Machine Learning in the MUSE Straw Tube Trackers: Update 1

Kyle Salamone

Center for Frontiers in Nuclear Science, Stony Brook University

February 28, 2025

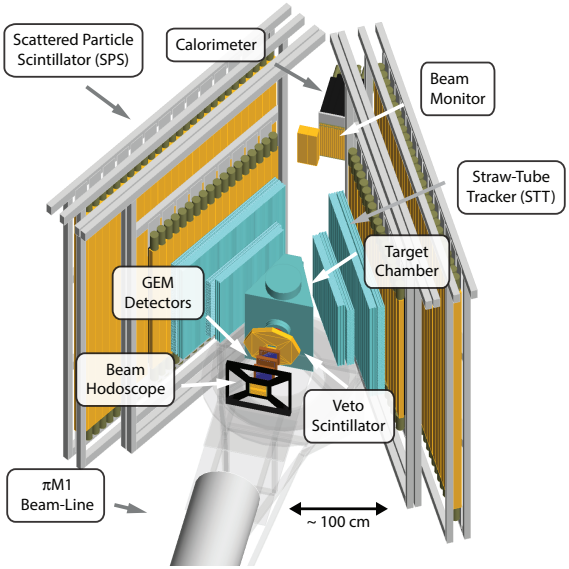
This material is based upon work supported by the National Science Foundation under NSF Grant PHY-2412703. The MUSE experiment is supported by the Department of Energy, NSF, PSI and the US-Israel Binational Science Foundation.

The MUon Scattering Experiment (MUSE)

- 2010: CREMA extract r_p through muonic hydrogen spectroscopy: $\sim 7.9\sigma$ from average ep scattering value at time
- The MUon Scattering Experiment (MUSE) was directly inspired by the proton radius puzzle
- Goals:
  - Precision measurement of r_p via ep and μp scattering
  - Precision study of TPE in ep and μp scattering
  - Direct test of lepton universality
- Housed at the πM1 beamline at the Paul Scherrer Institute

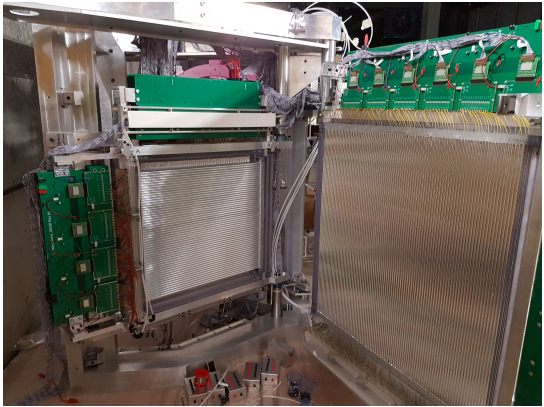
Experimental Setup

- θ acceptance: $20 - 100^\circ$
- $\pi M1$ Beam Line:
  - p$\in 115, 160, 210$ MeV/c
  - Mixed beam of e, μ, π
  - Both polarities of particles!



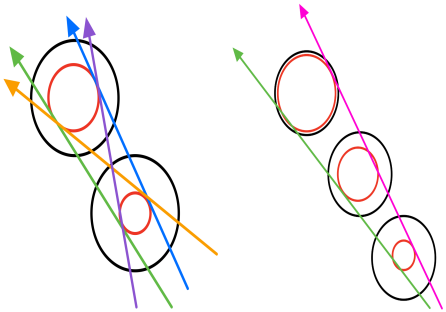
The Straw Tube Trackers (STT)

- Scattered particle tracking detector in MUSE
- Mirrored setup:
  - 20 planes of straws (10 horizontal, 10 vertical)
  - $\sim$ 3000 straws total
  - Smaller front chamber, larger rear chamber
  - 5.1mm straw radius, 60 and 90 cm long



