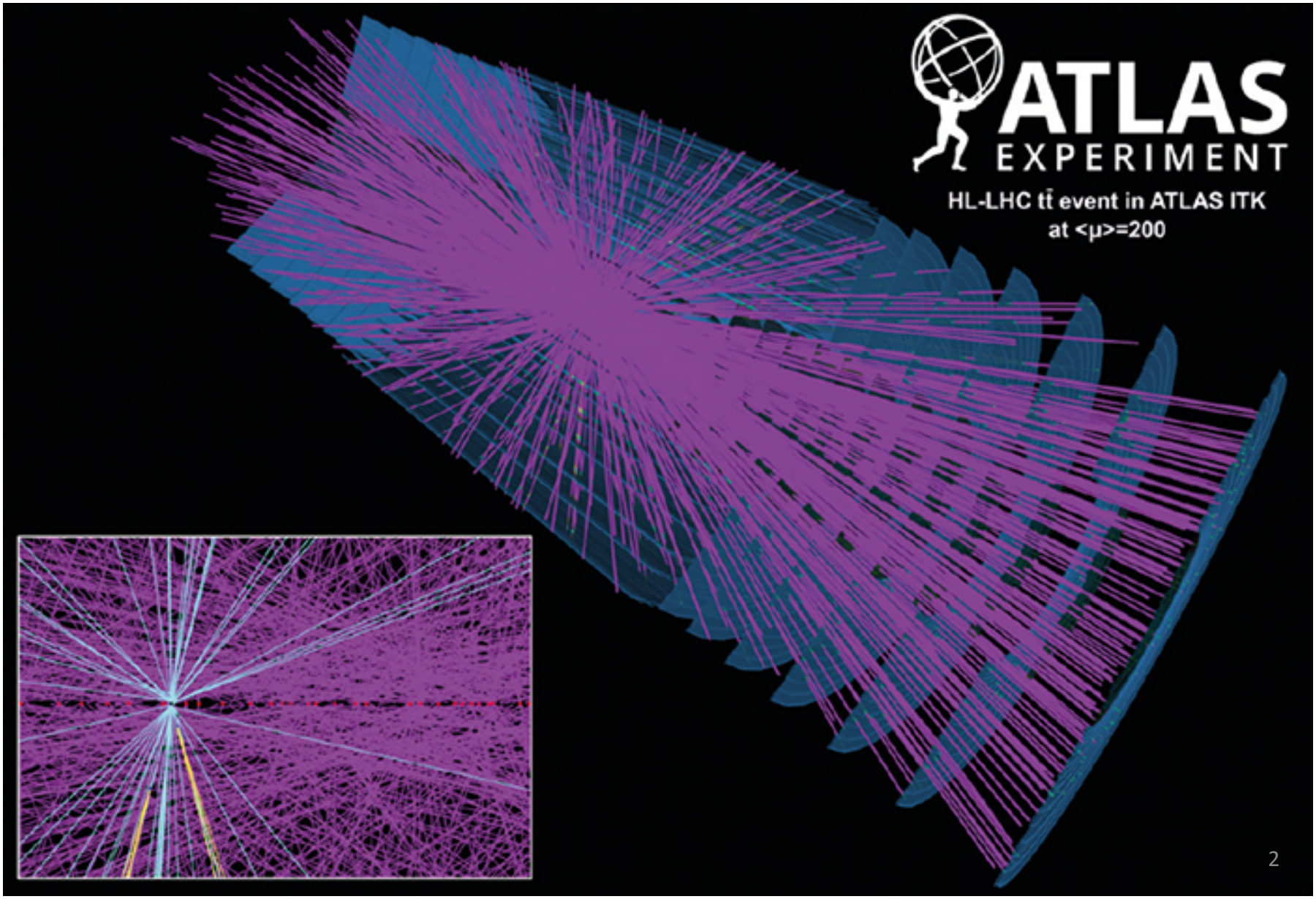
Track Reconstruction for a High-Luminosity LHC Detector using Deep Neural Networks

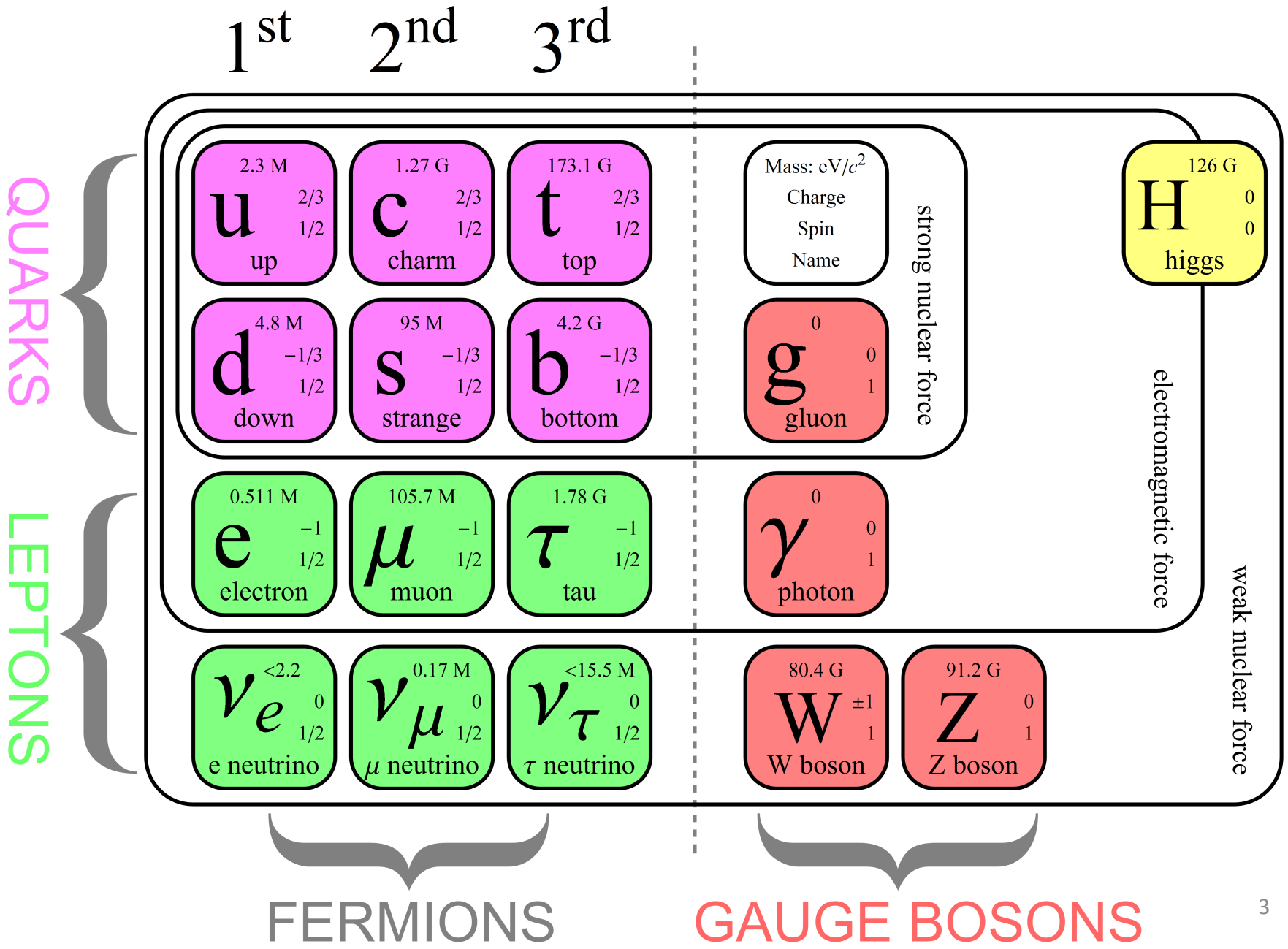
Luiza Adelina Ciucu
University of Geneva

Master Thesis Defence
March 2021

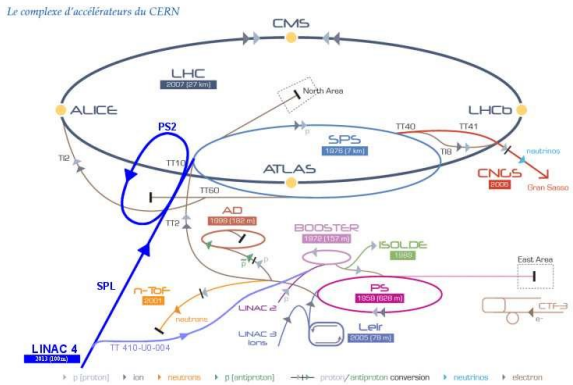
Very complex combinatorics for track reconstruction at HL-LHC.

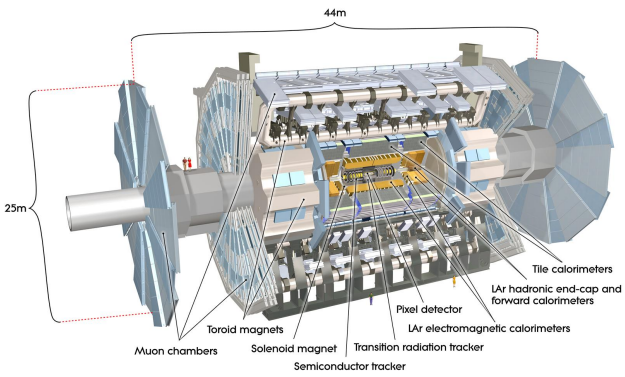


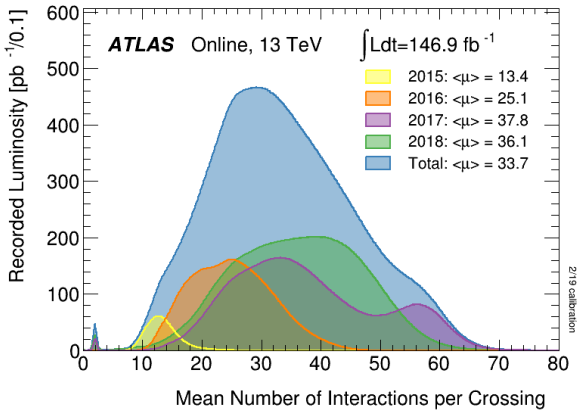
The Standard Model studies elementary particles & their interactions.



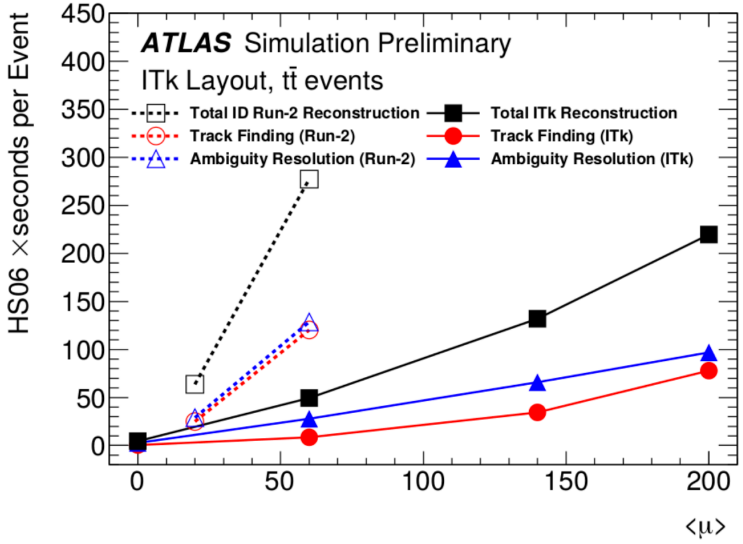
LHC experiments have typically 35 collisions per bunch crossing (mu=35), but plan to go to mu=200 for HL-LHC.

