

Laboratoire 1

Perceptron multicouche (MLP)

Durée : 1 séance

Par Chakib Tadj et Christophe Lamarche

mise à jour: 23 mai 2021

1 Description

Ce laboratoire a comme but de s'initier avec le langage informatique **Python**, l'environnement de programmation **Colaboratory** et les bibliothèques utilisées dans la conception de réseaux neuronaux.

Les réseaux de neurones conçus durant ce laboratoire devront extraire les paramètres les plus pertinents d'un signal de parole et réaliser la reconnaissance de la parole. Ce laboratoire consiste en la conception d'un réseau de neurones à rétropropagation du gradient de l'erreur (*Descente du gradient stochastique* [SGD]).

1.1 Exposé du problème

La difficulté du processus de traitement du signal de la parole réside dans la nature même de ce signal qui est pseudo-aléatoire. De ce fait, l'extraction des paramètres temporels de l'information contenus dans celui-ci devient problématique. Une façon de solutionner le problème est d'en extraire les paramètres les plus pertinents dans le domaine des fréquences. La FFT (Fast Fourier Transform / Transformée de Fourier Rapide) est un outil qui est largement utilisé dans ce domaine pour représenter le signal temporel en fréquence. En général, on filtre le signal sur une série de bancs de filtres selon une échelle logarithmique. Ceci est justifié par le fait que le système auditif humain est logarithmique quant à sa capacité de percevoir les plages de fréquences. Une fois les paramètres les plus pertinents du signal extrait (figure 1), ils sont utilisés pour entraîner le réseau de neurones. La tâche de celui-ci consiste à projeter les données dans un espace de dimension réduite permettant une meilleure discrimination entre les différentes formes du signal de parole.

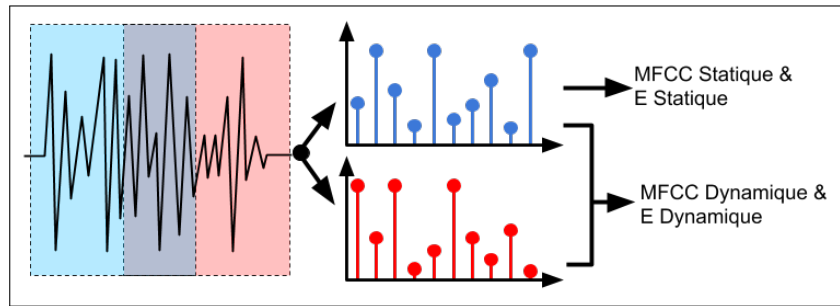


FIGURE 1 – Traitement du signal audio

Les données à l'intérieur des bases de données sont divisées en 60 fenêtres. Chaque fenêtre possède 12 MFCCs (*coefficients Cepstraux à Fréquences Mel*) statiques, une valeur de l'énergie statique, 12 MFCCs dynamiques et une valeur de l'énergie dynamique. Donc, chaque signal sonore possède 1560 coefficients et le chiffre lié au signal.

$$1560 = 60[\text{fenêtres}] \cdot (12[\text{MFCC statique}] + 1[\text{E statique}] + 12[\text{MFCC dynamique}] + 1[\text{E dynamique}])$$

Pour se faire, l'étudiant devra concevoir et analyser les performances de quatre réseaux de neurones afin de classifier les segments audios.

- Perceptron simple
- MLP de 2 couches
- MLP de 2 couches avec couche d'abandon (*Dropout*)
- MLP de 3 couches

2 Théorie

2.1 Python

Python est un langage de programmation orienté objet qui a eu un gain de popularité lors de son appropriation par le secteur de l'analyse de données (*Data Science*). Il est souvent décrit comme un langage de programmation facile à apprendre. L'utilisation de Python s'étend de la conception d'applications web à la création de réseaux neuronaux en passant par la création de jeux video. Python à l'aide de la librairie *Numpy* peut être utilisé afin de faire des calculs numériques similaires avec ce qui peut être fait avec *MATLAB*.

2.2 Colaboratory

Colaboratory est un environnement de programmation de type *Notebook* basé sur le *Project Jupyter*. Cet environnement est offert en ligne gratuitement pour quiconque (utilisation personnelle et utilisation professionnelle). Il intègre plusieurs des librairies d'apprentissage machine et met à la disposition de l'utilisateur des GPUs (*Graphical Processing Unit*) et des TPUs (*Tensor Processing Unit*) pour accélérer l'entraînement des modèles.

2.3 Tensorflow

Tensorflow est une librairie d'apprentissage machine *Open Source* basée sur la librairie **Keras**. Tensorflow offre la possibilité d'embarquer les modèles pour le web (avec tensorflow.js), les applications mobiles (avec Tensorflow Lite) et en production avec (Tensorflow Extended).

2.4 Autres librairies utilisées

- Tensorflow.keras.layer :
 - Module de Tensorflow offrant les différentes couches qui seront intégrées dans le réseau
- Numpy :
 - Librairie mathématique (La plus utilisée de Python)
 - Elle est utilisée pour faciliter la manipulation de tableau
- Pandas :
 - Librairie pour manipuler des bases de données (Dataframes)
 - Elle est utilisée pour importer les bases de données ".csv"
- Matplotlib.pyplot :
 - Librairie d'affichage graphique
 - Elle est utilisée pour la production de graphique et de tableau
- Sklearn :
 - Librairie d'apprentissage machine
 - Elle est utilisée pour la segmentation de la base de données d'entraînement avec KFold
- Scipy.stats :
 - Regroupement de librairies de mathématique, de science et d'ingénierie
 - Elle est utilisée pour calculer le mode des résultats des modèles pour déterminer la valeur à l'intérieur de la base de données mystère.

3 Étapes

3.1 Ouvrir un Notebook

Afin d'ouvrir un Notebook, il est préférable d'utiliser le navigateur Chrome. Toutefois, l'environnement est fonctionnel sur d'autres navigateurs.

Colaboratory est accessible à l'URL : 'https://colab.research.google.com/'. Afin de créer un nouvel environnement, il est nécessaire de se connecter avec un compte Google/Gmail.

Sur le site, vous êtes reçu par la figure 2. Choisissez l'option 'NEW NOTEBOOK'. Cette action démarre un notebook contenant une cellule de code tel qu'à la figure 3. Dans le cas où vous êtes reçu par un notebook nommé "*Welcome to Colaboratory*", assurez-vous que vous êtes bien connecté sur votre compte Google et sous l'onglet Fichier (*File*), sélectionnez l'objet 'Nouveau notebook' (*New Notebook*).

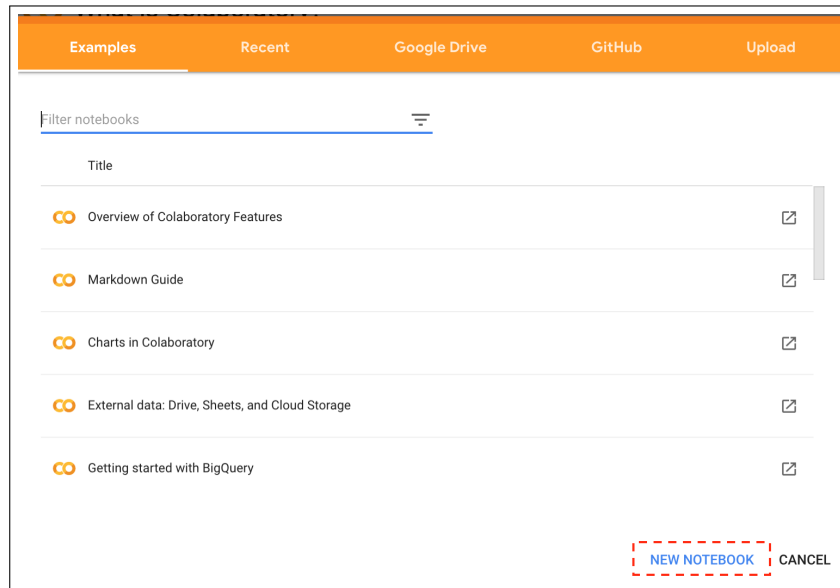


FIGURE 2 – Menu de départ



FIGURE 3 – Notebook de départ

3.2 Personnaliser le Notebook

Avant de commencer, il peut être intéressant de personnaliser le notebook afin d'améliorer l'expérience. Les étapes suivantes ne sont pas obligatoires, mais elles sont conseillées.

