AI for HEP

Alexander Khanov

Oklahoma State University

PHYS 5503, April 30, 2025

Outline

- Overview of AI/ML
- HEP use case 1: flavor tagging
  - ATL-PHYS-PUB-2022-027, https://cds.cern.ch/record/2811135
- HEP use case 2: anomaly detection
  - Phys. Rev. Lett. 132 (2024) 081801

Credits: Jacob Crosby

Machine learning

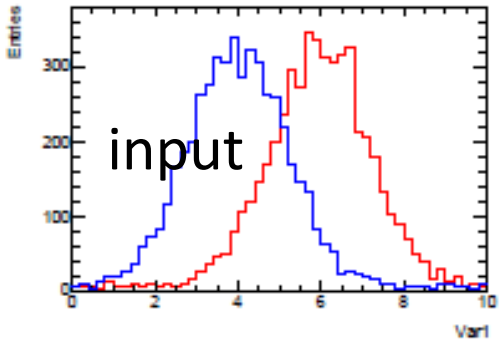
- Textbook definitions:
  - AI: ability of computers to perform cognitive tasks traditionally attributed to humans
  - ML: a subset of AI dedicated to teaching computers to extract patterns from data
- These definitions are not ideal for HEP applications
  - classical AI applications like quality assurance: humans can do it
  - pattern recognition and big data processing: humans can’t do it (because of data precision and/or size)

ML: formulation of the problem

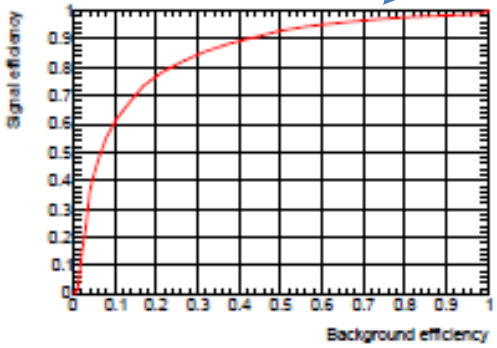
- Consider a multivariate function $\boldsymbol{y}=\boldsymbol{f}(\boldsymbol{x},\boldsymbol{p})$ ("model")
  - $\boldsymbol{x}$ (a vector): inputs ("features")
  - $\boldsymbol{y}$ (in general, a vector): outputs ("labels"), can be binary ("classification") or continuous ("regression"), simplest case is a single binary output (signal/background separation)
  - $\boldsymbol{p}$ (a vector): parameters
- The goal is to find $\boldsymbol{p}$ that optimizes $\boldsymbol{f}$ ("train the model")
  - given a set of $\{(\boldsymbol{x}_k,\boldsymbol{y}_k)\}$ pairs ("training set"), find $\boldsymbol{p}$ that minimizes the distance between $\boldsymbol{f}$ and $\boldsymbol{y}$ ("loss function")
  - a possible (but not the only one) choice for distance is mean squared loss $\chi^2=\Sigma_{jk}(y_{jk}-f_j(\boldsymbol{x}_k,\boldsymbol{p}))^2$
  - the hope is that, once trained, the model will correctly predict $\boldsymbol{y}$ for any $\boldsymbol{x}$

ML methods

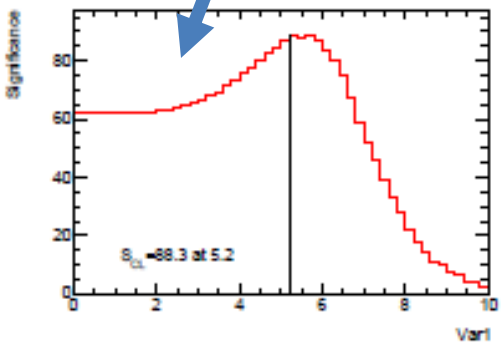
- Straightforward separation



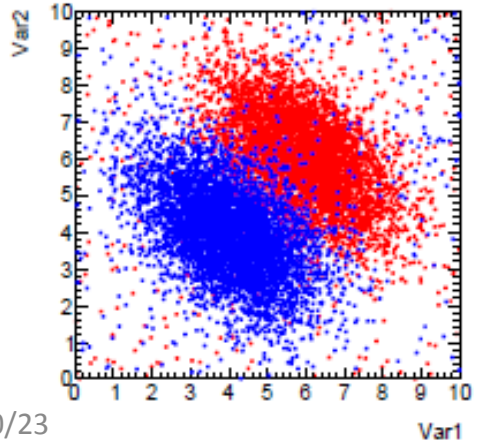
ROC curve

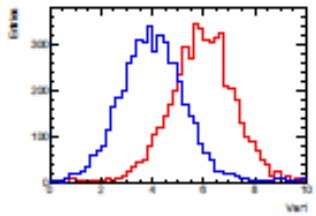


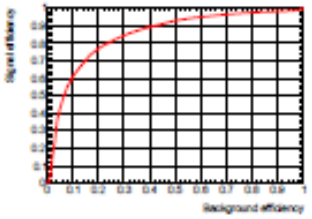
S_{CL}

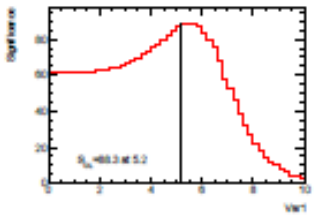


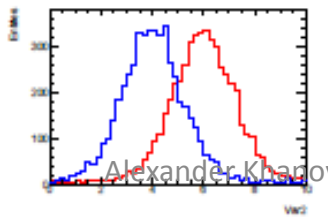
  - Can be done in many dimensions

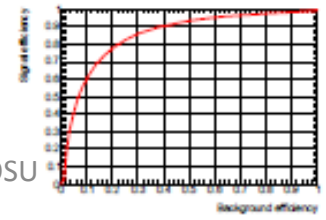


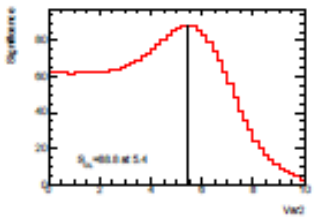






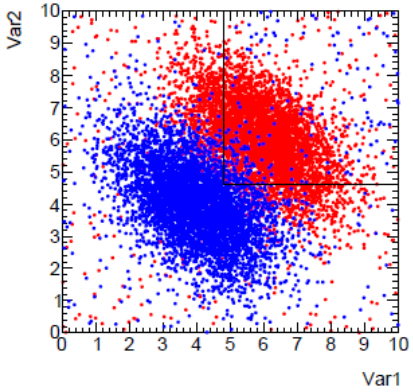


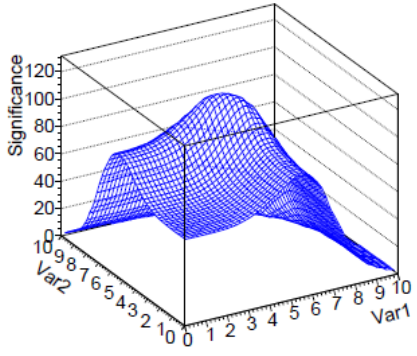


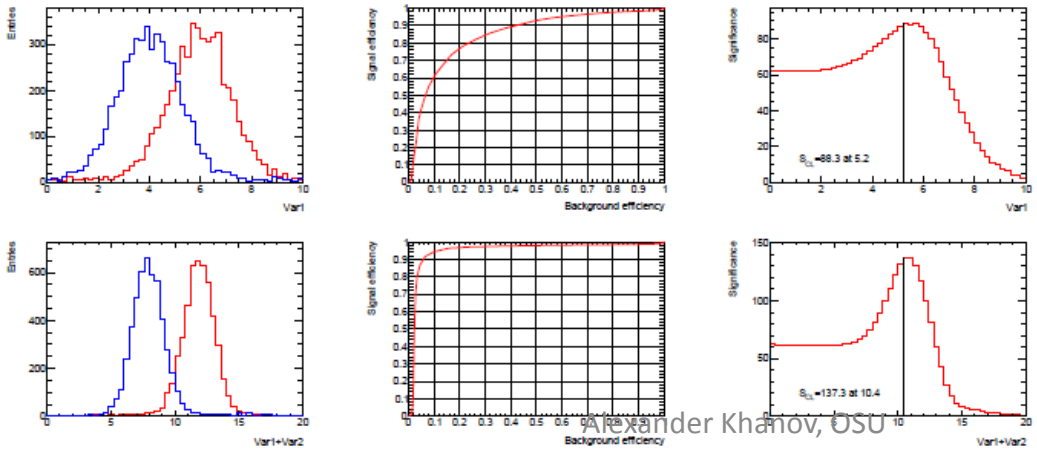


ML methods (2)

