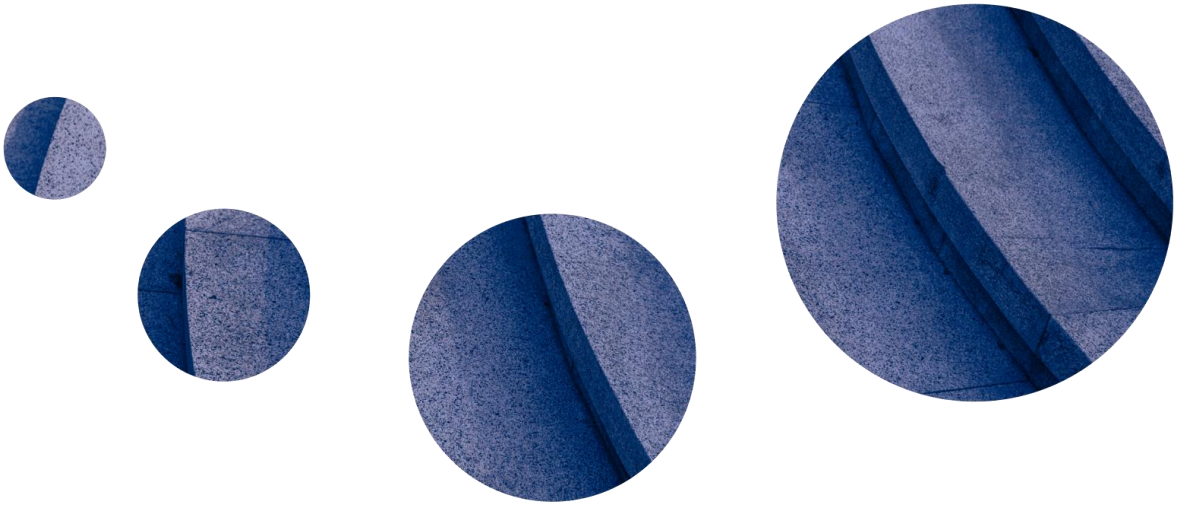


ARN – Laboratoire 03

Mice's sleep stages classification with MLP

Lestiboudois Maxime & Parisod Nathan

08/04/2025

Introduction

Dans ce laboratoire, nous nous sommes intéressés à la classification des stades de sommeil chez les souris à l'aide de réseaux de neurones multicouches (MLP).

Les différents stades de sommeil sont les suivants :

- Eveillé - awake
- Endormi profondément – n-rem
- Semi-endormi (sommeil paradoxal) – rem

Notre objectif est de concevoir, entraîner et évaluer des modèles permettant d'identifier au mieux les différents états du sommeil des souris à partir de signaux EEG.

Nous utiliserons plusieurs jeux de données issus de plusieurs souris (génétiquement identiques). Certains jeux de données seront utilisés pour entraîner nos modèles, et le reste sera utilisé pour effectuer un ensemble de tests.

À la suite d'une première phase de prétraitement des données dans laquelle nous avons normalisé les données, nous avons sélectionnés 25 caractéristiques pour mener différentes expériences et expérimenter leur modèle lié.

La première expérience consistera à distinguer les états "endormi" et "éveillé", tandis que la seconde expérience consistera à classifier les trois états.

Chaque modèle sera évalué à l'aide de la validation croisée, et ses performances seront mesurées et analyser à l'aide de matrices de confusion, de graphes et de score F1. Il s'agira également de déterminer si nos modèles sont en sur-apprentissage (overfitting) ou non.

Pour terminer, nous mettrons en œuvre le meilleur modèle possible pour déterminer les différents états de sommeil des souris afin de prendre part à la compétition interne. Pour cela, nous optimiserons au mieux en ajustant les hyperparamètres tel le changement de fonctions de coût, ou l'utilisation générale de nos différents outils.

Première expérience

Pour notre première expérience, notre objectif était de classifier les enregistrements EEG en deux catégories :

- Eveillé - Awake
- Endormi – Asleep

Préparation des données

Afin de réaliser une classification binaire, nous avons transformé les labels en deux classes.

