Deep Learning - Practical course

1 Install

1.1 Opening a session (for non-INSA participants)

A dedicated account has been created for participants outside INSA. To open a Windows session, use the following login details :
Login : `master2`
Password : `6qFT5h`
Once you are logged in, create a folder with your name on the local disk, in `C:\Users\master2`. From now on, you work in this folder only.

1.2 Python packages

Launch Anaconda Prompt and install packages `opencv`, `pytorch` et `torchvision`, using the following command lines :

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

During this course, Python scripts will be executed from the current command line window, or using the Spyder development environment (we recommend the last solution, as graphical displays with the `matplotlib` package do not seem to work properly when running from the command-line window)

1.3 Using the GPU server

INSA is equipped with a GPU server, called `SRV-ACAD`, located at IP address 192.168.53.88. In addition to your Windows account, you have an account on this server (with some exceptions, your login is made up of the first letter of your firstname followed by your surname. Your password is identical to your login).
During this course, you will open SSH sessions - remote connections - on your GPU server account. To do so, you will use a SSH client called MobaXTerm. Download it from `https://mobaxterm.mobatek.net` (choose version *MobaXterm Home Edition v11.0 (Portable edition)* in .zip format). Unzip it in your local working folder.

In order to test file transfer, from your local Windows account to your remote server account, create a dummy Python script in your local folder (using Spyder or Notepad++). For example :

```
print('This is a dummy program')
```

Lauch MobaXTerm. Push "Start Local Terminal". Then, open a SSH session on SRV-ACAD using the following command :

```
ssh your_server_login@192.168.53.88
```

Enter your server account password. You are now logged on your server account. From now on, every command entered will be executed on SRV-ACAD. The content of your account should appear in the left window. There is a button to upload a local file to your remote account. Upload you Python script, and execute it using the following command :

```
python3 your_dummy_Python_code.py
```

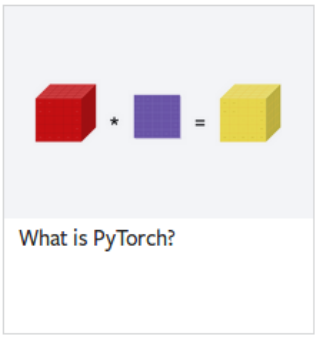
Note that all required Python packages are already installed on SRV-ACAD.

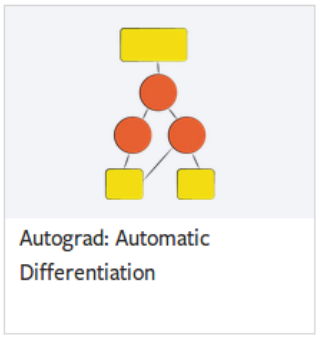
Files on your remote account can be edited by opening them from the left window of MobaXterm (this is the easiest way), or in command-line using `emacs`. In the last case, use manuals available at
`http://www.linux-france.org/article/appli/emacs/tut/emacs-tut.html`
`https://doc.ubuntu-fr.org/emacs`

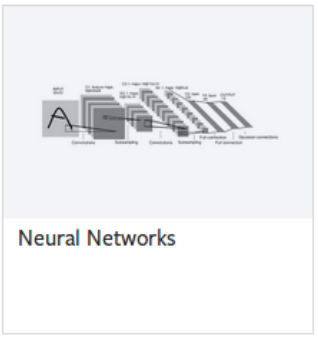
To close the SSH session, enter the `exit` command.

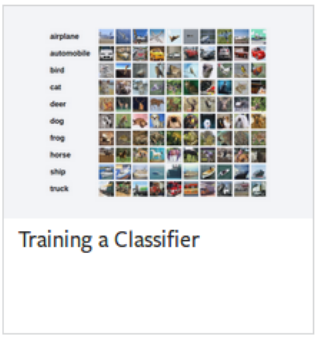
2 Tutorials

Go to `https://pytorch.org/tutorials/`, in *Getting Started* > *Deep Learning with PyTorch : A 60 Minute Blitz*. Follow the 4 first tutorials :


What is PyTorch?


Autograd: Automatic Differentiation


Neural Networks


Training a Classifier

Using these tutorials, you will get familiar with the basic building blocks of PyTorch (like `Tensor`s, `Module`s, automatic differentiation, loss functions and optimisation algorithms).

