Guide to Machine Learning in Pandora

Andy Chappell, Isobel Mawby, Matt Osbiston

March 28, 2025

Abstract

This document outlines the procedures for training and using deep neural networks within the Pandora pattern recognition framework.

1 Introduction

As of LArContent v04.00.00, Pandora has supported the use of neural networks via the LibTorch package. To date, a number of neural networks have found application in various reconstruction workflows across multiple LArTPC experiments. In particular, there are neural networks that handle neutrino interaction vertex identification, higher order vertex identification, separation of supernova neutrino interactions from low energy background interactions, and determination of the hierarchical relationships between reconstructed particles.

Each section of this document focuses on one such network and outlines the procedure for training and using that network.

2 Primary vertexing

2.1 Training set preparation

Stuff about in Pandora preparation.

Generation of the training samples is handled within the DlVertexingAlgorithm. The `PrepareTrainingSample` function generates a set of output CSV files for each of the views U, V and W.

XML stuff here

2.2 Training the network

Stuff about training the network.

2.3 Using the network

Stuff about inference.

Figure 1: No frogs were harmed in the training of these networks.

3 Higher order vertexing

3.1 Training set preparation

Stuff about in Pandora preparation.

3.2 Training the network

Stuff about training the network.

3.3 Using the network

Stuff about inference.

4 Interaction hierarchy building

4.1 Training set preparation

The training set is created by the DlNeutrinoHierarchyAlgorithm. The algorithm replaces `LArNeutrinoHierarchy` and is placed between between `LArNeutrinoCreation` and `LArNeutrinoDaughterVertices` in `PandoraSettings_Neutrino_DUNEFD.xml`. The following configuration is used to train the algorithm's networks:

```
<algorithm type = "LArMLPNeutrinoHierarchy">
   <NeutrinoPfoListName>NeutrinoParticles3D</NeutrinoPfoListName>
   <PfoListNames>TrackParticles3D ShowerParticles3D</PfoListNames>
   <TrainingMode>true</TrainingMode>
   <tool type = "LArMLPCheatHierarchy" description = "MLPCheatHierarchyTool"/>
   <tool type = "LArMLPPrimaryHierarchy" description = "MLPPrimaryHierarchyTool">
     <Normalise>false</Normalise>
     <TrainingMode>true</TrainingMode>
   </tool>
   <tool type = "LArMLPLaterTierHierarchy" description = "MLPLaterTierHierarchyTool">
     <Normalise>false</Normalise>
     <TrainingMode>true</TrainingMode>
   </tool>
</algorithm>
```

If ran in training mode, the algorithm will produce a training file, called `DLHierarchyTrainingFile.root`, which contains the following trees:

- `EventTree`,
- `PrimaryTrackTree`,
- `PrimaryShowerTree`,
- `LaterTierTrackTrackTree`, and
- `LaterTierTrackShowerTree`

The last four trees are the training trees and are flattened in order to minimise file size and I/O runtime. This means that each entry in the tree corresponds to an edge of a potential parent-child link. The `EventTree` is only used when tuning the network, where it is used to turn the flat trees into event-level trees.

4.2 Training the networks

The variables used in the algorithm's networks are orientation dependent i.e they depend on which 'endpoint' is assumed to be a particle's start/end position. This naturally leads to a two stage process when deciding whether a parent-child link is correct. In the first stage, one passes the variables of each definable edge through an 'edge model'. The edge model's output is a probability score for the following three classes: correct and in the correct orientation, correct but in the wrong orientation, or incorrect. In the second stage, the edge network outputs are concatenated together and passed through a 'classification model'. The classification model outputs a score representing the probability that the parent-child link is correct.

For tracks one can define two endpoints, and for showers we can define one endpoint (the shower region has no physical endpoint). This leads to the following workflows:

- Primary Tracks: Two edges can connect the neutrino and track particle. We have a branch model and a classification model.
- Primary Showers: One edge can connect the neutrino and shower particle. We only have a classification model.
- Track-Track links: Four edges can connect the track parent and track child particles. We have a branch model and a classification model.
- Track-Shower links: Two edges can connect the track parent and track child particles. We have a branch model and a classification model.

To train the primary models:

1. Run `PrimaryTier/WritePrimaryTierFile.ipynb` with `isTrackMode` set to `True` to obtain `/files/hierarchy_TRAIN_track.npz` and set to `False` to obtain `/files/hierarchy_TRAIN_shower.npz`.
2. Run `PrimaryTier/TrainPrimaryTierModel_tracks.ipynb` to train the primary track edge and classifier models and `PrimaryTier/TrainPrimaryTierModel_showers.ipynb` to to train the primary shower classifier model.

To train the 'later-tier' models:

1. Run `LaterTier/WriteLaterTierFile.ipynb` with `isTrackMode` set to `True` to obtain `/files/hierarchy_TRAIN_later_tier_track.npz` and set to `False` to obtain `/files/hierarchy_TRAIN_later_ti`
2. Run `LaterTier/TrainLaterTierModel.ipynb` with `isTrackMode` set to `True` to train the later-tier track-track edge and classifier models.
3. Run `LaterTier/TrainLaterTierModel.ipynb` with `isTrackMode` set to `False` to train the later-tier track-shower edge and classifier models.

After these steps one should have created the following models:

- `primary_track_branch_model`
- `primary_track_classifier_model`
- `primary_shower_classifier_model`
- `track_track_branch_model`
- `track_track_classifier_model`
- `track_shower_branch_model`
- `track_shower_classifier_model`

4.3 Using the network

The primary networks are invoked in the `DLPrimaryHierarchy` tool, which is called by the `LArNeutrinoHierarchy` algorithm in its `GetPrimaryScore` function. The tool returns a score reflecting how likely the input neutrino-child particle link is to be correct. This score is then used, with reference to predetermined thresholds, to construct the primary tier.

The later-tier networks are invoked in the `DLLaterTierHierarchy` tool, which is called by the `LArNeutrinoHierarchy` algorithm in its `GetLaterTierScore` function. The tool returns a score reflecting how likely the input parent-child particle link is to be correct. This score is then used, with reference to predetermined thresholds, to construct the later-tiers.

The predetermined thresholds are determined offline using the following workflow:

1. `AddScores.ipynb` is ran on the training file produced by the `LArNeutrinoHierarchy` algorithm. This notebook converts the input flat trees into event-level trees and runs the models to obtain the primary/later-tier scores for all potential links.

2. `TuneCut.ipynb` must then be ran. This notebook builds the hierarchies using the same logic as implemented in the `LArNeutrinoHierarchy` algorithm. The hierarchy building proceeds in a step-wise fashion, and in each stage thresholds are used to determine whether parent-child links are made. In this notebook, metrics are produced at each stage of the hierarchy building allowing the user to identify the optimal thresholds.

3. `HierarchyMetrics.ipynb` is finally ran. This notebook reports detailed hierarchy building metrics at each stage of the hierarchy building procedure.

5 Supernova neutrino and low energy background separation

5.1 Training set preparation

Stuff about in Pandora preparation.

5.2 Training the network

Stuff about training the network.

5.3 Using the network

Stuff about inference.

References