Tracking Reconstruction (Internal presentations)

Steve Ataucuri

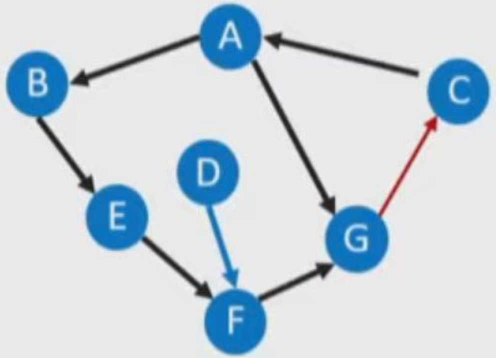
SPRACE

Graph Neural Network

Introduction

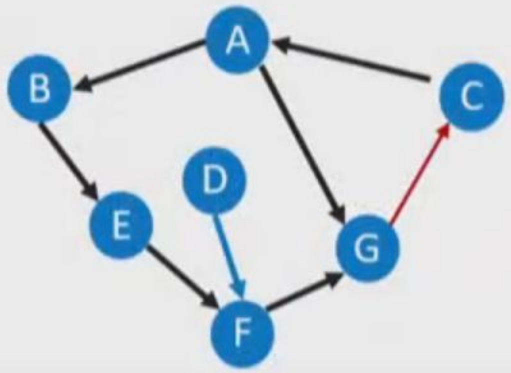
Graph Notation

- Nodes/Vertices
- Edges

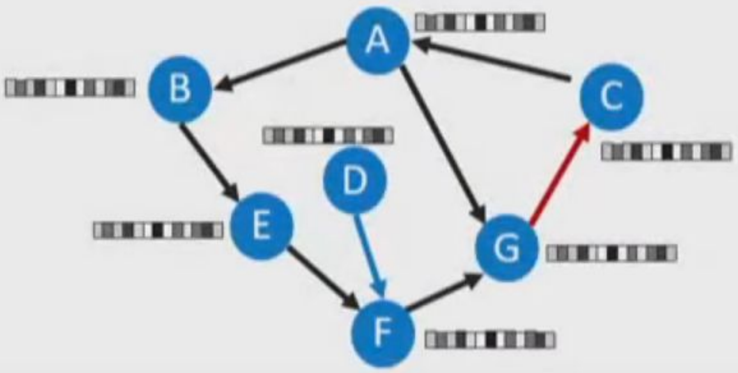


$$G = (V, E)$$

Graph Neural Networks

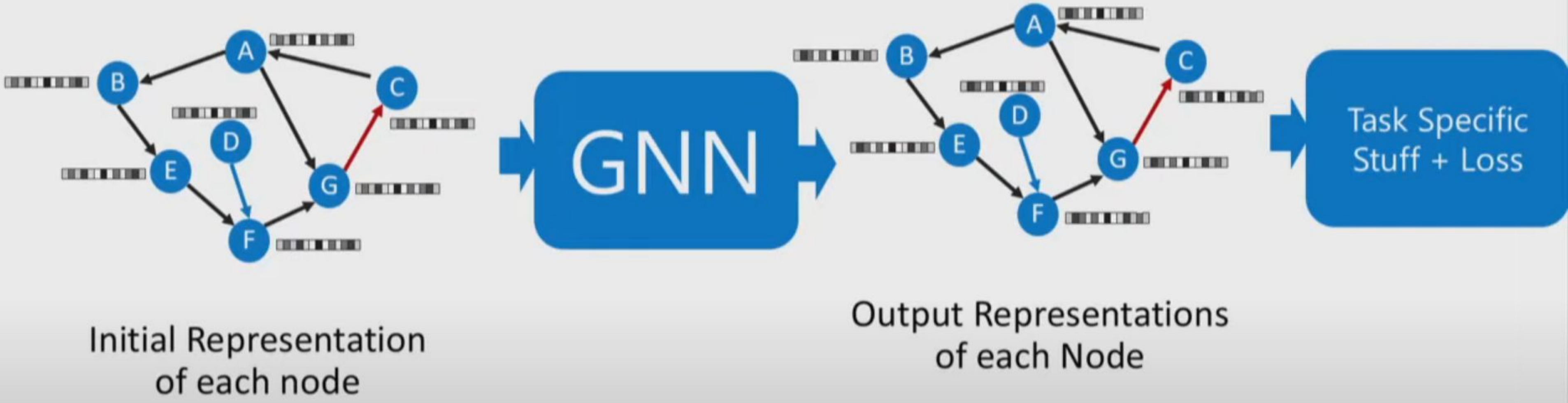


Graph Representation of Problem

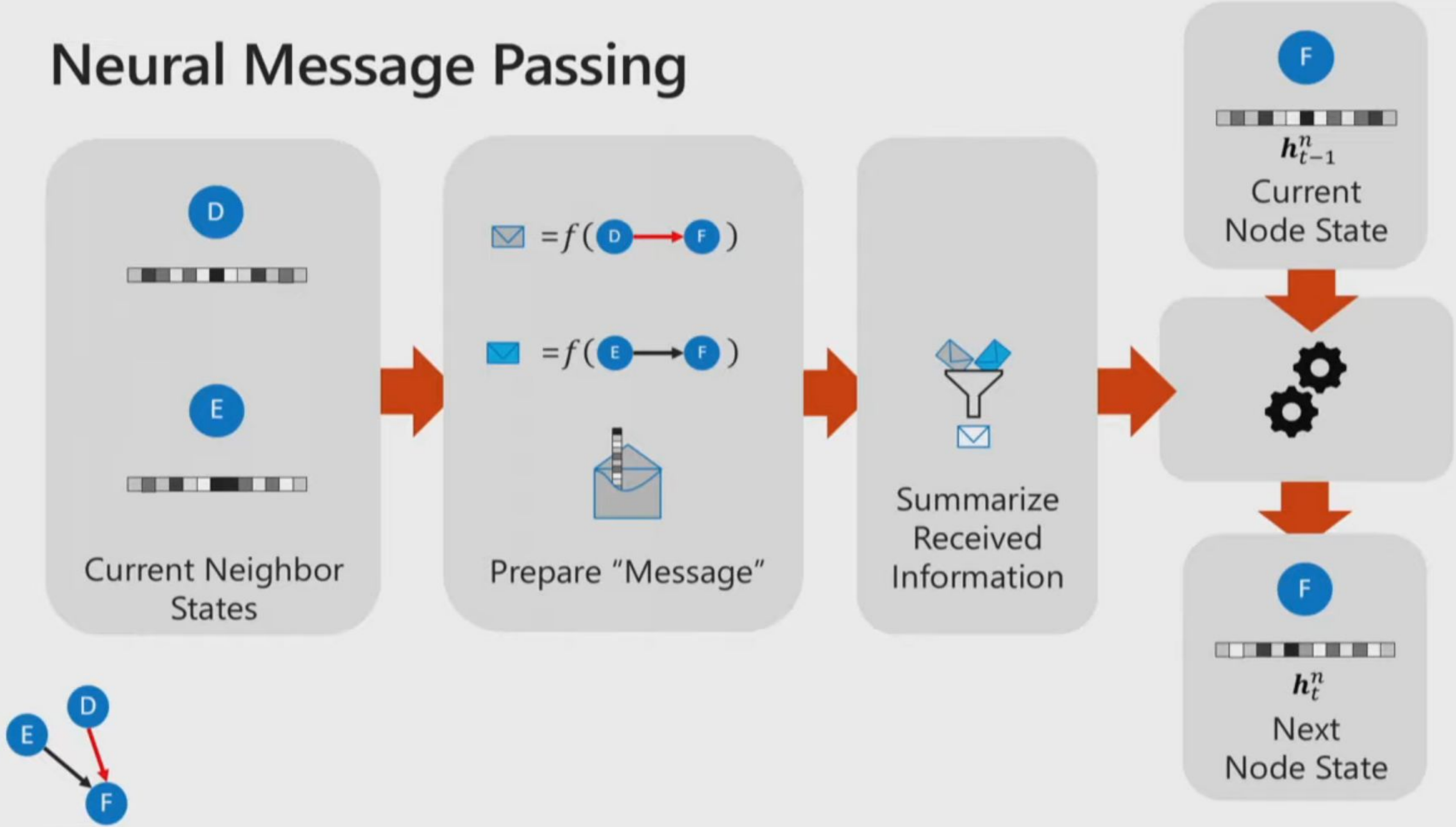


Initial Representation of each node

Graph Neural Networks



Neural Message Passing



Project - HepTrkX

Novel deep learning methods for track reconstruction

Steven Farrell[1,*], *Paolo* Calafiura[1], *Mayur* Mudigonda[1], Prabhat[1], *Dustin* Anderson[2], *Jean-Roch* Vlimant[2], *Stephan* Zheng[2], *Josh* Bendavid[2], *Maria* Spiropulu[2], *Giuseppe* Cerati[3], *Lindsey* Gray[3], *Jim* Kowalkowski[3], *Panagiotis* Spentzouris[3], and *Aristeidis* Tsaris[3]

[1]Lawrence Berkeley National Laboratory
