

SD207 - Réseaux de neurones et données non structurées

Florence d'Alché-Buc

Institut Mines-Télécom, Télécom ParisTech, LTCI CNRS

`florence.dalche@telecom-paristech.fr`



`first_deep.jpg`

Neuron network growth over 24 hours

growing-nets.jpg

Développement des réseaux de neurones chez l'enfant

connexion.jpg

Neurone

slide_neuron.jpg

Réseau de neurones formels (perceptron multi-couches)

reseau.jpg

Du neurone formel aux réseaux de neurones formels

1/2

- Neurone formel: Mc Culloch et Pitts, 1943
- Règle d'apprentissage du perceptron, Rosenblatt, 1957
- Minsky et Papert: capacité limitée du perceptron, 1959
- Apprentissage d'un perceptron multi-couches par rétropropagation du gradient, Y. Le Cun, 1985, Hinton et Sejnowski, 1986.
- Perceptron multi-couches = approximateur universel, Hornik et al. 1991
- Convolutional networks, 1995, Y. Le Cun et Y. Bengio
- Entre 1995 et 2008, peu d'expansion du domaine (pbs fonctions non convexes, temps d'apprentissage, pas de théorie)

Du neurone formel aux réseaux de neurones formels

1/2

- Généralisation des GPU (processeurs graphiques) 2005
- Très large ensemble d'images: Imagenet, Fei-Fei et al. 2008 (maintenant 11 millions d'images)
- Réseaux de neurones de plus en plus profonds appris avec de gigantesques bases de données
- Apprentissage initial non supervisé (avec auto encodeur)
- Word2vec (Mikolov et al. 2013)
- Dropout (Srivastava et al. 2014)

Programme du cours

- Données non structurées
- Rappel: neurone formel ou perceptron
- Perceptron multi-couches
- Autoencodeurs
- Un mot sur les réseaux convolutionnels

Outline

Définition du neurone formel

- une fonction d'activation
- un vecteur de poids et un biais (intercept)

$$f(x) = g(w^T x + b) \quad (1)$$

On choisit g différentiable

Fonctions d'activation du neurone formel

par exemple:



Mais aussi tanh (sortie entre -1 et 1).

Limitation du neurone formel

Limité aux données linéairement séparables:

counter-ex.jpg

Ajouter une couche de traitement intermédiaire

$$\Phi(x)_1 = \text{AND}(\bar{x}_1, x_2)$$

$$\Phi(x)_2 = \text{AND}(x_1, \bar{x}_2)$$

Maintenant, calculer :

$$f(x) = g(\Phi(x)^T w + b)$$

On parle de fonction de redescription (feature map) ou de représentation interne.

Grand avantage des réseaux de neurones à plus d'une couche:
apprentissage de la fonction Φ .

Approximateur universel

En 1991, Hornik et al. démontrent que la famille des perceptrons à une couche cachée et à $p + 1$ entrées est dense dans l'ensemble des fonctions continues de \mathbb{R}^p dans \mathbb{R} . Un MLP à une couche cachée est un **approximateur universel**.

Approximateur universel

En 1991, Hornik et al. démontrent que la famille des perceptrons à une couche cachée et à $p + 1$ entrées est dense dans l'ensemble des fonctions continues de \mathbb{R}^p dans \mathbb{R} . Un MLP à une couche cachée est un **approximateur universel**.

D'autres exemples d'approximateurs universels que vous connaissez :

- Régresseur linéaire : **NON**
- SVM avec noyau universel tel que le noyau Gaussien: **OUI**
- Forêts aléatoires : **OUI**
- Boosting de stumps: **OUI**

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

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

$$z_j = \tanh(a_j) \quad (3)$$

$$a_j = \sum_{i=0}^p w_{ji}^{(1)} x_i \quad (4)$$

Remarque sur la fonction de saturation

La tangente hyperbolique est choisie comme fonction de saturation, dérivable.

$$h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \quad (5)$$

$$h'(a) = 1 - h(a)^2 \quad (6)$$

En termes de calculs, cette fonction est très avantageuse car la dérivée se définit directement en terme de $h(a)$. Nous avons une propriété similaire pour la fonction sigmoïde: $g(a) = \frac{1}{1 + \exp(-\frac{1}{2}a)}$.

