

A Boosted kNN Regressor with 66 million parameters

Tommaso Dorigo

INFN, Sezione di Padova

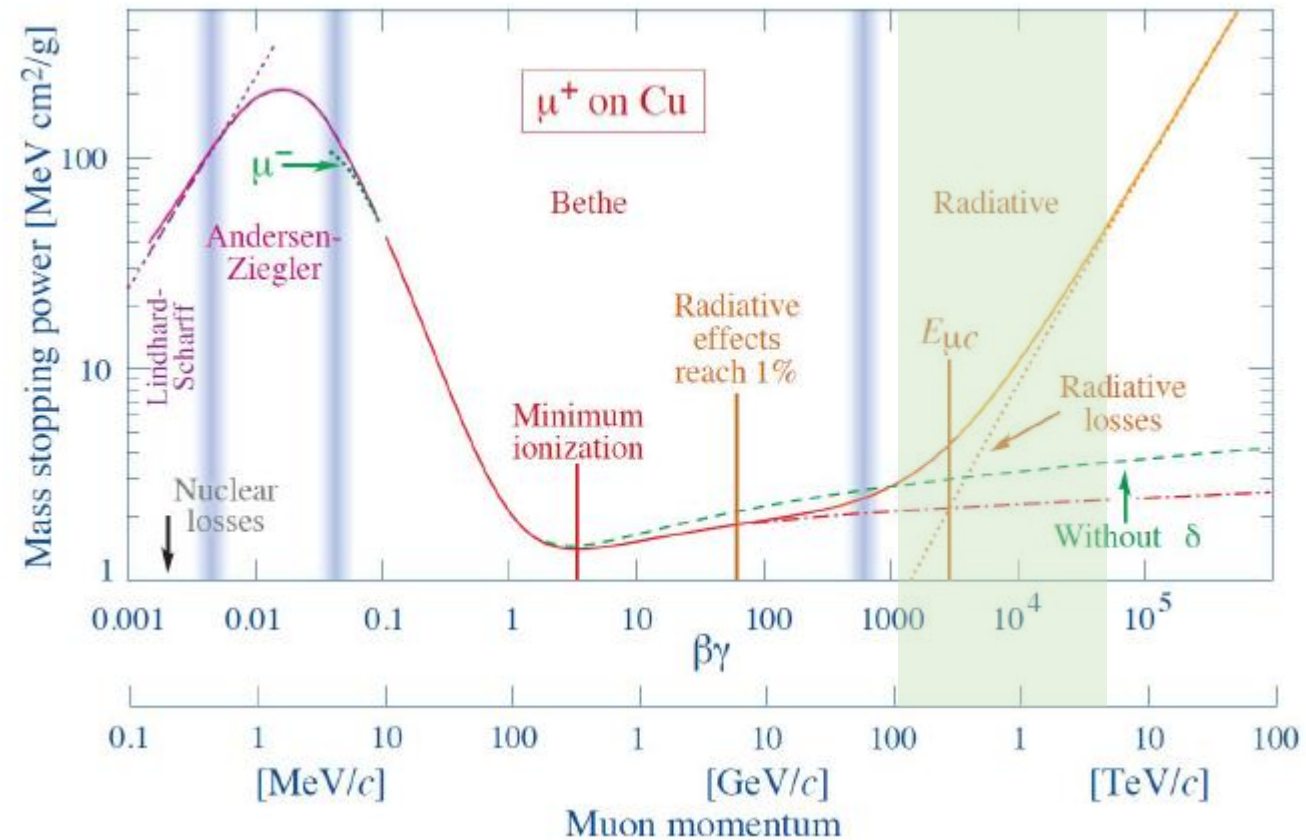
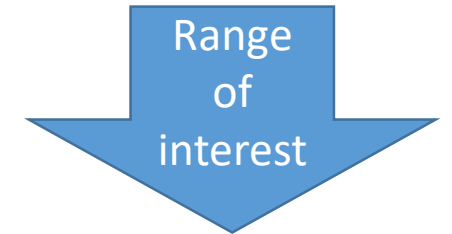


The problem: measuring muon energy in a calorimeter

Muons do not produce EM showers when traversing dense materials

Their behavior changes above a few 100 GeVs, when they start to radiate soft photons in significant amounts

But even then the energy loss in a thick calorimeter is typically of the order of a few percent



What to do above a few TeV?

Future E>E(LHC) colliders might produce new particles whose signature involves decay to ultra-high-energy muons. Yet above a few TeV we cannot rely on magnetic bending to measure their momentum:

- CMS has $\delta E/E = (0.06 : 0.17) E/\text{TeV}$
- ATLAS has $\delta E/E = (0.08 : 0.20) E/\text{TeV}$

→ Above 3-4 TeV muon energy becomes unmeasurable by bending methods.

The questions then are:

- Is there information in the **spatial distribution** of the energy deposits?
- **To what precision** can multi-TeV muon energy be measured in a very high granularity calorimeter?

To answer them, we developed a very complex CNN model

Simulation of a high-granularity calorimeter

GEANT4 → high-statistic samples of muons interacting in a high-granular, homogeneous PbWO_4 calorimeter

total depth = 2032 mm = $10 \lambda_0$

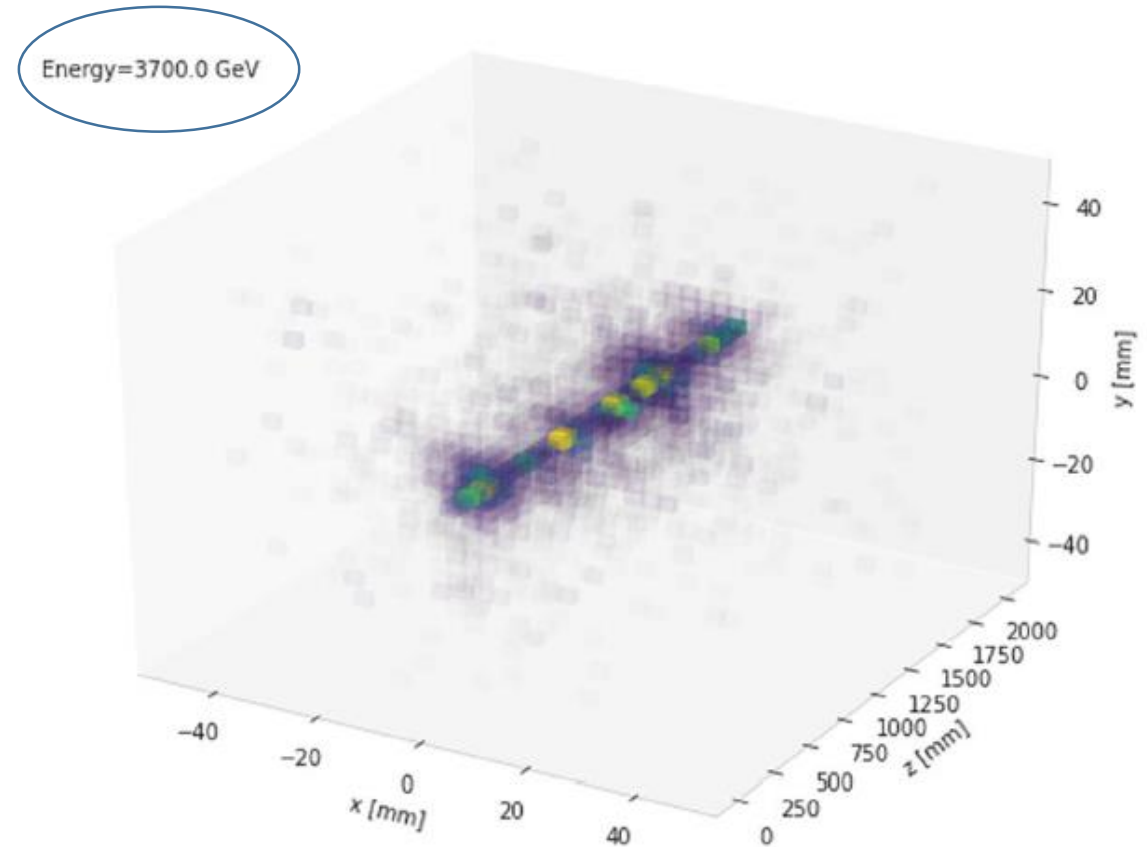
50 layers of 32x32 cells –

51,200 channels in total

cell size = 3.7 x 3.7 x 39.6 mm

We assume the calorimeter is embedded in a 2-Tesla B-field orthogonal to the muon incident direction. This has practically no effect for >1 TeV muon trajectories (<1 mm deflection)

We simulate about 1M muons in the energy range $E = [100 \text{ GeV}: 8000 \text{ GeV}]$

