Introduction au Deep Learning

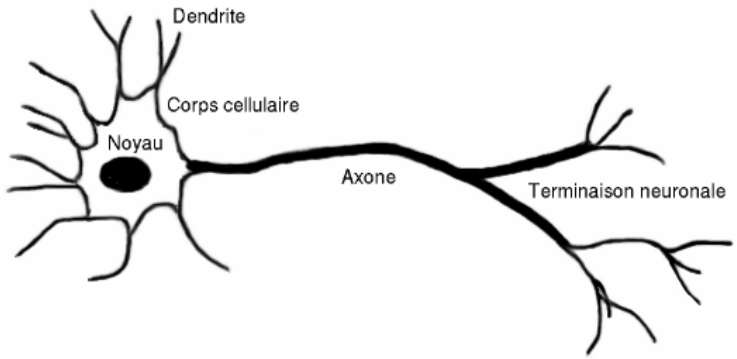
Le perceptron multicouche historique

J. Rynkiewicz

Université Paris 1

2020

Le perceptron simple



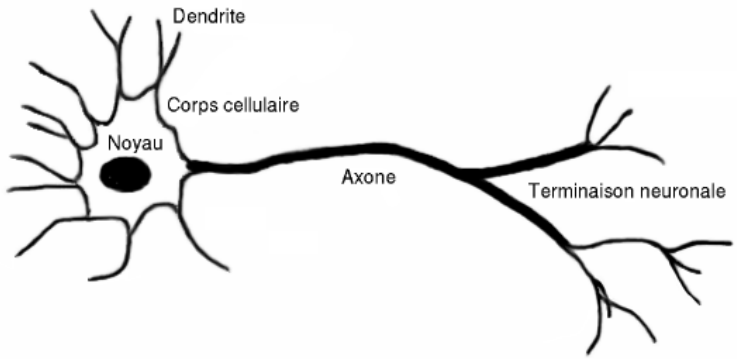
Issu de l'analogie avec la biologie, le neurone formel a été introduit par McCullogh et Pitts en 1943. Il est défini de la manière suivante :

- Des entrées réelles $x_i, i \in \{1, \cdots, d\}$
- Des poids $W_i, i \in \{0, \cdots, d\}$

Le poids W_0 est relié à une entrée constante, l'opposé de W_0 peut être vu comme une valeur seuil, au-delà de laquelle le neurone est activé.
Le neurone effectue les deux opérations suivantes en calculant :

1. Son potentiel : $W_0 + \sum_{i=1}^{d} W_i x_i$

Le perceptron simple



Issu de l'analogie avec la biologie, le neurone formel a été introduit par McCullogh et Pitts en 1943. Il est défini de la manière suivante :

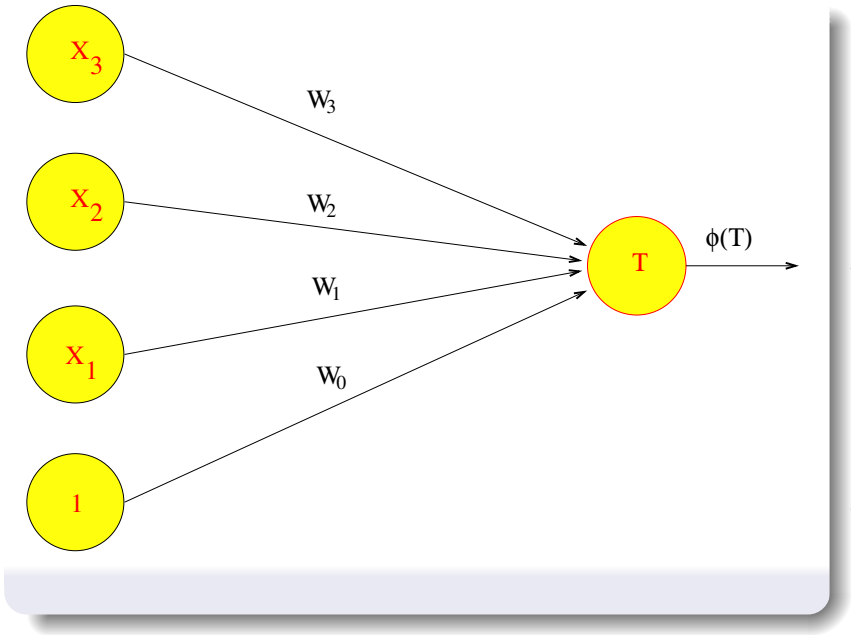
- Des entrées réelles $x_i, i \in \{1, \cdots, d\}$
- Des poids $W_i, i \in \{0, \cdots, d\}$

Le poids W_0 est relié à une entrée constante, l'opposé de W_0 peut être vu comme une valeur seuil, au-delà de laquelle le neurone est activé.
Le neurone effectue les deux opérations suivantes en calculant :

1. Son potentiel : $W_0 + \sum_{i=1}^{d} W_i x_i$
2. Son activation, grâce à une fonction d'activation ϕ : $\phi\left(W_0 + \sum_{i=1}^{d} W_i x_i\right)$

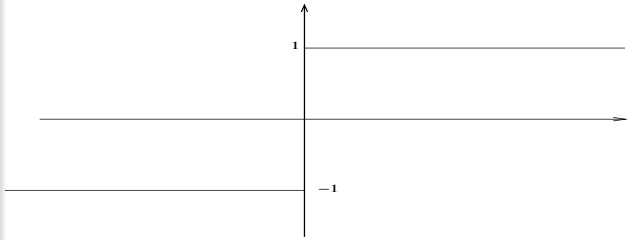
Le neurone formel

Les poids W_i représentent les poids synaptiques, $T := W_0 + \sum_{i=1}^{d} W_i x_i$ le potentiel et $F_W(x) := \phi(T)$ la sortie de l'axone. Les fonctions d'activations $\phi : \mathbb{R} \longrightarrow \mathbb{R}$ sont généralement non-linéaires, par exemple la fonction signe ou bien, la famille des fonctions sigmoïdes.



