FAQ “Machine learning in practice”

7 Nov 2017

*V. Codreanu,, D. Podăreanu*

*SURFsara*

How do I login? 2

With a whitelisted IP 2

Without a whitelisted IP 2

Working on Cartesius 3

Simple batch example 3

Simple interactive example 3

Job submission 4

Job monitoring 4

Filesystems 4

A simple Tensorflow MultiGPU example 5

Putting it all together 6

Running the command 10

Or in a jobscript 10

Lasagne-Theano example (with Jupyter) 11

Important notes:

* <https://userinfo.surfsara.nl/> contains useful information about our system(s), frequently asked questions and a few examples.
* <https://userinfo.surfsara.nl/systems/status> is the status page of all our systems.
* if you encounter difficulties and there is no similar scenario on the userinfo page ask your TA to contact us!

# How do I login?

You should already have your credentials in the email address used during registration. Please note that Cartesius is using a whitelist in order to allow access to the login node(s). If your IP is not on the list, you can still access the system through the “door node”. Your university might be whitelisted, however your home IP most likely isn’t. IP whitelisting is done by emailing [helpdesk@surfsara.nl](mailto:helpdesk@surfsara.nl). This information can be found at <https://userinfo.surfsara.nl/systems/cartesius/new-users>.

So, in practice, there are two ways to can connect to Cartesius:

### With a whitelisted IP

Linux/OSX: ssh [yourusername@cartesius.surfsara.nl](mailto:yourusername@cartesius.surfsara.nl)

Windows: In [Putty](http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) fill in under “Host Name (or IP address) “cartesius.surfsara.nl”

### Without a whitelisted IP

Linux/OSX: ssh [yourusername@doornode.surfsara.nl](mailto:yourusername@cartesius.surfsara.nl) after that select “Cartesius”

Windows: In [Putty](http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) fill in under “Host Name (or IP address) “doornode.surfsara.nl” after that select “Cartesius”.

If you want to forward ports from the remote (compute node) locally to your machine:

ssh -L 80:gcnX:5000 [myusername@cartesius.surfsara.nl](mailto:myusername@cartesius.surfsara.nl)

will forward port 5000 of gcnX to local port 80.

Alternatively see the Lasagne-Theano portion of the tutorial on how to create a reverse proxy.

Copying data should be done with

rsync

https://en.wikipedia.org/wiki/Rsync

# Working on Cartesius

On Cartesius we use SLURM to control resources: <https://slurm.schedmd.com/man_index.html>

Our userinfo pages contain more details about this:

* <https://userinfo.surfsara.nl/systems/shared/batch-howto>
* <https://userinfo.surfsara.nl/systems/cartesius/usage/batch-usage>

In general, a job consists of three parts:

* a line to tell which shell (bash, sh, tcsh) should execute the script
* one or more lines, starting with #SBATCH, which define the characteristics of the job (maximum duration, number of nodes, ...)
* one or more lines that are executed by the shell

### Simple batch example

So:

#!/bin/sh

#SBATCH --nodes=3

#SBATCH --partition=gpu\_short

#SBATCH --time=00:00:01 #1s job

#SBATCH --job-name=testjob

whoami

srun hostname

### Simple interactive example

Same functionality as in the previous example:

srun --nodes=3 --partition=gpu\_short --time=00:00:01 --job-name=testjob hostname

So in fact there are 2 ways to work on Cartesius: batch and interactive. One may see them as production and prototyping.

There are a few more interesting tricks to remember:

1. You can only allocate some resources if you want:

salloc --nodes=1 --partition=gpu\_short --time=00:01:00

watch squeue -u $(whoami)

1. A "wait job"

#!/bin/sh

#SBATCH --nodes=1

#SBATCH --partition=gpu\_short

#SBATCH --time=00:10:00

#SBATCH --job-name=testjob

sleep 10m

This allows you to get some resources and then login to the nodes directly (NOTE: Keep in mind your budget is **limited** and this is a **very inefficient way** to interact with Cartesius.)

### Job submission

sbatch jobscript

### Job monitoring

squeue -u username

Also, if you want to see the state of the system, you can do:

sinfo

Account info

accinfo

accuse -u username

### Filesystems

<https://userinfo.surfsara.nl/systems/cartesius/filesystems>

Main tricks:

Using scratch whenever possible

Using /dev/shm for **quick** access to data

# A simple Tensorflow MultiGPU example

git clone <https://github.com/aymericdamien/TensorFlow-Examples>

cd TensorFlow-Examples/examples/6\_MultiGPU

python multigpu\_basics.py

See the module environment

module avail

See the currently loaded modules

module list

Then, assuming a job like (saved with the name "myjob"):

#!/bin/sh

#SBATCH --nodes=1

#SBATCH --partition=gpu\_short

#SBATCH --time=00:01:00

#SBATCH --job-name=mgpuexample

module unload c/intel

module load cuda/8.0.44

module load cudnn/7.5-v5

module load python/2.7.11

module load mkl/17.0.1

module load c/intel/17.0.1

srun python multigpu\_basics.py

Submit it with:

sbatch myjob

Now you can monitor your job:

squeue -u $(whoami)

And once you get a node, you can login to it and then:

watch nvidia-smi

# Putting it all together

<https://github.com/tensorflow/tensorflow/tree/r0.10>

The git for the models is: <https://github.com/tensorflow/models>

Multi gpu training on the MNIST dataset

''' Multi-GPU Training Example.

Train a convolutional neural network on multiple GPU with TensorFlow.

This example is using TensorFlow layers, see 'convolutional\_network\_raw' example

for a raw TensorFlow implementation with variables.

This example is using the MNIST database of handwritten digits

(http://yann.lecun.com/exdb/mnist/)

Author: Aymeric Damien

Project: https://github.com/aymericdamien/TensorFlow-Examples/

'''

from \_\_future\_\_ import division, print\_function, absolute\_import

import numpy as np

import tensorflow as tf

import time

# Import MNIST data

from tensorflow.examples.tutorials.mnist import input\_data

mnist = input\_data.read\_data\_sets("/tmp/data/", one\_hot=True)

# Training Parameters

num\_gpus = 2

num\_steps = 200

learning\_rate = 0.001

batch\_size = 1024

display\_step = 10

# Network Parameters

num\_input = 784 # MNIST data input (img shape: 28\*28)

num\_classes = 10 # MNIST total classes (0-9 digits)

dropout = 0.75 # Dropout, probability to keep units

# Build a convolutional neural network

def conv\_net(x, n\_classes, dropout, reuse, is\_training):

# Define a scope for reusing the variables

with tf.variable\_scope('ConvNet', reuse=reuse):

# MNIST data input is a 1-D vector of 784 features (28\*28 pixels)

# Reshape to match picture format [Height x Width x Channel]

# Tensor input become 4-D: [Batch Size, Height, Width, Channel]

x = tf.reshape(x, shape=[-1, 28, 28, 1])

# Convolution Layer with 64 filters and a kernel size of 5

x = tf.layers.conv2d(x, 64, 5, activation=tf.nn.relu)

# Max Pooling (down-sampling) with strides of 2 and kernel size of 2

x = tf.layers.max\_pooling2d(x, 2, 2)

# Convolution Layer with 256 filters and a kernel size of 5

x = tf.layers.conv2d(x, 256, 3, activation=tf.nn.relu)

# Convolution Layer with 512 filters and a kernel size of 5

x = tf.layers.conv2d(x, 512, 3, activation=tf.nn.relu)

# Max Pooling (down-sampling) with strides of 2 and kernel size of 2

x = tf.layers.max\_pooling2d(x, 2, 2)

# Flatten the data to a 1-D vector for the fully connected layer

x = tf.contrib.layers.flatten(x)

# Fully connected layer (in contrib folder for now)

x = tf.layers.dense(x, 2048)

# Apply Dropout (if is\_training is False, dropout is not applied)

x = tf.layers.dropout(x, rate=dropout, training=is\_training)

# Fully connected layer (in contrib folder for now)

x = tf.layers.dense(x, 1024)

# Apply Dropout (if is\_training is False, dropout is not applied)

x = tf.layers.dropout(x, rate=dropout, training=is\_training)

# Output layer, class prediction

out = tf.layers.dense(x, n\_classes)

# Because 'softmax\_cross\_entropy\_with\_logits' loss already apply

# softmax, we only apply softmax to testing network

out = tf.nn.softmax(out) if not is\_training else out

return out

def average\_gradients(tower\_grads):

average\_grads = []

for grad\_and\_vars in zip(\*tower\_grads):

# Note that each grad\_and\_vars looks like the following:

# ((grad0\_gpu0, var0\_gpu0), ... , (grad0\_gpuN, var0\_gpuN))

grads = []

for g, \_ in grad\_and\_vars:

# Add 0 dimension to the gradients to represent the tower.

expanded\_g = tf.expand\_dims(g, 0)

# Append on a 'tower' dimension which we will average over below.

grads.append(expanded\_g)

# Average over the 'tower' dimension.

grad = tf.concat(grads, 0)

grad = tf.reduce\_mean(grad, 0)