- “w” => “awake” (éveillé)
- “n” et “r” => “asleep” (endormi)

Architecture du modèle - 1er essai

Nous avons conçu un perceptron multicouche (MLP) avec l’architecture suivante :

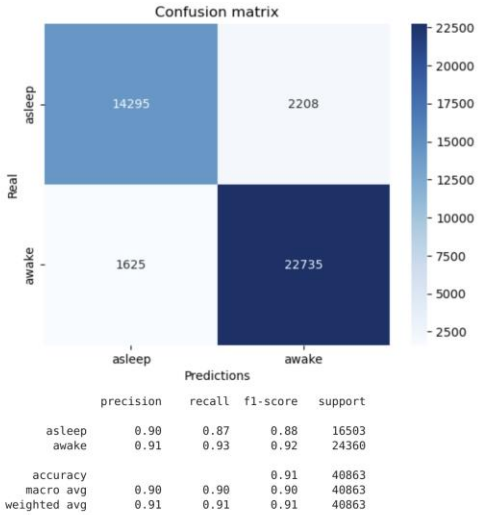
- Une couche d’entrée correspondant au nombre de features (25)
- Une première couche cachée de 64 neurones, avec la fonction d’activation ReLU.
- Une seconde couche cachée de 32 neurones, avec également la fonction d’activation ReLU.
- Une couche de sortie ne comportant qu’un seul et unique neurone avec la fonction d’activation sigmoïde. Cette fonction est particulièrement adaptée pour les tâches de classification binaire (ce qui est notre cas ici)

Nous avons compilé le modèle avec l’optimiseur **Adam**, la fonction de perte **Binary Cross-Entropy** et la métrique d’évaluation **Accuracy.**

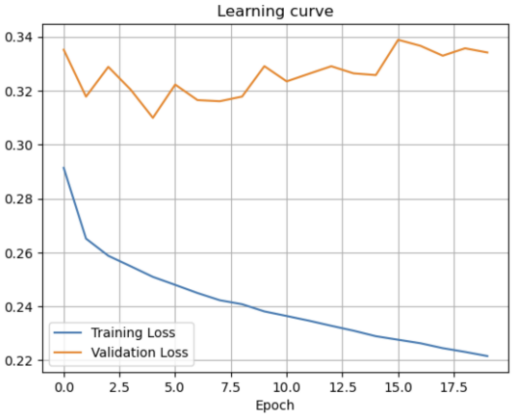
Entraînement et apprentissage – 1er essai

Le modèle a été entraîné pendant 20 epochs, avec un batch de taille 32. L’entraînement a été fait sur 20% des données pour la validation. Nous avions, comme précisé auparavant, normalisé nos données pour stabiliser et accélérer la convergence.

Nous avons imprimé le graphique d’apprentissage montrant l’évolution de la **loss** afin d’observer la dynamique d’apprentissage et les éventuels signes de surapprentissage (overfitting) tout au long des différentes epochs:



	precision	recall	f1-score	support
asleep	0.90	0.87	0.88	16503
awake	0.91	0.93	0.92	24360
accuracy			0.91	40863
macro avg	0.90	0.90	0.90	40863
weighted avg	0.91	0.91	0.91	40863



Analyse et ajustements

Nous remarquons ici que les résultats de performance de notre modèle sont relativement mauvais, surtout pour la courbe **Training Loss**. La courbe **Validation Loss**, a contrario, montre une certaine stabilité dans ses résultats malgré l'évolution du nombre d'epochs. Malgré cela, notre **matrice de confusion** et nos différents indices (comme le **F1-score**) restent étonnamment bons, ce qui montre que notre modèle fait de bonnes prédictions mais en étant peu sûr de lui.