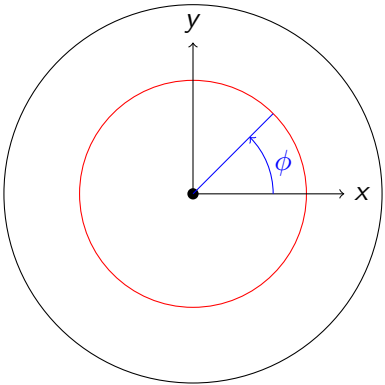
STT Tracking: Left Right Ambiguity

- Tracking is not straightforward: Left Right Ambiguity
- Makes χ^2 distribution in minimization complex
  - Many local minima for minimizer to get stuck in



Machine Learning Approach

- Idea: train a neural network to help resolve this ambiguity
- NN to predict ϕ in local straw frame
  - More accurately: predict $(\sin(\phi), \cos(\phi))$

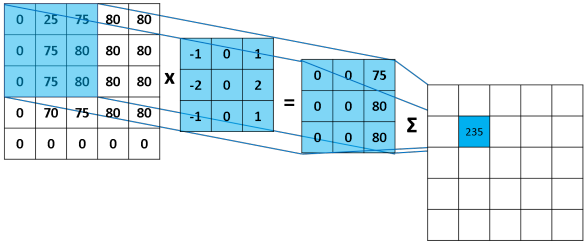


Training Tools

- Convolutional Neural Network structure (CNN)
- CosineAnnealingWarmRestarts LR scheduler
- Adam optimizer
- Loss function: custom wrapper around SmoothL1Loss for masking

Convolutional Layers

- Extract certain features from “image” like input
- Definitions:
  - Channels: specific feature or characteristic of the image
  - Kernel size: how big each “picture” is
  - Padding: pad input dimension with zeros
  - Stride: how many pixels we move the kernel per “step”
- Idea: treat planes of straws as 2D image



https://mlnotebook.github.io/post/CNN1/

NN Structure

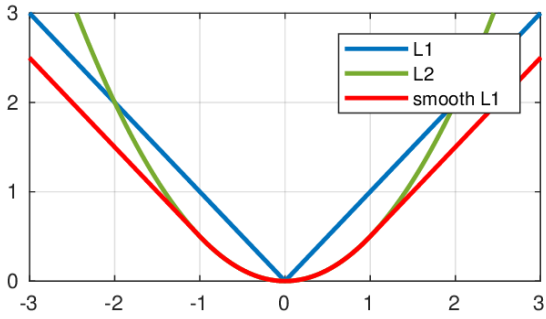
```python
class LeftRightLearner(nn.Module):
    def __init__(self):
        super(LeftRightLearner, self).__init__()
        self.fc = nn.Sequential(
            nn.Conv2d(in_channels=2,out_channels=128, kernel_size=(5,10), stride=(5,5), padding=(0,3)),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Conv2d(in_channels=128, out_channels=256, kernel_size=1),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Flatten(),
            nn.Linear(9216, 1024),
            nn.ReLU(),
            nn.Linear(1024, 2*input_size)
        )
    def forward(self, x):
        return self.fc(x)
```

- Structure of I/O:
  1. 10 arrays of length 89
  2. 2 input channels: binary 0/1, hit radius
  3. Output: $(\sin(\phi), \cos(\phi))$
- Training* time for 70 epochs with CUDA: $\sim$ 16.5 minutes

SmoothL1Loss

- Combines L1 (linear) and L2 (quadratic) loss functions
- β: hyperparameter, defines transition point
- Benefits: less sensitive to outliters than MSE, *can* prevent exploding gradients

$$\text{loss}(x, y) = \begin{cases} 0.5\,(x_n - y_n)^2 / \beta & \text{if} |x_n - y_n| < \beta \\ |x_n - y_n| - 0.5\beta & \text{otherwise} \end{cases}$$

