Differentiable Matrix Elements with MadJax

Lukas Heinrich, CERN
Michael Kagan, SLAC
Introduction

Matrix Elements Calculations are the bread and butter of high-energy physics
• most closely connected to the hard-processes we are interested in
• exploiting more information from Matrix Elements always useful

\[ \sigma(z \mid \theta) \]
\[ \sigma(\theta) = \int dz \, \sigma(z \mid \theta) \]
\[ p(x \mid \theta) = \frac{1}{\sigma(\theta)} \sigma(x \mid \theta) \]

This talk:
• automatic calculation of Matrix Element Derivatives inspired by ML methods

\[ \nabla_z \sigma(z \mid \theta) = \sigma(z \mid \theta) \]
\[ \nabla_\theta \sigma(z \mid \theta) = \sigma(z \mid \theta) \]
Context: the rise of differentiable programming

Modern Deep Learning at its core is about gradients
- vast number of neural net architectures but all optimized using gradient descent

New focus on making computational science pipeline differentiable
- allows us to freely mix & match ML modules with physics modules
- towards "physics"-aware ML methods
- trend goes beyond HEP

Beyond ML:
- gradients are useful in a lot of contexts
- but ML gave us new tools to compute gradients
Automatic Differentiation

AD is a technique to evaluate exact gradients without symbolic differentiation
• in short: clever use of matrix-free techniques to evaluate Jacobians

\[ f : \mathbb{R}^n \rightarrow \mathbb{R}^m \]

\[ y = f(x) \]
\[ dy = J_f \, dx \]

\[ J_f = \frac{\partial (y_1, \ldots, y_m)}{\partial (x_1, \ldots, x_n)} \]

Most popular AD frameworks from ML usually in python:

Often challenging to make code differentiable without a rewrite
• turns out: not so much for MadGraph
MadGraph

Disclaimer: we're not MadGraph experts, but the MG tooling made it easy for us

Idea: MadGraph is code-generator for Matrix-Element Code yielding code for

\[ [0,1]^{3n-4} \rightarrow \{ \vec{p}_1, \ldots, \vec{p}_n \} \]

\[ d\sigma(x | \theta) = d\sigma(p_1, \ldots, p_n, | \theta) \]

MadGraph supports various languages, but no differentiability

Idea: add differentiable export option!
MadJax adapts the existing Python Exporter (h/t V. Hirschi) to be differentiable
• uses JAX as a AD backend
• solves edge cases that are not easily differentiable (root-finding conditionals, ...)
• JIT compilation for much faster evaluation than MG5 Python export

Final Interface is simple:

```python
import madjax
mj = madjax.MadJax('ee_to_mumu')
random_variables = [0.2]*2
matrix_element = mj.matrix_element(
  E_cm=91.0, process_name='Matrix_1_mupmup_open'
)

pars = { ('mass', 23): 9.18800e01, ('sminput', 2): 1.166390e-05 }
value,gradients = matrix_element(pars, random_variables)
print(value)
gradients
```

$> cat ee_to_mumu.mg5
genereate mu+ mu- > e+ e-
output madjax ee_to_mumu

$> mg5_aMC
--mode=madjax_me_gen
-f ee_to_mumu.mg5

0.1099516438119396

{('mass', 23): DeviceArray(-0.01722685, dtype=float64),
('sminput', 2): DeviceArray(6805.51017429, dtype=float64)}
MadJax

MadJax adapts the existing Python Exporter (h/t V. Hirschi) to be differentiable
• uses JAX as a automatic differentiation backend
• solves a number of edge cases that are not immediatly differentiable internal to wave-function and phase-space code (root-finding, conditionals etc)
• enables JIT compilation for much faster evaluation than existing Python export

Final Interface is simple:

```bash
pip install madjax

# ee_to_mumu.mg5
generate mu+ mu- > e+ e-
output madjax ee_to_mumu

mg5_aMC
--mode=madjax_me_gen
-f ee_to_mumu.mg5
```
Example Use-Case I: Automatic Likelihood-Free Inference

Mining Gold is a series of methods or "Likelihood-Free Inference" using joint quantities of the reconstructed ($x$) and truth-level event ($z$) as noisy training label.

- Methods making use of gradient data are vastly more data-efficient.

$$r(x, z, \theta_0, \theta_1) = \frac{p(x, z | \theta_0)}{p(x, z | \theta_1)}$$

$$\nabla_{\theta} \log p(x, z | \theta)$$

\[ r(x, z, \theta_0, \theta_1) = p(x, z | \theta_0) \]

\[ p(x, z | \theta_1) \]
Current use-cases exploit analytic gradients (e.g. polynomial relations)

- MadJax enables gradient-based "Mining Gold" for arbitrary theory parameters
- Toy Example: gradient-based likelihood ratio estimation w.r.t Fermi Constant

$$r(x, G_F, G_F^0) = \frac{Z \rightarrow \mu\mu | G_F}{Z \rightarrow \mu\mu | G_F^0}$$
Example Use-Case II: Proposal Distribution for MC Integration

By far most important use-case for Matrix Elements:

- **calculate total cross-section** \( \sigma(x \mid \theta) \) **and sample** \( x \sim p(x \mid \theta) = d\sigma(x \mid \theta) / \sigma(\theta) \)
- Importance Sampling: use **good proposal distribution** \( q(x \mid \theta) \sim p(x \mid \theta) \)
  to reduce variance of \( \sigma(\theta) \) (e.g. VEGAS)
- gradients of MEs can help find **good proposals** in a data-efficient way
Example Use-Case II: Proposal Distribution for MC Integration

Natural Idea: use ML to find good proposal distribution.
- ML excels solving problems approximately - just what we need

Natural Tool: Normalizing Flows (e.g. see arXiv:2001.05486)
- flexible density that is easy to evaluate & sample
- start from normal distribution and apply repeated invertible change of variables

Evaluate: \( q(x) \, dx = q_{\mathcal{N}}(f^{-1}(z)) \, J_f(x) \, dz \)  
Sample: \( x = f(z) \) with \( z \sim q_{\mathcal{N}}(z) \)
Neural Flows are trained to minimize empirical Kullback-Leibler (KL) distance

- asymmetry of KL yields two complementary approaches

Training on combination: \( L = L_{fKL} + \omega L_{rKL} \) captures mix of both behaviors

- but reverse KL needs gradients of unnormalized values of \( p(x) \)
Example Use-Case II: Proposal Distribution for MC Integration

Implementing "Smooth Normalizing Flows" from arXiv:2110.00351
• method for neural flows on bounded manifolds
• trains on mixture of KL distances
• data-efficient: flow samples for rKL are cheap!

14-D Example: $qq \rightarrow tt \rightarrow bbqqqq$

![Graph showing improvement in flow description of ME]
Summary & Outlook

Machine Learning renewed focus on algorithmic differentiation of scientific code.

Matrix Element Code (with flexible code generation) a prime target in HEP
- Gradients (both wrt. theory and phase-space) useful for many applications with and without ML
- MadJax provides differentiable Matrix-Elements by using autodiff with JAX
- enables a number of previously inaccessible use-cases

If you have use for ME gradients, please let us know!
- always happy to collaborate