Recent developments in deep-learning applied to open HEP data

Giles Strong

3rd ATLAS ML Workshop, CERN - 17/10/2018

giles.strong@outlook.com

twitter.com/Giles_C_Strong

Amva4newphysics.wordpress.com

github.com/GilesStrong

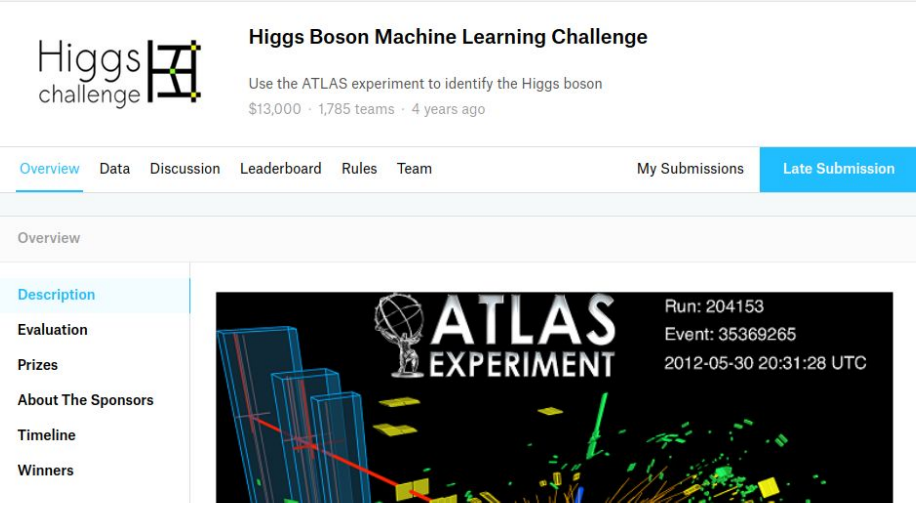
Introduction

ML in HEP and 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
- 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



Investigation overview

- 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
- I will be using it to demonstrate the cross-domain applicability of several recent methods:
  - A method of quickly optimising the learning rate
  - Two recent activation functions
  - Learning rate scheduling
  - Data augmentation
  - New ensembling techniques (in backup slides)

Basic information

Dataset description, evaluation metric, and basic classifier

Higgs ML dataset

- ATLAS 2012 MC full simulation with Geant 4
- Signal: Higgs to di-tau
- Backgrounds: $Z \rightarrow \tau\underline{\tau}$, $t\underline{t}$, and W decay
- Events selected for the semi-leptonic channel: $\tau\tau \longrightarrow (e \mid \mu) + \tau_{\mathrm{h}}$
- 250,000 labelled events for training, 550,000 unlabelled events for testing
- 31 features:
  - 3-momenta of main final-state and upto two jets (p_T ordered)
  - High-level features: angles, invariant masses, fitted di-tau mass (MMC), et cetera

Challenge aim

- Solutions must predict signal or background for each test event
- Solutions ranked via their [Approximate Median Significance]()
  - Quick, accurate, analytical approximation of full discovery significance
  - *s* = sum of weights of true positive events (signal events determined by the solution to be signal)
  - *b* = weights of false positive events (backgrounds events determined by the solution to be signal)
  - b_r = constant term (set to 10 for the challenge)

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

Classifier description

- The basic classifier I use is a 4-layer, fully connected network trained using Adam to minimise the sample-weighted binary cross-entropy of event class predictions
- An ensemble of 10 networks is trained on 80% of the training data
- The remaining 20% is used to compare architectures and optimise the threshold needed to classify the unlabelled test data
- The code used is available here, along with Docker and Binder instructions - (tag 1.0 = stable, reproduces results here)
- Relevant notebooks will be linked during the presentation