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

- Nous avons choisi: la sortie unique du régresseur MLP fournit une valeur réelle
- Pour un problème de classification à K classes, nous aurions choisi K sorties avec la fonction sigmoïde ou mieux softmax
- Pour un problème de régression à K sorties, nous aurions plutôt choisi, K sorties linéaires (ici $K = 1$)

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

$$h_c(x) = g\left(\sum_{j=0}^M w_{jc}^{(2)} z_j\right) \quad (7)$$

$$z_j = g(a_j) \quad (8)$$

$$a_j = \sum_{i=0}^p w_{ji}^{(1)} x_i \quad (9)$$

avec $g(t) = \frac{1}{1 + \exp(-1/2t)}$.

Apprentissage à partir de données

$$\mathcal{L}(W; \mathcal{S}) = \sum_{n=1}^N \ell(h(x_n), y_n)$$

Régression :

$$\ell(h(x_n), y_n) = (h(x_n) - y_n)^2$$

Classification (maximiser la vraisemblance): On interprète $f_c(x) = p(y = c|x)$ (plusieurs sorties: on peut utiliser la fonction softmax)

$$\ell(h(x), y) = -\log f_y(x)$$

Importante remarque: \mathcal{L} est non convexe et possède de nombreux minima locaux

- Le mieux que nous puissions faire, c'est trouver un bon minimum local
- C'est principalement pour cette raison que les MLP ont été pendant longtemps abandonnés en faveur des SVM/SVR plus faciles à optimiser

Algorithme d'optimisation

La rétropropagation du gradient

- L'idée est d'appliquer un algorithme de descente du gradient: on rétropropage une erreur à travers 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.
- Une fois toutes les corrections calculées, on met à jour les poids du réseau.
- L'algorithme peut s'appliquer globalement ou localement (nous allons voir ce que cela signifie)

La rétropropagation du gradient

Références:

- Y. LeCun: Une procédure d'apprentissage pour réseau à seuil asymétrique (a Learning Scheme for Asymmetric Threshold Networks), Proceedings of Cognitiva 85, 599-604, Paris, France, 1985.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning representations by back-propagating errors. Nature, 323, 533–536.

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 à 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 près que 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

Rappelons la descente de gradient globale

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

- ➊ $E = 1000$;
- ➋ $\varepsilon =$ petite valeur
- ➌ θ^0 valeur initiale; $t = 0$;
- ➍ Tant que ($E > \varepsilon$)
 - ▶ $\theta^{t+1} \leftarrow \theta^t - \eta_t \sum_{n=1}^N \frac{\partial c_n(\theta^t)}{\partial \theta^t}$
 - ▶ calculer $E = L(\theta^{t+1})$
- ➎ 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 $(E \geq \varepsilon)$ et $(nb_{cycle} < 500)$
 - ▶ $nb_{cycle} = nb_{cycle} + 1$
 - ▶ pour $\ell = 1$ à N
 - ★ Tirer uniformément un indice $n \in \{1, \dots, N\}$
 - ★ $\theta^{t+1} \leftarrow \theta^t - \eta_t \frac{\partial c_n(\theta^t)}{\partial \theta^t}$
 - ▶ calculer $E = L(\theta^{t+1})$

Descente de gradient stochastique avec minibatch de taille constante

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

- ❶ $E = 1000$;
- ❷ ε = petite valeur
- ❸ θ^0 valeur initiale; $t = 0$;
- ❹ $nb_{cycle} = 0$
- ❺ Tant que $(E \geq \varepsilon)$ et $(nb_{cycle} < 500)$
 - ▶ $nb_{cycle} = nb_{cycle} + 1$
 - ★ Tirer uniformément M fois un indice $n \in \{1, \dots, N\}$
 - ★ $\theta^{t+1} \leftarrow \theta^t - \eta_t \frac{\partial c_M(\theta^t)}{\partial \theta^t}$
 - ▶ 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 $\ell = \frac{1}{2}(h(x) - y)^2$

Calcul pour une donnée, algorithme local

Gradient par rapport aux poids de sortie:

$$\frac{\partial \ell}{\partial w_j^{(2)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_j^{(2)}} \quad (10)$$

