```
!pip install pennylane
```

```
import pennylane
```

```
!pip install numpy
!pip install energyflow
!pip install particle
!pip install torch torchvision
!pip install torch-geometric
!pip install tqdm
!pip install networkx
!pip install matplotlib
!pip install scikit-learn
```

```
import numpy as np
import energyflow
from particle import Particle

import torch
import torch.nn as nn
import torch.nn.functional as F
import pennylane as qml
from torch_geometric.nn import GCNConv, GATConv
from torch.nn import Linear, ReLU, Sigmoid, Tanh, ModuleList, LeakyReLU, Sequential, Dropout
from torch_geometric.nn import global_mean_pool
from torch_geometric.nn import MessagePassing
from torch_geometric.utils import remove_self_loops, add_self_loops, degree
from torch_geometric.data import Data
from torch_geometric.utils import to_networkx
from torch.nn import BCEWithLogitsLoss, CrossEntropyLoss
from torch.optim import Adam
from torch_geometric.loader import DataLoader
from tqdm import tqdm

import networkx as nx
import os
import matplotlib.pyplot as plt
import copy
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

Hyperparameters

```
n_connections = 3
input_dims = 8
hidden_dims = [8,8]
output_dims = 1
q_depth = 3
epochs = 20
lr = 1e-3
train_ratio = 0.6
val_ratio = 0.2
test_ratio = 0.2
batch_size = 64
```

```
path = "/content/QG_jets.npz"
```

Load the dataset

```
energyflow.qg_jets.load(num_data=10000, pad=True, ncol=4, generator='pythia', with_bc=False, cache_dir='~/data')
```

```
dataset = np.load(os.path.expanduser('~/data/datasets/QG_jets.npz'))
```

```
dataset.files
```

```
['X', 'y']
```

```
X = dataset['X']
y = dataset['y']
X.shape, y.shape
```

```
((100000, 139, 4), (100000,))
```

```
X.shape
```

```
(100000, 139, 4)
```

```
y.shape, y.dtype
```

```
((100000,), dtype('float64'))
```

Select jets with >= 10 particles

```
inds10 = np.logical_and(np.sum(X[:,:,0]>1e-10,axis=1)>=10,np.sum(X[:,:,0]>1e-10,axis=1)<=120)
```

```
X10 = X[inds10,:,:]
y10 = y[inds10]
```

```
X10.shape, y10.shape
```

```
((99870, 139, 4), (99870,))
```

Sort by transverse momentum

```
X_sorted = np.zeros((X10.shape[0], 10, X10.shape[2]))
for i in range(X10.shape[0]):
    sort_nodes = np.argsort(X10[i,:,0])[::-1]
    X_sorted[i,:,:] = X10[i,sort_nodes[:10],:]
```

```
X_sorted.shape
```

```
(99870, 10, 4)
```

+ Code + Text

```
X_sorted = X_sorted[:,:n_connections,:]
X_sorted.shape
```

```
(99870, 3, 4)
```

```
pid = Particle.from_pdgid(-2.11000000e+02)
pid.mass, pid.charge
```

```
(139.57039, -1.0)
```

```
def feature_engineer_dataset(X):

    # unpack all features
    pt = X[:,:,0]
    rapidity = X[:,:,1]
    phi = X[:,:,2]
    pdgids = X[:,:,3]

    # Calculate new feature values
    nonzero_pdgids = np.where(pdgids > 0)
    mass = np.zeros(pdgids.shape)
    mass_nonzeros = np.vectorize(Particle.from_pdgid)(pdgids[nonzero_pdgids])
    mass[nonzero_pdgids] = np.fromiter((i.mass for i in mass_nonzeros), float)

    mt = np.sqrt(np.square(mass) + np.square(pt))
    energy = mass*np.cosh(rapidity)
    px = pt*np.cos(phi)
    py = pt*np.sin(phi)
    pz = mt*np.sinh(rapidity)

    return torch.tensor(np.stack((pt, rapidity, phi, mt, energy, px, py, pz), axis=2), dtype=torch.float)
```

```
X_new = feature_engineer_dataset(X_sorted)
```

```
X_new.shape
```

```
torch.Size([99870, 3, 8])
```

```
p = Particle.from_pdgid(2212)
print(p)
```

```
p
```

```
num_particles = torch.zeros(X_new.shape[0])
num_photons = torch.zeros(X_new.shape[0])
num_charged_hadrons = torch.zeros(X_new.shape[0])
num_neutral_hadrons = torch.zeros(X_new.shape[0])
num_particles.shape
```

```
torch.Size([99870])
```

```
inds = np.where(X10[2,:,0] > 1e-10)[0]
inds.shape[0]
```

```
57
```

```
for i in range(len(X10)):
    inds = np.where(X10[i,:,0] > 1e-10)[0]
    num_particles[i] = inds.shape[0]

    num_photons[i] = len(np.where(X10[i,inds,3] == 22)[0])

    num_charged_hadrons[i] = len(np.where(np.logical_or((abs(X10[i,inds,3]) == 211),
                                 (abs(X10[i,inds,3]) == 321),
                                 (abs(X10[i,inds,3]) == 2212)
                                ) == True)[0])


    num_neutral_hadrons[i] = len(np.where(np.logical_or((abs(X[i,inds,3]) == 130),
                                 (abs(X[i,inds,3]) == 2112)
                                ) == True)[0])                   # neutral hadron
```

```
pdgids = X10[0,:,3]
nonzero_pdgids = np.where(pdgids > 0)
nonzero_pdgids
```

```
(array([ 0,  1,  5,  6,  7,  8, 10, 11, 12, 13, 14, 15, 16, 17]),)
```

```
pt = X10[0,nonzero_pdgids,0]
pt.shape, X10.shape
```

```
((1, 14), (99870, 139, 4))
```

```
pdgids = X10[:,:,3]
nonzero_pdgids = np.where(pdgids > 0)
mass_nonzeros = np.vectorize(Particle.from_pdgid)(pdgids[nonzero_pdgids])
charge_particles = np.zeros(pdgids.shape)
charge_particles[nonzero_pdgids] = np.fromiter((i.charge if i.charge > 0 else 0 for i in mass_nonzeros), float)
pt = X10[:,:,0]
charge_jet_2 = np.sum(charge_particles * ((pt**0.2)/np.sum(pt)**0.2) , axis=1)
charge_jet_5 = np.sum(charge_particles * ((pt**0.5)/np.sum(pt)**0.5) , axis=1)
charge_jet_7 = np.sum(charge_particles * ((pt**0.7)/np.sum(pt)**0.7) , axis=1)
```

```
charge_jet = np.sum(charge_particles, axis=1)
charge_jet.shape
```

```
(99870,)
```

```
print(charge_jet_2.shape)
charge_jet_2
```

```
(99870,)
```

```
array([0.07898153, 0.1725243 , 0.52408037, ..., 0.17009737, 0.48479047,
       0.4587912 ])
```

```
charge_jet_7
```

```
array([2.48108168e-05, 8.78291430e-05, 1.99322411e-04, ...,
       4.69282329e-05, 1.52115589e-04, 2.57501495e-04])
```

```
graph_features = torch.stack((num_particles, num_photons, num_charged_hadrons, num_neutral_hadrons, torch.tensor(charge_jet)), dim=1)
graph_features.shape
```

```
torch.Size([99870, 5])
```

```
graph_features
```

```
tensor([[18., 12.,  6.,  0.,  2.],
        [17.,  7.,  8.,  1.,  4.],
        [57., 24., 24.,  6., 13.],
        ...,
        [16.,  6.,  9.,  0.,  5.],
        [88., 55., 26.,  3., 13.],
        [40., 15., 19.,  5., 12.]], dtype=torch.float64)
```

```
graph_features_norm = graph_features / torch.amax(graph_features, dim=(0))
graph_features_norm
```

```
tensor([[0.1500, 0.1644, 0.0923, 0.0000, 0.0588],
        [0.1417, 0.0959, 0.1231, 0.0909, 0.1176],
        [0.4750, 0.3288, 0.3692, 0.5455, 0.3824],
        ...,
        [0.1333, 0.0822, 0.1385, 0.0000, 0.1471],
        [0.7333, 0.7534, 0.4000, 0.2727, 0.3824],
        [0.3333, 0.2055, 0.2923, 0.4545, 0.3529]], dtype=torch.float64)
```

```
graph_features_norm.shape
```

```
torch.Size([99870, 5])
```

Normalize the dataset

```
torch.amax(X_new,dim=(0,1))
```

```
tensor([5.0142e+02, 1.9360e+00, 6.5275e+00, 1.0556e+03, 3.2500e+03, 4.9331e+02,
        4.7369e+02, 3.0296e+03])
```

```
X_norm = X_new / torch.amax(X_new, dim=(0,1))
# fetch the maximum value for all feature (in last dimension) and normalize the dataset through max scaling.
X_norm.shape
```

```
torch.Size([99870, 3, 8])
```

```
def strip_zeros(X):
    X_useful = []
    for Xi in X:
        inds = np.where(Xi[:,0] > 1e-10)
        X_useful.append(Xi[inds])

    return X_useful
```

```
X_preprocessed = strip_zeros(X_norm)
len(X_preprocessed), X_preprocessed[0].shape, X_preprocessed[1].shape
```

```
(99870, torch.Size([3, 8]), torch.Size([3, 8]))
```

Form graph dataset

```
def create_graph(x):
    M = x.shape[0]  # number of nodes in graph (particles)
    N = M-1         # number of neighbors of each node to be considered for edge connections (1 extra for self loops)
    # (√N is considered optimal number of neighbors for k-NN on N points)
```

```
    edge_index = torch.zeros((2,M*N))
    edge_attr = torch.zeros((M*N, 1))
    # adj_node_pairs = torch.zeros((, ))

    for i, features in enumerate(x):
        # find N nearest neighbors in (ϕ,y) space
        distances = torch.sqrt((features[2] - x[:,2])**2 + (features[1] - x[:,1])**2)
        N_nearest = np.argsort(distances)[1:N+1]

        edge_index[0][i*N:(i+1)*N] = torch.tensor([i for _ in range(N)])
        edge_index[1][i*N:(i+1)*N] = N_nearest
        edge_attr[i*N:(i+1)*N] = distances[N_nearest].reshape(N,1)   # (max(distances) - distances[N_nearest])/(min(distances[distances > 0]

    edge_index = edge_index.to(torch.int)
    return edge_index, edge_attr


def create_graph_dataset(X, y, graph_features):
    dataset = []

    for Xi, yi, gi in zip(X,y,graph_features):

        edge_index, edge_attr = create_graph(Xi)
        data = Data(x=Xi, edge_index=edge_index, edge_attr=edge_attr,y=torch.tensor(yi, dtype=torch.long), graph_features=gi)
        dataset.append(data)


    return dataset


dataset = create_graph_dataset(X_norm[:12500, :3, :],y10[:12500], graph_features_norm[:12500])


len(dataset), dataset[0]
```

```
(12500,
 Data(x=[3, 8], edge_index=[2, 6], edge_attr=[6, 1], y=1, graph_features=[5]))
```

```
dataset[0].edge_index
```

```
tensor([[0, 0, 1, 1, 2, 2],
        [1, 2, 0, 2, 1, 0]], dtype=torch.int32)
```

```
dataset[0].edge_attr
```

Show hidden output

```
dataset[0].edge_attr
```

```
tensor([[0.0006],
        [0.0016],
        [0.0006],
        [0.0011],
        [0.0011],
        [0.0016]])
```

```
dataset[0].edge_attr
```

```
tensor([[0.0006],
        [0.0016],
        [0.0006],
        [0.0011],
        [0.0011],
        [0.0016]])
```

```
for i in range(8):
    quarks = len([j for j in range(i*12500, (i+1)*12500) if y[j]==1])
    gluons = len([j for j in range(i*12500, (i+1)*12500) if y[j]==0])
    print(i, quarks, gluons)
```

```
0 6325 6175
1 6266 6234
2 6247 6253
3 6190 6310
4 6336 6164
5 6170 6330
6 6297 6203
```

```
      7 6169 6331
```

```
batch_size = 64
```

```
data_train, data_val, data_test = torch.utils.data.random_split(dataset, [train_ratio, val_ratio, test_ratio])
```

```
train_dataloader = DataLoader(data_train,
                              batch_size = batch_size,
                              shuffle=True)
val_dataloader = DataLoader(data_val,
                            batch_size = batch_size,
                            shuffle=True)
test_dataloader = DataLoader(data_test,
                         batch_size = batch_size,
                         shuffle=True)
```

```
len(train_dataloader)
```

```
118
```

```
for batch in train_dataloader:
    print(f"Batch node features shape: {batch.x.shape}")
    print(f"Batch edge indices shape: {batch.edge_index.shape}")
    print(f"Batch edge attributes shape: {batch.edge_attr.shape}")
    print(f"Batch target shape: {batch.y.shape}")
    break
```

```
Batch node features shape: torch.Size([192, 8])
Batch edge indices shape: torch.Size([2, 384])
Batch edge attributes shape: torch.Size([384, 1])
Batch target shape: torch.Size([64])
```

Train functions

```
def run_model(model, epoch, loader, lossFn, optimizer, train=True, quantum=False):
    if train:
        model.train()
    else:
        model.eval()

    loss = 0
    net_loss = 0
    correct = 0

    for batch_idx, data in (tqdm(enumerate(loader)) if train else enumerate(loader)):

        target = data.y

        #This will zero out the gradients for this batch.
        optimizer.zero_grad()

        #Run the model on the train data
        output = model(data.x, data.edge_index.type(torch.int64), data.edge_attr, data.batch, data.graph_features)

        target = target.unsqueeze(1).float()

        # Calculate the loss
        loss = lossFn(output, target)
        net_loss += loss.data * batch_size

        if train:
            #dloss/dx for every Variable
            loss.backward()

            #to do a one-step update on our parameter.
            optimizer.step()

        pred = (output > 0).float()
        # since we are working with logits and not probabilities (sigmoid is applied while computing loss), we consider 0 as threshold

#         _, pred = torch.max(F.softmax(output,dim=1).data, dim=1)
```

```
            correct += (pred == target).sum()


        acc = correct / len(loader.dataset)
        net_loss /= len(loader.dataset)

        if train:
            print('Train', end=" ")
        else:
            print("Val", end=" ")

        # Print out the loss
        print('Epoch: {} \tLoss: {:.6f}, Accuracy: {}/{} ({:.0f}%)'.format(
              epoch, net_loss, correct, len(loader.dataset),
              100. * acc))

        return net_loss, acc


    def train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader):

        history = {'train_loss': [],'val_loss':[],'train_acc': [],'val_acc': [] }
        best_model = model
        best_val_loss = 1000

        for epoch in range(epochs):
            train_loss, train_acc = run_model(model, epoch, train_dataloader, lossFn, optimizer)
            val_loss, val_acc = run_model(model, epoch, val_dataloader, lossFn, optimizer, train=False)
            print()

            history['train_loss'].append(train_loss)
            history['train_acc'].append(train_acc)
            history['val_loss'].append(val_loss)
            history['val_acc'].append(val_acc)

            if(best_val_loss > val_loss):
                best_val_loss = val_loss
                best_model = model

        return history, best_model


    def test_eval(model, test_dataloader):

        preds = []
        labels = []
        acc = 0
        for data in test_dataloader:

            target = data.y
            labels.append(target.detach().cpu().numpy())

            output = model(data.x, data.edge_index.type(torch.int64), data.edge_attr, data.batch, data.graph_features)
            preds.append(output.detach().cpu().numpy())  # Convert to numpy array
            # probs = Sigmoid()(output).detach().cpu().numpy()  # Convert to numpy array
            # preds.append(copy.deepcopy(output))

            target = target.unsqueeze(1).float()
            pred = (output > 0).float()
            acc += (pred == target).sum().item()

        acc = acc / len(test_dataloader.dataset)
        print("Test accuracy: ", 100. * acc)

        labels = np.concatenate(labels, axis=0)  # Concatenate lists into a single numpy array
        preds = np.concatenate(preds, axis=0)  # Concatenate lists into a single numpy array


        return labels, preds


    def plot_loss(history, step=2):
        n = len(history['train_loss'])
        x = range(n)
        plt.plot(x, history['train_loss'], label='Train loss')
        plt.plot(x, history['val_loss'], label='Val loss')
        plt.plot(x, history['train_acc'], label='Train acc')
        plt.plot(x, history['val_acc'], label='Val acc')
```

```
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.xticks(range(0,n,step), range(1,n+1,step))

    plt.legend()
    plt.show()


def plot_auc(model, test_dataloader):
    labels, preds = test_eval(model, test_dataloader)
    auc = roc_auc_score(labels, preds)
    fpr, tpr, _ = roc_curve(labels, preds)
    plt.plot(fpr, tpr, label="AUC = {0}".format(auc))
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()


tensor_a = torch.randn(100, 4)

# Tensor of shape (12500, 1)
tensor_b = torch.randint(0, 10, (100, 1))

# Concatenate along the second dimension (dim=1)
result_tensor = torch.cat((tensor_a, tensor_b), dim=1)

print(result_tensor.shape)
```

```
torch.Size([100, 5])
```

```
eg = next(iter(train_dataloader))
eg.x.shape, eg.edge_index.shape, eg.edge_attr.shape, eg.batch.shape, eg.graph_features.shape
```

```
(torch.Size([192, 8]),
 torch.Size([2, 384]),
 torch.Size([384, 1]),
 torch.Size([192]),
 torch.Size([320]))
```

PyTorch GAT

```
from torch_geometric.nn import GATConv


class GAT(torch.nn.Module):
    def __init__(self, input_dim, hidden_dims, output_dim, activ_fn):
        super().__init__()
        layers = []
        layers.append(GATConv(input_dim, hidden_dims[0], edge_dim=1))

        for i in range(len(hidden_dims)-1):
            layers.append(GATConv(hidden_dims[i], hidden_dims[i+1], edge_dim=1))

        self.layers = ModuleList(layers)
        self.activ_fn = activ_fn
        self.classifier = Sequential(Linear(hidden_dims[-1]+5, 8),
                                     ReLU(),
#                                      Linear(16, 8),
#                                      ReLU(),
                                     Linear(8, output_dims)
                                    )

    def forward(self, x, edge_index, edge_weight, batch, graph_features):
        h = x
        for i in range(len(self.layers)):
            h = self.layers[i](h, edge_index, edge_weight)
            h = self.activ_fn(h)

        h = global_mean_pool(h, batch)   # readout layer to get the embedding for each graph in batch
        h = torch.cat((h, graph_features.float().reshape(-1,5)), dim=1)
        h = self.classifier(h)
        return h
```

PyTorch GCN

```
class GNN(torch.nn.Module):
    def __init__(self, input_dim, hidden_dims, output_dim, activ_fn):
        super().__init__()
        layers = []
        layers.append(GCNConv(input_dim, hidden_dims[0]))

        for i in range(len(hidden_dims)-1):
            layers.append(GCNConv(hidden_dims[i], hidden_dims[i+1]))

        self.layers = ModuleList(layers)
        self.activ_fn = activ_fn
        self.classifier = Sequential(Linear(hidden_dims[-1]+5, 16),
                                     ReLU(),
                                     Linear(16, 8),
                                     ReLU(),
                                     Linear(8, output_dims)
                                    )

    def forward(self, x, edge_index, edge_weight, batch, graph_features):
        h = x
        for i in range(len(self.layers)):
            h = self.layers[i](h, edge_index, edge_weight)
            h = self.activ_fn(h)

        h = global_mean_pool(h, batch)   # readout layer to get the embedding for each graph in batch
        h = torch.cat((h, graph_features.float().reshape(-1,5)), dim=1)
        h = self.classifier(h)
        return h


def find_ideal_particles(X, y, num_particles, min_particles, max_particles):
    train_loss = dict()
    val_loss = dict()
    train_acc = dict()
    val_acc = dict()
    auc = dict()

    for i in range(min_particles, max_particles+1):

        print("Particles per jet:",i)
        train_loss_i = []
        val_loss_i = []
        train_acc_i = []
        val_acc_i = []
        auc_i = []

        for j in range(8):

