A Hands-On Approach to Scientific Artificial Intelligence Programming

Iciar Alfaro, Carlos Bermejo, Elías Cueto, David González,
Mikel M. Iparraguirre, Lucas Tesán, Alicia Tierz, Pau Urdeitx

PRENSAS DE LA UNIVERSIDAD DE ZARAGOZA

A HANDS-ON APPROACH
TO SCIENTIFIC ARTIFICIAL
INTELLIGENCE PROGRAMMING

A HANDS-ON APPROACH TO SCIENTIFIC ARTIFICIAL INTELLIGENCE PROGRAMMING

Iciar Alfaro, Carlos Bermejo, Elías Cueto, David González, Mikel M. Iparraguirre, Lucas Tesán, Alicia Tierz, Pau Urdeitx

PRENSAS DE LA UNIVERSIDAD DE ZARAGOZA

colección
textos docentes

Colección de Textos Docentes, n.º 337

Prensas de la Universidad de Zaragoza. Edificio de Ciencias Geológicas, c/ Pedro Cerbuna, 12, 50009 Zaragoza, España. Tel.: 976 761 330
puz@unizar.es http://puz.unizar.es

une Esta editorial es miembro de la UNE, lo que garantiza la difusión y comercialización de sus publicaciones a nivel nacional e internacional.

ISBN 979-13-7014-158-5
Impreso en España
Imprime: Servicio de Publicaciones. Universidad de Zaragoza
D.L.: Z 1161-2026

To our students.

Acknowledgements

We would like to thank Professor Francisco Chinesta for the support he has given us over the years; without his friendship and scientific guidance, the development of a new line of research within our group would not have been possible.

The authors also acknowledge the support of the Spanish Ministry of Science and Innovation, through AEI/10.13039/501100011033 (grant no. PID2023-147373OB-I00). The authors acknowledge the support of the Ministry for Digital Transformation and the Civil Service, through the ENIA 2022 Chairs for the creation of the university-industry chairs in Artificial Intelligence, through grant TSI-100930-2023-1. The authors also acknowledge the support by Keysight-ESI Group through the ENIA Chair, as well as our students for their hard work, both in attending the courses and in the development of this material.

1. Introduction

This book brings together the teaching materials used in the courses on scientific artificial intelligence delivered by the Keysight-UZ Chair of National AI Strategy of the Ministry of Digital Transformation at the University of Zaragoza during the years of its existence, from 2024 onwards.

As will be seen, the material takes a predominantly practical approach and is designed to enable students to experiment with all the modules in the programme.

This course is designed for postgraduate students. Many aspects will be better understood if students have been introduced to basic concepts of physics during their undergraduate studies, particularly mechanics, which is the focus of this book. If they have a background in scientific computing—and particularly in computational engineering—and are familiar with the finite element, finite volume, and finite difference methods, they are sure to find the content both accessible and engaging!

The field of scientific artificial intelligence is fascinating. The possibility that artificial intelligence can make predictions—whether using experimental or synthetic data, or relying solely on the differential equations that govern the phenomena under study—is, quite simply, fascinating. The challenge is for this artificial intelligence to be truly predictive in an engineering sense: that the predictions it makes can be verified and validated, so that industry can ultimately place its trust in the benefits this new discipline can bring.

It is the task of engineers, physicists, and mathematicians to turn this family of techniques into the tool that industry needs right now. The responsibility lies with us, and it begins with training a new generation of researchers in the field who will, no doubt, bring this to a successful conclusion.

2. First steps: Environment installation

This chapter provides a comprehensive guide to setting up a robust Python environment for machine learning development. The necessary instructions for installing the key machine learning libraries—NumPy, Matplotlib, TensorFlow, PyTorch, ...—are included. This chapter is intended for beginners and intermediate users, to equip them with the tools and knowledge needed to work in machine learning.

