Differentiable Programming

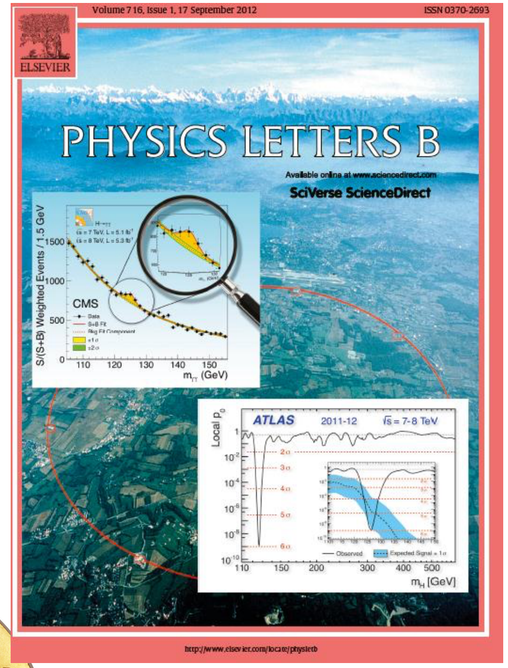
A Tutorial - IML Workshop 2022

Lukas Heinrich, TUM

Introduction

The reason why we have our IML workshops is in large part due to the success of Deep Learning in the last decade

2012:



ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky University of Toronto kriz@cs.utoronto.ca
Ilya Sutskever University of Toronto ilya@cs.utoronto.ca
Geoffrey E. Hinton University of Toronto hinton@cs.utoronto.ca

Abstract

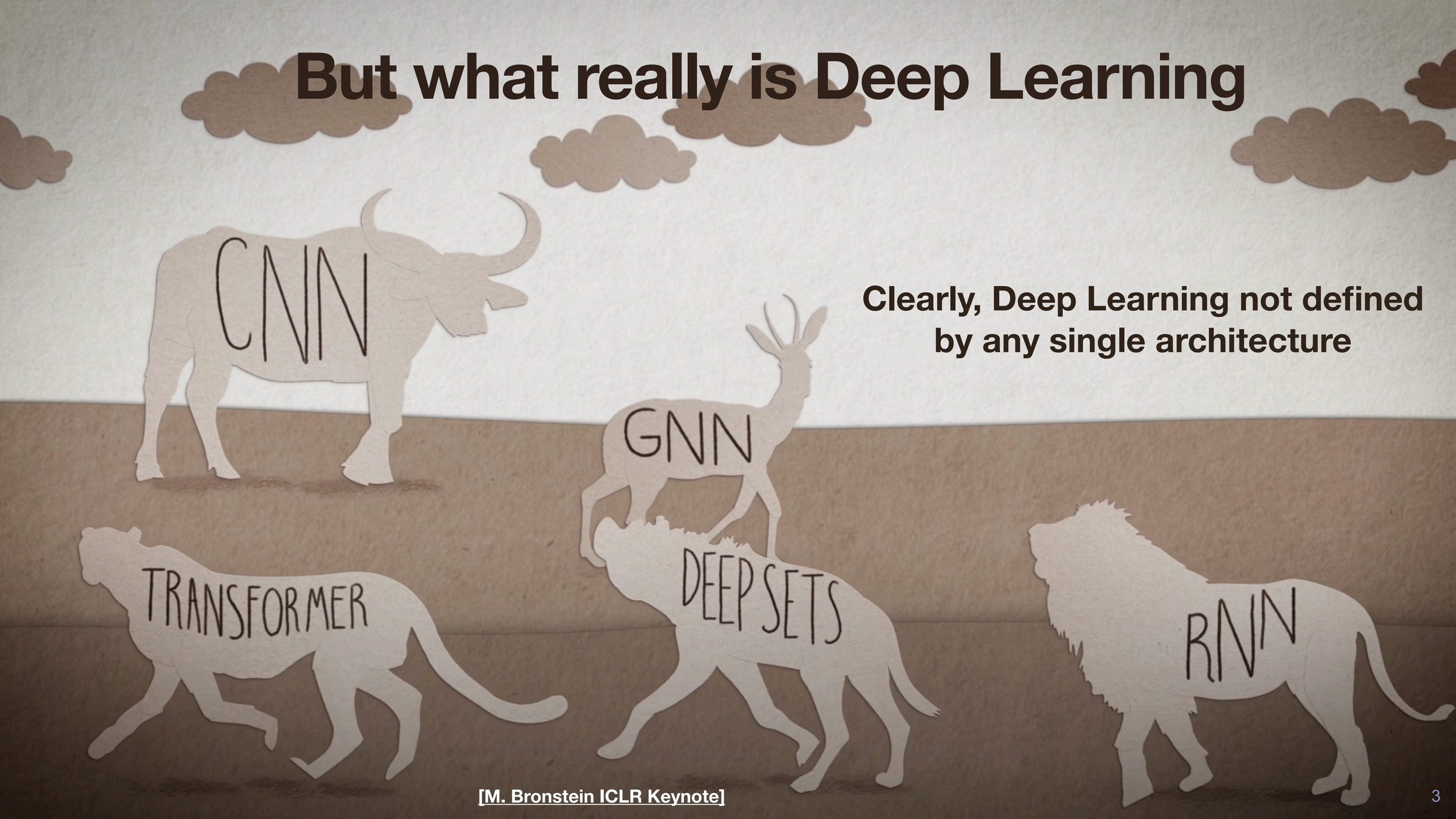
We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout"

https://papers.nips.cc › paper › 4824-imagenet-classific...

ImageNet Classification with Deep ... - NeurIPS Proceedings

by A Krizhevsky · 2012 · Cited by 107539 – Authors. Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. Abstract. We trained a large, deep convolutional neural network to classify the 1.3…

But what really is Deep Learning

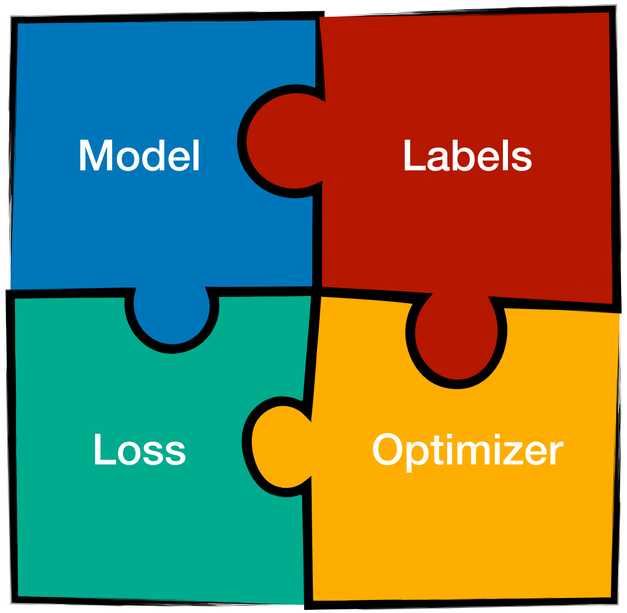


Clearly, Deep Learning not defined by any single architecture

[M. Bronstein ICLR Keynote]

The Ingredients of a (Deep) Learning system

How do produce a learned output



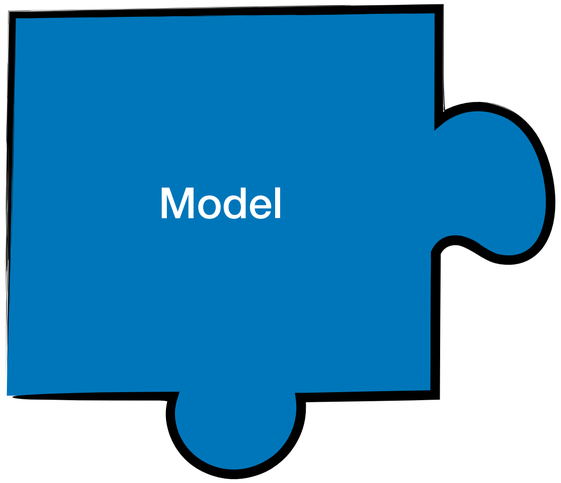
What to compare the output against

Formulate the learning objective

Navigating the Loss Landscape

We want to put physics into each of these!

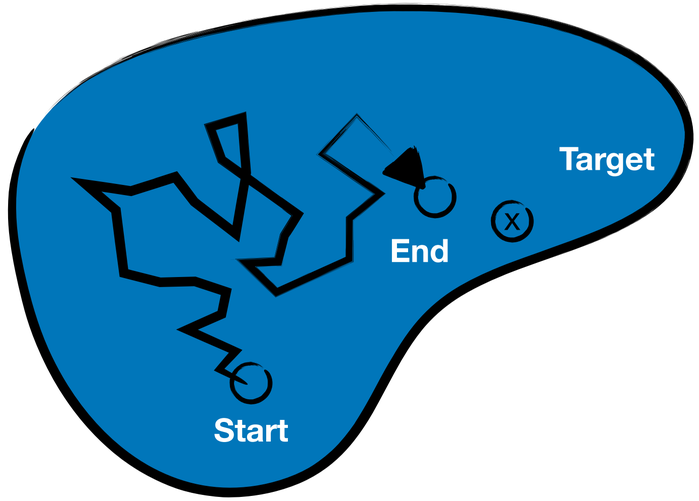
This talk: one weird trick to do it


Model

The Core of Deep Learning

Deep Learning is about searching through am extremely high-dimensional space

Space of Algorithms



Modern algorithms are parametrized by a lot of knobs

$$\hat{\phi} = \underset{\phi}{\operatorname{argmin}}\, \mathcal{L}(\phi)$$

Q: How could this possibly work?

SWITCH TRANSFORMERS: SCALING TO TRILLION PARAMETER MODELS WITH SIMPLE AND EFFICIENT SPARSITY

William Fedus* Google Brain liamfedus@google.com
Barret Zoph* Google Brain barretzoph@google.com
Noam Shazeer Google Brain noam@google.com

ABSTRACT

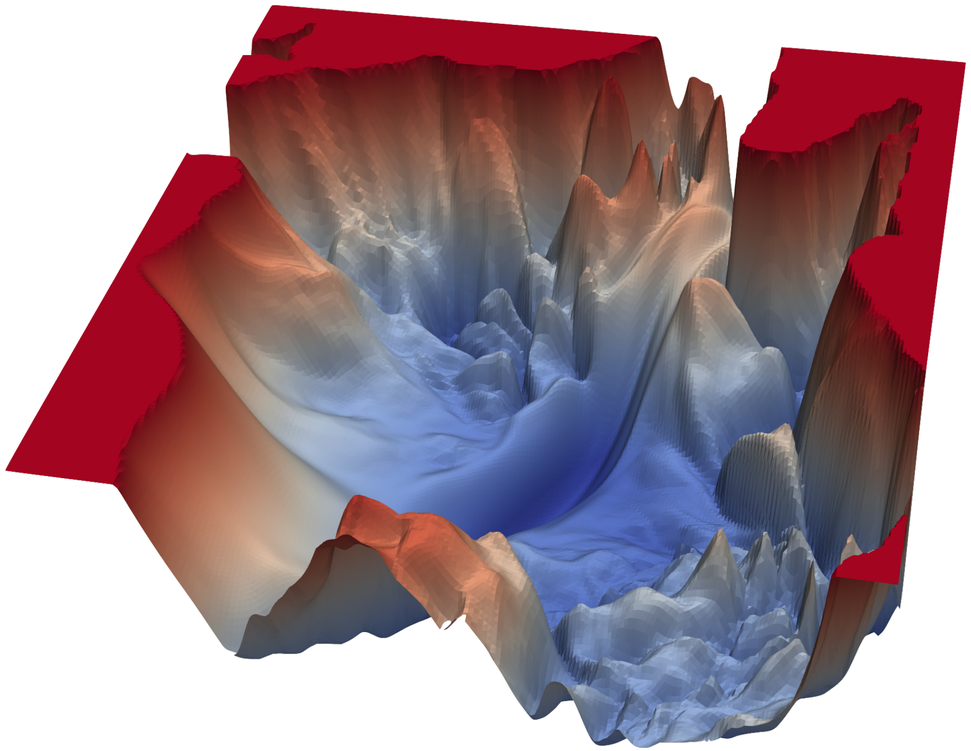
In deep learning, models typically reuse the same parameters for all inputs. Mixture of Experts (MoE) models defy this and instead select *different* parameters for each incoming example. The result is a sparsely-activated model – with an outrageous number of parameters – but a constant computational cost. However, despite several notable successes of MoE, widespread adoption has been hindered

arXiv:2101.03961

One Ingredient: a very good local sense of direction

Deep Learning relies crucially on an efficient computation of high-D gradients

$$\mathcal{L}(\phi) : \mathbb{R}^n \to \mathbb{R}$$



$$\nabla_\phi \mathcal{L} =$$

Automatic Differentiation

Numerical gradients $\Delta L/\Delta\phi$ hopeless in trillion-D, need exact gradients $\partial L/\partial\phi$

Automatic Differentiation: careful application of *chain rule to computer programs*

```python
import jax
import jax.numpy as jnp

def func(x):
  y = x
  for i in range(4):
    y += x[0]**2 + jnp.sin(x[1]) + jnp.exp(-x[2])
  y = y.sum()
  return y
```

exact gradients!

```python
gfunc = jax.value_and_grad(func)
gfunc(jnp.array([2.,3.,-2]))
```

```
(DeviceArray(141.36212, dtype=float32),
 DeviceArray([ 49.       , -10.8799095, -87.66867  ], dtype=float32))
```

TensorFlow PYTORCH

… but also C++, Fortran, …

$$f : \mathbb{R}^n \to \mathbb{R}^m$$

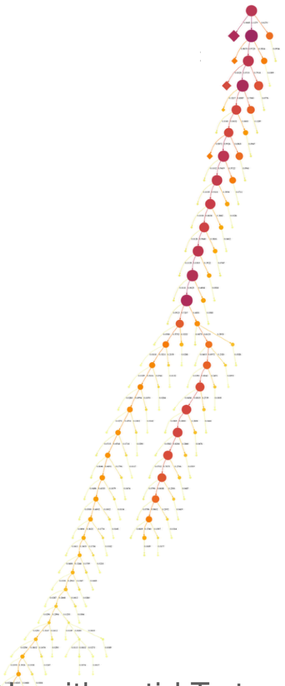
$$y = f(x) \qquad dy = J_f\, dx$$

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

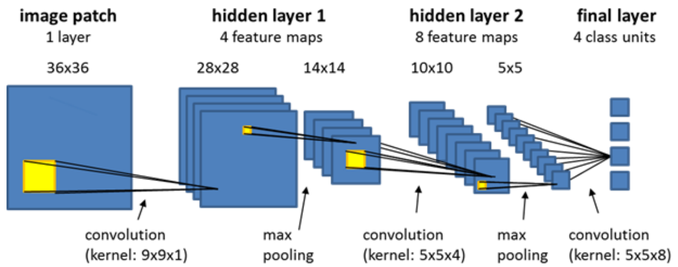
Inductive Bias

Architectures: By imposing structure on the program we can **bias learning towards sensible solutions** / reduce search space

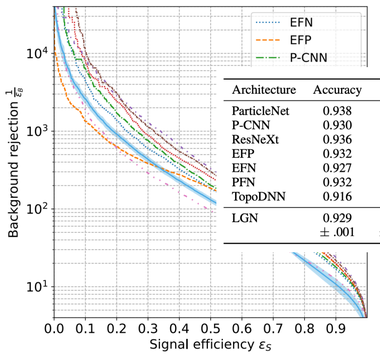
One direction: add physics bias on neural networks



[Jet NNs with anti-kT structure]

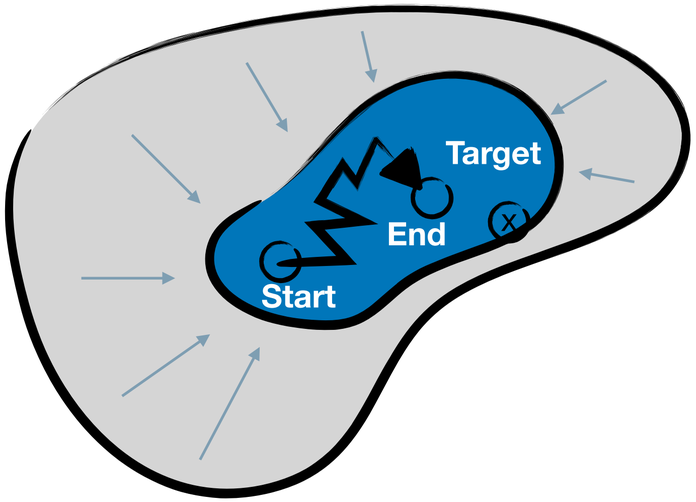


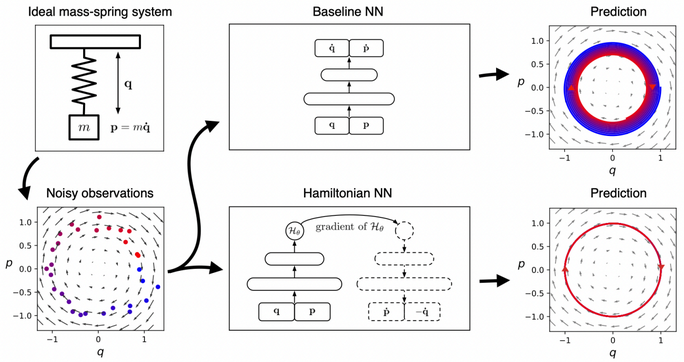
Translation Equivariance of CNNs



Architecture	Accuracy	AUC	$1/\epsilon_B$	#Param
ParticleNet	0.938	0.985	1298 ± 46	498k
P-CNN	0.930	0.980	732 ± 24	348k
ResNeXt	0.936	0.984	1122 ± 47	1.46M
EFP	0.932	0.980	384	1k
EFN	0.927	0.979	633 ± 31	82k
PFN	0.932	0.982	891 ± 18	82k
TopoDNN	0.916	0.972	295 ± 5	59k
LGN	0.929 $\pm$ .001	0.964 $\pm$ 0.018	435 ± 95	4.5k

Lorentz-Invariance





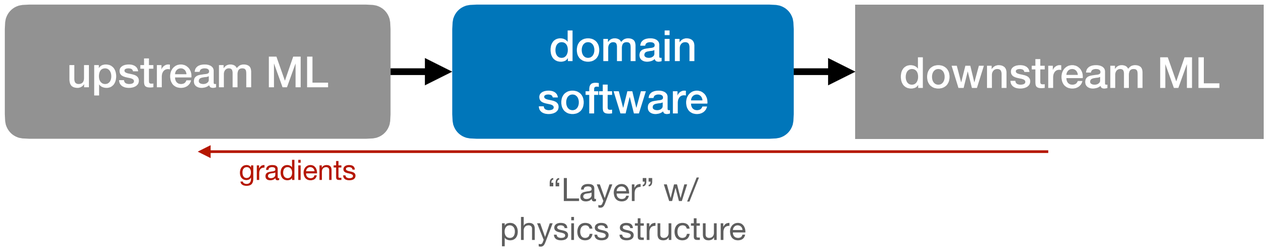
Hamiltonian Neural Nets

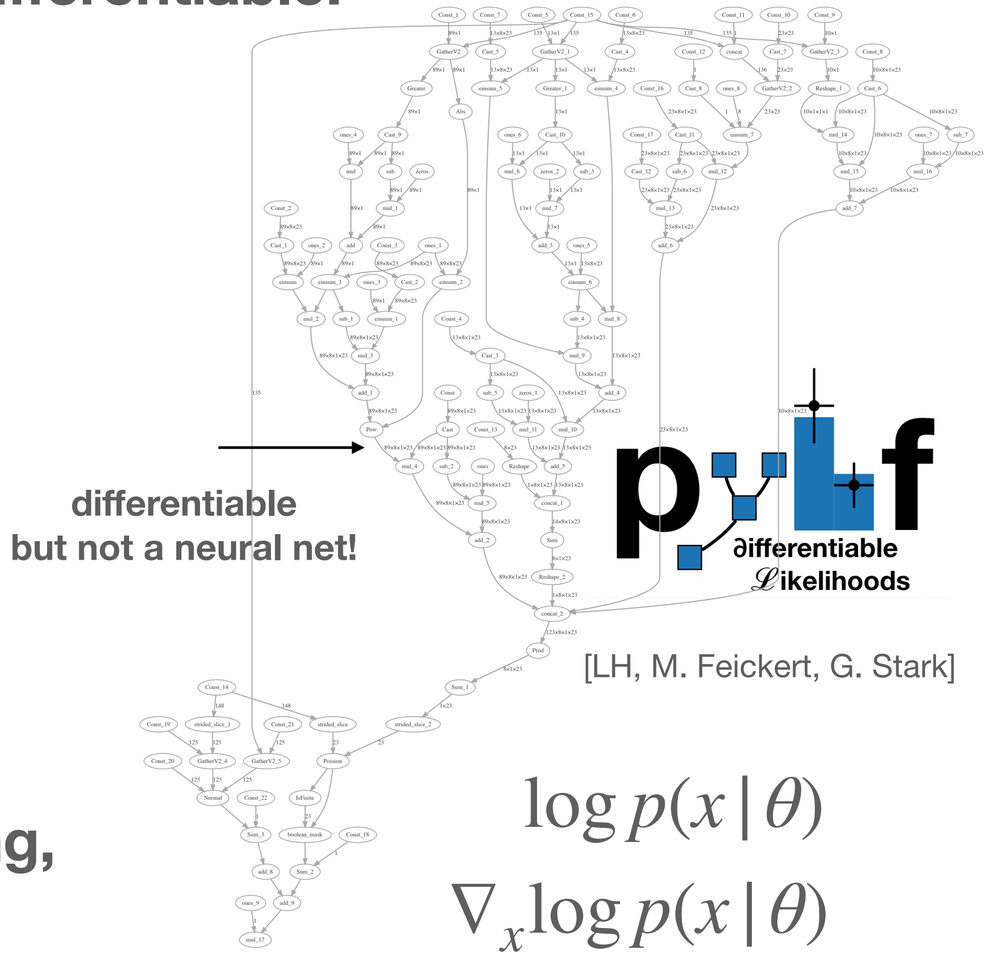
arXiv:1906.01563

The other direction:

Our software already contains a lot of the inductive bias or physics structure we would like to see in a data analysis → make it differentiable!

If we do, we can treat them like “layers” in neural networks and propagate gradients though them





[LH, M. Feickert, G. Stark]

Examples: differentiable statistical analysis, tracking, analysis frameworks, …

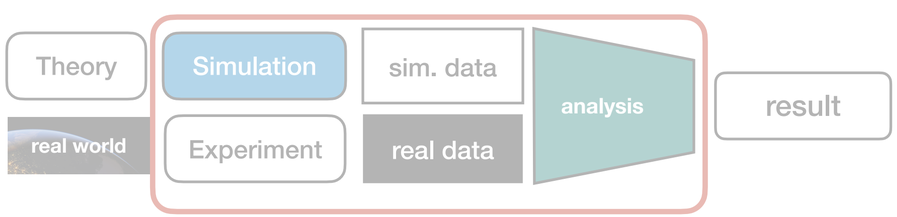
$$\log p(x \mid \theta)$$

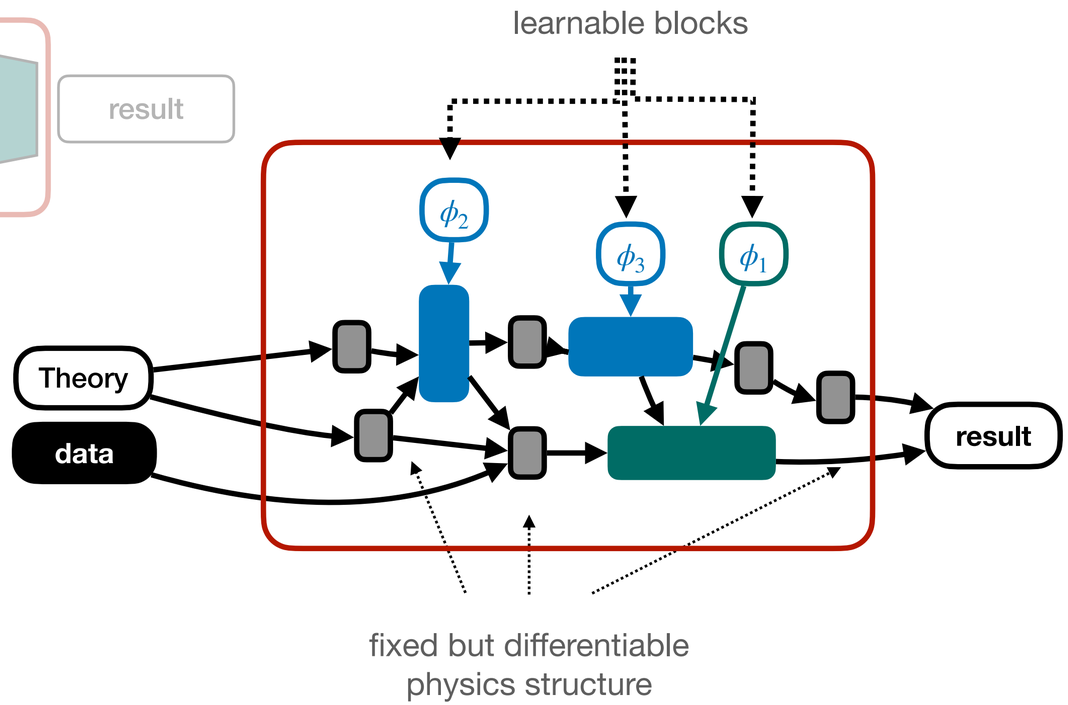
$$\nabla_x \log p(x \mid \theta)$$

$$\nabla_\theta \log p(x \mid \theta)$$

Mix & Match

Broader view of differentiable programming allows us to enforce physics structure where we need it to but also jointly optimise all “neural” blocks



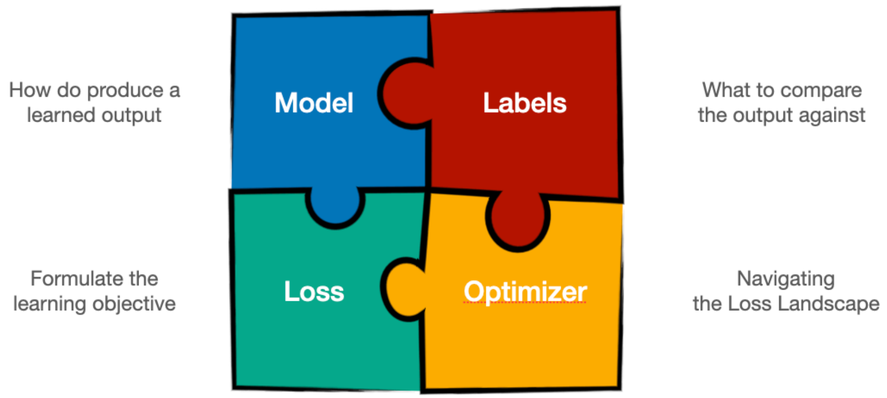


Differentiable Programming is the main topic of today’s tutorial

Differentiable Programming in ML

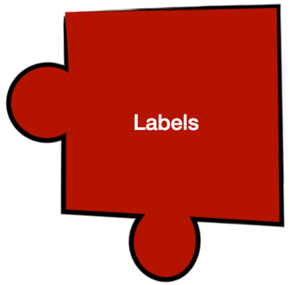
But we can use Differentiable Programming for more:

- **use it across the full ML landscape, not only to add inductive bias i.e. gradients allow us to define better losses, better labels**



- **There are many non-Deep Learning computations that would benefit from having access to gradient information e.g. Maximum-Likelihood Fitting**

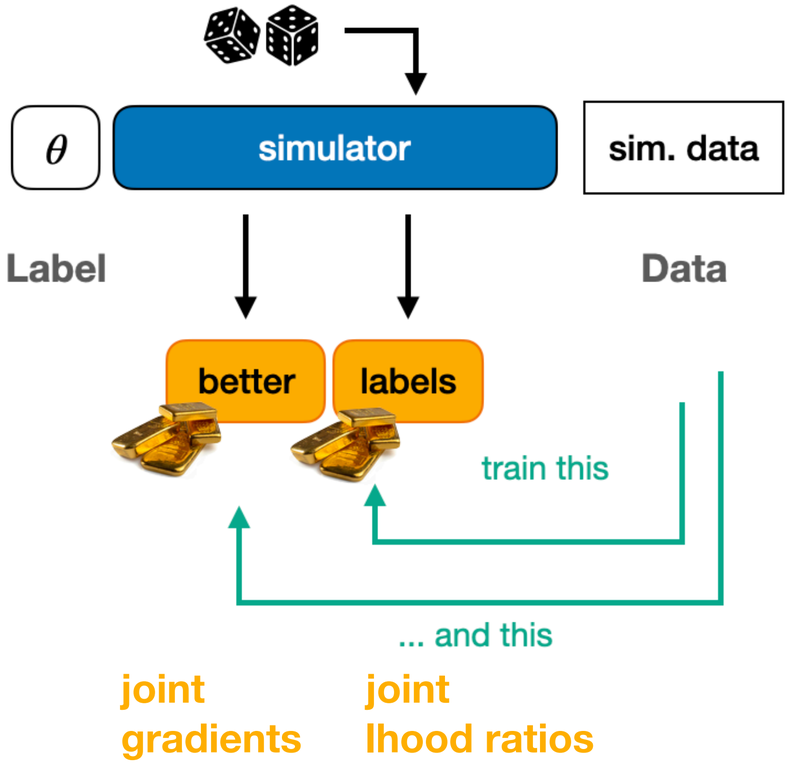
Differentiable Programming in ML

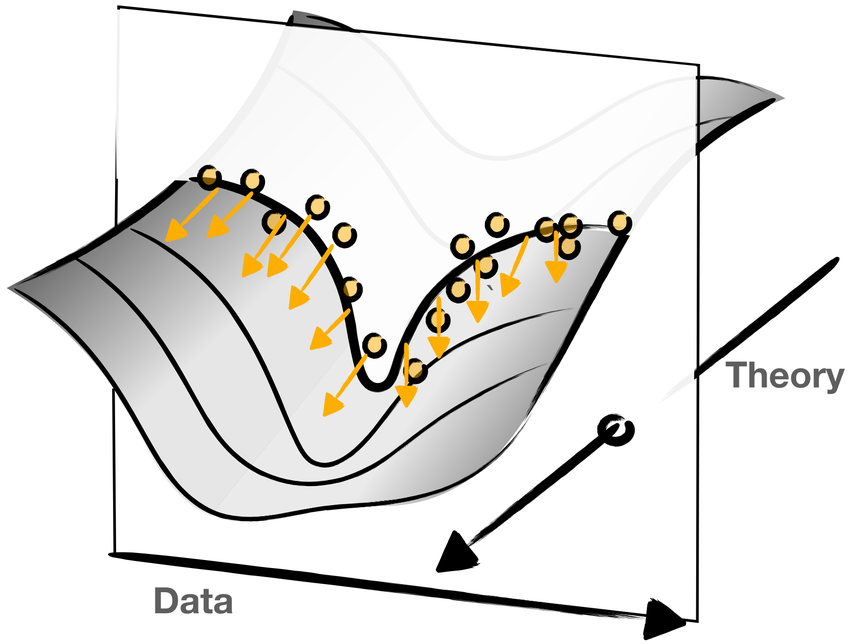


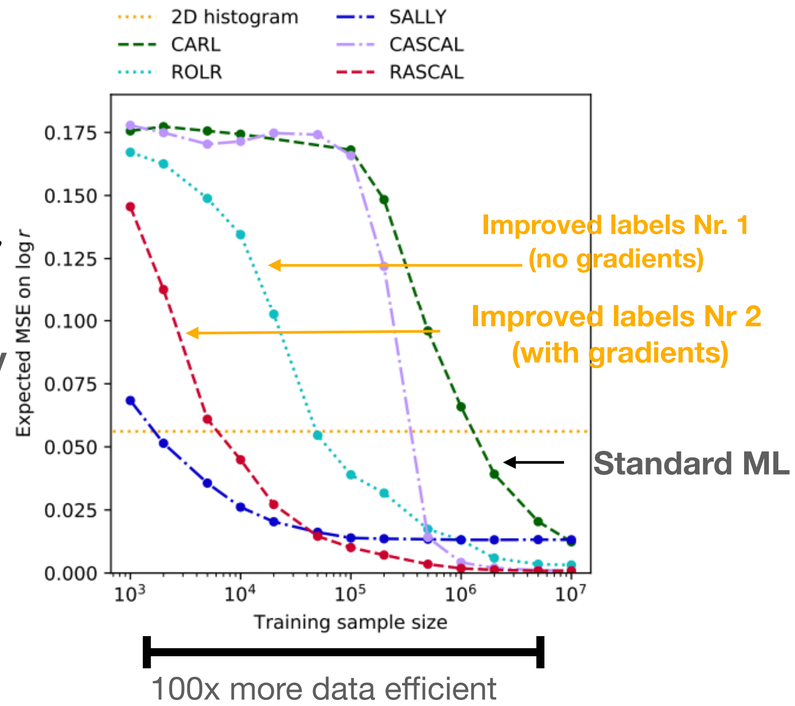
Better Labels:

Mining Gold is an approach to density ratio estimation that uses gradient information to get noisy but powerful labels: much more data efficient training

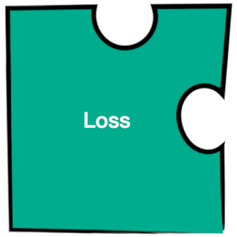
[Brehmer et a



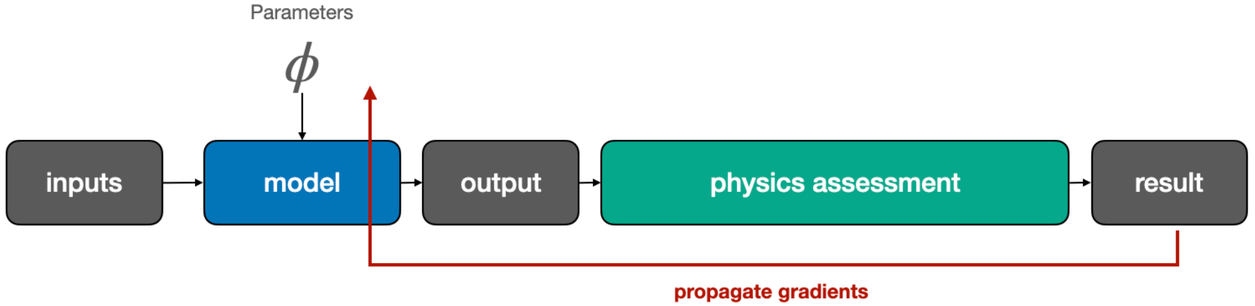




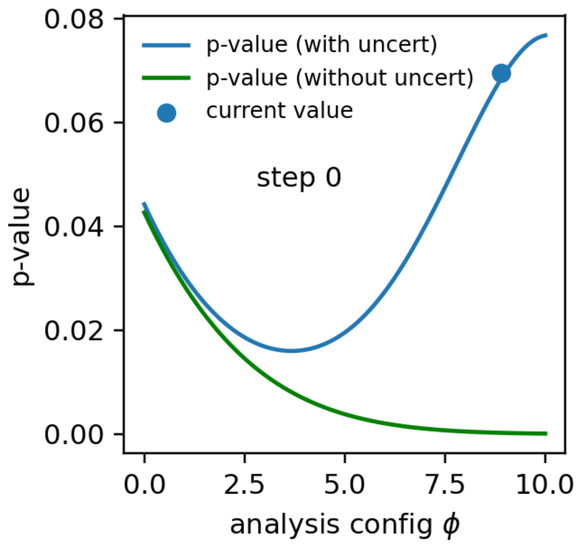
Differentiable Programming in ML

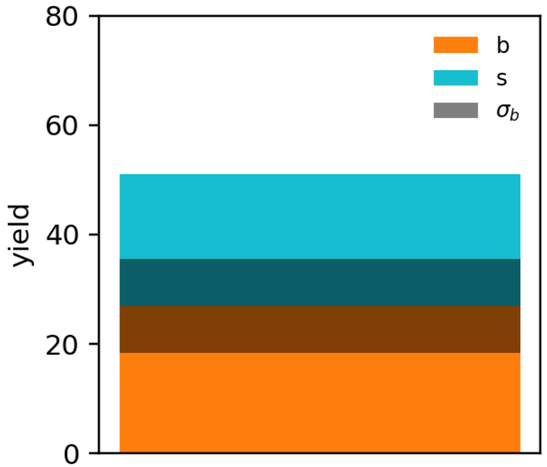


Better Losses:
Instead of having fancy architectures you can have fancy losses and optimize on the actual physics objective (note: boundary vs. model and loss is fluid)

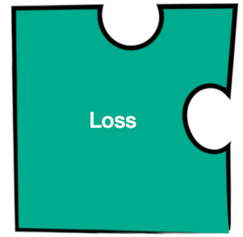


Check out the example of systematic-aware optimization with pyhf in this tutorial

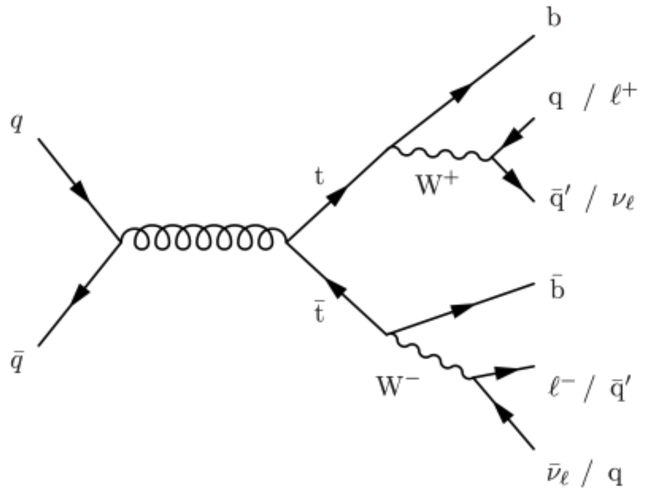


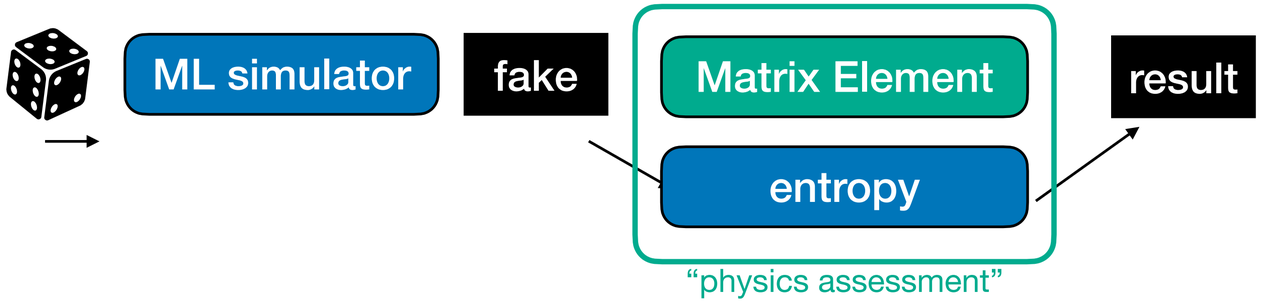


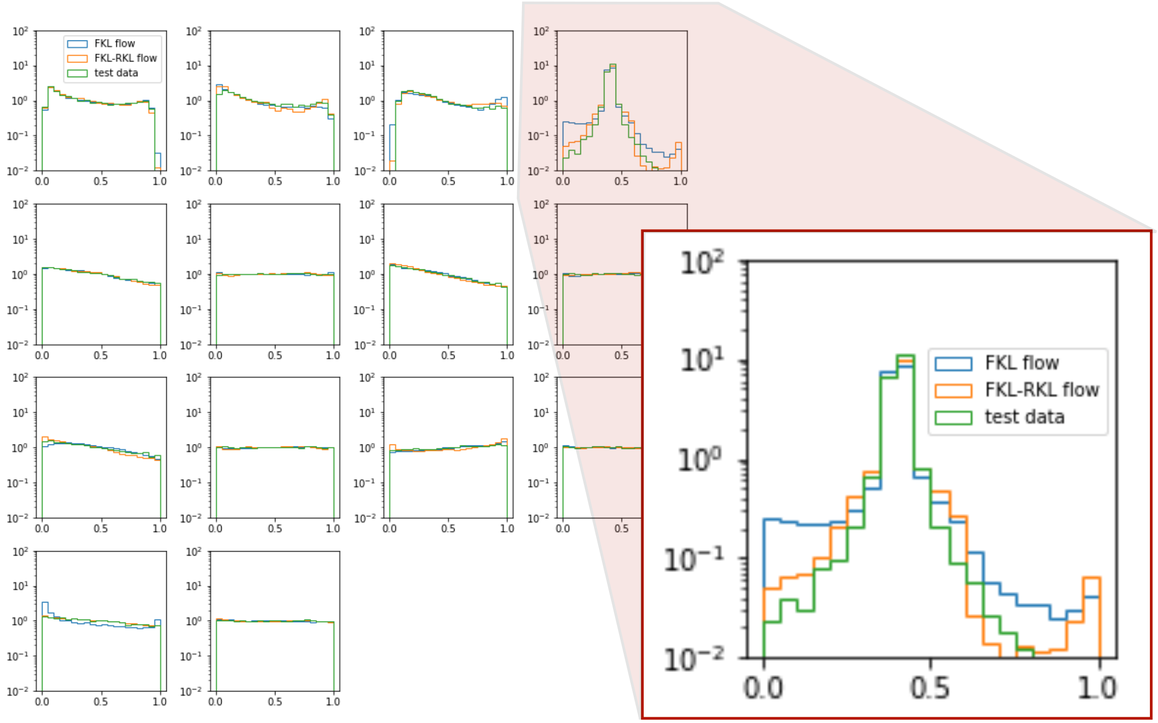
Differentiable Programming in ML



We can also use use gradients to train better genrative models by making our **simulators differentiable.** Check out the MadJax tutorial







better description of strong modes

`jax` - the best framework for diffable programming

Diffable Programming is a approach to programming, so many solutions exist

- PyTorch, Tensorflow are examples optimized for Deep Learning (i.e. fixed architectures, composition of layers, etc…)
- There are even FORTRAN, C++ and Julia automatic differentiation = differentiable programming frameworks

Practically speaking `jax` (https://github.com/google/jax) is one of the best frameworks for differentiable programming

- less focused on just Deep Learning
- if you have a numpy-based program, it's likely that you can make it diffable with jax as a simple drop-in replacement

This Tutorial is exlusively based on jax

Installation

Just Jax:

```
pip install jax
```

pyhf

```
pip install pyhf
```

Differentiable HEP Tools:

```
pip install relaxed
```

MadJax:

```
pip install git+https://github.com/madjax-hep/
madjax@master#egg=madjax
```

This Tutorial: Choose your own Adventure

There are three Binder repos you can work through

- **Introduction to Automatic Differentiation and JAX**

  Go to Repo Go To Binder **learn about jax and autodiff in general**

- **Differentiable Analysis Optimization with JAX with `pyhf` and `relaxed`**

  Go to Repo Go To Binder **learn how to do systematic-aware optimization**

- **Differentiable Matrix Elements with `madjax`**

  Go to Repo Go To Binder **learn how use differentiable matrix elements**