UNIVERSITY OF COPENHAGEN
FACULTY OF SCIENCE

”

Master’s thesis

Mads Ehrhorn Kjær

Convolutional neural network neutrino reconstruction in IceCube

Supervisor: Troels Petersen

Handed in: December 30, 2020

Contents

Acknowledgements

Thank you to my supervisor, Troels Petersen, for many interesting discussion and so much support during very difficult times; you made physics exciting, the hardship tolerable and provided much needed help and input.

To Bjørn Mølvig a special thanks: you gave me plenty of laughs, lots of friendship, and you made me better in the process.

Tom Stuttard provided IceCube data and expertise, without which no thesis could be made.

Oswin Krause, the machine learning wizard, gave invaluable help in creating the convolutional neural networks

This work is entirely dedicated to my mother and father; I could not have done this, nor survived the past seven years, without you. Thank you.

Abstract

The IceCube Neutrino Observatory, on the Geographic South Pole, finished construction almost exactly 10 years ago, and due to the project's success is due to receive an upgrade in 2022-23. This is expected to make it a world-leader in measurements of certain neutrino properties, dependent on the energy E and path length traveled L of the detected neutrinos. These properties need to be inferred from the observation, and subsequent reconstruction, of Cherenkov light from secondary particles, currently done using classical methods.

However, the advancement of machine learning (ML) in general—and neural networks in particular—open up the possibility of creating faster, more robust and scalable inference capabilities, as has been seen in areas such as image recognition and machine translation during the past decade.

This thesis develops a convolutional based neural network and accompanying tooling for low-energy neutrino reconstruction. A novel approach to data management in IceCube specifically for ML purposes is introduced, together with a new event viewer, and algorithm comparison dashboard, to aid in the training and comparison of different network performances, generalizable for use by other groups studying ML applications in IceCube.

A temporal convolutional network is able to outperform the current best classical algorithm Retro Reco at low energies up to around 50 GeV for both polar angle (a proxy for L) and E reconstruction.

Chapter 1

Introduction

Deep beneath the Amundsen-Scott station on the geographic South Pole, lurks a telescope in the ice. This observatory hunts for some of the most elusive particles in our universe, hoping they may answer foundational questions about nature, and glean insight into new physics that the Genevan collider—or its potential successors—may never be able to.

The IceCube Neutrino Observatory was built between 2005 and 2010, but its heritage stretches back several decades. Spanning one cubic kilometer of Antarctic ice, the telescope consists of some 86 strings, hot water drilled into the ice, with each string containing a number of photomultiplier tubes, capable of detecting photons originating from processes involving neutrinos—the aforementioned elusive particles.

Measurements of certain neutrino properties may help us understand phenomena such as why the universe is matter-dominated [1] and dark matter [2]. IceCube is also capable of searching for point-like sources [3], and serves in the SuperNova Early Warning System [4].

But one thing is the ability to observe derived photons in Antarctic ice; another is reconstruction from this light to infer properties of the originating particles. To date, classical methods, such as PegLeg [5] and Retro Reco[a] have mostly been used for this purpose, but the burgeoning proliferation of the use of machine learning (ML) for scientific discovery [6] inspires the development of new methods for this cause.

Whereas classical methods require a hypothesis and thus preset logic, ML "changes these [problems] from logic problems to statistical problems"[b]. That is to say, given a large enough—or rather, a representative enough—dataset, an ML algorithm may find statistical similarities in disparate neutrino events sharing similar underlying properties, giving the network an ability to predict these same properties for previously unseen data. As a "machine learning model is almost like a pure function at inference time"[c], owing to the universal approximation theorems, it should be possible to construct an ML reconstruction algorithm that is both faster, preciser, more extensible, captures a wider spectrum, easier to update, and is more future-proof than the current methods.

The intention of this thesis is to explore that proposition, specifically for up-going muon neutrinos in the sub-teraelectronvolt range.

[a] `https://github.com/IceCubeOpenSource/retro`

[b] `https://www.ben-evans.com/benedictevans/2019/9/6/face-recognition`

[c] Brennan Saeta, Software Engineer—TensorFlow, `https://www.swiftbysundell.com/podcast/58/`

This problem is selected as it is the focus of the oscillations group at IceCube, studying neutrino oscillations. The current best reconstruction algorithm for this case, Retro Reco, can take minutes to reconstruct a neutrino, while an ML-based approach can be expected to reconstruct several thousand each second. An additional focus will be the IceCube upgrade [7], which will install additional improved detectors, and for which there is no current functioning reconstruction algorithm.

1.1 Flow

To weave a common thread through this document, and ensure the reader is informed about the big picture throughout, the following section presents the flow of the thesis.

The story of an interaction in IceCube from physics to analysis proceeds at a very high level as follows: The universe is created with mechanisms that only allow a certain chirality of neutrinos to interact with forces other than gravity (section 2.2 on page 13). At some point an interaction will create such a particle (section 2.3 on page 15), which oscillates between different flavors (section 2.4 on page 16). As chance allows some neutrinos to interact in or around the instrumented ice, new particles are created that in turn send out a specific form of light (section 2.5 on page 21) which the photomultiplier tubes (section 3.2 on page 26) can detect. Today each in-ice digital optical module contains one photomultiplier tube. With the IceCube upgrade this changes (section 3.7 on page 33), and new optical modules will be fitted with additional photomultiplier tubes which should help with determining the direction of neutrinos. The detector is set up with certain triggers in differing levels (section 3.3 on page 27), to ensure that not just any light is recorded and saved to disk. Even further processing is done, as the bandwidth from pole to America is quite low, and so care must be taken in the choice of what to actually send down the wires for analysis. When the data arrives for analysis, different level of cleaning takes place to root out noise (section 3.4 on page 29). This pipeline also attempts to classify event types, using decision trees, until finally algorithms—none of them machine learning based—reconstruct neutrino properties of interest such as energy and direction (section 3.4 on page 29), and the events at all levels of cleaning are saved in an IceCube proprietary format, called I3 (section 3.5 on page 31).

With this part done, we turn towards machine learning generalities. First the basics of machine learning in general (section 4.1 on page 35), then deep learning in particular (section 4.2 on page 36). Special attention is given to temporal convolutional neural networks (section 4.3 on page 40), as this is the architecture used. Because loss functions are such an important part of any deep learning project, a section is devoted to this concept as well (section 4.4 on page 43), and for the most part, before data can be meaningfully trained in a neural network, some form of data transformation is required (section 4.5 on page 44).

The last part of the thesis concerns the work by the author. The data format provided by IceCube is not machine learning friendly, so a new solution fixing those shortcomings was built (section 5.3 on page 50). So that the reader is on the same page as the author, plots and metrics used for comparison are explained. The opponent algorithm is detailed (section 3.4 on page 29), before the experiments defining the algorithm is laid out (section 6.4 on page 60).

Chapter 2

Particle physics

Neutrinos are some of the universe's most interesting and mysterious particles, and offer one of the better paths to researching some still unsolved problems in physics. The Large Hadron Collider nailed down the final missing piece of the model, the Higgs boson, but questions still remain; and some of them concern neutrinos.

The masses of the neutrinos themselves is a mystery, while sterile neutrinos is a dark matter candidate and the possible CP-violating phase in the PMNS matrix can help explain baryon asymmetry. Needless to say, the more we discover about this little particle, the better we know the universe.

The concept of neutrino oscillations has the ability to constrain the previously mentioned CP-violating phase of the neutrino mixing matrix [8] and so the better we are able to measure this, the better the constraint will be.

This chapter will seek to outline the physics behind the interactions that are captured in the IceCube detector, including the concept of Cherenkov radiation, which is of critical importance to the process.

First, we shortly need to touch on what The Standard Model of particle physics is.

2.1 The Standard Model

The Standard Model (SM) of particle physics is the foundation of modern particle physics research. It was patched together during the 20th century by scientific collaboration, and tested through inter-country spending on giant particle accelerators.

The theory itself is a gauge quantum field theory, the mathematics of which is not particularly relevant to this thesis, and it describes three out of the four known fundamental interactions. Of these, the so-called weak one is of importance to the neutrino, while being a critical clue to why some particles acquire mass.

The SM describes two types of particles: fermions and bosons, as seen in fig. 2.1a on the following page. The distinguishing feature between the two sets is the statistics they obey; whereas the fermions follow Fermi-Dirac statistics—as a consequence of their half-integer spin—the bosons adhere to Bose-Einstein statistics. Matter in the universe consists of fermions, while the fundamental forces are mediated by the bosons.

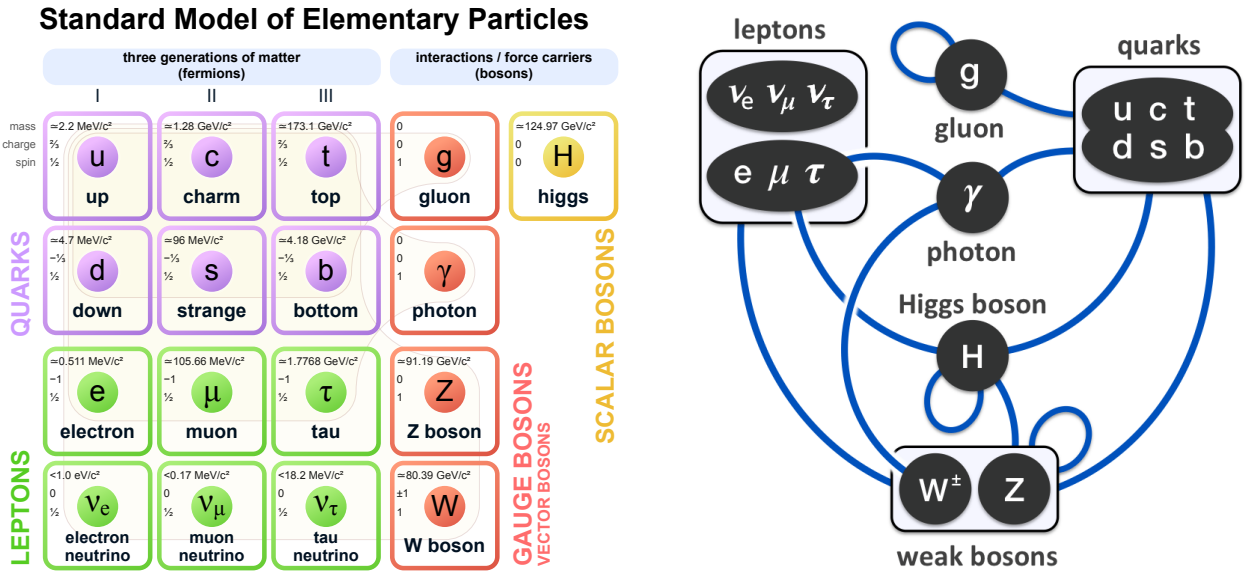


Figure 2.1: (a), from [9], shows the fermions and bosons of The Standard Model of particle physics. (b), from [10], shows the possible interactions in the SM between fermions and bosons.

The fermions are divided between the quarks and the leptons, differentiated by their participation in the strong interaction, carried by the massless gluon. This boson only acts on quarks, and has the peculiar behavior of creating new particles when quarks are pulled apart, resulting in the fact that it is only possible to observe hadrons (a composite particle made up of two or more quarks); this phenomenon, "color confinement", is the progenitor of jets in collider experiments such as the LHC.

Both the charged leptons and the quarks interact electromagnetically and weakly, forces mediated by the photon and the $W^{\pm}$ & Z^0 bosons respectively, but the leptons include a second sub-species that only interacts weakly: the neutral leptons, known as the neutrinos; see fig. 2.1b. Because they do not participate in anything but the weak force, neutrinos hardly ever interact, and are thus difficult (but not impossible!) to observe. An MeV neutrino has a cross section on the order of 1×10^{-44} cm^2, and a beam of such particles must travel through a light year of lead before half of them is blocked.

The Standard Model sports three generations of matter, differing in mass and flavor (a quantum number). The first generation contains the well-known electron and the up & down quarks, the second contains the muon (of great interest to IceCube, as we shall see) and the charm & strange quarks, while the third contains the tau and the top & bottom quarks.

It has been experimentally ascertained that no more than three generations can exist with a corresponding light neutrino of mass less than 45 GeV $c_0{}^{-2}$ [11]; the cause of the universe's choice of three generations is still unknown.

The experimental particle physicist uses Feynman diagrams to describe potential interactions between particles, depending on Fermi's golden rule, the Lorentz-invariant matrix element and for fermions the Dirac equation.

Paul Dirac and others before him searched for an extension of the Schrödinger equation, which incorporated relativity. Requiring adherence to the Einstein energy-momentum relation, Dirac found the matrix equation

$$\left(i\gamma^\mu \partial_\mu - m\right)\psi = 0, \tag{2.1}$$

which requires a four-component wave function (not incidentally explaining spin and anti-particles). This is the governing equation of all fermions.

Experiments generally proceed by calculating expected particle rates, and using either accelerators or passive detectors to compare reality to theory using Fermi's golden rule,

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

describing transition rates from an initial state $|i\rangle$ to a final state $|f\rangle$. The equation depends on the transition matrix T_{fi} from the expansion of the interaction with the perturbation Hamiltonian, with ρ the energy-dependent density of states. Equation (2.2) can be used to calculate the differential cross section

$$\frac{\mathrm{d}\sigma}{\mathrm{d}\Omega^*} = \frac{1}{64\pi^2 s}\frac{\mathrm{p}_f^*}{\mathrm{p}_i^*}\left|\mathcal{M}_{fi}\right|^2, \tag{2.3}$$

an expression of quantum mechanical probabilities for an interaction to happen, valid in the center-of-mass frame (COM), indicated by $*$. s is the squared COM energy, p the particle momenta, and Ω the solid angle a particle scatters into.

The SM thus provides observables, as long as one can calculate the Lorentz invariant matrix element $\mathcal{M}$, which in the fermion case can be done using eq. (2.1). This process is much simplified by Feynman diagrams, allowing the theorist to draw processes and using Feynman rules stitch together a matrix element describing the interaction, thus allowing one to (relatively) easily carry out complex cross section calculations by "simply" drawing diagrams! Examples of Feynman diagrams can be seen in fig. 2.4 on page 15.

2.2 The weak interaction

In the early 20th century the world of nuclear physics was abuzz. A great many experiments were conducted after Becquerel discovered radioactivity in 1896, some of which explored beta decay, an example of which can be seen in fig. 2.4b on page 15.

In this process a neutron turns into a proton, emitting an electron and an anti-neutrino. This was unexplained without the neutrino, because of two problems: First, the energy spectrum seemed continuous (see fig. 2.2 on the following page), unexpected as the energy should be mono-energetic if only the electron is emitted from the atom. Second, the electron is a spin-½ particle, while nitrogen-14, which showed beta decay, is a spin-1 particle; so where goes the spin?

Both issues were explained by the "neutron" (later renamed neutrino by Fermi) proposed by Pauli in 1930. The discrete energy spectrum now turns continuous, owing to the neutrino carrying away unseen energy, and the spin can be attributed to it as well, as long as the neutrino is a spin-½ particle.

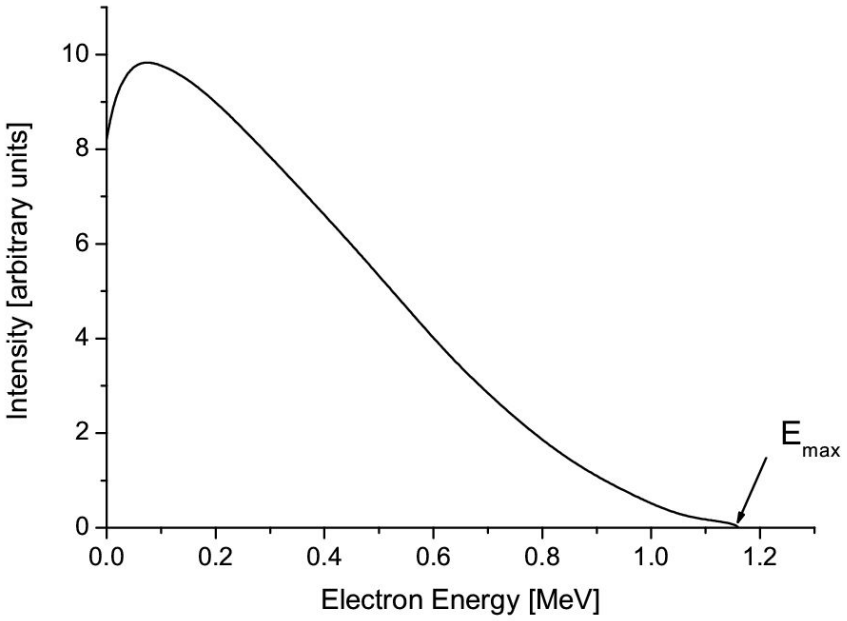
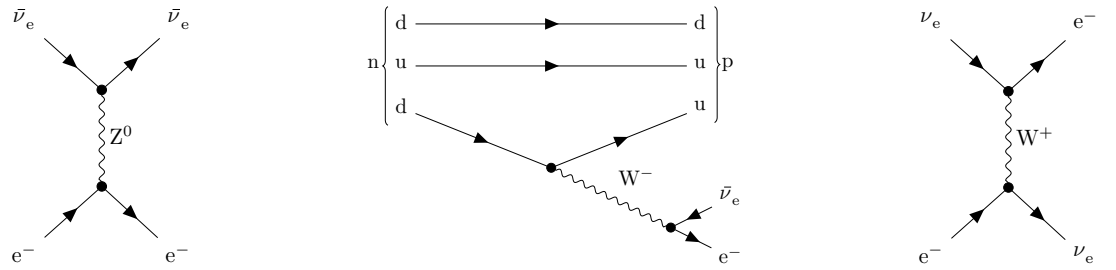


Figure 2.2: Energy spectrum of ^{210}B beta decay, with $E_{\max} = 1.16\,\text{MeV}$. Image from [12].

Figure 2.3: Helicity. Image from [14].

Fermi explained this with his theory of beta decay, proposing a four-fermion contact interaction, a direct interaction between the four particles. Although a successful theory, it runs into the same kind of ultraviolet problem that Planck worked on in the late 19th century: because the Fermi interaction cross section goes as $\sigma \approx G_{\text{F}}^2 E^2$, where G_{F} is the Fermi constant, it grows unbounded, leading to problems at higher energies. Additionally the weak interaction was observed in the Wu experiment [13] to violate parity, as beta decay of cobalt-60 showed a preferential decay direction for electrons, a phenomenon not explained by Fermi's theory. This was solved by the electroweak theory, introducing bosons that mediate the interaction over small distances after symmetry breaking via the Higgs mechanism; the W and Z bosons. The parity violation is explained by the V-A (vector minus axial vector) nature of the weak force, which only permits it to couple with left-handed particles and right-handed antiparticles; this concept is enshrined mathematically by γ^5 of the Dirac matrix family, formed as the product of the gamma matrices in eq. (2.1) on the previous page, which enters in the so-called chiral projection operator.

Describing this operator is made easier by reference to helicity states, which are more easily grasped conceptually. A helicity state is described in terms of a particle's component of spin along its



(a) Neutral current **(b)** β^- decay **(c)** Charged current

Figure 2.4: Feynman diagrams of neutral- and charged current processes

momentum, $h \equiv \mathbf{S} \cdot \mathbf{p}/\mathrm{p}$; see fig. 2.3 on the facing page. Chirality, on the other hand, is defined as the eigenstates of $\gamma^5 = i\gamma^0\gamma^1\gamma^2\gamma^3$, and in the ultra-relativistic regime, where $E \gg m$, the domain of neutrinos, helicity and chiral eigenstates are the same. This becomes important in the weak interaction, whose vertex factor from the matrix element is given as

$$\frac{ig_\mathrm{W}}{\sqrt{2}}\frac{1}{2}\gamma^\mu\left(1-\gamma^5\right), \tag{2.4}$$

where $1/2(1-\gamma^5)$ is known as the left-handed chiral projection operator. When this operates on a right handed chiral (or helicity, in the high energy limit) state, the result is zero, meaning only left-handed chiral particle states (and right-handed anti-particle states) may participate in the weak current.

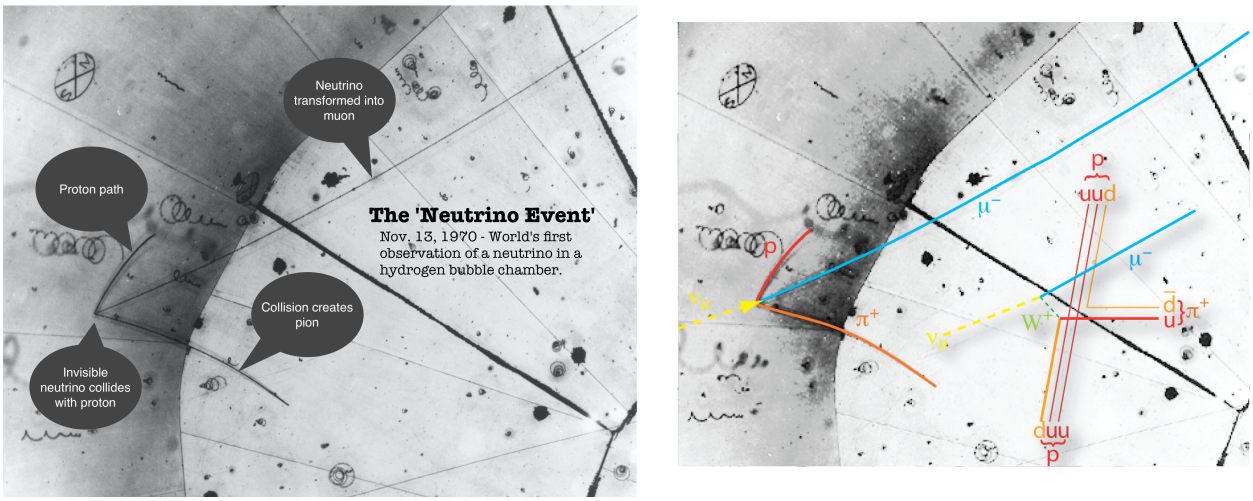
An intriguing possibility is presented by this property: as only (anti) neutrinos of one handedness interact weakly, there may be a whole universe of non-interacting neutrinos that only participate gravitationally. These "sterile neutrinos" are candidates for dark matter [15].

The interaction is termed "weak" as a consequence of its coupling strength, relative to the electromagnetic- and strong forces. This is due to the massive nature of its mediating bosons, giving them a shorter lifetime than they would possess otherwise. Furthermore, two of the bosons contain electric charge (the $\mathrm{W}^\pm$ bosons), while the last one is electrically neutral (the Z^0 boson), leading to the phenomenon of charged- and neutral currents.

2.3 Charged- and neutral currents

The nature of the weak force allows two fundamental types of interactions: charged- and neutral currents.

The charged current, as the Feynman diagram in fig. 2.4c shows, allows a charged lepton to turn into a neutral one. By absorbing a W boson, an electron may turn into an electron neutrino, as seen in fig. 2.4c, or—as in the case of beta decay, fig. 2.4b—a down quark may convert into an up quark, changing a neutrino into a proton, with the resulting W^- boson decaying into an electron and antielectron neutrino, conserving charge and lepton number. A startling real world example of the charged current in action is seen in fig. 2.5 on the following page, where a neutrino of sufficient energy collides with a proton, creating both a muon and hadronic decay products.



(a) (b)

Figure 2.5: The first observed neutrino event in history, from the Argonne National Laboratory 12-foot bubble chamber, on Friday the 13th of November 1970. It is clearly seen on fig. 2.5a that some unseen particle collides with a proton, and two new particles are created in the process. Figure 2.5b makes the process clearer using a Feynman diagram: the W boson interacts with a quark in the proton. The neutrino has enough energy to create an inelastic scattering, which results in an intermediate state (a Delta baryon), which quickly decays via the strong force into a proton and a pion.

The neutral current, on the other hand, seen in fig. 2.4a on the previous page, can deflect two particles, but does not change any of the properties of the interacting particles other than transferring momentum and spin.

In this way the weak interaction is special among the fundamental forces; it is the only one that is able to change flavors, it is the only one to couple to neutrinos, and it is—as we shall see—the only one that allows us to define left- and right-handedness unambiguously.

2.4 Neutrino oscillations

In the early days, the weak coupling strength of quarks was found to be smaller than that of lepton interactions. Not only that, but different quark interactions have different coupling strengths. Nicola Cabibbo developed a framework for explaining this discrepancy, wherein so-called weak eigenstates are distinct from the mass eigenstates that describe the quarks outside of weak interactions [16]. For example, both down- and strange quarks may decay into an up quark, giving a weak eigenstate

$$\mathrm{d}' = V_{\mathrm{ud}}\mathrm{d} + V_{\mathrm{us}}\mathrm{s}, \tag{2.5}$$

with $|V_{\mathrm{ud}}|^2$ the probability of a down quark decaying, and $|V_{\mathrm{us}}|^2$ the probability of a strange quark decaying.

The weak eigenstates, a superposition of down and strange quarks, is then the entity actually coupling to the up quark, and extending this to include the decay of charm quarks yields the Cabibbo matrix

$$\begin{bmatrix} \mathrm{d}' \\ \mathrm{s}' \end{bmatrix} = \begin{bmatrix} V_{\mathrm{ud}} & V_{\mathrm{us}} \\ V_{\mathrm{cd}} & V_{\mathrm{cs}} \end{bmatrix} \begin{bmatrix} \mathrm{d} \\ \mathrm{s} \end{bmatrix} = \begin{bmatrix} \cos\theta_{\mathrm{c}} & \sin\theta_{\mathrm{c}} \\ -\sin\theta_{\mathrm{c}} & \cos\theta_{\mathrm{c}} \end{bmatrix} \begin{bmatrix} \mathrm{d} \\ \mathrm{s} \end{bmatrix}, \tag{2.6}$$

with θ_{c} the Cabibbo angle.

The concept was extended to neutrino oscillations. This Nobel-winning idea was used to explain the apparent failure of the Homestake experiment, also known as the solar neutrino problem, described shortly in the following [16].

The Sun burns hydrogen via many processes, primarily in the pp cycle, wherein two protons fuse to create deuterium, a positron, an electron neutrino and excess energy:

$$\mathrm{p} + \mathrm{p} \longrightarrow \nu_{\mathrm{e}} + 1.442\,\mathrm{MeV}. \tag{2.7}$$

Other interactions occur, some (such as the β-decay of boron-8) producing neutrinos of energies sufficient to be detected on Earth, but the Standard Solar Model predicts that the Sun only produces electron neutrinos, the flux predicted to be on the order of $2 \times 10^{38}\ \mathrm{s}^{-1}$ [16]. Testing this model should be somewhat straightforward: fill an underground tank with appropriate liquid, wait for neutrino interactions and count how many atoms have changed. The tank needs to be underground to shield from cosmic rays other than solar neutrinos, the liquid should be smartly chosen such that it decays into something easily measurable, and the tank needs to be big enough for interactions to happen on a reasonable timescale. This describes the Homestake experiment: a tank was made out of a closed underground mine, and filled with perchloroethylene, containing plenty of chlorine. The interaction with a neutrino then proceeds as

$$\nu_{\mathrm{e}} + {}^{37}\mathrm{Cl} \longrightarrow {}^{37}\mathrm{Ar} + \mathrm{e}^{-}. \tag{2.8}$$

The argon produced by the reaction is radioactive, and can thus be counted.

The experiment, however, saw a deficit of neutrinos. This, of course, can have many causes: the calculations may be wrong, the experimental setup may be faulty or theories involved may be wrong. Raymond Davis, the experiment-runner, stood his ground and vouched for the correctness of the experimental setup, and some 30-odd years later the SNO experiment found that the *total* flux of neutrinos from the Sun (that is, the electron, muon and tau variants) was on the order of the predicted electron neutrino flux. It would thus seem that something had to give: either the Standard Solar Model is wrong, or neutrinos change flavor during propagation from the Sun to the Earth.

