Exercise 4: Graph Neural Network

Erik Stolt

What you did and how

We were tasked to implement a graph neural network (GNN) in order to predict the positions of neutrino interaction from the measured Cherenkov light. This involves working with data in the form of graphs with connections. In our case, each event contains a collection of detected photons (also called hits or pulses). Every photon is characterized by the time t it was detected, along with the x and y coordinates of the detector module that registered it. I started of by normalizing the data according to the mean and std of the training data, then saving these parameters for denormalization in the analysis later. I then implemented the GNN by using the *DynamicEdgeConv* function calculates the k-nearest neighbors from the feature space. Each pair is combined into a single array of size twice the feature size in order to perform Edge Convolution (EdgeConv). See appendix FYI. In the end, we train the network to adjust the internal weights of the GNN (especially inside the EdgeConv MLP's) so that, for any new event, the model can accurately predict the interaction position.

What results you obtained

I tried around with a lot of different hyperparameters and different network architectures. In the end I found that a good network for the MLP was four layers with batch normalization and three layers, all with 64 hidden nodes, for the GNN encoder. I also settled on trainng the model for 20 epochs (more did not give better results) and using batch size 32 and an initial learning rate of 5×10^{-5}. Furthermore, the optimal k seemed to be around 15. An example training print-out looked like this

```
Epoch 15/20 | Train Loss: 0.6803 | Val Loss: 0.7070 | lr: 0.0005 | ES: 1 | Time: 64.62s
Epoch 16/20 | Train Loss: 0.6850 | Val Loss: 0.7116 | lr: 0.0005 | ES: 2 | Time: 70.18s
Epoch 17/20 | Train Loss: 0.6774 | Val Loss: 0.6829 | lr: 0.0005 | ES: 3 | Time: 71.25s
```

where `ES` stands for the number of epochs where the validation loss did not improve with a tolerance of 0.05. And some plots:

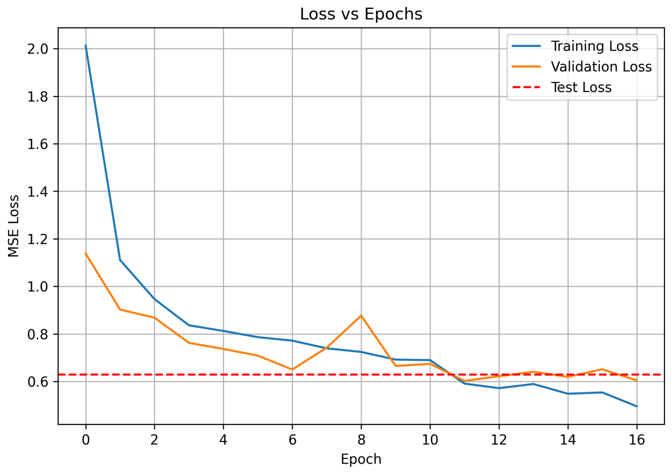

Figure 1: Training and validation loss as a function of the epoch. The dashed line is the test loss for the model.

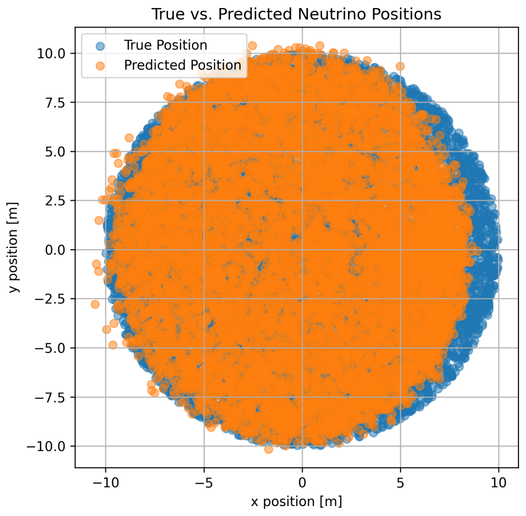

Figure 2: Scatter plot showing the predictions and true positions.

