Réseaux de neurones et modèles graphiques probabilistes

Florence d'Alché-Buc,
florence.dalche@telecom-paristech.fr

07-08/09/2015, Telecom Evolution, Paris, France

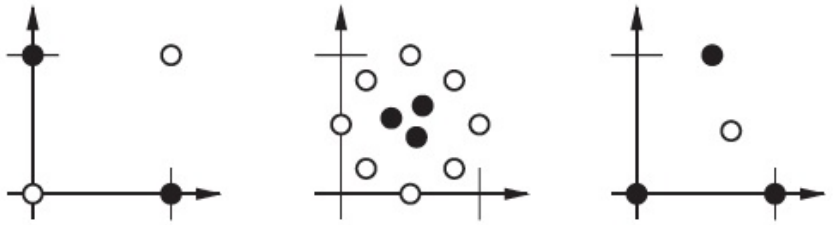
Programme du module

- Réseaux de neurones: perceptron multi-couches (Matin 07)
- Modèles graphiques probabilistes (07/09 et 08/09)
  - Machines de Boltzman restreintes
  - Réseaux bayésiens (Lionel Jouffe, Bayesia)
  - Modèles de Markov cachés pour le traitement de séquences (Laurence Likforman et son équipe)

Outline

Limites d'un perceptron

Le perceptron est un séparateur linéaire.



- XOR problem: un perceptron seul ne peut implémenter une fonction XOR
- Solution : on transforme les données en les plongeant dans un espace où elles sont linéairement séparables
  - Soit on rajoute une couche de neurones avant le neurone de sortie perceptron multi-couches (algorithme d'apprentissage par rétro-propagation du gradient), Werbos 1974, Le Cun 1985, Rumelhart et al. 1986.
  - Soit on transforme les données par une fonction de caractristique associée à un noyau.

Apprentissage d'un réseau de neurones multi-couches "feedforward"

- Pour apprendre un réseau de neurones, nous fixons *a priori* le nombre de neurones dans les couches cachées et le nombre de couches cachées
- En 1991, il a été prouvé qu'un MLP, réseau de neurones une couche cachée, était un approximateur universel
- Dans ce qui suit, nous expliquerons donc le fonctionnement en apprentissage et en prédiction des MLP une couche cachée.

Classification binaire supervisée

Cadre probabiliste et statistique

Soit X un vecteur aléatoire de $\mathcal{X} = \mathbb{R}^{p+1}$
X décrit ici le vecteur de caractéristiques ("features") avec une première coordonnée X_0 constante et égale à 1.
Y une variable aléatoire discrète $\mathcal{Y} = \{-1, 1\}$
Soit P la loi de probabilitéé jointe de (X,Y), loi fixée mais inconnue
Supposons que $S_{app} = \{(x_i, y_i), i = 1, \ldots, N\}$ soit un échantillon i.i.d. tiré de la loi P

- A partir de S_{app}, déterminer la fonction $h \in \mathcal{H}$ qui minimise $\mathcal{L}(h; S_{app})$

Apprendre un classifieur

Approche discriminante

- Définir
  - l'**espace de représentation** des données

Apprendre un classifieur

Approche discriminante

- Définir
  - l'**espace de représentation** des données
  - la **classe des fonctions** considérées

Apprendre un classifieur

Approche discriminante

- Définir
  - l'**espace de représentation** des données
  - la **classe des fonctions** considérées
  - la **fonction de coût** $\mathcal{L}$ à minimiser pour obtenir la meilleure fonction dans cette classe

Apprendre un classifieur

Approche discriminante

- Définir
  - l'**espace de représentation** des données
  - la **classe des fonctions** considérées
  - la **fonction de coût** $\mathcal{L}$ à minimiser pour obtenir la meilleure fonction dans cette classe
  - l'**algorithme de minimisation** de cette fonction de coût

Apprendre un classifieur

Approche discriminante

- Définir
  - l'**espace de représentation** des données
  - la **classe des fonctions** considérées
  - la **fonction de coût** $\mathcal{L}$ à minimiser pour obtenir la meilleure fonction dans cette classe
  - l'**algorithme de minimisation** de cette fonction de coût
  - une **méthode de sélection de modèle** pour définir les hyperparamètres