Sous l'onglet 'Outils' (*Tools*), cliquer sur l'option 'Paramètres' (*Settings*) (figure 5). Il est possible d'accéder le menu 'Paramètres' avec l'icône d'engrenage en haut à droite du notebook ou en haut à droite des cellules du notebook.



FIGURE 4 – Position des icônes des paramètres

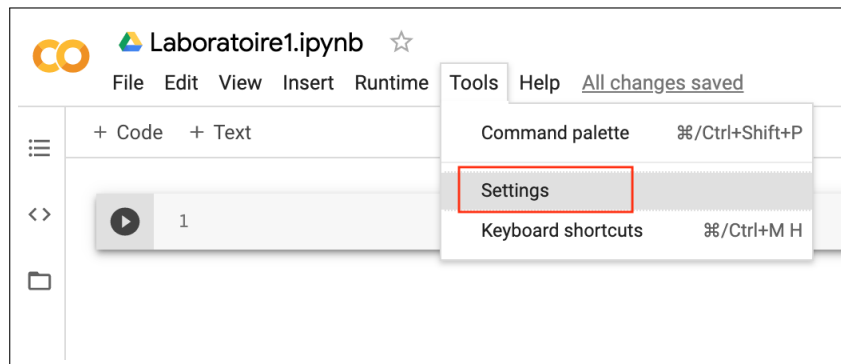


FIGURE 5 – Paramètres du notebook

Dans le menu ‘Paramètres’, vous êtes, par défaut, mis sur la page des paramètres du site (Figure 6). Dans cette page, il est possible de changer le thème afin qu’il devienne foncé et d’autoriser l’application *Colaboratory* d’accéder aux répertoires privés sur un compte *Github*. Cette dernière option peut être utile si vous prévoyez faire la gestion du laboratoire avec **Git**. Par défaut, *Colaboratory* enregistre sur votre *Google Drive* dans un fichier nommé ‘Colab Notebook’.

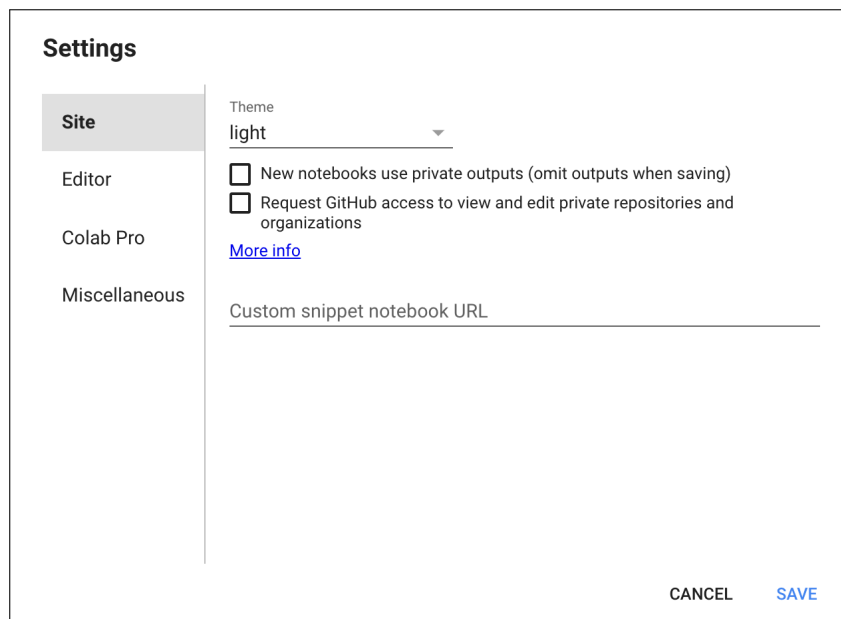


FIGURE 6 – Paramètres du site

Dans l’onglet ‘Éditeur’ (*Editor*), assurez-vous que les options soient identiques à la figure 7. Cela permet de faciliter la rédaction de code à l’intérieur du Notebook.

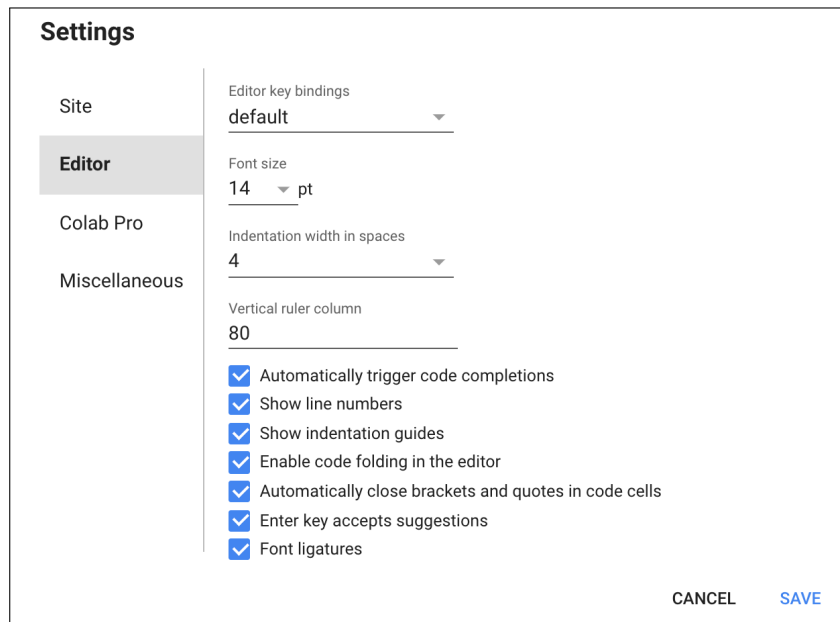


FIGURE 7 – Paramètres de l’éditeur

3.3 Ajouter une cellule de texte

Pour ajouter une cellule dans votre notebook, vous pouvez déplacer votre curseur en haut ou en bas d’une cellule existante pour faire afficher l’option d’ajout (voir figure 8). Lorsqu’une cellule est sélectionnée, les boutons ‘+ Code’ et ‘+ texte’ ajoutent une cellule à la suite de la cellule sélectionnée. L’onglet ‘Insertion’ (*Insert*) offre des fonctionnalités similaires.

Pour déplacer une cellule, il suffit d’utiliser les icônes de flèches en haut à droite des cellules. Il est possible de déplacer plusieurs cellules consécutives en les sélectionnant.



FIGURE 8 – Ajouter une cellule

Les cellules de texte sont utiles pour expliquer les résultats ou les théories d’une cellule de code. Les cellules de textes utilisent la nomenclature ‘Markdown’ pour le texte et ‘LaTeX’ pour les équations (exemple à la figure 9).

