



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 additions

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



2020 Strategy Statements

4. Other essential scientific activities for particle physics

Computing and software infrastructure

- There is a need for strong community-wide coordination for computing and software R&D activities, and for the development of common coordinating structures that will promote coherence in these activities, long-term planning and effective means of exploiting synergies with other disciplines and industry
- A significant role for artificial intelligence is emerging in detector design, detector operation, online data processing and data analysis
- Computing and software are profound R&D topics in their own right and are essential to sustain and enhance particle physics research capabilities
- More experts need to be trained to address the essential needs, especially with the increased data volume and complexity in the upcoming HL-LHC era, and will also help in experiments in adjacent fields.

d) Large-scale data-intensive software and computing infrastructures are an essential ingredient to particle physics research programmes. The community faces major challenges in this area, notably with a view to the HL-LHC. As a result, the software and computing models used in particle physics research must evolve to meet the future needs of the field. *The community must vigorously pursue common, coordinated R&D efforts in collaboration with other fields of science and industry to develop software and computing infrastructures that exploit recent advances in information technology and data science. Further development of internal policies on open data and data preservation should be encouraged, and an adequate level of resources invested in their implementation.*

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
 - [Icycle](#) 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	1 st place	2 nd place	3 rd 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](#)

GilesStrong bump version			00c5f9f 21 hours ago	🔄 635 commits
📁 .vscode	bound fastcore requirements	7 days ago		
📁 docs	bump version	21 hours ago		
📁 examples	Tests pass	21 hours ago		
📁 lumin	bump version	21 hours ago		
📄 .gitignore	Adding pivot training	9 days ago		
📄 .readthedocs.yml	style test	2 years ago		
📄 CHANGES.md	bump version	21 hours ago		
📄 CITATION.md	Adding citation	2 years ago		
📄 CONTRIBUTING.md	Required changes done in the contributing guide	8 months ago		
📄 LICENSE	bound fastcore requirements	7 days ago		
📄 MANIFEST.in	include missing files for sdist	10 months ago		
📄 README.md	bump version	21 hours ago		
📄 abbr.md	Docs for mat heads	15 months ago		
📄 build.md	Update examples	21 hours ago		
📄 requirements.txt	bound fastcore requirements	7 days ago		
📄 setup.cfg	Install stuff	2 years ago		
📄 setup.py	Corrections from pypi test	6 months ago		

README.md

📄 pypi v0.7.2 | python 3.6 | 3.7 | 3.8 | license Apache Software License 2.0 | docs passing | DOI 10.5281/zenodo.4322959

LUMIN: Lumin Unifies Many Improvements for Networks

LUMIN - a deep learning and data science ecosystem for high-energy physics.

science machine-learning statistics
deep-learning physics pytorch
hep

📖 Readme

📄 Apache-2.0 License

📄 Releases 15

📄 v0.7.2 - All your batch are ... (Latest)
21 hours ago

+ 14 releases

📄 Packages

No packages published
[Publish your first package](#)

