Differentiable programming and detector design optimization

Seminaire LLR - 22/11/2021

Julien Donini – Université Clermont Auvergne / LPC

Outline

Differentiable programming for HEP applications

- (formal) introduction to **automatic differentiation**
- **Optimization** use-cases: **analysis** optimization, detector **design**
- **MODE** collaboration
- **Project** example: differentiable programing for **muography**

Acknowledgements

Many of the material presented in this talk was shown at the **1st Workshop on Differentiable Programming** for experimental design organised by the **MODE collaboration** last September

Thanks to

Atilim Gunes Baydin

Tommaso Dorigo

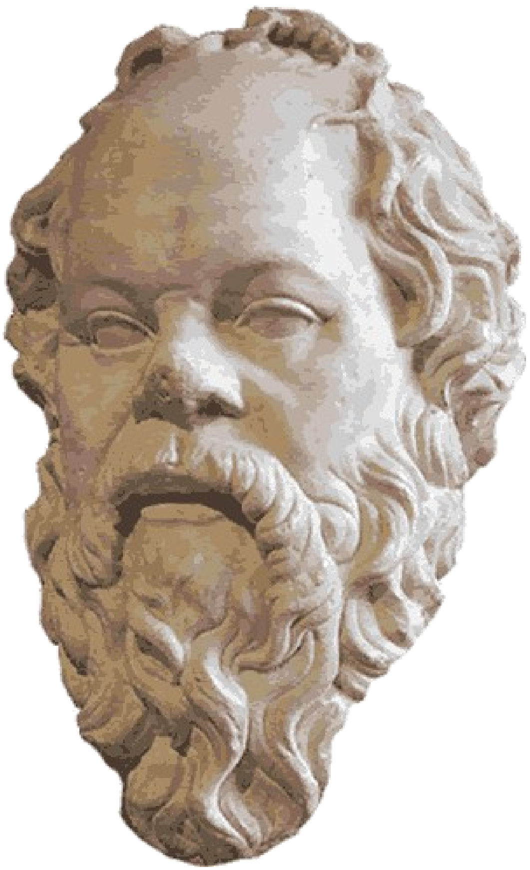
Giles Strong

Nathan Simpson

...

https://indico.cern.ch/event/1022938/

Disclaimer



"You know nothing, Jon Snow"

Warm up: ML basics

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Data

weights

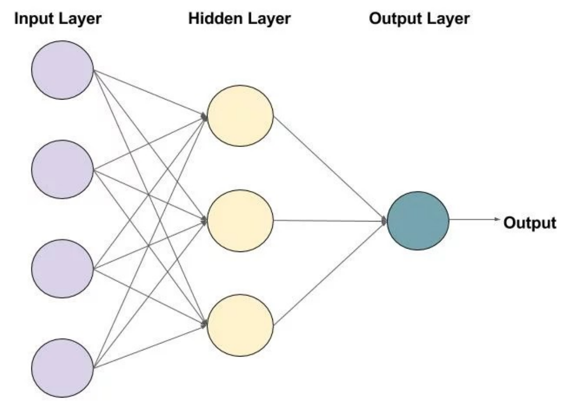
Output

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

Target

Training Neural Networks

Data
(features **x**)



Objective
(target **t**)

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

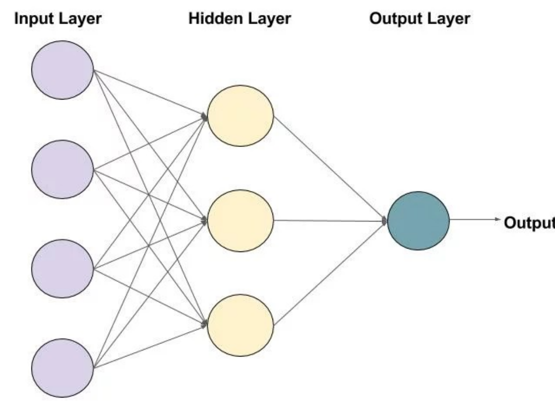
f : non-linear functions
W : parameters (weights)

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**

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

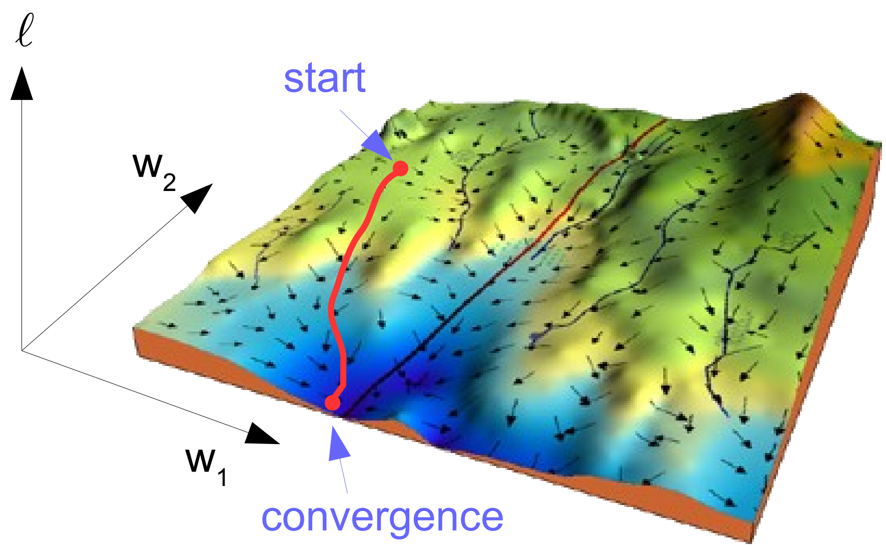
Gradient descent

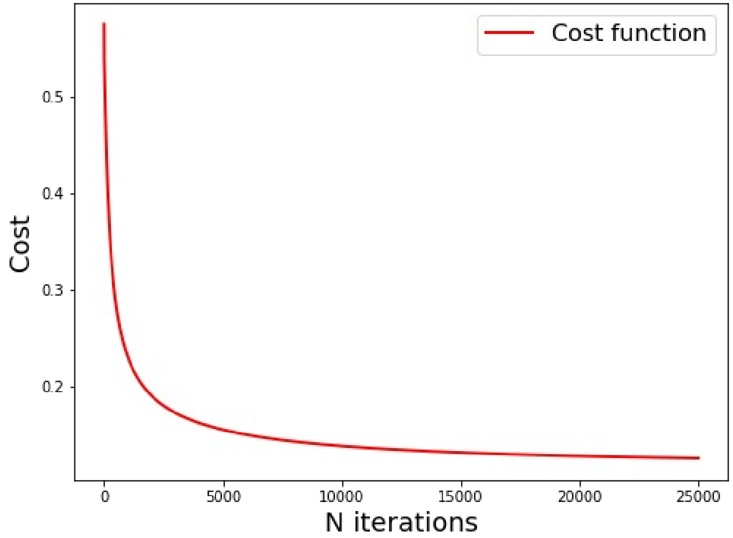
Start from initial set of weights **w** and subtract gradient of ℓ iteratively:

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

k: iteration, η: learning speed

Repeat until convergence.





Backpropagation in NN

Example: MLP network with 2 layers (1 hidden, 1 output)

Forward pass

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

Backward pass

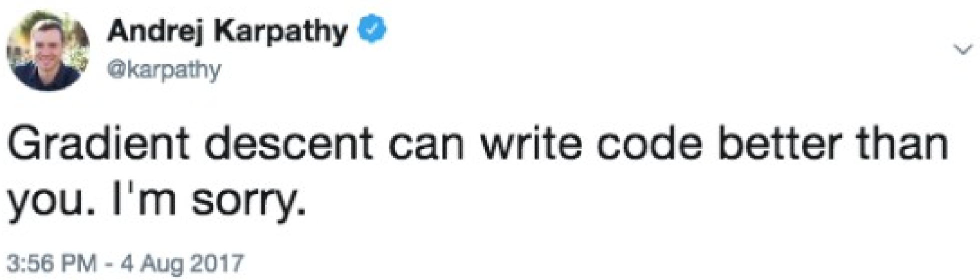
Use chain rule to compute derivatives of the loss $\ell(\mathbf{y}, \mathbf{t})$

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

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

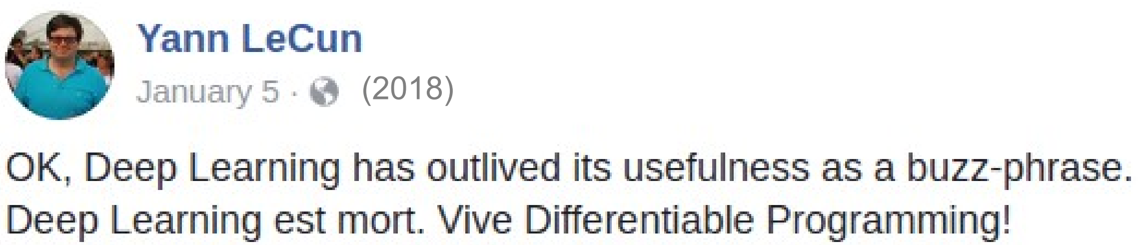
$$\frac{\partial \ell}{\partial \mathbf{W}^{(1)}} = \frac{\partial \ell}{\partial \mathbf{y}} \frac{\partial \mathbf{y}}{\partial \mathbf{s}^{(2)}} \frac{\partial \mathbf{s}^{(2)}}{\partial \mathbf{x}^{(1)}} \frac{\partial \mathbf{x}^{(1)}}{\partial \mathbf{s}^{(1)}} \frac{\partial \mathbf{s}^{(1)}}{\partial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

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


Andrej Karpathy
@karpathy

Gradient descent can write code better than you. I'm sorry.

3:56 PM - 4 Aug 2017


Yann LeCun
January 5 · (2018)

OK, Deep Learning has outlived its usefulness as a buzz-phrase.
Deep Learning est mort. Vive Differentiable Programming!