[2]California Institute of Technology
[3]Fermi National Accelerator Laboratory

Abstract. For the past year, the HEP.TrkX project has been investigating machine learning solutions to LHC particle track reconstruction problems. A variety of models were studied that drew inspiration from computer vision applications and operated on an image-like representation of tracking detector data. While these approaches have shown some promise, image-based methods face challenges in scaling up to realistic HL-LHC data due to high dimensionality

- LSTM + Loss Join Distribution
- Graphs Neural Network

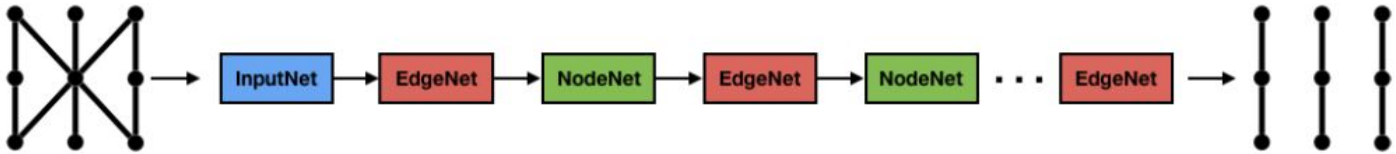


Figure 9. Diagram of the Graph Neural Network model which begins with an input transformation layer and has a number of recurrent iterations of alternating EdgeNetwork and NodeNetwork components. In this case, the final output layer is the EdgeNetwork, making this a segment classifier model.

- An **EdgeNetwork** computes weights for every edge of the graph using the features of the start and end nodes.
- A **NodeNetwork** computes new features for every node using the edge weight aggregated features of the connected nodes on the previous and next detector layers separately as well as the nodes' current features.

Both the EdgeNetwork and NodeNetwork are implemented as Multi-Layer Perceptrons (MLPs) with two layers each and hyperbolic tangent hidden activations.

The full Graph Neural Network model consists of an input transformation layer followed by recurrent alternating applications of the EdgeNetwork and NodeNetwork. The architecture for the segment classification network is illustrated in figure 9. With each iteration of the networks, the model propagates information through the graph, adaptively learning to strengthen important connections and weaken useless or spurious ones.

Proposal

- Binary hit classification :
  - learn to identify one track
  - Classification of nodes
  - 4 hits
  - Final sigmoid activation to predict two nodes belong to target track or not
- Binary segment classification of edges
  - learn to identify many tracks by classification the graph edges(hit pairs)
  - learn to distinguish true hits pairs , hits produced by the same particle

Methodology

- Graph Representation
- G = (V,E)
- G = (V =(r, ϕ, z) , E =$(\Delta\eta, \Delta\phi)$)
- The edges Label are 1 if two hits com from same track else 0
-
- Let’s got to partial results

Results

Approach Graph Neural Networks

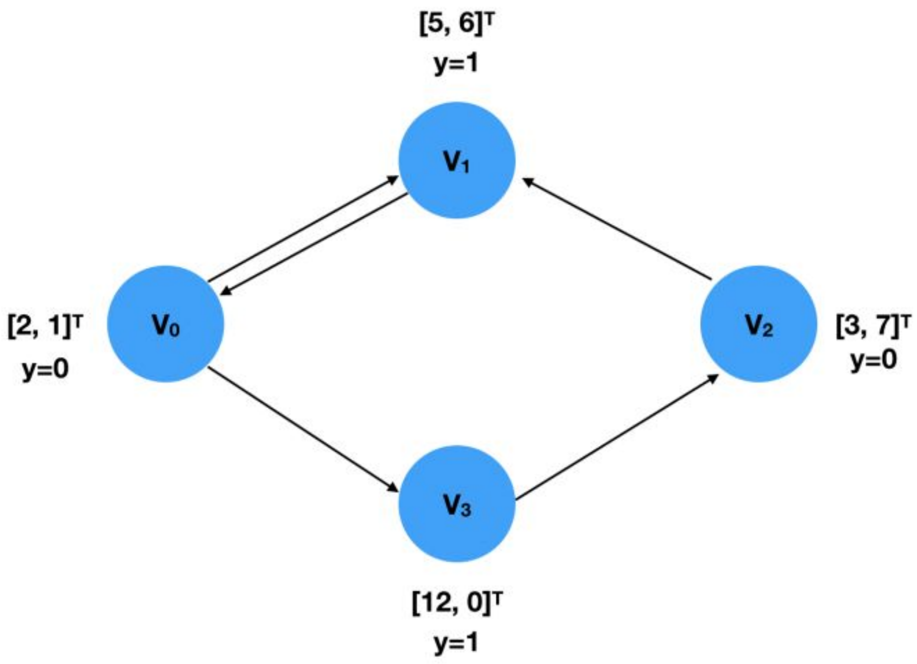
Working In Progress

- Previous Git-hub code Updated
- Added support for MLP with Gaussian
- In progress LSTM(stopped)
- In progress GNN
  - Reply results:
    - https://github.com/exatrkx/exatrkx-ctd2020
    - https://github.com/exatrkx/exatrkx-neurips19/tree/master/gnn-tracking
    - https://github.com/murnanedaniel/heptrkx-gnn-tracking

Working In Progress

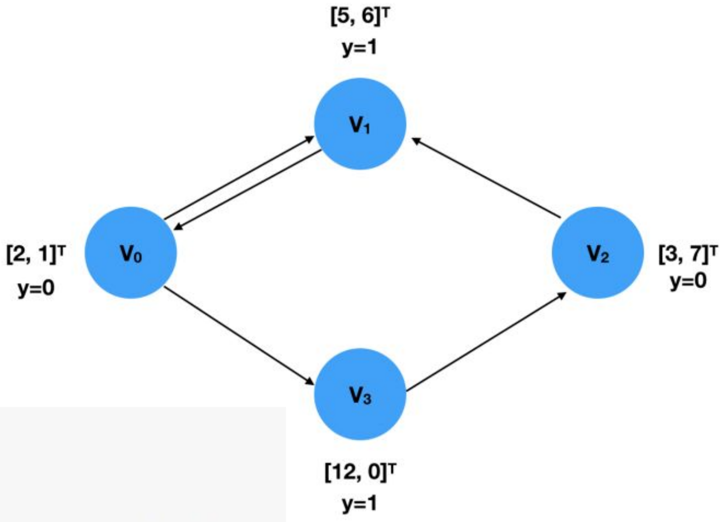
- Understanding concepts of GNN
- COO (Coordinate Format)
- Adaptation of Data to simple Graphs (nodes , edges)
- Create simples models

COO of data



Example Graph

COO of data



Example Graph

In [3]:
```
import torch
from torch_geometric.data import Data


x = torch.tensor([[2,1], [5,6], [3,7], [12,0]], dtype=torch.float)
y = torch.tensor([0, 1, 0, 1], dtype=torch.float)

edge_index = torch.tensor([[0, 2, 1, 0, 3],
                           [3, 1, 0, 1, 2]], dtype=torch.long)


data = Data(x=x, y=y, edge_index=edge_index)
data
```

Out[3]:
```
Data(edge_index=[2, 5], x=[4, 2], y=[4])
```

Toy data

```python

def dummy_graph(n_nodes, node_dim, edge_dim):
    # Connect every node together
    edge_index = torch.tensor([[i, j] for i in range(0, n_nodes)
                                      for j in range(i+1, n_nodes)]).t()
    n_edges = edge_index.shape[1]
    # Generate node and edge features
    x = torch.randn(n_nodes, node_dim)
    e = torch.randn(n_edges, edge_dim)
    # Construct the graph
    return torch_geometric.data.Data(x=x, edge_index=edge_index, edge_attr=e)

class DummyDataset(Dataset):

    def __init__(self, n_samples, n_nodes, node_dim, edge_dim):
        super(DummyDataset, self).__init__()
        self.graphs = [dummy_graph(n_nodes, node_dim, edge_dim) for i in range(n_samples)]

    def __getitem__(self, index):
        return self.graphs[index]

    def __len__(self):
        return len(self.graphs)

```

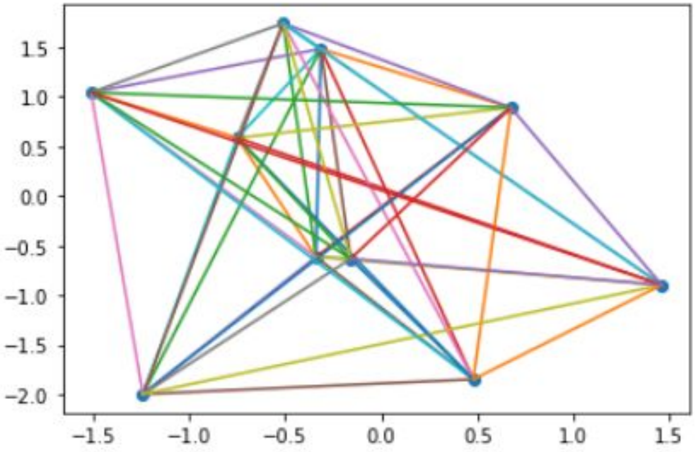
Toy data

In [5]:
```
# Dummy data config
n_nodes = 10
node_dim = 2
edge_dim = 3
```

In [6]:
```
data = dummy_graph(n_nodes, node_dim, edge_dim)
data
```

Out[6]:
```
Data(edge_attr=[45, 3], edge_index=[2, 45], x=[10, 2])
```

In [5]:
```
draw_graph(data)
```



Real Data

```
(base) sataucuri@headtop:~$ ls /data/sataucuri/heptrkx/data/hitgraphs_med_002/
event000001000_g000.npz      event000002365_g003.npz      event000003780_g006.npz      event000005146_g001.npz
event000001000_g000_ID.npz   event000002365_g003_ID.npz   event000003780_g006_ID.npz   event000005146_g001_ID.npz
event000001000_g001.npz      event000002365_g004.npz      event000003780_g007.npz      event000005146_g002.npz
event000001000_g001_ID.npz   event000002365_g004_ID.npz   event000003780_g007_ID.npz   event000005146_g002_ID.npz
event000001000_g002.npz      event000002365_g005.npz      event000003781_g000.npz      event000005146_g003.npz
event000001000_g002_ID.npz   event000002365_g005_ID.npz   event000003781_g000_ID.npz   event000005146_g003_ID.npz
event000001000_g003.npz      event000002365_g006.npz      event000003781_g001.npz      event000005146_g004.npz
event000001000_g003_ID.npz   event000002365_g006_ID.npz   event000003781_g001_ID.npz   event000005146_g004_ID.npz
event000001000_g004.npz      event000002365_g007.npz      event000003781_g002.npz      event000005146_g005.npz
event000001000_g004_ID.npz   event000002365_g007_ID.npz   event000003781_g002_ID.npz   event000005146_g005_ID.npz
event000001000_g005.npz      event000002366_g000.npz      event000003781_g003.npz      event000005146_g006.npz
event000001000_g005_ID.npz   event000002366_g000_ID.npz   event000003781_g003_ID.npz   event000005146_g006_ID.npz
event000001000_g006.npz      event000002366_g001.npz      event000003781_g004.npz      event000005146_g007.npz
event000001000_g006_ID.npz   event000002366_g001_ID.npz   event000003781_g004_ID.npz   event000005146_g007_ID.npz
event000001000_g007.npz      event000002366_g002.npz      event000003781_g005.npz      event000005147_g000.npz
event000001000_g007_ID.npz   event000002366_g002_ID.npz   event000003781_g005_ID.npz   event000005147_g000_ID.npz
event000001001_g000.npz      event000002366_g003.npz      event000003781_g006.npz      event000005147_g001.npz
event000001001_g000_ID.npz   event000002366_g003_ID.npz   event000003781_g006_ID.npz   event000005147_g001_ID.npz
event000001001_g001.npz      event000002366_g004.npz      event000003781_g007.npz      event000005147_g002.npz
event000001001_g001_ID.npz   event000002366_g004_ID.npz   event000003781_g007_ID.npz   event000005147_g002_ID.npz
event000001001_g002.npz      event000002366_g005.npz      event000003782_g000.npz      event000005147_g003.npz
event000001001_g002_ID.npz   event000002366_g005_ID.npz   event000003782_g000_ID.npz   event000005147_g003_ID.npz
event000001001_g003.npz      event000002366_g006.npz      event000003782_g001.npz      event000005147_g004.npz
event000001001_g003_ID.npz   event000002366_g006_ID.npz   event000003782_g001_ID.npz   event000005147_g004_ID.npz
event000001001_g004.npz      event000002366_g007.npz      event000003782_g002.npz      event000005147_g005.npz
event000001001_g004_ID.npz   event000002366_g007_ID.npz   event000003782_g002_ID.npz   event000005147_g005_ID.npz
event000001001_g005.npz      event000002367_g000.npz      event000003782_g003.npz      event000005147_g006.npz
```

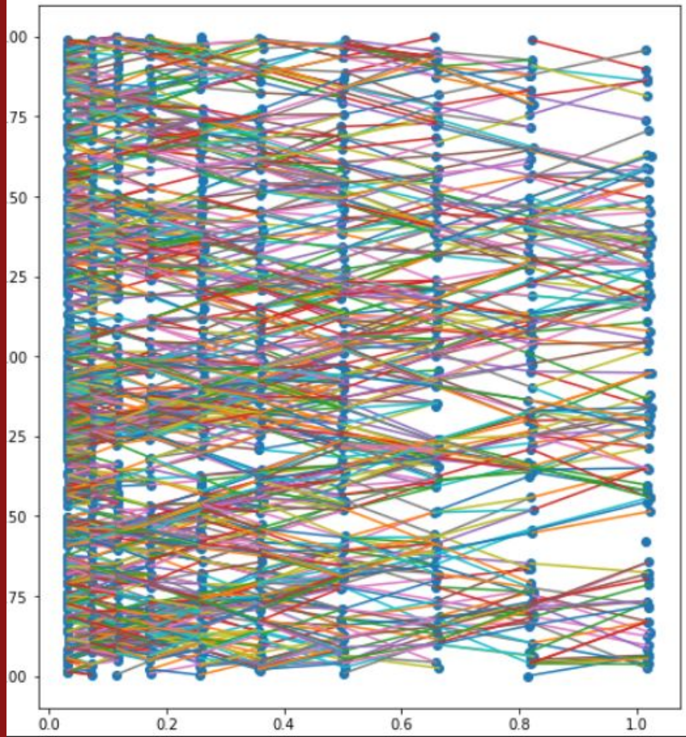
```python
import os
from collections import namedtuple

import numpy as np
from torch_geometric.data import Batch

def load_graph(filename):
    with np.load(filename) as f:
        x, y = f['X'], f['y']
        Ri_rows, Ri_cols = f['Ri_rows'], f['Ri_cols']
        Ro_rows, Ro_cols = f['Ro_rows'], f['Ro_cols']
        n_edges = Ri_cols.shape[0]
        edge_index = np.zeros((2, n_edges), dtype=int)
        edge_index[0, Ro_cols] = Ro_rows
        edge_index[1, Ri_cols] = Ri_rows
    return x, edge_index, y

class HitGraphDataset(Dataset):
    """PyTorch dataset specification for hit graphs"""

    def __init__(self, input_dir, n_samples=None):
        input_dir = os.path.expandvars(input_dir)
        filenames = [os.path.join(input_dir, f) for f in os.listdir(input_dir)
                     if f.startswith('event') and f.endswith('.npz')]
        self.filenames = filenames if n_samples is None else filenames[:n_samples]

    def __getitem__(self, index):
        x, edge_index, y = load_graph(self.filenames[index])
        return torch_geometric.data.Data(x=torch.from_numpy(x),
                                         edge_index=torch.from_numpy(edge_index),
                                         y=torch.from_numpy(y))

    def __len__(self):
        return len(self.filenames)
```

```
dataset = HitGraphDataset(input_dir=data_dir, n_samples=n_samples)
```

```
dataset.__getitem__(0)
```

```
a(edge_index=[2, 4932], x=[1809, 3], y=[4932])
```

```
draw_graph(dataset.__getitem__(0))
```

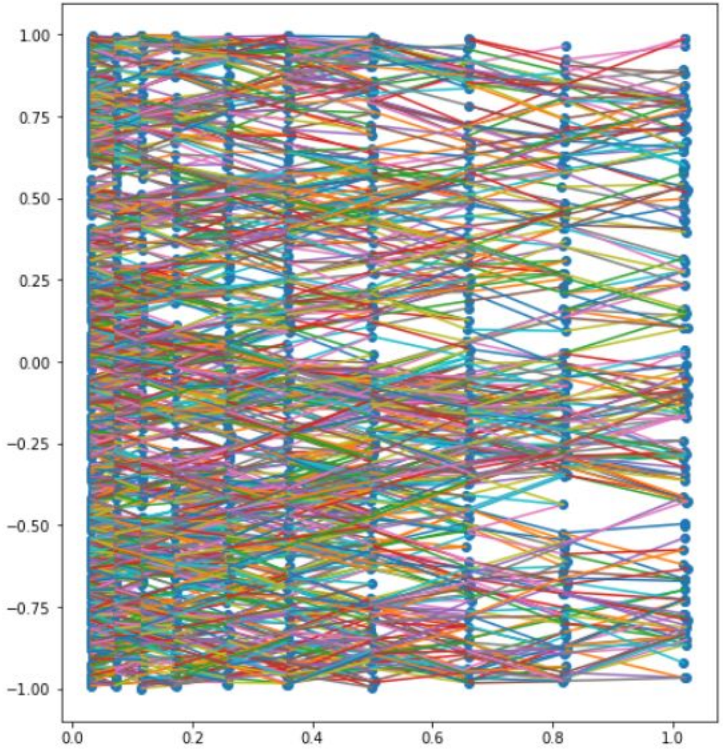



```
1 dataset = HitGraphDataset(input_dir=data_dir, n_samples=n_samples)
```

```
1 dataset.__getitem__(100)
```

```
Data(edge_index=[2, 7507], x=[2261, 3], y=[7507])
```

```
1 draw_graph(dataset.__getitem__(100))
```



```python
graph = dataset.__getitem__(100)
```

```python
graph.x
```

```
tensor([[ 0.0316,  0.3078,  0.0258],
        [ 0.0722,  0.3229,  0.0508],
        [ 0.1159,  0.3398,  0.0776],
        ...,
        [ 0.6597, -0.0054,  0.5783],
        [ 0.8209, -0.1432,  0.7372],
        [ 1.0207, -0.3250,  0.9432]])
```

```python
graph.edge_index
```

```
tensor([[   0,    0,    0,  ..., 2219, 2249, 2259],
        [   1,  173,  792,  ..., 2220, 2250, 2095]])
```

