# On the Usage of GPU

Juan Maroñas

December 2020

## Contents

## 1 Installed libraries

Before proceeding, I recommend going over section 2 (Appendix 1) for an insight on compilation and linking. You need to know how linux work in order to get many software properly working. As a general rule, first look for the library in the software that we want to use and if it is not then check if it is a Unix library that needs to be installed. You can consider the following steps:

- If the library is not found and it is a generic Linux library (I have previously searched the internet for that), the administrators are contacted to install it.
- If the library is not found but it is already installed, we can find its directory by executing grep on the path to the software we want to use, and then add the route to the library to our environment variable. Check section 2 if you do not have idea of what I am talking about.

### 1.1 NVIDIA LIBRARIES

The machines will have the Nvidia CUDA library and the cuDNN libraries for acceleration in Deep Learning. The routes to the directories of these libraries are in:

- CUDA X: "/usr/local/cuda-X/"
- cuDNN : "/usr/local/cudnn/cuda-X"

Where cuda-X represents the desired cuda version (example cuda-8). Each cudnn library (specific for each cuda) is installed under the cuDNN directory. For example to use cudnn7 for cuda 8 you must go to /usr/local/cudnn/cuda8/ and to use it for cuda9 you must go to /usr/local/cudnn/cuda9/

The following table shows the versions of CUDA and cuDNN available **Important note. Since ubuntu20 only cuda11.1 is available. I keep cudnn for older cuda10 versions but that will be probably removed soon. If there is a need to have cuda10 in the machines let me know and will reinstall through apt-get, although I prefer to keep cuda installed only through the instalers provided by Nvidia**:

| **cuda-10.0** | **cuda-10.1** | **cuda-10.2** | **cuda-11.1** |
|---|---|---|---|
| 7.3.0 | - | - | - |
| 7.3.1 | - | - | - |
| 7.4.1 | - | - | - |
| 7.6.5 | 7.6.5 | 7.6.5 | - |
| - | - | - | 8.0.4 |

Go inside the cuDNN directories of the cuda version you want to use to check for the name of that version. For example to use cudnn7 version 3.1 in cuda 10 you run:

- ls /usr/local/cudnn/cuda10/

And the output will show: dnn7.3.0 dnn7.3.1 dnn7.4.1 dnn7.6.5

Obviously dnn7.3.1 is your choice, so finally your desired library is under:

- /usr/local/cudnn/cuda10/dnn7.3.1/

Within these folders we find the directory with the binaries and the directories with the libraries. Depending on how you compile your software we may require some steps or others. And depending how your software (for example Tensorflow) is made. We provide a further review in section 2 (Appendix 1) on how UNIX manage compilation and linking. In general and for cuda 10 (note that it is the same for the rest):

- binaries (compiler...): "/usr/local/cuda-10.0/bin/"
- libraries: "/usr/local/cuda-10.0/lib64/"
- headers: "/usr/local/cuda-10.0/include/"

In the case of cudnn:

- libraries: "/usr/local/cudnn/cuda10/dnn7.3.1/lib64/"
- headers: "/usr/local/cudnn/cuda10/dnn7.3.1/include/"

In libraries we find dynamic and static versions of them.

## 1.2 Software and Unix libraries

See Github README file for details.

## 1.3 Nvidia Docker

We have installed nvidia-docker. If you want to use it contact the administrator to be added to the docker group.

## 1.4 Conda and not pip

Pip is not supported anymore, just conda.

Conda is installed in /usr/local/anaconda3/ . From now own we will refer to this path as conda_path.

There are several ways to create condas enviroments. Depending on how you do it it will install it to .local, to the root directory of conda.

To run conda you must type /usr/local/anaconda3/bin/conda.

If you want to add conda to your .bashrc so that conda is run automatically when you type ‘conda‘ then you have to run

```
/usr/local/anaconda3/bin/conda init
```

With this you can execute any of the commands below withouth having to use the absolute path /usr/local/anaconda3/bin and just use conda.

To create a python enviroment using conda type the following:

/usr/local/anaconda3/bin/conda create -y –no-default-packages –prefix my_enviroment python=3.7

This will create a conda enviroment using python 3.7 named my_enviroment.

To activate the enviroment type:

source /usr/local/anaconda3/bin/activate my_enviroment

Then you can install whatever package you want:

pip install tensorflow_gpu

To deactivate it type:

source /usr/local/anaconda3/bin/deactivate

**NOTE:** Examples on how to use conda to install software such as PyTorch or Tensorflow can be found in this repository.

**NOTE:** You might need to configure correctly the enviroment variables so that CUDA CUDNN or other libraries are found. See the appendix.

## 1.5 Merging Python Virtual Enviroments

Some tips to merge and use two different python enviroments. There are two solutions, one unix based and one python based.

