Introduction à PyTorch - TP 3- LIGHTNING

Université de Caen Normandie

Juin 2023

1 Introduction

L'objectif de ce TP est de revoir et approfondir les notions vues avec la librairie PyTorch Lightning [1] en réalisant un réseau de neurones capable de classifier des chiffres manuscrits.

Ce TP repose sur les TPs précédents, référez-vous à ces TP pour avoir un détail plus précis des étapes à réaliser.

Les étapes de ce TP sont les suivantes :

1. Préparer les données et les batches en créant la classe 'MyData' qui hérite de 'pl.LightningDataModule'.
2. Construire un modèle de réseau de neurones et les méthodes nécessaires pour l'entraîner en créant la classe 'MyModel' qui hérite de 'pl.LightningModule'.
3. Entraîner le modèle à l'aide des données préparées dans l'étape 1 et du classifieur de l'étape 2 en utilisant la console.
4. Faire le déploiement de l'application sur un serveur Web.

L'objectif est de vous mettre dans les conditions d'un développement réel d'application. Il vous est demandé de passer par la définition de fichier python et non de notebook.

2 Première étape : définition d'un "Data Module"

Nous allons commencer par travailler sur la préparation des données en utilisant le module `datamodule` [2] de `lightning`. Définissez dans un fichier my_dataset.py, la classe MyData permettant la gestion de la base de donnée.

Vous générerez des batches de 64 images et gardez 10% des images d'entrainement comme données de validation.

Vous définirez en particulier les méthodes :

— `__init__` : définition de constantes comme le chemin d'accès des données, la taille des batches ou le pourcentage d'images d'entrainement gardées comme données de validation. Ces constantes doivent être passées comme argument de l'init.

— `prepare_data` : préparation des données. Dans notre cas, décompression des archives par la fonction `torchvision.datasets.MNIST` avec l'attribut `download` à `True`.

— `setup` : définition des datasets. La méthode `torchvision.datasets.MNIST` va vous permettre de générer des datasets en leur appliquant des transformations. Nous vous demandons d'appliquer des transformations aléatoires pour augmenter la taille de la base avec

1. `https://lightning.ai/`
2. `https://lightning.ai/docs/pytorch/stable/data/datamodule.html`

`torchvision.transforms.RandAugment`.
Regardez dans la documentation le rôle de l'argument 'stage' et utilisez-le à bon escient. Cette variable pourra prendre les valeurs 'None', 'test' ou 'fit'.

- `train_dataloader` : utilise la méthode `DataLoader()` pour renvoyer un data loader sur les données de train.
- `val_dataloader` : utilise la méthode `DataLoader()` pour renvoyer un data loader sur les données de validation.
- `test_dataloader` : utilise la méthode `DataLoader()` pour renvoyer un data loader sur les données de test.
- `predict_dataloader` : utilise la méthode `DataLoader()` pour renvoyer un data loader sur les données de test.

3 Seconde étape : construction du réseau de classification

Dans un nouveau fichier, définissez la classe `MyModel` qui hérite de `lightning.LightningModule`, qui permet de construire le modèle de réseau de neurones et les différentes fonctions pour l'entrainer.

Le réseau de classification que nous vous demandons d'utiliser est un réseau de type 'resnet18' que vous utiliserez pré-entrainé pour une tâche de classification de type ImageNet. Contrairement au précédant TP, nous utiliserons ici la version de la librairie Timm.

Récupérer le réseau et ses poids peut se faire via :
`timm.create_model('resnet18', pretrained=True, num_classes=10, in_chans=1)`.

Vous devrez en particulier définir les méthodes :

- `__init__` : définition des différentes couches du réseau et des constantes.
- `forward` : calcule la sortie du réseau en fonction de son entrée.
- `training_step` : retourne la loss correspondant à un batch donné.
- `validation_step` : effectue le traitement d'un batch sur les données de validation.
- `test_step` : effectue le traitement d'un batch sur les données de test.
- `configure_optimizers` : retourne l'optimiseur choisi pour entrainer le réseau. Vous utiliserez un optimiser de type Adam.
- `example_input_array` : retourne un tableau torch de zéros de taille 1,1,28,28 . Cette méthode est une propriété de classe (`@property`).

4 Troisième étape : entrainement du réseau de neurones

Une fois les classes précédentes définies, vous êtes capable d'entrainer et d'évaluer le modèle. Définissez un fichier `main.py` permettant de créer un objet `LightningCLI`. Lancez l'apprentissage sur 3 époques en configurant correctement un fichier yaml.

5 Quatrième étape : analyse de l'apprentissage avec Tensorboard

Vous pouvez lancer l'outil Tensorboard pour analyser les résultats de votre réseau à l 'aide de la commande :

```
tensorboard --logdir=lightning_logs
```

Visualiser les différentes courbes d'apprentissage dans votre navigateur en vous connectant au serveur lancé par tensorboard.

Modifiez le fichier yaml de configuration de votre Trainer avec le champ :

```
logger:
  - class_path: lightning.pytorch.loggers.tensorboard.TensorBoardLogger
    init_args:
        save_dir: '.'
        name: 'lightning_logs'
        log_graph: true
```

Visualisez le graphe dans tensorboard.

6 Dernière étape : déploiement du réseau sous la forme d'un serveur web Gradio

Complétez le fichier app_deploy_gradio.py puis lancer depuis le terminal la commande : `lightning run app app_deploy_gradio.py`.

Créez une image png d'un chiffre manuscrit et vérifier que la classification est correcte. Attention le fond doit être en noir et le chiffre en blanc.