            X_filtered = X[j*12500:min(len(X), (j+1)*12500), :i, :]
            y_filtered = y[j*12500:min(len(X), (j+1)*12500)]
            num_particles_filtered = num_particles[j*12500:min(len(X), (j+1)*12500)]
            dataset = create_graph_dataset(X_filtered,y_filtered, num_particles_filtered)

            data_train, data_val, data_test = torch.utils.data.random_split(dataset, [train_ratio, val_ratio, test_ratio])

            train_dataloader = DataLoader(data_train,
                                  batch_size = batch_size,
                                  shuffle=True)
            val_dataloader = DataLoader(data_val,
                                        batch_size = batch_size,
                                        shuffle=True)
            test_dataloader = DataLoader(data_test,
                             batch_size = batch_size,
                             shuffle=True)

            model = GNN(input_dims,hidden_dims,output_dims, activ_fn=ReLU())

            optimizer = Adam(model.parameters(), 1e-2)
            lossFn = BCEWithLogitsLoss()
            history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)

            ind = np.argmin(history['val_loss'])
            train_loss_i.append(history['train_loss'][ind])
            val_loss_i.append(history['val_loss'][ind])
```

```
            train_acc_i.append(history['train_acc'][ind])
            val_acc_i.append(history['val_acc'][ind])

            labels, preds = test_eval(best_model, test_dataloader)
            auc_i.append(roc_auc_score(labels, preds))

            plot_loss(history)
            print("*"*100)
            print()

        print()
        print()

        train_loss[i] = train_loss_i
        val_loss[i] = val_loss_i
        train_acc[i] = train_acc_i
        val_acc[i] = val_acc_i
        auc[i] = auc_i

    return train_loss, val_loss, train_acc, val_acc, auc
```

With all particles, complete graph, edge_weights

```
hidden_dims = [8,8]
model = GNN(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

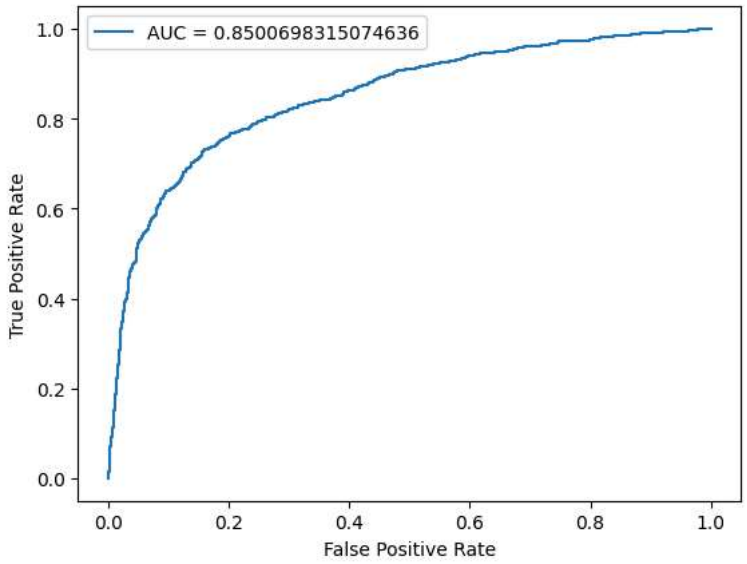
```
GNN(
  (layers): ModuleList(
    (0-1): 2 x GCNConv(8, 8)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=8, bias=True)
    (3): ReLU()
    (4): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
epochs = 40
optimizer = Adam(model.parameters(), 1e-2)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

Show hidden output

```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  78.4
```



100K jets

GAT - 5 particles

```
hidden_dims = [8,8]
model = GAT(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

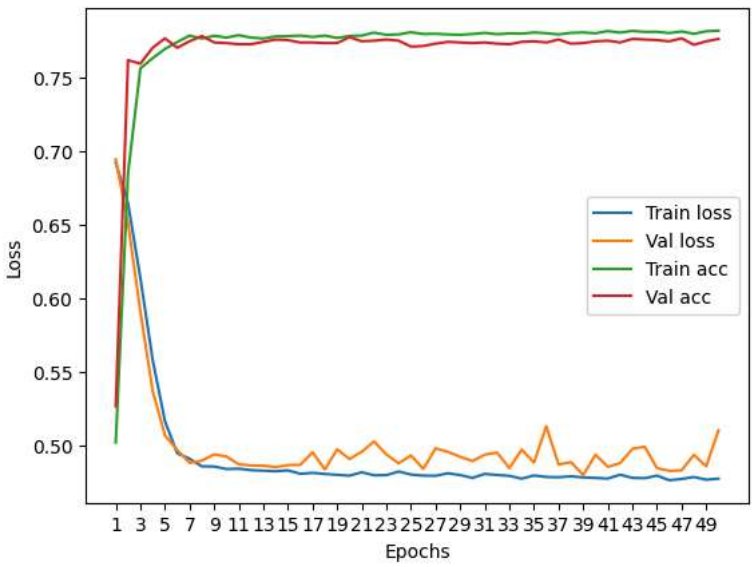
```
GAT(
  (layers): ModuleList(
    (0-1): 2 x GATConv(8, 8, heads=1)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
epochs = 50
optimizer = Adam(model.parameters(), 1e-3)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

```
118it [00:01, 101.27it/s]
Train Epoch: 0  Loss: 0.692132, Accuracy: 3766/7500 (50%)
Val Epoch: 0    Loss: 0.694276, Accuracy: 1317/2500 (53%)

118it [00:01, 102.21it/s]
Train Epoch: 1  Loss: 0.664310, Accuracy: 5137/7500 (68%)
Val Epoch: 1    Loss: 0.650777, Accuracy: 1904/2500 (76%)

118it [00:01, 107.18it/s]
Train Epoch: 2  Loss: 0.613835, Accuracy: 5670/7500 (76%)
Val Epoch: 2    Loss: 0.591179, Accuracy: 1898/2500 (76%)

118it [00:01, 99.66it/s]
Train Epoch: 3  Loss: 0.558224, Accuracy: 5722/7500 (76%)
Val Epoch: 3    Loss: 0.537232, Accuracy: 1925/2500 (77%)

118it [00:01, 73.86it/s]
Train Epoch: 4  Loss: 0.516582, Accuracy: 5769/7500 (77%)
Val Epoch: 4    Loss: 0.506945, Accuracy: 1941/2500 (78%)

118it [00:01, 85.74it/s]
Train Epoch: 5  Loss: 0.494751, Accuracy: 5804/7500 (77%)
Val Epoch: 5    Loss: 0.497063, Accuracy: 1925/2500 (77%)

118it [00:01, 104.00it/s]
Train Epoch: 6  Loss: 0.491124, Accuracy: 5837/7500 (78%)
Val Epoch: 6    Loss: 0.488292, Accuracy: 1936/2500 (77%)

118it [00:01, 101.29it/s]
Train Epoch: 7  Loss: 0.485904, Accuracy: 5822/7500 (78%)
Val Epoch: 7    Loss: 0.489798, Accuracy: 1945/2500 (78%)

118it [00:01, 96.94it/s]
Train Epoch: 8  Loss: 0.485817, Accuracy: 5836/7500 (78%)
Val Epoch: 8    Loss: 0.494080, Accuracy: 1934/2500 (77%)

118it [00:01, 92.92it/s]
Train Epoch: 9  Loss: 0.484115, Accuracy: 5827/7500 (78%)
Val Epoch: 9    Loss: 0.492596, Accuracy: 1933/2500 (77%)

118it [00:01, 100.91it/s]
Train Epoch: 10         Loss: 0.484345, Accuracy: 5839/7500 (78%)
Val Epoch: 10   Loss: 0.487345, Accuracy: 1931/2500 (77%)

118it [00:01, 105.36it/s]
Train Epoch: 11         Loss: 0.483349, Accuracy: 5827/7500 (78%)
Val Epoch: 11   Loss: 0.486505, Accuracy: 1931/2500 (77%)

118it [00:01, 74.20it/s]
Train Epoch: 12         Loss: 0.482968, Accuracy: 5822/7500 (78%)
Val Epoch: 12   Loss: 0.486379, Accuracy: 1935/2500 (77%)

118it [00:01, 84.81it/s]
Train Epoch: 13         Loss: 0.482617, Accuracy: 5833/7500 (78%)
```

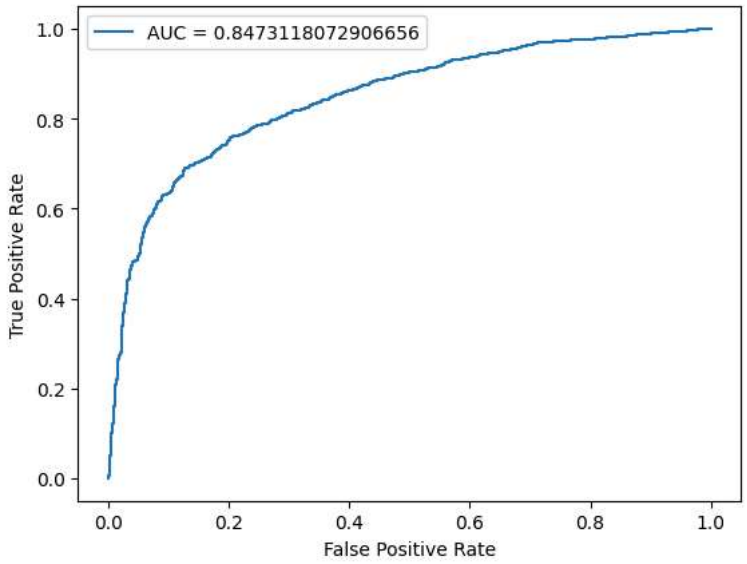
```
Val Epoch: 13   Loss: 0.485399, Accuracy: 1939/2500 (78%)

118it [00:01, 103.89it/s]
Train Epoch: 14         Loss: 0.483182  Accuracy: 5834/7500 (78%)
```

```
plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.44
```



GAT - With 5 particles, edge_weights, (total particles, photon, charged hadron, neutral hadron, charge of jet with k = 0.2, 0.5, 0.7) count

```
input_dims = 8
hidden_dims = [8,8]
model = GAT(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

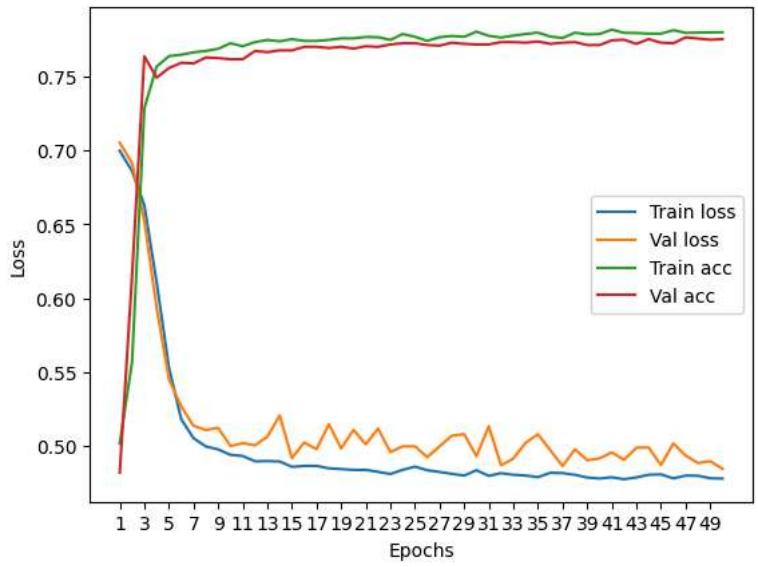
```
GAT(
  (layers): ModuleList(
    (0-1): 2 x GATConv(8, 8, heads=1)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
```

```
        )
      )
```

```
epochs = 50
optimizer = Adam(model.parameters(), 1e-3)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

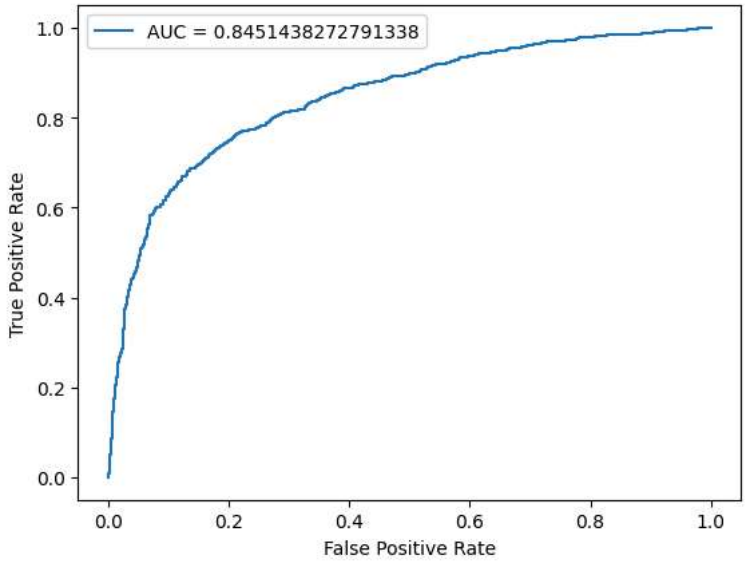
```
118it [00:01, 100.81it/s]
Train Epoch: 0  Loss: 0.699719, Accuracy: 3762/7500 (50%)
Val Epoch: 0    Loss: 0.705165, Accuracy: 1205/2500 (48%)

118it [00:01, 106.35it/s]
Train Epoch: 1  Loss: 0.686020, Accuracy: 4177/7500 (56%)
Val Epoch: 1    Loss: 0.691418, Accuracy: 1537/2500 (61%)

118it [00:01, 64.14it/s]
Train Epoch: 2  Loss: 0.662808, Accuracy: 5464/7500 (73%)
Val Epoch: 2    Loss: 0.652361, Accuracy: 1909/2500 (76%)

118it [00:01, 103.97it/s]
Train Epoch: 3  Loss: 0.610638, Accuracy: 5675/7500 (76%)
Val Epoch: 3    Loss: 0.593615, Accuracy: 1873/2500 (75%)

118it [00:01, 102.00it/s]
Train Epoch: 4  Loss: 0.553008, Accuracy: 5728/7500 (76%)
Val Epoch: 4    Loss: 0.545165, Accuracy: 1889/2500 (76%)

118it [00:01, 104.00it/s]
Train Epoch: 5  Loss: 0.518111, Accuracy: 5735/7500 (76%)
Val Epoch: 5    Loss: 0.526784, Accuracy: 1898/2500 (76%)

118it [00:01, 105.84it/s]
Train Epoch: 6  Loss: 0.505320, Accuracy: 5747/7500 (77%)
Val Epoch: 6    Loss: 0.513670, Accuracy: 1897/2500 (76%)

118it [00:01, 105.57it/s]
Train Epoch: 7  Loss: 0.499722, Accuracy: 5754/7500 (77%)
Val Epoch: 7    Loss: 0.510855, Accuracy: 1907/2500 (76%)

118it [00:01, 98.99it/s]
Train Epoch: 8  Loss: 0.497723, Accuracy: 5765/7500 (77%)
Val Epoch: 8    Loss: 0.512349, Accuracy: 1906/2500 (76%)

118it [00:01, 103.62it/s]
Train Epoch: 9  Loss: 0.494049, Accuracy: 5793/7500 (77%)
Val Epoch: 9    Loss: 0.500009, Accuracy: 1904/2500 (76%)

118it [00:01, 71.46it/s]
Train Epoch: 10         Loss: 0.493275, Accuracy: 5778/7500 (77%)
Val Epoch: 10   Loss: 0.501939, Accuracy: 1904/2500 (76%)

118it [00:01, 98.78it/s]
Train Epoch: 11         Loss: 0.489648, Accuracy: 5799/7500 (77%)
Val Epoch: 11   Loss: 0.500495, Accuracy: 1918/2500 (77%)

118it [00:01, 101.31it/s]
Train Epoch: 12         Loss: 0.489869, Accuracy: 5810/7500 (77%)
Val Epoch: 12   Loss: 0.506325, Accuracy: 1916/2500 (77%)

118it [00:01, 104.07it/s]
Train Epoch: 13         Loss: 0.489553, Accuracy: 5803/7500 (77%)
Val Epoch: 13   Loss: 0.520634, Accuracy: 1919/2500 (77%)

118it [00:01, 96.73it/s]
Train Epoch: 14         Loss: 0.485953, Accuracy: 5814/7500 (78%)
```

```
plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.4
```



12500 jets

With 10 particles, charge, complete graph, edge_weights, (photon, CH, NH) count

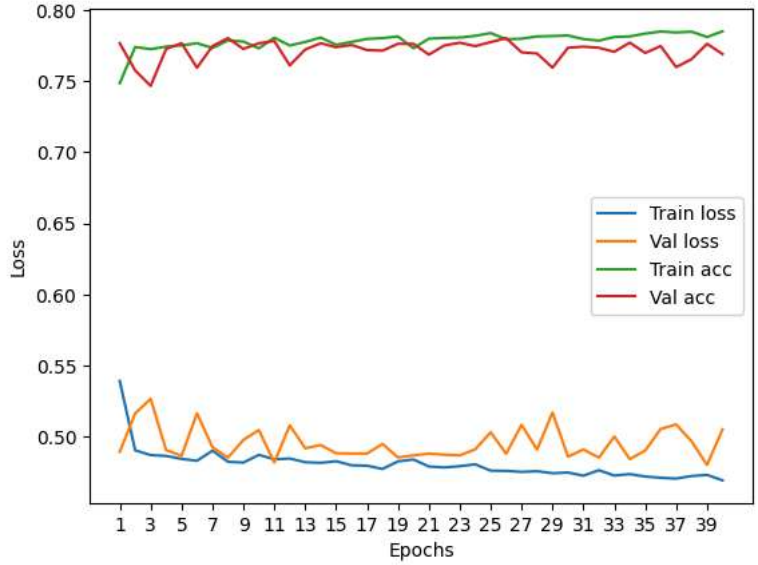
```
input_dims = 9
hidden_dims = [9, 9]
model = GNN(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

```
GNN(
  (layers): ModuleList(
    (0-1): 2 x GCNConv(9, 9)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=14, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=8, bias=True)
    (3): ReLU()
    (4): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
epochs = 40
optimizer = Adam(model.parameters(), 1e-2)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

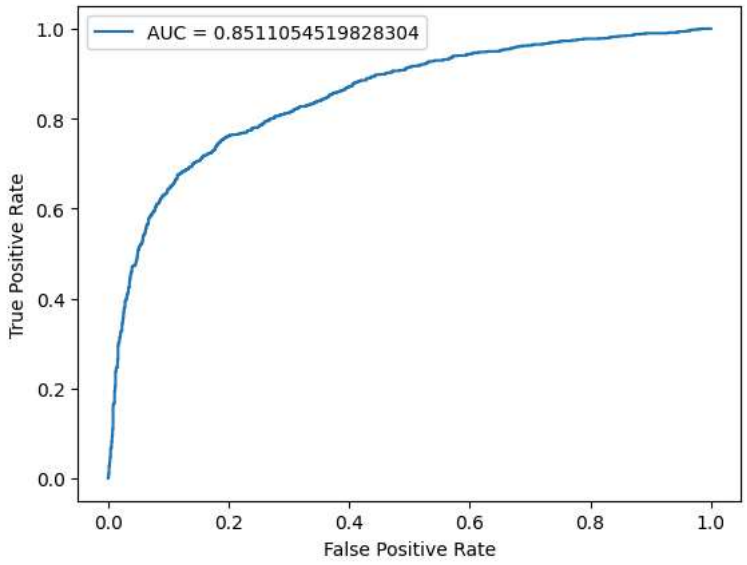
```
118it [00:01, 109.11it/s]
Train Epoch: 0  Loss: 0.538979, Accuracy: 5615/7500 (75%)
Val Epoch: 0    Loss: 0.489073, Accuracy: 1942/2500 (78%)

118it [00:01, 83.59it/s]
Train Epoch: 1  Loss: 0.490145, Accuracy: 5806/7500 (77%)
Val Epoch: 1    Loss: 0.516104, Accuracy: 1894/2500 (76%)

118it [00:00, 136.06it/s]
Train Epoch: 2  Loss: 0.486803, Accuracy: 5795/7500 (77%)
Val Epoch: 2    Loss: 0.526379, Accuracy: 1867/2500 (75%)

118it [00:00, 134.59it/s]
Train Epoch: 3  Loss: 0.486189, Accuracy: 5808/7500 (77%)
Val Epoch: 3    Loss: 0.490307, Accuracy: 1932/2500 (77%)

118it [00:00, 133.42it/s]
Train Epoch: 4  Loss: 0.484108, Accuracy: 5815/7500 (78%)
Val Epoch: 4    Loss: 0.486295, Accuracy: 1942/2500 (78%)

118it [00:00, 134.35it/s]
Train Epoch: 5  Loss: 0.482799, Accuracy: 5826/7500 (78%)
Val Epoch: 5    Loss: 0.516221, Accuracy: 1899/2500 (76%)