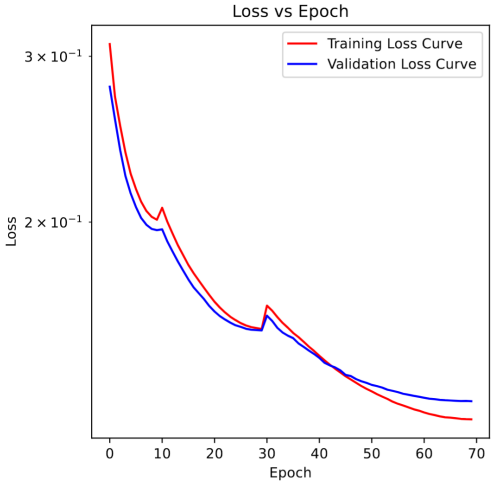


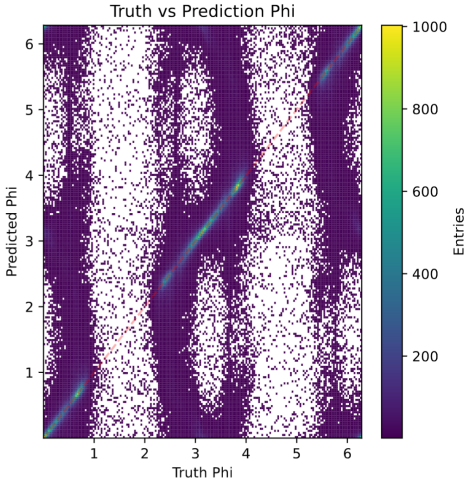
https://www.researchgate.net/figure/Plots-of-the-L1-L2-and-smooth-L1-loss-functions_fig4_321180616

Loss Function

```python
def MaskedTrigLoss(truth, predicted):
    mask = (truth != 0)

    sin_pred, cos_pred = predicted[:, :890], predicted[:,890:]
    sin_pred, cos_pred = sin_pred[mask], cos_pred[mask]
    sin_true, cos_true = torch.sin(truth[mask]), torch.cos(truth[mask])

    sin2_cos2 = torch.square(sin_pred) + torch.square(cos_pred)
    sin2_cos2_true = torch.ones_like(sin2_cos2)

    pred_vec = torch.stack([sin_pred, cos_pred, sin2_cos2], dim=1)
    true_vec = torch.stack([sin_true, cos_true, sin2_cos2_true], dim=1)

    return nn.SmoothL1Loss(beta=0.5)(pred_vec, true_vec)
```

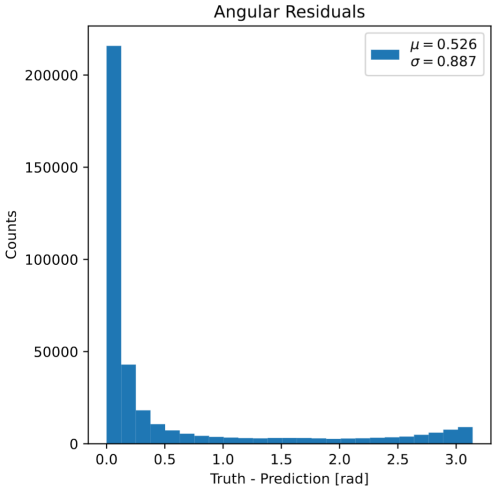
Results

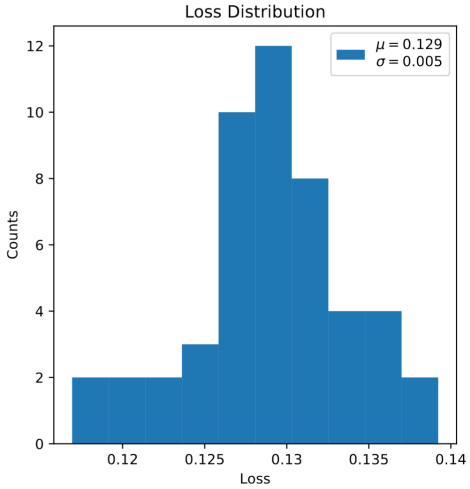




Results are on the validation set, not the training set.

