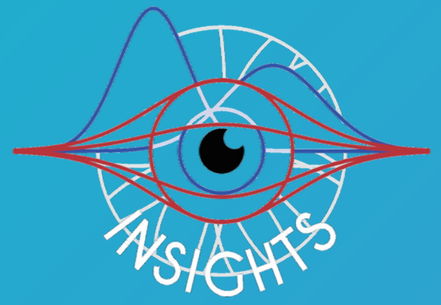
DIFFERENTIABLE ANALYSIS

Lukas Heinrich, **Nathan Simpson** <-- me



We optimise analysis* end-to-end by making the profile likelihood differentiable.

*currently just a toy problem

Typical analysis workflow:

data

Pre-processing
(reconstruction,
skimming, cuts)

observables

Invariant mass of
some system
Multivariate
discriminator

model

HistFactory likelihood
Some other
parametric fit
Data-driven likelihood

inference

CLs
Feldman-Cousins
Posterior sampling
Credible intervals
...etc

Typical analysis workflow:

data

Pre-processing
(reconstruction,
skimming, cuts)

observables

Something with
trainable
parameters φ

model

HistFactory likelihood
Some other
parametric fit
Data-driven likelihood

inference

CLs
Feldman-Cousins
Posterior sampling
Credible intervals
...etc

Train hard, play harder

Standard training goal for a signal/background discriminator: **binary cross entropy**

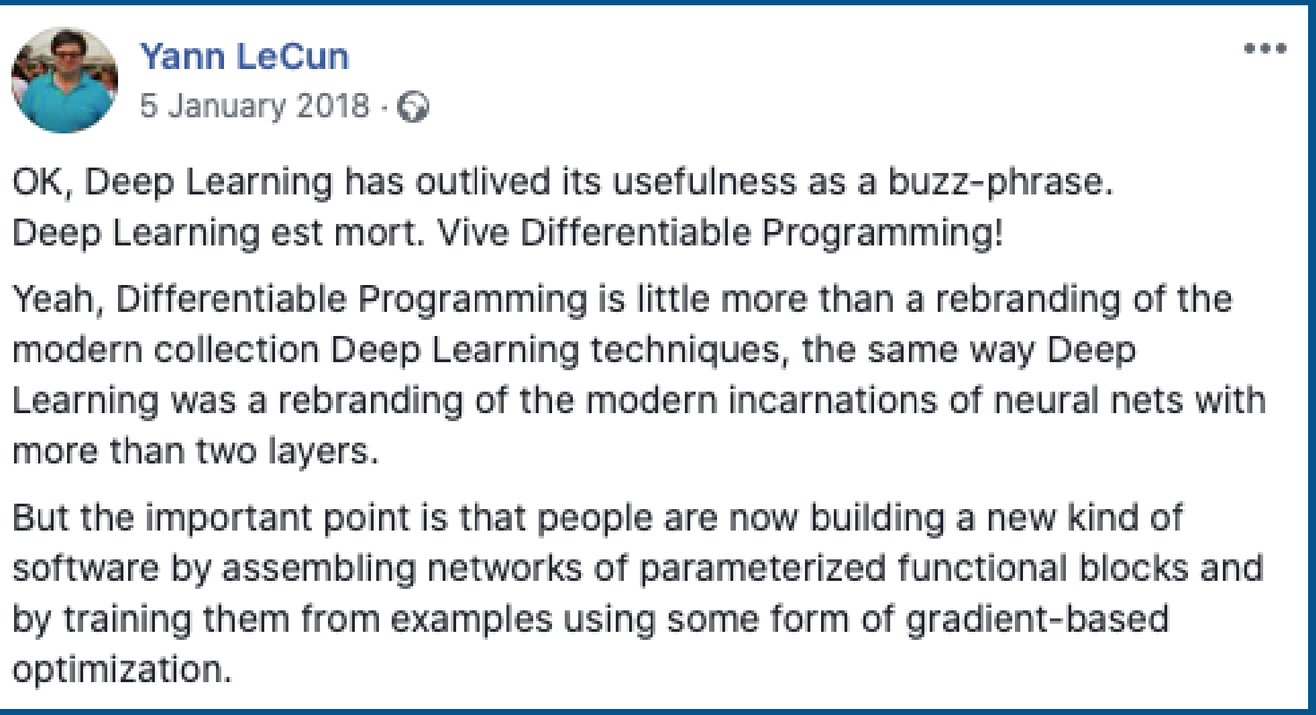
Makes sense to have good discriminative power,
but can we do better?

