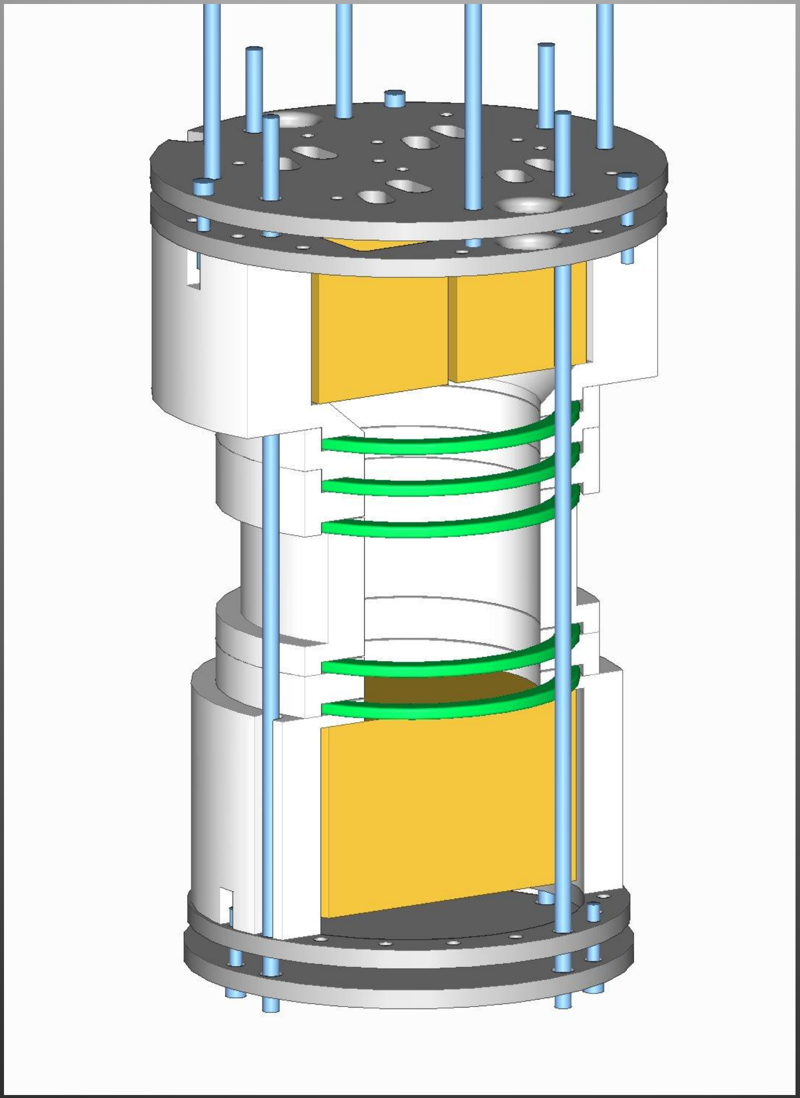
Position Reconstruction in Miniature Detector Using a Multilayer Perceptron

By Adam Levine

Introduction

- Detector needs algorithm to reconstruct point of interaction in horizontal plane

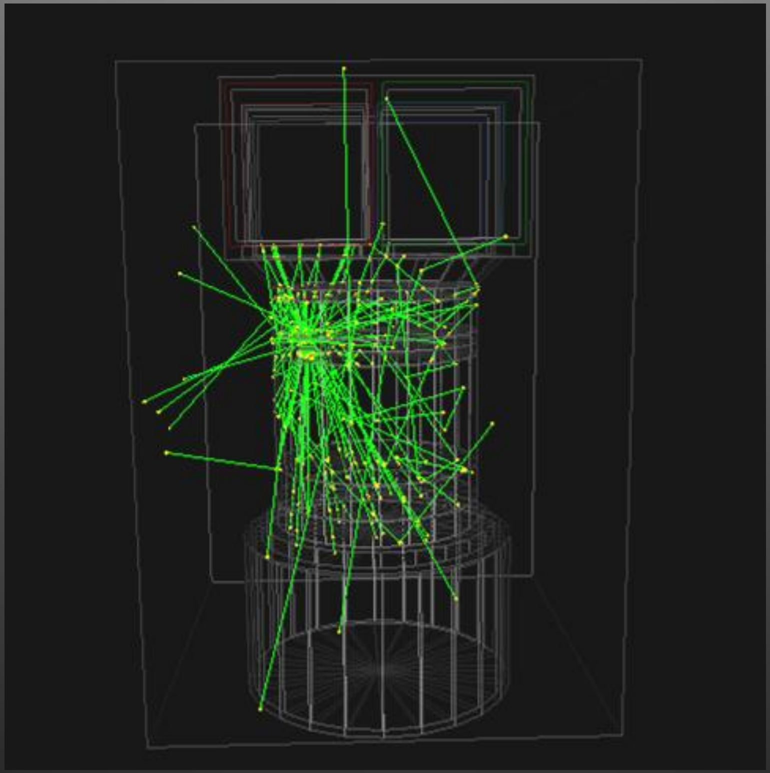


Geant4 Simulation

- Implement Geant4 C++ libraries
- Generate primary particles randomly and map

PMT signal to primary position

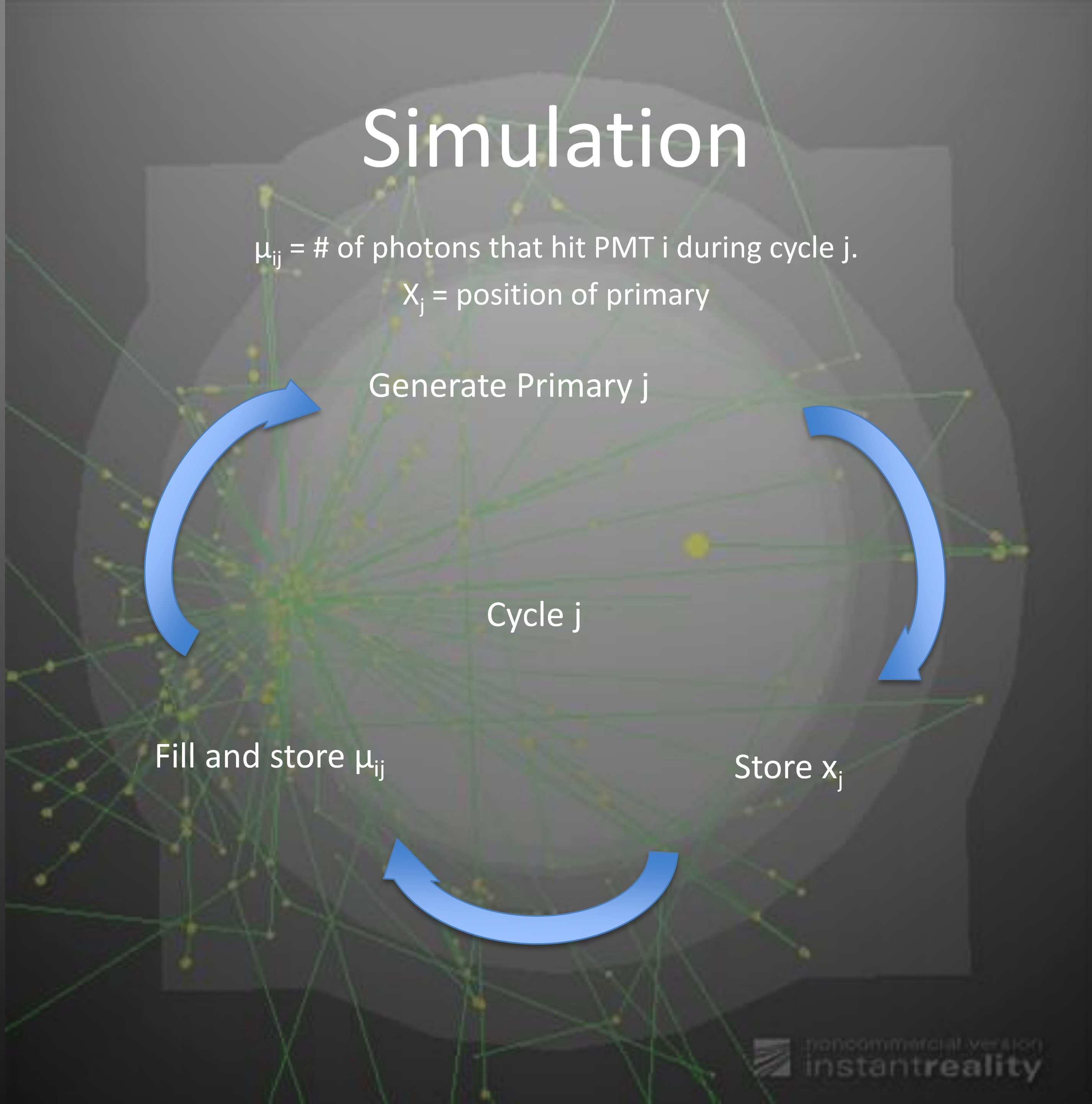
- Simulate S2 to get horizontal position, drift time to get vertical



