Modern Machine Learning in HEP

8th BCD ISHEP Cargèse School

27-31 March 2023

Julien Donini – Université Clermont Auvergne / LPC

Introduction

Machine learning is statistics + algorithms + computing power

HEP: used since the 80’s : old style **NN**, then **Boosted Decision Trees** “era”

Modern Machine learning since about 10 years

→ **Breakthrough** ideas supported by statisticians, computer scientists, etc

→ Increasing computing **power** to run efficiently **complex** algorithms

Powerful tool which enables us to do better but also **new kinds of physics**

2014 Kaggle Higgs **Challenge** (https://www.kaggle.com/c/higgs-boson)

- Improve measurements of **Higgs** decaying to pair of tau leptons
- 1800 participating teams (physicists, statisticians, computer scientists).

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

Winning solution performance equivalent to having **6 times more data** !

Outline

Part1: Modern ML for HEP

- Introduction to ML
- Example of advanced methods
- Differentiable programming

Part 2: Neural Networks and variational inference

Data

parameters

Output

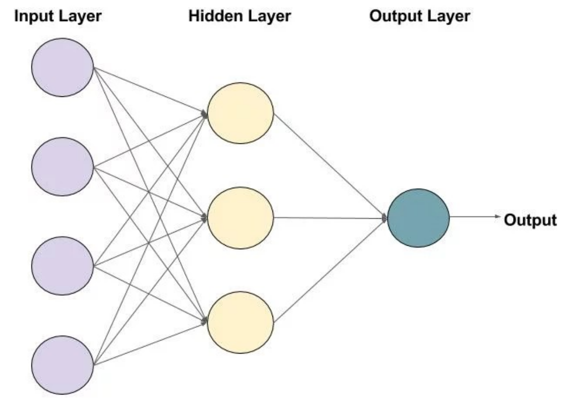
$$y(x,{\color{blue}w}) = {\color{blue}t}$$

Target

Training Neural Networks

Data

(features **x**)



Objective

(target **t**)

$\boxed{\mathbf{y} = f(\mathbf{x}, \mathbf{W})}$

f : non-linear functions

W : parameters (weights)

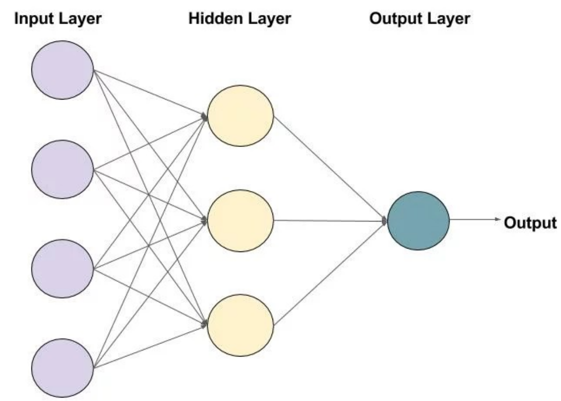
Cost function

"how close is the network output to the objective ?"

$$\ell(\mathbf{y}, \mathbf{t})$$

Training Neural Networks

Data
(features **x**)



Objective
(target **t**)

Update of weights **W**

$$\mathbf{W} \rightarrow \mathbf{W} - \eta \sum_{N} \frac{\partial \ell(\mathbf{y}, \mathbf{t})}{\partial \mathbf{W}}$$

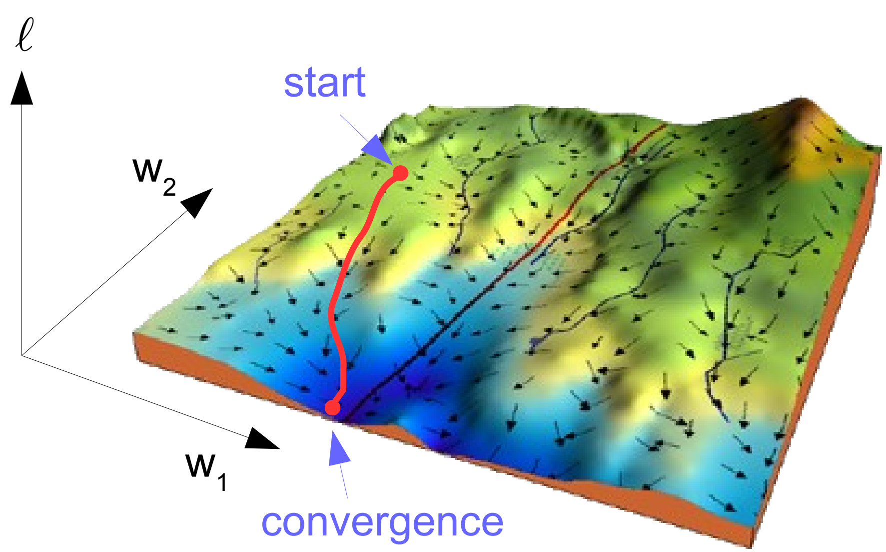
Gradient descent

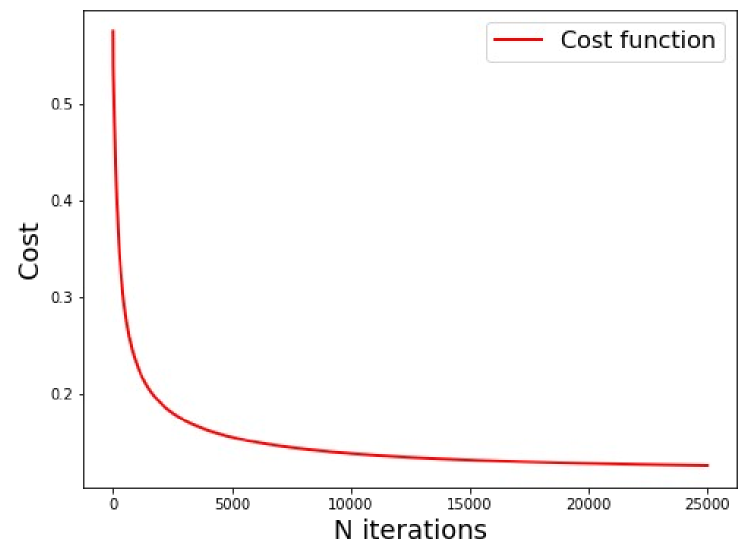
Start from initial set of weights **w** and subtract gradient of ℓ iteratively:

$$\boxed{\mathbf{W}^k \rightarrow \mathbf{W}^{k+1} = \mathbf{W}^k - \eta \sum_N \frac{\partial \ell(\mathbf{W}^k)}{\partial \mathbf{W}}}$$

k: iteration, η: learning speed

Repeat until convergence.





Modern ML

Data **X** drawn from some unknown **distribution** p(x)

Data

parameters

Output

$y(x, w)$

Learn p(x)

Learn p(x)

Learn p(x) **itself** → density estimation, eg Normalizing Flows

Conditional densities p(x | y) → conditional density estimation

Sampling from p(x) → generative modeling, eg GANs, VAEs, ...

Ratios of densities $p_1(x)/p_2(x)$ → classification, eg CNNs, RNNs, GNNs, ...

A Living Review of ML for HEP

Huge collection of references : https://iml-wg.github.io/HEPML-LivingReview/

Covered topics

- **ML reviews**: modern, historical, ...
- **Classification**: jet images, graphs, flavor tagging, …
- **Regression**: calibration, parameter estimation, matrix element, ...
- **Generative** models/density estimation: GAN, normalizing flows, …
- **Anomaly** detection: BSM searches, hardware faults, real time detection...
- Simulation-based **Inference**: parameter estimation, unfolding, ...
- **Uncertainty** Quantification: interpretation, mitigation, estimation, ...
- ...

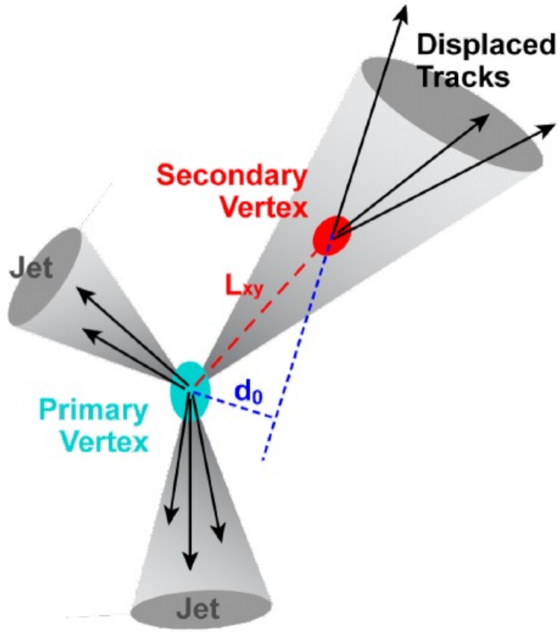
Many more “Proof-of-concept” than applications “in production”

examples

Flavor tagging

BSM searches

Flavor-tagging

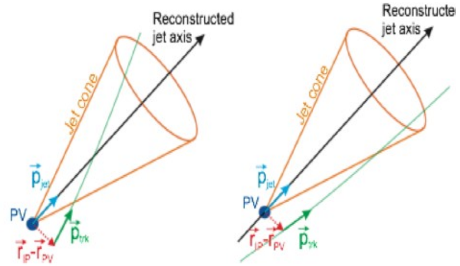


Goal: Discriminate b-jets from c-jets, light-jets, tau

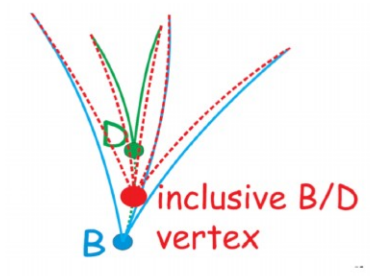
B-tagging algorithms utilize the long lifetime and displaced decays of b-hadrons to look for secondary vertices and displaced tracks

Flavor-tagging: Baseline Algorithms

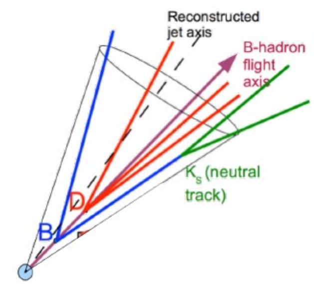
- **IP3D / IP2D:**
  - **Impact parameter algorithm**
  - Exploit (in)compatibility of track with PV



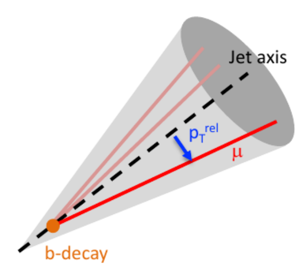
- **SV:**
  - **Inclusive Secondary vertexing**
  - Determination of single inclusive weak b-hadron decay vertex



- **JetFitter:**
  - **PV→B→D decay chain finding**
  - More detailed determination of decay vertex topology



- **SMT**: Soft-muon tagger BDT utilizes:
  - Muon kinematics and impact parameter
  - Track quality, to reject fakes and decays in flight

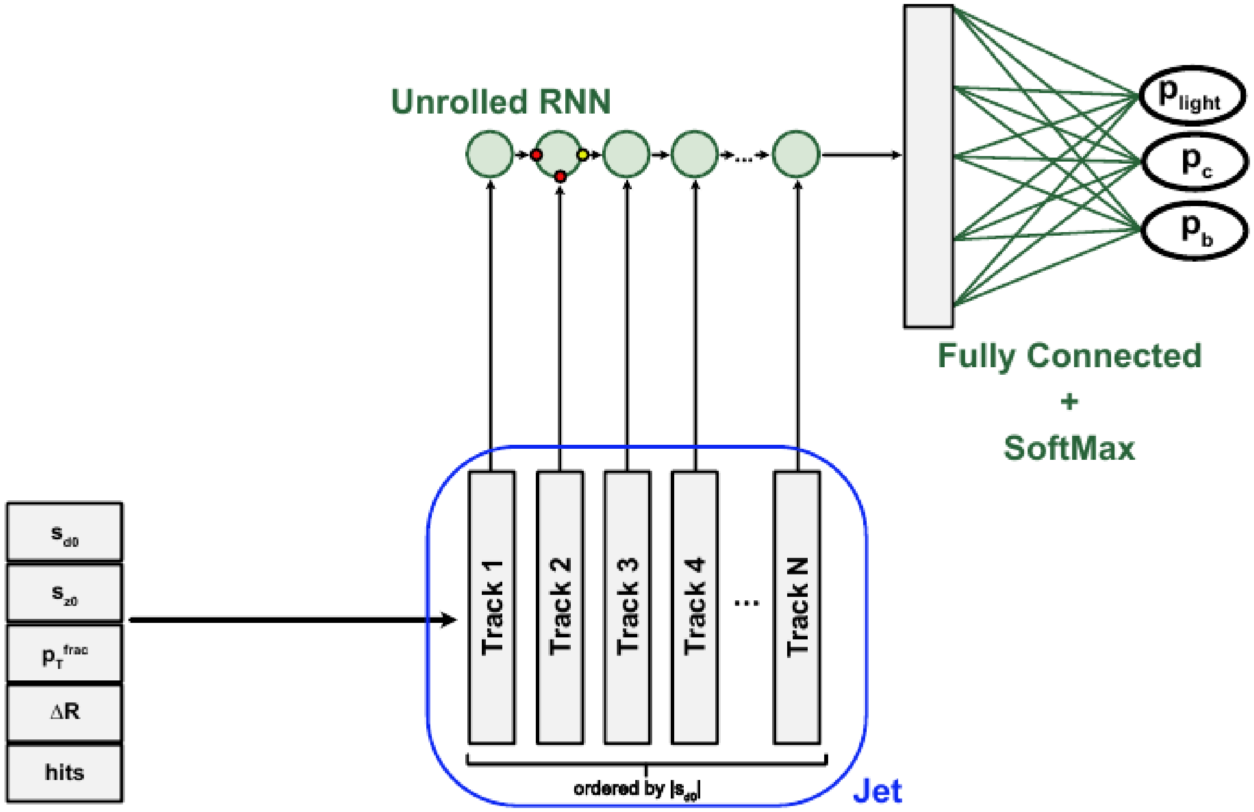


[slide M. Kagan]