118it [00:00, 125.07it/s]
Train Epoch: 6  Loss: 0.489915, Accuracy: 5801/7500 (77%)
Val Epoch: 6    Loss: 0.492293, Accuracy: 1937/2500 (77%)

118it [00:00, 133.52it/s]
Train Epoch: 7  Loss: 0.482125, Accuracy: 5841/7500 (78%)
Val Epoch: 7    Loss: 0.484991, Accuracy: 1951/2500 (78%)

118it [00:00, 134.72it/s]
Train Epoch: 8  Loss: 0.481525, Accuracy: 5835/7500 (78%)
Val Epoch: 8    Loss: 0.497472, Accuracy: 1932/2500 (77%)

118it [00:00, 137.60it/s]
Train Epoch: 9  Loss: 0.486945, Accuracy: 5799/7500 (77%)
Val Epoch: 9    Loss: 0.504390, Accuracy: 1942/2500 (78%)

118it [00:00, 126.98it/s]
Train Epoch: 10         Loss: 0.483824, Accuracy: 5855/7500 (78%)
Val Epoch: 10   Loss: 0.481593, Accuracy: 1946/2500 (78%)

118it [00:01, 97.09it/s]
Train Epoch: 11         Loss: 0.484369, Accuracy: 5814/7500 (78%)
Val Epoch: 11   Loss: 0.507701, Accuracy: 1903/2500 (76%)

118it [00:00, 119.86it/s]
Train Epoch: 12         Loss: 0.481780, Accuracy: 5833/7500 (78%)
Val Epoch: 12   Loss: 0.491599, Accuracy: 1931/2500 (77%)

118it [00:01, 115.11it/s]
Train Epoch: 13         Loss: 0.481362, Accuracy: 5856/7500 (78%)
Val Epoch: 13   Loss: 0.493904, Accuracy: 1942/2500 (78%)

118it [00:00, 130.49it/s]
Train Epoch: 14         Loss: 0.482461, Accuracy: 5818/7500 (78%)
```

```
input_dims = 8  # Correct the input dimensions to 8 to match your feature engineering
hidden_dims = [8, 8]  # Adjust hidden dimensions if necessary
model = GNN(input_dims, hidden_dims, output_dims, activ_fn=ReLU())


plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.72
```



GAT - With 10 particles, edge_weights, (total particles, photon, charged hadron, neutral hadron, net positive charge of jet) count

```
input_dims = 8
hidden_dims = [8,8]
model = GAT(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

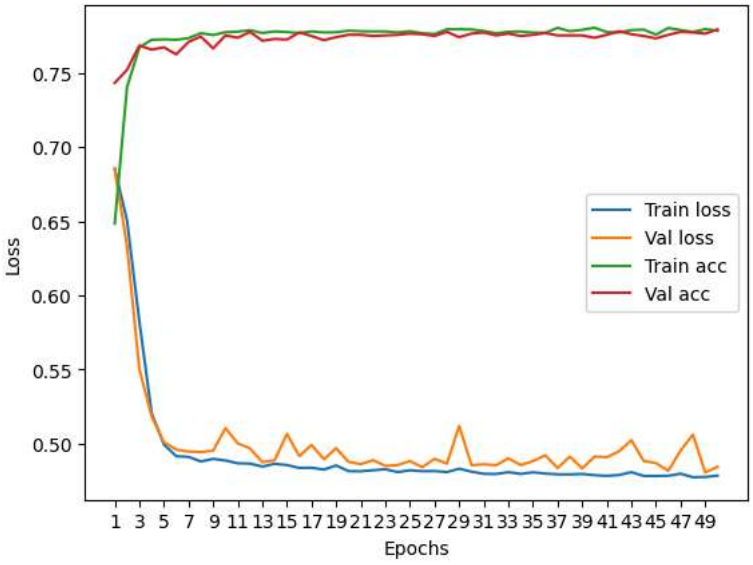
```
GAT(
  (layers): ModuleList(
    (0-1): 2 x GATConv(8, 8, heads=1)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
epochs = 50
optimizer = Adam(model.parameters(), 1e-3)
```

```
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

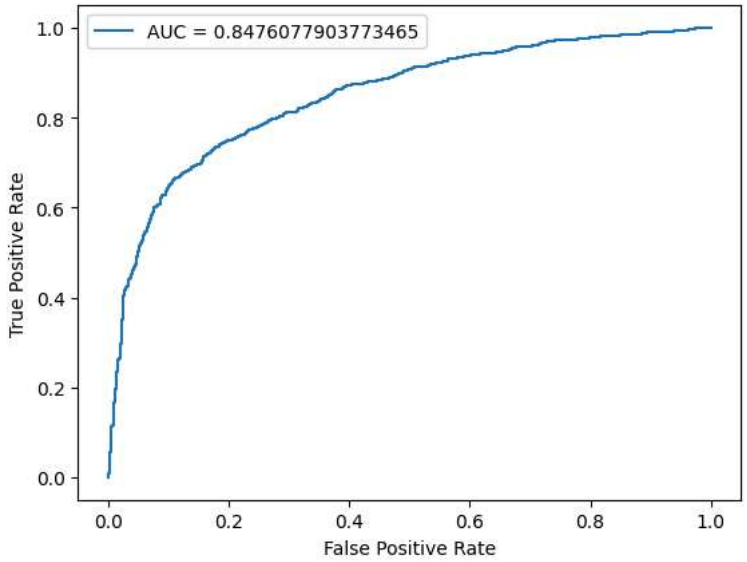
```
118it [00:01, 91.88it/s]
Train Epoch: 0  Loss: 0.685387, Accuracy: 4863/7500 (65%)
Val Epoch: 0    Loss: 0.684917, Accuracy: 1858/2500 (74%)

118it [00:01, 98.76it/s]
Train Epoch: 1  Loss: 0.650442, Accuracy: 5553/7500 (74%)
Val Epoch: 1    Loss: 0.633196, Accuracy: 1880/2500 (75%)

118it [00:01, 100.46it/s]
Train Epoch: 2  Loss: 0.582446, Accuracy: 5753/7500 (77%)
Val Epoch: 2    Loss: 0.550401, Accuracy: 1921/2500 (77%)

118it [00:01, 78.72it/s]
Train Epoch: 3  Loss: 0.520584, Accuracy: 5792/7500 (77%)
Val Epoch: 3    Loss: 0.518366, Accuracy: 1914/2500 (77%)

118it [00:01, 72.38it/s]
Train Epoch: 4  Loss: 0.499471, Accuracy: 5794/7500 (77%)
Val Epoch: 4    Loss: 0.501083, Accuracy: 1918/2500 (77%)

118it [00:01, 87.48it/s]
Train Epoch: 5  Loss: 0.491651, Accuracy: 5792/7500 (77%)
Val Epoch: 5    Loss: 0.496097, Accuracy: 1906/2500 (76%)

118it [00:01, 101.99it/s]
Train Epoch: 6  Loss: 0.491181, Accuracy: 5800/7500 (77%)
Val Epoch: 6    Loss: 0.494763, Accuracy: 1927/2500 (77%)

118it [00:01, 88.02it/s]
Train Epoch: 7  Loss: 0.488096, Accuracy: 5825/7500 (78%)
Val Epoch: 7    Loss: 0.494441, Accuracy: 1936/2500 (77%)

118it [00:01, 100.90it/s]
Train Epoch: 8  Loss: 0.489826, Accuracy: 5816/7500 (78%)
Val Epoch: 8    Loss: 0.495366, Accuracy: 1916/2500 (77%)

118it [00:01, 99.45it/s]
Train Epoch: 9  Loss: 0.488729, Accuracy: 5830/7500 (78%)
Val Epoch: 9    Loss: 0.510533, Accuracy: 1938/2500 (78%)

118it [00:01, 97.60it/s]
Train Epoch: 10         Loss: 0.486876, Accuracy: 5833/7500 (78%)
Val Epoch: 10   Loss: 0.500250, Accuracy: 1934/2500 (77%)

118it [00:01, 74.13it/s]
Train Epoch: 11         Loss: 0.486627, Accuracy: 5840/7500 (78%)
Val Epoch: 11   Loss: 0.496876, Accuracy: 1944/2500 (78%)

118it [00:01, 83.68it/s]
Train Epoch: 12         Loss: 0.484622, Accuracy: 5826/7500 (78%)
Val Epoch: 12   Loss: 0.487627, Accuracy: 1929/2500 (77%)

118it [00:01, 96.64it/s]
Train Epoch: 13         Loss: 0.486515, Accuracy: 5834/7500 (78%)
Val Epoch: 13   Loss: 0.488973, Accuracy: 1932/2500 (77%)

118it [00:01, 98.26it/s]
Train Epoch: 14         Loss: 0.485664, Accuracy: 5831/7500 (78%)
```

```
plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.16
```



GAT - With 5 particles, edge_weights, (total particles, photon, charged hadron, neutral hadron, charge of jet with k = 0.2, 0.5, 0.7) count

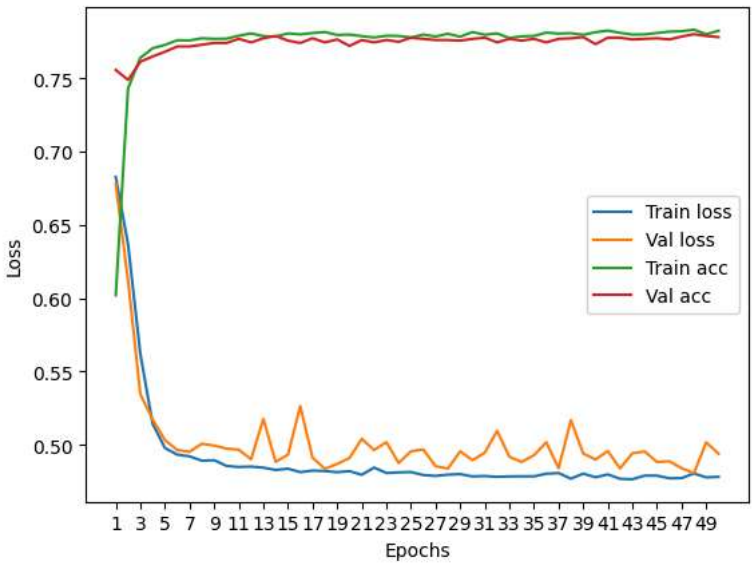
```
input_dims = 8
hidden_dims = [8,8]
model = GAT(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

```
GAT(
  (layers): ModuleList(
    (0-1): 2 x GATConv(8, 8, heads=1)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
epochs = 50
optimizer = Adam(model.parameters(), 1e-3)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

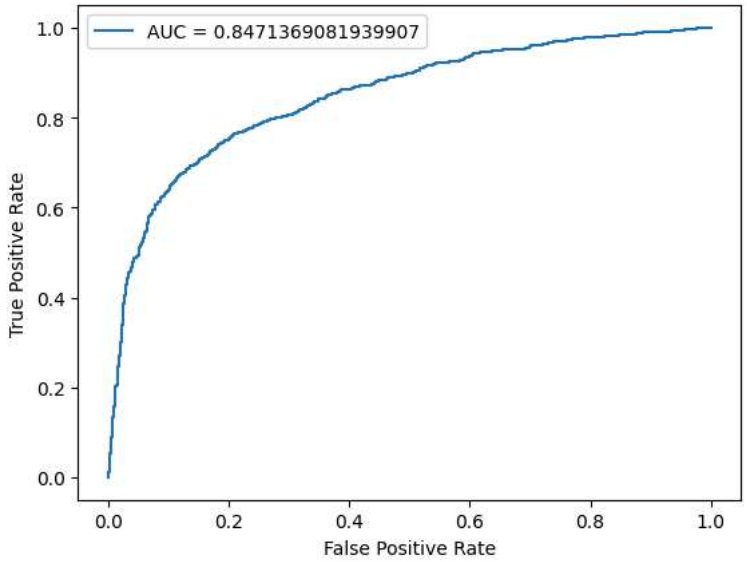
```
118it [00:01, 97.12it/s]
Train Epoch: 0  Loss: 0.682695, Accuracy: 4515/7500 (60%)
Val Epoch: 0    Loss: 0.677937, Accuracy: 1889/2500 (76%)

118it [00:01, 69.64it/s]
Train Epoch: 1  Loss: 0.636739, Accuracy: 5573/7500 (74%)
Val Epoch: 1    Loss: 0.612089, Accuracy: 1872/2500 (75%)

118it [00:01, 97.35it/s]
Train Epoch: 2  Loss: 0.561951, Accuracy: 5729/7500 (76%)
Val Epoch: 2    Loss: 0.534475, Accuracy: 1903/2500 (76%)

118it [00:01, 99.59it/s]
Train Epoch: 3  Loss: 0.514315, Accuracy: 5778/7500 (77%)
Val Epoch: 3    Loss: 0.517273, Accuracy: 1912/2500 (76%)

118it [00:01, 90.90it/s]
Train Epoch: 4  Loss: 0.497809, Accuracy: 5795/7500 (77%)
Val Epoch: 4    Loss: 0.503164, Accuracy: 1920/2500 (77%)

118it [00:01, 99.83it/s]
Train Epoch: 5  Loss: 0.493302, Accuracy: 5819/7500 (78%)
Val Epoch: 5    Loss: 0.496413, Accuracy: 1929/2500 (77%)

118it [00:01, 92.27it/s]
Train Epoch: 6  Loss: 0.492128, Accuracy: 5818/7500 (78%)
Val Epoch: 6    Loss: 0.495279, Accuracy: 1929/2500 (77%)

118it [00:01, 104.25it/s]
Train Epoch: 7  Loss: 0.489178, Accuracy: 5829/7500 (78%)
Val Epoch: 7    Loss: 0.500700, Accuracy: 1932/2500 (77%)

118it [00:01, 82.07it/s]
Train Epoch: 8  Loss: 0.489478, Accuracy: 5826/7500 (78%)
Val Epoch: 8    Loss: 0.499473, Accuracy: 1935/2500 (77%)

118it [00:01, 74.45it/s]
Train Epoch: 9  Loss: 0.485633, Accuracy: 5827/7500 (78%)
Val Epoch: 9    Loss: 0.497397, Accuracy: 1935/2500 (77%)

118it [00:01, 100.36it/s]
Train Epoch: 10         Loss: 0.484871, Accuracy: 5842/7500 (78%)
Val Epoch: 10   Loss: 0.496674, Accuracy: 1942/2500 (78%)

118it [00:01, 99.81it/s]
Train Epoch: 11         Loss: 0.485142, Accuracy: 5854/7500 (78%)
Val Epoch: 11   Loss: 0.490136, Accuracy: 1936/2500 (77%)

118it [00:01, 101.50it/s]
Train Epoch: 12         Loss: 0.484417, Accuracy: 5840/7500 (78%)
Val Epoch: 12   Loss: 0.517617, Accuracy: 1943/2500 (78%)

118it [00:01, 103.65it/s]
Train Epoch: 13         Loss: 0.482848, Accuracy: 5837/7500 (78%)
Val Epoch: 13   Loss: 0.488381, Accuracy: 1947/2500 (78%)

118it [00:01, 99.38it/s]
Train Epoch: 14         Loss: 0.483712, Accuracy: 5854/7500 (78%)
```

```
plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.72
```



GAT - With 3 particles, edge_weights, (total particles, photon, charged hadron, neutral hadron, net positive charge of jet) count

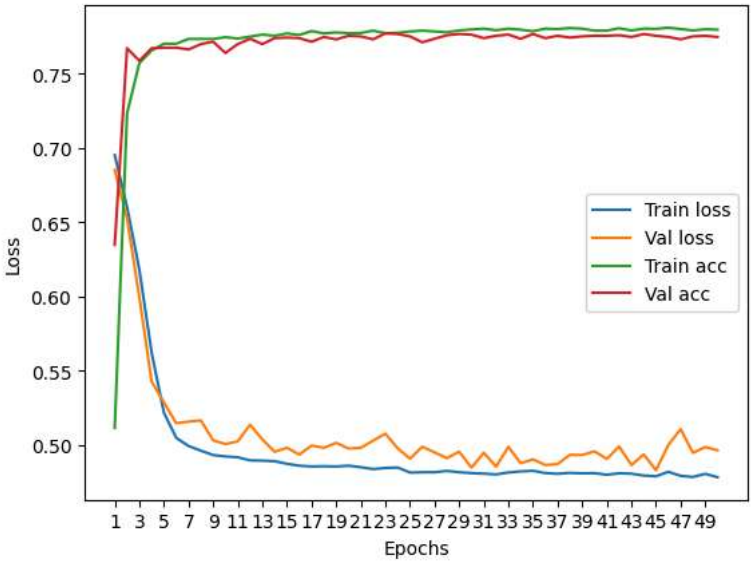
```
input_dims = 8
hidden_dims = [8,8]
model = GAT(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

```
GAT(
  (layers): ModuleList(
    (0-1): 2 x GATConv(8, 8, heads=1)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
epochs = 50
optimizer = Adam(model.parameters(), 1e-3)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

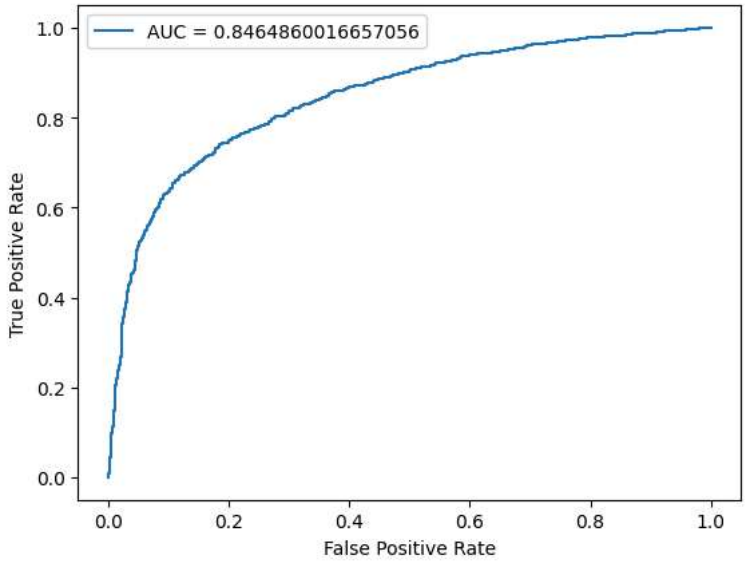
```
118it [00:01, 103.49it/s]
Train Epoch: 0  Loss: 0.695235, Accuracy: 3836/7500 (51%)
Val Epoch: 0    Loss: 0.685250, Accuracy: 1587/2500 (63%)

118it [00:01, 81.55it/s]
Train Epoch: 1  Loss: 0.660289, Accuracy: 5425/7500 (72%)
Val Epoch: 1    Loss: 0.652561, Accuracy: 1918/2500 (77%)

118it [00:01, 101.58it/s]
Train Epoch: 2  Loss: 0.617936, Accuracy: 5679/7500 (76%)
Val Epoch: 2    Loss: 0.599461, Accuracy: 1897/2500 (76%)

118it [00:01, 104.92it/s]
Train Epoch: 3  Loss: 0.562155, Accuracy: 5743/7500 (77%)
Val Epoch: 3    Loss: 0.542601, Accuracy: 1918/2500 (77%)

118it [00:01, 102.83it/s]
Train Epoch: 4  Loss: 0.521643, Accuracy: 5777/7500 (77%)
Val Epoch: 4    Loss: 0.528584, Accuracy: 1919/2500 (77%)

118it [00:01, 99.34it/s]
Train Epoch: 5  Loss: 0.504740, Accuracy: 5777/7500 (77%)
Val Epoch: 5    Loss: 0.514621, Accuracy: 1919/2500 (77%)

118it [00:01, 86.26it/s]
Train Epoch: 6  Loss: 0.499283, Accuracy: 5801/7500 (77%)
Val Epoch: 6    Loss: 0.515653, Accuracy: 1916/2500 (77%)

118it [00:01, 70.05it/s]
Train Epoch: 7  Loss: 0.496100, Accuracy: 5801/7500 (77%)
Val Epoch: 7    Loss: 0.516481, Accuracy: 1925/2500 (77%)

118it [00:01, 103.34it/s]
Train Epoch: 8  Loss: 0.493141, Accuracy: 5801/7500 (77%)
Val Epoch: 8    Loss: 0.503077, Accuracy: 1929/2500 (77%)

118it [00:01, 103.37it/s]
Train Epoch: 9  Loss: 0.492219, Accuracy: 5810/7500 (77%)
Val Epoch: 9    Loss: 0.500580, Accuracy: 1910/2500 (76%)

