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

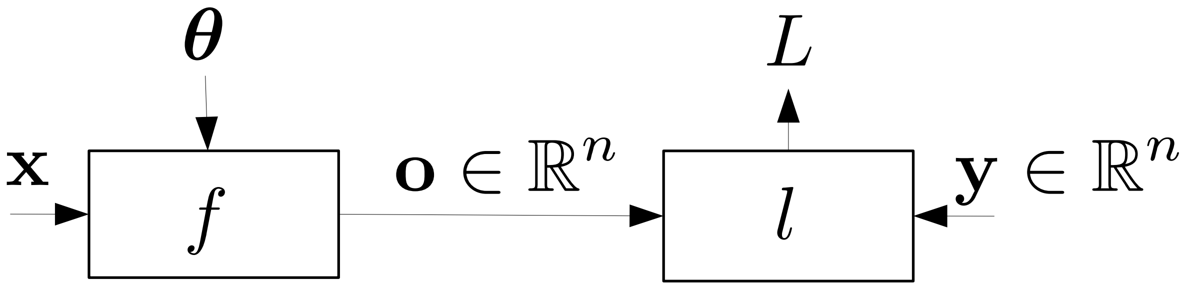
Guillaume Bourmaud

PLAN

VI) Apprentissage des paramètres d’un réseau de neurones

VI)

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

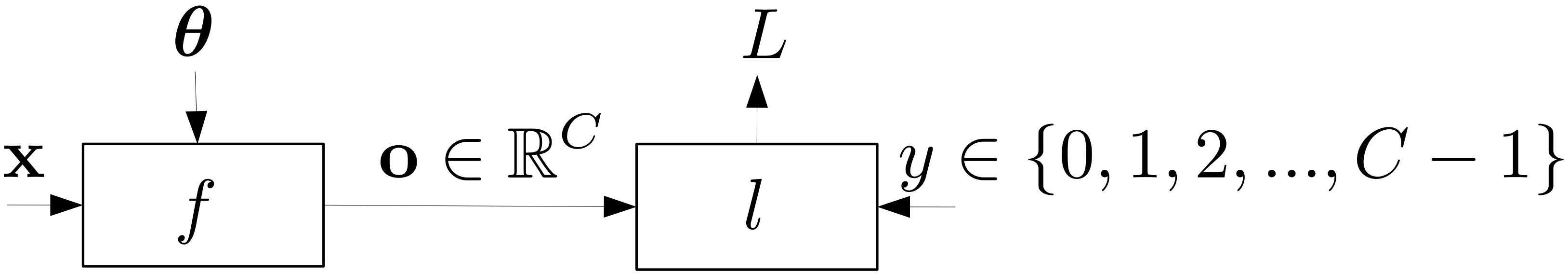


- Erreur quadratique $\parallel \mathbf{y} - \mathbf{o} \parallel_2^2 = \sum_{i=0}^{n-1} (\mathbf{y}[i] - \mathbf{o}[i])^2$

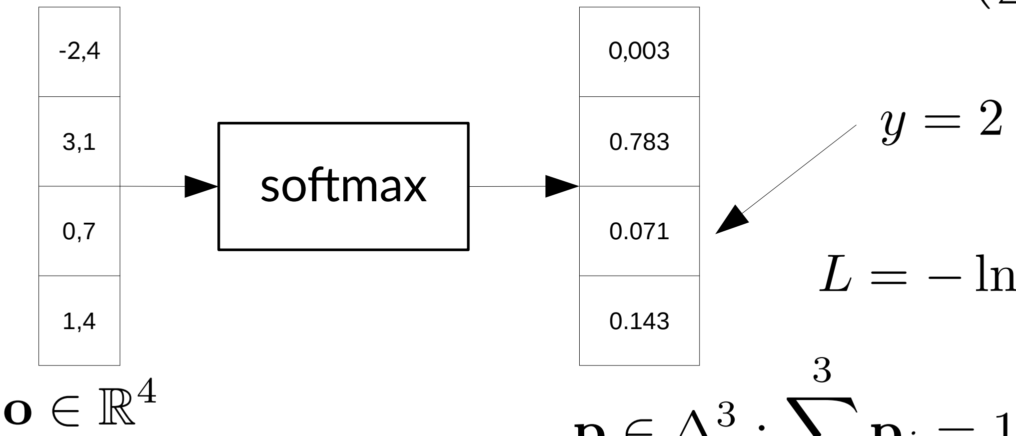
- Somme des valeurs absolues $\parallel \mathbf{y} - \mathbf{o} \parallel_1 = \sum_{i=0}^{n-1} |\mathbf{y}[i] - \mathbf{o}[i]|$

VI)

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



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



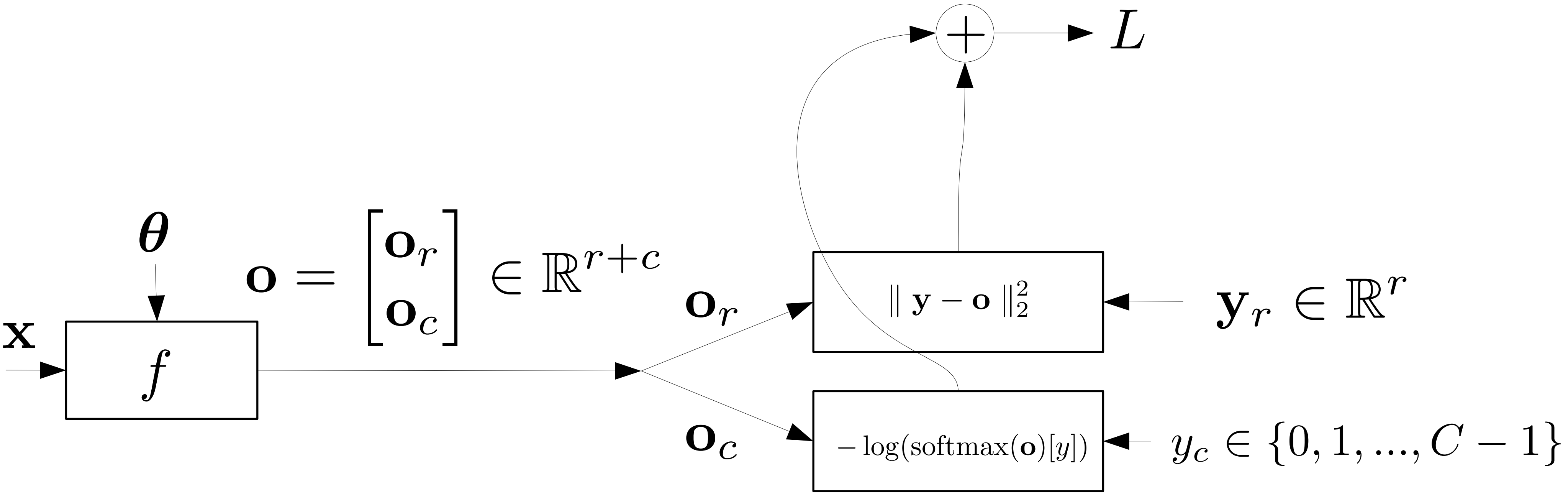
$$L = -\ln(0.071) = 2,647$$

$$\mathbf{p} \in \Delta^3 : \sum_{i=0}^{3} \mathbf{p}_i = 1 \text{ et } \mathbf{p}_i \geq 0 \text{ pour } i = 0, ..., 3$$

- Autre exemple de coût : Coût de Hinge

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

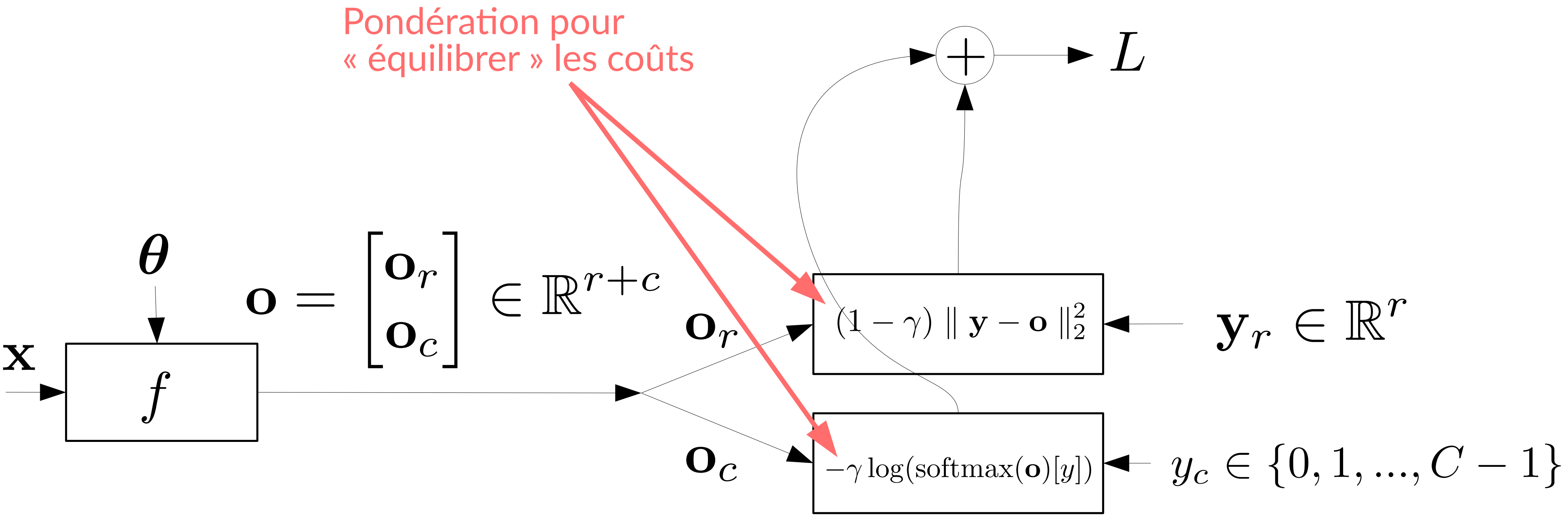
Choix du coût $l(\mathbf{y}, \mathbf{o})$: Combinaison de coûts



Exemple : classification et régression conjointe

VI)

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



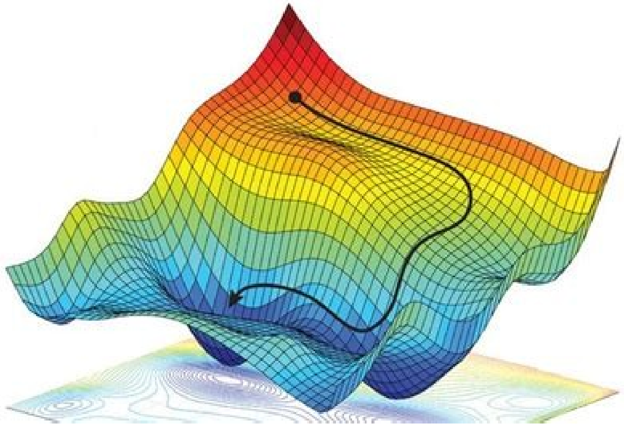
Exemple : classification et régression conjointe

Optimisation des paramètres

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

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

Descente de gradient $$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

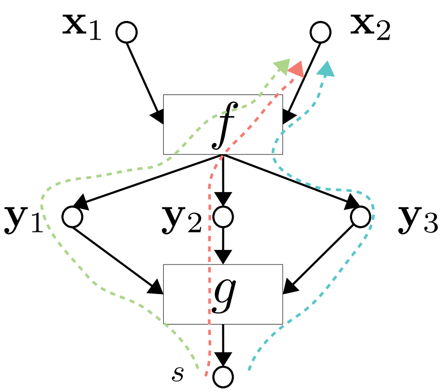


Pas d'apprentissage (« learning rate » en anglais)

Calcul du gradient

Rappel : Théorème de dérivation des fonctions composées

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



$$\frac{\partial s}{\partial \mathbf{x}_2} = \boxed{\frac{\partial s}{\partial \mathbf{y}_1}\Big|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_1}{\partial \mathbf{x}_2}} + \boxed{\frac{\partial s}{\partial \mathbf{y}_2}\Big|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_2}{\partial \mathbf{x}_2}} + \boxed{\frac{\partial s}{\partial \mathbf{y}_3}\Big|_{\mathbf{y}=f(\mathbf{x})} \frac{\partial \mathbf{y}_3}{\partial \mathbf{x}_2}}$$