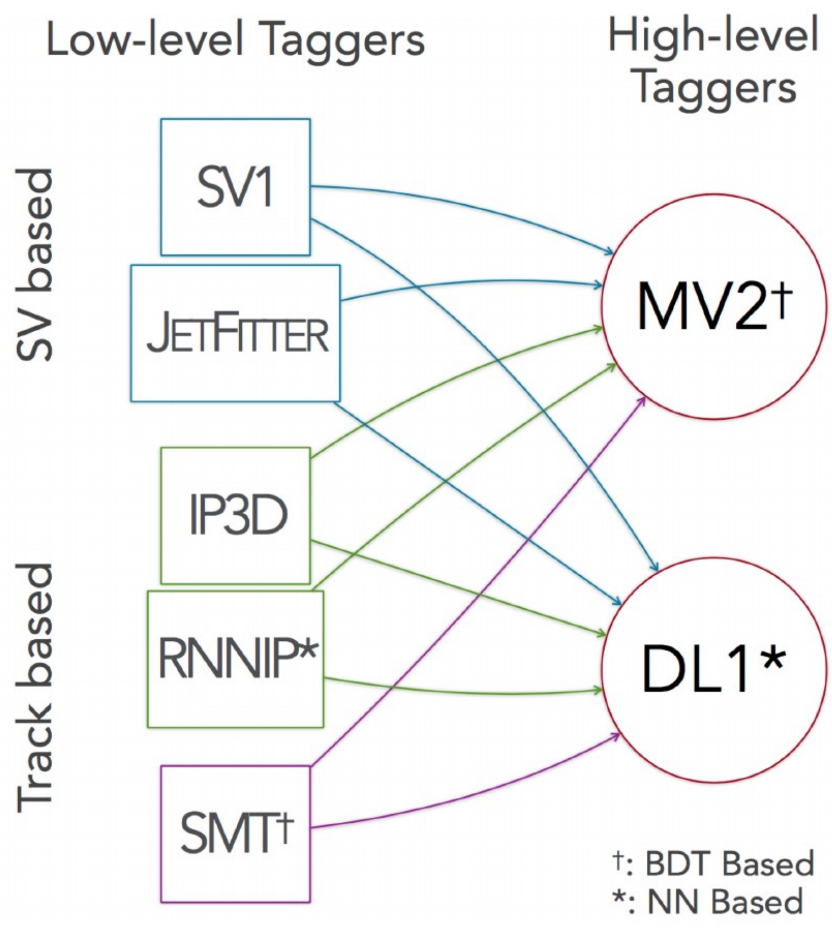
Flavor-tagging: Recurrent NN

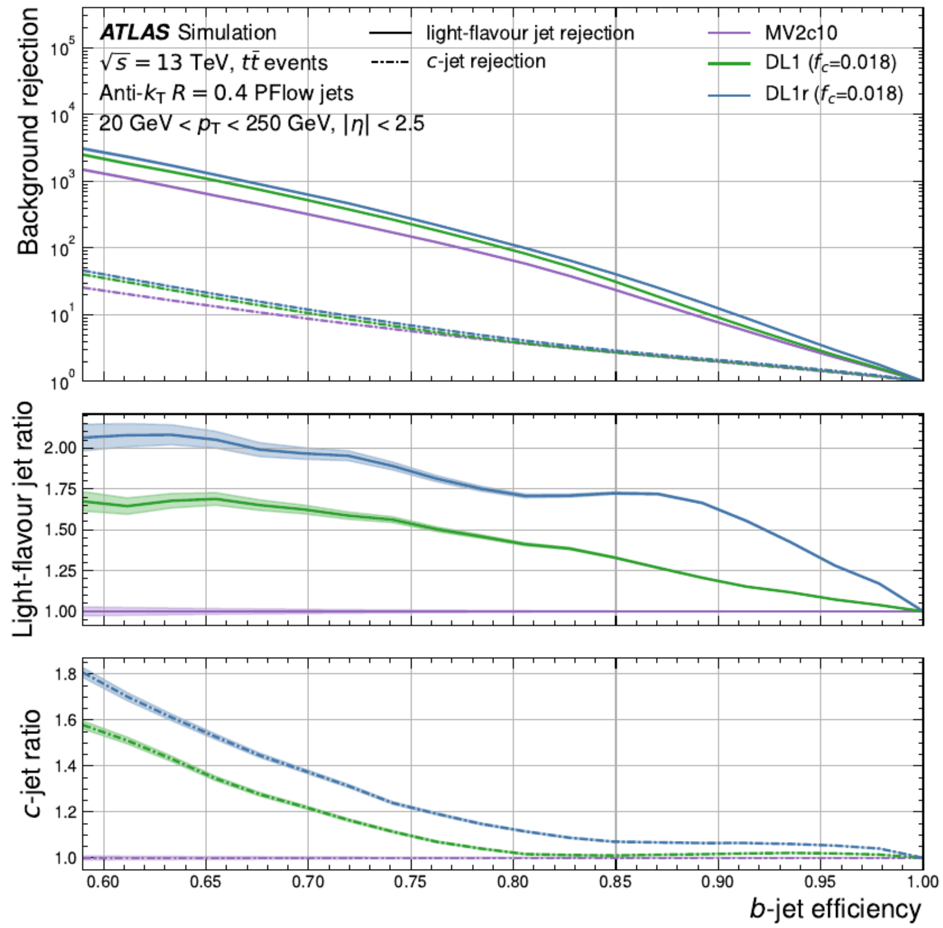
Recurent NN (sentence classification, NLP, …)

- Treat tracks as a sequence, ordered by impact parameter significance



High-Level Taggers

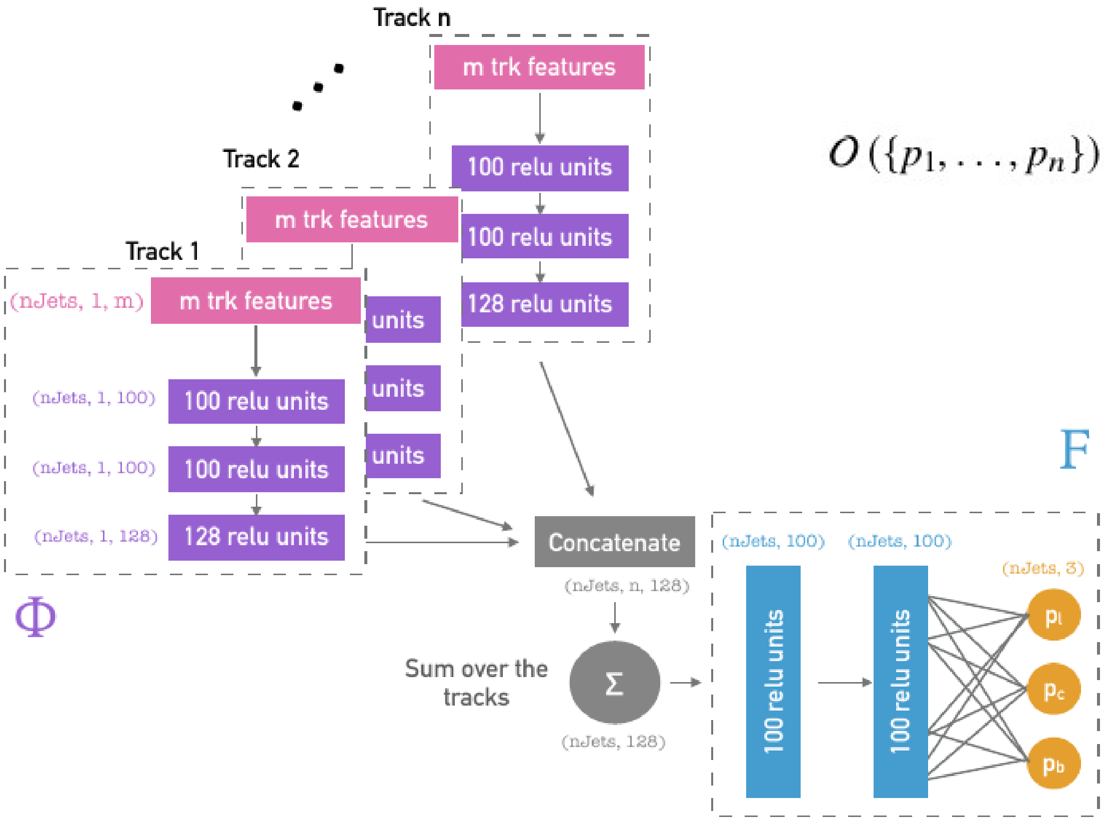




https://arxiv.org/abs/2211.16345

Improved algorithms: Deep Sets

The **Deep Sets** architecture treats each tracks as a set without any specific order = maintain benefit of RNN without requiring ordering



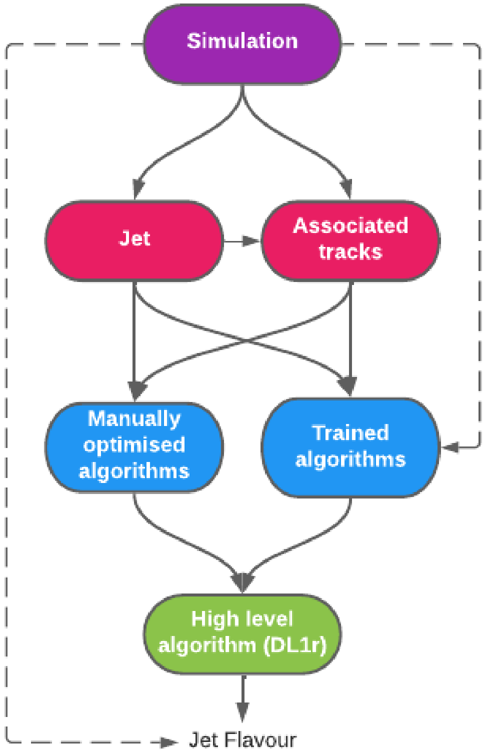
$$O(\{p_1, \ldots, p_n\}) = F\left(\sum_{i=1}^{n} \Phi(p_i)\right)$$

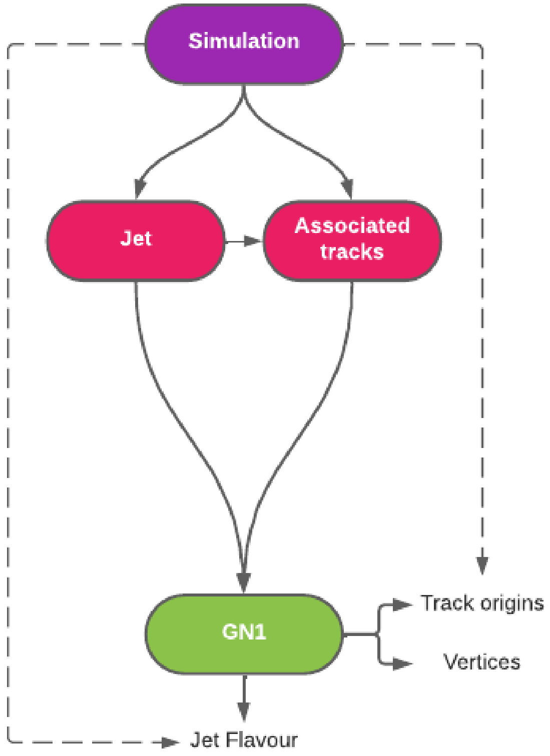
Faster training and improved performances (~ factor 2 bkgd rejection).

https://cds.cern.ch/record/2718948

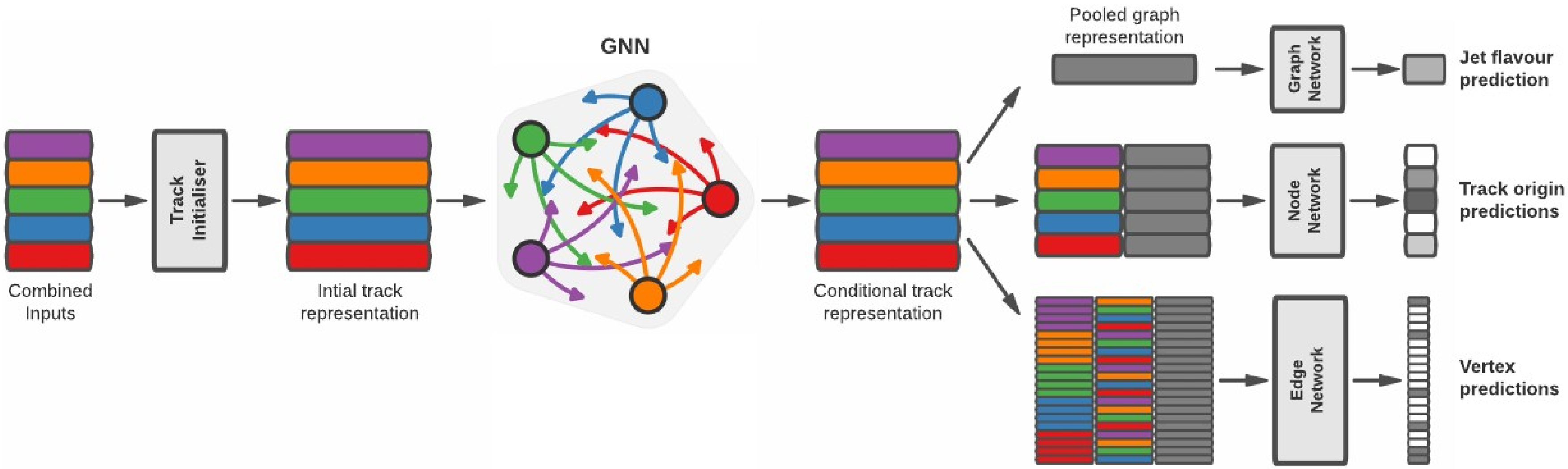
Improved algorithms: GNN

GNN* directly operates on tracks to perform b-tagging. It also performs vertexing and track classification, removing the need for low-level algorithms





* GNN ?? Read e.g. here https://arxiv.org/abs/2007.13681



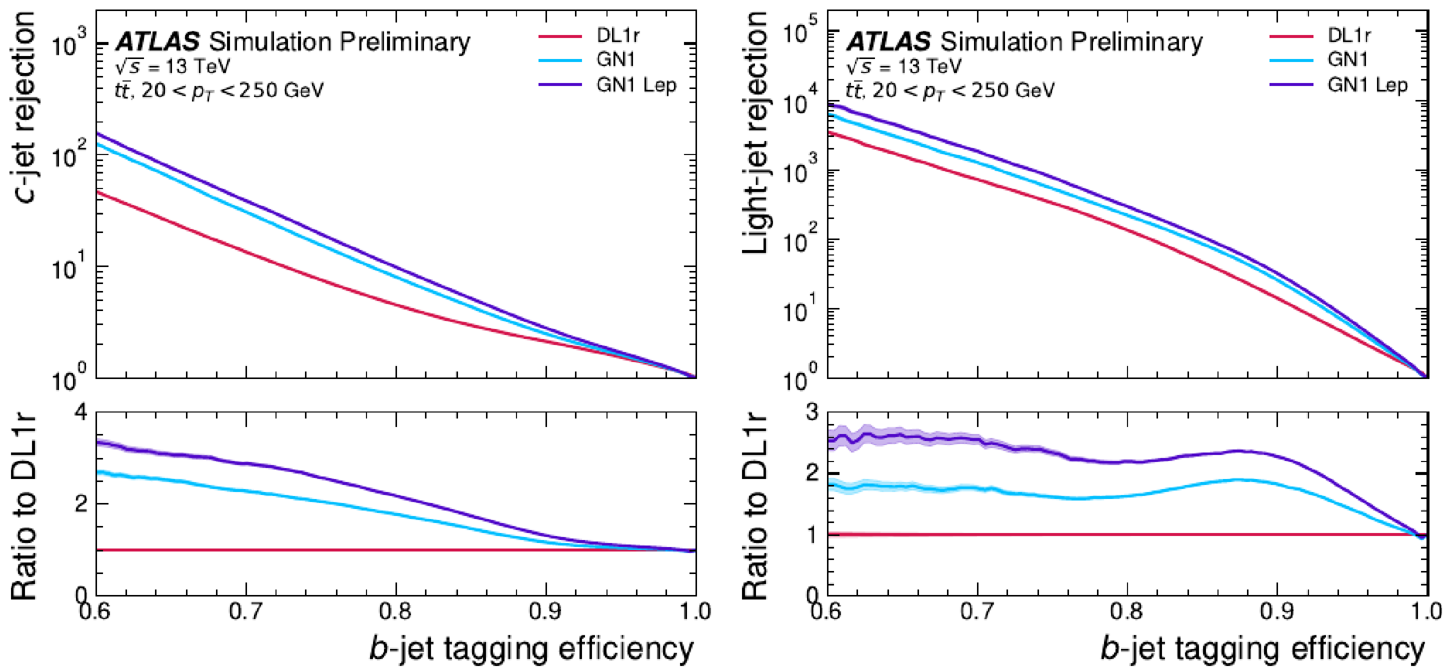
The model combines jet- and track-level information into a combined input, It is then fed into a per-track initialisation network, which outputs a latent representation of each track. These representations are used to populate the node features of a fully connected graph network.

