Réseaux de neurones

Machine learning

Le Chenadec Gilles (M103)
Quidu Isabelle & Thomas Hélène

Objectifs du cours

Introduction

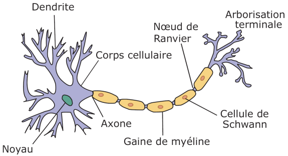
Apprentissage supervisé

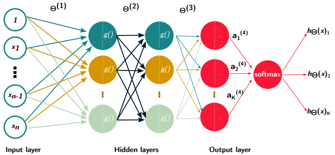
- Régression
- Classification supervisée
- Régression logistique
- Réseaux de neurones

Réseaux de neurones

Origines:

- Imitation le cerveau humain: inspiration du fonctionnement élémentaire du système nerveux
- Ont été très utilisé dans les années 80 et début 90 puis leur popularité a diminué.





Avantages

- Récente résurgence: technique State-of-the-art pour beaucoup d'applications
- Pas de modélisation analytique
- Adaptabilité aux données
- Architecture parallèle

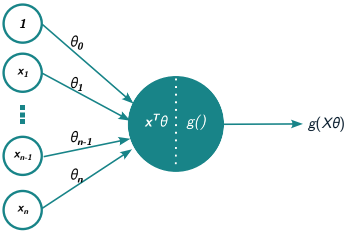
Inconvénients

- Boîte noire: apprentissage d'une fonction complexe incompréhensible
- Sur-apprentissage

Un neurone

Un neurone est une unité de calculs en 3 phases:

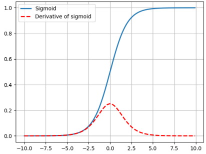
- lit l'information décrite par ses entrées ($\mathbf{x} = [1, x_1 \ldots, x_n]^T$)
- combine linéairement les entrées
  - $z(\mathbf{x}) = \mathbf{x}^T\boldsymbol{\theta} = \theta_0 + \sum_{i=1}^{n} \theta_i x_i$
  - paramètres: $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_n]^T$
- passe le résultat dans une fonction linéaire
  - $a(\mathbf{x}) = g(z(\mathbf{x})) = g\left(\mathbf{x}^T\boldsymbol{\theta}\right)$
  - $g(.)$: fonction d'activation (sigmoïde, ReLU)



Différents choix de la fonction d'activation

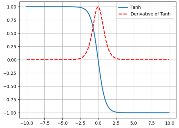
Sigmoïde: $g(x) = \sigma(x) = \frac{1}{1+\exp^{-x}}$

- Régression logistique
- rend les valeurs entre 0 et 1
- toujours positif, borné, monotone croissante



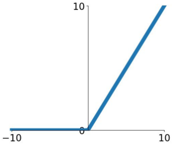
Tangente hyperbolique $g(x) = \tanh(x) = \frac{\exp^{2x} - 1}{\exp^{2x} + 1}$

- rend les valeurs entre -1 et 1
- positif et négatif, borné, monotone croissante



Rectified linear unite (ReLU): $g(x) = \mathbf{ReLU}(x) = max(0, x)$

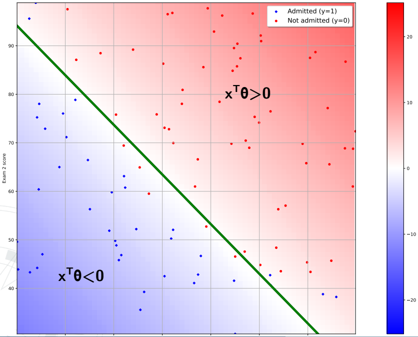
- toujours positive ou nulle, borné à 0 pour les valeurs négatives, pas de borne supérieure
- monotone croissante, tend à donner des neurones avec des activations parcimonieuses (*sparse*) i.e une large gamme de valeurs d'entrée (toutes celles négatives) sera mise à 0.

