Sparse 3D Heteroscedastic Convnet for Directionality

Majd Ghrear

Introduction

Modern high-definition gas TPCs can image low-energy electron recoils in 3-Dimensions, an example from the BEAST TPCs [1] is shown in Figure 1. These 3D images enable us to determine the initial direction of the electron recoils which has applications to

- solar neutrino physics, where the electron's directionality can be used to reconstruct the solar neutrino spectrum [2]
- non-wimp dark matter searches, where the electron's directionality can be used to distinguish a dark matter signal from backgrounds [2]
- X-ray polarimetry, where the electrons directionality can be used the determine the polarization of x-rays emitted by astrophysical objects [3].

We train a neural network to determine the initial direction of electron recoils and their uncertainties. The input is a 3D electron recoil track and the output is a 3D direction with uncertainty. Although this discussion is in the context of electron recoils, the techniques we implement are widely applicable to deep learning problems involving 3D directionality.

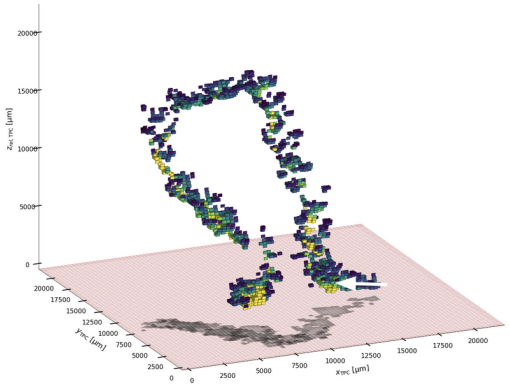

Figure 1 40 keV electron recoil track imaged with the BEAST TPCs [1]. The initial direction of the electron recoil is indicated by the white arrow.

To create our dataset, the DEGRAD framework [4] is used to simulate 2E5 electron recoil tracks at 40, 45, and 50 keV. The electrons are simulated in a 70% He 30% CO2 gas mixture at 760 torr. All DEGRAD recoils begin at the origin and are oriented in the +z-direction.

We train two convolutional neural networks (CNNs) on our simulation data. One is homoscedastic (only predicts the direction) and the other is heteroscedastic (predicts direction and uncertainty). We use CNNs because of their ability to learn small objects within an image (such as the head, tail, and curves in the recoil) and because convolutional layers require fewer

weights which helps avoid overfitting and allows for faster computation. The heteroscedastic model is selected because the uncertainty output is important for subsequent physics analyses.

Preprocessing and data splits

The degrad simulations require substantial preprocessing before they are suitable. The pre-processing steps taken for each recoil are:

1. Gaussian smearing is applied to each point in a recoil to take into account diffusion. The smearing applied is drawn uniformly from 160-466 microns.
2. The recoil is rotated to a random direction drawn from an isotropic distribution. The random direction is stored as the label (the true direction).
3. The recoil is mean-centered.

The next step is to bin the recoils track into a 120x120x120 grid with (500um)^3 resolution. In this format, a single event contains 1728000 features. However, the data is sparse - a small minority of the features are non-zero. Instead of binning the recoil tracks, which forces us to allocate memory to the zero entries, we determine the indices with non-zero entries and their corresponding values. These indices and values are then stored as a PyTorch sparse tensor in the COO format. Recoil tracks that extend beyond the specified grid are discarded. An illustration of the various stages of data processing is provided in Figure 2.

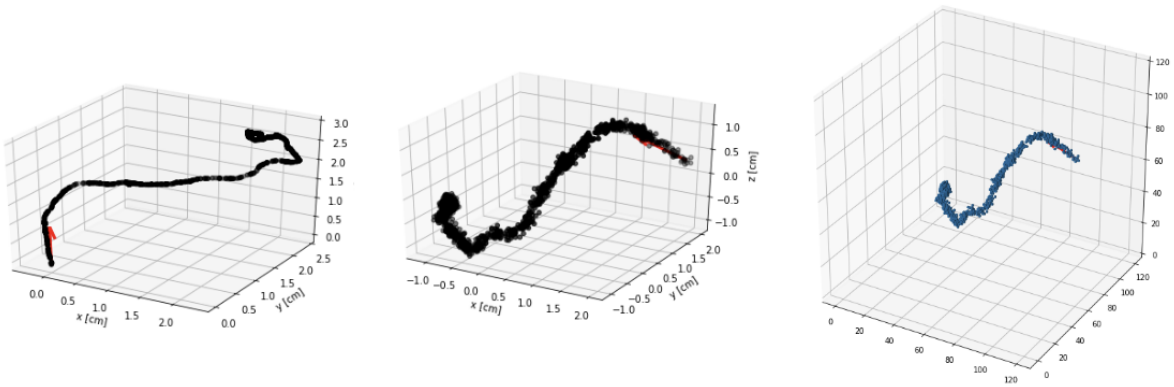

Figure 2 Left: an electron recoil as simulated by DEGRAD. Center: the DEGRAD simulation after it has been diffused, rotated, and mean-centered. Right: the processed simulation is then binned into a 120x120x120 grid with (500um)^3 resolution.

