Perceptron Multicouche

Introduction à l'apprentissage automatique
Master Sciences Cognitives
Aix Marseille Université

Alexis Nasr

Motivations

- Le perceptron multicouche est le premier réseau profond abordé dans ce cours.
- Il constitue le modèle de base à partir duquel des modèles plus complexes seront construits.

Objectifs

- Comprendre le rôle que joue les fonctions d'activation dans un réseau de neurones.
- Comprendre le problème de la disparition du gradient.
- Comprendre une nouvelle fonction d'erreur : l'entropie croisée.
- Comprendre la différence entre un encodage one-hot et un encodage dense.

Plan

Limite des modèles linéaires

- Le perceptron permet de trouver une solution “parfaite” aux problèmes de classification binaires lorsque les données sont **linéairement séparables**.
- Souvent, les données n’ont pas cette propriété !

L'exemple historique du XOR

- La fonction XOR est l'opération booléenne qui correspond au non exclusif :

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

- On cherche les poids w_1 et w_2 tels que l'équation :

$$w_1 x_1 + w_2 x_2 = y$$

  soit vérifiée pour tous les exemples.
- On ne s'intéresse pas à la généralisation, on souhaite juste une erreur nulle sur l'ensemble d'apprentissage.

L'exemple historique du XOR

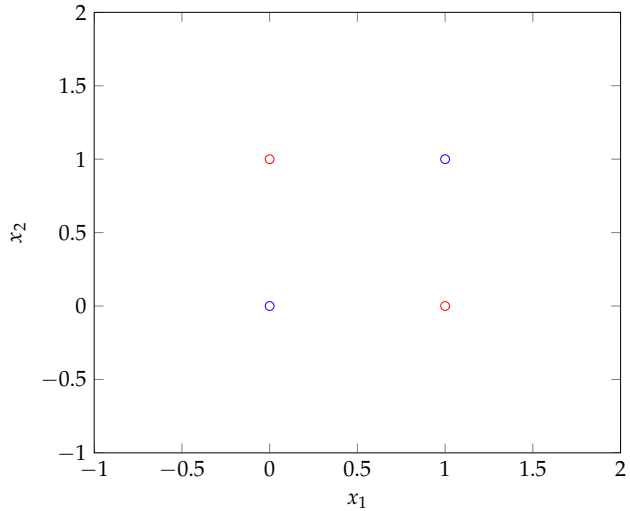
- Cela revient à trouver une solution au système d'équation suivant :

$$
\begin{aligned}
w_1 \times 0 + w_2 \times 0 &= 0 \\
w_1 \times 0 + w_2 \times 1 &= 1 \\
w_1 \times 1 + w_2 \times 0 &= 1 \\
w_1 \times 1 + w_2 \times 1 &= 0
\end{aligned}
$$

- qui n'en admet pas !

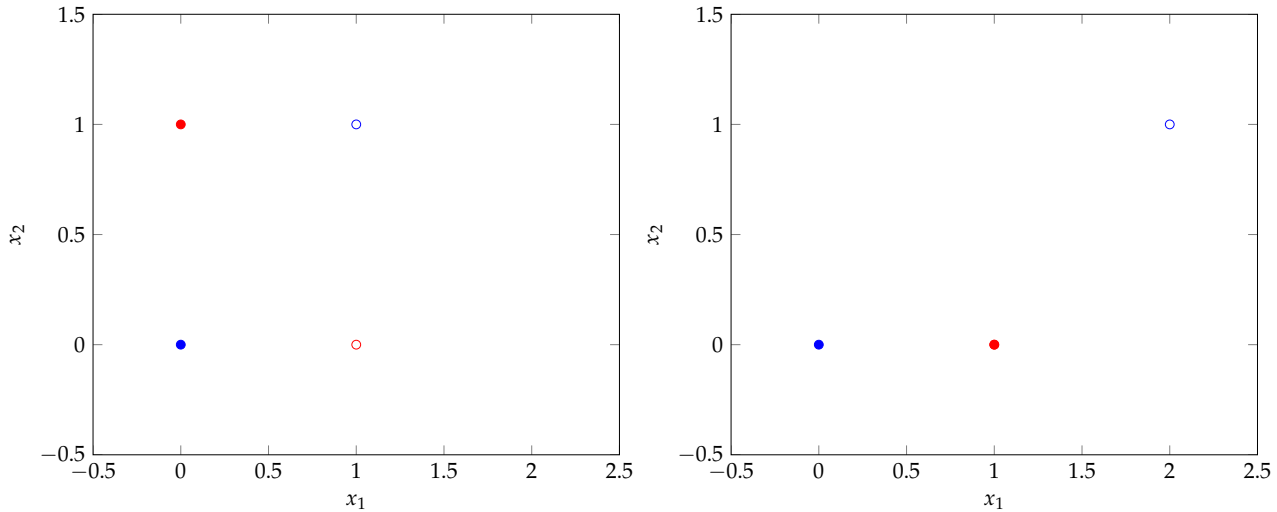
$$
\begin{aligned}
w_2 &= 1 \\
w_1 &= 1 \\
w_1 + w_2 &= 0
\end{aligned}
$$

