How to use machine learning libraries on DGX server

Zheng Lu (luz1@ornl.gov)

I will use TFGAN, a generative adversarial network library on tensorflow, as an example. Other libraries should be similar. Please replace anything in [], e.g. replace [UCAMS_id] with your 3-digit id.

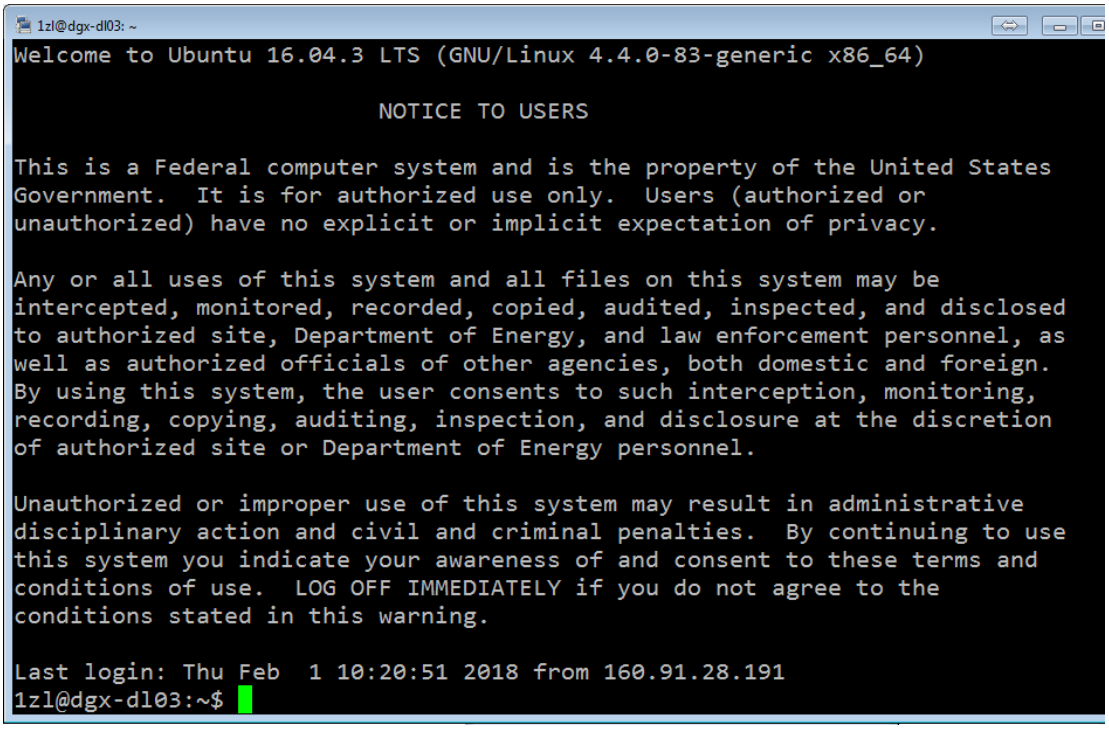
Preparation

1. Connect to DGX server using SSH.

Using your UCAMS 3-digit id and pin to login the DGX server via ssh. Windows users can use ssh softwares such as putty (https://www.chiark.greenend.org.uk/~sgtatham/putty/).

```
ssh [UCAMS_id]@dgx-dl03.ornl.gov
```

If you successfully login the system, you should see “Notice to users” and last loin information.



2. Check your sudo priviledge.

To use GPU resource on DGX server, you must have sudo priviledge to run nvidia-docker. You can use the following command to check it.

sudo -l

```
1zl@dgx-dl03:~$ sudo -l
Matching Defaults entries for 1zl on dgx-dl03:
    env_reset, mail_badpass,
    secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sb
n\:/snap/bin

User 1zl may run the following commands on dgx-dl03:
    (root) NOPASSWD: /usr/bin/docker, /usr/bin/nvidia-docker,
        /usr/bin/nvidia-smi
```

Run nvidia dockers

We use nvidia dockers on DGX server to run various machine learning tasks. We have several docker images pre-built on our DGX system. A nvidia docker image usually contains a machine learning library, e.g. tensorflow and several required software to run that library, e.g. python. When creating a container based on a docker image, you can think of all the libraries and softwares in that docker image are installed for you in a virtual environment.

