

TÉCNICO
LISBOA



LUMIN

A DATA SCIENCE AND DEEP LEARNING
ECOSYSTEM FOR HIGH-ENERGY PHYSICS

Giles Strong

IML Workshop, CERN - 17/04/2019

giles.strong@outlook.com

twitter.com/Giles_C_Strong

Amva4newphysics.wordpress.com

github.com/GilesStrong



CASE STUDY

2014 Higgs ML Kaggle Challenge

OVERVIEW

- Last year I spent quite a bit of time studying the [2014 Higgs ML Kaggle challenge](#)
- The dataset has several nice properties:
 - Representative of typical high-energy physics (HEP) problem in terms of task and dataset size
 - Realistic performance metric - Approximate Median Significance (AMS)
 - Well studied = strong baselines to measure performance

Higgs challenge

Higgs Boson Machine Learning Challenge

Use the ATLAS experiment to identify the Higgs boson

\$13,000 · 1,785 teams · 4 years ago

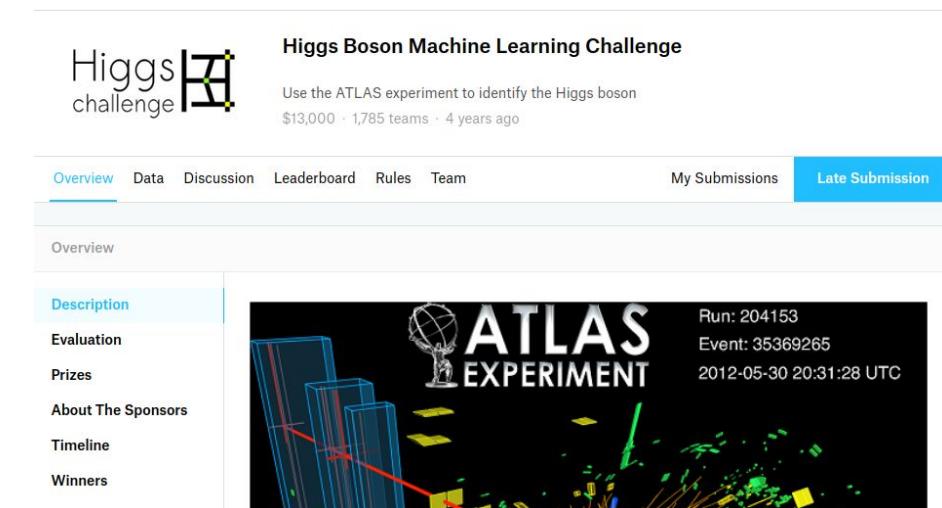
Overview Data Discussion Leaderboard Rules Team My Submissions Late Submission

Overview

Description Evaluation Prizes About The Sponsors Timeline Winners

ATLAS EXPERIMENT

Run: 204153
Event: 35369265
2012-05-30 20:31:28 UTC



RESULTS

- The aim was to test whether several recently developed deep neural network techniques could bring improvements in a HEP context
- Full study documented in [AMVA4NewPhysics Deliverable 1.4](#)
- Also reported at the [last IML meeting of 2018](#)
- Approach also used in [CMS HL-LHC projection studies](#) for CERN Yellow Report



AMVA4NEWPHYSICS ITN

WORK PACKAGE 1 - DELIVERABLE 1.4

Final Activity Report: Classification and Regression Tools in Higgs Measurements

AMVA4NewPhysics authors

October 31, 2018

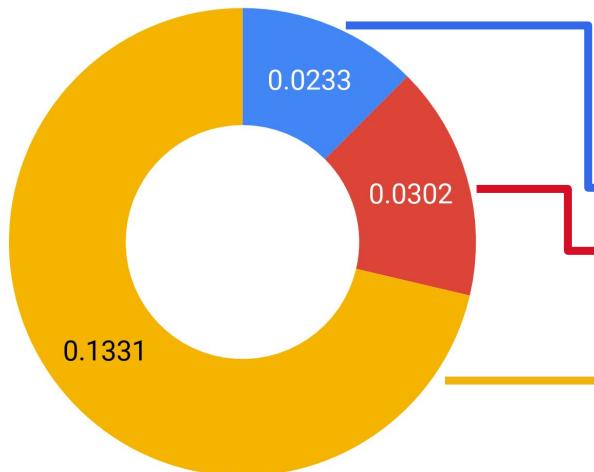
Abstract

This document describes the performance of the optimized algorithms developed for application to the classification and regression problems of WP1.

RESULTS SUMMARY

• The study resulted in a method which:	Solution	New	<u>1st place</u>
<ul style="list-style-type: none">Outperformed the winning solutionTrained 16 times quickerInferred 1.5 times quickerRan on a mid-budget laptop as opposed to a top-end graphics card	Method	10 DNNs	70 DNNs
	Train time	1.5 hours	24 hours
	Inference time	40 min	1 hour
	Score	3.818	3.806
	Hardware requirements	Intel i7-6500U <8 GB RAM (2016 laptop)	Titan GPU <24 GB RAM

IMPROVEMENT BREAKDOWN



- Initial model: ensemble of 10 ReLU based DNNs - score = 3.631
- Swish activation function → 3.654
- Cosine learning rate annealing with warm restarts → 3.6846
- Train-time/Test-time data augmentation → 3.818

SOFTWARE REQUIREMENTS

- The DNNs were implemented using Keras + Tensorflow
- However required functionality was missing, e.g.:
 - Swish activation function
 - [Learning Rate finder](#)
 - Learning rate schedules
 - HEP-specific data augmentation
 - Advanced ensembling
- The fastai library contains some of this functionality, but doesn't support sample weighting - necessary for HEP data

SOFTWARE REQUIREMENTS

- A lot of extra code had to be written to provide the necessary functionality
- However this sometimes required working with Tensorflow, which Keras was meant to be abstracting
- Similarly, other Keras functions weren't being used as intended due to data augmentation and the ensembling techniques
- Afterward the study I rewrote the entire framework from scratch using PyTorch for the networks

LUMIN

Lumin Unifies Many Improvements to Networks

OVERVIEW

- At its heart **LUMIN** contains the required functionality to reimplement the Higgs ML solution
- But it does so in an adaptable way, meaning it can easily be applied to other problems and tasks
- On top of this, it provides useful tools for both:
 - Machine learning & data science
 - HEP-specific tasks and evaluations
- LUMIN aims to become an ecosystem for physicists to quickly apply the best approaches to their analysis work

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

deep-learning machine-learning physics science statistics hep Manage topics

83 commits 2 branches 2 releases 1 contributor Apache-2.0

Branch: master New pull request Create new file Upload files Find File Clone or download ▾

GilesStrong Adding env note Latest commit ef7637a 7 days ago

.vscode hep_proc check 2 months ago

examples Moving to beta 7 days ago

lumin Moving to beta 7 days ago

.gitignore removing large files 2 months ago

LICENSE partials 3 months ago

MANIFEST.in Install stuff 2 months ago

README.md Adding env note 7 days ago

abbr.md all check 2 months ago

requirements.txt Preparing PyPI release, adding missing init 8 days ago

setup.cfg Install stuff 2 months ago

setup.py Moving to beta 7 days ago

README.md

pypi v0.1.0 python 3.6 | 3.7 license Apache Software License 2.0 DOI 10.5281/zenodo.2601858

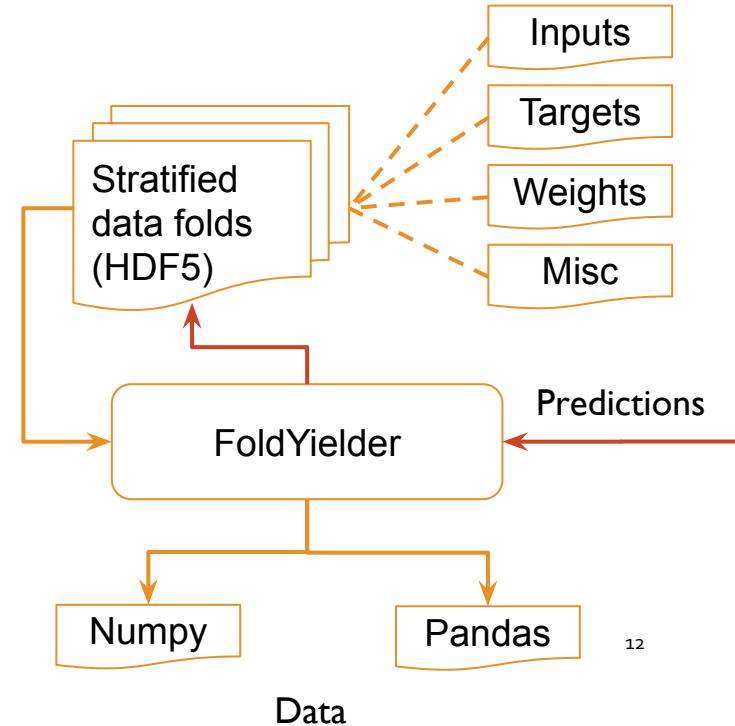
LUMIN: Lumin Unifies Many Improvements for Networks¹⁰

