

Introduction aux réseaux de neurones pour l'apprentissage supervisé (suite)

Guillaume Bourmaud

PLAN

I. Introduction

II. Apprentissage supervisé

III. Approches paramétriques

IV. Réseaux de neurones

V. Risques

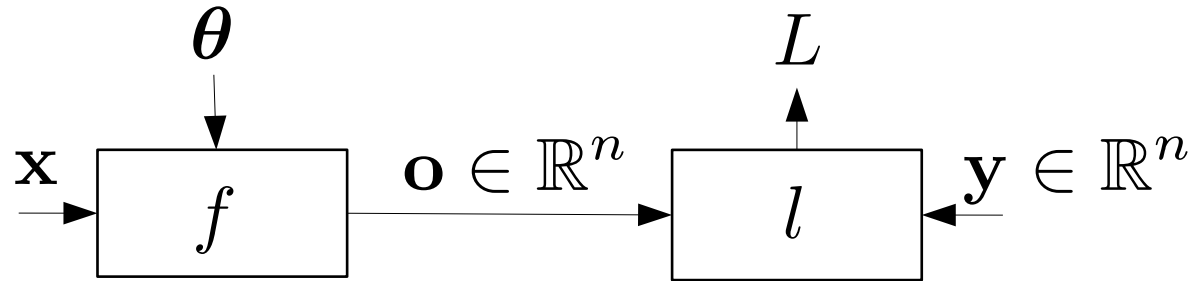
VI. Apprentissage des paramètres d'un réseau de neurones

VII. Réseaux de neurones à convolution

VIII. Réseaux de neurones profonds

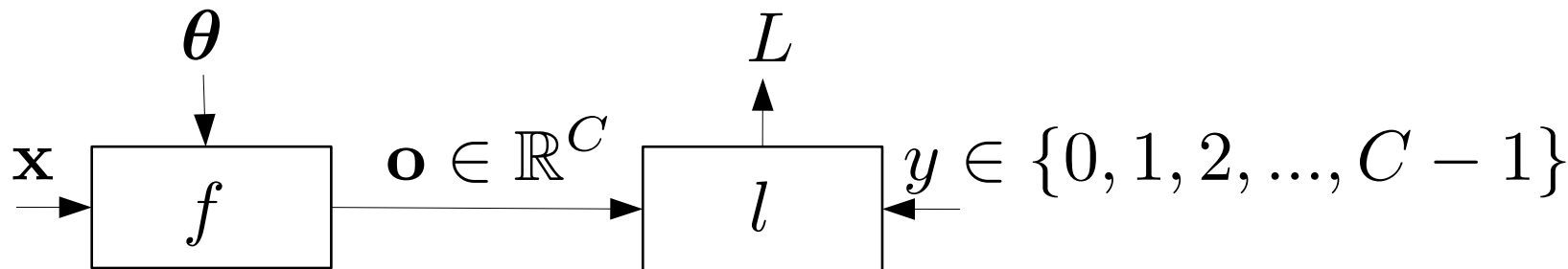
VI) Apprentissage des paramètres d'un réseau de neurones

Choix du coût $l(\mathbf{y}, \mathbf{o})$: Régression

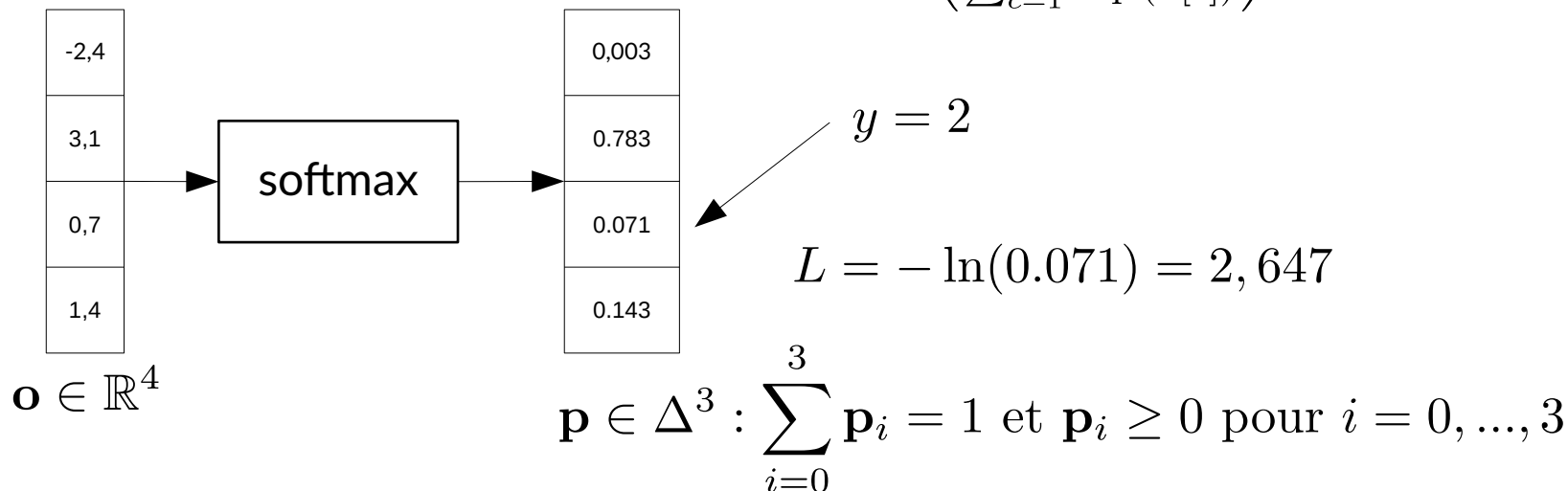


- Erreur quadratique $\|\mathbf{y} - \mathbf{o}\|_2^2 = \sum_{i=0}^{n-1} (\mathbf{y}[i] - \mathbf{o}[i])^2$
- Somme des valeurs absolues $\|\mathbf{y} - \mathbf{o}\|_1 = \sum_{i=0}^{n-1} |\mathbf{y}[i] - \mathbf{o}[i]|$

Choix du coût $l(\mathbf{y}, \mathbf{o})$: Classification



- « Cross-entropy » $-\log(\text{softmax}(\mathbf{o})[y]) = -\ln \left(\frac{\exp(\mathbf{o}[y])}{\sum_{c=1}^C \exp(\mathbf{o}[c])} \right)$

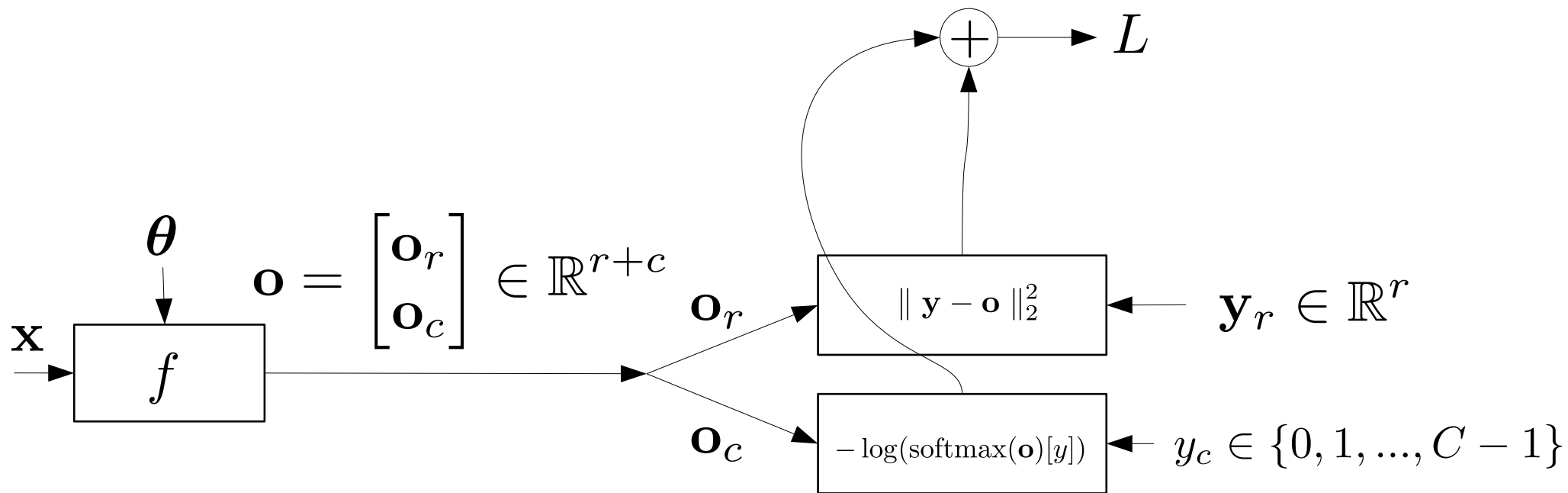


- Autre exemple de coût : Coût de Hinge

VI)

Choix du coût $l(\mathbf{y}, \mathbf{o})$: Apprentissage de métrique

Choix du coût $l(\mathbf{y}, \mathbf{o})$: Combinaison de coûts



Exemple : classification et régression conjointe

Optimisation des paramètres

$$\boldsymbol{\theta}^* = \arg \min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^N l(\mathbf{Y}_{\text{train},i}, f(\mathbf{X}_{\text{train},i}; \boldsymbol{\theta}))$$

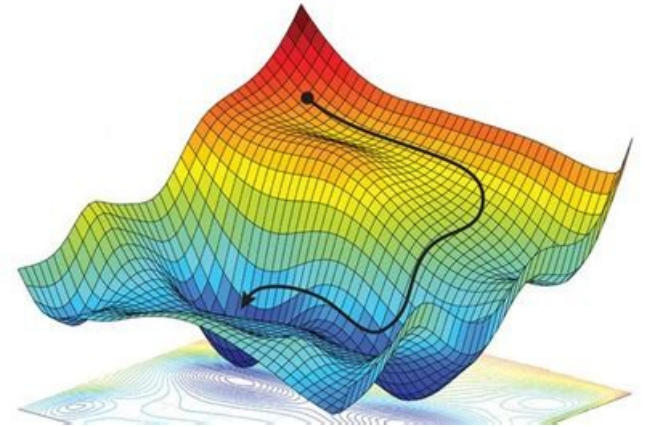
où

$$f(\mathbf{x}; \boldsymbol{\theta}) = f_L \left(f_{L-1} \left(\dots f_2 \left(f_1 \left(\mathbf{x}; \boldsymbol{\theta}^{(1)} \right); \boldsymbol{\theta}^{(2)} \right) \dots; \boldsymbol{\theta}^{(L-1)} \right); \boldsymbol{\theta}^{(L)} \right)$$

Descente de gradient

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \left. \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

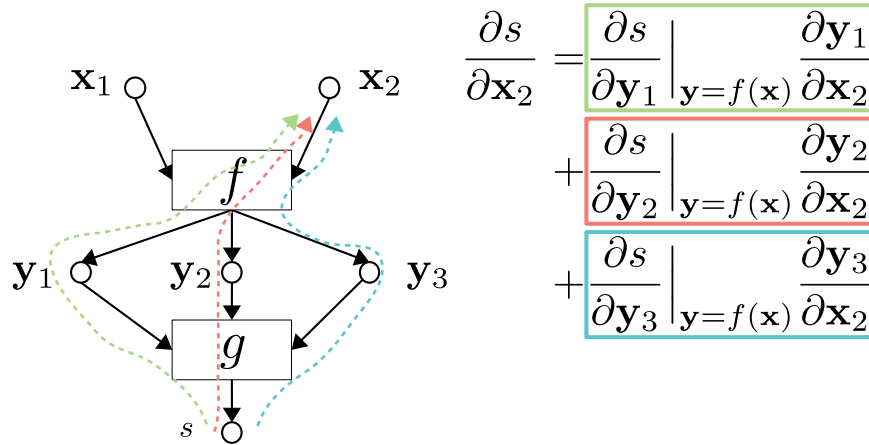
Pas d'apprentissage (« learning rate » en anglais)



Calcul du gradient

Rappel : Théorème de dérivation des fonctions composées

$$s = g(f(\mathbf{x})) \quad \text{avec} \quad s = g(\mathbf{y}) \quad \text{et} \quad \mathbf{y} = f(\mathbf{x})$$



Exemple

$$\frac{\partial s}{\partial \mathbf{x}_i} = \sum_{j=1}^M \frac{\partial s}{\partial \mathbf{y}_j} \bigg|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_j}{\partial \mathbf{x}_i}$$

Formule : dérivation élément par élément

Calcul du gradient (suite)

