CS5182 Computer Graphics Coding Essentials for Point Clouds Tutorial

2024/25 Semester A

City University of Hong Kong (DG)

Outline

- Prerequisites.
  - Env (Ubuntu/Win10)
  - Nvidia GPU
  - Python in Anaconda
  - Version relationship
- PyTorch
  - Custom C++ and CUDA extensions
  - Example
- Auxiliary libraries
  - Open3D
  - PyTorch Geometric
  - Trimesh

Prerequisites

- Choose a stable and popular system
  - For Linux, it is recommended to use Ubuntu16.04/Ubuntu18.04/Ubuntu20.04. Notice that the latest is not recommended right now because there are some incompatibilities. Choosing the right version will fix most of the quirky bugs.
  - For Windows user, use windows 10/11 is ok for most of the current point cloud repositories. However, for further usage of multi-GPU environment, it is highly recommended to get familiar with linux command line as soon as possible.

WSL

Prerequisites

- Getting access to right version for your toolkits (intro)
  - a parallel computing platform and application programming interface model created by Nvidia.
  - a free and open-source distribution of the programming languages Python and R, aiming to simplify package management and deployment.
  - an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing, primarily developed by Facebook's AI Research lab.

Prerequisites

- Getting access to right version for your toolkits (make sure you have nvidia gpu and right driver)
  - https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html

这个链接打不开，得复制黏贴打开才行，因为单击时单开的是单行

** CUDA 11.0 was released with an earlier driver version, but by upgrading to Tesla Recommended Drivers 450.80.02 (Linux) / 452.39 (Windows), minor version compatibility is possible across the CUDA 11.x family of toolkits.

The version of the development NVIDIA GPU Driver packaged in each CUDA Toolkit release is shown below.

Table 3: CUDA Toolkit and Corresponding Driver Versions

CUDA Toolkit	Toolkit Driver Version	
	Linux x86_64 Driver Version	Windows x86_64 Driver Version
CUDA 12.6 Update 1	>=560.35.03	>=560.94
CUDA 12.6 GA	>=560.28.03	>=560.76
CUDA 12.5 Update 1	>=555.42.06	>=555.85
CUDA 12.5 GA	>=555.42.02	>=555.85

Prerequisites

- Getting access to right version for your toolkits (install guide)
  - A newer blog for installing cuda on windows/linux
    - https://blog.csdn.net/qq_51375047/article/details/140957904
  - Official documentation for installing anaconda and init on linux
    - ANACONDA https://docs.anaconda.com/anaconda/install/linux/

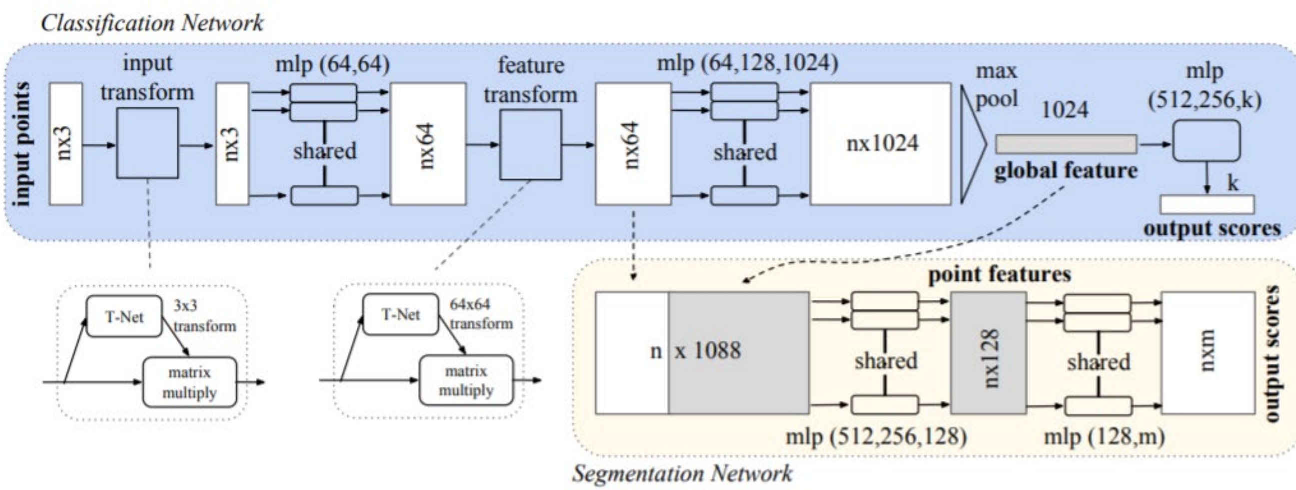
```
conda create -n your_env_name python=3.8
conda activate your_env_name
```

  - PyTorch installation under conda env
    - PyTorch https://pytorch.org/get-started/previous-versions/

```
# CUDA 11.8
conda install pytorch==2.4.0 torchvision==0.19.0 torchaudio==2.4.0  pytorch-cuda=11.8 -c pytorch -c nvidia
# CUDA 12.1
conda install pytorch==2.4.0 torchvision==0.19.0 torchaudio==2.4.0 pytorch-cuda=12.1 -c pytorch -c nvidia
# CUDA 12.4
conda install pytorch==2.4.0 torchvision==0.19.0 torchaudio==2.4.0 pytorch-cuda=12.4 -c pytorch -c nvidia
# CPU Only
conda install pytorch==2.4.0 torchvision==0.19.0 torchaudio==2.4.0 cpuonly -c pytorch
```

Example: 3D Classification PointNet

- Original paper:
  - [CVPR2017, Qi et al.] PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation
- Architecture
  - Input dim: Nx3 (N points. Each point has 3 values x, y and z)
  - Output dim:
    - For classification: k scores for all the k candidate classes
    - For segmentation: n × m scores for each of the n points and each of the m semantic subcategories.



Example: 3D Classification PointNet

- Original github (tensorflow version):
  - https://github.com/charlesq34/pointnet
- Pytorch implementation by fxia22
  - https://github.com/fxia22/pointnet.pytorch
- Ask these questions before reading the github code:
  - What datasets are used, where are they downloaded, and how are the data organized?
    - ModelNet40 / ShapeNet, public website, the dataset class of pytorch in dataset.py
  - What kind of tasks, what metrics are used to evaluate?
    - Classification / segmentation, Acc / mIOU (Mean Intersection over Union)
  - How is the model organized?
    - The nn.Module class of pytorch in model.py

Example: 3D Classification PointNet

- Original github (tensorflow version):
  - https://github.com/charlesq34/pointnet
- Pytorch implementation by fxia22
  - https://github.com/fxia22/pointnet.pytorch
- Ask these questions before reading the github code:
  - How to write a command line (cli) to run the entire training process?(i.e. find the main)
    - Here is train_xx.py, some works use main.py

Training

```
cd utils
python train_classification.py --dataset <dataset path> --nepoch=<number epochs> --dataset_type <modelnet40 |
python train_segmentation.py --dataset <dataset path> --nepoch=<number epochs>
```

Example: 3D Classification PointNet

- Original github (tensorflow version):
  - https://github.com/charlesq34/pointnet
- Pytorch implementation by fxia22
  - https://github.com/fxia22/pointnet.pytorch
- Ask these questions before reading the github code:
  - Is there any extension code that needs to be compiled first? (usually *.cpp/*.cu)
    - Sometimes, not all codes are written in pytorch. In the area of 3D point cloud, Due to the increase of data dimensions, the difficulty of preprocessing and visualization is generally higher than that of two-dimensional images. PyTorch therefore allows users to write their own C ++ extensions that can be compiled and integrated into the PyTorch main code.
    - E.g. the visualization tool

```
cd script
bash build.sh #build C++ code for visualization
bash download.sh #download dataset
```

Walk through: PyTorch Code

- The training process
- The dataset
- The model

```
.
├── LICENSE
├── misc
│   ├── modelnet_id.txt #'modelnet' dataset index
│   ├── num_seg_classes.txt #nums of seg classes
│   └── show3d.png #visualization
├── pointnet
│   ├── dataset.py #dataset process code module
│   ├── __init__.py #kind of python conventions,make this directory discoverable
│   └── model.py #model definition code module
├── README.md
├── scripts
│   ├── build.sh
│   └── download.sh
├── setup.py #Compile the basic operations
└── utils #the code of some basic operations
    ├── render_balls_so.cpp
    ├── show3d_balls.py
    ├── show_cls.py
    ├── show_seg.py
    ├── train_classification.py
    └── train_segmentation.py
```

