Réseau de Neurones - Classification

Réseau de Neurones (MLP) avec sortie Softmax

Théorie

Un réseau de neurones multi-couches (**MLP - Multi-Layer Perceptron**) est un modèle d'apprentissage supervisé basé sur des couches de neurones artificiels. Il est particulièrement efficace pour la classification non linéaire.

Dans notre cas, nous utilisons **une couche de sortie Softmax**, qui permet de normaliser les sorties du réseau en probabilités pour une classification multiclasses.

Hyperparamètres

Lors de l'entraînement d'un réseau de neurones, plusieurs hyperparamètres influencent la performance :

- **Taille du pas d'apprentissage (`learning_rate`)** : Déterminée automatiquement avec l'optimiseur Adam.
- **Nombre de couches cachées et neurones par couche** : Influence la capacité d'apprentissage du modèle.
- **Nombre d'époques (`epochs`)** : Nombre de fois que le modèle parcourt l'ensemble des données d'entraînement.
- **Taille du batch (`batch_size`)** : Nombre d'exemples utilisés pour calculer une mise à jour des poids.

Nous allons utiliser **Adam avec un taux d'apprentissage adaptatif**, ce qui signifie que le pas d'apprentissage s'ajustera automatiquement au fil des itérations.

Évaluation des performances

Comme pour les autres modèles de classification, nous utilisons :

- **Matrice de confusion** : Un tableau qui compare les classes réelles aux classes prédites. Chaque ligne représente une classe réelle et chaque colonne une classe prédite. Une bonne classification est indiquée par des valeurs élevées sur la diagonale principale, tandis que les erreurs de classification apparaissent en dehors de cette diagonale.
- **Accuracy (Précision globale)** : Le pourcentage de prédictions correctes parmi l'ensemble des données. Elle est utile lorsque les classes sont équilibrées, mais peut être trompeuse si certaines classes sont largement majoritaires.
- **Precision (Précision par classe)** : Pour une classe donnée, la précision mesure la proportion de prédictions correctes parmi toutes celles où le modèle a prédit cette classe. Une haute précision signifie que lorsqu'une classe est prédite, elle est souvent correcte.
- **Recall (Rappel par classe)** : Le rappel mesure la proportion de vrais exemples d'une classe qui sont correctement détectés par le modèle. Une haute valeur signifie que le modèle détecte bien les échantillons appartenant à cette classe.
- **F1-score** : Moyenne harmonique entre la précision et le rappel. Il équilibre ces deux mesures et est particulièrement utile lorsque les classes sont déséquilibrées.

Une **moyenne macro** est utilisée pour donner un aperçu global des performances du modèle en attribuant un poids égal à chaque classe. Une **moyenne pondérée** (weighted avg) prend en compte la fréquence de chaque classe afin de refléter plus précisément la performance globale dans un contexte de classes déséquilibrées.