After the graph network, the resulting node representations are used to predict the jet flavour, the track origins, and the track-pair vertex compatibility.

https://cds.cern.ch/record/2811135

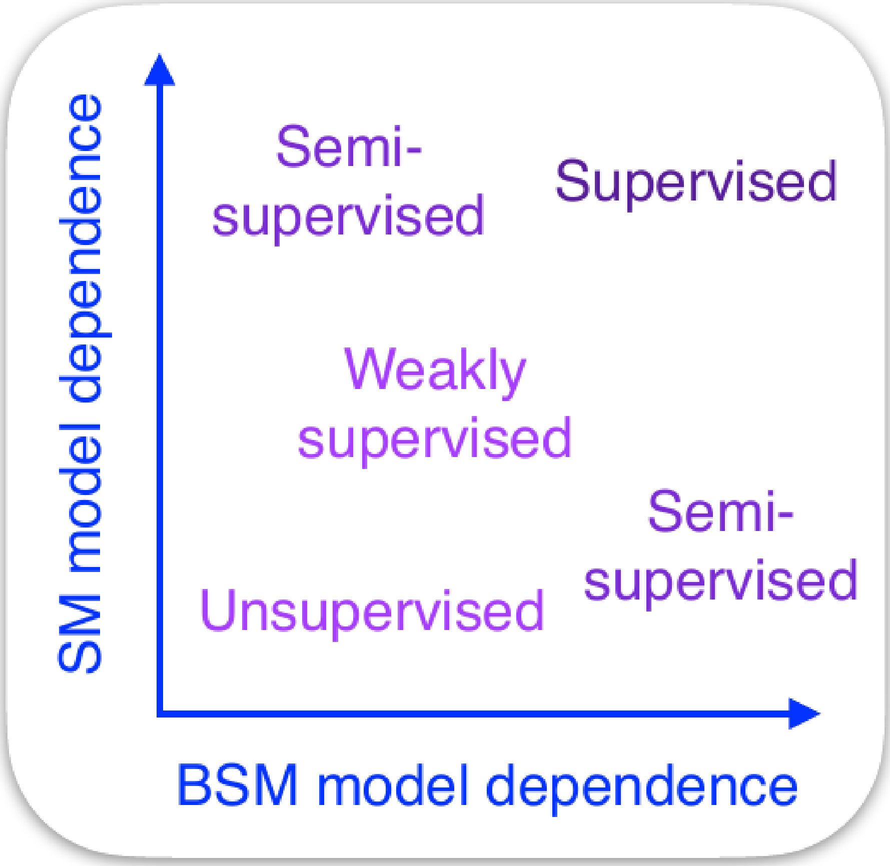
Improved algorithms: GNN

Performance improvement: factor 2-3 with respect to DL1+RNN



Story not over: other algorithmic improvements planned:
GNN with transformers/attention, etc.

BSM Searches



https://arxiv.org/abs/2112.03769

Anomaly detection as BSM Searches

Search for **unknown signal** (=anomaly) in data

Derive **background** model directly from **data**

Select region of phase-space potentially enriched in signal

Scan **multiple signatures** and variables

											$\mathrm{BSM} \to \mathrm{SM}_1 \times \mathrm{SM}_1$				$\mathrm{BSM} \to \mathrm{SM}_1 \times \mathrm{SM}_2$			$\mathrm{BSM} \to \mathrm{complex}$			
		e	μ	τ	q/g	b	t	γ	Z/W	H	q/g	γ/π^0's	b	$\cdots$	tZ/H	bH	$\cdots$	$\tau q q'$	$e q q'$	$\mu q q'$	$\cdots$
	e	[37, 38]	[39, 40]	[39]	∅	∅	∅	[41]	[42]	∅	∅	∅	∅		∅	∅	∅	∅	[43, 44]	∅	
	μ		[37, 38]	[39]	∅	∅	∅	[41]	[42]	∅	∅	∅	∅		∅	∅	∅	∅	∅	[43, 44]	
	τ			[45, 46]	∅	[47]	∅	∅	∅	∅	∅	∅	∅		∅	∅	∅	[48, 49]	∅	∅	
	q/g				[29, 30, 50, 51]	[52]	∅	[53, 54]	[55]	∅	∅	∅	∅		∅	∅	∅	∅	∅	∅	
	b					[29, 52, 56]	[57]	[54]	[58]	[59]	∅	∅	∅		[60]	∅	∅	∅	∅	∅	
	t						[61]	∅	[62]	[63]	∅	∅	∅		[64]	[60]	∅	∅	∅	∅	
	γ							[65, 66]	[67–69]	[68, 70]	∅	∅	∅		∅	∅	∅	∅	∅	∅	
	Z/W								[71]	[71]	∅	∅	∅		∅	∅	∅	∅	∅	∅	
	H									[72, 73]	[74]	∅	∅		∅	∅	∅	∅	∅	∅	
$\mathrm{BSM} \to \mathrm{SM}_1 \times \mathrm{SM}_1$	q/g										∅	∅	∅		∅	∅	∅	∅	∅	∅	
	γ/π^0's											[75]	∅		∅	∅	∅	∅	∅	∅	
	b												[76, 77]		∅	∅	∅	∅	∅	∅	
	$\vdots$																				
	$\vdots$																				

Searches for two-body resonances (arXiv:1907.06659)

ML for BSM Searches

Two open-dataset challenges fostered many novel ideas for anomaly detection

The LHC Olympics 2020

A Community Challenge for Anomaly Detection in High Energy Physics

Gregor Kasieczka (ed),[1] Benjamin Nachman (ed),[2,3] David Shih (ed),[4] Oz Amram,[5] Anders Andreassen,[6] Kees Benkendorfer,[2,7] Blaz Bortolato,[8] Gustaaf Brooijmans,[9] Florencia Canelli,[10] Jack H. Collins,[11] Biwei Dai,[12] Felipe F. De Freitas,[13] Barry M. Dillon,[8,14] Ioan-Mihail Dinu,[5] Zhongtian Dong,[15] Julien Donini,[16] Javier Duarte,[17] D. A. Faroughy[10] Julia Gonski,[9] Philip Harris,[18] Alan Kahn,[9] Jernej F. Kamenik,[8,19] Charanjit K. Khosa,[20,30] Patrick Komiske,[21] Luc Le Pottier,[2,22] Pablo Martín-Ramiro,[2,23] Andrej Matevc,[8,19] Eric Metodiev,[21] Vinicius Mikuni,[10] Inês Ochoa,[24] Sang Eon Park,[18] Maurizio Pierini,[25] Dylan Rankin,[18] Veronica Sanz,[20,26] Nilai Sarda,[27] Uroš Seljak,[2,3,12] Aleks Smolkovic,[8] George Stein,[2,12] Cristina Mantilla Suarez,[5] Manuel Szewc,[28] Jesse Thaler,[21] Steven Tsan,[17] Silviu-Marian Udrescu,[18] Louis Vaslin,[16] Jean-Roch Vlimant,[29] Daniel Williams,[9] Mikaeel Yunus[18]

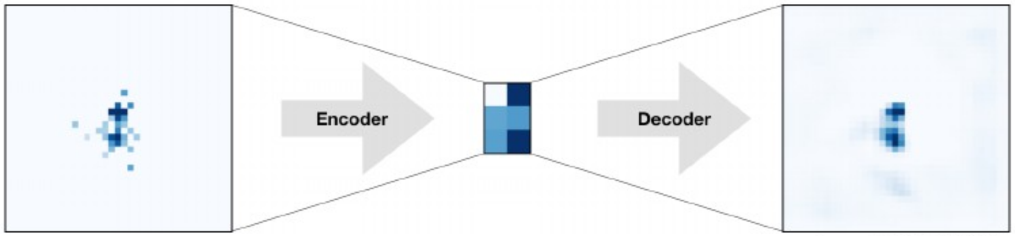
https://arxiv.org/abs/2101.08320

The Dark Machines Anomaly Score Challenge: Benchmark Data and Model Independent Event Classification for the Large Hadron Collider

T. Aarrestad[a] M. van Beekveld[b] M. Bona[c] A. Boveia[e] S. Caron[d] J. Davies[c] A. De Simone[f,g] C. Doglioni[h] J. M. Duarte[i] A. Farbin[j] H. Gupta[k] L. Hendriks[d] L. Heinrich[a] J. Howarth[l] P. Jawahar[m,a] A. Jueid[n] J. Lastow[h] A. Leinweber[o] J. Mamuzic[p] E. Merényi[q] A. Morandini[r] P. Moskvitina[d] C. Nellist[d] J. Ngadiuba[s,t] B. Ostdiek[u,v] M. Pierini[a] B. Ravina[l] R. Ruiz de Austri[p] S. Sekmen[w] M. Touranakou[x,a] M. Vaškevičiūte[l] R. Vilalta[y] J.-R. Vlimant[t] R. Verheyen[z] M. White[o] E. Wulff[h] E. Wallin[h] K.A. Wozniak[α,a] Z. Zhang[d]

https://arxiv.org/abs/2105.14027

ML for BSM Searches



from 1808.08992

Autoencoders

Fully unsupervised

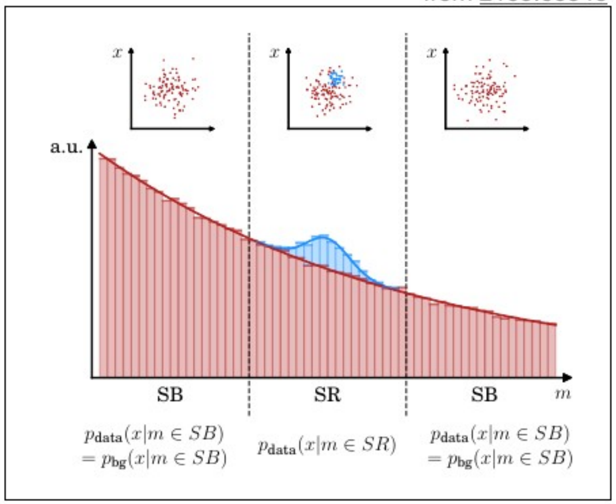
Sensitive to outliers (low $p(x)$)

Farina, Nakai & **DS** 1808.08992

Heimel et al 1808.08979

and many more!!

from 2109.00546



Enhanced bump hunts

Weakly supervised

Sensitive to overdensities (high $p_{data}(x)/p_{bg}(x)$)

CWoLa Hunting [Collins, Howe & Nachman 1805.02664, 1902.02634]

ANODE [Nachman & **DS** 2001.04990]

CATHODE [Hallin et al 2109.00546]

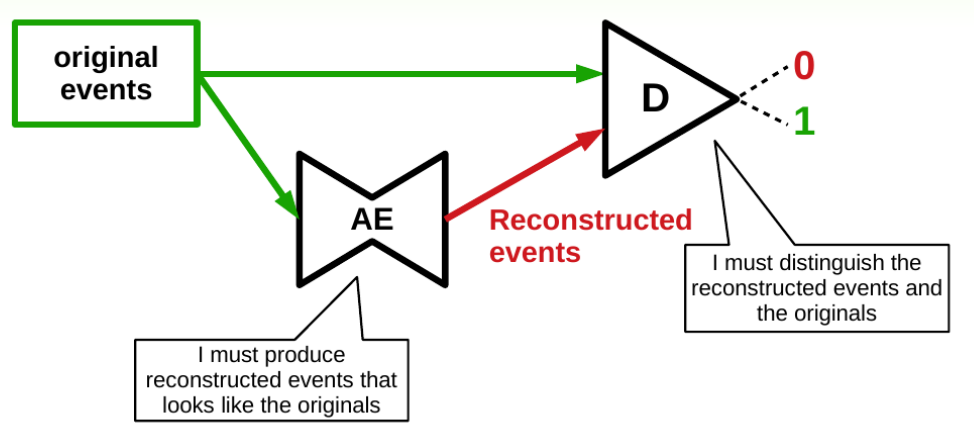
CURTAINS [Raine et al 2203.09470]

