# project-code

December 18, 2023

[6]:
```
import torch
vers = torch.__version__
print("Torch vers: ", vers)

# PyG installation
!pip install -q torch-scatter -f https://pytorch-geometric.com/whl/
↪torch-${TORCH}+${CUDA}.html
!pip install -q torch-sparse -f https://pytorch-geometric.com/whl/
↪torch-${TORCH}+${CUDA}.html
!pip install -q git+https://github.com/rusty1s/pytorch_geometric.git

import torch_geometric
```

```
Torch vers:  2.1.0
```

[7]:
```
from torch_geometric.datasets import UPFD
train_data = UPFD(root=".", name="gossipcop", feature="content", split="train")
test_data = UPFD(root=".", name="gossipcop", feature="content", split="test")
print("Train Samples: ", len(train_data))
print("Test Samples: ", len(test_data))
```

```
Train Samples:  1092
Test Samples:  3826
```

[8]:
```
sample_id=1
train_data[sample_id].edge_index
```

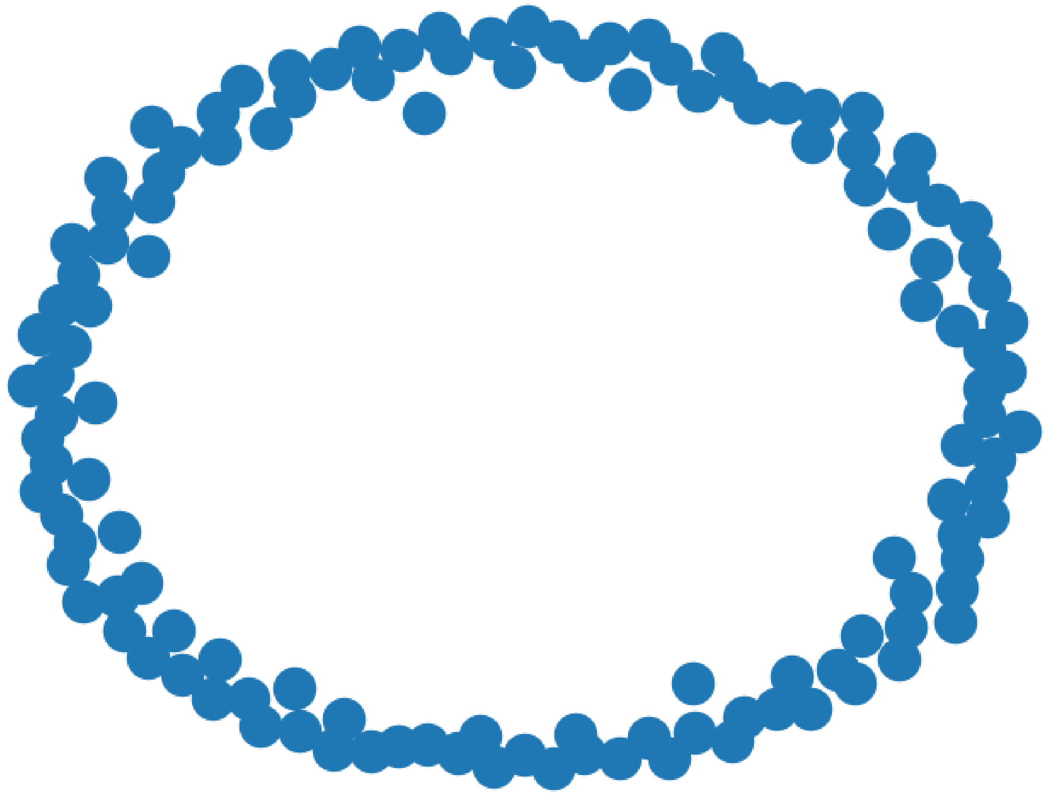
[8]:
```
tensor([], size=(2, 0), dtype=torch.int64)
```

[9]:
```
"""
Had to import this "manually" due to some errors.
"""

!pip install networkx
import networkx as nx

# From PyG utils
def to_networkx(data, node_attrs=None, edge_attrs=None, to_undirected=False,
```

```
                 remove_self_loops=False):
    if to_undirected:
        G = nx.Graph()
    else:
        G = nx.DiGraph()
    G.add_nodes_from(range(data.num_nodes))
    node_attrs, edge_attrs = node_attrs or [], edge_attrs or []
    values = {}
    for key, item in data(*(node_attrs + edge_attrs)):
        if torch.is_tensor(item):
            values[key] = item.squeeze().tolist()
        else:
            values[key] = item
        if isinstance(values[key], (list, tuple)) and len(values[key]) == 1:
            values[key] = item[0]
    for i, (u, v) in enumerate(data.edge_index.t().tolist()):
        if to_undirected and v > u:
            continue
        if remove_self_loops and u == v:
            continue
        G.add_edge(u, v)
        for key in edge_attrs:
            G[u][v][key] = values[key][i]
    for key in node_attrs:
        for i, feat_dict in G.nodes(data=True):
            feat_dict.update({key: values[key][i]})
    return G
```

```
Requirement already satisfied: networkx in ./anaconda3/lib/python3.11/site-
packages (3.1)
```

[10]:
```
nx.draw(to_networkx(train_data[sample_id]))
```



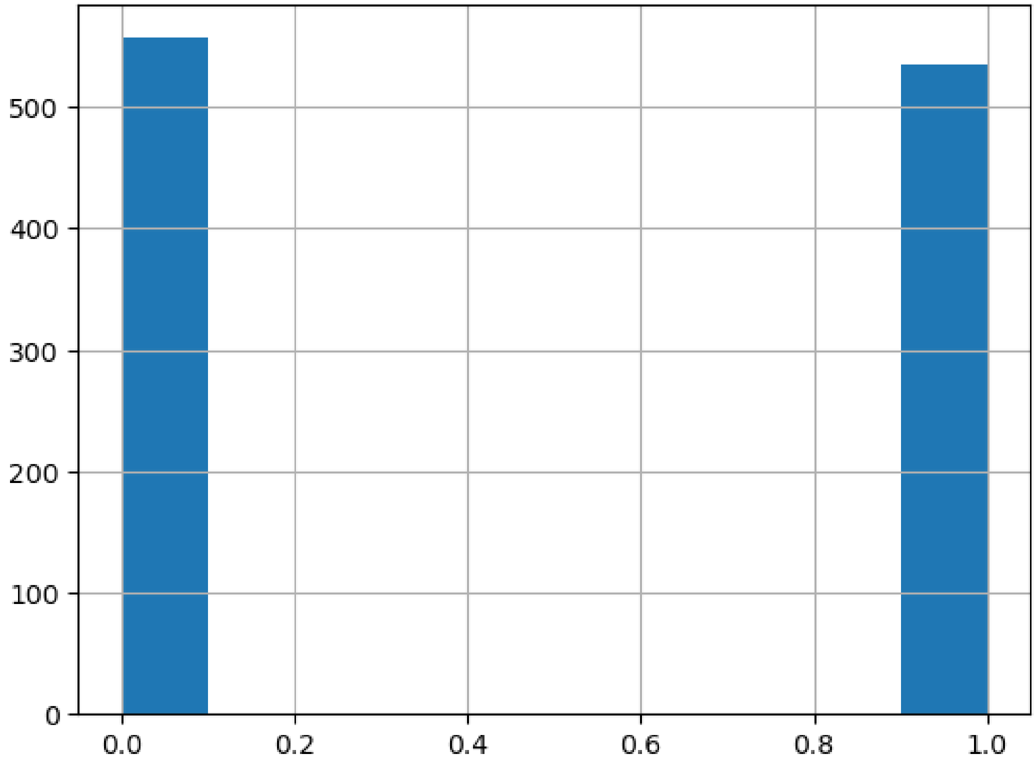
[11]:
```python
print(train_data[sample_id].x.shape)
train_data[sample_id].x
```

```
torch.Size([125, 310])
```

