Detecting Neutrinos in Deep Ice

Ethan Newell,* Dylan Perez,† Frank Drugge,‡ and Grace Zhu§
University of California San Diego
(Group 12)
(Dated: November 12, 2024)

The IceCube Observatory uses 5160 sensors to detect Cherenkov radiation from neutrinos in deep Antarctic ice. It aims to determine the origin of neutrinos by calculating incoming angles (azimuth and zenith). The extensive data detected by the IceCube needs to find a way to balance the computational efficiency and accuracy, which created the Kaggle competition, focusing on utilizing machine learning models to classify these angles, thereby avoiding computationally intensive analytical solutions.

The datasets correlate data in different tables with sensor ID and event ID. And they included information such as true azimuth and zenith angles, x, y, and z coordinates of each sensor, pulse times, and auxiliary data indicating pulse quality. To process these huge amounts of data, connect x, y, and z coordinates with each sensor for each event. In the model, each event was represented by a column vector, with boolean values converted to binary. Also, the model includes the detector's geometry layer, which corresponds to the sensor data (time and charge).

Applying the LSTM model will be a good choice to improve efficiency and reduce the processing time generated by dummy data. Still, by comparing the efficiency of the loss function, angular loss function $\omega = \sin(Z_{true}) \times \sin(Z_{pred}) \times (\cos(Az_{true}) \times \cos(Az_{pred}) + \sin(Az_{true}) \times \sin(Az_{pred})) + \cos(Z_{true}) \times \cos(Z_{pred})$ is better than MSE loss.

I. INTRODUCTION

The IceCube Observatory uses the unique properties of the Antarctic ice as a natural detection medium and instrumented thousands of Digital Optical Modules deep in the ground, observing the neutrino's Cherenkov radiation left in the deep ice. The main goal of this experiment is to detect the origin of these neutrinos, mostly by determining the angle that these neutrinos come in relative to normal.

Neutrinos are neutral particles without any charge, so they rarely interact with any other matter and leave any trace to track. When a high-energy neutrino collides with an ice molecule, the neutrino particle will produce a corresponding charged particle: electron, μ, or τ[1]. The particle μ will produce long signals throughout the detector, and the other two particles e^- and τ create more spherical, cascade-like events. Depending on the charged particles' energy, the speed they travel through the ice may be over the speed of light, and emitting Cherenkov radiation, which can be detected by the photomultiplier tubes housed within the DOMs.

The DOMs located at the bottom of the neutrino detector: IceCube detector, from about 1450 meters to 2450 meters. (Shown in FIG 1) Because the ice deep inside the ice layer is much more stable than on the surface and has better purity, the sensor will make it easier to detect the trace of neutrinos without unnecessary noises affecting the data collection. The optical sensors are enclosed in glass spheres, which are arranged in a grid-like pattern. The signals obtained by the sensor will be digitized and then transmitted to the surface. [2]

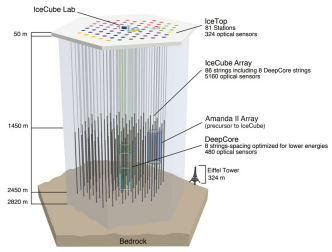

FIG. 1. Three dimensional layout of the neutrino detector [2]

These signals are created by the three charged particles. If the neutrino scatters off the ice, then the signal will contain nothing. The detector IceCube Observatory uses is extremely sensitive to those high-energy neutrinos, which have the range from 10^7 to 10^{21} eV [1], and the particle created would mostly be μ.

