Régressions linéaire et logistique, réseaux de neurones

Module 4

Objectifs

Objectifs

- maîtriser les algorithmes de régressions linéaire et logistique
- comprendre comment fonctionne et s'entraîne un réseau de neurones

Avant propos

Avant propos

- régression linéaire = plus simple des réseaux de neurones
- régression logistique = très légère variation sur régression linéaire qui permet de classifier
- bases qui serviront à comprendre les réseaux de neurones généraux

Plan du cours :

- introduction aux réseaux de neurones
- régression linéaire
- régression logistique
- réseaux de neurones généraux

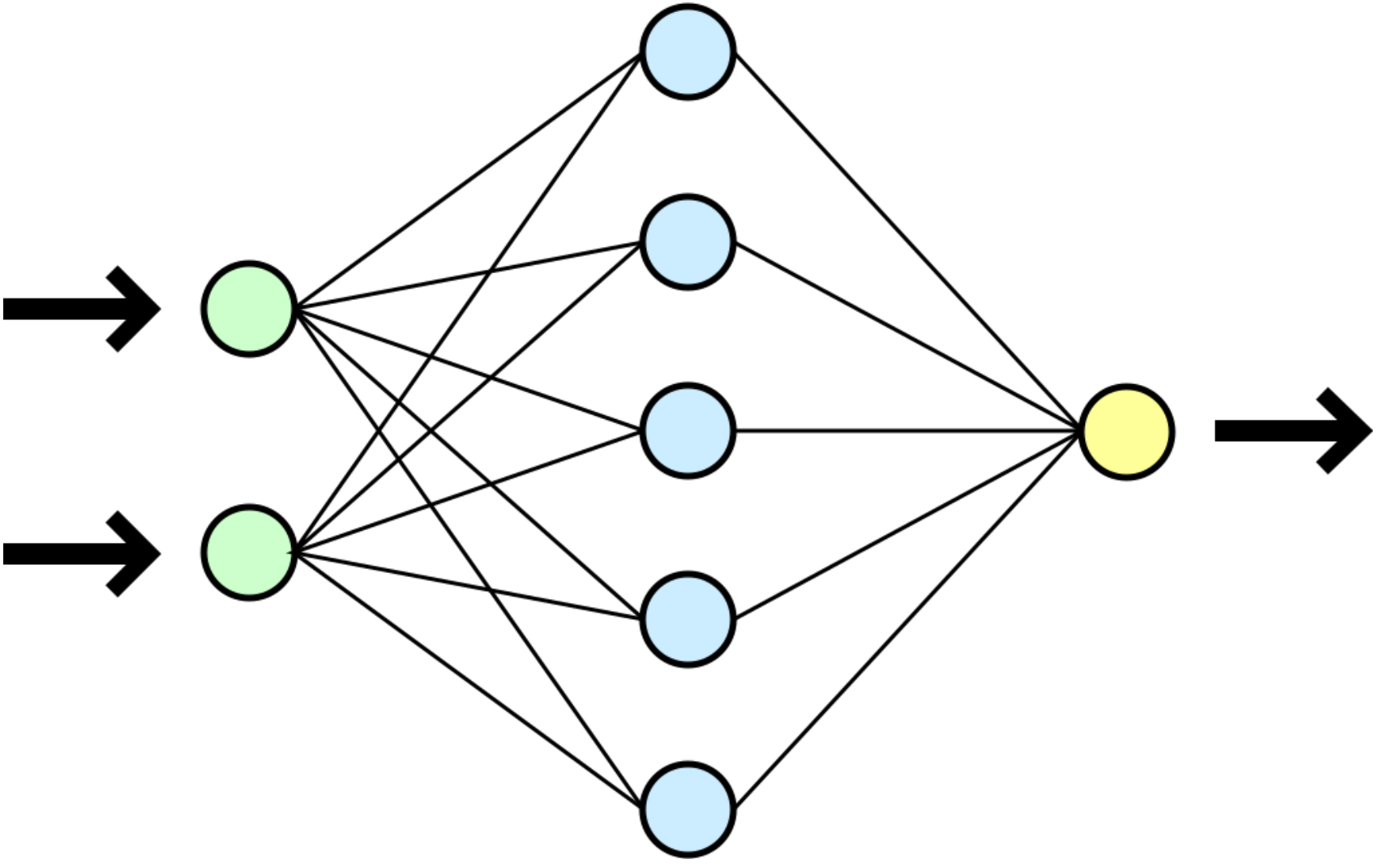
Introduction aux réseaux de neurones

Réseaux de neurones

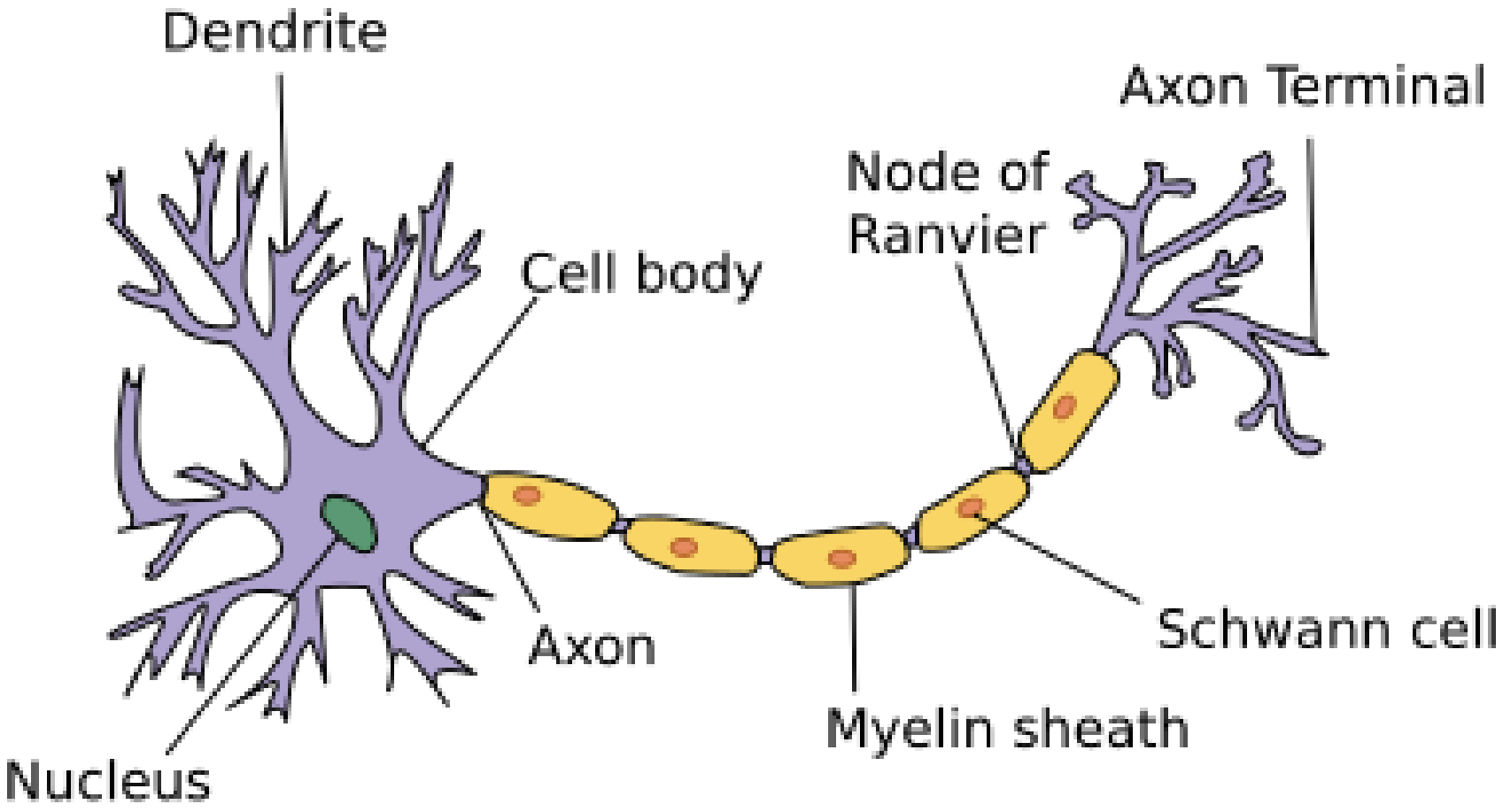
Intuition :

1 neurone $=$ 1 calcul simple
1M neurones $=$ 1M calculs simples
$=$ 1 calcul complexe

