ICENET

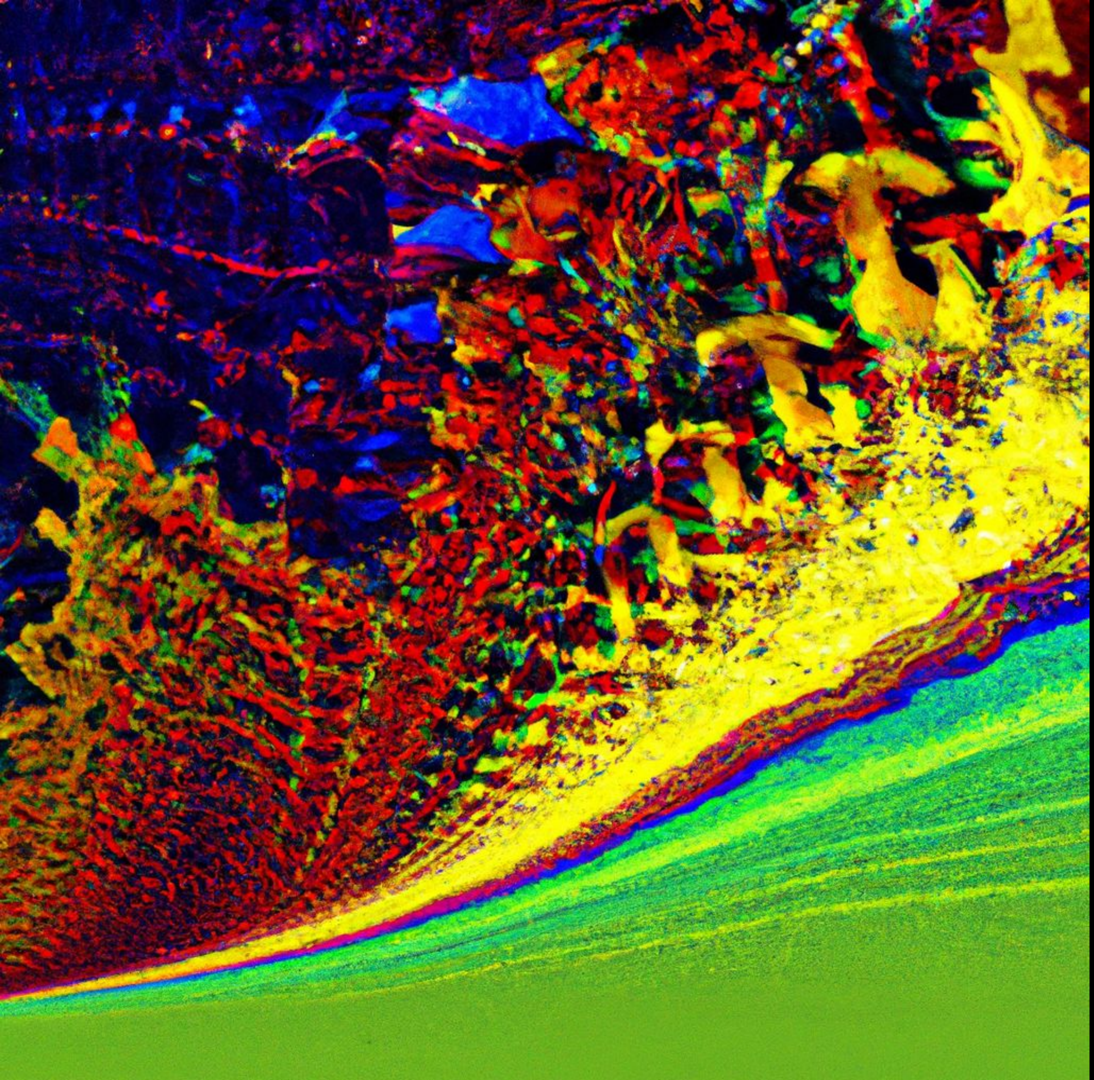
Mikael Mieskolainen
Imperial College London

CMS Machine Learning Forum, 8/2/2023

Code: github.com/mieskolainen/icenet

Docs: mieskolainen.github.io/icenet

m.mieskolainen@imperial.ac.uk



Functionality

Starting point

A python deep learning/ML library (bag-of-tricks) – started as a postdoc (Imperial, 2020 –)

- ❖ First for my own research, then later students started to use it – R&D, collaboration, education!
- ❖ Tensorflow had already decayed in ML/AI research → Start with **torch**

[JAX was an alternative, had perhaps slightly better autograd (e.g. higher order derivative recursion), but popularity a bit niche...]

