DEEP LEARNING IN HIGGS PHYSICS

Giles Strong

Supervised by Michele Gallinaro & João Varela

LIP PhD Students' Workshop, Minho University - 01/07/19

giles.strong@outlook.com

twitter.com/Giles_C_Strong

Amva4newphysics.wordpress.com

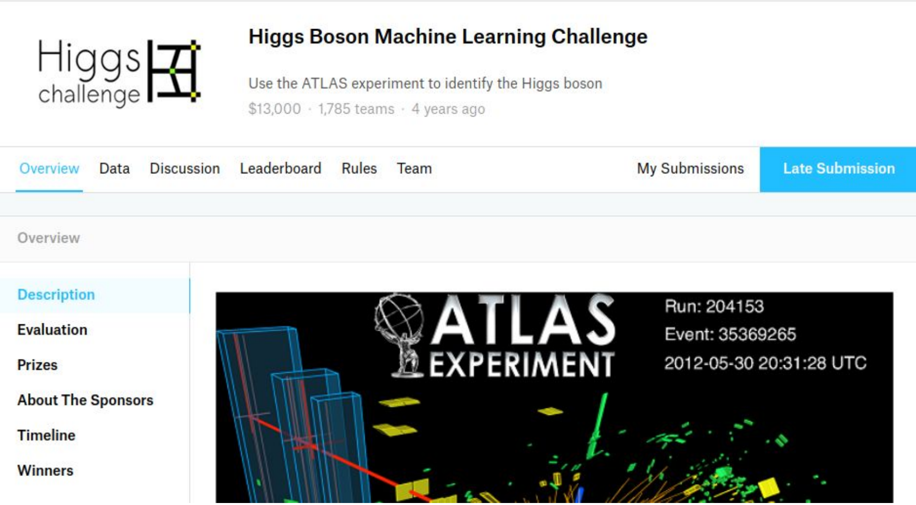
github.com/GilesStrong

ML IN HEP & ML INNOVATION

- In recent years, ML innovation in HEP has been growing to solve our domain-specific problems
  - E.g. Object reconstruction, detector simulation, particle ID
- Although these problems are domain specific, their solutions normally rely on applying and adapting techniques developed outside of HEP
- These techniques are continually being refreshed and updated, and are normally presented on benchmark datasets for some specific task
  - It is not always obvious whether they are appropriate for use in HEP

HIGGS ML KAGGLE CHALLENGE

- Launched in 2014, the Higgs ML Kaggle competition was designed to help stimulate outside interest in HEP problems
- The data contains simulated LHC collision data for Higgs to di-tau and several background processes
  - I.e. a typical HEP search analysis
- Realistic metric and data size
- Strong baselines from both physicists and data scientists
- A good public dataset for testing out improvement impacts



Higgs Boson Machine Learning Challenge

Use the ATLAS experiment to identify the Higgs boson

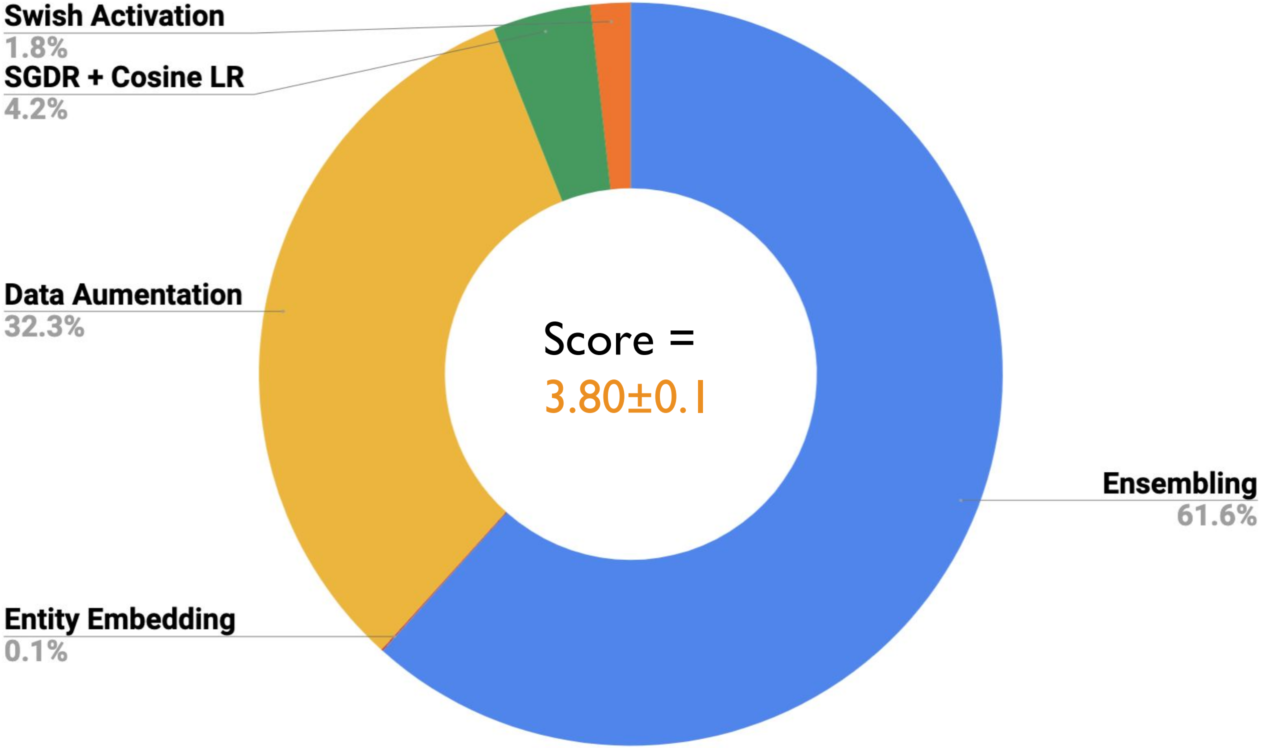
$13,000 · 1,785 teams · 4 years ago

IMPROVEMENT BREAKDOWN

Base Model

Score = 3.38±0.02

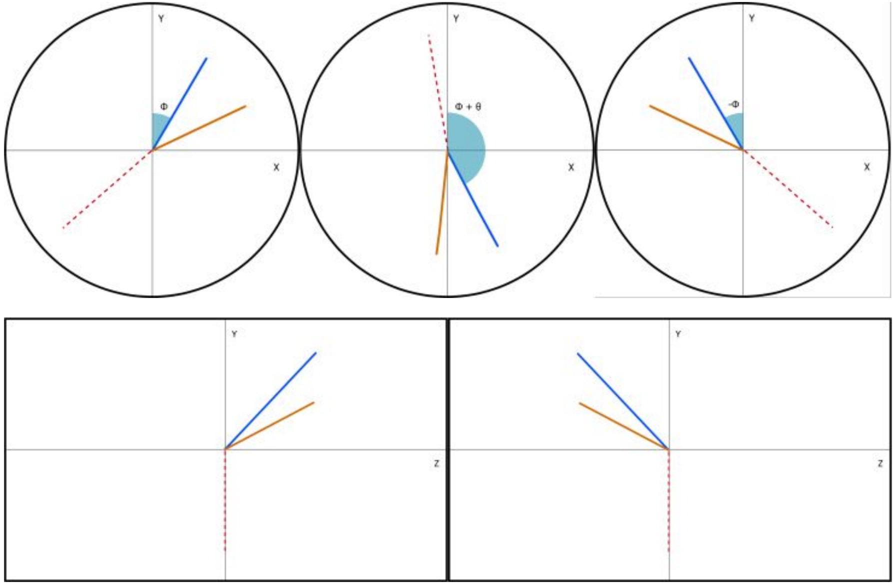
- 4x100 NN
- ReLU
- Constant LR
- ADAM
- He init.