CAPABILITIES

- LUMIN can be used to train neural networks for supervised classification and regression tasks using columnar data (features in columns - events in rows)
- Many options available ([full list](#)), such as:
 - Learning rate & momentum schedules; e.g. [cosine annealing](#), [One-cycle](#)
 - Advanced ensembling; e.g. [Snapshots](#), [Fast-Geometric](#), [Stochastic Weight Averaging](#)
 - [Entity embedding](#) of categorical features
 - See backup slides for summary of techniques
- Feature selection via importance, dependency, and dendograms
- Universal handling of sample weights

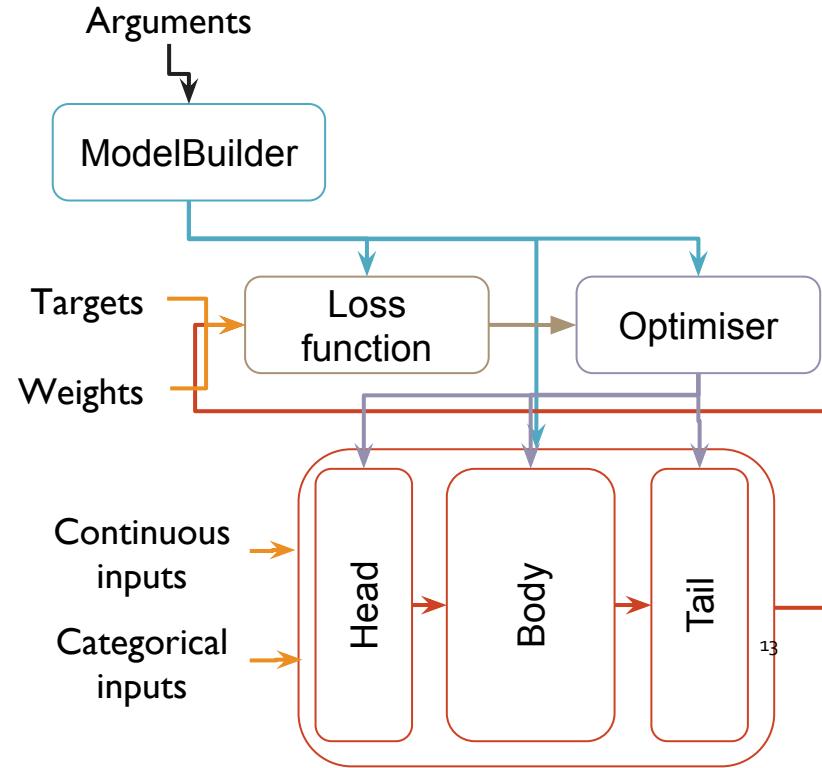
CENTRAL CLASSES: FoldYielder

- N-tuples extracted, processed, and saved to HDF5 file
 - [General](#) and [HEP-specific](#) pre-processing functions available
 - Data split into folds for ensembling and cross-validation ([helper functions available](#))
- [FoldYielder](#) interfaces with HDF5 file to provide data on demand as: Numpy arrays or Pandas DataFrames
- [Inheriting classes](#) can provide data augmentation at train and test time
- Many LUMIN functions expect, or act differently on, FoldYielders
 - Memory overhead reduced
 - Averaging over folds



CENTRAL CLASSES: ModelBuilder

- **ModelBuilder** initialised with specification for the NN architecture and optimiser
- Once initialised, will yield NNs and optimisers on request
- NNs consist of three modular sections:
 - **Head** - takes continuous and categorical inputs and scales up the width
 - **Body** - provides most of the computation
 - **Tail** - scales down the body to match the output dimension and provides the final activation layer
- These can be swapped and configured as required



CENTRAL CLASSES: Model

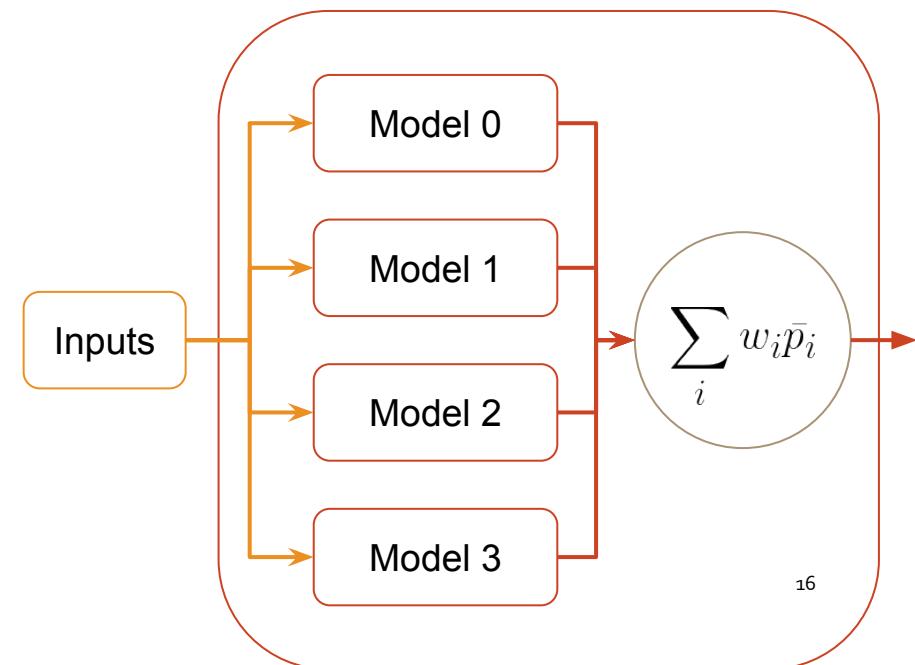
- Model is a wrapper class for the NNs created by ModelBuilders, providing methods for:
 - Fitting
 - Predicting
 - Evaluating
 - Loading/Saving/Exporting
- The recommended process for training is to use the fold_train_ensemble method (even for single models) to interface a FoldYielder with Model.fit
- Once finished, a new Model can be instantiated, and the weights saved during training can be loaded