Rappel : Théorème de dérivation des fonctions composées

$$s = g(f(\mathbf{x})) \quad \text{avec} \quad s = g(\mathbf{y}) \quad \text{et} \quad \mathbf{y} = f(\mathbf{x})$$

$$\frac{\partial s}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial s}{\partial \mathbf{x}_1} \\ \frac{\partial s}{\partial \mathbf{x}_2} \\ \frac{\partial s}{\partial \mathbf{x}_3} \\ \vdots \end{bmatrix}^\top = \begin{bmatrix} \sum_{j=1}^M \frac{\partial s}{\partial \mathbf{y}_j} \bigg|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_j}{\partial \mathbf{x}_1} \\ \sum_{j=1}^M \frac{\partial s}{\partial \mathbf{y}_j} \bigg|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_j}{\partial \mathbf{x}_2} \\ \sum_{j=1}^M \frac{\partial s}{\partial \mathbf{y}_j} \bigg|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_j}{\partial \mathbf{x}_3} \\ \vdots \end{bmatrix}^\top = \begin{bmatrix} \frac{\partial s}{\partial \mathbf{y}_1} \bigg|_{\mathbf{y}=f(\mathbf{x})} \\ \frac{\partial s}{\partial \mathbf{y}_2} \bigg|_{\mathbf{y}=f(\mathbf{x})} \\ \vdots \end{bmatrix}^\top \begin{bmatrix} \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_2} & \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_3} & \cdots \\ \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_1} & \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_2} & \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_3} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} = \underbrace{\frac{\partial s}{\partial \mathbf{y}} \bigg|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}}_{\doteq \tilde{f}(\mathbf{x}, \frac{\partial s}{\partial \mathbf{y}} \bigg|_{\mathbf{y}=f(\mathbf{x})})}$$

$$\frac{\partial s}{\partial \mathbf{y}} = \begin{bmatrix} \frac{\partial s}{\partial \mathbf{y}_1} \\ \frac{\partial s}{\partial \mathbf{y}_2} \\ \vdots \end{bmatrix}^\top = 1 \cdot \underbrace{\frac{\partial g(\mathbf{y})}{\partial \mathbf{y}}}_{\doteq \tilde{g}(\mathbf{y}, 1)}$$

$$\frac{\partial s}{\partial \mathbf{x}} = \tilde{f}(\mathbf{x}, \tilde{g}(\mathbf{y}, 1))$$

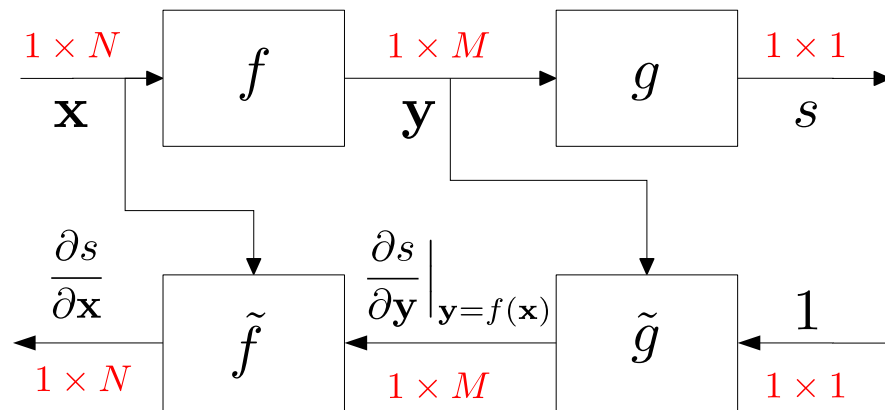
Composition de fonctions

Calcul du gradient (suite)

Rappel : Théorème de dérivation des fonctions composées

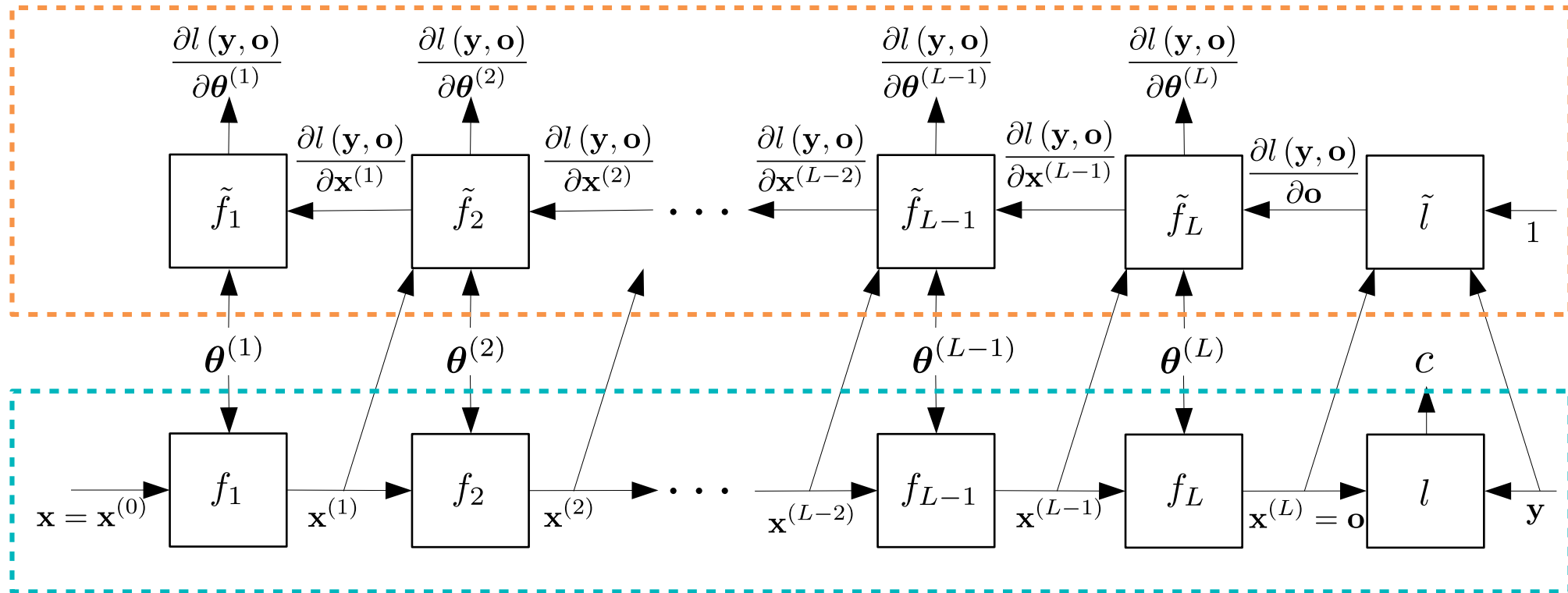
$$s = g(f(\mathbf{x})) \quad \text{avec} \quad s = g(\mathbf{y}) \quad \text{et} \quad \mathbf{y} = f(\mathbf{x})$$

$$\frac{\partial s}{\partial \mathbf{x}} = \tilde{f}(\mathbf{x}, \tilde{g}(\mathbf{y}, 1))$$



Calcul automatique du gradient

Rétropropagation ("Backward")



Propagation avant ("Forward")

"Differentiable Programming"

Initialisation des paramètres

- La méthode la plus utilisée initialise les paramètres des FC aléatoirement (distribution normale ou uniforme).

Exemple Kaiming init.

$$\mathbf{W}_0 = \sqrt{\frac{6}{n_{\text{in}}}} \mathcal{U}_{[-1,1]}(n_{\text{out}}, n_{\text{in}}) \quad \mathbf{b}_0 = \frac{1}{\sqrt{n_{\text{in}}}} \mathcal{U}_{[-1,1]}(n_{\text{out}})$$

He, K., et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." ICCV 2015.

- D'autres méthodes existent mais sont moins utilisées (car la précédente fonctionne bien en pratique).

Mishkin, D., & Matas, J. All you need is a good init. 2015

Apprendre sur une grande base de données annotées

Descente de gradient
(GD) :

$$\theta_{k+1} = \theta_k - \alpha \sum_{i=1}^{N_{\text{train}}} \frac{\partial l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))}{\partial \theta} \Big|_{\theta=\theta_k}$$

Descente de gradient
stochastique (SGD) :

$$\theta_{k+1} = \theta_k - \alpha \sum_{i \in \Omega_k} \frac{\partial l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))}{\partial \theta} \Big|_{\theta=\theta_k}$$

1	2	3	4	5	6	7	8

X_{train}

1	2	3	4	5	6	7	8

Y_{train}

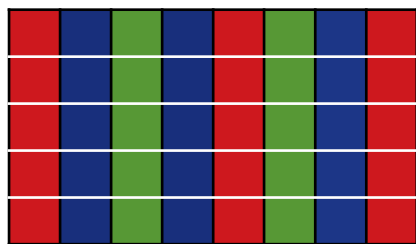
Tirage aléatoire, à chaque itération, de $|\Omega_k|$
éléments dans la base de données

Apprendre sur une grande base de données annotées (suite)

Descente de gradient stochastique (SGD) :

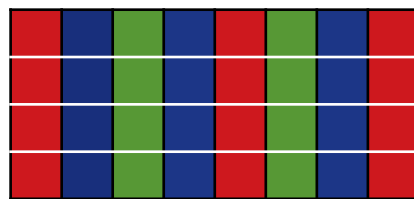
$$\theta_{k+1} = \theta_k - \alpha \sum_{i \in \Omega_k} \frac{\partial l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))}{\partial \theta} \Big|_{\theta = \theta_k}$$

1 2 3 4 5 6 7 8



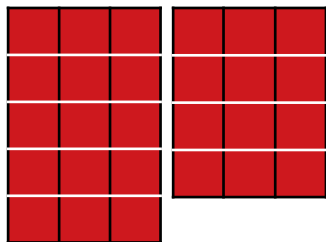
X_{train}

1 2 3 4 5 6 7 8



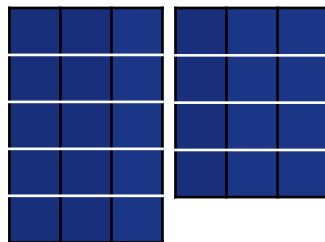
Y_{train}

1 5 8 1 5 8



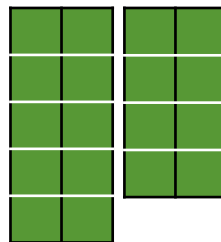
Minibatch 1

2 4 7 2 4 7



Minibatch 2

3 6 3 6



Minibatch 3

Une « epoch » :

- 1) Découper la base de données en « minibatches » de taille $|\Omega_k|$
- 2) Faire une itération de SGD sur chaque « minibatch »
- 3) Fin de l' « epoch », aller à 1)

Avantages et inconvénients de la SGD

Par rapport au gradient de la GD, le gradient de la SGD est « bruité » car calculé sur un « minibatch ».

Inconvénient : l'apprentissage peut être compliqué/lent si le « bruit » est trop important (exemple : taille du « minibatch » trop faible)

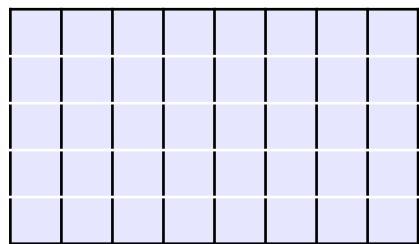
Avantage 1 : ce « bruit » peut permettre de sortir ou d'éviter de mauvais minima locaux

Avantage 2 : le gradient est très rapide à calculer

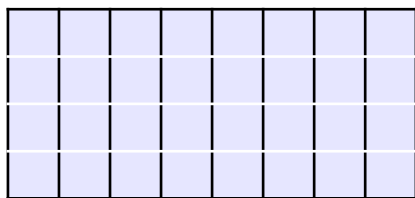
Avantage 3 (empirique) : l'utilisation de petits « minibatches » (32-512) conduit à une bien meilleure généralisation que l'utilisation de grands « minibatches »

VI)

Apprendre à « bien » prédire... sur de nouvelles données

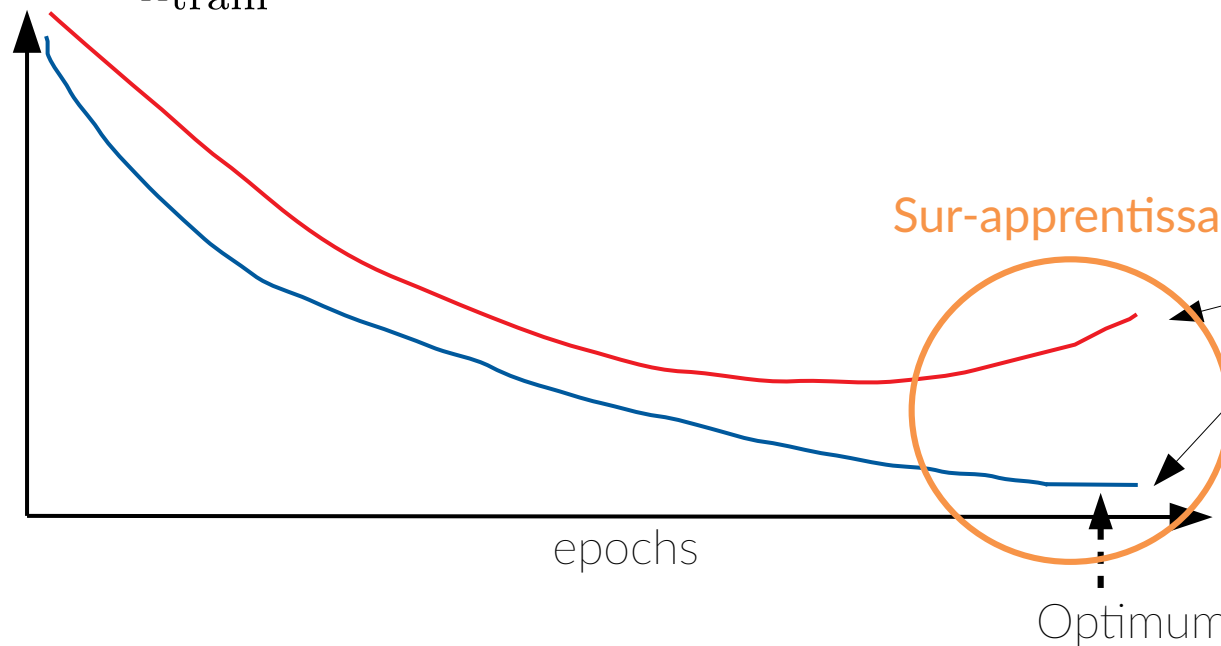


$\mathbf{X}_{\text{train}}$



$\mathbf{Y}_{\text{train}}$

$$\theta^* = \arg \min_{\theta} L_{\text{train}}(\theta)$$



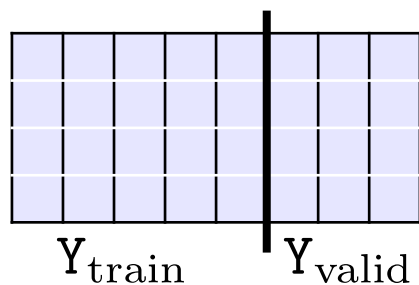
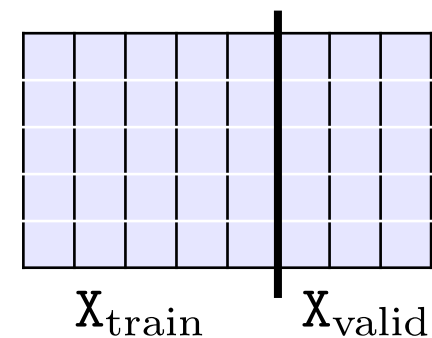
Sur-apprentissage

$$l(y_{\text{new}}, f(\mathbf{x}_{\text{new}}; \theta))$$

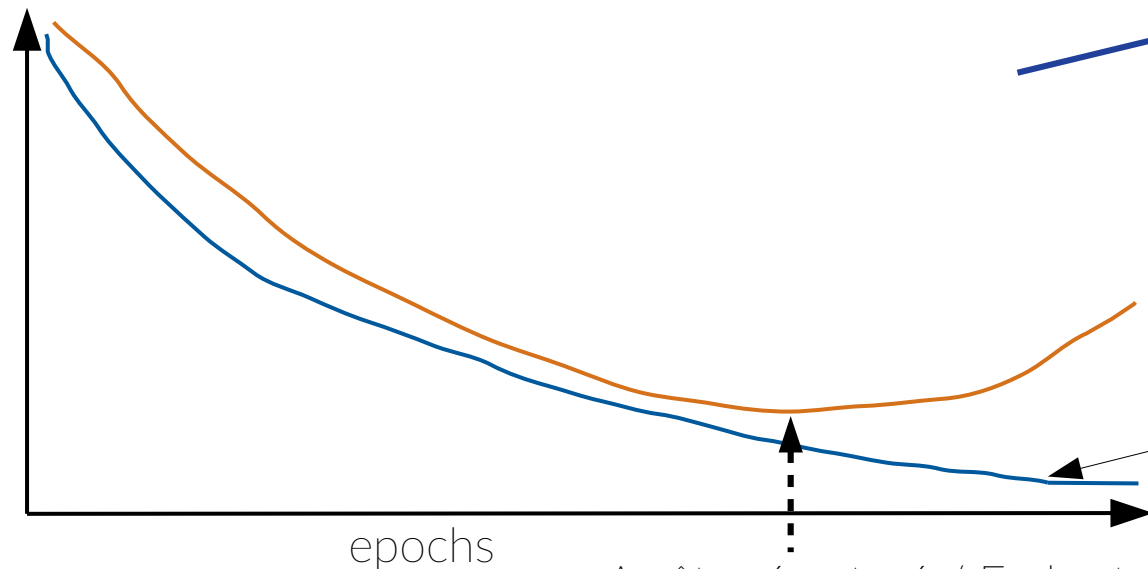
$$L_{\text{train}}(\theta) = \sum_{i=1}^{N_{\text{train}}} l(\mathbf{Y}_{\text{train},i}, f(\mathbf{X}_{\text{train},i}; \theta))$$

VI)

Apprendre à « bien » prédire... sur de nouvelles données (suite)



$$\theta^* = \arg \min_{\theta} L_{\text{train}}(\theta)$$

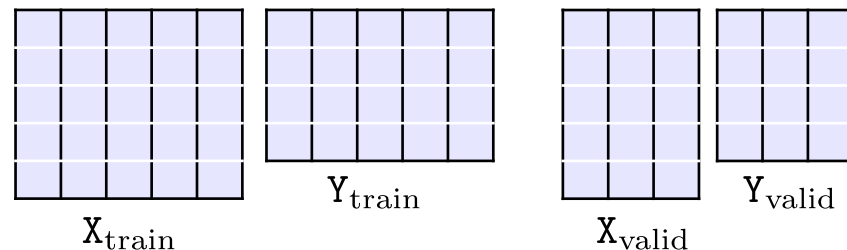


$$L_{\text{valid}}(\theta) = \sum_{i=1}^{N_{\text{valid}}} l(Y_{\text{valid},i}, f(X_{\text{valid},i}; \theta))$$

$$L_{\text{train}}(\theta) = \sum_{i=1}^{N_{\text{train}}} l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))$$

Résumé de l'étape d'apprentissage

- 1) Découper **une fois pour toutes** la base de données en
une base d'apprentissage (« training set »)
une base de validation (« validation set »)



- 2) Lancer une descente de gradient stochastique (SGD) avec arrêt prématuré

Au début d'une « epoch », découper la base d'apprentissage aléatoirement en « minibatches »

Faire une itération de SGD sur chaque « minibatch » : $\theta_{k+1} = \theta_k - \alpha \sum_{i \in \Omega_k} \frac{\partial l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))}{\partial \theta} \Big|_{\theta = \theta_k}$

A la fin d'une « epoch », calculer $L_{\text{valid}}(\theta) = \sum_{i=1}^{N_{\text{valid}}} l(Y_{\text{valid},i}, f(X_{\text{valid},i}; \theta))$

Stocker la valeur actuelle de θ si le coût de validation est plus faible que le précédent meilleur coût

Stopper l'entraînement lorsqu'on est en régime de « sur-apprentissage »

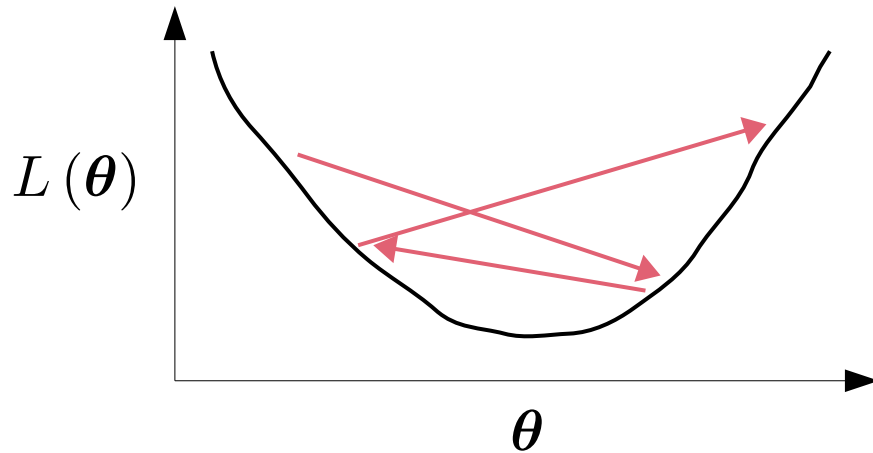
Bonnes pratiques

- Lancer un apprentissage sur un seul « minibatch » jusqu'à obtention d'un coût d'apprentissage de zéro
- Visualiser tout ce qu'il est possible de visualiser
 - Entrées → plage de valeurs (erreur classique : les données ne sont pas normalisées)
 - Sorties
 - Valeurs des paramètres
 - Valeur du pas d'apprentissage
 - Coûts (d'apprentissage, de validation, ...)
 - Gradients
 - ...

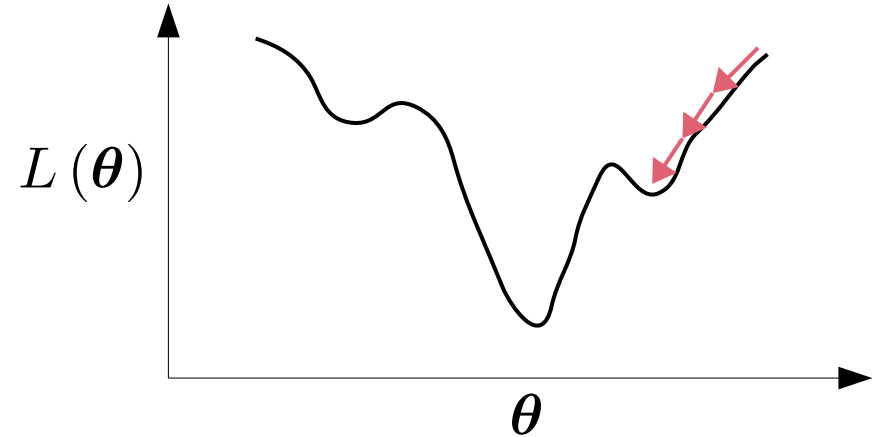
Définir la valeur du pas d'apprentissage

$$\theta_{k+1} = \theta_k - \alpha \sum_{i \in \Omega_k} \frac{\partial l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))}{\partial \theta} \Big|_{\theta=\theta_k}$$

Pas d'apprentissage



Pas d'apprentissage trop grand



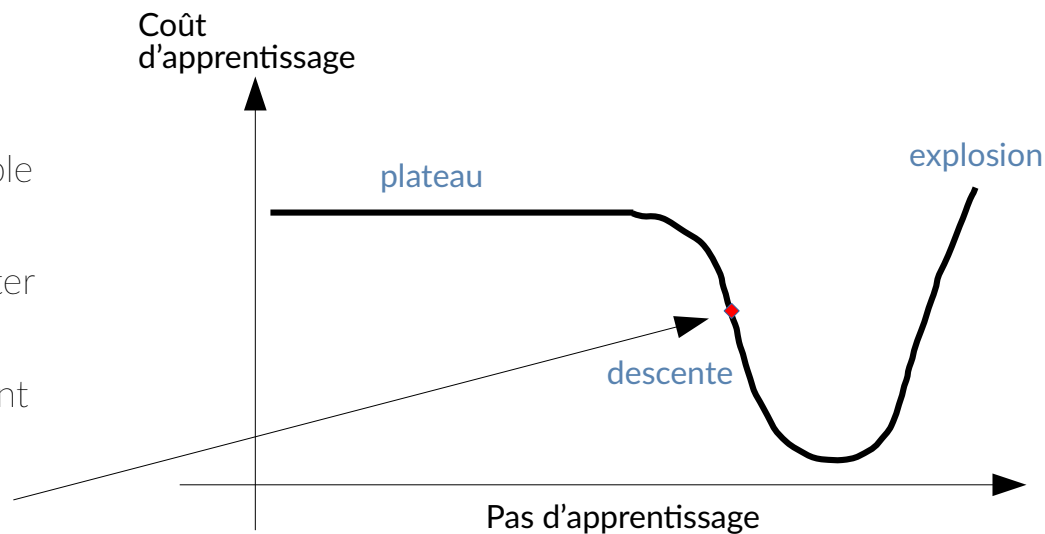
Pas d'apprentissage trop petit

Définir la valeur du pas d'apprentissage (suite)

Solution 1 (la plus utilisée) : Tester différentes valeurs du pas d'apprentissage en visualisant à chaque fois l'évolution du coût d'apprentissage (et du coût de validation)

