Travaux Pratiques: De la conception au déploiement de modèles de Deep Learning

Louis Fippo Fitime, Claude Tinku, Kerolle Sonfack
Département Génie Informatique, ENSPY

22 septembre 2025

Résumé

Ce document de Travaux Pratiques est conçu pour les étudiants en informatique souhaitant acquérir des compétences pratiques en **Deep Learning (DL) Engineering**. Au-delà de la théorie, ce TP se concentre sur les aspects pratiques et d'ingénierie du cycle de vie des modèles DL, incluant le développement, le suivi, l'empaquetage, et le déploiement en environnement de production.

Objectifs Pédagogiques

- Comprendre les concepts fondamentaux de l'apprentissage machine et profond.
- Maîtriser les étapes du cycle de vie d'un modèle de DL.
- Apprendre à utiliser **Git** et **GitHub** pour la collaboration et le versionnement.
- Découvrir et utiliser **MLflow** pour le suivi des expérimentations.
- Savoir empaqueter un modèle DL dans une application web avec **Flask** et le conteneuriser avec **Docker**.

1 Partie 1 : Fondations du Deep Learning (3h)

1.1 Concepts Théoriques

— **Rappel des modèles linéaires et de l'optimisation stochastique.** Expliquez brièvement, en quelques lignes, la différence entre la descente de gradient classique et la descente de gradient stochastique (SGD), et pourquoi la SGD est préférée dans le contexte du deep learning.

— **Compréhension des réseaux de neurones modernes.** Décrivez les rôles des couches d'entrée, cachées et de sortie. Expliquez le processus de rétropropagation du gradient (backpropagation) en termes simples.

1.2 Exercice 1 : Construction d'un réseau de neurones avec Keras

Dans cet exercice, vous allez construire, entraîner et évaluer un réseau de neurones fully-connected (dense) pour la classification des chiffres manuscrits du jeu de données MNIST.

Instructions :

1. Créez un nouveau répertoire de projet et un environnement virtuel Python.
2. Installez les bibliothèques nécessaires : `tensorflow` et `numpy`.
3. Écrivez le code Python suivant dans un fichier nommé `train_model.py`.

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

#Chargement du jeu de donn es MNIST
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.
    load_data()

#Normalisation des donn es
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0

#Redimensionnement des images pour les r seaux fully-connected
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)

#Construction du mod le
model = keras.Sequential([
keras.layers.Dense(512, activation='relu', input_shape=(784,)),
keras.layers.Dropout(0.2),
keras.layers.Dense(10, activation='softmax')
])

#Compilation du mod le
model.compile(
optimizer='adam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy']
```

```
)

#Entrainement du mod le
history = model.fit(
x_train,
y_train,
epochs=5,
batch_size=128,
validation_split=0.1
)

#  valuation   du mod le
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Pr cision sur les donn es de test: {test_acc:.4f}")

#Sauvegarde du mod le
model.save("mnist_model.h5")
print("Mod le sauvegard  sous mnist_model.h5")
```

Listing 1 – Code de formation du modèle MNIST

— **Question 1 :** Expliquez l'utilité des couches Dense et Dropout. Pourquoi la fonction d'activation softmax est-elle utilisée dans la couche de sortie pour ce problème de classification ?

— **Question 2 :** L'optimiseur adam est utilisé. Faites une recherche et expliquez brièvement en quoi il s'agit d'une amélioration par rapport à la SGD simple.

— **Question 3 :** Comment les concepts de "vectorisation" et de "calculs par lots" sont-ils appliqués dans le code ci-dessus ?

2 Partie 2 : Ingénierie du Deep Learning (5h)

2.1 Exercice 2 : Versionnement et collaboration avec Git, GitHub et Gitlab

Le versionnement est crucial dans le développement de modèles.
Instructions :

1. Créez un nouveau repository public sur GitHub ou Gitlab.
2. Initialisez un dépôt Git dans votre répertoire de projet local.

```
git init
git add .
git commit -m "Initial commit of the project"
```

3. Connectez votre dépôt local à votre dépôt GitHub ou Gitlab distant et poussez vos fichiers.

```
git remote add origin <URL_de_votre_d p t>
git push -u origin master
```

4. Créez un fichier `README.md` et décrivez votre projet. Ajoutez-le et poussez les modifications.

2.2 Exercice 3 : Suivi des expérimentations avec MLflow

MLflow est une plateforme de gestion du cycle de vie des modèles d'apprentissage.
Instructions :

1. Installez `mlflow` : `pip install mlflow`.
2. Modifiez votre script `train_model.py` pour y intégrer MLflow.

```
import mlflow
import mlflow.tensorflow
...

Variables pour les param tres
EPOCHS = 5
BATCH_SIZE = 128
DROPOUT_RATE = 0.2

#Lancement de la session de suivi MLflow
with mlflow.start_run():
# Enregistrement des param tres
mlflow.log_param("epochs", EPOCHS)
mlflow.log_param("batch_size", BATCH_SIZE)
mlflow.log_param("dropout_rate", DROPOUT_RATE)

# Construction et entra nement du mod le (utiliser les variables
    d finies)
# ...

```

```
# Enregistrement des m triques
mlflow.log_metric("test_accuracy", test_acc)

# Enregistrement du mod le complet
mlflow.keras.log_model(model, "mnist-model")
```

Listing 2 – Code de formation avec MLflow

2.3 Exercice 4 : Conteneurisation avec Docker et création d'une API

Pour le déploiement, il est essentiel de conteneuriser l'application.

Instructions :

1. Créez un fichier `requirements.txt` contenant les dépendances de votre projet (`tensorflow`, `numpy`, `flask`).
2. Créez un fichier `app.py` pour servir le modèle via une API web.

```
from flask import Flask, request, jsonify
import tensorflow as tf
from tensorflow import keras
import numpy as np

app = Flask(name)

Chargement du mod le Keras
model = keras.models.load_model('mnist_model.h5')

@app.route('/predict', methods=['POST'])
def predict():
data = request.json
# V rification des donn es
if 'image' not in data:
return jsonify({'error': 'No image provided'}), 400

image_data = np.array(data['image'])
# Assurez-vous que l'image est au bon format (1, 784) et normalis e
image_data = image_data.reshape(1, 784)
image_data = image_data.astype("float32") / 255.0

prediction = model.predict(image_data)
predicted_class = np.argmax(prediction, axis=1)[0]

return jsonify({
    'prediction': int(predicted_class),
    'probabilities': prediction.tolist()
})

if name == 'main':
```

```
app.run(host='0.0.0.0', port=5000)
```

Listing 3 – Code de l'API Flask

Instructions (suite) :

3. Créez un `Dockerfile` pour conteneuriser votre application.

```
#Utiliser une image de base Python
FROM python:3.9-slim

#D finir le r pertoire de travail
WORKDIR /app

#Copier le fichier des d pendances et les installer
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

#Copier le reste de l'application
COPY . .

#Exposer le port de l'application Flask
EXPOSE 5000

#Commande pour d marrer l'application
CMD ["python", "app.py"]
```

Listing 4 – Dockerfile

2.4 Exercice 5 : Déploiement et CI/CD

Question 1 : Expliquez comment un pipeline de CI/CD (par exemple, avec GitHub Actions) pourrait automatiser la construction et le déploiement de votre image Docker sur un service comme Google Cloud Run ou Amazon Elastic Container Service.

Question 2 : Une fois le modèle déployé, quels sont les indicateurs clés que vous mettriez en place pour le **monitoring** et le **débogage** en production ? Citez au moins trois types d'indicateurs.

3 Conclusion

À rendre : 25/09/2025

— Le lien vers votre repository GitHub ou Gitlab public.

— Un court rapport (fichier `.pdf` + lien Overleaf du fichier) répondant aux questions posées dans le TP.