Development Environment Setup

Deep Network Development - Spring 2026

February 12, 2026

0.1 Python

Throughout this semester, we will be using **Python 3.13**. The installation binaries can be found at https://www.python.org/downloads/. After installation, you can verify the setup with the following command:

```
$ python --version
```

0.2 Running Scripts

Python is an interpreted language, meaning we need the Python interpreter to run our programs. There are two ways to execute scripts:

1. From the interactive Python shell, which can be launched with the **python** command:

   ```
   $ python
   Python 3.13.6 (tags/v3.13.6:4e66535, Aug 6 2025, 14:36:00)
      [MSC v.1944 64 bit (AMD64)] on win32
   Type "help", "copyright", "credits" or "license" for more
      information.
   >>>
   ```

   Once in the shell, you can enter Python statements and they will be evaluated in real time

```
>>> print("Hello World")
Hello World
>>>
```

2. By providing a source file. Python source files are typically created with the **.py** extension:

```
# main.py
print("Hello World!")
```

then pass this source file to the interpreter:

```
$ python main.py
```

0.3 Installing Packages

Packages are installed using the **pip** package manager. Depending on your needs, there are several ways to install packages. For example, to install a few packages, use:

```
$ pip install <csomag1> <csomag2> ... <csomagn>
```

For example, to install the **numpy**, **pandas**, and **matplotlib** packages:

```
$ pip install numpy matplotlib pandas
```

Or, to install specific versions:

```
$ pip install numpy==2.3.2 pandas==2.3.1 matplotlib==3.10.5
```

To install multiple packages at once, you need a **requirements.txt** file listing the packages and optionally their versions. Then use **pip** with the **-r** flag followed by the **requirements.txt** file:

```
$ pip install -r requirements.txt
```

An example **requirements.txt** file:

```
# requirements.txt
numpy==2.3.2
pandas==2.3.1
matplotlib==3.10.5
scikit-learn>=1.5.0
requests
```

0.4 Installing PyTorch

Throughout this semester, we will be using PyTorch, an open-source machine learning library. PyTorch supports both CPU and GPU (CUDA) based computations, making it flexible across different hardware environments.

The lab exercises and exams have been designed to not require significant computational resources, so we recommend installing the **CPU** version. Installation commands for PyTorch can be found at https://pytorch.org/get-started/locally/, where you can select your platform, package manager, and CPU or CUDA version to get the appropriate installation command.

If you need to install an earlier version of PyTorch, the required commands can be found at https://pytorch.org/get-started/previous-versions/.

0.5 Jupyter Notebook

Throughout the semester, example code and exams will be provided in notebook format.

Jupyter Notebook is an interactive environment that allows you to run Python code, write documentation, and create data visualizations all within a single interface. A notebook is a document composed of cells, which can contain:

- **Code cells** (for running Python code)
- **Markdown cells** (for text explanations, with LaTeX formulas)
- **Output cells** (e.g., figures, tables, code execution results)

Jupyter Notebook can be installed with the following command:

```
$ pip install notebook
```

To launch the notebook editor:

```
$ jupyter notebook
```

This opens a browser window where you can create new notebooks or edit existing ones.

- **Shift + Enter** – Run cell and move to the next cell
- **Ctrl + Enter** – Run cell
- **Esc + A** – Insert cell above the current cell
- **Esc + B** – Insert cell below the current cell

Important!

Variables created in cells are placed in the global scope. Be careful not to accidentally overwrite existing variables, and avoid unintended side effects during code execution.

0.6 Jupyter Lab

Jupyter Lab is an enhanced version of Jupyter Notebook that provides a complete development environment for editing notebooks, scripts, and other files.

Jupyter Lab can be installed with the following command:

```
$ pip install jupyterlab
```

After installation, run:

```
$ jupyter lab
```

This opens a more modular interface in the browser that supports:

- Running multiple notebooks in parallel
- Sidebars for file and process management
- An integrated terminal for command-line tasks

0.7 Google Colab

Google Colab (Colaboratory) is an online Python development environment built on Jupyter Notebook, but it requires no installation or configuration. One of its greatest advantages is that it provides **free GPU and TPU access**, making it particularly useful for deep learning and data processing tasks.

Google Colab comes with its own Python environment. To install required packages, you can run shell commands in a code cell by prefixing them with **!**. For example:

```
!echo "Hello World!"
```

or

```
!pip install numpy
```

0.8 VSCode

One of the recommended text editors for development at home is **Visual Studio Code**. For Python support and notebook management, we recommend installing the following extensions:

- **Python** (Microsoft) – syntax highlighting, debugging, execution
- **Pylance** – advanced code completion and type checking
- **Jupyter** – Jupyter notebook support
- **autoDocstring** – automatic docstring generation for functions
- **Error Lens** – displays errors and warnings inline

You can run scripts from the built-in terminal or by clicking the triangle icon in the top-right corner. Running and managing notebooks works similarly to Jupyter Notebook.

0.9 Conda (Miniconda)

If you work on larger machine learning projects, you will likely use Linux – either directly, in a virtual machine, or in a container. Additionally, you may apply multiple machine learning methods simul-

taneously, each requiring different Python and package versions. One of the most popular tools for managing such parallel environments is **Anaconda** or its smaller variant, **Miniconda**.

Conda is a package and environment manager that allows you to use different versions of Python and other libraries. It is particularly useful when working on multiple Python projects and you need to ensure proper version and dependency compatibility.

Miniconda is a minimal distribution of Conda that includes only the essential components. It is smaller in size and faster to install than the full Anaconda distribution.

After installation, you can create an environment with the following command:

```
$ conda create -n test
```

You can also specify the desired Python version:

```
$ conda create -n test python=3.13
```

Or include a **requirements.txt** file:

```
$ conda create -n test --file requirements.txt python=3.13
```

0.10 Exam Environment

This semester, we will use the following packages. These are also available in the computer lab, and all lab materials and exam problem sets have been tested with these versions. To avoid errors due to version differences, we recommend working with a similar environment. A **requirements.txt** file will be provided.

Packages available during exams:

- **matplotlib** - 3.10.5
- **numpy** - 2.3.2
- **pandas** - 2.3.1
- **scikit-image** - 0.25.2
- **scikit-learn** - 1.7.1
- **scipy** - 1.16.1
- **torch** - 2.8.0

- **torchaudio** - 2.8.0
- **torchvision** - 0.23.0

During exams, we recommend using **Jupyter Lab** or **Jupyter Notebook**. Documentation will be available offline in **PDF** and **docstring** formats. The built-in documentation for functions or modules can be accessed using the **help()** function. For example:

```
>>> import torch
>>> help(torch.nn.Conv2d)
Help on class Conv2d in module torch.nn.modules.conv:

class Conv2d(_ConvNd)
 |  Conv2d(
 |      in_channels: int,
 |      out_channels: int,
 |      kernel_size: Union[int, tuple[int, int]],
 |      stride: Union[int, tuple[int, int]] = 1,
 |      padding: Union[str, int, tuple[int, int]] = 0,
 |      dilation: Union[int, tuple[int, int]] = 1,
 |      groups: int = 1,
 |      bias: bool = True,
 |      padding_mode: str = 'zeros',
 |      device=None,
 |      dtype=None
 |  ) -> None
 |
 |  Applies a 2D convolution over an input signal composed of
   several input
 |  planes.
...
```

or for numpy:

```
>>> import numpy as np
>>> np.info(np.array)
array(object, dtype=None, *, copy=True, order='K', subok=False,
    ndmin=0,
      like=None)

Create an array.

Parameters
...
```