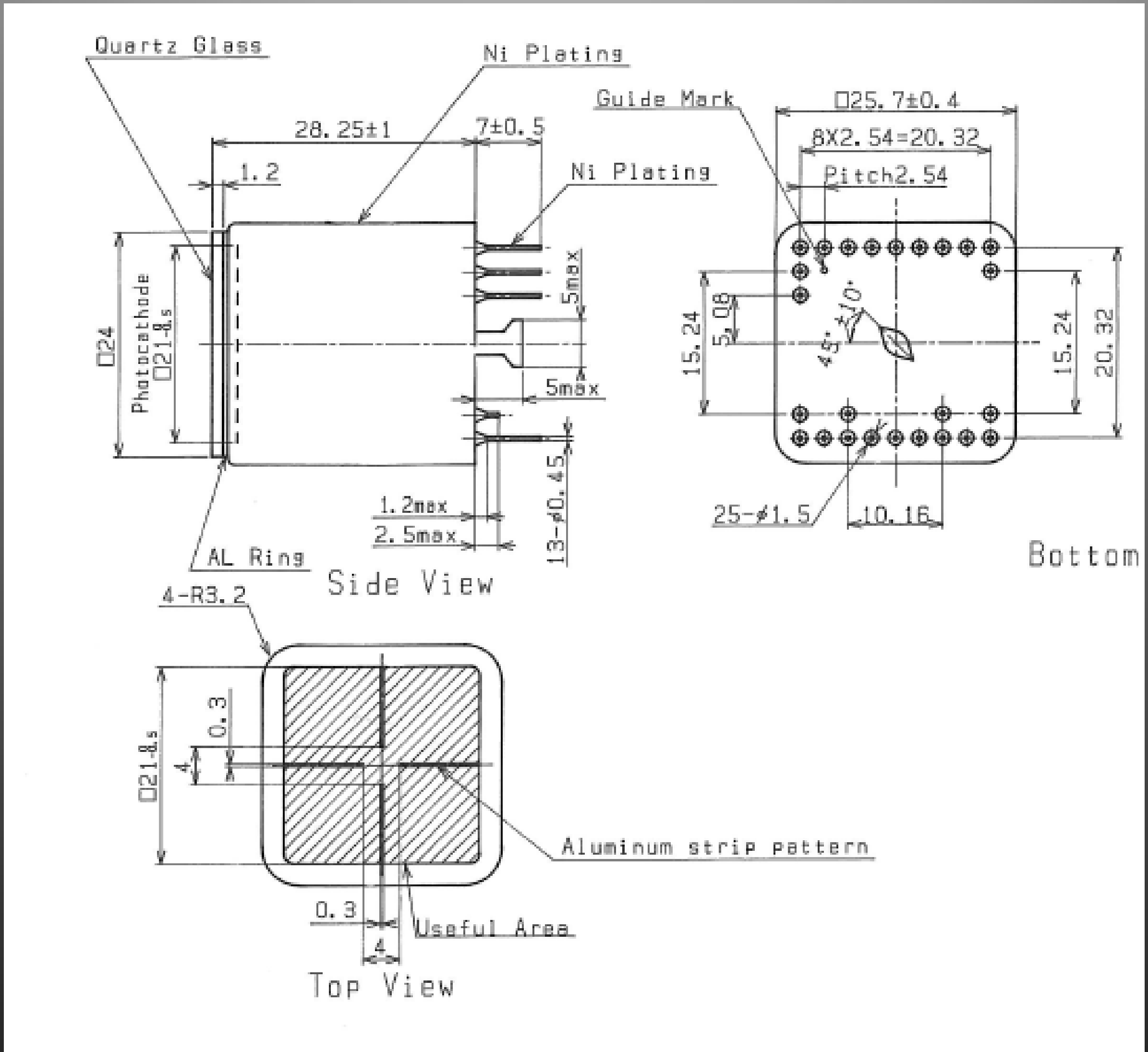
Simulation

μ_{ij} = # of photons that hit PMT i during cycle j.

X_j = position of primary



PMT Construction

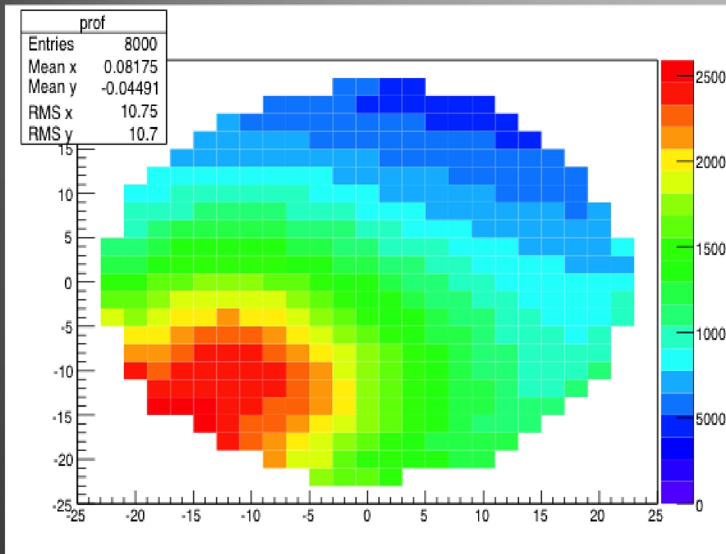


Simulation Stats

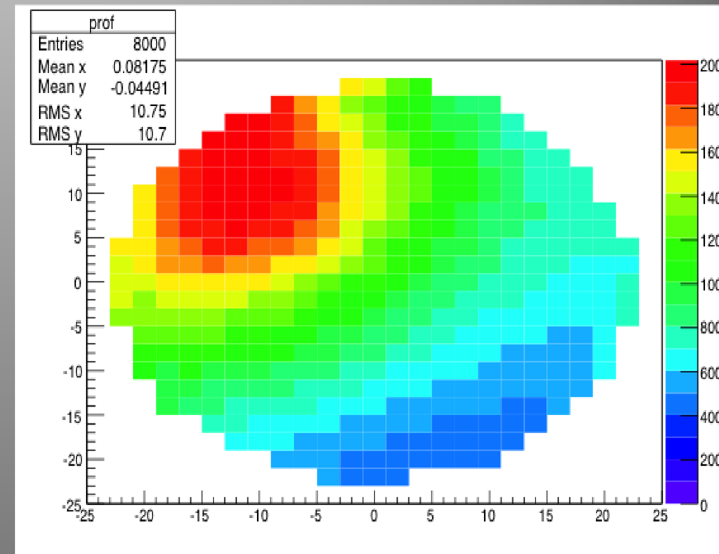
- Ran 8000 cycles on campus computer
- Each cycle, fired 1keV e^- into GXe just above LXe surface
- Scintillation yield of the GXe was set to 375000/keV (unphysical, just used to generate photons)
- Number was chosen so that the average number of photon hits per pmt per run ~10000

