Fondamentaux des réseaux de neurones

Compétences travaillées : calcul de propagation avant, interprétation des prédictions, normalisation des données, comptage de paramètres, descente de gradient, rétropropagation, fonctions d'activation, softmax et bases des convolutions.

Version web

Ex 1 — Forward pass et interprétation d'un neurone sigmoïde

g5Ka

★ ★

On considère un réseau réduit à un seul neurone sigmoïde, de paramètres $w = 2$ et $b = -1$, recevant une entrée $x = 1$. On rappelle que $\sigma(z) = \frac{1}{1+e^{-z}}$ et que $\sigma(1) \approx 0{,}73$.

1. Calculer $z = wx + b$, puis $\hat{p} = \sigma(z)$.

   ✓ On a $z = 2 \times 1 + (-1) = 1$, donc $\hat{p} = \sigma(1) \approx 0{,}73$.

2. La vraie classe est $y = 0$. Le modèle se trompe-t-il ? Avec quel degré de confiance prédit-il la classe 1 ?

   ✓ Le modèle prédit $\hat{p} \approx 0{,}73$, soit une probabilité de 73 % pour la classe 1. Or la vraie classe est $y = 0$: le modèle se trompe, avec un degré de confiance élevé (73 %) dans la mauvaise direction.

3. Sans calculer la perte d'entropie croisée binaire (BCE) exacte, anticiper si elle sera plutôt faible ou forte. Justifier.

   ✓ La BCE vaut $-\log(1-\hat{p}) \approx -\log(0{,}27) \approx 1{,}31$. Elle est *forte* : le modèle est confiant ($\hat{p} \approx 0{,}73$) mais dans la mauvaise classe. Plus le modèle est confiant et erroné, plus $-\log$ d'une probabilité proche de 0 est grande.

Ex 2 — Effet de la normalisation sur le gradient

I54c

★ ★

On dispose de deux exemples $x^{(1)} = 1000$ et $x^{(2)} = 2000$. La normalisation standard est effectuée avec une moyenne $\mu = 1500$ et un écart-type $\sigma = 500$.

1. Calculer les valeurs normalisées $\tilde{x}^{(1)}$ et $\tilde{x}^{(2)}$.

   ✓ $$\tilde{x}^{(1)} = \frac{1000 - 1500}{500} = -1, \qquad \tilde{x}^{(2)} = \frac{2000 - 1500}{500} = 1.$$

2. Si on oublie de normaliser et qu'on initialise $w = 0{,}01$, comment le gradient $\frac{\partial J}{\partial w}$ se compare-t-il proportionnellement à celui obtenu avec les données normalisées ? Qu'est-ce que ça change en pratique ?

✓

Sans normalisation, le gradient est multiplié par un facteur de l'ordre de $x \sim 1000$–2000, soit environ 1000 à 2000 fois plus grand qu'avec des données normalisées (où $x \in \{-1, 1\}$). En pratique, cela impose d'utiliser un taux d'apprentissage très faible pour éviter la divergence, et ralentit considérablement la convergence.

Ex 3 Comptage de paramètres d'un MLP

GFrG

⋆

On considère un réseau de neurones entièrement connecté avec une entrée de dimension 3, une couche cachée de 4 neurones ReLU, et une sortie scalaire.

1. Combien de paramètres ce réseau contient-il au total ? Détailler par couche.

✓

— Couche 1 (entrée → cachée) : $3 \times 4 = 12$ poids $+4$ biais $= 16$ paramètres.
— Couche 2 (cachée → sortie) : $4 \times 1 = 4$ poids $+1$ biais $= 5$ paramètres.

Total : $16 + 5 =$ **21** paramètres.

2. Si on double la taille de la couche cachée (8 neurones), combien de paramètres supplémentaires ajoute-t-on ?

✓

Avec $h = 8$:
— Couche 1 : $3 \times 8 + 8 = 32$ paramètres ($+16$ par rapport à $h = 4$).
— Couche 2 : $8 \times 1 + 1 = 9$ paramètres ($+4$ par rapport à $h = 4$).

Paramètres supplémentaires : $16 + 4 =$ **20**.

3. Même question pour une image 8×8 aplatie en entrée avec une couche cachée de 32 neurones.

✓

L'entrée aplatie est de dimension 64.
— Couche 1 : $64 \times 32 + 32 = 2080$ paramètres.
— Couche 2 : $32 \times 1 + 1 = 33$ paramètres.

Total : $2080 + 33 =$ **2113** paramètres.

Ex 4 Nombre de paramètres d'un réseau de neurones

nEkp

⋆

On considère deux réseaux de neurones ayant une seule couche cachée contenant h neurones et une unique sortie.
Le premier réseau reçoit une seule variable d'entrée : son architecture est $1 \to h \to 1$.
Le second réseau reçoit trois variables d'entrée : son architecture est $3 \to h \to 1$.
L'objectif est de comprendre comment le nombre total de paramètres dépend du nombre de variables d'entrée, lorsque le nombre de neurones cachés reste fixé.

