HAIDER AL SAADI 45 ECTS MASTERS THESIS

NEUTRINO RECONSTRUCTION WITH GRAPH NEURAL NETWORKS

SUPERVISOR: TROELS C. PETERSEN

Abstract

This thesis uses a graph neural network model to reconstruct simulated events in the IceCube Neutrino Observatory by treating samples in the energy range 1-1000 GeV as 4-dimensional graphs with edges determined by 4 nearest neighbors in space coordinates. The GNN model DynedgeEdgepool is developed and tested against the retro_reco algorithms predictions for the regression targets $energy_log10$ and *zenith* angle. The model records a relative improvement in the width of the error distribution to be 25% and 20% for energy and zenith respectively. The model reconstructs events at a rate of 6000 Hz, which is a significant improvement over the retro_reco algorithm, which spends 5-40 seconds per event. These improvements indicate that machine learning models could aid in reconstructing neutrino events and thus aid in determining the physical parameters for neutrino oscillations.

Dedicated to my family and friends.

Contents

1 Introduction

Deep below the Antarctic ice lies the largest currently existing man-made structure. The purpose of this structure, the IceCube neutrino telescope is to detect neutrinos. The detection of neutrinos through this telescope is intended to aid in solving many currently open problems in the fields of physics and astronomy. Currently, we understand very little about the properties of the neutrino. The fact that neutrinos interact only with the weak force is partly to blame, due to that fact making their rate of interaction very low and thus very hard. Neutrino detection systems can act as "cosmic messengers" allowing us map the universe with other means than normal electromagnetic light. For the first time in history, it will be possible to employ all four fundamental forces to explore the universe. Currently, the data from the telescope is reconstructed using a likelihood minimization method that heavily employs table look-up. The accuracy of these reconstructions is fairly decent, but the computational cost is prohibitively expensive, with a single neutrino detection taking 10-40 seconds to reconstruct. This forms a significant barrier to research because the telescope collects raw data at a rate of 2500 Hz. This problem gets further compounded when the algorithm has to be changed to consider different problems, as well as when the algorithm is updated and changes have to be applied to existing processed data. Machine-learning algorithms are known to produce sophisticated outputs at a very fast rate. However, the irregular structure of the IceCube detector and the low signal-to-noise ratio has inhibited the use of ML in IceCube, only applying Boosted Decision Trees for classification purposes quite late in the selection process, using statistical derived features instead of the direct data. Further development has been limited because of the lack of Machine-learning architectures that fit the IceCube data structure. With the advent of Graph Neural Networks, which views data as a graph, we can develop models that reconstruct events with both accuracy and speed. A graph is a data structure that models data as individual data-points connected to other data-points in a matter which can be specified by the creator of the graph. This allows for the modeling of arbitrary data-structures. The intent of this work is to add more weight to the argument that ML has developed to the point where it can be used on IceCube data, an argument which has to a degree been substantiated by previous masters theses GRU and T-CNN models. The model in this work demonstrates a relative change in the width of the error distribution for the reconstruction of $energy_{log10}$ and zenith of 25% and 20% over retro_reco, with world leading precision in the lower ends of the energy spectrum. The model reconstructs events at a pace of 6000 events per second, a significant improvement over current algorithms.

2 Particle physics and the Neutrino

2.1 Motivation

In 2012, experimental researchers at CERN confirmed the existence of the Higgs boson, and with it this all the particles of the standard model were confirmed to exist[1]. There are still many problems remaining in physics which the model can not explain, and some of those involve neutrinos.

[1] ATLAS collaboration. "observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc". https://arxiv.org/abs/1207.7214, July 2012

The neutrinos were initially thought to be massless particles, but evidence instead points to them having such a low mass that implicates them working in a fundamentally different physical manner than the other particles. The observed matter-antimatter asymmetry in the universe can potentially be explained by a charge-parity violating phase in the PMNS matrix.

Neutrinos are observed to oscillate between their three flavors, and this lets us constrain the CP-violating phase, and so the better we are able to measure the oscillations the better our constraint will be. Sterile neutrinos, which interact only with gravity, are a dark matter candidate.

All of these things combined make the neutrino of great interest to physicists, and that is why any information about neutrinos is valuable.

2.2 The Standard model

The standard model is the foundation of modern particle physics research, and describes two types of particles, fermions and bosons and three out of four fundamental interactions [2]. The difference between the two types of particles is that fermions have half-integer spin and bosons have integer spin, thus fermions follow Fermi-Dirac statistics and bosons follow Bose-Einsteins statistics. Fermions make up the matter of the universe and bosons are the force carriers which mediate interactions through the fundamental forces.

[2] B.R Martin. "nuclear and particle physics". Wiley Online books, March 2006

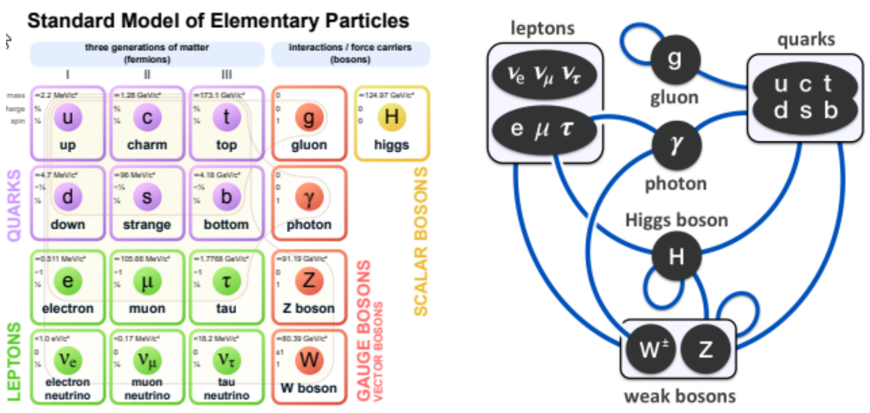


Figure 2.1: The figure on the left shows the particles and the figure on the right shows the possible interactions. Graphic generated by wikimedia

The fermions consist of the leptons and the quarks, which are differentiated by whether or not they interact with the strong force as mediated by the gluon. The gluon acts only on quarks. The gluon is the reason that new particles are created when quarks are pulled apart and why we can only directly detect hadrons(a composite made of two or more quarks).

As seen in figure (2.1)The electron, muon and tau leptons as well as the quarks interact electromagnetically by the photon and weakly by the W and Z bosons. The neutrinos however are special in that they interact only weakly, and this is the reason that they are so hard to spot in detection systems.

The matter particles can be divided into three generations, where each generation differs by their flavour quantum number and mass. The first generation has in it the electron, electron neutrino and the up and down quarks, the second the muon, muon neutrino,charge and strange quark and the third contains the tau, tau-neutrino,top and bottom quarks.

The potential interactions between particles are most easily described using feynman diagrams dependant on Fermi's golden rule, the Lorentz-invariant matrix element and in the case of fermions the Dirac Equation.

Paul Dirac extended the Schrodinger equation to incorporate relativity. Constraining it with the Einstein energy-momentum relation, Dirac found the following quantum mechanical relation:

$$(i\gamma^\mu d_\mu - m)\psi = 0 \tag{2.1}$$

Which requires a four component wave function. This equation governs the behavior of all fermions. In particle physics we collect information about particles and their interactions by either accelerating particles as in the case of LHC, or we try to passively detect particles as in IceCube, and compare our detected particle rates with the ones we expect from our calculations using Fermi's golden rule,

$$\Gamma_{fi} = 2\pi|T_{fi}|^2\rho(E_i) \tag{2.2}$$

Where the left hand side of Eq (2.2) denotes the transition rate from the initial quantum state $|i\rangle$ to the final quantum state $|f\rangle$. The right hand side has the transition matrix T_{fi} gained from expanding the interaction with the Hamiltonian of the perturbation and the energy-dependent density of states ρ. Using this, we can calculate the differential cross section

$$\frac{d\sigma}{d\Omega^*} = \frac{1}{64\pi^2 s}\frac{p_f^*}{p_i^*}|\mathcal{M}_{fi}|^2 \tag{2.3}$$

Which gives us the quantum mechanical probabilities that an interaction will happen. Eq. (2.3) is valid in the center-of-mass (COM) frame as denoted by *, which is the inertial frame where the center of mass remains at the origin. S denotes the squared COM energy, p the momentums of the particles and Ω the solid angle a particle scatters into.

As long as we can calculate $\mathcal{M}$, we can use the standard model to test observable parameters. In the case of fermions we can use Eq. (2.2) to calculate the matrix elements. This process can be quite cumbersome, but fortunately for us we are born in a universe where the great physicist Richard Feynman has provided us with Feynman diagrams. Feynman diagrams are a set of rules by which we can draw interactions and calculate the results from the diagrams, which greatly simplifies the process.

2.3 *Weak interaction*

The neutrino was first posited to fix problems arising from experiments investigating the phenomena of beta decay. In beta decay a neutron turns into a proton, emitting an electron and an anti-neutrino. Without the neutrino in this equation, physicists expected the emitted electron to be mono-energetic, but what they found instead was that the process produced an energy distribution. Secondly, for certain atomic isotopes, the spin could not be conserved if only an electron was emitted. Thus the neutrino was proposed, and with it both of these problems were solved.

The neutrino interacts only with the weak force, and the weak force has the special property that it interacts only with left-handed particles and right handed anti particles. A left handed particle is a particle whose spin is pointing the opposite direction of its momentum, and a right handed particle has its spin pointing along its momentum. Since neutrinos only interact with the weak force and only one "hand" of neutrinos or anti neutrinos interact in such a way, it is possible that there are neutrinos of the other handedness out there that interact only with gravity. These sterile neutrinos could possibly be the explanation for dark matter[3] .

[3] K. N. Abazajian et al. "light sterile neutrinos: A white paper". https://arxiv.org/abs/1204.5379, April 2012

The weak force is mediated by the two charged W^+ and W^- bosons and the neutral Z boson. These bosons have a very large rest mass and thus a very short lifetime. This is what makes the weak force "weak" in comparison to the electromagnetic- and strong forces. The fact that two of the bosons are charged and the third is not is what gives rise to the phenomenon of charged- and neutral currents. The difference between the two is that a charged current allows a charged lepton to turn into a neutral lepton, and it can change the flavor of a quark as well as its electrical charge, while a neutral current leaves the interacting particles quantum numbers unaffected, only transferring momentum, spin and energy. This has experimental consequences since we can only observe charged leptons in our detectors.

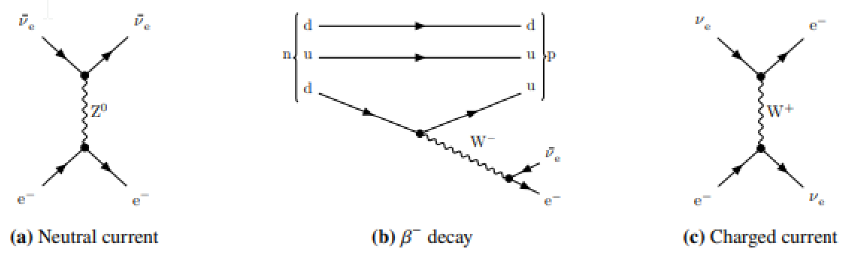


Figure 2.2: Feynman diagrams for beta decay, charged current and neutral current processes.

2.4 *Neutrino oscillations*

2.4.1 *Solar Neutrino problem*

In 1964, a paper was published reporting on a search for solar neutrinos [4]. The premise was quite simple; the sun undergoes a fusion process known as the proton-proton cycle. A product of this process is an electron neutrino, the number of which can be estimated using established physics. A large container filled with 615 tons of perchloroethylene was placed in a closed underground mine, and the solar neutrinos would interact with the chlorine in the tank through the process:

[4] John N. Bahcall. "solar neutrinos. i. theoretical. Physical review Letters page 300-302, March 1964

$$ve + {}^{37}CL \longrightarrow {}^{37}Ar + e^- \tag{2.4}$$

The theoretical prediction was that they should see 1.7 neutrinos per day from the sun, but instead they found 0.48 ± 0.04. This was a surprising result because of the large discrepancy between the predictions and the data. Another experiment was done much later by the SNO collaboration [5], which was able to detect all three types of neutrinos, and this experiment was consistent with predictions that neutrinos would oscillate.

[5] SNO Collaboration. "direct evidence for neutrino flavor transformation from neutral-current interactions in the sudbury neutrino observatory". Physical Review Letters 89.1, June 2002

2.4.2 *PMNS-Matrix*

An explanation for why all three types of neutrinos were detected from the sun had already been provided by Pontecorvo in 1957 [6]; namely that the neutrinos would oscillate between the different types. In 1962 the model was further developed by Maki, Nagakagawa and Sakata [7]. In their model the weak eigenstates of the neutrinos are linear combinations of mass-states who obey the Dirac equation Eq.(2.1). The weak eigenstates are related to the mass states by a 3x3 unitary matrix:

[6] B. Pontecorvo. "direct evidence for neutrino flavor transformation from neutral-current interactions in the sudbury neutrino observatory". `https://inspirehep.net/literature/42736`, October 1957

[7] Masami Nakagawa Ziro Maki and Shoichi Sakata. "remarks on the unified model of elementary particles". Progress of Theoretical Physics 28.5 page 870-880, November 1962

$$\begin{bmatrix} \nu_e \\ \nu_\mu \\ \nu_\tau \end{bmatrix} = \begin{bmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{bmatrix} \begin{bmatrix} \nu 1 \\ \nu 2 \\ \nu 3 \end{bmatrix} \tag{2.5}$$

Named the PMNS-matrix afer the three japanese authors and Pontecorvo. In general a $3x3$ complex matrix has 18 free parameters, but with the condition of unitarity $U^\dagger U = I$, where I is the identity matrix, we can reduce this to 3 mixing angles and 6 complex phases. 5 of which can be absorbed into the lepton states leaving 4 free parameters [8]. The 4 parameters are often chosen as three angles Θ_{ij} and one complex phase δ which violates charge-parity.

[8] J. W. F. Valle. "neutrino physics overview". Journal of Physics: Conference Series 53 page 473-505, November 2006

$$\begin{bmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{bmatrix} \begin{bmatrix} c_{13} & 0 & s_{13}e^{-i\delta} \\ 0 & 1 & 0 \\ s_{13}e^{i\delta} & 0 & c13 \end{bmatrix} \begin{bmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

The solutions for Eq.(2.1) can be written as

$$|\psi(t)\rangle = e^{-iEt}|\psi(0)\rangle \tag{2.7}$$

where ψ is the quantum state, E is the energy of the particle and t is the time at which we evaluate. The evolution of the quantum neutrino state can be written as

$$\psi(t)\rangle = \sum_{i=1}^{3} U_{\alpha i}|\nu_i(t)\rangle \tag{2.8}$$

where alpha denotes the type and the sum is over all the types. To proceed with finding the oscillation probabilities, we have to expand

Eq.(2.8) in the weak eigenbasis. If we invert the unitary matrix U in Eq. (2.5) using the unitarity condition giving us the inverted matrix U^*, and solving for the oscillation probability from state α to state β, we get the following

$$P(\nu_\alpha \rightarrow \nu_\beta) = \sum_{i,j=1}^{3} U_{\alpha i} U^*_{\beta i} U^*_{\alpha j} U_{\beta j} e^{-i(E_i - E_j)t} \tag{2.9}$$

We can see from Eq.(2.9) that if the terms in the exponential differ, oscillations are possible. For our purposes, we can assume the neutrino waves propagate as plane waves with the same momentum, so we let $p_i = p_j = p$ and use Einsteins energy-momentum relation

$$E^2 = p^2 + m^2 \tag{2.10}$$

Which we can use to find the differences in energy using a first order taylor approximation

$$E_i - E_j \approx \frac{m_i^2 - m_j^2}{2p} \tag{2.11}$$

If we assume the neutrinos propagate at the speed of light (not an unfair assumption to make considering their low mass and that they rarely interact), we can set $t = L$, which is the distance the wave has traveled, approximate the energy $E \approx p$ and introduce the term $\Delta m_{ij}^2 = m_i^2 - m_j^2$, we can rewrite the exponential terms in Eq.(2.9)

$$e^{i(E_i - Ej)t} \approx 1 - 2sin^2(\frac{\Delta m_{ij}^2 L}{4E}) + isin(\frac{\Delta m_{ij}^2 L}{2E}) \tag{2.12}$$

After some algebra and replacing the relevant terms in Eq. (2.11) with Eq (2.12), we can calculate the oscillation probability to be

$$P(\nu_\alpha \rightarrow \nu_\beta) = \sigma_{\alpha\beta} - 2\sum_{i,j=1}^{3} Re(U')sin^2(\frac{\Delta m_{ij}^2 L}{4E}) + \sum_{i,j=1}^{3} Im(U')sin(\frac{\Delta m_{ij}^2 L}{2E}) \tag{2.13}$$

where $U' = U_{\alpha i} U^*_{\beta i} U^*_{\alpha j} U_{\beta j}$. From Eq.(2.13) we can see the oscillation frequencies depend on the squared mass differences, their energies, the distance travel and the elements on the PMNS matrix. When detecting neutrinos, we are attempting to indirectly arrive at the values of that equation. We can extend Eq. (2.5) to include sterile neutrinos by simply adding another dimension:

$$\begin{bmatrix} \nu_e \\ \nu_\mu \\ \nu_\tau \\ \nu_s \end{bmatrix} = \begin{bmatrix} U_{e1} & U_{e2} & U_{e3} & U_{e4} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} & U_{\mu 4} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} & U_{\tau 4} \\ U_{s1} & U_{s2} & U_{s3} & U_{s4} \end{bmatrix} \begin{bmatrix} \nu 1 \\ \nu 2 \\ \nu 3 \\ \nu 3 \end{bmatrix} \tag{2.14}$$

We can derive oscillation frequencies the same way as we did before. The only known way of observing sterile neutrinos is through their oscillations. The precise determination of the neutrino oscillation parameters is the most important challenge in the field of neutrino physics. The parameters concerning electron and muon neutrinos are relatively well determined, but the parameters for muon and tau neutrinos are not. In Eq (2.13, the terms in the sine depend on the square of the mass difference and the ratio of the length to the energy of the particle. The mass difference for muon and tau neutrino is much smaller then the difference for the electron and muon neutrino. To observe the oscillation requires a large ratio L/E, and that is where IceCube is hoped to lead the field, since it will be the world leader in detection of low-energy particles with the coming upgraded detection system.

3
IceCube Neutrino Observatory

3.1 Mission

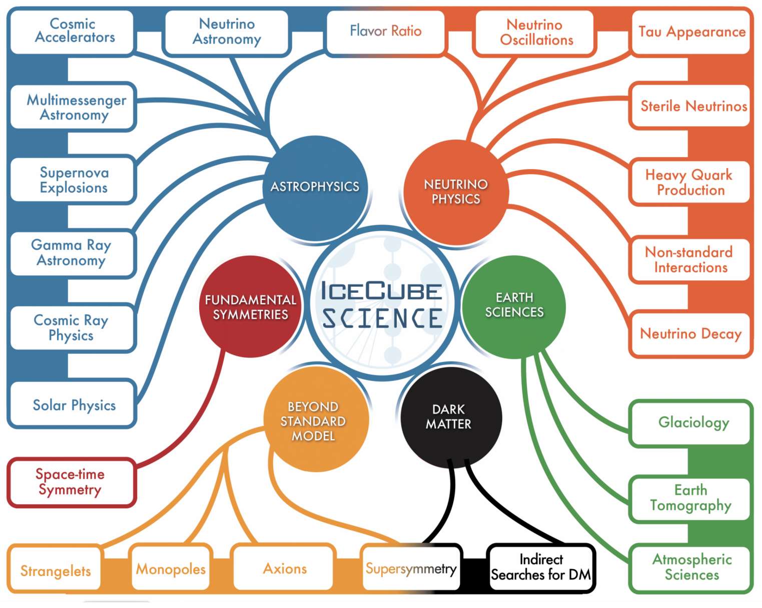


Figure 3.1: An overview over the research in the Icecube collaboration. Graphic by the IceCube collaboration.