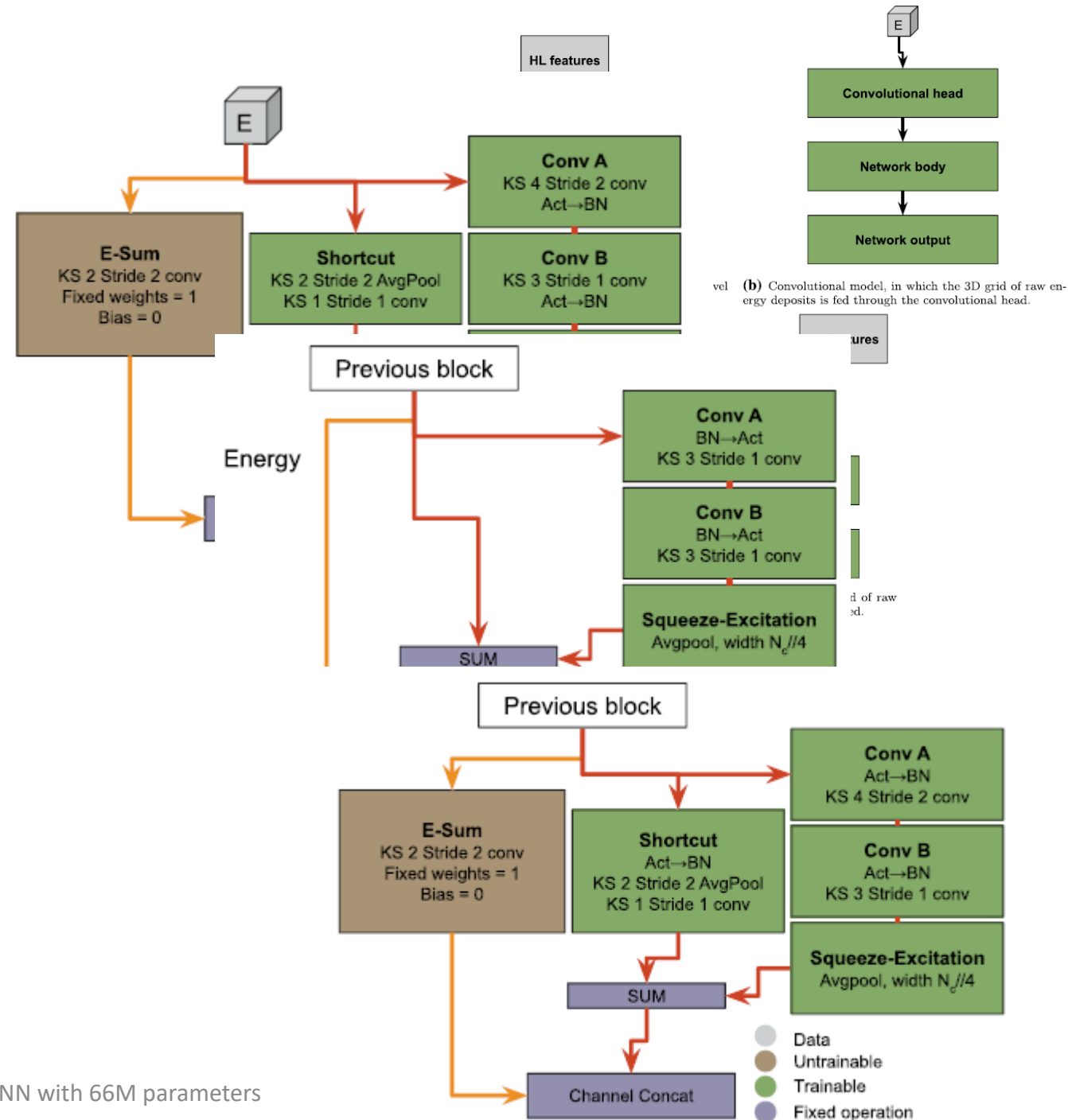


Note: this looks like a shower, but most photon deposits shown are of few MeV only

CNN model

The CNN we originally deployed uses convolutional layers to reduce the dimensionality of the pattern of energy deposits, and employs 28 high-level features in a dense layer.

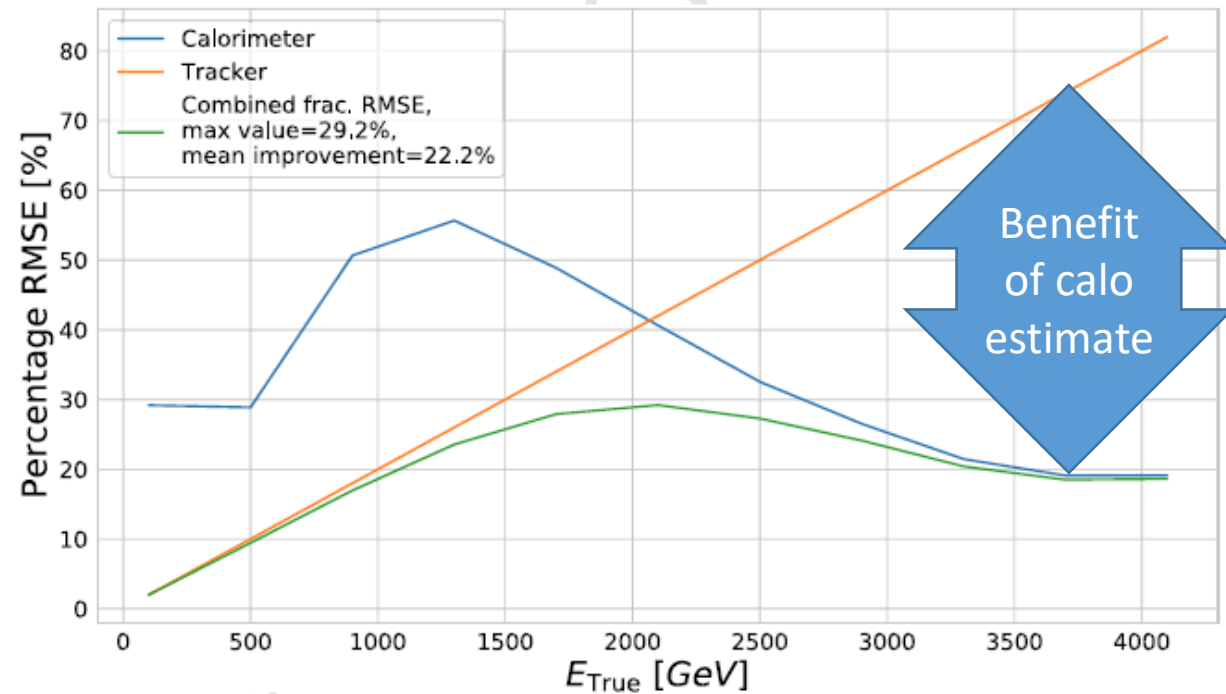
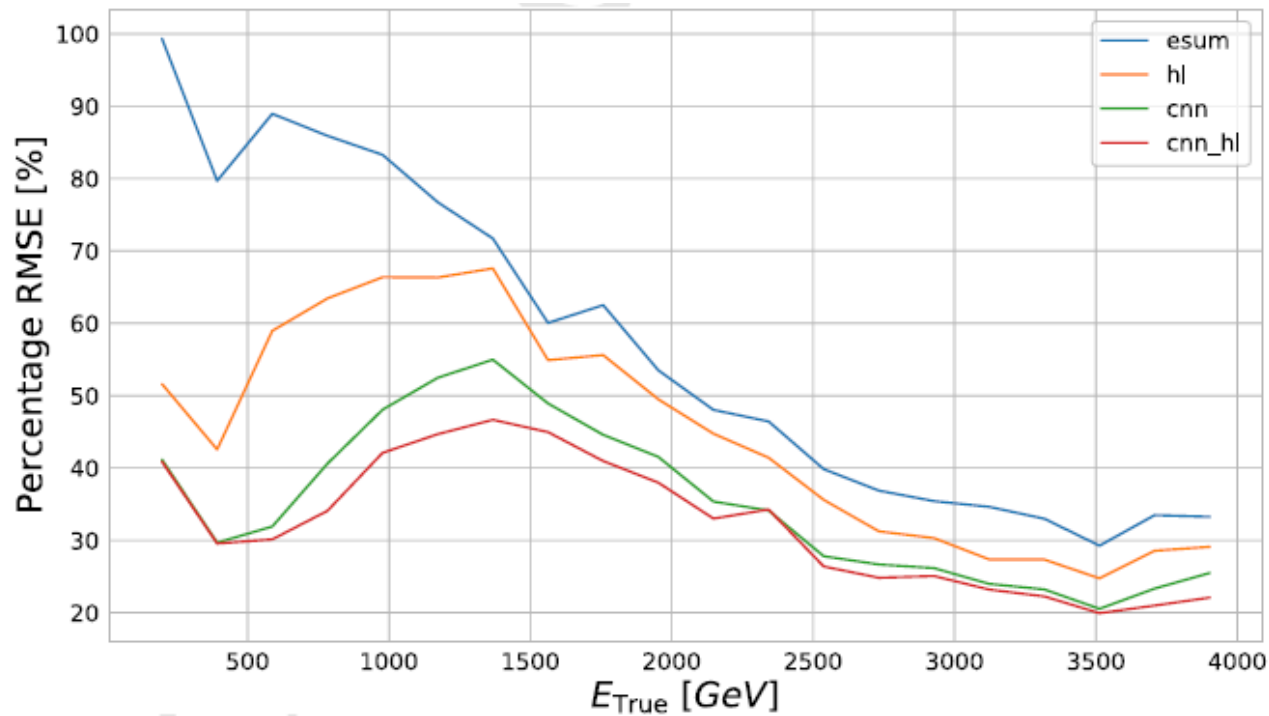
The 28 event features are the result of human-based «domain knowledge» about possible ways to aggregate the information on the spatial distribution of energy deposits (more later)



CNN results

The CNN model manages to recover 20% resolution for 4-TeV muons. It also demonstrates how **there is information in the spatial distribution** of photon deposits. A combination with a tracking measurement ($\delta E/E=0.2E/\text{TeV}$) is shown on the right.

Results are published in [J. Kieseler et al., Eur. Phys. Journ. C82 \(2022\) 79, arXiv:2107.02119](#)



Why a k-NN then?

The CNN results shown were actually produced several months *after* we first tried a k-NN algorithm on the problem, because I had some code handy and needed to produce a preprint in time for a funding application...