118it [00:01, 108.45it/s]
Train Epoch: 10         Loss: 0.491631, Accuracy: 5803/7500 (77%)
Val Epoch: 10   Loss: 0.502358, Accuracy: 1925/2500 (77%)

118it [00:01, 103.28it/s]
Train Epoch: 11         Loss: 0.489585, Accuracy: 5813/7500 (78%)
Val Epoch: 11   Loss: 0.513639, Accuracy: 1934/2500 (77%)

118it [00:01, 99.85it/s]
Train Epoch: 12         Loss: 0.489404, Accuracy: 5823/7500 (78%)
Val Epoch: 12   Loss: 0.503633, Accuracy: 1925/2500 (77%)

118it [00:01, 99.56it/s]
Train Epoch: 13         Loss: 0.489012, Accuracy: 5817/7500 (78%)
Val Epoch: 13   Loss: 0.495383, Accuracy: 1935/2500 (77%)

118it [00:01, 95.08it/s]
Train Epoch: 14         Loss: 0.487287, Accuracy: 5829/7500 (78%)
```

```
plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.4
```



GAT - With 3 particles, edge_weights, (total particles, photon, charged hadron, neutral hadron, net positive charge of jet) count

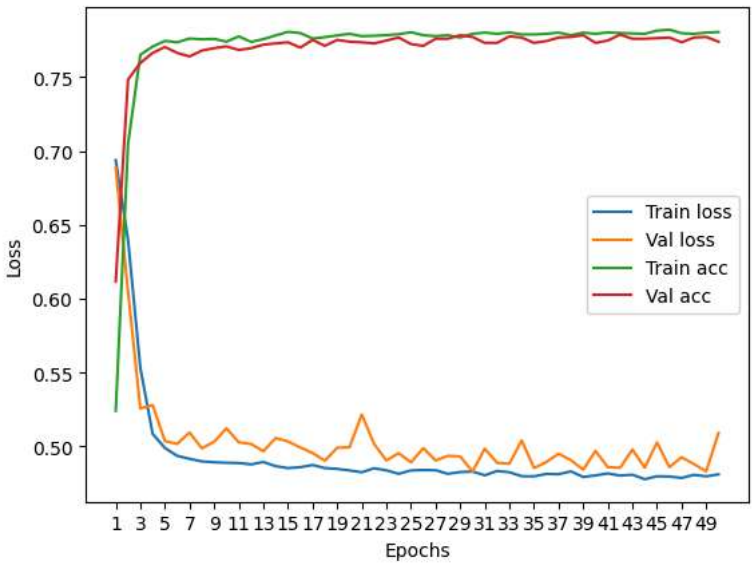
```
input_dims = 8
hidden_dims = [8,8]
model = GAT(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

```
GAT(
  (layers): ModuleList(
    (0-1): 2 x GATConv(8, 8, heads=1)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
epochs = 50
optimizer = Adam(model.parameters(), 1e-3)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

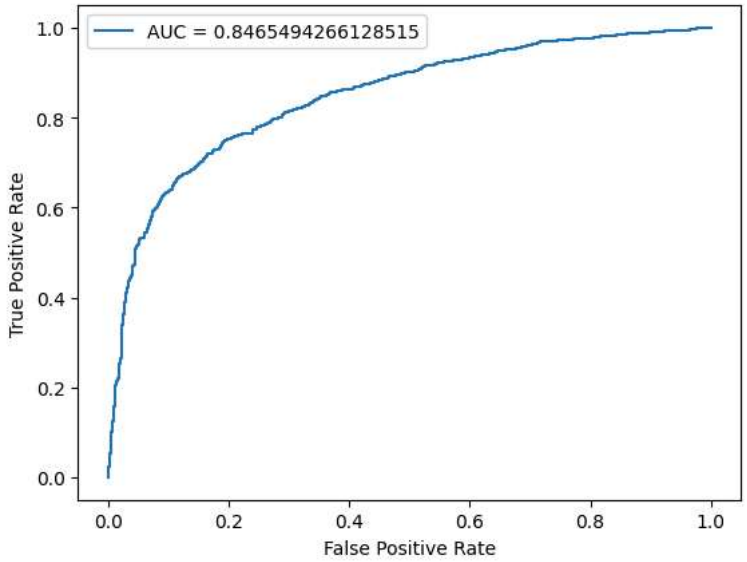
```
118it [00:01, 90.82it/s]
Train Epoch: 0  Loss: 0.693794, Accuracy: 3931/7500 (52%)
Val Epoch: 0    Loss: 0.688594, Accuracy: 1529/2500 (61%)

118it [00:01, 87.52it/s]
Train Epoch: 1  Loss: 0.639793, Accuracy: 5286/7500 (70%)
Val Epoch: 1    Loss: 0.604782, Accuracy: 1871/2500 (75%)

118it [00:01, 87.50it/s]
Train Epoch: 2  Loss: 0.553125, Accuracy: 5739/7500 (77%)
Val Epoch: 2    Loss: 0.525654, Accuracy: 1899/2500 (76%)

118it [00:01, 89.67it/s]
Train Epoch: 3  Loss: 0.508503, Accuracy: 5780/7500 (77%)
Val Epoch: 3    Loss: 0.528039, Accuracy: 1916/2500 (77%)

118it [00:01, 78.47it/s]
Train Epoch: 4  Loss: 0.498917, Accuracy: 5809/7500 (77%)
Val Epoch: 4    Loss: 0.503604, Accuracy: 1926/2500 (77%)

118it [00:01, 60.10it/s]
Train Epoch: 5  Loss: 0.493610, Accuracy: 5802/7500 (77%)
Val Epoch: 5    Loss: 0.501740, Accuracy: 1916/2500 (77%)

118it [00:01, 109.90it/s]
Train Epoch: 6  Loss: 0.491602, Accuracy: 5821/7500 (78%)
Val Epoch: 6    Loss: 0.509426, Accuracy: 1910/2500 (76%)

118it [00:01, 99.58it/s]
Train Epoch: 7  Loss: 0.489867, Accuracy: 5817/7500 (78%)
Val Epoch: 7    Loss: 0.498681, Accuracy: 1920/2500 (77%)

118it [00:01, 102.32it/s]
Train Epoch: 8  Loss: 0.489312, Accuracy: 5819/7500 (78%)
Val Epoch: 8    Loss: 0.503171, Accuracy: 1924/2500 (77%)

118it [00:01, 100.06it/s]
Train Epoch: 9  Loss: 0.488934, Accuracy: 5805/7500 (77%)
Val Epoch: 9    Loss: 0.512275, Accuracy: 1927/2500 (77%)

118it [00:01, 105.31it/s]
Train Epoch: 10         Loss: 0.488762, Accuracy: 5832/7500 (78%)
Val Epoch: 10   Loss: 0.502771, Accuracy: 1921/2500 (77%)

118it [00:01, 107.93it/s]
Train Epoch: 11         Loss: 0.487846, Accuracy: 5804/7500 (77%)
Val Epoch: 11   Loss: 0.501685, Accuracy: 1924/2500 (77%)

118it [00:01, 102.42it/s]
Train Epoch: 12         Loss: 0.489454, Accuracy: 5818/7500 (78%)
Val Epoch: 12   Loss: 0.496764, Accuracy: 1930/2500 (77%)

118it [00:01, 68.37it/s]
Train Epoch: 13         Loss: 0.486644, Accuracy: 5837/7500 (78%)
Val Epoch: 13   Loss: 0.505664, Accuracy: 1932/2500 (77%)

118it [00:01, 105.48it/s]
Train Epoch: 14         Loss: 0.485322, Accuracy: 5855/7500 (78%)
```

```
plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.2
```



With 3 particles, edge_weights, (total particles, photon, charged hadron, neutral hadron, net positive charge of jet) count

```
input_dims = 8
hidden_dims = [8,8]
model = GNN(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

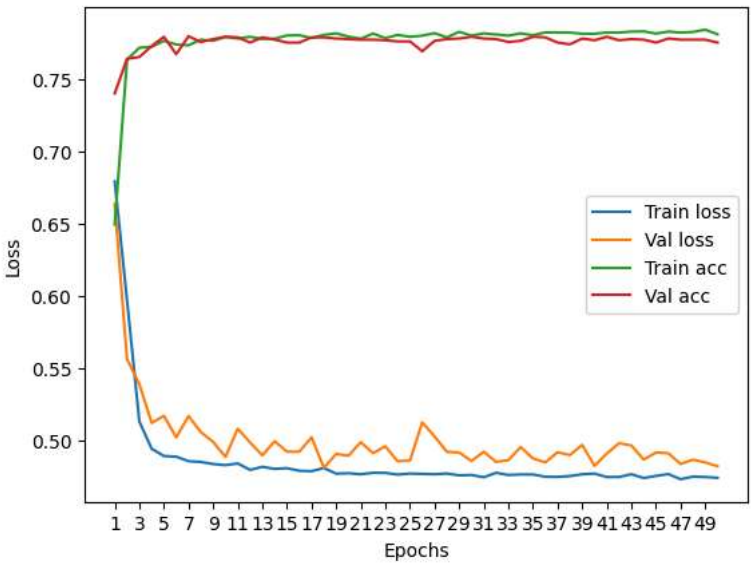
```
GNN(
  (layers): ModuleList(
    (0-1): 2 x GCNConv(8, 8)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=8, bias=True)
    (3): ReLU()
    (4): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
epochs = 50
optimizer = Adam(model.parameters(), 1e-3)
```

```
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

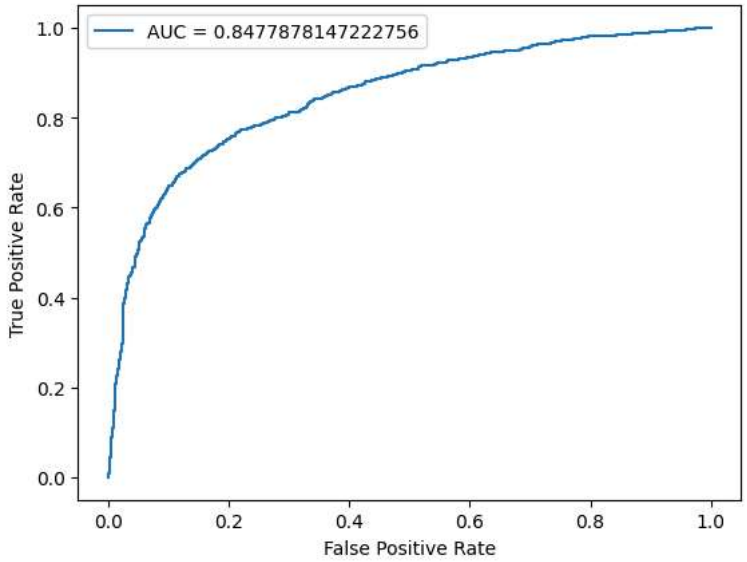
```
118it [00:00, 118.60it/s]
Train Epoch: 0  Loss: 0.679097, Accuracy: 4870/7500 (65%)
Val Epoch: 0    Loss: 0.663196, Accuracy: 1850/2500 (74%)

118it [00:00, 135.56it/s]
Train Epoch: 1  Loss: 0.596697, Accuracy: 5725/7500 (76%)
Val Epoch: 1    Loss: 0.555917, Accuracy: 1910/2500 (76%)

118it [00:00, 120.07it/s]
Train Epoch: 2  Loss: 0.512808, Accuracy: 5787/7500 (77%)
Val Epoch: 2    Loss: 0.538740, Accuracy: 1913/2500 (77%)

118it [00:01, 87.66it/s]
Train Epoch: 3  Loss: 0.493954, Accuracy: 5792/7500 (77%)
Val Epoch: 3    Loss: 0.511825, Accuracy: 1932/2500 (77%)

118it [00:01, 111.67it/s]
Train Epoch: 4  Loss: 0.488916, Accuracy: 5823/7500 (78%)
Val Epoch: 4    Loss: 0.516780, Accuracy: 1948/2500 (78%)

118it [00:01, 117.50it/s]
Train Epoch: 5  Loss: 0.488470, Accuracy: 5804/7500 (77%)
Val Epoch: 5    Loss: 0.501769, Accuracy: 1918/2500 (77%)

118it [00:00, 131.12it/s]
Train Epoch: 6  Loss: 0.485341, Accuracy: 5800/7500 (77%)
Val Epoch: 6    Loss: 0.516669, Accuracy: 1949/2500 (78%)

118it [00:00, 131.15it/s]
Train Epoch: 7  Loss: 0.484812, Accuracy: 5829/7500 (78%)
Val Epoch: 7    Loss: 0.505462, Accuracy: 1939/2500 (78%)

118it [00:00, 126.44it/s]
Train Epoch: 8  Loss: 0.483365, Accuracy: 5823/7500 (78%)
Val Epoch: 8    Loss: 0.498666, Accuracy: 1944/2500 (78%)

118it [00:00, 130.59it/s]
Train Epoch: 9  Loss: 0.482615, Accuracy: 5844/7500 (78%)
Val Epoch: 9    Loss: 0.488309, Accuracy: 1948/2500 (78%)

118it [00:00, 140.08it/s]
Train Epoch: 10         Loss: 0.483770, Accuracy: 5835/7500 (78%)
Val Epoch: 10   Loss: 0.507802, Accuracy: 1947/2500 (78%)

118it [00:00, 119.97it/s]
Train Epoch: 11         Loss: 0.479336, Accuracy: 5844/7500 (78%)
Val Epoch: 11   Loss: 0.498339, Accuracy: 1938/2500 (78%)

118it [00:00, 130.46it/s]
Train Epoch: 12         Loss: 0.481416, Accuracy: 5833/7500 (78%)
Val Epoch: 12   Loss: 0.489419, Accuracy: 1947/2500 (78%)

118it [00:01, 89.80it/s]
Train Epoch: 13         Loss: 0.480006, Accuracy: 5834/7500 (78%)
Val Epoch: 13   Loss: 0.499157, Accuracy: 1943/2500 (78%)

118it [00:01, 112.57it/s]
Train Epoch: 14         Loss: 0.480496, Accuracy: 5851/7500 (78%)
```

```
plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.56
```



With 3 particles _ edge_weights

```
hidden_dims = [8]
model = GNN(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

```
GNN(
  (layers): ModuleList(
    (0): GCNConv(8, 8)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=8, bias=True)
    (3): ReLU()
    (4): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
epochs = 40
optimizer = Adam(model.parameters(), 1e-2)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

```
118it [00:00, 152.07it/s]
Train Epoch: 0  Loss: 0.561166, Accuracy: 5333/7500 (71%)
Val Epoch: 0    Loss: 0.488534, Accuracy: 1945/2500 (78%)

118it [00:00, 138.52it/s]
Train Epoch: 1  Loss: 0.489384, Accuracy: 5800/7500 (77%)
Val Epoch: 1    Loss: 0.504438, Accuracy: 1932/2500 (77%)

118it [00:00, 154.10it/s]
Train Epoch: 2  Loss: 0.486767, Accuracy: 5819/7500 (78%)
Val Epoch: 2    Loss: 0.494417, Accuracy: 1929/2500 (77%)

118it [00:01, 98.58it/s]
Train Epoch: 3  Loss: 0.484695, Accuracy: 5802/7500 (77%)
Val Epoch: 3    Loss: 0.487004, Accuracy: 1934/2500 (77%)

118it [00:01, 106.15it/s]
Train Epoch: 4  Loss: 0.483865, Accuracy: 5832/7500 (78%)
Val Epoch: 4    Loss: 0.490236, Accuracy: 1941/2500 (78%)

118it [00:00, 147.87it/s]
Train Epoch: 5  Loss: 0.483229, Accuracy: 5842/7500 (78%)
Val Epoch: 5    Loss: 0.507822, Accuracy: 1905/2500 (76%)

118it [00:00, 151.55it/s]
Train Epoch: 6  Loss: 0.481793, Accuracy: 5854/7500 (78%)
Val Epoch: 6    Loss: 0.494572, Accuracy: 1950/2500 (78%)

118it [00:00, 155.69it/s]
Train Epoch: 7  Loss: 0.480552, Accuracy: 5824/7500 (78%)
Val Epoch: 7    Loss: 0.493094, Accuracy: 1940/2500 (78%)

118it [00:00, 149.49it/s]
Train Epoch: 8  Loss: 0.481374, Accuracy: 5852/7500 (78%)
Val Epoch: 8    Loss: 0.498773, Accuracy: 1924/2500 (77%)

118it [00:00, 154.57it/s]
Train Epoch: 9  Loss: 0.483028, Accuracy: 5829/7500 (78%)
Val Epoch: 9    Loss: 0.486595, Accuracy: 1943/2500 (78%)

118it [00:00, 160.69it/s]
Train Epoch: 10         Loss: 0.477720, Accuracy: 5854/7500 (78%)
Val Epoch: 10   Loss: 0.492599, Accuracy: 1953/2500 (78%)

118it [00:00, 150.25it/s]
Train Epoch: 11         Loss: 0.480358, Accuracy: 5842/7500 (78%)
Val Epoch: 11   Loss: 0.482977, Accuracy: 1945/2500 (78%)

118it [00:00, 150.38it/s]
Train Epoch: 12         Loss: 0.478480, Accuracy: 5865/7500 (78%)
Val Epoch: 12   Loss: 0.483049, Accuracy: 1944/2500 (78%)

118it [00:00, 153.49it/s]
Train Epoch: 13         Loss: 0.478600, Accuracy: 5859/7500 (78%)
Val Epoch: 13   Loss: 0.490705, Accuracy: 1934/2500 (77%)

118it [00:00, 144.26it/s]
Train Epoch: 14         Loss: 0.478814, Accuracy: 5855/7500 (78%)
```

With 3 particles

```
hidden_dims = [8]
model = GNN(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

```
GNN(
  (layers): ModuleList(
    (0): GCNConv(8, 8)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=8, bias=True)
    (3): ReLU()
    (4): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

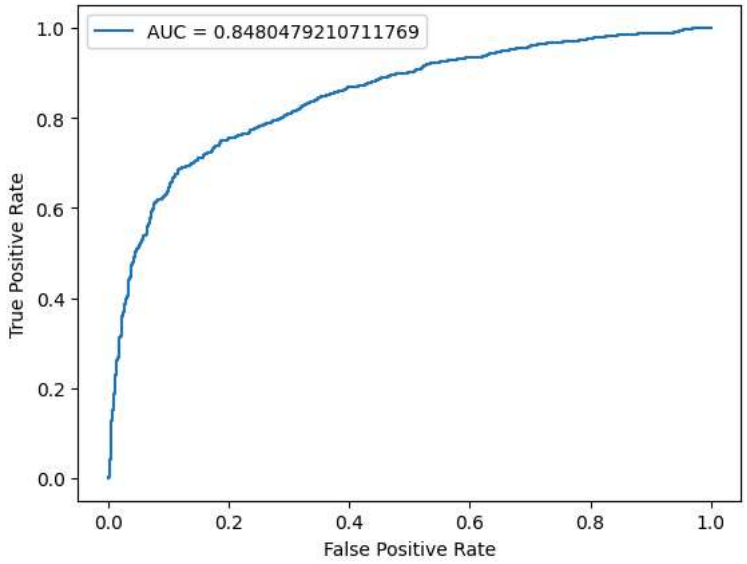
```
epochs = 40
optimizer = Adam(model.parameters(), 1e-2)
```

