CI-3 : Implémenter et comparer des méthodes numériques

CI-3-2 Traiter un ensemble de données numériques

Objectifs — ANALYSER RESOUDRE CALCULER VALIDER

A l'issue de la séquence, l'élève doit être capable de:

- C1 Proposer une démarche de résolution
  - Choisir une démarche de résolution d'un problème d'ingénierie numérique ou d'intelligence artificielle.
    - ⋆ Choix des algorithmes (réseaux de neurones, k plus proches voisins et régression linéaire multiple)
- C3 Mettre en œuvre une démarche de résolution numérique
  - Résoudre un problème en utilisant une solution d'intelligence artificielle
    - ⋆ Apprentissage supervisé.
    - ⋆ Choix des données d'apprentissage.
    - ⋆ Mise en œuvre des algorithmes (réseaux de neurones, k plus proches voisins et régression linéaire multiple).
    - ⋆ Phases d'apprentissage et d'inférence.

Table des matières

1 Intelligence artificielle : définition et méthodes d'apprentissage

1.1 Cadre d'utilisation de l'intelligence artificielle

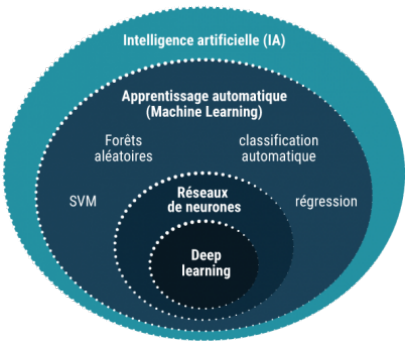
Contrairement au cours précédent sur les méthodes de résolutions d'un problème d'ingénierie, on ne dispose pas ici de fonction mathématique établissant un lien initial entre des données d'entrée et des données de sortie. L'objectif de ce cours est de traiter des données (*data*), c'est à dire un ensemble (généralement grand) de vecteurs (ensemble discret).

L'intelligence artificielle étudiée ici ne cherche pas à appliquer des méthodes numériques sur un modèle de connaissance ou de comportement mais plutôt à classer, évaluer, estimer ou modéliser des données avec l'aide de données d'apprentissage (*training data*) pour obtenir un modèle prédictif afin de traiter les nouvelles données.

1.2 Définition de l'intelligence artificielle

Historiquement, on trouve plusieurs apparitions du mot intelligence artificielle comme dans les années 1950 avec le test de Turing (*Computing Machinery and Intelligence*) ou en 1956, année pendant laquelle John McCarthy présente le test alpha-beta (Conférence de Dartmouth).

De la façon la plus simple, on peut appeler intelligence artificielle (IA ou *AI*), un ensemble de théories et de techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence humaine (cf le Larousse).



Par rapport à l'objet de ce cours, on concentre l'étude de l'IA sur un ensemble d'algorithmes permettant d'apprendre à partir de données pour traiter de nouvelles données. C'est l'apprentissage automatique (AA) ou Machine Learning.

L'étude des réseaux de neurones (RN) est une partie de l'apprentissage automatique (AA), elle même une partie de l'intelligence artificielle (IA).

1.3 L'apprentissage automatique - principe de base de l'intelligence artificielle

La méthode la plus célèbre de l'IA est l'apprentissage automatique (AA). Inventée en 1959 par Arthur Samuel pour doter le premier jeu de Dames d'IA, le programme basé sur l'AA, a appris à jouer aux Dames tout seul.

On distingue principalement trois grandes catégories d'apprentissage automatique :

- Apprentissage supervisé (*Supervised learning*) :
  Des labels (ou étiquettes) sont posées sur des données pour les identifier. Le label peut être une valeur continue (donc d'un ensemble dense comme $\mathbb{R}^n$, $\mathbb{C}$, ...), par exemple pour les cas de techniques de régression ou une valeur discrète (une catégorie comme un pays, une période de l'histoire, un entier, un booléen ...), pour des techniques de classifications.
  A partir des données d'entraînement, on lance une phase d'apprentissage pour établir un modèle de prédiction. Ce modèle doit déterminer pour les nouvelles données, la valeur à attribuer (cas de la régression) ou la catégorie la plus proche (cas de la classification). C'est la phase d'inférence.
  Lorsque les prédictions du modèle sont trop proches des données d'apprentissage, on parle alors de sur-apprentissage (*overfitting*). Le système risque de ne pas bien traiter les nouvelles données d'entrée. Ce qui compte c'est d'obtenir un bon résultat sur les données de test plutôt que sur les données d'entraînement.

  Remarque : on parle d'apprentissage supervisé profond (*Deep learning*) lorsqu'on utilise un réseau de neu-

rones ayant un nombre de couches plus ou moins important.

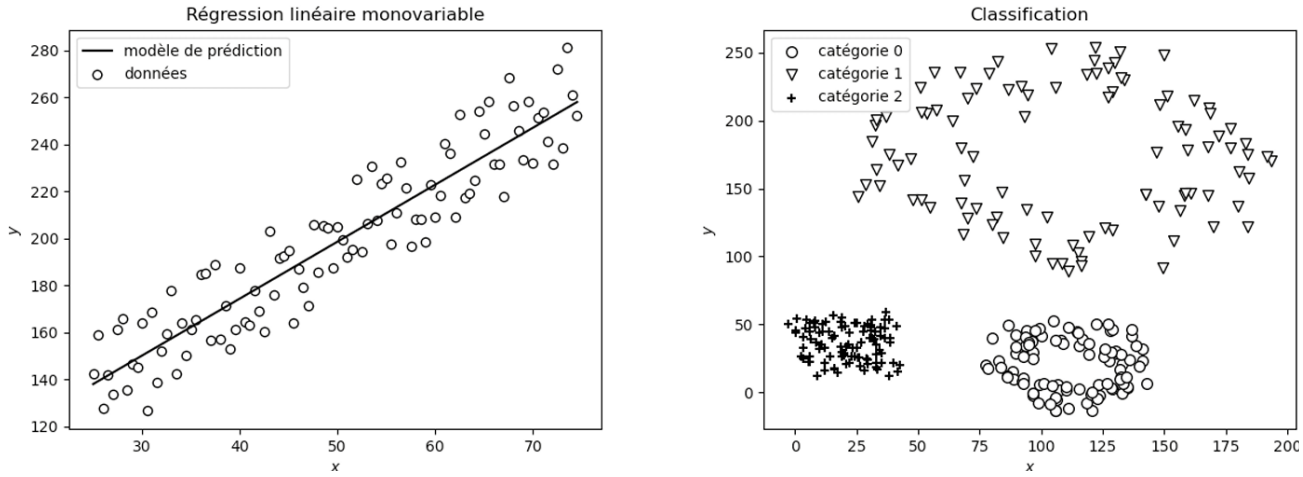


Figure 1 – Différences entre régression (à gauche) et classification (à droite)

- Apprentissage non supervisé (*Unsupervised learning*) :
  Cette forme d'apprentissage utilise des données d'apprentissage non étiquetées. On ne sait pas a priori ce que l'on cherche. L'objectif est donc de dégager des paramètres pertinents pour structurer les données. Cela peut être de partitionner les données (*clustering*) ou de réduire le nombre de paramètres influents sur la sortie (*embedding* ou *dimensionality reduction*).

  Remarque : on utilise parfois les algorithmes d'apprentissage non supervisé pour faire émerger des regroupements pertinents et donc des labels (ou étiquettes) pour l'apprentissage supervisé.