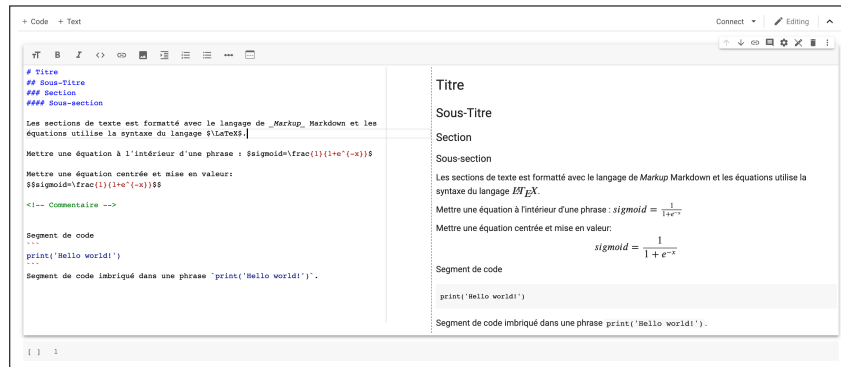


FIGURE 9 – Ajouter une cellule de texte

3.4 Ajouter une cellule de code

Les cellules de code sont le cœur du notebook. Ces cellules utilisent **Python 3**.



FIGURE 10 – Ajouter une cellule de code

Pour exécuter individuellement une cellule de code, il suffit de cliquer sur l'icône en haut à gauche de la cellule (figure 11). Davantage d'options d'exécution sont disponibles sous l'onglet 'Exécution' (*Runtime*) (figure 12). Le résultat de l'exécution est affiché sous la cellule. Les variables et fonctions d'une cellule sont accessibles après exécution pour les cellules de codes suivantes. En tout temps, il est possible d'arrêter le traitement d'une cellule de code en cliquant de nouveau sur l'icône d'exécution.



FIGURE 11 – Exécution d'une cellule de code

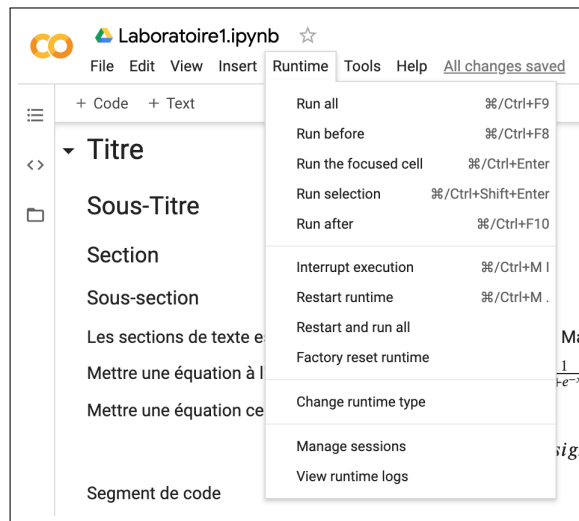


FIGURE 12 – Menu d’exécution de cellules de code

Le résultat à la dernière ligne d’une cellule de code est aussi affiché. Afin que le résultat de la dernière ligne ne soit pas affiché, ajouter ‘;’ à la fin de la ligne.

Note : lorsqu’un segment de code est demandé d’être retranscrit, ouvrez une nouvelle cellule de code afin de faciliter le déverminage (“debug”) en réduisant le temps d’exécution. Les segments de code sont conçus en sorte de respecter la norme PEP8, une description des fonctions est produite avec la structure du segment de codes de ce laboratoire 1 et des commentaires sont ajoutés pour améliorer la compréhension des commandes. Après avoir retranscrit un segment de code, compilez la cellule de code.

```

1 def nom_de_la_fonction(argument1, argument2="valeur par défaut"):
2     """
3     NOM_DE_LA_FONCTION # en majuscule
4
5     breve description de la fonction
6
7     @Arguments :
8         argument1 {type voulu pour l'argument1} : description de ce que doit etre l
9         argument2 {type voulu pour l'argument2} : description de ce que doit etre l
10        # (facultatif) explication du choix pour la valeur par défaut
11
12    @Return :
13        {type de l'objet retourne par la fonction} : description de l'objet
14        retourne par la fonction
15        # Si d'autres objets sont retournees, la meme syntaxe est utilisee pour
16        chaque objet.
17
18    (facultatif) Note : Information qui peut etre importante a connaitre pour l'
19    utilisation de la fonction

```


Segment de codes 1 – Nomenclature de la description des fonctions

3.5 Connecter à la machine virtuelle

Afin de pouvoir compiler les cellules de code, il est nécessaire de se connecter à la machine virtuelle du notebook. Pour se faire, cliquez sur le bouton “Connect” en haut à droite du notebook tel que dans la figure 13.



FIGURE 13 – Connecter à la machine virtuelle du notebook

3.6 Importer les librairies

Afin de produire les réseaux de neurones pour la classification des segments de la voix, il est primordial d’importer les librairies nécessaires. Dans la première cellule de code, copier les lignes du segment de codes 2.

```

1 import tensorflow as tf
2 from tensorflow.keras import layers
3 import numpy as np
4 import pandas as pd
5 import matplotlib.pyplot as plt
6 from scipy import stats
7 from sklearn.model_selection import KFold
8
9 # Pour l’affichage des figures sans plt.show()
10 %matplotlib inline

```

Segment de codes 2 – Importer les librairies

3.7 Importer les bases de données

Les bases de données utilisées dans le laboratoire sont disponibles sur le *Moodle* de la classe. Après le téléchargement de ceux-ci, assurez-vous que le dossier est bien décompressé. Du côté gauche du notebook de Colaboratory (figure 14), il y a trois onglets qui peuvent être utiles pour travailler avec le notebook. La première option est la table des matières du notebook. La table des matières est conçue automatiquement avec les titres dans les cellules de texte (figure 9). La deuxième option est une cellule de recherche de segments de code compatible avec le notebook. La dernière option est un navigateur de dossiers pour les fichiers de la machine virtuelle où est placé le notebook.

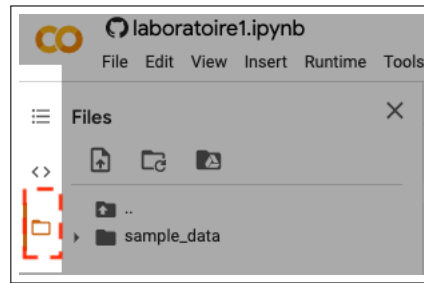


FIGURE 14 – Onglets du côté gauche du notebook

Sur l’onglet du navigateur de dossier, nous allons y déposer les fichiers “training.csv”, “test.csv” et “mystere.csv” présents dans le fichier précédemment décompressé. Pour ce faire, il faut glisser-déposer (*Drag and Drop*) directement dans la section de l’onglet fichier telle que dans la figure 15. Il faut noter que Colaboratory n’accepte pas les dossiers pour le glisser-déposer. Alors, il faut sélectionner directement les trois fichiers à l’intérieur du fichier.



FIGURE 15 – Déposer les fichiers dans la barre de fichiers

Lorsque les fichiers y sont bien déposés, votre onglet ressemblera à celui de la figure 16.

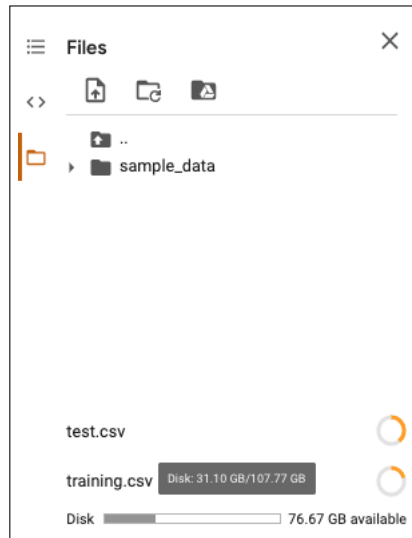


FIGURE 16 – Importer les bases de données pour être accessible par le notebook

Après que les fichiers sont importés, on peut vérifier qu'ils sont accessibles pour les prochaines étapes avec la commande suivante. Dans un notebook, lorsqu'une ligne de code commence avec un point d'exclamation (!), celle-ci réfère à une commande de terminal Linux. Dans le cas du segment de code 3, nous voulons afficher les fichiers sur le même niveau que le notebook.

```
1 !ls -lah
```

Segment de codes 3 – Voir les fichiers disponibles pour le notebook

```
1 !ls -lah

total 45M
drwxr-xr-x 1 root root 4.0K Jun 29 15:51 .
drwxr-xr-x 1 root root 4.0K Jun 29 14:50 ..
drwxr-xr-x 1 root root 4.0K Jun 25 17:02 .config
drwxr-xr-x 2 root root 4.0K Jun 29 14:51 .ipynb_checkpoints
-rw-r--r-- 1 root root 217K Jun 29 15:51 mystere.csv
drwxr-xr-x 1 root root 4.0K Jun 17 16:18 sample_data
-rw-r--r-- 1 root root 16M Jun 29 15:52 test.csv
-rw-r--r-- 1 root root 30M Jun 29 15:52 training.csv
```

FIGURE 17 – Résultat de la commande “ls -lah”

On voit bien, dans la figure 17, les fichiers “training.csv”, “test.csv” et “mystere.csv”.

