gatconv

March 26, 2024

[1]:
```
import torch
import numpy as np
import pandas as pd
import pyarrow.parquet as pq
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

import torch
import torch.nn as nn
from torch.utils.data import Dataset,DataLoader, random_split
import torch.nn.functional as F
from torchvision import models

import torch.optim as optim
from tqdm import tqdm

from sklearn.metrics import roc_auc_score, confusion_matrix ,roc_curve
import seaborn as sns

from torch.nn import Linear
```

[2]:
```
!pip install torch_geometric
!pip install networkx
```

```
/opt/conda/lib/python3.10/pty.py:89: RuntimeWarning: os.fork() was called.
os.fork() is incompatible with multithreaded code, and JAX is multithreaded, so
this will likely lead to a deadlock.
  pid, fd = os.forkpty()

Collecting torch_geometric
  Downloading torch_geometric-2.5.2-py3-none-any.whl.metadata (64 kB)
                         64.2/64.2 kB
1.4 MB/s eta 0:00:00
Requirement already satisfied: tqdm in /opt/conda/lib/python3.10/site-
packages (from torch_geometric) (4.66.1)
Requirement already satisfied: numpy in /opt/conda/lib/python3.10/site-packages
(from torch_geometric) (1.26.4)
Requirement already satisfied: scipy in /opt/conda/lib/python3.10/site-packages
```

```
(from torch_geometric) (1.11.4)
Requirement already satisfied: fsspec in /opt/conda/lib/python3.10/site-packages
(from torch_geometric) (2024.3.0)
Requirement already satisfied: jinja2 in /opt/conda/lib/python3.10/site-packages
(from torch_geometric) (3.1.2)
Requirement already satisfied: aiohttp in /opt/conda/lib/python3.10/site-
packages (from torch_geometric) (3.9.1)
Requirement already satisfied: requests in /opt/conda/lib/python3.10/site-
packages (from torch_geometric) (2.31.0)
Requirement already satisfied: pyparsing in /opt/conda/lib/python3.10/site-
packages (from torch_geometric) (3.1.1)
Requirement already satisfied: scikit-learn in /opt/conda/lib/python3.10/site-
packages (from torch_geometric) (1.2.2)
Requirement already satisfied: psutil>=5.8.0 in /opt/conda/lib/python3.10/site-
packages (from torch_geometric) (5.9.3)
Requirement already satisfied: attrs>=17.3.0 in /opt/conda/lib/python3.10/site-
packages (from aiohttp->torch_geometric) (23.2.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
/opt/conda/lib/python3.10/site-packages (from aiohttp->torch_geometric) (6.0.4)
Requirement already satisfied: yarl<2.0,>=1.0 in /opt/conda/lib/python3.10/site-
packages (from aiohttp->torch_geometric) (1.9.3)
Requirement already satisfied: frozenlist>=1.1.1 in
/opt/conda/lib/python3.10/site-packages (from aiohttp->torch_geometric) (1.4.1)
Requirement already satisfied: aiosignal>=1.1.2 in
/opt/conda/lib/python3.10/site-packages (from aiohttp->torch_geometric) (1.3.1)
Requirement already satisfied: async-timeout<5.0,>=4.0 in
/opt/conda/lib/python3.10/site-packages (from aiohttp->torch_geometric) (4.0.3)
Requirement already satisfied: MarkupSafe>=2.0 in
/opt/conda/lib/python3.10/site-packages (from jinja2->torch_geometric) (2.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in
/opt/conda/lib/python3.10/site-packages (from requests->torch_geometric) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-
packages (from requests->torch_geometric) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.10/site-packages (from requests->torch_geometric)
(1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.10/site-packages (from requests->torch_geometric)
(2024.2.2)
Requirement already satisfied: joblib>=1.1.1 in /opt/conda/lib/python3.10/site-
packages (from scikit-learn->torch_geometric) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/opt/conda/lib/python3.10/site-packages (from scikit-learn->torch_geometric)
(3.2.0)
Downloading torch_geometric-2.5.2-py3-none-any.whl (1.1 MB)
                          1.1/1.1 MB
16.5 MB/s eta 0:00:00
Installing collected packages: torch_geometric
```

```
Successfully installed torch_geometric-2.5.2
Requirement already satisfied: networkx in /opt/conda/lib/python3.10/site-
packages (3.2.1)
```

[3]:
```
from torch_geometric.nn import GCNConv, global_mean_pool, BatchNorm
from torch_geometric.nn import MessagePassing
from torch_geometric.utils import add_self_loops, degree
from torch.nn import Sequential as Seq, Linear, ReLU
import torch.nn.functional as F
```

