Implementation of DGCNN Model using PyTorch Geometric and Weights & Biases

This notebook demonstrates an implementation of the [Dynamic Graph CNN]() for point cloud segmnetation implemented using [PyTorch Geometric]() and experiment tracked and visualized using [Weights & Biases](). The code here is inspired by [this]() original implementation.

Install Required Packages

```
import os
import torch
os.environ['TORCH'] = torch.__version__
print(torch.__version__)
```

```
2.1.0+cu121
```

```
!pip install -q torch-scatter -f https://data.pyg.org/whl/torch-${TORCH}.html
!pip install -q torch-sparse -f https://data.pyg.org/whl/torch-${TORCH}.html
!pip install -q torch-cluster -f https://data.pyg.org/whl/torch-${TORCH}.html
!pip install -q git+https://github.com/pyg-team/pytorch_geometric.git
!pip install -q wandb
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.8/10.8 MB 43.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.0/5.0 MB 59.9 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.3/3.3 MB 29.1 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
  Building wheel for torch_geometric (pyproject.toml) ... done
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.1/2.1 MB 10.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 190.6/190.6 kB 12.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 254.1/254.1 kB 15.0 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.7/62.7 kB 8.0 MB/s eta 0:00:00
```

Import Libraries

```
!pip install torchmetrics
```

```
Collecting torchmetrics
  Downloading torchmetrics-1.2.1-py3-none-any.whl (806 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 806.1/806.1 kB 9.0 MB/s eta 0:00:00
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (1.23.5)
Requirement already satisfied: packaging>17.1 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (23.2)
Requirement already satisfied: torch>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (2.1.0+cu121)
Collecting lightning-utilities>=0.8.0 (from torchmetrics)
  Downloading lightning_utilities-0.10.0-py3-none-any.whl (24 kB)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.8.0->torchmetrics
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.8.0->torchr
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetrics) (3.13.1)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetrics) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetrics) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetrics) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetrics) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetrics) (2.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.8.1->torchmetrics)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.8.1->torchmetrics) (1.:
Installing collected packages: lightning-utilities, torchmetrics
Successfully installed lightning-utilities-0.10.0 torchmetrics-1.2.1
```

```
import os
import wandb
import random
import numpy as np
from tqdm.auto import tqdm

import torch
import torch.nn.functional as F

from torch_scatter import scatter
from torchmetrics.functional import jaccard_index

import torch_geometric.transforms as T
from torch_geometric.datasets import ShapeNet
from torch_geometric.loader import DataLoader
from torch_geometric.nn import MLP, DynamicEdgeConv
```

Initialize Weights & Biases

We need to call `wandb.init()` once at the beginning of our program to initialize a new job. This creates a new run in W&B and launches a background process to sync data.

```
wandb_project = "pyg-point-cloud" #@param {"type": "string"}
wandb_run_name = "train-dgcnn" #@param {"type": "string"}

wandb.init(project=wandb_project, name=wandb_run_name, job_type="tr

config = wandb.config

config.seed = 42
config.device = 'cuda' if torch.cuda.is_available() else 'cpu'

random.seed(config.seed)
torch.manual_seed(config.seed)
device = torch.device(config.device)

config.category = 'Airplane' #@param ["Bag", "Cap", "Car", "Chair",
config.random_jitter_translation = 2e-2
config.random_rotation_interval_x = 15
config.random_rotation_interval_y = 15
config.random_rotation_interval_z = 15
config.validation_split = 0.2  # 80-20 split
config.batch_size = 16
config.num_workers = 6

config.num_nearest_neighbours = 50
config.aggregation_operator = "max"
config.dropout = 0.5
config.initial_lr = 1e-3
config.lr_scheduler_step_size = 5
config.gamma = 0.8

config.epochs = 8
```

wandb_project: " pyg-point-cloud "

wandb_run_name: " train-dgcnn "

config.category: Bag

"Airplane" is not an allowed value for "config.category".

wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
Tracking run with wandb version 0.16.1
Run data is saved locally in `/content/wandb/run-20240102_052347-jcetxf6f`
Syncing run **train-dgcnn** to Weights & Biases (docs)
View project at https://wandb.ai/team-dubakur/pyg-point-cloud
View run at https://wandb.ai/team-dubakur/pyg-point-cloud/runs/jcetxf6f

Load ShapeNet Dataset using PyTorch Geometric

We now load, preprocess and batch the ModelNet dataset for training, validation/testing and visualization.

```
transform = T.Compose([
    T.RandomJitter(config.random_jitter_translation),
    T.RandomRotate(config.random_rotation_interval_x, axis=0),
    T.RandomRotate(config.random_rotation_interval_y, axis=1),
    T.RandomRotate(config.random_rotation_interval_z, axis=2)
])
pre_transform = T.NormalizeScale()


dataset_path = os.path.join('ShapeNet', config.category)

train_val_dataset = ShapeNet(
    dataset_path, config.category, split='trainval',
    transform=transform, pre_transform=pre_transform
)
```

```
Downloading https://shapenet.cs.stanford.edu/media/shapenetcore_partanno_segmentation_benchmark_v0_normal.zip
Extracting ShapeNet/Airplane/shapenetcore_partanno_segmentation_benchmark_v0_normal.zip
Processing...
Done!
```

Now, we need to offset the segmentation labels

```
segmentation_class_frequency = {}
for idx in tqdm(range(len(train_val_dataset))):
    pc_viz = train_val_dataset[idx].pos.numpy().tolist()
    segmentation_label = train_val_dataset[idx].y.numpy().tolist()
    for label in set(segmentation_label):
```

```
        segmentation_class_frequency[label] = segmentation_label.count(label)
class_offset = min(list(segmentation_class_frequency.keys()))
print("Class Offset:", class_offset)

for idx in range(len(train_val_dataset)):
    train_val_dataset[idx].y -= class_offset
```

100% 2349/2349 [00:09<00:00, 297.86it/s]

```
Class Offset: 0
```

```
num_train_examples = int((1 - config.validation_split) * len(train_val_dataset))
train_dataset = train_val_dataset[:num_train_examples]
val_dataset = train_val_dataset[num_train_examples:]


train_loader = DataLoader(
    train_dataset, batch_size=config.batch_size,
    shuffle=True, num_workers=config.num_workers
)
val_loader = DataLoader(
    val_dataset, batch_size=config.batch_size,
    shuffle=False, num_workers=config.num_workers
)
visualization_loader = DataLoader(
    val_dataset[:10], batch_size=1,
    shuffle=False, num_workers=config.num_workers
)
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 6 worker proces
  warnings.warn(_create_warning_msg(
```

Implementing the DGCNN Model using PyTorch Geometric

```
class DGCNN(torch.nn.Module):
    def __init__(self, out_channels, k=30, aggr='max'):
        super().__init__()

        self.conv1 = DynamicEdgeConv(MLP([2 * 6, 64, 64]), k, aggr)
        self.conv2 = DynamicEdgeConv(MLP([2 * 64, 64, 64]), k, aggr)
        self.conv3 = DynamicEdgeConv(MLP([2 * 64, 64, 64]), k, aggr)

        self.mlp = MLP(
            [3 * 64, 1024, 256, 128, out_channels],
            dropout=0.5, norm=None
        )

    def forward(self, data):
        x, pos, batch = data.x, data.pos, data.batch
        x0 = torch.cat([x, pos], dim=-1)

        x1 = self.conv1(x0, batch)
        x2 = self.conv2(x1, batch)
        x3 = self.conv3(x2, batch)

        out = self.mlp(torch.cat([x1, x2, x3], dim=1))
        return F.log_softmax(out, dim=1)


config.num_classes = train_dataset.num_classes

model = DGCNN(
    out_channels=train_dataset.num_classes,
    k=config.num_nearest_neighbours,
    aggr=config.aggregation_operator
).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=config.initial_lr)
scheduler = torch.optim.lr_scheduler.StepLR(
    optimizer, step_size=config.lr_scheduler_step_size, gamma=config.gamma
)
```

Training DGCNN and Logging Metrics on Weights & Biases

```python
def train_step(epoch):
    model.train()

    ious, categories = [], []
    total_loss = correct_nodes = total_nodes = 0
    y_map = torch.empty(
        train_loader.dataset.num_classes, device=device
    ).long()
    num_train_examples = len(train_loader)

    progress_bar = tqdm(
        train_loader, desc=f"Training Epoch {epoch}/{config.epochs}"
    )

    for data in progress_bar:
        data = data.to(device)

        optimizer.zero_grad()
        outs = model(data)
        loss = F.nll_loss(outs, data.y)
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

        correct_nodes += outs.argmax(dim=1).eq(data.y).sum().item()
        total_nodes += data.num_nodes

        sizes = (data.ptr[1:] - data.ptr[:-1]).tolist()
        for out, y, category in zip(outs.split(sizes), data.y.split(sizes),
                                    data.category.tolist()):
            category = list(ShapeNet.seg_classes.keys())[category]
            part = ShapeNet.seg_classes[category]
            part = torch.tensor(part, device=device)

            y_map[part] = torch.arange(part.size(0), device=device)

            iou = jaccard_index(
                out[:, part].argmax(dim=-1), y_map[y],
                task="multiclass", num_classes=part.size(0)
            )
            ious.append(iou)

        categories.append(data.category)

    iou = torch.tensor(ious, device=device)
    category = torch.cat(categories, dim=0)
    mean_iou = float(scatter(iou, category, reduce='mean').mean())

    return {
        "Train/Loss": total_loss / num_train_examples,
        "Train/Accuracy": correct_nodes / total_nodes,
        "Train/IoU": mean_iou
    }
```

```python
@torch.no_grad()
def val_step(epoch):
    model.eval()

    ious, categories = [], []
    total_loss = correct_nodes = total_nodes = 0
    y_map = torch.empty(
        val_loader.dataset.num_classes, device=device
    ).long()
    num_val_examples = len(val_loader)

    progress_bar = tqdm(
        val_loader, desc=f"Validating Epoch {epoch}/{config.epochs}"
    )

    for data in progress_bar:
        data = data.to(device)
        outs = model(data)

        loss = F.nll_loss(outs, data.y)
        total_loss += loss.item()

        correct_nodes += outs.argmax(dim=1).eq(data.y).sum().item()
        total_nodes += data.num_nodes

        sizes = (data.ptr[1:] - data.ptr[:-1]).tolist()
        for out, y, category in zip(outs.split(sizes), data.y.split(sizes),
                                    data.category.tolist()):
            category = list(ShapeNet.seg_classes.keys())[category]
            part = ShapeNet.seg_classes[category]
            part = torch.tensor(part, device=device)

            y_map[part] = torch.arange(part.size(0), device=device)

            iou = jaccard_index(
                out[:, part].argmax(dim=-1), y_map[y],
                task="multiclass", num_classes=part.size(0)
            )
            ious.append(iou)

        categories.append(data.category)

    iou = torch.tensor(ious, device=device)
    category = torch.cat(categories, dim=0)
    mean_iou = float(scatter(iou, category, reduce='mean').mean())

    return {
        "Validation/Loss": total_loss / num_val_examples,
        "Validation/Accuracy": correct_nodes / total_nodes,
        "Validation/IoU": mean_iou
    }
```

```python
@torch.no_grad()
def visualization_step(epoch, table):
    model.eval()
    for data in tqdm(visualization_loader):
        data = data.to(device)
        outs = model(data)

        predicted_labels = outs.argmax(dim=1)
        accuracy = predicted_labels.eq(data.y).sum().item() / data.num_nodes

        sizes = (data.ptr[1:] - data.ptr[:-1]).tolist()
        ious, categories = [], []
        y_map = torch.empty(
            visualization_loader.dataset.num_classes, device=device
        ).long()
        for out, y, category in zip(
            outs.split(sizes), data.y.split(sizes), data.category.tolist()
        ):
            category = list(ShapeNet.seg_classes.keys())[category]
            part = ShapeNet.seg_classes[category]
            part = torch.tensor(part, device=device)
            y_map[part] = torch.arange(part.size(0), device=device)
            iou = jaccard_index(
                out[:, part].argmax(dim=-1), y_map[y],
                task="multiclass", num_classes=part.size(0)
            )
            ious.append(iou)
        categories.append(data.category)
        iou = torch.tensor(ious, device=device)
        category = torch.cat(categories, dim=0)
        mean_iou = float(scatter(iou, category, reduce='mean').mean())

        gt_pc_viz = data.pos.cpu().numpy().tolist()
        segmentation_label = data.y.cpu().numpy().tolist()
        frequency_dict = {key: 0 for key in segmentation_class_frequency.keys()}
        for label in set(segmentation_label):
            frequency_dict[label] = segmentation_label.count(label)
        for j in range(len(gt_pc_viz)):
            # gt_pc_viz[j] += [segmentation_label[j] + 1 - class_offset]
            gt_pc_viz[j] += [segmentation_label[j] + 1]

        predicted_pc_viz = data.pos.cpu().numpy().tolist()
        segmentation_label = data.y.cpu().numpy().tolist()
        frequency_dict = {key: 0 for key in segmentation_class_frequency.keys()}
        for label in set(segmentation_label):
            frequency_dict[label] = segmentation_label.count(label)
        for j in range(len(predicted_pc_viz)):
            # predicted_pc_viz[j] += [segmentation_label[j] + 1 - class_offset]
            predicted_pc_viz[j] += [segmentation_label[j] + 1]

        table.add_data(
            epoch, wandb.Object3D(np.array(gt_pc_viz)),
            wandb.Object3D(np.array(predicted_pc_viz)),
            accuracy, mean_iou
        )

    return table


def save_checkpoint(epoch):
    """Save model checkpoints as Weights & Biases artifacts"""
    torch.save({
        'epoch': epoch,
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict()
    }, "checkpoint.pt")

    artifact_name = wandb.util.make_artifact_name_safe(
        f"{wandb.run.name}-{wandb.run.id}-checkpoint"
    )

    checkpoint_artifact = wandb.Artifact(artifact_name, type="checkpoint")
    checkpoint_artifact.add_file("checkpoint.pt")
    wandb.log_artifact(
        checkpoint_artifact, aliases=["latest", f"epoch-{epoch}"]
    )
```

```python
table = wandb.Table(columns=["Epoch", "Ground-Truth", "Prediction", "Accuracy", "IoU"])

for epoch in range(1, config.epochs + 1):
    train_metrics = train_step(epoch)
    val_metrics = val_step(epoch)

    metrics = {**train_metrics, **val_metrics}
    metrics["learning_rate"] = scheduler.get_last_lr()[-1]
    wandb.log(metrics)

    table = visualization_step(epoch, table)

    scheduler.step()
    save_checkpoint(epoch)

wandb.log({"Evaluation": table})
```

Training Epoch 1/8: 100% 118/118 [06:29<00:00, 2.72s/it]

Validating Epoch 1/8: 100% 30/30 [01:30<00:00, 2.48s/it]

100% 10/10 [01:05<00:00, 1.32s/it]

Training Epoch 2/8: 100% 118/118 [06:28<00:00, 2.72s/it]

Validating Epoch 2/8: 100% 30/30 [01:30<00:00, 2.48s/it]

```
Exception ignored in: <function _MultiProcessingDataLoaderIter.__del__ at 0x7bccdac1e
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py", line
    self._shutdown_workers()Exception ignored in:
```