Exemple

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

Formule : dérivation élément par élément

Calcul du gradient (suite)

Rappel : Théorème de dérivation des fonctions composées

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

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

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

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

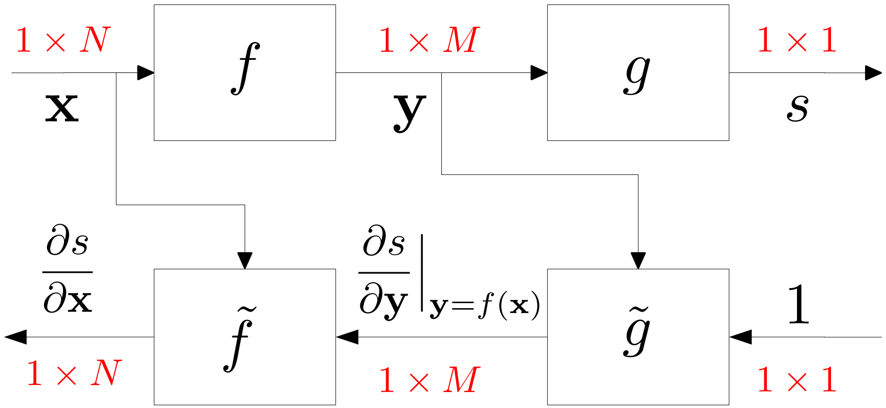
Composition de fonctions

Calcul du gradient (suite)

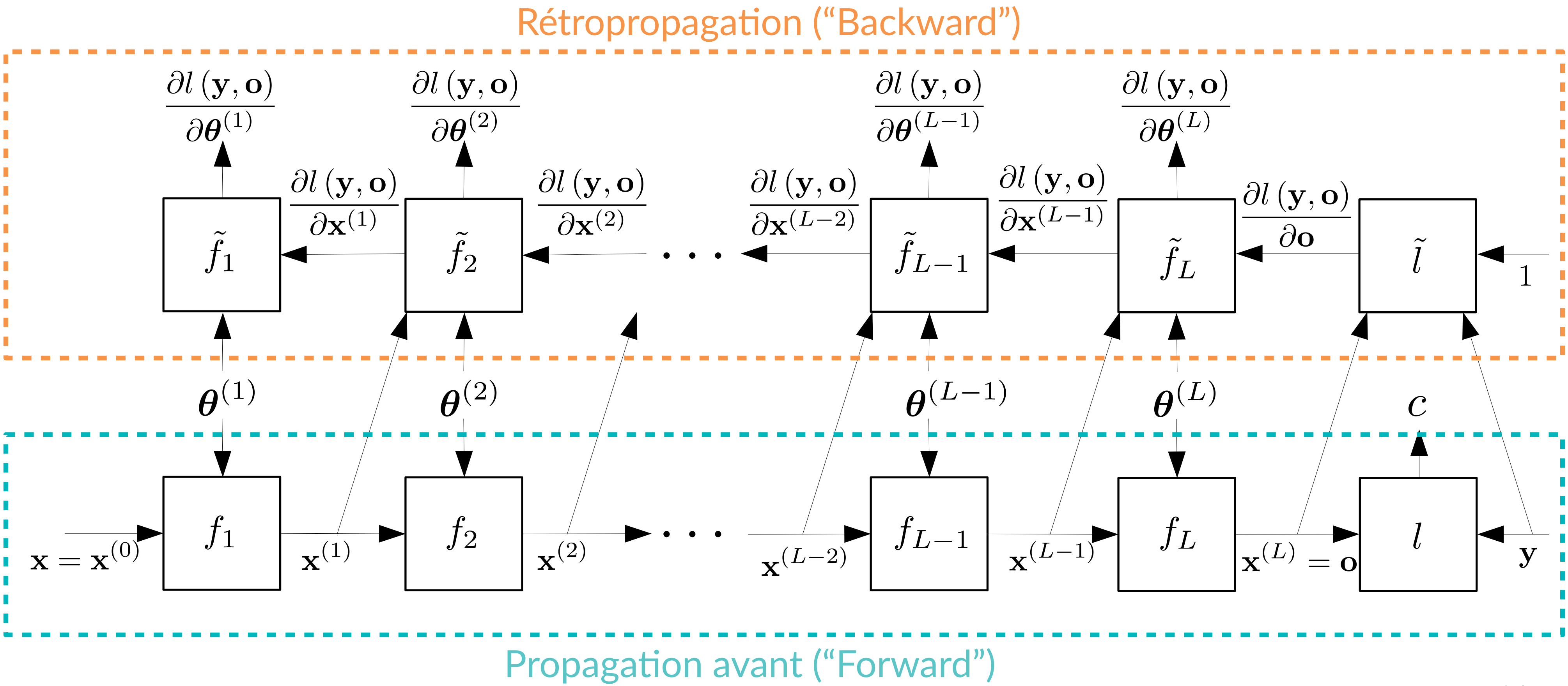
Rappel : Théorème de dérivation des fonctions composées

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

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



Calcul automatique du gradient



"Differentiable Programming"

Initialisation des paramètres

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

Exemple Kaiming init.

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

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

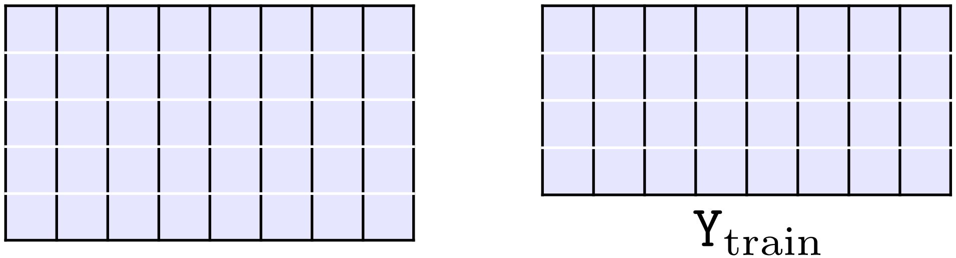
Descente de gradient (GD) :

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \sum_{i=1}^{N_{\text{train}}} \left. \frac{\partial l\left(\mathtt{Y}_{\text{train},i}, f\left(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}\right)\right)}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

Descente de gradient stochastique (SGD) :

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \sum_{i\in\Omega_k} \left. \frac{\partial l\left(\mathtt{Y}_{\text{train},i}, f\left(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}\right)\right)}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8



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

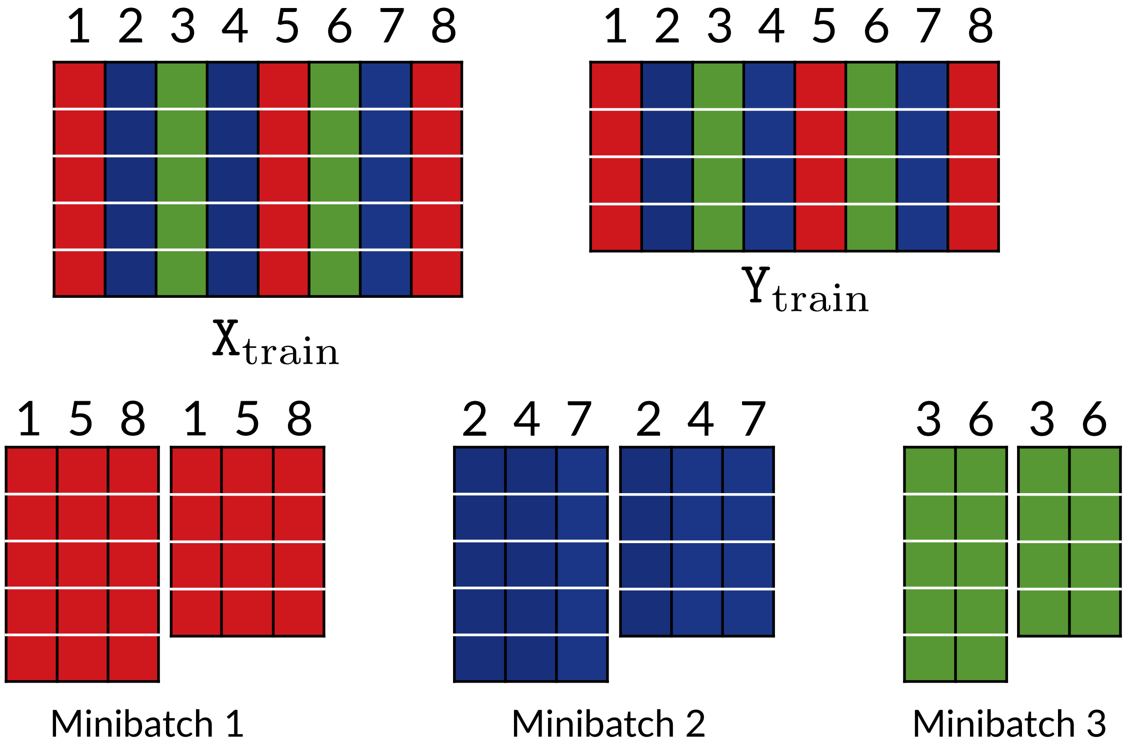
$\mathtt{X}_{\text{train}}$

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) :

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \sum_{i \in \Omega_k} \left. \frac{\partial l\left(\mathtt{Y}_{\mathrm{train},i}, f\left(\mathtt{X}_{\mathrm{train},i}; \boldsymbol{\theta}\right)\right)}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$



Une « epoch » :

1) Découper aléatoirement 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

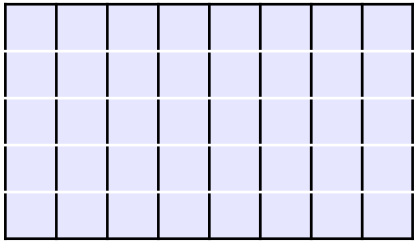
Avantage 2 : le gradient est très rapide à calculer

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

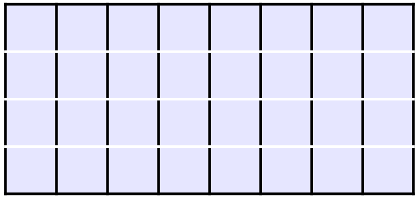
Keskar, N., et al. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.", ICLR 2017

VI)

Apprendre à « bien » prédire… sur de nouvelles données

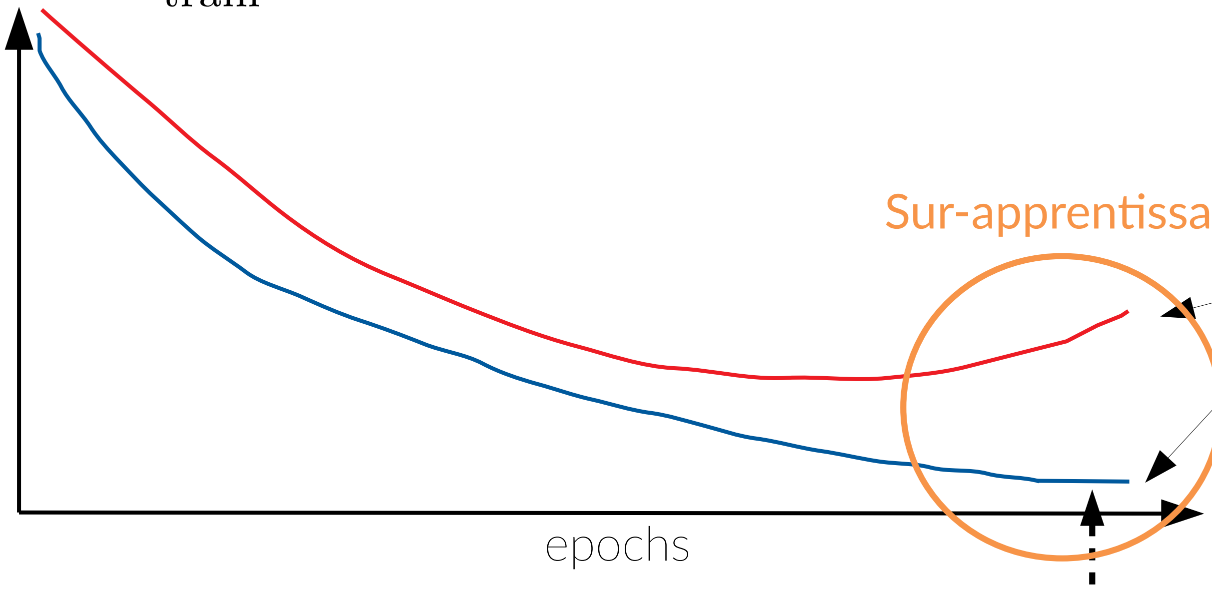


$\mathtt{X}_{\text{train}}$



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

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

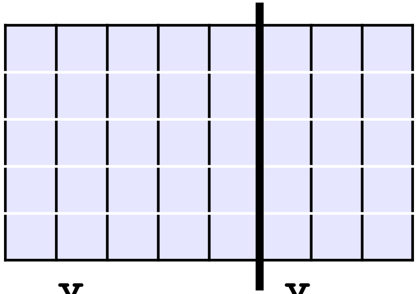


$$l\left(y_{\text{new}}, f\left(\mathbf{x}_{\text{new}}; \boldsymbol{\theta}\right)\right)$$