Interprétation géométrique



- On ne peut séparer les deux classes à l'aide d'une droite.
- Mais on pourrait appliquer une **transformation** ϕ aux données de manière à rendre les données linéairement séparables.
- L'application de la transformation permet d'aboutir à une nouvelle **représentation** des données.
- ϕ ne peut pas être linéaire, cela ne permettrait pas de résoudre le problème.

Transformation des données



- Un exemple de transformation :

$$\phi([0,0]^T) = [0,0]^T$$
$$\phi([0,1]^T) = [1,0]^T$$
$$\phi([1,0]^T) = [1,0]^T$$
$$\phi([1,1]^T) = [2,1]^T$$

- Les données sont maintenant linéairement séparables.
- Comment trouver automatiquement une transformation ϕ ?

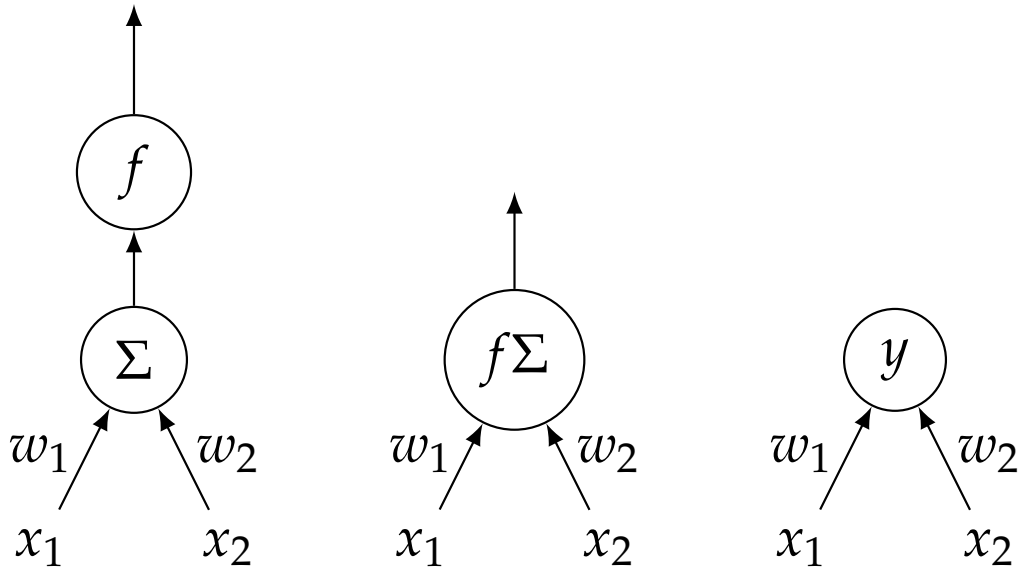
Le perceptron multicouche

- Un perceptron multicouches, appelé *Multi-Layered Perceptron*, noté MLP, est constitué d'unités de calcul élémentaires appelés **neurones**.
- Un neurone possède des entrées, qui sont des variables à valeur réelles, notées $x_1, \ldots x_n$, et une sortie, notée y.
- Chaque entrée est associée à un poids, noté $\mathbf{w}_i$.
- Le calcul effectué par un neurone consiste à multiplier la valeur de chaque entrée $\mathbf{x}_i$ par son poids $\mathbf{w}_i$ et d'en faire la somme.
- Le résultat de cette somme est additionné à une constante b (appelé le **biais**).
