A Tutorial on Differentiable Analysis & end-to-end learning

Nathan Simpson

PyHEP, 15/09/22

Lund University

ATLAS EXPERIMENT

Two software libraries:

relaxed

built with

neos

A suite of **differentiable operations** designed to target typical HEP use cases.

A method for **optimizing observables in an end-to-end way**, incorporating systematics

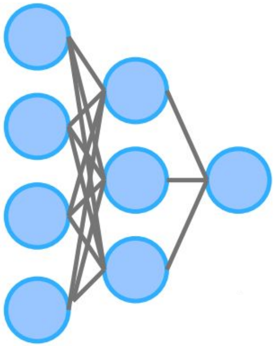
Also: a software package that implements helper functions for this use case

https://github.com/gradhep/relaxed

https://github.com/gradhep/neos

https://github.com/phinate/differentiable-analysis-examples

Tangent:
how do neural networks learn at all?

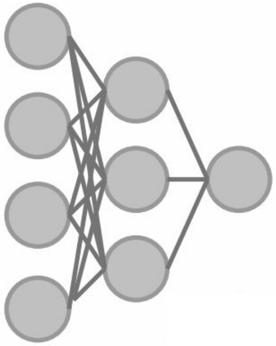


=

φ

“parameters”

+



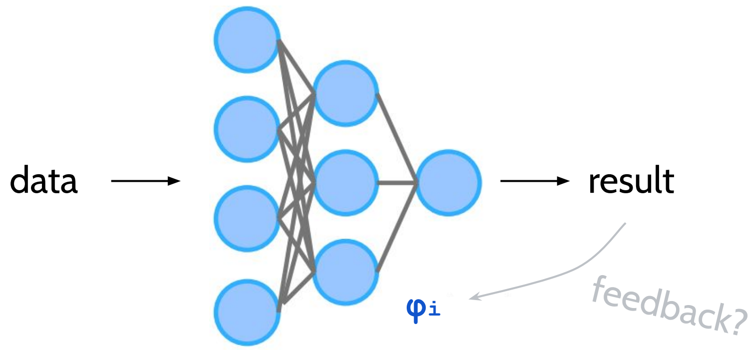
“architecture”

(how you combine parameters w/ data)

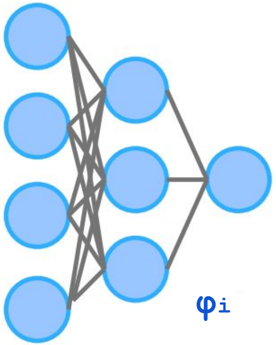
data

activation(weight*data + bias)

learnable free parameters


data
result
φi
feedback?

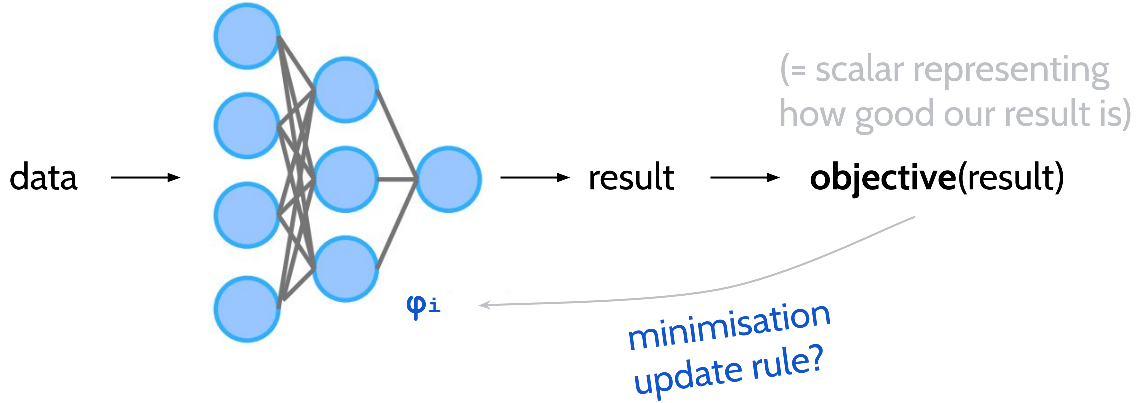
data →



→ result → **objective**(result)

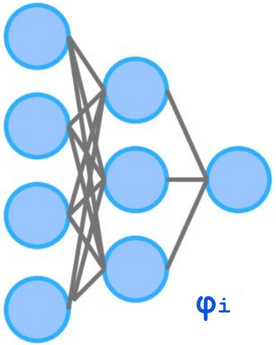
(= scalar representing how good our result is)

want to **minimise**



data
result
objective(result)
(= scalar representing
how good our result is)
φi
minimisation
update rule?

(= scalar representing how good our result is)

data → 

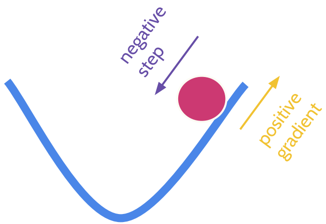
→ result → **objective**(result)

$$\underbrace{\text{data} \rightarrow \dots \rightarrow \text{result} \rightarrow \textbf{objective}(\text{result})}_{= \, \texttt{workflow}(\texttt{data}, \, \varphi_i)}$$

update rule: **gradient descent**

$$\varphi_{i+1} = \varphi_i - \texttt{lr} * \underbrace{d(\texttt{workflow}(\texttt{data}, \, \varphi_i))/d\varphi_i}_{\text{gradient of workflow w.r.t. current parameters}}$$

step size
(“learning rate”)



trying to roll downhill in param space!

We don’t need neural networks to do this!

(but they are often quite useful, so you’ll see some more later)

Same thing with a straight line:

e.g. for 2D data:
data on left of line = signal,
on right = background

Hard to say where "model" ends and "objective" begins.

data ⟶ $y = mx + c$ ⟶ result ⟶ **objective**(result)

$\varphi_i = \{m, c\}$

$= \texttt{workflow(data, } \varphi_i\texttt{)}$

still works!

$$\varphi_{i+1} = \varphi_i - \texttt{lr} * \underbrace{\texttt{d(workflow(data, } \varphi_i\texttt{))/d}\varphi_i}$$

as long as we can
calculate this gradient!

Same thing with a straight line:

e.g. for 2D data:
data on left of line = signal,
on right = background

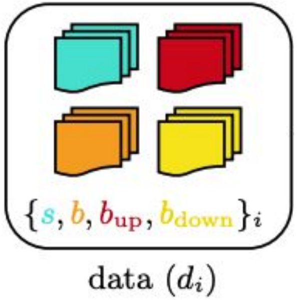
Idea:

Using *gradient descent*, we can optimise *any* workflow parameters with respect to *any* goal... <u>*if* the full workflow is differentiable.</u>

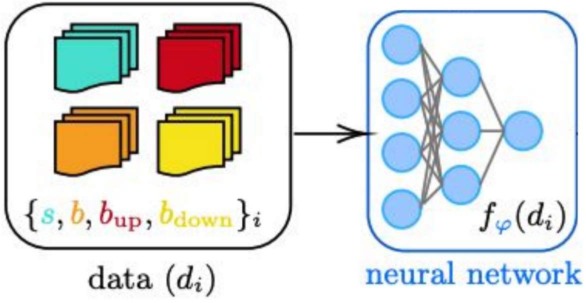
still works!

as long as we can calculate this gradient!

