TP - Python/Google Cloud/Docker

AI Frameworks

Brendan Guillouet

16 & 30 November 2020

Contents

1 Script *python*. 3
1.1 Files and folders organisation. . . . 3
1.2 Scripts . . . 3
1.3 Exercise . . . 4

2 Cloud Computing 6
2.1 *Google Cloud* vitrual machine instance. . . . 6
2.1.1 *gcloud* . . . 6
2.2 Code Execution . . . 7
2.2.1 Exercise . . . 7

3 Docker 9
3.1 Start a machine from compute engine. . . . 9
3.2 Virtual Machine setup. . . . 9
3.3 Dockerfile . . . 10
3.4 Image . . . 10
3.5 Container . . . 11
3.5.1 Interactive container . . . 11
3.5.2 Mounted Volume . . . 12
3.5.3 Background Mode . . . 12
3.6 Code Execution. . . . 12
3.7 Exercice . . . 13

Introduction

Cloud instances are usually used by data scientist to make machine and deep learning algorithm run in production, train it in order to use higher computational, etc. In this laboratory, we will go through the different steps that allow to use these type of instances. Here are the tasks we will achieve during this lab.

1. write two python scripts that will:
    - take parameters as an input, train and save convolutional neural network model,
    - load the trained model, and generate prediction on a dataset.
2. learn how to use ***Google Cloud*** instance.
    - configure the instance
    - make the python scripts run on the instance.
    - get the results back and analyze it locally.
3. learn how to use ***Docker*** container to make your code more easily reusable.

During this la we will use *Cats Vs Dogs* dataset and convolutional neural network algorithm (see HDDL image classification course for more info).

Exercice: Get the last improvement on the AI-Framework repository.

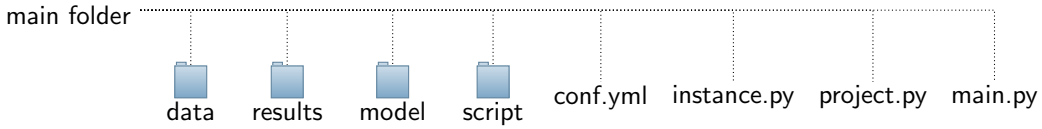
1 Script *python.*

The use of instances can be expensive. ***Google***, ***Amazon*** or ***Microsoft*** bill them hourly. They are used to make code run in production or when training a model require high computation power. It is not suitable to use it for exploratory work. Event if its possible to use *Jupyter* on these instances, it is not the most adapted tool.

Hence once the exploratory work has been done on jupyter, we need python script to run the algorithm. Good practices aims at first write a script locally (*i.e.* on your computer) and on a small amount of data to ensure the script run entirely. Once it works, we send the code and make it run on the instance.

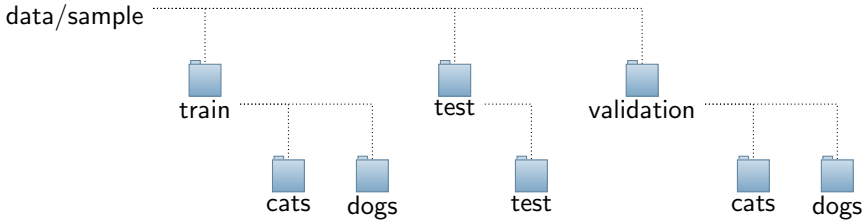
1.1 Files and folders organisation.

Files and folders organization need to be identical locally and on the instance. The *Cloud Computing* folder is organized as follow.



Exercice: *results* and *model* folders are empty and then not in the github. Create it. *data* folder is zipped. Unzipped it.

The **data** folder contains the data on which we will train, test and validate the algorithm, i.e. all the data required to make the code run. Here are how the data are organised.



This organization enable to labelled the data (images in cats folder are classified as cats).

The **model** folder will contains the different model that will be saved during the training.

The **results** folder will contains the different results file produce during the training that will be retrieve locally to be analyzed.

The **script** folder contains the python scripts.

conf.yml, instances.py, project.py, main.py : Code python permettant d'utiliser, et d'éxécuter les codes du scripts sur le serveur (Section 2.2 et 3.6.)

1.2 Scripts

Two scripts will be required to achieve our goal.

The **learning.py** script will enable to:

- load the data,
- train the mode,
- save the weight of the model (in the model folder),

- save various results (in the result folder) such as, learning and prediction time, train and test accuracy, history keras object (that contains loss and metric evolution over epochs).

The **prediction.py** script will enable to:

- download the previously trained model,
- use it to predict test data,
- save a csv file that contains the prediction (in the results folder).

subsectionScripts

1.3 Exercise

Open the *learning.py* and *prediction.py* scripts within script folders. Fill it following the instruction within it (and the instruction below), so that these scripts enable to apply the task described above.