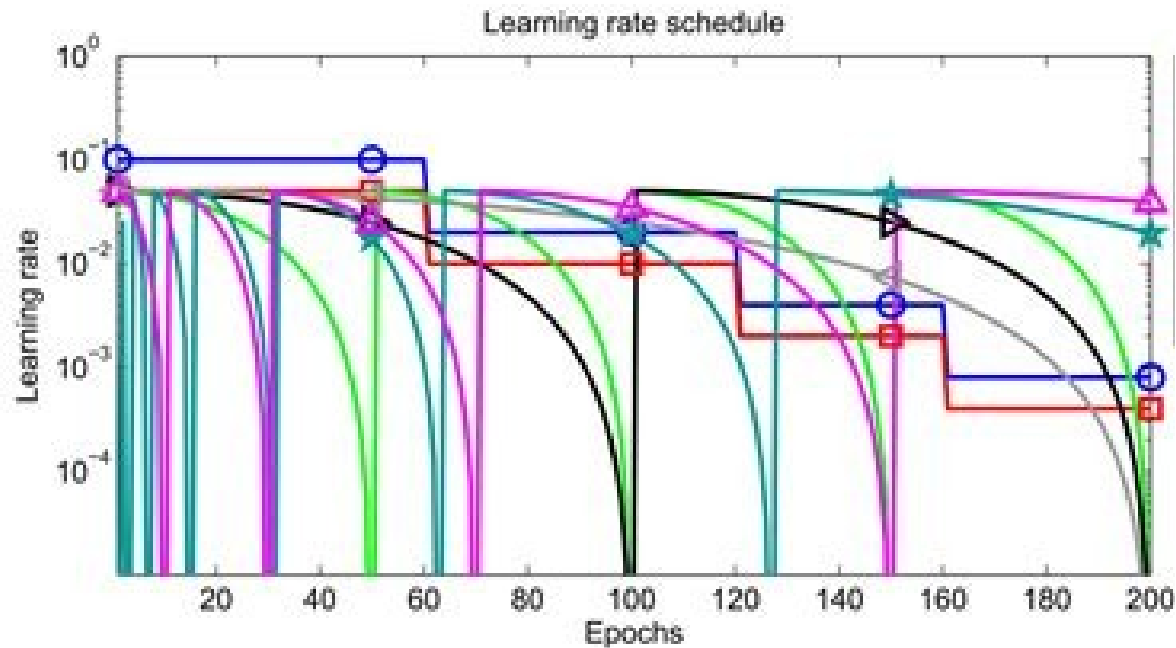
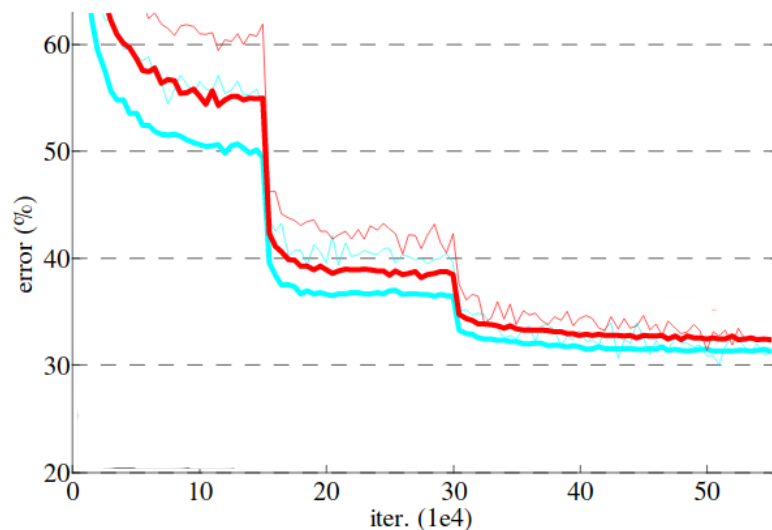
Solution 2 (rarement utilisée) :

- Lancer un entraînement en partant d'un pas très faible (e.g. $1e-7$).
- A chaque itération (i.e à chaque minibatch), augmenter le pas.
- Récupérer la valeur du pas correspondant au gradient le plus négatif.



Evolution du pas d'apprentissage durant l'optimisation

- Constant
- Décroissant
- Cyclique
- Réduction sur plateau



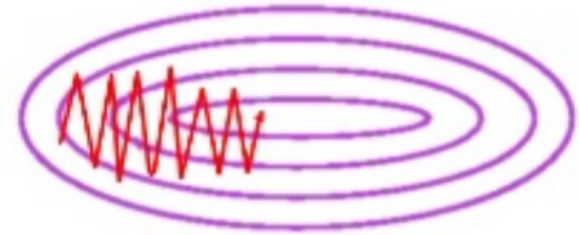
Loshchilov, I., & Hutter, F. "SGDR: Stochastic gradient descent with warm restarts." 2016

SGD avec moment (« SGD with momentum »)

$$\mathbf{g}_{k+1} = \sum_{i \in \Omega_k} \frac{\partial l(\mathbf{y}_{\text{train},i}, f(\mathbf{x}_{\text{train},i}; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta} = \boldsymbol{\theta}_k}$$

SGD

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \mathbf{g}_{k+1}$$

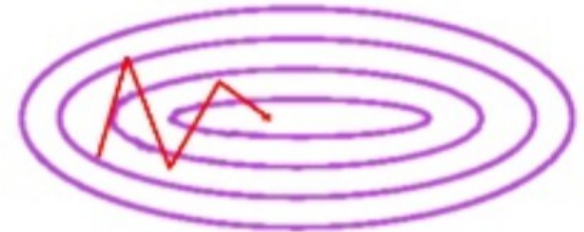


SGD avec moment

$$\mathbf{g}_{k+1} = \sum_{i \in \Omega_k} \frac{\partial l(\mathbf{y}_{\text{train},i}, f(\mathbf{x}_{\text{train},i}; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta} = \boldsymbol{\theta}_k}$$

$$\mathbf{m}_{k+1} = \beta \mathbf{m}_k + (1 - \beta) \mathbf{g}_{k+1}$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \mathbf{m}_{k+1}$$



« Adam: A Method for Stochastic Optimization »

$$\mathbf{g}_{k+1} = \sum_{i \in \Omega_k} \frac{\partial l(\mathbf{Y}_{\text{train},i}, f(\mathbf{X}_{\text{train},i}; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta} = \boldsymbol{\theta}_k}$$

$$\mathbf{m}_{k+1} = \frac{1}{1 - \beta_1^k} (\beta_1 \mathbf{m}_k + (1 - \beta_1) \mathbf{g}_{k+1})$$

$$\mathbf{v}_{k+1} = \frac{1}{1 - \beta_2^k} (\beta_2 \mathbf{v}_k + (1 - \beta_2) \mathbf{g}_{k+1}^2)$$

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \frac{\mathbf{m}_{k+1}}{\sqrt{\mathbf{v}_{k+1}} + \epsilon}$$

Carré de chaque élément de \mathbf{g}_k

Racine carrée de chaque élément de \mathbf{V}_{k+1}

A priori un tel apprentissage ne devrait **PAS** fonctionner

$$\theta^* = \arg \min_{\theta} L_{\text{train}}(\theta) = \arg \min_{\theta} \sum_{i=1}^{N_{\text{train}}} l(Y_{\text{train},i}, f(X_{\text{train},i}; \theta))$$

$$\text{où } f(\mathbf{x}; \theta) = f_L \left(f_{L-1} \left(\dots f_2 \left(f_1 \left(\mathbf{x}; \theta^{(1)} \right); \theta^{(2)} \right) \dots; \theta^{(L-1)} \right); \theta^{(L)} \right)$$

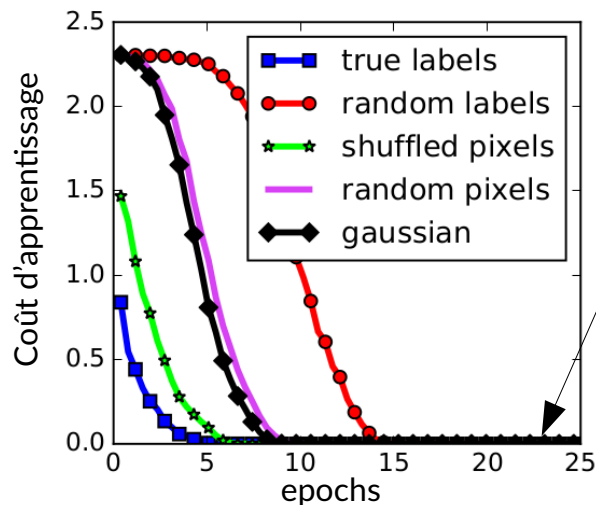
Raisonnement a priori

descente de gradient où
 f est non-convexe



mauvais minimum local

Résultats empiriques sur CIFAR10



Zéro !

Un des minima globaux
a été atteint.

A priori un tel apprentissage ne devrait **PAS** fonctionner (suite)

Raisonnement a priori

Coût d'apprentissage atteint zéro



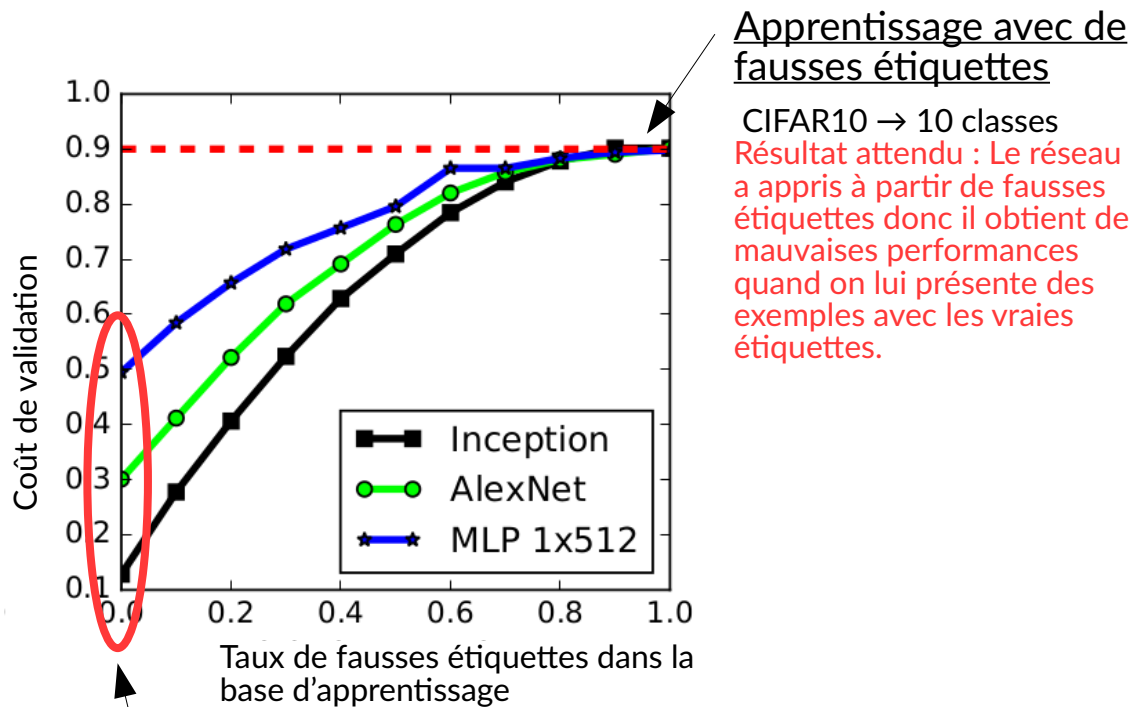
Le réseau a appris « par cœur » à associer la bonne étiquette pour chaque exemple de la base d'apprentissage



Les performances de généralisation seront très mauvaises

Zhang, C et al. (2017). Understanding Deep Learning requires rethinking generalization. ICLR

Résultats empiriques sur CIFAR10



Apprentissage avec les vraies étiquettes

→ Résultat inattendu : Quand le réseau apprend à partir de vraies étiquettes, il a une bonne capacité de généralisation.

A priori un tel apprentissage ne devrait **PAS** fonctionner (suite)

Comment se fait-il que l'étape d'apprentissage ait permis de trouver un minimum global qui généralise bien (sachant qu'il existe des minima globaux qui généralisent mal) ?

Au moins deux hypothèses :

- 1) Biais introduit par l'architecture (« inductive bias »)
- 2) Biais introduit par la descente de gradient stochastique

<https://guillefix.me/nnbias/>

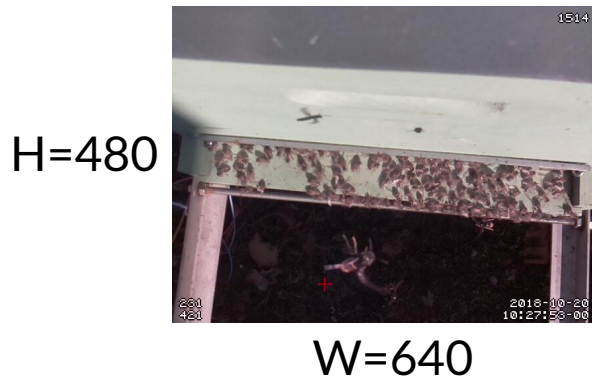
https://hackmd.io/75gt3X6WQbu1_A3pF8svWg

Valle-Pérez. G et al. (2019). Deep learning generalizes because the parameter-function map is biased towards simple functions. ICLR

Smith, S., et al. (2021). On the origin of implicit regularization on stochastic gradient descent. ICLR

VII) Réseaux de neurones à convolution

Limites d'une transformation affine générale (FC)



$$\mathbf{x} : 640 \times 480 \times 3 \approx 10^6$$

Exemple d'une simple transformation affine où la résolution de l'image d'entrée est préservée :

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

Diagram illustrating the dimensions of the variables in the affine transformation equation $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$:

- \mathbf{x} has dimensions $10^6 \times 1$.
- \mathbf{W} has dimensions $10^6 \times 10^6$.
- \mathbf{b} has dimensions $10^6 \times 1$.
- \mathbf{y} has dimensions $10^6 \times 1$.

Occupation mémoire de \mathbf{W} : 4 octets $\times 10^6 \times 10^6 = 4\text{To}$.

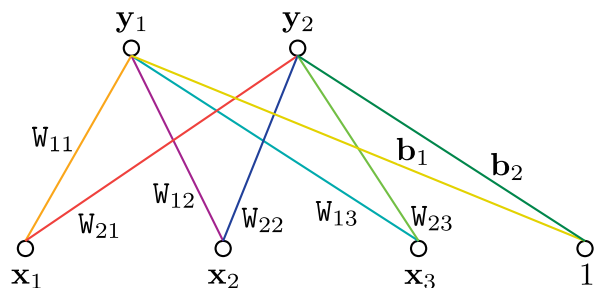
Nombre de « multiplications+additions » également très élevé.