A typical HEP analysis workflow

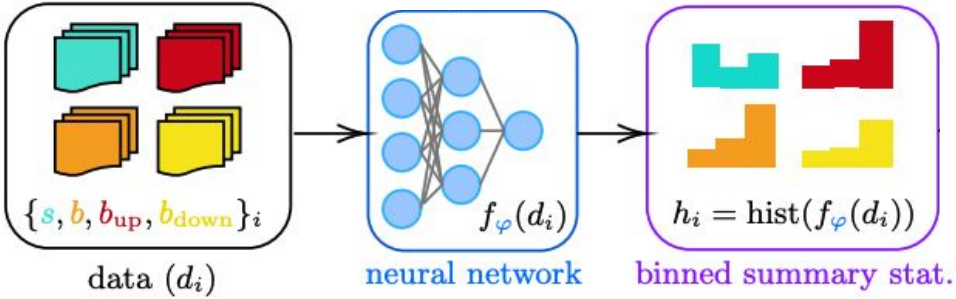


A typical HEP analysis workflow

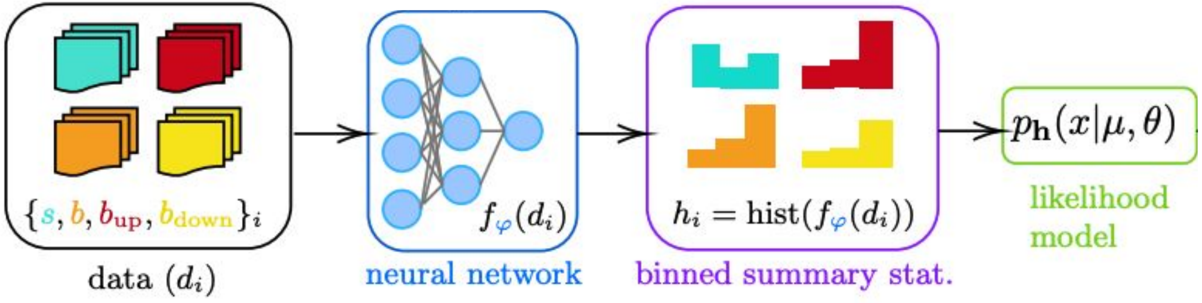


More abstractly: step with free parameters (e.g. event selection)