- Grid search (a.k.a. cut and count)
  - try all combinations of thresholds, pick the best
- Linear discriminants
  - Fisher discriminant: $f(\boldsymbol{x},\boldsymbol{p}) = \Sigma_i(p_i x_i)$

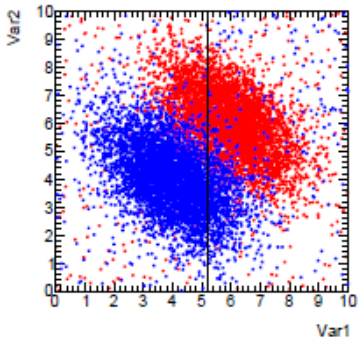


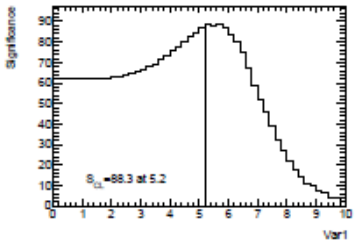




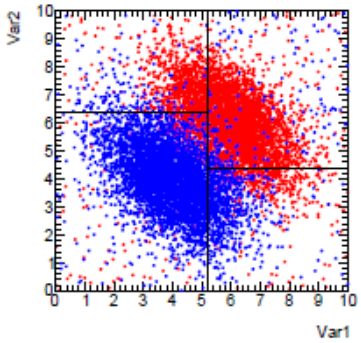
in our example, the optimal choice is $f=x_1+x_2$

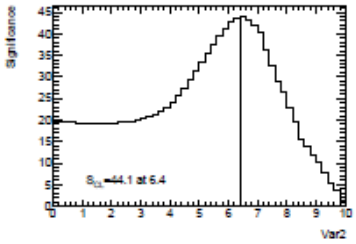
ML methods (3)

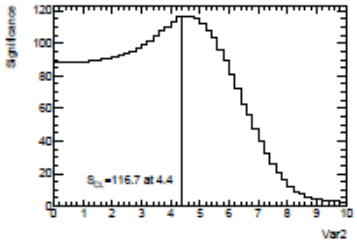




- Decision trees: optimize one threshold at a time



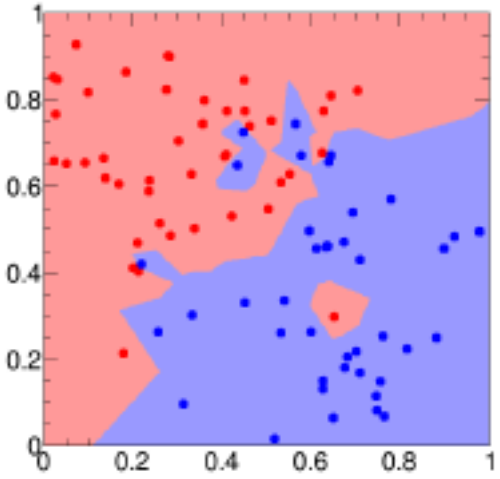




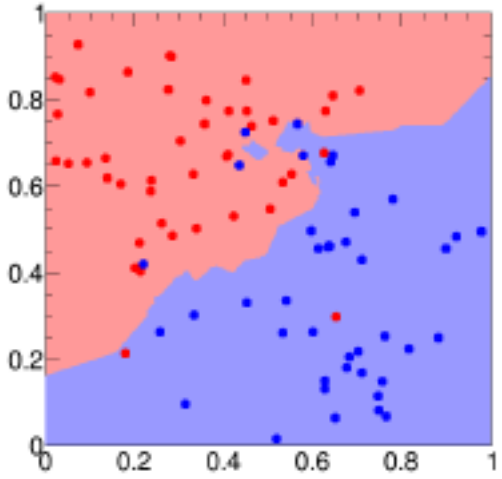
- Boosted decision trees: create many trees, take as the output their weighted sum

ML methods (4)

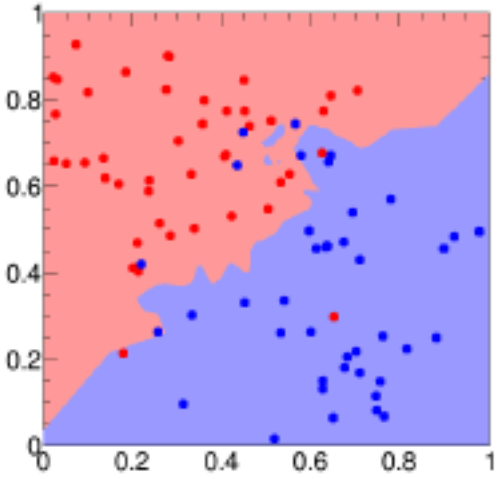
- k nearest neighbors
  - a non-parametric method: the number of parameters grows with the training set size
  - efficient for complicated topologies
  - can be used for regression



$k = 1$



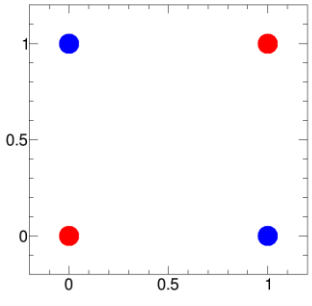
$k = 3$

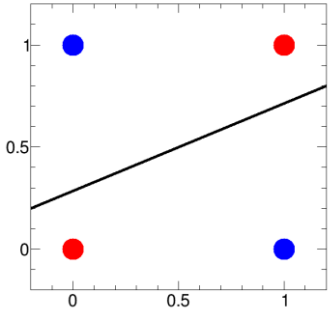


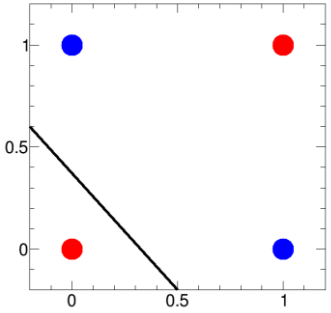
$k = 5$

Case for better ML methods

- The methods we considered so far are often not able to solve trivial problems
  - canonical example: the XOR problem (0,0)->0, (1,0)->1, (0,1)->1, (1,1)->0







not linearly solvable

Many ML methods fail when the number of inputs becomes large: can't evaluate statistical density for cells without training examples

Simple neural networks

- A simple NN consists of “layers” which are collections of “neurons”
  - a neuron *i* in layer *j* is characterized by its “activation value” a_i^j

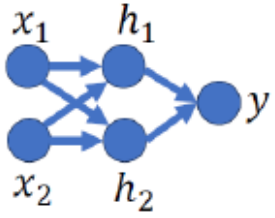
  $a_i^{j+1} = \sigma(\Sigma_k(w_{ik}^j\, a_i^j) + b_i^j)$
  - σ = “activation function”, *w* = “weights”, *b* = “biases”
- In order for NN to work, some σ must be non-linear

Simple neural networks (2)

- The XOR problem: linear activation function
  - $y = w_1 x_1 + w_2 x_2 + b$
  - minimizing, we'll get $w_1=w_2=0$, $b=0.5$ – bad ☹

- ReLU activation: $R(t) = \begin{cases} 0, t < 0 \\ t, t \geq 0 \end{cases}$

$$\begin{cases} h_1 = R(W_{11}x_1 + W_{21}x_2 + b_1) \\ h_2 = R(W_{12}x_1 + W_{22}x_2 + b_2) \\ y = w_1 h_1 + w_2 h_2 \end{cases}$$



- $W = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} 0 \\ -1 \end{pmatrix}$, $\mathbf{w} = \begin{pmatrix} 1 \\ -2 \end{pmatrix}$ does the trick:

x_1	x_2	h_1	h_2	y
0	0	0	0	0
1	0	1	0	1
0	1	1	0	1
1	1	2	1	0

Simple neural networks (3)

- An NN can solve any problem: universal approximation theorem
  - arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with a single hidden layer and any continuous sigmoidal nonlinearity

namely, if σ is a "sigmoid" ($\lim_{t\to-\infty}\sigma(t)=0$, $\lim_{t\to\infty}\sigma(t)=1$), then for any continuous function $f(\mathbf{x})$ defined on a unit cube, for any ε there is a sum of form

$$G(\mathbf{x})=\sum_{j=1}^{N}\alpha_j\sigma(\mathbf{y}_j^{\mathrm{T}}\mathbf{x}+\theta_j)$$

