TP Deep Learning

1 Installation

1.1 Ouverture d'une session (extérieurs)

Un compte spécial a été créé pour les participants extérieurs à l'INSA. Pour ouvrir une session Windows, utilisez les identifiants suivants :
Login : `master2`
Mot de passe : `6qFT5h`
Une fois connecté, créez un dossier à votre nom sur le disque local, dans `C:\Users\master2`. Vous travaillerez uniquement dans ce dossier.

1.2 Packages Python

Lancez Anaconda Prompt et installez les packages `opencv`, `pytorch` et `torchvision` avec les commandes suivantes :

```
conda install opencv -c anaconda
conda install pytorch-cpu -c pytorch
pip install torchvision
```

Par la suite, vous pourrez exécuter les programmes Python à partir de cette fenêtre de commande, ou à l'aide de l'environnement de développement Spyder (nous recommandons la 2ème solution, car les affichages à l'aide du package `matplotlib` semblent ne pas fonctionner lors d'une exécution en ligne de commande)

1.3 Utilisation du serveur GPU

L'INSA dispose d'un serveur de calcul GPU (`SRV-ACAD`), dont l'adresse IP est 192.168.53.88. En plus de votre compte pour ouvrir sous WIndows, vous disposez d'un compte sur ce serveur (sauf exception, le login est composé de la première lettre de votre prénom suivi de votre nom de famille. Votre mot de passe est identique à votre login).

Lors de ce TP, vous ouvrirez des sessions SSH (connexion à distance) sur votre compte sur un serveur GPU. Vous allez pour cela utiliser le client SSH MobaXTerm. Téléchargez-le à partir de `https://mobaxterm.mobatek.net` (choisissez la version *MobaXterm Home Edition v11.0 (Portable edition)* en .zip). Décompressez l'archive dans votre dossier local.

Afin de tester le transfert de fichier de votre compte Windows local vers votre compte serveur, créez un fichier Python bidon dans votre dossier de travail (en utilisant Spyder ou Notepad++). Par exemple :

```
print('Ce programme est bidon')
```

Lancez MobaXTerm. Appuyez sur "Start Local Terminal". Ouvrez ensuite une session SSH sur SRV-ACAD en entrant la commande

```
ssh votre_login_serveur@192.168.53.88
```

Entrez votre mot de passe serveur. Vous êtes maintenant connecté sur votre compte distant. Désormais, toutes les commandes entrées sont exécutées sur SRV-ACAD. Le contenu de votre dossier personnel doit apparaître dans la fenêtre de gauche. Un bouton permet d'uploader un fichier local sur votre compte serveur. Uploadez votre source Python et exécutez-le :

`python3 votre_programme_bidon.py`

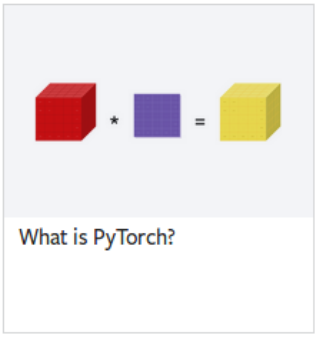
Remarque : tous les packages Python nécessaires sont déjà installés sur SRV-ACAD.

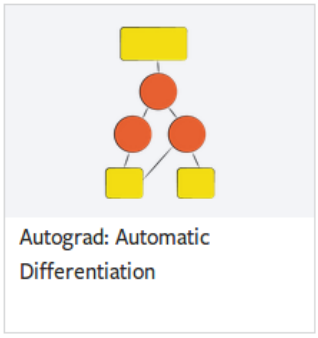
Vous pouvez modifier un fichier sur le serveur en l'ouvrant à partir de la fenêtre gauche de MobaXterm (le plus simple), ou en ligne de commande avec `emacs`. Auquel cas, vous pouvez consulter son manuel sur
`http://www.linux-france.org/article/appli/emacs/tut/emacs-tut.html`
`https://doc.ubuntu-fr.org/emacs`

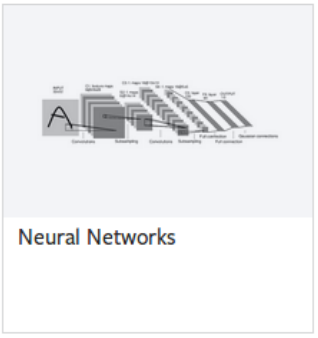
Pour fermer la session SSH, entrez la commande `exit`

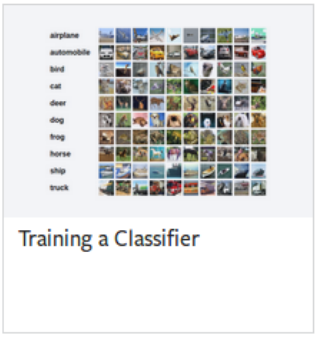
2 Tutoriaux

Allez sur `https://pytorch.org/tutorials/`. Allez dans *Getting Started* > *Deep Learning with PyTorch : A 60 Minute Blitz*. Vous allez suivre les 4 premiers tutoriels :


What is PyTorch?


Autograd: Automatic Differentiation


Neural Networks


Training a Classifier

Ils vont vous permettre de vous familiariser avec les éléments de base de PyTorch (comme les `Tensors`, les `Modules`, la différentiation automatique, les fonctions de *loss* et les algorithmes d'optimisation).

Attention, la version Windows de PyTorch ne supporte pas encore le *multithread*. Dans le code du tutoriel n°4, aux lignes 73 et 78, remplacez `num_workers=2` par `num_workers=0`. A la première exécution, le tutoriel 4 télécharge la base CIFAR10 (`https://www.cs.toronto.edu/ kriz/cifar.html`), ce qui nécessite un certain temps.

