Lab 1: Predicting Runtimes on Rosie   **100/100** Points

3/14/2023

Attempt 1 ⌄

◯ **REVIEW FEEDBACK**
3/9/2023

Attempt 1 Score:
100/100

💬 View Feedback

Unlimited Attempts Allowed

⌄ **Details**

Pacing

This is an **individual** lab. PLEASE start it today so that your long jobs will have time to complete.

There is nothing to report on the first several sections of the lab. But you should work through it anyway, testing that the commands work for you and trying to understand what they do. You will be using these commands in the second half of the lab and running them yourself gives you important preparation for working the second half of the lab.

Be sure to use GlobalProtect if running this lab from off-campus.

Launching the course Singularity Image

Running an interactive shell without GPU

Once you are logged into Rose, run the command:

```
srun hostname
```

This will allocate a node and run the ~~~~~~~~ command on that node. The

‹ Previous		Next ›
(https://msoe.instructure.com/courses/13814/modules/items/546794)	Try Again	(https://msoe.instructure.com/courses/13814/modules/items/543831)

```
srun --nodelist=dh-node1 hostname
```

This should run the command on dh-node1 and print the name of this node to the console.

Now run

```
srun --pty bash
```

This will start an interactive bash shell on an allocated node. You should see your prompt change to reflect what node you are on. `--pty` stands for "pseudo terminal" or "psuedo typer" or something similar. You need to include it to make the prompts appear for your shell. Without it, nothing will print and you may think that you aren't into the node allocated by `srun` yet even though you are.

Try typing some commands like `ls` or `pwd` at the prompt. You are on the node, and can see and explore all the files on that node, including the shared filesystem and all the local files in the /scratch folder on that node. We wil be using the /scratch folder to reduce our dependency on the network between the shared drives and the nodes this quarter.

Quick self-test: Do you know what the difference between `/scratch`, `~/scratch`, `./scratch`, and just `scratch` are?

From the bash session on the node, start a singularity job:

```
singularity shell /data/cs3450/pytorch20.11.3.sif
```

Note that the prompt changes to show you are in the singularity container:

```
Singularity>
```

Quick self-test: What is a container? What does it mean for a command to run within a container vs. running straight on the node?

The container shell will look a lot like the regular shell. You can type `ls`, `pwd`, etc.

But within the shell, you get access to our Python instance. Try this:

use as our deep learning library all quarter long.

It's also good to see if you can import our textbook's library:

```
from d2l import torch as d2l
```

This should silently succeed. The d2l library works with three different libraries – mxnet, PyTorch, and Tensorflow. It provides a common Facade/Adapter that work with all three libraries and perform many tasks such as loading datasets for you. The unusual import is to make it easier to switch between different underlying libraries. We just use the torch library this quarter.

But, since we didn't request a GPU, none of the GPU commands you will use later in this lab will work yet (commands like `nvidia-smi` from the srun interactive prompt and Singularity prompt or `torch.cuda.current_device()` from the Python prompt). The next section will talk through how to request GPU resources.

Now type `exit()` to leave the Python shell and `exit` to leave the singularity session, and `exit` bash session on your node to the management node.

Running an interactive shell with a GPU

Let's repeat the above commands, but add GPU abilities.

To request a single GPU from SLURM when logging in interactively, use

```
srun --gpus=1 --pty bash
```

This will again assign you a node, perhaps different than your previous one. Since most of your files are on the shared network drive, it usually doesn't make a difference.

Now launch a singularity image with our Python libraries, this time requesting NVIDIA support (–nv) and binding the data and scratch directories (-B /data:/data -B /scratch:/scratch) so we can access that folder from within the image

Try Again

and allocate a GPU for programming purposes:

```
python
import torch
torch.cuda.current_device()
```

This should succeed and print 0, indicating you successfully allocated GPU index 0.