Nous pouvons constater les points suivants :

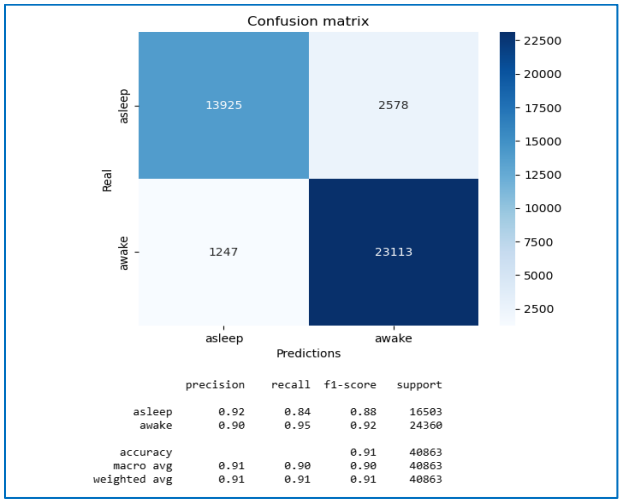
- **Loss d'entraînement** diminue régulièrement donc le modèle apprend bien sur les données d'entraînement.
- **Loss de validation** reste relativement stable, voire augmente légèrement ce qui indique un signe de **surapprentissage** (*overfitting*).
- Le **gap** croissant entre les deux courbes à partir d'environ l'époque 5 à 10 confirme cette tendance.

Le modèle apprend bien, mais sa capacité à généraliser diminue. Afin d'améliorer le modèle, nous avons décidé d'ajouter une couche cachée et de répartir les neurones ainsi : 64 -> 32 -> 16 avec une activation **ReLU** partout, sauf en sortie ou nous avons à nouveau utilisé une **sigmoïde**. Nous avons ajouté de la **BatchNormalization** après chaque couche et un dropout de 0,3 pour éviter **l'overfitting**. Nous avons également mis en place une condition **d'EarlyStopping** pour arrêter le modèle si ce dernier ne s'améliore plus.

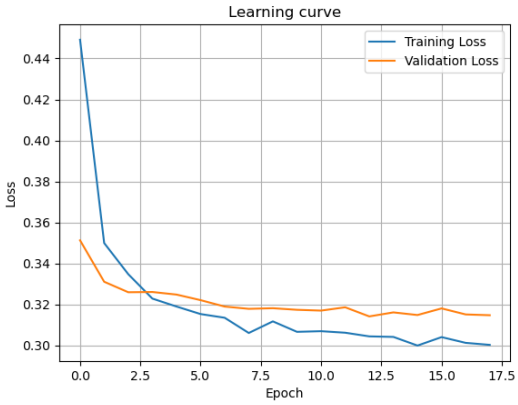
Afin de garder un code python propre, vous ne trouverez que la version optimale du code.

Analyse du 2ème essai

Voici la courbe de performance (**loss** vs **epochs)** et la **matrice de confusion** de notre 2ème essai :



	precision	recall	f1-score	support
asleep	0.92	0.84	0.88	16503
awake	0.90	0.95	0.92	24360
accuracy			0.91	40863
macro avg	0.91	0.90	0.90	40863
weighted avg	0.91	0.91	0.91	40863



Tout de suite, les résultats sont beaucoup plus pertinents. En effet, avec une **accuracy globale** de 91% et un **F1-score global** de 0,91 notre modèle a trouvé un bon équilibre entre précision et rappel sur nos deux classes. La **validation loss** est stable voire en légère baisse ce qui veut dire que nous n'avons **pas de surapprentissage.** La **training loss** descend régulièrement ce qui implique un bon apprentissage de notre modèle. Concernant la **matrice de confusion**, les quelques erreurs sont relativement équilibrées, ce qui est excellent.

Seconde expérience

Dans cette seconde expérience, l'objectif est de classifier les signaux EEG dans les trois états de sommeil suivants :

- Eveillé - **Awake**
- Endormi - **N-REM**
- Sommeil paradoxal - **REM**

Préparation des données

Contrairement à la première expérience, nous conservons ici les trois classes originales sans les regrouper. Chaque label ('w', 'n', 'r') est encodé avec un **LabelEncoder**, puis transformé en représentation **one-hot** (vecteurs de probabilité) à l'aide de **to_categorical**.

Nous avons également normalisé nos 25 caractéristiques avec un **StandardScaler**, comme dans la première expérience, afin d'assurer une convergence efficace du modèle.