1. Exprimer en fonction de h le nombre de paramètres du réseau $3 \to h \to 1$. Détailler par couche.

✓

— Couche d'entrée vers couche cachée : $3h$ poids et h biais, soit $4h$ paramètres.
— Couche cachée vers sortie : h poids et 1 biais, soit $h+1$ paramètres.

Le nombre total de paramètres est donc

$$4h + (h+1) = \mathbf{5h+1}.$$

2. Exprimer en fonction de h le nombre de paramètres du réseau $1 \to h \to 1$.

✓

— Couche d'entrée vers couche cachée : h poids et h biais, soit $2h$ paramètres.
— Couche cachée vers sortie : h poids et 1 biais, soit $h+1$ paramètres.

Le nombre total de paramètres est donc

$$2h + (h+1) = \mathbf{3h+1}.$$

3. Combien de paramètres supplémentaires le réseau $3 \to h \to 1$ possède-t-il par rapport au réseau $1 \to h \to 1$? Interpréter ce résultat.

✓

La différence vaut

$$(5h+1) - (3h+1) = \mathbf{2h}.$$

Le réseau à trois entrées possède donc **2h** paramètres supplémentaires.
Cela s'explique par le fait que, par rapport au réseau à une seule entrée, on ajoute 2 variables d'entrée supplémentaires, chacune étant connectée aux h neurones de la couche cachée. Cela ajoute donc $2h$ poids.

4. Application numérique : retrouver le résultat lorsque $h = 8$.

✓

Si $h = 8$, alors :

$$5h + 1 = 5 \times 8 + 1 = 41$$

pour le réseau $3 \to 8 \to 1$, et

$$3h + 1 = 3 \times 8 + 1 = 25$$

pour le réseau $1 \to 8 \to 1$.
La différence est donc

$$41 - 25 = 16.$$

Ex 5 **Une étape de descente de gradient sur un modèle linéaire** L9Tu

★ ★

On cherche à ajuster un modèle linéaire de la forme

$$\hat{y} = wx + b,$$

où x désigne l'entrée, $\hat{y}$ la valeur prédite par le modèle, et y la valeur cible.
On considère un seul exemple d'entraînement :

$$x = 2, \qquad y = 1.$$

Les paramètres initiaux du modèle sont

$$w = 3, \qquad b = 0,$$

et on choisit un taux d'apprentissage

$$\alpha = 0{,}1.$$

On utilise la fonction de coût quadratique moyenne

$$J = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right)^2.$$

Dans cet exercice, comme on ne considère qu'un seul exemple, on a $m = 1$.
L'objectif est de calculer à la main une étape de descente de gradient, puis d'interpréter le sens de la mise à jour obtenue.

1. Calculer la prédiction initiale $\hat{y}$.

   ✓ On applique la formule

   $$\hat{y} = wx + b.$$

   Donc

   $$\hat{y} = 3 \times 2 + 0 = 6.$$

2. Calculer le gradient du coût par rapport à w, noté $\frac{\partial J}{\partial w}$.

   ✓ Pour un lot de m exemples, on a

   $$\frac{\partial J}{\partial w} = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^{(i)} - y^{(i)} \right) x^{(i)}.$$

   Ici, comme $m = 1$, cela donne

   $$\frac{\partial J}{\partial w} = (\hat{y} - y)x.$$

   En remplaçant par les valeurs numériques :

   $$\frac{\partial J}{\partial w} = (6 - 1) \times 2 = 10.$$

3. Effectuer une itération de descente de gradient pour calculer la nouvelle valeur de w.

   ✓ La mise à jour s'écrit

   $$w \leftarrow w - \alpha \frac{\partial J}{\partial w}.$$

   Donc

   $$w \leftarrow 3 - 0{,}1 \times 10 = 3 - 1 = 2.$$

   La nouvelle valeur du paramètre est donc

   $$\boxed{w = 2}.$$

4. Calculer la nouvelle prédiction obtenue après cette mise à jour. La prédiction se rapproche-t-elle de la cible ?

✅ Après mise à jour, on a $w = 2$ et $b = 0$, donc

$$\hat{y}_{\text{nouveau}} = 2 \times 2 + 0 = 4.$$

La prédiction passe donc de 6 à 4.
Comme la cible vaut 1, la nouvelle prédiction est encore trop grande, mais elle est plus proche de la cible que la prédiction initiale. La mise à jour va donc dans le bon sens.

5. Sans refaire tous les calculs, expliquer pourquoi il était cohérent que la mise à jour fasse diminuer w.

✅ La prédiction initiale vaut $\hat{y} = 6$, alors que la cible vaut $y = 1$: le modèle prédit une valeur trop grande.
Comme ici $x = 2 > 0$, la prédiction

$$\hat{y} = wx + b$$

augmente lorsque w augmente, et diminue lorsque w diminue.
Pour rapprocher la prédiction de la cible, il est donc logique de diminuer w. C'est bien ce que produit la descente de gradient, puisque w passe de 3 à 2.

