Introduction to Machine Learning: Part II

Elham E Khoda

University of Washington

US ATLAS ML Training 2023

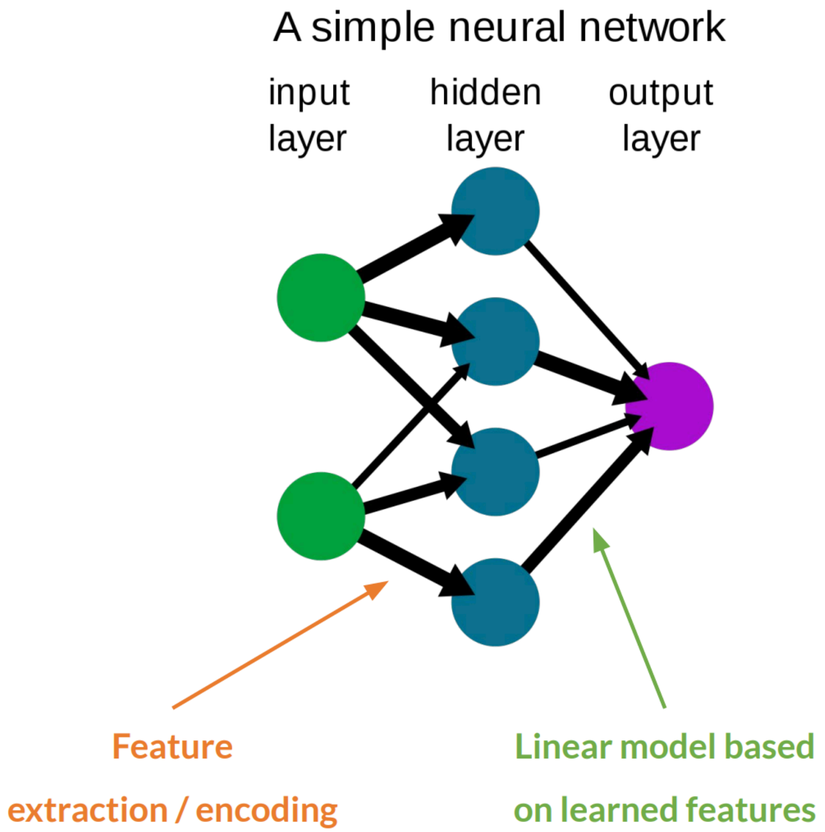
July 25, 2023

UNIVERSITY of WASHINGTON

Accelerated AI Algorithms for Data-Driven Discovery

ATLAS EXPERIMENT

Visualizing How NN works: A Simple Example



Conventionally, **feature extraction** is done by humans with domain- / problem-specific expertise

- Experience, a priori physics knowledge, etc.
- Expensive, difficult

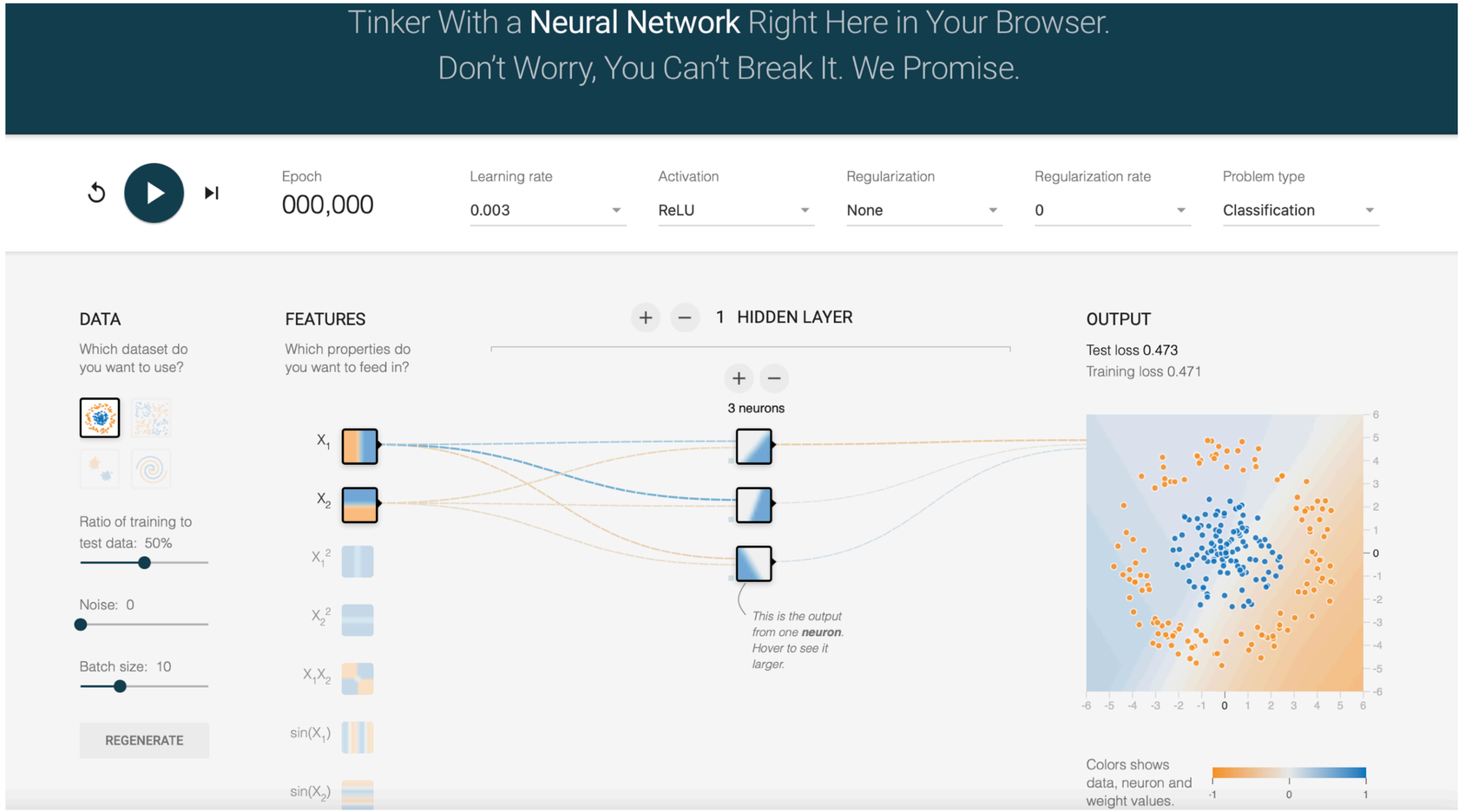
Linear models are easy to solve, but limited