Other than the deep ice detector, IceCube can also detect high-energy neutrinos coming from far away in the universe. These detectors detect around 275 million cosmic rays every day, but only a tiny amount of these detected rays are neutrinos. It detects 275 atmospheric neutrinos and about 100,000 per year.[3]

* enewell@ucsd.edu
† Dbperez@ucsd.edu
‡ fdrugge@ucsd.edu
§ yuz236@ucsd.edu

II. DATASET

The dataset is from the Kaggle competition. [4] The total of 663 datasets can mostly be separated into three parts:
`[train/test]_meta.parquet`: As the metadata files for training and testing. It contains variables: `batch_id, event_id, first/last_pulse index, and true azimuth and true zenith` It is noticeable that the variable `[azimuth/zenith]` are the angles in radians of the neutrino. And they have different ranges: azimuth ranges from $[0, 2\pi]$, and for zenith, it is ranges from $[0, \pi]$.

	batch_id	event_id	first_pulse_index	last_pulse_index	azimuth	zenith
1800000	10	29296372	0	43	3.517532	1.616892
1800001	10	29296374	44	132	5.775634	0.199164
1800002	10	29296414	133	180	3.715111	0.769641
1800003	10	29296416	181	241	2.613874	1.399564
1800004	10	29296437	242	310	4.110431	1.054067

FIG. 2. Train_meta.parquet [4]

`[train/test]/batch_[n].parquet`: There are 652 batches, which contain tens of thousands of events with thousands of pulses. It has the variables:`event_id, sensor_id, time, charge, and auxiliary`. The variable of time is the time of the pulse in nanoseconds within the event time window. The variable charge is an estimate of the light in the pulse. It is important for the variable auxiliary that the boolean value shows whether the pulse is fully digitized or not. If $auxiliary == True$, there was more likely to originate from noise, which is not fully digitized. And $auxiliary == False$, the pulse is contributed to the trigger decision and fully digitized.

event_id	sensor_id	time	charge	auxiliary
24	3918	5928	1.325	True
24	4157	6115	1.175	True
24	3520	6492	0.925	True
24	5041	6665	0.225	True
24	2948	8054	1.575	True

FIG. 3. train_batch_1.parquet [4]

`sensor_geometry.csv`: which provides each sensor's position by the three-dimension (x, y, z)

	sensor_id	x	y	z
0	0	-256.14	-521.08	496.03
1	1	-256.14	-521.08	479.01
2	2	-256.14	-521.08	461.99
3	3	-256.14	-521.08	444.97
4	4	-256.14	-521.08	427.95

FIG. 4. sensor_geometry.csv [4]

From the model used to train the data and the preprocessing of the data, combining the [train/test]/batch_[n].parquet and sensor_geometry.csv by the unique sensor_ID would be the first step. Our thoughts are inspired by the sample data given in Kaggle. It relates the time that the events happens, and also shows how may the energy effect the prediction. That they uses the dots to represent each event, the different colors are showing the time spend in each event, and the size of those dots are gonna be the charge of all pulses. It clearly shows the features of IceCube events.

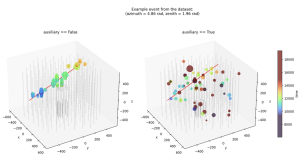

FIG. 5. sample graph prediction[4]

III. METHODS

The primary challenge for our group presented itself as preparing the dataset and conceptually understanding the problem to solve. Given the enormity of the dataset and the limited computing capacity we wanted to create two models. One utilizing a fully connected neural network where every neuron and its properties were represented in the first layer, with two more linear layers behind it of much smaller size, 64 neurons per hidden layer. Each layer utilizing ReLU activation function. We were unable to get this model to work.
In order for us to pass each event to the model, we had to isolate individual events from each batch and turn it into a one dimensional vector. We had various approaches for how we prepared this. Our data from each event was of varying lengths so the first task was to ensure that the length was consistent so that we could define our input size. Our first approach was to insert the data into a vector of zeros, with the full length 5160×7 where each datapoint would be places into corresponding to its sensor id. The second approach consisted of truncating the data per event to a number of sensors that would always be valid. We took the 20 most active sensors and used this array as what we attempted to train our linear model on. There is still some issue with the methods that we used to pre-process our data when we ran our model, from one epoch to another, it was unable to track the gradient descent it was making and backpropogate correctly. We have narrowed this down to our preprocessing, where we believe it is some of the other python packages, like NumPy, which is throwing PyTorch from tracking the gradient descent.
Some engineering done by our group was since we are working in angular space for our output, not cartesian, we didn't want to use a simple mean squared loss function. We defined our angular distance as