Differentiable programming

In practice: gradient-based **optimization** methods where the **derivatives** come from executing **differential code** via **automatic differentiation**

$$f : x \in \mathbb{R}^n \to \mathbb{R} \xrightarrow[\text{differentiation}]{\text{automatic}}$$

Gradient: $\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)$

Higher order derivatives: $-\eta H_f^{-1} \nabla_f$
Hessian matrix in Newton's method

➔ **Software** composed of **differentiable and parameterized building blocks,** optimized via **automatic differentiation**

Differentiable programming

Recommended reading: **Automatic Differentiation in Machine Learning: a Survey**

Baydin, Pearlmutter, Radul, Siskind. 2018. Journal of Machine Learning Research.

https://arxiv.org/abs/1502.05767

*"**4 methods** for the computation of **derivatives** in computer programs :*

*(1) **manually** working out derivatives and coding them;*

*(2) **numerical** differentiation using finite difference approximations;*

*(3) **symbolic** differentiation using expression manipulation in computer algebra*

*(4) **automatic** differentiation, also called algorithmic differentiation"*

How to code derivatives ?

$$l_1 = x$$
$$l_{n+1} = 4 l_n (1 - l_n)$$
$$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$$

Manual Differentiation

$$f'(x) = 128x(1-x)(-8+16x)(1-2x)^2(1-8x+8x^2)+64(1-x)(1-2x)^2(1-8x+8x^2)^2-64x(1-2x)^2(1-8x+8x^2)^2-256x(1-x)(1-2x)(1-8x+8x^2)^2$$

Coding

```
f(x):
  v = x
  for i = 1 to 3
    v = 4*v*(1 - v)
  return v
```

or, in closed-form,

```
f(x):
  return 64*x*(1-x)*((1-2*x)^2)
    *(1-8*x+8*x*x)^2
```

Coding

Symbolic Differentiation of the Closed-form

```
f'(x):
  return 128*x*(1 - x)*(-8 + 16*x)
    *((1 - 2*x)^2)*(1 - 8*x + 8*x*x)
    + 64*(1 - x)*((1 - 2*x)^2)*((1
    - 8*x + 8*x*x)^2) - (64*x*(1 -
    2*x)^2)*(1 - 8*x + 8*x*x)^2 -
    256*x*(1 - x)*(1 - 2*x)*(1 - 8*x
    + 8*x*x)^2
```

f'(x$_0$) $= f'(x_0)$
Exact

Automatic Differentiation

```
f'(x):
  (v,dv) = (x,1)
  for i = 1 to 3
    (v,dv) = (4*v*(1-v), 4*dv-8*v*dv)
  return (v,dv)
```

f'(x$_0$) $= f'(x_0)$
Exact

Numerical Differentiation

```
f'(x):
  h = 0.000001
  return (f(x + h) - f(x)) / h
```

f'(x$_0$) $\approx f'(x_0)$
Approximate

[1502.05767]

Automatic differentiation

Automatic (algorithmic) differentiation (AD)

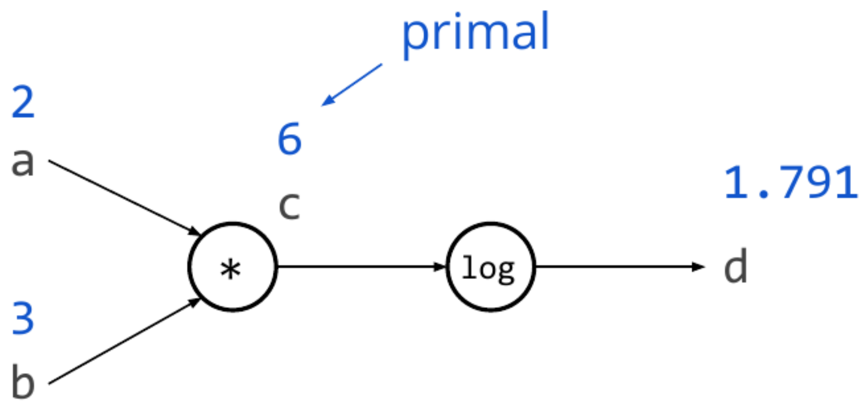
- **Numerical derivative evaluations** rather than derivative **expressions**
- Composition of **operations** for which **derivatives are known** (trace)
- **No need to rearrange** the code in a closed-form expression
- Accurate at **machine precision**

Example:

$$f(a, b) = \log(ab) \longrightarrow$$

```
f(a, b):
  c = a * b
  d = log(c)
  return d
```

Represented by a **computational graph** showing dependencies



1.791 = f(2, 3)

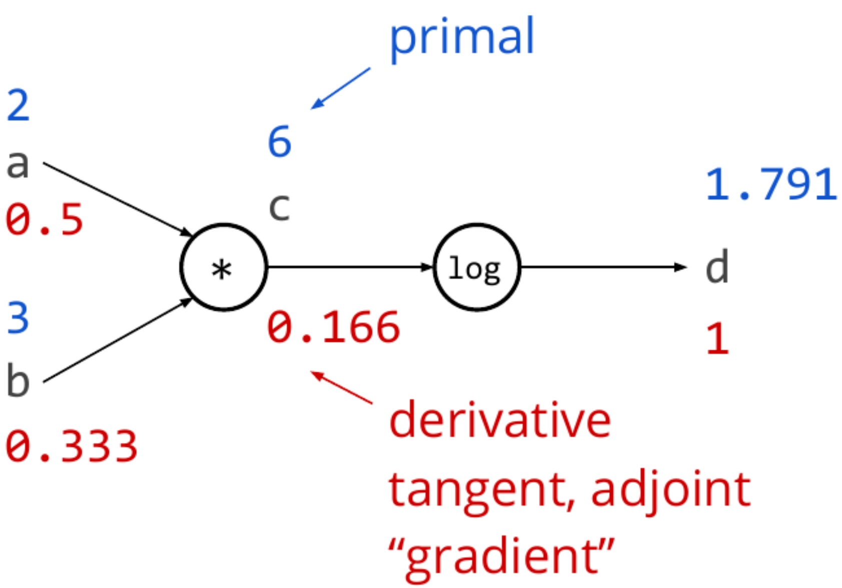
[taken from G. Baydin]

Example:

$$f(a, b) = \log(ab)$$

```
f(a, b):
   c = a * b
   d = log(c)
   return d
```

Represented by a **computational graph** showing dependencies




```
1.791 = f(2, 3)
[0.5, 0.333] = f'(2, 3)
```

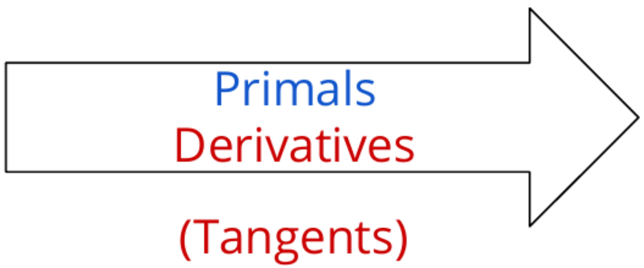
$$\nabla f(a, b) = (1/a, 1/b)$$

[taken from G. Baydin]

Automatic differentiation

Two **main modes**, both based on chain rule

Forward mode

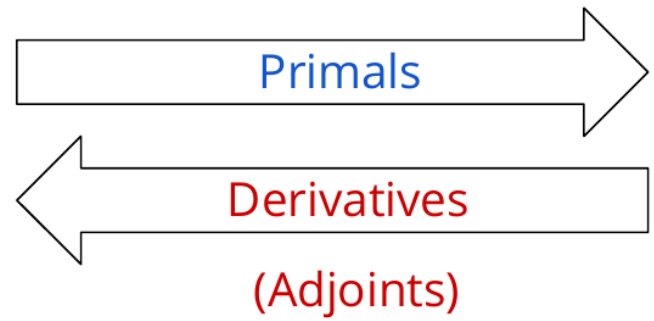


Associate each **intermediate variable** v_i with a derivative $\dot{v}_i = \frac{\partial v_i}{\partial x}$

Apply chain rule to each **elementary operations** in Forward propagation

Best suited for $f : \mathbb{R}^n \to \mathbb{R}^m, n \ll m$

Reverse mode (backpropagation)



Propagates derivatives **backwards from output** $\bar{v}_i = \frac{\partial f}{\partial v_i}$

Two phases

1. Calculate **intermediate** variables v_i
2. Calculate **derivatives**: output → input

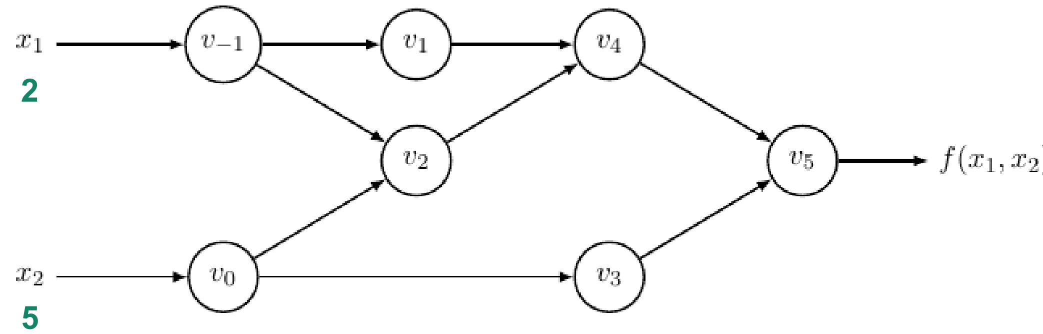
Best suited for $f : \mathbb{R}^n \to \mathbb{R}^m, m \ll n$

[figure G. Baydin]

Example

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

Each **intermediate variable** is associated to $\dot{v}_i = \dfrac{\partial v_i}{\partial x}$



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	$= v_5$	$= 11.652$

Forward Tangent (Derivative) Trace		
$\dot{v}_{-1}$	$= \dot{x}_1$	$= 1$
$\dot{v}_0$	$= \dot{x}_2$	$= 0$
$\dot{v}_1$	$= \dot{v}_{-1}/v_{-1}$	$= 1/2$
$\dot{v}_2$	$= \dot{v}_{-1} \times v_0 + \dot{v}_0 \times v_{-1}$	$= 1 \times 5 + 0 \times 2$
$\dot{v}_3$	$= \dot{v}_0 \times \cos v_0$	$= 0 \times \cos 5$
$\dot{v}_4$	$= \dot{v}_1 + \dot{v}_2$	$= 0.5 + 5$
$\dot{v}_5$	$= \dot{v}_4 - \dot{v}_3$	$= 5.5 - 0$
$\boldsymbol{\dot{y}}$	$\boldsymbol{= \dot{v}_5}$	$\mathbf{= 5.5}$

Forward mode example, evaluated at $(x_1, x_2) = (2, 5)$ and setting $\dot{x}_1 = 1$ to compute $\dot{y} = \dfrac{\partial y}{\partial x_1}$

[1502.05767]

Digression: dual numbers

Forward mode can be viewed as evaluating a function using **dual numbers**

Numbers defined as $v + \dot{v}\epsilon$ where $\epsilon \neq 0$ and $\epsilon^2 = 0$

Properties (using Taylor expansion):

$$f(v + \dot{v}\epsilon) = f(v) + f'(v)\dot{v}\epsilon$$

$$\begin{aligned} f(g(v + \dot{v}\epsilon)) &= f(g(v) + g'(v)\dot{v}\epsilon)) \\ &= f(g(v)) + \boxed{f'(g(v))g'(v)}\dot{v}\epsilon \end{aligned}$$

Composite function derivative !

