

Binary Classification Model for Decay Events

Murdock Aubry, Prince Bhaura, Yijie Wang

University of Toronto

November 28, 2023

Table of Contents

1 Brief Introduction to NuGraph GNN

2 Binary Classification of Decay Events

3 Model Implementation

Table of Contents

1 Brief Introduction to NuGraph GNN

2 Binary Classification of Decay Events

3 Model Implementation

Brief Introduction to NuGraph

- ① NuGraph is a graph neural network (GNN) designed for reconstructing particle interactions in neutrino physics detector environments.
- ② Trained and tested on DUNE data, this model is primarily used for the classification of detector hit particle type.
- ③ Additional functions include background hit rejection, event classification, clustering and vertex reconstruction.

Github: [NuGraph](#)

Data Collection: Time Projection Chamber

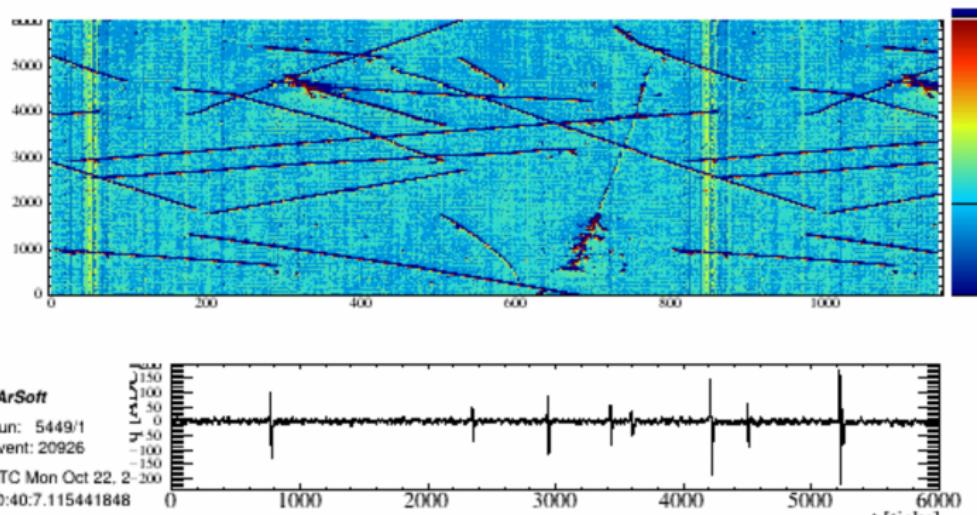


Figure: Example of pedestal-subtracted data for one ProtoDUNE-SP wire plane.
Source: [Introduction to DUNE. Vol 1](#)