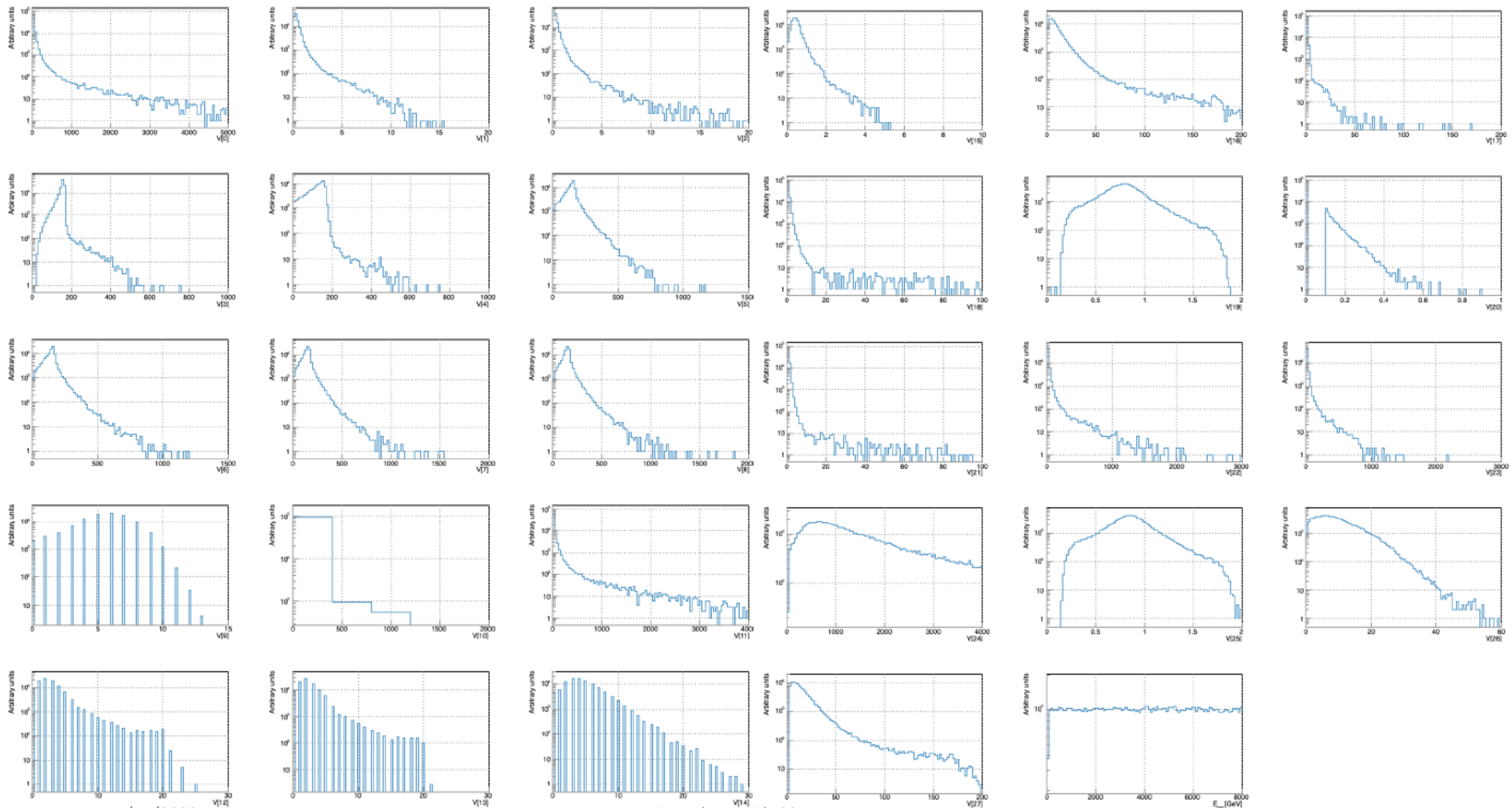
The k-NN was eventually re-run on the full dataset used for the CNN publication, after many improvements

Here I am describing the [final version of the k-NN](#) we developed.

High-Level Features

A k-NN cannot possibly make sense of a 51,200-dimensional space, so we cooked up 28 high-level features to aggregate information in various ways:

- total energy in cells for various values of min cell energy
- moments of energy distribution around track in xy space, in z slices
- number of clusters of cells (above energy threshold)
- number of towers in clusters
- energy of clusters
- imbalance in x and y of energy distribution
- fit to curvature from energy depositions (useful for E in few-100 GeV range)

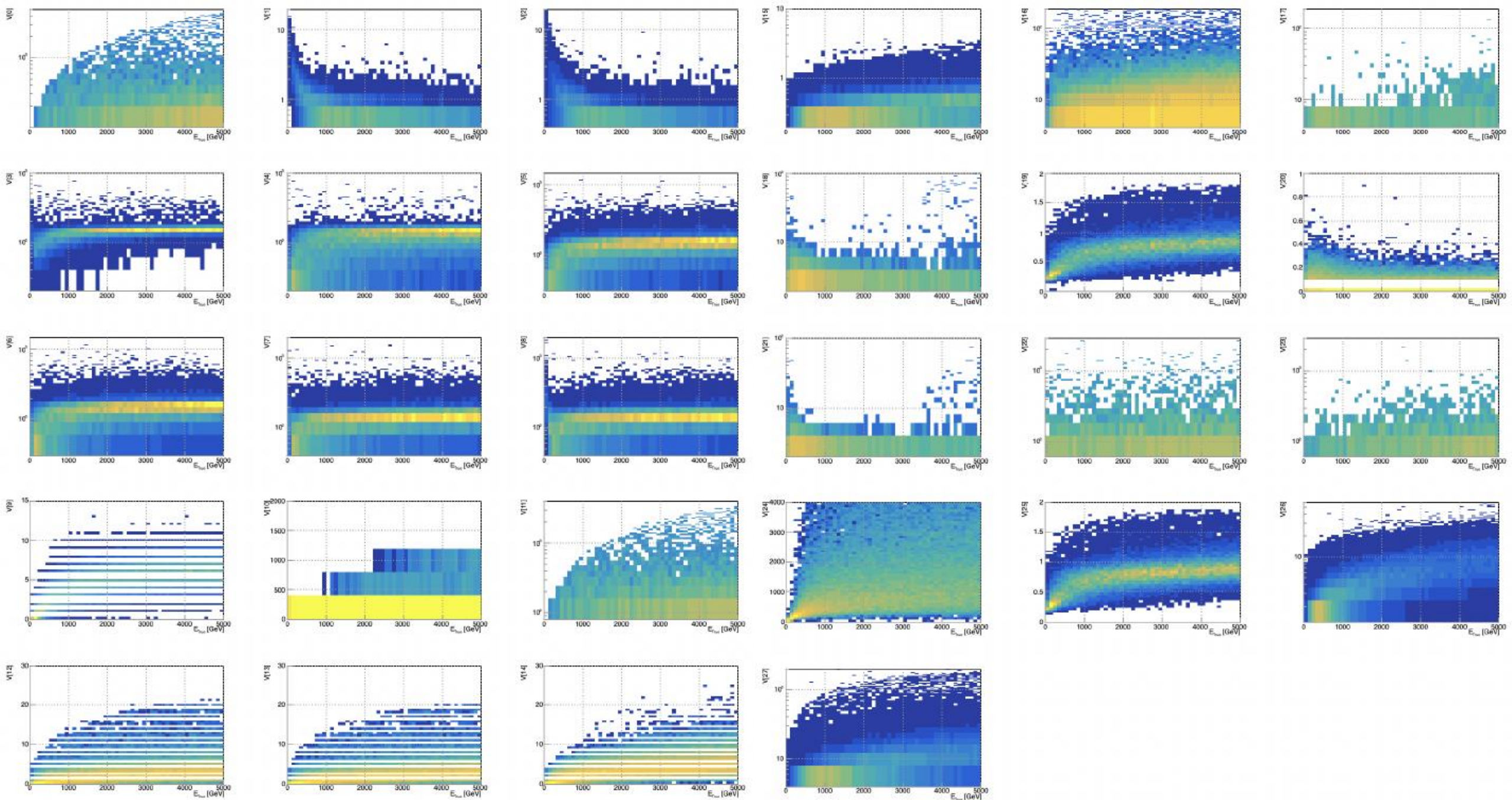


5/11/2022

T. Dorigo, a kNN with 66M parameters

muon energy

7




5/11/2022

T. Dorigo, a kNN with 66M parameters

kNN Construction

The kNN estimate is constructed as the **average estimate from several pools of weak learners**, each pool trained independently on partially disjunct subsets of training data.

Weak learners address the curse of dimensionality (which makes even 28 dimensions too many for local averaging) by **defining the distance in subspaces**, ignoring in turn some of the features through an indicator function, $I(d) = 1/0$ if feature d is considered or not:

e.g. $I = \{0,1,1,1,1,1,0,0,1,1,1,1,0,1,1,0,0,1,0,0,1,1,0,0,0,1,1,1\}$  $\Delta(i, j)^2 = \sum_{d=1}^D I(d)(x_i^{(d)} - x_j^{(d)})^2$

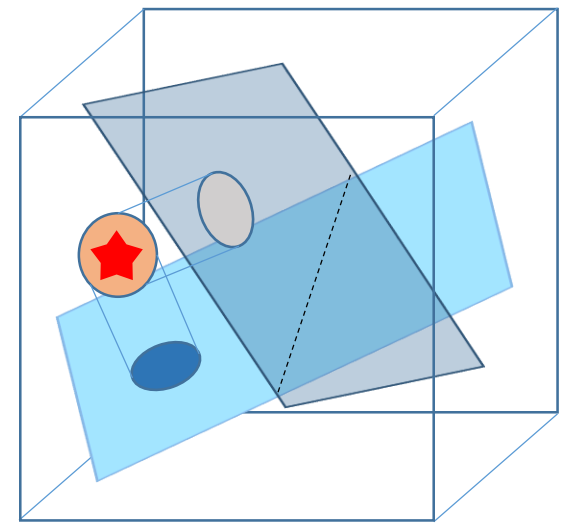
Features in the definition above are standardized to have unit variance and zero mean.

The prediction for test event j can be written as

$$E(j) = \frac{\sum_{m=1}^k E(i_{\text{kNN}}(m))}{k}$$

where $i_{\text{kNN}}(m)$ is the index of the m -th closest training event to j in the subspace, $\Delta(m, j)$.

Pooling of weak learners



The subsampling through $I()$ brings in a loss of information and turns the problem into one of identifying advantageous subspaces and combinations

To reduce information loss, we consider N_{wl} weak learners, each performing a kNN average in a different subspace through a different indicator $I_{wl}()$.

The regressors are then combined in a weighted average:

$$E(j) = \sum_{i=1}^{N_{wl}} W_{wl}(i) E_i(j)$$

Weights $W_{wl}(i)$ are optimized by gradient descent.

The loss of information remains, but its effect is tamed by the added flexibility of the model, and optimized weights W_{wl} .

Overparametrization

One of the aces in the sleeve of DNNs is **overparametrization** – it smoothens the loss function landscape and eases convergence to the real minimum. To inject overparametrization in a kNN one may only **rely on training data**.

Each training event affects the prediction by its position in space (fixed) and by the value of energy (also fixed). One can still **inject flexibility by biasing the energy value, and altering the weight of the event in the averaging**.