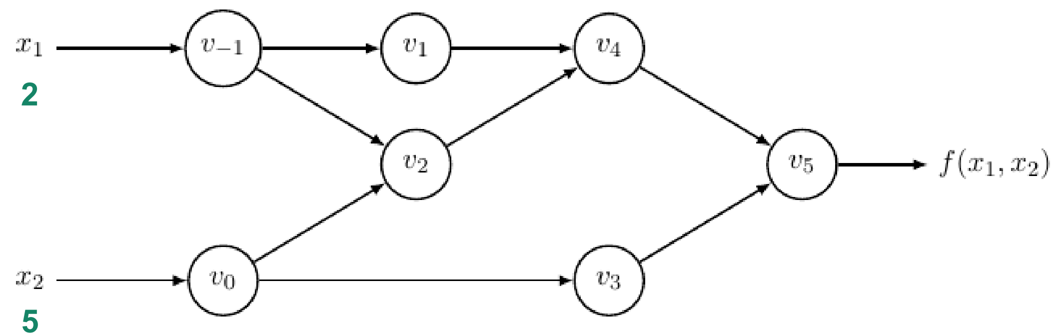
In practice, implement **specific code** to handle the **dual operations** so that **function f and its derivative are simultaneously computed** (operator overloading)

Reverse mode (backpropagation)

Example

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

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



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

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]

Current state of differentiable programming

Evolution of frameworks

From: **coarse-grained (module level) backprop**
Towards: **fine-grained, general-purpose automatic differentiation**

theano *2008* →

torch7 *2011* → **torch-autograd** *2015*

PyTorch *2016* →

HIPS autograd *2014* →

TensorFlow *2015* → TensorFlow eager exec *2017* → **TensorFlow 2** *2019* →

JAX *2018* →

Auto-diff tools: http://www.autodiff.org/

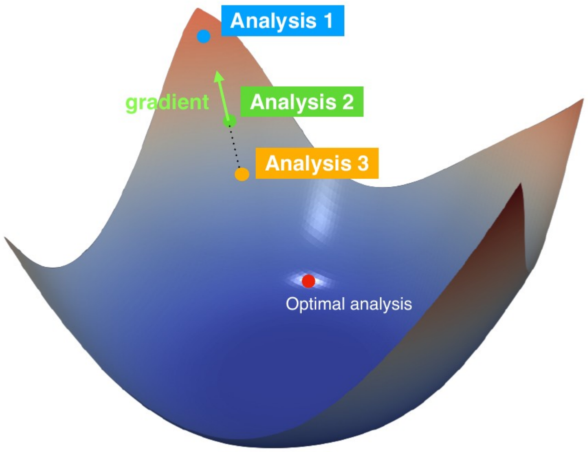
[slide G. Baydin]

Differentiable programming in HEP

Incorporating automatic differentiation in HEP software

Differentiable Programming in Analysis Code

- **Optimize** free **parameters** of an analysis with respect to the **desired physics objective**
- End-to-end **differentiable** analysis **workflow**



[figure N. Simpson]

Differentiable Programming in Simulation Code

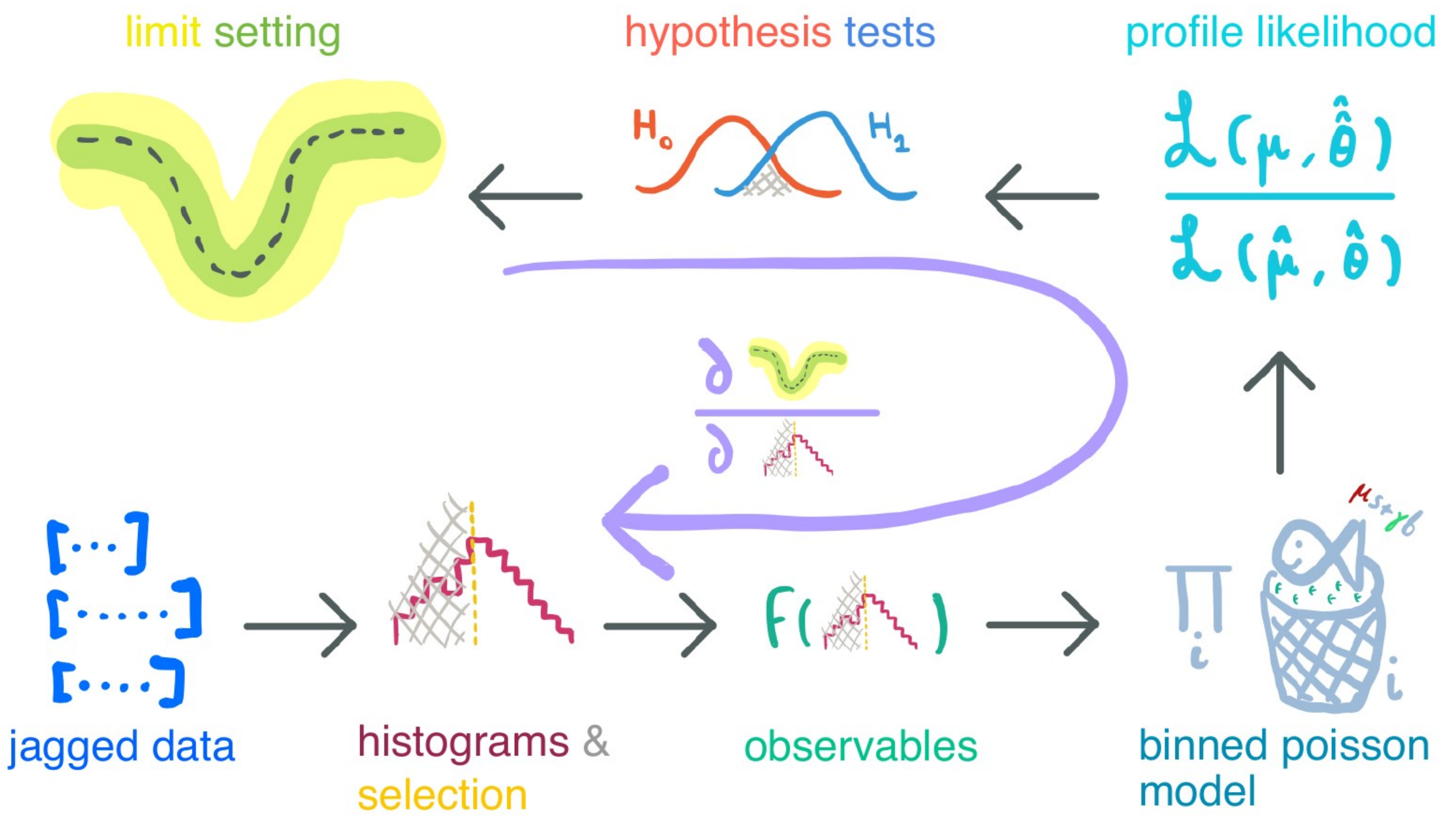
- Compute **gradient** for simulated samples with respect to **parameters of simulation**
- **EFT**, **Cosmology**, **MadGraph** (evaluation of matrix element using autodiff), ...

Differentiable programming in High Energy Physics, SnowMass 2021

Differentiable analysis: NEOS

End-to-end optimized **analysis pipelines** that use the **analysis sensitivity** including **systematic** uncertainties as the **objective** function

github.com/gradhep/neos

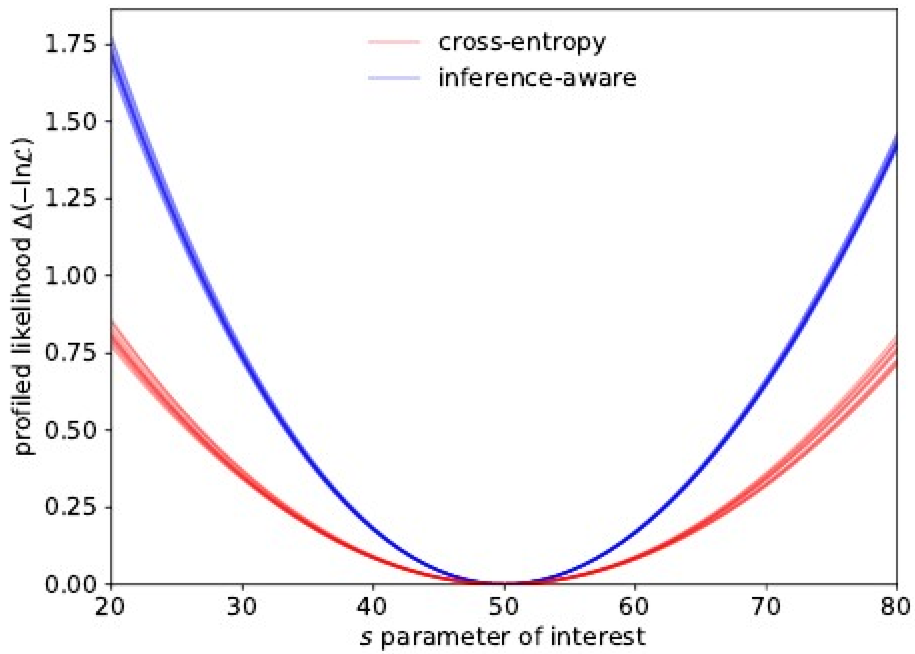


[Slide Nathan Simpson]

