IMPROVEMENTS TO LUMIN & INFERNO IN PYTORCH

Giles Strong

IML Meeting, Online - 16/03/21

giles.strong@outlook.com

twitter.com/Giles_C_Strong

gilesstrong.github.io/website

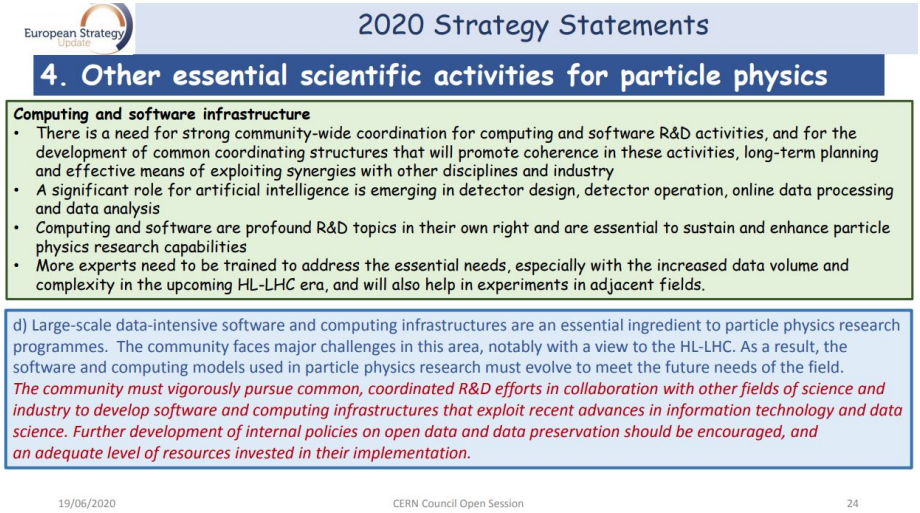
github.com/GilesStrong

LUMIN

What it is, and recent addictions

MACHINE LEARNING IN HEP

- Many analyses and experiment software now aim to benefit from using machine learning approaches
  - Often necessary in order to achieve competitive performance
  - ML is now an integral part of HEP
- But! Hardware and timing for model training can be a limitation for analysis-level researchers



Identified in [2020 update of the European Strategy for Particle Physics] as essential R&D