Score = 3.80±0.1

DATA AUGMENTATION FOR HEP

- 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

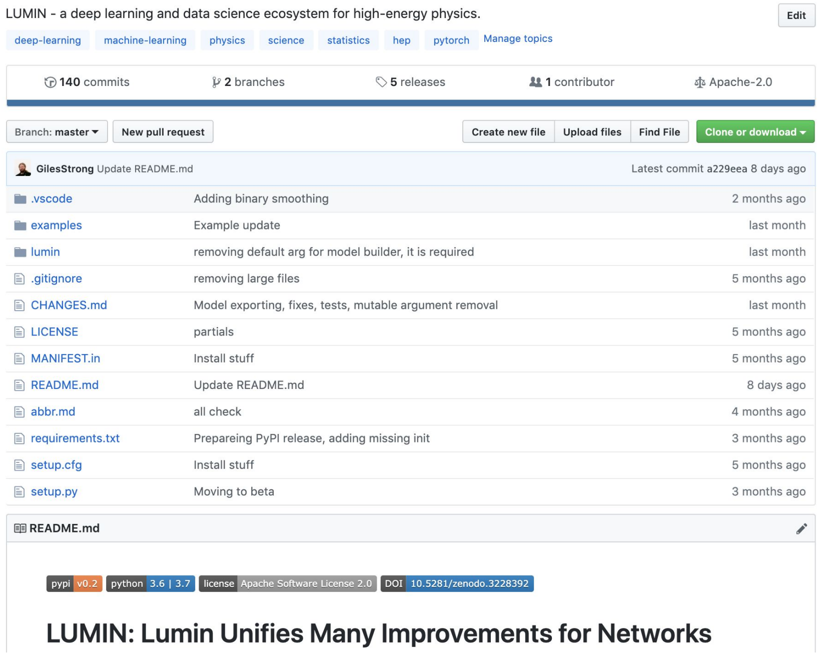


SOFTWARE REQUIREMENTS

- Original study used Keras, but:
  - 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
  - Ensembling also difficult without dedicated classes
- Afterward the original study I rewrote the entire framework from scratch using PyTorch for the networks

LUMIN

- 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.

GilesStrong Update README.md — Latest commit a229eea 8 days ago

.vscode	Adding binary smoothing	2 months ago
examples	Example update	last month
lumin	removing default arg for model builder, it is required	last month
.gitignore	removing large files	5 months ago
CHANGES.md	Model exporting, fixes, tests, mutable argument removal	last month
LICENSE	partials	5 months ago
MANIFEST.in	Install stuff	5 months ago
README.md	Update README.md	8 days ago
abbr.md	all check	4 months ago
requirements.txt	Prepareing PyPI release, adding missing init	3 months ago
setup.cfg	Install stuff	5 months ago
setup.py	Moving to beta	3 months ago

README.md

pypi v0.2 | python 3.6 \| 3.7 | license Apache Software License 2.0 | DOI 10.5281/zenodo.3228392

LUMIN: Lumin Unifies Many Improvements for Networks

LUMIN ON HIGGS ML

- Solution matches winning solution in performance
- On similar hardware:
  - 70 times quicker to train
  - 240 times quicker to apply
- Even a laptop can easily train it
- 1-cycle training schedule can further halve train time with only a minor drop in performance

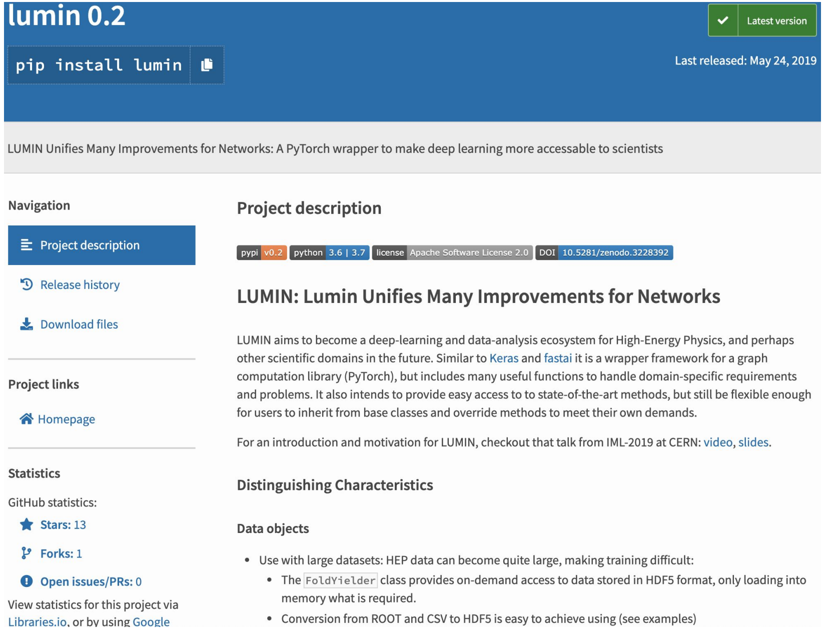
Solution	LUMIN-GPU	LUMIN-CPU	1st place
Method	10 DNNs		70 DNNs
Train time	20 minutes	35 minutes	24 hours
Inference time	15 seconds	120 seconds	1 hour
Score	3.80±0.01		3.806
Hardware	Nvidia 1080 Ti GPU	Intel Core i7-8559U (MacBook Pro 2018)	Nvidia Titan GPU

USE & DISSEMINATION

- Original Higgs ML study documented in AMVA4NewPhysics D1.4
- Higgs ML study repeated using LUMIN, and being re-documented as standalone paper
  - Public presentations: Higgs ML, LUMIN
- Similar solution applied to a di-Higgs HL-LHC projection study
  - CMS-PAS-FTR-18-019, CERN-LPCC-2018-04, & eventual CERN Yellow Report
  - Bengala & Santo (LIP summer students, 2018) found a 30% improvement over a basic NN model
- Currently applying method to CMS di-Higgs search on 2016 & 17 data

LUMIN STATUS

- LUMIN is stored on a public Github repo: https://github.com/GilesStrong/lumin
  - Installable from source or PIP: https://pypi.org/project/lumin/
- Still in beta; latest release is v0.2
- Documentation being written
- Five examples showing most of the capabilities:
  - Classification & regression
  - Export to ONNX & Tensorflow
- Feedback & contributions are most welcome: I'm hoping this can become a more general tool for the experimental-science community



lumin 0.2

```
pip install lumin
```

Last released: May 24, 2019

LUMIN Unifies Many Improvements for Networks: A PyTorch wrapper to make deep learning more accessable to scientists

Navigation

Project links

Statistics

GitHub statistics:

- Stars: 13
- Forks: 1
- Open issues/PRs: 0

View statistics for this project via Libraries.io, or by using Google

Project description

pypi v0.2 | python 3.6 | 3.7 | license Apache Software License 2.0 | DOI 10.5281/zenodo.3228392

LUMIN: Lumin Unifies Many Improvements for Networks

LUMIN aims to become a deep-learning and data-analysis ecosystem for High-Energy Physics, and perhaps other scientific domains in the future. Similar to Keras and fastai it is a wrapper framework for a graph computation library (PyTorch), but includes many useful functions to handle domain-specific requirements and problems. It also intends to provide easy access to to state-of-the-art methods, but still be flexible enough for users to inherit from base classes and override methods to meet their own demands.

For an introduction and motivation for LUMIN, checkout that talk from IML-2019 at CERN: video, slides.

Distinguishing Characteristics

Data objects

- Use with large datasets: HEP data can become quite large, making training difficult:
  - The `FoldYielder` class provides on-demand access to data stored in HDF5 format, only loading into memory what is required.
  - Conversion from ROOT and CSV to HDF5 is easy to achieve using (see examples)

SUMMARY

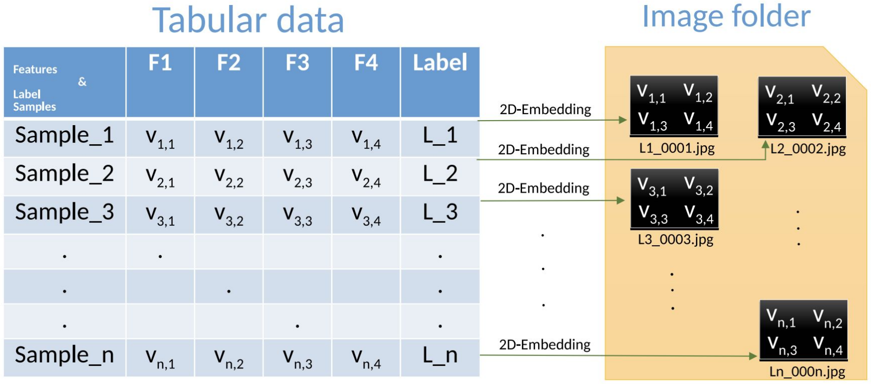
- Have tested many changes to neural network architectures and training & application schemes
- These show large improvements to model performance
- Methodologies are already being applied in CMS analyses
- Package allowing easy implementation of these improvements is now public and continually being developed

EXTRA SECTION: SUPER TML

SuperTML INTRODUCTION

- SuperTML: Two-Dimensional Word Embedding for the Precognition on Structured Tabular Data, March 2019, Sun et al., arxiv.org/abs/1903.06246
- Proposes to transform tabular data into images by printing feature values as text on black backgrounds, either with fixed font-size or varying font-size with feature importance
- Pretrained CNNs (SE-Net 154) (SE-Net 154) can then be refined to perform the desired task as normal
- Claims to get an AMS of 3.979

Can their result be reproduced?



