IMPROVEMENTS TO ML FOR SEARCHES AT THE LHC

Giles Strong

DMML Colloquium, Online - 20/05/20

giles.strong@outlook.com

twitter.com/Giles_C_Strong

Amva4newphysics.wordpress.com

github.com/GilesStrong

OVERVIEW

1. The LHC
2. Searches in high-energy particle physics
3. The HiggsML Kaggle challenge
4. The impact of DNN methods
5. Summary

THE LARGE HADRON COLLIDER

How to probe fundamental interactions

THE LHC

- 27 km circumference machine located on French-Swiss border
- Began operation in 2009, still in use
- Accelerated particles to high energies and collides them
- Collision data allows us to study particle interactions

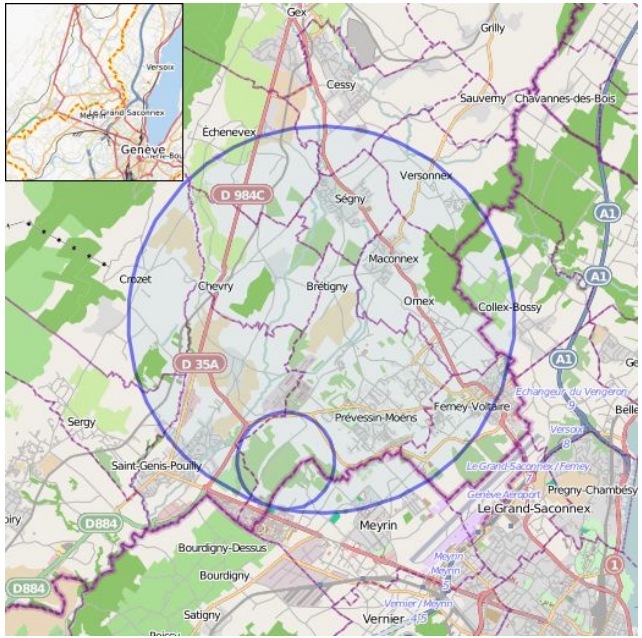


Image Source, OpenStreetMap

PROTON COLLISIONS

- Bunches of O(10^{11}) protons accelerated to almost the speed of light in opposite directions
- Proton *beams* brought to collider head on
- Such high-energy collisions cause interactions between the particles inside the protons
- Many different interactions can occur
- Need to analyse the resulting debris to *reconstruct* the collision process
- Each collision is referred to as an *event*

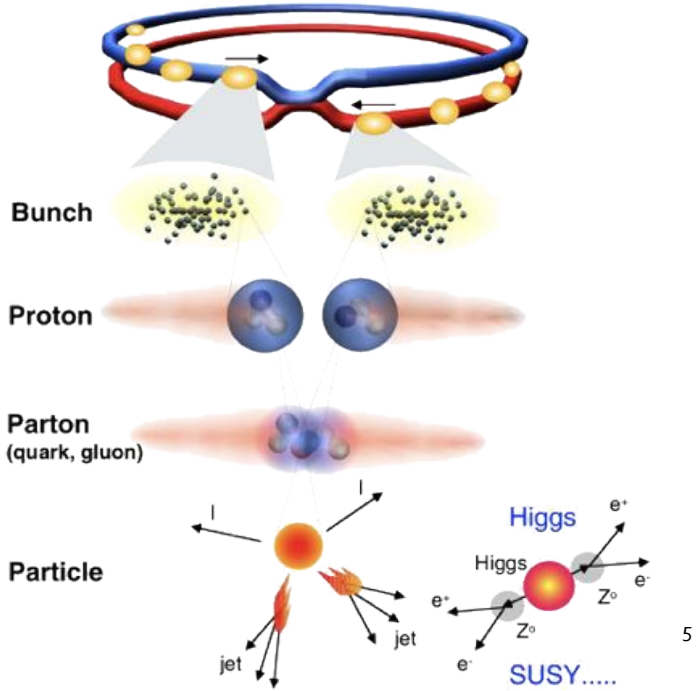


Image source, Smith, 2011

PARTICLE DETECTORS

- Specialised detectors used to record particle collisions
- Consist of many subsystems to record different types of particles
- Provide measurements of particle properties:
  - Momentum
  - Electrical charge
  - Energy

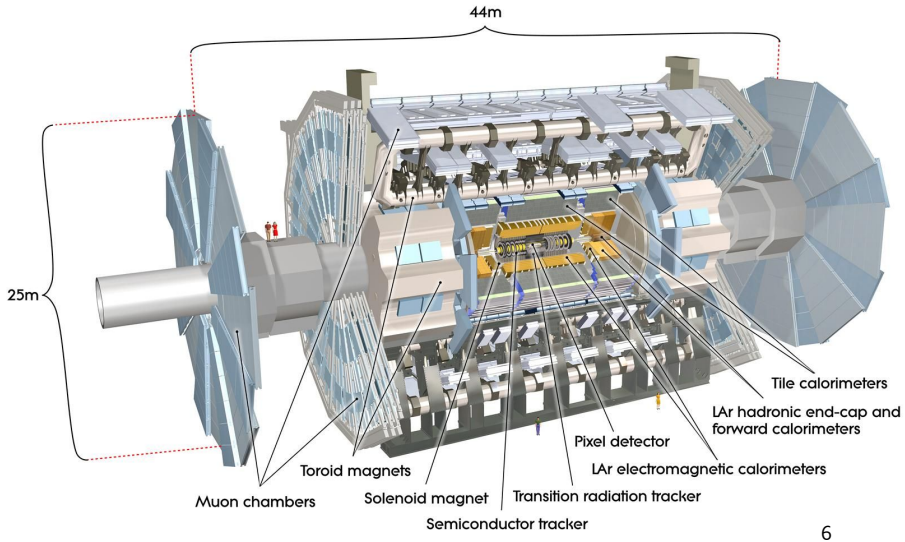


Image source, ATLAS, 2008

PARTICLE DETECTORS

- Different particles interact with different parts of the detector
- Note: we only 'see' what the detectors record
  - Trackers and muon chambers show spatial 'hits'
  - Calorimeters show energy deposits

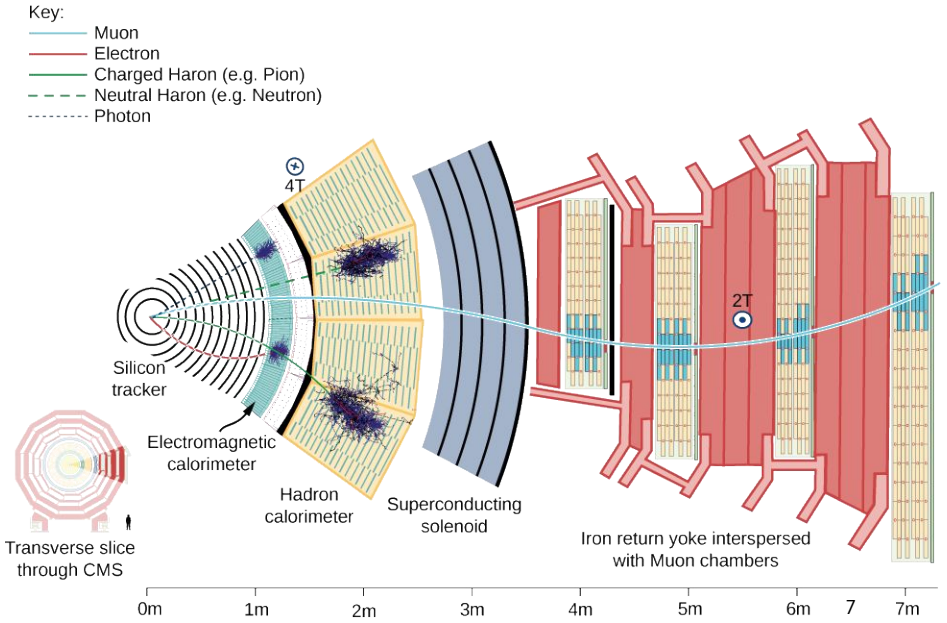
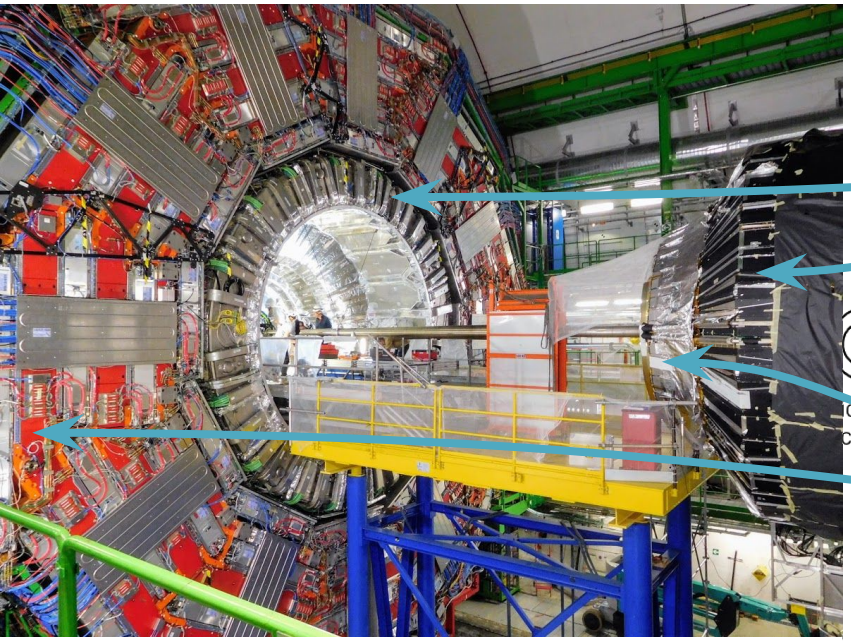
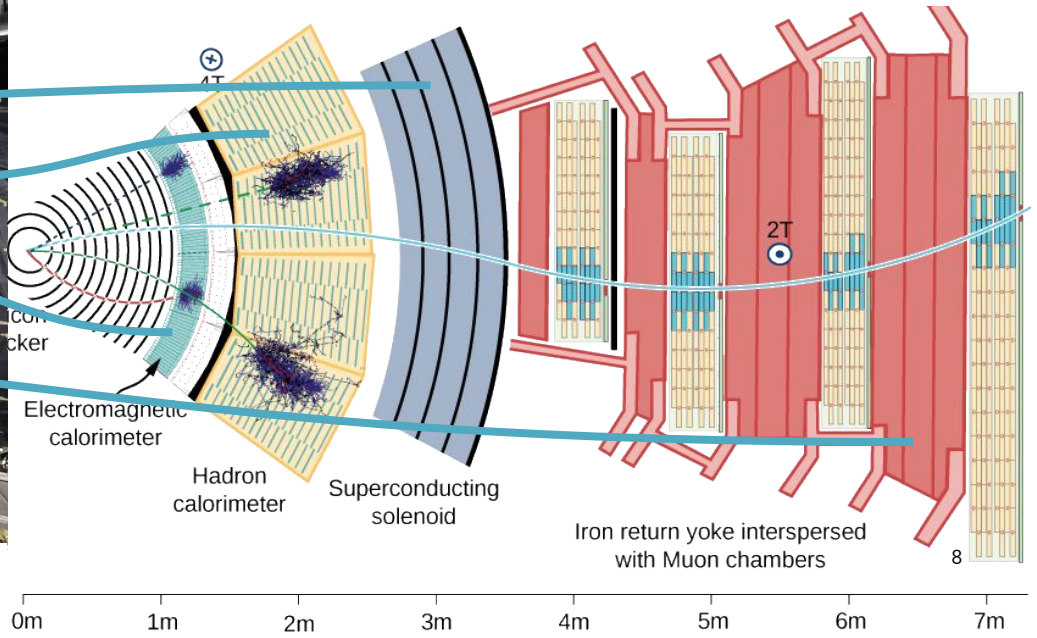


