Projet MDI343
Apprentissage profond, réseau de neurones, *etc.*

Antoine BIARD & Vincent BODIN

Résumé

Ce projet s'attelle à la question de la représentation des données en *machine learning*. Ces représentations, utilisées ensuite par l'algorithme de décision - régression logistique, SVM... - jouent un rôle primordial car porteuses de l'information. A la méthode classique de création de *features* par des connaissances *a priori*, on oppose des méthodes d'extraction de représentation par des structures dites profondes. Notre projet a consisté essentiellement en deux parties : une première partie de compréhension et d'étude des principales méthodes, et une seconde d'implémentation et de test sur quelques unes de ces méthodes (RBM, MLP, DBN).

Table des matières

1 Introduction : de l'importance de la représentation des données

Ces dernières années se sont imposées massivement des méthodes dites d'apprentissage profond - *deep learning* en anglais. La communauté scientifique a accru son intérêt en la matière au vue de leurs résultats, dans des domaines aussi variés que la vision par ordinateur ou la reconnaissance vocale. Atteignant souvent des performances au moins équivalentes à l'état de l'art, de nombreux algorithmes de type profond ont été créés pour tenter d'améliorer encore la représentation.

La représentation est une clef fondamentale dans l'élaboration d'un processus d'intelligence artificielle, en ceci qu'elle concentre l'information des données. On distingue en effet une première étape d'apprentissage de la représentation, puis une seconde d'apprentissage, à partir de cette représentation, d'une décision - pour de la classification par exemple. Les algorithmes effectuant cette dernière tâche sont hautement dépendants des données fournies en entrée. Une amélioration de la représentation entraîne de fait une meilleure performance dans la prise de décision. Nous renvoyons à la deuxième section de [1] pour des quantifications d'amélioration de la performance avec des représentations plus élaborées.

Traditionnellement, les descripteurs - ou encore *features* en anglais - sont extraits de façon dite supervisée à partir des données. Ceci nécessite une connaissance *a priori* dans le domaine, puisque cela requiert souvent une importante étape de traitement des données pour en extraire l'information - citons parmi tant d'autre en vision les points de Harris et les SIFT ou encore le spectrogramme et les *Mel-frequency cepstrum coefficients* en audio. Cela repose sur l'idée que l'homme a la capacité d'extraire et d'organiser l'information intrinsèque des données. Pour autant, dans un cadre d'intelligence artificielle pure, il serait désirable de s'affranchir de cette hypothèse et d'implémenter des algorithmes capables d'apprendre une représentation convenable sans supervision - non-supervisé. En outre, les types de données ayant tendance à devenir de plus en plus variés - image, son, web... - une telle connaissance dans chaque domaine s'avère illusoire et le manque de généralité de ces méthodes se paye dans la nécessité d'expertise dans chaque domaine. Il s'agit d'implémenter des méthodes extrayant des données une représentation abstraite.

Une manière d'atteindre ce but est l'extraction d'une représentation par apprentissage profond - *deep learning*. Une telle représentation s'obtient par l'ajout de couches successives. Une première représentation est obtenue par un processus à partir des données pures, puis cette représentation est elle-même raffinée, et ainsi de suite. Dans l'espoir d'obtenir une représentation abstraite, la non-linéarité devient nécessaire puisque sans elle, l'ajout de deux couches linéaires serait équivalent à une unique couche - le concept de profondeur perdrait sens.

Nous allons par la suite détailler les algorithmes d'apprentissage profond les plus classique, puis nous comparerons les performances de certains d'entre eux.

2 Revue élémentaire des méthodes de *deep learning*

2.1 Réseau de neurones

Un réseau de neurone peut être vu comme l'utilisation en série et parallèle plusieurs neurones simples, cf. Fig.(1). Un neurone est un noeud, avec un certain nombre d'entrée, et une fonction d'activation, disons σ - souvent tangente hyperbolique ou une sigmoïde, l'idée étant d'avoir une fonction proche de la fonction seuil, qui donne une activation *soft*. Pour un neurone ayant p entrée, chacun pondéré par un poids w_i, et un *offset* b, la sortie du neurone est :

$$h_{w,b}(x) = \sigma(w^T x) = \sigma\left(\sum_{i=1}^{p} w_i x_i + b\right) \tag{1}$$

Dans le cas d'un réseau de neurones, en utilisant une formule de chaîne où chaque sortie de neurone intermédiaire devient l'entrée du neurone suivant, on obtient un réseau du type de la Fig.(1). On voit ici apparaître naturellement l'idée de profondeur dans l'apprentissage par accumulation de couche de neurone, bien qu'il s'agisse du modèle le plus basic d'apprentissage profond. La fonction de sortie dépend de tous les paramètres w, b à chaque couche et chaque noeuds. La difficulté majeure dans l'apprentissage profond réside en l'estimation des paramètres optimaux - dans le sens où la sortie est la représentation la plus adéquate.