We introduce two sets of parameters **$\mathbf{b}(\mathbf{wl}, \mathbf{i})$, $\mathbf{w}(\mathbf{wl}, \mathbf{i})$** for each weak learner:

$$E_{wl}(j) = \frac{\sum_{m=1}^k E_{wl}(i_{\text{kNN}, wl}(m))w(wl, m)}{\sum_{m=1}^k w(wl, m)} + \sum_{m=1}^k b(wl, m).$$

with $O(600k)$ training events, $O(10)$ weak learners per pool, and $O(5)$ pools, this boils down to about **66M free parameters** in the final model.

But can they be trained ??

Gradient Descent

Once a loss function is defined, it is straightforward to **compute derivatives and create a mechanism to navigate the parameter space in the direction of maximum negative gradient**. But the kNN relies for the prediction on data which cannot be the same used to optimize the parameters.

So we split training data in a **training** and a **prediction** set. The former is used in batches to feed gradient descent and learn optimal parameters.

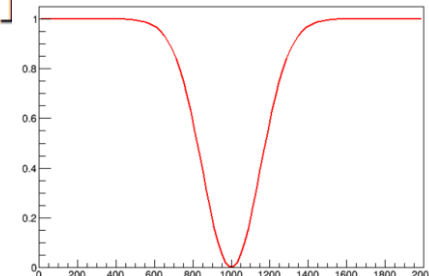
A first-guess loss for a regression task is the MSE: if T is true energy and P its prediction, we want to minimize

$$\alpha_0 \sum_{i=1}^{N_{batch}} \frac{(T_i - P_i)^2}{\sigma_0^2}$$

In our problem we have long non-Gaussian tails in many features. To reduce the effect of outliers, we write the loss as

$$L_1 = \frac{\alpha_0}{N_{batch}} \sum_{i=1}^{N_{batch}} \left[1 - \exp\left(-\frac{(T_i - P_i)^2}{2\sigma_0^2}\right) \right]$$

and tune the sigma parameter to focus on acceptable resolution.



Bias Penalization

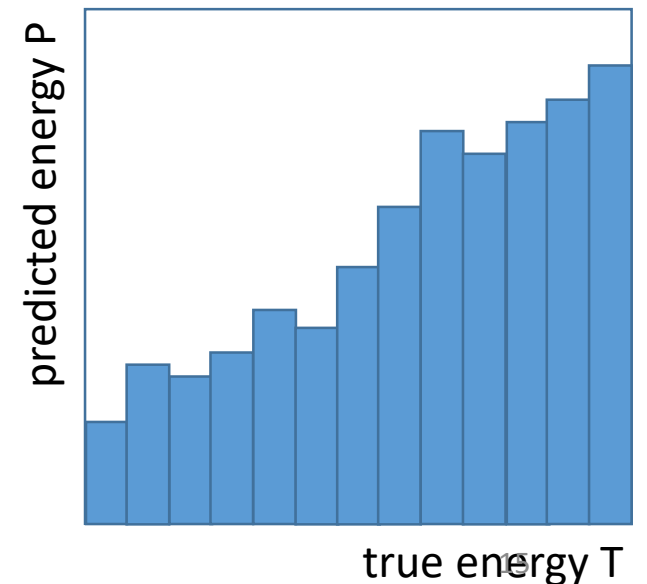
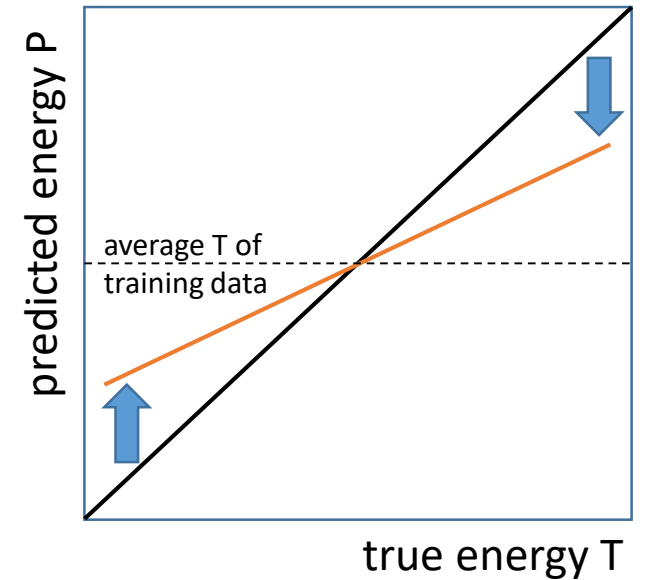
Since we average the energy of muons in a finite energy range ([0.05-8 TeV]), it is to be expected that our estimate will be biased toward the center of the range

A way to reduce this effect is to penalize the loss with the following term, with $m \neq n$:

$$L_2 = \alpha_1 \sum_{m=1}^{N_T} \sum_{n=1}^{N_T} [\hat{P}_m - \hat{P}_n - T_m + T_n]^2 / \sigma(m, n)^2$$

N_T is the number of bins of tested energy; P and T denote predicted and true energy.

The sum considers all pairs of bins to enforce that predictions «line up» along the diagonal



Bias Penalization

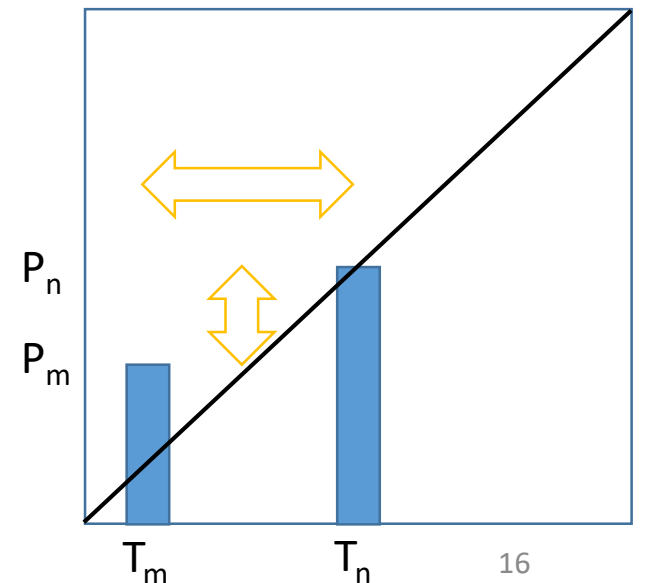
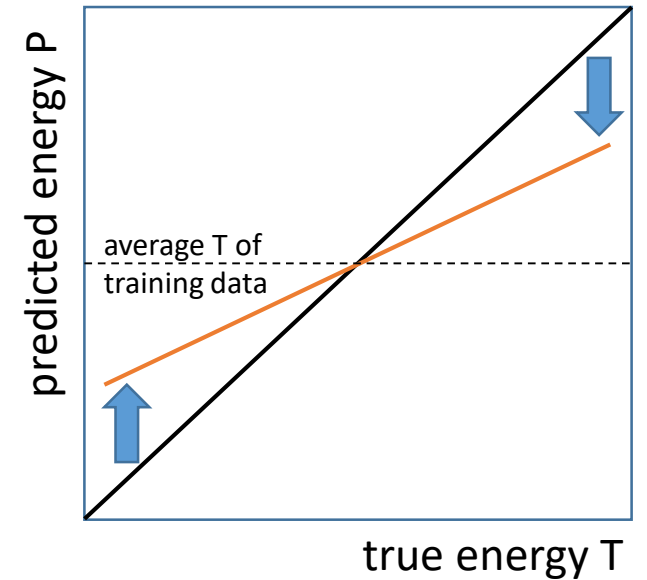
Since we average the energy of muons in a finite energy range ([0.05-8 TeV]), it is to be expected that our estimate will be biased toward the center of the range

A way to reduce this effect is to penalize the loss with the following term, with $m \neq n$:

$$L_2 = \alpha_1 \sum_{m=1}^{N_T} \sum_{n=1}^{N_T} [\hat{P}_m - \hat{P}_n - T_m + T_n]^2 / \sigma(m, n)^2$$

N_T is the number of bins of tested energy; P and T denote predicted and true energy.

The sum considers all pairs of bins to enforce that predictions «line up» along the diagonal



The denominator in L_2

$$L_2 = \alpha_1 \sum_{m=1}^{N_T} \sum_{n=1}^{N_T} [\hat{P}_m - \hat{P}_n - T_m + T_n]^2 / \sigma(m, n)^2$$

The denominator is constructed by assuming that uncertainties in energy deposits scale with Poisson statistics, so that an estimate of the uncertainty in the difference at the numerator above as

$$\sigma_{\Delta_{mn,obs} - \Delta_{mn,exp}}^2 \simeq \frac{T_m}{N_m} + \frac{T_n}{N_n}$$

with N_m the number of events in bins m . This gives equal importance to deviations over a small energy difference $T_m - T_n$ and over a large one. A way to give more weight to larger differences is to add that term to the denominator. Our choice then is

$$\sigma(m, n)^2 = \frac{T_m N_n + T_n N_m}{N_m N_n (T_m - T_n)}$$

Other figures of merit

The loss as defined above is not necessarily the best proxy to the success of our regression task, as **we are mainly concerned with the high-end part of the spectrum**, where a curvature measurement would be inadequate.