The IceCube Collaboration has over 400 physicists and covers a wide range of fields, all of them studying data collected from deep under the antarctic ice. Listed below is an overview of some of the research being done as depicted in figure (3.1)

3.1.1 Cosmic alert system

Because neutrinos have such a uniquely low rate of interaction, their speed in mediums typically exceeds that of light. This is particularly relevant for highly energetic events in space such as gamma ray bursts, where the neutrinos are created and subsequently escape before the light. Thus, on earth, with our neutrino detection systems, we can detect a burst of neutrinos coming from a certain direction and point our telescopes in that direction to see the supernova in its full glory. IceCube is a member of the Supernova Early warning System, which is a collaboration between many neutrino detection experiments all around the world. In this system each experiment records possible cosmic alerts and if these alerts pass a series of quality checks within a 10 second window, a message is sent out to a list of interested astrophysicists who redirect their telescopes [1].

[1] K. Scholberg. "the supernova early warning system". https://arxiv.org/abs/0803.0531, March 2008

3.1.2 *Mapping the universe*

Similar to how we use light to get a picture of the universe, we can also use neutrinos to understand the structure of the universe better. By reconstructing the angles from which neutrinos come from and measuring their flux over a long period of time, we can hopefully detect emission sources such as gamma-ray bursts, which are extremely highly energetic bursts that last at the most a few hours, and are thought to be extremely rare. For a trivial example, it is possible to get a picture of the moon using the small deficit in events coming from that direction. This can also be used for angular calibration of muons.

3.1.3 *Neutrino oscillations*

In chapter 2, we ended up with Eq 2.13 to describe the oscillation frequencies. These frequencies depend on parameters we are very interested in, and in IceCube we determine these neutrino mixing parameters by observing patterns in the atmospheric neutrino flux created by the oscillation. The atmospheric neutrino flux is created by the interaction of cosmic rays with our atmosphere, creating neutrinos that then oscillate into other states. Icecube has been able to measure neutrinos with energies as low as 5 GeV [2]. The low energy regime is particularly interesting since it lets us observe muon neutrino disappearance(it's oscillation into different types) over a range of distances up to the diameter of the earth.

[2] M. G. Aartsen et al. "measurement of atmospheric neutrino oscillations at 6– 56 gev with icecube deepcore". https://arxiv.org/abs/0803.0531, February 2018

3.1.4 *Searching for the unknown*

While neutrinos are interesting, IceCube constitutes the largest detector volume ever considered. It is therefore interesting to consider, if other types of interactions beyond those of the Standard Model can be observed in the detector. To that end there are teams working on strange and unexplained phenomena that can not be attributed to noise.

3.2 *Current detection system*

The basic unit of the detection system is the DOM(digital optical module). The digital optical module is a device that uses photomultiplier tubes (PMT) to detect the light stemming from Cherenkov radiation. The system has 5160 DOM's on 86 vertical strings. These strings are placed in a hexagonal structure with 125 meters between them, and the DOM's in the strings are placed 17 meters apart. The entire system has a volume of about one kilometer, with a specialized smaller array called Deepcore in its center. The overarching structure is hexagonal in shape, which means that it is not rotationally invariant.

The DOMS are all located between 1450 and 2450 meters below the surface, where the ice is clearest, with no DOM's between 2000 and 2100 meters due to the existence of a region with low clarity referred to as the dust layer. The deepcore section consists of 6 strings in the center of the detector at a depth starting about 2100 meters. Above the dust layer deepcore strings have 10 DOM's spaced 10 meters apart and under the dust layer, they have 50 special high quantum efficiency (HQE) DOM's spaced 7 meters apart. The deepcore section is the section that allows us to detect the lowest energy neutrinos, because of the better DOMs and tighter spacing, as well as the employment of non-deepcore sections to filter unwanted background (mainly from muons).

Far above the rest of the sections and on top of the ice, a pair of frozen water tanks are installed close to each individual string, with each tank containing two DOM's spaced 58 cm apart. This is known as IceTop and is used to study cosmic muons as well as inform the rest of the detector that an event potentially stems from a cosmic muon.

3.3 *IceCube Upgrade*

Sometime in the year 2022/2023 the detection system will be upgraded. The upgrade consists of 7 strings which will have a total of 700 detector modules. The upgrade will have 3 types of detectors; Ordinary DOM's, Multi-PMT DOM's (mDOM's) and Dual Optical modules(D-eggs). An mDOM has 24 PMTs evenly spaced throughout it's spherical shape. This gives us a better a sense of the direction of the pulses. D-eggs consists of Two HQE DOMs, one pointing up and one pointing down, along with 12 LED's installed for calibration purposes. This is referred to as the IceCube Upgrade and i will refer to the data from there as Upgrade Data. The intent behind Upgrade is that it will increase the resolution in the low energy regime and make

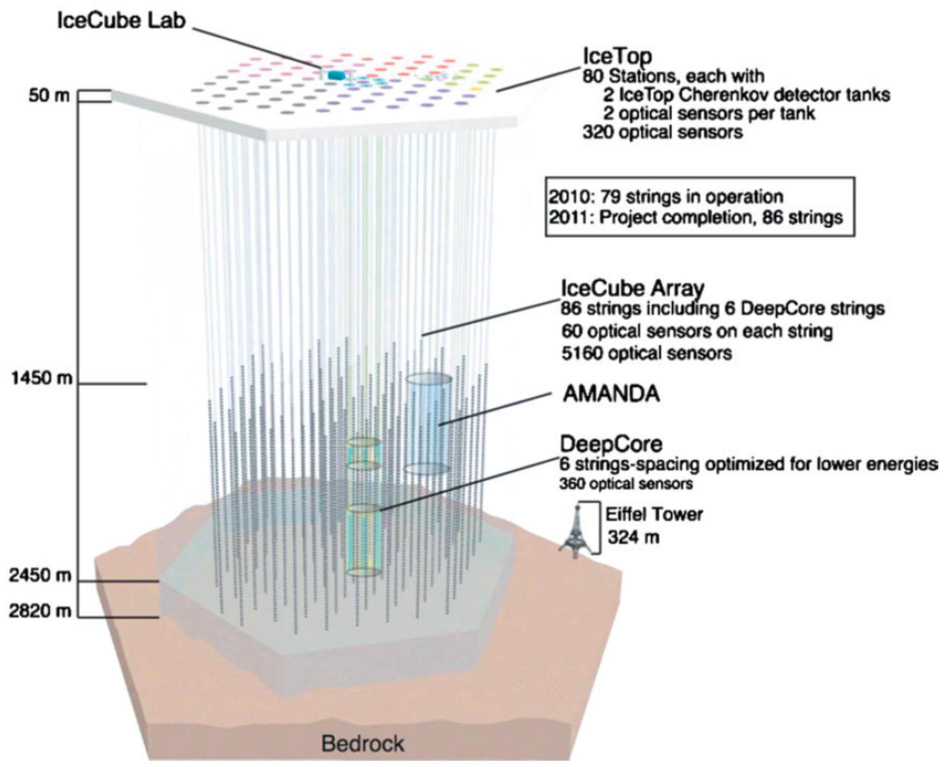

IceCube Lab
IceTop
80 Stations, each with
2 IceTop Cherenkov detector tanks
2 optical sensors per tank
320 optical sensors
2010: 79 strings in operation
2011: Project completion, 86 strings
IceCube Array
86 strings including 6 DeepCore strings
60 optical sensors on each string
5160 optical sensors
AMANDA
DeepCore
6 strings-spacing optimized for lower energies
360 optical sensors
Eiffel Tower
324 m
50 m
1450 m
2450 m
2820 m
Bedrock

Figure 3.2: The Detection system. The strings are indicated in grey with black boxes on them indicating DOM's. Graphic generated by Wikimedia

it easier to calibrate the system/detectors. Upgrade is the first step towards IceCube Gen2, which is a large expansion that will double the amount of DOMs in the array[3].

[3] Aya Ishihara (for the IceCube Collaboration). "the icecube upgrade – design and science goals e". https://arxiv.org/abs/1908.09441, August 2019

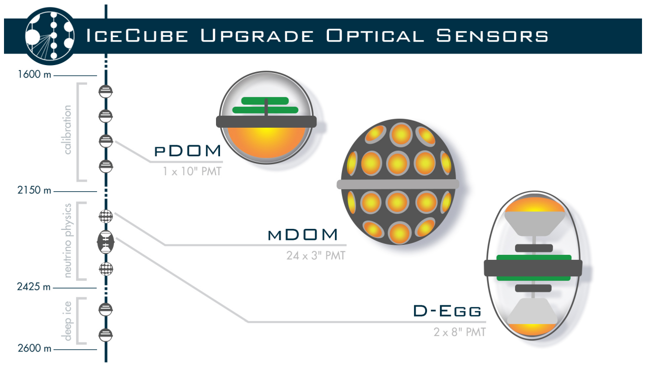


Figure 3.3: The different types of DOMs in IceCube Upgrade and their placement. Graphic from https://icecube.wisc.edu/gallery/nsf-approves-funding-for-icecube-upgrade/

4
Icecube Data

The data that the IceCube collaboration has stems from two sources. The first is data that comes from the detector array, and the second is simulated data. Both of these are catalogued in the file format .i3, which has been developed for Icecube. This file format can be read using Icetray, which is an environment developed for the file format. The data is structured like an ordered dictionary, wherein a principal key corresponds to either a particle attribute or a given detectors output in a feature. Luckily, previous students have gone through the trouble of dealing with this file format and created pipelines allowing me to deal with databases which accept SQL queries(in python SQlite), thus removing the need for a deep dive into the specifics. The databases in this work have been created through Mads Ehrhorns pipeline[1]. As the goal of this thesis is to improve upon the current algorithm Retro-Reco, only simulated data will be considered, as this is the data which allows us to directly compare performance.

[1] Mads Ehrhorn Kjær. "convolutional neural network neutrino reconstruction in icecube", December 2020

4.1 Simulation data

The simulated data has been generated by Monte Carlo simulations developed and implemented by various people and groups in Icecube. This class of simulation algorithms is used because it simulates events probabilistically, which is necessary for interactions that are probabilistic in nature. The simulations themselves are very complex and are the results of extensive development, and so their internal mechanics and ideas will not be expounded upon. What they have in common is their overarching goal. What they seek is to simulate is what happens in the detector array for a given particle or event, or more broadly, what happens when a specific configuration for a generative process of particles interacts with the detector. The simulations model not only the particle interactions from a theoretical particle physics perspective, but also models environmental and experimental details that are crucial to ensure accuracy, such as the properties of the ice, detector efficiences and the existence of the dust layer. They also model the noise inherent in the detectors, which regularly produce readout even with no particles present. For this work only the event generator GENIE [2] has been used, which simulates the interaction between neutrinos and ice molecules.

[2] C. Andreopoulos et al.). "the genie neutrino monte carlo generator". https://www.sciencedirect.com/science/article/abs/pii/S0168900209023043?via%3Dihub, February 2010

To get a full picture of how to deal with the real data it would be necessary to use many simulators, since they typically deal with different sources of signal. Furthermore, different simulations exist for the standard Icecube setup and the Upgrade setup, which have different entries in their databases. The SQlite databases are structured around events stemming from a single particle, which is not always the case in the real array.

4.2 *Events*

The SQLITE databases each consist of two tables,features and truth. The truth table is structured such that each event has one row in it, with an amount of columns equal to the reconstructable attributes of the particle, such as energy, particle type, interaction vertex etc. The feature table consists of pulses, which is a read-out from the detector containing such things as time of detection, placement of DOM and other attributes of the DOMs. The features have a variable amount of rows for each event with an entry in each row corresponding to the event number, an unique identifier for that event(unique only for a given simulation run, not across simulations!).

Feature	Description	Current	Upgrade
event_no	event number	✓	✓
string	string number	✓	✓
dom	DOM number	✓	✓
dom_x	x-position	✓	✓
dom_y	y-position	✓	✓
dom_z	z-position	✓	✓
time	time-position	✓	✓
charge	measured charge	✓	✓
lc	local coincidence	✓	✓
pulse_width	the width of the pulse	✓	✓
SplitInIcePulses	If it comes from a split pulse	✓	✓
SRTInIcePulses	Whether or not it survives SRT	✓	✓
pmt_x	x-component of vector		✓
pmt_y	y-component of vector		✓
pmt_z	z-component of vector		✓
pmt_area	surface area of PMT		✓
pmt_type	type of DOM		✓

Table 4.1: Table of the column names in the feature tables of the SQlite databases.

Going through the entries in table (4.1), *event_no* is the previously mentioned event number, string refers to which string the detector is attached, *dom* refers to the individual DOM on a given string, *dom_xyz* are numbers that refers to their spatial coordinates in the detector, *time* refers to its temporal placement in the given window of time it records a signal, *charge* refers to the mean value of the charge waveform when fitted to a gaussian, *lc* stands for Local coincidence, a boolean that signifies that a pulse close in time and space has also been detected, *pulse_width* denotes the uncertainty on the recording in nanoseconds, *SplitInIcePulses* denotes whether or not a given waveform signal has been split into two pulses, *SRTInIcePulses* denotes whether or not a pulse survives the SRT cleaning algorithm, *pmt_xyz* is a vector denoting the individual PMT's relative position on the new DOM types D-egg and mDOM, *pmt_area* denotes the area of the DOM and finally *pmt_type* denotes which type of DOM it belongs to.

Truth	Description
event_no	event number
energy_log10	base 10 logarithm to the energy in GeV
position_x	x-position of interaction vertex
position_y	y-position of interaction vertex
position_z	z-position of interaction vertex
direction_x	x-component of particle path
direction_y	y-component of particle path
direction_z	z-component of particle path
azimuth	azimuth angle of interaction vertex
zenith	zenith angle of interaction vertex
pid	particle type
stopped_muon	whether or not the muon stopped in the detector
muon_tracklength	length of the path of muon

Table 4.2: Table of column names in the truth tables of the SQlite databases

Going through each entry in table (4.2), we start with the same identifier *event_no* as in the feature table, then we have the true energy of the particle given in GeV and transformed with base 10 logarithm called *energy_log10*. We have the three coordinates of the interaction vertex *position_xyz* and the three components of the direction of the particles velocity direction_xyz. Applying a spherical coordinate transformation we can get the two angles *azimuth* and *zenith*. The *PID* of the particle denotes which type of particle gave rise to the event and follows the particle numbering convention [3]. The last two entries only apply if the instigating particle is a muon, with *stopped_muon* whether or not the particle stopped inside the detector or continued through it and *muon_tracklength* denoting the length of the track that the muon scored through the detector.

[3] T. Sjostrand L. Garren, I.G. Knowles and T. Trippe). "monte carlo particle numbering scheme". https://link.springer.com/article/10.1007/BF02683426, January 2000

4.3 *Data cleaning*

The Icecube detectors are constantly recording tons of data, a large majority of which does not stem from any particle interactions. To be able to analyze this data, we make several selections and discard a lot of detections. We can consider these cleanings to be organized into 7 levels. The data selection comes from the OscNext team [4]. The extensive cleaning and selection processes are necessary due to the difficulty of reconstructing events with low signal-to-noise ratios and the long reconstruction time.

[4] oscNext team. "oscnext". https://wiki.icecube.wisc.edu/images/8/82/OscNext_v00.04.pdf, April 2020

4.3.1 *Level 1 cleaning*

The first cleaning occurs on the raw data. Data that passes this level must pass a single multiplicity trigger Check(SMT). This check labels a detection as an event if several detectors also trigger within a short window of time.

4.3.2 *Level 2*

The first step on this level is to discard all events that do not have more than 3 SMT's. Then the SRT algorithm is run on this selection, wherein events that do not survive are discarded. SRT stands for Seeded Radius-time, and it is an algorithm that starts at pulses that survive the local coincidence condition(another pulse close by in time and space) which acts as the "seed", and then classifying other pulses within 150 meters and 1 nanosecond (hence Radius-Time) as also surviving SRT-cleaning. These newly added pulses are then used as seeds to further expand the collection of SRT-surviving pulses, ending only when there are no more pulses within the correct Radius and time of already surviving pulses. This essentially acts as a causality check, using pulses we are more sure about to discard pulses that can't be related to them. Furthermore, since certain events(depending on their particulars) must have both detections in Deepcore and the rest of the detector, we can use a lack of detection outside Deepcore to act as a veto. Currently all Upgrade Data is at most cleaned to this level.

4.3.3 *Level 3*

In this level cuts are made across entire events using derived features that allow us to remove events which are exceptionally different from real data. The exact features can be found in OscNext.

4.3.4 *Level 4*

After data survives level 3 it has a decent resemblance to real data, and this justifies using the machine learning algorithm LightGBM, a Boosted Decision Tree (BDT) algorithm, to classify events into either Muon events, Pure noise events and Events which are neutrino candidates.

4.3.5 *Level 5*

Due to the hexagonal structure of the detector arrays, muons can occasionally pass through "corridors" in the detector array and thus survive up to this level. At this level all events that do not exist in the Deepcore section are removed, and an additional cut is made to decrease the amount of muons passing through corridors.

4.3.6 *Level 6*

At this level the retro-Reco algorithm is run on the high statistics area of the level 5 data. The Retro-Reco calculates likelihoods and uses a table look-up method to do it's regression/classifications, timing out if convergence is not reached within 10 iterations. This leaves some event/ event targets unreconstructed.

4.3.7 *level 7*

The level 6 data is run through machine learning algorithms (BDT) for the same purposes as in level 4 but this time on the data from level 6, and final results come from this level.

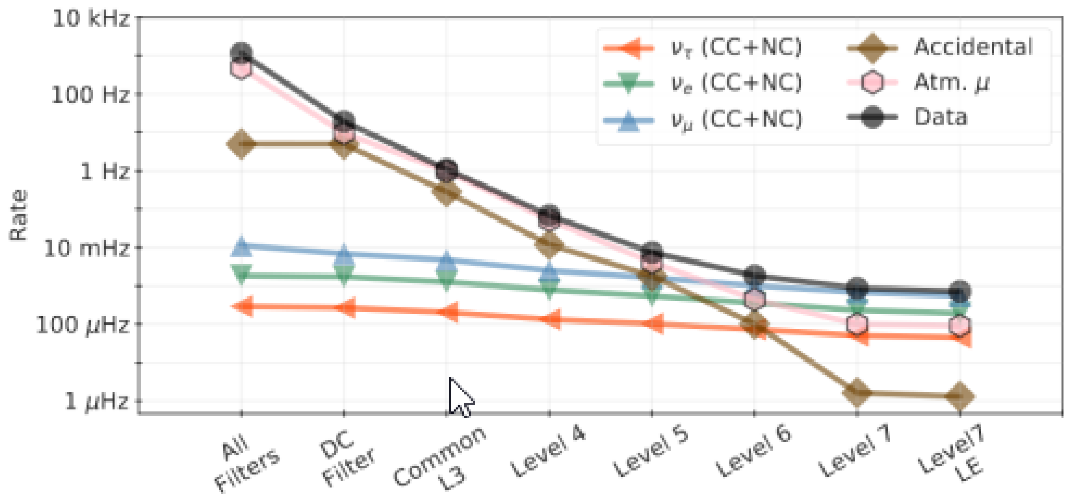


Figure 4.1: Event rate comparison across filters. All filters is the general level 2 filter used in the collaboration, DC is the level 2 Deepcore filter described above.

