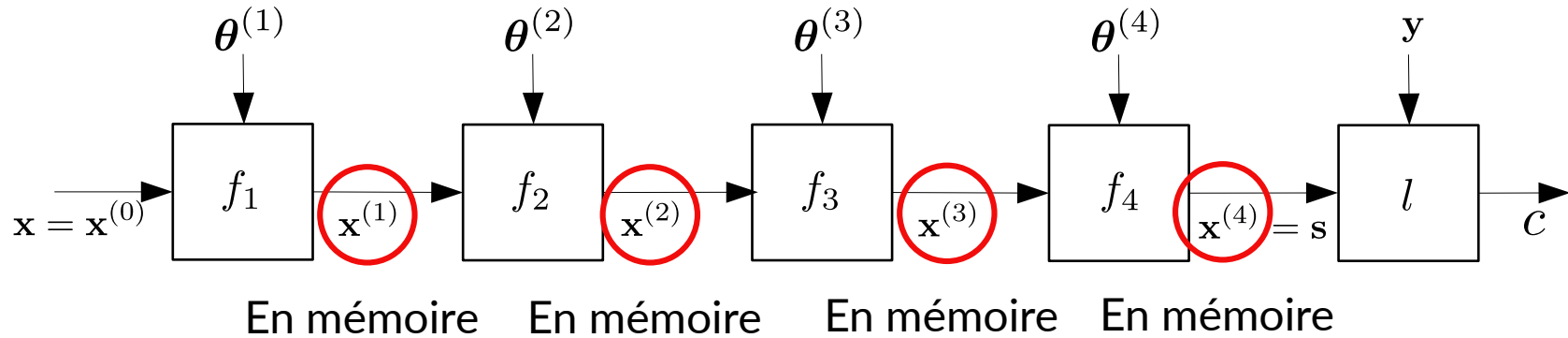
III)

## Rétropropagation « classique »

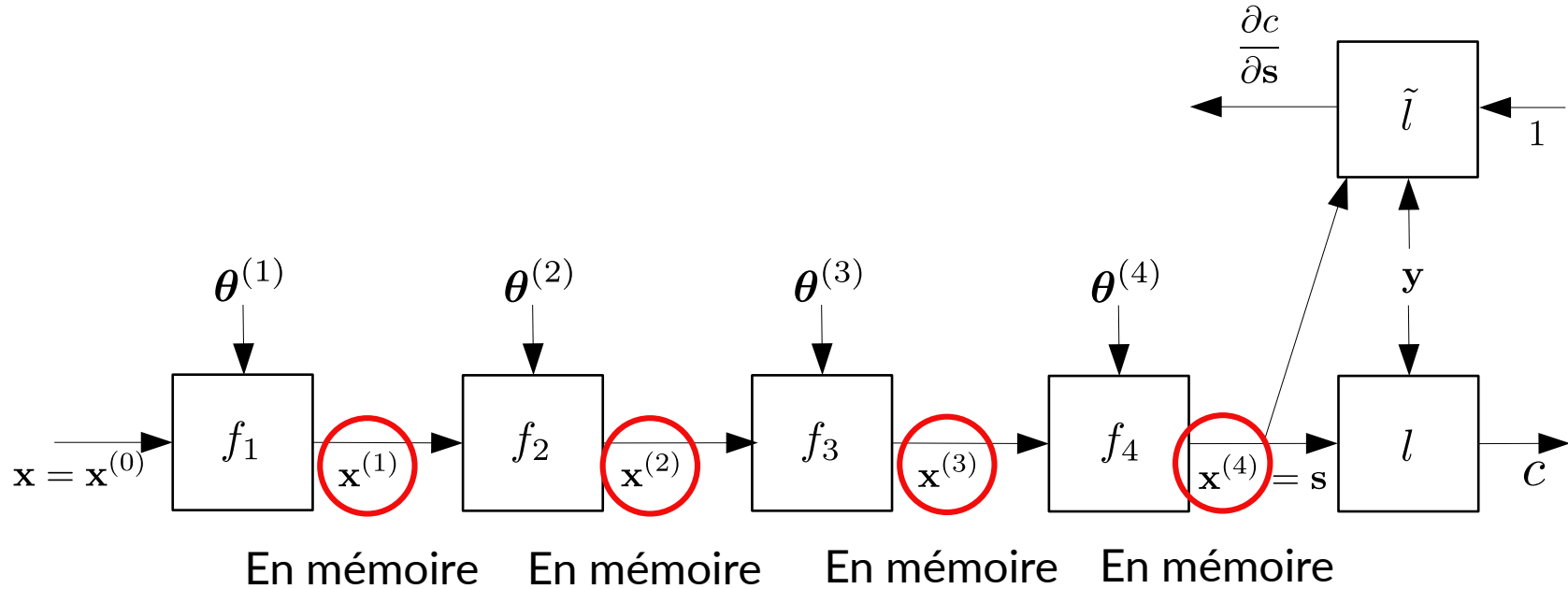


III)

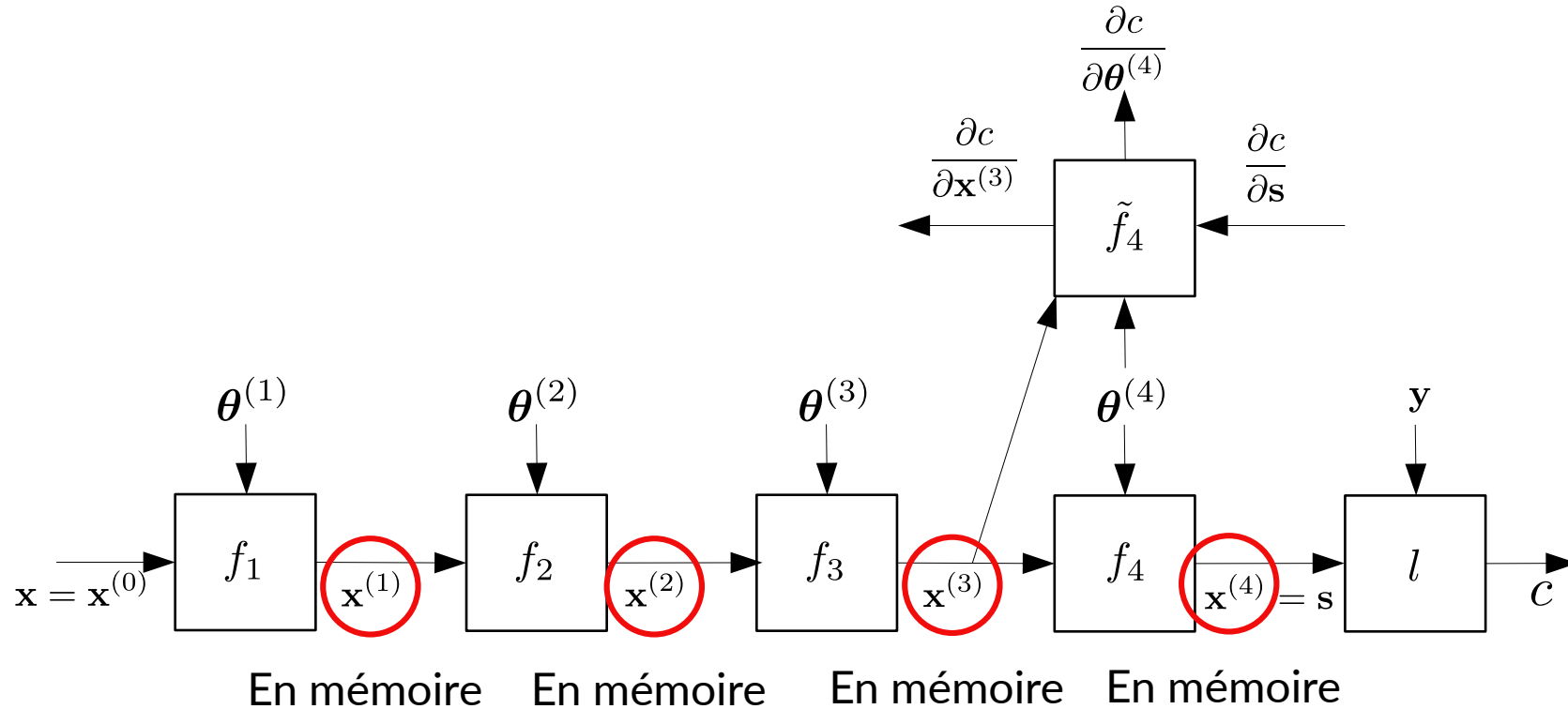
## Rétropropagation « classique »



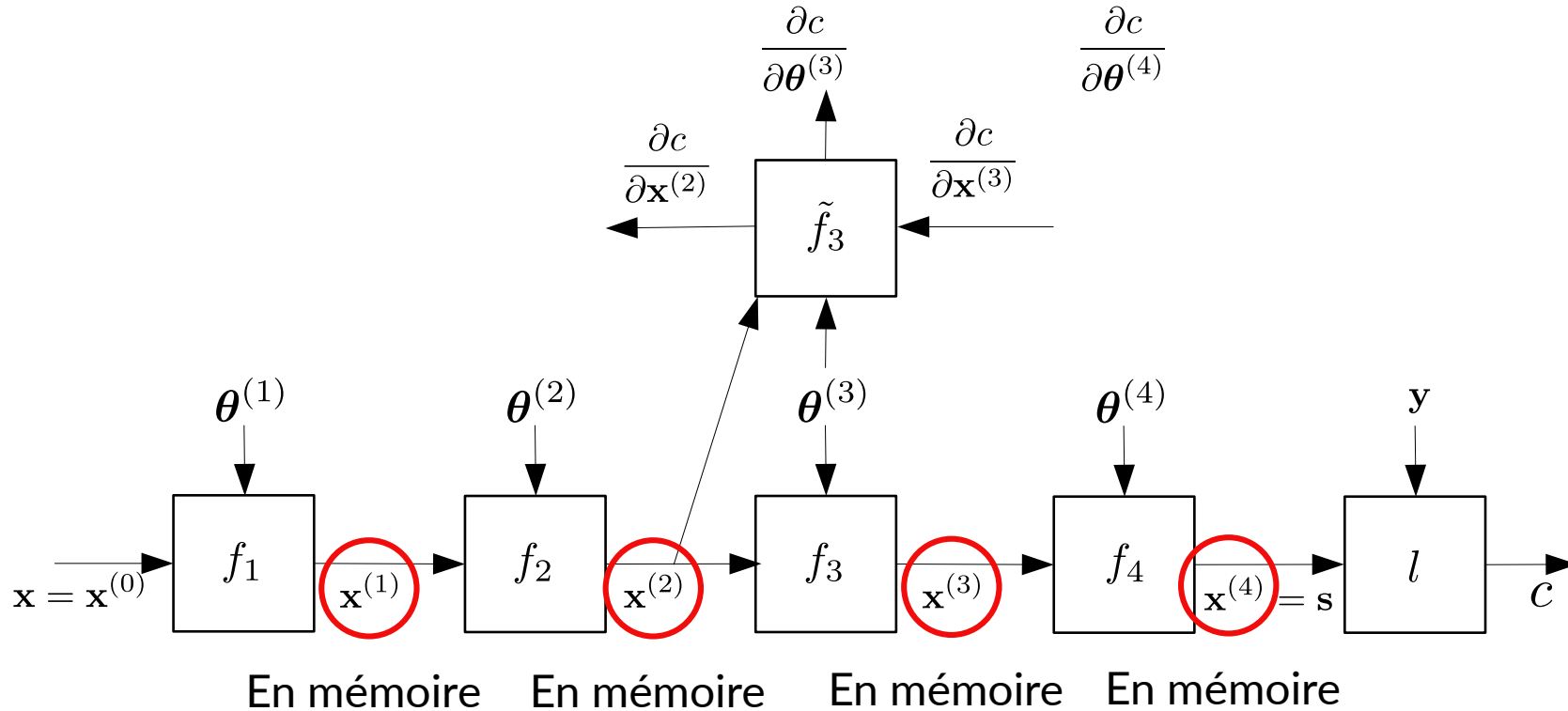
# Rétropropagation « classique »