Method testing

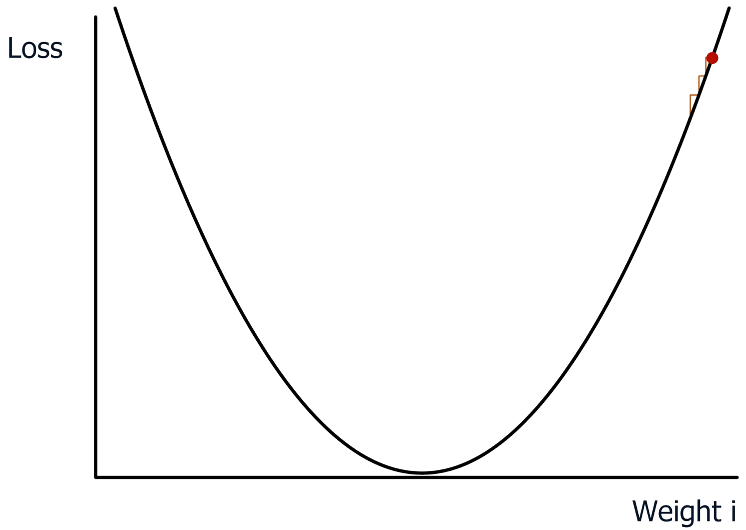
Learning rate finder

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

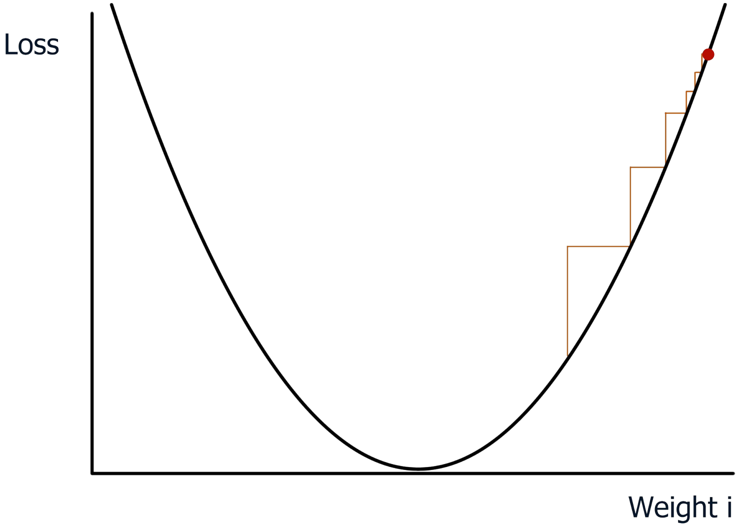
Learning rate finder

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



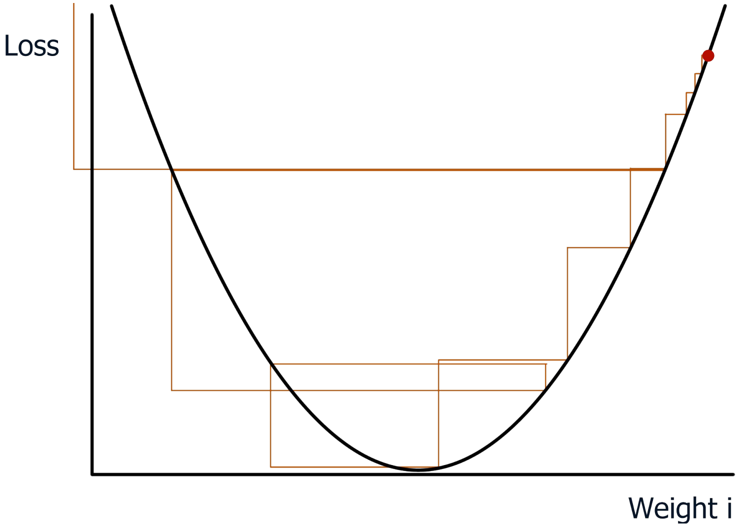
Learning rate finder

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



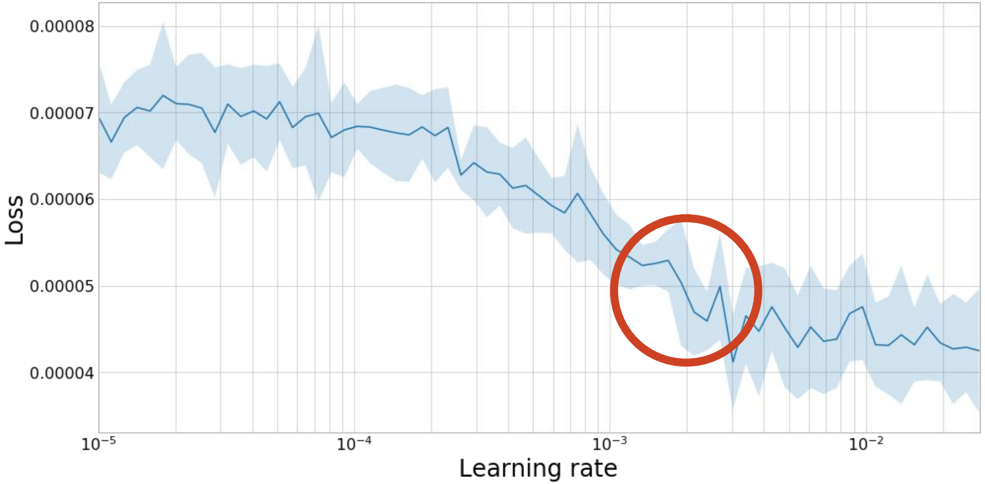
Learning rate finder

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)