[4]:
```
from torch_geometric.data import Data
from sklearn.neighbors import kneighbors_graph
from torch_geometric.data import Dataset, Data, DataLoader
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import NearestNeighbors
import networkx as nx
from torch_geometric.utils import to_networkx



from torch_geometric.loader import DataLoader
from torch_geometric.nn import GCNConv, global_mean_pool
from torch.nn import Linear
import torch.nn.functional as F
from torch.utils.data import random_split
from torch.nn import Linear, Sequential, ReLU, Dropout
from torch_geometric.nn import GATConv
```

[5]:
```
chunk_size = 25

# List of Parquet file paths
file_paths = [
    '/kaggle/input/task2-24/QCDToGGQQ_IMGjet_RH1all_jet0_run0_n36272.test.
 ↪snappy.parquet',
    '/kaggle/input/task2-24/QCDToGGQQ_IMGjet_RH1all_jet0_run1_n47540.test.
 ↪snappy.parquet',
    '/kaggle/input/task2-24/QCDToGGQQ_IMGjet_RH1all_jet0_run2_n55494.test.
 ↪snappy.parquet'
]

# Initialize an empty list to store dataframes
dfs = []

# Loop through each file path
for file_path in file_paths:
    # Create a Parquet file reader object
    parquet_file = pq.ParquetFile(file_path)
```

```
    # Determine the total number of rows in the file
    total_rows = parquet_file.metadata.num_rows

    # Calculate the number of chunks
    num_chunks = total_rows // chunk_size + (1 if total_rows % chunk_size else␣
↪0)

    # Loop over the file in chunks
    for chunk_index in range(num_chunks):
        # Read a chunk of rows from the file
        chunk = parquet_file.read_row_group(chunk_index, columns=None)
        df = chunk.to_pandas()

        # Append the DataFrame to the list
        dfs.append(df)

# Concatenate all the DataFrames into a single DataFrame
data = pd.concat(dfs, ignore_index=True)
```

[6]:
```
def to_3d(arr):
    vishak=[]
    for i in range (0,3):
        vis=np.stack(np.stack(arr)[i],axis=-1)
        vishak.append(vis)
    vishak=np.array(vishak)
    return vishak
```

[7]:
```
data["X_jets"]  = data["X_jets"].apply(to_3d)
```

[8]:
```
def image_to_graph(image, patch_size=25, n_neighbors=5):
        """
        Convert an image to a graph of its 5x5 patches.

        Parameters:
        - image: A (125, 125, 3) numpy array.
        - patch_size: Size of the square patches (default 5).
        - n_neighbors: Number of neighbors for KNN (default 5).

        Returns:
        - nodes: An array of node features.
        - edges: A list of tuples (i, j, mse) representing edges and their MSE.
        """
        # Validate image shape

        assert image.shape[0] == image.shape[1], "Image must be square."
```

```
        # Number of patches along one dimension
        num_patches = image.shape[0] // patch_size

        # Initialize nodes and edges
        nodes = []
        edges = []

        # Create patches and flatten them to create node features
        for i in range(0, image.shape[0], patch_size):
            for j in range(0, image.shape[1], patch_size):
                patch = image[i:i+patch_size, j:j+patch_size, :].reshape(-1)
                nodes.append(patch)

        nodes = np.array(nodes)

        # Use KNN to find nearest neighbors for each node
        nbrs = NearestNeighbors(n_neighbors=n_neighbors+1,␣
↪algorithm='ball_tree').fit(nodes)
        distances, indices = nbrs.kneighbors(nodes)

        # Calculate MSE for each pair of neighbors and create edges
        for i in range(indices.shape[0]):
            for j in range(1, indices.shape[1]):  # Start from 1 to skip␣
↪self-connection
                mse = mean_squared_error(nodes[i], nodes[indices[i, j]])
                edges.append((i, indices[i, j], mse))

        return nodes, edges
```

