CC-MEI 2024: Introduction to Deep Learning Programming in the Cloud

Jordi Torres

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Our world has changed, and it has changed forever due to the impact of the recent, rapid growth of artificial intelligence (AI) technology. And what has caused this AI explosion? Undoubtedly, supercomputers provided through the Cloud are one of the critical pillars of AI's progress, speeding up the training of algorithms. The reason is that deep learning (DL) algorithms are highly parallel, making them not only conducive to taking advantage of GPU acceleration but also scalable to multiple GPUs and nodes. The following hands-on exercises will introduce you to using the Cloud for training a DL code and learn how to program a parallel supercomputer for training a real artificial intelligent problem.

Important note: Beware of "copy & paste" as symbols such as commas or dashes can be incorrectly copied from this document to the console. If a command/code doesn't work correctly, write it directly!

Hands-on 1-5

Hands-on Reports

Write a report for each exercise that includes all the tasks, detailing the steps that were taken, the code used, and the results. Once finished, generate a PDF version and submit it to the corresponding inbox at the "racó" intranet.

1. IA in the Cloud

In this section, we will learn how to train a neural network using a Jupyter Notebook in the Google Cloud.

Google Colab

Google Colab is a Jupyter notebook based runtime environment that allows you to run code on the Google Cloud that can be accessed remotely through a browser. Google Colab supports accelerators, both GPU and TPU instances.

Task 1.1

Open a browser of your choice, go to colab.research.google.com, and sign in using your Google account. Click on a new notebook to create a new runtime instance or download a GitHub file. To create a GPU/TPU-enabled runtime, click on the "runtime" label in the toolbar menu, click "Change runtime type", and then select GPU or TPU.

Jupyter notebook

Training a DL model in Colab is very easy because it is not required to set up a custom runtime environment for the user (equivalent to *module load* in Marenustrum); it's all handled by him. For example, train a basic DL model to recognize handwritten digits (the model is basic, categorizes images as numbers, and recognizes them) trained on the MNIST dataset. The code uses Tensorflow library, and the data can be loaded from the standard Keras dataset archive. Execute by your own the following code to understand the details explained by the teacher in the class:

https://github.com/jorditorresBCN/HPC-DL/blob/main/GettingStarted_DL.ipynb

```
import tensorflow as tf
from tensorflow import keras

import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 2s 0us/step
```

```
import matplotlib.pyplot as plt
plt.imshow(x_train[8], cmap=plt.cm.binary)
```

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

x_train /= 255
x_test /= 255
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
```

```
from tensorflow.keras.utils import to_categorical
```

```
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(10, activation='sigmoid', input_shape=(784,)))
model.add(Dense(10, activation='softmax'))
model.summary()
```

```
model.compile(loss="categorical_crossentropy",
              optimizer="sgd",
              metrics = ['accuracy'])
```

```
model.fit(x_train, y_train, epochs=5)
```

```
test_loss, test_acc = model.evaluate(x_test, y_test)
print('Test accuracy:', test_acc)
```

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
                range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Observación')
    plt.xlabel('Predicción')

from collections import Counter
from sklearn.metrics import confusion_matrix
import itertools

# Predict the values from the validation dataset
Y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred, axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test, axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
```

```
plot_confusion_matrix(confusion_mtx, classes = range(10))
```

```
x_test_old = x_test.reshape(10000, 28,28)
plt.imshow(x_test_old[11], cmap=plt.cm.binary)
predictions = model.predict(x_test)
np.argmax(predictions[11])
print(predictions[11])
```

Task 1.2

In your Colab, download the GitHub file `https://github.com/jorditorresBCN/HPC-DL/blob/main/GettingStarted_DL.ipynb` . Define, train, and evaluate a new model that improves the Accuracy obtained by the basic model (teacher's model). (a) Explain the improvement applied: More epochs? More layers? More neurons per layer? Print and analyze the confusion matrix. (b) Compare the optimizers SGD and Adam for your neural networks. Describe the results of this task in the lab report.

Convolutional Neural Network

Below, we describe the code that implements the solution for the previous image classification problem using convolutional neural networks. You can find this code in the same .ipynb file as the previous one. Feel free to use the same Colab notebook to analyze this code while executing it. The neural network-based solution is the one we will employ in future tasks.

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten

model = Sequential()
model.add(Conv2D(32, (5, 5), activation='relu', input_shape=(28, 28,
1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))

from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
```

```
model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
from tensorflow.keras.utils import to_categorical

#mnist = tf.keras.datasets.mnist(train_images, train_labels),
(test_images, test_labels) =
mnist.load_data(path='/gpfs/projects/nct00/nct00008/basics-
utils/mnist.npz')

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()

print (train_images.shape)
print (train_labels.shape)
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
model.fit(train_images, train_labels, batch_size=100, epochs=5,
verbose=1)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

Task 1.3

Define, train, and evaluate a new model that improves the Accuracy obtained by the basic convolutional model (teacher's model). Explain the improvement applied. Describe the results of doing this task in the lab report.

2. Supercomputers for IA

Marenostrum 4 is divided into two primary partitions: (1) A general-purpose partition based on Intel processors (named MN4) and (2) an accelerated partition based on IBM Power processors and Nvidia accelerators (named CTE-power).

Your BSC credentials as a student will allow you to access both partitions, but you need to log in to different login nodes. You can access both BSC supercomputing resources from your local terminal using the secure shell (ssh). Look at section 8 to see the options to simplify the consecutive connections. Once you are logged, either in MN4 or CTE-power, you will find yourself in an interactive node where you can execute interactive jobs (small jobs) and submit execution jobs to the rest of the nodes of the partition chosen using the SLURM batch scheduling system. In this section, we will introduce the Martenostrum accelerated partition.

CTE-power characteristics (Marenostrum accelerated partition)

Hardware and OS

CTE-POWER is a cluster based on IBM Power9 processors, with a Linux Operating System and an Infiniband interconnection network.It has two login nodes and 52 compute nodes, each of them with the following configuration:

- 2 x IBM Power9 8335-GTH @ 2.4GHz (3.0GHz on turbo, 20 cores, and 4 threads/core, total 160 threads per node)
- 512GB of main memory distributed in 16 dimms x 32GB @ 2666MHz
- 2 x SSD 1.9TB as local storage
- 2 x 3.2TB NVME
- 4 x GPU NVIDIA V100 (Volta) with 16GB HBM2.
- Single Port Mellanox EDR
- GPFS via one fiber link 10 GBit

The operating system is the Red Hat Enterprise Linux Server 7.5 alternative. Detailed information about CTE-power can be found in the User's Guide[1]. More details of its characteristics can be found in the CTE-POWER user's guide and the information on the manufacturer of the AC922 servers[2].

File system

The Marenostrum file system is based on the IBM General Parallel File System (GPFS), a high-performance shared-disk file system providing fast, reliable data access from all cluster nodes to a global filesystem.

[1] https://www.bsc.es/user-support/power.php

[2] https://www.redbooks.ibm.com/redpapers/pdfs/redp5472.pdf

GPFS allows parallel applications simultaneous access to a set of files (even a single file) from any node with the GPFS file system mounted while providing high control over all file system operations. In addition, GPFS can read or write large blocks of data in a single I/O operation, thereby minimizing overhead. GPFS can be accessed from all the nodes and Data Transfer Machine (dt01.bsc.es). The available GPFS directories and file systems are:

- `/gpfs/home`: After login, it is the default work area where users can save source codes, scripts, and other personal data. The space quota is individual (with a relatively lessened capacity). It is not recommended to run jobs; please run your jobs on your group's /gpfs/projects or /gpfs/scratch instead. An incremental backup will be performed daily only for /gpfs/home.
- `/gpfs/projects`: It's intended for data sharing between users of the same group or project. All members of the group share the space quota.
- `/gpfs/scratch`: Each user has their directory under this partition, for example, to store temporary job files during execution. All members of the group share the space quota.
- `/apps`: Over this filesystem reside applications and libraries already installed on the machine for everyday use. Users cannot write to it.

CTE-POWER computer server node (Image from bsc.es)

Connecting to CTE-power

Once you have a login and its associated password, you can get into the cluster using Secure Shell (ssh) through the login nodes

- plogin1.bsc.es

- plogin2.bsc.es

You must use Secure Shell (ssh) tool to login in or transfer files into the cluster to enable secure logins over an insecure network. (BSC does not accept incoming connections from protocols like telnet, ftp, rlogin, rcp, or rsh commands).

Once connected to the machine, you will receive a UNIX shell prompt and usually be in your home ($HOME) directory. In this hands-on, we assume that the student is not new to UNIX environments.

To change your password, you have to login to the Data Transfer machine:

```
mylaptop$> ssh {username}@dt01.bsc.es
```

With the same 'username' and 'password' as in the cluster. Then, you have to use the 'passwd' command. We encourage you to change your password because everybody knows it :-).

The login nodes in both Marenostrum 4 partitions are the only nodes accessible from the outside, but no connections are allowed from the cluster to the outside world for security reasons. All file transfers from/to the outside must be executed from your local machine and not within the cluster:

To copy files or directories from MN4 to an external machine:

```
mylaptop$> scp -r {username}@dt01.bsc.es:"MN4_SOURCE_dir"
"mylaptop_DEST_dir"
```

To copy files or directories from an external machine to MN4:

```
mylaptop$> scp -r "mylaptop_SOURCE_dir"
{username}@dt01.bsc.es:"MN4_DEST_dir"
```

Appendix: Review of Linux basics

Below is a brief manual of some main Linux commands for beginners. These commands help you navigate, manage files, and perform everyday tasks in a Linux terminal.

List files and directories in the current directory:

```
ls
```

Print the current working directory (i.e., your current location in the file system):

```
pwd
```

Change directory. Use it to navigate to a different directory:

```
cd <directory_path>
```

Create a new directory:

```
mkdir <directory_name>
```

Remove an empty directory:

```
rmdir <directory_name>
```

Remove files or directories. Be cautious as it is a permanent action:

```
rm <file_name>
rm -r <directory_name>
# Use -r for recursive deletion of directories and their contents.
```

Copy files or directories from one location to another.

```
cp <source_file> <destination_path>
cp -r <source_directory> <destination_path>
# Use -r for recursive copying of directories and their contents.
```

Move or rename files or directories:

```
mv <source> <destination>
```

Display the beginning lines of a file:

```
head <file_name>
```

Display the last lines of a file:

```
tail <file_name>
```

Search for a pattern in a file or output:

```
grep <pattern> <file_name>
```

Access the manual pages for a command to get more information:

```
man <command_name>
```

Display information about active processes:

```
ps aux
```

Terminate a process using its PID (Process ID):

```
kill <PID>
```

Task 2.1

2.1 Change your password*.

2.2 Send an email to jordi.torres@bsc.es including the following:

- **As a subject: Your Marenostrum account number assigned that you will use for CC.MEI labs**
- **Name and contact email of all group members.**

2.3 Login to MN4 using one of the two login nodes and check if you can use the Linux command line (e.g., List files and directories in the current directory). If any problem appears, please contact the teacher.

(*) The new password will become effective a few minutes after the change.

HPC Software stack

It is important to remember that before executing a Deep Learning application (in TensorFlow), it is required to load all the packages that build the application's software stack environment (in Colab, it is not necessary to set up a custom runtime environment, it's all handled previously by Google). The hands-on exercises in this course use the BSC's CTE-Power cluster at BSC.

Software Environment

Typically, users initialize their environment when they log in by setting environment information for every application they will reference during the session. The *Environment Modules*[3] package is a tool that simplifies shell initialization and lets users easily modify their environment during the session with modulefiles. Each modulefile contains the information needed to configure the shell for an application or a compilation. Modules can be loaded and unloaded dynamically in a clean fashion. All popular shells are supported, including bash, ksh, zsh, sh, csh, tcsh, as well as some scripting languages such as perl.

Modules can be invoked in two ways: by name alone or by name and version. Invoking them by name implies loading the default module version. This is usually the most recent version tested to be stable (recommended) or the only version available.

```
module load intel
```

Invoking by version loads the version specified by the application. As of this writing, the previous command and the following one load the same module.

```
module load intel/2017.4
```

The most important commands for modules are these:

- *module list* shows all the loaded modules
- *module avail* shows all the modules the user is able to load
- *module purge* removes all the loaded modules
- *module load* <*modulename*> loads the necessary environment variables for the selected modulefile
- *module unload* <*modulename*> removes all environment changes made by module load command
- *module switch* <*oldmodule*> \<*newmodule*> unloads the first module and loads the second module

You can run "module help" anytime to check the command's usage and options.

The latest Intel compilers provide the best possible optimizations for the Xeon Platinum architecture of MN4. For this reason, by default, when starting a new session on the system,

[3] http://modules.sourceforge.net/)

the basic modules for the Intel suite will be automatically loaded. That is the compilers (intel/2017.4), the Intel MPI software stack (impi/2017.4) and the math kernel libraries MKL (mkl/2017.4) in their latest versions.

The default GCC provided by the system is version 4.8.5. Different versions can be loaded via the provided modules to better support new and old hardware features. For example, in MareNostrum, you can find GCC 9.2.0 loading it as:

```
module load gcc/9.2.0
```

Load required modules for deep learning applications

At the CTE-POWER supercomputer, it is done through modules, which can be done with the command module load before running the corresponding .py code.

In our case study, we will load the following modules that include the required libraries:

```
module load gcc/8.3.0 cuda/10.2 cudnn/7.6.4 nccl/2.4.8
tensorrt/6.0.1 openmpi/4.0.1 atlas/3.10.3 scalapack/2.0.2 fftw/3.3.8
szip/2.1.1 ffmpeg/4.2.1 opencv/4.1.1 python/3.7.4_ML
```

python/3.7.4_ML module contains TensorFlow packages.

Basics in training a DL program using a supercomputer

From any login node (plogin1.bsc.es, plogin2.bsc.es), you can access the rest through "ssh". These nodes are intended for editing, compiling, preparing, and submitting batch executions.

If some job run needs more CPU time than allowed (10 minutes), it must be done through the batch system.

Writing the .py program

You can save your `.ipnb` as a `.py`. The saved code (convolutional model) will look like the one below, being a code that can be executed as a Python code using the `python` command:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
print(tf.__version__)

from tensorflow.keras import Sequential
from tensorflow.keras.layers import ...
.
.
.

model = Sequential()
```

```
model.add(...)
.
.
.

from keras.utils import to_categorical
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()
train_images = ...
test_images = ...
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
model.compile(...)
model.fit(...)
test_loss, test_acc = model.evaluate(...)

print('Test accuracy:', test_acc)
```

Remember that the Marenostrum supercomputer does not have external access; therefore it cannot download MNIST data directly from the Internet. We have previously loaded them into the GPFS file system, specifically in the file `/gpfs/projects/nct00/nct00008/basics-utils/mnist.npz`. To be able to use them, we simply have to add the path in:

```
mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) =
mnist.load_data(path='/gpfs/projects/nct00/nct00008/basics-
utils/mnist.npz')
```

We will assume that the name of the file that contains the code will be MNIST.py, which we will use as a first case study to show how to launch programs in the CTE-POWER supercomputing.

Interactive execution of code

How to execute our code in the login node? (remember, we are still connected to the login node). Due to it being a regular Python code, we can use Python:

```
python MNIST.py
```

If we want to detach the standard outputs and the standard error messages, we can add this argument 2>err.txt:

```
python MNIST.py 2>err.txt
```

Redirecting the standard error allows us to see the result of the training that gives us the Keras by the standard output without the information related to the execution environment:

```
Epoch 1/5
600/600 [======] - 2s 3ms/step - loss: 0.9553 - accuracy: 0.7612
Epoch 2/5
600/600 [======] - 1s 2ms/step - loss: 0.2631 - accuracy: 0.9235
Epoch 3/5
600/600 [======] - 2s 3ms/step - loss: 0.1904 - accuracy: 0.9446
Epoch 4/5
600/600 [======] - 2s 3ms/step - loss: 0.1528 - accuracy: 0.9555
Epoch 5/5
600/600 [======] - 2s 3ms/step - loss: 0.1288 - accuracy: 0.9629
313/313 [======] - 1s 2ms/step - loss: 0.1096 - accuracy: 0.9671
Test accuracy: 0.9671000242233276
```

Task 2.2

(a) Write your MNIST classifier program with the file name MNIST.py. Include all code in the answers of the exercise survey, indicating the different parts:

- **Import required libraries,**
- **Load and Preprocess the Data,**
- **Define the Model,**
- **Define the Optimizer and the Loss function,**
- **Train the Model and finally**
- **Evaluate the Model.**

(b) Load required modules.

(c) launch your MNIST.py sequential program in one login node* of the CTE-POWER cluster (detach the standard outputs and the standard error messages).

(*) it can be done in the login node because it takes only a few seconds, not doing this for larger programs.