- *NN isn't aware of systematics!*
- *Could do adversarial training, but hard to scale to many uncertainties*

What we really want is to optimise our observables with respect to all steps in our downstream inference.

Motivation: see dguest et al's review: "Deep Learning and Its Application to LHC Physics",
specifically this section:
3.1. What Is the Optimization Objective?

Jargon alert: differentiable programming



Yann LeCun
5 January 2018 ·

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

Yeah, Differentiable Programming is little more than a rebranding of the modern collection Deep Learning techniques, the same way Deep Learning was a rebranding of the modern incarnations of neural nets with more than two layers.

But the important point is that people are now building a new kind of software by assembling networks of parameterized functional blocks and by training them from examples using some form of gradient-based optimization.

Optimization by gradstudentient descent

observables +gradients

Something with trainable parameters φ

model +gradients

HistFactory likelihood

Some other parametric fit

Data-driven likelihood

inference +gradients

CLs

Feldman-Cousins

Posterior sampling

Credible intervals

...etc

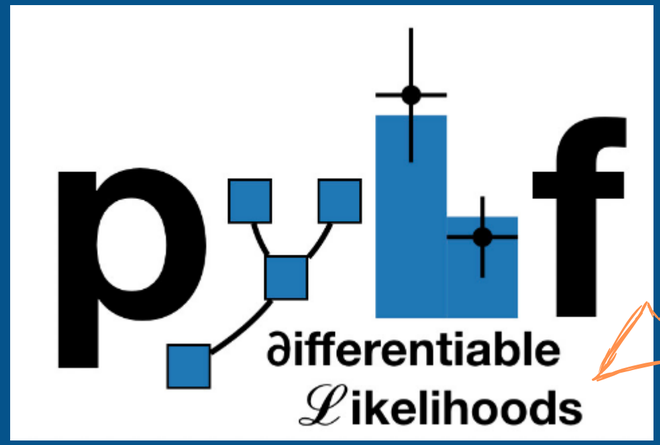
$$\frac{d(\text{inference})}{d(\varphi)}$$

Starting point: pyhf

Pure python implementation of HistFactory

Model step is already differentiable!

Uses multiple ML libraries as backends, including pytorch, tensor flow, and recently...



already has differentiable likelihoods!

A new foe has appeared!

backend

JAX: numpy, but gradients

Check out the project at github.com/google/jax!

JAX is a modern autograd library that can natively differentiate pure python and numpy functions up to arbitrary order.

e.g:

```
import jax

x = jax.numpy.sin
jax.grad(x)(0.) # could also do grad(grad(...))

-> DeviceArray(1., dtype=float32)
```

It can also do other cool stuff, like auto-vectorisation with *jax.vmap*, and just-in-time compilation of 'pure' functions with *jax.jit*!

so... are we done?

