

16 Introduction à l'apprentissage automatisé

16.1 L'Intelligence Artificielle, qu'est ce que c'est, à quoi ça sert?

16.1.1 Premières définitions – IA, données

Définition – Intelligence Artificielle

Wikipedia – L'intelligence artificielle (IA) est « l'ensemble des théories et des techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence ».

En première approche, l'IA peut être assimilée à un ensemble d'algorithmes permettant de réaliser des tris, des classifications.

Exemple –

SSD/MobileNet/COCO (Tensorflow/OpenCV)

person(1) motorcycle(1) donut(1)

motorcycle: 91.19% 34%

Identification d'objets, en temps réel, dans une vidéo.

On remarquera sur l'exemple de gauche que l'algorithme parvient à identifier sur l'image des zones puis à identifier à quoi elles correspondent. On remarquera aussi que la flamme est identifiée comme étant un donut...

[exploring-opencv-deep-learning-object-detection-library](#) & <https://scikit-learn.org/>.

Comparaison du résultats d'algorithmes de classification permettant de classer un nuage de points bleus et rouges de transparences différentes.

Pour utiliser un algorithme d'IA il est nécessaire de disposer de données... en général beaucoup de données.

16.1 L'Intelligence Artificielle, qu'est ce que c'est, à quoi ça sert?	1
16.2 Mécanismes d'apprentissages	5
16.3 Algorithmes d'apprentissage	8

- Analyser les principes d'intelligence artificielle.
 - Régression et classification, apprentissages supervisé et non supervisé.
 - Phases d'apprentissage et d'inférence.
 - Modèle linéaire monovariable ou multivariable.
- Interpréter et vérifier la cohérence des résultats obtenus expérimentalement, analytiquement :
 - Ordre de grandeur. Matrice de confusion (tableau de contingence), sensibilité et spécificité d'un test.
- Résoudre un problème en utilisant une solution d'intelligence artificielle :
 - Apprentissage supervisé.
 - Choix des données d'apprentissage.
 - Mise en œuvre des algorithmes (k plus proches voisins et régression linéaire multiple).
 - Phases d'apprentissage et d'inférence.

Définition – Données

Les données utilisées par un algorithme d'IA peuvent être sous différentes formes :

- ▶ des données quantitatives continues qui peuvent prendre n'importe quelles valeurs dans un ensemble de valeurs (par exemple la température) ;
- ▶ des données quantitatives discrètes qui peuvent prendre un nombre limité de valeurs dans un ensemble de valeurs (par exemple nombre de pièces d'un logement) ;
- ▶ des données qualitatives (ordinales ou nominales suivant qu'on puisse les classer ou non, par exemple des couleurs, des notes à un test d'opinion ...).

Pour pouvoir traiter ces données, il peut être nécessaire qu'elles soient organisées sous une certaine forme. On peut par exemple identifier :

- ▶ les données structurées, dans une base de données par exemple ;
- ▶ les données semi-structurées, dans un fichier csv, xml ou json par exemple ;
- ▶ les données non structurées comme un image, du texte, ou une vidéo.

Exemple –

Le site <https://www.kaggle.com/> partage des sets de données.

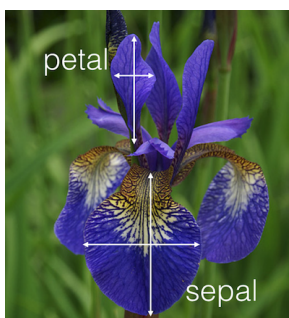
On peut par exemple retrouver un dataset semi-structuré au format JSON ou CSV des chansons disponibles sur Spotify, sorties entre 1922 et 2021. (<https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks>).

Le site <https://storage.googleapis.com/> propose des sets de photos étiquetées selon 600 classes.

Définition – Observations – Caractéristiques

On appelle observation (ou individu ou objet) une « ligne de donnée » qui va être utilisée par un algorithme d'apprentissage.

Une observation est composée de caractéristiques qui varient pour chacun des observations.

**Exemple –**

Dans le cas de la base de données des Iris, présente par exemple dans `scikit-learn`, les données sont composées de 150 observations, chacune composée de 4 caractéristiques : longueur et largeur du sépale ainsi que longueur et largeur du pétale.

Définition – Big data – Wikipedia

Mégadonnées ou données massives. Le big data désigne les ressources d'informations dont les caractéristiques en termes de volume, de vitesse et de variété imposent l'utilisation de technologies et de méthodes analytiques particulières pour générer de la valeur, et qui dépassent en général les capacités d'une seule et unique machine et nécessitent des traitements parallélisés.

Que fait-on avec ces données ? En général les algorithmes vont chercher un lien entre ces données. Dans le cas où il existe un lien connu par le *Data Scientist* on parle d'apprentissage supervisé.

Si ce lien n'est pas connu, on parle d'apprentissage non supervisé ou de clustering.

16.1.2 Apprentissage automatisé – Machine Learning

Définition – Apprentissage automatique – Machine learning

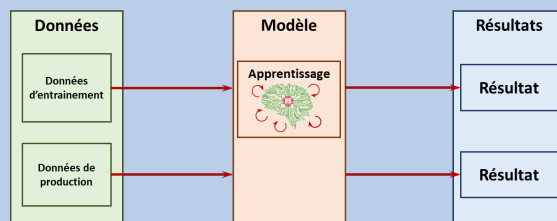
L'apprentissage automatique est un champ de l'intelligence artificielle dont l'objectif est d'analyser un grand volume de données afin de déterminer des motifs et de réaliser un modèle prédictif.

L'apprentissage comprend deux phases :

- l'entraînement (ou apprentissage) est une phase d'estimation du modèle à partir de données d'observations ;
- la mise en production du modèle (inférence) est une phase pendant laquelle de nouvelles données sont traitées dans le but d'obtenir le résultat souhaité.

L'entraînement peut être poursuivi même en phase de production.

Exemple –



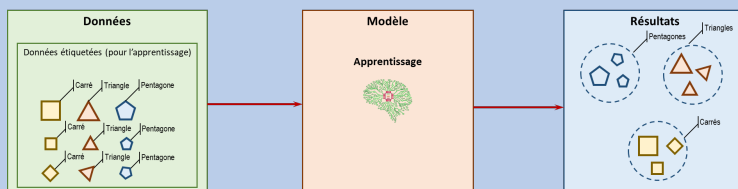
Définition – Apprentissage supervisé – Apprentissage

Tâche d'apprentissage au cours duquel l'algorithme (ou fonction de prédiction) va, à partir d'un ensemble de données **étiquetées**, déterminer un lien entre un ensemble les données et les étiquettes.

Définition – Apprentissage supervisé – Inférence

Tâche au cours duquel l'algorithme (ou fonction de prédiction) va, à partir d'un ensemble de données **non étiquetées**, prédire l'étiquette.

Exemple –



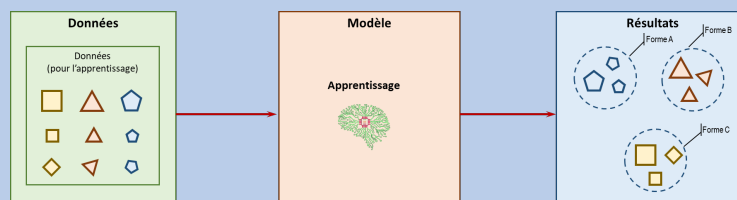
Exemple –

Soit un ensemble d'images en noir et blanc représentant des chiffres de 0 à 9. L'algorithme doit dans un premier temps *apprendre* le lien entre l'image et le chiffre. Dans un second temps, l'algorithme devra déterminer le chiffre en fonction d'une image seule.

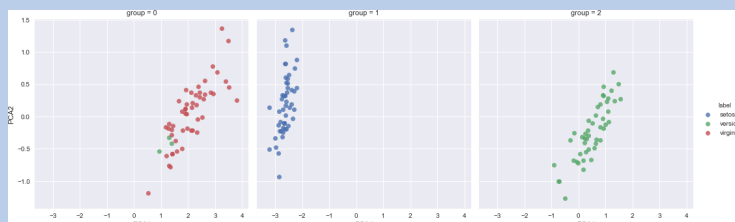
Définition – Apprentissage non supervisé – Clustering

Tâche d'apprentissage au cours duquel l'algorithme (ou fonction de prédiction) va, à partir d'un ensemble de données **non étiquetées**, déterminer un lien entre les données (et les regrouper).

Exemple –



Classification d'iris en utilisant un algorithme d'apprentissage non supervisé.

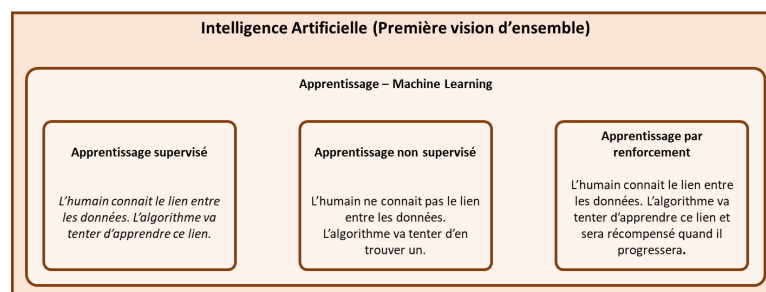


On remarque (lorsque l'image est en couleur) que sur des mesures sur 150 iris, l'algorithme a réussi à retrouver les 3 différentes espèces, sans les connaître, à 3 erreurs près.