Walk through: PyTorch Code

- The dataset
  - Take ModelNet dataset used in PointNet as an example.
  - The structure of the dataset building class is shown in the figure.
  - Our data source is the original coordinate information xyz information of the point cloud, which is often saved with these formats, .ply, .pcd, .obj or .xyz. And you can use the software meshlab to visualize .ply, .obj or .xyz. Or use the library named Open3D to visualize .pcd.
  - .ply and .obj can also contain more information like rgb, normal and connections, which make them a 'mesh'.

```
class ModelNetDataset(data.Dataset):
    def __init__(self, ...

    def __getitem__(self, index): ...


    def __len__(self): ...
```

A9-vulcan_aligned.**xyz**

refined_reconstruction_0180.**ply**

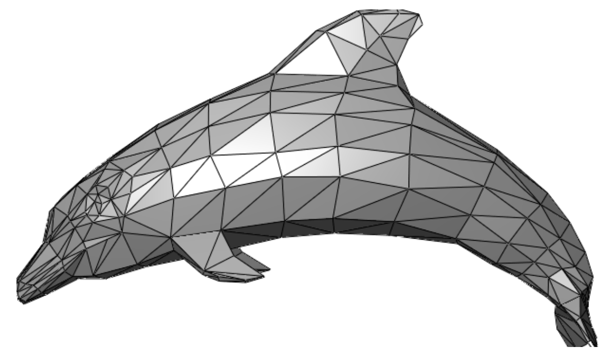
merged_0002.**obj**

pointcloud0.**pcd**

Walk through: PyTorch Code

- Example data structure

Point cloud with color
rendering for segmetation



Polygon mesh data

Walk through: PyTorch Code

- The dataset

```
class ModelNetDataset(data.Dataset):
    def __init__(self,
                 root,
                 npoints=2500,
                 split='train',
                 data_augmentation=True):
        self.npoints = npoints
        self.root = root
        self.split = split
        self.data_augmentation = data_augmentation
        self.fns = []
        with open(os.path.join(root, '{}.txt'.format(self.split)), 'r') as f:
            for line in f:
                self.fns.append(line.strip())

        self.cat = {}
        with open(os.path.join(os.path.dirname(os.path.realpath(__file__)), '../misc/modelnet_id.txt'), 'r') as f:
            for line in f:
                ls = line.strip().split()
                self.cat[ls[0]] = int(ls[1])

        print(self.cat)
        self.classes = list(self.cat.keys())
```

Walk through: PyTorch Code

- The dataset

```
    def __getitem__(self, index):
        fn = self.fns[index]
        cls = self.cat[fn.split('/')[0]]
        with open(os.path.join(self.root, fn), 'rb') as f:
            plydata = PlyData.read(f)
        pts = np.vstack([plydata['vertex']['x'], plydata['vertex']['y'], plydata['vertex']['z']]).T
        choice = np.random.choice(len(pts), self.npoints, replace=True)
        point_set = pts[choice, :]

        point_set = point_set - np.expand_dims(np.mean(point_set, axis=0), 0)  # center
        dist = np.max(np.sqrt(np.sum(point_set ** 2, axis=1)), 0)
        point_set = point_set / dist  # scale

        if self.data_augmentation:
            theta = np.random.uniform(0, np.pi * 2)
            rotation_matrix = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])
            point_set[:, [0, 2]] = point_set[:, [0, 2]].dot(rotation_matrix)  # random rotation
            point_set += np.random.normal(0, 0.02, size=point_set.shape)  # random jitter

        point_set = torch.from_numpy(point_set.astype(np.float32))
        cls = torch.from_numpy(np.array([cls]).astype(np.int64))
        return point_set, cls


    def __len__(self):
        return len(self.fns)
```