All of these commands can be put together into a single command. Again `exit()`/`exit` all the way back to the managment node before running this:

```
srun --gpus=1 singularity run --nv -B /data:/data -B /scratch:/scratch /data/cs3450/
pytorch20.11.3.sif python -c "import torch;print('Lab Prediting Runtime: Device:',to
rch.cuda.current_device())"
```

Key self-test: In the command above, can you point to the command that allocates a node for you? How about the option that allocates GPUs on that node for you?

Key self-test, continued: What does `pytorch20.11.3.sif` provide that you couldn't use without it? (If you aren't sure, review the steps you have completed so far – this question is answered earlier in this lab handout.)

Key self-test, continued: Which command allows you to specify a container, `srun` (SLURM run) or `singularity run`?

Key self-test, continued: How can we tell that a GPU was successfully allocated?

One of the benefits of running your jobs in a "single-line batch" mode like this is that you can copy-paste the whole command to your notes more easily, making it easier to reproduce your work later on.

In the sections that follow, we allocate a node and spin up the singularity image in a single step, just as in the command above. But you are welcome to do these two steps separately (like we did earlier in this lab) as you become more comfortable with these commands!

Adding 16 CPUs to the GPU

From a management node, run the command:

```
srun --partition=teaching  --pty --gpus=1 --cpus-per-gpu=16 singularity shell --nv -
B /data:/data -B /scratch:/scratch /data/cs3450/pytorch20.11.3.sif
```

(This used to include the option `--time=180`, but you don't neeed to set time limits on commands in this lab. The default of 24 hours is OK.)

~~If you do not specify a time limit (e..g, `--time=180` limits the runtime to three hours), it may default to 24 hours.~~

This will give you a bash prompt within the Singularity image:

```
Singularity>
```

Recall that a Singularity image is a lightweight environment running within the host operating system that gives you access to the host filesystem while allowing custom software to be installed as if it were installed by root. In this case, we have the pytorch library installed.

Within the Singularity image, run Python and within the Python interactive prompt, import torch:

```
python
import torch
torch.cuda.current_device()
```

This should succeed and print 0, indicating you successfully allocated GPU index 0.

Now, how many CPUs is `torch` set up to use?

```
torch.get_num_threads()
```

This should report 16. You can also set the number of threads, but that won't necessarily increase the number of corse `torch` can use.

Running a command on the DGX

In this lab, you will also train on the DGX, our largest node type. We have

with your instructor. Instead, run the single command `nvidia-smi` in a singularity instance without needing an interactive job:

```
srun --partition=dgx --gpus=1 --cpus-per-gpu=16 singularity run --nv -B /data:/data
-B /data:/scratch/data /data/cs3450/pytorch20.11.3.sif nvidia-smi
```

Or, if you want to run multiple commands within the same run on the DGX, you can place them in a shell script:

```
cat > example.sh
echo "Hello, world!"
nvidia-smi
ls
```

Again, type Ctl-D to finish writing the file. (Or use the editor of your choice.) Then, run the script, being sure to set the execute permission on the file first:

```
chmod u+x example.sh # Set the file to be executable by the current user (you).
srun --partition=dgx --gpus=1 --cpus-per-gpu=16 singularity run --nv -B /data:/data
-B /data:/scratch/data /data/cs3450/pytorch20.11.3.sif ./example.sh
```

Checking on the GPUs

When you run the command `nvidia-smi` on the `--partition=teaching` you can see that the GPU cards on these nodes are Tesla-T4s:

```
srun --partition=teaching --gpus=1 --cpus-per-gpu=16 singularity run --nv -B /data:/
data -B /scratch:/scratch /data/cs3450/pytorch20.11.3.sif nvidia-smi
```

This will output something like:

```
Sat Aug 29 11:26:23 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 11.0      |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:60:00.0 Off |                    0 |
| N/A   44C    P0    75W /  70W |   2955MiB / 15109MiB |     97%      Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
```

```
srun --partition=dgx --gpus=1 --cpus-per-gpu=16 singularity run --nv -B /data:/data
-B /data:/scratch/data /data/cs3450/pytorch20.11.3.sif nvidia-smi
```

shows that the GPUs on the DGX are V100's.

A note to rememer for later (you don't need to try this now): When you are on the T4 nodes, you can see how efficiently your code is using the GPUs with the `nvidia-smi` command. Once you start a job, you can ssh directly into that node by typing `ssh dh-node19` (for example). Then you can run your `nvidia-smi` command using the same GPU already allocated to you on that node.

Converting an srun command to an sbatch script

(This section is optional.)

Some of the commands take much longer to run. For these commands, using an sbatch script instead of srun will document your run configuration and ensure your job does not exit when you log out. (You could also use tmux to ensure your job doesn't exit, but that won't record your run configurations for you.)

Switching to sbatch is simple. Take your srun command and place each of the arguments in a script file in a comment that starts with SBATCH. For example, to convert this srun command that tests for GPUs with nvidia-smi:

```
srun --partition=dgx --gpus=1 --cpus-per-gpu=16 singularity run --nv -B /data:/data
-B /data:/scratch/data /data/cs3450/pytorch20.11.3.sif nvidia-smi
```

to a sbatch script, we would create this script, e.g., saved in `nvidia-smi-example.sh`:

```
#!/bin/bash

#SBATCH --partition=dgx
#SBATCH --gpus=1
#SBATCH --cpus-per-gpu=16
singularity run --nv -B /data:/data -B /data:/scratch/data /data/cs3450/pytorch20.1
1.3.sif nvidia-smi
```

Note that the singularity run command should be on a single line, including

Try Again

```
sbatch nvidia-smi-example.sh
```

Note that only the srun arguments are included as sbatch arguments. The singularity arguments are still included within an srun command within the file.

Usually your job will start immediately and you will see a message with the job number:

```
Submitted batch job 18285
```

You can follow the job with the `tail -f` command, modifying the number to match your job's:

```
tail -f slurm-18285.out
```

This will open tail in a mode where it will follow (-f) your running job, printing output to the screen as it is written to the file. When your program completes, tail will not notice this. Either way, when you are done looking at the file, you can use Ctl-C to exit tail. This does not cancel your slurm job – only the tail command!

To see if your program is still running, use the `squeue` command:

```
squeue
```

This will show all the jobs currently running. If your job is included in the list, you will know if it is running.

`squeue` is also useful for figuring out why your job hasn't started yet.

A comparison of srun and sbatch for a simple example

(again optional)

To help you sort out the various moving parts, here is a brief summary table comparing two configurations:

Try Again

An example of a complete command in config 1 is:

```
srun --pty singularity shell /data/cs3450/pytorch20.11.3.sif
```

You would interactively type, e.g., `ls` in the prompt for this one.

An example of a complete command in config 2 is:

```
sbatch example.sh
```

where example.sh contains:

```
#!/bin/bash
singularity run /data/cs3450/pytorch20.11.3.sif ls
```

You need to include the ls command on the same line as the singularity run command.

A debugging tip when using SBATCH

If you are using `sbatch`, be sure to copy and keep a different shell file for each experiment you run, even if the script is identical between runs. Not only does this give you documentation of what your shell files said for later notice, it also avoids a bug in which even though you edit the shell script, but the shell script and `gan.py` do not change on the node where the commands are run. This can be very confusing, because you are seing the view of the file from the management node. If you ever feel as if the running job is seeing an old version of a file rather than what you see, this could be why.

Again, the solution is simple: When you edit a shell script, copy it first before editing the copy.

A general debugging tip – to avoid bus errors

I believe bus errors are caused when your process runs out of memory – perhaps GPU memory? To avoid this, run `squeue` to see what nodes have jobs, and specify a node that is not currently running a job. From what students

‹ Previous
(https://msoe.instructure.com/courses /13814/modules/items/546794)

Try Again

Next ›
(https://msoe.instructure.com/courses /13814/modules/items/543831)

9 of 17   4/2/23, 11:52

The Rosie Visualizer and Dashboard

The **Rosie Dashboard's visualizer ⤷ (http://dashboard.hpc.msoe.edu /visualizer/)** also provides much interesting information about the nodes.

When using the visualizer, remember that your "GPU 0" may not be GPU 1 on the dashboard. Each student sees their GPU as "GPU 0" regardless of which GPU they were assigned.

Experiment

As a reminder, it is important that you complete the steps above even though you are not required to comment on them in your report.

Copy gan.py into your working directory:

```
cp /data/datasets/pokemon/gan.py ~/path/where/Im/working/
```

Start the report

Start a report as a Word document or some other formatted document that you can print to PDF. On the first page, include the course number, your name, the date, the lab number and lab name, e.g.:

```
CS3450
Phileas Fogg
1 Jan 2023
Lab 1: Predicting Runtimes
```

Include a title on your document and section headers in your document. You can use the lab number and name as a title for your document if you wish.

Also include an introduction section describing the lab in your own words. (You would be surprised how many students forget this! Yet it is the part of a report most likely to be read in industry and academia!)

The **NVIDIA V100 has a nominal double-precision throughput of 16.4 single-precision (float) TFLOPS** ⧉ **(https://images.nvidia.com/content /technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf)** (floating point operations per second). The **T4 has a throughput of 8.1 single-precision TFLOPS** ⧉ **(https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf)** . In your report, record what ratio you expect between the times these two systems will take to train a neural network. Briefly explain your reasoning. Assume that only one GPU is used on each node. I don't know of an authoritative source defining what counts as a FLOP, but **this Stack Overflow answer** ⧉ **(https://stackoverflow.com/a/329243/14260454)** has the ring of truth for me (Dr. Yoder).

Experimentally Determine the Runtime from Running One Epoch

Now, take the provided script, and confirm that the number of epochs is 1: (In deep learning, an epoch is one pass over the training data)

```
NUM_EPOCHS = 1
```

at the top of the script.

In the following sections, we will walk you through how to run one epoch of training on a T4 node with two CPUs, on a T4 node with eighteen CPUs, on a T4 node with a single T4 GPU, and on the DGX with a single V100 GPU. As you run each of these tests, record the total number of seconds to execute the script in each environment in a summary table. (You may also wish to record the exact output on timing reported at the end of the script.)

We will discuss how to do the runs on the T4 and DGX in the next few sections.

Running on a T4 Node with two CPUs

To run on a T4 node in interactive mode, you can use the same interactive command discussed earlier, but specifying the `cpus_per_task=2`

assigned to the same physical core.

```
srun --partition=teaching --pty --cpus-per-task=2 singularity shell -B /data:/data -
B /scratch:/scratch /data/cs3450/pytorch20.11.3.sif
```

Within your running singularity image, you can use the `nproc` command to determine how many CPUs you have and the `nvidia-smi` command should fail since your SLURM job doesn't have a GPU.

Make sure the gan.py file has the desired number of epochs in the constant near the top, as discussed earlier. `less gan.py` can be helpful for this.

From the directory containing the `gan.py` file, run this command:

```
python gan.py
```

Record the total running time in your table.

When you are done with the interactive session, type `exit()` to quit Python and `exit` to terminate the Singularity image.

Running on a T4 Node with sixteen hyperthreaded CPU cores

Each T4 node has four GPUs and 72 CPUs. So from a resource allocation perspective, it's OK to use sixteen CPUs per student – four students should still be able to use a node at one time.

To do a run with sixteen CPUs, simple replace the cpu argument with `--cpus-per-task=16` as you work through the previous section.

T4 Node with a GPU (and eight CPUs)

From a mgmt node (not a worker node or Singularity image), run the command:

```
srun --partition=teaching --pty --gpus=1 --cpus-per-gpu=8 singularity shell --nv -B
/data:/data -B /scratch:/scratch /data/cs3450/pytorch20.11.3.sif
```

(This used to include the option `--time=60`, but you don't neeed to set time limits on commands in this lab. The default of 24 hours is OK.)

Perhaps check that you have a GPU:

```
python gan.py
```

Again record the total running time in your table.

Running on a DGX Node with a GPU (and eight CPUs)

From the same folder in your base Rosie login (not a Singularity image), run the command:

```
srun --partition=dgx --gpus=1 --cpus-per-gpu=8 singularity run --nv -B /data:/data -
B /data:/scratch/data /data/cs3450/pytorch20.11.3.sif python gan.py
```

Also note that the local /scratch folder does not exist on the dgx nodes, so we are faking it by mapping the networked /data folder as if it were a /scratch/data folder within our image. But the files will actually be accessed through the network when running this command. This doesn't seem to slow it down too much in my test, but you are welcome to comment on this in your own experiments as well!

Predict the Runtime for 20 epochs

Now that you have actual runtimes in a variety of computing environments, predict the runtime to train through 20 epochs in each environment. Include your predictions in the same table in a new column.

Consider switching to a batch job

Look at your runtimes, and based on the runtime you expect, consider using a tool that allows you to disconnect like the `sbatch` tool discussed under the optional sections above. Some students instead use `tmux` or `nohup` with `srun`. Although we do not provide instruction on how to use these tools, they are also an option.

Experimentally Determine the Runtime from Running 20 Epochs

Try Again

printed at the end of the script.

~~This is important: Be sure to set the timeout to at least twice (perhaps four times) what you expect the network to take to run. For example, if you expect it to take 3600 seconds, set the timeout to at least 120 minutes. It's also OK to remove the timeout when you are running the job in batch mode.~~

All the jobs should finish within 24 hours, so if you predict any job will run for longer than 24 hours, please contact your instructor promptly. If, in very rare cases, your job times out at 24 hours, simply report this as a job that took more than 24 hours to run.

Discussion

Include a concluding discussion in your report. How closely did your predictions match the reality? Explain any differences. Also, be sure to discuss how the relative speed between the DGX and the T4 compared with your predictions.

And, as a quick checklist, confirm your report has: (See previous requirements for details)

- Title information and introduction
- Prediction of relative training speed on the DGX vs the T4
- Actual runtime for training for one epoch in each configuration
- Predicted time to train through 20 epochs based on the one epoch training time
- Actual runtime for training twenty epochs.

Just for Fun

Figuring out how to share Rosie better

The SE junior-year project **Rosie Dashboard** ⬘ **(https://dashboard.hpc.msoe.edu/)** is getting quite useful. Consider looking at its output alongside srun to get a sense for what external factors are impacting the runtime of your jobs. The more we can make GPUs the limiting

the capability to train and modify the structure of AlexNet here at MSOE.

Let me know if you are interested in exploring this. Training (and predicting training times for) AlexNet is very similar to this lab, it just takes a bit longer. A full train takes a few days using a single T4, though I think the bottleneck may not be the T4, since GPU utilization appears lower than it should be.

Looking at the images produced by the GAN

`gan.py` trains on Pokemon images and attempts to generate random novel Pokemon characters. You can see its work by downloading the imgs.npy file it writes to your machine, e.g., with

scp **username@rosie:path/within/your/home/folder/on/rosie/imgs.npy (mailto:username@rosie:path/within/your/home/folder/on/rosie/imgs.npy)** .

and, from a python prompt, running:

```
import numpy as np
from matplotlib import pyplot as plt
imgs =  np.load('imgs.npy')
plt.imshow(imgs)
```

You can also look at the training history, if you download the file `losses.npy`:

```
from matplotlib import pyplot as plt
import numpy as np
import os
losses = np.load('losses.npy')
plt.plot(losses[:,0],losses[:,1:3])
plt.legend(['discriminator','generator'])
plt.xlabel('epoch')
plt.ylabel('loss')
```

Deliverables

Submit your report in PDF form.

❮ Previous

(https://msoe.instructure.com/courses /13814/modules/items/546794)

Try Again

Next ❯

(https://msoe.instructure.com/courses /13814/modules/items/543831)

15 of 17   4/2/23, 11:52

Try Again

Previous

(https://msoe.instructure.com/courses
/13814/modules/items/546794)

Try Again

Next ›

(https://msoe.instructure.com/courses
/13814/modules/items/543831)