Architecture du modèle – 1er essai

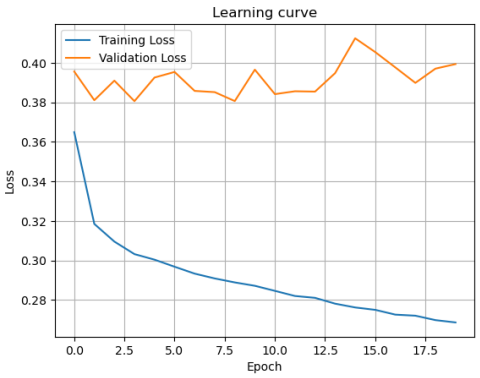
La première tentative de **MLP** était structurée comme suit :

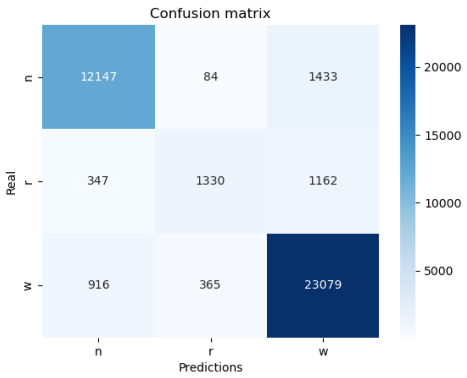
- Une couche d'entrée correspondant aux 25 features
- Deux couches cachées : 64 puis 32 neurones, activation **ReLU**
- Une couche de sortie avec 3 neurones (une par classe), activation **Softmax**

Le modèle a été compilé avec l'optimiseur **Adam**, la fonction de perte **Categorical Cross-Entropy** et la métrique d'évaluation **Accuracy**.

Entraînement et apprentissage – 1er essai

L'entraînement a été réalisé sur 20 epochs, avec un batch size de 32. La validation s'est faite sur 20 % du jeu de données. Nous avons généré la courbe de loss afin d'évaluer le comportement du modèle au cours de l'apprentissage :






```
1277/1277 ━━━━━━━━━━━━━━━━━━━━ 1s 544us/step
              precision    recall  f1-score   support

           n       0.91      0.89      0.90     13664
           r       0.75      0.47      0.58      2839
           w       0.90      0.95      0.92     24360

    accuracy                           0.89     40863
   macro avg       0.85      0.77      0.80     40863
weighted avg       0.89      0.89      0.89     40863
```

Analyse initiale des performances

Malgré une précision globale correcte (près de 89 %), nous avons constaté que la classe REM était fortement désavantagée. **Le recall de la classe r** n'était que de 0.47 avec un F1-score faible (~0.58). Cela signifiait que le modèle avait du mal à reconnaître cet état, souvent confondu avec **w** ou **n**.

La **validation loss** était instable voire en légère augmentation, signe possible de **surapprentissage**. Nous avons donc ajusté notre modèle.

Optimisation du modèle

Pour améliorer les performances et mieux détecter les phases de **REM**, nous avons modifié l'architecture du réseau de neurones en y intégrant :

- 3 couches cachées : 128 → 64 → 32 neurones
- Fonction d'activation **ReLU** à chaque couche
- Ajout de **Batch Normalization** et de **Dropout (0.3)** après chaque couche
- Activation **Softmax** en sortie
- Optimiseur changé en **Adamax** (plus stable)
- Ajout d'un **EarlyStopping** basé sur la validation loss

Enfin, comme la classe **r** était **sous-représentée**, nous avons introduit des poids de classe (class_weight) calculés automatiquement pour équilibrer l'influence de chaque classe durant l'entraînement.

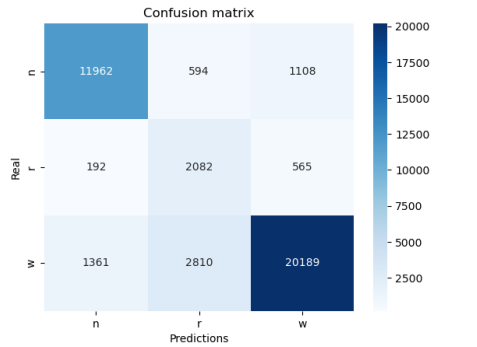
Note : Nous avons à nouveau décidé de ne garder que le code optimal dans le notebook.