INFERNO

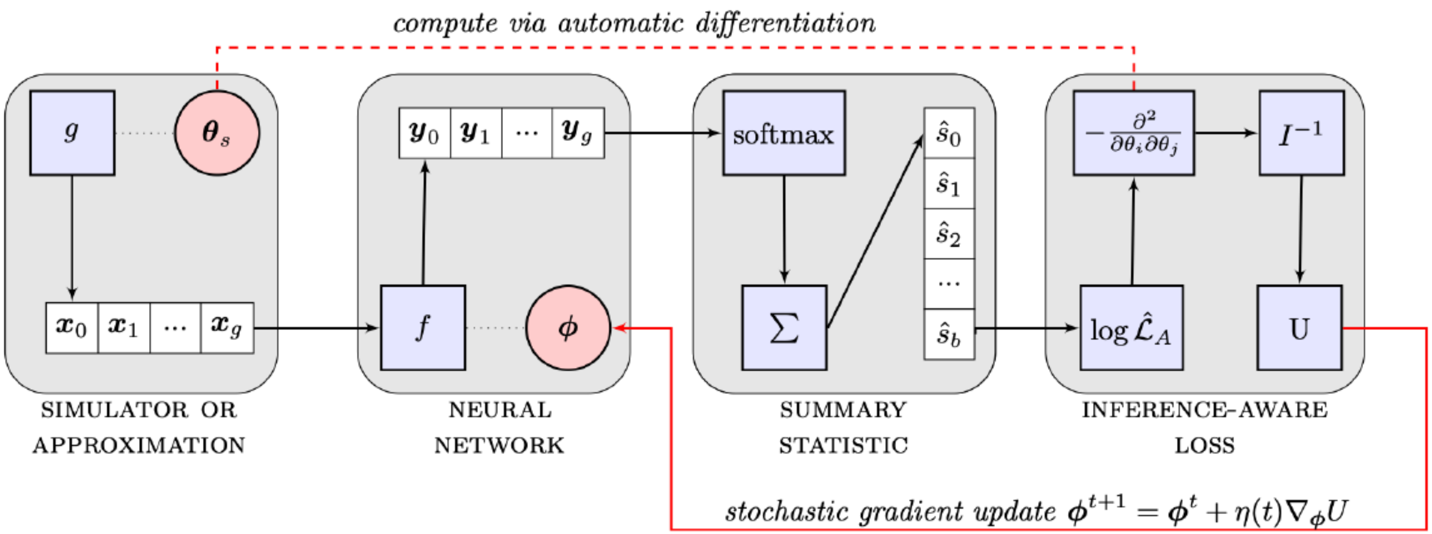
Inference Aware **N**eural **O**ptimization [1806.04743, de Castro, Dorigo]

- **Include nuisance parameters** in the loss function and directly minimize **variance of parameters of interest**



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

INFERNO algorithm



Algorithm 1 Inference-Aware Neural Optimisation.

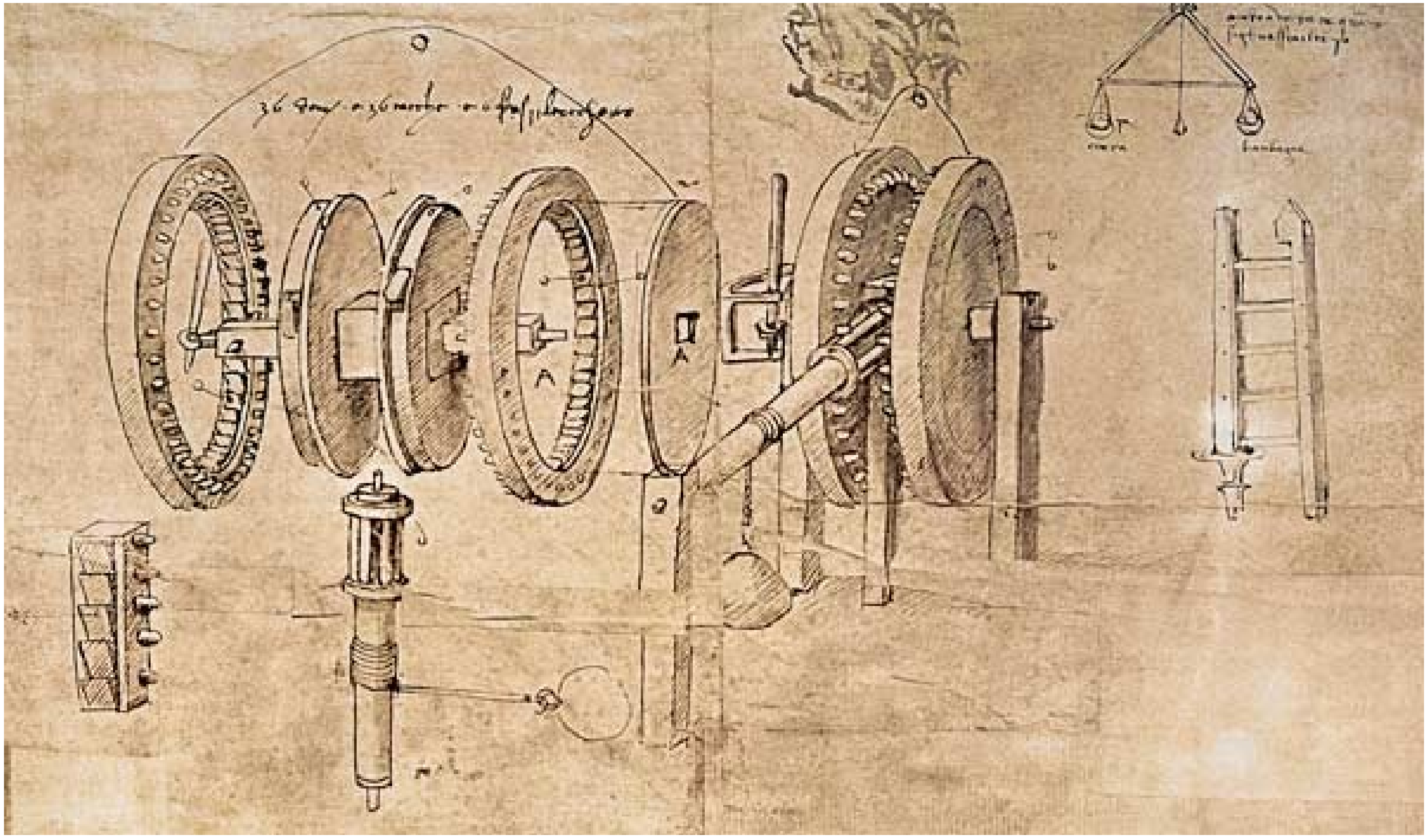
Input 1: differentiable simulator or variational approximation $g(\boldsymbol{\theta})$.
Input 2: initial parameter values $\boldsymbol{\theta}_s$.
Input 3: parameter of interest $\omega_0 = \theta_k$.
Output: learned summary statistic $\boldsymbol{s}(D; \boldsymbol{\phi})$.

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

[taken from 1806.04743]

Optimization of detector design

Can automatic differentiation be applied to detector optimization ?



Optimization of detector design

Design of detectors for particle physics applications traditionally **relies** on **individual optimization of each 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

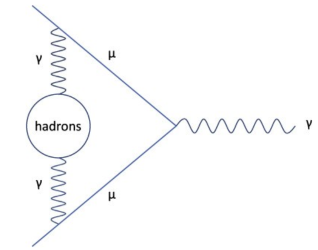
Optimization of a **joint problem** ≠ different from **individual optimization**

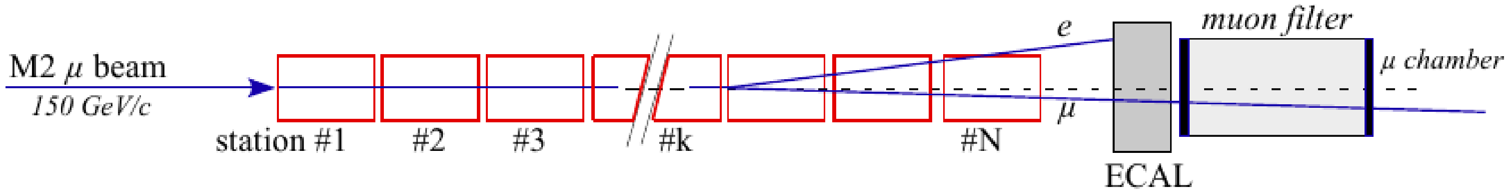
$$argmax_{x,y}(\mathcal{L}(x,y)) \neq \left[argmax_x(\int \mathcal{L}(x,y)dy), argmax_y(\int \mathcal{L}(x,y)dx)\right]$$

Proof of concept: MUonE experiment

Example of geometry optimization: **MUonE** experiment

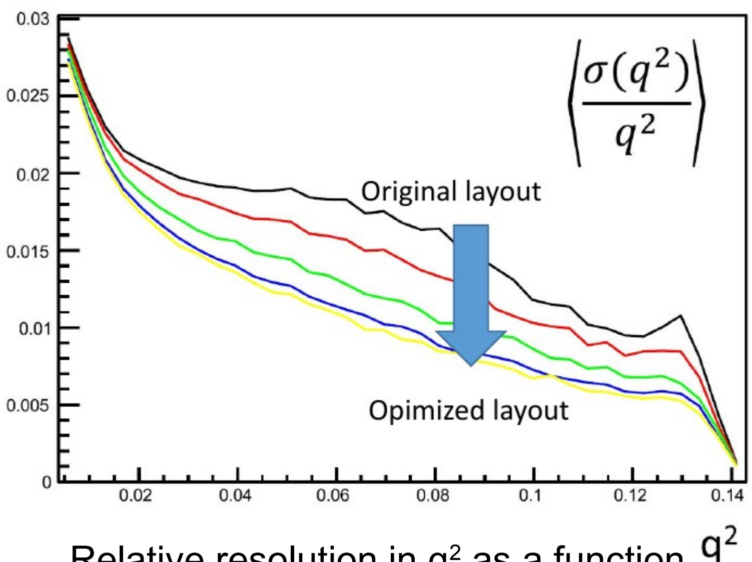
- MUonE: high precision **muon-electron** differential cross section
  → **hadronic contributions** to **g-2 muon anomaly**