My workflow: develop first models outside *ICENET* in isolated sandbox repos, then migrate to *ICENET* if the method seems directly applicable for ***physics***.

Raw input processing without ROOT

Anything goes basically (ROOT, pandas, HDF5, pickle …), emphasis on ROOT files

- **uproot** based processing; **awkward** array and/or numpy object output
- **Parallel processing** of multiple ROOT files accelerated with **RAY** (Berkeley HPC) – overcomes lack of real multithreading in Python (global interpreter lock) and more flexible than multiprocess of Python. Server-client model, also can distribute execution *transparently* via SSH.
- Custom utility functions to overcome e.g. some limitations of uproot

www.ray.io awkward-array.org uproot.readthedocs.io

Config steering via YAML

- ‘Canonical workflow’ YAML-templates; steering and ML model definitions
- Can also just break the workflow, design is mostly functions + dictionaries
- YAML meta-generation: python e.g. generates steering YAML-files based on certain folder structure (e.g. different SM backgrounds in different (sub)-folders, each with different integrated x-section weights etc.)

- YAML/Python kept “pretty flat” – No heavy OOP of type C++, no complicated design patterns, no highly engineered abstractions

- YAML standard is slightly extended, e.g. includes are possible

Conversion to ML data structures

- Simple fixed dimensional arrays (→BDT, MLP)
- **CMS nanoAOD**: leptons / jets / sec. vertices / custom collections with maximum object count per event (zero-padded) → flat fix dim. arrays out
- Fixed dimensional tensor arrays (→for classic CNN type models)
- DeepSets compatible ‘set arrays’

- Most advanced is torch-geometric based event structure. Graph construction from awkward arrays with (custom) nanoAOD structure. Custom tools to make this easier. Graphs parallels processed (again) with Ray.

pytorch-geometric.readthedocs.io

ML preprocessing

- ‘Columnar’ pre-selection cuts
- Boolean correlation statistics “**parallel *N*-cut flow**” (2^N pass/fail combinations)

- Physics process cross-section based mixing, weighting (resampling)
- Differential kinematic re-weighting (1D, 2D-histogram based)
- Neural reweighting also possible; train a classifier, use output as a weight (use e.g. “temperate” calibration if needed, see arxiv.org/abs/1706.04599)

- Basic input variable normalization, e.g. z-score (classic shallow learning methods may need), robust variants

ML training automatization

- All-in-memory or from-the-disk in a loop – RAM vs CPU-time, problem dependent

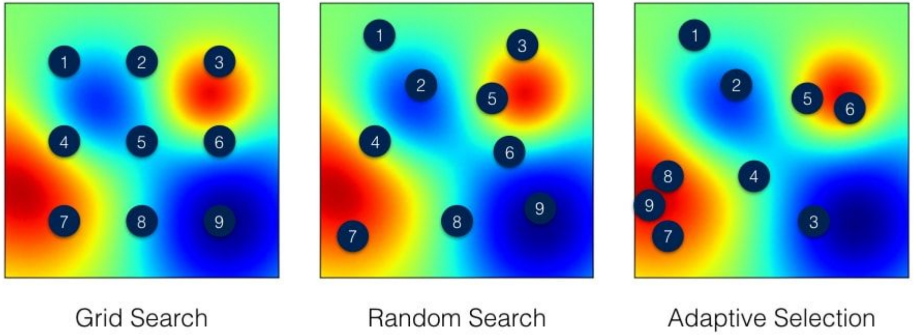
- **Basic quality control** of input variables; point statistics, distributions
- Basic quality control of losses (per loss term, main + regularizations) and AUC
- Various loss functions
- Standard torch stoch. gradient optimizers (AdamW, Adam ..) + gradient clipping etc.