# Rétropropagation « classique »

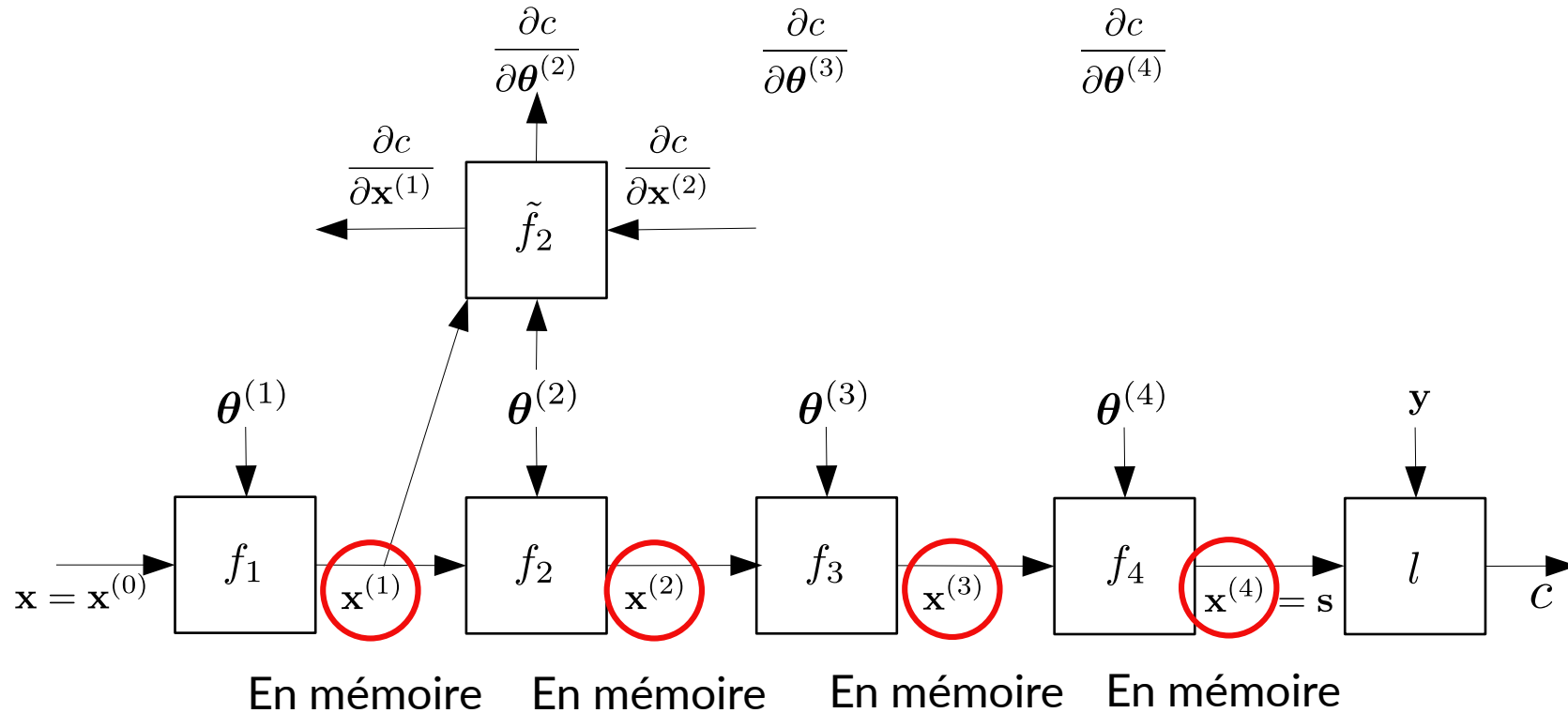


# Rétropropagation « classique »

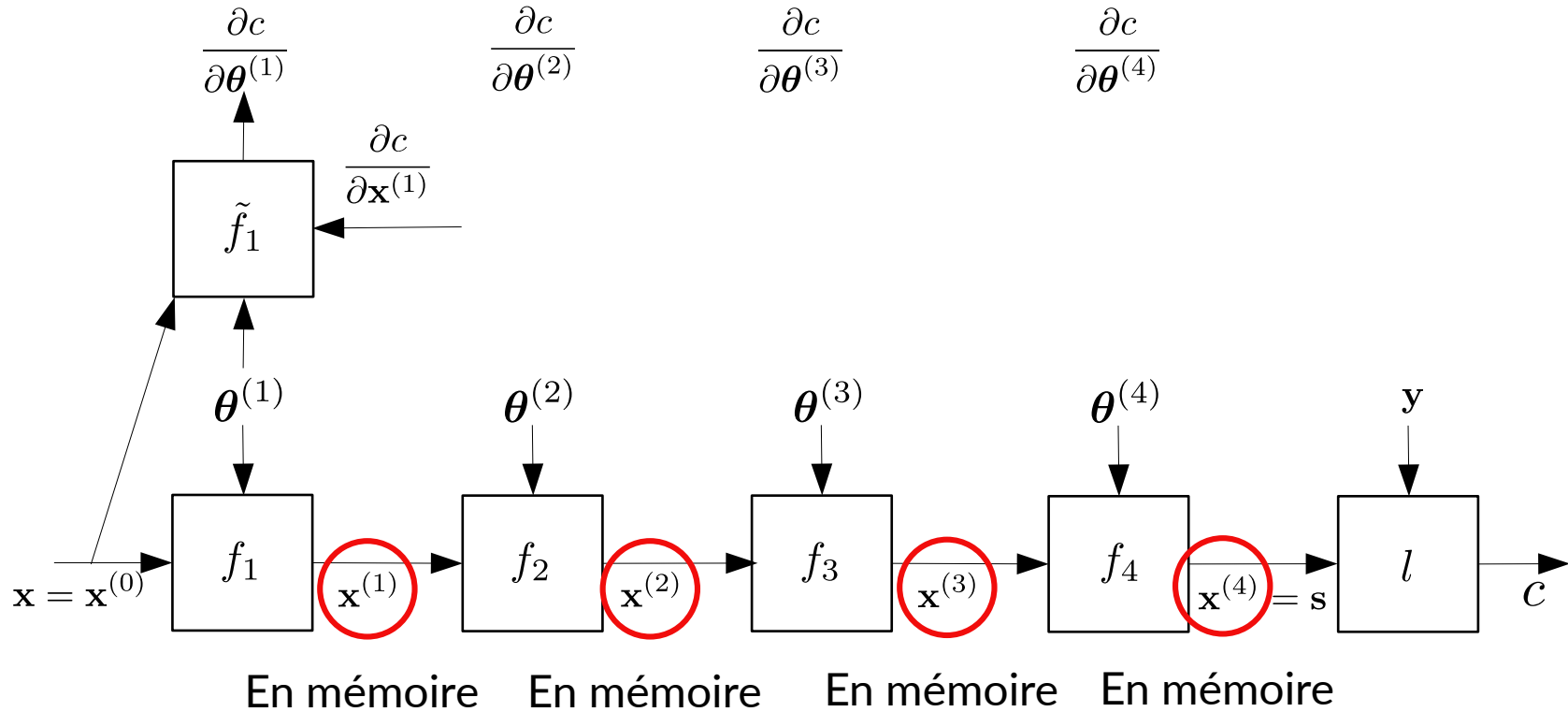


III)

## Rétropropagation « classique »

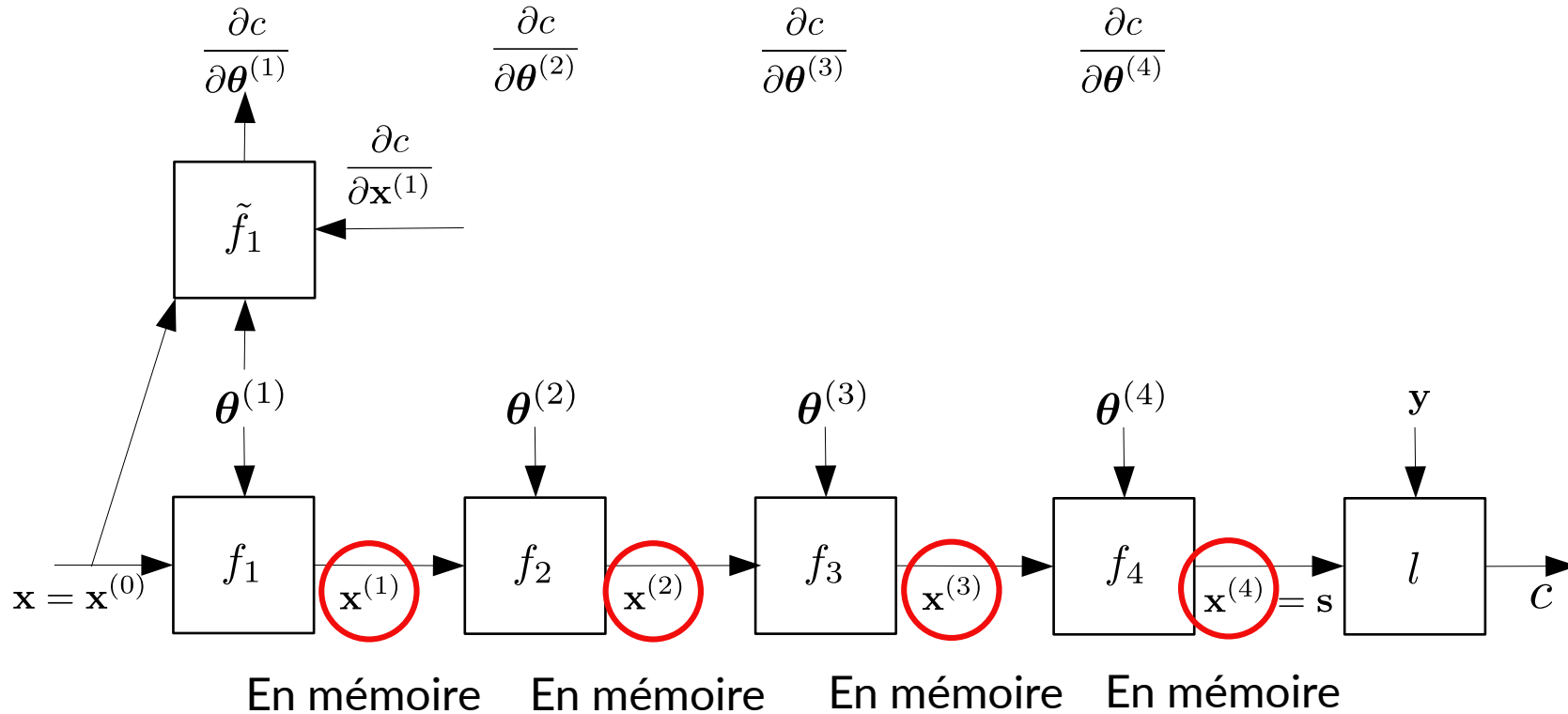


# Rétropropagation « classique »



Toutes les activations sont conservées en mémoire pour effectuer la rétropropagation.