3.8 Préparer les bases de données

Pour que les bases de données soient accessibles au modèle, il faut utiliser la fonction de lecture de fichier “.csv” offert par la librairie *Pandas* (Segment de codes 4).

```

1 # Transfert des datasets vers des variables (DataFrames)
2 train_ds = pd.read_csv("training.csv")
3 test_ds = pd.read_csv("test.csv")

```

Segment de codes 4 – Lire les bases de données

La méthode ‘head’ affiche les cinq premières valeurs d’une base de données. Tandis que la méthode ‘tail’ affiche les cinq dernières. Pour obtenir les informations générales à propos d’une base de données, il suffit d’utiliser la méthode ‘info’.

```

1 print("__Train__")
2 print(train_ds.info())
3 print(train_ds.head())
4 print(train_ds.tail())

```

Segment de codes 5 – Avoir de l’information de la base de données d’entraînement

La figure 18 est le résultat du segment de codes 5.

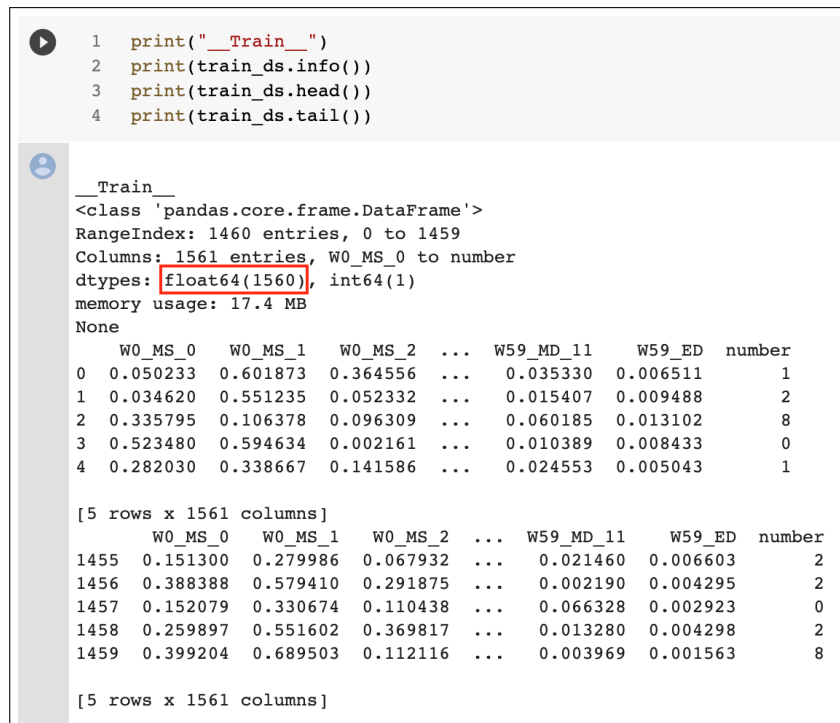


FIGURE 18 – Résultat de la recherche d’information de la base de données d’entraînement

Afin de concevoir la base de données de validation croisée avec la théorie du k-fold, nous allons concevoir une fonction (segment de codes 6) qui sera, par la suite, utilisée dans la fonction d’entraînement (“training”). Cette fonction retournera les bases de données d’entraînement, de validation croisée et de test formatées pour être directement utilisées dans la méthode “fit” du modèle de *Tensorflow*. Le k-fold est une technique où la base de données d’entraînement est

divisée en k parties. Avec cette segmentation, une des parties est utilisée pour des validations croisées et celles restantes seront utilisées pour l'entraînement du modèle.

```

1 def create_sets(train_ds, test_ds, n_fold=5):
2     """
3     CREATE_SETS
4
5     Transforme les bases de donnees afin d'être compatible avec l'entraînement
6     du modele avec la methode "fit"
7
8     @Arguments :
9         train_ds {pd.DataFrame} : Base de donnees d'entraînement
10        test_ds {pd.DataFrame} : Base de donnees de test
11        n_fold {int} : nombre de division pour le kfold
12
13    @Return :
14        {tensorflow.BatchDataset} : Base de donnees d'entraînement
15        {tensorflow.BatchDataset} : Base de donnees de validation croisee
16        {tensorflow.BatchDataset} : Base de donnees de test
17    """
18    # Melanger les valeurs de les bases de donnees
19    train = train_ds.sample(frac=1).reset_index(drop=True)
20    test = test_ds.sample(frac=1).reset_index(drop=True)
21    # Diviser les variables et les resultats
22    train_output = train.pop("number")
23    test_output = test.pop("number")
24
25    # Extraire les indexes pour le kfold une seule fois
26    for train_index, vc_index in KFold(n_fold).split(train) :
27        # Creer la base de donnees d'entraînement
28        X_train = (train.values)[train_index]
29        y_train = (train_output.values)[train_index]
30
31        # Creer la base de donnees de validation croisee
32        X_vc = (train.values)[vc_index]
33        y_vc = (train_output.values)[vc_index]
34
35        # Creer la base de donnees de test
36        X_test = test.values
37        y_test = test_output.values
38        break
39
40    # Transformer les bases de donnees pour etre utilisees directement dans fit
41    # Les donnees sont melangees une autre fois avant d'être retournees
42    train_set = tf.data.Dataset.from_tensor_slices((X_train, y_train))
43    train_set = train_set.shuffle(len(X_train))
44    train_set = train_set.batch(1)
45
46    vc_set = tf.data.Dataset.from_tensor_slices((X_vc, y_vc))
47    vc_set = vc_set.shuffle(len(X_vc))
48    vc_set = vc_set.batch(1)
49
50    test_set = tf.data.Dataset.from_tensor_slices((X_test, y_test))
51    test_set = test_set.shuffle(len(X_test))
52    test_set = test_set.batch(1)
53

```

54 `return train_set, vc_set, test_set`

Segment de codes 6 – Création des bases de données pour l’entraînement

La fonction “KFold” utilisée à l’intérieur de “create_sets” est conçue afin de produire k segmentations de bases de données. Dans notre cas, nous voulons faire qu’une seule segmentation, car le but n’est pas de faire l’évaluation du modèle, mais de montrer l’outil. Alors, la boucle “for in” est arrêtée après la première itération à l’aide du “break”.

3.9 Perceptron

3.9.1 Concevoir le modèle

Dans ce laboratoire, on s’intéresse à concevoir des perceptrons multicouches (MLP). Ces réseaux de neurones sont séquentiels soit que la sortie d’une couche est l’entrée d’une seule autre couche. Dans ce laboratoire, la couche de sortie doit posséder 10 neurones pour respecter la quantité de classe dans les bases de données.

