Théorie des graphes

Création d'un modèle de classification des chauves-souris du canton de Fribourg via un Perceptron multicouche

Les liens cachés entre les mathématiques, la biologie et l'informatique

Travail de Maturité

Thibault Liaudat
Samuel Russo

Collège du Sud, Bulle
Avril 2019

Travail de maturité réalisé
sous la direction de
Laurent Karth-Robadey

(Mathématiques)

Table des matières

Avant-propos

Ce Travail de Maturité a pris ses racines en novembre 2017 au Japon. L'un des rédacteurs était alors en échange linguistique sur l'archipel nippon pour une durée d'une année. Cependant, ceci ne l'a pas découragé à prendre contact avec M. Laurent Karth-Robadey afin de lui proposer un sujet pour l'élaboration du Travail de Maturité. C'est ainsi que, par quelques échanges de courriers électroniques, la thématique des réseaux de neurones prit forme.

Afin de voir sa proposition acceptée par la Direction de l'établissement du Collège du Sud, il fallait réunir au mininum trois personnes. Pour ce faire il prit contact avec différents étudiants potentiels à la collaboration de la thématiques.

C'est de ceci que naquit la collaboration entre Samuel Russo et Thibault Liaudat pour la réalisation de ce Travail de Maturité. Le troisième étudiant a ,quant à lui, rédigé son travail de manière individuelle.

Que Monsieur Laurent Karth-Robadey soit remercié pour l'aide ainsi que le soutien apportés pour la création de ce projet.

Introduction

0.1 Motivations et Objectifs

0.1.1 Motivations

> "As far as laws of mathematics refer to reality, they are not certain ; and as they are certain, they do not refer to reality."
>
> *-Albert Einstein*

Les mathématiques sont souvent perçues comme l'une des branches les plus complexes du cursus scolaire, et ceci vient du fait de leur abstraction. Il faut comprendre des concepts en se reposant sur certains axiomes, qui ne sont pas de vrais miroirs de la réalité. C'est sur ceux-ci que sont fondés les restes des mathématiques, qui ne sont donc elles aussi à leur tour qu'un miroir imparfait de notre vérité quotidienne. Mais c'est de là que naquit notre motivation pour ce travail de maturité : trouver les liens invisibles qui lient les mathématiques à notre quotidien.

C'est pour cela que nous avons envie de transmettre cette curiosité à d'autres personnes afin qu'elles aussi puissent éprouver de l'intérêt dans ce domaine, et même à d'autres comme celui de la biologie et de l'informatique pour ainsi leur faire voir les liens qui connectent toutes ces branches, et la beauté de ceux-ci.

Une autre de nos motivations est celle de l'intérêt pour les nouveautés. L'une des principales avancées technologiques de ce millénaire est l'explosion du digital, presque tout est maintenant numérisé. Cela va de nos voitures jusqu'à nos montres qui n'indiquaient que l'heure il y a de cela quelques années, alors qu'aujourd'hui nous pouvons avoir des rapports détaillés de notre activité physique simplement en portant une montre connectée au poignet. Ces quantités incroyables de données ainsi collectées ont alors créé un nouveau domaine de recherche : le Big Data. Et à partir

de celui-ci, beaucoup d'algorithmes d'optimisation de nos problèmes quotidiens sont nés. Ils ont alors été regroupés sous l'appellation globale de Machine Learning [1], et c'est dans cela que nous espérons vous plonger lors de ce travail de maturité.

0.1.2 Objectifs

Notre principal objectif va être d'expliquer le principe du réseau de neurones - un exemple de *Machine Learning* - pour ensuite l'appliquer en créant un modèle de classification des groupes acoustiques des chauves-souris recensées dans le canton de Fribourg.

Notre deuxième objectif, cette fois-ci plus général, est de montrer à l'aide de notre exemple concret que les mathématiques jouent un rôle omniprésent dans notre société actuelle.

Notre public cible est l'élève de $3^{\text{ème}}$ gymnase en option spécifique physique et application des maths, avec une connaissance de base de l'informatique.

0.2 Organisation

Pour obtenir la plus grande simplicité possible, ce travail a été conçu en trois parties distinctes.

(1) Le modèle mathématique d'un réseau de neurones

(2) La mesure et collecte des données des cris des chauves-souris

(3) La mise en application du modèle sur les données collectées

La première partie est exclusivement centrée sur le réseau de neurones, de son principe de base à ses composantes complexes. La seconde aborde les données collectées par *FRIBat-CCO* à analyser et à classifier. La dernière est la partie pratique du travail de maturité. Elle comporte tout le processus ainsi que le produit final de la création d'un modèle de classification qui sera utilisé pour classifier les groupes acoustiques des chauves-souris vivant dans le canton de Fribourg, principalement à proximité du Lac de Pérolles.

1. Cet anglicisme peut aussi être traduit par "apprentissage automatique"

Chapitre 1

Réseaux de neurones

Le cerveau humain est l'une des merveilles de ce monde. Notre capacité à résoudre des problèmes à première vue simples – comme reconnaître un chat sur une image ou pouvoir distinguer le chiffre 1 du chiffre 2 – mais qui en réalité cachent une complexité phénoménale, nous permet de prendre des décisions adaptées à chaque situation. Imaginez-vous, la région primaire du cortex visuel, appelée V1, contient à elle seule plus de 140 millions de neurones. Mais la vision humaine ne se contente pas seulement de cette région, mais aussi d'une série entière de cortex visuels – V2, V3, V4, et même V5 – qui développent l'image progressivement, et cela en moins d'un 20$^{\text{ème}}$ de seconde. S'ensuit la question qui passionne beaucoup de chercheurs aujourd'hui : « Peut-on simuler le cerveau, ses milliards de neurones, sur un super-ordinateur ? » [1].

Dans ce travail nous allons développer une des techniques actuelles qui permet de simuler le comportement d'un élément du cerveau en particulier, le neurone, comme celui représenté dans la figure 1.1. Dans les sections qui suivent, nous allons vous présenter le modèle mathématique qui est inspiré de celui-ci, et comme un seul neurone n'aurait finalement pas un grand intérêt pour résoudre les défis du quotidien, nous vous présenterons aussi l'idée qui a révolutionné l'intelligence artificielle au XX$^{\text{e}}$ et XXI$^{\text{e}}$ siècle, le réseau de neurones.

Dans ce chapitre, les principales sources utilisées sont les suivantes : [NIE15], [RAS16], [CMB18], [LBBOM98] et [19].

1. L'EPFL développe notamment le projet *Blue Brain* qui a pour but de simuler l'entièreté du cerveau humain.

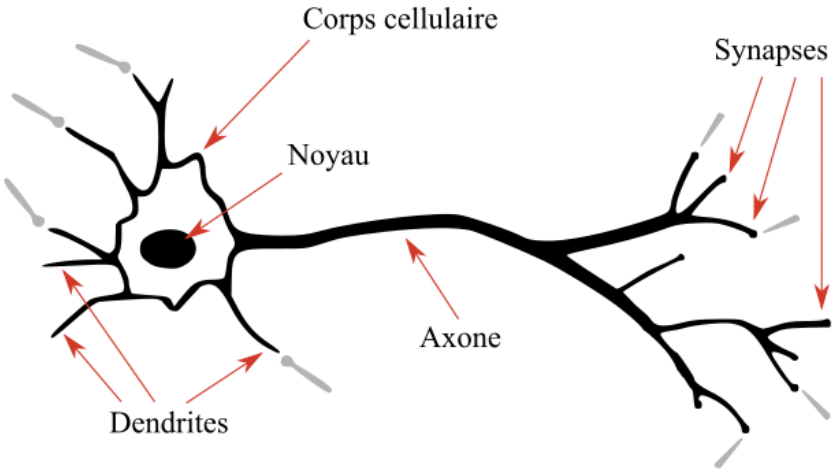


FIGURE 1.1 – Neurone biologique. [17]

1.1 Machine learning et chauves-souris

L'implémentation d'un réseau de neurones artificiels sur un ordinateur va nous permettre de faire un premier pas vers un sous-domaine de l'intelligence artificielle, le machine learning. Ce dernier est la science qui cherche à faire agir un ordinateur comme un être humain, en le faisant apprendre comme celui-ci, en lui injectant des données et des informations. C'est grâce à ce procédé que nous allons créer un algorithme qui va apprendre à partir d'une grande base de données - les cris des chauves-souris se trouvant dans la réserve naturelle du Lac de Pérolles, dans le canton de Fribourg - et qui pourra ensuite les classifier comme un biologiste le ferait. Voici le principe même de l'intelligence artificielle : *la machine qui imite l'homme.*

1.2 Classifier, oui, mais ça veut dire quoi ?

Selon *Larousse* classifier consiste à « ranger méthodiquement des choses par classes, par catégories ». Les biologistes ayant déjà défini chaque espèce selon leurs critères, il ne nous reste plus qu'à y ranger les spécimens à l'intérieur de celles-ci. Mais comme la nature est bien faite, il n'existe pas deux spécimens identiques. Si un ordinateur se contentait d'une simple analyse des critères un par un, les résultats ne seraient pas optimaux. Il nous faut donc mettre tous ces critères en parallèle, les analyser les uns par rapport aux autres pour obtenir une meilleure classification. Pour mettre différents critères en parallèle, nous pouvons par exemple les placer dans un espace

où chaque axe correspond à un critère donné, puis représenter les individus comme des points respectivement à leurs caractéristiques, comme le montre la figure 1.2. Pour séparer ces deux individus, il nous suffit de trouver une ligne qui sépare le plan

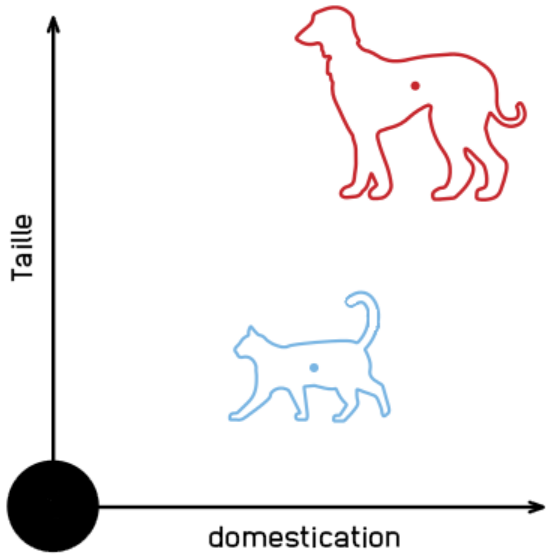


FIGURE 1.2 – Différents individus comme différents points. [10]

en deux comme le montre la figure 1.3. Comme vu dans le cours de mathématiques de deuxième année, nous pouvons dire que tous les individus qui résolvent la condition suivante se trouvent au dessus de la ligne, et ils seront donc classés comme des canidés si l'on choisit des valeurs correctes pour a, b et c :

$$a \cdot domestication + b \cdot taille + c > 0$$

Ceux qui résolvent la condition suivante se trouvent en dessous ou sur la ligne, et seront donc classés comme des félins :

$$a \cdot domestication + b \cdot taille + c \leq 0$$

Si l'on peut effectivement séparer les deux classes d'individus par une unique droite, le problème est dit *linéairement séparable*. Il nous faut donc trouver cette droite optimale, afin que tous les spécimens se retrouvent classifiés du bon côté de celle-ci. Mais voilà, comment trouver cette ligne ? Quelles sont les méthodes les plus adaptées ? L'algorithme présenté à partir d'ici est l'une des solutions actuelles.

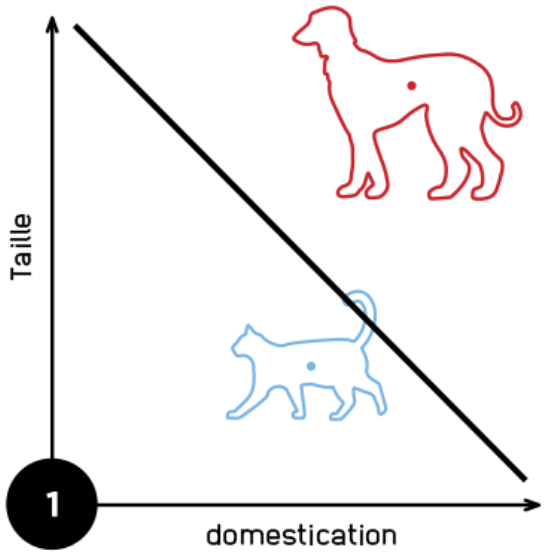


FIGURE 1.3 – Droite permettant de séparer les deux individus.

1.3 Le perceptron, un neurone artificiel

1.3.1 Construction

La représentation d'un neurone biologique en tant que modèle mathématique qui nous intéresse a été développée en 1957 par Frank Rosenblatt sous le nom de *perceptron*. Il permet de simuler une prise de décision basique. Ici, le terme *simuler* est important, car il nous montre bien que la machine ne comprend pas ce qu'elle fait, elle agit uniquement sur des chiffres et c'est à nous de donner une signification à ceux-ci. Le graphe du perceptron (voir figure 1.4) est décomposé en plusieurs parties. Il prend n entrées binaires, x_1,x_2,x_3,...,x_n, et produit une seule sortie - que nous noterons [2] $\pi(x_1, x_2, ..., x_n)$ - elle aussi binaire. L'importance de chaque entrée est représentée par un poids, w_1,w_2,w_3,...,w_n. Le point central est une fonction de somme, où toutes les entrées sont multipliées par leur poids puis tous ces produits sont additionnés. Nous obtenons donc la somme $\sum_{j=1}^{n} w_j x_j$. On additionne ensuite à cette somme une valeur arbitraire, le *biais*, qui symbolise la facilité que le perceptron aura à s'activer puis on passe le tout dans la fonction d'activation. Dans le cas du perceptron, nous utilisons la fonction en escalier – souvent appelée *Step Function* –

2. Ici, le choix du symbole π se fait pour cause de sonorité à l'oral.

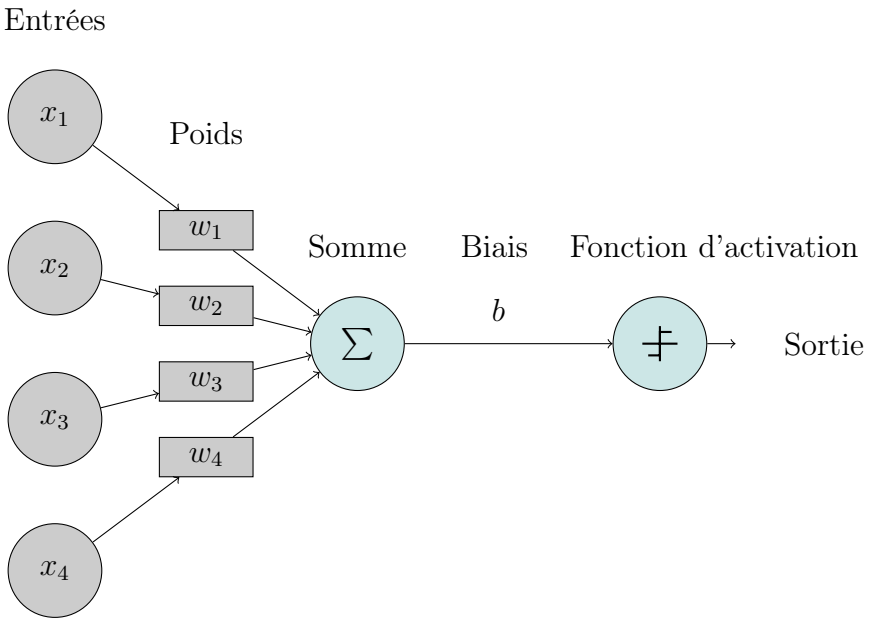


FIGURE 1.4 – Perceptron à 4 entrées

qui retourne 0 pour une valeur négative ou nulle et 1 pour une valeur strictement positive (voir figure 1.5). Exprimé plus clairement, nous avons la définition suivante :

$$\pi(x_1, x_2, ..., x_n) = step\Big(\sum_{j=1}^{n} w_j x_j + b\Big) = \begin{cases} 0 & \text{si } \sum_{j=1}^{n} w_j x_j + b \leq 0, \\ 1 & \text{si } \sum_{j=1}^{n} w_j x_j + b > 0. \end{cases} \tag{1.1}$$

Nous dirons que le neurone est *activé* si $\pi(x_1, x_2, ..., x_n) = 1$. Cette notation en somme étant un peu lourde, à la place de voir chaque entrée comme une variable individuelle, nous pouvons toutes les concentrer sous la forme d'un vecteur $\vec{x}$ à n dimensions. Nous pouvons aussi procéder de la même manière pour les poids et créer un vecteur $\vec{w}$. En effectuant le produit scalaire de ces deux vecteurs, nous obtenons la même somme que celle de l'équation précédente :

$$\vec{x} \cdot \vec{w} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = \sum_{j=1}^{n} w_j x_j$$

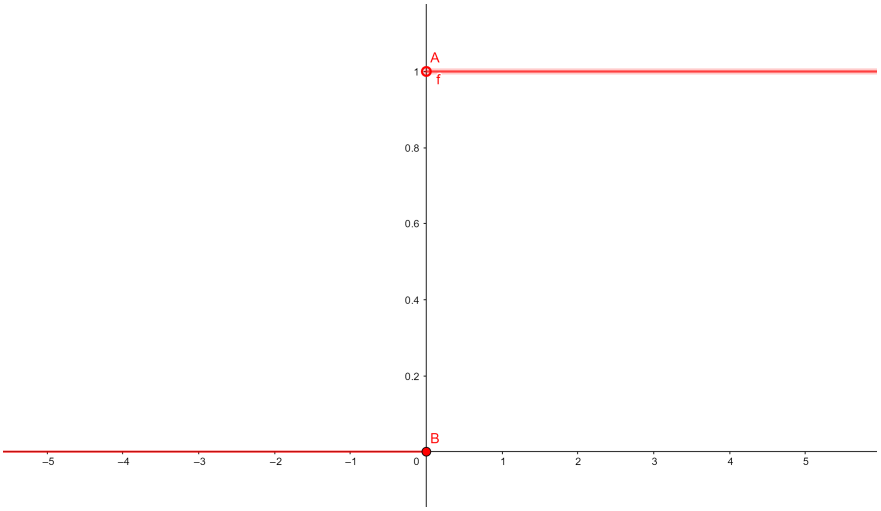


FIGURE 1.5 – Fonction en escalier.

L'équation (1.1) peut alors s'écrire de la façon suivante :

$$\pi(\vec{x}) = step(\vec{x} \cdot \vec{w} + b) = \begin{cases} 0 & \text{si } \vec{w} \cdot \vec{x} + b \leq 0 \\ 1 & \text{si } \vec{w} \cdot \vec{x} + b > 0 \end{cases} \tag{1.2}$$

1.3.2 Exemples d'utilisation du perceptron

Un modèle mathématique sans exemple étant plutôt froid et peu significatif, cette section présentera comment cette méthode permet de résoudre différents problèmes *linéairement séparables*. Comme vu plus tôt, le perceptron prend différentes valeurs en entrées et les somme selon leur importance, ce qui nous permet alors de simuler des modèles de prise de décisions basiques. Comme première illustration nous allons créer un perceptron qui nous permet de choisir entre aller en cours de mathématiques, ou *ne pas* y aller. Ce qui est important ici, c'est que notre sortie soit binaire, donc il n'y a que deux choix : aller en cours, ou non. La prochaine étape à la construction de notre modèle est le choix des différents facteurs qui vont faire différer notre envie d'aller en cours. Dans cet exemple, nous allons prendre arbitrairement les facteurs suivants : *Suis-je malade ?, Suis-je motivé ?, Suis-je bon en mathématiques ?, Le cours est-il intéressant ?*. Pour notre modélisation, nous considérons que tous ces facteurs sont des questions fermées, donc binaires qui n'ont toutes que deux réponses possibles : oui ou non. La prochaine étape est de choisir les poids et le biais de notre perceptron, respectivement l'importance que nous accordons à chaque entrée, et la

facilité que notre perceptron aura à sortir une réponse positive [3]. Dans ce cas aussi, nous sommes face à un choix arbitraire. La figure 1.6 illustre un choix possible de notre modélisation.

