Semaine IMT Grand-Est

Introduction à l'apprentissage automatique, la science de l'intelligence artificielle

Séance 5

Perceptrons multi-couches,
réseaux de neurones artificiels

Frédéric Sur

https://members.loria.fr/FSur/enseignement/IMT_GE/

Plan

Perceptron et « ou exclusif »

Critique de Minsky et Papert :

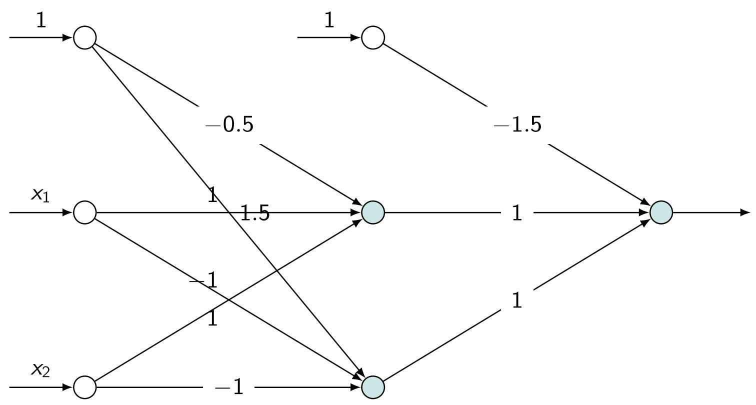
x_1		x_2	z
0	XOR	0	0
0	XOR	1	1
1	XOR	0	1
1	XOR	1	0

$\rightarrow$ impossible à implanter dans le perceptron

Perceptron à « couche cachée » et XOR

Propriété :

x_1 XOR x_2 $=$ (x_1 OR x_2) AND (NOT(x_1) OR NOT(x_2))



En chaque neurone « bleu » : activation $H(x) = \begin{cases} 0 \text{ si } x < 0 \\ 1 \text{ si } x \geq 0 \end{cases}$

Généralisation

Théorème : toute formule logique peut être convertie en une formule équivalente sous *forme normale conjonctive*.

$\rightarrow$ conjonction (**et**) de clauses disjonctives
clause disjonctive = disjonction (**ou**) de variables ou de négations de variables

$\rightarrow$ éventuellement « beaucoup » de **et**.

Remarques :

x_1 ou x_2 ou ... ou $x_d = H\left(\sum x_i - 0.5\right)$

x_1 et x_2 et ... et $x_d = H\left(\sum x_i + 0.5 - d\right)$

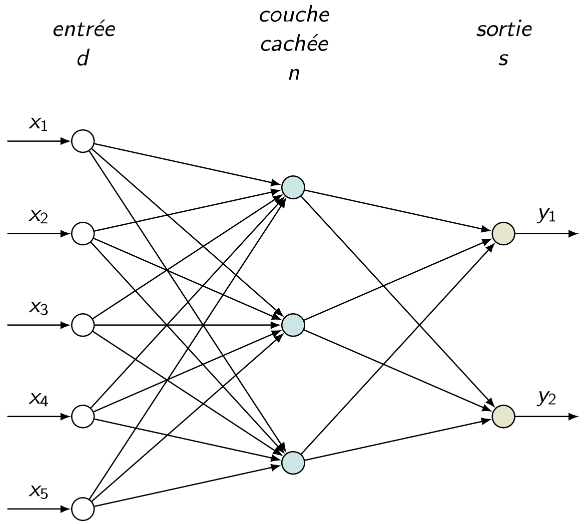
Conséquence : toute fonction booléenne peut être représentée par un réseau de perceptrons à seuil à **une** couche cachée.

fonction booléenne $f : (x_1, \dots, x_d) \mapsto \{0,1\}$, où $\forall i,\ x_i \in \{0,1\}$

$\rightarrow$ jusqu'à 2^{d-1} neurones dans couche cachée, donc $\mathcal{O}(d2^d)$ paramètres