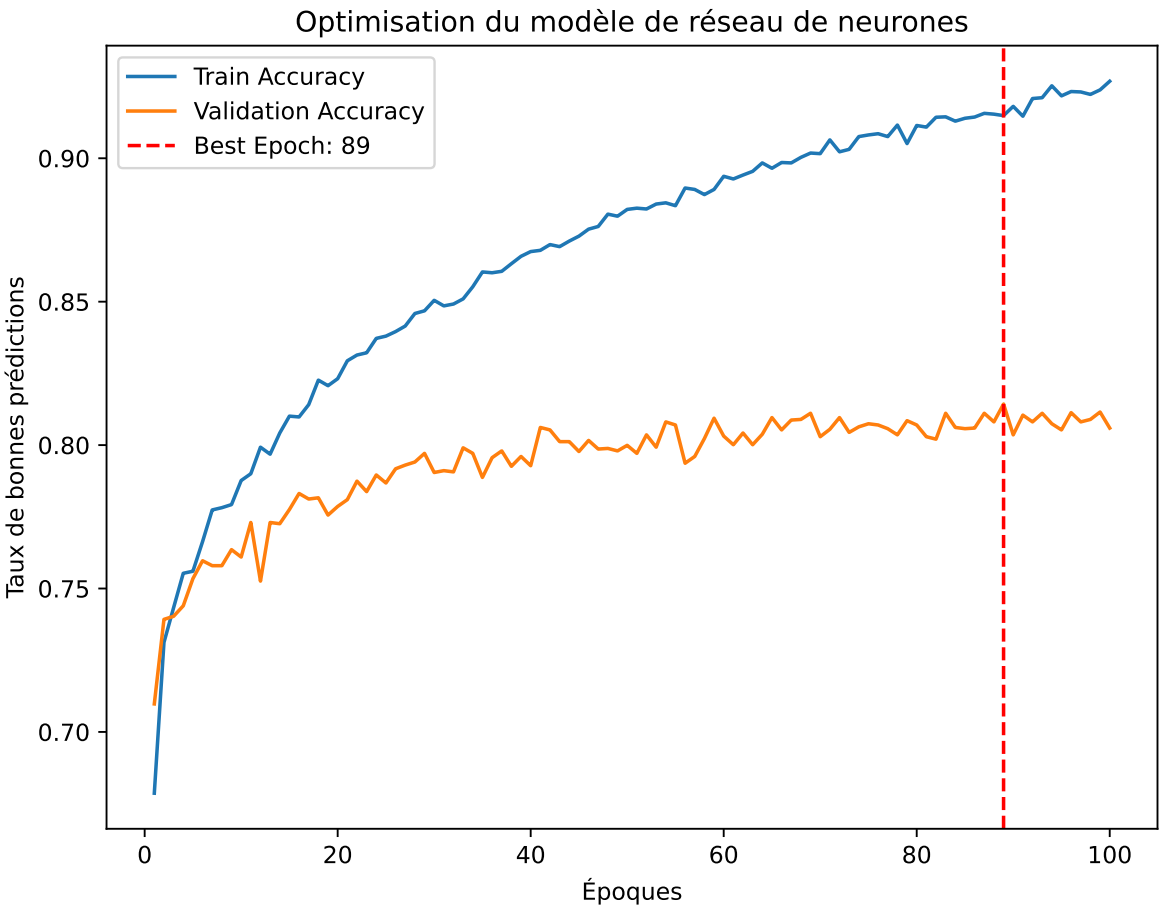
Exemple en Python

```
import os
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

#  Désactivation des logs TensorFlow et des avertissements inutiles
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"  # Supprime les logs TensorFlow
tf.get_logger().setLevel("ERROR")  # Supprime les logs internes de TensorFlow
```

```
#  Chargement des ensembles de données
train_data = pd.read_csv('covertype_train.csv')
val_data = pd.read_csv('covertype_val.csv')
test_data = pd.read_csv('covertype_test.csv')

#  Préparation des données
X_train = train_data.drop('Cover_Type', axis=1)
y_train = train_data['Cover_Type'] - 1  # Ajustement des labels pour correspondre à l'indexat

X_val = val_data.drop('Cover_Type', axis=1)
y_val = val_data['Cover_Type'] - 1

X_test = test_data.drop('Cover_Type', axis=1)
y_test = test_data['Cover_Type'] - 1

#  Normalisation des données
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

#  Définition du modèle
model = keras.Sequential([
    keras.layers.Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(len(set(y_train)), activation='softmax')
])

#  Compilation du modèle avec Adam (sans CUDA)
optimizer = keras.optimizers.Adam()
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy', metrics=['accuracy

#  Entraînement du modèle (sans affichage de logs)
history = model.fit(X_train, y_train, epochs=100, validation_data=(X_val, y_val), batch_size=

#  Détermination de la meilleure époque
best_epoch = history.history['val_accuracy'].index(max(history.history['val_accuracy'])) + 1
best_val_acc = max(history.history['val_accuracy'])

#  Affichage des courbes d'entraînement
plt.figure(figsize=(8, 6))
plt.plot(range(1, 101), history.history['accuracy'], label='Train Accuracy')
```

```
plt.plot(range(1, 101), history.history['val_accuracy'], label='Validation Accuracy')
plt.axvline(best_epoch, color='r', linestyle='--', label=f'Best Epoch: {best_epoch}')
plt.xlabel("Époques")
plt.ylabel("Taux de bonnes prédictions")
plt.title("Optimisation du modèle de réseau de neurones")
plt.legend()
plt.show()

#  Ré-entraîner le modèle avec la meilleure époque (sans logs)
model.fit(X_train, y_train, epochs=best_epoch, batch_size=32, verbose=0)

#  Évaluation sur l'ensemble de test (sans logs)
y_test_pred = model.predict(X_test, verbose=0)
y_test_pred_classes = y_test_pred.argmax(axis=1)

#  Affichage de la matrice de confusion et des métriques finales
conf_matrix = confusion_matrix(y_test, y_test_pred_classes)
print(f"\n Meilleure époque : **{best_epoch}** avec une précision de validation de **{best_v
print("\n Matrice de confusion (les lignes = classes réelles, les colonnes = classes prédite
print(conf_matrix)

print("\n Évaluation sur l'ensemble de test")
print(classification_report(y_test, y_test_pred_classes))
```

```
2025-02-20 20:33:07.518620: I tensorflow/core/util/port.cc:153] oneDNN custom operations are
2025-02-20 20:33:07.529403: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Un
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1740079987.541040  574191 cuda_dnn.cc:8310] Unable to register cuDNN factory: Att
E0000 00:00:1740079987.544559  574191 cuda_blas.cc:1418] Unable to register cuBLAS factory: A
2025-02-20 20:33:07.558110: I tensorflow/core/platform/cpu_feature_guard.cc:210] This Tensorf
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild Tensorf
/home/ensai/.local/lib/python3.10/site-packages/keras/src/layers/core/dense.py:87: UserWarnin
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
2025-02-20 20:33:08.757949: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152]
```



Meilleure époque : **89** avec une précision de validation de **0.8143**

Matrice de confusion (les lignes = classes réelles, les colonnes = classes prédites) :

```
[[1385  292    3    0    5    2   21]
 [ 296 1914   16    0   11   24    0]
 [   1   15  237    4    1   23    0]
 [   0    0    6   12    0    3    0]
 [  10   21    2    0   40    1    0]
 [   0   11   44    2    2   85    0]
 [  30    4    0    0    0    0  125]]
```

Évaluation sur l'ensemble de test

	precision	recall	f1-score	support
0	0.80	0.81	0.81	1708

1	0.85	0.85	0.85	2261
2	0.77	0.84	0.80	281
3	0.67	0.57	0.62	21
4	0.68	0.54	0.60	74
5	0.62	0.59	0.60	144
6	0.86	0.79	0.82	159
accuracy			0.82	4648
macro avg	0.75	0.71	0.73	4648
weighted avg	0.82	0.82	0.82	4648