# Introduction aux réseaux de neurones

Matériel de cours rédigé par Pascal Germain, 2018

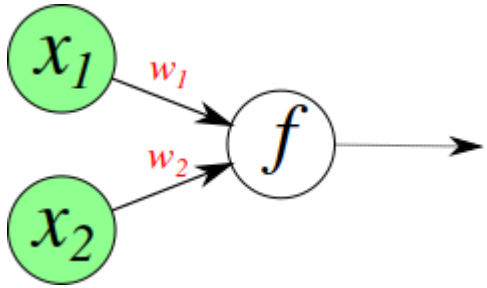
---

Out[1]: voir/cacher le code.

# Cacher ce neurone que je ne saurais voir

Un *réseau de neurones artificiels* n'est qu'un simple *graphe de calcul*, qui exprime une fonction (possiblement) complexe comme une succession d'opérations simples.

Commencons par examiner un réseau de neurones très simple, possédant deux neurones d'entrées et un neurone de sortie. Ce réseau représente une fonction $R_{\mathbf{w}} : \mathbb{R}^2 \to \mathbb{R}$, que nous illustrons ainsi:




Le réseau $R_{\mathbf{w}}$ ci-dessus reçoit deux valeurs en entrée: $x_1$ et $x_2$. Ces valeurs sont respectivement multipliées par les poids $w_1$ et $w_2$. Le neurone de droite reçoit les valeurs $w_1 x_1$ et $w_2 x_2$, et les combine en appliquant la fonction $f$. La valeur obtenue correspond à la sortie du réseau.

$$R_{\mathbf{w}}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = f(w_1 x_1 + w_2 x_2)\,.$$

# Régression: La neurone linéaire

Par exemple, si $f$ est la fonction identitée $f(x) = x$, ce réseau représente l'opération:

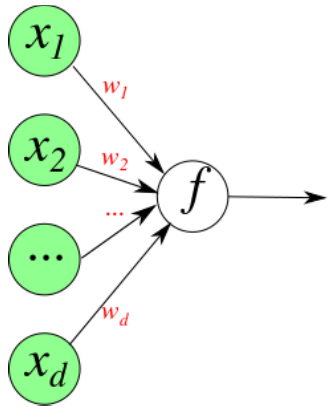
$$R_{\mathbf{w}}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = f(w_1 x_1 + w_2 x_2) = w_1 x_1 + w_2 x_2 \,.$$

Nous avons donc ici d'un réseau de neurones dans sa plus simple expression, représentant un produit scalaire entre un vecteur $\mathbf{x} = (x_1, x_2)$ et un vecteur $\mathbf{w} = (w_1, w_2)$ :

$$R_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} \,.$$

En réalité, un réseau de neuronnes possédera généalement plus de 2 entrées. On notera $d$ la dimension de l'espace d'entré du réseau (le nombre de neurones d'entrée). On a donc $\mathbf{x} \in \mathbb{R}^d$ et $\mathbf{w} \in \mathbb{R}^d$.

$$R_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{d} w_i x_i = \mathbf{w} \cdot \mathbf{x}\,.$$



## Entraînement du réseau

Typiquement, l'entraînement d'un réseau de neurones consistera a présenter au réseau un ensemble d'apprentissage afin qui «apprenne» les poids $\mathbf{w}$.

Cet **ensemble d'apprentissage** sera noté $S$ et contiendra $n$ observations,

$$S = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\},$$

où une *observation* est un couple entrée-sortie $(\mathbf{x}, y)$.

## Fonction de perte

Afin de guider le processus d'apprentissage, nous devons choisir une fonction de perte $L\big(y', y\big)$. L'apprentissage consistera alors à résoudre le problème suivant:

$$\min_{\mathbf{w}} \left[ \frac{1}{n} \sum_{i=1}^{n} L\Big( R_{\mathbf{w}}(\mathbf{x}_i), y_i \Big) \right].$$