- Closed form solution or convex optimization
- But linear, obviously…

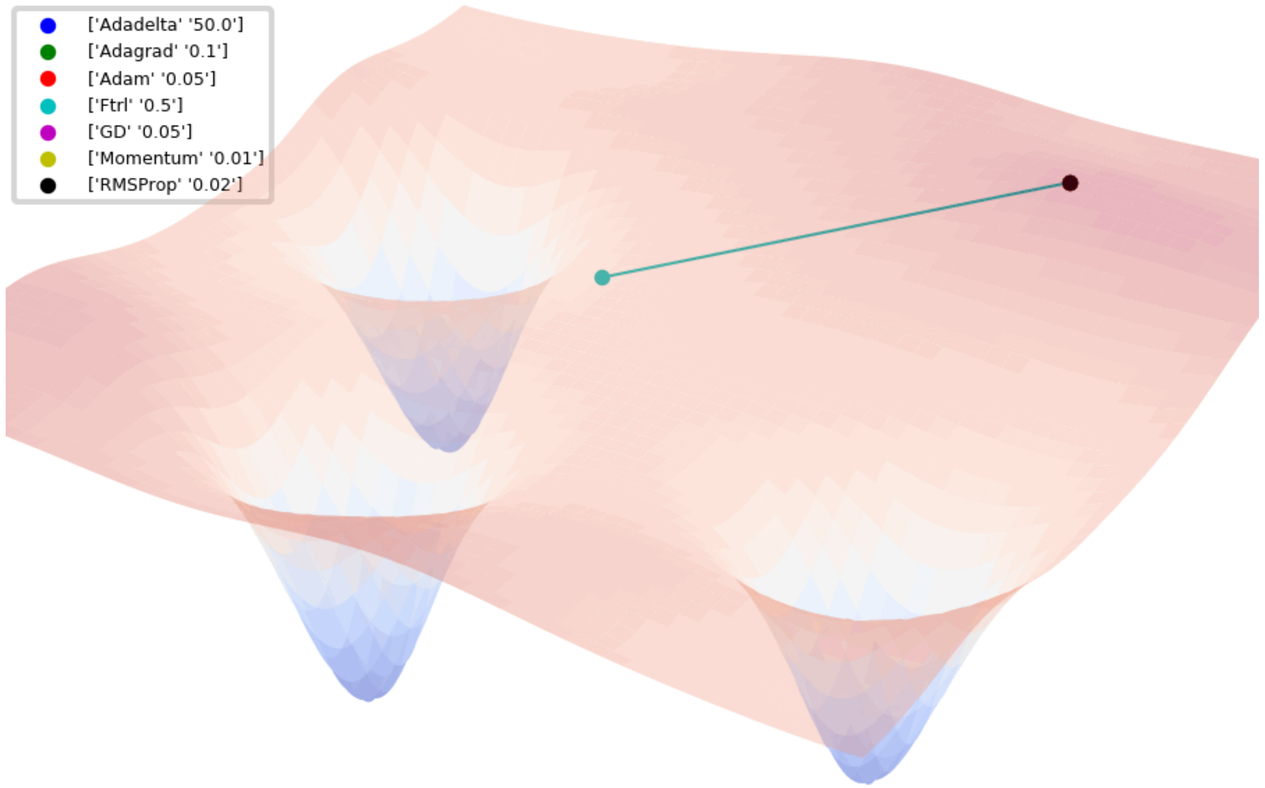
Neural Network Visualization

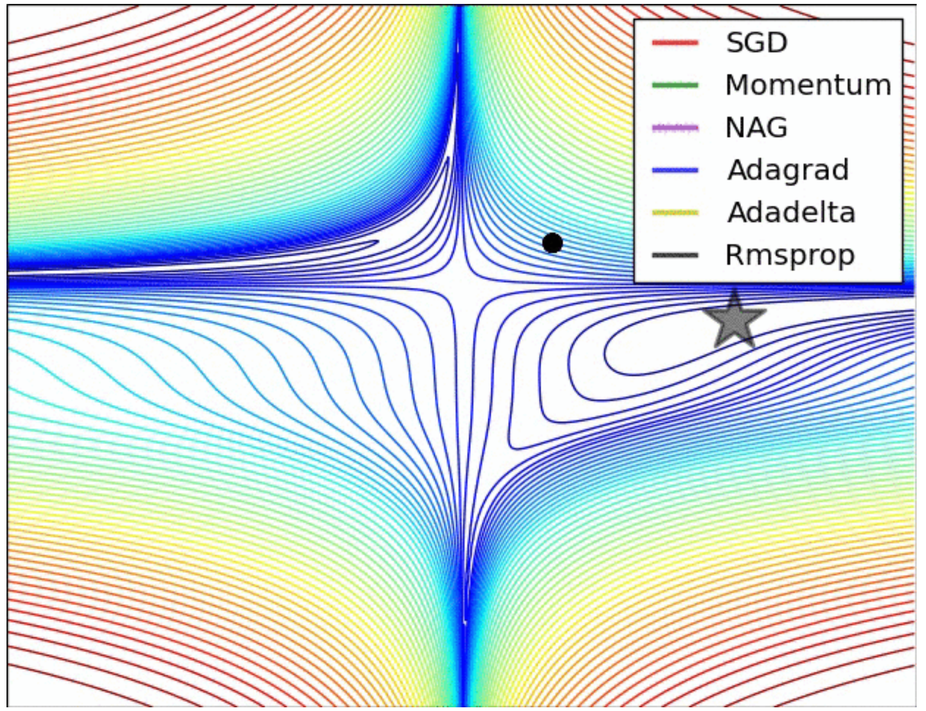
Links for visualization examples:

- Simple NN training live on browser: Link
- Visualizing hidden layers & decision boundaries: Link