- **Model distillation** supported (arxiv.org/abs/1503.02531)
- **Deep Domain Adaptation** (grad. reverse with GNNs) formally supported (testing under work), XGBoost (ICEBOOST) with DA under construction

ML hyperparameter optimization

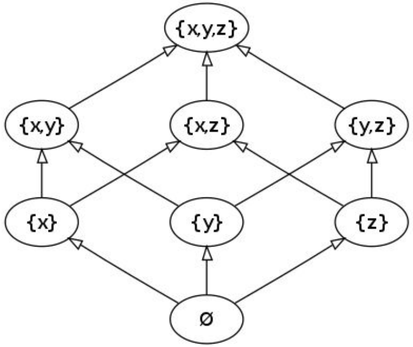
For model parameters such as #layers, #neurons, activation type, operators …

Implemented via **RayTune**, steered in ICENET via YAML-files; describe which model parameters + their type (discrete set, real valued) and search ranges and the optimizer (random search, Optuna, Bayesian optimization …)



docs.ray.io/en/latest/tune/index.html

ML evaluation automatization



Emphasis on classification

ROCs with Efron's percentile bootstrap stat. uncertainties ++

Compare easily and 1-to-1 different models, cuts, models with reduced input etc.

"Powerset expansion"

YAML-steered cut-based categorization constructions, e.g. ROC evaluated for different phase-space slices / final state categories / MC signal model points (mass, lifetime ..)

Future reservation

Various continuous regression tasks, Inverse Problems (neural unfolding, efficiency inversion) etc. evaluation automatization currently *not in ICENET* (other codebases for those)

ML deployment

Under development in an on-going Dark QCD search analysis

- ➢ Uses *ICENET* machinery (read ROOT files, applies elementary precuts, ML-input processing, computes the model predictions)

- ➢ Use Imperial / Blackett lab computing grid, launch conda (typical Oracle grid engine)
- ➢ Write new ROOT-files containing only ML-scores, 1-to-1 event matched with the nanoAOD input ROOT-files
- ➢ Read ML-scores in as ROOT Friend Trees in nanoAOD-tools based analysis
- ➢ **Workflow successfully tested**, performance & scaling under work

Alternative deployment schemes possible, e.g. convert XGBoost BDT to standalone C++ (code will be made available for this)

Workflows

Install conda env, install requirements, run

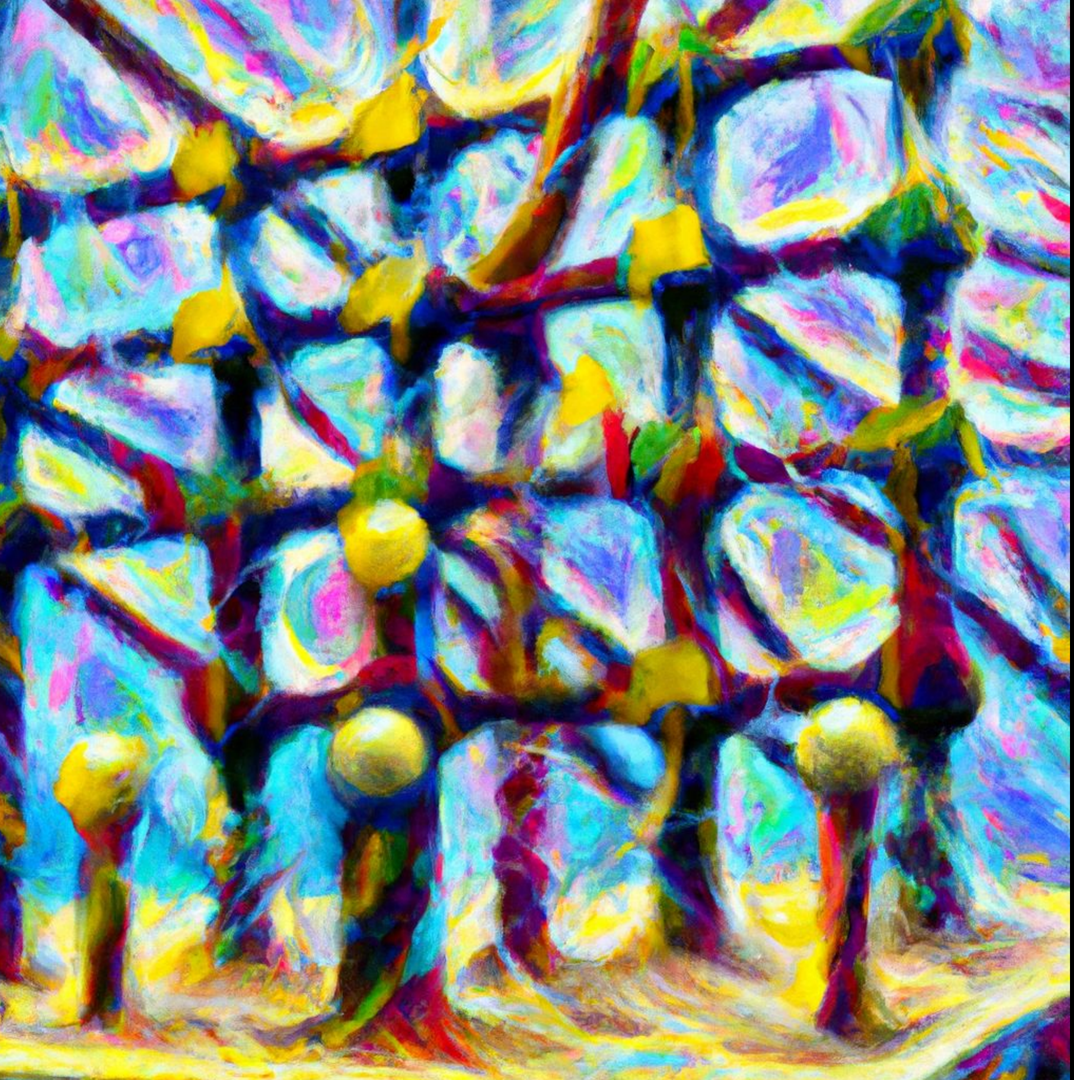
Github actions CI (all workflows not there due to test ROOT input files – will update)