Data Collection: Time Projection Chamber

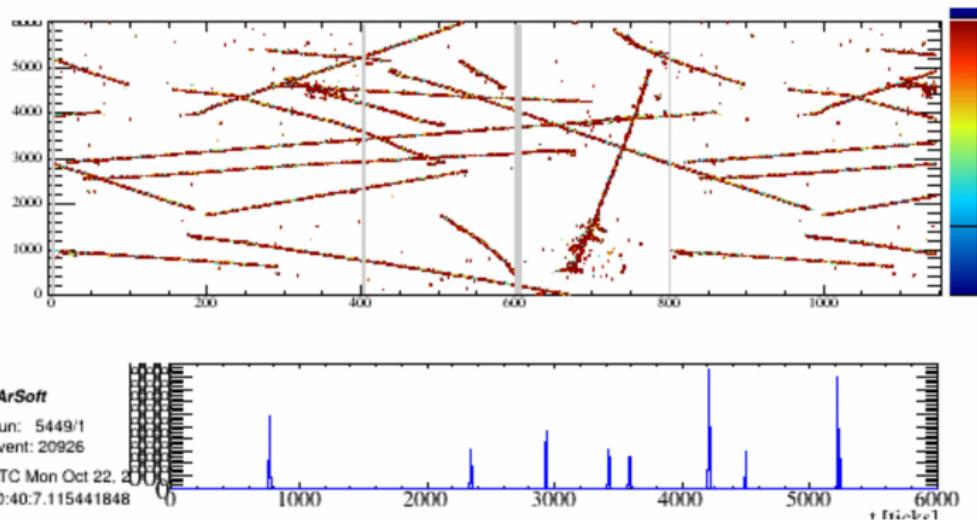
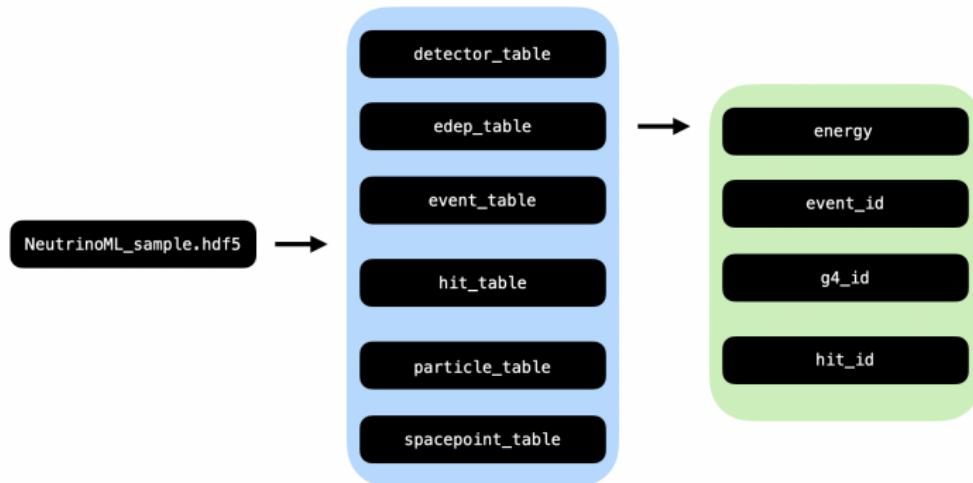


Figure: Calibrated, deconvoluted pedestal-subtracted data for one ProtoDUNE-SP wire plane. Source: [Introduction to DUNE. Vol 1](#)

Data Collection: .HDF5



Algorithm Structure

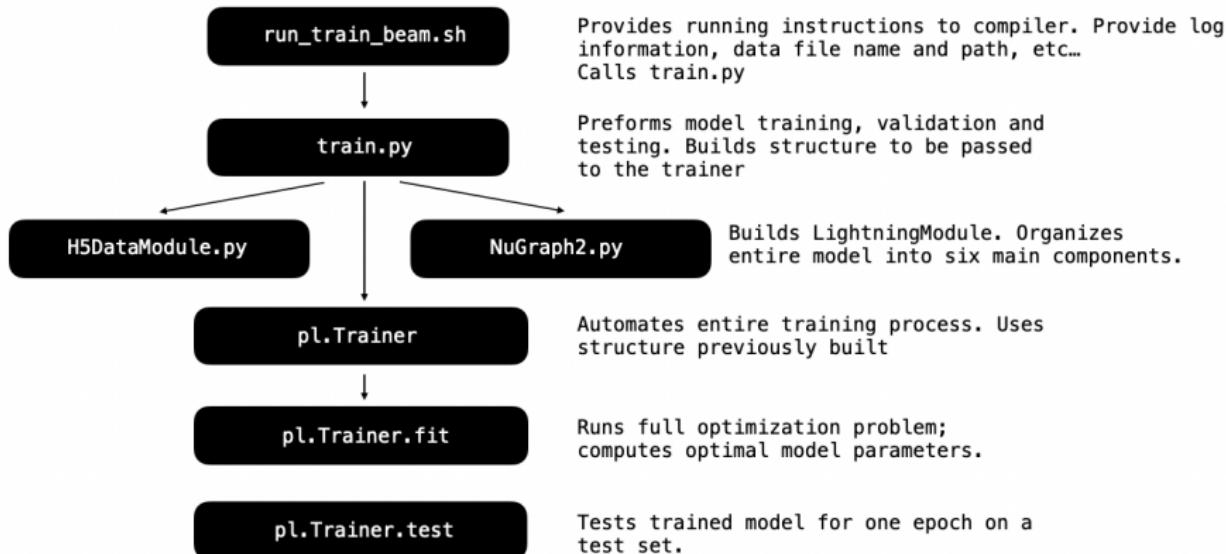


Figure: Description of NuGraph Neural Network. See [LightningModule](#) for further documentation.