- Ce résultat constitue lui-même l'entrée d'une fonction non linéaire f, appelée **fonction d'activation**.
- Le résultat de ce calcul constitue la sortie $\hat{\mathbf{y}}$ du neurone :

$$\hat{\mathbf{y}} = f(\Sigma_{i=1}^{n} \mathbf{w}_i \mathbf{x}_i + b)$$

- Il est appelé **activation** du neurone.

Représentation graphique

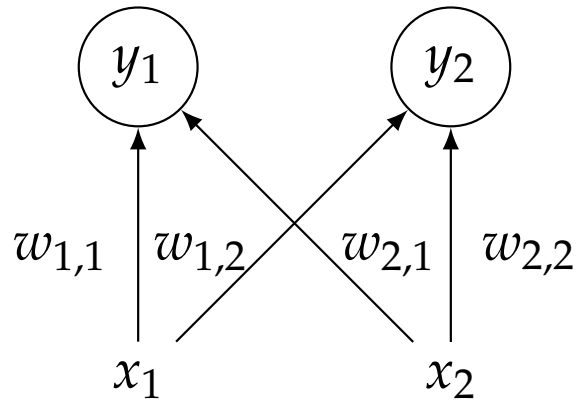


- Les neurones sont parfois représentés graphiquement.
- La représentation de gauche détaille les différentes étapes du calcul.
- Celle du milieu condense la somme et la fonction d'activation en un seul nœud.
- Il est souvent pratique de définir une variable qui prend pour valeur l'activation d'un neurone, comme dans la figure de droite.

Interconnexions

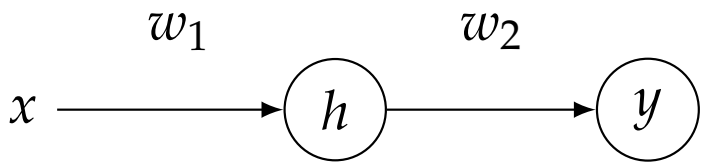
- Les neurones peuvent être inter-connectés pour constituer un **réseau de neurones** (RN).
- Deux modes de connexion sont possibles :
  - Des neurones peuvent partager des entrées.
  - la sortie d'un neurone peut constituer l'entrée d'un autre neurone.

Partage d'entrée



$$y_1 = f(w_{1,1}x_1 + w_{2,1}x_2 + b_1)$$
$$y_2 = f(w_{1,2}x_1 + w_{2,2}x_2 + b_2)$$

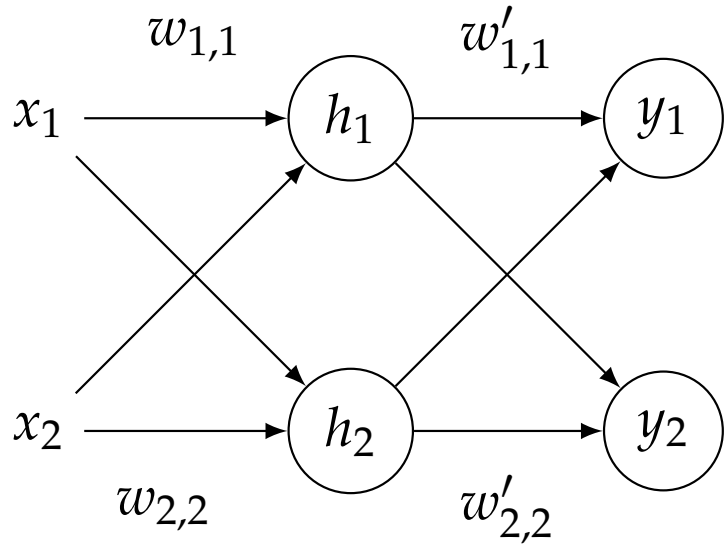
Composition



- La sortie d'un neurone constitue l'entrée d'un autre.