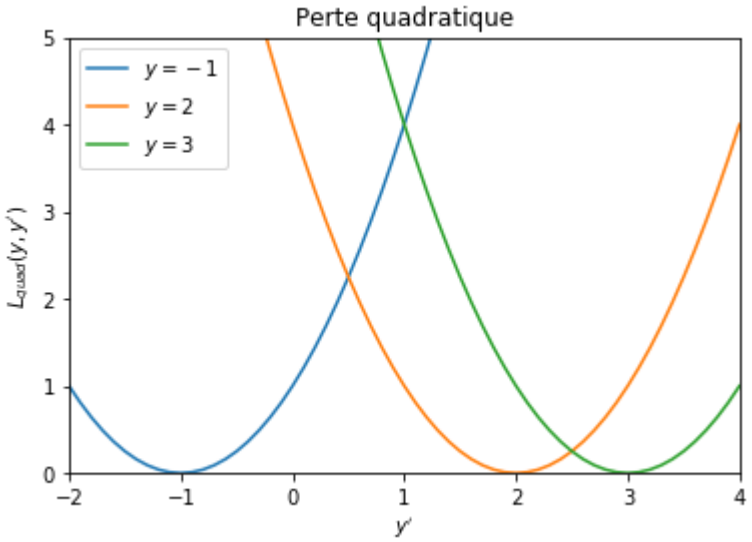
## Perte quadratique

Prenons l'exemple de la perte quadratique:

$$L_{\text{quad}}\left(y', y\right) = (y' - y)^2 \, .$$

Ici, on note $y'$ l'étiquette prédite par le réseau de neurones et $y$ la véritable étiquette d'une observation $\mathbf{x}$.

Out[2]: voir/cacher le code.



Le problème d'apprentissage devient alors

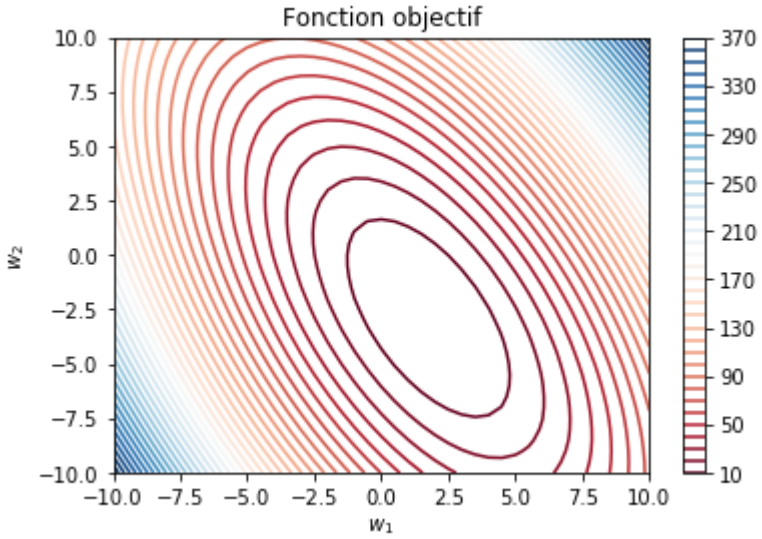
$$\min_{\mathbf{w}} \left[ \frac{1}{n} \sum_{i=1}^{n} (\mathbf{w} \cdot \mathbf{x}_i - y_i)^2 \right].$$

Imaginons un ensemble d'entraîment à 3 observations:

$$S = \{ \quad ((1,1), \quad -1), \\ ((0,-1), \quad 3), \\ ((2,\tfrac{1}{2}), \quad 2) \quad \}$$

Illustrons la fonction objectif correspondante:

Out[3]: voir/cacher le code.



## C'est la méthode des moindres carrés!

On sait comment trouver la réponse exacte au problème d'optimisation.

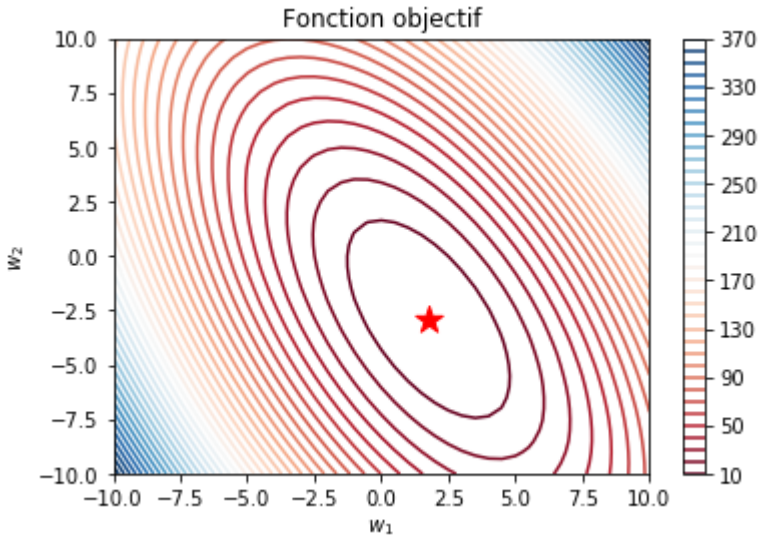
En représentant les observations $S$ sous forme matricielle,