Tabular data

Features & Label Samples	F1	F2	F3	F4	Label
Sample_1	$v_{1,1}$	$v_{1,2}$	$v_{1,3}$	$v_{1,4}$	L_1
Sample_2	$v_{2,1}$	$v_{2,2}$	$v_{2,3}$	$v_{2,4}$	L_2
Sample_3	$v_{3,1}$	$v_{3,2}$	$v_{3,3}$	$v_{3,4}$	L_3
.	.				.
.		.			.
.			.		.
Sample_n	$v_{n,1}$	$v_{n,2}$	$v_{n,3}$	$v_{n,4}$	L_n

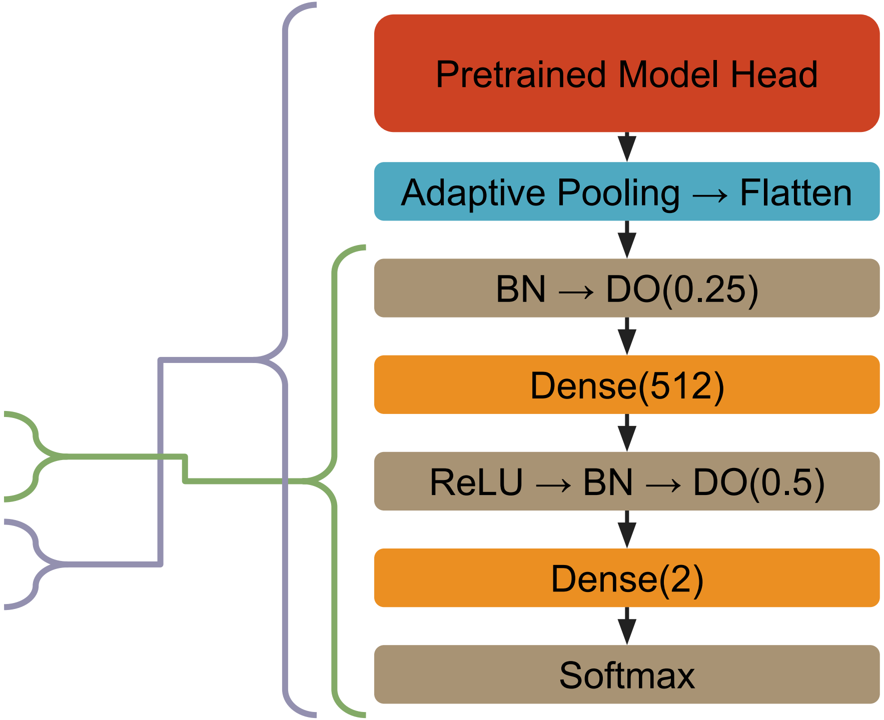
Image folder

2D-Embedding
2D-Embedding
2D-Embedding
2D-Embedding
$v_{1,1}$ $v_{1,2}$ $v_{1,3}$ $v_{1,4}$
L1_0001.jpg
$v_{2,1}$ $v_{2,2}$ $v_{2,3}$ $v_{2,4}$
L2_0002.jpg
$v_{3,1}$ $v_{3,2}$ $v_{3,3}$ $v_{3,4}$
L3_0003.jpg
$v_{n,1}$ $v_{n,2}$ $v_{n,3}$ $v_{n,4}$
Ln_000n.jpg

TML = Tabular Machine Learning

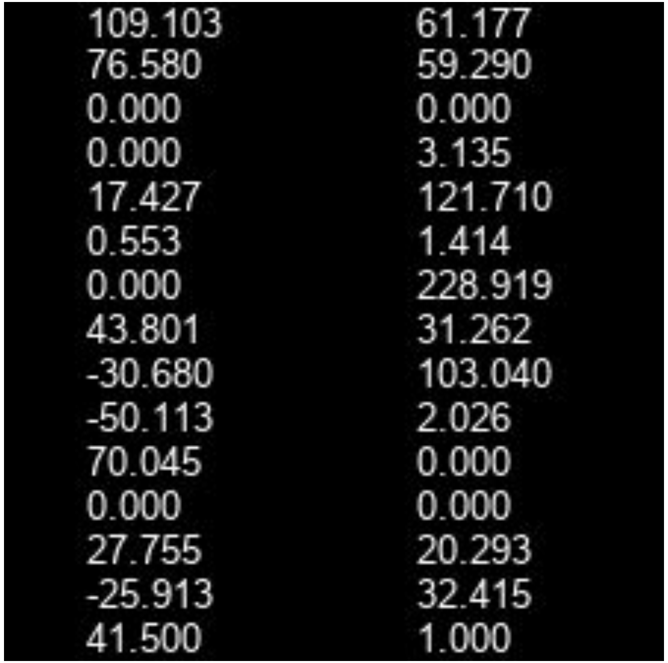
MODEL TRAINING

- Models trained in Fastai (PyTorch wrapper)
- Final dense layer of pretrained model removed and replaced with two dense layers with two output nodes
- Two-step training process:
  - First only the final two dense layers are trained
  - Pretrained head is unfrozen and entire model is trained



INITIAL ATTEMPT

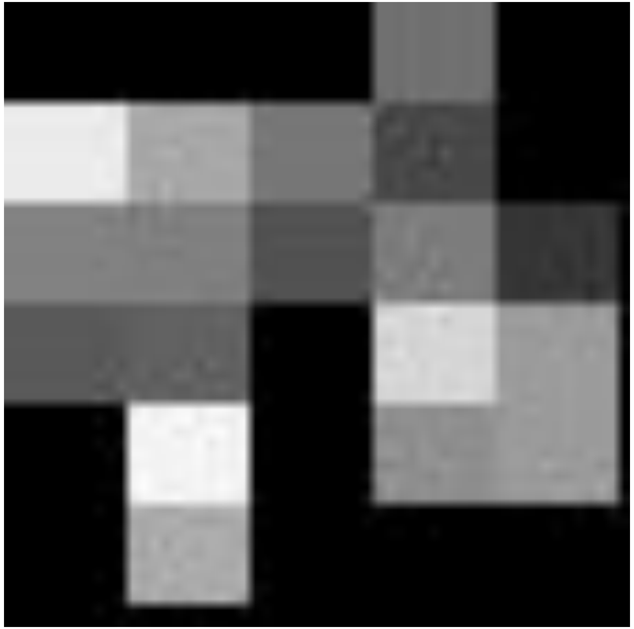
- Settings:
  - 224x224 images - same size paper uses
  - ResNet34 - smaller, simpler model
- Surprisingly, it seemed to learn something
  - Validation AMS around 3
  - Test AMS around 2.75
- Train-time about 47 minutes
  - C.f. 20 minutes for fully-connected approach
- Data size: 13 GB
  - Original data size = 200 MB



```
109.103        61.177
76.580         59.290
0.000          0.000
0.000          3.135
17.427         121.710
0.553          1.414
0.000          228.919
43.801         31.262
-30.680        103.040
-50.113        2.026
70.045         0.000
0.000          0.000
27.755         20.293
-25.913        32.415
41.500         1.000
```

SMALLER IMAGES

- Tried variety of models and encoding schemes
- Latest scheme uses smaller images
  - (56x56) encoded as soid grey-scale blocks
- Data size → 3.1 GB
- Used SE-net 50
- Was able to move to larger batch size (512), train-time → 15 minutes
- Ok performance - better than single ReLU DNN:
  - Validation AMS 3.66
  - Test AMS between 3.3 - 3.5
- Similar performance to SE-Net 154 on 224X224 pixel images, but this takes 12 hours to train



CURRENT STATUS

- Have tried a variety of encoding methods and models and have not been able to recover their 3.979 score (my highest AMS is about 3.5)
  - Have also tested for a few mistakes which the authors may have made
  - Github repo with solutions available here: https://github.com/GilesStrong/SuperTML_HiggsML_Test
  - Complete presentation from recent CMS ML Workshop: https://github.com/GilesStrong/SuperTML_HiggsML_Test/blob/master/presentations/GS_CMSML_SuperTML.pdf
  - Have contacted the authors with a few questions; waiting to hear back
- If the claimed improvement is realised, the method would allow the benefits of transfer learning to be applied to most data analysis work at CERN

FULL SUMMARY

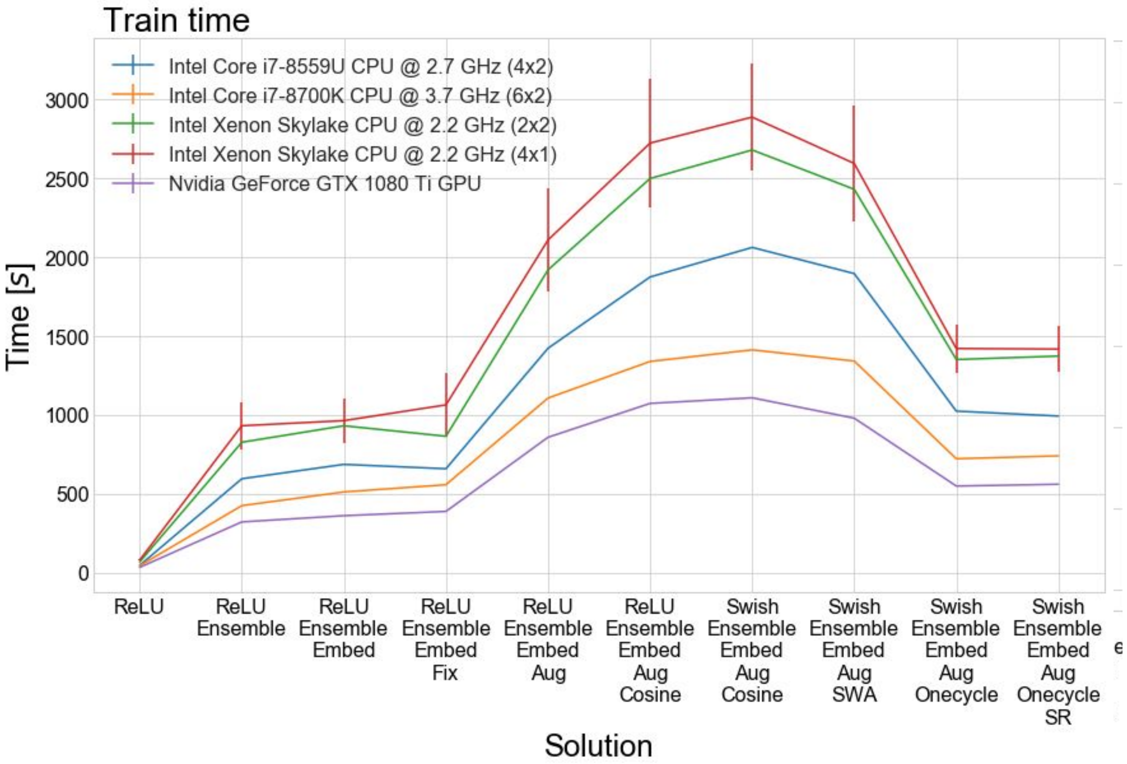
- Have tested many changes to neural network architectures and training & application schemes
- These show large improvements to model performance
- Methodologies are already being applied in CMS analyses
- Package allowing easy implementation of these improvements is now public and continually being developed
- Have investigated potentially powerful method but have not been able to reproduce the claimed results

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 Nº675440**

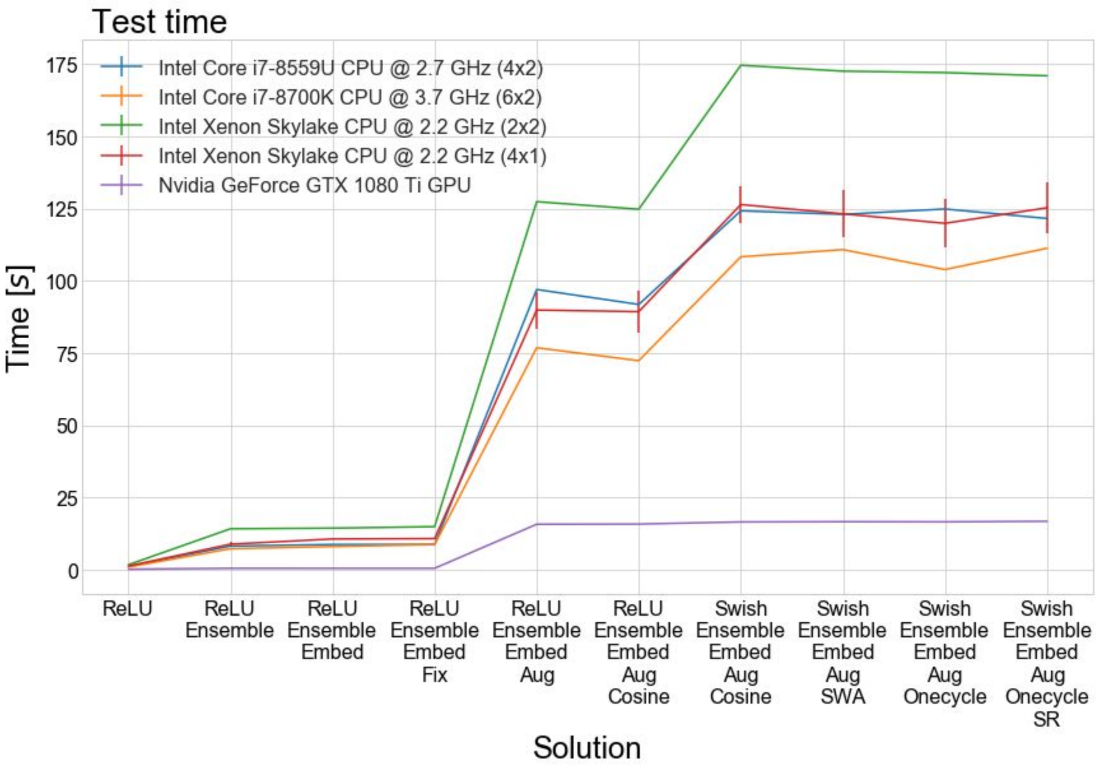
BACKUP SLIDES

LUMIN TIMING

TRAIN TIME

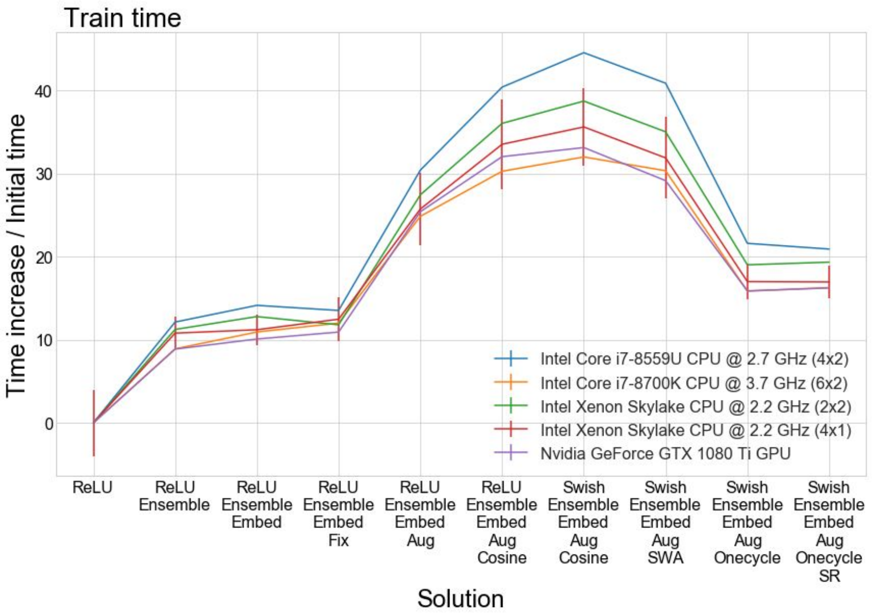

Train time
Intel Core i7-8559U CPU @ 2.7 GHz (4x2)
Intel Core i7-8700K CPU @ 3.7 GHz (6x2)
Intel Xenon Skylake CPU @ 2.2 GHz (2x2)
Intel Xenon Skylake CPU @ 2.2 GHz (4x1)
Nvidia GeForce GTX 1080 Ti GPU
Time [s]
3000
2500
2000
1500
1000
500
0
ReLU
ReLU Ensemble
ReLU Ensemble Embed
ReLU Ensemble Embed Fix
ReLU Ensemble Embed Aug
ReLU Ensemble Embed Aug Cosine
Swish Ensemble Embed Aug Cosine
Swish Ensemble Embed Aug SWA
Swish Ensemble Embed Aug Onecycle
Swish Ensemble Embed Aug Onecycle SR
Solution