Apprendre un classifieur

Approche discriminante

- Définir
  - l'**espace de représentation** des données
  - la **classe des fonctions** considérées
  - la **fonction de coût** $\mathcal{L}$ à minimiser pour obtenir la meilleure fonction dans cette classe
  - l'**algorithme de minimisation** de cette fonction de coût
  - une **méthode de sélection de modèle** pour définir les hyperparamètres
  - une méthode d'évaluation des performances

Exemple d'un réseau de neurones multi-couches "feedforward"

Prenons comme exemple un MLP à une couche de sortie de taille K=1, une couche cachée de taille $M+1$, un vecteur d'entrée de taille $p+1$ pour la régression

Famille de fonctions $\mathcal{H}_{mlp} = \{h_{mlp} : \mathcal{R}^{p+1} \rightarrow \mathcal{Y}\}$

$$h_{MLP}(x) = \sum_{j=0}^{M} w_j^{(2)} z_j \quad (1)$$

$$z_j = \tanh(a_j) \quad (2)$$

$$a_j = \sum_{i=0}^{p} w_{ji}^{(1)} x_i \quad (3)$$

Architecture d'un réseau de neurones multi-couches "feedforward"

- Nous avons choisi: la sortie unique du régresseur MLP fournit une valeur rélle
- Pour un problème de classification $K = C$ classes, nous aurions choisi K sorties avec tanh
- Pour un problème de régression $K = C$ sorties, nous aurions plutôt choisi, K sorties linéaires (ici $K = 1$)

; $S_{app}) = \sum_{n=1}^{N}(h_{MLP}(x_n) - y_n)^2$(5)Importante remarque: $\mathcal{L}$ est non convexe et possède de nombreux minima locaux

- Le mieux que nous pouvons faire, c'est trouver un bon minimum local

- C'est principalement pour cette raison que les MLP ont été pendant longtemps abandonés en faveur des SVM

Algorithme d'optimisation

La rétropropagation du gradient

- L'idée est d'appliquer un algorithme de descente du gradient à chacune des couches, en débutant par la dernière couche
- On utilise la règle de dérivation en chaîne: $\frac{\partial L(W)}{\partial w_{ji}^{(1)}} = \frac{\partial L(W)}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}^{(1)}}$ pour pouvoir corriger les poids de la couche cachée.
- L'algorithme peut s'appliquer globalement ou localement (nous allons voir ce que cela signifie)

Références:

Y. LeCun: Une procédure d'apprentissage pour réseau à seuil asymmetrique (a Learning Scheme for Asymmetric Threshold Networks), Proceedings of Cognitiva 85, 599-604, Paris, France, 1985.

Rappelons la descente de gradient ordinaire

Soit une fonction $\mathcal{C}(\theta)$ dépendant de θ:

- Les valeurs θ telles que $\frac{\partial \mathcal{C}(\theta)}{\partial \theta} = 0$ correspondent des minima ou des maxima de cette fonction.
- Lorsque $\mathcal{C}$ est strictement convexe en θ, l'algorithme que nous allons présenter s'approche de la solution aussi prèsque possible.
- Cependant, même quand $\mathcal{C}$ n'est pas s. convexe, nous pouvons toujours essayer de trouver un "bon" minimum local.

Idée: corriger θ itérativement par: $\theta^{t+1} \leftarrow \theta^t - \eta_t \frac{\partial \mathcal{C}(\theta)}{\partial \theta}$
Après chaque mise à jour, le gradient est ré-évalué pour le nouveau vecteur de paramètre et la correction est à nouveau calculée

Rappel gradient pour un paramètre θ scalaire

Voir figure au tableau.

Rappelons la descente de gradient globale

$\mathcal{C}(\theta) = \sum_{n=1}^{N} c_n(\theta)$

1. E = 1000;
2. ϵ= petite valeur
3. θ^0 valeur initiale; t = 0;
4. Tant que $(E > \epsilon)$
   - $\theta^{t+1} \leftarrow \theta^t - \eta_t \sum_{n=1}^{N} \frac{\partial c_n(\theta^t)}{\partial \theta^t}$
   - calculer $E = L(\theta^{t+1})$
5. Fournir θ courant

Choix de η_k

Théorème:
Si la série $(\sum_k \eta_k)$ diverge et si $(\sum_k \eta_k^2)$ converge alors l'algorithme de gradient converge vers un minimum local.

Descente de gradient stochastique et locale

$\mathcal{C}(\theta) = \sum_{n=1}^{N} c_n(\theta)$

1. E = 1000;
2. ϵ= petite valeur
3. θ^0 valeur initiale; t = 0;
4. $nb_{cycle} = 0$
5. Tant que $(E \geq \epsilon)$ et $(nb_{cycle} < 500)$
   - $nb_{cycle} = nb_{cycle} + 1$
   - pour $\ell = 1$ à N
     - Tirer uniformément un indice $n \in \{1, \ldots, N\}$
     - $\theta^{t+1} \leftarrow \theta^t - \eta_t \frac{\partial c_n(\theta^t)}{\partial \theta^t}$
   - calculer $E = L(\theta^{t+1})$

Rétropropagation du gradient (1/4)

On souhaite appliquer la règle de descente de gradient aux poids de la couche 1 et de la couche 2 (ici réduite à une unité de sortie).
Soit $\ell_n = \frac{1}{2}(h(x_n) - y_n)^2$

Calcul pour une donnée, algorithme local

Correction des poids de sortie:

$$\frac{\partial \ell_n}{\partial w_j^{(2)}} = \frac{\partial \ell_n}{\partial h(x_n)} \frac{\partial h(x_n)}{\partial w_j^{(2)}} \tag{6}$$

Correction des poids de la couche cachée:

$$\frac{\partial \ell_n}{\partial w_{ji}^{(1)}} = \frac{\partial \ell_n}{\partial h(x_n)} \frac{\partial h(x_n)}{\partial w_{ji}^{(1)}} \tag{7}$$

Rétropropagation du gradient local (2/4): les calculs

Calcul pour une donnée, algorithme local

Corriger les poids de sortie:

$$\frac{\partial \ell_n}{\partial w_j^{(2)}} = \frac{\partial \ell_n}{\partial h(x_n)} \frac{\partial h(x_n)}{\partial w_j^{(2)}} \tag{8}$$

$$\frac{\partial \ell_n}{\partial h(x_n)} = h(x_n) - y_n \tag{9}$$

$$\frac{\partial h(x_n)}{\partial w_j^{(2)}} = \frac{\partial (w_j^{(2)} z_j + \sum_{k \neq j} w_k^{(2)} z_k)}{\partial w_j^{(2)}} \tag{10}$$

$$\frac{\partial h(x_n)}{\partial w_j^{(2)}} = z_j \tag{11}$$

(12)

Rétropropagation du gradient local (3/4): les calculs

Calcul pour une donnée, algorithme local

Corriger les poids de la couche cachée:

$$\frac{\partial \ell_n}{\partial w_{ji}^{(1)}} = \frac{\partial \ell_n}{\partial h(x_n)} \frac{\partial h(x_n)}{\partial w_{ji}^{(1)}} \tag{13}$$

$$\frac{\partial h(x_n)}{\partial w_{ji}^{(1)}} = (1 - (\tanh(a_j))^2) w_j^{(2)} x_i \tag{14}$$

Rétropropagation du gradient (4/4): l'algorithme

Pour une descente locale pour un exemple n tiré uniformément:

- Calculer pour x_n, $h(x_n)$
- Corriger la couche (2) : ici unique neurone de sortie
- Pour tout j=0 à M
  - $w_j^{(2),t+1} \leftarrow w_j^{(2),t} - \eta_t \frac{\partial \ell_n}{\partial w_j^{(2),t}}$
- Corriger la couche (1):
- Pour tout j=0 à M et i =0 à p:
  - $w_{ji}^{(1),t+1} \leftarrow w_{ji}^{(1),t} - \eta_t \frac{\partial \ell_n}{\partial w_{ji}^{(1),t}}$