CENTRAL CLASSES: Callback

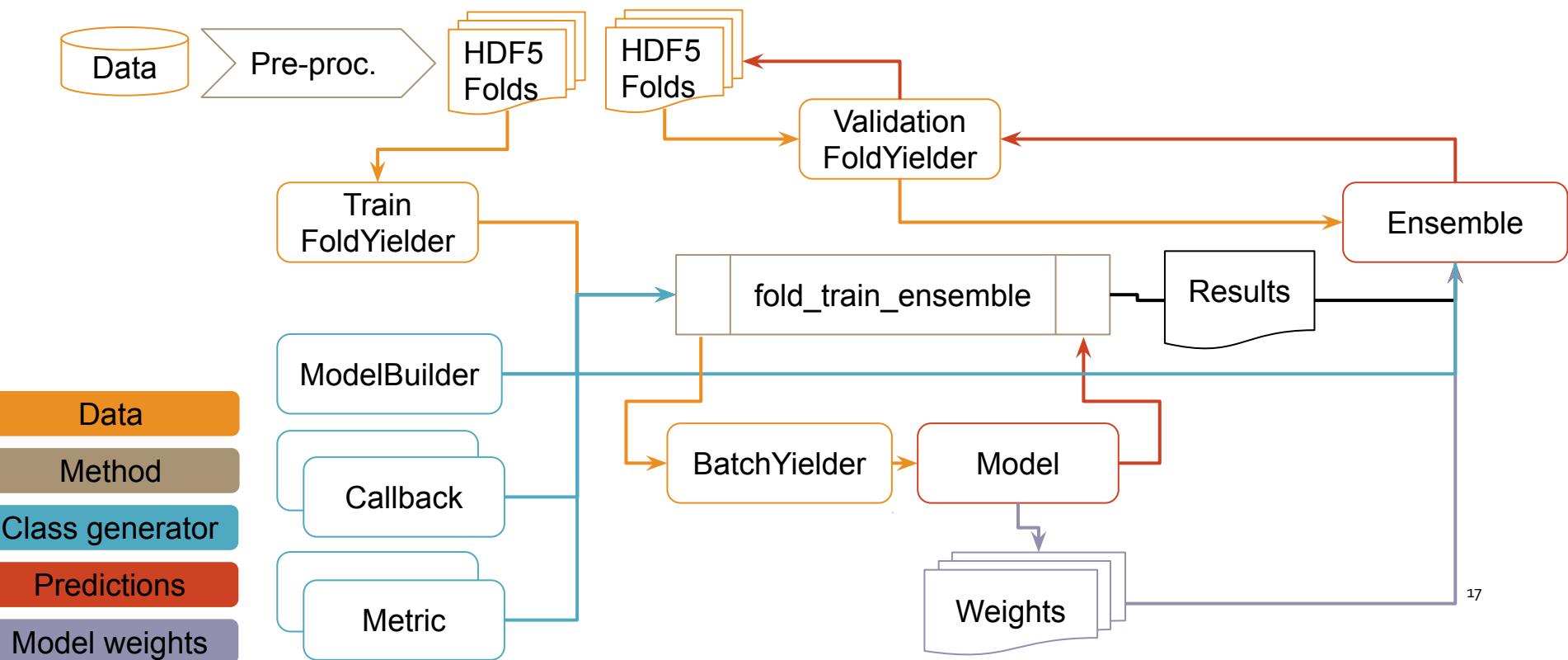
- Similar to Keras, the [Callback](#) class can be used to adjust the training procedure, model, et cetera whilst training is ongoing
- The [fold_train_ensemble](#) method looks for two special types:
 - Subclasses of [AbsModelCallback](#) with a get_loss method will affect the early stopping of training by considering the validation loss of any model variant the callback maybe maintaining, e.g. a [Stochastic Weight Average](#) of models
 - Subclasses of [AbsCyclicalCallback](#), e.g. cosine annealing, will:
 - Change early-stopping patience from epochs without improvement to cycles without improvement
 - Save model weights after every cycle (e.g. for [SSE](#) or [FGE](#))
 - Be passed to any subclasses of AbsModelCallback, e.g so model weights are only averaged at the end of a cycle

CENTRAL CLASSES: Ensemble

- The [Ensemble](#) class loads model trained by [fold_train_ensemble](#) and provides bulk methods for prediction, saving/loading/exporting, and feature importance evaluation
- Individual predictions of models can be weighted by some metric, e.g validation loss, or given equal weighting
- Can also build ensembles from (weighted) snapshots of models for [SSE](#) or [FGE](#)



CLASS INTERACTION



OTHER HIGHLIGHTS: SET DISCRIMINATION

- Uses random forests to quickly see whether one's validation set is indistinguishable from one's training and testing sets
 - If not, highlights which features are most different
- Could also be used to spot mis-modelling of collider data in Monte Carlo samples

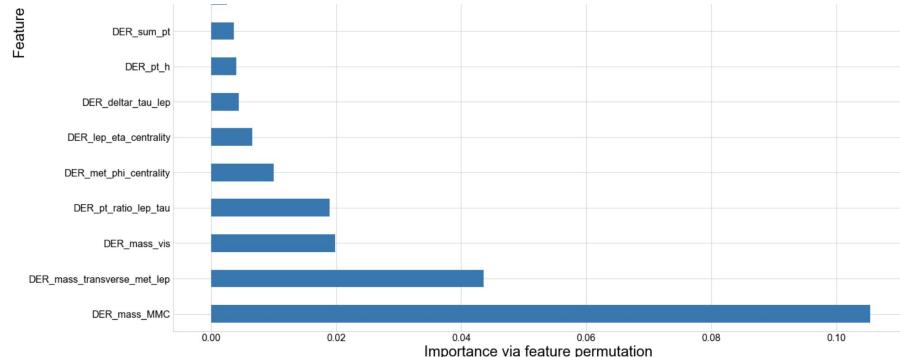
AUC for test-validation discrimination = 0.4998

Top 10 most important features are:

index	Feature	Importance
0	DER_mass_vis	0.001422
1	DER_prodeta_jet_jet	0.001204
2	PRI_lep_py	0.001204
3	PRI_lep_px	0.001204
4	PRI_tau_px	0.001204
5	DER_pt_h	0.001094
6	PRI_lep_pz	0.001094
7	PRI_jet_subleading_px	0.001094
8	DER_lep_eta_centrality	0.000985
9	DER_pt_ratio_lep_tau	0.000328

OTHER HIGHLIGHTS: FEATURE SELECTION

- Uses auto-optimising random forests to check permutation importance of features
 - Retrains on a reduced set of features to check for performance drop when low importance features are removed
 - Permutation importance also available for interpreting NNs and ensembles - allows for sanity checking
- Feature dependency (ease of regression of a feature using other features) also available via rfpimp package
 - Accounts for multicollinearities, which permutation importance would not pick up



```
12 features found with importance greater than 0.001:  
['DER_mass_MMC', 'DER_mass_transverse_met_lep', 'DER_mass_vis', 'DER_pt_ratio_lep_tau', 'DER_met_phi_centrality',  
 'DER_lep_eta_centrality', 'DER_delta_tau_lep', 'DER_pt_h', 'DER_sum_pt', 'DER_deltaeta_jet_jet', 'DER_mass_jet_jet',  
 'PRI_jet_num']
```

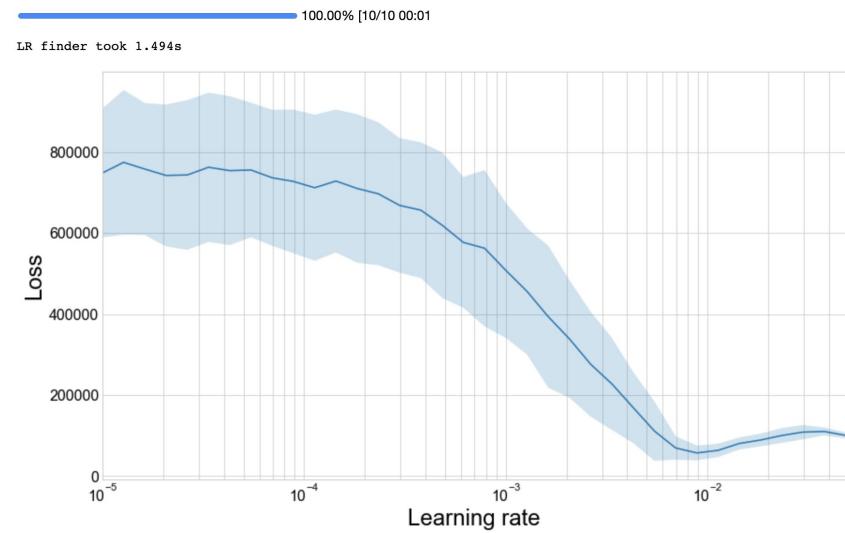
Optimising new RF

Total time: 00:09

```
Comparing RF scores, higher = better  
All features: 0.83233  
Top features: 0.83062
```

OTHER HIGHLIGHTS: LR FINDER

- The [Smith LR range test](#) allows one to quickly find optimal learning rates for a given architecture and dataset
- Passing a FoldYielder will show the mean and standard deviation for a range of trainings = smoother curve



STATE OF LUMIN

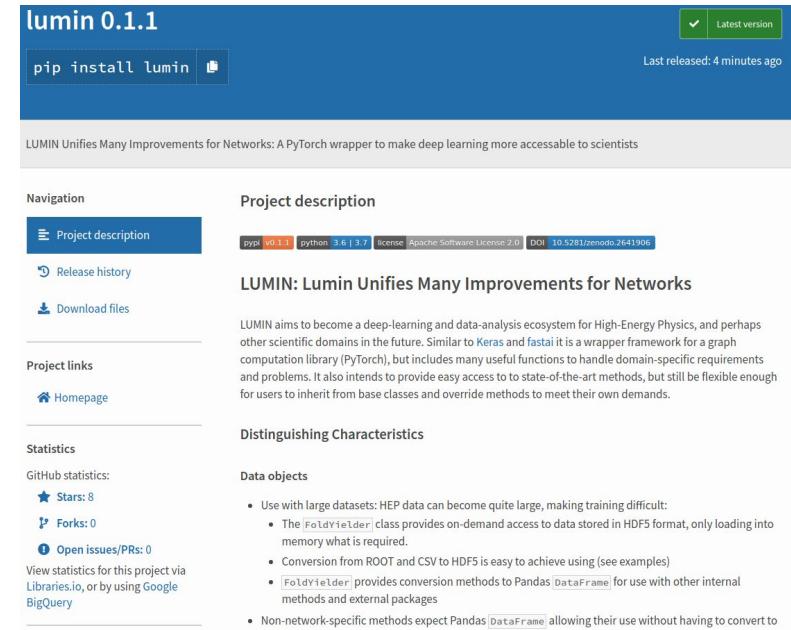
LUMIN ON HIGGS ML

- Currently reconfirming results from original study
- LUMIN appears to reproduce results, but averaging over many runs puts score on par with winning solution
- However, even greater speed up over original code
 - NB, change in test-time augmentation means inference time should timesed by 4 to compare to original times