# Keep in mind that the Variables are redundant because they are shared

# across towers. So .. we will just return the first tower's pointer to

# the Variable.

v = grad\_and\_vars[0][1]

grad\_and\_var = (grad, v)

average\_grads.append(grad\_and\_var)

return average\_grads

# Place all ops on CPU by default

with tf.device('/cpu:0'):

tower\_grads = []

reuse\_vars = False

# tf Graph input

X = tf.placeholder(tf.float32, [None, num\_input])

Y = tf.placeholder(tf.float32, [None, num\_classes])

# Loop over all GPUs and construct their own computation graph

for i in range(num\_gpus):

with tf.device('/gpu:%d' % i):

# Split data between GPUs

\_x = X[i \* batch\_size: (i+1) \* batch\_size]

\_y = Y[i \* batch\_size: (i+1) \* batch\_size]

# Because Dropout have different behavior at training and prediction time, we

# need to create 2 distinct computation graphs that share the same weights.

# Create a graph for training

logits\_train = conv\_net(\_x, num\_classes, dropout,

reuse=reuse\_vars, is\_training=True)

# Create another graph for testing that reuse the same weights

logits\_test = conv\_net(\_x, num\_classes, dropout,

reuse=True, is\_training=False)

# Define loss and optimizer (with train logits, for dropout to take effect)

loss\_op = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(

logits=logits\_train, labels=\_y))

optimizer = tf.train.AdamOptimizer(learning\_rate=learning\_rate)

grads = optimizer.compute\_gradients(loss\_op)

# Only first GPU compute accuracy

if i == 0:

# Evaluate model (with test logits, for dropout to be disabled)

correct\_pred = tf.equal(tf.argmax(logits\_test, 1), tf.argmax(\_y, 1))

accuracy = tf.reduce\_mean(tf.cast(correct\_pred, tf.float32))

reuse\_vars = True

tower\_grads.append(grads)

tower\_grads = average\_gradients(tower\_grads)

train\_op = optimizer.apply\_gradients(tower\_grads)

# Initialize the variables (i.e. assign their default value)

init = tf.global\_variables\_initializer()

# Start Training

with tf.Session() as sess:

# Run the initializer

sess.run(init)

# Keep training until reach max iterations

for step in range(1, num\_steps + 1):

# Get a batch for each GPU

batch\_x, batch\_y = mnist.train.next\_batch(batch\_size \* num\_gpus)

# Run optimization op (backprop)

ts = time.time()

sess.run(train\_op, feed\_dict={X: batch\_x, Y: batch\_y})

te = time.time() - ts

if step % display\_step == 0 or step == 1:

# Calculate batch loss and accuracy

loss, acc = sess.run([loss\_op, accuracy], feed\_dict={X: batch\_x,

Y: batch\_y})

print("Step " + str(step) + ": Minibatch Loss= " + \

"{:.4f}".format(loss) + ", Training Accuracy= " + \

"{:.3f}".format(acc) + ", %i Examples/sec" % int(len(batch\_x)/te))

step += 1

print("Optimization Finished!")

# Calculate accuracy for MNIST test images

print("Testing Accuracy:", \

np.mean([sess.run(accuracy, feed\_dict={X: mnist.test.images[i:i+batch\_size],

Y: mnist.test.labels[i:i+batch\_size]}) for i in range(0, len(mnist.test.images), batch\_size)]))

Running the command

python multigpu\_cnn.py

### Or in a jobscript

#!/bin/sh

#SBATCH -N 1

#SBATCH -p gpu

#SBATCH -t 1:00:00

module load cuda/8.0.44 cudnn/8.0-v5.1 python/2.7.11 gcc/4.9.2

srun -u python multigpu\_cnn.py

# Lasagne-Theano example (with Jupyter)

Jupyter requires port 5010 to be forwarded. For that we'll open a reverse tunnel.

You can do that by:

1. ssh -L 5010:localhost:5010 vis.cartesius.surfsara.nl
2. If you have for example the job script:

#!/bin/bash

#SBATCH -t 10:00

#SBATCH -N 1

#SBATCH -p gpu

module load python/2.7.11

module load cuda/7.0.28

module load cudnn/7.0-v3

module load gcc/4.9.2

ssh -o StrictHostKeyChecking=no -f -N -p 22 -R 5010:localhost:5010 int3

#$SLURM\_SUBMIT\_HOST

jupyter notebook --no-browser --port 5010

Notice the ssh line and the "-R" option. After you submit the job, you should be able to open a client on your local computer with (in a browser):

0.0.0.0:5010

In order to test the job script from above, we can clone the example code from:

<https://github.com/craffel/Lasagne-tutorial>

(git clone <https://github.com/craffel/Lasagne-tutorial)>