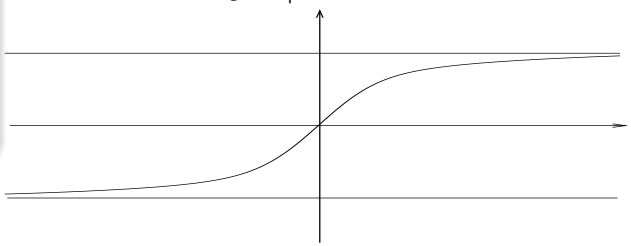
Fonction signe

$$\begin{cases} \phi(x) = 1 \text{ si } x \geq 0 \\ \phi(x) = -1 \text{ si } x < 0 \end{cases}$$



Fonction sigmoide

$$\phi(x) = c\frac{e^{kx}-1}{e^{kx}+1} + r;\ c, k > 0$$



Problème de classification linéaire

Soit deux sous ensembles $\mathcal{S}^+$ et $\mathcal{S}^-$ de $\mathbb{R}^d$. Ces deux ensembles sont séparables si et seulement si il existe $W = (W_0, \cdots, W_d) \in \mathbb{R}^{d+1}$ tel que :

$$\forall x \in \mathcal{S}^+ : W_0 + \sum_{i=1}^{d} W_i x_i > 0$$
$$\forall x \in \mathcal{S}^- : W_0 + \sum_{i=1}^{d} W_i x_i < 0$$

Un perceptron F_W avec la fonction signe pour fonction d'activation peut donc séparer ces deux ensembles :

$$F_W(x) = 1 \text{ si } x \in \mathcal{S}^+$$
$$F_W(x) = -1 \text{ si } x \in \mathcal{S}^-$$

Apprentissage du perceptron

1. On initialise les poids aléatoirement
2. Au temps t un vecteur x est présenté, soit $\varepsilon > 0$.
   - si $F_W(x) = 1$ au lieu de -1 :
   $$\forall i \in \{1, \cdots, d\} : W_i(t+1) = W_i(t) + \varepsilon x_i$$
   - si $F_W(x) = 1$ au lieu de -1 :
   $$\forall i \in \{1, \cdots, d\} : W_i(t+1) = W_i(t) - \varepsilon x_i$$
   - si $F_W(x)$ donne la bonne réponse, on ne change pas les poids du perceptron.

Problème de classification linéaire

Soit deux sous ensembles $\mathcal{S}^+$ et $\mathcal{S}^-$ de $\mathbb{R}^d$. Ces deux ensembles sont séparables si et seulement si il existe $W = (W_0, \cdots, W_d) \in \mathbb{R}^{d+1}$ tel que :

$$\forall x \in \mathcal{S}^+ : W_0 + \sum_{i=1}^{d} W_i x_i > 0$$
$$\forall x \in \mathcal{S}^- : W_0 + \sum_{i=1}^{d} W_i x_i < 0$$

Un perceptron F_W avec la fonction signe pour fonction d'activation peut donc séparer ces deux ensembles :

$$F_W(x) = 1 \text{ si } x \in \mathcal{S}^+$$
$$F_W(x) = -1 \text{ si } x \in \mathcal{S}^-$$

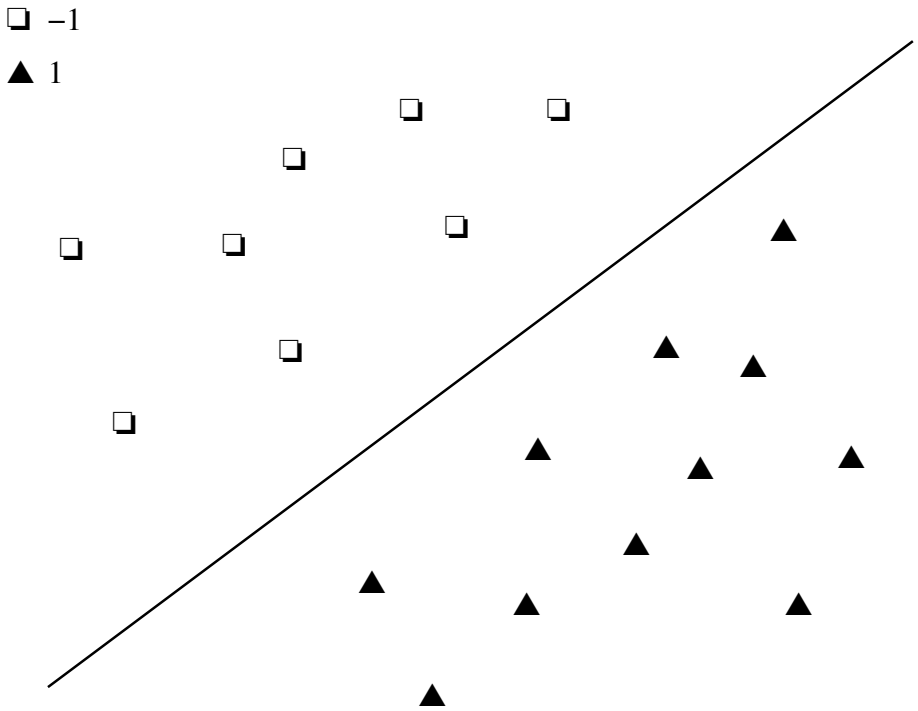
Apprentissage du perceptron

1. On initialise les poids aléatoirement
2. Au temps t un vecteur x est présenté, soit $\varepsilon > 0$.
   - si $F_W(x) = 1$ au lieu de -1 :
   $$\forall i \in \{1, \cdots, d\} : W_i(t+1) = W_i(t) + \varepsilon x_i$$
   - si $F_W(x) = 1$ au lieu de -1 :
   $$\forall i \in \{1, \cdots, d\} : W_i(t+1) = W_i(t) - \varepsilon x_i$$
   - si $F_W(x)$ donne la bonne réponse, on ne change pas les poids du perceptron.

le vecteur poids du perceptron convergera après un nombre fini d'itérations.

Problème linéairement séparable

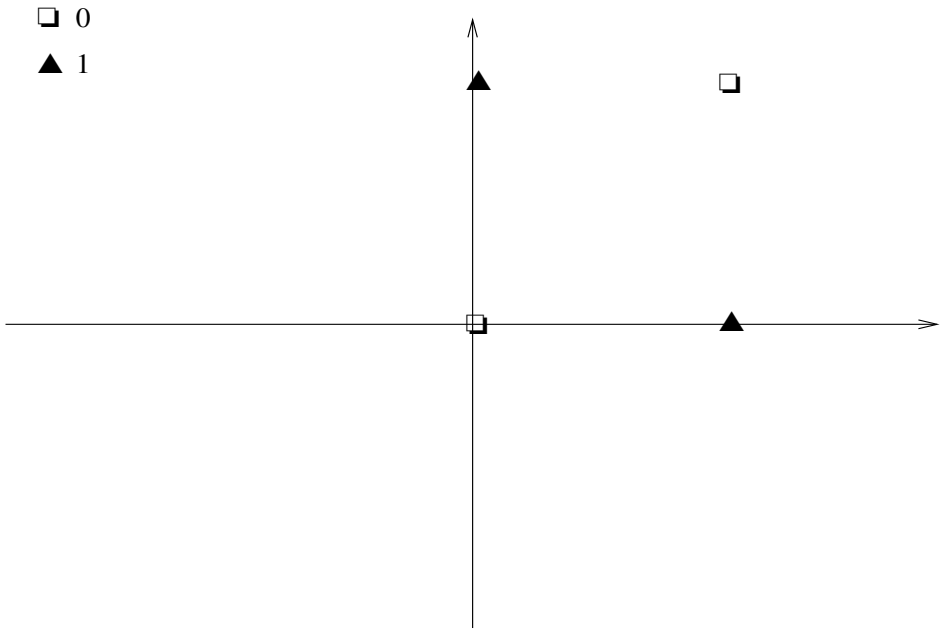
Le perceptron va trouver une des droites qui sépare les deux nuages de points à classifier.