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 0 & -1 \\ 2 & \frac{1}{2} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} -1 \\ 3 \\ 2 \end{bmatrix}$$

On sait que $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \approx (1.76, -2.90)$

voir: https://fr.wikipedia.org/wiki/M%C3%A9thode_des_moindres_carr%C3%A9s (https://fr.wikipedia.org/wiki/M%C3%A9thode_des_moindres_carr%C3%A9s)

Out[4]: voir/cacher le code.



## Régularisation

Comme tout algorithme d'apprentissage, un réseau de neurone est susceptible de *surapprendre* les données d'apprentissage. Nous verrons dans ce cours quelques techniques pour se prémunir les réseaux de neurones contre le surapprentisage.

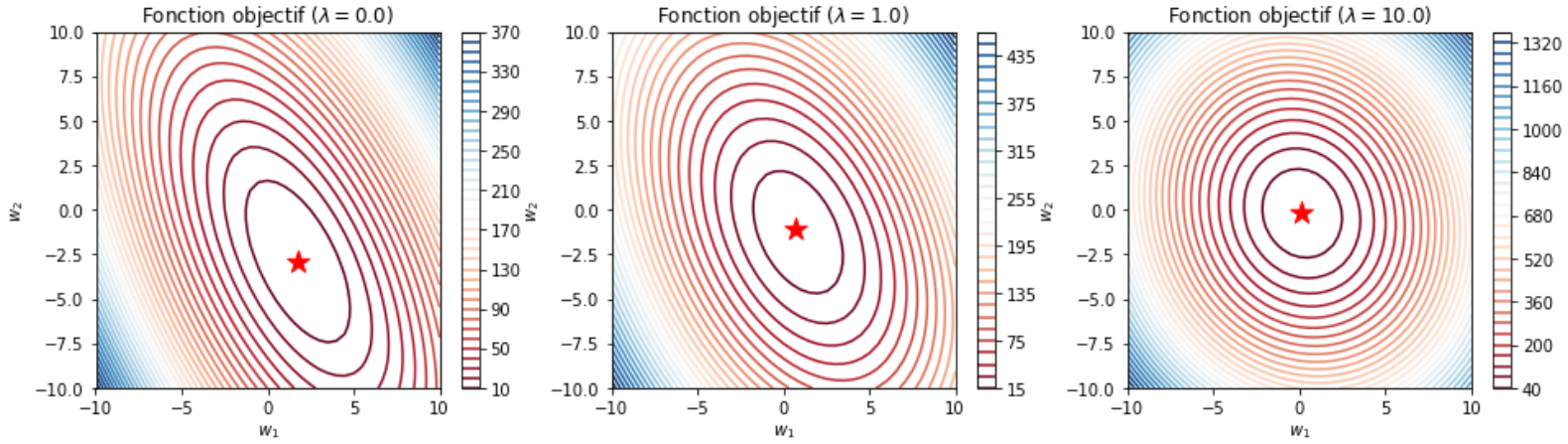
# Régularisation $\mathbf{L^2}$

La première technique est d'ajouter un régularisateur à la fonction objectif. Il est fréquent d'utiliser la norme Euclidienne des poids $\mathbf{w}$ au carré comme fonction de régularisation:

$$\min_{\mathbf{w}} \left[ \frac{1}{n} \sum_{i=1}^{n} L\big(R_{\mathbf{w}}(\mathbf{x}_i), y_i\big) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right].$$

Ici, $\lambda > 0$ est un *hyperparamètre* de la procédure d'entraîment du réseau de neurone.

Out[5]: voir/cacher le code.



