COURS DE TROISIÈME ANNÉE
INGÉNIEUR ISIMA

Autoencodeurs

Vincent Barra - Christophe Tilmant

ISIMA

Table des matières

1 AUTOENCODEUR PARCIMONIEUX

On considère un problème d'apprentissage à partir d'un ensemble d'apprentissage $\mathcal{E}_a = \left\{(x^{(i)}, y^{(i)}), 1 \leq i \leq m\right\}$. Les réseaux de neurones permettent de définir un ou des modèles non linéaires $h_{W,b}(x)$, de paramètres W, b à optimiser, qui répondent au problème posé (classification, régression).

1.1 Quelques rappels

1.1.1 Perceptron

Commençons par décrire le perceptron, le plus simple des réseaux de neurones, celui constitué d'un seul neurone. La figure 1 en présente un exemple. Le neurone prend en entrée x_1, x_2, x_3 et une entrée constante à 1 (pour modéliser le biais b, cf. cours sur les réseaux de neurones) et calcule :

$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} w_i x_i + b)$$

où $f : \mathbb{R} \mapsto \mathbb{R}$ est la fonction d'activation.
Dans la suite, cette dernière sera la fonction sigmoïde $f(z) = \frac{1}{1+e^{-z}}$, de dérivée $f'(z) = f(z)(1-f(z))$. Rappelons qu'un autre choix pour $f(.)$ est la fonction tangente hyperbolique $f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, de dérivée $f'(z) = 1 - (f(z))^2$

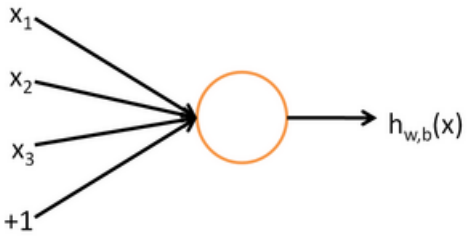

FIGURE 1 – Représentation d'un neurone

Remarque : ce simple réseau modélise la régression logistique.

1.1.2 Généralisation

Un réseau de neurones plus général peut alors être construit, comme la juxtaposition de plusieurs de ces neurones élémentaires, de telle sorte que la sortie d'un neurone serve d'entrée à un ou plusieurs autres neurones. C'est par exemple le modèle du perceptron multicouche rencontré dans un cours précédent.
La figure 2 présente un exemple de PMC, avec une rétine à trois entrées, deux couches cachées et une sortie vectorielle. Les neurones étiquetés +1 correspondent au biais b sur chaque couche.

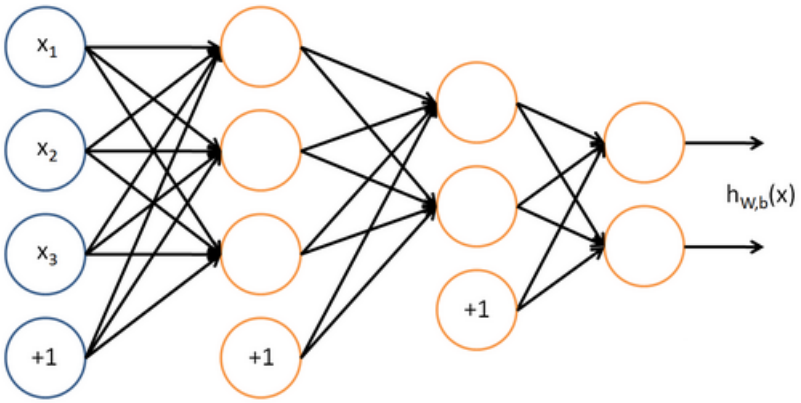

FIGURE 2 – Réseau de neurones

1.1.3 Notations

Soit L le nombre de couches du réseau. La couche $1 \leq i \leq L$ est notée L_l.
Les paramètres du réseau sont notés $(W,b) = (W^{(l)}, b^{(l)}, 1 \leq l \leq L)$, où $W_{ij}^{(l)}$ est le poids synaptique associé à la connexion entre le neurone j de la couche l et le neurone i de la couche $l+1$. $b_i^{(l)}$ est le biais associé au neurone i dans la couche l. Enfin, $m^{(l)}$ désigne le nombre de neurones de la couche l, hors neurone de biais.

La valeur de sortie du neurone i dans la couche l est notée $a_i^{(l)}$. Pour $l = 1$ (la rétine), on note aussi $a_i^{(1)} = x_i$ la i-ème entrée.
On note enfin $z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(1)} x_j + b_i^{(1)}$ le potentiel du neurone i dans la couche l, de sorte que $a_i^{(l)} = f(z_i^{(l)})$.
Pour un jeu de paramètres (W,b) donné, le réseau calcule $h_{W,b}(x) \in \mathbb{R}^d$.