Plan

Perceptron multicouche

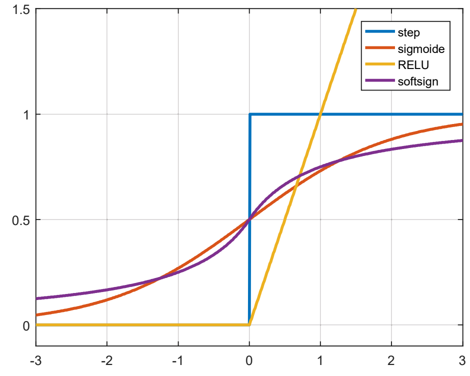


Ici, perceptron multicouche à **une couche cachée**

Nombre de paramètres : $dn + ns$

Fonction d'activation σ d'un neurone

Sortie d'un neurone : $z = \sigma(\sum_i w_i x_i)$

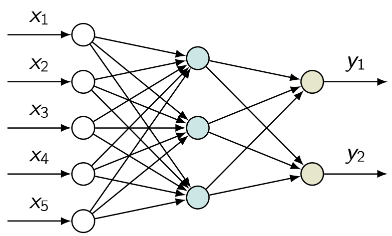


Step : cf perceptron de Rosenblatt

comme on va le voir, on est intéressé par des activations **dérivables**

Vocabulaire : perceptron multicouche (*multilayer perceptron*, MLP)
réseaux de neurones à propagation avant (*feedforward neural network*)

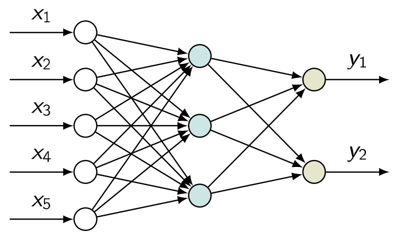
Perceptron multicouche : propagation de l'information



$X = (x_1, \ldots x_d)$ observation en entrée
$w_k = (w_{1,k}, \ldots, w_{n,k})$ poids en entrée du k-ème neurone de la couche cachée
σ : fonctions d'activation pour les neurones des couches cachées

$\rightarrow$ sortie du k-ème neurone caché : $z_k = \sigma(w_k \cdot x)$

Perceptron multicouche : sortie (régression)



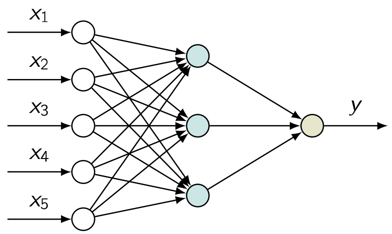
Régression : dimension de sortie = nombre de neurones de sortie

$$\forall p, \ y_p = w_p^s \cdot z$$

avec z vecteur des sorties des neurones de la dernière couche cachée

(pas de fonction d'activation sur les neurones de la couche de sortie)

Perceptron multicouche : sortie (classification bi-classe)

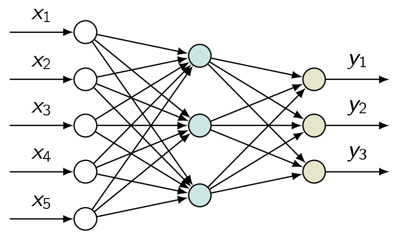


Classification bi-classe :
un seul neurone de sortie, $y = w^s \cdot z$

Règle de décision :
si $y > 0$, classification dans classe $\mathcal{C}_1$; si $y < 0$, dans $\mathcal{C}_2$

Pour obtenir une probabilité a posteriori :
activation en sortie : $\sigma(y)$ où σ sigmoïde (fonction logistique)

Perceptron multicouche : sortie (classification multiclasse)



Classification multiclasse :
une sortie par classe, puis « post-traitement » :

$$\text{SoftMax}(y_1, \ldots, y_P) = \frac{1}{\sum_{k=1}^{P} \exp(y_k)} (\exp(y_1), \ldots, \exp(y_P)) \ \in [0,1]^P$$

Règle de décision :
Observation x dans la classe avec la réponse la plus élevée
(cf. probabilité a posteriori $p(\mathcal{C}_p|x)$)