It is important to establish a reproducible Python environment using Conda to manage project-specific dependencies. In particular, set up essential tools, including PyTorch for deep learning development. There is a framework that abstracts repetitive code to build more scalable, readable, and modular deep learning pipelines, called `PyTorch Lightning`. It will be useful to generate a requirements file to document all dependencies and achieve consistent configuration across all systems.

2.1 Set-up of the working environment

For first steps, begin by setting up a battle-tested development environment—this same configuration will power all of this book's projects, ensuring consistent behavior across every application.

At this point, a question may arise: are Conda environments indispensable? Working with isolated Conda environments is crucial for three key reasons:

- Dependency management: different projects often require conflicting package versions. Conda environments prevent the "dependency hell" problem by keeping each project's requirements separate.

- Conda allows you to specify and install a particular version of the Python interpreter for each environment. This is essential for maintaining compatibility with legacy code or testing your projects against the latest releases without affecting the rest of your system.

- Reproducibility: an environment file (`environment.yml`) lets exactly recreate the same setup across different machines and collaborators — critical for reliable results.

- Project Isolation Experiments will not interfere with each other or break your base Python installation. The user can safely test new packages without system-wide consequences.

Pro tip

Always create a new environment per project, and document dependencies using:

```
conda env export > environment.yml
```

```
# Install versions with known parameter mismatch
!pip install -q seaborn==0.10.1
# Install a compatible matplotlib version
!pip install -q matplotlib==3.2.2 --no-cache-dir

import seaborn as sns
import matplotlib.pyplot as plt

print("seaborn version:", sns.__version__)
print("matplotlib version:", plt.matplotlib.__version__)

try:
    # 'sns.histplot' didn't exist in seaborn 0.10.1, so this will raise
        an AttributeError
    sns.histplot(data=[1, 2, 2, 3, 3, 3])
    plt.title("Histogram")
    plt.show()
except Exception as e:
    print("Unexpected error:")
    print(e)
```

2.1.1 The nontrivial consequences of ignoring environments

Let's see a real example of version conflicts. In Table 2.1, we can check the compatibility summary table for Torch, Lightning, NumPy, and Matplotlib.

To add some context to the table:

- TensorFlow and Numba require NumPy $\leq$ 2.1, so NumPy 2.x conflicts with them.
- Thinc (used by spaCy) requires NumPy $\geq$ 2.0, conflicting with PyTorch ecosystems needing NumPy 1.x.

Library	**Recommended Version**	**Key Dependencies / Requirements**	**Common Incompatibilities**	**Notes / Recommendations**
PyTorch (torch)	$2.0.x, 2.1.x$	Compatible with NumPy $\geq$1.20 and ≤ 2.0	NumPy $\geq$2.0 may cause internal errors or binding issues	Keep NumPy $\leq$ 2.0 to avoid C/C++ binding problems with PyTorch
PyTorch Lightning	2.0.x	Depends on compatible PyTorch	Same as PyTorch, relies on NumPy ≤ 2.0	Generally no issues with Matplotlib; no strict Matplotlib version restrictions
NumPy	1.26.x	Numerical base, library compatibility	NumPy 2.x causes incompatibility with TensorFlow, Numba, SciPy, Thinc, others	Version 1.26.x is stable for most frameworks; NumPy 2.x is very new and can cause conflicts
Matplotlib	3.8.x	Compatible with NumPy $\geq$1.17	Some functions like `stem(use_line_collection=True)` changed after 3.1	Matplotlib 3.8.x works fine with NumPy 1.26.x; watch for API changes in specific functions

Table 2.1: Library compatibility and recommendations

- Google Colab preinstalls some dependencies with fixed versions (pandas, packaging, tensorflow), which may cause version conflicts.
- Matplotlib function signatures may have changed between versions; check documentation when using arguments like use_line_collection.

To build a small script to automatically check installed library versions and suggest compatible upgrades/downgrades. Take a look to key missing or changed features across versions:

- PyTorch Lightning: trainer module/class.
  - NOT available in very early versions (≤ 1.0).
  - Present and stable in versions $1.0+$.
- PyTorch (torch): Complex CUDA support.
  - Enhanced in recent versions; older versions lack features for newer GPU architectures.
- PyTorch Geometric (PyG): Version-specific compatibility for Graph Neural Networks.
  - Essential for the advanced GNN architectures discussed in subsequent chapters.