Extending the Cabibbo hypothesis to account for this, we have

$$\begin{bmatrix} \nu_{\mathrm{e}} \\ \nu_{\mu} \end{bmatrix} = \begin{bmatrix} U_{\mathrm{e}1} & U_{\mathrm{e}2} \\ U_{\mu 1} & U_{\mu 2} \end{bmatrix} \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix}. \tag{2.9}$$

Now the weak eigenstates are described as superpositions of the mass eigenstates ν_1 and ν_2, and a muon neutrino produced at time $t = 0$ will have the wave function

$$|\psi(0)\rangle = U_{\mu 1}\,|\nu_1\rangle + U_{\mu 2}\,|\nu_2\rangle = \cos\theta\,|\nu_2\rangle - \sin\theta\,|\nu_1\rangle\,. \tag{2.10}$$

Having traveled L at time T it will be described as

$$|\psi(L,T)\rangle = \cos\theta\,|\nu_2\rangle\, e^{-i\phi_2} - \sin\theta\,|\nu_1\rangle\, e^{-i\phi_1}, \tag{2.11}$$

with the phases $\phi_i = p_i \cdot x = E_i T - \mathrm{p}_i L$. Expressed in terms of the weak eigenstates, eq. (2.11) can be written as

$$|\psi(L,T)\rangle = e^{-i\phi_1}\left[\left(e^{i\Delta\phi_{12}} - 1\right)\cos\theta\sin\theta\,|\nu_\mathrm{e}\rangle - \left(e^{i\Delta\phi_{12}}\cos^2\theta + \sin^2\theta\right)|\nu_\mu\rangle\right], \tag{2.12}$$

where $\Delta\phi_{12} = \phi_1 - \phi_2$, showing that any non-zero value of $\Delta\phi_{12}$ will incur mixing of the weak eigenstates in the wave function of the propagating neutrino. Simplifying eq. (2.12), we can write it as

$$|\psi(L,T)\rangle = c_\mathrm{e}\,|\nu_\mathrm{e}\rangle + c_\mu\,|\nu_\mu\rangle, \tag{2.13}$$

and it is seen that the probability of a muon neutrino turning into an electron neutrino becomes

$$P(\nu_\mu \to \nu_\mathrm{e}) = |\langle\nu_\mathrm{e}|\psi(L,T)\rangle|^2 = c_\mathrm{e}c_\mathrm{e}^*. \tag{2.14}$$

Using eq. (2.12) this becomes

$$P(\nu_\mu \to \nu_\mathrm{e}) = \sin^2(2\theta)\sin^2\left(\frac{\Delta\phi_{12}}{2}\right), \tag{2.15}$$

which in the ultra-relativistic limit becomes

$$P(\nu_\mu \to \nu_\mathrm{e}) = \sin^2(2\theta)\sin^2\left[\frac{\left(m_1^2 - m_2^2\right)L}{4E_\nu}\right] = \sin^2(2\theta)\sin^2\left(\Delta_{12}\right), \tag{2.16}$$

with

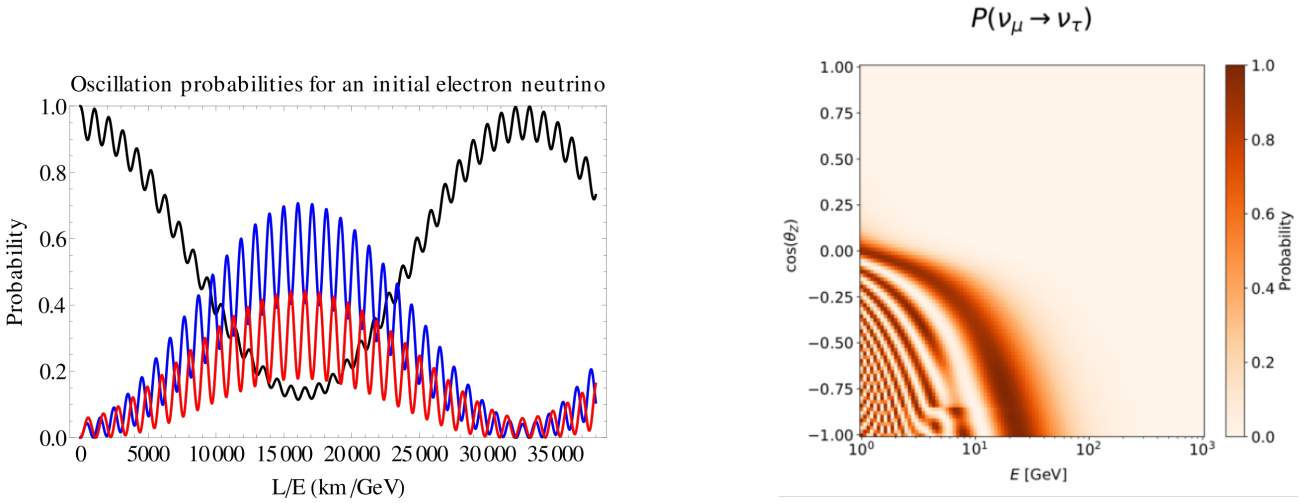
$$\Delta_{ij} = \frac{\left(m_i^2 - m_j^2\right)L}{4E_\nu}. \tag{2.17}$$

In plain words, then, the probability, or oscillation, depends firstly on the mixing angle, secondly the difference of the squared masses of the mass eigenstates, thirdly the energy of the neutrino and lastly the length it travels. An example of oscillation patterns can be seen in fig. 2.6 on the next page.

In many cases one uses the survival probability, which is

$$P(\nu_\mu \to \nu_\mu) = c_\mathrm{e}c_\mathrm{e}^* = 1 - P(\nu_\mu \to \nu_\mathrm{e}). \tag{2.18}$$

The two-flavor treatment is sufficient for many purposes, but nature has bestowed upon us three generations of matter (or, at least, three generations of light neutrinos). To accommodate this fact, it is



(a) **(b)**

Figure 2.6: (a), from [17], shows the theoretical calculation of neutrino oscillations on the long baseline. Here, an electron neutrino is created (black line), and has some probability of turning into a muon neutrino (blue line) and tau neutrino (red line). (b), from [18], shows the muon neutrino to tau neutrino probability in IceCube, a function of both energy and pathlength (parameterized by the cosine of the zenith angle of the neutrino). Here one can see the effects of neutrinos interacting with matter, the Mikheyev Smirnov Wolfenstein (MSW) effect, changing the energy levels of the mass eigenstates during propagation on interactions with electrons in the dense core of the Earth. This changes the oscillation profile, seen for up-going neutrinos at energies below 10 GeV.

possible to extend eq. (2.9) on page 17 to

$$\begin{bmatrix} \nu_e \\ \nu_\mu \\ \nu_\tau \end{bmatrix} = \begin{bmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{bmatrix} \begin{bmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{bmatrix}. \tag{2.19}$$

This unitary matrix can be parameterized as

$$\begin{bmatrix} U_{e1} & U_{e2} & U_{e3} \\ U_{\mu 1} & U_{\mu 2} & U_{\mu 3} \\ U_{\tau 1} & U_{\tau 2} & U_{\tau 3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_{23} & s_{23} \\ 0 & -s_{23} & c_{23} \end{bmatrix} \begin{bmatrix} c_{13} & 0 & s_{13}e^{-i\delta} \\ 0 & 1 & 0 \\ -s_{13}e^{i\delta} & 0 & c_{13} \end{bmatrix} \begin{bmatrix} c_{12} & s_{12} & 0 \\ -s_{12} & c_{12} & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.20}$$

where $c_{ij} = \cos\theta_{ij}$, $s_{ij} = \sin\theta_{ij}$ and δ is a complex phase owing to the unitarity condition of the matrix[a]. Moreover, eq. (2.18) on the preceding page now becomes

$$P(\nu_\mu \to \nu_\mu) = 1 - 4|U_{\mu 1}|^2|U_{\mu 2}|^2 \sin^2\Delta_{21} - 4|U_{\mu 1}|^2|U_{\mu 3}|^2 \sin^2\Delta_{31} - 4|U_{\mu 2}|^2|U_{\mu 3}|^2 \sin^2\Delta_{32}, \tag{2.21}$$

showing the possibility of experimentally determining the values of the neutrino mixing matrix, provided one can set up an experiment where enough neutrinos are captured, among other things.

[a] In fact the matrix requires six complex phases to specify all nine independent parameters. However, five of these can be absorbed into the definitions of the lepton phases.

In the case of IceCube, this survival probability can be approximated as[19]

$$P(\nu_\mu \to \nu_\mu) \approx 1 - \sin^2\left(2\theta_{23}\right) \sin^2\left(1.27\frac{\Delta m^2_{32} L}{E}\right). \tag{2.22}$$

Incidentally, the lepton mixing matrix also provides the possibility of CP violation by the weak force. In this case, applying a CP transformation on a right-handed anti-neutrino results in a left-handed neutrino, allowed in the matrix element of weak interactions, and the asymmetry in this case is then

$$P(\nu_\mathrm{e} \to \nu_\mu) - P(\bar{\nu}_\mathrm{e} \to \bar{\nu}_\mu) = 16\Im\left\{U^*_{\mathrm{e}1} U_{\mu 1} U_{\mathrm{e}2} U^*_{\mu 2}\right\} \sin\Delta_{12} \sin\Delta_{13} \sin\Delta_{23}. \tag{2.23}$$

Therefore, if the complex phase δ of the PMNS matrix is zero, there will be no CP violation in the neutrino sector. CP violation is required for explaining the baryon asymmetry, and has been ruled out of both the strong- and the electromagnetic interactions, and so it is hoped that neutrino oscillations will provide the answer.

This of course also means that if we observe neutrino oscillations, it cannot be a massless particle. However, the observable in oscillation probabilities is the difference of the squared masses, meaning it is the only thing we can experimentally say anything about. As of this writing, the best fit values, from [20], is on the order of

$$\Delta m^2_{21} \approx 7.4 \times 10^{-5}\,\mathrm{eV}^2 \tag{2.24}$$

$$|\Delta m^2_{32}| \approx 2.5 \times 10^{-3}\,\mathrm{eV}^2. \tag{2.25}$$

The absolute mass scale is constrained by the density of relic neutrinos from The Big Bang to be [16]

$$\sum_{i=1}^{3} m_{\nu_i} \lesssim 1\,\mathrm{eV}. \tag{2.26}$$

Because we only know the mass differences, it is not a priori possible to say in which order the mass hierarchy goes. All we can say currently, is that the difference between ν_1 and ν_2 is much less than the difference between ν_3 and ν_2, but we do not know if ν_3 is heavier than the other two flavors. The mass ordering problem can tested using IceCube [5].

This should be cause for some consternation, or—perhaps more—excitement. First of all, the Standard Model does not automatically describe neutrino mass, and if it is introduced into the Lagrangian it requires that right-handed chiral neutrinos exist.

Equation (2.21) on the preceding page depends on different Δs. Because of eq. (2.24) it is reasonable to approximate $\Delta_{31} \approx \Delta 32$, which means the survival probability depends on Δ_{21} and Δ_{32}. This leads to two types of oscillations, the long and short baselines, that develop on two dissimilar length scales; this can be seen in fig. 2.6a on the preceding page, where two characteristic oscillation patterns can be seen for each neutrino type.

Even so, the masses of the neutrinos are on the order of 1×10^6 smaller than the electron (see fig. 2.7 on the next page), and so something else than the Higgs mechanism could be the cause of their mass—this is exciting! Suggestions, such as the seesaw mechanism [16], attempt to explain this, by

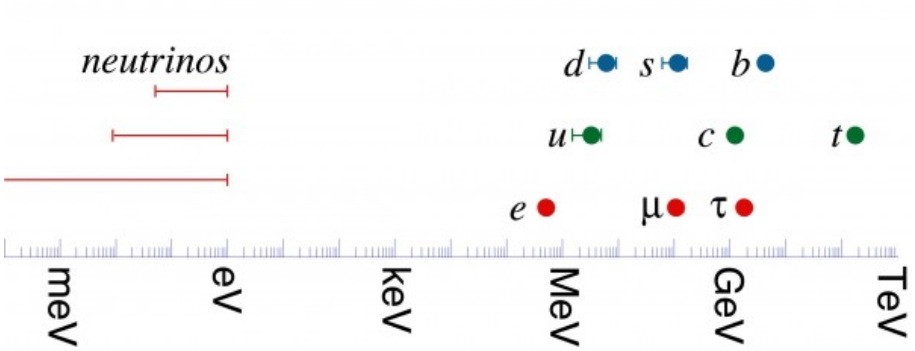


Figure 2.7: Particle mass scales. The neutrinos are a million times lighter than the electron, the lightest charged lepton. The top quark also stands out in the hierarchy as a very heavy particle. Image from [21].

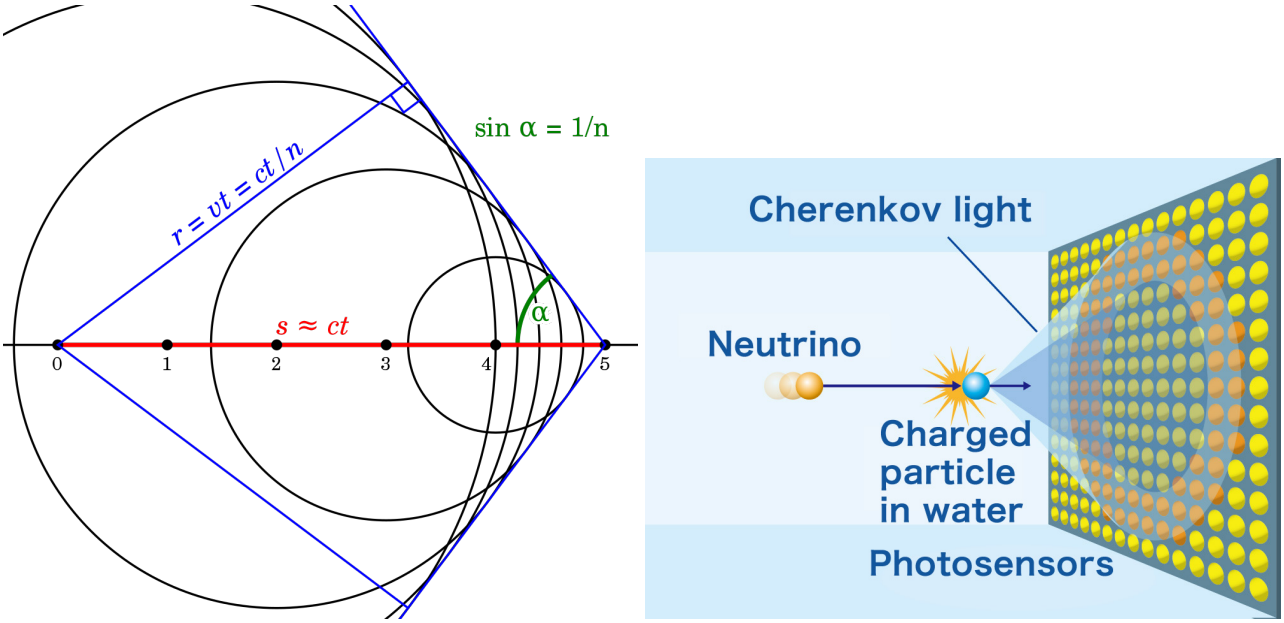
introducing a heavy neutrino partner to each one of the lighter types. Additionally, neutrinos and anti-neutrinos may be the same particle, a so-called Majorana particle.

All this to say that the study of neutrinos provide plenty of exciting avenues of physics beyond the Standard Model, and IceCube is in an excellent position to provide answers, and as such the quality of the reconstruction of the incident neutrinos, especially the energy and the zenith angle, is of the highest importance.

2.5 Cherenkov radiation

Light, in a vacuum, moves at the universally constant speed of c When traveling through a medium, the refractive index $n = c/v$ (where v is the phase velocity of light in the medium) determines the phase velocity of the photons due to constructive interference with the atoms of the material. In a dielectric medium, a charged particle may travel faster than light, and will in its path polarize the molecules of the material. After the passage, the material relaxes into an unpolarized state, sending out photons in the process, and because these travel at the refracted speed of light they conspire to create light at an angle with the particle where constructive interference occurs; see fig. 2.8a on the following page. Here, a particle traveling at $v_\mathrm{p} \approx c$ emits Cherenkov radiation at an angle $\sin\alpha = r/s = v/v_\mathrm{p}$, and so the angle depends on the index of refraction and the speed of the charged particle.

Of course the angle produces a cone in three dimensional space. This is used in particle detectors such as Super-Kamiokande where a tank of liquid is surrounded by an array of photomultiplier tubes. These will pick up Cherenkov cones as seen in fig. 2.8b on the next page, and from the shapes particle properties can be inferred which in turn may shed light on properties of the neutrinos that originated the interaction; not unlike IceCube, where the Cherenkov light is what is actually observed by the photomultiplier tubes.



(a) (b)

Figure 2.8: (a), from [22], shows the geometry involved in Cherenkov radiation. (b), from [23], shows how a Cherenkov detector uses this radiation to infer particle properties.

Chapter 3

IceCube

One of IceCube's raison d'êtres is the study of neutrinos, its primary tool the conversion of light pulses into particle properties. This is not an easy task, inasmuch as nature rarely exhibits the simplicity of its governing equations. IceCube consists of two primary detectors, IceCube and DeepCore, with a third, "the Upgrade", expected to be installed in the next few years.

3.1 Detector

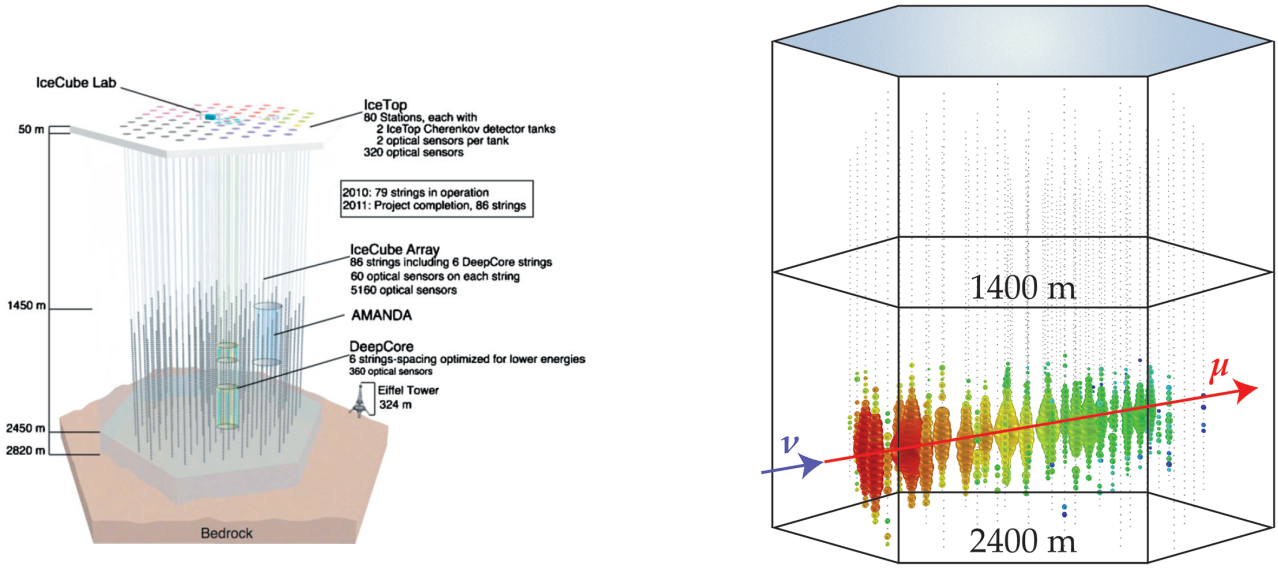
As we have seen, the neutrino only participates weakly in the Standard Model. This makes the neutrino a very elusive particle, able to generally travel very far before interacting. The rate of neutrino interactions is a function of the flux ϕ, the cross section σ, and the number of target particles N_{target}, i.e.

$$N_\nu \propto \phi \times \sigma \times N_{\text{target}}, \tag{3.1}$$

which means that if your flux is $2 \times 10^{38}\ \text{s}^{-1}$, and the cross section $1 \times 10^{-44}\ \text{cm}^2$, you, to paraphrase Roy Schneider, are gonna need a bigger boat. But if one thing is a big enough detector, containing a sufficient number of target particles, another is background noise: you generally want to ensure that the only thing interacting within your detector volume is by-products of neutrinos. In many cases these secondary particles are muons, but those are plentiful in nature from other sources, and your detector will thus require screening from the outside world.

There are generally two ways of solving this. One is to build a container and fill it up with whatever substance suits the purpose, and then finding somewhere to stow it away such that it is shielded from outside charged leptons, typically done by positioning the container inside an abandoned mine deep underground. The other way is to utilize nature to its fullest, and find somewhere that both provides the interaction substance *and* shields from unwanted particles.

The Homestake experiment, mentioned in section 2.4 on page 16, made use of the first type of detector in the late 1960s. Many others—such as the Kamioka Observatory in Japan, and the Sudbury Neutrino Observatory in Canada—followed this path, but a race of sorts was soon on to build a detector of the second variety; detailed in [24]. Starting with DUMAND in the 70s, one set of scientists were focused on water Cherenkov detectors. In this case one suspends strings of detectors in deep water to observe Cherenkov radiation by charged particles travelling faster than c through the water. DUMAND was never successfully completed, but another team—led by Francis Halzen—was inspired to use ice instead of water, and in 1996, a year after DUMAND's cancellation, set up AMANDA on the South



(a) **(b)**

Figure 3.1: (a), from [25], shows an artist rendition of the IceCube detector as a scale comparison to the Eiffel Tower. (b), from [26], shows a neutrino event in IceCube. This is a high energy track-like event, for which angular reconstruction is possible even by eye.

Pole. When this proved a success, funding for the IceCube Neutrino Observatory was approved, and construction completed in 2010.

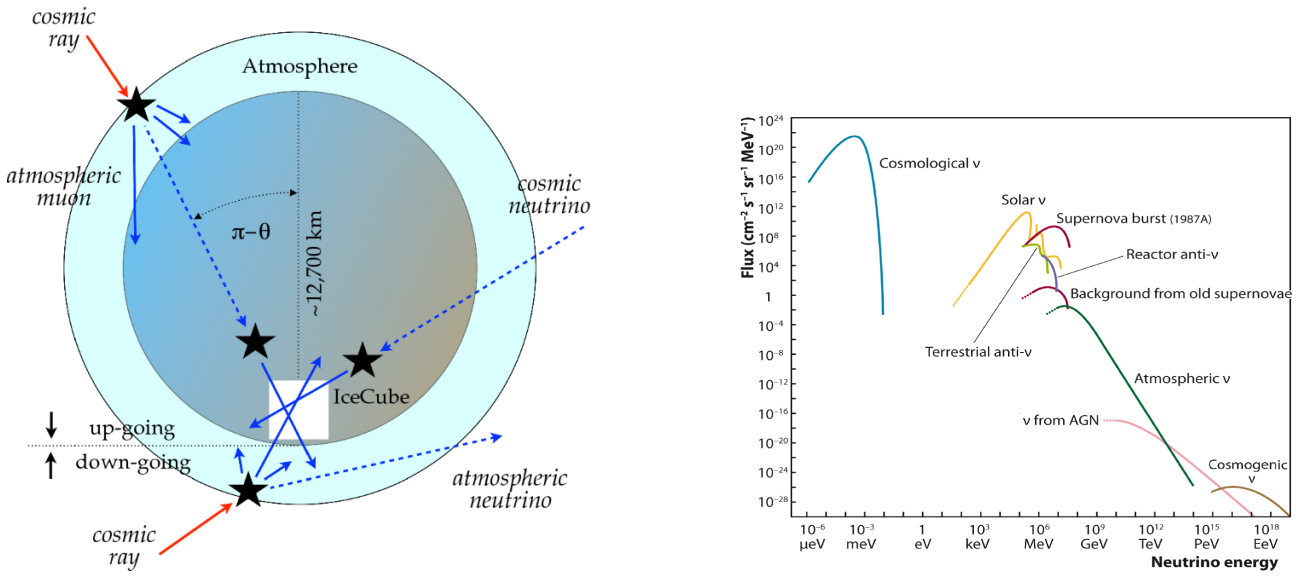
The IceCube instigators recognized that the Antarctic ice provides an excellent source of neutrino interaction material, and the Earth provides a near perfect shield from background muons.

The detector is built by using hot water to drill holes in the Antarctic ice 2.5 km deep, and lowering down long strings on which are situated the actual Cherenkov light detectors. Each string consists of 60 of these detectors, called digital optical modules (DOMs), and each DOM consists of assorted electronics, a photomultiplier tube that does the actual light detection, and shielding from the environment. The DOM is further detailed in section 3.2 on page 26.

IceCube is split into the in-ice array, i.e. the stringed DOMs, and an IceTop array. This consists of detectors on the surface, housing water Cherenkov detectors that may act as a useful veto on particles: if they were seen by IceTop, they were probably down-going particles, and thus background, while the neutrino events present as up-going particles created via interactions in the Earth, where only neutrinos penetrate; see fig. 3.2a on the facing page. The detector is tuned to particles in the GeV to PeV range, where atmospheric neutrinos dominate, as seen in fig. 3.2b on the next page.

IceCube records data based on certain triggers (section 3.3 on page 27), and sends filtered data to the Northern Hemisphere by satellite (about $100\,\mathrm{GB\,d^{-1}}$ [29]), with the raw data sent by sneakernet[a] (about $1\,\mathrm{TB\,d^{-1}}$ [29]). The surface also hosts the IceCube Lab, containing supporting electronics such as computers and communication systems. The whole detector is located near, and supported by, the Amundsen-Scott South Pole station, situated at the Geographic South Pole and administered by the US National Science Foundation.

[a]That is, data transported by sneakers, i.e. shipped via physical media rather than the internet.



(a) **(b)**

Figure 3.2: (a), from [27], shows the types of rays relevant to IceCube, defining up- and down-going muons. Neutrinos—created by atmospheric interaction with cosmic rays or in cosmic processes—interact either in the atmosphere (creating background muons), or in the Earth (creating up-going muons). (b), from [28], shows the energy spectrum of neutrinos from different sources. Atmospheric neutrinos dominate in the energy region supported by IceCube.

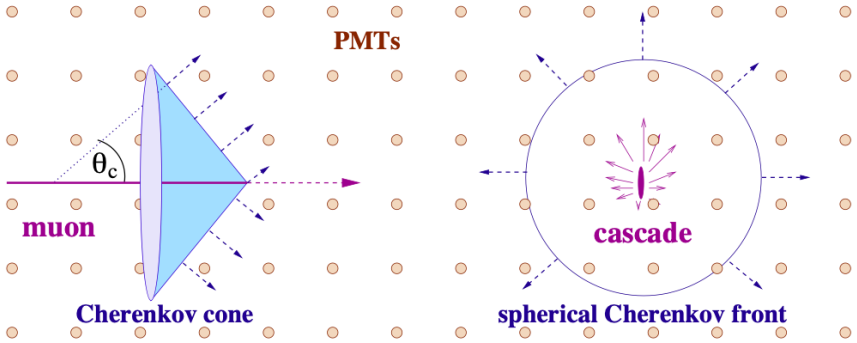


Figure 3.3: Detection modes of the AMANDA detector (built following the same principles as IceCube). This makes clear that the Cherenkov light is what is actually observed, from a muon in the left figure, and a cascade in the right. The track-like signatures are detected from Cherenkov cones, while the cascades present as Cherenkov spheres. Image from [30].

The observatory detects Cherenkov light from secondary particles—visualized in fig. 3.3 on the previous page—from both charged- and neutral current neutrino interactions, with the large volume of ice providing a plentiful basin of potentially interacting ice molecules. In the case of neutral current interactions, the event signature is that of a cascade: the neutrino scatters of a charged lepton, escaping the detector again, with the deposited energy resulting in electromagnetic and hadronic cascades; hence the name. An electromagnetic cascade (or "shower") comes from a high energy electron suffering bremsstrahlung, radiating a photon which creates an electron-positron pair. This pair production again experiences bremsstrahlung, and the process continues until the energy is spent. Hadronic cascades result from the nuclear interaction of hadrons (composite particles made up of two or more quarks) and the concept of color confinement which disallows a quark from existing alone; it requires less energy to create a companion quark from the void. These cascades can be contained fully within the detector, meaning the energy can be easily inferred, but it is difficult to reconstruct a track that points to the origin of the neutrino. Charged current (CC) interactions, on the other hand, present the possibility of track-like signals, which make for a much better pointing resolution. However, the flavor of the neutrino in this case determines the signature:

- An electron neutrino CC interaction will produce an electron, which will create an electromagnetic shower, as the electron instigates pair production, and is slowed down by bremsstrahlung. This looks like a cascade signature
- A tau lepton will live very shortly (on the order of 1×10^{-13} s) owing to its massive nature, and decays again either hadronically (creating a cascade) or leptonically (creating neutrinos and a muon or an electron, with about equal probability)
- Muons are special: they do not decay immediately, as taus, and are less affected by the processes stopping electrons. They are thus able to leave long tracks in the detector, a track-like signature simply because their dominant energy loss mechanism is ionisation, whilst electrons also lose energy to bremsstrahlung

All three flavors create a hadronic cascade at the interaction vertex, X in $\nu_\alpha + N \rightarrow \ell_\alpha + X$ with N a nucleon, ν_α a neutrino and ℓ_α a lepton, both of flavor α.

In the case of a tau neutrino, there is the possibility of another (as of yet unseen) signature: the double-bang. A tau will have a decay length of $50\,\mathrm{m\,PeV^{-1}}$ [27], so very high energy tau neutrinos may interact and first create a hadronic cascade, then a track from a high energy tau, which then decays 50 m or more away. The signatures can be seen in fig. 3.4 on the facing page.