PMT hits versus Position of Primary

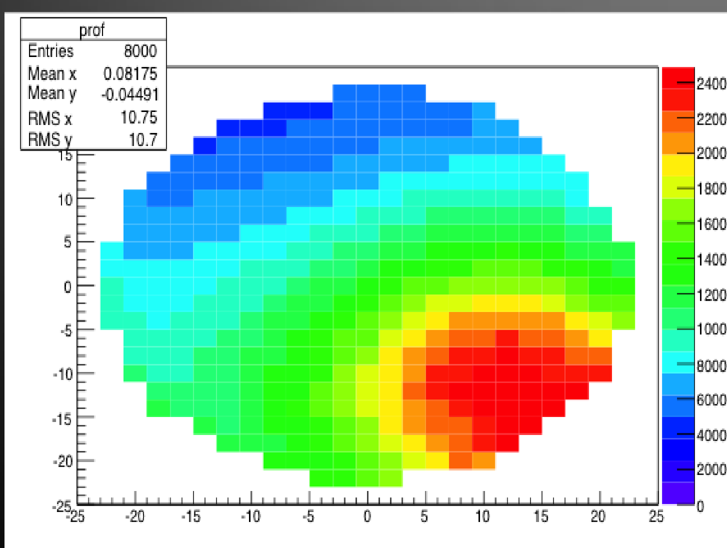
PMT 1



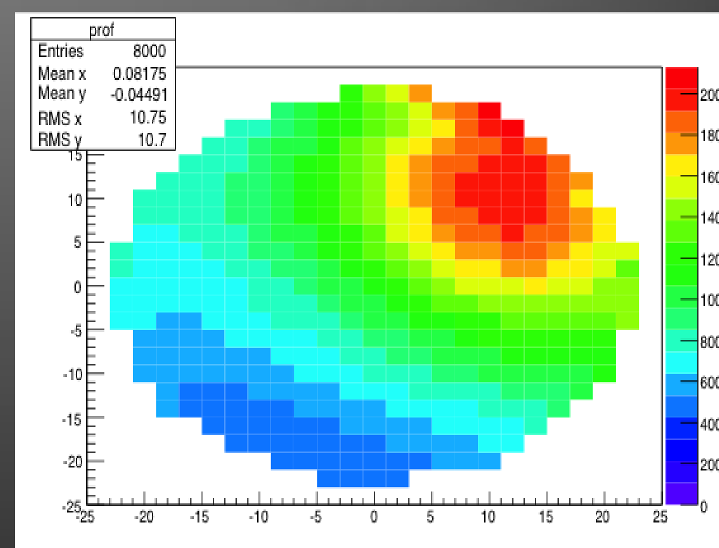
PMT 2



PMT 3



PMT 4

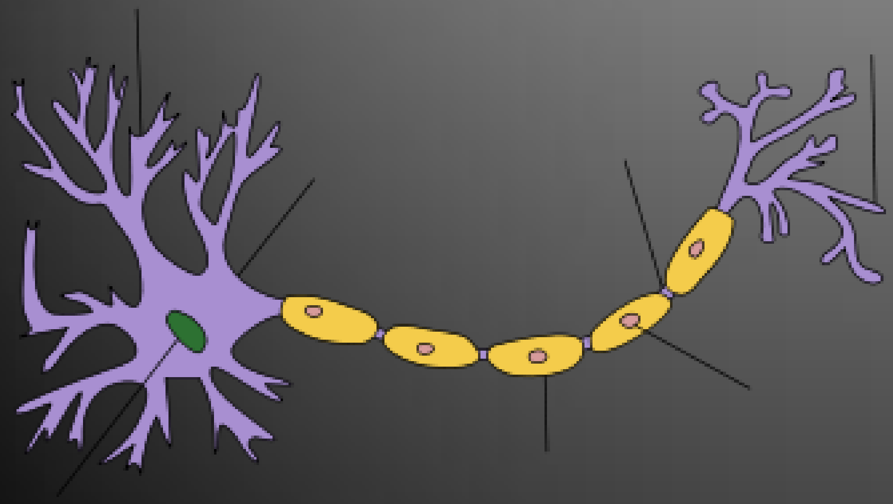


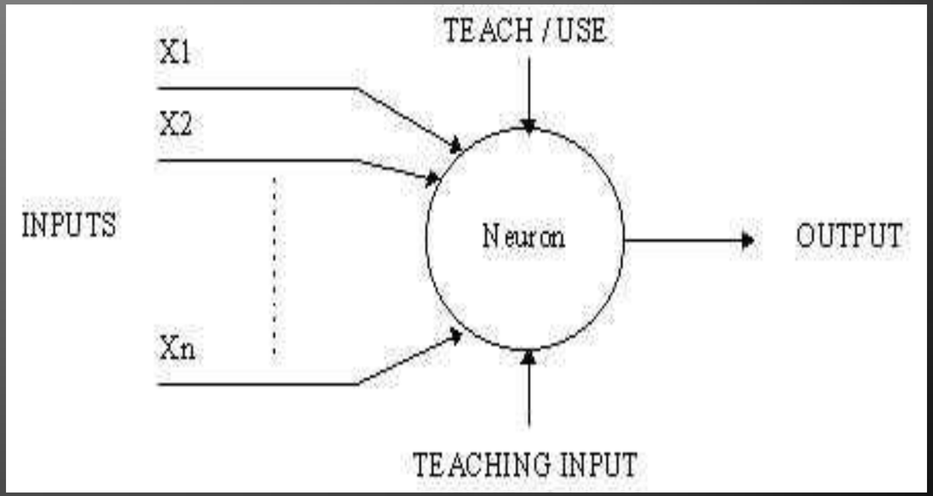
Making the Algorithm

- Goal: Find a function ƒ: R^N -> R^2 (where N is the number of PMTs) that assigns a PMT signal to its primary's position
- N=4 if we , N=16 if we do
- Work backwards to train a *Neural Network*

What is a Neural Network?

- A neural network is a structure that processes and transmits information
- Modeled directly after the biological neuron





What is a MultiLayer Perceptron?

- Subset of Artificial Neural Networks
- Uses structure of neurons, along with training algorithm and an objective functional
- Reduces problem to extremization of functional/function
- Implement FLOOD Open Source Neural Networking library

MultiLayer Perceptron Structure

- Take in scaled input, calculate hidden layer vector with N components where N is the number of hidden neurons
- Send each component through an “Activation Function” often threshold functions that range between 0 and 1 or -1 and 1
- Repeat, until out of hidden layers, send it through Objective Function and then unscale the output.

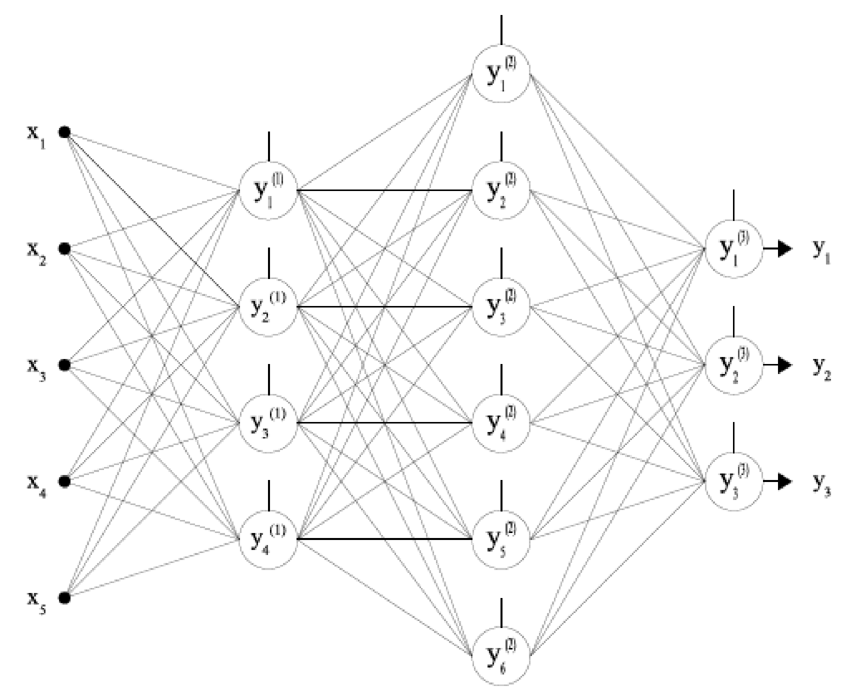
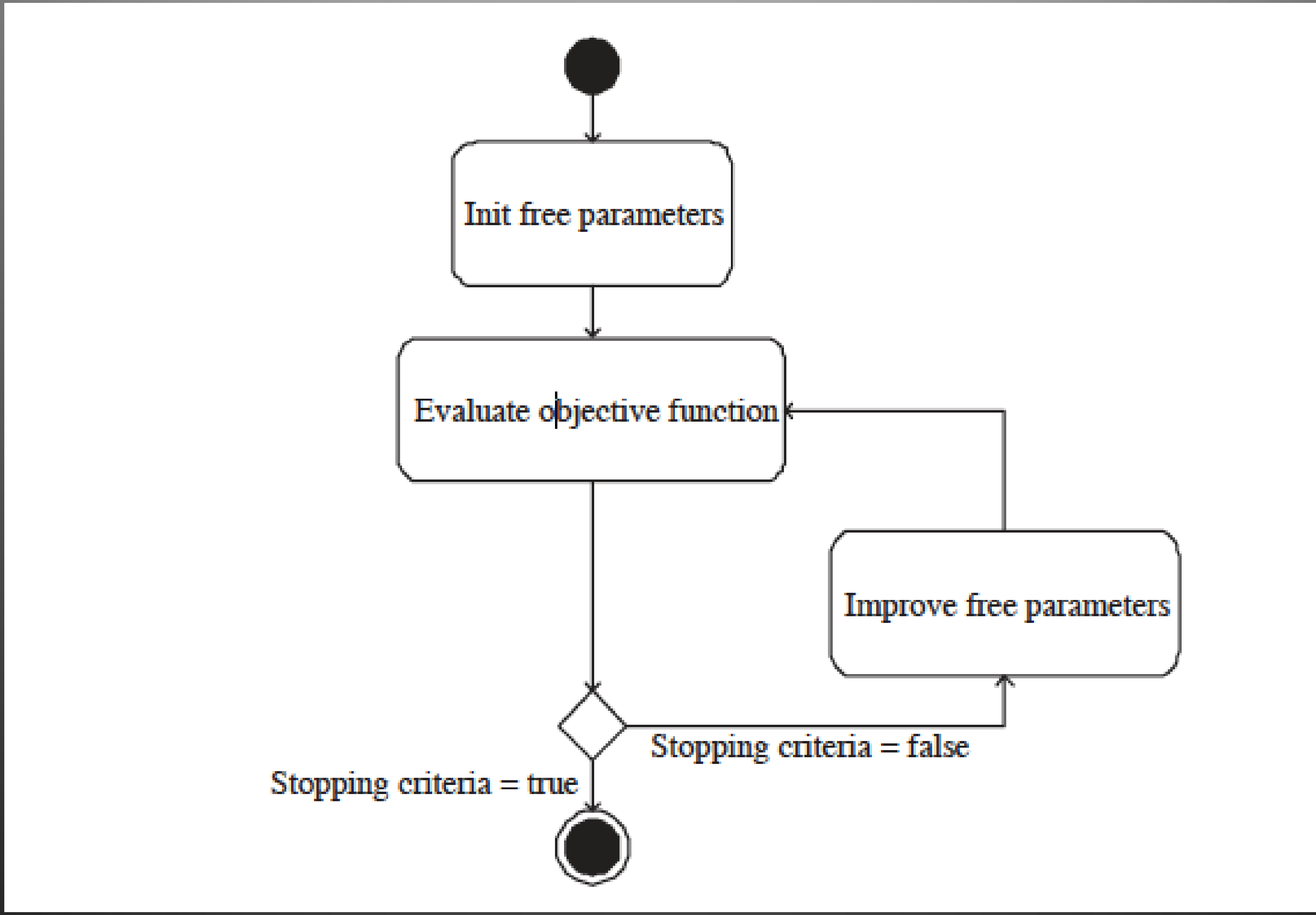


Figure 4.2: Multilayer perceptron example.

Training Structure



The Math Behind the MultiLayer Perceptron

$$\mu_i$$

$$\sum_i \mu_i \cdot w_{ij} = h_j$$

$$f_{aj}(h_j)$$

Repeat Until Out of Hidden Layers

$$o_j = O(f_{aj}(h_j))$$

TRAIN (if needed)

Unscale output, Send through objective function

KEY:
W_{ij} = Weight Matrix
μ_i = input vector
f_{aj} = activation function
O= output activation function
o_j = output vector

Objective Function and Training Algorithm

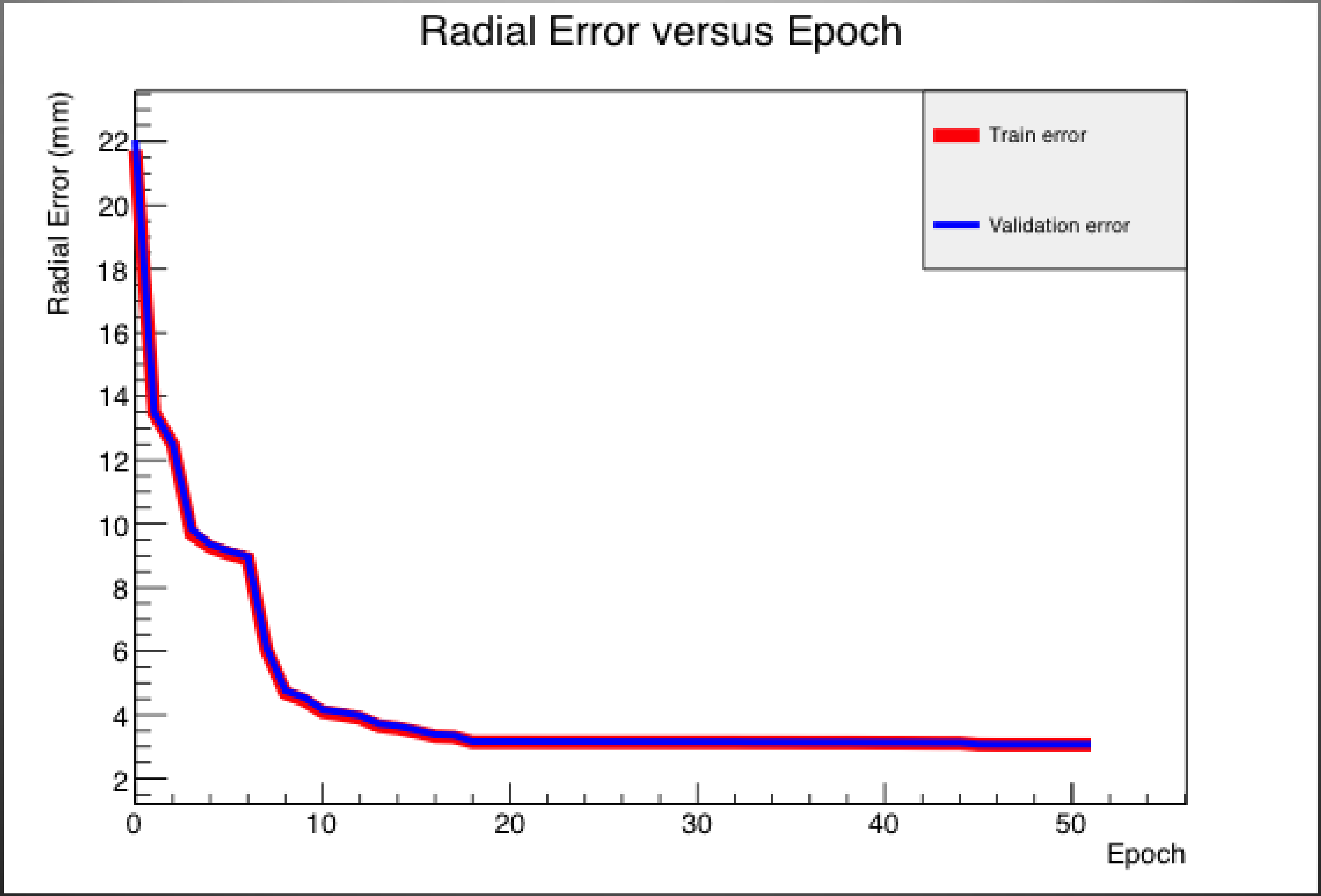
- Used Conjugate Gradient algorithm to train
- Calculates gradient of Objective function in parameter space, steps down function until stopping criteria are reached

x_i = ideal position

o_i = outputted position

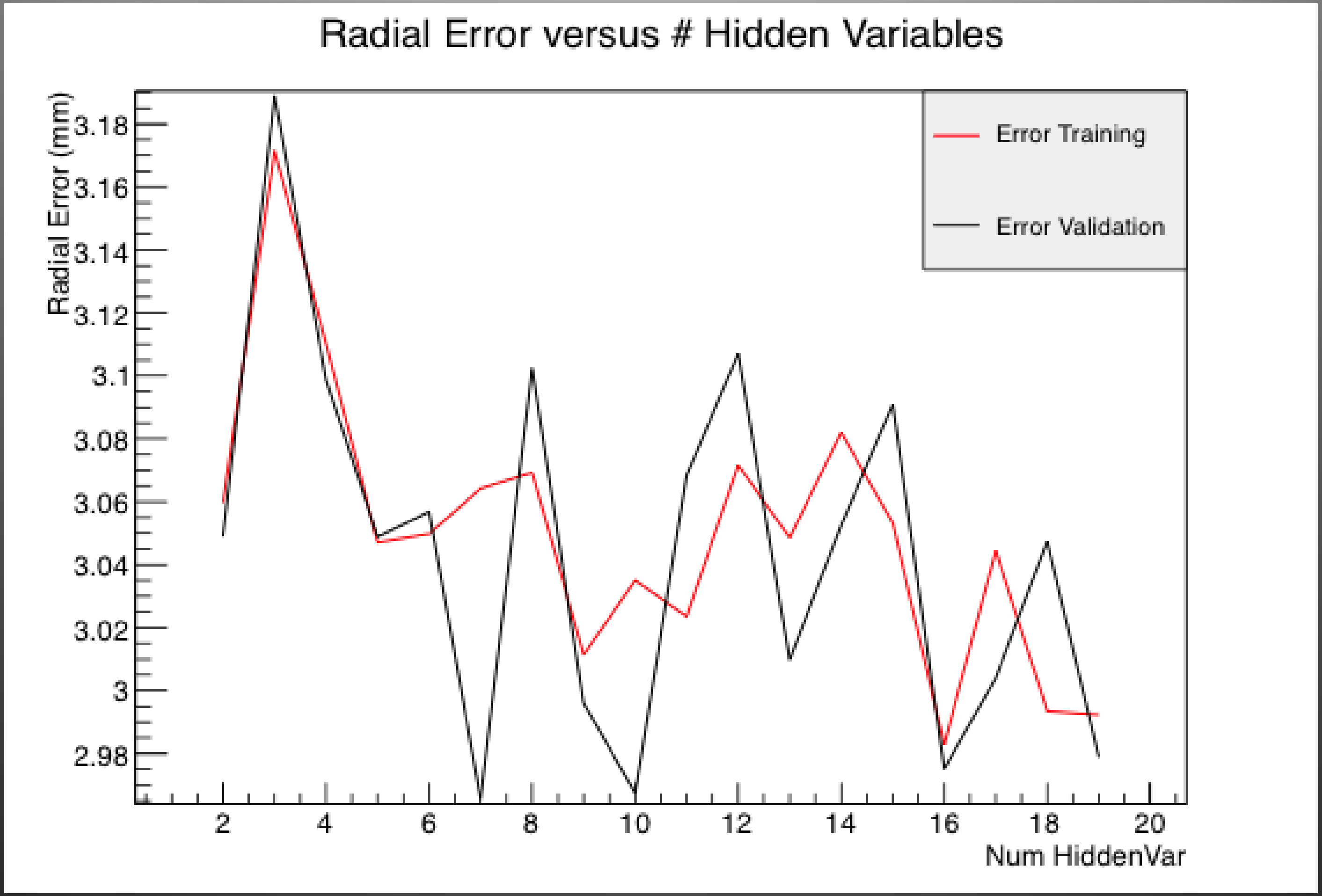
$$Obj(x_i, o_i) = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(\sum_{i=1}^{2}(x_i - o_i)^2)}$$

Radial Error vs. Epoch



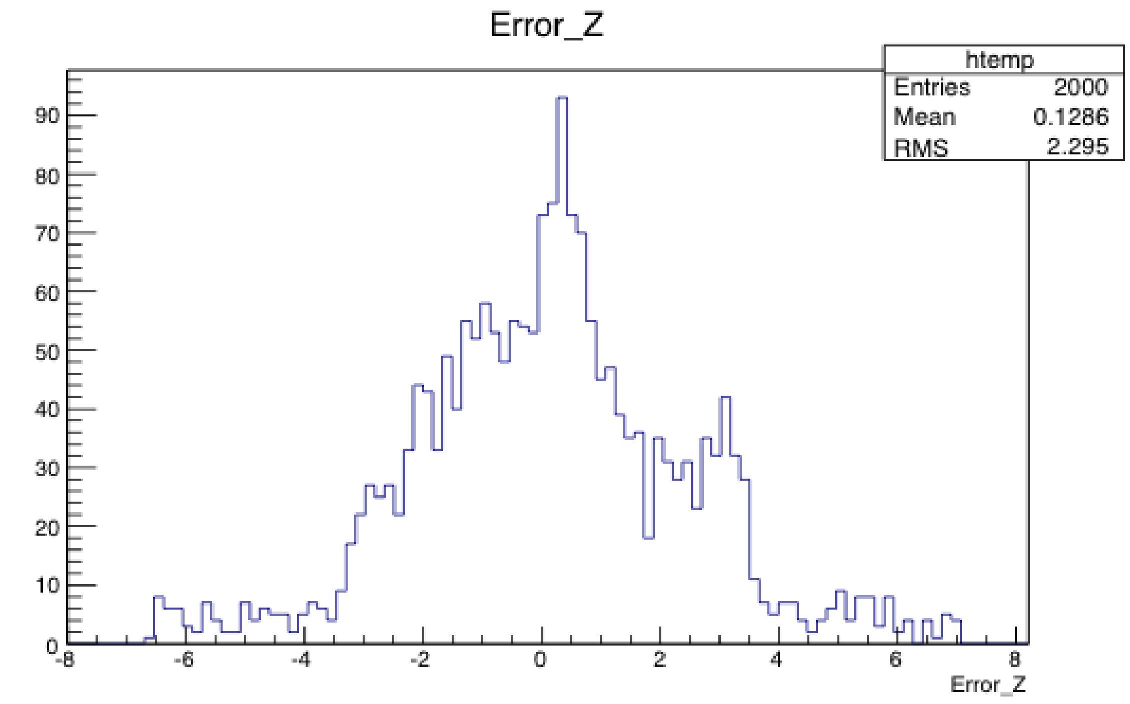
Used to check if overtraining has occurred.

