

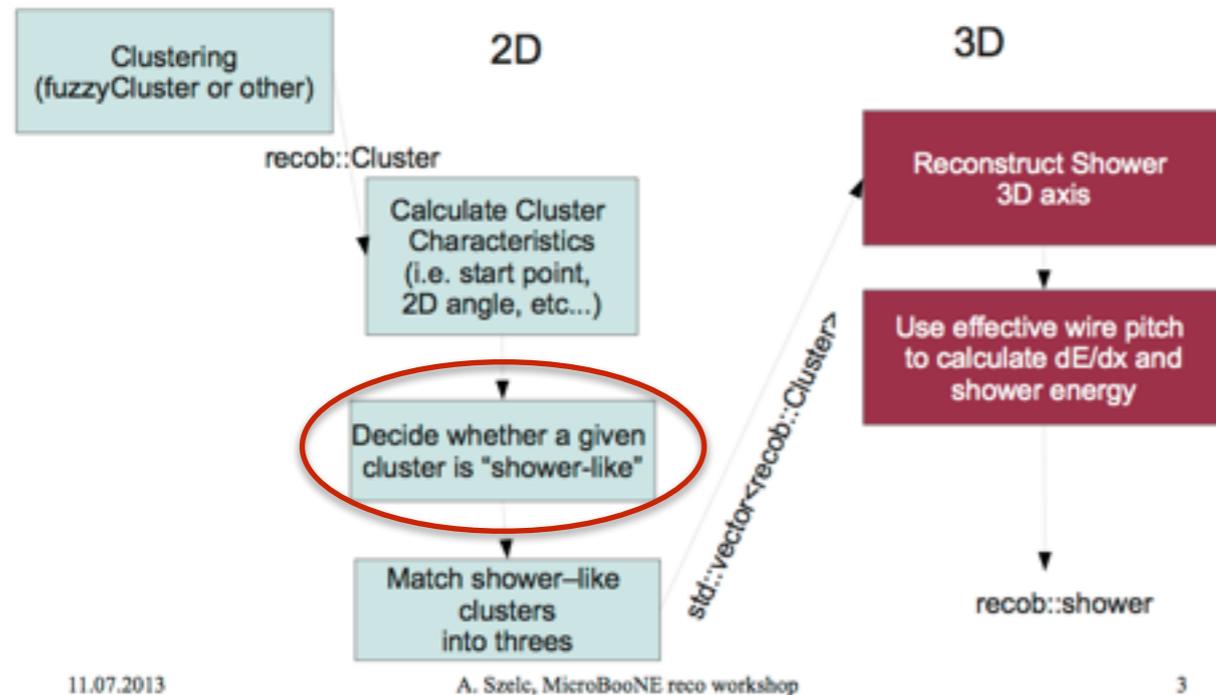


Track Shower Separation

Corey Adams, Yale
3/13/14 Analysis Tools Meeting

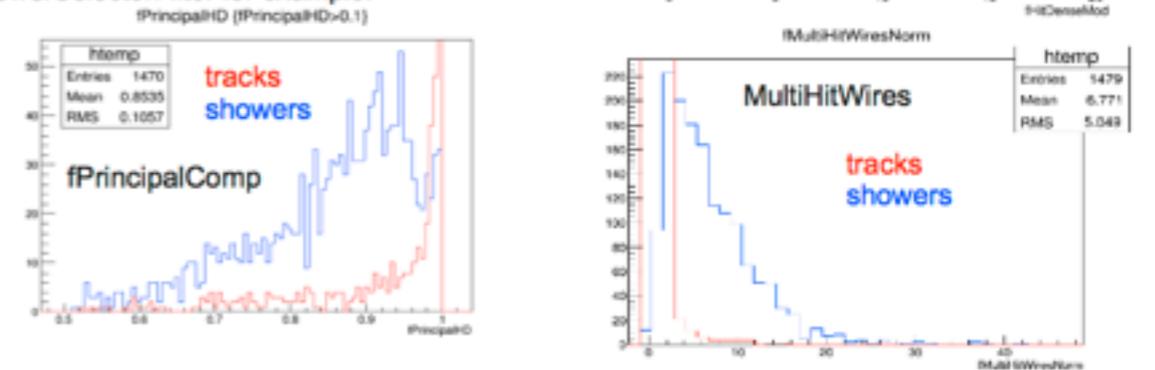
Why separate tracks and showers?

The Current Paradigm of Reconstructing Showers



2D Clustering → Shower-like Cluster Selection

- When Calculating the "direction weight" and local start points a lot of helper parameters get calculated.
- These plus others can be used to differentiate tracks from showers.
- Currently The most powerful are Modified Hit Density, MultiHitWires and PrincipalComponents.
- These methods are exercised in ShowerSelectorFilter for example.