Optimizing **geometry** of the detector

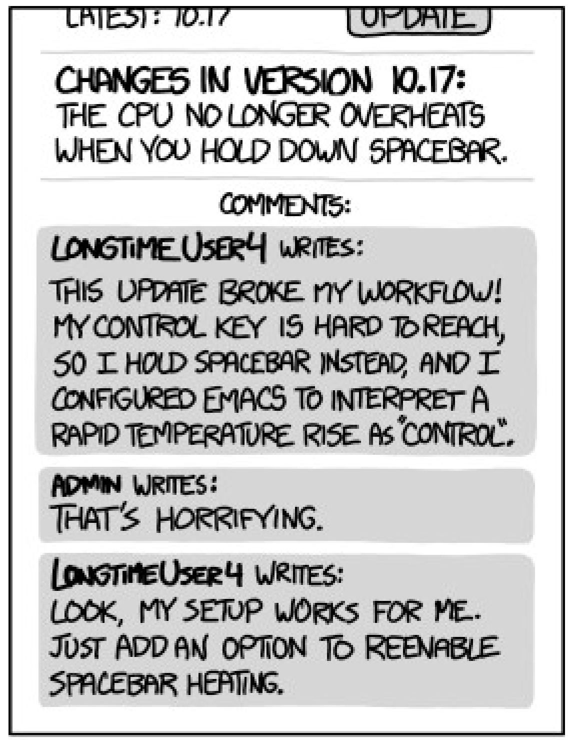
- Likelihood minimization (not AD)
- **Factor 2 improvement** in FOM
- No increase of **detector cost**



Relative resolution in q^2 as a function of q^2. The higher black line is the original proposal by the MUonE coll.

[T. Dorigo, https://doi.org/10.1016/j.physo.2020.100022]

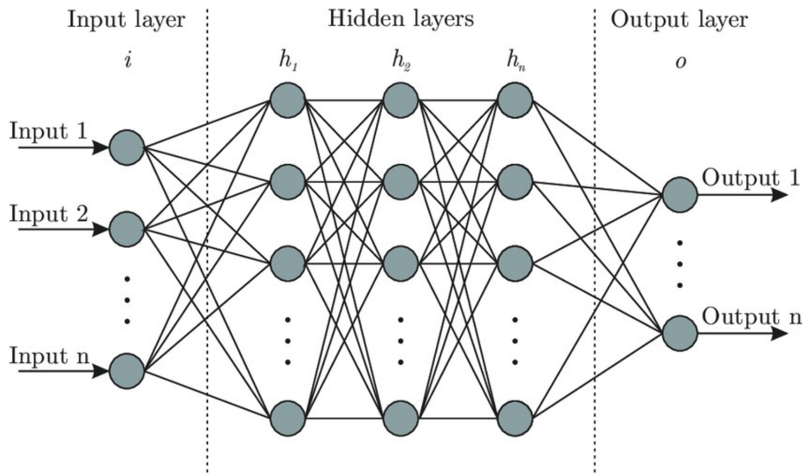
Generic optimization pipeline



EVERY CHANGE BREAKS SOMEONE'S WORKFLOW.

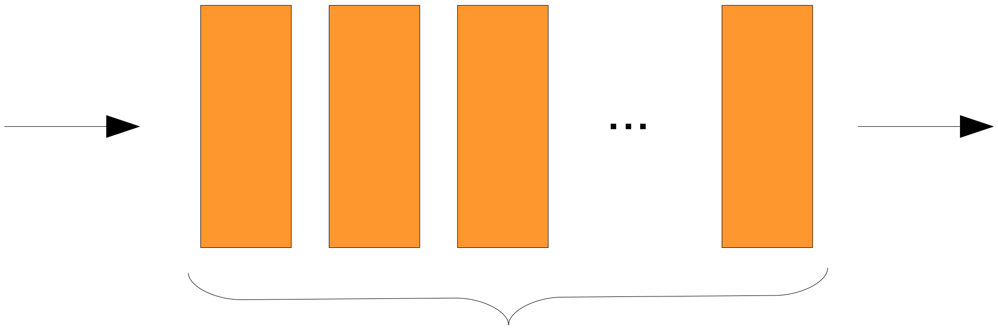
Generic optimization pipeline

Data



NN output

Data
(particles)



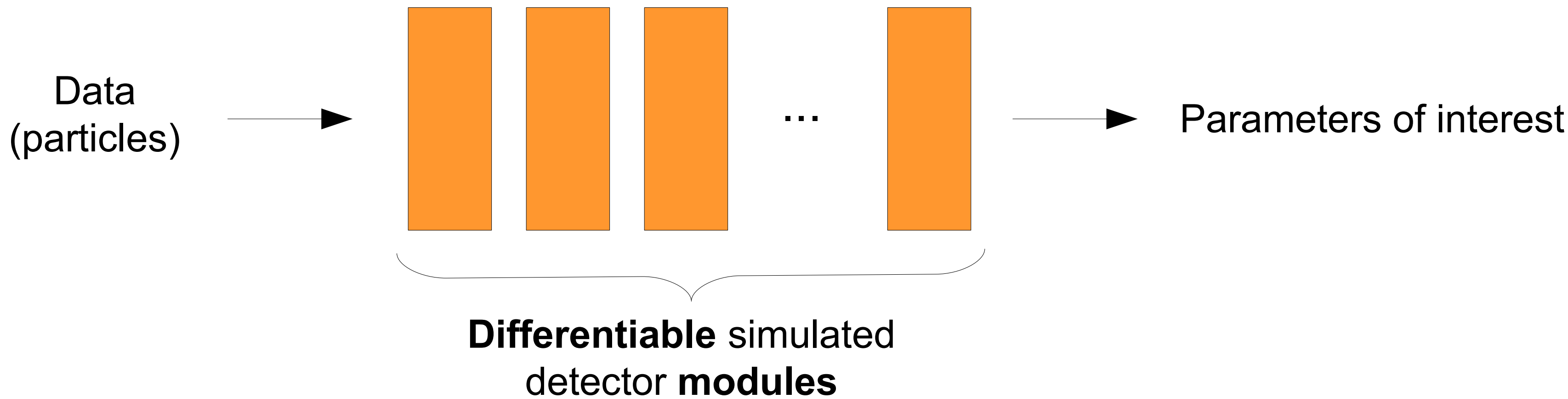
Parameters of interest

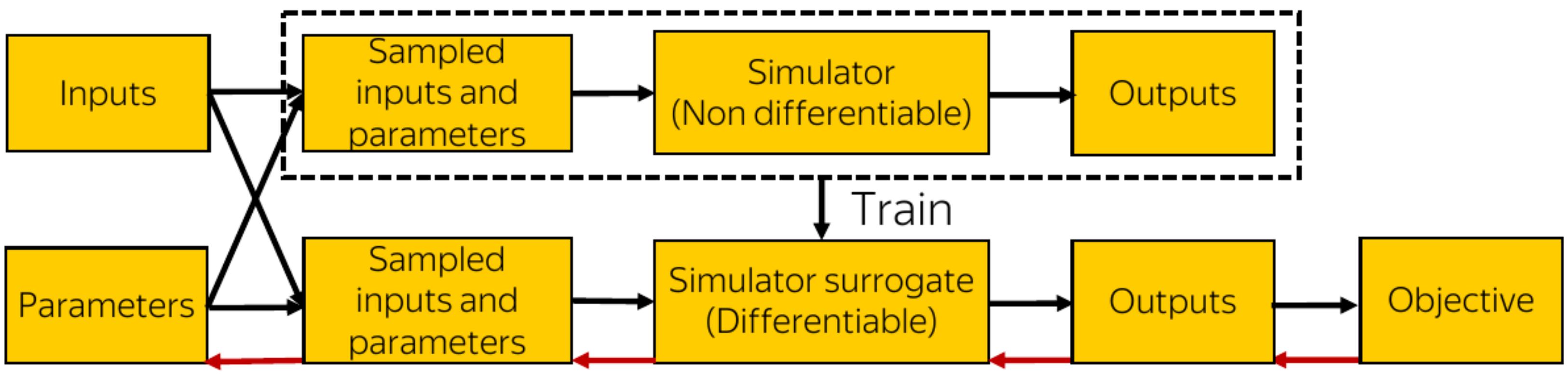
Differentiable simulated
detector **modules**

Minimization of objective function through automatic differentiation

Generic optimization pipeline

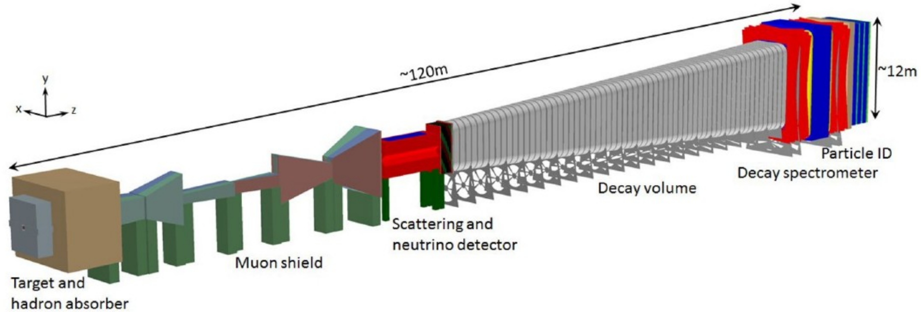
What if simulator is **not differentiable** ? Try generative **surrogate**



