WORKING OF DEEP LEARNING LIBRARY (PYTORCH) ON REMOTE LINUX SYSTEMS WITH GPU

OVERVIEW

PyTorch is a Python-based scientific computing package. It is an automatic differentiation library that is useful to implement neural networks. Just like how you transfer a Tensor onto the GPU, you transfer the neural network onto the GPU.

A Python virtual environment can be set-up on Arc to install and run PyTorch. An example of using PyTorch with CIFAR-10 Dataset in the Python virtual environment on Arc with GPU nodes is discussed further in the sections below. Performance of the deep learning model on CPU and GPU nodes are tabulated in the last part of this section.

INSTALLING AND RUNNING PYTORCH ON ARC

1. **Login to Arc using your user credentials

```
localhost $ ssh -X -p 22 username@arc.utsa.edu
```

**Note (Mac Users): - Mac users will have to download and install XQuartz for launching GUI-based applications on remote Linux systems.
**Note (Windows Users): -Windows users will have to download and install Xming/Mobaxterm for launching GUI-based applications on remote Linux systems.

2. Switch to GPU node for an hour using the following command

SINGLE GPU:

```
[username@login001]$ srun -p gpu1v100 -N 1 -n 1 -t \
```

```
01:00:00 --pty bash
```

MULTIPLE GPU:

```
[username@login001]$ srun -p gpu2v100 -N 1 -n 1 -t \
01:00:00 --pty bash
```

3. Create and activate a Python Virtual Environment, enter the following commands sequentially

```
[username@gpu001]$ pip install virtualenv
[username@gpu001]$ virtualenv mypython
[username@gpu001]$ source mypython/bin/activate
```

Note:- To deactivate the environment, enter command “deactivate mypython”.

4. To check the availability of the GPU units on a node, use the following command:

SINGLE GPU:

```
(mypython)[username@gpu001]$ nvidia-smi
```

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.56       Driver Version: 460.56   CUDA Version: 11.2         |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100S-PCI...  Off  | 00000000:3B:00.0 Off |                    Off |
| N/A   32C  P0    35W / 250W |      0MiB / 32510MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
```

```
|=============================================================================|
|  No running processes found                                                 |

+-----------------------------------------------------------------------------+
```

MULTIPLE GPU:

```
(mypython)[username@gpu031]$ nvidia-smi
```

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.56       Driver Version: 460.56     CUDA Version: 11.2       |
|-------------------------------+----------------------+----------------------+
| GPU  Name         Persistence-M| Bus-Id      Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|       Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100S-PCI...  Off  | 00000000:3B:00.0 Off |                  Off |
| N/A   30C   P0     34W / 250W |        0MiB / 32510MiB |   0%     Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100S-PCI...  Off  | 00000000:D8:00.0 Off |                  Off |
| N/A   32C   P0     36W / 250W |        0MiB / 32510MiB |   0%     Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI           PID   Type   Process name                 GPU Memory |
|        ID   ID                                                     Usage      |
|=============================================================================|
|  No running processes found                                                 |

+-----------------------------------------------------------------------------+
```

