CSCI E-80A (25387):

Introduction to Artificial Intelligence

Companion Course to E-100 Science of Intelligence

© Prof. Brian Subirana
Director, MIT Auto-ID Laboratory
77 Mass Ave, Cambridge MA 02139

Python Environments Tutorial

CENTER FOR
Brains
Minds+
Machines

Table of Contents

SUMMARY

This week we will take a deeper dive into the mathematical concepts behind deep learning including an overview of training algorithms, namely gradient descent, to optimize neural networks for specific problems. We will learn as well about some of the pitfalls of this technology, including the curse of dimensionality, and concepts of universality and generalization of gradient descent algorithms.
In your coding assignment, we will set up your coding environment for the class. We will be using a Python and package version control software called Anaconda3. This assignment will be a review of its core functionality, while at the same time taking a deep dive into how it stores and manages different versions of packages and Python releases. By the end of the assignment, you will be running a program that trains a deep learning model on one of your Conda environments.

INTRODUCTION

To begin this assignment, we will install all the necessary software to set up your coding environment for the course. This includes a coding language, Python (https://www.python.org/), a Python version manager, Anaconda3 (https://www.anaconda.com/) and a text editor, Atom (https://atom.io/). Note that this course is based on Mac OS, we therefore encourage you to work with a UNIX based system (either Linux or MacOS). We will do our best to help you troubleshoot otherwise.

Atom is a text editor with code highlighting similar to others such as Sublime Text. First off, please follow the link below to install Atom on your device:
https://atom.io

Let's now install Anaconda3. Follow the guide in the link below to install Anaconda3 for your Operating System:
https://docs.anaconda.com/anaconda/install/

Anaconda3 is a program that allows us to create environments which store different versions of Python and packages completely independently from each other. We will be using it to ensure that the code from this course isn't broken or breaks the code from other projects or courses. We will now go over the very basic commands for Anaconda3, and its terminal package manager 'Conda', as well as learn about how Anaconda3 installs Python distributions, packages and so on while keeping them separate from the global distributions in your computer.

Throughout this course, you will often be prompted to "run" commands. This refers to typing a command in your "Terminal" application (for MacOS users, which can be found in the "Utilities" folder) or "Command Prompt" for Windows. We will now learn the basic commands to create an environment with conda.

Commands

To create a conda environment, run the following command, where YOURENVNAME represents the name of your environment:

```
conda create -n YOURENVNAME python=x.x
```

To activate your environment, run the following:

```
conda activate YOURENVNAME
```

To deactivate the environment, run:

```
conda deactivate
```

There may be a point where you have multiple environments set up on your machine, and you want to see all of them. To list all your environments, run:

```
conda info --envs
```

Finally, to delete an environment, run:

```
conda remove --name YOURENVNAME –-all
```

Conda Environments Assignment

This assignment is based on the code explained in the "*Commands*" section, make sure you review it before you get on with this assignment.

All the work you do in this section should be properly recorded and compiled into a final report, combined with the other exercises. You will find the guidelines for this report at the end of the assignment.

- Install Anaconda.
- Create three new environments: one for Python 2.7 (name: *py27*), one for Python 3.6 (name: *py36*) and one for Python 3.7 (name: *py37*).
- Get information on your environments using the "info" command, and delete the environment *py37*.

VERSION CONTROL WITH CONDA

We will now build our first "Hello World" python program and run it in different environments, with the goal of learning about the importance of version management in programming. Starting on this section, and throughout the assignment, we will be using 'Bash' commands, which is the terminal language used by UNIX systems (including MacOS and Linux), such as `cd`, `ls`, `mkdir`, etc (you will learn what each of these do soon). If you have a Windows computer, refer to the Further Reading section below, and if you're already familiar with them this will serve as a review.

Further Reading (Optional)

This assignment goes through some commands that are specific to UNIX systems (which is, computers with either MacOS or Linux installed). If you have a Windows computer do not worry, all commands we use have equivalents. Below are some resources you might find useful.

Windows and Unix command line equivalents. *By Ben Bullock*
https://www.lemoda.net/windows/windows2unix/windows2unix.html

If you are still not able to find what you are looking for, Google can be a very handy tool. If that doesn't help either, check out Stack Overflow, one of the leading Q&A websites for programmers.