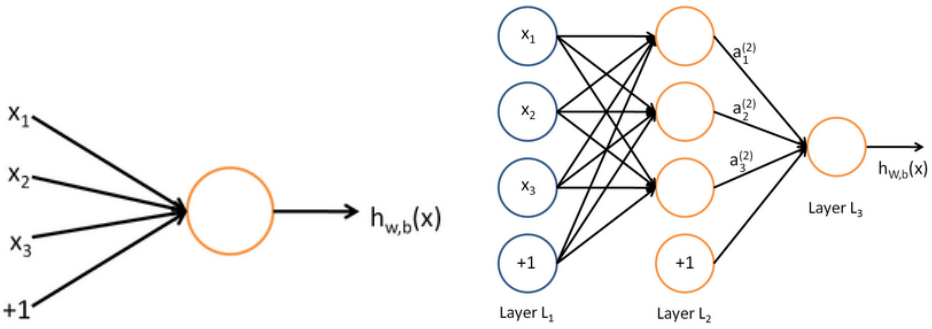

FIGURE 1 – (gauche) Un neurone avec trois entrées (x_1, x_2, x_3) et un *offset* ; (droite) un réseau de neurones de taille $(3, 3, 1)$ avec des *offset* aux deux premières couches.

2.2 Multilayer Perceptron (MLP)

Le second modèle classique en apprentissage profond est le perceptron multicouche (MLP). Il s'agit du premier exemple d'algorithme utilisant la méthode dite de *backpropagation*. Bien que peu utilisé seul en pratique, car la fonction d'optimisation n'est pas convexe et il y a de grand risque de trouvé un optimum local et non global, le MLP correspond aux prémisses du *deep learning* et sera réutilisé pour des modèles plus complexes par la suite.

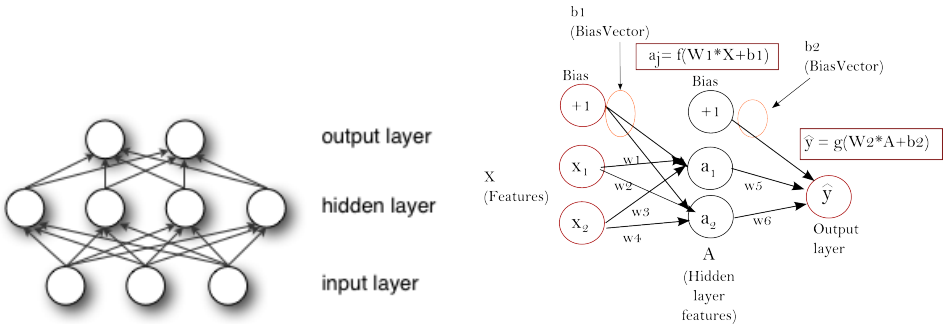

FIGURE 2 – (gauche) Un MLP avec une seule couche de variables cachées ; (droite) structure de MLP avec l'ajout de biais et les notions de poids.

Le MLP possède une fonction d'activation σ - tangente hyperbolique ou la fonction logistique souvent - des noeuds de biais b et des poids W associés à chaque arrête. Pour chaque couche, le paramètre est donc $\theta = (W, b)$, et pour une entrée x de dimension d - *i.e.* d noeuds - la sortie de cette couche est :

$$s(x) = \sigma(Wx + b) \tag{2}$$

Ceci forme un vecteur de taille disons m - sur la Fif.(2) par exemple pour la première couche $d = 3$ et $m = 4$ - qui est envoyé comme entrée à la couche suivante.

La nouveauté du MLP comparé au réseau de neurone réside en la manière d'apprendre l'hyper-paramètre $\theta = (W_1, \cdots, W_p, b_1, \cdots, b_p)$ - p est le nombre de couches. L'algorithme en question sera développé plus tard mais nous en donnons un aperçu. L'initialisation s'effectue classiquement de manière aléatoire - bien que dans le cas du DBN on lui envoie des poids définis en entrée. Dans [2], il est expliqué comment tirer les poids initialement. L'algorithme contient ensuite deux étapes :

Forward pass. On part des variables visibles v et on remonte le graphe avec les poids de la structure. On en déduit une représentation.

Backpropagation. A partir de cette représentation, on effectue le chemin inverse en descendant le graphe et en comptabilisant les erreurs commises. En comparant les résultats de la descente avec les variables visibles, on peut mettre à jour les poids.

Cela est répété un certain nombre de fois et cette mise à jour par *backpropagation* permet d'obtenir une représentation convenable. Toutefois ceci est très dépendant de l'initialisation.

Notons que ceci ne fournit qu'une représentation. Pour effectuer une classification, on utilise souvent une base de données étiquetée. Chaque représentation est associée à une étiquette et on se sert des représentations comme *feature* de l'algorithme de décision - régression logistique, SVM...

2.3 Restricted Boltzmann Machine (RBM)

2.3.1 Machine de Boltzmann

Une machine de Boltzmann est un réseau de variables aléatoires couplées. On se restreint au cas où les données sont binaire, on note les variables visibles v et les cachées h. Une machine de Boltzmann est représentée en Fig.(3) à gauche.

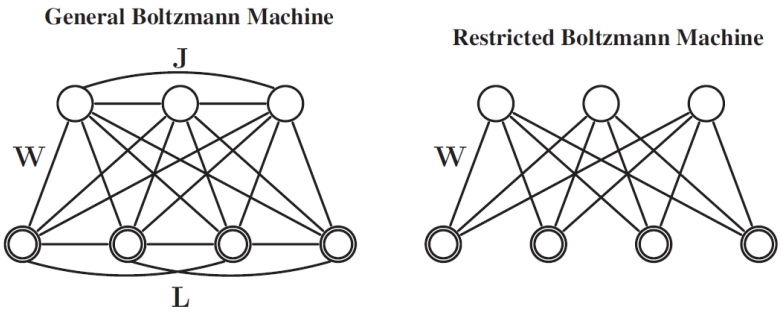

FIGURE 3 – (gauche) Une machine de Boltzmann générale ; (droite) machine de Boltzmann restreinte (RBM).

L'énergie des états (v, h) prend alors la forme :

$$E(v, h; \theta) = -v^T W h - \frac{1}{2} v^T L v - \frac{1}{2} h^T J h \tag{3}$$

(où l'on ne prend pas en compte ici les termes de biais qui correspondent à un recentrage des (v, h)). Il s'agit d'un cas de *energy-based models* (EBM). Dans ce cas, la probabilité d'un état v s'écrit :

$$p(v; \theta) = \frac{1}{\mathcal{Z}} \sum_h e^{-E(v,h;\theta)} \tag{4}$$

où $\mathcal{Z} = \sum_{x,h} e^{-E(v,h;\theta)}$ est la fonction de partition. Ces modèles ont la propriété que le gradient de $\log p(v)$ - log-vraisemblance - se décompose en deux termes, une phase positive et une négative. Pour autant la vraisemblance n'est pas calculable car exponentielle à la fois en le nombre de variables cachées et visibles.

2.3.2 Restricted Boltzmann Machine

Comme montré dans la Fig.(3), un RBM est une machine de Boltzmann où les interactions inter-couches sont gelées. De par la forme de son graphe, le RBM a des propriétés d'indépendances conditionnelles qui rendent l'inférence facile en pratique :

$$\begin{aligned} p(h|v) &= \prod_i p(h_i|x) \\ p(v|h) &= \prod_j p(x_j|h) \end{aligned} \tag{5}$$

Par suite on a :

$$\begin{aligned} p(h_i = 1|v) &= \sigma\left(\sum_j W_{ji}x_j + d_i\right) \\ p(x_j = 1|h) &= \sigma\left(\sum_i W_{ji}h_i + b_j\right) \end{aligned} \tag{6}$$

où σ est la fonction d'activation, et les (b, d) sont les biais. Pour autant, comme la plupart du temps, la fonction de partition reste incalculable en pratique, mais ces formules de factorisation permettent d'effectuer de l'inférence sans avoir à la calculer.

Nous en reparlerons plus en détail dans la partie d'implémentation, mais l'entraînement d'un RBM nécessite d'effectuer des étapes d'échantillonnage - de Gibbs - souvent par des méthodes MCMC. Il existe plusieurs méthodes pour accélérer le processus, nous parlerons de la *Contrastive Divergence* (CD) et du *Stochastic Maximum Likelihood* (SML).

2.4 Deep Belief Networks (DBN)

Un DBN est une structure d'apprentissage de représentation qui consiste en un empilement de RBM. Le premier RBM prend en entrée les variables visibles - données du problème - et, par échantillonnage en extrait un ensemble de variables cachées. Celles-ci sont alors utilisées comme entrée de la seconde couche et ainsi de suite, cf. Fig.(4). L'entraînement est donc de type *greedy*. Finalement, la probabilité jointe entre x et h s'écrit :

$$p(x, h^1, \cdots, h^l) = \left(\prod_{k=0}^{l-2} p(h^k|h^{k+1})\right) p(h^{l-1}, h^l) \tag{7}$$

avec $x = h^0$. Nous verrons l'implémentation de l'algorithme en détail plus tard. En modèle graphique, cette décomposition s'interprète comme un graphe dont la dernière couche est non-orientée tandis que toutes les précédentes sont des graphes orientés.