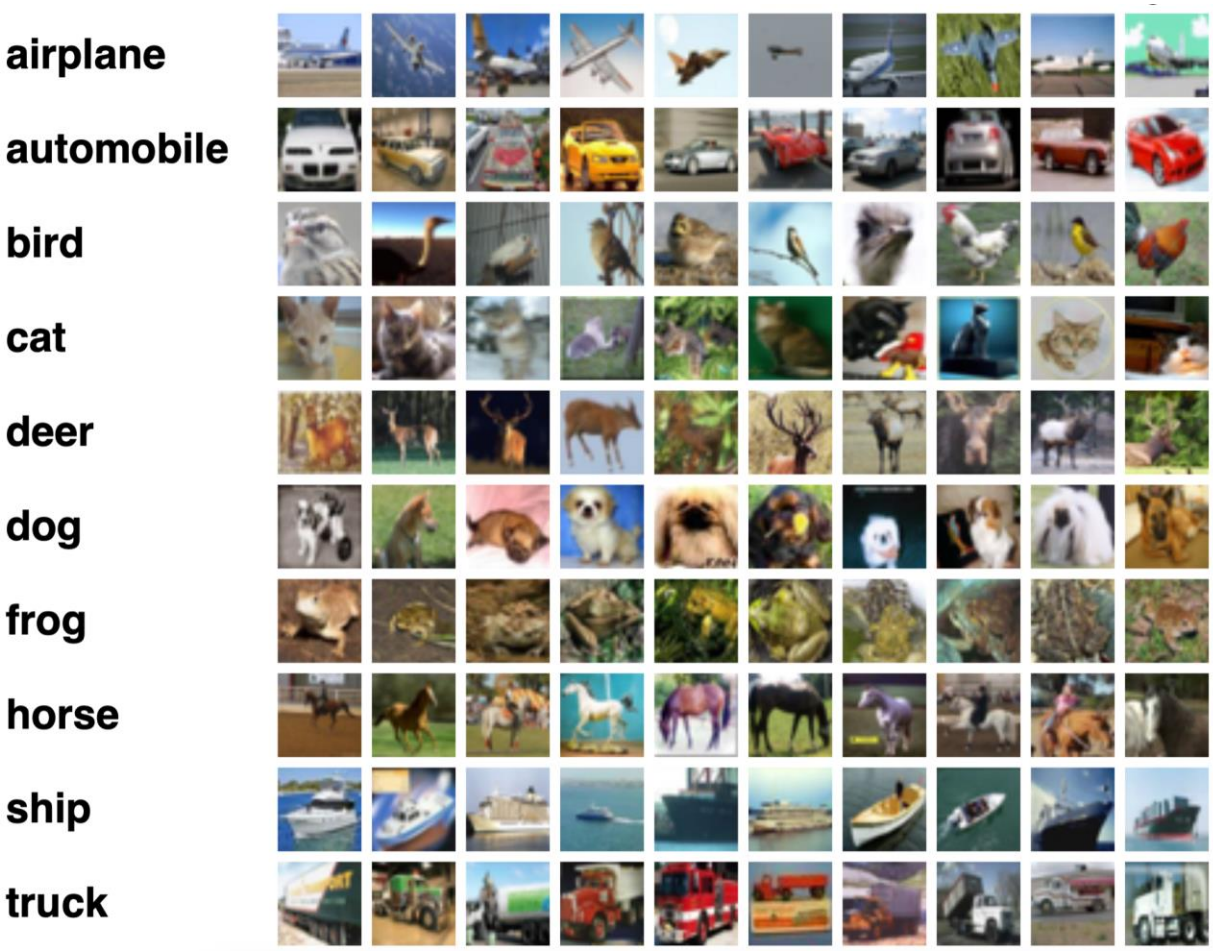
5. THE CIFAR-10 DATASET (CIFAR-10 DATASET)

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training

batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are the classes in the dataset, as well as 10 random images from each:



The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

RUNNING THE PYTORCH EXAMPLE

6. A sample deep learning model on CIFAR10 dataset using PyTorch and the computation power of the GPU is shown in Listing 1. All this code does is train the model on the CIFAR10 dataset to classify images of different birds, animals etc and calculate the prediction accuracy on the test dataset.

```
import torch
import torchvision
import torchvision.transforms as transforms
import numpy as np
import torch.nn as nn
import torch.nn.functional as F

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Assuming that we are on a CUDA machine, this should print a CUDA device:

print(device)

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

dataiter = iter(trainloader)
images, labels = dataiter.next()

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 216, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(216, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
net = Net()
net.to(device)
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)

for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    a = enumerate(trainloader, 0)
    for i, data in a:
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data[0].to(device), data[1].to(device)
        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
dataiter = iter(testloader)
images, labels = dataiter.next()

net = Net()
net.load_state_dict(torch.load(PATH))
outputs = net(images)
_, predicted = torch.max(outputs, 1)

correct = 0
total = 0
# since we're not training, we don't need to calculate the gradients for our outputs
with torch.no_grad():
    for data in testloader:
        images, labels = data
        # calculate outputs by running images through the network
        outputs = net(images)
        # the class with the highest energy is what we choose as prediction
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
```

```
        correct += (predicted == labels).sum().item()

print('Accuracy of the network on the 10000 test images: %d %%' % (
    100 * correct / total))

# prepare to count predictions for each class
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}

# again no gradients needed
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1


# print accuracy for each class
for classname, correct_count in correct_pred.items():
    accuracy = 100 * float(correct_count) / total_pred[classname]
    print("Accuracy for class {:5s} is: {:.1f} %".format(classname,
                                                        accuracy))
```