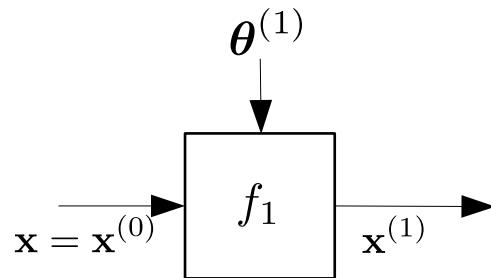
## Rétropropagation « classique »



Toutes les activations sont conservées en mémoire pour effectuer la rétropropagation.  
 « Gradient checkpointing » = conserver uniquement quelques activations en mémoire

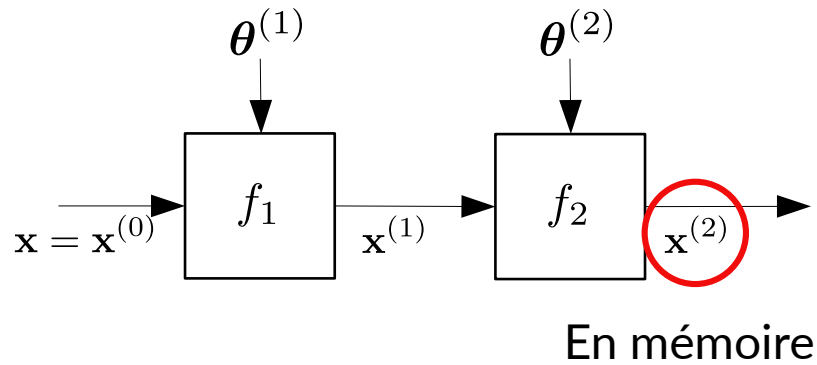
III)

## Exemple de « checkpointing » des couches 2 et 4



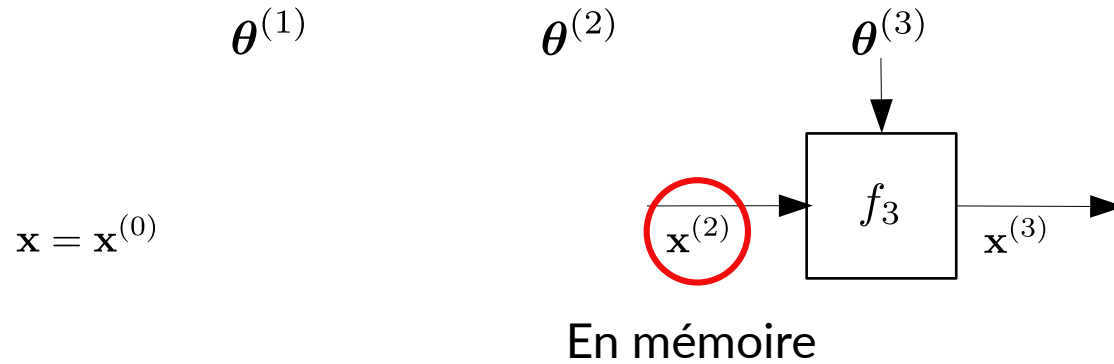
III)

## Exemple de « checkpointing » des couches 2 et 4



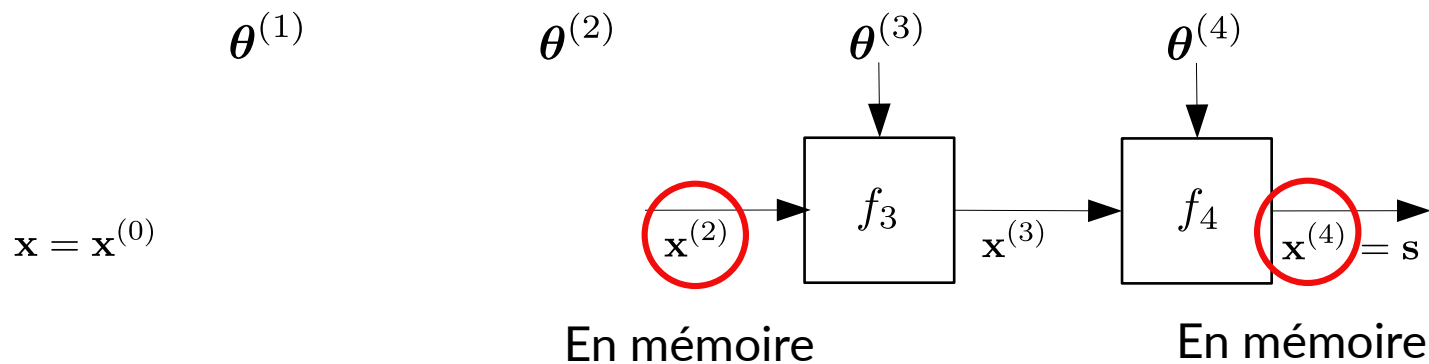
III)

## Exemple de « checkpointing » des couches 2 et 4



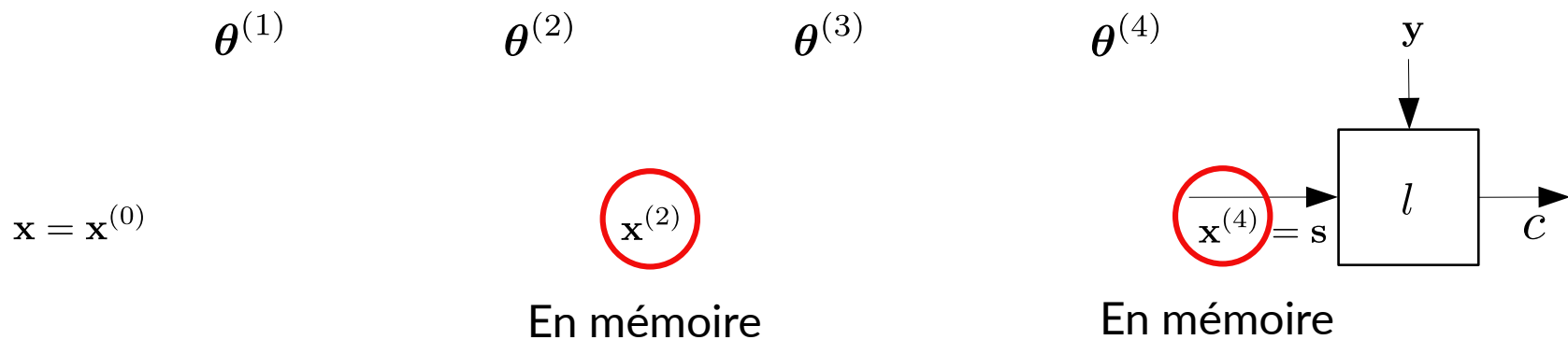
III)

## Exemple de « checkpointing » des couches 2 et 4



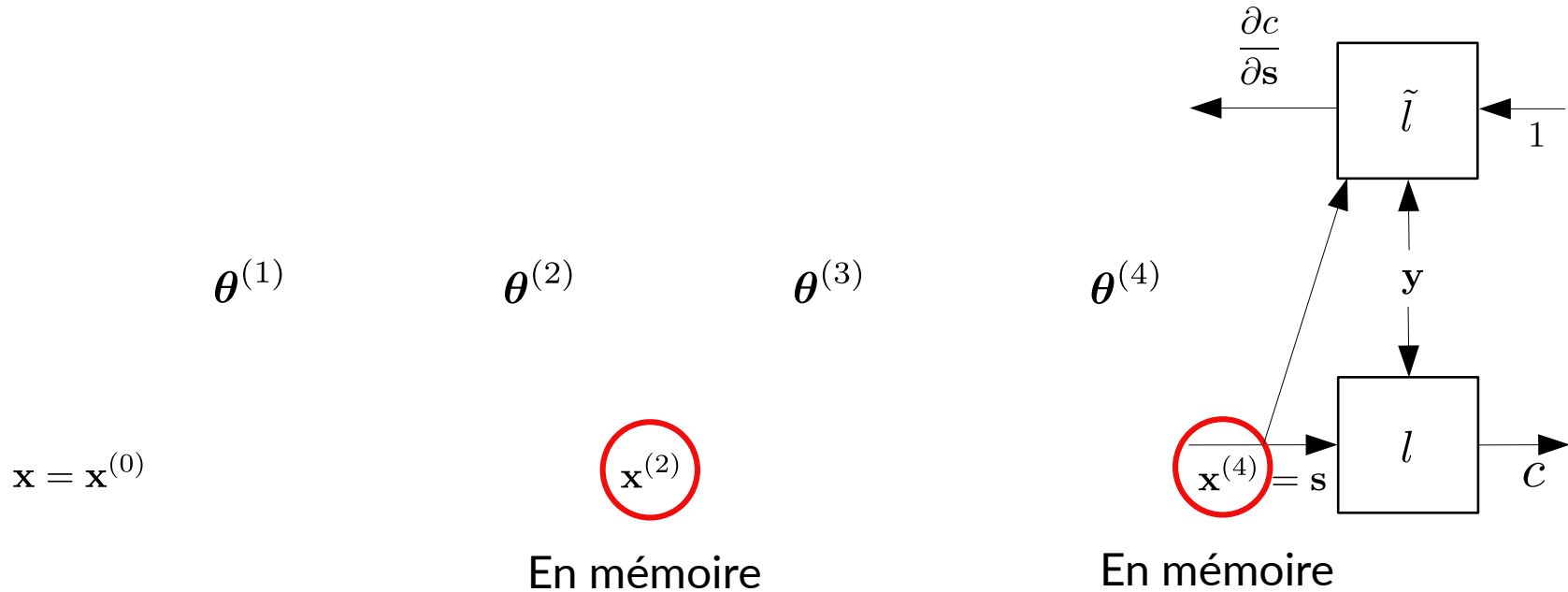
III)