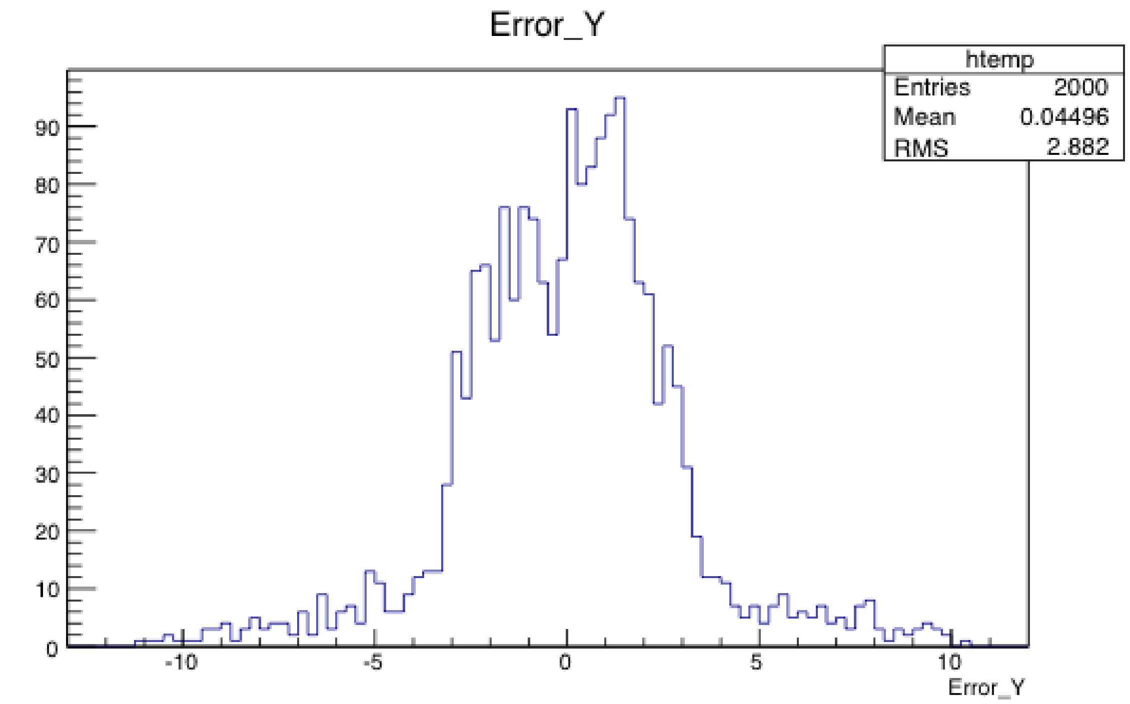
Final Radial Error vs. Number of Hidden Neurons



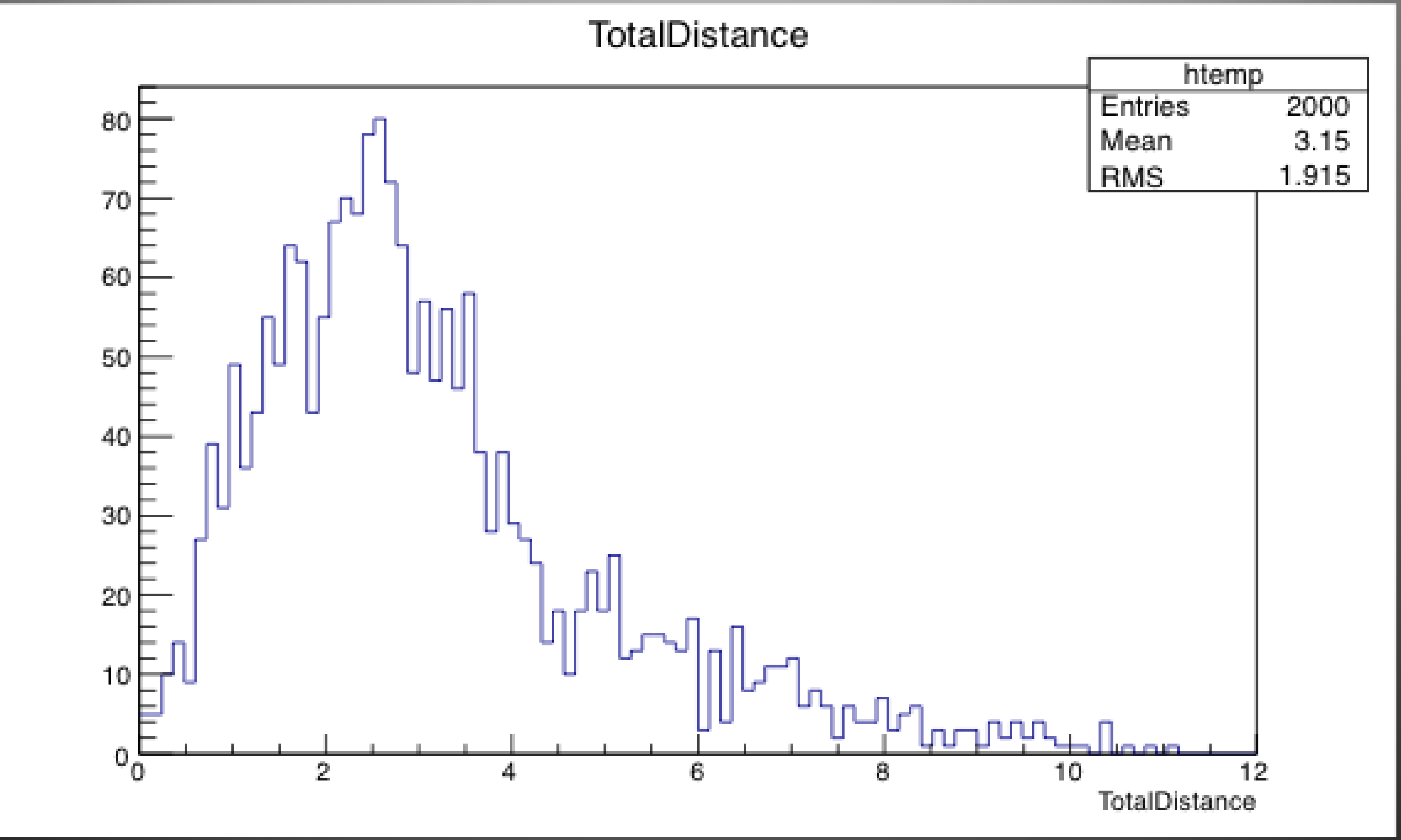
Odd Point: Overtraining doesn't seem to be happening even up to 19 hidden layer neurons!

