Introduction to Blue Waters and Pytorch

Xiaobo Dong

Agenda

- Blue Waters
  - How to login
  - How to transfer files
  - How to run interactive job
  - How to submit batch job
  - How to edit file in Blue Waters
  - Common errors and error messages
  - System Summary
- Pytorch
  - HW 1 in Pytorch
    - How to load data
    - How to define a model
    - How to train model
    - How to save model
  - How to use pre-trained model
  - About autograd
  - How to use Jupyter Notebook on your local machine

How to login Blue Waters

- ssh YOUR_BW_ID@bwbay.ncsa.illinois.edu

```
[dong6@linux-a2 ~]$ ssh instr030@bwbay.ncsa.illinois.edu
```

Your user ID

```
Password:
Last login: Mon Aug 26 22:45:29 2019 from linux-a2.ews.illinois.edu

Access by OTP or Two Factor Certificate Authority only.
Use myproxy-logon -s tfca.ncsa.illinois.edu -p 7512 for gsissh access.
gsissh or ssh -o PreferredAuthentications=keyboard-interactive for OTP access.


Blue Waters Admin Team




Last login: Mon Aug 26 22:45:29 2019 from bwedge.ncsa.illinois.edu

   ___  __            _      __     __
  / _ )/ /_ _____    | | /| / /__ _/ /____ _______
 / _  / / // / -_)   | |/ |/ / _ `/ __/ -_) __(_-<
/____/_/\_,_/\__/    |__/|__/\_,_/\__/\__/_/ /___/
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Batch and Scheduler configuration.
  Queues:  normal (default), high, low, debug
  Features:  "xe" (default), "xk", "x" (xe or xk non-specific)
             "xehimem" (128GB mem), "xkhimem" (64GB mem)
  30 min default wall time, 48 hr maximum
  -lnodes=X:ppn=Y syntax supported.

All SSH traffic on this system is monitored.

Questions?  Mail help+bw@ncsa.illinois.edu to create a support ticket.
For known issues:  https://bluewaters.ncsa.illinois.edu/known-issues
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

instr030@h2ologin1:~>
```

Shared directory and scratch directory

- The shared directory for this course is located in /projects/training/bayw/ .
- /scratch is different than~/scratch. The ~ is shorthand for your own local directory where /scratch is on the root directory. For example, my~/scratch is located at /u/training/instr030/scratch. Do not try to do anything with the root /scratch.
- The scratch directory is supposed to be wiped occasionally (I believe every 30 days although I do not think I have ever actually seen it wiped). Its probably not a good idea to keep important files in here. You can scp files back from Blue Waters to your local computer for storage if they are small enough.

How to transfer files

- Transfer from your local machine to BW
  - scp localuser@localIP:~/codeForBlueWaters.py ~/scratch/.
  - scp -r localuser@localIP:~/directoryForBlueWaters ~/scratch/.
- Transfer from BW to your local machine
  - scp codeForLocalMachine.py localuser@localIP:LocalMachineDirectory/.
  - scp -r DirectoryForLocalMachine localuser@localIP:LocalMachineDirectory/.
- Transfer from BW shared folder to your own BW directory
  - cp -r /projects/training/bayw/tutorial ~/scratch/.

Jobs on Blue Waters

- Two type of jobs – interactive and batch
- Interactive mode for debug and optimization
- Batch mode for normal job runs