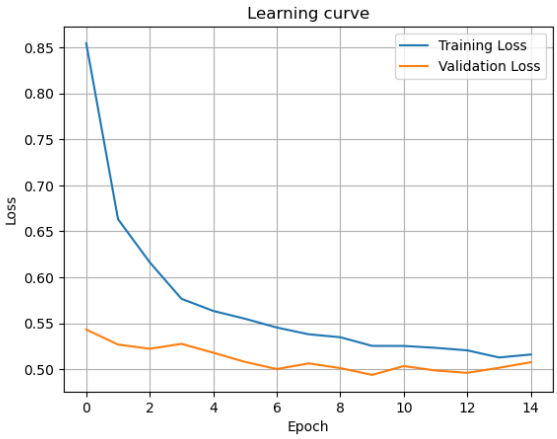
Analyse du modèle optimisé

Voici la courbe d'apprentissage et la matrice de confusion obtenues après optimisation :

	precision	recall	f1-score	support
n	0.89	0.88	0.88	13664
r	0.38	0.73	0.50	2839
w	0.92	0.83	0.87	24360
accuracy			0.84	40863
macro avg	0.73	0.81	0.75	40863
weighted avg	0.87	0.84	0.85	40863

```
# Generate and display the confusion matrix
cm = confusion_matrix(y_true, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_, yt
plt.xlabel('Predictions')
plt.ylabel('Real')
plt.title('Confusion matrix')
plt.show()
```





Le modèle a montré une amélioration significative, notamment sur la classe **REM** :

- Le **recall est passé à 0.73**, ce qui montre une bien meilleure détection
- La **précision** reste modérée (0.38), mais le modèle a su trouver un meilleur équilibre entre faux positifs et faux négatifs
- Les classes **n** et **w** restent bien identifiées (F1 de 0.88 et 0.87)
- L'**accuracy globale est de 84 %**, avec un **F1-score macro de 0.75**

Ces résultats démontrent que notre modèle, bien que perfectible, parvient à capturer les subtilités des **signaux EEG** pour discriminer les trois états du sommeil avec une performance satisfaisante. L'usage du **Dropout**, du **BatchNormalization**, des **class weights** et d'un entraînement contrôlé par **EarlyStopping** a été décisif pour éviter le surapprentissage et maximiser la généralisation.

Nous avons ainsi validé notre architecture MLP optimisée pour l'expérience 2.

Compétition

Pour la compétition, notre objectif était similaire à l'objectif de l'expérience 2, soit classifier les enregistrements EEG dans les trois catégories Eveillé - Sommeil profond – Sommeil paradoxal.

Préparation des données

Nous avons combiné les deux ensembles de données d'entraînement en un seul grand jeu de données sans mélange aléatoire afin de préserver la structure temporelle.

Nous avons sélectionné 25 features basées sur les amplitudes comme variables d'entrées.

Les labels (états) ont été encodés numériquement (0, 1, 2 pour les valeurs Eveillé - Sommeil profond et Sommeil paradoxal respectivement). Le fichier **test_pred.npy** contient cependant bel et bien le vecteur de prédiction avec des lettres.

Les features ont été normalisées afin de stabiliser et accélérer la convergence de notre réseau de neurones.

Sachant que la répartition des classes était déséquilibrée, nous avons appliqué une fonction "class_weight" afin de mieux équilibrer la contribution de chaque classe lors de la perte à l'entraînement.

Architecture du modèle

Nous avons conçu un modèle perceptron multicouche (MLP) avec les caractéristiques suivantes :

- Une couche d'entrée correspondant au nombre de features.
- Une première couche cachée de 128 neurones, avec la fonction d'activation ReLU, suivie d'une Batch Normalization et d'un Dropout de 30%.
- Une seconde couche cachée de 64 neurones, avec la fonction d'activation ReLU, suivie également d'une Batch Normalization et d'un Dropout de 30%.
- Une troisième couche cachée de 32 neurones, avec toujours la fonction d'activation ReLU, la Batch Normalization et le Dropout de 30%.
- Une couche de sortie composée de 3 neurones (pour nos trois classes) avec une fonction d'activation Softmax.