## Exemple de « checkpointing » des couches 2 et 4



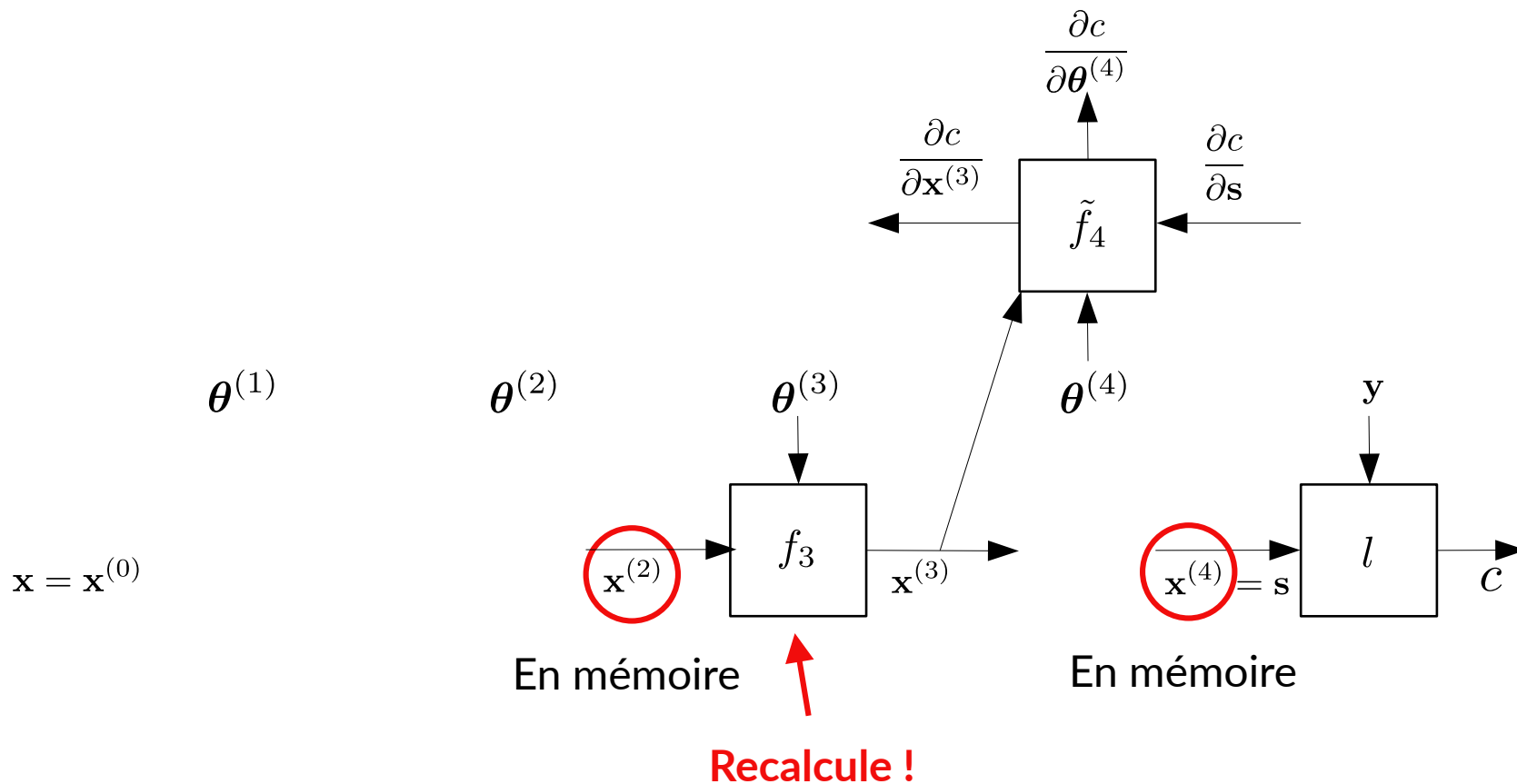
III)

## Exemple de « checkpointing » des couches 2 et 4



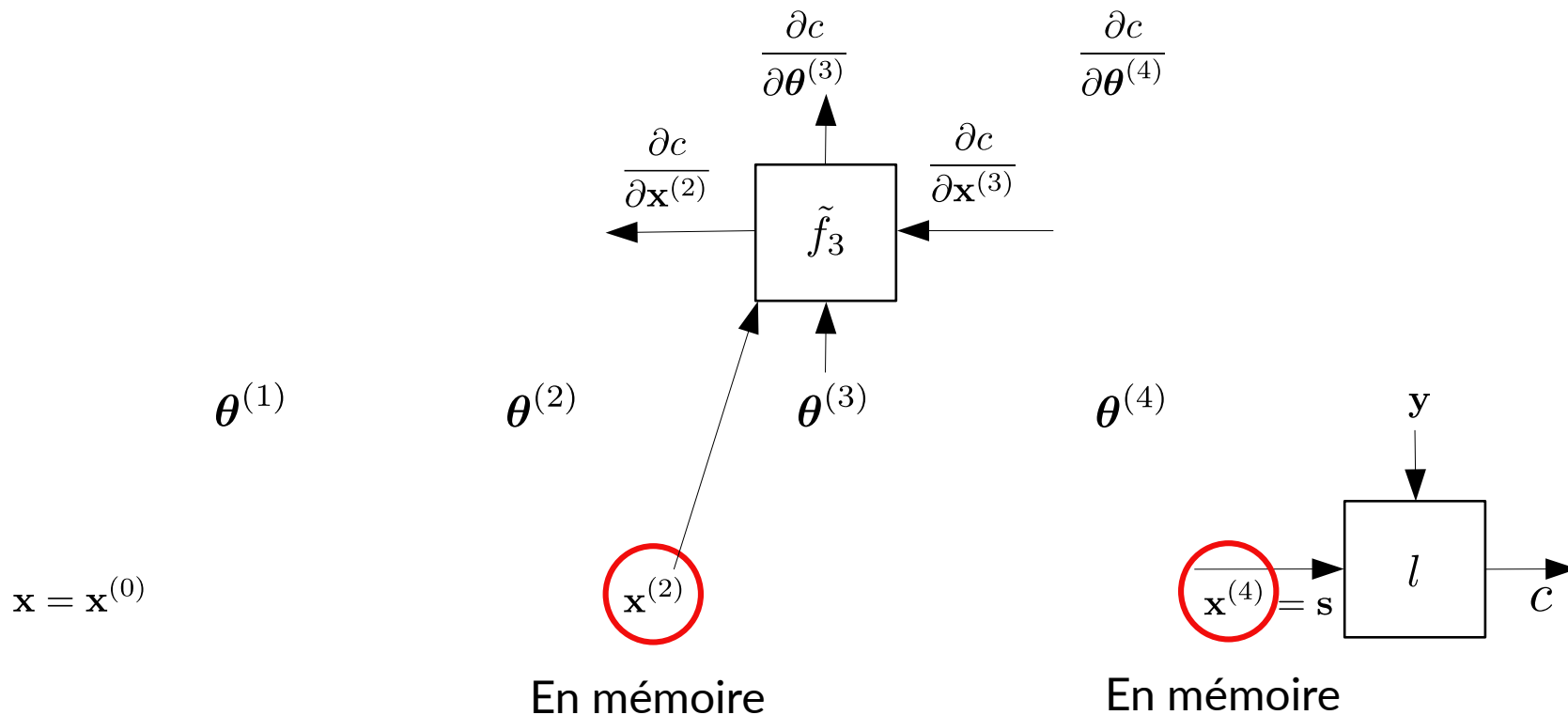
III)

## Exemple de « checkpointing » des couches 2 et 4



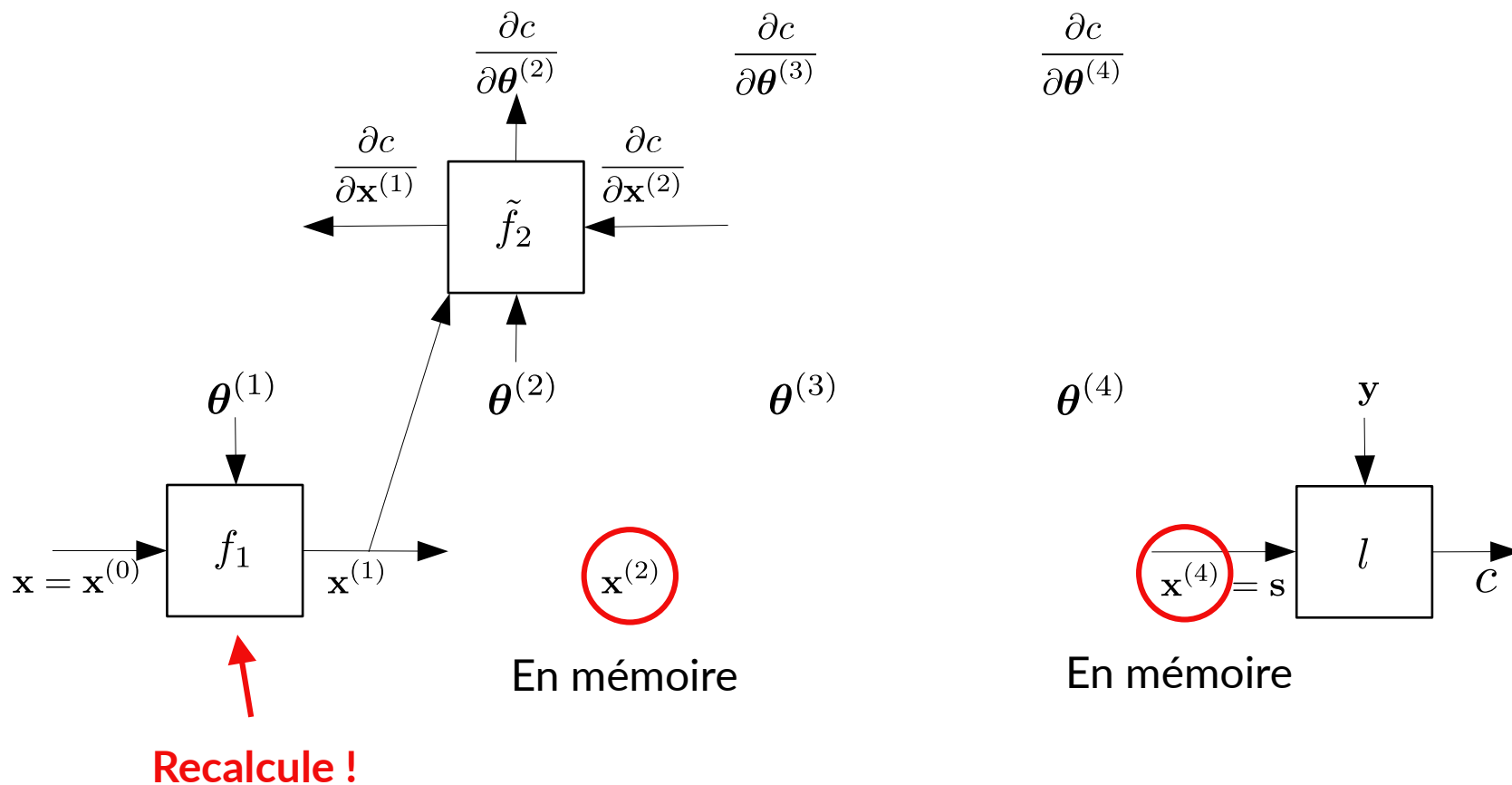
III)