Solution	LUMIN-GPU	LUMIN-CPU	<u>1st place</u>
Method	10 DNNs	70 DNNs	
Train time	20 minutes	40 minutes	24 hours
Inference time	15 seconds	130 seconds	1 hour
Score	3.806 ± 0.002	3.806	
Hardware	Nvidia 1080 Ti GPU	Intel Core i7-8559U (MacBook Pro 2018)	Nvidia Titan GPU

CURRENT STATUS

- LUMIN is stored on a public Github repo:
<https://github.com/GilesStrong/lumin>
- Installable from source or PIP:
<https://pypi.org/project/lumin/>
 - Both methods require manual installation of additional, independent packages not on PIP
- Still in beta; latest release is v0.1.1
- OSX and Linux are supported
- No documentation or unit tests yet
- [Four examples](#) showing most of the capabilities:
 - Binary & multiclass classification
 - Single & multi target regression



The screenshot shows the PyPI project page for 'lumin 0.1.1'. At the top right, there's a green button labeled 'Latest version' with a checkmark. Below it, a message says 'Last released: 4 minutes ago'. The main heading is 'lumin 0.1.1'. A search bar contains the command 'pip install lumin'. A grey banner below the header states 'LUMIN Unifies Many Improvements for Networks: A PyTorch wrapper to make deep learning more accessible to scientists'. On the left, a sidebar has sections for 'Navigation' (Project description, Release history, Download files), 'Project links' (Homepage), and 'Statistics' (GitHub statistics: Stars: 8, Forks: 0, Open issues/PRs: 0). The main content area has a section titled 'Project description' with details: 'python 3.6 | 3.7', 'license Apache Software License 2.0', and 'DOI 10.5281/zenodo.2641906'. Below this is a section titled 'LUMIN: Lumin Unifies Many Improvements for Networks' which describes the project's aim to become a deep-learning and data-analysis ecosystem for High-Energy Physics and other scientific domains. It mentions using PyTorch as a graph computation library and includes many useful functions for domain-specific requirements and problems. The page also lists 'Distinguishing Characteristics' and 'Data objects'.

FUTURE

- Documentation is a necessary step
- As the number of users and contributors (hopefully) grows, tests will need to be added; currently the examples are doubling as tests.
- Really, though, the package needs people trying it out, playing around, and giving feedback on:
 - Bugs
 - Design & layout choices
 - General suggestions
- Contributions are most welcome: I'm hoping this can become a more general tool for the experimental-science community, but am only able to²⁴ really cover my research areas

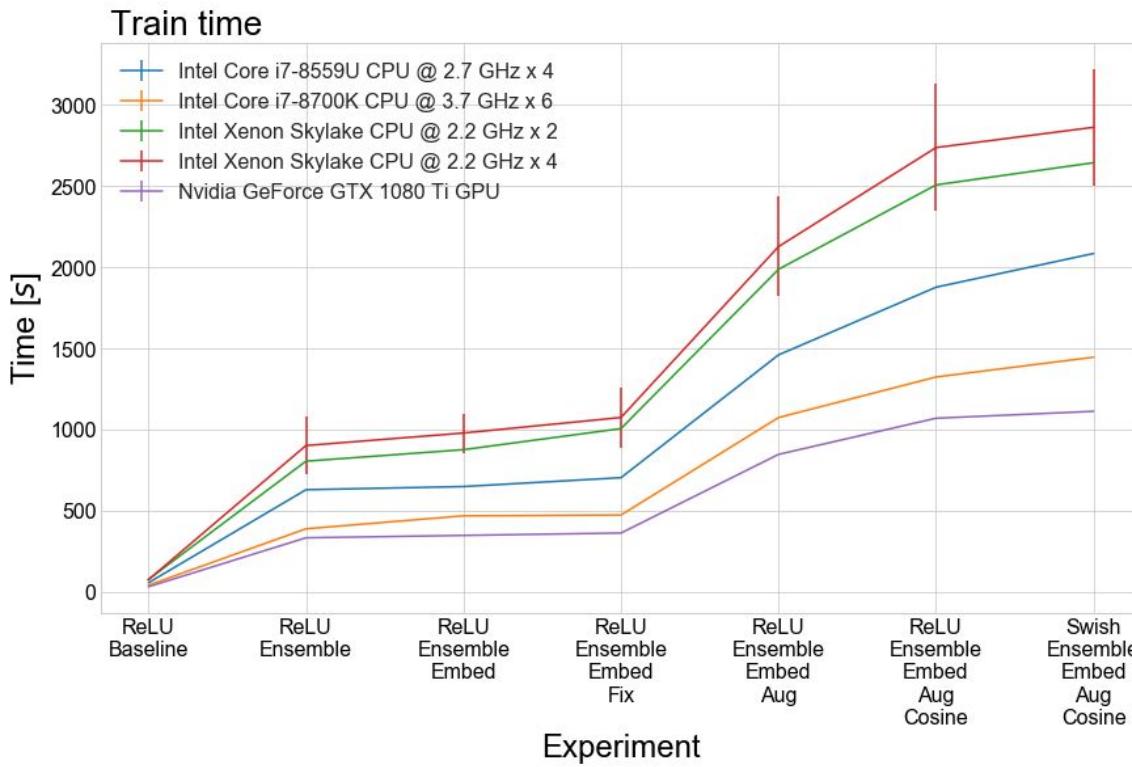


This Report is part of a project that has received funding from the **European Union's Horizon 2020 research and innovation programme under grant agreement N°675440**