Ideal coordinates minus outputted coordinates (mm)





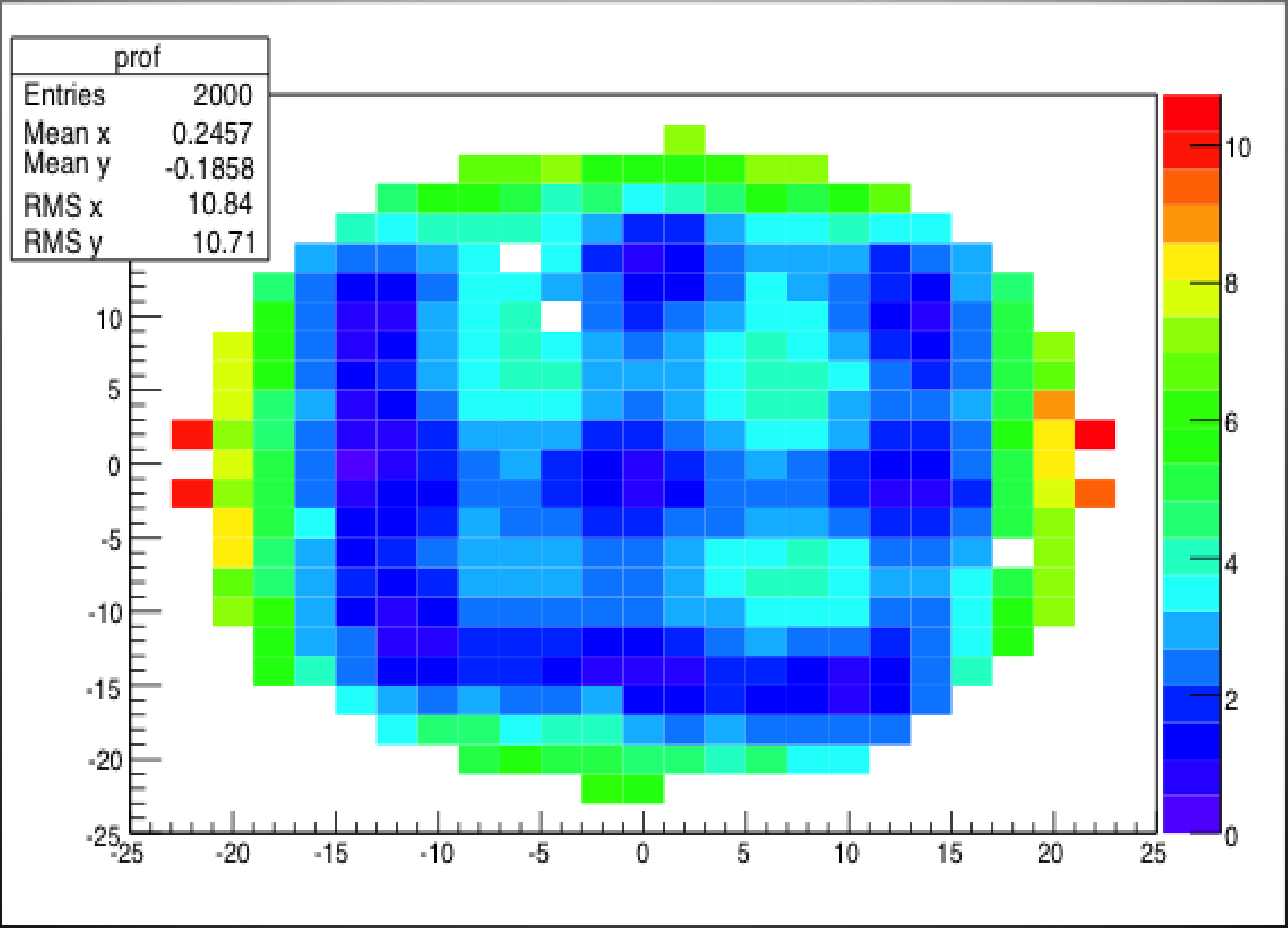
Error(mm) of 2000 primaries after Perceptron has been trained