Scalar Product: ω

$$\omega = [sin(Z_{true}) * sin(Z_{pred})] \times [cos(\mathrm{Az}_{true}) * cos(\mathrm{Az}_{pred})]$$

$$+[sin(\mathrm{Az}_{true}) * sin(\mathrm{Az}_{pred})] + [cos(Z_{true}) * cos(Z_{pred})]$$

$$\Rightarrow \text{Angular distance: } \Omega = cos^{-1}(\omega)$$

This could have been the point at which our model was losing the ability to track the gradient descent, our group is still working on the issues with the fully connected neural net model. One feature we engineered was the 'center of charge' which would be the x,y,z coordinate of an analogy to center of mass. This would tell our model a good 'initial guess', we repeated this same feature for data with the `auxiliary = False` and for the entire event. Unfortunately since our fully connected NN model was unable to work, we have been unable to use this feature yet. Our weighted sum has a simple definition:

$$\frac{\Sigma[x_i \times \mathrm{Charge}_i]}{\Sigma \mathrm{Charge}_i}$$

We then asked our friend Christian Amezcua to help us come up with a method for generating our own dummy data so that we were able to use only what we needed to prototype our LSTM Model. Our LSTM model has one LSTM layer, followed by two linear layers with 64 inputs to 32 outputs and then 32 inputs to our two outputs. These layers used a ReLU activation function as well. Even though the dummy data is still large, the amount of time it takes to prepare sets for use by the model is significantly shorter. We were able to have this LSTM model make small improvements over 50 epochs. After we were able to train on synthetic data, we were getting to the point that we were going to be able to train on our real data. We decided to only extract one batch of data from the IceCube data set, and then preprocess all of that particular data. While doing this, we realized that we were going to need a lot more time, as just the preprocessing step was going to take roughly an hour. At this point is when we decided that we were going to have to just double our dummy dataset in order to try to get significant increase in our model performance.

Here is the link to our repository. [5].

```
class NeutrinoModel(nn.Module):
    def __init__(self):
        super(NeutrinoModel, self).__init__()
        self.lstm = nn.LSTM(input_size=4, hidden_size=64, num_layers=2, batch_first=Tru
        self.fc1 = nn.Linear(64, 32)
        self.fc2 = nn.Linear(32, 2)  # Predicting azimuth and zenith

    def forward(self, x):
        h_lstm, _ = self.lstm(x)
        h_lstm = h_lstm[:, -1, :]  # Take the last output of the LSTM
        x = torch.relu(self.fc1(h_lstm))
        x = self.fc2(x)
        return x
```

FIG. 6. Model Summary

IV. RESULTS

Due to the sheer size of the IceCube data, we weren't able to successfully train our model on the real data. Instead, we had to produce our own synthetic data in order to reduce the size down to something that was manageable, but still had all of the same characteristics as the true observatory data. By doing this, we were able to successfully train the model, but the results weren't as great as we were expecting. On the early runs of the model, we used 10 batches, each with 100 events in them. Although this dataset was much smaller than the real data, it still contained enough data points to sufficiently train the model. The training loss of the early runs always converged at ≈ 2, which is high when considering the loss of the model.

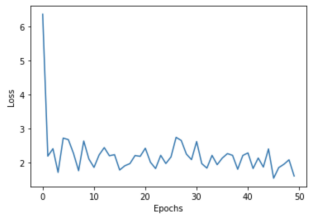

FIG. 7. Training Loss

The validation loss for the early runs of the model also converged at around the same value, which indicated that the model stopped learning too early. We continued to play with the parameters of our model, changing batch size, number of epochs, etc. This didn't seem to lower our loss, so we opted to attempt to double the data size. Instead of 10 batches, each with 100 events, we increased to 10 batches, with 200 events each. Although the model had more data to train on, the model actually ended up performing worse than our smaller data set model.