github.com/mieskolainen/icenet/blob/master/.github/workflows/icenet-install-test.yml

Workflows as modules

❖ icedqcd / Dark QCD search (XGBoost + GNNs), development on-going
❖ icehnl / HNL search proof-of-concept (*ICEBOOST*: BDT with neural mutual info)
❖ icebrk / B-parking R(K) ratio prototyping via novel “weighted-combinatorics” scheme
❖ iceid / Low-pT electron ID with GNNs / BDT
❖ icetrg / B-parking di-electron HLT trigger prototyping with BDT / simple cuts
❖ icehgcal / HGCAL Hackathon GNN prototyping, two different problems (Summer 22)

YAML-configuration: /configs, **Shell launch**: /tests, **Steering**: /analysis



Models

Models

Currently out-of-the-box

- ❖ Bulk MLPs and their variants + dropouts, 1D batch-norm etc.
- ❖ DeepSets
- ❖ Classic CNNs (2D, 3D, …)
- ❖ XGBoost and XGBoost + torch autograd based losses (*ICEBOOST*)
- ❖ Bulk VAE / AE (generative, anomalies)
- ❖ Normalizing flows (generative / density estimators, [anomalies])
- ❖ Graph Neural Networks (torch-geometric); node, edge and graph level prediction tasks, various operators pre-interfaced

Basic torch conventions followed: adding new torch models relatively easy; add YAML-descriptions + model class .py + modify certain directives → in future perhaps more streamlined recipe

Novel models – ICEBOOST *(details in Appendix)*

Use XGBoost (empirically very strong for HEP problems) + autograd machinery of torch + torch NN models to construct custom loss functions for XGBoost

→ **A novel hybrid**: *Neural Mutual Information* (arxiv.org/abs/1801.04062) constrained BDT → make the BDT output statistically independent of some other variable (e.g. object tagger score) while balancing strong BDT performance

→ **Possible other applications**: Neural Domain Adaptation for XGBoost (philosophically different than deep feature space alignment, studies on-going)

For earlier work, see: Kasieczka, Shih, Robust Jet Classifiers through Distance Correlation, PRL (2020)

Novel models – ICEBOOST *(details in Appendix)*

Toy tests showed excellent statistical dependence minimization / maximization behavior→ approach is compatible with iterative tree boosting!