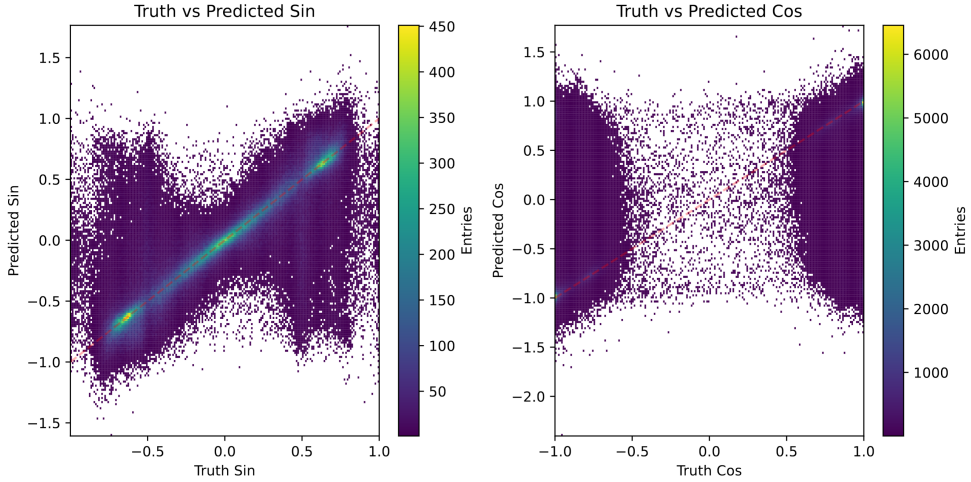
Results





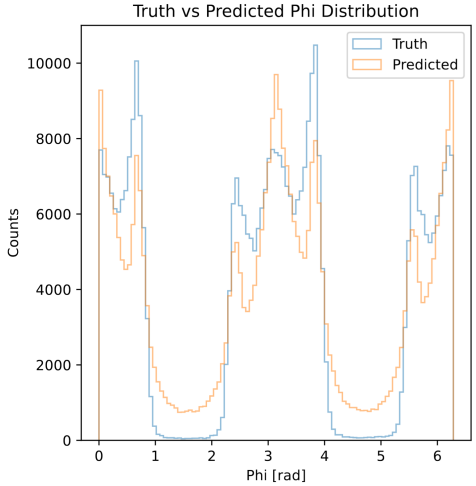
Results are on the validation set, not the training set.

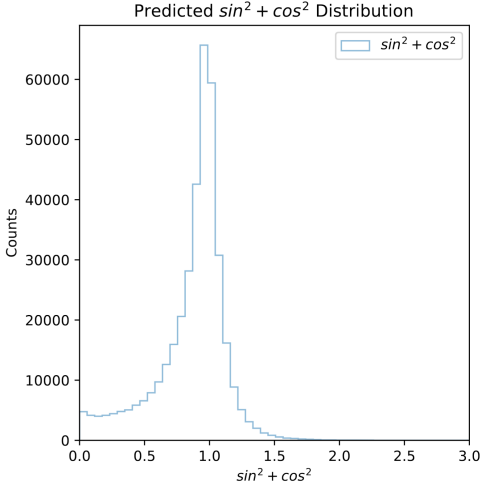
Results



Results are on the validation set, not the training set.

Results





Results are on the validation set, not the training set.

Thoughts

- Results are very promising
- Based on loss curves - not done training!
- Can test on run data, seems to perform okay (far from perfect) - implementation in main code...