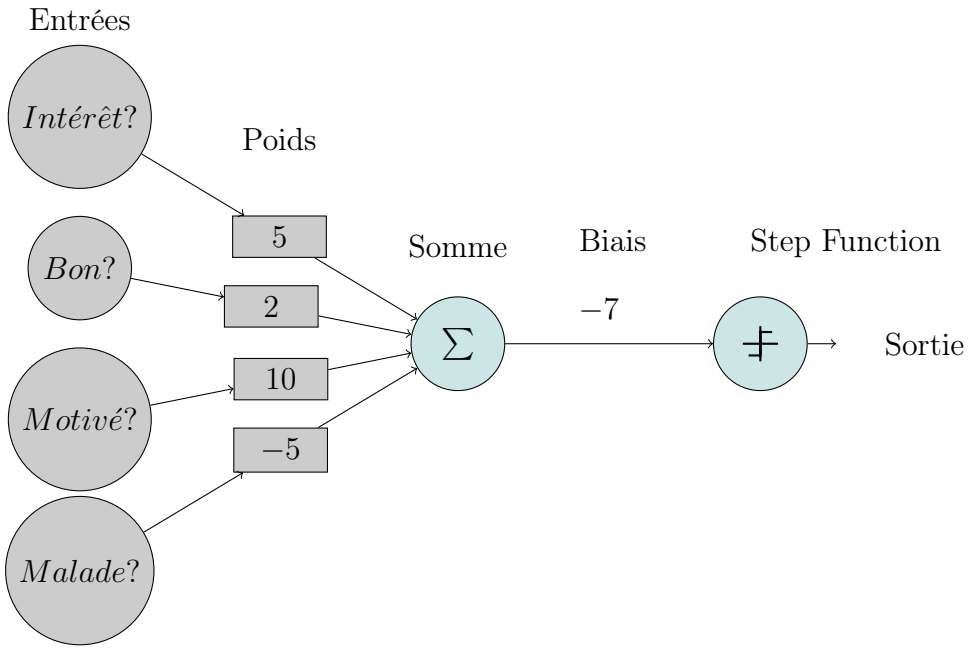


FIGURE 1.6 – Perceptron pour prendre la décision d'aller en cours de mathématiques

Calculons maintenant notre prise de décision avec les conditions suivantes : Je ne suis pas malade, je ne suis pas bon en mathématiques, je suis motivé et le cours est intéressant. Toutes les réponses *négatives* sont représentées par un 0, et celles *positives* par un 1. Pour la première étape, il nous faut sommer tous les facteurs selon leur importance et ajouter le biais :

$$\pi\left(\begin{bmatrix}1\\0\\1\\0\end{bmatrix}\right) = step\left(\begin{matrix}intérêt\\bon\\motivé\\malade\end{matrix}\begin{bmatrix}1\\0\\1\\0\end{bmatrix} \cdot \begin{bmatrix}5\\2\\10\\-5\end{bmatrix} - 7\right) = step(8) = 1$$

3. Nous pourrions parler dans ce cas-ci de motivation intrinsèque, qui n'est pas dûe à des stimuli extérieurs

La sortie étant 1, le modèle nous conseille donc d'aller en cours. Et voici un autre cas, où il nous conseille de ne pas y aller :

$$\pi\left(\begin{bmatrix}0\\0\\1\\1\end{bmatrix}\right) = step\left(\begin{pmatrix}intérêt & bon & motivé & malade\end{pmatrix}\begin{bmatrix}0\\0\\1\\1\end{bmatrix} \cdot \begin{bmatrix}5\\2\\10\\-5\end{bmatrix} - 7\right) = step(-2) = 0$$

Un autre exemple d'utilisation plus formel est la simulation de la porte logique *NAND*[4] dont la table de vérité est la suivante :

A	B	A *NAND* B
0	0	1
0	1	1
1	0	1
1	1	0

FIGURE 1.7 – Table de vérité de la porte logique NAND

Elle peut être simulée à partir du perceptron de la figure 1.8.

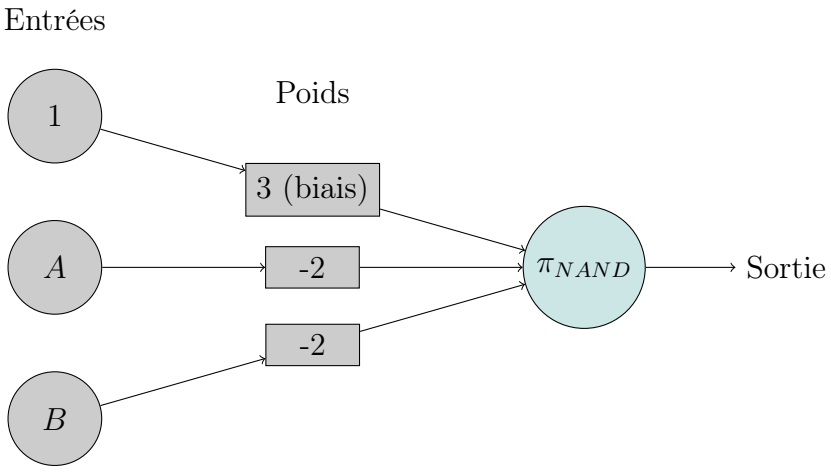


FIGURE 1.8 – Perceptron simulant la porte logique NAND

Pour faciliter les calculs par la suite, nous préférons ici transformer le biais b comme un poids, que nous appellerons w_0. L'entrée x_0 liée à ce poids aura toujours la valeur de 1. Le résultat du calcul reste donc le même, mais la forme change légèrement.

4. Les portes logiques sont la base de tout calcul informatique. Elles peuvent par exemple représenter une addition. La porte NAND exprime la logique « *Si les deux entrées sont allumées, alors la sortie ne l'est pas. Pour tous les autres cas, la sortie est allumée.* »

Voici à quoi ressemble notre perceptron actuellement, en considérant le biais comme le poids w_0 :

$$\pi_{NAND}(\vec{x}) = \begin{cases} 0 & \text{si } \vec{w} \cdot \vec{x} \leq 0, \\ 1 & \text{si } \vec{w} \cdot \vec{x} > 0, \end{cases} \tag{1.3}$$

avec $\vec{x}$ et $\vec{w}$ qui valent

$$\vec{x} = \begin{bmatrix} x_0 = 1 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \text{ et } \vec{w} = \begin{bmatrix} w_0 = \text{biais} \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}.$$

Vérifions que le perceptron de la figure 1.8 permet donc bien de simuler la porte logique NAND :

$$\pi_{NAND}\left(\begin{bmatrix}1\\0\\0\end{bmatrix}\right) = step\left(\begin{matrix}Biais\\A\\B\end{matrix}\begin{bmatrix}1\\0\\0\end{bmatrix} \cdot \begin{bmatrix}3\\-2\\-2\end{bmatrix}\right) = step(3) = 1$$

$$\pi_{NAND}\left(\begin{bmatrix}1\\0\\1\end{bmatrix}\right) = step\left(\begin{matrix}Biais\\A\\B\end{matrix}\begin{bmatrix}1\\0\\1\end{bmatrix} \cdot \begin{bmatrix}3\\-2\\-2\end{bmatrix}\right) = step(1) = 1$$

$$\pi_{NAND}\left(\begin{bmatrix}1\\1\\0\end{bmatrix}\right) = step\left(\begin{matrix}Biais\\A\\B\end{matrix}\begin{bmatrix}1\\1\\0\end{bmatrix} \cdot \begin{bmatrix}3\\-2\\-2\end{bmatrix}\right) = step(1) = 1$$

$$\pi_{NAND}\left(\begin{bmatrix}1\\1\\0\end{bmatrix}\right) = step\left(\begin{matrix}Biais\\A\\B\end{matrix}\begin{bmatrix}1\\1\\1\end{bmatrix} \cdot \begin{bmatrix}3\\-2\\-2\end{bmatrix}\right) = step(-1) = 0$$

1.3.3 Apprentissage du perceptron

Nous voilà avec un modèle qui nous permet de simuler certaines de nos décisions, mais il n'est pas extrêmement utile car jusqu'à maintenant nous avons dû faire nous même des choix arbitraires pour les poids, et trouver ceux qui nous convenaient le mieux. Le concept intéressant à introduire ici est celui de l'apprentissage : le modèle doit apprendre par lui-même, en trouvant les poids idéaux pour le problème auquel

il fait face. Pour trouver une solution à ce problème, nous devons nous pencher encore une fois sur notre propre manière de fonctionner : comment ai-je appris personnellement qu'un chat était un chat, un chien un chien, et l'eau de l'eau ? C'est par *l'exemple* de nos parents, qui pointaient le chat en disant "Chat !", et le chien en disant "Chien !". Ce qui nous mène à nous poser la question : pouvons-nous simuler cette apprentissage par l'exemple pour créer un perceptron qui apprend ? Vous vous

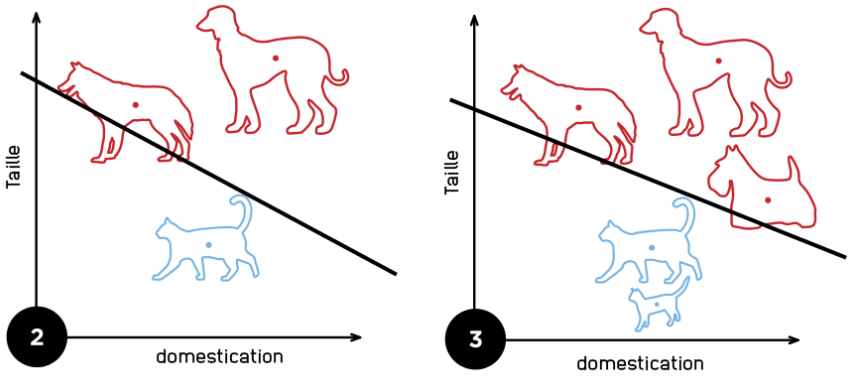


FIGURE 1.9 – Exemple d'apprentissage par l'exemple.

doutez bien que si nous vous en parlons, la réponse est oui ! Il faut bien se rendre compte qu'en faisant cela, nous apprenons à l'ordinateur à apprendre : c'est donc une intelligence artificielle capable d'apprendre par elle-même, simplement en regardant des données. Et plus le nombre d'exemples différents est élevé, plus l'algorithme sera performant comme le montre la figure 1.9. L'importance de ce processus est si considérable que nous pouvons alors parler d'une révolution de l'ère scientifique : de la première démarche empirique à l'approche théorique de Newton, suivie par l'ère de la simulation informatique, nous pouvons désormais essayer de modéliser le monde par une approche se basant sur l'exploration de données massives, et de nouvelles grandes découvertes s'ouvrent possiblement devant nous [5].

Voici comment nous allons procéder : à partir d'un ensemble de données qui contient les objets et leur label [6] respectif, nous initialiserons tous les poids du perceptron 0, et à chaque fois que le perceptron π se trompera dans sa prédiction, nous lui taperons sur les doigts, c'est-à-dire que nous changerons ses poids pour qu'il obtienne une meilleure réponse la fois suivante. S'il ne se trompe pas, alors nous ne changerons *pas* ses poids. L'algorithme du perceptron a la particularité qu'il nous faut itérer

5. Pour plus de précision à ce sujet, se référer à [CMB18].
6. Le label désigne la classe *véritable* de l'objet. Par exemple, le label de l'image d'un chat est "chat".

plusieurs fois à travers la base d'exemples pour atteindre des poids idéaux, mais attention à ne pas en abuser ! Faisons un parallèle avec la situation d'un élève qui se prépare à un examen : si celui-ci révise trop sur les exercices au point de les apprendre par coeur, alors il sera confus lors de l'examen, et évidemment, s'il ne révise pas assez, alors il donnera aussi des réponses confuses. Il faut donc trouver le juste milieu pour avoir un nombre d'itérations du set d'exemples qui soit optimal. Pour choisir ce nombre d'itérations, il nous faudra travailler de manière empirique, et donc au cas par cas. Mais pour l'instant, car nous allons utiliser des exemples simples, nous arrêterons d'itérer lorsque les poids du perceptron ne changent plus : le perceptron prédit donc correctement pour tous les exemples de la base de données.

Algorithme d'apprentissage du perceptron

Voici l'algorithme d'apprentissage du perceptron comme il est présenté par [Den15], où t est le nombre d'itérations réalisées, t_{max} le nombre d'itérations maximal et S notre base de données d'exemples. Celle-ci peut s'écrire sous la forme suivante : $S = \{(\vec{x_1}, y_1), (\vec{x_2}, y_2), ..., (\vec{x_n}, y_n)\}$, où $\vec{x_i}$ est le vecteur d'entrée, y_i la valeur désirée (soit 0, soit 1) et n le nombre d'exemples dans la base de données :

(a) Initialiser le vecteur des poids $\vec{w}$ à $\vec{0}$

(b) *Pour* itération 1 à t_{max} *faire* $\vec{w_t} \leftarrow \vec{w_{t-1}}$

  (i) Pour chaque exemple $(\vec{x_i}, y_i) \in S$ *faire*

    i. Calculer $\pi_t(\vec{x_i})$

    ii. *Si* $\pi_t(\vec{x_i}) \neq y_i$ *alors*

      A. Ajuster w : $\vec{w_t} \leftarrow \vec{w_t} + \vec{x_i}$ si $y_i = 1$, ou w : $\vec{w_t} \leftarrow w_t - \vec{x_i}$ si $y_i = 0$

    iii. *Fin si*

  (ii) *Fin pour*

(c) *Fin Pour*

L'étape la plus importante ici est la ligne A, car c'est celle qui exprime la mise à jour des poids. Elle peut paraître un peu étrange à première vue, mais voici son raisonnement : lorsque la sortie est trop faible, c'est à dire que $\pi_t(x)$ est égal à 0, alors il faut augmenter les poids, et vice-versa pour le cas où la sortie est trop grande. Le changement dans les poids se fait selon l'entrée. Cet algorithme fonctionne pour n'importe quel cas linéairement séparable. Vous trouverez dans [6] la preuve de cette convergence.

Pour illustrer nos propos, nous allons créer un perceptron π_{AND} qui apprend par lui-même la porte logique AND.

Création d'un perceptron AND

Voici le résultat que le perceptron π_{AND} [7] devrait nous donner quand il sera entraîné :

$$\pi_{AND}\left(\begin{bmatrix}1\\0\\0\end{bmatrix}\right) = 0, \quad \pi_{AND}\left(\begin{bmatrix}1\\0\\1\end{bmatrix}\right) = 0,$$
$$\pi_{AND}\left(\begin{bmatrix}1\\1\\0\end{bmatrix}\right) = 0, \quad \pi_{AND}\left(\begin{bmatrix}1\\1\\1\end{bmatrix}\right) = 1.$$

Nous avons donc un ensemble d'exemples qui peut aussi être écrit sous la forme $S = \{(\vec{x_1}, y_1), (\vec{x_2}, y_2), (\vec{x_3}, y_3), (\vec{x_4}, y_4)\}$, où le vecteur $\vec{x_i}$ représente les entrées et y_i représente la valeur désirée. Dans ce cas-ci nous avons :

$$S = \{\left(\begin{bmatrix}1\\0\\0\end{bmatrix}\right), 0), (\left(\begin{bmatrix}1\\0\\1\end{bmatrix}\right), 0), (\left(\begin{bmatrix}1\\1\\0\end{bmatrix}\right), 0), (\left(\begin{bmatrix}1\\1\\1\end{bmatrix}\right), 1)\}.$$

A partir d'ici, le vecteur $\vec{w}$ sera exprimé ainsi :

$$\vec{w_t} = \begin{bmatrix} w_{t,0} \\ w_{t,1} \\ w_{t,...} \\ w_{t,n} \end{bmatrix},$$

où t indique le nombre d'itérations à travers l'ensemble des données.

L'étape 0, dite d'initialisation de la procédure algorithmique est d'initialiser le vecteur $\vec{w}$ à $\vec{0}$:

$$\vec{w_0} = \begin{bmatrix} w_{0,0} \\ w_{0,1} \\ w_{0,2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

7. La porte AND exprime la logique « *Si les deux entrées sont allumées, alors la sortie aussi. Pour tous les autres cas, la sortie est éteinte.* »

Nous pouvons alors commencer l'étape 1 d'itération à travers la base d'exemples S. En voici les résultats :

$$\pi_1(\vec{x_1}) = step(\vec{w_0} \cdot \vec{x_1}) = step\Big(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \Big) = step(0) = 0 \overset{\checkmark}{=} y_1$$

$$\pi_1(\vec{x_2}) = step(\vec{w_0} \cdot \vec{x_2}) = 0 \overset{\checkmark}{=} y_2$$

$$\pi_1(\vec{x_3}) = step(\vec{w_0} \cdot \vec{x_3}) = 0 \overset{\checkmark}{=} y_3$$

$$\pi_1(\vec{x_4}) = step(\vec{w_0} \cdot \vec{x_4}) = 0 \neq y_4$$

Comme le perceptron π_1 s'est trompé, nous devons modifier ses poids. Comme vu dans la description de l'algorithme, lorsque $y_i = 1$ il nous faut augmenter ceux-ci par l'entrée x_i, comme voici :

$$\vec{w_1} = \vec{w_0} + \vec{x_4} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Nous pouvons donc commencer la deuxième itération :

$$\pi_2(\vec{x_1}) = step(\vec{w_1} \cdot \vec{x_1}) = 1 \neq y_1$$

Dès le premier cas nous avons une mauvaise prédiction, nous appliquons donc la méthode de mise à jour des poids. Mais cette fois-ci, vu que $y_1 = 0$ il nous faut soustraire x_1 aux poids actuels :

$$\vec{w_2} = \vec{w_1} - \vec{x_1} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

En continuant dans la même logique, nous obtenons alors une mise à jour des poids qui se fait comme suit :

$$\vec{w_3} = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} \quad \dots \quad \vec{w_5} = \begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix}$$

A partir de la $5^{\text{ème}}$ itération, les poids ne changent plus. En voici la vérification :

$$\pi_{AND}(\vec{x_1}) = step(\vec{w_5} \cdot \vec{x_1}) = step\Big(\begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \Big) = 0 \overset{\checkmark}{=} y_1$$

$$\pi_{AND}(\vec{x_2}) = step\Big(\begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \Big) = 0 \overset{\checkmark}{=} y_2$$

$$\pi_{AND}(\vec{x_3}) = step\Big(\begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \Big) = 0 \overset{\checkmark}{=} y_3$$

$$\pi_{AND}(\vec{x_4}) = step\Big(\begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \Big) = 1 \overset{\checkmark}{=} y_4$$

Nous remarquons effectivement que toutes les données de la base d'exemples sont bien classées, et que donc les poids ne changeront plus.

1.3.4 Interprétation géométrique du perceptron

Comme vu dans la section 1.2, nous sommes à la recherche d'une droite de séparation entre les différentes classes de notre ensemble de données. Dans le cas de la porte logique AND, nous avons donc les points $E(0,0), F(1,0), G(1,1)$ et $H(0,1)$ qui doivent être séparés en deux classes : E, F et H dans la première classe qui a comme valeur 0, et G dans la deuxième classe avec une valeur de 1. Nous voulons représenter un perceptron qui a deux poids et un biais dans un espace à 2 dimensions. Voici donc son équation :

$$w_1 \cdot A + w_2 \cdot B + w_0 \cdot 1 = 0 \tag{1.4}$$

En remplaçant les poids w_0, w_1 et w_2 par ceux du perceptron π_{AND}, nous obtenons cette équation qui est donc une droite :

$$2A + B - 2 = 0 \tag{1.5}$$

En représentant notre ensemble de données S et la droite (1.5) sur un même graphe, nous obtenons la figure 1.10. Nous remarquons que nous avons bel et bien une droite

de séparation entre les différents points. Tous ceux en dessous ou sur la droite[8] valent 0, et ceux au-dessus valent donc 1. Faisons une vérification pour le point E, qui représente donc $\vec{x_1}$.

$$2 \cdot 0 + 0 - 2 = -2$$
$$\text{Et nous avons bien que } step(-2) = 0 \overset{\checkmark}{=} y_1$$

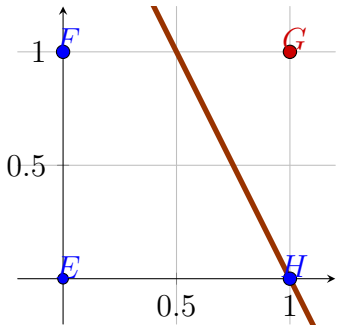


FIGURE 1.10 – Représentation graphique du perceptron AND

Généralisation de la géométrie du perceptron à n dimensions

Dans la section 1.2 et celle précédente, nous avons expliqué qu'il fallait rechercher une droite afin de séparer les différents points dans le plan. Cela est vrai, mais uniquement dans un cas à deux dimensions, où nous avons donc uniquement 2 entrées à notre perceptron. Mais malheureusement, comme dans l'exemple du cours de mathématiques, il y a en général plus que deux entrées. Encore pire, imaginez-vous : pour analyser numériquement une image de 784 pixels, il faudrait 784 entrées à notre perceptron ! Qui dit beaucoup d'entrées dit beaucoup de poids, et si nous voulions représenter l'exemple de cours de mathématiques de la section 1.3.2 dans un espace, celui-ci serait alors en 4 dimensions, et une droite ne peut séparer ces données : nous avons besoin alors d'un hyperplan. Dans un espace de dimensions n de coordonnées $(x_1, x_2, ..., x_n)$, un hyperplan est défini par une équation de la forme :

$$w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n = 0$$

Pour le cas en 3 dimensions, c'est-à-dire lorsqu'on a un perceptron avec trois entrées, cette équation définit alors un plan :

$$w_0 + w_1x_1 + w_2x_2 + w_3x_3 = 0,$$

8. Car nous utilisons la fonction step

où x_1,x_2 et x_3 sont les coordonnées de notre espace[9]. L'algorithme du perceptron va donc nous permettre de trouver cet hyperplan de séparation. Vu que nous utilisons la step fonction, nous pouvons définir alors nos deux classes ainsi :

$$\text{Le point appartient à la classe « 1 » lorsque}$$
$$w_0 + w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n > 0$$

$$\text{Le point appartient à la classe « 0 » lorsque}$$
$$w_0 + w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n \leq 0$$

Nous pouvons donc maintenant traiter la géométrie de n'importe quel perceptron, même si celui-ci a 784 poids, et qu'il faudrait le représenter dans autant de dimensions ! Mais ce perceptron reste tout de même plutôt basique, car il ne permet de séparer que des données binaires et linéairement séparables. Admettons que nous avions plus de deux classes à traiter, comme pour le cas pratique de la classification des chauves-souris que nous développerons par la suite, comment devrions-nous nous y prendre ? Pire encore, imaginez-vous un cas où il nous faudrait au minimum deux lignes pour séparer les deux classes, comme avec le problème de la simulation de la porte logique *XOR* (voir figure 1.11), c'est-à-dire un cas non linéairement séparable, où les individus ne sont pas classifiables grâce à une seule ligne de séparation. Il nous faut alors aller une étape plus loin, et s'attaquer à une évolution de l'algorithme du perceptron que nous allons voir dans le chapitre qui suit.

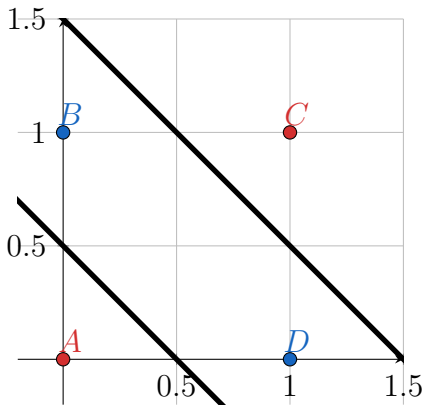


FIGURE 1.11 – Séparation des données de la porte logique *XOR* par deux lignes

9. Dans ce cas-ci, elles sont d'habitude nommées x, y et z

1.4 Perceptron multicouche

Comme vu dans la section précédente, un simple perceptron ne peut pas résoudre de problème non-linéaire, celui-ci ne représentant qu'un seul hyperplan. Ces problèmes étant très courants dans la vie quotidienne, le perceptron aurait été quasi inutile si l'idée d'en connecter plusieurs entre eux n'avait pas émergé. Le perceptron multicouche est donc composé de plusieurs couches, nous permettant de décomposer des problèmes complexes en plusieurs parties, comme par exemple avec la porte logique XOR : celle-ci est le résultat de l'application des portes NAND et OR [10] sur les deux entrées, puis d'une porte AND. Le perceptron de porte logique XOR π_{XOR} nous donne la figure 1.12 lorsqu'il est représenté sous la forme d'un réseau de neurones.