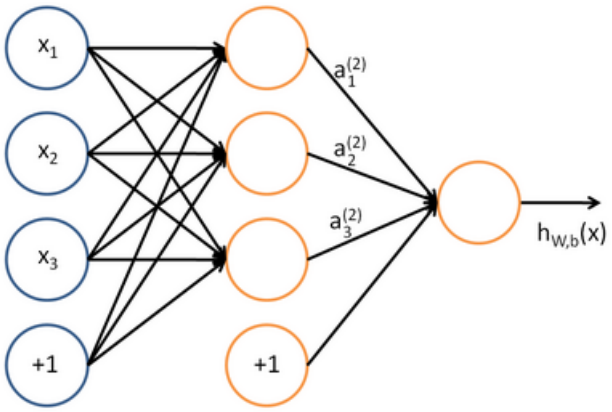

FIGURE 3 – Réseau de neurones

Dans l'exemple de la figure 3, on a plus précisément :

$$\begin{aligned}
a_1^{(2)} &= f\left(W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)}\right) \\
a_2^{(2)} &= f\left(W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)}\right) \\
a_3^{(2)} &= f\left(W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 + W_{33}^{(1)} x_3 + b_3^{(1)}\right) \\
h_{W,b}(x) &= a_1^{(3)} = f\left(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)} + b_1^{(2)}\right)
\end{aligned}$$

Ces notations permettent d'écrire de manière plus compacte l'ensemble des opérations réalisées par un réseau de neurones. En effet, si l'on permet à $f(\cdot)$ d'agir sur un vecteur point à point (i.e., $f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$), alors

$$\begin{aligned}
z^{(2)} &= W^{(1)} x + b^{(1)} && (1) \\
a^{(2)} &= f(z^{(2)}) && (2) \\
z^{(3)} &= W^{(2)} a^{(2)} + b^{(2)} && (3) \\
h_{W,b}(x) &= a^{(3)} = f(z^{(3)}) && (4)
\end{aligned}$$

Cette étape est dite propagation avant ("forward propagation.").
Plus généralement, puisque $a^{(1)} = x$ sur la rétine, alors pour tout $1 \leq l \leq L-1$:

$$z^{(l+1)} = W^{(l)} a^{(l)} + b^{(l)} \quad \text{et} \quad a^{(l+1)} = f(z^{(l+1)}) \tag{5}$$

1.1.4 Algorithme de rétropropagation du gradient

Soit $\mathcal{E}_a = \left\{ (x^{(i)}, y^{(i)}), 1 \leq i \leq m \right\}$ un ensemble d'apprentissage. L'apprentissage du réseau, qui consiste à trouver les paramètres (W, b), peut être réalisé par descente de gradient : pour un exemple (x, y), on définit une fonction de coût associée à cet exemple :

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2$$

La fonction de coût associée à $\mathcal{E}_a$ est alors

$$\begin{aligned} J(W,b) &= \left[\frac{1}{m} \sum_{i=1}^{m} J(W, b; x^{(i)}, y^{(i)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{L-1} \sum_{i=1}^{m^{(l)}} \sum_{j=1}^{m^{(l+1)}} \left(W_{ji}^{(l)} \right)^2 \\ &= \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \right] + \frac{\lambda}{2} \sum_{l=1}^{L-1} \sum_{i=1}^{m^{(l)}} \sum_{j=1}^{m^{(l+1)}} \left(W_{ji}^{(l)} \right)^2 \end{aligned}$$

Le premier terme de $J(W, b)$ est l'erreur quadratique moyenne sur $\mathcal{E}_a$, le second terme est un terme de régularisation qui privilégie les valeurs basses des poids, et aide à prévenir le phénomène de surapprentissage. Notons que ce second terme n'est pas appliqué aux termes de biais $b_i^{(l)}$. L'hyperparamètre λ contrôle l'importance relative de la régularisation.

$J(W, b)$ est utilisée à la fois en classification et en régression. Dans le premier cas (classification binaire), $y = 0$ ou $y = 1$ (la fonction d'activation sigmoïde renvoie une valeur dans $[0, 1]$) ou $y = -1$ ou $y = 1$ (la tangente hyperbolique vit sur $[-1, 1]$). Dans le second cas, il faut juste s'assurer que les sorties sont dans $[0, 1]$ (cas de la sigmoïde) ou $[-1, 1]$ (cas de la tangente hyperbolique).

L'objectif est alors de minimiser $J(W, b)$ par rapport à W et b. Pour cela, les paramètres $W_{ij}^{(l)}$ et $b_i^{(l)}$ sont tout d'abord initialisés à des valeurs proches de zéro (suivant par exemple une loi normale $\mathcal{N}(0, \varepsilon^2)$ pour un petit ε, ou un schéma type initialisation de Xavier). Il est important que les paramètres soient initialisés à des valeurs différentes, car sinon tous les neurones des couches cachées apprendront la même fonction des entrées ($W_{ij}^{(1)}$ identique pour tout i, de telle sorte que $a_1^{(2)} = a_2^{(2)} = a_3^{(2)} = \ldots$ pour tout x).
Un algorithme d'optimisation, comme la descente de gradient, est ensuite appliqué. Puisque $J(W, b)$ est non-convexe, l'algorithme est susceptible de converger vers un minimum local.
Une itération de la descente de gradient met à jour W et b de la manière suivante :

$$
\begin{aligned}
W_{ij}^{(l)} &= W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) \\
b_i^{(l)} &= b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W,b)
\end{aligned}
$$

où α est le taux d'apprentissage (learning rate). Les dérivées partielles sont calculées à l'aide de l'algorithme de rétropropagation du gradient. Etant donné un exemple (x,y), une propagation avant est effectuée, pour calculer toutes les activations du réseau, y compris $h_{W,b}(x)$ (voir par exemple les équations (1) à (4)). Pour chaque neurone i de la couche l, on calcule ensuite une erreur $\delta_i^{(l)}$ qui mesure la contribution de ce neurone à l'erreur totale du réseau. Pour un neurone de la couche de sortie, cette erreur $\delta_i^{(L)}$ est facile à calculer puisqu'il suffit de mesurer la différence entre la sortie calculée par le réseau et la sortie théorique y correspondante. Pour les neurones des couches cachées, $\delta_i^{(l)}$ est calculée comme une somme pondérée des erreurs des neurones qui utilisent $a_i^{(l)}$ comme entrée.
Connaissant les dérivées partielles $\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y)$ et $\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y)$, de la fonction de coût $J(W,b;x,y)$ sur présentation d'un exemple (x,y), l'algorithme 1 calcule les dérivées partielles de $J(W,b)$:

$$
\begin{aligned}
\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) &= \left[\frac{1}{m} \sum_{k=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(k)},y^{(k)}) \right] + \lambda W_{ij}^{(l)} \\
\frac{\partial}{\partial b_i^{(l)}} J(W,b) &= \frac{1}{m} \sum_{k=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J(W,b;x^{(k)},y^{(k)})
\end{aligned}
$$

Cet algorithme s'écrit également en notation vectorielle : en notant $\bullet$ le produit de Hadamard, i.e. tel que si $a = b \bullet c$, alors pour tout i $a_i = b_i c_i$, et en permettant à f' d'agir sur un vecteur point à point ($f'([z_1,z_2,z_3]) = [f'(z_1),f'(z_2),f'(z_3)]$), l'algorithme 2 propose la version vectorielle.
Une itération de l"algorithme de descente du gradient peut alors être décrite (algorithme 3). Ici, $\Delta W^{(l)}$ est une matrice de même taille que $W^{(l)}$, et $\Delta b^{(l)}$ est un vecteur de même taille que $b^{(l)}$. L'algorithme final consiste à répéter ces itérations jusqu'à convergence.

1.2 Autoencodeurs et représentation parcimonieuse

Les réseaux de neurones qui ont été décrits jusqu'à lors nécessitaient une base d'apprentissage. On s'intéresse ici au cas où seuls des exemples non étiquetés $\{x^{(1)},x^{(2)},x^{(3)},\ldots\}$ sont disponibles, avec $x^{(i)} \in \mathbb{R}^n$.
Dans sa version la plus simple, un réseau de neurones auto-encodeur est un algorithme d'apprentissage non supervisé qui apprend à l'aide de la rétropropagation, en imposant que les sorties désirées sont égales aux entrées, i.e. $\forall i \quad y^{(i)} = x^{(i)}$. Le réseau se compose d'une couche d'entrée, d'une couche cachée, et de la couche de sortie. Le sous réseau

Algorithme 1: Algorithme de rétropropagation

Calculer les activations des couches $L_2, L_3, \cdots L_L$

pour *chaque neurone i de la couche L* **faire**

$\delta_i^{(L)} = \frac{\partial}{\partial z_i^{(L)}} \frac{1}{2} \left\| y - h_{W,b}(x) \right\|^2 = -(y_i - a_i^{(L)}) \cdot f'(z_i^{(L)})$

pour $l = L-1, L-2, L-3, \ldots, 2$ **faire**

pour *chaque neurone i de la couche l* **faire**

$\delta_i^{(l)} = \left(\sum_{j=1}^{m^{(l+1)}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$

pour *tout i,j,l* **faire**

$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$

$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}$.

Algorithme 2: Algorithme de rétropropagation : version vectorielle

Calculer les activations des couches $L_2, L_3, \cdots L_L$

pour *la couche L* **faire**

$\delta^{(L)} = -(y - a^{(L)}) \bullet f'(z^{(L)})$

pour $l = L-1, L-2, L-3, \ldots, 2$ **faire**

$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)} \right) \bullet f'(z^{(l)})$

$\nabla_{W^{(l)}} J(W,b;x,y) = \delta^{(l+1)} (a^{(l)})^T$

$\nabla_{b^{(l)}} J(W,b;x,y) = \delta^{(l+1)}$

Algorithme 3: Itération de l'algorithme de descente de gradient

pour *tout l* **faire**

$\Delta W^{(l)} := 0$

$\Delta b^{(l)} := 0$

pour $i = 1$ *à* m **faire**

Utiliser la rétropropagation pour calculer $\nabla_{W^{(l)}} J(W,b;x,y)$ et $\nabla_{b^{(l)}} J(W,b;x,y)$

$\Delta W^{(l)} = \Delta W^{(l)} + \nabla_{W^{(l)}} J(W,b;x,y)$

$\Delta b^{(l)} = \Delta b^{(l)} + \nabla_{b^{(l)}} J(W,b;x,y)$

$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$

$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$

composé de la rétine et de la couche d'entrée est appelé encodeur, celui constitué de la couche cachée et de la couche de sortie le décodeur.
Le réseau tente donc d'apprendre $h_{W,b}(x) \approx x$, et essaye ainsi d'approcher l'identité, ce qui semble à première vue trivial. Cependant, en raison des contraintes imposées au réseau (nombre limité de neurones cachés par exemple), il est possible de découvrir via ce mécanisme des informations structurelles intéressantes sur les données d'éntrée (encodées via l'encodeur).

Supposons par exemple que les entrées x soient les niveaux de gris des pixels d'une image 10×10 . On a alors $n = 100$ entrées, et on pose $m^{(2)} = 50$ neurones cachés dans la couche L_2. Puisque la couche cachée ne comporte que la moitié de neurones de la rétine, le réseau doit apprendre une version compressée (encodée) de l'image d'entrée : en d'autres termes, à partir de la seule donnée des activations de la couche cachée $a^{(2)} \in \mathbb{R}^{50}$, le réseau doit reconstruire (décoder) l'image d'entrée x composée des 100 pixels. Comme une image porte toujours une structure spatiale liée à l'observation, certaines des entrées sont correlées, et le réseau capture ces relations.