Réseaux de neurones

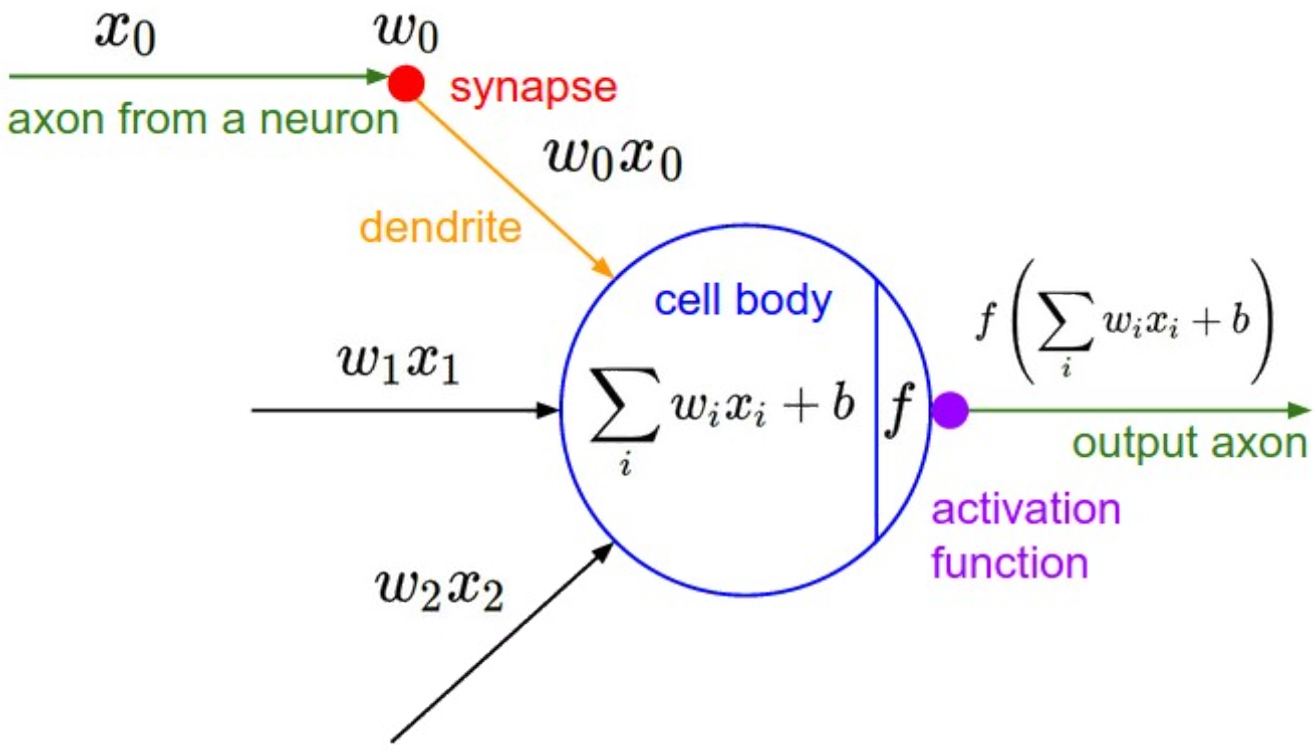


Neurone biologique



Neurone artificiel

Simulation extrêmement basique d'un neurone : somme pondérée + activation.



Apprentissage

Modification des poids des neurones (poids des sommes).

But : la somme finale = output attendu.

Fonction de perte (rappel module 3)

Mesurer la qualité du modèle :

$$L(\widehat{\mathbf{y}}, \mathbf{y})$$

Plus cette perte est proche de 0, meilleur est le modèle.

Apprentissage — modélisation

Apprendre = minimiser la fonction de perte.

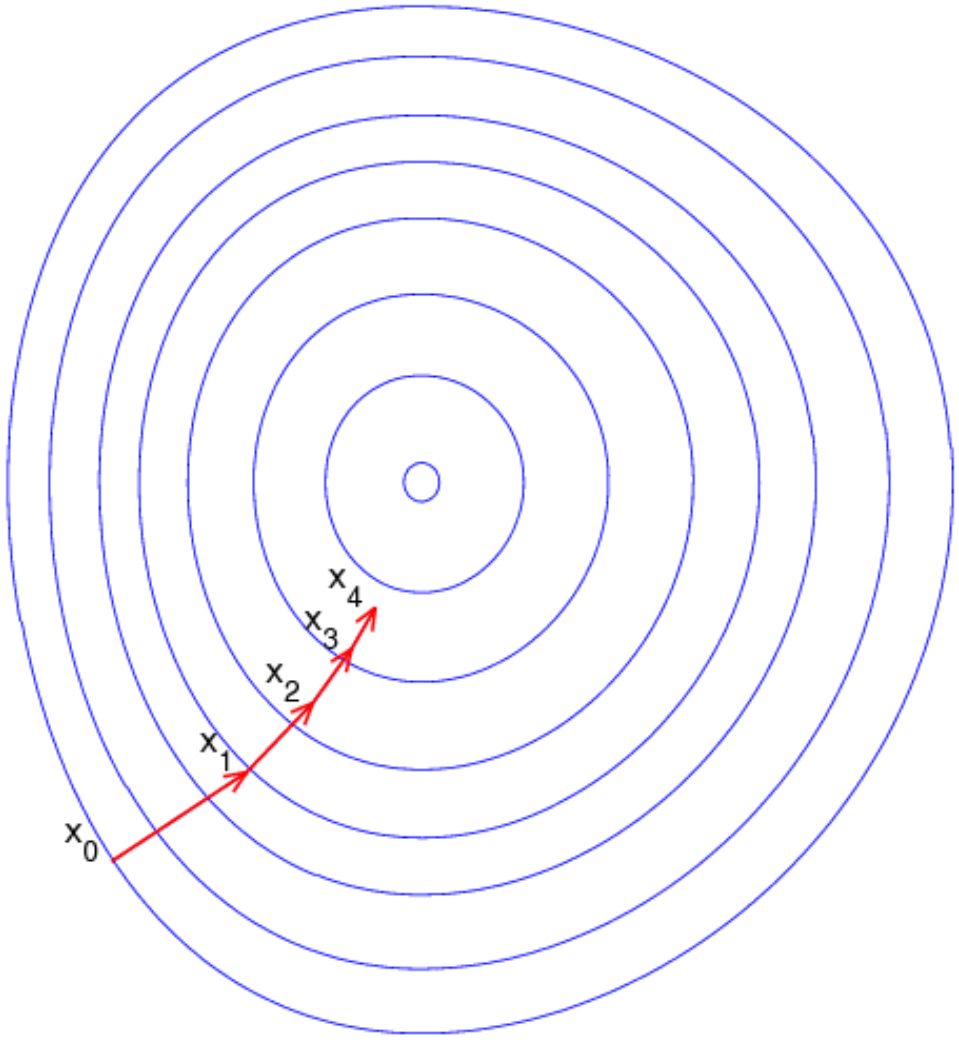
$$\arg\min_{\widehat{\mathbf{y}}} L(\widehat{\mathbf{y}}, \mathbf{y})$$

Apprentissage — optimisation par descente du gradient

Minimisation de L en allant à l'opposé du gradient, par pas de taille α.

$$w \leftarrow w - \alpha \nabla_w L$$

Apprentissage — optimisation par descente du gradient

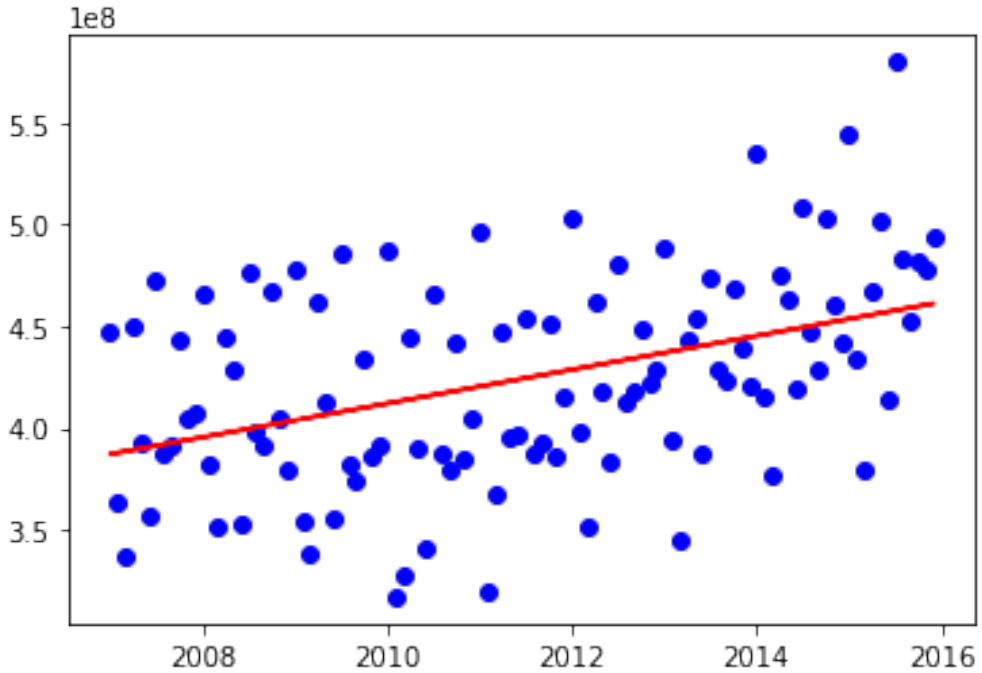


Régression linéaire

Régression linéaire simple

Étant donné une variable en entrée, prédire une variable continue en sortie.

