Séminaire sur les mathématiques de l'apprentissage machine

Semaine I : Introduction aux réseaux profonds de neurones.

G. Roy-Fortin

Service des enseignements généraux
École de technologie supérieure

18 septembre 2019

Une tâche

- Considérons une tâche simple pour un humain : classifier des chiffres manuscrits.

Figure – Chiffres tirés de la base de données MNIST.

- Très difficile à faire directement !
- L'*apprentissage machine* est un ensemble de techniques mathématiques et statistiques permettant à un algorithme d'apprendre par lui-même.

Le modèle humain

- L'idée générale derrière un *réseau de neurones* est ... un réseau de neurones !

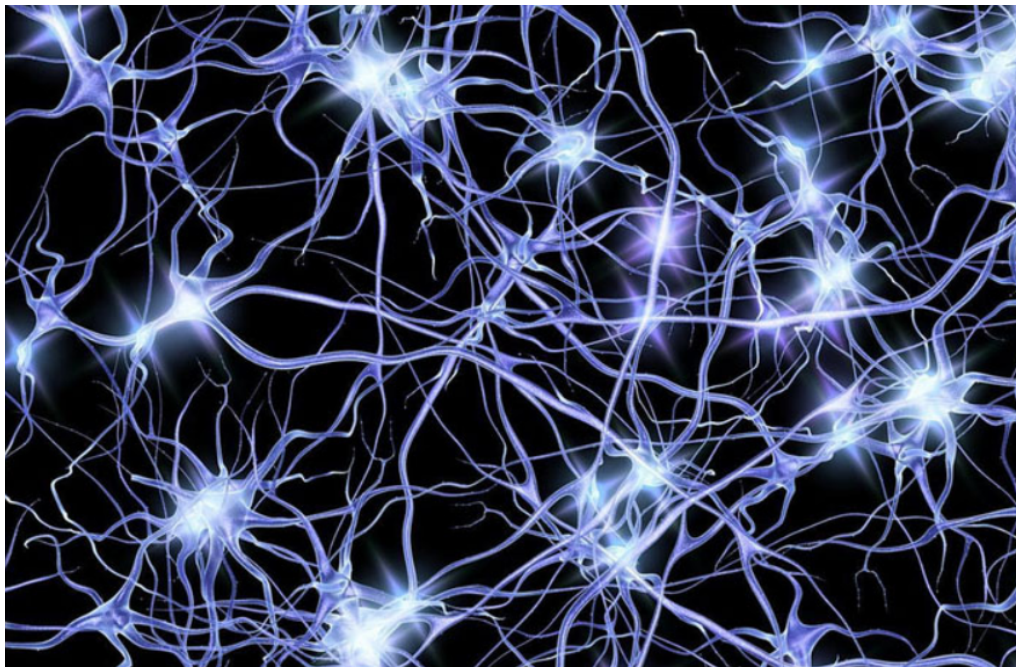

Figure – Neurones dans le cortex humain.

- On veut créer un système de neurones reliés entre eux et qui peuvent transmettre un signal plus ou moins actif.

Un exemple d'application : reconnaissance d'images

Une application centrale des réseaux de neurones : la reconnaissance d'image.

- ImageNet : 14 millions d'images classées sous 1000 étiquettes.
- Compétition : entraînement sur un sous-ensemble fixe d'images, test sur le reste.

Un réseau profond de neurones

- En général, un réseau profond de neurones (en anglais deep neural net) est un graphe de la forme :

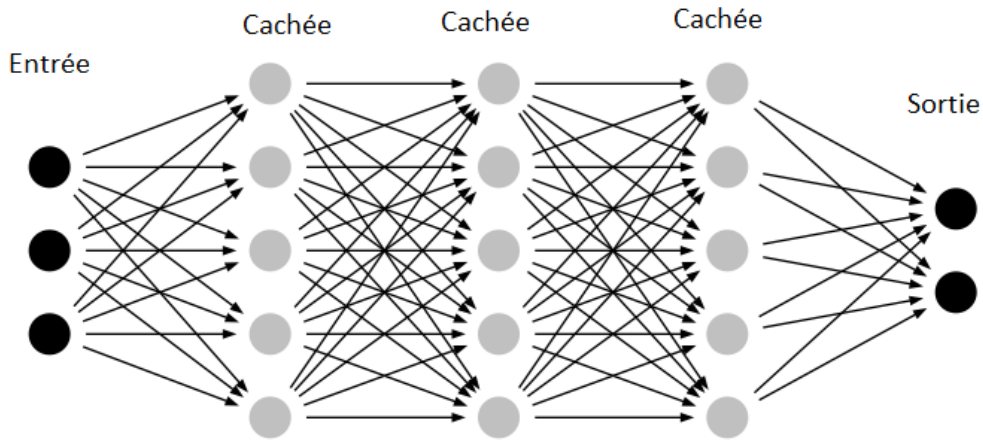

Figure – Un réseau de neurones avec 3 couches cachées.