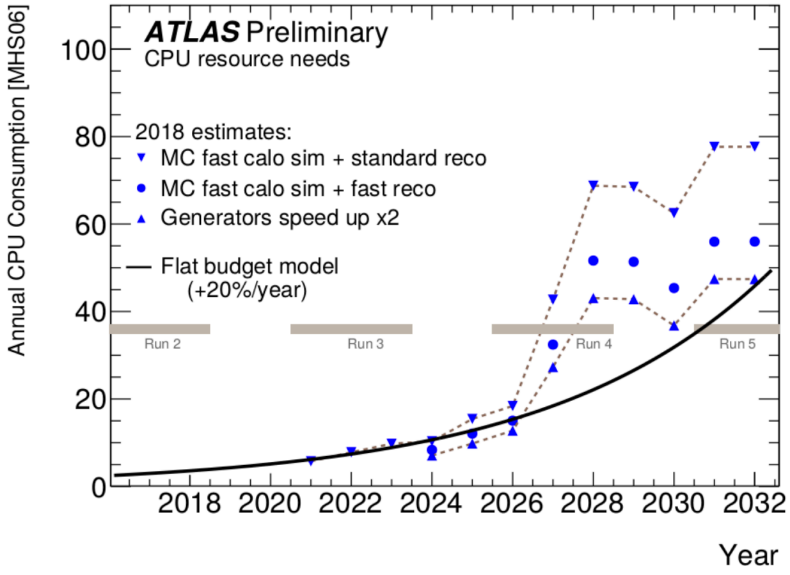






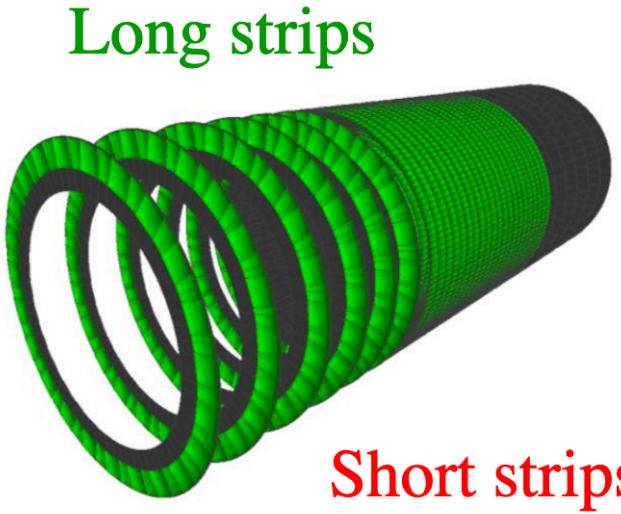
At mu=200 track reconstruction needs CPU beyond LHC resources.



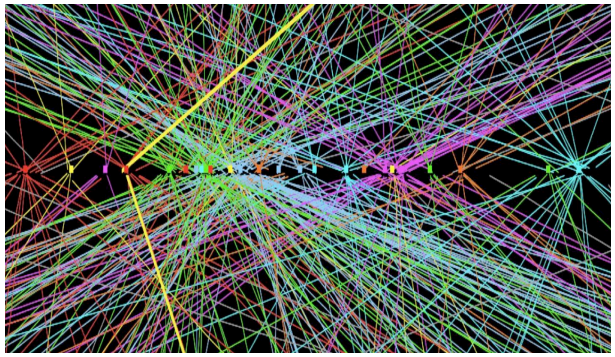


A potential solution is to use new techniques like machine learning.

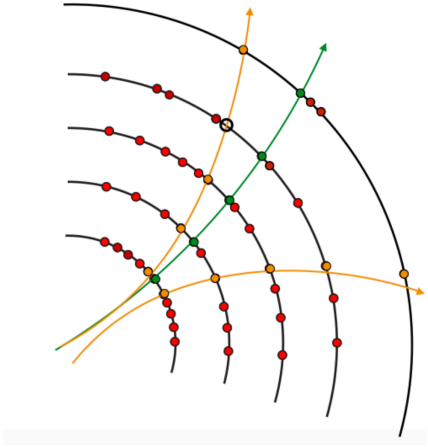
The TrackML challenge offers simulated ttbar collisions at mu=200.
From hits in the inner detector are reconstr. primary vertices & tracks.



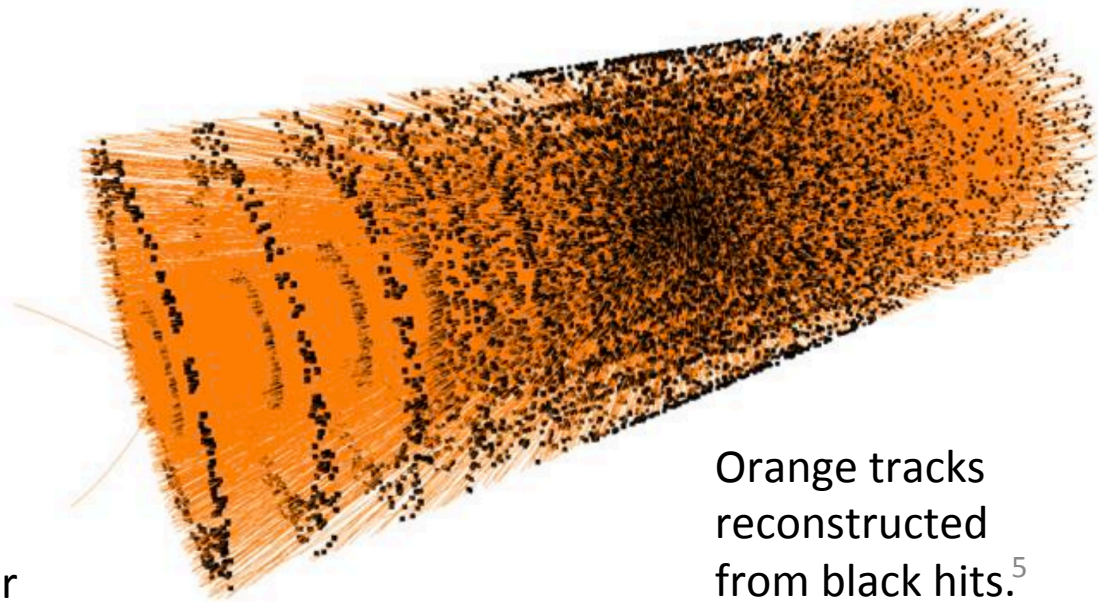
3 silicon detectors in the inner detector



Vertex reconstruction



Track reconstruction

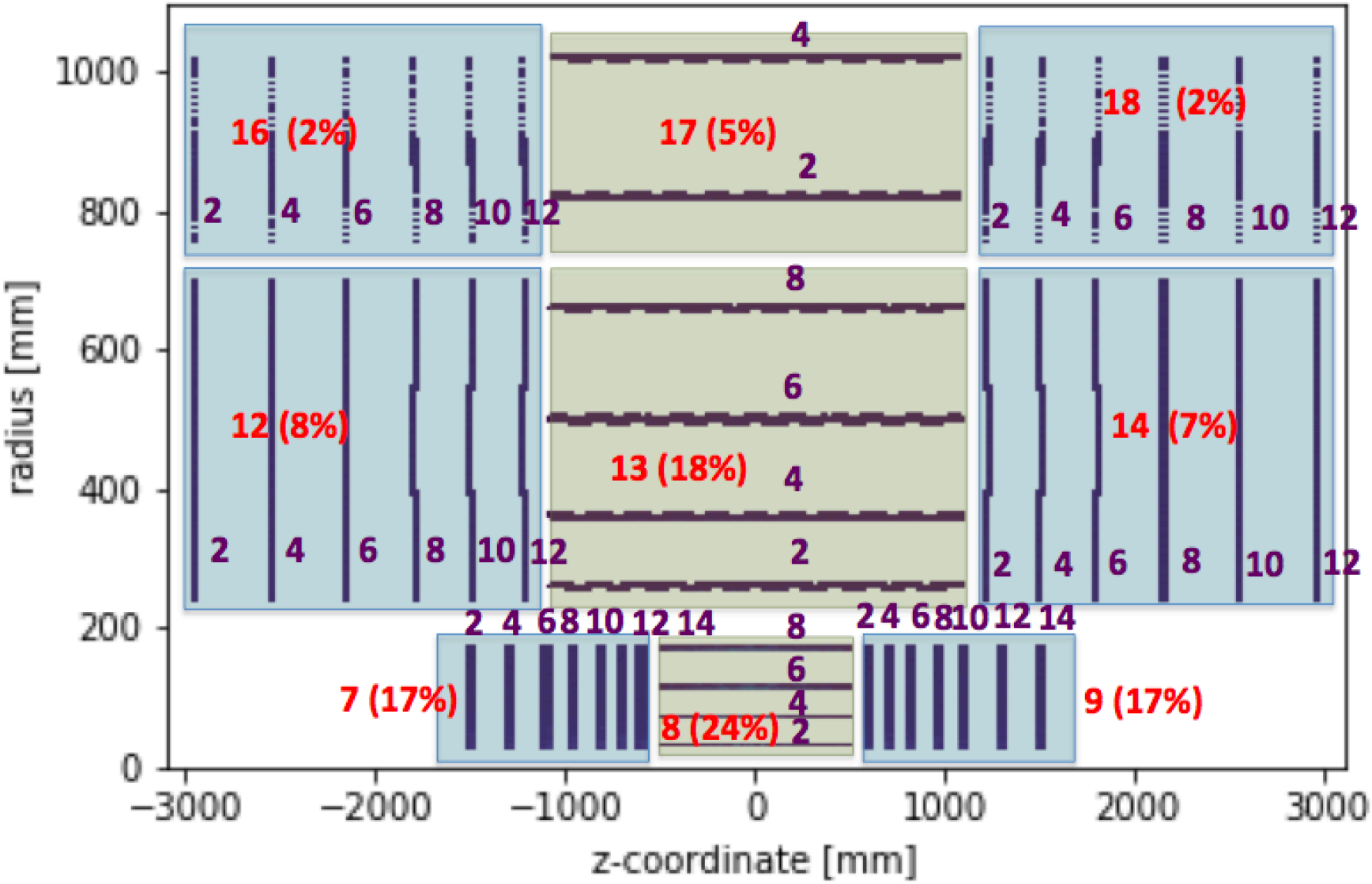


Orange tracks reconstructed from black hits.

Barrel (with horizontal layers). End caps (with vertical layers).