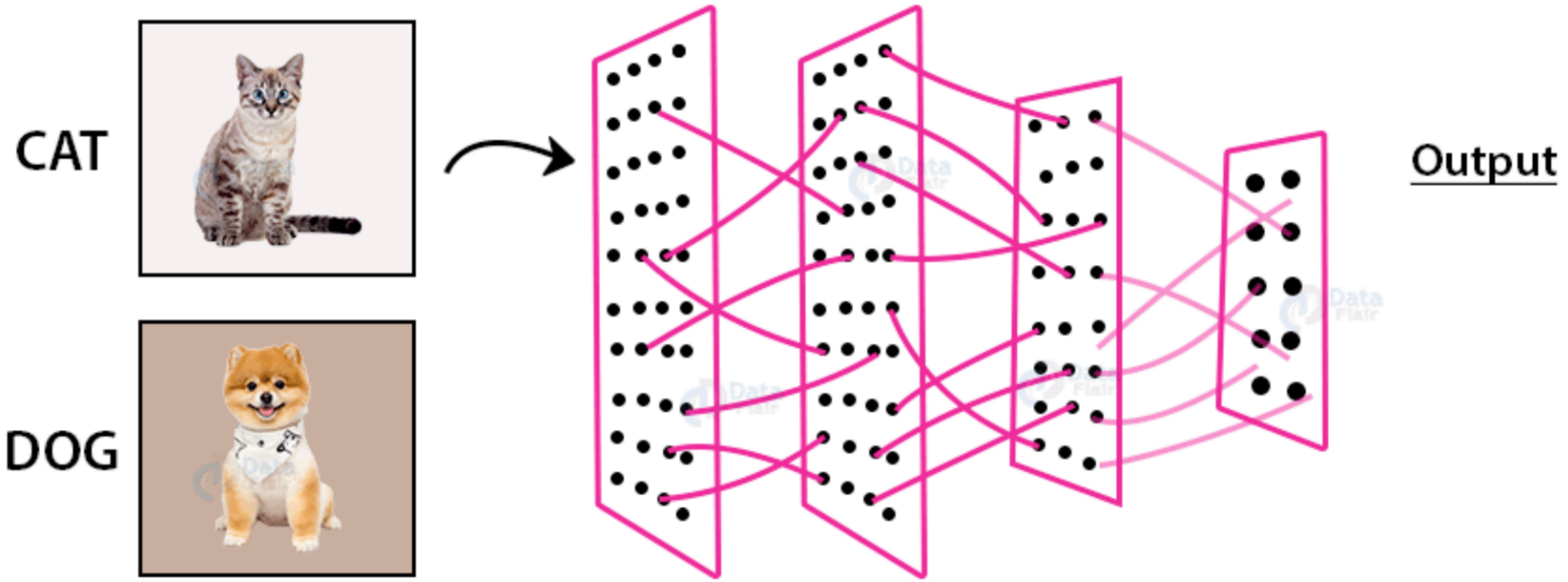


Optimizer: visualization





Classifier help categorize

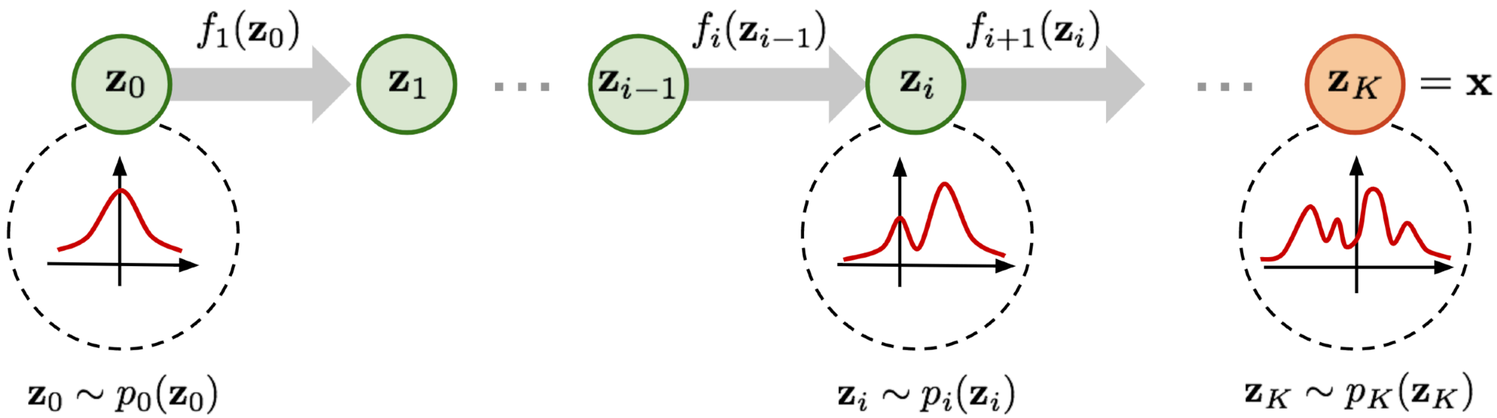


Likelihoods

Statistical inference is all about likelihoods, turns out ML is great at learning them !

$$P(data \mid theory)$$

Eg. Normalizing Flows, a class of generative models (see more on Wednesday)



Likelihood ratios

Generative models are hard to train, but often, all you need are likelihood ratios

- Hypothesis testing (H_0 vs H_1)
- Re-weighting

A classifier is all you need !

A neural network trained to classify between data from θ_0 and θ_1 approximates:

$$c(x_i, \theta_0, \theta_1) = \frac{P(x_i \mid \theta_1)}{P(x_i \mid \theta_0) + P(x_i \mid \theta_1)}$$

Neyman-Pearson Lemma

"Likelihood ratio is the best test statistic for hypothesis tests"

$$\lambda(x, \theta_0, \theta_1) = \frac{p(x \mid \theta_0)}{p(x \mid \theta_1)}$$

$$c(x_i, \theta_0, \theta_1) = \frac{P(x_i \mid \theta_1)}{P(x_i \mid \theta_0) + P(x_i \mid \theta_1)}$$

$$\lambda(x_i, \theta_0, \theta_1) = \frac{1}{c(x_i, \theta_0, \theta_1)} - 1$$

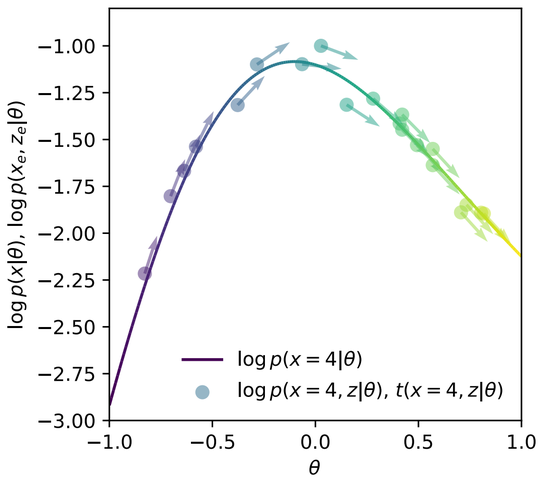
We get a likelihood ratio *per event*, unbinned hypothesis testing ! (See Aishik's ATLAS **work** on this)

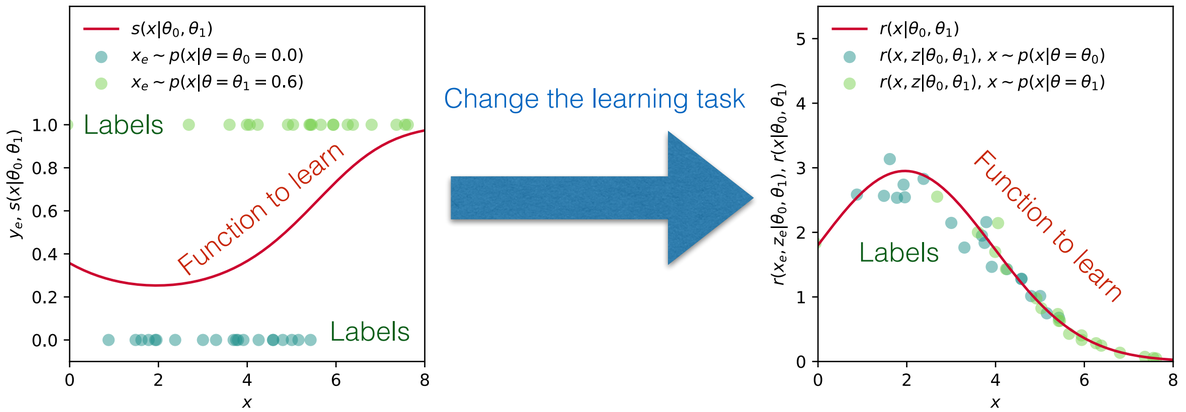
Simulator-Based Inference

Amortised: In histogram analyses, statistical fits are computationally expensive, in SBI the inference is lightning fast! Computational cost to train on simulation, but fast to evaluate on data.

For trustable likelihood ratios:

- Train large ensemble of networks, average the outputs
- Matrix element amplitudes as target labels
- Gradient information as auxiliary tasks



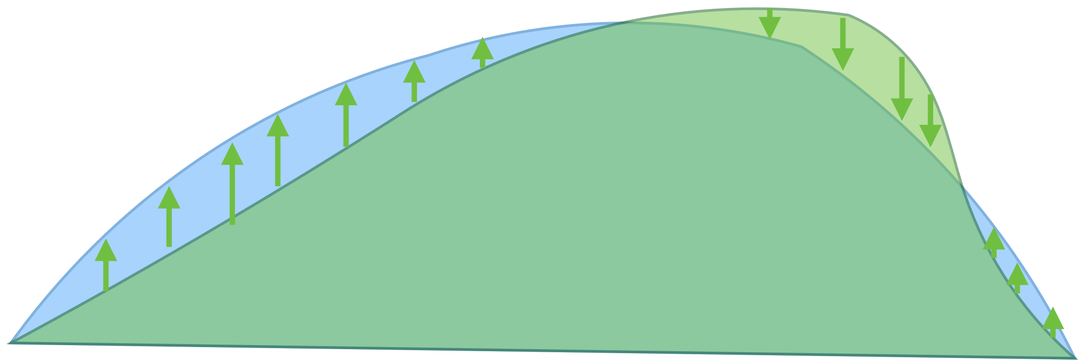


Brehmer et al.

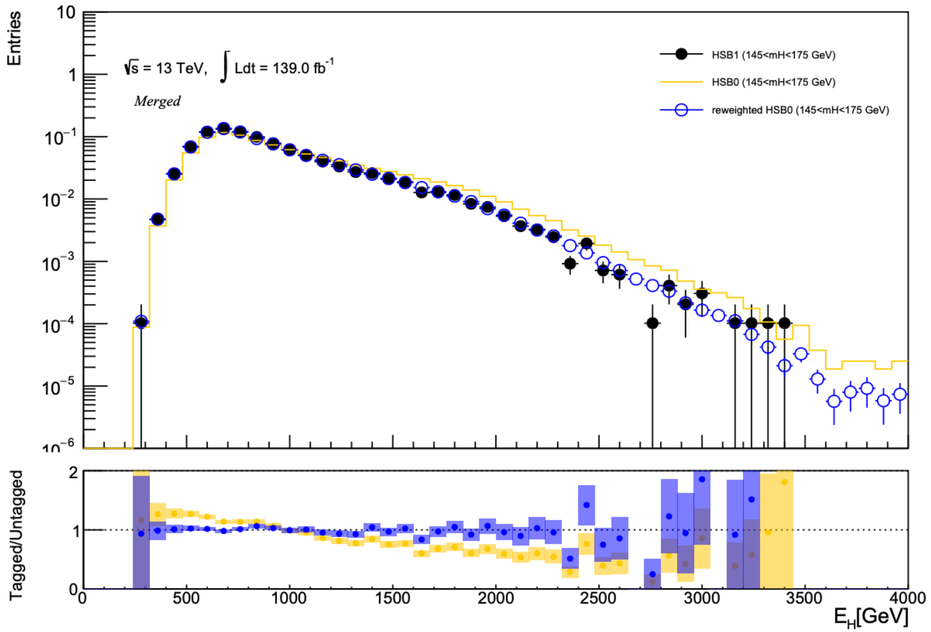
Re-weighting

Likelihood ratios let you re-weight samples, without binning !

Simple idea powering the next generation of ML for HEP tools like multi-dimensional, unbinned unfolding (see more on Thursday)



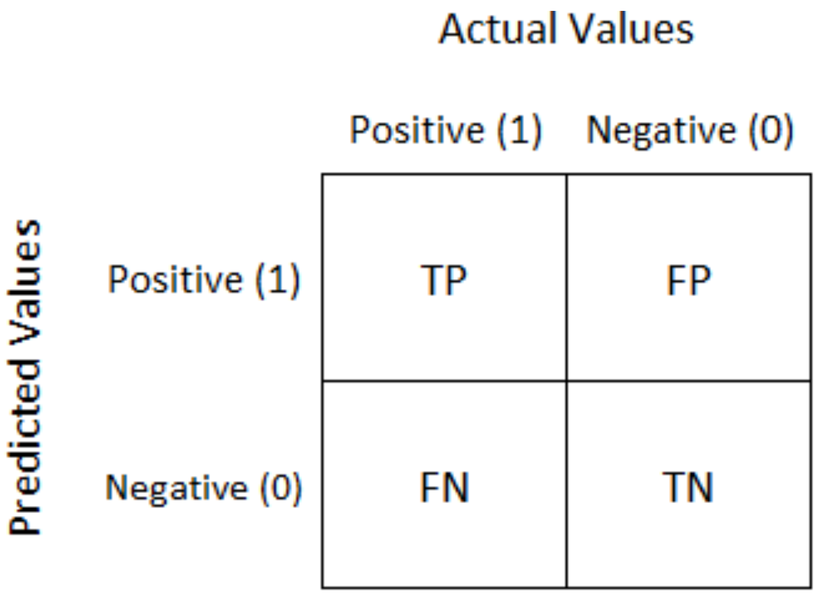
NN-based reweighting in ATLAS analysis



Classifier Confusion Matrix

Let's consider binary classification:

- Two classes 0 and 1
- *Confusion matrix* is a 2x2 table

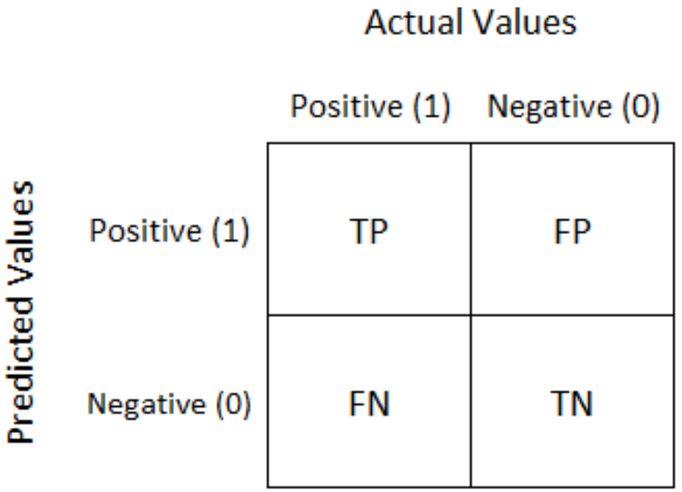


Actual Values: True/False
Predicted values: Positive/Negative

If a model predicts probabilities instead of class labels(0&1) then it is crucial to choose a reasonable threshold