Dans ce laboratoire, on utilise la fonction de perte “SparseCategoricalCrossentropy” qui a la particularité de faire le calcul sur un vecteur où toutes les valeurs sont zéro sauf l’index de la valeur voulue qui est mis à un. Dans nos bases de données, il y a dix résultats possibles 0,1,2,3,4,5,6,7,8 et 9. Alors, les vecteurs de sortie doivent être de taille 1 par 10 (voir table 1). Pour cette raison, la dernière couche doit toujours posséder dix neurones.

TABLE 1 – Représentation vectorielle des nombres

Nombre	Représentation
0	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
1	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
2	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
3	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
4	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
5	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
6	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
7	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
8	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
9	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

Dans Tensorflow, les modèles sont construits à l’aide d’agrégation de couches offertes par la librairie Keras. Lorsque le modèle est conçu, il est nécessaire de le compiler (avec la méthode “compile”) pour valider l’architecture pour l’entraînement. Afin de valider l’espace mémoire utiliser par les poids, il faut définir le datatype (dtype) de la couche. Dans notre cas, le *dtype* des couches doivent être de “float64” pour ne pas nécessiter une conversion de type avec les valeurs en entrée (voir figure 19)

```

__Train__
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Columns: 1561 entries, W0_MS_0 to number
dtypes: float64(1560), int64(1)
memory usage: 17.4 MB
None

```

FIGURE 19 – DataType de l'entrée

Tel que vu en classe, la fonction d'optimisation utilisée est la descente de gradient. Dans la librairie Tensorflow, cet algorithme est référé par le sigle SGD (Descente de Gradient Stochastique). D'autres fonctions d'optimisation sont disponibles et sont majoritairement des variantes de SGD. La fonction de perte définie dans la compilation du modèle est le "*SparseCategoricalCrossentropy*". Celle-ci est souvent utilisée pour la classification d'objet. La métrique observée dans notre cas est la précision du modèle. Soit, combien de fois le modèle obtient la bonne classe pour la donnée respective.

```

1 # Creer un modele sequentiel
2 perceptron = tf.keras.Sequential([
3     layers.Dense(10, activation='sigmoid', dtype='float64')
4 ])
5
6 # Compiler le modele pour l'entrainement
7 perceptron.compile(optimizer='SGD',
8                     loss= tf.keras.losses.SparseCategoricalCrossentropy(),
9                     metrics=['accuracy'])

```

Segment de codes 7 – Fonction pour l'entraînement du modèle

3.9.2 Concevoir une fonction personnalisée pour l'entraînement d'un modèle

Afin d'identifier les rendements du modèle avec les trois bases de données, il est intéressant de concevoir une fonction personnalisée pour l'entraînement du modèle. Le but de cette fonction est qu'elle retourne les statistiques obtenues lors de l'entraînement. Python offre le dictionnaire (*dict*) pour bien structurer les résultats. Les valeurs à l'intérieur d'un dictionnaire s'accèdent grâce aux clés de celui-ci (par exemple "mon_dict["ma_cle]"). Les clés du dictionnaire retourné par la fonction sont **train_accuracy**, **train_loss**, **vc_accuracy**, **vc_loss**, **test_accuracy** et **test_loss**.

```

1 def training(model, train_ds, test_ds, epochs=10, n_fold=5, verbose_train=1,
2             verbose_vc=1, verbose_test=1):
3     """
4     TRAINING
5
6     Fonction servant a entrainer un model tout en verifiant les capacites du
7     modele

```

```

6     sur les bases de donnees de validation croisee et de test
7
8     @Arguments :
9         model {tensorflow.python.keras.engine.sequential.Sequential} : Modele
tensorflow a entrainer
10         train_ds {pd.DataFrame} : Base de donnees d'entrainement
11         test_ds {pd.DataFrame} : Base de donnees de test
12         epochs {int} : Nombre d'epoques a faire pour l'entrainement du model
13         n_fold {int} : Nombre de fold pour le kfold
14         verbose_train {int} : 1 pour afficher l'avancement de l'apprentissage, 0
aucun affichage
15         verbose_vc {int} : 1 pour afficher l'avancement de l'evaluation, 0 aucun
affichage
16         verbose_test {int} : 1 pour afficher l'avancement de l'evaluation, 0
aucun affichage
17
18     @Return:
19         {dict} : 'train_accuracy' {list} : liste de la precision du modele sur la
base de donnees d'entrainement
20                 'train_loss' {list} : liste des resultat de la fonction de
perte du modele sur la base de donnees d'entrainement
21                 'vc_accuracy' {list} : liste de la precision du modele sur la
base de donnees de validation croisee
22                 'vc_loss' {list} : liste des resultats de la fonction de perte
du modele sur la base de donnees de validation croisee
23                 'test_accuracy' {list} : liste de la precision du modele sur la
base de donnees de test
24                 'test_loss' {list} : liste des resultats de la fonction de
perte du modele sur la base de donnees de test
25         """
26     # Initialisation des listes recevants
27     train_accuracy = []
28     train_loss = []
29     test_accuracy = []
30     test_loss = []
31     vc_accuracy = []
32     vc_loss = []
33
34     # Creer les bases de donnees pour l'entrainement
35     train_set, vc_set, test_set = create_sets(train_ds, test_ds, n_fold)
36
37     # Evaluer le modele avant l'entrainement
38     result = model.evaluate(train_set, verbose=verbose_train)
39     train_accuracy.append(result[1])
40     train_loss.append(result[0])
41     # ---
42     result = model.evaluate(vc_set, verbose=verbose_vc)
43     vc_accuracy.append(result[1])
44     vc_loss.append(result[0])
45     # ---
46     result = model.evaluate(test_set, verbose=verbose_test)
47     test_accuracy.append(result[1])
48     test_loss.append(result[0])
49
50     # Boucle d'entrainement
51     for e in range(epochs):
52         print('Epoch #{}'.format(e+1))
53         # ---

```



```

54     result = model.fit(train_set, epochs=1, verbose=verbose_train)
55     train_accuracy.append(result.history['accuracy'][-1])
56     train_loss.append(result.history['loss'][-1])
57     # ---
58     result = model.evaluate(vc_set, verbose=verbose_vc)
59     vc_accuracy.append(result[1])
60     vc_loss.append(result[0])
61     # ---
62     result = model.evaluate(test_set, verbose=verbose_test)
63     test_accuracy.append(result[1])
64     test_loss.append(result[0])
65
66     return {'train_accuracy':train_accuracy,
67           'train_loss':train_loss,
68           'vc_accuracy':vc_accuracy,
69           'vc_loss':vc_loss,
70           'test_accuracy':test_accuracy,
71           'test_loss':test_loss}

```

Segment de codes 8 – Fonction pour l’entraînement du modèle

3.9.3 Concevoir une fonction personnalisée pour afficher les statistiques de l’entraînement

Après l’entraînement d’un modèle, nous voulons savoir comment a été l’apprentissage du modèle. Pour ce faire, il est primordial de produire une figure affichant les statistiques produites par l’entraînement. Comme nous avons créé une fonction précédemment pour l’entraînement de modèle, nous pouvons utiliser les résultats de celui-ci. La fonction “training” retourne un dictionnaire avec un formatage précis qui sera facile à manipuler. Cette fonction réduira la manipulation pour les prochaines étapes.

```

1 def figure(training_dict, subtitle, figsize=(12,6)):
2     """
3     FIGURE
4
5     Produit une figure avec les statistiques de la fonction 'training'
6
7     @Arguments :
8         training_dict {dict(list)} : Dictionnaire retournee par la fonction
          training
9         figsize {Tuple} : Dimension de la figure
10    """
11    # Demarre une figure avec la taille (en pouce) donner en arguments
12    plt.figure(figsize=figsize,facecolor='w')
13
14    # Titre global de la figure
15    plt.suptitle(subtitle)
16
17    # Premiere sous-figure -> Precision du modele durant l'entrainement
18    plt.subplot(121)
19    plt.title('Precision du modele')
20    plt.xlabel('Nombre d\'epoques')
21    plt.ylabel('Precision de la classification')
22    # Produit une courbe pour chaque base de donnees
23    plt.plot(training_dict['train_accuracy'], 'r', label='training')