In addition, **we assume we will combine the calorimetric measurement with a tracking one with a resolution $\delta E/E = 0.2E/\text{TeV}$.**

The combination yields

$$RMS_{comb}(E) = \sqrt{\frac{RMS_{tr}(E)^2 RMS_{cal}(E)^2}{RMS_{tr}(E)^2 + RMS_{cal}(E)^2}}$$

and we define as a figure of merit the function $MaxRes = \max_{E=0.05-4 \text{ TeV}} RMS_{comb}(E)$

Further, we gauge the discrimination power of predicted energies for 4 vs 2 TeV muons and 3 vs 1 TeV muons by computing the following proxies:

$$Discr_{24} = \frac{\hat{E}_p(4) - \hat{E}_p(2)}{\sqrt{\sigma_{E_p}(4)^2 + \sigma_{E_p}(2)^2}}$$

$$Discr_{13} = \frac{\hat{E}_p(3) - \hat{E}_p(1)}{\sqrt{\sigma_{E_p}(3)^2 + \sigma_{E_p}(1)^2}}$$

* Features pruning

Regardless of our dimensionality reduction based on feature subsampling, it is useful to identify and **get rid of variables that are not useful for the regression task.**

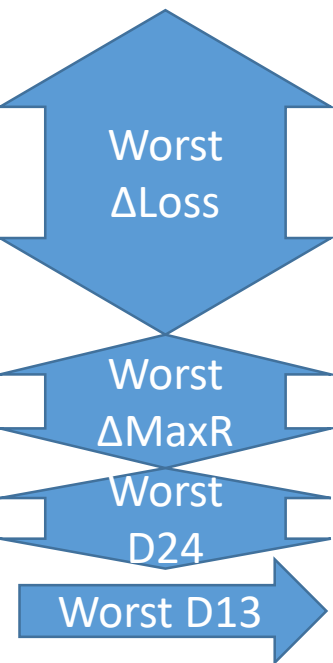
The identification is difficult, as the feature space is complex and the interdependencies non trivial.

We run 1639 independent regression tasks using **randomly generated single weak learners** on the same training data sample (50,000 events), with subspace dimension variable between 8 and 15, and compute the average value of figures of merit **as a function of their inclusion or exclusion of each feature.**

* Pruning study results

Variable	Δ Loss	Δ Max res. ($\times 10^4$)	Δ 2-4 TeV discr.	Δ 1-3 TeV discr.
4	0.2939 ± 0.0336	-2.60 ± 1.80	-0.00105 ± 0.00138	-0.01427 ± 0.00200
5	0.2531 ± 0.0352	6.91 ± 1.76	-0.00432 ± 0.00123	-0.01120 ± 0.00219
6	0.2299 ± 0.0377	11.91 ± 1.73	-0.00505 ± 0.00125	-0.00803 ± 0.00222
7	0.2034 ± 0.0366	17.80 ± 1.77	-0.00770 ± 0.00126	-0.01002 ± 0.00219
3	0.1646 ± 0.0332	4.40 ± 1.82	-0.00079 ± 0.00140	-0.00357 ± 0.00207
25	0.1575 ± 0.0363	0.67 ± 1.86	-0.00506 ± 0.00139	-0.00585 ± 0.00215
8	-0.0006 ± 0.0365	3.40 ± 1.81	0.00303 ± 0.00145	0.01068 ± 0.00221
11	0.0910 ± 0.0341	2.93 ± 1.74	-0.00366 ± 0.00116	-0.00650 ± 0.00197
18	0.0898 ± 0.0346	-1.02 ± 1.69	-0.00422 ± 0.00116	-0.00633 ± 0.00198
10	0.0985 ± 0.0331	2.61 ± 1.73	-0.00368 ± 0.00118	-0.00507 ± 0.00193
20	0.0997 ± 0.0359	-2.91 ± 1.764	-0.00309 ± 0.00116	-0.01228 ± 0.00203

(larger is worse) (larger is worse) (smaller is worse) (smaller is worse)



Listed here are variables that are among the **six worst on any one of the four figures of merit.**

The variables which do **not** worsen the performance of the corresponding figure of merit are highlighted in boldface.

We finally reject all listed variables except variable 8, which is only present in the list due to the (relatively noisy) max res.

* Weights and biases initialization and learning

$W()$ and $b()$ need to be initialized. $b()$ is set to zero, $W()$ to a value that is $=1.0$ below 5 TeV (4 TeV is the upper limit of the regression range), and smoothly decays to 0 for $E=8$ TeV with a sigmoid (same approach was used by CNN paper).

During training, the weights and biases get tweaked by the learning process. For $W()$ you still see some remainder of the initial trend, but with large spread

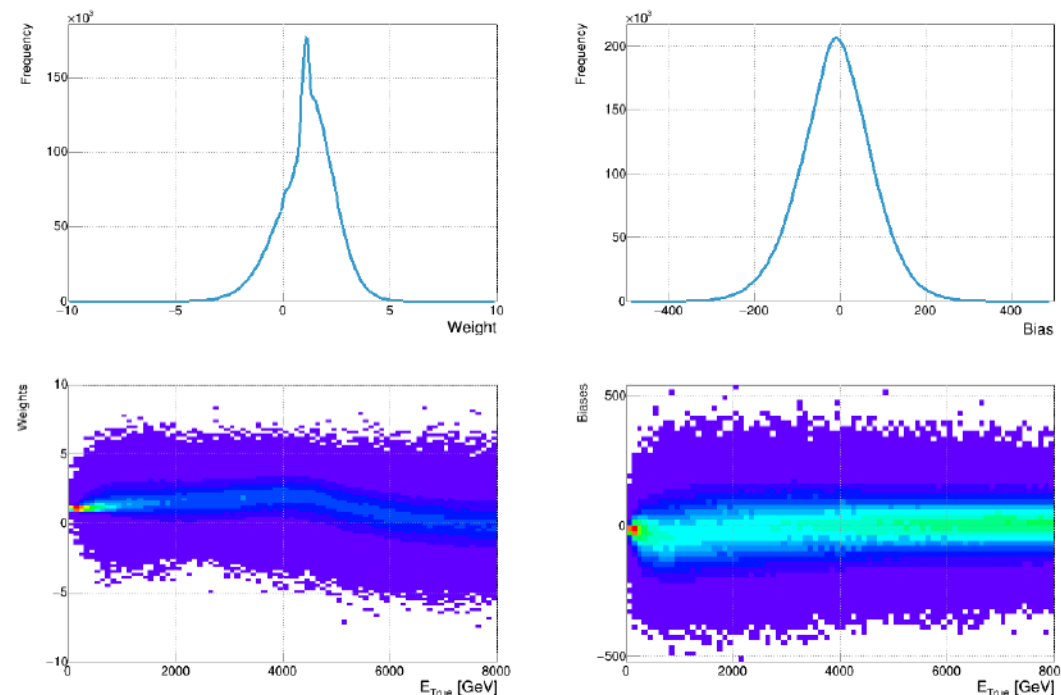


Figure 7: Example of the distribution of weights obtained by an optimization run with 400,000 training events and 5 learners. Top left: distribution of event weights; top right: distribution of event biases (in GeV). Bottom left: weights versus true muon energy; bottom right: biases (in GeV) versus true muon energy.

Final run

For each of the four sets of 8 regressors with $N_{wl}=5,10,15,20$, **two are chosen based on the FoMs already discussed**, and using cross validation. Finally, we run the eight sets on 80,000 test events and 400,000 or 300,000 training events.

The total number of parameters used in the final regression is

$N_{weights} = (2*5 + 2*10)*400,000 + (2*15+2*20)*300,000$ and the same number of biases, plus 100 global weak learner weights. The total is thus $(12M+21M)*2 +100 = 66,000,100$ parameters.

Each job needs about **one week** to complete the training.

The final loss from the averaging of the 8 regressors is 33.734, MaxRes is 0.3372, Discr24 is 1.3318, and Discr13 is 0.7025. These numbers are better than those of the 8 inputs.

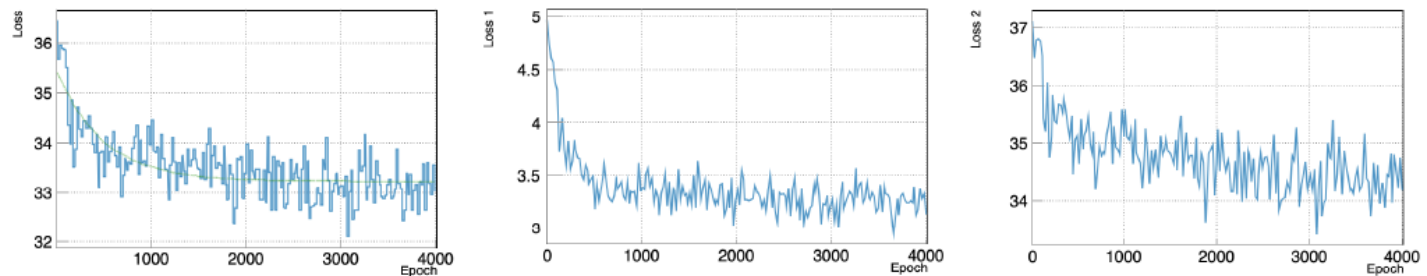


Figure 8: Evolution of the loss function and its components with the gradient descent iterations (epochs). Left: total loss function (L); center: non-linearity penalization term (L_2), right: modified MSE term (L_1). The two terms on the center and right do not add up to the one on the left because of a rescaling factor.

Results

The regression shows an almost linear response, indicating that the bias correction penalization helped. The MSE reaches down to 22%E at 4 TeV.