A realistic proof-of-principle made during Heavy Neutral Lepton search (event level BDT score constrained to be max-independent of a displaced jet NN-tagger score) →for ABCD bgk estimation.

A balance trade-off between maximum classification performance and mutual information minimization → **a problem bounded optimum**. However, there is much “slack” which can be squeezed, while retaining high performance and low correlation measures, *simultaneously*.

Event level ROCs (high is good)

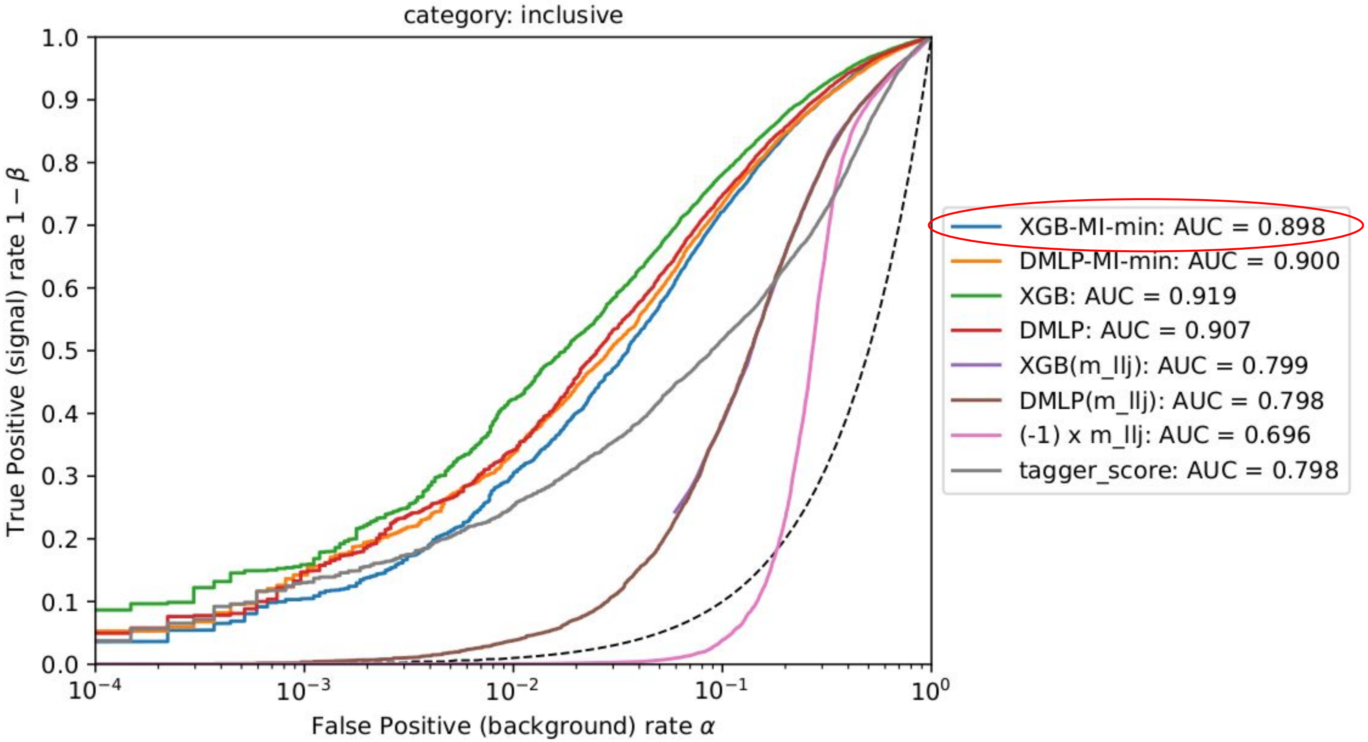
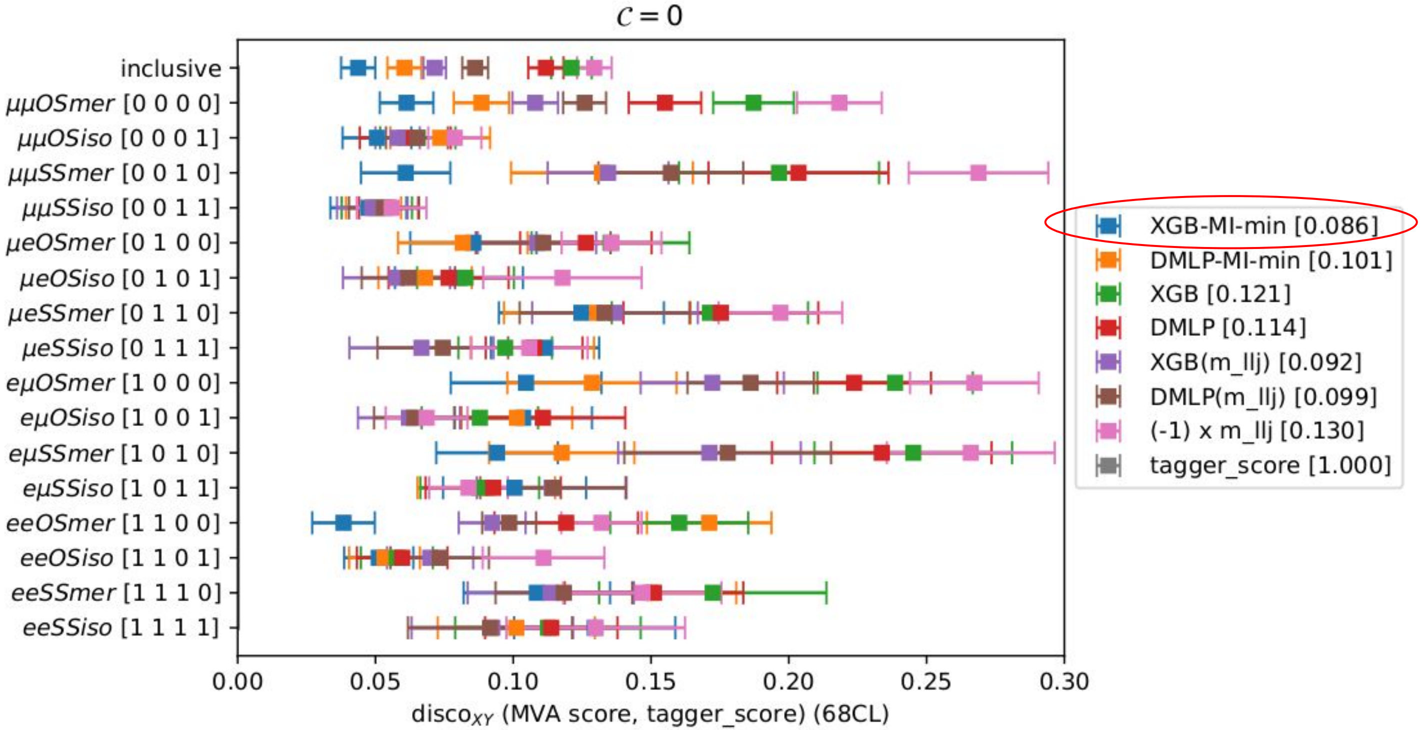