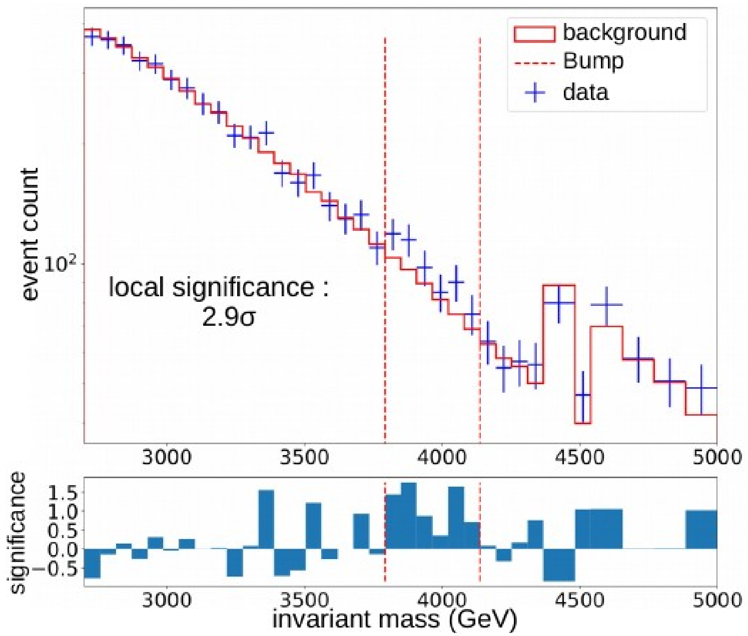
and more…

[slide D. Shih]

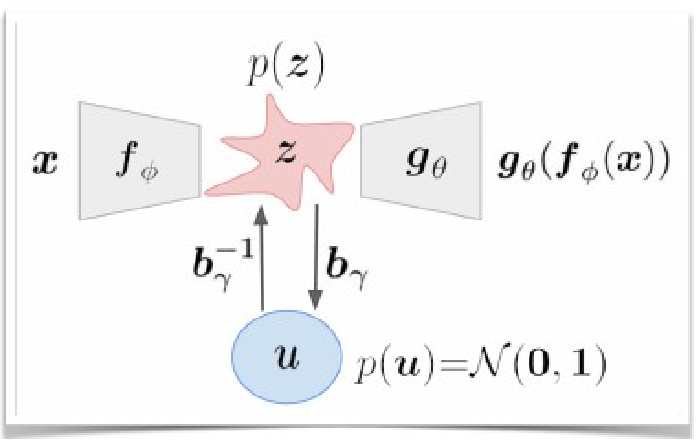
ML for BSM Searches

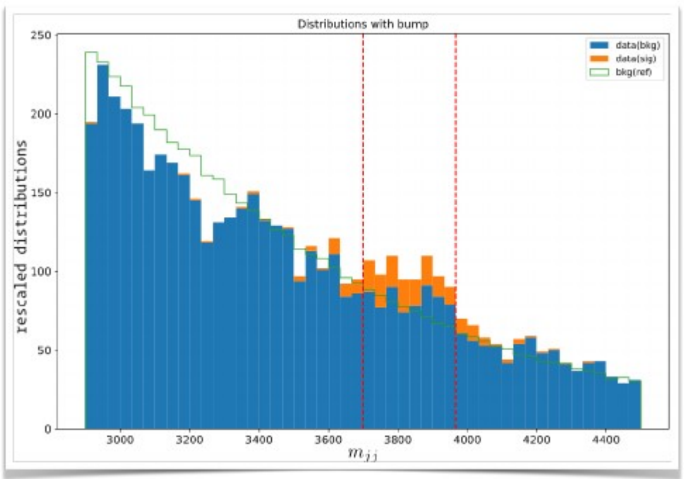
Adversarial auto-encoder (L. Valsin et al.)





Probabilistic auto-encoder (I. Dinu)



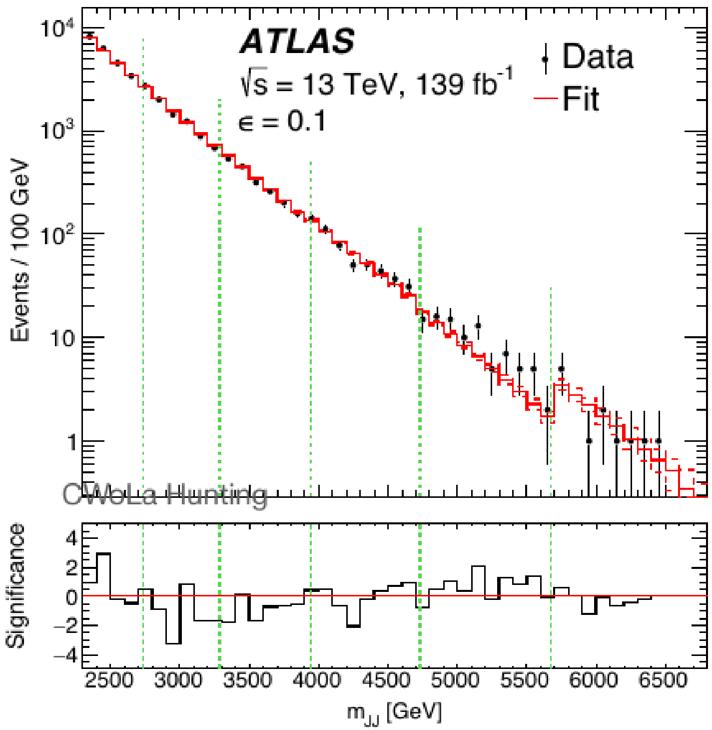


Work documented in community paper Rep. Prog. Phys. 84 124201

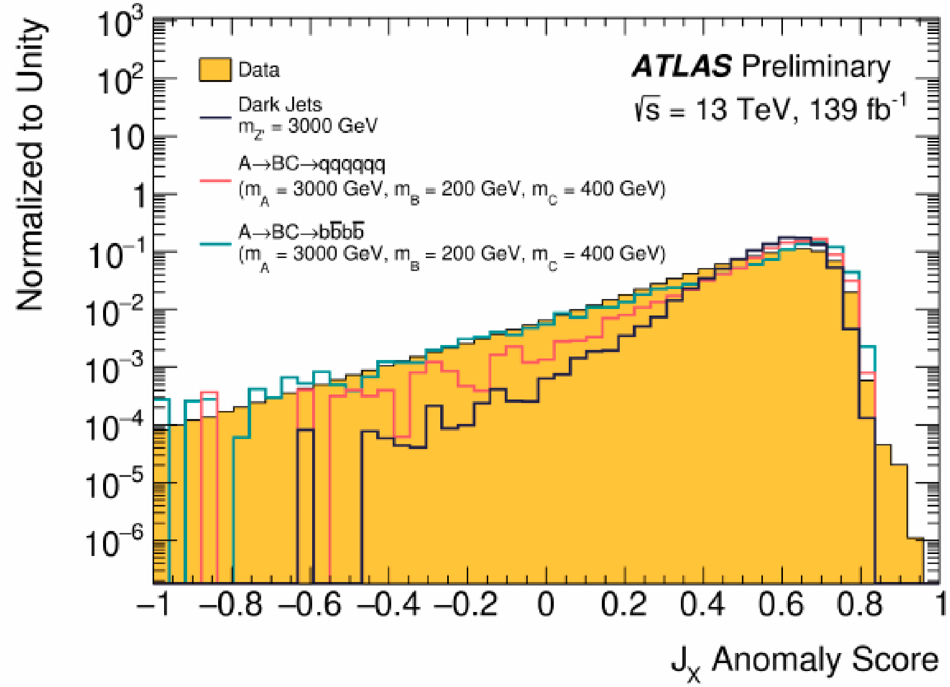
ML for BSM Searches

Proof-of-concept are becoming actual LHC searches:

CWoLa Hunting
ATLAS, PRL **125** 131801 (2020)

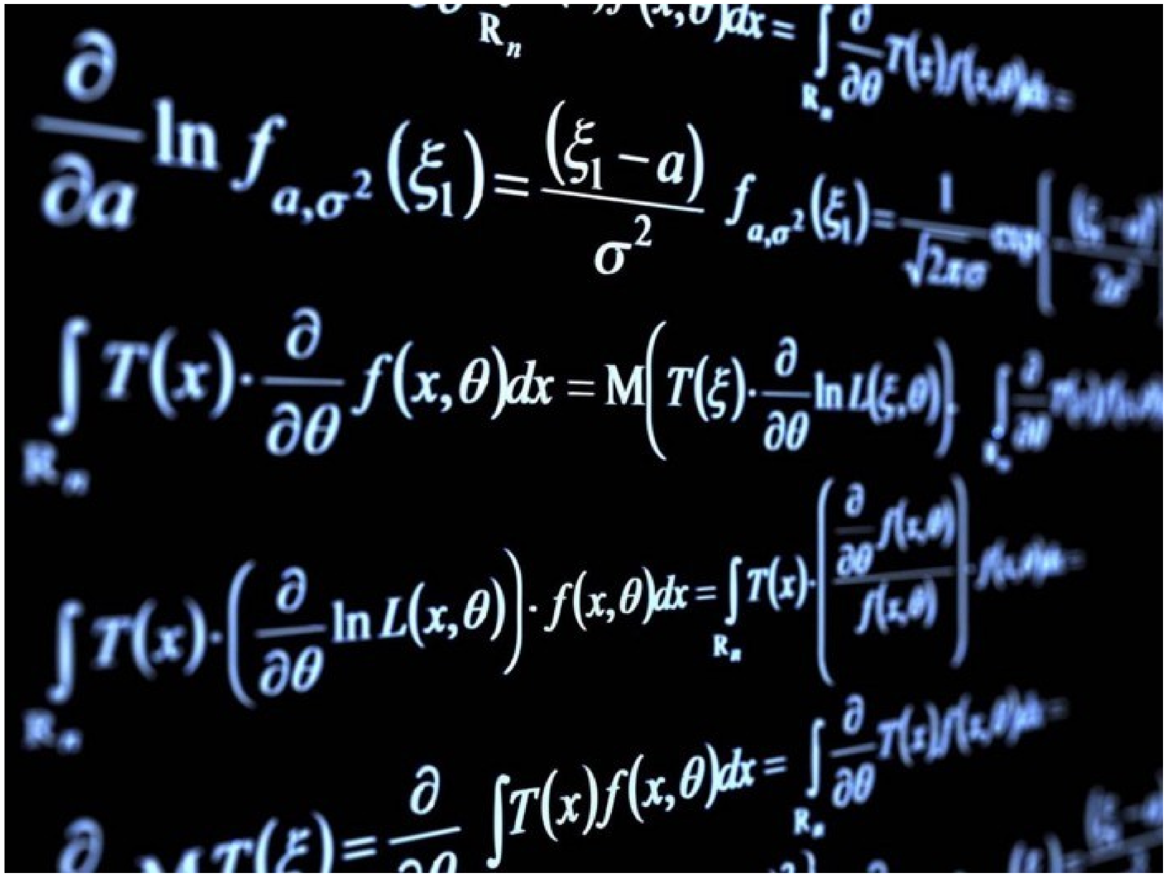


RNN VAE
ATLAS-CONF-2022-045



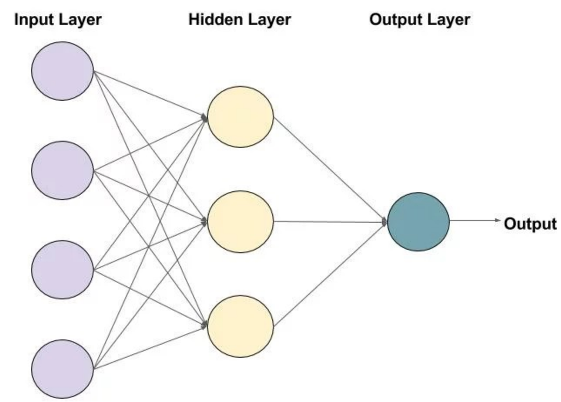
[slide D. Shih]

Automatic differentiation



Remember this slide ?

Data
(features **x**)



Objective
(target **t**)

Update of weights **W**

$$\mathbf{W} \rightarrow \mathbf{W} - \eta \sum_N \boxed{\frac{\partial \ell(\mathbf{y}, \mathbf{t})}{\partial \mathbf{W}}}$$

Loss functions

The **loss** measures how **close** the network **output** is to the **objective**

NN output

Target value

$$\text{Error} = \frac{1}{N}\sum_{i=1}^{N} \ell(\mathbf{y}(\mathbf{x_i},\mathbf{w}),\mathbf{t_i})$$

Training events

Examples:

$$\begin{cases} \ell(y_i,t_i) = (\mathbf{y}(\mathbf{x_i},\mathbf{w}) - \mathbf{t_i})^2 \\ \ell(y_i,t_i) = t_i \ln(y(\mathbf{x_i},\mathbf{w})) + (1-t_i)\ln(1-y(\mathbf{x_i},\mathbf{w})) \end{cases}$$

Mean square error

Cross entropy

The loss must be **minimized** → we need to compute its **gradient**

But **y** itself is a function of functions, with a lot of non-linearities → **complex !**

Example NN with 2 layers

Input data

$$\mathbf{x}$$

$$\downarrow$$

Hidden layer

$$\mathbf{s}^{(1)} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\downarrow$$

$$\mathbf{x}^{(1)} = f(\mathbf{s}^{(1)})$$

$$\downarrow$$

Output layer

$$\mathbf{s}^{(2)} = \mathbf{W}^{(2)}\mathbf{x}^{(1)} + \mathbf{b}^{(2)}$$

$$\downarrow$$

$$\mathbf{x}^{(2)} = f(\mathbf{s}^{(2)})$$

$$\downarrow$$

$$\mathbf{y}(\mathbf{x}) = \mathbf{x}^{(2)}$$

NN output

Forward pass

Backward pass

Use **chain rule** to compute **derivatives** of the loss $\ell(\mathbf{y}, \mathbf{t})$

$$\frac{\partial \ell}{\partial \mathbf{W}^{(2)}} = \frac{\partial \ell}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{s}^{(2)}} \frac{\partial \mathbf{s}^{(2)}}{\partial \mathbf{W}^{(2)}}$$

$$= \frac{\partial \ell}{\partial \mathbf{y}} \frac{\partial f(\mathbf{s}^{(2)})}{\partial \mathbf{s}^{(2)}} \mathbf{x}^{(1)}$$

$$\frac{\partial \ell}{\partial \mathbf{W}^{(1)}} = \frac{\partial \ell}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{s}^{(2)}} \frac{\partial \mathbf{s}^{(2)}}{\partial \mathbf{x}^{(1)}} \frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{s}^{(1)}} \frac{\partial \mathbf{s}^{(1)}}{\partial \mathbf{W}^{(1)}}$$

$$= \frac{\partial \ell}{\partial \mathbf{y}} \frac{\partial f(\mathbf{s}^{(2)})}{\partial \mathbf{s}^{(2)}} \frac{\partial \mathbf{s}^{(2)}}{\partial \mathbf{x}^{(1)}} \frac{\partial f(\mathbf{s}^{(1)})}{\partial \mathbf{s}^{(1)}} \mathbf{x}$$

Automatic differentiation

Automatic (algorithmic) differentiation (AD)

- **Numerical derivative evaluations** rather than derivative **expressions**
- Composition of **operations** for which **derivatives are known**
- **No need to rearrange** the code in a closed-form expression
- Accurate at **machine precision**