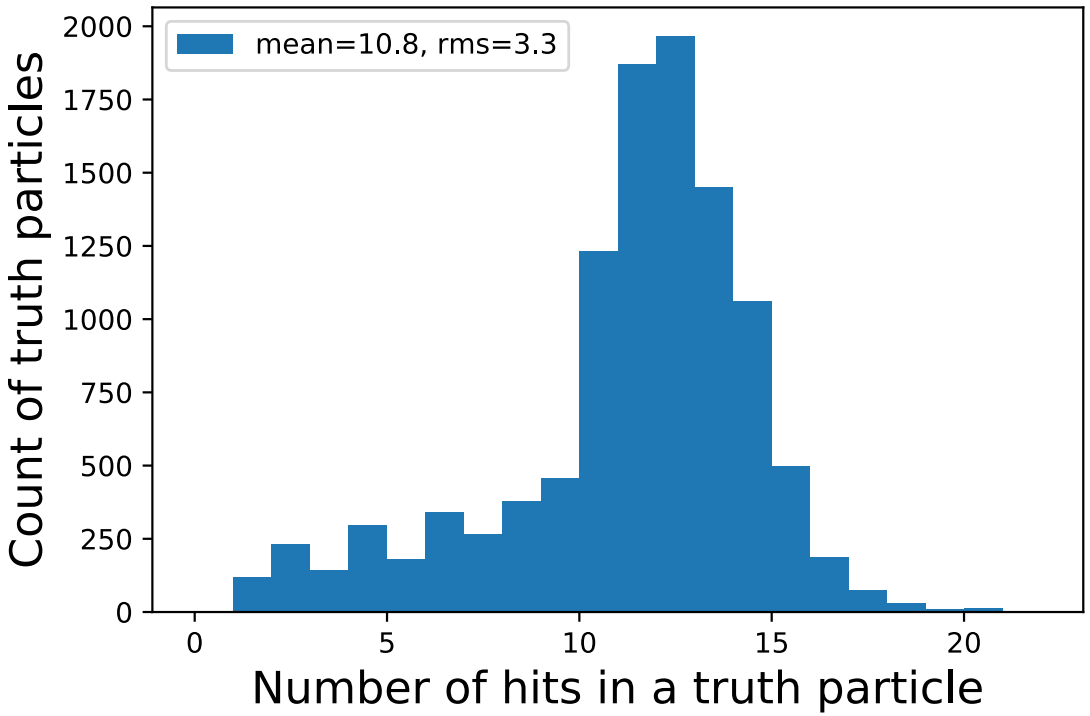
The detector's volume_id in red, and the layer_id in violet.

Percentage of hits in a volume_id shown in parentheses.

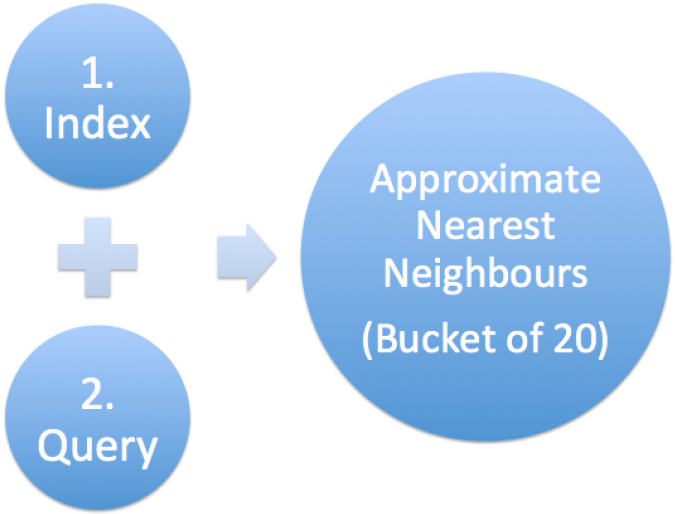


Distribution of number of hits in truth particles.
Reconstructed hits into buckets using approximate nearest neighbors.

Truth level



Reconstructed level

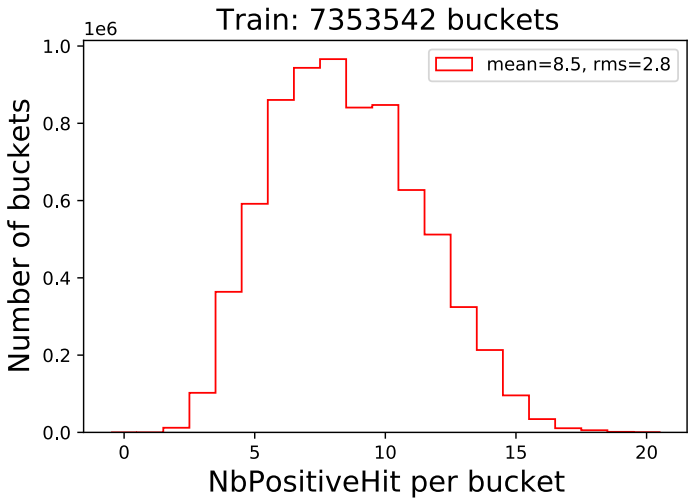


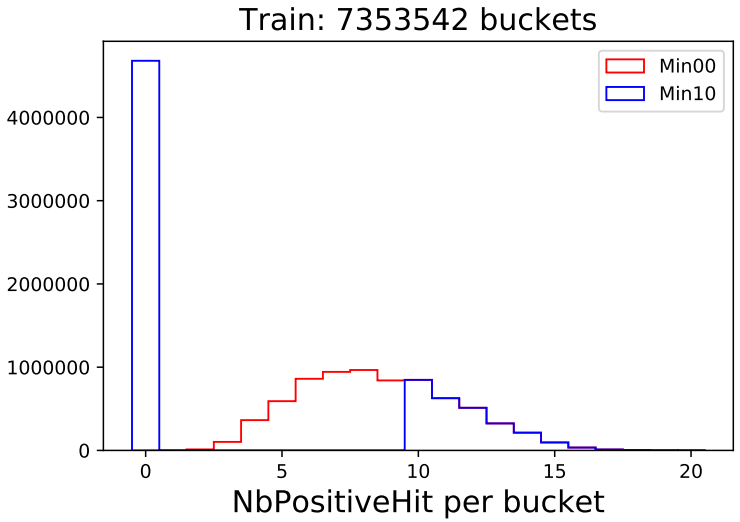
An average truth particle as 11 hits. All particles have less than 20 hits.

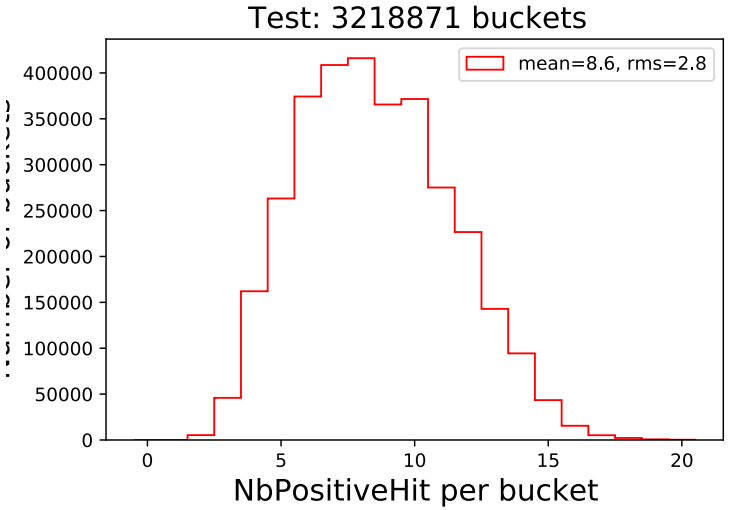
We aim to reconstruct particle tracks from groups of 20 hits close to a straight line (bucket).

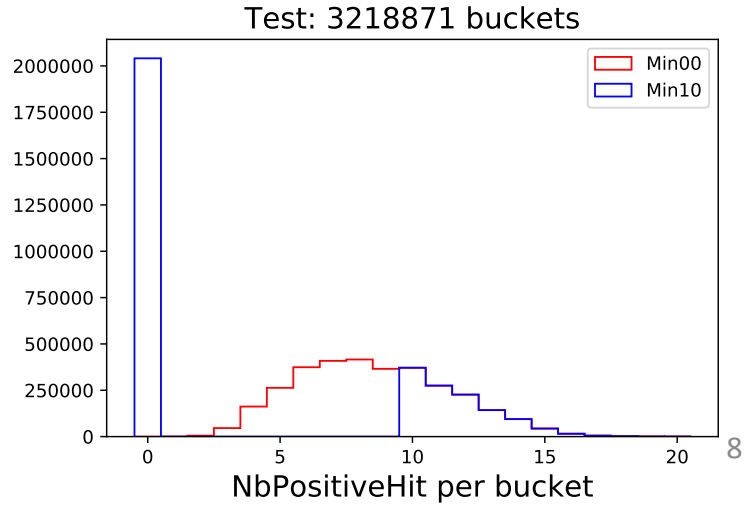
Buckets with 1-9 number of positive hits are artificially set to 0 positive hits. Reason is that we search for particles with at least 10 hits.

Majority particle = truth particle with the largest number of hits in the bucket
Positive hit = 1 = hit belonging to the majority particle
Negative hit = -1 = hit not belonging to the majority particle
Y coordinate = number of hits (histogram)









Problem to solve: Given a bucket of 20 hits, identify those hits belonging to the particle with the largest number of hits in the bucket (majority particle).

Multi-label

- Several questions simultaneously (one for each hit)

Classification

- The questions have categorical answers (yes or no)

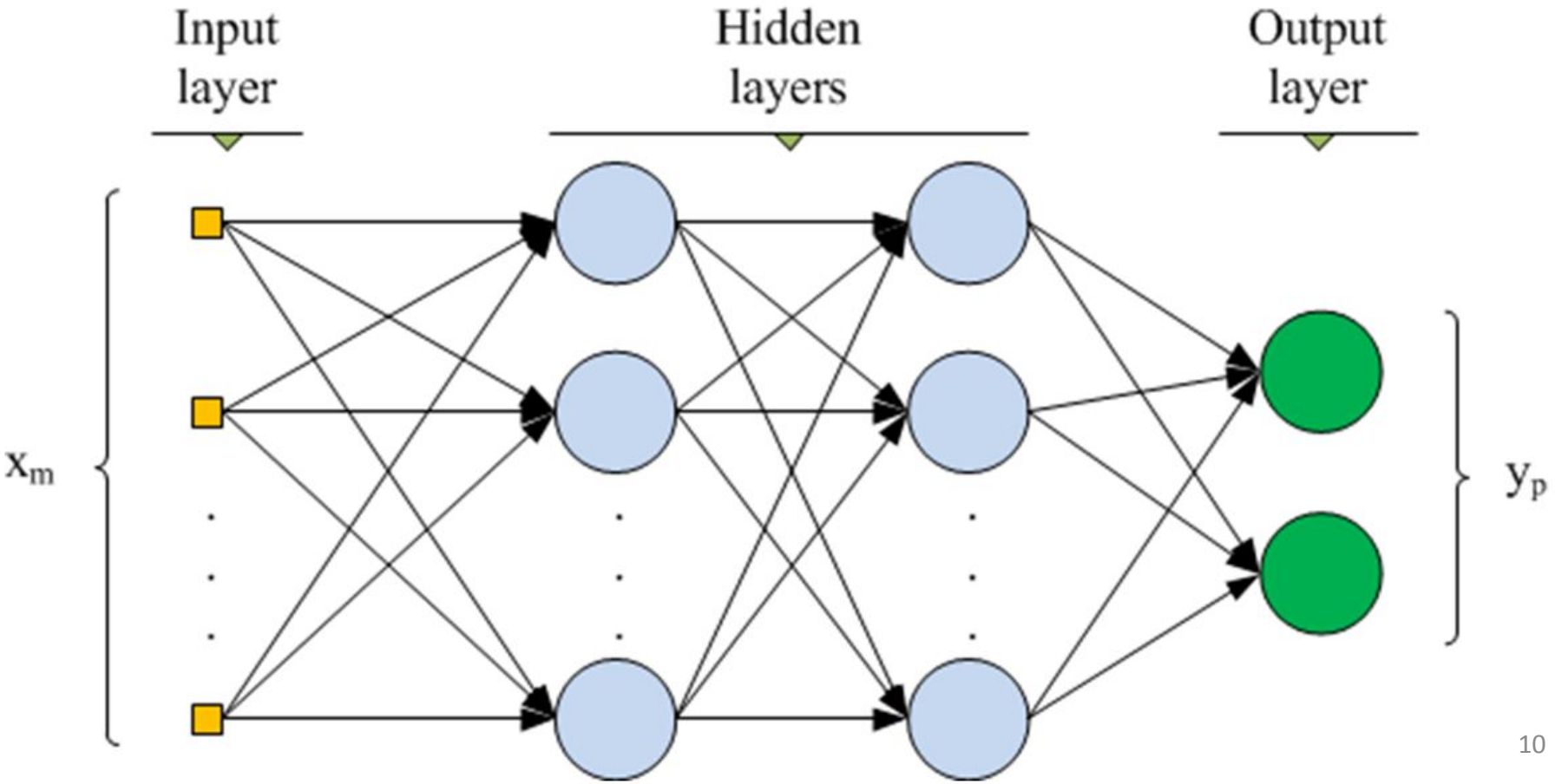
Binary

- Only two categorical labels: yes or no (1 or -1)