**This is a python based solution.** We create in our home, lets suppose /home/prhlt/, an enviroment variable (it could be created with conda or just installed directly in .local folder):

- cd /home/prhlt/
- virtualenv -p python2.7 myenviroment
- source myenviroment/bin/activate
- pip install scipy
- deactivate

We create an enviroment with scipy installed. Now, to use both environments it is as easy, just follow the next steps. We activate our TF enviroment (follow TF steps):

- source /usr/local/anaconda3/bin/activate /usr/local/tensorflow/python2.7/1.4.1

Then, in your python script:

```
#My new state of the art model
import tensorflow
import scipy #if you do it here it will raise error
import sys
sys.path.extend(["/home/prhlt/myenviroment/lib/python2.7/site-packages/"
,"/usr/local/keras/python2.7/lastversion/lib/python2.7/site-packages/"])

import scipy #after adding the myenviroment lib path everything works perfectly
import keras
import tensorflow
```

**A unix based solution:** There is a unix based solution (thanks to Mario for finding it). Suppose we create an enviroment to install scipy or whatever:

- export LD_LIBRARY_PATH=LD_LIBRARY_PATH:/usr/local/cuda-9.0/lib64/
- export LD_LIBRARY_PATH=/usr/local/cudnn/cuda8/dnn6/lib64:$LD_LIBRARY_PATH # do this if necessary
- virtualenv -p /usr/bin/python3.5 myapp
- you activate virtualenv and install whatever you need (pip install scipy)
- deactivate

Now after you have it just type:

- realpath /usr/local/tensorflow/python3.5/1.4.1/lib/python3.5/site-packages > myapp/lib/python3.5/site-packages/base_venv.pth
- realpath /usr/local/keras/python3.5/lastversion/lib/python3.5/site-packages >> myapp/lib/python3.5/site-packages/base_venv.pth

And now you can use keras and your enviroment with tensorflow without having to modify your .py files. **Just activate** you enviroment myapp and import keras and tensorflow.

This can also be done with conda enviroments, or with any path in your system. For instance with conda enviroments we can do:

```
$conda_path/bin/create --name /tmp/env_name
realpath /usr/local/tensorflow/python3.5/1.4.1/lib/python3.5/site-packages >
/tmp/env_name/lib/$python_version/site-packages/base_env.pth
source $conda_path/bin/activate env_name
```

## 1.6 Launch as fast as Joan Andreu

If you want to launch in several machines at the same time from your computer there is an easy way to do it. The trick is to use dsh command. I put here the tutorial provided by Lorenzo:
First install dsh

```
sudo apt-get install dsh
```

The configure it (for secure connection):

```
sudo sed -i "s:remoteshell =rsh:remoteshell =ssh:g" /etc/dsh/dsh.conf
```

Create the machines.lst file. There are two options, first we can add the machines to the default file /etc/dsh/machines.list or create a new one and use the option -f to redirect dsh to it. The file is very simple: a txt file where each line is a machine in the format username@machine address as in a normal ssh. Example:

```
lquirosd@150.222.222.44
lquirosd@150.222.222.44
lquirosd@150.222.222.45
```

Password less login. Because we are using ssh a password is required for each time you login to a machine. To prevent that we can enable password-less ssh communication from the host where you've installed dsh:

1. Run ssh-keygen -t rsa on the host machine.
2. this command will generate a id rsa.pub file. Copy the contents of this file to the $USER/.ssh/authorized keys file on each of the machines listed on the machines.lst file.
3. run ssh to each machine to ensute it works as expected.

Execute commands in all the machines. To execute any command in the machines, just use the option -c in the command. For example, to check the memory usage in some GPU we can use the command

```
 nvidia-smi --query-gpu=memory.total,memory.used,memory.free --format=csv
```

, to run it on all the machines:

```
dsh −f <pointer to machines file> −M −c "nvidia−smi
−−query−gpu=memory.total,memory.used,memory.free
−−format=csv"
```

## 2 Appendix 1. An Insight on Linux compilation and link stages

There are two basic environment variables for the use of machines: PATH y LD_LIBRARY_PATH.

As we know, the systems based on linux have certain directories where the executables of the system are placed by default, and therefore, is where the operating system looks for said binaries. For example we have: /bin, /sbin, /usr/local/bin, /usr/local/sbin... When we want the operating system to look for binaries in other folders we must add it to the path variable. For example, if I want the Nvidia compiler to be detected automatically, We can do:

export PATH=$PATH:/usr/local/cuda-8.0/bin/

So when we run a program and it gives us the typical error: "Can not found NVCC" is because it is not found in $PATH.

We also know that software is usually supplied by libraries and that for several reasons dynamic linkage is advantageous. So the dynamic linking phase has two stages: one is to find the libraries while compiling and another is to find the libraries when it is executed (this is done by the linker /usr/bin/ld).

Again the linux based systems have default directories where the linker finds the libraries that we install: /lib, /usr/lib, /lib64. In the compilation phase there are two ways to tell the linker where to find possible libraries.

One is through the flag -L and another is through the flag -rpath. The difference is very simple: -L indicates only where the libraries are but it does not save the directory where they are in the binary, while -rpath saves it. Therefore to a software compiled with -L we must indicate when we execute (in the runtime phase) where the libraries are and to a software compiled with -rpath no, since having saved the directory the linker automatically goes there to search.

For that reason, the configuration of our enviroments depends on how our compiled software or our precompiled software (TF, torch...) is done. We use variable LD_LIBRARY_PATH to indicate the system where to search for other libraries when they are not found in the default directories. For example, for those softwares that make use of cudnn we must place here where the specific version of cudnn is. Even if our software is already compiled (like tensorflow) we must tell the linker where to find the libraries. Note that compiling with rpath does not make sense when distributing software because we may place cudnn wherever we want. As example place (to use cudnn6 for cuda8):

```
export LD_LIBRARY_PATH=/usr/local/cudnn/cuda8/dnn6/lib64/
```

Many of the linking stage can be saved by using ldfconfig. Example:

```
sudo echo ”/path_to_library” >> /etc/ld.so.conf; sudo ldconfig
```

## 3 Acknowledgment

Thanks to Mario Parreño for testing software and to Lorenzo Quiros and Mario for helpfull discussion and recommendations. Also thanks to Miguel and Jose for support in many aspects.