Jets as graphs

April 1, 2024

[1]:
```
import os
import pickle
import random

import h5py
import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch_geometric.transforms as T
from types import SimpleNamespace
from sklearn.neighbors import NearestNeighbors
from torch.nn import Linear
from torch.optim.lr_scheduler import StepLR
from torch.utils.tensorboard import SummaryWriter
from torch_geometric.data import Data
from torch_geometric.loader import DataLoader
from torch_geometric.nn import (MLP, GCNConv, GINConv, global_add_pool,
                                global_mean_pool)
from tqdm import tqdm
```

[2]:
```
def set_seed(seed=42):
    print("Setting seed:", seed)
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True


def split(input_list, proportions=[0.8, 0.1, 0.1]):
    random.shuffle(input_list)
    total_length = len(input_list)
    lengths = [int(total_length * proportion) for proportion in proportions]

    parts = []
```

```
    start_idx = 0
    for length in lengths:
        parts.append(input_list[start_idx : start_idx + length])
        start_idx += length

    return parts


def evaluate_model(model, dataloader, criterion, device):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
        for data in dataloader:
            inputs, targets = data.x.to(device), data.y.to(device)
            outputs = model(
                inputs, data.edge_index.to(device), data.batch.to(device),␣
 ↪len(targets)
            )
            loss = criterion(outputs, targets)
            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += targets.size(0)
            correct += (predicted == targets).sum().item()

    loss = running_loss / len(dataloader)
    accuracy = correct / total

    return loss, accuracy
```

[3]:
```
class GIN(torch.nn.Module):
    """https://github.com/pyg-team/pytorch_geometric/blob/master/examples/
 ↪compile/gin.py"""

    def __init__(self, in_channels, hidden_channels, out_channels, num_layers):
        super().__init__()

        self.convs = torch.nn.ModuleList()
        for _ in range(num_layers):
            mlp = MLP([in_channels, hidden_channels, hidden_channels])
            self.convs.append(GINConv(nn=mlp, train_eps=False))
            in_channels = hidden_channels

        self.mlp = MLP(
```

```
            [hidden_channels, hidden_channels, out_channels], norm=None,␣
↪dropout=0.5
        )

    def forward(self, x, edge_index, batch, batch_size):
        for conv in self.convs:
            x = conv(x, edge_index).relu()
        # Pass the batch size to avoid CPU communication/graph breaks:
        x = global_add_pool(x, batch, size=batch_size)
        return self.mlp(x)


class GCN(torch.nn.Module):
    """https://colab.research.google.com/github/wandb/examples/blob/pyg/
↪graph-classification/colabs/pyg/Graph_Classification_with_PyG_and_W%26B.
↪ipynb"""

    def __init__(self, hidden_channels, inp, out, drop=0.5):
        super(GCN, self).__init__()
        torch.manual_seed(12345)
        self.drop = drop
        self.conv1 = GCNConv(inp, hidden_channels)
        self.conv2 = GCNConv(hidden_channels, hidden_channels)
        self.conv3 = GCNConv(hidden_channels, hidden_channels)
        self.lin = Linear(hidden_channels, out)

    def forward(self, x, edge_index, batch):
        # 1. Obtain node embeddings
        x = self.conv1(x, edge_index)
        x = x.relu()
        x = self.conv2(x, edge_index)
        x = x.relu()
        x = self.conv3(x, edge_index)

        # 2. Readout layer
        x = global_mean_pool(x, batch)  # [batch_size, hidden_channels]

        # 3. Apply a final classifier
        x = F.dropout(x, p=self.drop, training=self.training)
        x = self.lin(x)

        return x
```