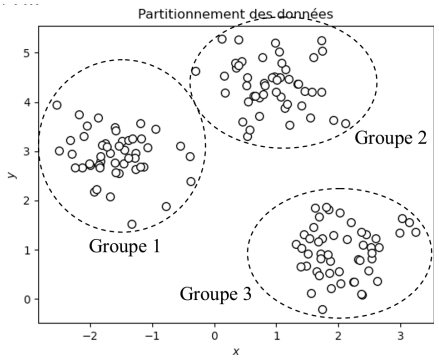


Figure 2 – partitionnement

- Apprentissage par renforcement (*Reinforcement learning*) :
  L'apprentissage par renforcement est une forme d'apprentissage supervisé incrémental qui utilise des données arrivant successivement pour modifier le comportement du système. Cette forme d'apprentissage est particulièrement utilisé en robotique, dans les jeux ou dans les chatbots capables de s'améliorer en fonction des réactions des utilisateurs.
  Pour cela, dans la conduite de procédés en temps réel (*process*), on développe un système (*agent*) qui améliore ses performances en intégrant un jeu de récompenses (*rewards*) modifiant ses propres paramètres pour maximiser les récompenses.
  On a donc un état, une action, une récompense et en fonction de la récompense on change l'état pour établir une nouvelle action afin d'obtenir une meilleure récompense.

Remarque : seul l'apprentissage supervisé est au programme.

1.4 Qualité d'un modèle de prédiction

1.4.1 Évaluation de la régression

L'erreur quadratique moyenne (*Mean Squared Error*) peut permettre d'évaluer la pertinence d'un modèle de prédiction d'une grandeur appartenant à un ensemble dense. Avec $\hat{y}_i$ la valeur prédite, y_i la valeur réelle et N le nombre de données, l'erreur quadrique moyenne MSE est définie par :

$$\text{MSE} = \frac{1}{\text{N}} \cdot \sum_{i=1}^{\text{N}} (y_i - \hat{y}_i)^2$$

L'erreur quadratique moyenne se calcule sur le jeu de données de test une fois la phase d'apprentissage terminée. La valeur de R^2, le coefficient de détermination de Pearson indique la pertinence ou la qualité du modèle.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2} \quad \text{où} \quad \overline{y} = \frac{1}{N}.\sum_{i=1}^{N} y_i$$

$\overline{y}_i$ est la valeur moyenne des valeurs réelles.

R^2 varie entre 0 et 1. Le cas de la prédiction parfaite est atteint si R = 1.

1.4.2 Évaluation de la classification

La pertinence d'un modèle de classification peut être évalué par une matrice de confusion. Il s'agit d'une matrice carrée M_c dont la dimension (nombre de lignes et nombre de colonnes) est celle du nombre de labels (ou étiquettes) associés au modèle. Une case i, j de la matrice M_c correspond au nombre de données de label i qui ont été identifiées comme possédant le label j. Les données bien classées sont donc celles de la diagonale.

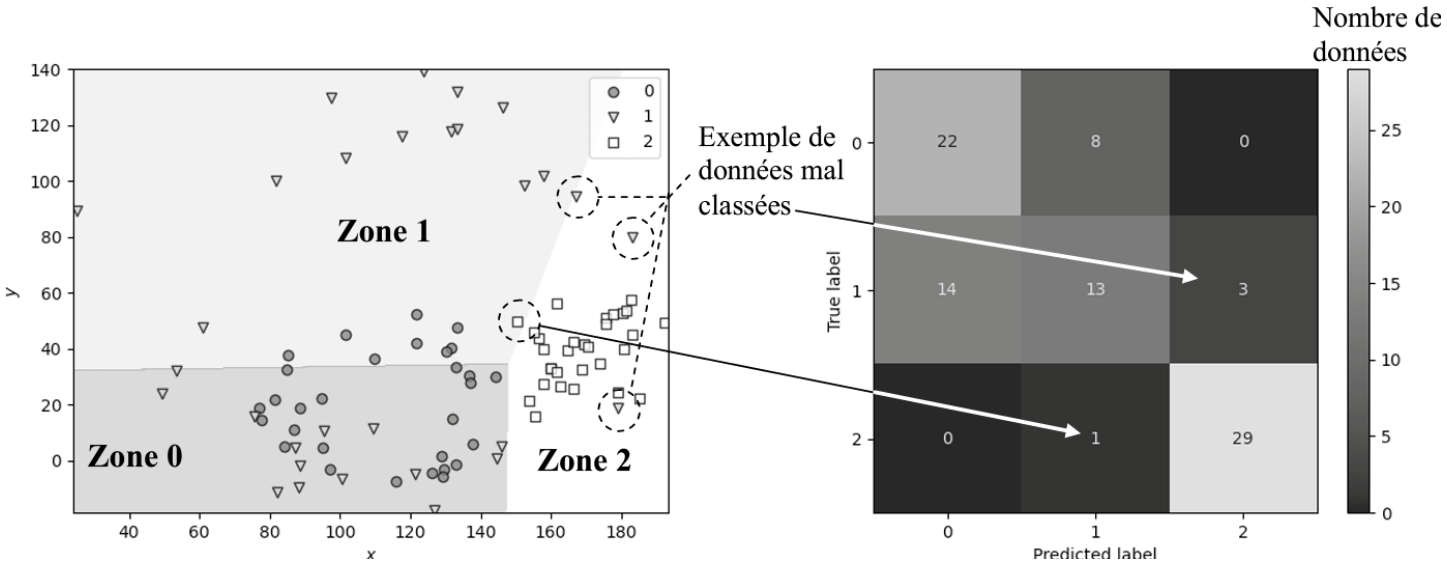


Figure 3 – Matrice de confusion associé à des données d'entrée de dimensions 2 associée à 3 types d'étiquettes

L'exemple de la Fig 3 est basé sur des données de dimensions 2 (x, y) dont l'étiquette appartient à 3 catégories : 0, 1 et 2. L'apprentissage a permis d'identifier 3 zones d'appartenance (Zone 0, Zone 1 et Zone 2) à partir de données d'apprentissage. Les nouvelles données du jeu de test sont représentées sur le graphique de gauche de la Fig 3. Le graphique de droite donne la matrice de confusion correspondante.

Définition : Précision (*Accurancy*)

Ratio entre les bonnes prédictions et le nombre total de données.

Définition : Erreur (*Error*)

Ratio entre les fausses prédictions et le nombre total de données.

Remarque : précision + erreur = 1

Définition : Sensibilité ou rappel (*Sensitivity* ou *recall*)

Probabilité que l'appartenance d'une donnée à une catégorie soit prédite vraie si elle l'est en réalité.

Définition : Précision pour une catégorie (*Precision*)

Ratio entre le nombre de bonnes prédictions d'appartenance à une catégorie et le nombre total de données prédites comme appartenant à cette même catégorie.

Définition : Spécificité (*Specificity*)

Probabilité que l'appartenance d'une donnée à une catégorie soit prédite fausse si en réalité la donnée n'appartient effectivement pas à celle-ci.

1.5 Prétraitement des données

Afin d'éviter des problème de conditionnement de matrices pour la méthode des moindres carrés ou d'inégalité de traitement des coefficients w_j de l'estimateur lors de la descente de gradient, il convient d'opérer un prétraitement des données (*data preprocessing*). On peut alors procéder à la :

- **normalisation** des données pour que chaque valeur de paramètre soit comprise entre 0 et 1
- **standardisation** des données pour la moyenne soit nulle et que l'écart type soit unitaire (particulièrement pour la méthode de descente de gradient).