For each function a **computational graph** is constructed
→ **evaluation** of the function (forward pass)
→ calculation of **gradient** (backward pass)

Computational graph (example)

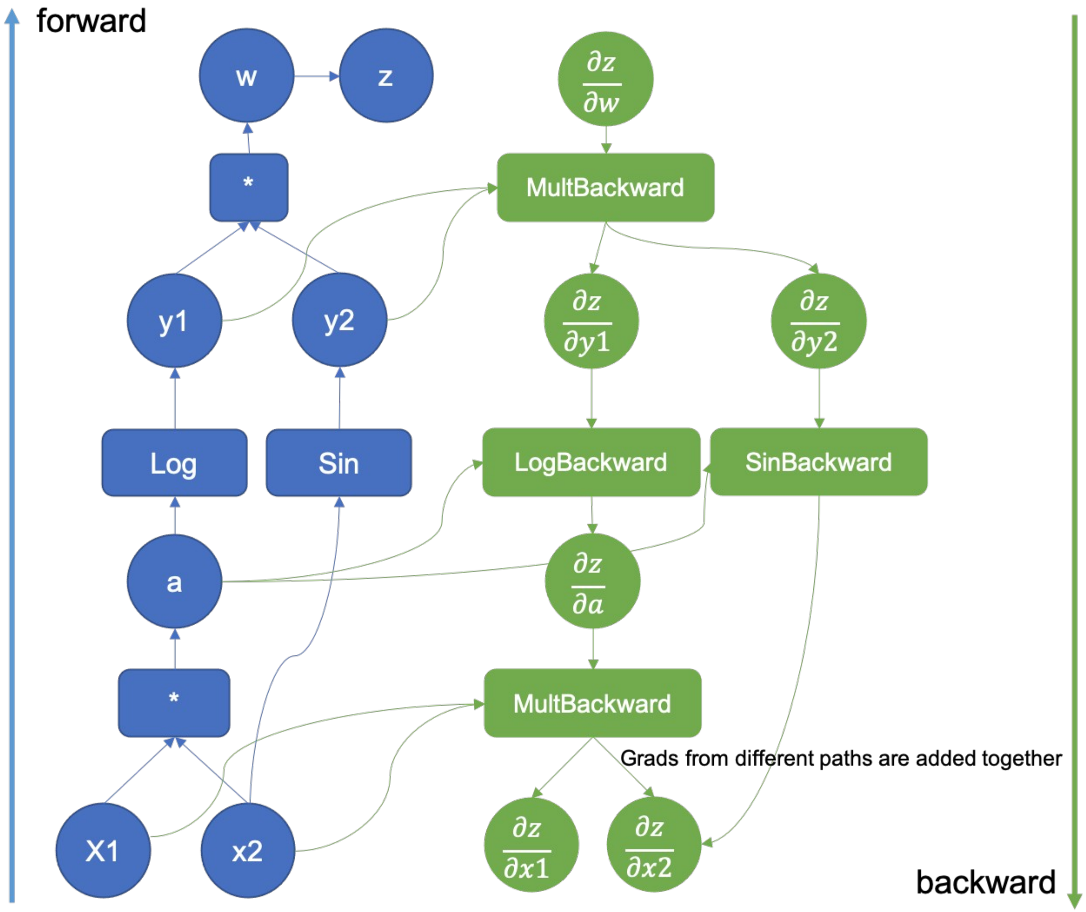
Forward pass

$$\vec{y} = \begin{pmatrix} \log(x_1 x_2) \\ \sin(x_2) \end{pmatrix}$$

$$z = y_1 y_2$$

Backward pass

$$\begin{pmatrix} \frac{\partial z}{\partial x_1} \\ \frac{\partial z}{\partial x_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} \end{pmatrix}^T \begin{pmatrix} \frac{\partial z}{\partial y_1} \\ \frac{\partial z}{\partial y_2} \end{pmatrix} = \begin{pmatrix} \frac{y_2}{x_1} \\ \frac{y_2}{x_2} + y_1 \cos(x_1) \end{pmatrix}$$

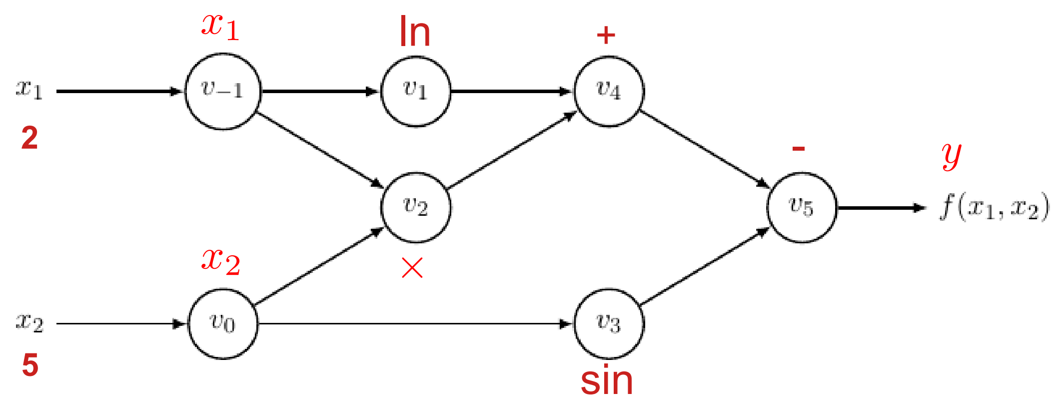


(see: https://pytorch.org/blog/overview-of-pytorch-autograd-engine/)

Example

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin(x_2)$$

Propagates derivatives **backwards** from output $\quad \bar{v}_i = \dfrac{\partial y}{\partial v_i}$



Forward Primal Trace		
v_{-1}	$= x_1$	$= 2$
v_0	$= x_2$	$= 5$
v_1	$= \ln v_{-1}$	$= \ln 2$
v_2	$= v_{-1} \times v_0$	$= 2 \times 5$
v_3	$= \sin v_0$	$= \sin 5$
v_4	$= v_1 + v_2$	$= 0.693 + 10$
v_5	$= v_4 - v_3$	$= 10.693 + 0.959$
y	$= v_5$	$= 11.652$

Reverse Adjoint (Derivative) Trace			
$\boldsymbol{\bar{x}_1}$	$\boldsymbol{= \bar{v}_{-1}}$		$\mathbf{= 5.5}$
$\boldsymbol{\bar{x}_2}$	$\boldsymbol{= \bar{v}_0}$		$\mathbf{= 1.716}$
$\bar{v}_{-1}$	$= \bar{v}_{-1} + \bar{v}_1 \frac{\partial v_1}{\partial v_{-1}}$	$= \bar{v}_{-1} + \bar{v}_1 / v_{-1}$	$= 5.5$
$\bar{v}_0$	$= \bar{v}_0 + \bar{v}_2 \frac{\partial v_2}{\partial v_0}$	$= \bar{v}_0 + \bar{v}_2 \times v_{-1}$	$= 1.716$
$\bar{v}_{-1}$	$= \bar{v}_2 \frac{\partial v_2}{\partial v_{-1}}$	$= \bar{v}_2 \times v_0$	$= 5$
$\bar{v}_0$	$= \bar{v}_3 \frac{\partial v_3}{\partial v_0}$	$= \bar{v}_3 \times \cos v_0$	$= -0.284$
$\bar{v}_2$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_2}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_1$	$= \bar{v}_4 \frac{\partial v_4}{\partial v_1}$	$= \bar{v}_4 \times 1$	$= 1$
$\bar{v}_3$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_3}$	$= \bar{v}_5 \times (-1)$	$= -1$
$\bar{v}_4$	$= \bar{v}_5 \frac{\partial v_5}{\partial v_4}$	$= \bar{v}_5 \times 1$	$= 1$
$\bar{v}_5$	$= \bar{y}$	$= 1$	

Reverse mode example, evaluated at $(x_1, x_2) = (2, 5)$. Both $\dfrac{\partial y}{\partial x_1}$ and $\dfrac{\partial y}{\partial x_2}$ are computed on the same reverse pass starting from the output

$$\bar{v}_5 = \bar{y} = \frac{\partial y}{\partial y} = 1$$

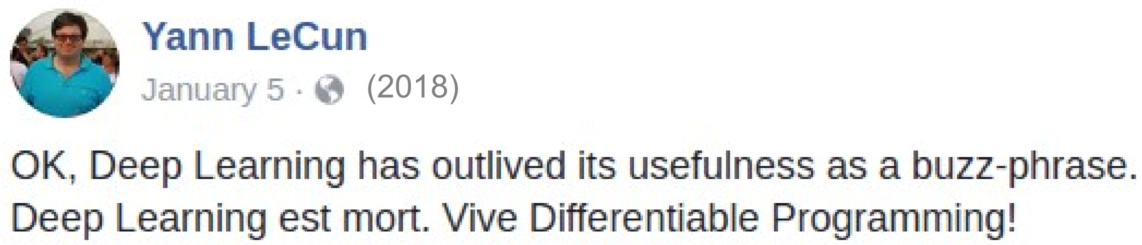
[1502.05767]

Can we push this idea further ?

Automatic differentiation is used to optimize complex networks

Could we use it to optimize **complex problems** ?

Yes: differentiable programming



Gradient-based **optimization** methods

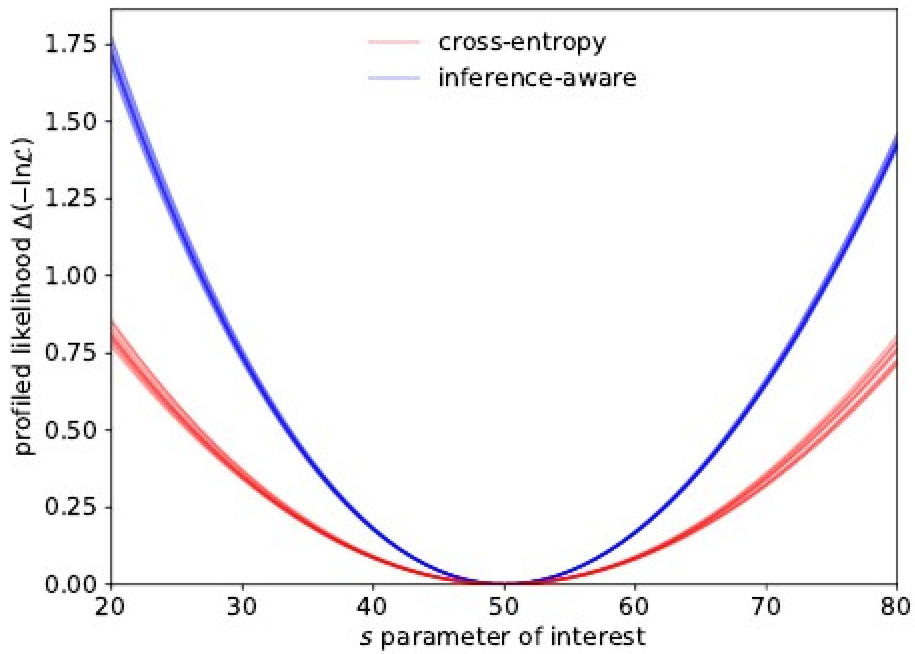
➔ Code composed of **differentiable and parameterized building blocks**

➔ **Software** optimized via **automatic differentiation**

ML for Precision Measurements

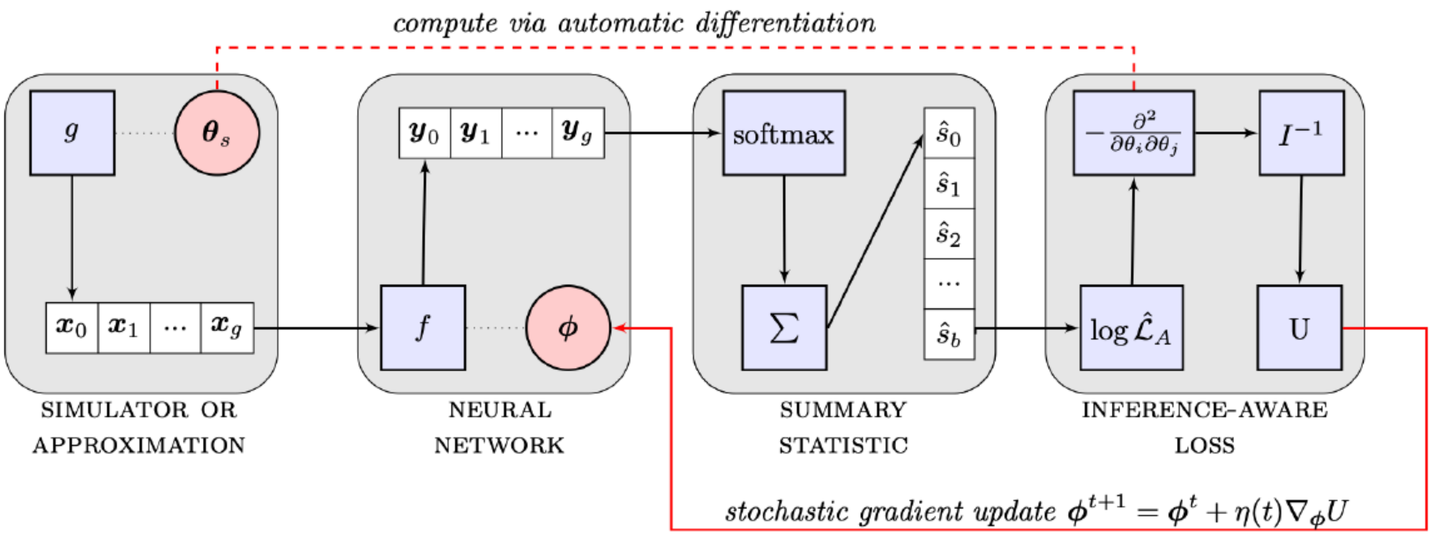
Inference Aware **N**eural **O**ptimization [1806.04743, de Castro, Dorigo]

- **Include nuisance parameters** in the loss function and directly minimize **precision of parameters of interest** (e.g. signal strenght measurement)



Profiled likelihood around the expectation value for the parameter of interest for **inference-aware** models and **cross-entropy** loss based models.