Contrainte très forte : $\hat{y}_i = \theta_0 + \theta_1 x_i$



Exemple

Prédiction de vos bénéfices après un an étant donné les actions que vous avez chez Airbus :

Input 500

Output 37

Régression linéaire simple

Trouver les meilleurs θ_0 et θ_1 avec :

$$\widehat{\mathbf{y}} = \theta_0 + \theta_1 \mathbf{x}$$

Définition d'une fonction de perte

Pour savoir si nos θ_0 et θ_1 sont bons :

- calcul de $\hat{y}_i$ pour chaque x_i du training set
- calcul des $y_i - \hat{y}_i$ (résiduels)
- grands résiduels = mauvais modèle

$\rightarrow$ Fonction de perte = mesurer la taille des résiduels

Fonction de perte

Pour un seul exemple, la perte est définie comme :

$$L(\hat{y}_i, y_i) = (y_i - \hat{y}_i)^2$$

Pour tous les exemples, on la définit comme la moyenne des pertes :

$$L(\widehat{\mathbf{y}}, \mathbf{y}) = \frac{(\widehat{\mathbf{y}} - \mathbf{y})^2}{n}$$

avec n taille du dataset.

Apprentissage

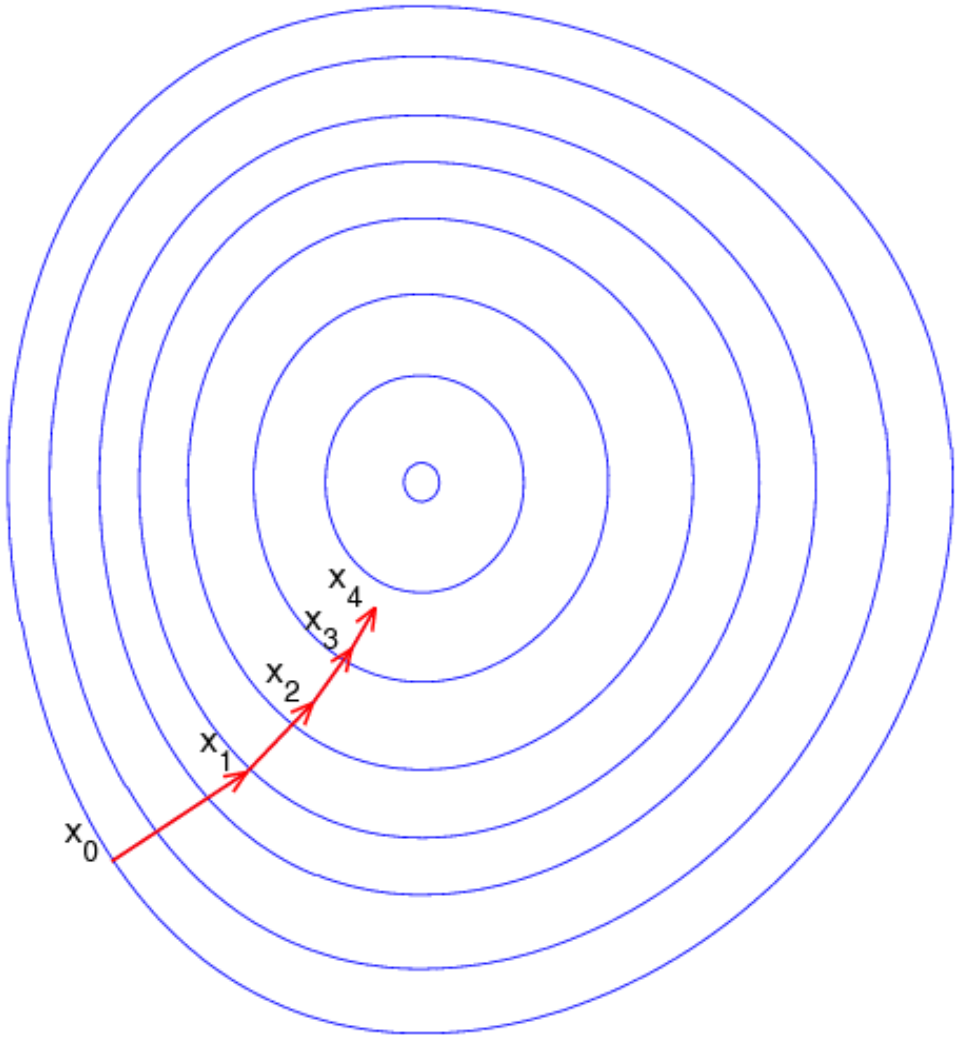
- solution directe existe pour les petits datasets
- descente de gradient dans la pratique (s'adapte à tout)

Apprentissage

Algorithme :

- calculer le gradient de θ par rapport à L
- ajuster θ avec la règle $\theta \leftarrow \theta - \alpha \nabla_\theta L$
- répéter jusqu'à `max_iter` itérations ou convergence

Apprentissage



Régression linéaire multiple

Étant donné plusieurs variables en entrée, prédire une variable continue en sortie.

Exemple

Prédiction de vos bénéfices après un an étant donné votre portfolio :

Inputs $[(\text{LVMH}, 2000), (\text{TOTAL}, 1500), (\text{AIRBUS}, 500)]$

Output 42

Modélisation

Trouver les meilleurs θ_0 à θ_n avec :

$$y_i = \theta_0 + \sum_{k=1}^{n} \theta_k x_{ik}$$

Apprentissage

Même algorithme que pour la régression linéaire simple !

Limitations

- **TRÈS** grosse hypothèse de linéarité
- suppose que les variables sont normalement distribuées
- dans la pratique, on peut pallier quelques limitations

Practice time

TP : `day2/Linear regression.ipynb`

Régression logistique

Régression logistique

Étant donné une variable en entrée, prédire une variable continue en sortie, **dans** $[\mathbf{0}, \mathbf{1}]$**.**

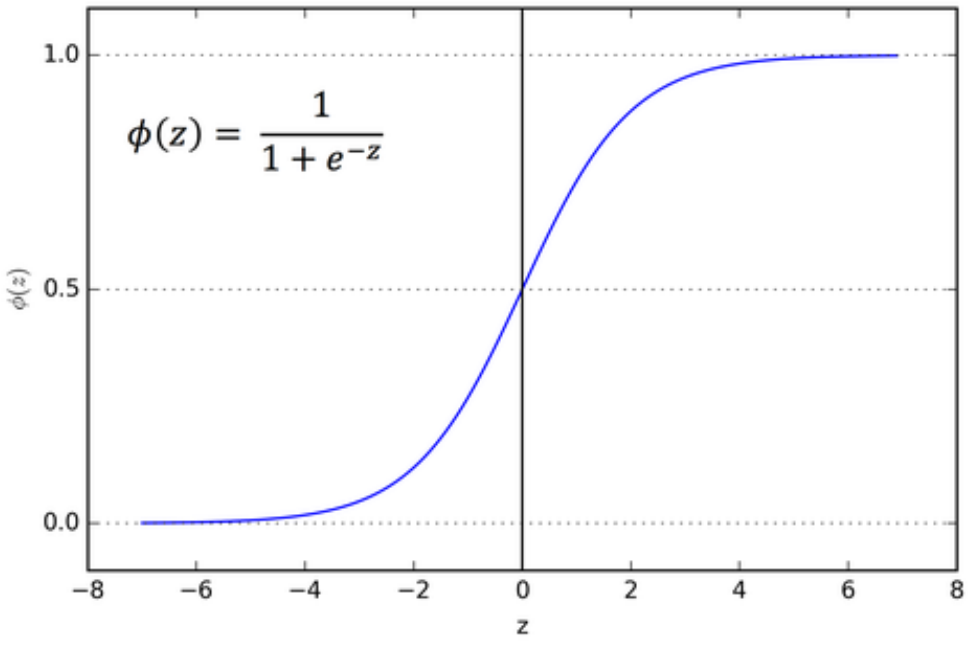
$\rightarrow$ classification

Modélisation

Trouver les meilleurs θ_0 à θ_n avec :

$$\hat{y}_i = \sigma(\theta_0 + \sum_{k=1}^{n} \theta_k x_{ik})$$

Fonction sigmoid



Entropie croisée binaire

$y_i \in \{0, 1\}$, $\hat{y}_i \in [0, 1]$.

$$\mathrm{BCE}(\hat{y}_i, y_i) = y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)$$

Apprentissage

Même algorithme que pour la régression linéaire !

Régression logistique multi-classes

Régression logistique multi-classes

Étant donné des variables en entrée, prédire n variable dans $\{0, 1\}$ en sortie, avec $\sum_{\mathbf{i=1}}^{\mathbf{n}} \mathbf{y_i} = \mathbf{1}$.

$\rightarrow$ output = loi de probabilité

Idée clef

- mener n régressions linéaires en parallèle (avoir n neurones d'output)
- ajouter une normalisation pour garantir que l'output somme à 1

Normalisation

Utilisation d'un softmax :

$$\text{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{n} e^{z_k}}$$

- besoin de tous les outputs pour normaliser
- somme à 1
- produit des sorties plutôt sparse (compresse vers 0 et 1)

Apprentissage

Encore une fois, l'apprentissage ne change pas.