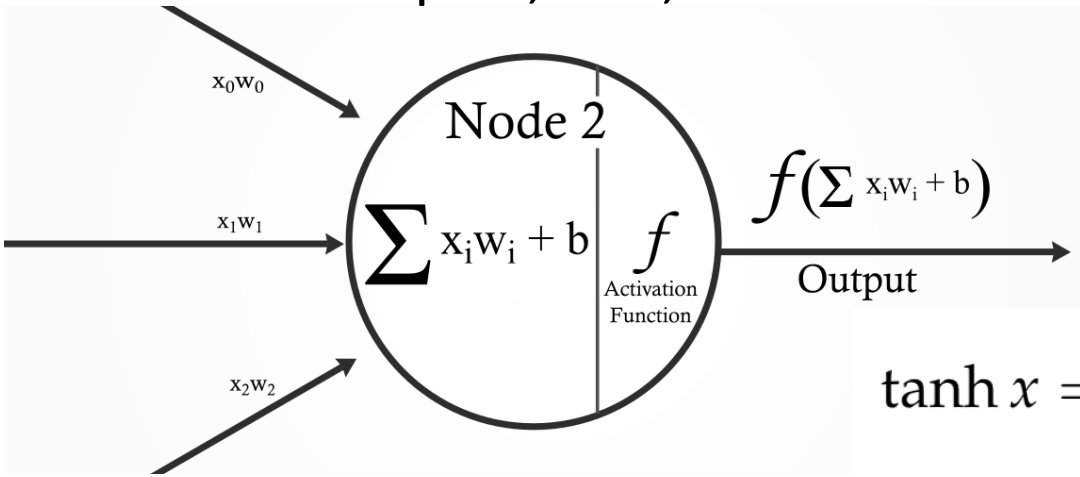
For a bucket input and out fixed: Input = 20x3 (x,y,z). Output = 20x1.
This problem can be answered via machine learning (ML),
In particular deep neural networks (DNN).

The question we try to answer: find the function using a deep neural network (DNN) that for each bucket (of 20 hits) has the following input and output structure:

input	(x,y,z)_1	(x,y,z)_2	(x,y,z)_3	…	(x,y,z)_18	(x,y,z)_19	(x,y,z)_20	60 elem
Output	1	-1	1	…	-1	1	1	20 elem

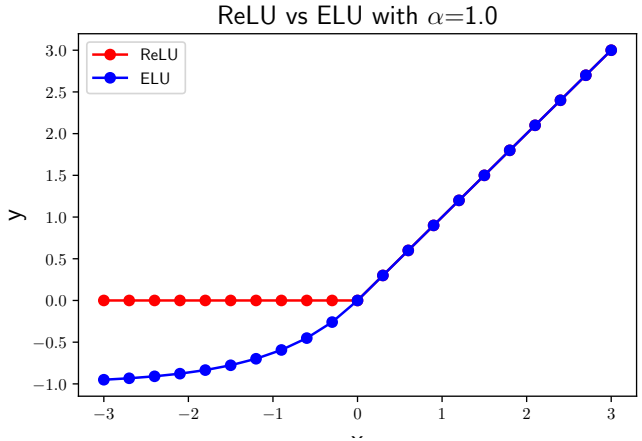


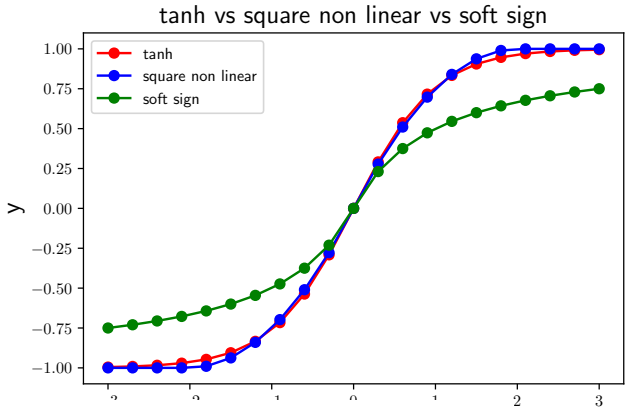
One neuron: inputs, bias, activation function, output



$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Activation functions for the hidden layers and for the last layer.





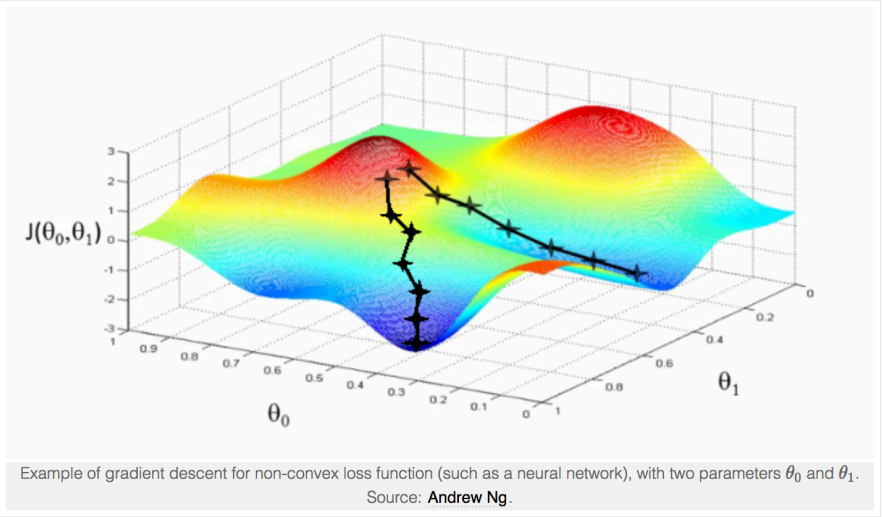
$$\text{ReLU}: \; R(x) = \max(0, x)$$

$$\text{ELU}: \; E(x = \begin{cases} \alpha(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases}$$

$$\text{SQNL}\,(x) = \begin{cases} -1, & x < -2.0 \\ x + \frac{x^2}{4}, & -2.0 \leq x < 0 \\ x - \frac{x^2}{4}, & 0 \leq x < 2.0 \\ 1, & x > 2.0 \end{cases}$$

$$\text{SOSI}\,(x) = \frac{x}{1 + |x|}$$

A NN learns by minimizing the loss function via gradient descent and back propagation.



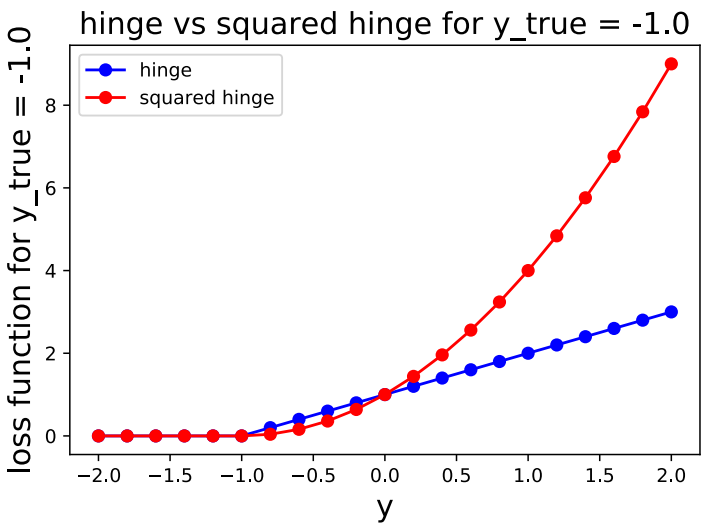
Example of gradient descent for non-convex loss function (such as a neural network), with two parameters θ_0 and θ_1.
Source: Andrew Ng.

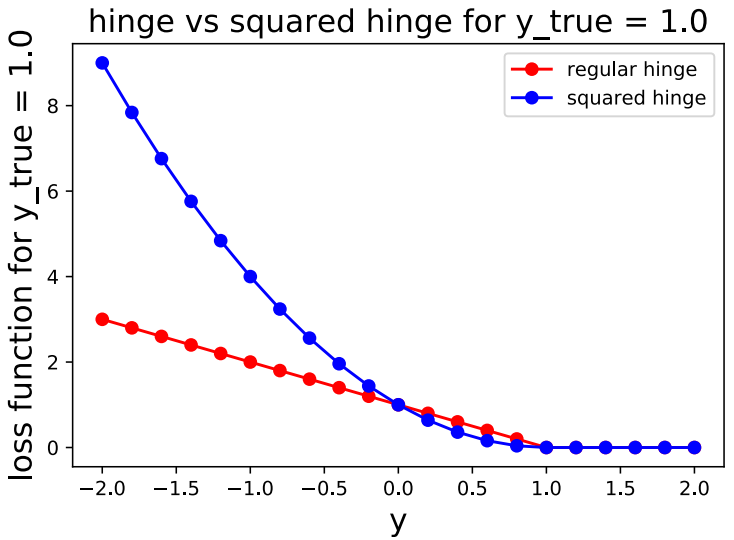
Like water falling off a mountain
at every turn on the steepest slope.

Partial derivative of the loss
function w.r.t. weights and biases,
Using optimisers like Adam & Adadelta.