Il faut aussi séparer les données en deux jeux : les données d'entraînement (*training data*) et les données de test (*test data*). Les premières permettent d'élaborer le modèle prédictif ; les secondes permettent d'en évaluer la pertinence.

2 Algorithmes associés à l'intelligence artificielle

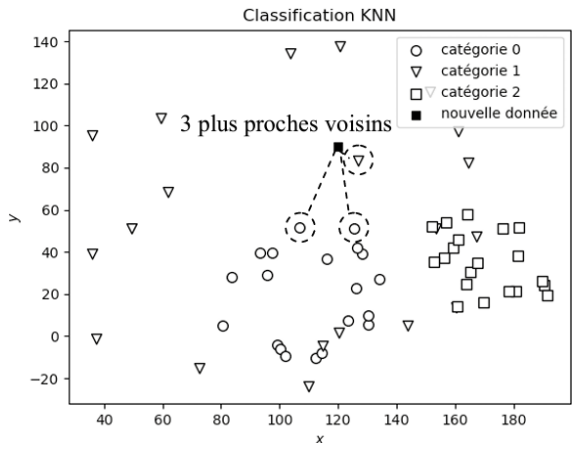
2.1 Algorithme des k plus proches voisins (Algorithme *k-NN*)

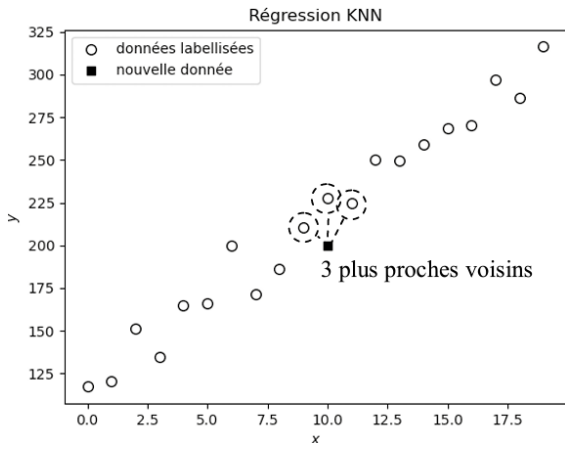
2.1.1 Principe

L'algorithme des k plus proches voisins (*k-Nearest Neighbors*) consiste à trouver dans un jeu de données labellisées (données d'apprentissage), les k données d'entrée les plus proches de la nouvelle donnée dont on souhaite prédire le label (donc la sortie). La phase d'apprentissage se résume à charger les données déjà labellisées.

Lorsque les k plus proches voisins sont identifiés, on attribue à la nouvelle donnée :

- en régression, la valeur moyenne des k plus proches voisins. On peut aussi pondérer la valeur de chacun des voisins en fonction de l'inverse de sa distance à la nouvelle donnée.
- en classification, le label le plus fréquent parmi les k plus proches voisins. En cas d'égalité entre plusieurs catégories, il est possible de déterminer le barycentre de chacune des catégories majoritaires et affecter à la nouvelle donnée le label de la catégorie dont le barycentre est le plus proche.





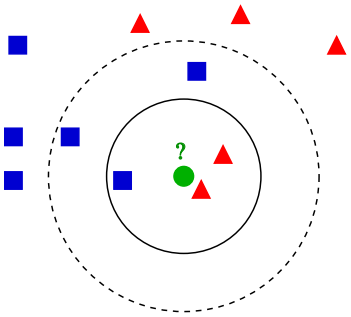
2.1.2 Distances

L'algorithme k-NN fait intervenir la notion de distance entre données. Avec deux données $\vec{u}$ et $\vec{v}$ de même nature et de dimensions n (vecteurs de $\mathbb{R}^n$, nombres binaires codés sur n bits, chaines de n caractères), on pose $\vec{u} = (u_j)_{j=1}^n$ et $\vec{v} = (v_j)_{j=1}^n$. Il existe plusieurs façons de calculer la distance entre $\vec{u}$ et $\vec{v}$:

- **distance euclidienne** liée à la norme euclidienne : $d_{\mathrm{E}}(\vec{u},\vec{v}) = \sqrt{\sum_{j=1}^{n}(u_j - v_j)^2}$
- **distance de Manhattan** liée à la norme 1 : $d_{\mathrm{M}}(\vec{u},\vec{v}) = \sum_{j=1}^{n}|u_j - v_j|$
- **distance de Hamming**, somme des différences $d_{\mathrm{H}}(\vec{u},\vec{v}) = \mathrm{Card}(\{j|u_j \neq v_j\}) = \#\{j|u_j \neq v_j\}$

2.1.3 Influence du paramètre k

Généralement, le paramètre k de l'algorithme k-NN est petit (et entier !...). Avec $k = 1$, il s'agit d'affecter à la nouvelle donnée le label de son plus proche voisin.



L'exemple ci-dessous donné par Antti Ajanki sur Wikipedia montre l'influence de k.

L'échantillon de test (point vert) pourrait être classé soit dans la première classe de carré bleu ou la seconde classe de triangles rouges. En effet :

- si $k = 3$ (cercle en ligne pleine) il est affecté à la classe des triangles car il y a deux triangles et seulement un carré dans le cercle considéré.
- si $k = 5$ (cercle en ligne pointillée) il est affecté à la classe des carrés (3 carrés face à deux triangles dans le cercle externe).

2.2 Régression linéaire multiple

2.2.1 Modèle de prédiction

Le modèle de prédiction avec un algorithme d'apprentissage (AA) consiste à faire passer une donnée d'entrée x en donnée de sortie y à partir d'un ensemble d'apprentissage de N données $(x_i,y_i)_{i=1}^{\mathrm{N}}$. On considère que les données d'entrée sont de dimension n. On pose alors $x_i = (x_{ij})_{j=1}^{n}$ et $\mathrm{X}_i = \begin{pmatrix} 1 & x_{i1} & \dots & x_{in} \end{pmatrix}$ avec $x_{i0} = 1$.

Dans le cas de la régression linéaire multiple, y est une variable continue et on introduit une famille de poids w (*weight*) tel que $w = (w_j)_{j=0}^{n}$, soit $\mathrm{W} = \begin{pmatrix} w_0 & \dots & w_n \end{pmatrix}^{\mathrm{T}}$. Le terme w_0 est appelé biais. L'aspect linéaire multiple tient dans la formule :

$$\begin{aligned} y_i(x_i,w) &= w_0 + w_1.x_{i1} + \ldots + w_n.x_{in} \\ &= w_0 + \textstyle\sum_{j=1}^{n} w_j.x_{ij} = \mathrm{X}_i.\mathrm{W} \end{aligned}$$

Comme $\mathrm{N} \gg n$, il est rarement possible de trouver une unique famille w qui permette d'obtenir $\forall i \in [\![1;\mathrm{N}]\!]$ $y_i(x_i,w) = w_0 + \sum_{j=1}^{n} w_j.x_{ij}$. Il s'agit donc d'apprendre à partir des données, quelle famille $w = (w_j)_{j=0}^{n}$ minimise l'erreur de prédiction qu'on appelle perte (*loss*).

On note $\hat{y}_i$ la valeur prédite par le modèle composé des poids $\hat{w}$ et ε_i l'erreur de prédiction pour la donnée d'entrée x_i. On obtient alors $y_i = \hat{y}_i(x_i,w) + \varepsilon_i$

Ainsi, d'un point de vue matricielle avec N données d'entrée de dimension n, $\hat{\mathrm{Y}} = \mathbb{X}.\hat{\mathrm{W}}$ où $\mathbb{X}$ est une matrice $(\mathrm{N}, n+1)$:

$$\hat{\mathrm{Y}} = \mathbb{X}.\hat{\mathrm{W}} \quad \Rightarrow \quad \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{\mathrm{N}} \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_{\mathrm{N}1} & x_{\mathrm{N}2} & \dots & x_{\mathrm{N}n} \end{pmatrix} . \begin{pmatrix} \hat{w}_0 \\ \hat{w}_1 \\ \vdots \\ \hat{w}_n \end{pmatrix} \quad \text{et} \quad \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_{\mathrm{N}} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{\mathrm{N}} \end{pmatrix} - \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_{\mathrm{N}} \end{pmatrix} \quad \text{avec} \quad \mathbb{X} = \begin{pmatrix} \mathrm{X}_1 \\ \mathrm{X}_2 \\ \vdots \\ \mathrm{X}_{\mathrm{N}} \end{pmatrix}$$

2.2.2 Méthode des moindres carrés

Pour obtenir un bon modèle de prédiction (cf Partie2.2.1), il faut minimiser la perte (*loss*), c'est à dire la différence entre la valeur prédite $\hat{y}_i$ et la valeur réelle y_i de la donnée x_i en choisissant la meilleure famille w qu'on appelle alors estimateur. La perte est souvent appelée coût.

La méthode des moindres carrés ordinaire (MCO ou *LSE* pour *Least squares approximation*) permet d'estimer la perte, qu'on note J_{MCO}, en prenant la somme des carrés des erreurs ε_i (SSE pour *Sum Squarred Error*) sur les N prédictions :

$$J_{MCO}(\hat{w}) = \sum_{i=1}^{N} \varepsilon_i^2 = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Puisque $\hat{w}$ est de dimension $n+1$, pour minimiser $J_{MCO}(\hat{w})$ il faut résoudre un système linéaire de $n+1$ équations à $n+1$ inconnues (les $\hat{w}_j$, $\forall j \in [\![0,n]\!]$) : $\forall k \in [\![0,n]\!]$, $\dfrac{\partial J_{MCO}(\hat{w}_0,\hat{w}_1,\dots,\hat{w}_n)}{\partial \hat{w}_k} = 0$. Or $\forall k \geq 1$, $\dfrac{\partial J_{MCO}\left((\hat{w}_j)_{j=0}^{n}\right)}{\partial \hat{w}_k} = 0$ conduit à :

$$\frac{\partial \sum_{i=1}^{N}\left(y_i - \left(\hat{w}_0 + \sum_{j=1}^{n} \hat{w}_j.x_{ij}\right)\right)^2}{\partial \hat{w}_k} = 0 \Rightarrow \sum_{i=1}^{N} x_{ik}.\left(\left(\hat{w}_0 + \sum_{j=1}^{n} \hat{w}_j.x_{ij}\right) - y_i\right) = 0 \Rightarrow \sum_{i=1}^{N} x_{ik}.\left(\hat{w}_0 + \sum_{j=1}^{n} \hat{w}_j.x_{ij}\right) = \sum_{i=1}^{N} x_{ik}.y_i$$

$$\text{et si } k = 0 \text{ alors } \sum_{i=1}^{N}\left(\left(\hat{w}_0 + \sum_{j=1}^{n} \hat{w}_j.x_{ij}\right) - y_i\right) = 0 \quad \Rightarrow \quad N.\hat{w}_0 + \sum_{j=1}^{n}\left(\sum_{i=1}^{N} x_{ij}\right).\hat{w}_j = \sum_{i=1}^{N} y_i$$

ce qui se met sous la forme matricielle :

$$\begin{pmatrix} N & \sum_{i=1}^{N} x_{i1} & \dots & \sum_{i=1}^{N} x_{in} \\ \sum_{i=1}^{N} x_{i1} & \sum_{i=1}^{N} x_{i1}.x_{i1} & \dots & \sum_{i=1}^{N} x_{in}.x_{i1} \\ \vdots & \vdots & & \vdots \\ \sum_{i=1}^{N} x_{in} & \sum_{i=1}^{N} x_{i1}.x_{in} & \dots & \sum_{i=1}^{N} x_{in}.x_{in} \end{pmatrix} . \begin{pmatrix} \hat{w}_0 \\ \hat{w}_1 \\ \vdots \\ \hat{w}_n \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_{i1}.y_i \\ \vdots \\ \sum_{i=1}^{N} x_{in}.y_i \end{pmatrix} \quad \Rightarrow \quad \left(\mathbb{X}^T.\mathbb{X}\right).\hat{W} = \mathbb{X}^T.Y$$

Si $\left(\mathbb{X}^T.\mathbb{X}\right)$ est inversible (les données sont a priori aléatoires), alors $\hat{W} = \left(\mathbb{X}^T.\mathbb{X}\right)^{-1}.\mathbb{X}^T.Y$ ce qui donne une solution optimale au problème des moindres carrés.

2.2.3 Cas de la régression monovariable (linéaire ou multiple)

Dans le cas de la régression linéaire monovariable, le problème se résume à l'équation de droite $\hat{y}_i = \hat{w}_0 + \hat{w}_1.x_i$. Le problème matriciel $\left(\mathbb{X}^T.\mathbb{X}\right).\hat{W} = \mathbb{X}^T.\hat{Y}$ devient :

$$\begin{pmatrix} N & \sum_{i=1}^{N} x_i \\ \sum_{i=1}^{N} x_i & \sum_{i=1}^{N} x_i^2 \end{pmatrix} . \begin{pmatrix} \hat{w}_0 \\ \hat{w}_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i.y_i \end{pmatrix}$$

$$\text{et si } \left(\mathbb{X}^T.\mathbb{X}\right) \text{ est inversible } \quad \begin{pmatrix} \hat{w}_0 \\ \hat{w}_1 \end{pmatrix} = \frac{1}{N.\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2} . \begin{pmatrix} \sum_{i=1}^{N} x_i^2 & -\sum_{i=1}^{N} x_i \\ -\sum_{i=1}^{N} x_i & N \end{pmatrix} . \begin{pmatrix} \sum_{i=1}^{N} y_i \\ \sum_{i=1}^{N} x_i.y_i \end{pmatrix}$$

Cependant, parfois, le modèle monovariable linéaire est insuffisant. On lui préfère alors un modèle monovariable polynomiale qui peut se ramener à un modèle linéaire multiple.

En prenant par exemple P, un polynôme de degré 2 tel que $P(x) = a.x^2 + b.x + c$, il s'agit de trouver au mieux les coefficients a, b et c tel que pour tout point (x_i, y_i) des données d'apprentissage, la prédiction $\hat{y}_i = P(x_i)$ soit