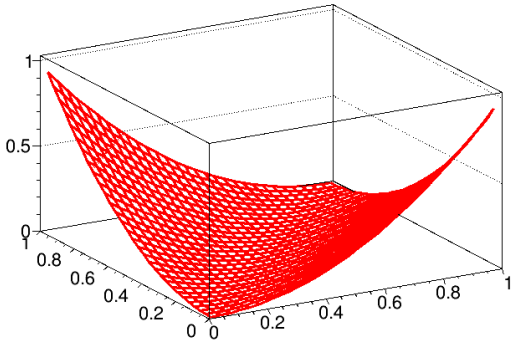
such that $|G(\mathbf{x})-f(\mathbf{x})|<\varepsilon$ for all $\mathbf{x}$

NN training

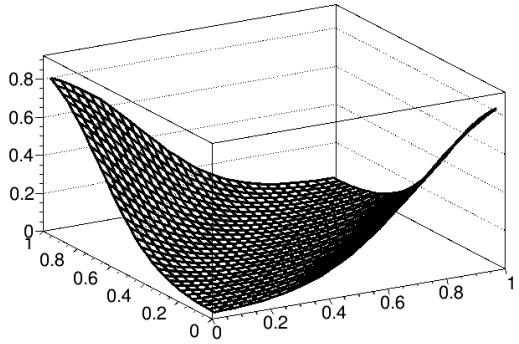
- In general, any optimization algorithm would work
- XOR problem example:

  $f(x_1, x_2)=(x_1-x_2)^2$

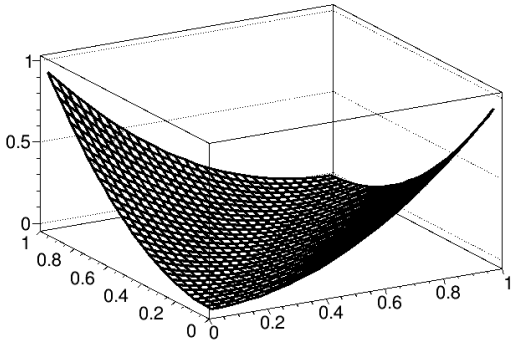
  – consider a NN with n neutrons in a single hidden layer



target function



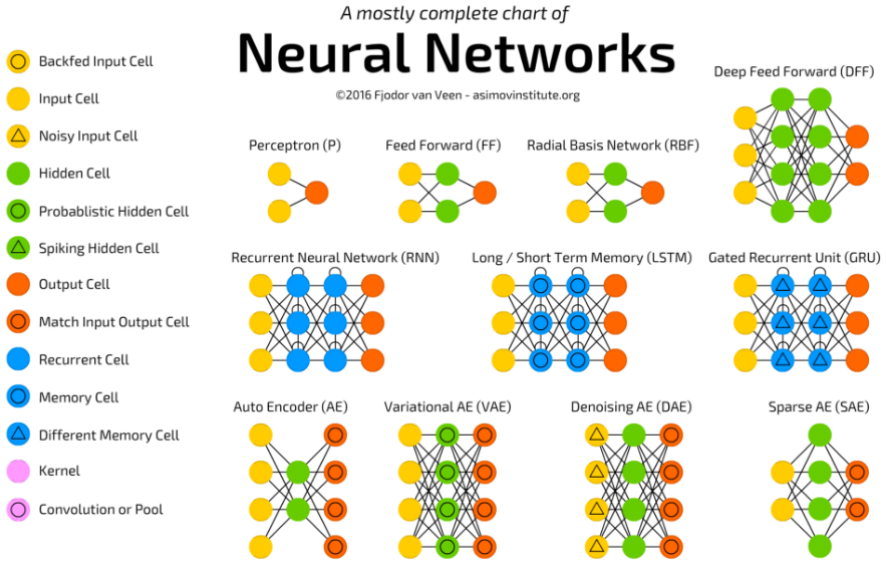
NN, n=2

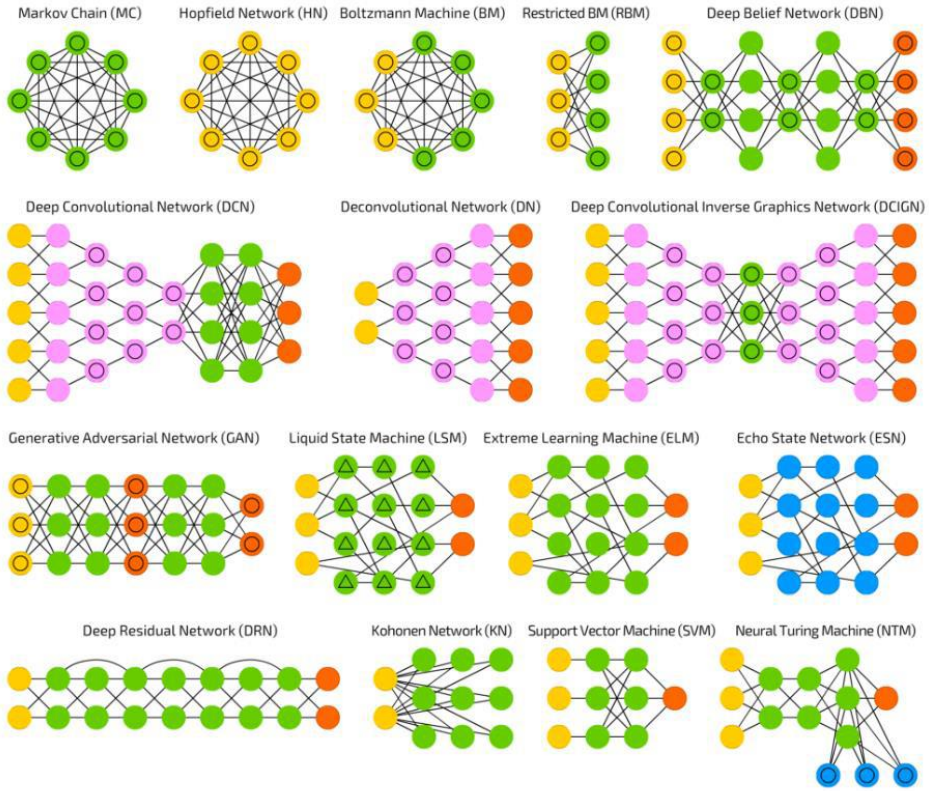


NN, n=4

- In practice, large NNs are trained using iterative gradient-based optimizing procedures
- Popular frameworks: TensorFlow / PyTorch

There are many NN types!





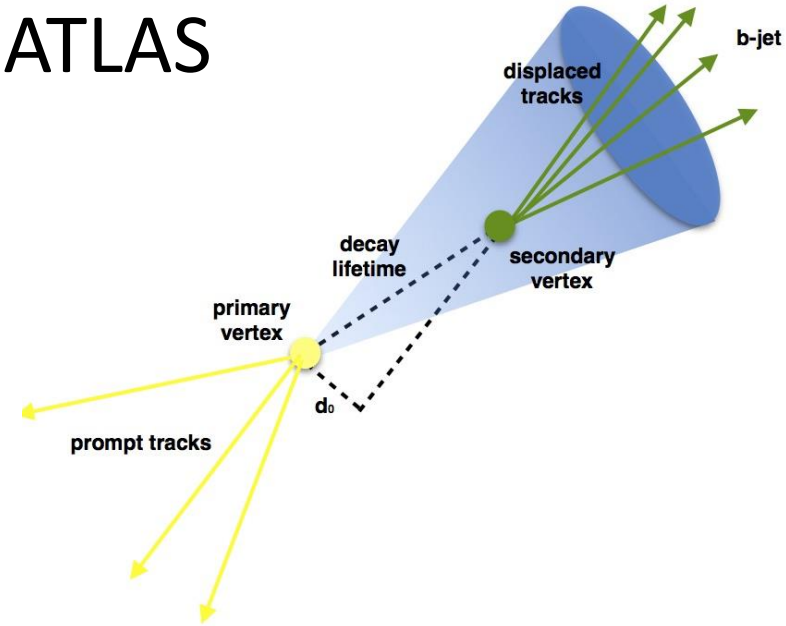
- a picture popular several years ago

HEP inputs

- The data from particle collisions arrives in a form of “events”
  - an event = a snapshot of detector response taken when there seems to be something interesting
- While it is possible to apply ML techniques to raw data, a typical HEP analysis deals with “objects”
  - muons
  - electrons
  - individual particles (“tracks”)
  - jets: collimated bunches of particles due to quarks or gluons
  - missing momentum: indicates presence of neutrinos or BSM
- Objects are mainly represented as 4-vectors, but may have other properties (e.g. EM fraction for jets)
- The number of objects of each type typically varies
  - if using NNs, need to apply “padding” (fix the maximum number of objects of a given type and replace the rest with zeros)
- The order of objects doesn’t matter (recurrent NNs don’t work)