[9]:
```
class QuarkGluonDataset(Dataset):

    def __init__(self, dataframe, root='', transform=None, pre_transform=None):
        """
        Custom dataset for quarks and gluons classification.

        Parameters:
        - image_list: A list of (125, 125, 3) numpy arrays.
        - labels: A list of integers (0 or 1) representing the class labels for␣
↪the images.
        """
        self.dataframe = dataframe
        super(QuarkGluonDataset, self).__init__(root, transform, pre_transform)


    def len(self):
        return len(self.dataframe)
```

```
    def get(self, idx):
        # Convert an image to graph data
        image = self.dataframe.iloc[idx]['X_jets']
        image = image.transpose(1,2,0)
        label = self.dataframe.iloc[idx]['y']
#         print(type(image))
        nodes, edges = image_to_graph(image)

        # Convert to PyTorch tensors
        x = torch.tensor(nodes, dtype=torch.float)  # Node features
        edge_index = torch.tensor([(i, j) for i, j, _ in edges], dtype=torch.
↪long).t().contiguous()  # Edge indices
        edge_attr = torch.tensor([mse for _, _, mse in edges], dtype=torch.
↪float).unsqueeze(1)  # Edge attributes
        y = torch.tensor([label], dtype=torch.long)  # Label

        return Data(x=x, edge_index=edge_index, edge_attr=edge_attr, y=y)
```

[10]:
```
dataset = QuarkGluonDataset(data)
```

[11]:
```
next(iter(dataset))
```

[11]: Data(x=[25, 1875], edge_index=[2, 125], edge_attr=[125, 1], y=[1])

[12]:
```
len(dataset)
```

[12]: 5573

[13]:
```
dataset_size = len(dataset)
train_size = int(0.8 * dataset_size)
test_size = dataset_size - train_size

# Perform the random split
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# Create the DataLoaders for the train and test sets
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
valid_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

[14]:
```
# next(iter(train_loader))
```

[15]:
```
class CorrectedGNN(torch.nn.Module):
    def __init__(self, node_in_channels, edge_in_channels, hidden_channels):
        super(CorrectedGNN, self).__init__()
        self.conv1 = GATConv(node_in_channels, hidden_channels,␣
↪add_self_loops=True)
```

```
        self.conv2 = GATConv(hidden_channels, hidden_channels,␣
↪add_self_loops=True)
        self.lin = Linear(hidden_channels, 2)
        self.dropout_rate = 0.5

    def forward(self, data):
        x, edge_index, edge_attr = data.x, data.edge_index, data.edge_attr
        batch = data.batch

        # First GAT layer
        x = F.relu(self.conv1(x, edge_index))
        x = F.dropout(x, p=self.dropout_rate, training=self.training)

        # Second GAT layer
        x = F.relu(self.conv2(x, edge_index))
        x = F.dropout(x, p=self.dropout_rate, training=self.training)

        # Global mean pooling
        x = global_mean_pool(x, batch)

        # Apply final classifier
        x = self.lin(x)

        return F.log_softmax(x, dim=1)

# Example model instantiation
node_in_channels = 1875   # Number of node features
edge_in_channels = 1      # Number of edge features
hidden_channels = 64      # Hidden layer size
model = CorrectedGNN(node_in_channels, edge_in_channels, hidden_channels)
print(model)
```