ROC and AUC

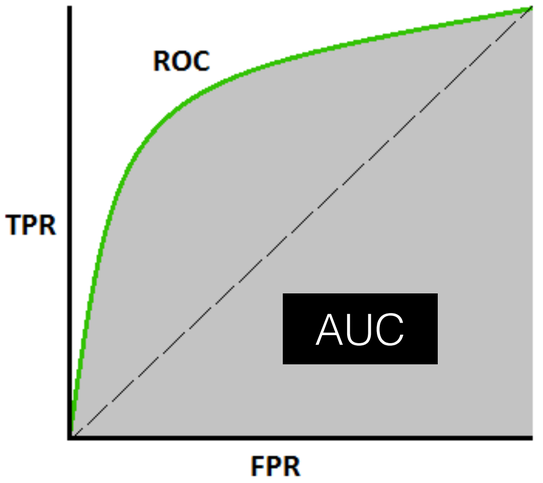
- ROC = Receiver Operating Characteristics
- AUC = Area under the Curve
- Generally used when the output is a probability, [0,1]



True Positive Rate =

False Positive Rate =

AUC & ROC curve is a performance measurement for the classification problems at various threshold settings



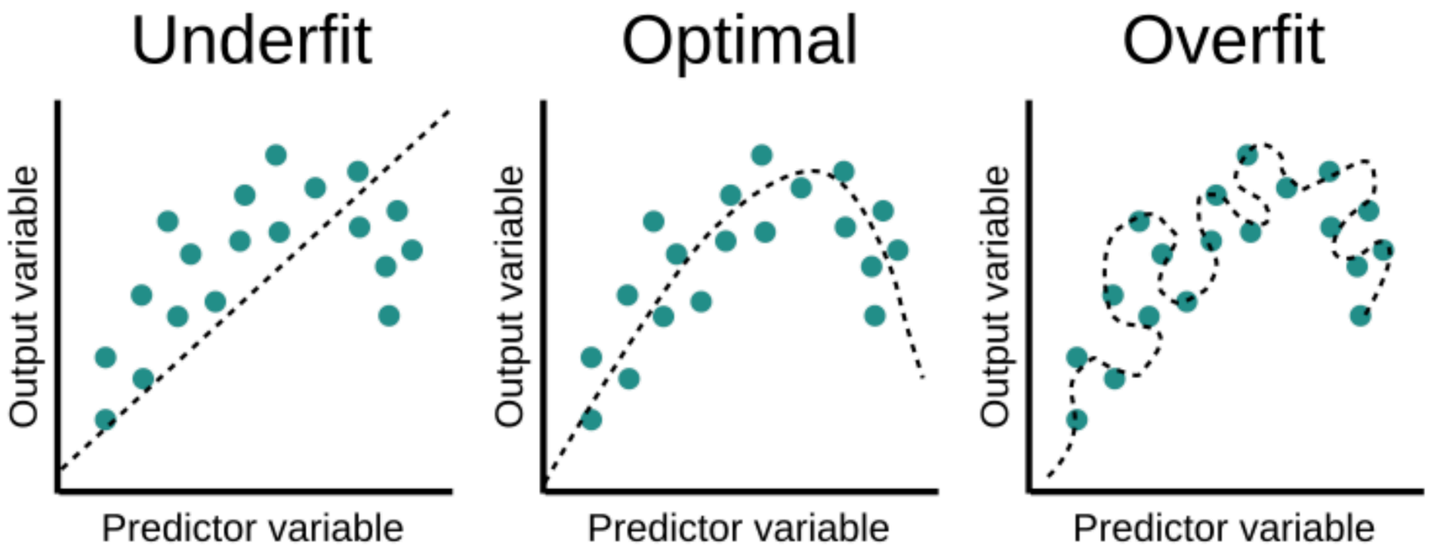
AUC - ROC Curve [Image 2] (Image courtesy: My Photoshopped Collection)

Underfitting and Overfitting

A model with “high bias” pages very little attention to the training data and over simplifies the model

Performs poorly on training as well as testing data

➡ **Underfitting**



Easy to solve this problem!

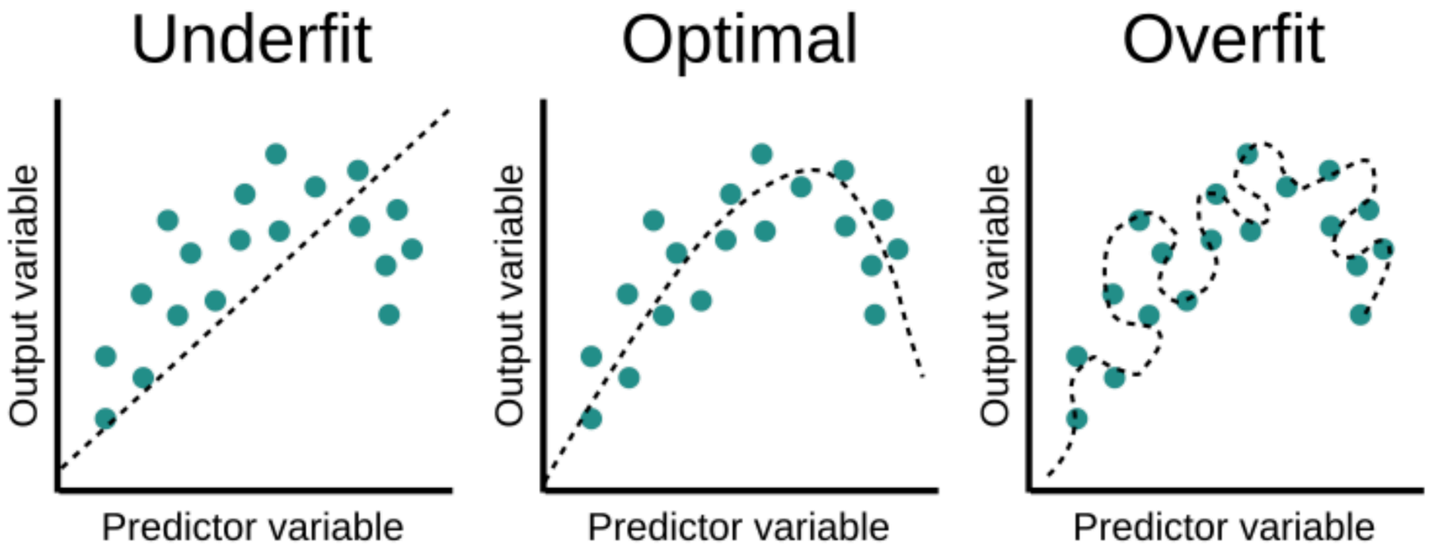
- Increase model complexity
- Increase the number of features, performing feature engineering
- Remove noise from the data
- Increase the number of epochs or increase the duration of training to get better results