Use case 1: flavor tagging

- Jet tagging as an example of HEP pattern recognition
- Evolution of ML algorithms
- GN1 and GN2 taggers in ATLAS



Jet flavor tagging 101

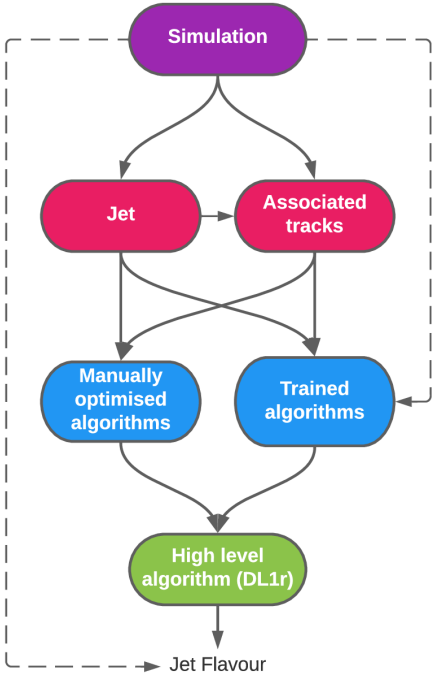
- Jet flavor tagging: identification of jets originating from b- and c-quarks
  - a b-jet is a jet that contains a B-hadron (decided by ghost association)
  - a c-jet is a jet that does not contain B-hadrons but contains a D-hadron
  - all other jets (except τ-jets) are called light
- Flavor tagging algorithm: a method to detect b- and c-jets
  - relies on significant lifetime of B/D hadrons
  - two main approaches based on either tracks with large impact parameter (IP) or explicit reconstruction of secondary vertices (SV)
- Performance of tagging algorithms is characterized by b-tagging efficiency ε_b (probability to correctly identify a b-jet) vs mistag rate ε_l (probability to misidentify a light jet as a b-jet)
  - ε_l as a function of ε_b is known as a ROC curve
  - similarly, the c-tagging performance is described by a ε_c vs ε_b ROC curve
- To be useful for physics analyses, the performance of the tagging algorithm needs to be calibrated against real data
  - since the calibration procedure is cumbersome, only a few points on the ROC curve (working points, WP) are used
  - physics analyses pick a WP that best suits their needs

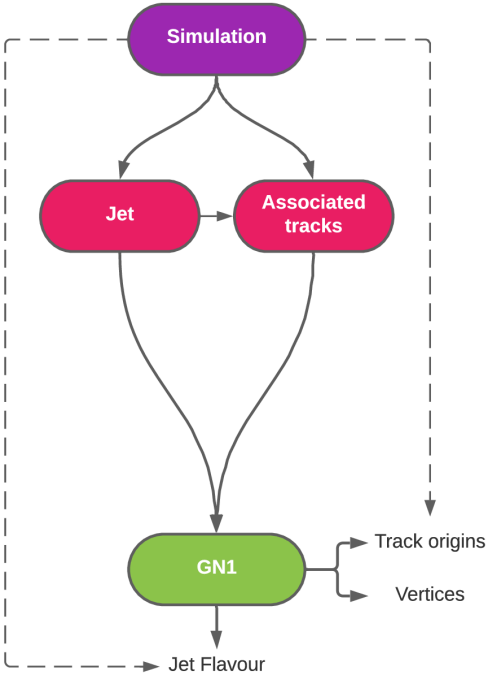
Evolution of tagging algorithms

- Low level algorithms: IP3D, SV1, JetFitter
  - likelihood based algorithms looking at track IPs or SVs
  - limited consideration of track parameter correlations
- Combinations of low-level algorithms: IP3D+SV1
- Multivariate combination of low-level algorithms based on boosted decision trees: MV2
- Neural Network combination: DL1 and its flavors
  - DL1: original (IP3D+SV1+JetFitter)
  - DL1r: IP3D replaced with recurrent NN (RNNIP)
  - DL1d: RNNIP replaced with deep sets NN (DIPS)
- Graph Neural Network: GN1

GN1 overview

- GN1: graph NN with direct track input
- Why the new algorithm?
  - improved performance (of course)
  - flexibility: no need to re-optimize low-level taggers for a new task (Xbb, c-tagging,…)
  - better insight into tagging process (auxiliary vertex and track origin predictions)



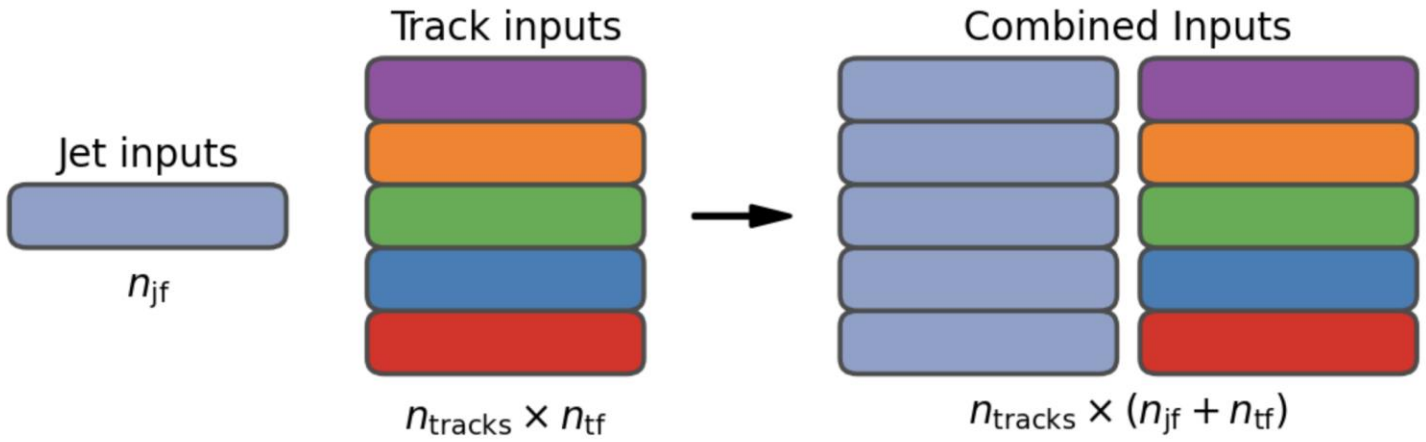


GN1 model

- Inputs: two jet variables (p_T, η) and n_{tracks}×21 tracking variables ($n_{tracks} \leq 40$)
  - five track parameters + their uncertainties (q/p, direction relative to the jet axis, track IP in transverse and longitudinal plane)
  - hit patterns
  - (optional) lepton track ID

rescaled to mean=1, var=1

- Labels: jet flavor (b, c, light)
- Auxiliary training objectives
  - track origin (pileup, fake, primary, b, b→c, c, other secondary)
  - track-pair vertex compatibility (binary)