[11]:
```
tensor([[0.5220, 0.5120, 0.4817,  …, 0.6874, 0.1023, 0.1529],
        [0.4610, 0.4544, 0.6279,  …, 0.2055, 0.1667, 0.2500],
        [0.4461, 0.4728, 0.4978,  …, 0.8356, 0.0556, 0.0000],
        …,
        [0.5574, 0.4934, 0.5588,  …, 0.3562, 0.0556, 0.2647],
        [0.5274, 0.5889, 0.4425,  …, 0.9589, 0.1111, 0.1176],
        [0.5220, 0.5120, 0.4817,  …, 0.6874, 0.1023, 0.1529]])
```

[12]:
```python
import pandas as pd
labels = [data.y.item() for i, data in enumerate(train_data)]
df = pd.DataFrame(labels, columns=["Labels"])
df["Labels"].hist()
```

[12]:
```
<Axes: >
```




[13]:
```
from torch_geometric.loader import DataLoader
train_loader = DataLoader(train_data, batch_size=128, shuffle=True)
test_loader = DataLoader(test_data, batch_size=128, shuffle=False)
```

[14]:
```
from torch_geometric.nn import global_max_pool as gmp
from torch_geometric.nn import GATConv
from torch.nn import Linear


class GNN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super().__init__()

        # Graph Convolutions
        self.conv1 = GATConv(in_channels, hidden_channels)
        self.conv2 = GATConv(hidden_channels, hidden_channels)
        self.conv3 = GATConv(hidden_channels, hidden_channels)

        # Readout
        self.lin_news = Linear(in_channels, hidden_channels)
        self.lin0 = Linear(hidden_channels, hidden_channels)
        self.lin1 = Linear(2*hidden_channels, out_channels)
```

```python
    def forward(self, x, edge_index, batch):
        # Graph Convolutions
        h = self.conv1(x, edge_index).relu()
        h = self.conv2(h, edge_index).relu()
        h = self.conv3(h, edge_index).relu()

        # Pooling
        h = gmp(h, batch)

        # Readout
        h = self.lin0(h).relu()

        # According to UPFD paper: Include raw word2vec embeddings of news
        # This is done per graph in the batch
        root = (batch[1:] - batch[:-1]).nonzero(as_tuple=False).view(-1)
        root = torch.cat([root.new_zeros(1), root + 1], dim=0)
        # root is e.g. [   0,    14,    94,   171,   230,   302, ... ]
        news = x[root]
        news = self.lin_news(news).relu()

        out = self.lin1(torch.cat([h, news], dim=-1))
        return torch.sigmoid(out)

GNN(train_data.num_features, 128, 1)
```

[14]:
```
GNN(
  (conv1): GATConv(310, 128, heads=1)
  (conv2): GATConv(128, 128, heads=1)
  (conv3): GATConv(128, 128, heads=1)
  (lin_news): Linear(in_features=310, out_features=128, bias=True)
  (lin0): Linear(in_features=128, out_features=128, bias=True)
  (lin1): Linear(in_features=256, out_features=1, bias=True)
)
```

[15]:
```python
from sklearn.metrics import accuracy_score, f1_score

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = GNN(train_data.num_features, 128, 1).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=0.01)
loss_fnc = torch.nn.BCELoss()

def train(epoch):
    model.train()
    total_loss = 0
    for data in train_loader:
        data = data.to(device)
```

```
        optimizer.zero_grad()
        out = model(data.x, data.edge_index, data.batch)
        loss = loss_fnc(torch.reshape(out, (-1,)), data.y.float())
        loss.backward()
        optimizer.step()
        total_loss += float(loss) * data.num_graphs
    return total_loss / len(train_loader.dataset)

@torch.no_grad()
def test(epoch):
    model.eval()
    total_loss = 0
    all_preds = []
    all_labels = []
    for data in test_loader:
        data = data.to(device)
        out = model(data.x, data.edge_index, data.batch)
        loss = loss_fnc(torch.reshape(out, (-1,)), data.y.float())
        total_loss += float(loss) * data.num_graphs
        all_preds.append(torch.reshape(out, (-1,)))
        all_labels.append(data.y.float())

    # Calculate Metrics
    accuracy, f1 = metrics(all_preds, all_labels)

    return total_loss / len(test_loader.dataset), accuracy, f1


def metrics(preds, gts):
    preds = torch.round(torch.cat(preds))
    gts = torch.cat(gts)
    acc = accuracy_score(preds, gts)
    f1 = f1_score(preds, gts)
    return acc, f1
```

[16]:
```
for epoch in range(20):
    train_loss = train(epoch)
    test_loss, test_acc, test_f1 = test(epoch)
    print(f'Epoch: {epoch:02d} |  TrainLoss: {train_loss:.2f} | '
          f'TestLoss: {test_loss:.2f} | TestAcc: {test_acc:.2f} | TestF1:␣
 ↪{test_f1:.2f}')
```

```
Epoch: 00 |  TrainLoss: 0.81 | TestLoss: 0.69 | TestAcc: 0.50 | TestF1: 0.00
Epoch: 01 |  TrainLoss: 0.69 | TestLoss: 0.69 | TestAcc: 0.50 | TestF1: 0.67
Epoch: 02 |  TrainLoss: 0.69 | TestLoss: 0.70 | TestAcc: 0.50 | TestF1: 0.00
Epoch: 03 |  TrainLoss: 0.69 | TestLoss: 0.68 | TestAcc: 0.50 | TestF1: 0.67
Epoch: 04 |  TrainLoss: 0.68 | TestLoss: 0.68 | TestAcc: 0.50 | TestF1: 0.67
```

```
Epoch: 05 |  TrainLoss: 0.68 | TestLoss: 0.69 | TestAcc: 0.50 | TestF1: 0.67
Epoch: 06 |  TrainLoss: 0.69 | TestLoss: 0.68 | TestAcc: 0.50 | TestF1: 0.00
Epoch: 07 |  TrainLoss: 0.67 | TestLoss: 0.66 | TestAcc: 0.72 | TestF1: 0.62
Epoch: 08 |  TrainLoss: 0.65 | TestLoss: 0.65 | TestAcc: 0.53 | TestF1: 0.11
Epoch: 09 |  TrainLoss: 0.65 | TestLoss: 0.91 | TestAcc: 0.50 | TestF1: 0.00
Epoch: 10 |  TrainLoss: 0.75 | TestLoss: 0.65 | TestAcc: 0.89 | TestF1: 0.90
Epoch: 11 |  TrainLoss: 0.65 | TestLoss: 0.65 | TestAcc: 0.54 | TestF1: 0.14
Epoch: 12 |  TrainLoss: 0.65 | TestLoss: 0.65 | TestAcc: 0.50 | TestF1: 0.67
Epoch: 13 |  TrainLoss: 0.64 | TestLoss: 0.62 | TestAcc: 0.89 | TestF1: 0.90
Epoch: 14 |  TrainLoss: 0.62 | TestLoss: 0.60 | TestAcc: 0.91 | TestF1: 0.90
Epoch: 15 |  TrainLoss: 0.60 | TestLoss: 0.57 | TestAcc: 0.89 | TestF1: 0.88
Epoch: 16 |  TrainLoss: 0.61 | TestLoss: 0.57 | TestAcc: 0.68 | TestF1: 0.53
Epoch: 17 |  TrainLoss: 0.59 | TestLoss: 0.57 | TestAcc: 0.64 | TestF1: 0.44
Epoch: 18 |  TrainLoss: 0.58 | TestLoss: 0.57 | TestAcc: 0.63 | TestF1: 0.42
Epoch: 19 |  TrainLoss: 0.52 | TestLoss: 0.50 | TestAcc: 0.90 | TestF1: 0.90
```

[17]:

```
for data in test_loader:
    data = data.to(device)
    pred = model(data.x, data.edge_index, data.batch)
    df = pd.DataFrame()
    df["pred_logit"] = pred.detach().numpy()[:,0]
    df["pred"] = torch.round(pred).detach().numpy()[:,0]
    df["true"] = data.y.numpy()
    print(df.head(10))
    break
```

```
   pred_logit  pred  true
0    0.598197   1.0     1
1    0.496605   0.0     1
2    0.409992   0.0     0
3    0.519230   1.0     1
4    0.289776   0.0     0
5    0.417364   0.0     0
6    0.390881   0.0     0
7    0.665851   1.0     1
8    0.290718   0.0     0
9    0.383876   0.0     0
```

My code starts from here. The code above this must not be modified.