[4]:
```
def run(args):
    print(args)

    writer = SummaryWriter()
```

```
    set_seed(args.seed)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    processed_path = os.path.expanduser(
        "~/data/quark-gluon_"
        + str(args.dry_run)
        + "_"
        + str(args.normalize)
        + ".pkl"
    )

    if os.path.exists(processed_path):
        with open(processed_path, "rb") as f:
            dataset = pickle.load(f)
        print("Data loaded successfully.")
    else:
        with h5py.File(
            os.path.expanduser("~/data/quark-gluon_data-set_n139306.hdf5"),
            "r",
        ) as f:
            X = np.array(f["X_jets"])
            y = np.array(f["y"])
            if args.dry_run:
                X = X[:40000]
                y = y[:40000]

        dataset = []
        for i, img in tqdm(enumerate(X)):
            non_black_pixels = np.where(img.sum(axis=2) > 0)
            x_coords, y_coords = non_black_pixels

            coords = np.vstack((x_coords, y_coords)).T
            node_features = img[x_coords, y_coords]
            nbrs = NearestNeighbors(n_neighbors=args.k, algorithm="auto").
↪fit(coords)
            _, indices = nbrs.kneighbors(coords)

            edge_index = []
            for j in range(len(coords)):
                for neighbor_idx in indices[j]:
                    # if j != neighbor_idx:  # Exclude self-loops
                    edge_index.append((j, neighbor_idx))

            edge_index = torch.tensor(edge_index, dtype=torch.long).t()

            data = Data(
                x=torch.tensor(node_features, dtype=torch.float),
```

```
                edge_index=edge_index,
                y=torch.tensor(int(y[i])),
            )

            if args.normalize:
                data = T.NormalizeFeatures()(data)

            dataset.append(data)

        # Pickle the list to the file
        with open(processed_path, "wb") as f:
            pickle.dump(dataset, f)

    if args.dry_run:
        dataset = split(dataset, [0.33, 0.33, 0.33])
    else:
        dataset = split(dataset)

    train_loader = DataLoader(dataset[0], batch_size=args.batch_size,␣
↪shuffle=True)
    test_loader = DataLoader(dataset[1], batch_size=args.batch_size,␣
↪shuffle=False)
    val_loader = DataLoader(dataset[2], batch_size=args.batch_size,␣
↪shuffle=False)

    model = GIN(
        in_channels=3,
        hidden_channels=args.hidden,
        out_channels=2,
        num_layers=args.layers,
    ).to(device)

    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(
        model.parameters(), lr=args.lr, weight_decay=args.weight_decay
    )
    scheduler = StepLR(optimizer, step_size=args.step_size, gamma=args.gamma)

    early_stop_thresh = args.step_size * 2
    best_accuracy = -1
    best_epoch = -1

    for epoch in range(args.epochs):
        writer.add_scalar("epoch", epoch)
        model.train()
        running_loss = 0.0
        correct = 0
```

```
        total = 0

        for data in train_loader:
            inputs, targets = data.x.to(device), data.y.to(device)

            optimizer.zero_grad()
            outputs = model(
                inputs, data.edge_index.to(device), data.batch.to(device),␣
↪len(targets)
            )
            loss = criterion(outputs, targets)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(outputs, 1)
            total += targets.size(0)
            correct += (predicted == targets).sum().item()

        train_loss = running_loss / len(train_loader)
        train_acc = correct / total

        val_loss, val_acc = evaluate_model(model, val_loader, criterion, device)
        writer.add_scalar("lr", optimizer.param_groups[0]["lr"])
        scheduler.step()
        writer.add_scalar("train_acc", train_acc)
        writer.add_scalar("train_loss", train_loss)
        writer.add_scalar("val_acc", val_acc)
        writer.add_scalar("val_loss", val_loss)
        print(
            f"Epoch [{epoch + 1}/{args.epochs}], "
            f"Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.4f}, "
            f"Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.4f}, "
        )

        if val_acc > best_accuracy:
            best_accuracy = val_acc
            best_epoch = epoch
            torch.save(
                {
                    "model_state_dict": model.state_dict()
                },
                "best_model.pth",
            )

        elif epoch - best_epoch > early_stop_thresh:
            print("Early stopped training at epoch %d" % (epoch + 1))
```

```
            break

    checkpoint = torch.load("best_model.pth")
    model.load_state_dict(checkpoint["model_state_dict"])

    test_loss, test_acc = evaluate_model(model, test_loader, criterion, device)
    print(f"Accuracy: {test_acc:.4f}")
    writer.add_scalar("accuracy", test_acc)
    writer.add_scalar("loss", test_loss)
    writer.close()
    return test_acc
```

