

Machine learning for vision and multimedia

(01 URPOV)

Lab 00 – Environment Setup
Lia Morra

2024–2025

Environment setup

- During the course we will exploit the following two options:
 - ♦ Google Colaboratory (Colab) – **preferred**
 - online environment with free access to GPU resources (with time limitations)
 - no setup required
 - notebook based (more on that later...)
 - preferred choice for lab exercises and fast experimentation
 - ♦ Python/Anaconda environment on your personal computer/laptop:
 - effective option for learning and debugging (even if you do not have a GPU)
 - more useful for large projects
-

Deep learning software stack

Level	Functionalities	Options
Analysis	Graphical representation Interaction	Tensorbard, DIGITS, ...
Deep learning workflow management library	High-level network definition Dataset management	Keras, PyTorch , Lasagne, ...
Computational graph management	Forward/backpropagation Gradient computation	Tensorflow, PyTorch , Caffe, Caffe2, ...
Deep learning primitives	Low-level operations GPU optimization	cuDNN , CUDA
Hardware		CPU, GPU, TPU

The present guide covers installation of the libraries in bold

Useful Libraries

Additional useful libraries include:

- ◆ Numpy: matrix and multi-dimensional array manipulations
 - ◆ Scikit-learn: general purpose machine learning and data analytics library
 - ◆ Matplotlib: Python-based plotting library
 - ◆ Pandas: efficient data manipulation library
-

Introduction to Anaconda

- An easy personal setup uses **Anaconda** as Python distribution and **conda** as package manager*. Anaconda is an open-source distribution of Python and R for data science.
- The distribution was created to simplify package management and is compatible with Windows and Linux.
- Anaconda support the creation of **virtual environments**, i.e. named and isolated copies of Python that maintain their own files, directories and paths:
 - ♦ Different environments can use different specific version of libraries or even Python interpreter without affecting one another
 - ♦ Environments can be exported and recreated in different platforms, facilitating reproducibility across systems
- An `.yaml` environment file can be used to create and replicate the same environment

*For Windows and Linux Systems

Installing Pytorch*

1. Install Anaconda from <https://www.anaconda.com/>
2. (Optional) Install NVIDIA GPU driver ≥ 418.39
3. Open Anaconda Prompt App (Windows10) or Bash command line (Linux)
4. Create environment: `conda create --name pytorch_env`
5. Activate environment: `conda activate pytorch_env`
6. Install Pytorch: `conda install pytorch torchvision torchaudio -c pytorch`
7. Verify the installation by running the following Python code:

```
import torch
print(torch.__version__)
```

*For Windows and Linux Systems

A note on GPU use

- In order for Tensorflow/Pytorch to run on GPU, the driver must be compatible with the CUDA version.
- See for further information:
<https://docs.nvidia.com/deploy/cuda-compatibility/index.html>

A note on versions

- All the material for this class will be based on Python 3.x
 - ♦ Python has officially dropped support for Python 2
- There could be minor differences in versions across systems (last update September 2024)
 - ♦ Colab at the moment includes Pytorch v2.4.1
 - ♦ Anaconda supports v2.4.1
 - ♦ Differences should be minor (for our purposes)

Verify GPU status with nvidia-smi

DRIVER
VERSION

```
(C:\MachineLearningSetup\Anaconda3) c:\MachineLearningSetup\Projects>nvidia-smi
Sat Sep 26 17:21:31 2020
```

NVIDIA-SMI 456.38				Driver Version: 456.38		CUDA Version: 11.1	
GPU	Name	TCC/WDDM		Bus-Id	Disp.A	Volatile Uncorr. ECC	
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage		GPU-Util	Compute M.
=====							
0	GeForce GTX 960M	WDDM	00000000:01:00.0	Off			N/A
N/A	62C	P0	N/A / N/A	40MiB / 4096MiB		0%	Default

Processes:					
GPU	GI	CI	PID	Type	Process name
	ID	ID			
=====					
No running processes found					

```
(C:\MachineLearningSetup\Anaconda3) c:\MachineLearningSetup\Projects>
```

MEMORY
USE

Testing installation

- Test whether Pytorch is correctly imported and whether GPU (if available) is found

✓
0 s



```
import torch  
print(torch.__version__)
```



2.4.1+cu121

✓
0 s



```
[3] print(torch.cuda.is_available())
```



True

✓
0 s



```
[6] print(torch.cuda.device_count())
```



1

✓
0 s



```
print(torch.cuda.get_device_name(0))
```



Tesla T4

Colab

1. Create a user account
 2. Go to <https://colab.research.google.com/>
 3. Create new notebook
 4. Activate GPU hardware acceleration
 - ◆ Runtime → Cambia tipo di runtime / Change runtime environment
 5. Main libraries (numpy, tensorflow, pytorch) are already preinstalled
 - ◆ Current version at time of writing is 2.4.1
-

Jupyter notebooks

“Notebook documents (or “notebooks”, all lower case) are documents produced by the [Jupyter Notebook App](#), which contain both computer code (e.g. python) and rich text elements (paragraph, equations, figures, links, etc...). Notebook documents are both human-readable documents containing the analysis description and the results (figures, tables, etc..) as well as executable documents which can be run to perform data analysis.”

Notebooks can be created either using Colab or a Jupyter Notebook server.

Order of
execution

Jupyter Demo Last Checkpoint: 8 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

This short notebook generates and print a bi-dimensional random dataset

In [1]:

```
import libraries
import numpy as np
from matplotlib import pyplot as plt
```

First generate data with linear distribution

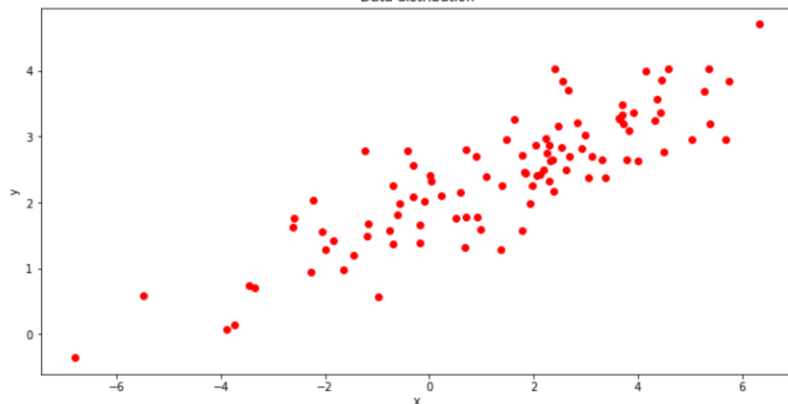
In [2]:

```
X = 2.5 * np.random.randn(100) + 1.5 # Array of 100 values with mean = 1.5, stddev = 2.5
res = 0.5 * np.random.randn(100) # Generate 100 residual terms
y = 2 + 0.3 * X + res # Actual values of Y
```

In [3]:

```
plt.figure(figsize=(12, 6))
plt.plot(X, y, 'ro') # scatter plot showing actual data
plt.title('Data distribution')
plt.xlabel('X')
plt.ylabel('y')
plt.show()
```

Data distribution



In []:

Select type of cell (code or text)

Output of each cell is
generated and printed right
underneath

Launching Jupyter notebooks

- A Jupyter notebooks server can be launched from the command line (Windows/Linux) or Notebook App (Windows).
- Available notebooks will be then available through the default browser on localhost:8888 (can be configured).

```
(keras_tf) c:\MachineLearningSetup\Projects>jupyter notebook
[I 19:32:42.001 NotebookApp] JupyterLab alpha preview extension loaded from C:\MachineLearningSetup\Anaconda3\lib\site-packages\jupyterlab
JupyterLab v0.27.0
Known labextensions:
[I 19:32:42.003 NotebookApp] Running the core application with no additional extensions or settings
[I 19:32:42.346 NotebookApp] Serving notebooks from local directory: c:\MachineLearningSetup\Projects
[I 19:32:42.351 NotebookApp] 0 active kernels
[I 19:32:42.351 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/?token=38ee193bd99e749ee3740236f23c383ad09807e766d9f614
[I 19:32:42.351 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 19:32:42.351 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,
to login with a token:
    http://localhost:8888/?token=38ee193bd99e749ee3740236f23c383ad09807e766d9f614
[I 19:32:44.431 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

Jupyter notebook vs. Python IDE

What to love about notebooks

- Mixing code + text = (potentially) very well documented code
- Allows to describe in detail data and data manipulations, very useful for data science
- More interactive than Python console, allows to easily understand what is going on
- Web-based technology allows to access remote (work-from-home) or cloud computing resources easily
- Great for quick experimenting, prototyping and demos

What to hate about notebooks

- Arbitrary execution order **can lead to very weird, unexpected and unreproducible behavior** as the output of each cell depends on the **hidden** state of previous variables
- Notebooks make it more difficult to follow good software engineering practices, resulting in messy code with poor modularity
- They do not support command line interfaces: parameters and paths are often hard-coded in the cells, batch processing is not supported, not suitable for deployment
- Large projects with multiple classes are better handled with traditional python modules
- Notebooks make version control quite harder and step-by-step debugging is cumbersome to implement

Solution: use notebooks for lab exercises, smaller projects, web-based access

For larger projects and/or physical GPUs, switch to a more traditional IDE-based environment (e.g., PyCharm)