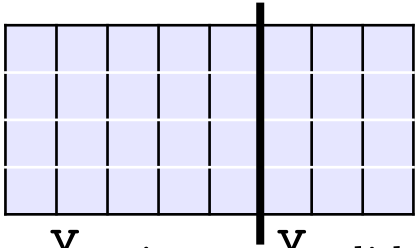
$$L_{\text{train}}\left(\boldsymbol{\theta}\right) = \sum_{i=1}^{N_{\text{train}}} l\left(\mathtt{Y}_{\text{train},i}, f\left(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}\right)\right)$$

VI)

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

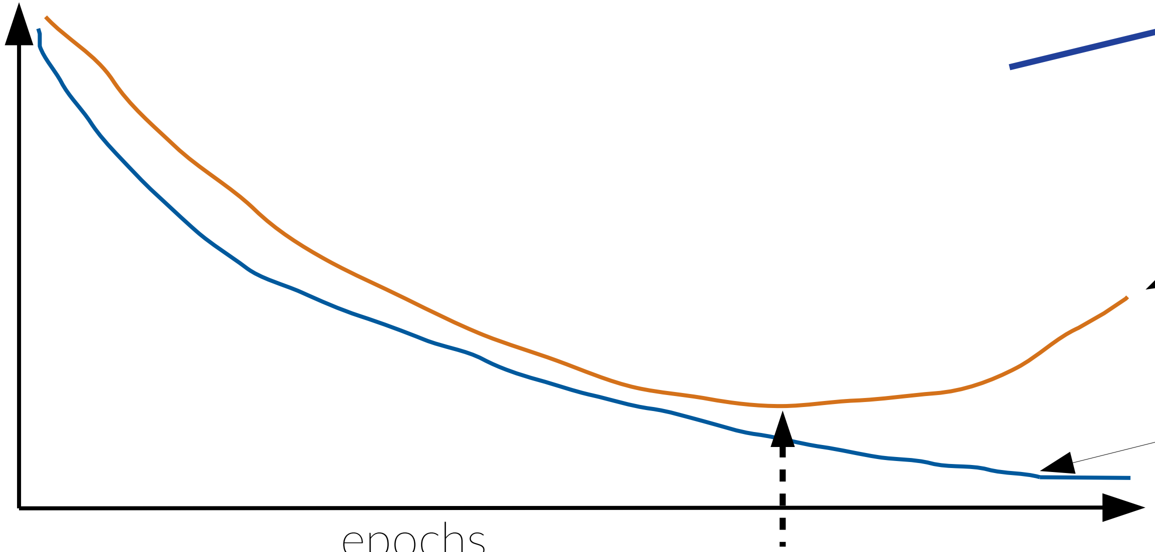


$\mathtt{X}_{\text{train}}$ $\mathtt{X}_{\text{valid}}$



$\mathtt{Y}_{\text{train}}$ $\mathtt{Y}_{\text{valid}}$

$$\cancel{\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L_{\text{train}}\left(\boldsymbol{\theta}\right)}$$



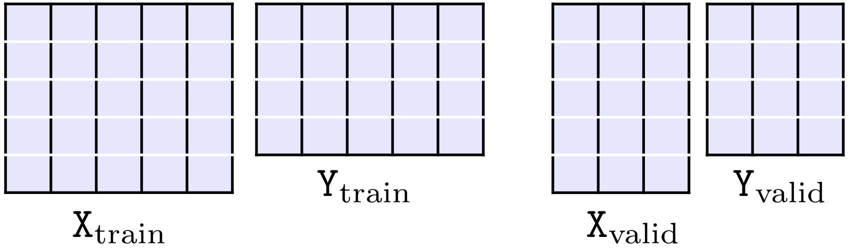
$$L_{\text{valid}}\left(\boldsymbol{\theta}\right) = \sum_{i=1}^{N_{\text{valid}}} l\left(\mathtt{Y}_{\text{valid},i}, f\left(\mathtt{X}_{\text{valid},i}; \boldsymbol{\theta}\right)\right)$$

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

VI)

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 » : $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \sum_{i \in \Omega_k} \left. \frac{\partial l \left(\mathtt{Y}_{\text{train},i}, f \left(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta} \right) \right)}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$

A la fin d'une « epoch », calculer $L_{\text{valid}}(\boldsymbol{\theta}) = \sum_{i=1}^{N_{\text{valid}}} l \left(\mathtt{Y}_{\text{valid},i}, f \left(\mathtt{X}_{\text{valid},i}; \boldsymbol{\theta} \right) \right)$

Stocker la valeur actuelle de $\boldsymbol{\theta}$ 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

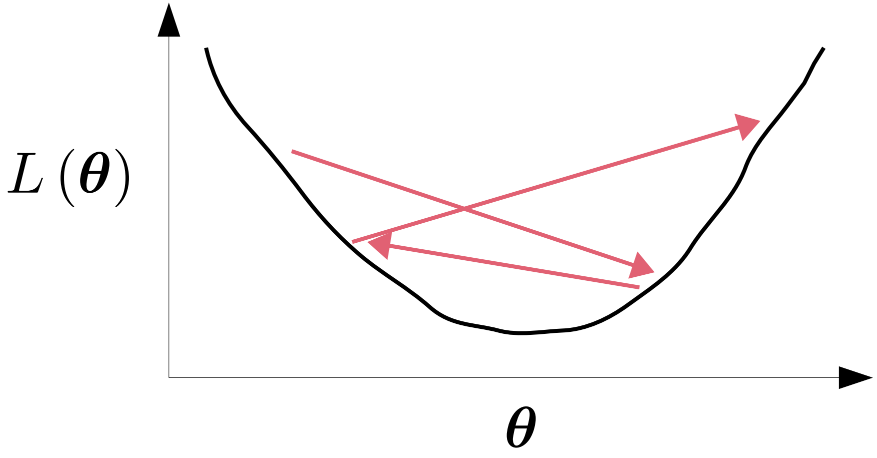
- Lancer un apprentissage sur un seul « minibatch » jusqu'à obtention d'un coût d'apprentissage de zéro

- 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
  - ...

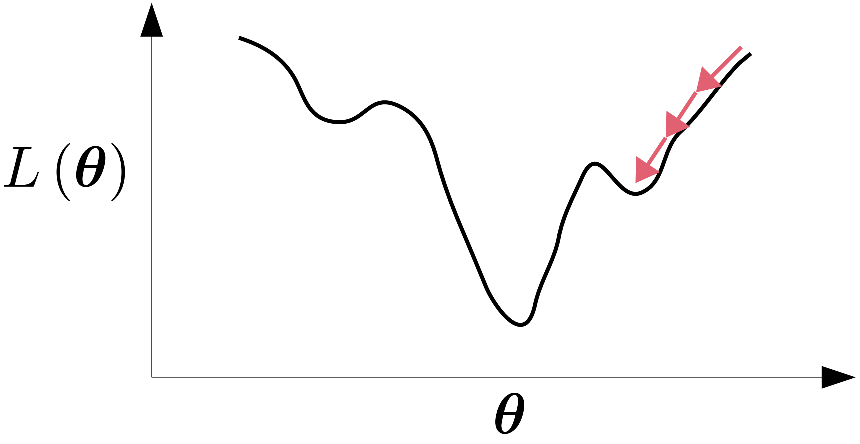
Définir la valeur du pas d'apprentissage

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \sum_{i \in \Omega_k} \left. \frac{\partial l\left(\mathtt{Y}_{\mathrm{train},i}, f\left(\mathtt{X}_{\mathrm{train},i}; \boldsymbol{\theta}\right)\right)}{\partial \boldsymbol{\theta}} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}_k}$$

Pas d'apprentissage



Pas d'apprentissage trop grand



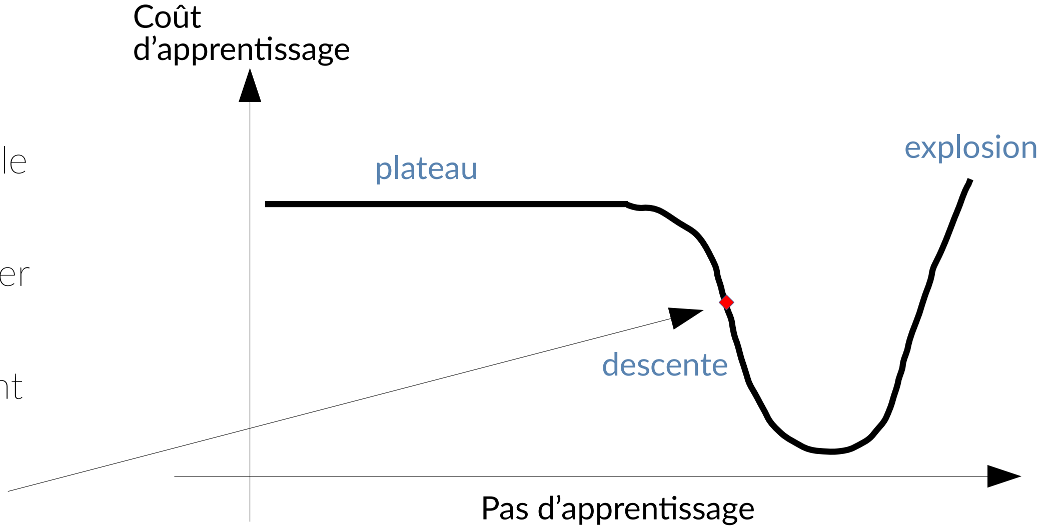
Pas d'apprentissage trop petit

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.