TEST TIME


Test time
Intel Core i7-8559U CPU @ 2.7 GHz (4x2)
Intel Core i7-8700K CPU @ 3.7 GHz (6x2)
Intel Xenon Skylake CPU @ 2.2 GHz (2x2)
Intel Xenon Skylake CPU @ 2.2 GHz (4x1)
Nvidia GeForce GTX 1080 Ti GPU
Time [s]
175
150
125
100
75
50
25
0
ReLU
ReLU Ensemble
ReLU Ensemble Embed
ReLU Ensemble Embed Fix
ReLU Ensemble Embed Aug
ReLU Ensemble Embed Aug Cosine
Swish Ensemble Embed Aug Cosine
Swish Ensemble Embed Aug SWA
Swish Ensemble Embed Aug Onecycle
Swish Ensemble Embed Aug Onecycle SR
Solution

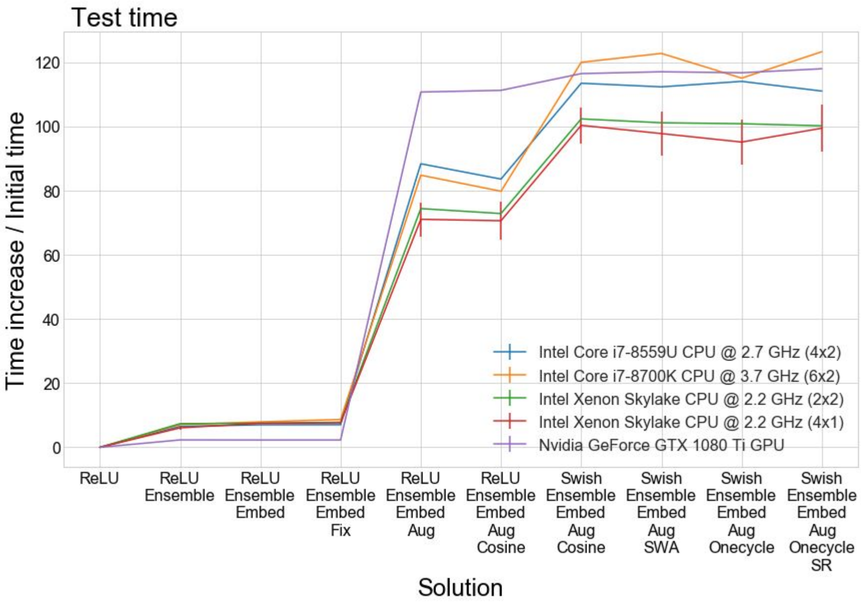
RELATIVE TRAIN-TIME INCREASE

- All additions bring about a 20 times increase in train time
- Accounting for ensemble training, 2 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?)