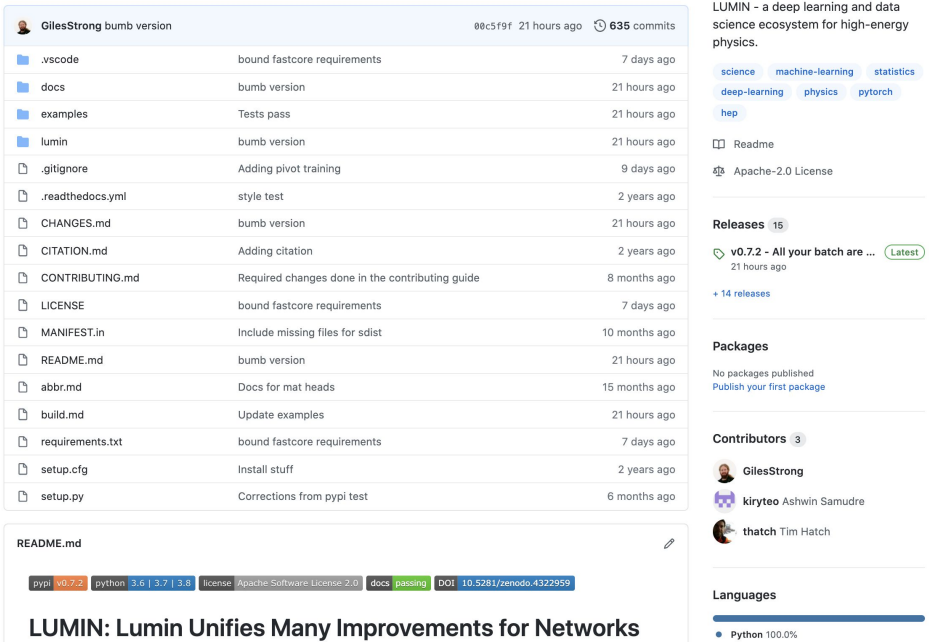
MODERN DEEP-LEARNING TECHNIQUES

- Strong, 2020 studied the impact of new DNNs techniques on performance and timing using benchmark HEP dataset (HiggsML)
  - HEP-specific data augmentation
  - 1cycle learning-rate scheduling
  - New architecture, activation function, etc.
- Solution matched top performance, but trained in 14 minutes on a laptop CPU
  - 86% effective speedup over 1st-place GPU (accounting for hardware improvements)
  - Competitive performance achievable by analysis-level researchers

	Our solution	1st place	2nd place	3rd place
Method	10 DNNs	70 DNNs	Many BDTs	108 DNNs
Train-time (GPU)	8 min	12 h	N/A	N/A
Train-time (CPU)	14 min	35 h	48 h	3 h
Test-time (GPU)	15 s	1 h	N/A	N/A
Test-time (CPU)	3 min	???	???	20 min
Score	3.806 ± 0.005	3.80581	3.78913	3.78682

LUMIN

- LUMIN is a PyTorch wrapper library designed primarily for HEP
  - Drop-in implementations of techniques and methods
  - Modular network design
  - HEP-specific data augmentation
  - Support for matrix & tensor data
  - Model & data interpretation
- Links:
  - Docs
  - Github
  - Colab examples
  - Issues - contributions welcome!
  - PyPI



USAGE

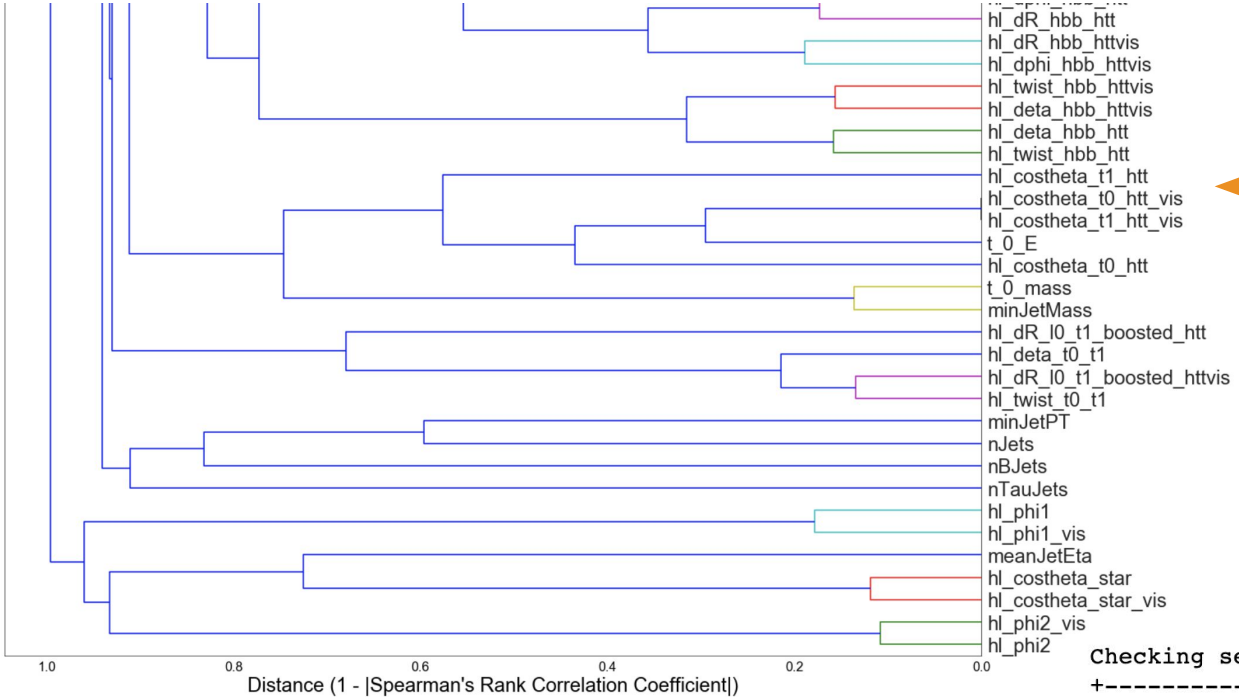
- LUMIN can be used to train neural networks for supervised classification, regression, and adversarial tasks using:
  - Columnar data (features in columns - events in rows)
  - And/or matrix data with arbitrary dimensions (i.e. 1D of 4-vectors, 2D & 3D grids of data, et cetera)
- Trained models can be exported to ONNX and TensorFlow
  - Can run in CMSSW via Tensorflow interface, see e.g cms_hh_tf_inference