## Exemple de « checkpointing » des couches 2 et 4



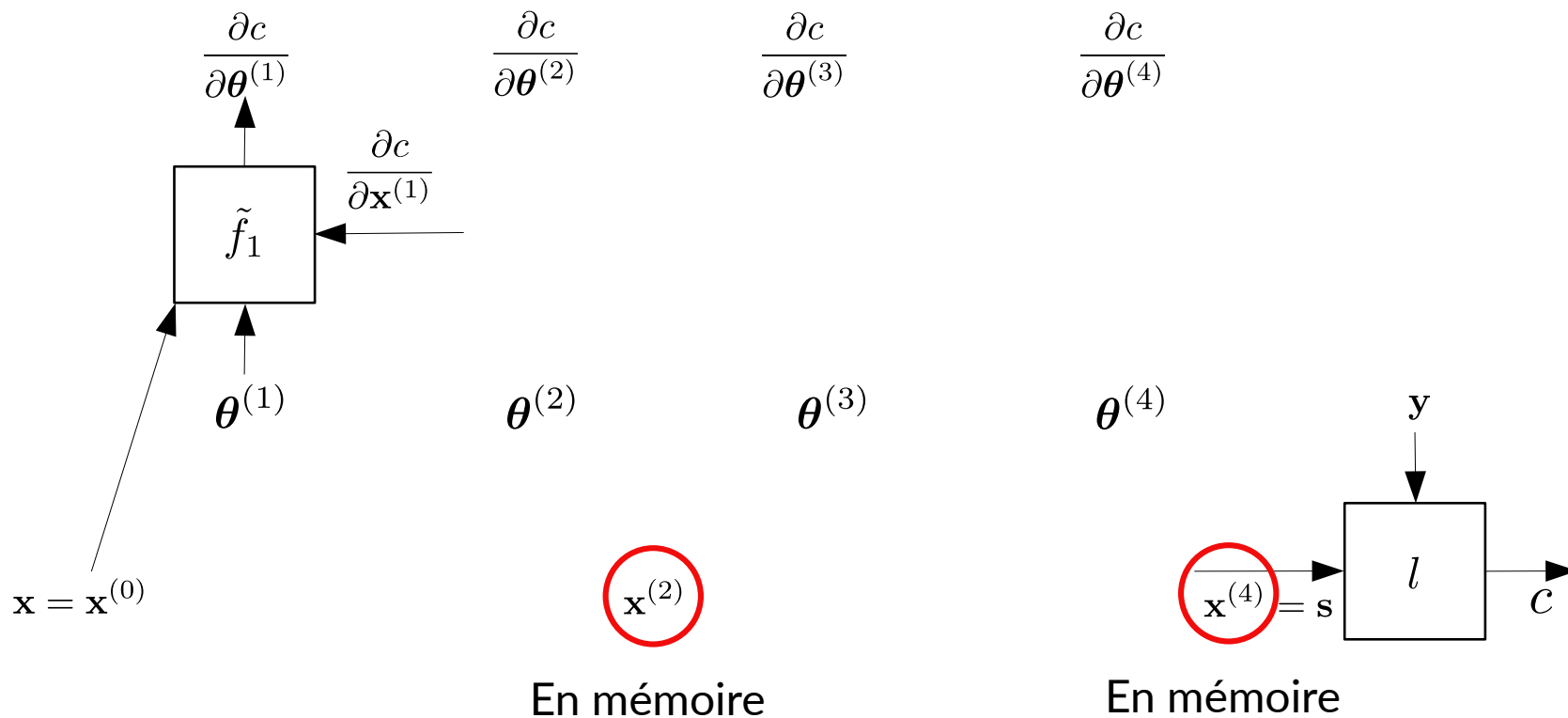
III)

## Exemple de « checkpointing » des couches 2 et 4



III)

## Exemple de « checkpointing » des couches 2 et 4



## Résumé « checkpointing »

Réduit l'empreinte mémoire de la rétropropagation

→ permet d'entraîner de plus gros réseau ou d'augmenter la taille du minibatch

Nécessite de recalculer des activations

→ Accroît le temps de rétropropagation et donc le temps d'entraînement

## IV) Apprentissage multi-GPU

## Résumé des ingrédients du « Deep Learning »

1) Grande base de données étiquetées

2) « Bonne » architecture de réseau de neurones profond

- ▶ « Perceptron » multicouche, Réseau de neurones à convolution, Transformer
- ▶ Optimisation par descente de gradient stochastique (AdamW, etc.)

3) Grande capacité de calculs en parallèle (GPUs)

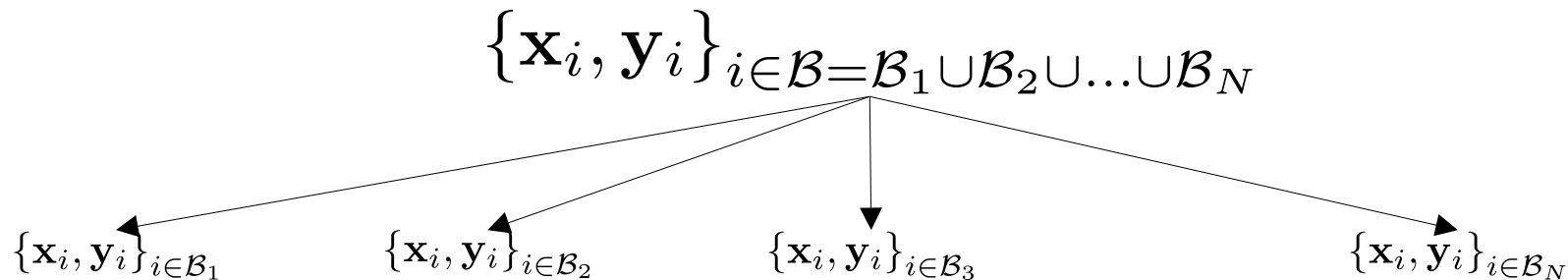
## Grande capacité de calculs en parallèle

Hypothèse : On dispose de plusieurs GPUs sur lesquels rentrent de « petits » minibatches

$$\{\mathbf{x}_i, \mathbf{y}_i\}_{i \in \mathcal{B} = \mathcal{B}_1 \cup \mathcal{B}_2 \cup \dots \cup \mathcal{B}_N}$$

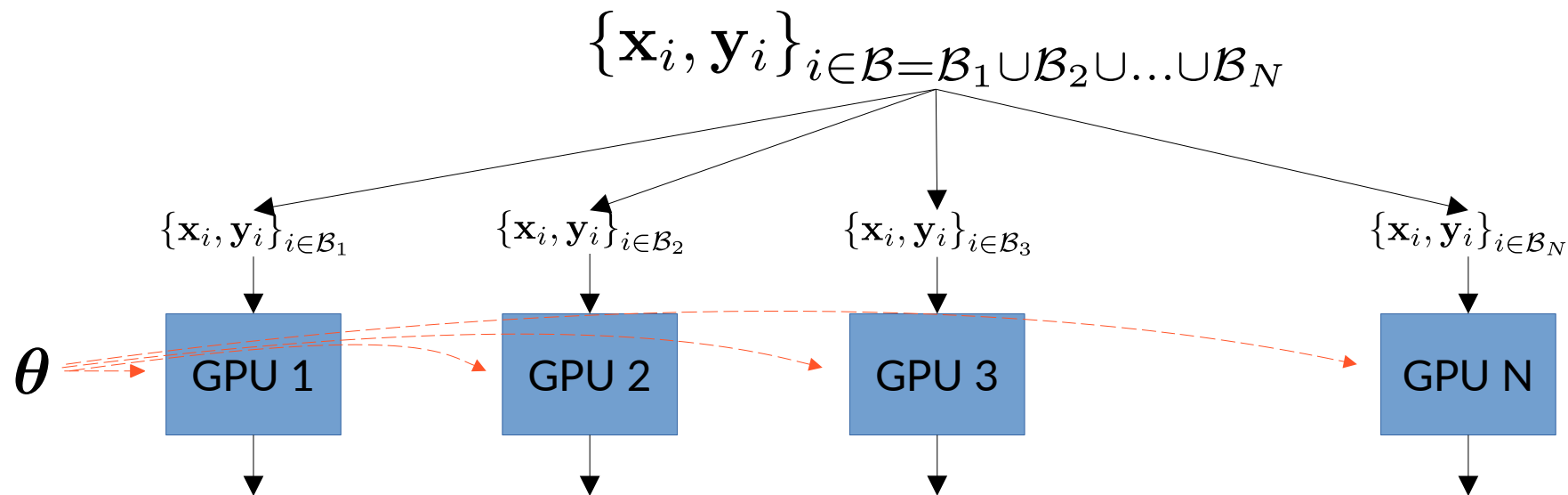
## Grande capacité de calculs en parallèle

Hypothèse : On dispose de plusieurs GPUs sur lesquels rentrent de « petits » minibatches



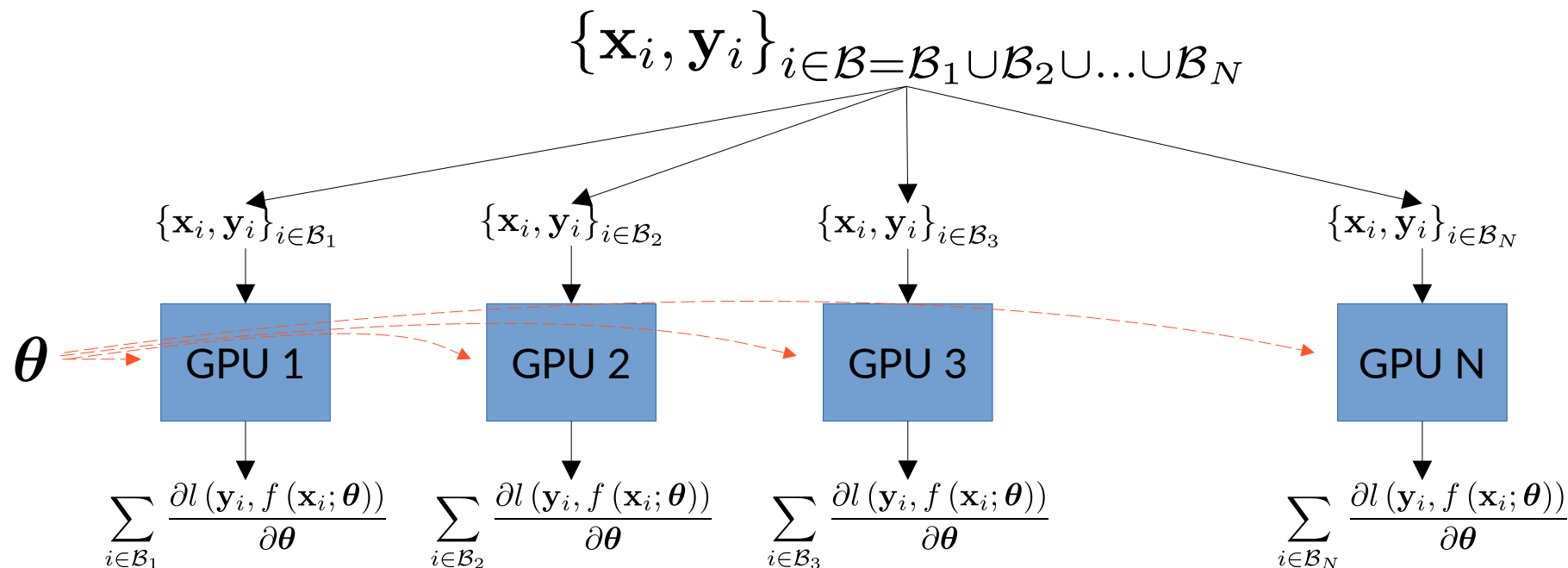
# Grande capacité de calculs en parallèle

Hypothèse : On dispose de plusieurs GPUs sur lesquels rentrent de « petits » minibatches