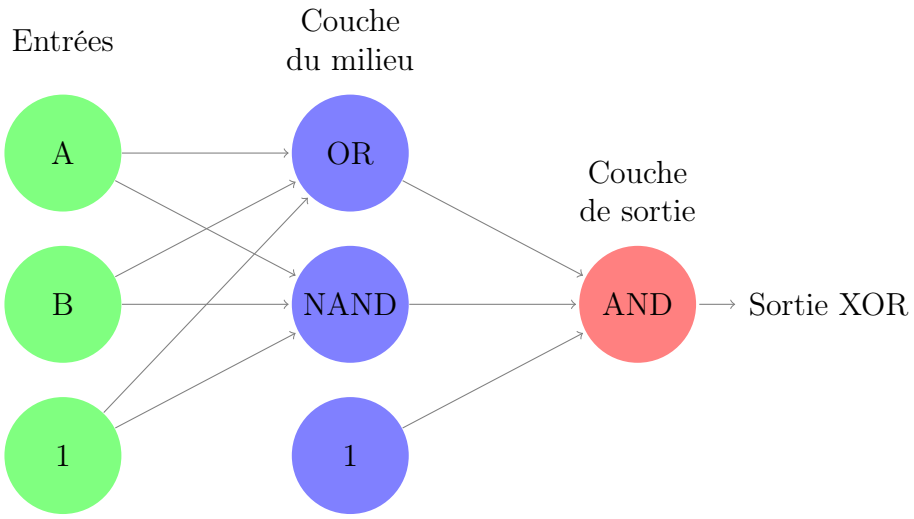


FIGURE 1.12 – Graphe d'un perceptron multicouche qui permet de simuler la porte logique XOR.

Nous connaissons déjà les poids pour la porte logique NAND (voir figure 1.8) et de la porte logique AND (voir figure 1.10). Voici les poids et le biais du perceptron simulant la porte logique OR :

$$\vec{w}_{OR} = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}$$

Pour nous aider dans nos calculs par la suite, nous vous proposons de répartir ces différents poids dans des matrices. Une première matrice est constituée des liens

10. La porte OR exprime la logique "Si les deux entrées sont éteintes, alors la sortie est éteinte. Pour tous les autres cas, la sortie est allumée."

entre les entrées et la couche du milieu : chaque ligne de la matrice est composée des poids de chaque perceptron constituant la couche du milieu. Cette première matrice a donc la forme de :

$$\mathbf{W}_1 = \begin{bmatrix} 1 & 0 & 0 \\ w_{OR,0} & w_{OR,1} & w_{OR,2} \\ w_{NAND,0} & w_{NAND,1} & w_{NAND,2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 1 \\ 3 & -2 & -2 \end{bmatrix} \tag{1.6}$$

Grâce à la notation des transposées[11], nous pouvons simplifier l'équation (1.6) à cette forme :

$$\mathbf{W}_1 = \begin{bmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, & \vec{w}_{OR}, & \vec{w}_{NAND} \end{bmatrix}^T \tag{1.7}$$

En suivant la même logique pour les poids reliant la couche du milieu à la couche de sortie, nous obtenons une seconde matrice de la forme :

$$\mathbf{W}_2 = \begin{bmatrix} w_{AND,0} & w_{AND,1} & w_{AND,2} \end{bmatrix} = \begin{bmatrix} (\vec{w}_{AND})^T \end{bmatrix} = \begin{bmatrix} \vec{w}_{AND} \end{bmatrix}^T = \begin{bmatrix} -2 & 2 & 1 \end{bmatrix}$$

Maintenant que nous avons formalisé tous nos poids, nous vous proposons de vérifier si notre MLP[12] fonctionne comme il se doit. Le calcul peut se décomposer en deux étapes : un premier calcul pour déterminer si les neurones de la couche du milieu s'allument ou non en utilisant les poids $\mathbf{W}_1$, puis une deuxième étape pour savoir si la sortie s'allume ou non en utilisant les poids $\mathbf{W}_2$. Quant à l'entrée, elle est désignée à partir d'ici comme la matrice $\mathbf{X}$ pour permettre une cohérence dans les calculs.

$$\begin{aligned} g(\mathbf{W_1 X}) &= g\Big(\begin{bmatrix} 1 & 0 & 0 \\ w_{OR,0} & w_{OR,1} & w_{OR,2} \\ w_{NAND,0} & w_{NAND,1} & w_{NAND,2} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \Big) \\ &= \begin{bmatrix} g(x_0) \\ g(w_{OR,0} \cdot x_0 + w_{OR,1} \cdot x_1 + w_{OR,2} \cdot x_2) \\ g(w_{NAND,0} \cdot x_0 + w_{NAND,1} \cdot x_1 + w_{NAND,2} \cdot x_2) \end{bmatrix} = \begin{bmatrix} 1 \\ \pi_{OR}(\mathbf{X}) \\ \pi_{AND}(\mathbf{X}) \end{bmatrix}, \end{aligned} \tag{1.8}$$

où $g(x)$ représente la fonction d'activation ($g(x_0) = 1$ car x_0 vaut toujours 1).

11. Pour rappel, l'opération de la transposée se définit ainsi : les colonnes deviennent des lignes et vice-versa.

12. perceptron multicouche. Se référer au glossaire pour plus d'informations.

Nous utiliserons la notation $\boldsymbol{\pi}(\mathbf{X}_i)$ pour désigner l'opération réalisée par un MLP sur une entrée X_i. Les poids $\mathbf{W}_2$ prennent en entrée les valeurs des neurones de la couche du milieu que nous venons de calculer. En utilisant un calcul analogue à l'équation (1.8), nous pouvons supposer que le réseau nous donnera alors la sortie suivante :

$$\boldsymbol{\pi}(\mathbf{X}_i) = g(\mathbf{W}_2 \cdot g(\mathbf{W}_1\mathbf{X}_i)), \tag{1.9}$$

Comme nous allons utiliser ici aussi la fonction en escalier, voici la définition de celle-ci quand elle s'applique sur une matrice colonne :

$$\mathbf{step}\left(\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}\right) = \begin{bmatrix} step(x_0) \\ step(x_1) \\ \vdots \\ step(x_n) \end{bmatrix}$$

Cette définition nous permet de développer l'équation (1.9) sous la forme

$$\boldsymbol{\pi}(\mathbf{X}_i) = \mathbf{step}(\mathbf{W}_2 \cdot \mathbf{step}(\mathbf{W}_1\mathbf{X}_i)). \tag{1.10}$$

Voici la vérification pour le cas où A vaut 1, et B vaut 0. La sortie devrait donc avoir la valeur de 1 [13]. La matrice d'entrée $\mathbf{X}_1 = (1, 1, 0)^T$, nous pouvons donc calculer le résultat suivant :

$$\boldsymbol{\pi}(\mathbf{X}_1) = step(\mathbf{W}_2 \cdot \mathbf{step}\left(\begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 1 \\ 3 & -2 & -2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}\right))$$

$$= step(\mathbf{W}_2 \cdot \mathbf{step}\left(\begin{bmatrix} 1 \\ 0.5 \\ 1 \end{bmatrix}\right)) = step\left(\begin{bmatrix} -2 & 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}\right)$$

$$= step(1) \overset{\checkmark}{=} 1$$

Voici la vérification d'un des quatre cas possibles pour le MLP qui simule la porte logique XOR. En utilisant la même méthode de calcul, nous pouvons vérifier si

13. car XOR(1,0) = 1, se référer à la figure 1.12

celui-ci fonctionne effectivement sur tous les cas possibles :

$$\boldsymbol{\pi}(\mathbf{X}_2) = step(\mathbf{W}_2 \cdot \mathbf{step}\left(\begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 1 \\ 3 & -2 & -2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \right)) = step(1) \overset{\checkmark}{=} 1$$

$$\boldsymbol{\pi}(\mathbf{X}_3) = step(\mathbf{W}_2 \cdot \mathbf{step}\left(\begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 1 \\ 3 & -2 & -2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right)) = step(-1) \overset{\checkmark}{=} 0$$

$$\boldsymbol{\pi}(\mathbf{X}_4) = step(\mathbf{W}_2 \cdot \mathbf{step}\left(\begin{bmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 1 \\ 3 & -2 & -2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right)) = step(0) \overset{\checkmark}{=} 0$$

Nous voilà donc avec un nouveau modèle nous permettant de simuler n'importe quel problème, qu'il soit linéaire ou non.

1.4.1 Classification multi-classe

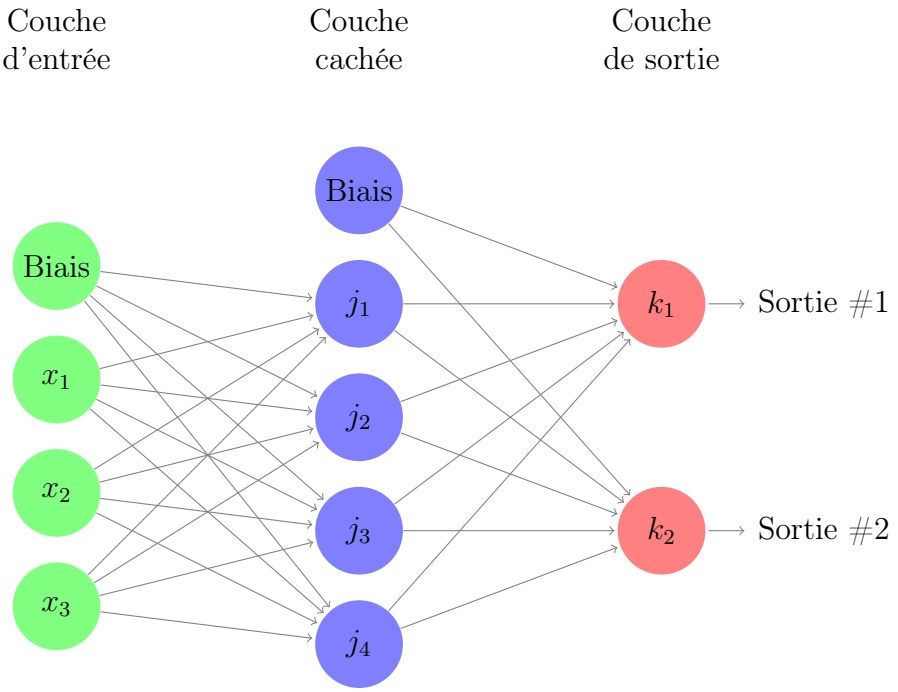


FIGURE 1.13 – Graphe d'un MLP à 3 entrées, une couche cachée à 4 neurones, et 2 sorties.

En extrapolant l'exemple de la section précédente, nous pouvons donner une forme générale à un MLP, comme le montre la figure 1.13 qui se décompose en trois types de couches différentes : la couche d'entrée, la ou les couche(s) cachée(s) et la couche

de sortie. Les poids qui relient la couche d'entrée à la couche cachée sont représentés par la matrice $\mathbf{W}_1$ et ceux qui relient la couche cachée à la couche de sortie par la matrice $\mathbf{W}_2$. Vous remarquerez que cette nouvelle forme nous permet d'avoir au minimum deux sorties différentes, ici représentées par k_1 et k_2. Deux sorties impliquent donc que nous pouvons traiter des cas où il faut faire la distinction entre 4 classes différentes, c_1, c_2, c_3 et c_4 :

$$c_1 : \begin{bmatrix} k_1 = 0 \\ k_2 = 0 \end{bmatrix} \quad c_2 : \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad c_3 : \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad c_4 : \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Plus le nombre de sorties est grand, plus le nombre de classes pouvant être séparées l'est aussi. Si nous devons classifier C classes, nous devons alors au minimum avoir $\log_2 C$ neurones de sortie. Cependant, cette méthode de *codage* de la sortie ne permet pas au réseau d'être efficace lors de l'apprentissage [14], c'est pour cela que nous préférerons interpréter la couche de sortie d'un MLP de la manière suivante :

$$c_1 : \begin{bmatrix} k_1 = 1 \\ k_2 = 0 \\ k_3 = 0 \end{bmatrix} \quad c_2 : \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad c_3 : \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

où chaque neurone de la couche de sortie représente donc une classe différente.

1.4.2 Fonction d'activation sigmoïde

Jusqu'à maintenant, les différents réseaux créés se contentaient d'entrées et de sorties binaires, ce qui limite grandement leur application. Pour pouvoir élargir celle-ci, il nous faut changer de fonction d'activation. Pour l'instant, nous avons utilisé la fonction dite *en escalier*, comme expliqué dans la section 1.3. Celle-ci avait été choisie car elle permettait de simuler le comportement d'un neurone biologique et de condenser toutes les valeurs en deux classes uniques [15] : les *0* et les *1*. Nous devons donc introduire ici une nouvelle fonction : la fonction sigmoïde, que nous noterons σ. Celle-ci est définie par l'équation suivante :

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{1.11}$$

14. Voir [CMB18] pour plus d'informations
15. Celles-ci représentant respectivement l'état de *repos* et l'état *on fire* d'un neurone.

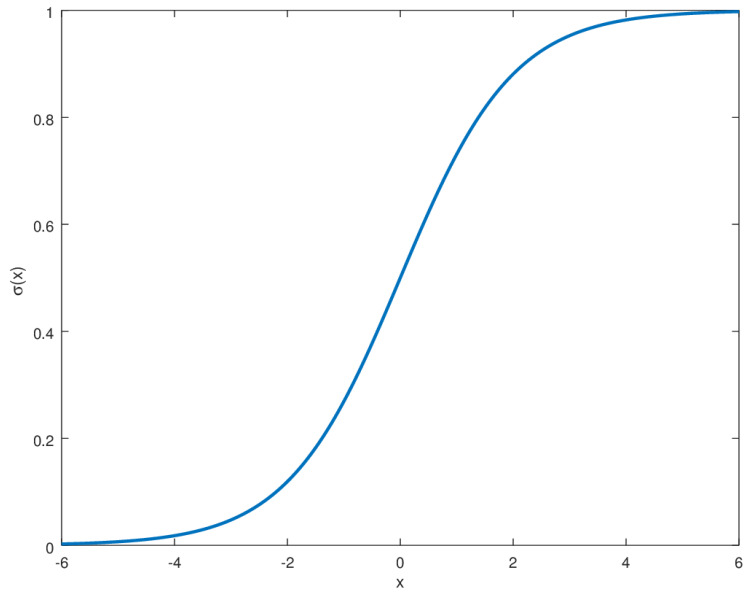


FIGURE 1.14 – Fonction sigmoïde.

Comme le montre la figure 1.14, la fonction sigmoïde ressemble à la fonction *step*. Ceci va permettre aux réseaux créés avec la fonction sigmoïde de conserver les mêmes propriétés intéressantes que ceux qui ont la fonction *step* comme fonction d'activation. La sortie d'un perceptron utilisant cette nouvelle fonction d'activation sera donc la suivante :

$$\pi(\vec{x}) = \sigma(\vec{w} \cdot \vec{x}) \tag{1.12}$$

À partir de la définition de la fonction sigmoïde de l'équation (1.11), nous pouvons donc l'écrire sous la forme suivante :

$$\pi(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x})}} \tag{1.13}$$

Pour la sortie d'un perceptron multicouche, nous pouvons poser l'équation suivante à partir de l'équation (1.9) :

$$\boldsymbol{\pi}(\mathbf{X}) = \sigma(\mathbf{W}_2 \cdot \sigma(\mathbf{W}_1 \mathbf{X})) \tag{1.14}$$

À partir d'ici, nous utiliserons l'équation (1.14) comme référence pour le calcul de $\boldsymbol{\pi}(\mathbf{X})$.

Nouvelles applications possibles : probabilités

Pour faire sens de cette nouvelle fonction, nous vous proposons d'utiliser à nouveau l'analogie du cours de maths faite dans la section 1.3.2. Le perceptron utilisant la fonction d'activation *step* nous permet de prendre la décision binaire suivante : aller en cours, ou ne *pas* y aller. En simulant la même prise de décision, mais cette fois-ci avec un perceptron utilisant la fonction d'activation σ, nous pouvons alors quantifier l'envie que nous avons d'aller en cours de maths. La fonction σ ne donnant que des valeurs dans l'intervalle entre 0 et 1, la sortie du perceptron peut être interprétée comme une probabilité, un taux d'envie. En reprenant les mêmes poids que dans l'exemple de la section 1.3.2, nous pouvons donc obtenir ce type de prédiction :

$$\pi\left(\begin{bmatrix}1\\0\\0\\1\\1\end{bmatrix}\right) = \sigma\left(\begin{pmatrix}\textit{biais}\\\textit{intérêt}\\\textit{bon}\\\textit{motivé}\\\textit{malade}\end{pmatrix}\begin{bmatrix}1\\0\\0\\1\\1\end{bmatrix} \cdot \begin{bmatrix}-7\\5\\2\\10\\-5\end{bmatrix}\right) = \sigma(-2) \cong 0.119,$$

qui nous indique donc que nous avons une envie de $11,9\%$ d'aller en cours de maths selon ce modèle.

Pour un réseau avec un minimum de deux sorties, nous pouvons interpréter le résultat des neurones de sorties de la manière qui suit.

Nous avons trois classes différentes, c_1,c_2 et c_3 :

$$c_1 : \begin{bmatrix}k_1 = 1\\k_2 = 0\\k_3 = 0\end{bmatrix} \quad c_2 : \begin{bmatrix}0\\1\\0\end{bmatrix} \quad c_3 : \begin{bmatrix}0\\0\\1\end{bmatrix}$$

Et voici la sortie d'un MLP qui est conçue pour les classifier :

$$\boldsymbol{\pi}(\mathbf{X}) = \begin{bmatrix}0.642\\0.215\\0.143\end{bmatrix}$$

Chaque neurone de sortie donnant la probabilité qu'un individu appartienne à la classe respective à celui-ci, nous pouvons interpréter la sortie $\boldsymbol{\pi}(\mathbf{X})$ ci-dessus de la manière suivante : il y a 64,2% de chance que l'individu appartienne à la classe c_1, 21,5% pour la classe c_2 et 14,3% pour la classe c_3.

1.4.3 Apprentissage

Comme vu dans la section 1.3.3, l'apprentissage artificiel peut se réaliser grâce à l'*exemple*. L'algorithme d'apprentissage du perceptron multicouche repose lui aussi sur ce concept. À la différence du perceptron utilisant la fonction d'activation *step*, nous avons besoin de deux nouvelles notions afin de faire apprendre un MLP : l'erreur de classification, et l'impact qu'a chaque poids sur cette erreur.

Erreur de classification

Afin d'expliquer le concept d'erreur de classification, nous vous proposons de prendre un exemple. Nous avons un MLP qui pour l'instant n'est pas entraîné, et une base d'exemple qui a la forme $S = \{(\mathbf{X}_1, \mathbf{Y}_1), (\mathbf{X}_2, \mathbf{Y}_2), ..., (\mathbf{X}_n, \mathbf{Y}_n)\}$, où $\mathbf{X}_i$ est la matrice d'entrée, $\mathbf{Y}_i$ la sortie désirée et n le nombre d'exemples dans la base de données. Son objectif est de classifier trois classes différentes, c_1, c_2 et c_3. Avec le MLP que nous utilisons en exemple, nous obtenons actuellement la sortie suivante pour l'entrée $\mathbf{X}_1$:

$$\boldsymbol{\pi}(\mathbf{X_1}) = \begin{matrix}\text{Probabilité } c_1 \\ \text{Probabilité } c_2 \\ \text{Probabilité } c_3\end{matrix} \begin{bmatrix}0.32 \\ 0.31 \\ 0.42\end{bmatrix} \text{ alors que la sortie désirée } \mathbf{Y}_1 \text{ est } \begin{bmatrix}0 \\ 0 \\ 1\end{bmatrix}.$$

En représentant $\boldsymbol{\pi}(\mathbf{X_1})$ et $\mathbf{Y}_1$ dans un espace en 3 dimensions, nous obtenons la figure 1.15. Le vecteur qui relie les deux points représente alors l'*erreur de classification*. Voici le calcul des composantes de ce vecteur, que nous nommerons $\vec{E}$:

$$\vec{E} = \begin{bmatrix}0 \\ 0 \\ 1\end{bmatrix} - \begin{bmatrix}0.32 \\ 0.31 \\ 0.42\end{bmatrix} = \begin{bmatrix}-0.32 \\ -0.31 \\ 0.58\end{bmatrix}$$

En calculant le carrée la norme de ce vecteur, nous pouvons donc avoir une valeur pour l'erreur de classification :

$$E = (-0.32)^2 + (-0.31)^2 + (0.58)^2 = 0.5349$$

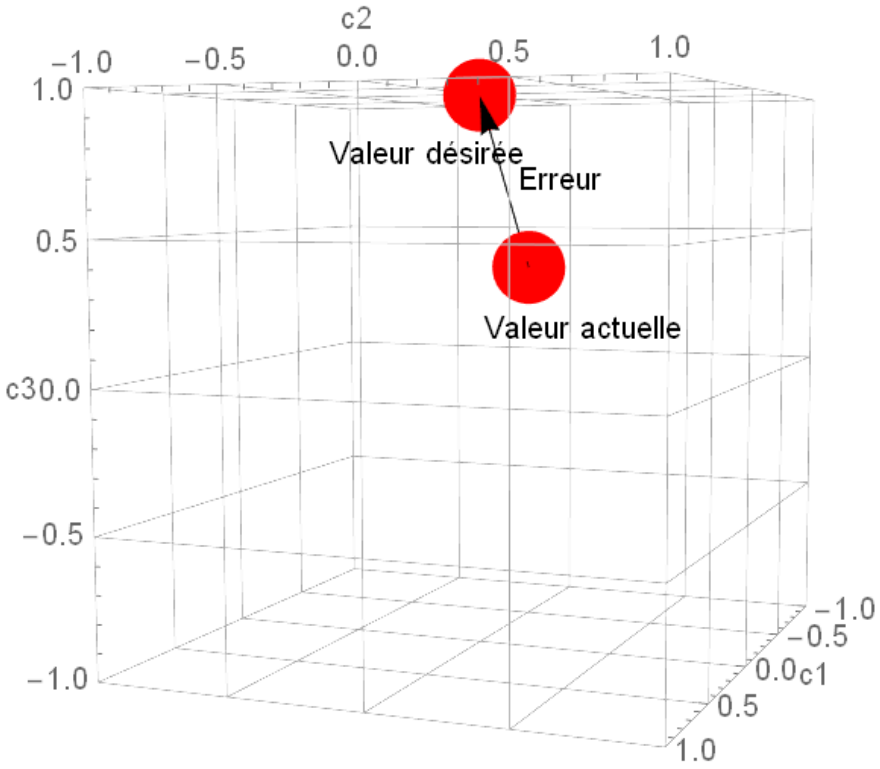


FIGURE 1.15 – Représentation de l'erreur dans l'espace de sortie.