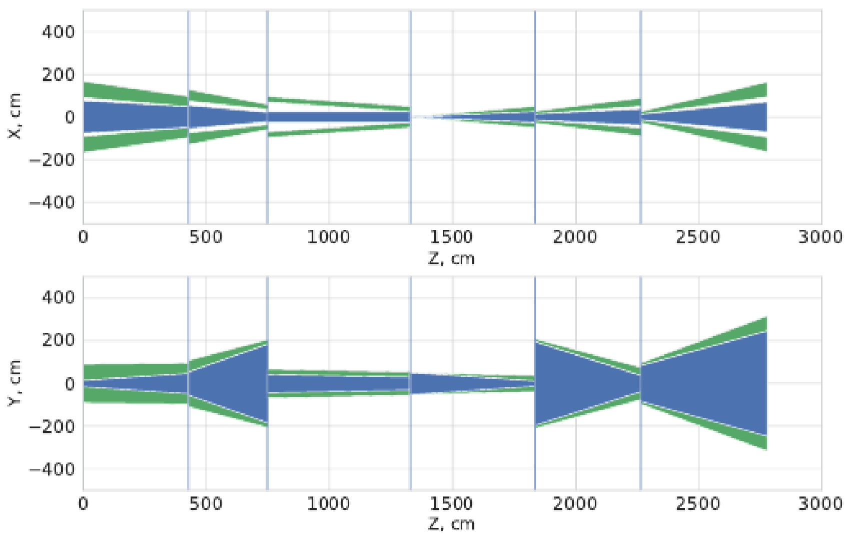


Black-Box Optimization with Local Generative Surrogates, S. Shirobokov, V. Belavin, M. Kagan, A. Ustyuzhanin, A. G. Baydin, https://arxiv.org/abs/2002.04632

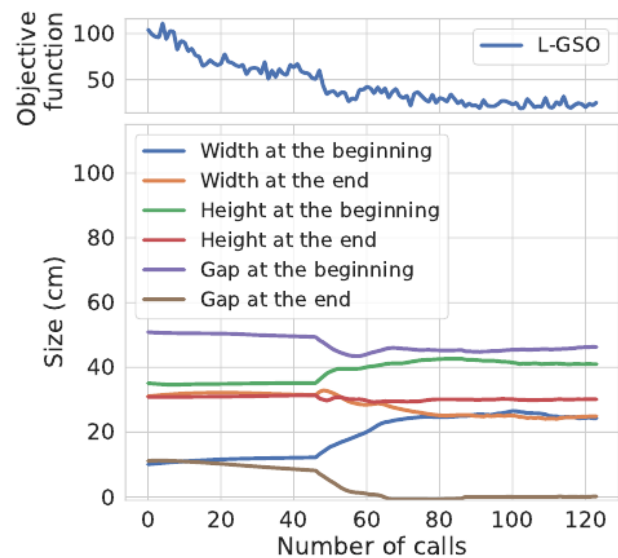
Minimize muon background fluxes in the **SHIP** steel **magnet** by varying its **geometry**



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



Evolution of 6 parameters
during **optimization**

[2002.04632]

Machine-Learning ***O***ptimized ***D***esign of ***E***xperiments

MODE Collaboration

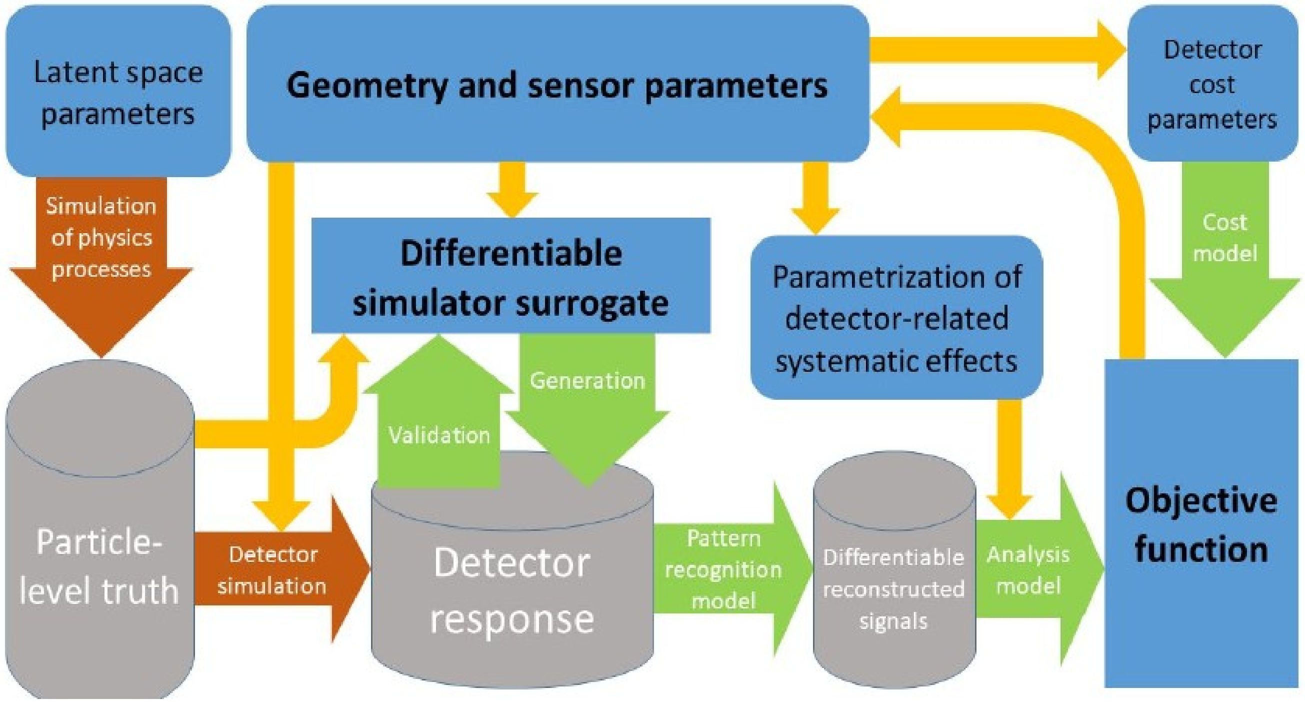
https://mode-collaboration.github.io

A. G. Baydin[5], A. Boldyrev[4], K. Cranmer[8], P. de Castro Manzano[1], T. Dorigo[1], C. Delaere[2], D. Derkach[4], J. Donini[3], A. Giammanco[2], J. Kieseler[7], G. Louppe[6], L. Layer[1], P. Martinez Ruiz del Arbol[9], F. Ratnikov[4], G. Strong[1], M. Tosi[1], A. Ustyuzhanin[4], P. Vischia[2], H. Yarar[1] + 8 members that joined recently

1 INFN, Sezione di Padova (and associates from Padova and Naples Universities), Italy
2 Université Catholique de Louvain, Belgium
3 Université Clermont Auvergne, France
4 Laboratory for big data analysis of the Higher School of Economics, Russia
5 University of Oxford
6 Université de Liege
7 CERN
8 New York University
9 IFCA

MODE ultimate goals

The target of **MODE** is to design and offer to the community a scalable, versatile **architecture** that can provide **end-to-end optimization of particle detectors**, proving it on a number of **different applications** across different **domains**



[taken from T. Dorigo]

MODE workshop(s)

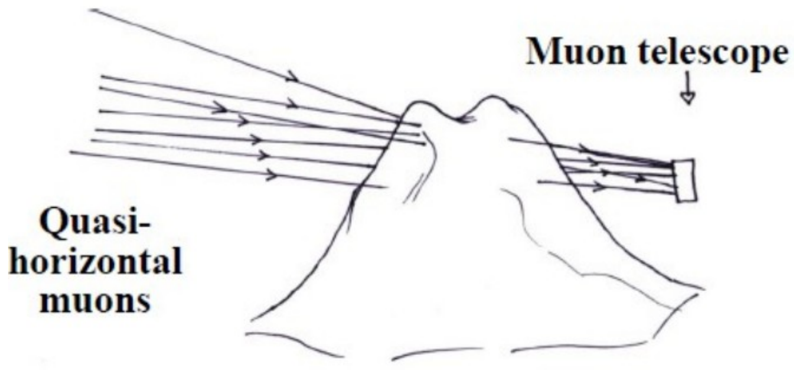
Started series of **workshop** on **automatic differentiation for experimental design**

- 1st edition: 6–8 September 2021, Louvain-la-Neuve, https://indico.cern.ch/event/1022938/
- Sponsored by JENAA (Joint APPEC, ECFA and NuPECC) and IRIS-HEP
- 105 participants (one third of which present in person), about 30 talks

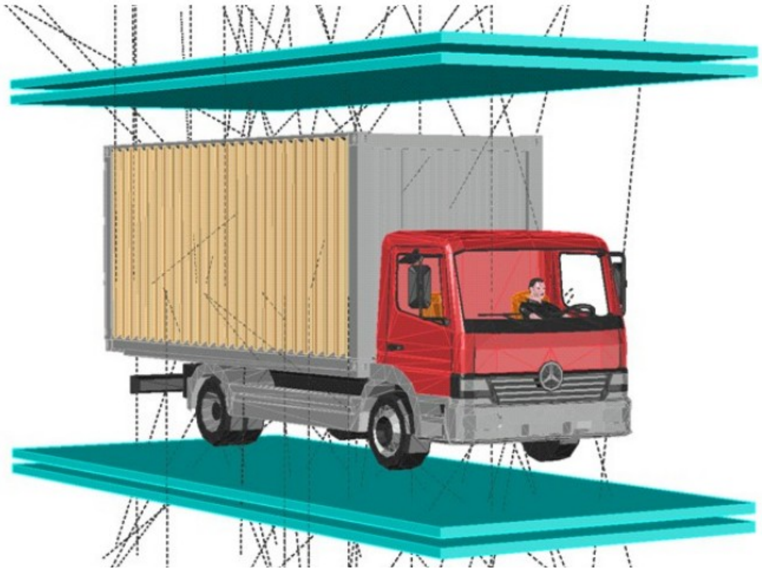
Differentiable programming for muography

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

Muon absorption

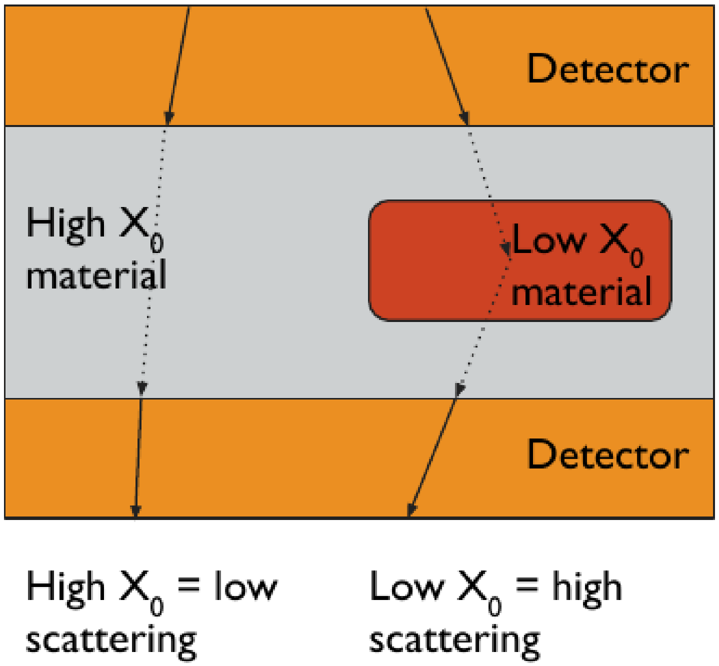


Muon scattering



[images : A. Giammanco]

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**, ...