Here are some instruction to complete this exercise.

Parameters and arguments. Applying the following command in a terminal allow run the script.

```
python learning.py
```

However, the script should be able to take arguments into account, either for manage path if the script is run locally or on the instance, of for manage model parameters. Hence, using the **argparse** (*cf. lecture slides*) library, make your script enable to received the following arguments:

- data-dir: path to the data folder
- results-dir: path to the results folder
- model-dir: path to the model folder
- epochs: number of epochs
- batch-size: size of the batch

Hence, the script can be run this way

```
python learning.py --data-dir data --epochs 10 --batch-size 256
```

Save/Load results & model. Working with Jupyter allow to display and analyze any objects created. When you work with script, one of the main difference, is that any objects that are build within a script can't be retrieve once the script if over, unless you save them. It is then really important that you save any objects that will be require later (weights of the model) or you want to analyze (prediction of the models). They are a large number of manner to save and load python

objects, and a lot of library have their own methods. Within your script you will use

- keras save/load_model method to save model.
- pickle (*cf. lectures slides*) dump/load library to save python object
- pandas to_csv/read_csv

It is also important to specify file name so that you never overwrite an existing file. Common mistake is to name the model weight files *model_weight*. Hence if you change your parameters (number of epochs), rerun the file, previous file will be always overwrite, and you might not remember to which file correspond to which parameters. Hence name all the files you saved in order that you can retrieve all parameters with which they have been generated.

Complete Pipeline Ensure that you can run both script entirely and that all the files (model, results, prediction) are produces in the correct folder. When you've done call your professor so that he/she can verify.

NB: Solutions are available within *learning_solution.py* and *prediction_solution.py* files. Play the game and do not look at it untill you try your own solution.

2 Cloud Computing

2.1 *Google Cloud* vitrual machine instance.

Default project should by My First Project
Present the default VM instances selection. Require to install Nvidia-driver. It can be done but pain in the ass
Exercise: Follow these steps to configure and start de DeepLearning VM Machines.

- Go on Deep Learning VM's AI Platform' Google Cloud Marketplace.
- Click on LAUNCH/LANCER and select the default project.
- You're now on a page that very looks like the create VM instance page. Choose a deployment name and configure the vm with the following settings:
  - Zone: *europe-west-1b* (in Belgium).
  - Machine type: *4vCPUs - 15 GB Memory, n1-standard-4* (scrool on first block menu).
  - Gpus : *1 - NVIDIA Tesla K80.*
  - Framework: Tensorflow Entreprise 2.1 (CUDA 10.1).
  - Select *Install NVIDIA GPU driver automatically on first startup?.*
- Click on DEPLOY. You'll be redirected on the deployment page where you can observe the deployment's progression. It will take a few minutes.

Go back now on the in the *VM instances* section (in *Compute Engine* menu). You should now see your VM. The green symbol indicates that it's running. **From now you start billing**.

If you click on SSH you arrive on the instance's terminal. This is a very easy way to have access to the machine, but it won't allow you to send or receive data from your local post. So we won't use this tool during this TP but the ***gcloud*** command-line tool.

2.1.1 *gcloud*

gcloud is a command-line tool that enable to manage your instance from your local terminal. You can start it, stop it, send data on it *etc.* from your local terminal.
As with *git*, you first have to "initialize" your gcloud account locally to connect it to your google cloud account and to your project.
Exercise:

- Follow the *Initialize the SDK* tutorial on the official documentation
  https://cloud.google.com/sdk/docs/quickstart-debian-ubuntu.
- Connect to your instance by running the following command **on your local terminal**:

```
gcloud compute ssh VMNAME
```

Note: when using this tool to connect to your vm, you do not arrive on the same home directory than when using the previous tool.
Python is already install on your machine. The version, 3.5 is not the most recent but it's enough to run our code. Tensorflow 2.1 is already install as well as numpy pandas etc..

Before going further, uou will have to install two packages, *unzip* and *pv* that will allow you to unzip data. To do so, apply the following command on the terminal of your vm's instance.

```
sudo apt-get install unzip pv
```

2.2 Code Execution

You code is running locally and you instance is set up and ready to run. You now have to send the data and code to this machine, run it and get the data back using the *gcloud* tools.
As you have learned now, it's better to first write a script that will apply all the command you need. *gcloud* command a particularly complex to write as you can see in the example below where it is displayed a *gcloud* command that enable to run python code with arguments on the VM.

```
gcloud compute ssh nom_instance --zone europe-west-1 --ssh-key-file DirKeyFile
    --command 'python3 learning.py --data_dir DirData -- results_dir DirResults
    --epochs 10 --batch_size 128'
```

Moreover various steps needs to be applied:

- Turn the VM on
- recreate directory organization
- send the data
- send the code
- execute the scripts
- get result file for exploration.