Limitation du perceptron simple

Celui-ci ne peut pas résoudre de nombreux problèmes non-linéaires :

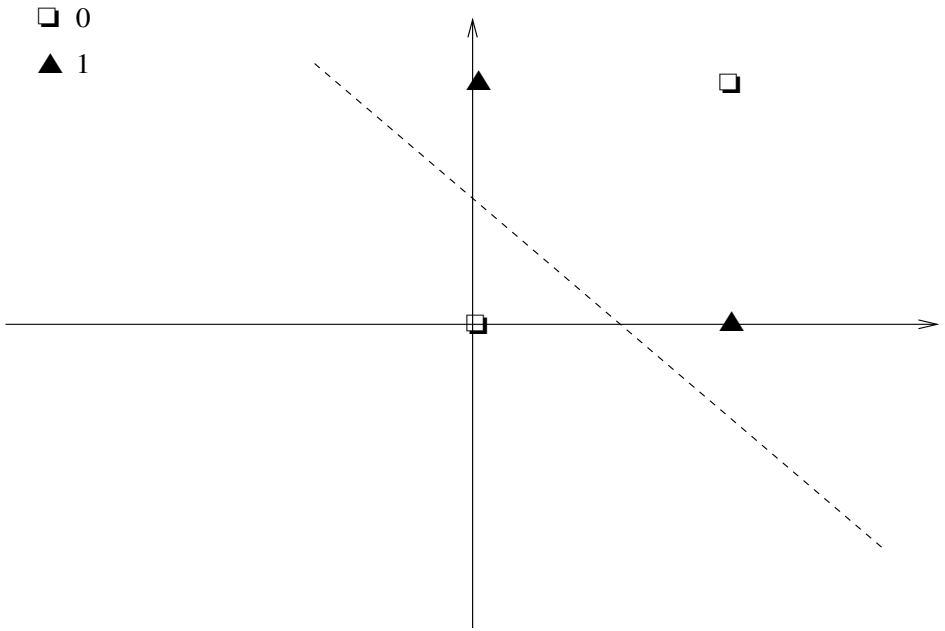
Le problème XOR



Limitation du perceptron simple

Celui-ci ne peut pas résoudre de nombreux problèmes non-linéaires :

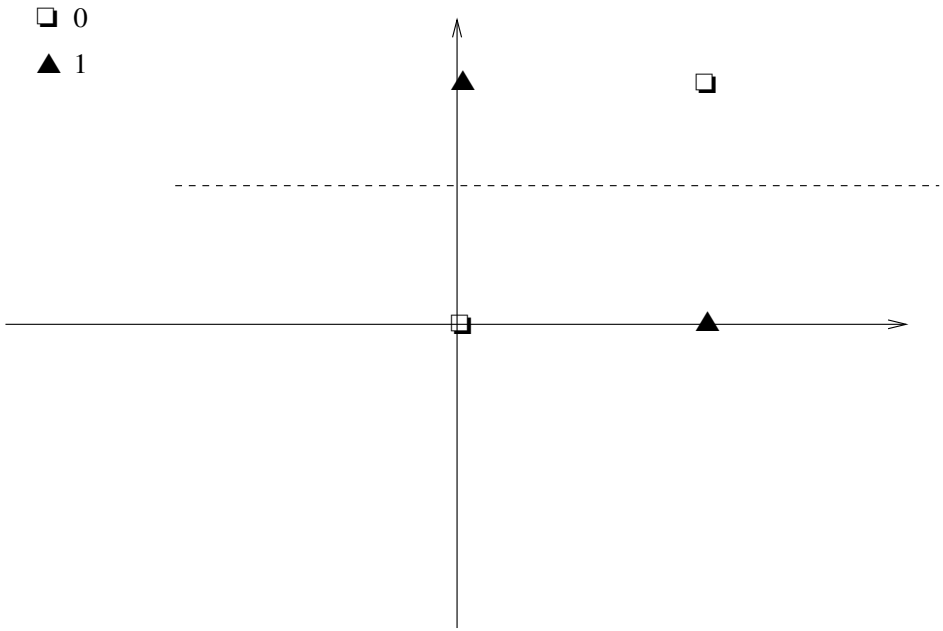
Le problème XOR



Limitation du perceptron simple

Celui-ci ne peut pas résoudre de nombreux problèmes non-linéaires :

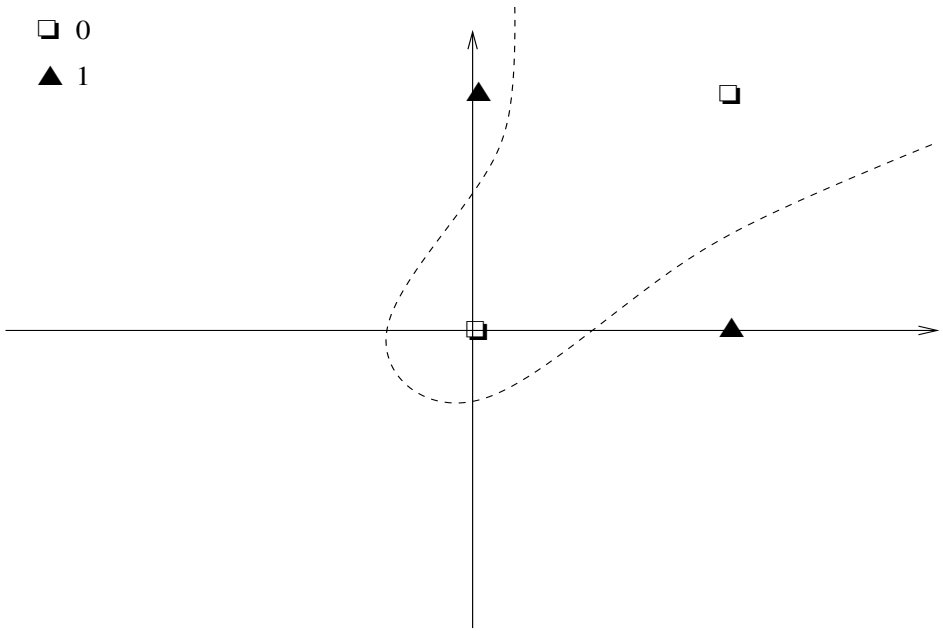
Le problème XOR



Limitation du perceptron simple

Celui-ci ne peut pas résoudre de nombreux problèmes non-linéaires :