3.2 DOMs

The DOMs are situated on 2.5 km long strings, spaced 17 m apart (7 m for DeepCore, 2.4 m for Upgrade). They are essentially in-ice, high quality light detectors, capable of detecting single photons [32]. Being self-contained in the sense that all recording capabilities are on-device, and according to pre-set triggers (section 3.3 on the next page), they will record and save pulses and transmit them via cabling to IceCube Lab on the surface. When a photon is detected, the DOM starts a recording, lasting for up to 6.4 µs, such that later arriving photons are included in the "hit" [32]. This operation is independent, and as such each DOM does not know what other DOMs are seeing. However, the PMT hit rate is recorded in millisecond intervals, because an overall increase in hits might indicate a

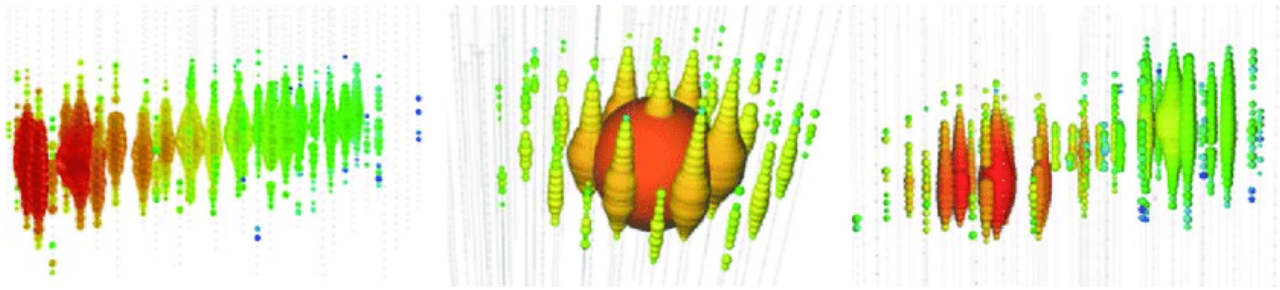


Figure 3.4: Three IceCube event signatures. The size of each colored sphere is proportional to the charge of the pulse detected by the PMT, and the color is related to the timing of the signal; red hits are earlier than blue. The left figure shows a track-like muon event, while the signature in the middle is a cascade. On the right is shown a double-bang event, where two cascades and a track is seen. The two left-most events are real, the double-bang is simulated. Image from [31].

cosmic event, which must be acted upon promptly, and a separate local coincidence wiring—connecting closest DOMs—ensures quick recognition of (near) simultaneous hits [32].

The PMT waveforms are digitized by two circuits, ATWD (Analog Transient Waveform Digitizer) and fADC (fast Analog to Digital Converter). These differ in their sampling time, 427 ns for ATWD, 6.4 µs for fADC. Different conditions leads to different payloads sent from the DOM to the surface; if the local coincidence is triggered, both the fADC and the ATWD waveforms are sent.

The DOMs also contain "flasher boards" outfitted with LEDs, used for calibration purposes such as measuring their positions and the optical properties of the ice.

It should be noted that the DOMs are pretty reliable; by 2016 87 DOMs out of 5160 were not operational [32]; this is rather fortuitous, inasmuch as they are non-repairable (at least they're impossible to retrieve from the ice).

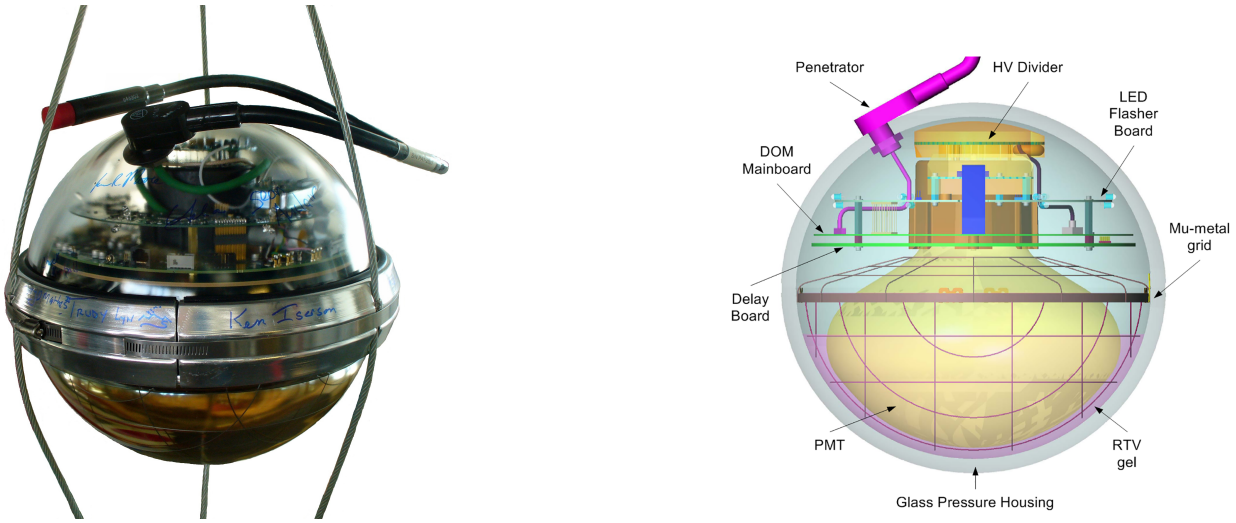
The location of the detector, in the dark ice deep beneath the Antarctic, leads to the majority of noise in the PMTs being "dark noise", that is quantum effects such as thermal emission, radioactive decays and the luminescence of the glass; these effects have no outside sources, but have a rate of 560–780 Hz [32]. The background muons, for comparison, are at a rate of 5–25 Hz [32].

Interestingly, the noise rate decreases after installation, as the DOMs "freeze in" [32].

As can be seen in fig. 3.5b on the following page, the entire southern hemisphere of the DOMs is taken up by the PMT. Accordingly, all DOMs point downwards, a design choice changed in the Upgrade (section 3.7 on page 33). Communication and power is handled by the cabling—clearly seen in fig. 3.5a on the following page—, connecting all DOMs on a string, terminating on the surface in the IceCube Lab.

3.3 Triggers and on-pole reconstruction

As any other particle detector, IceCube employs triggers to maintain the otherwise unmanageable data levels at a reasonable size. The raw data rate, after triggers have been applied, is 1 TB d^{-1}, which is whittled down to around 100 GB by event selections performed by the on-Pole servers, appropriate for



(a) (b)

Figure 3.5: (a), from [33], shows a picture of a DOM, (b) [34] shows a schematic of a DOM.

transfer via satellite to the Northern Hemisphere [29], where events can be analyzed further. There are several triggers at work, with different functionalities in mind; one trigger, the "Single Multiplicity Trigger" (SMT), uses the local coincidence wiring and acts when N or more locally coincident hits (LC) are detected within a sliding window n µs, extended until the local coincidence condition is no longer satisfied. With $N = 8$ and $n = 5$ the hit rate of this particular trigger is 2100 Hz [29]. Another is the string trigger, looking for a certain number of hits along a string, useful for slow-moving, completely up-going, muons. In general, the triggers rely on the speed of light in ice to figure out whether hits are causally connected, or appear to be noise. Additionally, because an event may fulfill the conditions set out in several triggers, hit-merging is performed, ensuring that unique hits are not present in several events. On the other extreme, events can be coincident, needing algorithms to split events from each other.

When triggers have been activated, detector data is sent to IceCube Lab, which uses on-premise servers to perform fast event selection and preliminary reconstruction to determine if a muon is up- or down-going and whether several muons are present in one event. The reconstruction is important as a filter, because the largest source of noise in the detector comes from background atmospheric muons. Using earth-filtering is the easiest way of determining whether a muon is atmospheric or caused by a neutrino, as the ground stops all of the former, but allows passage of the latter, but this does present a challenge: if one wants to determine (approximately) whence a muon came, one needs to assemble the disparate hits in the detector into something resembling a line. As several thousand events are read out each second, the algorithm must be fast, and fast on the available hardware (which tends not to be replaced often, seeing as how it is situated at one of the hardest to reach locations on Earth). The servers at the IceCube Lab perform first a quick reconstruction, "linefit" [35]—a minimizer using a Huber penalty—used as a seed for the "Single-Photo-Electron-Fit" (SPE), a maximum likelihood estimation [30]. This reconstruction has a median angular resolution (the arc-distance between reconstruction and the true track) on Monte Carlo (MC, section 5.1 on page 45) data of $\theta_{\text{med}} = 4.2°$ [35].

IceCube has two sources of unwanted noise, when detecting neutrinos: background atmospheric muons produced by cosmic rays, which trigger the detector at a rate of around 3000 kHz [36], and noise

produced directly in the DOMs. This noise has two components, thermal and non-thermal [37]. The thermal variety stems from spontaneous emission of single electrons from the photocathodes, while the non-thermal comes from radioactive decays in the DOM glass and sees a rate of 500–800 Hz [29].

3.4 Post processing

When data arrives from the Pole, different analysis groups apply different post-processing measures. In the following we shall focus on the methods applied by the oscillation working group, a suite of processing software termed "oscNext", documented in [37]. Six levels are used, each removing different types of noise and muons using several different methods (some of which are machine learning based). The muon and noise is reduced by a factor $O(10^5)$ by the post-processing, and at the final level a neutrino rate of $O(1\,\text{mHz})$ survives with a few % background muons and almost no noise. After all cuts, at the final level, between 22 % and 36 % simulated neutrinos remain, depending on current type and flavor. In the case of charged current muon neutrinos, at level 2 the rate is 6.16 mHz, while at level 5 it is 1.39 mHz, with a neutrino efficiency of 22.6 % [37].

The post-processing levels are shortly touched on in the following.

3.4.1 L2

L2 is run on files received from the Pole. It first passes finds events passing the SMT3 trigger, meaning the SMT described in section 3.3 on page 27 with $N = 3$. Next, a Seeded Radius-Time (SRT) cleaning algorithm is run, which relies on the fact that Cherenkov photons will be clustered in space and time, while noise is random. The SRT algorithm uses this on the LC hits, used as seeds, with non-LC hits outside some pre-set space-time volume discarded, while those inside are kept and used as seeds for further iteration. Lastly, a DeepCore fiducial volume, and a DeepCore veto volume are defined, and using a center of gravity calculation using hits in the fiducial volume, hits from the veto region are discarded based on their derived velocity, on the basis that such a hit might be related to a crossing muon.

3.4.2 L3

At level 3, a number of cuts are made on several derived variables, described in [37]. The main point of this level is to bring the data into agreement with MC, the discrepancies mainly stemming from processes for which there is no simulation. Muons are identified by looking for light in the upper regions of the detector, while noise is found by looking at hit timing or number of hits. Several variables are derived and cut on; examples include `NchCleaned` (number of hit DOMs in the cleaned pulses series) cut at ≥ 6) to identify noise, `VertexGuess Z` (z-coordinate of the first pulse in the cleaned pulses series) cut at $< -120\,\text{m}$ to identify muons and `DCFiducialHits` (the number of hit DeepCore fiducial DOMs in the cleaned pulses series) cut at > 2 to identify noise.

L3 removes around 95 % of the background muon events, 99 % of pure noise and keeps 60 % of atmospheric neutrino events relative to level 2, such that the event rate is below 1 Hz [37].

3.4.3 L4

By level 4 there is good agreement between MC and data, important because at this step machine learning algorithms for classification are introduced. Two classifiers are used, both Boosted Decision Trees (BDT) LightGBM models: one for pure noise, another for background (atmospheric) muons, both trained on a number of derived variables, and the background sample for the muon classifier being real detector data. The pure noise rate is reduced at L4 from 36.6 mHz to under 0.3 mHz with $\approx 96\,\%$ of actual neutrino events intact, while 94 % background muons are removed, keeping 87 % of neutrinos.

Relative to L3, 99 % of pure noise and over 94 % of atmospheric muons events are removed, while 80 % of neutrinos are kept, resulting in roughly a 100 : 10 : 1 (muon : neutrino : noise) ratio.

3.4.4 L5

Level 5's main function is removing so-called sneaky muons. These are particles that "sneak in" to DeepCore, without hitting any IceCube strings, a possibility caused by the hexagonal pattern of the detector. This is done by calculating the COG of an event, finding the closest DeepCore string to that, and if a set number of DOMs along some previously defined poorly instrumented corridors, within some volume cylinder and time, are hit, the event is dropped. Further, SPE is run on the event, and the calculated direction compared to the corridors; if it does not line up with any corridor, the event is kept. A starting containment cut is also employed, cutting events beginning outside, or close to the edge of, DeepCore.

After L5, the rates are 2.16 mHz for *all* neutrinos, 0.93 mHz for atmospheric muons and 0.07 mHz for pure noise.

3.4.5 L6

L6 splits the remaining data into two sets, verification and high statistics. One algorithm (or, rather, two: SANTA/LEERA) is run on the verification set, while another is run on the high statistics set (RetroReco). SANTA is a direction reconstruction algorithm, while LEERA does energy, and they are simpler, although less efficient, than RetroReco, but less impacted by detector uncertainties. They are fast, but require only unscattered photons, and are intended to be used for checking data/MC agreement.

RetroReco, on the other hand, uses a likelihood function, employing table based lookups which simulate photon propagation from PMTs, under the assumption that scattering and absorption are time symmetric processes. The tables take up ≈ 1 TB, but due to this being extremely impractical RAM-wise, some clever tricks are employed to lower memory consumption (which is still significant when compared to older reconstruction algorithms).

Additionally, reconstructions are measured in seconds (see fig. 3.6 on the next page), which makes it seem like an obvious candidate for improvements using machine learning; exactly the point of this thesis, using GENIE-MC generated muon neutrinos, with RetroReco reconstruction, numbering 7,280,939 events in total.

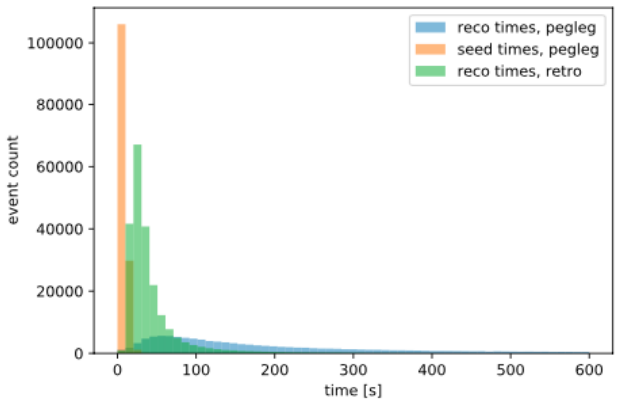


Figure 3.6: Reconstruction comparisons between PegLeg (the previous reconstruction algorithm) and RetroReco (the current). Reconstructions are measured in seconds per event. From [37].

3.5 I3File format

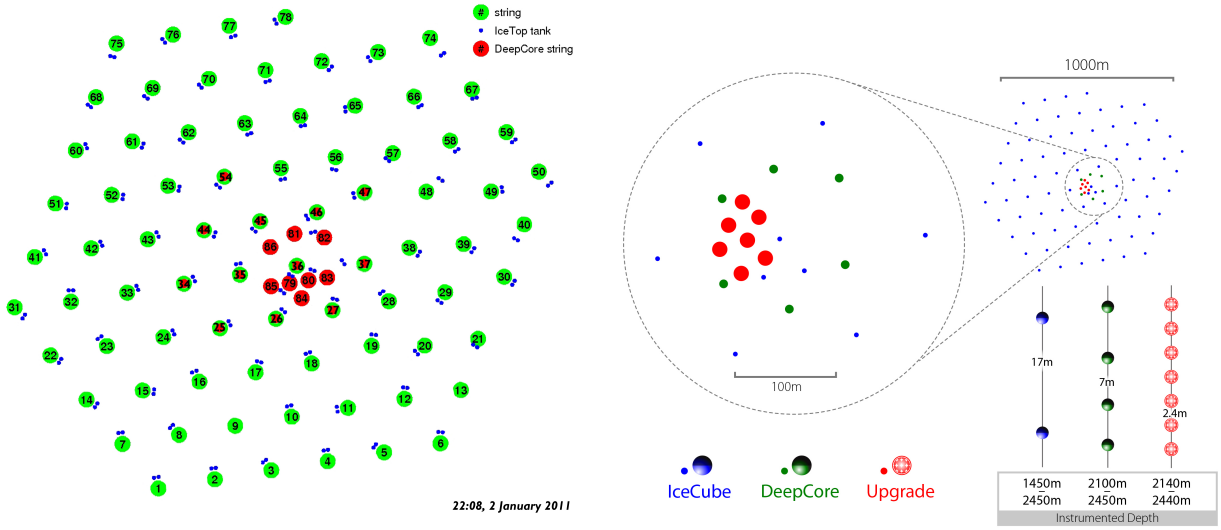
All the following information was retrieved from the IceCube documentation server, at `https://docs.icecube.aq/icerec/V05-01-04/projects/dataio/portable_binary_archive.html`. The author does not know of any scientific paper detailing the I3File format.

The IceCube data, be it from the detector or MC generated, is kept in a portable binary archive format called `I3Files`. A binary file stands in contrast to text files, which include known formats such as XML and JSON. There are pros and cons of this choice: first, binary files tend to be much smaller than text files. This is simply because storing e.g. a 64-bit integer in a text file is done using some form of character encoding (e.g. ASCII) which can take up over 20 bytes, whilst the same number can be stored in a binary format using exactly 8 bytes. Secondly, binary files tend to be much quicker to read, because the information does not need to be decoded. The portability is achieved by being built on the Boost C++ library's archive format, widely available on all major platforms, although I3Files have diverged since, and require an old Boost version.

The files basically consist of so-called I3Frames, which may hold either primitives (such as integers, floats, etc.) or objects. The I3Frames are serialized, and so anything contained within them must be serializable.

An event can be contained within an I3Frame, and will contain information such as pulses and their associated DOMs, indexed by keys, which can then be looked up in a complementary geometry file. Reading the files can conveniently be done through a Python interface called IceTray, enabling the user to use a modern language for processing and analysis.

The format has stood the test of time, and has been in use since its creation in 2006, but there are a couple of issues with it: first (and most minor) of all, it requires the installation of IceCube software on the target machine. This is a minor annoyance, as one can build this in relatively short time, but it does not have to be compiled and is not available easily e.g. via PyPi (and with good reasons, as it contains many dependencies). The second issue—which is rather large for machine learning applications—is I3Files inability to seek in frames. In other words, the file format is sequential, and you have to loop through the frames to find the one you are interested in.



(a) (b)

Figure 3.7: (a), from [38], shows the string pattern of IceCube and DeepCore, (b), from [7], shows the proposed upgrade.

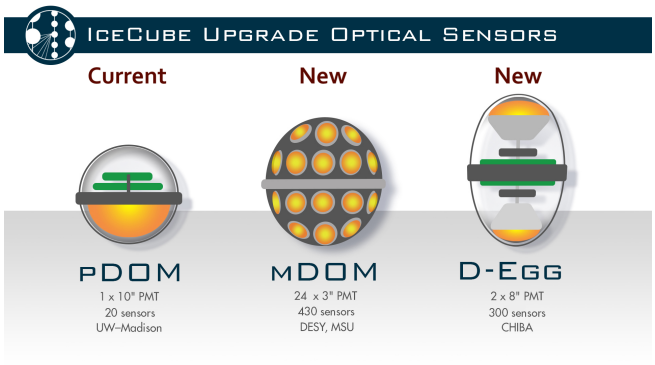
This is unusable for machine learning applications, where it is a requirement that your sets are defined in advance, *and* are easily retrievable on demand, as your training will probably include randomization each epoch.

This was addressed in this project by using SQLite; see section 5.3 on page 50.

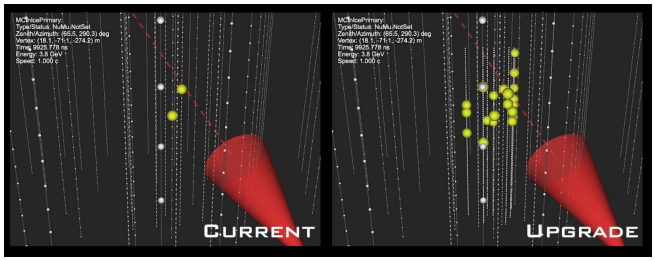
3.6 DeepCore

Before all of IceCube's strings were even deployed—a laborious task only possible during the short austral summer months—it was recognized that the energy range of the detector might not be sufficient for certain analysis purposes. As such, DeepCore was intended to lower the useful energy limit, doing so by installing a new set of strings, closer together, enclosed by IceCube proper. Each new string has a DOM spacing of 7 m vs. IceCube's 17 m, and many PMTs are of an improved quantum efficiency relative to the "old" PMTs.

The IceCube DOMs include DOM-to-DOM communication capabilities, which was used in the early years to measure ice properties. This data was used to determine the optimal placement of DeepCore, around 2100–2450 m, in the clearest ice measured, escaping the antarctic dust layer, left behind some 65,000 years ago [39], which sees a significant uptick in scattering. DeepCore comprises eight new strings, with an average inter-string horizontal spacing of 72 m, and is logically joined with seven near IceCube strings such that the two types on the whole comprise the DeepCore detector; some strings are placed only 42 m apart. Each new DeepCore string has 50 DOMs below 2100 m, with the remaining 10 above the dust layer. All these measures amount to a new lower limit on the neutrino energy useful for analysis to 10 GeV [39]. The top-down geometry can be seen in fig. 3.7a.



(a)



(b)

Figure 3.8: (a) shows new DOM types installed with the IceCube upgrade. Figure 3.8b shows a simulated 3.8 GeV track-like event and its Cherenkov cone. More DOMs light up in the Upgrade configuration, giving more information for track reconstruction. Figures are from the IceCube Collaboration [40].

DeepCore has made possible precision measurements of neutrino oscillations, and has also been used in the hunt for WIMPs[b].

3.7 IceCube Upgrade

The IceCube Upgrade, a prelude to the humongous 8 km^3 IceCube-Gen2, aims to repeat DeepCore's success, and add an extra trick: new DOM designs. The Upgrade features two new types of DOMs, the mDOM and the D-Egg; see fig. 3.8. Whereas IceCube and DeepCore features DOMs whose PMTs only point downward, Upgrade's new DOM types feature instead PMTs that point otherwise. In the simplest case, D-Egg[c], one PMT points up, the other down, while the mDOM[d] possesses 24 PMTs arranged in a spherical fashion.

The seven new Upgrade strings will be deployed in the same clear-ice region as DeepCore, and will feature even closer DOMs that are 3 m apart, and an inter-string spacing of around 20 m. This will enhance the physics potential by enabling an even lower energy threshold, a significantly improved directional precision, more precise energy estimates, and possibly the potential to see double bang events or just statistically identify tau events. Finally, the installation (centered around IceCube string 36) may serve to improve the overall string calibration of IceCube.

[b] Weakly interacting massive particles, a dark matter candidate
[c] Dual optical sensors in an Ellipsoid Glass for Gen2, a somewhat contrived acronym
[d] multi-PMT Digital Optical Module

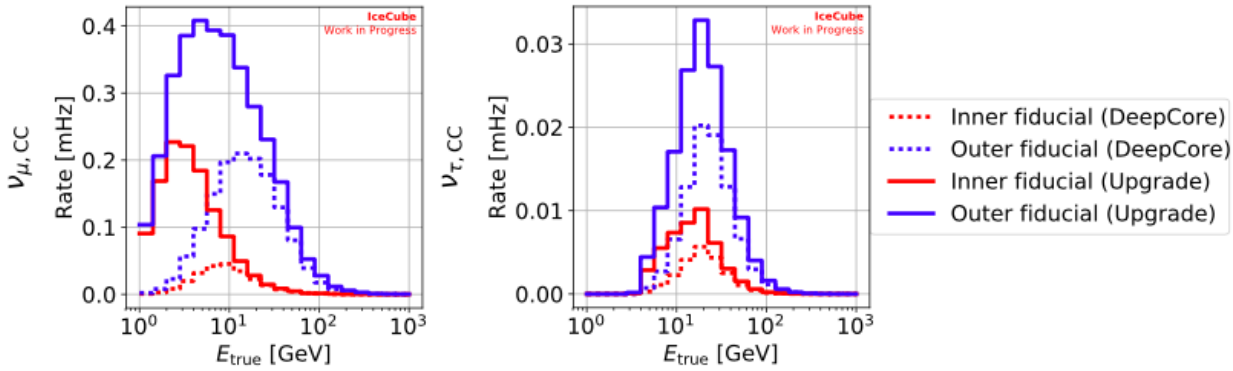


Figure 3.9: The fully contained atmospheric ν_μ and ν_τ rates from charged current interactions in a cylindrical volume with a 50 m radius and a height of 275 m ("inner fiducial"; red lines) or a 145 m radius and same height ("outer fiducial"; blue lines), both centered on DeepCore, for DeepCore (dotted lines) and Upgrade (solid lines). A significant enhancement in rates is seen around the interesting energy range of ≈ 30 GeV. Image from [7].

The low energy capabilities of Upgrade are important to ν_τ oscillation studies, because the muon to tau oscillation,

$$P(\nu_\mu \to \nu_\tau) \approx \sin^2\left(2\theta_{23}\right) \cos^4\left(\theta_{13}\right) \sin^2\left(\frac{\Delta m^2_{31} L}{4E}\right), \tag{3.2}$$

has an oscillation maximum for ν_τ at 25 GeV, with L equal to Earth's diameter[7]; the directionality of the new DOMs provide a better pointing resolution, which of course is crucial to determine the zenith angle as a proxy for L.

Figure 3.9 shows the Upgrade's improvement in muon- and tau neutrino rates. Upgrade will aim to test the unitarity of the PMNS matrix, which is rather exciting because non-unitarity always points to new physics, seeing as how it is related to the normalization of probabilities.

Furthermore, the Upgrade will contain various calibration equipment which will measure certain properties of the ice. These measurement can be applied retroactively, and will hopefully enhance all IceCube/DeepCore datasets.

The Upgrade is to be installed during the austral summer of 2022–23, but as of this writing there is no known reconstruction algorithm working for ν_μ, but seeing as the additional information contained in the new DOMs is simply new features for a machine learning algorithm, the work in this thesis is applied to Upgrade MC muon neutrinos.

Chapter 4

Machine learning

Contrary to public expectation these days, machine learning (ML) is neither sexy nor particularly exciting. The misnomer "AI" is often applied to ML, evoking expectations of some general computer intelligence of the science fiction variety.

ML is a varied field, with many different subgenres, of which Deep Learning (DL) is the one that this thesis is concerned with.

However, it might be beneficial to first describe what ML as a field is, before delving into the specifics of DL.

4.1 ML basics

The fundamental schism between traditional programming and ML is this: programs apply logic to inputs, and produce outputs, while ML uses outputs on inputs to produce logic.

Machine learning generally comes in three flavors: supervised learning, unsupervised learning and reinforcement learning. As supervised learning is the tool used in this thesis, it is the only flavor that we shall discuss.

Generally, supervised machine learning entails having a way to map inputs to outputs (e.g. decision trees in tree based ML, artificial neurons in deep learning), access to some labelled dataset and for each data point the ability to calculate an error. This error is encapsulated in a loss functions, which takes as input the model's guess for the data point value and the true value (the "label") and returns a real number, the "loss". The name of the game is then the minimization of this loss, as an error of zero would mean a perfect estimation of the label value.

The input/output data is referred to as "training data", and it is in essence what decides the logic of the model; thus care must be taken with regards to it, as inherent biases in the training data will propagate to the decisions the model makes. This is a continuation of the old programmer's adage, "garbage in, garbage out", because an ML algorithm will only give back what is fed to it; it is not sentient. This is a very active area of discussion at the time of writing, and the summer of 2020 gave a very good example of the issue; see fig. 4.1 on the following page.

Training data is not necessarily easy to come by—and may, for some companies, present an impenetrable moat, if the competition does not have access to the same dataset—but particle physics is special in this regard as Monte Carlo data (section 5.1 on page 45) is typically plentiful; however, it

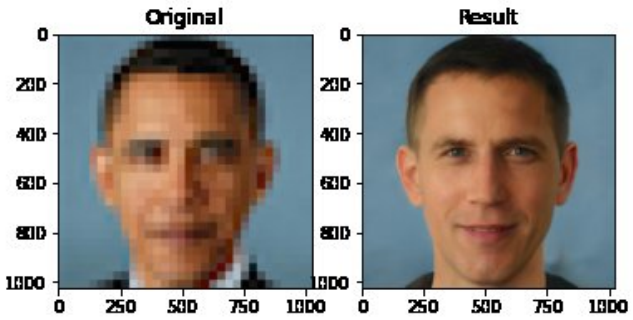


Figure 4.1: The "Face-Depixelizer" [41] takes a blurred image of a person, and generates an image from it. However, the model was trained on primarily white people, and so reconstructs an obviously blurred image of Barack Obama as a white male. Image from [42].

should be noted that Monte Carlo data itself is simulated, based on assumptions the laws of physics, and thus represents an inherently biased version of how the universe works.

While the laws of physics used are typically very well known, the physical system in which they are simulated is almost an approximation. In the case of IceCube, the ice properties, the dust layers, PMT efficiencies and signal response as a function of signal type and angle are not precisely known, leading to differences between simulation and real data.

4.2 Artificial neural networks

Artificial neural networks (ANN) have a much longer story than would probably be thought, seeing how they currently stand as pillars of futurism. ANNs can be said to draw inspiration from nature as the simplest version, a feedforward neural network, uses artificial neurons, modeled on the neurons of a brain, first proposed in the forties [43].

The artificial neuron takes $m + 1$ inputs, consisting of values x_j weighted by w_{km} and applies some "activation function" ϕ to their sum, such that

$$y_k = \phi\left(\sum_{j=0}^{m} w_{kj} x_j\right). \tag{4.1}$$

The activation function is typically non-linear, e.g. a sigmoid

$$S(x) = \frac{e^x}{e^x + 1}, \tag{4.2}$$

or, the popular choice these days, a rectifier

$$f(x) = x^+ = \max(0, x). \tag{4.3}$$

To explain how ANNs work, a simple example using simple linear regression is illuminating.