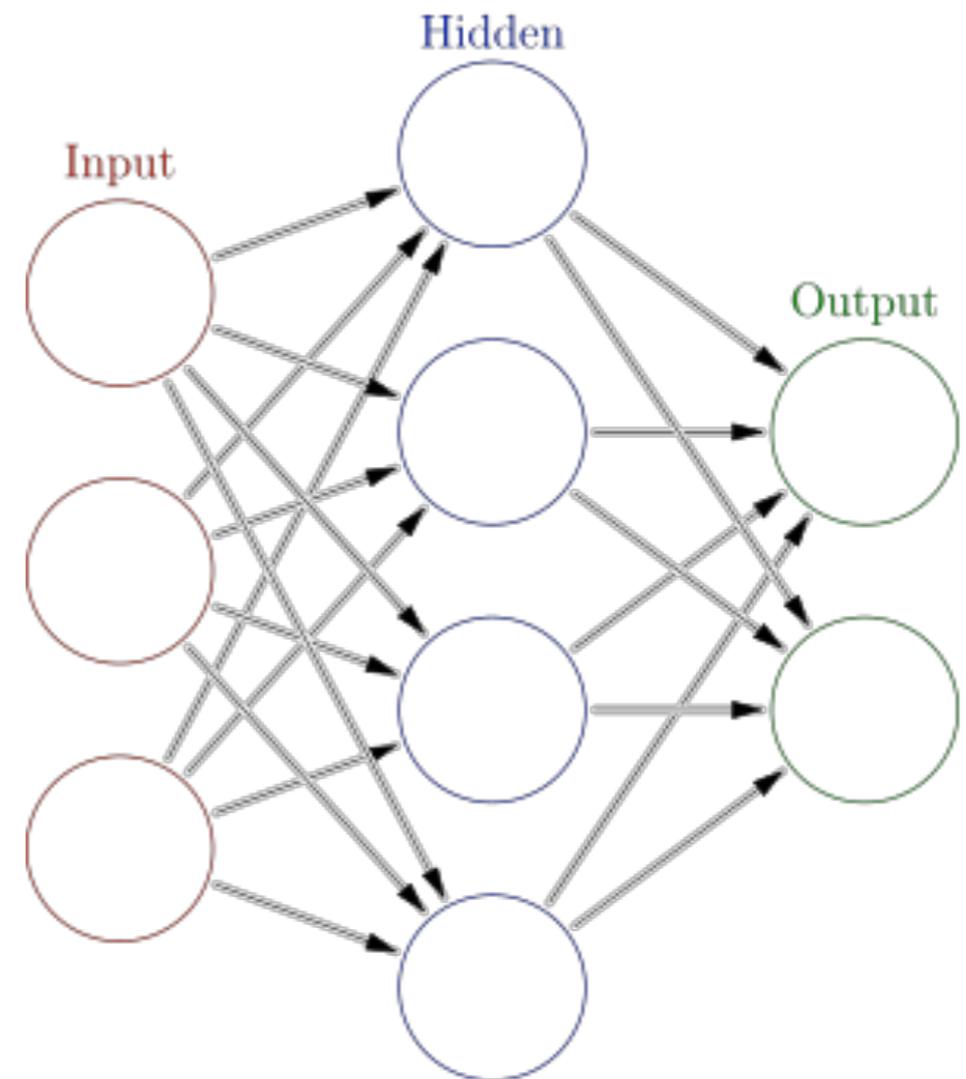
Andrzej, docdb 2990

This work has been done before but I wanted to try a new method to combine all of the previous results.

I incorporate (read: steal) a lot of the good ideas that have come before me.

Method: Artificial Neural Network

- There are a lot of useful parameters to make the distinction between tracks and showers, but they are all related in a complicated (and unknown) way.
- Proposed solution: use an artificial neural network to find the patterns and make the distinction between tracks and showers.
- Neural networks are excellent tools to recognize difficult patterns from complicated examples and extend to new data.



Implement an Artificial Neural Network?

- No way. Also didn't use the root TMVA package.
- Chose to use the Fast Artificial Neural Network (FANN) library - <http://leenissen.dk/fann/wp/>
- Library is open source, released under LGPL, has great documentation, many useful features implemented.
- Cross platform, easy to build and install, very easy to create, train, store, and use a neural network.