observables

something with trainable parameters φ in jax

inference

CLs calculations in jax

...not quite

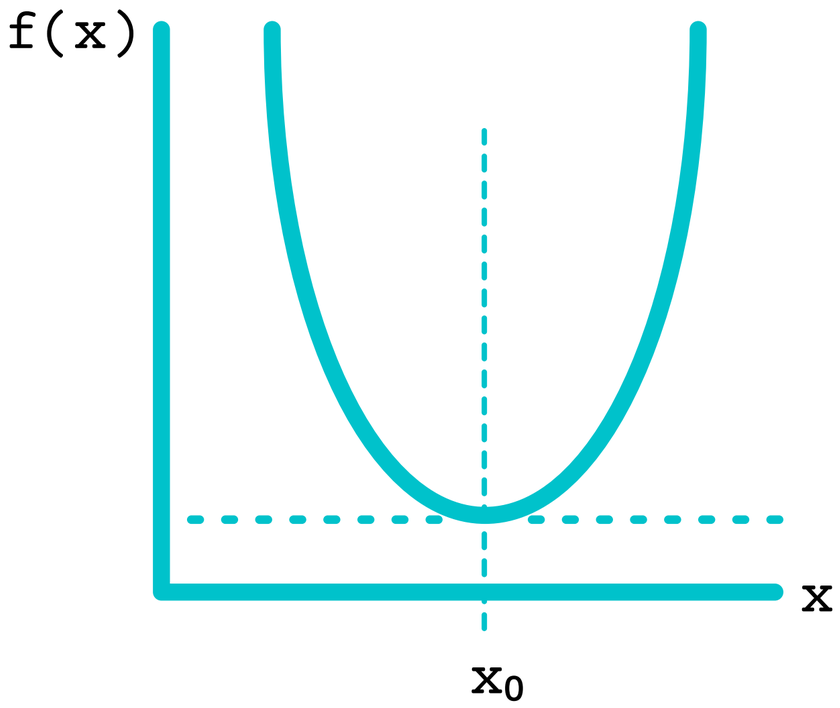
CLs values use the profile likelihood:

$$\lambda(\mu) = \frac{\mathcal{L}(\mu, \hat{\hat{\theta}})}{\mathcal{L}(\hat{\mu}, \hat{\theta})}$$

Both numerator and denominator involve maximum likelihood fits!
Could track gradients in every unrolled iteration of the fits, but hard to do.

Solution?

Fixed-point differentiation



A fixed point x of a function is where
`func(x) = x`

`minimize(f, x_init) -> x₀`
`minimize(f, x_init = x₀) -> x₀`

The basic idea of fixed point differentiation is to only calculate the gradients in the neighbourhood of the fixed point.

Gradients with respect to what?
*f is the model, which depends on the yields s,b (which depend on **φ**)*


"FAX is the magic sauce that allows us to implicitly differentiate"
Lukas Heinrich

fax: fixed-point jax

Implicit and competitive differentiation in JAX.

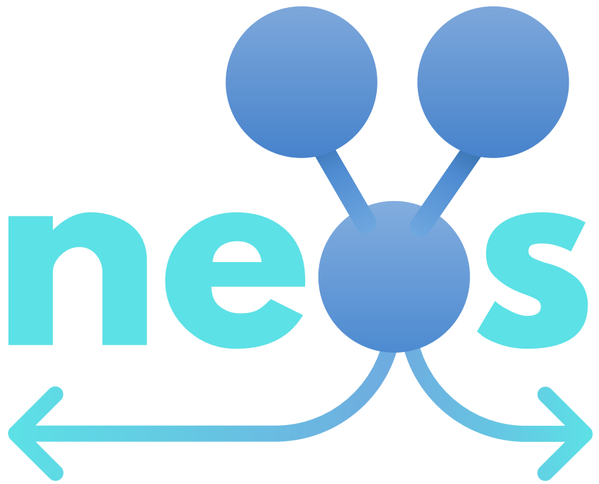
Our "competitive differentiation" approach uses Competitive Gradient Descent to solve the equality-constrained nonlinear program associated with the fixed-point problem. A standalone implementation of CGD is provided under fax/competitive/cga.py and the equality-constrained solver derived from it can be accessed via `fax.constrained.cga_lagrange_min` or `fax.constrained.cga_ecp`. An implementation of implicit differentiation based on Christianson's two-phases reverse accumulation algorithm can also be obtained with the function `fax.implicit.two_phase_solver`.