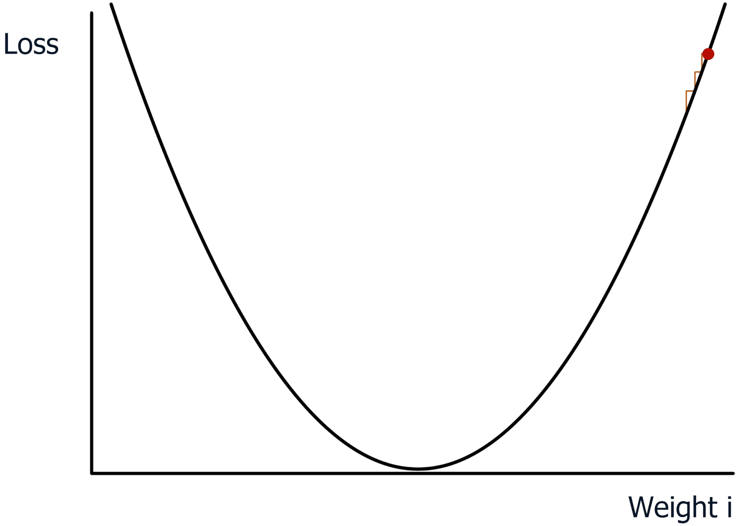
METHOD SUMMARIES

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

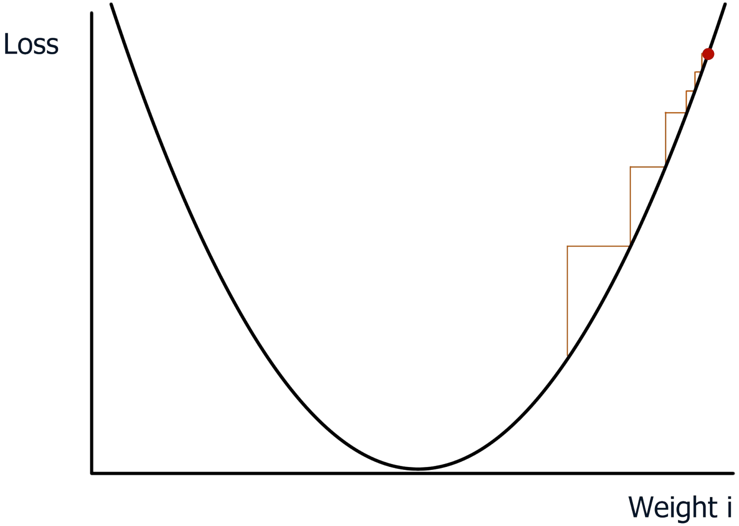
Learning rate finder

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



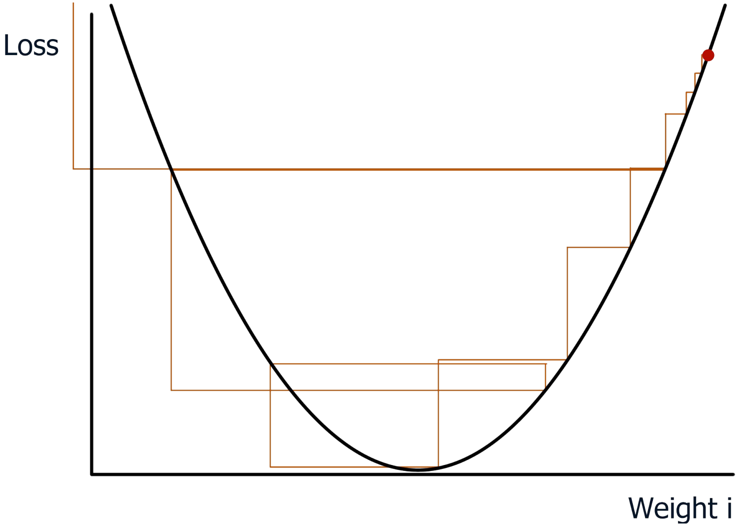
Learning rate finder

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



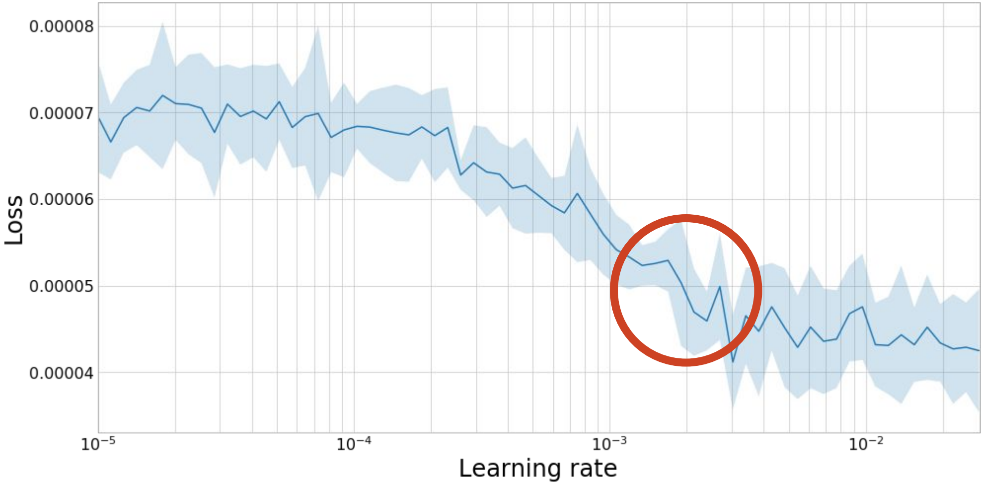
Learning rate finder

1. Starting from a tiny LR (~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)



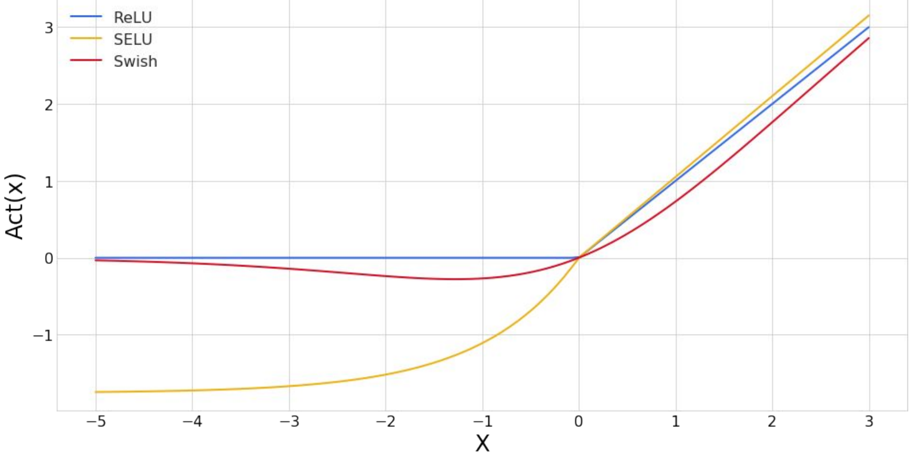
Learning rate finder

- The optimum LR is the highest LR at which the loss is still decreasing
- Further explanation in this lesson



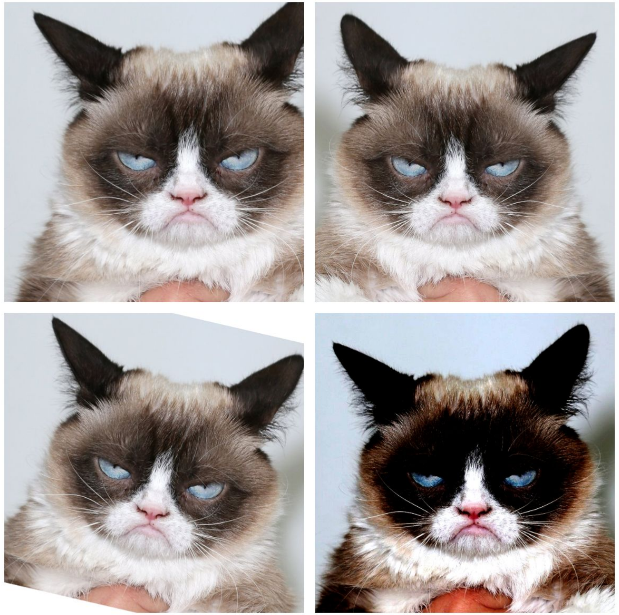
ACTIVATION FUNCTIONS

- 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



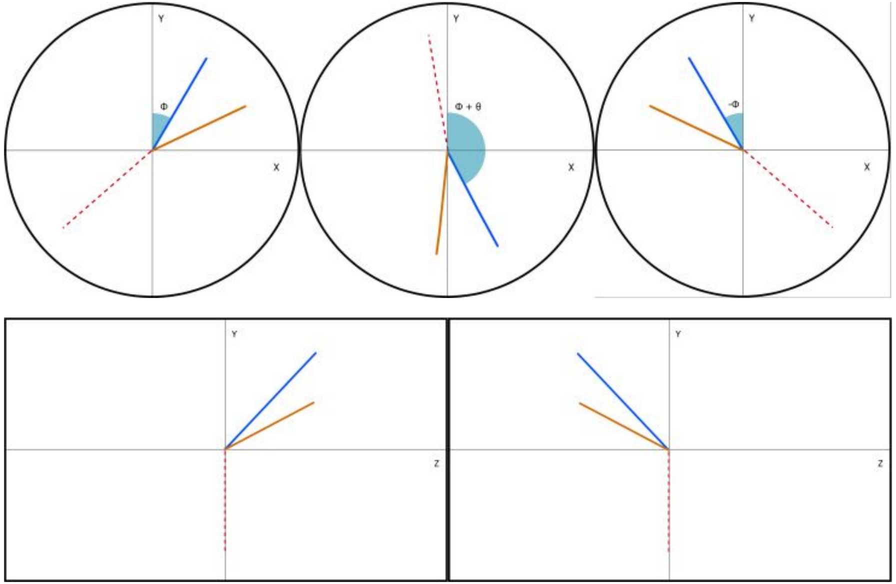
Data augmentation

- Data augmentation involves applying transformations to input data such that the 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.



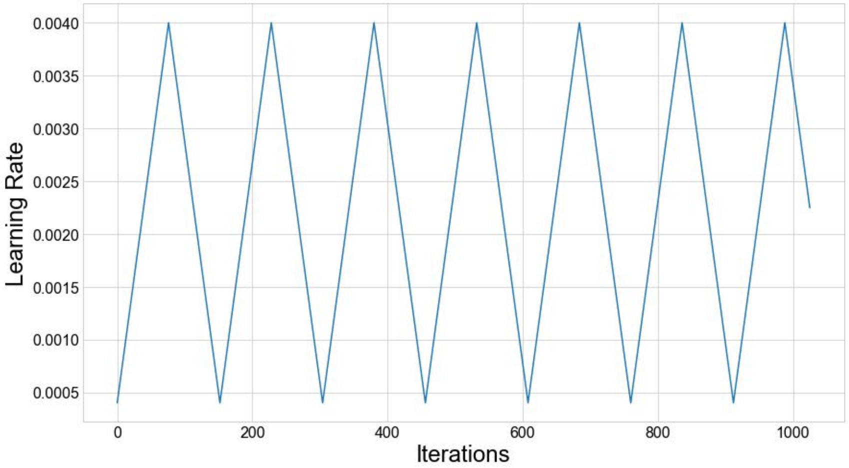
Data augmentation

- 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



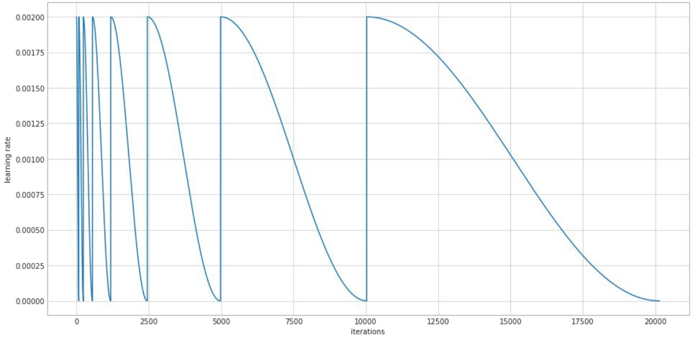
LINEAR CYCLING

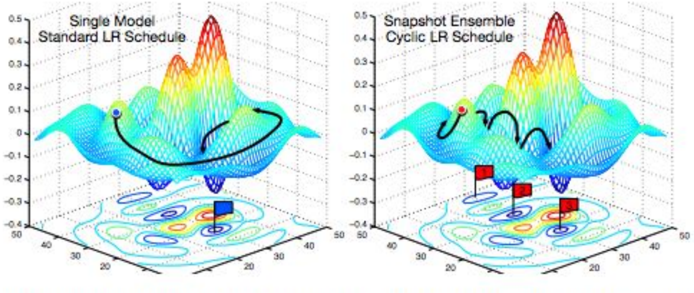
- Adjusting the LR during training is a common technique for achieving better performance
- Normally this involves decreasing the LR once the validation loss becomes flat
- Smith 2015 suggests instead to cycle the LR between high and low bounds
- Smith 2017 showed that the additional application of an out-of-phase schedule to the optimiser momentum/beta$_1$ coefficient can sometimes lead to *super convergence*
- Linear cycle example



COSINE ANNEALING

- Loshchilov and Hutter 2016 instead suggests that the LR should be 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