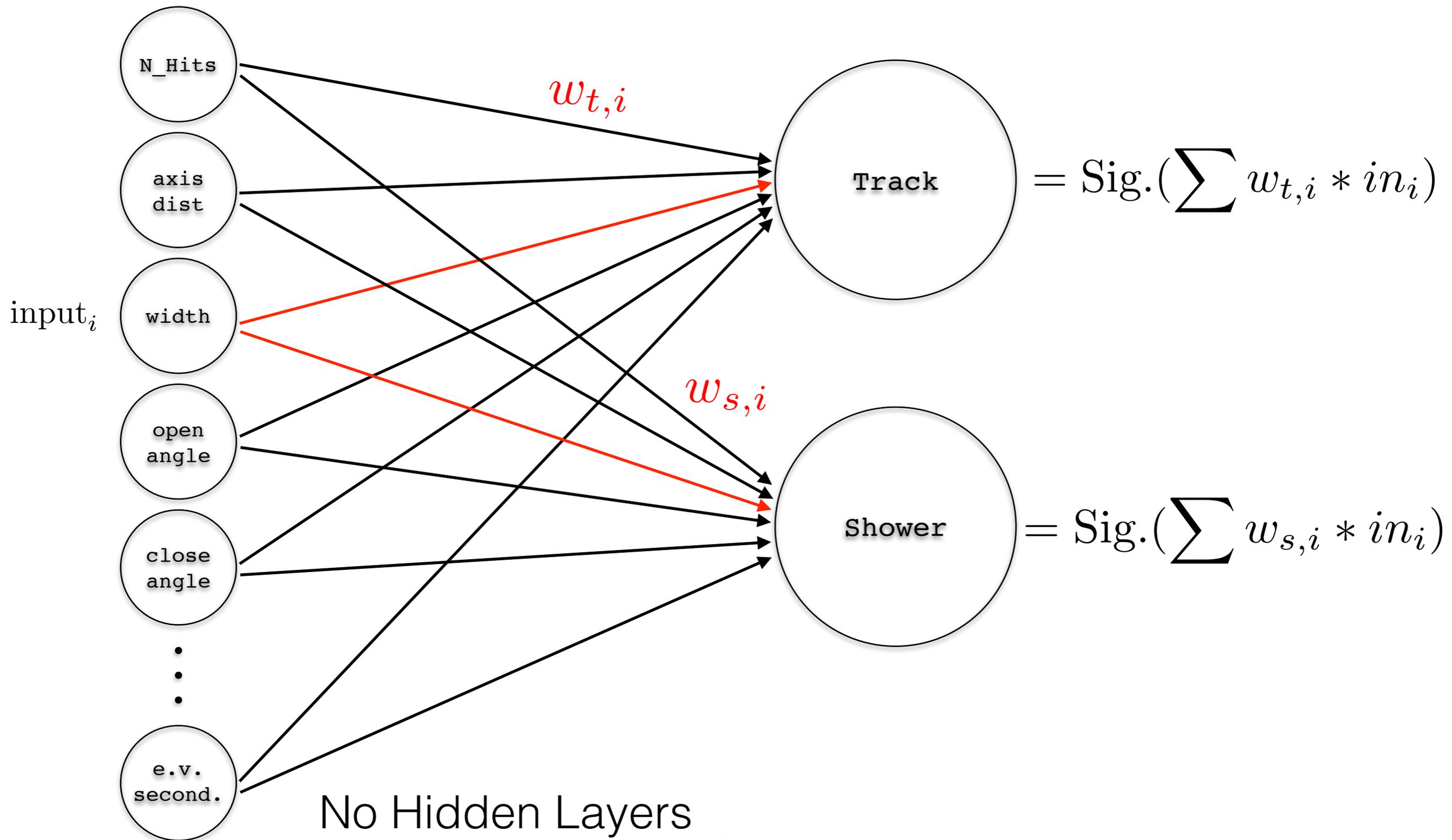
Input to ANN

- Can't just feed a `recob::cluster` and associated hit list into a neural network, so instead determine (with input from authors of shower and cluster finding algorithms) a list of parameters that can be calculated from a cluster and hit list.
- Created a `RecoAlg` designed to quickly and efficiently calculate these parameters
 - Very minimal `larsoft/art` dependancies in order to be portable (to, say, `LArLight`)

The input parameters (Feature Vector)

```
// This list enumerates the items in the feature vector, in order.
// They must all be floats!
float N_Hits;           // Number of hits in this cluster
float axis_dist;       // Length along the assumed axis of the cluster
float width;           // Width of cluster, perp to the axis above
float opening_angle;   // angle which contains some (high) percent of the hits
                       // from the start point of the cluster
float closing_angle;   // same as above, but from the end point of the shower
float opening_angle_HH; // as above, but only counting hits with charge
                       // above a threshold.
float closing_angle_HH; // As above, but only hits with charge above thresh
float hit_density_1D;   // Number of hits per length, collapsing into 1D
float hit_density_2D;   // Number of hits divided by (length*width)
float end_x;           // All showers are shifted to put the "start" at (0,0)
float end_y;           // this and previous are the x, y points in cm, cm space
float mean_x;          // the location of the center point, simply using hit peaks
float mean_y;          // the same
float mean_x_charge_wgt; // the center, but weighing each point by charge
float mean_y_charge_wgt; // the center, but weighing each point by charge
float multi_hit_wires; // the number of wires with more than one hit
float N_Wires;         // the number of wires with hits in this cluster
float eigenvalue_principal; // the principal eigenvalue from PCA
float eigenvalue_secondary; // the secondary eigenvalue from PCA
```