[5]:
```
args = SimpleNamespace(
        epochs=300,
        k=2,
        lr=0.005,
        batch_size=256,
        weight_decay=1.0e-05,
        hidden=256,
        dry_run=False,
        drop=0.5,
        step_size=10,
        gamma=0.5,
        normalize=True,
        layers=5,
        seed=42
    )
run(args)
```

```
namespace(epochs=300, k=2, lr=0.005, batch_size=256, weight_decay=1e-05,
hidden=256, dry_run=False, drop=0.5, step_size=10, gamma=0.5, normalize=True,
layers=5, seed=42)
Setting seed: 42
Data loaded successfully.
Epoch [1/300], Train Loss: 0.9811, Train Acc: 0.6816, Val Loss: 0.6317, Val Acc:
0.6889,
Epoch [2/300], Train Loss: 0.5984, Train Acc: 0.6972, Val Loss: 0.6862, Val Acc:
0.6667,
Epoch [3/300], Train Loss: 0.5919, Train Acc: 0.7024, Val Loss: 0.5839, Val Acc:
0.7098,
Epoch [4/300], Train Loss: 0.5870, Train Acc: 0.7036, Val Loss: 0.6414, Val Acc:
0.6634,
Epoch [5/300], Train Loss: 0.5903, Train Acc: 0.7020, Val Loss: 0.6062, Val Acc:
0.6785,
Epoch [6/300], Train Loss: 0.5883, Train Acc: 0.7038, Val Loss: 0.6230, Val Acc:
0.6859,
Epoch [7/300], Train Loss: 0.5885, Train Acc: 0.7034, Val Loss: 0.5769, Val Acc:
```

0.7098,
Epoch [8/300], Train Loss: 0.5866, Train Acc: 0.7023, Val Loss: 0.5828, Val Acc:
0.7028,
Epoch [9/300], Train Loss: 0.5846, Train Acc: 0.7062, Val Loss: 0.5947, Val Acc:
0.6875,
Epoch [10/300], Train Loss: 0.5818, Train Acc: 0.7063, Val Loss: 0.5892, Val
Acc: 0.6984,
Epoch [11/300], Train Loss: 0.5773, Train Acc: 0.7102, Val Loss: 0.5794, Val
Acc: 0.7035,
Epoch [12/300], Train Loss: 0.5779, Train Acc: 0.7110, Val Loss: 0.5711, Val
Acc: 0.7164,
Epoch [13/300], Train Loss: 0.5776, Train Acc: 0.7098, Val Loss: 0.5946, Val
Acc: 0.6942,
Epoch [14/300], Train Loss: 0.5757, Train Acc: 0.7114, Val Loss: 0.9035, Val
Acc: 0.5338,
Epoch [15/300], Train Loss: 0.5773, Train Acc: 0.7101, Val Loss: 0.5671, Val
Acc: 0.7182,
Epoch [16/300], Train Loss: 0.5763, Train Acc: 0.7103, Val Loss: 0.6627, Val
Acc: 0.6753,
Epoch [17/300], Train Loss: 0.5778, Train Acc: 0.7108, Val Loss: 0.5711, Val
Acc: 0.7130,
Epoch [18/300], Train Loss: 0.5780, Train Acc: 0.7115, Val Loss: 0.5697, Val
Acc: 0.7136,
Epoch [19/300], Train Loss: 0.5765, Train Acc: 0.7111, Val Loss: 0.5868, Val
Acc: 0.6965,
Epoch [20/300], Train Loss: 0.5748, Train Acc: 0.7121, Val Loss: 0.5702, Val
Acc: 0.7171,
Epoch [21/300], Train Loss: 0.5719, Train Acc: 0.7146, Val Loss: 0.5707, Val
Acc: 0.7131,
Epoch [22/300], Train Loss: 0.5724, Train Acc: 0.7141, Val Loss: 0.5638, Val
Acc: 0.7186,
Epoch [23/300], Train Loss: 0.5707, Train Acc: 0.7152, Val Loss: 0.5994, Val
Acc: 0.6848,
Epoch [24/300], Train Loss: 0.5709, Train Acc: 0.7146, Val Loss: 0.5902, Val
Acc: 0.6925,
Epoch [25/300], Train Loss: 0.5712, Train Acc: 0.7132, Val Loss: 0.6296, Val
Acc: 0.6679,
Epoch [26/300], Train Loss: 0.5709, Train Acc: 0.7150, Val Loss: 0.5693, Val
Acc: 0.7137,
Epoch [27/300], Train Loss: 0.5703, Train Acc: 0.7153, Val Loss: 0.5681, Val
Acc: 0.7170,
Epoch [28/300], Train Loss: 0.5711, Train Acc: 0.7145, Val Loss: 0.5683, Val
Acc: 0.7153,
Epoch [29/300], Train Loss: 0.5702, Train Acc: 0.7139, Val Loss: 0.5755, Val
Acc: 0.7062,
Epoch [30/300], Train Loss: 0.5703, Train Acc: 0.7157, Val Loss: 0.5650, Val
Acc: 0.7154,
Epoch [31/300], Train Loss: 0.5679, Train Acc: 0.7165, Val Loss: 0.6139, Val