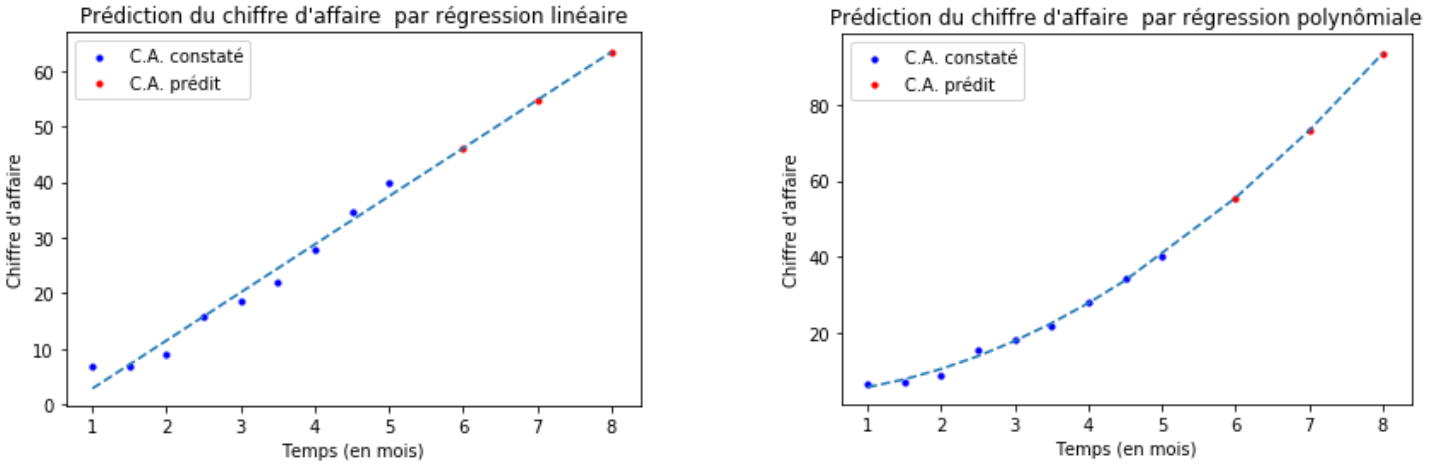


Figure 4 – Exemples d'estimation par régression linéaire et polynomiale

proche de y_i. Or $\hat{y}_i = \mathrm{P}(x_i) = a.x_i^2 + b.x_i + c = \hat{w}_0 + \hat{w}_1.x_{i1} + \hat{w}_2.x_{i2} = \mathrm{X}_i.\hat{\mathrm{W}}$ avec $\hat{\mathrm{W}} = \begin{pmatrix} c \\ b \\ a \end{pmatrix}$ et $\mathrm{X}_i = \begin{pmatrix} 1 & x_i & x_i^2 \end{pmatrix}$

La Fig 4 donne deux exemples permettant de prédire un chiffre d'affaire, avec deux modèles prédictifs (régression monovariable linéaire et polynomiale). On peut alors généraliser la méthode à tout polynôme de degré n.

2.2.4 Descente de gradient

Avec la méthode des moindres carrés (cf Partie 2.2.2), il possible de déterminer directement l'estimateur $\hat{w}$ du modèle de prédiction en inversant un système (voir Pivot de Gauss du cours CI-3-1 méthodes numériques). Or le calcul de l'estimateur peut devenir très long à cause de la quantité de données et de la dimension du problème de régression. On applique alors une méthode de descente de gradient (*batch gradient descent*) pour avoir une valeur approchée de l'estimateur $\hat{w}$.

La méthode de descente de gradient est une méthode itérative visant à minimiser la perte $\mathrm{J}(\hat{w})$. Partant d'un estimateur initial $\hat{w}^{(0)}$, on introduit un paramètre η, le taux d'apprentissage (*learning rate*), afin d'établir la relation de récurrence :

$$\hat{w}^{(k+1)} = \hat{w}^{(k)} - \eta.\nabla \mathrm{J}\left(\hat{w}^{(k)}\right) \quad \text{avec} \quad \nabla \mathrm{J}\left(\hat{w}^{(k)}\right) = \left(\frac{\partial \mathrm{J}\left(\hat{w}^{(k)}\right)}{\partial \hat{w}_0^{(k)}}, \ldots, \frac{\partial \mathrm{J}\left(\hat{w}^{(k)}\right)}{\partial \hat{w}_n^{(k)}}\right)^{\mathrm{T}} = -2.\mathbb{X}^{\mathrm{T}}.\left(y - \hat{y}^{(k)}\right)$$

On distingue 3 critères d'arrêt pour l'algorithme :

- critère d'arrêt sur le nombre d'itérations : $k = k_{max}$
- critère d'arrêt sur la stagnation de l'algorithme : $\left\|\hat{w}^{(k+1)} - \hat{w}^{(k)}\right\| \leq \varepsilon$
- critère d'arrêt sur la taille du résidu : $\left\|\nabla \mathrm{J}\left(\hat{w}^{(k)}\right)\right\| \leq \varepsilon$

En reprenant la fonction coût $\mathrm{J_{MCO}}$, $\mathrm{J_{MCO}}\left(\hat{w}^{(k)}\right) = \sum_{i=1}^{\mathrm{N}} \left(y_i - \hat{y}_i^{(k)}\right)^2$. Ainsi : $\dfrac{\partial \mathrm{J}\left(\hat{w}^{(k)}\right)}{\partial \hat{w}_j^{(k)}} = -2.\sum_{i=1}^{\mathrm{N}} x_{ij}.\left(y_i - \hat{y}_i^{(k)}\right)$

Pour accélérer le processus, il est possible de prendre une donnée au hasard au lieu d'effectuer le calcul sur les N données. Il s'agit alors de la méthode du gradient stochastique (*online*). Ainsi on pioche un $i \in [\![1;\mathrm{N}]\!]$, et $\forall j \in [\![0,n]\!]$: $\dfrac{\partial \mathrm{J}\left(\hat{w}^{(k)}\right)}{\partial \hat{w}_j^{(k)}} = -2.x_{ij}.\left(y_i - \hat{y}_i^{(k)}\right)$

2.2.5 Régularisation et autres calculs de perte

Afin d'éviter le sur-apprentissage (données d'entrainement trop proches du modèle), la régularisation permet d'imposer une contrainte supplémentaire en modifiant le calcul de la perte via l'ajout de coefficients de pénalité λ_i réglés pour obtenir de meilleures performances. On distingue ainsi :

- **la régression Ridge :** $J_{ridge}(\hat{w}) = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda_1 . \sum_{j=1}^{n} \hat{w}_i^2$ avec $\lambda_1 \geq 0$
- **la régression LASSO (*Least Absolute Shrinkage and Selection Operator*) :**
  $J_{LASSO}(\hat{w}) = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda_2 . \sum_{j=1}^{n} |\hat{w}_i|$ avec $\lambda_2 \geq 0$
- **la régression *elastic Net* :** $J_{\text{Elastic Net}}(\hat{w}) = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2 + \lambda_1 . \sum_{j=1}^{n} \hat{w}_i^2 + \lambda_2 . \sum_{j=1}^{n} |\hat{w}_i|$ avec $\lambda_1 \geq 0$ et $\lambda_2 \geq 0$

De façon à normer la perte ou simplifier les calculs en ne divisant pas par deux, on pose :

$$J = \frac{1}{2}.J_{MCO}(\hat{w}) = \frac{1}{2}.\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad \text{ou} \quad J = \frac{1}{2.N}.J_{MCO}(\hat{w}) = \frac{1}{2.N}.\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 = \frac{1}{2.N}.\sum_{i=1}^{N}(y_i - X_i.\hat{W})^2$$

2.3 Régression logistique et problème de classification binaire

2.3.1 Classification binaire

Dans le cas de la régression linéaire multiple la variable de sortie est continue (cf Partie 2.2.1). Dans celui de la régression logistique, la sortie est une variable discrète. L'algorithme de régression logistique fait donc partie des algorithmes de classification.

En classification binaire la sortie n'a que deux états possibles (par exemple 0 ou 1). On ajoute alors une frontière de décision qui permet de déterminer la sortie $\hat{y}_i$ en fonction de l'entrée z_i, z_i étant lié aux x_i.