Image source, Barney, 2016

PARTICLE DETECTORS





Strong, 2018

RECONSTRUCTION

- Raw detector readout far to complicated for direct analysis
  - CMS tracker alone = 130M channels
- Instead must reconstruct known *physics objects* from raw information
  - E.g. Tracker hits (white points in left image) → Connected tracks (green lines in right image)
  - Track reconstruction was the subject of the recent TrackML competition

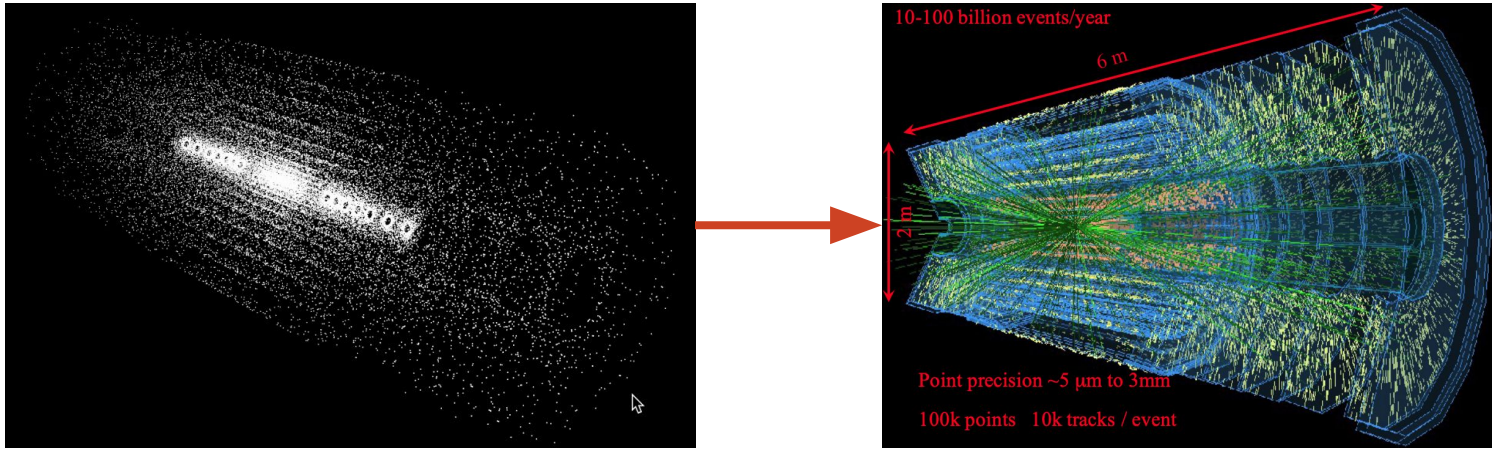


Image source, Rousseau, 2018

RECONSTRUCTION

- Several reconstruction algorithms used
  - Involves a combination of traditional and ML approaches
- Reconstruction reduces each event to a list of objects, each with relevant properties
- E.g. right:
  - 1 muon (red)
  - 1 electron (blue)
  - 2 jets (cyan)
  - Missing energy (green)

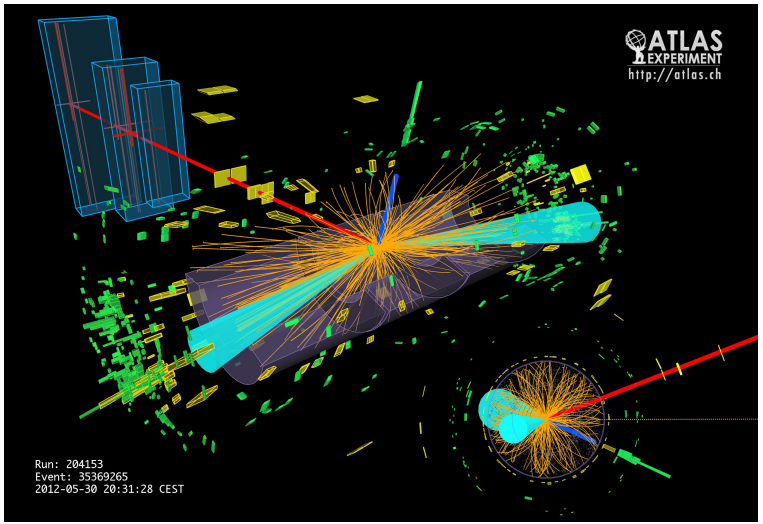
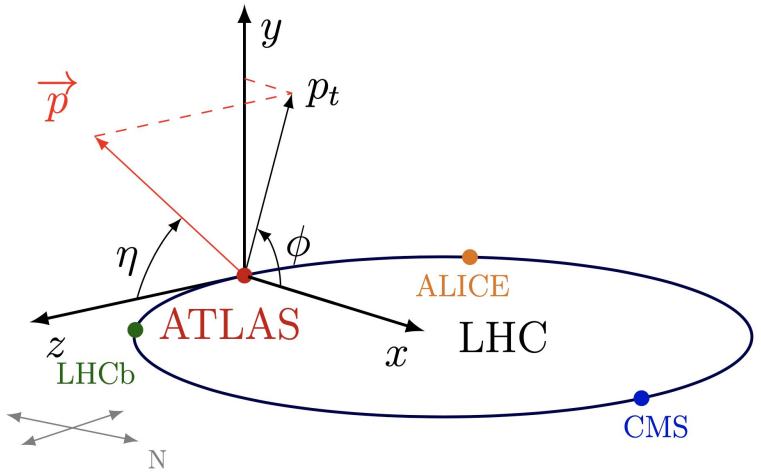


Image source, ATLAS, 2013

OBJECT PROPERTIES

- Kinematic properties of objects considered in terms of their momentum
- Each momentum vector p composed of 3 components:
  - P_t = magnitude transverse to beam axis
  - η = geometric measure of angle from beam axis
  - ϕ = azimuthal angle about beam axis
  - Can also be in terms of Cartesian coordinates (p_x,p_y,p_z)
- Energy (or rest-mass) also included
  - Together with momentum, for the *4-momentum* of the object (p, E)