Open in Colab

AUTOMATED FEATURE SELECTION

- Powerful & tunable methods to:
  - Safely filter clusters of monotonically related features; auto_filter_on_linear_correlation
  - Select features based on frequency of importance; repeated_rf_rank_features
  - Safely filter features based on mutual dependence; auto_filter_on_mutual_dependence
- Very useful for wide data, e.g.:
  - Predictive analytics with >200 features and no domain theory
  - CMS di-Higgs search with > 100 high-level features

FEATURE SELECTION: CLUSTERING & REMOVAL OF CORRELATED FEATURES

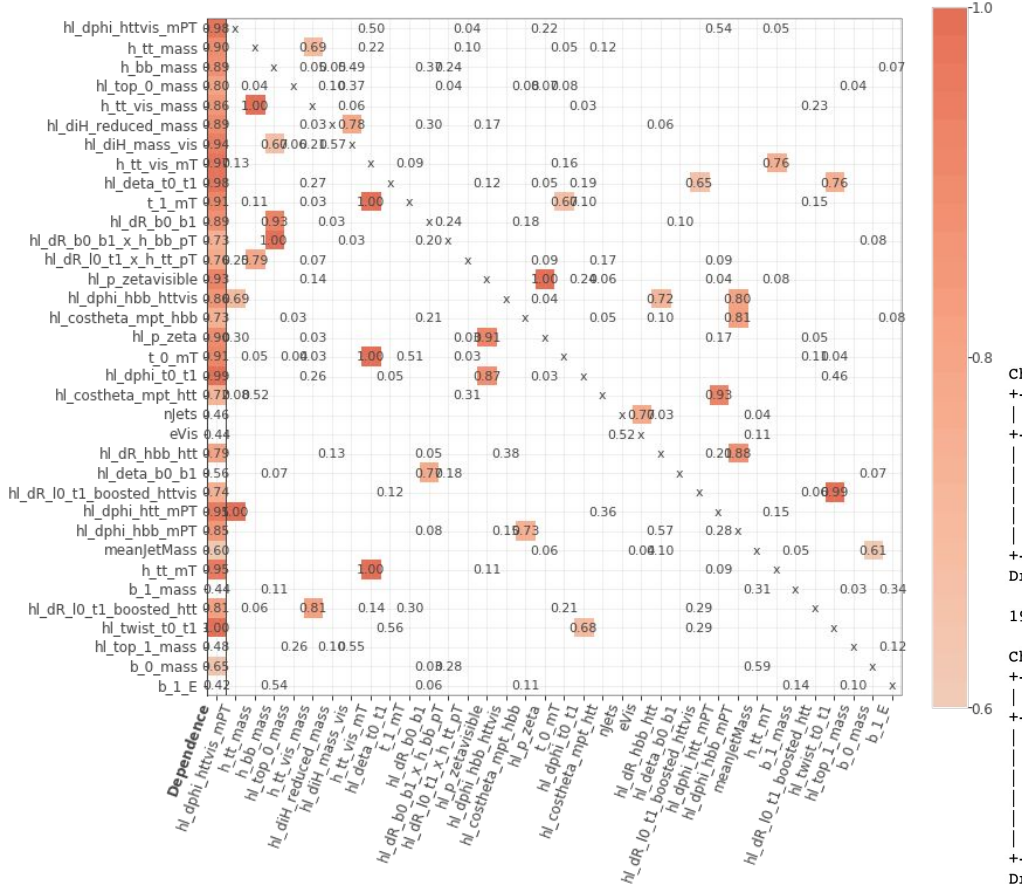


Cluster monotonically related features

Try to remove features without loss of performance

```
Checking set: ['t_0_mT', 'hl_dphi_t0_mPT']
+----------------+-------------+-------------+
|    Removed     |  OOB Score  |  Val Score  |
+----------------+-------------+-------------+
|      None      | 0.901±0.005 | 0.902±0.002 |
|     t_0_mT     | 0.899±0.004 | 0.901±0.002 |
| hl_dphi_t0_mPT | 0.902±0.005 | 0.903±0.002 |
+----------------+-------------+-------------+
Dropping hl_dphi_t0_mPT
```

FEATURE SELECTION: REMOVAL OF MUTUALLY DEPENDENT FEATURES



Train regressors for each feature

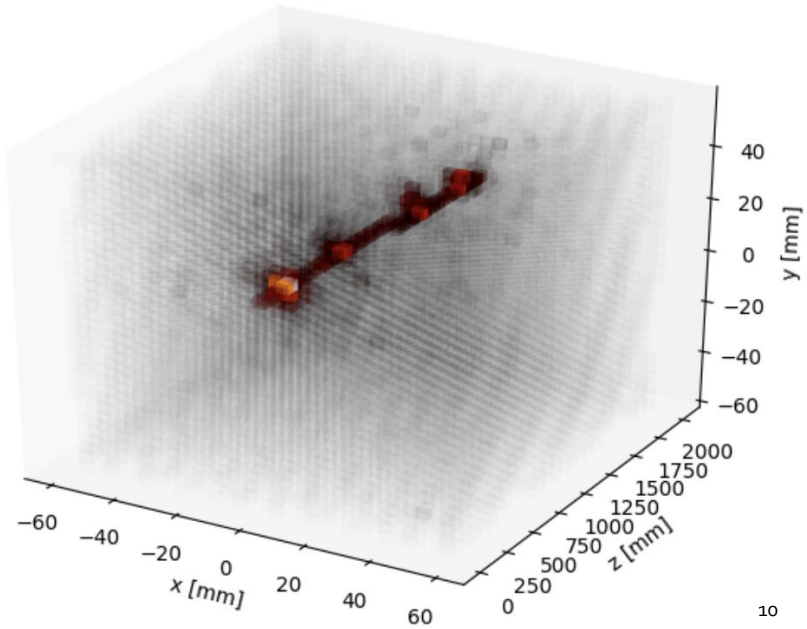
Try to remove predictable without loss of performance

```
Checking ['hl_twist_t0_t1', 'hl_dR_l0_t1_boosted_htt', 'h_tt_mT', 'hl_dphi_hbb_mPT']