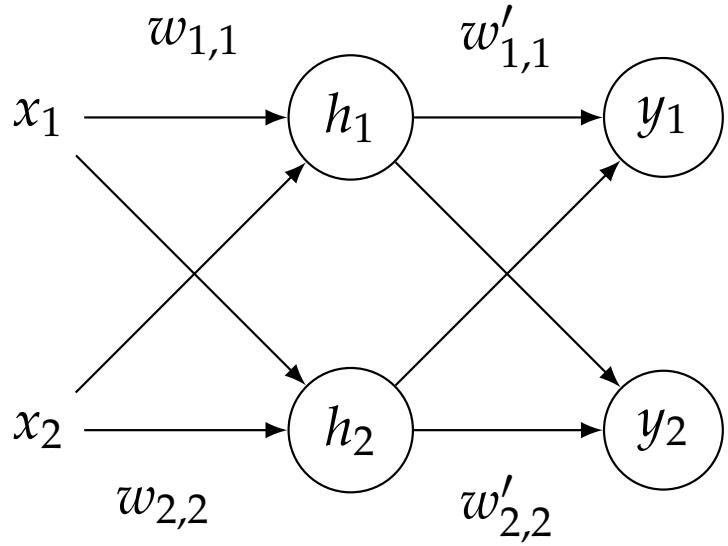
$$y = f_2(w_2 \times h + b_2)$$
$$h = f_1(w_1 x + b_1)$$

La notion de couche



- Dans un RN, les neurones sont généralement organisés en **couches**
- les sorties des neurones d'une couche constituent les entrés des neurones de la couche suivante.
- Lorsque chaque neurone d'une couche est connecté à tous les neurones de la couche suivante, le réseau est dit **entièrement connecté**.

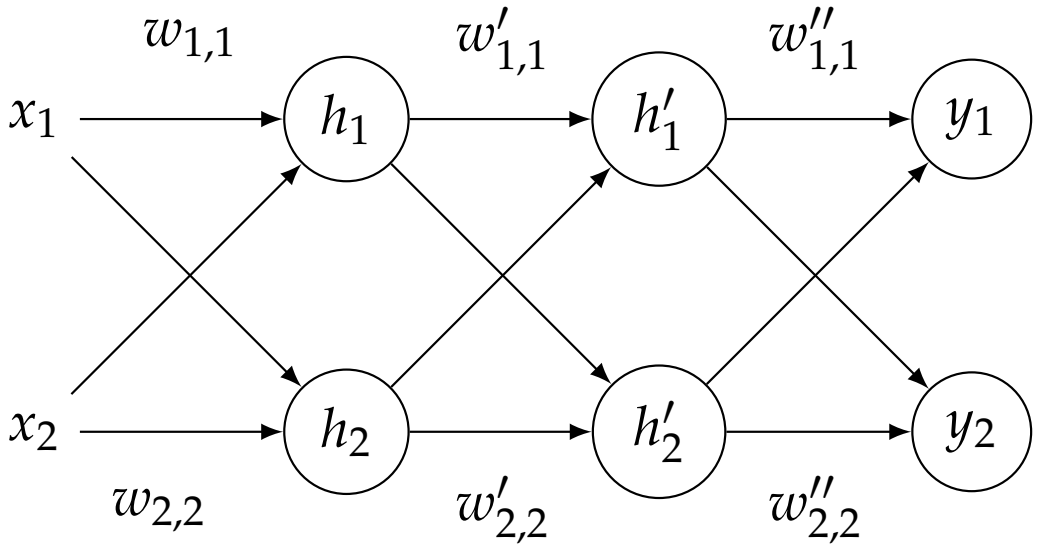
La notion de couche



- Calculs réalisés par le réseau :

$$
\begin{aligned}
h_1 &= f_1(w_{1,1} \times x_1 + w_{2,1} \times x_2 + b_1) \\
h_2 &= f_1(w_{1,2} \times x_1 + w_{2,2} \times x_2 + b_2) \\
y_1 &= f_2(w'_{1,1} \times h_1 + w'_{2,1} \times h_2 + b'_1) \\
y_2 &= f_2(w'_{1,2} \times h_1 + w'_{2,2} \times h_2 + b'_2)
\end{aligned}
$$