How to run interactive job

- CPU job
  - qsub -I -l nodes=1:ppn=32:xe -l walltime=01:00:00
- GPU job
  - qsub -I -l gres=ccm -l nodes=1:ppn=16:xk -l walltime=01:00:00
  - module add ccm
  - ccmlogin
- Run python in interactive mode
  - module load python/2.0.1
  - aprun -n 1 -N 1 python main.py
    - -n :Number of processing elements PEs for the application
    - -N :Number of PEs to place per node

```
instr030@h2ologin2:~> qsub -I -l gres=ccm -l nodes=1:ppn=16:xk -l walltime=01:00:00
INFO: Job submitted to account: bayw
qsub: waiting for job 10269637.bw to start
qsub: job 10269637.bw ready

------------------------------------------
Begin Torque Prologue on nid27637
at Wed Aug 28 06:50:26 CDT 2019
Job Id:                   10269637.bw
Username:                 instr030
Group:                    TRAIN_bayw
Job name:                 STDIN
Requested resources:      gres=ccm,nodes=1:ppn=16:xk,walltime=01:00:00,neednodes=1:ppn=16:xk
Queue:                    normal
Account:                  bayw
End Torque Prologue:  0.106 elapsed
------------------------------------------
In CCM JOB:  10269637.bw  JID  10269637  USER  instr030  GROUP  TRAIN_bayw  WLM  torque
Initializing CCM environment, Please Wait
Warning: The -E option is deprecated and has no effect
CCM Start success, 1 of 1 responses
Directory: /u/training/instr030
Wed Aug 28 06:50:33 CDT 2019
instr030@nid27637:~> module add ccm
instr030@nid27637:~> ccmlogin
instr030@nid25486:~>
```

How to submit a batch job

- Write your own PBS script and use the following command to submit a job
  - qsub run.psb
- Check your status of your job
  - qstat –u YOUR_BW_ID

```
instr030@h2ologin3:~/pytorch_tutorial> qstat -u instr030

bwsched.ncsa.illinois.edu: Blue_Waters
                                                                  Req'd       Req'd       Elap
Job ID                  Username    Queue    Jobname          SessID  NDS   TSK   Memory      Time    S   Time
----------------------- ----------- -------- ---------------- ------ ----- ------ --------- --------- - ---------
10283612.bw             instr030    normal   your_job_name      1699     1     16       --   02:00:00 R  00:00:22
10283613.bw             instr030    normal   your_job_name       --      1     16       --   02:00:00 Q       --
```

  - ***Note: R means job is running; Q means job is queueing; C means job is completed.***

- Delete your unwanted job
  - qdel xxxxxx.bw

How to check your hour usage

- Use the command
  - usage

```
instr030@h2ologin3:~/pytorch_tutorial> usage
Proj         Mach   Login            Usage   Status    Proj_alloc    Proj_usage   Proj_expire   Full_name
-------------------------------------------------------------------------------------------------------------
jpg          bw     instr030          0.00   Active          1.00          0.00   07/31/2020    Instructor 030, Blue Waters
TRAIN_bayr   bw     instr030          0.00   Active      25000.00      48364.81   09/30/2019    Instructor 030, Blue Waters
TRAIN_bayw   bw     instr030          0.00   Active     100000.00          0.00   12/15/2019    Instructor 030, Blue Waters
```

PBS Script

- Specify resource needed
- Provide file names for stdout and stderr
- Define environmental variables
- Load needed modules
- Launch the job via the aprun command

Example of a PBS script for a GPU job

```
#!/bin/bash
#PBS -l nodes=01:ppn=16:xk
#PBS -l walltime=02:00:00
#PBS -N your_job_name
#PBS -e $PBS_JOBID.err
#PBS -o $PBS_JOBID.out
#PBS -m bea
#PBS -M YOUR_NETID@illinois.edu

cd /u/training/instr030/code_directory

. /opt/modules/default/init/bash # NEEDED to add module commands to shell
module load python/2.0.1
#module load cudatoolkit
aprun -n 1 -N 1 python main.py
```

Example of a PBS Script for a CPU job

```
#!/bin/bash
#PBS -l nodes=01:ppn=32:xe
#PBS -l walltime=02:00:00
#PBS -N your_job_name
#PBS -e $PBS_JOBID.err
#PBS -o $PBS_JOBID.out
#PBS -m bea
#PBS -M YOUR_NETID@illinois.edu

cd /u/training/instr030/code_directory

. /opt/modules/default/init/bash # NEEDED to add module commands to shell
module load python/2.0.1
#module load cudatoolkit
aprun -n 1 -N 1 python main.py
```

Explanation of PBS Script

- `#PBS -l nodes=01:ppn=16:xk`

  This is used to declare a GPU node. Here xk is to indicate it is a GPU node. if you plan to use CPU, you can use

  `#PBS -l nodes=01:ppn=32:xe`

- `#PBS -l walltime=02:00:00`

  This is used to set a limitation on the running time of your code. The job will stop at the time of min(your code running time, walltime)

- `#PBS -N your_job_name`

  This is to set your job name, which could be used when you check your job status.