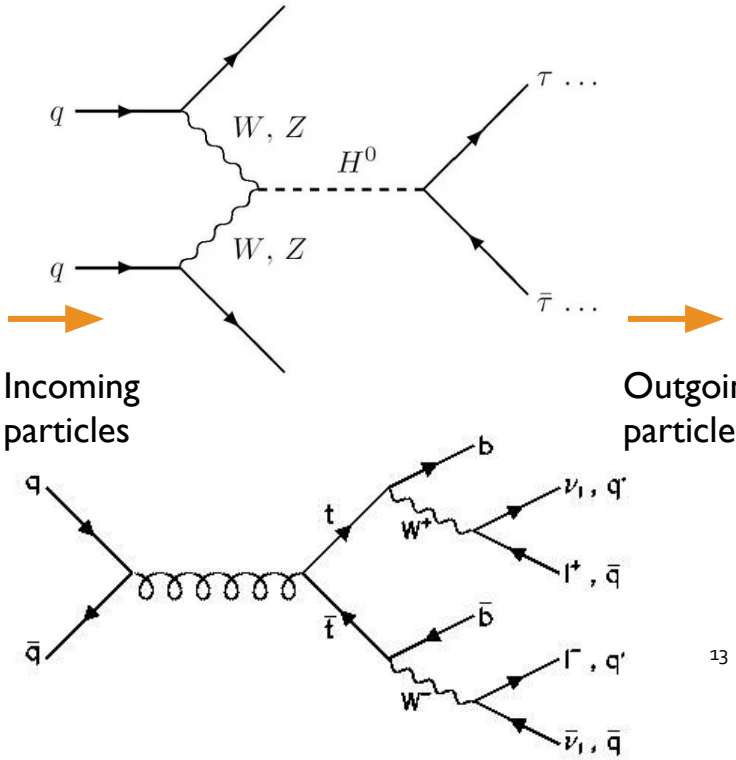


SEARCHES IN HEP

How to discover new particles

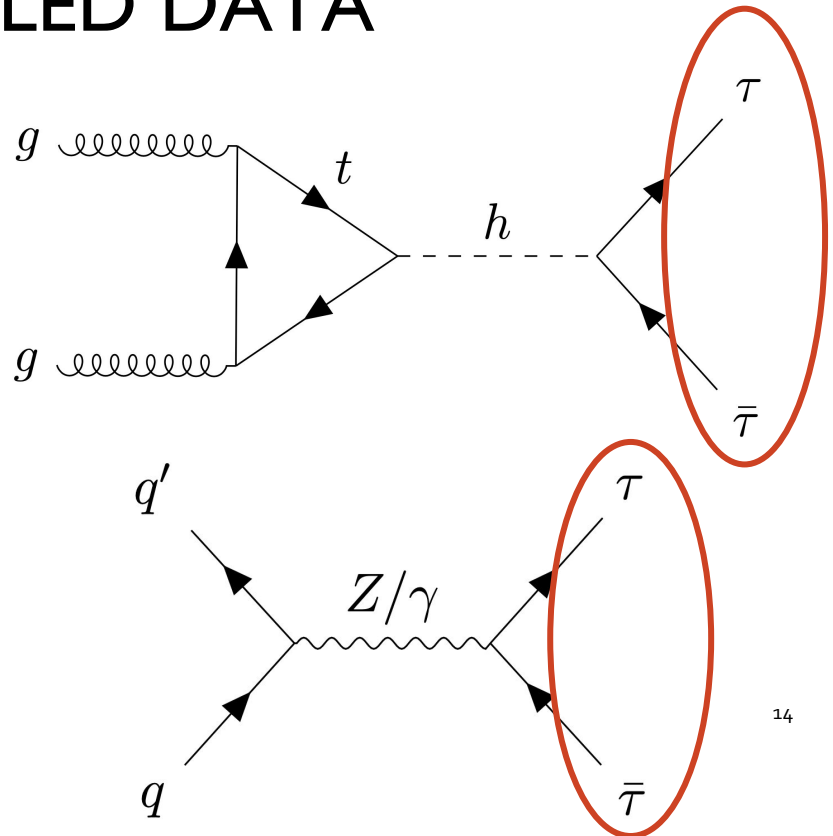
RANDOM PROCESSES

- A *search* involves showing that a new particle was present in the collisions
  - E.g. the collision process involved the Higgs boson
- But the LHC can generate many different types of collisions, not just the one being searched for
- Collisions effectively occur via random processes, with interactions having different probabilities of occuring
  - Cannot control which types of collisions occur, must simply collide lots of times and hope



UNLABELLED DATA

- Having recorded collision data, we have no way to know exactly what interactions took place
  - Particle detectors only record the remnants of collisions
  - E.g. Right: Both collision processes contain different interactions, but both lead to the same outgoing particles
- Data is unlabelled w.r.t. the collision processes



HYPOTHESIS TESTING

- We refer to the new process as our *signal*, and the other, previously discovered processes as our *background*
- Term the problem as a hypothesis test between:
  - H_b, the background-only hypothesis: signal doesn't exist
  - H_{s+b}, the signal+background hypothesis: the signal exists and contributes in addition to the background
- Each process contributes as Poisson counts to the number of events recorded in a predefined subspace of the data:

$$P(n|\mu_{\mathrm{s}},\mu_{\mathrm{b}}) = \frac{(\mu_{\mathrm{s}}+\mu_{\mathrm{b}})^n}{n!}e^{-(\mu_{\mathrm{s}}+\mu_{\mathrm{b}})}$$

HYPOTHESIS TESTING

- λ is the likelihood ratio of the two hypotheses:

$$\lambda = \frac{P(n|0,\mu_\mathrm{b})}{P(n|\hat{\mu}_\mathrm{s},\mu_\mathrm{b})} = \left(\frac{\mu_\mathrm{b}}{\hat{\mu}_\mathrm{s}+\mu_\mathrm{b}}\right)^n e^{\hat{\mu}_\mathrm{s}} = \left(\frac{\mu_\mathrm{b}}{n}\right)^n e^{n-\mu_\mathrm{b}}$$

  - Where $\hat{\mu}_\mathrm{s} = n - \mu_\mathrm{b}$

- This is used to build a test statistic, q_0:

$$q_0 = \begin{cases} -2\ln\lambda & \text{if } n > \mu_\mathrm{b}, \\ 0 & \text{otherwise} \end{cases}$$

HYPOTHESIS TESTING

- Level of disagreement with observed data and H_b quantified by a *p*-value which can be approximated using the Gaussian CDF, Φ:

$$p = 1 - \Phi\left(\sqrt{q_0}\right)$$

- In HEP, a *p*-value < 2.9×10^{-7}, is often required for *observation* of a new signal (see e.g. Dorigo, 2015 for history and discussion)

HYPOTHESIS TESTING

- *p*-values are usually converted to a *Z*-score (the significance):

$$Z = \sqrt{q_0} = \sqrt{2\left(n \ln\left(\frac{n}{\mu_\mathrm{b}}\right) - n + \mu_\mathrm{b}\right)}$$

- A significance of 5 or more corresponds to the *p*-value for observation
- N.B. this section is highly compressed, see e.g. Cowan et al., 2010 for further details

MONTE CARLO SIMULATION

- In order to compute the significance, but we still need to know the expected background μ_b
- Luckily, particle theory & stochastic modelling allow us to simulate particle collisions and the detector (see e.g. Buckley et al., 2011, Geant4, Delphes)
- The analysis flow can be applied to this *Monte Carlo* (MC) data just like the recorded data
  - We have the truth labels for the MC data and can estimate μ_b and μ_s
- The MC data can also be used for training ML models
- Isn't a 1→1 correspondence between MC events and observed events
  - Instead each MC event has a weight indicating how many observed events it corresponds to

BLIND ANALYSIS

- HEP analyses normally performed *blind*
  - Observed data only used once entire analysis is fixed and internally reviewed.
  - Must optimise using MC data
- Need a metric to guide optimisation

EXPECTED SIGNIFICANCE

- Replace n_{obs} with the mean number of signal and background: $n_{obs}=s+b$
- Approximate median significance AMS is now:

$$\mathrm{AMS}_2 = \sqrt{2\left((s+b)\ln\left(1+\frac{s}{b}\right)-s\right)}$$

- s and b estimated purely from MC*
  - Sum up event weights of MC in signal region for signal and background separately
  - No observed data required to compute expected performance

*Simplification: can sometimes, or are forced to, estimate backgrounds from signal-depleted regions of observed data

OPTIMISATION

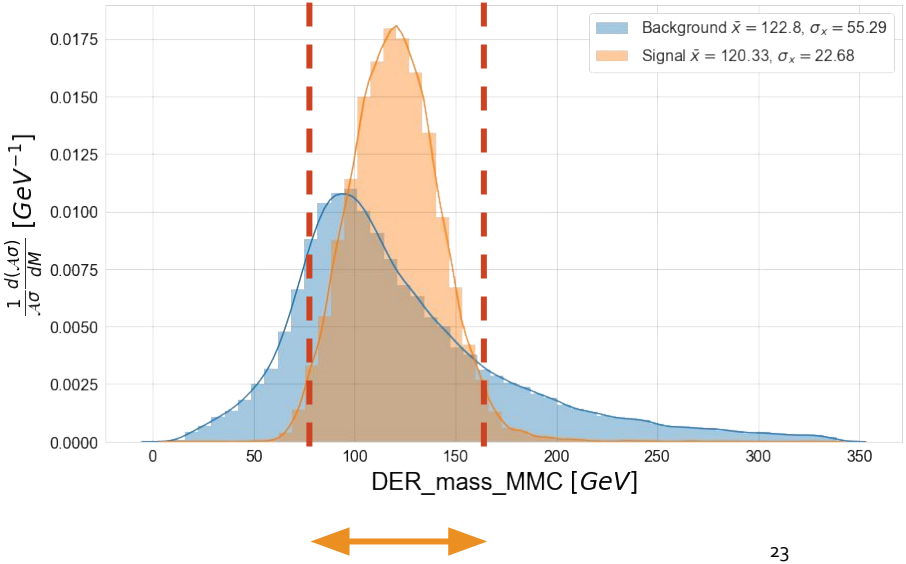
- For b >> s, the AMS reduces to:

$$\mathrm{AMS}_3 = \frac{s}{\sqrt{b}}$$

- I.e. the analysis sensitivity improves with the signal to background ratio
- Need to define the *signal region* carefully

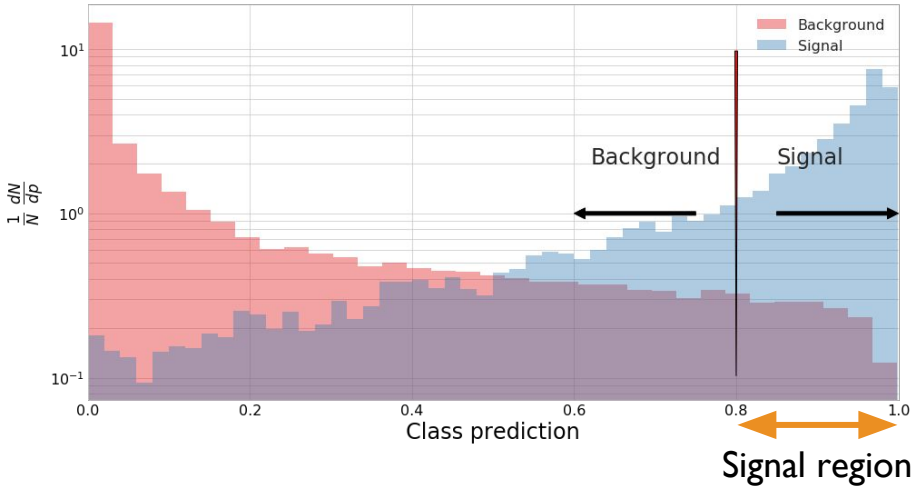
TRADITIONAL APPROACH

- Use physics knowledge to define subset of possible variables to check manually
- Define cut(s) to optimise sensitivity
- The variable(s) used are often complex combinations of basic information
  - E.g. masses, specific angles in a certain rest-frame, (s)transverse masses
  - But many different variable combinations exist: difficult to check all of them