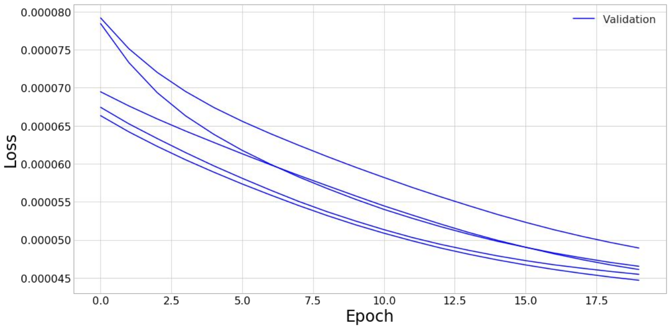
Learning rate finder

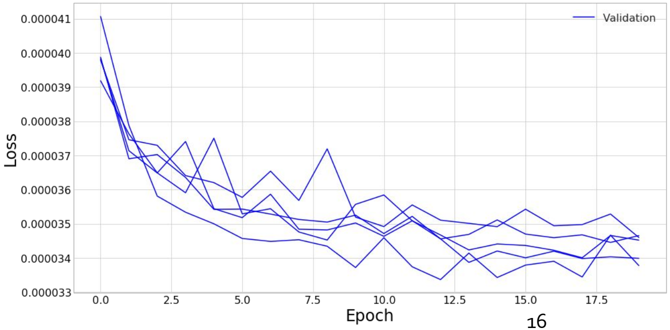
- The optimum LR is the highest LR at which the loss is still decreasing
- Further explanation in this lesson



Experiment

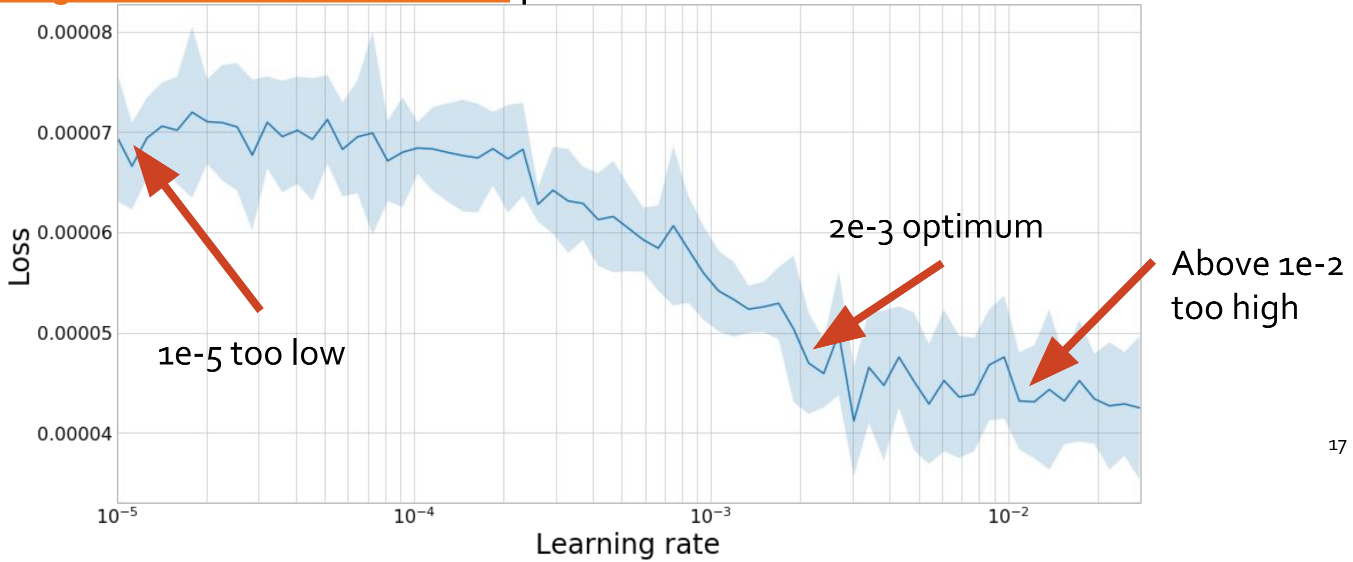
- Train classifier in cross-validation for three LR values (1e-5, 1e-3, & 1e-1) for fixed number of epochs
- Examine rate of convergence and mean AMS
- 1e-5 too slow for training, AMS = 1.97
- 1e-1 too large to converge, AMS = 1.07
- 1e-3 about right, AMS = 3.26