- À chaque neurone on associe un *biais* b_j^ℓ
- À chaque connexion, on associe un *poids* w_{jk}^ℓ
- $\ell = 1, \ldots, L$ sont les *couches* du réseau
- a_j^ℓ : l'*activation*, ou puissance du signal, du j-ème neurone de la couche ℓ.

- On calcule l'activation d'un neurone via une *fonction d'activation*

$$a = \sigma(\mathbf{a}^T \cdot \mathbf{x} + b).$$

- Pour le moment, on utilisera seulement la fonction d'activation *sigmoïde* : $x \mapsto \dfrac{1}{1 + e^{-x}}$.

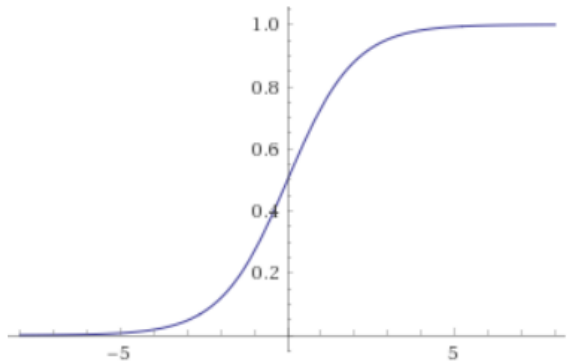

Figure – La fonction sigmoïde

- Le but des fonctions d'activation : introduire une *non-linéarité* dans le réseau.
- Le modèle complet est en en fait une fonction $f : \mathbb{R}^n \to \mathbb{R}^m$.

Déjà, on peut se poser des questions structurelles :

- Quelle architecture pour designer le réseau : nombres de neurones, de couches profondes etc.
- Quelle fonction d'activation : σ, ReLU, tangente hyperbolique etc.

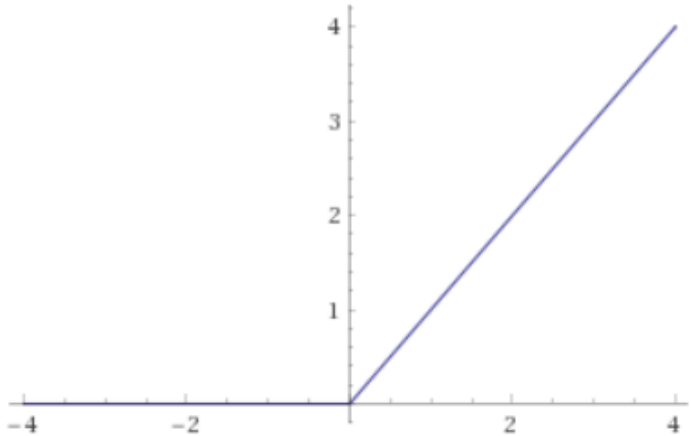

Figure – La fonction ReLU : rectified linear unit.

Il y a, dans un certain sens, beaucoup (trop ?) de liberté.

Un peu de terminologie

- On distingue les réseaux *feedfoward* : $\rightarrow$ et les réseaux dits *récurrents* : $\leftrightarrow$.
- On distingue aussi l'apprentissage *supervisé* où on possède des *données d'entraînement étiquetées* de l'apprentissage *non-supervisé*
- La notion d'apprentissage *profond* vient simplement de l'existence des couches cachées dans le modèle.
- Moralement, la "profondeur" permet d'introduire une *hiérarchisation* des concepts.

Quel réseau pour notre tâche ?

- Entrée : un neurone / pixel. Sortie : 10 neurones, un pour chaque chiffre.

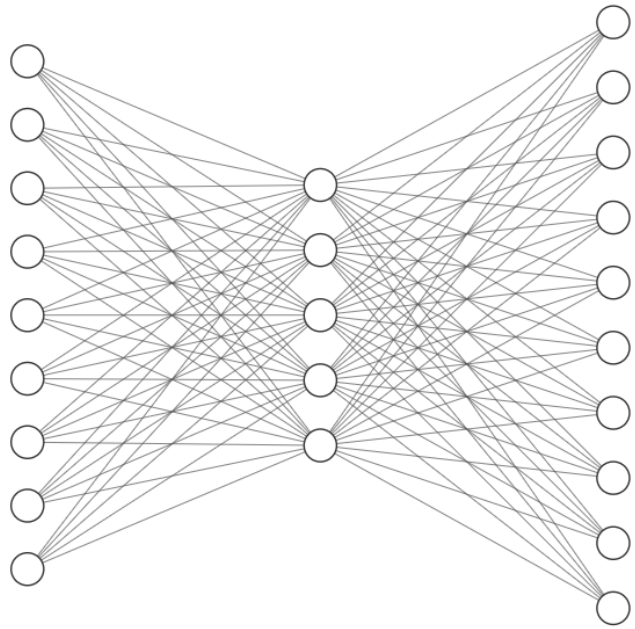

Figure – Notre premier réseau !