Nous pouvons généraliser cet exemple, et fixer l'erreur de classification pour un exemple $(\mathbf{X}_i, \mathbf{Y}_i)$ avec C classes à classifier, comme ceci :

$$E(\vec{w}, \mathbf{X}_i, \mathbf{Y}_i) = \sum_{l=1}^{C} (c_l \text{ prédit} - c_l \text{ actuel})^2. \tag{1.15}$$

Descente de gradient

Dans la section 1.3.3, nous avons justifié l'algorithme d'apprentissage du perceptron ainsi : « à chaque fois que le perceptron se trompe, nous lui tapons sur les doigts : nous corrigeons ses poids ». Maintenant que nous avons *quantifié* cette erreur, nous pouvons corriger les poids en fonction de l'erreur de classification E. L'algorithme d'apprentissage consiste alors à réduire cette erreur sur chaque exemple, afin que l'erreur de classification *globale* de l'ensemble des données soit réduite. En langage plus mathématique, nous pouvons donc traduire cela par « rechercher le minimum d'une fonction ». Ce que nous pouvons changer pour réduire cette erreur – et donc minimiser la fonction d'erreur– ce sont les poids de notre modèle et il nous faut alors comprendre comment ceux-ci impactent l'erreur. Si nous reformulons, nous pouvons

dire que le problème qui s'offre à nous est le suivant : « Comment le changement de valeur d'un poids w_i impacte le changement de valeur de E ? ». En répondant à cette question, nous pourrions donc réduire l'erreur en changeant correctement le poids w_i. Pour trouver la réponse à cette question, nous vous proposons de la traduire en langage mathématique :

$$\frac{\partial E(\vec{w}, \mathbf{X}_i, \mathbf{Y}_i)}{\partial w_i} = ? \tag{1.16}$$

Ce que nous recherchons effectivement est la dérivée partielle de l'erreur E en fonction d'un poids w_i. Il nous suffit donc de trouver la réponse à la question (1.16) pour pouvoir comprendre comment changer la valeur de w_i afin de minimiser E.

Pour mieux comprendre la notion de dérivée partielle, nous vous proposons d'observer la figure 1.16. Celle-ci représente en 3 dimensions une fonction à deux variables $f(x, y)$. En prenant uniquement une tranche de la courbe de la fonction représentée, comme le plan rose par exemple, nous pouvons alors observer comment la fonction change selon y. Réciproquement, en tranchant la courbe avec le plan bleu, nous pouvons observer comment la fonction change selon x. Les lignes noires représentent les tangentes qui ont comme pente respectivement la dérivée partielle de x et de y en ce point.

Le vecteur formé par les deux dérivées partielles d'une fonction à deux variables est appelé *gradient*, et il est représenté mathématiquement par le symbole ∇. Il représente donc la pente d'un point donné de cette fonction. Le gradient de la fonction précédente se note donc ainsi :

$$\nabla f(x, y) = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix}$$

Pour mieux comprendre ce que signifie le *gradient*, nous vous proposons l'analogie qui suit : admettons que nous nous retrouvons perdu en montagne, et qu'il y a un épais brouillard, comment atteindre le bas de la montagne, sachant que nous ne pouvons voir qu'à 5 mètres autour de nous ? Il nous suffit de regarder la pente de la surface autour de nous, et se diriger vers dans la direction la plus raide. Voilà comment fonctionne la méthode qui s'appelle *descente de gradient*. En calculant le gradient d'une fonction en un point, nous pouvons donc savoir dans quelle direction et sens sa variation est maximale, et simplement suivre le sens opposé pour atteindre le minimum de la fonction [16].

16. Mais celle-ci doit remplir certains critères, se référer à [CMB18] pour plus d'informations.

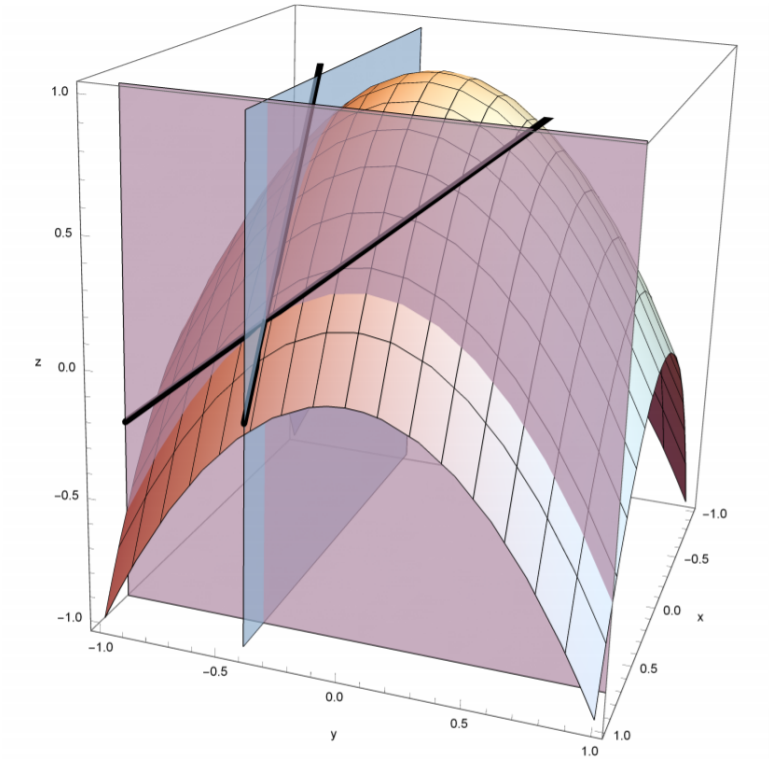


FIGURE 1.16 – Fonction à deux variables, où ses dérivées partielles sont les pentes des tangentes.

Application de la descente de gradient

Pour une approche plus visuelle, nous pouvons observer la figure 1.17 qui montre comment suivre le gradient permettant d'atteindre le minimum de la fonction. Dans cette figure, les axes du graphique représentent les poids w_0 et w_1 d'un perceptron qui a pour but de simuler la condition logique *not*[17]. La fonction représentée est alors le calcul de l'erreur de classification *globale* $E(w_0, w_1)$ sur l'ensemble de la base de données S selon les valeurs de w_0 et w_1. Voici l'ensemble de la base de donnée :

$$S = \{(\begin{bmatrix}1\\0\end{bmatrix}, 1), (\begin{bmatrix}1\\1\end{bmatrix}, 0)\}$$

17. Cette porte logique prend une seule valeur en entrée, et exprime la logique suivante : « Si 1, alors 0. Si 0, alors 1. »

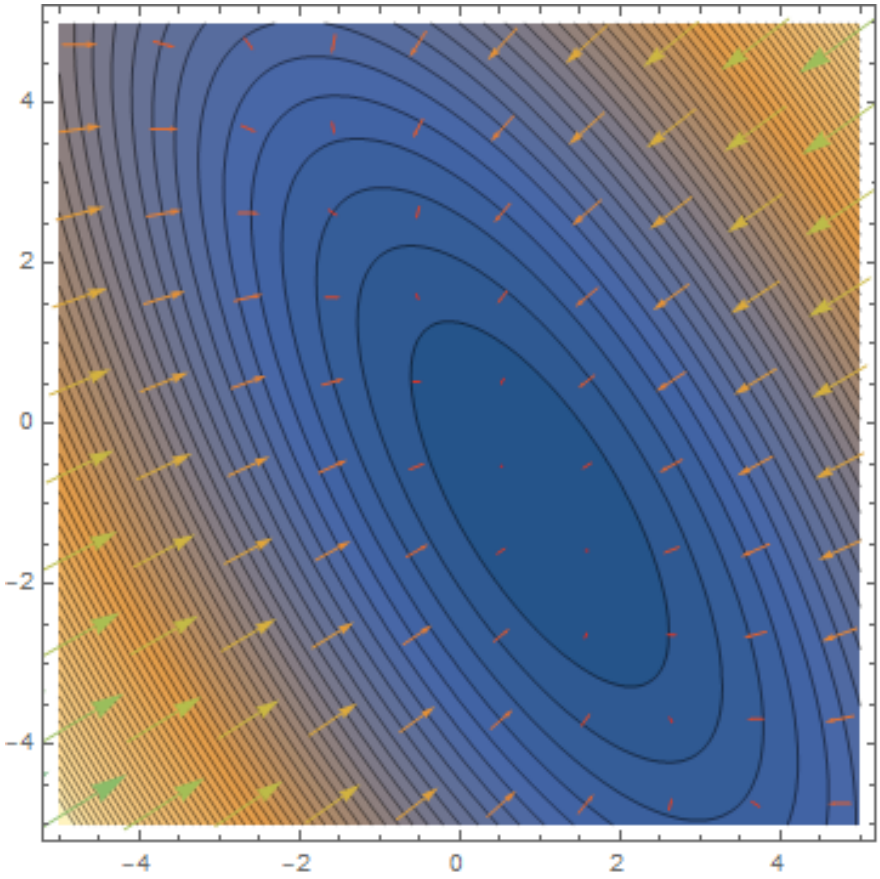


FIGURE 1.17 – Lignes de niveau de $E(w_0, w_1)$ et gradients de $E(w_0, w_1)$ en différents points.

Pour éviter de tomber dans un minimum *local*[18] de la fonction d'erreur, nous initialisons les poids de manière aléatoires et vérifions que le résultat final était effectivement le minimum de la fonction en répétant l'ensemble de la procédure décrite ci-dessous à plusieurs reprises.

Nous initialisons donc les poids de manière aléatoire. Voici la valeur de w_0 et w_1 ainsi que la sortie actuelle de $\pi(\vec{x})$ pour chaque exemple d'apprentissage :

$$\vec{w} = \begin{bmatrix} -3 \\ -2 \end{bmatrix} \quad \pi(\vec{x_1}) \cong 0.0474 \quad \pi(\vec{x_2}) \cong 0.0067$$

Pour les mettre à jour, il nous faut donc appliquer la méthode de la descente de gradient. Nous savons que le changement qui doit être appliqué au vecteur $\vec{w}$ est le

18. Nous cherchons effectivement le minimum *global* de la fonction. Si nous reprenons l'analogie de la montagne, un minimum local représente un simple creux, alors que nous cherchons à nous rendre au bas de la vallée.

gradient de descente, nous pouvons donc poser l'équation suivante :

$$\Delta\vec{w} = -\nabla E(w_0, w_1) \tag{1.17}$$

Le gradient ne donnant que la direction à prendre, il nous faut choisir la grandeur du pas à faire à chaque itération. La grandeur de ce pas ne sera pas constante, car la norme du gradient devient de plus en plus petite lorsqu'on s'approche du minimum de la fonction. Nous allons nommé cette constante α, et plus généralement « taux d'apprentissage ». À partir de l'équation (1.17), nous pouvons alors poser l'équation suivante :

$$\vec{w}_{t+1} = \vec{w}_t - \alpha\nabla E(w_0, w_1) \tag{1.18}$$

Le calcul de ces dérivées partielles étant assez conséquent, se référer à [CMB18] pour le calcul complet. En voici le résultat :

$$\vec{w}_{t+1} = \vec{w}_t - \alpha \cdot (\vec{y_i} - \pi(\vec{x_i})) \cdot \pi(\vec{x_i}) \cdot (1 - \pi(\vec{x_i})) \cdot \vec{x_i} \tag{1.19}$$

À partir de l'équation (1.19), nous pouvons mettre à jour nos poids de manière à minimiser la fonction d'erreur. Nous choisissons un α de 1 pour l'exemple.

Les poids obtenus après 49 itérations sont les suivants :

$$\vec{w}_{49} = \begin{bmatrix} 7.9454 \\ -11.0696 \end{bmatrix}$$

Nous remarquons effectivement que cette méthode nous permet de trouver une combinaison de poids optimaux pour atteindre un minimum de la fonction d'erreur :

$$\pi(\vec{x_1}) = \sigma\Big(\begin{bmatrix}1\\0\end{bmatrix} \cdot \begin{bmatrix} 7.9454 \\ -11.0696 \end{bmatrix}\Big) \cong 0.9996 \overset{\checkmark}{\cong} 1$$

$$\pi(\vec{x_2}) = \sigma\Big(\begin{bmatrix}1\\1\end{bmatrix} \cdot \begin{bmatrix} 7.9454 \\ -11.0696 \end{bmatrix}\Big) = 0.0421 \overset{\checkmark}{\cong} 0$$

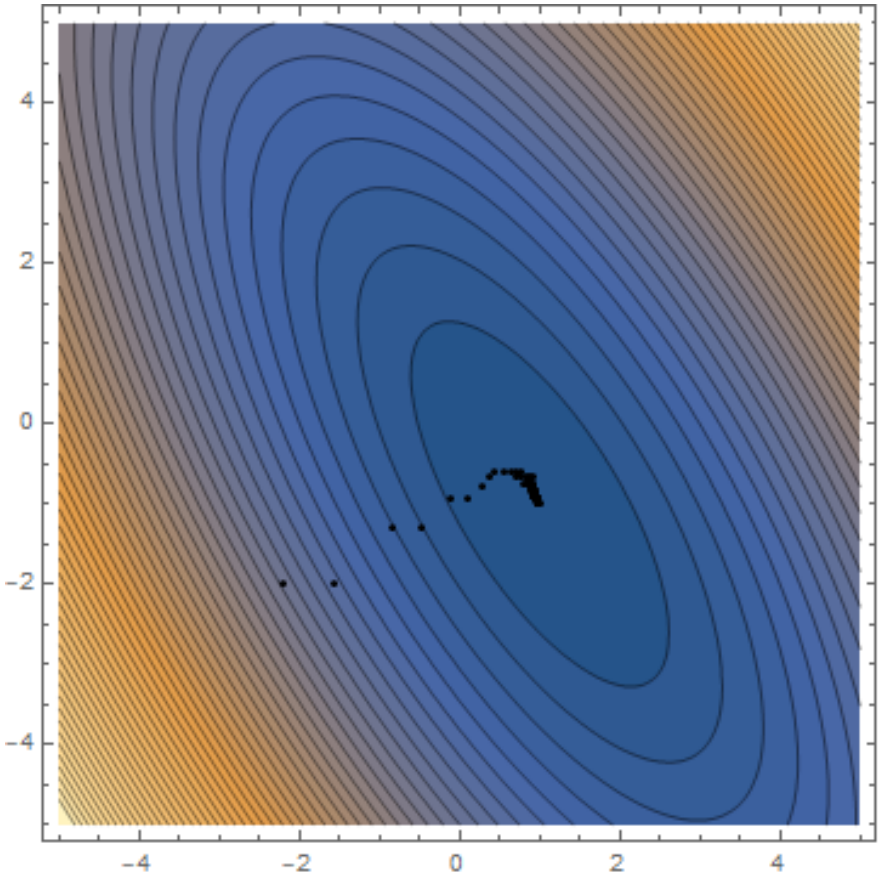


FIGURE 1.18 – Points représentant l'évolution des poids.

Le résultat [19] de la mise à jour des poids se trouve dans le figure 1.18.

Nous avons donc réussi à avoir un algorithme d'apprentissage artificiel, le perceptron multicouche, qui peut apprendre à partir d'une base de données d'exemples à classifier des problèmes non-linéaires et possédant plusieurs classes différentes.

Taux d'apprentissage

Nous avons dû introduire le taux d'apprentissage lors du développement réalisé dans la section précédente. Celui-ci étant un paramètre important de l'apprentissage, il est intéressant de comprendre son impact sur celui-ci. La figure 1.19 montre à quel point il est important : si celui-ci est trop grand, l'algorithme divergera et les poids n'atteindront jamais le minimum de la fonction. Pour réduire un maximum le

19. Le lecteur aguerri remarquera effectivement que ce graphique ne représente pas les résultats de la mise à jour des poids avec la fonction d'activation sigmoïde, mais bien avec la fonction d'activation identité ($f(x) = x$). Cela rend le graphique plus intuitif.

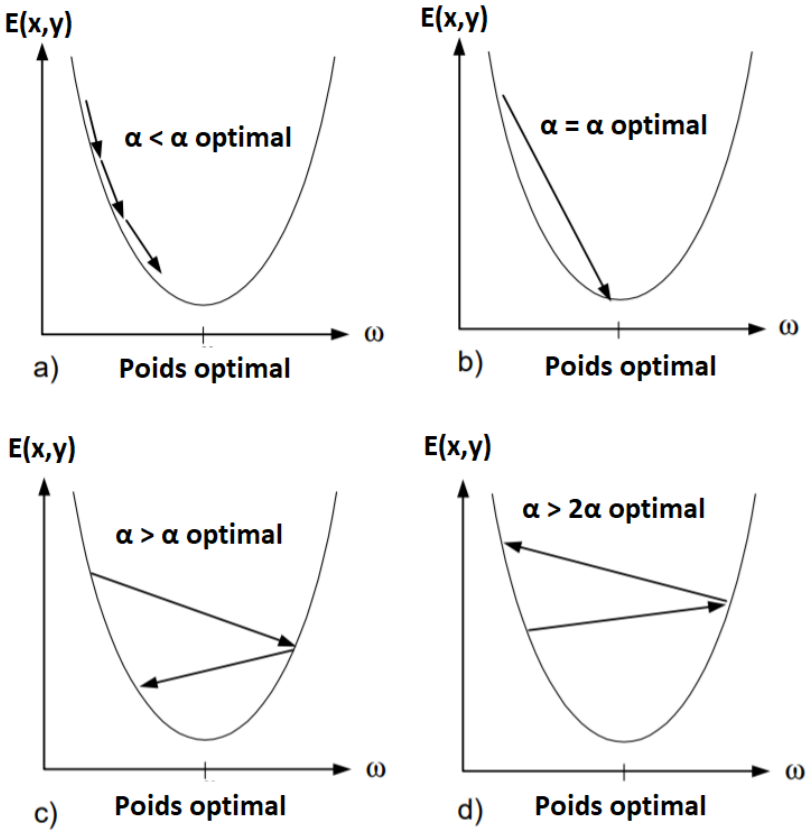


FIGURE 1.19 – Variation du poids w selon α. [LBBOM98]

nombre d'itérations de l'algorithme, il faut donc trouver l'alpha α le plus proche du *alpha* α dit optimal, qui nous permettrait de minimiser la fonction de la manière plus efficace possible. Pour trouver cette valeur d'*alpha* α, il n'existe malheureusement pas de théorie nous permettant de la calculer. Il faudra donc le choisir au cas par cas de manière empirique. Une valeur standard, utilisée par convention, de celui-ci est de 0.001. Nous étudierons plus précisément l'impact de celui-ci sur un exemple réel dans le chapitre 3.

Chapitre 2

Écologie acoustique des chauves-souris et base de données fournie par *FRIbat-CCO*

Les chauves-souris sont dotées d'une particularité étonnante : elles volent pendant la nuit (même la plus obscure). Cette spécificité est due à une adaptation : les chauves-souris ont développé un sonar performant qui leur permet d'utiliser la méthode de l'écholocation ou l'écholocalisation, pour s'orienter et chasser leurs proies (généralement des insectes).

Cette méthode permet aux chauves-souris de se faire une image sonore de leur environnement en émettant des cris à haute fréquence (entre 10 et 120 kHz pour les espèces du canton de Fribourg), elles en captent les échos avec leurs grandes oreilles, les analysent et construisent une image tridimensionnelle de l'environnement, en fonction de l'intensité des échos et du temps qu'ils ont mis pour revenir.

Les caractéristiques de ces cris, comme la fréquence, la durée, l'intervalle entre ceux-ci, varient non seulement d'une espèce à l'autre, mais aussi au sein d'une seule et même espèce, selon le type de vol et l'environnement qui l'entoure.

L'objectif de ce chapitre est alors de définir les différents groupes acoustiques que nous allons utiliser pour la mise en application. En premier lieu, nous allons donc définir ce qu'est l'écologie acoustique en elle-même ainsi que l'usage de l'écholocation et expliquer la lecture des sonogrammes utilisés. En deuxième partie, nous allons différencier les trois groupes acoustiques des chauves-souris d'après certains documents reçus. Finalement, la dernière partie de ce chapitre est, quant à elle, centrée sur l'analyse des différentes variables de notre base de données ainsi que le processus

de collecte des données fournie par *FRIbat-CCO*[1].

2.1 Définition de l'écologie acoustique

"L'écologie acoustique est l'étude des relations d'un être vivant et de son environnement sonore". [16]

Dans notre cas, l'être vivant étudié est la chauve-souris ou le chiroptère. Nous allons donc par ce chapitre établir une analyse détaillée des cris émis par ces mammifères et dans quel type d'environnement ils sont utilisés.