Experiment

- Optimum LR as found using LR finder is compatible with experiment
- Link to experiment notebook
- Full training with of a ReLU-based model produces a validation AMS of 3.72

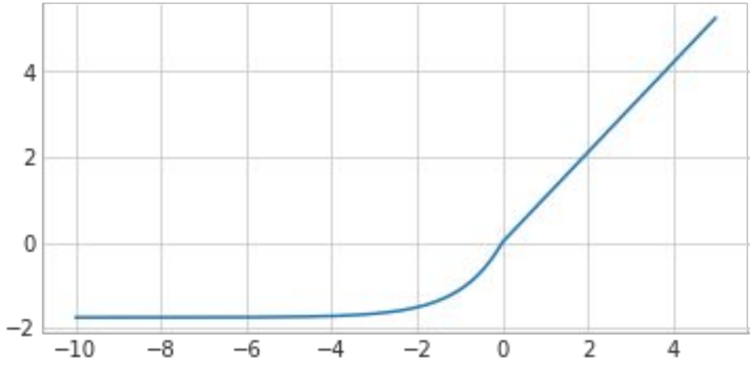


Method testing

Activation functions

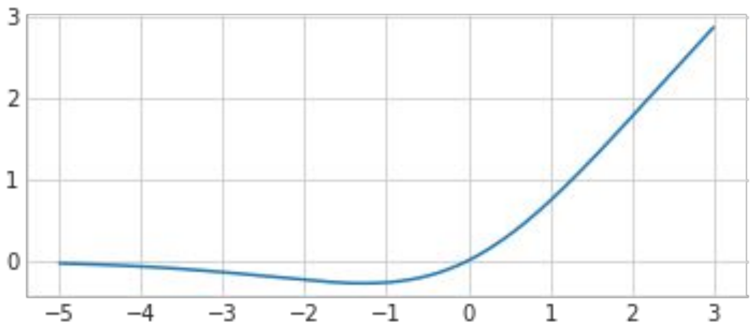
Choice of activation function

- Rectified linear unit appears to be the default choice in contemporary DL
- Several modifications and new activations have been proposed in recent years
- The Scaled Exponential Linear Unit (SELU) (Klambauer et al., 2017) allows networks to *self-normalise* without need of batch normalisation
- The paper demonstrates applicability to wide range of tasks



Choice of activation function

- The Swish activation function (Ramachandran et al., 2017) also shown to provide incremental improvement over other activation functions
- The paper reports results for image classification and language translation, but suggests is can be used inplace of ReLU in any NN



Experiment

- Train classifiers in CV for fixed number of epochs
- Weight initialisation scheme set for each activation function
- LR Finder used to optimise LR for each activation function
- Mean AMS:
  - ReLU: 3.28
  - SELU: 3.18
  - Swish: 3.45
- Link to comparison
- Full training with Swish produces a validation AMS of 3.78

Method testing

Learning-rate schedules

Learning-rate cycles

- 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, which can sometimes lead to *super convergence* (Smith 2017)
- Smith 2018 introduces the 1cycle schedule which further improves the super convergence
- All three papers demonstrate on image classification problems

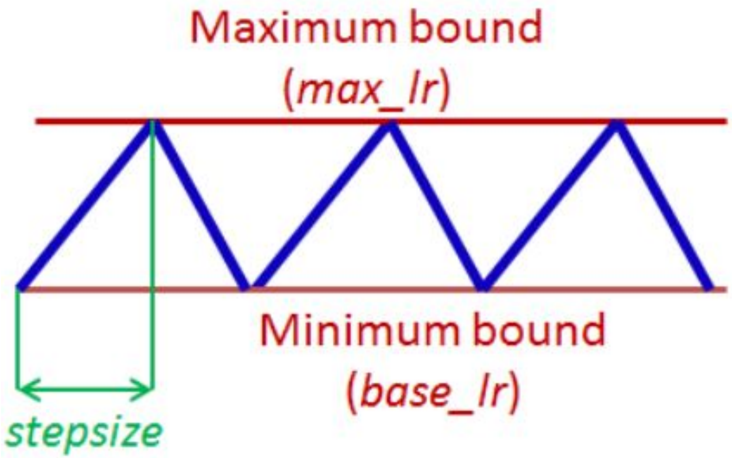
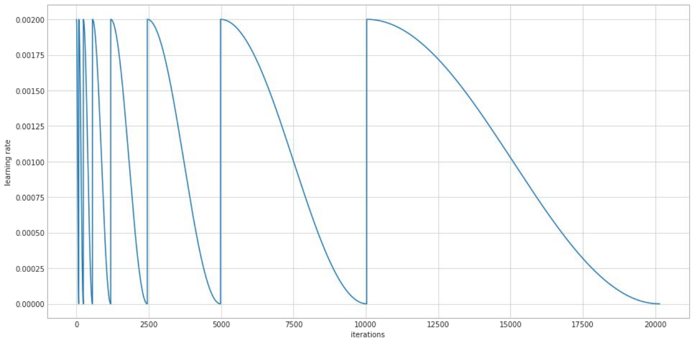
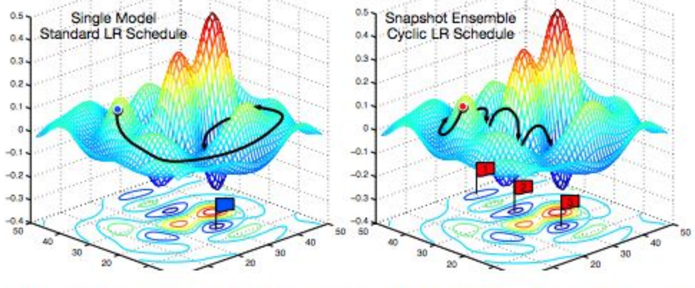


