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

Partie 1 : Fondations du Deep Learning (3h)

1.1 Concepts Théoriques

Différence entre descente de gradient classique et stochastique

- Descente de gradient classique (batch gradient descent) :
  Calcule le gradient sur **toutes les données d'entraînement** avant de mettre à jour les paramètres.
  → Avantage : direction de descente précise.
  → Inconvénient : très lent et coûteux pour les grands ensembles de données.
- Descente de gradient stochastique (SGD) :
  Met à jour les paramètres **après chaque exemple** (ou petit lot de données, *mini-batch*).
  → Avantage : plus rapide, nécessite moins de mémoire, permet d'échapper plus facilement aux minima locaux.
  → C'est pourquoi **la SGD est préférée en deep learning**, car les réseaux ont souvent des millions de paramètres et des données massives.

Réseaux de neurones modernes

- Couche d'entrée : reçoit les données brutes (ex. pixels d'une image, valeurs numériques, etc.).
- Couches cachées : effectuent des transformations non linéaires et apprennent des **représentations internes** complexes.
- Couche de sortie : produit la prédiction finale (ex. probabilité d'une classe, valeur numérique, etc.).

Rétropropagation du gradient (Backpropagation)

C'est le processus qui permet **d'apprendre les bons poids** dans le réseau :

1. On fait une **prédiction** avec le réseau (phase *forward*).
2. On calcule **l'erreur** entre la prédiction et la vraie valeur (fonction de coût).
3. On **propage cette erreur en sens inverse** dans le réseau (phase *backward*).
4. À chaque couche, on ajuste les poids grâce au **gradient** de l'erreur par rapport à ces poids (en utilisant la descente de gradient ou la SGD).

Ainsi, le réseau apprend progressivement à **réduire l'erreur** et à améliorer ses prédictions.

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

Couches Dense

- Une couche **Dense (ou fully connected)** relie **chaque neurone** de la couche précédente à **chaque neurone** de la couche suivante.
- Elle permet au réseau d'apprendre des **combinaisons complexes de caractéristiques** extraites des pixels.

Couche Dropout

- La couche **Dropout(0.2)** désactive **aléatoirement 20 %** des neurones à chaque étape d'entraînement.
- Cela évite que le modèle **mémorise** trop bien les données d'entraînement (**overfitting**) et améliore sa **généralisation** sur les nouvelles données.

Fonction d'activation Softmax

- Dans la dernière couche :
- on utilise **softmax** car on a un **problème de classification multiclasse (10 chiffres)**.
- Softmax transforme les valeurs de sortie en **probabilités** comprises entre 0 et 1, dont la somme vaut 1.

Adam (Adaptive Moment Estimation)

- Combine les avantages de deux optimisations :
  - **Momentum** : garde une mémoire des gradients précédents pour accélérer la convergence.
  - **RMSProp** : adapte le taux d'apprentissage **individuellement pour chaque poids**.
- Adam ajuste donc automatiquement la **vitesse d'apprentissage** pour chaque paramètre en fonction de son historique.

Vectorisation

- Au lieu de traiter **chaque image pixel par pixel**, TensorFlow effectue les opérations sur **des matrices complètes** (vecteurs et tenseurs).

→ Toutes les images sont transformées en un seul grand tableau 2D (60000 × 784).
Ainsi, les calculs (produits matriciels, activations, etc.) sont réalisés **simultanément** sur tout un lot grâce à la vectorisation **GPU/CPU**,

Calculs par lots (batch processing)

- Dans l'entraînement :

- → Le réseau traite **128 images à la fois** avant de mettre à jour les poids.
- Cela réduit le **bruit stochastique**, améliore la **stabilité des gradients**, et permet une **meilleure utilisation de la mémoire GPU**.

Partie 2 : Ingénierie du Deep Learning (5h)

2.4 Exercice 5

Un pipeline de CI/CD (Continuous Integration / Continuous Deployment) permet d'automatiser l'ensemble du processus de construction, de test et de déploiement d'une application. Dans le cas d'une API Dockerisée pour un modèle MNIST, un pipeline avec GitHub Actions pourrait être configuré pour se déclencher automatiquement dès qu'une modification est poussée sur le dépôt GitHub. La première étape consiste à **construire l'image Docker** contenant le modèle et l'API, puis à exécuter des tests automatisés pour vérifier que l'application fonctionne correctement. Ensuite, l'image Docker peut être **poussée vers un registre de conteneurs** tel que Docker Hub, Google Container Registry ou Amazon Elastic Container Registry. Enfin, le pipeline peut automatiquement **déployer cette image sur un service cloud**, comme Google Cloud Run ou Amazon ECS, en mettant à jour la version en production sans intervention manuelle. Cette automatisation permet de garantir que chaque modification du code est testée et déployée rapidement et de manière fiable, réduisant ainsi les risques d'erreurs humaines et accélérant la mise en production des nouvelles versions du modèle.

Une fois le modèle déployé, il est important de suivre certains indicateurs pour garantir qu'il fonctionne correctement et détecter rapidement les problèmes :

1. **Performance et disponibilité**
   - **Temps de réponse (latence)** des requêtes à l'API.
   - **Taux d'erreur HTTP** (ex. 500, 404).
   - Objectif : détecter les lenteurs ou crashs du service.
2. **Usage et charge**
   - Nombre de requêtes par minute.
   - Nombre d'utilisateurs simultanés.
   - Objectif : prévoir la scalabilité ou détecter les pics inattendus.
3. **Qualité des prédictions**
   - Taux de prédiction correcte si tu as un **jeu de test en ligne** ou un retour utilisateur.
   - Suivi de **dérive du modèle** : si les données entrantes changent et que les performances chutent.
4. **Logs et erreurs détaillées**
   - Logs d'application pour détecter exceptions, erreurs ou comportements inattendus.

- Utile pour le débogage rapide.