As seen in figure (4.1) borrowed from a paper by the IceCube collaboration [5], each successive cleaning level significantly increases the fraction of signal. What is not indicated on the figure is the amount of pulses that are cut away by things like the SRT-cleaning, but these are not that important since the precision of the data is proportional to $\sqrt{N}$, where N is the amount of pulses, and we typically dont significantly impact the magnitude of the amount of pulses.

[5] M.G AArtsen et al. "measurement of atmospheric tau neutrino appearance with icecube deepcore ". `https://journals.aps.org/prd/abstract/10.1103/PhysRevD.99.032007`, February 2019

5
Deep Learning

5.1 The types of machine learning and their uses

In earlier decades, most of the work using computers has been focused on developing and optimizing algorithms that solve equations and provide mathematically rigorous estimates , and for a long time , this was the best and perhaps only way to efficiently use digital technology. However, with the incredible advancements in computing power, data storage and data collection, a lot of attention is being given instead to methods and algorithms that rely on setting up a system which by itself learns underlying patterns in the data. These methods are known as Machine learning or Deep learning, named so because of the extensive use of machine intelligence or as a reference to deep neural networks, which are networks designed to emulate the human brain using many layers of artificial neurons. Deep learning can be most generally thought of as consisting of three different types, all structurally different in their approach and thus better suited for different problem setups. Furthermore, these subtypes can be further divided into more sub-types and this process of sub-division can continue for quite a while. For brevity (and because i found a neat image) i will not descend lower than three levels.

The first is termed Reinforcement Learning. In Reinforcement Learning, score functions or KPI, key performance indicators, are constructed such that the algorithm is rewarded when it performs well and punished when it performs poorly. An example of this could be to create an AI that plays a competitive video game, where the AI is rewarded for the number of enemies it kills or the time it stays alive. It could also be a robot with that learns to traverse a landscape efficiently that is punished for falling over. The interplay between data and machine here is that the machine calculates a state from past and present data, uses the state to calculate an output that produces a new state, and then evaluates the movement from one state to the other. This type is not well-suited to the problem at hand.

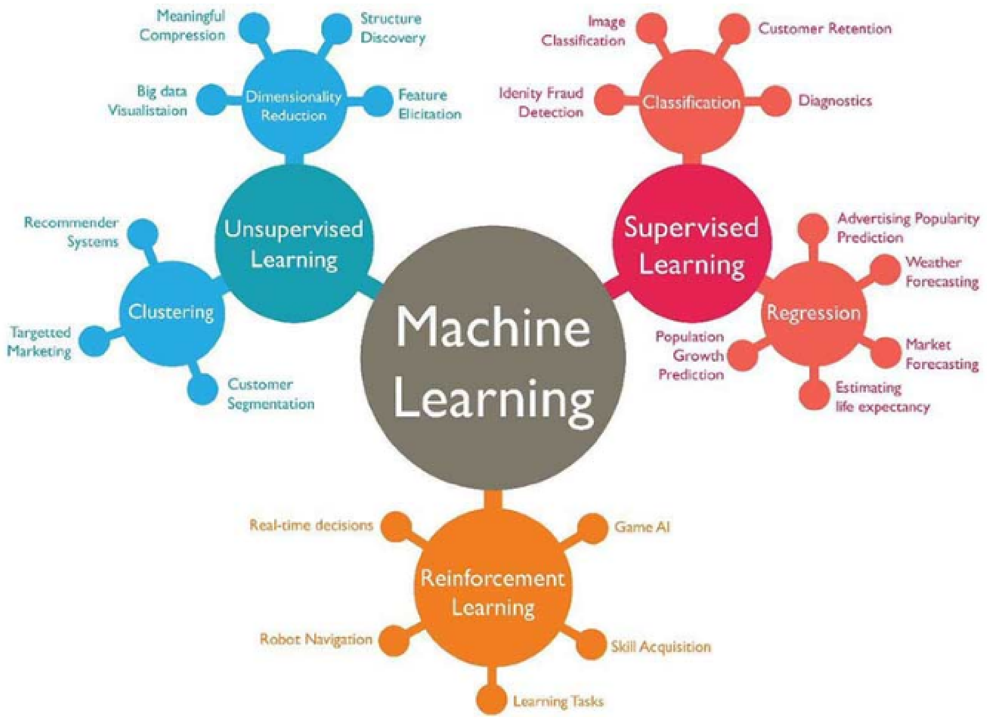


Figure 5.1: The most general types of machine learning. Graphic generated by Vincent Granville https://www.datasciencecentral.com/profiles/ of-machine-learning-algorithms-in-one-picture

The second type is termed Unsupervised Learning. With these methods the machine is fed data with no labels, meaning data that has no truth values, and the machine is then asked to perform some sort of operation. For example, upon being fed a data-set consisting of geological data from the earth and the moon, such a machine can be asked to perform a clustering algorithm to find two clusters, and hopefully those two clusters will neatly separate the earth data from the moon data. A potential use for IceCube could be attempting to separate data from noise.

Lastly, but certainly not least, is Supervised Learning. In Supervised learning, the machine is fed labelled data, and then asked to predict the labels for the data. The two sub-types of this is regression, wherein a value is predicted, and classification, wherein the data is divided into different classes and the machine is asked to predict which class each part of the data belongs to. For reconstruction, this could be regressing on any number of features of the event, such as energy, location and angle etc. or classifying which type of particle resulted in the event. There are many different types of supervised learning algorithms such as decision trees, genetic algorithm , support-vector machines etc. but we will only limit ourselves to neural networks.

5.2 *Neural Networks*

The basic component of artificial neural networks is the neuron. The Neuron takes in N inputs, applies a weight w_i to each of them, adds them together and adds a bias b, after which it applies an activation function σ. The final result is then z, the output of the neuron.

$$z_i = \sigma(b_i + \Sigma w_i x_i) \tag{5.1}$$

Where x_i is the input and $b_i w_i$ are learnable parameters. A neural network consists of multiple connected neurons, where one neuron's output gets forwarded as the input to another neuron. This is where the term feedforward and forward pass originates from. This gives rise to the fully-connected feedforward network, which is a network consisting of layers of unconnected neurons that feedforward to each of the neurons in the next layer. In this setup, each neuron j from layer k $z_j k$ learns a set of parameters and a bias. The input to the first layer comes from the data as such:

The amount of functions a network can represent generally increases faster. for an example see the following:

Rongjie Lai Feng-Lei Fan and Ge Wang. Quasi-equivalence of width and depth of neural networks. https://assets.researchsquare.com/files/rs-92324/v1_stamped.pdf?c=1603402581, October 2020

$$z_{i0} = \sigma(b_{i0} + \Sigma w_{i0} x_i) \tag{5.2}$$

and the following layers treat the outputs of the previous layer as inputs:

$$z_i k = \sigma(b_i k + \Sigma w_i k z_{ik-1}) \tag{5.3}$$

The final layer acts as the output layer. By convention, when referring to the first layer, it is not a reference to the layer described by 5.2 but rather the set of neurons who contain the input information and are thus termed the input layer. The layers between this first layer and the final output layer are known as the hidden layers, since the states of the individual neurons and their weights are difficult if not impossible to analyze and thus almost never inspected.

The forward pass describes how the network generates an output from an input. In the case of labelled data, we know what the output should be, but initially the network will generate an effectively random output. The difference between the generated output and the output we believe the input data should provide is known as the *error*. To improve the networks capability to accurately predict the output, we need to update the networks weights and biases in a corrective manner. This updating process is known as training the network and the method used is called backpropagation.

5.3 *Backpropagation*

In the previous section i mentioned that we train the network to minimize the error. Formally, if we consider the input data to be a set

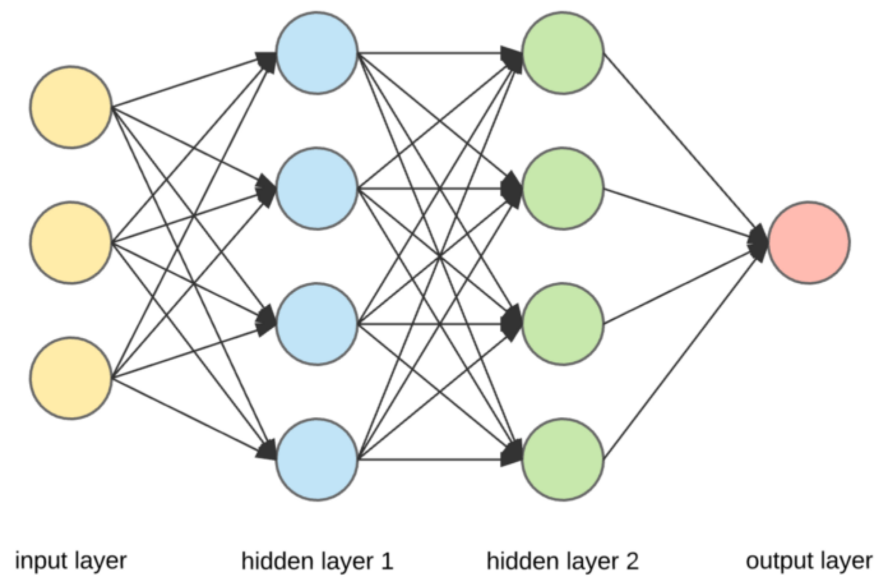


Figure 5.2: Diagram of fully connected feedforward network. Graphic generated by Arden Dertat https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

of examples of the form $(x_1, y_1)....(x_N, y_N$, where x_i denotes the set of variables forming the input data , known as features, y_i denotes the label and N is the amount of training example , we can view the entire network as a function $g : X- > Y$, where X is the (input space) and Y is (output space). Each function g is an element of G, which is the set of possible functions a given network configuration can become by applying different weights and biases known as the *hypothesis space*. The error between the output and the label is transformed according to a *loss* function $f(g(x)|y)- > R$, where R is the set of real numbers. The objective of training is then to find the g that minimizes $f(g(X), Y)$, where we are using the joint probability model for the loss functions transformation. The key to traversing the hypothesis space is to move in the opposite direction of the gradient with regards to f.This method is known as gradient descent. Given an example network g with L layers and a training example (x_i, y_i) , we apply the forward pass and end up with the error ϵ in the output layer with the loss $f(\epsilon, y_i)$. We can now calculate the error for each neuron j in the output layer

Optimizing for the traversal of hypothesis space can be quite difficult and is indeed why loss functions are necessary. More on this in later sections

$$\delta_{Lj} = \frac{\partial f(\epsilon, y_i)}{\partial z_{Lj}} z'_{Lj}(c_{Lj}) \tag{5.4}$$

where z_{Lj} is the previously mentioned output, c_{lj} is the weighted input before σ is applied to it, $\frac{\partial f(\epsilon, y_i)}{\partial z_{Lj}}$ denotes the partial derivative of the loss function with regards to the activation function,$z'_{Lj}(c_{Lj})$ is the derivative of z_{Lj} with regards to c_{Lj} and δ_L is the error for the neuron in the layer. We can rewrite this in matrix form using element-wise matrix multiplication known as the Hadamard product $\odot$.

$$\delta_L = \nabla_z f(\epsilon, y_i) \odot z_L^{'}(c_L) \tag{5.5}$$

where $\nabla_z f(epsilon, y_i)$ is a vector whose components are the partial derivatives of the loss function with regards to each individual activation z. δ_L is the vector containing each neurons error and is equivalent to δ_{Lj} from 5.4.

We can then propagate this error back to the layer $L-1$ and from there to $L-2$ and so on until we reach the first layer. This is where the term backpropagation stems from. In general, the error vector for layer k is given by:

$$\delta_k = ((w_{k+1}^T)\delta_{l+1}) \odot z_{c_k}^{'} \tag{5.6}$$

where w_{k+1}^T is the transposed weight matrix for layer $k+1$. Now that we have the errors for each neuron, we can find the gradient of the bias which is simply:

$$\frac{f(\epsilon, y_i)}{b_{kj}} = \delta_{kj} \tag{5.7}$$

which lets us apply a update to the bias:

$$b_{kj}^{update} = b_{kj} - \alpha \frac{f(\epsilon, y_i)}{b_{kj}} \tag{5.8}$$

Where α is the *learning rate* parameter, which is a scaling for the correction. For the gradient of the weights we have the following:

$$\frac{f(\epsilon, y_i)}{w_{kjv}} = \delta_{kj} z_{(k-1)jv} \tag{5.9}$$

where w_{kjv} is the weight going between neuron k_j and k_{-1j}. $z_{(k-1)jv}$ is the activated output that gets passed through to w_{kjv}. To correct the weights we apply:

$$w_{kjv}^{update} = w_{kjv} - \alpha \frac{f(\epsilon, y_i)}{w_{kjv}} \tag{5.10}$$

We have now arrived at how to update the network given a single training example. When this process is applied to an entire set of training data, it is referred to as a training epoch.We want to make full use of our data, and the naive way to proceed would be to calculate the changes made to the weights and biases over an epoch running through the entire data set and then apply them. One of the reasons this is not done is that is is computationally much less expensive to run through training epochs to convergence if we update as soon as we have calculated a change in parameters. This is usually done in batches of examples, where the amount of examples in a given batch is called the *batchsize*. The principal reason the naive way is undesirable is that the intention behind any predictive model is not to perfectly fit a given labelled data set(since this can be done with any sufficiently complex model), but to be able to generalize and correctly predict unseen and unlabelled data. To aid in this cause, we need to further analyze the data, the hypothesis space and the traversal of the hypothesis space.

5.4 *Data as Probability distribution*

I previously mentioned that we are using a joint probability model for the loss functions transformation. For the data we are using a conditional probability model. The conditional probability model comes from treating the data as arising from an underlying distribution of features which give rise to different labels. For classification, we are assuming that the distribution of the data is a function of different and potentially overlapping distributions. For an example, consider a data set consisting of the height in meters of adult males, where some of the data is labelled as coming from the Netherlands and the other is labelled as coming from India. We know that the underlying distributions from the two countries are gaussian functions, one with a mean value of 183*cm* and the other has a mean value 165*cm*.

An assumption is also made that the data set is large and varied enough to represent the two distributions.

The data set itself will then look like two gaussian peaks with

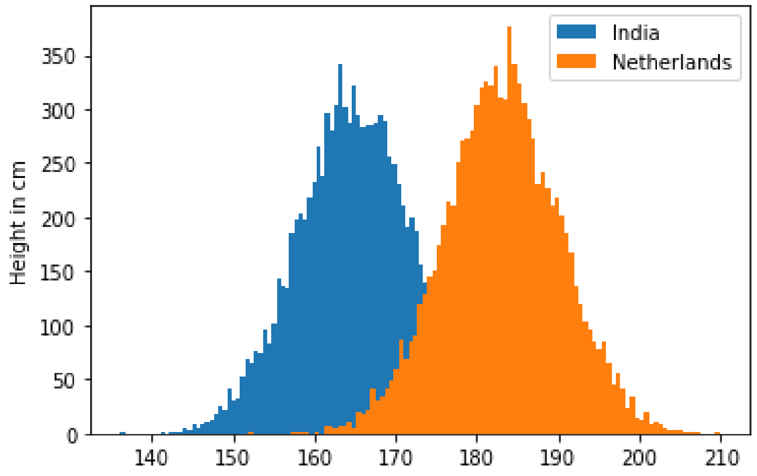


Figure 5.3: Histogram of fictional height values for Netherlands and India.

some overlap as in figure 5.3.A neural network trained to classify these heights as either stemming from the Netherlands or India predicting with certainty, that is giving either the label Netherlands or India would be mathematically unsound, since the information is simply not present in the data itself. For an example, consider a point where the height is 173*cm*. This height could very easily be either an adult male from either country. Therefore, the network has to be necessarily trained to predict the probability with which each height stems from one of the given distributions. In the case of regression, the picture is slightly muddled. In our assumptions up until this point, we have not yet dealt with the problem of random error or "noise". If we were to regress on data wherein each set of features only leads to one specific label, i.e the input space forms a bijective mapping to output space, it would be quite simple to train a network. Unfortunately this is almost never the case in machine learning, and for IceCube it certainly isn't. The noise is perhaps the most significant aspect when considering the theoretical capacity for any network to generalize to unseen data, and as previously mentioned, providing a general model for unseen data is the entire point!

5.5 Generalization error

The most general purpose of training a model is to generalize to data that the model has not been trained on. It can be shown [1] that the more ways a network can represent a given set of data, the higher the generalization error becomes. This is known as the bias-variance tradeoff. High-variance models can represent the variance in the data well, but are at the mercy of noisy or unrepresentative data.This is typically known as overfitting to the training data. High-bias models can whisk away the noise, but faulty assumptions can lead to not capturing the real variance in the data. This is typically known as underfitting to the training data. It is easiest to think in terms of this trade-off by analyzing where we inject bias into the model, since we generally introduce variance by changing the various ways we introduce bias. The most general ways we introduce bias are by how we traverse the data set, the choice of loss functions, choice of optimizer and network configuration.

[1] Wassily Hoeffding. "probability inequalities for sums of bounded random variables". `https://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500830`, 2012

5.6 Train-test split and validation

Data sets are usually constructed in a manner according to the data collection process, and this can bias the model towards this collection process. The first step is therefore randomly shuffle the training examples. Then we split the data into two sets, a training set and a test set, where the test set is not used when training the model or tuning the model parameters. The size of the split here is what determines the bias-variance tradeoff. The larger the training set is in relation to the test set, the larger the bias, and inversely, a larger test set increases variance and decreases bias.

A further splitting of the training set into a training set and validation set can also be made. This can give us an early estimate of the viability of the model and allow for easier fine-tuning of model parameters. After using these sets to generate the model architecture and being sufficiently sure of its generalization ability, we can then train and evaluate the model on the complete data set. For the analysis of the validation set it is typically sufficient to use the average value of the loss function, but to determine the general viability of a model we need to introduce metrics by which we can evaluate it, but first we need to understand how the biases inherent in the ways we use the training data affect our models.

5.7 *Training on data*

The way we traverse the data set is one of the most important sources of biases possible in a neural network, since it fundamentally determines the structure of the patterns the network is taught to discern. Furthermore, the choices we make here also determine the type of networks we can employ for meaningful results. In the previous section i mentioned that we randomly shuffle the training examples so as to not unduly bias towards the data collection process. While this process would be sufficient in the formal example of data of the type $(x_1, y_1)....(x_N, y_N)$, where x_i is a vector of features in a training example, this is not the structure of the data we have in Icecube. Each of our training examples in Icecube consists of N pulses with M features in them, where the example itself is given a single label y, and the way we iterate through these pulses is a source of bias. The neural network structure discussed in previous sections would be somewhat incompatible with our data, since the label is not associated with any one pulse but with the entire training example. Previous work has had some success viewing the examples as a time ordered series of pulses, and iterating through them using something called a Gated Recurrent Unit, as per Bjørn Mølvig's masters thesis, or viewing the time ordered series as an image and applying a temporal convolutional kernel to the image as per Mads Ehrhorn's masters thesis. In this thesis i have used the representation of data as a graph and applied graph neural network operations to it, both of which will be expanded upon in the next chapter. The choices made with regards to the data set determine which general network architectures and operators we can employ. As previously mentioned, we wish to run the model training process to convergence, by which we mean that we want to train the model until it is not capable of becoming more accurate by changing its weights and biases. This training process is determined largely by the traversal of hypothesis space. Since the aim is to minimize the error which stems from a given point in hypothesis space, we can design a transformation of the error space by the use of loss functions into a loss space, which combined with the network architecture gives us the loss landscape. We traverse the loss landscape using the backpropagation method with the free parameter called the learning rate. Before we can make our choices regarding the learning rate, we need to understand the loss landscape and how it determines the viability of any general model.

5.8 Loss landscapes

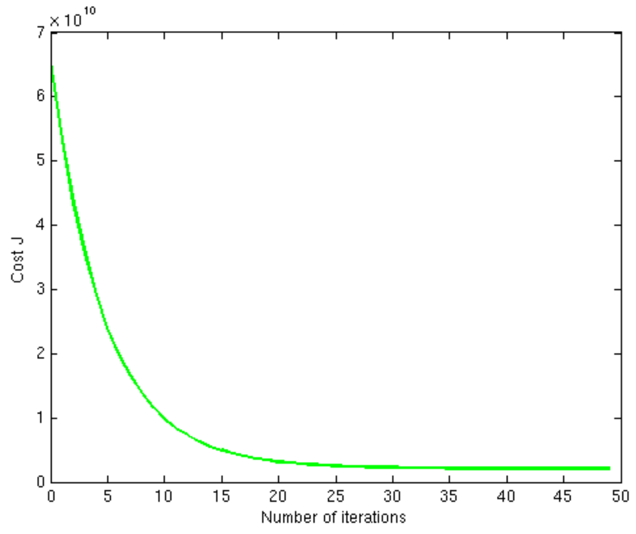


Figure 5.4: An example of the convergence process. The model could be considered to have converged around 25-30 iterations.

We run training epochs until the model has converged. If we have a well designed loss landscape, the average loss as a function of the number of training epochs will look something like figure (5.4):

We understand the model to have converged somewhere at the right-hand tail of the plot, where it either does not significantly change the loss upon training another epoch or it oscillates around some central loss value. We usually use *early stopping* to determine which point in hypothesis we use, wherein we stop the training process once there has been no improvement for a set number of epochs.

Figure (5.4) can be hard to replicate, since loss landscapes are generally highly non-convex, where the complexity of the landscape usually rises very quickly with the number of parameters in the architecture. The Cost is often used as a synonym for Loss.

The loss landscape is determined by not only the network architecture, but also by the choice of loss function. Which loss function is chosen is crucial, since the properties of the loss function transfer

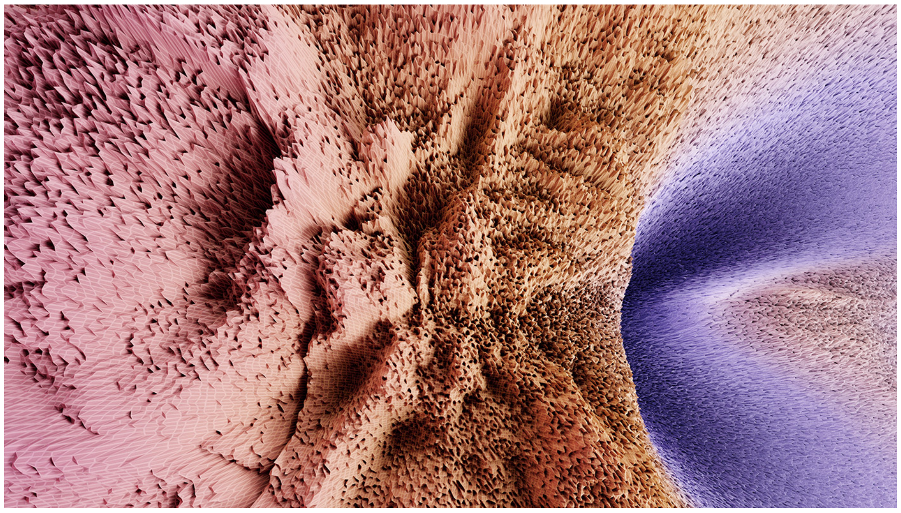


Figure 5.5: An example of a non-convex landscape. Height and color indicates the value of the loss function, where the blue area returns the lowest value. Image generated by varying two weights in a model with more than 10000 weights. Image from losslandscape.com

onto the properties of landscapes that is derived from it. Typically the loss landscapes of ML models are highly complex and non-convex and a example of such a loss landscape is displayed in fig (5.5).

5.9 *Loss functions and labels*

The properties that we are seeking in the loss function depend on the properties of the labels we are aiming at predicting and will also determine the type of optimization the network will be trained for. For an example, consider a data set where the labels can differ in magnitude by dozens or hundreds of orders. If we were to attempt to model this purely by using the absolute error as our loss function, the network would be heavily weighted towards representing the higher end of the distribution well and ignoring the lower end. That is why we must carefully consider which loss functions we use with regards to a given goal. A common goal for all loss functions is the width of the error distribution:

$$w(\epsilon) = IQR(\epsilon) \tag{5.11}$$

Where IQR is the interquartile range.The inter-quartile range is the difference between two quartiles, where a quartile is a value that splits the data set in two according to a percentile. The percentile for the first quartile we will be using is 16, that is the value at which 16% of the lowest errors lie beneath. The percentile for the second quartile is 84, the value at which 84% of the errors lie beneath.These percentiles are chosen to simulate a gaussian distribution.

5.9.1 *Energy regression*

The ultimate goal of energy regression is to minimize the error, and the quantity we are most interested in is the relative error in the following form:

$$\epsilon = \frac{E_{NN} - E_{True}}{E_{True}} \tag{5.12}$$

Where the NN subscript denotes the result from the neural network. This measure is also known as the percentage error, since it gives the error in percentages of the true value. While this would technically avoid the aforementioned problem of weighting the higher end of the distributions too heavily, the fact that the network needs to be able to give outputs that cover several orders of magnitude is undesirable, since it is generally known that this causes a decrease in prediction quality. We can somewhat fix this by preprocessing the data a bit and taking the logarithm to the energies:

$$\epsilon = \frac{log_{10}E_{NN} - log_{10}E_{True}}{log_{10}E_{True}} \tag{5.13}$$

This is not sufficient, since this effectively changes the base of the logarithm to be the true energy. This means that we need to remove the denominator, and if we do that, we can use the property of logarithms that a subtraction equals division:

$$\epsilon = log_{10}E_{NN} - log_{10}E_{True} = log10(\frac{E_{NN}}{E_{True}}) \tag{5.14}$$

We can not use this directly, since it is unbounded from below and so a network attempting to minimize this will just continue giving a lower value. To fix this we can take the absolute value of this quantity, but this quantity is both quite steep in its curve and also not differentiable at 0. That is why we use the log10 hyperbolic cosine of the difference, whose slope is easier to traverse.

$$\epsilon = log_{10}Cosh(E_N N - E_{True}) \tag{5.15}$$

5.9.2 *Angular regression*

For the case of the Angular regression, we have the problem of periodicity. In short, a prediction that is "off" by 2 pi in the case of zenith angle should in fact give the same result as one which has the "correct angle", and our choice of loss function should reflect that. We have a multitude of choices in this regard. In the approach called Von Mises-Fisher Sine-cosine loss approach we transform the truth value to be:

$$p(\phi_t ruth) = \begin{bmatrix} sin(\phi_{truth}) \\ cos(\phi_{truth}) \\ 1 \end{bmatrix} \tag{5.16}$$

and ask the model to produce three outputs:

$$output = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{5.17}$$

The loss function is chosen such that it minimizes the angle between vector p and a version of the output vector where c=1. It does this by using the negative log likelihood trick as such:

$$\epsilon = -ln(c) + ln(4\pi) + c + ln(1 - e^{2c}) - c * cos(\delta\phi) \tag{5.18}$$

where $\delta\phi$ is the angle difference between the two previously mentioned vectors. A full derivation is available in a paper on training seq2seq models [2].

[2] Sachin Kumar and Yulia Tsvetkov. "von mises-fisher loss for training sequence to sequence models with continuous outputs ". https://arxiv.org/pdf/1812.04616.pdf, March 2019

5.10 *Learning rate optimization*

I previously introduced the learning rate parameter α. The choice of this parameter is perhaps the most important one with regards to the networks ability to traverse the loss landscape and computational time.

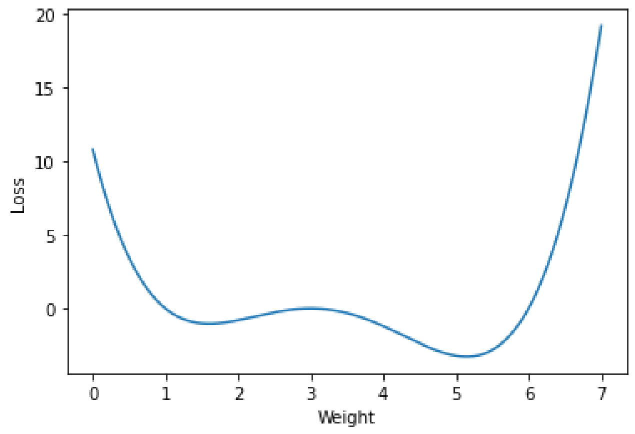


Figure 5.6: A fictional loss landscape, with loss on the vertical axis and parameters of the network on the horizontal axis

To understand why, consider the fictional loss landscape in figure (5.6).The place we want to end up in is the bottom of the valley on the right, but our starting position might be on any place on the line. If we start anywhere on the right valley, we balance lowering the learning rate such that we overshoot the bottom of the valley less and increasing it to lower computation time. However, if we start somewhere in the valley on the left, we have the additional problem that we want to ultimately "jump" over to the valley on the right. This ends up imposing a lower bound for the learning rate parameter, since for a given point there is a value for the parameter that must be exceeded for the jump to be possible.

While this hopefully has illustrated the central concerns with choosing α, we must remember that our loss landscapes are typically highly complex and so too our networks, and thus the parameter can be very hard to optimize. We need an optimization strategy for α, and of the many that exists we choose the ADAM optimizer, due to its robustness and relative insensitivity compared to other optimizers. Full documentation for this optimizer can be found at [3], but in brief terms, the optimizer calculates individual learning rates for each neuron using the "momentum", a measure that allows it to statistically determine which neurons would be most impacted by a learning rate that is set too high or too low.

[3] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980, December 2014

6
Graph Neural Networks

6.1 Brief introduction to graph Theory

Graph theory is the theory of Graphs, which are a mathematical structure that models pairwise relations between mathematical objects. Formally, a simple graph $G = (V, E)$ consists of a set of vertices V,also commonly called nodes, and edges $E \subseteq \{\{x, y\} | x, y \in V \text{ and } x \neq y\}$. An edge starts at one node and ends at another. The neighborhood of a node n is $f(n) = \{m \in V | (n, m) \in E$. An adjacency matrix is a $|V| * |V|$ matrix, where $|V|$ denotes the amount of elements in V, each element A_{ij} is 1 if $(i, j) \in E$ or 0 if $(i, j) \notin E$.

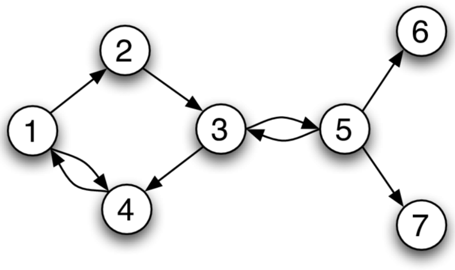


Figure 6.1: An example of a directed graph with 7 nodes and a total of 9 edges.

When we associate features X with nodes and features X^E with edges, it is called an attributed graph, which is the focus of this work. We are dealing with simple graphs where each node can have at most one edge to a given other node and there are no nodes with edges to themselves. Furthermore, we will be dealing with directed graphs, where a node m can have an edge going to node n without the reverse being true. See figure (6.1) for an example of this structure.

The intent behind representing data as a graph is to use the fact that some data points are associated with others. This has previously been exploited with techniques such as CNNs, convolutional neural networks, which view the data as images and directly associates neighboring points. The successful use of CNN models inspired the development of graph neural networks, since graphs are the most general form of modeling relational data.

There are two fundamentally different ways of extracting information from a graph. The difference lies in how the Graph Laplacian is interpreted. The Graph Laplacian is the difference between the adjacency matrix and the degree matrix, which is a diagonal matrix whose elements contain information about the total amount of edges a given node has. In Spectral Graph Theory, a eigen-decomposition is performed on the graph laplacian and the information is interpreted as a frequency space. In Spatial Graph theory, the Laplacian is directly interpreted as a measure of the spatial connectivity of the graph. For more see Paul Kurasovs paper [1].

[1] Pavel Kurasov. Graph laplacians and topology. https://projecteuclid.org/journals/arkiv-for-matematik/volume-46/issue-1/Graph-Laplacians-and-topology/10.1007/s11512-007-0059-4.full, 2008

Since our data does not come prepackaged as graphs, we need to process into ones. The choices made in this regard are discussed in later chapters.

6.2 *Message Passing*

Just like standard neural networks, graph neural networks can be thought of as a function that takes an input, namely graphs, and produces an output in the form of a label. GNNs, same as NNs can be either thought of as constructed in layers or as a series of operations. The fundamental difference is that where NNs building blocks are layers of neuronal activations, the GNN's building block is the message passing scheme.

The basic idea behind a message passing scheme is that you take a graph G and produce a new graph representation G′, by having each node pass a message to its neighbors. For a given node m with a neighbor n, it receives the message $message = f(n)W$ from its neighbor, where f is some operator and W is a learnable parameter. G′ is then constructed by having all nodes receive all messages from all their neighbors, and then either including or excluding their own values from G.

How you proceed from the new graph G′ depends entirely on the purpose of the GNN. GNN's are very powerful in the sense that they can not only be used to predict labels for nodes or labels for graphs, but also properties of the graphs themselves. For example, a time evolution of a social media network is to some degree possible to predict[2].

[2] Shengjie Min, Zhan Gao, Jing Peng, Liang Wang, Ke Qin, and Bo Fang. "stgsn — a spatial–temporal graph neural network framework for time-evolving social networks ". https://arxiv.org/pdf/1905.10990.pdf, February 2021

For our problem, we need to proceed from the graph G′ to possibly G″ or some other number of more complex representations to a single label for a graph. The general term for such a function or operator is called an aggregation function. In this thesis, we will typically only use aggregation functions consisting of pooling operators followed by a standard neural network decoding layer. When we add more layers of different graph operators to our network, we encounter a fundamental constraint of GNNs that NNs do not seem to have, namely that past a certain point, adding more message passing schemes starts to quickly reduce the quality of predictions, even on data that has been trained on. This limit is even reached rather quickly, with most GNNs worsening after 4 or 5 layers [3].

[3] Thomas N. Kipf and Max welling. "semi-supervised classification with graph convolutional networks ". https://arxiv.org/abs/1609.02907, September 2016

The reason this happens is quite simple. A message passing scheme is essentially a way to accumulate information about a nodes neighborhood. When two are applied, the new graph G" and its nodes have information about their neighborhoods neighborhood. This process continues as more schemes are added, and the more it continues, the more it washes the information present in the relations given by the edges and the more noise is introduced. Furthermore, a node is also it's neighbors neighbor, and so increasing the amount of layers has the additional effect of considering a node to be its own neighbor in a sense. For an intuitive example, consider your own acquaintances on social media. It should be immediately apparent why there is meaningful information in your relation to them. If we hop to your acquaintances acquaintances, the link between you is significantly weaker. If we hop 6 degrees away, we cover most of the world, per the common myth of six degrees of separation.

7
Development Process

This section will detail the broader strokes of the work i have done on this masters thesis. As all Masters students do, i spent the first period of time familiarising myself with the literature and the projects involved. Thus i became aware of the fact that the employment of Neural networks in IceCube was relatively new, but had shown promising results through the Masters theses of Mads Ehrhorn and Bjørn Mølvig. Bjørn paved the way and showed that an improvement of the Retro-Reco algorithm was possible using a Bi-directional GRU model, and Mads's success with a Temporal Convolutional neural (T-CNN) network model indicated that models which took advantage of temporal patterns could be the way forward. This section includes a description of the work i did on the simulated upgrade data, but it can be skipped entirely if so desired.

7.1 Choice of library

One of the first steps involved in developing machine learning algorithm is the choice of programming language and library. I chose Python, both because i was familiar with the language and, because most research in machine learning is done using python libraries. When it became time to choose a library i had settled on attempting to develop a Graph neural network, since Rasmus Ørsøe, who was currently doing his masters thesis on the same subject, had outperformed the rest using these kinds of models. Rasmus Ørsøe had become aware of a new extension to pytorch-geometric, which is a library for graph networks in pytorch, called pytorch-geometric-temporal. He was about to finish his thesis and thus could not attempt an entirely new library, but i was more than willing to try, since i believed that the success with the T-CNN had shown building time-dependence more directly into the model would be fruitful. Pytorch-geometric-temporal was developed by a machine learning engineer working for Astra-Zeneca, Benedek Rozemberczki, and had only been released July 2020. I spent a bit more than a month trying most operators and models in the library. After writing to the library's author and explaining what i was trying to achieve with what kind of data, i had, he advised me to not proceed with the library. By this time, i had become familiar with pytorch and pytorch-geometric and so that ended up being my choice.

7.2 *Pytorch-geometric*

The pytorch-geometric library contains a multitude of layers and operators. I knew that Rasmus Ørsøe had primarily used the Edgeconv operator for his model called Dynedge, and so i wished to use a model using something different to either beat or reach his models performance. At this time, i developed models using every single other operator i thought could be used for our purposes, which was about half the operators in the library. The one i had the most success with was the Gated Graph operator, which excited me since it used a GRU(Gated Recurrent Unit), which is a neural network method also used by Bjørn Mølvug. After spending a bit more than a week developing this model on energy regression and making steady progress, i decided to test the very simplest application of the Edgeconv i could think of (one application of the operator, a pooling operator and a single neural network decoding layer) and found that this easily beat the average logcosh loss after training by a factor of 10. With my deadline steadily approaching, i decicded to change tactics and instead work on improving the model that Rasmus had developed. This would prove to be quite difficult, because many of the choices that had been made in its construction, such as number of neurons in each layer, number of Edgeconv Operators and configuration of pooling layers had already been extensively optimized. I did end up with some improvement, especially in the low-energy regime, which we will see in a later chapter, but first i have to explain the dynedge model.

7.3 *Dynedge*

The dynedge model fundamentally uses four different building blocks. The first is the Edgeconv operator, the second is the K-nearest neighbor algorithm, the third a feature concatenation, the fourth a node aggregation operator and the fifth a simple Fullyconnected Feedforward. Before i show what the final model looks like, i will explain what these individual blocks do.

7.3.1 *FF-layer*

The FF-layer is the simplest layer structure described in Chapter (5), a layer where each neuron is connected to each feature. Note that some of these are in fact two feedforward layers, one feeding into the other. The exact structure is available in the appendix.

7.3.2 *Edgeconv Operator and KNN*

The Edgeconv operator originates from a paper released in 2018 [1]. The operator acts as the workhorse of their point-cloud model that is meant to learn geometric shapes. Edgeconv acts on nodes per the following:

$$x_j^{'} = Aggr(\{f(x_j, x_i)|x_i \in E(n_j)\}) \tag{7.1}$$

[1] Yongbin Sun et al. Yue Wang. "dynamic graph cnn for learning on point clouds ". https://arxiv.org/abs/1801.07829, June 2018

where x_j, x_i are the node features of node i and j and $E(n_j)$ denotes the neighborhood of node j.$Aggr()$ is a message aggregation scheme and $f(x_j, xi)$ is a function that acts on two sets of node features.In dynedge, $Aggr()$ is chosen to be the addition function, and $f(x_j, xi)$ is chosen to be two FF networks, one feeding into the other, with a LeakyRELU activation between them and after them. The neighbors are chosen using the KNN algorithm, which finds the K nearest neighbors over certain features. The model with which I'm competing uses 8 neighbors in the three spatial coordinates.

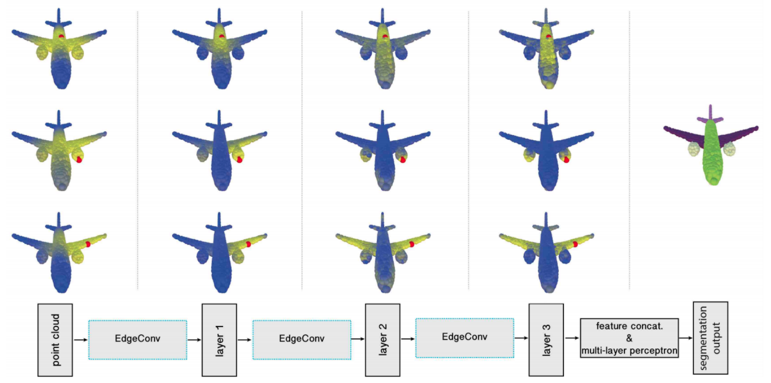


Figure 7.1: An example of the application of the Edgeconv operator from . The colors in the edgeconv layers denote the distance from the red point to the rest. Notice how it gets better at grouping together things we as humans recognize as similar like the wings.

7.3.3 *Node Aggregation*

Node aggregation is the method by which many nodes are condensed down to one node, thus allowing for the many-to-one casting necessary to get a single label for the graph.The four agggregation schemes used here involve taking the mean, adding them all together, taking the minimum value and taking the maximum value.

7.3.4 Feature concatenation

Feature concatenation is very simple, in that it takes different graph representation and simply concatenates their features together. For example, if we have 4 graphs each with N nodes and M features, a feature concatenatation would result in 1 graph with N nodes and $M * 4$ features.

7.3.5 Model architecture

Now that the building blocks are here, we can dissect the diagram in figure (7.2).

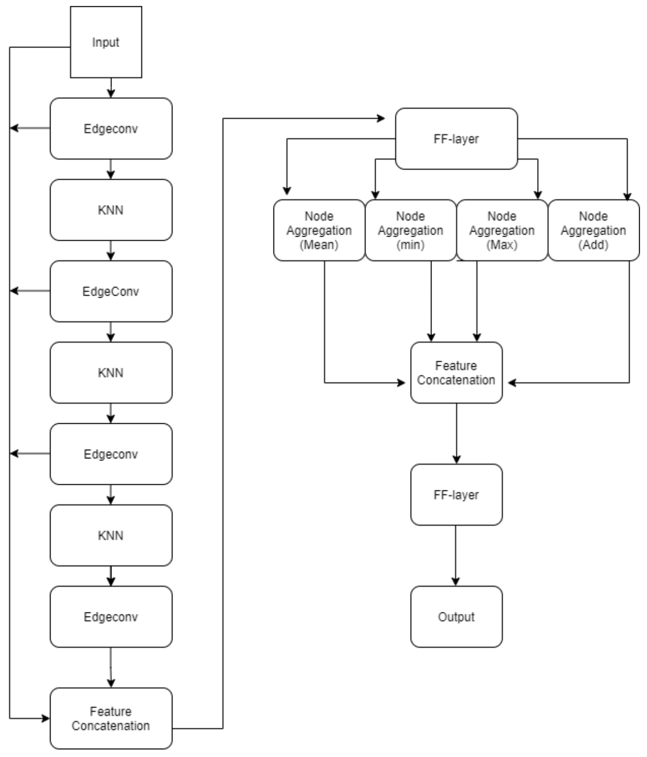


Figure 7.2: The Dynedge Model. Arrows denote the flow of information.

Starting from the input, we see that the input feeds forward to an Edgeconv operator, which feeds forward to KNN operator. In fact, we calculate new neighbors a total of three times in the block to the left, and the reason that it is exactly this recalculation which allows the network to tease out the higher ordering geometric groupings. The column furthest to the left tells us that each of the 5 graphs, the input graph and the four generated by successive Edgeconv operations are fed into a feature concatenation scheme. This has shown itself to be superior than to simply taking the output of the last Edgeconv operator, and this is perhaps because it allows the network to find a pattern in how it gets from one representation to the next. The output from the feature concatenation is then fed to a FF-layer which acts to decode these complex graph representation. Four node aggregation schemes are run in tandem, taking the Mean,minimum,maximum and the addition, and the results from these aggregation are fed to another feature aggregation. In the development process, it was found that each node aggregation scheme by itself varied in the quality it added to the network across multiple feature ranges, but none of them could be conclusively said to be superior to the others(This concurs with my experience in using node aggregation when developing other models). Using all four in tandem and allowing the network figure out for itself how to determine which one is best suited for a given case was found to be the best option.

Note here that the KNN graphs that are fed the output from an Edgeconv operation no longer calculate their neighbors in real space but in a representational space. Fortunately, it seems like we can still limit ourselves to three dimensions when calculating neighbors in this space, since the model learns to put the appropriate features such that when it calculates neighbors it does so in a meaningful way.

7.4 *Developing on Dynedge*

Dynedge was already a well optimized when i got to it. After acquiring an overview over the optimization that had already been done, i attempted several things which had not. I will give only a brief overview of some of these since none of them ended in success. Note that some of them were also tested with each other, meaning that i both tested them as being the only change and also while changing multiple things. All development was on the database*dev_numu_train_l*5*_retro_*001, which is a database containing level 5 data with only muon neutrinos and muon antineutrinos. The dynedge training was done with a learning rate of 0.01 using the ADAM optimizer and was trained over 10 epochs.

7.4.1 *Graph Architecture*

In chapter (6) i stressed the importance of how the graph is constructed conveys certain information to the network. For example, there is certainly some information to be had when detectors are not even detecting anything even when they should be, which is in fact something we use to veto certain events occurring in Deepcore but not detected outside it. Most of my changes to the graph architecture involved either adding features to each node that contained some sort of information about the nodes(such as adding a feature that involved a coordinate transform of its spatial coordinates) or adding entirely new nodes to the graph which contained global information about the graph (such as adding a new node whose features were the averages of the features in the node). Sadly, i could not optimize dynedge enough or my choices were too poor to gain a performance improvement.

7.4.2 *Hyperparameter optimization and configuration*

Another category of attempts was attempting to optimize things which i felt had not been sufficiently optimized. For example, i found out that Dynedge's number 8 for the KNN algorithm was the same for each KNN-algorithm run, and i ran a time consuming computation varying the different K's in each KNN block. This had perhaps the best result of my attempts, in that some of these configurations were not significantly worse than Dynedge with some even being almost entirely similar.I got similar results when attempting to implement my own version of attention and context layers into the various concatenation and aggregation layers. Nevertheless, i determined that this was not the way to proceed, due to there being no improvement in any energy range. To improve dynedge i decided that i needed more than what dynedge had to offer.

7.5 *Extending Dynedge*

After i decided that it was not fruitful to directly optimize dynedge, i revisited the different operators and layers in pytorch-geometric. Now that i had more experience, i could also better understand the conditions by which i could justify developing a certain approach. Firstly, i came to understand that initial attempts at models were almost entirely guaranteed to give worse results than dynedge, and so the condition for developing a given extension should be that the hit in performance is "reasonable" and that the quality I'm looking for is any improvement at all, for example an improvement in a given energy range. Furthermore i had the deep suspicion that adding more layers that simply produced a new graph representation would not provide an improvement, and this was confirmed with the few i did test. I turned my attention to the pooling layers. A pooling layer is a layer that acts through some process coarsen the graph by acting on node features, nodes or both. For example, a simple pooling layer could be a *average_pool* layer which creates a new graph representation. This type of pooling has shown itself to be useful in CNN's where it acts to transform local data into regional data. Pooling can also act on nodes and serve to reduce the total amount of nodes in the graph by selecting nodes to discard through some function. I tested all the applicable ones on Dynedge, setting the pooling layers at various places in the network architecture. Through this i discovered that the Edgepooling pooling layer had a very slight improvement in the lower energy range spectrum.

7.5.1 *Edgepooling*

The Edgepooling operator comes from [2]. It is an operator that iterates over the nodes and their neighbors, scores them and then contracts nodes together according to this scoring. The first step is to compute the raw score for a given neighbor:

[2] Frederik Diehl. "edge contraction pooling for graph neural networks ". https://arxiv.org/pdf/1905.10990.pdf, May 2019

$$r(E_{ij}) = W * (x_i||x_j) + b \tag{7.2}$$

where x_i amd x_j are the node features, W and b are learnable parameters. Once all the scores for each neighbor has been calculated, a scoring function is applied to it. I found the best performing scoring function to be the softmax function, so the final score becomes:

$$s_{ij} = 0.5 + softmax_{r*j}(r_{ij} \tag{7.3}$$

where the subscript $r * j$ denotes that the softmax function has been taken over all neighbors, and the 0.5 is added for numerical stability. With these edge scores we iteratively contract node ignoring those which have a newly-merged node incident. By contracting, we combine the edges and add their node features multiplied by their final score:

$$\tilde{x}_{ij} = s_{ij}(x_i + x_j) \tag{7.4}$$

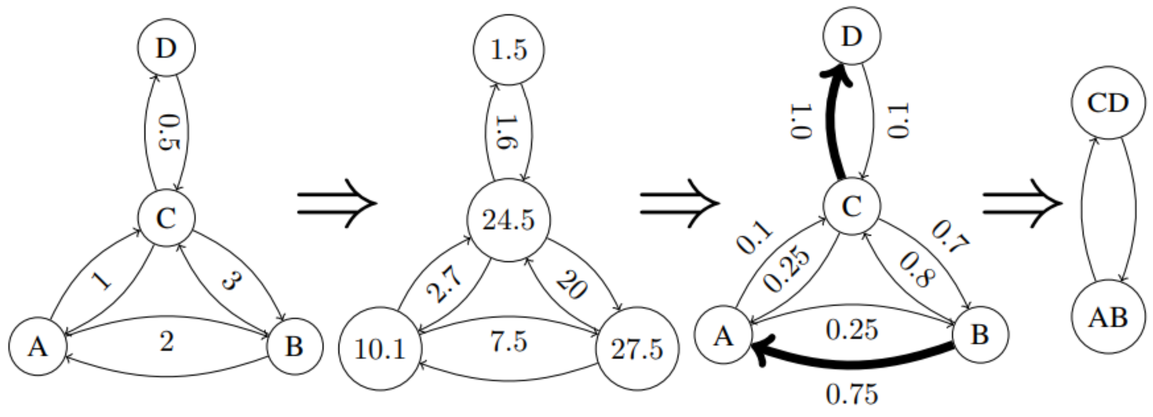


Figure 7.3: Edgepool operating on four nodes. Both direction of a node are assumed to have the raw score. The first figure shows the raw scores for each edge, the second the computed score score and the sum over all incoming raw scores, the third showing final scores for each edge Arrows denote the flow of information

7.5.2 *DynedgeEdgepool*

Having settled on optimizing the extension of Dynedge with the Edgepooling operator, i decided to called the model DynedgeEdgepool. In the development process, the first thing i had to figure out is where to place it in the architecture. I knew i could not place it inbetween the different Edgeconv operators in figure (7.2), since it would lead to the feature concatenation receiving different amount of nodes and that this simply does not compute. I found a slight amount of success placing it after the feature concatenation, but i determined that this was a mirage, since it merely acted to decrease its influence on the final result. I settled on setting it in front of the entire dynedge model and developing from that point on. My initial suspicion on it's lopsided performance(better on low energy, worse on higher) was that it the model was too complex, and so most of my optimization involved changing the specific parameters of the layers. This was not a fruitful approach and eventually i found that the best approach was fiddling with the learning rate and number of epochs. The final model trains on a learning rate of 0.0002 and trains for 50 epochs. The final graph architecture was chosen as 4 nearest neighbors in position. Exact parameters for the model are in the appendix and so is a comparison with Dynedge.

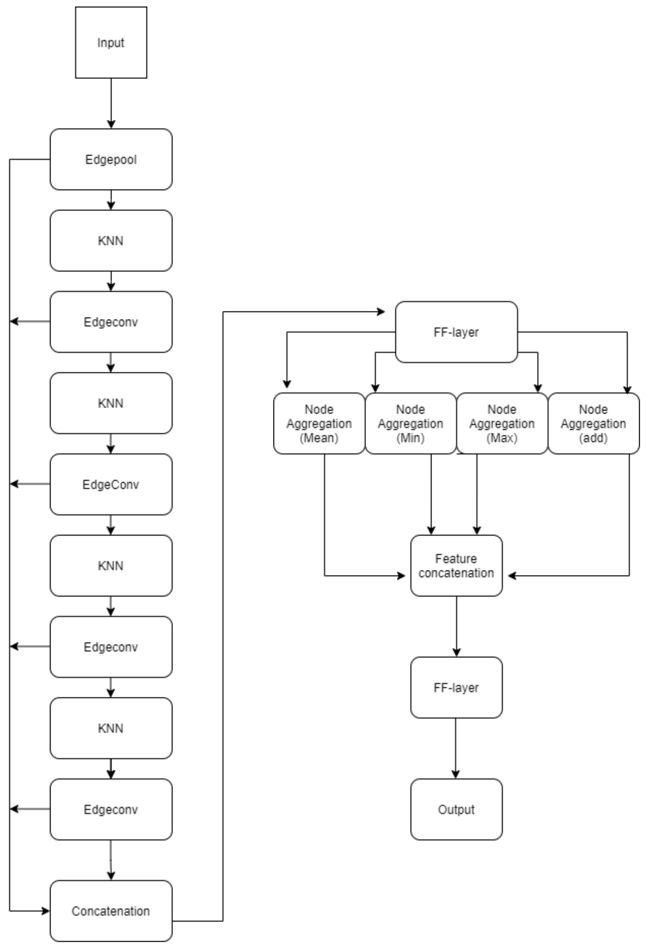


Figure 7.4: The final model architecture. Note that the input no longer feeds in to the first feature concatenation, and that the contracted nodes edges are recalculated

8 Results

In this chapter i present the reconstruction of energy and zenith angle from the final model in chapter (7). These are the two most important parameters for our purposes, since the probability of observing a tau neutrino that arises from a muon neutrino decreases with higher energies and a smaller volume of matter traversed in the detector (abs(cos(zenith)>0)). It consists of $\sim$ 4 million muon neutrinos and $\sim$ -14 muon anti neutrinos, for a total of $\sim$ 5.8 million particles. Both interaction types are included. All training and prediction was done on SRT-cleaned inputs. The training takes 16 hours, with the reconstruction of events done at a rate of an average of 6000 Hz on a Nvidia 3090 GPU. In the following i will plot various performance measures on which DynedgeEdgepool is evaluated as well as retroreco. Several of these will have an outline of a histogram of events in the background to let the reader visually identify the areas where there is more information. All data is preprocessed with the Robustscaler transformation from scikit-learn, which transforms the data in the following manner

The level 5 database used can be found at `https://sid.erda.dk/wsgi-bin/ls.py?share_id=ETCAbdoluA;current_dir=dev_numu_train_l5_retro_001;flags=f`.

$$x^i_{transform} = \frac{x^i - median(x)}{IQR(x)} \tag{8.1}$$

Where x^i is the i'th element of set x and IQR signifies the interquartile range. This transformation rescales and reduces the impact of large outliers of a distribution, thus making it more "robust". Prior to plotting the values are transformed back.

8.1 Energy Reconstruction

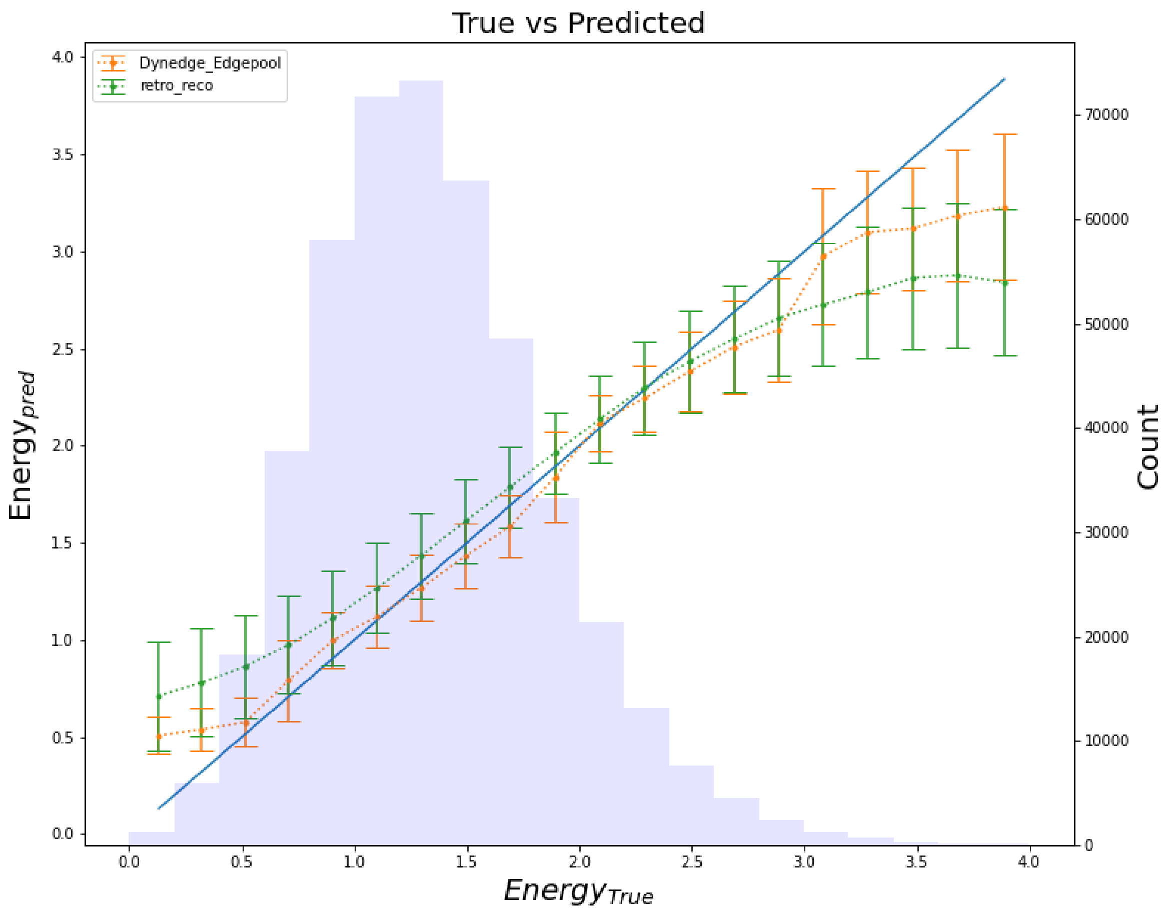
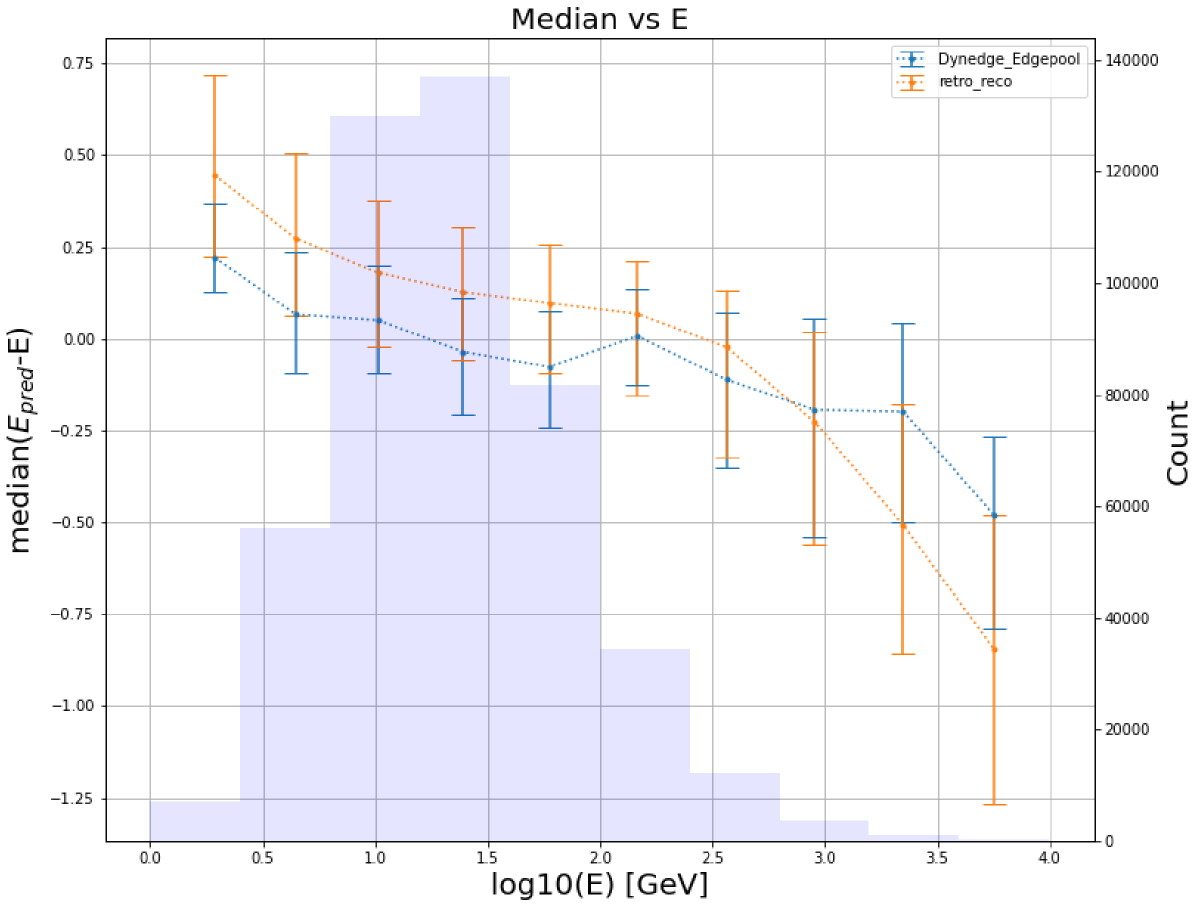


Figure 8.1: A direct comparison of predicted values as a function of true values. The error bars are chosen as the standard deviation of the prediction.The blue line is a theoretical perfect fit.