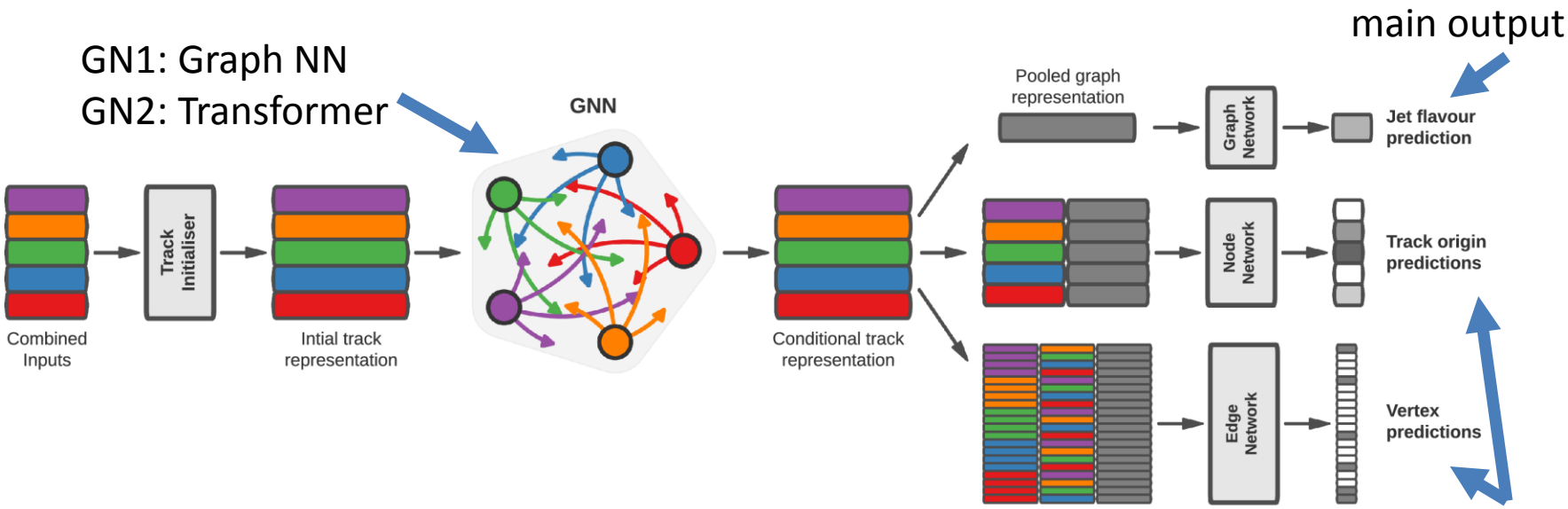
Architecture

- Inputs are fed into a per-track initialization network (3 hidden + 1 output layer ×64 neurons)
- Outputs (latent track representations) are used to populate a fully connected GNN (a node = a track)
- Resulting node representations are fed to classification networks

Network	Hidden layers	Output
Node	128,64,32	7
Edge	128,64,32	1
Graph	128,64,32,16	3

GN1: Graph NN

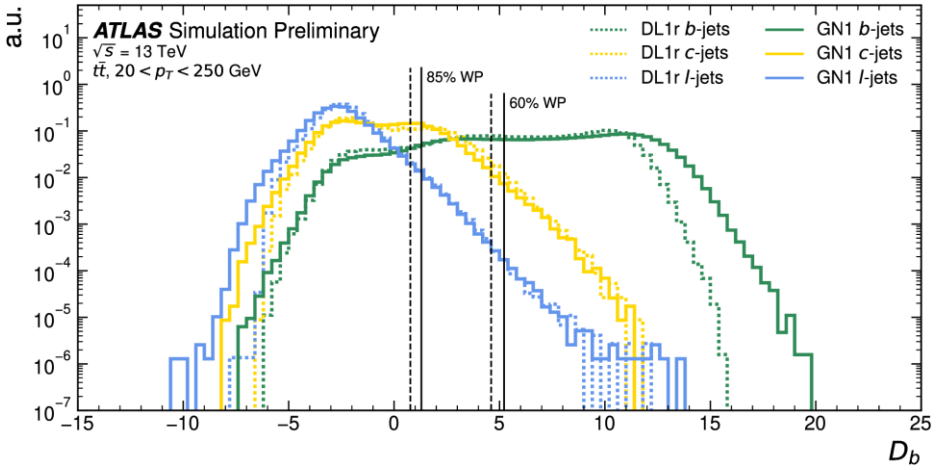
GN2: Transformer



Training

- MC samples: $t\bar{t}$→l+jet/dileptons, Z'→$q\bar{q}$ (flat jet p_T up to 5 TeV, equal $b\bar{b}$/$c\bar{c}$/light)
  - 30M jets (60% $t\bar{t}$ + 40% Z')
- Goal: minimize total loss $L_{total}=L_{jet}+\alpha L_{vertex}+\beta L_{track}$
  - L_{jet}: categorical cross entropy loss over jet flavors
  - L_{vertex}: binary cross entropy loss averaged over track pairs
  - L_{track}: categorical cross entropy loss over track origins
- Optimal choice: $\alpha=1.5$, $\beta=0.5$
  - verified that the algorithm works with L_{jet} minimization alone

Performance



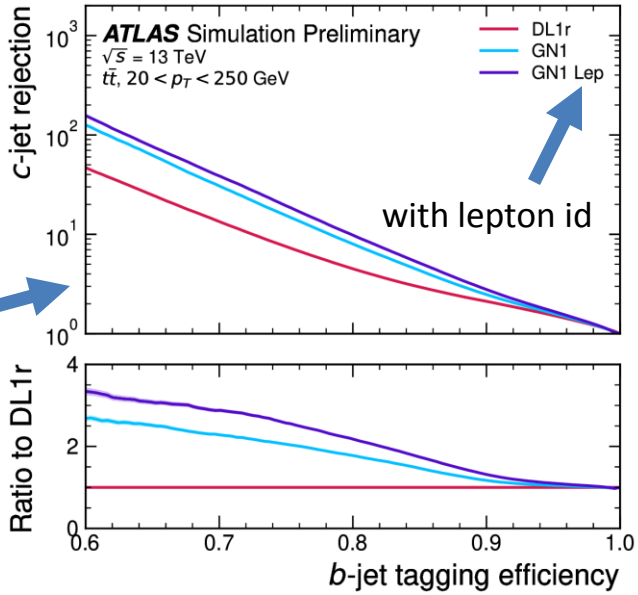
b-tag score: $D_b = \log \frac{p_b}{(1-f_c)p_l + f_c p_c}$

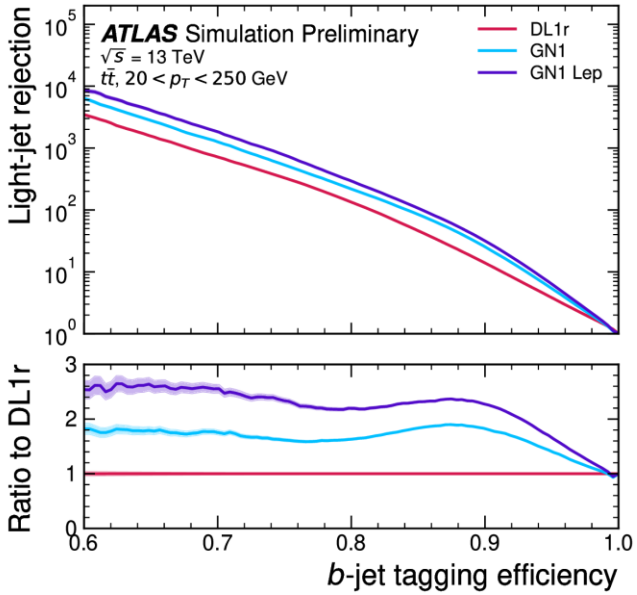
fraction of c-jets (0.05)

60% WP

70% WP

85% WP



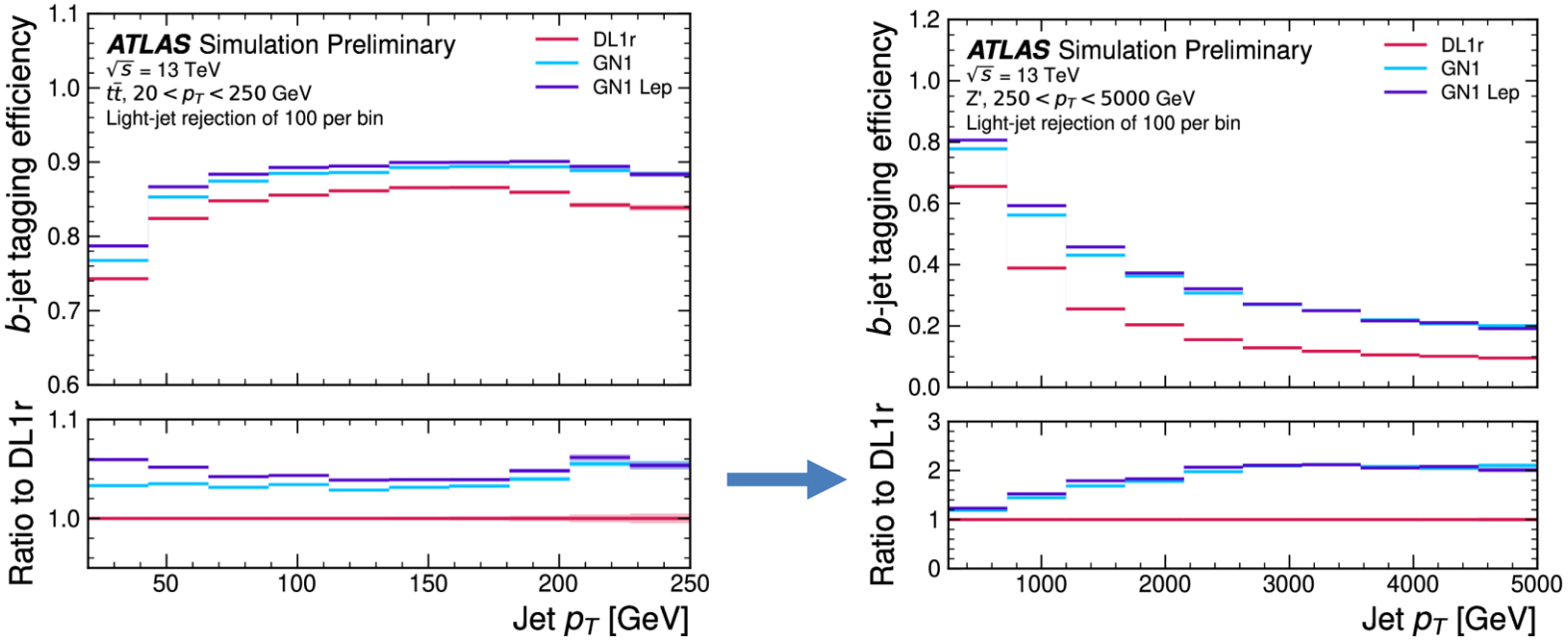


significant improvement in both light and c-jet rejection (×~2 at 70% WP)