## C'est la régression de Ridge (c.-à-d. les moindres carrés régularisés)

On sait comment trouver la réponse exacte au problème d'optimisation.

En représentant les observations $S$ sous forme matricielle,

$$\mathbf{X} = \begin{bmatrix} 1 & 1 \\ 0 & -1 \\ 2 & \frac{1}{2} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} -1 \\ 3 \\ 2 \end{bmatrix},$$

on sait que $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}d)^{-1}\mathbf{X}^T\mathbf{y}$ .

| $\boldsymbol{\lambda}$ | $\longrightarrow$ | $\boldsymbol{w_1^*}$ | $\boldsymbol{w_2^*}$ |
|---|---|---|---|
| 0 | $\longrightarrow$ | 1.76 | -2.90 |
| 1 | $\longrightarrow$ | 0.73 | -1.05 |
| 10 | $\longrightarrow$ | 0.13 | -0.15 |

voir: https://fr.wikipedia.org/wiki/R%C3%A9gularisation_de_Tikhonov (https://fr.wikipedia.org/wiki/R%C3%A9gularisation_de_Tikhonov)

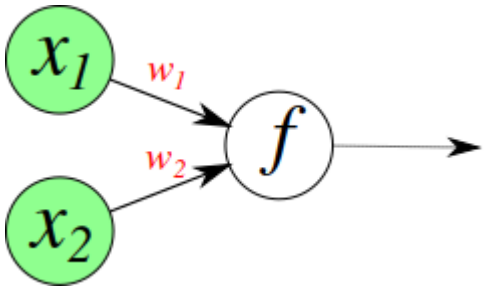
# Classification: La neurone sigmoïdale

La combinaison de la neurone de sortie linéaire et de la perte quadratique convient bien aux problèmes de régression (où les étiquettes sont des valeurs réelles ($y \in \mathbb{R}$).

Imaginons maintenant que nous voulons résoudre un problème de classification binaire:

$$y \in \{0, 1\}$$

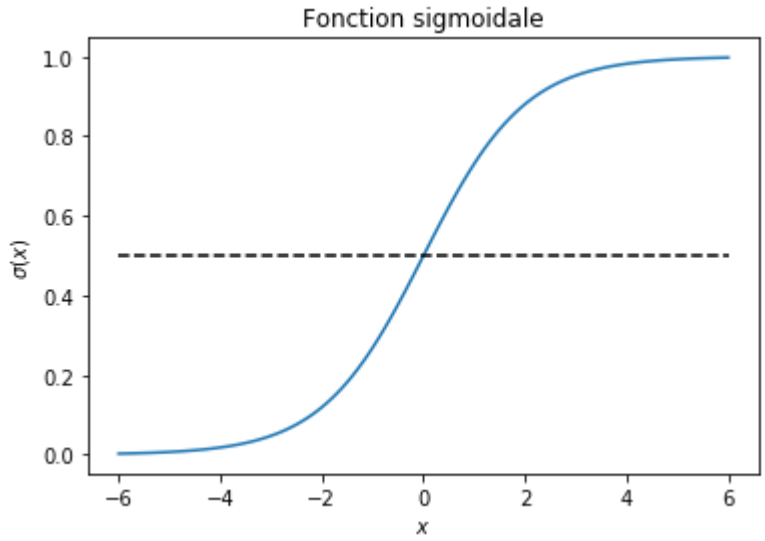
Reprenons notre réseau de neurones simplifié.



Nous allons considérer que la neurone de sortie applique la **fonction sigmoidale**, habituellement notée $\sigma$:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \, .$$

Out[6]: voir/cacher le code.



La sortie du réseau de neurones sera donc comprise entre $0$ et $1$. On considère que l'étiquette prédite est $y = 0$ lorsque

$$R_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w} \cdot \mathbf{x}) < 0.5\,,$$

et $y = 1$ sinon.

Une sortie près de $0.5$ est interprétée comme une grande incertitude envers la résultat. Inversement, plus la sortie est près de $0$ ou de $1$, plus on considère que le réseau est *confiant* envers sa décision.