```

```

24 plt.plot(training_dict['vc_accuracy'], 'b--', label='vc')
25 plt.plot(training_dict['test_accuracy'], 'y', label='test')
26 # Limite de la precision pour rendre les valeurs entre 0% et 100% visibles
27 plt.ylim(0,1.01)
28 # Produire un grille en arriere-plan
29 plt.grid(b=True, which='major', color='#666666', linestyle='-.')
30 plt.minorticks_on()
31 plt.grid(b=True, which='minor', color='#999999', linestyle='-.', alpha=0.2)
32 plt.legend() # Afficher les labels des courbes
33
34 # Deuxieme sous-figure -> Resultat de la fonction de perte durant l'
    entraînement
35 plt.subplot(122)
36 plt.title('Perte du modele')
37 plt.xlabel('Nombre d\'epoques')
38 plt.ylabel('Resultat de la fonction de perte')
39 # Produire une courbe pour chaque base de donnees
40 plt.plot(training_dict['train_loss'], 'r', label='training')
41 plt.plot(training_dict['vc_loss'], 'b--', label='vc')
42 plt.plot(training_dict['test_loss'], 'y', label='test');
43 # S'assurer que la valeur minimum de l'ordonnee soit 0
44 plt.ylim(0);
45 # Produire un grille en arriere-plan
46 plt.grid(b=True, which='major', color='#666666', linestyle='-.')
47 plt.minorticks_on()
48 plt.grid(b=True, which='minor', color='#999999', linestyle='-.', alpha=0.2)
49 plt.legend()
50 return

```

Segment de codes 9 – Fonction pour l’affichage des statistiques de l’entraînement

La fonction “plt.subplot()” a comme argument un chiffre qui correspond à la disposition des sous-figures. La centaine indique combien de rangées de sous-figures seront dans la figure. La dizaine est pour le nombre de colonnes de sous-figures. Dans notre cas, nous aurons deux figures mises côte à côte telle que dans la figure 20. L’unité correspond à quelle est la sous-figure qui nous voulons formater.

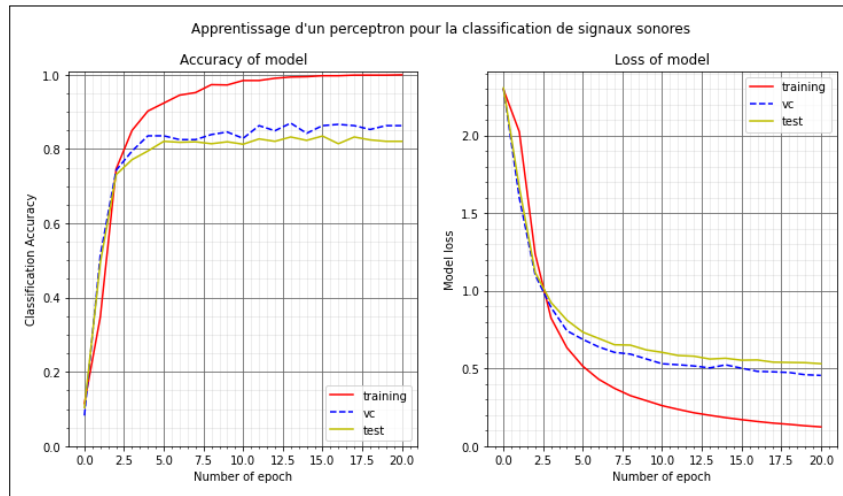


FIGURE 20 – Figure produite par la fonction “figure”

3.9.4 Entraîner le modèle

Avec la fonction “training” conçue précédemment, il est facile de faire l’entraînement du perceptron. Il suffit de mettre la variable contenant le perceptron, les deux bases de données (*train_ds* et *test_ds*) et le nombre d’époques voulu pour l’entraînement dans la fonction comme ci-dessous.

```
1 result = training(perceptron, train_ds, test_ds, epochs=25)
```

Segment de codes 10 – Entraînement du perceptron

Il est possible de réduire le nombre d’affichages dans la console à l’aide des arguments de verbose de la fonction “training”.

3.9.5 Afficher les statistiques du modèle

La fonction “figure” linéarise l’affichage graphique des résultats obtenus durant l’entraînement.

```
1 figure(result, "Apprentissage d'un perceptron pour la classification des signaux  
sonores")
```

Segment de codes 11 – Affichage des résultats de l’entraînement

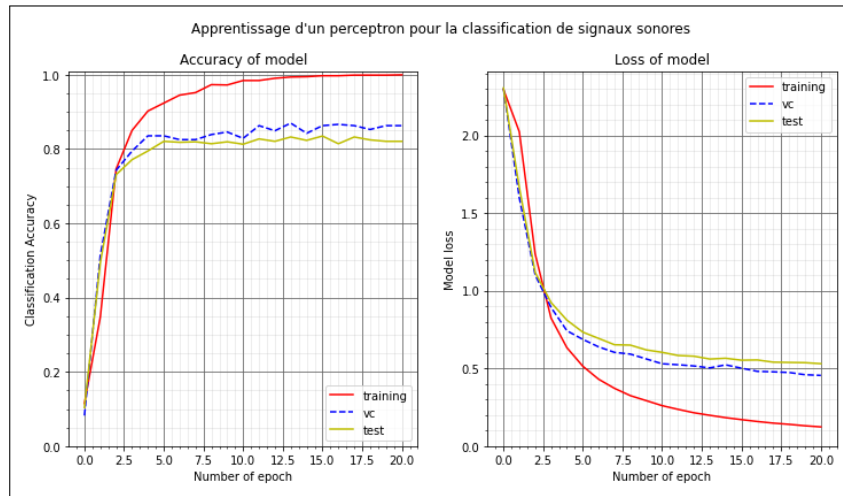


FIGURE 21 – Affichage des graphiques avec les résultats de l’entraînement du perceptron

3.9.6 Enregistrer le modèle

Afin de ne pas perdre ce perceptron, il est nécessaire de l’enregistrer. Tensorflow possède sa propre structure pour sauvegarder un modèle. Pour ce faire, il faut faire appel à la méthode “save”. Cette méthode nécessite le nom désiré pour le fichier de sauvegarde. Je recommande de mettre tous les fichiers de sauvegarde dans un seul dossier. Dans le segment de code 12, la sauvegarde est nommée “perceptron” et est mise à l’intérieur du dossier “modeles”.

```
1 perceptron.save('modeles/perceptron')
```

Segment de codes 12 – Enregistrer le Perceptron

3.10 MLP à deux couches

Après la création d’un perceptron, il est intéressant de voir qu’est-ce que peut apporter une couche supplémentaire à la classification des données.

3.10.1 Concevoir le modèle

Avec les connaissances développées pour la création du perceptron, faites la conception d’un perceptron multicouche (MLP) de deux couches. Après l’entraînement, il est possible de vérifier la structure du modèle avec la méthode “summary”.

```
1 mlp2c = tf.keras.Sequential([
2     layers.Dense(512, activation='sigmoid', dtype='float64'),
3     layers.Dense(10, activation='sigmoid', dtype='float64')
4 ])
5
6 # Compiler le modele pour l'entrainement
7 mlp2c.compile(optimizer='SGD',
```

```

8         loss= tf.keras.losses.SparseCategoricalCrossentropy(),
9         metrics=['accuracy'])

```

Segment de codes 13 – Conception d’un perceptron multicouches

Le segment de codes 14 est une technique utilisée pour produire des réseaux neurones avec un architecture plus flexible. Cette technique sera plus pertinente dans les prochains laboratoires.