Listing 1 : Image Classification PyTorch Example with GPU

6. The Deep Learning Model can be run either in batch mode using a Slurm batch job-script or interactively on a GPU node.

Running the model in existing Interactive-Mode: The program can be run interactively on a GPU node using the following set of commands and the output will be displayed on the terminal:

SINGLE GPU:

For deep learning models, training time is the essential parameter which determines the performance of the model. Time taken by the model training can be computed by running the following command:

```
[username@gpu001]$ time python3 program_name.py
```

A snippet of the output from the command line is shown here:

```
cuda:0
[1,  2000] loss: 1.997
[1,  4000] loss: 1.656
[1,  6000] loss: 1.528
[1,  8000] loss: 1.417
[1, 10000] loss: 1.362
[1, 12000] loss: 1.326
[2,  2000] loss: 1.252
[2,  4000] loss: 1.206
[2,  6000] loss: 1.167
[2,  8000] loss: 1.164
[2, 10000] loss: 1.132
[2, 12000] loss: 1.091
Finished Training

Accuracy of the network on the 10000 test images: 62 %
Accuracy for class plane is: 69.4 %
Accuracy for class car   is: 78.6 %
Accuracy for class bird  is: 41.1 %
Accuracy for class cat   is: 48.1 %
Accuracy for class deer  is: 61.2 %
Accuracy for class dog   is: 51.8 %
Accuracy for class frog  is: 59.7 %
Accuracy for class horse is: 71.5 %
Accuracy for class ship  is: 72.2 %
Accuracy for class truck is: 75.6 %
```

```
real 1m39.540s
user 2m40.883s
sys  0m24.707s
```

MULTIPLE GPU:

```
[username@gpu031]$ time python3 program_name.py
```

A snippet of the output from the command line is shown here:

```
cuda:0
[1,  2000] loss: 2.013
[1,  4000] loss: 1.674
[1,  6000] loss: 1.530
[1,  8000] loss: 1.466
[1, 10000] loss: 1.390
[1, 12000] loss: 1.337
[2,  2000] loss: 1.273
[2,  4000] loss: 1.223
[2,  6000] loss: 1.205
[2,  8000] loss: 1.189
[2, 10000] loss: 1.131
[2, 12000] loss: 1.138
Finished Training

Accuracy of the network on the 10000 test images: 60 %
Accuracy for class plane is: 72.8 %
Accuracy for class car   is: 86.6 %
Accuracy for class bird  is: 39.0 %
Accuracy for class cat   is: 34.6 %
Accuracy for class deer  is: 44.1 %
Accuracy for class dog   is: 47.1 %
Accuracy for class frog  is: 69.8 %
Accuracy for class horse is: 81.4 %
Accuracy for class ship  is: 72.0 %
```

```
Accuracy for class truck is: 56.8 %

real 1m35.931s
user 12m14.331s
sys  0m33.245s
```

Running the model in Batch-Mode: Interactive jobs can only be executed until a particular time frame. In order to run your job for more than that timeframe, you need to submit your model training as a batch job to the cluster. A sample Slurm batch job-script to run the python program of deep learning model in batch mode is shown in Listing 2. This script should be run from a login node.

```
#!/bin/bash
#SBATCH -J program_name
#SBATCH -o program_name.txt
#SBATCH -p gpu1v100
#SBATCH -N 1
#SBATCH -n 1
#SBATCH --time=01:00:00

source mypython/bin/activate

time python3 program_name.py
```

Listing 2: Batch Job Script for PyTorch GPU code (job_script1.slurm)

Note: In order to run the code in Listing 1 with one or more GPU devices, change the partition name(-p) in the Listing 2 batch job script. In order to run the code with GPU with one device in Listing 1, change the partition name to *gpu1v100*. In order to run the code with GPU with multiple devices in Listing 1, change the partition name to *gpu2v100*.