👤 Contributors 3

👤 GilesStrong

👤 kiryteo Ashwin Samudra

👤 thatch Tim Hatch

📄 Languages

• Python 100.0%

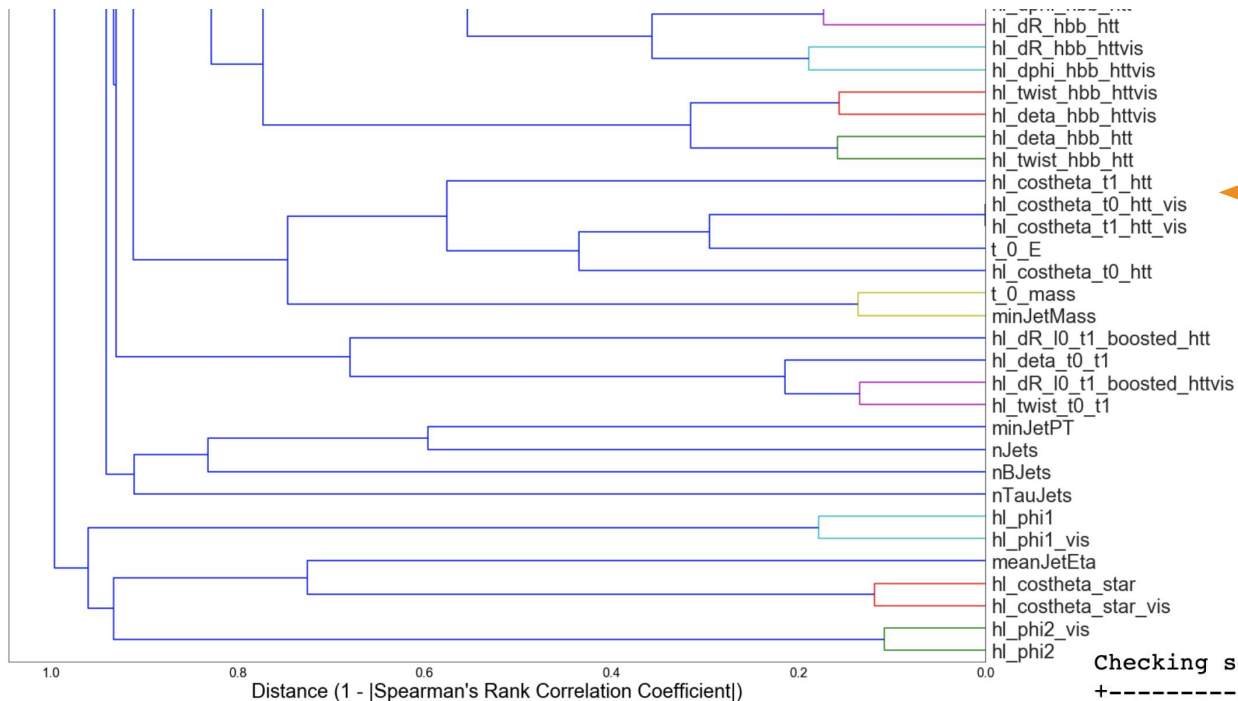
USAGE

- 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. ID 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](#)

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

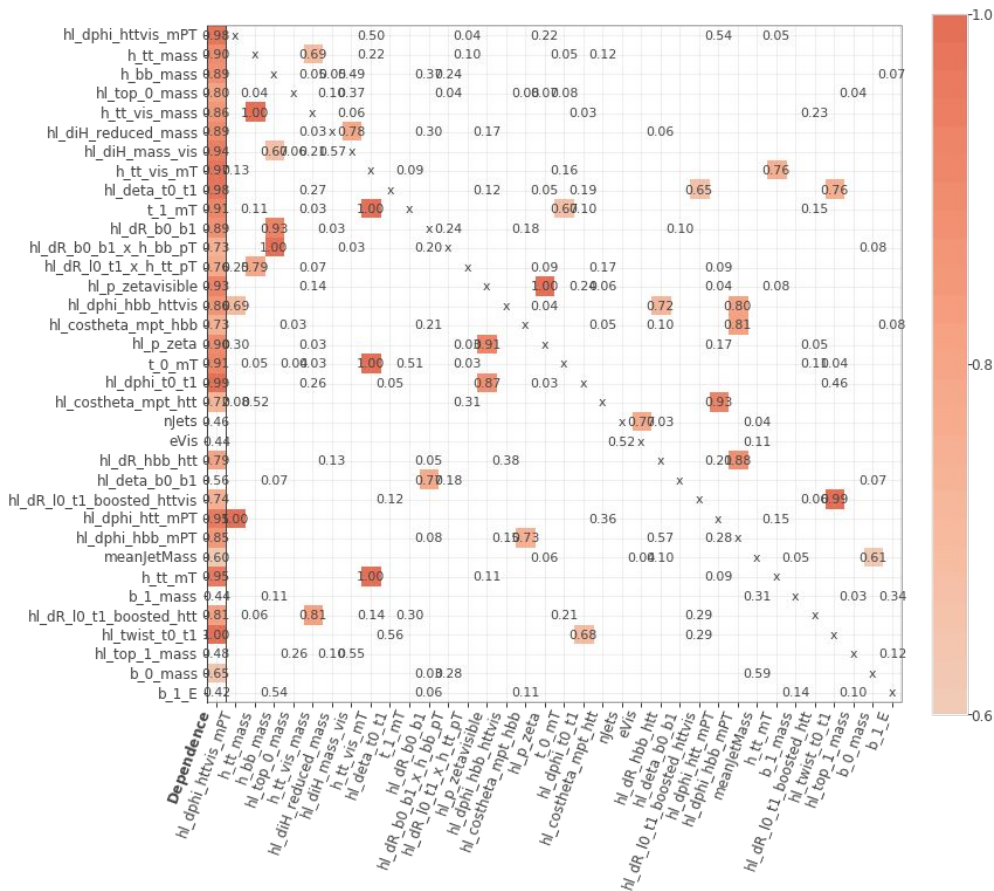
Try to remove features without loss of performance