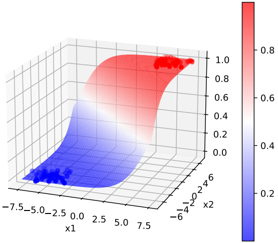


Capacité d'un neurone

Un neurone peut:

- résoudre uniquement de **classification binaire**
  - En utilisant la fonction d'activation sigmoïde (bornée entre 0 et 1), un neurone peut être interprété comme un estimateur de la probabilité de $h_{\boldsymbol{\theta}}(\mathbf{x}) = a(\mathbf{x}) = \mathbb{P}\left\{y = 1|\mathbf{x}\right\}, y \in \{0, 1\}$
  - connu comme le **classifieur de régression logistique**
  - si $h_{\boldsymbol{\theta}}(\mathbf{x}) > 0.5$ alors la classe 1 est prédite, sinon c'est la classe 0.
- La fonction d'activation $\tanh$ peut mener à un classification binaire équivalente.

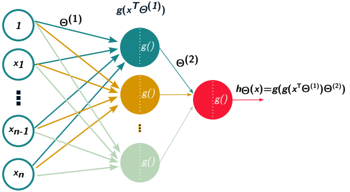




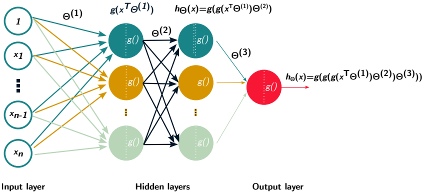
Réseau de neurones avec des couches cachées

Le **réseau de neurones entièrement connectés** permet de résoudre des problèmes avec des frontières non-linéaires:

- par empilement des neurones entièrement connectées dans une couche cachée



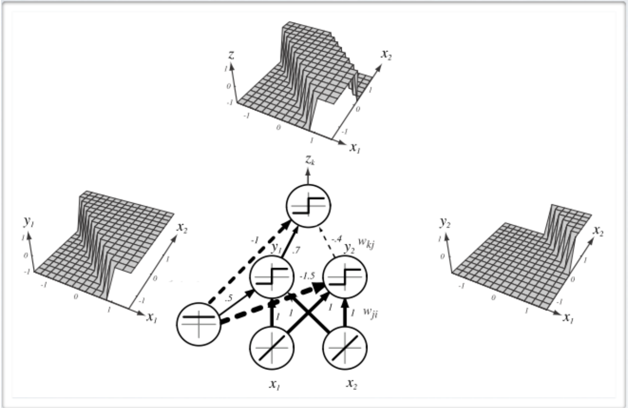
- en rajoutant des couches cachées



Tous les neurones de la couche cachée sont connectés à un seul neurone de sortie (i.e: seule une **classification binaire** est possible)

Capacité d'un réseau de neurones à une couche cachée (deux neurones)

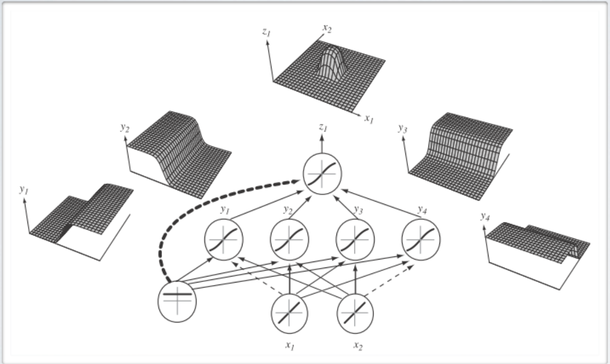
Réseaux de neurones



(from Pascal Vincent's slides)

Capacité d'un réseau de neurones à une couche cachée (plus de deux neurones)

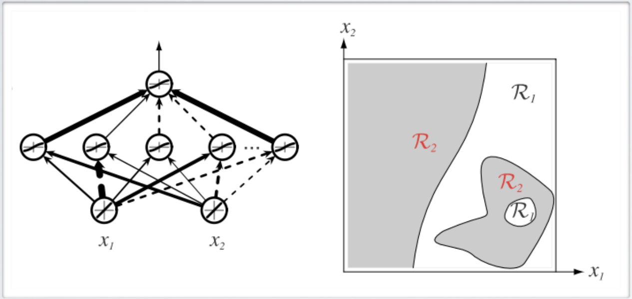
Réseaux de neurones



(from Pascal Vincent's slides)

Capacité d'un réseau de neurones à une couche cachée (plus de deux neurones)

Réseaux de neurones

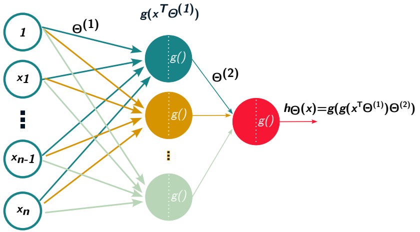


(from Pascal Vincent's slides)

Il est prouvé[1] théoriquement que si on choisit un nombre suffisant de neurones, n'importe quelle fonction peut-être approximée (approximateur universel).