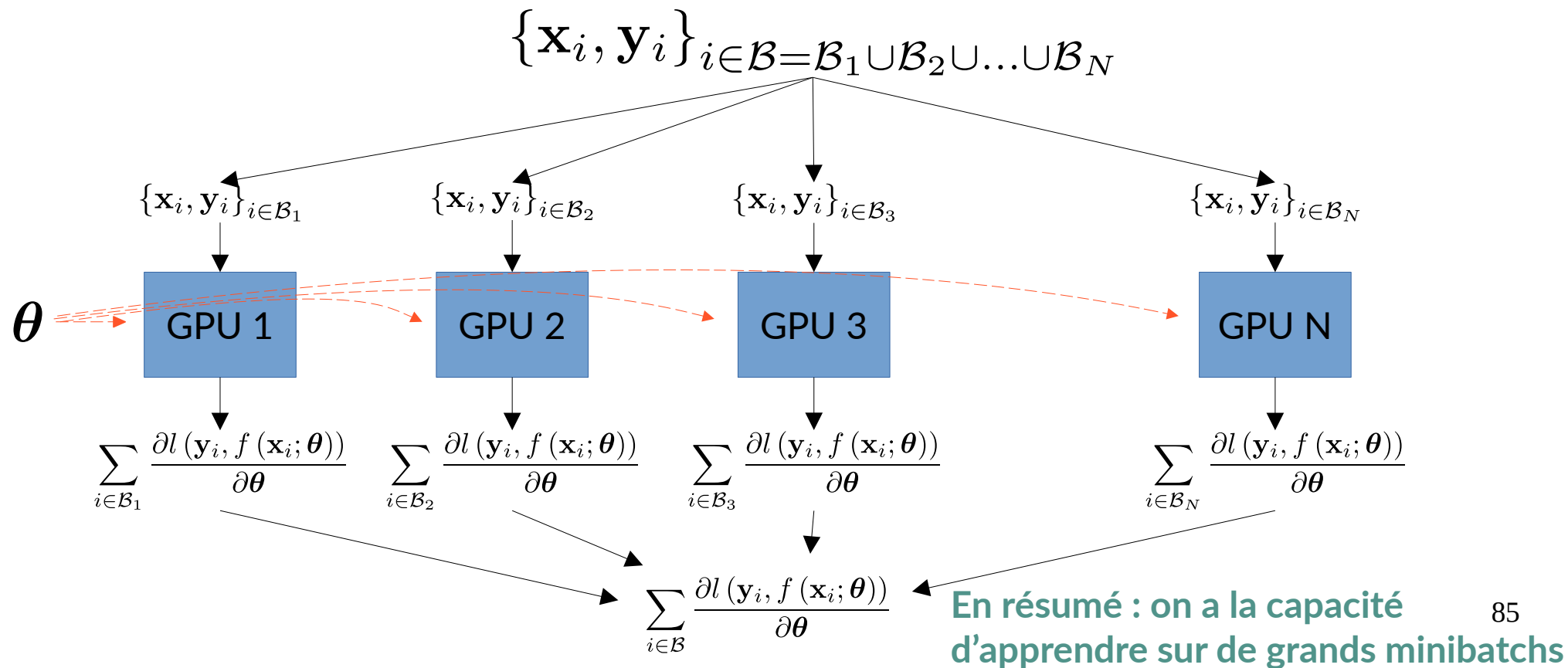
# Grande capacité de calculs en parallèle

Hypothèse : On dispose de plusieurs GPUs sur lesquels rentrent de « petits » minibatches



# Grande capacité de calculs en parallèle

Hypothèse : On dispose de plusieurs GPUs sur lesquels rentrent de « petits » minibatches



## Grande capacité de calculs en parallèle

Hypothèse : On dispose de plusieurs GPUs sur lesquels rentrent de « petits » minibatches



On a la capacité d'apprendre sur de grands minibatches

## Grande capacité de calculs en parallèle

Hypothèse : On dispose de plusieurs GPUs sur lesquels rentrent de « petits » minibatches



On a la capacité d'apprendre sur de grands minibatches



Peut-on exploiter cette capacité pour réduire le temps d'entraînement ?

## Grande capacité de calculs en parallèle

Hypothèse : On dispose de plusieurs GPUs sur lesquels rentrent de « petits » minibatches



On a la capacité d'apprendre sur de grands minibatches



Peut-on exploiter cette capacité pour réduire le temps d'entraînement ?



Oui ! Il suffit d'augmenter le pas d'apprentissage (« learning rate ») et de modifier l'évolution du pas d'apprentissage (« scheduling ») !

## Augmentation du pas d'apprentissage

« **Linear Scaling Rule** » : Quand la taille du minibatch est multipliée par  $s$ , multiplier le pas d'apprentissage par  $s$ .

# Augmentation du pas d'apprentissage

« **Linear Scaling Rule** » : Quand la taille du minibatch est multipliée par  $s$ , multiplier le pas d'apprentissage par  $s$ .

$s$  itérations avec un minibatch de taille  $n$

$$\theta_{k+s} = \theta_k - \alpha \frac{1}{n} \sum_{j=0}^{s-1} \sum_{i \in \mathcal{B}_j} \frac{\partial l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))}{\partial \theta} \Big|_{\theta = \theta_{k+j}}$$

1 itération avec un minibatch de taille  $sn$

$$\hat{\theta}_{k+1} = \theta_k - \hat{\alpha} \frac{1}{sn} \sum_{j=0}^{s-1} \sum_{i \in \mathcal{B}_j} \frac{\partial l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))}{\partial \theta} \Big|_{\theta = \theta_k}$$

# Augmentation du pas d'apprentissage

« **Linear Scaling Rule** » : Quand la taille du minibatch est multipliée par  $s$ , multiplier le pas d'apprentissage par  $s$ .

$s$  itérations avec un minibatch de taille  $n$

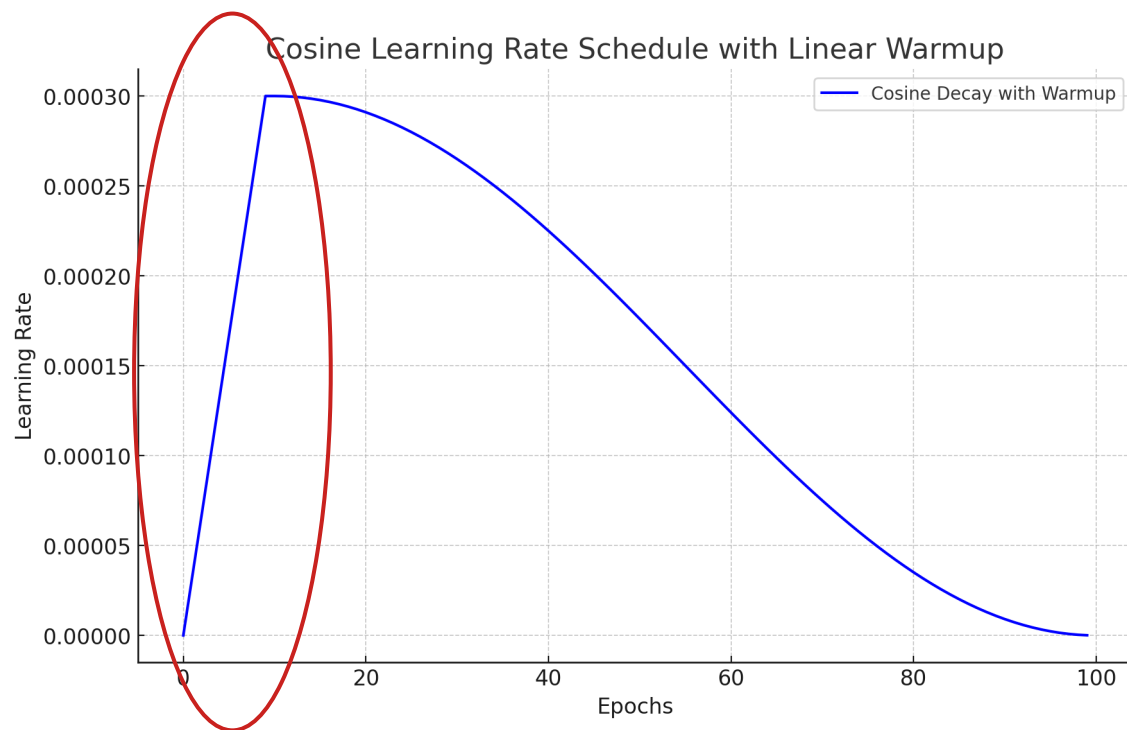
$$\theta_{k+s} = \theta_k - \alpha \frac{1}{n} \sum_{j=0}^{s-1} \sum_{i \in \mathcal{B}_j} \left. \frac{\partial l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))}{\partial \theta} \right|_{\theta = \theta_{k+j}}$$

1 itération avec un minibatch de taille  $sn$

$$\hat{\theta}_{k+1} = \theta_k - \hat{\alpha} \frac{1}{sn} \sum_{j=0}^{s-1} \sum_{i \in \mathcal{B}_j} \left. \frac{\partial l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))}{\partial \theta} \right|_{\theta = \theta_k}$$

Si  $\left. \frac{\partial l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))}{\partial \theta} \right|_{\theta = \theta_{k+j}} \approx \left. \frac{\partial l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))}{\partial \theta} \right|_{\theta = \theta_k}$  et  $\hat{\alpha} = s\alpha$  alors  $\theta_{k+s} \approx \hat{\theta}_{k+1}$

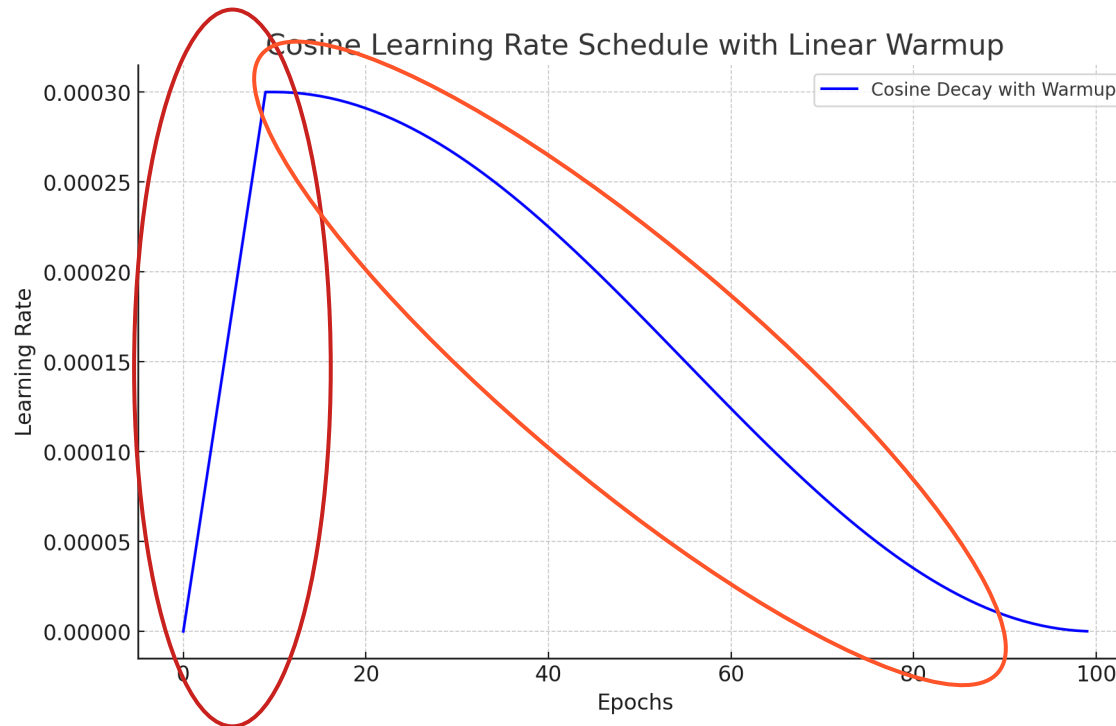
# Évolution du pas d'apprentissage



« Linear warmup » : augmentation progressive et **rapide** du pas d'apprentissage