Opération de « convolution » = transformation affine spécifique

“Fully Connected”

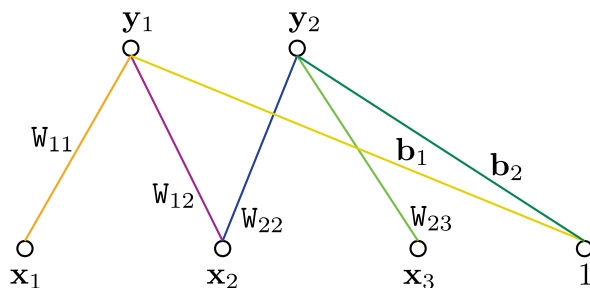
=

Transformation affine générale



$$\begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Localement connecté

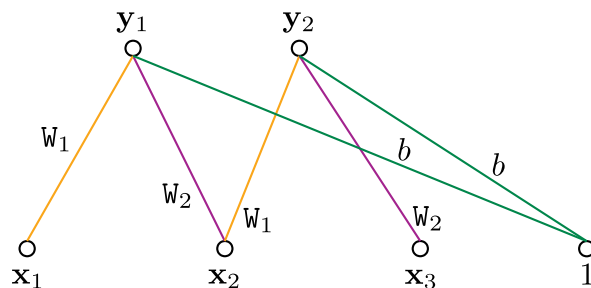


$$\begin{bmatrix} W_{11} & W_{12} & 0 & 0 \\ 0 & W_{22} & W_{23} & 0 \\ 0 & 0 & W_{33} & W_{34} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Équivariance par translation

=

« Convolution »



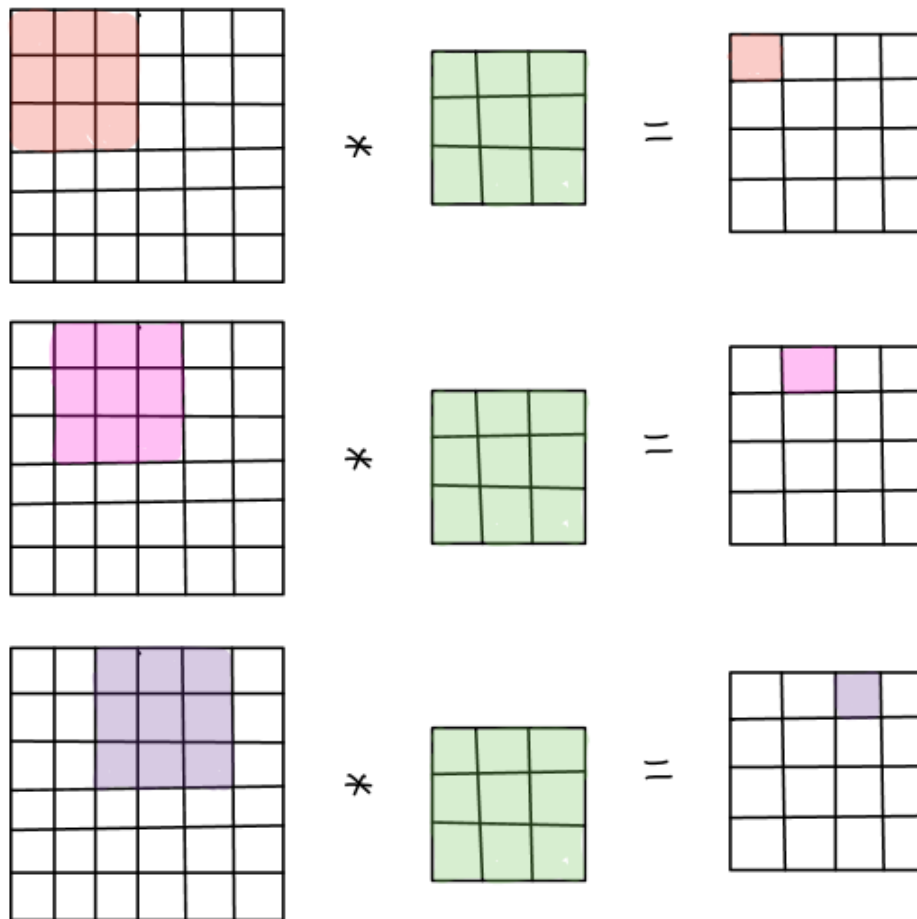
$$\begin{bmatrix} W_{11} & W_{12} & 0 & 0 \\ 0 & W_{11} & W_{12} & 0 \\ 0 & 0 & W_{11} & W_{12} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix}$$

Beaucoup moins de paramètres à stocker

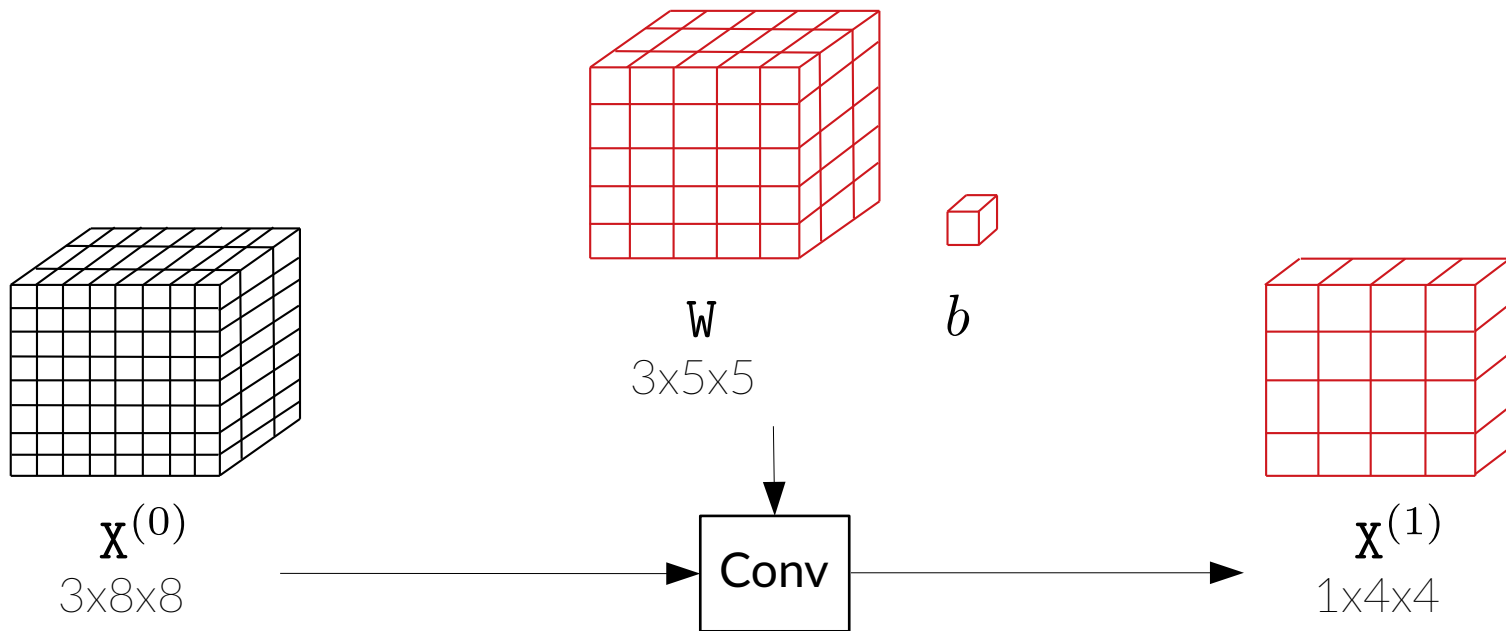
Beaucoup moins de « multiplications+additions »

Opération de « convolution » en 2D

En fait, il s'agit d'une
intercorrélacion



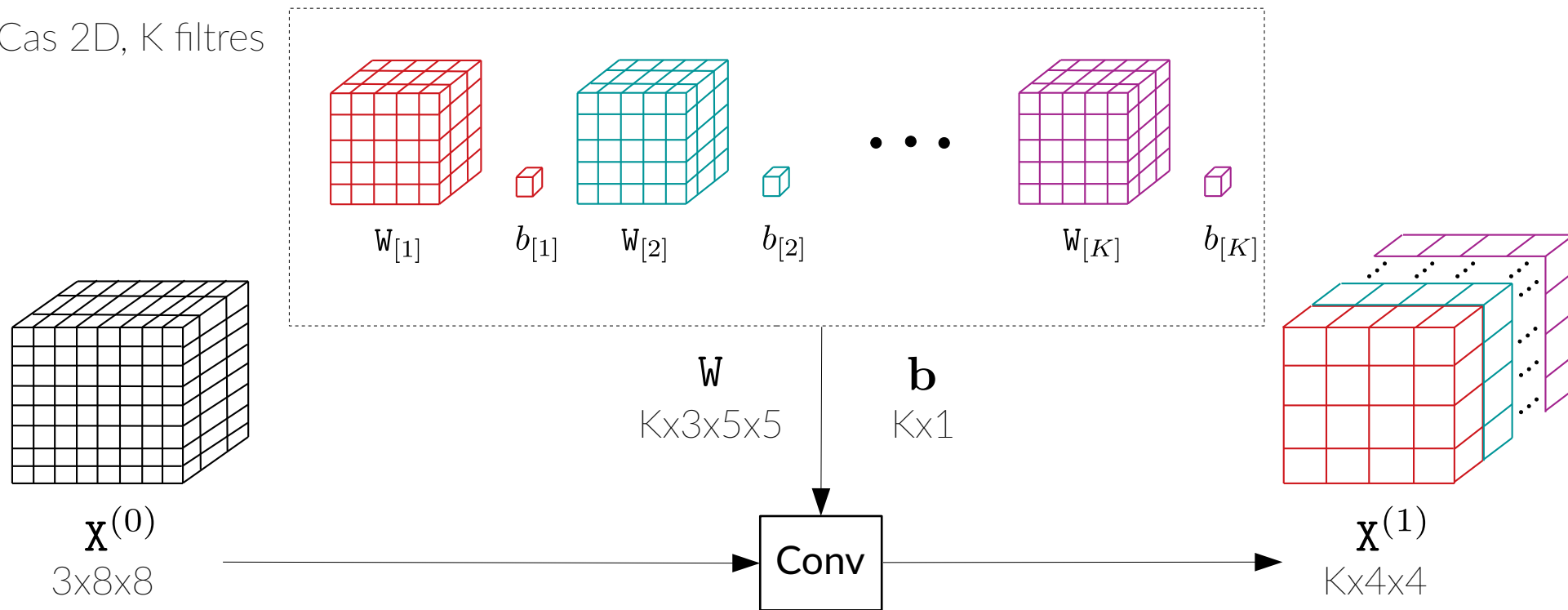
Couche de convolution : un seul filtre



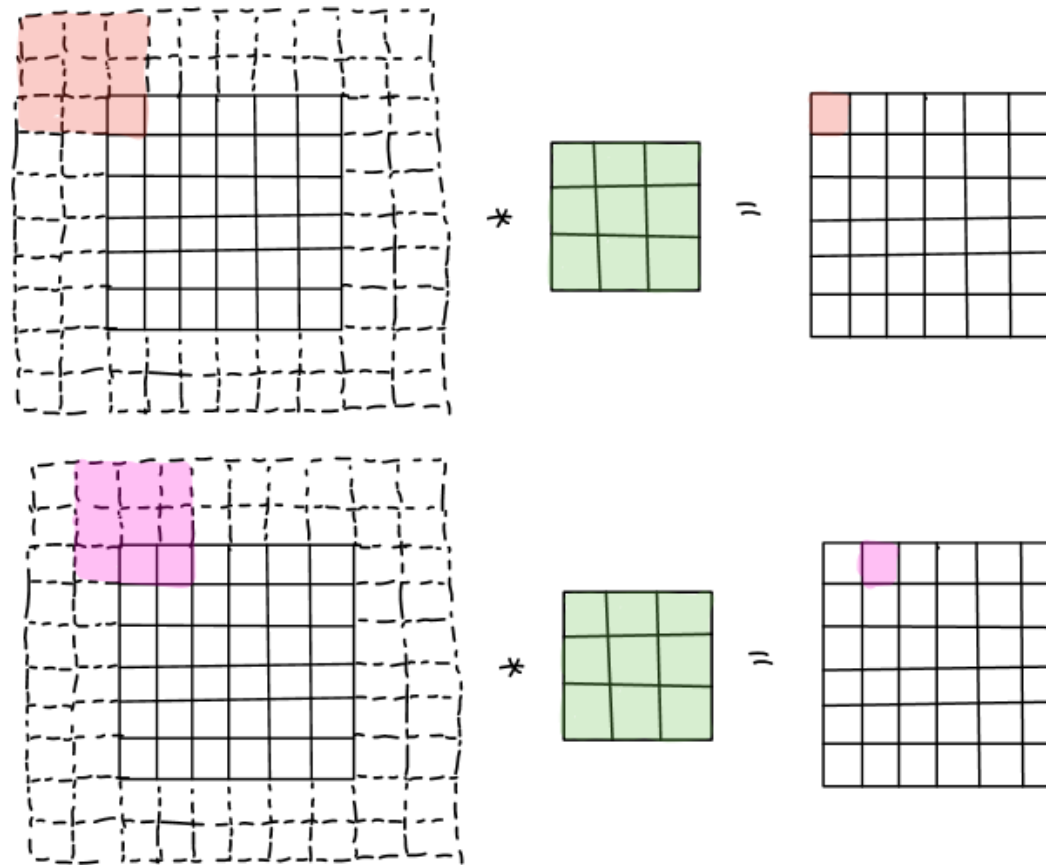
$$X_{i,j}^{(1)} = \sum_{k=0}^2 \sum_{m=0}^4 \sum_{n=0}^4 W_{k,m,n} X_{k,i+m,j+n}^{(0)} + b$$

Couche de convolution : K filtres

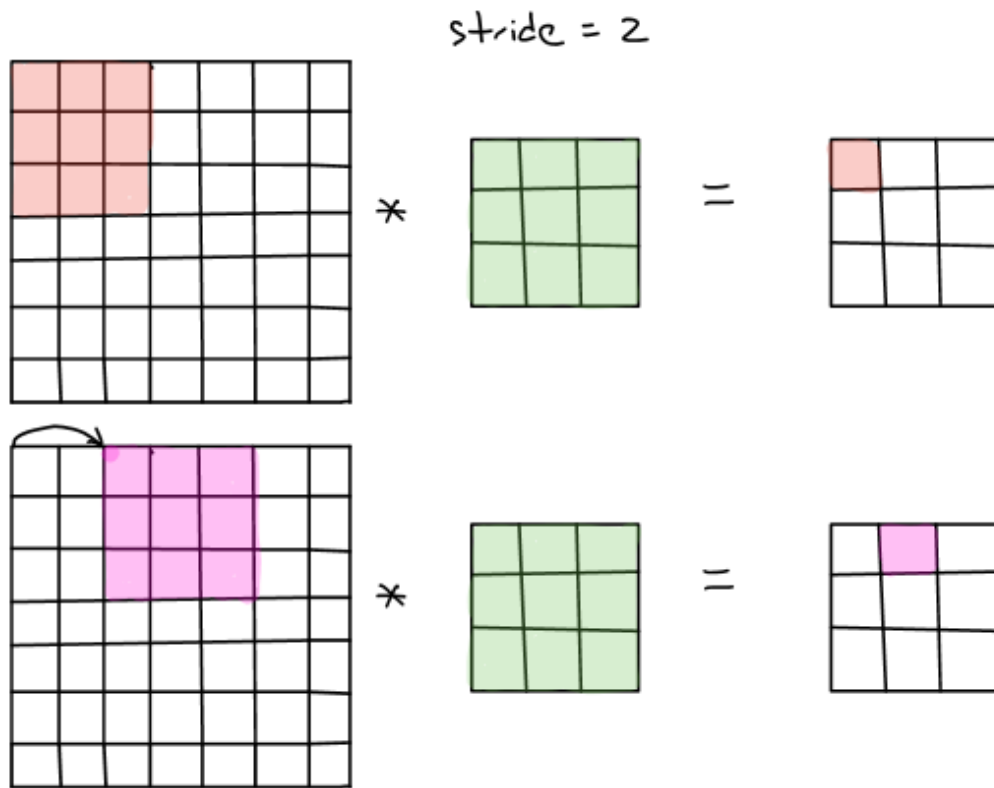
Cas 2D, K filtres



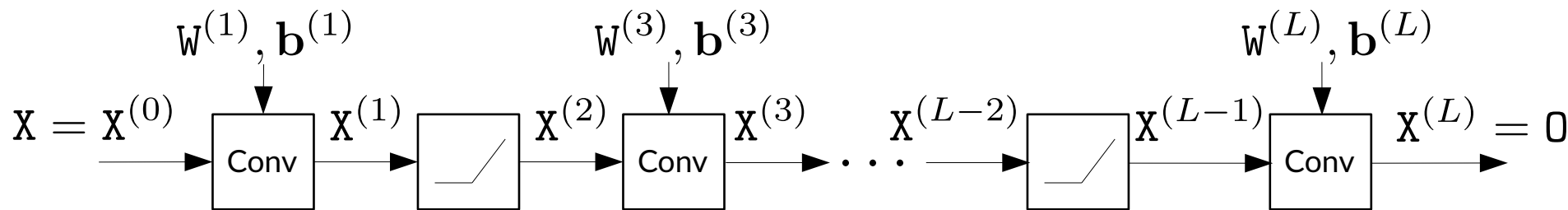
« Zero padding »



« Stride »



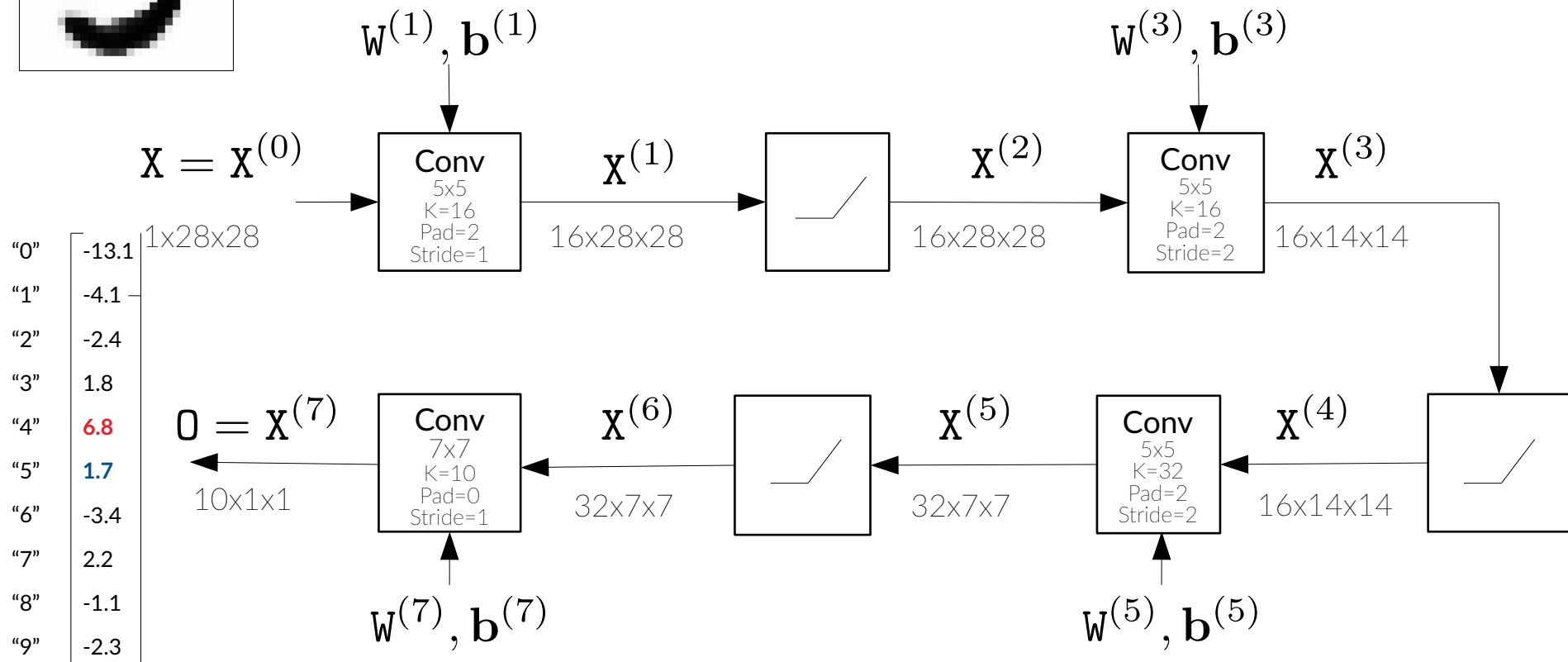
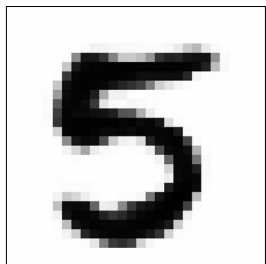
Réseau de neurones à convolution (CNN)



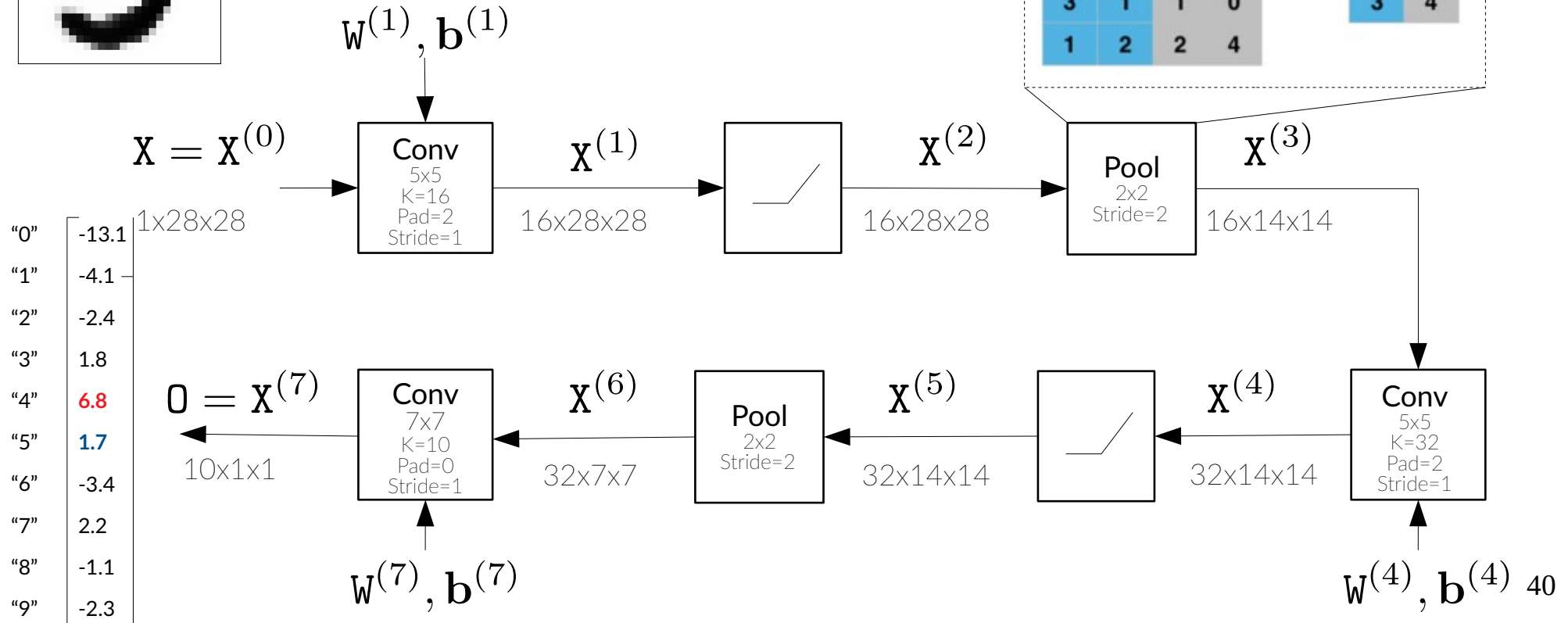
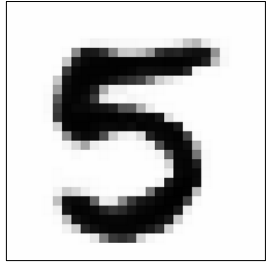
MLP – transformations affines générales + transformations affines spécifiques
=
CNN

→ même initialisation des paramètres que pour le MLP

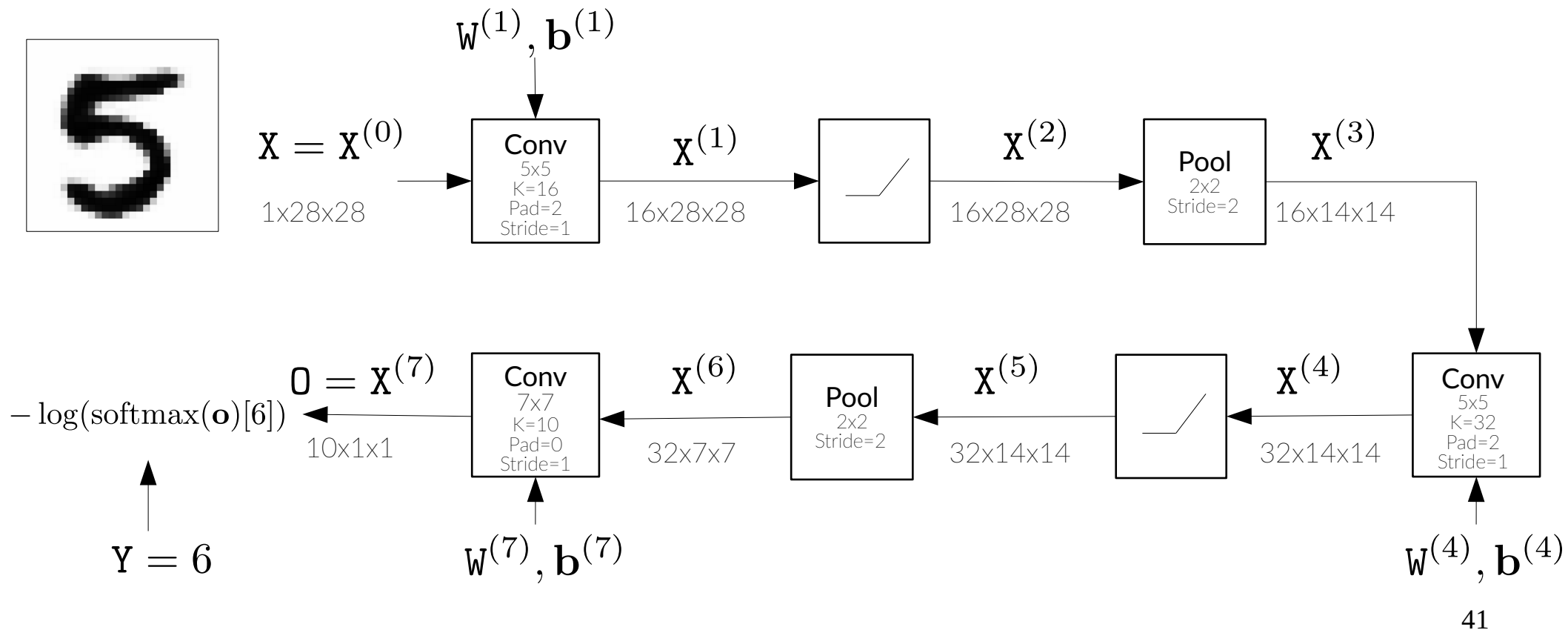
Exemple d'architecture : CNN pour MNIST



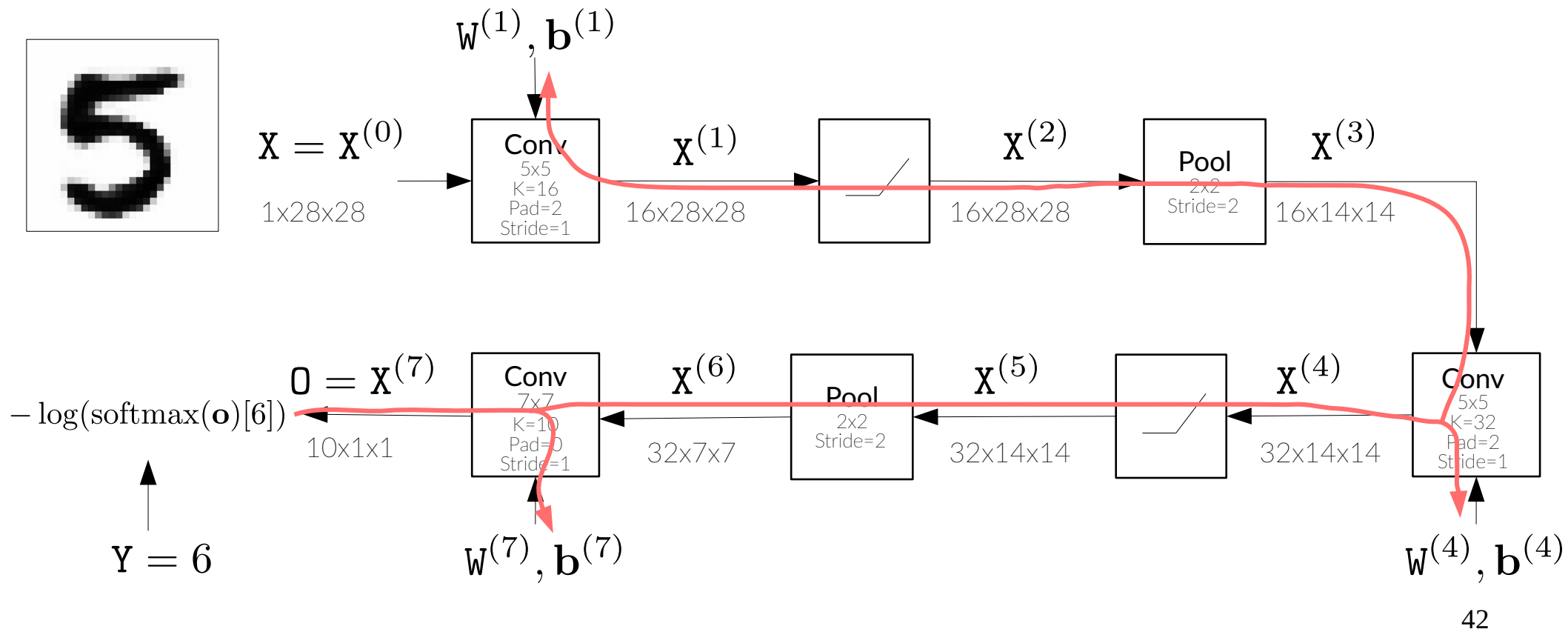
Autre exemple d'architecture : CNN pour MNIST



Apprentissage CNN pour MNIST

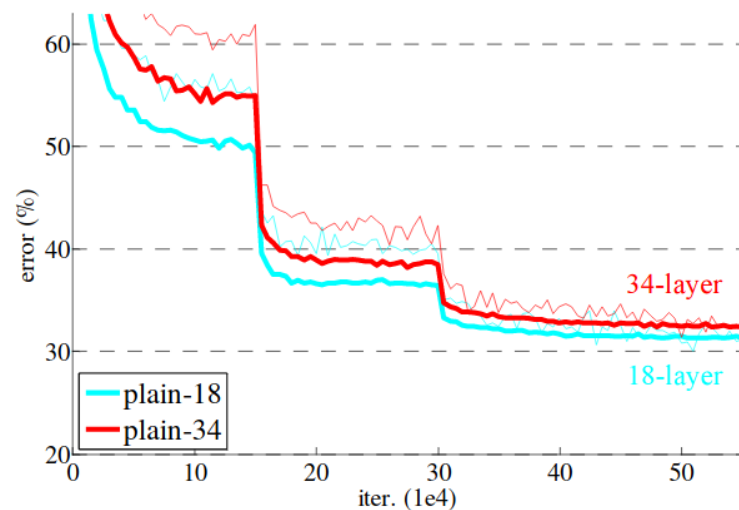


Apprentissage CNN pour MNIST (suite)

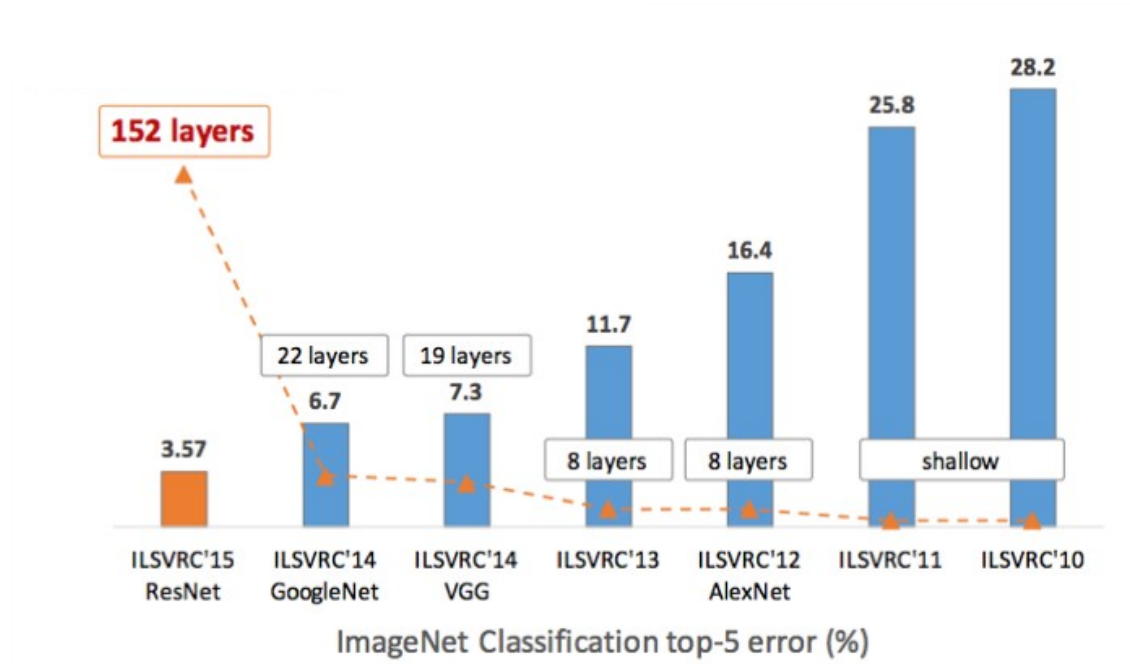


VIII) Réseaux de neurones profonds

Réseaux de neurones profonds

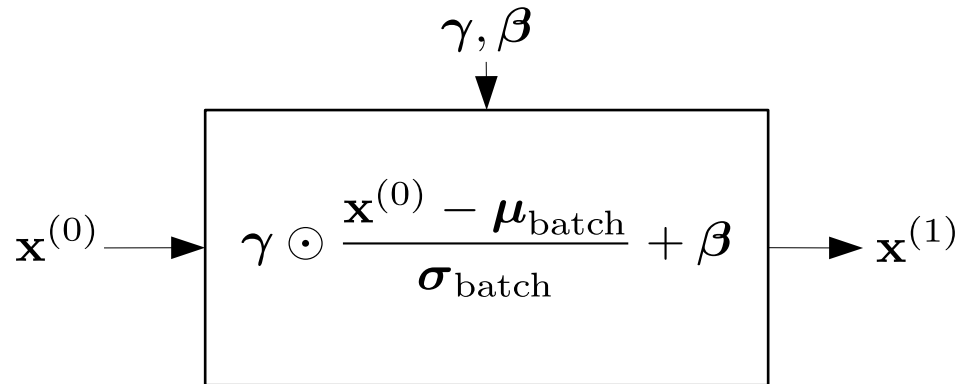


Réseaux de neurones profonds (suite)



Source : <https://medium.com/@Lidinwise/the-revolution-of-depth-fac174924f5>

Couche de “Batch Normalization”



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

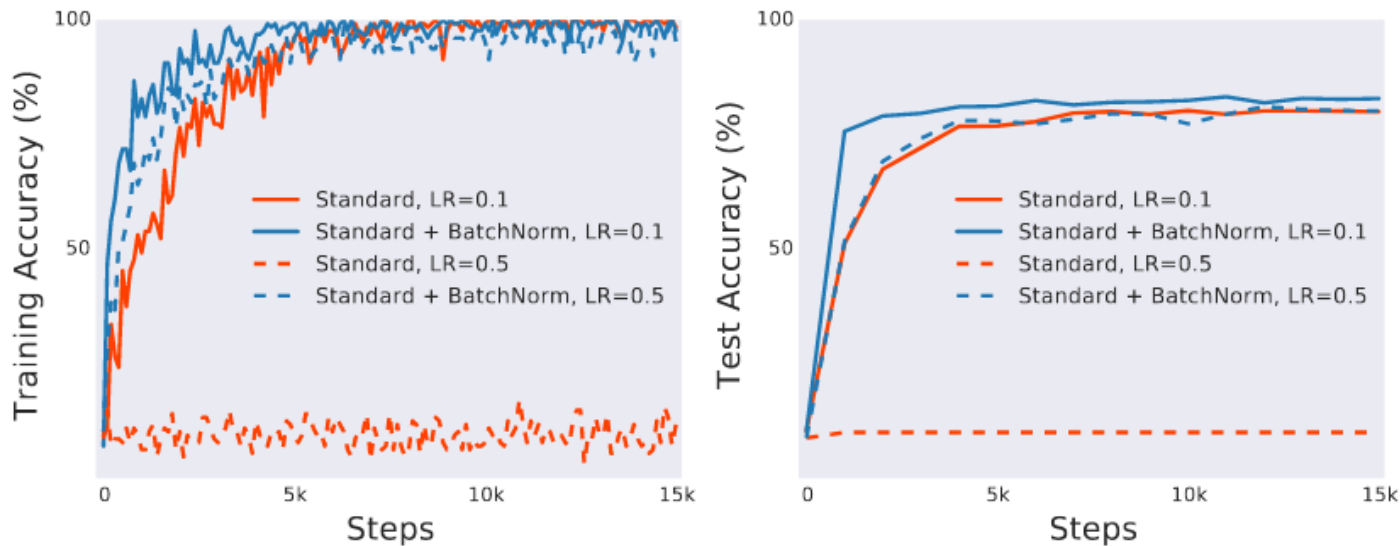
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

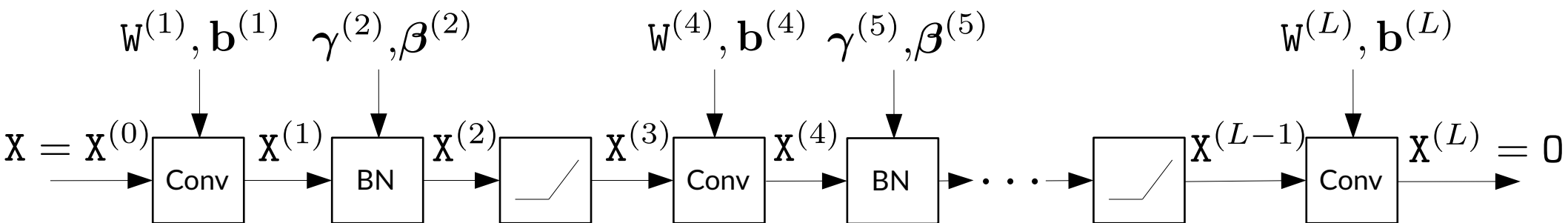
Couche de “Batch Normalization” (suite)



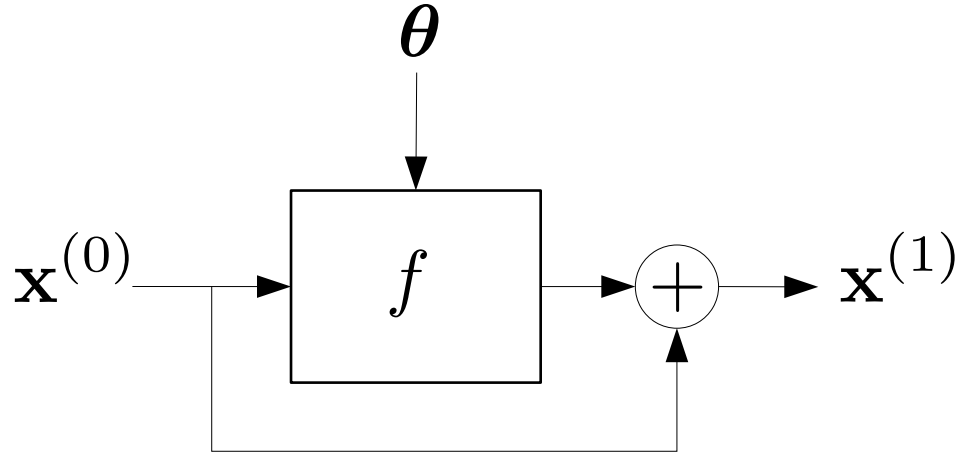
L'utilisation de couches de « batch normalization » rend le problème d'optimisation plus « lisse », ce qui implique :

- Initialisation des paramètres moins critique
- Possibilité d'utilisation un plus grand pas d'apprentissage

Couche de “Batch Normalization” (suite)



Connexion résiduelle

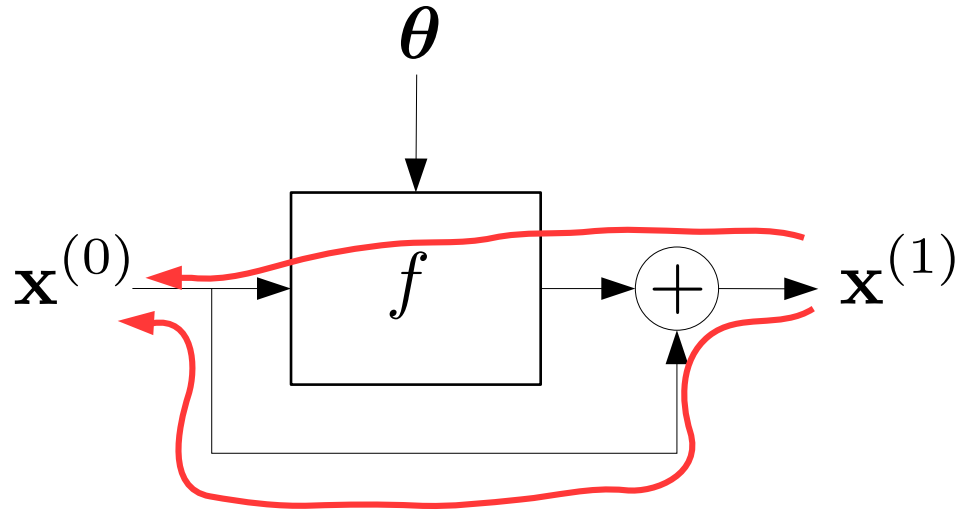


$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + f(\mathbf{x}^{(0)}; \theta)$$

Rend une couche plus « linéaire » donc

- Réduit sa « capacité » → augmentation du nombre de couches et donc de la consommation et de la mémoire pour un même résultat

Connexion résiduelle (suite)

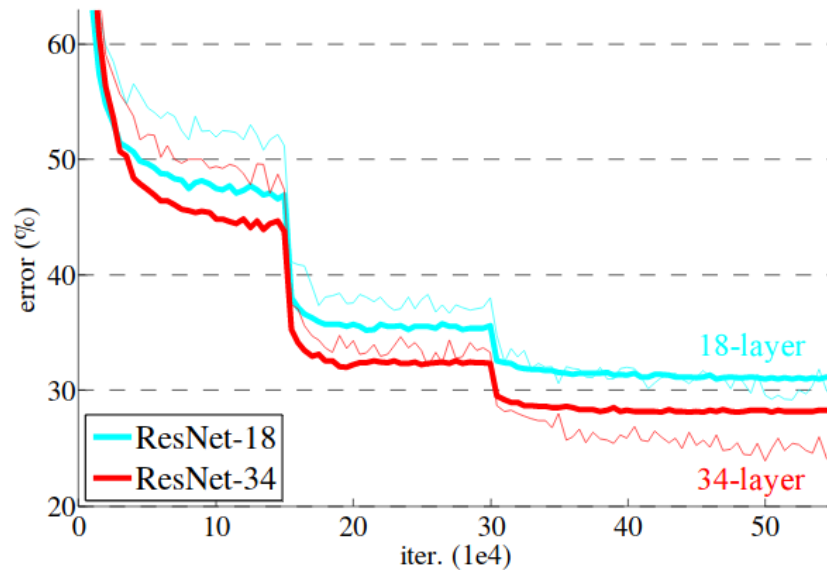


Rend une couche plus « linéaire » donc

- Réduit sa « capacité » \rightarrow augmentation du nombre de couches et donc de la consommation et de la mémoire pour un même résultat
- Mais facilite la propagation du gradient \rightarrow plus de couches conduit à de meilleurs résultats (en théorie)

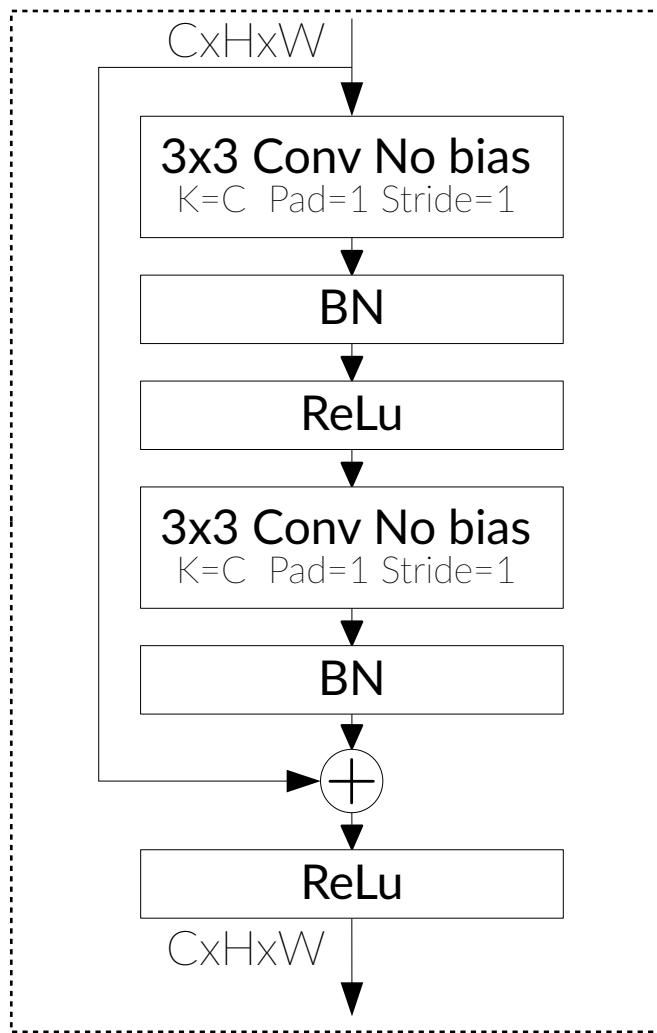
$$\frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{x}^{(0)}} = \mathbf{I} + \frac{\partial f(\mathbf{x}^{(0)}; \boldsymbol{\theta})}{\partial \mathbf{x}^{(0)}}$$

Connexion résiduelle (suite)



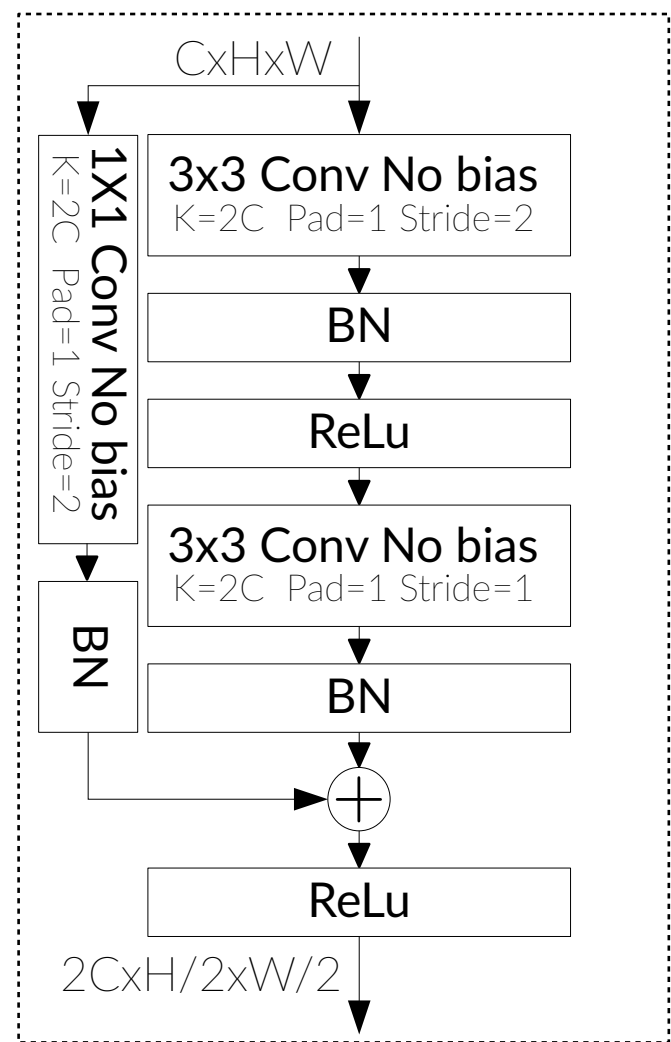
Attention résultat
obtenu en combinant
« batch norm » et
« residual connection »

VIII)

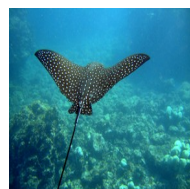


ResBlock A

ResNet



ResBlock B

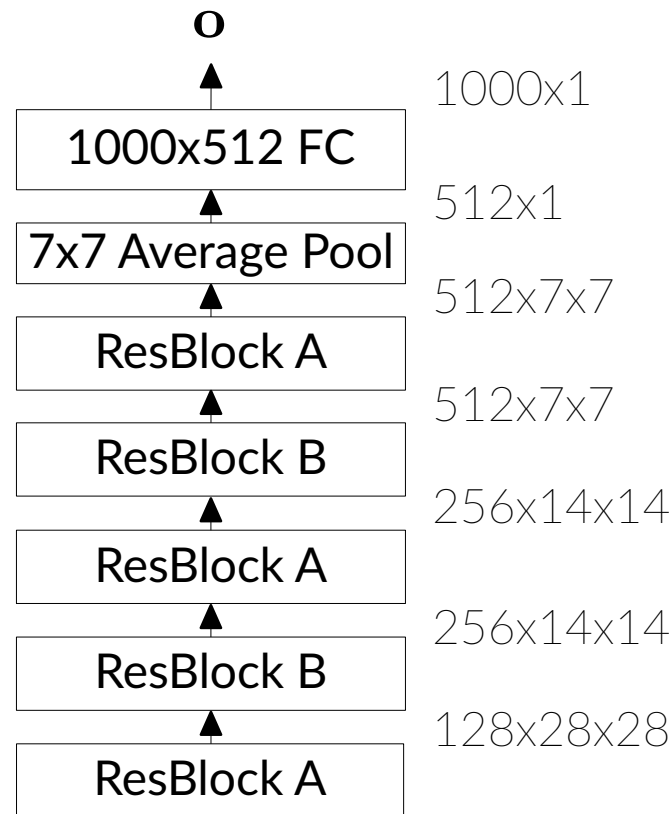
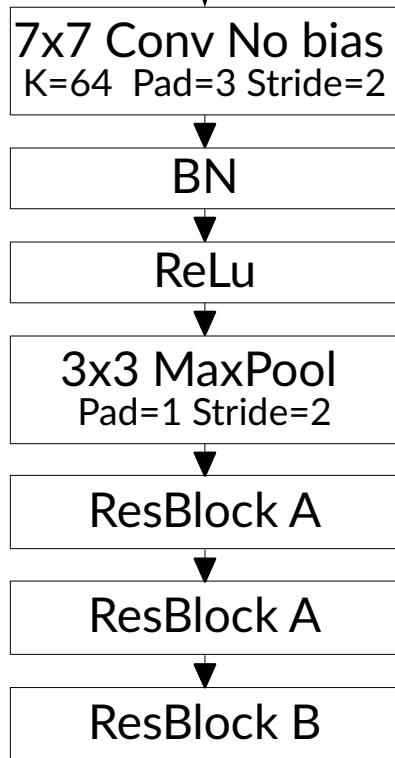


ResNet 18

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

Passage
en grande
dimension

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



Précision vs Nombre de paramètres vs Nombre d'opérations