ATL-PHYS-PUB-2022-027

Performance (2)

- Improvement in b-tagging efficiency at fixed mistag rate (0.01) is particularly significant at large jet p_T
- Vertexing performance: inclusive b-vertex reco efficiency ~80%
- Track classification performance: weighted AUC ~0.95

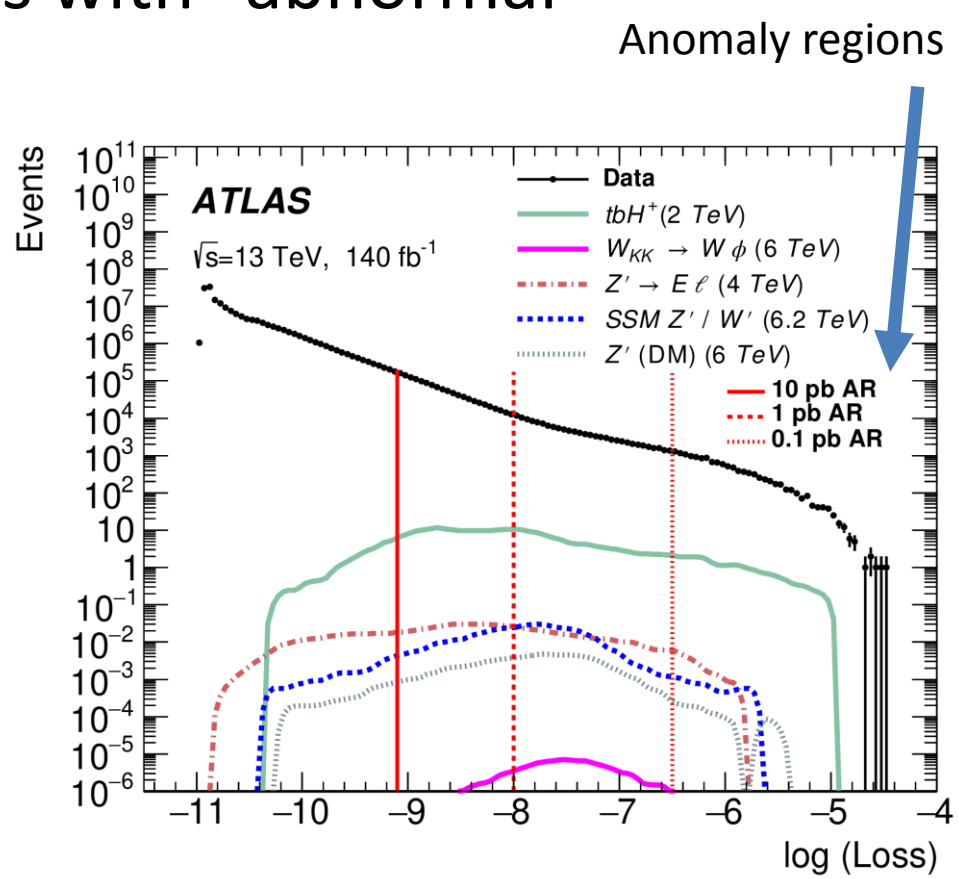


Flavor tagging summary

- GN1 is a novel jet tagger based on graph NN architecture and trained with auxiliary training targets
  - shown to significantly improve flavor tagging performance compared to the current ATLAS base line tagger (DL1r)
- Flexible, easier to optimize, simpler to maintain
- Demonstrates improved track classification performance and high b-tagging vertex finding efficiency
  - looking for further improvements due to in-depth tuning of NN configurations, better modeling (simulation of interaction of B/D hadrons), improved pileup jet rejection

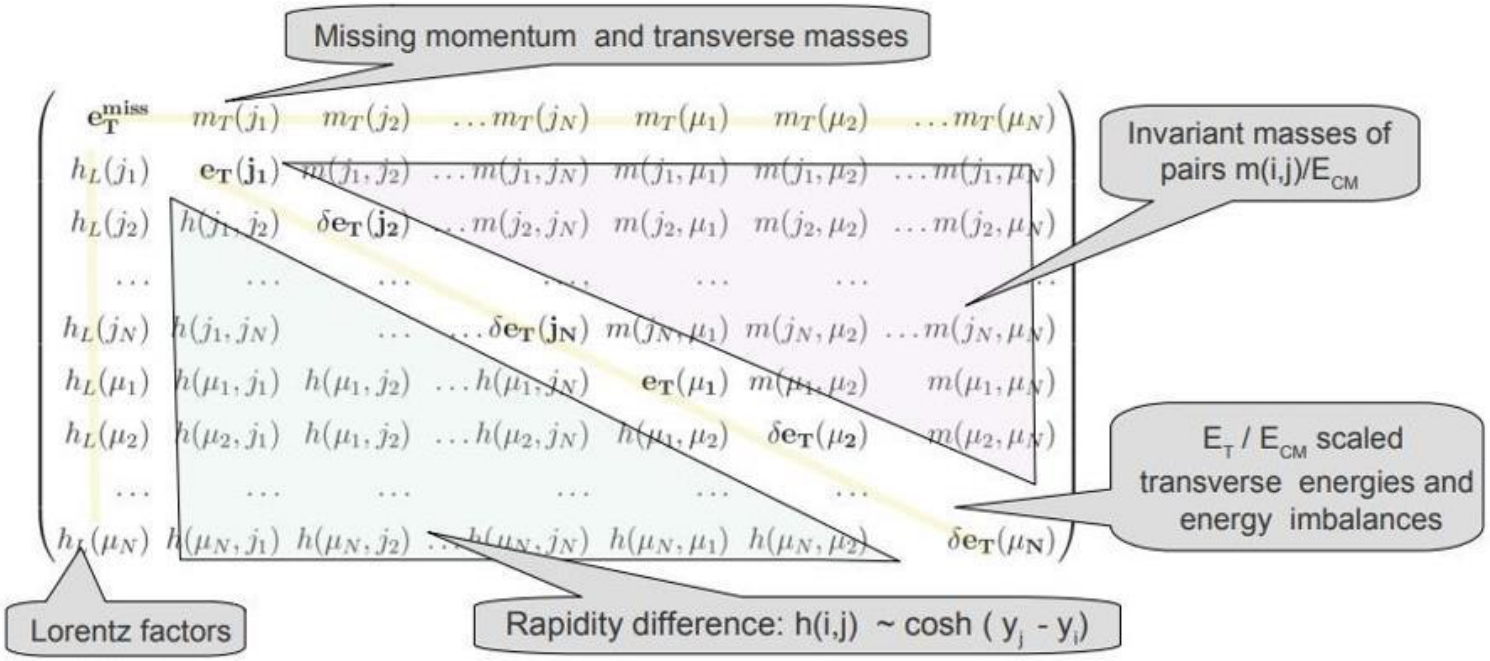
Use case 2: search for new physics using anomaly detection

- The idea: look for events with “abnormal” kinematics
  - train the NN on 1% of all data, obtain the loss function
  - For events with abnormally large loss, fit the invariant mass distributions with a smooth curve, look for bumps