Nombre de neurones dans la couche cachée

Cas d'étude : d entrées, un neurone de sortie, $\sigma^s(x) = x$.
Si M neurones dans la couche cachée :

$$F(x) = \sum_{k=1}^{M} w_k^s \sigma(w_k \cdot x + b_k)$$

ici, $\forall k \in \{1, \dots, M\}$, $w_k \in \mathbb{R}^d$, $b_k \in \mathbb{R}$, et $w_k^s \in \mathbb{R}$

$\rightarrow$ on suppose disposer d'un algorithme d'apprentissage des w_k

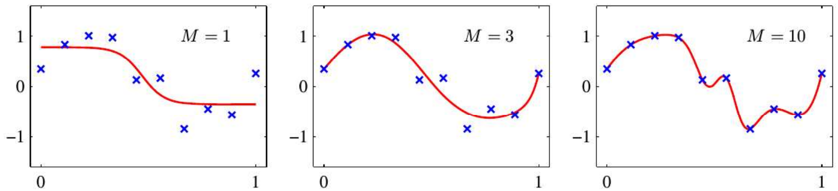


Illustration : C. Bishop, *Pattern Recognition and Machine Learning*, Springer 2006

Théorème d'approximation universelle

Théorème de Cybenko (1989) (et autres chercheurs)

Soient ϕ une fonction strictement croissante continue, et C un compact de $\mathbb{R}^n$.
Pour tout $\varepsilon > 0$ et f continue sur C, il existe $M \in \mathbb{N}$, et pour tout $i \in \{1, \dots, M\}$, $v_i, b_i \in \mathbb{R}$, $w_i \in \mathbb{R}^n$, tels que la fonction :

$$F(x) = \sum_{i=1}^{M} v_i \, \phi(w_i \cdot x + b_i)$$

vérifie : $\forall x \in C, |F(x) - f(x)| < \varepsilon$

Interprétation : toute fonction réelle continue sur un compact peut être approchée d'aussi près que l'on veut par la sortie d'un perceptron à une couche cachée (« hauteur » M, activation : $\phi = \sigma$)

Remarque : c'est un résultat d'existence ; ne dit pas comment construire le réseau (choix de M) ni comment fixer les poids pour approcher une fonction donnée avec une précision ε donnée.

Perceptron multicouche et classifieur universel

Hypothèse : $C_1, \dots, C_K$ partition mesurable de C compact

Fonction de classification multiclasse : $f(x) = i$ si $x \in \mathcal{C}_i$.

Théorème de Lusin : il existe un ensemble $D \subset C$ avec $\lambda(D) \geq (1 - \varepsilon)\lambda(C)$ tel que f coïncide sur D avec g continue.

Conclusion avec le théorème de Cybenko :
il existe un perceptron multi-couche qui approche f à ε près, sauf sur un ensemble de mesure ε.

$\rightarrow$ il y a des classifications incorrectes, mais la mesure de l'ensemble des points incorrectement classés peut être rendue aussi petite que l'on veut

Remarque : il s'agit toujours un résultat d'*existence*...

Cf. *Approximation by Superpositions of a Sigmoidal Function*,
G. Cybenko, 1989

Plan

Erreur d'un perceptron multicouche : notations

$(X_n, Y_n)_{n=1\dots N}$: base d'apprentissage

- $Y_n = (y_{n1}, \dots y_{ns}) \in \mathbb{R}^s$ si régression (s neurones de sortie)
- $Y_n = 0$ ou 1 si classification biclasse (avec σ en sortie)
- $Y_n = (0, 0, 1, 0, \dots, 0)$ si classification $s > 2$ classes

Sortie du perceptron sur $X_n = (x_{n1}, \dots, x_{nd})$ (entrée : d neurones) :
$\hat{Y}_n = (\hat{y}_{n1}, \dots, \hat{y}_{ns})$.

