Deep Learning Course: Lab 1

Alex Stergiou, Esther Rodrigo, Fawaz Sammani

3rd October 2023

1 Environment

Throughout these lab sessions, we will be working with **Python**. Python is a scripting language which is widely used nowadays. In particular, Python is among the most popular programming languages for data science and machine learning. This tutorial gives you an introduction on how to setup your own environments and a brief exercise session on the libraries and elements we will use. If you are familiar and have Python 3, Jupyter Notebook, Google Colab (or any similar to code in Python) already installed in your laptop, you can jump to Section 1.2.

1.1 General set up

We **highy recommend** that you work online with **Google Colab**. However, you also have the choice to setup everything at your own personal computer and work with Ana/Miniconda.

1) Google Colab: Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. Specifically for this course, it will simplify things that you keep your code on the could and you are easily able to share such code with us. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs [6]. You can access Google Colab via the following url `https://colab.research.google.com/`

2) Anaconda/Miniconda: However, you also have the choice to setup everything at your own personal computer. For that you should install either Anaconda or Miniconda (a minimal version of Anaconda with just the basic packages installed that saves you some disk space) [4]. Anaconda is one of the most popular Python data science platforms. It is packed with several common data science packages, including the ones listed in 1.2. Note that these packages except for Python are not included in Miniconda and you should manually install them using the pip command. We will go through that in the next steps. Anaconda also allows us to create, maintain virtual environments, and easy installations of additional data science packages in case it is needed.

Below is a step-by-step guide to set up your environment. If you decide to work with Google Colab (as we recommend), you can skip this and jump to section 1.2.

In the steps below you can find how to install Miniconda and the packages that we are going to use on Linux. Note that although the instructions are for Linux, it is straightforward to install Miniconda on Windows or Mac OS, and then follow the steps from step 2 onwards. If you can import these packages, everything is working properly with your installation.

1. Install Miniconda: Download and install Miniconda from the link `https://docs.conda.io/en/latest/miniconda.html`. Don't forget to choose the right version for your OS. You can download the file with Python 3.9 pre-installed. When you download your file open a linux terminal and run the following commands:

```
chmod +x your_file.sh
./your_file.sh
```

2. Then create a new python environment with Python 3.9 (or any other version of python 3 that you prefer). To do so run:

```
conda create -n python3.9 python=3.9
```

3. Activate your newly installed environment:

```
conda activate python3.9
```

4. Install Numpy by typing:

```
pip install numpy
```

5. Install scikit-learn by typing:

```
pip install sklearn
```

6. Install matplotlib by typing:

```
pip install matplotlib
```

7. Install scikit-learn by typing:

```
pip install notebook
```

After executing the previous steps without any errors, you should have a working Python 3.11 environment. You can validate that your installation is working properly using the following steps.

1. Open your terminal and activate your Python 3.11 environment (execute this step in the case that you are not already inside the Python 3.11 environment):

```
conda activate python3.9
```

2. Invoke Python:

```
python
```

3. Import Numpy:

```
import numpy
```

4. Import matplotlib :

```
import matplotlib
```

5. Import scikit-learn:

```
import sklearn
```

6. Close the previous terminal or exit() from the python terminal. Open you jupyter notebook and navigate with you browser into the .ipynb file that you want to execute:

```
jupyter notebook
```

1.2 Required libraries/packages

During the exercise sessions, the following packages/libraries/elements will be required in your working environment:

- Python: throughout the lab sessions, we will use Python 3.11 (or the latest version available).
- Numpy [1]: a fundamental package for scientific computing with Python. During the first three lab sessions, we will code using the numpy and pytorch libraries. From lab 4, we will switch to only Pytorch. Numpy allows for
  - a powerful N-dimensional array object
  - sophisticated (broadcasting) functions
  - tools for integrating C/C++ and Fortran code
  - useful linear algebra, Fourier transform, and random number capabilities

  In general, Numpy allows us to perform efficient operations with matrices in an easy way.
- scikit-learn [2]: a simple and efficient tool for data mining and data analysis. It contains efficient implementations of many common machine learning models.
- Matplotlib [3]: a Python 2D plotting library. We will need it to draw figures, graphs, etc. from a Python program
- Jupyter Notebook [5]: a handy web application that allows us to write and execute Python code in the web browsers. We will mostly work with it through the exercise sessions.