La notion de couche



- x_1 et x_2 constituent la **couche d'entrée** (ce ne sont pas des neurones).
- y_1 et y_2 constituent la **couche de sortie**.
- h_1 et h_2 constituent une première **couche cachée**.
- h'_1 et h'_2 constituent une deuxième couche cachée.
- le nombre de couche constitue la **profondeur** du réseau.

Ecriture matricielle

- Ces calculs peuvent aussi être formulés sous forme matricielle.
- On représente les entrées x_1 et x_2 sous la forme d'un vecteur $\mathbf{x}$
- Les poids des arcs reliant les neurones de la couche d'entrée sous la forme d'une matrice $\mathbf{W}$
- Les poids reliant les neurones de la première couche à ceux de la seconde sous la forme d'une matrice $\mathbf{W}'$

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{bmatrix} \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \mathbf{W}' = \begin{bmatrix} w'_{1,1} & w'_{2,1} \\ w'_{1,2} & w'_{2,2} \end{bmatrix} \mathbf{b}' = \begin{bmatrix} b'_1 \\ b'_2 \end{bmatrix}$$

- Le calcul précédent se réecrit alors sous la forme suivante :

$$\mathbf{h} = f_1(\mathbf{W}\mathbf{x} + \mathbf{b})$$
$$\mathbf{y} = f_2(\mathbf{W}'\mathbf{h} + \mathbf{b}')$$

Prise en compte du biais

- Il est possible d'ajouter une dimension au vecteur $\mathbf{x}$ et une colonne pour prendre en compte les biais $\mathbf{b}$ et $\mathbf{b}'$.
  On obtient alors

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{2,1} & b_1 \\ w_{1,2} & w_{2,2} & b_2 \end{bmatrix} \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} \mathbf{W}' = \begin{bmatrix} w'_{1,1} & w'_{2,1} & b'_1 \\ w'_{1,2} & w'_{2,2} & b'_2 \end{bmatrix}$$

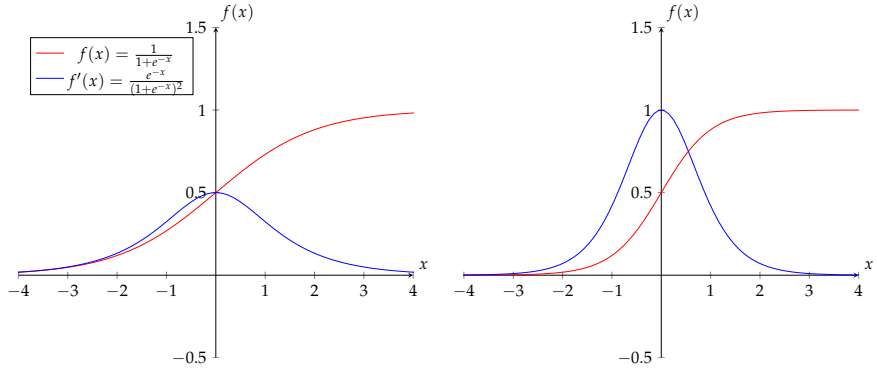
  Ce qui permet de réecrire les équations plus simplement :

$$\mathbf{h} = f_1(\mathbf{W}\mathbf{x})$$
$$\mathbf{y} = f_2(\mathbf{W}'\mathbf{h})$$