Nous avons compilé le modèle avec l'optimiseur **Adamax,** la fonction de perte **Categorical Cross-Entropy** et la métrique **Accuracy**.

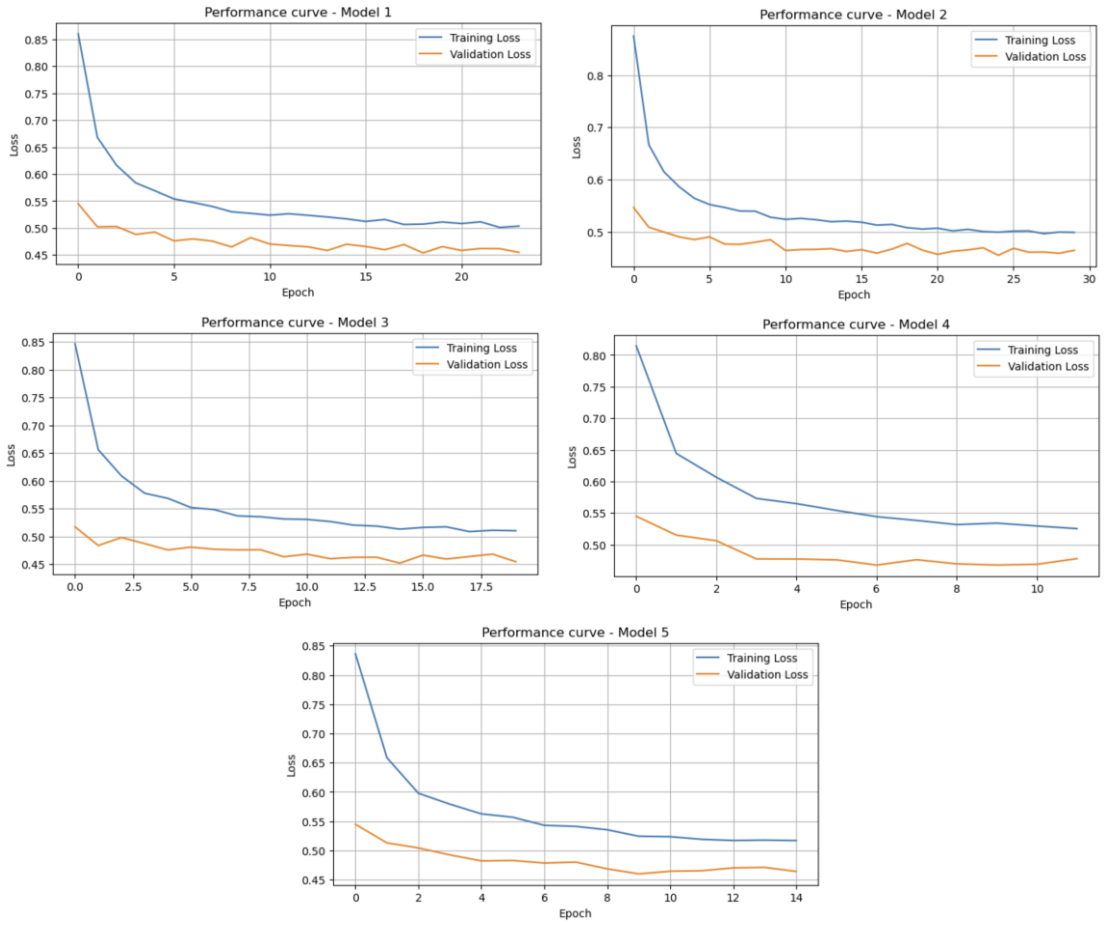
Entraînement et apprentissage

L'entraînement a été effectué en plusieurs étapes :

1) Validation croisée, **TimeSerieSplit** :
   Nous avons effectué une validation croisée en 10 parties (10-fold) sans mélange temporel. Pour chaque **fold**, nous avons appliqué un **EarlyStopping**, basé sur la perte de validation, avec une patience de 3 epochs.
2) Entraînement final avec 5 modèles indépendants :

