Réseaux à composantes continues

Emmanuel Bengio

28 juillet 2014

Machine Learning

- Utiliser des données pour apprendre quelque chose d'utile
  - Classifier
  - Prédire une valeur
  - Prédire une probabilité
- Images, son, ADN, texte, symptômes, météo, etc.
- Algorithmes statistiques, optimisation
- On apprends à partir des données du monde réel

Deep Learning

- Tourne autour de la notion de représentation
- On “empile” plusieurs couches de transformations
- Abstraction
- Apprentissage

Représentation?

- Pixels, intensités sonores, *bag of words*
- Features “géométriques”
- Probabilités
- Représentation intermédiaire quelconque

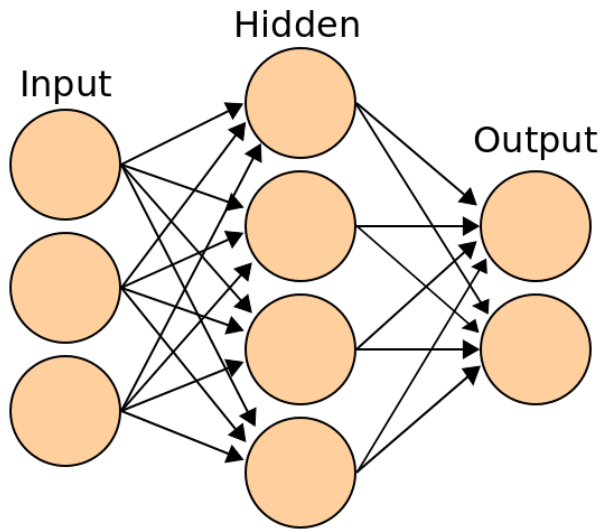
Bonne représentation?

- Question difficile, quelques pistes:
  - “Démêler” les facteurs de variations
  - Préserver l'information
- Comment passer de données brutes à des bonnes représentations?
  - PCA, clustering
  - Sparse coding
  - Manifold learning
  - Autoencodeurs

Réseaux de neurones

Couche cachée typique, $\boldsymbol{h}$ est une représentation intermédiaire.

$$\boldsymbol{h} = f(W\boldsymbol{x} + \boldsymbol{b}) \quad f(\boldsymbol{t}) = \frac{1}{1 + e^{-\boldsymbol{t}}}$$



$$\boldsymbol{y} = f(W^{(y)}\boldsymbol{h} + \boldsymbol{b}^{(y)})$$

Réseaux de neurones

- Apprentissage des W et $\boldsymbol{b}$ via descente de gradient, on minimise une fonction de perte $\mathcal{L}$ par $\frac{\partial W_{ij}}{\mathcal{L}}, \frac{\partial b_j}{\mathcal{L}}$
- Classification, régression
  $\mathcal{L}(y, t) = -(y \log t + (1 - y) \log(1 - t))$
- Autoencodeurs
  $\mathcal{L}(x, x^{(r)}) = \left(x - x^{(r)}\right)^2$

Autoencodeurs

$$x \rightarrow s \rightarrow x^{(r)}$$

On reconstruit $x^{(r)}$ depuis la représentation intermédiaire s, tel que $x^{(r)}$ soit fidèle à x.

$$\begin{aligned} \boldsymbol{s} &= f(\boldsymbol{b}^{(s)} + W^{(s)} f(\boldsymbol{b}^{(x)} + W^{(x)} \boldsymbol{x})) \\ \boldsymbol{x}^{(r)} &= f(\boldsymbol{b}^{(xr)} + W^{(x)^T} f(\boldsymbol{b}^{(sr)} + W^{(s)^T} \boldsymbol{s})) \end{aligned}$$

s devrait donc préserver l'information.

Autoencodeurs

- Denoising autoencoder
  - On force le réseau à débruiter son entrée
- Contractive autoencoder
  - On force le réseau à moins faire varier la représentation dans certaines directions

Motivation

- Données brutes en très haute dimension d
- Hypothèse du manifold
  - $n \ll d$ facteurs de variations des données
- Activations binaires des représentations
- Une représentation continue (au sens topologique), plus abstraite?
  - Translation, rotation
  - Degré d'expression
  - Paramètres d'identité

Composantes continues

- On force la reconstruction de $\boldsymbol{h}$ à s'activer de manière binaire autour de certaines configurations de $\boldsymbol{s}$.

