Bootcamp IA — Atelier Guidé : Réseaux de neurones basiques (Keras & Scikit-learn)

Département IA & Ingénierie des Données
Institut Supérieur d'Informatique (ISI Dakar)

Samedi 20 Septembre 2025
11h15 – 13h00

Objectif de l'atelier

Cet atelier s'adresse aux étudiants de Master. Il vise à :

- Découvrir l'implémentation d'un **réseau de neurones simple** pour la classification.
- Comparer la version **Scikit-learn (MLPClassifier)** et la version **Keras (Sequential API)**.
- Visualiser la courbe d'apprentissage et interpréter les résultats.

Données utilisées

Nous travaillerons avec le dataset **MNIST (chiffres manuscrits)** disponible via `Keras.datasets`.

- Images : 28x28 pixels (niveaux de gris).
- Classes : 10 (chiffres de 0 à 9).

Organisation des participants

- Répartis en **groupes de 5**.
- Environnement : Python + GPU si disponible (Google Colab recommandé).
- Durée : **1h45**.
- Livrables : code + graphiques + mini-analyse des performances.

Plan de travail

1. **Chargement et prétraitement des données**.
2. **Réseau de neurones basique avec Scikit-learn (MLPClassifier)**.
3. **Réseau de neurones avec Keras Sequential**.
4. **Comparaison des résultats et discussion**.

Code de départ

Préparation des données MNIST

```python
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

# Charger MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Normaliser les pixels entre 0 et 1
X_train = X_train.reshape(-1, 28*28) / 255.0
X_test  = X_test.reshape(-1, 28*28) / 255.0

# One-hot encoding pour Keras
y_train_cat = to_categorical(y_train, 10)
y_test_cat  = to_categorical(y_test, 10)

print("Train:", X_train.shape, y_train.shape)
print("Test :", X_test.shape, y_test.shape)
```

Réseau de neurones avec Scikit-learn (MLPClassifier)

```python
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report

mlp = MLPClassifier(hidden_layer_sizes=(64,), activation="relu",
                    solver="adam", max_iter=20, random_state=42)

mlp.fit(X_train[:10000], y_train[:10000])  # Pour gagner du temps

y_pred = mlp.predict(X_test[:2000])
print("Accuracy (MLP sklearn):", accuracy_score(y_test[:2000], y_pred))
print(classification_report(y_test[:2000], y_pred))
```

Réseau de neurones avec Keras Sequential

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
    Dense(64, activation="relu", input_shape=(784,)),
    Dense(32, activation="relu"),
    Dense(10, activation="softmax")
])

model.compile(optimizer="adam", loss="categorical_crossentropy", metrics
    =["accuracy"])
history = model.fit(X_train, y_train_cat, epochs=5, batch_size=128,
                    validation_split=0.1, verbose=2)

# Évaluation
loss, acc = model.evaluate(X_test, y_test_cat, verbose=0)
```

```python
print(f"Accuracy (Keras NN): {acc:.3f}")

# Courbes d'apprentissage
plt.plot(history.history["accuracy"], label="train acc")
plt.plot(history.history["val_accuracy"], label="val acc")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.title("Courbe d'apprentissage - Réseau de neurones (Keras)")
plt.legend()
plt.show()
```

À retenir

Points clés

- Les réseaux de neurones simples permettent déjà des performances élevées.
- **Scikit-learn** facilite la mise en place rapide mais reste limité pour des architectures complexes.
- **Keras** offre plus de contrôle et d'extensibilité.
- L'importance de la normalisation et du suivi des courbes d'apprentissage est cruciale.