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

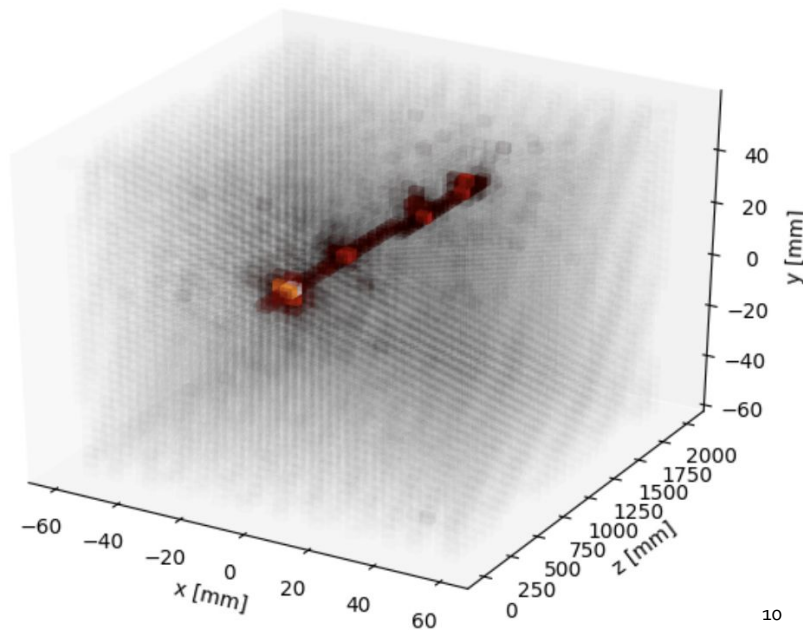
Checking ['hl_dphi_htt_mPT', 'hl_dphi_t0_t1', 't_0_mT', 'hl_p_zeta', 'hl_dphi_hbb_httvis']

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

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,

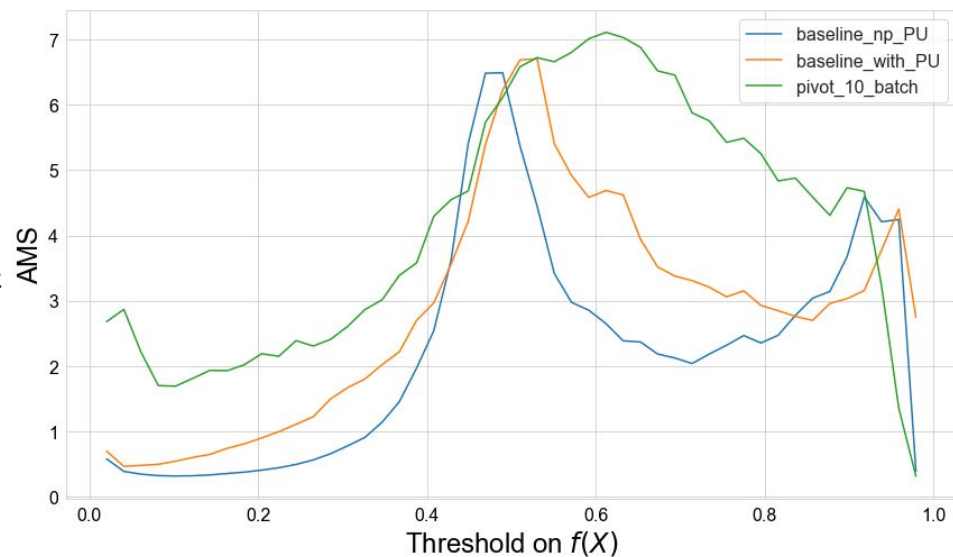
Source: [Dorigo, Kieseler, Layer, Strong, 2020](#)