Permet de se placer dans une « bonne » région de l'espace des paramètres

# Évolution du pas d'apprentissage

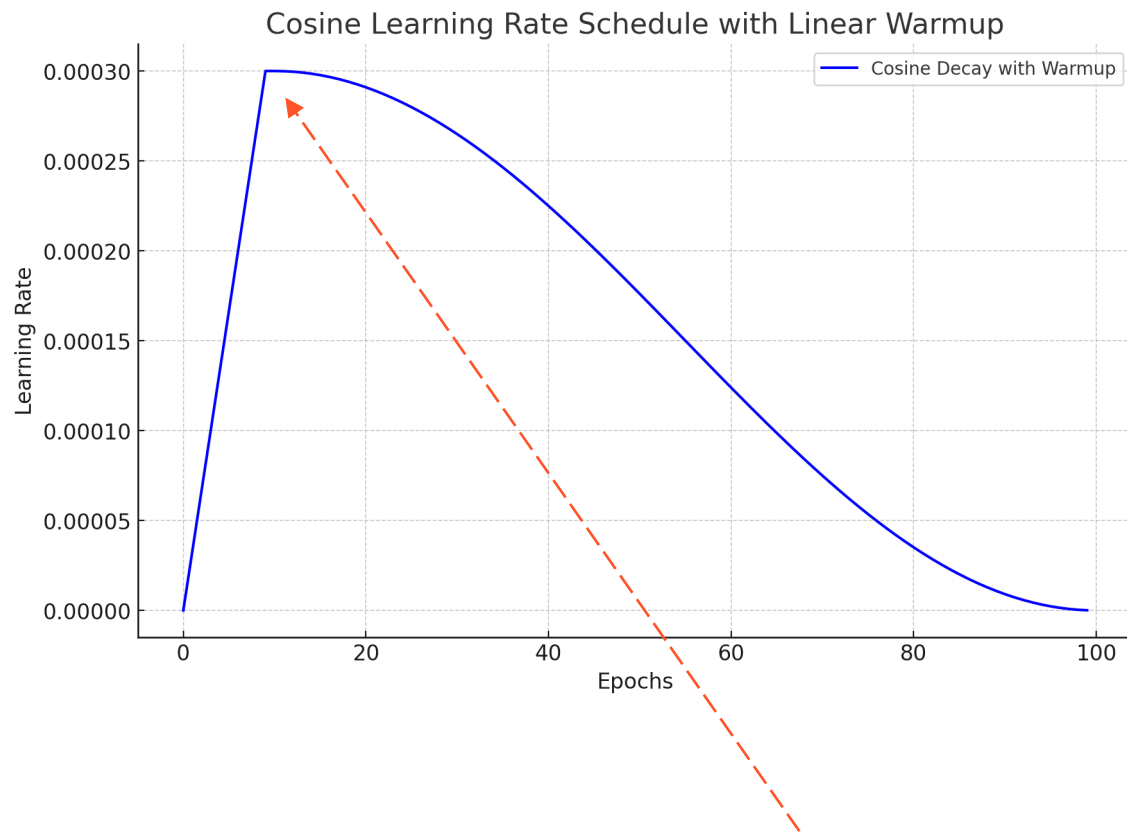


« Linear warmup » : augmentation progressive et **rapide** du pas d'apprentissage

« Decay » : décroissance progressive et **lente** du pas d'apprentissage

Décroissance « classique » du pas d'apprentissage

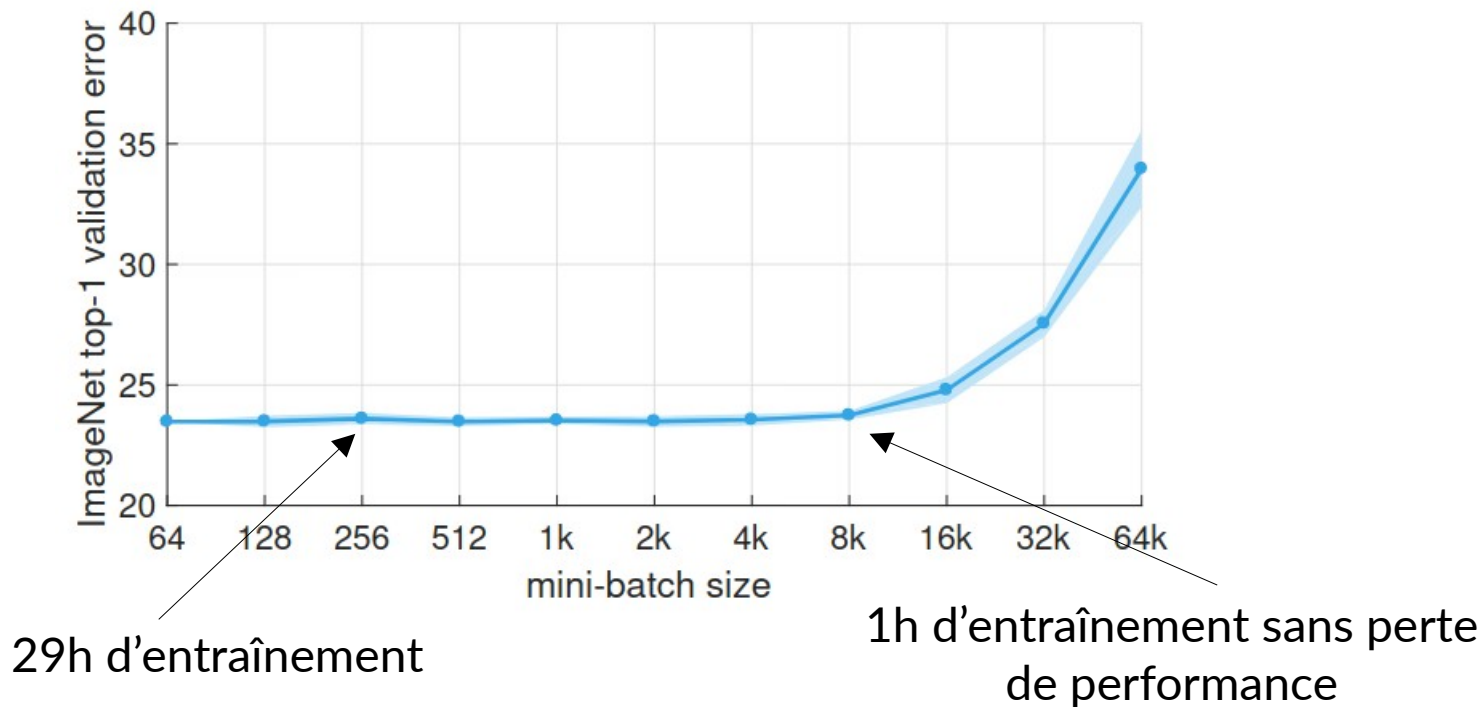
# Évolution du pas d'apprentissage



Valeur du pic définie par la « linear scaling rule ».

## Grande capacité de calculs en parallèle

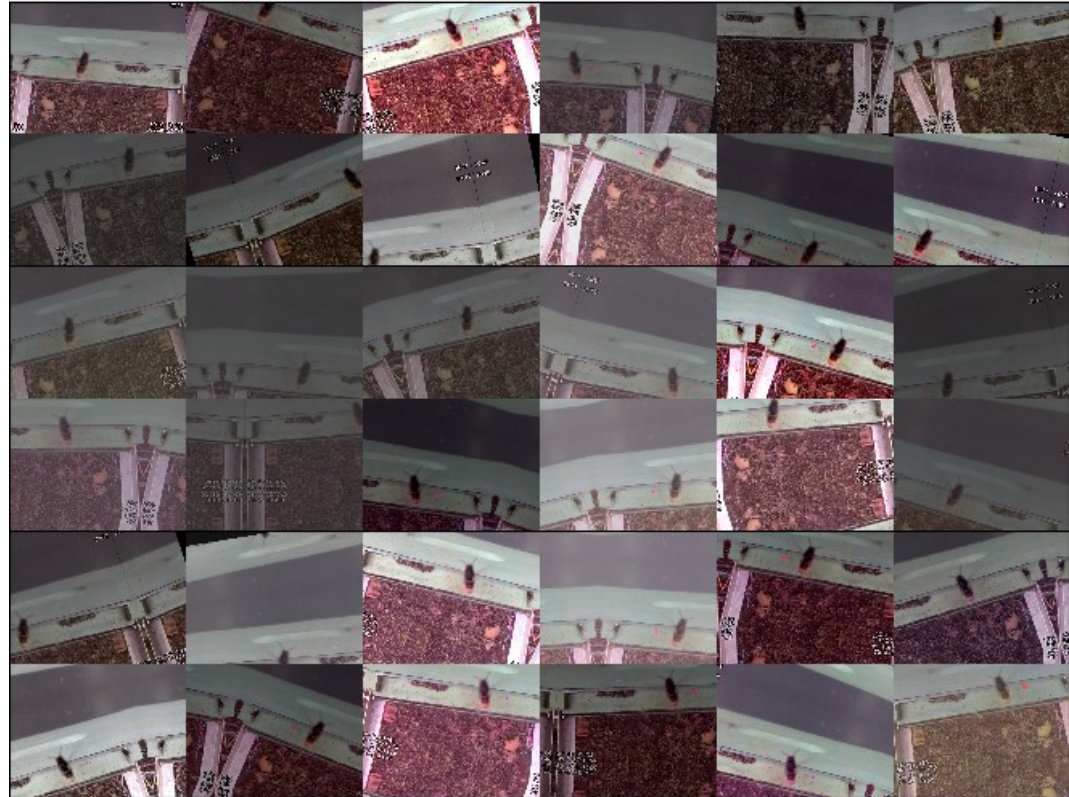
Il suffit d'augmenter le pas d'apprentissage (« learning rate ») et de modifier l'évolution du pas d'apprentissage (« scheduling ») !



# Annexes

# Augmentation de données

- Mirroir
- Transformation affine
- Perturbation couleur
- Effacement
- Bruit
- ...