Définition – Apprentissage par renforcement

Si au cours de l'apprentissage supervisé, un mécanisme de récompense est mis en œuvre pour améliorer les performances du modèle, on parle d'apprentissage par renforcement.

On peut donc faire une synthèse des méthodes d'apprentissage dont nous avons pris connaissance (il en existe d'autres).



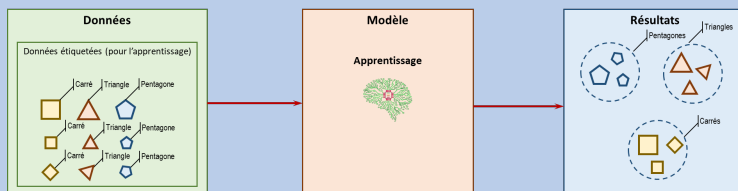
Comment situer le deep learning parmi ces méthodes d'apprentissage ? – Pour moi, l'utilisation du terme « deep learning » apparaît lorsqu'on commence à utiliser des réseaux de neurones... mais les réseaux de neurones peuvent être utilisés dans chacune des méthodes d'apprentissage sus-citées...

16.1.3 Encore des définitions...

Définition – Classification

En apprentissage automatique, on appelle problème de classification les problèmes où les étiquettes sont discrètes.

Exemple –



Dans le cas où nous souhaitons regrouper par forme, il s'agit d'un problème de classification. Les classes discrètes sont les formes (triangles, carrés, pentagones ...). Il serait aussi possible d'opérer un regroupement par couleur. Les classes seraient donc différentes.

Définition – Régression

En apprentissage automatique, on appelle problème de régression les problèmes où les étiquettes sont continues.

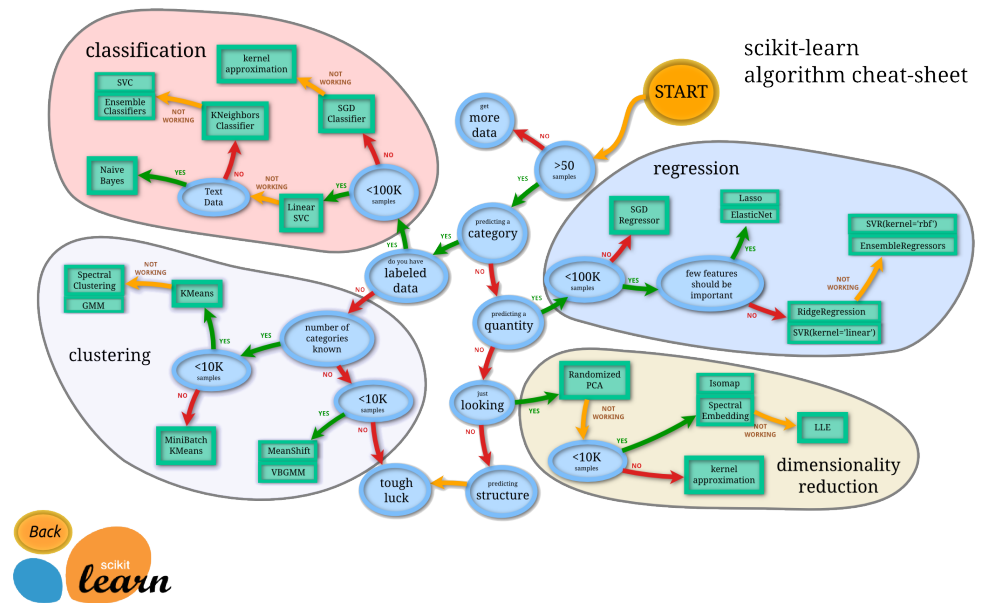
Exemple –

Dans le cas où nous souhaiterions que l'algorithme détermine l'aire ou le périmètre de la forme, les étiquettes seraient alors continues sur \mathbb{R}_*^+ .

16.2 Mécanismes d'apprentissages

16.2.1 Classification des algorithmes d'apprentissage

La bibliothèque `scikit-learn` propose une classification de divers algorithmes en fonction des apprentissages automatiques (hors réseaux de neurones).



16.2.2 Validation du modèle

Une fois un algorithme d'apprentissage choisi, on se pose la question de la validation du modèle. Quels critères et outils vont nous permettre de considérer que notre apprentissage est « bon » ?

Lors d'un problème de classification, il est par exemple possible de déterminer les écarts entre les valeurs prédites par l'algorithme et les valeurs cibles.

Critères de validation des problèmes de classification

Définition – Valeur prédictive positive

Valeur prédictive positive (*accuracy classification score*) :

$$\text{Justesse} = \frac{\text{Nombre de prédictions vraies}}{\text{Nombre de prédictions totales}}.$$

```
1 import sklearn.metrics as skm
2
3 # X(numpy.ndarray) : données d'entrées
4 # Y(numpy.ndarray) : données cibles correspondantes
5 # Y_pred(numpy.ndarray) : données prédites par l'algorithme de classification
6
7 print(skm.accuracy_score(Y, Y_pred))
```

Définition – Matrices de confusion

La matrice de confusion est une métrique permettant de déterminer la qualité d'une classification. En abscisses sont indiquées les valeurs réelles, et en ordonnées les valeurs prédites.

Exemple –

Dans l'exemple ci-contre, nous cherchons à classer des iris, grâce à un algorithme, selon 3 familles : les setosa, les versicolor et les virginica. Sur la diagonale, sont retrouvées les iris dont l'espèce a été correctement prédite. En revanche, 3 versicolor ont été classées parmi les virginica et 2 virginica ont été classifiées dans les versicolor.

Remarque

La matrice de confusion présentée est dite non-normalisée. Si la répartition des étiquettes n'est pas uniforme (en fonction des classes), il est probable qu'il y ait davantage d'erreurs pour les classes ayant beaucoup d'étiquettes. Pour palier ce problème on peut utiliser des matrices de confusions normalisées (on divise alors chaque terme de la matrice par la somme des éléments de la même ligne).

	setosa	versicolor	virginica
setosa	50	0	0
versicolor	0	47	3
virginica	0	2	48
	setosa	versicolor	virginica

Valeurs réelles

Valeurs prédites

Critères de validation des problèmes de régression**Définition – Erreur moyenne quadratique**

Appelée également *mean squared error*, il s'agit d'une moyenne d'écart au carré :

$$msq = \frac{1}{N} \sum_{i=1}^n \|Y_i - \hat{Y}_i\|^2$$

en notant Y_i la valeur réelle (étiquetée) et \hat{Y}_i la valeur estimée par l'algorithme.

16.2.3 Séparation des données/Gestion des données?

Lorsque l'on souhaite disposer d'un modèle défini par un algorithme d'IA, il faut en premier lieu disposer de données.

Types de données – Apprentissage supervisé

En apprentissage supervisé, il est nécessaire de connaître des données d'entrées ainsi que les sorties correspondantes (appelées aussi cibles ou target).

Dans le cadre d'une programmation Python avec la bibliothèque `scikit-learn` les données d'entrées et de sorties peuvent être implémentées en utilisant des `numpy.ndarray`.

Ainsi, si les données d'entrées, stockées dans la variable `data` sont composées de n observations elles mêmes composées de p catégories, le `shape` de `data` sera (n, p) .

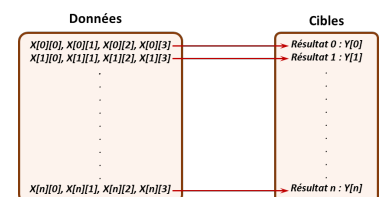
La donnée cible sera quant à elle un vecteur de n valeurs.

Exemple –

Données d'entrées : matrice X de n observations avec 4 caractéristiques.

Données de sortie : vecteur Y de n résultats.

Dans le cadre d'une classification, où r résultats sont possibles. On peut attribuer une valeur (entière) entre 1 et r à chacun des résultats, notamment si ceux-ci sont des données qualitatives.



1: <https://scikit-learn.org/stable/modules/preprocessing.html>

Normalisation des données¹

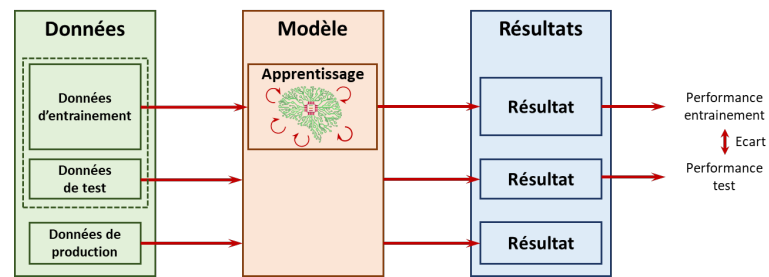
Séparation des données

Lors du démarrage d'un apprentissage supervisé, on dispose d'un jeu de données comprenant les données d'entrée et les cibles correspondantes. Il est d'usage de séparer ces données en deux parties :

- les données d'entraînement permettant... d'entraîner le modèle. Dans la pratique on utilise entre 60 et 80 % des données de base ;
- les données de test, permettant de valider l'apprentissage (ou le modèle), dans la pratique entre 20 et 40 % des données de base.