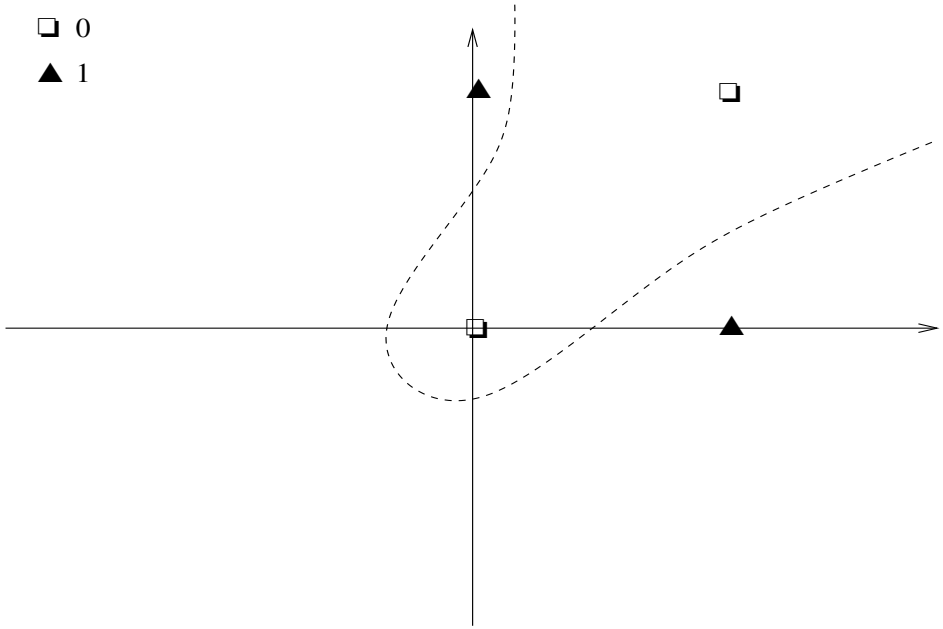
Le problème XOR



Limitation du perceptron simple

Celui-ci ne peut pas résoudre de nombreux problèmes non-linéaires :

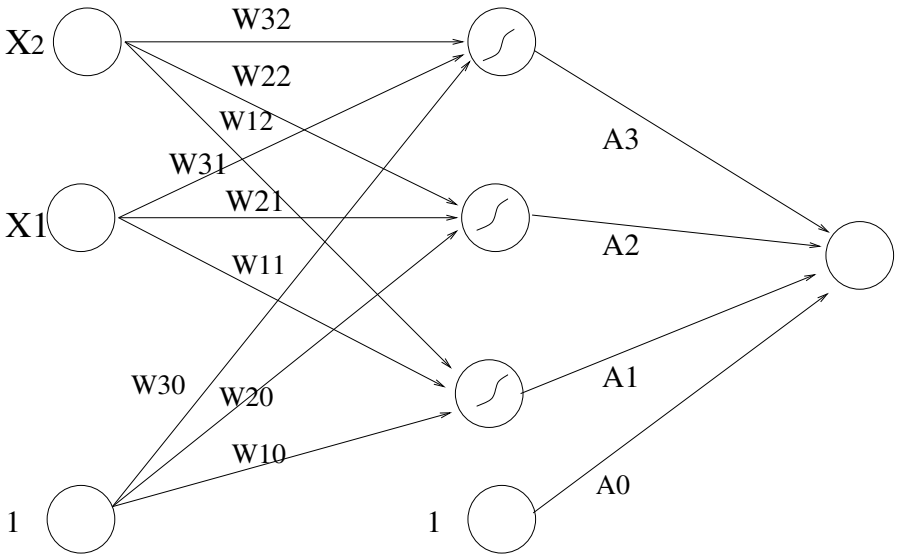
Le problème XOR



Remarque : On peut arriver à séparer cet ensemble linéairement si on travaille dans le sur-ensemble :$\{x, y, xy\}$

$$x + y - 0.5 - 2xy > 0 \text{ si } (x, y) \in \{(0, 1), (1, 0)\}$$
$$x + y - 0.5 - 2xy < 0 \text{ si } (x, y) \in \{(0, 0), (1, 1)\}$$

Le perceptron multicouche (MLP)



$$f_\theta(x_1, x_2) = a_0 + a_1\phi(w_{10} + x_1 \times w_{11} + x_2 \times w_{21}) + a_2\phi(w_{20} + x_1 \times w_{21} + x_2 \times w_{22}) + a_3\phi(w_{30} + x_1 \times w_{31} + x_2 \times w_{32})$$

où ϕ est la fonction d'activation.
Dans toute la suite
$\theta = (w_{10}, \cdots, a_N)$ représentera la vecteur paramètre du MLP.

Théorème

Soit un perceptron multicouches à une couche cachée, où ϕ est une fonction strictement croissante et bornée. Soit K un compact de $\mathbb{R}^m$.
Alors, pour toute fonction f continue à support compact ($f \in C(K)$), $f : \mathbb{R}^m \longrightarrow \mathbb{R}^s$ et pour $\epsilon > 0$, il existe un MLP avec N unités cachées et un vecteur de paramètre θ, tel que $\forall (x_1, \cdots, x_m) \in \mathbb{R}^m$

$$\|f(x_1, \cdots, x_m) - f_\theta(x_1, \cdots, x_m)\| < \epsilon$$

Estimation des poids du MLP

On dispose d'une suite d'observations $(x_t, y_t)_{1 \leq t \leq n}$.$x_t$ sont les variables explicatives et y_t les variables à expliquer. L'estimation revient à minimiser, en θ, la moyenne empirique d'une fonction de coût :
$C_n(\theta) = \frac{1}{n} \sum_{t=1}^{n} e(f_\theta(x_t), y_t) := \frac{1}{n} \sum_{t=1}^{n} e_t$

Fonctions de coûts :

1. Régression
   - Si $y_t \in \mathbb{R}$, $e_t = e(f_\theta(x_t), y_t) = (f_\theta(x_t) - y_t))^2$.
   - Si $y_t \in \mathbb{R}^s$, $e_t = e(f_\theta(x_t), y_t) = \|f_\theta(x_t) - y_t)\|^2$, la norme euclidienne.