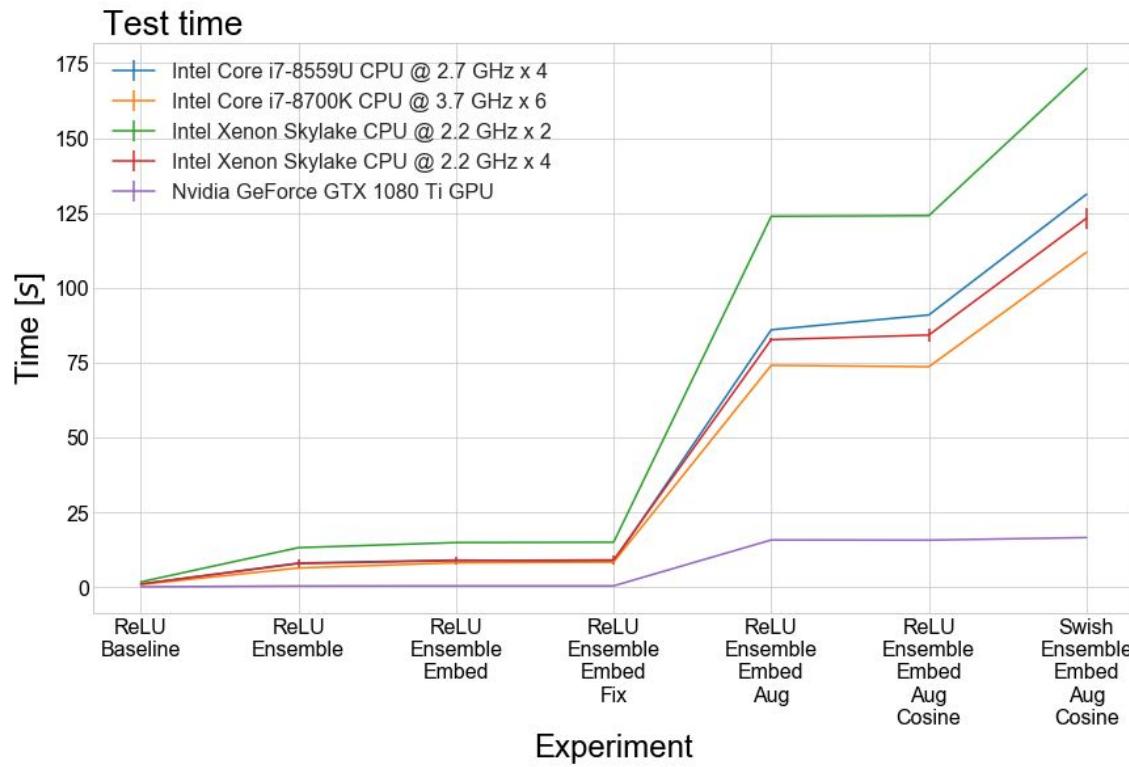
BACKUP SLIDES

LUMIN TIMING

TRAIN TIME

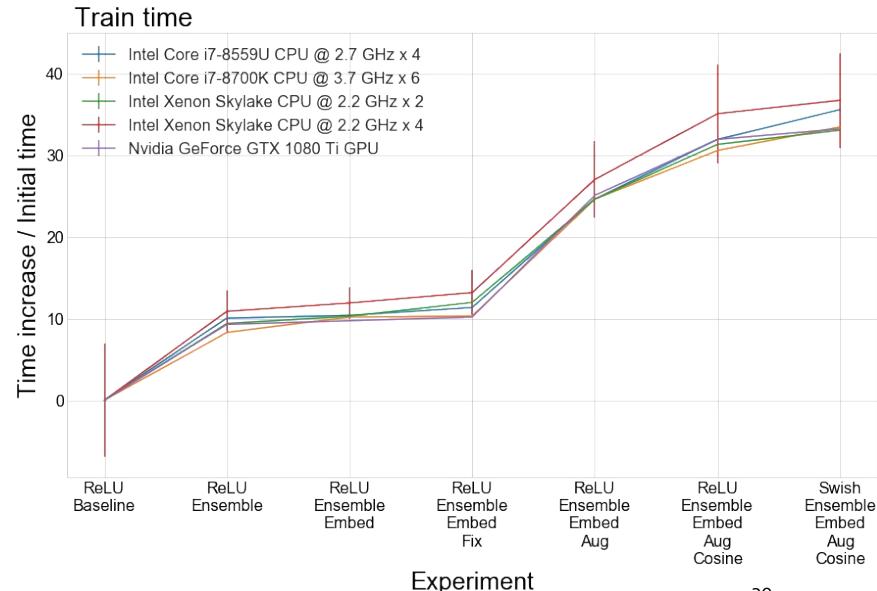


TEST TIME



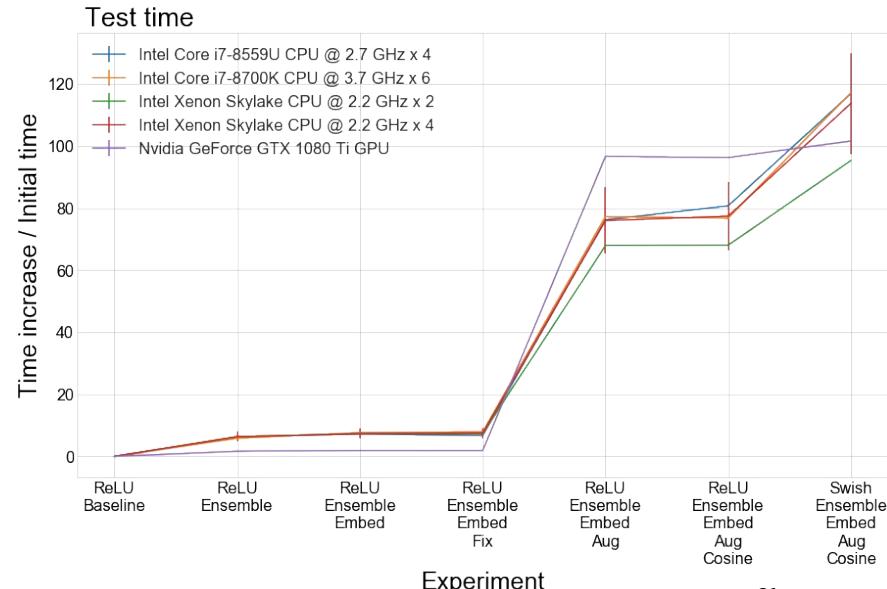
RELATIVE TRAIN-TIME INCREASE

- All additions bring about a 35 times increase in train time
- Accounting for ensemble training, 3.5 times increase



RELATIVE TEST-TIME INCREASE

- All additions bring about a 110 times increase in inference time
- Accounting for ensemble training, ~11 times increase
- Increases on par with expectation
 - Just under 10 times for ensembling
 - NB, timing includes time for data loading - hence why GPU appears to show super-sublinear scaling for ensembling
 - ~8 times for test-time augmentation (8 sets of augmentations are applied)
 - Increase for Swish due to exponential evaluation (and lack of JIT compilation?)



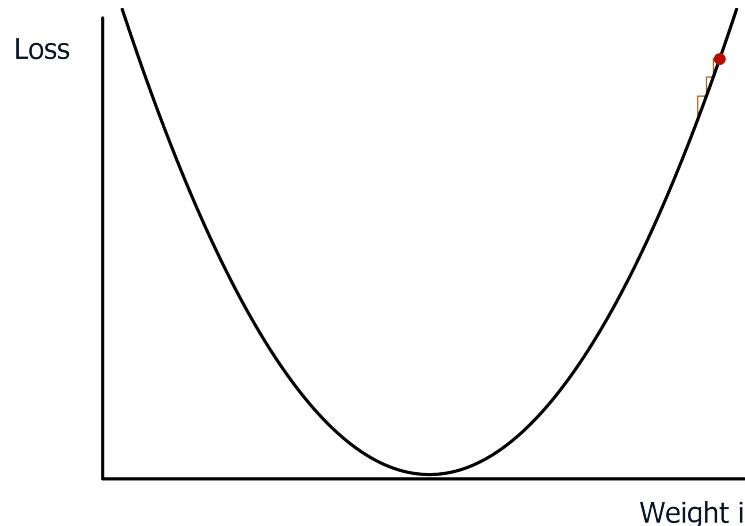
METHOD SUMMARIES

Learning rate finder

- “[The Learning Rate] is often the single most important hyperparameter and one should always make sure that it has been tuned” - Bengio, [2012](#)
- Previously this required running several different trainings using a range of LRs
- The LR range test (Smith [2015](#) & [2018](#)) can quickly find the optimum LR using a single epoch of training

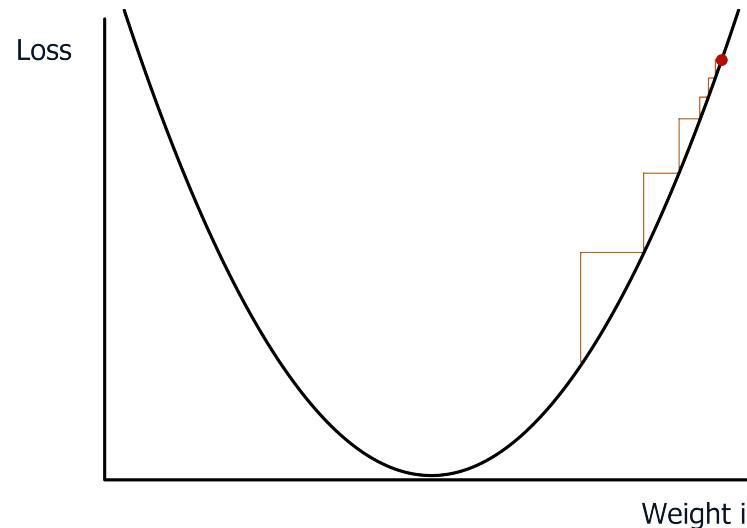
Learning rate finder

- I. Starting from a tiny LR ($\sim 1e-7$), the LR is gradually increased after each minibatch