Le nombre de neurones des couches cachées n'est pas la seule contrainte qui peut être imposée au réseau. Il est par exemple possible d'imposer aux neurones d'être "inactifs" la plupart du temps, en définissant l'inactivité comme une valeur de sortie du neurone proche de zéro (pour une sigmoïde, ou -1 pour une tangente hyperbolique). On parle alors de réseau parcimonieux.
Pour modéliser cette contrainte, et pour rester dans le cas précédent, on note $a_j^{(2)}(x)$ l'activation du neurone caché j lorsque l'entrée x est présentée au réseau. On note également

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} \left[a_j^{(2)}(x^{(i)}) \right]$$

l'activation moyenne du neurone caché j
L'objectif est alors d'imposer $\hat{\rho}_j = \rho$, où ρ est une valeur proche de zéro : ainsi l'activation moyenne de chaque neurone caché doit être faible. Pour ce faire, un terme de pénalité est ajouté à la fonction d'optimisation. De nombreux choix sont possibles, et par exemple pour un réseau à une couche cachée

$$\sum_{j=1}^{m^{(2)}} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}.$$

avec $m^{(2)}$ le nombre de neurones dans la couche cachée. Cette pénalisation est basée sur la divergence de Kullback-Leibler (KL), et peut être réécrite

$$\sum_{j=1}^{m^{(2)}} \mathrm{KL}(\rho || \hat{\rho}_j),$$

avec $\mathrm{KL}(\rho||\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$ la divergence de Kullback-Leibler (KL) entre une variable aléatoire de loi de Bernoulli de moyenne ρ et une variable aléatoire de loi de

Bernoulli de moyenne $\hat{\rho}_j$.
On peut alors montrer que $\mathrm{KL}(\rho||\hat{\rho}_j) = 0$ si $\hat{\rho}_j = \rho$, et KL croît de façon monotone lorsque $\hat{\rho}_j$ s'éloigne de ρ.

Le coût global est alors

$$J_{\text{sparse}}(W,b) = J(W,b) + \beta \sum_{j=1}^{m^{(2)}} \mathrm{KL}(\rho||\hat{\rho}_j), \tag{6}$$

où β contrôle le poids du dernier terme de pénalité. $\hat{\rho}_j$ dépend implicitement de W et b puisque l'activation des neurones cachés dépend de ces paramètres.
Le calcul des dérivées partielles et la descente de gradient changent peu. Pour un réseau à une couche cachée par exemple, $\delta_i^{(2)}$ on calcule maintenant

$$\delta_i^{(2)} = \left(\left(\sum_{j=1}^{m^{(3)}} W_{ji}^{(2)} \delta_j^{(3)} \right) + \beta \left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i} \right) \right) f'(z_i^{(2)}).$$

Il faut alors connaître $\hat{\rho}_i$, et donc faire dans un premier temps une propagation avant sur tous les exemples x permettant de calculer les activations moyennes.
Dans le cas où la base d'apprentissage est suffisamment petite, elle tient entièrement en mémoire et les activations peuvent être stockées pour calculer $\hat{\rho}_i$. Les activations stockées peuvent alors être utilisées dans l'étape de rétropropagation sur l'ensemble des exemples. Dans le cas contraire, le calcul de $\hat{\rho}_i$ peut être fait en accumulant les activations calculées exemple par exemple, mais sans sauvegarder les valeurs de ces activations. Une seconde propagation sur chaque exemple sera alors nécessaire pour permettre la rétropropagation.

1.3 Visualiser un autoencodeur entraîné

L'entraînement d'un réseau parcimonieux à une couche cachée étant effectué, avec une contrainte d'activation minimale, il peut être intéressant de visualiser la fonction apprise par l'auto-encodeur.
Considérons le cas où l'entraînement est effectué sur une image de taille 10×10. Chaque neurone caché i calcule :

$$a_i^{(2)} = f\left(\sum_{j=1}^{100} W_{ij}^{(1)} x_j + b_i^{(1)} \right).$$

La visualisation de la fonction calculée par chaque neurone caché i (sans le biais b, elle ne sera donc fonction que des paramètres $W_{ij}^{(1)}$) est effectuée à l'aide d'une image 2D. En considérant $a_i^{(2)}$ comme une fonction non linéaire de x, il est intéressant de se demander quelle image x active le neurone i au maximum. Pour répondre à cette question de manière non triviale, il est nécessaire d'imposer quelques contraintes sur x. Si l'on suppose que

l'entrée est telle que $||x||^2 \leq 1$, alors on peut montrer que l'entrée x qui maximise l'activation du neurone caché i est définie en calculant chaque pixel $x_j, 1 \leq j \leq 100$ comme

$$x_j = \frac{W_{ij}^{(1)}}{\sqrt{\sum_{j=1}^{100} (W_{ij}^{(1)})^2}}.$$

La figure 4 présente les 100 images correspondant aux 100 neurones cachés d'un réseau parcimonieux à une couche cachée, entrainé avec des images naturelles 10×10 prétraitées (blanchiment des images, qui enlève la redondance des entrées et décorrèle partiellement les pixels adjacents).