ML APPROACH

- Ideally want single variable which the two classes are most linearly separable*
- Treat problem as a binary classification task between signal and background
- Can then *cut* on model prediction to define the signal region
- Cut optimised using the expected significance (AMS)
  - N.B. A full analysis would actually use density estimation, e.g. histograms and run the hypothesis test in multiple bins



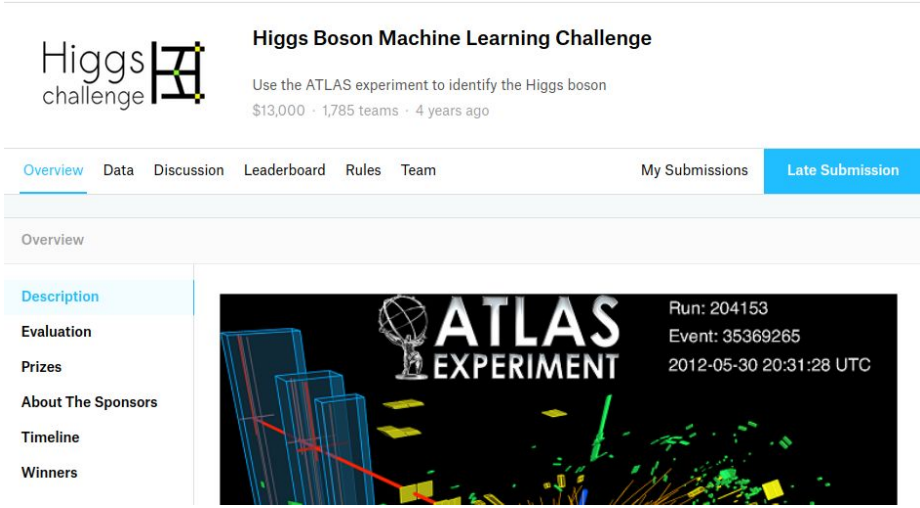
*Simplification: Sometimes multiple variables are used in different regions, categories of the data, or 2D distributions may be used

THE HIGGSML KAGGLE CHALLENGE

Crowdsourcing research problems

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
- Participants were tasked with classifying the events in order to optimise the Approximate Median Significance
- The competition was highly successful, and helped introduce new methods to HEP, as well as produce more widely used tools, such as XGBoost



HIGGSML DATASET

- Columnar data format
  - 30 input features
  - 1 target feature (signal or background)
  - Per event weight
- Low-level features (PRI_):
  - 3-momenta of several physics objects
- High-level features (DER_):
  - physics-inspired non-linear combinations of low-level info.
- Some undefined values (NaN)
  - N.B. These are not *missing* data, but due to certain particles not being present in a given event

	0	1	2	3	4	5	6	7	8	
DER_mass_MMC	NaN	16.095997	90.885002	99.398003	1.355930e+02	166.511002	98.335999	NaN	142.727997	1.077890
DER_mass_transverse_met_lep	9.274100e+01	65.269997	32.129002	6.896999	3.209800e+01	84.802002	17.827000	1.756320e+02	80.502998	1.303600
DER_mass_vis	1.760700e+01	13.707000	57.873001	61.672001	8.001500e+01	84.711998	68.535004	3.562680e+02	122.069000	7.012300
DER_pt_h	6.749985e-01	54.443001	94.754997	30.483000	2.939900e+01	84.246002	30.757000	7.577800e+01	36.125999	1.506100
DER_deltaeta_jet_jet	NaN	NaN	NaN	3.366000	NaN	1.389000	NaN	3.830000e-01	NaN	
DER_mass_jet_jet	NaN	NaN	NaN	255.675995	NaN	57.353008	NaN	9.273601e+01	NaN	
DER_prodeta_jet_jet	NaN	NaN	NaN	-2.773000	NaN	1.879000	NaN	2.084000e+00	NaN	
DER_deltar_tau_lep	5.620000e-01	0.453000	1.683000	3.005000	2.630000e+00	2.460000	2.749000	3.978000e+00	2.780000	3.111000
DER_pt_tot	6.749996e-01	13.585000	29.142000	25.535000	2.939900e+01	24.561001	2.781000	4.129000e+01	5.254000	1.506100
DER_sum_pt	6.234200e+01	112.331001	140.067001	153.602005	8.689200e+01	169.492996	102.294998	2.509330e+02	109.105003	6.823100
DER_pt_ratio_lep_tau	1.269000e+00	1.053000	0.712000	0.957000	5.170000e-01	2.886000	1.306000	1.440000e+00	2.143000	1.362000
DER_met_phi_centrality	-1.382000e+00	-0.101000	1.356000	1.380000	-1.273000e+00	1.385000	1.401000	-2.190000e-01	-1.074000	-1.413000
DER_lep_eta_centrality	NaN	NaN	NaN	0.990000	NaN	0.217000	NaN	-1.056691e-08	NaN	
PRI_tau_pt	2.748000e+01	29.261999	42.894001	31.579000	5.729300e+01	24.849001	29.907000	4.148500e+01	23.036999	2.888800
PRI_tau_eta	-1.178000e+00	-1.383000	-0.650000	-0.445000	-1.730000e-01	-0.105000	1.134000	1.863000e+00	-2.166000	1.824000
PRI_tau_phi	-2.795000e+00	0.346000	0.233000	-1.944000	-2.324000e+00	0.525000	0.509000	-5.270000e-01	1.564000	-1.944000
PRI_lep_pt	3.486200e+01	30.799999	30.544001	30.224001	2.959900e+01	71.707001	39.066002	5.975600e+01	49.368000	3.934300
PRI_lep_eta	-1.702000e+00	-1.753000	0.232000	-0.415000	-3.690000e-01	0.679000	0.690000	-2.110000e+00	0.506000	1.255000
PRI_lep_phi	-2.592000e+00	0.083000	-1.201000	1.335000	1.337000e+00	2.857000	-3.061000	-3.160000e-01	2.329000	1.115000
PRI_met_pt	6.177600e+01	43.596001	38.935001	30.208000	6.284200e+01	48.012001	13.587001	1.370370e+02	32.839001	6.467999
PRI_met_phi	4.710000e-01	2.280000	-0.232000	1.564000	5.740000e-01	1.242000	2.428000	-2.970000e+00	-0.863000	1.957000
PRI_met_sumet	5.266899e+01	157.854004	174.643997	298.371002	1.931640e+02	267.796997	122.044998	2.618760e+02	141.309006	2.404140
PRI_jet_num	-1.611710e-08	1.000000	1.000000	2.000000	-1.611710e-08	2.000000	1.000000	3.000000e+00	1.000000	-1.611710
PRI_jet_leading_pt	NaN	52.269001	66.628998	47.298000	NaN	37.284000	33.321999	5.804000e+01	36.699001	
PRI_jet_leading_eta	NaN	0.182000	0.126000	1.439000	NaN	2.231000	2.518000	-1.647000e+00	1.356000	
PRI_jet_leading_phi	NaN	-1.898000	2.750000	0.537000	NaN	-1.388000	-0.755000	1.250000e+00	-1.363000	
PRI_jet_subleading_pt	NaN	NaN	NaN	44.500999	NaN	35.653000	NaN	4.643100e+01	NaN	
PRI_jet_subleading_eta	NaN	NaN	NaN	-1.927000	NaN	0.842000	NaN	-1.265000e+00	NaN	
PRI_jet_subleading_phi	NaN	NaN	NaN	-2.432000	NaN	-1.538000	NaN	-7.750000e-01	NaN	
PRI_jet_all_pt	9.804058e-07	52.269001	66.628998	91.800003	9.804058e-07	72.936996	33.321999	1.496920e+02	36.699001	9.804058
gen_target	0.000000e+00	0.000000	0.000000	1.000000	0.000000e+00	0.000000	1.000000	0.000000e+00	0.000000	1.000000
gen_weight	4.510015e+00	1.937753	1.681611	0.001503	1.716444e+00	0.744056	0.018636	7.440562e-01	3.088507	1.863612

SCORING

- Data consists of:
  - 250,000 training events
  - 550,000 testing events, of which 100,000 public & 450,000 private
- Training, public, and private datasets all normalised to have the same weight sum
  - I.e. sum of weights of signal and background in each dataset is the same
- Solutions ranked by AMS:
  - Kaggle submissions made by submitting Boolean signal|background predictions for each testing event
  - I.e. both model **and** cut must be optimised using training data

Full dataset now public; can compute testing AMS without Kaggle submission

TOP SOLUTIONS

	1st place	2nd place	3rd place
Method	70 DNNs	Many BDTs	108 DNNs
Train-time (GPU)	12 h	N/A	N/A
Train-time (CPU)	35 h	48 h	3 h
Test-time (GPU)	1 h	N/A	N/A
Test-time (CPU)	???	???	20 min
Score	3.80581	3.78913	3.78682

THE IMPACT OF DNN METHODS

A summary of arXiv:2002.01427

GENERAL ML REQUIREMENTS IN HEP

- During data collection (Level 1 trigger):
  - Train algorithm once, use for several years = train time not a concern
    - Probably GPUs available
  - Application time, extremely quick (40 MHz) = quantification, pruning, lookup-tables
    - Hardware-based: FPGA, ASIC
- During data collection (High-Level trigger):
  - Train algorithm once, use for several years = train time not a concern
    - Probably GPUs available
  - Application time, very quick (30 kHz) = quantification, pruning
    - Software based: CPU cluster

GENERAL ML REQUIREMENTS IN HEP

- During data processing (reconstruction):
  - Train algorithm once, use for a year = train time not a concern
    - Probably GPUs available
  - Application time, moderately quick, but high volume = CPU cluster
- During data analysis (e.g. HiggsML):
  - Train algorithm multiple times at short notice = train time < 1 day
    - Cannot assume GPU access, must work well on CPU
  - Application time, can be slow = must process entire dataset in under a few hours
    - Cannot assume GPU access, must work well on CPU

IMPROVEMENTS TO ANALYSIS-LEVEL CLASSIFIER

- Can have large impacts on the sensitivity of the search
- But limited hardware and short training and application times can be a limit
- Improvements to performance welcome, provided timing doesn't increase
- Decrease in timing & hardware requirements welcome, provide performance doesn't drop (too much)
- Must work on CPU

MOTIVATION

- Given the level of work that went into the solutions to the HiggsML challenge, it is a nice HEP-specific benchmark dataset for evaluating the possible benefits of new techniques
  - Also representative in metric and sample size
- I will be using it to demonstrate the cross-domain applicability of several recent methods:
  - Better treatment of categorical features
  - Domain-specific data-augmentation
  - A new activation function
  - Learning rate scheduling
  - Densely-connected networks

INVESTIGATION APPROACH

- The study aims to replicate the Kaggle challenge: solution and cut will be fixed prior to testing on private data
- Want to know the change in general performance:
  - Solutions trained and evaluated multiple times using different random seeds
  - Optimisation metrics based on averaged AMS values
- Multiple hardware configurations used to check timing
  - Timing considered during solution development, but performance was main driver

PREPROCESSING

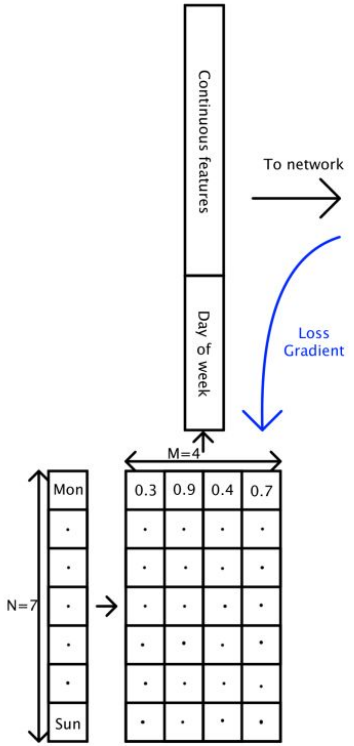
- Low-level features transformed to Cartesian coordinates (ϕ is cyclical and η is non-linear)
- 80:20 training:validation data split
- Input features standardised and normalised (computed on training data)
- NaNs replaced with zeros (post-preprocessing)
- Weights of training data normalised to one for signal & background separately:
  - Weight will be used in loss function to account for class-imbalance and relative importance of events

BASELINE MODEL

- The basic classifier is:
  - 4-layer 100 neuron, fully-connected network, with ReLU activations
  - Adam to minimise the weighted binary cross-entropy of event class predictions
  - Learning rate found using LR range test (Smith 2015 & 2018, see backups)
- Stratified 10-fold cross-validation of training data used to train an ensemble of 10 NNs
  - Models saved on improvement on their validation folds
  - During application, each model is weighted inversely according to its validation-fold loss
- Baseline achieves metric-score of 3.664±0.007

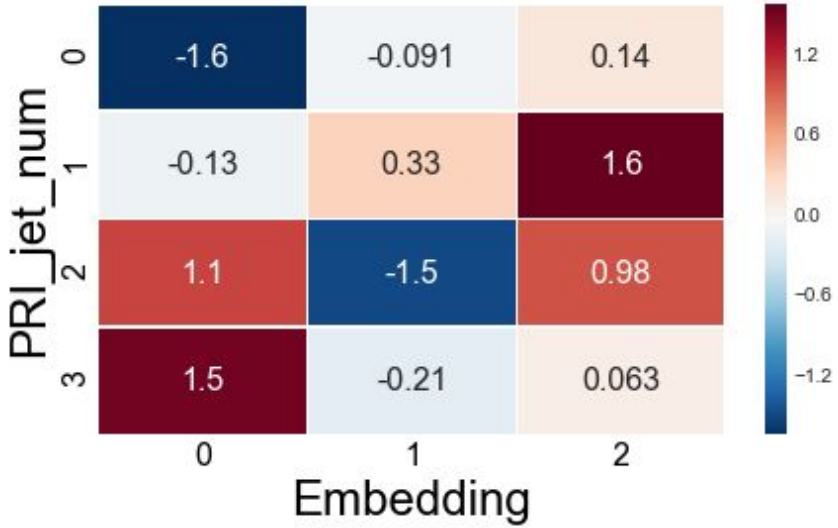
CATEGORICAL ENTITY EMBEDDING

- 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)



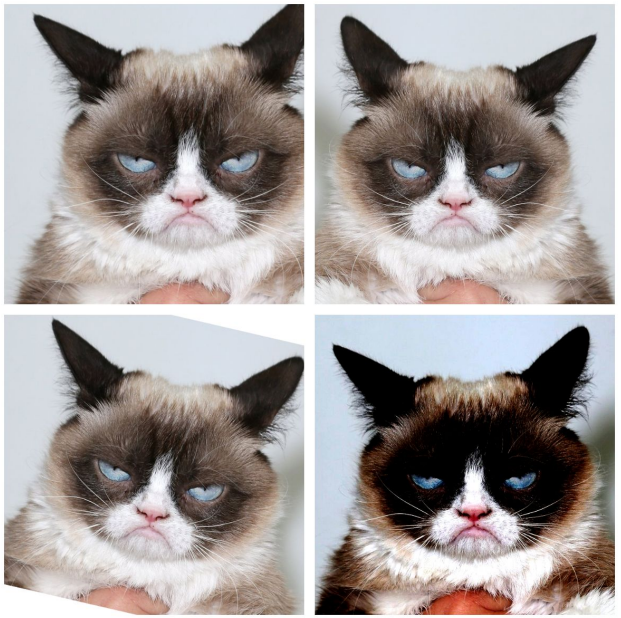
CATEGORICAL ENTITY EMBEDDING

- Embedding values start from random initialisation
- Receive gradient during backpropagation and are learnt just like any other network parameter
- Embedding of the number of jets in each event gives:
  - Moderate performance improvement $3.664 \pm 0.007 \rightarrow 3.71 \pm 0.02$
  - Small increase in train & application time



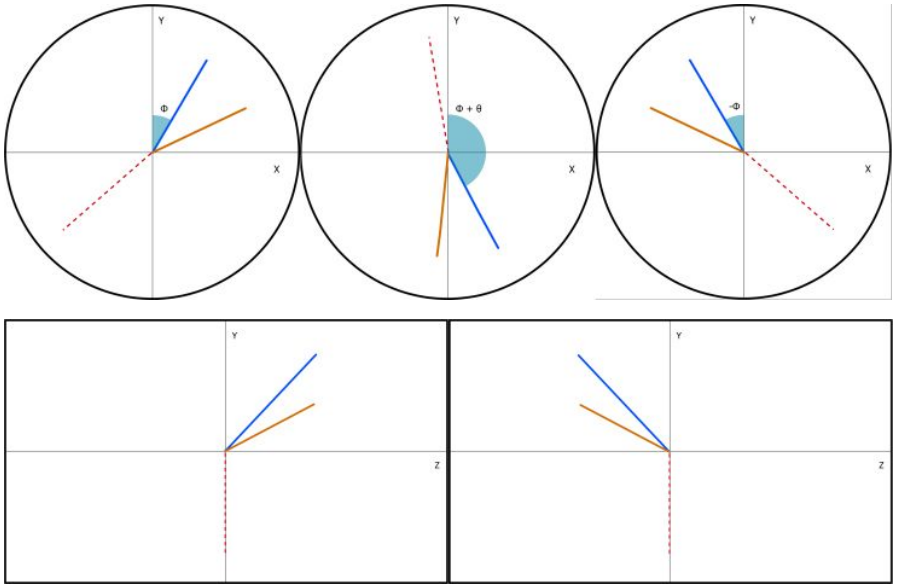
DATA AUGMENTATION

- 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

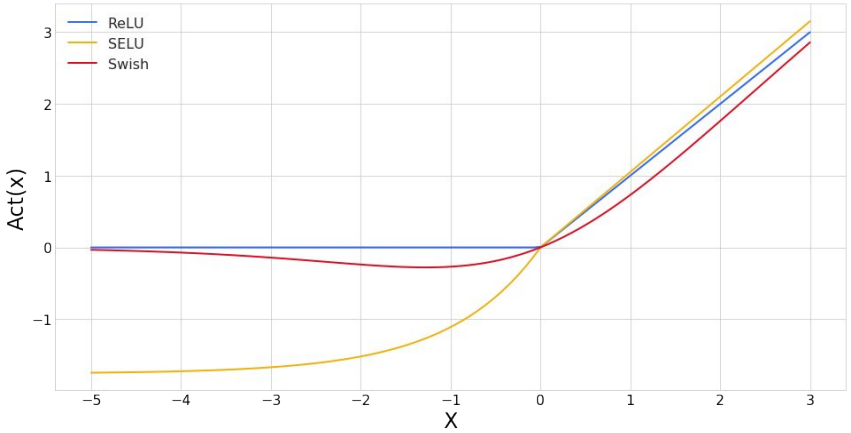
- Train-time data augmentation implemented by randomly rotating events in ϕ and randomly flipping in the z and x-axes
- At application-time the mean prediction is taken over a set of 32 transformations corresponding to 8 ϕ orientations for each possible set of flips in z and x
- Using data augmentation results in:
  - Large performance improvement $3.71 \pm 0.02 \rightarrow 3.79 \pm 0.01$
  - Very large increase in train & application time (but still reasonable to use)

SGD WITH WARM RESTARTS

- Tested Loshchilov and Hutter, 2016 (SGD with warm restarts)
  - Cosine annealed LR schedule
- Achieved slight improvement in performance
  - Eventually replaced with 1cycle (coming up soon)
  - See backup slides for details

SWISH ACTIVATION FUNCTION

- The Swish activation function (Ramachandran et al., 2017) found via reinforcement learning
  - Provides a region of negative gradient
  - Shown to provide incremental improvement over other activation functions
- Provides:
  - Small performance improvement 3.80 ±0.02→3.81±0.02
  - Small increase in train and application time
- N.B. Had previously tested SELU (Klambauer et al., 2017) , but Swish performed better

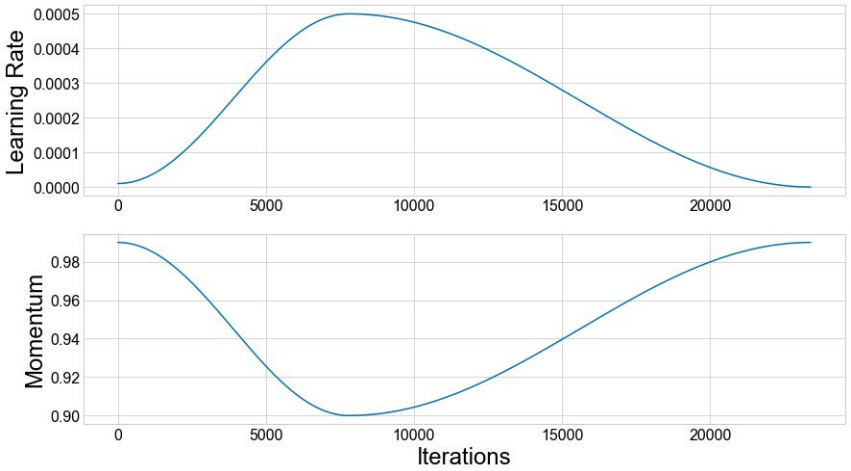


ADVANCED ENSEMBLING

- Tested several methods:
  - Huang et al. 2017 (Snapshot ensembling (SSE))
    - Produces ensembles in a single training
  - Garipov et al. 2018 (Fast geometric ensembling (FGE))
    - Produces larger ensembles in a single training
  - Izmailov et al. 2018 (Stochastic weight averaging (SWA))
    - Approximates FGE in a single model
- SWA provided reduced training time and replaced cosine annealing
  - Was then replaced by 1cycle (coming up next)
  - See Sec. 4.8 of paper for details

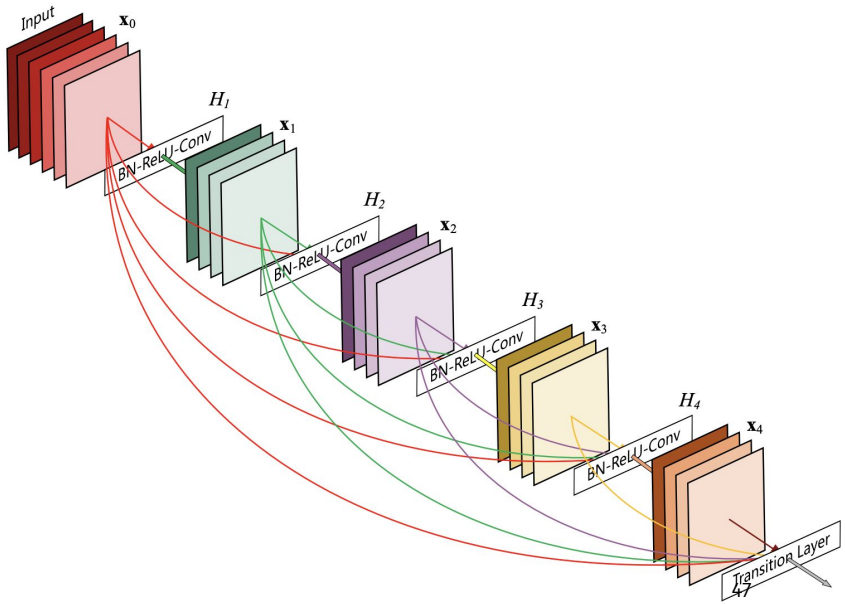
1CYCLE SCHEDULE

- Smith 2018 introduces the 1cycle schedule
- This involves running through a single cycle of increasing and then decreasing the lR, with a similar, inverted schedule applied to momentum/beta$_1$
  - Provides very fast convergence due to high-LR balanced by momentum
  - Original paper used linear interpolation
  - FastAI found a cosine interpolation was better
- Reduces training time by over 50% with no change in performance



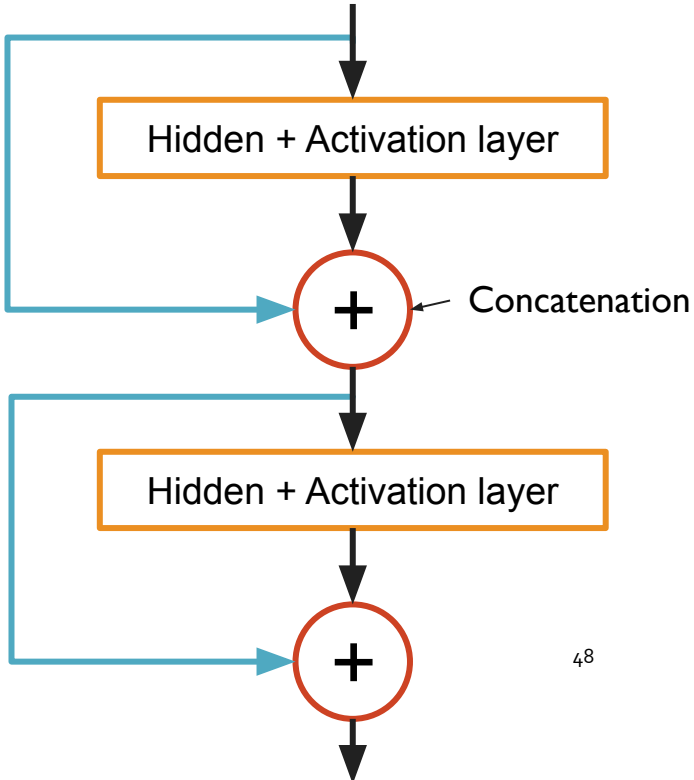
DENSE CONNECTIONS

- Huang et al. 2016 presents Densenet, a CNN architecture in which channel-wise concatenation is used to pass all the feature-maps from all previous layers to all subsequent layers
- Information is never 'lost', i.e. each layer has access to all the original inputs and weights have more direct gradient flow
- Reduces required number of free-parameters and enables 'deep supervision'



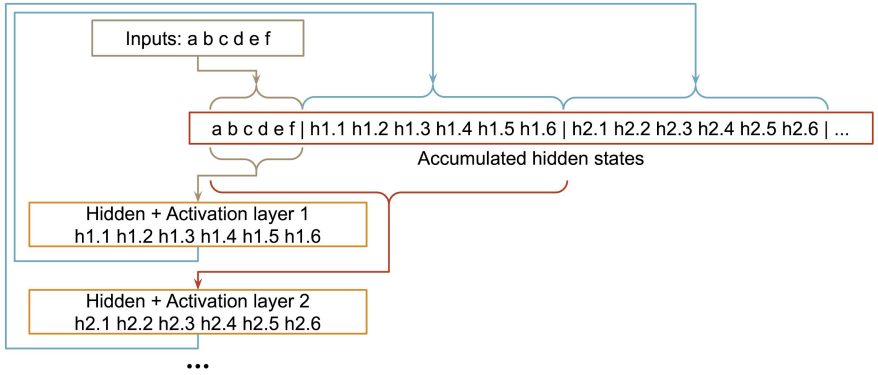
DENSE CONNECTIONS

- Even though the DNN here is not convolutional, the same idea can still be applied
- After each layer, the output tensor is concatenated (width-wise) with input tensor
- Not quite as efficient as Densenet since number layer weights increase with each layer, but still provides feature propagation



DENSE CONNECTIONS

- Can alternatively be thought of as sequentially appending to a collection of hidden states
- Places less reliance on exact settings of width and depth of network layers by protecting against over-parametrisation
  - Reduced layer widths to number of inputs (33)
  - Increased number of layers to 6 (was 4)
  - Reduces number of free parameters by a third
- Provides:
  - Small performance improvement 3.81 ±0.02→3.82±0.02
  - Small increase in train time

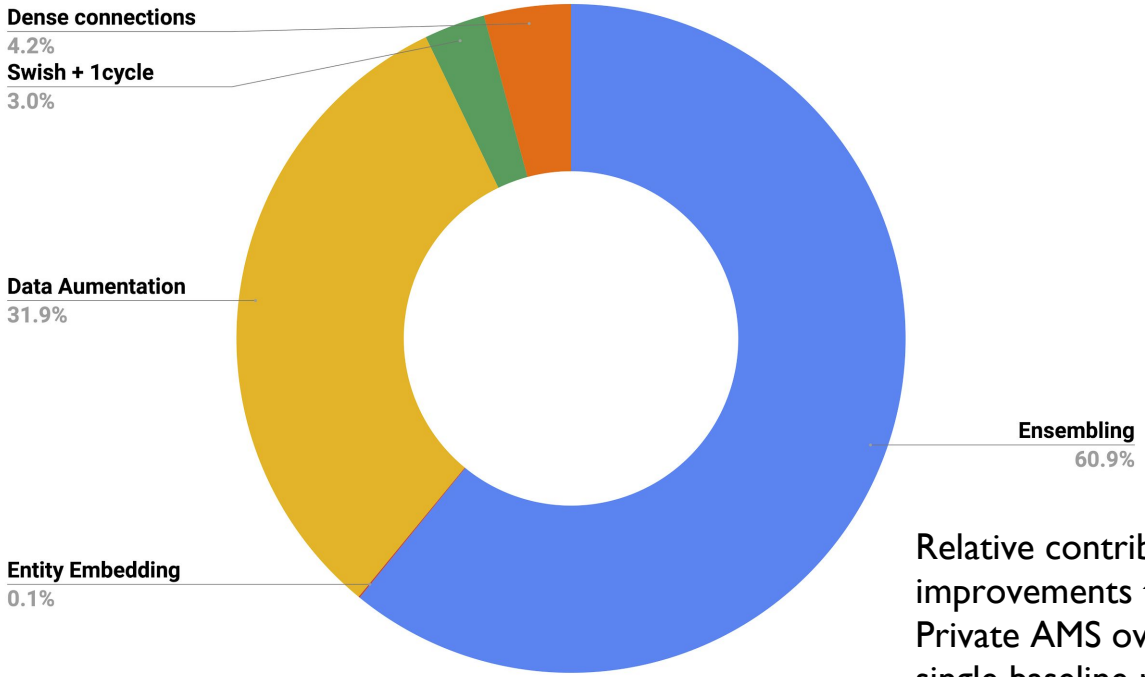


TESTING

- Model fixed and private AMS computed
  - Solution here matches 1st-place performance
- Hardware for mine:
  - GPU: Nvidia 1080 Ti
  - CPU: Intel i7-8559U (MacBook Pro 2018)
  - More hardware timings in backups
- Accounting for difference in GPU (Titan → 1080 Ti) processing power, 1st-place:
  - Trains in 100 minutes (mine 92% quicker)
  - Tests in 8 minutes (mine 97% quicker)
  - N.B. Doesn't include software changes (LISP→PyTorch)

	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

IMPROVEMENT CONTRIBUTIONS

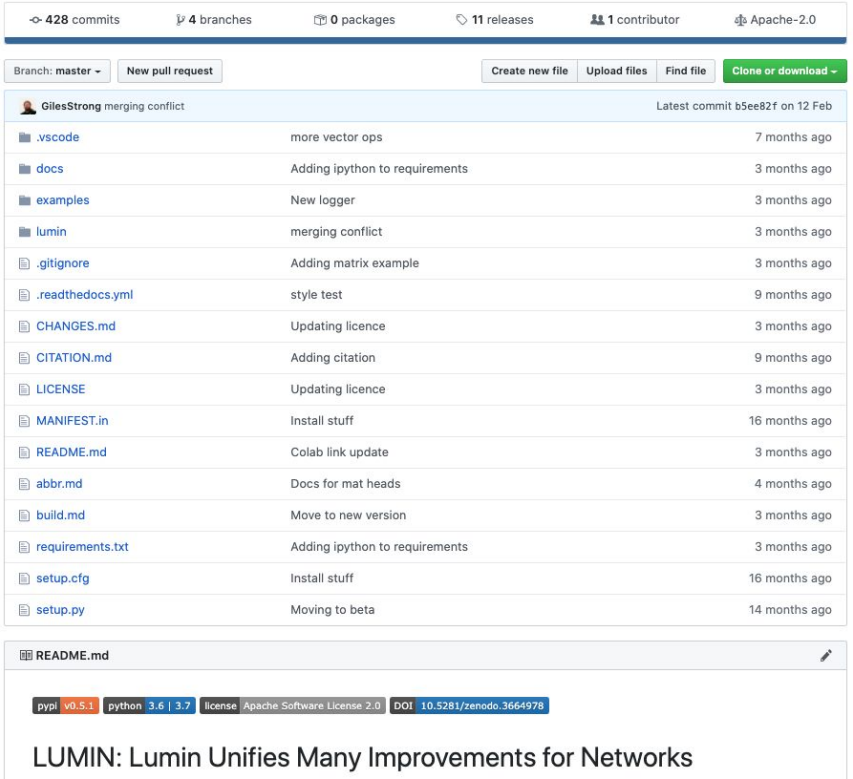


Relative contributions improvements to Private AMS over single baseline model

SUMMARY

LUMIN

- LUMIN is a PyTorch wrapper library I developed in tandem with the HiggsML study
- Provides implementations for the methods tested
- Also includes other useful methods & classes for working with HEP data and columnar data in general, and more
  - E.g. recent update adds RNNs, CNNs, and a few graph-nets
- Links:
  - Docs
  - Github
  - Colab examples



SUMMARY

- ML used in many parts of the experiments performed at the LHC
- But algorithms can be further improved by staying up-to-date with the field of deep-learning
- HiggsML study showed new methods:
  - Bring genuine improvements in performance
  - Reductions in train and application time
  - Reductions in hardware requirements: can run powerful algorithms on a laptop CPU, particularly useful to analysis-level researchers with limited hardware
  - Study code available here

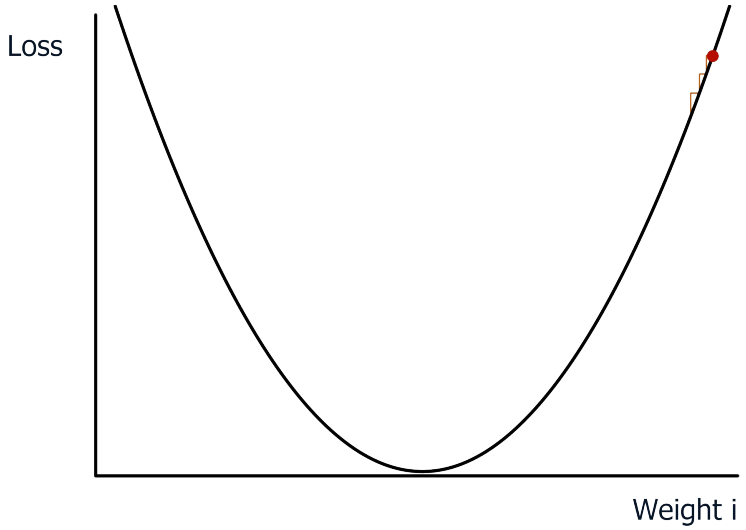
Backup slides

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

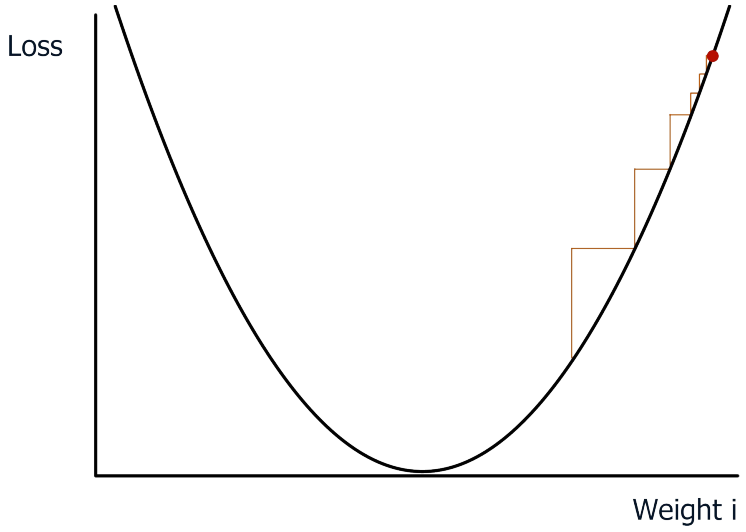
LEARNING RATE FINDER

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



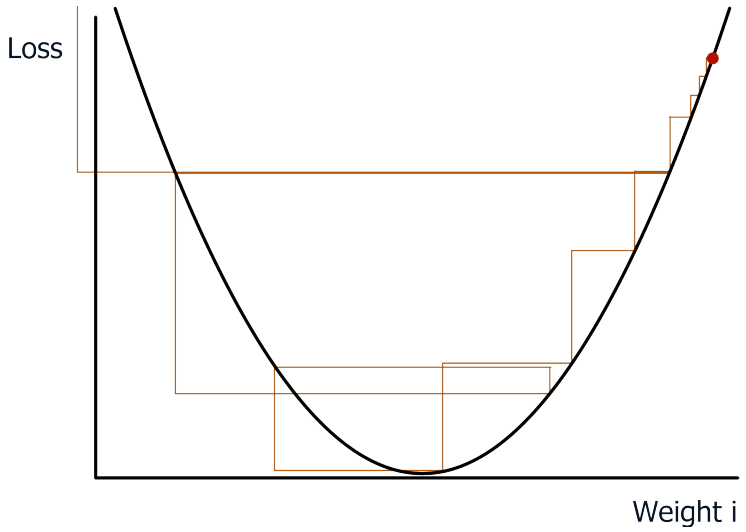
LEARNING RATE FINDER

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



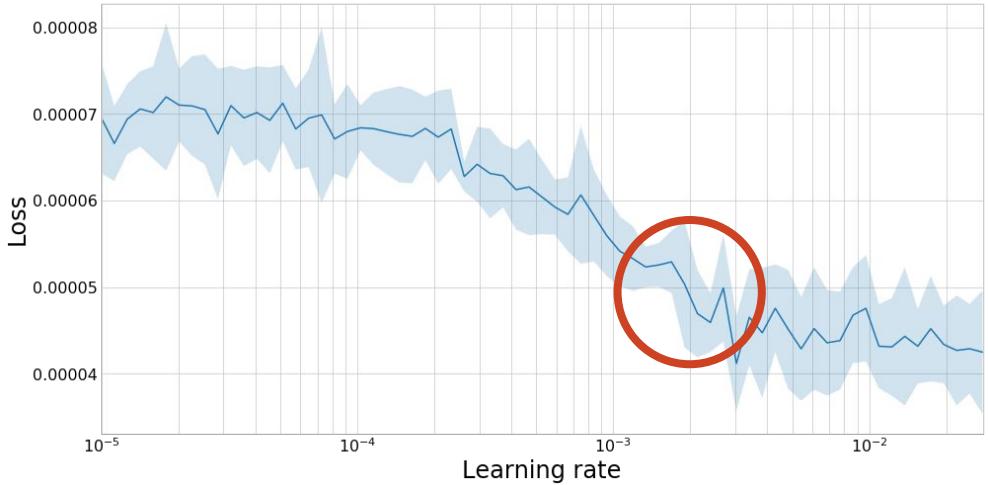
LEARNING RATE FINDER

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

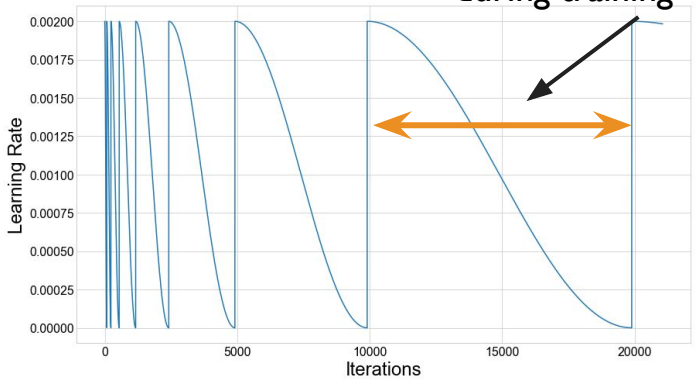
- The optimum LR is the highest LR at which the loss is still decreasing
- Further explanation in this lesson

