TP - PyTorch & Deep Learning

TP2 - Réseau de Neurones avec PyTorch sur CIFAR-10

Objectifs du TP

Dans ce TP, vous allez implémenter un réseau de neurones en utilisant PyTorch pour classer des images issues du jeu de données CIFAR-10.
Le dataset CIFAR-10 contient 60 000 images de taille 32x32 pixels, réparties en 10 classes. Vous serez amené à construire, entraîner et évaluer un réseau de neurones simple pour résoudre ce problème de classification.

N'oubliez pas de commenter soigneusement votre code afin d'expliquer vos choix et de permettre à vos pairs de comprendre votre raisonnement.

Pour répondre aux question posées dans l'énoncé, écrivez simplement un (ou plusieurs) commentaires dans votre code, au niveau de la partie concernée.

Chargement des Données CIFAR-10

1. Installez le package `torchvision`. Torchvision est un package python comprenant des utilitaires pour le traitement d'images avec pytorch.
2. Utilisez `torchvision.datasets.CIFAR10` pour charger le jeu de données, avec `train=True` pour l'ensemble d'entraînement et `train=False` pour l'ensemble de test.
   Vous devrez convertir les images en tenseurs pour qu'un modèle Pytorch puisse les utiliser. Libre à vous de vous y prendre comme bon vous semble, mais sachez qu'il existe un module spécifique `transforms` intégré à `torchvision` que vous trouverez ici.

Prenez un moment pour appréhender les données avant de passez à la suite. Réfléchissez au type de tâche que vous allez effectuer à partir de ces images, à la fonction de perte que vous allez utiliser, et à la structure de votre réseau.

Préparation des Données

Si chargé depuis torchvision, le jeu de données est déjà séparé en ensembles d'entraînement et de validation.
1. Utilisez des `DataLoader` pour gérer les ensembles d'entraînement et de validation.
2. Assurez-vous que les labels sont bien distribués dans les deux ensembles. Pourquoi voudriez-vous vérifier cela ?

Définir un Réseau de Neurones

1. Créez un réseau de neurones simple. Assurez-vous d'utiliser des fonctions d'activation entre les couches.
2. Imprimez un résumé de votre modèle avec `summary()` pour vérifier sa structure.

Implémenter la Boucle d'Entraînement et de Validation

1. Définissez la fonction de perte. Justifiez votre choix.
2. Définissez l'optimiseur. Ne justifiez pas votre choix ;)
2. Implémentez une boucle d'entraînement.
3. Affichez l'évolution des pertes d'entraînement et de validation au cours des époques. Notez que cela implique de calculer la perte sur le jeu de test. Si vous rencontrez de l'overfitting, essayez de résoudre le souci.

✅ Évaluer le Modèle sur l'Ensemble de Test

1. Utilisez le modèle entraîné pour prédire les labels sur l'ensemble de test.
2. Générez une matrice de confusion pour visualiser les performances du modèle sur chaque classe. Quelle(s) classe(s) pose(nt) le plus de difficulté à votre modèle ?
   Pour cette question vous êtes autorisés, si vous le souhaitez, à utiliser la fonction `confusion_matrix()` de sci-kit learn
3. Affichez quelques exemples d'images mal classifiées par votre modèle. Pourquoi pourraient-elles être difficiles à classer ?

Tensorboard

Vous vous rendrez compte que suivre l'entrainement de votre modèle en affichant les courbes de perte n'est pas pratique: lorsque vous affichez les courbes, votre code s'arrête. Si vous voulez les sauvegarder, il faut le faire manuellement..

Pour faire ce qu'on appelle de l'*experiment tracking*, de nombreuses solutions existent, dont une des plus populaires: Tensorboard.
Allez faire un tour sur la documentation pytorch de Tensorboard et implémentez le suivi de votre entrainement avec Tensorboard. Vous pouvez même sauvegarder des images.

Améliorations du Modèle

1. Ajoutez une régularisation par dropout avec la classe du même nom `Dropout()` dans votre modèle pour limiter un éventuel surapprentissage.
   Testez différentes valeurs pour voir l'influence du dropout sur les performances du modèle.
2. Expérimentez avec différentes architectures (par exemple, ajouter plus de couches ou modifier le nombre de neurones) et comparez les résultats avec la version initiale du modèle. A l'aide d'un commentaire, décrivez ce que vous avez testé et pourquoi.
3. Normalisez vos images: pour un meilleur apprentissage, vous voulez que vos entrées soient comprises dans la région $[-1, 1]$, avec une moyenne à 0.

Utilisation de l'Augmentation de Données

1. Ajoutez une augmentation de données à votre pipeline d'entraînement (par exemple, par rotation, translation, zoom, etc.).
   Vous pourrez utiliser par exemple la classe `transforms.Compose` de torchvision, de nombreuses classes sont directement utilisables avec la liste complète ici
2. Entraînez le modèle avec et sans augmentation de données et comparez les courbes d'apprentissage.