The final dataset contains 553712 simulations. The dataset is randomly split into two subsets: 80% training and 20% validation. The models were trained on the training set and the validation set was used for early stopping. The independent test set, used to evaluate the models, is discussed in the “Evaluating the Models” section.

Architecture and Hyperparameters

A data loader is used to load batches of Pytorch sparse tensors and pass on their indices and values to the CNNs. The batch size is a hyperparameter that is set as 64. The CNNs first convert the provided information into a spconv.SparseConvTensor [5,6]. Spconv provides heavily-optimized sparse convolution and submanifold sparse convolution implementations, previously discussed in Refs. [7,8]. Sparse convolution allows us to convolve and downsample our input data without expressing them as dense tensors. This significantly improves the speed and reduces the memory usage of our models. Submanifold sparse convolution is a slightly modified sparse convolution algorithm that preserves sparseness, allowing for deeper networks. These sparse convolution topics are discussed in Refs. [5,6,7,8,9,10] and they are essential for deep learning with highly-segmented 3D data.

Both heteroscedastic and homoscedastic models begin with the same spconv sequential block, illustrated in Figure 3. This block passes the input spconv.SparseConvTensor through several convolutional, relU, and maxpooling layers and then converts the data into a regular PyTorch tensor of shape (20 x 12 x 12 x 12). Afterward, the tensor is flattened to a column of length 34560. The architecture following the block in Figure 3 differs between the two models. For the homoscedastic model, a fully-connected architecture processes the flattened input into 3 outputs, which are interpreted as the cartesian coordinates of a 3D unit vector. For the heteroscedastic model, a dual-headed dense architecture is used to process the flattened input into a 3D unit vector along one head and an associated uncertainty along the other. The full details of the architectures are presented in Figure 4.

```
  (net): SparseSequential(
    (0): SparseConv3d(1, 50, kernel_size=[6, 6, 6], stride=[2, 2, 2], padding=[0, 0, 0], dilation=[1, 1, 1], output
_padding=[0, 0, 0], algo=ConvAlgo.Native)
    (1): ReLU()
    (2): SparseMaxPool3d(kernel_size=[2, 2, 2], stride=[2, 2, 2], padding=[0, 0, 0], dilation=[1, 1, 1], algo=ConvA
lgo.MaskImplicitGemm)
    (3): SparseConv3d(50, 30, kernel_size=[4, 4, 4], stride=[1, 1, 1], padding=[0, 0, 0], dilation=[1, 1, 1], outpu
t_padding=[0, 0, 0], algo=ConvAlgo.Native)
    (4): ReLU()
    (5): SparseConv3d(30, 20, kernel_size=[3, 3, 3], stride=[1, 1, 1], padding=[0, 0, 0], dilation=[1, 1, 1], outpu
t_padding=[0, 0, 0], algo=ConvAlgo.MaskImplicitGemm)
    (6): ReLU()
    (7): SparseMaxPool3d(kernel_size=[2, 2, 2], stride=[2, 2, 2], padding=[0, 0, 0], dilation=[1, 1, 1], algo=ConvA
lgo.MaskImplicitGemm)
    (8): ToDense()
```

Figure 3 Sparse convolution and maxpooling layers implemented with spconv. The relevant parameters are detailed in the figure. Both heteroscedastic and homoscedastic models begin with this architecture. The input is a 1x120x120x120 spconv.SparseConvTensor and the output is a regular 20 x 12 x 12 x 12 PyTorch tensor.

The optimizer used is Adam, with hyperparameters: 0.0001, 0.94, 0.999, and 1E-7 for the learning rate, beta1, beta2, and epsilon, respectively. The hyperparameters were not optimized. A combination of hyperparameters that allowed the models to train with a reasonable speed was found and the same parameters were used for both models. Rigorous hyperparameter optimization is left for future work. Early stopping is implemented to stop training if the validation loss has not decreased for the last 2 epochs. For both models, the weights which resulted in the lowest validation loss are saved, as shown in Figure 5.