+--------------------------+----------------+----------------+
|         Removed          |   OOB Score    |   Val Score    |
+--------------------------+----------------+----------------+
|           None           |  0.935±0.0008  |  0.934±0.0004  |
|      hl_twist_t0_t1      | 0.9349±0.0006  | 0.9339±0.0002  |
| hl_dR_l0_t1_boosted_htt  |  0.935±0.0005  | 0.9341±0.0002  |
|          h_tt_mT         |  0.934±0.0006  | 0.9337±0.0005  |
|     hl_dphi_hbb_mPT      | 0.9348±0.0006  | 0.9339±0.0006  |
+--------------------------+----------------+----------------+
Dropping hl_dR_l0_t1_boosted_htt

19 predictable features found to pass mutual dependence threshold of 0.8

Checking ['hl_dphi_htt_mPT', 'hl_dphi_t0_t1', 't_0_mT', 'hl_p_zeta', 'hl_dphi_hbb_httvis']
+--------------------+----------------+----------------+
|      Removed       |   OOB Score    |   Val Score    |
+--------------------+----------------+----------------+
|        None        |  0.934±0.0009  | 0.9344±0.0006  |
|  hl_dphi_htt_mPT   | 0.9341±0.0003  | 0.9343±0.0002  |
|   hl_dphi_t0_t1    | 0.9347±0.0003  | 0.9344±0.0006  |
|       t_0_mT       | 0.9338±0.0006  | 0.9338±0.0006  |
|     hl_p_zeta      | 0.9348±0.0006  | 0.9345±0.0002  |
| hl_dphi_hbb_httvis | 0.9346±0.0007  | 0.9344±0.0003  |
+--------------------+----------------+----------------+
Dropping hl_p_zeta
```

Open in Colab

HIGHER-ORDER DATA

- Early versions of LUMIN just supported columnar data, i.e. for analysis-level research
- Now can run models over higher-order data for reconstruction algorithms, e.g.:
  - 1D series of 4-vectors with RNNs/CNNs/GNNs
  - 2D jet images with CNNs
  - 3D detector volumes with CNNs
  - Users can provide their own custom architectures
  - Flat data can be passed in parallel to high-order data using MultiHead



3D data example: energy deposits in a calorimeter,
Source: Dorigo, Kieseler, Layer, Strong, 2020

CALLBACKS IMPROVEMENTS

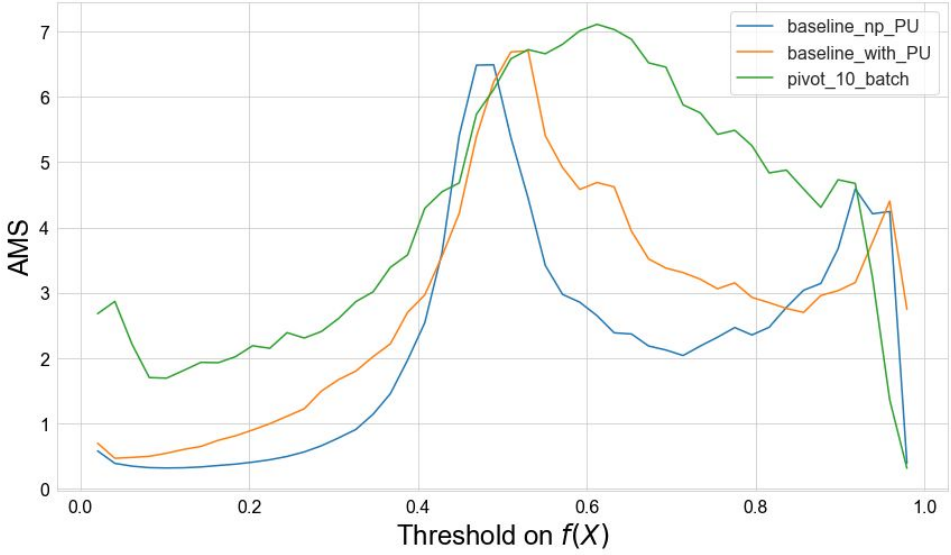
- Re-written model training loop to include more interjection points for callbacks
  - Allows for more possibilities and greater control of model training
- New *stateful* training means that all aspects of training (data, other callbacks, etc.) are accessible by each callback:
  - Each callback is aware of every other callback
  - Can modify all aspects of training
  - Callbacks can create new callbacks
  - Callbacks can manually compute losses rather than using a basic loss function

- Training start & end
- Epoch start & end
- Fold start & end
- Batch start & end
- After forwards pass
- Before & after backwards pass
- Before & after prediction

Open in Colab

LEARNING TO PIVOT

- Louppe, Kagan, Cranmer 2016
- An adversarial model tries to predict parameters based on outputs of main model
  - Forces main model to become invariant to parameters
- Implemented as callback in LUMIN
  - Can be dropped in to standard training loop without having to write a special adversarial loop



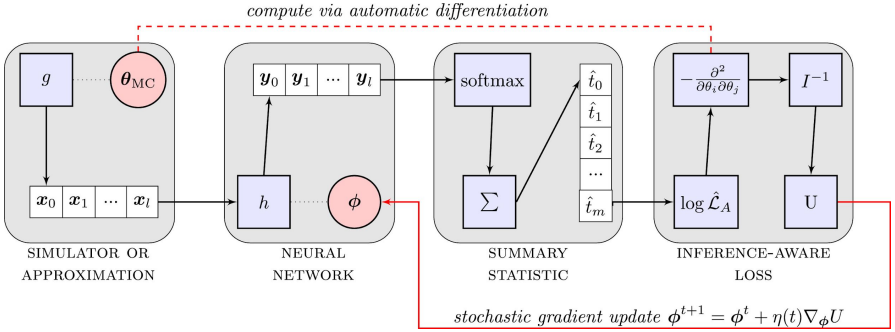
Pivot training forces classifier to become resilient to pile-up jets

PYTORCH_INFERNO

INFERNO overview and new implementation

INFERNO

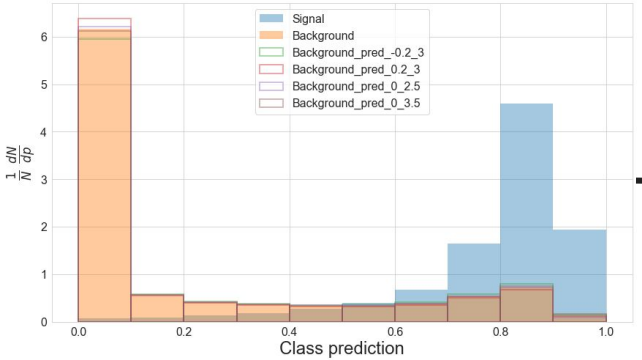
- de Castro & Dorigo 2018
- Directly optimise DNN for stat. Inf.
  - DNN output is binned summary statistic
    - Softmax output - can hard-assign after training
  - Loss is inversely proportional to the uncertainty on parameter of interest
    - Computed from inverted Hessian of likelihood w.r.t. parameters
    - Includes nuisances on both input features & normalisation
  - I.e. DNN encouraged to become sensitive to PoI and resilient to nuisances

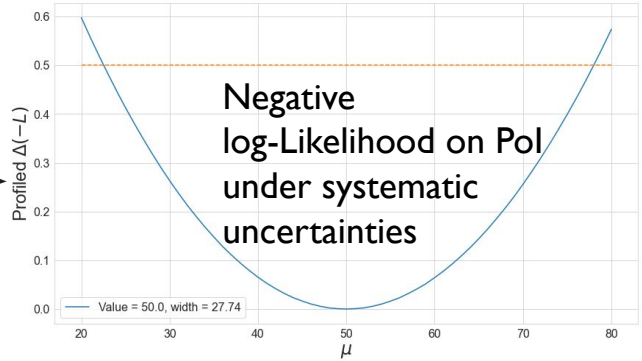


Source: paper

INFERNO BENEFIT - TOY EXAMPLE

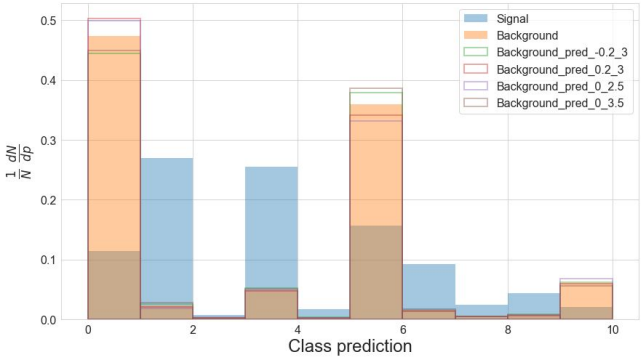
Binned output of binary cross-entropy classifier

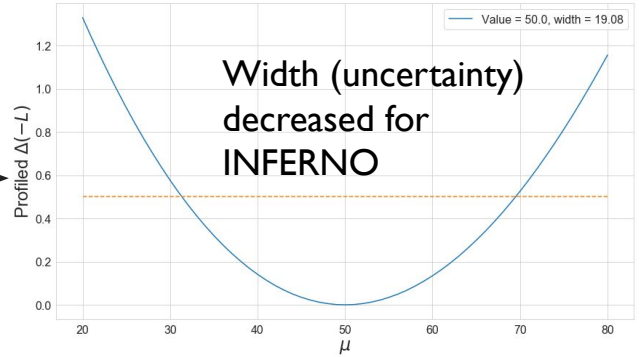




Negative log-Likelihood on PoI under systematic uncertainties

Hard-assigned output of INFERNO model





Width (uncertainty) decreased for INFERNO

INFERNO SUMMARY

1. Modify input data by nuisances set to zero (nuisances are tensors with gradient)
2. Forwards pass through network
3. Compute signal & bkg. shapes and normalise by rates
4. Compute profile log-likelihood (NLL) at Asimov count (sig+bkg and no nuisances)
5. Compute hessian of NLL w.r.t nuisances and PoI
6. Loss is the PoI element of the inverted hessian $((\nabla^2 \text{NLL})^{-1})_{\text{PoI,PoI}}$

IMPLEMENTATION

Requirements

- Either:
  - Access to input data before forward pass (paper version)
  - Or access to pre-modified data for up/down systematic shifts (interpolation approximation version)
- Access to model when computing loss (might be avoidable in certain tensor libs)
  - Need to remove gradient due to nuisances on predictions

Difficulties

- Losses normally expected to be a function that receives only predictions and targets
- Most callbacks and recorders expect loss to be averaged over data in batch (i.e. non-reduced element-wises losses exist), but INFERNO is not an averaged quantity.

IMPLEMENTATION

- Implement as a callback
  - Persistent class with access to the DNN
- on_batch_begin - modify data before forward pass
- on_forward_end - compute loss and manually set value
  - Requires training loop to be:
    - Aware of loss-setting-callbacks
    - Fine-grained enough (e.g. Keras 2 only has on_batch_begin on_batch_end (unsure about tf.keras))

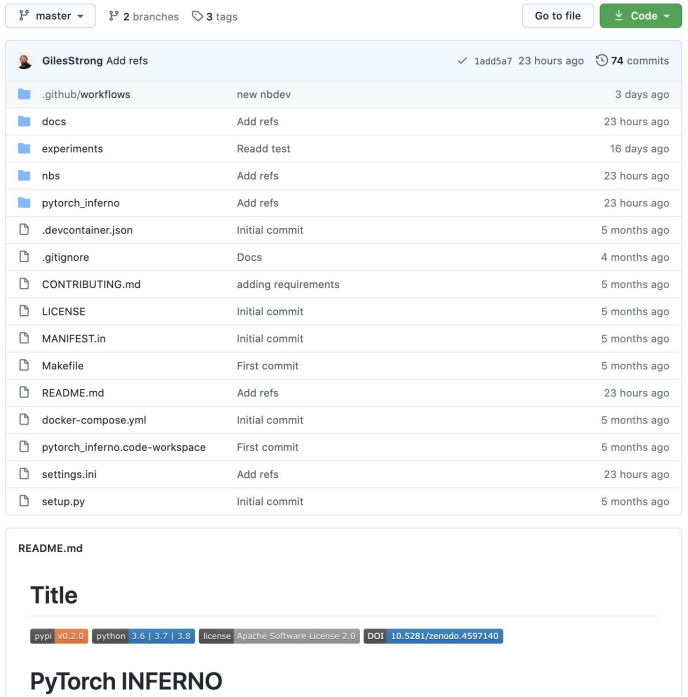
```
def _fit_batch(self, x:Tensor, y:Tensor, w:Tensor) -> None:
    self.x,self.y,self.w = to_device(x,self.device),to_device(y,self.device),to_device(w,self.device)
    for c in self.cbs: c.on_batch_begin()
    self.y_pred = self.model(self.x)
    if self.state != 'test' and self.loss_func is not None:
        self.loss_func.weights = self.w
        self.loss_val = self.loss_func(self.y_pred, self.y)
    for c in self.cbs: c.on_forwards_end()
    if self.state != 'train': return

    self.opt.zero_grad()
    for c in self.cbs: c.on_backwards_begin()
    self.loss_val.backward()
    for c in self.cbs: c.on_backwards_end()
    self.opt.step()
    for c in self.cbs: c.on_batch_end()
```

Modify data to include nuisances

Compute loss and set self.loss_val

PYTORCH INFERNO

- Provides minimal framework for:
  - Training & applying PyTorch DNNs
    - Plus necessary callback system
  - Computing profile likelihoods for inferring parameters of interest under uncertainty
- Successfully reproduces paper results
- Introduces an approximation method which works directly on ±1 sigma shifted data
- Links:
  - Docs
  - Github
  - PyPI



DISCUSSION

- Main aim of library to demonstrate non-disruptive implementation of INFERNO
  - LUMIN (compatible) version foreseen
- Tensorflow implementations:
  - Tensorflow 1: paper-inferno - Pablo de Castro
    - Custom framework
    - Loss computed via custom training loop
  - Tensorflow 2: inferno - Lukas Layer
    - Single Jupyter Notebook (runnable on Colab)
    - Loss computed via custom training loop

BLOG SERIES

- 5-part series on INFERNO and (param inference in HEP)
- Introduces & uses PyTorch package
- Parts 1 & 2 - intro to param inference
- Part 3 - ML classifier for summary stat.
- Part 4 - INFERNO for summary stat.
- Part 5 - Approximating INFERNO for easier application

SUMMARY

- LUMIN is a powerful library which is continuing to grow and develop
- Provides convenient access to recently published methods and ideas
- Already used in a diverse range of research tasks
- Would benefit greatly from user feedback
  - And more contributors!

- INFERNO has the potential to greatly improve HEP measurements
  - And other domains
- But needs to be tested out in a range of real-world problems
  - E.g. Lukas Layer is currently testing it on CMS open-data
- Can be tricky to set up, but implementations now exist for both Tensorflow 1 & 2, and PyTorch

BACKUP SLIDES

PROCESS - ONE UPDATE STEP

1. Minibatch: x - inputs, y - targets
2. Cache tensor of nuisances at nominal values (zero)
3. Modify inputs according to shape nuisances: x←x+nuisances
   a. E.g. $x_1 \leftarrow x_1 + 0$, $x_2 \leftarrow x_2 * (x_2 + 0)/x_2$,
   b. Nuisances don't change input values but allow gradients to be computed
   c. Instead add "the potential to be modified"
4. Pass x through NN (with softmax)
5. Predictions y_p: probabilities per bin/class
6. Index signal & background using y
   a. Sum counts per bin → sig & bkg shapes
   b. Detach/stop gradient on x_{bkg} and pass through NN → Asimov bkg template
      i. Ideally remove gradient from already computed bkg shape, but depends on tensor library
7. Compute stats.:
   a. $t_{exp} = N_s \cdot shape_{sig} + N_b \cdot shape_{bkg}$
   b. $t_{asimov} = N_{s,true} \cdot shape_{sig} + N_{b,true} \cdot shape_{bkg,asimov}$

PROCESS - ONE UPDATE STEP

8. Build Poisson likelihood:
   a. NLL = -Pois(t_{exp}).log_prob(t_{asimov}).sum()
   b. N_s, N_b, and shape nuisances already at nominal values, NLL minimised, profiling unnecessary
   c. Add constraints on nuisances if present
9. Compute Hessian of NLL w.r.t. PoI and nuisances: ∇^2NLL (2D square matrix),
   a. N.B at minimum, ∇NLL = 0
   b. Hessian diagonal = effect of each param on NLL
   c. Hessian off-diagonal (symmetric) = interplay between params
10. Invert Hessian & return element corresponding to PoI as loss value
    a. Want PoI Hessian element to be as large as possible: NLL narrower in PoI axis
    b. But want nuisance elements to be as small as possible: NLL flatter in nuisance axes
    c. Inversion "mixes" elements in Hessian
    d. Minimising PoI element of inverted Hessian leads to desired result
11. Backprop loss value and update weights as normal