Mesure du coût d'erreur : $E(w) = \frac{1}{N}\sum_{n=1}^{N} E_n(w)$

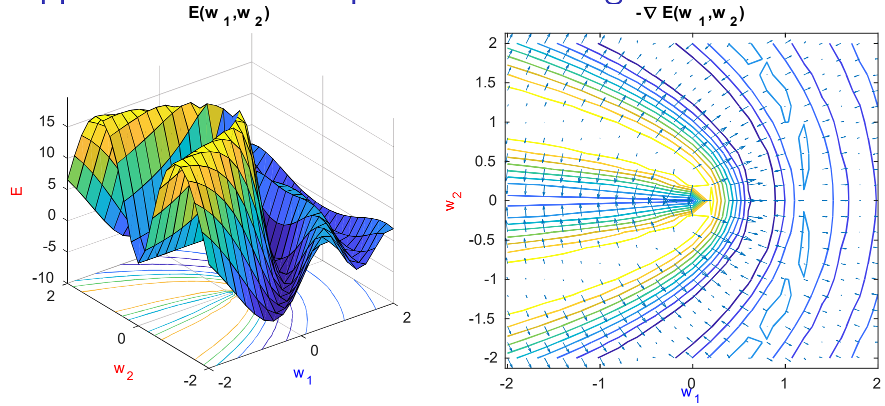
où w est l'ensemble des poids du MLP.

Exemples :

- $E_n(w) = \frac{1}{2}||Y_n - \hat{Y}_n||^2$ pour régression
- $E_n(w) = -\sum_{j=1}^{K} Y_{nj}\log(\hat{Y}_{nj})$ pour classif. $s > 2$ (entropie croisée)
- $E_n(w) = -Y_n \log(\widehat{Y}_n) - (1 - Y_n)\log(1 - \widehat{Y}_n)$ pour classif. biclasse

Objectif : choisir des poids w qui minimisent $E(w)$.

Rappel : minimisation par descente de gradient



Rappel : $\nabla E(w_1, w_2) = \begin{pmatrix} \frac{\partial E}{\partial w_1}(w_1, w_2) \\ \frac{\partial E}{\partial w_2}(w_1, w_2) \end{pmatrix}$

Algorithme de descente de gradient : $\eta > 0$,
si $\mathbf{w} = (w_1, w_2)$, on itère : $\mathbf{w}_{n+1} = \mathbf{w}_n - \eta\nabla E(\mathbf{w}_n)$

soit : $w_i \leftarrow w_i + \Delta w_i$ avec $\Delta w_i = -\eta\frac{\partial E}{\partial w_i}(w_1, w_2)$ $(i \in \{1, 2\})$

Question : choix de η ?

Rappel : dérivation des fonctions composées (*chain rule*)

$g : \mathbb{R}^p \to \mathbb{R}^q$
$f : \mathbb{R}^q \to \mathbb{R}$

$h = f \circ g : \mathbb{R}^p \to \mathbb{R}$

$x = (x_1, \dots, x_p) \in \mathbb{R}^p$
$y = g(x) = (x_1, \dots, x_q) \in \mathbb{R}^q$

$$\forall i \in [1, p],\ \frac{\partial h}{\partial x_i}(x) = \sum_{j=1}^{q} \frac{\partial f}{\partial y_j}(y)\,\frac{\partial g_j}{\partial x_i}(x)$$

Cas $q = 1$:

$$\forall i \in [1, p],\ \frac{\partial h}{\partial x_i}(x) = f'(y)\,\frac{\partial g}{\partial x_i}(x)$$

Descente de gradient et perceptron multicouche (1)

w_{ji} : poids du neurone i (couche n) vers le neurone j (couche $n+1$)

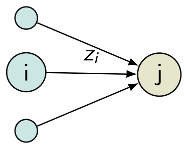
$$\Delta w_{ji} = -\eta\frac{\partial E}{\partial w_{ji}} = -\frac{\eta}{N}\sum_{n=1}^{N}\frac{\partial E_n}{\partial w_{ji}}$$

Signal arrivant au neurone j : $a_j = \sum_i w_{ji} z_i$
où z_i ($= \sigma(a_i)$) est le signal sorti par le neurone i connecté à j

Dérivation composée : (dériv. comp. avec $a_j = g(w_{ji}) \in \mathbb{R}$)

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}$$

On a $\frac{\partial a_j}{\partial w_{ji}} = z_i$, et en posant $\delta_j = \frac{\partial E_n}{\partial a_j}$:



$$\boxed{\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i}$$

Descente de gradient et perceptron multicouche (2)