Acc: 0.6753,
Epoch [32/300], Train Loss: 0.5683, Train Acc: 0.7172, Val Loss: 0.5648, Val Acc: 0.7176,
Epoch [33/300], Train Loss: 0.5676, Train Acc: 0.7155, Val Loss: 0.5737, Val Acc: 0.7105,
Epoch [34/300], Train Loss: 0.5671, Train Acc: 0.7170, Val Loss: 0.5677, Val Acc: 0.7173,
Epoch [35/300], Train Loss: 0.5671, Train Acc: 0.7169, Val Loss: 0.5730, Val Acc: 0.7125,
Epoch [36/300], Train Loss: 0.5668, Train Acc: 0.7170, Val Loss: 0.5625, Val Acc: 0.7182,
Epoch [37/300], Train Loss: 0.5677, Train Acc: 0.7166, Val Loss: 0.5641, Val Acc: 0.7181,
Epoch [38/300], Train Loss: 0.5669, Train Acc: 0.7173, Val Loss: 0.5629, Val Acc: 0.7217,
Epoch [39/300], Train Loss: 0.5658, Train Acc: 0.7190, Val Loss: 0.5719, Val Acc: 0.7128,
Epoch [40/300], Train Loss: 0.5658, Train Acc: 0.7185, Val Loss: 0.5633, Val Acc: 0.7200,
Epoch [41/300], Train Loss: 0.5650, Train Acc: 0.7186, Val Loss: 0.5603, Val Acc: 0.7232,
Epoch [42/300], Train Loss: 0.5648, Train Acc: 0.7196, Val Loss: 0.5598, Val Acc: 0.7219,
Epoch [43/300], Train Loss: 0.5642, Train Acc: 0.7187, Val Loss: 0.5619, Val Acc: 0.7212,
Epoch [44/300], Train Loss: 0.5643, Train Acc: 0.7203, Val Loss: 0.5662, Val Acc: 0.7127,
Epoch [45/300], Train Loss: 0.5635, Train Acc: 0.7196, Val Loss: 0.5753, Val Acc: 0.7108,
Epoch [46/300], Train Loss: 0.5634, Train Acc: 0.7202, Val Loss: 0.5628, Val Acc: 0.7211,
Epoch [47/300], Train Loss: 0.5644, Train Acc: 0.7190, Val Loss: 0.5597, Val Acc: 0.7220,
Epoch [48/300], Train Loss: 0.5640, Train Acc: 0.7195, Val Loss: 0.5597, Val Acc: 0.7206,
Epoch [49/300], Train Loss: 0.5632, Train Acc: 0.7198, Val Loss: 0.5594, Val Acc: 0.7229,
Epoch [50/300], Train Loss: 0.5638, Train Acc: 0.7202, Val Loss: 0.5715, Val Acc: 0.7101,
Epoch [51/300], Train Loss: 0.5628, Train Acc: 0.7196, Val Loss: 0.5598, Val Acc: 0.7211,
Epoch [52/300], Train Loss: 0.5622, Train Acc: 0.7204, Val Loss: 0.5597, Val Acc: 0.7210,
Epoch [53/300], Train Loss: 0.5621, Train Acc: 0.7210, Val Loss: 0.5594, Val Acc: 0.7232,
Epoch [54/300], Train Loss: 0.5626, Train Acc: 0.7202, Val Loss: 0.5586, Val Acc: 0.7218,
Epoch [55/300], Train Loss: 0.5623, Train Acc: 0.7204, Val Loss: 0.5602, Val

```
Acc: 0.7206,
Epoch [56/300], Train Loss: 0.5624, Train Acc: 0.7209, Val Loss: 0.5589, Val
Acc: 0.7223,
Epoch [57/300], Train Loss: 0.5624, Train Acc: 0.7205, Val Loss: 0.5605, Val
Acc: 0.7225,
Epoch [58/300], Train Loss: 0.5625, Train Acc: 0.7206, Val Loss: 0.5669, Val
Acc: 0.7153,
Epoch [59/300], Train Loss: 0.5615, Train Acc: 0.7200, Val Loss: 0.5604, Val
Acc: 0.7218,
Epoch [60/300], Train Loss: 0.5619, Train Acc: 0.7212, Val Loss: 0.5648, Val
Acc: 0.7187,
Epoch [61/300], Train Loss: 0.5613, Train Acc: 0.7208, Val Loss: 0.5589, Val
Acc: 0.7211,
Epoch [62/300], Train Loss: 0.5605, Train Acc: 0.7216, Val Loss: 0.5590, Val
Acc: 0.7225,
Early stopped training at epoch 62
Accuracy: 0.7255
```

[5]: 0.7254845656855707

[ ]: