Graph Neural Networks for Clustering in DUNE

Jeremy Hewes
DUNE FD sim/reco meeting
16th September 2019

Introduction

- Update on progress to apply **graph convolutional network (GCN)** techniques for reconstruction in LArTPCs.

- There are many potential applications for such approaches in LArTPC reconstruction.

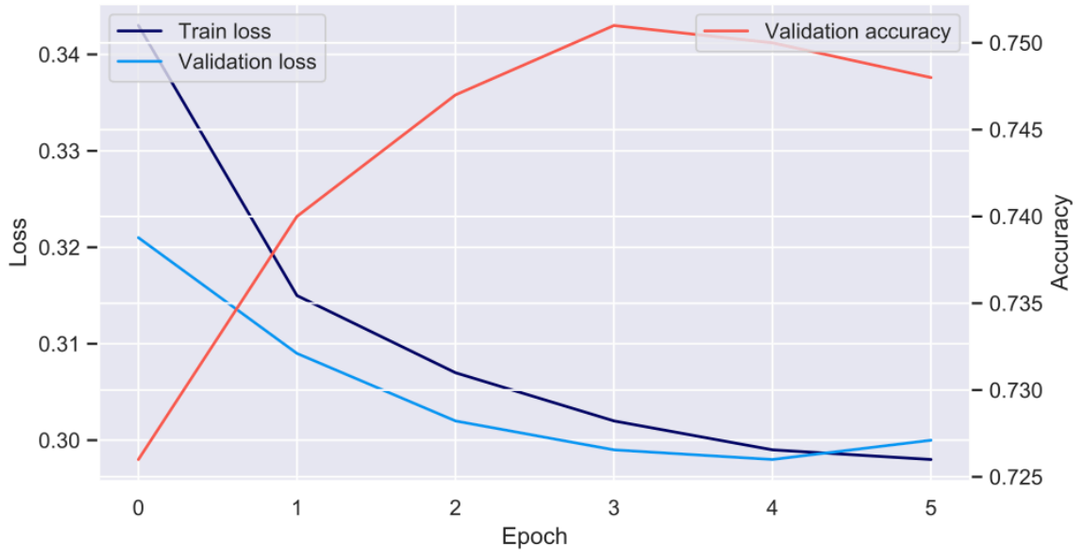
- This study specifically discusses using such networks to **cluster spacepoints** by classifying connections between nodes.

- This work performed in collaboration with **Exa.TrkX**, who have pursued such techniques with great success in the LHC world (arXiv:1810.06111).

Reminder

Network performance

- Some small evidence of learning over the first few epochs, but clearly **much room for improvement**!
- Succeeded in initial goal: **constructing workflow** to produce graphs and train networks in TPCs

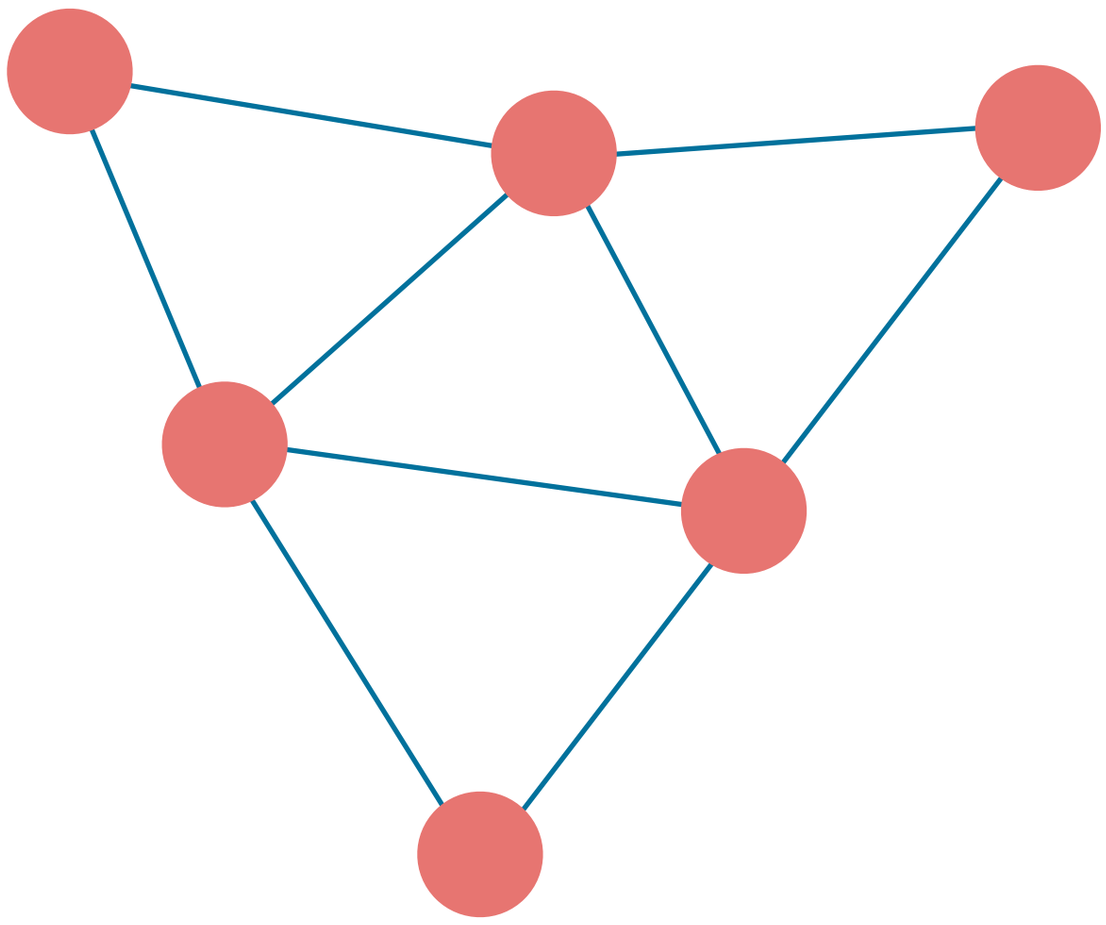


- **Next steps**:
  - Train on WireCell spacepoints.
    - Less dense point clouds may prevent the need to train on cluster-wise graphs.
  - Strip out some vestigial machinery from the HEP.TrkX network.
  - Investigate node classification in more detail.
  - Explore entirely different graph constructions.
    - Cluster-wise graphs for particle ID?

GNNs for DUNE – J. Hewes – 10th June 2019 11

Graph convolution technique

- Describe information structure as a **graph** represented by **nodes** and **edges**.



- **Nodes** are generalised as quantised objects with some arbitrary set of **features**.
- **Edges** describe the **relationships** between nodes.
- Perform convolutions on nodes and edges to learn relationships within the graph.
- Output is user-defined:
  - Classify nodes or edges.
  - Classify full graph.
  - Regression outputs.

What's new?

- Previous GCN implementation used a graph network coded up in standard PyTorch.
- This implementation used dense tensor multiplications, which very quickly run into serious memory issues.
  - To get around things, I was forced into sub-optimal places like clustering per-particle and heavily restricting the size of the graph.
- Moved to **pytorch-geometric**, a PyTorch toolset specifically designed for graph convolutions.
- Contains specialised data structures, implementations of many different flavours of graph network, and uses sparse matrix multiplication.
  - Previously, clustering a full event in one shot was not possible.
  - Using pytorch-geometric, such approaches are absolutely viable.

Simulation

- Simulation is **atmospheric neutrino** interactions in the **full 10kt geometry**.
- Produced three 600k MC training samples.
  - $\nu_{e,\mu,\tau}$ (each flux-swapped from initial ν_e & ν_μ flux).
  - Standard simulation chain, run reconstruction up to 3D spacepoint finding (**Pandora** & **SpacePointSolver**).
- Since the principal objective of this study is clustering, focus on only one sample for now as a test case: ν_μ interactions.
- Write output to **HDF5** files for downstream processing.
  - Currently using HighFive C++ wrapper for HDF5 writing.
  - SCD made H5CPP available in the DUNE environment, but it is not compatible with art.
  - Hoping to get **ntuple** (FNAL HDF5 tool for HEP) available for DUNE.

Input production

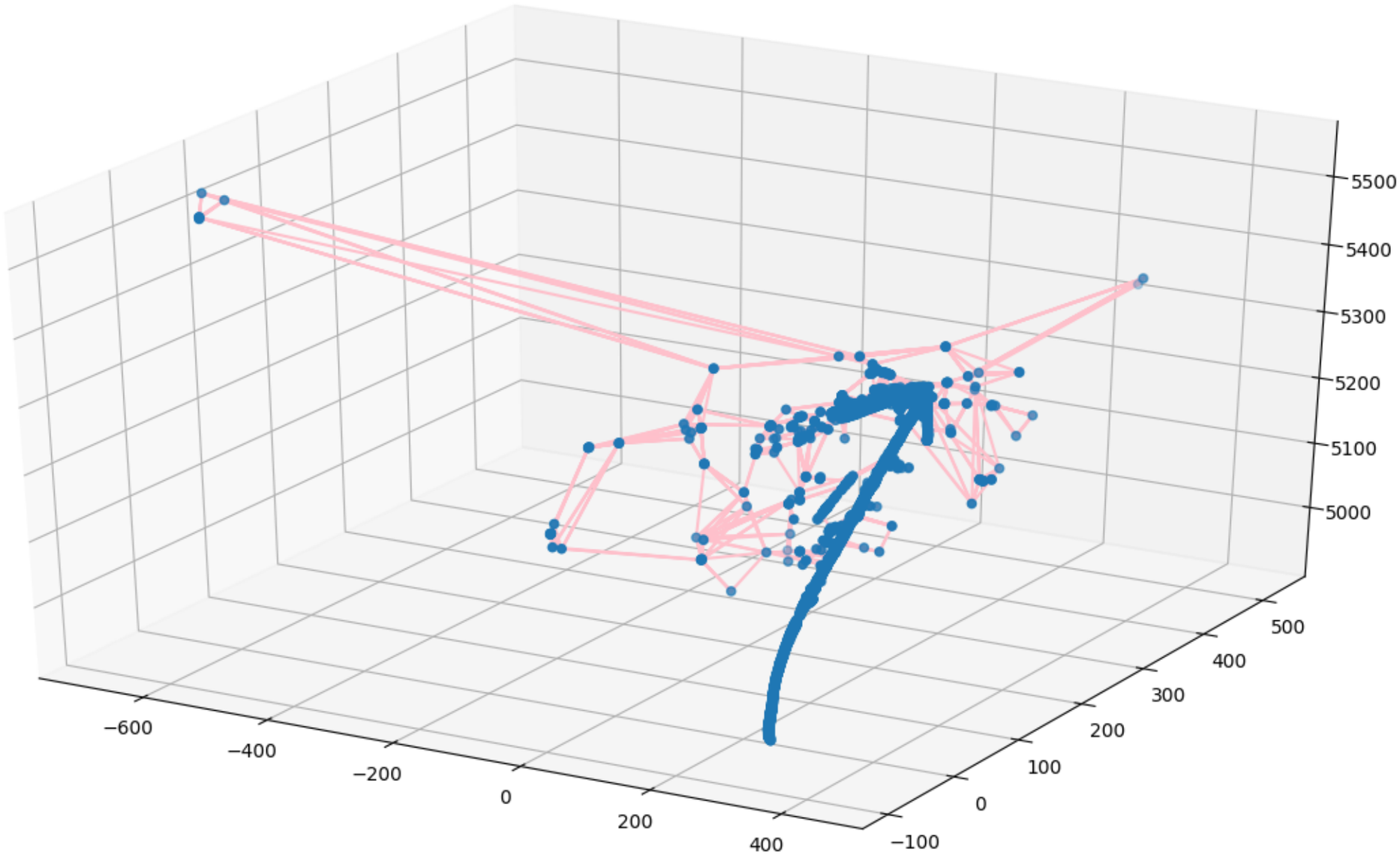
- Graphs are produced from **reconstructed spacepoints**.
  - The initial goal of this study was to cluster spacepoints from WireCell.
  - Produced graphs using both Pandora and SpacePointSolver.
    - These studies will show the latter, since they are the most direct analogue to WireCell spacepoints.
- Construct a graph node for each spacepoint.
- Node features include **charge of associated hits**, **xyz position**, **number of nearest neighbours**, and the **G4 track ID of the MC particle that contributed the most charge**.
  - G4 ID is obviously not used during training, but is used to construct the ground truth during preprocessing.
    - Edge label is 1 if nodes share the same G4 ID, 0 if not.
    - A more complex ground truth definition may be necessary (see later).

Preprocessing

- Preprocess graphs from HDF5 format into pytorch-geometric data format.
- For a graph with **n** nodes, there are **n^2** possible edges.
  - In order to limit the size of the graph, it's necessary to restrict the number of edges to reduce overhead while still retaining enough for the output to make sense.
- For each node, keep only the closest **k** nodes with a lower node index.
- For these studies, I chose **k=5**.
  - There'll be some discussion later on why I intend to increase this number for subsequent studies.

Graph inputs

- Spacepoint graph looks something like this:

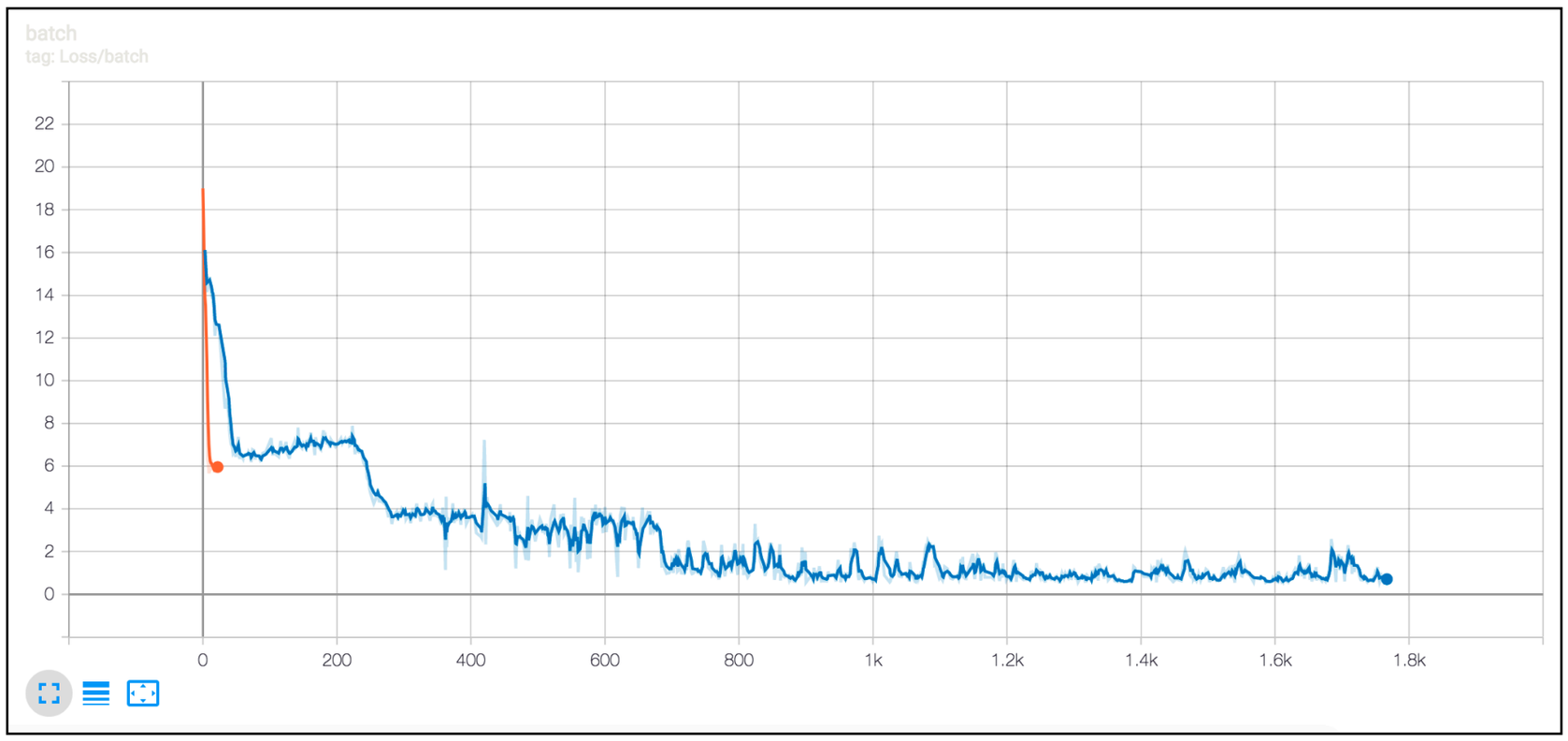


Network architecture

- Architecture is similar to previous implementation.
- Alternate between **node network**, which aggregates edge features in latent space into latent node features, and **edge network** that aggregates latent node features into new latent edge features.
- Utilise 64 latent features for both nodes and edges (note this is different to last time, where collapsed down to a single edge feature with each iteration).
- This is often referred to as a **message-passing** network.
  - With each iteration, features from adjacent nodes are convolved into a node's latent features. Over multiple network iterations, features from any given node propagate out further and further across the graph.

Training

- Training objective is **binary cross-entropy** loss on edge labels.
- When training over four GPUs simultaneously, training one epoch of ~200k graphs takes around **seven minutes**.
- **Batch size of 100** distributed across those four GPUs.
- Network trains very quickly, and then plateaus at a loss of around 0.5.

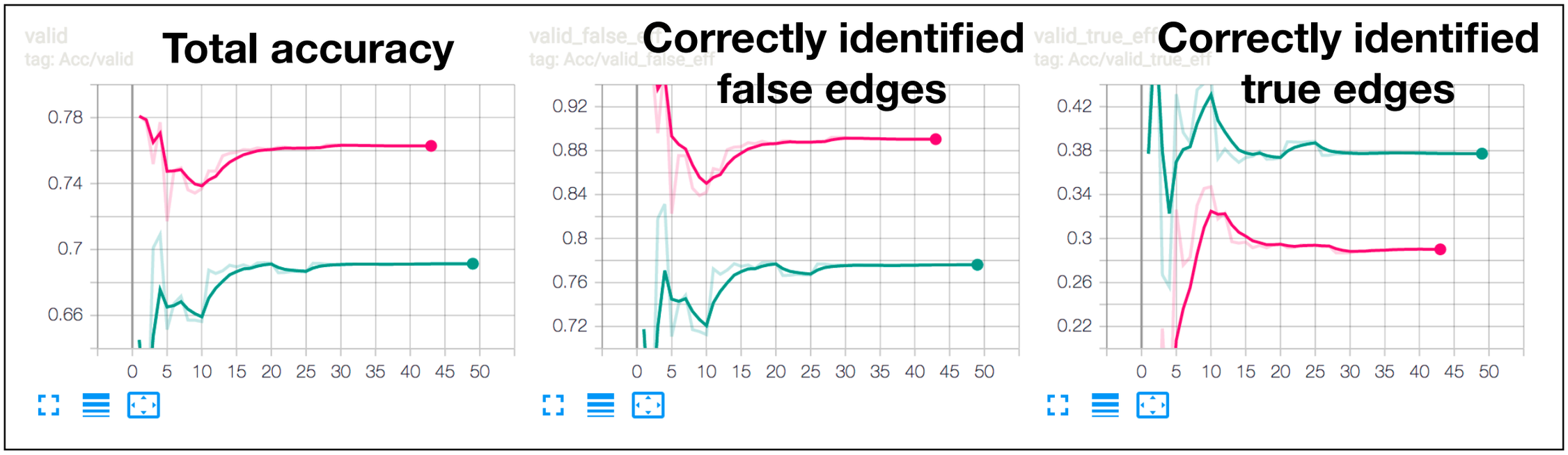


Training parameters	
Learning rate	0.0005
Optimiser	Adam
Message passing iterations	6

Performance

- Initial loss reduction is encouraging, but in this case, misleading.
- The network is lazy!
  - The majority of edges in the graph are false, and so the network can achieve high accuracy by simply classifying all edges as false.
- We can prevent it from doing this by weighting the loss function during training.
  - Weight true edges up, and false edges down, to encourage the network to pay more attention to true edges.
  - This is a balancing act! Weight too heavily, and the network will learn the opposite lesson, ie. classify all edges as true.
  - Finding the right weights to get the network to sit in a middle ground.