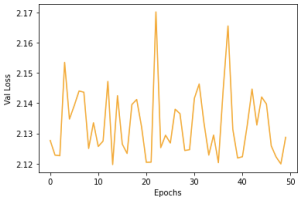

FIG. 8. Validation Loss for bigger data

This stumped us, as it seems obvious that more data to train on would increase the robustness of our model, but it seemed to do the opposite. We think that the results we are getting have to do with the fact that the hyperparameters for our model are not tuned correctly for this model. Over the summer, the group plans on meeting to continue to improve our model, for our own sake. The results aren't exactly what we were expecting, but there are definite improvements to be made. When comparing our results to the results from the Kaggle competition, our results were much different. This is most likely due to the fact that the Kaggle competition used much more complex models. Given the time constraint and computational resources, it was difficult to employ the models given by the competition. Our results are significant be-

cause they show that, although the model isn't performing well, it is still able to train on the data. One of the hardest parts of this project was being able to get the data to a form that was suitable to input into the model, and to train it on. We were able to solve that problem, so now it seems that we are potentially using the wrong machine learning models to train. Our model consisted of an LSTM layer, followed by a few linear layers. This seemed like the best method off the bat but has proved to not be as robust on larger sets of data, which this happens to be.

V. CONCLUSION

In this project, we took data from the Ice Cube Kaggle competition and attempted to reproduce results of a well-performing model in the competition. This dataset was formed through the signal of neutrino events leaving charges on detectors arranged periodically inside of a "cube" of ice(hence the name). This data consists of millions of events which we struggled to deal with given our computational power. By creating some synthetic data, we were able to approach a model that would be able to train on a larger dataset, given appropriate compute time and power. In this project, we learned the importance of setting up clear directions and planning for issues in development. This facet of the project was most apparent when dealing with data. Finding ways to make data clean and usable in order to input to an appropriate model absorbed a lot more time than it should have, and this is due to not having a solid plan for correct format, or getting it to work with a model. In hindsight, we should have started with a model and synthetic data, like we were able to do nearer to the end of the project, which is a pattern that we will internalize as people who will likely work with these kinds of models in the future. This project gave us challenges, successes, failures, and most importantly experiential lessons, which will serve us all in the future. For our future plans, we are going to be revisiting this model over the summer, in an attempt to train the model on real IceCube data. We were getting close to being able to do this, but the computational resources at our disposal made this difficult. After we are able to get our data into the proper format for the model, we are going to continue to make improvements on it. It has become clear through our testing that the bigger the data size is for this model, the worse it performs. This will lead to us possibly changing the layers in our model, so that we are able to then get a loss that we are happy with. When we compare our latest model to our baseline, it performs worse, so there is an inherit problem with the model we currently have. To get our model to improve, we will be consulting the Kaggle competition models that did well, and seeing what we can adopt from those models. Also, we need to figure out a more efficient way to preprocess all of our data, considering we want to train this model on the real data. This is also going to prove to be a challenge we face, but will continue to make improvements on.

ACKNOWLEDGMENTS

Thank you to Christian Amezcua for helping us with generating dummy data and for assisting in the LSTM model as well. Thank you as well to Professor Duarte and Li for all of the help over the course of the quarter.

[1] W. contributors, Icecube neutrino observatory — wikipedia, the free encyclopedia.
[2] I. N. Observatory, Icecube science.
[3] Q. U. Gazette, A frozen lab: Detecting neutrinos in the ice, accessed: 2024-06-13.
[4] Kaggle, Icecube - neutrinos in deep ice (2023), accessed: 2024-06-13.
[5] Github:neutrinos in deep ice (2024).