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

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

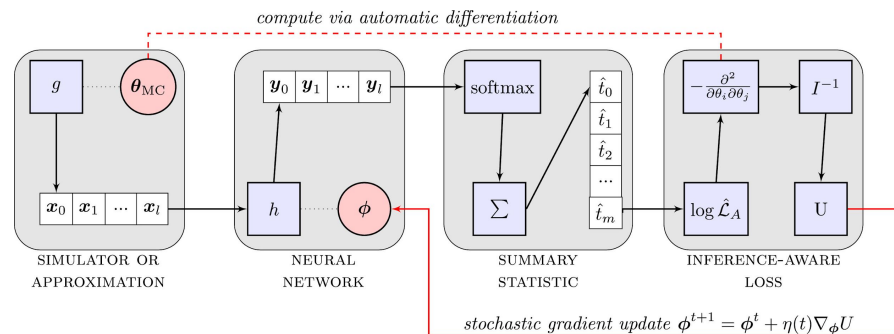


PYTORCH_INFERNNO

INFERNNO overview and new implementation

INFERNO

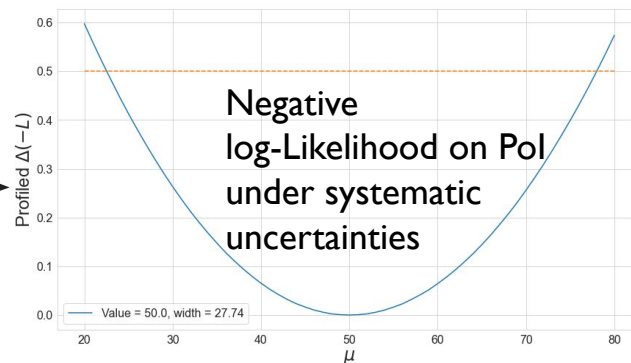
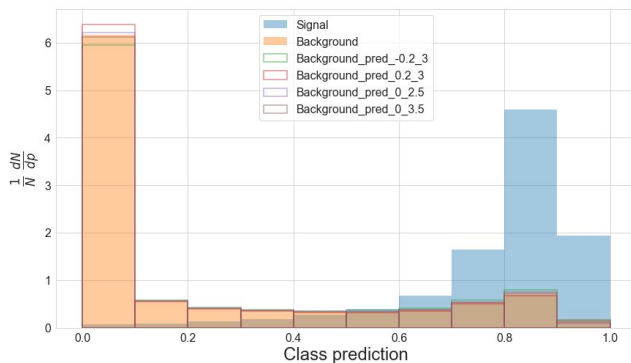
- [de Castro & Dorigo 2018](#)
- Directly optimise DNN for stat. Inf.
 - DNN output is binned summary statistic
 - Softmax output - can hard-assign after training
 - Loss is inversely proportional to the uncertainty on parameter of interest
 - Computed from inverted Hessian of likelihood w.r.t. parameters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



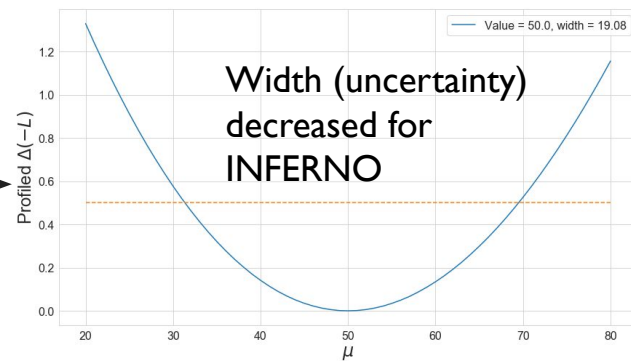
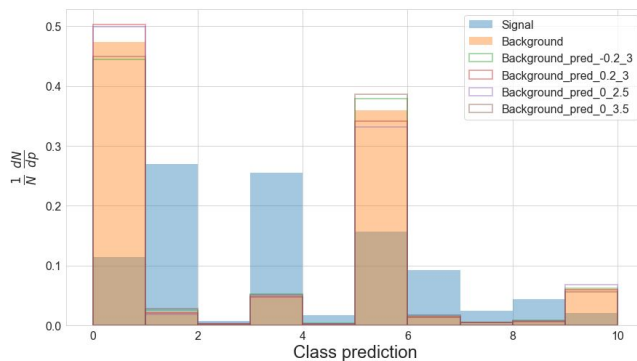
Source: paper