On commence donc par réaliser un apprentissage sur les données d'entraînement. Cet entraînement produit des résultats. Connaissant les cibles correspondant aux données d'entraînement, on peut donc en déduire une performance du modèle sur le traitement des données d'entraînement.

On utilise alors le modèle en lui donnant les données de test. De même, on peut donc en déduire une performance du modèle sur le traitement des données de test.



Se pose alors le problème de comment séparer les données. En effet, les données de base pouvant potentiellement être triées (ordonnées par rapport à une des caractéristiques), et sélectionner une partie pourrait créer un biais dans l'apprentissage.

scikit-learn permet de séparer aléatoirement des données en fixant le pourcentage de données de validation.

```
1 from sklearn.model_selection import train_test_split
2 # X(numpy.ndarray) : données d'entrées
3 # Y(numpy.ndarray) : données cibles correspondantes
4 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33)
```

On perçoit donc que la qualité de l'apprentissage peut dépendre du choix utilisé lors de la séparation des données. De plus, certains choix de paramètres permettant d'affiner le modèle sont aussi liés aux données d'entraînement sélectionnées. Une des solutions pour résoudre ce problème est d'avoir recours à la validation croisée².

2: https://scikit-learn.org/stable/modules/cross_validation.html

16.3 Algorithmes d'apprentissage

16.3.1 Algorithme des k plus proches voisins

Présentation

L'algorithme des k plus proches voisins (auss appelé k -nearest neighbors algorithm – k -NN) permet de réaliser des opérations de régression et de classification sur un ensemble de données.

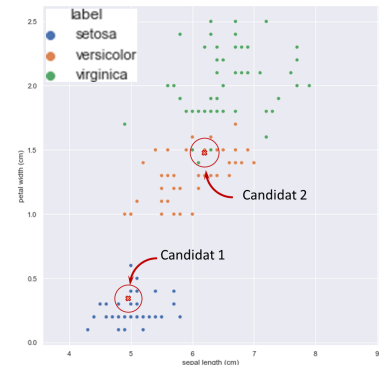
Pour une première approche, cette méthode permet d'estimer la classe d'une nouvelle donnée en déterminant la norme entre cette nouvelle donnée et l'ensemble des données du jeu d'entraînement. Parmi les k plus proches voisins, on recherche la classe majoritaire et on attribue cette classe à la nouvelle donnée.

Exemple –

Dans le cas ci-contre, on cherche à classer les iris selon 3 catégories (setosa, versicolor et virginica). Les données étiquetées sont représentées dans le plan longueur du sépale en abscisse et largeur du pétale en ordonnée.

Soient les candidats 1 et 2, deux iris dont on connaît les caractéristiques mais pas l'espèce. Les 5 plus proches voisins du candidat 1 sont des setosa. Il est donc probable que le candidat 1 soit un setosa aussi.

Concernant le candidat 2, parmi les 5 plus proches voisins, 3 sont des virginica et 2 sont des versicolor. On peut faire l'hypothèse que c'est un virginica.



L'algorithme

Prenons comme exemple des données sous la forme $[x, y, c]$ où x et y représentent des coordonnées et c la classe d'appartenance.

Première étape – Calculer la distance Il faut tout d'abord définir une fonction de distance. On peut par exemple utiliser la distance euclidienne. Pour deux vecteurs x_1

et x_2 de taille N , $d = \sqrt{\sum_{i=1}^n (x_{1,i} - x_{2,i})^2}$.

```
1 import math as m
2 def distance(x1:list, x2:list) -> float :
3     distance = 0.
4     for i in range(len(x1)):
5         distance += (x1[i]-x2[i])**2
6     return m.sqrt(distance)
```

Deuxième étape – Détermination des plus proches voisins Pour cette étape, on dispose de données pour l'entraînement `data_train` et d'une donnée à tester `data_test`. Il va falloir calculer la distance entre la donnée à tester et les données d'entraînement puis trier les données.

Enfin, on retournera les k lignes les plus proches de `data_test`.

```
1 def get_voisins(data_train:list, data_test, k:int) -> list :
2     distances = []
3     for ligne in data_train :
4         d = distance(ligne,data_test)
5         distances.append([ligne,d])
6     # Tri de la liste suivant la seconde colonne (distances)
7     distances.sort(key=lambda t : t[1])
8     voisins = []
9     for i in range(k) :
10         voisins.append(distances[i][0])
11     return voisins
```

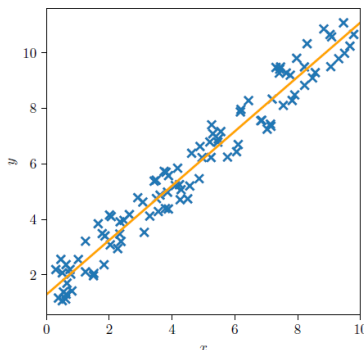
Troisième étape – Prédiction Une fois qu'on connaît les k plus proches voisins, on regarde quelle est la classe majoritaire parmi ces voisins.

```
1 def prediction(data_train:list, data_test, k:int) -> list :
2     voisins = get_voisins(data_train, data_test, k)
3     sorties = [ligne[-1] for ligne in voisins]
4     predict = max(set(sorties), key=sorties.count)
5     return predict
```

Prédiction avec scikit-learn Un exemple rapide pour réaliser une classification k -NN avec scikit-learn.

```
1 import numpy as np
2 from sklearn import datasets
3 from sklearn.neighbors import KNeighborsClassifier
4
5 iris_X, iris_y = datasets.load_iris(return_X_y=True)
6
7
8 # A random permutation, to split the data randomly
9 np.random.seed(0)
10
11 # Chargement des données et séparation des données
12 indices = np.random.permutation(len(iris_X))
13 iris_X_train = iris_X[indices[:-10]]
14 iris_y_train = iris_y[indices[:-10]]
15 iris_X_test = iris_X[indices[-10:]]
16 iris_y_test = iris_y[indices[-10:]]
17
18 # Création du classifieur
19 knn = KNeighborsClassifier() # KNeighborsClassifier(n_neighbors=k) pour
    choisir k
20 # Entraînement du classifieur
21 knn.fit(iris_X_train, iris_y_train)
22 # Prédiction sur le jeu de test (à comparer avec iris_y_test)
23 knn.predict(iris_X_test)
```

3: Chloé-Agathe Azencott



16.3.2 Régression univariée ³

Dans le cadre de la régression linéaire univariée, on dispose de n observations : $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_i, y_i), \dots, (x_n, y_n)\}$ et des étiquettes y_i associées. La régression linéaire consiste à choisir une fonction de prédiction f de la forme, $\forall x \in \mathbb{R}, f(x) = ax + b$ avec $(a, b) \in \mathbb{R}^2$. Il va donc falloir minimiser l'erreur entre les prédictions et la base de données d'entraînement. La minimisation se note :

$$(\hat{a}, \hat{b}) = \arg \min_{(a,b) \in \mathbb{R}^2} \frac{1}{n} \sum_{i=1}^n (y_i - (ax_i + b))^2.$$

(Méthode des moindres carrés)

On montre que si $\sum_{i=1}^n x_i^2 \neq \left(\sum_{i=1}^n x_i\right)^2$ le problème a une unique solution. Sinon le système est indéterminé et il y a une infinité de solutions.