Kurt Hornik, "Approximation capabilities of multilayer feedforward networks", Neural Networks, vol.4, no.2, 1991, p.251–257 (DOI 10.1016/0893-6080(91)90009-T)

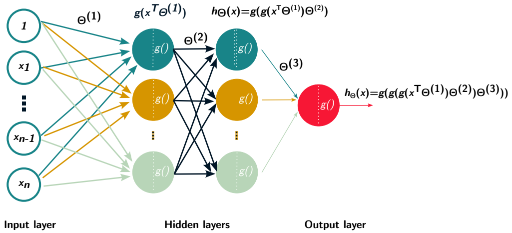
Réseau de neurones avec une couche cachée



$\mathbf{\Theta}^{(1)}_{i,j}$: poids entre le neurone i de la couche cachée $l=1$ et l'entrée j

- chaque neurone de la couche cachée a
  - une préactivation $\mathbf{z}(\mathbf{x}) = (z(\mathbf{x})_i)_{i=1,\dots}$ avec $z(\mathbf{x})_i = \sum_j \mathbf{\Theta}^{(1)}_{i,j} x_j$
  - une activation $\mathbf{a}(\mathbf{x}) = (a(\mathbf{x}))_{i=1,\dots} = (g(z(\mathbf{x})_i))_{i=1,\dots}$
- Matriciellement, c'est plus facile à écrire
  - la préactivation de la couche cachée: $\mathbf{z}^{(1)}(\mathbf{x}) = \mathbf{\Theta}^{(1)}\mathbf{x}^T$
  - l'activation de la couche cachée: $\mathbf{a}^{(1)}(\mathbf{x}) = g(\mathbf{z}^{(1)}(\mathbf{x}))$
- Pour la couche de sortie: $h_{\mathbf{\Theta}}(\mathbf{x}) = a^{(2)}(\mathbf{x}) = g(\mathbf{z}^{(2)}(\mathbf{x})) = g(\mathbf{\Theta}^{(2)}\mathbf{a}^{(1)}(\mathbf{x}))$

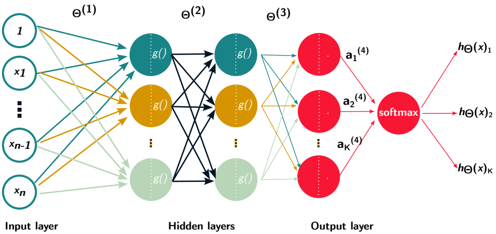
Réseau de neurones avec L couches cachées



- $l \in \{1, \ldots, L\}$ est l'indice des couches cachées du réseau
- Les neurones de la couche l sont connectés à tous les neurones de la couche (l -1)
- $\boldsymbol{\Theta}_{i,j}^{(l)}$: poids entre neurones i de la couche l et des neurones j de la couche $l-1$
- Préactivation: fonctions d'activation de la couche l :$\mathbf{z}^{(l)}(\mathbf{x}) = \boldsymbol{\Theta}^{(l)}\mathbf{a}^{(l-1)}(\mathbf{x})$
- Activation: $\mathbf{h}^{(l)}(\mathbf{x}) = g(\mathbf{z}^{(l)}(\mathbf{x}))$

Multiclass classification

Pour faire de la classification multiclasse



- Définir K neurones de sorties
- Choisir la fonction d'activation: **softmax()**

$$\text{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_k \exp(a_k)}, \ldots, \frac{\exp(a_k)}{\sum_k \exp(a_k)} \right]^T$$

  - avec $(\mathbf{a}_k^{(L+1)}) \forall k$ les préactivations de la dernière couche
  - i.e. estimer la probabilité conditionnelle $\mathbb{P}\left\{y = k|\mathbf{x}\right\}, k \in \{1, \ldots, K\}$

Apprentissage des réseaux de neurones

A partir d'un **ensemble d'apprentissage** (E), déterminer les meilleurs paramètres du réseau de neurones qu'on cherche à entraîner pour une tâche T.