2.2 Sonar des chauves-souris

Les chauves-souris produisent des sons de la même façon que nous, en déplaçant l'air à travers leurs cordes vocales, ce qui les fait vibrer. Certaines chauves-souris émettent des sons de leur bouche, qu'elles tiennent ouverte en volant. D'autres émettent des sons par le nez. Les scientifiques ne comprennent pas tout à fait comment fonctionne cette technique, mais ils croient que la structure étrange du nez de certaines d'entre elles sert à concentrer le bruit pour mieux repérer les insectes et autres proies.

Ce son se comporte de la même manière que nos cris : c'est une onde qui rebondit sur tout objet qu'elle rencontre. Les chauves-souris produisent donc une onde sonore et écoutent attentivement son écho avec leurs grandes oreilles. Le cerveau de la chauve-souris traite les informations de telle manière qu'il peut déterminer le temps qu'il a fallu pour que le son revienne, et ainsi calculer la distance qui la sépare de l'obstacle ou la proie. Afin d'avoir une meilleure compréhension de cette méthode, la figure 2.1 la conceptualise. Cette méthode est appelée l'écholocation et peut être comparée à celle utilisée par un sonar d'un sous-marin pour localiser des navires ou quand un homme crie au sommet d'un canyon et qu'il perçoit son écho, comme si quelqu'un lui répondait.

Le sonar de la chauve-souris est si performant qu'elle peut reconnaître si sa proie est à droite ou à gauche en comparant le moment où le son atteint son oreille droite à celui où le son atteint son oreille gauche : si le son de l'écho atteint celle de droite avant d'atteindre celle de gauche, la proie est évidemment à droite. Ses oreilles sont

1. *FRIbat-CCO* est un groupe fribourgeois d'étude et protection des chauves-souris qui s'occupe aussi de sensibiliser le grand public à la protection de cet animal et d'assurer sa conservation.

donc dotées d'une multitude de plis qui l'aident à déterminer la position verticale de sa proie. Les échos venant d'en bas toucheront les plis de l'oreille externe à un point différent des sons venant d'en haut, et ils sonneront donc différemment quand ils atteindront l'oreille interne de la chauve-souris.

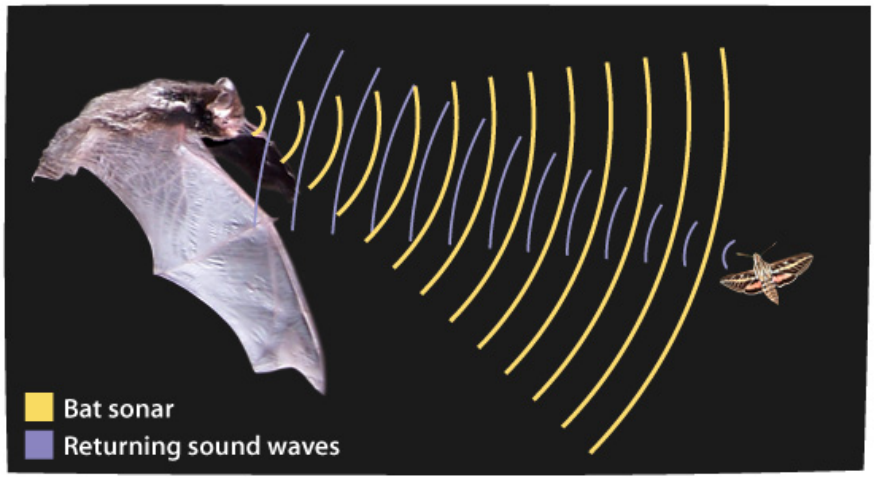


FIGURE 2.1 – Représentation de l'usage de l'écholocation d'une chauve-souris pour chasser une proie. Les sons émis par la chauve-souris sont représentés par les ondes sonores jaunes, les ondes violettes montrent les ondes sonores qui se réfléchissent sur la proie. [23]

La chauve-souris peut même connaître la taille et la direction de sa victime en fonction de l'intensité et de la fréquence de l'écho. L'onde sonore qu'elle émet réfléchira moins sur un petit objet et produira donc une résonance moins forte. Et si la proie s'éloigne de la chauve-souris, l'écho de retour aura un ton plus grave que le son original, tandis que l'écho d'un animal se déplaçant vers la chauve-souris aura un son plus aigu. Cette différence est dûe à l'effet Doppler [2].

La chauve-souris traite toutes ces informations inconsciemment, de la même manière que nous traitons les informations visuelles et sonores que nous recueillons avec nos yeux et nos oreilles. Une chauve-souris forme une image d'écholocalisation dans sa tête comme l'image que nous formons dans notre tête d'après nos informations visuelles. Les chauves-souris traitent également l'information visuelle, contrairement à ce que nous croyons, la plupart des chauves-souris ont une vision assez aigue. Elles utilisent l'écholocalisation en conjonction avec la vision, et non pas au lieu de celle-ci.

2. Quand un objet émet un son à une certaine fréquence et qu'il se rapproche du récepteur, celui-ci recevoir l'onde sonore avec une fréquence plus élevée et percevra alors un plus plus aigu. De façon réciproque, si l'émetteur s'éloigne du récepteur, celui-ci percevra un son plus grave.

2.3 Sonogramme d'un cri de chauve-souris

Cette section est séparée en trois parties : la première consiste à définir la notion de son et de fréquence sonore, la deuxième porte sur l'explication d'un sonogramme et la troisième présente les stratégies utilisées d'écholocation.

2.3.1 Son

Un son peut être composé de plusieurs ondes sonores de fréquences différentes. Une onde sonore est une variation de pression qui se propage dans un milieu compressible (comme l'air, l'eau ou toute autre matière liquide ou solide). La source de cette perturbation est un objet qui cause une vibration, comme les cordes vocales d'une personne. Cette vibration perturbe les particules dans le milieu environnant ; elles perturbent celles qui se trouvent à côté d'elles, et ainsi de suite. La perturbation crée un mouvement vers l'extérieur sous forme de vagues, comme lorsqu'une goutte d'eau tombe dans un lac. L'onde transporte l'énergie sonore à travers le milieu, généralement dans toutes les directions et de façon moins intense lorsqu'elle s'éloigne de la source.

2.3.2 Fréquence d'un son

La fréquence d'un son est la mesure du nombre de vibrations par seconde émises par l'objet qui le produit : si le nombre est petit, nous percevons un son grave. Au contraire, si le son est aigu, cela veut dire que le nombre de vibrations est élevé.

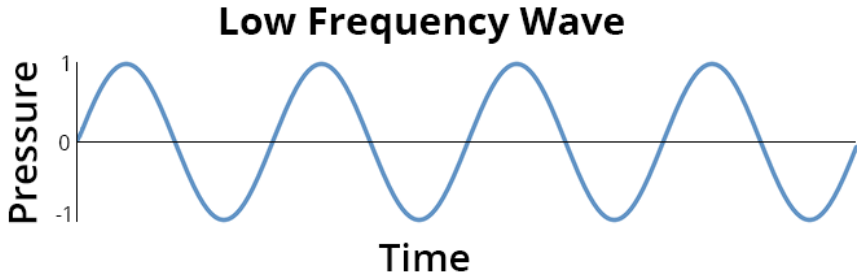


FIGURE 2.2 – Onde sonore de basse fréquence en fonction de la pression et du temps, exprimée en seconde. [15]

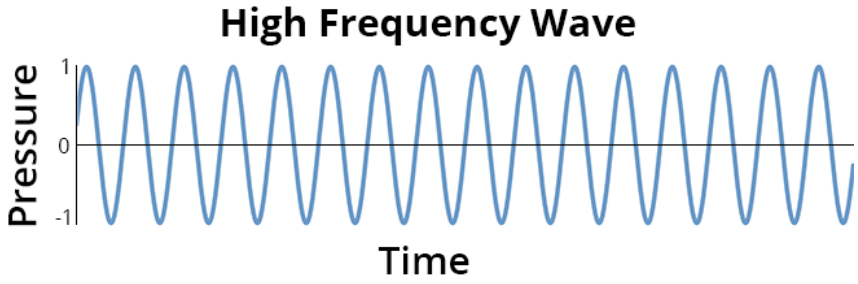


FIGURE 2.3 – Onde sonore de haute fréquence en fonction de la pression et du temps, exprimée en seconde. [15]

Pour la plupart des chiroptères, le son qu'ils émettent a une fréquence très élevée, si élevée que nous les humains ne pouvons percevoir leurs cris [3].

2.3.3 Sonogramme

Le sonogramme est une représentation graphique d'un signal sonore d'après sa fréquence et son évolution dans le temps. D'autres caractéristiques peuvent apparaître sur un sonogramme, comme l'amplitude. Celle-ci peut être représentée par une variation de couleurs de la courbe retraçant le son.

2.3.4 Stratégies d'écholocation

Nous pouvons distinguer deux types de stratégies dans l'usage que les chauves-souris adoptent avec leur sonar :

(1) cris émis pour des obstacles proches 2.4

(2) cris émis pour des obstacles lointains 2.5

D'un premier point de vue, les deux sonogrammes peuvent paraître effrayants, cependant il est très simple de les comprendre. Les traits dégradés correspondent à des cris de chauves-souris et sont placés de telle manière que l'axe horizontal traduit le temps du son émis (exprimé en milliseconde) tandis que l'axe vertical traduit la fréquence (en kilohertz) ; cette unité est expliquée au point 2.3.2. La nuance de couleurs représentée sur les figures 2.4 et 2.5 montre l'amplitude [4] du son d'une chauve-souris

3. Les fréquences que les hommes peuvent percevoir vont de 20 à 20 000 Hz.
4. L'amplitude est la variation de la pression du son : moins le son est fort, moins l'amplitude est importante

(plus la couleur est bleue, plus l'amplitude est forte) et l'axe des ordonnés fait référence à la fréquence du cri.

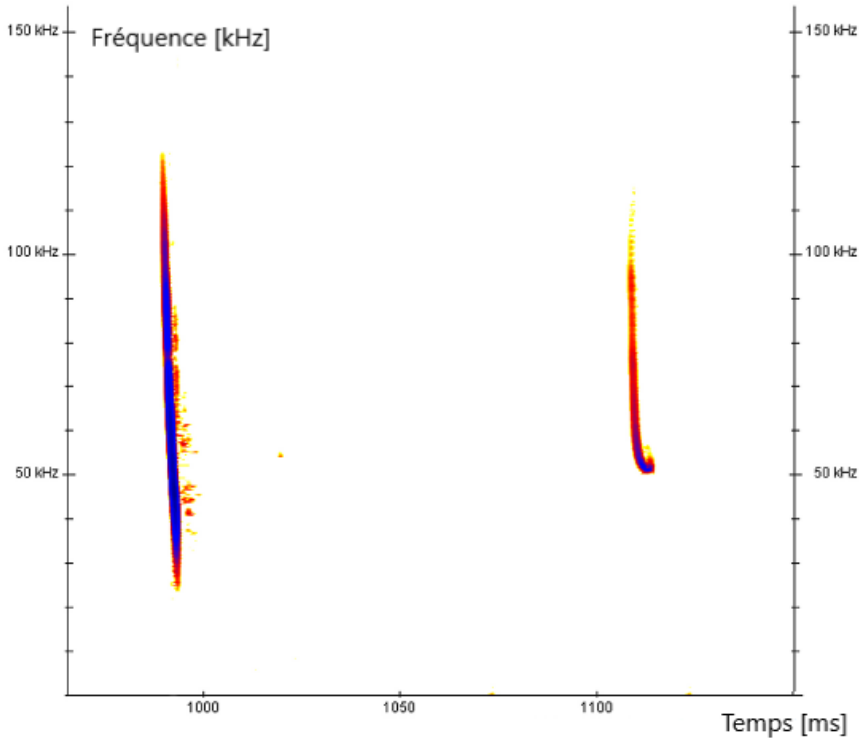


FIGURE 2.4 – Première stratégie pour localiser des obstacles (ou proies) proches. Le cri est très bref et a une fréquence moyenne élevée.

La figure 2.4 correspond aux cris que lance la chauve-souris afin de reconnaître des obstacles proches lors de son vol. Cette stratégie consiste à émettre des ultrasons en balayant une vaste largeur de bande [5] (plus de 50 kHz) et avec une fréquence moyenne élevée. La chauve-souris collecte ainsi de nombreuses informations de très bonne qualité, mais de courte portée. Ce sont alors les espèces chassant dans des environnements semi-ouverts voire fermés, comme les oreillards et les murins, qui recourent à cette stratégie.

À l'inverse, la figure 2.5 correspond aux cris que lance la chauve-souris afin de reconnaître des obstacles éloignés. Les cris sont alors de fréquence plutôt basse (de 10 à 30 kHz) sur une faible largeur de bande (moins de 5 kHz), ce qui a comme conséquence d'avoir un son de longue portée, selon les lois de l'acoustique. Mais en contrepartie, la précision et la qualité de l'information données par l'écho sont

5. La largeur de bande est la différence entre les deux fréquences limites, dans notre cas, lors d'un seul cri.

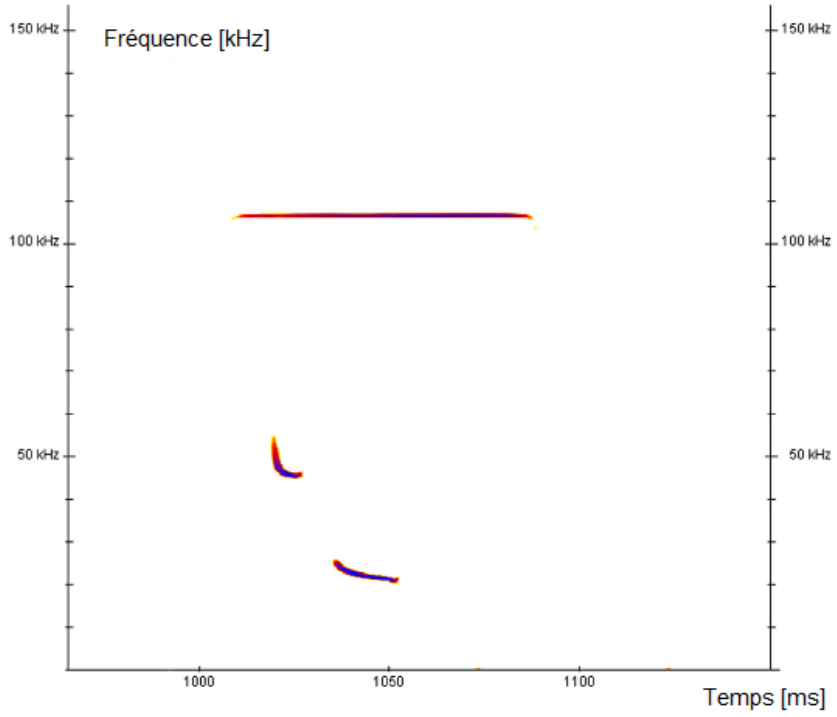


FIGURE 2.5 – Deuxième stratégie pour localiser des obstacles (ou proies) lointains. Le cri a une fréquence moyenne basse et ont une faible largeur de bande.

moindres. Ce sont alors les espèces de haut vol comme les noctules, les sérotines et les pipistrelles qui sont plus favorables à utiliser cette technique.

Il est important de relever que chaque espèce a recours aux deux tactiques présentées, cependant certaines espèces utiliseront plus une stratégie que l'autre d'après le milieu dans lequel elles opèrent.

2.4 Base de données

Afin de mieux comprendre les données utilisées pour la mise en application de notre travail de maturité, nous allons faire mention dans cette section du processus de la collecte de données, du matériel utilisé lors de celle-ci ainsi que des différentes variables qu'elle comporte. De plus, cette section portera sur les trois grandes espèces recensées lors de la collecte des données et leur distinction. Les principales sources pour cette section ont été envoyées par le groupe *FRIbat-CCO*.

2.4.1 Processus de la collecte des données

Afin de mieux comprendre les données utilisées pour la mise en application de notre travail de maturité, il est essentiel de faire mention du processus de la collecte des données et du matériel utilisé pour celle-ci. Nous rappelons que l'ensemble des données vient uniquement de la région du Lac de Pérolles, dans le canton de Fribourg, en Suisse.

Toutes les illustrations utilisées dans cette section 2.4.1 nous ont été transmises par M. Jérôme Gremaud qui est membre du groupe d'étude et de protection des chauves-souris dans le canton de Fribourg *FRIbat-CCO*. Il a été alors convenu qu'elles soient strictement utilisées dans le cadre de ce travail de maturité afin de mieux conceptualiser le processus de collecte des informations liées à la base de données.

Matériel utilisé

Premièrement, les systèmes d'enregistrement qui ont été utilisés par *FRIbat-CCO* pour collecter les données des cris des chauves-souris sont le *BATLOGGER M* et le *D500X*[6].

Hormis leur nomenclature différente, ces deux appareils ont la même fonction, celle d'enregistrer en temps réel les ultrasons émis par les chiroptères. Les deux appareils ont plus ou moins la même plage de sensibilité sonore, ils captent des sons qui peuvent avoir une fréquence de 10 kHz à 150 kHz pour le premier et de 15 kHz à 190 kHz pour le second. Les deux systèmes sont conçus pour des observations à long terme ; ils peuvent généralement être laissés sur le terrain pendant plusieurs mois. Le système de déclenchement permet aux appareils de démarrer automatiquement l'enregistrement d'un cri, c'est-à-dire que dès qu'un son perçu dépasse le niveau de fréquence choisi, l'enregistrement commence.

En plus de l'enregistrement normal des cris des chauves-souris, d'autres données et informations sont collectées et stockées. La caractéristique la plus importante est le récepteur GPS intégré sur le système du *BATLOGGER M*. Chaque enregistrement est référencé par les coordonnées de sa position. Ces informations sont particulièrement précieuses dans les installations complexes avec plusieurs appareils d'enregistrement.

6. Toutes les informations transmises sont basées sur deux sites et référencé par Jérôme Gremaud, membre de l'organisation qui nous a envoyé la base de données, pour plus d'informations se référer à [2] et [7].

FIGURE 2.6 – Systèmes d'enregistrement utilisés pour la collecte de données.

Après avoir collecté toutes ces données, il a fallu les trier, les analyser, etc. Pour ce faire, *FRIbat-CCO* a utilisé le logiciel *BatScope 3*. Ce logiciel a été conçu par l'institut fédéral suisse de recherche WSL [7]. Cet institut a comme mission d'étudier les modifications de l'environnement ainsi que la protection des habitats naturels, de surveiller l'état et l'évolution des environnements (les forêts, les paysages, la biodiversité).

Ce logiciel a ainsi permis d'analyser les spectrogrammes [8] et les cris ainsi que les classer en espèce à l'aide de différents algorithmes. Il a aussi permis de suivre l'évolution des positions d'enregistrements GPS du système *BATLOGGER M* avec Google Maps et finalement d'exporter les résultats, les classifications et les enregistrements pour les partager avec d'autres utilisateurs. Pour résumer, *BatScope 3* a permis de rassembler les informations, les trier pour, en dernier lieu, les partager de façon à ce qu'ils forment une base de données claire et complète.

Déroulement

La première étape du processus du recueillement des informations est la mise en place des différents systèmes d'enregistrement présentés auparavant. Comme le montre les

7. *Die Eidgenössische Forschungsanstalt für Wald, Schnee und Landschaft WSL*, qu'on pourrait traduire par l'institut fédéral suisse de recherches sur la forêt, la neige et le paysage

8. Un spectogramme une représentation picturale d'un spectre sonore, ici les ultrasons des chauves-souris

figures 2.7 et 2.8, ces systèmes ont été installés à différents endroits à proximité du Lac de Pérolles.

FIGURE 2.7 – Mise en place d'un des systèmes d'enregistrement des ultrasons émis par les chauves-souris dans les hauteurs du Lac de Pérolles.

Ces appareils ont été ainsi laissés pendant un certain temps afin de rassembler le plus de données possibles, puis ont été récupérés pour récolter ces dernières.

La seconde étape de la procédure n'est plus sur le terrain, mais en laboratoire à Fribourg. Elle a comme visée de rassembler les informations, d'analyser les spectogrammes, de trier les données et ainsi de former une base de données claire et complète. Comme cité à la section 2.4.1, le logiciel *BatScope 3* a fortement aidé pour cette étape ; il est important de rappeler que la base de données comporte plus de 450'000 lignes.

La raison pour laquelle le groupe d'étude et de protection *FRIbat-CCO* a choisi cette région est assez évidente. En effet, le Lac de Pérolles (voir figure 2.9) est doté d'une biodiversité conséquente tant au niveau de la flore que de la faune. Plus de 570 espèces florales, dont 72 menacées, ont été observées dans le périmètre de la réserve[9] ainsi que 68 espèces d'oiseaux, dont 27 sont menacées, et rien que 8 espèces

9. Le Lac de Pérolles, qui est un lac artificiel et situé aux portes de la ville de Fribourg fait l'objet d'une réserve naturelle afin de pouvoir mieux préserver la faune et la flore présente en ces lieux. Cette réserve constitue également un rôle économique dans la production de bois et d'énergie hydraulique.

FIGURE 2.8 – Mise en place d'un des systèmes d'enregistrement des ultrasons émis par les chauves-souris au niveau du Lac de Pérolles.

de chiroptères, la pipistrelle commune étant la seule qui ne soit pas menacée.

Une autre raison, plus pratique, est la proximité de cette région avec l'emplacement du laboratoire du groupe fribourgeois de protection et d'étude des chauves-souris, basé dans la ville de Fribourg. Cela a permis d'avoir une sécurité et une surveillance plus appuyée sur les différents systèmes mis en place pour permettre le bon fonctionnement de l'opération.

2.4.2 Différents groupes acoustiques analysés

Nous allons maintenant présenter les différents groupes acoustiques qui se trouvent dans la base de données fournie par *FRIbat-CCO* : le murin, la pipistrelle et la noctule. Le sonogramme de chaque groupe acoustique est présenté dans la figure 2.10.

Murin

Le premier groupe correspond aux murins, le nom usuel donné au genre *Myotis*. Ces chauves-souris doivent leur nom à la similitude de leurs oreilles avec celles des souris

FIGURE 2.9 – Réserve naturelle du Lac de Pérolles.

(en grec *myotis* signifie « oreilles de souris ») .

En analysant leur sonogramme, nous constatons tout de suite que ces types de chauves-souris utilisent la première stratégie, celle pour localiser les obstacles proches et leur nourriture (principalement des insectes terrestres). Ceci est dû à un environnement de chasse très bas et encombré ; leur territoire de chasse est semé d'embûches (ex. roches, buissons, arbres,...). En effet, toutes ces espèces, dont le genre de vie est nocturne, dorment le jour dans les grottes, les puits de mine, les greniers, ou toute autre sorte de cavité sombre, ce qui aura par la suite un impact sur la stratégie qu'elles utilisent pour se repérer la nuit. De plus, le murin se distingue par la haute fréquence de son cri (qui peut atteindre plus de 100kHz) et sa durée très courte.

Pipistrelle

Le deuxième groupe correspond aux pipistrelles (le genre *Pipistrellus*). Leur mode de vie est relativement similaire au groupe précédent, mais elles ont un sonogramme très différent. En effet, d'après la figure 2.10, leur cri est moins aigu que celui des murins (seulement un maximum de 65 kHz), et sa durée est plus longue. Une autre différence est l'amplitude du cri : la pipistrelle émet un son plus doux et avec une variation de volume plus grande que le murin. Ceci est dû à un environnement de chasse légèrement moins encombré et situé à une altitude plus haute (voir la

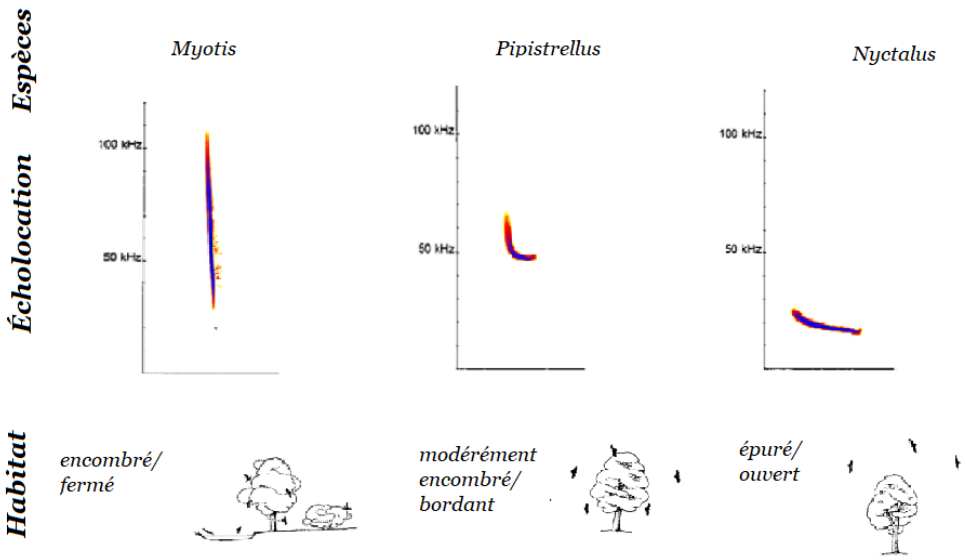


FIGURE 2.10 – Sonogrammes du murin, de la pipistrelle et de la noctule (de gauche à droite). [RUS12]

figure 2.10). Les pipistrelles capturent majoritairement des petits papillons et des moustiques.

Noctule

Le troisième et dernier groupe correspond aux noctules (le genre *Nyctalus*). Ces chauves-souris sont parmi les plus grandes d'Europe. Leur habitat est alors différent des deux autres genres. Elles préfèrent les endroits ouverts et se retrouvent dans les forêts, les grands parcs, les pâturages et les grands jardins. Elles chassent principalement des gros insectes et même parfois des petits oiseaux.

Les caractéristiques de l'écholocation des noctules sont très différentes des murins et des pipistrelles. La durée d'un cri peut aller jusqu'à 30 millisecondes (ce qui est considérable comparé aux autres) et la fréquence est de même beaucoup moins grande (seulement 30 kHz).

2.4.3 Variables de la base de données

Cette section a pour but de clarifier et d'expliquer la base de données fournie par *FRIbat-CCO* que nous allons utiliser dans la partie applicative. La figure 2.11 est une capture d'écran d'une partie de cette base de données, celle-ci contenant 405'126 cris répartis en 14'959 séquences. Dans la partie *mise en pratique* de ce travail, notre objectif sera de donner une espèce à chaque cri, et non à chaque séquence. Les raisons de ce choix se trouvent dans la section 3.4.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	SequenceUU	CallUUID	FrequenceM	FrequenceM	Duree	FrequenceP	BandWith	IntervaPre	IntervalPos	SNR	AnyClassific	ManualClass	ManualClass
2	DF6FD39D-26	34F8EB44-44	56	45.013	11.04	45.624	10.987	0	162.6112	35.845	Fail	Pip.pipi	
3	DF6FD39D-26	9AE52AAC-79	53.864	44.403	11.36	45.013	9.461	162.6112	294.5024	32.942	Fail	Pip.pipi	
4	DF6FD39D-26	0D5C9291-47	56	44.403	19.36	45.013	11.597	294.5024	287.5392	48.717	Fail	Pip.pipi	
5	DF6FD39D-26	73B7E2B2-F5	53.864	44.098	18.4	45.929	9.766	287.5392	214.6304	48.797	Fail	Pip.pipi	
6	DF6FD39D-26	51F544E4-EF	53.864	43.488	17.28	44.098	10.376	214.6304	100.352	32.186	Fail	Pip.pipi	
7	DF6FD39D-26	5CBE0DFA-34	52.338	43.793	8.96	44.403	8.545	100.352	107.7248	55.799	Fail	Pip.pipi	
8	DF6FD39D-26	F607434A-52	53.253	43.182	10.72	44.098	10.071	107.7248	103.2192	38.418	Fail	Pip.pipi	
9	DF6FD39D-26	6F6792F9-47	49.591	43.182	11.84	44.098	6.409	103.2192	99.1232	49.923	Fail	Pip.pipi	
10	DF6FD39D-26	DF1D4991-AF	53.864	43.793	9.44	44.098	10.071	99.1232	90.9312	39.301	Fail	Pip.pipi	
11	E3354780-FF	439D54D4-3D	74.005	46.844	7.36	49.591	27.161	0	266.6496	38.314	Fail	Pip.nath	Pip.pipi
12	E3354780-FF	360EA38B-AA	54.169	45.929	15.68	47.15	8.24	266.6496	337.92	40.003	Fail	Pip.nath	Pip.pipi
13	E3354780-FF	85B67C07-23	39.52	37.994	9.92	38.91	1.526	337.92	320.3072	48.23	Pass	Pip.nath	Pip.pipi
14	E3354780-FF	AE6ECE8D-17	42.267	38.3	9.6	40.131	3.967	320.3072	380.5184	40.602	Pass	Pip.nath	Pip.pipi
15	E3354780-FF	57B0BF6B-98	39.52	37.994	12.64	38.605	1.526	380.5184	206.0288	47.961	Fail	Pip.nath	Pip.pipi
16	E3354780-FF	23EB8D81-99	39.825	38.3	10.08	39.52	1.525	206.0288	111.0016	49.781	Pass	Pip.nath	Pip.pipi
17	E3354780-FF	D6EB092A-C3	40.131	38.3	12.48	39.52	1.831	111.0016	314.9824	53.75	Fail	Pip.nath	Pip.pipi
18	E3354780-FF	AF257D5D-18	40.131	38.91	8.8	39.52	1.221	314.9824	190.0544	46.904	Pass	Pip.nath	Pip.pipi
19	E3354780-FF	D099F466-A2	39.52	37.994	8.32	39.215	1.526	190.0544	303.5136	63.771	Pass	Pip.nath	Pip.pipi
20	E3354780-FF	A65F6D7A-07	39.52	37.994	10.08	39.52	1.526	303.5136	195.7888	69.492	Pass	Pip.nath	Pip.pipi
21	E3354780-FF	6193A3D7-D8	39.52	38.3	12.48	39.215	1.22	195.7888	193.7408	65.209	Fail	Pip.nath	Pip.pipi
22	F441893D-13	83A805D0-6C	54.779	46.234	15.52	47.15	8.545	206.0288	241.664	46.781	Fail	Pip.pipi	
23	F441893D-13	35E4B91E-87	52.948	46.234	18.08	46.844	6.714	241.664	100.352	46.959	Fail	Pip.pipi	
24	F441893D-13	B4C7370A-69	52.948	46.539	12.48	47.455	6.409	100.352	86.4256	33.398	Fail	Pip.pipi	
25	F441893D-13	F9A25A51-06	58.746	46.234	10.4	47.15	12.512	86.4256	103.2192	48.214	Fail	Pip.pipi	
26	F441893D-13	0DC9427D-BF	58.746	46.234	13.76	46.844	12.512	103.2192	103.2192	40.46	Fail	Pip.pipi	
27	F441893D-13	3E366463-D1	58.441	46.234	11.36	46.844	12.207	103.2192	94.6176	50.373	Fail	Pip.pipi	
28	F441893D-13	DD601F4E-91	53.558	47.15	3.36	48.37	6.408	94.6176	98.304	47.06	Pass	Pip.pipi	
29	F441893D-13	80CCE67F-A4	63.324	46.539	14.24	47.15	16.785	98.304	98.7136	51.578	Fail	Pip.pipi	
30	F441893D-13	8717AFDC-A0	53.253	46.844	14.72	47.15	6.409	98.7136	147.456	37.953	Fail	Pip.pipi	

FIGURE 2.11 – Segment de la base de donnée de FRIbat-CCO Fribourg.

Les colonnes A et B donnent les identifiants de chaque séquence et de chaque cri. Ces informations ne sont pas utiles pour la mise en application car nous utiliserons notre propre moyen d'indexation. Les colonnes C et D donnent la fréquence maximale et minimale d'un cri en kilohertz. La colonne E contient la durée des cris en millisecondes. La colonne F donne la fréquence où l'amplitude est la plus forte.

La colonne G contient les largeurs de bande [10] des cris, c'est-à-dire la différence entre la fréquence maximale (FrequenceMax, la colonne C) et minimale (FrequenceMin, la colonne D) de celui-ci. Elle est mesurée en kilohertz.

La colonne J contient le rapport signal/bruit (abrégé SNR, signal-to-noise ratio). Ce rapport compare un niveau de puissance de signal à un niveau de puissance de

10. La traduction anglaise de ce terme est *bandwith*

bruit, exprimé en décibels [11]. Plus le nombre de ce rapport est élevé, meilleure est la qualité, puisqu'il y a plus d'informations utiles (le signal) qu'il n'y a de données indésirables (le bruit). Un rapport supérieur à 1 : 1 indique qu'il y a un niveau de puissance de signal plus grand que celui de bruit. Par exemple, un appareil indique un rapport signal/bruit de 20 dB [12] (décibels), cela signifie que le signal est de 20 dB supérieur au niveau du bruit. Dans la base de données, ce rapport est aussi exprimé en décibels.

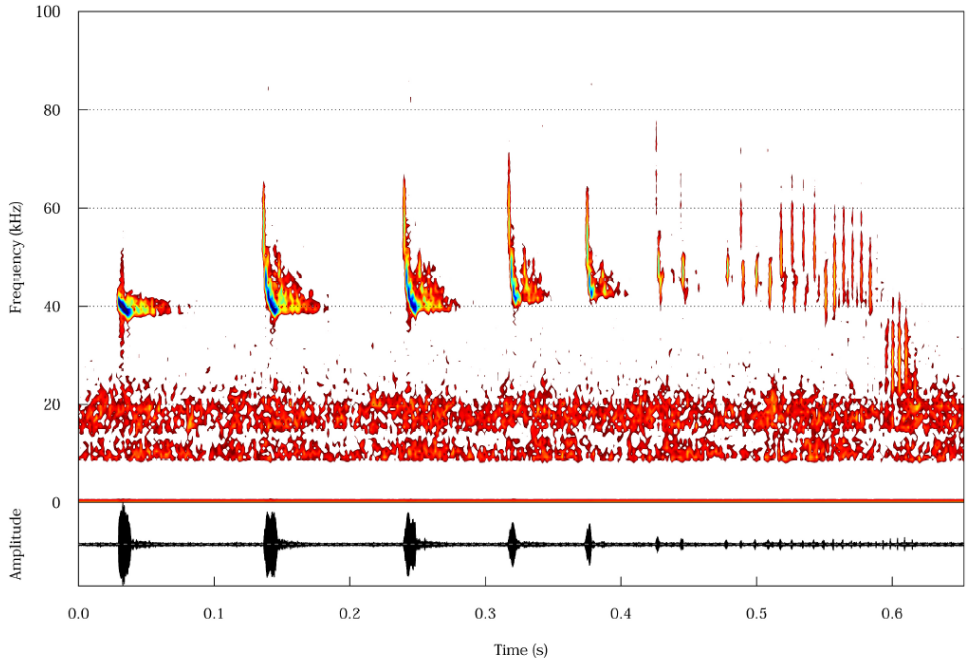


FIGURE 2.12 – Graphe représentant la variation des intervalles entre chaque cri émis par une chauve-souris, la partie bleue correspond au moment où l'amplitude est la plus forte. La figure décrit un moment de chasse.

Les colonnes H et I contiennent la durée en millisecondes entre chaque cri. La colonne H contient le temps écoulé avant le cri et la colonne I contient la durée de temps écoulé après celui-ci. En effet, les cris des chauves-souris enregistrés dans la base de données sont en séquence : l'animal n'émet pas qu'un seul son mais plusieurs et à un intervalle varié, comme le montre la figure 2.12.

La colonne K permet de savoir si un cri appartient réellement à une chauve-souris ou non : *pass* si c'est le cas et *fail* si ça ne l'est pas. Les biologistes de *FRIbat-CCO* ont sorti des critères permettant d'estimer si un ultrason provient d'une chauve-souris

11. L'unité utilisée pour mesurer le niveau sonore
12. Se référer au glossaire pour plus d'informations

ou non en analysant les données. Ils ont donc créé un test statistique à partir de cette analyse, et nous nous reposerons donc sur celui-ci en ne prenant en compte que les cris qui sont considérés comme appartenant à une chauve-souris.

La colonne L contient la classification semi-manuelle d'un cri, c'est-à-dire que ce sont les biologistes de *FRIbat-CCO* qui en analysant le spectogramme de la séquence ont vérifié si l'espèce proposée par BatScope était correcte dans les cas jugés peu fiables par le logiciel. Pour ce qui est de la colonne M, elle est utilisée lorsque sur une même séquence il y a deux individus de deux espèces différentes qui crient en même temps. Ce cas étant très rare, nous avons décidé de l'ignorer pour la partie applicative.

2.4.4 Analyse des variables selon les groupe acoustiques

Les graphes suivants montrent la relation qu'a chaque groupe acoustique avec les différentes variables que nous allons utiliser pour la création du modèle de classification. Nous utilisons ici les notations suivante : Pip. pour pipistrelle, Myo. pour murin et Nyc. pour noctule. L'aire sous les courbes représente la quantité totale des chauves-souris. Ils ont été réalisés grâce au logiciel d'analyse statistique Orange 3 [DCE+13].

Même si nous allons utiliser toutes ces variables pour la classification, il est intéressant de les juger selon leur importance qu'elles auront dans la classification. Pour cela, il faut observer l'intervalle qu'occupe chaque groupe acoustique dans un graphe : plus celui-ci est grand, moins le critère est efficace pour le groupe concerné.

Par exemple, en observant la figure 2.13, nous pouvons constater que le critère du SNR n'est pas efficace pour distinguer les pipistrelles des deux autres groupes. Par contre, il peut être utile pour distinguer un murin d'une noctule, l'intervalle des deux groupes ne se chevauchant que sur une zone réduite. Il en va ainsi de même pour d'autres critères comme le pic de fréquence (voir figure 2.16), l'amplitude (voir la figure 2.17) et la durée (voir la figure 2.18). D'après la figure 2.15, le maximum de la fréquence est un critère valable pour la distinction entre une pipistrelle et une noctule.

Un meilleur critère de classification est le minimum de la fréquence, comme le montre la figure 2.14. Celui-ci est très utile pour distinguer les noctules des deux autres groupes, l'intervalle dans lequel se situe le groupe ne chevauchant pas celui des deux autres.

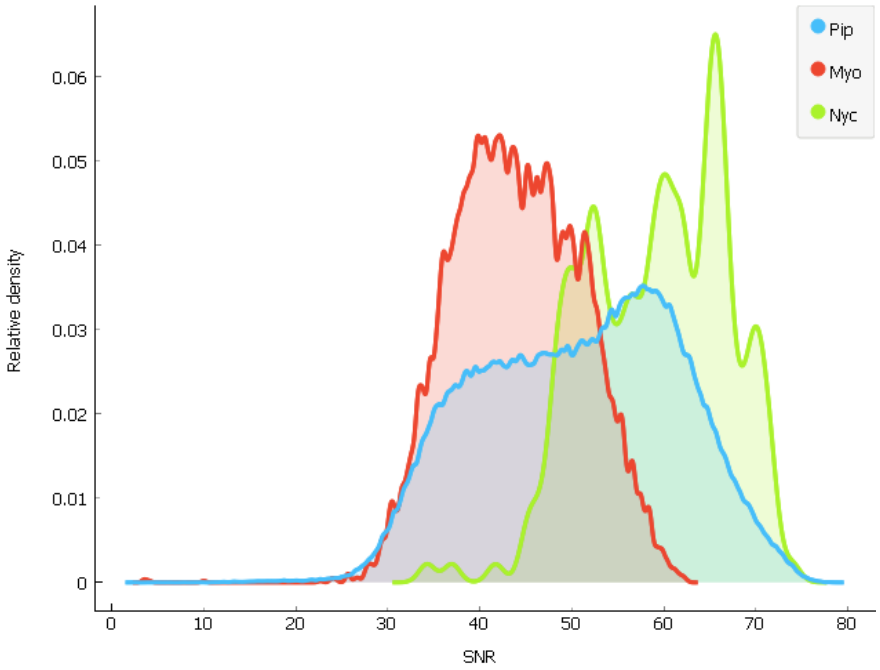


FIGURE 2.13 – SNR selon les groupes acoustiques.

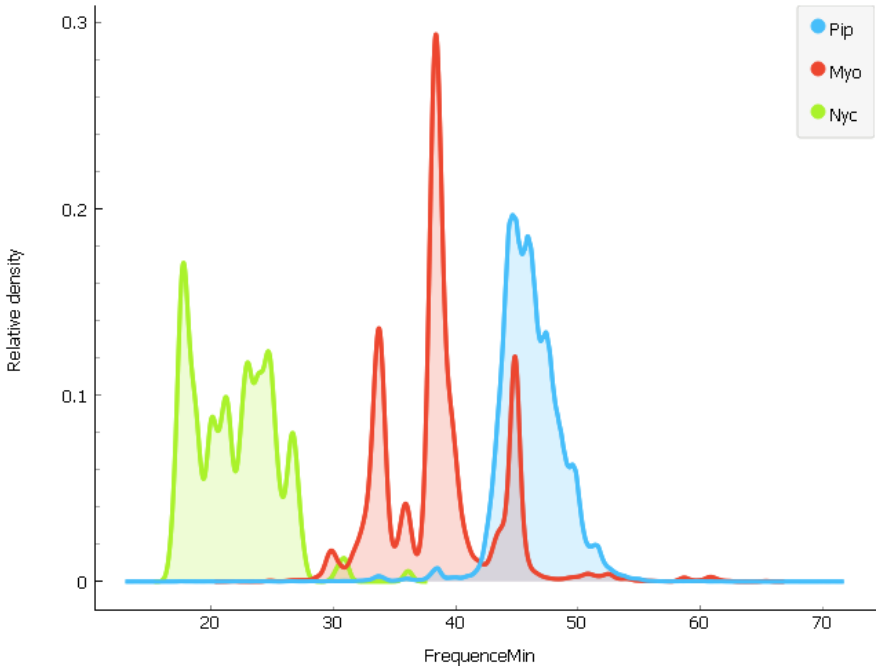


FIGURE 2.14 – Min. de fréquence selon les groupes acoustiques.

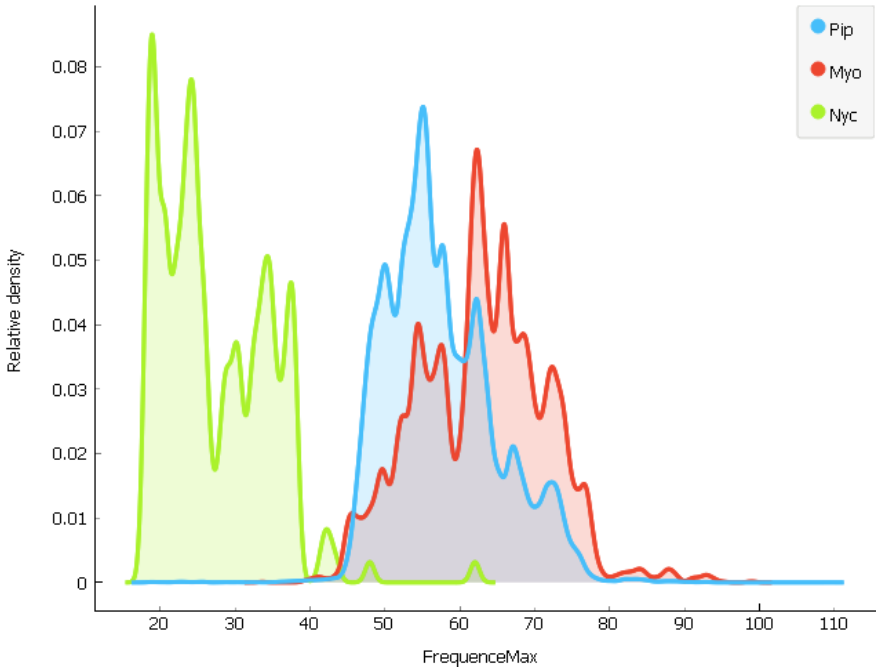


FIGURE 2.15 – Max. de fréquence selon les groupes acoustiques.

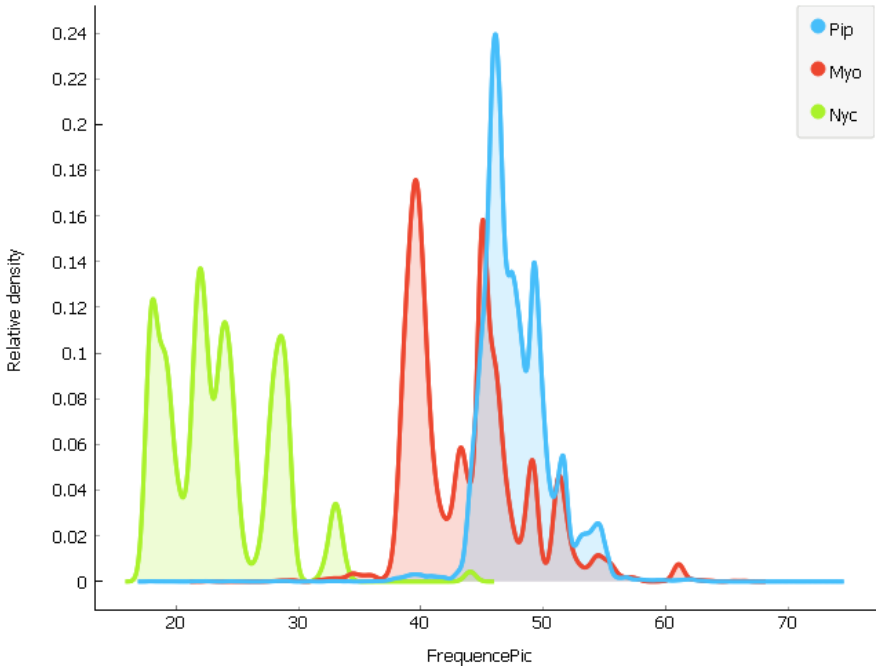


FIGURE 2.16 – Pic de fréquence selon les groupes acoustiques.

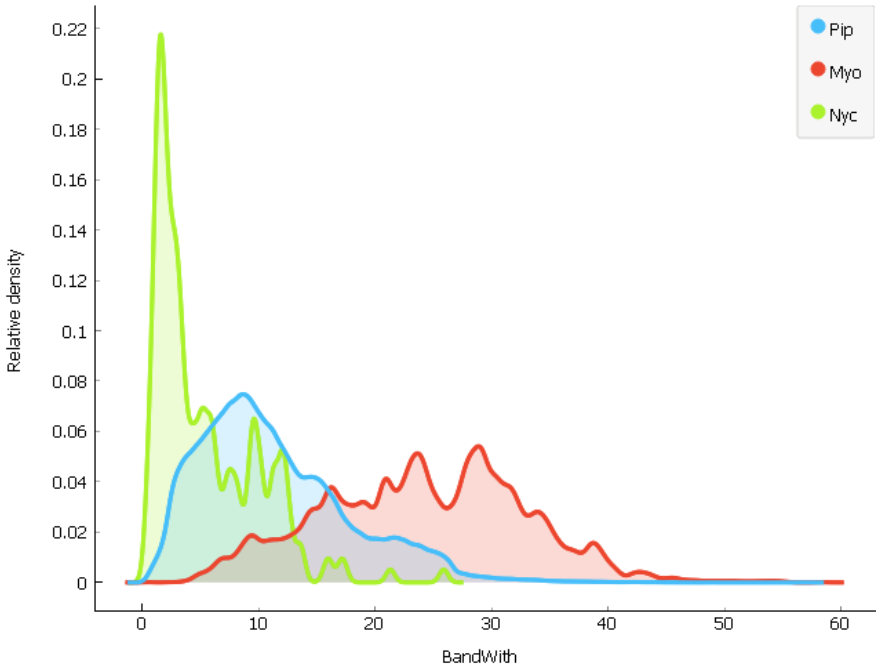


FIGURE 2.17 – Amplitude selon les groupes acoustiques.

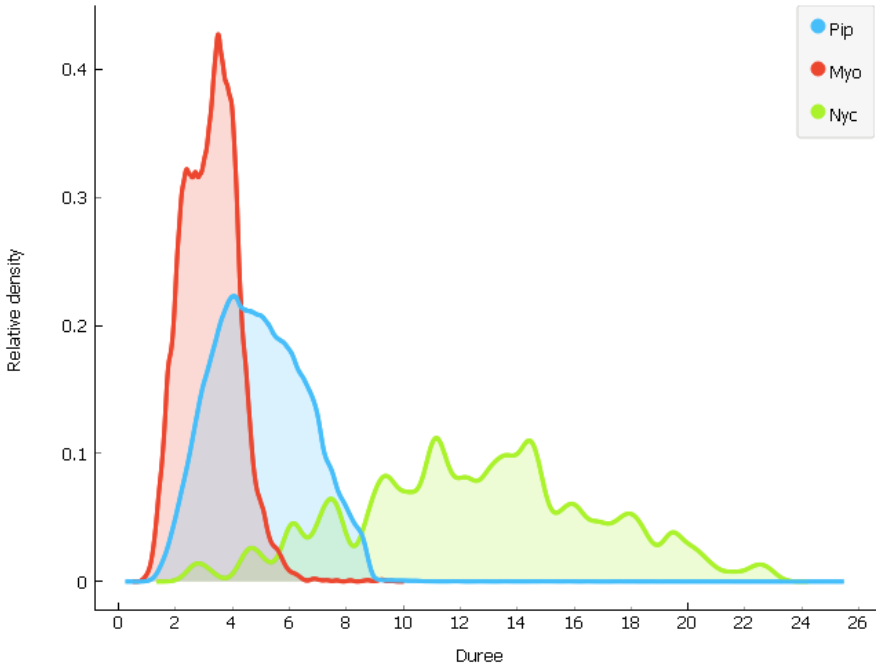


FIGURE 2.18 – Durée du cri selon les groupes acoustiques.

Chapitre 3

Mise en pratique : création du modèle BATMAN

Dans ce chapitre nous allons développer le processus de création d'un modèle de classification des chauves-souris dans le canton de Fribourg. Nous l'avons nommé *BATMAN*. La première section sur les raisons et les objectifs de création du modèle a comme référence principale l'article scientifique [OB17] des créateurs du logiciel *BatScope 3*. Pour la section comportant le réglage des paramètres du modèle, nous nous sommes principalement aidé de l'article de Yann Lecun et Leon Bottou sur la rétropropagation [LBBOM98]. Le code que nous avons développé sur python pour créer le modèle se trouve dans l'annexe A.

Cette mise en pratique nous permet donc de faire la synthèse de trois domaines scientifiques différents que nous souhaitions aborder : le domaine des mathématiques, développé dans le premier chapitre, le domaine biologique du chapitre 2, puis finalement le domaine de l'informatique.

3.1 Raisons de création du modèle et objectif

Comme vu dans le chapitre 2, il existe déjà aujourd'hui le programme *Batscope 3* qui permet de classifier les chauves-souris d'Europe centrale. Celui-ci est actuellement utilisé par les biologistes du domaine, tels que ceux de *FRIbat-CCO*, pour réaliser des analyses sur une population de chauves-souris. Il utilise 6 modèles de machine learning différents afin de classifier les chauves-souris par espèces et fait travailler

ces modèles en coopération [1] pour obtenir le meilleur résultat possible. Un point important qui est dégagé par les créateurs du logiciel est que pour optimiser celui-ci, il serait intéressant d'entraîner à nouveau les différents modèles mais sur une base de données locale. La raison derrière ceci est la variabilité des individus selon leur région à l'intérieur d'une même espèce : les chauves-souris du nord de la France auront donc un cri légèrement différent que celles du canton de Fribourg, même si elles appartiennent à la même espèce, voir au même groupe acoustique. A partir de ce contexte, nous pouvons donc nous fixer deux raisons quant à la création de ce modèle :

— Alléger les calculs en passant d'un logiciel qui se base sur 6 modèles de machine learning différents, à un modèle unique de réseau de neurones.

— Régionaliser le logiciel *Batscope* à la région du Lac de Pérolles pour de futures analyses de *FRIbat-CCO*.

Notre objectif est alors le suivant :

— Créer un classificateur de type MLP qui estime au mieux l'appartenance d'un cri venant du secteur du Lac de Pérolles à l'un des trois groupes acoustiques principaux présents sur celui-ci. Nous rappelons que ces trois groupes sont donc les murins, les noctuloïdes et les pipistrelloïdes.

3.2 Processus de création

Nous tenons tout d'abord à rappeler ici les caractéristiques de la base de données que nous avons en notre possession : elle contient 405'126 ultrasons, dont 270'000 ont été reconnus comme appartenant à une chauve-souris par un test statistique mis en place par *FRIbat-CCO*. Chaque séquence à été classifiée par les biologistes de *FRIbat-CCO*en utilisant *Batscope 3* grâce à la méthode suivante : chaque cri est classé par *Batscope 3* par espèce. Le logiciel estime ensuite l'espèce la plus probable pour la séquence analysée. Si le logiciel estime que sa prédiction n'est pas fiable, elle est vérifiée par les biologistes qui corrigent celle-ci en cas de besoin. Nous avons donc une base de donnée qui a été labelisée d'une manière semi-manuelle.

1. Pour plus de précision sur l'apprentissage en coopération d'algorithme de machine learning, se référer à la section correspondante de [CMB18].

3.2.1 Préparation des données

Dans [LBBOM98], il est montré que les réseaux de neurones sont très influençables par des entrées qui n'ont pas été standardisées, donc mises à la même échelle [2]. La mise à l'échelle est réalisée grâce au module de *preprocessing* de scikitlearn qui effectue celle-ci de la manière suivante : plus une valeur est grande, plus elle sera proche de 1, plus elle est petite, plus elle sera proche de -1. La deuxième étape importante de la préparation des données est que les cris que nous avons reçus sont classés par espèce et non par groupe acoustique. Certains cris n'appartenant pas aux trois groupes acoustiques que nous voulons classifiés, nous avons créé une autre classe pour les englober, les *chiroptera*, afin de permettre aussi à l'algorithme de reconnaître qu'un cri n'appartient à *aucun* des trois groupes. La dernière étape est la séparation des données en deux groupes : le groupe d'apprentissage et le groupe de test. Nous avons effectué cette séparation pour évaluer la précision du modèle sur un ensemble dont il n'a jamais vu les données. L'ensemble test est de 50'000 cris. Pour la préparation des données nous avons donc réalisé ces trois étapes :

— Standardiser les données dans un intervalle arbitraire de $[-1, 1]$.

— Classer les espèces par groupe acoustique, selon les critères abordés au chapitre 2.

— Séparer les données en deux : un groupe d'apprentissage et un groupe de test.

3.2.2 Réglage du taux d'apprentissage

Pour le réglage du paramètre du taux d'apprentissage de notre modèle, nous nous sommes appuyés sur certaines lois heuristiques proposées par [CMB18] et [LBBOM98]. Nous en avons donc déduit qu'une première architecture convenable pour le nombres de neurones et de couches cachées du réseau est 100-100-100, c'est-à-dire de 3 couches cachées, avec 100 neurones chacune. À partir de cette architecture, nous avons effectué des tests sur différents taux d'apprentissage afin de voir dans quel intervalle il nous faut rechercher celui-ci pour les autres architectures que nous allons développer. Celles-ci étant proche de 100-100-100, le taux d'apprentissage optimal n'en sera pas affecté énormément. Nous pouvons donc nous permettre d'utiliser cette architecture de base comme une bonne estimation pour la suite. Les résultats de ces différents tests du réglage du taux d'apprentissage sont présentés dans la figure 3.1. Chaque

2. La raison est que si l'une des variables de l'entrée a des valeurs beaucoup plus grandes que le reste, cette variable aurait une importance intrinsèque non gérée par les poids du réseau.

courbe sur cette figure représente l'évolution de la valeur de la fonction d'erreur selon le nombre d'itérations à travers l'ensemble des données pour différents taux d'apprentissage. Nous pouvons en conclure que l'intervalle d'un taux d'apprentissage pour des calculs efficaces se situe entre 10^{-1} et 10^{-3}.

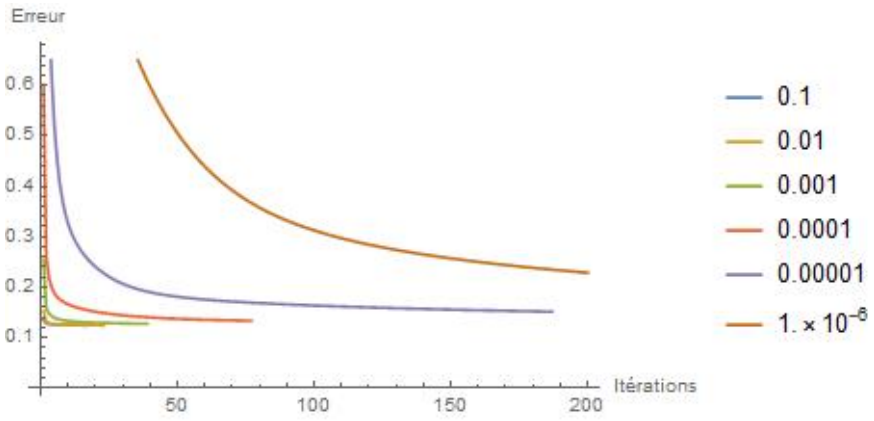


FIGURE 3.1 – Variation de la valeur de la fonction d'erreur selon différents taux d'apprentissage pour une architecture 100-100-100,

3.2.3 Choix de l'architecture

Ici aussi, nous avons dû nous satisfaire de quelques lois heuristiques proposées dans [CMB18] et [LBBOM98]. Pour faire le choix entre les différents modèles, nous avons créé une matrice de confusion pour chaque modèle et retenu celle qui avait le meilleur taux de classification par groupe. Un exemple de mauvais modèle de classification se trouve sur la figure 3.2. Nous pouvons remarquer que 93.94% des noctules se sont retrouvées classifiées comme des murins, les deux cris étant sans doute trop difficiles à distinguer pour un réseau d'une complexité réduite à une architecture de type 5-4. Une comparaison des courbes d'erreurs des deux modèles que nous considérons les plus efficaces se trouve dans la figure 3.3. Ceux-ci atteignent une erreur très faible en très peu d'itérations. Nous avons choisi le modèle le plus convainquant en se basant sur deux critères :

— Peu d'itérations d'apprentissage

— Une classification par groupe acoustique suffisante

Le modèle le plus convainquant selon ces différents critères a donc été celui possédant une architecture de type 50-100-50, avec un taux d'apprentissage de 0.02. Ce modèle est représenté dans la figure 3.4

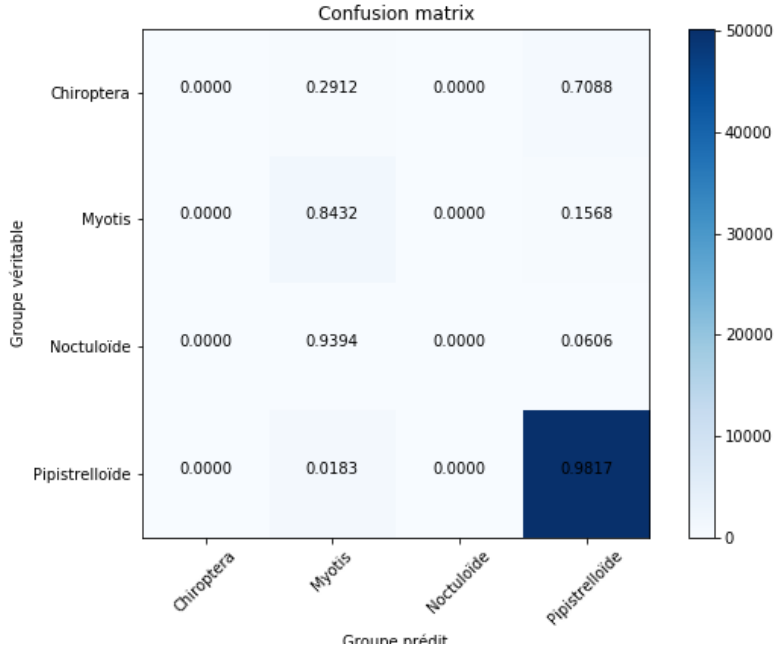


FIGURE 3.2 – Matrice de confusion, alpha = 0.001, architecture de type 5-4

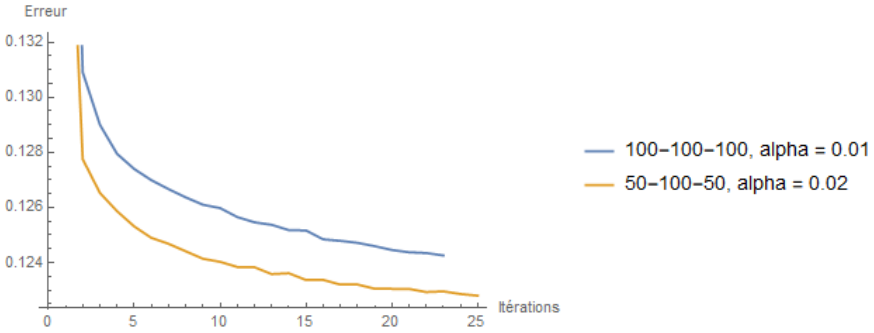


FIGURE 3.3 – Comparaison des courbes d'erreur selon deux modèles différents

3.3 Résultats

La figure 3.5 représente la matrice de confusion du modèle avec une architecture 50-100-50. Nous l'avons nommé *BATMAN*. La précision[3] globale du modèle est de 95.9% sur l'ensemble test de 50'000 cris. Par groupe acoustique, nous obtenons une précision spécifique de 98.38% pour les pipistrelloïdes, 87.5% pour les noctuloïdes et 85.78% pour les murins. L'ensemble des cris reconnus comme appartenant à des chauves-souris par *FRIbat-CCO* mais ne faisant pas parti d'une de ces trois classes

3. Dans le passage suivant, la précision fait référence au taux de classification correcte.

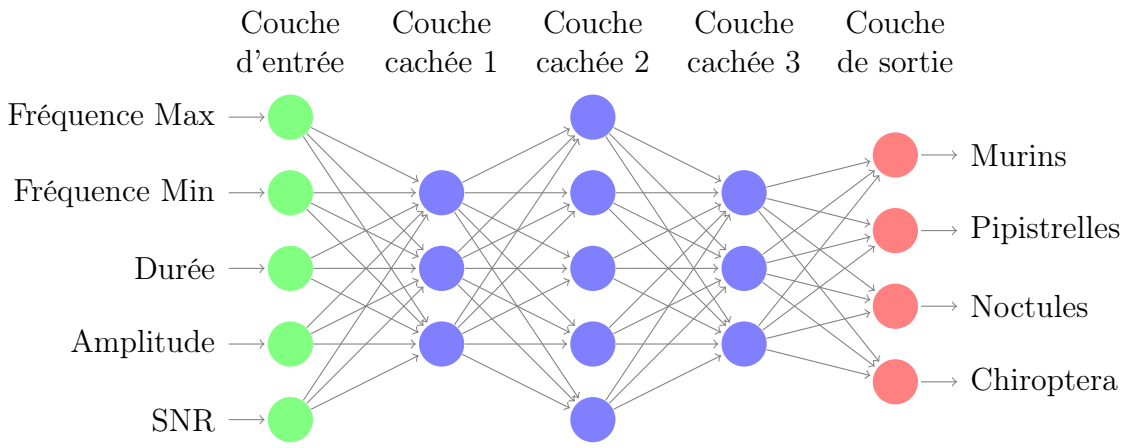


FIGURE 3.4 – Graphe représentatif du MLP *BATMAN* avec 5 entrées, 3 couches cachée et 4 sorties.

(qui auraient donc dû être mis dans la classe *chiroptera*) ont été classés incorrectement.

3.4 Discussion et améliorations possibles

Dans les résultats, nous observons que la classe des pipistrelloïdes a un bien meilleur taux de classification que les deux autres classes. Nous en déduisons que le modèle *BATMAN* est *surentraîné*[4] sur celle-ci. Nous estimons que la cause de ce surentraînement est la suivante : les pipistrelloïdes représentent plus de 90% de la base d'apprentissage. La raison d'une telle répartition des groupes dans la base de données est que celle-ci vient d'un environnement naturel, et le nombre de pipistrelloïdes est effectivement beaucoup plus élevé comparé aux autres groupes acoustiques dans la région du Lac de Pérolles. Le modèle *BATMAN* étant basé sur un réseau de neurones de type MLP, son objectif d'apprentissage est de minimiser l'erreur pour l'*ensemble* des données d'apprentissage, comme vu dans la section 1.4.3. Celui-ci va donc être optimisé en prenant en compte la probabilité qu'a chaque cri de se faire classifier dans les différents groupes acoustiques, celle des pipistrelloïdes étant beaucoup plus élevée que les autres. Cette approche est effectivement efficace dans le cas où l'application que les biologistes auraient du modèle serait d'analyser la population *globale* de chauves-souris dans une région. Alors qu'elle serait mauvaise dans le cas où l'application du modèle serait une analyse *spécifique* d'un cri unique, c'est-à-dire un modèle où l'on voudrait que l'erreur spécifique de classification de chaque classe

4. Se référer à la section 1.3.3 pour une explication intuitive du surentraînement

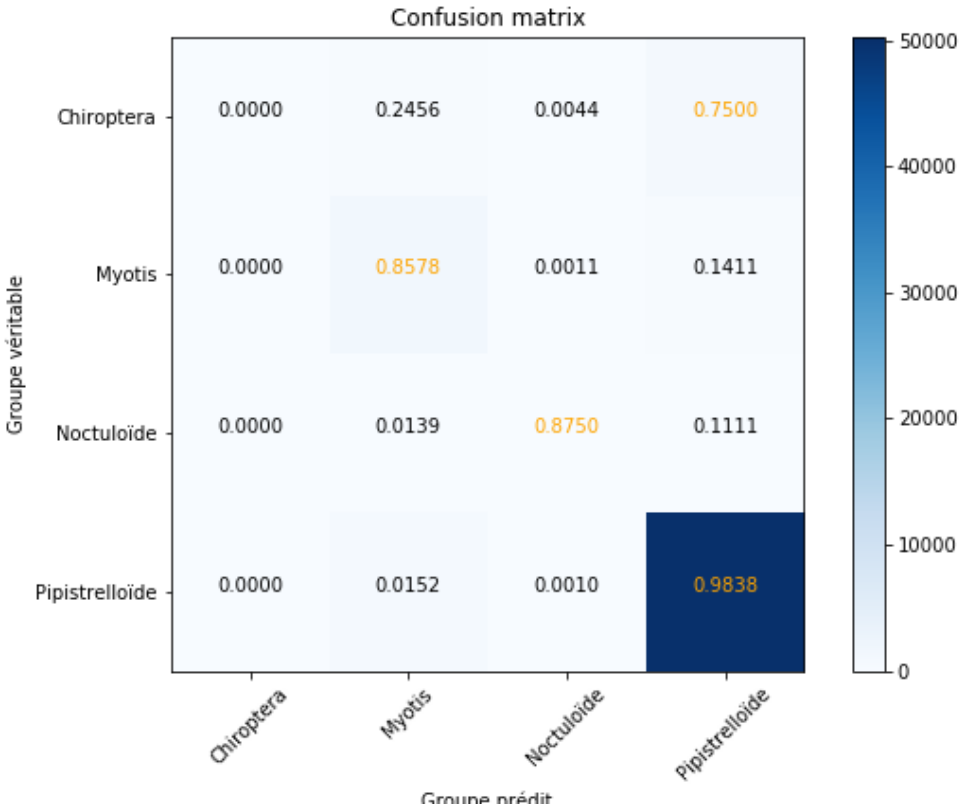


FIGURE 3.5 – Matrice de confusion du modèle BATMAN

soit à peu près égale. Dans ce cas-là, il faudrait que le ratio qu'a chaque groupe dans l'ensemble d'apprentissage soit égal.

Un autre point intéressant à aborder est que le modèle réalise son apprentissage cri par cri, alors que l'optimal serait qu'il puisse analyser une séquence en entier. Pour cela, nous remarquons bien qu'un simple réseau de neurones de type MLP ne suffit pas, il faudrait se diriger vers des réseaux qui possèdent une sorte de *mémoire interne*[5] afin de pouvoir prendre en compte tous les différents cris d'une même séquence avec leur intervalle.

Le dernier point que nous souhaitons aborder est celui des choix des paramètres du modèle. Pour l'instant, ceux-ci sont uniquement basés sur certaines règles heuristiques (donc non purement théoriques) proposées dans différents articles scientifiques. Il existe aujourd'hui des techniques d'optimisation de la descente de gradient telles que la méthode utilisée par l'algorithme *Adam*[6] qui permet de moins dépendre des paramètres arbitraires comme le taux d'apprentissage.

5. Se référer à [13] pour plus d'informations à ce sujet
6. Se référer à [RUD16] pour plus d'informations à ce sujet.

Conclusion

Le meilleur modèle de classification que nous avons pu réalisé grâce à la technique des réseaux de neurones développée au chapitre 1 et à la compréhension des données par la biologie acoustique que nous avons abordée dans le chapitre 2, a une précision empirique globale de 95.9% sur un ensemble de 50'000 exemples de test. Le modèle que nous avons utilisé a les caractéristiques suivantes : c'est un perceptron multicouche, avec 3 couches cachées qui ont respectivement 50, 100 et 50 neurones chacune. Le taux d'apprentissage du modèle a été fixé à 0.02. Nous avons obtenu une précision spécifique pour chaque groupe acoustique présent sur le Lac de Pérolles de 85.78% pour les murins, de 87.50% pour les noctuloïdes et de 98.38% pour les pipistrelloïdes.

Glossaire

MLP. – Multilayer Perceptron : perceptron multicouche

kHz. – kilohertz : unité de fréquence (10^3 Hertz)

dB. – Décibel : unité de puissance sonore

SNR. – Signal-to-noise ratio : rapport signal/bruit

WSL. – Wald, Schnee und Landschaft : l'institut fédéral suisse de recherches sur la forêt, la neige et le paysage

GPS. – Global Positioning System : système de localisation par satellite

Pip. – Espèce de chauve-souris : la pipistrelle (du lat. *Pipistrellus*)

Myo. – Espèce de chauve-souris : le murin (du lat. *Myotis*)

Nyc. – Espèce de chauve-souris : la noctule (du lat. *Nyctalus*)

Remerciements

Nous voulons remercier dans un premier temps M. Laurent Karth-Robadey, notre professeur de séminaire, pour sa grande disponibilité et ses précieux conseils quant à la rédaction de ce travail de maturité. Nous tenons également à adresser un tout grand merci à toute l'équipe de *FRIbat-CCO* et particulièrement M. Jérôme Gremaud pour avoir assuré la partie applicative de ce travail de maturité. C'est notamment grâce à leur collaboration que ce projet s'est dirigé vers le sujet abordé. Nous remercions finalement toutes les personnes qui ont eu la patience et la minutie de relire et de corriger notre travail.

Le logiciel qui a permis de rédiger ce texte est libre. Il s'agit de LaTeX. Nous tenons donc à remercier ici l'ensemble des développeurs libres qui partagent non seulement leur savoir, mais aussi leur travail.

Bibliographie

[CMB18] Antoine CORNUÉJOLS, Laurent MICLET, and Vincent BARRA. *Apprentissage artificiel*. Eyrolles, 2018. 9, 18, 29, 34, 37

[DCE+13] Janez DEMŠAR, Tomaž CURK, Aleš ERJAVEC, Črt GORUP, Tomaž HOČEVAR, Mitar MILUTINOVIČ, Martin MOŽINA, Matija POLAJNAR, Marko TOPLAK, Anže STARIČ, Miha ŠTAJDOHAR, Lan UMEK, Lan ŽAGAR, Jure ŽBONTAR, Marinka ŽITNIK, and Blaž ZUPAN. Orange : Data mining toolbox in python. *Journal of Machine Learning Research*, 14 :2349–2353, 2013.

[Den15] François Denis. Tp3 perceptron : Master informatique année 1. 2015. 19

[GMR16] Jérome GREMAUD, Benoît MAGNIN, and Emmanuel REY. *LUMIERE ! Les chauves-souris du canton de Fribourg*. Fribat-CCO Fribourg et musée d'histoire naturelle de Fribourg, 2016. Ouvrage sur les chauves-souris, explique l'écologie acoustique.

[LBBOM98] Yann LECUN, Leon BOTTOU, Genevieve B. ORR, and Klaus-Robert MUELLER. Efficient backprop. 1998. 9, 39

[NIE15] Michael A. NIELSEN. *Neural Networks and Deep Learning*. Determination Press, 2015. 9

[OB17] Martin OBRIST and Ruedi BOESCH. Batscope manages acoustic recordings, analyses calls and classifies bat species automatically. 2017. Article scientifique sur le développement de Batscope.

[PVG+11] F. PEDREGOSA, G. VAROQUAUX, A. GRAMFORT, V. MICHEL, B. THIRION, O. GRISEL, M. BLONDEL, P. PRETTENHOFER, R. WEISS, V. DUBOURG, J. VANDERPLAS, A. PASSOS, D. COURNAPEAU, M. BRUCHER, M. PERROT, and E. DUCHESNAY.

Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12 :2825–2830, 2011.

[RAS16] Tariq RASHID. *Make Your Own Neural Network*. CreateSpace, 2016. Ouvrage anglais introduisant la notion de réseau de neurones pour les débutants, accessible à tous les personnes intéressées par les mathématique et la programmation. 9

[RUD16] Sebastian RUDER. An overview of gradient descent optimization algorithms. *CoRR*, 2016.

[RUS12] Jon RUSS. *British Bat Calls : A Guide to Species Identification*. Pelagic Publishing, 2012. Ouvrage sur les chauves-souris, source pour illustrer les sonogrammes des groupes acoustiques.

Webographie

[1] Bandwidth of Signals. https://www.toppr.com/guides/physics/communication-systems/bandwidth-of-signals/ (dernière consultation le 16 janvier 2019).

[2] Batlogger M datalogger for recodring ultrasound vocalizations of bats. https://www.batlogger.com/en/products/batlogger_m/ (dernière consultation le 04 mars 2019).

[3] BatScope 3. https://www.wsl.ch/de/services-und-produkte/software-websites-und-apps/batscope-3.html (dernière consultation le 05 mars 2019).

[4] Jean-Yves BAUDOT. Les distances. http://www.jybaudot.fr/Analdonnees/distances.html (dernière consultation le 25 mars 2019).

[5] Confusion Matrix. https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py (dernière consultation le 08 mars 2019).

[6] Michael COLLINS. Convergence Proof for the Perceptron Algorithm. http://www.cs.columbia.edu/~mcollins/courses/6998-2012/notes/perc.converge.pdf (dernière consultation le 24 mars 2019). 19

[7] D500X Ultrasound Detector/Recorder. http://www.batsound.com/?p=10 (dernière consultation le 04 mars 2019).

[8] Jean DEBORD. Calcul matriciel. http://uel.unisciel.fr/physique/outils_nancy/outils_nancy_ch11/co/apprendre_ch11_12.html (dernière consultation le 31 mars 2019).

[9] Définition de la fréquence sonore. https://www.greenfacts.org/fr/glossaire/def/frequencesonore.htm (dernière consultation le 12 janvier 2019).

[10] Elizabeth GOODSPEED. A diagram showing a perceptron updating its linear boundary as more training examples are added. https://commons.wikimedia.org/wiki/File:Perceptron_example.svg (dernière consultation le 24 mars 2019), key = PercEx. 11

[11] Représentation du gradient dans un espace en 3 dimensions. https://bit.ly/2HE8rPW (dernière consultation le 24 mars 2019).

[12] Tom HARRIS. Bats and Echolocation. https://animals.howstuffworks.com/mammals/bat2.htm (dernière consultation le 18 janvier 2019).

[13] Andrej KARPATHY. The Unreasonable Effectiveness of Recurrent Neural Networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/ (dernière consultation le 24 mars 2019).

[14] Réserve natuelle du lac de Pérolles. https://www.fr.ch/en/node/102266 (dernière consultation le 06 mars 2019).

[15] Frequency. https://dosits.org/science/sound/characterize-sounds/frequency/ (dernière consultation le 12 mars 2019).

[16] Etienne NOISEAU Baptiste LANASPEZE Matthieu CROCQ, Yannick DAUBY. Glossaire d'écologie sonore. https://www.wildproject.org/journal/4-glossaire-ecologie-sonore (dernière consultation le 21 décembre 2018).

[17] Bastien MAURICE. Représentation neurone biologique. https://deeplylearning.fr/cours-theoriques-deep-learning/fonctionnement-du-neurone-artificiel/ (dernière consultation le 24 mars 2019). 10

[18] Margaret ROUSE. What is a sound wave ? https://whatis.techtarget.com/definition/sound-wave (dernière consultation le 14 mars 2019).

[19] Grant SANDERSON. 3Blue1Brown - Neural Networks. https://www.3blue1brown.com/videos/2017/10/9/neural-network (dernière consultation le 24 mars 2019). 9

[20] Neural Network(supervised). https://scikit-learn.org/stable/modules/neural_networks_supervised.html (dernière consultation le 08 mars 2019).

[21] Signal-to-noise ratio. https://en.wikipedia.org/wiki/Signal-to-noise_ratio (dernière consultation le 16 janvier 2019).

[22] The Nature of Sound. https://physics.info/sound/ (dernière consultation le 04 mars 2019).

[23] ASU Arizona State University. What is Echolocation ? https://askabiologist.asu.edu/echolocation (dernière consultation le 27 décembre 2018).

[24] Unsere Mission : Forschung für Mensch und Umwelt. https://www.wsl.ch/de/ueber-die-wsl.html (dernière consultation le 05 mars 2019).

Table des figures

Annexe A

Code pour le modèle *BATMAN*

Vous trouverez dans cette annexe le code pour le modèle de classification des chauves-souris du Lac de Pérolles que nous avons créé. Celui-ci a été réalisé au moyen du module de machine learning scikit-learn sous Python 3, avec l'aide de l'interface Jupyter Notebook. Nous avons aussi eu recours à l'utilisation du module Panda pour gérer le fichier de la base donnée. Nous tenons à préciser que nous n'avons pas entièrement rédiger la fonction *plot confusion matrix*, mais que celle-ci est basée sur la fonction qui est développé dans l'article [5].

Batman

March 14, 2019

In [3]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import itertools

def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True,
                          alpha=0.001,layers=100):
    """
    plot_confusion_matrix(
        ConfusionMatrix cm,
        list target_names,
        str title, str cmap,
        bool normalize, int alpha, int layers)
    --> None.
    Fonction permettant de dessiner la matrice de confusion du module scikitlearn.
    """
    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
```

```
            plt.yticks(tick_marks, target_names)

        if normalize:
            cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


        thresh = cm.max() / 1.5 if normalize else cm.max() / 2
        for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
            if normalize:
                plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                         horizontalalignment="center",
                         color="black")
            else:
                plt.text(j, i, "{:,}".format(cm[i, j]),
                         horizontalalignment="center",
                         color="black")


        plt.tight_layout()
        plt.ylabel('Groupe véritable')
        plt.xlabel('Groupe prédit\nPrécision={:0.4f}; Erreur={:0.4f}'.format(
            accuracy, misclass))
        plt.savefig("batman"+str(alpha)+"_"+str(layers)+".png")
        plt.show()
```

```
In [10]: bats_data = pd.read_excel("bats.xlsx") #Loader les données
         bd = bats_data
```

```
In [6]: #Choix des colonnes: FrequenceMax, FrequenceMin,Duree,FrequencePic,Bandwith
        first = bd.ix[:,2:7]
        #Choix des colonnes: SNR, AnyClassification, ManualClassificationSP1
        second = bd.ix[:,9:12]
        #Concaténation des colonnes sélectionnées
        data_frame = pd.concat([first,second], axis=1)
        #Trie selon le test statistique
        data_frame = data_frame.loc[lambda d: d.AnyClassification == "Pass",:]
        data_frame = data_frame.drop("AnyClassification", axis =1)
        #Enlève les élements qui ne sont pas des chauves-souris
        data_frame = data_frame.loc[lambda d: d.ManualClassificationSP1 != "None",:]
        data = data_frame.values
        #Les 4 classes que nous souhaitons classées
        target_class = ["Chiroptera","Myotis","Noctuloïde","Pipistrelloïde"]
```

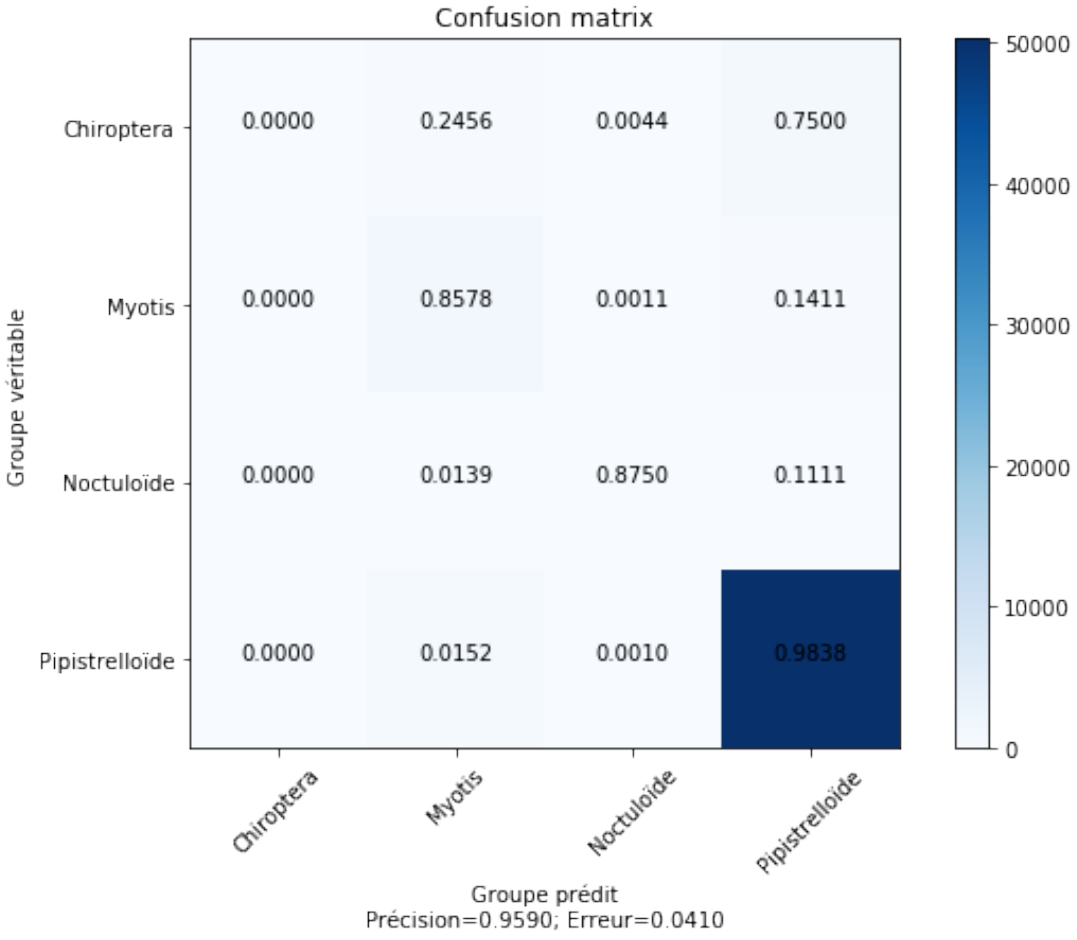
```
In [7]: X = [] #Valeur des entrées
        e_y = [] #Valeurs désirées par espèce
        for bat in data:
            e_y.append(bat[6])
            X.append(bat[0:6])
```

In [8]:
```
y = [] #Valeur désirée par groupe accoustique
for bat in e_y:
    if bat[:3] == "Ept":
        y.append("Nyc")
    else:
        y.append(bat[:3])
```

In [9]:
```
#Split des données en set d'exemples d'entrainement et de test
#Mélange aléatoire des données avec shuffle = True
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2,
                                                    shuffle = True)
#Modèle scikit-learn pour standardiser les données
scaler = StandardScaler()
scaler.fit(X_train)
#Standardise les données d'entrainement
X_train = scaler.transform(X_train)
#Standardise les données de test
X_test = scaler.transform(X_test)
```

In [20]:
```
#Création du modèle de classification "BATMAN" avec le module scikit-learn.
#SGD = descente de gradient stochastique
batman = MLPClassifier(solver ="sgd", learning_rate_init = 0.02,
                       hidden_layer_sizes=(50,100,50),max_iter=500)
#Apprentissage sur le set d'exemples d'entrainement (X_train, y_train)
batman.fit(X_train,y_train)
#Nombre d'itérations avant convergence de l'algorithme
print(batman.n_iter_)
#Prédiction sur le set d'exemples de test
y_pred = batman.predict(X_test)
#Création de la matrice de confusion
cm = confusion_matrix(y_test,y_pred)
plot_confusion_matrix(cm, target_class, normalize = True,
                      alpha="0.02", layers = "50-100-50")
```

25


Confusion matrix
Chiroptera
Myotis
Noctuloïde
Pipistrelloïde
Groupe véritable
0.0000
0.2456
0.0044
0.7500
0.0000
0.8578
0.0011
0.1411
0.0000
0.0139
0.8750
0.1111
0.0000
0.0152
0.0010
0.9838
Chiroptera
Myotis
Noctuloïde
Pipistrelloïde
Groupe prédit
Précision=0.9590; Erreur=0.0410
50000
40000
30000
20000
10000
0

Résumé

Dans le premier chapitre, nous avons présenté comment fonctionne un réseau de neurones. Nous nous sommes principalement attardés sur le perceptron et le perceptron multicouche.

Dans le deuxième chapitre, nous avons développé les différentes caractéristiques de l'écologie acoustique, puis expliqué la procédure de récolte des données acoustiques des chauves-souris autour du Lac de Pérolles dans le canton de Fribourg.

Dans le troisième chapitre, nous avons mis en pratique ce qui a été développé dans les deux chapitres précédents, en créant un modèle de classification des chauves-souris du Lac de Pérolles à l'aide d'un perceptron multicouche. Nous avons notamment comparé différents paramètres pour le modèle ainsi que présenté les résultats de précision du meilleur modèle que nous avions.

Mots clés : Chauve-souris, Classification, Perceptron multicouche, Biologie acoustique, Réseaux de neurones

Déclaration personnelle

Thibault Liaudat
Chemin du Grand Pré 18
1618 Châtel-St-Denis

Samuel Russo
Route de Rombuet 8
1616 Attalens

(1) Nous certifions que le travail

Création d'un modèle de classification des chauves-souris du canton de Fribourg via un Perceptron multicouche

Les liens cachés entre les mathématiques, la biologie et l'informatique

été réalisé par nous conformément au « Guide de travail » des collèges et aux « Lignes directrices » de la DICS concernant la réalisation du Travail de Maturité.

(2) Nous prenons connaissance que notre travail sera soumis à une vérification de la mention correcte et complète de ses sources, au moyen d'un logiciel de détection de plagiat. Pour assurer notre protection, ce logiciel sera également utilisé pour comparer notre travail avec des travaux écrits remis ultérieurement, afin d'éviter des copies et de protéger notre droit d'auteur. En cas de soupçon d'atteintes à notre droit d'auteur, nous donnons notre accord à la direction de l'école pour l'utilisation de notre travail comme moyen de preuve.

(3) Nous nous engageons à ne pas rendre public notre travail avant l'évaluation finale

(4) Nous nous engageons à respecter la Procédure d'archivage des travaux de maturité en vigueur dans notre école.

(5) Nous autorisons la consultation de notre travail par des tierces personnes à des fins pédagogiques et/ou d'information interne à l'école

Lieu, date : Signatures :