Figure: Event category: Inclusive. The dashed line is a random guess (AUC=0.5).

Distance correlation performance (low is good)



16 mutually exclusive final state categories:

e / mu ×
lepton pair charge ×
isolated / merged

Figure: $\mathcal{C} = 0$ (background). Non-linear Distance Correlation $\in [0, 1]$. Pay attention to the maximum outliers → ABCD problems.

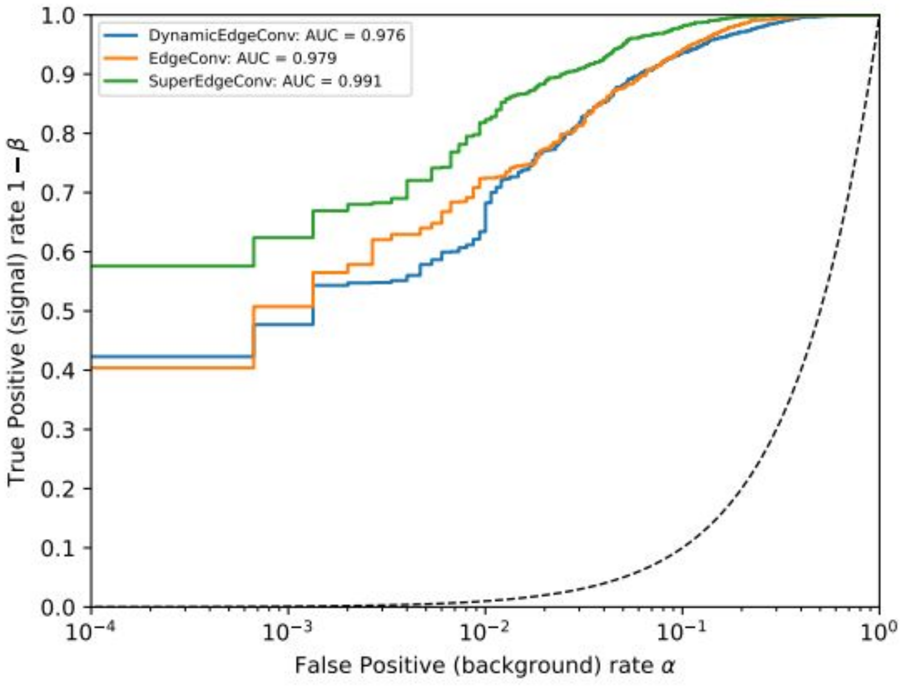
Novel models – SuperEdgeConv GNN

A simple generalization (superset) of **EdgeConv** (arxiv.org/abs/1801.07829)

- ❖ GNN message passing operator with Mandelstam invariants: $\boldsymbol{s} = (\boldsymbol{p}_i - \boldsymbol{p}_j)^2$, $\boldsymbol{t} = (\boldsymbol{p}_i - \boldsymbol{p}_j)^2$ … on graph edges spanned between input particle 4-momenta. Also include $\boldsymbol{z}_i * \boldsymbol{z}_j$ type operations in the latent $\boldsymbol{z}$-space (orig. only $\boldsymbol{z}_i - \boldsymbol{z}_j$)
- ❖ Both tricks motivated by how QFT scattering amplitudes are a function of local 2-point kinematics between final state legs.

→ **Empirically improved performance**, also faster than e.g. attention based

Electron (S) vs Pion (B) classification ROCs



HGCAL hackathon (summer 22)

EdgeConv, DynamicEdgeConv, SuperEdgeConv

Future?

Models: bunch of different methods under testing / development, later perhaps available in ICENET / separate repositories

- Conditional normalizing flows, diffusion (inverse problems / generative)
- GNN + Transformer based graph clustering (will talk about this @ CHEP23)

Workflow design: once new SW tech / practises pop out from AI industry, will update designs & workflows

Scaling up: investigate latest practises for multi-GPU / large scale deep training

Relational learnable models beyond graphs? See pyneuralogic.readthedocs.io

Thanks

Rob Bainbridge, Julia Dancu, Vilius Cepaitis, Kai Hong Law, Prijith Babu Pradeep, Jay Odedra, Max Hart, Alex Tapper, Oliver Buchmueller, Yuta Takahashi, Matthias Komm, Matthew Citron

For providing various input / problems to be solved / useful discussions / testing the code along different projects

Appendix

Other tools in ICENET

- Classic Tag & Probe efficiency and scale factor estimation via resonance fits
- Various minimal Python implementations; e.g. signal upper limits (asymptotic, MC), classic statistics problems etc.
- Autograd techniques explored for computing gradients, Hessians … of mathematical functions
- Various utility classes/functions, e.g. 4-vector algebra for graph constructions

https://github.com/mieskolainen/icenet/tree/master/icefit

https://github.com/mieskolainen/icenet/tree/master/icenet/tools

ICEBOOST details

Mutual Information

A generic measure of statistical dependence from Information Theory (invented by Shannon in the 1940s, using concepts from statistical mechanics). Let us have two random variables X and Y. They can be scalars or vectors. The mutual information is

$$\mathcal{MI}(X,Y) = \int dP_{XY} \log\left(\frac{p(X,Y)}{p(X)p(Y)}\right),$$

where $p(X,Y)$ is the full joint distribution, $p(X)$ and $p(Y)$ are the marginal distributions and the measure is $dP_{XY} \equiv dXdY\, p(X,Y)$. This is simply the Kullback-Leibler divergence between the joint and the product of marginal distributions.