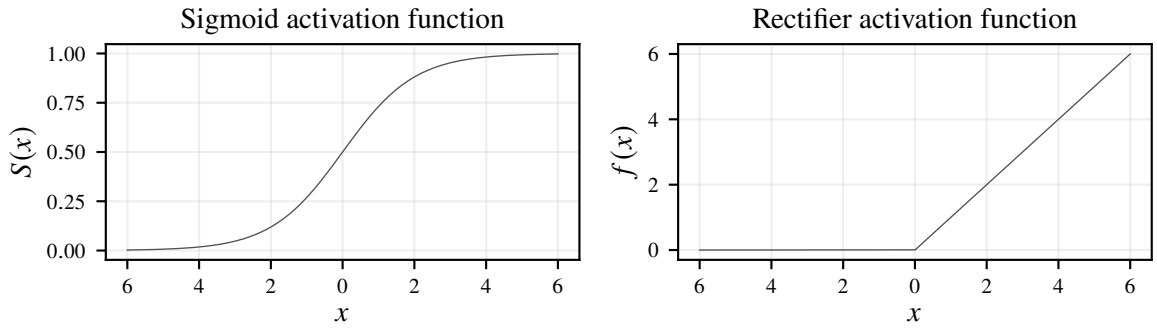


Figure 4.2: Two activation functions, a sigmoid and a rectifier. Both "activate" an input, rather like a Heaviside step function.

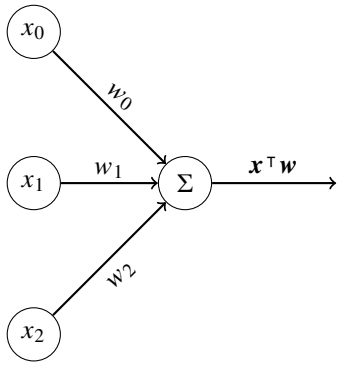


Figure 4.3: The linear perceptron.

In linear regression, a 200 year old technique, a linear relationship between n dependent variables, y_k, and input variables $\boldsymbol{x}_k$, is assumed, such that the output can be approximated as

$$y_k \approx w_0 x_{k0} + w_1 x_{k1} + \cdots + w_m x_{km} = \sum_{i=0}^{m} x_{ki} = \boldsymbol{x}_k^\intercal \boldsymbol{w}, \tag{4.4}$$

where $x_{k0} = 1$ such that w_0 is the bias.

This is a linear perceptron, an artificial neuron with a linear activation function; see fig. 4.3, modeled as a directed acyclic graph. Regression then proceeds by minimizing the error, e.g. the sum of squared residuals

$$\mathcal{L}(\boldsymbol{w}) = \sum_{k=1}^{n} \left(y_k - \boldsymbol{x}_k^\intercal \boldsymbol{w} \right)^2, \tag{4.5}$$

a function of the weights. We are then able to ascertain, by using the derivative of the loss function with respect to the weights,

$$\frac{\partial \mathcal{L}}{\partial w_i} = -2 \sum_{k=1}^{n} \left(y_k - \boldsymbol{x}_k^{\mathsf{T}} \boldsymbol{w} \right) x_{ki}, \tag{4.6}$$

the relationship between the minimum of the loss function and the weight w_i. The loss function, then, needs to be differentiable, in which case the minimum can be reached via the backpropagation algorithm. This algorithm employs the chain rule to calculate the gradient of the loss function with respect to the weights—a layer at a time, iterating backwards—and another algorithm, such as gradient descent, adjusts the weights by moving opposite to the gradient iteratively, such that the weight at iteration $t + 1$ is

$$w_{i,t+1} = w_{i,t} - \alpha \frac{\partial \mathcal{L}}{\partial w_i}, \tag{4.7}$$

with α the learning rate. The learning rate is a hyperparameter, meaning it needs to be adjusted by means outside the neural network, and ensures that the steps taken do not overshoot the minimum; on the other hand, a too small learning rate will result in training taking longer than necessary, as more steps must be taken than needed. The algorithm terminates at some point when the absolute value of the gradient is less than some preset tolerance.

Because neural networks these days tend to involve very large datasets, the application of gradient descent can be a slow process owing to the summing of the gradients of all training examples. This burden can be reduced by applying stochastic gradient descent which calculates an approximation of the true gradient by sampling a subset of the training data at each iteration, done on batches of training samples, incidentally allowing application of parallelization techniques.

The problem of choosing the correct learning rate is solved by optimizers such as Adam [44], employing adaptive learning rates for each weight in the network and calculating the average of the first- and second moments of the gradient at each iteration. These are then controlled using two hyperparameters as decay rates, β_1 and β_2, typically set at $\beta_1 = 0.9$ and $\beta_2 = 0.99$.

The backpropagation algorithm has been implemented in certain frameworks, most noteworthy TensorFlow and PyTorch, as differentiable programming by automatic differentiation, which allows the chain rule to be easily (and quickly!) employed. This is either done by building a static graph of the network, compiled before running the program, and containing derivative information at all nodes[a], or operator overloading which defines derivatives of functions/tensors in the language itself[b]. This is the true enabling technology of the two frameworks, and some languages—such as Swift—even aim to include automatic differentiation as a first-class language feature [45] giving users a type-safe way of running machine learning; it is an exciting time for ML.

While automatic differentiation enables the popular frameworks to run backpropagation efficiently, another key technology has helped neural networks (and ML in general) achieve a breakthrough in to 2010s: graphical processing units (GPUs). These were first introduced to the mass market in the 1990s to accelerate and improve graphics performance of desktop computers, but their parallel nature has meant it is suited well for neural networks that require matrix multiplication, as we have seen. GPUs are

[a] This is how TensorFlow 1.x works

[b] This is how PyTorch and TensorFlow 2.x work by default

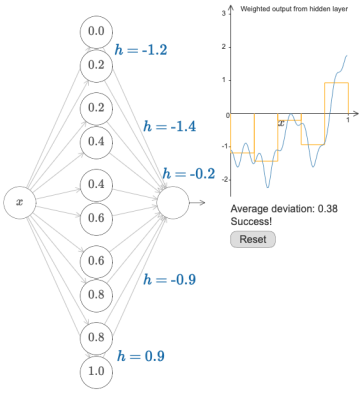


Figure 4.4: Example of a wide neural network approximating the function $f(x) = 0.2 + 0.4x^2 + 0.3x \sin(15x) + 0.05 \cos(50x)$, taken from the excellent [48].

designed such that they are efficient at a small number of specific operations, in contrast to the more generalist nature of CPUs, and tend to have an enormous amount of computing cores relative to CPUs; a new Apple MacBook may have 8 CPU cores, whereas the latest desktop GPU from Nvidia sports 8704. Nvidia also provides an API for doing tensor calculations with its cores, CUDA, which is what PyTorch and TensorFlow interfaces with, such that the common data scientist can operate in familiar languages such as Python, instead of using C++ and CUDA, but it does mean that Nvidia, for the moment, has a near-monopoly on machine learning acceleration.

The toy linear regression example will converge to the best linear fit, but it is only useful for problems that are linearly separable; by its linear nature, it is only capable of linear transformations. This rather limits the application of a perceptron, inasmuch as many real-world problems are non-linear with respect to the input variables, and this observation leads to the introduction of non-linear activation functions into the network. Doing so actually allows a single layer neural network to approximate any continuous function, as shown in [46, 47].

The results refer to "sigmoidal", or "squashing", functions, examples of which are the sigmoid and the rectifier shown in fig. 4.2 on page 37. The result can be graphically understood by inspection of fig. 4.4: the composition of the activation functions by each neuron allows us to approximate the value of any continuous function, provided the network is wide enough; the result holds as well for multivariate functions.

Thus we end up with "multi-layer feedforward networks" which can approximate almost any function, which consist of a vector of inputs multiplied by a matrix (the weights connected to a neuron) with a bias added (all of which is an affine transformation) passed through a non-linear activation function.

The question then becomes, why are the networks of today *deep* instead of *wide*? The answer is that universality has also been proved for deep networks [49], and these types of network exhibit extremely useful properties, explained in section 4.3 on the next page, that we can benefit from.

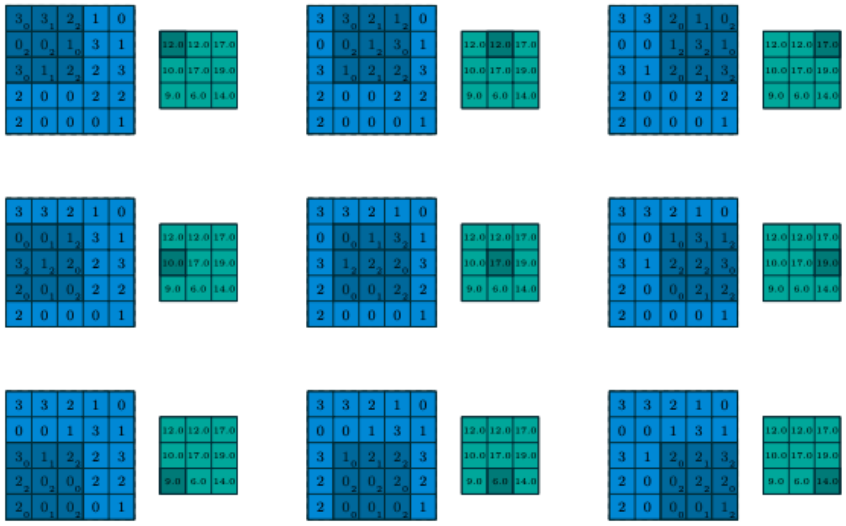


Figure 4.5: An example of discrete convolutions. The dark blue 3×3 kernel slides over the input, and sums the product of the weights and inputs forming a feature map (green matrix) in the process. Image from [50].

4.3 Temporal convolutional neural networks

A problem arises, however, when we want to apply neural networks to images, something that is easily recognized as extremely useful (both for good and objectively bad purposes). Say we flatten a 100×100 image into a vector of length 10,000, and feed it to a multi-layer feedforward network; each neuron in the first layer will now have 10,000 weights that need to be learned! Additionally, the internal structure of the image is disregarded, in that there can be meaning to the height, width and channel axes.

This is remedied by introducing a linear transformation preserving the structure of the input: a convolution. "Convolutional neural networks" (CNNs) create feature maps by employing discrete convolutions via kernels that slide over the input and aggregate information in the receptive field; see fig. 4.5.

CNNs first came to prominence in the realm of image recognition primarily because they impose a sense of locality on the network which meshes well with images: there are probably local features that are important for the output, which could not be taken into account if the structure was lost by applying simple fully connected layers directly to the input; it is also wasteful, as the local information can probably be pooled and share weights. Not incidentally, pooling layers are the second feature of CNNs that made them world famous. These down-sample the output of a convolutional layer by (typically) taking the max value of some subset of values of the image[c], discarding information but serving to reduce parameter size.

Standard CNNs have enjoyed enormous success over the past ten years, and many image applications were built using these. Another type of networks, "recurrent neural networks" (RNNs), though, took the

[c] Pooling layers can also work by taking averages or some other metric.

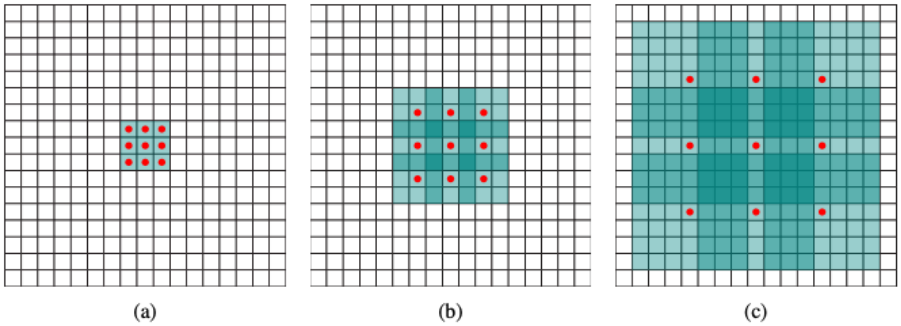


Figure 4.6: (a): F_1 is produced from F_0, and each element has a receptive field of 3×3. (b): F_2 is produced from F_1, and each element has a receptive field of 7×7. (c): F_3 is produced from F_2, and each element has a receptive field of 15×15. Image from [53].

machine translation, speech recognition and time series prediction world with storm. RNNs have storage, internal states that can be used for memory in series where connected events may happen at different intervals.

Recently, however, it has been shown that a new type of CNN, "temporal convolutional networks" (TCNs) [51], perform better in a wide variety of tasks once dominated by RNNs. The million-dollar idea of TCNs are dilated convolutions, used to great effect in [52]. Whereas discrete convolution is defined as

$$F(s) = (\boldsymbol{x} * f)(s) = \sum_{i=0}^{k-1} f(i) \cdot \boldsymbol{x}_{s-i}, \tag{4.8}$$

where $\boldsymbol{x} \in \mathbb{R}^n$ is an input sequence and $f : \{0, \ldots, k-1\} \to \mathbb{R}$ a filter, the dilated convolution operator is defined as

$$F(s) = \left(\boldsymbol{x} *_d f\right)(s) = \sum_{i=0}^{k-1} f(i) \cdot \boldsymbol{x}_{s-d \cdot i}. \tag{4.9}$$

This, then, inserts steps of size d into the filter, and reduces to a regular convolution for $d = 1$. The brilliance of dilated convolutions lies in the fact that it enables the receptive field—the locations in the input array higher level layers are connected to—to grow exponentially in size, while the parameters only grow linearly. This is done by increasing the stride, d, in subsequent layers, as shown in fig. 4.6: in (a) F_1 is produced from F_0, and each element has a receptive field of 3×3. In (b) F_2 is produced from F_1, and each element now has a receptive field of 7×7. Lastly, in (c), F_3 is produced from F_2, and each element has a receptive field of 15×15. However, F_1 through F_3 have the same number of parameters. Thus when we use a larger dilation, an output at the top level can represent a wider range of inputs [51].

TCNs are introduced as a new tool for sequence modelling tasks, and are built on Fully Convolutional Networks [54], distinguished by having all hidden layers be the same size of the input, employing zero padding of kernel size - 1 to ensure the correct output shape. TCNs go a step further by only convolving with inputs from time t and earlier to ensure a sense of causality in the network; there is no leakage from the future into the past. It is simple to tweak the convolutions, such that this causality is violated, and the network may use inputs from all time steps. To aid deep network stability, residual connections [55]

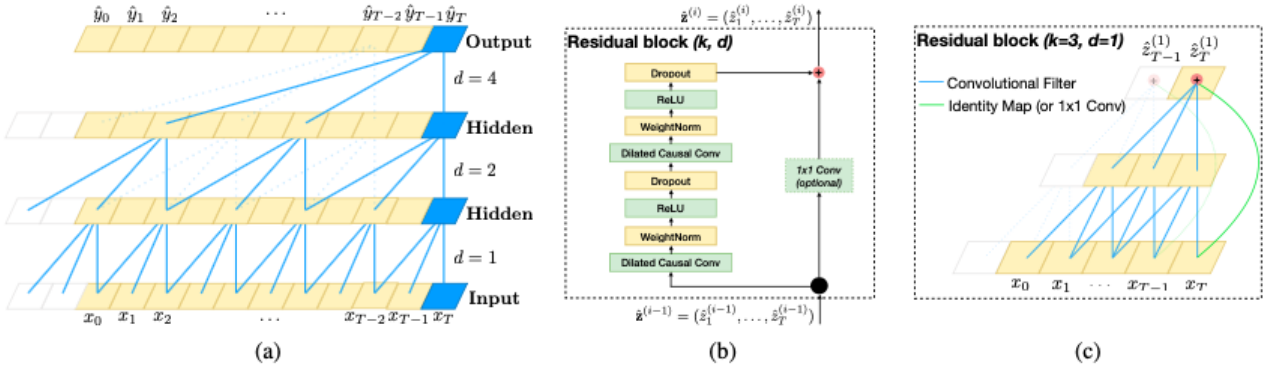


Figure 4.7: (a): a demonstration of dilated convolutions with dilation factor $d \in \{1, 2, 4\}$ and filter size 3. The output is easily able to cover a large receptive field. (b): The TCN residual block, with two dilated convolutions each followed by normalization and activation. (c): The residual connection from input to output. Image from [51].

have been used. These make the output dependent on the addition of the convolution to the identity, i.e.

$$o = \sigma[\boldsymbol{x} + \mathcal{F}(\boldsymbol{x})], \tag{4.10}$$

with σ the activation function, $\boldsymbol{x}$ some input and $\mathcal{F}$ the network transformation; now, the network learns a modification to the identity mapping. It should be noted that the residual connection for a TCN is not directly added to the output, as this would have shape implications. Instead, a 1×1 convolution is applied, to ensure the correct size of the residual. The TCN residual block consists of two dilated convolutions, a weight normalization [56]—which parameterizes the weights in the network to facilitate faster learning—layer (not used in this work; batch normalization, explained shortly, was used instead) followed by a ReLU (Rectified Linear Unit) layer and a dropout layer. The architecture can be seen in fig. 4.7.

Batch normalization [57] is applied in order to reduce dependence on the careful selection of correct learning rate, because in a deep network the input to any layer is dependent on previous layers, and so changes to any parameter balloons through the network. Thus some sense of normalization that preserves the important structure but suppresses the scale is beneficial.

Batch normalization relies on the gradient of batches, briefly touched on earlier in the discussion on stochastic gradient descent. One calculates the approximate gradient,

$$\frac{1}{m}\frac{\partial}{\partial\Theta}\mathcal{L}(\boldsymbol{x}_i, \Theta), \tag{4.11}$$

where m is the size of the batch, $\boldsymbol{x}_i$ is the i'th training sample and Θ the network parameters that will minimize the loss function $\mathcal{L}$. To normalize the output of a layer we calculate the mean and variance of the values of the mini-batch and normalize to those. Furthermore, the values are shifted and scaled by the learnable parameters γ and β.

The values are passed through a ReLU, shown in fig. 4.2 on page 37. Lastly, a dropout layer randomly—with probability p, a hyperparameter—drop a node at training time for each mini-batch, in effect training a different network each time; when the time comes for testing, all nodes are used, but

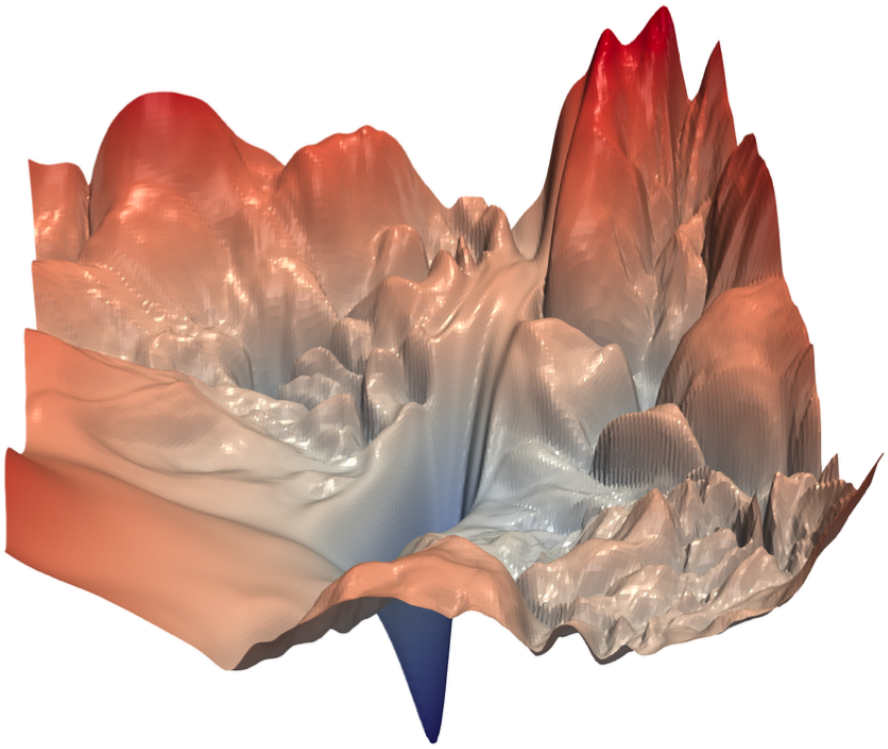


Figure 4.8: A complicated neural net loss landscape. Image from [59].

they are now reduced by a factor p. This forces nodes to probabilistically take on more or less responsibility for the inputs [58], making the network more robust and less prone to overfitting.

In this thesis a TCN is employed, and a regular 1D convolutional network as well. In a 1D CNN the kernel slides in only one direction, but is otherwise similar to image convolution networks. The 1D CNN version was the starting point, before TCNs were discovered, and does have prediction power, albeit using more parameters (i.e. a larger network) than TCNs.

4.4 Loss functions

The choice of loss function influences the performance of the network.

Of course, the function needs to be differentiable, or backpropagation will fail. Some loss functions are typically used for classification purposes (where e.g. a label 0/1 is the target), such as cross-entropy, while others—such as MSE, MAE, Huber and logcosh—are popular regression losses, and these are highlighted in vrefsec:algorithm.

Some care must be taken in implementing the loss function. As an example, one may build a network that produces several outputs, of varying scale, which can be a big problem for the loss function. Say that on output variable is in general much larger than another, then that one will completely dominate the loss values, and the smaller variable will be drowned out. This can be remedied in several ways: firstly, one might scale the input variable to be roughly the same size, as detailed in section 4.5 on the following page; this ensures that no variable dominates the loss function. Secondly, it is possible to

assign weights to the different terms of a loss function, and thus solving any potential scale issues. Lastly, it is possible (as was chosen in this thesis) to simply focus on one variable as output, completely eliminating the issue, as long as this variable is on a sensible scale. The reason for this choice was more convenience than anything else, as it presented a nice way of segmenting and focusing the output prediction types and analysis.

In addition, it is possible to create an ensemble network or meta-learner, which takes as input reconstructions from other networks/loss functions, and learns what loss function best suits which purpose. This has been tried with great success in other works [60], and is surely an avenue to investigate further.

4.5 Pre-processing

Pre-processing is ever important in the data science space. Often datasets will contain missing, garbled or even complete nonsense values, and these must be pruned. This step can be said for, the IceCube problem, to be handled by the processing levels described in section 3.4 on page 29 which cleans noise and events that do not look like data, but the pre-processing work does not stop there in neural network settings. Because backpropagation relies on gradients, it is necessary these do not explode, or else training can proceed no further. To manage the scale of the inputs, then, the trick employed by many is to simple re-scale the distributions. This can of course be done by removing the mean and scale according to the variance, such that

$$\text{transformed}(x_i) = \frac{x_i - \mu}{\sigma}, \tag{4.12}$$

with μ the mean, σ the standard deviation and x_i the input feature of the i'th sample. However, this method is sensitive to outliers in the data, which will skew both the mean and the standard deviation, so instead robust scaling can be employed which uses the median and the interquartile range (IQR) instead. The IQR is the difference between the third and first quartiles of a dataset, that is the middle number between the minimum and the median, and the middle value between the median and the maximum respectively. The robust scaler is then defined as

$$\text{transformed}(x_i) = \frac{x_i - \text{median}}{\text{IQR}}, \tag{4.13}$$

and is implemented as such in e.g. `scikit-learn` [61]. If the dataset is skewed it may be beneficial to first log transform it, before applying scaling, but there are also other non-linear methods, such as a quantile transformation, which transforms the data into a new distribution (e.g. Gaussian or uniform), robust to outliers. However, as this is a non-linear transformation, it distorts any linear relationships between variables.

It should be noted that the energy and charge truth variables (see chapter 5 on the facing page) are transformed to $\log_{10}$.

Chapter 5

Data

5.1 Monte Carlo simulation

Because the nature of quantum mechanics is probabilistic, simple, analytical calculations rarely suffice for theoretical predictions in high energy physics (HEP). In order to develop a reconstruction algorithm, one needs to be able to test the algorithm on labelled data; should the algorithm be of the machine learning variety, this data is even integral to the development of the algorithm, as we have seen.

The HEP community relies on the Monte Carlo (MC) method for this. The canonical introduction to MC is that of the calculation—or, more correctly, the approximation—of π: draw a circle inside a square, uniformly scatter points on the square, take the ratio of points inside the circle to total number of points scattered (and multiply by four).

The requirement of uniformity is important, as else the distribution will be skewed and the calculation wrong. As seen in fig. 5.1 the number of points also plays a role; the more points, the better the approximation, which seems intuitive enough. This translates to particle physics rather well. Generally, observables in particle physics take the form of [62]

$$O = \int \mathrm{d}\Phi_n(p_a + p_b, p_1, \ldots, p_n)\, \frac{|\mathcal{M}|^2}{8K(s)} \Theta(O, p_1, \ldots, p_n), \tag{5.1}$$

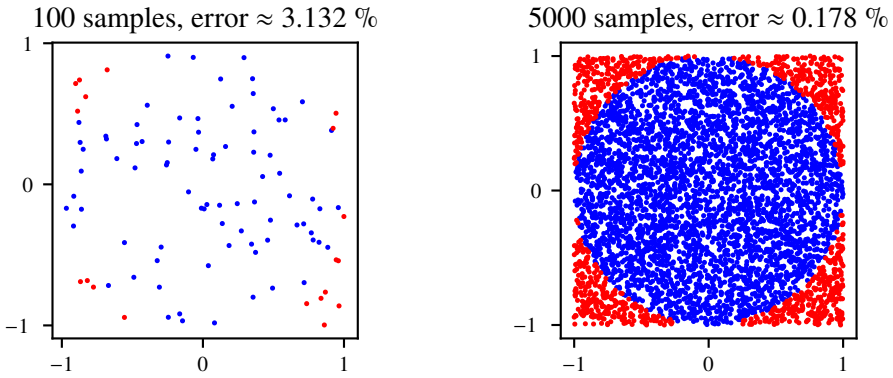


Figure 5.1: Two runs of the π MC algorithm with 100 samples on the left, 5000 samples on the right. The points outside the circle are colored red, while the points inside are colored blue. The shape of the circle is easily seen on the right, where the error in calculating π is around 0.025 %.

with n outgoing particles from a collision between a and b, where $\mathrm{d}\Phi_n$ is the Lorentz invariant phase space, $1/8K(s)$ is a kinematical factor, $\mathcal{M}$ is the familiar matrix element and Θ is a function defining the observable and experimental cuts. The matrix element can be calculated, as discussed in section 2.1 on page 11[a], and the phase space is then up to MC methods to solve.

The nuts and bolts are not important for this short discussion, but the long and short of it is that the final state momenta from the collision can be generated from re-expressing the phase space and inserting uniformly randomly generated numbers; rather like the approximation of π.

While this handles the event generation part of the equation, simulation is needed afterwards to simulate properties of materials etc.

5.2 Distributions and selections

The source of the data used is I3 files, as detailed in section 3.5 on page 31, and moved to SQLite (section 5.3 on page 50) for training. It is structured in features and truth, where the features represent detector data (simulated for MC) and the truth is the truth variables from the MC generator.

The detector data contains 13 distinct features (and two cleaning states), extracted from the I3 files, while the truth contains 12. It should be noted that in [60] feature engineering on the same dataset was attempted, resulting in no gains, and so is not re-applied here.

Some features—the string number, DOM number, PMT number and PMT type—are not used for training, but have been useful for analysis purposes, while the directional PMT features are only useful for Upgrade purposes (in that for DeepCore, the values are always the same ($[0, 0, 1]$)) because all PMTs point downwards.

In the end, the problem is one of converting 6 or 9 features into an output.

In the following, the ISO convention of spherical coordinate systems is used such that θ refers to the zenith angle, and ϕ the azimuthal angle.

5.2.1 oscNext

The DeepCore dataset used contains CC interaction muon neutrinos, capped at 1000 GeV due to lack of data above that point; in addition, the focus is on low energy events, further providing reason to cut there. The azimuthal distribution is uniform, and the neutrinos are mostly up-going. See fig. 5.2 on page 48.

5.2.2 Upgrade

The Upgrade dataset was capped as well at 1000 GeV, and a cut was made on at least 4 activated DOMs in the Upgrade fiducial volume. However, the dataset still contains noise, and it is clear that the energy

[a] There are complications as there are higher-order Feynman diagrams; some can be calculated perturbatively, others cannot and must be solved in different ways.

Table 5.1: Features used in the dataset with datatype. Each entry represents an activated PMT, having recorded a light pulse.