Afin de créer un ensemble de modèles, nous avons entraîné 5 modèles supplémentaires, avec 90% de données (10% restants pour validation). Chaque modèle a été entraîné pendant maximum de 50 epochs, avec un **EarlyStopping** (patience de **5 epochs**)

Nous avons imprimé les graphiques d'apprentissage montrant l'évolution de la **loss** afin d'observer la dynamique d'apprentissage et les éventuels signes de **surapprentissage** (overfitting) tout au long des différentes epochs des différents modèles :

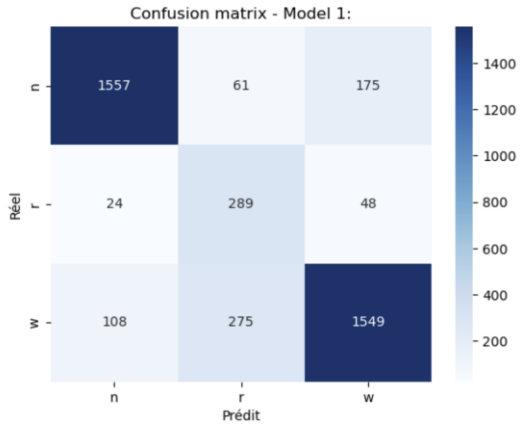


Nous observons ici que les loss d'entraînement diminuent progressivement, montrant un bon apprentissage des données. Les loss de validation restent globalement stables, avec quelques oscillations, signe que nos modèles généralisent plutôt bien, sans surapprentissage marqué.

L'ajout de **Batch Normalization** et de **Dropout** semblent également avoir permis de contrôler et amoindrir le surapprentissage de manière efficace.

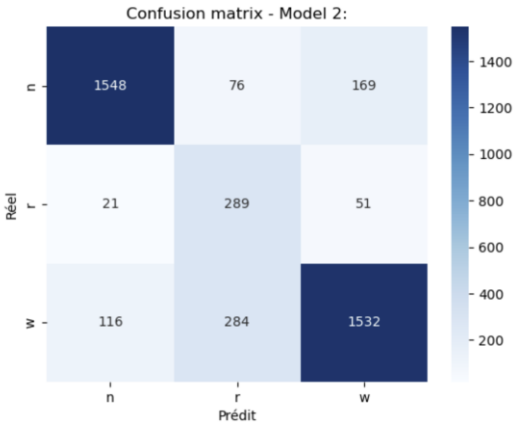
Evaluation des performances

Pour chaque modèle, nous avons généré une matrice de confusion afin de visualiser les erreurs de classifications :



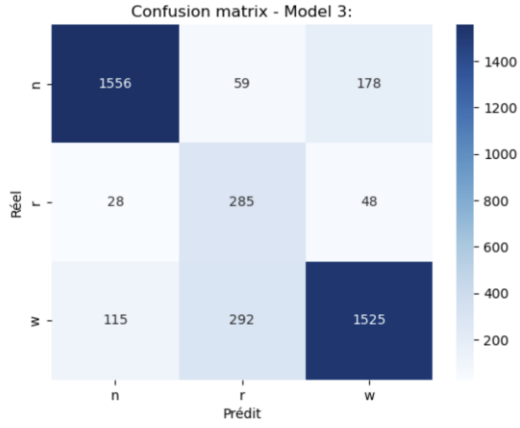
Classification report - Model 1:

	precision	recall	f1-score	support
n	0.92	0.87	0.89	1793
r	0.46	0.80	0.59	361
w	0.87	0.80	0.84	1932
accuracy			0.83	4086
macro avg	0.75	0.82	0.77	4086
weighted avg	0.86	0.83	0.84	4086