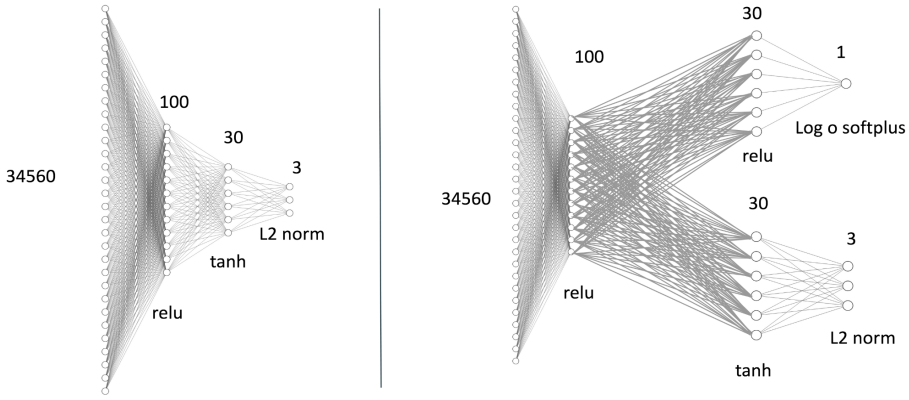

Figure 4 The neural network layers following the spconv sparse convolution block, shown in Figure 3. Left: for the homoscedastic model, a fully-connected architecture is used. Right: for the heteroscedastic model, a dual-head architecture is used.

The Loss Function(s)

The homoscedastic model uses the cosine similarity loss which is expressed as

$$\text{CosineSimilairityLoss } = \frac{-1}{N}\sum_{i=1}^{N}\frac{\mathbf{y}_i \cdot \hat{\mathbf{y}}_i}{\max(|\mathbf{y}_i||\hat{\mathbf{y}}_i|, \varepsilon)} \tag{1}$$

Above, the sum is over the batch, $\mathbf{y}_i$ is the label, $\hat{\mathbf{y}}_i$ is the prediction, and ε is a small number to avoid dividing by 0. In our case, $\mathbf{y}_i$ and $\hat{\mathbf{y}}_i$ are unit vectors so the denominator equals 1.

For the heteroscedastic model, we must use the negative log-likelihood (NLL) loss which, in our case, requires a specialized probabilistic model. Fortunately, this was developed in 1982 and it's known as The 5-parameter Fisher–Bingham distribution [11]

$$f(\mathbf{x}) = \frac{1}{\mathrm{c}(\kappa,\beta)}\exp\{\kappa\boldsymbol{\gamma}_1^T\cdot\mathbf{x} + \beta[(\boldsymbol{\gamma}_2^T\cdot\mathbf{x})^2 - (\boldsymbol{\gamma}_3^T\cdot\mathbf{x})^2]\}$$

Where the normalization constant is

$$c(\kappa,\beta) = 2\pi\sum_{j=0}^{\infty}\frac{\Gamma(j+\frac{1}{2})}{\Gamma(j+1)}\beta^{2j}\left(\frac{1}{2}\kappa\right)^{-2j-\frac{1}{2}} I_{2j+\frac{1}{2}}(\kappa)$$

Above, κ is an analog of variance, β determines the ellipticity of the distribution, $\boldsymbol{\gamma}_1$ is the mean direction, $\boldsymbol{\gamma}_2$ and $\boldsymbol{\gamma}_3$ are the major and minor axis. Making the simplifying assumption that the uncertainty in a predicted direction is isotropic, we set β=0. This gives us

$$f(\mathbf{x}) = \frac{1}{\mathrm{c}(\kappa,\beta)}\exp\{\kappa\boldsymbol{\gamma}_1^T\cdot\mathbf{x}\}$$

and

$$c(\kappa, 0) = 4\pi(\kappa^{-1})\sinh(\kappa)$$

Now, γ_1 is interpreted as the prediction ($\hat{\mathbf{y}}_i$), and x is interpreted as the label ($\mathbf{y}_i$). The likelihood of a given batch is then

$$\mathcal{L} = \prod_{i=1}^{N} \frac{\kappa_i}{4\pi \sinh(\kappa_i)} e^{\kappa_i(\mathbf{y}_i \cdot \hat{\mathbf{y}}_i)}$$

and the negative log-likelihood (divided by batch size) is

$$\frac{-ln\mathcal{L}}{N} = \frac{-1}{N}\sum_{i=1}^{N} ln(\frac{\kappa_i}{4\pi \sinh(\kappa_i)}) + \kappa_i(\mathbf{y}_i \cdot \hat{\mathbf{y}}_i) \tag{2}$$

Note that, because $\mathbf{y}_i$ and $\hat{\mathbf{y}}_i$ are unit vectors, if we assume κ is constant, then minimizing Eqn. 2 is equivalent to minimizing Eqn. 1. Hence, our models are counterparts (see Ref. [12] for more discussion).