Fonctions d'activation

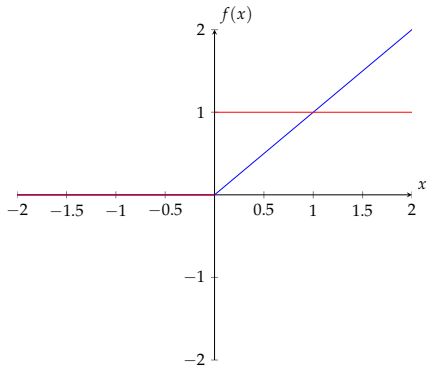
- La fonction d'activation d'un réseau permet d'introduire de la non-linéarité dans la fonction $\mathbf{y} = f(\mathbf{x})$.
- C'est elle qui permet de transformer les données afin qu'elles soient linéairement séparables.
- Plusieurs fonctions d'activation sont possibles, parmi lesquelles :
  - La fonction sigmoïde
  - La fonction tangente hyperbolique
  - La fonction ReLU
  - La fonction softplus

Fonctions sigmoïde et tangente hyperbolique



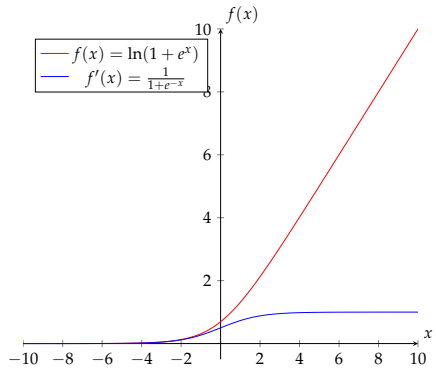
- Beaucoup utilisées au début des réseaux de neurones.
- Elles ont l'avantage d'être dérivables, ce qui permet de calculer le gradient de la fonction d'erreur.
- Elles ont l'inconvénient de posséder des dérivées qui prennent des valeurs très faibles sur la majeure partie de leur domaine.
- Lors du calcul du gradient de la fonction d'erreur d'un réseau profond, on réalise des multiplications en chaîne de la fonction dérivée.
- Le gradient a tendance à prendre des valeurs de plus en plus petites, phénomène connu sous le nom de **disparition du gradient**.
- Lors de l'apprentissage, les couches inférieures (proches de l'entrée) sont modifiées très lentements, voire pas du tout.

La fonction ReLU



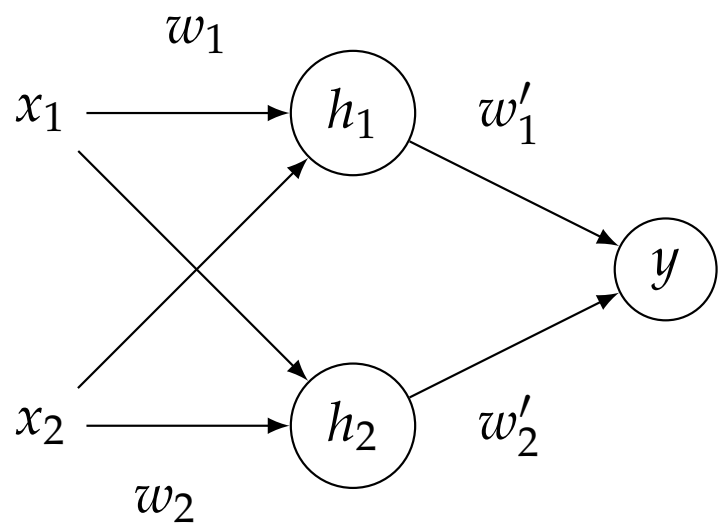
- Elle échappe au problème de la disparition du gradient.
- Mais elle n'est pas dérivable en 0 !
- Dans la pratique cela n'est pas très problématique car la valeur 0 est très rarement atteinte
- Mais ce n'est pas très joli !