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 from sklearn import datasets, linear_model
4 from sklearn.metrics import mean_squared_error, r2_score
5
6 # Load the diabetes dataset
7 diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)
8
9 # Use only one feature
10 diabetes_X = diabetes_X[:, np.newaxis, 2]
11
12 # Split the data into training/testing sets
13 diabetes_X_train = diabetes_X[:-20]
14 diabetes_X_test = diabetes_X[-20:]
15
16 # Split the targets into training/testing sets
17 diabetes_y_train = diabetes_y[:-20]
18 diabetes_y_test = diabetes_y[-20:]
19
20 # Create linear regression object
21 regr = linear_model.LinearRegression()
22
23 # Train the model using the training sets
24 regr.fit(diabetes_X_train, diabetes_y_train)
25
26 # Make predictions using the testing set
27 diabetes_y_pred = regr.predict(diabetes_X_test)
28
29 # The coefficients
30 print('Coefficients: \n', regr.coef_)
31 # The mean squared error
32 print('Mean squared error: %.2f'
33       % mean_squared_error(diabetes_y_test, diabetes_y_pred))
34 # The coefficient of determination: 1 is perfect prediction
35 print('Coefficient of determination: %.2f'
36       % r2_score(diabetes_y_test, diabetes_y_pred))
37
38 # Plot outputs
39 plt.scatter(diabetes_X_test, diabetes_y_test, color='black')
40 plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)
41
42 plt.xticks(())
43 plt.yticks(())
44
45 plt.show()

```

Bibliographie

- 1 Éric Biernat et Michel Lutz. *Data science : fondamentaux et études de cas*. Eyrolles.



17 Introduction aux réseaux de neurones

05 NUM

17.1 Introduction

17.1.1 Bref historique

Dans les années 1940, les chercheurs tentent de fabriquer une machine capable d'apprendre à partir de données fournies, de mémoriser des informations et de traiter des informations incomplètes. Pour cela, ils essayent de réaliser des modèles mathématiques de neurones biologiques. S'en suit alors la naissance des premiers modèles de perceptrons.

L'apprentissage automatisé va alors connaître des hauts et des bas, au gré des avancées scientifiques et technologiques. Dans les années 1960, un des premiers coups d'arrêt fût provoqué par la non-capacité des réseaux de neurones à traiter des problèmes non linéaires. Dans les années 1980, la rétropropagation du gradient fut proposée. Mais devant le manque de capacité des ordinateurs, la recherche marquât un second coup d'arrêt. Dans les années 1990/2000, l'apparition des réseaux convolutifs et leur capacité à analyser les données des images relançât alors les recherches dans ce domaine.

17.1.2 Exemples d'applications des réseaux de neurones

<https://fr.wikiversity.org/>

- ▶ Banque : prêts et scoring.
- ▶ Cartes de crédit : détection des fraudes.
- ▶ Finance : analyse d'investissements et de fluctuations des taux de change.
- ▶ Assurance : couverture assurantielle et estimation des réserves.
- ▶ Marketing : ciblage des prospections, mesures et comparaisons des campagnes et des méthodes.
- ▶ Archéologie : identification et datation de fossiles et d'ossements.
- ▶ Défense : identification de cibles.
- ▶ Environnement : prévisions de la qualité de l'air et de l'eau.
- ▶ Production : contrôles qualité.
- ▶ Médecine : diagnostics médicaux.
- ▶ Energies : estimations des réserves, prévisions de prix.
- ▶ Pharmacie : efficacité de nouveaux médicaments.
- ▶ Psychologie : prévisions comportementales.
- ▶ Immobilier : études de marchés.

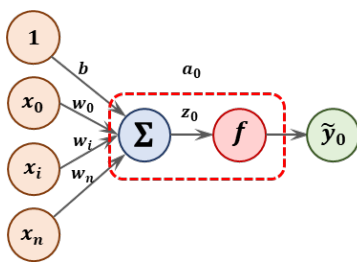
17.1	Introduction	13
17.2	Le neurone, les réseaux de neurones	14
17.3	Réseaux de neurones	15
17.4	Pour aller plus loin...	20

- ▶ Analyser les principes d'intelligence artificielle.
 - Phases d'apprentissage et d'inférence.
 - Réseaux de neurones (couches d'entrée, cachées et de sortie, neurones, biais, poids et fonction d'activation).
- ▶ 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).
 - Phases d'apprentissage et d'inférence.

- Recherche scientifique : identification de spécimens, séquençages de protéines.
- Télécommunication : détection des pannes de réseaux.
- Transport : maintenance des voies.

17.2 Le neurone, les réseaux de neurones

17.2.1 Modèle de neurone



Définition – Neurone (ou perceptron)

Prenons la représentation suivante pour un neurone. On note :

- \mathbf{X} le vecteur d'entrée constitué des données x_i . Ce vecteur constitue la couche d'entrée ;
- w_i les poids (poids synaptiques) ;
- b le biais ;
- z_0 la somme pondérée des entrées ;
- f une fonction d'activation ;
- \tilde{y}_0 : la valeur de sortie du neurone.

On a donc, dans un premier temps :

$$z_0 = b + \sum_{i=0}^n w_i x_i.$$

Après la fonction d'activation, on a donc en sortie du neurone :

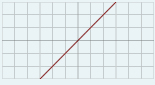
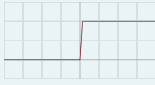


$$\tilde{y}_0 = a_0 = f(z_0) = f\left(b + \sum_{i=0}^n w_i x_i\right).$$

Remarque

1. La notation tilde (\tilde{y}_0) vient du fait que la valeur de sortie d'une neurone est une valeur estimée qu'il faudra comparer à y_0 valeur de l'étiquette utilisée pour l'apprentissage supervisé.
2. Par la suite, dans la représentation graphique on ne fera apparaître ni la somme pondérée ni la fonction d'activation, mais seulement la valeur de sortie du neurone (notée par exemple a_0).

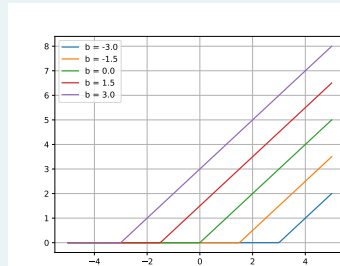
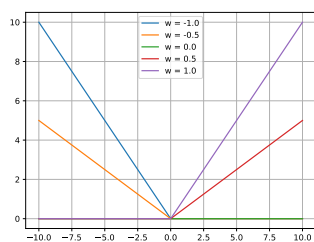
Définition – Fonction d'activation

Les fonctions d'activation sont des fonctions mathématiques appliquées au signal de sortie (z). Il est alors possible d'ajouter des non linéarités à la somme pondérée. On donne ci-dessous quelques fonctions usuelles :

Identité	Heaviside	Logistique (sigmoïde)	Unité de rectification linéaire (ReLU)
			
$f(x) = x$	$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$	$f(x) = \frac{1}{1 + e^{-x}}$	$f(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases}$

Remarque

Influence des poids et des biais sur la sortie du perceptron en utilisant une fonction d'activation ReLU.