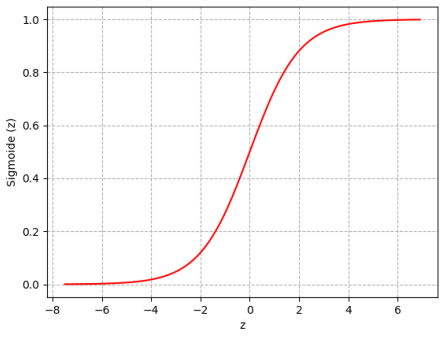
Exemple : fonction d'Heaviside : $\begin{cases} \hat{y}_i = 0 \text{ si } z_i < 0 \\ \hat{y}_i = 1 \text{ si } z_i \geq 0 \end{cases}$

2.3.2 Fonction logistique

La spécificité du problème posé précédemment amène à utiliser une fonction plus régulière que la fonction d'Heavide : on introduit alors la fonction logistique, aussi appelée fonction sigmoïdale ou sigmoïde S définie par :

$$\sigma(z) = \frac{1}{1+e^{-z}} \quad \text{avec} \quad \sigma'(z) = \sigma(z).(1-\sigma(z))$$

On remarque que cette fonction est toujours comprise entre 0 et 1.



Il existe d'autre fonctions de ce type, comme par exemple la fonction tangente hyperbolique. Dans la suite, on se focalisera sur la fonction logistique sigmoïde. A partir de cette fonction logistique, il est possible de définir une frontière ou seuil de décision, qui en règle générale est égal à 0,5.

$$\begin{cases} \hat{y}_i = 0 \text{ si } \sigma(z_i) < 0{,}5 \\ \hat{y}_i = 1 \text{ si } \sigma(z_i) \geq 0{,}5 \end{cases}$$

Avec des notations identiques à celles utilisées dans la Partie 2.2, on note que $z_i = w_0 + \sum_{j=1}^{n} w_j.x_{ij}$. Dans le cas de classification à une variable, $z_i = w_0 + w_1.x$.

Remarque : z_i correspond au y_i de la régression. C'est la phase d'agrégation. Or y étant associé à la sortie, donc après la phase d'activation, on garde y pour l'image de la fonction d'activation et on note z son antécédent.

2.3.3 Fonction perte associée à la régression logistique

La fonction perte J_{MCO} n'est pas adaptée à la fonction sigmoïde. En effet, il existe plusieurs minimum locaux ce qui peut bloquer la descente de gradient vers le minimum global. On construit donc une fonction perte compatible avec le modèle de régression logistique.

La nouvelle fonction perte s'appuie sur la fonction logarithme. Pour comprendre son fonctionnement, on sépare les cas où $y = 0$ et les cas où $y = 1$.

- **Cas où $y = 1$:** L'objectif est de pénaliser la machine si elle prédit une valeur proche de 0, alors que la sortie vaut 1. On pose $J(\hat{w}) = -\ln(\sigma(z))$
- **Cas où $y = 0$:** A l'inverse, cette fois-ci il faudra pénaliser la machine par une grande erreur si celle-ci prédit 1, alors que $y = 0$. On pose $J(\hat{w}) = -\ln(1-\sigma(z))$
- **Fonction perte associée au modèle de régression logistique :** en reprenant les deux approches :

$$J(\hat{w}) = -\sum_{i=1}^{N} (y_i.\ln(\hat{y}_i) + (1-y_i).\ln(1-\hat{y}_i))$$ ce qui donne $$J(\hat{w}) = -\sum_{i=1}^{N} (y_i.\ln(\sigma(z)) + (1-y_i).\ln(1-\sigma(z)))$$

2.3.4 Descente de gradient

Tout comme dans la Partie 2.2.4, la descente de gradient est une méthode itérative, avec η, le taux d'apprentissage et $\nabla J\left(\hat{w}^{(k)}\right)$, le gradient à l'itération k :

$$\hat{w}^{(k+1)} = \hat{w}^{(k)} - \eta.\nabla J\left(\hat{w}^{(k)}\right)$$

Pour déterminer le gradient, le plus *simple* est d'utiliser une dérivation en chaine : $\frac{\partial J(\hat{w})}{\partial \hat{w}_j} = \frac{dJ(\hat{w})}{dz}.\frac{\partial z}{\partial \hat{w}_j}$.

Or si $g(z) = \ln(\sigma(z))$ alors $g'(z) = \frac{\sigma'(z)}{\sigma(z)} = 1-\sigma(z)$ et si $g(z) = \ln(1-\sigma(z))$ alors $g'(z) = -\frac{\sigma'(z)}{1-\sigma(z)} = -\sigma(z)$ Ainsi :

$$\frac{dJ(\hat{w})}{dz_i} = -\sum_{i=1}^{N} (y_i.(1-\sigma(z_i)) + (1-y_i).(-\sigma(z_i))) = \sum_{i=1}^{N} (\sigma(z_i) - y_i) \text{ et } \forall j \geq 1 \ \frac{\partial z_i}{\partial \hat{w}_j} = x_{ij} \text{ puis pour } j = 0, \ \frac{\partial z_i}{\partial \hat{w}_0} = 1$$

$$\text{Ainsi } \forall j \geq 1 \ \frac{\partial J(\hat{w})}{\partial w_j} = \left(\sum_{i=1}^{N} (\sigma(z_i) - y_i).x_{ij}\right) \text{ et pour } j = 0 \ \frac{\partial J(\hat{w})}{\partial w_0} = \left(\sum_{i=1}^{N} (\sigma(z_i) - y_i)\right)$$

En posant $x_{i0} = 1$, on obtient $z_i = X_i.\hat{W}$, d'où $\nabla J(\hat{w}) = \mathbb{X}^T.\left(\Sigma\left(X_i.\hat{W}\right) - Y\right)$ avec $\Sigma_i = \sigma(z_i) = \sigma(X_i.\hat{W})$.

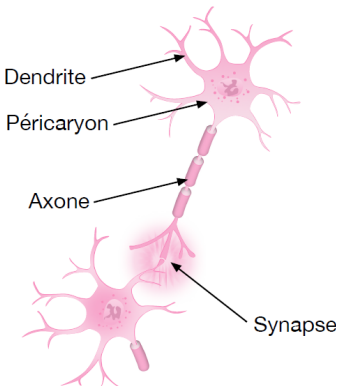
$$\hat{W}^{(k+1)} = \hat{W}^{(k)} - \eta.\mathbb{X}^T.\left(\Sigma\left(\mathbb{X}.\hat{W}^{(k)}\right) - Y\right)$$

3 Réseaux de neurones

3.1 Pourquoi un tel nom ?

Les réseaux de neurones artificiels doivent leur nom à leur imitation des réseaux de neurones du système nerveux humain. Un neurone est composé des 3 parties principales :

- **la dendrite :** son rôle est de capter les excitations d'autres axones de neurones adjacents, de former les potentiels postsynaptiques et de les transmettre au péricaryon. L'intensité du potentiel postsynaptique est liée à différentes propriétés et états des synapses.
- **le péricaryon :** son rôle est de combiner et additionner tous les potentiels postsynaptiques. Si le potentiel seuil est atteint alors le péricaryon génère un potentiel d'action.
- **l'axone :** son rôle est de transmettre le potentiel d'action à l'extrémité du neurone pour communiquer l'excitation aux neurones voisins via les synapse.



3.2 Le perceptron

Le perceptron (ou neurone artificiel) est créé pour imiter mathématiquement les principes de fonctionnement du neurone (Fig 5).

Il possède plusieurs entrées binaires $(x_j)_{j=1}^n$ (via les dendrites) et à chaque entrée x_j correspond un poids w_j (lié à la sensibilité des synapses). La somme de chaque entrée x_j multipliée par son poids w_j est ensuite comparée à un seuil $-w_0$, l'opposé du biais (jouant le rôle de potentiel de seuil).