Use a simple network for now:



Need to train the network on known data sets

- Using single particles for now, and creating a “training list” of input vectors and answers.
- FANN library contains training algorithm to iteratively change the weights w_i until the overall error is minimized.

```
#include "fann.h"
#include <iostream>

int main(){
    std::cout << "Starting the training program ... " << std::endl;
    struct fann * ann = fann_create_standard(2, 19, 2);
    fann_train_on_file(ann, "FANN_testing_output.txt", 200000, 500, 0.001);
    fann_save(ann, "track_shower_discrim.net");
    fann_destroy(ann);
    return 0;
}
```

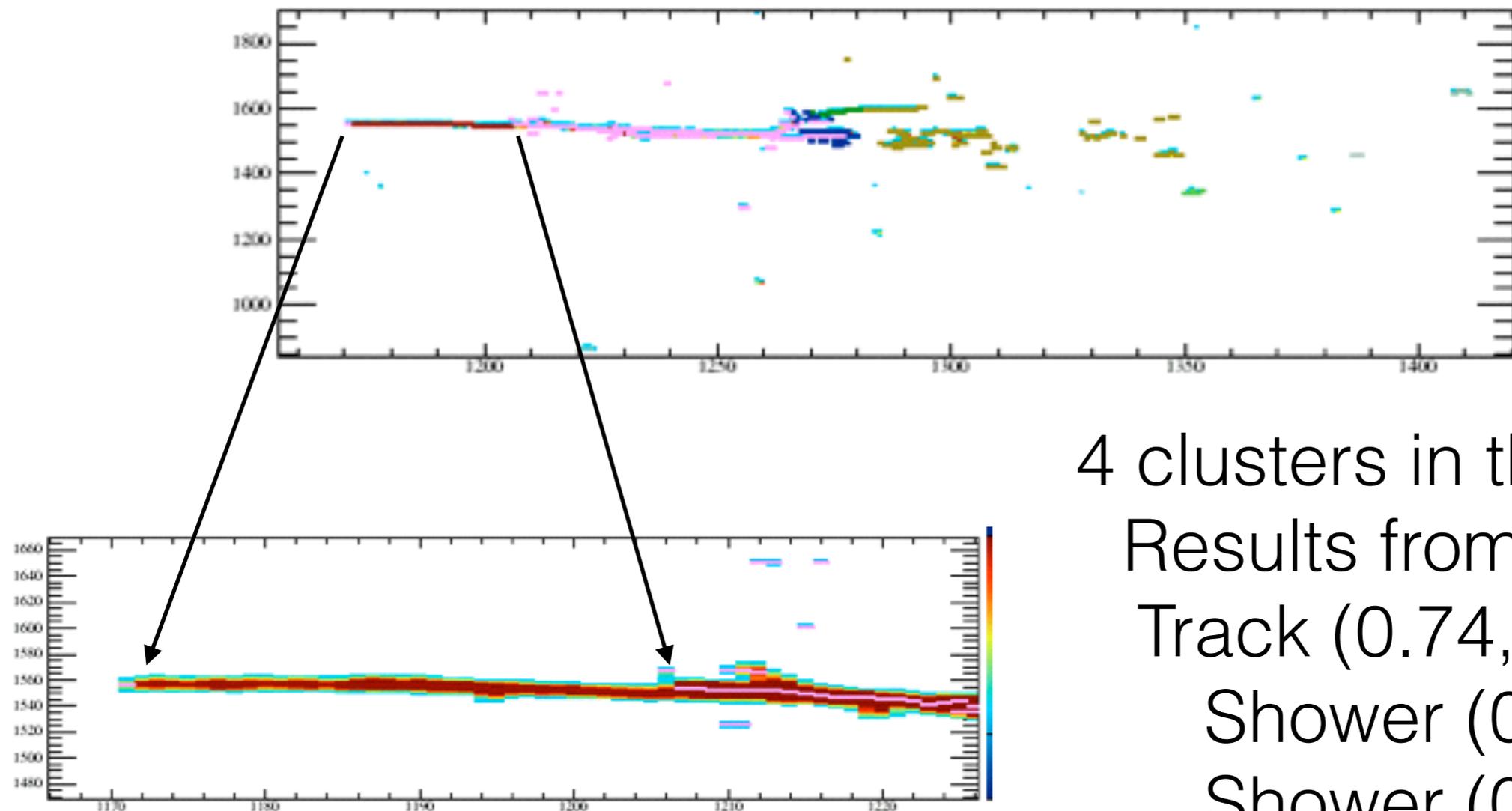
Preliminary Results

- Trained a neural network on single electrons and muons, and then evaluated performance on new events. Only looked at big ($N_{\text{Hits}} > 30$) clusters.