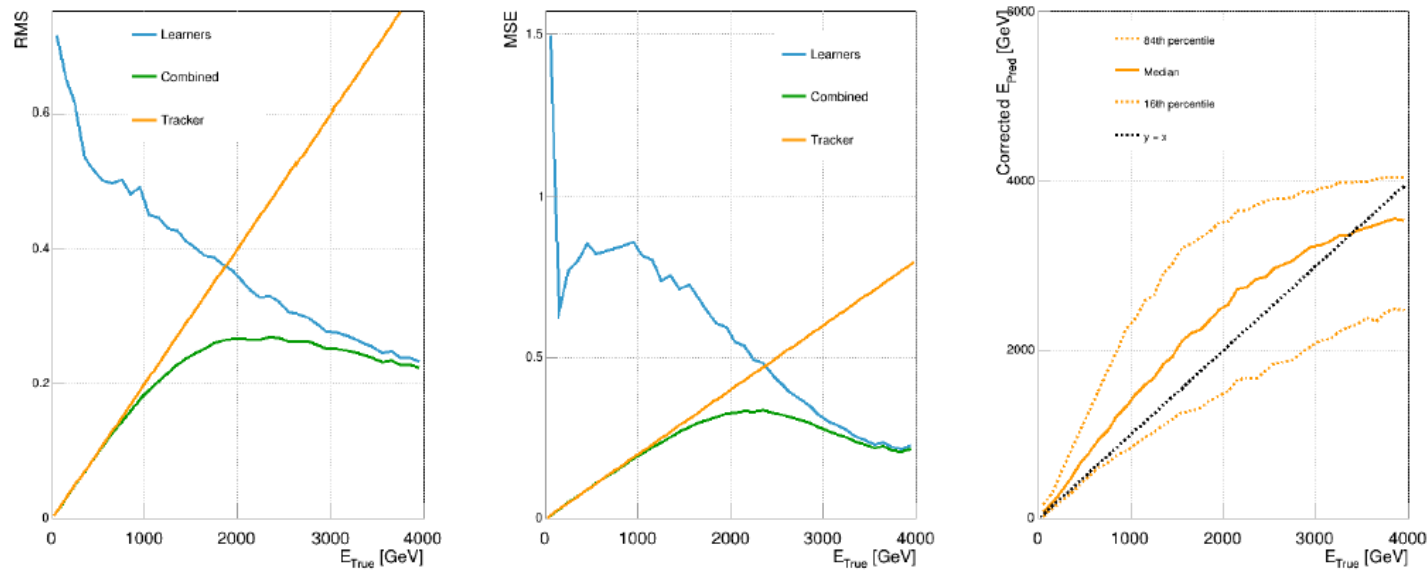
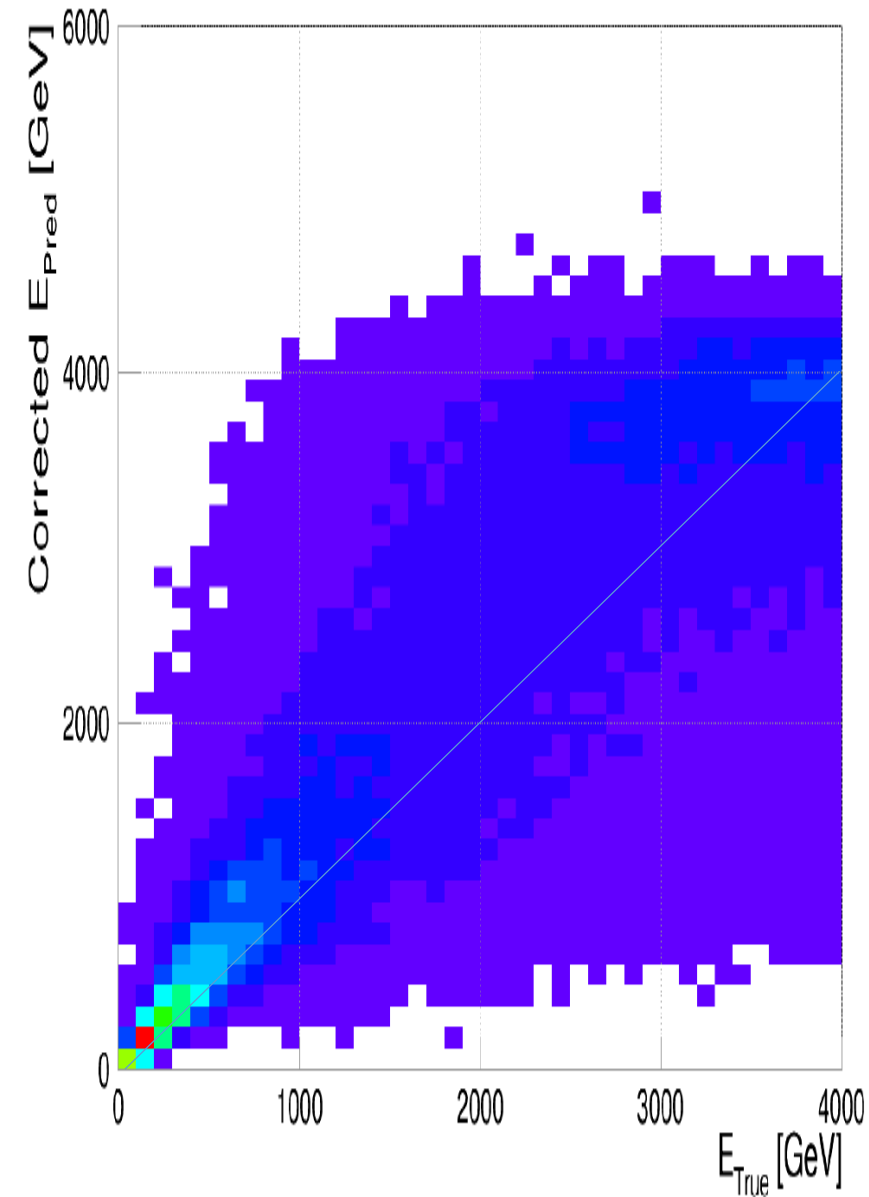


Figure 11: Left: RMS resulting from learners-only, tracker-only and combined predictions, against true muon energy; center: MSE resulting from learners-only, tracker-only and combined predictions, against true muon energy; right: median, 84th, and 16th percentiles of corrected predicted energy against true muon energy.



Results/2

For a comparison, we look at results of a NN (orange), a default kNN (pink), and Xboost.

The results of our k-NN outperform the standard k-NN and are overall similar to those of NN and BDT methods, and even slightly better than those at high E.

But the CPU and analysis load is non comparable!

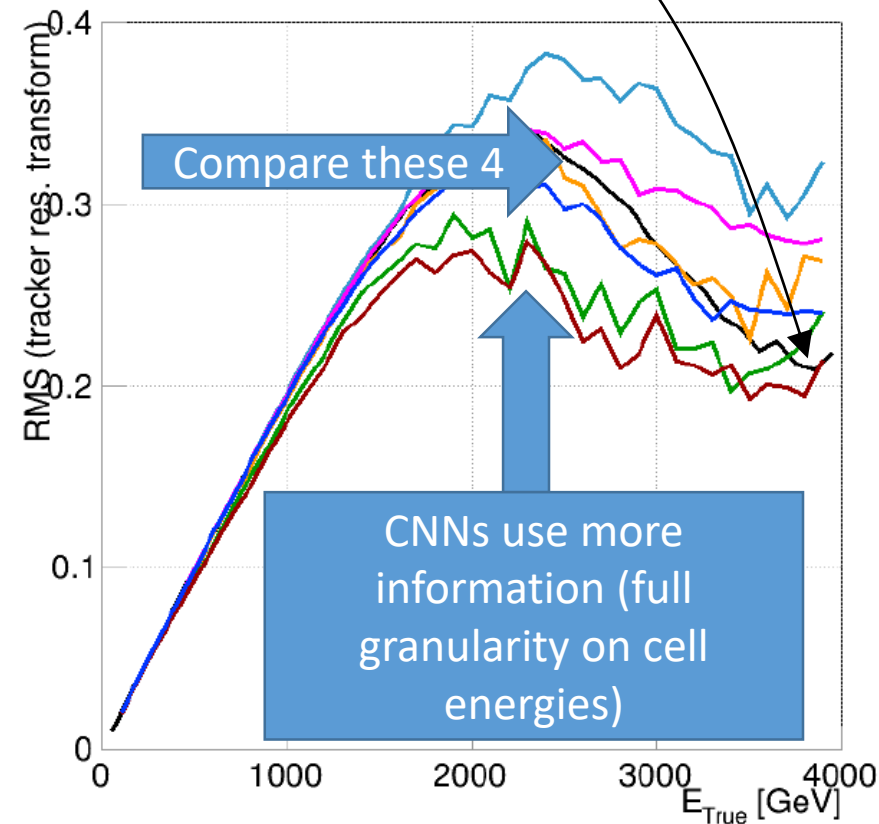
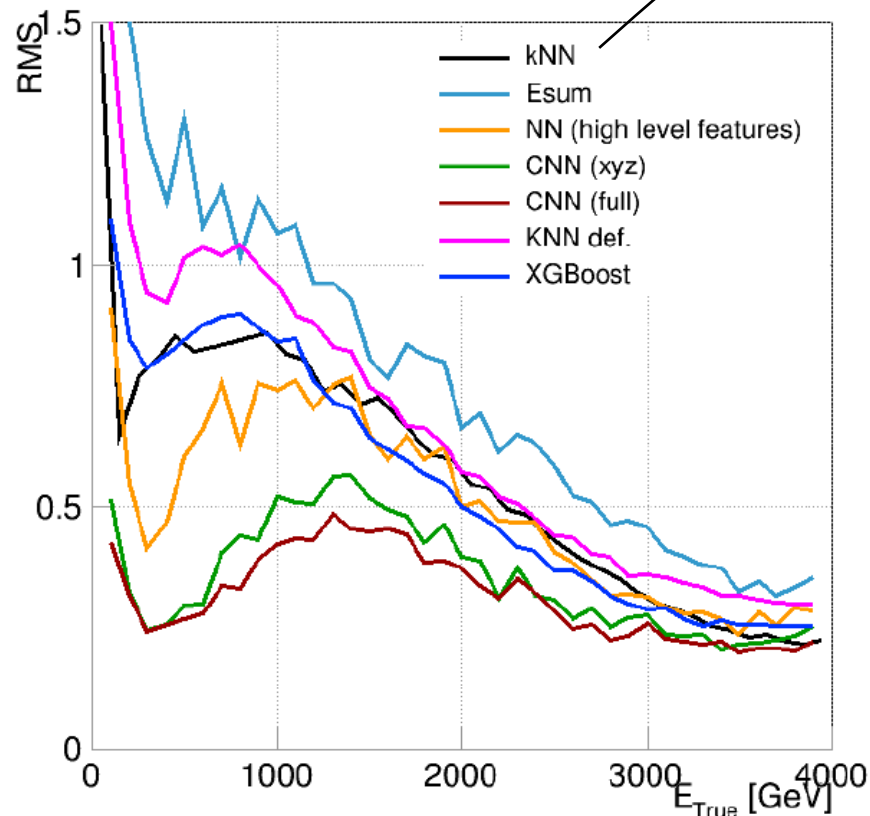


Figure 13: Left: comparison of the mean squared error of predictions of different algorithms employing the same training and test data. Right: comparisons of the mean squared error of the combinations of tracker and calorimeter regressed predictions. Black: deep regression kNN (described in this article); light blue: energy sum model; orange: neural network with high level features; green: CNN with spatial features; red: full CNN; magenta: classical kNN; blue: XGBoost.