Name	Description	Type
`string`	String number of activated PMT	`int`
`dom`	DOM number of activated PMT	`int`
`pmt`	PMT number of activated PMT	`int`
`dom_x`	x location of activated PMT (IceCube coord. system)	`float`
`dom_y`	y location of activated PMT (IceCube coord. system)	`float`
`dom_z`	z location of activated PMT (IceCube coord. system)	`float`
`time`	Relative time coordinate of hit	`int`
`charge_log10`	The charge recorded in the pulse in $\log_{10}$	`float`
`pmt_x`	x component of the PMT pointing unit vector	`float`
`pmt_y`	y component of the PMT pointing unit vector	`float`
`pmt_z`	z component of the PMT pointing unit vector	`float`
`pmt_type`	Type of PMT (IceCube, mDOM, D-Egg)	`int`
`pulse_width`	Pulse width, DAC-dependent	`int`
`SplitInIcePulses`	Pulse included in `SplitInIcePulses` cleaning	`bool`
`SRTInIcePulses`	Pulse included in `SRTInIcePulses` cleaning	`bool`

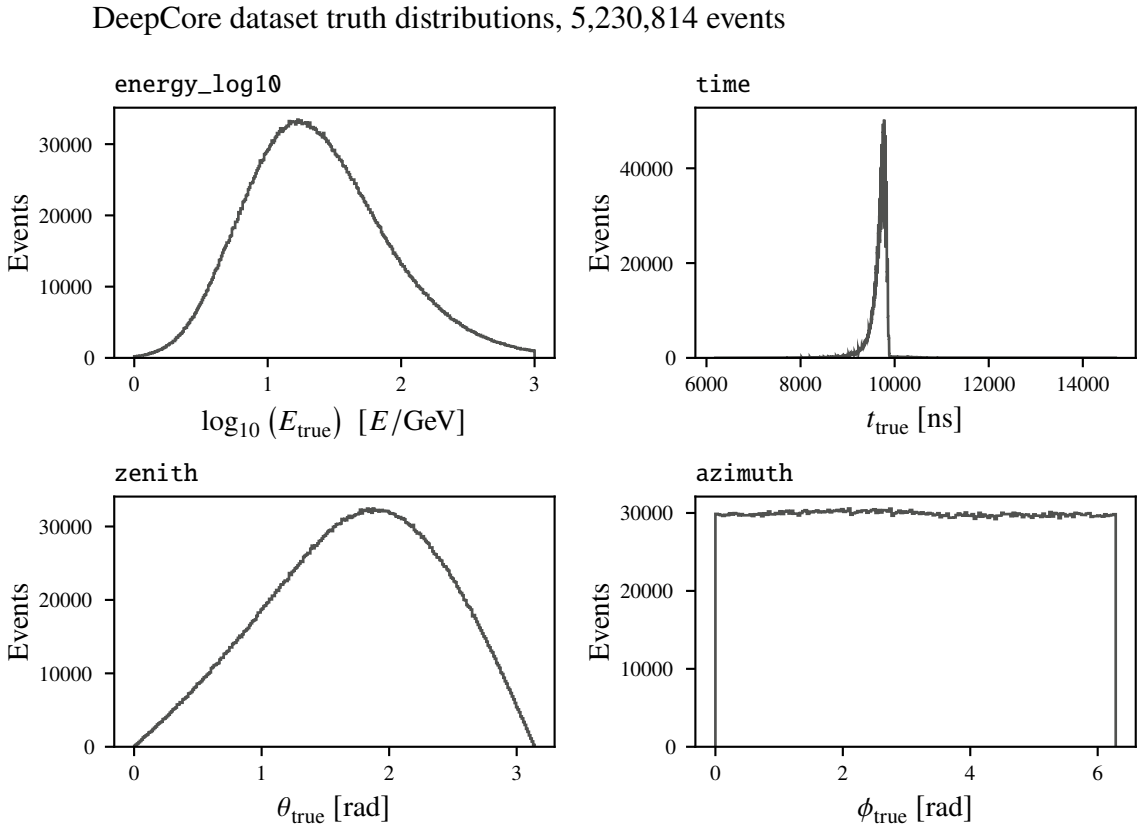


Figure 5.2: Truth variable distributions for the DeepCore dataset used. The distributions match data well, seeing as they have passed L5 (vrefsec:l5), and it is obvious that the neutrinos are mostly up-going from the zenith distribution in the bottom left figure. The energy range has been cut at 1000 GeV, simply because there is a dearth of data there and it is not particular interesting to oscillation analysis. Naturally the azimuth distribution is uniform, as there is no preferred direction or asymmetry there. The time distribution is heavily peaked around 10,000 ns by definition; when an event is triggered, it is set at this time, and 10,000 ns before and after is read out.

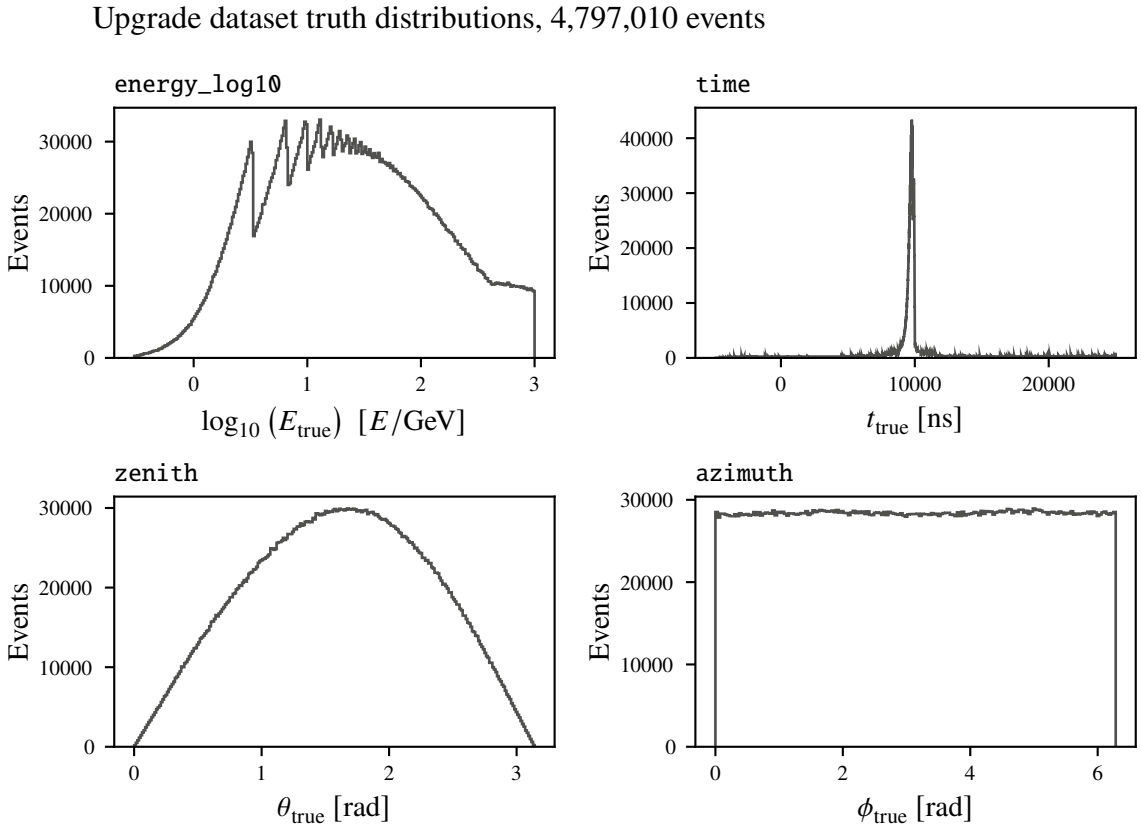


Figure 5.3: Truth variable distributions for the Upgrade dataset used. The azimuthal distribution is similar to the DeepCore set, but the similarities end there. First of all, the energy distribution displays ridges, artifacts of stitching together of different MC runs; further, it ends abruptly at 1000 GeV. The zenith distribution is more skewed towards down-going neutrinos, while the time distribution shows noise with large outliers. This is caused by the Upgrade dataset not yet having the same post-processing pipeline as DeepCore/oscNext, meaning cuts have not yet been applied. This makes it rather hard to get a good working neural network algorithm on the dataset.

Table 5.2: Truth labels used in the dataset with datatype. Each entry represents an MC generated particle. While the directional and spherical angle are in some ways redundant, both are kept as it makes it easier to run algorithms reconstruction directional unit vectors or spherical angles.

Name	Description	Type
`energy_log10`	Energy of particle in $\log_{10}$	`float`
`time`	Relative interaction time of the particle	`float`
`position_x`	x-component of interaction position	`float`
`position_y`	y-component of interaction position	`float`
`position_z`	z-component of interaction position	`float`
`direction_x`	x-component of particle pointing vector (going to)	`float`
`direction_y`	y-component of particle pointing vector (going to)	`float`
`direction_z`	z-component of particle pointing vector (going to)	`float`
`azimuth`	Azimuthal direction of particle (coming from)	`float`
`zenith`	Polar direction of particle (coming from)	`float`
`pid`	Particle ID	`int`
`interaction_type`	Current type (neutral/charged) of interaction	`int`

distribution has issues, mainly due to stitching together MC runs of different energy levels. Much works needs to be done on this dataset, in a pipeline similar to the post-processing of oscNext (section 3.4 on page 29), before great results can be obtained. The distributions can be seen in fig. 5.3 on the previous page.

5.3 SQLite

As laid out in section 3.5 on page 31, the IceCube native file format is not appropriate for machine learning applications. Thus another format had to be used for the occasion. First HDF5 was used. HDF5 is a hierarchical data format, containing "Datasets" and "Groups". Datasets are arrays, similar to those of NumPy, containing data of a single type and a rectangular shape, and Groups are containers whereby the file is organized. The format seems to be very popular in the scientific setting, but lacks a truly native Python library (i.e. a library that does not need to be externally installed), and does not seem to have a simple query language attached to it. Furthermore, performance issues were encountered when more than 6 features were to be extracted at a time, and so alternatives were explored rather early in the process.

The choice soon felt on `pickle` files, with the inspiration coming from typical ML image applications. Here, it is typical to organize image files in folders: in a classification task, where an algorithm must learn to classify cats and dogs, the images may be organized as follows:

```
training
└── cat
```

```
event = {
    "features": {
        "dom_x": [-2.09, 0.27, 2.39, -0.03],
        "dom_y": [-2.20, 0.81, 1.08, -1.85],
        "dom_z": [-0.21, 0.48, 0.80, 1.19],
        "time": [-1.40, -1.39, -1.23, -1.10],
        "charge_log10": [-0.62, -0.18, -0.50, -0.09],
    },
    "truth": {
        "energy_log10": -1.00,
        "time": -0.80,
        "azimuth": 1,
        "zenith": 1,
        "pid": -14,
    },
}
```

Listing 1: An example of an event in Python dictionary form. The data has been transformed by a robust scaler, as detailed in section 4.5 on page 44.

```
    dog
test
    cat
    dog
```

This was extended to the problem at hand, with each event being saved as Python dictionaries in `pickle` files, named by event number, such that one event might look as in listing 1. `pickle` is Pythons format for serializing objects, meaning one can save any Python object to disk with it. This proved to be a very efficient solution in training, as opening pickle files is an embarrassingly parallel problem, and it can be done very quickly even in Python.

However, the structure presented a problem: dealing with millions of events means dealing with millions of files, and performance degraded significantly on other tasks. As an example, the datasets were created on an HPC cluster with no access to GPUs. Therefore, the sets had to be transferred to an external machine containing GPUs, and this was not easily done using standard tools such as `rsync`. `tar` was employed, but the act of taring, transferring, and untaring was tedious, and would become a problem if new datasets were to be created as it hindered fast application of new ideas.

The dataset was then converted into `shelve`, a Python native persistent database which can hold as values arbitrary Python objects, such as dictionaries. Each key was then an event number, and its value event information; as in listing 1. Performance was unfortunately found to be so bad as to be unusable, and so finally SQLite was discussed.

SQL (Structured Query Language) is a language for querying relational databases, invented in 1974. Its use is very common in the private sector, and SQLite specifically claims to be the most prevalent database in use at the moment[b]. The reason for this seems to be the fact that most—if not all—languages have baked-in support, and whereas most other relational databases work in a

[b] `https://www.sqlite.org/mostdeployed.html`

client-server fashion, SQLite is file-oriented. This means it is easy, if not downright trivial, to package a database in e.g. a mobile application.

It does not seem to this author that SQLite has gained any traction in the ML community. Indeed, most example regression tasks found on the internet contains data in CSV files, or maybe in the NPY format, but as datasets get bigger, both of these are bad choices. Besides the fact that CSV is a text file, and so suffers from the problems laid out in section 3.5 on page 31, both formats need to be loaded completely into memory on reading, and this can quickly run into memory capacity issues. SQLite, on the other hand, is opened with a connection, and data is not fetched until a query is actually run. This allows the dataset contained in the SQLite file to be of arbitrary size (limited by file system and other externalities) with no memory impact; you simply fetch the training events you want at runtime. This is aided by the fact that SQLite uses B-trees to index its rows: a relational database, SQLite included, organizes the data in two-dimensional tables, and optimizes the search time by employing unique indices. In lieu of an index, a search in a table would run in linear time, because the program would have to search through each row until the wanted index is found. Binary search algorithms changes this to logarithmic time, $O(\log n)$ (with n the number of entries), by sorting the index and checking the middle element; if this element is not equal to the wanted index, the half where the index cannot be found is discarded, and the algorithm run again. B-trees is a general structure that can implement binary search efficiently, and thus allows SQLite to fetch indices in logarithmic time.

However, the IceCube data does not immediately lend itself to this structure. As can be seen from listing 1 on the preceding page the features are of shape $M \times N$ where M is the length of the event and N the number of features, while the truth labels are of size $1 \times P$ with P the number of labels. The natural database would accommodate this, and there exists solutions that, called NoSQL, which hold data in a JSON-like format. However, no widespread format was found that is file-oriented instead of client-server, a requirement for portability; thus SQLite was settled on. Each row in the truth table has a unique event number as its primary key, while each event (spread over several rows) in the features table has the same event number.

The structure can be seen in SQL diagram form in fig. 5.4 on the next page.

More intangible benefits derive from using SQLite. First of all, SQLite supports SQL. This means that one is able to search easily in databases using clauses to narrow the search, and this aids tremendously in analyzing the data and making cuts, for example. For example,

```
select event_no from truth where energy_log10 between 1.0 and 2.0
```

would give all event numbers where $\log_{10}\left(E_{\text{truth}}\right)$ is between 1 and 2. Another thing, which is not immediately obvious, is the fact that SQL databases enjoy extremely wide app support. This means that there's a large selection of apps (free and otherwise) that allow querying and visualization of the database, an extremely nice feature.

Because IceCube has an abundance of data and MC files, from different analysis groups, and different generators, it was infeasible to gather all data in one SQLite dataset. Therefore, a solution was implemented by the author, wherein one database contains meta info from all available IceCube files. This has many advantages. Firstly, it provides one central repository for everyone doing machine learning on IceCube. This means there is one source of truth for event numbers, meaning everyone can agree on what data is used for training, and what is to be blinded and used for test. Secondly, it facilitates quick iteration, by allowing the user to create different sets quickly with different cuts, without the size of the database being to large to be portable.

Figure 5.4: The structure of a dataset, with the features table holding (MC) detector data from events, and the truth table holding the (MC) particle truth variables.

Code for this, called "CubeDB", can be found on Github at `https://github.com/ehrhorn/cubedb`, created by the author. This code relies on the I3 files being available, but should otherwise be able to run on all machines that have Docker or Singularity installed, facilitated by a compiled Docker image that includes all IceCube software necessary to fetch data from I3 files. CubeDB takes a SQL query as an argument, and uses this to select events from I3 files and builds a SQLite dataset from it. It also contains scripts for building a meta database, and thus is ready to be implemented should a group wish to do so[c], and the software is already in use at the Niels Bohr Institute by at least one other person pursuing IceCube machine learning.

It is not hard to imagine a "professional" workflow for the implementation of this solution; the meta database could run on a true SQL server, accessible via (protected) API calls, while the dataset creation tool CubeDB could run on a generic web server with access to all I3 files contained in the meta database. A GUI for CubeDB can be built using web technology, which would allow IceCube ML researchers to easily "order" the creation of a new dataset to be delivered to some shared disk location. In any case, much of the development of this thesis centered on CubeDB, because there is no default solution in IceCube for this, meaning *any* solution would be of immense benefit to new students starting an ML project; a central source of truth on event numbers and training samples/test samples would also be very valuable, and both can be provided by CubeDB.

[c]The solution does need documenting and tests, however

Chapter 6

Design

6.1 Defining the problem

With the given dataset, it is possible to train a neural network to predict the following properties of an incoming neutrino: energy, interaction vertex, interaction time and direction. As the oscillation studies—as described in section 2.4 on page 16—depend on the energy and the polar angle (as a proxy for the length traveled through the Earth), these two variables are the ones examined in this work.

The algorithm performance will be evaluated based on $\log_{10}(E)$ and θ resolution in 18 energy bins in the range 1–1000 GeV of simulated charged current muon neutrinos "detected" in DeepCore, 6 bins per order of magnitude. To gauge the performance the algorithm will be tested against the current best DeepCore reconstruction algorithm, Retro Reco (section 3.4.5 on page 30).

In the following sections first the custom made tooling used to solve the problem is described, whereafter hyperparameter- and model experiments are performed.

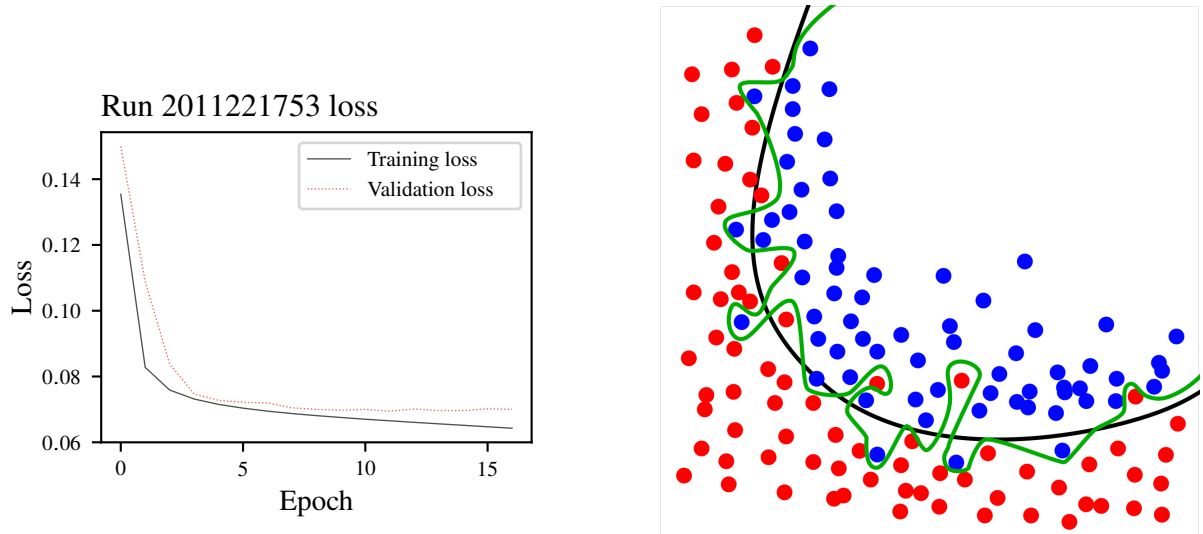
The full DeepCore, charged current muon neutrino dataset (section 5.2 on page 46) where Retro Reco reconstruction is available contains 7,280,939 events in total, where 2,050,125 are blinded, set aside as a test set, leaving 5,230,814 for training. Of these 1,046,163 are used for validation, which means 4,184,651 are trainable. To save time, most experiments were run on a set containing around 400,000 events, and all experiments employ early stopping.

Early stopping runs an inference pass on the validation step at the end of each epoch, and records the loss function value. When the loss is seen to improve on the training set, but not the validation set, it is a sign of overfitting to the training data, because the network is no longer able to infer well on unseen data.

6.2 Tooling

Having defined the problem, solving it requires finding an appropriate architecture. Before that is possible, one needs the pipeline up and running, because training cannot proceed without the data structures defined and built, and evaluation cannot proceed without calculating the metrics and comparing one run to another.

While CubeDB was built for the creation of datasets, CubeFlow (`https://gitlab.com/ehrhorn/cubeflow`) served as the code for training and metric calculation. Early on in the process, it was discovered that IceCube's event viewer, Steamshovel, while extremely



(a) (b)

Figure 6.1: (a) shows an actual example of the training and validation loss values during an experiment. The two diverge before the 10th epoch with the validation loss staying somewhat flat, and the training loss decreasing. This indicates that the network has begun overfitting to the training data, an example of which is shown by the green line in (b) (from [63]). This will lead to a decreased inference capacity of the network because it is unable to generalize to unseen data.

powerful was difficult (and in fact, impossible on the author's available machines) to compile and run, it was decided to build an event viewer by using modern frameworks which supported data in the SQLite format from CubeDB. The software was named Powershovel, a play on IceCube's native Steamshovel event viewer[a], and the source code is found at `https://gitlab.com/ehrhorn/powershovel`. It was soon realized that more visualization tasks were needed, in addition to event viewing, and Powershovel was thus amended to contain two more sections, Distributions and Runs.

Distributions shows dataset variable histograms, calculated when a new set is created. This is important for visual inspection of the variable distributions, to ensure nothing went wrong in the creation, to test if training and validation samples are similar, and to confirm that distributions are not changed by the transformations described in section 4.5 on page 44; a screenshot of the distribution screen, showing the variable `energy_log10`, can be seen in fig. 6.2 on the facing page.

The histograms are calculated and saved in `pickle` files during dataset creation such that the tool is actually responsive and useful, and can load a new dataset—and a new variable—in milliseconds. Powershovel is made using Python, Streamlit and Plotly to ensure wide support, modern web technologies and ease of use.

Figure 6.3 on the next page shows the event viewer page where a user may inspect events from different SQLite datasets. This uses a subset of events from the true dataset database, in order to load quickly, an issue because all events need to be loaded up front, such that the user may filter on e.g. true energy of the event. The event view is static, with the time dimension instead indicated by color, but the 3D plot is interactive such that the user may pan and zoom to view the event from different angles. This page was also adapted so that it is possible to upload I3 files for viewing; it was spurred on by the IceCube group

[a] A steam shovel is a power shovel; steam shovels were replaced by diesel-powered shovels in the 1930s

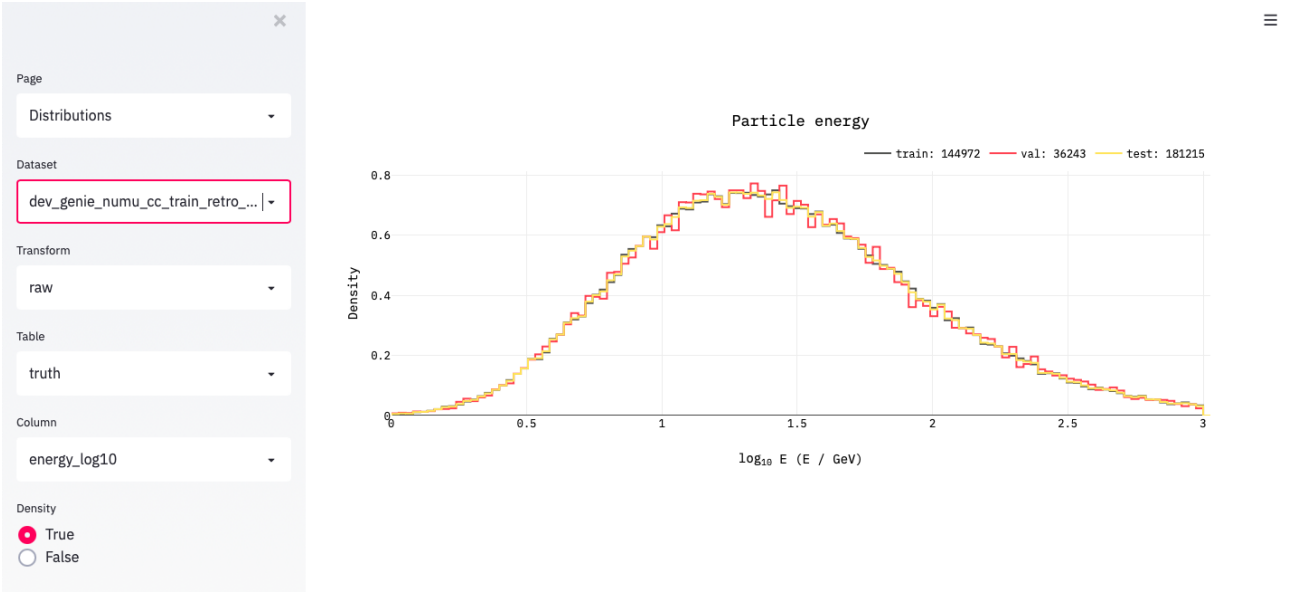


Figure 6.2: The true energy distribution of a DeepCore dataset shown in Powershovel. Note the ability to chose transformation (raw or scaled), table (features or truth) and variable.

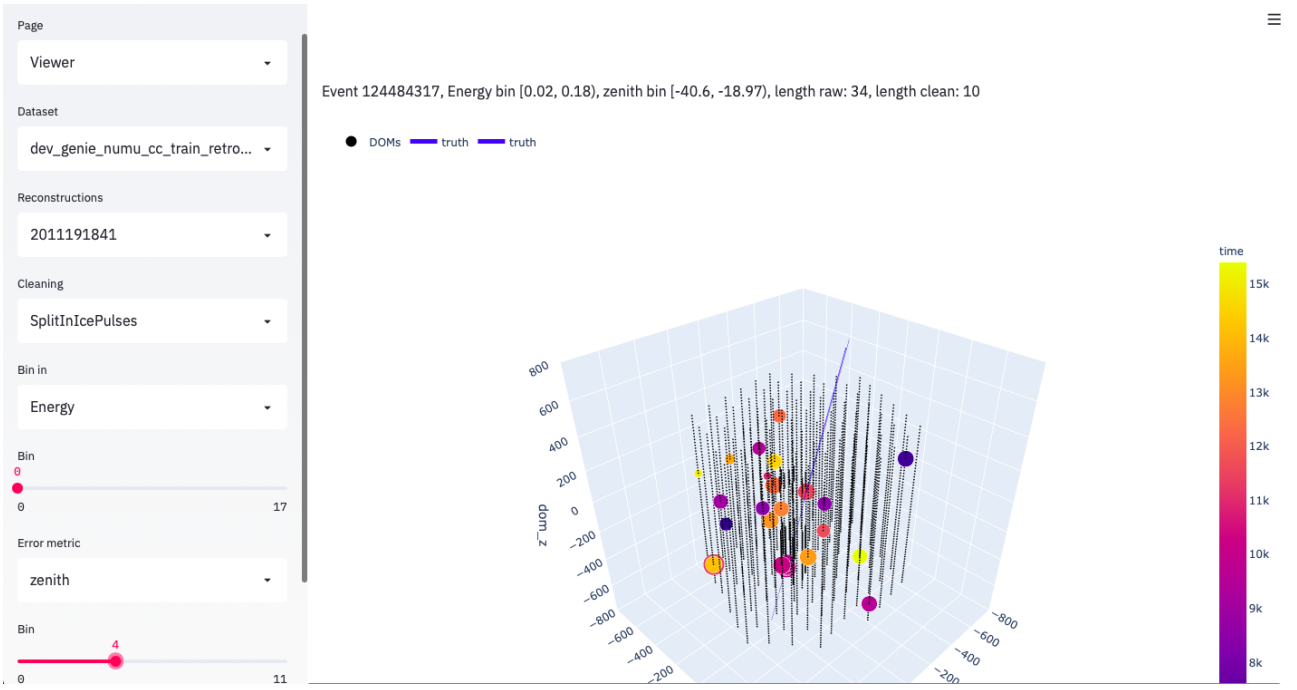


Figure 6.3: The Powershovel event viewer page. Each light pulse is indicated by the colored sphere, superimposed on the black DOMs. The size of the pulse is proportional to its charge, while the color is decided by the relative time offset of the pulse. The user can see relevant reconstructions, such as direction or vertex position, if any are available for the dataset, while the events are binned in energy. The user may also filter on reconstruction metrics; as shown in the screenshot, this view has been filtered on showing events where the absolute value of the zenith reconstruction error is 18.97–40.6°.

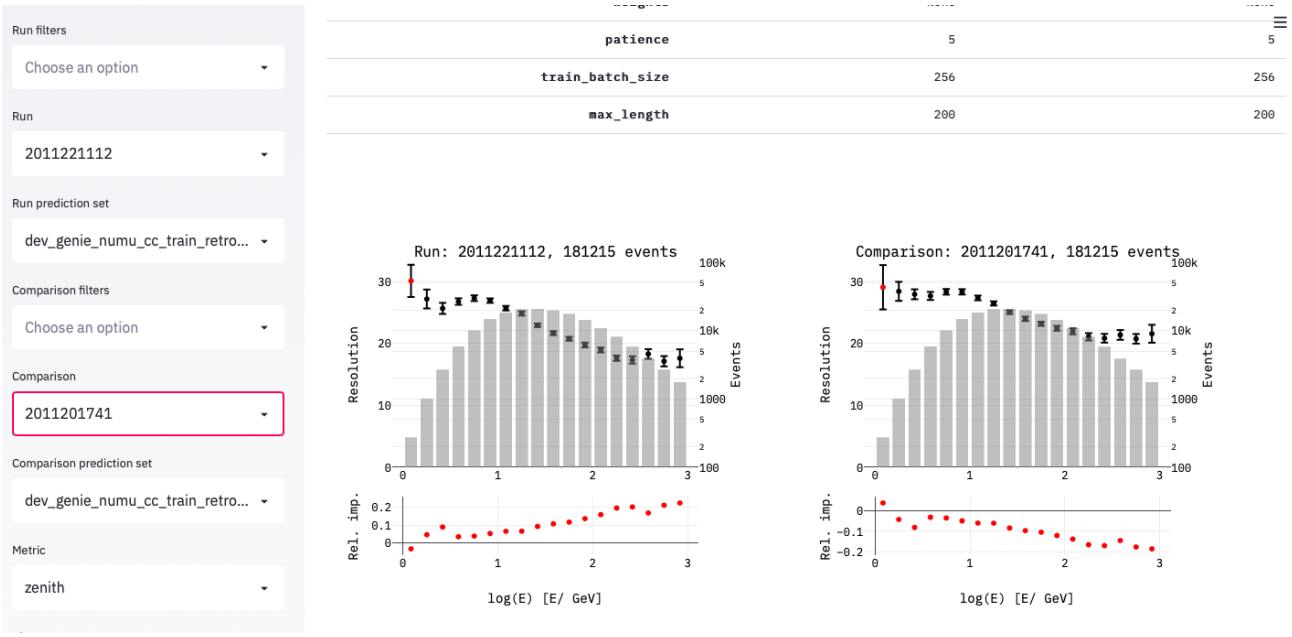


Figure 6.4: The Runs page showing two different runs, side by side. This shows the zenith resolution, as detailed in section 6.3 on page 60. Notice the "Rel. imp." in the bottom of each figure, showing the relative change in performance between the two runs.

at The Niels Bohr Institute having trouble with Steamshovel, and so Powershovel was amended and sent to them for evaluation.