Régularisation et early stopping

- **Early Stopping**
  - Une première méthode de régularisation a été proposée dans les années 90: il s'agit d'arrêter l'apprentissage avant le surapprentissage: on évite de se rapprocher trop près d'un minimum !
- **Régularisation**
  - On peut plus rigoureusement définir: $\mathcal{L}(W, \mathcal{S}_{app} = \sum_n (h(x_n) - y_n)^2 + \lambda_2 ||w^{(2),*}||^2 + \lambda_1 ||w^{(1),*}||^2$
  - On évitera de régulariser $w_0^{(2)}$, $w_{j0}^{(1)}$ et $w_{0i}^{(2)}$. Le $*$ signifie qu'on ne considère pas ces coordonnées-là.

En pratique, on note que : $||w^{(1),*}||^2 = \sum_{ji, j\neq 0, i\neq 0} (w_{ji}^{(1)})^2$.

Sélection de modèles

Le MLP a plusieurs hyperparamètres:

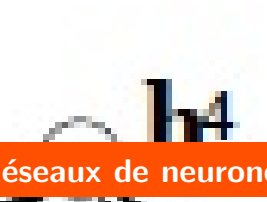
- Nb de couches cachées
- Tailles des couches cachées
- paramètre λ
- nb_{cycle}, ϵ
- $\eta_t = \frac{\gamma}{1+t}$

La plupart sont trouvés par VALIDATION CROISEE.

- **Pour**
  - Flexibilité au niveau des sorties: plusieurs classes, etc..
  - Technique éprouvé depuis 1985
  - Algorithme de gradient stochastique se plie bien aux besoins du BIG DATA
  - Bénéficie des architectures GPU
  - PLUG and PLAY: on peut enchaîner différents traitements dans un même paradigme
- **Contre**
- Fonction de perte non convexe : pas de minimum global
- Descente de gradient nécessite souvent de nombreux ajustemens
- Pas de cadre théorique
- Beaucoup de développements *ad hoc*