Une interprétation *Bayésienne* de la sortie du neurone sigmoïdal est de la voir comme la probabilité, selon le réseau $R_{\mathbf{w}}$, que $y = 1$ pour une certaine observation $\mathbf{x}$:

$$P(y = 1 \,|\, \mathbf{x}; \mathbf{w}) \;=\; \sigma(\mathbf{w} \cdot \mathbf{x})\,.$$

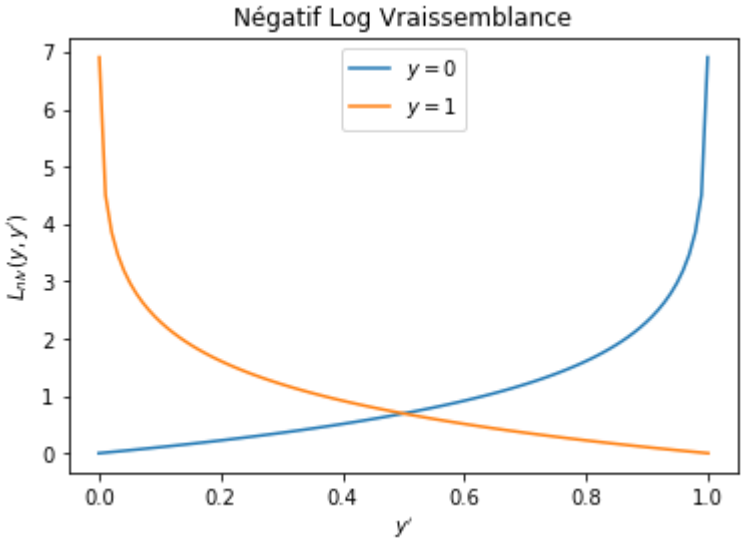
Conséquemment:

$$P(y = 0 \,|\, \mathbf{x}; \mathbf{w}) \;=\; 1 - P(y = 1 \,|\, \mathbf{x}; \mathbf{w}) \;=\; 1 - \sigma(\mathbf{w} \cdot \mathbf{x})\,.$$

Lors de l'apprentissage, on désire pénaliser le réseau d'autant plus que la probabilité attribuée à l'étiquette d'une observation s'éloigne de la véritable étiquette. La fonction de perte utilisée dans ce contexte est nommé la perte du **négatif log vraissemblance**:

$$\begin{aligned} L_{\mathrm{nlv}}\left(y', y\right) &= -\,y \log(y') - (1-y)\log(1-y') \\ &= \begin{cases} -\log(1-y') & \text{si } y = 0\,, \\ -\log(y') & \text{si } y = 1\,. \end{cases} \end{aligned}$$

Out[7]: voir/cacher le code.



**Récapitulons.**

1. La neurone de sortie applique la fonction sigmoidale à la somme de ses entrées:

$$\sigma(\mathbf{w}\cdot\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}\,.$$

1. Si $\sigma(\mathbf{w}\cdot\mathbf{x}) > 0.5$, on déclare que $y = 1$.

2. Lors de l'apprentissage, on pénalise selon

$$\begin{aligned} L_{\text{nlv}}\Big(\sigma(\mathbf{w}\cdot\mathbf{x}), y\Big) &= -y\log(\sigma(\mathbf{w}\cdot\mathbf{x})) - (1-y)\log(1-\sigma(\mathbf{w}\cdot\mathbf{x})) \\ &= \quad \vdots \\ &= -y\mathbf{w}\cdot\mathbf{x} + \log(1+e^{\mathbf{w}\cdot\mathbf{x}}) \end{aligned}$$

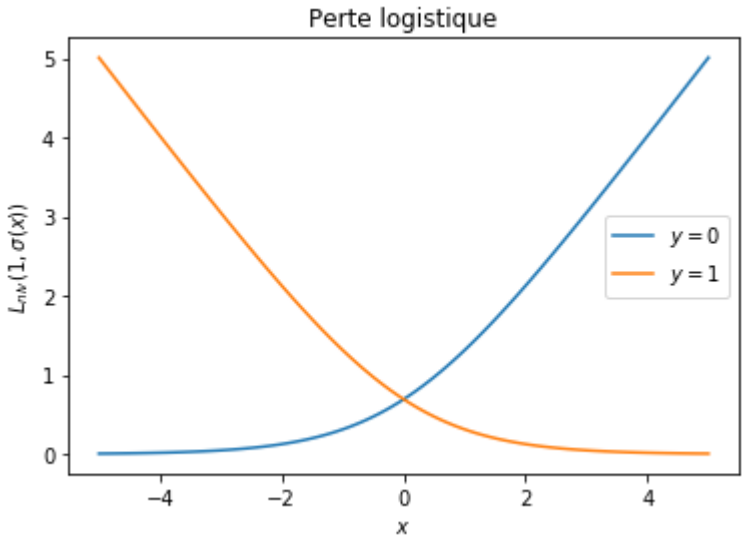
La fonction objectif est alors:

$$\min_{\mathbf{w}} \left[ \frac{1}{n} \sum_{i=1}^{n} -y_i \mathbf{w} \cdot \mathbf{x}_i + \log(1 + e^{\mathbf{w} \cdot \mathbf{x}_i}) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right].$$

## Il s'agit de la régression logistique!

### À démontrer en exercice

Out[8]: voir/cacher le code.

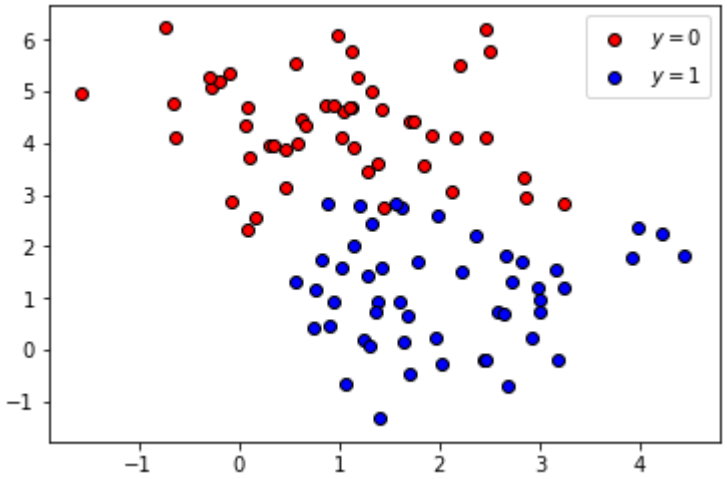


## Exemple de classification

Ci-dessous, nous générons au hasard un ensemble de 100 observations en deux dimensions à l'aide de la fonction `sklearn.datasets.make_blobs`

(voir: http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html))

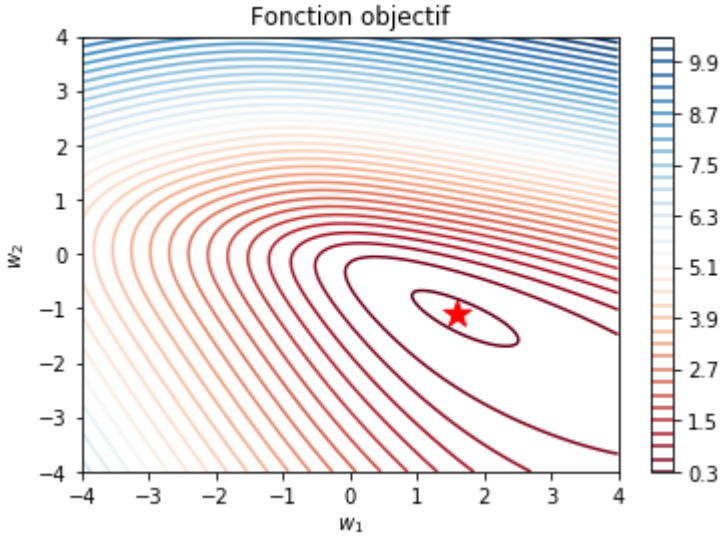
Out[9]: voir/cacher le code.



Illustrons la fonction à optimiser (avec $\lambda = 0.01$):

$$\frac{1}{n}\sum_{i=1}^{n} -y_i \mathbf{w}\cdot\mathbf{x}_i + \log(1+e^{\mathbf{w}\cdot\mathbf{x}_i}) + \frac{\lambda}{2}\|\mathbf{w}\|^2 .$$