Walk through: PyTorch Code

- The model

```python
class PointNetCls(nn.Module):
    def __init__(self, k=2, feature_transform=False):
        super(PointNetCls, self).__init__()
        self.feature_transform = feature_transform
        self.feat = PointNetfeat(global_feat=True, feature_transform=feature_transform)
        self.fc1 = nn.Linear(1024, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, k)
        self.dropout = nn.Dropout(p=0.3)
        self.bn1 = nn.BatchNorm1d(512)
        self.bn2 = nn.BatchNorm1d(256)
        self.relu = nn.ReLU()

    def forward(self, x):
        x, trans, trans_feat = self.feat(x)
        x = F.relu(self.bn1(self.fc1(x)))
        x = F.relu(self.bn2(self.dropout(self.fc2(x))))
        x = self.fc3(x)
        return F.log_softmax(x, dim=1), trans, trans_feat
```

Help links: https://pytorch.org/tutorials/beginner/examples_nn/two_layer_net_module.html

Walk through: PyTorch Code

- The training process
  - Module importing, user parameters parsing and random setting

```python
from __future__ import print_function
import argparse
import os
import random
import torch
import torch.nn.parallel
import torch.optim as optim
import torch.utils.data
from pointnet.dataset import ShapeNetDataset, ModelNetDataset
from pointnet.model import PointNetCls, feature_transform_regularizer
import torch.nn.functional as F
from tqdm import tqdm
```

About the version of python 2.x or 3.x, you can ignore

Parameter parser, which accepts user input parameters

For some system operation like making or changing directories

Generate random numbers. Tdata set needs to be randomly shuffled

The whole pytorch package

A package for multi gpu training

The optimizer package for deep learning in pytorch

The father class of pytorch dataloader, making dataset iterable

function libraries for neural network training

Progress bar visualization

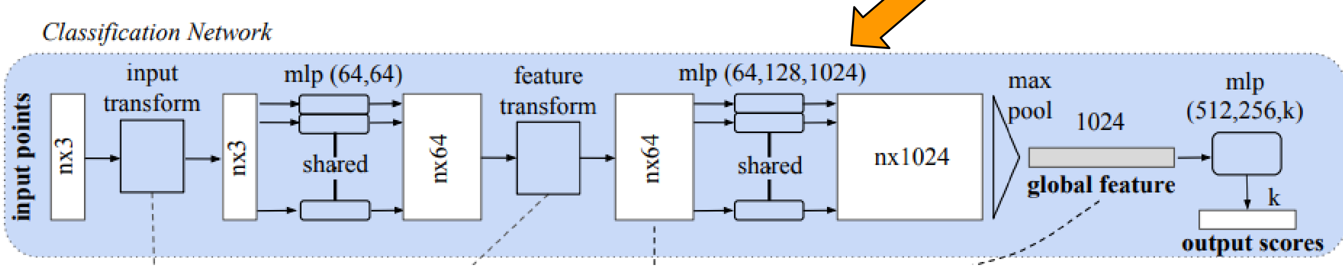
```python
parser = argparse.ArgumentParser()
parser.add_argument(
    '--batchSize', type=int, default=32, help='input batch size')
```

```python
opt.manualSeed = random.randint(1, 10000)
print("Random Seed: ", opt.manualSeed)
random.seed(opt.manualSeed)
torch.manual_seed(opt.manualSeed)
```

```python
class ShapeNetDataset(data.Dataset): ...

class ModelNetDataset(data.Dataset): ...
```

```python
class PointNetCls(nn.Module): ...

def feature_transform_regularizer(trans): ...
```

Corresponding to the module in the Paper (PointNet)



Walk through: PyTorch Code

- The training process
  - 2. Dataloader

```python
if opt.dataset_type == 'shapenet':
    dataset = ShapeNetDataset(
        root=opt.dataset,
        classification=True,
        npoints=opt.num_points)

    test_dataset = ShapeNetDataset(
        root=opt.dataset,
        classification=True,
        split='test',
        npoints=opt.num_points,
        data_augmentation=False)
elif opt.dataset_type == 'modelnet40':
    dataset = ModelNetDataset(
        root=opt.dataset,
        npoints=opt.num_points,
        split='trainval')

    test_dataset = ModelNetDataset(
        root=opt.dataset,
        split='test',
        npoints=opt.num_points,
        data_augmentation=False)
else:
    exit('wrong dataset type')
```

```python
dataloader = torch.utils.data.DataLoader(
    dataset,
    batch_size=opt.batchSize,
    shuffle=True,
    num_workers=int(opt.workers))

testdataloader = torch.utils.data.DataLoader(
        test_dataset,
        batch_size=opt.batchSize,
        shuffle=True,
        num_workers=int(opt.workers))
```