Conclusions

Deep learning achieves great results in a number of problems, and that is wonderful! - but we should not forget that **there is nothing really special in it**: overparametrization and gradient descent can be applied also to more mundane tools, with similar outcomes (but way more CPU-hungry)

In the future, high-energy muons will pose a challenge to high-energy colliders. We argue that high-granularity calorimeters are suitable for a measurement of their energy at arbitrarily high values

Thanks for your attention!

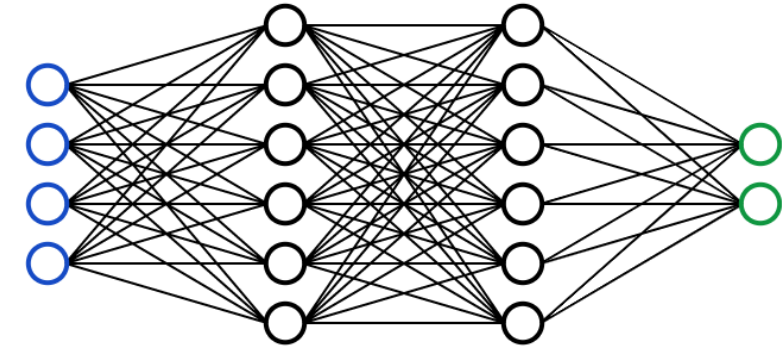
Before 2012...

Long before most of us started thinking about machine learning (ML) tools for HEP, statisticians developed and used a huge weaponry of statistical learning tools

- Linear discriminant analysis is a 1936 brainchild of R. Fisher
- k-NN was invented in 1951 (Fix, Hodges)
- Clustering dates back to 1938-39 (Zubin, Tryon)
- Cross-validation was invented in 1974 (Allen, Stone)
- Boosting comes from ideas of the late eighties, then Schapire 1990.

When ML boomed in the early 2000s, it was largely a computer-driven revolution rather than the discovery of new tools

The rise of deep learning in HEP



DNNs use skyrocketed in HEP after 2012, when BDTs and NNs were used for the Higgs discovery (2012 is also the turning point in the imagenet challenge). A true paradigm change!

Further evidence of the benefit of ML tools for HEP was given by the [Kaggle Higgs challenge](#) [Kaggle 2014], with 1800 teams participating (physicists, statisticians, computer scientists). Task: separate $H \rightarrow \tau\tau$ decays from backgrounds in LHC simulated data

The most effective solution was based on a pool of DNNs, with emphasis on cross-validation

Alternative methods commonly used in HEP (xgboost, Bayesian NN, etc.) were beaten soundly

Solution	Score
Gabor Melis (DNN pooling)	3.806
MultiBoost	3.405
TMVA boosted trees	3.200
Naive Bayesian classifier	2.060
1D cut based selection	1.535

equiv. to 6 times more data!

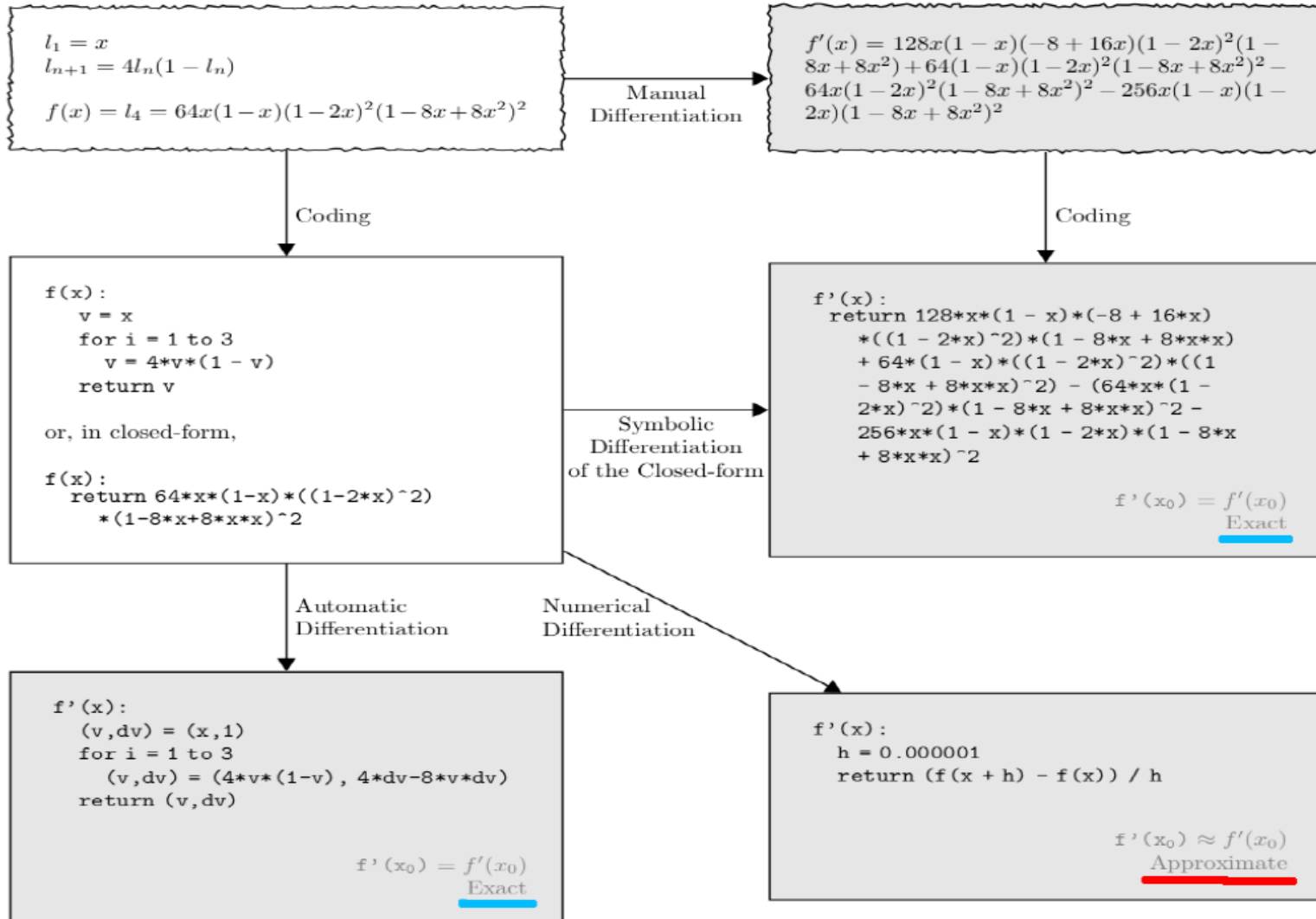
The weaponry of the 2020s

Current trends of deep learning applications in particle physics include

- DNNs for classification, particle ID, signal discrimination, deep regression
- CNNs for image-based classification (e.g. boosted jets)
- Graph NNs for event reconstruction
- Differentiable pipelines for incorporation of nuisance parameters in supervised learning tasks
- Implementation of NNs in FPGAs for online data acquisition
- VAEs for anomaly detection
- GANs for generative models
- End-to-end optimization of experiments

The above tools essentially use the same engine under the hood: the **chain rule of differentiable calculus**, which allows **gradient descent**

* Automatic Differentiation



Wouldn't it be nice if you coded a problem and the dependence of variables in a program, and the language took care of figuring out how functions vary depending on parameters, and **carry out the complicated task of propagating derivatives around?**

Those of us who have done this manually can't be happier by seeing the rise of Pytorch, TF, etc.

Manual differentiation

Automatic differentiation is great, and is speeding up progress... But **manual differentiation** also works!

In this talk I will discuss a result produced with unrefined c++ code, by manually implementing a calculation of a loss function and its derivatives

The message will be that **what is really important is the statistical model**, not the tools that we apply to it to get our answer...

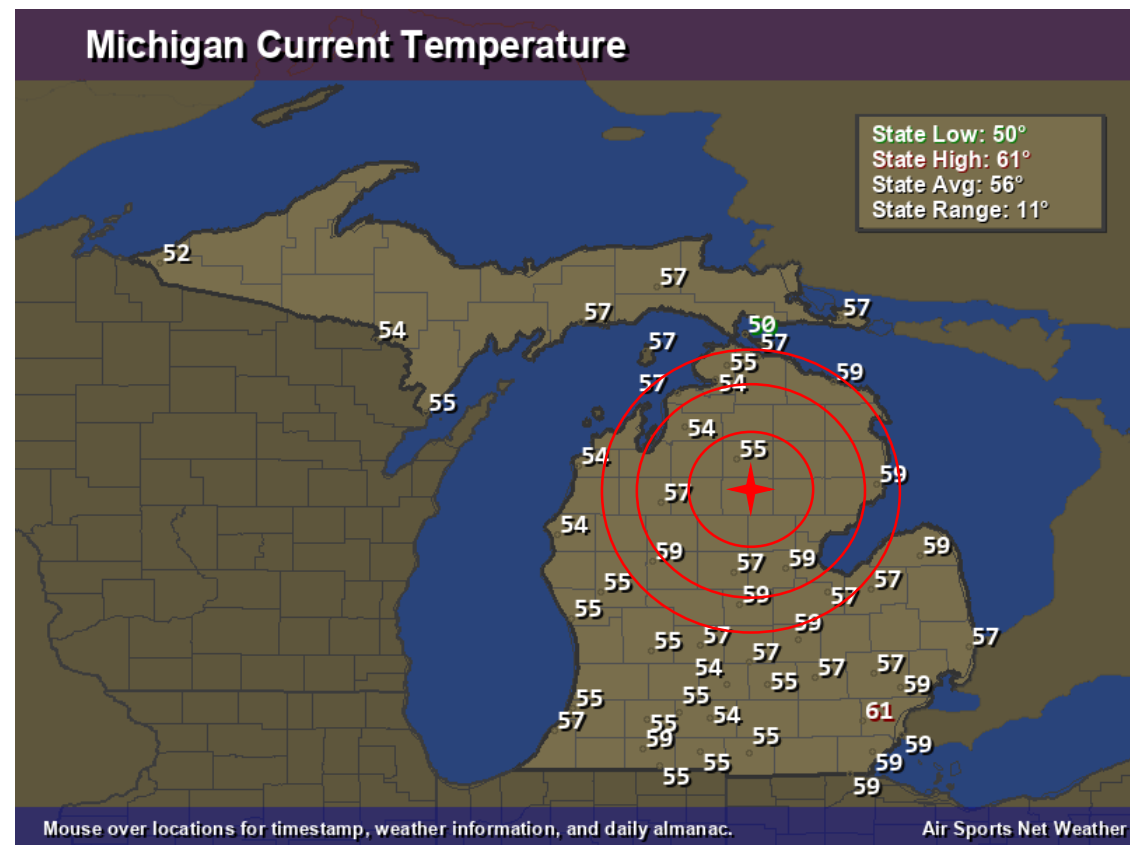
* What is a k-NN?