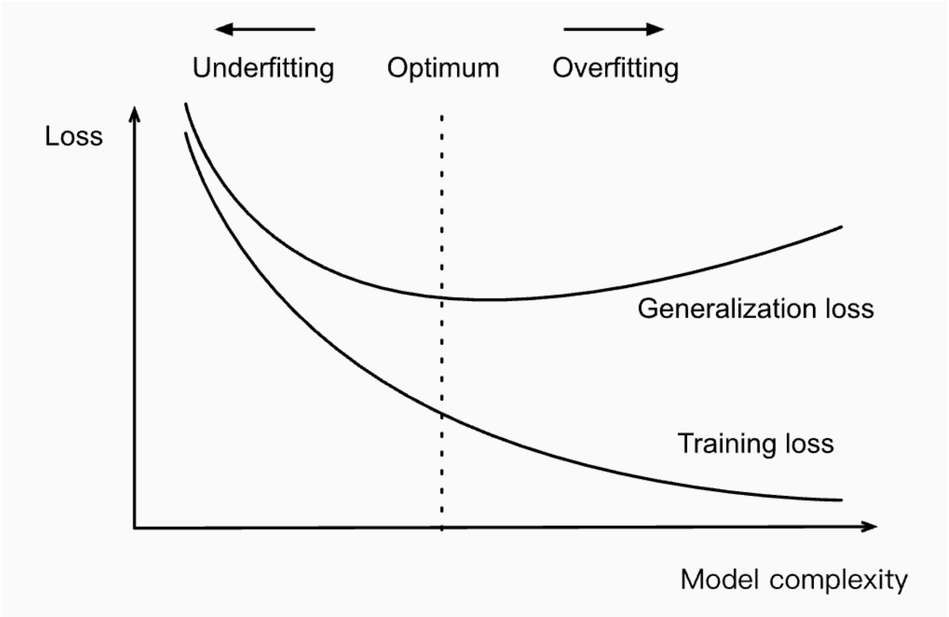
Underfitting and Overfitting

A model fits training data so well that it leaves little or no room for generalization over new data

We say that the model has "*high variance*"

➡ **Overfitting**





Why Overfitting?

A model fits training data so well that it leaves little or no room for generalization over new data

We say that the model has "*high variance*"
➡ **Overfitting**

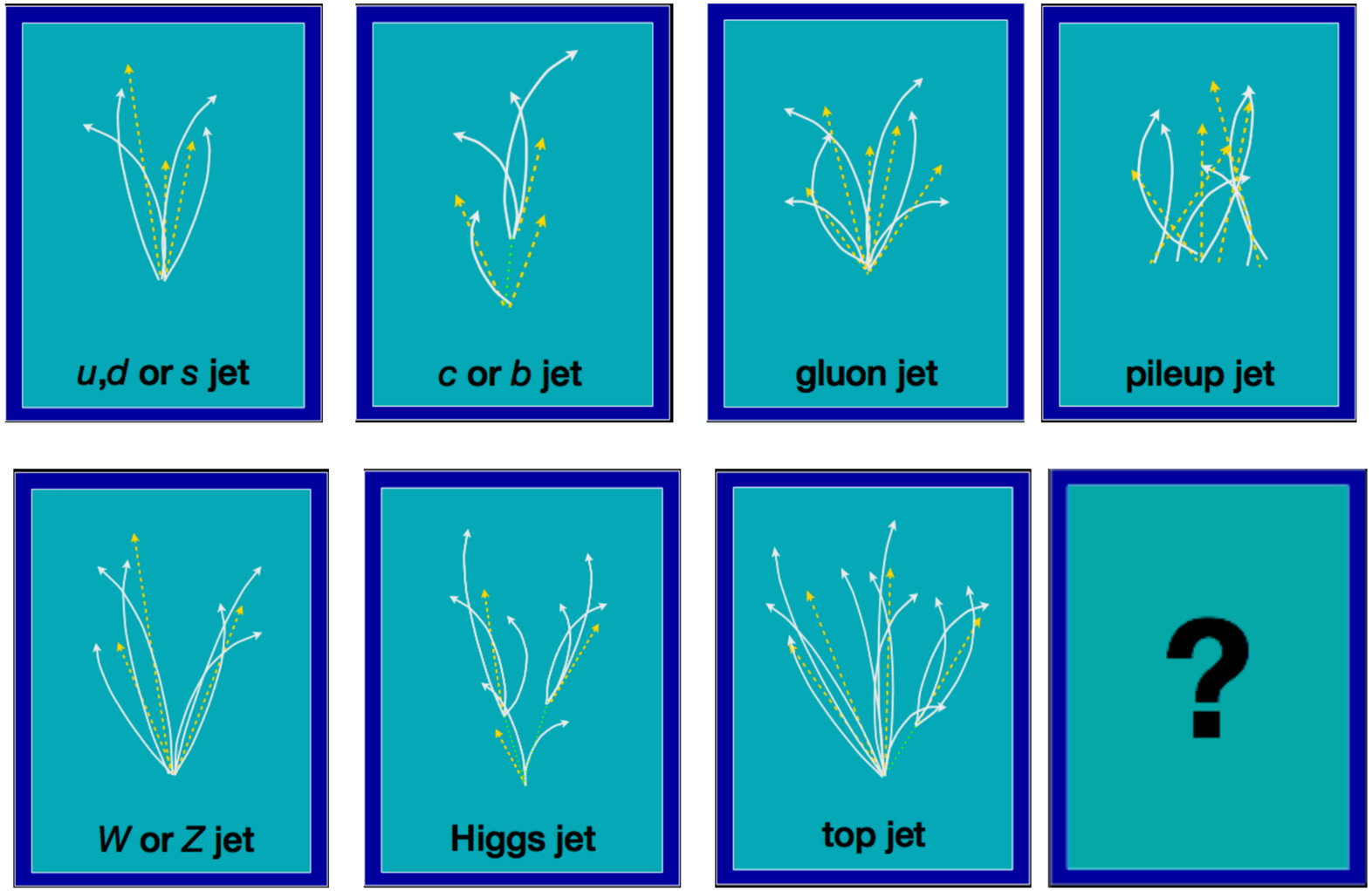
- **Small training data:** Does not contain enough data samples to accurately represent all possible input data values
- **Noisy data:** The training data contains large amounts of irrelevant information
- **Long training:** The model trains for too long on a single sample set of data
- **High model complexity:** It learns the noise within the training data

Lets practice a bit!