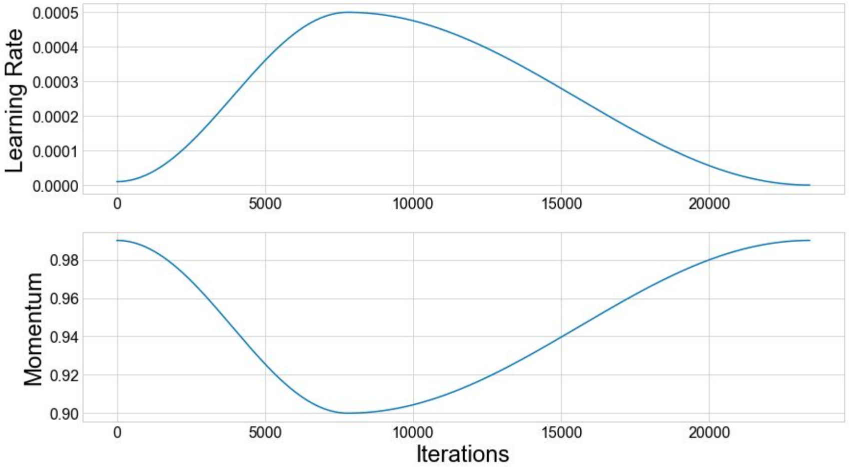




Lower figure - Huang et al., 2017, arXiv:1704.00109

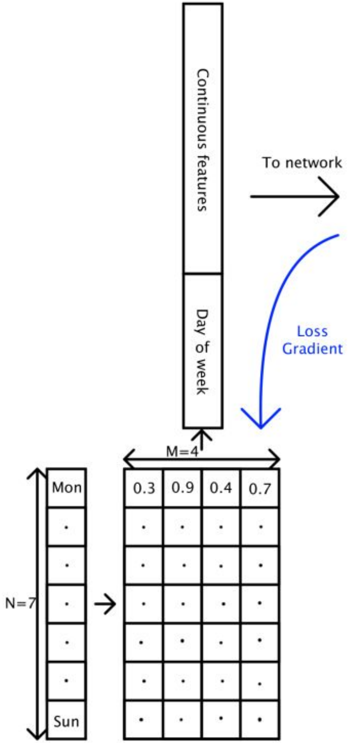
ONE CYCLING

- 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/beta$_1$
- Original paper used linear interpolation
- FastAI found a cosine interpolation was better
- LUMIN includes both interpolations
- Onecycle example



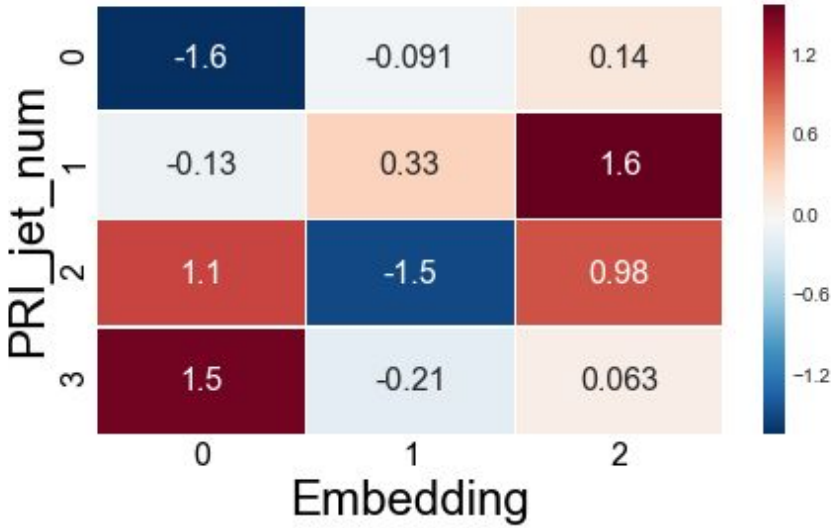
ENTITY EMBEDDING

- 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



PRI_jet_num	0	1	2
0	-1.6	-0.091	0.14
1	-0.13	0.33	1.6
2	1.1	-1.5	0.98
3	1.5	-0.21	0.063

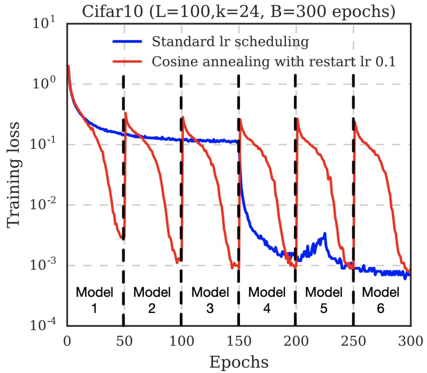
ENSEMBLING

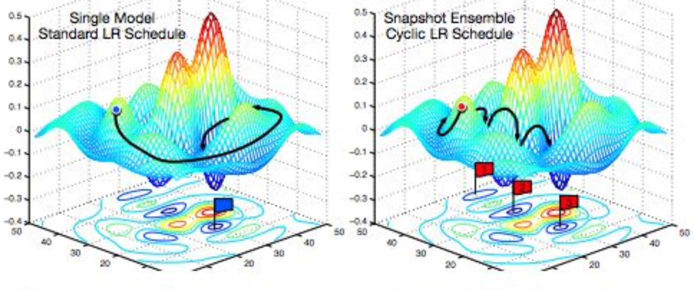
- Basic ensembling requires training multiple models, and then averaging over their predictions
- Advantages:
  - Greater generalisation to unseen data
  - Often a more powerful model
  - Provides resistance against possibility of bad training
- Disadvantages
  - Increases train time
  - Increases inference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

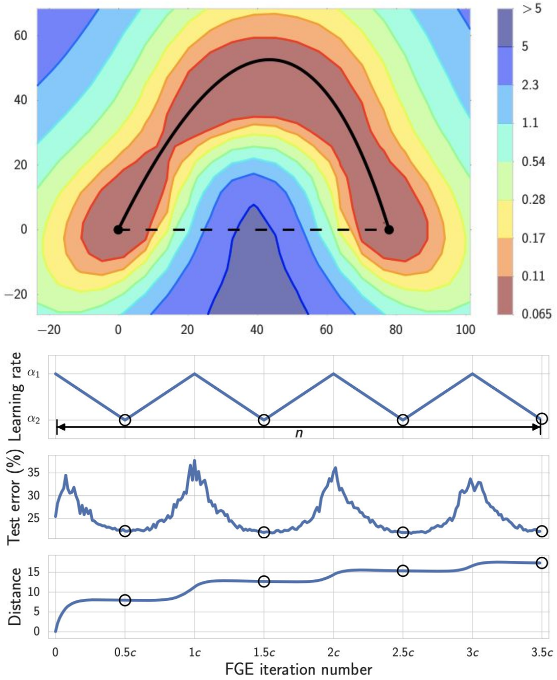




Figures - Huang et al., 2017, arXiv:1704.00109

FAST GEOMETRIC ENSEMBLING

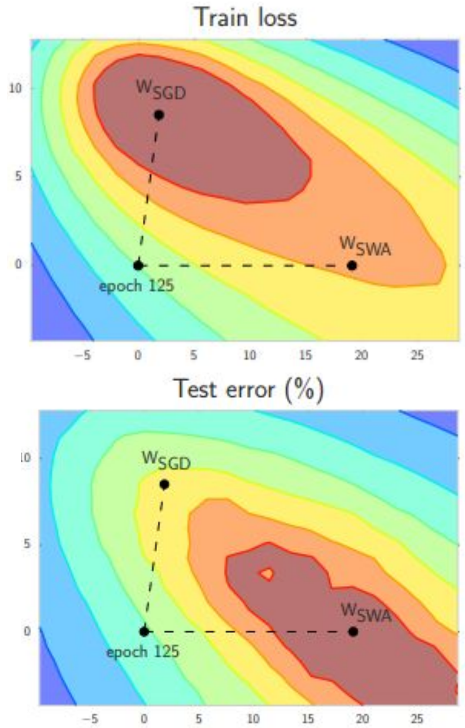
- FGE further improves on SSE by finding that loss minima are connected by curves of constant loss (rare to find true minima - almost certainly a saddlepoint/valley given dimensionality)
- Using a high-frequency linearly cycled LR it aims to direct the model along these curves, saving snapshots along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



Figures - Garipov et al., Feb. 2018, arXiv:1802.10026

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



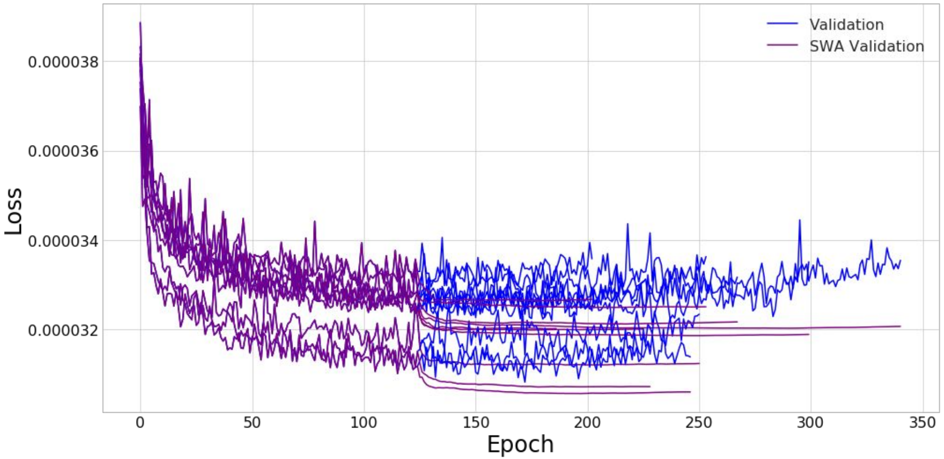
Train loss

Test error (%)

Figures - Izmailov et al., Mar. 2018, arXiv:1803.05407

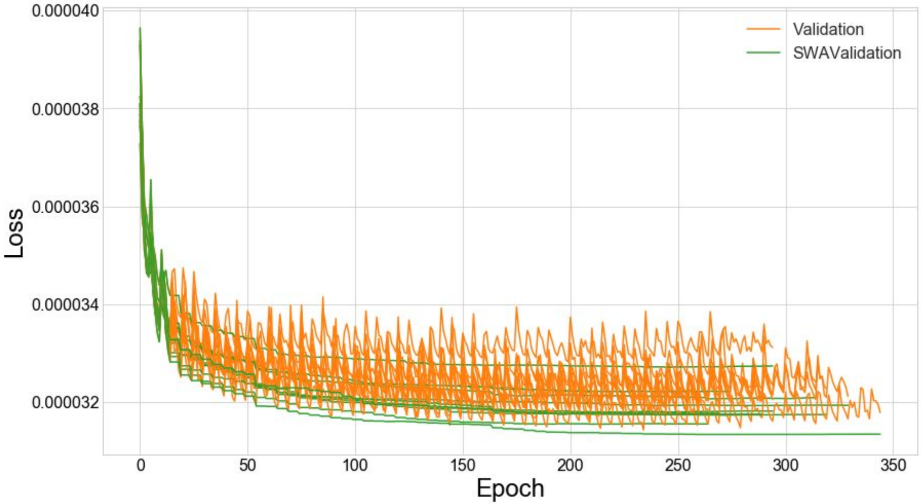
STOCHASTIC WEIGHT AVERAGING

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



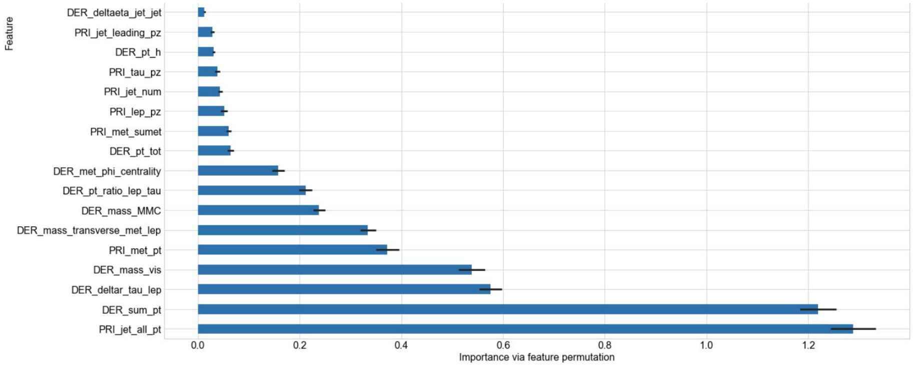
STOCHASTIC WEIGHT AVERAGING

- 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



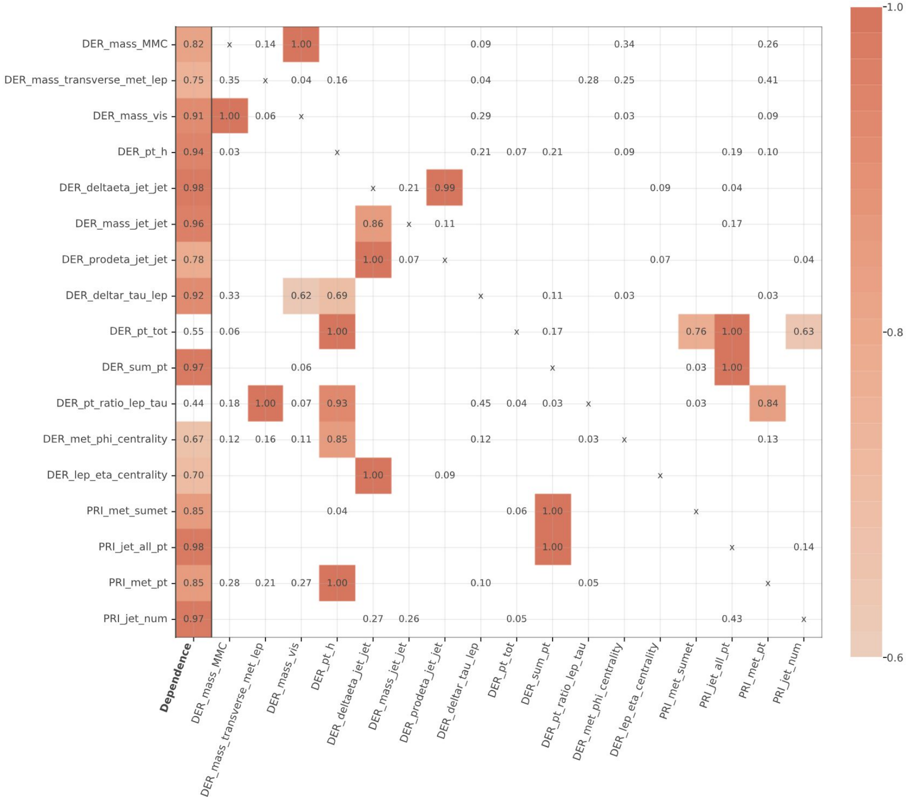
PERMUTATION IMPORTANCE

- Method to quantify the importance of each input feature
    - a. Loss of trained mode evaluated on training data
    - b. Copy of data made and one feature column is shuffled = info destroyed
    - c. Loss re-evaluated
    - d. 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



	Dependence	DER_mass_MMC	DER_mass_transverse_met_lep	DER_mass_vis	DER_pt_h	DER_deltaeta_jet_jet	DER_mass_jet_jet	DER_prodeta_jet_jet	DER_deltar_tau_lep	DER_pt_tot	DER_sum_pt	DER_pt_ratio_lep_tau	DER_met_phi_centrality	DER_lep_eta_centrality	PRI_met_sumet	PRI_jet_all_pt	PRI_met_pt	PRI_jet_num
DER_mass_MMC	0.82	x	0.14	1.00					0.09				0.34				0.26	
DER_mass_transverse_met_lep	0.75	0.35	x	0.04	0.16				0.04			0.28	0.25				0.41	
DER_mass_vis	0.91	1.00	0.06	x					0.29				0.03				0.09	
DER_pt_h	0.94	0.03			x				0.21	0.07	0.21		0.09			0.19	0.10	
DER_deltaeta_jet_jet	0.98					x	0.21	0.99						0.09		0.04		
DER_mass_jet_jet	0.96					0.86	x	0.11								0.17		
DER_prodeta_jet_jet	0.78					1.00	0.07	x						0.07				0.04
DER_deltar_tau_lep	0.92	0.33		0.62	0.69				x		0.11		0.03				0.03	
DER_pt_tot	0.55	0.06			1.00					x	0.17				0.76	1.00		0.63
DER_sum_pt	0.97			0.06							x				0.03	1.00		
DER_pt_ratio_lep_tau	0.44	0.18	1.00	0.07	0.93				0.45	0.04	0.03	x			0.03		0.84	
DER_met_phi_centrality	0.67	0.12	0.16	0.11	0.85				0.12			0.03	x				0.13	
DER_lep_eta_centrality	0.70					1.00		0.09						x				
PRI_met_sumet	0.85				0.04					0.06	1.00				x			
PRI_jet_all_pt	0.98										1.00					x		0.14
PRI_met_pt	0.85	0.28	0.21	0.27	1.00				0.10			0.05					x	
PRI_jet_num	0.97					0.27	0.26			0.05						0.43		x