,

Figure 8.2: The median of the Error. The error bars are the width of the error distribution.

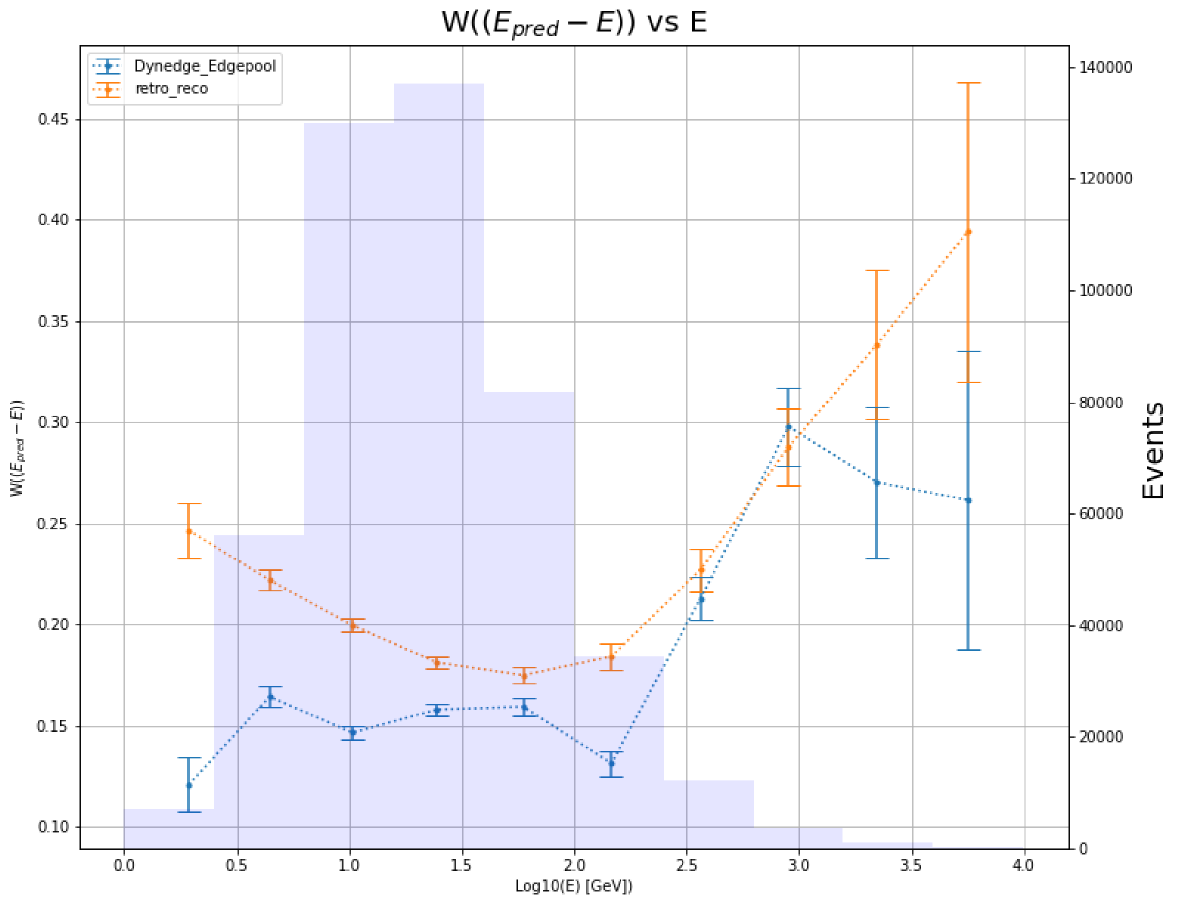


Figure 8.3: The width of the error distribution. The error bars are the errors on the width.

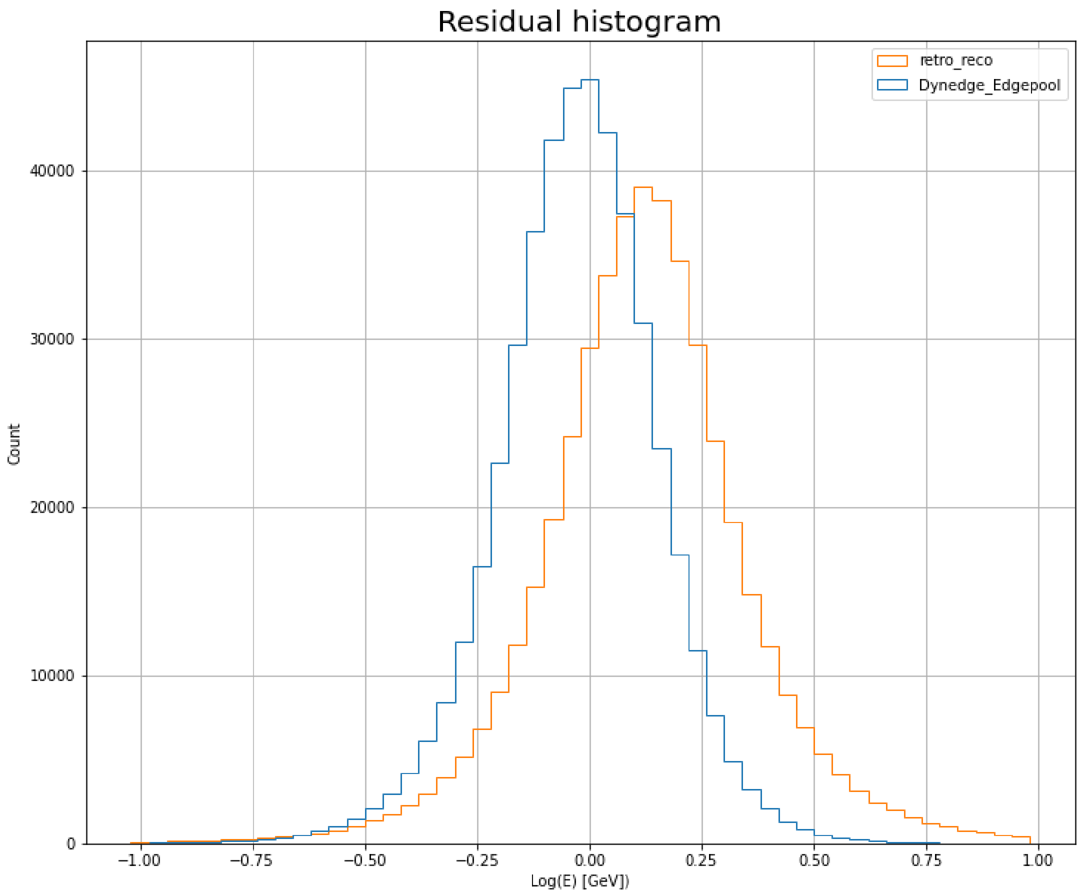


Figure 8.4: A histogram of the residual values. Note that $retro_{reco}$ is not centered.

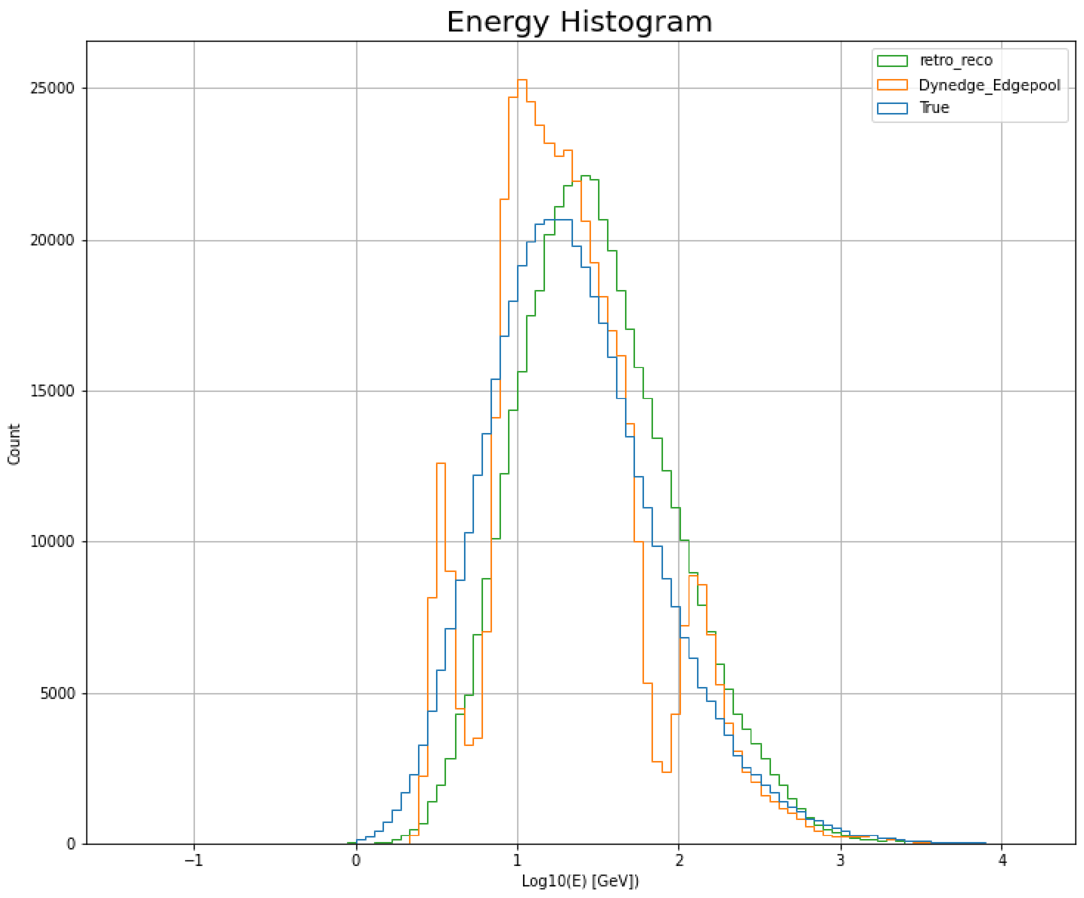


Figure 8.5: Histograms of reconstructed values. Note the existence of two "valleys" at around 0.7 and 2 log10 GeV

8.2 Zenith angle reconstruction

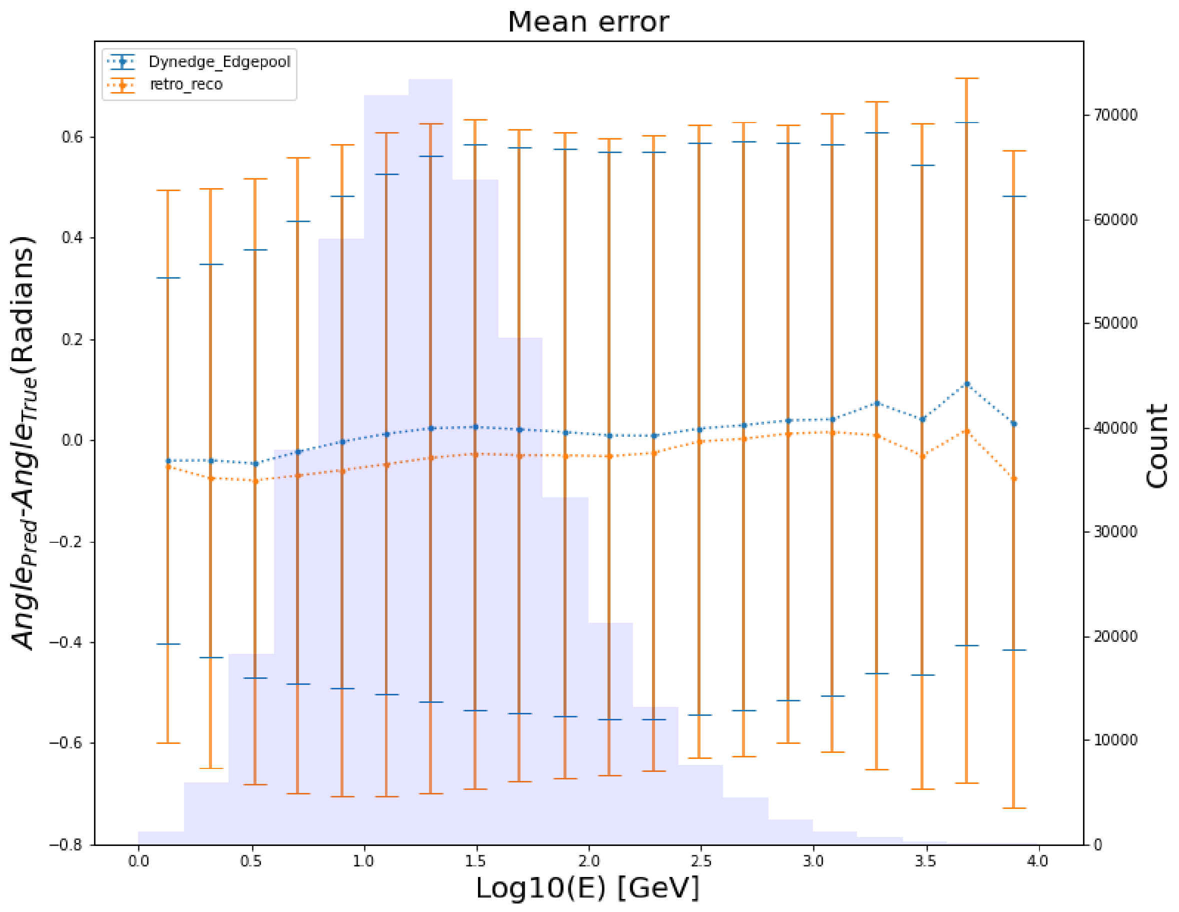


Figure 8.6: A direct comparison of predicted values as a function of true values. The error bars are chosen as the standard deviation of the prediction.

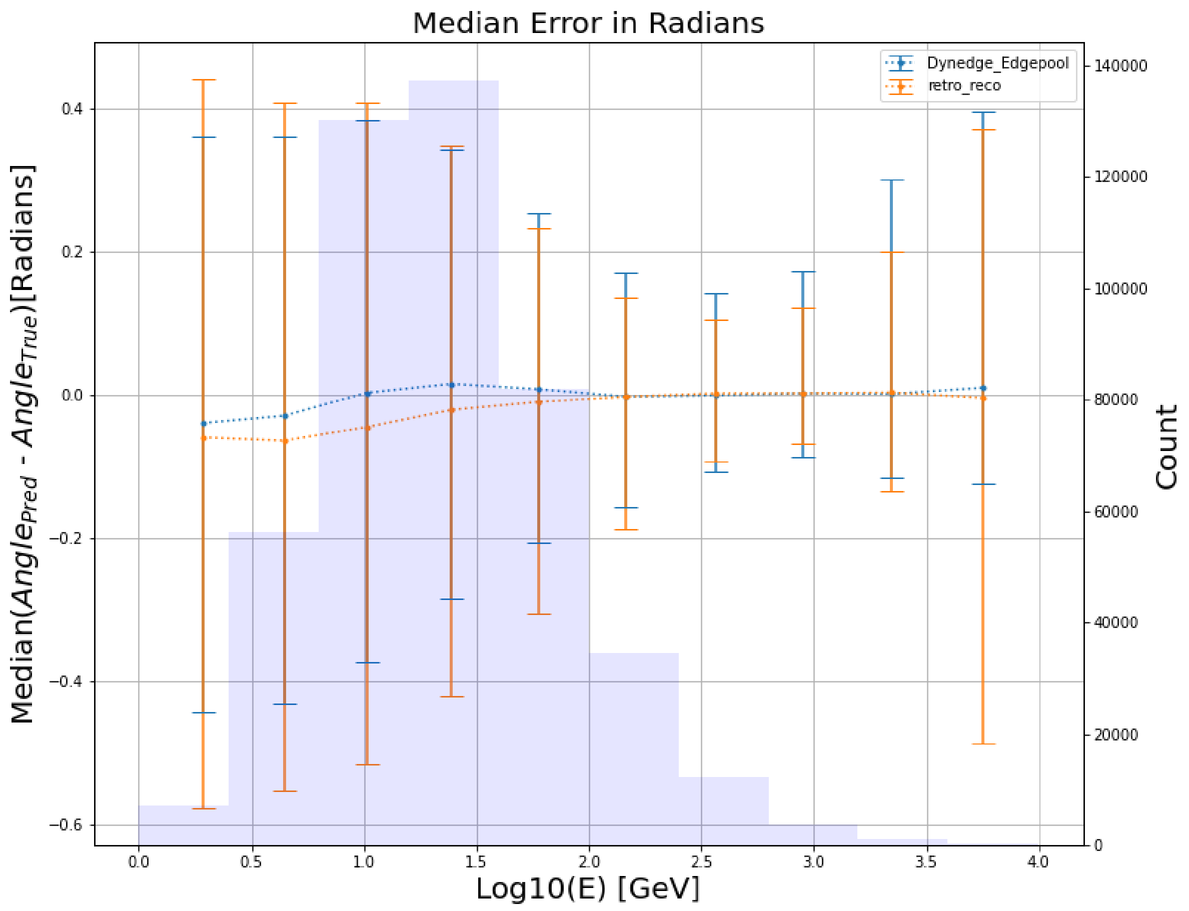


Figure 8.7: The median of the Error. The error bars are the width of the error distribution.