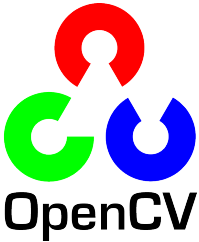
Data Results

```
Testing model...
Model inference took 0.018435376001434634 seconds.
Plane 1, Straw 33: Truth: 0.541, Prediction: 2.987 Difference 2.446
Plane 2, Straw 34: Truth: 0.541, Prediction: 3.938 Difference 2.887
Plane 3, Straw 34: Truth: 0.541, Prediction: 4.603 Difference 2.222
Plane 4, Straw 35: Truth: 0.541, Prediction: 0.067 Difference 0.475
Plane 6, Straw 44: Truth: 0.541, Prediction: 0.548 Difference 0.007
Tested!
Testing model...
Model inference took 0.0007275600000866689 seconds.
Plane 1, Straw 17: Truth: 3.672, Prediction: 3.778 Difference 0.106
Plane 3, Straw 15: Truth: 0.531, Prediction: 0.558 Difference 0.027
Plane 5, Straw 11: Truth: 3.672, Prediction: 3.723 Difference 0.051
Plane 6, Straw 10: Truth: 3.672, Prediction: 3.578 Difference 0.094
Plane 7, Straw 10: Truth: 3.672, Prediction: 3.819 Difference 0.147
Plane 9, Straw 9: Truth: 3.672, Prediction: 6.050 Difference 2.377
Tested!
```

```
Testing model...
Model inference took 0.0006295849998423364 seconds.
Plane 0, Straw 28: Truth: 3.678, Prediction: 0.349 Difference 2.954
Plane 1, Straw 27: Truth: 3.678, Prediction: 0.221 Difference 2.826
Plane 3, Straw 26: Truth: 3.678, Prediction: 1.044 Difference 2.635
Plane 5, Straw 21: Truth: 3.678, Prediction: 2.633 Difference 1.046
Plane 6, Straw 20: Truth: 3.678, Prediction: 0.706 Difference 2.972
Plane 7, Straw 20: Truth: 3.678, Prediction: 0.809 Difference 2.869
Plane 9, Straw 19: Truth: 3.678, Prediction: 0.397 Difference 3.002
Tested!
```

C++ Implementation

- Trained model in Python, analysis code in C++/ROOT
- Many methods tested to implement...
  - Torch C++ API
  - onnx2code
  - ONNXRuntime
  - ROOT TMVA/SOFIE
- Tested all on performance, ease of use: decided on *OpenCV*



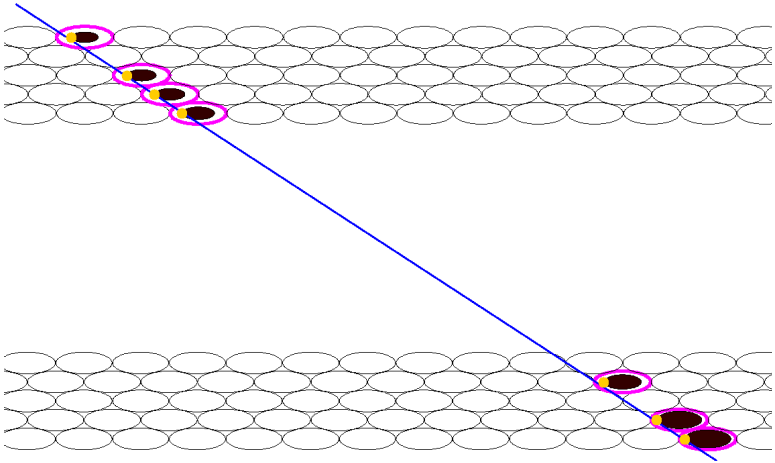
Sample OpenCV Code

```cpp
std::string model_path = "line_model_10.onnx";
cv::dnn::Net model = cv::dnn::readNetFromONNX(model_path);
model.setPreferableBackend(cv::dnn::DNN_BACKEND_OPENCV);
model.setPreferableTarget(cv::dnn::DNN_TARGET_CPU);
cv::setNumThreads(1);
std::array<int, 4> input_shape = {1, 2, 10, 89};
cv::Mat input_mat = cv::Mat::zeros(4, input_shape.data(), CV_32F);
```

```cpp
std::vector<size_t> good_indices;
for(int k=0;k<10;k++)
{
    for(int l=0;l<89;l++)
    {
        int flat_index = l + 89*k;
        if(input[0][0][k][l] > 0.f) good_indices.push_back(k*89 + l);
        input_mat.at<float>(flat_index) = input[0][0][k][l];
        input_mat.at<float>(flat_index + 89*10) = input[0][1][k][l];
    }
}

std::vector<std::string> outLayerNames = model.getUnconnectedOutLayersNames();
model.setInput(input_mat);
std::vector<cv::Mat> result;
model.forward(result, outLayerNames);
```

Tracking Implementation

- Prediction gives set of points that lie on each drift tube - fit this to a line $\rightarrow$ seed for full fit!



Conclusions/Future

- Predictor CNN progressing very well
- Needs to be trained on more simulated data
- Fine tune NN structure, hyperparameters, loss function, etc.
- Work on optimizing C++ implementation