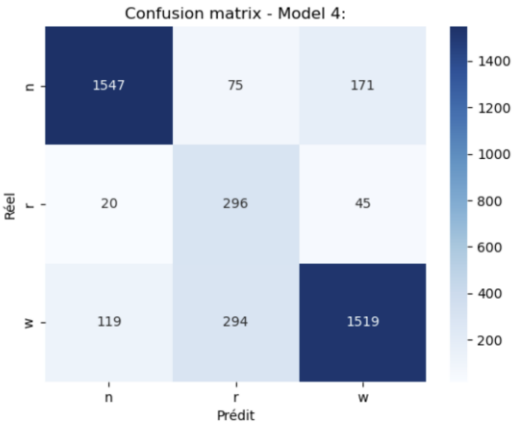
Classification report - Model 2:

	precision	recall	f1-score	support
n	0.92	0.86	0.89	1793
r	0.45	0.80	0.57	361
w	0.87	0.79	0.83	1932
accuracy			0.82	4086
macro avg	0.75	0.82	0.76	4086
weighted avg	0.86	0.82	0.83	4086



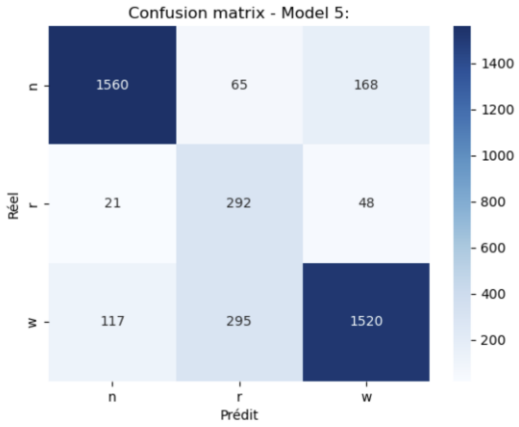
Classification report - Model 3:

	precision	recall	f1-score	support
n	0.92	0.87	0.89	1793
r	0.45	0.79	0.57	361
w	0.87	0.79	0.83	1932
accuracy			0.82	4086
macro avg	0.74	0.82	0.76	4086
weighted avg	0.85	0.82	0.83	4086



Classification report - Model 4:

	precision	recall	f1-score	support
n	0.92	0.86	0.89	1793
r	0.45	0.82	0.58	361
w	0.88	0.79	0.83	1932
accuracy			0.82	4086
macro avg	0.75	0.82	0.76	4086
weighted avg	0.86	0.82	0.83	4086



Classification report - Model 5:

	precision	recall	f1-score	support
n	0.92	0.87	0.89	1793
r	0.45	0.81	0.58	361
w	0.88	0.79	0.83	1932
accuracy			0.83	4086
macro avg	0.75	0.82	0.77	4086
weighted avg	0.86	0.83	0.83	4086

Les résultats montrent des précisions élevées pour les classes **awake** et **nRem**, tandis que la classe **Rem** montrent plus de difficulté à être prédite, sans doute à cause de sa proximité à la fois avec les catégories awake et asleep, ou de sa plus faible représentation dans le dataset.

Pour finir sur nos prédictions finales sur l'ensemble test, nous avons moyenné les probabilités produites par les 5 modèles entraînés. Le label final correspond à la classe ayant la moyenne maximale. Nous pouvons retrouver ces prédictions dans le fichier "test_pred.npy".

Remarque

Notre stratégie choisie d'utiliser un modèle MLP simple enrichi avec Dropout et la Batch Normalization, combinée avec l'ensemble de nos modèles, nous a permis d'atteindre de très bonnes performances tout en limitant le risque de surapprentissage.

Pour améliorer encore nos résultats, il serait possible d'essayer des architectures plus complexes, ou d'optimiser davantage les hyperparamètres.

Conclusion et perspectives

Nous avons développé trois types de modèle de classification des états **nRem**, **Rem** et **Awake** par rapport aux EEG des souris, tous trois basés sur un **MLP optimisé**.

Cette approche a permis d'obtenir des modèles fiables en limitant **l'overfitting**.

Les performances obtenues sont très satisfaisantes, avec une bonne séparation des états de vigilance et des scores de précision élevés. Quelques erreurs de classification subsistent, notamment entre **NREM** et **REM**, ce qui est attendu compte tenu de la similarité biologique entre ces états.

Globalement, notre approche s'est révélée efficace et adaptée aux spécificités des données EEG.
Des pistes d'amélioration supplémentaires pourraient inclure un choix plus fin des hyperparamètres ou l'exploration d'architectures plus complexes.