Stack Overflow.
https://stackoverflow.com/

We will first create a folder where to keep our code. We can do so in the terminal, after navigating to the Desktop (or wherever you want to create your folder) using the command cd as follows.

```
cd path/to/desktop
```

Then we can use the `mkdir` command to create a directory, which we will call "hello_world":

```
mkdir hello_world
```

Navigate inside it, and create a python file named `hello_world.py` using the command `touch` (note that this command can be used to create any kind of file). We use the symbols `&&` to concatenate two bash commands together.

```
cd hello_world && touch hello_world.py
```

We can open this file with our text editor of choice, in this case atom.

```
atom hello_world.py
```

In it we will write code to print "Hello World" on the terminal screen. We can do this using Python's print function. We write the following line of code in our file and save it:

```
print("Hello World!")
```

Perfect! To run this code we will need to run a conda environment where the right version of Python is installed. Let's try to connect to our Python 3.6 conda environment as follows. (if you deleted it before, just create a new environment as explained above).

```
conda activate py36
```

Finally we can run our code.

```
python hello_world.py
```

The terminal should output the sentence "Hello World!" as expected. Let's now try to run this same code on a different environment. We first need to deactivate our environment.

```
conda deactivate
```

And we can now run our Python 2.7 environment as follows.

```
conda activate py36
```

Finally we can run our code once again.

```
python hello_world.py
```

You will notice that your code outputs an error! Don't worry this is expected. Python 3 introduced a new way of writing python print statements, using

```
print("Hello World!")
```

instead of

```
print "Hello World!"
```

You will see this happening often with both python itself and its packages, which deprecate code often to improve their functionality and maintanibility. Using environments to "freeze" the package and language versions of our code is very useful to ensure our code works consistently over time.

Version Control with Conda Assignment

This assignment is based on the code explained in the "*Version Control*" section, make sure you review it before you get on with this assignment.

All the work you do in this section should be properly recorded and compiled into a final report, combined with the other exercises. You will find the guidelines for this report at the end of the assignment.

- Go through the Version Control with Conda section code and replicate it in your machine. Record all errors.

DIRECTORY STRUCTURE AND BASH

Let’s take a deeper dive into what anaconda does behind the scenes.

Further Reading (Optional)

Some of the commands we will use have very deep functionality depending on the flags attached to them (e.g. `-h` in the command `du -h`). If you want to find out more about them, you can check resources such as:

Basic Linux/Unix Commands with Examples *By Guru99*
https://www.guru99.com/must-know-linux-commands.html
or
Stack Overflow.
https://stackoverflow.com/

We can find where Anaconda is installed in our system by looking through all directories with the name “anaconda3” in our machine. We use the UNIX command “find” to do just that (make sure you’re in your user’s folder in the terminal before running this command, you can ensure this by running the command `cd` before). We use the command `sudo` to run this command in “admin” (sudo stands for “super user do”), so we don’t get errors because of denied permission to check on certain folders.

```
sudo find . -name anaconda3 2> /dev/null
```

This command might take a while, don’t worry! It has to look through all your computer files to find all folder names that match the string “anaconda3”. The “2> /dev/null” part of the command is simply there to avoid annoying warnings in some machines when running, so don’t worry too much about that. When it is done running, you should get the location of your anaconda3 folder.

```
cd path/to/anaconda3
```

One of the particularities of Anaconda is that it comes pre-installed with many software packages that are useful for data science. This is convenient for data scientists and people who use those packages regularly, but can be inconvenient for those who do not and want to conserve their hard drive space. The Anaconda 3 Distribution for Mac, for example, is almost 7 GB in size. You can check this for yourself using the following command.

```
du -sh
```

This will output the file size for the whole folder (note that it will probably be larger than 7Gb because you had to install several versions of Python for both of your environments). We can see that by looking specifically at the size of the envs folder.

```
du -sh envs
```

If you look inside this folder you will find a folder for each one of your environments.

```
ls envs
```

Let's look inside one of these environment folders. Navigate inside one as follows.

```
cd envs/py36
```

Let's see where python is stored in the environment. First make sure your environment is activated, then use the following command.

```
which python
```

You should get an output that looks something like this:

```
/Users/[User]/anaconda3/envs/py36/bin/python
```