La sortie s est égale à 1 si $z > 0$ (neurone actif). Sinon, la sortie est égale à 0 (neurone inhibé). Cette description correspond au cas où la fonction d'activation est la fonction d'Heaviside.

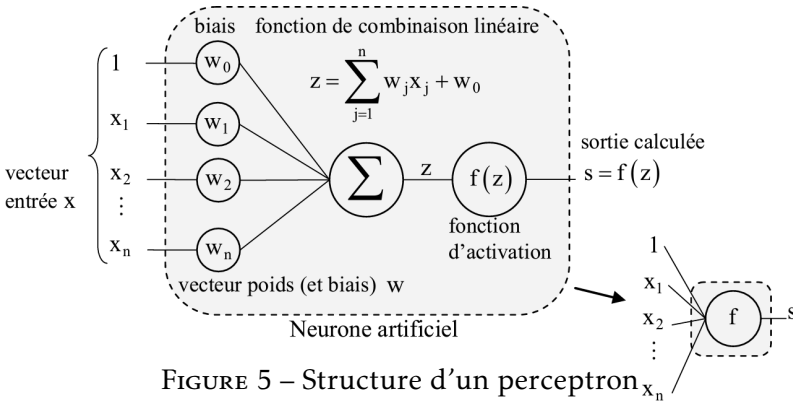
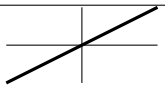
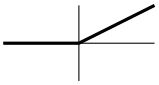
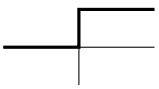


Figure 5 – Structure d'un perceptron

Fonction	Allure	Expression $f(z)$	Dérivée $f'(z)$	Étendue
Linéaire		$A.z$	A	$]-\infty,\infty[$
Unité de rectification linéaire (ReLU)		$\begin{cases} z \text{ si } z \leq 0 \\ 0 \text{ sinon} \end{cases}$	$\begin{cases} 1 \text{ si } z \leq 0 \\ 0 \text{ sinon} \end{cases}$	$]0,\infty[$
Heaviside		$\begin{cases} 1 \text{ si } z \leq 0 \\ 0 \text{ sinon} \end{cases}$	$\begin{cases} 0 \text{ si } z \neq 0 \\ ? \text{ sinon} \end{cases}$	$]0,1[$
Tangente hyperbolique		$\tanh(z)$	$1 - f(z)^2$	$]-1,1[$
Sigmoïde (logistique)		$\frac{1}{1-e^{-z}}$	f(z).(1-f(z))	$]0,1[$

Pour traiter un problème de réseau de neurones (RDN), on utilise une des méthodes de descente de gradient. La discontinuité de la fonction d'Heaviside pose problème. On utilise d'autres fonctions d'activation permettent d'introduire de la non-linéarité dans la façon de traiter les données : $s = f(z) = f\left(w_0 + \sum_{i=1}^{n} x_i.w_i\right)$.

La fonction sigmoïde est une fonction de type S couramment utilisée, qui décrit une bijection de $\mathbb{R}$ dans $[0,1]$. Lorsque l'entrée de la fonction sigmoïde est très grande ou très petite, la sortie entre dans la zone « plate » à faible gradient. A contrario, lorsque l'entrée est proche de 0, la valeur de la dérivée sigmoïde est plus grande. Par ailleurs, la fonction sigmoïde produit directement un état de demi-saturation (sortie = 0,5) lorsque le seuil d'activation est tout juste atteint (entrée = 0), ce qui ne correspond pas aux véritables réseaux neuronaux biologiques. En général, seuls 1 à 4% des neurones du cerveau sont actifs (sortie > 0) simultanément et ne sont pas facilement saturés.

La fonction unité linéaire rectifiée (ReLU) est plus conforme aux modèles physiologiques et est plus adaptée pour éviter les gradients qui explosent ou disparaissent. Elle peut également simplifier le calcul et accélérer la convergence car sa dérivée est toujours égale à 1 si l'entrée est positive.

3.3 Perceptron multicouche

Les réseaux de neurones (MPL pour *Multi-Layer Perceptron*) ont une fonction de combinaison linéaire et possèdent l'architecture donnée Fig 6 :

- **la couche d'entrée** comporte autant de neurones que de variables explicatives (dimension de x). La fonction d'activation est le plus souvent la fonction identité ($f(z) = z$).
- **les couches cachées** peut être plus ou moins nombreuses, constituées d'un nombre plus ou moins important de neurones.
- **la couche de sortie** comporte autant de neurons que de variables de sortie (sauf quelques cas particulier).

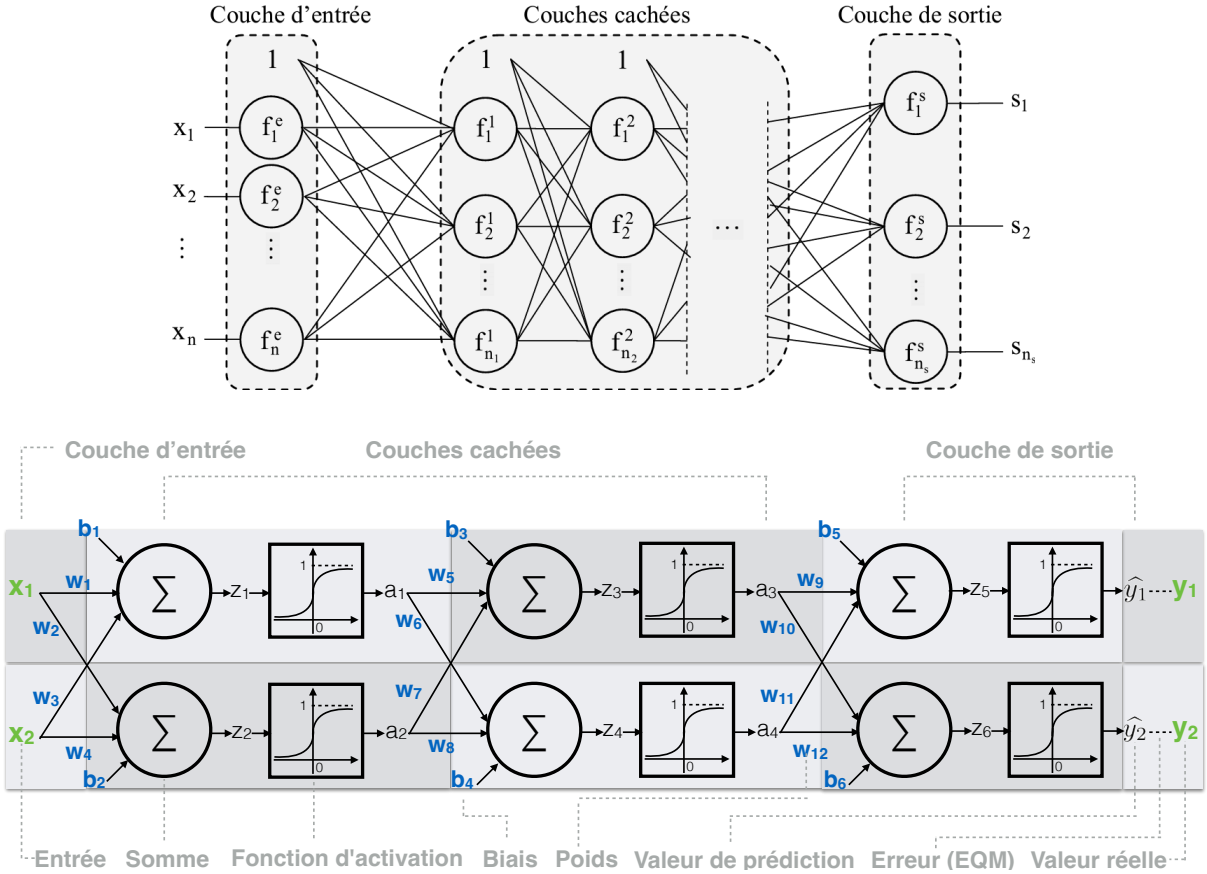


Figure 6 – Structure des réseaux de neurones

Remarques :

- Une combinaison de neurones à fonction de combinaison linéaire ($z = w_0 + \sum_{j=1}^{n} w_j.x_j$) permet de produire une modèle prédictif global non linéaire.
- Il est possible d'approximer toute fonction continue avec un réseau de neurones idoine.
- Ajouter des couches cachées ou des neurones cachés démultiplient le pouvoir explicatif du modèle de prédiction.