Practice time

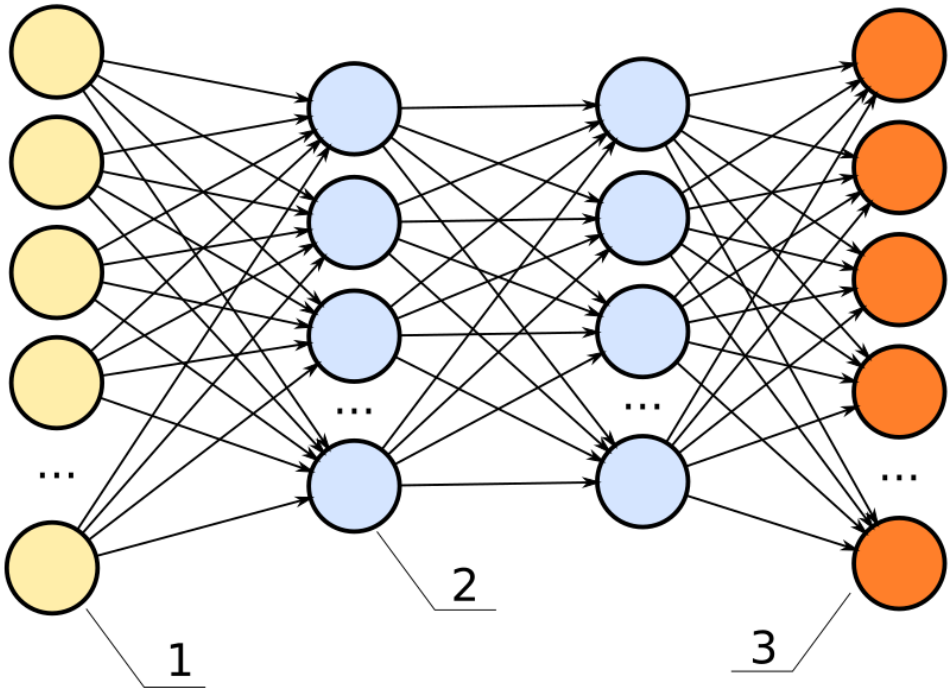
TP : day2/Logistic regression.ipynb

Réseaux de neurones généraux

Différences avec les régression linéaires et logistiques

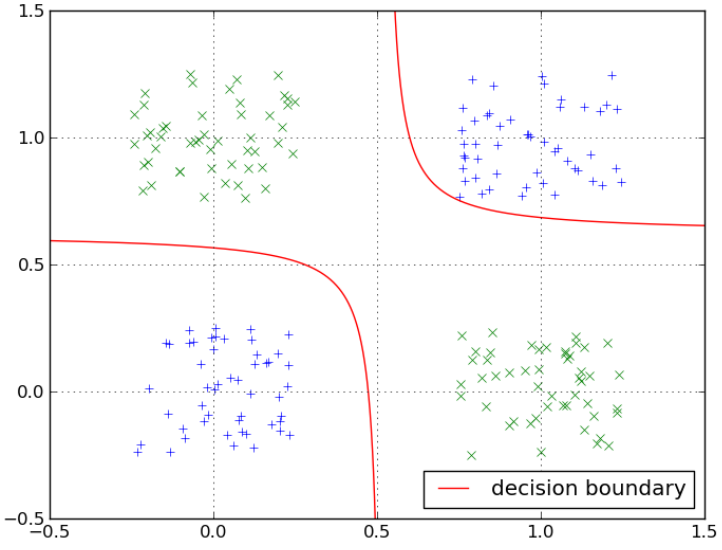
- pas seulement une couche d'output (profondeur)
- pas seulement des fonctions linéaires (activations)
- des types de neurones particuliers pour certains réseaux