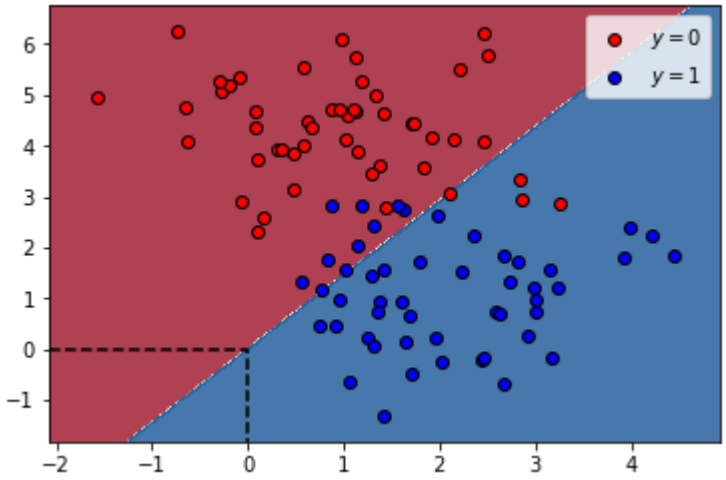
Out[10]: voir/cacher le code.




L'optimum se trouve en $\mathbf{w}^* \approx \begin{bmatrix} 1.60 \\ -1.10 \end{bmatrix}$.

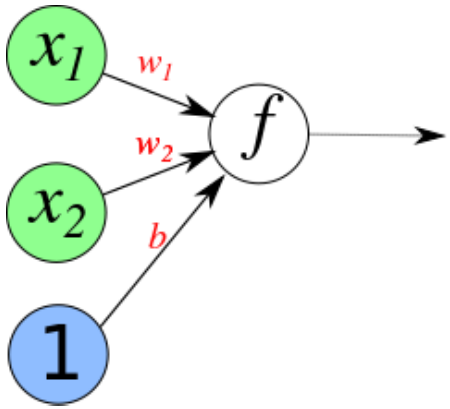
Ce qui correspond au prédicteur suivant.

Out[11]: voir/cacher le code.



# Ajout d'un biais

Pour éviter de restreindre la fonction de prédiction à passer par l'origine, on ajoute un **biais**:



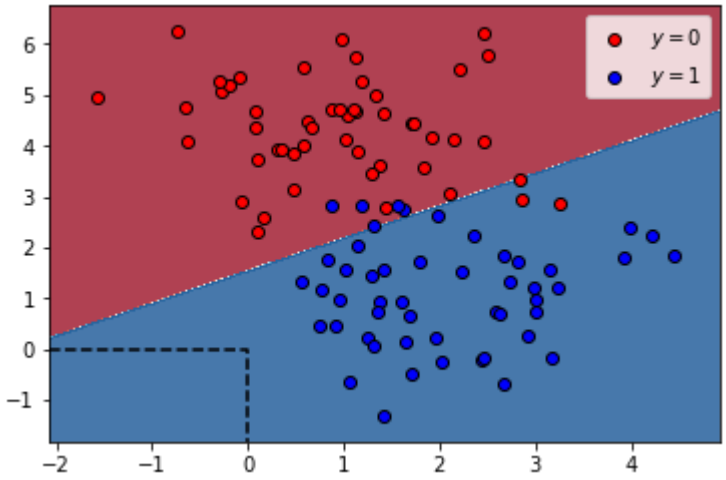

Dans le cas de notre réseau simple, la fonction de prédiction devient alors:

$$R_{\mathbf{w},b}(\mathbf{x}) = f\left(\sum_{i=1}^{d} w_i x_i + b\right) = f\big(\mathbf{w}\cdot\mathbf{x} + b\big).$$

Retournons à notre exemple de régression logistique (c'est-à-dire le réseau de neurone où la neurone de sortie $f$ est la **fonction sigmoïdale** $\sigma$ et a perte du **négatif log vraissemblance** $L_{\text{nlv}}$. Le problème d'optimisation **avec biais** s'exprime ainsi:

$$\min_{\mathbf{w},b} \left[ \frac{1}{n} \sum_{i=1}^{n} -y_i(\mathbf{w} \cdot \mathbf{x}_i + b) + \log(1 + e^{\mathbf{w}\cdot\mathbf{x}_i + b}) + \frac{\lambda}{2}\|\mathbf{w}\|^2 \right].$$

Out[12]: voir/cacher le code.




L'optimum se trouve en $\mathbf{w}^* \approx \begin{bmatrix} 0.97 \\ -1.52 \end{bmatrix}, b \approx 2.33$.