La fonction SoftPlus



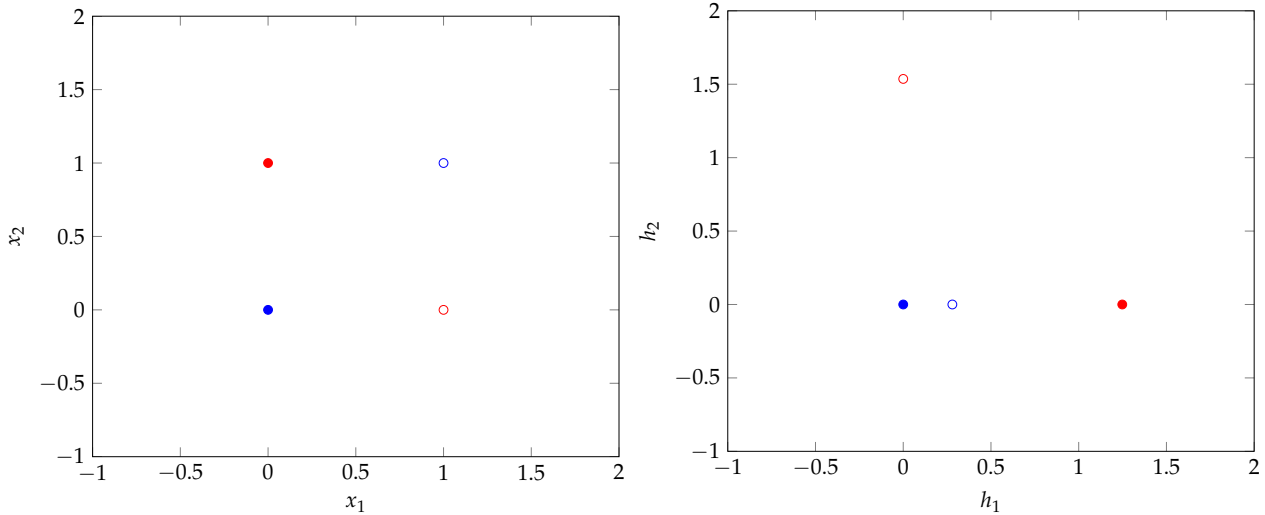
- Version dérivable de ReLU
- La dérivée de SoftPlus est la fonction sigmoïde
- Peu utilisée en pratique

Retour sur le problème du XOR



- Il est possible de résoudre le problème du XOR à l'aide d'un réseau simple
- La couche cachée (h_1, h_2) permet de trouver une transformation des données
- La couche de sortie (y) effectue une combinaison linéaire

Transformation trouvée par le MLP



$$h_1 = ReLU(1.2484406x_1 - 1.5363349x_2 - 0.00033336)$$
$$h_2 = ReLU(-1.247532x_1 + 1.5365471x_2 - 0.00017454)$$

x_1	x_2	h_1	h_2	y
0	0	0	0	0.299
0	1	0	1,53	0.717
1	0	1.24	0	0.709
1	1	0.28	0	0.299

Implémentation du réseau avec keras

```
import numpy as np
from keras.models import Sequential
from keras.layers.core import Activation, Dense

training_data = np.array([[0,0],[0,1],[1,0],[1,1]], "float32")
target_data = np.array([[0],[1],[1],[0]], "float32")

model = Sequential()
model.add(Dense(2, input_dim=2, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='mean_squared_error', optimizer='adam', metrics=['binary_accuracy'])

model.fit(training_data, target_data, epochs=1000, verbose=2)

print(model.predict(training_data))
```

Classifieur multi-classes

- On peut réaliser un classifieur à K classes grâce à un MLP dont la couche de sortie est de dimension K.
- Chaque neurone de la couche de sortie est associé à une classe.
- A l'issue du calcul des activations, on choisit la classe i, correspondant au neurone ayant l'activation la plus élevée :

$$i = \underset{j=1...K}{\arg\max}\, y_j$$

- Si on associe la valeur 1 à la classe i et 0 aux autres classes, on obtient un **encodage one-hot** de la couche de sortie.

1			i			K
0	0	0	1	0	0	0

Encodage *one hot*

- L'encodage one-hot ou encodage 1 parmi n consiste à encoder une variable pouvant prendre n valeurs sur n bits dont un seul prend la valeur 1.
- Le numéro du bit valant 1 est le numéro de la valeur prise par la variable.
- Dans un problème de classification à K classe, on peut représenter une classe par un vecteur one-hot.
- La classe i sera représentée par le vecteur suivant :

1			i			K
0	0	0	1	0	0	0

- L'encodage one-hot a l'inconvénient d'être gourmand en espace mémoire, la représentation d'une valeur parmi K nécessite K bits.
- Alors que l'encodage en base 2 nécessite $\log_2(K)$ bits.