It is entirely possible that you do not know what a k-NN is, so here is my one-slide explanation of a regressor k-NN.

Imagine you need to estimate the temperature in Roscommon co., Michigan (red star) based on the data on the right.

You might choose to take the state average of 56°F; that would be a low variance/high bias choice

A more principled procedure is to check the temperature of neighboring counties.



As temperature is connected to **spatial distance**, you may want to include/exclude data points based on their distance from Roscommon.

In k-NN, you specify k and the distance rule.

The more you restrict the range of counties which you allow in your average, the more «relevant» they become: bias decreases; but variance grows. 32

Hyperball optimization

An additional feature of the developed algorithm is an adaptive shape of the k-balls, by giving weights to the distance components proportionally to the variance exhibited, in the proximity of the test point, by the quantity to be estimated.

An estimate of the bias due to the variability of E as a function of x in the surrounding of the test point (x=0 here) is provided by fitting with a parabola the deviations from the mean.

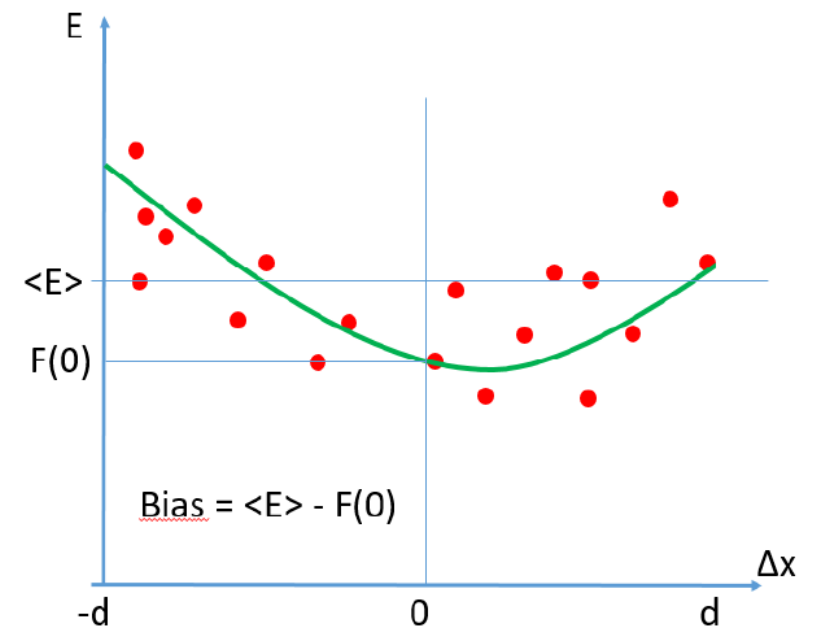
This corresponds to simply getting the average energy in a wide region [-d,d] and in a central region $E_0 = \langle E \rangle([-d/3, d/3])$, and computing the bias as

$$b = \langle E \rangle - E_0.$$

The biases can then be used to give more or less importance to the different features in the calculation of the distance:

$$d = \sum_{i=1}^{N_D} \frac{1}{b_i/b_{max} + \epsilon} (x_{i,j} - y_{i,j})^2$$

The calculation requires to study the vicinity of the test point with large k (3-5 times the normal value) and is thus very CPU expensive.



Validation of pruning procedure

We validate the pruning procedure by running a regression on all 28 variables, and then removing gradually the worst features.

Excluded variables	Loss	Max res.	2-4 TeV discr.	1-3 TeV discr.
None	38.4141	0.330749	0.706453	1.12254
3, 4, 5, 6	37.5701	0.32778	0.709819	1.15602
3, 4, 5, 6, 7, 25	37.2649	0.327508	0.715047	1.1493
3, 4, 5, 6, 7, 25, 10, 18, 20	37.3583	0.325874	0.718947	1.15443

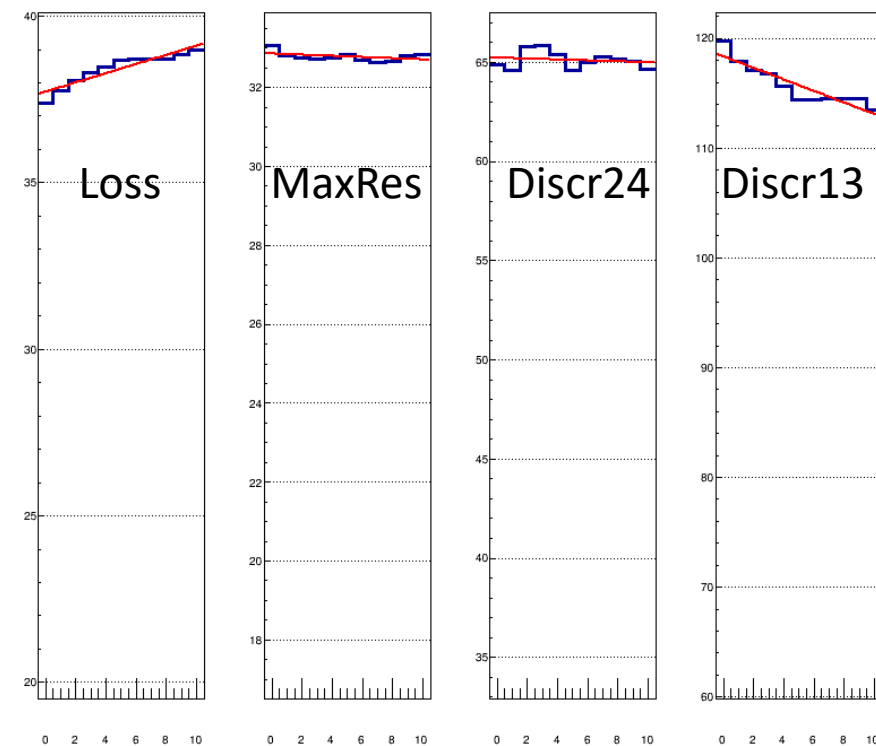
As expected, we observe a decrease of the loss and MaxRes, and an increase of the discrimination power

Validation of pruning/2

Further evidence of the validity of the choice comes from [starting off from the best 5 features and adding the worst ones one by one](#). We see a clear increase of the loss and a decrease of Discr13; the other two FoMs remain basically untouched

Added variables	(lower is better)		(higher is better)	
	Loss	Max res.	2-4 TeV discr.	1-3 TeV discr.
0	37.40	0.3306	0.6486	1.1972
1	37.77	0.3282	0.6458	1.1787
2	38.06	0.3276	0.6580	1.1714
3	38.31	0.3272	0.6583	1.1674
4	38.46	0.3276	0.6536	1.1567
5	38.69	0.3283	0.6459	1.1438
6	38.73	0.3270	0.6498	1.1444
7	38.70	0.3264	0.6526	1.1451
8	38.70	0.3266	0.6518	1.1451
9	38.85	0.3280	0.6503	1.1449
10	38.97	0.3285	0.6465	1.1384

Minimum value of graphs is 50% of maximum



* Regressor details

- k is set to 100. This was not «properly» optimized but the regression performance was checked for a few values in [10,500]. CPU limitations make too large values impractical; smaller values start to make the loss very noisy (especially L2)
 - large k has benefit of giving more flexibility to predictions ($200 * N_{wl} + N_{wl}$ parameters per test point)
- 300,000 / 400,000 events used for prediction
- 400,000 events are used for training (batch GD)
- 100,000 events are used for testing
- 18 variables used out of 28 (10 excluded by pruning)

* Weak learners choice

We used 32 sets of weak learners: 8 sets of 5 learners, 8 sets of 10 learners, 8 sets of 15 learners, and 8 sets of 20 learners.

Each set was separately determined by optimization searches running on cross-validation sets of training data, without an optimization of $w()$ and $b()$ parameters. The procedure was as follows:

1. Select by bootstrapping 10,000 training events
2. Define at random N_{wl} weak learners, by selecting a variable fraction (from 30% to 80%) of the active features among the 18;
3. Iterate to modify active flags, by flipping some of the flags on or off, then performing the regression on 5000-event batches; compare loss, minimize it

Flags of weak learner in pool (0:27)	Individual loss of WL
1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 1 1 0 1 0	37.22
0 1 1 0 0 0 0 0 1 1 0 0 0 1 1 1 0 0 0 0 0 1 0 0 1 0 1 1	37.37
1 1 1 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 1 0 0 0 0 1 1	37.34
1 0 1 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1	37.60
1 1 1 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0	37.53
Global loss for this batch	37.02

Table 5: Active subspace dimensions for five weak learners in a 5-WL set.

The resulting sets of weak learners perform significantly better than the original random sets. Typical decreases of loss by 2% to 5% observed.

Other recipes, based on genetic breeding, did not produce good results.

* Other hyperparameters

- Alpha parameters (loss multipliers) must be defined for the two loss components L1, L2. **This is tricky and requires some fine-tuning**
- The learning rate must be set to a value that allows smooth descent. Also tricky
- Learning rate is also updated during gradient descent via additional scheduling parameters, depending on the steepness of the descent
- N_{batch} is the number of training events used to evaluate the loss during learning – must be large enough for L2 to be meaningful (40 bins, want $O(100 \text{ GeV})$ uncertainty on predictions per bin with RMS of $O(1 \text{ TeV}) \rightarrow 100 \text{ events per bin} \rightarrow$ **5000 events per batch**)