- `#PBS -e $PBS_JOBID.err`

  This is to set your job error message file. If you code has some error, the error message will be shown up in this file.

- `#PBS -o $PBS_JOBID.out`

  This is to set your job output file. If you have print function in your code, the print output will be shown in this file.

- `#PBS -m bea`

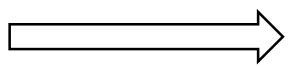
- `#PBS -M YOUR_NETID@illinois.edu`

  This is to set an Email notification when your job begins to run and finish.

- `cd /u/training/instr030/code_directory`

  The script will run in the 'code_directory'. You can store this script anywhere, but this script will run the file in the directory you defined.

- `module load python/2.0.1`



  - module load python/2.0.1 # For pytorch 0.4.0
  - module load bwpy # support distributed pytorch with version 0.3.0

- `aprun -n 1 -N 1 python main.py`

  This part is similar to the command when you run python in interactive mode.

Output file of a job.

- XXXX.bw.err(XXXX.bw.ERR)
  - This contains all the error and warning message of your Python code.
  - XXXX.bw.err means the job is done. XXXX.bw.ERR means the job is running

- XXXX.bw.out(XXXX.bw.OUT)
  - This contains all the output of your print function in Python.
  - XXXX.bw.out means the job is done. XXXX.bw.OUT means the job is running

How to edit file in Blue Waters

- Edit file in Blue Waters with vim
  - Use vim(vi) to edit your python file
  - *vim main.py*
  - Press *i* to enter insert mode
  - Press *shift+;* to go to commend mode.
    - Enter *q* :normal quit
    - Enter *q!* :quit without save
    - Enter *x* :quit and save
    - Enter */example* : search the string ‘example’ in the file.
- Similarly, you can also edit file in Blue Waters with Emacs

- Edit file on your local machine
  - Download your file to your local machine and edit it with your favorite text editor and upload to Blue Waters once it is done.

The sample files

- We provide sample files for feedforward neural networks on MNIST dataset. These files are located at:

  /projects/training/bayw/pytorch_tutorial

  - main.py : main file to train a single layer feedforward neural network classifier on MNIST dataset.
  - model.py: the single layer feedforward neural network is defined in model.py file
  - run.pbs : PBS script to submit a GPU job.

Common errors and error messages

- OOM killer terminated this process. This error message results when your application exceeds the available memory on a node.

- Claim exceeds reservation's node-count. This error message results when the combination of PBS nodes and aprun options (for example, –N, –S, –ss, –sn, –m) requires more nodes than were reserved for you by the qsub command.).