1. Show Nvidia docker images.

You may want to see what docker images do we have on DGX server first with the following command.

sudo docker images

```
1zl@dgx-dl03:~$ sudo docker images
REPOSITORY                  TAG                             IMAGE ID            CREATED
abm_pytorch                 17.12_sklearn                   6fc42ce5ee00        39 hours
tensorflow/tensorflow       1.5.0-devel-gpu-py3             3fc7a04f7d92        5 days a
tensorflow/tensorflow       1.5.0-devel-gpu                 cfc1345e3d5c        5 days a
tensorflow/tensorflow       latest-gpu                      e6f4580b979c        5 days a
tensorflow_jbk              latest                          2ab1b6278441        6 days a
dwpytorch                   17.12                           c2d469cdeed9        3 weeks
pytorch                     17.12_g77                       bcfa3927c5c1        3 weeks
nvcr.io/nvidia/pytorch      17.12                           5ac6ff8f9a81        2 months
hello-world                 latest                          f2a91732366c        2 months
nvcr.io/nvidia/cuda         9.0-cudnn7-devel-ubuntu16.04    634be617d3ed        2 months
nvcr.io/nvidia/pytorch      17.11                           fb6782fcfd54        2 months
nvcr.io/nvidia/tensorflow   17.10                           9b6a599f403c        3 months
nvcr.io/nvidia/pytorch      17.10                           41a39926dad1        3 months
```

2. Create container

Once you have decided which docker image to use, you can create a container for that docker.

Note that container won't save your work to harddrive automatically. So you must pass a directory name to the container when creating it. Otherwise all the data created by the container will lose once you stop or kill the container.

So, you can make a directory first with following command:

mkdir [data_folder]

Now you can create the container:

sudo nvidia-docker run -v [data_folder]:[path_for_mapping] -it --name=[container_name] --net=host --rm [image_name]:[tag]

```
1zl@dgx-dl03:~/models/research/gan/mnist$ sudo nvidia-docker run -v /home/1z
l:/data -it --name=zlu_test1 --net=host --rm tensorflow/tensorflow:1.5.0-dev
el-gpu
root@dgx-dl03:~#
```

There are a few things to notice here:

- [path_for_mapping] is the location where your [data_folder] will mapping to in the container. Make sure to store all you computation results and intermediate results in the [path_for_mapping] directory so that after you stopping or killing your container, the data is still available.
- Remember to specify a container_name so that later you know which container is yours. By default, the system will generate a random name.
- Image_name and tag are from previous section when you run "sudo docker images"
- Once you are in the container, you can see your user name is change from your 3-digit id to root.
- Do not use "exit" or "ctrl-c" in your container command line unless you want to kill your container.

3. Check environment

You can now check if the desired machine learning library has been already installed with the following command:

python -c 'import tensorflow as tf; print(tf.__version__)' # for Python 2

python3 -c 'import tensorflow as tf; print(tf.__version__)' # for Python 3

```
root@dgx-dl03:~# python -c 'import tensorflow as tf; print(tf.__version__)'
 # for Python 2
1.5.0
```

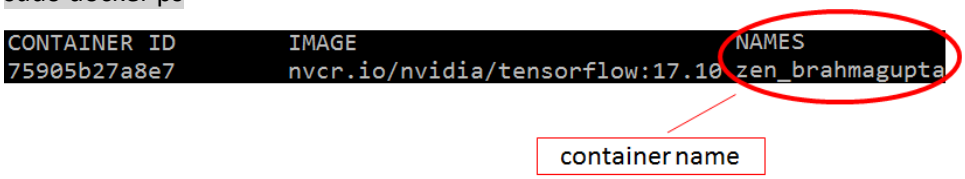
This shows tensorflow has been installed already and the version is 1.5.0

If you see "no module named …" error, this means your library has not been installed correctly.

4. Container operations

- Detach a container:
  If you still have codes running in the container and just want to exit the container and let it run in background. You can run the following command:
  Ctrl-p + Ctrl-q

- Stop a container:
  If you have done with your container and don't need it anymore, you can simply kill it by running the following command:
  - If you are still attached to the container:
    Exit or Ctrl-c
  - If you have detached from the container:
    sudo docker kill [container name]
    Make sure you kill the correct container and not other people's container!
    Double check your container name.
    You can run the following command to see all the running containers.
    sudo docker ps



- Re-attach to a container:
  You can re-attach to the container by:
  sudo docker attach [container name]
- Open a new bash window for a container:
  You can open multiple bash window for a container:
  sudo docker exec –it [container name] /bin/bash
  When exit the extra bash window, you can simply use:
  Exit
  This will not kill the container.

Using GPU in containers

There are 4 Nvidia Tesla V100 GPU in our DGX server. Normally, when you run your code in tensorflow, it will try to allocate all GPU memory of all 4 GPUs. A good way to share the server with other people is to limit your usage to a single GPU or fraction of a single GPU.

1. Select GPU to use

To specify which GPU you want to use. You can set the environment parameter "CUDA_VISIBLE_DEVICES". You can set it either in bash or in your python code.

- In bash, this will affect all codes running after:
  export CUDA_VISIBLE_DEVICES="[index to GPUs]"

```
root@dgx-dl03:~# export CUDA_VISIBLE_DEVICES="1,3"
root@dgx-dl03:~# echo $CUDA_VISIBLE_DEVICES
1,3
```

- In python code, this will only affect the code containing this command:
  import os
  os.environ["CUDA_VISIBLE_DEVICES"] = "[index to GPUs]"

2. Limit your GPU memory use:

For tensor flow, you can limit the GPU memory usage in your code:

```
# Assume that you want to allocate ~8GB out of 16GB:
gops = tf.GPUOptions(per_process_gpu_memory_fraction=0.5)
sess = tf.Session(config=tf.ConfigProto(gpu_options=gops))
```

Using TFGAN library

TFGAN is a light weight library for Generative Adversarial Networks. Open sourced by Google in December 2017. It provides simple function calls that cover the majority of GAN use-cases. You can just use the modules you want — loss, evaluation, features, training, etc. are all independent. When you use TFGAN, you'll be using the same infrastructure that many Google researchers use, and you'll have access to the cutting-edge improvements that Google develop with the library.

1. Requirements

The TFGAN library is built-in the tensorflow 1.5.0, earlier versions may not have it. You can test it by running the following command:

```
python -c "import tensorflow.contrib.gan“
```

2. TFGAN examples

The TFGAN examples and tutorials are in tensorflow models repository:
https://github.com/tensorflow/models

You can download the models repository with:

```
git clone https://github.com/tensorflow/models.git
```

The location of TFGAN is in models/research/gan

3. Run the MNIST example

You can run the MNIST example with the following commands:

```
cd models/research/gan/mnist
./launch_jobs.sh [gan_type] [models_location]
```

```
root@dgx-dl03:/data# cd models/research/gan/mnist
root@dgx-dl03:/data/models/research/gan/mnist# ./launch_jobs.sh unconditional /data/models
Dataset files already exist. Exiting without re-creating them.
Starting training unconditional GAN for 300 steps...
INFO:tensorflow:Summary name Generator/fully_connected/weights:0 is illegal; using Generator
INFO:tensorflow:Summary name Generator/fully_connected/BatchNorm/beta:0 is illegal; using Ge
INFO:tensorflow:Summary name Generator/fully_connected_1/weights:0 is illegal; using Generat
```

[gan_type] can be “unconditional”, “conditional” and “infogan”

[models_location] is the directory where you download model repository to, e.g. /data/models

4. Current problems with TFGAN examples

- MNIST
  The launch script contains a bug. However, I have just submitted it on the github (https://github.com/tensorflow/models/issues/3295). It should be solved in future versions.

- CIFAR
  The cifar example won't run correctly on our DGX server. The training will run for around 20 steps and then it will hang. I don't know the exact problem yet. However, the problem might be somewhere in CUDA drivers. Because when I run the program on CPU only, it runs well.