INFERNO Algorithm

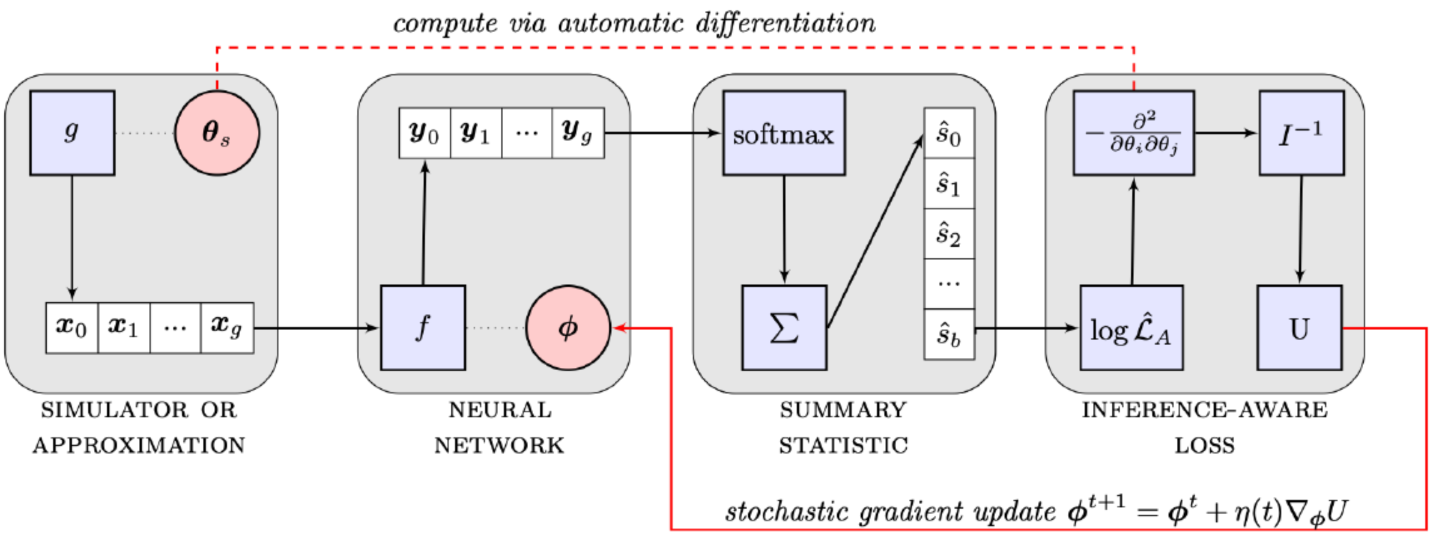


NN output summary statistics from input data

Loss function: **uncertainty** on parameter of interest (U)

Obtained by computing full **hessian** of the **Likelihood** with respect to all **nuisance** parameters

INFERNO Algorithm



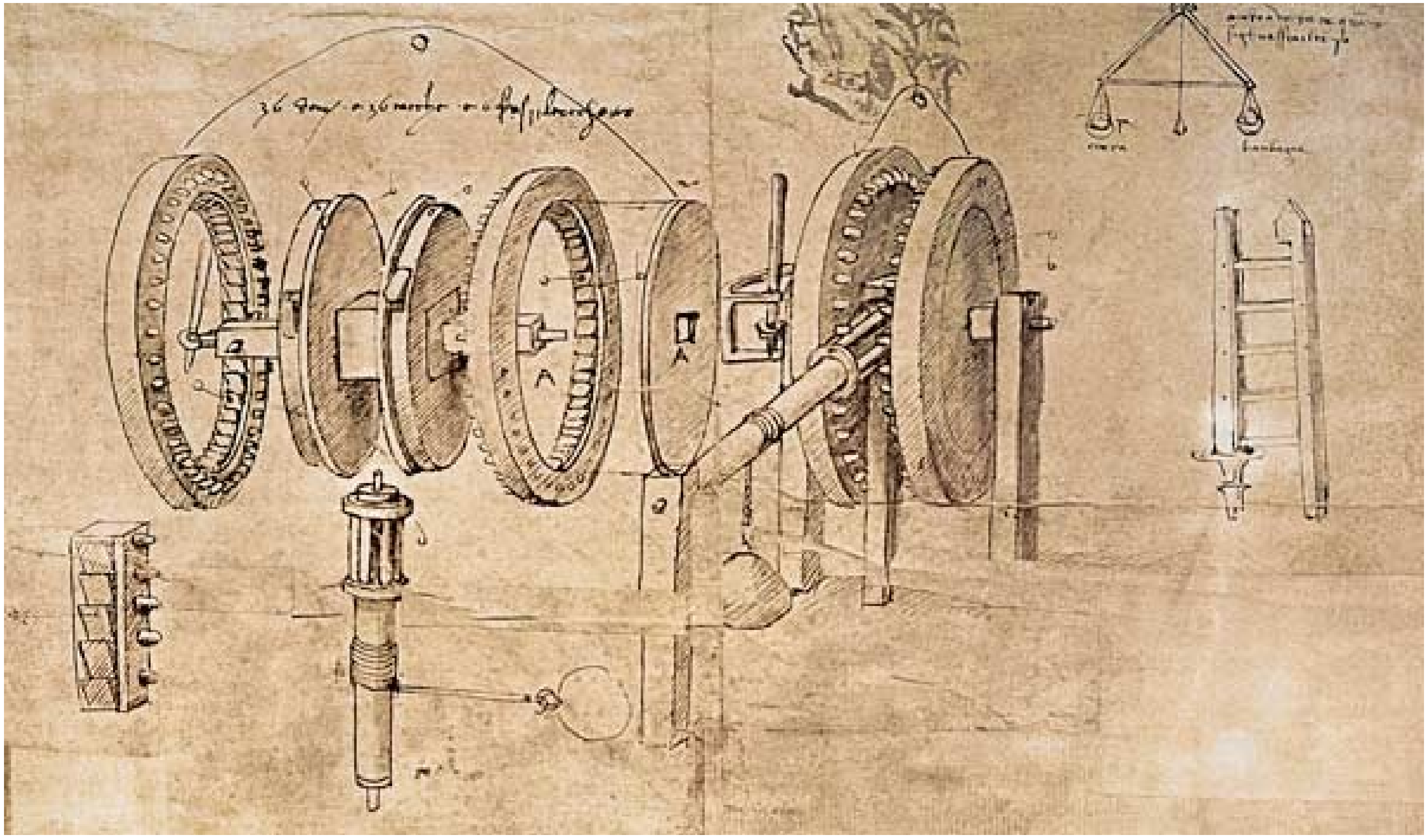
Algorithm 1 Inference-Aware Neural Optimisation.

Input 1: differentiable simulator or variational approximation $g(\boldsymbol{\theta})$.
Input 2: initial parameter values $\boldsymbol{\theta}_s$.
Input 3: parameter of interest $\omega_0 = \theta_k$.
Output: learned summary statistic $\boldsymbol{s}(D; \boldsymbol{\phi})$.

1: **for** $i = 1$ to N **do**
2: Sample a representative mini-batch G_s from $g(\boldsymbol{\theta}_s)$.
3: Compute differentiable summary statistic $\hat{\boldsymbol{s}}(G_s; \boldsymbol{\phi})$.
4: Construct Asimov likelihood $\mathcal{L}_A(\boldsymbol{\theta}, \boldsymbol{\phi})$.
5: Get information matrix inverse $I(\boldsymbol{\theta})^{-1} = \boldsymbol{H}_{\boldsymbol{\theta}}^{-1}(\log \mathcal{L}_A(\boldsymbol{\theta}, \boldsymbol{\phi}))$.
6: Obtain loss $U = I_{kk}^{-1}(\boldsymbol{\theta}_s)$.
7: Update network parameters $\boldsymbol{\phi} \rightarrow \mathrm{SGD}(\nabla_{\boldsymbol{\phi}} U)$.
8: **end for**

For a detailed explanation of the algorithm see G. Strong blog post

Can automatic differentiation be applied to detector optimization ?



Optimization of Detector Design

Design of detectors traditionally **relies** on individual **optimization of subdetector**

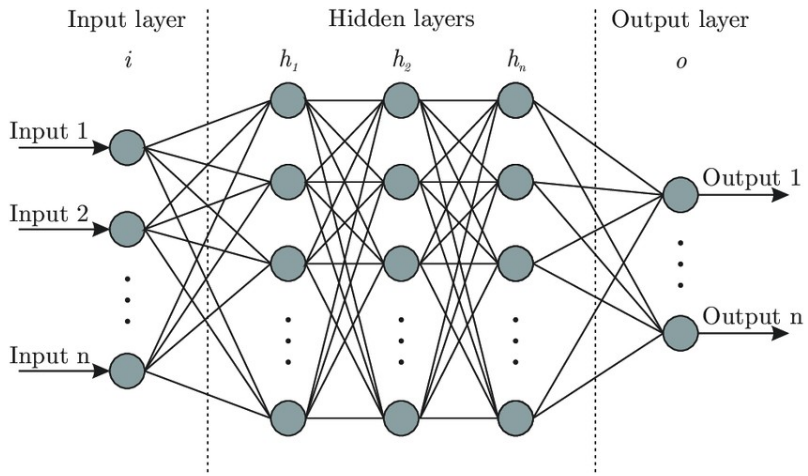
- **Track first, destroy later**
  - First detect ionization tracks in tracker, then measure energy deposits from destructive interaction with thick calorimeters
- **Per-subdetector optimization**
  - subdetector-specific figures of merit (e.g. momentum resolution)
- **Impact on physics goals** typically considered in a second step

Optimization of a **joint problem** ≠ different from **individual optimization**

$$argmax_{x,y}(\mathcal{L}(x,y)) \neq \left[argmax_x\left(\int \mathcal{L}(x,y)dy\right), argmax_y\left(\int \mathcal{L}(x,y)dx\right)\right]$$

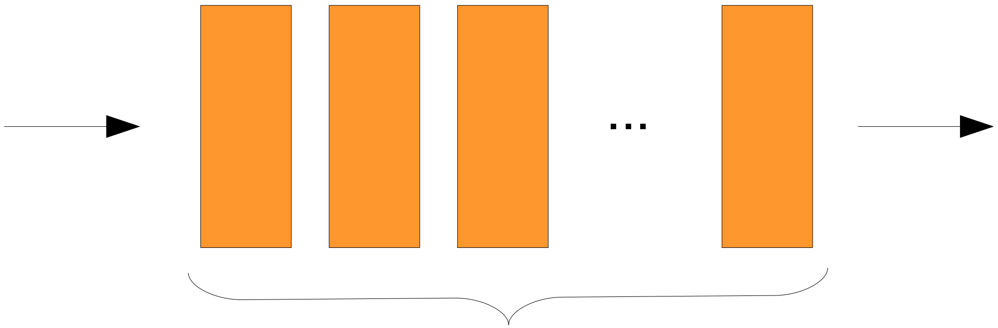
ML for Detector Optimization

Data



NN output

Data
(particles)



Parameters of interest
(physics)

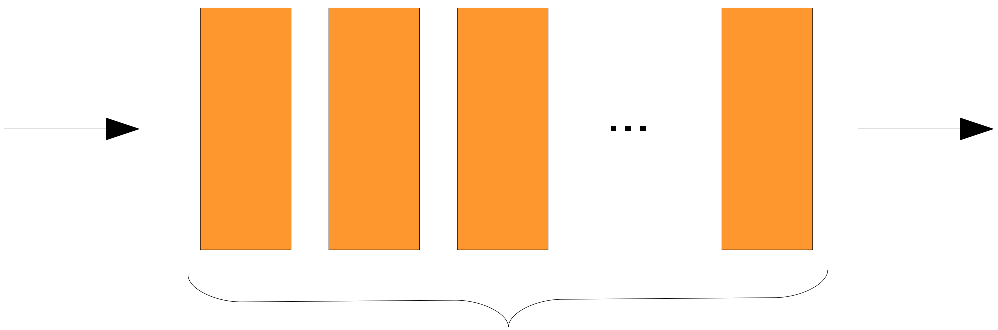
Differentiable simulated
detector **modules**

Minimization of objective function through automatic differentiation

ML for Detector Optimization

What if simulator is **not differentiable** ? Try differentiable **surrogate models**

Data (particles)



Parameters of interest

Differentiable simulated detector **modules**

GEANT4 → 10^{10} events **SLOW but ACCURATE**

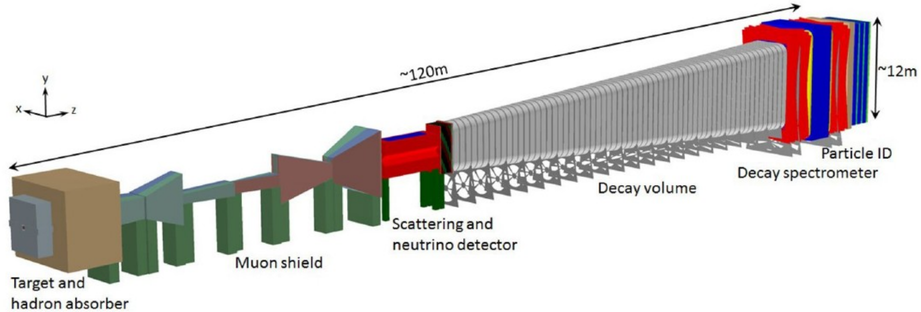
GEANT4 → 10^{5} events → Surrogate model → 10^{10} events

(GAN, VAE, Normalizing Flow, …)
Learn underlying distribution of GEANT4 events

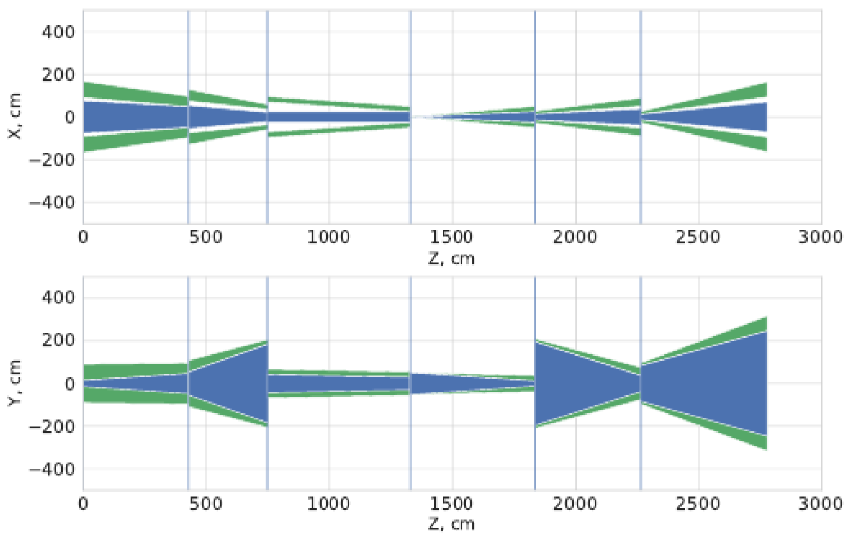
Differentiable

Muon shielding in SHIP

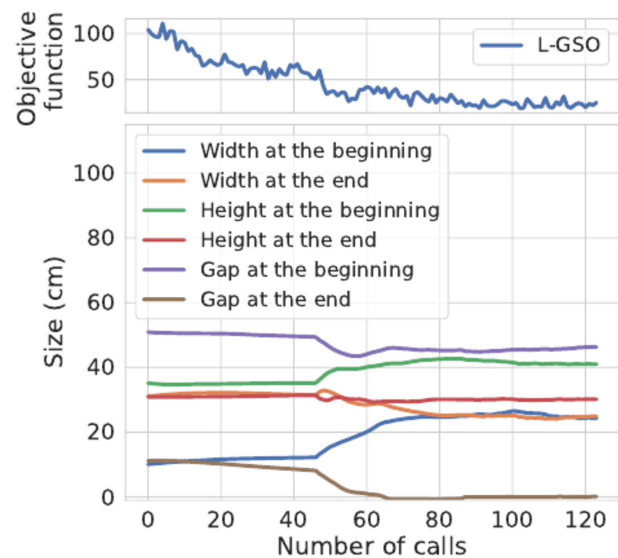
Minimize muon background fluxes in the **SHIP** steel **magnet** by varying its **geometry**