The base conda environment comes with a python version installed as well. You can find out by running the command `conda deactivate` (you should see on the left side of your terminal your environment change to (base)), and then running the same `which python` command. The output should look something like this.

```
/Users/[User]/anaconda3/bin/python
```

Most Operating Systems come with a predownloaded Python version as well. To find out where this Python is stored, deactivate all your conda environments and once again run the `which python` command. The output should now look something like this:

```
/usr/local/bin/python
```

(note that if you don't have a python version installed in your main computer you will get an error, disregard this and keep going with the assignment)

Notice that the python installation for each of the cases we looked at is completely independent. This is very useful because as we saw previously it allows us to avoid errors from deprecated code between python versions. This will as well give us the security that we will not be breaking our previous code when working on a new project in a different environment.

The Python you are using in each environment is indeed installed in the environment folder, more specifically in the "bin" (which is not the garbage, but rather the binary folder). Let's see where packages are installed in the environment. We can use the “find” command to find all folders with “site-packages” on their name. First navigate to your environment folder.

```
cd path/to/anaconda3/envs/py36
```

And then run the find command.

```
find . -name site-packages
```

You should be able to find a folder similar to ./lib/python3.6/site-packages. All of your packages for your environment are stored here.

Let’s try and take a look at all these packages that come preinstalled with conda. We will first activate the “base” environment.

```
conda activate
```

And now we will use Python’s package manager, “pip”, to get a list of all the installed packages.

```
pip list
```

You should get a very long list of packages that come preinstalled with conda, such as numpy, matplotlib, and other data science packages. You obviously didn’t install these yourself. Note that these packages don’t get automatically installed in your new environments. Let’s check that by activating one of our environments and looking at the list of packages.

```
conda activate py36
pip list
```

You should only see a list of about 5 packages that come by default, and potentially some other packages that you may have installed globally previously. Note that you can use pip to install any packages you want on to your environment. Let’s try and install Numpy (a scientific library for Python) using the command below

```
pip install numpy
```

When it is done, if you run the command pip list once again, you should be able to see the package “numpy” has been added to the list. You will need to use the pip command often to add or remove packages and keep them up to date. See below for a couple standard commands:

```
pip install __
```
(to install package __)

```
pip install -r __.txt
```
(to install packages from text file __)

```
pip install __ --upgrade
```
(to upgrade package __)

```
pip uninstall __
```
(to uninstall package __)

Directory Structure and Bash Assignment

This assignment is based on the code explained in the "*Directory Structure and Bash*" section, make sure you review it before you get on with this assignment.

All the work you do in this section should be properly recorded and compiled into a final report, combined with the other exercises. You will find the guidelines for this report at the end of the assignment.

- Go through the Directory Structure and Bash section code and replicate it in your machine. Record all outputs.
- Write one page describing in your own words how Anaconda3 and its environments affect what Python distribution and packages are run.

CREATING AND DEPLOYING AN ENVIRONMENT

Now that you know how to create an environment, we will now set up our programming environment and run some code to make sure it is working properly. Download the file “installation_guide.zip” from _______ and unzip it. Change directories into the “installation_guide” directory:

```
cd [path/to/installation_guide]
```

Create an environment with **Python 3.6** (the Python version here is very important for the code to work properly), and download all the dependencies needed for the code by running the following command:

```
pip install –r requirements.txt
```

Run the “generic_vns_function.py” file.

```
python generic_vns_function.py
```

If the script works, after some warnings are printed, you should see something like this:

```
60000/60000 [==============================] - 3s 54us/step - loss: 0.6901 -
acc: 0.8352 - val_loss: 0.3195 - val_acc: 0.9192
```

This is your computer training a computer vision model. You will learn about this and other models later in the course. If the file ran successfully, then congratulations! You have successfully configured your environment.

Creating and Deploying an Environment Assignment

This assignment is based on the code explained in the "*Creating and Deploying an Environment*" section, make sure you review it before you get on with this assignment.

All the work you do in this section should be properly recorded and compiled into a final report, combined with the other exercises. You will find the guidelines for this report at the end of the assignment.

- Go through the section code and replicate it in your machine. Record all outputs.

Congratulations! You are done with this week's assignment. Starting from next week, you will use the environment you created to write and run code similar to the one you tested today.