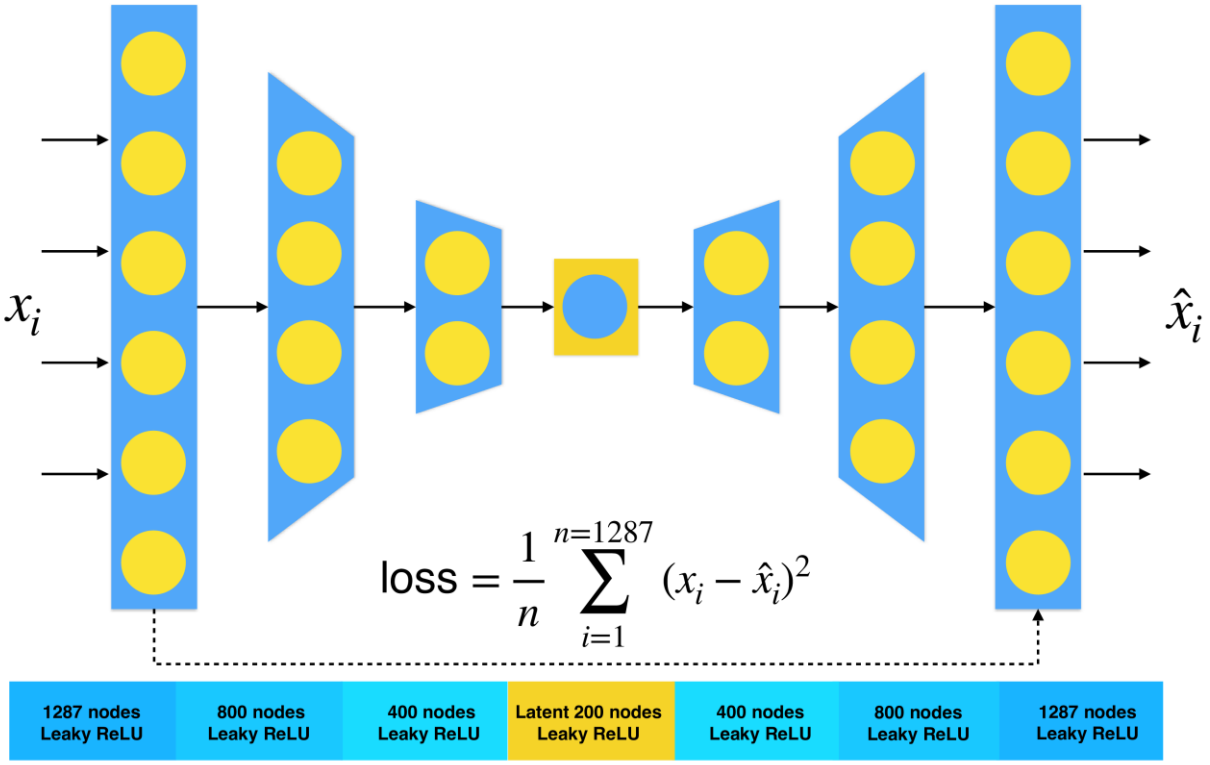
AD: inputs

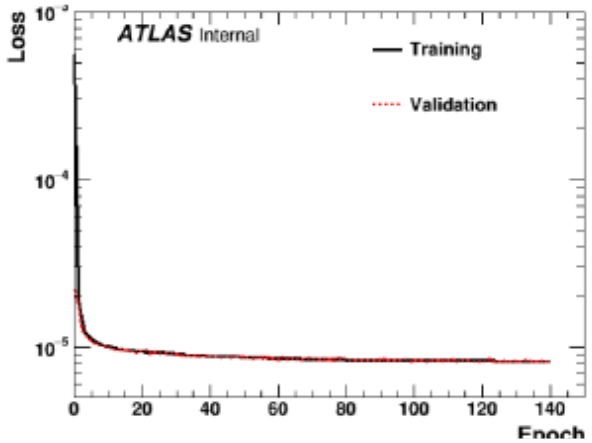
- Input space: rapidity mass matrix (NIM A 931 (2019) 92)
  - Lorentz invariant under boosts along the longitudinal axes
  - single and two-particle densities for each particle or jet
  - dimensionless, normalized to (0,1)
  - fixed size

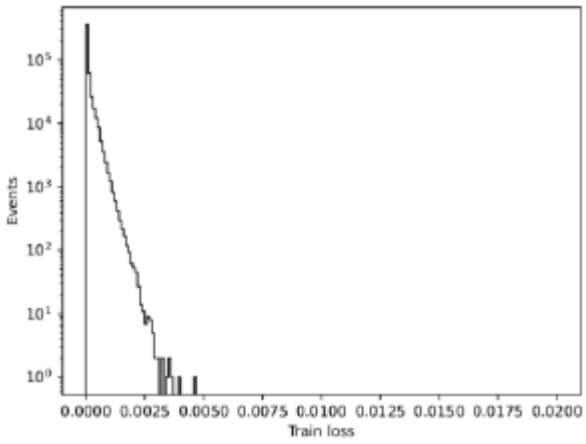


AD model: AutoEncoder

- 1287 inputs/outputs (RMM values except inv. masses)
- Leaky RELU activation for all layers