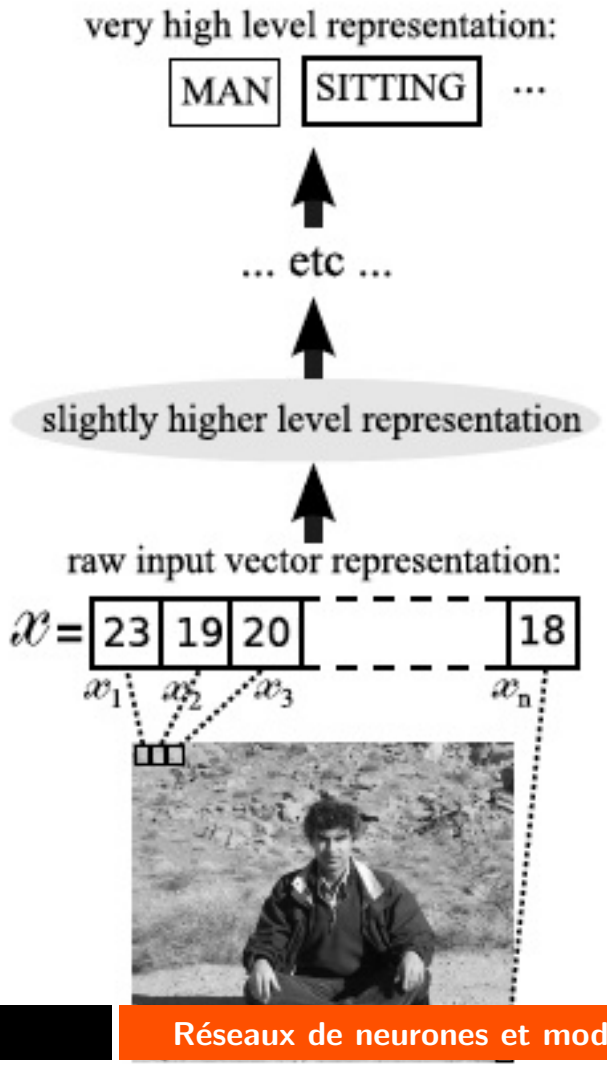
Apprentissage des réseaux dits profonds

A partir de 3 couches cachées, on parle de "deep learning", ce type de réseau est a priori intéressant pour traiter des données complexes comme des images ou du texte. Même si un réseau à une couche cachée est en principe un approximateur universel, cela ne veut pas dire qu'un réseau à une couche cachée fournit la meilleure représentation et les meilleures performances.
Malgré le risque de surapprentissage, on s'est intéressé récemment à l'apprentissage de réseaux profonds.
Apprendre un réseau profond par simple rétropropagation du gradient en fonctionne pas (Bengio et al. 2007; Erhan et al. 2009). Le réseau tombe dans des minima locaux très mauvais sans une bonne initialisation.



Apprentissage des réseaux dits profonds

Image Y. Bengio



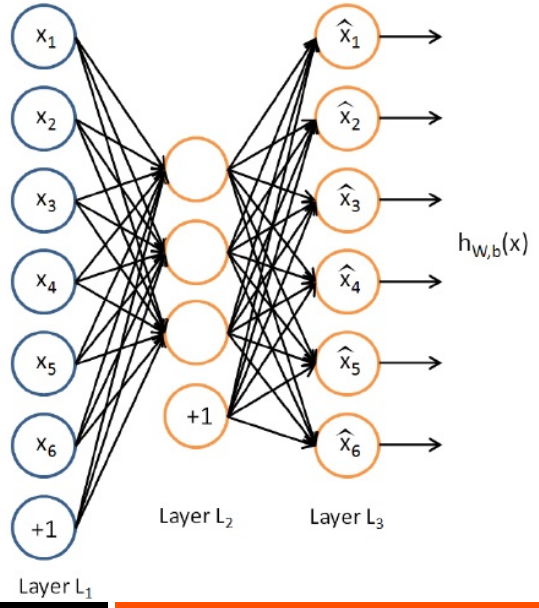
Apprentissage des réseaux dits profonds

Les réseaux à plusieurs couches sont aujourd'hui appris en étant initialisés par un apprentissage non supervisé souvent à l'aide d'autoencodeurs ou de Machines de Boltzman restreintes (RBM). Nous allons voir comment...

Autoencodeurs

Autoencodeurs

Un autoencodeur (aussi appelé *réseau diabolo*) est un réseau à une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. Ce type de réseau cherche à construire une représentation interne (la couche du milieu) en apprenant prédire l'entrée à partir de celle-ci: $x \approx g(x)$.



Apprentissage d'autoencodeurs

- Un auto-encodeur s'apprend par rétropropagation du gradient. Dans le cas des autoencodeurs parcimonieux, la fonction de perte utilisée est en général le critère quadratique pénalisé par un terme de régularisation qui contraint l'activité moyenne de chaque unité de la couche cachée à rester limitée.
- L'autoencodeur n'a d'autre intérêt que d'apprendre des représentations internes dans le cas de données complexs

Autoencodeur et apprentissage d'un réseau "feed-forward" profond (Erhan et al.)

Pour chaque couche cachée en démarrant par la plus proche de l'entrée x, on définit ses poids en les extrayant de la première couche d'un autoencodeur appris sur:

- Poids de la couche 2: on apprend un autoencodeur $x \approx g(x)$. On initialise les poids de la couche 2 par ceux de la couche 2 de l'autoencodeur
- Poids de la couche 3: on apprend un autoencodeur $h_1(x) \approx g(h_1(x))$. On initialise les poids de la couche 3 par ceux de la couche 2 de ce nouvel autoencodeur
- etc...
- Ensuite, on apprend de manière supervisée à partir de cette initialisation

Machines de Boltzman restreintes

Intéressons-nous maintenant à d'autres réseaux de neurones utiles à l'apprentissage des réseaux profonds. Ces réseaux diffèrent de ceux que nous avons vus précédemment car leurs unités de calcul sont stochastiques et non déterministes.
Nous arrivons ainsi à l'intersection entre le domaine des *réseaux de neurones* et celui des *modèles probabilistes graphiques*.