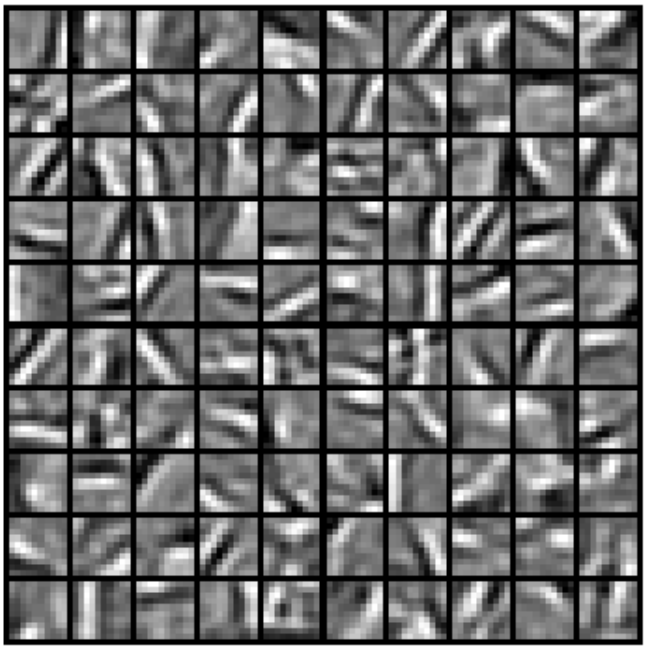

FIGURE 4 – Exemple d'autoencodeur

Chaque image dans la figure 4 est l'image qui active de manière maximale chacun des 100 neurones de la couche cachée. Chaque neurone a appris à détecter des bords dans différentes directions et orientations.
Les applications naturelles de ces autoencodeurs sont la reconnaissance de forme et d'objets, et la vision d'une manière plus générale. D'autres domaines (reconnaissance de la parole par exemple) sont également possibles.

1.4 Auto apprentissage

En supposant disposer d'un bon algorithme d'apprentissage, l'une des manières les plus sures d'améliorer les performances est de donner à cet algorithme plus de données, notamment étiquetées, pour l'apprentissage. Cependant, ceci peut être très coûteux (obtention des labels).
La technique d'apprentissage à partir de données non étiquetées, présentée précédemment avec les autoencodeurs, représente alors une voie d'amélioration intéressante. Même si un exemple non étiqueté porte moins d'information qu'un exemple qui l'est, la facilité d'obtention d'un grand nombre des premiers, alliée à un autoencodeur, doit permettre d'améliorer les performances d'apprentissage à un coût bien moindre, et de manière aussi efficace, que l'obtention d'un grand nombre de données étiquetées.
Dans les techniques dites d'auto apprentissage, on donne à un algorithme type autoencodeur un grand nombre de données non étiquetées. L'autoencodeur apprend alors une représentation de ces données, via ses neurones cachés. Dans un problème de classification, on utilise alors le réseau construit sur un (éventuellement petit) ensemble de données étiquetées relatives au problème.

1.4.1 Apprendre des caractéristiques

Soit un ensemble $\{x_u^{(1)}, x_u^{(2)}, \ldots, x_u^{(m_u)}\}$ de m_u exemples non étiquetés. Un autoencodeur parcimonieux est entraîné sur cet ensemble (figure 5).

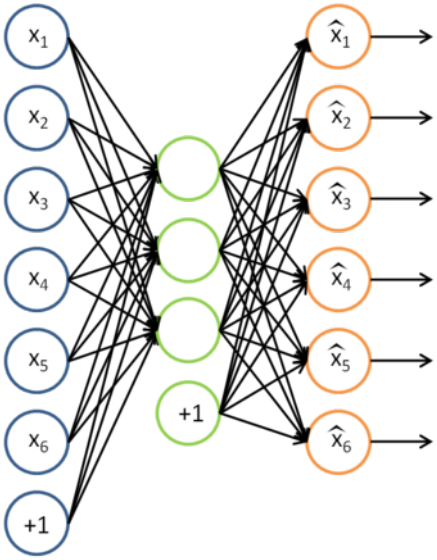

FIGURE 5 – Apprentissage de l'autoencodeur parcimonieux

Les paramètres $W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$ étant alors définis, pour une nouvelle entrée x, il est possible de calculer le vecteur des activations a des neurones cachés. Il est également possible de visualiser l'algorithme de calcul des activations a en utilisant le réseau de la figure 6, obtenu simplement à partir du réseau précédent en supprimant la couche de sortie.