1.3 Libraries for Neural Networks

Nowadays, neural network models, especially deep models, have been dominating a majority of machine learning tasks, from face detection and object retrieval to speech recognition and language understanding. There is a number of libraries for deep learning, released recently by big tech companies and laboratories, such as:

- **Tensorflow** from Google
- **Torch and PyTorch** from Facebook
- **CNTK** from Microsoft
- **Caffe** from UC Berkeley
- **Theano** from University of Montreal

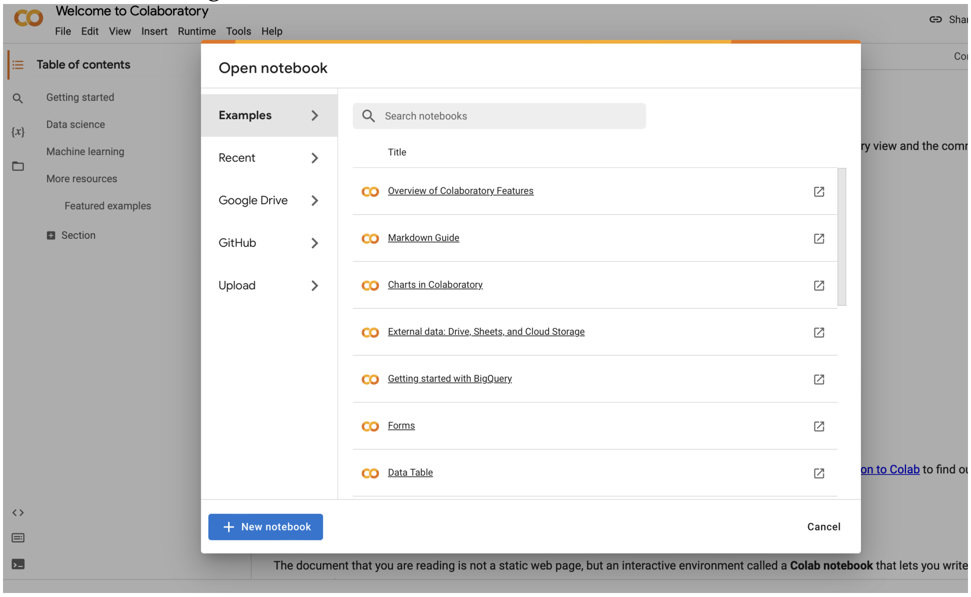
In the first three lab sessions, we will implement simple machine learning architectures with toy applications using two libraries, **Numpy and Pytorch**. However, implementing an efficient neural network model often requires a lot of effort, and implemeting from stratch (like in numpy) becomes an impossible task. For that, in real projects, it is common and highly recommended to make use of existing libraries. Pytorch seems to be the most popular one at the moment. Thus, we will use Pytorch for the exercises of the Deep Learning course (and numpy only on the first two lab sessions so you can compare). First, make sure you have finished installing Pytorch as specified below.

2 Setting up Pytorch

Select among 1) and 2) based on your decision in Section 1.1.

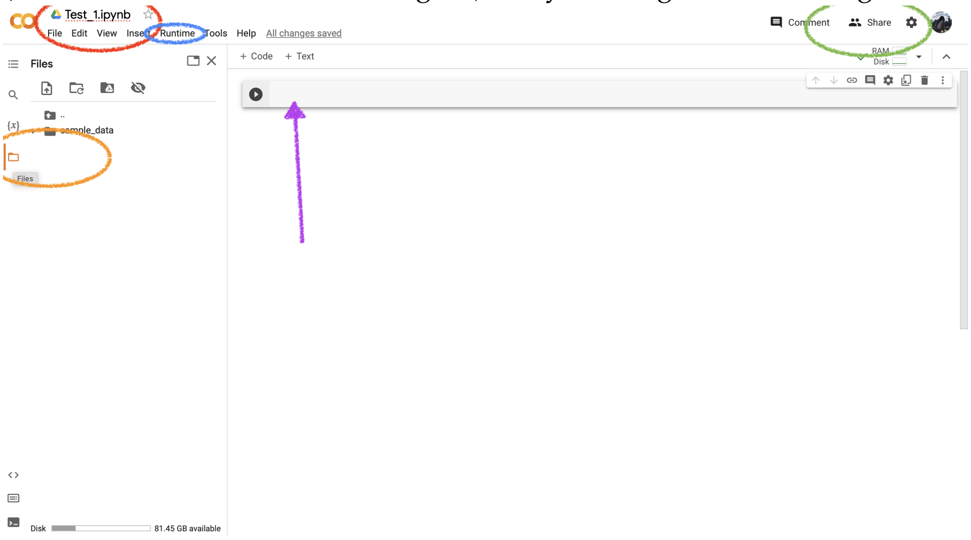
1) Google Colab: It is recommended that you work online with Google Colab. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. **Pytorch is directly available through Google Colab.** You can access Google Colab via the following url `https://colab.research.google.com/`.