Figure - Smith, 2015, arXiv:1506.01186

Learning-rate cycles

- 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

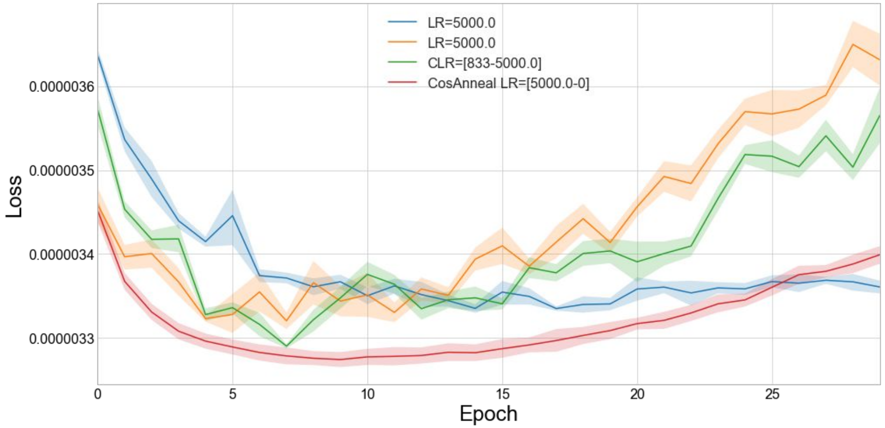




Lower figure - Huang et al., 2017, arXiv:1704.00109

Experiment

- A previous experiment comparing the use of different learning rate schedules indicated that the cosine annealing with restarts provide better performance
- The experiment here showed only minor improvements using the cosine annealing
- Validation AMS drops slightly (3.78->3.77) but other improvements seen in training and validation metrics

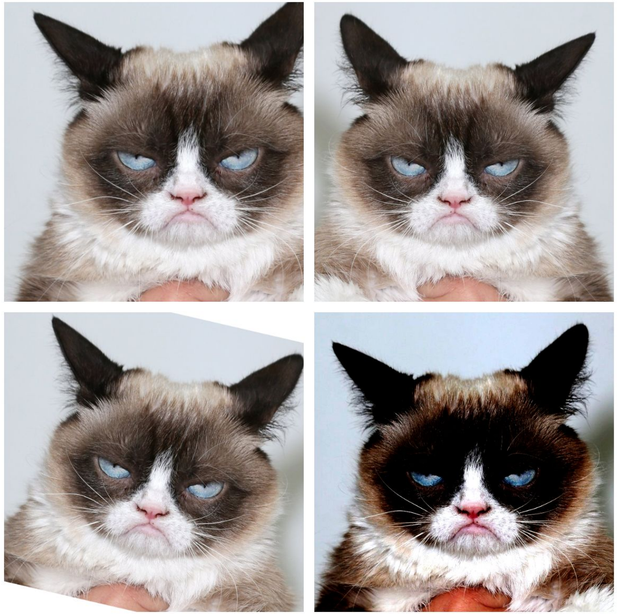


Method testing

Data augmentation

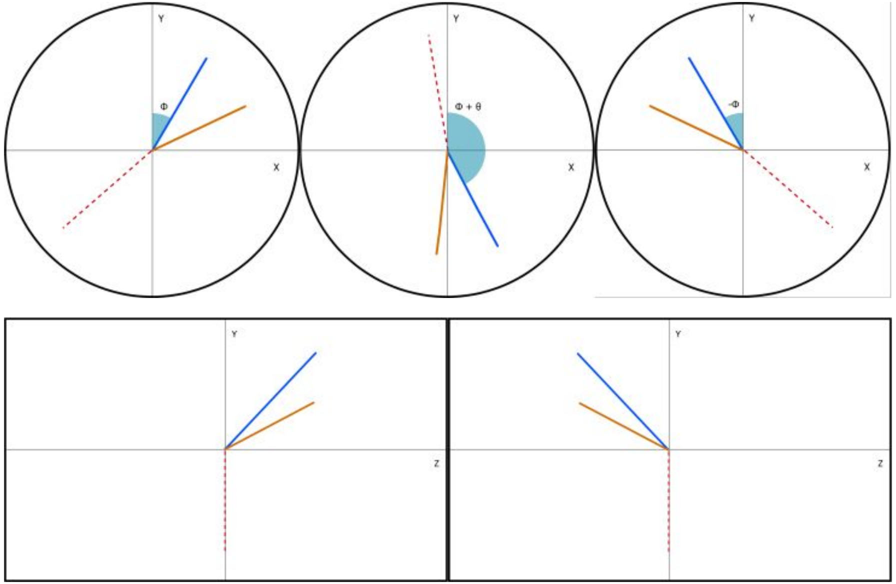
Data augmentation

- 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



Experiment

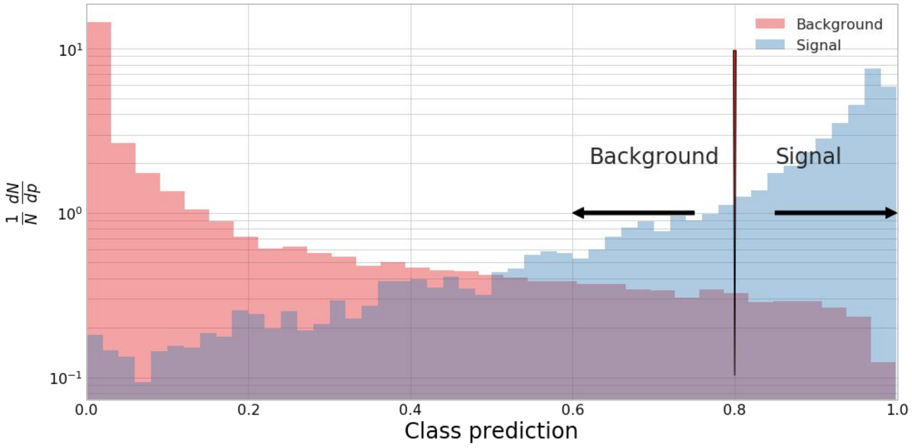
- Train-time data augmentation is implemented here by randomly rotating events in phi and randomly flipping in the Z and X-axes
- At test-time the mean prediction is taken over a set of 32 transformations corresponding to 8 phi orientations for each possible set of flips in Z and X
- Using data augmentation results in a very large improvement in validation AMS:
  - 3.97 when cosine annealing is used
  - 3.88 using a constant LR (confirming the hypothesis that the LR schedule improves performance)