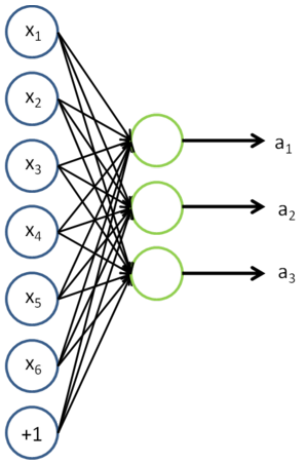

FIGURE 6 – Exemple d'autoencodeur

Supposons maintenant disposer d'un ensemble $\{(x_l^{(1)}, y^{(1)}), (x_l^{(2)}, y^{(2)}), \dots (x_l^{(m_l)}, y^{(m_l)})\}$ de m_l données étiquetées. Plutôt que de représenter le premier exemple comme $x_l^{(1)}$, il est possible de proposer $x_l^{(1)}$ à l'autoencodeur pour obtenir le vecteur d'activations $a_l^{(1)}$. L'exemple peut alors soit être représenté par $a_l^{(1)}$, soit par le couple $(x_l^{(1)}, a_l^{(1)})$. L'ensemble de données étiquetées devient alors $\{(a_l^{(1)}, y^{(1)}), (a_l^{(2)}, y^{(2)}), \dots (a_l^{(m_l)}, y^{(m_l)})\}$ dans le premier cas, et $\{((x_l^{(1)}, a_l^{(1)}), y^{(1)}), ((x_l^{(2)}, a_l^{(1)}), y^{(2)}), \dots, ((x_l^{(m_l)}, a_l^{(1)}), y^{(m_l)})\}$ dans le second. En pratique, la représentation par couples fonctionne plutôt mieux, mais est plus gourmande en représentation mémoire.
Enfin, un algorithme d'apprentissage supervisé (SVM ou autres) peut être entrainé sur ces données pour prédire les valeurs y : étant donné un exemple test x_{test}, celui-ci sert tout d'abord d'entrée à l'autoencodeur pour calculer a_{test}. Puis a_{test} ou le couple $(x_{\text{test}}, a_{\text{test}})$ est utilisé en entrée du classifieur (souvent softmax) pour obtenir une prédiction.
A noter que lors de la procédure d'apprentissage des caractéristiques, où les données non étiquetées $\{x_u^{(1)}, x_u^{(2)}, \dots, x_u^{(m_u)}\}$ ont utilisées, il se peut qu'un pré-traitement (soustraction de la moyenne, utilisation d'une ACP pour obtenir une représentation des données sous la forme d'une matrice $U^T x$, blanchiment des données...) ait été effectué. Dans ce cas, ces mêmes paramètres de pré-traitement doivent être utilisés lors des phases suivantes (obtention des activations et test).

1.4.2 Origine des données

Deux scénarios peuvent être rencontrés. Dans le premier cas, le plus général, aucune hypothèse n'est faite sur la distribution des données non étiquetées. En particulier, rien n'impose à ce qu'elles soient tirées selon la même distribution que les données étiquetées x_l. Dans le second cas, souvent appelé apprentissage semi-supervisé, les données non étiquetées proviennent de la même distribution que les données étiquetées.
Prenons un exemple simple : l'objectif de l'apprentissage est de classifier des images de piétons et d'arbres. En supposant qu'une base de données conséquente d'images non étiquetées soit disponible, l'entraînement d'un autoencodeur permet d'apprendre des ca-

ractéristiques sur le problème posé. Puisque certaines des données non étiquetées peuvent être autre chose que des piétons ou des arbres (dans le cas d'une grande base d'images téléchargée par exemple), les données non étiquetées sont tirées selon une distribution différente des données étiquetées.
A l'inverse, et le cas est plus rare, si vous savons disposer d'une base de données d'images, qui sont exclusivement soit des piétons soit des arbres, mais que les labels sont inconnus, alors ces données peuvent être utilisées pour entraîner l'autoencodeur. Les données non étiquetées et étiquetées sont issues de la même distribution.

1.5 Cas de nombreuses données étiquetées

Etant donné un ensemble d'apprentissage $\{(x_l^{(i)}, y^{(i)}), 1 \leq i \leq m_l\}$, nous avons montré comment remplacer les entrées originales $x^{(i)}$ par les activations $a^{(i)}$ calculées par l'autoencodeur parcimonieux, et ainsi remplacer l'ensemble d'apprentissage par un ensemble $\{(a^{(i)}, y^{(i)}), 1 \leq i \leq m_l\}$, qui vient nourrir un classifieur permettant d'estimer les labels $y^{(i)}$. Dans le cas par exemple où le classifieur est un neurone qui effectue une régression logistique, l'étiquetage se fait comme présenté en figure 7.

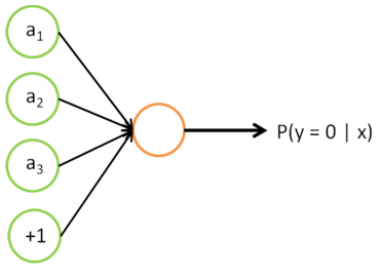

FIGURE 7 – Classifieur logistique

Considérons maintenant le classifieur global, appris en utilisant cette méthode (figure 8). La structure est celle d'un réseau de neurones.

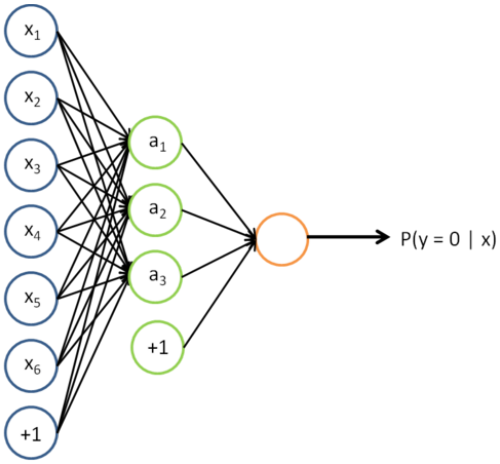

FIGURE 8 – Classifieur global

Les paramètres de ce modèle sont appris en deux étapes. La première couche des poids $W^{(1)}$, reliant les entrées x aux activations des neurones cachés a est calculée en utilisant l'autoencodeur. La seconde couche $W^{(2)}$ reliant les activations a aux sorties y est calculée à l'aide du classifieur.
Suivant ces deux étapes (parfois appelées pré-entraînement), il est alors possible de tenter de réduire encore l'erreur d'apprentissage, par exemple en effectuant une descente de gradient, initialisée sur les paramètres appris, sur les données d'entraînement étiquetées $\{(x_l^{(i)}, y^{(i)}), 1 \leq i \leq m_l\}$.
Notons qu'il est également possible, mais peu avantageux, d'utiliser cette approche, en supposant que le classifier voit non pas les exemples du type $(a^{(i)}, y^{(i)})$, mais plutôt $((x^{(i)}, a^{(i)}), y^{(i)})$. Le modèle est alors un réseau de neurones dans lequel les entrées sont reliées à la couche de sortie.
Dans les deux cas, ces techniques sont utilisées seulement si un grand ensemble de données d'apprentissage étiquetées est disponible.