L'entraînement du DBN repose en outre sur un *fine-tuning* des paramètres obtenus par apprentissage par couches sur les RBM. Le *fine-tuning* s'effectue soit selon un *proxy* de la log-likelihood - cas non-supervisé - ou d'un critère d'erreur dans le cas supervisé. Nous nous plaçons dans le cas supervisé, et la méthode de classification appliquée est le MLP.

Nous pouvons donner ici une justification rapide du *pre-training* par couche. Dans le cas de deux couches par exemple, il a été montré [3] que :

$$\log p(x) = \mathrm{KL}(Q(h^{(1)}|x)\|p(h^{(1)}|x)) + \mathrm{H}_{Q(h^{(1)|x)}} + \sum_h Q(h^{(1)}|x)\left(\log p(h^{(1)}) + \log p(x|h^{(1)})\right) \tag{8}$$

où KL est la divergence de Kulbac-Leibler, H l'entropie, $Q(h^{(1)}|x)$ la distribution *a posteriori* du premier RBM (appris seul) et $p(h^{(1)}|x)$ celle après l'apprentissage entier du DBN. Si l'initialisation est $W^{(2)} = W^{(1)T}$ alors $Q(h^{(1)}|x) = p(h^{(1)}|x)$ et la divergence est nulle. Fixer ensuite $W^{(1)}$ et optimiser selon $W^{(2)}$ ne peut qu'augmenter la log-vraisemblance.

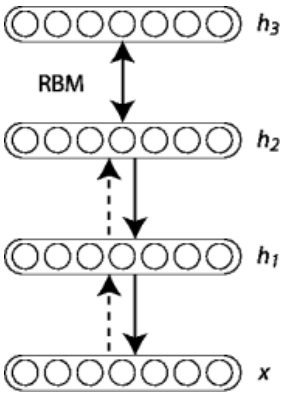

FIGURE 4 – Un *deep belief network* (DBN), la dernière couche est non orientée, tandis que toutes les autres le sont.

2.5 Deep Boltzmann Machine (DBM)

Un DBM est une structure profonde qui, en première approche tout au moins, ressemble fortement à un DBN. De la même manière que le DBN, le DBM extrait une représentation abstraite des données. Également, un faible nombre de données étiquetées est nécessaire à la tâche de classification, ce qui peut être intéressant dans certains cas. Il est dit dans [4] que l'inférence dans les DBM est plus robuste que dans les DBN car les entrées ambiguës ont moins d'impact.

Un DBM est constitué de RBM empilés comme dans le cas des DBN. L'étape de *pre-training* se résume à un entraînement couche par couche - *greedy* - de ces RBM. L'idée pour autant est de modifier un peu cette étape pour éviter le double passage quand on fait les étapes de haut vers le bas et bas vers le haut. Ainsi, pour le niveau le plus bas - plus bas RBM - les entrées sont doublées avec les poids visibles-cachés inchangés, cf. Fig.(5). Une étape d'évaluation du DBM basé sur des méthode de Monte Carlo - *Annealed Importance Sampling* - a ensuite lieu. Finalement, on termine comme dans le cas du DBN par du *fine-tuning* avec un MLP par exemple - qui converti la représentation en étiquette.

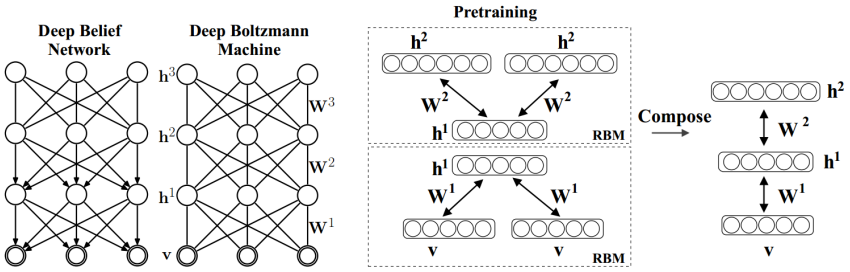

FIGURE 5 – (extrême gauche) Un DBN avec sa dernière couche non orientée ; (gauche) graphe de DBM, toutes les couches sont non orientées ; (droite) *pre-training* des RBM couche par couche ; (extrême droite) composition des couches pour former le DBM.