6. Si l'on oubliait le facteur $\frac{1}{m}$ dans la formule du gradient, quelle serait ici la nouvelle valeur de w ? Cet oubli est-il problématique dans ce cas ?

✅ Ici, comme $m = 1$, enlever le facteur $\frac{1}{m}$ ne change rien :

$$\frac{\partial J}{\partial w} = 10.$$

On obtient donc la même mise à jour :

$$w \leftarrow 3 - 0{,}1 \times 10 = 2.$$

Dans ce cas précis, l'oubli du facteur $\frac{1}{m}$ n'a donc aucun effet.
En revanche, lorsque $m > 1$, oublier ce facteur conduit à un gradient trop grand, donc à des mises à jour trop importantes, ce qui peut perturber la convergence.

Ex 6 — Effet du taux d'apprentissage sur la descente de gradient

MLC3

★ ★

On se place dans la situation suivante : $w = 3$, gradient $= 4$.

1. Avec $\alpha = 0{,}1$: quelle est la nouvelle valeur de w ?

✅ $w \leftarrow 3 - 0{,}1 \times 4 = 2{,}6$.

2. Avec $\alpha = 1$: quelle est la nouvelle valeur de w ?

✅ $w \leftarrow 3 - 1 \times 4 = -1$.

3. Avec $\alpha = 10$: quelle est la nouvelle valeur de w ? Que se passe-t-il si le gradient reste ≈ 4 à l'itération suivante ?

✓ $w \leftarrow 3 - 10 \times 4 = -37$. Si le gradient reste ≈ 4, la mise à jour suivante donne $w \leftarrow -37 - 40 = -77$. Le paramètre w diverge : le coût augmente au lieu de diminuer. Un taux d'apprentissage trop élevé empêche la convergence.

Ex 7 **Rétropropagation et problème du neurone mort** ★ ★ ★ gkVL

On étudie, sur un exemple simple, le calcul de la propagation avant et de la rétropropagation du gradient dans un réseau de neurones de régression.
Le réseau possède :

- une seule variable d'entrée ;
- une couche cachée de 2 neurones avec fonction d'activation ReLU ;
- une seule sortie réelle.

On adopte la convention suivante : les exemples sont rangés par lignes et les poids sont multipliés à droite. Ainsi, pour un lot de données X, on a

$$Z^{[1]} = XW^{[1]} + b^{[1]}, \qquad A^{[1]} = \mathrm{ReLU}(Z^{[1]}), \qquad \hat{y} = A^{[1]}W^{[2]} + b^{[2]}.$$

Dans cet exercice, on considère un seul exemple, donc

$$X = \begin{bmatrix} 1 \end{bmatrix}, \qquad y = \begin{bmatrix} 0 \end{bmatrix}.$$

Les paramètres du réseau sont

$$W^{[1]} = \begin{bmatrix} 1 & -2 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} 0 & 0 \end{bmatrix}, \quad W^{[2]} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b^{[2]} = \begin{bmatrix} 0 \end{bmatrix}.$$

On utilise la fonction de coût

$$J = \frac{1}{2}(\hat{y} - y)^2.$$

Dans la suite, le symbole $\odot$ désigne le produit terme à terme, et

$$\mathrm{ReLU}'(z) = \begin{cases} 1 & \text{si } z > 0, \\ 0 & \text{si } z < 0. \end{cases}$$

1. Calculer $Z^{[1]}$, puis $A^{[1]}$. En déduire la prédiction $\hat{y}$.

✓

$$Z^{[1]} = XW^{[1]} + b^{[1]} = \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} 1 & -2 \end{bmatrix} + \begin{bmatrix} 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & -2 \end{bmatrix}.$$

$$A^{[1]} = \mathrm{ReLU}(Z^{[1]}) = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

$$\hat{y} = A^{[1]}W^{[2]} + b^{[2]} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix}.$$

2. Calculer le coût J.

✓

$$J = \frac{1}{2}(1-0)^2 = \frac{1}{2}.$$

3. Calculer

$$\delta^{[2]} = \hat{y} - y, \qquad dW^{[2]} = (A^{[1]})^\top \delta^{[2]}, \qquad db^{[2]} = \delta^{[2]}.$$

✓

$$\delta^{[2]} = \begin{bmatrix} 1 \end{bmatrix}.$$

$$dW^{[2]} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

$$db^{[2]} = \begin{bmatrix} 1 \end{bmatrix}.$$

4. Calculer

$$dA^{[1]} = \delta^{[2]}(W^{[2]})^\top, \qquad \delta^{[1]} = dA^{[1]} \odot \mathrm{ReLU}'(Z^{[1]}).$$

Que remarque-t-on pour le deuxième neurone caché ?