	N Events	N Clusters	ID'd As Shower	ID'd As Track	Not ID'd
Single Electrons	250	1771	1694	38	39
Single Muons	445	1355	18	1331	6

Success Rate: ~95%

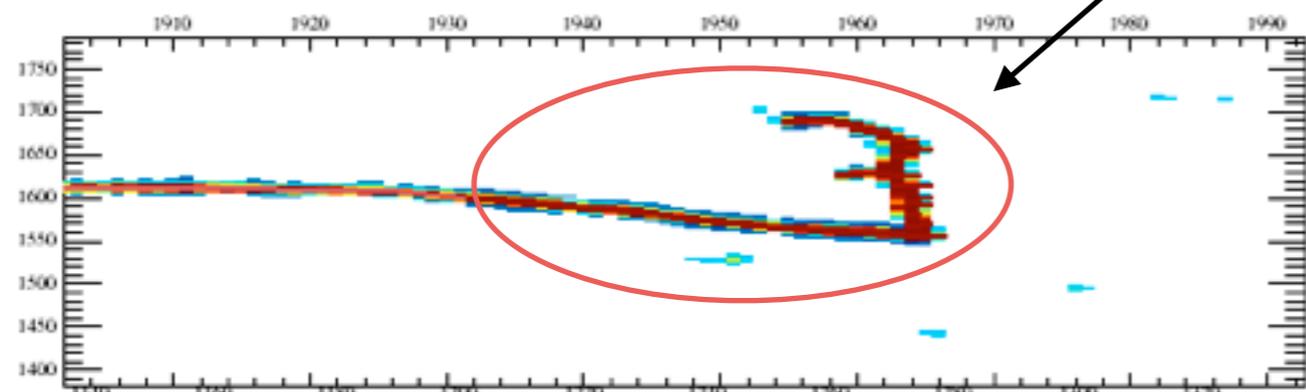
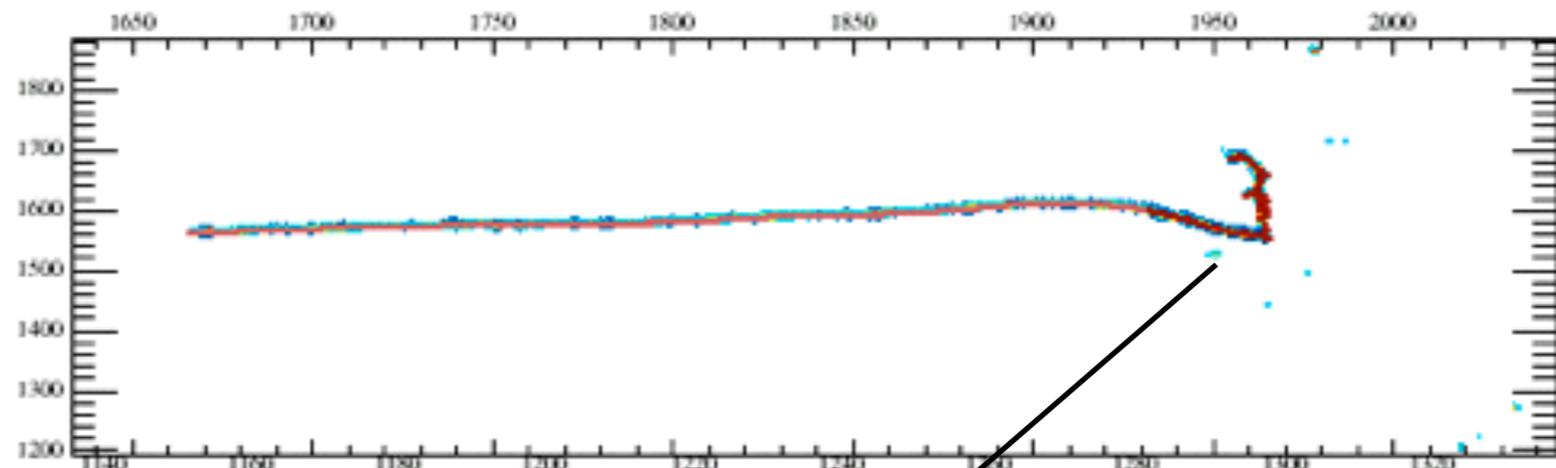
An Electron Failure Event