3.4 Fonction d'activation et fonction de perte

La forme générale d'une perte est : $J(\hat{w}) = \sum_{i=1}^{N} L(\hat{y}_i(x_i, \hat{w}), y_i) + \lambda.R(\hat{w})$ où L est la fonction de perte (cf Partie 2.2.2) et R, la fonction de régularisation (cf Partie 2.2.5).

Quel que soit le problème à résoudre, la fonction d'activation dans les couches cachées est le plus souvent la fonction ReLU car elle entraîne des calculs simples (cf Partie 3.2).

3.4.1 Réseau de neurones pour une régression linéaire

Dans le cas de la régression linéaire, les neurones de la couche de sortie ont une fonction d'activation linéaire.

Pour les moindres carrés, la fonction de perte est l'erreur au carrée : $L(\hat{y}_i, y_i) = (\hat{y}_i - y_i)^2$ ou $L(\hat{y}_i, y_i) = \frac{1}{2}.(\hat{y}_i - y_i)^2$.

3.4.2 Réseau de neurones pour un problème de classification binaire

Pour un problème de classification binaire, la fonction d'activation de l'unique neurone de la couche de sortie peut être la sigmoïde. En ce qui concerne la fonction de perte, on préfère l'entropie croisée binaire (*binary cross entropy*) :

$$L(\hat{y}_i, y_i) = -(y_i.\ln(\hat{y}_i) + (1 - y_i).\ln(1 - \hat{y}_i))$$

Cette fonction permet de calculer la « distance » d'une probabilité estimée $\hat{y}_i$ à la valeur attendue réelle y_i. Il est possible de montrer que les résultats sont meilleur en classification binaire qu'avec la méthode des moindres carrés.

3.4.3 Réseau de neurones pour un problème de classification à m catégories et multi-labels

Pour un problème de classification à m catégories et multi-labels, on utilise autant de neurones dans la couche de sortie que de catégories. Les sorties prédites $\hat{y}_i$ et y_i sont des vecteurs de dimension m. $\hat{y}_{ij}$ et y_{ij} en sont les $j$èmes composantes.

Pour tous les neurones de la couche de sortie, la fonction d'activation sigmoïde peut être utilisée, associée à une fonction de perte de type entropie croisée (*cross entropy*) : $L(\hat{y}_i, y_i) = -\sum_{j=1}^{m} (y_i.\ln(\hat{y}_i))$.

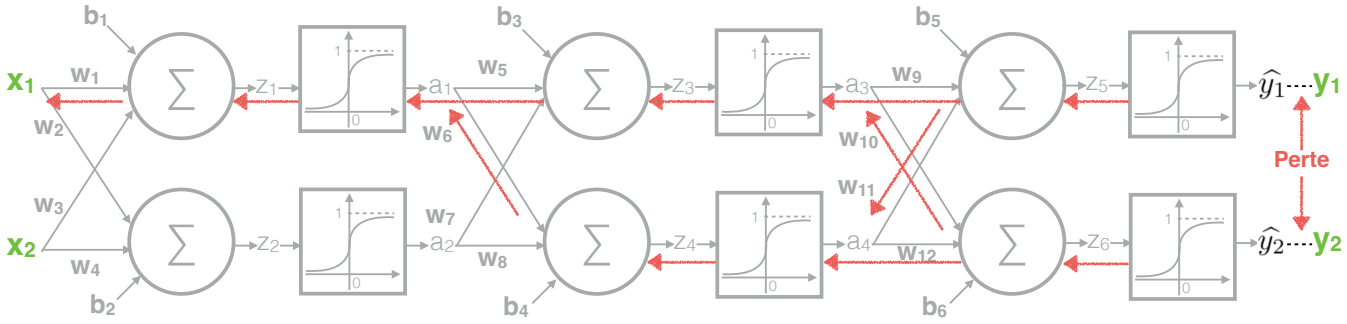
3.4.4 Réseau de neurones pour un problème de classification à m catégories et non multi-labels

Pour un problème de classification à m catégories et non multi-labels, l'unique neurone de la couche de sortie peut utiliser comme fonction d'activation la fonction logistique softmax. z et $f(z)$ sont alors des vecteurs de dimension m correspondant au nombre de catégories :

$$f(z)_j = \frac{e^{z_i}}{\sum_{i=1}^{m} e^{z_i}} \text{ où } f(z)_j \text{ est la probabilité d'appartenance à la catégorie } j$$

3.5 Détermination du modèle de prédiction

L'algorithme de rétro-programmation du gradient est souvent utilisé pour déterminer le modèle de prédiction.



- On commence par normaliser et standardiser les données.
- Les données sont mélangées avant le début de l'apprentissage.
- On fixe les valeurs des poids et biais du réseau de neurones avec des valeurs aléatoires
- On applique l'algorithme *online* :
  - Pour chaque donnée d'entrée :
    - ⋆ On calcul les sorties de chaque neurone couche après couche, en propageant les résultats vers l'avant (*forward*) jusqu'à la couche de sortie.
    - ⋆ On déterminer l'erreur de prédiction de la couche de sortie.
    - ⋆ On propage l'erreur de prédiction de la couche de sortie.
    - ⋆ On propage l'erreur en arrière (*back programmation*) jusqu'à la première couche cachée.
    - ⋆ On met à jour tous les poids de toutes les couches en minimisant la perte
  - Tant que la condition d'arrêt n'est pas atteint, on ré-effectue un cycle complet d'apprentissage (`epoch`) comme décrit précédemment.

3.6 Paramètre constant, hyperparamètre

Si la majorité des paramètre sont entraînables, il y a des paramètres constants non entraînables dont les valeurs sont fixées avant le processus d'apprentissage. Ils peuvent être définis comme des hyperparamètres et ont une influence cruciale sur les performances du modèle. Pour un réseau de neurones primaire, voici quelques hyperparamètres typiques :

- **Structure du réseau**
  Tels que le nombre de couches, le nombre de neurones par couche, le type de fonction d'activation, etc.
- **Paramètre d'optimisation**
  Tels que la méthode d'optimisation, le taux d'apprentissage, la taille du lot, ...

L'ajustement des hyperparamètres est un problème d'optimisation non linéaire, non différentiable et non convexe qui ne peut être résolu avec les méthodes d'optimisation classiques (par exemple, la descente de gradient mentionnée ci-dessus). Pour chaque combinaison d'hyperparamètre, un entrainement entier doit être effectué pour évaluer les performances.

Bibliographie

Cours de SI MP/PSI chez Ellipse - Cours de Xihe Ge, Ressources UPSTI et bien sûr Machine Learnia