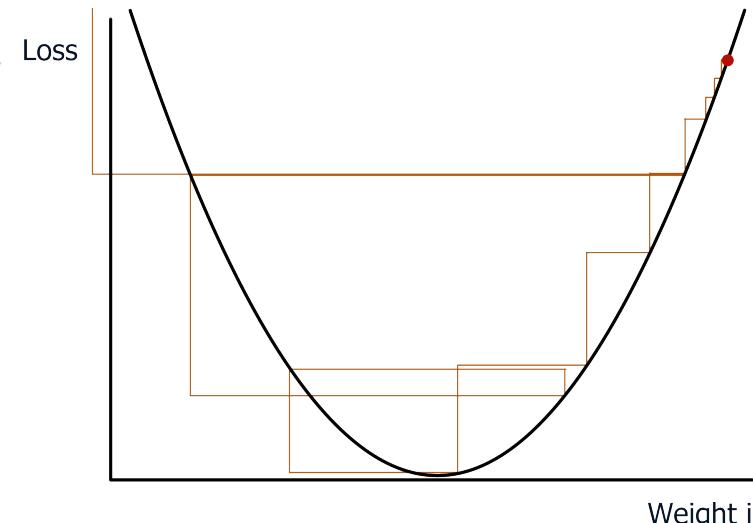
Learning rate finder

1. Starting from a tiny LR ($\sim 1e-7$),
the LR is gradually increased after
each minibatch
2. Eventually the network starts
training (loss decreases)



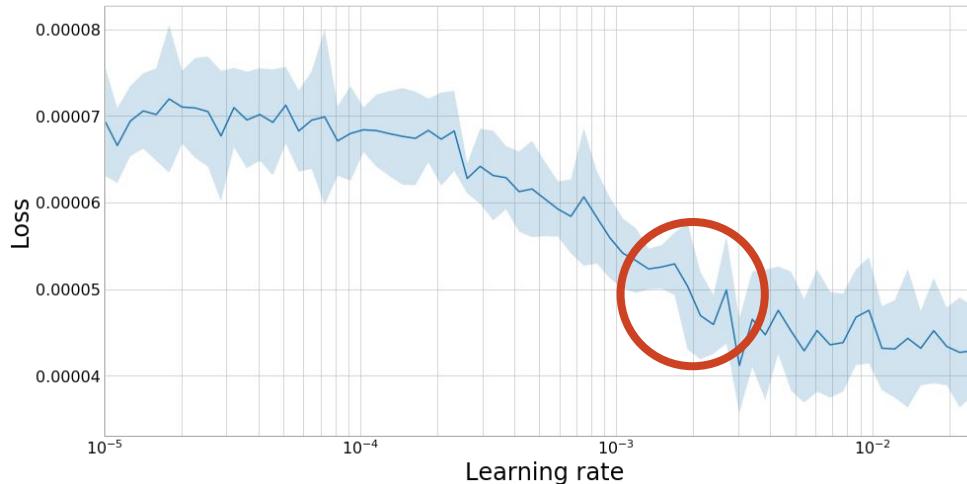
Learning rate finder

1. Starting from a tiny LR ($\sim 1e-7$), the LR is gradually increased after each minibatch
2. Eventually the network starts training (loss decreases)
3. At a higher LR the network can no longer train (loss plateaus), and eventually the network diverges (loss increases)



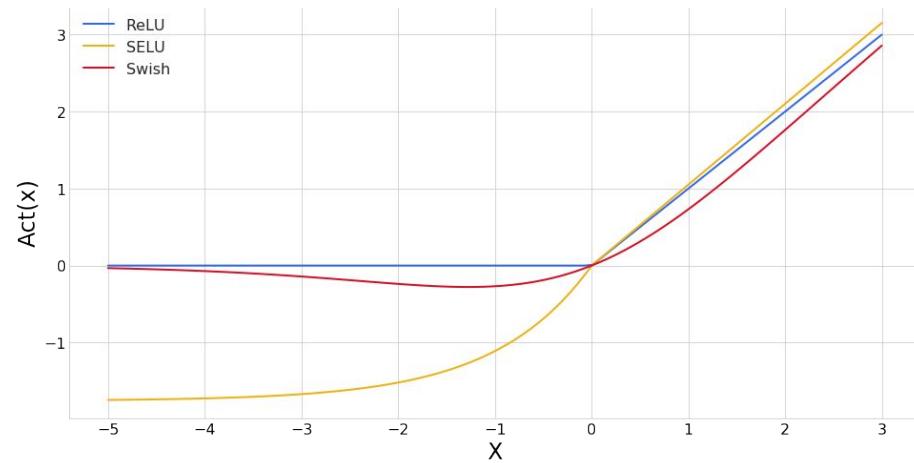
Learning rate finder

- The optimum LR is the highest LR at which the loss is still decreasing
- Further explanation in this [lesson](#)



ACTIVATION FUNCTIONS

- Whilst ReLU is a common activation function, newer ones are continually being introduced
- SELU uses carefully derived scaling coefficients allow networks to self-normalise, removing any need for batch normalisation
 - SELU Example
- Swish, found via reinforcement learning, provides a region of negative gradient
 - Swish example



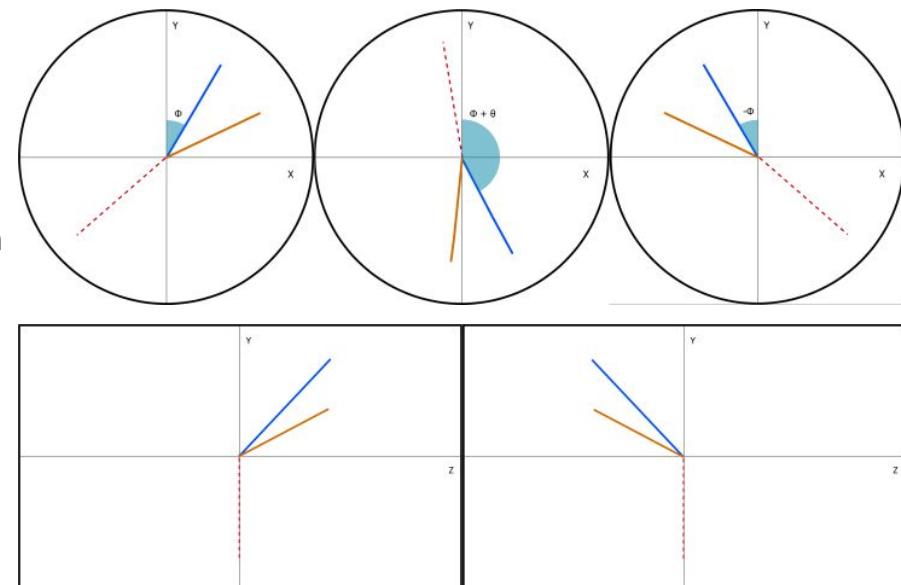
Data augmentation

- Data augmentation involves applying transformations to input data such that a new data point is created, but the underlying class is unchanged
- This is well used in image classification to artificially increase the amount of training data (train-time augmentation), e.g Krizhevsky et al. [2012](#)
- It can also be applied at test time by predicting the class of a range of augmented data and then taking an average of the predictions.



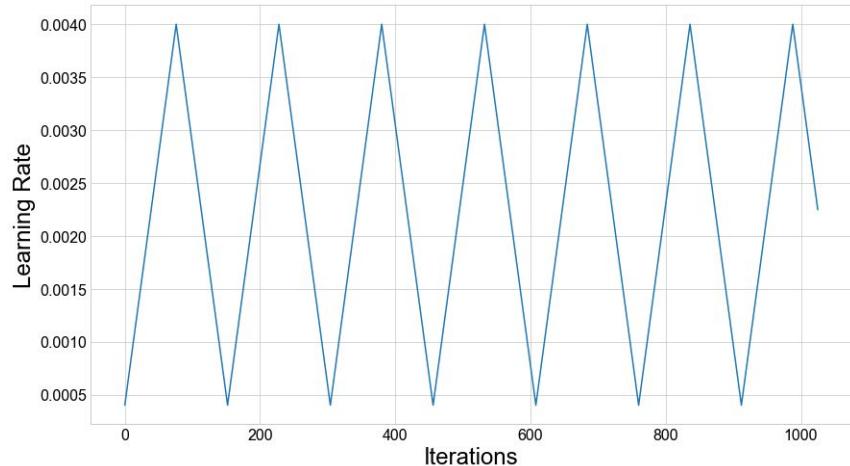
Data augmentation

- Correct application of augmentation relies on exploiting invariances within the data: domain specific
- At the CMS and ATLAS detectors, the initial transverse momentum is zero, therefore final states are produced isotropically in the transverse plane: the class of process is invariant to the rotation in azimuthal angle
- Similarly, the beams collide head on with equal energy: therefore final states are produced isotropically in Z-axis
- Alternative is to remove symmetries by setting common alignment for events
 - [Data augmentation example](#)
 - [Data fixing example](#)