Aux **neurones de sortie** :

$\delta_k = \frac{\partial E_n}{\partial a_k}$ est calculable en fonction des sorties du perceptron

Exemples :

- régression : $E_n = \frac{1}{2}||Y_n - \hat{Y}_n||^2$ où $\hat{Y}_n = (a_l)_{1 \leq l \leq s}$

donc $\frac{\partial E_n}{\partial a_k} = \frac{1}{2}\frac{\partial}{\partial a_k}\sum_l (Y_{nl} - a_l)^2$

$\rightarrow$ $\boxed{\delta_k = Y_{nk} - \hat{Y}_{nk}}$

- classification : $E_n = \sum_{l=1}^{s} Y_{nl}\log(\hat{Y}_{nl})$ où $(\hat{Y}_n) = \mathrm{SoftMax}(a_k)$

donc $\frac{\partial E_n}{\partial a_k} = \frac{\partial}{\partial a_k}\sum_l Y_{nl}\left(a_l - \log(\sum_m \exp(a_m))\right)$

$= Y_{nk} - \frac{\exp(a_k)}{\sum_m \exp(y_m)}\sum_l Y_{nl}$, or $\sum_l Y_{nl} = 1$

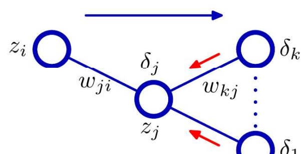
$\rightarrow$ $\boxed{\delta_k = Y_{nk} - \hat{Y}_{nk}}$ (!?)

Descente de gradient et perceptron multicouche (3)

Aux **neurones cachés** : (dériv. comp. avec $\underbrace{(a_k)_k}_{\text{couche } n+1} = \underbrace{g(a_j)}_{\text{couche } n}$)

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$$

pour k parcourant *les neurones vers lesquels j envoie des informations*



De plus : $\frac{\partial a_k}{\partial a_j} = \frac{\partial}{\partial a_j}\sum_i w_{ki}\sigma(a_i) = w_{kj}\sigma'(a_j)$

donc : $\boxed{\delta_j = \sigma'(a_j)\sum_k w_{kj}\delta_k}$

Conclusion : on peut calculer tous les δ sur le réseau par *rétropropagation de l'erreur*

Apprentissage et rétropropagation des erreurs

Algorithme – on itère jusqu'à satisfaction d'un critère d'arrêt :

1. pour chaque observation x_n en entrée :
   1. propagation pour calculer les a_i/z_i en chaque neurone et les valeurs de sortie du PMC
   2. calcul des δ_k en sortie par comparaison à la valeur attendue
   3. rétropropagation des δ : calcul des δ_j de chaque unité cachée
   4. calcul des $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$
2. calcul de $\Delta w_{ji} = -\frac{\eta}{N}\sum_{n=1}^{N}\frac{\partial E_n}{\partial w_{ji}}$ et mise à jour des poids $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$

En pratique (cf. occupation mémoire, accélération convergence) :

- version *online* / *algorithme du gradient stochastique* : mise à jour des poids pour *chaque* observation de la base d'apprentissage, $\Delta w_{ji} = -\frac{\eta}{N}\frac{\partial E_n}{\partial w_{ji}}$
- version *mini-batch* / *par lot* : mise à jour des poids après parcours d'un sous-ensemble d'observations : $\Delta w_{ji} = -\frac{\eta}{N}\sum_{n \in B_m}\frac{\partial E_n}{\partial w_{ji}}$ t.q. $\cup_m B_m = \{1, \ldots, N\}$