Dans un premier temps, exécutez le code des tutoriels en local.

Dans un second temps, exécutez le tutoriel n°4 sur le serveur ACAD. Copiez le source et l'archive de la base CIFAR10 `cifar-10-python.tar.gz` sur votre compte serveur, ce qui évitera un nouveau téléchargement (l'arborescence doit être la même que sur la version locale : placez l'archive dans un dossier `data`, lui-même à côté du `.py`). Pour que le code puisse s'exécuter sur le serveur et tirer parti du GPU, vous devez :

- désactiver toutes les instructions relatives à l'affichage : commentez les appels aux fonctions de `matplotlib` (`show`, `imshow`, ...).
- indiquer à PyTorch d'utiliser CUDA : les tenseurs et les couches du réseau seront stockés dans la mémoire GPU. Les calculs seront parallélisés au maximum. Pour ce faire, utilisez la fonction `cuda()` lors de l'initialisation du réseau,
  `net = Net().cuda()`
  et lorsque les images/labels sont chargés,
  `inputs, labels = data[0].cuda(), data[1].cuda()`

Si le GPU est effectivement utilisé, vous devez observer un gain de vitesse assez net par rapport à la version CPU.

Pendant ces tutoriels, les documentations des bibliothèques sont à consulter sans modération :

- `https://pytorch.org/docs`
- `https://docs.scipy.org/doc`

— https://matplotlib.org/contents.html

En particulier, vous devez apprendre à utiliser les classes représentant les briques de base pour construire un réseau, dérivées de `torch.nn.Module`,

— `torch.nn.Linear` : couche dense (ou *fully-connected*),
— `torch.nn.Conv2d` : couche convolutive,
— `torch.nn.MaxPool2d` : couche de *max-pooling*,
— `torch.nn.ReLu`, `torch.nn.Tanh`, `torch.nn.Sigmoid` : couches d'activations non-linéaires,

les classes représentant les fonctions de *loss*, dérivées de `torch.nn._Loss`,

— `torch.nn.L1Loss`,
— `torch.nn.CrossEntropyLoss`,
— `torch.nn.NLLLoss`,

et les classes représentant les algorithmes d'optimisation, dérivées de `torch.optim.Optimizer`, notamment `torch.optim.SGD`.

3 Classification sur ImageNet

Le but de l'exercice est de réaliser une classification d'images provenant de la base ImageNET (`http://www.image-net.org/`), en vous inspirant de l'architecture du tutoriel n°4. Vous allez travailler avec un sous-ensemble, appelé tiny-imageNet, téléchargeable à partir de Celene INSA ou à l'adresse suivante :

`https://julien-mille.github.io/public_html/tiny_imagenet_20classes.zip`

La base comporte 20 classes. Chaque classe contient 500 images RGB 64×64. Les fichiers source fournis contiennent 2 classes dérivées de `torch.utils.data.Dataset`, qui permettent de parcourir les ensembles d'apprentissage et de test. Pour chaque classe, les 450 premiers exemples sont utilisés pour l'apprentissage, les 50 restants pour les tests.

En vous inspirant du tutoriel n°4 sur CIFAR10, vous allez construire un réseau composé de plusieurs "séquences convolutives" suivis de plusieurs couches denses (ou *fully-connected*). On rappelle qu'une séquence convolutive est composée d'une couche de convolution, d'une fonction d'activation non-linéaire et d'un *max-pooling*.

Vous travaillerez toujours en local dans un premier temps. Après avoir vérifié que le code s'exécute correctement, exécutez-le sur le serveur (en activant CUDA)

3.1 Exercice 1

Tout d'abord, essayez avec 2 séquences convolutives (avec des activations ReLU) et de 2 couches denses. Testez avec des tailles de filtres (paramètre `kernel_size`) relativement petites, dans les couches de convolution (3×3, 5×5, 7×7). Choisissez le nombres de *feature maps* par couche (paramètres `in_channels` et `out_channels`), et le nombre de neurones dans la 1ère couche dense.

Utilisez la *cross entropy* comme *loss*, et la descente de gradient stochastique (SGD) pour l'optimisation. Affichez la fonction de coût au fur et à mesure des *epochs*. Sur l'ensemble de test, affichez la valeur de la fonction de coût et générez la matrice de confusion.

3.2 Exercice 2

Relancez l'apprentissage et le test, après avoir intercalé 1 ou 2 séquence(s) convolutive(s) supplémentaire(s), le but étant d'améliorer les résultats obtenus précédemment. Répétez cette opération tant que l'expérience est concluante !

3.3 Exercice 3

Revenez à une architecture à 2 séquences convolutives et 2 couches denses. Vous allez automatiser la recherche des meilleurs hyperparamètres, en utilisant un ensemble de validation.

Modifiez la gestion du dataset de manière à avoir 3 ensembles : dans chaque classe, les 400 premiers exemples seront utilisés pour l'apprentissage, les 50 suivants pour la **validation** et les 50 derniers pour le test. Pour cela, vous devez comprendre le fonctionnement de `__getitem__` dans les classes `TinyImageNetDatasetTrain` et `TinyImageNetDatasetTest`. Modifiez ces classes et ajoutez une classe `TinyImageNetDatasetValidation`.

Choisissez l'ensemble des hyperparamètres (taille des filtres des couches convolutives, nombre de *feature maps* des couches convolutives, nombre neurones dans la 1ère couche dense) et les intervalles dans lesquels vous allez les faire varier. Pour chaque configuration, entrainez le réseau sur l'ensemble d'apprentissage, testez-le sur l'ensemble de validation, et retenez la configuration donnant le meilleur résultat. La configuration donnant le meilleur taux de reconnaissance sur l'ensemble de validation est celle qui sera conservée en phase de test.