2.6 Auto-Encoder

La dernière méthode que nous présentons succinctement est celle des *auto-encoder*. Partant d'un ensemble d'apprentissage $x^{(t)}$ on extrait des descripteurs $h^{(t)}$ à l'aide d'une fonction paramétrique $f_\theta(x)$. Une autre fonction g_θ est appelé le décodeur et permet de faire la transformation inverse, $r = g_\theta(h)$ s'appelle la reconstruction. Une erreur de reconstruction est définie $L(x^{(t)}, r)$. Finalement, l'entraînement de l'auto-encodeur consiste à trouver le paramètre θ vérifiant :

$$\mathcal{J}_{\text{DAE}}(\theta) = \sum_t L(x^{(t)}, g_\theta(f_\theta(x^{(t)}))) \tag{9}$$

La forme la plus usuelle pour les fonctions f_θ et g_θ est :

$$\begin{array}{rcl} f_\theta(x) & = & s_f(b + Wx) \\ g_\theta(h) & = & s_g(d + W'h) \end{array} \tag{10}$$

où typiquement s_f et s_g sont des sigmoïdes ou des tangentes hyperboliques.

2.6.1 Implémentation du MLP

Nous avons implémenter l'algorithme le plus classique dans le cas de l'entraînement du MLP : la *backpropagation*. Cet algorithme se décompose en deux étape, une première de propagation, et une seconde de mise à jour des poids.

Initialisation. L'initialisation est cruciale dans ces algorithmes de *backpropagation*. De plus, comme nous allons utiliser le MLP dans le cas de poids initialisés par empilement de RBM - pour le DBN - notre implémentation doit prendre en compte deux possibilités. Soit on lui envoie les poids, soit on utilise le MLP à part entière et dans ce cas là les poids doivent être initialisés. Dans [2], il est expliqué que les poids doivent vérifiés (si n_1 et n_2 indiquent le nombre d'éléments au couches d'entrées et de sorties respectivement) :

$$\begin{array}{ll} \left[-\sqrt{\frac{6}{n_1+n_2}}, \sqrt{\frac{6}{n_1+n_2}}\right] & \text{pour activation en tanh} \\ \left[-4\sqrt{\frac{6}{n_1+n_2}}, 4\sqrt{\frac{6}{n_1+n_2}}\right] & \text{pour activation en sigmoïd} \end{array} \tag{11}$$

Propagation. La propagation correspond à une étape de montée du graphe pour générer les sorties correspondantes puis à une étape de descente qui génère des δ. Ceci nécessite une étiquette y que l'on compare avec la variable cachée finale h - éventuellement ceci peut être vectoriel. Nous verrons dans Alg.(1) l'algorithme précisément.

Mise à jour des poids. Pour chaque synapse on met à jour les poids avec les δ calculé par propagation. Ceci nécessite un paramètre, le *learning rate* qui influence fortement la qualité et la vitesse de l'apprentissage.

La prédiction dans le cas d'un MLP consiste uniquement à, une fois les poids établis, utiliser les données et monter le graphe pour retrouver leur représentation. En ce sens, on considère un MLP comme un pur outils de représentation

Algorithm 1 Entraînement du Multi-Layer Perceptron (MLP)

Input: X : matrice d'entrainement, y : étiquettes d'entraînement, l_r : *learning_rate*, *epochs* : nombre d'itérations, (W : liste de matrice de poids)
Output: W : liste de matrice de poids optimaux

```
Rajoute une colonne de 1 à la matrice X (biais),
for k ∈ range(epochs) do
    states = [X[i]] où i est un indice de ligne tiré aléatoirement,
    for j monte le graphe (propagation) do
        states.append(states[j] · W[j])
    end for
    error = y[i] - states[end]
    δ = [error ∗ ∂σ(states[end])]
    for j descend le graphe (backpropagation) do
        deltas.append((deltas[end] · W[j]^T) × ∂σ(states[j]))
    end for
    δ.revese()
    for i monte le graphe (mise à jour) do
        W[i] += l_r(vecteur de nombres de composant^T · δ)
    end for
end for
```

- vision non-paramétrique. On peut toutefois lui rajouter une régression logistique à l'issue - lorsque l'on dispose de données supervisées - qui permet d'en faire un outils de classification supervisée.

Références

[1] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning : A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[2] Yoshua Bengio and Xavier Glorot. Understanding the difficulty of training deep feedforward neural networks. 9 :249–256, May 2010.

[3] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, pages 504–507, 2006.

[4] Ruslan Salakhutdinov and Geoffrey Hinton. Deep Boltzmann machines. 5 :448–455, 2009.