- l'ensemble des paramètres est: $\boldsymbol{\Theta} = \{\boldsymbol{\Theta}^{(1)}, \ldots, \boldsymbol{\Theta}^{(L+1)}\}$
- le **modèle** du réseau de neurones multicouches est composé de **fonctions imbriquées** menant à une fonction à n'importe **quelle complexité**

$$h_{\boldsymbol{\theta}}(x) = g\left[\boldsymbol{\Theta}^{(L)} \ldots g\left[\boldsymbol{\Theta}^{(3)} g\left[\boldsymbol{\Theta}^{(2)} g\left[\boldsymbol{\Theta}^{(1)} \mathbf{x}^T\right]\right]\right] \ldots\right]$$

Trouver $\boldsymbol{\Theta}$:

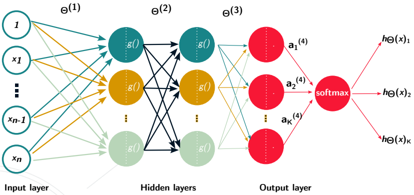
- minimisant une mesure de performance P sous la forme d'une fonction de coût $J(\boldsymbol{\Theta})$
- par descente de gradient
  - $\boldsymbol{\Theta}_{ij}^{(l)} \longleftarrow \boldsymbol{\Theta}_{ij}^{(l)} - \alpha \frac{\partial}{\partial \boldsymbol{\Theta}_{ij}^{(l)}} J\left(\boldsymbol{\Theta}\right)$
  - estimation des gradients par rapport à tous les paramètres
  - **algorithme de rétropropagation** (*backpropagation*)

Fonction de coût

- L'apprentissage va consister à minimiser une fonction dépendante des paramètres et des données d'apprentissage
  - minimisation de la fonction de coût $\arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
  - Par exemple: cross-entropie pour la tâche T de classification

$$J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=0}^{m-1} \sum_{k=1}^{K} \left[-y_k^i \log\left(h_{\boldsymbol{\theta}}\left(\mathbf{x}^i\right)\right)_k - \left(1 - y_k^i\right) \log\left(1 - h_{\boldsymbol{\theta}}\left(\mathbf{x}^i\right)_k\right)\right]$$

- La fonction de coût est changée car passage à un problème multiclasse: recodage de $y^i \in \{1, ..., K\}$ en un codage "one-hot-encoding" $y^i_{\text{one-hot}}$



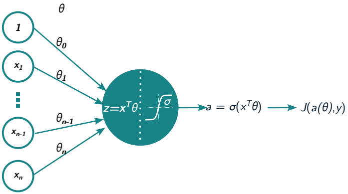
$$y^i_{\text{one-hot}} = \left(y_k^i\right)_{k\in\{1,\ldots,K\}} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ldots \text{or} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

Backpropagation pour la régression logistique (1/2)

L'apprentissage va consister en deux passes:

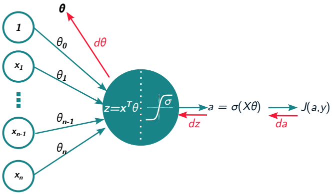
Feedforward pass

- calcul des (pré-)activations et du coût $J(\boldsymbol{\theta})$ pour un couple $(\mathbf{x}, y)$



Backward pass

- calcul des gradients par rétropropagation des gradients pour un couple $(\mathbf{x}, y)$



Objectif:

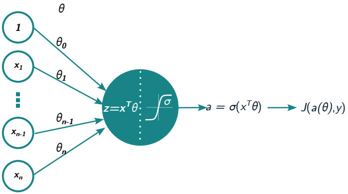
- calculer tous les $\frac{\partial J(a,y,\theta)}{\partial \theta}$ pour la descente de gradient
- avec $J(a, y, \theta) = -y \log(a) - (1-y)\log(1-a)$

Backpropagation pour la régression logistique (2/2)

L'apprentissage va consister en deux passes:

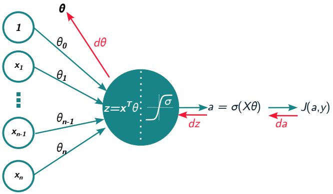
Feedforward pass

- calcul des (pré-)activations et du coût $J(\boldsymbol{\theta})$ pour un couple $(\mathbf{x}, y)$



Backward pass

- calcul des gradients par rétropropagation des gradients pour un couple $(\mathbf{x}, y)$



Utilisation de la règle de dérivation en chaîne

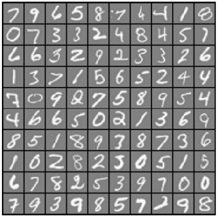
$$da = \frac{\partial J(a,y)}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a} = \frac{-(1-a)y + a(1-y)}{a(1-a)} = \frac{a-y}{a(1-a)}$$

$$dz = \frac{\partial J(a,y)}{\partial z} = \frac{\partial J(a,y)}{\partial a}\frac{\partial a}{\partial z} = da \cdot \sigma'(z) = \frac{a-y}{a(1-a)}\sigma(z)(1-\sigma(z)) = a - y$$

$$d\theta = \frac{\partial J(a,y)}{\partial \theta} = dz \cdot \frac{\partial z}{\partial \theta} = (a-y) \cdot x$$

Apprentissage des réseaux de neurones (1/1)

Exemple du tp3: on reprend le pb de reconaissance des chiffres manuscrits



- ***i***: indice des $m(= 5000)$ données d'apprentissage
- ***X***: matrice (ici à $n = 20 \cdot 20 = 400$ colonnes) des descripteurs, input variable, feature
- ***y***: variable cible, target, output variable dont les valeurs sont entre 0 et 9

On fournit les poids d'un réseau de neurones:

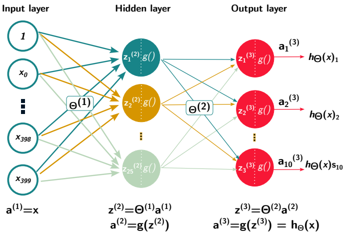
- prenant en 400 données d'entrée
- ayant une couche cachée de 25 neurones
- ayant 10 neurones de sorties
- $\boldsymbol{\Theta}^{(1)}$ la matrice 25x401 liant les 25 neurones à toutes les entrées
- $\boldsymbol{\Theta}^{(2)}$ la matrice 10x26 liant les 10 neurones de sortie à toutes les sorties des neurones de la couche cachée
- $\mathbf{y}$ doit être recodé en "one-hot"

Apprentissage des réseaux de neurones (1/2)

L'apprentissage va consister en deux passes:

Feedforward pass

- calcul des (pré-)activations et du coût $J(\boldsymbol{\theta})$

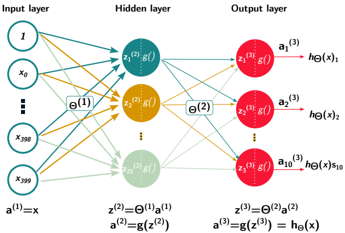


Apprentissage des réseaux de neurones (2/2)

L'apprentissage va consister en deux passes:

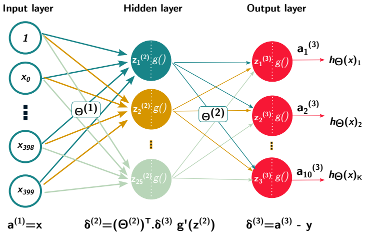
Feedforward pass

- calcul des (pré-)activations et du coût $J(\boldsymbol{\theta})$



Backward pass

- calcul des gradients par rétropropagation des gradients



Conclusions

- Un neurone
  - $=$ régression logistique
  - classification binaire et linéaire
- Réseau de neurones
  - classification multiclasse et non-linéaire
  - apprentissage d'une fonction imbriquée (la sortie d'un neurone est l'entrée d'un neurone)
  - dont la complexité dépend du nombre de couches et du nombre de neurones (approximateur universel)

Apprentissage

- à partir des données
- avec une fonction de coût J reflétant la tâche T à accomplir
- à minimiser par rétropropagation des gradients
- pour trouver les paramètres optimaux
- paramètres en grand nombre