✓

$$dA^{[1]} = \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \end{bmatrix}.$$

$$\mathrm{ReLU}'(Z^{[1]}) = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

$$\delta^{[1]} = \begin{bmatrix} 1 & 1 \end{bmatrix} \odot \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

Le deuxième neurone caché est inactif car sa préactivation vaut $-2 < 0$. Sa dérivée est donc nulle et son gradient est bloqué à 0.

5. En déduire

$$dW^{[1]} = X^\top \delta^{[1]}, \qquad db^{[1]} = \delta^{[1]}.$$

✓

$$dW^{[1]} = \begin{bmatrix} 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

$$db^{[1]} = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

6. Quels paramètres seront modifiés lors d'une étape de descente de gradient ?

✓

Les gradients associés au premier neurone caché sont non nuls, ainsi que le biais de sortie. En revanche, les paramètres liés au deuxième neurone caché ne seront pas modifiés sur cet exemple, car leur gradient est nul.

Ex 8 — Calcul de la fonction softmax et perte entropie croisée — 01Bu

★ ★

Un réseau multiclasse produit les scores bruts $z = [3,\ 1,\ 0]$ pour 3 classes. On rappelle que $\text{softmax}(z)_k = \frac{e^{z_k}}{\sum_j e^{z_j}}$, et que $e^0 = 1$, $e^1 \approx 2{,}7$, $e^3 \approx 20$.

1. Calculer approximativement $\hat{p}_1$, $\hat{p}_2$, $\hat{p}_3$.

✓

$\sum_j e^{z_j} \approx 20 + 2{,}7 + 1 = 23{,}7$.

$$\hat{p}_1 \approx \frac{20}{23{,}7} \approx 0{,}84, \quad \hat{p}_2 \approx \frac{2{,}7}{23{,}7} \approx 0{,}11, \quad \hat{p}_3 \approx \frac{1}{23{,}7} \approx 0{,}04.$$

2. Vérifier que leur somme vaut bien 1.

✓

$0{,}84 + 0{,}11 + 0{,}04 \approx 0{,}99 \approx 1$.

3. La vraie classe est 1. La perte CCE vaut $-\log(\hat{p}_1)$. Est-elle faible ou forte ?

✓

$-\log(0{,}84) \approx 0{,}17$. La perte est *faible* : le modèle attribue une probabilité élevée (84 %) à la bonne classe.

Ex 9 — Réseau entièrement linéaire : équivalence avec un neurone simple — 4G7c

★ ★ ★

On considère un réseau $1 \to 3 \to 1$ dont toutes les activations sont linéaires (identité). Ce réseau est-il différent d'un neurone simple $1 \to 1$? Justifier. La réponse change-t-elle par rapport au cas $1 \to 1 \to 1$ avec $h = 1$?

✓

Non. Toute composition de fonctions linéaires est linéaire. Formellement, avec $W^{[1]} \in \mathbb{R}^{1\times 3}$, $W^{[2]} \in \mathbb{R}^{3\times 1}$:

$$\hat{y} = W^{[2]}\big(W^{[1]}x + b^{[1]}\big) + b^{[2]} = \underbrace{W^{[2]}W^{[1]}}_{\tilde{w}\in\mathbb{R}}\,x + \underbrace{W^{[2]}b^{[1]} + b^{[2]}}_{\tilde{b}\in\mathbb{R}}.$$

Le réseau équivaut donc à un neurone simple $\hat{y} = \tilde{w}x + \tilde{b}$. La réponse est la même que pour $h = 1$: quelle que soit la largeur de la couche cachée, un réseau entièrement linéaire n'a pas plus de capacité d'expression qu'un modèle linéaire simple.

Ex 10 — Convolution 2D à la main — mode valid — i7Xp

★

On considère l'image I et le noyau K suivants :

$$I = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \qquad K = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

On utilise le mode **valid** (pas de padding) et un stride de 1.

1. Quelle est la taille de la carte de sortie ? Justifier à l'aide de la formule $\text{out} = \lfloor (\text{in} + 2p - k)/s \rfloor + 1$.

✓

Avec $\text{in} = 3$, $p = 0$, $k = 2$, $s = 1$:

$$\text{out} = \lfloor (3 + 0 - 2)/1 \rfloor + 1 = 1 + 1 = 2.$$

La sortie est donc de taille 2×2.

2. Calculer à la main tous les coefficients de la carte de sortie $S = I \star K$.

✓

Le noyau renversé est $K' = \text{flip}(K) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ (symétrique, donc $K' = K$).
Les quatre fenêtres 2×2 extraites de I (en mode valid) sont :

$$F_{00} = \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}, \quad F_{01} = \begin{pmatrix} 2 & 3 \\ 5 & 6 \end{pmatrix},$$
$$F_{10} = \begin{pmatrix} 4 & 5 \\ 7 & 8 \end{pmatrix}, \quad F_{11} = \begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix}.$$