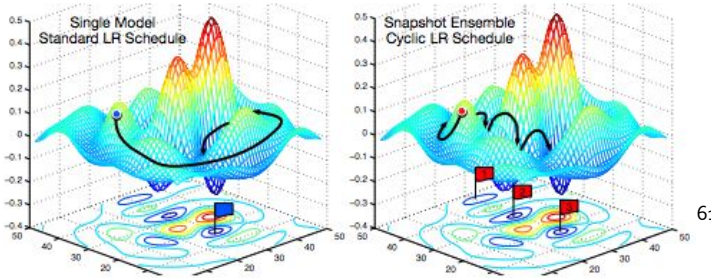


SGD WITH WARM RESTARTS

- 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
- Loshchilov and Hutter 2016 instead suggests that the LR should be decay as a cosine with the schedule restarting once the LR reaches zero
  - cosine annealing
- Huang et al. 2017 later suggests that the discontinuity allows the network to discover multiple minima in the loss surface

Can change cycle length during training

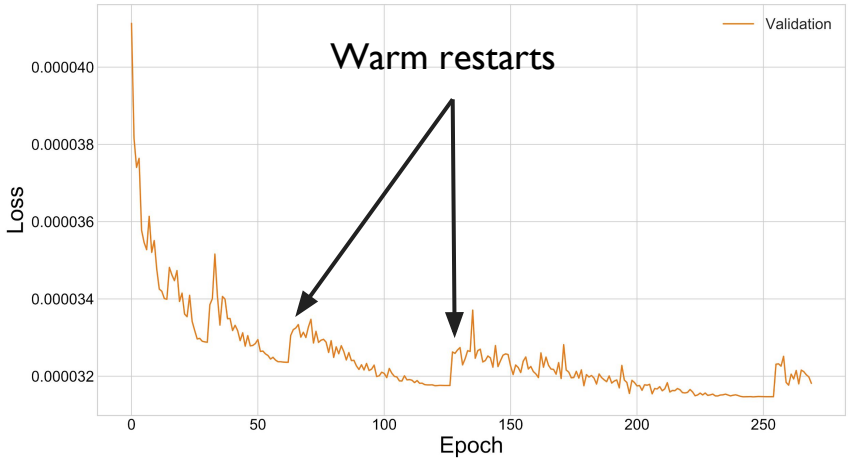




Lower figure - Huang et al., 2017, arXiv:1704.00109

SGD WITH WARM RESTARTS

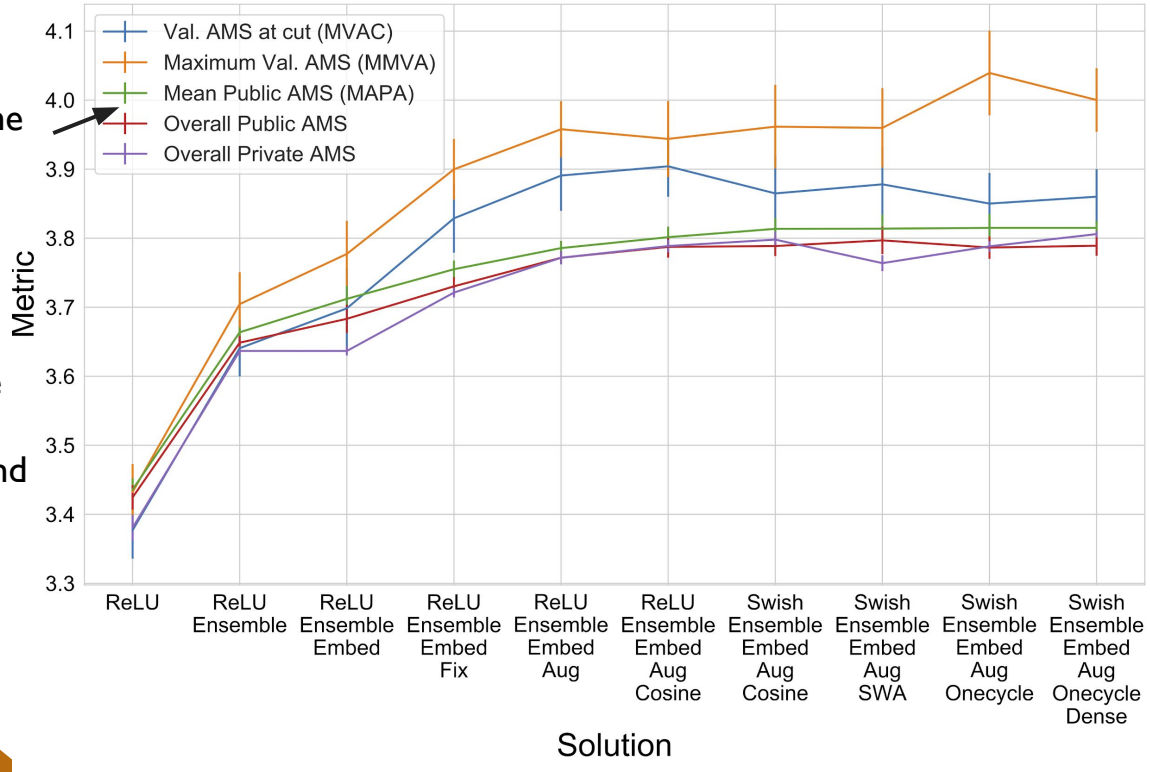
- Used cosine annealing and doubled the cycle-length with each restart
- Results in
  - Small performance improvement 3.79 ±0.01→3.80±0.02
  - Very large increase in train time (but still reasonable to use)



METRIC EVOLUTION

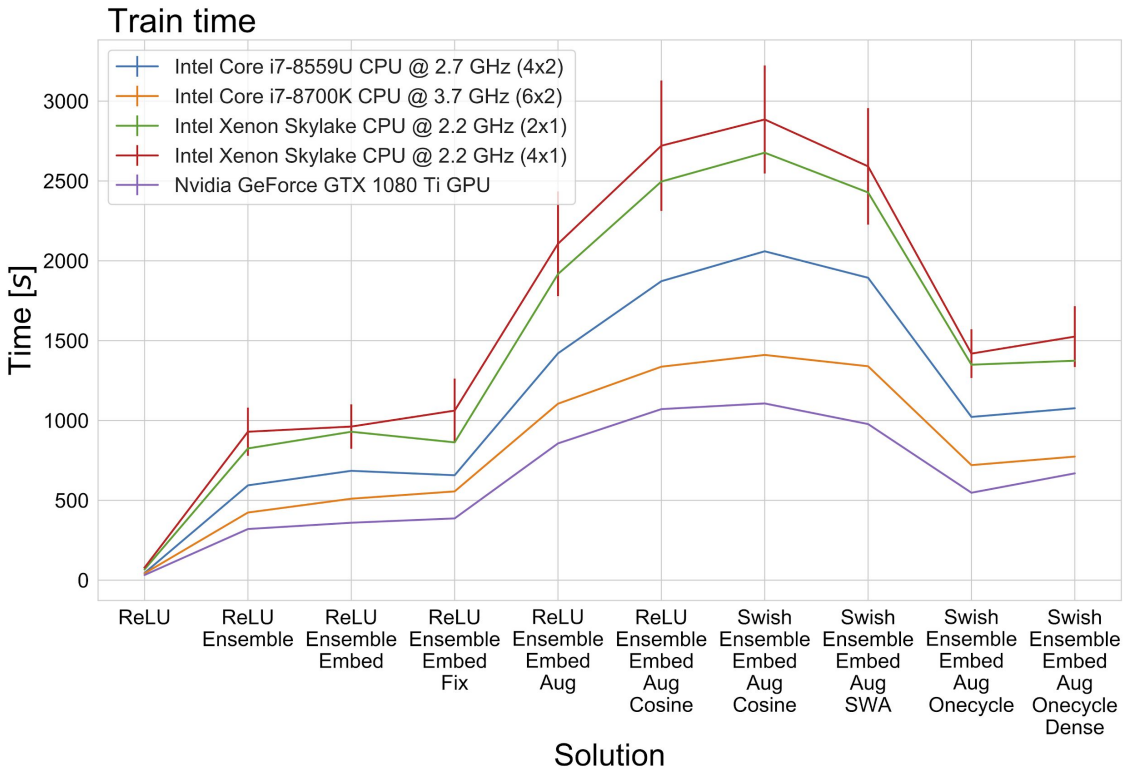
MAPA was the main optimisation metric

Overall Public|Private AMSs only checked at end



MVAC & MMVA were two other optimisation metrics, but were known to be optimistic

TRAINING TIME



TESTING TIME