On peut ainsi voir qu'avec la fonction d'activation ReLU, plus le poids sera grand en valeur absolue, plus le neurone amplifiera le signal d'entrée.

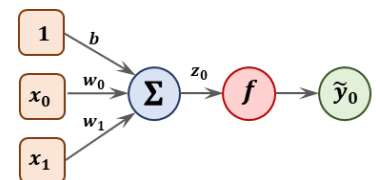
Le biais permettra de prendre en compte le « niveau » du signal d'entrée à partir duquel, le signal doit être amplifié, ou non.

Exemple –

Prenons un neurone à deux entrées binaires. Initialisation les poids et le biais avec des valeurs aléatoires : $w_0 = -0,3$, $w_1 = 0,8$ et $b = 0,2$.

On peut donc évaluer l'ensemble des sorties calculable par le neurone.

x_0	x_1	z	Id.	H.	Sig.	ReLu
0	0	0,2	0,2	1	0.549	0,2
0	1	1	1	1	0.731	1
1	0	-0.1	-0.1	0	0.475	0
1	1	0.7	0.7	1	0.668	0.7



17.3 Réseaux de neurones

17.3.1 Modélisation d'un réseau de neurones

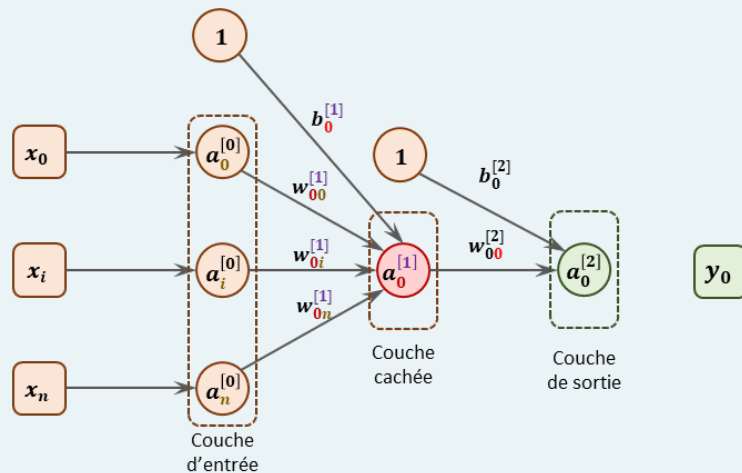
Définition – Couches

Un réseau de neurones est un ensemble de neurones reliés, par couches, entre eux. Dans un réseau de neurones **dense** tous les neurones de la couche i sont reliés à tous les neurones de la couche $i + 1$.

- Couche d'entrée : cette couche est une copie de l'ensemble des données d'entrées. Le nombre de neurones de cette couche correspond donc aux nombre de données d'entrées. On note $\mathbf{X} = (x_0, \dots, x_n)$ le vecteur d'entrées.

<https://playground.tensorflow.org/>

- Couche cachée (ou couche intermédiaire) : il s'agit d'une couche qui a une utilité intrinsèque au réseau de neurones. Ajouter des neurones dans cette couche (ou ces couches) permet donc d'ajouter de nouveaux paramètres. Pour une couche, la même fonction d'activation est utilisée pour tous les neurones. En revanche la fonction d'activation utilisée peut être différente pour deux couches différentes. Les fonctions d'activations des couches intermédiaires sont souvent non linéaires.
- Couche de sortie : le nombre de neurones de cette couche correspond au nombre de sorties attendues. La fonction d'activation de la couche de sortie est souvent linéaire. On note $\mathbf{Y} = (y_0, \dots, y_p)$ le vecteur des sorties.



En utilisant la loi de comportement du modèle de perceptron, on peut donc exprimer $\mathbf{Y} = \mathcal{F}(\mathbf{X})$ où \mathcal{F} est une fonction dépendant des entrées, des poids et des biais.

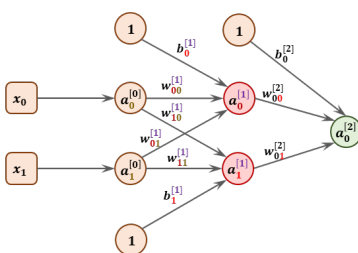
Notations :

- on note $w_{jk}^{[\ell]}$ les poids permettant d'aller vers la couche ℓ depuis le neurone k vers le neurone j ;
- $b_j^{[\ell]}$ le biais permettant d'aller sur le neurone j de la couche ℓ ;
- $f^{[\ell]}$ la fonction d'activation de la couche ℓ ;
- $n^{[\ell]}$ le nombre de neurones de la couche ℓ .

Définition – Équation de propagation

Pour chacun des neurones $a_j^{[\ell]}$ on peut donc écrire l'équation de propagation qui lui est associé :

$$a_j^{[\ell]} = f^{[\ell]} \left(\sum_{k=0}^{n^{[\ell-1]}} (w_{jk}^{[\ell]} a_k^{[\ell-1]}) + b_j^{[\ell]} \right) = f^{[\ell]} (z_j^{[\ell]}).$$



Exemple –

Prenons un réseau de neurones à 3 couches :

- 1 couche d'entrée à 2 neurones ;
- 1 couche cachée à 2 neurones, de fonction d'activation f_1 ;
- 1 couche de sortie à 1 neurone, de fonction d'activation f_2 ;

Initialisation les poids et le biais avec des valeurs aléatoires : $w_0 = -0,3, w_1 = 0,8$

et $b = 0, 2$.

Il est possible d'écrire que $y_0 = a_0^{[2]} = f_2 \left(b_0^{[2]} + w_{00}^{[2]} a_0^{[1]} + w_{01}^{[2]} a_1^{[1]} \right)$.

Par ailleurs : $a_0^{[1]} = f_1 \left(b_0^{[1]} + w_{00}^{[1]} a_0^{[0]} + w_{01}^{[1]} a_1^{[0]} \right)$ et $a_1^{[1]} = f_1 \left(b_1^{[1]} + w_{10}^{[1]} a_0^{[0]} + w_{11}^{[1]} a_1^{[0]} \right)$.

Au final, on a donc

$$y_0 = a_0^{[2]} = f_2 \left(b_0^{[2]} + w_{00}^{[2]} \left(f_1 \left(b_0^{[1]} + w_{00}^{[1]} a_0^{[0]} + w_{01}^{[1]} a_1^{[0]} \right) \right) + w_{01}^{[2]} \left(f_1 \left(b_1^{[1]} + w_{10}^{[1]} a_0^{[0]} + w_{11}^{[1]} a_1^{[0]} \right) \right) \right)$$

Définition – Paramètres

Les paramètres du réseau de neurones sont les poids et les biais, autant de valeurs que l'entraînement devra déterminer.

Méthode –

Calcul du nombre de paramètres – à vérifier

Soit un jeu de données étiquetées avec n entrées et p sorties.

On construit un réseau possédant ℓ couches et a_ℓ le nombre de neurones de la couche ℓ . Dans ce cas, la première couche est la couche d'entrée ($a_1 = n$) et la dernière couche est la couche de sortie ($a_\ell = p$).

Nombre de poids : $n_w = \sum_{i=1}^{\ell-1} (a_i \times a_{i+1})$.

Nombre de biais : $n_b = \sum_{i=2}^{\ell} (a_i)$.

Au final, le nombre total de paramètre à calculer est donné par $N = n_w + n_b$.

Objectif

Soit un jeu de données étiquetées. On note \mathbf{X} le vecteur des données d'entrées. On note \mathbf{Y} le vecteur des données de sorties. On note $\hat{\mathbf{Y}}$ le vecteur de sortie calculé par le réseau de neurones.

L'objectif de la phase d'apprentissage du réseau de neurones est de déterminer les valeurs de l'ensemble des poids et des biais de telle sorte que l'écart entre \mathbf{Y} et $\hat{\mathbf{Y}}$ soit minimal.

17.3.2 Fonction de coût

Dans le but de minimiser l'écart entre la sortie du réseau de neurones et la valeur réelle de la sortie, on utilise une fonction de coût (ou fonction de perte). Il est possible de définir plusieurs types de fonctions, notamment en fonction du type de problème à traiter (classification ou régression par exemple).

Définition – Fonction coût régression

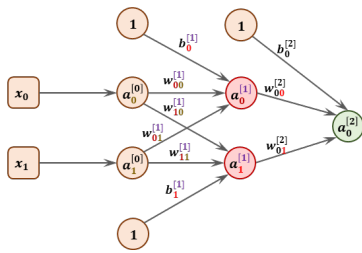
Notons nb le nombre de données dans la base d'entraînement. Dans le cadre d'un problème de régression, on peut définir la fonction coût comme la moyenne des erreurs quadratique entre la valeur donnée par l'équation de propagation et la

valeur de l'étiquette :

$$C = \frac{1}{nb} \sum_{i=1}^{nb} (Y_i - \mathbf{Y}_i)^2$$

Objectif

L'objectif est dès lors de déterminer les poids et les biais qui minimisent la fonction coût.



17.3.3 Notion de rétropropagation – Descente de gradient

En réutilisant l'exemple ci-contre, nous allons présenter succinctement comment est minimisée la fonction coût. Pour cela, il va falloir dériver la fonction coût par rapport à chacune des variables.

Cherchons uniquement à déterminer le coût par rapport à un seul vecteur d'entrée du jeu d'entraînement. On a alors :

- $C = (\mathbf{Y}_i - \mathbf{Y}_i)^2 = (a_0^{[2]} - y)^2$;
- $a_0^{[2]} = f^{[2]}(z_0^{[2]})$;
- $z_0^{[2]} = \sum_{k=0}^1 (w_{0k}^{[2]} a_k^{[1]}) + b_0^{[2]}$.

Commençons par déterminer la dérivée partielle par rapport à un poids de la couche de sortie :

$$\frac{\partial C}{\partial w_{00}^{[2]}} = \frac{\partial C}{\partial a_0^{[2]}} \frac{\partial a_0^{[2]}}{\partial z_0^{[2]}} \frac{\partial z_0^{[2]}}{\partial w_{00}^{[2]}}.$$

De même, on peut calculer la dérivée partielle du coût par rapport au biais : $\frac{\partial C}{\partial b_0^{[2]}} =$

$$\frac{\partial C}{\partial a_0^{[2]}} \frac{\partial a_0^{[2]}}{\partial z_0^{[2]}} \frac{\partial z_0^{[2]}}{\partial b_0^{[2]}}.$$

Calculons les dérivées nécessaires :

$$\begin{aligned} \text{► } \frac{\partial C}{\partial a_0^{[2]}} &= \text{► } \frac{\partial a_0^{[2]}}{\partial z_0^{[2]}} = \text{► } \frac{\partial z_0^{[2]}}{\partial w_{00}^{[2]}} = a_0^{[1]}. & \text{► } \frac{\partial z_0^{[2]}}{\partial b_0^{[2]}} = 1. \end{aligned}$$

$$\text{On a donc, } \frac{\partial C}{\partial w_{00}^{[2]}} = 2(a_0^{[2]} - y) f'^{[2]}(z_0^{[2]}) a_0^{[1]} \text{ et } \frac{\partial C}{\partial b_0^{[2]}} = 2(a_0^{[2]} - y) f'^{[2]}(z_0^{[2]}).$$

Prenons le cas où la fonction d'activation est la fonction identité. On a alors $\frac{\partial C}{\partial w_{00}^{[2]}} =$

$$2(a_0^{[2]} - y) a_0^{[1]} \text{ et } \frac{\partial C}{\partial b_0^{[2]}} = 2(a_0^{[2]} - y) \dots$$

On va ainsi pouvoir exprimer $\frac{\partial C}{\partial w_{00}^{[2]}}$, $\frac{\partial C}{\partial w_{01}^{[2]}}$, $\frac{\partial C}{\partial b_0^{[2]}}$, ... On pourrait ici écrire 9 équations en fonction des différents poids, des biais et des entrées x_0 et x_1 .

À partir de cela, on va modifier les poids comme suit :

$$\begin{aligned} \blacktriangleright w_{00,i+1}^{[2]} &= w_{00,i}^{[2]} - \eta \left. \frac{\partial C}{\partial w_{00}^{[2]}} \right|_{w_{00,i}^{[2]}} ; \\ \blacktriangleright b_{0,i+1}^{[2]} &= b_{0,i}^{[2]} - \eta \left. \frac{\partial C}{\partial b_0^{[2]}} \right|_{b_{0,i}^{[2]}} . \end{aligned}$$

On réitère ensuite les opérations précédentes jusqu'à ce que la fonction coût ait été suffisamment réduite.

Définition – Taux d'apprentissage

On définit l'hyperparamètre $\eta \in [0, 1[$ comme étant le taux d'apprentissage. Si ce taux d'apprentissage est très grand, l'algorithme d'apprentissage mettra beaucoup de temps à trouver le minimum. S'il est trop grand, le minimum peut ne jamais être trouvé.

Définition – Gradient

Dés lors, dans le cas présenté, on peut définir le gradient comme le vecteur

$$\text{grad}C = \begin{bmatrix} \frac{\partial C}{\partial w^{[1]}} \\ \frac{\partial C}{\partial b^{[1]}} \\ \vdots \\ \frac{\partial C}{\partial w^{[l]}} \\ \frac{\partial C}{\partial b^{[l]}} \end{bmatrix} .$$

17.3.4 Fin d'apprentissage

Définition – Epoch

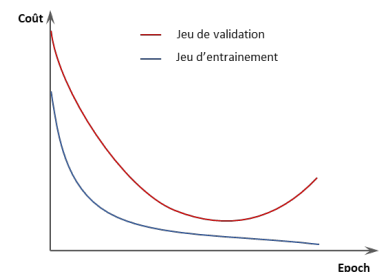
On appelle *epoch* un cycle d'apprentissage où tous les poids et tous les biais ont été mis à jour en faisant passer toutes les données du jeu d'entraînement dans les algorithmes de propagation et de rétropropagation.

Les méthodes pour stopper l'apprentissage sont essentiellement empiriques. On pourrait en effet fixer un nombre d'époch ou une valeur d'erreur admissible et s'arrêter à ce moment là. Dans la pratique, se focaliser sur l'erreur n'est généralement pas satisfaisant. En effet, il y a risque de « surapprentissage ».

Dans la figure suivante, on réalise un entraînement sur le jeu de données (une epoch) à la fin de l'époch on dispose d'un premier modèle de réseau de neurones. On détermine alors l'erreur commise en utilisant le jeu d'entraînement puis l'erreur commise sur le jeu de validation. On calcule ensuite l'erreur commise par le modèle sur le jeu de validation. (On rappelle que le jeu de validation ne sert pas à modifier les poids et les biais.)

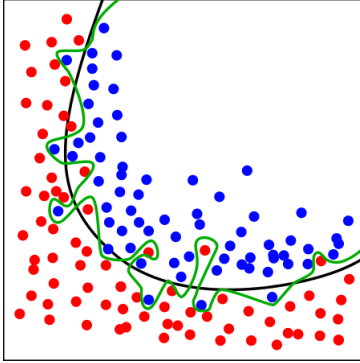
On réalise de même à la fin de l'époch suivante etc...

« Logiquement » l'erreur décroît toujours avec le jeu d'entraînement car la descente du gradient a pour objectif de réduire cette erreur.



Sur le jeu de validation, l'erreur décroît pendant un certain nombre d'époch puis augmente.

Il existe en fait un stade à partir duquel le réseau de neurones se spécialise sur le jeu d'entraînement et devient donc incapable de réaliser des prédictions fiables sur un nouveau jeu de données. On parle de surentraînement (ou d'overfitting).



Exemple – Wikipedia

La ligne verte représente un modèle sur-appris et la ligne noire représente un modèle régulier. La ligne verte classe trop parfaitement les données d'entraînement, elle généralise mal et donnera de mauvaises prévisions futures avec de nouvelles données. Le modèle vert est donc au final moins bon que le noir.

17.4 Pour aller plus loin...

Les réseaux présentés ci-dessus sont les réseaux dits denses (ou fully-connected). On utilise d'autres types de réseaux pour une meilleure prédiction en fonction des données d'entrées :

- ▶ réseaux convolutifs (CNN) pour l'analyse d'images ;
- ▶ réseaux de neurones récurrents pour les données temporelles *etc.*

Le cours suivant présente les réseaux de type NARX et MRAC qui sont utilisés dans Matlab pour gérer les signaux temporels. Ils ont la particularité de réutiliser les sorties estimées par le réseau comme entrées décalées.

TD 1

Contrôle du correcteur de facteur de puissance par Intelligence Artificielle ★ – Sujet

CCINP – PSI – 2024 – Modélisation.

05 NUM

Le correcteur de facteur de puissance est utilisé pour minimiser les pertes en lignes engendrées par l'installation électrique d'un petit studio.

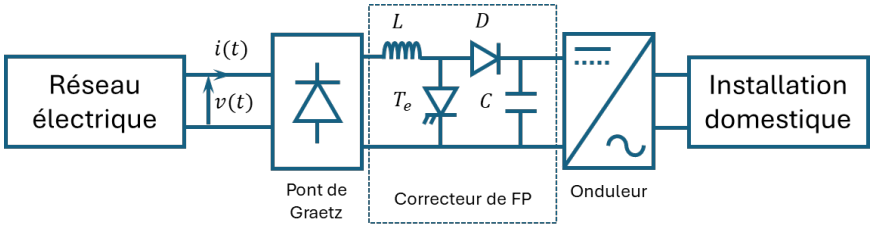


FIGURE 17.1 – Correcteur de facteur de puissance utilisé pour une installation domestique