  - PyG often lags behind the latest PyTorch releases. It requires specific binaries that must match both the PyTorch version and the CUDA toolkit, making version pinning indispensable to avoid installation failures.

- NumPy: Version 2.x.
  - Backward-incompatible changes in internal C API, causing binding errors with some libraries.
  - New functions and deprecations compared to $1.x$ (e.g., changes in random API).

- Matplotlib: `plt.hist()` vs `sns.histplot()` (Seaborn)
  - `sns.histplot` introduced in seaborn 0.11.0, absent in older versions.

So, for a brief summary:

- Check the official changelogs or documentation for breaking changes.
- When backporting code to older versions, expect missing modules/classes or different method signatures.
- For new features, upgrade cautiously and verify dependency compatibility.

The next code install seaborn and matplotlib compatible with pandas 2.x in Colab:

```
# Installing specific, compatible versions of seaborn and matplotlib
!pip install -q seaborn==0.13.1

import seaborn as sns
import matplotlib.pyplot as plt

print("seaborn version:", sns.__version__)
print("matplotlib version:", plt.matplotlib.__version__)

try:
    sns.histplot(data=[1, 2, 2, 3, 3, 3])
    plt.title("Histogram (compatible)")
    plt.show()
except Exception as e:
    print("Unexpected error:")
    print(e)
```

2.2 Conda environment

Let us look at the few steps to follow for installing the Conda environment. Although the instructions below are specifically tailored for Google Colab to ensure a smooth start for everyone, we strongly recommend following a similar setup on your local machine. The logic, commands, and environment management workflow remain nearly identical, making it easy to transition your projects from the cloud to a professional local development workstation.

2.2.1 Step 1: Install Conda

Miniconda is a lightweight Conda distribution that allows us to manage isolated Python environments and packages [Anac16]. In this step, how to install Miniconda in the Colab system and configure the necessary environment variables so the system recognizes Conda commands, are described. This is essential because Colab doesn't come with Conda pre-installed like other environments.

In this step it is necessary:

- Download the Miniconda installer.
- Make the installer executable.
- Run the installation to /usr/local.
- Configure the system PATH to include Conda.
- Verify the Conda installation.

This first step is done by running the next code.

```
# Download and install Miniconda silently in /usr/local
!wget -nc https://repo.anaconda.com/miniconda/Miniconda3-la
test-Linux-x86_64.sh
!chmod +x Miniconda3-latest-Linux-x86_64.sh
!bash ./Miniconda3-latest-Linux-x86_64.sh -b -f -p /usr/local

# Add conda to the current Python environment's PATH
import sys
sys.path.append('/usr/local/lib/python3.8/site-packag
es')
import os
os.environ['PATH'] = "/usr/local/bin:" + os.environ['PATH']
# Verify installation
!conda --version
```

2.2.2 Step 2: Create new Conda environment.

Creating isolated environments is a best practice for managing different projects with their own dependencies. Here, an environment called "Zaragoza" with Python 3.10 and basic scientific packages pre-installed, are established. The version checks ensure everything was set up correctly. Conda environments prevent conflicts between package versions across different projects.

In this step, it is necessary:

- Create a new environment named "Zaragoza" with Python 3.10.
- Pre-install core scientific packages (NumPy, SciPy, Matplotlib).
- Check the Python and NumPy versions in our new environment.
- List all available environments to verify creation.

The second step is done by running the next code.

```
# Create a Conda environment named "Zaragoza" with Python 3.10 and
    common packages
!pip install tabulate
!conda create -n Zaragoza python=3.10 numpy scipy pandas matplotlib
    tqdm scikit-learn ipywidgets -y --quiet
# Run Python and check version using the Conda environment
!conda run -n Zaragoza python -c "import sys; print('Python version:',
    sys.version)"
!conda run -n Zaragoza python -c "import numpy as np; print('NumPy
    version:', np.__version__)"

import subprocess
from tabulate import tabulate
# Run 'conda env list' and capture the output
output = subprocess.run(["conda", "env", "list"], capture_output=True,
    text=True)
# Split the output into lines and remove empty lines
lines = output.stdout.strip().split("\n")
# First line is the header, skip the second (divider), process the rest
envs = []
for line in lines[2:]:
    if line.strip():  # ignore blank lines
        parts = line.split()
        name = parts[0]
        path = parts[-1]
        active = "*" if "*" in parts else ""
        envs.append([name, path, active])
# Display results as a compact markdown-style table
print(tabulate(envs, headers=["Environment", "Path", "Active"],
    tablefmt="github"))
```

2.2.3 Step 3: Install additional packages

In this step, we expand the environment's capabilities by installing PyTorch and PyTorch Lightning, essential frameworks for deep learning. We specify the CPU-only version for compatibility with Colab's environment. Installing in the isolated environment ensures these dependencies won't affect other potential environments or the base system.

In this step, it is necessary:

- Install PyTorch suite (torch, torchvision, torchaudio).
- Add PyTorch Lightning from conda-forge.
- Activate the environment to verify installation.

This third step is done by running the next code.

```
# Install PyTorch and PyTorch Lightning in the Conda environment
    "Zaragoza"
!conda install -n Zaragoza pytorch torchvision torchaudio cpuonly -c
    pytorch -y --quiet
!conda install -n Zaragoza pytorch-lightning -c conda-forge -y --quiet

# Activate the environment (alternative method)
!source activate Zaragoza

# For Colab, you might need to use:
# !conda activate Zaragoza
```

2.2.4 Step 4: Basic environment usage

Now that we have our environment set up, it's important to understand how to properly use it. This step shows the basic commands for activating the environment and verifying you're working within it. Remember to always activate your environment before working on your project.

In this step, it is necessary:

- Properly activate our Conda environment.
- Verify you're working in the correct environment.
- List installed packages.
- Deactivate when finished.

Pro tip

Validate your environment matches ours with:

```
conda list --export > environment_audit.txt
```

This final step is done by running the next code.

```
# Activate the environment (Colab-specific method)
!conda activate Zaragoza
# Verify you're in the correct environment
!conda info
# Check installed packages in the active environment
!conda list -n Zaragoza
# To deactivate when finished (though not strictly necessary in Colab)
!conda deactivate
```

2.2.5 Exporting and recreating Conda environment

Let us export the environment to:

- An environment file: `environment.yml`.
- A requirements file: `requirements.txt`.

```
## Copy the environment with YAML
# List of Packages in Zaragoza environment
print('Packages in Zaragoza:')
!conda list -n Zaragoza | grep -E
    '^(python|numpy|matplotlib|scipy|torch)'

# Export environment to YAML
!conda env export -n Zaragoza > environment.yml
# Create new environment (copy of Zaragoza) from exported YAML
!conda env create -n conda_cpy -f environment.yml
# List of Packages in conda_cpy environment
print('Packages in conda_cpy:')
!conda list -n conda_cpy | grep -E
    '^(python|numpy|matplotlib|scipy|torch)'
```

And recreate the environment on the new system:

- Copy environment with `environment.yml`.
- Create new environment and install packages based on `requirements.txt`.

Contents

ISBN 979-13-7014-158-5

colección
textos docentes

TECNOLÓGICAS

PROYECTO FINANCIADO POR SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

Financiado por la Unión Europea
NextGenerationEU

GOBIERNO DE ESPAÑA
MINISTERIO PARA LA TRANSFORMACIÓN DIGITAL Y DE LA FUNCIÓN PÚBLICA
SECRETARÍA DE ESTADO DE DIGITALIZACIÓN E INTELIGENCIA ARTIFICIAL

Plan de Recuperación, Transformación y Resiliencia

España | digital 2026

KEYSIGHT