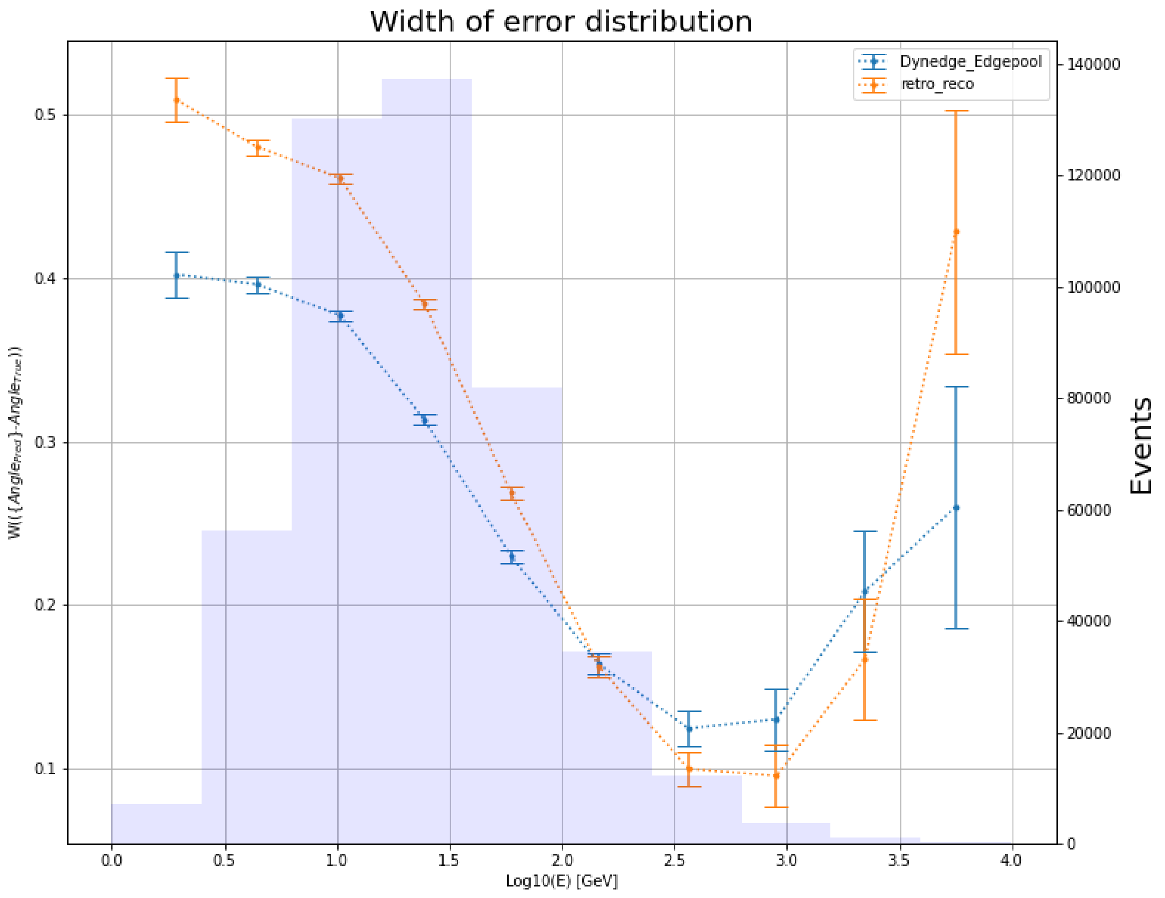


Figure 8.8: The width of the error distribution. The error bars are the errors on the width.

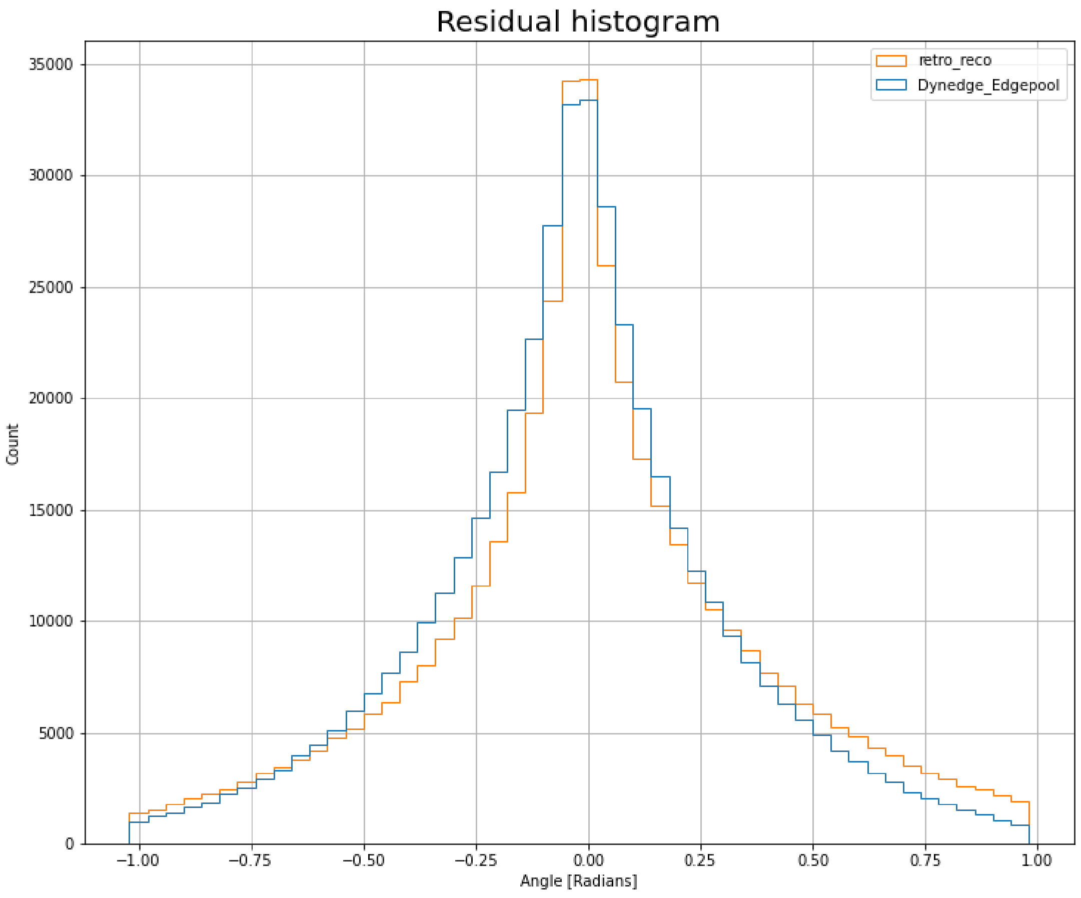


Figure 8.9: A histogram of the residual value.s

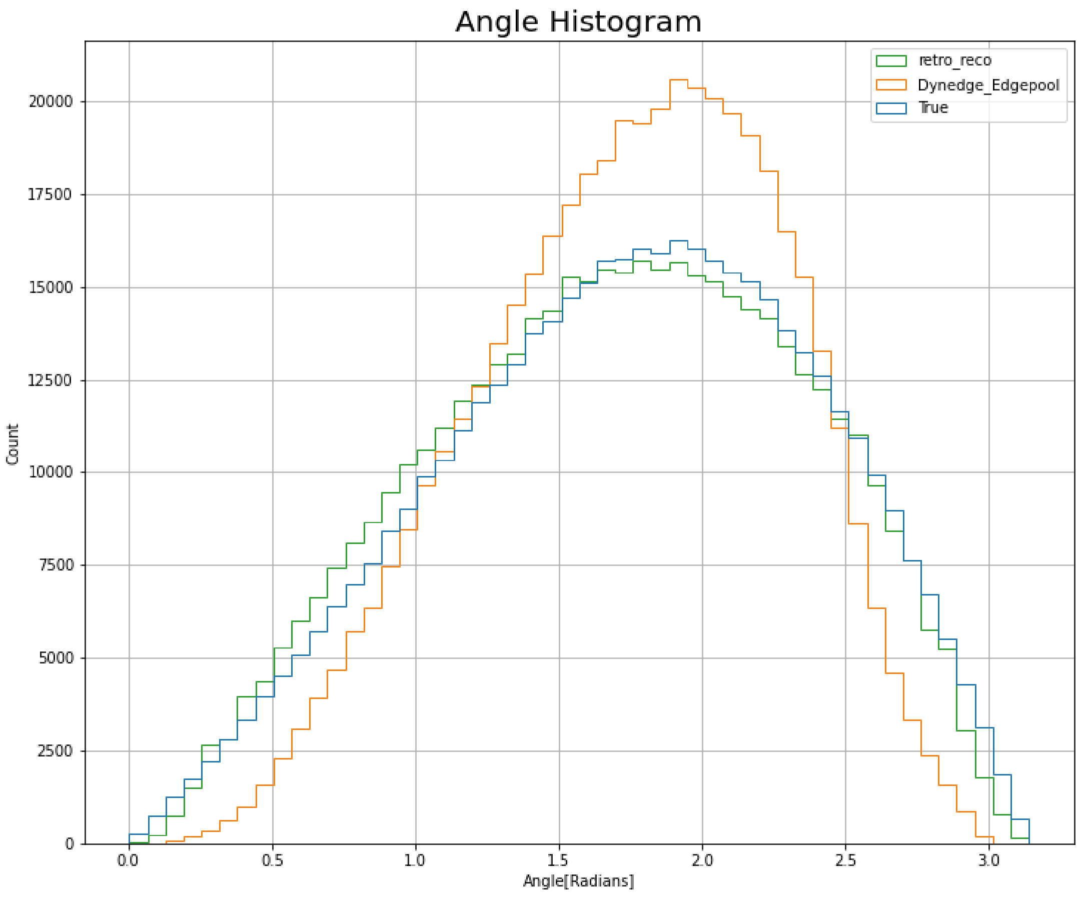


Figure 8.10: A histogram of reconstructed values shown against the histogram of true values.

8.3 Interpretation

In the case of energy regression, my model outperforms retro_reco across almost all energy ranges. In particular, significant improvements are gained in the low energy range (0-1.5 Log10(GeV)), which is the most important for neutrino oscillations as seen in fig(3).Across the entire energy range, the relative improvement given as:

$$relativewidth = \frac{width_{dynedgeEdgepool}}{width_{retro\_reco}} \tag{8.2}$$

comes out to ~ 1.25 or a 25% improvement. The histogram of values fig (5) shows retro_reco having a distribution that resembles the true distribution more. Especially troubling is the appearance of two valleys, one at around 0.7 log10 GeV and the other at 2 log10 GeV, where the count of values is low. This seems to show a structural bias that i failed to ascertain the cause of. In my investigations it seemed to have a weak correlation to the number of pulses in an event. The existence of the peak to the left of 1.5 log10 GeV is consistent with figure (8.1, where values higher than 1.5 are consistently predicted too low and thus shifted to the left in the histogram.

In the case of the zenith regression, the competition is fiercer. While i do beat it in the lower energy ranges and in the high sample-size area as per fig (8.8), there is still a significant portion in the high energy(2.5-3 Log10(GeV)) area where retro_reco wins out. The relative improvement comes out over all energy ranges to be about 20%, which signifies that for both targets my model is significantly more consistent. In the histogram of values fig (8.10) retro_reco performs significantly better. Part of that bias stems from the general behavior that ML does not like to guess at the far ends of a distributions, which is further compounded by the fact that a guess at the tail has a good chance of simply applying the periodic conditions to them and being "rolled" over.

9 Conclusion and Outlook

9.1 Conclusion

The central issue that this work tries to address is whether or not machine learning models have developed to the point where they can applied to IceCube data. The results of the investigation lead me to believe that the answer to this is a resounding yes. The gains in the accuracy can not necessarily be extrapolated to all the data that comes out of IceCube, since this was run over a level 5 database that had already had many layers of processing applied to it, but the question of whether or not it is worth the time and effort to pursue the development of ML methods must be answered in the affirmative when considering the hurdle that computation time poses. I would venture that even had the model been slightly worse than retro_reco rather than better, this would still given weight to the argument for ML. The most important measure for our purposes is the relative improvement in width of error, since this is the measure that directly indicates whether or not the architecture can represent the variance in the data, and in this regard the model demonstrates an improvement. The results of this thesis, when combined with the results from previous masters theses, provide a strong argument for the development of ML in IceCube. It is possible that many of the cleaning levels described in chapter (4) could be replaced with GNN, especially the levels selections are made on classifications, since GNN's are generally capable of producing good classifications.

9.2 Outlook

If i were to extend this project, there are a couple of things i would have liked to investigate:

- Develop the model to act as a full replacement for retro_reco. Currently the model has to be trained separately on each individual target to be able to reconstruct, and thus it can not use information about its reconstruction of other targets. The ultimate goal would be to make a model that takes in the pulses of an event and returns out all the targets in the truth tables.

- Develop ML models for cleaning noise from the data. I briefly experimented with replacing the SRT-algorithm with a ML model that would try to emulate it, which did not work, but i think the general idea could still be fruitful.
- Perhaps i would have taken the time to rewrite the Edgepool operator from scratch. As it stands, the operator from the library method seems to use significant amount of CPU-calculations which takes up more than half the time of reconstruction. Rewriting it to use GPU calculations could reduce the time by more than half.
- I would have like to have worked more on the upgrade data, especially with the noise-labelling that was being simulated.

Appendices

DynedgeEdgepool Model

```python
class dynedgeEdgepool(torch.nn.Module):
    def __init__(self, input_size=4, output_size = 1,x_col = 0,y_col = 1, z_col = 2, mode = 'custom', k = [4,4,4,4], c=3, device = 'cuda' ):
        ####
        # INPUTS:
        # input_size    : INTEGER - dimension of input tensor.DEFAUlT -4
        # output_size   : INTEGER - dimension of output tensor. DEFAUlT -1
        # x             : INTEGER - column index in input tensor for x-coordinate of DOM position. DEFAULT - 0
        # y             : INTEGER - column index in input tensor for y-coordinate of DOM position. DEFAULT - 1
        # z             : INTEGER - column index in input tensor for z-coordinate of DOM position. DEFAULT - 2
        # k             : INTEGER - number of neighbours. DEFAULT - 4
        # device        : STRING  - the device ID on which the model is run. DEFAULT - 'cuda'
        # c             : INTEGER - the dimension factor. DEFAULT - 3
        # target        : STRING  - specifies which version of dynedge to run. ['energy', 'angle', 'classifcation']
        #                   target = energy         : Regresses energy_log10. Use in conjuction with 'logcosh' loss function.
        #                   target = angle          : Regresses either zenith or azimuth. Use in conjunction with 'vonMisesSineCosineLoss
        #                   target = pid            : Use in conjuction with torch.loss.CrossEntropyLoss
        #

        super(dynedgeEdgepool, self).__init__()
        self.k = k
        self.mode = mode
        self.device = device
        self.pos_idx = [x_col,y_col,z_col]


        l1, l2, l3, l4, l5,l6,l7 = input_size,c*16*2,c*32*2,c*42*2,c*32*2,c*16*2,output_size

        if mode == 'angle':
            output_size = 3            # VonMisesSineCosineLoss requires three dimensionsional output
        if mode == 'energy':
            output_size = 1            # logcosh requires one-dimensional output
        if mode == 'pid':
            output_size = 2            # CrossEntropyLoss requires two-dimensional output

        self.nn_conv1 = torch.nn.Sequential(torch.nn.Linear(l1*2,l2),torch.
     nn.LeakyReLU(),torch.nn.Linear(l2,l3),torch.nn.LeakyReLU()).to(device)

        self.conv_add = EdgeConv(self.nn_conv1,aggr = 'add')

        self.nn_conv2 = torch.nn.Sequential(torch.nn.Linear(l3*2,l4),torch.
     nn.LeakyReLU(),torch.nn.Linear(l4,l3),torch.nn.LeakyReLU()).to(device)

        self.conv_add2 = EdgeConv(self.nn_conv2,aggr = 'add')

        self.nn_conv3 = torch.nn.Sequential(torch.nn.Linear(l3*2,l4),torch.
     nn.LeakyReLU(),torch.nn.Linear(l4,l3),torch.nn.LeakyReLU()).to(device)

        self.conv_add3 = EdgeConv(self.nn_conv3,aggr = 'add')

        self.nn_conv4 = torch.nn.Sequential(torch.nn.Linear(l3*2,l4),torch.
     nn.LeakyReLU(),torch.nn.Linear(l4,l3),torch.nn.LeakyReLU()).to(device)

        self.conv_add4 = EdgeConv(self.nn_conv4,aggr = 'add')

        self.edgepool=EdgePooling(l1,add_to_edge_score=0.5,)

        self.nn1 = torch.nn.Linear(l3*4+ l1,l4)
        self.nn2   = torch.nn.Linear(l4,l5)
        self.nn3 =  torch.nn.Linear(4*l5,l6)
        self.nn4 = torch.nn.Linear(l6,l7)
        self.relu = torch.nn.LeakyReLU()
        self.tanh = torch.nn.Tanh()
        self.Softmax=torch.nn.Softmax(dim=0)

    def forward(self, data):
        k = self.k
        device = self.device
        mode   = self.mode
        pos_idx = self.pos_idx
        x, edge_index, batch = data.x, data.edge_index, data.batch
        x,edge_index,batch,_=self.edgepool(x,edge_index,batch)
        edge_index = knn_graph(x=x[:,pos_idx],k=k[0],batch=batch).to(device)


        a = self.conv_add(x,edge_index)


        edge_index = knn_graph(x=a[:,pos_idx],k=k[1],batch=batch).to(device)

        b = self.conv_add2(a,edge_index)

        edge_index = knn_graph(x=b[:,pos_idx],k=k[2],batch=batch).to(device)

        c = self.conv_add3(b,edge_index)

        edge_index = knn_graph(x=c[:,pos_idx],k=k[3],batch=batch).to(device)

        d = self.conv_add4(c,edge_index)

        x = torch.cat((x,a,b,c,d),dim = 1)
        del a,b,c,d
        x = self.nn1(x)
        x = self.relu(x)
        x = self.nn2(x)


        x=self.pool4cat(x,batch)
        x = self.relu(x)
        x = self.nn3(x)

        x = self.relu(x)
        x = self.nn4(x)

        if mode == 'angle':
            x[:,0] = self.tanh(x[:,0])
            x[:,1] = self.tanh(x[:,1])


        return x
    def pool4cat(self,x,batch):
        a,_ = scatter_max(x, batch, dim = 0)
        b,_ = scatter_min(x, batch, dim = 0)
        c = scatter_sum(x,batch,dim = 0)
        d = scatter_mean(x,batch,dim= 0)
        x = torch.cat((a,b,c,d),dim = 1)
        return x
```