Upper bound on MI is typically unknown, but the lower bound is 0, which is when X and Y are independent. This is a stronger statement than being uncorrelated (probed by linear Pearson correlations).

Donsker-Varadhan representation of Kullback-Leibler divergence D_{KL}

The practical problem is that MI is hard to compute for continous random variables – hard even with histograms. However, in a modern way, it can be estimated with neural networks. One neural MI-estimator[2] is MINE: (https://arxiv.org/abs/1801.04062)

This is based on Donsker-Varadhan representation based lower bound on MI:

$$\mathcal{MI}(X,Y) = D_{KL}(p(x,y)|p(x)p(y)) \geq \mathbb{E}_{p(x,y)}[T_\theta(x,y)] - \log(\mathbb{E}_{p(x)}\mathbb{E}_{p(y)}[e^{T_\theta(x,y)}]),$$

where T_θ is a near arbitrary integrable function attaining the empirical maximum of the DV-expression. This function T_θ we learn with a neural network! In training, the product of marginal distributions are obtained effectively simply by picking (mixing) variables from independent events (the right hand side of the last expression).

[2] I have implemented also an alternative strategy based on the "likelihood ratio trick", which learns a classifier to discriminate between unmixed XY 'signal' and mixed $X \otimes Y$ 'background'. Then I transform the classifier output P with $P/(1-P) \sim p(x,y)/[p(x)p(y)]$, and take the logarithm.

MI-regularized Total Loss

The total[3] loss function tries to balance between having a strong classifier but also providing simultaneously an output, which is independent of the external variable (HNL-tagger score). The loss function to be minimized is simply

$$\mathcal{L} = \mathcal{CE}[p, y] + \beta \mathcal{MI}[p, s],$$

where y is the true class label (signal $= 1$, background $= 0$), p is the class prediction $[0, 1]$, $\beta > 0$ is the regularization strength hyperparameter and s is the tagger score (could be also a vector random variable).

$\mathcal{CE}$ is the typical classification Cross Entropy loss (Bernoulli likelihood) and $\mathcal{MI}$ is the neural Mutual Information estimate.

[3] Note that the MI-network does have its own separate loss function

Tech details 1/2

- I use torch neural networks and torch autograd to compute gradients for xgboost by interfacing them with xgboost-custom loss function. Xgboost wants gradients $\partial_p \mathcal{L}$ and diagonal second order derivatives $\partial_p^2 \mathcal{L}$ (Hessian diagonal), i.e. it uses second order gradient descent.

- The Hessian diagonal is quite slow to compute by autograd in this context, because it needs to be computed for each event per boost iteration, and computed w.r.t the $\mathcal{MI}$-network → as a fast approximation set to identity (one). This neglects the loss curvature estimation which is compensated by the following boost iteration.

- Within pure torch, one could have gradients flowing "twice" within one minibatch and interleave the Classifier and MI-model autograd backward calls [`.backward(retain_graph=True)`], or do alternated training (seems more stable). Adaptive gradient norm clipping may be needed between the classifier and the MI-model parameter gradients.

Tech details 2/2

A sliced MI-loss, sum over mutually exclusive final state categories

$$\mathcal{L}_{\mathcal{MI}} \sim \beta\mathcal{MI} \rightarrow \frac{1}{|\mathrm{cat}|}\sum_{i\in\mathrm{cat}} \beta_i \mathcal{MI}_i, \tag{1}$$

where each β_i can be different, in principle. One can also try to weight each term according to category statistics (e.g. $\sqrt{N}$ significance based). This category expansion (can/will) make a difference, because $\mathcal{MI}$-term does not obey additivity – MI of a sum of distributions $\neq$ average over MIs per distribution – c.f. the classification $\mathcal{CE}$-term is additive.

This requires as many MI-networks / estimators as there are categories, e.g. 1 xgboost + 16 neural networks. Fully automated in the code to arbitrary number of categorical cuts via Boolean matrices.

N.B. in our application, only background events are selected in the loss function for the $\mathcal{MI}$-term, but naturally the $\mathcal{CE}$-term uses both background & signal events.