Estimation des poids du MLP

On dispose d'une suite d'observations $(x_t, y_t)_{1\leq t\leq n}$.$x_t$ sont les variables explicatives et y_t les variables à expliquer. L'estimation revient à minimiser, en θ, la moyenne empirique d'une fonction de coût :
$C_n(\theta) = \frac{1}{n}\sum_{t=1}^{n} e(f_\theta(x_t), y_t) := \frac{1}{n}\sum_{t=1}^{n} e_t$

Fonctions de coûts :

1. Régression
   - Si $y_t \in \mathbb{R}$, $e_t = e(f_\theta(x_t), y_t) = (f_\theta(x_t) - y_t))^2$.
   - Si $y_t \in \mathbb{R}^s$, $e_t = e(f_\theta(x_t), y_t) = \|f_\theta(x_t) - y_t)\|^2$, la norme euclidienne.
2. Classification
   - Si $y_t \in (1, \cdots, K)$ représente K classes.
     Le MLP aura K sorties $\left(f_\theta^l(x_t)\right)_{1\leq l\leq K}$. On aura :
     $P_\theta(Y_t = k\,|X_t = x_t) = \frac{\exp\left(f_\theta^k(x_t)\right)}{\sum_{l=1}^{K}\exp\left(f_\theta^l(x_t)\right)}$ et la log-vraisemblance conditionnelle :

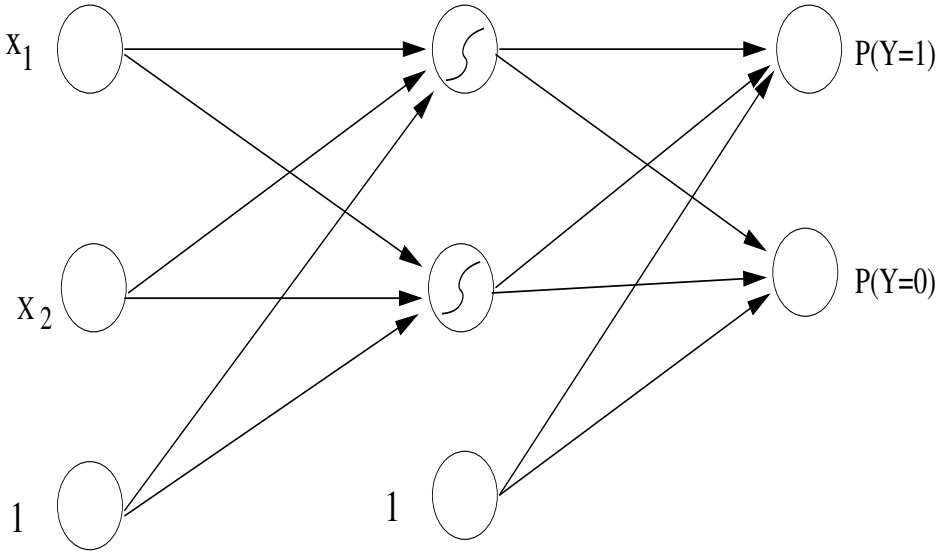
$$l_\theta(\ldots) = \sum_{t=1}^{n}\sum_{k=1}^{K} \mathbf{1}_k(y_t)\log\left(P_\theta(Y_t = k\,|x_t)\right)$$

   - D'où la fonction de coût (opposée de la log-vraisemblance) :

$$e_t = e(f_\theta(x_t), y_t) = -\sum_{k=1}^{K}\left[\mathbf{1}_k(y_t)\log\frac{\exp\left(f_\theta^k(x_t)\right)}{\sum_{l=1}^{K}\exp\left(f_\theta^l(x_t)\right)}\right]$$

Calcul du gradient

- Pour des observations fixées $\left(\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}, \cdots, \begin{pmatrix} x_n \\ y_n \end{pmatrix} \right)$, on cherche et minimiser la fonction $\theta \mapsto C_n(\theta) = \frac{1}{n} \sum_{t=1}^{n} e(f_\theta(x_t), y_t)$, où $\theta \in \mathbb{R}^d$. Il n'y a pas de solution analytique à ce problème, on calcule donc les dérivées pour $t \in \{1, \cdots, n\}$ de $\left(\frac{\partial e(f_\theta(x_t), y_t)}{\partial \theta_i} \right)_{1 \leq i \leq d}$ et on procède numériquement.
  - Pour illustrer ce calcul, on va considèrer un modèle élémentaire avec $x_t = \begin{pmatrix} x_{1t} \\ x_{2t} \end{pmatrix}$ et $y_t \in \{0, 1\}$:



Exemple du mlp élémentaire

Le mlp élémentaire avec $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $y_t \in \{0,1\}$ et $\theta = (W^1_{10}, \cdots, W^2_{22})$:

$$e(f_\theta(x), y) = \mathbf{1}_{\{0\}}(y) \ln\left(p_{1\theta}(x_1, x_2)\right) + \mathbf{1}_{\{1\}}(y) \ln\left(p_{2\theta}(x_1, x_2)\right)$$

avec

- Le première couche $z^1 = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$.
- La deuxième couche
  $z^2 = \phi^2(W^1 z^1) = \begin{pmatrix} \tanh(w^1_{11} x_1 + w^1_{12} x_2 + w^1_{10}) \\ \tanh(w^1_{21} x_1 + w^1_{22} x_2 + w^1_{20}) \end{pmatrix}$
- La troisième couche $z^3 = \begin{pmatrix} p_1(\theta) \\ p_2(\theta) \end{pmatrix} = \phi^3(W^2 z^2) =$
  $$\begin{pmatrix} \frac{\exp\left(w^2_{11} z^2_1 + w^2_{12} z^2_2 + w^2_{10}\right)}{\exp\left(w^2_{11} z^2_1 + w^2_{12} z^2_2 + w^2_{10}\right) + \exp\left(w^2_{21} z^2_1 + w^2_{22} z^2_2 + w^2_{20}\right)} \\ \frac{\exp\left(w^2_{21} z^2_1 + w^2_{22} z^2_2 + w^2_{20}\right)}{\exp\left(w^2_{11} z^2_1 + w^2_{12} z^2_2 + w^2_{10}\right) + \exp\left(w^2_{21} z^2_1 + w^2_{22} z^2_2 + w^2_{20}\right)} \end{pmatrix}$$
- La fonction de coût $e\left(\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}, y\right) = \mathbf{1}_{\{0\}}(y) \ln(p_1) + \mathbf{1}_{\{1\}}(y) \ln(p_2)$