La figure 17.1 présente le schéma de principe du dispositif complet : la tension aux bornes du condensateur C est placée en entrée de l'association d'un onduleur de tension et de l'installation électrique du studio. Une boucle de régulation permet de réguler cette tension à une valeur constante.

L'installation étudiée comporte un ensemble box internet-télévision, un système d'éclairage, un radiateur électrique, un chauffe-eau.

Chacun des quatre éléments de l'installation électrique considérée présente trois modes de fonctionnement. Un niveau x_i allant 0, 1 ou 2 est associé à chacun de ces modes. Le tableau 17.1 donne la correspondance entre le niveau x_i et le mode de fonctionnement de chaque élément. Certaines des puissances consommées par l'élément correspondant sont également données.

TABLE 17.1 – Modes de fonctionnement

	Box-TV – x_1	Eclairage – x_2	Radiateur – x_3	Chauffe-eau – x_4
$x_i = 0$	éteint, 0 W	éteint, 0 W	éteint, 0 W	éteint, 0 W
$x_i = 1$	veille, 100 W	puissance moitié, 50 W	puissance moitié	maintient à température 50 W
$x_i = 2$	allumé, 200 W	puissance maximale, 100 W	puissance maximale	puissance maximale

Le dispositif de la figure 17.1 permet de réduire les pertes en ligne engendrées par l'absorption d'un courant impulsionnel. Celles-ci tendent vers leur valeur minimale lorsque le paramètre Δ tend vers 0. Cependant, cette diminution se fait au prix d'une augmentation de la fréquence de commutation moyenne f_{cm} et donc des pertes au sein du transistor Tr . Si la puissance dissipée dans Tr ne peut pas être exploitée, il est alors nécessaire de rechercher une valeur du paramètre Δ correspondant à une minimisation des pertes totales : pertes en ligne + pertes dans Tr .

La valeur du paramètre Δ permettant de minimiser les pertes totales, notée Δ_{opt} , a été évaluée empiriquement pour quatorze combinaisons entre les différents modes de fonctionnement des quatre éléments de l'installation électrique étudiée. Ces valeurs sont présentées dans le tableau 17.2. Le paramètre Δ s'exprime en volts et est positif. Lorsque la valeur -1 est indiquée pour Δ_{opt} , cela signifie que pour minimiser les pertes totales, il est préférable de désactiver le correcteur de facteur de puissance et de connecter l'installation directement au réseau électrique.

TABLE 17.2 – Valeurs optimales du paramètre Δ pour quatorze combinaisons différentes

Box-TV x_1	Eclairage x_2	Radiateur x_3	Chauffe-eau x_4	Δ_{opt} (V)
0	0	0	1	-1
0	0	1	1	-1
0	0	2	0	-1
0	0	0	2	-1
0	0	1	2	-1
0	1	0	0	0,18
2	2	0	0	0,32
2	2	2	0	0,49
2	2	2	2	-1
0	0	1	0	-1
0	0	2	1	-1
0	0	2	2	-1
1	0	0	0	0,22
2	2	1	0	0,42

Contrôle du paramètre Δ avec un réseau de neurones

Il faut mettre en place un contrôleur capable de fournir la consigne Δ_{opt} en fonction des différentes combinaisons (x_1, x_2, x_3, x_4) . Le contrôleur associé au correcteur de facteur de puissance doit permettre en temps réel de :

- soit de désactiver le correcteur de facteur de puissance en connectant l'installation électrique du studio au réseau électrique (cas $\Delta = -1$);
- soit de fixer la valeur du paramètre Δ lorsque le correcteur de facteur de puissance est activé.

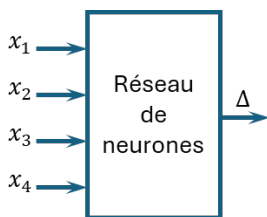


FIGURE 17.2 – Entrées-sortie du réseau de neurones

Le tableau 17.2 nous donne la correspondance entre les états des appareils x_i et le réglage Δ du correcteur. Les données d'entraînement sont celles qui vont servir à l'entraînement du réseau de neurones. Les données de test sont celles qui vont permettre de valider si l'entraînement s'est bien effectué. D'un point de vue système, cela revient à chercher la relation entre les données d'entrées x_i et la sortie Δ (figure 17.2).

L'objectif est de trouver un modèle de comportement qui permet, à partir du tableau 17.2 de retrouver les valeurs d'entraînement et de prédire les valeurs de tests.

Pour y arriver, l'idée est d'entraîner un réseau de neurones de manière supervisée à partir des données du tableau 17.2.

Question 1 Quels sont les avantages et les inconvénients du choix d'un réseau de neurones pour modéliser la commande ?

Question 2 L'apprentissage supervisé est-il pertinent ? Justifier votre réponse.

Définition d'un perceptron

Le réseau de neurones choisi est un perceptron multicouches à une première couche cachée (couche d'entrée), une deuxième couche cachée, et une couche de sortie. Le système prend en entrée un vecteur à quatre composantes (x_1, x_2, x_3, x_4) . La couche d'entrée est composée de quatre neurones numérotés de 1 à 4. La deuxième couche cachée contient également quatre neurones numérotés de 1' à 4'. La couche de sortie contient un seul neurone numéroté 1''. La sortie de la couche de sortie est Δ . Chaque neurone prend en entrée les sorties de la couche précédente et renvoie une unique sortie.

La figure 17.3 présente cette structure.

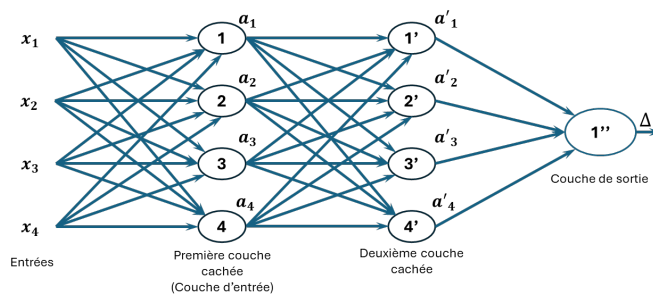


FIGURE 17.3 – Structure du réseau de neurones choisi

Le détail d'un neurone est défini par la figure 17.4.

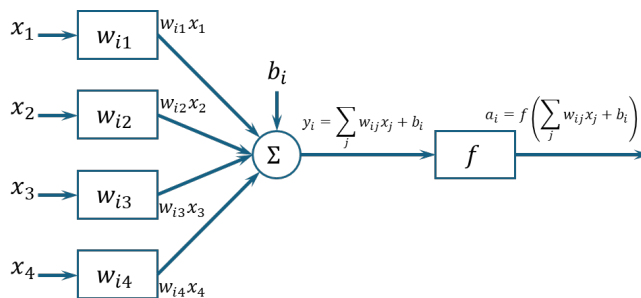


FIGURE 17.4 – Structure d'un neurone i de la première couche

La figure 17.4 d'un neurone numéroté « i » de la première couche permet, à partir des valeurs d'entrées (x_1, x_2, x_3, x_4) de calculer la valeur de sortie a_i . Les sorties de la première couche deviennent les entrées de la couche suivante et ainsi de suite. Tous les neurones vont se comporter comme ceux de la première couche. Il est à noter que la sortie du dernier neurone correspond à Δ .

La fonction d'activation choisie est une sigmoïde définie par $f : \mathbb{R} \rightarrow \mathbb{R}, x \rightarrow f(x) = \frac{1}{1 + e^{-x}}$. En Python, l'utilisation de la fonction `exp` de la librairie `numpy` permet de calculer, pour une entrée `x` de type tableau, la matrice $f(x)$, issue de l'application de la fonction f à tous les éléments de la matrice `x`. Cette fonction d'activation se traduit par la fonction suivante.