Update the git repo:
https://github.com/usatlas-ml-training/lbnl-2023/

Lets use the notebook from here:
https://github.com/usatlas-ml-training/lbnl-2023/tree/main/intro_lecture2

Jet Classification Example



How to Prevent Overfitting?

These are some popular techniques

Early Stopping
Pauses the training phase before the machine learning model learns the noise in the data

Reduce the network's capacity
By removing layers or reducing the number of elements in the hidden layers

Regularization
collection of training/optimization techniques that try to eliminate factors that do not impact the prediction outcomes

Data Augmentation
Large dataset will reduce overfitting. Data augmentation helps to increase the size of the dataset

Regularization: L1/L2

Regularizer on the NN weights (absolute values of weights / squared weights)

L2 regularization is perhaps the most common form of regularization

For every weight, w, in the network we add $\frac{1}{2}\lambda w^2$ to the objective

where λ is the regularization strength

L1 regularization is another relatively common form of regularization, where for each weight we add the term $\lambda \mid w \mid$

It is possible to combine the L1 regularization with the L2 regularization: $\frac{1}{2}\lambda w^2 + \lambda \mid w \mid$

Regularization: Dropout

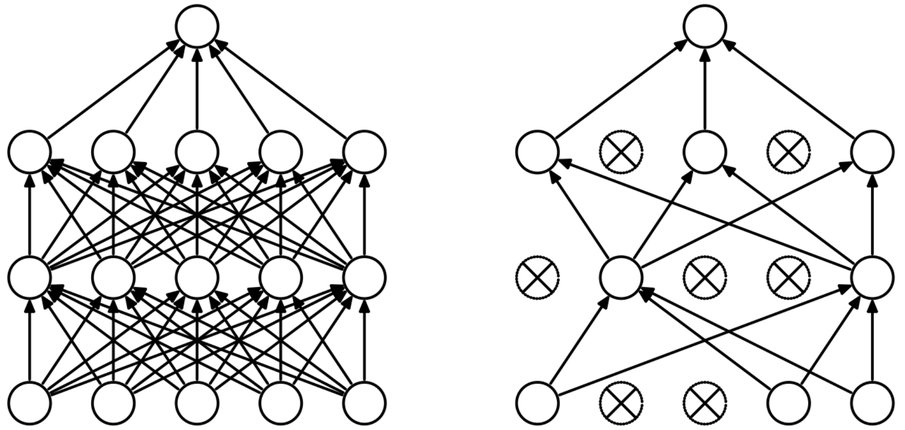
Dropout is an extremely effective, simple and recently introduced regularization technique by Srivastava et al. in

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

that complements the other methods

During Training:

Dropout is implemented by only keeping a neuron active with some probability p (a hyperparameter), or setting it to zero otherwise



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Righ**
An example of a thinned net produced by applying dropout to the network on the le
Crossed units have been dropped.

Hyperparameter Tuning (1/4)

Hyperparameter tuning playbook by google

Choose the model architecture

Summary: *When starting a new project, try to reuse a model that already works.*

- Choose a well established, commonly used model architecture to get working first
- Try to find a paper that tackles something as close as possible to the problem at hand

Choosing the optimizer

Summary: *Start with the most popular optimizer for the type of problem at hand.*

Stick with well-established, popular optimizers, especially when starting a new project

Well-established optimizers that we like include (but are not limited to):

- SGD with momentum (we like the Nesterov variant)
- Adam and NAdam, which are more general than SGD with momentum.
- Note that Adam has 4 tunable hyperparameters and they can all matter!

Hyperparameter Tuning (2/4)

Hyperparameter tuning playbook by google

Choose the batch size

Summary: *The batch size governs the training speed and shouldn't be used to directly tune the validation set performance. Often, the ideal batch size will be the largest batch size supported by the available hardware*

- The batch size is a key factor in determining the *training time* and *computing resource consumption*
- Increasing the batch size will often reduce the training time

→ Allows hyperparameters to be tuned more thoroughly within a fixed time interval

- The batch size should *not be* treated as a tunable hyperparameter for validation set performance
- For an optimized network, the same final performance should be attainable using any batch size (see Shallue et al. 2018)

Hyperparameter Tuning (3/4)

Hyperparameter tuning playbook by google

Choose the Choosing the initial configuration

Summary: *quickly determine the starting points with manual exploration then do a more through check*

- Before beginning hyperparameter tuning we must determine the starting point like

1. the model configuration (e.g. number of layers)
2. the optimizer hyperparameters (e.g. learning rate)
3. the number of training steps

Determining this initial configuration will require some manually configured training runs and trial-and-error.

Choosing the number of training steps involves balancing the following tension:

- Training for more steps can improve performance and makes hyperparameter tuning easier (see Shallue et al. 2018)
- Training for fewer steps means that each training run is faster, allowing more experiments to be run in parallel.

Hyperparameter Tuning

Several tools allow you to do hyper parameter scans and hyperparameter optimization

- Ray-tune 

- Weights & Bias Sweep 

- TensorBoard HParams 

- Keras Tuner
- Scikit-Optimize 
- Optuna 

All of these tools have grid search, random search and Bayesian Optimization implemented

Pick the one you like!

Tools for ML experiments visualization

You need to do some or a lot of experimenting with model improvement ideas

- Visualizing differences between various ML experiments becomes crucial

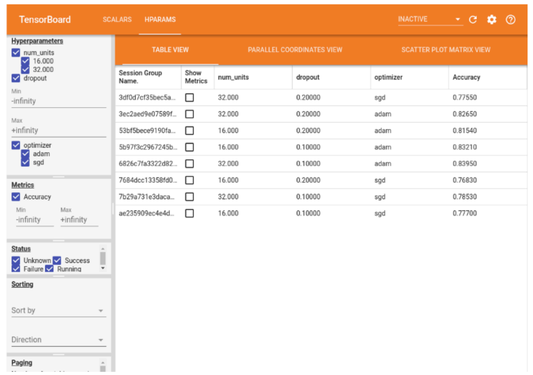
There are several popular tools tools: Weights & Biases, TensorBoard, Comet, MLflow etc

- Tracking and visualizing metrics such as loss and accuracy
- Monitor learning curves
- Visualize CPU/GPU utilization