```
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

```
118it [00:00, 144.72it/s]
Train Epoch: 0  Loss: 0.564105, Accuracy: 5450/7500 (73%)
Val Epoch: 0    Loss: 0.501897, Accuracy: 1925/2500 (77%)

118it [00:00, 154.81it/s]
Train Epoch: 1  Loss: 0.488391, Accuracy: 5806/7500 (77%)
Val Epoch: 1    Loss: 0.483679, Accuracy: 1923/2500 (77%)

118it [00:00, 149.33it/s]
Train Epoch: 2  Loss: 0.489041, Accuracy: 5802/7500 (77%)
Val Epoch: 2    Loss: 0.487830, Accuracy: 1927/2500 (77%)

118it [00:00, 148.32it/s]
Train Epoch: 3  Loss: 0.484678, Accuracy: 5828/7500 (78%)
Val Epoch: 3    Loss: 0.498875, Accuracy: 1951/2500 (78%)

118it [00:00, 141.75it/s]
Train Epoch: 4  Loss: 0.485214, Accuracy: 5838/7500 (78%)
Val Epoch: 4    Loss: 0.492653, Accuracy: 1926/2500 (77%)

118it [00:00, 135.09it/s]
Train Epoch: 5  Loss: 0.484175, Accuracy: 5811/7500 (77%)
Val Epoch: 5    Loss: 0.492041, Accuracy: 1937/2500 (77%)

118it [00:00, 142.71it/s]
Train Epoch: 6  Loss: 0.484268, Accuracy: 5834/7500 (78%)
Val Epoch: 6    Loss: 0.501211, Accuracy: 1938/2500 (78%)

118it [00:00, 143.48it/s]
Train Epoch: 7  Loss: 0.484967, Accuracy: 5826/7500 (78%)
Val Epoch: 7    Loss: 0.481098, Accuracy: 1942/2500 (78%)

118it [00:00, 148.69it/s]
Train Epoch: 8  Loss: 0.481687, Accuracy: 5831/7500 (78%)
Val Epoch: 8    Loss: 0.498758, Accuracy: 1941/2500 (78%)

118it [00:01, 99.64it/s]
Train Epoch: 9  Loss: 0.480818, Accuracy: 5850/7500 (78%)
Val Epoch: 9    Loss: 0.515829, Accuracy: 1923/2500 (77%)

118it [00:01, 107.85it/s]
Train Epoch: 10         Loss: 0.479272, Accuracy: 5855/7500 (78%)
Val Epoch: 10   Loss: 0.488042, Accuracy: 1943/2500 (78%)

118it [00:00, 150.20it/s]
Train Epoch: 11         Loss: 0.478259, Accuracy: 5860/7500 (78%)
Val Epoch: 11   Loss: 0.513751, Accuracy: 1923/2500 (77%)

118it [00:00, 149.09it/s]
Train Epoch: 12         Loss: 0.479499, Accuracy: 5842/7500 (78%)
Val Epoch: 12   Loss: 0.493281, Accuracy: 1922/2500 (77%)

118it [00:00, 152.72it/s]
Train Epoch: 13         Loss: 0.484882, Accuracy: 5855/7500 (78%)
Val Epoch: 13   Loss: 0.491427, Accuracy: 1946/2500 (78%)

118it [00:00, 152.32it/s]
Train Epoch: 14         Loss: 0.478649, Accuracy: 5850/7500 (78%)
```

```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.75999999999999
```



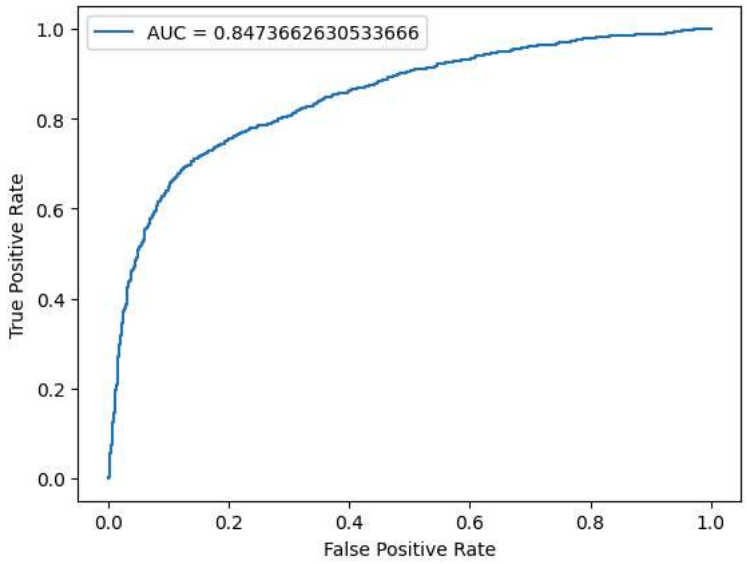

```
epochs = 40
optimizer = Adam(model.parameters(), 1e-2)
lossFn = BCEWithLogitsLoss()
history = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

```
118it [00:00, 135.60it/s]
Train Epoch: 0  Loss: 0.474103, Accuracy: 5884/7500 (78%)
Val Epoch: 0    Loss: 0.501149, Accuracy: 1946/2500 (78%)

118it [00:00, 149.02it/s]
Train Epoch: 1  Loss: 0.472130, Accuracy: 5880/7500 (78%)
Val Epoch: 1    Loss: 0.502167, Accuracy: 1944/2500 (78%)

118it [00:00, 146.46it/s]
Train Epoch: 2  Loss: 0.471066, Accuracy: 5879/7500 (78%)
Val Epoch: 2    Loss: 0.488936, Accuracy: 1934/2500 (77%)

118it [00:01, 99.54it/s]
Train Epoch: 3  Loss: 0.471310, Accuracy: 5910/7500 (79%)
Val Epoch: 3    Loss: 0.484902, Accuracy: 1928/2500 (77%)

118it [00:01, 105.06it/s]
Train Epoch: 4  Loss: 0.473547, Accuracy: 5904/7500 (79%)
Val Epoch: 4    Loss: 0.483683, Accuracy: 1926/2500 (77%)

118it [00:00, 119.30it/s]
Train Epoch: 5  Loss: 0.469339, Accuracy: 5876/7500 (78%)
Val Epoch: 5    Loss: 0.498772, Accuracy: 1926/2500 (77%)

118it [00:01, 101.21it/s]
Train Epoch: 6  Loss: 0.472043, Accuracy: 5878/7500 (78%)
Val Epoch: 6    Loss: 0.493881, Accuracy: 1941/2500 (78%)

118it [00:00, 142.35it/s]
Train Epoch: 7  Loss: 0.469617, Accuracy: 5888/7500 (79%)
Val Epoch: 7    Loss: 0.500067, Accuracy: 1934/2500 (77%)

118it [00:00, 143.97it/s]
Train Epoch: 8  Loss: 0.469736, Accuracy: 5882/7500 (78%)
Val Epoch: 8    Loss: 0.483934, Accuracy: 1942/2500 (78%)

118it [00:00, 147.38it/s]
Train Epoch: 9  Loss: 0.470885, Accuracy: 5881/7500 (78%)
Val Epoch: 9    Loss: 0.495652, Accuracy: 1936/2500 (77%)

118it [00:00, 148.26it/s]
Train Epoch: 10         Loss: 0.469331, Accuracy: 5884/7500 (78%)
Val Epoch: 10   Loss: 0.485438, Accuracy: 1929/2500 (77%)

118it [00:00, 159.85it/s]
Train Epoch: 11         Loss: 0.470513, Accuracy: 5881/7500 (78%)
Val Epoch: 11   Loss: 0.491252, Accuracy: 1930/2500 (77%)

118it [00:00, 145.63it/s]
Train Epoch: 12         Loss: 0.471068, Accuracy: 5903/7500 (79%)
Val Epoch: 12   Loss: 0.505132, Accuracy: 1928/2500 (77%)
```

```
118it [00:00, 128.67it/s]
Train Epoch: 13         Loss: 0.468544, Accuracy: 5895/7500 (79%)
Val Epoch: 13   Loss: 0.495942, Accuracy: 1925/2500 (77%)

118it [00:01, 105.95it/s]
Train Epoch: 14         Loss: 0.472929, Accuracy: 5880/7500 (78%)
```

```
plot_auc(model, test_dataloader)
```

```
Test accuracy:  75.76
```




```
[p.numel() for p in model.parameters()]
```

```
[8, 64, 208, 16, 128, 8, 8, 1]
```

With 2 particles

```
hidden_dims = [8]
model = GNN(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

```
GNN(
  (layers): ModuleList(
    (0): GCNConv(8, 8)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=8, bias=True)
    (3): ReLU()
    (4): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

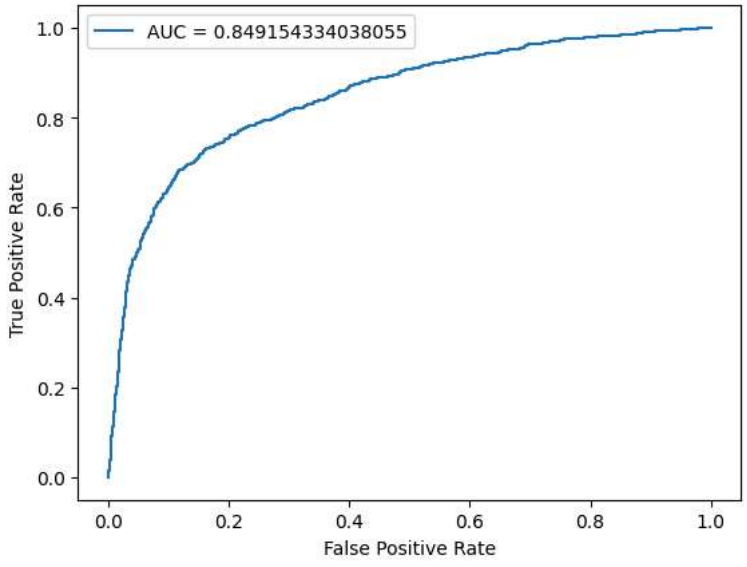
```
epochs = 40
optimizer = Adam(model.parameters(), 1e-2)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

```
118it [00:01, 106.43it/s]
Train Epoch: 0  Loss: 0.557038, Accuracy: 5368/7500 (72%)
Val Epoch: 0    Loss: 0.511172, Accuracy: 1891/2500 (76%)

118it [00:01, 92.21it/s]
Train Epoch: 1  Loss: 0.488654, Accuracy: 5819/7500 (78%)
Val Epoch: 1    Loss: 0.496672, Accuracy: 1932/2500 (77%)

118it [00:00, 135.91it/s]
Train Epoch: 2  Loss: 0.485010, Accuracy: 5812/7500 (77%)
Val Epoch: 2    Loss: 0.499217, Accuracy: 1922/2500 (77%)

118it [00:00, 144.80it/s]
```

```
Train Epoch: 3  Loss: 0.485859, Accuracy: 5838/7500 (78%)
Val Epoch: 3    Loss: 0.496044, Accuracy: 1948/2500 (78%)

118it [00:00, 143.49it/s]
Train Epoch: 4  Loss: 0.483584, Accuracy: 5837/7500 (78%)
Val Epoch: 4    Loss: 0.491622, Accuracy: 1953/2500 (78%)

118it [00:00, 134.80it/s]
Train Epoch: 5  Loss: 0.484336, Accuracy: 5854/7500 (78%)
Val Epoch: 5    Loss: 0.486693, Accuracy: 1942/2500 (78%)

118it [00:00, 144.32it/s]
Train Epoch: 6  Loss: 0.488297, Accuracy: 5803/7500 (77%)
Val Epoch: 6    Loss: 0.508254, Accuracy: 1908/2500 (76%)

118it [00:00, 151.46it/s]
Train Epoch: 7  Loss: 0.486080, Accuracy: 5828/7500 (78%)
Val Epoch: 7    Loss: 0.491006, Accuracy: 1945/2500 (78%)

118it [00:00, 149.43it/s]
Train Epoch: 8  Loss: 0.483610, Accuracy: 5855/7500 (78%)
Val Epoch: 8    Loss: 0.528864, Accuracy: 1896/2500 (76%)

118it [00:00, 148.74it/s]
Train Epoch: 9  Loss: 0.479956, Accuracy: 5850/7500 (78%)
Val Epoch: 9    Loss: 0.482293, Accuracy: 1945/2500 (78%)

118it [00:00, 141.99it/s]
Train Epoch: 10         Loss: 0.481803, Accuracy: 5863/7500 (78%)
Val Epoch: 10   Loss: 0.491594, Accuracy: 1953/2500 (78%)

118it [00:00, 119.07it/s]
Train Epoch: 11         Loss: 0.480290, Accuracy: 5864/7500 (78%)
Val Epoch: 11   Loss: 0.483397, Accuracy: 1941/2500 (78%)

118it [00:01, 97.22it/s]
Train Epoch: 12         Loss: 0.481330, Accuracy: 5860/7500 (78%)
Val Epoch: 12   Loss: 0.484732, Accuracy: 1940/2500 (78%)

118it [00:00, 146.67it/s]
Train Epoch: 13         Loss: 0.480060, Accuracy: 5835/7500 (78%)
Val Epoch: 13   Loss: 0.480474, Accuracy: 1934/2500 (77%)

118it [00:00, 146.21it/s]
Train Epoch: 14         Loss: 0.481068, Accuracy: 5839/7500 (78%)
```

```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  75.84
```



With 1 particle

```
hidden_dims = [8]
model = GNN(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

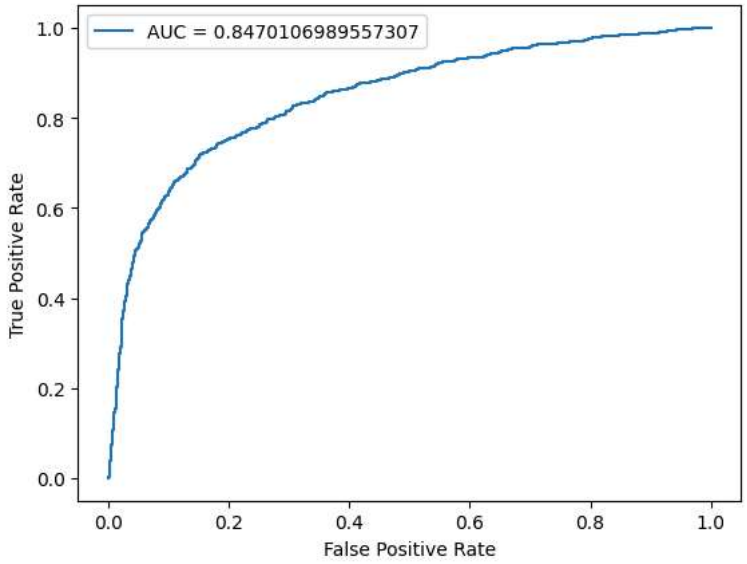
```
GNN(
  (layers): ModuleList(
    (0): GCNConv(8, 8)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=8, bias=True)
    (3): ReLU()
    (4): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
epochs = 40
optimizer = Adam(model.parameters(), 1e-2)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

```
118it [00:00, 144.62it/s]
Train Epoch: 0  Loss: 0.575444, Accuracy: 5280/7500 (70%)
Val Epoch: 0    Loss: 0.499509, Accuracy: 1928/2500 (77%)

118it [00:00, 135.23it/s]
Train Epoch: 1  Loss: 0.488655, Accuracy: 5806/7500 (77%)
Val Epoch: 1    Loss: 0.485575, Accuracy: 1941/2500 (78%)

118it [00:00, 145.35it/s]
Train Epoch: 2  Loss: 0.483837, Accuracy: 5821/7500 (78%)
Val Epoch: 2    Loss: 0.497599, Accuracy: 1947/2500 (78%)

118it [00:00, 148.09it/s]
Train Epoch: 3  Loss: 0.481679, Accuracy: 5824/7500 (78%)
Val Epoch: 3    Loss: 0.507081, Accuracy: 1934/2500 (77%)

118it [00:00, 147.06it/s]
Train Epoch: 4  Loss: 0.484168, Accuracy: 5819/7500 (78%)
Val Epoch: 4    Loss: 0.507094, Accuracy: 1913/2500 (77%)

118it [00:00, 142.57it/s]
Train Epoch: 5  Loss: 0.482809, Accuracy: 5825/7500 (78%)
Val Epoch: 5    Loss: 0.479887, Accuracy: 1946/2500 (78%)

118it [00:00, 146.28it/s]
Train Epoch: 6  Loss: 0.479815, Accuracy: 5848/7500 (78%)
Val Epoch: 6    Loss: 0.494627, Accuracy: 1949/2500 (78%)

118it [00:00, 130.49it/s]
Train Epoch: 7  Loss: 0.480975, Accuracy: 5843/7500 (78%)
Val Epoch: 7    Loss: 0.495464, Accuracy: 1927/2500 (77%)

118it [00:01, 102.02it/s]
Train Epoch: 8  Loss: 0.482672, Accuracy: 5831/7500 (78%)
Val Epoch: 8    Loss: 0.521352, Accuracy: 1884/2500 (75%)

118it [00:00, 125.40it/s]
Train Epoch: 9  Loss: 0.484800, Accuracy: 5830/7500 (78%)
Val Epoch: 9    Loss: 0.500427, Accuracy: 1923/2500 (77%)

118it [00:00, 147.48it/s]
Train Epoch: 10         Loss: 0.480686, Accuracy: 5865/7500 (78%)
Val Epoch: 10   Loss: 0.493070, Accuracy: 1922/2500 (77%)

118it [00:00, 134.70it/s]
Train Epoch: 11         Loss: 0.481329, Accuracy: 5831/7500 (78%)
Val Epoch: 11   Loss: 0.499622, Accuracy: 1948/2500 (78%)

118it [00:00, 162.69it/s]
Train Epoch: 12         Loss: 0.478978, Accuracy: 5857/7500 (78%)
Val Epoch: 12   Loss: 0.483586, Accuracy: 1942/2500 (78%)

118it [00:00, 149.75it/s]
Train Epoch: 13         Loss: 0.478562, Accuracy: 5857/7500 (78%)
Val Epoch: 13   Loss: 0.487354, Accuracy: 1935/2500 (77%)

118it [00:00, 148.90it/s]
Train Epoch: 14         Loss: 0.477194, Accuracy: 5856/7500 (78%)
```

```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.75999999999999
```



All particles

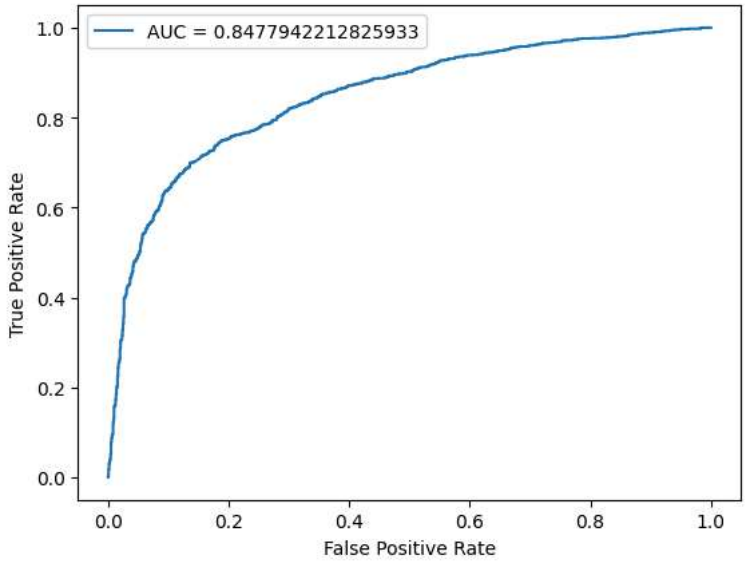
```
epochs = 40
optimizer = Adam(model.parameters(), 1e-2)
lossFn = BCEWithLogitsLoss()
history = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

```
118it [00:00, 147.53it/s]
Train Epoch: 0  Loss: 0.473355, Accuracy: 5880/7500 (78%)
Val Epoch: 0    Loss: 0.497959, Accuracy: 1938/2500 (78%)

118it [00:00, 149.52it/s]
Train Epoch: 1  Loss: 0.473777, Accuracy: 5856/7500 (78%)
Val Epoch: 1    Loss: 0.487524, Accuracy: 1945/2500 (78%)

118it [00:01, 113.85it/s]
Train Epoch: 2  Loss: 0.472921, Accuracy: 5878/7500 (78%)
Val Epoch: 2    Loss: 0.500302, Accuracy: 1941/2500 (78%)

118it [00:01, 93.80it/s]
Train Epoch: 3  Loss: 0.471411, Accuracy: 5891/7500 (79%)
Val Epoch: 3    Loss: 0.492999, Accuracy: 1937/2500 (77%)

118it [00:00, 154.52it/s]
Train Epoch: 4  Loss: 0.473263, Accuracy: 5886/7500 (78%)
Val Epoch: 4    Loss: 0.487558, Accuracy: 1934/2500 (77%)

118it [00:00, 145.88it/s]
Train Epoch: 5  Loss: 0.470998, Accuracy: 5882/7500 (78%)
Val Epoch: 5    Loss: 0.493592, Accuracy: 1938/2500 (78%)

118it [00:00, 146.67it/s]
Train Epoch: 6  Loss: 0.471490, Accuracy: 5900/7500 (79%)
Val Epoch: 6    Loss: 0.485276, Accuracy: 1945/2500 (78%)

118it [00:00, 147.28it/s]
Train Epoch: 7  Loss: 0.472759, Accuracy: 5877/7500 (78%)
Val Epoch: 7    Loss: 0.481972, Accuracy: 1942/2500 (78%)

118it [00:00, 144.74it/s]
Train Epoch: 8  Loss: 0.471838, Accuracy: 5883/7500 (78%)
Val Epoch: 8    Loss: 0.495335, Accuracy: 1929/2500 (77%)