Remarques sur la rétropropagation

- Rétropropagation = astuce pour calculer ∇E, c'est différent de l'apprentissage proprement dit (qui consiste en la minimisation du coût d'erreur E, grâce à ∇E)
- Importance de considérer des fonctions d'activation dérivables
  Remarque : pour la sigmoïde : $\sigma'(t) = \sigma(t)(1 - \sigma(t)) \leq 1/4$
  et : $\delta_j = \sigma'(a_j)\sum_k w_{kj}\delta_k$
  $\rightarrow$ donc risque de *vanishing gradient* si plusieurs couches cachées
  cela motive l'utilisation de ReLU (*rectified linear unit*)
- Les neurones peuvent avoir des fonctions d'activation différentes
- Adaptable à d'autres types d'architectures de réseaux
- Relativement indépendante de l'algorithme de recherche de l'optimum du coût d'erreur
  $\rightarrow$ il y a plus « efficace » que la descente de gradient à pas fixe,
  cf documentation `scikit-learn`

Problème du surapprentissage (*overfitting*)

On cherche à minimiser l'erreur empirique (cf séance 2)...

Erreur en fonction du nombre d'*epochs* :

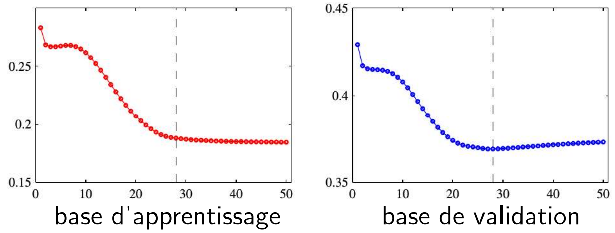


Illustration : C. Bishop, *Pattern Recognition and Machine Learning*, Springer 2006

→ *early stopping* : on arrête l'apprentissage quand l'erreur sur un ensemble de validation commence à augmenter

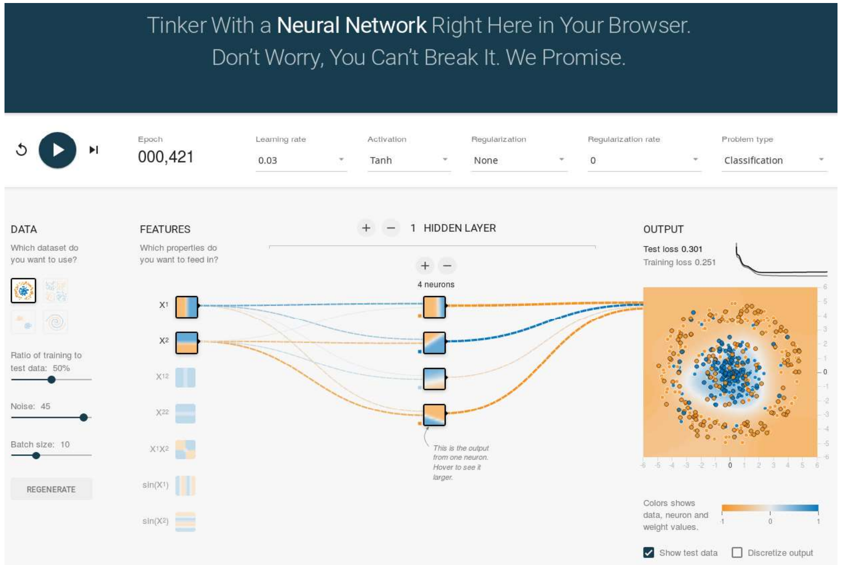
→ régularisation : on contrôle la ≪ complexité ≫ du réseau en cherchant plutôt à minimiser (avec hyperparamètre $\lambda > 0$) :

$$E(w) + \frac{\lambda}{2}||w||_2^2$$

(remarque : cela passe bien dans l'algorithme de rétropropagation)

Expériences en TP

https://playground.tensorflow.org



Plan

Conclusion

- le perceptron multicouche permet d'aller au-delà des classifieurs linéaires (perceptron, régression logistique, etc.)
  → classifieur et approximateur universels
- la rétropropagation des erreurs permet la mise en œuvre de l'apprentissage des poids du réseau par minimisation de l'erreur empirique
- augmenter le nombre de couches permet de réduire la complexité du modèle (nombre de paramètres)
- révolution ∼10ans : *deep learning*, *convolutional neural networks*
  → progrès dans la compréhension des propriétés théoriques
  → convolution = filtre de base du traitement du signal
  → disponibilité de grandes bases de données
  → puissance de calcul et architectures matérielles adaptées (GPU)