De la rétropropagation aux GANs : exploration des réseaux de neurones en reconnaissance d'images

Jules Marceau

Table des matières

Introduction

La reconnaissance de motifs sur image pose un problème algorithmique important pour obtenir des résultats convaincants tout en conservant un coût temporel et spatial raisonnable en raison de la possible grande variabilité des données et des faibles nuances à déterminer pour bien les départager. C'est sur ces problèmes que l'architecture en réseaux de neurones et la descente de gradient ont démontré de bonnes capacités d'apprentissage.

Nous nous concentrerons sur la reconnaissance de caractères en partant d'un réseau de neurones dense puis chercherons à améliorer nos résultats par la convolution et, finalement, la génération synthétique de données par réseaux antagonistes génératifs.

1 Réseaux de neurones

Les réseaux de neurones sont une généralisation du modèle de perceptron, un classificateur binaire linéaire. Ce sont une liste de couches contenant des nœuds et marqués par des réels.

On définit : $\left(a_i^{(\ell)}\right)_{l,i}$ les nœuds i de la couche ℓ, $\left(w_{i,j}^{(\ell)}\right)_{l,i,j}$ les poids reliant le nœud j de la couche ℓ au nœud i de la couche ℓ, et $b_i^{(\ell)}$ le biais associé au nœud $a_i^{(\ell)}$.

1.1 Propagation avant

L'algorithme de propagation avant définit l'obtention des nœuds internes jusqu'à la couche finale à partir des paramètres du réseau et d'une fonction d'activation :

$$\forall j,\ z_j^{(\ell+1)} = \sum_{i=1}^{n_l} w_{j,i}^{(\ell)}\, \sigma\big(z_i^{(\ell)}\big) + b_j^{(\ell)}.$$

La fonction d'activation permet au réseau d'approcher des fonctions non linéaires ; elle se doit donc elle-même d'être non linéaire. On peut toutefois entraîner un réseau de neurones linéaire sans fonction d'activation.

1.2 Descente de gradient

Maintenant que l'on dispose d'une structure pour approcher des fonctions, nous souhaitons obtenir les paramètres $w_{i,j}^{(\ell)}$ et $b_i^{(\ell)}$ qui, pour toute entrée x_i de notre jeu de données, fourniront la classification adéquate.

La fonction de coût C est définie de sorte que l'on puisse objectivement juger la pertinence de nos paramètres. Le principe de la descente de gradient vient de l'approximation de la variation de cette fonction :

$$\Delta C(v_1, \ldots, v_k) \approx \nabla C \cdot (\Delta v_1, \ldots, \Delta v_k).$$

En choisissant $(\Delta v_1, \ldots, \Delta v_k) = -\eta\, \nabla C$, on obtient

$$\Delta C(v_1, \ldots, v_k) = -\eta\, \|\nabla C\|^2 < 0,$$

on a donc réduit l'erreur.

Pour la classification, l'erreur quadratique moyenne est communément utilisée :

$$C\big(a_1^{(L)}, \ldots, a_m^{(L)}\big) = \tfrac{1}{2} \sum_{i=1}^{m} \big(y_i - a_i^{(L)}\big)^2.$$

L'algorithme de rétropropagation [1] intervient pour déterminer le gradient d'une certaine fonction de coût en fonction de ses paramètres. En posant $\delta_i^{(\ell)} = \partial C / \partial z_i^{(\ell)}$, que l'on appelle *terme d'erreur*, on remarque que l'on sait calculer l'erreur à la couche de sortie ℓ :

$$z_i^{(\ell)} = \sum_{k=1}^{n_{l-1}} a_k^{(\ell-1)}\, w_{i,k}^{(\ell)} + b_i^{(\ell)}.$$

Pour le cas d'une fonction de coût quadratique, on obtient :

$$\delta_i^{(L)} = \big(a_i^{(L)} - y_i\big)\, \sigma'\big(z_i^{(L)}\big).$$

Par la règle de la chaîne, on a, pour remonter de la couche $l+1$ à ℓ :

$$\delta_i^{(\ell)} = \sum_{j=1}^{n_{l+1}} \delta_j^{(\ell+1)}\, w_{j,i}^{(\ell+1)}\, \sigma'\big(a_i^{(\ell)}\big).$$

Enfin, les gradients des paramètres sont :

$$\frac{\partial C}{\partial b_i^{(\ell)}} = \delta_i^{(\ell)} \quad \text{et} \quad \frac{\partial C}{\partial w_{i,j}^{(\ell)}} = a_j^{(\ell-1)}\, \delta_i^{(\ell)}.$$

On sait alors déterminer, en partant de la dernière couche, tous les gradients de nos fonctions.