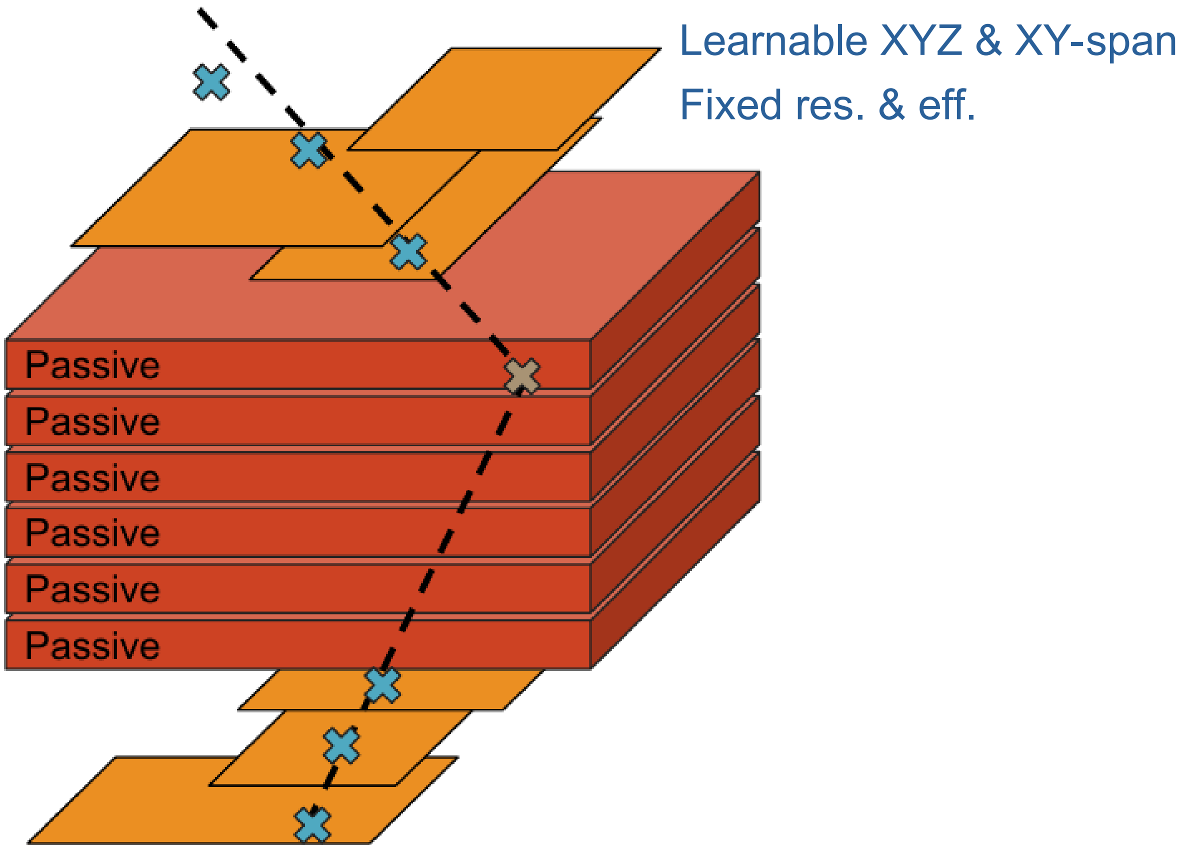
[see G. Strong talk]

TomOpt: Tomography Optimization

“Simple” use-case : **muon scan of volume of unknown density**

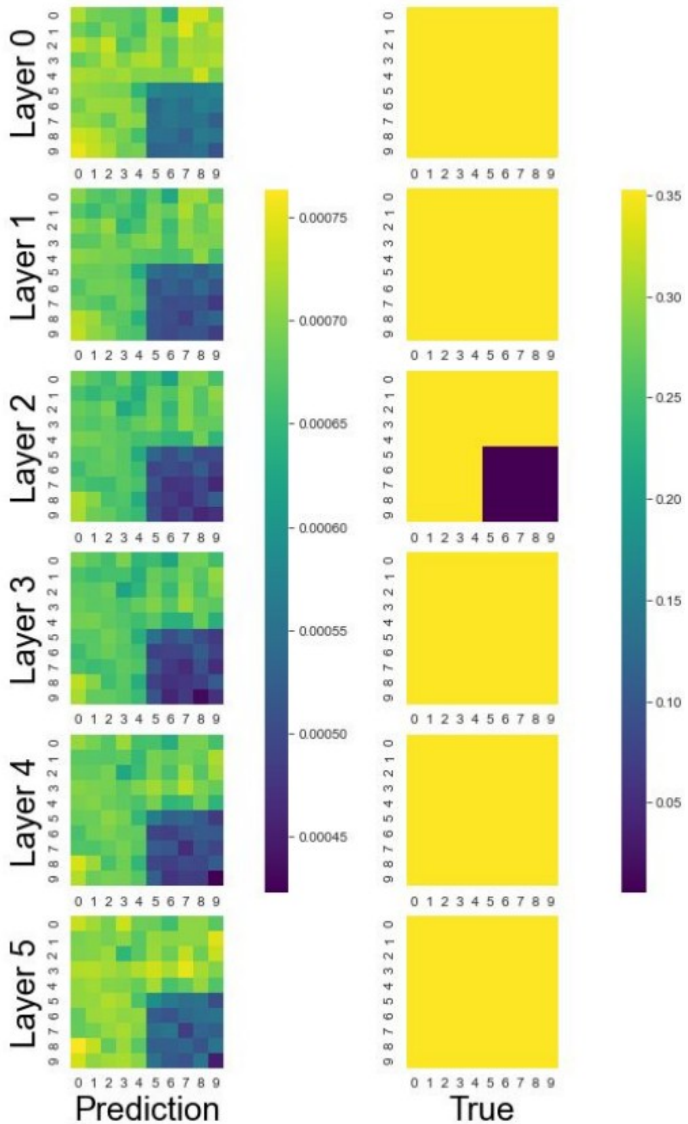
Still under **development**: code and results from **Giles Strong**

Promizing first results already achieved



[G. Strong talk]

TomOpt: Tomography Optimization



Example volume

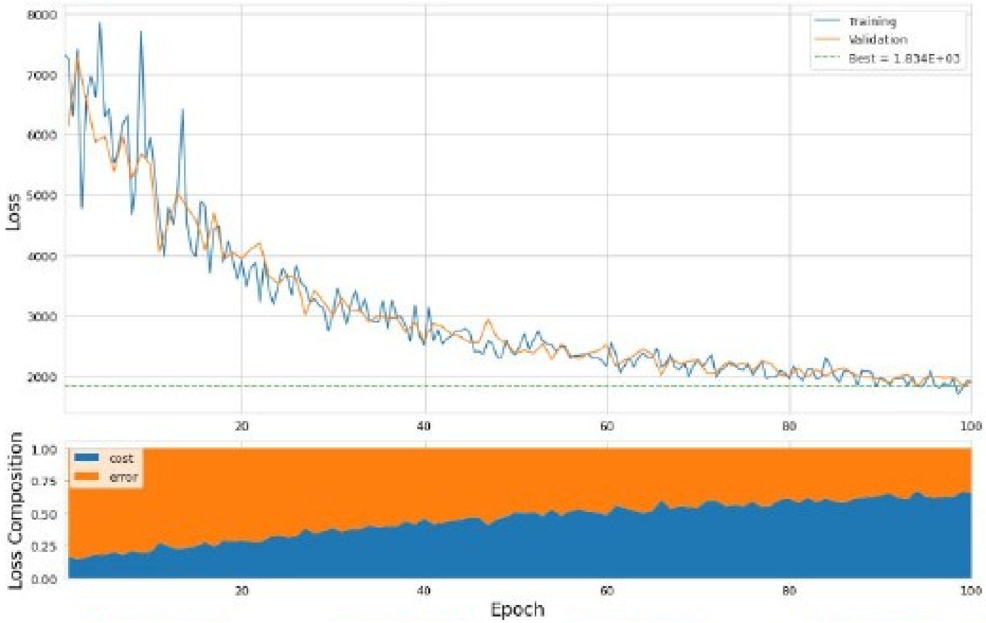
- Block of lead (X_0=0.005612m)
- Surrounded by beryllium (X_0=0.3528m)

Prediction based on 100k muons

- 2h computation time
- Lead block clearly visible, but high z uncertainty in scatter location causes 'ghosting' above and below

[G. Strong talk]

TomOpt: Tomography Optimization



Loss function:

$$\mathcal{L}_{\text{Error}} = \frac{1}{N_{\text{voxels}}} \sum_{i=1}^{N_{\text{voxels}}} \frac{\left(X_{0,i,\text{True}} - X_{0,i,\text{Pred.}}\right)^2}{w_i}$$

$$\mathcal{L}_{\text{Cost}} = \sum_{i=1}^{N_{\text{panels}}} f_i \left(\text{span}_{x,i}, \text{span}_{y,i}\right)$$

$$\mathcal{L} = \mathcal{L}_{\text{Error}} + \alpha \mathcal{L}_{\text{Cost}}$$

Still a long way to go, but an important milestone for this use case

[G. Strong talk]

Conclusion

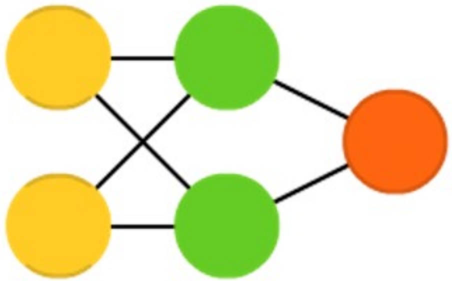
Differentiable programming paradigm opens to many different **applications**

For HEP: end-to-end **optimization** of analysis, simulators, detectors, ...