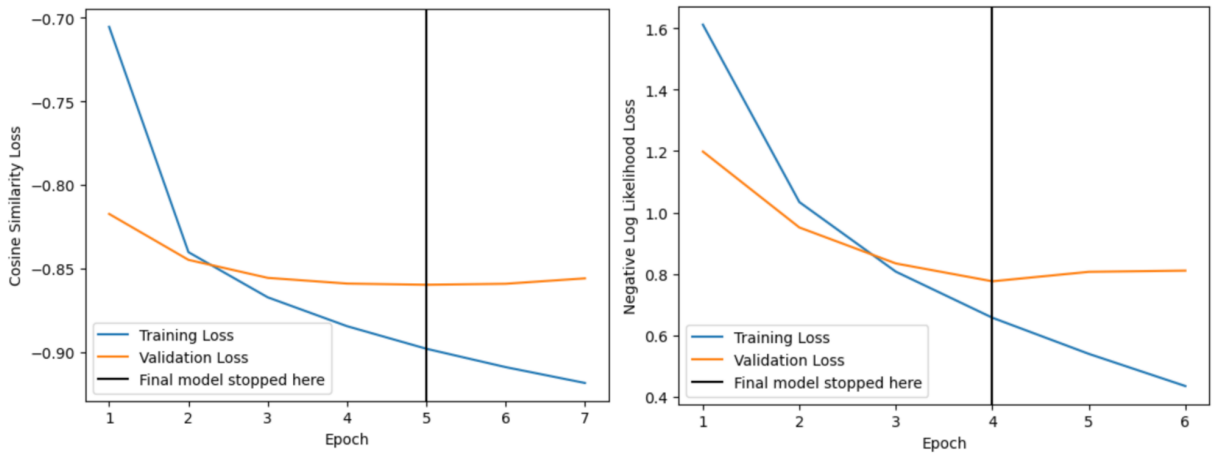

Figure 5 Left: training and validation loss for the homoscedastic model. Right: training and validation losses for the heteroscedastic model. The black vertical lines indicate the final models, with the lowest validation loss.

Evaluating the models

The training and validation losses for both models are displayed in Figure 5. To assess the performance of the models, we simulate a new independent test set of 20k electron recoils at 50 keV and 40 keV. The test simulations are processed exactly as previously described, except the Gaussian smearing applied is fixed at 443 and 200 um for the 50 and 40 keV simulations, respectively.

We assess the performance of the models by comparing them to the non-machine learning (NML) algorithm discuss in Ref. [13] generalized to 3D. Since Ref. [13] does not discuss how to assign a head-tail to the final predicted vector we implement two approaches. In the first, we determine the head/tail using the skewness of the distribution and in the second, we simply assign the correct head/tail to the vector. We use the CosineSimilairtyLoss as a metric by which we compare models since it is an intuitive score (a score of -1 indicates perfect performance and 0 indicates random guesses).

The results are summarized in Figure 6, where the left-hand side shows performance on the 40 keV test set with 200 um smearing and the right side shows performance on the 50 keV test set with 443 um smearing. The y-axis is the CosineSimilairtyLoss and the x-axis shows the percentage of the events that are not used when computing the cosine similarity (efficiency cut). The red diamond shows the performance of the homoscedastic model; since it only outputs directions, we cannot make any efficiency cuts on its predictions. The blue line shows the performance of the heteroscedastic model, we use the uncertainty prediction (κ) to make efficiency cuts on its predictions. The orange circle shows the performance of the NML model, since the algorithm can fail for a subset of the data we are forced to make a single efficiency cut. The downward arrow indicates how much the performance can be improved with perfect head/tail assignment instead of using skewness (i.e. the best possible performance of the NML model). The black star indicates the performance if we do not use the standard parameters in the NML method but rather select the parameters which minimize the CosineSimilairityLoss on the test set via a grid search.

