ARN: Practical Work 3

Rémi Ançay & Lucas Charbonnier

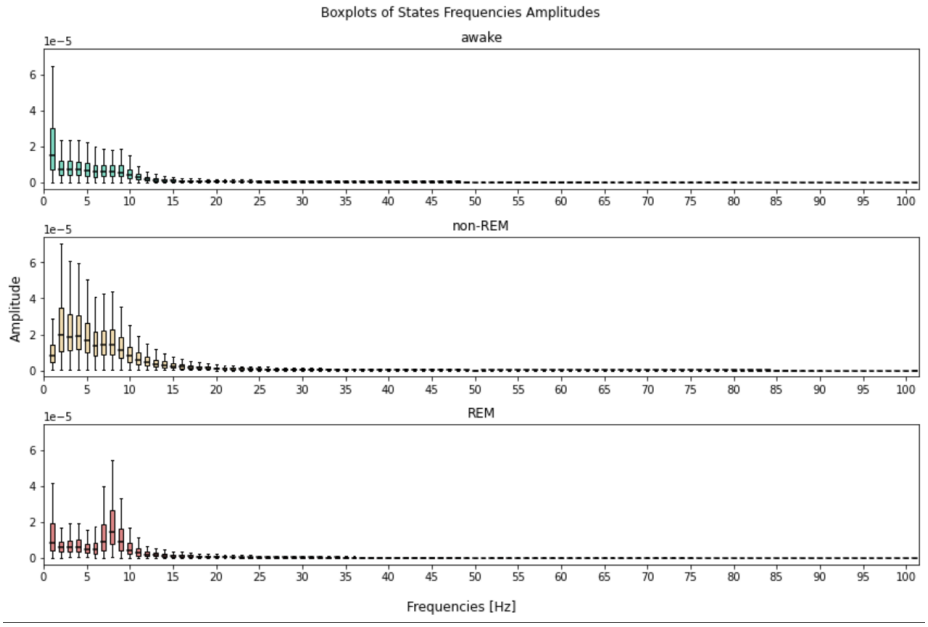
Rapport du travail pratique

Préparation des données

Pour commencer, il faut d'abord charger les données dans un dataframe pandas. Les données sont les même que pour le labo précédent et nous les avons chargées à l'aide de la commande read_csv() de pandas.

Sélection des features

Pour éviter que notre modèle ne soit trop lent à entraîner, nous n'avons gardé que 25 features parmi les 100 colonnes données dans le dataset original. Nous avons décidé de ne garder que les 25 premières fréquences car c'est celles qui montrent le plus de changement en fonction de l'état dans lequel la souris se trouvent, comme le montrent les boxplots suivants, extraits du labo précédent :



Normalisation des données

Afin que les données soient utilisables par un modèle de machine learning, nous avons également normalisé toutes les données du dataset. Nous avons utilisé le StandardScaler de la bibliothèque sklearn.

Préparation des folds pour la validation croisée

Nous avons mis en place une validation croisée à 3 folds, à l'aide de l'outil KFold de la bibliothèque sklearn.

Expérience 1 : Prédiction de deux classes

Création du modèle

Le modèle utilisé pour les deux expériences est un perceptron multicouches (MLP). Après quelques essais, nous avons trouvé que une seule couche cachée avec 2 neurones nous donnait des résultats suffisants. Augmenter le nombre de couches ou le nombre de neurones par couche n'a pas mené à de meilleurs résultats.

Au final, notre modèle est donc composé de 25 perceptrons en entrée (les 25 fréquences choisies), de deux perceptrons dans une couche cachée et avec une sortie composée d'un seul neurone (avec une valeur de 0 lorsque la souris est réveillée et 1 lorsque elle est endormie).

Nous avons choisie d'utiliser un sigmoïde comme fonction d'activation car elle donnait de meilleurs résultats qu'un ReLU (REctified Linear Unit) ou qu'une tangente hyperbolique. Nous avons également choisi un taux d'apprentissage de 0.01 et un « momentum » de 0.9.