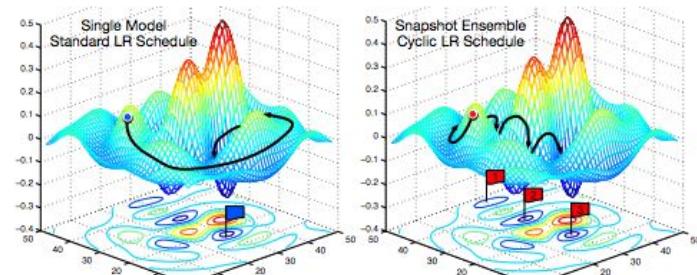
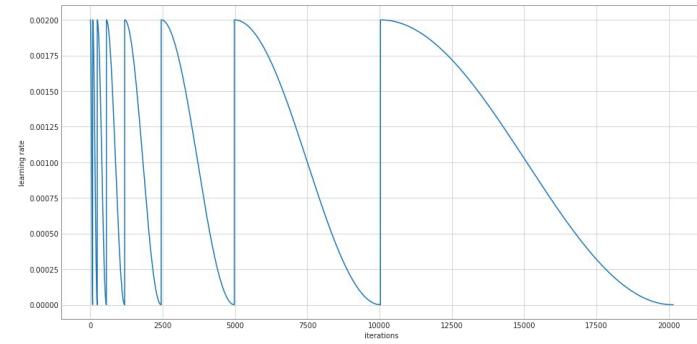
LINEAR CYCLING

- Adjusting the LR during training is a common technique for achieving better performance
- Normally this involves decreasing the LR once the validation loss becomes flat
- Smith [2015](#) suggests instead to cycle the LR between high and low bounds
- Smith [2017](#) showed that the additional application of an out-of-phase schedule to the optimiser momentum/beta₁ coefficient can sometimes lead to *super convergence*
- [Linear cycle example](#)



COSINE ANNEALING

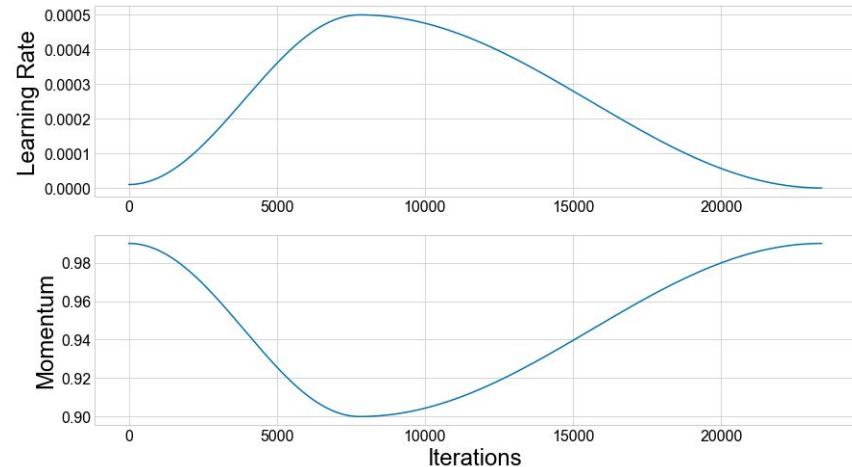
- Loshchilov and Hutter [2016](#) instead suggests that the LR should decay as a cosine with the schedule restarting once the LR reaches zero
- Huang et al. [2017](#) later suggests that the discontinuity allows the network to discover multiple minima in the loss surface
- 2016 paper demonstrates on image and EEG classification
- [Cosine annealing example](#)



42

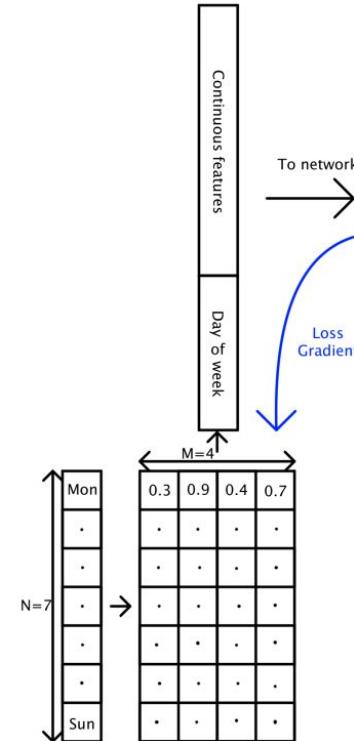
ONE CYCLING

- Smith [2018](#) introduces the 1cycle schedule which further improves the super convergence
- This involves running through a single cycle of increasing and then decreasing the LR, with a similar, inverted schedule applied to momentum/beta₁
- Original paper used linear interpolation
- FastAI found a cosine interpolation was better
- LUMIN includes both interpolations
- [Onecycle example](#)



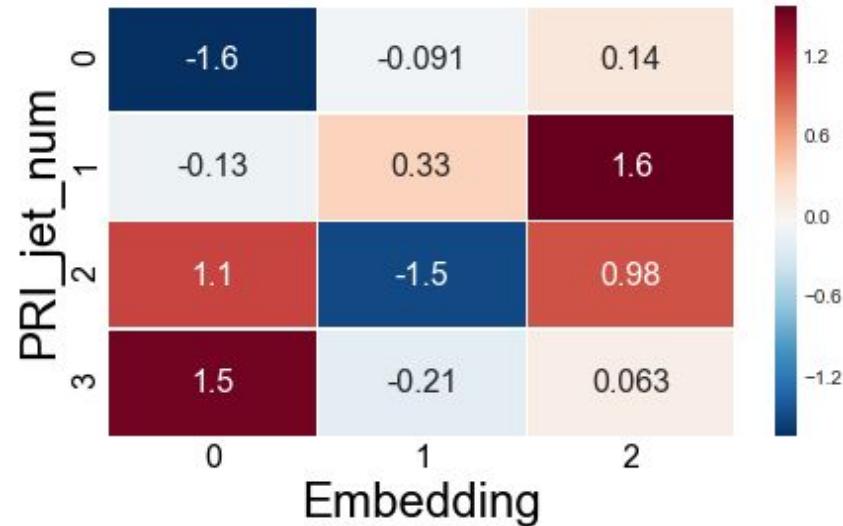
ENTITY EMBEDDING

- Categorical features = features with discrete values and no numerical comparison
- Normal to 1-hot encode as Boolean vector (Monday → 1000000)
- But potentially means a large number of extra inputs to NN (day of year = 365 inputs)
- Guo 2016 learns lookup tables which provide a compact, but rich, representation of categorical values as vector of floats (Monday → 0.3,0.9,0.4,0.7)



ENTITY EMBEDDING

- Embedding values start from random initialisation
- Receive gradient during backpropagation and are learnt just like any other network parameter
- Once learnt, embeddings can then be transferred to other tasks which require embedding the same, or similar, feature (better than random init.)
- [Random init. Example](#)
- [Embedding loading example](#)



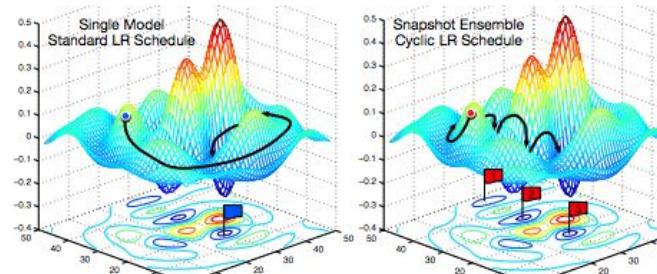
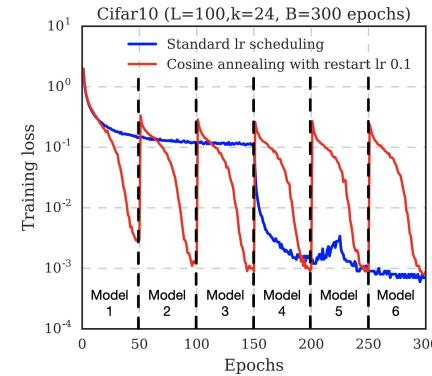
ENSEMBLING

- Basic ensembling requires training multiple models, and then averaging over their predictions
- Advantages:
 - Greater generalisation to unseen data
 - Often a more powerful model
 - Provides resistance against possibility of bad training
- Disadvantages
 - Increases train time
 - Increases inference time