NuGraph2.py Structure

Main Components	NuGraph2.py	Second Components
<code>__init__</code>	Initializes structure. Reads information for model (i.e. learning rates, separates planes, number of iterations, etc...) Calls <code>Encoder</code> , <code>NexusNet</code> , <code>PlaneNet</code> } Distinct architectures Specifies <code>decoder</code> based on inputs.	
<code>training_step</code>	Performs a single training step For each decoder in the list, computes loss, returns total loss.	
<code>validation_step</code>	Performs single validation step	
<code>test_step</code>	Performs single test step	
<code>predict_step</code>	Takes in a batch, calls 'step'.	
<code>configure_optimizers</code>	Specifies optimization method; currently uses <code>AdamW</code> Adaptively adjusts learning rate via <code>torch.optim.lr_scheduler</code>	<code>forward</code> <code>step</code> <code>on_training_start</code> <code>on_test_epoch_end</code> <code>log_memory</code>

Figure: See [AdamW](#)

, [OneCycleLR](#).

Table of Contents

1 Brief Introduction to NuGraph GNN

2 Binary Classification of Decay Events

3 Model Implementation

Building Training Data

- New training data will need to be configured as follows:
 - Event follows the decay mode we are interested in
 - Event does not follow the decay mode we are interested in
- This configuration can be done by looking at the physical characteristics (e.g energy, momentum, spin) which can be found in the existing data
- Create a script to classify decay events of interest from existing data
 - Once classified, attach a label (e.g 1 → decay of interest has happened, 0 → decay of interest has not happened) and send to new training model

Outgoing Hadron Decay Modes

- Focus will be on looking for Hadron decay modes (which include π^\pm/ρ^\pm)
- Information about the outgoing τ produced in $\nu_\tau(\bar{\nu}_\tau)$ nuclear reactions can be inferred through physical characteristics of the visible decay products
- Current plan is to classify data through Pion decay modes with the correct spins

Table of Contents

1 Brief Introduction to NuGraph GNN

2 Binary Classification of Decay Events

3 Model Implementation

Proposed Model Adjustments

- Alter model to be able to handle this new class of data
- Reconfigure H5DataModule.py
- Choose ideal architecture (i.e. mimic NexusNet, PlaneNet, etc...)
- Choose ideal loss function (i.e CE? LogCosh?)

Layer Description

Feedforward Layers

Encoder

NexusNet

PlaneNet

Two-layer sequential network with one linear and one non-linear layer:

$$z = \tanh(Wx + b)$$

W, b are learned parameters

EdgeNet: Two-layer sequential network with Softmax.

$$z = \sigma(Wx + b) \quad (\sigma(x))_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

NodeNet: Four-layer sequential network.

$$z = \tanh \left(W_2 \tanh(W_1 x + b_1) + b_2 \right)$$

W_i, b_i are learned parameters

Very similar to NexusNet

Decoder Descriptions

DecoderBase

Base class for all NuGraph decoders

Specifies loss update based on cooling parameters

Updates metrics (loss information over time)

SemanticDecoder

Convolve down to a single score per semantic class for each plane

Uses RecallLoss

FilterDecoder

Convolve down to a single score, filters out noise. Uses BCELoss

EventDecoder

Convolve graph node features down to a single score for entire event

Uses RecallLoss

VertexDecoder

Uses LogCoshLoss

score: model's guess

Loss Functions

RecallLoss

Really just cross-entropy loss:

$$\mathcal{L}(\theta) = \text{Avg}_i CE(y_i, \sigma(f(x_i, \theta)))$$

$$CE(p, \hat{p}) = - \sum_i p_i \ln(\hat{p}_i)$$

LogCoshLoss

Uses the soft plus function;

$$X = \sqrt{\sum_i (f(x_i, \theta) - y_i)^2}$$

$$L(x) = \log(1 + \exp(x))$$

Smooth analog of ReLU

$$\mathcal{L}(\theta) = \text{Avg}_i (X + F(-2X) - \log(2))$$