Artificial Intelligence II: Deep learning methods

Lab Setup Guide

Vlad Vasilescu Ana Neacșu

March 16, 2025

Contents

1 Intro

This guide is intended for setting up the environment to run the algorithms presented at Lab. We'll be using `Python 3.10` and `Jupyter notebooks`. If you prefer working without installing any local stuff, you can jump directly to Google Colab Section.

1.1 Python & Virtual Environments

You can find the installers for `Python 3.10` here. After installing, check if you can access it by opening a terminal and run

```
> python --version
```

to verify if the correct version is installed. If no Python executable can be found, see How to Add Python to PATH.

1.2 Creating a Virtual Environment

Virtual environments in Python are isolated environments that enable you to manage and install dependencies for your Python projects, separately from the system-wide Python installation. Below, you can find two options for creating and working with virtual environments:

Option #1: `conda`

Create a new environment:

```
> conda create --name ia2 python=3.10
```

Activate it:

```
> conda activate ia2
```

Deactivate the current env:

```
> conda deactivate
```

Option #2: `venv`

Create a new environment:

```
> python -m venv ia2
```

Activate it:

- For Windows:

  ```
  > ia2_venv\Scripts\activate
  ```

- For Linux:

  ```
  > source ia2_venv/bin/activate
  ```

Deactivate the current env:

```
> deactivate
```

1.3 Installing Packages

Once inside a virtual environment, you'll need to install some packages that your project needs. Packages in Python can be installed using the `pip` package manager:

```
> pip install <package_name>
```

or

```
> python -m pip install <package_name>
```

If you want to install a specific version of a package, you should do:

```
> pip install <package_name>==<package_version>
```

You can generate a list of dependencies for the current virtual environment by running:

```
> pip freeze > requirements.txt
```

This will create a file `requirements.txt` with all installed Python packages and their corresponding versions.

If you want to work on an existing project which provides an already generated `requirements.txt`, you can run the following:

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

This will install all packages specified in the given file.

Note: A `requirements.txt` is provided on the course's GitHub page, with all necessary packages for running the Lab notebooks.

2 Jupyter Notebooks

Jupyter notebooks are interactive computing environments that allow users to create and share documents containing live code, equations, visualizations, and narrative text. The name "Jupyter" is derived from the core programming languages it supports: Julia, Python, and R.

A notebook is comprised of *cells* which may include Python, Markdown or HTML code. A Python cell can be executed by `Ctrl+Enter` or `Shift+Enter`, or through the `Run` button from the interface. The output (if any) is immediately displayed under each cell.

2.1 Working Offline

In order to locally launch a Jupyter Notebook, you'll need to have `Jupyter Lab` installed in your environment:

```
> pip install jupyterlab
```

Now you can open `Jupyter Lab` by running:

```
> jupyter lab
```

in your working folder. This will automatically open it in your browser, and in the console it will output something like:

```
To access the server, open this file in a browser:
    file:<some_path>.html
Or copy and paste one of these URLs:
    http://localhost:8888/lab?token=<some_chars>
    http://127.0.0.1:8888/lab?token=<some_chars>
```

The URLs can be copy-pasted into a browser of your choice to open the `Jupyter Lab` interface.

You can check out this tutorial on how to get started with Jupyter.

2.2 Working Online – Google Colab

Google Colab is a free cloud service provided by Google that allows users to write and execute Python code through the browser. One of it's most important features is the free access o GPU and TPU (Tensor Processing Unit) resources, which we'll make use of during Lab. You can check out their tutorial.

To successfully execute the notebooks we'll be using at Lab, be sure to include all the other `.py` files, containing implemented dependencies, and the other folders with

images or data. You could also set up a folder in your Google Drive and mount it in the notebook. Or, after uploading your notebook (`.ipynb` file) here, you can manually add all the other files by accessing the `Files > Upload` section in the left menu.

Note: Simply uploaded external files will be deleted after some time, with the runtime being considered idle. In this case, it is recommended to progressively save your work by mounting a folder from your Google Drive.

3 Git Basics

Git is a distributed version control system used for tracking changes in source code during software development. When you're building an open source project, you'll need a way to document or track your code, to keep your work organized, which is what Git does.

GitHub, on the other hand, is a web-based platform for hosting Git repositories and collaborating with others on software projects. GitHub uses Git underneath, letting you manage your local Git repositories.

You can install Git on your machine by following these steps.

3.1 Git CLI commands

In order to use Git on one of your projects, navigate to the projects' folder, opne a terminal and run:

```
> git init
```

which initializes an empty Git repository in the projects' directory. To add files to the staging area (prepare them for committing), use the following command:

```
> git add <file_name>
```

Which will include the files in your Git repo for tracking. To see which files are included in your Git repo you can run:

```
> git status
```

This will list all tracked and un-tracked files, files staged for *commit* and files from your Git which have been modified (i.e. the current status of your repo). When you want to take a snapshot of your project, you can do:

```
> git commit -m "Changed file <x>"
```

which basically creates a checkpoint in the *commit tree*, encapsulating all changes that have been made relative to the previous commit. To see all previous commits, run:

```
> git logs
```

which prints out the previous commits details (hash, message, date, author).

In a scenario when you want to add a new feature or fix some bugs, the usage of *branches* comes in very handy. To learn how Git branching works, we recommend checking the interactive tutorial *Learn Git Branching*.

3.2 Connect to Github

Create a GitHub account here and create a new repo, choosing a name, a description and its visibility settings.

In the folder where your local repo is initialized, you can add the newly created GitHub repo as a *remote repository*:

```
> git remote add <alias_name> <repository_URL>
```

Usually, the main remote repository has an `<alias_name> = origin`. You can add multiple remote repositories by using different aliases.

Before transferring your work to the remote, you need to first check if there were any changes made on the remote meanwhile. To do so, run:

```
> git fetch <alias_name>
```

To visualize the *difference* between your current local branch and the remote branch (i.e. if there were any changes made on the remote but not on the local), you would do:

```
> git diff <alias_name>/<remote_branch_name>
```

which will print out all differences between your local branch (where `diff` was run) and the remote branch of choice (after doing `fetch`).

To combine your local branch with the changes made on remote, run:

```
> git merge <alias_name>/<remote_branch_name>
```

Another choice for combining two branches is through *rebase*.

The `fetch` and `merge` steps could be executed at once by running:

```
> git pull <alias_name> <remote_branch_name>
```

which will prompt you to resolve any existing conflicts.

When all conflicts have been resolved and all desired changes made, you would want to push your local changes up to the remote repo:

```
> git push <alias_name> <remote_branch_name>
```

which will push all local changes on the remote repo at the given branch.

If you just want to work on an existing project hosted on GitHub you could do:

```
> git clone <repository_URL>
```

and all corresponding remote resources will be copied in a new local folder, where you can start working.

For an in-depth discussion regarding advanced Git tactics you can check Atlassian's Git tutorials and Toptal's Advanced Git Guide.