```
1 def f(x) :
2     return (1/(1+np.exp(-x)))
```

Question 3 Calculer la dérivée de la fonction f notée $f'(x)$ et écrire une fonction Python notée `f_prime` qui prend en argument `x` et qui renvoie la dérivée $f'(x)$. L'argument d'entrée `x` est de type tableau.

Nous noterons :

- w_{ij} : le poids entre l'entrée j et le neurone i , c'est un nombre réel;
- b_i : le biais du neurone i , c'est un nombre réel;
- f : la fonction d'activation du neurone i qui s'applique à $\sum_j w_{ij}x_j + b_i$.

Le processus permettant de trouver les valeurs de b_i et de w_{ij} va être itératif. Les valeurs initiales seront choisies aléatoirement.

Notations :

- ▶ les grandeurs scalaires seront notées en minuscule, par exemple w ;
- ▶ les grandeurs matricielles seront notées en majuscule, par exemple W ;
- ▶ l'indice k fait référence à la couche, par exemple W_2 est pour la deuxième couche.

Phase d'inférence

La phrase d'inférence consiste à calculer la sortie à partir d'un vecteur d'entrée connu. La sortie du dernier neurone du réseau est la valeur Δ recherché et peut s'écrire sous la forme suivante : $\Delta = f(W_3(f(W_2(f(W_1X + B_1))))$.

Question 4 Donner les dimensions des matrices W_3 , W_2 , B_2 et b_3 .

On cherche à modéliser la couche d'entrée du réseau de neurones de la figure 17.3.

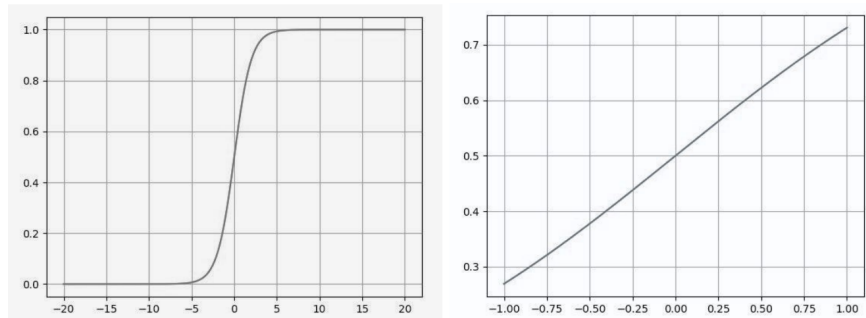
Question 5 Donner l'expression de la matrice W_1 qui vérifie :

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = W_1 \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} - & - & - & - \\ - & - & - & - \\ - & - & - & - \\ - & - & - & - \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}.$$

Question 6 Ecrire en Python la fonction `infereur_couche` qui va prendre en argument le vecteur d'entrées noté X , la matrice des points notée W , le vecteur biais noté B et qui renvoie le vecteur des sorties de la couche noté A .

La figure 17.5 montre le tracé de la fonction d'activation sigmoïde.

FIGURE 17.5 – Représentation graphique de la fonction sigmoïde f pour $x \in [-20, 20]$ (à gauche) et $x \in [-1, 1]$ (à droite)



Question 7 Calculer $A = f(Y)$ dans le cas où : $A = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix}$, $Y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix}$, $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$,

$$W = \begin{pmatrix} 0,3 & 0,2 & 0,25 & -0,03 \\ 0,4 & -0,3 & 0,6 & -0,3 \\ -0,4 & 0,3 & -0,6 & -0,5 \\ -1 & -0,75 & -0,1 & -0,5 \end{pmatrix}, B = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Notons que, pour cette première phase d'apprentissage qu'on appelle souvent l'initialisation, les valeurs des poids ont été générées aléatoirement et les biais choisis nuls. En reproduisant le même calcul pour les couches suivantes, nous calculons que $\Delta = 0,59 \text{ V}$. Cela signifie que pour l'éclairage allumé et tous les appareils éteints, il faudrait un réglage de $\Delta = 0,59 \text{ V}$.

Cependant le tableau 17.2 indique que pour cette combinaison d'états, le réglage optimal est $\Delta_{\text{opt}} = 0,18 \text{ V}$.

Il faut modifier les valeurs des poids et des biais pour que la sortie Δ du réseau de neurones se rapproche de Δ_{opt} et ce pour toutes les combinaisons d'entrées possibles. Autrement dit, pour toute combinaison d'entrée (x_1, x_2, x_3, x_4) , on souhaite minimiser l'erreur (choisie quadratique ici) entre la sortie Δ calculée par le réseau de neurones et le réglage optimal Δ_{opt} , déterminé à l'avance. Pour ne pas alourdir les notations

on notera désormais t le réglage optimal Δ_{opt} . Le calcul d'erreur $(t - \Delta)^2$ va être utilisé pour calculer des « meilleures » valeurs de poids et de biais. C'est la phase d'entraînement qui fait partie de la partie suivante.

Rétropropagation¹

1: REVOIR NOTATIONS

Nous disposons de $N = 9$ données d'entraînement c'est-à-dire toutes les paires $\{X_\ell, t_\ell\}$ où $X_\ell = (x_1, x_2, x_3, x_4)$ est le vecteur de la combinaison des entrées t_ℓ est simplement la valeur optimale Δ_{opt} , évaluée empiriquement et qui constitue la valeur cible souhaitée en sortie du réseau de neurones.

Ces données d'entraînement sont recensées dans le tableau 17.2. Pour chaque donnée d'entraînement $\{X_\ell, t_\ell\}$, on notera Δ_t le résultat réellement renvoyé par le réseau de neurones. Le but de la phase de rétropropagation est de calculer des valeurs de paramètres W et B qui « rapprochent » tous les Δ_ℓ calculés pour les X_ℓ des valeurs optimales t_ℓ . Une instance de rétropropagation s'effectue en cherchant les valeurs des paramètres W et B qui minimisent la fonction de coût quadratique entre la cible t_ℓ et la valeur calculée $\Delta_\ell = \mathcal{L}_\ell = (t_\ell - \Delta_\ell)^2$.

Pour des raisons de lisibilité, on notera $\mathcal{L} = \mathcal{L}_\ell$.

Pour ce faire, les poids W et les biais B sont mis à jour par la méthode de la descent du gradient, étudiée dans les questions ci-après. Afin de se fixer les idées, on s'intéresse à la première couche du réseau de neurones, ayant pour entrée le vecteur X et pour sortie le vecteur d'activation $A : A = f(W_A X + B_1)$. Le fonctionnement des autres couches sera identique. Pour des raisons de lisibilité, on posera désormais $W = W_1$, $B = B_1$, $Y = WX + B$. On a donc : $A = f(WX + B) = f(Y)$.

Les valeurs des paramètres W et B de cette couche sont mis à jour grâce aux formules

$$\text{suivantes : } \begin{cases} W = W - \alpha \frac{\partial \mathcal{L}}{\partial W} \\ B = B - \alpha \frac{\partial \mathcal{L}}{\partial B} \end{cases}$$

où α est le taux d'apprentissage, un paramètre à choisir manuellement. On adopte la notation suivante :

$$\frac{\partial \mathcal{L}}{\partial W} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial w_{11}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{14}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \mathcal{L}}{\partial w_{41}} & \cdots & \frac{\partial \mathcal{L}}{\partial w_{44}} \end{pmatrix}, \frac{\partial \mathcal{L}}{\partial X} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial x_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial x_4} \end{pmatrix}, \frac{\partial \mathcal{L}}{\partial A} = \begin{pmatrix} \frac{\partial \mathcal{L}}{\partial a_1} \\ \vdots \\ \frac{\partial \mathcal{L}}{\partial a_4} \end{pmatrix}.$$

$$\text{Nous avons } \frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial a_1} \frac{\partial a_1}{\partial w_{ij}} + \cdots + \frac{\partial \mathcal{L}}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}} + \cdots + \frac{\partial \mathcal{L}}{\partial a_4} \frac{\partial a_4}{\partial w_{ij}} \text{ et } a_i = f\left(\sum_j w_{ij} x_j + b_i\right).$$

Question 8 Montrer que $\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial \mathcal{L}}{\partial a_i} f'(y_i) x_j$.

Question 9 En utilisant la relation précédente, donner la relation matricielle entre $\frac{\partial \mathcal{L}}{\partial W}$, $\frac{\partial \mathcal{L}}{\partial A}$, $f'(Y)$ et X , puis entre $\frac{\partial \mathcal{L}}{\partial W}$, $\frac{\partial \mathcal{L}}{\partial A}$, W , X et B .

En poursuivant l'analyse il est possible de trouver la relation suivante qui va permettre d'exprimer l'erreur en entrée en fonction de l'erreur en sortie : $\frac{\partial \mathcal{L}}{\partial X} = \frac{\partial \mathcal{L}}{\partial A} (WX f'(WX + B))^2$.

2: xxxx

Il est à remarquer que dans la formule précédente, la multiplication se fait terme à terme.

Question 10 Ecrire la fonction Python `retropropagation_couche` qui prend en argument :

- ▶ la valeur d'entrée d'une couche, notée X ;
- ▶ la matrice de poids d'une couche notée W ;
- ▶ le vecteur de biais de la couche noté B ;
- ▶ le vecteur d'erreur en sortie de la couche, noté Ea ;
- ▶ la valeur du coefficient d'apprentissage, notée α .

Cette fonction doit calculer $E_W = \frac{\partial \mathcal{L}}{\partial W}$, $E_B = \frac{\partial \mathcal{L}}{\partial B}$, $E_X = \frac{\partial \mathcal{L}}{\partial X}$. Cette fonction doit renvoyer :

- ▶ la matrice de poids W mise à jour ;
- ▶ la vecteur de biais B mis à jour ;
- ▶ le vecteur gradient du coût $E_X = \frac{\partial \mathcal{L}}{\partial X}$.

Ainsi, après la phase d'initialisation, en appliquant une succession de phase d'inférence et de rétropropagation, pour chacune des données d'entraînement, et ce, plusieurs fois, on peut faire diminuer la valeur de la fonction de coût total jusqu'à atteindre la précision demandée : $L_{\text{tot}} = \frac{1}{N} \sum_{\ell=1}^N (t_{\ell} - \Delta_{\ell})^2 = \frac{1}{N} \sum_{\ell=1}^N L_{\ell}$.

Analyse des résultats

L'évolution de la fonction coût total L_{tot} pendant la phase d'entraînement est donnée sur la figure 17.6. Elle comporte 100 itérations.

Le graphique montre que la phase d'entraînement, constituée d'une succession d'inférences et de rétropropagation, a permis de diminuer la fonction coût, c'est-à-dire l'erreur quadratique moyenne entre le résultat Δ du modèle et les valeurs cibles des données d'entraînement. Il y a désormais peu d'écart pour les données d'entraînement.

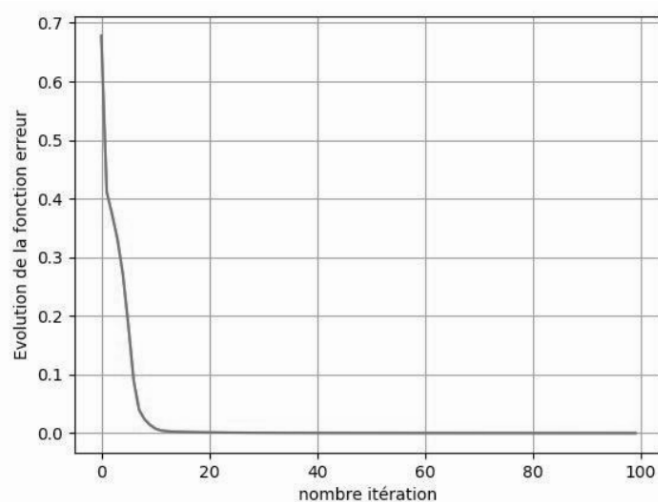


FIGURE 17.6 – Entraînement modèle

Le tableau 17.3 présente les données d'entraînement et les données de tests, et sa dernière colonne recense les valeurs Delta calculées par le réseau de neurones.

Sur les 14 données disponibles, $N = 9$ données sont utilisées lors de la phase d'entraînement et les 5 autres sont réservées comme données tests, afin de tester la robustesse

	box-TV x_1	éclairage x_2	radiateur x_3	chauffe-eau x_4	$\Delta_{opt}(V)$	$\Delta(V)$
Données d'entraînement	0	0	0	1	-1	-0,981
	0	0	1	1	-1	-0,979
	0	0	2	0	-1	-0,961
	0	0	0	2	-1	-0,983
	0	0	1	2	-1	-0,980
	0	1	0	0	0,18	0,178
	2	2	0	0	0,32	0,341
	2	2	2	0	0,49	0,474
Données test	2	2	2	2	-1	-0,970
	0	0	1	0	-1	-0,919
	0	0	2	1	-1	-0,977
	0	0	2	2	-1	-0,978
	1	0	0	0	0,22	0,015
	2	2	1	0	0,42	0,412

TABLE 17.3 – Valeurs optimales du paramètre Δ pour quatorze combinaisons différentes

du modèle en soumettant des entrées non utilisées pour la phase d'entraînement, afin de vérifier si la prédiction du modèle est bonne même sur des données nouvelles.

Question 11 Analyser les résultats. Ce réseau de neurones est-il capable de bien prédire la valeur optimale de A .

Question 12 Pour améliorer le modèles est-il préférable d'augmenter le nombre d'itérations de l'entraînement ou d'augmenter le nombre de données de tests ? Justifier votre réponse.