- Si l'image d'entrée est $64 \times 64 = 4096$ pixels et avec 5 neurones dans la couche cachée, on a :
  $4096 * 5 + 5 * 10 \approx 20000$ poids et $4096 + 5 + 10$ biais !

Apprendre via la descente de gradient

Comment mesurer cette performance ?

- Notre but est de modifier les paramètres w, b du réseau pour que celui-ci augmente sa performance à la tâche considérée.
- On définit une *fonction de coût*

$$C = C(w, b) = \frac{1}{2n} \sum_x ||\mathbf{y}(x) - \mathbf{a}^L||_{L^2}^2.$$

- x : donnée d'entraînement étiquetée, $\mathbf{y}(x)$: sortie désirée pour l'entrée x, $\mathbf{a}^L$: vecteur des activations à la sortie du réseau.
- On veut évidemment *minimiser* $C(w, b) \geq 0$!

On entraîne donc le réseau via les règles :

$$\begin{cases} w_k \mapsto w_k' = w_k - \eta \frac{\partial C}{\partial w_k}, \\ b_j \mapsto b_j' = b_j - \eta \frac{\partial C}{\partial b_j}. \end{cases}$$

Le nombre η est appelé *taux d'apprentissage*.

Puisque $C = \frac{1}{n} \sum_x C_x$, avec

$$C_x = \frac{1}{2} ||\mathbf{y}(x) - a||_{L^2}^2,$$

calculer ∇C exige de balayer toutes les données d'entraînement !

Pour pallier cet obstace, on introduit la *descente stochastique de gradient*.

SGD : Stochastic Gradient Descent

- On choisit aléatoirement un *mini-groupe* de m données d'entraînement étiquetées : $x_1, x_2, ..., x_m$.
- On suppose que la fonction de coût C est telle que

$$\nabla C \approx \frac{1}{m} \sum_{j=1}^{m} \nabla C_{x_j} = \frac{1}{2m} \sum_{j=1}^{m} ||\mathbf{y}(x) - \mathbf{a}||_{L^2}^2 .$$

- On obtient alors

$$\begin{cases} w_k \mapsto w_k' = w_k - \eta \frac{1}{m} \sum_{j=1}^{m} \frac{\partial C_{x_j}}{\partial w_k}, \\ b_j \mapsto b_j' = b_j - \eta \frac{1}{m} \sum_{j=1}^{m} \frac{\partial C_{x_j}}{\partial b_j}. \end{cases}$$

Quelques questions

- Quel est l'espace des fonctions qu'on peut espérer engendrer avec des réseaux profonds ?
- Quel rôle joue vraiment la profondeur d'un réseau ?
- Interprétation des hyperparamètres et de leur impact sur l'apprentissage : qualité, vitesse etc.
- Dans quel espace vivent les données d'entrée ?

Comment calculer ∇C_x ?

Pour exécuter la descente stochastique de gradient, on doit calculer ∇C_x, qui est un vecteur qui contient toutes les dérivées partielles du type

$$\frac{\partial C}{\partial w_{jk}^{\ell}}, \quad \frac{\partial C}{\partial b_j^{\ell}}.$$

La manière la plus directe de procéder est d'estimer

$$\frac{\partial C}{\partial w_{jk}^{\ell}} \approx \frac{C(w_{jk}^{\ell} + \epsilon) - C(w)}{\epsilon}.$$

Or, cette approche exige de calculer $C(w_{jk}^{\ell} + \epsilon)$ pour chaque paramètre ! Et donc, si on a 30 000 poids et biais dans le réseau, il faudra faire autant de traversée complète du réseau.

Solution : *propagation arrière* !

Les quatre équations de la backpropagation

(BP1)

$$\delta^L = \frac{\partial C}{\partial a_j^L}\,\sigma'(z_j^L)$$

(BP2)

$$\delta_j^\ell = \sum_k w_{kj}^{\ell+1}\,\delta_k^{\ell+1}\,\sigma'(z_j^\ell)$$

(BP3)

$$\frac{\partial C}{\partial b_j^\ell} = \delta_j^\ell$$

(BP4)

$$\frac{\partial C}{\partial w_{jk}^\ell} = a_k^{\ell-1}\,\delta_j^\ell$$