Gradient par rapport aux poids de la couche cachée:

$$\frac{\partial \ell}{\partial w_{ji}^{(1)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_{ji}^{(1)}} \quad (11)$$

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

Calcul pour une donnée, algorithme local

Gradient pour les poids de sortie:

$$\frac{\partial \ell}{\partial w_j^{(2)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_j^{(2)}} \quad (12)$$

$$\frac{\partial \ell}{\partial h(x)} = h(x) - y \quad (13)$$

$$\frac{\partial h(x)}{\partial w_j^{(2)}} = \frac{\partial g(w_j^{(2)} z_j + \sum_{k \neq j} w_k^{(2)} z_k)}{\partial w_j^{(2)}} \quad (14)$$

$$(15)$$

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

Calcul pour une donnée, algorithme local

Gradient pour les poids de la couche cachée:

$$\frac{\partial \ell}{\partial w_{ji}^{(1)}} = \frac{\partial \ell}{\partial h(x)} \frac{\partial h(x)}{\partial w_{ji}^{(1)}} \quad (16)$$

$$\frac{\partial h(x)}{\partial w_{ji}^{(1)}} = w_j^{(2)} \frac{\partial g(\sum_k w_{jk} x_k)}{\partial w_{ji}^{(1)}} \quad (17)$$

$$\frac{\partial h(x)}{\partial w_{ji}^{(1)}} = (1 - g(\sum_k w_{jk} x_k)^2) w_j^{(2)} x_i \quad (18)$$

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

Pour une descente locale pour un exemple x_n tiré uniformément:

- ❶ Calculer pour x_n , $h(x_n)$
- ❷ Calculer les gradients: $\frac{\partial \ell_n}{\partial w_j^{(2),t}}$ puis $\frac{\partial \ell_n}{\partial w_{ji}^{(1),t}}$
- ❸ Corriger tous les poids avec les gradients préalablement calculés:
 - ▶ Corriger la couche (1):
 - ▶ Pour tout $j=0$ à M :
 - ★ $w_j^{(1),t+1} \leftarrow w_j^{(1),t} - \eta_t \frac{\partial \ell_n}{\partial w_j^{(1),t}}$
 - ▶ Corriger la couche (2) : ici unique neurone de sortie
 - ★ $w^{(2),t+1} \leftarrow w^{(2),t} - \eta_t \frac{\partial \ell_n}{\partial w^{(2),t}}$

Outline

Régularisation et early stopping

• Early Stopping

- ▶ Une première méthode de régularisation a été proposée dans les années 90: il s'agit d'arrêter *a priori* l'apprentissage prématurément avant le sur-apprentissage: on évite de se rapprocher trop près d'un minimum !

• Régularisation

- ▶ On peut plus rigoureusement définir:
$$\mathcal{L}(W, \mathcal{S}_{app} = \sum_n \ell(h(x_n), y_n) + \lambda_2 \|w^{(2),*}\|^2 + \lambda_1 \|w^{(1),*}\|^2$$
- ▶ 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 : $\|w^{(1),*}\|^2 = \sum_{j,i,j \neq 0, i \neq 0} (w_{ji}^{(1)})^2$.

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 ajustements
- Pas de cadre théorique
- Beaucoup de développements *ad hoc*

Outline

Réseaux dits profonds

Image Y. Bengio



first_deep.jpg

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.

Pourquoi utiliser plusieurs couches cachées?

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.

Apprentissage des réseaux dits profonds

Malgré le risque de surapprentissage,

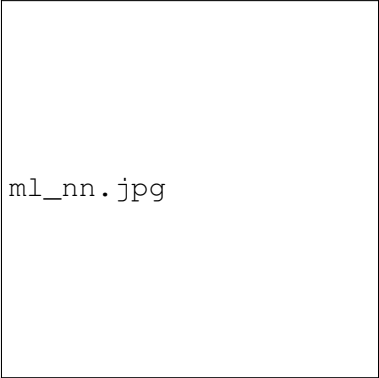
deux bonnes raisons de s'intéresser aux réseaux profonds