Function we use: Two-phase solver <-- paper linked for the brave

```
@misc{gehring2019fax,
  author = {Clement Gehring, Pierre-Luc Bacon, Florian Schaefer},
  title = {{FAX: differentiating fixed point problems in JAX}},
  note = {Available at: https://github.com/gehring/fax},
  year = {2019}
}
```

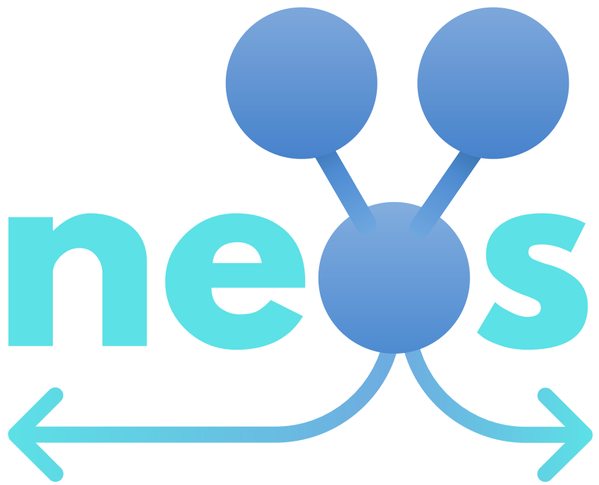
Huge thanks to Clement Gehring!

All at once, everything is differentiable



neural end-to-end optimised statistics

github.com/pyhf/neos



nice ~~neural~~ end-to-end optimised statistics

github.com/pyhf/neos

Problem setup:

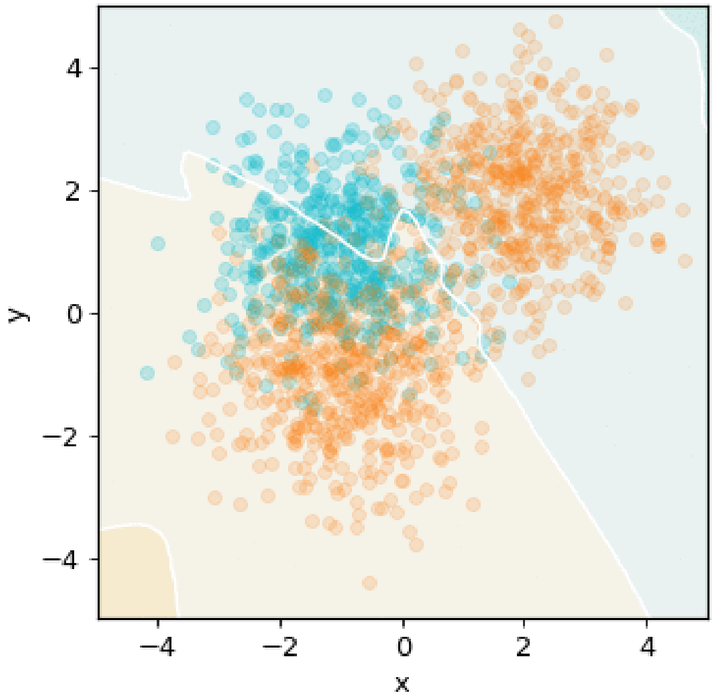
- Signal data drawn from a 2D multivariate gaussian.
- Background data has two modes -- 2D gaussians with different means and covariances.
- Histogram constructed 'continuously', i.e. NN(φ) outputs two logits (a,b) for an event, then the event is put into all bins, weighted by the logits.
- Background yields are the mean of the bin counts for each mode, and the per-bin uncertainty is the difference/2.
- Model constructed from yields for signal and background events,
- Global and constrained fits are wrapped with the two_phase_solver method to get gradients wrt yields (which depend on φ)
- Calculate CLs, then backprop with grad(CLs)(φ)