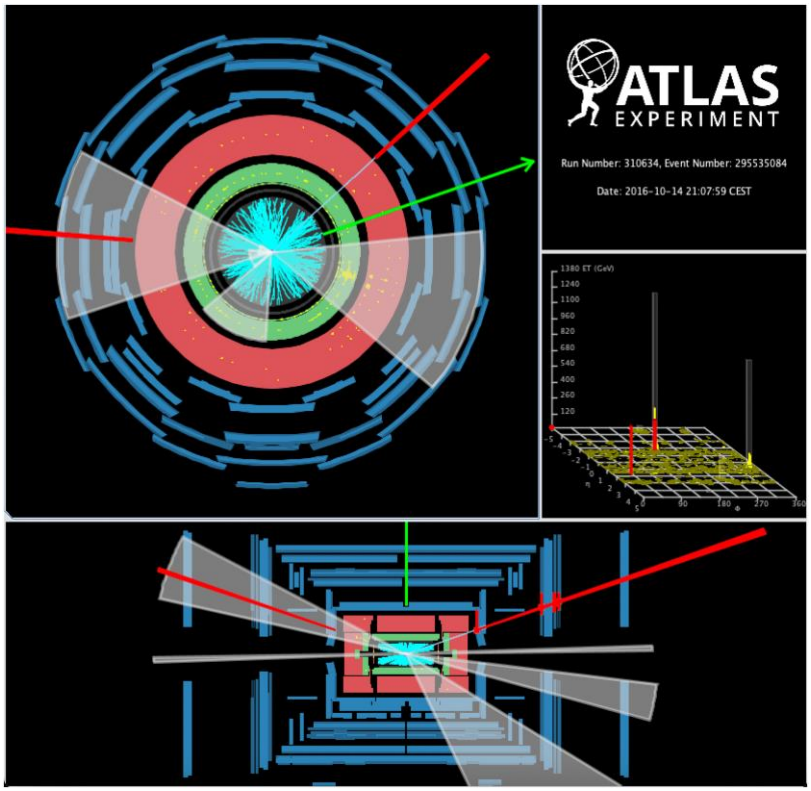


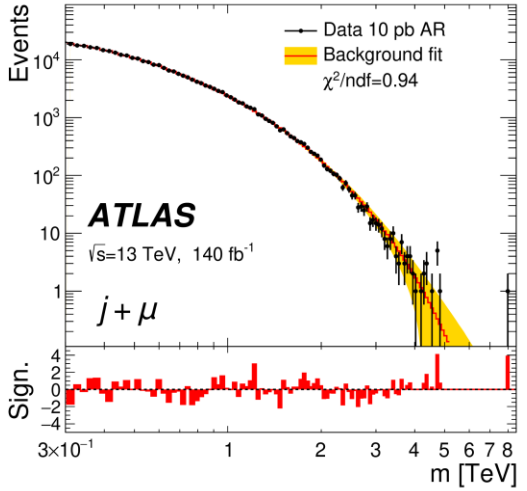


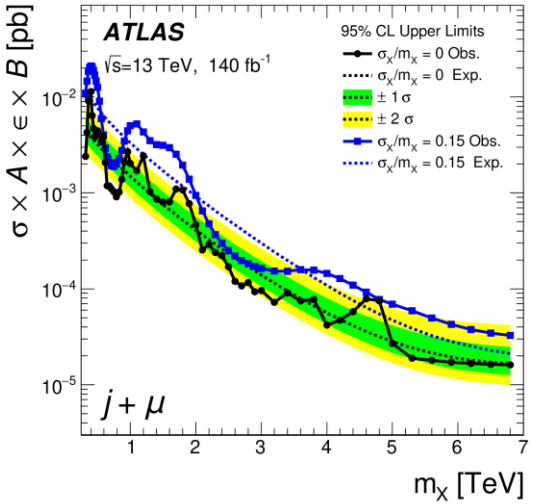


AD results

- No BSM discovery, quite a few interesting events







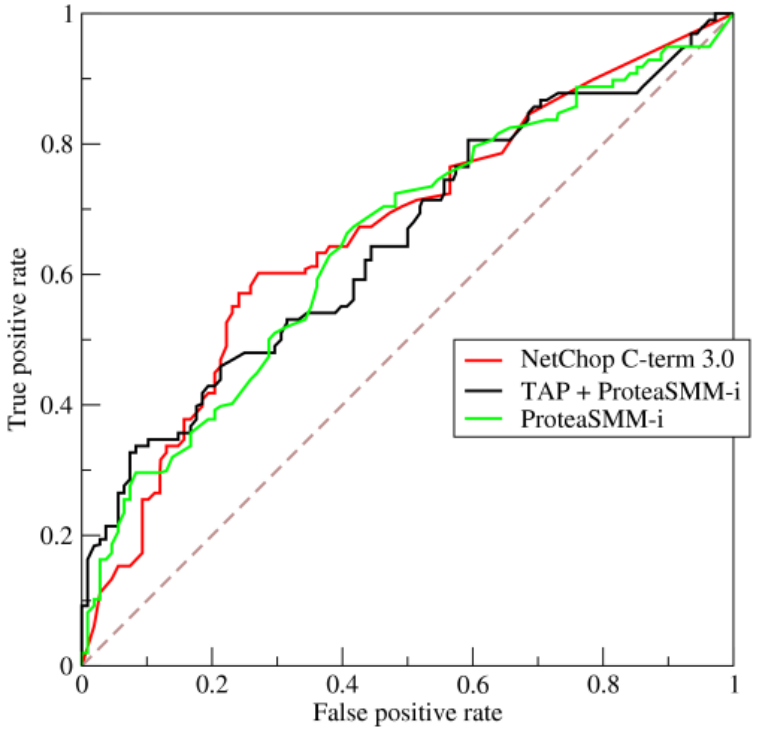
AD summary

- The discovery sensitivity shows a large improvement after the anomaly region selection, which is illustrated using several benchmark BSM models
- The analysis sets 95% CL upper limits on contributions from generic Gaussian signals to the studied invariant mass distributions
- Compared to previous limits without anomaly region selections, the reported model-independent limits have a stronger potential to exclude generic heavy states with complex decay modes

Backup

ROC curves

- ROC = Receiver Operation Characteristic
- Developed by electrical engineers during WWII to characterize radar operation



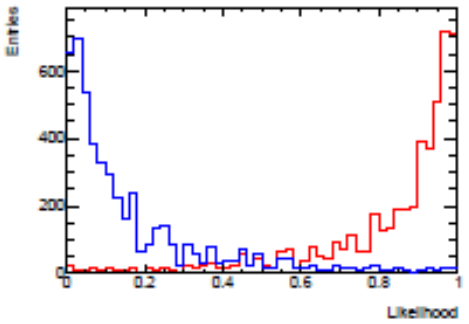
S/B separation figures of merit

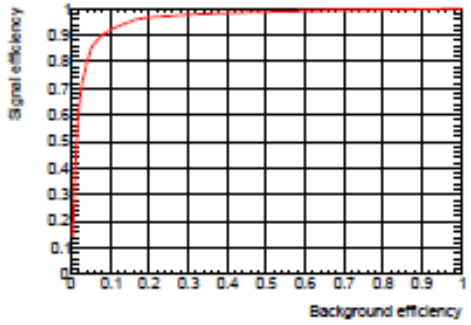
- Approximate figure of merit: significance $S = s/\sqrt{b}$
- A better figure of merit: optimizing the likelihood ratio L(S+B)/L(B)

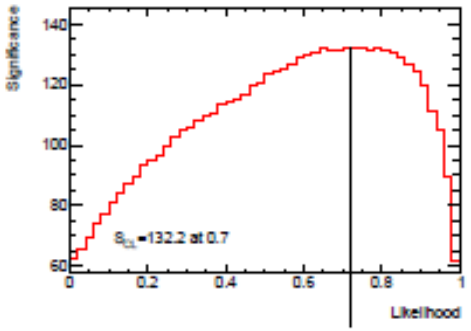
$$S_{\mathrm{CL}} = \sqrt{2((s+b)\ln(1+s/b) - s)}$$

Likelihood

- Likelihood: $L=s/(s+b)$,
  $s=\Pi(x_i)$, $y=1$
  $b=\Pi(x_i)$, $y=0$
- Doesn't consider correlations – this can be fixed by diagonalizing the inputs

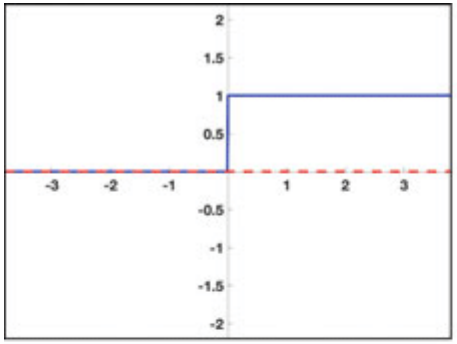




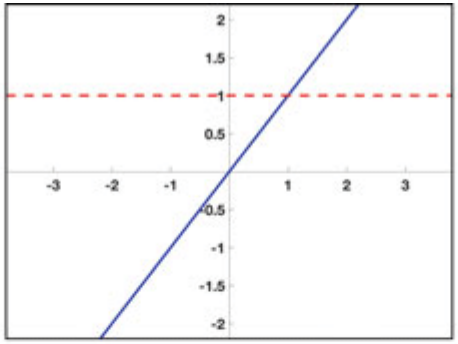


Popular activation functions

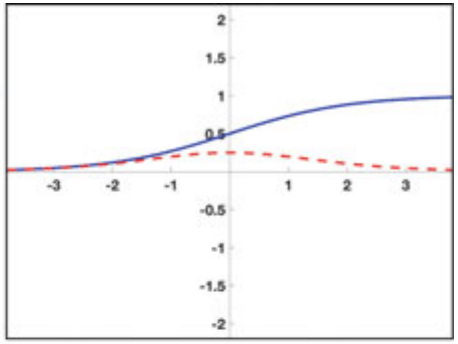
Step



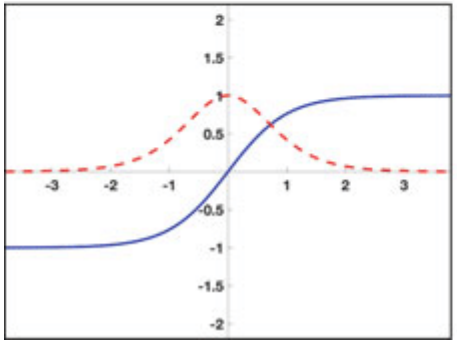
Linear



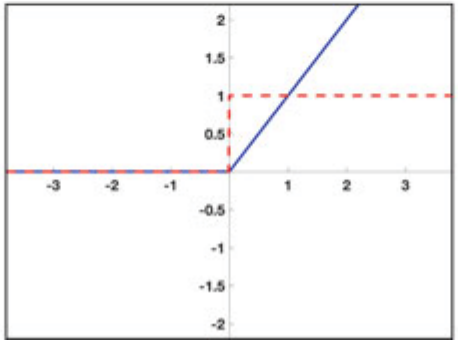
Sigmoid



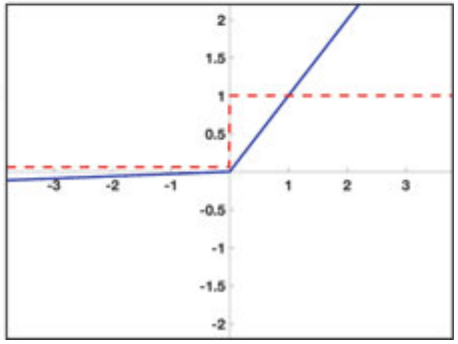
tanh



Rectified Linear Unit



Leaky ReLU



Loss functions for classification

- Categorical cross-entropy loss (a.k.a. softmax) for N observations, $C > 2$ exclusive classes, where $y_{ij} = \{0, 1\}$ is true label, and p_{ij} is predicted probability for observation $i = 1 \dots N$, class $j = 1 \dots C$:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} y_{ij} \log(p_{ij})$$

- Binary cross-entropy loss for N observations, where $y_i = \{0, 1\}$ is true label, and p_i is predicted probability for observation $i = 1 \dots N$: special case for categorical cross-entropy loss with two exclusive classes

$$L = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$