2 EMPILEMENT D'AUTOENCODEURS

2.1 Introduction

Dans la partie précédente, nous avons construit un réseaux de neurones à trois couches. Bien qu'il soit déjà performant, ce réseau reste d'expression assez faible, les activations cachées $a^{(2)}$ n'étant calculées que sur une seule couche. Comme dans le cas des réseaux convolutifs, il est possible de jouer sur la multiplicité des couches cachées. La présence de nombreuses couches cachées va permettre de calculer des caractéristiques beaucoup plus complexes et informatives des entrées. Chaque couche calculant une transformation non linéaire de la couche précédente, le pouvoir de représentation de ces réseaux s'en trouve amélioré.

2.2 Présentation

Un empilement d'autoencodeurs est un réseau de neurones constitué de n couches d'autoencodeurs parcimonieux. Les sorties de chaque couche sont connectées aux entrées de la couche suivante.
Soient $W^{(k,1)}, W^{(k,2)}, b^{(k,1)}, b^{(k,2)}$ les poids et biais du k^e encodeur. L'étape d'encodage de l'empilement d'autoencodeurs est effectuée en réalisant l'étape d'encodage de chaque couche depuis l'entrée jusqu'à la sortie (forward) :

$$\begin{aligned} a^{(l)} &= f(z^{(l)}) \\ z^{(l+1)} &= W^{(l,1)} a^{(l)} + b^{(l,1)} \end{aligned}$$

L'étape de décodage est réalisée dans le sens inverse (backward) :

$$\begin{aligned} a^{(n+l)} &= f(z^{(n+l)}) \\ z^{(n+l+1)} &= W^{(n-l,2)} a^{(n+l)} + b^{(n-l,2)} \end{aligned}$$

L'information est stockée dans les activations $a^{(n)}$ de la couche cachée la plus profonde. Ce vecteur donne donc une représentation des entrées en terme de caractéristiques d'ordre élevé. D'une manière générale, les caractéristiques de l'empilement peuvent être utilisées pour des problèmes de classification, en fournissant $a^{(n)}$ à un classifieur.

2.3 Entraînement

L'entraînement est réalisé par la méthode gloutonne couche à couche. La première couche est entraînée sur les entrées, pour obtenir les paramètres $W^{(1,1)}, W^{(1,2)}, b^{(1,1)}, b^{(1,2)}$. Les entrées sont alors transformées en un vecteur des activations des neurones cachés. La seconde couche est entraînée sur ce vecteur pour obtenir $W^{(2,1)}, W^{(2,2)}, b^{(2,1)}, b^{(2,2)}$. Le processus est répété sur chaque couche k, en utilisant la sortie de la couche $k-1$ comme entrée de la couche k.
Cette méthode entraîne les paramètres de chaque couche individuellement, laissant les autres inchangés.

2.4 Exemple

Supposons un empilement d'autoencodeurs à deux couches cachées, dans un problème de classification .
La première étape consiste à entraîner un autoencodeur parcimonieux sur les données d'entrée x_k (figure 9), ce qui donne accès à des valeurs de sortie $h_k^{(1)}$ des neurones de la première couche cachée.

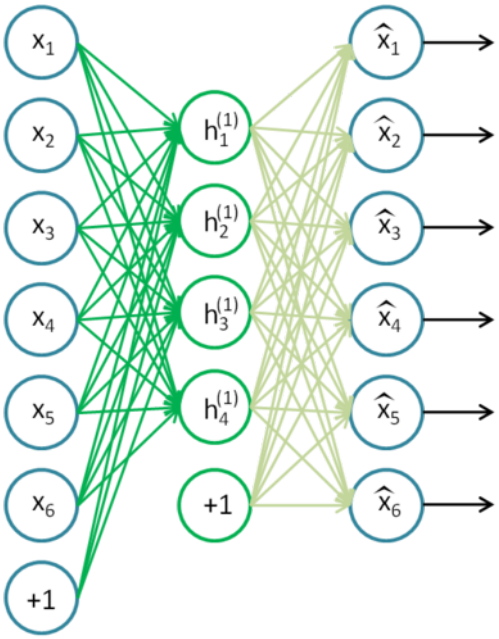

FIGURE 9 – Apprentissage du premier autoencodeur parcimonieux

Ces activations servent d'entrée à un second autoencodeur parcimonieux pour apprendre les $h_k^{(2)}$ (figure 10).