118it [00:00, 146.97it/s]
Train Epoch: 9  Loss: 0.474056, Accuracy: 5885/7500 (78%)
Val Epoch: 9    Loss: 0.497144, Accuracy: 1920/2500 (77%)

118it [00:00, 142.84it/s]
Train Epoch: 10         Loss: 0.471548, Accuracy: 5888/7500 (79%)
Val Epoch: 10   Loss: 0.484024, Accuracy: 1934/2500 (77%)

118it [00:00, 146.31it/s]
Train Epoch: 11         Loss: 0.470305, Accuracy: 5911/7500 (79%)
Val Epoch: 11   Loss: 0.493880, Accuracy: 1949/2500 (78%)
```

```
118it [00:00, 149.51it/s]
Train Epoch: 12         Loss: 0.470776, Accuracy: 5886/7500 (78%)
Val Epoch: 12   Loss: 0.494255, Accuracy: 1946/2500 (78%)

118it [00:00, 152.11it/s]
Train Epoch: 13         Loss: 0.472771, Accuracy: 5870/7500 (78%)
Val Epoch: 13   Loss: 0.485146, Accuracy: 1939/2500 (78%)

118it [00:01, 108.30it/s]
Train Epoch: 14         Loss: 0.469991, Accuracy: 5901/7500 (79%)
```

```
plot_auc(model, test_dataloader)
```

```
Test accuracy:  77.52
```




```
hidden_dims = [10,16,8]


model = GNN(input_dims,hidden_dims,output_dims, activ_fn=ReLU())
model
```

```
GNN(
  (layers): ModuleList(
    (0): GCNConv(8, 10)
    (1): GCNConv(10, 16)
    (2): GCNConv(16, 8)
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=16, bias=True)
    (1): ReLU()
    (2): Linear(in_features=16, out_features=8, bias=True)
    (3): ReLU()
    (4): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

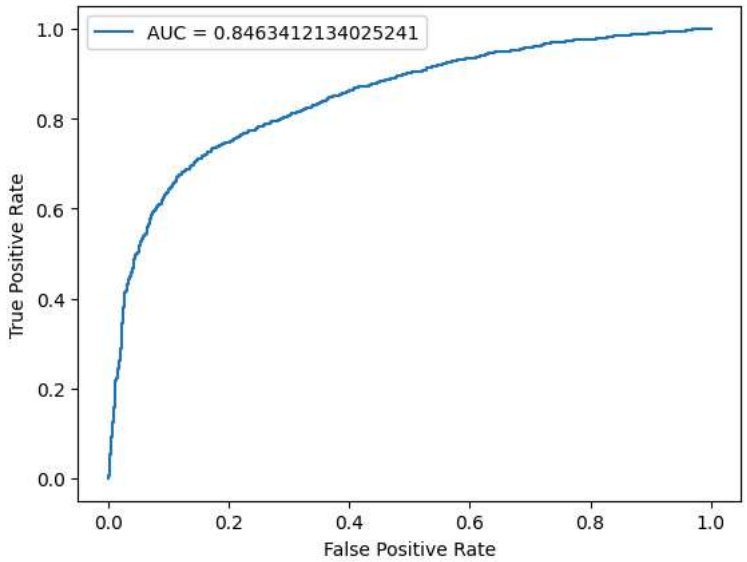
```
optimizer = Adam(model.parameters(), 1e-3)
lossFn = BCEWithLogitsLoss()
history = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

```
118it [00:01, 104.31it/s]
Train Epoch: 0  Loss: 0.693192, Accuracy: 4417/7500 (59%)
Val Epoch: 0    Loss: 0.693324, Accuracy: 1891/2500 (76%)

118it [00:01, 113.69it/s]
Train Epoch: 1  Loss: 0.622892, Accuracy: 5673/7500 (76%)
Val Epoch: 1    Loss: 0.536389, Accuracy: 1944/2500 (78%)

118it [00:01, 109.19it/s]
Train Epoch: 2  Loss: 0.499540, Accuracy: 5764/7500 (77%)
Val Epoch: 2    Loss: 0.505514, Accuracy: 1942/2500 (78%)

118it [00:01, 112.18it/s]
Train Epoch: 3  Loss: 0.489579, Accuracy: 5800/7500 (77%)
Val Epoch: 3    Loss: 0.499787, Accuracy: 1940/2500 (78%)
```

```
118it [00:01, 109.77it/s]
Train Epoch: 4  Loss: 0.488051, Accuracy: 5801/7500 (77%)
Val Epoch: 4    Loss: 0.506792, Accuracy: 1939/2500 (78%)

118it [00:01, 110.60it/s]
Train Epoch: 5  Loss: 0.484599, Accuracy: 5811/7500 (77%)
Val Epoch: 5    Loss: 0.503477, Accuracy: 1934/2500 (77%)

118it [00:01, 105.71it/s]
Train Epoch: 6  Loss: 0.486789, Accuracy: 5824/7500 (78%)
Val Epoch: 6    Loss: 0.495274, Accuracy: 1939/2500 (78%)

118it [00:01, 76.11it/s]
Train Epoch: 7  Loss: 0.482794, Accuracy: 5816/7500 (78%)
Val Epoch: 7    Loss: 0.487679, Accuracy: 1924/2500 (77%)

118it [00:01, 110.45it/s]
Train Epoch: 8  Loss: 0.482740, Accuracy: 5804/7500 (77%)
Val Epoch: 8    Loss: 0.489359, Accuracy: 1945/2500 (78%)

118it [00:01, 108.97it/s]
Train Epoch: 9  Loss: 0.481010, Accuracy: 5825/7500 (78%)
Val Epoch: 9    Loss: 0.494738, Accuracy: 1938/2500 (78%)

118it [00:01, 107.74it/s]
Train Epoch: 10         Loss: 0.480573, Accuracy: 5823/7500 (78%)
Val Epoch: 10   Loss: 0.484388, Accuracy: 1933/2500 (77%)

118it [00:01, 115.14it/s]
Train Epoch: 11         Loss: 0.481500, Accuracy: 5823/7500 (78%)
Val Epoch: 11   Loss: 0.490350, Accuracy: 1946/2500 (78%)

118it [00:01, 113.65it/s]
Train Epoch: 12         Loss: 0.479834, Accuracy: 5845/7500 (78%)
Val Epoch: 12   Loss: 0.488499, Accuracy: 1939/2500 (78%)

118it [00:01, 116.22it/s]
Train Epoch: 13         Loss: 0.479222, Accuracy: 5830/7500 (78%)
Val Epoch: 13   Loss: 0.490010, Accuracy: 1939/2500 (78%)

118it [00:01, 113.96it/s]
Train Epoch: 14         Loss: 0.481209, Accuracy: 5850/7500 (78%)
```

```
plot_auc(model, test_dataloader)
```

```
Test accuracy:  77.16
```



Classical GAT - Custom Graph Attention Layer

```
class GATConv(MessagePassing):
    def __init__(self, in_channels, out_channels):
        super().__init__(aggr='add')  # "Add" aggregation (Step 5).
#         self.heads = heads
#         seld.concat = concat
        self.lin = Linear(in_channels, out_channels, bias=False)
        self.attn = nn.Sequential(Linear(out_channels*2, 8),
```

```
                                ReLU(),
                                Linear(8,1),
                                LeakyReLU(0.2)
                               )
            total_out_channels = out_channels
            self.bias = nn.Parameter(torch.empty(total_out_channels))
            self.reset_parameters()

        def reset_parameters(self):
            super().reset_parameters()
            self.lin.reset_parameters()

            for layer in self.attn:
                if hasattr(layer, 'reset_parameters'):
                    layer.reset_parameters()

            self.bias.data.zero_()

        def forward(self, x, edge_index, edge_attr):
            # x has shape [N, in_channels]
            # edge_index has shape [2, E]

            # H, C = self.heads, self.out_channels

            x = self.lin(x)    #.view(-1, H, C)

            # Step 1: Add self-loops to the adjacency matrix.
            edge_index, edge_attr = remove_self_loops(edge_index, edge_attr)
            edge_index, edge_attr = add_self_loops(edge_index, edge_attr, num_nodes=x.size(0))

            # alpha = self.edge_update(x_i, x_j, edge_attr=edge_attr)

            # propagate_type: (x: OptPairTensor, alpha: Tensor)
            out = self.propagate(edge_index, x=x)

            # out = out.mean(dim=1)

            # Step 6: Apply a final bias vector.
            out = out + self.bias

            return out


        def message(self, x_i, x_j):
            x_edge = torch.cat((x_i,x_j),dim=1)
            x_edge = self.attn(x_edge)
            return x_edge.view(-1,1) * x_j


    class GAT(nn.Module):

        def __init__(self, input_dims, hidden_dims, output_dims, activ_fn = LeakyReLU(0.2)):

            super().__init__()
            layers = []
            layers.append(GATConv(input_dims, hidden_dims[0]))

            for i in range(len(hidden_dims)-1):
                layers.append(GATConv(hidden_dims[i], hidden_dims[i+1]))

            self.layers = ModuleList(layers)
            self.activ_fn = activ_fn
            self.classifier = Sequential(
                               Linear(hidden_dims[-1]+5, 8),
                               ReLU(),
                               Linear(8,1)
                              )

        def forward(self, x, edge_index, edge_attr, batch, graph_features):
            """
            Defining how tensors are supposed to move through the *dressed* quantum
            net.
            """

            h = x
            for i in range(len(self.layers)):
                h = self.layers[i](h, edge_index, edge_attr)
```

```
            h = self.activ_fn(h)

        h = global_mean_pool(h, batch)   # readout layer to get the embedding for each graph in batch
        h = torch.cat((h, graph_features.float().reshape(-1,5)), dim=1)
        h = self.classifier(h)
        return h
```

Classical GCN - Custom GraphConv Layer

```
class GCNConv(MessagePassing):
    def __init__(self, in_channels, out_channels):
        super().__init__(aggr='add')  # "Add" aggregation (Step 5).
        self.lin = Linear(in_channels, out_channels, bias=False)
        self.bias = nn.Parameter(torch.empty(out_channels))
        self.reset_parameters()

    def reset_parameters(self):
        self.bias.data.zero_()

    def forward(self, x, edge_index, edge_weight):
        # x has shape [N, in_channels]
        # edge_index has shape [2, E]

        # Step 1: Add self-loops to the adjacency matrix.
        edge_index, _ = add_self_loops(edge_index, num_nodes=x.size(0))

        # Step 2: Linearly transform node feature matrix.
        # Apply the quantum circuit to each element of the batch and append to q_out
#         q_out = torch.Tensor(0, self.n_qubits)

#         for xi in x:
#             q_out_elem = self.qc(xi).float().unsqueeze(0)
#             q_out = torch.cat((q_out, q_out_elem))
        out = self.lin(x)

        # Step 3: Compute normalization.
        row, col = edge_index
        deg = degree(col, out.size(0), dtype=out.dtype)
        deg_inv_sqrt = deg.pow(-0.5)
        deg_inv_sqrt[deg_inv_sqrt == float('inf')] = 0
        norm = deg_inv_sqrt[row] * deg_inv_sqrt[col]

        # Step 4-5: Start propagating messages.
        out = self.propagate(edge_index, x=edge_weight*out, norm=norm)

        # Step 6: Apply a final bias vector.
        out = out + self.bias

        return out

    def message(self, x_j, norm):
        # x_j has shape [E, out_channels]

        # Step 4: Normalize node features.
        return norm.view(-1, 1) * x_j


class GCN(nn.Module):

    def __init__(self, input_dims, hidden_dims, output_dims, activ_fn = LeakyReLU(0.2)):

        super().__init__()
        layers = [GCNConv(input_dims, hidden_dims)]
        self.layers = ModuleList(layers)
        self.activ_fn = activ_fn
        self.classifier = nn.Linear(hidden_dims, output_dims)

    def forward(self, x, edge_index, edge_attr, batch):
        """
        Defining how tensors are supposed to move through the *dressed* quantum
        net.
        """

        h = x
        for i in range(len(self.layers)):
```

```
            h = self.layers[i](h, edge_index, edge_attr)
            h = self.activ_fn(h)

        h = global_mean_pool(h, batch)   # readout layer to get the embedding for each graph in batch
        h = self.classifier(h)

        # return the two-dimensional prediction from the postprocessing layer
        return h
```

LeakyReLU

Learning rate = 1e-3

```
epochs = 20


model = GCN(input_dims, hidden_dims, output_dims)
model
```

```
GCN(
  (layers): ModuleList(
    (0-2): 3 x GCNConv()
  )
  (activ_fn): LeakyReLU(negative_slope=0.2)
  (classifier): Linear(in_features=8, out_features=1, bias=True)
)
```

```
class GCN(nn.Module):
    def __init__(self, input_dims, hidden_dims, output_dims, activ_fn=LeakyReLU(0.2)):
        super().__init__()
        layers = [GCNConv(input_dims, hidden_dims[0])] # Use the first element of hidden_dims for the first layer

        for i in range(len(hidden_dims) - 1):
            layers.append(GCNConv(hidden_dims[i], hidden_dims[i + 1]))  # Use consecutive elements of hidden_dims for subsequent layers

        self.layers = ModuleList(layers)
        self.activ_fn = activ_fn
        self.classifier = nn.Linear(hidden_dims[-1], output_dims)  # Use the last element of hidden_dims for the classifier input

    def forward(self, x, edge_index, edge_attr, batch):
        """
        Defining how tensors are supposed to move through the *dressed* quantum
        net.
        """

        h = x
        for i in range(len(self.layers)):
            h = self.layers[i](h, edge_index, edge_attr)
            h = self.activ_fn(h)

        h = global_mean_pool(h, batch)  # readout layer to get the embedding for each graph in batch
        h = self.classifier(h)

        # return the two-dimensional prediction from the postprocessing layer
        return h


[p.numel() for p in model.parameters()]
```

```
[10, 80, 16, 160, 8, 128, 8, 1]
```

```
plot_loss(history)
```

Learning rate = 1e-2

```
model = GCN(input_dims, hidden_dims, output_dims)
model
```

```
GCN(
  (layers): ModuleList(
    (0-2): 3 x GCNConv()
  )
  (activ_fn): LeakyReLU(negative_slope=0.2)
```

```
    (classifier): Linear(in_features=8, out_features=1, bias=True)
  )
```

```
plot_loss(history)
```

ReLU

Learning rate = 1e-3

GAT - 10 particles, 100K jets

```
model = GAT(input_dims, [16,64,32,8], output_dims, activ_fn = ReLU())
model
```

```
GAT(
  (layers): ModuleList(
    (0-3): 4 x GATConv()
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
params = [p.numel() for p in model.parameters()]
sum(params)
```

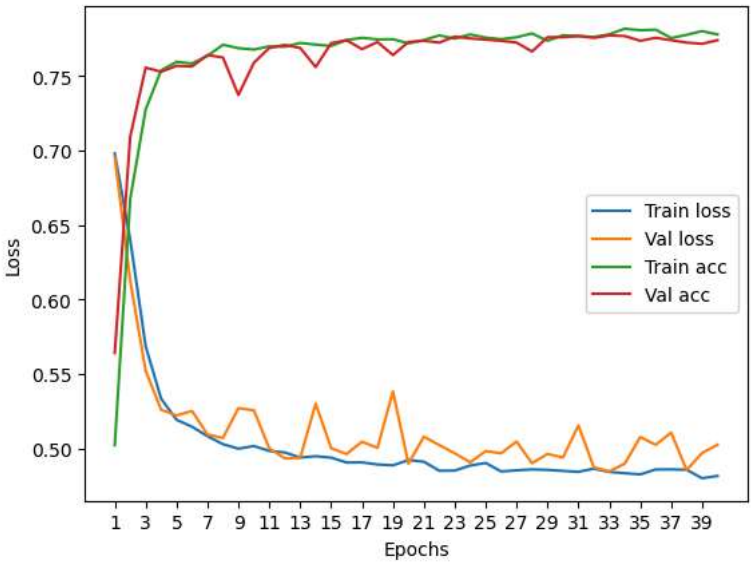
```
5685
```

```
epochs=40
optimizer = Adam(model.parameters(), 1e-3)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

```
118it [00:02, 48.10it/s]
Train Epoch: 0  Loss: 0.697958, Accuracy: 3766/7500 (50%)
Val Epoch: 0    Loss: 0.695898, Accuracy: 1410/2500 (56%)

118it [00:01, 69.94it/s]
Train Epoch: 1  Loss: 0.640130, Accuracy: 5005/7500 (67%)
Val Epoch: 1    Loss: 0.612622, Accuracy: 1773/2500 (71%)

118it [00:01, 78.28it/s]
Train Epoch: 2  Loss: 0.568548, Accuracy: 5457/7500 (73%)
Val Epoch: 2    Loss: 0.552092, Accuracy: 1889/2500 (76%)

118it [00:01, 76.61it/s]
Train Epoch: 3  Loss: 0.533332, Accuracy: 5653/7500 (75%)
Val Epoch: 3    Loss: 0.525924, Accuracy: 1882/2500 (75%)

118it [00:01, 75.22it/s]
Train Epoch: 4  Loss: 0.519162, Accuracy: 5696/7500 (76%)
Val Epoch: 4    Loss: 0.522093, Accuracy: 1892/2500 (76%)

118it [00:01, 72.89it/s]
Train Epoch: 5  Loss: 0.514520, Accuracy: 5688/7500 (76%)
Val Epoch: 5    Loss: 0.525069, Accuracy: 1891/2500 (76%)

118it [00:02, 53.12it/s]
Train Epoch: 6  Loss: 0.508166, Accuracy: 5726/7500 (76%)
Val Epoch: 6    Loss: 0.509338, Accuracy: 1910/2500 (76%)

118it [00:01, 77.40it/s]
Train Epoch: 7  Loss: 0.502801, Accuracy: 5782/7500 (77%)
Val Epoch: 7    Loss: 0.506899, Accuracy: 1906/2500 (76%)

118it [00:01, 71.95it/s]
Train Epoch: 8  Loss: 0.499762, Accuracy: 5765/7500 (77%)
Val Epoch: 8    Loss: 0.526998, Accuracy: 1843/2500 (74%)

118it [00:01, 76.22it/s]
Train Epoch: 9  Loss: 0.501531, Accuracy: 5758/7500 (77%)
Val Epoch: 9    Loss: 0.525524, Accuracy: 1897/2500 (76%)
```

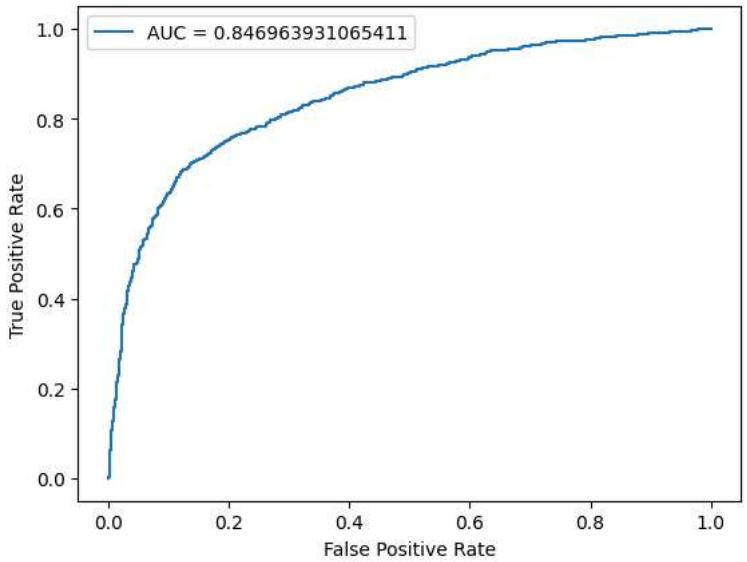
```
118it [00:01, 75.42it/s]
Train Epoch: 10         Loss: 0.498253, Accuracy: 5774/7500 (77%)
Val Epoch: 10   Loss: 0.499937, Accuracy: 1922/2500 (77%)

118it [00:01, 72.37it/s]
Train Epoch: 11         Loss: 0.497236, Accuracy: 5772/7500 (77%)
Val Epoch: 11   Loss: 0.493261, Accuracy: 1927/2500 (77%)

118it [00:02, 51.26it/s]
Train Epoch: 12         Loss: 0.493826, Accuracy: 5791/7500 (77%)
Val Epoch: 12   Loss: 0.493579, Accuracy: 1922/2500 (77%)

118it [00:01, 75.76it/s]
Train Epoch: 13         Loss: 0.494698, Accuracy: 5783/7500 (77%)
Val Epoch: 13   Loss: 0.529994, Accuracy: 1890/2500 (76%)

118it [00:01, 72.20it/s]
Train Epoch: 14         Loss: 0.493747, Accuracy: 5776/7500 (77%)
```

```
plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.56
```



GAT - 5 particles, 100K jets

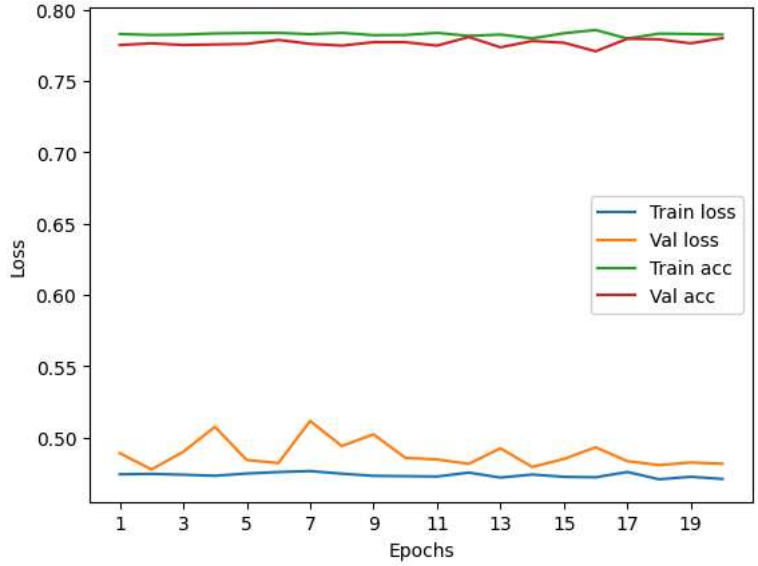
```
model = GAT(input_dims, hidden_dims, output_dims, activ_fn = ReLU())
model
```

```
GAT(
  (layers): ModuleList(
    (0-2): 3 x GATConv()
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
optimizer = Adam(model.parameters(), 1e-3)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

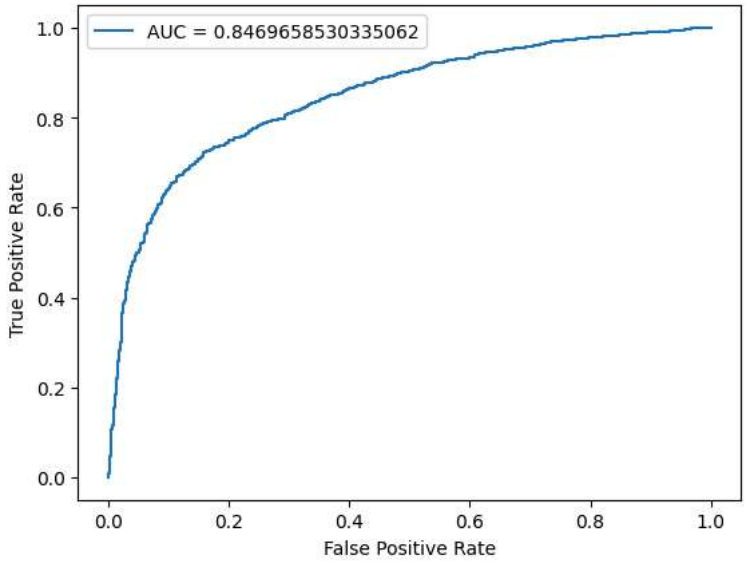
```
118it [00:01, 86.92it/s]
Train Epoch: 0  Loss: 0.474260, Accuracy: 5872/7500 (78%)
Val Epoch: 0    Loss: 0.489037, Accuracy: 1938/2500 (78%)

118it [00:01, 108.28it/s]
Train Epoch: 1  Loss: 0.474543, Accuracy: 5867/7500 (78%)
Val Epoch: 1    Loss: 0.477644, Accuracy: 1941/2500 (78%)

118it [00:01, 106.68it/s]
Train Epoch: 2  Loss: 0.473956, Accuracy: 5869/7500 (78%)
Val Epoch: 2    Loss: 0.489964, Accuracy: 1938/2500 (78%)

118it [00:01, 108.59it/s]
Train Epoch: 3  Loss: 0.473217, Accuracy: 5875/7500 (78%)
Val Epoch: 3    Loss: 0.507557, Accuracy: 1939/2500 (78%)

118it [00:01, 107.72it/s]
Train Epoch: 4  Loss: 0.474795, Accuracy: 5877/7500 (78%)
Val Epoch: 4    Loss: 0.484297, Accuracy: 1940/2500 (78%)

118it [00:01, 110.04it/s]
Train Epoch: 5  Loss: 0.475829, Accuracy: 5878/7500 (78%)
Val Epoch: 5    Loss: 0.482124, Accuracy: 1947/2500 (78%)

118it [00:01, 109.01it/s]
Train Epoch: 6  Loss: 0.476501, Accuracy: 5871/7500 (78%)
Val Epoch: 6    Loss: 0.511585, Accuracy: 1940/2500 (78%)

118it [00:01, 91.08it/s]
Train Epoch: 7  Loss: 0.474687, Accuracy: 5878/7500 (78%)
Val Epoch: 7    Loss: 0.494100, Accuracy: 1937/2500 (77%)

118it [00:01, 73.32it/s]
Train Epoch: 8  Loss: 0.473156, Accuracy: 5866/7500 (78%)
Val Epoch: 8    Loss: 0.502243, Accuracy: 1943/2500 (78%)

118it [00:01, 97.98it/s]
Train Epoch: 9  Loss: 0.472956, Accuracy: 5867/7500 (78%)
Val Epoch: 9    Loss: 0.485800, Accuracy: 1943/2500 (78%)

118it [00:01, 112.82it/s]
Train Epoch: 10         Loss: 0.472686, Accuracy: 5878/7500 (78%)
Val Epoch: 10   Loss: 0.484665, Accuracy: 1937/2500 (77%)

118it [00:01, 110.04it/s]
Train Epoch: 11         Loss: 0.475426, Accuracy: 5862/7500 (78%)
Val Epoch: 11   Loss: 0.481551, Accuracy: 1952/2500 (78%)

118it [00:01, 109.65it/s]
Train Epoch: 12         Loss: 0.471955, Accuracy: 5869/7500 (78%)
Val Epoch: 12   Loss: 0.492479, Accuracy: 1934/2500 (77%)

118it [00:01, 112.16it/s]
Train Epoch: 13         Loss: 0.474089, Accuracy: 5849/7500 (78%)
Val Epoch: 13   Loss: 0.479370, Accuracy: 1945/2500 (78%)

118it [00:01, 112.41it/s]
Train Epoch: 14         Loss: 0.472494, Accuracy: 5876/7500 (78%)
```

```
plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  77.12
```



12500 jets

GAT - 10 particles

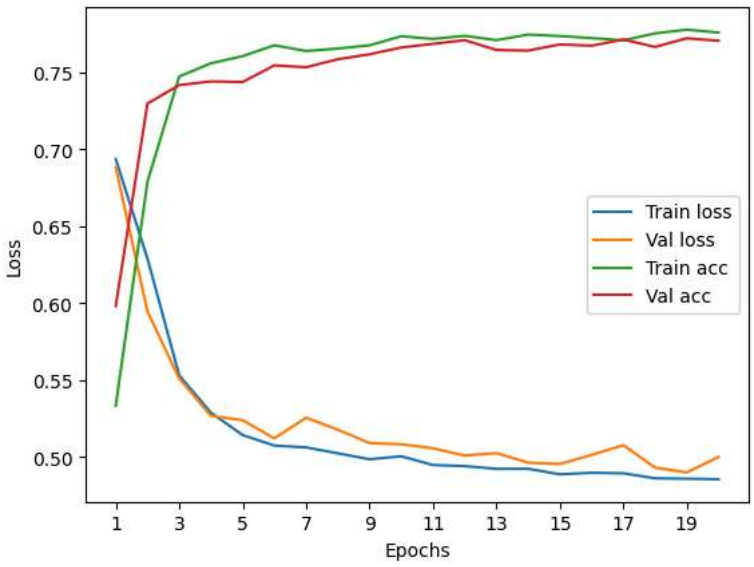
```
model = GAT(input_dims, hidden_dims, output_dims, activ_fn = ReLU())
model
```

```
GAT(
  (layers): ModuleList(
    (0-2): 3 x GATConv()
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
optimizer = Adam(model.parameters(), 1e-3)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

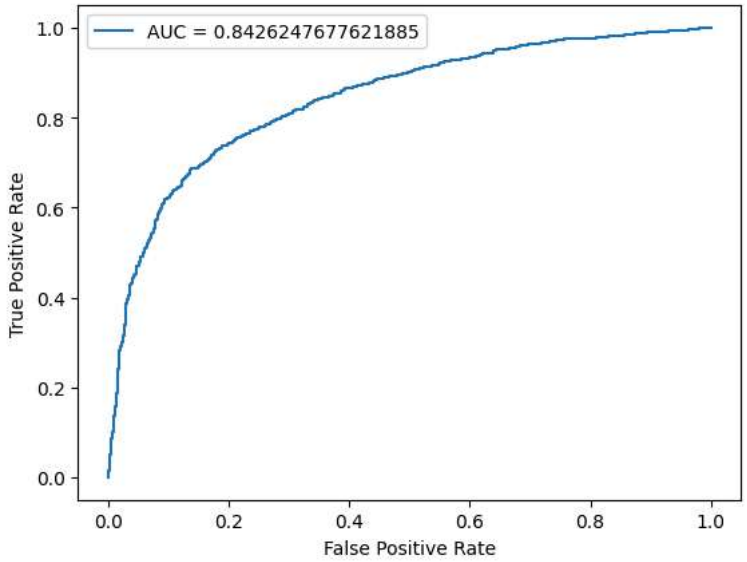
```
118it [00:01, 82.03it/s]
Train Epoch: 0  Loss: 0.693500, Accuracy: 3999/7500 (53%)
Val Epoch: 0    Loss: 0.688200, Accuracy: 1495/2500 (60%)

118it [00:01, 96.06it/s]
Train Epoch: 1  Loss: 0.628886, Accuracy: 5091/7500 (68%)
Val Epoch: 1    Loss: 0.594494, Accuracy: 1824/2500 (73%)

118it [00:01, 94.51it/s]
Train Epoch: 2  Loss: 0.552970, Accuracy: 5604/7500 (75%)
Val Epoch: 2    Loss: 0.550952, Accuracy: 1854/2500 (74%)

118it [00:01, 90.44it/s]
Train Epoch: 3  Loss: 0.528746, Accuracy: 5668/7500 (76%)
Val Epoch: 3    Loss: 0.526657, Accuracy: 1860/2500 (74%)

118it [00:02, 56.26it/s]
Train Epoch: 4  Loss: 0.514176, Accuracy: 5703/7500 (76%)
Val Epoch: 4    Loss: 0.523860, Accuracy: 1859/2500 (74%)

118it [00:01, 90.15it/s]
Train Epoch: 5  Loss: 0.507371, Accuracy: 5756/7500 (77%)
Val Epoch: 5    Loss: 0.512032, Accuracy: 1886/2500 (75%)

118it [00:01, 98.35it/s]
Train Epoch: 6  Loss: 0.506163, Accuracy: 5728/7500 (76%)
Val Epoch: 6    Loss: 0.525462, Accuracy: 1883/2500 (75%)

118it [00:01, 96.90it/s]
Train Epoch: 7  Loss: 0.502314, Accuracy: 5740/7500 (77%)
Val Epoch: 7    Loss: 0.517671, Accuracy: 1896/2500 (76%)

118it [00:01, 97.13it/s]
Train Epoch: 8  Loss: 0.498477, Accuracy: 5756/7500 (77%)
Val Epoch: 8    Loss: 0.508970, Accuracy: 1904/2500 (76%)

118it [00:01, 91.68it/s]
Train Epoch: 9  Loss: 0.500384, Accuracy: 5800/7500 (77%)
Val Epoch: 9    Loss: 0.508206, Accuracy: 1915/2500 (77%)

118it [00:03, 30.66it/s]
Train Epoch: 10         Loss: 0.494714, Accuracy: 5787/7500 (77%)
Val Epoch: 10   Loss: 0.505580, Accuracy: 1921/2500 (77%)

118it [00:01, 63.42it/s]
Train Epoch: 11         Loss: 0.494018, Accuracy: 5802/7500 (77%)
Val Epoch: 11   Loss: 0.500873, Accuracy: 1927/2500 (77%)

118it [00:01, 97.00it/s]
Train Epoch: 12         Loss: 0.492219, Accuracy: 5781/7500 (77%)
Val Epoch: 12   Loss: 0.502452, Accuracy: 1911/2500 (76%)

118it [00:01, 92.60it/s]
Train Epoch: 13         Loss: 0.492257, Accuracy: 5808/7500 (77%)
Val Epoch: 13   Loss: 0.496258, Accuracy: 1910/2500 (76%)

118it [00:01, 97.42it/s]
Train Epoch: 14         Loss: 0.488689, Accuracy: 5801/7500 (77%)
```

```
plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  76.44
```



GAT - 5 particles

```
model = GAT(input_dims, hidden_dims, output_dims, activ_fn = ReLU())
model
```

```
GAT(
  (layers): ModuleList(
    (0-2): 3 x GATConv()
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
a = [p.numel() for p in model.parameters()]
a, sum(a)
```

```
([10,
  80,
  160,
  8,
  8,
```

```
          1,
          16,
          160,
          256,
          8,
          8,
          1,
          8,
          128,
          128,
          8,
          8,
          1,
          104,
          8,
          8,
          1],
         1118)
```

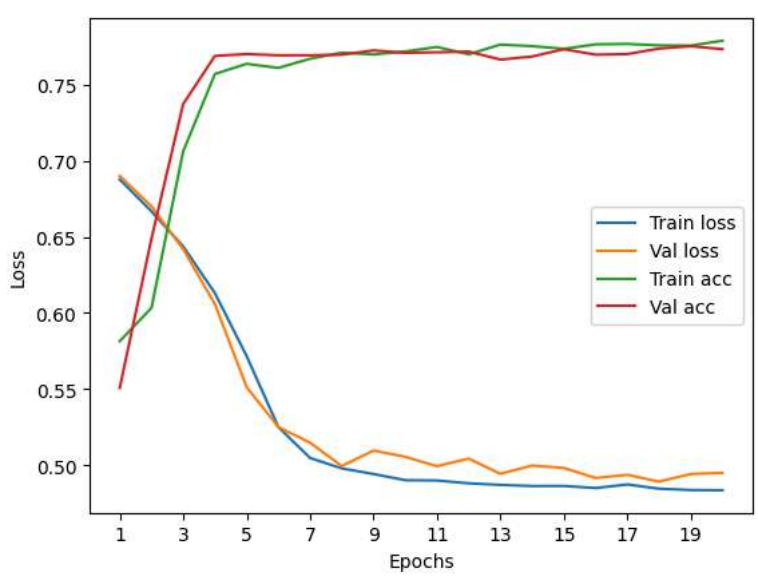
Start coding or generate with AI.

```
optimizer = Adam(model.parameters(), 1e-3)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

```
118it [00:01, 93.73it/s]
Train Epoch: 0  Loss: 0.687646, Accuracy: 4361/7500 (58%)
Val Epoch: 0    Loss: 0.690014, Accuracy: 1377/2500 (55%)

118it [00:01, 92.81it/s]
Train Epoch: 1  Loss: 0.666721, Accuracy: 4525/7500 (60%)
Val Epoch: 1    Loss: 0.670027, Accuracy: 1622/2500 (65%)

118it [00:01, 95.49it/s]
Train Epoch: 2  Loss: 0.643687, Accuracy: 5296/7500 (71%)
Val Epoch: 2    Loss: 0.641844, Accuracy: 1843/2500 (74%)

118it [00:01, 91.77it/s]
Train Epoch: 3  Loss: 0.613032, Accuracy: 5677/7500 (76%)
Val Epoch: 3    Loss: 0.605831, Accuracy: 1922/2500 (77%)

118it [00:01, 80.79it/s]
Train Epoch: 4  Loss: 0.571732, Accuracy: 5727/7500 (76%)
Val Epoch: 4    Loss: 0.551363, Accuracy: 1925/2500 (77%)

118it [00:01, 72.27it/s]
Train Epoch: 5  Loss: 0.525083, Accuracy: 5707/7500 (76%)
Val Epoch: 5    Loss: 0.525116, Accuracy: 1923/2500 (77%)

118it [00:01, 93.88it/s]
Train Epoch: 6  Loss: 0.504787, Accuracy: 5752/7500 (77%)
Val Epoch: 6    Loss: 0.514815, Accuracy: 1923/2500 (77%)

118it [00:01, 95.58it/s]
Train Epoch: 7  Loss: 0.497955, Accuracy: 5782/7500 (77%)
Val Epoch: 7    Loss: 0.499468, Accuracy: 1924/2500 (77%)

118it [00:01, 94.97it/s]
Train Epoch: 8  Loss: 0.494269, Accuracy: 5773/7500 (77%)
Val Epoch: 8    Loss: 0.509684, Accuracy: 1931/2500 (77%)

118it [00:01, 88.45it/s]
Train Epoch: 9  Loss: 0.490210, Accuracy: 5788/7500 (77%)
Val Epoch: 9    Loss: 0.505536, Accuracy: 1927/2500 (77%)

118it [00:01, 96.58it/s]
Train Epoch: 10         Loss: 0.489988, Accuracy: 5810/7500 (77%)
Val Epoch: 10   Loss: 0.499368, Accuracy: 1928/2500 (77%)

118it [00:01, 94.54it/s]
Train Epoch: 11         Loss: 0.488159, Accuracy: 5774/7500 (77%)
Val Epoch: 11   Loss: 0.504358, Accuracy: 1929/2500 (77%)

118it [00:01, 71.76it/s]
Train Epoch: 12         Loss: 0.487130, Accuracy: 5822/7500 (78%)
Val Epoch: 12   Loss: 0.494384, Accuracy: 1916/2500 (77%)

118it [00:01, 79.50it/s]
Train Epoch: 13         Loss: 0.486372, Accuracy: 5814/7500 (78%)
Val Epoch: 13   Loss: 0.499811, Accuracy: 1921/2500 (77%)

118it [00:01, 94.70it/s]
```

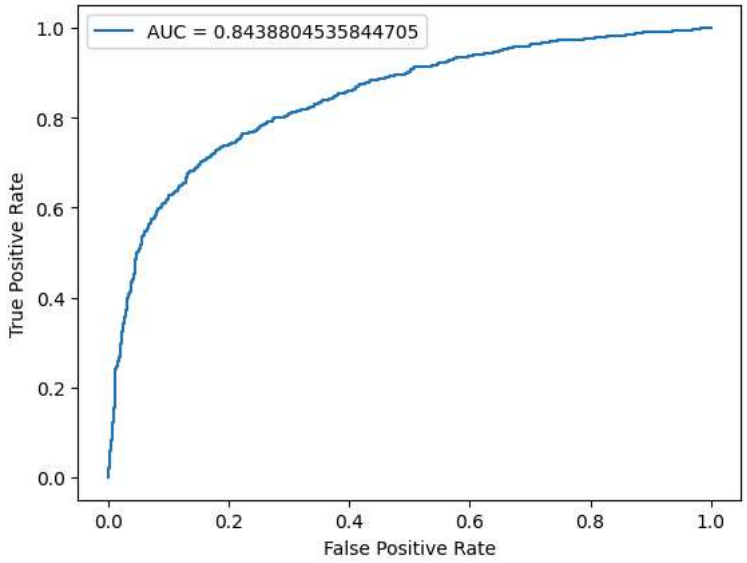
Train Epoch: 14 Loss: 0.486422 Accuracy: 5801/7500 (77%)

```
plot_loss(history)
```




```
plot_auc(model, test_dataloader)
```

```
Test accuracy:  76.68
```



GAT

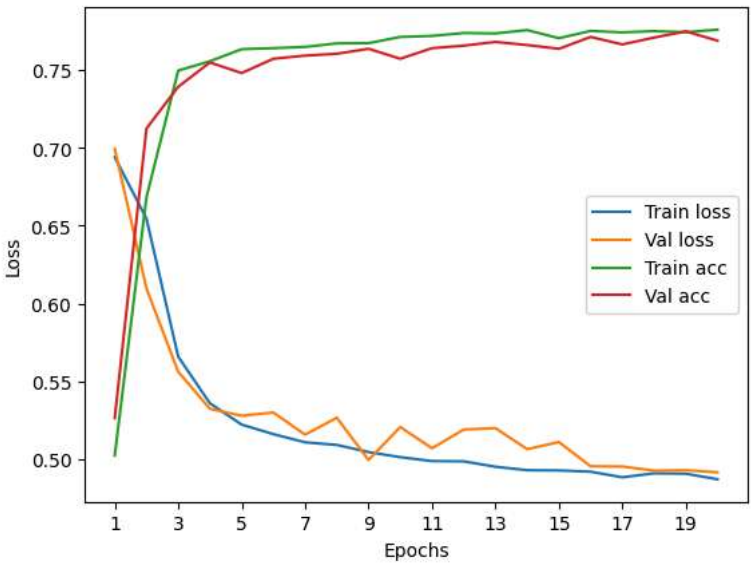
```
model = GAT(input_dims, hidden_dims, output_dims, activ_fn = ReLU())
model
```