The Runs page shows the result of a training run. After training, CubeFlow runs reconstructions on a given dataset, saves them to a SQLite database, calculates errors and metrics (see section 6.3 on page 60), makes histograms, and stores them in `pickle` files for fast retrieval. The runs have a common naming convention, which includes certain key indicators of the algorithm (e.g. maximum event length, loss function, etc.) which is then used for filtering purposes such that the user can quickly see all algorithms using for example an MSE loss function. Multiple filters can be applied simultaneously.

The page has a two-column layout, which accommodates side-by-side comparison of two runs. This is crucial, as there is no single figure of merit[b] summarizing the performance in one number. Most figures have a subfigure below it, showing the data basis for calculating that figure; and because figures are binned in energy, a slider allows the user to choose what energy bin this subfigure should show data from. This can be seen in fig. 6.5 on the facing page.

The creation of this tool and CubeDB represents a large part of the man-hours of this project, but the tool was used extensively during development of the machine learning algorithm, detailed in section 6.4 on page 60. Although not written by a computer science graduate, nor a professional programmer, the code is shared freely in the hope that any idea is taken up and implemented by IceCube.

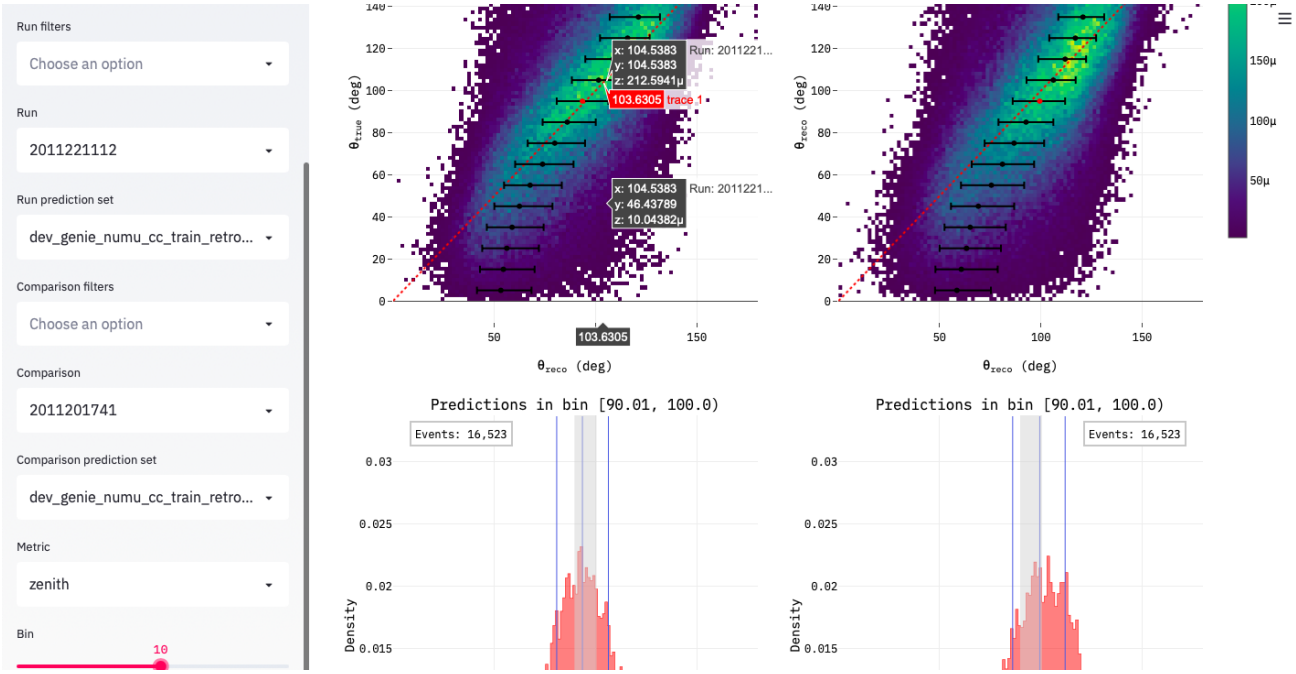


Figure 6.5: The Runs screen, further down the page than shown in fig. 6.4 on the facing page. Two 2D histogram plots are shown, true zenith vs. the reconstructed zenith. The energy bin selection is shown in the bottom left corner, and below the 2D histograms 1D histograms showing the distribution of zenith predictions in the selected energy bin can be seen. Furthermore, to highlight the interactive nature of Plotly (the framework drawing all plots in Powershovel) extra info is visible the left 2D histogram, a consequence of the cursor hovering over it. This allows the user to see values for every point in the histogram. Zooming and panning is also possible.

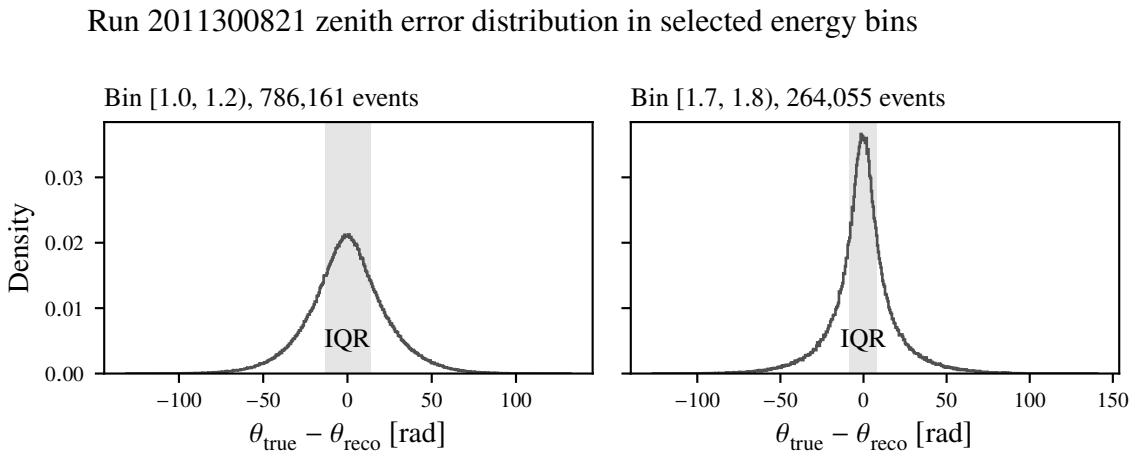


Figure 6.6: Error resolution in energy bins. Top row shows the error distribution in energy bins [1.0, 1.2) (left) and [1.7, 1.8)

6.3 Metrics

To define the performance of an algorithm the resolution is chosen. The resolution is here defined as the normalized interquartile range of the error distribution in a certain energy bin, i.e.

$$w \approx \frac{\text{IQR}}{1.349}, \tag{6.1}$$

an example of which is shown in the top row of fig. 6.6 on the preceding page. The normalization is chosen because for a normal distribution

$$\text{IQR} \approx 1.349\sigma, \tag{6.2}$$

meaning that for a normal distribution w would be σ.

The error distribution is different for different metrics, but represents how well the algorithm reconstructs a neutrino; it is zero for perfect reconstruction.

To give some sort of error estimate on w bootstrapping is employed with the BCa method [64] to give a 90 % confidence interval. The IQR reported by energy bin and its confidence interval can be seen in e.g. fig. 6.5 on the previous page.

6.4 Algorithm

The best algorithm is chosen by essentially running experiments of different neural network designs. Early experiments used a simple 1D CNN, which slides a kernel across the data in one direction (hence the name), followed by fully connected layers. Later experiments employed TCNs, as these were seen to much improve the performance with a similar amount of parameters and similar performance with much fewer parameters.

Because the inner workings of a neural network is somewhat opaque, it can be difficult to reason about the design with supreme confidence, and so for new tasks[c] experimenting is the only way to reason about the most effective design.

6.4.1 Hyperparameters

There is also the case of hyperparameters. Not only must one choose an appropriate loss function, general values such as the learning rate need also be tuned, and even the shape of the scheduler that controls the change of learning rate over time. Furthermore problem specific parameters may arise, such as the maximum length of an event in the case of IceCube reconstruction, necessary because

[b] Resolution is used for evaluating different algorithms, but it is not a scalar. See section 6.3

[c] Staples of machine learning, such as the MNIST task, have years of experimentation behind it; in that sense it is an old task, contrary to the one in this thesis

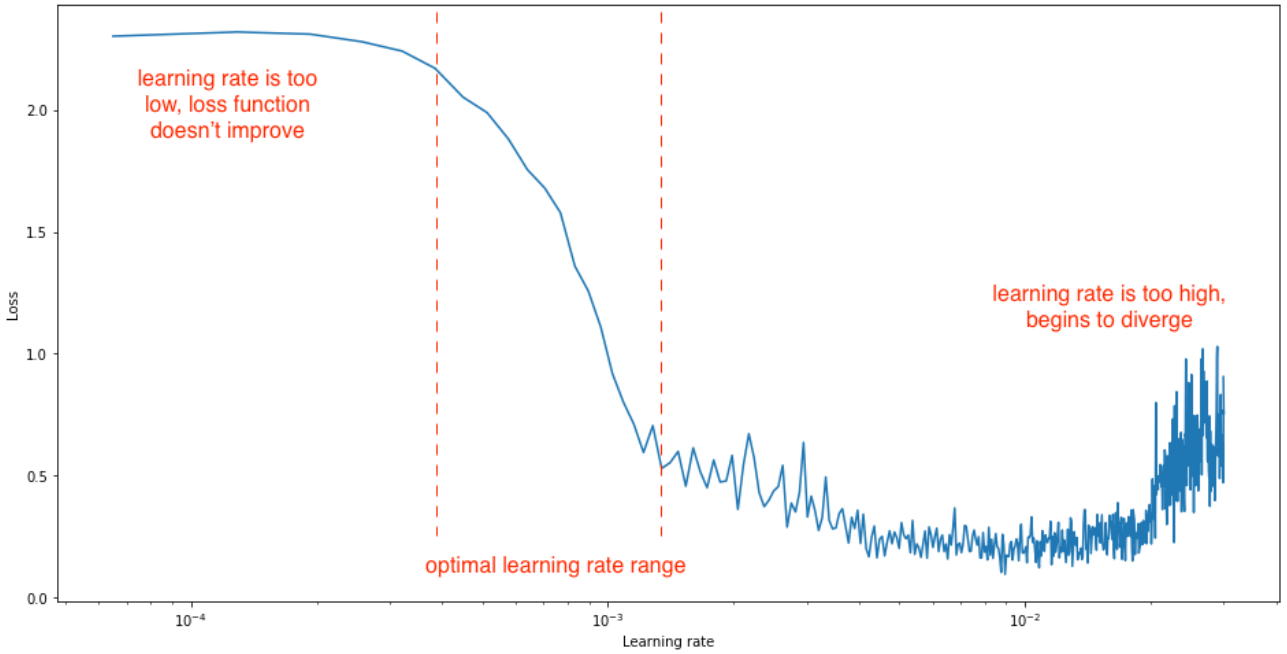


Figure 6.7: An example of an LRRT run. At first, the learning rate is too low, and the loss is plateauing. In the optimal range, the loss descends, until the loss becomes unstable and eventually explodes. Figure from [66].

tensors going into the neural network must have the same shape, meaning zero-padding to some common value is required[d].

The data itself also comes in different flavors, in a sense; one may choose the collection of pulses in an event where SRT cleaning (used in level 2 cleaning, but can be implemented at any stage; see section 3.4.1 on page 29) has been applied, which removes light that is not causally connected to the event, and must thus be considered noise.

Experiments were run on smaller sets (around 200,000 events) to facilitate quick turnaround under the assumption that adding more data will improve performance incrementally. The choice of learning rate was done via a learning rate finder, or "the Learning Rate Range Test" (LRRT) [65], an attempt at automating the process.

LRRT runs a learning rate scan—between two values chosen manually by the user—over one epoch, increasing the learning rate between each mini-batch, the training loss is recorded at each step. When the rate is too low, the training loss will plateau, then descent and finally—when the learning rate is too high—explode; the user should then choose the learning rate to be where the training loss is descendant, as that represents the most optimal range of learning rates. An example is shown in fig. 6.7.

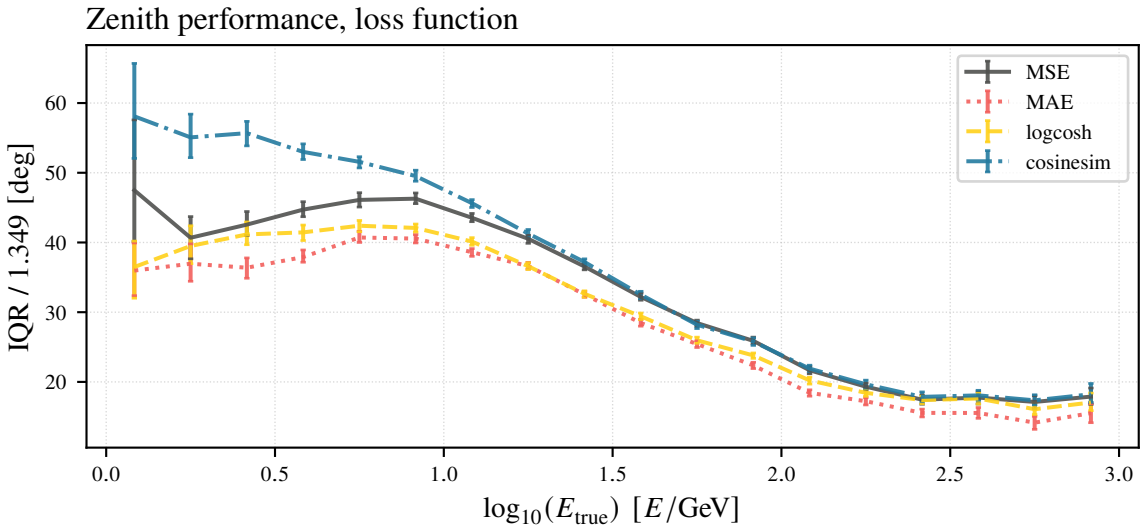


Figure 6.8: The zenith resolution over 18 energy bins from 1–1000 GeV. The lines serve to guide the eye, as the plot becomes too busy as a scatter plot with more than two groups. Zenith performance on small DeepCore dataset with three different loss functions, MSE, MAE and logcosh. Logcosh consistently performs worse than MSE, while MSE and MAE are similar at energies above 10 GeV. However, MSE seems to perform best at very low energy.

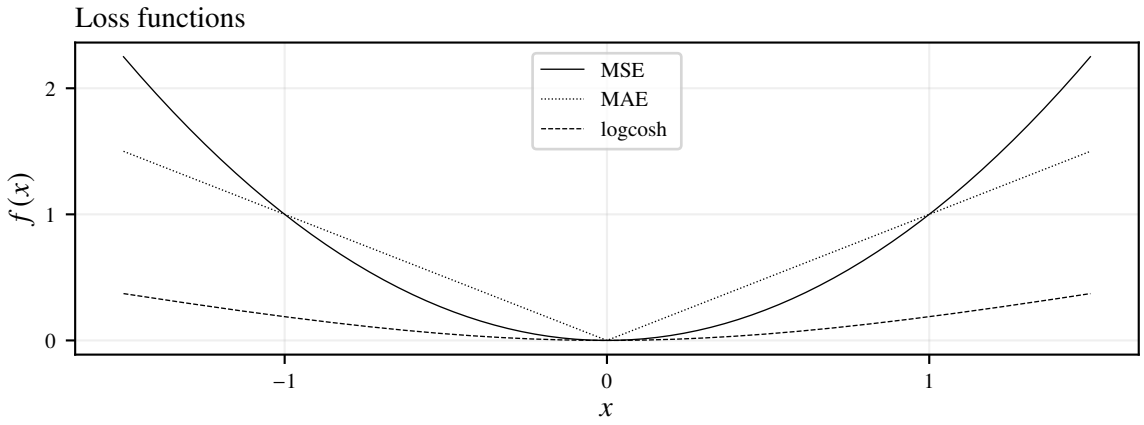


Figure 6.9: Loss functions MSE, MAE and logcosh. The outsize weight assignment of MSE is seen at values over $x = 1$, while the linear fashion and kink at $x =$ is visible for MAE. Logcosh is smooth and does not assign outsize weight to outliers.

6.4.2 Loss function

The choice of loss function has an impact on the performance of a model. A handful were tested, with the usual suspects Mean Squared Error (MSE), Mean Absolute Error (MAE) and the logarithm of the hyperbolic cosine ($\log[\cosh(x)]$, termed simply logcosh) performing the best, as shown in fig. 6.8 on the facing page using the zenith regression task as an example.

The three loss functions—MSE, MAE and logcosh—are staples of machine learning, but perform their duties differently. MSE,

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_{\text{true}} - y_{\text{reco}}\right)^2, \tag{6.3}$$

assigns loss quadratically, which means that outliers are assigned an outsize weight. Here, n is the number of samples in a mini-batch, y_{true} the true value and y_{reco} the reconstructed value. MAE,

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_{\text{true}} - y_{\text{reco}}|, \tag{6.4}$$

is an attempt to remedy this by assigning a linear weight. It has a constant derivative almost everywhere, but is non-differentiable at $y_{\text{true}} = y_{\text{reco}}$, so must be approximated by a smooth function at that point.

Logcosh loss,

$$\text{logcosh} = \sum_{i=1}^{n}\log[\cosh(y_{\text{true}} - y_{\text{reco}})], \tag{6.5}$$

is differentiable everywhere, and not as sensitive to outliers, because $\text{logcosh} \approx x^2/2$ for small x and $\text{logcosh} \approx |x| - \log(2)$ for large x, avoiding the problems of MSE.

6.4.3 Prediction type

The prediction of angles[e] can proceed in several ways.

Because the azimuthal angle wraps around at 2π, it introduces problems for the neural network because it cannot understand that e.g. 6.27 rad and 0.01 rad are points close to each other on the circle. This may be remedied by predicting the x, y and z components of the directional vector associated with an event, which has the benefit of not relying on circular statistics. The loss function may even contain a penalty term ensuring that the network attempts to predict unit vectors. This was implemented, but was not found to make a significant difference compared to no penalty.

Another avenue is to only predict the zenith angle, which does not suffer from circular wraparound, as it is restricted to $[0°, 180°]$. This leads to another issue: is it possible to input the raw detector

[d] Even the nature of the zero-padding must be decided; do you zero-pad the end of the arrays, or maybe both ends (with the detector data in the middle)?

[e] Or, rather, zenith, which is the quantity of importance for oscillation studies

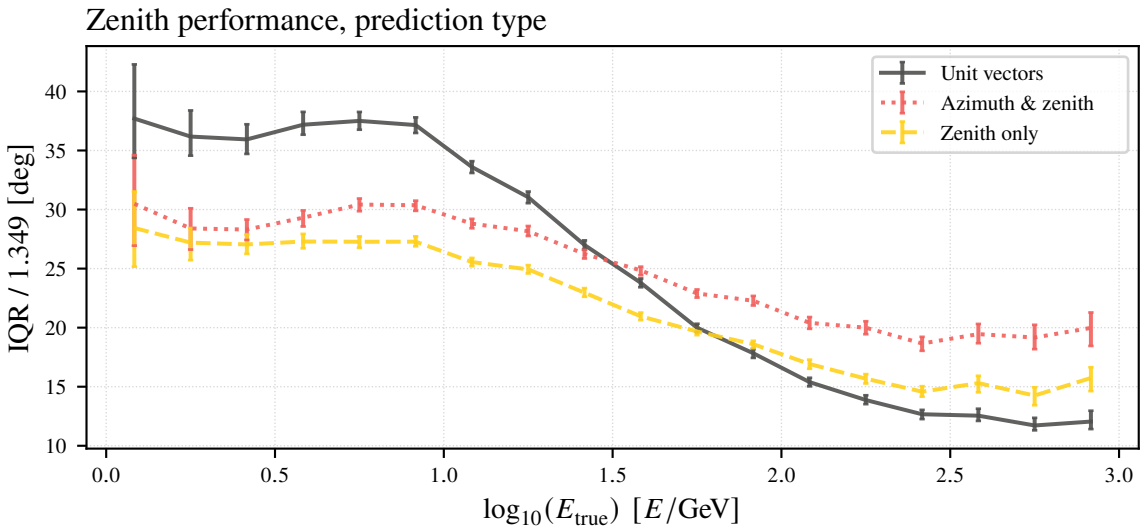


Figure 6.10: Zenith resolution performance for different prediction types. Only predicting the zenith coordinate of the neutrino is seen to perform best over the largest range of energies, while the prediction of unit vectors outperforms it at higher energies (100 GeV and above).

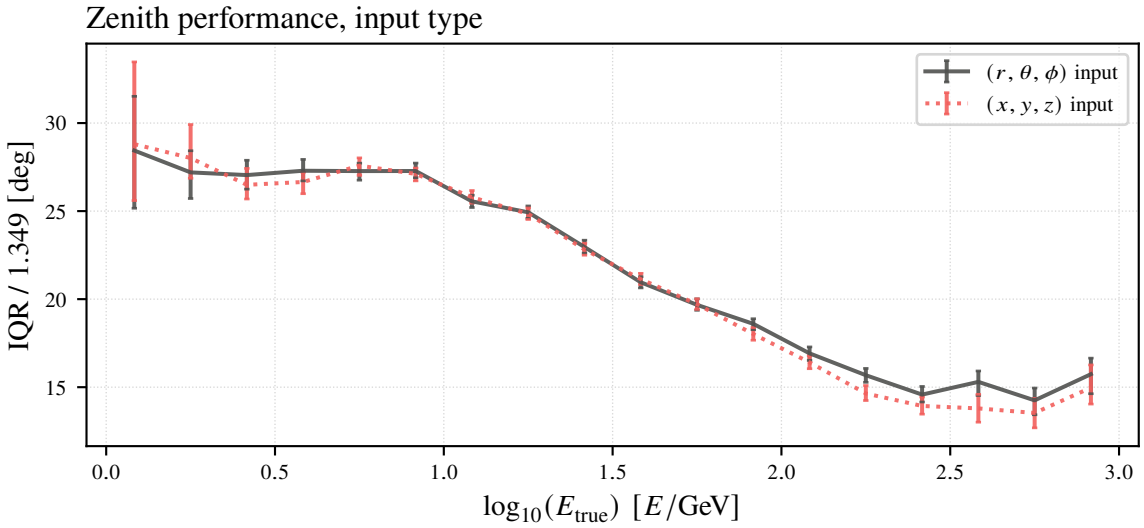
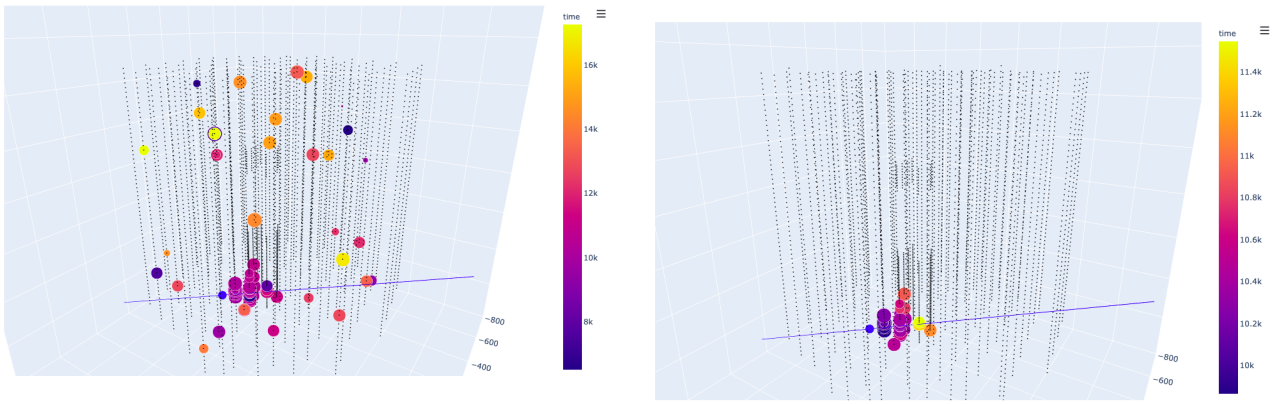


Figure 6.11: The difference between inputting the original detector data, in Cartesian coordinates, to the network, or doing a spherical transform beforehand, when predicting the neutrino's zenith directional coordinate. The performance is similar.

data—which contains Cartesian input coordinates for the DOM positions—and expect the neural network to be able to map it to a spherical coordinate system?

This was tested, as the thesis was that converting the coordinates beforehand might lessen the burden on the neural network, and the result can be seen in fig. 6.11. The performance is similar, and goes to show



(a) (b)

Figure 6.12: Event ID 124492073, a 35.8 GeV event shown in Powershovel with no SRT cleaning (fig. 6.12a) and with (fig. 6.12b). This is a track-like muon neutrino event, the neutrino path indicated by the blue line. Each black dot represents a DOM, and the pulses are colored according to the "plasma" color map with yellower pulses being later in time than bluer pulses. The size of the pulse is proportional to its charge. As is evident, SRT cleaning removes a large amount of pulses in this example, and the cleaned event leaves a sense of direction in the pulse which can be inferred by a human eye, and surely is easier to infer for a neural network. Note that the color scale is different on the two plots; this is caused by a missing feature not yet implemented in Powershovel.

the power of a universal approximator: the network will discover the correlations itself, even between coordinate transformations.

Lastly, it is possible to predict *both* the azimuthal and polar angle in the same network; this was, however, seen to lead to a slight decrease in zenith resolution, as seen in fig. 6.10 on the facing page.

6.4.4 Cleaning

Every event includes extra SRT cleaning, which removes any pulses that are unphysical (see section 3.4.1 on page 29). SRT cleaning tends to remove a large amount of pulses—as is shown in fig. 6.12—which is only noise, and should thus tend to aid neural networks in training. Networks were trained with/without SRT cleaning, and it was found to be indeed so; an example is shown in fig. 6.13 on the next page. As low energy events do not tend to leave long, finely articulated tracks in DeepCore, this cleaning improves the resolution as the network does not need to pay attention to noise.

6.4.5 Maximum event length

The tensors that are input into the neural network need to be of the same shape, and thus some padding is necessary. All tensors are therefore padded up to some pre-set maximum length, and events that are longer than this limit are instead composed of a random subsample of the total event.

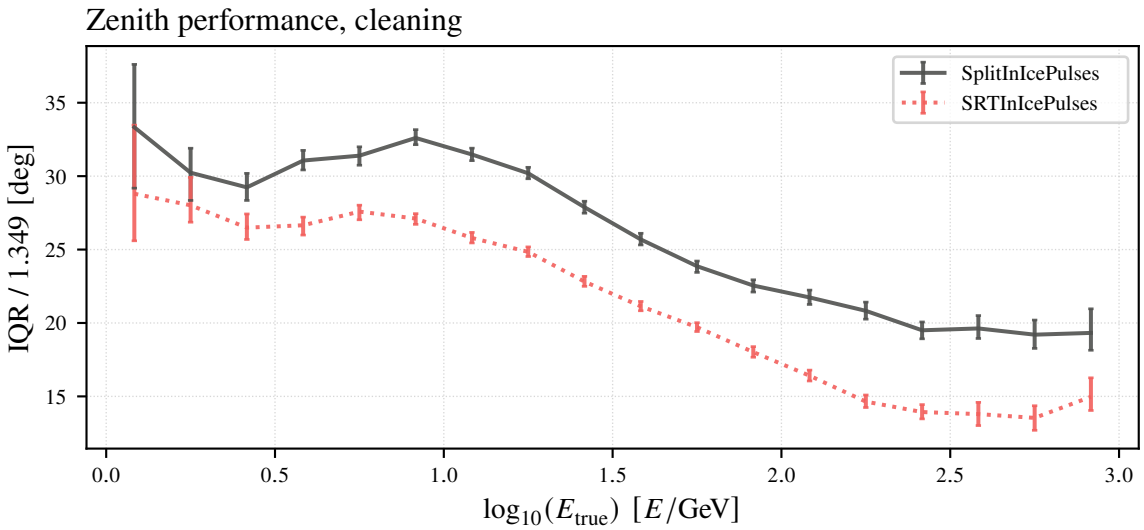
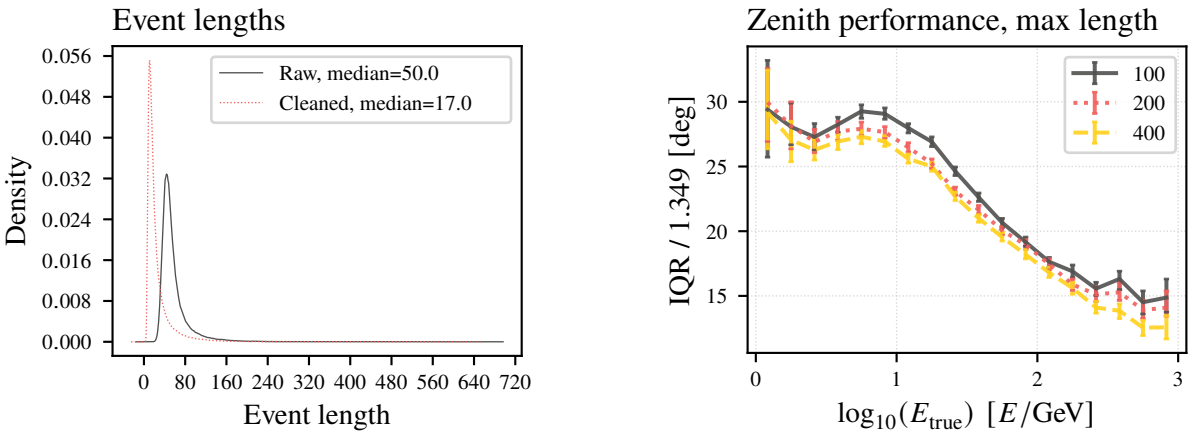


Figure 6.13: Impact of cleaning on zenith performance. SplitInIcePulses indicates events with no extra pulses removed by SRT cleaning, whilst SRTInIcePulses is the opposite. There is a clear difference in performance, which should be expected as the SRT cleaning removes unphysical pulses.



(a) **(b)**

Figure 6.14: The distributions of event lengths with/without SRT cleaning (Raw/Cleaned) as KDEs are shown in fig. 6.14a. Figure 6.14b shows the effect of different maximum event length on the performance of the network. For 200 and 400 the performance is similar, while somewhat decreased for 100. This may be caused by the limitation of a maximum length of 100 "throws away" too much event information. This is reinforced by the fact that at the highest energy bin the performance of 200 decreases, probably because higher energy events are typically longer, while the performances are very similar at low energies.

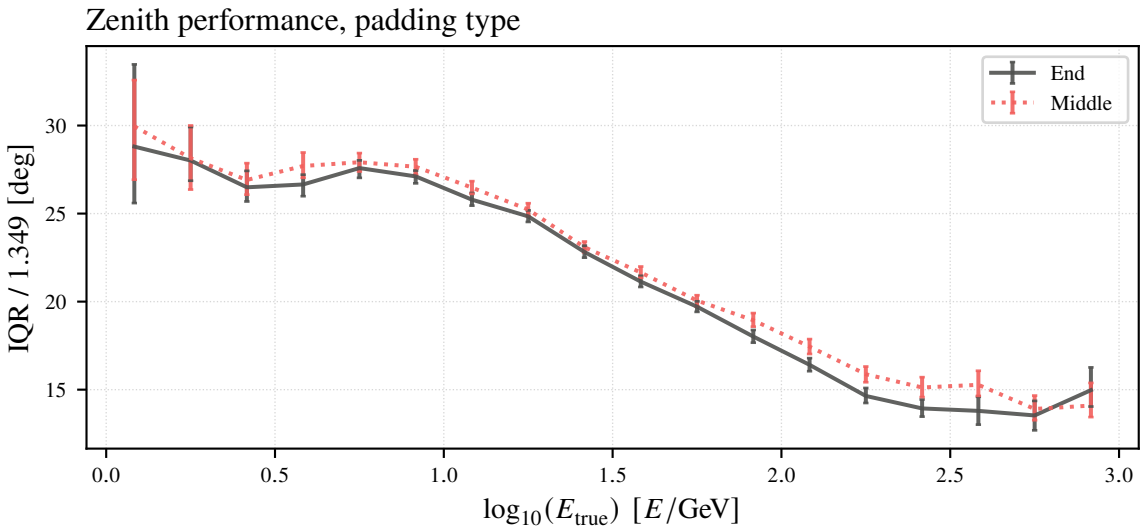


Figure 6.15: Network performance with different padding types. Because all input tensors are required to have the same shape, each event is zero-padded up to the maximum event length. "End" padding pads the end of the array, while the poorly named "middle" places the event in the middle, and pads both ends. The performance is similar for both.

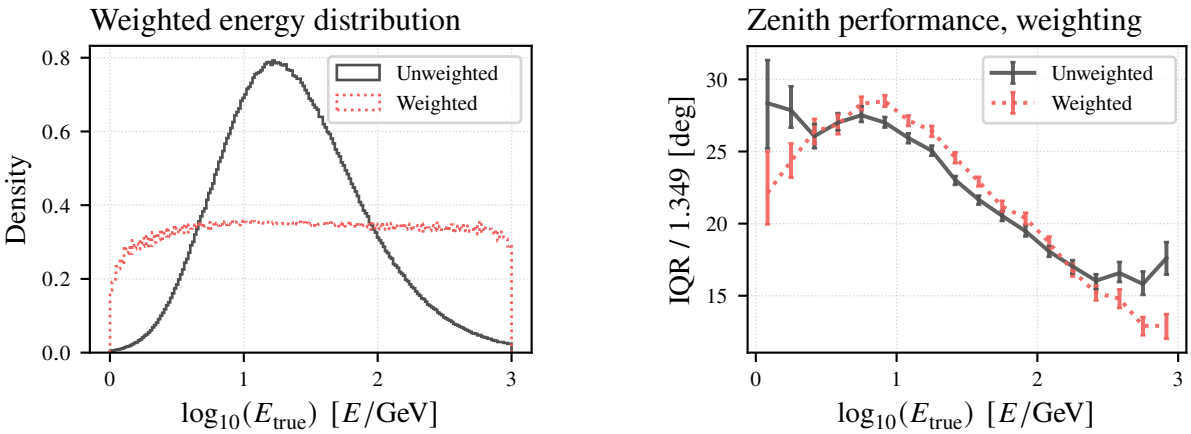
The median length of a raw event is 50 and 17 for SRT cleaned ones. 200 was chosen as the best performant maximum length, when training time, memory usage and resolution performance was taken into account; see fig. 6.14 on the facing page.

6.4.6 Array padding

The style of zero-padding was also varied, although with no discernible effect as shown in fig. 6.15. Either the event was padded with zeros after the last entry and up to the maximum length, or the event was put into the middle of the array, with both ends then padded. The network, however, seems to understand that zeros indicate no pulse, as the performance is very similar between the two.

6.4.7 Weighting

It is common practice in both machine learning and high energy physics to re-weight a distribution. In the case of machine learning, it is often done such that distribution imbalances are evened out, and for IceCube most neutrinos are around the 10–100 GeV mark, as seen in fig. 6.16a on the following page. Thus re-weighting to a uniform distribution is very useful for extremely low energy events, under 10 GeV, although performance suffers most everywhere else; see fig. 6.16 on the next page.



(a) **(b)**

Figure 6.16: Figure 6.16a shows the energy distribution of the largest DeepCore muon neutrino dataset. Superimposed is the weighted distribution, where each entry counts with the value of its weight instead of 1. The distribution has been weighted to a uniform distribution. Figure 6.16b shows performance with/without re-weighting by energy bins. The tails naturally improve their performance, while the bulk performs worse.

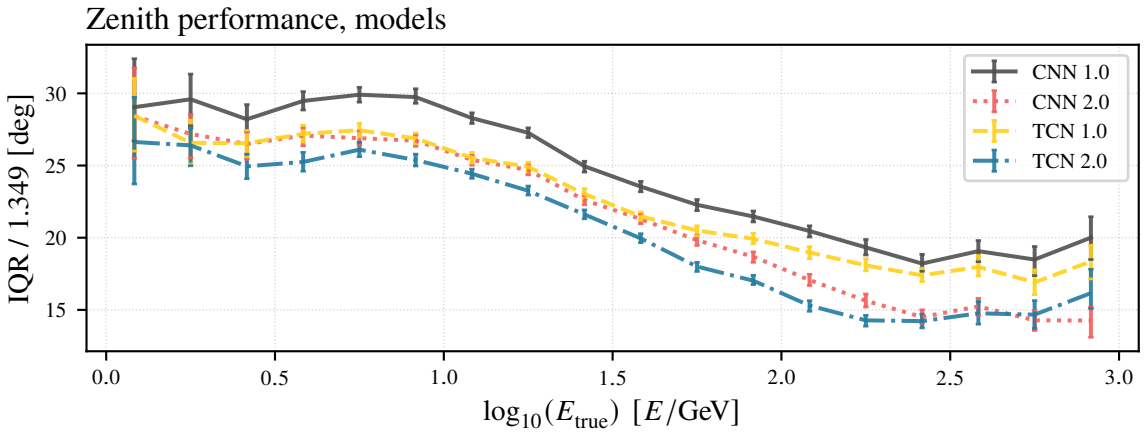


Figure 6.17: Performance of four different networks. The low-parameter TCN 1.0 performs promising vs. CNNs with orders of magnitude more parameters, but fails at higher energies; this is remedied by TCN 2.0, which trades low amount of parameters for performance.

6.4.8 Model architecture

The preceding discussion holds as well for zenith prediction as for energy prediction (bar prediction type and input type, which are not energy relevant). The same goes for the design of the network; where performance increased on zenith prediction, performance increased on energy prediction.

Several designs have been tested out, both of the CNN and TCN variety. The first working version, "CNN 1.0", consists of five convolutional layers (kernel size 5), each doubling the number of filters starting with 32, using ReLUs, max pool and batch normalization in each block. This is then fed through four fully connected layers until terminating in an output layer.

The network was found to much improve by removing most convolutional layers, with "CNN 2.0" only containing two of these, with two fully connected layers afterwards. The difference in performance, seen in fig. 6.17 on the facing page, is rather striking everywhere but the lowest energy bins.

CNN 1.0 is a large network, containing 5,735,009 trainable parameters, cut down to 1,718,241 in CNN 2.0.

"TCN 1.0" is a temporal convolutional network with one stack of 64 filters (kernel size 2), dilations (1, 2, 4, 8, 16, 32), ReLUs and causal padding. It only has 92,161 trainable parameters, yet performs as well as CNN 2.0 until around 100 GeV. Upping the filters to 256, the number of stacks to 2, and adding batch normalization results in 3,038,209 trainable parameters for "TCN 2.0"[f], the winning network of the bunch.

These models are compared visually in fig. 6.17 on the preceding page. With the best network—architecture and hyperparameters—in hand, the next task is to compare it to IceCube's current reconstruction algorithm, Retro Reco.

[f] More—many more!—network designs were trialed than two CNNs and two TCNs; for reasons of brevity, they are not discussed here.

Chapter 7

Results

And now, with the board set and the pieces moved, we come to it at last: the results.

The following results were trained on the full muon neutrino charged current dataset, and tested on the corresponding test set using the aforementioned TCN 2.0 network. The network performs very well on low energy events, up to around 50 GeV, whereafter it performs worse than Retro Reco, in the zenith case by a significant amount.

Figures 7.1 and 7.2 on the following page and on page 73 shows error distributions from selected bins, covering where CubeFlow performs best, and where Retro Reco does; it is at once comforting and amazing to see how similar the errors are distributed by energy. Comforting because the two wildly different algorithms seem to face the same challenges, amazing because one algorithm (Retro Reco) a priori knows physics, while the other (CubeFlow) entirely does not, but infers it based on the truth.

In the energy case, both algorithms skew at low and high energies, predicting too high values at low energy, too low at high. One might surmise that CubeFlow suffers there because of the lack of training data in these ranges, but the similar performance of Retro Reco suggests that might not be the whole story, and maybe the problem is endemic to the detector.

This bias is also clearly seen in fig. 7.3 on page 74, both the top and bottom histograms. CubeFlow, while biased similarly, is less so at the most populated energy range around 10–50 GeV.

Figure 7.4 on page 75 shows the zenith angle reconstruction and error histograms. These are not biased at lower energies, although performance of course suffers there because low energy events do not leave a long track-like signature. However, they are biased at as a function of the true polar angle. This effect is most pronounced for CubeFlow, and may be caused by the fact that the dataset is mostly composed of up-going neutrinos, leaving less training data for down-going (low zenith value) events. Relatively fewer neutrinos are coming straight up (as seen in fig. 5.2 on page 48) explaining the worse performance at high zenith values.

It should be noted that the lower histograms in figs. 7.3 and 7.4 on page 74 and on page 75 is basically a two dimensional view of the data summarized in figs. 7.5 and 7.6 on page 76 and all other figures in that vein.

To gauge the contribution of features, permutation importance has been employed. The straight-forward way of doing this would be to train a model with/without a certain feature, and compare the loss values between the two runs. Permutation importance works by running $n + 1$ inference passes with a trained model, where n is the number of features. One run is a normal inference, and represents the baseline while all following inference passes shuffle the values of one feature which becomes random noise for

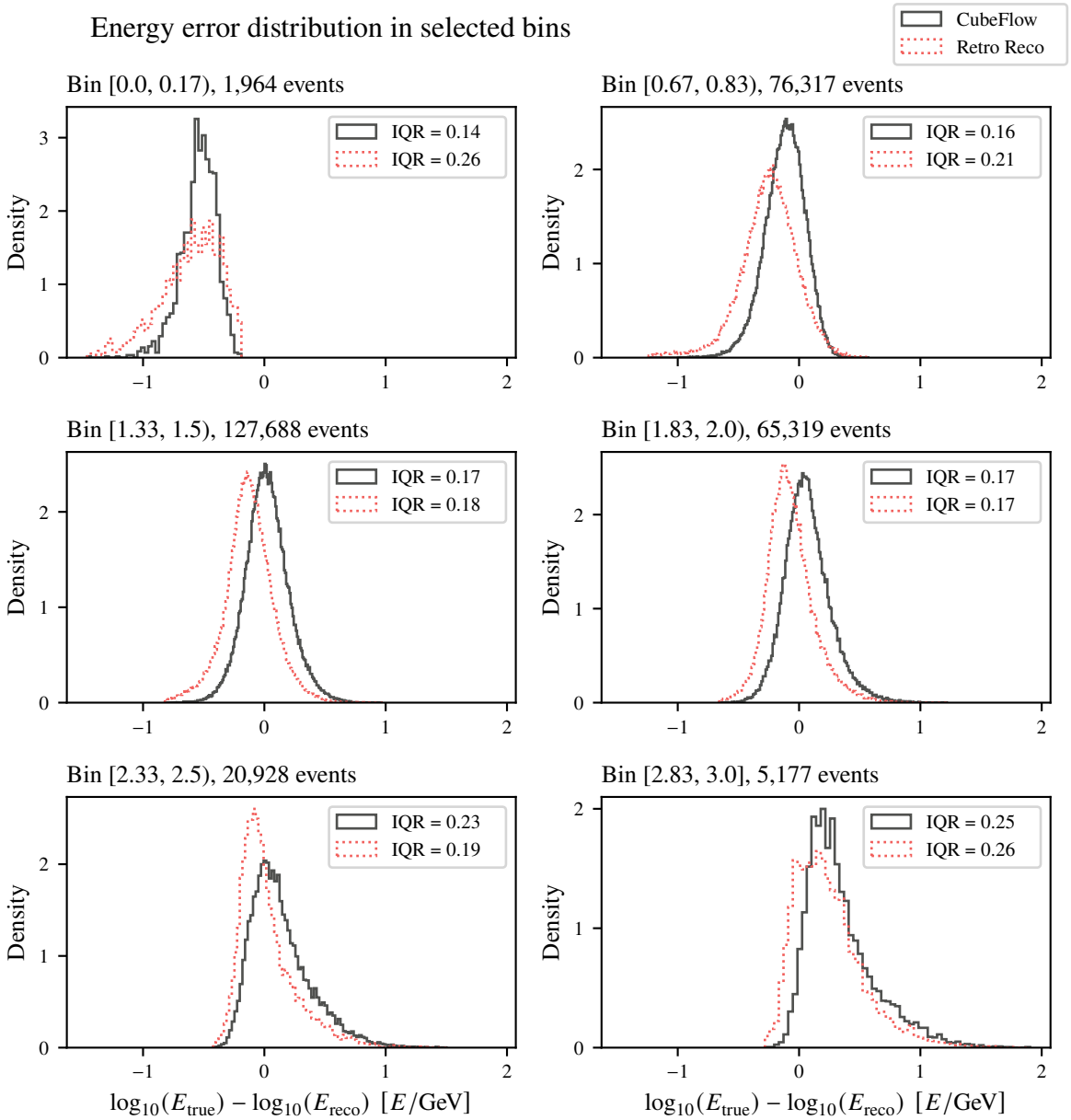


Figure 7.1: Energy error distribution in certain selected bins, representative of the overall performance difference. At low energy CubeFlow performs better, seen by a narrower distribution than that of Retro Reco. It is evident that both algorithms have an inherent bias: at low energy they tend to overshoot the energy, while at higher energies they undershoot. In the case of the neural network, this is probably caused by the relative lack of training data in the tails.

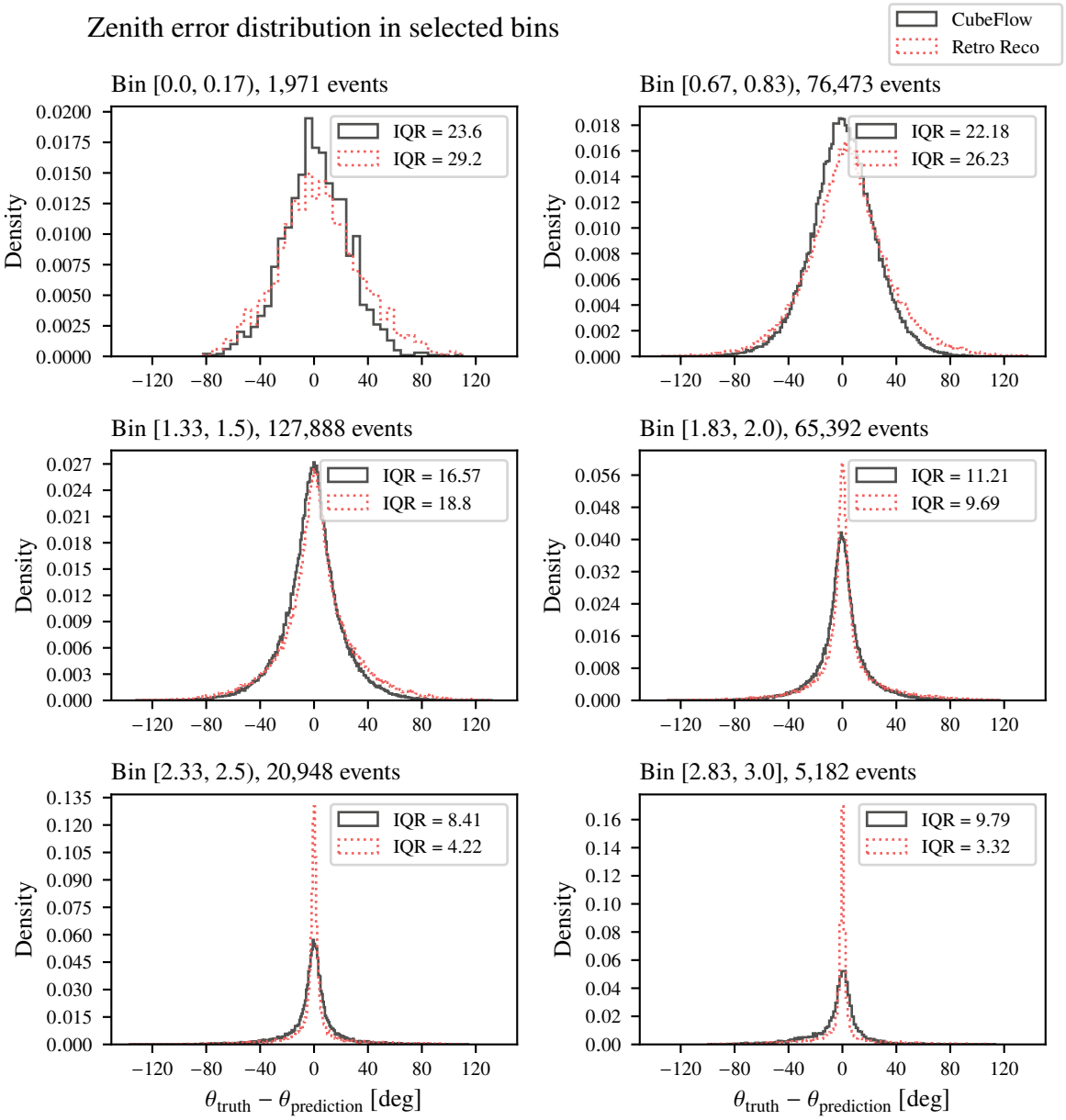


Figure 7.2: Zenith error distribution in certain selected bins, representative of the overall performance difference. Contrary to the energy case, there is no large inherent bias in zenith reconstruction, at least as a function of true energy. The width of the CubeFlow error distribution can be seen to be narrower than the Retro Reco case.

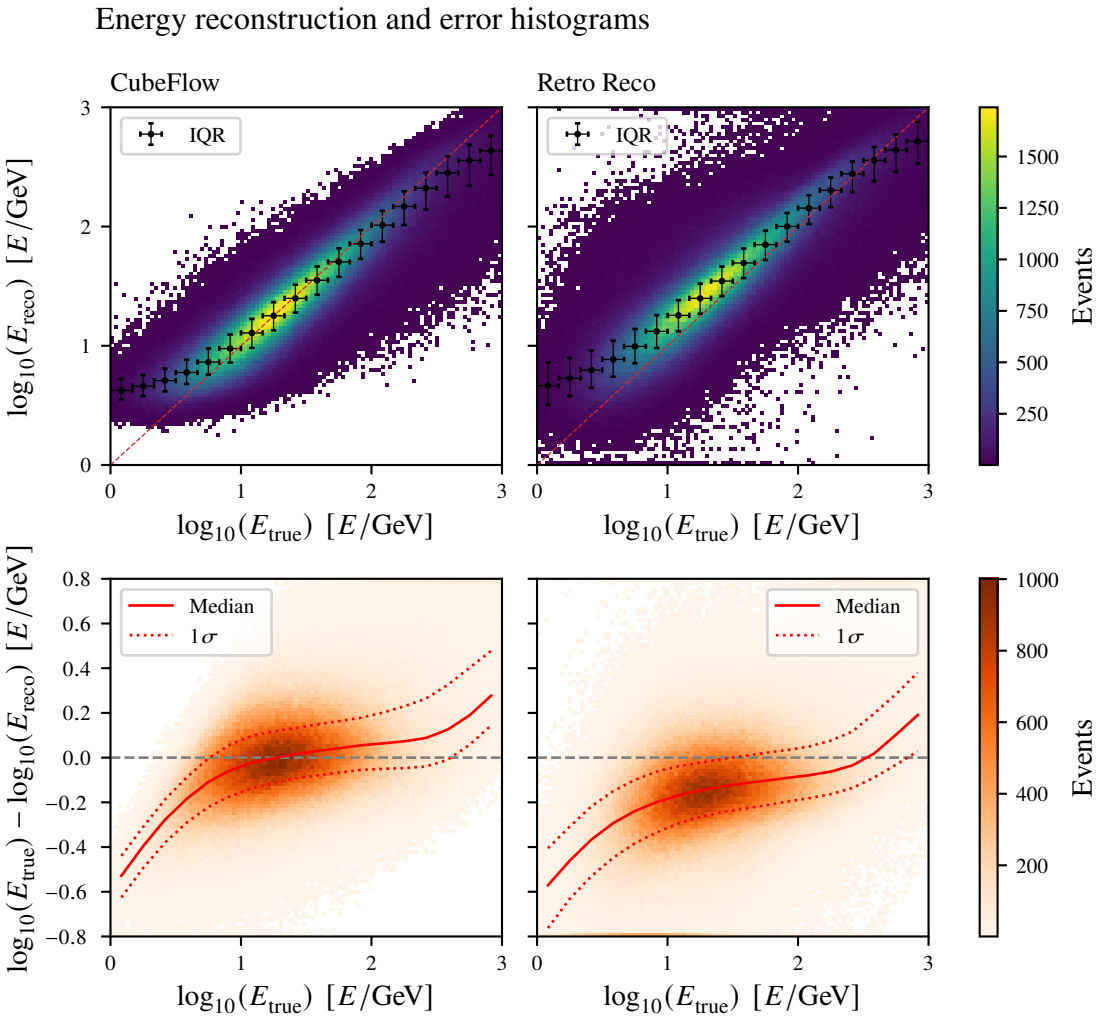


Figure 7.3: Energy prediction (top row) and error (bottom row) 2D histograms for CubeFlow (left column) and Retro Reco (right column). In the energy case, these two types of histograms are basically the same, only shifted. However, for consistency both are provided, and the top histograms are in a sense unprocessed (we calculate no error metric), so any issues with the error calculation would show up here; thus they are good to have. The bias can, as expected, be seen in both types of histograms, and the top row makes it very clear that CubeFlow lies along the diagonal line more than Retro Reco, indicating less bias. The black dots on the top row histograms show the median in the bin defined by what would typically be the x error bars. The y error bars show the IQR in that same bin, while the dotted red diagonal line traces out $y = x$, where perfect reconstructions lie. For the error histograms (bottom row) the solid red line shows the median, while the dotted red lines are 1σ. The grey dotted line lies along $y = 0$, where the reconstruction is error-free.

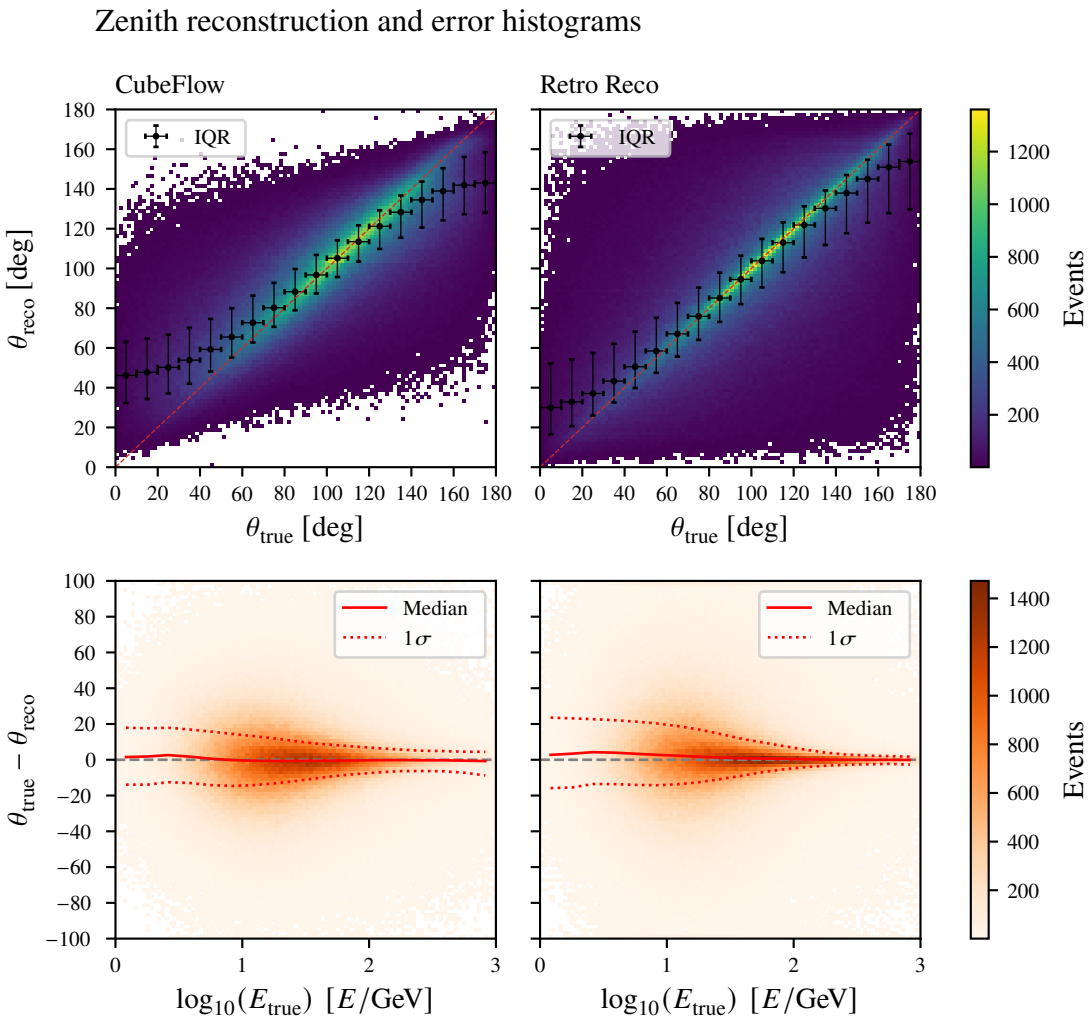


Figure 7.4: Zenith prediction (top row) and error (bottom row) 2D histograms for CubeFlow (left column) and Retro Reco (right column). The top row histograms clearly show a bias at low and high zenith values, most pronounced for CubeFlow. Bottom row histograms show no bias as a function of energy, while the resolution is seen to narrow with higher energy. The black dots on the top row histograms show the median in the bin defined by what would typically be the x error bars. The y error bars show the IQR in that same bin, while the dotted red diagonal line traces out $y = x$, where perfect reconstructions lie. For the error histograms (bottom row) the solid red line shows the median, while the dotted red lines are 1σ. The grey dotted line lies along $y = 0$, where the reconstruction is error-free.