Hardmax *vs* Softmax

- L'encodage one-hot de la couche de sortie est quelquefois appelé aussi **hardmax** : on ne retient de l'activation la plus élevée m uniquement qu'elle est la plus élevée.
- Il peut être intéressant de garder plus d'information, par exemple si m est beaucoup plus élevée que les activations des autres neurones (des autres classes).
- On peut utiliser pour cela la fonction $softmax()$.

Softmax

- La fonction $softmax()$ transforme un vecteur de K valeurs réelles en un vecteur de K valeurs réelles dont la somme vaut 1.
- Les valeur en entrée peuvent être positives, négatives, nulles, supérieures ou inférieures à 1, la fonction $softmax()$ les transforme en valeurs comprises entre 0 et 1, qui peuvent être interprétées comme des **probabilités**.
- Si une valeur en entrée est grande, elle sera transformée en une probabilité élevée et si elle est petite ou négative, elle sera transformée en une probabilité faible.

$$softmax(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{K} e^{y_j}}$$

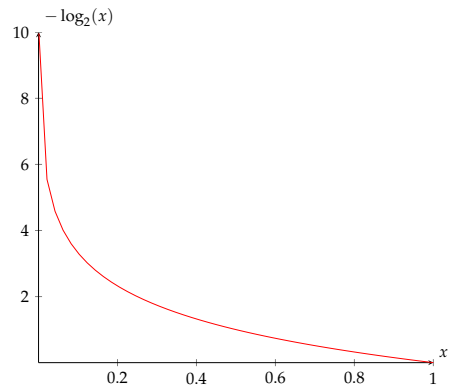
Entropie croisée

- On appelle q la distribution de probabilités calculée par le réseau
- On appelle p la distribution de probabilités de référence
- Quelle fonction d'erreur $E(p,q)$ pour de la classification en n classes ?
- On souhaite que $E(p,q)$ soit minimale si $q(i) = 1$ et $p(i) = 1$
- et qu'elle soit maximale si $q(i) = 0$ et $p(i) = 1$.
- Une fonction qui vérifie cette propriété est l'**entropie croisée** :

$$H(p,q) = -\sum_{i=1}^{n} p(i) \log_2(q(i))$$

référence p	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0
prédiction q	0	0	1	0	0	0.2	0.1	0.5	0.2	0	0.2	0	0.1	0	0.3
erreur $E(p,q)$			0					1					3.32		

La fonction $f(x) = -\log_2(x)$



Estimation par maximum de vraisemblance

- Dans le cas où $p(i) = 1$, alors $H(p, q) = -\log_2(q(i))$

$$
\begin{aligned}
\mathbf{w}^* &= \arg\min_{\mathbf{w}\in\mathbb{R}^k} \sum_{i=1}^{N} H(p_i, q_i) \\
&= \arg\min_{\mathbf{w}\in\mathbb{R}^k} \sum_{i=1}^{N} -\log_2(q_i(y_i)) \\
&= \arg\max_{\mathbf{w}\in\mathbb{R}^k} \sum_{i=1}^{N} \log_2(q_i(y_i)) \\
&= \arg\max_{\mathbf{w}\in\mathbb{R}^k} \prod_{i=1}^{N} q_i(y_i)
\end{aligned}
$$

- Minimiser l'entropie croisée revient à maximiser la probabilité des données.
- Ce mode d'estimation est appelé estimation par maximum de vraisemblance.

Apprentissage

- Les paramètres du MLP sont estimés à l'aide des données d'apprentissage :

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

- On dispose des valeurs observées pour l'entrée $\mathbf{x}$ et la sortie y, mais pas pour les couches cachées h.
- Du fait de la non-linéarité des fonctions d'activation, la fonction d'erreur est généralement **non convexe**.
- L'apprentissage se fait par descente du gradient de la fonction d'erreur, **sans garantie d'atteindre le minimum global**.

Sources

- Ian Goodfellow, Yoshua Bengio, Aaron Courville, *Deep Learning*, MIT Press, 2016.