Cubuk et al. (2019). Autoaugment: Learning augmentation strategies from data. CVPR

Cubuk et al. (2020). Randaugment: Practical automated data augmentation with a reduced search space. CVPR

# “Adversarial examples”



“panda”

57.7% confidence

+  $\epsilon$



=



“gibbon”

99.3% confidence

# “Adversarial patches”



Sources : “ADVHAT: Real-world adversarial attack on ArcFace face ID system” (<https://arxiv.org/pdf/1908.08705.pdf>)  
“Robust Physical-World Attacks on Deep Learning Visual Classification” (<https://arxiv.org/pdf/1707.08945.pdf>)

# Annexe : Application à la détection d'objets dans une image

# Détection d'objets

**Classification**



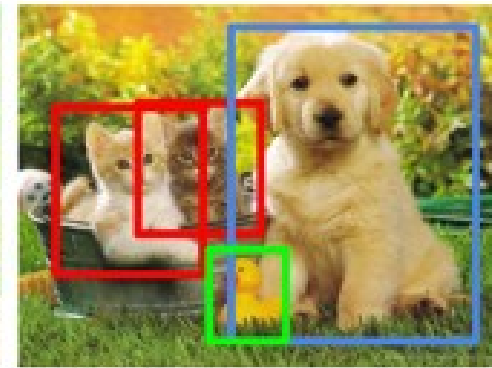
CAT

**Classification  
+ Localization**



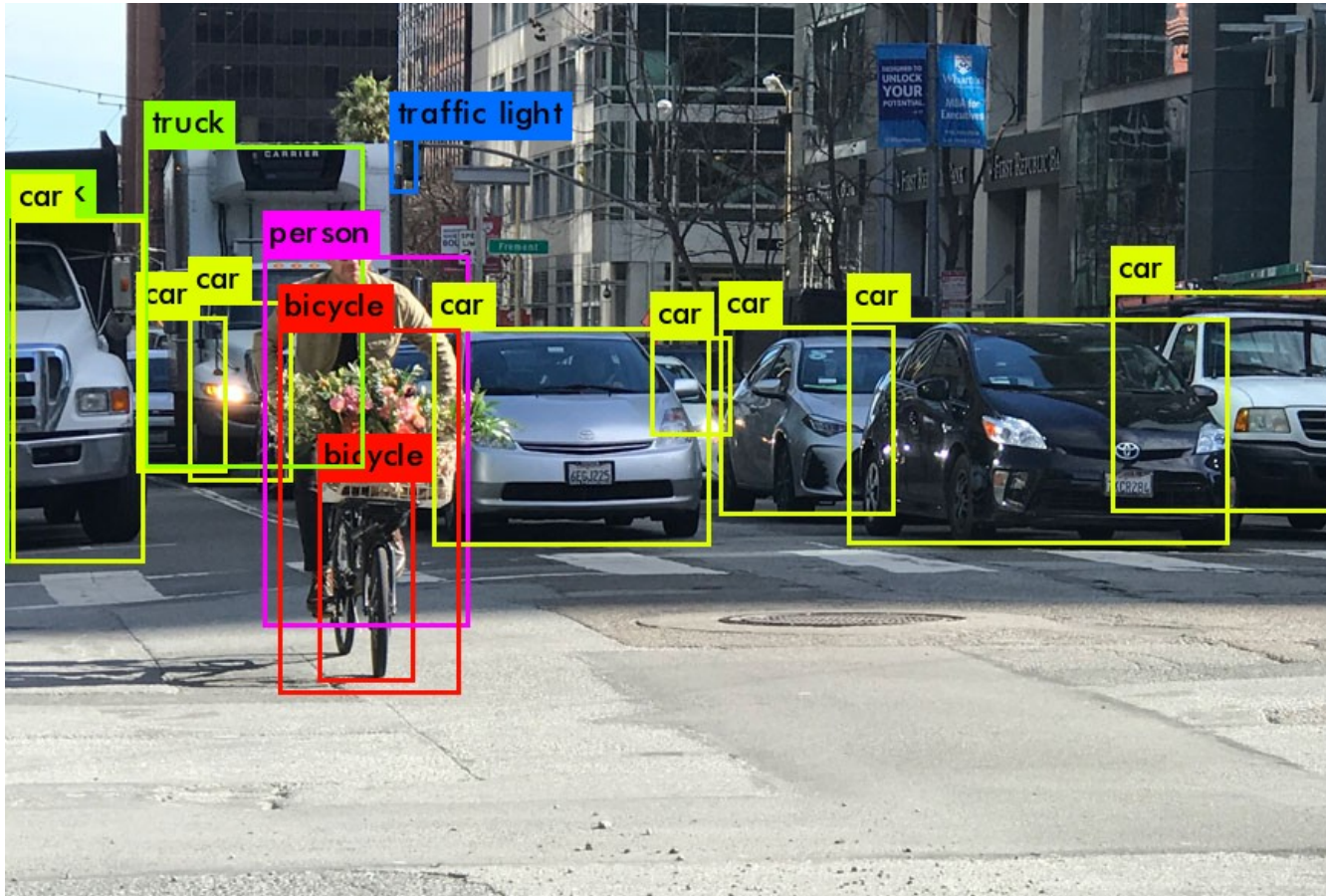
CAT

**Object Detection**



CAT, DOG, DUCK

# Exemple de détection d'objets



# Formulation du problème

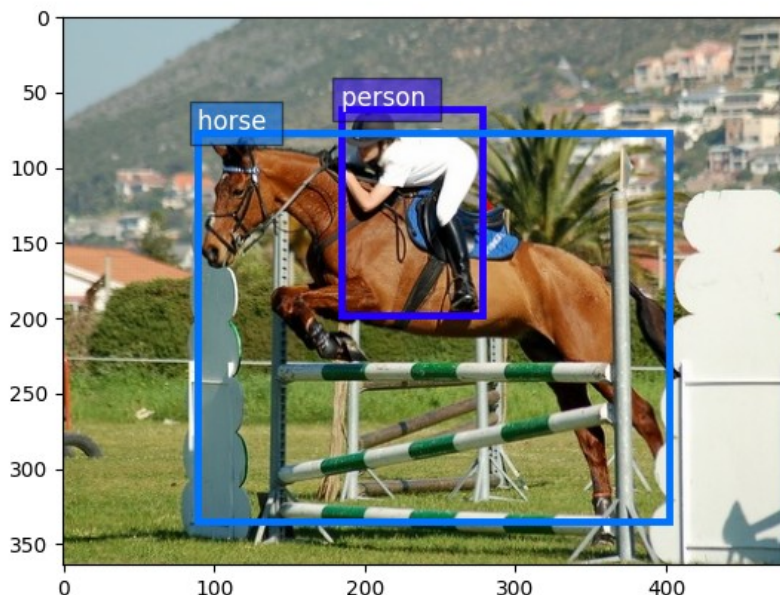
- Objectif
  - Prédire une boîte englobante autour de chaque objet,
  - Prédire la classe de l'objet,
  - En utilisant un réseau de neurones en apprentissage supervisé.
- Questions
  - Quelles données annotées disponibles ?
  - Quelle architecture ?
  - Quelle fonction de coût ?

# Quelles données annotées disponibles ?

## Microsoft COCO dataset (80 classes)

## Pascal VOC dataset (20 classes)

1: 'person',	31: 'skis',	61: 'dining table',
2: 'bicycle',	32: 'snowboard',	62: 'toilet',
3: 'car',	33: 'sports ball',	63: 'tv',
4: 'motorcycle',	34: 'kite',	64: 'laptop',
5: 'airplane',	35: 'baseball bat',	65: 'mose',
6: 'bs',	36: 'baseball glove',	66: 'remote',
7: 'train',	37: 'skateboard',	67: 'keyboard',
8: 'truck',	38: 'srfboard',	68: 'cell phone',
9: 'boat',	39: 'tennis racket',	69: 'microwave',
10: 'traffic light',	40: 'bottle',	70: 'oven',
11: 'fire hydrant',	41: 'wine glass',	71: 'toaster',
12: 'stop sign',	42: 'cp',	72: 'sink',
13: 'parking meter',	43: 'fork',	73: 'refrigerator',
14: 'bench',	44: 'knife',	74: 'book',
15: 'bird',	45: 'spoon',	75: 'clock',
16: 'cat',	46: 'bowl',	76: 'vase',
17: 'dog',	47: 'banana',	77: 'scissors',
18: 'horse',	48: 'apple',	78: 'teddy bear',
19: 'sheep',	49: 'sandwich',	79: 'hair drier',
20: 'cow',	50: 'orange',	80: 'toothbrsh',
21: 'elephant',	51: 'broccoli',	
22: 'bear',	52: 'carrot',	
23: 'zebra',	53: 'hot dog',	
24: 'giraffe',	54: 'pizza',	
25: 'backpack',	55: 'dont',	
26: 'mbrella',	56: 'cake',	
27: 'handbag',	57: 'chair',	
28: 'tie',	58: 'coch',	
29: 'sitcase',	59: 'potted plant',	
30: 'frisbee',	60: 'bed',	



Person:  
1: person

Animal:  
2: bird  
3: cat  
4: cow  
5: dog  
6: horse  
7: sheep

Vehicle:  
8: aeroplane  
9: bicycle  
10: boat  
11: bus  
12: car  
13: motorbike  
14: train

Indoor:  
15: bottle  
16: chair  
17: dining table  
18: potted plant  
19: sofa  
20: tv/monitor

Comment prédire le nombre d'objets et pour chaque objet sa boîte et sa classe ?

Exemple de solution : CenterNet (Zhou et. al, Objects as points, 2019)



keypoint heatmap [C]



local offset [2]

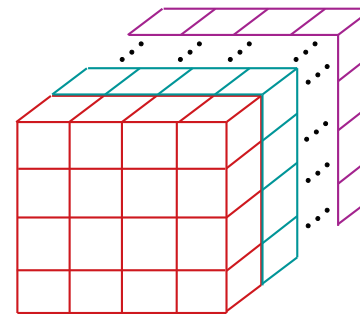


object size [2]

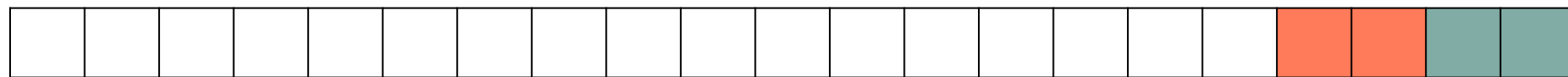
# CenterNet



$3 \times H \times W$



$(C+2+2) \times H/4 \times W/4$

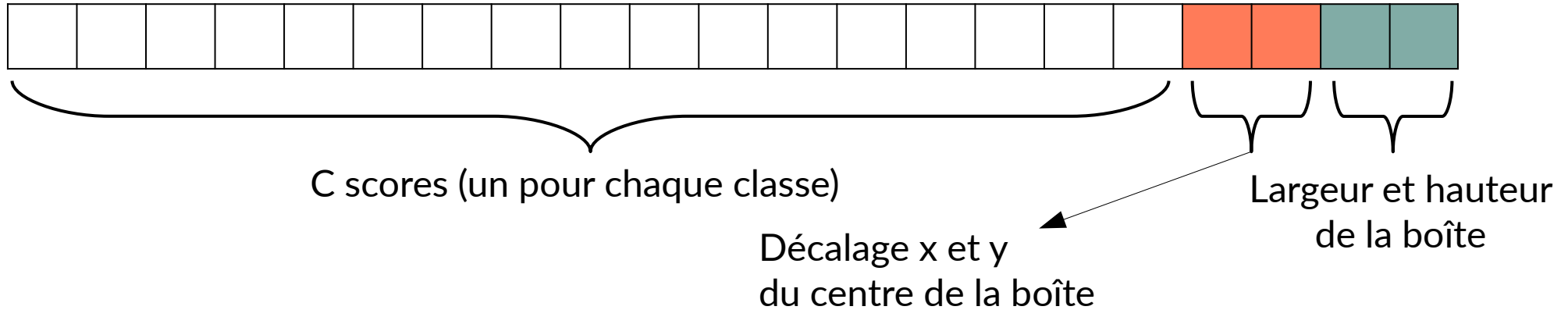


C scores (un pour chaque classe)

Décalage x et y  
du centre de la boîte

Largeur et hauteur  
de la boîte<sub>109</sub>

# CenterNet (suite)



Fonction de coût = somme de trois fonctions de coût

- Pour les scores : « pixelwise logistic regression » (i.e. sur chaque case grise)
- Pour le décalage : régression L1 (sur les cases oranges s'il y a une boîte sinon rien)
- Pour la largeur et la hauteur : régression L1 (sur les cases vertes s'il y a une boîte sinon rien)

# CenterNet (suite)