This does actually look like a track

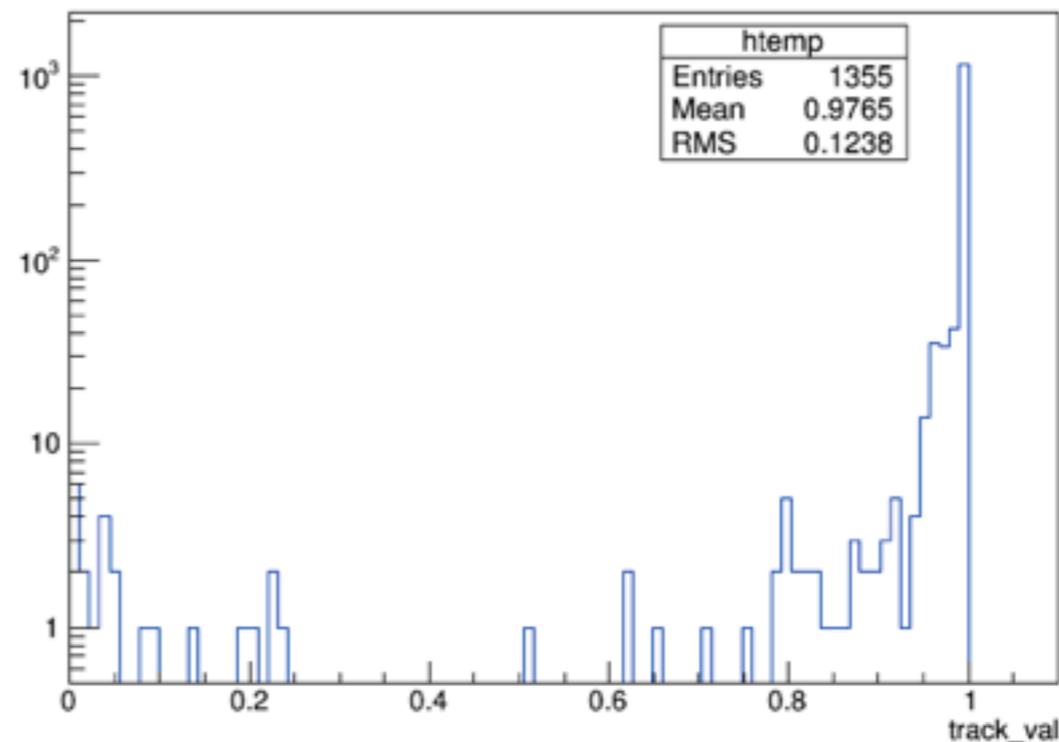
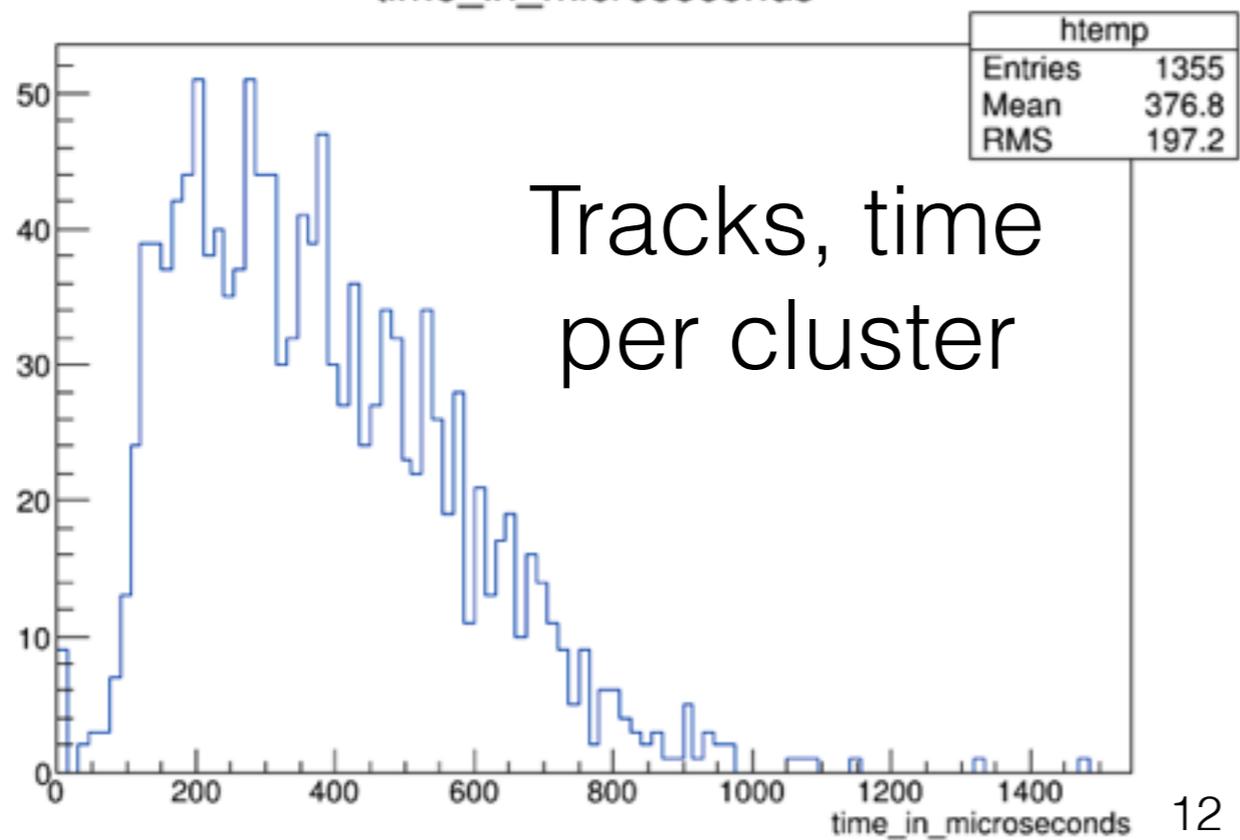
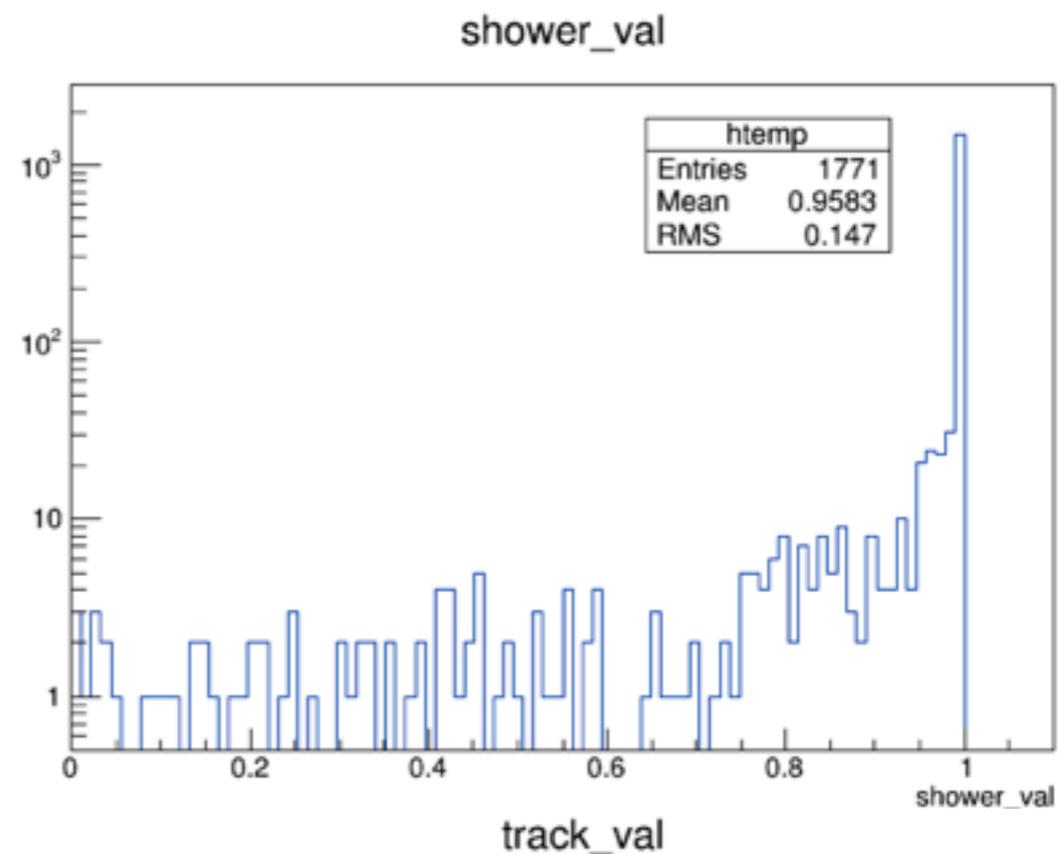