Figure 1: Using the access link provided above for Google Colab, you will see the following



The document that you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write

Figure 2: Create a new Notebook by pressing on the *New notebook* button (blue bottom left button on the figure) and you will get the following screen.



If you have worked with Jupyter Notebook, you will straightforwardly understand the idea: you can write text and run code in the same page.

In such screen you can

- define the notebook title (red circle), upload folders and files (orange circle), share the notebook (green circle) and much more.
- For resource intensive tasks (we will let you know when it is required), you can use a GPU by selecting Runtime (blue circle) → Change runtime type → Hardware accelerator → GPU.
- In File (look left on the same row as blue circle), you can create a new notebook or upload existing ones on your computer (most of the lab sessions we will provide an skeleton code so you will need to upload it here)
- Finally, code and text are written in cells (purple arrow).

2) Anaconda/Miniconda: However, you can install it in Anaconda/Miniconda. If you decide to work with Google Colab (as we recommend), you can skip this and jump to Section 3.

Activate your python environment and then run the following commands:

conda install pytorch torchvision torchaudio cpuonly -c pytorch

Note that this command will install the CPU version of `pytorch`. If you have a Nvidia graphic card, with `CUDA` installed, it is recommended to install the GPU version of `pytorch` instead, to utilize the computing power of your graphic's card. For a more detailed installation guideline, you can check the following link: `https://pytorch.org/get-started/locally/`.

3 Working with Pytorch

If you are not familiar with Pytorch, we **highly** recommend you to go through the following tutorial from Pytorch webpage. **During the following sections, we ask you to think what the outputs of the print() functions will be before running them.**

3.1 Tensors

Tensors are a specialized data structure very similar to arrays and matrices. In PyTorch, we use tensors to encode the inputs and outputs of a model, as well as the model's parameters.

Tensors are similar to NumPy's ndarrays, except that tensors can run on GPUs or other hardware accelerators. In fact, tensors and NumPy arrays can often share the same underlying memory, eliminating the need to copy data (see Bridge with NumPy). Tensors are also optimized for automatic differentiation (we'll see more about that later in the Autograd section). If you're familiar with ndarrays, you'll be soon comfortable with the Tensor API.

```
import torch
import numpy as np
```

3.1.1 Initializing a Tensor

Tensors can be initialized in various ways. Take a look at the following examples:

Directly from data. Tensors can be created directly from data. The data type is automatically inferred.

```
data = [[1, 2],[3, 4]]
x_data = torch.tensor(data)
```

From a NumPy array. Tensors can be created from NumPy arrays (and vice versa - see Bridge with NumPy).

```
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
```

From another tensor. The new tensor retains the properties (shape, datatype) of the argument tensor, unless explicitly overridden.

```
x_ones = torch.ones_like(x_data)
# retains the properties of x_data
print(f"Ones Tensor: \n {x_ones} \n")

x_rand = torch.rand_like(x_data, dtype=torch.float)
# overrides the datatype of x_data
print(f"Random Tensor: \n {x_rand} \n")
```

With random or constant values: shape is a tuple of tensor dimensions. In the functions below, it determines the dimensionality of the output tensor.

```
shape = (2,3,)
rand_tensor = torch.rand(shape)
ones_tensor = torch.ones(shape)
zeros_tensor = torch.zeros(shape)

print(f"Random Tensor: \n {rand_tensor} \n")
print(f"Ones Tensor: \n {ones_tensor} \n")
print(f"Zeros Tensor: \n {zeros_tensor}")
```

3.1.2 Operations and functions with tensors

torch.reshape Returns a tensor with the same data and number of elements as input, but with the specified shape. When possible, the returned tensor will be a view of input. Otherwise, it will be a copy. Contiguous inputs and inputs with compatible strides can be reshaped without copying, but you should not depend on the copying vs. viewing behavior.

A single dimension may be -1, in which case it's inferred from the remaining dimensions and the number of elements in input.

Example:

```
a = torch.arange(4.)
print(torch.reshape(a, (2, 2)))
    #-> tensor([[0., 1.], [2., 3.]])
b = torch.tensor([[0, 1], [2, 3]])
print(torch.reshape(b, (-1,)) )
    #-> tensor([ 0,  1,  2,  3])
```

Torch.view Returns a new tensor with the same data as the original tensor but of a different shape. The returned tensor shares the same data and must have the same number of elements, but may have a different size. For a tensor to be viewed, the new view size must be compatible with its original size and stride.

```
x = torch.randn(4, 4)
print(x.size())
y = x.view(16)
print(y.size())
z = x.view(-1, 8)
# the size -1 is inferred from other dimensions
z.size()
```

```python

a = torch.randn(1, 2, 3, 4)
print(a.size())
b = a.transpose(1, 2)
# Swaps 2nd and 3rd dimension
print(b.size())
c = a.view(1, 3, 2, 4)
# Does not change tensor layout in memory
print(c.size())
print(torch.equal(b, c))
```

torch.dot() Computes the dot product of two 1D tensors.

```python
torch.dot(torch.tensor([2, 3]), torch.tensor([2, 1]))
# -> tensor(7)
```

torch.mm Performs a matrix multiplication of the matrices mat1 and mat2. If mat1 is a (n×m) tensor, mat2 is a (m× p) tensor, out will be a (n× p) tensor. This function does not broadcast. For broadcasting matrix products, use torch.matmul().

```python
mat1 = torch.randn(2, 3)
mat2 = torch.randn(3, 3)
torch.mm(mat1, mat2)
# -> tensor([[ 0.4851,  0.5037, -0.3633], [-0.0760, -3.6705,
   2.4784]])
```

torch.matmul() Matrix product of two tensors. The behavior depends on the dimensionality of the tensors (see `https://pytorch.org/docs/stable/generated/torch.matmul.html#torch.matmul`)

```python
# vector x vector
tensor1 = torch.randn(3)
tensor2 = torch.randn(3)
torch.matmul(tensor1, tensor2).size() # -> torch.Size([])
# matrix x vector
tensor1 = torch.randn(3, 4)
tensor2 = torch.randn(4)
torch.matmul(tensor1, tensor2).size() # ->  torch.Size([3])
# batched matrix x broadcasted vector
tensor1 = torch.randn(10, 3, 4)
tensor2 = torch.randn(4)
torch.matmul(tensor1, tensor2).size() # -> torch.Size([10, 3])
# batched matrix x batched matrix
tensor1 = torch.randn(10, 3, 4)
tensor2 = torch.randn(10, 4, 5)
```

```
torch.matmul(tensor1, tensor2).size() # -> torch.Size([10, 3,
    5])
# batched matrix x broadcasted matrix
tensor1 = torch.randn(10, 3, 4)
tensor2 = torch.randn(4, 5)
torch.matmul(tensor1, tensor2).size() # -> torch.Size([10, 3,
    5])
```

Exercise 1. Calculate on your own (without coding it on python) the results of the previous print() or .size() functions. After, run the code in Google Colab and compare with your results.

Exercise 2. Solve the exercises proposed in Lab_1.ipynb

References

[1] http://www.numpy.org/.

[2] http://scikit-learn.org/.

[3] https://matplotlib.org/.

[4] https://www.anaconda.com.

[5] http://jupyter.org/

[6] https://research.google.com/colaboratory/faq.html