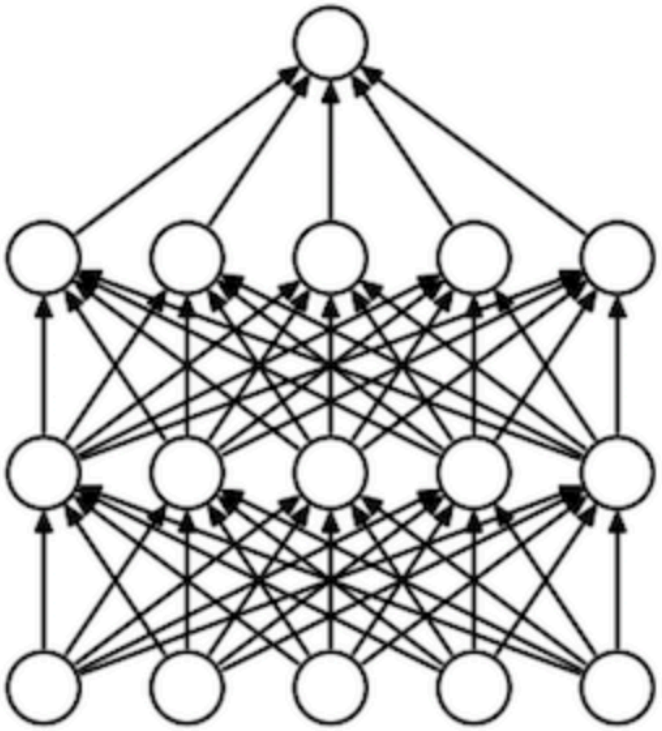
$$\text{Loss function hinge : } l(y) = \max(0, 1 - t \cdot y)$$

$$\text{Loss function squared hinge : } l(y) = [\max(0, 1 - t \cdot y)]^2$$

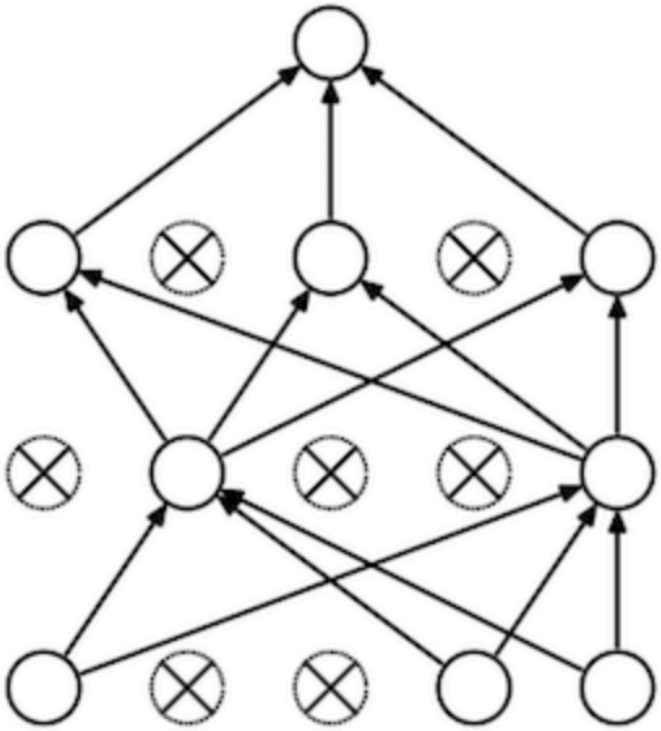




Dropout is a technique during training to drop randomly some percentage of nodes, to avoid over-fitting. We use it with fraction 20%.



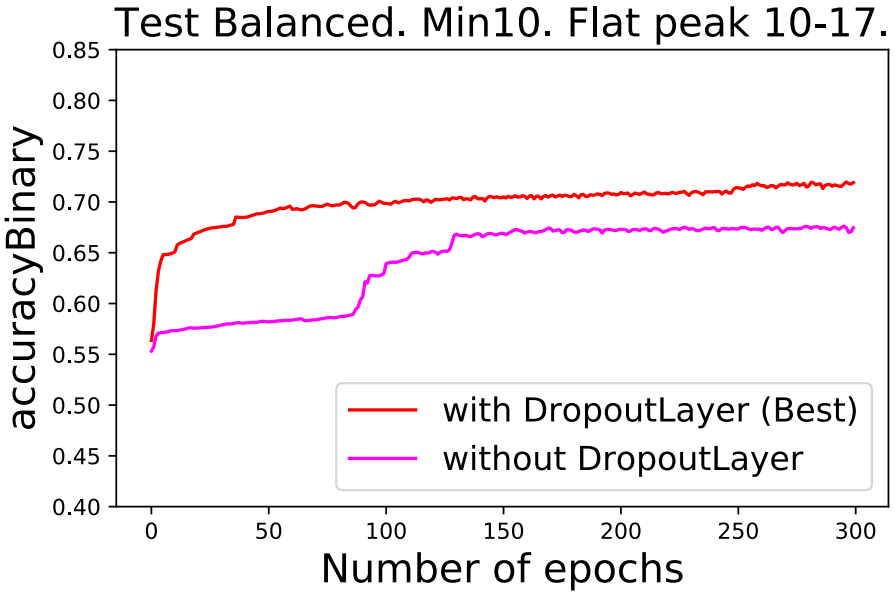
(a) Standard Neural Net

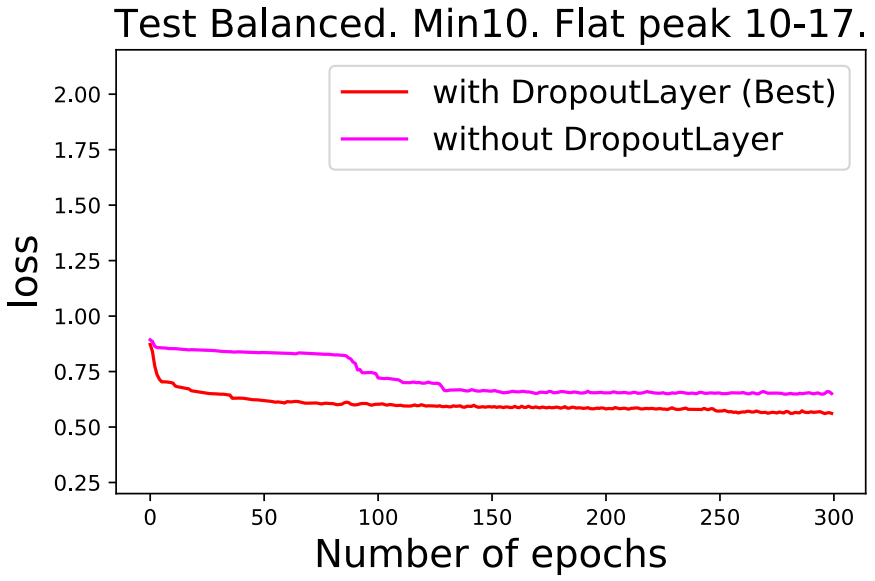


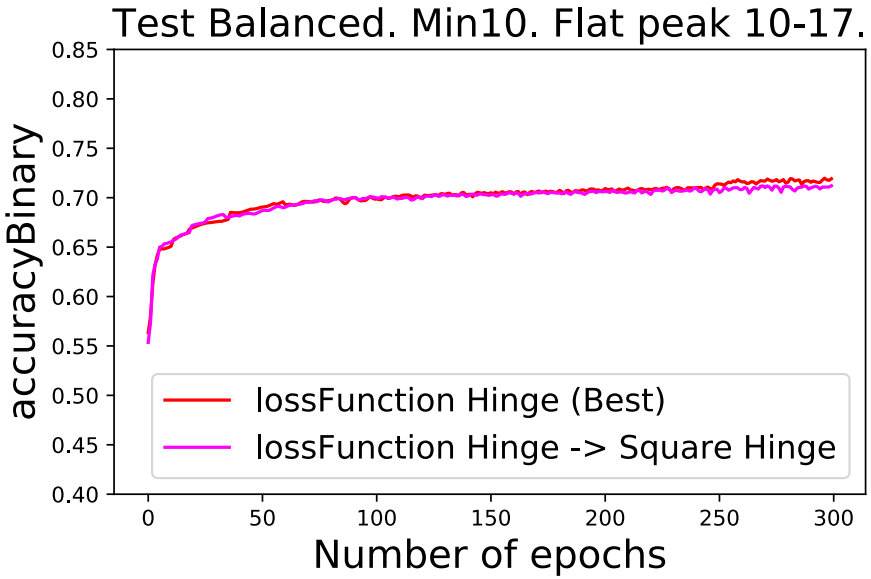
(b) After applying dropout.

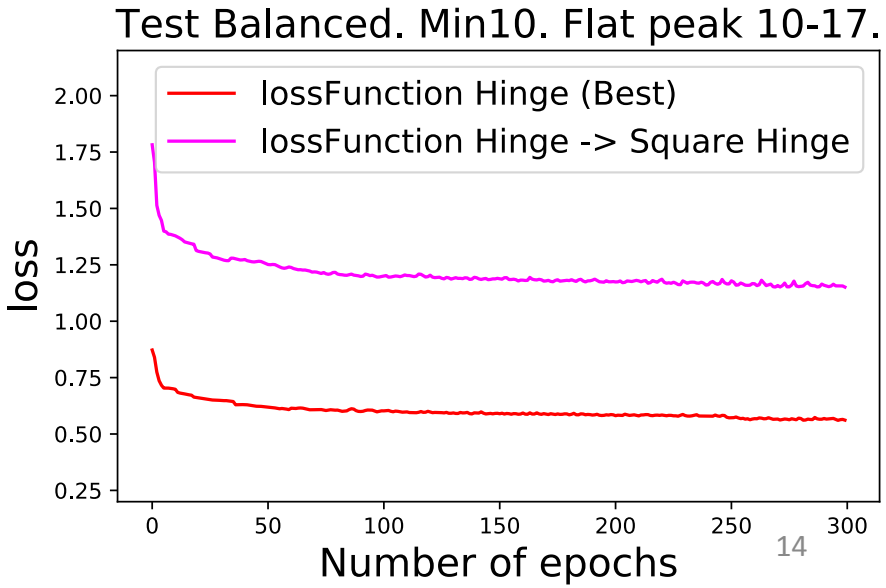
During training, dropout randomly deactivates some neurons as a method for combatting overfitting. Source: Srivastava et al.

Examples of hyper-parameter tuning for dropout layer & loss function.