Produits scalaires terme à terme avec K' :

$$S_{00} = 1 \cdot 0 + 2 \cdot 1 + 4 \cdot 1 + 5 \cdot 0 = 6,$$
$$S_{01} = 2 \cdot 0 + 3 \cdot 1 + 5 \cdot 1 + 6 \cdot 0 = 8,$$
$$S_{10} = 4 \cdot 0 + 5 \cdot 1 + 7 \cdot 1 + 8 \cdot 0 = 12,$$
$$S_{11} = 5 \cdot 0 + 6 \cdot 1 + 8 \cdot 1 + 9 \cdot 0 = 14.$$

Donc :

$$S = \begin{pmatrix} 6 & 8 \\ 12 & 14 \end{pmatrix}.$$

3. Que détecte intuitivement ce noyau K ?

✓

Le noyau $K = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ somme les deux pixels situés sur la diagonale anti-principale (haut-droite et bas-gauche) de chaque fenêtre. Il réagit aux variations diagonales (à 45) dans l'image, et peut être vu comme un détecteur de corrélation diagonale. Ce n'est ni un détecteur horizontal ni vertical pur.

Ex 11 Calcul de taille de sortie ⋆

pL3m

Pour chacune des configurations suivantes, on rappelle la formule :

$$\text{out} = \left\lfloor \frac{\text{in} + 2p - k}{s} \right\rfloor + 1,$$

où p désigne le padding, k la taille du noyau et s le stride.

1. Compléter le tableau suivant en calculant la taille de sortie pour chaque configuration.

#	Entrée	Noyau	Padding	Stride
a	10×10	3×3	0	1
b	10×10	3×3	1	1
c	10×10	5×5	2	1
d	16×16	3×3	0	2
e	28×28	5×5	0	1
f	32×32	3×3	1	2

✓

a. $\lfloor (10+0-3)/1 \rfloor + 1 = 8$. Sortie : 8×8.
b. $\lfloor (10+2-3)/1 \rfloor + 1 = 10$. Sortie : 10×10.
c. $\lfloor (10+4-5)/1 \rfloor + 1 = 10$. Sortie : 10×10.
d. $\lfloor (16+0-3)/2 \rfloor + 1 = \lfloor 13/2 \rfloor + 1 = 6 + 1 = 7$. Sortie : 7×7.
e. $\lfloor (28+0-5)/1 \rfloor + 1 = 24$. Sortie : 24×24.
f. $\lfloor (32+2-3)/2 \rfloor + 1 = \lfloor 31/2 \rfloor + 1 = 15 + 1 = 16$. Sortie : 16×16.

2. Pour les cas b et c : que remarque-t-on sur la taille de sortie par rapport à l'entrée ? Comment appelle-t-on ce type de padding ?

✓

Dans les deux cas, la sortie a la même taille que l'entrée (10×10). On parle de padding **same** : le padding est choisi précisément pour que out = in, ce qui est pratique pour empiler des couches sans réduire la résolution spatiale.

Ex 12 **Interprétation de filtres classiques** ★ ★ kR9v

On donne les quatre noyaux suivants :

$$K_A = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad K_B = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix},$$

$$K_C = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix}, \quad K_D = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}.$$

1. Associer chaque noyau à l'un des rôles suivants : **lissage**, **détection de contours tous azimuts**, **détection de contours verticaux (Sobel X)**, **Laplacien discret**.

✓

— K_A : **lissage** (filtre moyenneur uniforme).
— K_B : **détection de contours tous azimuts** (filtre laplacien centré sur 8, réagit dans toutes les directions).
— K_C : **détection de contours verticaux** (filtre de Sobel X, sensible aux variations horizontales d'intensité).
— K_D : **Laplacien discret** (approximation de $\nabla^2 I$ avec les 4-voisins).

2. Pour K_A : calculer la somme de ses coefficients. Qu'implique ce résultat sur les zones uniformes de l'image ?

✓ $\sum K_A = 9 \times \frac{1}{9} = 1$. Sur une zone uniforme (tous les pixels égaux à une constante c), la convolution donne $c \times 1 = c$: l'intensité est préservée. Le filtre ne crée ni n'annule de luminosité ; c'est un filtre **passe-bas normalisé**.

3. Pour K_B : calculer la somme de ses coefficients. Qu'implique ce résultat sur les zones uniformes ?

✓ $\sum K_B = 8 + 8 \times (-1) = 8 - 8 = 0$. Sur une zone uniforme, la convolution donne 0 : les zones plates apparaissent en noir dans la carte de sortie. Seules les transitions d'intensité produisent une réponse non nulle ; c'est caractéristique d'un filtre **passe-haut** ou détecteur de contours.

4. Pour K_C : si on l'applique à une image dont toutes les colonnes sont identiques, que vaut la sortie ? Pourquoi ?

✓ Si toutes les colonnes sont identiques, l'intensité ne varie pas horizontalement. Or K_C calcule $-I(\cdot, j-1) + I(\cdot, j+1)$ (pondéré), ce qui est une différence entre colonne de droite et colonne de gauche. Cette différence est nulle si toutes les colonnes sont égales. La sortie est donc entièrement **nulle**. K_C ne réagit qu'aux variations horizontales d'intensité (i.e. aux contours verticaux).

5. Calculer $K_B + 9K_A$ (addition élément par élément). Interpréter le résultat.

✓
$$K_B + 9K_A = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 0 \end{pmatrix}.$$

On obtient 9 fois le filtre identité. Cela illustre que K_B peut s'interpréter comme « identité (mise à l'échelle) − filtre moyenneur » : il soustrait la moyenne locale pour ne conserver que les variations.

Ex 13 — Effet du stride sur la taille et la résolution spatiale

nT2w

★ ★

On dispose d'une image 6×6 et d'un noyau moyenneur 3×3 (tous les coefficients valent $1/9$), avec padding $p = 1$.

1. Calculer la taille de sortie pour les strides $s = 1$, $s = 2$ et $s = 3$.

✓ Avec in $= 6$, $p = 1$, $k = 3$:
- $s = 1$: $\lfloor (6 + 2 - 3)/1 \rfloor + 1 = 5 + 1 = 6$.
- $s = 2$: $\lfloor (6 + 2 - 3)/2 \rfloor + 1 = \lfloor 5/2 \rfloor + 1 = 2 + 1 = 3$.
- $s = 3$: $\lfloor (6 + 2 - 3)/3 \rfloor + 1 = \lfloor 5/3 \rfloor + 1 = 1 + 1 = 2$.

2. Pour stride $= 2$: combien de fois le noyau se déplace-t-il en ligne ? En colonne ?

✓ La sortie est 3×3 : le noyau occupe 3 positions distinctes en ligne et 3 en colonne. Il effectue donc $3 \times 3 = 9$ placements au total (contre 36 pour stride $= 1$).

3. Deux pixels voisins de l'image originale (distance 1) correspondent-ils toujours à deux pixels voisins dans la sortie avec stride = 2 ? Qu'est-ce que cela implique sur la résolution spatiale ?

✓ Non. Avec stride = 2, deux positions consécutives du noyau sont séparées de 2 pixels dans l'image d'origine. Un pixel sur deux de l'entrée est donc « sauté ». La résolution spatiale est divisée par 2 : des structures plus fines que 2 pixels ne peuvent plus être distinguées dans la sortie. C'est un phénomène de **sous-échantillonnage** (aliasing possible si le signal n'est pas suffisamment lisse).

4. On souhaite obtenir une sortie de taille 3×3 à partir d'une image 7×7, avec un noyau 3×3 et **sans padding**. Quel stride faut-il utiliser ?

✓ On cherche s tel que $\lfloor (7 + 0 - 3)/s \rfloor + 1 = 3$, soit $\lfloor 4/s \rfloor = 2$, donc $2 \leqslant 4/s < 3$, ce qui donne $4/3 < s \leqslant 2$. Le seul entier dans cet intervalle est $s = 2$.
Vérification : $\lfloor 4/2 \rfloor + 1 = 2 + 1 = 3$.

Ex 14 **Noyaux séparables et gain de calcul** ⋆ ⋆ ⋆ sB4q

Un noyau K de taille $k \times k$ est dit **séparable** s'il peut s'écrire comme le produit extérieur de deux vecteurs :

$$K = \mathbf{v} \cdot \mathbf{h}^\top,$$

où $\mathbf{v} \in \mathbb{R}^k$ est un vecteur colonne et $\mathbf{h} \in \mathbb{R}^k$ un vecteur ligne. Dans ce cas, appliquer K revient à effectuer deux convolutions 1D successives : d'abord $\mathbf{h}$ horizontalement, puis $\mathbf{v}$ verticalement. On considère le noyau gaussien 3×3 :

$$K_g = \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}.$$

1. Vérifier que K_g est séparable en trouvant $\mathbf{v}$ et $\mathbf{h}$ tels que $K_g = \mathbf{v} \cdot \mathbf{h}^\top$.

✓ On pose $\mathbf{v} = \frac{1}{4}\begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$ et $\mathbf{h} = \frac{1}{4}\begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$. Alors :

$$\mathbf{v}\mathbf{h}^\top = \frac{1}{16}\begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}\begin{pmatrix} 1 & 2 & 1 \end{pmatrix} = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix} = K_g. \quad \checkmark$$

2. Soit une image de taille $N \times N$ et un noyau de taille $k \times k$. Comparer le nombre de multiplications pour appliquer K_g en 2D directement versus en deux passes 1D successives. Quel est le gain ?

✓

— **Convolution 2D directe :** pour chaque pixel de sortie, on effectue k^2 multiplications. Il y a N^2 pixels. Coût total : N^2k^2.
— **Deux passes 1D :** la première passe (horizontale, noyau $1 \times k$) coûte N^2k multiplications ; la seconde passe (verticale, noyau $k \times 1$) coûte également N^2k. Coût total : $2N^2k$.

Le rapport est $\frac{k^2}{2k} = \frac{k}{2}$. Pour $k = 3$, on divise le nombre de multiplications par $3/2 \approx 1{,}5$; pour $k = 7$ (gaussien large), par 3,5. Le gain est d'autant plus important que k est grand.

3. Le noyau Sobel $K_C = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix}$ est-il séparable ? Si oui, trouver sa décomposition.

✓

Les lignes de K_C sont $(1, 0, +1)$, $(2, 0, +2)$ et $(1, 0, +1)$, qui sont toutes proportionnelles au vecteur $(1, 0, +1)$. Le noyau est donc de rang 1, i.e. séparable.

On pose $\mathbf{h}^\top = \begin{pmatrix} -1 & 0 & +1 \end{pmatrix}$ et $\mathbf{v} = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix}$. Alors :

$$\mathbf{v}\mathbf{h}^\top = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \begin{pmatrix} -1 & 0 & +1 \end{pmatrix} = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} = K_C. \quad \checkmark$$

Ex 15 **Composition de convolutions et champ réceptif** cF8j

★ ★ ★

On empile deux couches de convolution successives, chacune avec un noyau 3×3, padding $= 0$ et stride $= 1$.

1. Après la première couche, quelle est la taille de la sortie si l'entrée est 7×7 ?

✓

$\lfloor (7 + 0 - 3)/1 \rfloor + 1 = 5$. Sortie : 5×5.

2. Après la deuxième couche, quelle est la taille finale ?

✓

$\lfloor (5 + 0 - 3)/1 \rfloor + 1 = 3$. Sortie finale : 3×3.

3. Quel est le champ réceptif d'un pixel de la sortie finale (c'est-à-dire la zone de l'image d'entrée qui a influencé sa valeur) ? Montrer le raisonnement.

✓

Un pixel de la sortie de la couche 2 dépend d'une fenêtre 3×3 de la sortie de la couche 1. Chaque pixel de cette fenêtre 3×3 dépend lui-même d'une fenêtre 3×3 de l'image d'entrée. La fenêtre totale couverte dans l'entrée est donc de taille $(3 - 1) + (3 - 1) + 1 = 5$, soit un champ réceptif de $\mathbf{5 \times 5}$.
Formule générale pour L couches de noyaux $k \times k$ (stride $= 1$, pas de padding) :

$$\text{champ réceptif} = L(k - 1) + 1.$$

Ici : $2(3 - 1) + 1 = 5$.

4. Comparer avec une unique convolution 5×5 appliquée directement sur l'image 7×7 (padding $= 0$). Même champ réceptif ? Même taille de sortie ?

✓

Taille de sortie : $\lfloor (7 + 0 - 5)/1 \rfloor + 1 = 3 \times 3$. Identique.
Champ réceptif : 5×5. Identique.
Les deux architectures voient la même zone de l'image pour chaque pixel de sortie.

5. Combien de paramètres comporte chaque approche (sans biais) ? Quelle configuration est plus économique ?

✓

— **Deux couches** 3×3 (1 seul filtre par couche, pour simplifier) : $3^2 + 3^2 = 9 + 9 = 18$ paramètres.
— **Une couche** 5×5 : $5^2 = 25$ paramètres.

Deux couches 3×3 sont donc plus économiques ($18 < 25$) tout en couvrant le même champ réceptif. C'est l'une des raisons pour lesquelles les architectures modernes (VGG, ResNet) privilégient des petits noyaux empilés plutôt que de grands noyaux uniques.

Ex 16 **Convolution 2D complète à la main avec padding same** mH6z

★ ★ ★

On considère l'image I et le noyau K suivants :

$$I = \begin{pmatrix} 1 & 0 & 1 & 2 \\ 3 & 1 & 0 & 1 \\ 2 & 2 & 1 & 0 \\ 0 & 1 & 3 & 1 \end{pmatrix}, \qquad K = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}.$$

On applique ce filtre en mode **same** (padding $p = 1$) avec stride $= 1$.

1. Écrire l'image paddée I_p (6×6 avec des zéros sur les bords).

✓

$$I_p = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 2 & 0 \\ 0 & 3 & 1 & 0 & 1 & 0 \\ 0 & 2 & 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

2. Rappeler le noyau renversé $K' = \text{flip}(K)$. Que remarque-t-on ?

✓

Le retournement (rotation de 180°) de K donne :

$$K' = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}.$$

On remarque que $K' = -K$: le noyau est **antisymétrique par rotation de** 180°. Cela implique que la convolution $I \star K$ et la corrélation croisée $I \star\star K$ diffèrent d'un signe.

3. Calculer les 4 coefficients de la ligne du milieu de la sortie, aux positions $(1, 0)$, $(1, 1)$,

$(1,2)$, $(1,3)$ (indexation ligne/colonne à partir de 0).

✓ On travaille sur I_p et on extrait les fenêtres 3×3 centrées sur les pixels $(1,0)$ à $(1,3)$ de I (ce qui correspond aux lignes d'indices 1 à 3 de I_p grâce au décalage de $p = 1$). Le produit scalaire se fait avec K'.