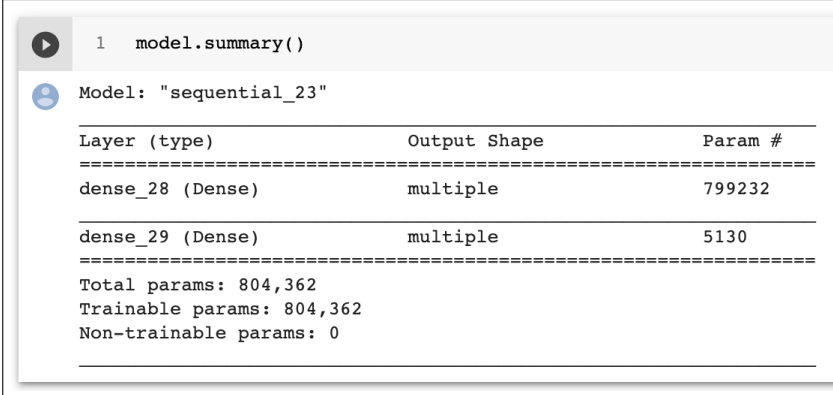
```

1 # Créer une couche d'entree
2 # La dimension des Tensors utilises est de (None, 1, 1560)
3 model_input = layers.Input(shape=(None, 1, train_ds.shape[1]-1))
4
5 # Créer les couches du reseau de neurones
6 x = layers.Dense(512, activation='sigmoid', dtype='float64')(model_input)
7 model = layers.Dense(10, activation='sigmoid', dtype='float64')(x)
8
9 # Valider le modele
10 model = tf.keras.Model(model_input, model)
11
12 # Compiler le modele pour l'entrainement
13 model.compile(optimizer='SGD',
14               loss= tf.keras.losses.SparseCategoricalCrossentropy(),
15               metrics=['accuracy'])

```

Segment de codes 14 – Conception d’un perceptron multicouches (technique supplémentaire)

La valeur de neurones dans la première couches n’est pas forcée telle que celle de la dernière couche. Alors, il est possible de la modifier afin de voir l’effet qu’ont les différentes quantité de neurones.



```

1 model.summary()

```

Model: "sequential_23"

Layer (type)	Output Shape	Param #
dense_28 (Dense)	multiple	799232
dense_29 (Dense)	multiple	5130

Total params: 804,362
Trainable params: 804,362
Non-trainable params: 0

FIGURE 22 – Exemple de l’utilisation de la méthode “summary” sur un modèle de deux couches denses

3.10.2 Entraîner le modèle

Faites l’entraînement du MLP à deux couches à l’aide de la fonction “training”.

Si vous obtenez des erreurs lors de l’entraînement, assurez-vous qu’il ait bien dix neurones pour la couche en sortie.

3.10.3 Afficher les statistiques du modèle

Affichez les résultats obtenus durant l'entraînement du MLP.

3.10.4 Enregistrer le modèle

Sauvegardez le modèle entraîné sous le même dossier avec le nom “mlp2c” pour MLP à deux couches.

3.11 MLP à deux couches et une couche d'abandon

Une problématique avec l'entraînement est le surentraînement soit lorsque le modèle se spécialise davantage sur les données que sur le concept. Une des solutions pour palier au surentraînement est l'ajout d'une couche d'abandon ou selon le terme original “Dropout” (Srivastava et al. (2014)).

3.11.1 Concevoir le modèle

Pour le laboratoire, nous allons mettre la couche d'abandon entre les deux couches totalement connectées (“Dense”). De cette manière, la dernière couche devra prendre en compte des données manquantes de la couche précédente pour concevoir un choix. Ajoutez la couche suivante entre les deux couches “Dense”.

```
1 layers.Dropout(0.2, dtype='float64')
```

Segment de codes 15 – Appel de la couche Dropout

Le premier argument pour l'initialisation de la couche Dropout est le pourcentage (entre 0 et 1) d'abandon de la couche précédente. L'abandon des neurones sera fait aléatoirement.

3.11.2 Entraîner le modèle

Faites l'entraînement du MLP à deux couches avec abandon à l'aide de la fonction “training”.

Si vous obtenez des erreurs lors de l'entraînement, assurez-vous qu'il ait bien dix neurones pour la couche en sortie et que toutes les couches sont du même dtype.

3.11.3 Afficher les statistiques du modèle

Affichez les résultats obtenus durant l'entraînement du MLP avec abandon.

3.11.4 Enregistrer le modèle

Sauvegardez le modèle entraîné sous le même dossier avec le nom “mlp2cd”.

3.12 MLP à trois couches

Après l'entraînement fait avec les perceptrons multicouches à deux couches, il est intéressant de voir qu'est-ce que peut apporter une autre couche supplémentaire à la classification des données.

3.12.1 Concevoir le modèle

Avec les connaissances développées précédemment, faites la conception d'un perceptron multi-couches (MLP) de trois couches sans couches d'abandon. Après l'entraînement, il est possible de vérifier la structure du modèle avec la méthode “**summary**”.

3.12.2 Entraîner le modèle

Faites l'entraînement du MLP à trois couches à l'aide de la fonction “**training**”.

Si vous obtenez des erreurs lors de l'entraînement, assurez-vous qu'il ait bien dix neurones pour la couche en sortie.

Lorsque la profondeur d'un modèle augmente, on peut apercevoir l'effet de la disparition du gradient. Celle-ci est causée par la fonction d'activation utilisée dans les couches du modèle. La disparition de gradient réfère au gradient qui tend vers zéro lorsque ses arguments tendent vers l'infini (positif ou négatif) (voir figure 24). Une fonction pouvant créer une disparition des gradients est la fonction sigmoïde. Une technique pour corriger la situation est d'utiliser la fonction d'activation “relu” au lieu de “sigmoïde” et d'ajouter une couche “Softmax” au modèle.

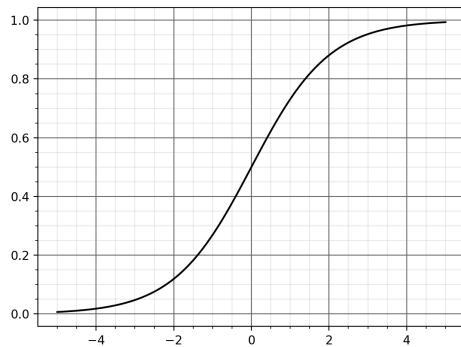


FIGURE 23 – Fonction d'activation Sigmoïde

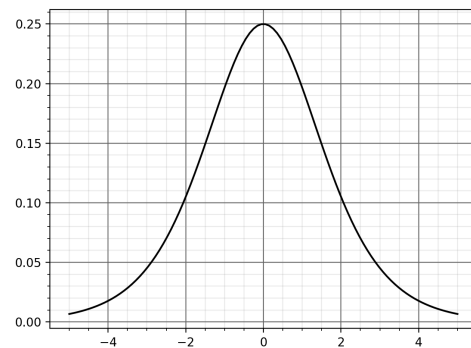


FIGURE 24 – Dérivée de la fonction d'activation Sigmoïde

Afin que votre MLP à trois couches puisse classifier les signaux sonores, changez le modèle pour celui du segment de code 16 et faites l'entraînement de celui-ci.

```
1 mlp3c = tf.keras.Sequential([
2     layers.Dense(512, activation='relu', dtype='float64'),
3     layers.Dense(128, activation='relu', dtype='float64'),
4     layers.Dense(10, activation='relu', dtype='float64'),
5     layers.Softmax(dtype='float64')
6 ])
```

Segment de codes 16 – Correction de la disparition du gradient

3.12.3 Afficher les statistiques du modèle

Affichez les résultats obtenus durant l’entraînement du MLP à l’aide de la fonction “figure”.