Important : *multithreading* support is not fully implemented in the Windows version of PyTorch. The code of tutorial 4 will crash if you use it 'as is'. At lines 73 and 78, replace `num_workers=2` by `num_workers=0` to make it work. When tutorial 4 is run for the first time, it downloads the CIFAR10 dataset (`https://www.cs.toronto.edu/ kriz/cifar.html`), which may take some time.

First, run the tutorials on the local machine.

Then, run tutorial 4 on server SRV-ACAD. Copy the Python source as well as the CIFAR10 dataset, `cifar-10-python.tar.gz`, to your remote account, which will prevent Python to download the dataset again (the file tree should be the same as on the local machine : put the archive in a `data` folder beside the `.py` file). In order the code to run and take advantage of the GPU, you should :

- disable all instructions related to display : comment every call to functions of `matplotlib` (`show`, `imshow`, ...).
- tell PyTorch to use CUDA : tensors and network layers should be stored on the GPU memory, so that computations are made in parallel as soon as possible. To do so, use the `cuda()` function when initializing the network,
  `net = Net().cuda()`
  and when loading input images and labels,
  `inputs, labels = data[0].cuda(), data[1].cuda()`

If the GPU is actually used, you should get a significant speed gain, in comparison to the CPU version.

While following the tutorials, do not hesitate to have a look at manuals for the libraries used :

- `https://pytorch.org/docs`
- `https://docs.scipy.org/doc`
- `https://matplotlib.org/contents.html`

In particular, you should learn how to use classes representing basic building blocks of network, which inherit from `torch.nn.Module`,

— `torch.nn.Linear` : dense (= *fully-connected*) layer,
— `torch.nn.Conv2d` : convolutional layer,
— `torch.nn.MaxPool2d` : *max-pooling* layer,
— `torch.nn.ReLu`, `torch.nn.Tanh`, `torch.nn.Sigmoid` : non-linear activation layer,

classes implementing loss functions, inheriting from `torch.nn._Loss`,

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

and classes implementing optimization algorithms, inheriting from `torch.optim.Optimizer`, in particular `torch.optim.SGD`.

3 Classification on ImageNet

The purpose of the next exercises is to classify images from the ImageNET dataset (`http://www.image-net.org/`), drawing your inspiration froom the architecture built in tutorial 4. You will work with a subset of ImageNET, called tiny-imageNet, that can be downloaded from Celene INSA or from

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

The subset is made up of 20 classes. Each class contains 500 RGB images of size 64×64. Provided source files contain 2 classes derived from `torch.utils.data.Dataset`, which can be used to iterate over the training and test sets. For each class, the first 450 samples are used for training, the remaining 50 ones for testing.

As in tutorial 4, you should build a network made up of several "convolutional sequences" followed by several dense layers. Let us recall that a "convolutional sequence" is composed of a convolutional layer, a non-linear activation layer and a max-pooling layer.

Always work on the local machine first. Once you have checked that the code runs properly, run it on the server (remember to activate CUDA).

3.1 Exercise 1

First, try with 2 convolutional sequences (with ReLU activation) and 2 dense layers. Test with relatively small filter sizes (parameter `kernel_size`) in convolutional layers (3×3, 5×5, 7×7). Choose the number of feature maps per layer (parameter `in_channels` and `out_channels`), as well as the number of neurons in the first dense layer.

Use the cross entropy as loss function, and stochastic gradient descent (SGD) for optimization. Print the cost function as *epochs* are iterated. On the test set, print the cost function and generate the confusion matrix.

3.2 Exercise 2

Launch training and testing again, after having inserted 1 or 2 additional convolutional sequence(s), with the purpose of improving over previous results. Repeat this operation while classification gets better !

3.3 Exercise 3

Switch back to an architecture with 2 convolutional sequences and 2 dense layers. The purpose of this exercise is to automate the search of hyperparameters, using a validation set.

Modify how datasets are handled in order to get 3 sets : in each class, the first 400 samples should be used for training, the next 50 ones for validation and the last 50 ones for testing. To do so, you should understand how the `__getitem__` member function operates, in classes `TinyImageNetDatasetTrain` and `TinyImageNetDatasetTest`. Modify these classes and add a class `TinyImageNetDatasetValidation`.

Choose the set of hyperparameters (sizes of filters and number of feature maps in convolution layers, number of units in the first dense layer) and ranges within which you make them vary. For each configuration, train the network on the training set, test it on the validation set, and keep the configuration leading to the best result. This configuration of hyperparameters will be the one kept to compute the final recognition rate on the test set.