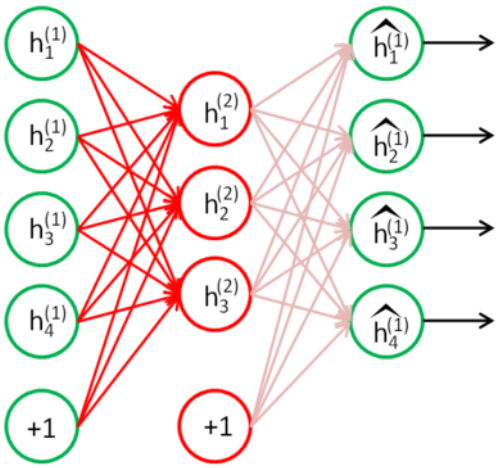

FIGURE 10 – Apprentissage du second autoencodeur parcimonieux

Les $h_k^{(2)}$ sont enfin passées en entrée d'un classifieur type softmax (figure 11).

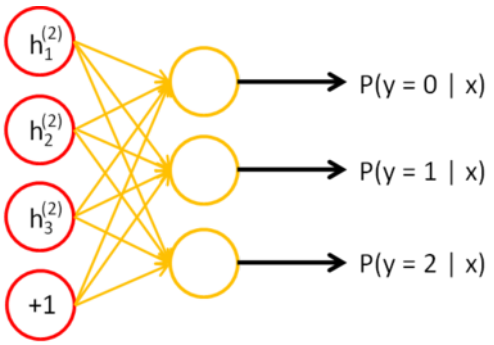

FIGURE 11 – Couche classifieur

Les trois couches sont finalement combinées pour former un empilement d'autoencodeurs à deux couches cachées et un classifieur final (figure 12).

2.5 Discussion

Un empilement d'autoencodeurs permet généralement de capturer des informations hiérarchiques à différentes échelles suivant les couches. Ainsi par exemple, la première couche tend à apprendre des caractéristiques du premier ordre dans les données d'entrée (comme les contours dans une image). La seconde couche apprend des caractéristiques de second ordre, correspondant à des motifs présents dans les caractéristiques de premier ordre (par exemple quels sont les couples de contours orientés qui apparaissent souvent, pour former des détecteurs de coins). Les couches suivantes capturent des caractéristiques d'ordre plus élevé.

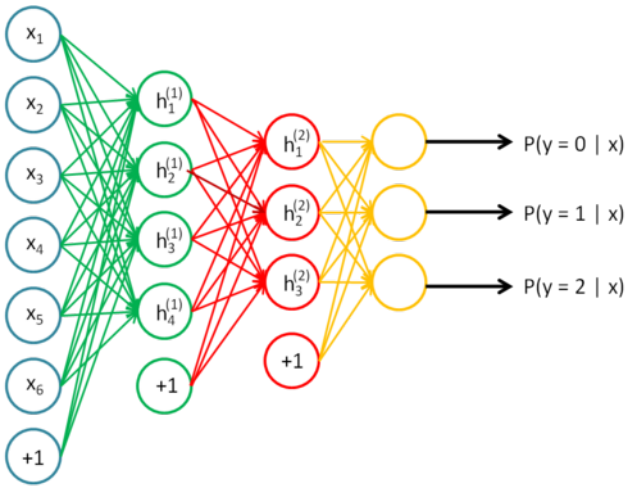

FIGURE 12 – Empilement d'autoencodeurs final

2.6 Paramétrage fin d'un empilement d'autoencodeurs

Le paramétrage fin (Fine-tuning) est une stratégie souvent rencontrée en apprentissage profond, qui peut améliorer de manière significative les performances des réseaux construits, et en particulier ici d'un empilement d'autoencodeurs. D'un point de vue global, le fine-tuning considère toutes les couches d'un empilement d'autoencodeurs comme un seul et même modèle de sorte qu'en une itération, tous les poids sont améliorés.

L'algorithme de rétropropagation, qui s'applique à un nombre arbitraire de couches, est utilisé pour calculer les gradients sur toutes les couches de l'empilement.
Réécrivons l'algorithme de propagation, point à point (Algorithme 4). Dans cet algorithme, on considère que la dernière couche du réseau représente les entrées du classifieur (type softmax). En utilisant une régression de type softmax, la couche finale modélisant la régression se calcule par $\nabla J = \theta^T (I - P)$ où I sont les labels d'entrée et P le vecteur des probabilités conditionnelles.

Algorithme 4: Algorithme de rétropropagation

Faire une passe avant, pour calculer les activations sur les couches $L_2 \cdots L_L$
pour *la couche* L **faire**
  $\delta^{(L)} = -(\nabla_{a^L} J) \bullet f'(z^{(L)})$
pour $l = L-1, L-2, L-3, \ldots, 2$ **faire**
  $\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$
Calculer les dérivées partielles
  $\nabla_{W^{(l)}} J(W,b;x,y) = \delta^{(l+1)} (a^{(l)})^T$,
  $\nabla_{b^{(l)}} J(W,b;x,y) = \delta^{(l+1)}$
avec $J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)}) \right]$

3 PARTIE PRATIQUE

Le fichier `squeletteAE.py` vous étant donné, il est demandé de construire deux auto-encodeurs :

1. un premier avec une couche cachée.
2. un second dont l'encodeur et le décodeur ont chacun deux couches.

Vous devez spécifiez pour chacun de ces autoencodeurs :

— l'initialisation des poids et biais.

— le ou les couches de l'encodeur et du décodeur (voir ici).

Vous ferez varier les paramètres de vos autoencodeurs (nombre de neurones sur les couches cachées, learning rate, nombre d'itérations, taille des batchs...), et observerez :

— dans le cas de l'autoencodeur à une couche cachée, les activations de la couche cachée sous la forme d'une image

— les images reconstruites pour les deux autoencodeurs produits, et vous jugerez de la qualité de la reconstruction.