Basic ensembling example

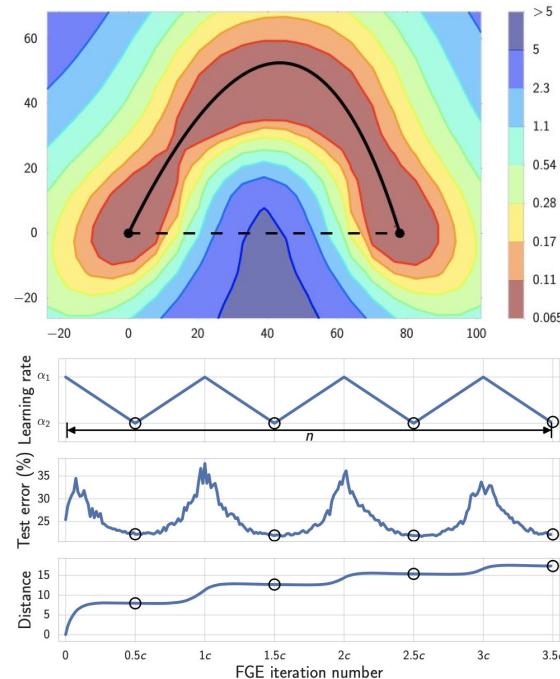
SNAPSHOT ENSEMBLING

- SSE uses the idea that a warm-restarted LR will find multiple minima across the loss surface
- Just before the LR jump, when the network is in a minima, a snapshot of the model weights is saved
- The model then jumps out and begins converging to different minima
- This allows one to create an ensemble of models in a single training



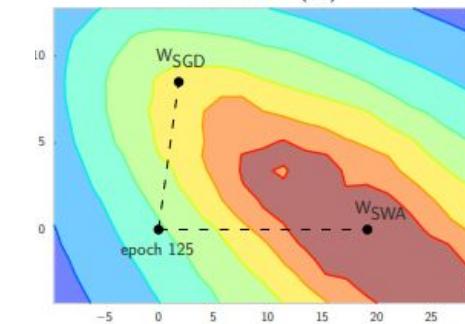
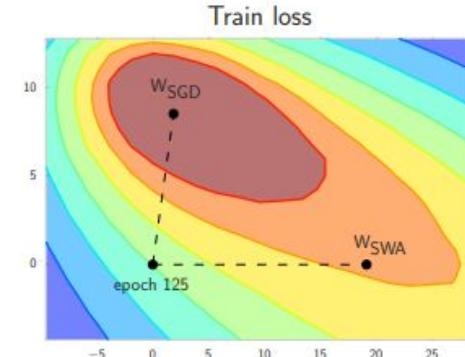
FAST GEOMETRIC ENSEMBLING

- FGE further improves on SSE by finding that loss minima are connected by curves of constant loss (rare to find true minima - almost certainly a saddlepoint/valley given dimensionality)
- Using a high-frequency linearly cycled LR it aims to direct the model along these curves, saving snapshots along the way
- FGE example



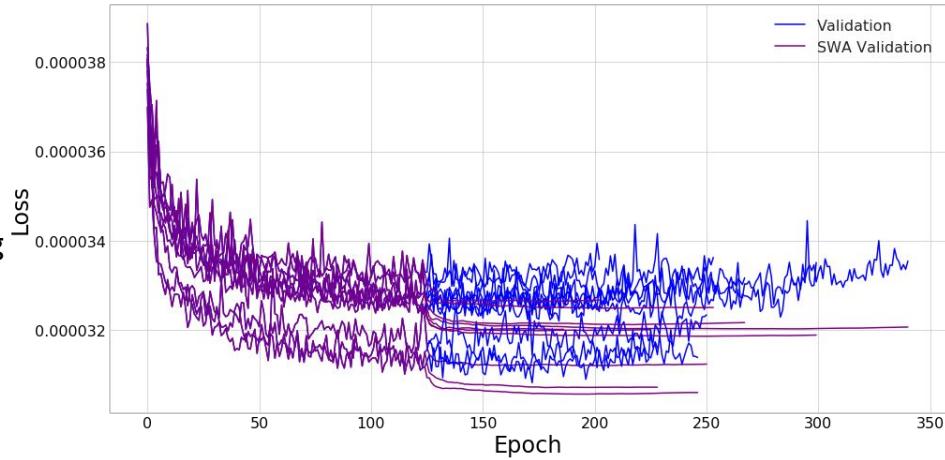
STOCHASTIC WEIGHT AVERAGING

- Both SSE and FGE solve the problem of increased training time, but still incur an increased inference time
- SWA solves this problem by averaging in *weight space* rather than *model space*
- As the model is trained, a running average of the model parameters is kept
- Can be used with both constant, cycled, and annealed LR schedules
- SWA example



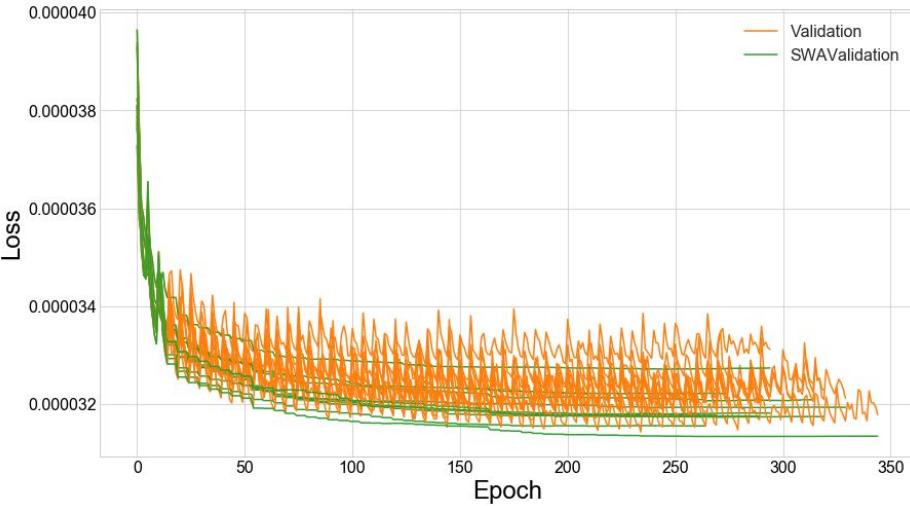
STOCHASTIC WEIGHT AVERAGING

- Published SWA requires setting a priori when to start averaging
 - Too early = spoilt by initial weights being bad
 - Too late = no exploration



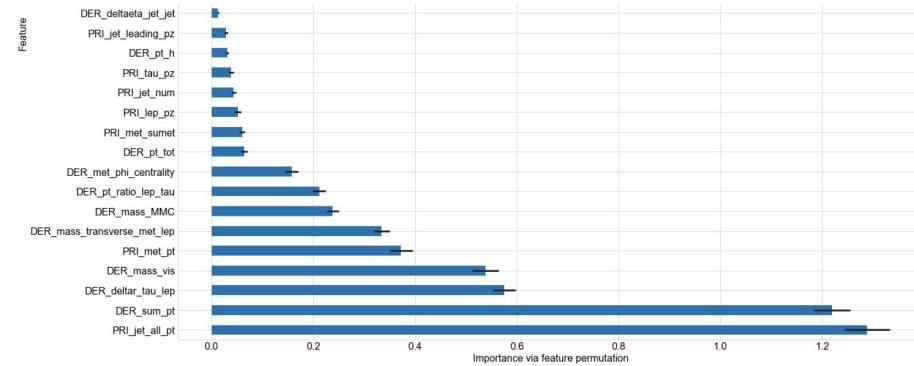
STOCHASTIC WEIGHT AVERAGING

- LUMIN's implementation can track multiple averages, replacing and restarting them if new averages would be better
- Slightly slower, but ensures you never need to train once beforehand to know when to start averaging



PERMUTATION IMPORTANCE

- Method to quantify the importance of each input feature
 - Loss of trained mode evaluated on training data
 - Copy of data made and one feature column is shuffled = info destroyed
 - Loss re-evaluated
 - Increase in loss corresponds to feature importance - Large change implies heavy reliance by model
- Can either be used interpret a trained model, or to reduce input feature-space
- Does not work for collinear features - model can recover information from another feature, so importance is reported to be lower than it actually is



FEATURE DEPENDENCE

- For data with collinear feature, feature dependence can be evaluated
- This involves training regressors to predict the value of a certain feature given the other features
- The more performant the regressor, the more information about that feature is carried by the other features
- Can be used to *carefully* remove training features
- See this for more information:
<https://explained.ai/rf-importance/index.html>