Local generative surrogate solution is **shorter** and has **lower mass** than other proposal, hence improving **efficacity** of the experiment and reducing its **cost**



Geometry of the magnet
42 parameters to optimize



[2002.04632]

Evolution of 6 parameters
during **optimization**

Machine-Learning Optimized Design of Experiments

MODE Collaboration

https://mode-collaboration.github.io

A. G. Baydin[5], A. Belias[10], A. Boldyrev[4], K. Cranmer[8], P. de Castro Manzano[1], T. Dorigo[1,14], C. Delaere[2], D. Derkach[4], J. Donini[3], F. Fanzago[1], A. Giammanco[2], C. Glaser[11], L. Heinrich[12], J. Kieseler[7], C. Krause[13], M. Lagrange[2], M. Lamparth[12], G. Louppe[6], L. Layer[1], F. Nardi[3,14], P. Martinez Ruiz del Arbol[9], F. Ratnikov[4], P. Stowell[15], G. Strong[1], M. Tosi[1,14], A. Ustyuzhanin[4], P. Vischia[2], H. Yarar[1] ,H. Zaraket[16]

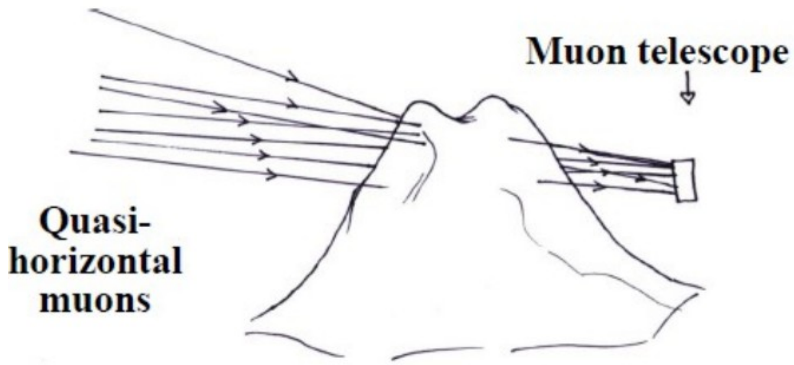
1 INFN, Italy
2 Université Catholique de Louvain, Belgium
3 Université Clermont Auvergne, France
4 Laboratory for big data analysis of the HSE, Russia
5 University of Oxford
6 Université de Liege
7 CERN
8 New York University
9 IFCA, Spain
10 GSI, Germany
11 Uppsala Universitet, Sweden
12 TU Munchen, Germany
13 Rutgers University, US
14 Università di Padova, Italy
15 Durham University, UK
16 Lebanese University, Lebanon

Sponsored by

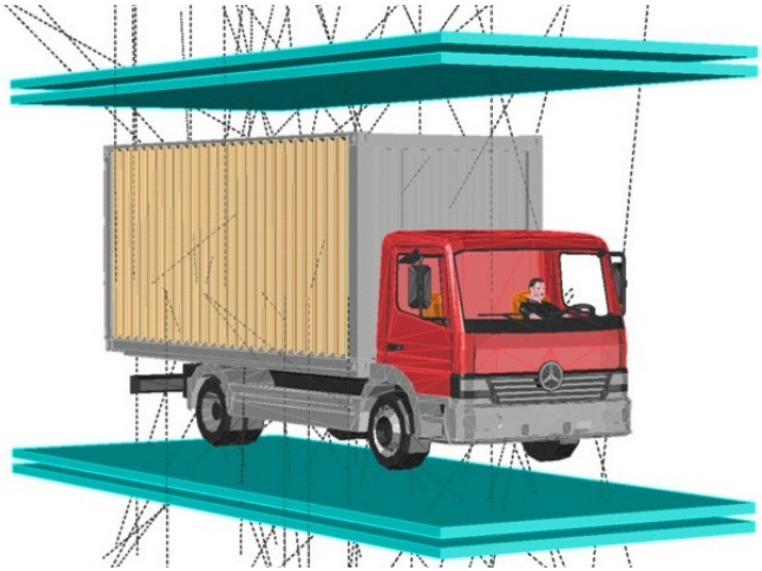
Differentiable programming for muography

Tomography: exploit **atmospheric muon** flux to **map** the interior of **objects**

Muon absorption



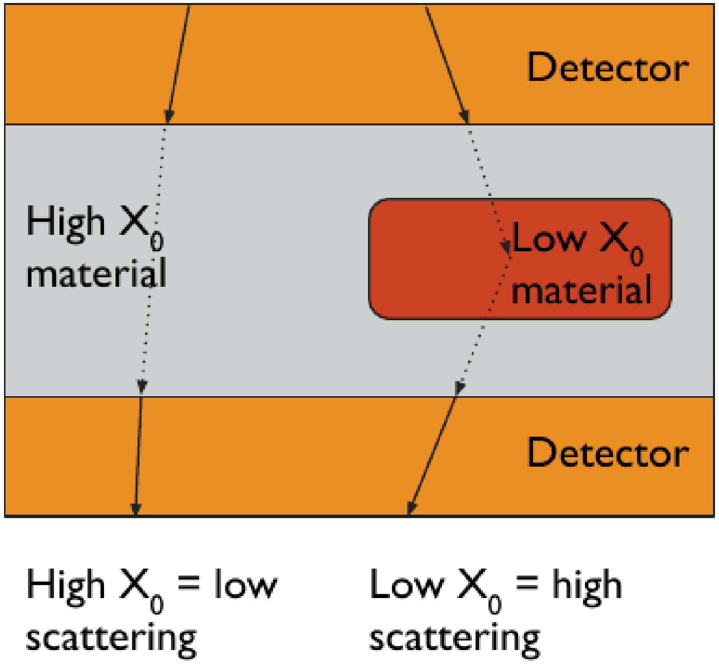
Muon scattering



[images : A. Giammanco]

Muon tomography

Volume with unknown **composition** sandwiched between **detectors**



Infer $\mathbf{X_0}$ (radiation length) of volume by measuring **muon scattering**

How should detectors be positionned for best performances ?

- i.e Muon detection **accuracy**, **resolution** on X_{0}, ...
- But also: **cost**, **size**, ...

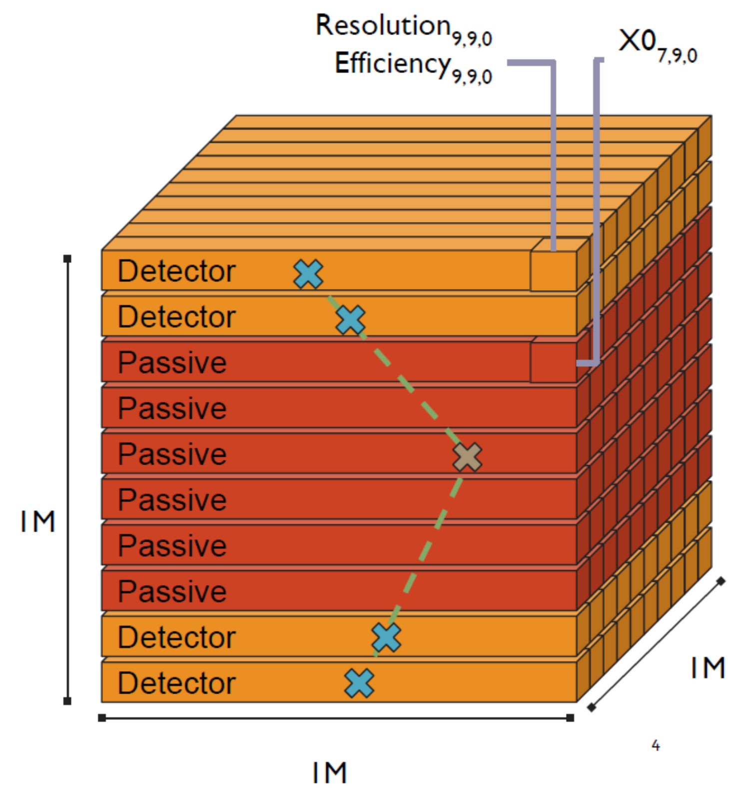
[see G. Strong talk]

TomOpt: Tomography Optimization

Muon scan of volume of unknown density

Volume of unknown density (*e.g.* one including a high-Z block of 0.5x0.1x0.1 m^3 somewhere inside a 0.6x1x1m^3 of low-Z material)

The system «learns» how to compromise cost and precision to optimize the inference on the Z map, and where detector elements are less useful



Python package for differential optimisation of muon-tomography detectors

G.Strong, T.Dorigo, F.Fanzago, A.Giammanco, M.Lagrange, M.Lamparth, F.Nardi, and P.Vischia

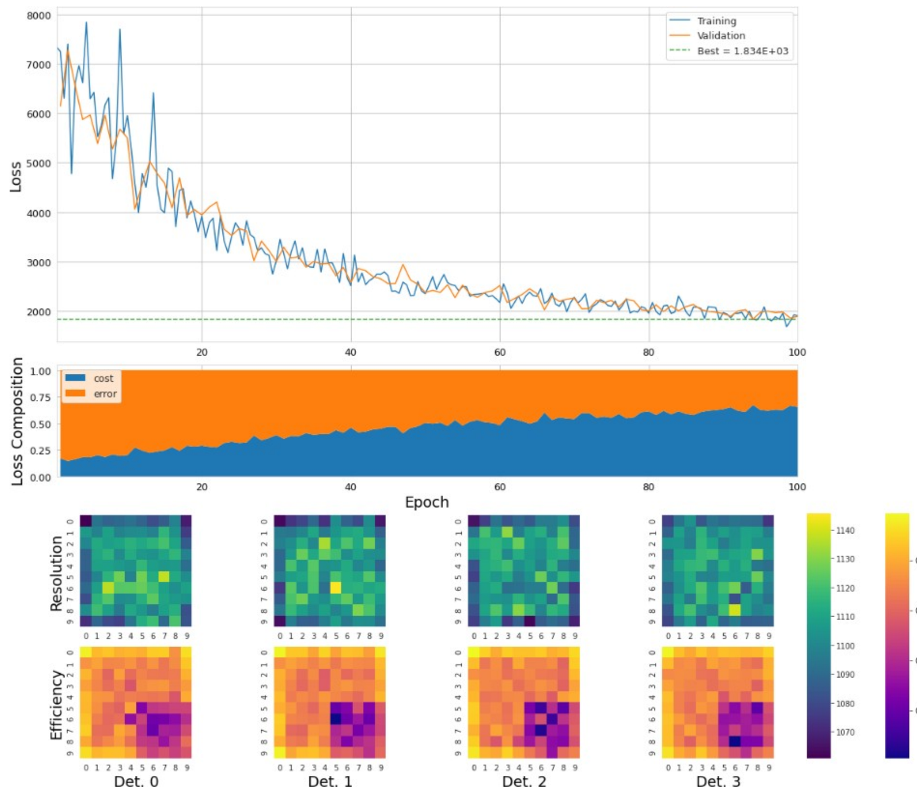
TomOpt: Tomography Optimization

Result of a run of 100 epochs training, followed by a prediction with 100k muons

Shows training of a differentiable model of a schematic muon tomography apparatus.

The **loss** is a combination of detector cost (itself a function of sensors efficiency and resolution) and RMSE on rad length estimate

Still a long way to go..., but an important milestone



Above, top to bottom: loss, loss composition, resolution map, and efficiency map of detection elements after minimization.

[code and results Giles Strong]

Conclusion

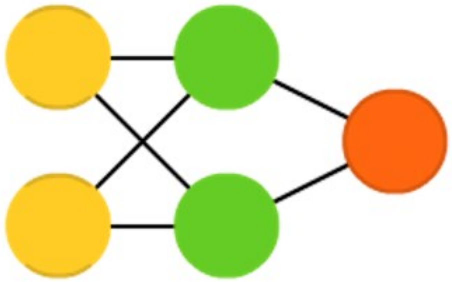
Modern ML is not doing what we did before, but more quickly. It is really doing physics that could not be **possible** otherwise.

Huge **potential** for new physics searches, triggering, fast simulation, instrumentation, theory, etc.

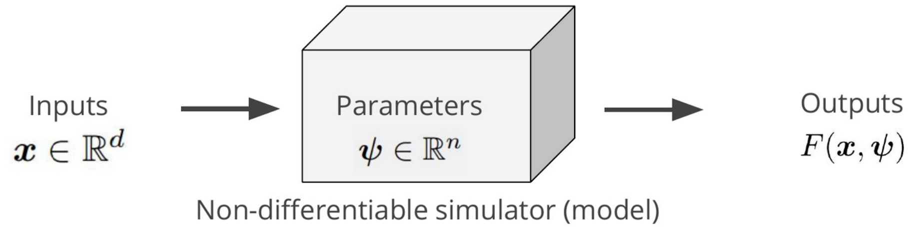
Very **active** field in HEP in recent years with lots of ideas and developments.

Exciting opportunities for (young) physicists !