Walk through: PyTorch Code

- The training process
  - 3. The model building (in this code example the model a classifier)

```
classifier = PointNetCls(k=num_classes, feature_transform=opt.feature_transform)

if opt.model != '':
    classifier.load_state_dict(torch.load(opt.model))


optimizer = optim.Adam(classifier.parameters(), lr=0.001, betas=(0.9, 0.999))
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=20, gamma=0.5)
classifier.cuda()
```

Walk through: PyTorch Code

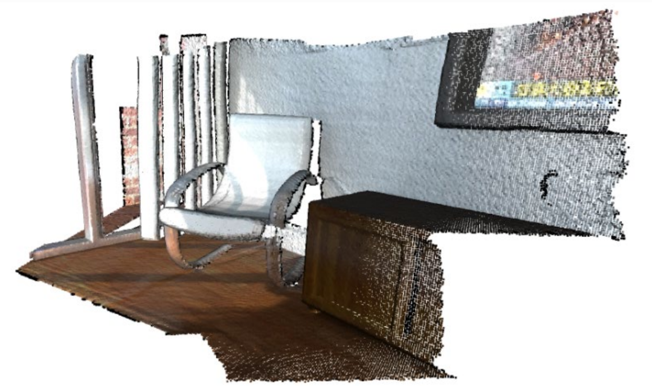
- The training process
  - 4. The training loop & saving the weights obtained by training

```
# main train loop for specific epoch
for epoch in range(opt.nepoch):
    scheduler.step()
    for i, data in enumerate(dataloader, 0):
        points, target = data
        target = target[:, 0]
        points = points.transpose(2, 1)
        points, target = points.cuda(), target.cuda()
        optimizer.zero_grad()
        classifier = classifier.train()
        pred, trans, trans_feat = classifier(points)
        loss = F.nll_loss(pred, target)
        if opt.feature_transform:
            loss += feature_transform_regularizer(trans_feat) * 0.001
        loss.backward()
        optimizer.step()
        pred_choice = pred.data.max(1)[1]
        correct = pred_choice.eq(target.data).cpu().sum()
        print('[%d: %d/%d] train loss: %f accuracy: %f' % (epoch, i, num_batch, loss.item(), correct.item() / float(opt.batchSize)))

        if i % 10 == 0:
            j, data = next(enumerate(testdataloader, 0))
            points, target = data
            target = target[:, 0]
            points = points.transpose(2, 1)
            points, target = points.cuda(), target.cuda()
            classifier = classifier.eval()
            pred, _, _ = classifier(points)
            loss = F.nll_loss(pred, target)
            pred_choice = pred.data.max(1)[1]
            correct = pred_choice.eq(target.data).cpu().sum()
            print('[%d: %d/%d] %s loss: %f accuracy: %f' % (epoch, i, num_batch, blue('test'), loss.item(), correct.item()/float(opt.batchSize)))

    torch.save(classifier.state_dict(), '%s/cls_model_%d.pth' % (opt.outf, epoch))
```

Auxiliary libraries

- Open3D
  - an open-source library that supports rapid development of software that deals with 3D data
  - based on C++ but can be used in python api

```
print("Load a ply point cloud, print it, and render it")
pcd = o3d.io.read_point_cloud("../../test_data/fragment.ply")
o3d.visualization.draw_geometries([pcd],
                                  zoom=0.3412,
                                  front=[0.4257, -0.2125, -0.8795],
                                  lookat=[2.6172, 2.0475, 1.532],
                                  up=[-0.0694, -0.9768, 0.2024])
```



- PyTorch Geometric
  - a geometric deep learning extension library for PyTorch.
  - it consists of various methods for deep learning on graphs and other irregular structures, also known as geometric deep learning, from a variety of published papers.

```
import torch
import torch.nn.functional as F
from torch_geometric.nn import GCNConv
```