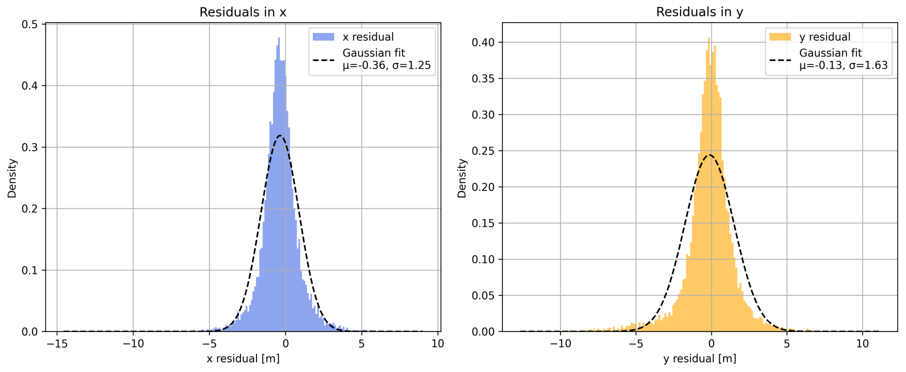

Figure 3: Normalized residual plot for the x positions (left) and the y positions (right).

Notice that Fig. 2 does not actually say a lot since we do not know how far away a prediction is from the true position. It only tells us that our predictions seems reasonable since they are located inside (mostly) the detector region. This is why we need Fig. 3 which shows the residual for both the x and y position, i.e., Residual $= \frac{\text{Predicted}_i - \text{True}_i}{\sigma_i}$. This should ideally follow a Gaussian $\mathcal{N}(0, 1)$ with correct uncertainty estimates and no systematic bias, see Appendix 2. This decently follows a gaussian but the model is overconfident since it has a distinct peak with low std.

What challenges you encountered and what could be improved

One of the main problems I encountered was that the predictions only fell into the first quadrant i.e., $x, y > 0$. This was due to me applying ReLu on the output layer, therefore clipping all negative values. This was easily resolved by removing the activation function in the output layer.

One other "problem" I had was to get familiar with the awkward arrays. This had a little learning curve but I got it in the end.

Possible improvements is as usual in ML problems, to find the right network architecture. One can for example try random grid search to find suitable hyperparameters. In the end though, I was able to accurately predict the position of the incoming neutrino interactions from the measured Cherenkov light. I also tried adding dropout without any improvements.

Link to GitHub

Link:

Appendix 1

Here,

$$\begin{aligned}\text{Nodes} &= \text{detector modules in IceCube},\\ \text{Edges} &= \text{connections indicating relationships between modules}.\end{aligned}$$

That is, Each neutrino interaction event is one graph with many nodes. However, we have two types of node features. We have initial features (measurement data) and hidden (learned) nodes. These do not have the same dimension, but we specify the number of hidden nodes in the code e.g. 64. We simply build a graph by connecting each node (undirected) to its k-nearest-neighbors. The procedure of calculating its connection, between two neighbors, can be described as

$$e_{ij} = h_\theta(x_i, x_i - x_j),$$

where h_θ is implemented as a small neural network e.g. MLP. Furthermore, the feature difference captures the locality of the neighboring nodes. So, for a node i we calculate all edges e_{ij} then we apply aggregation which combines all these edges (via sum, mean or max etc) into a new hidden node value given by

$$x_i' = \text{AGGREGATE}(e_{ij})_{j=1}^{k}.$$

This is the output of the first GNN layer. GNNs then do the same procedure for all nodes which creates a new graph, from which the procedure repeats.

Appendix 2

Assume our dataset is generated according to the process

$$y = f(\boldsymbol{x}) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2),$$

where $\boldsymbol{x}$ are the input features, $f(\boldsymbol{x})$ is the true underlying function, and ϵ is zero-mean Gaussian noise with fixed variance σ^2. Let $\hat{y} = \hat{f}(\boldsymbol{x})$ be the model's prediction of the mean. If the model approximates the true function well, i.e., $\hat{f}(\boldsymbol{x}) \approx f(\boldsymbol{x})$, then the residual

$$r = y - \hat{y} = f(\boldsymbol{x}) + \epsilon - \hat{f}(\boldsymbol{x}) \approx \epsilon,$$

is approximately distributed as the original noise:

$$r \sim \mathcal{N}(0, \sigma^2).$$

If the model also provides an estimate of the uncertainty $\hat{\sigma}(\boldsymbol{x}) \approx \sigma$ (where σ is the std of the predictions), we can define the normalized residual as

$$z = \frac{y - \hat{y}}{\hat{\sigma}(\boldsymbol{x})} \approx \frac{y - \hat{y}}{\sigma}.$$

Under the assumption that both the predicted mean and uncertainty are accurate, the normalized residuals should follow a standard normal distribution:

$$z \sim \mathcal{N}(0, 1).$$

This forms the basis for using the distribution of normalized residuals as a diagnostic tool for assessing the calibration of both the predictive mean and the uncertainty.

Code (for backup)