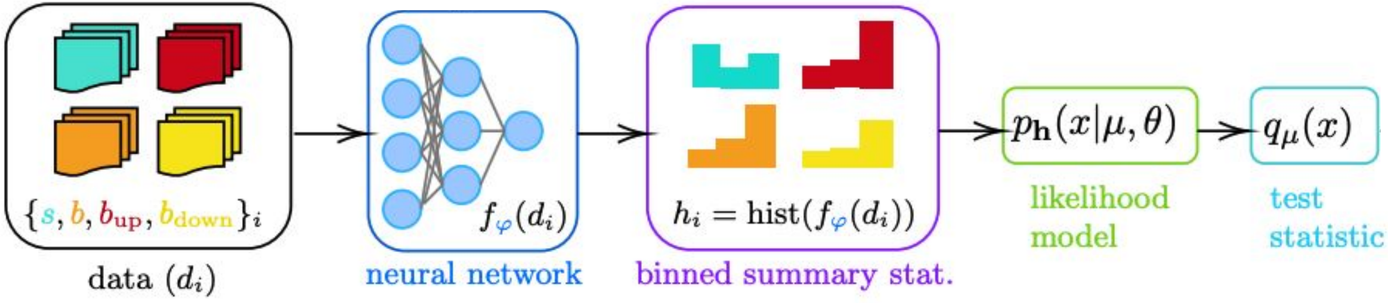
A typical HEP analysis workflow



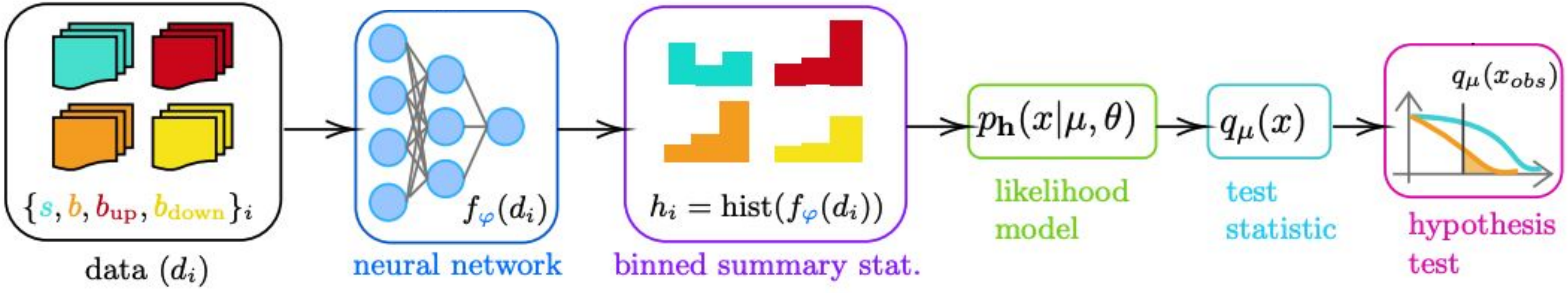
A typical HEP analysis workflow



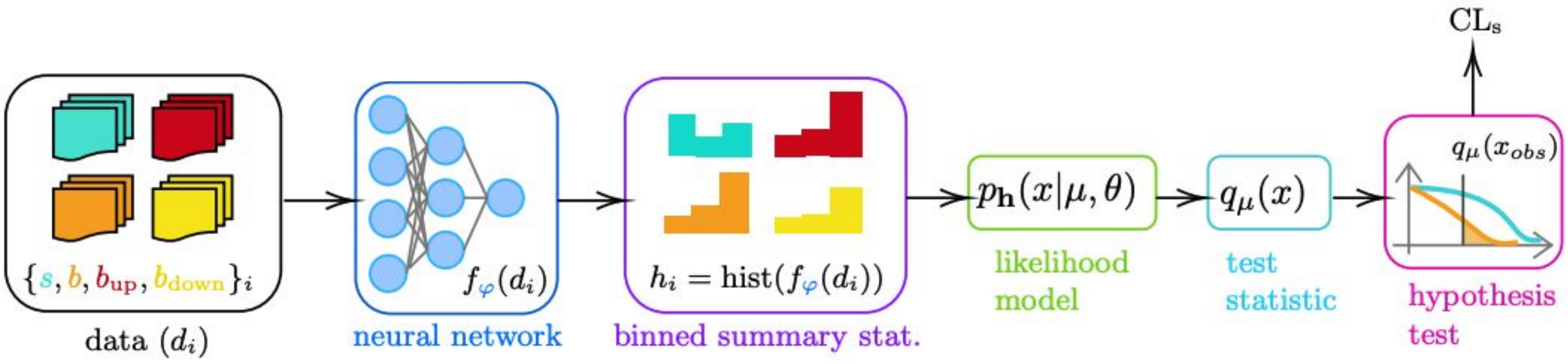
A typical HEP analysis workflow



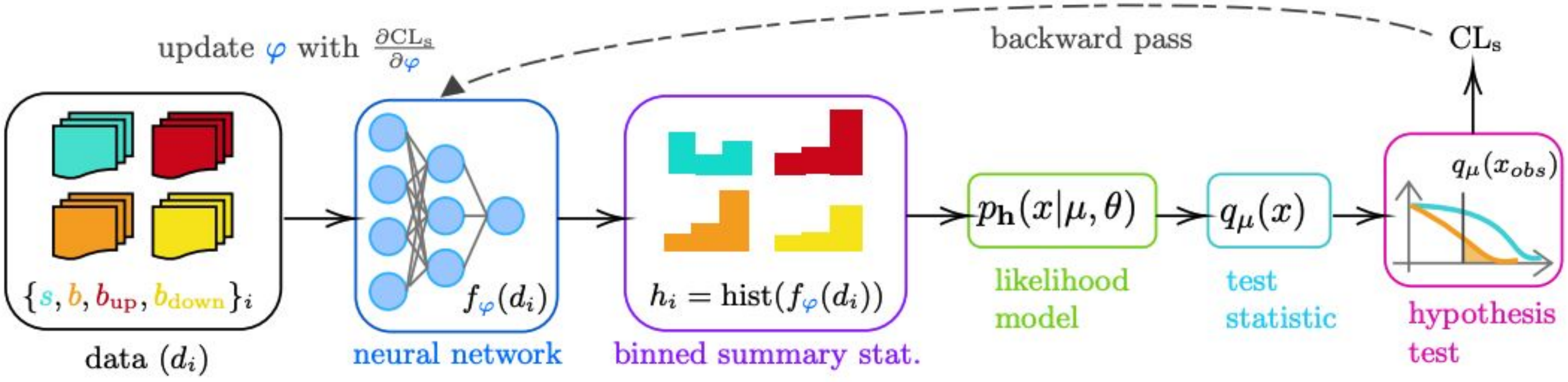
A typical HEP analysis workflow



A typical HEP analysis workflow



A typical HEP analysis workflow



In equation form:

$$\frac{\partial\,\text{significance}}{\partial\,\varphi} = \frac{\partial\,\text{significance}}{\partial\,\text{profile likelihood}} \times \frac{\partial\,\text{profile likelihood}}{\partial\,\text{modelling}} \times \frac{\partial\,\text{modelling}}{\partial\,\text{histogram}} \times \frac{\partial\,\text{histogram}}{\partial\,\varphi}$$

(chain rule)

but wait, this is all `code` right?
how do we differentiate a computer program?

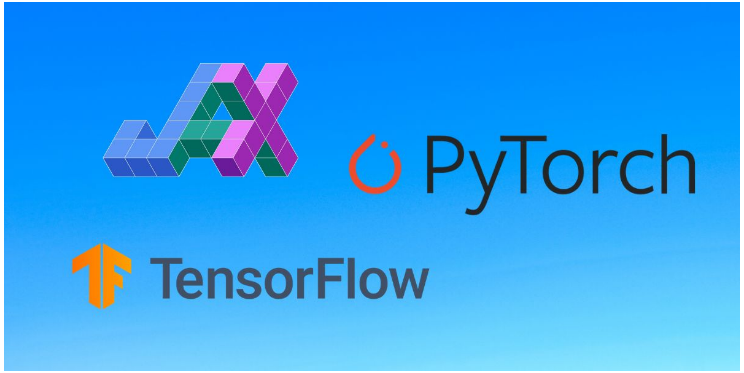
How might we get those gradients?

Automatic differentiation!

Quick explanation:

- Any program can be broken down into a series of *primitive* operations (+, -, /, *, log, exp...)
- These have **known derivatives!**
- Can then compose these derivatives via the **product rule** to get the gradient of the whole program!

→ exact, efficient gradients



Thanks to deep learning's prominence, we have many great software libraries [JAX, PyTorch, TensorFlow] that take advantage of hardware acceleration (GPUs, TPUs)

Img source: AskPython.com

So is that it?

We code up our analysis in PyTorch and fit the model?

...not quite :)

Not all operations can be broken into differentiable primitives, because not all operations are differentiable!

Need to figure out a way to "relax" some operations to allow us to take their gradients.

Pictured: One very discrete boi.

Every step of the workflow needs to be differentiable!

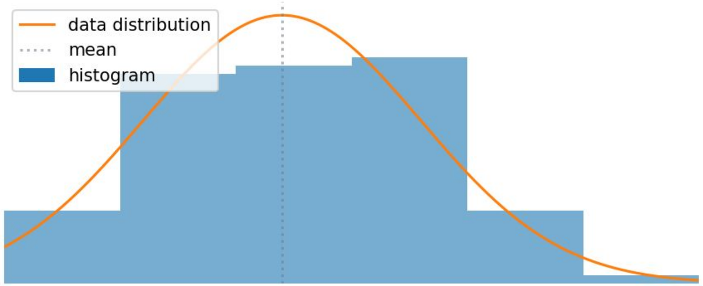
$$\frac{\partial\,\text{significance}}{\partial\,\varphi} = \overbrace{\frac{\partial\,\text{significance}}{\partial\,\text{profile likelihood}}}^{\text{already differentiable}} \times \underbrace{\frac{\partial\,\text{profile likelihood}}{\partial\,\text{modelling}} \times \frac{\partial\,\text{modelling}}{\partial\,\text{histogram}} \times \frac{\partial\,\text{histogram}}{\partial\,\varphi}}_{\text{not necessarily differentiable a priori}}$$

=> Let’s change that!

*See Kyle Cranmer's (heavily cited) paper on this: arxiv.org/abs/hep-ex/0011057

In one slide: Making <u>analysis</u> differentiable

Example: histograms [very discrete!]



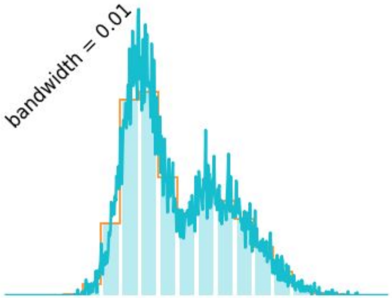
*See Kyle Cranmer's (heavily cited) paper on this: arxiv.org/abs/hep-ex/0011057

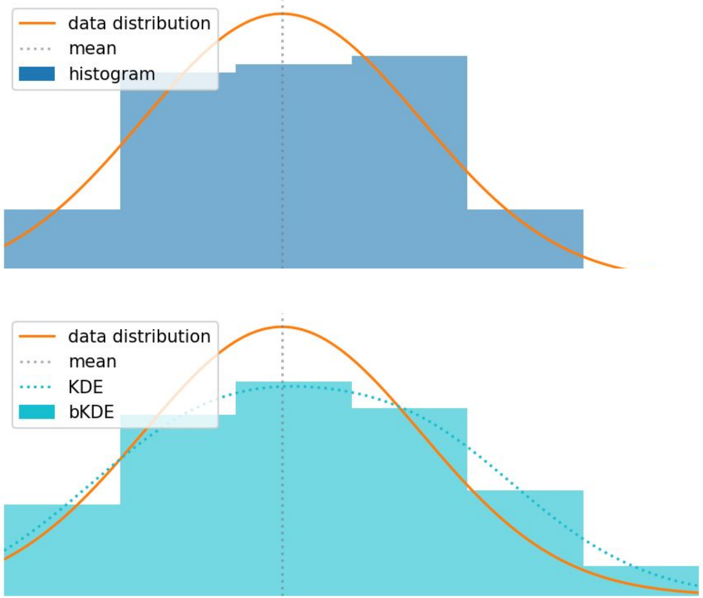
In one slide: Making analysis differentiable

Example: histograms [very discrete!]

We developed a histogram-alternative using **kernel density estimates** (KDEs). [used already in HEP!]*

Integrating the KDE over a set of intervals gives the notion of "bins". => Binned KDE (**bKDE**)





*See Kyle Cranmer's (heavily cited) paper on this: arxiv.org/abs/hep-ex/0011057

In one slide: Making analysis differentiable

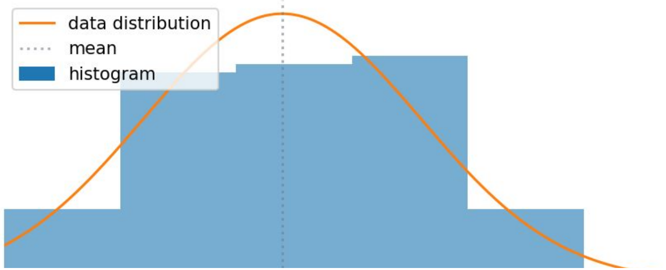
Example: histograms [very discrete!]

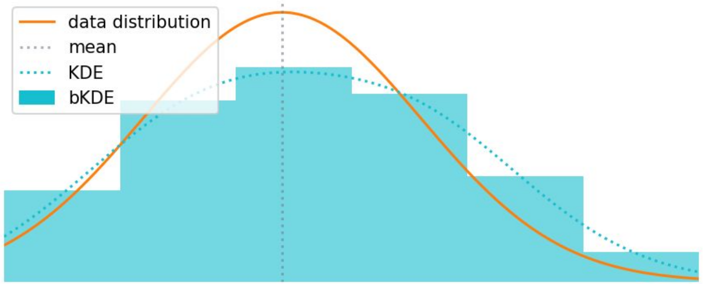
We developed a histogram-alternative using **kernel density estimates** (KDEs). [used already in HEP!]*

Integrating the KDE over a set of intervals gives the notion of "bins". => Binned KDE (**bKDE**)

Also have:

- **differentiable cuts** (sigmoid)
- **differentiable likelihood-building** through `pyhf`
- **differentiable fitting** due to exploiting the implicit function theorem



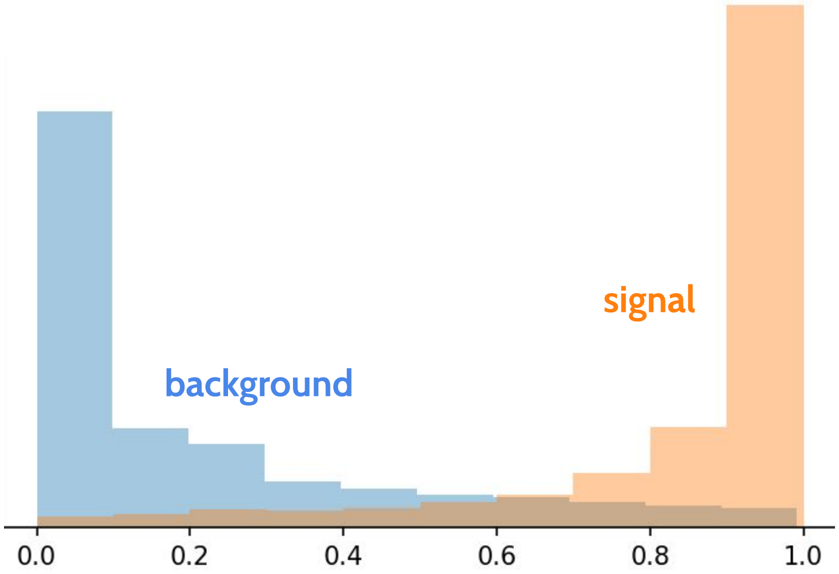


Now time for some code!

What makes a good observable?

Searches for new physics endeavour to maximally discriminate simulated signal data from background processes.

But is this *really* what we want?



e.g. neural network w/ 1-D output, trained to minimize binary cross-entropy

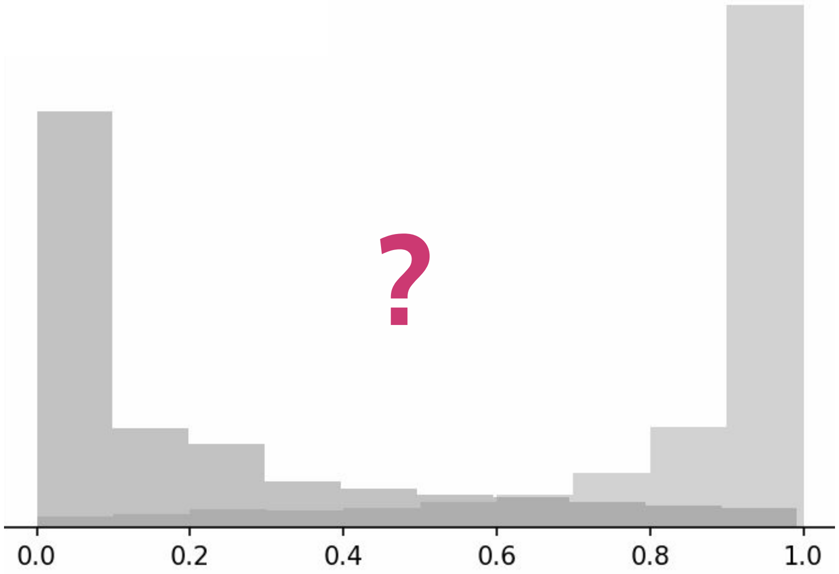
What makes a good observable?

Searches for new physics endeavour to maximally discriminate simulated signal data from background processes.

But is this *really* what we want?

e.g. what happens when we include **systematic variations** of the signal/background?

- Not guaranteed to produce a sensitive observable for all templates!
- Observable knows nothing about how we model + profile over the uncertainty!



> “(...) sensitivity to high-level physics questions *must account for systematic uncertainties*, which involve a nonlinear trade-off between the typical machine learning performance metrics and the systematic uncertainty estimates. ”

Deep Learning and its applications to LHC Physics, section 3.1,
D.Guest, K.Cranmer, D.Whiteson, 2018
arxiv.org/abs/1806.11484

(emphasis not in original text)

Can we <u>learn</u> to incorporate systematics?

Same thing with a straight line:

e.g. for 2D data:
data on left of line = signal,
on right = background

Idea 2:

We can **<u>directly optimise</u> the discovery significance/CLs** of our analysis this way!
-> <u>Systematic aware</u> [profiling]

still works!

as long as we can
calculate this gradient!

Oh baby it’s code time!

This work was partially supported by the Insights ITN, funded by the European Union's Horizon 2020 research and innovation programme, call H2020-MSCA-ITN-2017, under Grant Agreement n. 765710.

Work now supported by the Swedish Science Council and Lund University directly.

(until November)

That's it!

If you want to:

> discuss *more about this* in any way
> have an *interesting use case*
> talk about *future opportunities*
> send me *pet images*

please reach out! email: **n.s@cern.ch**
I'd love to hear from you :)

and thanks for listening!

one of my cats, enjoying the homely comfort of the washing machine

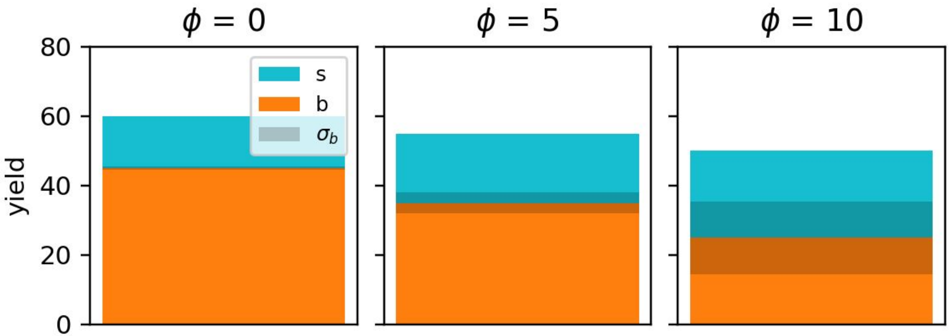
Seeing it in practice

Taken from my tutorial repo:

github.com/gradhep/differentiable-analysis-examples/

Toy example: 1-bin counting experiment

$$s = 15 + \varphi$$
$$b = 45 - 2\varphi$$
$$\sigma_b = 1 + (\varphi/5)^2$$

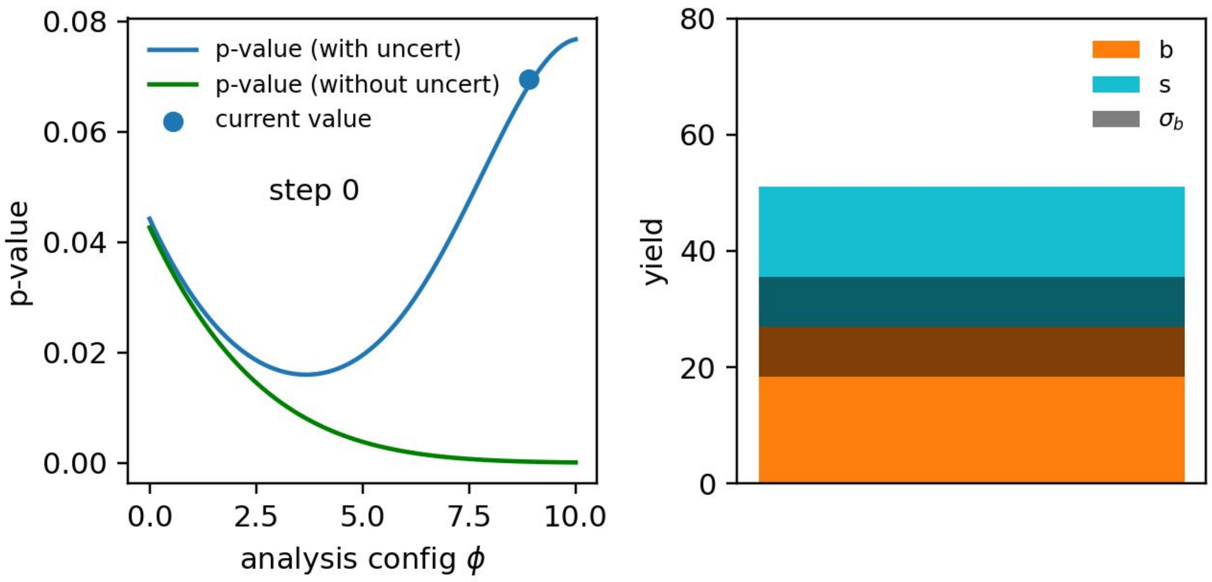


Increasing φ

Increasing s/b ratio

Increasing uncertainty on background

Learning to discover: 1-bin example

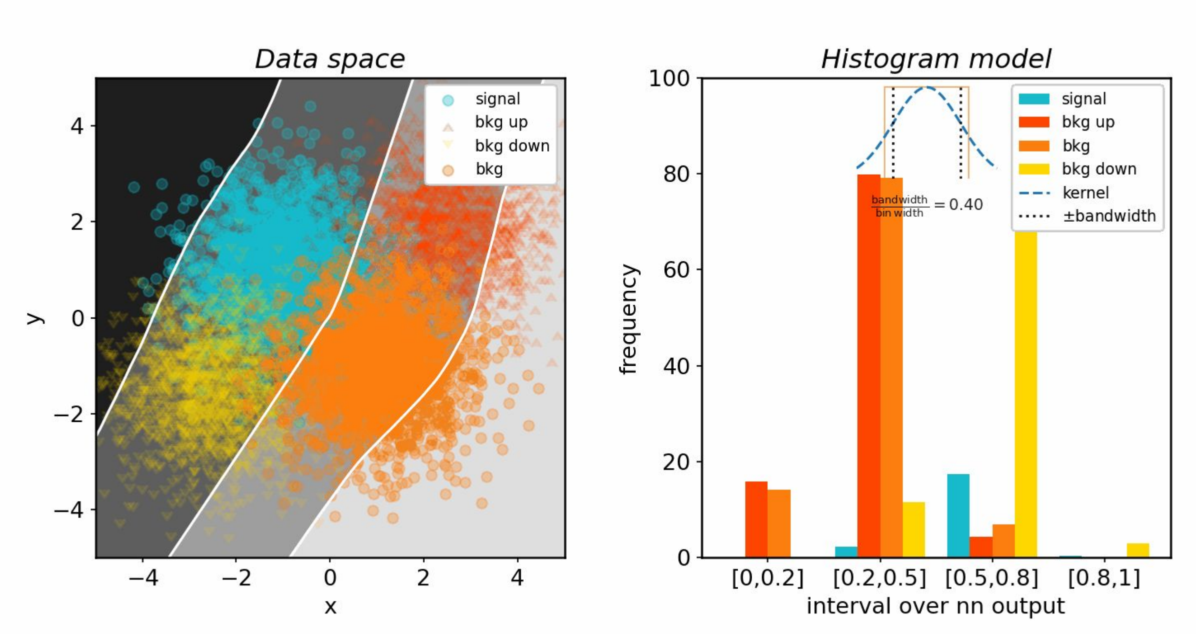


We're able to recover the optimal significance in our toy problem!

Intuitively, we're trading off uncertainty and s/b ratio in order to give the best result.

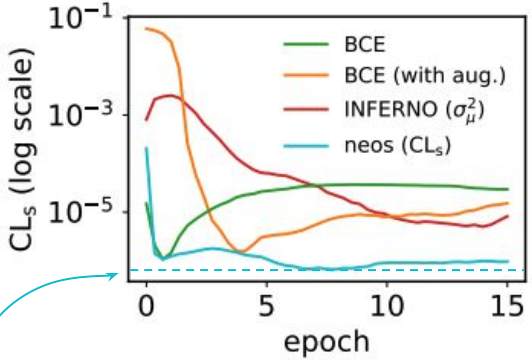
for pdf viewers: optimisation with respect to significance is able to find the optimal significance accounting for uncertainty (minimum of blue curve)

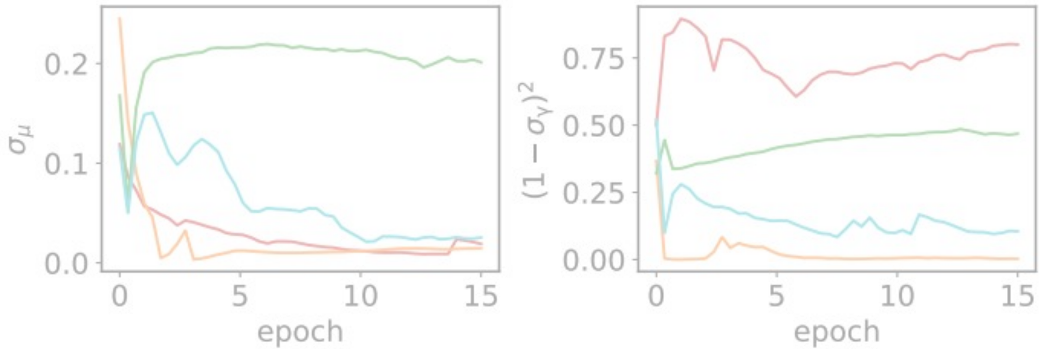
Optimising a neural network observable (neos)



for pdf viewers: neural network contours wrap around the signal blob, but also balance the background variations to minimise uncertainty.

Optimising a neural network observable (neos)





neos gets **better CLs** than all other tried methods!

additional plots that show:

> cross-section uncertainty is also optimised for free

> no over/underconstraint of nuisance parameter

Optimising a neural network observable (neos)

More fun details and context in our preprint! :)

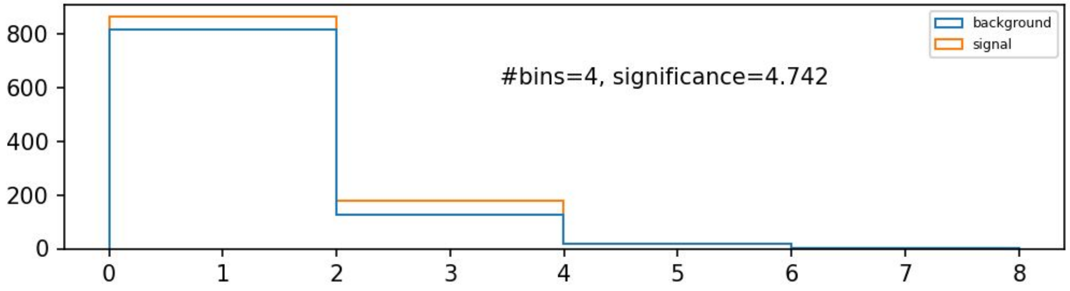
In collaboration with Lukas Heinrich: arxiv.org/abs/2203.05570

code:

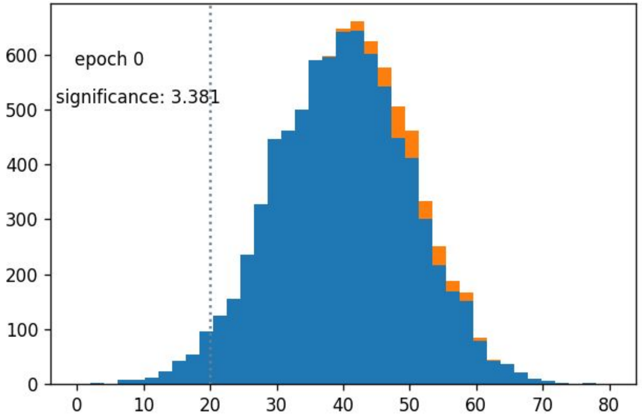
github.com/gradhep/neos

You can optimize anything!

binning!



cuts!



relaxed

https://github.com/gradhep/relaxed

Backup

You want to know how it scales!

Me too!

IRIS-HEP is very interested in this, and plans to support it for the “Analysis grand challenge” on open data, but may need more personpower.

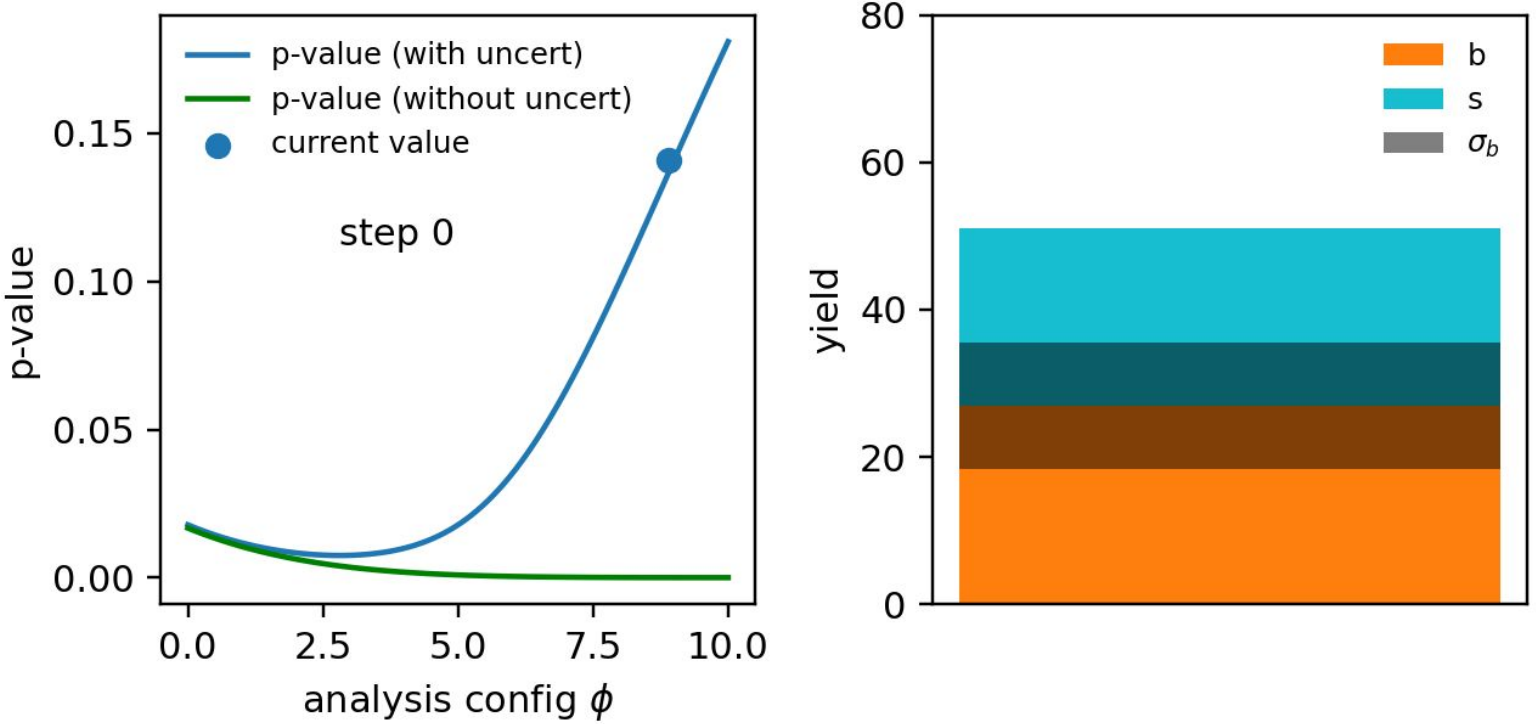
Very much open to collaboration on any use case!

example concerns:

> batch size may need to sufficiently represent analysis (so could require lots more VRAM compared to usual approach)

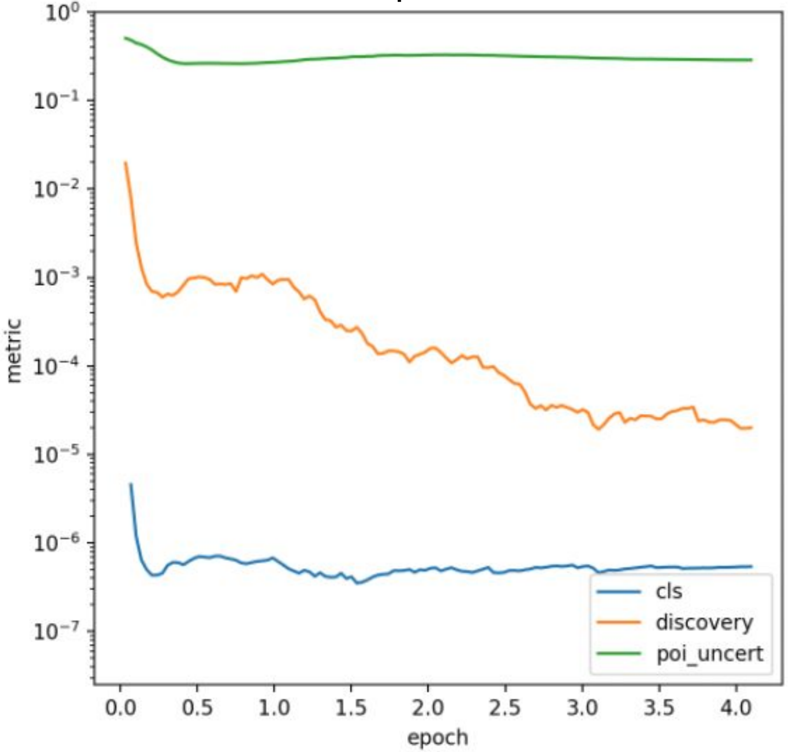
> every minibatch update = one run of the analysis, so may need lots more compute (but GPUs + autodiff are very powerful!)

Discovery significance (it still works!)

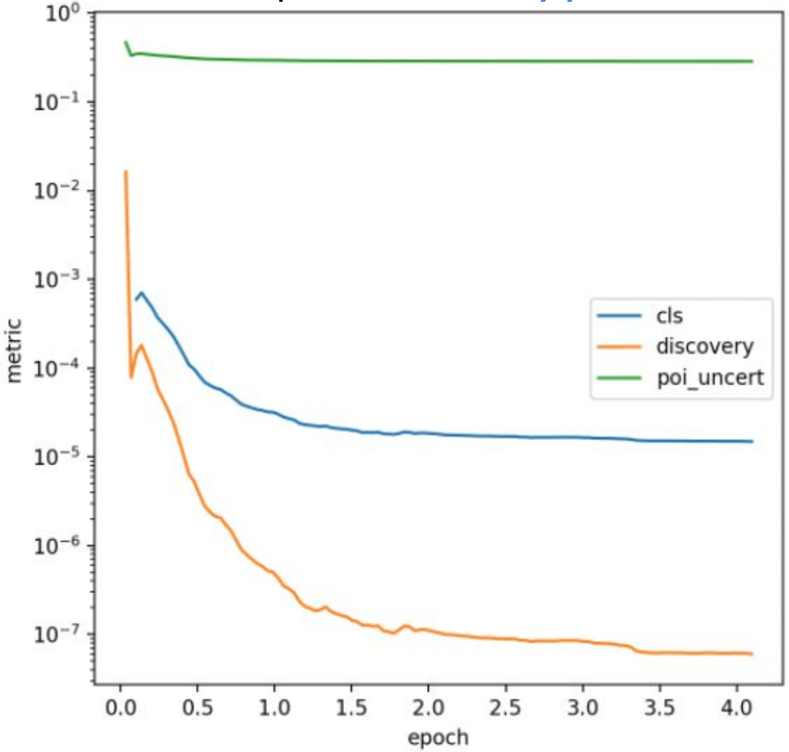


Differences between discovery p-value and CLs

Train to optimize **CLs**

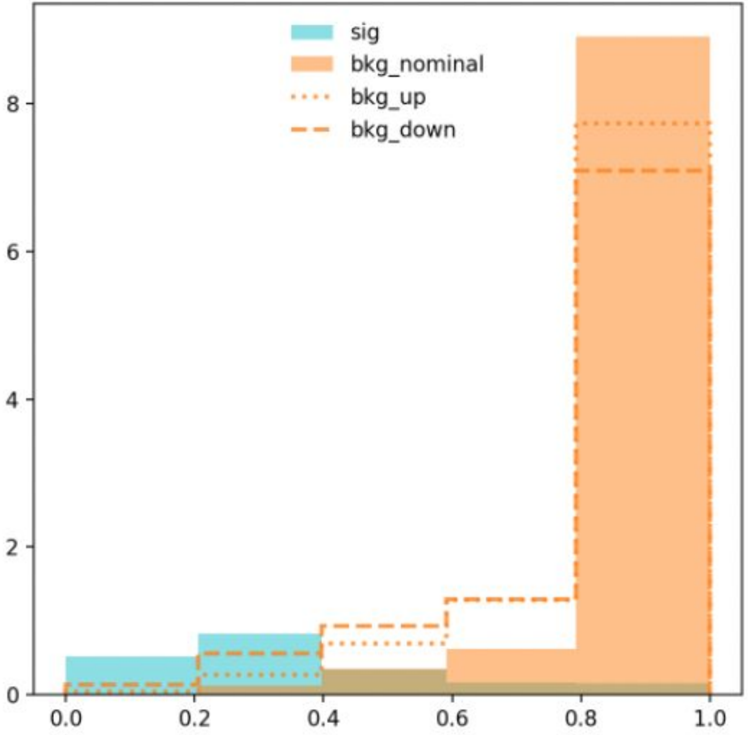


Train to optimize **discovery p-value**

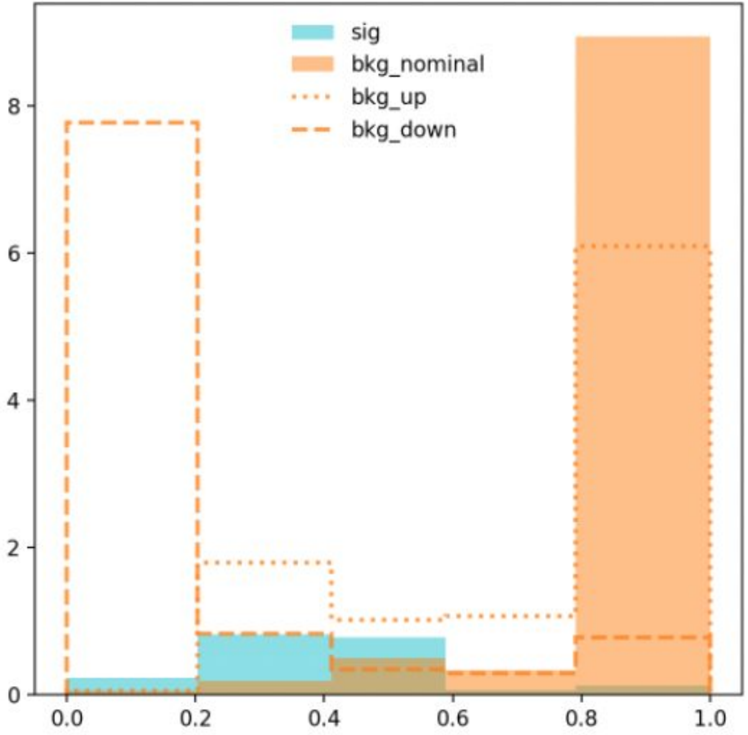


Differences between discovery p-value and CLs

Train to optimize **CLs**

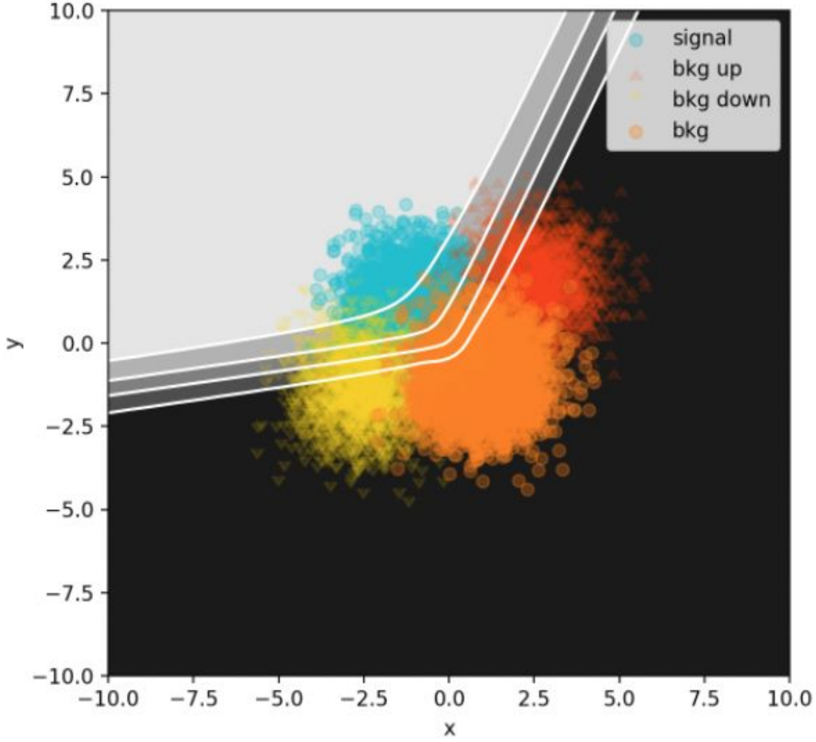


Train to optimize **discovery p-value**

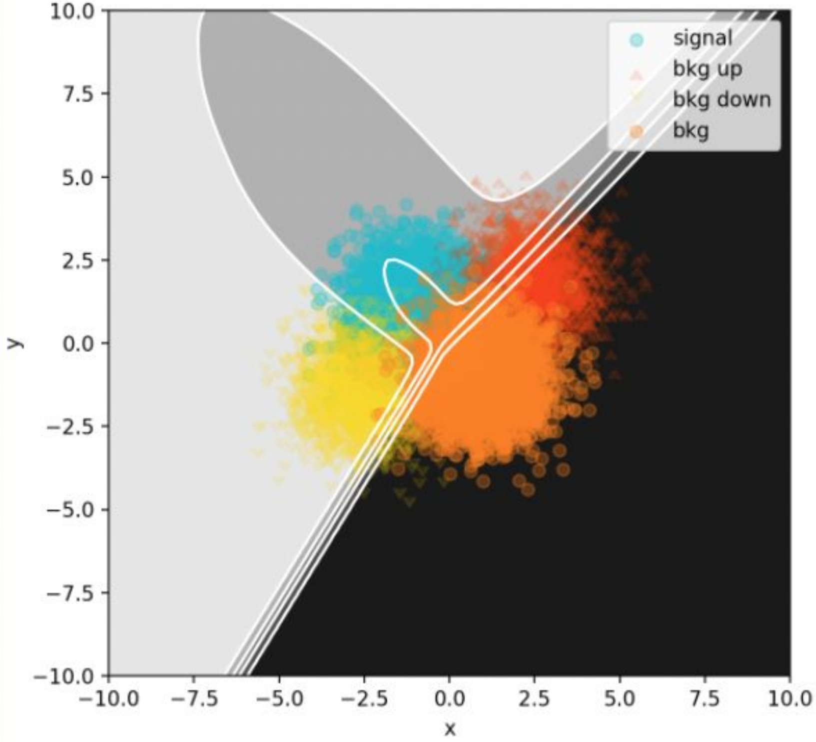


Differences between discovery p-value and CLs

Train to optimize **CLs**



Train to optimize **discovery p-value**



Which bandwidth to pick?