Calcul de la dérivée

- Pour tout paramètre $\theta_i \in \{w^1_{10}, \cdots, w^2_{22}\}$, on peut utiliser la chaîne de calculs suivante :

$$\frac{\partial e(f_\theta(x),y)}{\partial \theta_i} = \frac{\partial e(f_\theta(x),y)}{\partial z^3_1}\frac{\partial z^3_1}{\partial z^2_1}\frac{\partial z^2_1}{\partial \theta_i} + \frac{\partial e(f_\theta(x),y)}{\partial z^3_1}\frac{\partial z^3_1}{\partial z^2_2}\frac{\partial z^2_2}{\partial \theta_i} + \frac{\partial e(f_\theta(x),y)}{\partial z^3_2}\frac{\partial z^3_2}{\partial z^2_1}\frac{\partial z^2_1}{\partial \theta_i} + \frac{\partial e(f_\theta(x),y)}{\partial z^3_2}\frac{\partial z^3_2}{\partial z^2_2}\frac{\partial z^2_2}{\partial \theta_i}$$

  mais ce calcul est redondant, les quantités colorées apparaissent plusieurs fois.
- L'algorithme de rétro-propagation calcule d'abord et sans redondance $\delta^L_i = \frac{\partial e(f_\theta(x),y)}{\partial z^L_i}$:

$$\delta^L_i = \frac{\partial e(f_\theta(x),y)}{\partial z^L_i} = \sum_j \frac{\partial e(f_\theta(x),y)}{\partial z^{L+1}_j}\frac{\partial z^{L+1}_j}{\partial z^L_i} = \sum_j \delta^{L+1}_j \frac{\partial z^{L+1}_j}{\partial z^L_i}$$

- Comme $z^{L+1} = \phi^{L+1}(W^L z^L)$, avec $\phi^{L+1}(u) = \left(\phi^{L+1}(u_1), \cdots, \phi^{L+1}(u_{d_{out}})\right)^T$. On a $\frac{\partial z^{L+1}_j}{\partial z^L_i} = \phi'^{L+1}(W^L z^L)_i w^L_{ji}$ avec $\phi'^{L+1}(u) = \left(\phi'^{L+1}(u_1), \cdots, \phi'^{L+1}(u_{d_{out}})\right)^T$.
- On en déduit facilement $\frac{\partial e(f_\theta(x),y)}{\partial w_{ij}} = \sum_j \frac{\partial e(f_\theta(x),y)}{\partial z^{L+1}_j}\frac{\partial z^{L+1}_j}{\partial w^L_{ij}} = \sum_j \delta^{L+1}_j \frac{\partial z^{L+1}_j}{\partial w^L_{ij}}$

Expression matricielle du calcul de la dérivée

On peut résumer matriciellement le calcul de la dérivée grâce à l'algorithme de rétropropagation pour un MLP avec N *Couches* :

- Propagation :
$$Z^{L+1} = \phi^{L+1}(W^L z^L), L = 1, \cdots, N.$$
- Rétro-propagation, en notant pour deux vecteurs u, v de $\mathbb{R}^d$, $u \times v = (u_1 v_1, \cdots, u_d v_d)^T$:
$$\delta^L = \left(W^{L\,T} \delta^{L+1}\right) \times \phi'^L(W^{L-1} z^{L-1}), L = N, \cdots, 2$$
$$\text{et } \delta^{N+1} := \left(\frac{\partial e(f_\theta(x), y)}{\partial z_i^{N+1}}\right)_{1 \le i \le d_{output}}.$$
- Calcul du gradient pour $L = 1, \cdots, N$:
$$\left(\frac{\partial e(f_\theta(x), y)}{\partial w_{ij}^L}\right)_{1 \le i \le d_{out},\, 1 \le j \le d_{in}} = \delta^{L+1} \otimes Z^L,$$
$$\left(\frac{\partial e(f_\theta(x), y)}{\partial w_{i0}^L}\right)_{1 \le i \le d_{out}} = \delta^{L+1}.$$

  où $\otimes$ est le produit extérieur de deux vecteurs : $v \in \mathbb{R}^n$, $u \in \mathbb{R}^m$,
$$v \otimes u = \begin{pmatrix} v_1 u_1 & \cdots & v_1 u_m \\ \vdots & \cdots & \vdots \\ v_n u_1 & \cdots & v_n u_m \end{pmatrix}$$

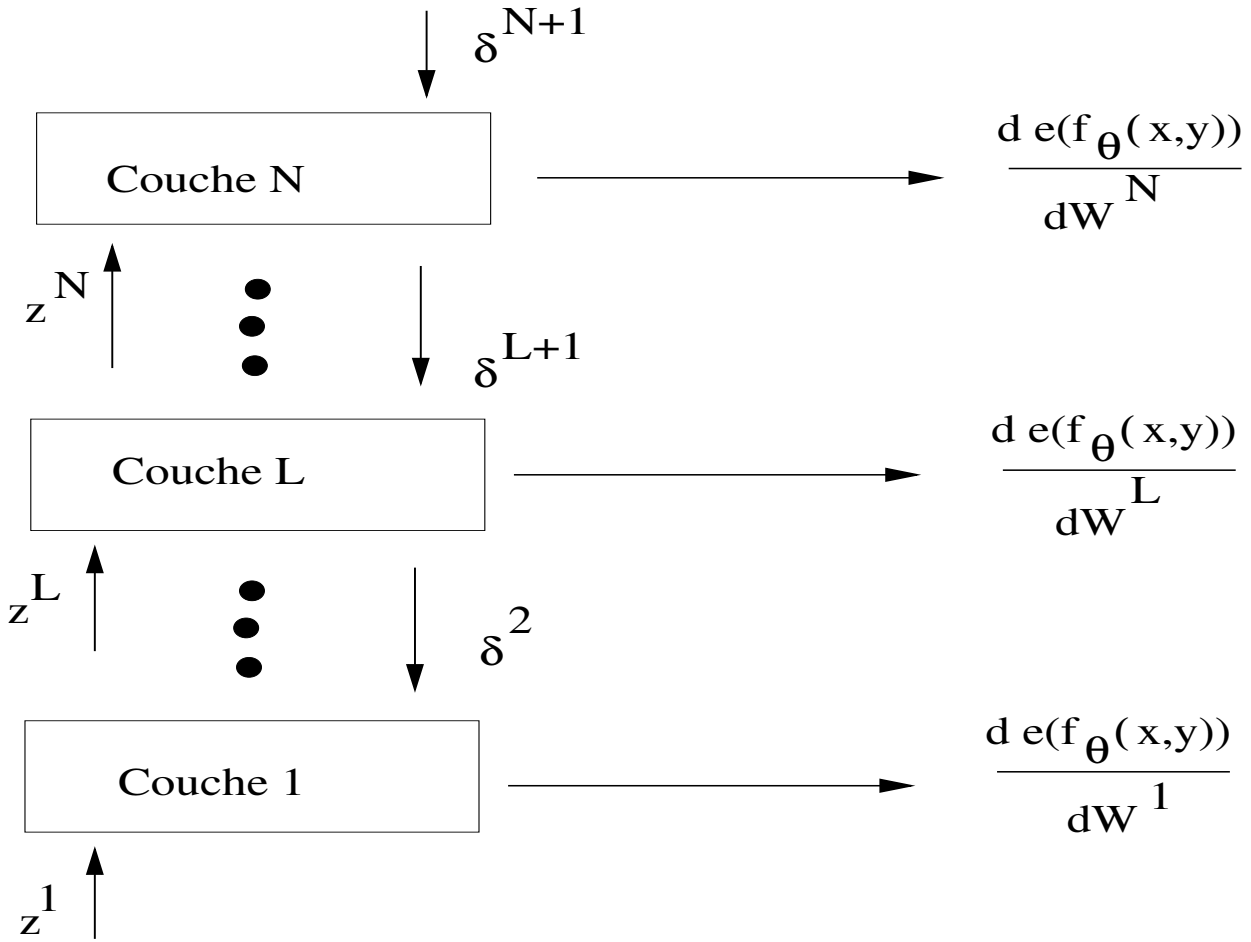
Application au mlp élémentaire

Dans notre exemple élémentaire, $N = 2$ et si on encode y sur le simplexe : $0 := \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ et $1 := \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ on aura :

- $z^1 = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$.
- $z^2 = \tanh(W^1 z^1)$.
- $z^3 = softmax(W^2 z^2)$.
- $\delta^3 = z^3 - y$.
- $\delta^2 = \left(W^{2\,T} \delta^3\right) \times \phi'^2(W^1 z^1) =$
  $\left(W^{2\,T} \delta^3\right) \times \left(z^2 - \begin{pmatrix} 1 \\ 1 \end{pmatrix}\right) \times \left(z^2 + \begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)$.
- $\left(\frac{\partial e(f_\theta(x),y)}{\partial w^2_{ij}}\right)_{1\leq i\leq 2,\, 1\leq j\leq 2} = \delta^3 \otimes z^2$, $\left(\frac{\partial e(f_\theta(x),y)}{\partial w^2_{i0}}\right)_{1\leq i\leq 2} = \delta^3$
- $\left(\frac{\partial e(f_\theta(x),y)}{\partial w^1_{ij}}\right)_{1\leq i\leq 2,\, 1\leq j\leq 2} = \delta^2 \otimes z^1$, $\left(\frac{\partial e(f_\theta(x),y)}{\partial w^1_{i0}}\right)_{1\leq i\leq 2} = \delta^2$.

Approche modulaire de la dérivation

On voit que, pour calculer la dérivée de la fonction de coût par rapport au poids, il suffit de propager les z^L de rétropropager les δ^L et d'effectuer les calculs à l'intérieur de la couche. On peut ainsi construire un réseau complexe en empilant les couches et en envoyant les résultats des calculs d'une couche aux éventuelles couches suivantes et précédentes.



Optimisation différentielle

Pour chaque couple (x_t, y_t), on peut calculer, grâce à l'algorithme de rétropropagation, les dérivés $\left(\frac{\partial e_t}{\partial \theta_i}\right)$, donc le vecteur gradient : ∇e_t. On peut utiliser ce gradient pour estimer les poids optimaux de plusieurs manières
Dans tous les cas, on choisit au hasard $\theta(0)$ (les composantes de $\theta(0)$ doivent être non nulles).

1. Par une descente de gradient stochastique (on-line).

2. Par une descente de gradient batch (off-line). Cette méthode n'est plus utilisée en Deep Learning car le nombre de paramètres est trop grand.

3. Par une descente de gradient mini-batch de taille B (entre le on-line et le off-line). C'est la méthode de prédilection du Deep Learning car cela permet de paralléliser les calculs.

Optimisation différentielle

Pour chaque couple (x_t, y_t), on peut calculer, grâce à l'algorithme de rétropropagation, les dérivés $\left(\frac{\partial e_t}{\partial \theta_i}\right)$, donc le vecteur gradient : ∇e_t. On peut utiliser ce gradient pour estimer les poids optimaux de plusieurs manières
Dans tous les cas, on choisit au hasard $\theta(0)$ (les composantes de $\theta(0)$ doivent être non nulles).

1. Par une descente de gradient stochastique (on-line).
   - $\forall t \in \{1, \cdots, n\}$ mettre à jour le vecteur paramètre :

$$\theta(t+1) = \theta(t) - \varepsilon_{t+1} \nabla e_{t+1}$$

2. Par une descente de gradient batch (off-line). Cette méthode n'est plus utilisée en Deep Learning car le nombre de paramètres est trop grand.
   - $\forall l \in \{1, \cdots, L\}$ mettre à jour le vecteur paramètre :

$$\theta(l+1) = \theta(l) - \varepsilon_{l+1} \frac{1}{n} \sum_{t=1}^{n} \nabla e_t$$

3. Par une descente de gradient mini-batch de taille B (entre le on-line et le off-line). C'est la méthode de prédilection du Deep Learning car cela permet de paralléliser les calculs.
   - $\forall l \in \{1, \cdots, L\}$ mettre à jour le vecteur paramètre :

$$\theta(l+1) = \theta(l) - \varepsilon_{l+1} \frac{1}{B} \sum_{t=l\times B+1}^{t=l\times(B+1)} \nabla e_t$$

Algorithme stochastique

$\theta(l+1) = \theta(l) - \varepsilon_l \frac{1}{B} \sum_{t=l\times B+1}^{t=l\times(B+1)} \nabla e_t$

Cet algorithme est l'algorithme standard du Deep Learning.

- Si la suite ε_t vérifie :

$$\sum_{t=1}^{\infty} \varepsilon_t = \infty \text{ et } \sum_{t=1}^{\infty} \varepsilon_t^2 < \infty$$

et que $(\theta(l))_{l\in\mathbb{N}}$ reste bornée.
L'algorithme du gradient stochastique converge presque sûrement vers un minimum local de $\theta \longmapsto C(\theta) = \mathbb{E}\left[e(f_\theta(X), Y)\right]$ quand le nombre d'observations croît vers l'infini.

- Dans la pratique, on prend ε_t reste constant sur des dizaines d'itérations et décroît lentement en restant plus grand que 10^{-6}.
- Comme on ne dispose pas d'une infinité d'observations, l'algorithme est obligé de repasser plusieurs fois sur les mêmes données.

Algorithme stochastique

$\theta(l+1) = \theta(l) - \varepsilon_l \frac{1}{B} \sum_{t=l\times B+1}^{t=l\times(B+1)} \nabla e_t$

Cet algorithme est l'algorithme standard du Deep Learning.

- Si la suite ε_t vérifie :

$$\sum_{t=1}^{\infty} \varepsilon_t = \infty \text{ et } \sum_{t=1}^{\infty} \varepsilon_t^2 < \infty$$

et que $(\theta(l))_{l\in\mathbb{N}}$ reste bornée.
L'algorithme du gradient stochastique converge presque sûrement vers un minimum local de $\theta \longmapsto C(\theta) = \mathbb{E}\left[e(f_\theta(X), Y)\right]$ quand le nombre d'observations croît vers l'infini.

- Dans la pratique, on prend ε_t reste constant sur des dizaines d'itérations et décroît lentement en restant plus grand que 10^{-6}.
- Comme on ne dispose pas d'une infinité d'observations, l'algorithme est obligé de repasser plusieurs fois sur les mêmes données.

Accélération

1. Momentum : Cette méthode conserve en mémoire la mise-à-jour à chaque étape et calcule la suivante comme une combinaison convexe du gradient pour le nouveau mini-batch et la modification précédente

$$\begin{cases} \Delta(l+1) = \alpha\Delta(l) + (1-\alpha)\frac{1}{B}\sum_{t=l\times B+1}^{t=l\times(B+1)} \nabla e_t \\ \theta(l+1) = \theta(l) - \varepsilon_{l+1}\Delta(l+1) \end{cases}$$

2. Adagrad ou Adam : Ces algorithmes améliorent la méthode du gradient stochastique en déterminant automatiquement un taux d'apprentissage pour chaque paramètre.

Algorithme batch

$\theta(t+1) = \theta(t) - \varepsilon_{t+1} \frac{1}{n} \sum_{t=1}^{n} \nabla e_t$

Si ε est suffisamment petit cet algorithme est assuré de converger vers un minimum local de

$$\theta \longmapsto \mathbb{E}\left[C(f_\theta(X), Z)\right].$$

Algorithme batch

$\theta(t+1) = \theta(t) - \varepsilon_{t+1} \frac{1}{n} \sum_{t=1}^{n} \nabla e_t$

Si ε est suffisamment petit cet algorithme est assuré de converger vers un minimum local de

$$\theta \longmapsto \mathbb{E}\left[C(f_\theta(X), Z)\right].$$

Accélération

modification de la direction de descente par une estimation de l'inverse de la Hessienne.
Les deux algorithmes les plus populaires sont le BFGS et le Levenberg-Marquardt (LM).

- Le LM ne s'applique qu'aux fonctions de coût quadratique. Il est plus performant que le BFGS si le MLP n'a qu'une sortie.
- Dans le cas multidimensionnel le BFGS est généralement plus rapide.
- les deux algorithmes fournissent une estimation de l'inverse de la matrice Hessienne.

Ces méthodes ne sont, généralement, plus adaptées au Deep Learning car le nombre de paramètres et le nombre d'observations sont très grands.

Le surapprentissage

- Les algorithmes d'optimisations stochastiques marchent bien en pratique. Ils convergent souvent vers un très bon minimum local de la fonction de coût.
- Comme ces modèles ont beaucoup de paramètres, il y a souvent plus de paramètres que d'observations, il faut faire attention au surapprentissage.
- Le surapprentissage correspond à un ajustement trop étroit ou exact à un ensemble particulier de données. Le modèle apprend "par coeur" les données d'apprentissage et ne peut pas généraliser sur de nouvelles données.
- Formellement :
  - l'algorithme minimise "très bien" $\theta \mapsto C_n(\theta) = \frac{1}{n}\sum_{t=1}^{n} e(f_\theta(x_t), y_t)$
  - mais le $\hat{\theta}$ choisi aura une erreur de généralisation $\mathbb{E}(e(f_{\hat{\theta}}(X), Y))$ grande.

Un exemple de surapprentissage

Bruit blanc

On tire un échantillon $((x_t(1), x_t(2), y_t)_{1 \leq t \leq 30}$, où $(X_1, X_2, Y) \sim \mathcal{N}(0, I_3)$ et on essaye de prévoir la variable Y en fonction du couple (X_1, X_2). Comme Y est indépendante de (X_1, X_2) et centrée la meilleur prédiction est identiquement nulle : $\mathbb{E}(Y | X_1, X_2) = 0$.

```
library(nnet)
library(rgl)
set.seed(1)
x <- matrix(rnorm(60),30,2)
y <- matrix(rnorm(30),30,1)
res.mod <- nnet(x,y,size=10,nbstart=10,maxit=1000,linout=T)
xp <- runif(10000,min=min(x[,1]),max=max(x[,1]))
yp <- runif(10000,min=min(x[,2]),max=max(x[,2]))
matp <- cbind(xp,yp)
mod.pred <- predict(res.mod,matp)
plot3d(c(-2,-2,2,2),c(-2,2,-2,2),c(20,20,-20,-20),type="n",ax
points3d(xp,yp,mod.pred,col="green")
spheres3d(x[,1],x[,2],y,radius=0.5,col="red")
```

Autre exemple de surapprentissage (1)

Toujours un bruit blanc

- Maintenant, on tire un échantillon i.i.d. de grande dimension et de grande taille : $((X_t, Y_t)_{1 \leq t \leq 100000}$, où $(X_t) \sim \mathcal{N}(0, I_{2000})$ et $Y_t \sim \mathcal{N}(0, 1)$, indépendant de X_t. on essaye de prévoir la variable Y en fonction de la variable X. Comme Y est indépendante de X et centrée la meilleur prédiction est identiquement nulle : $\mathbb{E}(Y|X) = 0$.
- On entraine cinq réseaux de neurones différents. Tous les réseaux ont deux couches cachées et des fonctions d'activations ReLU : $\phi(x) = \max(0, x)$. Ces modèles diffèrent par le nombre d'unités cachées sur chacune des couches : de 2^3 à 2^7.
- Chaque modèle est optimisé par la méthode du gradient stochastique, avec un mini batch de taille 64, un momentum de 0.9 et un pas constant de 0.01.
- Pour évaluer le surapprentissage on évalue le modèle estimé sur 100000 données de test indépendantes des données d'apprentissage.

Autre exemple de surapprentissage (2)

Toujours un bruit blanc

On obtient les résultats suivants :

TABLE – Comparaison des différentes architectures

Nb d'unités cachées	Nb de paramètres	données	erreur quadratique moyenne
2^3	16089	entrainement test	0.72 1.31
2^4	32305	entrainement test	0.47 1.67
2^5	65121	entrainement test	0.15 2.62
2^6	132289	entrainement test	0.06 2.16
2^7	272769	entrainement test	0.02 1.61

Régularisation du MLP

- Il faut un moyen de contrôler la puissance de modélisation du MLP.
- On limite sa complexité, mais il faut faire attention à le laisser capable de modéliser un problème éventuellement très compliqué.
- On cherche un compromis entre complexité (variance) et modélisation (biais).

Pendant l'apprentissage du réseau profond

- Weights decay
- Drop out
- Batch normailzation

Sélection du meilleur modèle

- Meilleure erreur sur un ensemble de validation (hold out).
- Cette méthode est quasiment optimale si on dispose de suffisamment de données dans le cadre de la classification.