Note: These 2000 points were not used to train the Perceptron

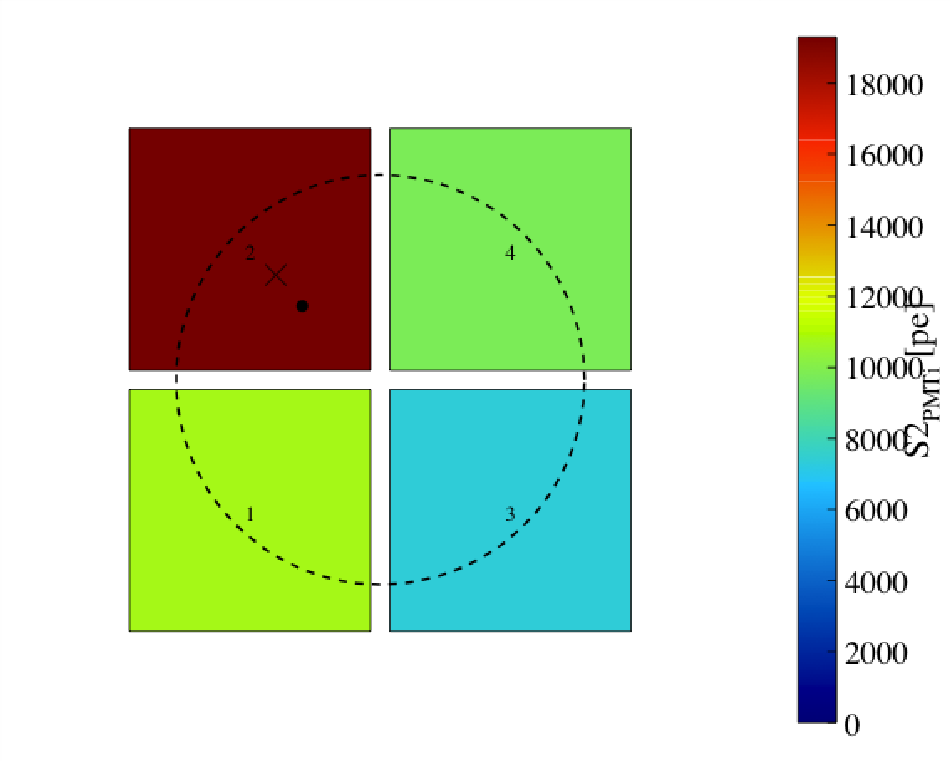
GOAL: Get Mean down to ~1 mm

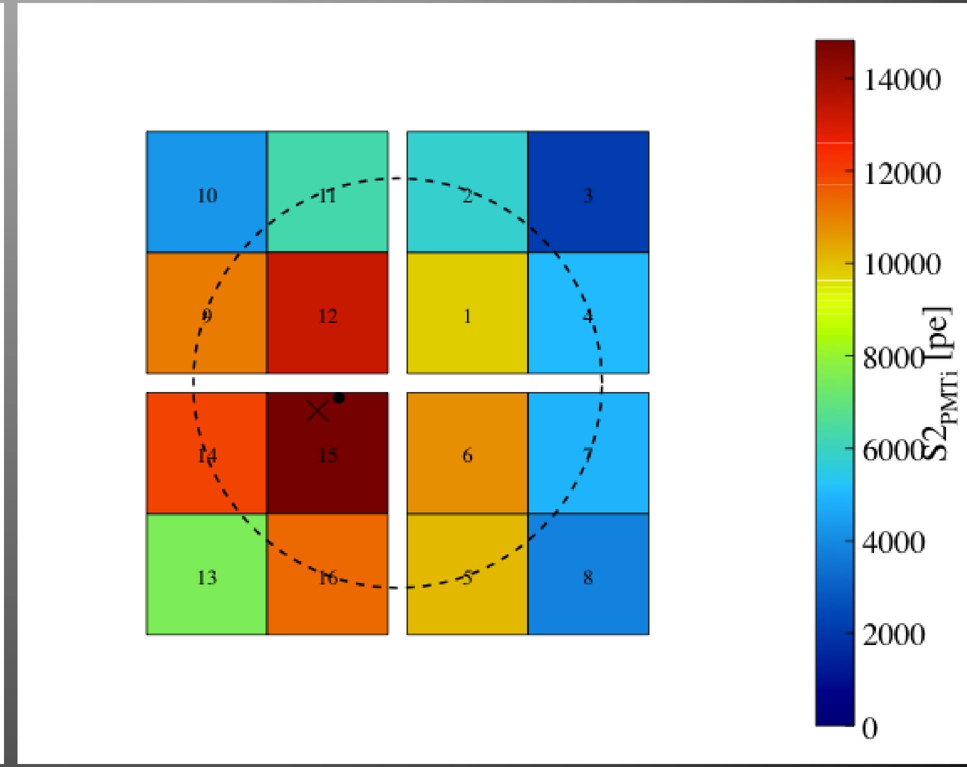
Error(mm) vs. primary position



Example

Both outputs used perceptron trained with just 4 PMTs



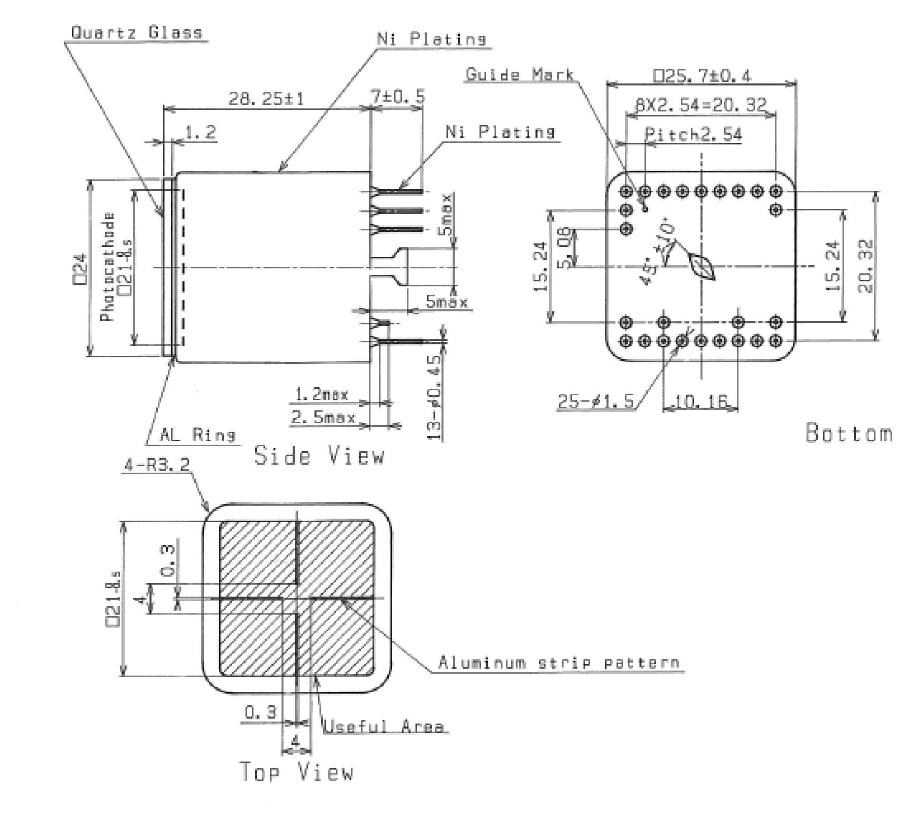


What’s Next

- Still need to figure out why radial error seems to plateau at around 3mm

Possible Solutions:

Simulate extra regions of sensitivit y to effectively increase number of PMTs

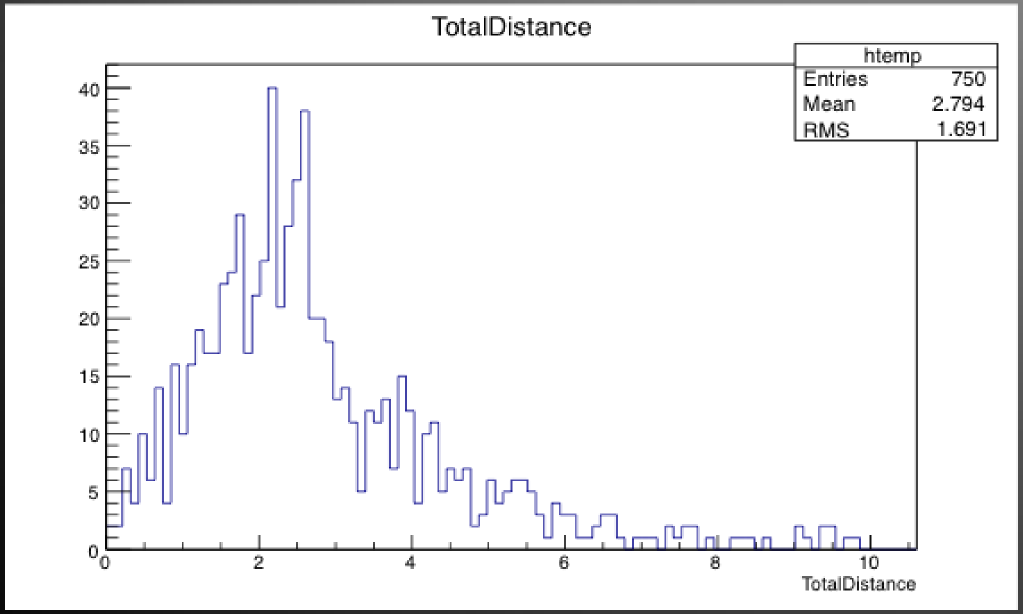


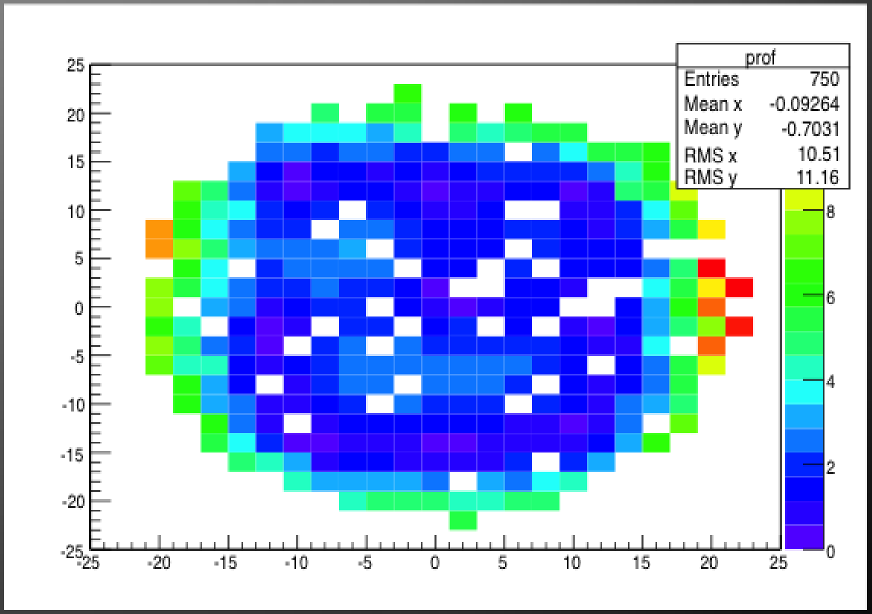
Also: Not getting 100% reflectivity in TPC

With extra SubDetectors

- Quickly ran the simulation 3000 times with this added sensitivity (16 distinct sensitive regions)

Preliminary Graphs:





Still need to run more simulations…