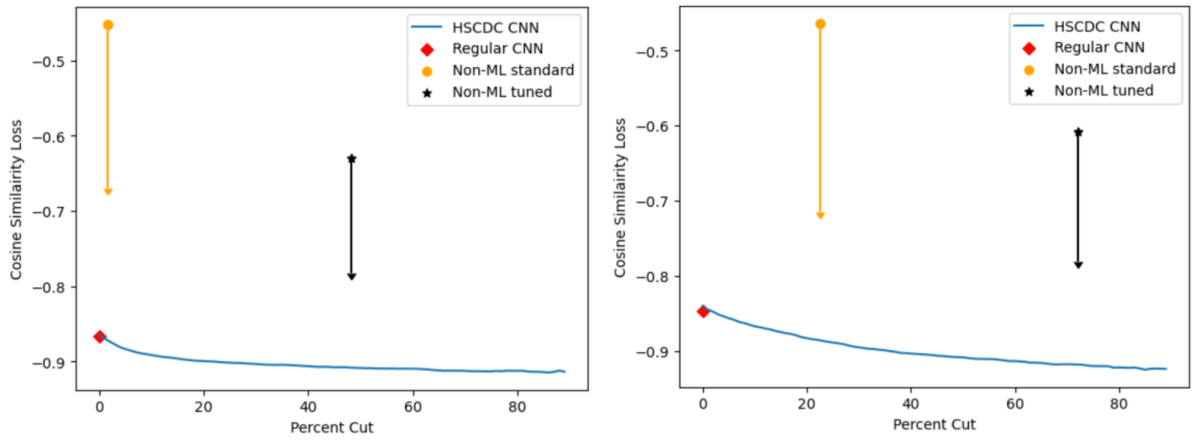

Figure 6 Left: Performance on the 40 keV dataset. Right: Performance on the 50 keV dataset. For a detailed discussion of the quantities displayed, see the paragraph directly above.

We find that in all cases, the deep learning models significantly outperform the non-machine learning models, even under the most optimistic assumptions. Furthermore, we found that the Heteroscedastic model can match the performance of the Homoscedastic model. However, the Heteroscedastic model also provides an uncertainty output that can be used for efficiency cuts, event weighting, and more. Future work could involve improving performance by training on more data, optimizing the hyperparameters, and using submanifold sparse convolution layers to create deeper neural networks. It would also be interesting to see what happens if we use the full 5-parameter Fisher–Bingham distribution without our isotropic simplification.

Disclaimer

This is a continuation of my ICS 635 project. Below, I list the new components in this project

- Pytorch implementation (instead of Keras)
- Deeper architecture
- Significantly improved performance
- The use of sparse convolutional layers
- Heteroscedastic neural regression to determine uncertainty
- New NLL loss function with proper probabilistic treatment
- Comparison to a non-ML method

References

[1] Jaegle, I., et al. "Compact, directional neutron detectors capable of high-resolution nuclear recoil imaging." Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 945 (2019): 162296.

[2] Vahsen, Sven E., Ciaran AJ O'Hare, and Dinesh Loomba. "Directional recoil detection." Annual Review of Nuclear and Particle Science 71 (2021): 189-224.

[3] Weisskopf, Martin C., et al. "The imaging x-ray polarimetry explorer (IXPE)." Results in Physics 6 (2016): 1179-1180.

[4] Stephen Biagi. “Degrad — transport of electrons in gas mixtures.” https://degrad.web.cern.ch/degrad/.

[5] Spconv: Spatially Sparse Convolution Library, https://github.com/traveller59/spconv (2022)

[6] Yan, Yan, Yuxing Mao, and Bo Li. "Second: Sparsely embedded convolutional detection." Sensors 18.10 (2018): 3337.

[7] Graham, Benjamin, Martin Engelcke, and Laurens Van Der Maaten. "3d semantic segmentation with submanifold sparse convolutional networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018.

[8] Graham, Benjamin, and Laurens Van der Maaten. "Submanifold sparse convolutional networks." *arXiv preprint arXiv:1706.01307* (2017).

[9] Tang, Haotian, et al. "TorchSparse: Efficient point cloud inference engine." *Proceedings of Machine Learning and Systems* 4 (2022): 302-315.

[10] Tang, Haotian, et al. "Searching efficient 3d architectures with sparse point-voxel convolution." *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXVIII*. Cham: Springer International Publishing, 2020.

[11] The Fisher-Bingham Distribution on the Sphere Journal of the Royal Statistical Society. Series B (Methodological) Vol. 44, No. 1 (1982), pp. 71-80 (10 pages)

[12] Stirn, Andrew, et al. "Faithful heteroscedastic regression with neural networks." International Conference on Artificial Intelligence and Statistics. PMLR, 2023.

[13] Di Marco, Alessandro, et al. "A weighted analysis to improve the x-ray polarization sensitivity of the imaging x-ray polarimetry explorer." *The Astronomical Journal* 163.4 (2022): 170.