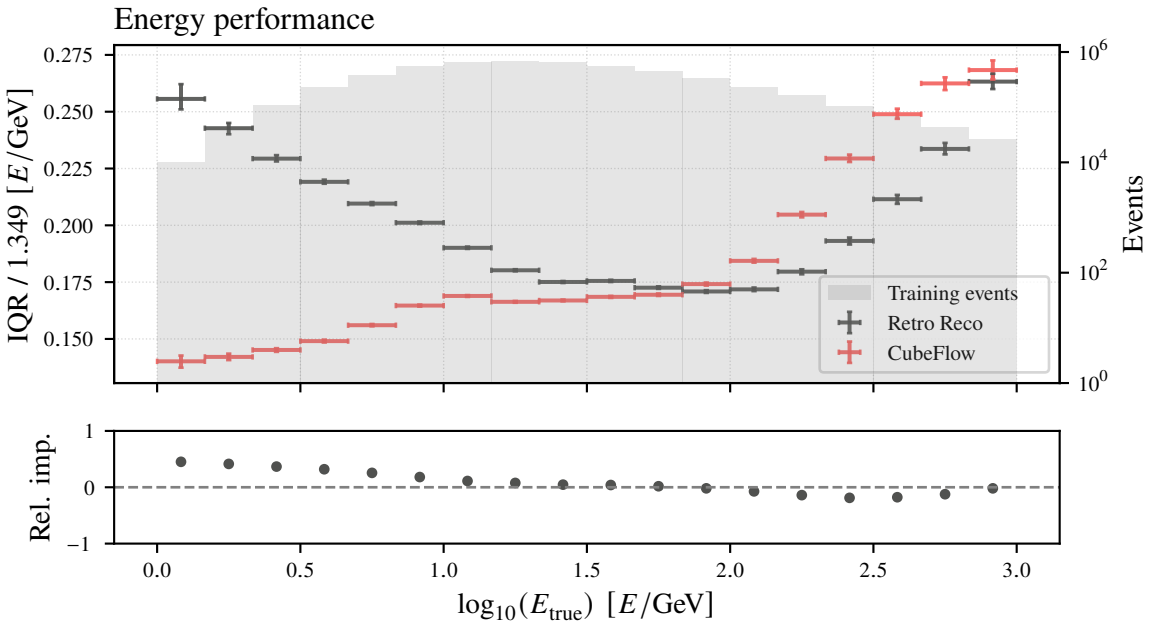


Figure 7.5: The resolution performance of CubeFlow (red) and Retro Reco (black) for energy reconstruction. CubeFlow performs much better at low energies, while Retro Reco wins at higher energies. The amount of training events in each energy bin is superimposed (using a log scale) with grey bars. The lower plot shows the improvement of CubeFlow relative to Retro Reco, constrained between −100 % and 100 %.

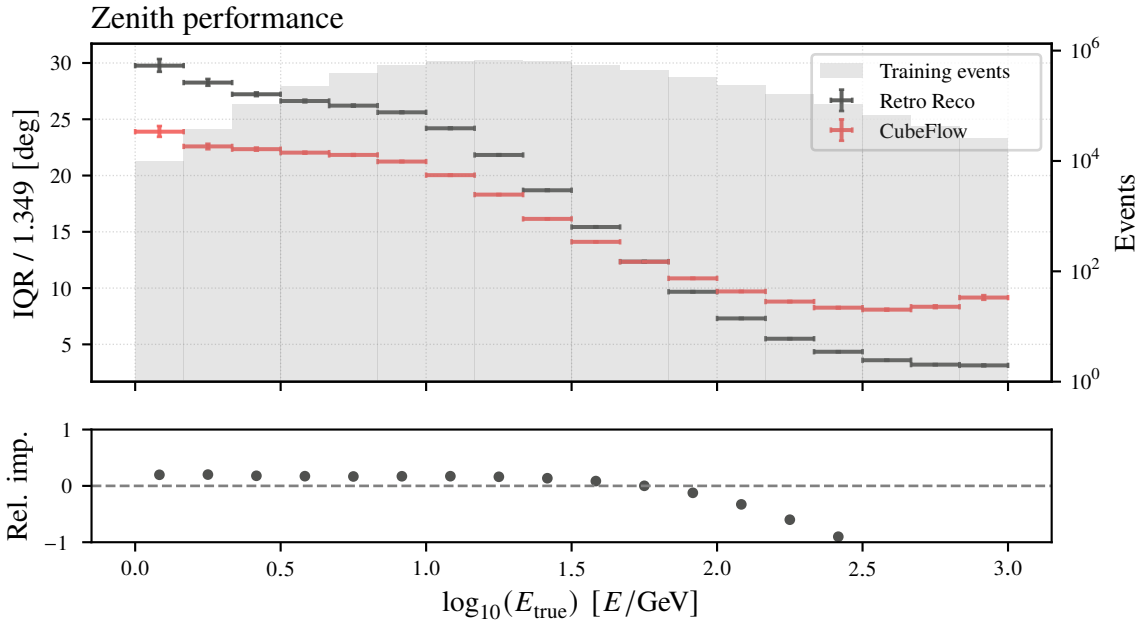


Figure 7.6: The resolution performance of CubeFlow (red) and Retro Reco (black) for zenith reconstruction. CubeFlow performs better at low energies, while Retro Reco wins at higher energies. The amount of training events in each energy bin is superimposed (using a log scale) with grey bars. The lower plot shows the improvement of CubeFlow relative to Retro Reco, constrained between −100 % and 100 %.

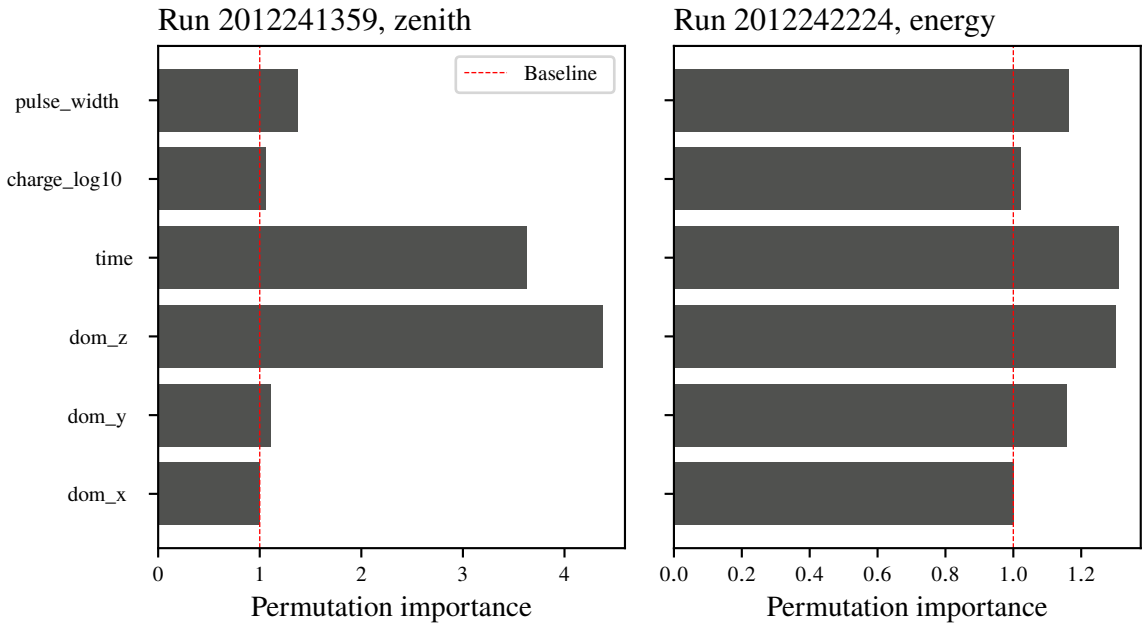


Figure 7.7: Permutation importance for zenith reconstruction (left) and energy reconstruction (right). At no great surprise the zenith reconstruction relies heavily on the z-coordinate of the activated DOMs, with the relative DOM activation time second-most important. Both the z- and the y-coordinates are important in the energy case, with time continuing to be important. Pulse width is critical to the performance of the network as well.

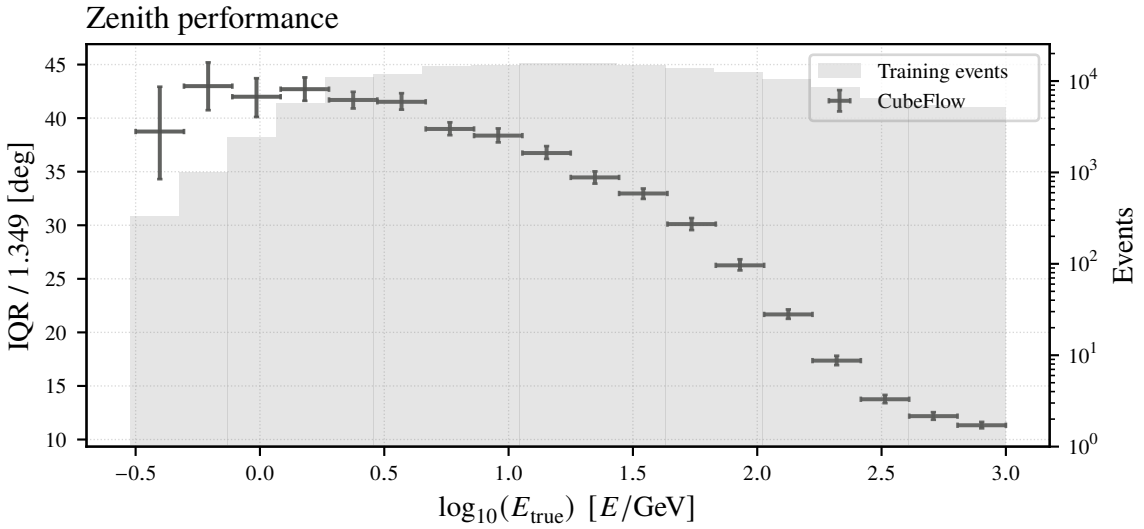


Figure 7.8: Zenith resolution performance of CNN 2.0 on the upgrade dataset. The performance is worse than on DeepCore data, even with new hardware features fed to the network, probably due to excessive noise. The network *does* have some predictive power, though, starting to improve performance around 10 GeV. This is comforting for the future, as a noise removal pipeline akin to oscNext should drastically improve the results.

the network. In this way one may see the value of a feature; if the loss does not change, the feature does not contribute to the networks inference capabilities. If, however, the loss suffers, the network relies on the feature being present, and the degree to which it does can be quantified as the permuted loss value divided by the baseline—a high permutation importance value thus represents an important feature.

Figure 7.7 on the preceding page shows the application of permutation importance to this problem. It is unsurprising that the z-coordinate of the activated DOMs plays a significant role in determining the zenith value of a neutrino, owing to the fact that $z = \cos\theta$. Relative DOM activation times is similarly important, also not surprising as it gives the order in which the pulses arrive.

The energy reconstruction also uses the y-coordinate to a larger degree than the zenith reconstruction, because the movement in the plane is probably important for the reconstruction. In both cases pulse width seems to be important, related to the resolution of the pulse information gathered by the DOM.

Figures 7.5 and 7.6 on page 76 show the overall performance of the best performing network, with the best performing hyperparameters as outlined in chapter 6 on page 55. Until around 50 GeV CubeFlow outperforms Retro Reco, both in energy and zenith reconstruction. In the zenith case the performance drops off rapidly, and it should be noted that the lower plot—which shows the relative improvement—has been constrained to ±100 %.

The CNN 2.0 algorithm turned out to work best on the Upgrade dataset. Here, only zenith reconstruction (see fig. 7.8) gave some usable results, while energy construction showed poor predictive power as seen in fig. 7.9 on the facing page. The network tends to predict one value around 30 GeV for events where it cannot do a meaningful reconstruction, thereby minimizing the loss. This is probably caused by the excessive amount of noise in the dataset, as no cuts akin to oscNext have been performed on the set (no similar pipeline exists). However, as expected the methodology transferred with no issues,

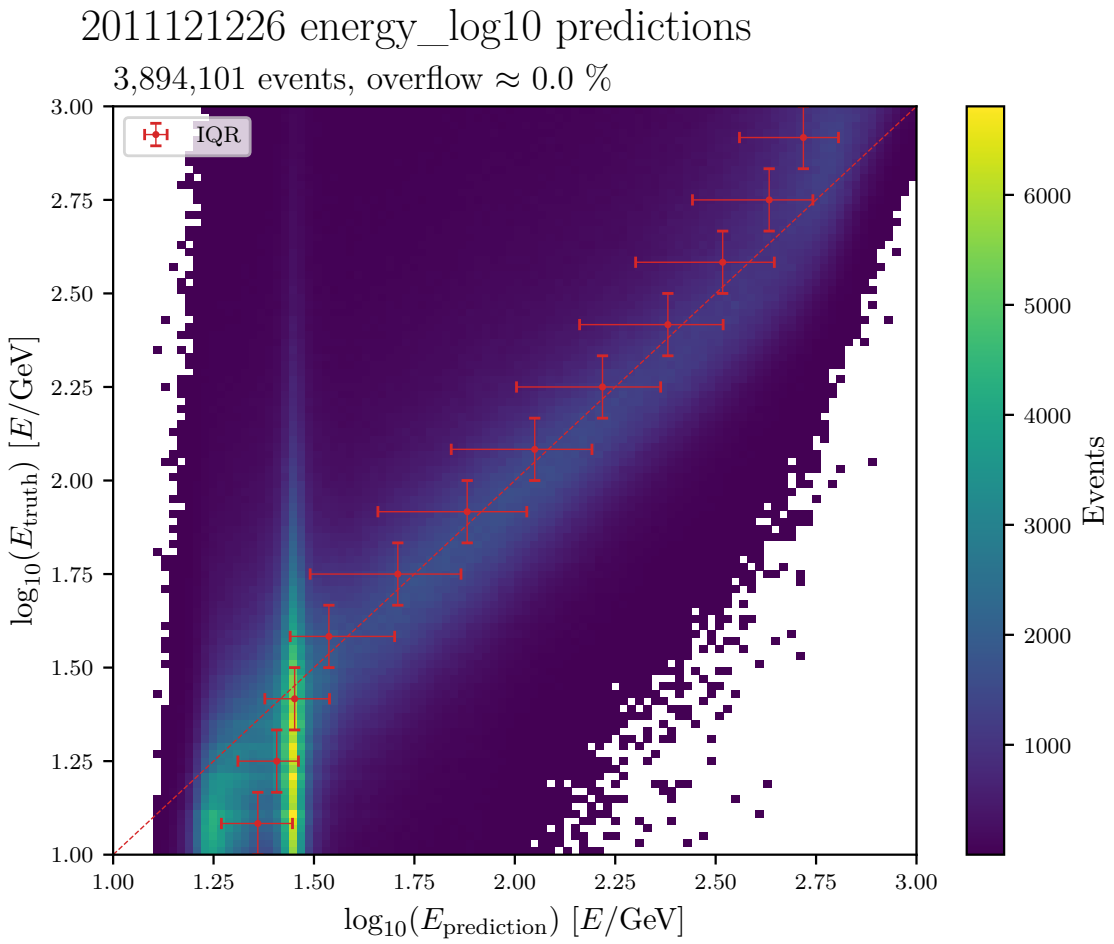


Figure 7.9: An example from an energy reconstruction run on the upgrade dataset. The network does not perform well, probably because of noise, defaulting to a loss-minimizing value for events where no clear reconstruction can be performed. This should be remedied by noise removal.

amounting to simply adding new features (the directional pointing vectors of the PMTs) as inputs to the network; this bodes well for the future of deep learning as applied to the Upgrade set, pending work on cuts to bring the set in line with real data and the removal of noise.

Chapter 8

Summary and outlook

If the neural network approach only performed as well, or better than Retro Reco, it would be a contender to replace it. This is firstly because the inference speed, at 15,000 events per second, is vastly improved compared to the minutes it takes Retro Reco to reconstruct an event. Secondly, a neural network has less trouble generalizing to new detector states: whereas Retro Reco relies on some hypothesis and accompanying tables, the neural network simply needs data. Thus where the old-school algorithm is simply unable to work on Upgrade data without a re-write, CubeFlow would only need training on Upgrade MC data; a feat that can be accomplished in weeks if not days, if the code and hardware infrastructure is already in place. Even though a bespoke, old-school physics-based algorithm might work better, a neural network ally might get the collaboration up and running quickly, while the algorithm is worked out. As such, there is no reason *not* to use neural networks in IceCube.

Although the network presented in this thesis is by no means perfect, it does show significant improvement at low energies, relevant for oscillation analysis. The network itself is simple enough, so there are many avenues for improvement. This work would be significantly helped by the right tooling, such as CubeDB and Powershovel, because groups working on different algorithms would be able to compare on even terms, and use the same data with common references (event IDs) resting assured that the same events are used for training and testing across the board.

The scalability of neural networks is evident, as the Upgrade data was added and used as late as November; a working algorithm, hampered by noise, was up and running in mere minutes after the dataset was created, because it essentially only represents new data and the addition of new features (PMT directional vectors). Added hardware sensitivity coupled with enhanced reconstruction through ML will allow the Upgrade to achieve, and even succeed, in its goal of world leading measurements of $\sin^2\theta_{23}$ and Δm^2_{32} through muon neutrino survival and muon to tau oscillations at the oscillation extremum around 25 GeV, a range where CubeFlow performs well even in DeepCore.

Supernova alarms is another case where the fast reconstruction of an ML algorithm is useful. The promise of multi-messenger astronomy is the many faceted observation of cosmic events, and a GPU on the South Pole may be able to point towards something interesting in the sky with milliseconds of increased neutrino activity, alerting electromagnetic telescopes to start turning to look.

Of course the methods lack any validation in real data, a fact that might be remedied in time by observing the moon's shadow; because the moon stop most muons one would expect a deficit in its general direction, and the resolution can quantify the performance of the network.

In any case, neural networks have arrived in every area of physics, and can no longer be overlooked. Newer methods, such as generative adversarial networks, are beginning to be used at the LHC as a

supplement to MC data generation because it can generate new data from random noise by running a min-max game with a discriminator network; this reduces the computational burden of MC generation quite significantly. This might also be of interest to IceCube, and as soon as neural networks are a staple of the collaboration toolbox, such ideas have an easier time being accepted.

Bibliography

[1] Animesh Chatterjee, Moon Moon Devi, Monojit Ghosh, Reetanjali Moharana, and Sushant K Raut, "Probing cp violation with the three years ultra-high energy neutrinos from icecube", arXiv preprint arXiv:1312.6593 (2013) (cited on page 9).

[2] S Baur, "Dark matter searches with the icecube upgrade", arXiv preprint arXiv:1908.08236 (2019) (cited on page 9).

[3] MG Aartsen, M Ackermann, J Adams, and JA Aguilar, "Time-integrated neutrino source searches with 10 years of icecube data", Physical reviewă (2020) (cited on page 9).

[4] L Köpke, "Supernova neutrino detection with icecube", arXiv preprint arXiv:1106.6225 (2011) (cited on page 9).

[5] M Leuermann, M Krämer, and CHV Wiebusch, "Testing the neutrino mass ordering with icecube deepcore", s3.cern.ch (2018) (cited on pages 9, 20).

[6] M Raghu and E Schmidt, "A survey of deep learning for scientific discovery", arXiv preprint arXiv:2003.11755 (2020) (cited on page 9).

[7] A Ishihara, "The icecube upgrade - design and science goals", arXiv preprint arXiv:1908.09441 (2019) (cited on pages 10, 32, 34).

[8] K Abe, R Akutsu, A Ali, C Alt, and C Andreopoulos, "Constraint on the matter-antimatter symmetry-violating phase in neutrino oscillations", arXiv preprint arXivă (2019) (cited on page 11).

[9] Wikimedia Foundation, *File:standard model of elementary particles.svg*, `https://commons.wikimedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg` (visited on 12/29/2020) (cited on page 12).

[10] Wikimedia Foundation, *File:elementary particle interactions in the standard model.png*, `https://commons.wikimedia.org/wiki/File:Elementary_particle_interactions_in_the_Standard_Model.png` (visited on 12/29/2020) (cited on page 12).

[11] D Decamp et al., "Determination of the number of light neutrino species", Physics Letters B **231**, 519–529 (1989) (cited on page 12).

[12] Wikimedia Foundation, *File:beta spectrum of rae.jpg*, `https://en.wikipedia.org/wiki/File:Beta_spectrum_of_RaE.jpg` (visited on 12/29/2020) (cited on page 14).

[13] Chien-Shiung Wu, Ernest Ambler, RW Hayward, DD Hoppes, and Ralph Percy Hudson, "Experimental test of parity conservation in beta decay", Physical review **105**, 1413 (1957) (cited on page 14).

[14] Wikimedia Foundation, *File:right left helicity.jpg*, https://en.wikipedia.org/wiki/File:Right_left_helicity.jpg (visited on 12/29/2020) (cited on page 14).

[15] Alexey Boyarsky, M Drewes, T Lasserre, S Mertens, and O Ruchayskiy, "Sterile neutrino dark matter", Progress in Particle and Nuclear Physics **104**, 1–45 (2019) (cited on page 15).

[16] Mark Thomson, *Modern particle physics* (Cambridge University Press, 2013) (cited on pages 16–17, 20).

[17] Wikimedia Foundation, *File:oscillations electron long.svg*, https://en.wikipedia.org/wiki/File:Oscillations_electron_long.svg (visited on 12/29/2020) (cited on page 19).

[18] Niels Bohr Institute, *Neutrino oscillation*, https://www.nbi.ku.dk/english/research/experimental-particle-physics/icecube/neutrino-oscillation/ (visited on 12/29/2020) (cited on page 19).

[19] MG Aartsen et al., "Neutrino oscillation studies with icecube-deepcore", Nuclear Physics B **908**, 161–177 (2016) (cited on page 20).

[20] nu-fit.org, *V5.0: three-neutrino fit based on data available in july 2020*, http://www.nu-fit.org/?q=node/228 (visited on 12/29/2020) (cited on page 20).

[21] Hitoshi Murayama, *Hitoshi murayama*, http://hitoshi.berkeley.edu (visited on 12/29/2020) (cited on page 21).

[22] Wikipedia, *Fil:cherenkov wavefront.svg*, https://da.wikipedia.org/wiki/Fil:Cherenkov_Wavefront.svg (visited on 12/29/2020) (cited on page 22).

[23] PhysicsOpenLab, *Diy water cherenkov detector*, http://physicsopenlab.org/2016/04/24/diy-cherenkov-detector/ (visited on 12/29/2020) (cited on page 22).

[24] Mark Bowen, *The telescope in the ice* (St. Martins Press, 2017), page 448 (cited on page 23).

[25] Wikimedia Commons, *File:icecube-architecture-diagram2009.png*, https://commons.wikimedia.org/wiki/File:Icecube-architecture-diagram2009.PNG (visited on 12/29/2020) (cited on page 24).

[26] IceCube Neutrino Observatory, *The detection of neutrinos in icecube*, https://masterclass.icecube.wisc.edu/en/learn/detecting-neutrinos (visited on 12/29/2020) (cited on page 24).

[27] Markus Ahlers, Klaus Helbing, and Carlos Pérez de los Heros, "Probing particle physics with icecube", The European Physical Journal C **78**, 1–51 (2018) (cited on pages 25–26).

[28] Christian Spiering, "Towards high-energy neutrino astronomy", in *From ultra rays to astroparticles* (Springer, 2012), pages 231–263 (cited on page 25).

[29] *Event triggering in the icecube data acquisition system*, Vol. AIP Conference Proceedings 1630(1) (American Institute of Physics, 2014), pages 154–157 (cited on pages 24, 28–29).

[30] J Ahrens et al., "Muon track reconstruction and data selection techniques in amanda", Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **524**, 169–194 (2004) (cited on pages 25, 28).

[31] Kevin J Meagher, "Neutrino astronomy with icecube", Proceedings of the International Astronomical Union **12**, 322–329 (2016) (cited on page 27).

[32] MG Aartsen et al., "The icecube neutrino observatory: instrumentation and online systems", Journal of Instrumentation **12**, P03012 (2017) (cited on pages 26–27).

[33] Wikimedia Commons, *File:icecube dom taklampa.jpg*, `https://commons.wikimedia.org/wiki/File:ICECUBE_dom_taklampa.jpg` (visited on 12/29/2020) (cited on page 28).

[34] Rasha Abbasi et al., "The icecube data acquisition system: signal capture, digitization, and timestamping", Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **601**, 294–316 (2009) (cited on page 28).

[35] MG Aartsen et al., "Improvement in fast particle track reconstruction with robust statistics", Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **736**, 143–149 (2014) (cited on page 28).

[36] MG Aartsen et al., "Characterization of the atmospheric muon flux in icecube", Astroparticle physics **78**, 1–27 (2016) (cited on page 28).

[37] Summer Blot et al., "Oscnext (v00.04)", (2020) (cited on pages 29, 31).

[38] IceCube Neutrino Observatory, *Diagrams*, `https://icecube.wisc.edu/gallery/view/140` (visited on 12/29/2020) (cited on page 32).

[39] Rasha Abbasi et al., "The design and performance of icecube deepcore", Astroparticle physics **35**, 615–624 (2012) (cited on page 32).

[40] IceCube Neutrino Observatory, *Nsf approves funding for icecube upgrade*, `https://icecube.wisc.edu/gallery/press/view/2328` (visited on 12/29/2020) (cited on page 33).

[41] *Pulse: self-supervised photo upsampling via latent space exploration of generative models*, Vol. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (2020), pages 2437–2445 (cited on page 36).

[42] Chicke3gg, `https://twitter.com/Chicken3gg/status/1274314622447820801` (visited on 12/29/2020) (cited on page 36).

[43] Warren S McCulloch and Walter Pitts, "A logical calculus of the ideas immanent in nervous activity", The bulletin of mathematical biophysics **5**, 115–133 (1943) (cited on page 36).

[44] Diederik P Kingma and Jimmy Ba, "Adam: a method for stochastic optimization", arXiv preprint arXiv:1412.6980 (2014) (cited on page 38).

[45] Apple, *Differentiable programming manifesto*, `https://github.com/apple/swift/blob/main/docs/DifferentiableProgramming.md` (visited on 12/29/2020) (cited on page 38).

[46] George Cybenko, "Approximation by superpositions of a sigmoidal function", Mathematics of control, signals and systems **2**, 303–314 (1989) (cited on page 39).

[47] Kurt Hornik, Maxwell Stinchcombe, and Halbert White, "Multilayer feedforward networks are universal approximators.", Neural networks **2**, 359–366 (1989) (cited on page 39).

[48] Neural Networks and Deep Learning, *A visual proof that neural nets can compute any function*, http://neuralnetworksanddeeplearning.com/chap4.html (visited on 12/29/2020) (cited on page 39).

[49] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang, "The expressive power of neural networks: a view from the width", Advances in neural information processing systems **30**, 6231–6239 (2017) (cited on page 39).

[50] Vincent Dumoulin and Francesco Visin, "A guide to convolution arithmetic for deep learning", arXiv preprint arXiv:1603.07285 (2016) (cited on page 40).

[51] Shaojie Bai, J Zico Kolter, and Vladlen Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling", arXiv preprint arXiv:1803.01271 (2018) (cited on pages 41–42).

[52] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu, "Wavenet: a generative model for raw audio", arXiv preprint arXiv:1609.03499 (2016) (cited on page 41).

[53] Fisher Yu and Vladlen Koltun, "Multi-scale context aggregation by dilated convolutions", arXiv preprint arXiv:1511.07122 (2015) (cited on page 41).

[54] *Fully convolutional networks for semantic segmentation*, Vol. Proceedings of the IEEE conference on computer vision and pattern recognition (2015), pages 3431–3440 (cited on page 41).

[55] *Deep residual learning for image recognition*, Vol. Proceedings of the IEEE conference on computer vision and pattern recognition (2016), pages 770–778 (cited on page 41).

[56] Tim Salimans and Durk P Kingma, "Weight normalization: a simple reparameterization to accelerate training of deep neural networks", Advances in neural information processing systems **29**, 901–909 (2016) (cited on page 42).

[57] Sergey Ioffe and Christian Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift", arXiv preprint arXiv:1502.03167 (2015) (cited on page 42).

[58] Jason Brownlee, *A gentle introduction to dropout for regularizing deep neural networks*, https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/ (visited on 12/29/2020) (cited on page 43).

[59] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein, "Visualizing the loss landscape of neural nets", Advances in neural information processing systems **31**, 6389–6399 (2018) (cited on page 43).

[60] Bjørn H. Mølvig, "Low-energy neutrino reconstructions using recurrent neural networks", Master's thesis (University of Copenhagen, Copenhagen, 2020) (cited on pages 44, 46).

[61] scikit-learn, *Sklearn.preprocessing.robustscaler*, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html (visited on 12/29/2020) (cited on page 44).

[62] Stefan Weinzierl, "Introduction to monte carlo methods", arXiv preprint hep-ph/0006269 (2000) (cited on page 45).

[63] Wikimedia Commons, *File:overfitting.svg*, https://commons.wikimedia.org/wiki/File:Overfitting.svg (visited on 12/29/2020) (cited on page 56).

[64] Bradley Efron, "Better bootstrap confidence intervals", Journal of the American statistical Association **82**, 171–185 (1987) (cited on page 60).

[65] *Cyclical learning rates for training neural networks*, Vol. 2017 IEEE Winter Conference on Applications of Computer Vision (WACV) (IEEE, 2017), pages 464–472 (cited on page 61).

[66] Jeremy Jordan, *Setting the learning rate of your neural network.* `https://www.jeremyjordan.me/nn-learning-rate/` (visited on 12/29/2020) (cited on page 61).