Research Cluster Instructions

Connecting to the Cluster

1. **Accessing the Cluster**: To begin, ensure you have access to the Discovery cluster(if not, please apply for one at rc request). You can log in by using the SSH command:

   ```
   ssh <your_username>@login.discovery.neu.edu
   ```

   You will then be prompted to enter your email password. Once logged in, you will be directed to your home directory`$Home`, located at `/home/<your_username>`. Please be aware that this is a login node, and extensive computational tasks should not be performed here. Instead, you will need to request a compute node to train and test your code.

2. **Recommended SSH Login Method**: The preferred(and **RECOMMENDED**) method for logging into Discovery is via SSH. First, generate an SSH key, then add your `id_rsa.pub` file to the `authorized_keys` file in discovery. For detailed instructions, refer to the guide for Mac. Once set up correctly, you will be able to log in without needing to enter your password each time.

Note: It is not recommended to use VSCode for connecting to Discovery. Since the VSCode Server is not integrated into Discovery, it may take a significant amount of time to initialize VSCode each time you log in, as well as increase CPU usage on the login node.

Note: Additionally, using Open OnDemand (OOD) for connecting to Discovery is also discouraged due to poor connection reliability experienced while using OOD.

Note: If anyone finds a good solution to these connection problems, please let me know so we can adopt a better approach!

Basic Environment Setup

1. **Requesting a Compute Node**
   To request a compute node, use the following `srun` command syntax:

   ```
   srun [options] [command]
   ```

   Example:

   ```
   srun --partition=gpu --nodes=1 --gres=gpu:v100-sxm2:1 --cpus-per-
   task=2 --mem=10GB --time=02:00:00 --pty /bin/bash
   ```

   - `--partition` specifies the node type, such as `gpu`.
   - `--gres` specifies the type of GPU requested, in this case, a `v100-sxm2`.

- `/bin/bash` starts a bash shell on the compute node.

2. **Conda Setup**
   After you access the compute node, load the required versions of Anaconda and CUDA:

```
module load anaconda3/2022.05 cuda/11.8
```

   Then, create and activate a Conda environment, and install packages as you would on your local machine:

```
conda create --prefix=<env_path> -c conda-forge python=3.10 -y
conda activate <env_path>
conda install <package>
```

3. **Where to Store Environments and Data**
   You have three main directories for storing environments and data:

   - `/scratch/<your_username>`: Temporary storage, though files are not frequently cleared.
   - `/home/<your_username>`: Temporary storage we suppose, though documentation is unclear about usage limits.ins
   - `/work/<groupname>`: Permanent storage. You must request access by submitting a storage request.

4. **Next Time Login** First request a compute node, and then load the anaconda(*perhaps you don't need*) and actiavte the conda env. You can also check exsited env using

```
conda env list
```

Now, you're ready to make full use of the Research Cluster!

Let me know if anyone find something helpful! I'll keep updating it!

Training DeepGD on the Research Cluster

This guide covers my current steps for training DeepGD on the Research Cluster.

1. **Create an Environment and Clone the Code**
   First, request a compute node with a GPU and load Anaconda and CUDA 11.8:

```
srun --partition=gpu --nodes=1 --gres=gpu:v100-sxm2:1 --cpus-per-
task=2 --mem=10GB --time=02:00:00 --pty /bin/bash
module load anaconda3/2022.05 cuda/11.8
```

Next, create a Conda environment in your scratch or work directory. In my case, I used `/scratch/li.xuefen/gdtest`:

```
conda create --prefix=/scratch/li.xuefen/gdtest -c conda-forge
python=3.10 -y
conda activate /scratch/li.xuefen/gdtest
```

Then, navigate to the scratch directory and clone the DeepGD repository:

```
cd /scratch/li.xuefen/
git clone https://github.com/yolandalalala/DeepGD.git
cd DeepGD
```

2. **Install Dependencies**
   Install the required dependencies:

```
pip install -r requirements.txt
```

Next, install PyTorch, TorchVision, and Torchaudio with CUDA 11.8 compatibility:

```
pip install torch==2.0.0+cu118 torchvision==0.15.0+cu118
torchaudio==2.0.0+cu118 --extra-index-url
https://download.pytorch.org/whl/cu118
```

Additionally, install any other necessary packages:

```
pip install pandas matplotlib attrs ssgetpy
```

3. **Running Jupyter Notebooks**
   Although I haven't tried running Jupyter notebooks directly, I currently convert the `.ipynb` files to Python scripts (e.g., `try.py`) and run them with:

```
python try.py
```

For file transfer, you can either use OOD or sshfs, and you also need to delete`%load_ext autoreload` and `%autoreload 2`.

I have tried using jupyter and successfully connected to it. But I have no idea how to use it. If anyone find a way, please notify me so that we can adopt a better practice. Here's what i have done to launch jupyter:

First suppose you have requested a compute node(e.g. `d1024`) and use conda env, then on your own machine

```
ssh -L 8889:localhost:8889 li.xuefen@login.discovery.neu.edu
ssh -L 8889:localhost:8889 li.xuefen@d1024
```

Then on computer node:

```
conda install jupyterlab -y
jupyter lab --no-browser --port=8889
```

You may need to adjust the port accordingly. After launch jupyter, you can access through web browser using URLs provided in jupyterLab(e.g.http://localhost:8889/lab?token=e821090dee9d25c256814c12c4a4fef77ded8450bf847bda))

4. **Making DeepGD a Module**
   To ensure DeepGD is recognized as a module, add an `__init__.py` file:

```
touch deepgd/__init__.py
```

5. **Finally Train DeepGD**

```
python try.py
```