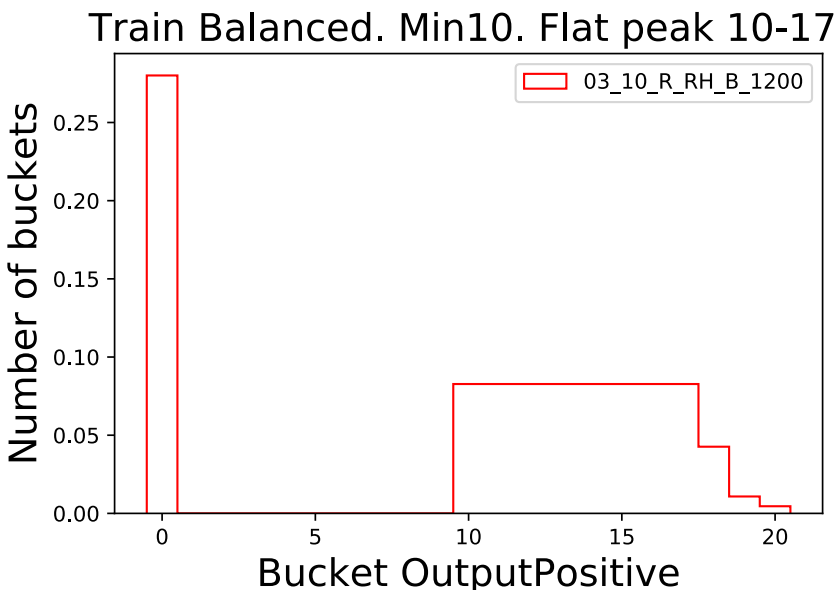


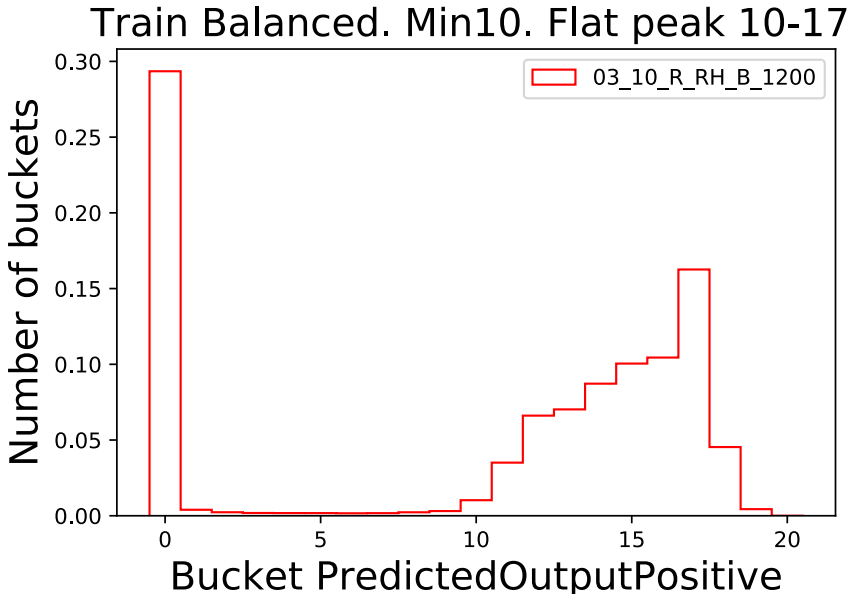


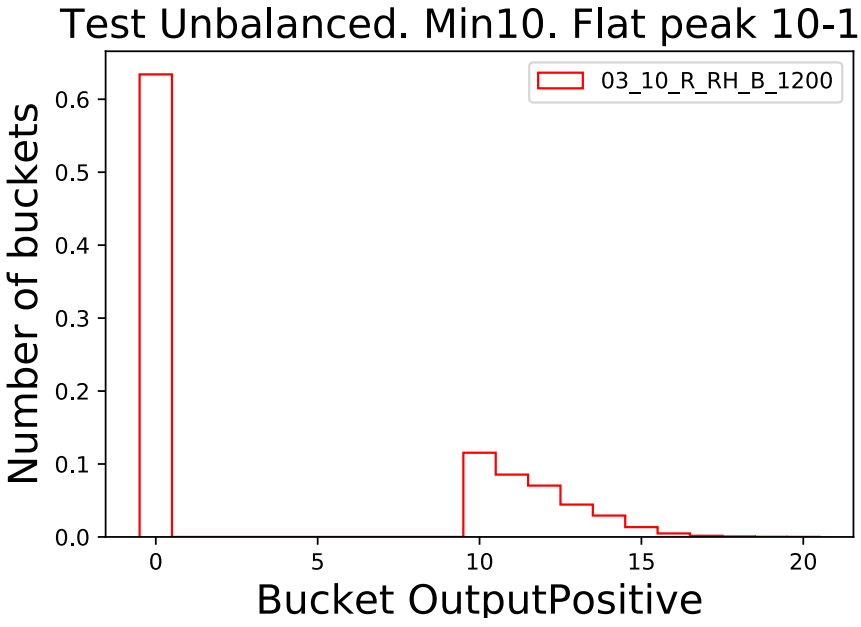
Hyper-parameter tuning and the resulting best model.

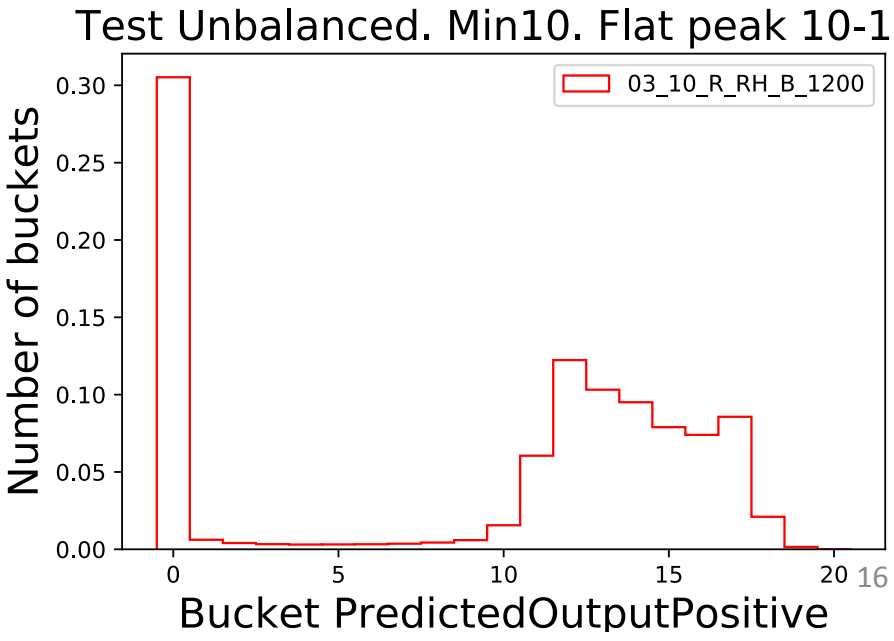
Hyper-parameter	Several choices testted	Choice for best model
Number nodes per input layer	20x(x,y,z)=60	20x(x,y,z)=60
Number nodes per output layer	20	20
Number hidden layer	2, 3, 4	3
Number nodes per hidden layer	20x3, 20x6, 20x10, 20x14	20x10
Activation function input	Linear	Linear
Activation function output	TANH, SQNL, SOSI	TANH
Activation function hidden	eLU, ReLU	ReLU
Dropout layer	no, yes	yes
Loss function	Squared Hinge, Regular Hinge	Regular Hinge
Optimizer	AdaDelta, Adam	Adam
k-fold	50% Train, 50% Test 70% Train, 30% Test	70% Train, 30% Test

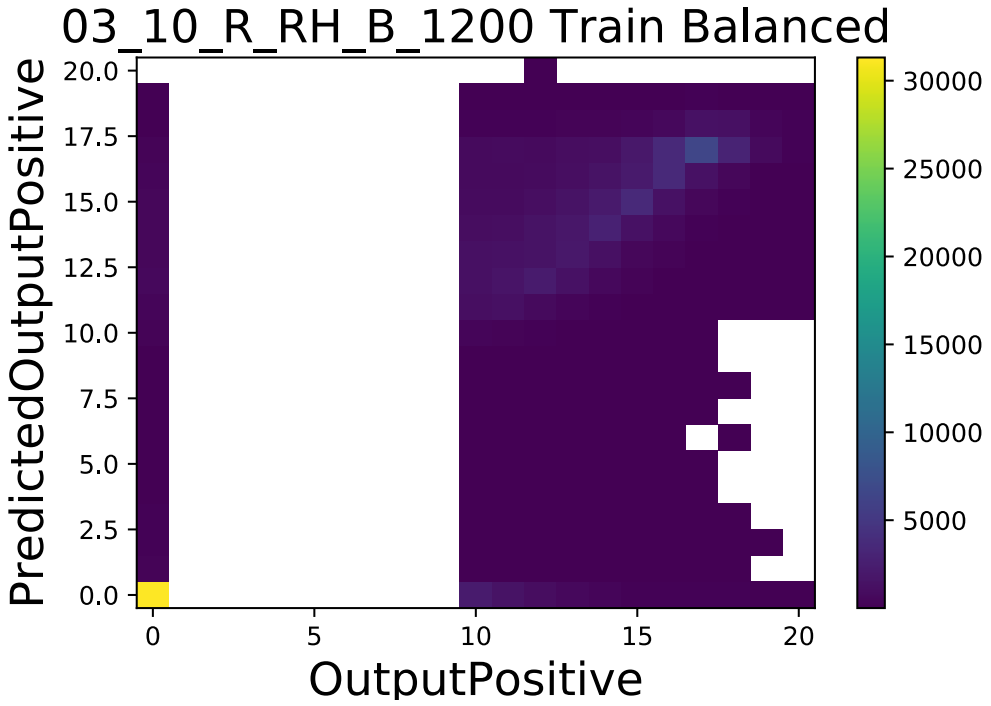
Train in balanced to have equal numbers of positive & negative hits. Set 10-17 to same value of 17, then reduce at 0. Test stays unbalanced.







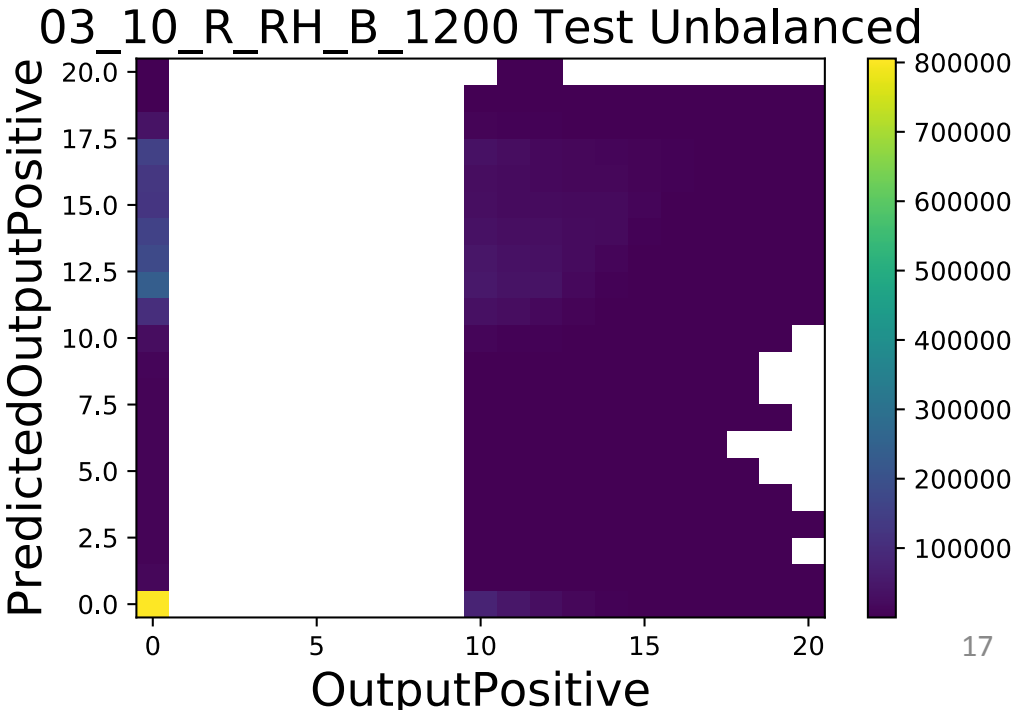




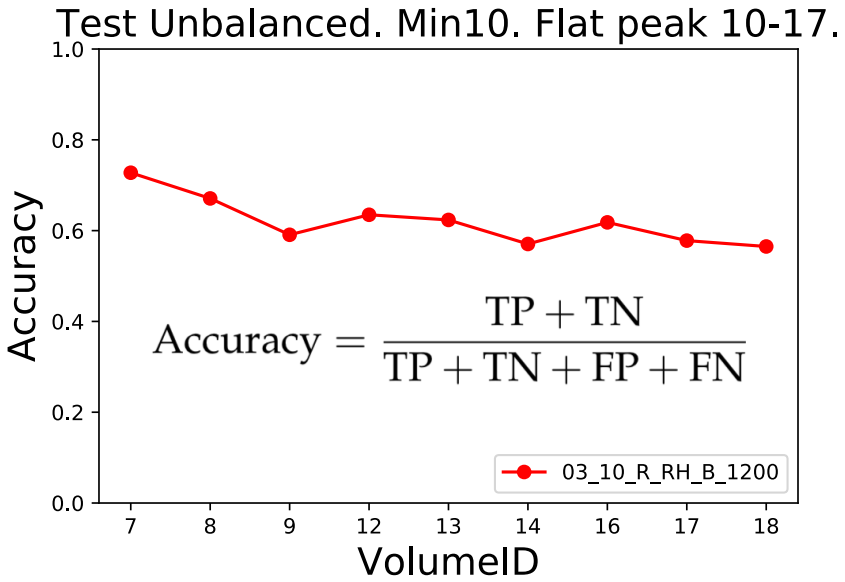
Confusion matrix:
21 x 21

Train balanced:
Peak is along the diagonal.