- l'amélioration des capacités de calcul et de mémoire (GPU)
- la disponibilité de gigantesques bases de données (Imagenet, Fei-Fei, 2008)

Apprendre un réseau profond par simple rétropropagation du gradient ne fonctionne pas si bien que cela (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



ml_nn.jpg

Apprentissage des réseaux dits profonds

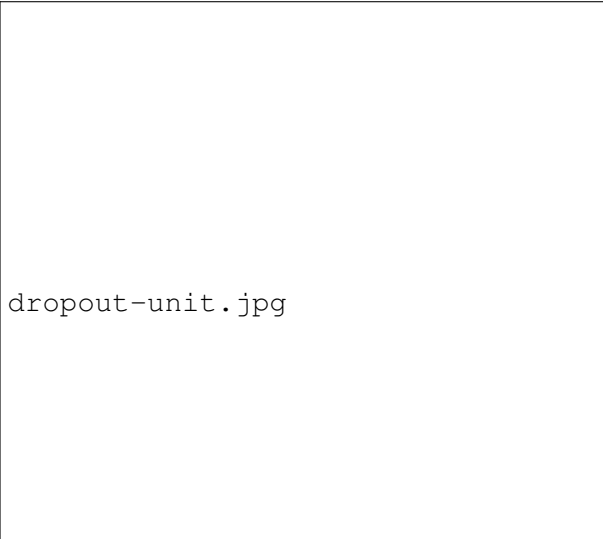
Deux améliorations notables

- Dropout
- Auto-encodeurs

Eviter l'overfitting des réseaux profonds par *dropout*

1/3

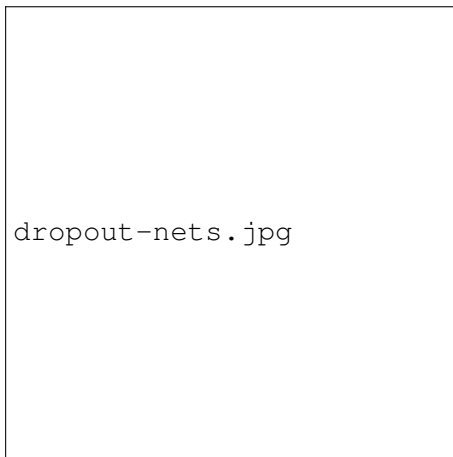
Pour les réseaux profonds (> >2 couches):



dropout-unit.jpg

Eviter l'overfitting des réseaux profonds par *dropout*

2/3




Une interprétation:

Si on dispose de m neurones au total, c'est comme si on apprenait avec 2^m réseaux clairsemés et au moment du test, on superpose tous

Eviter l'overfitting des réseaux profonds par *dropout*

3/3



`dropout-res.jpg`

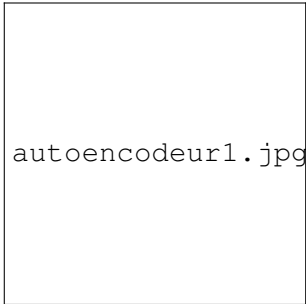
Apprentissage des réseaux dits profonds

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 diabololo*) 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 à prédire l'entrée à partir de celle-ci: $x \approx g(x)$.



autoencodeur1.jpg

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 complexes

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 $x \approx h(x)$. 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 $f_1(x) \approx h(f_1(x))$. 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

Réseaux convolutionnels pour les images

Y. Le Cun.

conv-net.jpg

Cortex visuel

visual-cortex.png

Different layers

Receptive fields (convolution layer): radial basis functions (gaussian kernels)

Max pooling:



Skip-gram modele pour codage de mots

Mikolov et al. 2013.



NB: représentations continues des mots déjà proposées en 2002/2003.

Références

- Le super cours de Hugo Larochelle (youtube)
- Notes de cours IT6266, Université de Montréal, Equipe de Yoshua Bengio.
- Learning Deep Architectures for AI, Yoshua Bengio, Foundations Trends in Machine Learning, 2009
- Dropout: A simple way to prevent overfitting, Srivastava et al. JMLR 2014
- Pattern Recognition and Machine Learning, C. Bishop, Springer, 2006.
- <http://deeplearning.net/tutorial/>: pour tout document y compris implémentations...