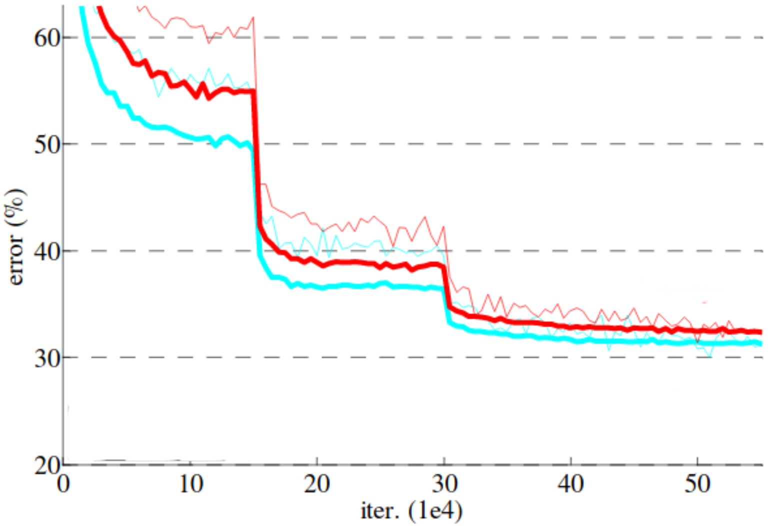


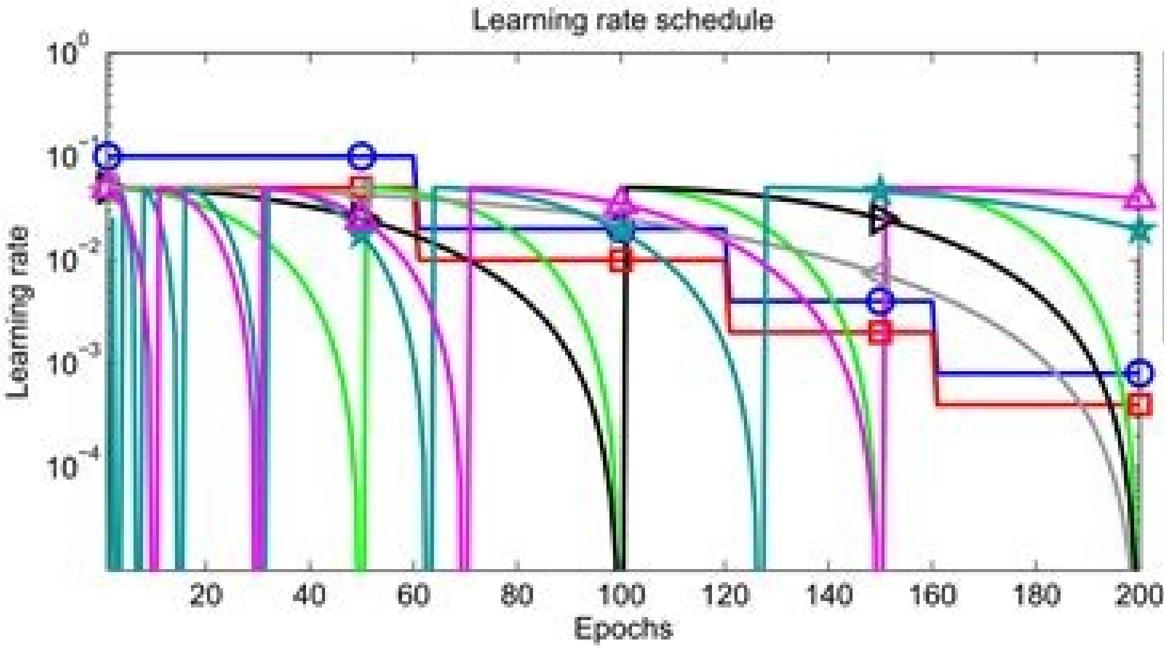
Smith, L. N. "Cyclical learning rates for training neural networks.", WACV 2017

VI)

Evolution du pas d’apprentissage durant l’optimisation

- Constant
- Décroissant
- Cyclique
- Réduction sur plateau





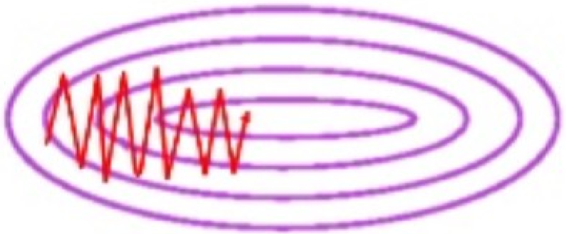
Loshchilov, I., & Hutter, F. "SGDR: Stochastic gradient descent with warm restarts." 2016

SGD avec moment (« SGD with momentum »)

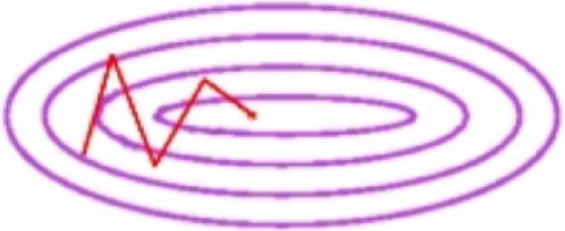
SGD

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

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



SGD avec moment



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

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

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

Source : https://image.slidesharecdn.com/

VI)

« Adam: A Method for Stochastic Optimization »

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

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

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

Carré de chaque élément de $\mathbf{g}_k$

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

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

Kingma, D. P., & Ba, J. L. (2015). Adam: A method for stochastic gradient descent. In ICLR: International Conference on Learning Representations.

VI)

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

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L_{\text{train}}(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N_{\text{train}}} l\left(\mathtt{Y}_{\text{train},i}, f\left(\mathtt{X}_{\text{train},i}; \boldsymbol{\theta}\right)\right)$$

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

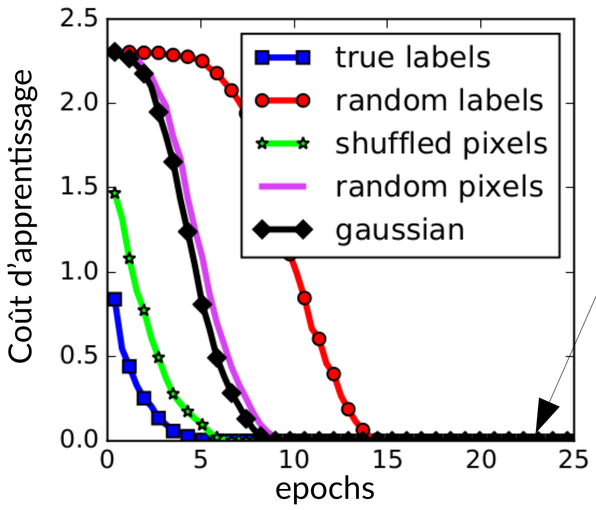
<u>Raisonnement a priori</u>

descente de gradient où f est non-convexe

↓

mauvais minimum local

<u>Résultats empiriques sur CIFAR10</u>



Zéro !

Un des minima globaux a été atteint.

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

VI)

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

Raisonnement a priori

Coût d'apprentissage atteint zéro

↓

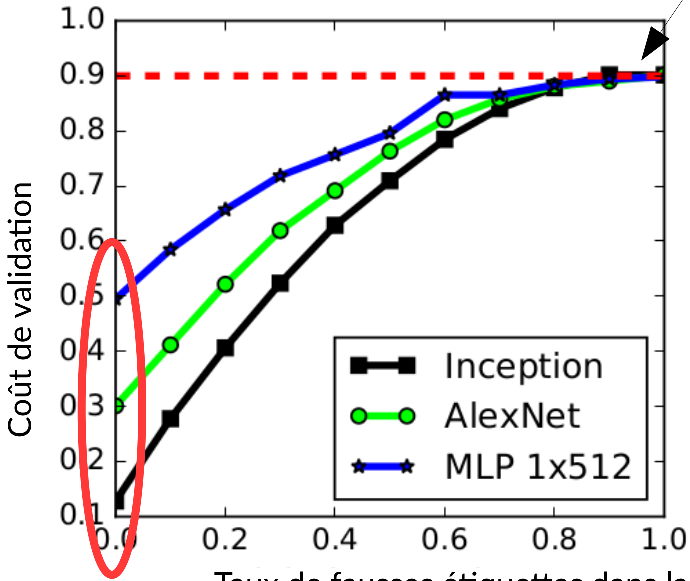
Le réseau a appris « par cœur » à associer la bonne étiquette pour chaque exemple de la base d'apprentissage

↓

Les performances de généralisation seront très mauvaises

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

Résultats empiriques sur CIFAR10



Taux de fausses étiquettes dans la base d'apprentissage

Apprentissage avec de fausses étiquettes

CIFAR10 → 10 classes

Résultat attendu : Le réseau a appris à partir de fausses étiquettes donc il obtient de mauvaises performances quand on lui présente des exemples avec les vraies étiquettes.

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

VII)

Limites d'une transformation affine générale (FC)

H=480 

W=640

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

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

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

10^6x1 10^6x10^6 10^6x1 10^6x1

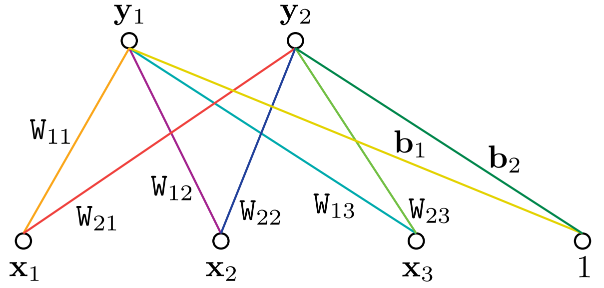
Occupation mémoire de W : 4 octets x 10^6x10^6 = 4To.

Nombre de « multiplications+additions » également très élevé.

VII)

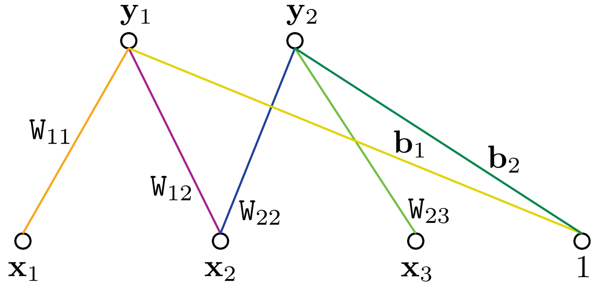
Opération de « convolution » = transformation affine spécifique

“Fully Connected”
=
Transformation affine générale



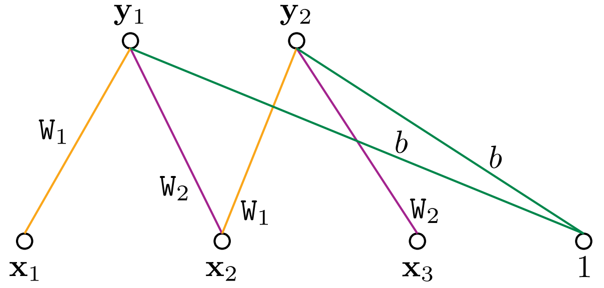
$$\begin{bmatrix} W_{11} & W_{12} & W_{13} & W_{14} \\ W_{21} & W_{22} & W_{23} & W_{24} \\ W_{31} & W_{32} & W_{33} & W_{34} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \end{bmatrix} + \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix}$$

Localement connecté



$$\begin{bmatrix} W_{11} & W_{12} & 0 & 0 \\ 0 & W_{22} & W_{23} & 0 \\ 0 & 0 & W_{33} & W_{34} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \end{bmatrix} + \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix}$$

Équivariance par translation
=
« Convolution »



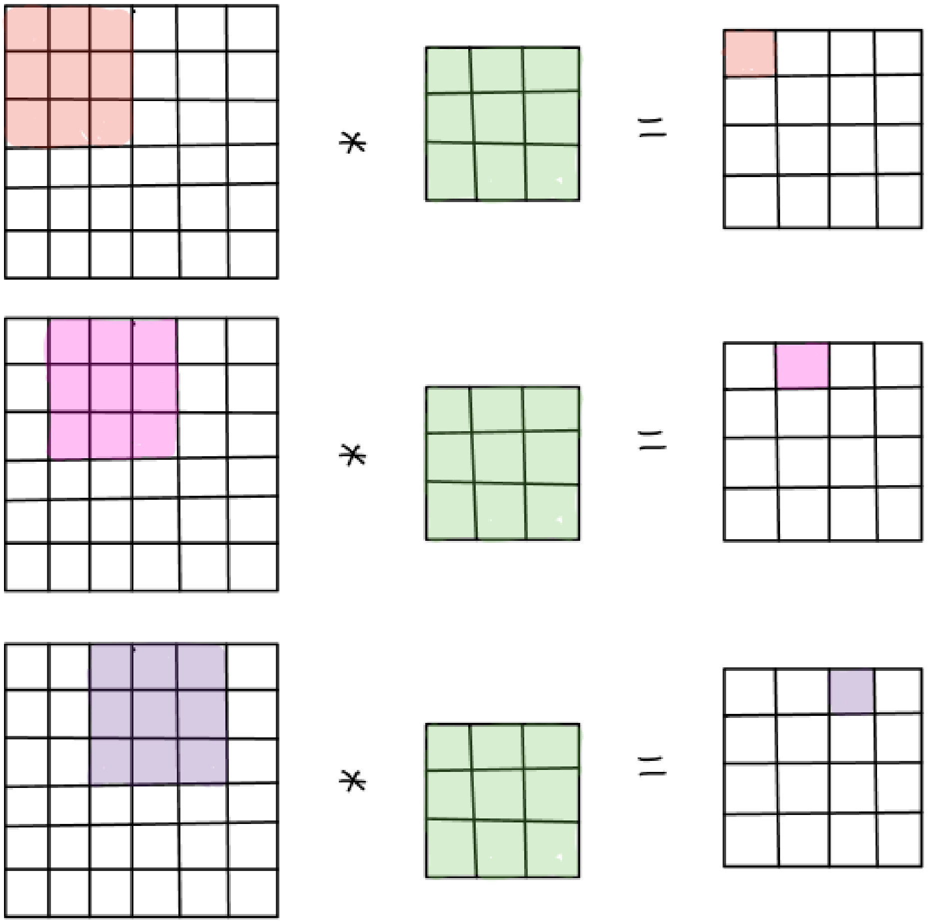
$$\begin{bmatrix} W_{11} & W_{12} & 0 & 0 \\ 0 & W_{11} & W_{12} & 0 \\ 0 & 0 & W_{11} & W_{12} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \\ \mathbf{x}_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix}$$