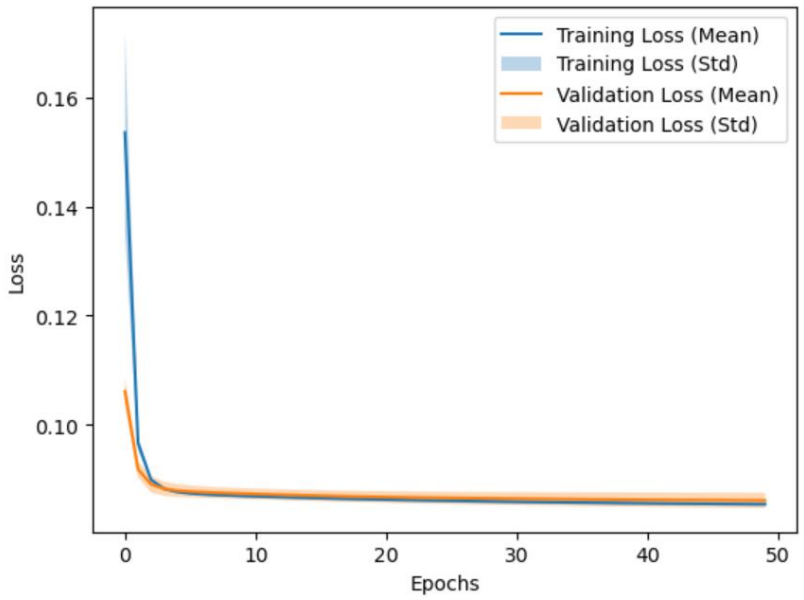
Entraînement

Avant d'entraîner le modèle, nous avons dû légèrement modifier les données en entrées.

Premièrement, il faut retirer la colonne « state » des données pour éviter que le modèle dispose de la réponse à ce qu'il essaie de prédire (pour des raisons évidentes). Ensuite, il faut changer les valeurs « attendues » car, pour l'instant, les données sont regroupées en 3 classes : « awake », « rem », « n-rem ».

En l'occurrence, on souhaite juste prédire si la souris est éveillée ou non donc on peut convertir ces 3 classes en seulement deux valeurs possibles : 0 pour « awake » et 1 pour « rem » ou « n-rem ». En pratique, nous avons effectué un replace sur les datasets de test/validation.

Voici un graphe des performances obtenues lors de l'entraînement et de la validation :

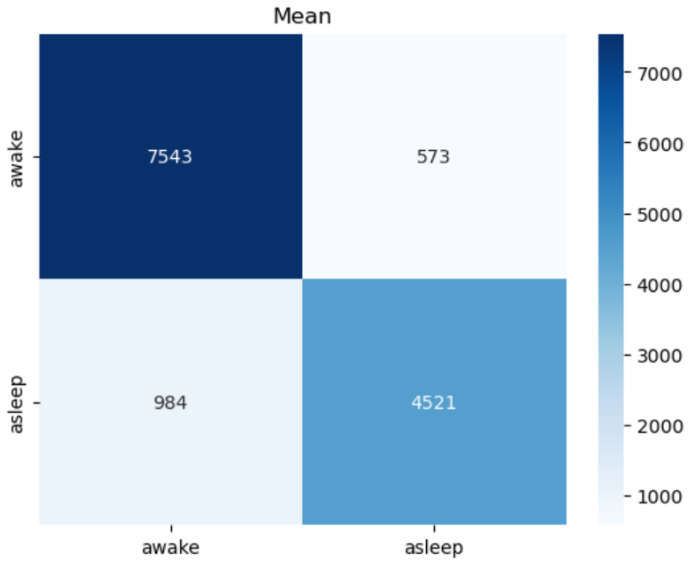


On peut voir que les résultats d'entraînement et de validation sont très proches, ce qui indique qu'il n'y a pas d'overfitting. Cependant, on voit également que notre modèle atteint très vite le maximum de son potentiel et commence rapidement à stagner.

Résultats obtenus et analyse

Le F1-score obtenu pour ce modèle est de 0.85, ce qui n'est pas mauvais mais pas non plus très haut.