Outline

Définition d'un modèle graphique probabiliste

Jusqu'ici nous avons vu des modèles déterministes: la fonction calculée par un réseau de neurones de type MLP ou encore un arbre de décision, ou un SVM est une fonction déterministe. A une entrée, correspond une unique sortie.
Nous nous intéressons maintenant des modèles probabilistes: nous cherchons **modéliser la probabilité jointes de variables aléatoires**.

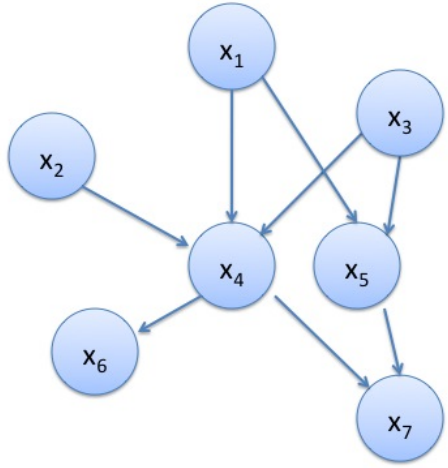
Exemple de modèle graphique probabiliste

- Modèle à variable latente: $p(x, z) = p(x|z)p(z)$

- Exemple: Modèle de mélange
  - Soit X variable aléatoire continue qui suit une loi de probabilité fixée mais inconnue P(X), nous supposons: $\mathrm{p(x)} = \sum_{k=1}^{K} \pi_k p(x|Z = k)$
  - Soit Z une variable aléatoire latente (i.e non observée). Z=k indique la classe de k: $p(x|Z = k)$ définit la densité de probabilité conditionnelle cette information de classe

Exemple : Réseau bayésien

$$p(x_1, x_2, \ldots, x_8) = p(x_1)p(x_2)p(x_3)p(x_4|x_1, x_2, x_3)p(x_5|x_1, x_3)p(x_6|x_7)p(x_7|x_4, x_5)$$



Utile pour le diagnostic, capturer des connaissances expertes.

Modèle graphique probabiliste

Définition

Un modèle graphique probabiliste est un graphe dont les noeuds sont associés à des variables aléatoires et les arêtes (dirigées ou non) entre deux noeuds définissent une relation probabiliste entre ces deux variables. Le graphe indique comment factoriser la loi de probabilité jointe.

Tâches principales:

- Apprendre ces modèles : par exemple, par maximisation de la vraisemblance pénalisée ou non
- Propager de l'information (inférence)
- Les utiliser pour générer des données

Modèle graphique probabiliste

Dans un graphe non orienté, l'absence d'arête entre deux noeuds indique une indépendance conditionnelle au reste du graphe.
Dans un graphe orienté, on dit que X a pour variable parente Y si une flèche va de Y vers X. X peut avoir plusieurs parents. Y peut avoir plusieurs enfants.

Intérêt des modèles graphiques probabilistes

- POUR
  - Cadre probabiliste : on a un modèlle ! pas uniquement de la prédiciton
  - possibilité d'enchaîner plusieurs modèles, propagation d'incertitude
  - Apprentissage fréquentiste ou bayésien
  - Apprentissage bayésien : les param`tres sont vus comme des variables alḿatoirs et on cherche à estimer $p(\theta|data)$ en utilisant les données et un *a priori* sur les donńdes: $p(\theta)$
  - Richesse des modèles
- CONTRE
  - Apprentissage pose des problèmes NP-difficiles, besoin d'approximations pour les résoudre
  - Gourmands en données

Modèle graphique probabiliste pour la classification

- Réseau bayésien avec une ou plusieurs variable de classe cible
- Approximation du classifieur bayesien:
  $\hat{P}(Y = k|x) = \frac{p(x|Y=k).p(Y=k)}{p(x)}$ où $p(x|Y = k)$ est modélisée par un modèle de mélange