Dynedge Comparison

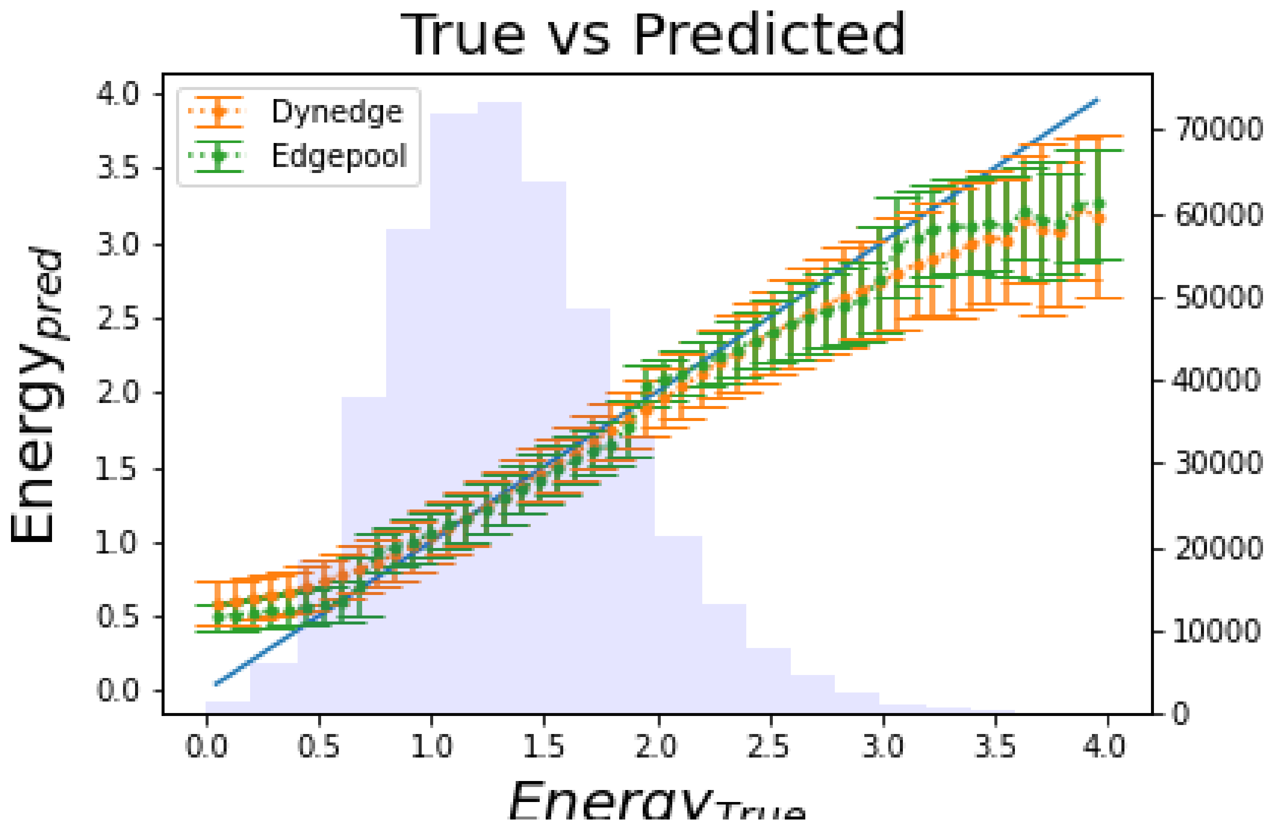
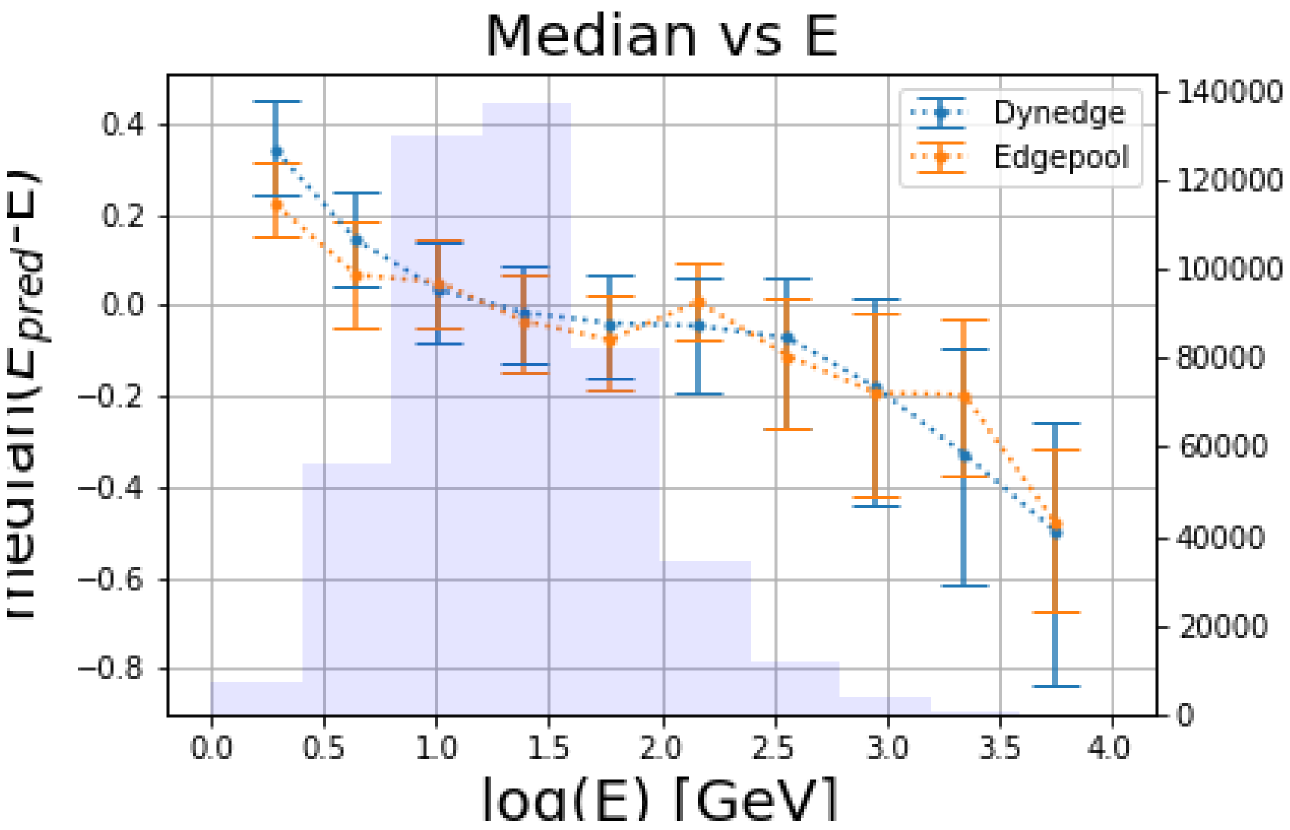


Figure 1: A direct comparison of predicted values as a function of true values. The error bars are chosen as the standard deviation of the prediction.The blue line is a theoretical perfect fit.



,

Figure 2: The median of the Error. The error bars are the width of the error distribution.

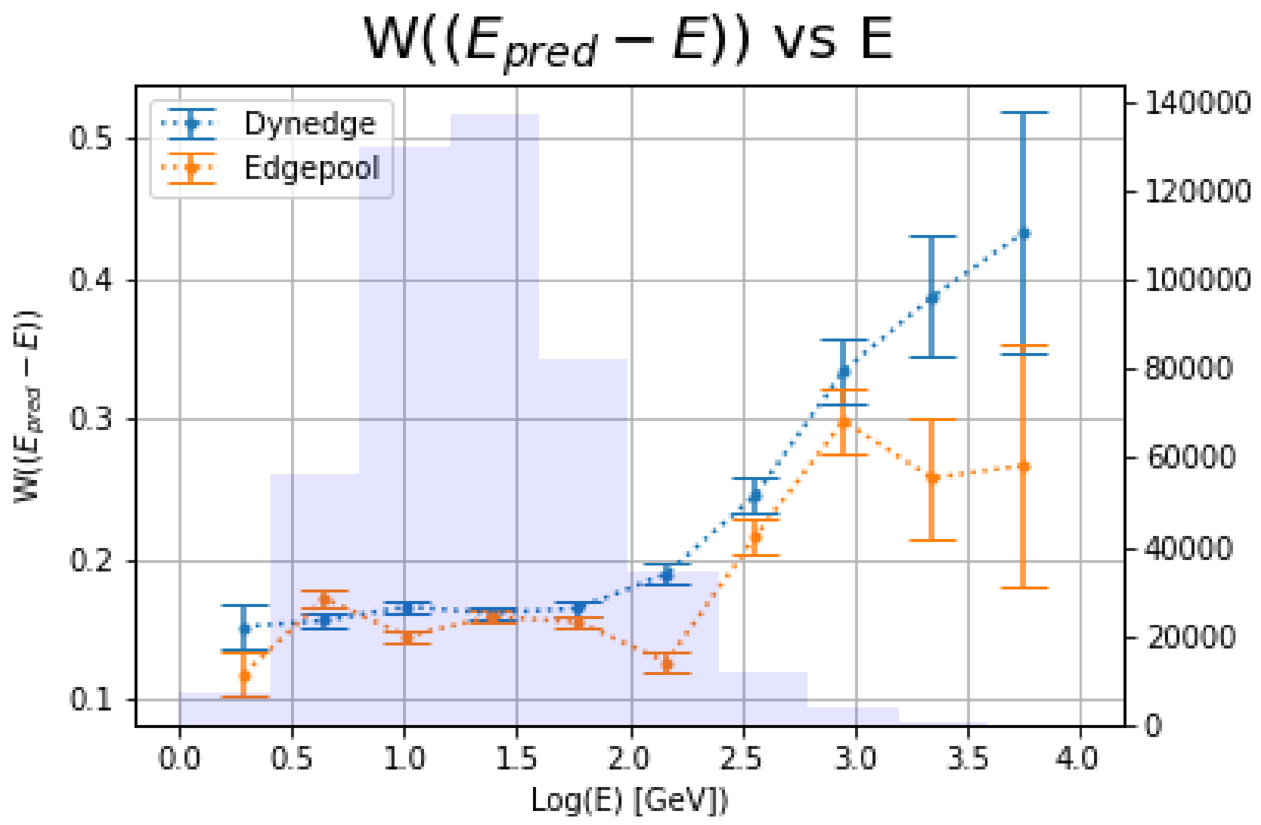


Figure 3: The width of the error distribution. The error bars are the errors on the width.

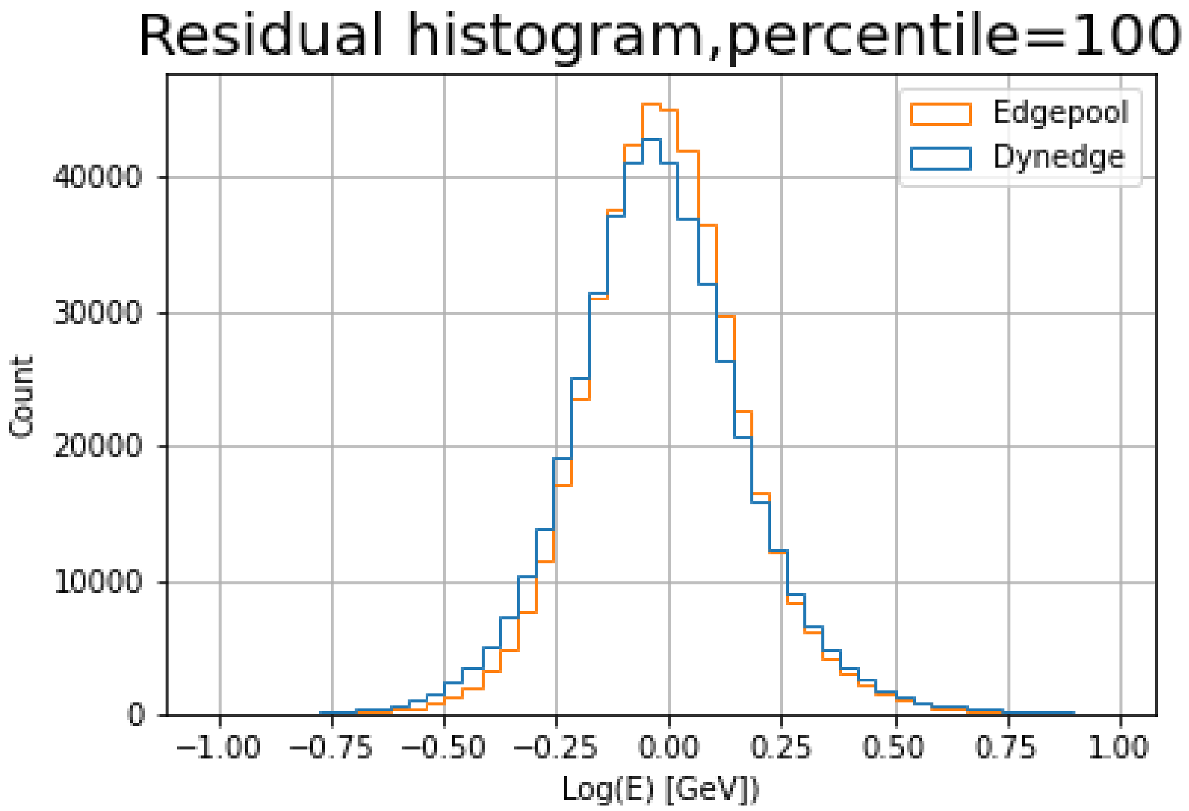


Figure 4: A histogram of the residual values. Note that $retro_{reco}$ is not centered.

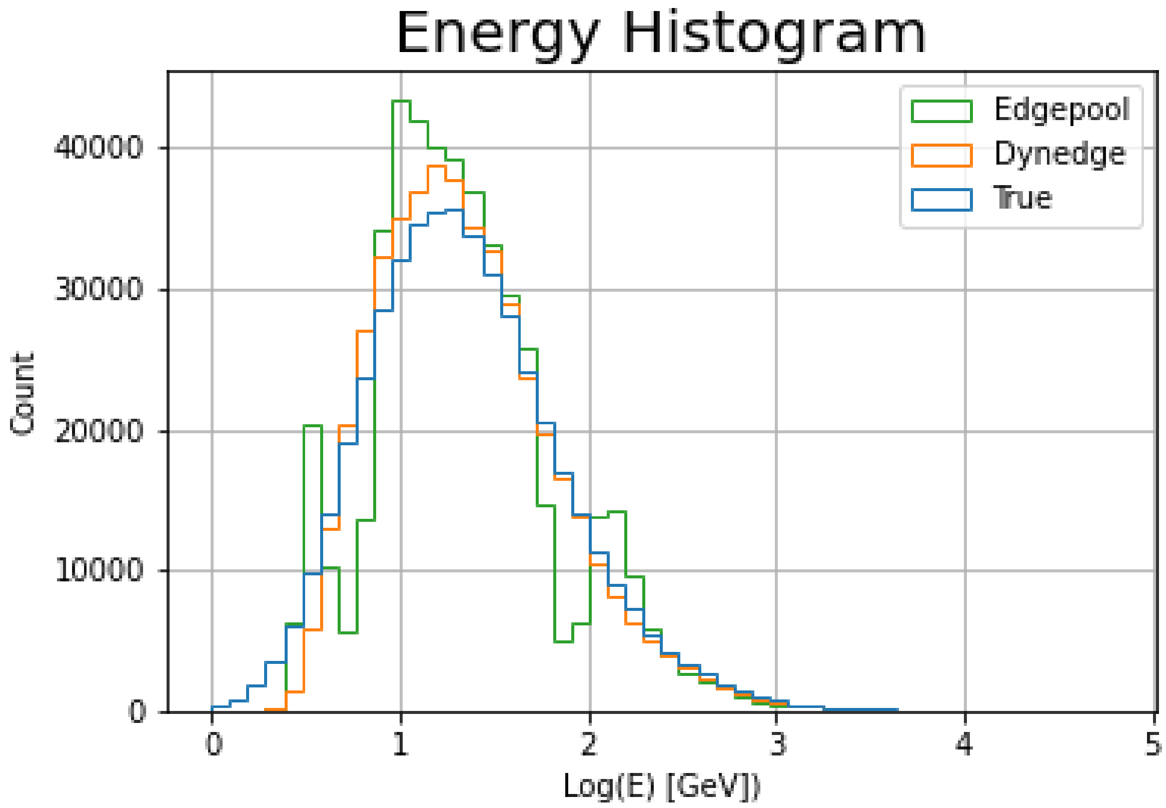


Figure 5: Histograms of reconstructed values.

Github repository

The code used to create graphs, run models and plot can be found at `https://github.com/sinjako/Neutrino-reconstruction-GNN`. It requires pytorch, pytorch-geometric and sqlite packages installed with python version 3.7 or greater. The graphsaving module and the various Run_model modules are run using multiprocessing, with Run_model modules using both CPU and GPU's concurrently. The CPU's are used to feed data into the GPU to reduce idling time.

Bibliography

John N. Bahcall. “solar neutrinos. i. theoretical. Physical review Letters page 300-302, March 1964.

ATLAS collaboration. "observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc”. https://arxiv.org/abs/1207.7214, July 2012.

SNO Collaboration. “direct evidence for neutrino flavor transformation from neutral-current interactions in the sudbury neutrino observatory”. Physical Review Letters 89.1, June 2002.

Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980, December 2014.

Frederik Diehl. "edge contraction pooling for graph neural networks ”. https://arxiv.org/pdf/1905.10990.pdf, May 2019.

C. Andreopoulos et al.). "the genie neutrino monte carlo generator”. https://www.sciencedirect.com/science/article/abs/pii/S0168900209023043?via%3Dihub, February 2010.

K. N. Abazajian et al. "light sterile neutrinos: A white paper”. https://arxiv.org/abs/1204.5379, April 2012.

M. G. Aartsen et al. “measurement of atmospheric neutrino oscillations at 6– 56 gev with icecube deepcore”. https://arxiv.org/abs/0803.0531, February 2018.

M.G AArtsen et al. "measurement of atmospheric tau neutrino appearance with icecube deepcore ”. https://journals.aps.org/prd/abstract/10.1103/PhysRevD.99.032007, February 2019.

Rongjie Lai Feng-Lei Fan and Ge Wang. Quasi-equivalence of width and depth of neural networks. https://assets.researchsquare.com/files/rs-92324/v1_stamped.pdf?c=1603402581, October 2020.

Aya Ishihara (for the IceCube Collaboration). “the icecube upgrade – design and science goals e”. https://arxiv.org/abs/1908.09441, August 2019.

Wassily Hoeffding. "probability inequalities for sums of bounded random variables". https://www.tandfonline.com/doi/abs/10.1080/01621459.1963.10500830, 2012.

Thomas N. Kipf and Max welling. "semi-supervised classification with graph convolutional networks ". https://arxiv.org/abs/1609.02907, September 2016.

Mads Ehrhorn Kjær. "convolutional neural network neutrino reconstruction in icecube", December 2020.

Sachin Kumar and Yulia Tsvetkov. "von mises-fisher loss for training sequence to sequence models with continuous outputs ". https://arxiv.org/pdf/1812.04616.pdf, March 2019.

Pavel Kurasov. Graph laplacians and topology. https://projecteuclid.org/journals/arkiv-for-matematik/volume-46/issue-1/Graph-Laplacians-and-topology/10.1007/s11512-007-0059-4.full, 2008.

T. Sjostrand L. Garren, I.G. Knowles and T. Trippe). "monte carlo particle numbering scheme". https://link.springer.com/article/10.1007/BF02683426, January 2000.

B.R Martin. "nuclear and particle physics". Wiley Online books, March 2006.

Shengjie Min, Zhan Gao, Jing Peng, Liang Wang, Ke Qin, and Bo Fang. "stgsn — a spatial–temporal graph neural network framework for time-evolving social networks ". https://arxiv.org/pdf/1905.10990.pdf, February 2021.

oscNext team. "oscnext". https://wiki.icecube.wisc.edu/images/8/82/OscNext_v00.04.pdf, April 2020.

B. Pontecorvo. "direct evidence for neutrino flavor transformation from neutral-current interactions in the sudbury neutrino observatory". https://inspirehep.net/literature/42736, October 1957.

K. Scholberg. "the supernova early warning system". https://arxiv.org/abs/0803.0531, March 2008.

J. W. F. Valle. "neutrino physics overview". Journal of Physics: Conference Series 53 page 473-505, November 2006.

Yongbin Sun et al. Yue Wang. "dynamic graph cnn for learning on point clouds ". https://arxiv.org/abs/1801.07829, June 2018.

Masami Nakagawa Ziro Maki and Shoichi Sakata. “remarks on the unified model of elementary particles". Progress of Theoretical Physics 28.5 page 870-880, November 1962.