Beaucoup moins de paramètres à stocker

Beaucoup moins de « multiplications+additions »

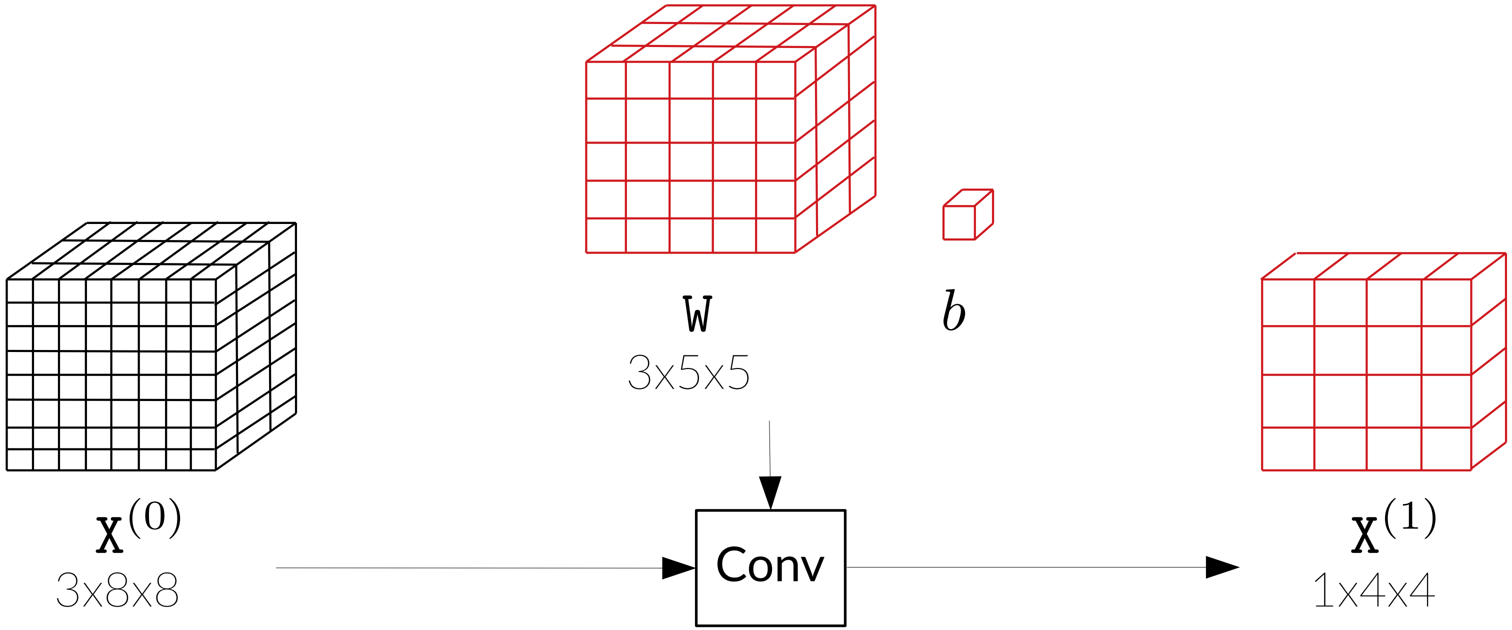
Opération de « convolution » en 2D

En fait, il s'agit d'une intercorrélation



Source : https://guandi1995.github.io/Padding/

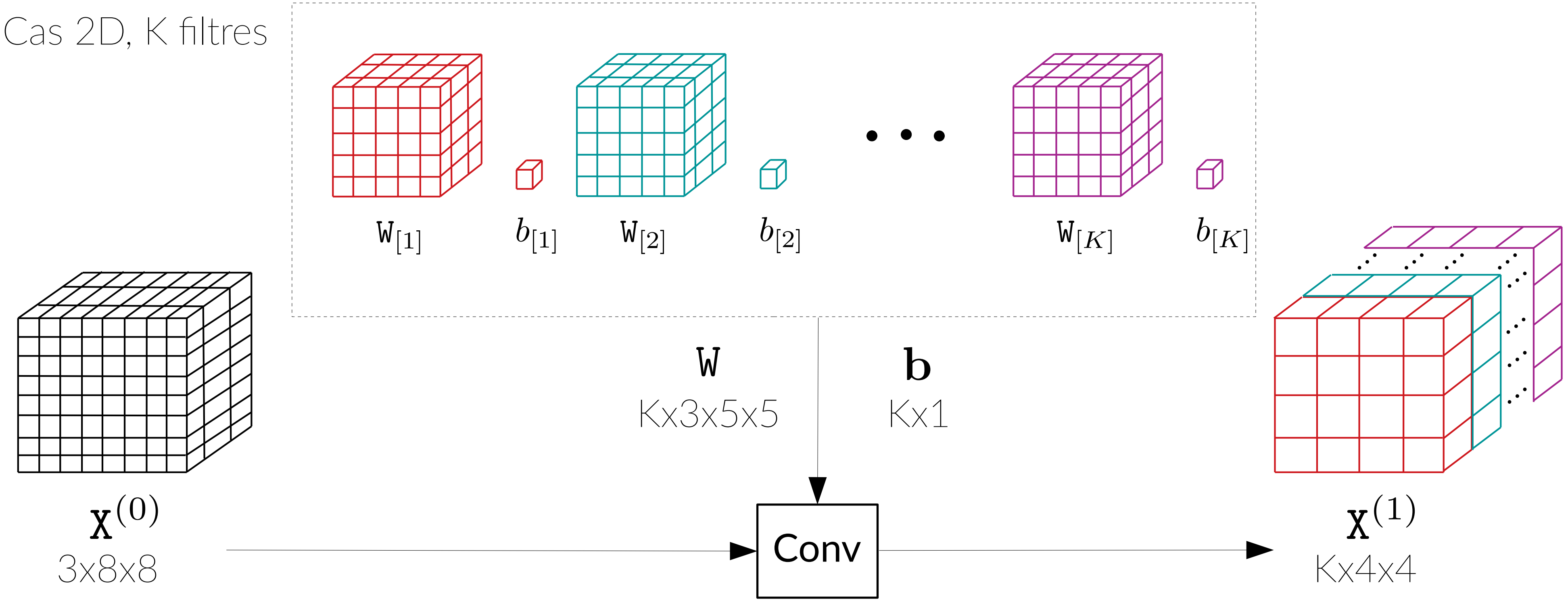
Couche de convolution : un seul filtre



$$\mathtt{X}^{(1)}_{i,j} = \sum_{k=0}^{2}\sum_{m=0}^{4}\sum_{n=0}^{4} \mathtt{W}_{k,m,n}\mathtt{X}^{(0)}_{k,i+m,j+n} + b$$

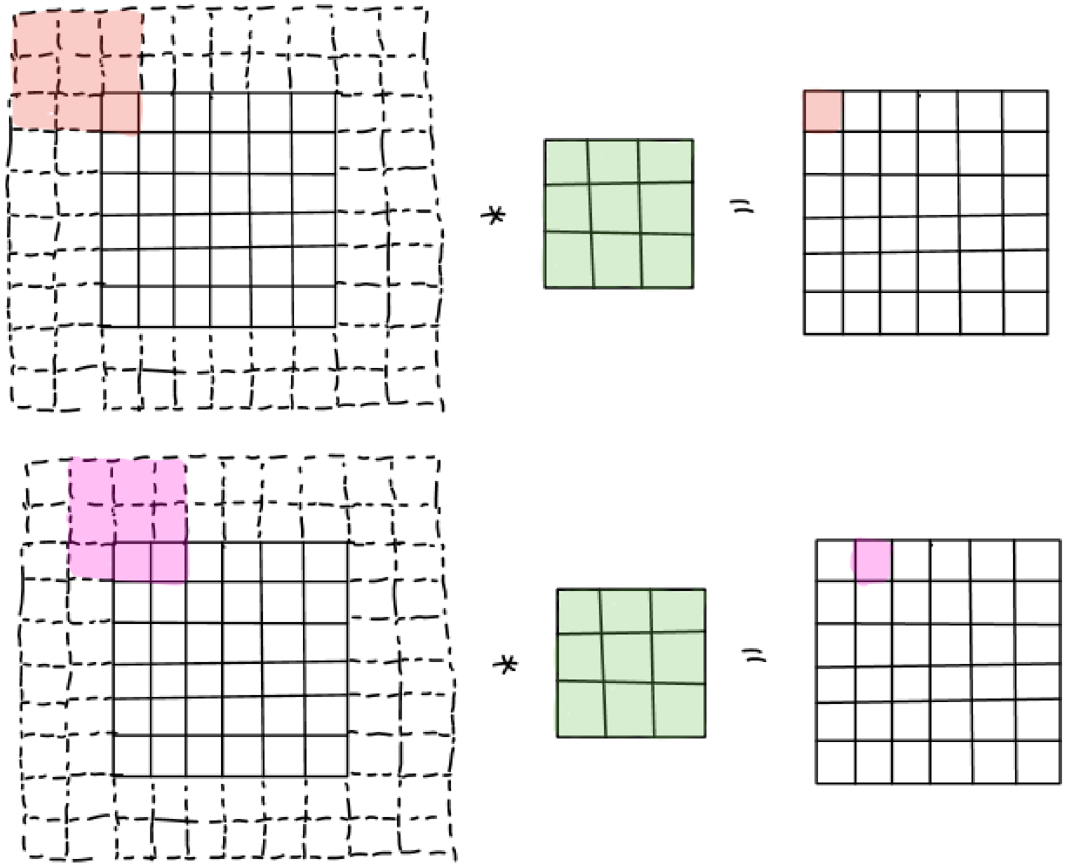
VII)

Couche de convolution : K filtres



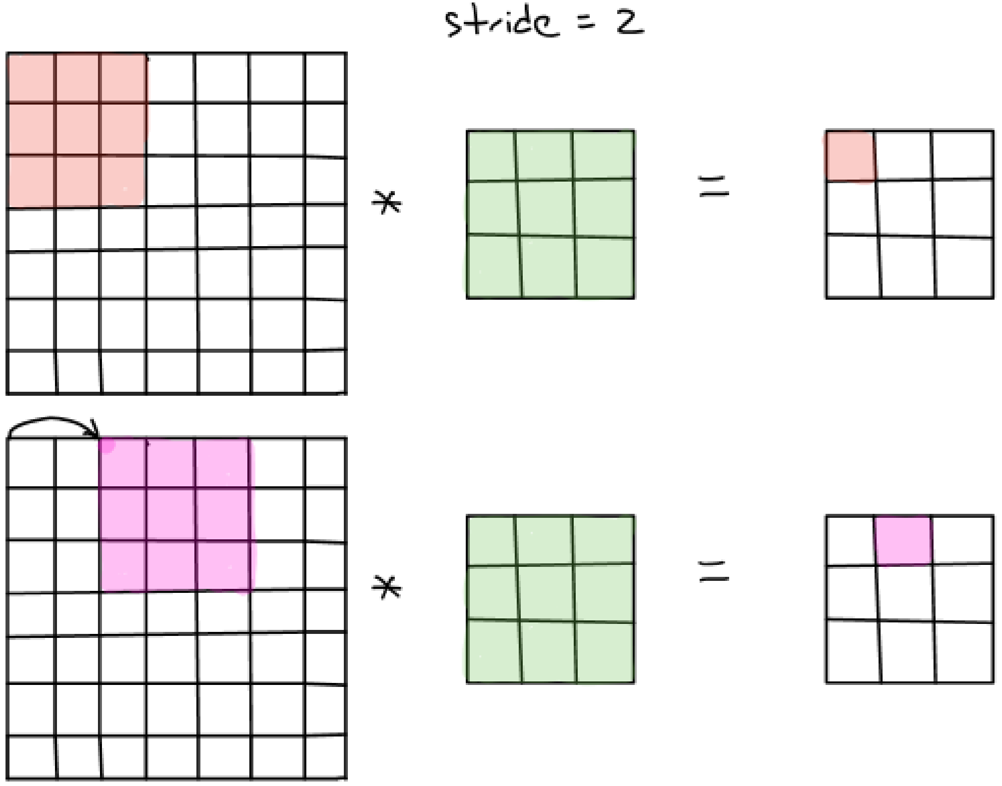
VII)

« Zero padding »



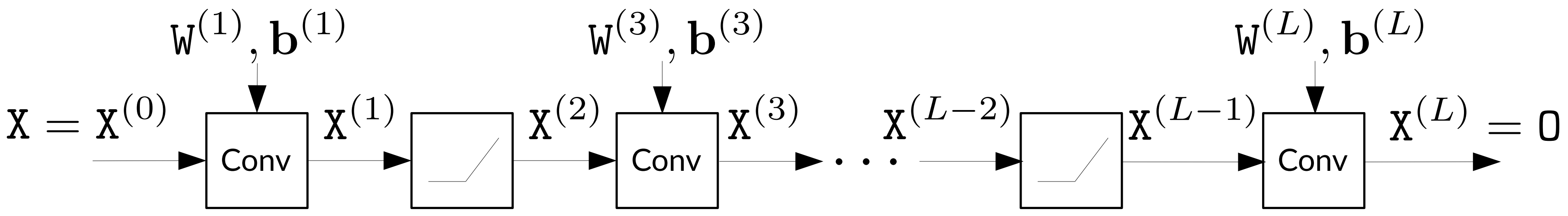
Source : https://guandi1995.github.io/Padding/

« Stride »



Source : https://guandi1995.github.io/Padding/

Réseau de neurones à convolution (CNN)



MLP – transformations affines générales + transformations affines spécifiques

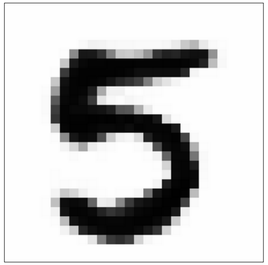
=

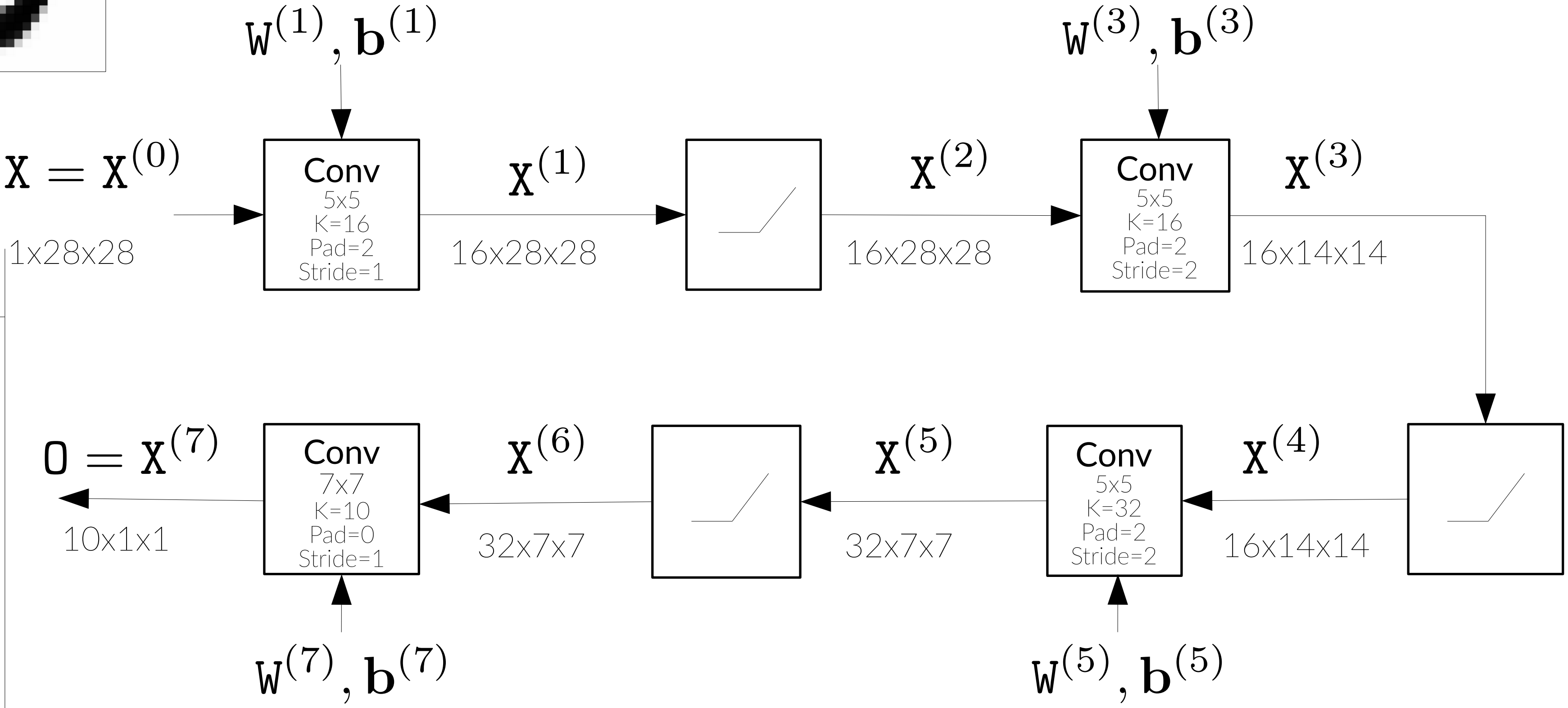
CNN

→ même initialisation des paramètres que pour le MLP

VII)

Exemple d'architecture : CNN pour MNIST

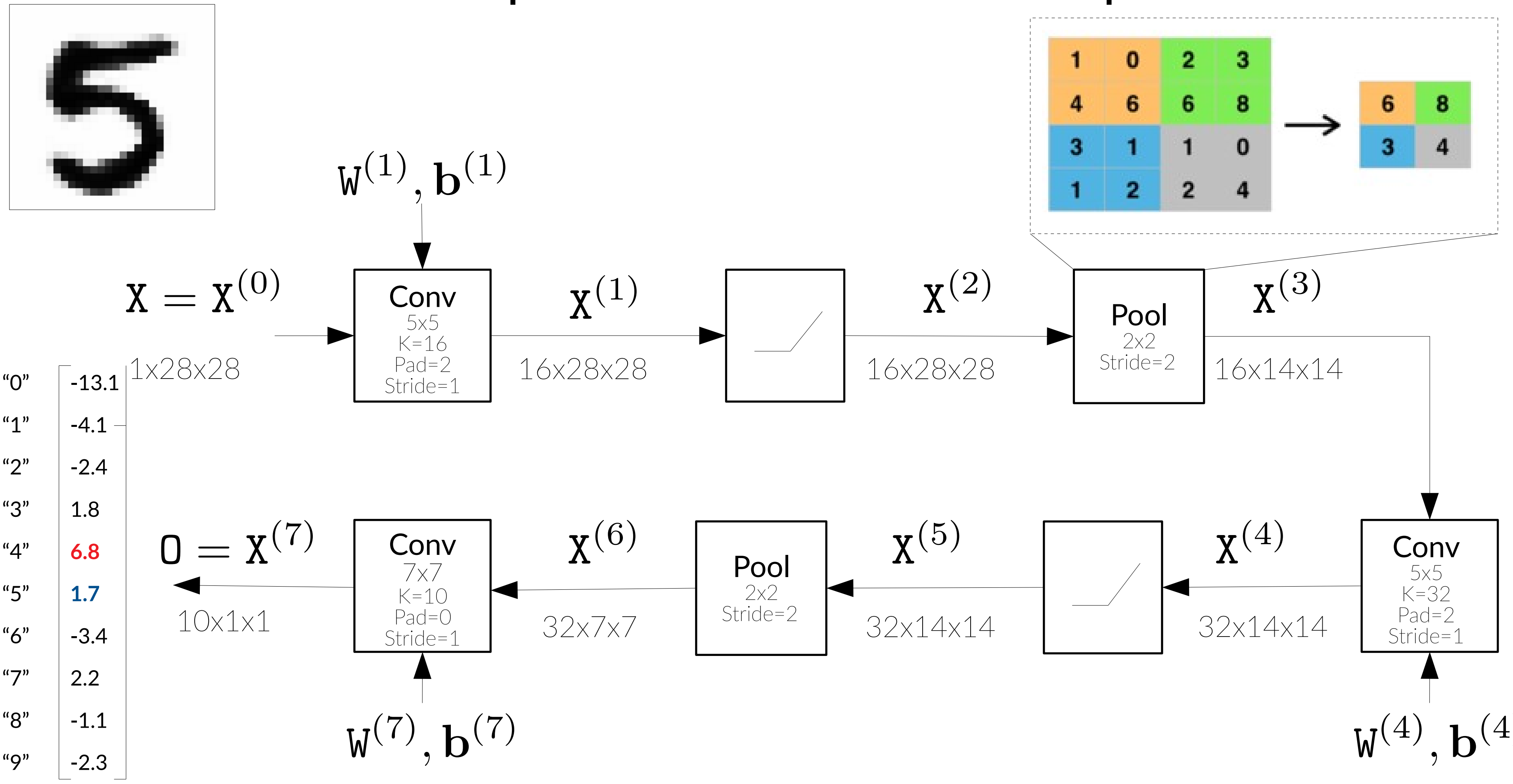




"0"	-13.1
"1"	-4.1
"2"	-2.4
"3"	1.8
"4"	6.8
"5"	1.7
"6"	-3.4
"7"	2.2
"8"	-1.1
"9"	-2.3

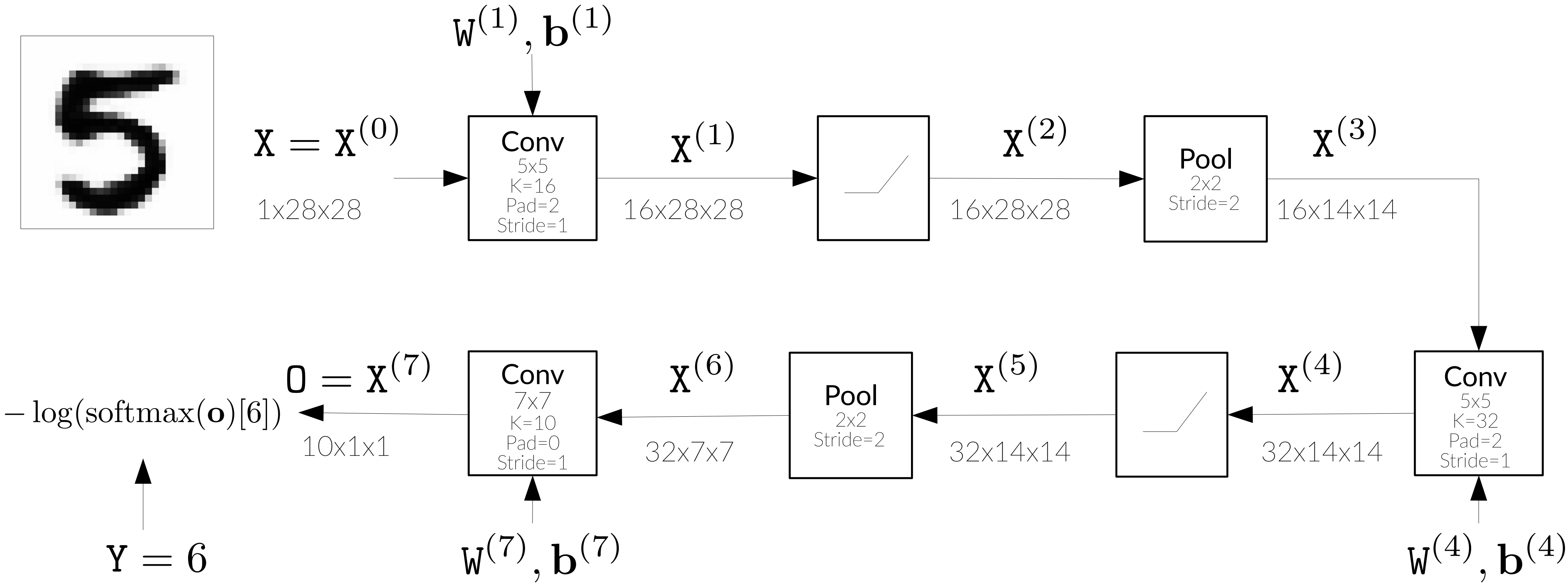
VII)