Further tests



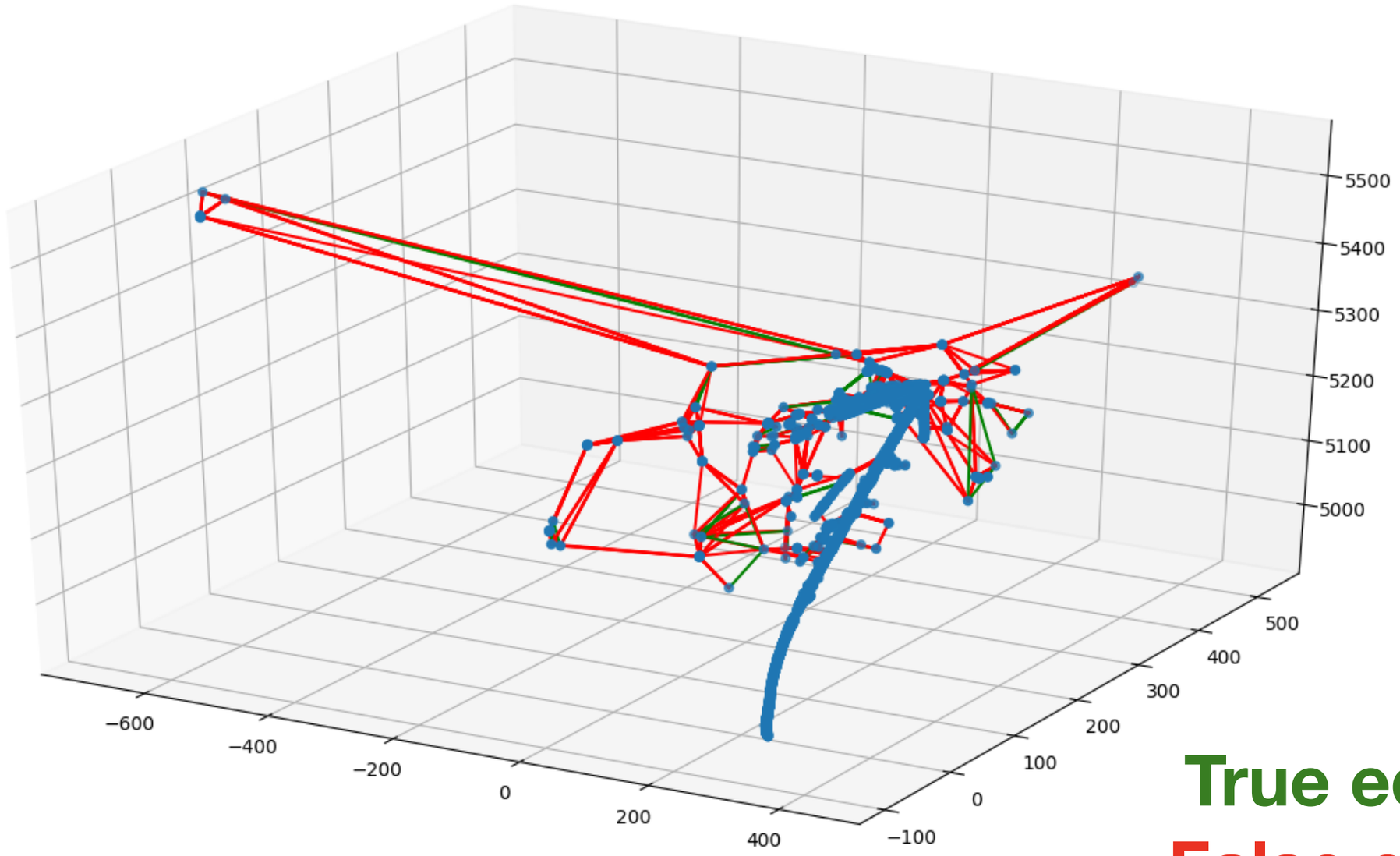
- Weight true edges up and true edges down.
- Monitor efficiency for true and false edges independently.
- Network starting to learn, but plenty of tradeoff in efficiency labelling true and false edges.
- Starting to wonder how optimised graph structure is for learning…

True weight: 1.5
False weight: 0.7

True weight: 1.5
False weight: 0.6

Graph analysis

- Look closer at graph definition:



True edges

False edges

Graph analysis

- Not clear that a naïve definition of ground truth (matching true G4 ID between spacepoints) is good enough.
- From closer analysis, 20-40% of nodes in a typical graph have no true edge attached.
- Simple solution to this may be to simply increase the limit on number of edges per node, but I suspect a better definition of
- A couple of ideas to explore here:
  - Take particle parentage into account when defining ground truth?
  - See if G4 IDs from small particles are confusing things here & possibly come up with some kind of grouping scheme?

Summary

- Working on **graph convolutional networks** for clustering in the DUNE far detector.

- Produced high-stats **atmospheric neutrino** samples in the **full 10kt** geometry for training purposes.

- Moved to **PyTorch Geometric** framework, and achieved vast improvements in terms of training efficiency and stability as a result.

- Next steps:
  - Investigate graph definition, including **ground truth**, in more detail.
  - Experiment with different weighting schemes & network architectures.