System Summary on Blue Waters nodes

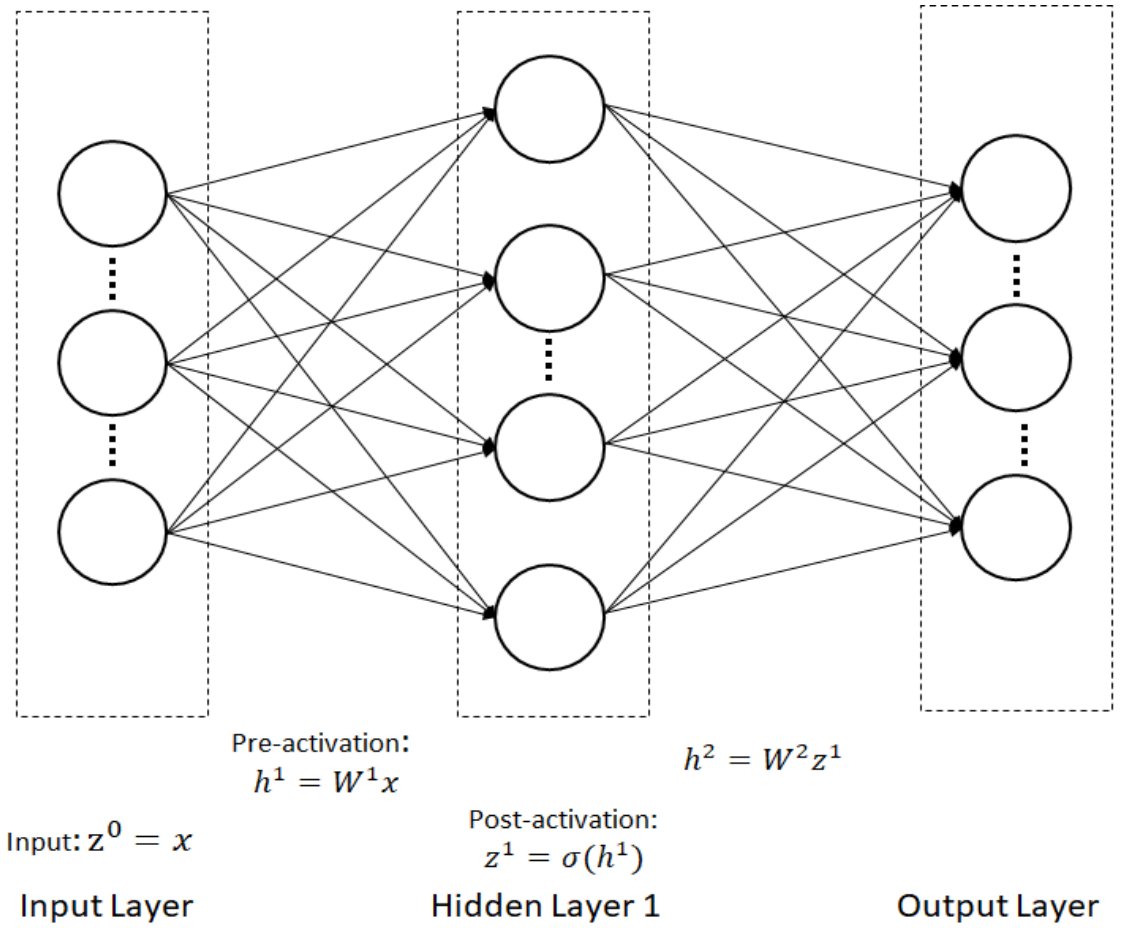
XE Compute Node	
AMD 6276 Interlagos Processors	2
Bulldozer Cores*	16
Integer Scheduling Units**	32
Memory / Bulldozer Core	4 GB
Total Node Memory	64 GB
Peak Performance	313.6 GF
Memory Bandwidth	102.4 GB/s

XK Compute Node	
AMD 6276 Interlagos Processors	1
Bulldozer Cores*	8
Integer Scheduling Units**	16
Memory / Bulldozer Core	4 GB
Node System Memory	32 GB
GPU Memory	6 GB
Peak CPU Performance	156.8 GF
CPU Memory Bandwidth	51.2 GB/s
CUDA cores	2688
Peak GPU Performance (DP)	1.31 TF
GPU Memory Bandwidth (ECC off)***	250 GB/s

Pytorch

Pytorch tutorial HW 1 example

- Implement HW 1 in Pytorch
  - With hidden size: 500
  - With nonlinear function: ReLu
- Demo on Jupyter notebook



CPU version

```python
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import time

input_size = 784
hidden_size = 500
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001
scheduler_step_size = 10
scheduler_gamma = 0.1
train_dataset = torchvision.datasets.MNIST(root='./',
                                           train=True,
                                           transform=transforms.ToTensor()
                                           download=True)
test_dataset = torchvision.datasets.MNIST(root='./',
                                          train=False,
                                          transform=transforms.ToTensor())
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

```python
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

```python
model = NeuralNet(input_size, hidden_size, num_classes)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                            step_size=scheduler_step_size,
                                            gamma=scheduler_gamma)
total_step = len(train_loader)
start_time = time.time()
```

```python
for epoch in range(num_epochs):
    scheduler.step()
    correct = 0
    total = 0
    for images, labels in train_loader:
        # Move tensors to the configured device
        images = images.reshape(-1, 28*28)
        labels = labels
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    train_accuracy = correct/total
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in test_loader:
            images = images.reshape(-1, 28*28)
            labels = labels
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    test_accuracy = correct/total
    print ('Epoch {}, Time {:.4f}, Loss: {:.4f}, Train Accuracy: {:.4f},Test Accuracy: {:.4f}'
           .format(epoch, time.time()-start_time, loss.item(),train_accuracy,test_accuracy))
    torch.save(model.state_dict(), 'epoch-{}.ckpt'.format(epoch))
```

GPU version

```python
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import time

input_size = 784
hidden_size = 500
num_classes = 10
num_epochs = 5
batch_size = 100
learning_rate = 0.001
scheduler_step_size = 10
scheduler_gamma = 0.1
train_dataset = torchvision.datasets.MNIST(root='./',
                                           train=True,
                                           transform=transforms.ToTensor(),
                                           download=True)
test_dataset = torchvision.datasets.MNIST(root='./',
                                          train=False,
                                          transform=transforms.ToTensor())
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

```python
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out
```

```python
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = NeuralNet(input_size, hidden_size, num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                            step_size=scheduler_step_size,
                                            gamma=scheduler_gamma)
total_step = len(train_loader)
start_time = time.time()
```

```python
for epoch in range(num_epochs):
    scheduler.step()
    correct = 0
    total = 0
    for images, labels in train_loader:
        # Move tensors to the configured device
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    train_accuracy = correct/total
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in test_loader:
            images = images.reshape(-1, 28*28).to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    test_accuracy = correct/total
    print ('Epoch {}, Time {:.4f}, Loss: {:.4f}, Train Accuracy: {:.4f},Test Accuracy: {:.4f}'
           .format(epoch, time.time()-start_time, loss.item(),train_accuracy,test_accuracy))
    torch.save(model.state_dict(), 'epoch-{}.ckpt'.format(epoch))
```

Load MNIST dataset from PyTorch torchvision dataset

```
# MNIST dataset
train_dataset = torchvision.datasets.MNIST(root='./',
                                           train=True,
                                           transform=transforms.ToTensor(),
                                           download=True)
test_dataset = torchvision.datasets.MNIST(root='./',
                                          train=False,
                                          transform=transforms.ToTensor())
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

Load data for custom dataset

```
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self):
        # TODO
        # 1. Initialize file paths or a list of file names.
        pass
    def __getitem__(self, index):
        # TODO
        # 1. Read one data from file (e.g. using numpy.fromfile, PIL.Image.open).
        # 2. Preprocess the data (e.g. torchvision.Transform).
        # 3. Return a data pair (e.g. image and label).
        pass
    def __len__(self):
        # You should change 0 to the total size of your dataset.
        return 0

# You can then use the prebuilt data loader.
custom_dataset = CustomDataset()
train_loader = torch.utils.data.DataLoader(dataset=custom_dataset,
                                           batch_size=64,
                                           shuffle=True)
```

Pytorch Tensor and Numpy Array conversion

```
device = torch.device('cuda'if torch.cuda.is_available() else 'cpu')
# Assume we have GPU
x = np.ones((10,1))                    # x is numpy array on CPU
y = torch.from_numpy(x)                # y is tensor on CPU
y.to(device)                           # y is tensor on GPU
z = y.cpu().numpy()                    # z is numpy array on CPU
```

Define your module with nn.Module

- Single layer with
- ReLu nonlinear function

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out

model = NeuralNet(input_size, hidden_size, num_classes).to(device)
```

Define your module with nn.Module

- Single layer with
- tanh nonlinear function

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.Tanh()
        self.fc2 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(out)
        return out

model = NeuralNet(input_size, hidden_size, num_classes).to(device)
```

Define your module with nn.Module

- Multiple layers with
- ReLu nonlinear function

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.fc2 = nn.Linear(hidden_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        out = self.fc1(x)
        out = self.relu(out)
        out = self.fc2(x)
        out = self.relu(out)
        out = self.fc3(out)
        return out
```

```
model = NeuralNet(input_size, hidden_size, num_classes).to(device)
```

Define loss optimizer and scheduler

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
                                            step_size=scheduler_step_size,
                                            gamma=scheduler_gamma)
```

There are different optimizers. For example, you can use Adam optimizer as following

```
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

Update learning rate

Use Pytorch Scheduler to update learning rate

```
>>> # Assuming optimizer uses lr = 0.05 for all groups
>>> # lr = 0.05     if epoch < 30
>>> # lr = 0.005    if 30 <= epoch < 60
>>> # lr = 0.0005   if 60 <= epoch < 90
>>> # ...
>>> scheduler = StepLR(optimizer, step_size=30, gamma=0.1)
>>> for epoch in range(100):
>>>     train(...)
>>>     validate(...)
>>>     scheduler.step()
```

Manually update learning rate

```
lr = 0.5
optimizer = optim.SGD(model.parameters(), lr=learning_rate)

for epoch in num_epoch:
    train(...)
    validate(...)
    optimizer.param_group['lr'] = 0.5*optimizer.param_group['lr']
```

Train and test your model

```
# Training
for epoch in range(num_epochs):
    scheduler.step()
    for data, labels in train_loader:
        # Forward pass
        outputs = model(data)
        loss = criterion(outputs, data)
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

# Testing
with torch.no_grad():
    for data, labels in test_loader:
        outputs = model(data)
        # Count the number of correct prediction
    # Calcuate your testing accuracy
```

Train and test your model

Train:

```
for epoch in range(num_epochs):
    scheduler.step()
    correct = 0
    total = 0
    for images, labels in train_loader:
        # Move tensors to the configured device
        images = images.reshape(-1, 28*28).to(device)
        labels = labels.to(device)
        # Forward pass
        outputs = model(images)
        loss = criterion(outputs, labels)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    train_accuracy = correct/total
```

Test:

```
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in test_loader:
            images = images.reshape(-1, 28*28).to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    test_accuracy = correct/total
```

Save and load model

For Inference

```
# Save and load the entire model.
torch.save(resnet, 'model.ckpt')
model = torch.load('model.ckpt')

# Save and load only the model parameters (recommended).
torch.save(resnet.state_dict(), 'params.ckpt')
resnet.load_state_dict(torch.load('params.ckpt'))
```

For Inference and/or Resuming Training

```
torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': loss,
            ...
            }, PATH)


model = TheModelClass(*args, **kwargs)
optimizer = TheOptimizerClass(*args, **kwargs)

checkpoint = torch.load(PATH)
model.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
epoch = checkpoint['epoch']
loss = checkpoint['loss']
```

How to use Pre-trained model

```
# Download and load the pretrained ResNet-18.
resnet = torchvision.models.resnet18(pretrained=True)
# If you want to finetune only the top layer of the model, set as below.
for param in resnet.parameters():
    param.requires_grad = False
# Replace the top layer for finetuning.
resnet.fc = nn.Linear(resnet.fc.in_features, 100) # 100 is an example.

# Forward pass.
images = torch.randn(64, 3, 224, 224)
outputs = resnet(images)
print (outputs.size()) # (64, 100)
```

About Autograd

```python
# Create tensors.
x = torch.tensor(1., requires_grad=True)
w = torch.tensor(2., requires_grad=True)
b = torch.tensor(3., requires_grad=True)