3.12.4 Enregistrer le modèle

Sauvegardez le modèle entraîné sous le même dossier avec le nom “mlp3c”.

3.13 Déterminer la classe des segments audios mystères

Une troisième base de données a été offerte pour simuler la mise en production d’un modèle. Pour ce faire, cette base de données ne possède pas la colonne “number” référant à la signification du signal sonore. Afin de savoir qu’elle est le résultat correspondant à ces données, il serait intéressant de voir quels sont les résultats prédits par les modèles entraînés durant le laboratoire.

```
1 _mystere_ds = pd.read_csv("mystere.csv")
2 print(_mystere_ds.info())
3 mystere_ds = tf.data.Dataset.from_tensor_slices(_mystere_ds.values)
4 mystere_set = mystere_ds.batch(1)
```

Segment de codes 17 – Chargement de la base de données mystères

```
1 # Chargement du perceptron entraine precedement
2 model = tf.keras.models.load_model('modeles/perceptron')
3 # Obtenir seulement l'index de la valeur la plus grande de la reponse du modele
4 np.argmax(model.predict(mystere_set), axis=1)
```

Segment de codes 18 – Chargement du perceptron

Afin de voir les résultats obtenus pour tous les modèles du laboratoire, nous pouvons utiliser une boucle “for in”.

```
1 # Concevoir une liste vide prete a recevoir les resultats
2 result = []
3 # Voir la prediction de chaque modele
4 for model_dir in ['modeles/perceptron', 'modeles/mlp2c', 'modeles/mlp2cd', 'modeles/
mlp3c']:
5     model = tf.keras.models.load_model(model_dir)
6     prediction = model.predict(mystere_set)
7     # Retourne l'index ayant la plus grande valeur
8     result.append(np.argmax(prediction, axis=1))
9
10 # Transferer la liste en array de numpy
11 result = np.array(result)
12
13 # Afficher les resultats
14 print("Resultat de tous les modeles")
15 print(result)
16 print("Resultat commun entre les modeles")
17 print((stats.mode(result, axis=0))[0])
```

Segment de codes 19 – Prédire le résultat des données avec tous les modèles

Une fonctionnalité a noté est d’ajouter une couche “Softmax” à la suite d’un modèle avec la méthode “add”. Le segment de codes 20 fait que les valeurs en sortie du réseau de neurones est la probabilité que la sortie soit une classe donnée comparativement à une valeur arbitraire tel que donnée lors de l’entraînement.


```

1 model.add(layers.Softmax(dtype='float64'))
2 # Recompile le modele afin qu'il puissent faire la prediction
3 model.compile(optimizer='SGD',
4               loss= tf.keras.losses.SparseCategoricalCrossentropy(),
5               metrics=['accuracy'])

```

Segment de codes 20 – Utilisation de la méthode “add”

4 À mettre dans le rapport

Avec le rapport, il faut remettre le notebook (‘.ipynb’) et le dossier “modeles”

Pour télécharger le fichier du notebook, cliquez l’option “Télécharger .ipynb” (*“Download .ipynb”*) sous l’onglet fichier.

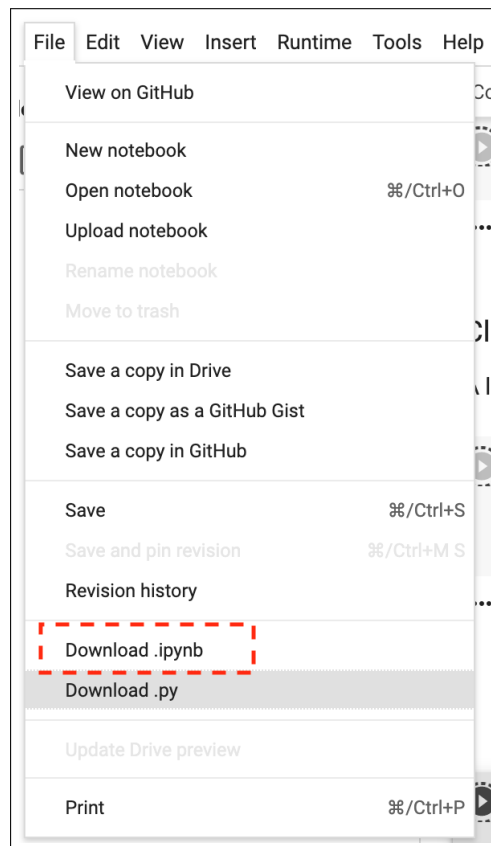


FIGURE 25 – Télécharger le notebook

Afin de télécharger le dossier “modeles”, il faut le compresser auparavant. Pour se faire, ouvrir une nouvelle cellule de code et insérer la commande ci-dessous.

```
1 !zip -r modeles.zip ./modeles
```

Segment de codes 21 – Compresser un fichier sur une session

Faites attention au point d'exclamation [!] au début de la commande

Par la suite, il est possible de télécharger en se dirigeant sur l'icône de fichier dans la barre de gauche du notebook tel que représenté dans la figure 26.

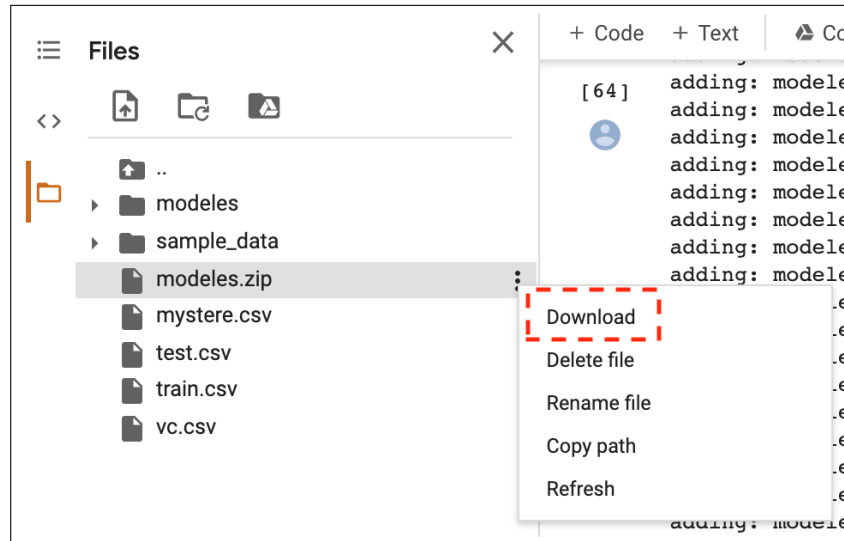


FIGURE 26 – Télécharger le dossier compressé

TABLE 2 – Contenu du rapport

Section	Taille minimum	Pondération
Introduction	3 lignes	5%
Théorie	5 lignes	20%
Résultats	5 lignes	40%
Conclusion	5 lignes	35%

4.1 Description des sections

4.1.1 Introduction

Dans l'introduction, il faut décrire le but du laboratoire et de résumer la procédure du laboratoire.

4.1.2 Théorie

Dans la section théorique, il est nécessaire d'expliquer les théories et les concepts qui sont mis de l'avant par le laboratoire.

4.2 Résultats

La section des résultats doit inclure les graphiques, les tableaux obtenus durant le laboratoire. Il est nécessaire d'inclure une figure décrivant l'entraînement pour chaque modèle. De plus, il est nécessaire d'expliquer le résultat observé à l'intérieur de la figure.

4.2.1 Conclusion

Finalement, la conclusion doit contenir un simple retour sur le but, une évaluation des résultats et une ouverture sur l'amélioration du laboratoire.

Références

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout : A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, (15) :1929–1958, 6 2014.