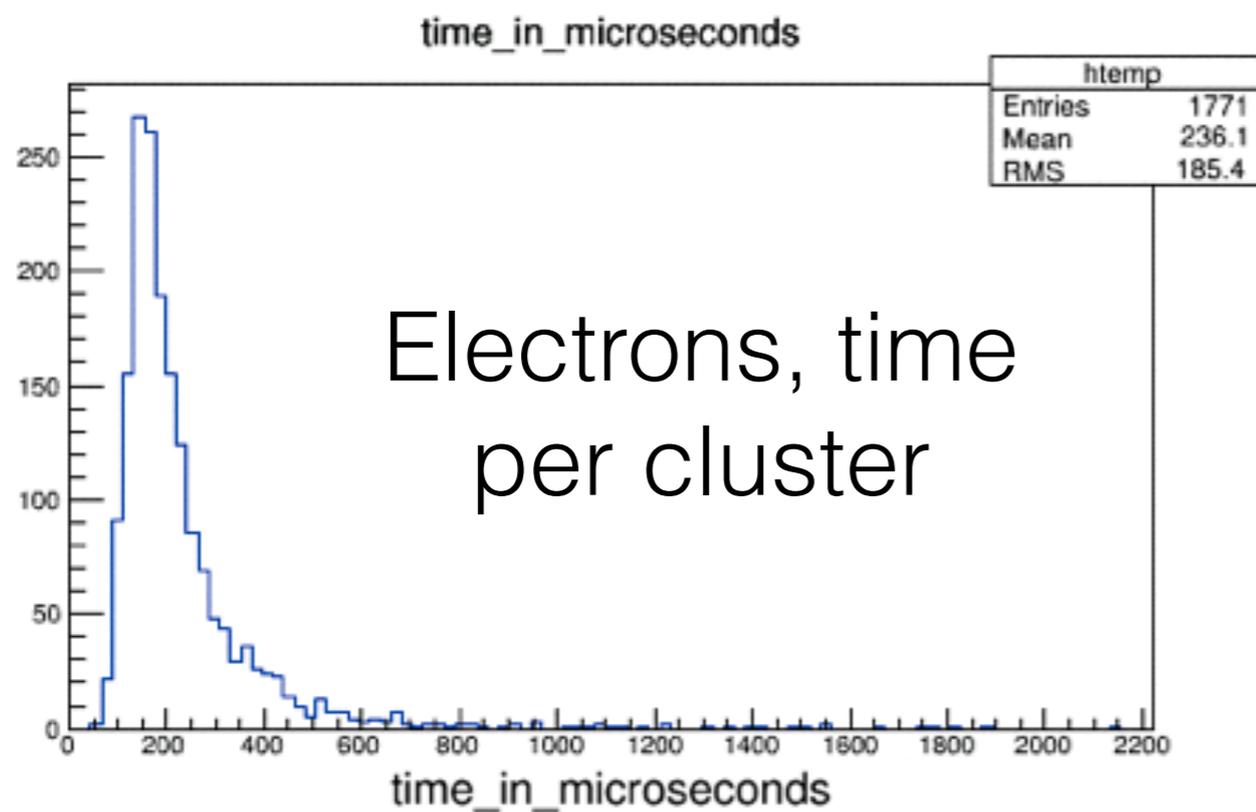
4 clusters in this view
Results from ANN:
Track (0.74, 0.25)
Shower (0, 1)
Shower (0, 1)
Shower (0, 1)