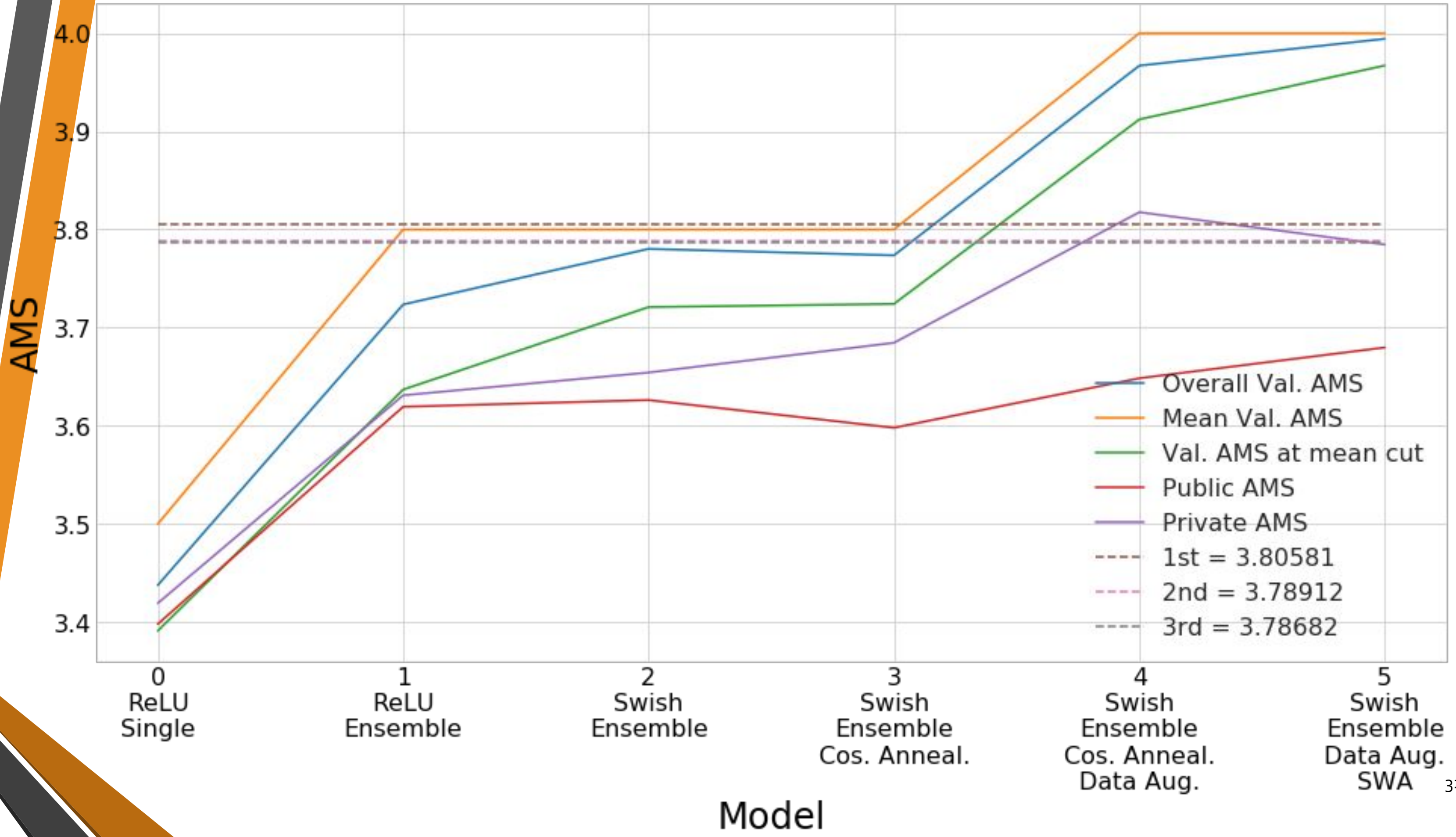
More in-depth explanation of HEP-data augmentation here

Comparison and conclusion

AMS evolution

- Cut on prediction computed by bootstrapping the validation data (20% of training set) 512 times and computing the mean optimum cut
- Can compute multiple AMSs:
    - Overall Val. AMS = maximum AMS on validation data
    - Mean Val. AMS = mean maximal AMS on bootstrapped validation data
    - Val AMS at Mean cut = AMS on validation data at bootstrap cut
    - Public AMS = AMS on public test set (18% of test set)
    - Private AMS = AMS on private test set (72% of test set)





AMS
4.0
3.9
3.8
3.7
3.6
3.5
3.4
Overall Val. AMS
Mean Val. AMS
Val. AMS at mean cut
Public AMS
Private AMS
1st = 3.80581
2nd = 3.78912
3rd = 3.78682
0 ReLU Single
1 ReLU Ensemble
2 Swish Ensemble
3 Swish Ensemble Cos. Anneal.
4 Swish Ensemble Cos. Anneal. Data Aug.
5 Swish Ensemble Data Aug. SWA
Model

Comparison of methods

Solution	New	1st place	2nd place	3rd place
Method	10 DNNs	70 DNNs	Large number of BDTs	108 DNNs
Train time	2 hours	24 hours	48 hours	3 hours
Inference time	1.5 hours	1 hour	???	20 minutes
Score	3.818	3.806	3.789	3.787
Hardware requirements	Intel i7-6500U <8 GB RAM (2016 laptop)	Titan GPU <24 GB RAM	>=8-core CPU >=64 GB RAM (m2.4.xlarge)	2012 quad-core laptop

Conclusion

- Even accounting for four years' worth of improvements in software and hardware, using the recent methods we are able to able to achieve similar performance to the winning solutions in a much quicker time
- Still, main improvements beyond finding decent LR, however, come from ensembling and data augmentation
- Data augmentation requires considering the symmetries of the inputs with respect to the classes, but is worth doing
- Fast Geometric Ensembling or Stochastic Weight Averaging could be promising methods of enesembling complex models with slow train time - see backup slides