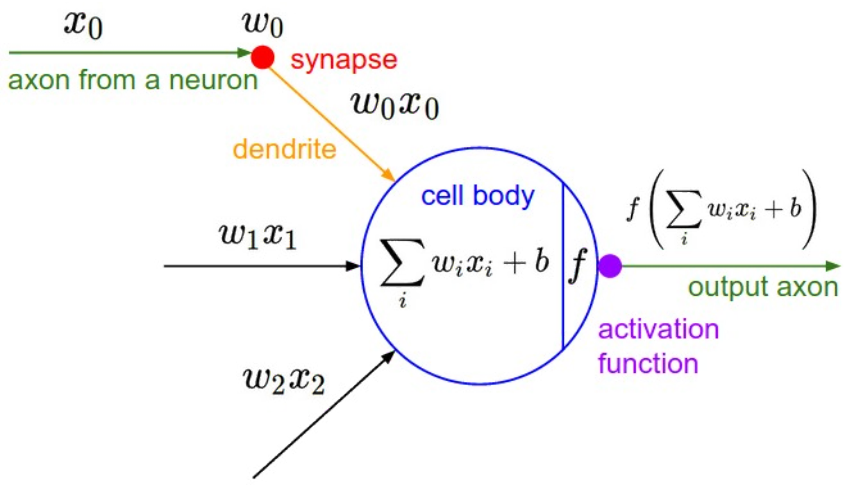
Profondeur



→ nécessité d'adapter l'apprentissage

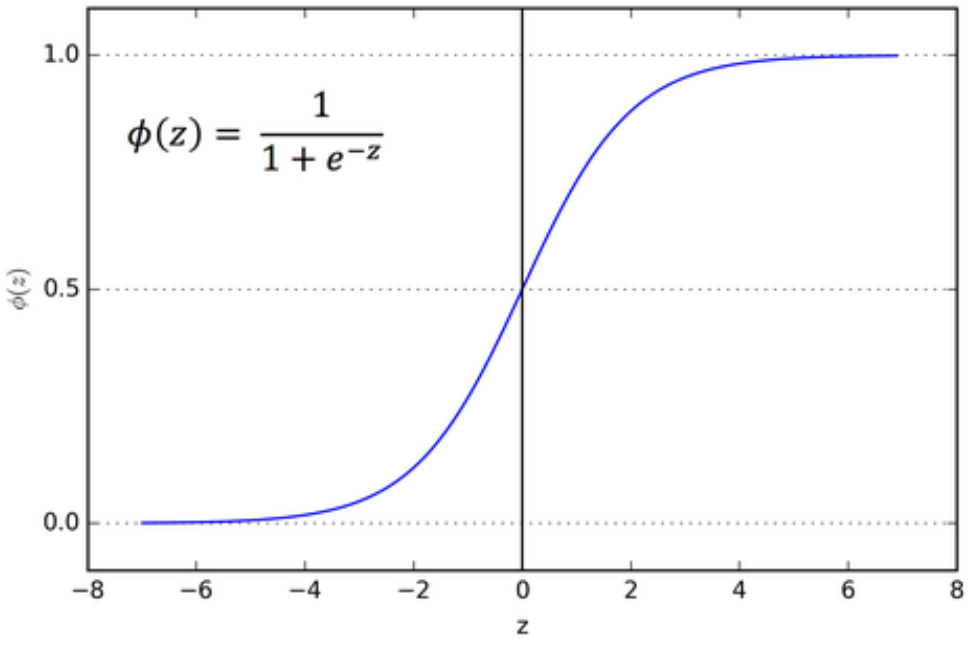
Non-linéarité





$\rightarrow$ f non linéaire nécessaire ! (sigmoid, tanh, ReLU, ...)

Non-linéarité

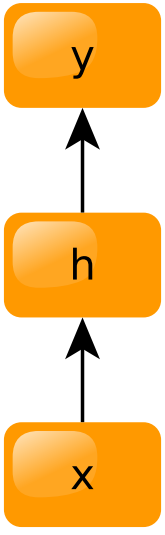


Apprentissage

- définition de perte
- descente de gradient
- si réseau profond, nécessité de calculer beaucoup de dérivées

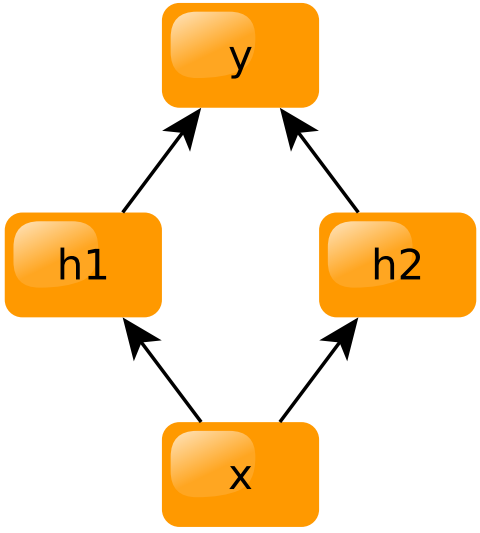
$\rightarrow$ utilisation d'une règle de chainage pour ne pas tout recalculer

Règle de chainage — cas simple



$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial h}\frac{\partial h}{\partial x}$$

Règle de chainage — deux chemins



$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial h_1}\frac{\partial h_1}{\partial x} + \frac{\partial y}{\partial h_2}\frac{\partial h_2}{\partial x}$$

Diversité des fonctions de pertes

Il existe des pertes :

- pour des classements
- pour des objectifs multiples
- avec des propriétés mathématiques particulières
- avec des gains de temps d'entrainement
- modélisées par des réseaux de neurones (!!!)

Il existe sûrement une loss pour votre problème spécifique !

Conclusion

Conclusion

- neurone = somme pondérée (+ activation pour les réseaux généraux)
- apprentissage = trouver les bons poids des sommes
- métrique = fonction de perte
- technique = rétropropagation des gradients
- régression linéaire = régression, régression logistique = classification
- les deux sont des réseaux de neurones sans couche cachée