```
CorrectedGNN(
  (conv1): GATConv(1875, 64, heads=1)
  (conv2): GATConv(64, 64, heads=1)
  (lin): Linear(in_features=64, out_features=2, bias=True)
)
```

[16]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f'Using device: {device}')

# Move the model to the chosen device
model.to(device)

# Modify the train function to include loss in tqdm
def train(model, train_loader, optimizer, criterion):
    model.train()
```

```python
    total_loss = 0
    with tqdm(train_loader, desc="Training") as tepoch:
        for data in tepoch:
            data = data.to(device)
            optimizer.zero_grad()
            out = model(data)
            loss = criterion(out, data.y)
            loss.backward()
            optimizer.step()
            total_loss += loss.item() * data.num_graphs
            tepoch.set_postfix(loss=loss.item())
    return total_loss / len(train_loader.dataset)

# Similar modifications for the test/validation function, including data␣
 ↪transfer to the device
def test(model, loader, criterion):
    model.eval()
    correct = 0
    total_loss = 0
    with torch.no_grad():
        for data in tqdm(loader, desc="Evaluating", leave=False):
            data = data.to(device)
            out = model(data)
            loss = criterion(out, data.y)
            total_loss += loss.item() * data.num_graphs
            pred = out.argmax(dim=1)
            correct += int((pred == data.y).sum())
    return correct / len(loader.dataset), total_loss / len(loader.dataset)

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = torch.nn.CrossEntropyLoss()

best_val_acc = 0.0
for epoch in range(1, 20):
    train_loss = train(model, train_loader, optimizer, criterion)
#     train_acc, _ = test(model, train_loader, criterion)
    val_acc, val_loss = test(model, valid_loader, criterion)

    if val_acc > best_val_acc:
        best_val_acc = val_acc
        torch.save(model.state_dict(), 'best_model.pth')
        print(f"Saved Best Model: Epoch {epoch}, Val. Acc.: {val_acc:.4f}")

    print(f'Epoch: {epoch:03d}, Train Loss: {train_loss:.4f}, Val. Loss:␣
 ↪{val_loss:.4f}, Val. Acc.: {val_acc:.4f}')
```

```
Using device: cpu
```

```
Training: 100%|     | 140/140 [02:12<00:00,  1.06it/s, loss=0.716]

Saved Best Model: Epoch 1, Val. Acc.: 0.6502
Epoch: 001, Train Loss: 0.6747, Val. Loss: 0.6509, Val. Acc.: 0.6502

Training: 100%|     | 140/140 [02:11<00:00,  1.07it/s, loss=0.764]

Saved Best Model: Epoch 2, Val. Acc.: 0.6529
Epoch: 002, Train Loss: 0.6179, Val. Loss: 0.6409, Val. Acc.: 0.6529

Training: 100%|     | 140/140 [02:09<00:00,  1.08it/s, loss=0.626]

Epoch: 003, Train Loss: 0.5491, Val. Loss: 0.6666, Val. Acc.: 0.6439

Training: 100%|     | 140/140 [02:11<00:00,  1.06it/s, loss=0.193]

Epoch: 004, Train Loss: 0.4685, Val. Loss: 0.7088, Val. Acc.: 0.6448

Training: 100%|     | 140/140 [02:12<00:00,  1.06it/s, loss=0.237]

Epoch: 005, Train Loss: 0.3851, Val. Loss: 0.7839, Val. Acc.: 0.6251

Training: 100%|     | 140/140 [02:11<00:00,  1.06it/s, loss=0.279]

Epoch: 006, Train Loss: 0.3030, Val. Loss: 0.8936, Val. Acc.: 0.6341

Training: 100%|     | 140/140 [02:10<00:00,  1.07it/s, loss=0.184]

Epoch: 007, Train Loss: 0.2759, Val. Loss: 0.9411, Val. Acc.: 0.6439

Training: 100%|     | 140/140 [02:10<00:00,  1.07it/s, loss=0.0683]

Epoch: 008, Train Loss: 0.1916, Val. Loss: 1.0604, Val. Acc.: 0.6404

Training: 100%|     | 140/140 [02:10<00:00,  1.07it/s, loss=0.201]

Epoch: 009, Train Loss: 0.1483, Val. Loss: 1.1669, Val. Acc.: 0.6413

Training: 100%|     | 140/140 [02:10<00:00,  1.07it/s, loss=0.0901]

Epoch: 010, Train Loss: 0.1109, Val. Loss: 1.3023, Val. Acc.: 0.6439

Training: 100%|     | 140/140 [02:10<00:00,  1.07it/s, loss=0.0469]

Epoch: 011, Train Loss: 0.1016, Val. Loss: 1.3563, Val. Acc.: 0.6439

Training: 100%|     | 140/140 [02:16<00:00,  1.03it/s, loss=0.0305]

Epoch: 012, Train Loss: 0.1070, Val. Loss: 1.4630, Val. Acc.: 0.6395

Training: 100%|     | 140/140 [02:13<00:00,  1.05it/s, loss=0.0585]

Epoch: 013, Train Loss: 0.0689, Val. Loss: 1.5963, Val. Acc.: 0.6386

Training: 100%|     | 140/140 [02:13<00:00,  1.05it/s, loss=0.0904]

Epoch: 014, Train Loss: 0.0580, Val. Loss: 1.5971, Val. Acc.: 0.6448

Training: 100%|     | 140/140 [02:11<00:00,  1.07it/s, loss=0.0145]

Epoch: 015, Train Loss: 0.0906, Val. Loss: 1.6821, Val. Acc.: 0.6448

Training: 100%|     | 140/140 [02:12<00:00,  1.06it/s, loss=0.0114]
```

```
Epoch: 016, Train Loss: 0.0438, Val. Loss: 1.8080, Val. Acc.: 0.6422
Training: 100%|      | 140/140 [02:12<00:00,  1.05it/s, loss=0.165]
Epoch: 017, Train Loss: 0.0428, Val. Loss: 1.8286, Val. Acc.: 0.6323
Training: 100%|      | 140/140 [02:10<00:00,  1.07it/s, loss=0.0176]
Epoch: 018, Train Loss: 0.0427, Val. Loss: 1.8692, Val. Acc.: 0.6422
Training: 100%|      | 140/140 [02:13<00:00,  1.05it/s, loss=0.0259]
Epoch: 019, Train Loss: 0.0302, Val. Loss: 1.9645, Val. Acc.: 0.6413
```

[ ]:

[ ]:

[ ]: