Apprentissage statistique et réseaux de neurones

Chapitre 3: Réseaux de neurones convolutionnels.

Frédéric Richard
frederic.richard@univ-amu.fr

Master Mathématiques appliquées, statistique (1ère année), Parcours Data Science

2025

Outline

Apprentissage statistique.

- Hypothèse : observations $(x_i, y_i)_{i=1}^n$ sont des réalisations d'un échantillon i.i.d. selon la même loi de distribution inconnue P que des variables (X, Y) à valeurs dans $\mathcal{X} \times \mathcal{Y}$.
- Objectif : prédire Y à partir de X en utilisant une famille $\mathcal{F}$ de fonctions paramétrées par θ : pour $f_\theta \in \mathcal{F}$:
- Prévision de Y est donnée par $\widehat{Y} = f_\theta(X)$.
- Fonction de coût ℓ : mesure l'erreur de prévision de Y par $\widehat{Y}$.
- Dans l'absolu, on chercherait θ minimisant le risque théorique :

$$\mathbb{E}(\ell(f_\theta(X), Y)) = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f_\theta(x), y) dP(x, y).$$

- P étant inconnue, on minimise le risque empirique :

$$E_n(\ell(f_\theta(x), y)) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i).$$

Régression multinomiale.

- Y une v. a. à valeurs dans un espace de classes $[\![1, K]\!]$.
- $X = (X_j)_{j=1}^p$ v.a. à valeurs dans $\mathbb{R}^p$.
- Modèle de distribution :
  - Paramètres : $\theta = (\theta^{(k)})_{k=1}^K$ avec $\theta^{(k)} \in \mathbb{R}^{p+1}$.
  - Modèle de $\mathbb{P}(Y = k)$:

$$f_\theta(x) = \left(\frac{\exp\left(\theta_0^{(k)} + \sum_{j=1}^p \theta_j^{(k)} x_j\right)}{\sum_{m=1}^K \exp\left(\theta_0^{(m)} + \sum_{j=1}^p \theta_j^{(m)} x_j\right)} \right)_{k=1}^K .$$

- Entropie croisée :

$$\ell(f_\theta(x), y) = -\sum_{k=1}^K 1_{y=k} \log\left(f_\theta(x)_k\right).$$

Outline

Interprétation en termes de réseaux de neurones.

Modèle : $f_\theta(x) = \left(\frac{\exp\left(\theta_0^{(k)} + \sum_{j=1}^{p} \theta_j^{(k)} x_j\right)}{\sum_{m=1}^{K} \exp\left(\theta_0^{(m)} + \sum_{j=1}^{p} \theta_j^{(m)} x_j\right)} \right)_{k=1}^{K}$.

Ingrédients :

- Pour $x \in \mathbb{R}^p$ et $w \in \mathbb{R}^{p+1}$, $D(x; w) = w_0 + \sum_{j=1} w_j x_j$.
- Soft-max (K classes) : pour $z \in \mathbb{R}^K$, $\varphi(z) = \left(\frac{\exp(z_k)}{\sum_{m=1}^{K} \exp(z_m)} \right)_{k=1}^{K}$.

Réseau de neurones :

- Couche d'entrée : $x^{(0)} = x \in \mathbb{R}^p$.
- Couche dense à K neurones nécessitant $K\,(p+1)$ poids :

$$z^{(0)} = (z_k^{(0)})_{k=1}^{K} \text{ avec } z_k^{(0)} = D(x^{(0)}; \theta^{(k)}).$$

- Activation "soft-max" : $x^{(1)} = \varphi(z^{(0)})$.

Application à l'image.

Image discrète (en dimension 2) :

$$\mathrm{f} = (\mathrm{f}[\mathrm{m},\mathrm{n}], (\mathrm{m},\mathrm{n}) \in [\![0, \mathrm{N}_1 - 1\}] \times [\![0, \mathrm{N}_2 - 1]\!]) \,.$$

Pour appliquer la régression multinomiale, on vectorise I en définissant

$$\mathrm{x}_\mathrm{j} = \mathrm{f}[\mathrm{m},\mathrm{n}], \;\; \mathrm{j} = (\mathrm{m}-1)\mathrm{N}_2 + \mathrm{n} + 1,$$

pour $\mathrm{j} = [\![1, \mathrm{p}]\!]$ avec $\mathrm{p} = \mathrm{N}_1\,\mathrm{N}_2$.
Inconvénients de l'application de la régression multinomiale à l'image $\mathrm{x} = (\mathrm{x}_\mathrm{j})_{\mathrm{j}=1}^{\mathrm{p}}$:

- La dimension de p est très grande (typiquement 2^{16} pour des images de taille standard 256×256). Le nombre de paramètres sur la couche dense (en $\mathrm{O}(\mathrm{p}+1)$) est trop important.
- La vectorisation fait perdre l'information spatiale qui est véhiculée par les indices (m,n) de l'image I.

La couche de convolution.

Pour définir la première couche on peut remplacer l'opération $D(x; w) = w_0 + \sum_{j=1} w_j x_j$ par une convolution $C(I; w) = I * w$ de l'image I avec un noyau w à support $[\![-M_1, M_1]\!] \times [\![-M_2, M_2]\!]$: Pour tout $(m, n) \in [\![0, N_1 - 2M_1]\!] \times [\![0, N_2 - 2M_2]\!]$,

$$C(f; w)[m, n] = \sum_{u=-M_1}^{M_1} \sum_{v=-M_2}^{M_2} I[m + M_1 - u, n + M_2 - v] w[u, v].$$

Avantages :

- l'opération ne nécessite que $(2M_1 + 1)(2M_2 + 1)$ paramètres
- et tient compte de la structure spatiale de l'image.
- $C(f; w)[m, n]$ s'inteprête en termes de présence du motif w dans le voisinage de (m, n).

Réseau avec une couche de convolution.

- Couche d'entrée : $f^{(0)} = f$, image de taille $N \times N$.
- Couche de convolutions avec J filtres de support $[\![-M, M]\!]^2$:

$$\forall\ j \in [\![1, J]\!],\ \ f^{(1,j)} = C(f^{(0)}; w^{(j)}),$$

$f^{(1,j)}$: image de taille de l'ordre de N^2. $f^{(1)} = (f^{(1,j)})_{j=1}^{J}$: carte de caractéristiques (feature map).

- Aplatissement de $f^{(1)}$ en un vecteur x de taille $p \simeq J * N^2$
- Couche dense à K neurones :

$$z = (z_k)_{k=1}^{K} \text{ avec } z_k = D(x; \theta^{(k)}).$$

- Activation "soft-max" : $x^{(1)} = \varphi(z^{(0)})$.
- Bilan des paramètres :
  - Couche de convolution : $J \times (M+1)^2$.
  - Couche dense : $J \times N^2 \times K$, toujours trop !

Sous-échantillonage.

- Couche d'entrée : $f^{(0)} = f$, image de taille $N \times N$.
- Couche de convolutions avec J filtres de support $[\![-M, M]\!]^2$: $f^{(1,j)}$.
- Sous-échantillonage d'un facteur Q :

$$\forall (m, n) \in [\![0, N/Q]\!]^2, g^{(1,j)}[m, n] = f^{(1,j)}[Qm, Qn].$$

- Vectorisation de $g^{(1)}$ en x, de taille approximative $p = \left(\frac{N}{Q}\right)^2$
- Couche dense à K neurones.
- Activation "soft-max" : $x^{(1)} = \varphi(z^{(0)})$.
- Bilan des paramètres :
  - Couche de convolution : $J \times (M+1)^2$.
  - Couche dense : $J \times \left(\frac{N}{Q}\right)^2 \times K$, réduction d'un facteur Q^2.

Pooling.

Après la couche de convolution, on peut faire un pooling :

- Average pooling : moyenne locale
  Pour tout $j \in [\![1, J]\!]$ et $(m, n) \in [\![0, N-1]\!]^2$,

$$\tilde{g}^{(1,j)}[m, n] = \frac{1}{P^2} \sum_{u,v=[\![-P/2,P/2-1]\!]} f^{(1,j)}[m+u, n+u].$$

  inteprétation : filtre linéaire passe-bas.

- Max pooling : maxima locaux :
  Pour tout $j \in [\![1, J]\!]$ et $(m, n) \in [\![0, N-1]\!]^2$,

$$\tilde{g}^{(1,j)}[m, n] = \max_{u,v=[\![-P/2,P/2-1]\!]} \left| f^{(1,j)}[m+u, n+v] \right|.$$

  interprétation : filtre non-linéaire.

- Puis, sous-échantillonnage d'un facteur Q (en général, P = Q).

Comment multiplier les couches ?

- Succession de U couches convolutionnelles :
pour $\mathrm{u} = [\![1, \mathrm{U}]\!]$,

$$\forall\ \mathrm{j} \in [\![1, \mathrm{J_u}]\!],\ \ \mathrm{f}^{(\mathrm{u,j})} = \mathrm{C}(\mathrm{f}^{(\mathrm{u}-1,\boldsymbol{\kappa}(\mathrm{u,j}))}; \mathrm{w}^{(\mathrm{u,j})}),$$

avec des noyaux $\mathrm{w}^{(\mathrm{u,j})}$ de filtres de support $[\![-\mathrm{M_u}/2, \mathrm{M_u}/2 - 1]\!]^2$.

- Cela peut se ramener à un réseau à une seule couche (avec des noyaux plus grand).

- On rompt avec la structure linéaire en ajoutant des fonctions d'activation non linéaires :

$$\forall\ \mathrm{j} \in [\![1, \mathrm{J_u}]\!],\ \ \mathrm{f}^{(\mathrm{u,j})} = \boldsymbol{\varphi}^{(\mathrm{u})}\left(\mathrm{C}(\mathrm{f}^{(\mathrm{u}-1,\boldsymbol{\kappa}(\mathrm{u,j}))}; \mathrm{w}^{(\mathrm{u,j})})\right),$$

où $\boldsymbol{\varphi}^{(\mathrm{u})}$ est une fonction non linéaire (par ex., $\tanh(\mathrm{t})$).

Succession typique de couches convolutionnelles.

Pour $u = [\![1, U]\!]$:

$$\forall\ j \in [\![1, J_u]\!],\ \ f^{(u,j)} = \mathrm{Pool}_{Q_u}\left(\varphi^{(u)}\left(C(f^{(u-1,\kappa(u,l))}; w^{(u,j)})\right)\right).$$

- $C(f^{(u-1,\kappa(u,l))}; w^{(u,j)})$ convolution de l'image $f^{(u-1,\kappa(u,l)}$ de la couche précédente avec le filtre de noyau $w^{(u,j)}$.
- $\varphi^{(u)}$: activation non linéaire.
- Pool_{Q_u} : pooling et sous-échantillonage de facteur Q_u.

Bilan paramètres de la couche : $J_u \times M_u^2$ (J_u nombre de filtres, M_u^2 taille des filtres).

Taille des images $f^{(u,j)}$: $N_u \times N_u$ où $N_u = N_{u-1}/Q_u$.

Taille du tenseur à l'issue de la couche u : $J_u \times \left(\frac{N_{u-1}}{Q_u}\right)^2$.

Exemple : réseau LeNet (1998).

Classification de la base MNIST (http ://yann.lecun.com/).

Type	Cartes	Image	Noyau	Pas	Activ.
Entrée	1	32x32	-	-	-
Conv.	6	28x28	5x5	1	tanh
Av. Pooling	6	14x14	2x2	2	tanh
Conv.	16	10x10	5x5	1	tanh
Av. Pooling	16	5x5	2x2	2	tanh
Conv.	120	1x1	5x5	1	tanh
Dense	-	84	-	-	tanh
Dense	-	10	-	-	softmax

Outline

Problème d'optimisation.

- Base d'apprentissage : observations $(x_i, y_i)_{i=1}^n$.
- Prédiction de y_i par x_i obtenue sur la dernière couche $\hat{y}_i(w) = f_i^{(U)}$ d'un réseau défini de manière récursive :
  Couche d'entrée : $f_i^{(0)} = x_i$ et, pour $u = [\![1, U]\!]$,

$$\begin{cases} g_i^{(u)} &= S^{(u)}(f_i^{(u-1)}; w^{(u)}), \\ f_i^{(u)} &= \varphi^{(u)}(g_i^{(u)}). \end{cases}$$

  où $S^{(u)}$ est une couche dense ou une couche de convolution, $w^{(u)}$ sont les poids de cette couche, et $\varphi^{(u)}$ une activation.
- Fonction de coût :

$$E_n(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \hat{y}_i(w)).$$

Algorithme d'optimisation.

- Fonction à minimiser : $E_n(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \hat{y}_i(w))$.
- Descente de gradient : une itération t

$$w(t+1) = w(t) - \frac{\rho_t}{n}\sum_{i=1}^{n} \nabla\ell(y_i, \hat{y}_i(w)),$$

- Descente de gradient stochastique par mini-lots (batchs).

Une époque (epochs) $e+1$:

- Initialisation : $\tilde{w}(1) = w(e)$ puis pour les itérations $t = [\![1, T-1]\!]$,

$$\tilde{w}(t+1) = \tilde{w}(t) - \frac{\rho_e}{m}\sum_{i\in B_w(t)} \nabla\ell(y_i, \hat{y}_i(\tilde{w}(t))),$$

où, à t fixé, les $B_u(t)$ sont des ensembles disjoints de m indices tirés aléatoirement dans $[\![1, n]\!]$.

- $w(e+1) = \tilde{w}(T)$

Différentielle de la fonction de coût.

$$E_n(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_i^{(U)}) \quad \text{avec} \quad \left\{ \begin{array}{rcl} g_i^{(u)} & = & S^{(u)}(f_i^{(u-1)}; w^{(u)}), \\ f_i^{(u)} & = & \varphi^{(u)}(g_i^{(u)}). \end{array} \right.$$

$$\frac{dE_n}{dw^{(v)}}(w) = \frac{1}{n}\sum_{i=1}^{n} \frac{d\ell}{dz}\left(y_i, \frac{df_i^{(L)}}{dw^{(v)}}\right).$$

$$\frac{df_i^{(u)}}{dw^{(v)}} = d\varphi^{(u)}(g_i^{(u)}) \ \frac{dg_i^{(u)}}{dw^{(v)}}.$$

$$\frac{dg_i^{(u)}}{dw^{(v)}} = \left\{ \begin{array}{lll} S^{(u)}(f_i^{(u-1)}; \cdot) & \text{si} & u = v, \\ S^{(u)}\left(\frac{df_i^{(u-1)}}{dw^{(v)}}; w^{(u)}\right) & \text{si} & v > u, \\ 0 & \text{si} & v < u. \end{array} \right.$$

Phénomène de disparition du gradient.

$$u = [\![1, U]\!], \quad \left|\frac{df_i^{(u)}}{dw^{(v)}}\right| \leq \left|d\varphi^{(u)}(g_i^{(u)})\right| \left|\frac{df_i^{(u-1)}}{dw^{(v)}}\right| \left|w^{(u)}\right|.$$

- La norme de la différentielle n'est pas du même ordre à toutes les couches.
- Certaines fonctions d'activations $\varphi^{(u)}$ peuvent avoir un effet saturant et annuler les différentielles ($|d\varphi^{(u)}(g_i^{(u)})| \simeq 0$).
- Conséquence : Apprentissage très lent des couches hautes.

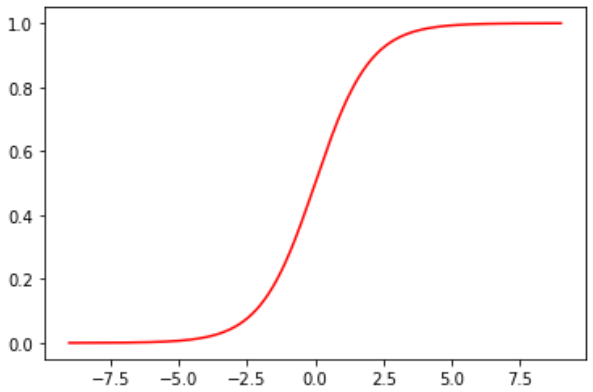
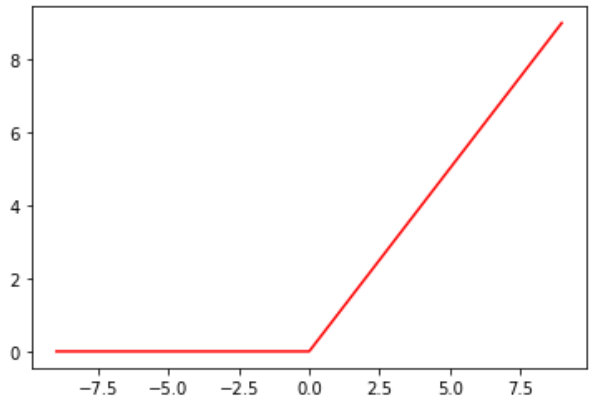


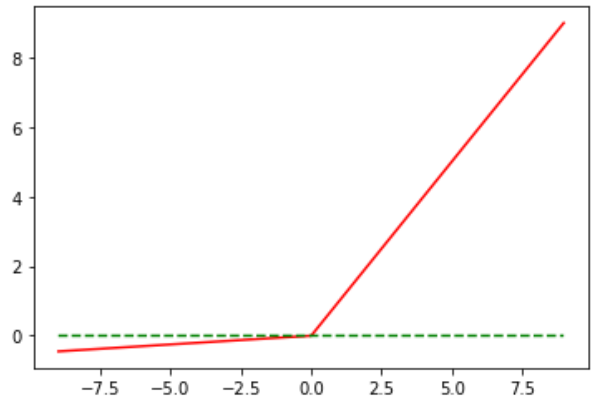
Figure – Effet saturant de la fonction d'activation logistique

Fonctions d'activation non saturantes.



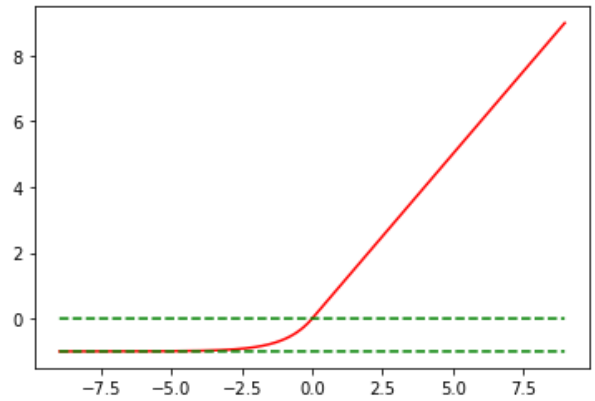
ReLU,
Glorot et Bengio, 2010

$\varphi(t) = \max(0, t)$



leaky ReLU(α),
ex. $\alpha = 0.01$

$\varphi(t) = \max(\alpha t, t)$



ELU(α)
Clevert et al., 2015

$\varphi(t) = \alpha(\exp(t) - 1)$
si $t < 0$ et t sinon

Normalisation par lot (Batch normalisation).

- Problème de l'"internal covariate shift" : statistiques des sorties d'une couche varient en fonction de la couche précédente.
- Solution (Ioffe et Szegedy, 2015) : centrage et normalisation des sorties de chaque couche de convolution (ou dense).

Sorties de la uième couche de convolution :

$$g_i^{(u)} = S^{(u)}(f_i^{(u-1)}; w^{(u)}).$$

Estimation de l'espérance et de la variance de $g^{(u)}$ sur un lot B :

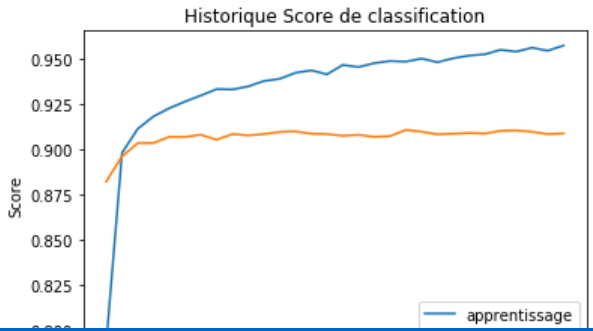
$$\hat{\mu}_B^{(u)} = \frac{1}{|B|}\sum_{i\in B} g_i^{(u)} \quad \text{et} \quad \hat{v}_B^{(u)} = \frac{1}{|B|}\sum_{i\in B}(g_i^{(u)} - \hat{\mu}_B^{(u)})^2.$$

Centrage et normalisation des sorties :

$$\tilde{g}_i^{(u)} = \frac{1}{\sqrt{\hat{v}_B^{(u)}) + \varepsilon}}(g_i^{(u)} - \hat{\mu}_B^{(u)}).$$

Problème du surajustement

- Sur-ajustement : le modèle appris est bien ajusté aux données d'apprentissage mais donne des prévisions médiocre lorsqu'appliqué à des données nouvelles.
- Problème statistique qui vient du fait que la complexité du modèle (nombre de paramètres) est trop grand par rapport aux données d'apprentissage.
- Le sur-ajustement peut se détecter lors de l'optimisation sur des données de validation.
- Il se diagnostique également après l'optimisation sur des données de test.



Régularisation.

- Pénalisation du problème d'optimisation :

$$\tilde{E}_\lambda(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \hat{y}_i) + \lambda R(w),$$

où $R(w)$ pénalise les grandes valeurs de w (par exemple, $R(w) = |w|_2^2$ ou $R(w) = |w|_1$).

- En contraignant les solutions, la pénalisation réduit la complexité du modèle et peut atténuer le sur-ajustement.
- Elle permet aussi de mieux poser le problème d'optimisation en termes d'existence et de stabilité des solutions.
- En keras, les options de régularisation se mettent sur les couches.

Dropout.

- Hinton, 2012 et Srivastava et al. 2014.
- Principe : Au cours de l'optimisation, on gêle aléatoirement des sorties de couches pour se concentrer sur l'apprentissage d'un sous-modèle (sous-graphe) du réseau.
- Sur des couches u (sauf la dernière), on tire un échantillon i.i.d $\mathrm{b}^{(\mathrm{u})}$ de loi de Bernoulli $\mathcal{B}(\mathrm{p})$ de même dimension que les sorties $\mathrm{g}_{\mathrm{i}}^{(\mathrm{u})}$. Puis on effectue les opérations :

$$\left\{\begin{array}{lcl} \mathrm{g}_{\mathrm{i}}^{(\mathrm{u})} & = & \mathrm{S}^{(\mathrm{u})}(\mathrm{f}_{\mathrm{i}}^{(\mathrm{u}-1)}; \mathrm{w}^{(\mathrm{u})}), \\ \tilde{\mathrm{g}}_{\mathrm{i}}^{(\mathrm{u})} & = & \mathrm{g}_{\mathrm{i}}^{(\mathrm{u})} \otimes \mathrm{b}^{(\mathrm{u})}, \\ \mathrm{f}_{\mathrm{i}}^{(\mathrm{u})} & = & \boldsymbol{\varphi}^{(\mathrm{u})}(\tilde{\mathrm{g}}_{\mathrm{i}}^{(\mathrm{u})}). \end{array}\right.$$

où $\otimes$ est une produit point par point.

Autres techniques pour atténuer le sur-ajustement.

- Augmentation des données : ajouter des transformations des données d'apprentissage (translation, rotation, changement de contraste,...), le plus souvent à la volée pendant l'apprentissage (pour éviter d'utiliser trop de stockage.
- Ré-utilisation des couches convolutionnelles (représentation de l'image) d'un réseau déjà appris sur une base d'images pour classification d'images d'une autre base (en général, plus petite). Fine-tuning lorsque modification de certaines couches convolutionnelles lors de l'apprentissage du nouveau classifieur.