Ci-dessous, la moyenne des matrices de confusion pour tous les folds :



Expérience 2 : Prédiction de trois classes

Création du modèle

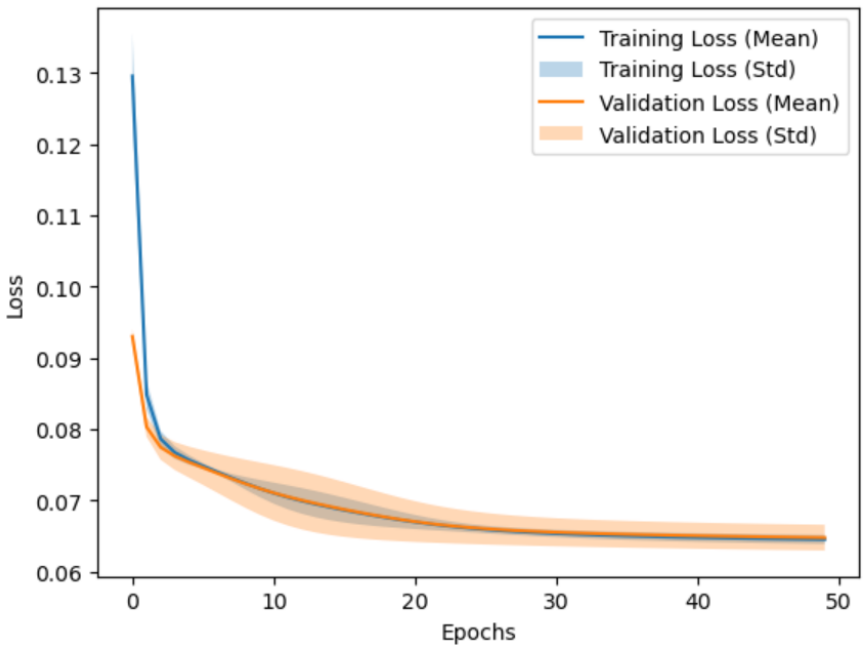
Pour cette deuxième expérience, nous avons dû modifier l'architecture de notre modèle pour qu'il puisse prédire trois classes différentes au lieu de 2.

Cela implique de changer la manière dont la prédiction est encodée en sortie. Le moyen le plus simple est d'utiliser un encodage « one-hot ». Il y aura donc 3 neurones de sorties et c'est celui qui a la valeur la plus élevée qui donnera la prédiction.

À part cette modification, nous avons utilisés les même paramètres que pour la première expérience.

Entraînement

Voici le graphe de l'entraînement de notre modèle :

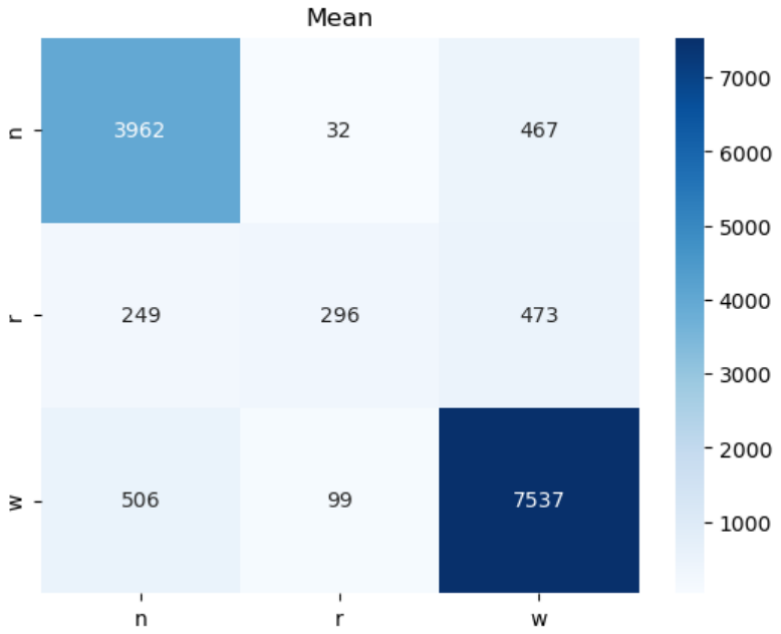


À la fin de l'entraînement, nous avons également dû transformer la sortie « one-hot » en « w,r,n » pour que l'on puisse créer une matrice de confusion.