Backup material



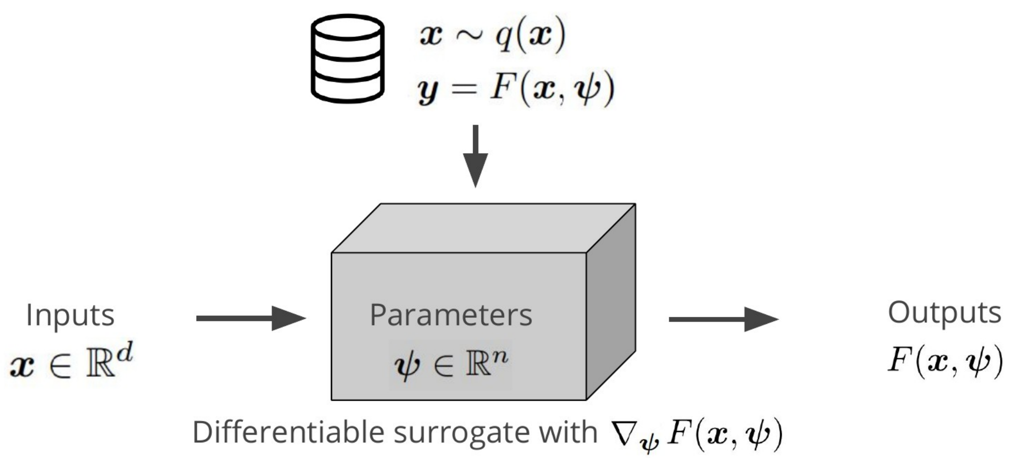
Surrogates for differentiability



- Run simulator many times
- Generate a (large) dataset of input - output pairs capturing simulator's behavior

$$\boldsymbol{x} \sim q(\boldsymbol{x})$$
$$\boldsymbol{y} = F(\boldsymbol{x}, \boldsymbol{\psi})$$

- Use the dataset to learn a differentiable approximation of the simulator (e.g., a deep generative model)



[slides G. Baydin]

Surrogates for differentiability

Algorithm 1 Local Generative Surrogate Optimization (L-GSO) procedure

Require: number N of ψ, number M of $\boldsymbol{x}$ for surrogate training, number K of $\boldsymbol{x}$ for $\boldsymbol{\psi}$ optimization step, trust region U_ϵ, size of the neighborhood ϵ, Euclidean distance d

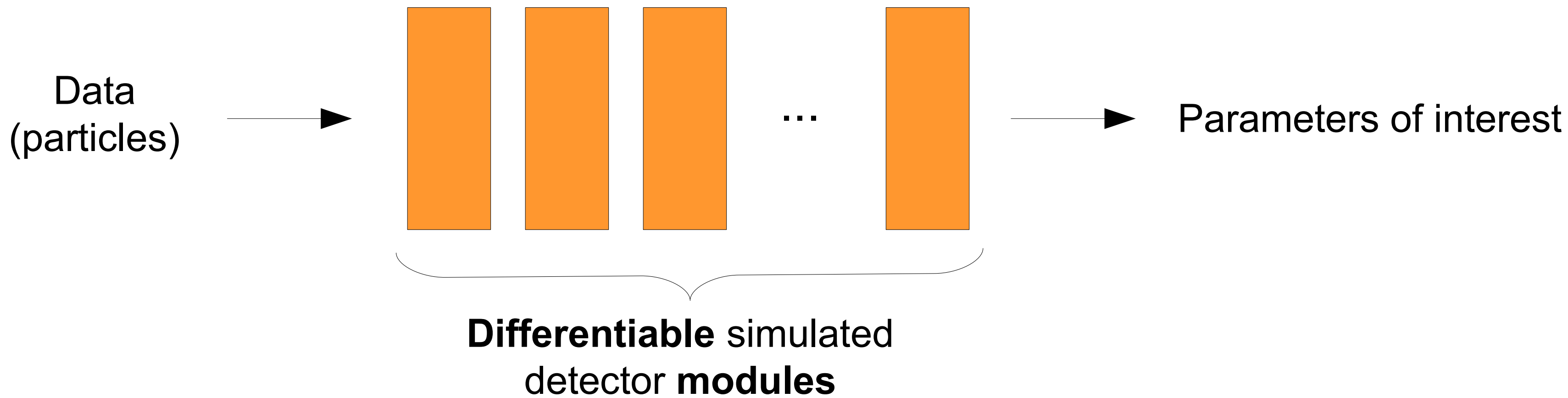
1: Choose initial parameter $\boldsymbol{\psi}$
2: **while** $\boldsymbol{\psi}$ has not converged **do**
3: Sample $\boldsymbol{\psi}_i$ in the region $U_\epsilon^{\boldsymbol{\psi}}$, $i = 1, \dots, N$
4: For each $\boldsymbol{\psi}_i$, sample inputs $\{\boldsymbol{x}_j^i\}_{j=1}^M \sim q(\boldsymbol{x})$
5: Sample $M \times N$ training examples from simulator $\boldsymbol{y}_{ij} = F(\boldsymbol{x}_j^i; \boldsymbol{\psi}_i)$
6: Store $\boldsymbol{y}_{ij}, \boldsymbol{x}_j^i, \boldsymbol{\psi}_i$ in history H $i = 1, \dots, N; j = 1, \dots, M$
7: Extract all $\boldsymbol{y}_l, \boldsymbol{x}_l, \boldsymbol{\psi}_l$ from history H, iff $d(\boldsymbol{\psi}, \boldsymbol{\psi}_l) < \epsilon$
8: Train generative surrogate model $S_\theta(\boldsymbol{z}_l, \boldsymbol{x}_l; \boldsymbol{\psi}_l)$, where $\boldsymbol{z}_l \sim \mathcal{N}(0, 1)$
9: Fix weights of the surrogate model θ
10: Sample $\bar{\boldsymbol{y}}_k = S_\theta(\boldsymbol{z}_k, \boldsymbol{x}_k; \boldsymbol{\psi})$, $\boldsymbol{z}_k \sim \mathcal{N}(0,1)$, $\boldsymbol{x}_k \sim q(\boldsymbol{x})$, $k = 1, \dots, K$
11: $\nabla_{\boldsymbol{\psi}} \mathbb{E}[\mathcal{R}(\bar{\boldsymbol{y}})] \leftarrow \frac{1}{K} \sum_{k=1}^{K} \frac{\partial \mathcal{R}}{\partial \bar{\boldsymbol{y}}_k} \frac{\partial S_\theta(\boldsymbol{z}_k, \boldsymbol{x}_k; \boldsymbol{\psi})}{\partial \boldsymbol{\psi}}$
12: $\boldsymbol{\psi} \leftarrow \mathrm{SGD}(\psi, \nabla_{\boldsymbol{\psi}} \mathbb{E}[\mathcal{R}(\bar{\boldsymbol{y}})])$
13: **end while**

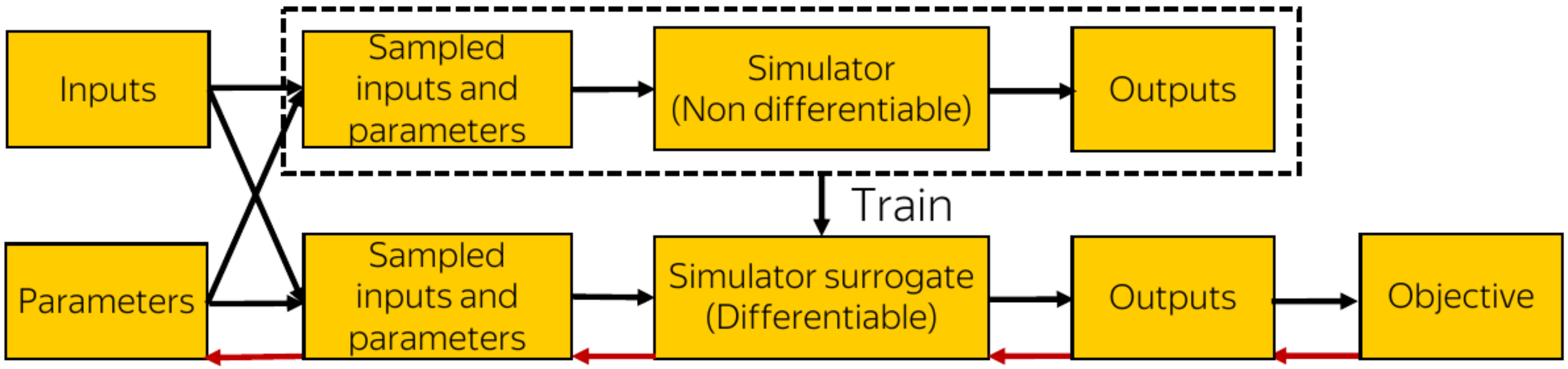
$$\boldsymbol{\psi}^* = \arg\min_{\boldsymbol{\psi}} \mathbb{E}[\mathcal{R}(\boldsymbol{y})] = \arg\min_{\boldsymbol{\psi}} \int \mathcal{R}(\boldsymbol{y}) p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\psi}) q(\boldsymbol{x}) d\boldsymbol{x} d\boldsymbol{y}$$
$$\approx \arg\min_{\boldsymbol{\psi}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{R}(F(\boldsymbol{x}_i; \boldsymbol{\psi}))$$

$$\nabla_{\boldsymbol{\psi}} \mathbb{E}[\mathcal{R}(\boldsymbol{y})] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\psi}} \mathcal{R}(S_\theta(\boldsymbol{z}_i, \boldsymbol{x}_i; \boldsymbol{\psi}))$$

ML for Detector Optimization

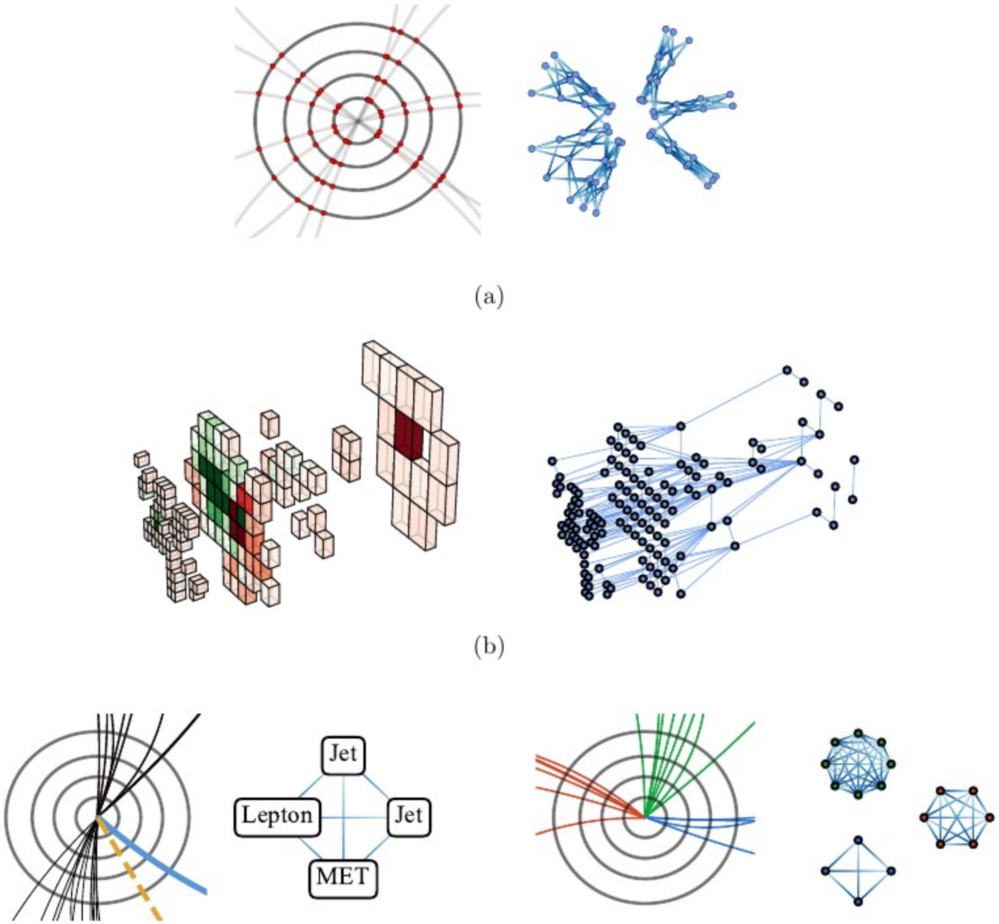
What if simulator is **not differentiable** ? Try differentiable **surrogate models**





Black-Box Optimization with Local Generative Surrogates, S. Shirobokov, V. Belavin, M. Kagan, A. Ustyuzhanin, A. G. Baydin, https://arxiv.org/abs/2002.04632

GNN for HEP



(a) (b) (c) (d)

Figure 2. HEP data lend itself to being represented as a graph for many applications: (a) clustering tracking detector hits into tracks, (b) segmenting calorimeter cells, (c) classifying events with multiple types of physics objects, (d) jet classification based on the particles associated to the jet.

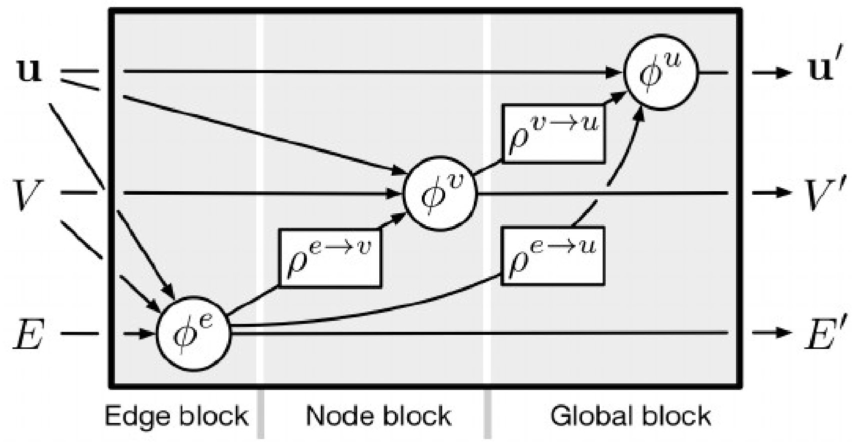
[arXiv:2007.13681]

A graph can be represented by, G = (u, V, E), with N_v vertices and N_e edges. The u represents graph-level attributes. The set of nodes is V and the set of edges is E.

A GN's stages of processing are as follows.

$$
\begin{aligned}
\mathbf{e}'_k &= \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right) & \bar{\mathbf{e}}'_i &= \rho^{e \to v}\left(E'_i\right) & &\triangleright \text{ Edge block}\\
\mathbf{v}'_i &= \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right) & \bar{\mathbf{e}}' &= \rho^{e \to u}\left(E'\right) & &\triangleright \text{ Vertex block}\\
\mathbf{u}' &= \phi^u\left(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}\right) & \bar{\mathbf{v}}' &= \rho^{v \to u}\left(V'\right) & &\triangleright \text{ Global block}
\end{aligned}
$$

A GN block contains 6 internal functions: 3 *update functions* (ϕ^e, ϕ^v, and ϕ^u) and 3 *aggregation functions* ($\rho^{e \to v}$, $\rho^{e \to u}$, and $\rho^{v \to u}$).



[arXiv:2007.13681]