All these steps can lead to mistake and require adjustment. It's then better to use script.

2.2.1 Exercise

Open the following three scripts **instance.py**, **project.py** and **main.py** on a Python Editor (Spyder for example) and observe the code.

instance.py : It contains a InstanceManager's python class that allow to apply gcloud command through python code with pre-setting VM arguments (such as the zone, the instance name or the direction of the ssh key file. It also have two options that allows to print the command or not the command and execute or not the command (which will be very usefull to check your code before making it run.
Exercice: Set the following variable of the InstanceManagerclass: *zone*,: *instance_name* and the direction: *ssh_key_file*. Take a look at the different function of the class, and make sure you understand each of it.

project.py : It's a python class that allow to apply gcloud command through python code. As the instance.py command allow to apply very generic command, the ProjectManager class allow to preset number of parameters that is specific to the current project, such as direction of the folders.
Exercice: Set the following variable of the ProjectManager class: *local_folder*, and *remote_folder*. Take a look at the different function of the class, and make sure you understand each of it.

main.py : It contains the script you will run in order to apply the all pipeline.
Exercice: Check that ProjectManager class is set to only print the command. Run the script on your local terminal and check command that it would make run. Ensure that everyhing looks good. Then run the script with *execute_command* set to True. If errors appears correct it until the code works entirely.

- You should now have the different results files downloaded locally. On *Jupyter* open the different files and display the loss evolution along the different epochs.
- **BONUS**: Download the complete CatsVsDogs *zip* dataset following this lien. Make the code run with this data.

3 Docker

In the previous section, we have used a *Deep Learning's VM* from *AI Platform's Google Cloud Marketplace*. This VM already have most of the tools we need to run algorithm, only two packages (unzip and pv) needed to be install.

It does not exist a pre configured VM for all the different application and this kind of VM are not the same from a cloud computed services to another. This is why you need to configure a VM from scratch whatever the cloud service you're using and whatever the tools you need. This is what **docker** is made for.
In this section we will see

- How to setup an empty VM to make it use a docker images and GPUS.
- How to build a docker image.
- How to start, run and end a container from an image.
- Maker run a complete training pipeline from your local machine trough a container in your remove VM.

3.1 Start a machine from compute engine.

We will now configure a virtual machine from compute engine's Google Cloud services.
Exercise:

- From the main menu (click on the *hamburger*), select *Compute Engine* and then *VM instances*.
- Click on *create an instance*. You are now on the setup pages of the instance.:
  - Choose a name and *europe-west1-b* as a region and zone.
  - Choose a N1 series.
  - In **machine type**, select a standard configuration with s4 vCPU and 15Gb of memory. Add a Tesla K80 GPU. *You might select more powerfull card later :).*
  - In the **Boot disk** section, select *Ubuntu 18.04 LTS*. You can also configure the boot size disk. Set it to 50Go.
  - Click on create.

Attention.

- Your VM start automatically. So from now, your credit start decreasing.
- Your VM may not start, if all GPU for the selected region are used. If so start again and select another region.

3.2 Virtual Machine setup.

Docker allows to build container that are isolated from each other. Theses containers can contain all the softwars and libraries you need. However they are still some tools that you need to install on the virtual machine itself:

- The **nvidia-driver** still need to be install so that the different application can access the GPUS.

  Exercise: Follow the different step of this tutorial to install the nvidia-driver on your VM.

- You also need to install **docker** itself and the **NVIDIA-Docker container toolkit**. The installation of this tools is perfectly described in their respecting websites: Docker and Nvidia-Docker. For an ease of use, all these installation commands has been resumed in the *docker/bash_docker_install.sh* file.

  Exercise: Send the *docker_utils* folder in the virtual machine you have just set up with the following command

  ```
  gcloud compute scp --recurse --zone #YOUR_ZONE #PATH_TO_FOLDER
      instance-2:~  --ssh-key-file #PATH_TO_SSH_KEY
  ```

  On the VM, move to the *docker_utils* you have just sent and run the script:

  ```
  sh bash_docker_install.sh
  ```

  Your virtual machine is now ready to be used.

3.3 Dockerfile

To run the code we use in the previous function we need an image that recreate most of the tools that was already installed in the *Deep Learning's VM* instance. All this requirements are defined in a ***Dockerfile***.
We can either build an image *from scratch* and define all the layer or build an image from *an existing dockerfile* and add the layer we want. We will use the second solution and build an image from one of the official **dockerfile's Tensorflow image.** These images are available on this repository. Go on this pages and open the *gpu.Dockerfile* image.

Question: What is the operating system on which the image is build? What is the python version? What is the tensorflow version?

In the *GoogleCloud/docker_utils* folder there is a docker file. Open it.

Question: On which image this docker file is based? On line 4 and 5 there is some command to set up the local time of the server. What are doing line 7 and 9?