Résultats obtenus et analyse

Ce nouveau modèle a obtenu un F1-Score de 0.86, ce qui est légèrement meilleur qu'avec le modèle précédent. Ce résultat est intéressant car on pourrait croire que les résultats seraient moins bons vu que le risque de se tromper de classe est plus grand étant donné qu'il y a 3 classes au lieu de seulement 2.

La matrice de confusion :



Compétition

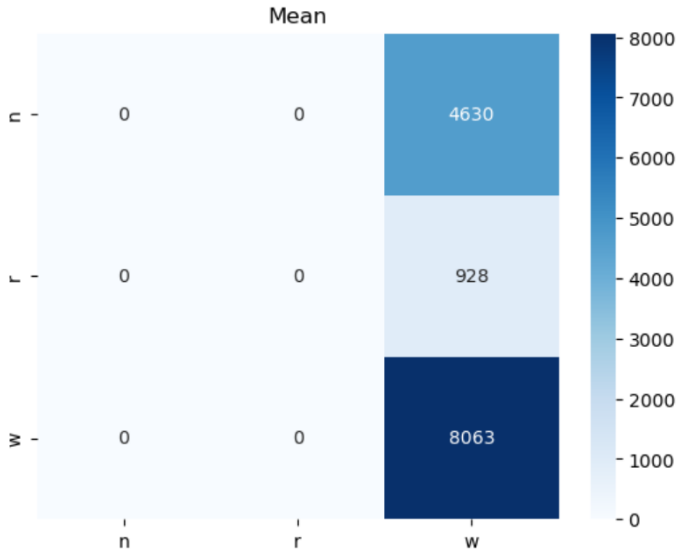
Nous n'avons pas pris le temps de pousser la compétition très loin, étant donné qu'il y avait d'autres labos à rendre (et un festival de musique à organiser, entre autres).

Cependant, nous avons quand même essayé 2 techniques pour améliorer les performances de notre modèle.

ACP

Tout d'abord, nous avons essayé d'utiliser un ACP sur toutes les fréquences enregistrées des souris pour laisser le modèle choisir les meilleures fréquences et voir si cela permettait d'obtenir de meilleurs résultats.

Comme on peut le voir sur la matrice de confusion ci-dessus, l'utilisation d'une ACP n'a pas très bien marché car apparemment tous les enregistrements étaient considérés comme « awake » donc il y a peut-être une erreur dans la manière dont nous l'avons implémenté. De plus, étant donné le petit nombre de features (une centaine), nous pensons que l'utilisation d'une ACP n'est pas forcément adapté dans cette situation.

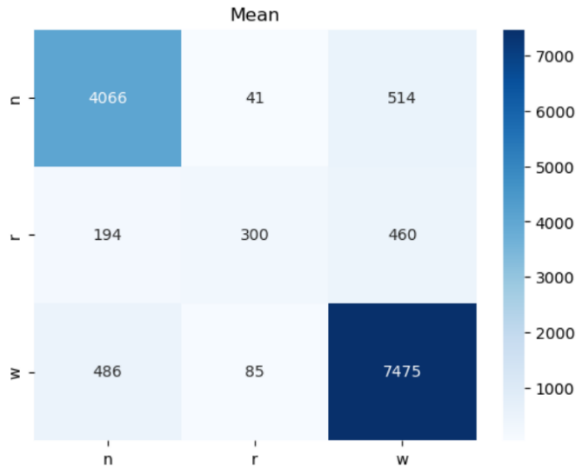


Changement de la « loss function »

Nous avons essayé de changer la loss function et utiliser une « categorical crossentropy » au lieu d'une « Mean Square Error (MSE) ».

C'est la seule modification que nous avons implémentée et qui nous a donné un meilleur résultat. Cela nous a permis de passer d'un F1-score de 0.86 à 0.87, donc plutôt négligeable.

Voici la matrice de confusion quand on utilise une « categorical crossentropy » :



Résultats

Nos résultats de prédiction pour le dataset « EEG_mouse_data_test » se trouvent dans le fichier **test_pred.npy**.