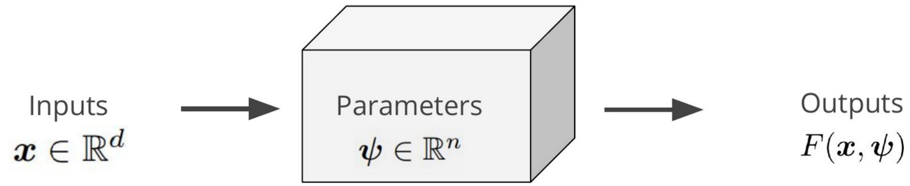
MODE collaboration: ML optimization of **detector design**

- Several **projects**: muon tomography (advanced), muon collider detector shielding (starting), Hybrid calorimeter (staring) + few others considered
- We know this is a **challenging** and **ambitious** task !
- Objective is **not to substitute experts** in detector design
- **Domain knowledge crucial** in setting up analysis workflow
- Consider joining and bring you **use case**

Backup material



Surrogates for differentiability

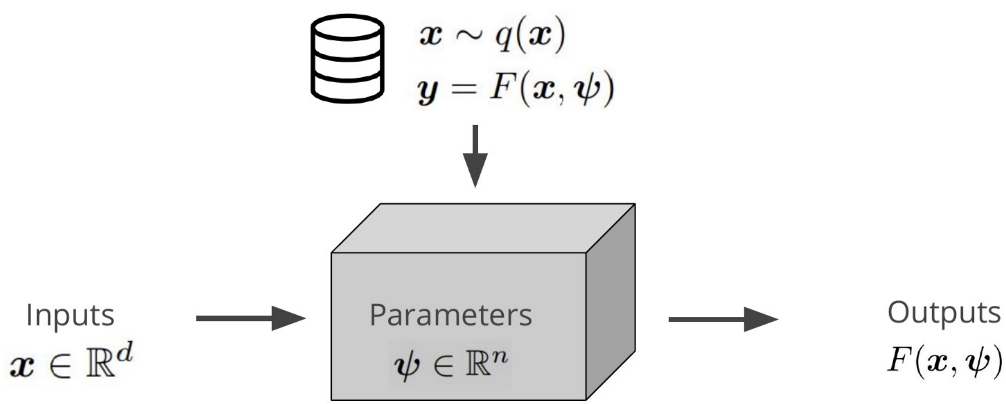


Non-differentiable simulator (model)

- 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)



Differentiable surrogate with $\nabla_{\boldsymbol{\psi}} F(\boldsymbol{x}, \boldsymbol{\psi})$

[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 ψ 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**

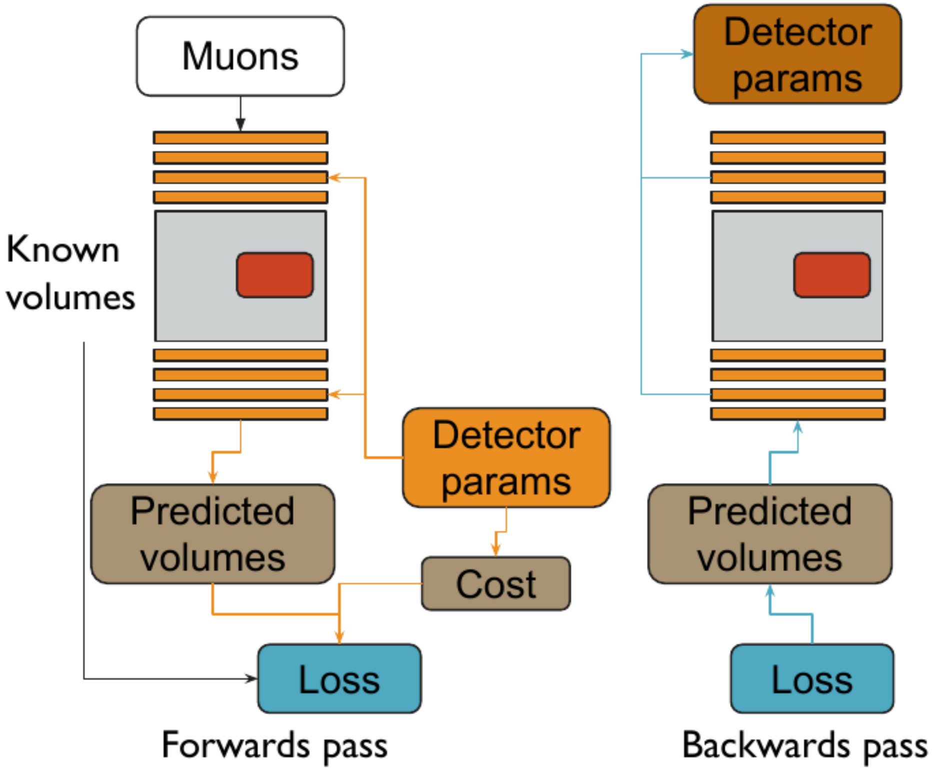
$$\boldsymbol{\psi}^* = \arg\min_{\boldsymbol{\psi}} \mathbb{E}[\mathcal{R}(\boldsymbol{y})] = \arg\min_{\boldsymbol{\psi}} \int \mathcal{R}(\boldsymbol{y}) p(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{\psi}) q(\boldsymbol{x}) d\boldsymbol{x} d\boldsymbol{y}$$

$$\approx \arg\min_{\boldsymbol{\psi}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{R}(F(\boldsymbol{x}_i; \boldsymbol{\psi}))$$

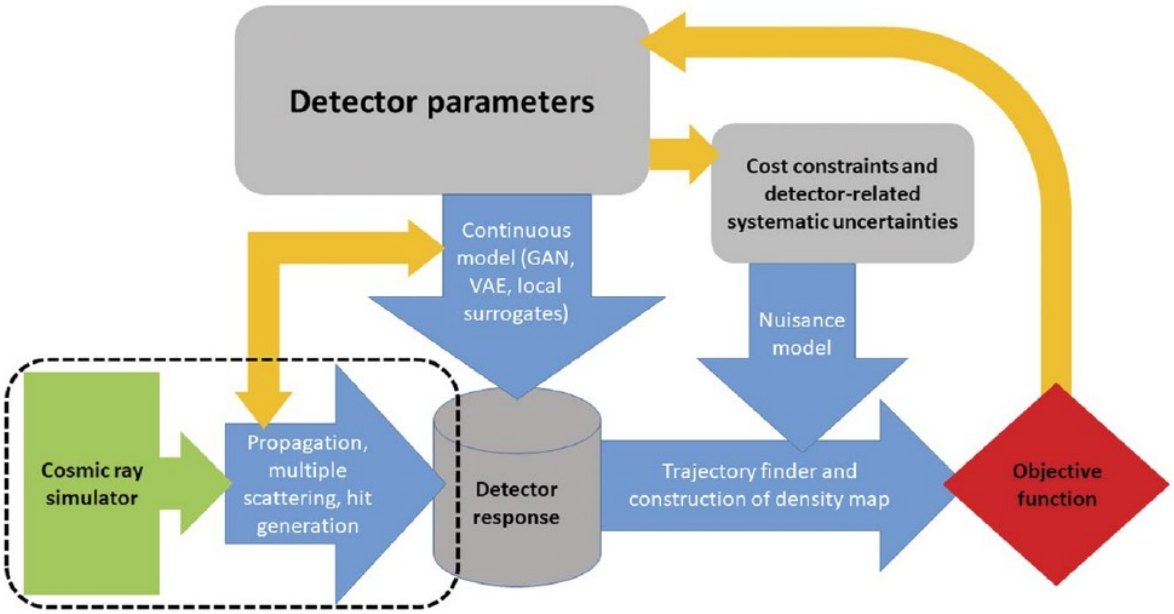
$$\nabla_{\boldsymbol{\psi}} \mathbb{E}[\mathcal{R}(\boldsymbol{y})] \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\psi}} \mathcal{R}(S_\theta(\boldsymbol{z}_i, \boldsymbol{x}_i; \boldsymbol{\psi}))$$

TomOpt: end-to-end optimization

- Consider instead simulating muon propagation and expressing the entire inference chain as a differentiable system
  - We can now compute the analytical effects of detector parameters (position, size, resolution, etc.) on system outputs
- Now express the desired task as a loss function
  - E.g. error on X_0 predictions, detector costs, time to achieve desired resolution
- We can now backpropagate the loss gradient to detector parameters and optimise via gradient descent
  - Just like a neural network



[G. Strong talk]



Conceptual layout of an optimization pipeline for a muon radiography apparatus. Modules within the dashed black box inform the validation of a continuous model and are not part of the optimization flow

[G. Strong talk]

Joining MODE ?

If you are doing experimental research in HEP, astro-HEP, neutrino physics, or high-energy nuclear physics, or if you are working at spin-offs involving, *e.g.*, muon tomography, hadron therapy, or other endeavours which operate with instruments that extract information from the interaction of energetic radiation with matter, you are very likely to have a use case – a system liable to benefit from a study with differentiable programming.

The idea of MODE is to bring together ML experts who are developing the interfaces for these applications, with the researchers who have problems to solve in their area of interest

We cannot offer a solution to any given problem (we lack the manpower to work on-demand), but together we may work toward it

□ **Consider joining MODE, and bring your use case!**

Do I have a use case checklist:

Are you involved in the design, assembly, or upgrade of an instrument?

Can you specify one or a set of desirable scientific goals from its use?

Are those goals achieved through information processing?

If your answers to all are «yes», **you have something to optimize** and chances are this can't be done without a deep learning model of the full information extraction chain.

[slide T. Dorigo]