Test unbalanced
Less clear, somewhat diagonal.



Metrics for hits per volumID: accuracy, precision, recall.

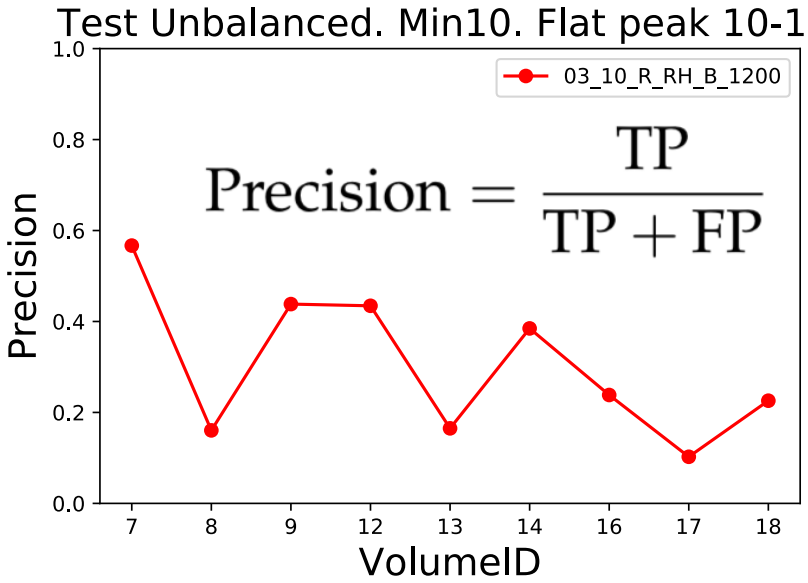


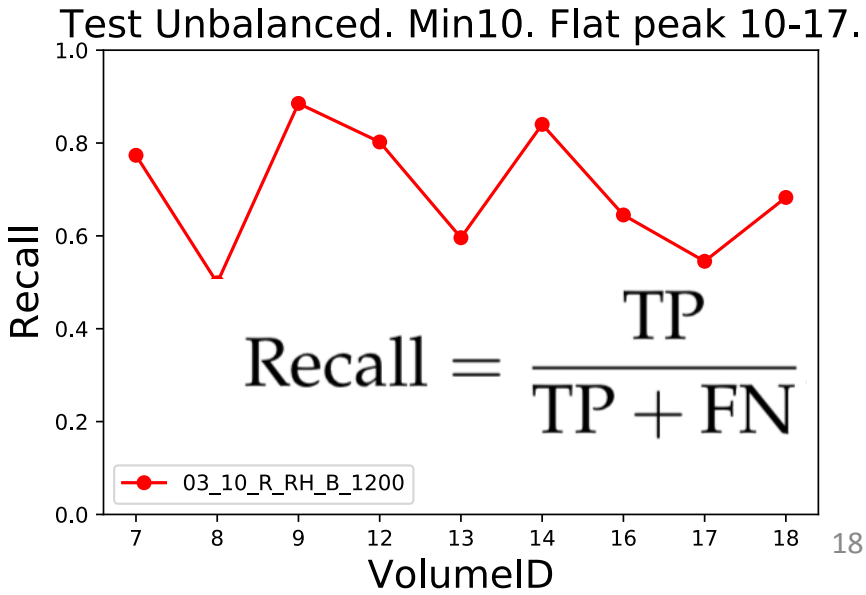
Confusion matrix for one hit.

True Positive (TP)	False Positive (FP)
False Negative (FN)	True Negative (TN)

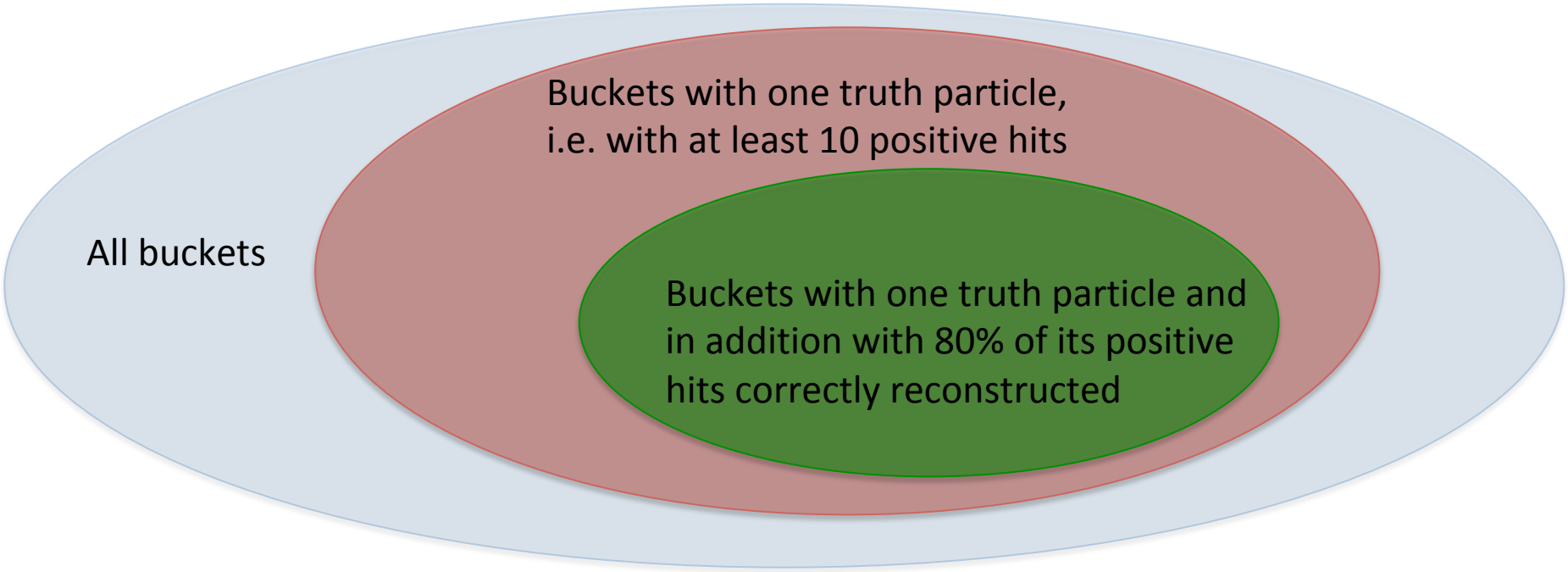
Accuracy not useful when unbalanced dataset.

Then better use precision and recall.
There is a precision-recall trade-off.
Then can not be both max (1) at the same time.





Efficiency of track / particle reconstruction is about 70%.



- nbParticleTruth = subset of buckets with at least 10 positive hits
- nbParticleReco = subset of nbParticleTruth with at least 80% of positive hits reconstructed also as positive hits (nbTruePositiveHits / nbPositveHits > 80%)
- rrack / particle reconstruction efficiency = nbParticleReco / nbParticleTruth

Sample	eff	nbBucket	nbParticleTruth	nbParticleReco
Train Balanced	84.2%	130k	94k	79k
Test Balanced	74.9%	62k	45k	34k
Test Unbalanced	71.3%	3219k	1178k	840k

Conclusion: Results and future plans.

Context

- Tracking is key for charged particle reconstruction in inner detector
- Tracking at pile-up mu=200 at HL-LHC requires more CPU than CERN resources

Problem

- Given an event, create groups of 20 hits called buckets, with hits aligned in the same direction using approximate nearest neighbor methods
- For each hit in a given bucket, predict if it belongs or not to the particle with the largest number of hits (majority particle). Positive hits belong to the majority particle.
- Input = 20 x 3 (x, y, z). Output = 20 x 1 (1 or -1).
- It is a multi-class binary classification problem.

Solution

- Problem solved with machine learning (ML), in particular deep neural network (DNN).
- Events split 70% in train, 30% in test
- Buckets with nbPositiveHit < 10 are set to nbPositiveHit=0
- Train sample leveled 10-17 and then balanced positive and negative hits.
- Hyper-parameters are tuned.

Results

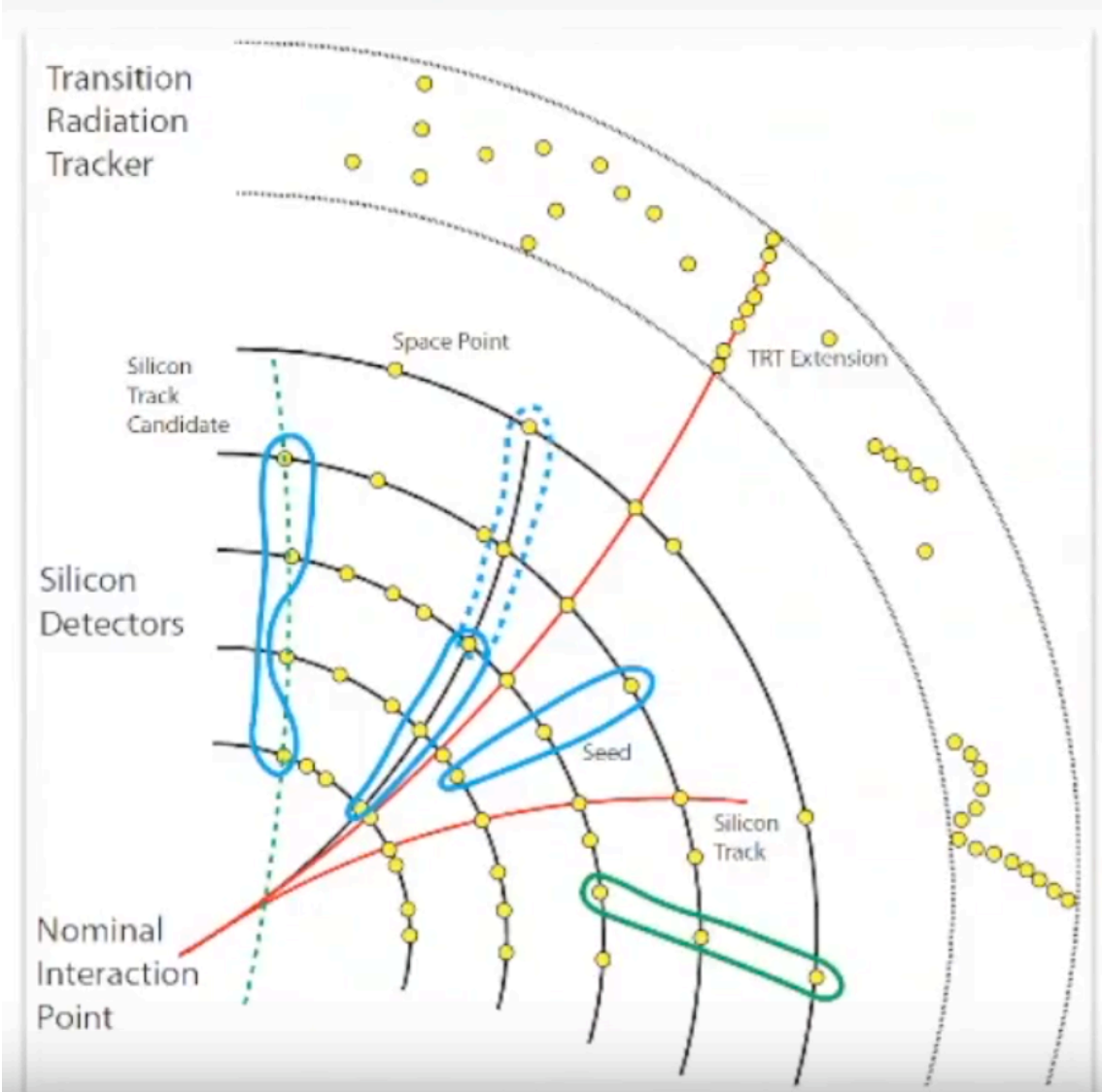
- Hit-level metrics: Accuracy ~ 65% Precision ~ 30%, Recall ~ 70%.
- Track/Particle-level metrics: Reconstruction efficiency ~ 70%.