Autre exemple d'architecture : CNN pour MNIST



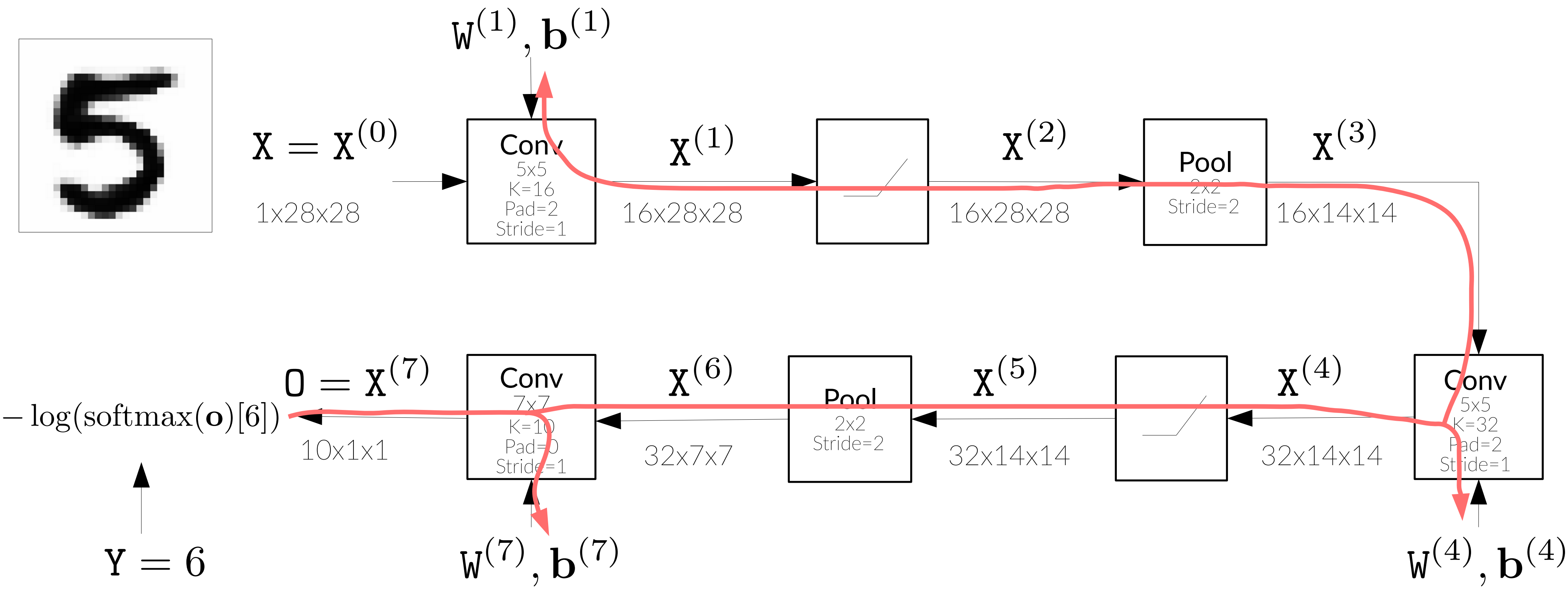
VII)

Apprentissage CNN pour MNIST



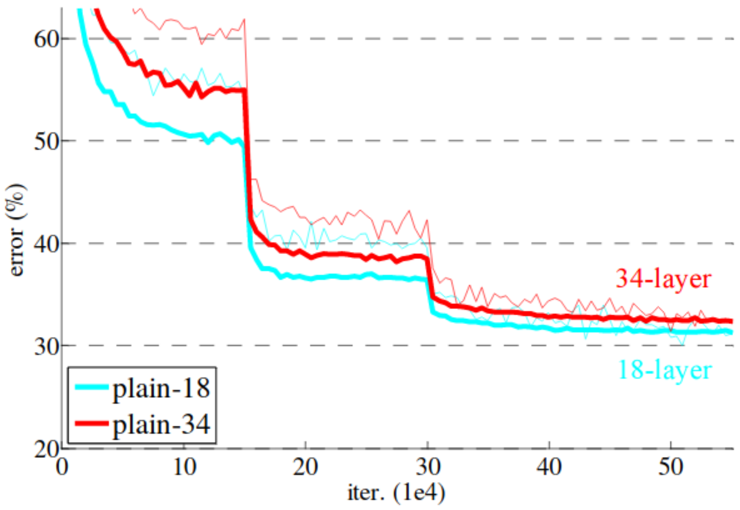
VII)

Apprentissage CNN pour MNIST (suite)



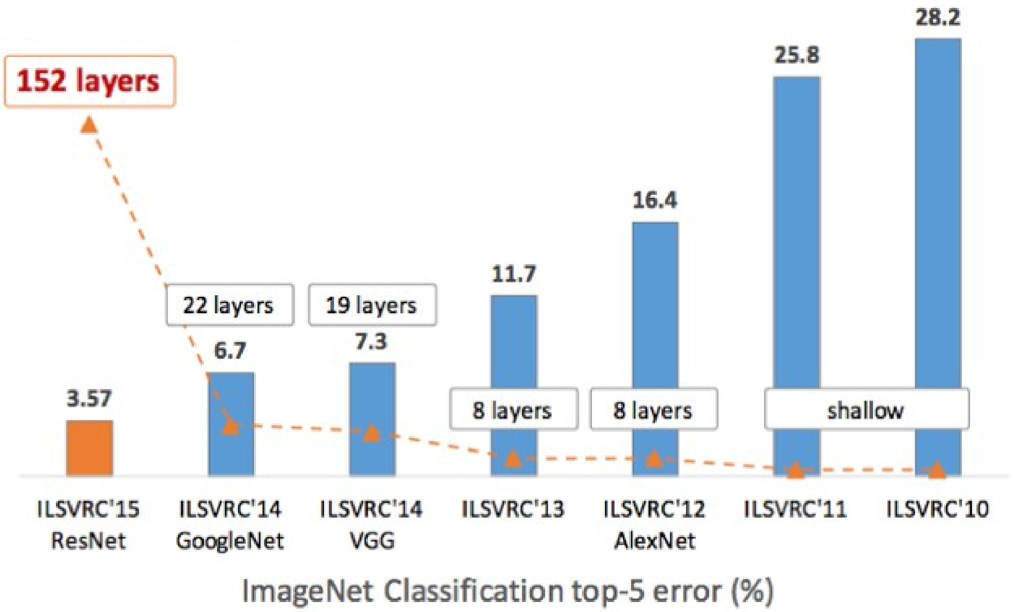
VIII) Réseaux de neurones profonds

Réseaux de neurones profonds



He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

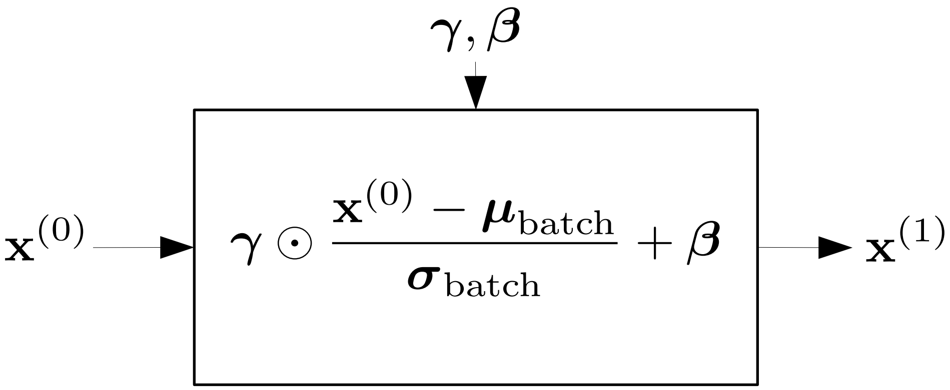
Réseaux de neurones profonds (suite)



Source : https://medium.com/@Lidinwise/the-revolution-of-depth-facf174924f5

VIII)

Couche de “Batch Normalization”



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

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

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

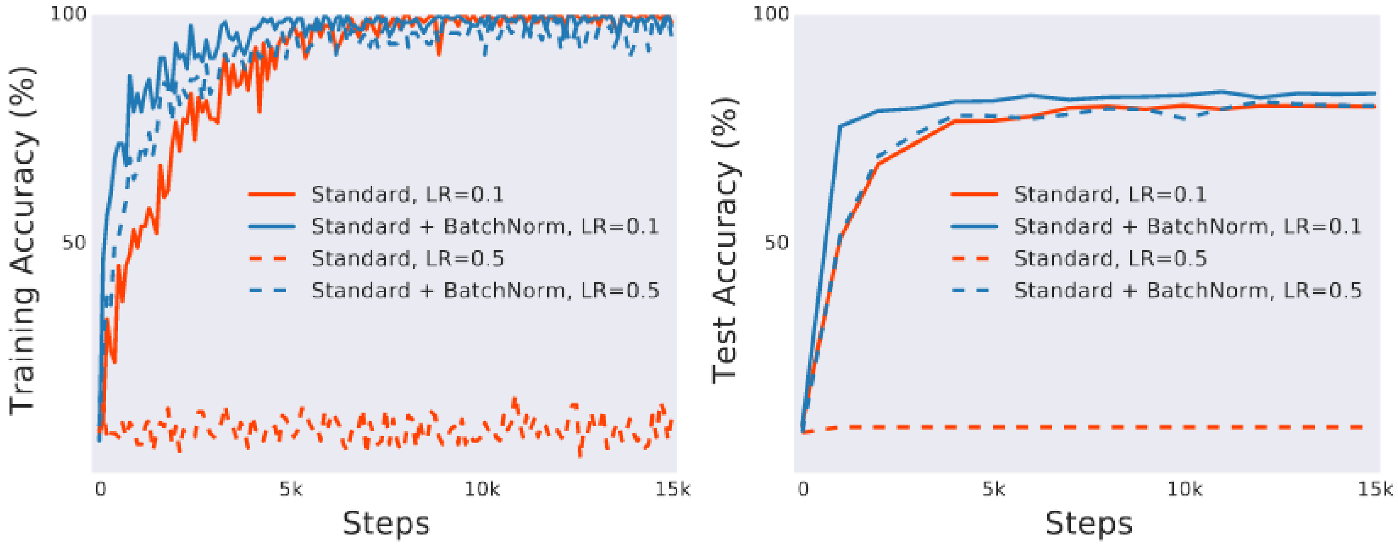
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

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

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

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

Couche de “Batch Normalization” (suite)

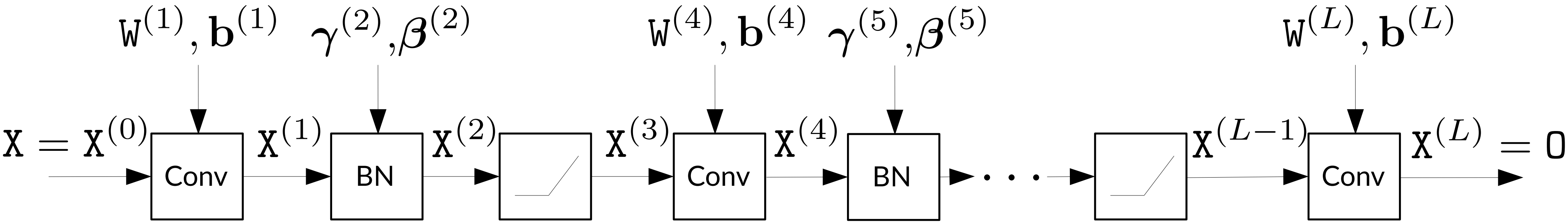


L’utilisation de couches de « batch normalization » rend le problème d’optimisation plus « lisse », ce qui implique :

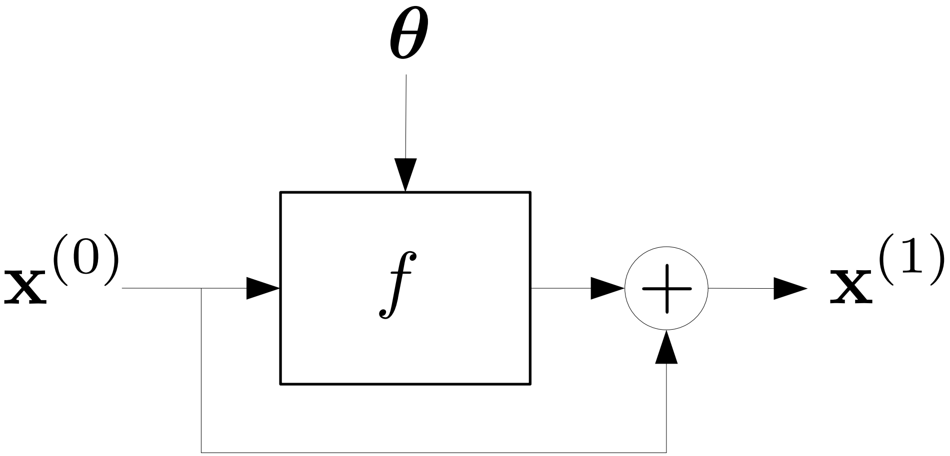
- Initialisation des paramètres moins critique
- Possibilité d’utilisation un plus grand pas d’apprentissage

Santurkar, Shibani, et al. "How does batch normalization help optimization?." Advances in neural information processing systems. 2018.