Position $(1,0)$ — fenêtre $I_p[1:4,\ 0:3]$:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 3 & 1 \\ 0 & 2 & 2 \end{pmatrix} \odot \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} = 0+0+0+0+0+2+0+0+2 = 4.$$

Position $(1,1)$ — fenêtre $I_p[1:4,\ 1:4]$:

$$\begin{pmatrix} 1 & 0 & 1 \\ 3 & 1 & 0 \\ 2 & 2 & 1 \end{pmatrix} \odot K' = (-1)+0+1+(-6)+0+0+(-2)+0+1 = -7.$$

Position $(1,2)$ — fenêtre $I_p[1:4,\ 2:5]$:

$$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} \odot K' = 0+0+2+(-2)+0+2+(-2)+0+0 = 0.$$

Position $(1,3)$ — fenêtre $I_p[1:4,\ 3:6]$:

$$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \odot K' = (-1)+0+0+0+0+0+(-1)+0+0 = -2.$$

Ligne du milieu de la sortie : $\begin{pmatrix} 4 & -7 & 0 & -2 \end{pmatrix}$.

4. Interpréter le résultat : ce filtre est-il symétrique ? Que détecte-t-il ?

✓ K n'est pas symétrique : il vaut $+1$ à gauche et -1 à droite (colonnes). C'est le filtre de **Sobel X** : il calcule approximativement la dérivée partielle horizontale $\partial I/\partial x$. Une valeur positive indique une transition sombre → clair de gauche à droite ; une valeur négative indique clair → sombre. Il détecte donc les **contours verticaux** (variations horizontales d'intensité).

Ex 17 **Équivariance par translation et max pooling** eT5r

★ ★ ★ ★

Soit I une image discrète et $T_d(I)$ l'image I translatée de d pixels vers la droite, définie par $[T_d(I)](i,j) = I(i, j-d)$. On note $\star$ la convolution 2D discrète :

$$(I \star K)(i,j) = \sum_{m,n} I(i-m,\ j-n)\, K(m,n).$$

1. Démontrer formellement que $T_d(I) \star K = T_d(I \star K)$.

✓

$$[T_d(I) \star K](i,j) = \sum_{m,n} [T_d(I)](i-m,\, j-n)\, K(m,n)$$
$$= \sum_{m,n} I(i-m,\, j-n-d)\, K(m,n).$$

D'autre part :

$$[T_d(I \star K)](i,j) = (I \star K)(i,\, j-d) = \sum_{m,n} I(i-m,\, (j-d)-n)\, K(m,n) = \sum_{m,n} I(i-m,\, j-n-d)\, K(m,n).$$

Les deux expressions sont égales, donc $T_d(I) \star K = T_d(I \star K)$. □

2. Pourquoi cette propriété est-elle précieuse pour la reconnaissance d'objets dans une image ?

✓

L'équivariance par translation garantit que si un objet est décalé dans l'image d'entrée, sa *représentation* (carte d'activation) est décalée de la même façon dans la sortie. Le réseau « voit » le même motif quelle que soit sa position : il n'est pas nécessaire d'apprendre séparément le même filtre pour chaque position possible de l'objet. Cela réduit drastiquement le nombre de paramètres nécessaires et favorise la généralisation.

3. Le max pooling brise-t-il cette propriété d'équivariance ? Illustrer sur un exemple 1D avec un signal $[0, 1, 0, 3]$, un signal translaté d'un pixel $[0, 0, 1, 0]$ (bord droit ignoré), et pool size = 2.

✓

Signal $s_1 = [0, 1, 0, 3]$:
- Fenêtre $[0, 1]$: max = 1.
- Fenêtre $[0, 3]$: max = 3.

Sortie : $[1, 3]$.
Signal translaté $s_2 = [0, 0, 1, 0]$ (translation de 1 pixel) :
- Fenêtre $[0, 0]$: max = 0.
- Fenêtre $[1, 0]$: max = 1.

Sortie : $[0, 1]$.
La sortie de s_1 poolée était $[1, 3]$; celle de s_2 est $[0, 1]$, qui n'est pas simplement une translation de $[1, 3]$. **L'équivariance exacte est donc brisée** par le max pooling.

4. Malgré cette rupture, pourquoi dit-on que le max pooling confère une certaine **invariance** par translation ? Distinguer équivariance et invariance.

✓

- **Équivariance** : une transformation de l'entrée produit une transformation *identique* de la sortie. La convolution est équivariante.
- **Invariance** : une transformation de l'entrée ne change *pas* la sortie. Le max pooling tend vers l'invariance.

Le max pooling agrège sur des fenêtres locales : tant que l'activation maximale reste dans la *même fenêtre* après une petite translation, la sortie est inchangée. Il confère donc une invariance **locale** (à l'échelle de la fenêtre de pooling). Combiné à l'empilement de couches, cela produit une invariance croissante à mesure que l'on monte dans le réseau : les couches profondes deviennent peu sensibles à la position exacte des motifs.