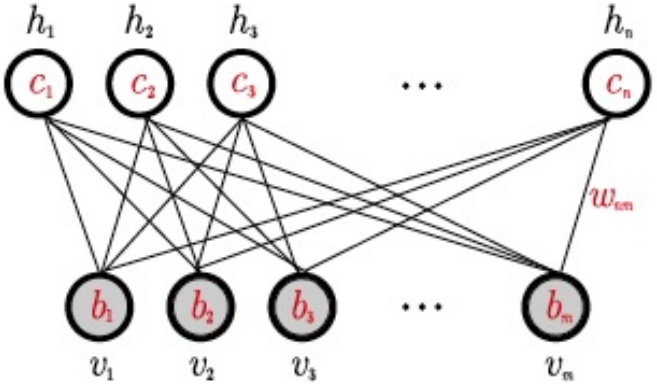
Outline

Réseaux de neurones stochastiques

Intéressons-nous à un autre type de réseau de neurones, les machines de Boltzman restreintes.

- En 1985, sont d'abord proposées les machines de Boltzman (Hinton, 1985): il s'agit de réseaux d'unités de calcul stochastique connectées de manière bidirectionnelle
- Malgré l'intérêt de ce type de réseau, l'apprentissage d'un tel réseau pour n'importe quel type de connexions est difficile.
- Mais le problème d'apprentissage se simplifie quand on impose des contraintes ce réseau

Machine de Boltzman restreinte



- Une machine de Boltzman restreinte est constituée de deux couches: une couche d'unités dites visibles et une couche d'unités cachées

Machine de Boltzman restreinte

- la couche d'unités visibles correspond aux variables observées
- cette couche d'entrée est reliée la couche d'unités dites cachées par des connexions **non orientées**
- chaque unité cachée i correspond une variable aléatoire binaire notée V_i. la taille de cette couche est notée m.
- la taille de l'entrée est notée p.

Machine de Boltzman restreinte

But: Modéliser les deux types de variables visibles et cachées. Une fois le modèle appris, pour une nouvelle observation, on peut trouver le vecteur des valeurs cachées: ce vecteur peut tre vu comme un vecteur de caractéristiques qui fournit une autre représentation de l'observation données.
Les RBM sont utilisées pour découvrir automatique de nouvelles **représentations** des données qui peuvent ensuite tre injectées comme **entrées** dans des réseaux de neurones "feedforward" ou mme dans d'autres classifieurs ou régresseurs (SVM, arbres).

Machine de Boltzman restreinte: vision modèle graphique

Comment calculer la sortie de ce réseau ?

Il faut décrire: P(V=v, H=h) la loi de probabilité jointe des deux types de vecteurs aléatoires binaires

$P(V = v, H = h) = \frac{\exp(-E(v,h))}{Z}$

On appelle ce type de loi une distribution de Gibbs

La fonction E(v,h) est une fonction d'énergie

Machine de Boltzman restreinte: vision modèle graphique

$$E(h, v) = -\sum_{ij} w_{ij} h_i v_j - \sum_j b_j v_j - \sum_i c_i h_i \qquad (15)$$

Nous avons les poids w_{ij} qui relient l'unité visible j l'unité cachée i ainsi que des termes de biais qui pondèrent l'importance relative des chaque v_j ou h_j

Indépendance conditionnelle:

$P(H_i = h_i, H_j = h_j | V = v) = P(H_i | V = v) P(H_j | V = v)$

$P(V_i = v_i, V_j = v_j | H = h) = P(V_i | H = h) P(V_j | H = h)$

Machine de Boltzman restreinte: vision réseau de neurones

Comment calculer la sortie de ce réseau ?