Data: Jeu de données $\mathcal{D} = \{(x, y)\}$, taux d'apprentissage η
Result: Paramètres mis à jour $\{w_{i,j}^{(\ell)}, b_i^{(\ell)}\}$
initialisation;
foreach $(x, y) \in \mathcal{D}$ **do**
  propagation_avant(x);
  calculer $\delta^{(L)}$;
  for $l = L - 1$ **to** 1 **do**
    calculer $\delta^{(\ell)}$ à partir de $\delta^{(\ell+1)}$;
  end
  calculer $\dfrac{\partial C}{\partial w_{i,j}^{(\ell)}}, \dfrac{\partial C}{\partial b_i^{(\ell)}}$;
  foreach *couche* l, *neurones* i, j **do**
    $w_{i,j}^{(\ell)} \leftarrow w_{i,j}^{(\ell)} - \eta \frac{\partial C}{\partial w_{i,j}^{(\ell)}}$;
    $b_i^{(\ell)} \leftarrow b_i^{(\ell)} - \eta \frac{\partial C}{\partial b_i^{(\ell)}}$;
  end
end

Algorithm 1: Descente de gradient par rétropropagation

1.2.1 Résultats

L'architecture composée d'une seule couche cachée à 120 neurones semble optimale dans la reconnaissance de chiffres. Avec seulement 50 neurones, les résultats sur les données montrent une stagnation autour de 90 % probablement due à une incapacité à repérer suffisamment de caractéristiques par manque de paramètres. Les résultats croissent avec le nombre de passages sur la totalité de la base de données mais tendent aussi vers un plafond autour de 97 %.

Époque	**Précision (%)**
1	95.1
2	96.1
3	96.5

TABLE 1.1 – Précision obtenue pour différentes époques d'entraînement (réseau dense).

2 Convolution

Une manière d'améliorer plus encore les capacités de notre architecture est l'utilisation de couches convolutives basées sur l'opération de convolution définie de la manière suivante : pour un noyau k de taille $K \times K$, on a

$$y_{i,j} = \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} x_{i+u,\, j+v}\, k_{u,v}.$$

La capacité des convolutions à extraire des caractéristiques d'une image (notamment les bords des formes qui y sont présentes) et la conservation du lien spatial entre zones proches de l'image motivent son utilisation dans notre réseau [2].

Les couches convolutives sont définies comme suit : pour un réseau profond, chaque carte de caractéristiques (feature map) à la couche ℓ et canal p s'écrit

$$x^{(\ell)}_{p,i,j} = \sum_{p'=1}^{P'} \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} w^{(\ell)}_{p,\,p',\,u,\,v}\, x^{(\ell-1)}_{p',\,i+u,\,j+v} + b^{(\ell)}_{p},$$

avec :

- P' le nombre de canaux (profondeur) en entrée de la couche ℓ ;
- $w^{(\ell)}_{p,\,p',\,u,\,v}$ le filtre 4D reliant le canal p' de la couche ℓ au canal p de la couche ℓ ;
- k la taille spatiale du filtre ;
- $b^{(\ell)}_{p}$ le biais ajouté au canal p de la couche ℓ.

2.0.1 Rétropropagation

On veut calculer

$$\frac{\partial C}{\partial x^{(\ell)}_{p,i,j}} = \sum_{p'=1}^{P''} \sum_{i',j'} \frac{\partial C}{\partial x^{(\ell+1)}_{p',\,i',\,j'}}\, \frac{\partial x^{(\ell+1)}_{p',\,i',\,j'}}{\partial x^{(\ell)}_{p,\,i,\,j}}.$$

Or, dans une convolution :

$$x^{(\ell+1)}_{p',\,i',\,j'} = \sum_{p=1}^{P'} \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} w^{(\ell)}_{p',\,p,\,u,\,v}\, x^{(\ell)}_{p,\,i'+u,\,j'+v} + b^{(\ell)}_{p'}.$$

Ainsi,

$$\frac{\partial x^{(\ell+1)}_{p',\,i',\,j'}}{\partial x^{(\ell)}_{p,\,i,\,j}} = \begin{cases} w^{(\ell)}_{p',\,p,\,u,\,v}, & \text{si } i = i' + u \text{ et } j = j' + v,\ u, v \in \{0, \ldots, k-1\}, \\ 0, & \text{sinon.} \end{cases}$$

En posant $i' = i - u$ et $j' = j - v$, on obtient

$$\frac{\partial C}{\partial x^{(\ell)}_{p,i,j}} = \sum_{p'=1}^{P''} \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} \frac{\partial C}{\partial x^{(\ell+1)}_{p',\,i-u,\,j-v}}\, w^{(\ell)}_{p',\,p,\,u,\,v}.$$

On ajoute une couche de sous-échantillonnage qui réduit la dimension d'une couche par deux en sélectionnant, dans une zone de 2×2, le nœud maximum (max-pooling). Cela permet de concentrer les caractéristiques de la couche et ainsi de réduire le temps mémoire et calcul tout en forçant le réseau à extraire le plus important de la couche.