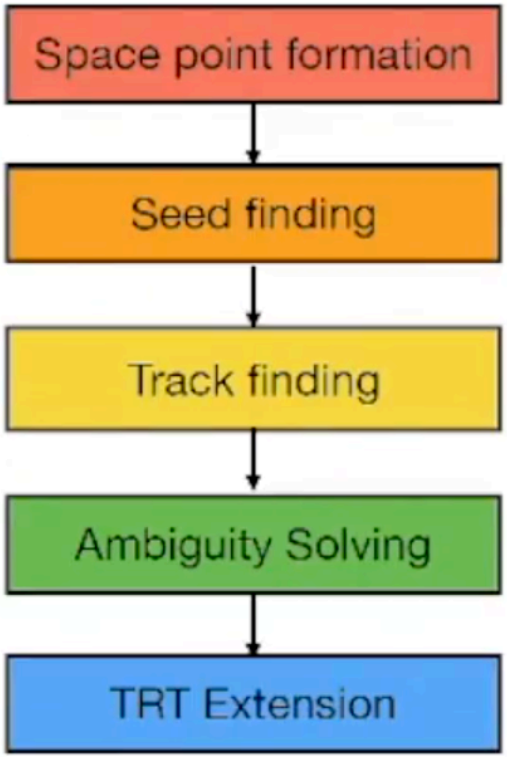
Future Plans

- Improve model by using more events, better hyper-parameter tuning, new models.
- Study performance for various particle directions and momenta within the detector

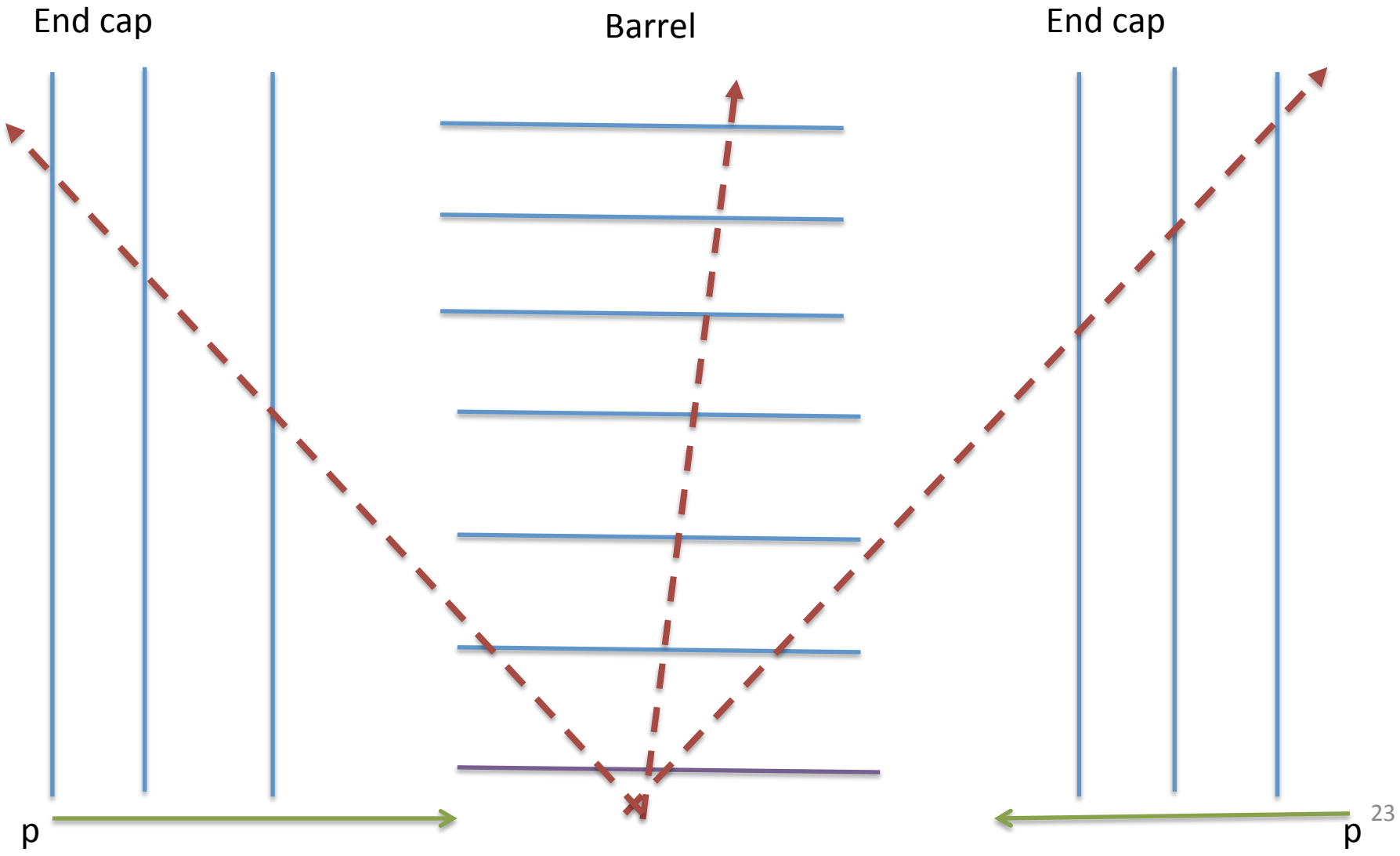
Backup slides

LHC detectors reconstruct and measure the kinematic properties of particles produced in proton-proton collisions. Charged particles ionize matter, producing hits in the various layers of the inner detector. Our task is to reconstruct tracks from hits.





Schematic view of the general inner detector with layers.
Charged particles leave tracks
after starting in the central layer closest to the beam.



NN learning optimizers: Adam vs Adadelta

Adam optimization is a stochastic gradient descent (SGD) method that is based on adaptive estimation of first-order and second-order moments.

- ADAM = Adadelta + Stochastic Gradient Descent (SGD) + Momentum
- AdaDelta = rescales gradients based on accumulated second order information
- Momentum = smoothens gradients based on first order information

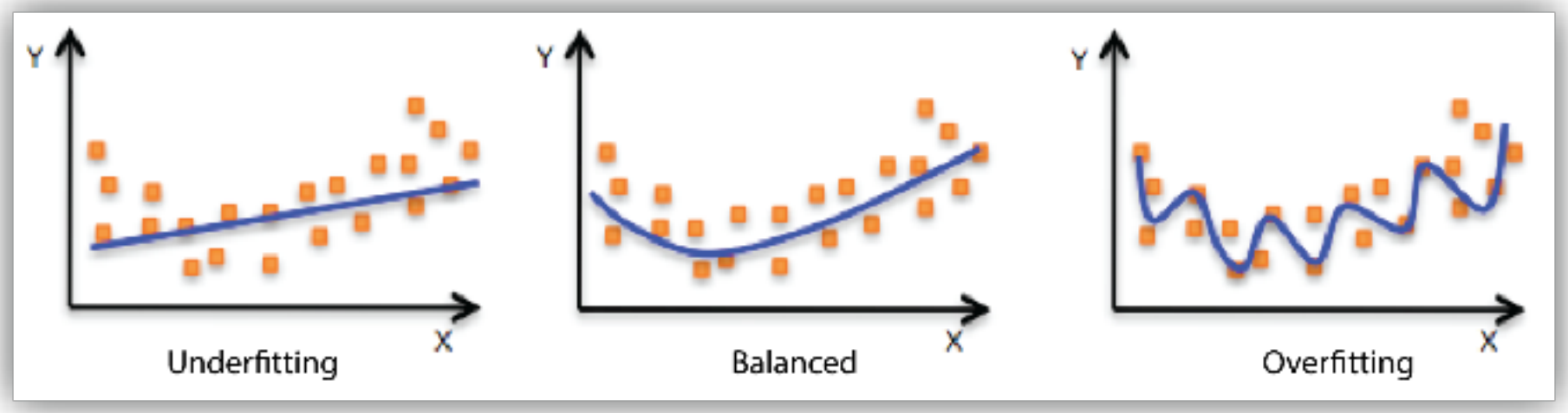
Alpha = 0.001 (learning rate)
Beta_1 = 0.9 (The exponential decay rate for the 1st moment estimates.)
Beta_2 = 0.999 (The exponential decay rate for the 2nd moment estimates.)

Adadelta is a more robust extension of Adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients.

alpha = 0.001 (learning rate)
rho = 0.9 (The exponential decay rate for the 1st moment estimates.)

In this study various learning rates were tried, but no change was observed.
So the default learning rate of 0.001 was used.
Adam was chosen for the best model.

Over-fit (over-train) vs under-fit (under-train)



Under-fit: model performs bad in training data. The model is too simple. Solution:

- Add more relevant features correlated to the target.
- Reduce regularization.

Over-fit: model performs well in training data, but bad in test data. Model memorizes the behavior of training data and is not able to generalize in test to new data slightly changed. Solution is to reduce the model flexibility by:

- Use fewer features
- Use fewer bins per features
- Increase regularization

If both train and test data performance are bad:

- Increase the amount of training data samples
- Increase the number of epochs

Regularization methods to avoid over-fitting

Regularization techniques add to the loss function which modifies the resulting weights. In this project we did not use regularization, but also did not notice over-fitting, as performance in test is similar to the performance in train.

What is minimized actually is the
cost function = loss function + regularization term

Typical loss function: mean squared error (MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

L1 regularization: cost function = MSE + L1 Regularization

$$\mathrm{MSE} + \lambda \sum_{i=1}^{n} |w_i|$$

L2 regularization: cost function = MSE + L2 Regularization

$$\mathrm{MSE} + \lambda \sum_{i=1}^{n} W^2_i$$