[64]:

```
def fgsm_attack(model, loss_fn, data, epsilon):
    # Check if edge_index is valid
    if data.edge_index.numel() == 0 or data.edge_index.max() >= data.x.size(0):
        return data

    if data.edge_index.numel() == 0:
        # Skip processing for graphs without edges
        return data
```

```
    data = data.clone()
    data.x.requires_grad = True

    model.eval()
    output = model(data.x, data.edge_index, data.batch)
    target = data.y.unsqueeze(1).float()

    loss = loss_fn(output, target)
    model.zero_grad()
    loss.backward()

    # Apply FGSM attack and then detach the result
    data.x = (data.x + epsilon * data.x.grad.sign()).detach()
    return data
```

[65]:
```
def train(model, train_loader, optimizer, loss_fn, device, epsilon):
    model.train()
    total_loss = 0

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

        if data.edge_index.numel() == 0 or data.edge_index.max() >= data.x.
 ↪size(0):
            continue

        data_adv = fgsm_attack(model, loss_fn, data, epsilon)
        data_adv = data_adv.to(device)

        optimizer.zero_grad()
        output = model(data.x, data.edge_index, data.batch)
        output_adv = model(data_adv.x, data_adv.edge_index, data_adv.batch)

        loss = loss_fn(output, data.y.float().unsqueeze(1))
        loss_adv = loss_fn(output_adv, data_adv.y.float().unsqueeze(1))
        combined_loss = loss + loss_adv

        combined_loss.backward()
        optimizer.step()
        total_loss += combined_loss.item()

    return total_loss / len(train_loader)
```

[66]:
```
@torch.no_grad()  # Disable gradient computation during validation
def validate(model, val_loader, device):
    model.eval()  # Set the model to evaluation mode
```

```
    correct = 0
    total = 0

    for data in val_loader:
        data = data.to(device)
        outputs = model(data.x, data.edge_index, data.batch)

        # Assuming the output is a probability and using 0.5 as the threshold
        predicted = (outputs > 0.5).float().view(-1)
        total += data.y.size(0)
        correct += (predicted == data.y.float().to(device)).sum().item()

    accuracy = correct / total
    return accuracy
```

[67]:

```
from torch_geometric.loader import DataLoader

# Assuming test_data is your validation dataset
val_loader = DataLoader(test_data, batch_size=128, shuffle=False)

# Now val_loader can be used in the training and validation loop
```

[68]:

```
epsilon_values = [0, 0.01, 0.02, 0.05, 0.1]
num_epochs = 5
results = {}

for epsilon in epsilon_values:
    # Initialize or reset your model and optimizer here
    model = GNN(train_data.num_features, 128, 1).to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=0.01)

    for epoch in range(num_epochs):
        train_loss = train(model, train_loader, optimizer, loss_fnc, device,␣
 ↪epsilon)

    results[epsilon] = (train_loss, val_accuracy)

# Print or analyze the results
for epsilon, (train_loss, val_accuracy) in results.items():
    print(f"Epsilon: {epsilon}, Train Loss: {train_loss}, Validation Accuracy:␣
 ↪{val_accuracy}")
```

```
Epsilon: 0, Train Loss: 0.9276018937428793, Validation Accuracy:
0.8515420805018296
Epsilon: 0.01, Train Loss: 0.9210668139987521, Validation Accuracy:
0.8515420805018296
Epsilon: 0.02, Train Loss: 0.7729580534829034, Validation Accuracy:
```

```
0.8515420805018296
Epsilon: 0.05, Train Loss: 1.084211852815416, Validation Accuracy:
0.8515420805018296
Epsilon: 0.1, Train Loss: 0.622085796462165, Validation Accuracy:
0.8515420805018296
```