2.0.2 Résultats

Le gain apporté par cette technique est clairement visible et permet de monter jusqu'à plus de 98 % de réussite. Le coût en termes de calcul est toutefois devenu plus élevé et m'a motivé à implémenter l'algorithme de descente de gradient en programmation parallèliste (*multi-threading*), ce qui a diminué par un facteur de 2 à 3 le temps d'entraînement.

Époque	**Précision (%)**
1	96.7 (+1.6)
2	97.6 (+1.5)
3	98.0 (+1.5)

TABLE 2.1 – Précision obtenue pour différentes époques d'entraînement (réseau convolutionnel).

3 Réseaux antagonistes génératifs

Il est généralement bénéfique d'étendre la base de données en ajoutant des rotations, du floutage ou même du bruit sur notre jeu de données pour rendre plus robuste le réseau. Toutefois, ces modifications demandent de l'intervention humaine, ce qui amoindrit le principe d'autonomie de l'apprentissage. Les réseaux antagonistes génératifs (GAN) tentent d'automatiser ce procédé en synthétisant automatiquement des données à partir d'exemples.

Le principe des réseaux antagonistes génératifs [3] repose sur l'affrontement de deux réseaux, un discriminateur et un générateur, qui s'affrontent dans un jeu à somme nulle pour chacun obtenir l'ascendant sur l'autre : voir la figure 3.1.

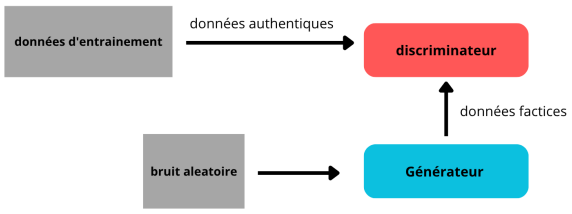

FIGURE 3.1 – Schéma de principe d'un réseau antagoniste génératif (GAN).

L'entraînement des réseaux se fait en deux phases :

1. Le discriminateur reçoit des images — synthétisées par le générateur ou issues du jeu de données réel — puis effectue une descente de gradient sur l'exemple en question pour progresser.
2. Les paramètres du discriminateur sont gelés. Sur une donnée factice passée pour véritable, on effectue une rétropropagation sur le discriminateur ; une fois la première couche d'erreur atteinte, on la donne comme dernière couche au réseau générateur dans une descente de gradient. Enfin, on dégèle le discriminateur.

La seconde étape revient à faire une rétropropagation sur un réseau étendu composé du générateur et du discriminateur et donc à motiver le générateur à berner le discriminateur.

3.1 Premiers résultats

On observe des formes puis une convergence vers le noir. Par ailleurs, le taux de succès du discriminateur au cours de l'entraînement est très élevé et avoisine constamment les 90 %. Cela indique probablement une domination du réseau discriminateur sur le réseau générateur : si le discriminateur a fini d'apprendre, son gradient est faible et le générateur n'apprend que très peu.

Entraîner le générateur plus que le discriminateur pourrait résoudre le problème. Changer les objectifs en passant de 0.1 pour une image factice et 0.9 pour une véritable image (plutôt que 0 et 1) pourrait aussi forcer le discriminateur à continuer d'apprendre et non à saturer les fonctions de normalisation et obtenir des résultats parfaits. Il semble également, empiriquement,

que la fonction d'activation `tanh` soit plus optimale que la `sigmoid` pour le générateur dans sa dernière couche.

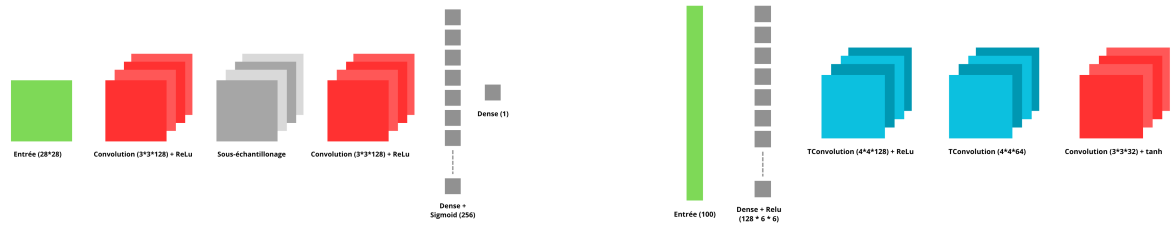

FIGURE 3.2 – Architectures du discriminateur (gauche) et du générateur (droite).

3.2 Nouveaux résultats

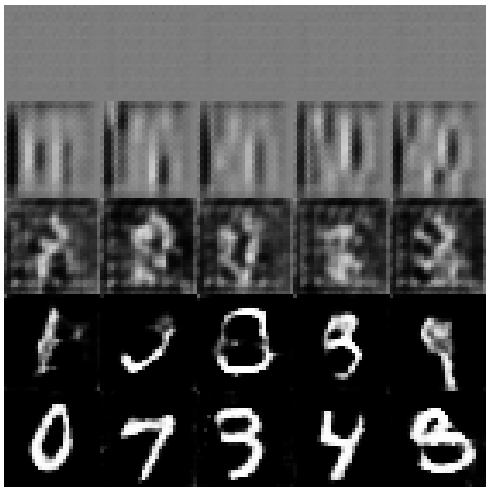

FIGURE 3.3 – Images générées après 160, 1200, 2300, 15000 et 40000 exemples vus.

Si l'on prend deux vecteurs aléatoires et que l'on génère une série de vecteurs issue d'une interpolation linéaire entre les deux, on obtient :

FIGURE 3.4 – Interpolation dans l'espace latent du GAN.

3.3 Explication

Le générateur crée des images à partir de vecteurs aléatoires dans $[-1, 1]^{100}$. Une fois entraîné, tout vecteur de cet espace doit être lié à un chiffre ; or les opérations sont presque toutes continues. On observe alors naturellement une transition de métamorphose où, à tout moment, l'image obtenue reste un chiffre entre les deux chiffres.

3.4 Impact sur la classification

Le dernier test consiste à observer l'effet des données générées synthétiquement par le GAN sur l'entraînement d'un classificateur convolutionnel de chiffres. On crée six bases de données avec des proportions variables de données synthétiques (obtenues par l'entraînement de 10 GAN spécialisés pour chaque chiffre). Les résultats sur ces bases de données sont présentés ci-dessous :

TABLE 3.1 – Précision (%) selon l'époque et la proportion de données synthétiques

Époque	**D1**	**D2**	**D3**	**D4**	**D5**	**D6**
1	97.17	96.80	96.50	95.80	94.23	86.73
2	97.62	97.65	97.05	97.25	95.80	87.50
3	97.87	97.92	97.44	97.30	96.41	86.30

TABLE 3.2 – Part de données synthétiques dans chaque sous-ensemble

Sous-ensemble	**Données synthétiques**
D1	16,7 %
D2	33,3 %
D3	50 %
D4	66,7 %
D5	83 %
D6	100 %

Analyse. L'intégration de données synthétiques exerce une influence limitée tant que leur part reste inférieure à 50 %. Au-delà, la précision décroît sensiblement : le scénario D6 (100 % synthétique) perd plus de 10 points par rapport à D1. Cette dégradation peut s'expliquer par la qualité encore insuffisante des images générées : chaque GAN n'a été entraîné qu'une époque et aucun diltrage par discriminateur n'a lieu lors de la creation des bases de données. Un apprentissage plus long et un filtrage automatique des sorties de faible qualité devraient améliorer la situation. Malgré cette réserve, l'augmentation de données par GAN demeure prometteuse : elle permet de limiter le coût de stockage tout en diversifiant les exemples vus par le réseau.

Bibliographie

Bibliographie

[1] D. E. Rumelhart, G. E. Hinton et R. J. Williams, « Learning representations by back-propagating errors », *Nature*, 323 (6088) : 533-536, 1986. `https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf`

[2] K. O'Shea et R. Nash, « An Introduction to Convolutional Neural Networks », prépublication *arXiv :1511.08458*, 2015. `https://arxiv.org/pdf/1511.08458`

[3] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville et Y. Bengio, « Generative Adversarial Nets », *Advances in Neural Information Processing Systems 27* (NeurIPS 2014), p. 2672-2680. `https://arxiv.org/pdf/1406.2661`