```
GAT(
  (layers): ModuleList(
    (0-2): 3 x GATConv()
  )
  (activ_fn): ReLU()
  (classifier): Sequential(
    (0): Linear(in_features=13, out_features=8, bias=True)
    (1): ReLU()
    (2): Linear(in_features=8, out_features=1, bias=True)
  )
)
```

```
optimizer = Adam(model.parameters(), 1e-3)
lossFn = BCEWithLogitsLoss()
history, best_model = train_model(model, optimizer, lossFn, epochs, lr, train_dataloader, val_dataloader)
```

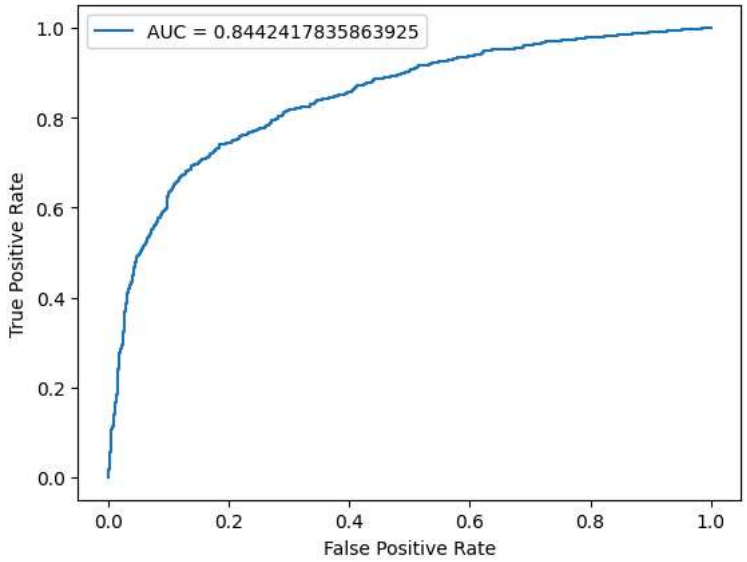
```
118it [00:01, 67.73it/s]
Train Epoch: 0  Loss: 0.693969, Accuracy: 3768/7500 (50%)
Val Epoch: 0    Loss: 0.699346, Accuracy: 1316/2500 (53%)

118it [00:01, 98.39it/s]
Train Epoch: 1  Loss: 0.654511, Accuracy: 5012/7500 (67%)
Val Epoch: 1    Loss: 0.609609, Accuracy: 1781/2500 (71%)

118it [00:01, 95.45it/s]
Train Epoch: 2  Loss: 0.565882, Accuracy: 5622/7500 (75%)
Val Epoch: 2    Loss: 0.556185, Accuracy: 1848/2500 (74%)

118it [00:01, 96.94it/s]
Train Epoch: 3  Loss: 0.535929, Accuracy: 5667/7500 (76%)
Val Epoch: 3    Loss: 0.532354, Accuracy: 1887/2500 (75%)

118it [00:01, 94.47it/s]
Train Epoch: 4  Loss: 0.522236, Accuracy: 5725/7500 (76%)
Val Epoch: 4    Loss: 0.527916, Accuracy: 1870/2500 (75%)

118it [00:01, 98.86it/s]
Train Epoch: 5  Loss: 0.516109, Accuracy: 5730/7500 (76%)
Val Epoch: 5    Loss: 0.530006, Accuracy: 1893/2500 (76%)

118it [00:01, 101.85it/s]
Train Epoch: 6  Loss: 0.510861, Accuracy: 5736/7500 (76%)
Val Epoch: 6    Loss: 0.515743, Accuracy: 1898/2500 (76%)

118it [00:01, 74.68it/s]
Train Epoch: 7  Loss: 0.509171, Accuracy: 5753/7500 (77%)
Val Epoch: 7    Loss: 0.526639, Accuracy: 1901/2500 (76%)

118it [00:01, 73.73it/s]
Train Epoch: 8  Loss: 0.504589, Accuracy: 5754/7500 (77%)
Val Epoch: 8    Loss: 0.499499, Accuracy: 1909/2500 (76%)

118it [00:01, 95.28it/s]
Train Epoch: 9  Loss: 0.501299, Accuracy: 5784/7500 (77%)
Val Epoch: 9    Loss: 0.520682, Accuracy: 1893/2500 (76%)

118it [00:01, 96.70it/s]
Train Epoch: 10         Loss: 0.498817, Accuracy: 5789/7500 (77%)
Val Epoch: 10   Loss: 0.507034, Accuracy: 1910/2500 (76%)

118it [00:01, 98.81it/s]
Train Epoch: 11         Loss: 0.498631, Accuracy: 5803/7500 (77%)
Val Epoch: 11   Loss: 0.519036, Accuracy: 1914/2500 (77%)

118it [00:01, 100.48it/s]
Train Epoch: 12         Loss: 0.495164, Accuracy: 5801/7500 (77%)
Val Epoch: 12   Loss: 0.519954, Accuracy: 1920/2500 (77%)

118it [00:01, 100.75it/s]
Train Epoch: 13         Loss: 0.492957, Accuracy: 5817/7500 (78%)
Val Epoch: 13   Loss: 0.506449, Accuracy: 1915/2500 (77%)

118it [00:01, 95.62it/s]
Train Epoch: 14         Loss: 0.492800, Accuracy: 5778/7500 (77%)
```

```
plot_loss(history)
```




```
plot_auc(best_model, test_dataloader)
```

```
Test accuracy:  76.55999999999999
```

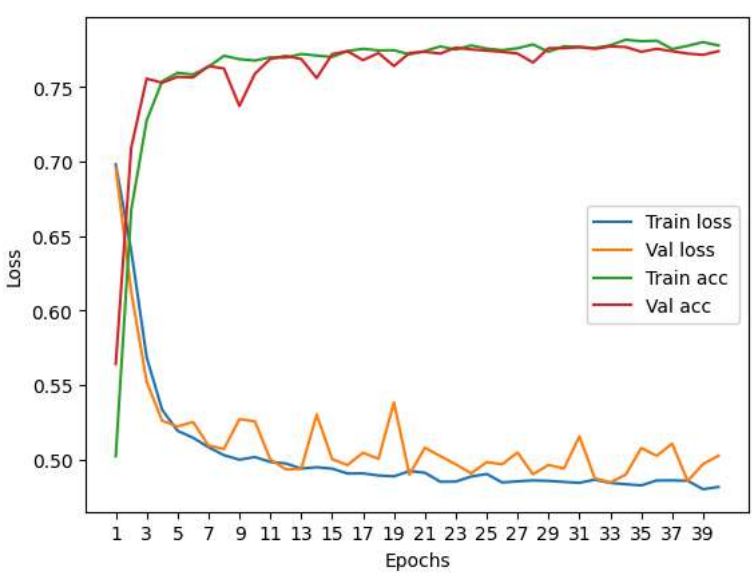


GCN

```
model = GCN(input_dims, hidden_dims, output_dims, activ_fn = ReLU())
model
```

```
GCN(
  (layers): ModuleList(
    (0-2): 3 x GCNConv()
  )
  (activ_fn): ReLU()
  (classifier): Linear(in_features=8, out_features=1, bias=True)
)
```

```
plot_loss(history)
```



Learning rate = 1e-2

```
model = GCN(input_dims, hidden_dims, output_dims, activ_fn = ReLU())
model
```

```
GCN(
  (layers): ModuleList(
    (0-2): 3 x GCNConv()
  )
  (activ_fn): ReLU()
  (classifier): Linear(in_features=8, out_features=1, bias=True)
)
```

```
plot_loss(history)
```

Parameterized quantum circuit - Node Embedding QNN

```
def H_layer(nqubits):
    """Layer of single-qubit Hadamard gates.
    """
    for idx in range(nqubits):
        qml.Hadamard(wires=idx)

def encoder(w):
    """Layer of parametrized qubit rotations around the x axis.
    """
    for idx, element in enumerate(w):
        qml.RY(element, wires=idx)

def Rot_layer(gate, w):
    """Layer of parametrized qubit rotations around the y axis.
    """
    for idx, element in enumerate(w):
        gate(element, wires=idx)


def entangling_layer(nqubits):
    """Layers of CZ and RY gates.
    """
    for i in range(0, nqubits - 1):  # Loop over even indices: i=0,2,...N-2
        qml.CNOT(wires=[i, i + 1])

    qml.CNOT(wires=[nqubits-1, 0])


def quantum_net(n_qubits, q_depth):

    dev = qml.device("default.qubit", wires=n_qubits)
```

```
    @qml.qnode(dev, interface='torch')
    def quantum_circuit(inputs, q_weights_flat):
        """
        The variational quantum circuit.
        """

        # Reshape weights
        q_weights = q_weights_flat.reshape(q_depth, 2, n_qubits)

        # Embed features in the quantum node
        qml.AngleEmbedding(inputs, wires = range(n_qubits), rotation="Y")

        # Sequence of trainable variational layers
        for k in range(q_depth):
            Rot_layer(qml.RY, q_weights[k][0])
            entangling_layer(n_qubits)
            Rot_layer(qml.RZ, q_weights[k][1])

        # Expectation values in the Z basis
        exp_vals = [qml.expval(qml.PauliZ(position)) for position in range(n_qubits)]
        return exp_vals

    return qml.qnn.TorchLayer(quantum_circuit, {"q_weights_flat": (2*q_depth*n_qubits)}), quantum_circuit
```

Visualizing the quantum layer

```
_, qc = quantum_net(8,1)
inputs = torch.randn(8)
params = torch.randn(2*8)
print(qml.draw(qc)(inputs, params))
```

```
 0: ─╭AngleEmbedding(M0)──RY(-0.41)─╭●─────────────────╭X──RZ(0.02)──┤  <Z>
 1: ─├AngleEmbedding(M0)──RY(0.08)──╰X─╭●──────────────│───RZ(0.05)──┤  <Z>
 2: ─├AngleEmbedding(M0)──RY(-0.10)────╰X─╭●───────────│───RZ(-1.24)─┤  <Z>
 3: ─├AngleEmbedding(M0)──RY(-0.42)───────╰X─╭●──────│───RZ(-0.56)─┤  <Z>
 4: ─├AngleEmbedding(M0)──RY(0.25)──────────╰X─╭●────│───RZ(1.17)──┤  <Z>
 5: ─├AngleEmbedding(M0)──RY(-0.57)─────────╰X─╭●───│───RZ(0.92)──┤  <Z>
 6: ─├AngleEmbedding(M0)──RY(0.35)──────────╰X─╭●─╰X──RZ(-0.05)──┤  <Z>
 7: ─╰AngleEmbedding(M0)──RY(0.13)──────────╰X─╰●──RZ(0.47)──┤  <Z>

 M0 =
 tensor([ 1.4687,  0.6915, -1.4321, -0.2672,  1.6790,  1.0876,  2.2087,  0.3476])
```

```
batch_dim = 5
x = torch.zeros((batch_dim, 3))
qlayer, _ = quantum_net(3,1)
qlayer(x).shape
```

```
torch.Size([5, 3])
```

Quantum Classifier

1. Matrix Product State Classifier

```
def MPS(n_qubits):
    dev = qml.device("default.qubit", wires=n_qubits)

    @qml.qnode(dev, interface='torch')
    def quantum_circuit(inputs, q_weights_flat):
        """
        The variational quantum classifier.
        """

        # Reshape weights
        q_weights = q_weights_flat[:-1].reshape(n_qubits-1, 2)

        # Embed features in the quantum node
        qml.AngleEmbedding(inputs, wires = range(n_qubits), rotation="Y")

        # Sequence of trainable variational layers
        for k in range(n_qubits-1):
            qml.RY(q_weights[k][0], wires=k)
```

```
            qml.RY(q_weights[k][1], wires=k+1)
            qml.CZ(wires=[k,k+1])

        qml.RY(q_weights_flat[-1], wires=n_qubits-1)

        # Expectation values in the Z basis
        return [qml.expval(qml.PauliZ(n_qubits - 1))]
        # return [qml.expval(qml.operation.CVObservable(q_weights_flat[-1], qml.PauliZ(n_qubits - 1)))]

    return qml.qnn.TorchLayer(quantum_circuit, {"q_weights_flat": (2*n_qubits - 1)}), quantum_circuit


_, qc = MPS(6)
inputs = torch.randn(6)
params = torch.randn(2*6-1)
print(qml.draw(qc)(inputs, params))
```

```
0: ─╭AngleEmbedding(M0)──RY(-0.17)─╭●──────────────────────────────────────────────────────────────────────────────
1: ─├AngleEmbedding(M0)──RY(1.60)──╰Z──RY(0.07)─╭●──────────────────────────────────────────────────────────────────
2: ─├AngleEmbedding(M0)──RY(-0.87)─────────────╰Z──RY(1.74)─╭●──────────────────────────────────────────────────────
3: ─├AngleEmbedding(M0)──RY(-2.35)──────────────────────────╰Z──RY(-0.35)─╭●──────────────────────────────────────
4: ─├AngleEmbedding(M0)──RY(-1.05)──────────────────────────────────────╰Z──RY(-0.67)─╭●──────────────────────────
5: ─╰AngleEmbedding(M0)──RY(0.69)──────────────────────────────────────────────────╰Z

──────────────┤
──────────────┤
──────────────┤
──────────────┤
──────────────┤
──RY(-0.54)──┤  <Z>

M0 =
tensor([-0.1231,  1.7047,  0.2194, -2.0932, -0.3683, -2.1188])
```

2. Tensor Tree Network Classifier

```
def TTN(n_qubits):

    dev = qml.device("default.qubit", wires=n_qubits)

    @qml.qnode(dev, interface='torch')
    def quantum_circuit(inputs, q_weights_flat):
        """
        The variational quantum classifier.
        """

        # Embed features in the quantum node
        qml.AngleEmbedding(inputs, wires = range(n_qubits), rotation="Y")

        n_layers = int(np.log2(n_qubits))
        i = 0
        n_params = int(2**(np.log2(n_qubits)+1)-2 +1)

        for layer in range(n_layers):
            n_gates = n_qubits//(2**(layer+1))
            for j in range(n_gates):
                qubit0 = j * (n_qubits//(2**(n_layers-layer-1))) + 2**layer - 1
                qubit1 = j * (n_qubits//(2**(n_layers-layer-1))) + 2**(layer+1) - 1
                qml.RY(q_weights_flat[i], wires=qubit0)
                qml.RY(q_weights_flat[i+1], wires=qubit1)
                qml.CZ(wires=[qubit0,qubit1])
                i += 2

        qml.RY(q_weights_flat[-1], wires=n_qubits-1)

        # Expectation values in the Z basis
        return [qml.expval(qml.PauliZ(n_qubits - 1))]

    return qml.qnn.TorchLayer(quantum_circuit, {"q_weights_flat": int(2**(np.log2(n_qubits)+1)-2 +1)}), quantum_circuit


_, qc = TTN(8)
inputs = torch.randn(8)
params = torch.randn(int(2**(np.log2(8)+1)-2 +1))
print(qml.draw(qc)(inputs, params))
```

```
 0: ─╭AngleEmbedding(M0)──RY(1.69)──╭●──────────────────────────────────────╮
 1: ─├AngleEmbedding(M0)──RY(0.82)──╰Z──RY(0.79)──╭●─────────────────────────╮
 2: ─├AngleEmbedding(M0)──RY(-0.27)─╭●────────────╰──────────────────────────╮
 3: ─├AngleEmbedding(M0)──RY(-0.63)─╰Z──RY(-0.55)─╰Z──RY(-0.17)─╭●──────────╮
 4: ─├AngleEmbedding(M0)──RY(0.77)──╭●──────────────────────────╰──────────╮
 5: ─├AngleEmbedding(M0)──RY(-0.61)─╰Z──RY(0.94)──╭●──────────╰──────────╮
 6: ─├AngleEmbedding(M0)──RY(0.64)──╭●────────────╰──────────╰──────────╮
 7: ─╰AngleEmbedding(M0)──RY(0.86)──╰Z──RY(-0.83)─╰Z──RY(-0.94)─╰Z──RY(-0.08)─┤  <Z>

 M0 =
 tensor([-2.0013, -0.8022,  0.4389,  0.2997,  1.3314, -1.2411,  1.4708,  0.2401])
```

Quantum Enhanced GAT

```
class QGATConv(MessagePassing):
    def __init__(self, in_channels, depth, attn_model):
        super().__init__(aggr='add')  # "Add" aggregation (Step 5).
        self.bias = nn.Parameter(torch.empty(in_channels))
        self.reset_parameters()
        self.n_qubits = in_channels
        self.qc, _ = quantum_net(self.n_qubits, depth)

        if attn_model == "MPS":
            self.attn, _ = MPS(in_channels*2)
        else:
            self.attn, _ = TTN(in_channels*2)

        self.readout = Linear(1, 1)
        self.attn = Linear(in_channels*2, 1)


    def reset_parameters(self):
        self.bias.data.zero_()

    def forward(self, x, edge_index, edge_attr):
        # x has shape [N, in_channels]
        # edge_index has shape [2, E]

        # Step 1: Add self-loops to the adjacency matrix.
        edge_index, edge_attr = remove_self_loops(edge_index, edge_attr)
        edge_index, edge_attr = add_self_loops(edge_index, edge_attr, num_nodes=x.size(0))

        # Step 2: Linearly transform node feature matrix

        q_out = self.qc(x).float()

        # Step 3: Compute normalization.
#         row, col = edge_index
#         deg = degree(col, q_out.size(0), dtype=q_out.dtype)
#         deg_inv_sqrt = deg.pow(-0.5)
#         deg_inv_sqrt[deg_inv_sqrt == float('inf')] = 0
#         norm = deg_inv_sqrt[row] * deg_inv_sqrt[col]

        # Step 4-5: Start propagating messages.
        out = self.propagate(edge_index, x=q_out)

        # Step 6: Apply a final bias vector.
        out = out + self.bias

        return out

    def message(self, x_i, x_j):
        # x_j has shape [E, out_channels]

        x_edge = torch.cat((x_i,x_j),dim=1)
        x_edge = self.attn(x_edge)
        x_edge = self.readout(x_edge)
        return x_edge.view(-1, 1) * x_j


class QGAT(nn.Module):

    def __init__(self, input_dims, depth, n_layers, activ_fn = LeakyReLU(0.2), classifier="MPS", attn_model="MPS")

        super().__init__()
        layers = []
```

```
        for i in range(n_layers):
            layers.append(QGATConv(input_dims, depth, attn_model))

        self.layers = ModuleList(layers)
        self.activ_fn = activ_fn
        if classifier == "MPS" :
            self.classifier, _ = MPS(input_dims+5)
        elif classifier == "TTN":
            self.classifier, _ = TTN(input_dims+5)

        self.readout = Linear(1,1)

    def forward(self, x, edge_index, edge_attr, batch, graph_features):
        """
        Defining how tensors are supposed to move through the *dressed* quantum
        net.
        """

        h = x
        for i in range(len(self.layers)):
            h = self.layers[i](h, edge_index, edge_attr)
            h = self.activ_fn(h)

        h = global_mean_pool(h, batch)   # readout layer to get the embedding for each graph in batch
        h = torch.cat((h, graph_features.float().reshape(-1,5)), dim=1)
        h = self.classifier(h)
        h = self.readout(h)
        return h
```

Quantum-enhanced GCN

```
class QGCNConv(MessagePassing):
    def __init__(self, in_channels):
        super().__init__(aggr='add')  # "Add" aggregation (Step 5).
        self.bias = nn.Parameter(torch.empty(in_channels))
        self.reset_parameters()
        self.n_qubits = in_channels
        self.qc, _ = quantum_net(self.n_qubits, q_depth)

    def reset_parameters(self):
        self.bias.data.zero_()

    def forward(self, x, edge_index, edge_weights):
        # x has shape [N, in_channels]
```