# Build a computational graph.
y = w * x + b    # y = 2 * x + 3

# Compute gradients.
y.backward()

# Print out the gradients.
print(x.grad)    # x.grad = 2
print(w.grad)    # w.grad = 1
print(b.grad)    # b.grad = 1
```

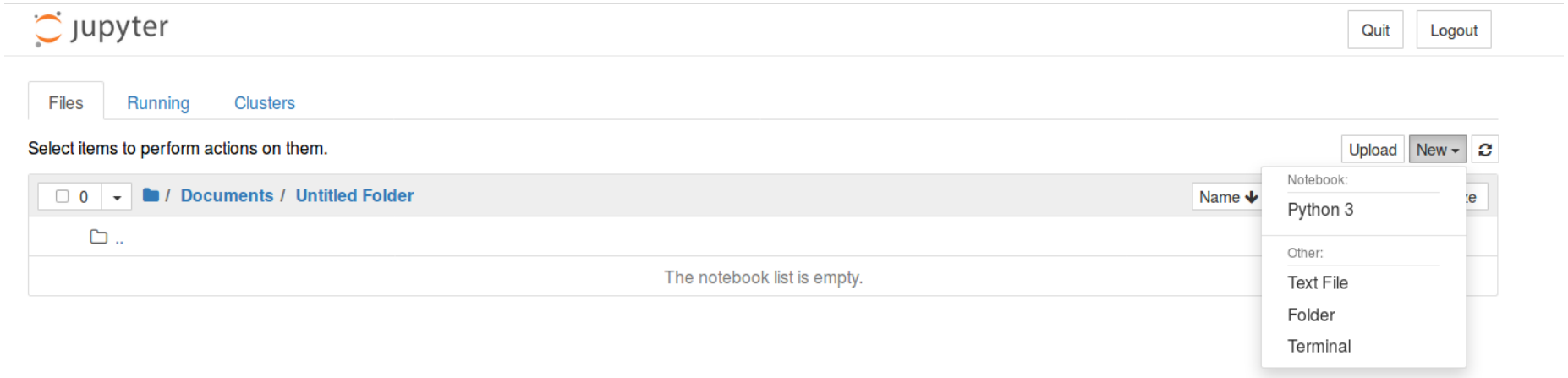
```python
# Create tensors of shape (10, 3) and (10, 2).
x = torch.randn(10, 3)
y = torch.randn(10, 2)
# Build a fully connected layer.
linear = nn.Linear(3, 2)
print ('w: ', linear.weight)
print ('b: ', linear.bias)
# Build loss function and optimizer.
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(linear.parameters(), lr=0.01)
# Forward pass.
pred = linear(x)
# Compute loss.
loss = criterion(pred, y)
print('loss: ', loss.item())
# Backward pass.
loss.backward()
# Print out the gradients.
print ('dL/dw: ', linear.weight.grad)
print ('dL/db: ', linear.bias.grad)
# 1-step gradient descent.
optimizer.step()
# You can also perform gradient descent at the low level.
# linear.weight.data.sub_(0.01 * linear.weight.grad.data)
# linear.bias.data.sub_(0.01 * linear.bias.grad.data)

# Print out the loss after 1-step gradient descent.
pred = linear(x)
loss = criterion(pred, y)
print('loss after 1 step optimization: ', loss.item())
```

About Jupyter Notebook

- Download Anaconda from https://www.anaconda.com/distribution/#download-section
- Use command
  - jupyter notebook
- If the notebook is not automatically open, you can open it by the url, which can be found in the terminal.

http://localhost:8888/?token= XXXXX

- To create a notebook, open “New” and then “Python 3”.



Reference

- Blue Waters:
  - Official Doc:
    https://Blue Waterss.ncsa.illinois.edu/documentation
  - Blue Waters Tutorial:
    https://courses.engr.illinois.edu/ie534/fa2019/secure/Blue_Waters_Help_Document.pdf

- Pytorch
  - All examples from:
    https://github.com/yunjey/pytorch-tutorial/blob/master/tutorials/01- basics/pytorch_basics/main.py
  - Official Doc:
    https://pytorch.org/docs/stable/
  - Other resources:
    https://www.analyticsvidhya.com/blog/2018/02/pytorch-tutorial/
    Sample codes:
    /projects/training/bayw/pytorch_tutorial (On Blue Waters)

Q&A