Oh, and I lied about the pyhf. (we use a dummy version with minimal functionality)

neos : As easy as 1, 3, 2

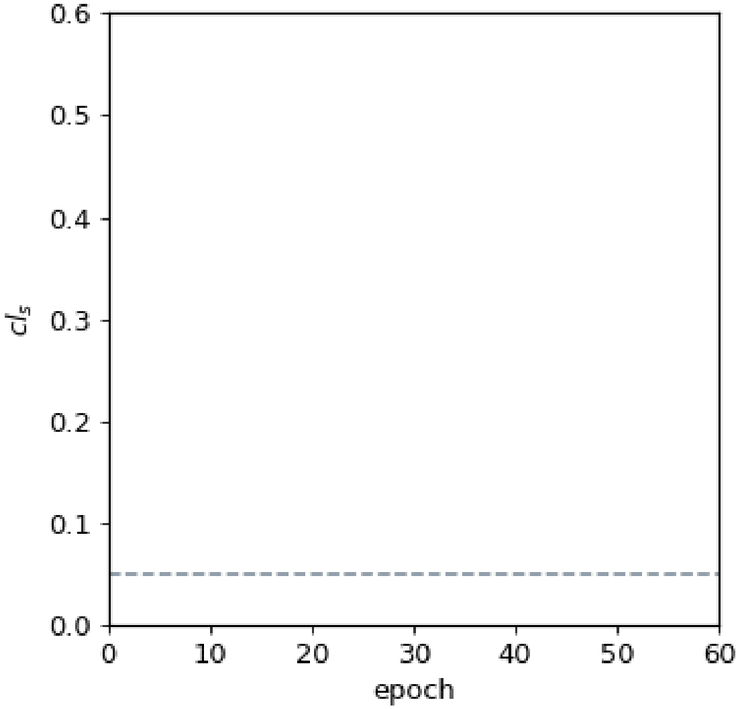
1)

Predict logits for each event



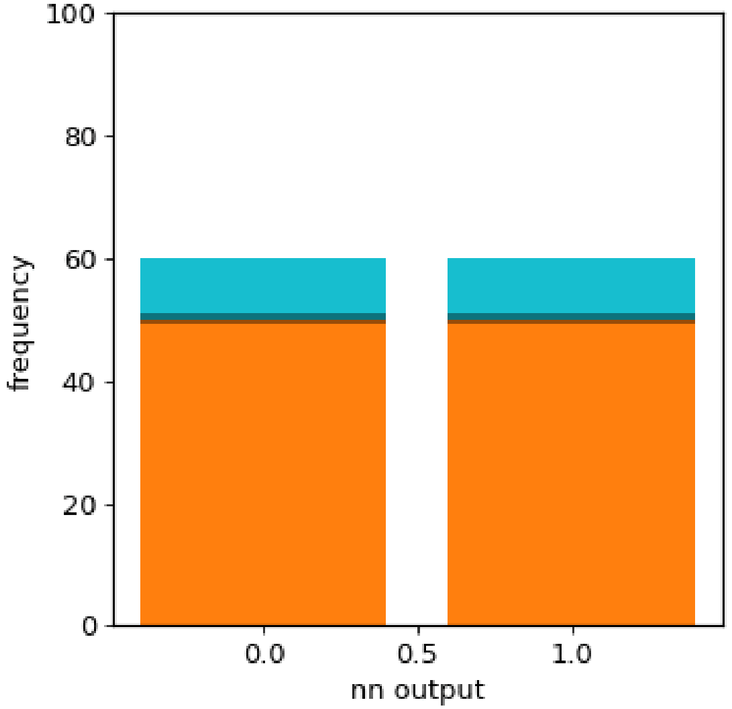
3)

Calculate CLs differentiably, and backpropagate wrt NN weights



2)

Bin the result weighted by the logits, and construct model



Future work:

Go beyond two bins (not too tricky)

Don't lie about pyhf :)

Incorporate more familiar systematics, e.g. 3 blobs for nominal, up, downward variations

https://knowyourmeme.com/memes/future-sending-exes-texts

thanks for listening :)

and nice to meet you all!

Twitter: @phi_nate @lukasheinrich_