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

Pre-processing steps

1. Infinities, NaNs, and -999 (default value for absent jets) values replaced with zeros
    - Prevents bias of later pre-processing steps
2. Vectors transformed to Cartesian coordinates
    - ϕ - cyclical and η - non-linear; NNs found to work best in fully-linear system
3. Random train-validation split, stratified by class
4. Standardisation and normalisation transformation fitted to training data, applied to training, validation and testing sets

Method testing

Stochastic weight-averaging

Fast ensembling

- Inspired by Loshchilov and Hutter 2016 (SGD with restarts via cosine annealing), Huang et al. 2017 showed that an ensemble of NNs may be built from a single training by saving a copy of the model before each restart (snapshot ensembling)
- Wilson et al. Feb. 2018 further improves on this idea by forcing the weight evolution along curves of constant loss which are found to connect loss minima (Fast Geometric Ensembling)
- FGE was found to outperform snapshot ensembling, but one still incurs increased inference time due to having to evaluate several models
- Wilson et al. Mar 2018 introduces a method which approximates FGE using a single model: stochastic weight-averaging

Stochastic weight averaging

- Previous ensembling methods took averages in *model-space*, SWA instead makes the ensemble purely in weight-space:
- It finds that (cyclical) SGD models reach regions of high performance, but never find the optimal point in terms of generalisation.
- (Fast Geometric) ensembling then works by moving the average prediction to the optimal point by averaging over models.
- SWA works by moving to the optimal point by directly averaging the weights

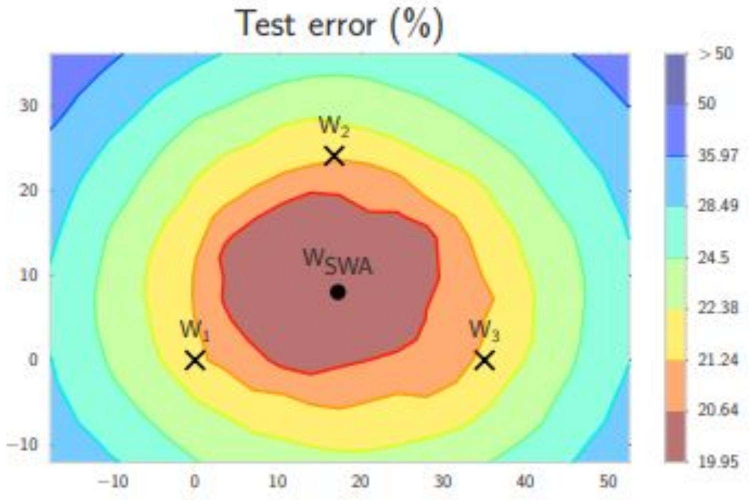
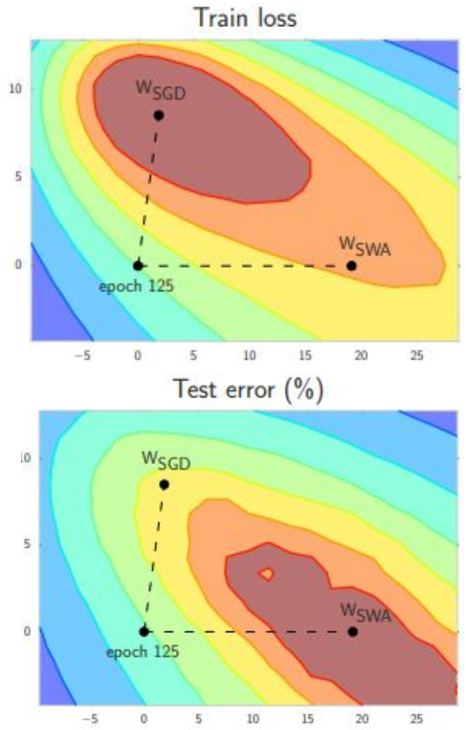


Figure - Wilson et al., Mar., 2018, arXiv:1803.05407

Stochastic weight averaging

- Training begins as normal
- Once the network begins to enter the region of high performance a copy of the weights is created
- The original model continues to train via SGD as normal but after each update, the new weights are added in a running average to the copy
- All though shown on image classification, the authors state that SWA is architecture agnostic



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

Experiment

- When activated SWA showed large decreases in validation-fold loss, and high suppression of statistical fluctuations
- The mean AMS during CV (4.04) and the overall AMS on the validation data (3.99) were the highest seen so far
- Running on the test data, showed large drops in performance, however
- N.B.: I experimented with various setups but the best one seemed to be starting SWA after a fixed number of epochs and to use a constant LR
- Link to experiment