INFERNO BENEFIT - TOY EXAMPLE

Binned output of binary cross-entropy classifier



Hard-assigned output of INFERNO model



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 Pol
6. Loss is the Pol element of the inverted hessian $((\nabla^2 \text{NLL})^{-1})_{\text{Pol,Pol}}$

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](#)

master 2 branches 3 tags Go to file Code

GilesStrong Add refs 1add5a7 23 hours ago 74 commits

.github/workflows	new nbdev	3 days ago
docs	Add refs	23 hours ago
experiments	Readd test	16 days ago
nbs	Add refs	23 hours ago
pytorch_inferno	Add refs	23 hours ago
.devcontainer.json	Initial commit	5 months ago
.gitignore	Docs	4 months ago
CONTRIBUTING.md	adding requirements	5 months ago
LICENSE	Initial commit	5 months ago
MANIFEST.in	Initial commit	5 months ago
Makefile	First commit	5 months ago
README.md	Add refs	23 hours ago
docker-compose.yml	Initial commit	5 months ago
pytorch_inferno.code-workspace	First commit	5 months ago
settings.ini	Add refs	23 hours ago
setup.py	Initial commit	5 months ago

README.md

Title

pypi v0.2.0 python 3.6 | 3.7 | 3.8 license Apache-Software License 2.0 DOI 10.5281/zenodo.4597140

PyTorch INFERNO

About

PyTorch implementation of inference aware neural optimisation (de Castro and Dorigo, 2018 <https://www.sciencedirect.com/science/article/pii/S0010465519301948>)

gilesstrong.github.io/pytorch_inferno/

[inferno](#) [pytorch](#)
[statistical-inference](#) [neural-networks](#)
[likelihood-free-inference](#)

Readme

Apache-2.0 License

Releases 3

v0.2.0.a (Latest)
yesterday

+ 2 releases

Packages

No packages published

Languages

Jupyter Notebook 99.6%
Other 0.4%

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:
 $x \leftarrow x + \text{nuisances}$
 - 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 \rightarrow sig & bkg shapes
 - b. Detach/stop gradient on x_{bkg} and pass through NN \rightarrow Asimov bkg template
 - i. Ideally remove gradient from already computed bkg shape, but depends on tensor library
7. Compute stats.:
 - a. $t_{\text{exp}} = N_s \cdot \text{shape}_{\text{sig}} + N_b \cdot \text{shape}_{\text{bkg}}$
 - b. $t_{\text{asimov}} = N_{s,\text{true}} \cdot \text{shape}_{\text{sig}} + N_{b,\text{true}} \cdot \text{shape}_{\text{bkg,asimov}}$

PROCESS - ONE UPDATE STEP

8. Build Poisson likelihood:

- a. $\text{NLL} = -\text{Pois}(t_{\text{exp}}).\text{log_prob}(t_{\text{asimov}}).\text{sum}()$
- 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. Pol and nuisances: $\nabla^2\text{NLL}$ (2D square matrix),

- a. N.B at minimum, $\nabla\text{NLL} = 0$
- b. Hessian diagonal = effect of each param on NLL
- c. Hessian off-diagonal (symmetric) = interplay between params

10. Invert Hessian & return element corresponding to Pol as loss value

- a. Want Pol Hessian element to be as large as possible: NLL narrower in Pol 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 Pol element of inverted Hessian leads to desired result

11. Backprop loss value and update weights as normal