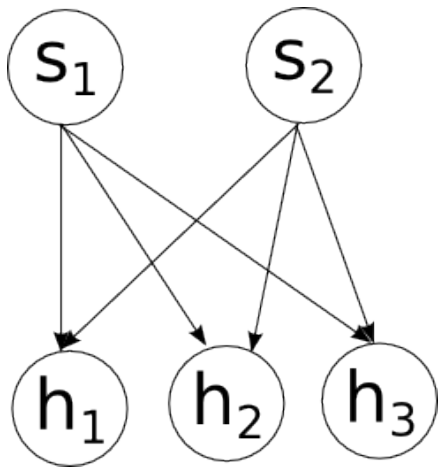
$$h_i^{(r)} = \exp\left(-(s-\mu_i)^T D_i (s-\mu_i)\right)$$

Une unité $h_i^{(r)}$ donnée est active si s est proche de μ_i (un paramètre) dans les dimensions spécifiées par D_i (autre paramètre)

Composantes continues

$$h_i^{(r)} = \exp\left(-(s-\mu_i)^T D_i (s-\mu_i)\right)$$

- s_j, ensemble de μ_i, d'où le "continu"
- e.g. translation vers la droite, 10 μ_i pour les 10 translations possibles (0;0.1;0.2;...;0.8;0.9)
- Chaque valeur active un *unique* h_i.

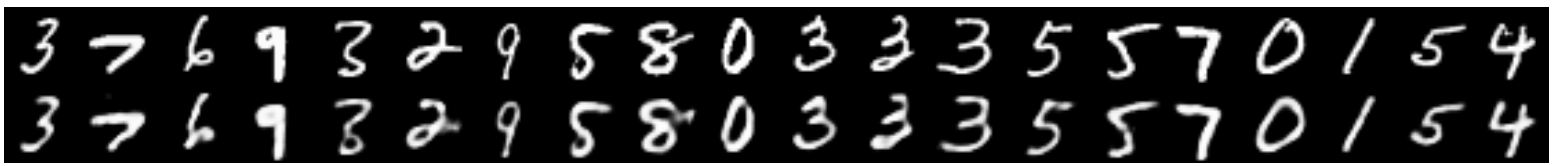


Projet

- Theano, librairie Python de calcul multi-dimensionnel
- Code qui crée, optimise et teste notre modèle (ainsi qu'un baseline)
- Code pour *finetune* ces modèles (apprentissage supervisé)
- Code pour visualiser les poids, les gradients, et les résultats (très important!)

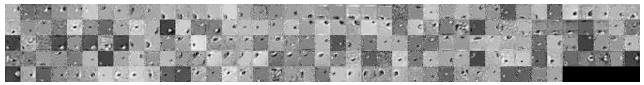
Apprentissage non-supervisé

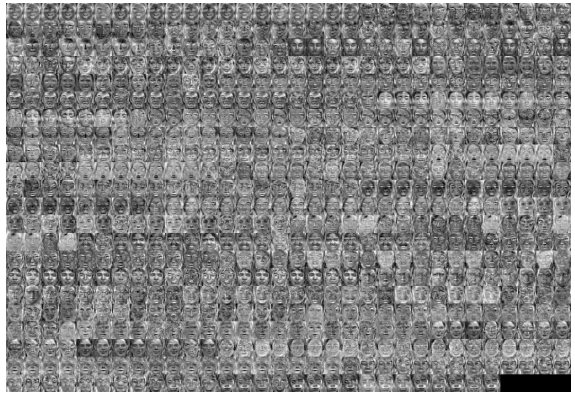
Visages (TFD), MNIST, figures géométriques.



Apprentissage non-supervisé

Les poids de la première couche





Problèmes

- Très sensible au taux d'apprentissage
- Sensible à la configuration de D et μ
- Long à entrainer

Résultats numériques

En termes d'erreur de reconstruction, notre modèle fait presque aussi bien qu'un autoencodeur standard.
Sur MNIST, ~10% de plus (log-likelihood)
Sur *Faces*, ~3-5% de plus

Apprentissage supervisé

Si on utilise les poids appris dans la dernière tâche et qu'on les *finetune* pour apprendre à classifier, on obtient de bons résultats.

	MNIST	*Faces*
Notre méthode	12.96%	17.60%
AE	6.52%	19.50%
Réseau à convolution	0.9%	14.2%

Ici en fait j'ai comparé notre méthode avec un autoencodeur *à capacité égale.*

Futur

- On peut faire mieux
  - Initialisation de D
  - architecture
- Comparer avec d'autres méthodes similaire
- Fonctions de régularisation

Projet

- source disponible sur github: github.com/bengioe/ccae
- rapport et comptes rendus: bengioe.github.io/ccae