If you are currently on a GPU node and would like to switch back to the login node then please enter the exit command as follows:

```
(mypython)[username@gpu001]$ exit
```

The job-script shown in Listing 2 can be submitted as follows:

```
[username@login001]$ sbatch job_script1.slurm
```

The output from the Slurm batch-job shown in Listing 2 can be checked by opening the output file as follows:

```
(mypython)[username@login001]$ cat program_name.txt
```

SINGLE GPU:

```
cuda:0
[1,  2000] loss: 1.989
[1,  4000] loss: 1.659
[1,  6000] loss: 1.513
[1,  8000] loss: 1.415
[1, 10000] loss: 1.372
[1, 12000] loss: 1.320
[2,  2000] loss: 1.233
[2,  4000] loss: 1.211
[2,  6000] loss: 1.177
[2,  8000] loss: 1.159
[2, 10000] loss: 1.131
[2, 12000] loss: 1.122
Finished Training

Accuracy of the network on the 10000 test images: 61 %
Accuracy for class plane is: 61.9 %
Accuracy for class car   is: 76.9 %
Accuracy for class bird  is: 53.7 %
Accuracy for class cat   is: 35.7 %
Accuracy for class deer  is: 56.7 %
```

```
Accuracy for class dog   is: 55.7 %
Accuracy for class frog  is: 61.2 %
Accuracy for class horse is: 73.1 %
Accuracy for class ship  is: 80.3 %
Accuracy for class truck is: 64.5 %

real 1m39.540s
user 2m40.883s
sys  0m24.707s
```

MULTIPLE GPU:

```
cuda:0
[1,  2000] loss: 1.995
[1,  4000] loss: 1.637
[1,  6000] loss: 1.503
[1,  8000] loss: 1.441
[1, 10000] loss: 1.377
[1, 12000] loss: 1.292
[2,  2000] loss: 1.225
[2,  4000] loss: 1.188
[2,  6000] loss: 1.172
[2,  8000] loss: 1.136
[2, 10000] loss: 1.107
[2, 12000] loss: 1.102
Finished Training

Accuracy of the network on the 10000 test images: 61 %
Accuracy for class plane is: 73.4 %
Accuracy for class car   is: 76.9 %
Accuracy for class bird  is: 38.3 %
Accuracy for class cat   is: 30.9 %
Accuracy for class deer  is: 59.6 %
Accuracy for class dog   is: 51.8 %
```

```
Accuracy for class frog  is: 79.3 %
Accuracy for class horse is: 78.9 %
Accuracy for class ship  is: 60.2 %
Accuracy for class truck is: 64.7 %

real 1m35.931s
user 12m14.331s
sys  0m33.245s
```

PERFORMANCE OF IMAGE CLASSIFICATION ON CPU, GPU, GPU WITH DATA PARALLELISM

Table 1. Performance parameters of Image Classification

Node	Number of GPU cards	Accuracy	Time
compute1(CPU)	0	62%	real 3m0.464s user 112m34.797s sys 1m48.916s
gpu1v100 (GPU)	1	62%	real 1m39.540s user 2m40.883s sys 0m24.707s
gpu2v100 (GPU)	2	60%	real 1m35.931s user 12m14.331s sys 0m33.245s

The table shown above contrasts the training performance of models running on CPU and GPU nodes. The model's training time decreases when it runs on GPU with single and decreases further on multiple GPU devices, this shows the computation power of GPU compared to CPU. The training accuracy has very less effect with the node on which it is running.

REFERENCES

1. Data Parallel Tutorial
2. Cifar-10 Tutorial
3. Cifar-10 Dataset HTML