Couche de “Batch Normalization” (suite)



Connexion résiduelle



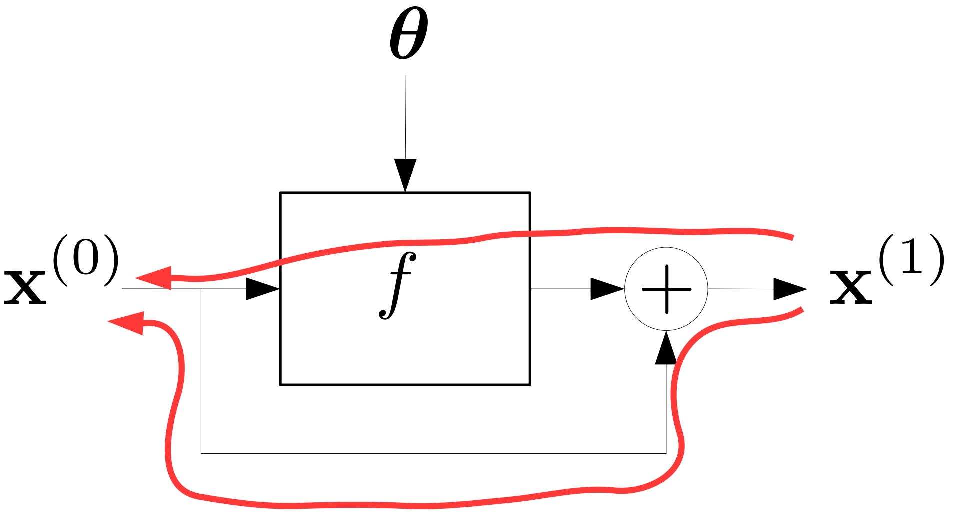
$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + f(\mathbf{x}^{(0)}; \boldsymbol{\theta})$$

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

He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Connexion résiduelle (suite)



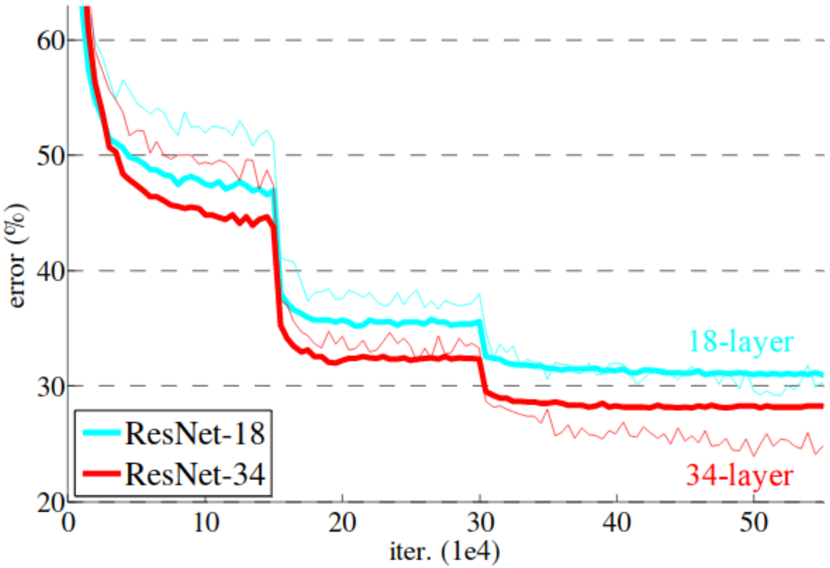
$$\frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{x}^{(0)}} = \mathrm{I} + \frac{\partial f(\mathbf{x}^{(0)}; \boldsymbol{\theta})}{\partial \mathbf{x}^{(0)}}$$

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

He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Connexion résiduelle (suite)

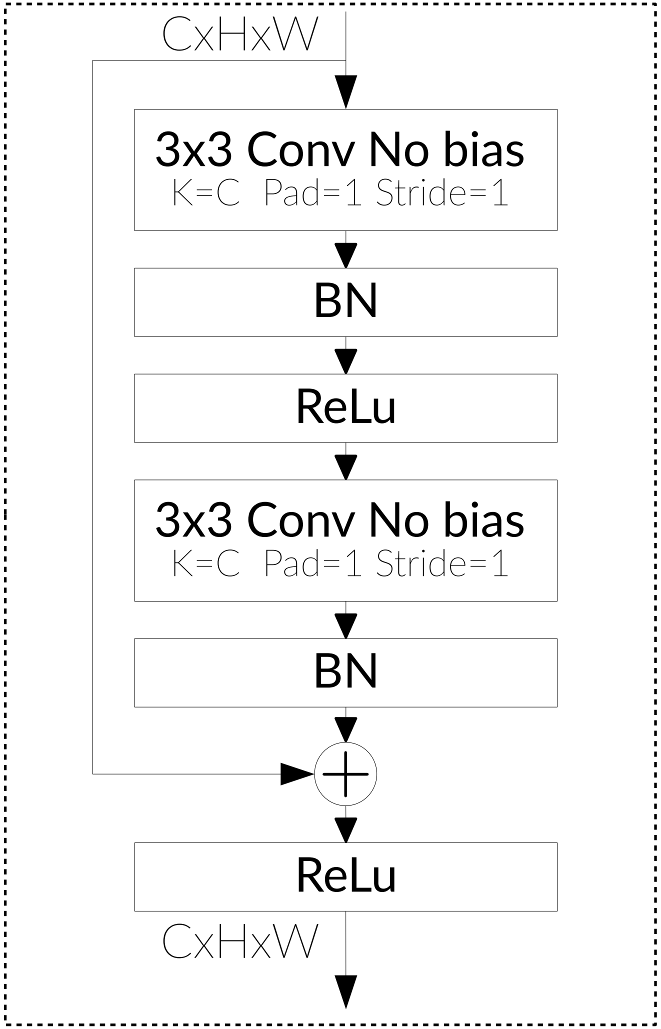


Attention résultat obtenu en combinant « batch norm » et « residual connection »

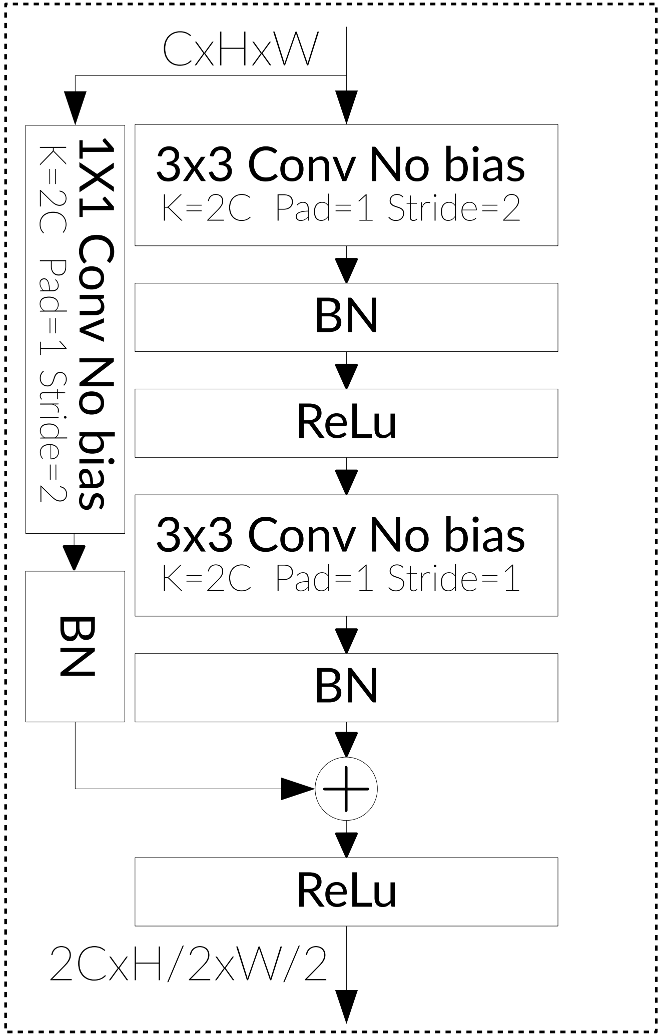
He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

VIII)

ResNet



ResBlock A



ResBlock B

He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

VIII)

ResNet 18

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

Passage en grande dimension

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

ResBlock A

128x28x28

ResBlock B

256x14x14

ResBlock A

256x14x14

ResBlock B

512x7x7

ResBlock A

512x7x7

7x7 Average Pool

512x1

1000x512 FC

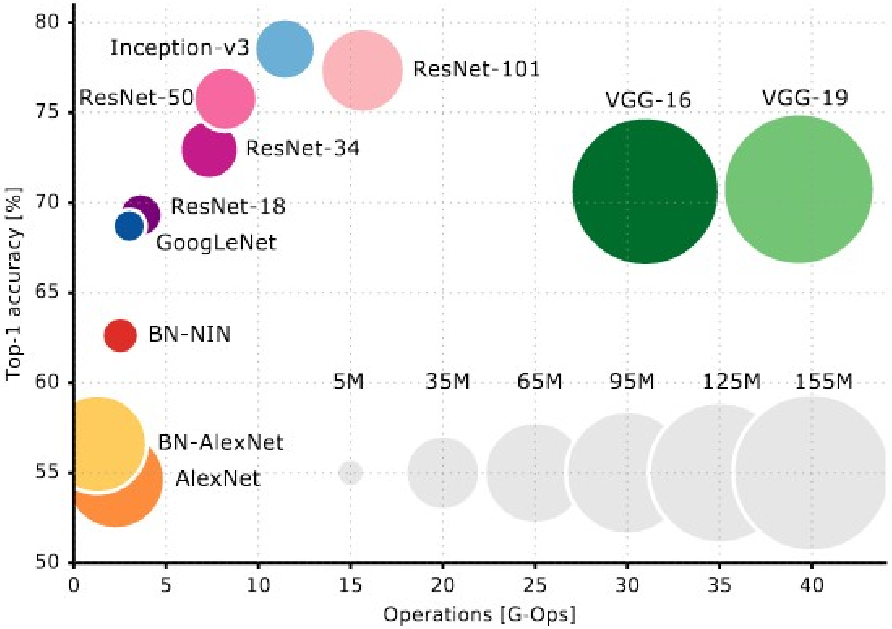
1000x1

$\mathbf{o}$

He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

VIII)

Précision vs Nombre de paramètres vs Nombre d'opérations



Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. arXiv preprint arXiv:1605.07678.