Models

```
import torch
from torch.nn import Linear
from torch_geometric.nn import GCNConv


class GCN(torch.nn.Module):
    def __init__(self):
        super(GCN, self).__init__()
        torch.manual_seed(12345)
        self.conv1 = GCNConv(dataset.num_features, 4)
        self.conv2 = GCNConv(4, 4)
        self.conv3 = GCNConv(4, 2)
        self.classifier = Linear(2, dataset.num_classes)

    def forward(self, x, edge_index):
        h = self.conv1(x, edge_index)
        h = h.tanh()
        h = self.conv2(h, edge_index)
        h = h.tanh()
        h = self.conv3(h, edge_index)
        h = h.tanh()  # Final GNN embedding space.

        # Apply a final (linear) classifier.
        out = self.classifier(h)

        return out, h

model = GCN()
print(model)
```

```
GCN(
```

```python
class NodeModel(nn.Module):
    def __init__(self, node_dim, edge_dim, num_hidden):
        super(Edge, self).__init__()
        # message passing
        self.model = nn.Sequential(
            nn.Linear(node_dim+edge_dim, num_hidden),
            nn.ReLU(),
            nn.Linear(num_hidden, edge_dim),
            nn.ReLU()
        )
    def forward(self, x_sender, x_receiver, e, batch=None):
        inputs = torch.cat([x_sender, x_receiver, e], 1)
        return self.model(inputs)

class EdgeModel(nn.Module):
    """A simple node module"""

    def __init__(self, node_dim, edge_dim, hidden_dim):
        super(NodeModule, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(edge_dim + node_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, node_dim),
            nn.ReLU()
        )

    def forward(self, x, edge_index, e, batch=None):
        # Sum edge features at each receiver
        senders, receivers = edge_index
        aggr = scatter_add(e, receivers, dim=0)
        inputs = torch.cat([x, aggr], 1)
        return self.network(inputs)
```

```python
class GraphNeuralNewtwork(nn.Module):
    """A simple graph network"""

    def __init__(self, input_node_dim, input_edge_dim,
                 hidden_node_dim, hidden_edge_dim,
                 n_graph_iters=1):
        super(GNN, self).__init__()
        self.n_graph_iters = n_graph_iters
        self.node_encoder
        # Meta layer take a graph as input and returns and updated graph as output
        self.graph_layer = MetaLayer(
            edge_model=EdgeModel(input_node_dim, input_edge_dim, hidden_edge_dim),
            node_model=NodeModedl(input_node_dim, input_edge_dim, hidden_node_dim)
        )

    def forward(self, data):
        return self.graph_layer(data.x, data.edge_index, data.edge_attr)
```

Repository

- Github
  - https://github.com/stonescenter/graph-tracking/tree/master/notebooks

Fin

Perguntas?