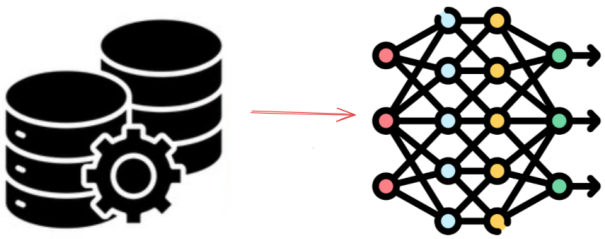
UNIVERSITÉ DE TOULON

UE73
TP 2
Pipeline entre données et modèle



Enseignant : Chareyre Adam

Master 1 Informatique | Université de Toulon | 16 décembre 2025

Résumé

L'objectif de ce TP est de vous faire mettre en œuvre, étape par étape, une pipeline complet d'apprentissage supervisé appliqué à la détection de pulses acoustiques à partir de données audio. À partir de spectrogrammes déjà construits, vous allez préparer les données en appliquant différentes méthodes de standardisation, puis les organiser en ensembles d'entraînement, de validation et de test.

Vous implémenterez ensuite un modèle de classification basé sur un perceptron multicouche (MLP) à l'aide de la bibliothèque PyTorch. Vous serez amenés à définir l'architecture du réseau, à initialiser correctement les paramètres, à choisir les hyperparamètres d'apprentissage et à écrire une boucle d'entraînement complète.

Enfin, vous évaluerez les performances de votre modèle à l'aide de plusieurs métriques vues en cours, calculées à chaque époque sur les ensembles d'entraînement et de validation. Ce TP a pour but de vous donner une vision concrète et cohérente des différentes étapes nécessaires à l'entraînement et à l'évaluation d'un modèle d'apprentissage automatique sur des données réelles.

Chapitre 1

Préparation du dataset et apprentissage supervisé

1.1 Question 1 – Standardisation des données

Avant l'entraînement d'un modèle de classification, il est essentiel de standardiser les données d'entrée afin de faciliter l'optimisation et d'accélérer la convergence.

On rappelle qu'une standardisation consiste à transformer les données selon la formule :

$$x' = \frac{x - \mu}{\sigma}$$

où μ est la moyenne et σ l'écart-type.

Il existe plusieurs stratégies possibles pour standardiser un jeu de spectrogrammes :

— **Standardisation indépendante par spectrogramme** : chaque spectrogramme X_i est standardisé avec sa propre moyenne μ_i et son propre écart-type σ_i.

— **Standardisation globale du dataset** : un seul couple (μ, σ) est calculé sur l'ensemble du dataset, puis appliqué à tous les spectrogrammes.

— **Standardisation par bande de fréquence** : chaque bande de fréquence est standardisée indépendamment (un couple (μ_f, σ_f) par fréquence).

1. Choisissez la méthode de standardisation des spectrogrammes qui vous paraît la plus pertinente dans le cadre de la détection de pulses acoustiques et implémentez là.
2. Justifiez brièvement votre choix (quel impact attendu sur l'apprentissage ?).

1.2 Question 2 – Construction des ensembles d'entraînement, de validation et de test

Un pipeline d'apprentissage supervisé nécessite de séparer les données en plusieurs ensembles disjoints :

— un **ensemble d'entraînement** (*trainset*) ;

— un **ensemble de validation** (*validset*) ;

— un **ensemble de test** (*testset*).

1. À partir du programme créé précedemment, executez le pour creer le dataset de train, valid et test sur les trois dossiers adaptés.
2. Pour chacun des ensembles (*train*, *valid*, *test*), sauvegardez :
   — les entrées dans un fichier `x.npy` ;
   — les étiquettes dans un fichier `y.npy`.

Si votre pipeline devient lente, vous êtes encouragés à optimiser votre code. Vous pouvez notamment utiliser :

— `tqdm` pour suivre la progression ;
— `cProfile` pour profiler l'exécution ;
— `snakeviz` pour visualiser les résultats du profilage.

1.3 Question 3 – Création de votre MLP

Vous allez maintenant implémenter un premier modèle de classification basé sur un perceptron multicouche (*Multi-Layer Perceptron*, MLP).

1. Écrivez une classe `MyMLP` héritant de `torch.nn.Module`.
2. Le réseau devra être composé de :
   — 4 couches linéaires (`Linear`) ;
   — une fonction d'activation `ReLU` après chaque couche linéaire ;
   — une couche de sortie suivie d'une activation adaptée à la classification binaire.
3. Initialisez les poids et les biais de chaque couche linéaire selon la méthode de Xavier (distribution uniforme), conformément à ce qui a été vu en cours.

Ajoutez également une couche de `Dropout` après chaque couche linéaire afin de limiter le sur-apprentissage.

1.4 Question 4 – Chargement du dataset avec PyTorch

Vous allez maintenant préparer vos données pour l'entraînement avec PyTorch.

1. Écrivez une fonction `load_dataset` qui :
   — charge les fichiers `x.npy` et `y.npy` d'un ensemble donné à l'aide de `np.load` ;
   — convertit les données en tenseurs PyTorch ;
   — instancie un objet `TensorDataset`.
2. Répétez cette opération pour les ensembles d'entraînement, de validation et de test.

Vous utiliserez ensuite ces `TensorDataset` pour créer des `DataLoader` adaptés à l'apprentissage par batch.

1.5 Question 5 – Boucle d'entraînement

Vous allez maintenant implémenter la boucle d'entraînement du modèle.

1. Définissez les hyperparamètres suivants :
   — le nombre d'époques d'entraînement ;
   — la taille des mini-lots (*batch size*) ;
   — le taux d'apprentissage (*learning rate*) ;
   — un optimiseur (par exemple `torch.optim.AdamW`) ;
   — un scheduler de taux d'apprentissage (par exemple `torch.optim.lr_scheduler.OneCycleLR`) ;
   — la fonction de coût `torch.nn.BCEWithLogitsLoss`.
2. Écrivez la boucle d'entraînement complète :
   — passage en mode entraînement ;
   — propagation avant ;
   — calcul de la perte ;
   — rétropropagation ;
   — mise à jour des poids ;
   — mise à jour du scheduler.

1.6 Question 6 – Évaluation et métriques

Afin d'évaluer les performances de votre modèle, vous devez implémenter les métriques vues en cours.

1. Écrivez les fonctions permettant de calculer les six métriques suivantes :
   — Accuracy ;
   — Précision ;
   — Recall ;
   — F1-score ;
   — taux de faux positifs ;
   — taux de faux négatifs.
2. À chaque époque, calculez et affichez ces métriques :
   — sur l'ensemble d'entraînement ;
   — sur l'ensemble de validation.