Rapport de Travaux Pratiques 1

Conception et Déploiement d'un Modèle de Deep Learning

TOLOKOUM David 21P478
Département Génie Informatique (M2-GI)

Résumé

Ce rapport documente la réalisation du Travail Pratique 1, axé sur la construction et le déploiement d'un modèle de Deep Learning pour la classification des images MNIST. Il couvre les concepts théoriques fondatementaux, l'implémentation d'un Réseau de Neurones Multi-Couches (MLP) avec Keras, le suivi d'expérimentation via MLflow et la conteneurisation de l'API de prédiction à l'aide de Docker. L'accent est mis sur la mise en œuvre des bonnes pratiques d'ingénierie ML (MLOps).

Table des matières

1 Partie 1 : Fondations du Deep Learning

1.1 Concepts Théoriques

1.1.1 Différence entre Descente de Gradient Classique et Stochastique (SGD)

La **Descente de Gradient Classique (Batch GD)** calcule le gradient de la fonction de coût sur **l'intégralité** de l'ensemble de données d'entraînement. Chaque mise à jour des poids nécessite de parcourir toutes les données.

- **Avantage** : Convergence vers un minimum plus stable et direct.
- **Inconvénient** : Extrêmement lent et exigeant en mémoire pour les jeux de données volumineux, comme ceux du Deep Learning.

La **Descente de Gradient Stochastique (SGD)** et, plus couramment, la **Mini-Batch SGD** (utilisée dans le TP), ne calcule le gradient que sur **un seul échantillon** (SGD pure) ou un **petit lot** (*batch*) d'échantillons.

- **Avantage** : **Rapidité** des mises à jour des poids, permettant une convergence beaucoup plus rapide. Le bruit introduit par le mini-batch aide à sortir des minima locaux peu profonds.
- **Justification DL** : La SGD est préférée dans le contexte du Deep Learning car elle permet d'entraîner des modèles sur d'immenses jeux de données en un temps raisonnable et est plus efficace pour explorer l'espace des paramètres.

1.1.2 Rôles des Couches et Rétropropagation

- **Couche d'Entrée (*Input Layer*)** : Reçoit les données brutes (ici, 784 pixels par image MNIST) et les transmet à la première couche cachée. Son rôle est la dimensionnalité d'entrée.
- **Couches Cachées (*Hidden Layers*)** : Effectuent des transformations non linéaires sur les données d'entrée. Elles extraient progressivement des caractéristiques de plus en plus complexes (bords, formes, motifs).
- **Couche de Sortie (*Output Layer*)** : Produit le résultat final de la classification. Le nombre de neurones correspond au nombre de classes (10 pour MNIST).

Le processus de **Rétropropagation du Gradient (*Backpropagation*)** est l'algorithme qui permet l'apprentissage.

1. **Passe Avant (*Forward Pass*)** : Les données traversent le réseau de l'entrée vers la sortie, générant une prédiction et un calcul de la perte (*Loss*).
2. **Passe Arrière (*Backward Pass*)** : Le gradient de la perte est calculé par rapport à chaque poids, de la couche de sortie vers la couche d'entrée (d'où le nom "rétropropagation").
3. **Mise à Jour** : Les gradients sont utilisés par l'optimiseur (Adam) pour ajuster les poids, minimisant ainsi l'erreur du réseau.

   [Image of Backpropagation in a Neural Network]

1.2 Exercice 1 : Construction du Réseau de Neurones

1.2.1 Question 1 : Utilité des Couches et Fonction Softmax

- **Couche Dense (Dense Layer)** : C'est la couche fondamentale d'un MLP, où chaque neurone est connecté à tous les neurones de la couche précédente. Elle effectue la transformation linéaire $Z = W \cdot X + b$. La couche cachée de 512 neurones permet d'apprendre des représentations de haut niveau des données MNIST.
- **Couche Dropout** : C'est une technique de régularisation. Pendant l'entraînement, elle désactive aléatoirement (ici 20% des neurones) à chaque itération.
  - **Utilité** : Cela empêche les neurones de devenir co-dépendants et de sur-apprendre les données d'entraînement (*overfitting*), améliorant la capacité de généralisation du modèle.
- **Fonction d'Activation Softmax** : Elle est utilisée dans la couche de sortie pour les problèmes de **classification multi-classes** (comme MNIST, qui a 10 classes).

— **Utilité** : Elle prend les sorties brutes du réseau (logits) et les transforme en une distribution de probabilités, où la somme de toutes les probabilités est égale à 1. La classe avec la probabilité la plus élevée est la prédiction.

1.2.2 Question 2 : L'optimiseur Adam

L'optimiseur **Adam** (*Adaptive Moment Estimation*) est une amélioration significative par rapport à la SGD simple.

— **Momentum** : Adam intègre le concept de Momentum (similaire à la SGD avec Momentum) qui aide à accélérer la descente dans les directions cohérentes et amortit les oscillations.
— **Taux d'Apprentissage Adaptatif (*Adaptive Learning Rate*)** : Contrairement à la SGD simple qui utilise un taux d'apprentissage unique pour tous les paramètres, Adam maintient un taux d'apprentissage séparé et adaptatif pour **chaque paramètre individuel** du réseau, basé sur une estimation des moments des gradients précédents.
— **Amélioration par rapport à la SGD** : Adam converge généralement **plus rapidement** et nécessite **moins d'ajustements** manuels (comme le taux d'apprentissage) que la SGD simple, ce qui le rend optimal pour la majorité des applications de Deep Learning.

1.2.3 Question 3 : Vectorisation et Calculs par Lots

Ces concepts sont au cœur des performances de TensorFlow/Keras, qui exploitent le calcul parallèle sur GPU.

— **Vectorisation** : Elle est appliquée en transformant les images 28×28 en vecteurs plats de taille 784. Dans les lignes `x_train.reshape(60000, 784)`, les données sont converties en une matrice optimisée pour les opérations matricielles (produits matriciels $\mathbf{W} \cdot \mathbf{X}$) plutôt que des boucles Python lentes, accélérant ainsi le calcul des couches `Dense`.
— **Calculs par Lots (*Batch Computing*)** : Il est appliqué par le paramètre `batch_size=128` lors de l'entraînement (`model.fit`).
  — **Utilité** : Au lieu de traiter les 60 000 images d'entraînement une par une (SGD pure) ou toutes en même temps (Batch GD), le modèle utilise des lots de 128 échantillons pour calculer le gradient moyen et mettre à jour les poids. Ceci optimise l'utilisation des ressources matérielles et est plus efficace que la SGD pure.

2 Partie 2 : Ingénierie du Deep Learning (MLOps)

2.1 Exercice 3 : Suivi des Expérimentations avec MLflow

L'intégration de MLflow (`Listing 2`) permet d'assurer la **traçabilité** et la **reproductibilité** des résultats.

— **Enregistrement des Paramètres** : `mlflow.log_param()` enregistre les hyperparamètres (`EPOCHS=5`, `BATCH_SIZE=128`, etc.) qui ont été utilisés.
— **Enregistrement des Métriques** : `mlflow.log_metric()` enregistre la performance finale (`test_accuracy: 0.9778`).
— **Enregistrement du Modèle** : `mlflow.keras.log_model()` enregistre le modèle complet, permettant de le charger et de le servir en production plus tard (réalisé dans `app.py`).

2.2 Exercice 5 : Déploiement et Monitoring (Questions 1 et 2)

2.2.1 Question 1 : Pipeline CI/CD pour le Déploiement Docker

Un pipeline de CI/CD (par exemple, avec GitHub Actions) automatise le passage du code à la production.

1. **Déclenchement (*Trigger*)** : Le pipeline se déclenche automatiquement lors d'un push sur la branche `main`.
2. **Intégration Continue (CI) - Construction** :

— L'action `checkout` récupère le code du dépôt.
— L'action `docker/build-and-push-action` utilise le `Dockerfile` pour construire l'image Docker `mnist-api:latest`.
— L'image est ensuite poussée vers un registre de conteneurs (ex : Google Container Registry (GCR) ou Docker Hub).

3. **Déploiement Continu (CD) - Service** :
— L'action de déploiement (ex : `google-github-actions/deploy-cloudrun` pour Google Cloud Run) est exécutée.
— Le service Cloud Run est mis à jour en pointant vers la nouvelle version de l'image Docker dans le registre.
— Le nouveau service est rendu disponible, sans intervention manuelle.

2.2.2 Question 2 : Indicateurs Clés de Monitoring en Production

Une fois le modèle déployé via l'API Flask conteneurisée, les indicateurs suivants sont cruciaux pour le monitoring et le débogage :

1. **Indicateurs de Performance du Modèle (*Model Health*)** :
— **Latence de Prédiction** : Temps moyen et percentile P95/P99 pour le traitement d'une requête (doit être faible, ex : < 100 ms).
— **Dérive de Données (*Data Drift*)** : Comparaison de la distribution des données d'entrée reçues en production par rapport aux données d'entraînement. Une dérive peut entraîner une chute de performance silencieuse.
2. **Indicateurs de Stabilité et de Débogage (*System Health*)** :
— **Taux d'Erreur (Code HTTP 5xx)** : Fréquence des échecs de l'API (erreurs serveur, timeouts).
— **Utilisation des Ressources** : Charge CPU et mémoire du conteneur Docker. Un pic indique un besoin de mise à l'échelle ou un problème de fuite mémoire.
3. **Indicateurs d'Affaires (*Business Metrics*)** :
— **Volume de Requêtes** : Nombre de prédictions par minute/heure pour planifier la capacité.
— **Taux de Confiance** : Distribution des probabilités de Softmax. Si la confiance devient trop faible en moyenne, cela signale que le modèle ne comprend plus les entrées.

3 Conclusion

Ce TP a permis d'acquérir une expérience pratique sur la chaîne de valeur du Deep Learning, en allant de la théorie de la SGD et de la rétropropagation, à l'application concrète des outils MLOps (MLflow, Docker) pour le déploiement d'une application de classification en production.