A Track Failure Event



Michel Electron
tagged as shower
(0.03, 0.99)

Performance



How to use this method

- Created a Reco Alg to handle all interface between ANN, parameters calc, and so on.

Feed in ANN file name,
and conversions to CM.

Then can call methods
isTrack(cluster, hitlist)

or

**isShower(cluster,
hitlist)**

```
class TrackShowerAlg
{
public:
    TrackShowerAlg();
    TrackShowerAlg(std::string annName, float wireToCM, float timeToCM);
    ~TrackShowerAlg();

    void SetANNFile(std::string);
    void SetConversion(float wireToCM, float timeToCM);
    void Init();

    bool isTrack(art::Ptr<recob::Cluster> &, std::vector<art::Ptr<recob::Hit> > &);
    bool isTrack(std::vector<art::Ptr<recob::Hit> > &, art::Ptr<recob::Cluster> &);
    bool isShower(art::Ptr<recob::Cluster> &, std::vector<art::Ptr<recob::Hit> > &);
    bool isShower(std::vector<art::Ptr<recob::Hit> > &, art::Ptr<recob::Cluster> &);

private:
    // Object that calculates parameters:
    FANNParamsAlg fann_params_alg;
    // Object that is the neural network
    struct fann * ann;

    std::string FANN_File_Name;
};
```

Downsides to this Method

- Need to train algorithm -> No MC truth in data
 - Going to need to do a handscan and manually identify tracks and showers on real data.
 - I am going to implement a tool for this in LArLight, because it is very fast and has a good viewer for individual clusters.
 - The manual scan will also probably improve the results.
 - Physics reason: Won't get tricked by electrons that start out like tracks, or tracks that have sections that look like showers
 - Machine Learning Reason: Training on a batch file can have pitfalls like local (and not global) minimums in chi-sq.
- If upstream algorithms change, probably need to retrain the neural network.
- Would need to add another external library (but I don't think that should really be a problem)

Thanks for listening.
Questions?