This dockerfile is the one we will use to build the docker image on the VM. This files is already on the VM as you send the all *docker_utils* folder on the VM at the previous step.

3.4 Image

Let us build the image from the Dockerfile on the VM.

Exercise: Complete the following command in order to build and image from your dockerfile. Use the slide of the course to help you. Run it from your VM. What do you observe at the execution of this command?

```
sudo docker build -t #ACOMPLETER -f #ACOMPLETER #ACOMPLETER
```

Once the image is built (it takes a few minutes), run the following command that allow you to list the image available on you VM.

```
sudo docker image ls
```

Question How many images are on the VM? To what they refer to?

3.5 Container

You've just build your own docker *image*. Next step is to launch a *container* from this *image*. A *container* is an instantiation of an *image*. You can have different *container* from an *image*.

3.5.1 Interactive container

Exercise: Complete the following command in order to launch a container in an interactive mode and without volume. Use the slide of the course to help you.

```
sudo docker run ACOMPLETER --name ACOMPLETER ACOMPLETER
```

You are now inside a docker container. Within it, open a python console. Try to import *tensorflow* to check it is correclty installed. Run the following command to check that *tensorflow* has access to GPU.

```
from tensorflow.python.client import device_lib
MODE = "GPU" if "GPU" in [k.device_type for k in device_lib.list_local_devices()
] else "CPU"
```

It looks like we have all the tools we need to run our code. Except that we still don't have acces to the data. We will need to build a container with a volume to fix this problem. Before that, we will see how to quit and re-launch a container.

Exercise:

- Leave the python console with the following command: *ctrl+d*. And then leave the container with the same command: *ctrl+d*.
- List all the container with the following command :

  ```
  sudo docker container ls
  ```

  What do you see? Run the same command with the *-a* option. What do you see?
- When leaving a container, it automatically close. To re-use it in interactive mode you need to first start it.

  ```
  sudo docker start #container_id
  ```

- Check that the status of the container as change (with the *ls* command). The *attach* yourself to the container with the following command :

  ```
  sudo docker attach #container_it
  ```

- Leave again the container and remove it with the following command/

  ```
  sudo nvidia-docker rm #container_it
  ```

  Check that container has been well removed (with the *ls* command).

3.5.2 Mounted Volume

Let us see know how to access the data from a container. For that we need to *mount* a volume inside the container.

Exercise: Complete the following command in order to run an interactive container that the *docker_utils folder* will be accessible within the *root* folder which.

```
sudo docker run -it --name #ACOMPLETER
-v #ACOMPLETER:/root/#ACOMPLETER #ACOMPLETER
```

Check that the data can be accessed from the container. Now, from the container, create a *tmp* folder within the *docker_utils* folder. Quit the container and open the *docker_utils* folder. What do you see?

You're now able to create a container with all the tools and the data you need!

3.5.3 Background Mode

Last thing we need do know is how to launch a container in a *background* mode. Running a container in an *interactive* mode is usefull to developp and understant how a container work. But it's better to be able to launch some code within the countainer without being into it.

Exercise: Complete the following command in order to run an background container that the *docker_utils folder* will be accessible within the *root* folder which.

```
sudo docker run ACOMPLETER --name ACOMPLETER
-v ACOMPLETER
```

Check your container is launched by executing the following command:

```
sudo docker container ls
```

What do you see?

Now that, the container is launched you can run some code within it using the *exec* command. First, list the folder within the root directory with the following command:

```
sudo docker exec #CONTAINER_NAME ls /root/
```

Exercise: In your container that is running in *background* mode, create a folder within the container within the container's volume with the *exec* command. Check that this folder can be accesed in your VM. Create now a folder in your VM and check in can be accessed in your background container. After that, stop and remove the container and check that it is well removed.

3.6 Code Execution.

It's really easy to use docker container trough *gcloud command*.When you run *gcloud compute ssh* you just have to add the *–container* option follow by the name of the container in which you want the code to be executed.
For example the same command that is display in Section 2.2, can be executed in a container this way:

```
gcloud compute ssh nom_instance --zone europe-west-1 --ssh-key-file DirKeyFile
    --container NameContainer
    --command 'python3 learning.py --data_dir DirData -- results_dir DirResults
                                    --epochs 10 --batch_size 128'
```

Hence to adapt the *main.py* script so that it use the container you have to add the following steps :

- Launch a background container,
- Add the *–container* option for option allowing to run python code ,
- Stop and removed the container.

The *main_docker.py* file has been written in order to be adapted with this steps.

3.7 Exercice

- Within the *main_docker.py* file, check that the ProjectManager class is set to only print the command. Complete the various variable *TO COMPLETE* Run the script on your local terminal and check command that it would make run.
- Once everything is clear for you, change the execute_command option and execute the *main_docker.py* script and interprete the differents output of the terminal.