$$P(H_i = 1|V = v) = \sigma(\sum_j w_{ij} v_j + c_i) \tag{16}$$

and

$$P(V_j = 1|H = h) = \sigma(\sum_i w_{ij} h_i + b_j) \tag{17}$$

Apprentissage des poids d'une RBM comme un modèle graphique

Par maximisation de la vraisemblance d'après Fisher et Igel, 2012 (tutorial RBM):

$$\begin{aligned}\frac{\partial \ln \mathcal{L}(\boldsymbol{\theta} \,|\, \boldsymbol{v})}{\partial w_{ij}} &= -\sum_{\boldsymbol{h}} p(\boldsymbol{h} \,|\, \boldsymbol{v}) \frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial w_{ij}} + \sum_{\boldsymbol{v}, \boldsymbol{h}} p(\boldsymbol{v}, \boldsymbol{h}) \frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial w_{ij}} \\ &= \sum_{\boldsymbol{h}} p(\boldsymbol{h} \,|\, \boldsymbol{v}) h_i v_j - \sum_{\boldsymbol{v}} p(\boldsymbol{v}) \sum_{\boldsymbol{h}} p(\boldsymbol{h} \,|\, \boldsymbol{v}) h_i v_j \\ &= p(H_i = 1 | \, \boldsymbol{v}) v_j - \sum_{\boldsymbol{v}} p(\boldsymbol{v}) p(H_i = 1 | \, \boldsymbol{v}) v_j \ .\end{aligned}$$

$$\begin{aligned}\frac{1}{\ell} \sum_{\boldsymbol{v} \in S} \frac{\partial \ln \mathcal{L}(\boldsymbol{\theta} \,|\, \boldsymbol{v})}{\partial w_{ij}} &= \frac{1}{\ell} \sum_{\boldsymbol{v} \in S} \left[-\mathbb{E}_{p(\boldsymbol{h} \,|\, \boldsymbol{v})} \left[\frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial w_{ij}} \right] + \mathbb{E}_{p(\boldsymbol{h}, \boldsymbol{v})} \left[\frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial w_{ij}} \right] \right] \\ &= \frac{1}{\ell} \sum_{\boldsymbol{v} \in S} \left[\mathbb{E}_{p(\boldsymbol{h} \,|\, \boldsymbol{v})} \left[v_i h_j \right] - \mathbb{E}_{p(\boldsymbol{h}, \boldsymbol{v})} \left[v_i h_j \right] \right]\end{aligned}$$

Problème: des approximations sont nécessaires: on échantillonne suivant la loi courante. (Contrastive divergence algorithm).

Apprentissage des poids d'une RBM comme un modèle graphique

Algorithm 1. k-step contrastive divergence

Input: RBM $(V_1, \ldots, V_m, H_1, \ldots, H_n)$, training batch S
Output: gradient approximation Δw_{ij}, Δb_j and Δc_i for $i = 1, \ldots, n$, $j = 1, \ldots, m$

```
init Δw_ij = Δb_j = Δc_i = 0 for i = 1,...,n, j = 1,...,m
forall the v ∈ S do
    v^(0) ← v
    for t = 0,...,k − 1 do
        for i = 1,...,n do sample h_i^(t) ~ p(h_i | v^(t))
        for j = 1,...,m do sample v_j^(t+1) ~ p(v_j | h^(t))
    for i = 1,...,n, j = 1,...,m do
        Δw_ij ← Δw_ij + p(H_i = 1 | v^(0)) · v_j^(0) − p(H_i = 1 | v^(k)) · v_j^(k)
        Δb_j ← Δb_j + v_j^(0) − v_j^(k)
        Δc_i ← Δc_i + p(H_i = 1 | v^(0)) − p(H_i = 1 | v^(k))
```

Cet algorithme est implémenté en scikit-learn. Voir l'application: `http://scikit-learn.org/stable/auto_examples/plot_rbm_logistic_classification.html`

Utilisation d'une RBM en pré-traitement d'un classifieur

La RBM apprise permet de pré-traiter les entrées.
On extrait ainsi les valeurs les plus probables des variables latentes et on les donne en entre d'un régresseur logistique; les performances sont considérablement améliorées.

Références

- An introduction to Restricted Boltzman machines, Fisher and Iger.
- Notes de cours IT6266, Université de Montréal, Equipe de Yoshua Bengio.
- Learning Deep Architectures for AI, Yoshua Bengio, Foundations Trends in Machine Learning, 2009
- Pattern Recognition and Machine Learning, C. Bishop, Springer, 2006.
- `http://deeplearning.net/tutorial/`: pour tou document y compris implémentations...