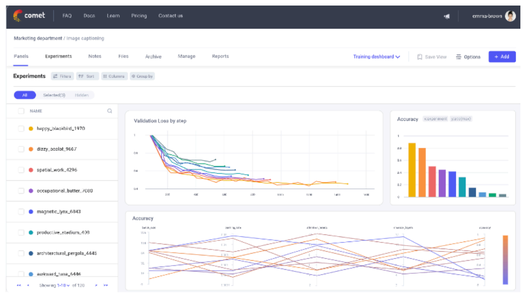
Weights & Biases

Weights & Biases

TensorBoard



Comet

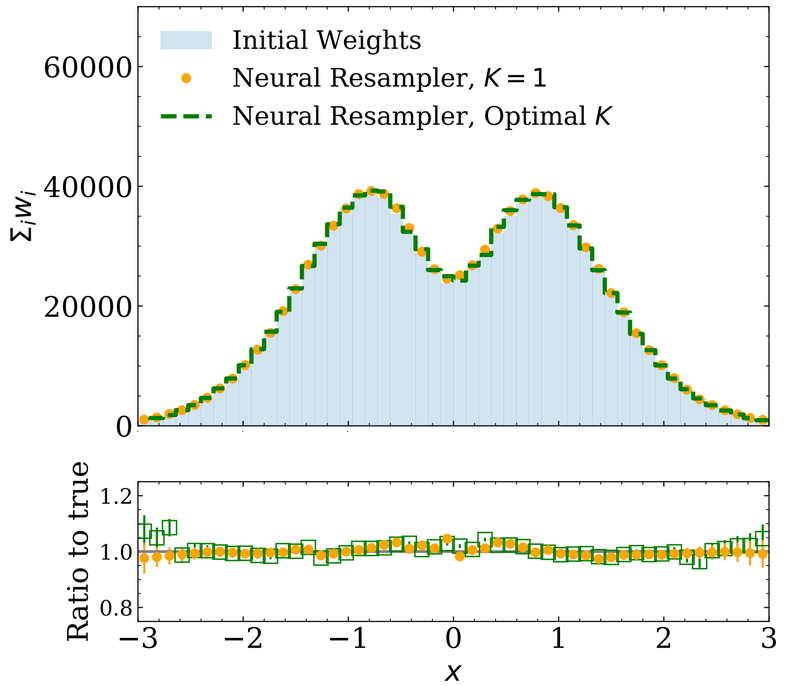


mlflow

for managing the end-to-end machine learning lifecycle

Negative Event Weight

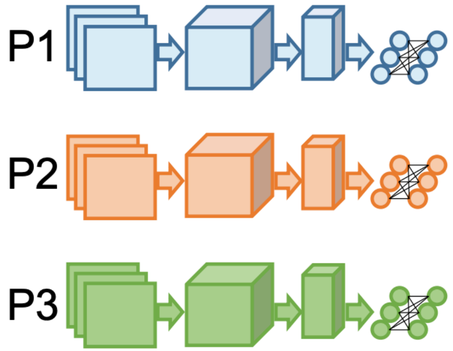
- Only certain BDT packages can handle negative weighted events (LightGBM yes, XGBoost no)
- For NNs, negative weights make logical sense, loss is multiplied by a negative weight and everything works as you would expect. So use your negative sample weights!
- Javier's slide: page 37 (Backpropagation)
- The more challenging problem: very large variance of weights (by orders of mag)
- Often, you can even re-weight them with ML to get rid of negative weights! Neural Resampler, Unweighting with generative models



Thank You!

Extra Slides

Deep Learning parallelization strategies

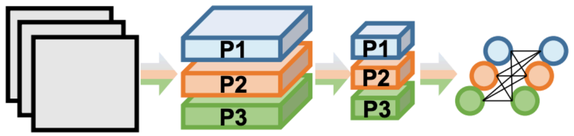


Data Parallelism

Distribute input samples

Model replicated across devices

Most common

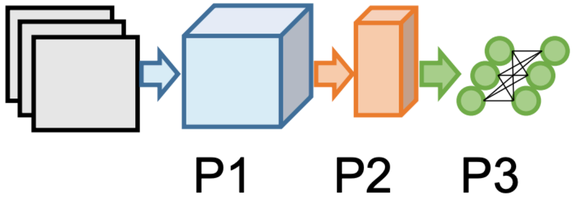


Model Parallelism

Distribute network structure (layers)

Needed for massive models that don't fit in device memory

Becoming more common



Layer Pipelining
Partition by layer

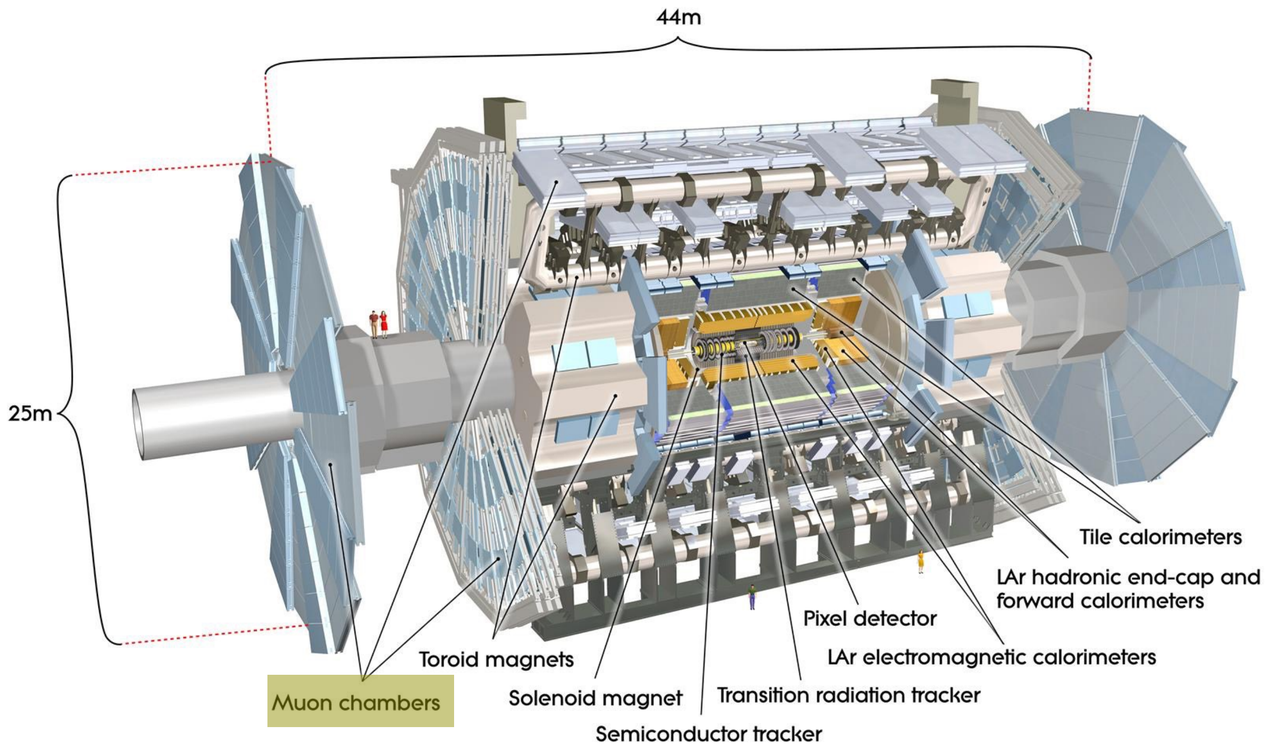
arxiv:1802.09941

The ATLAS Experiment

General purpose detector

Toroidal Magnet: 0.5 T

Muon Spectrometer:
Four different detector technology

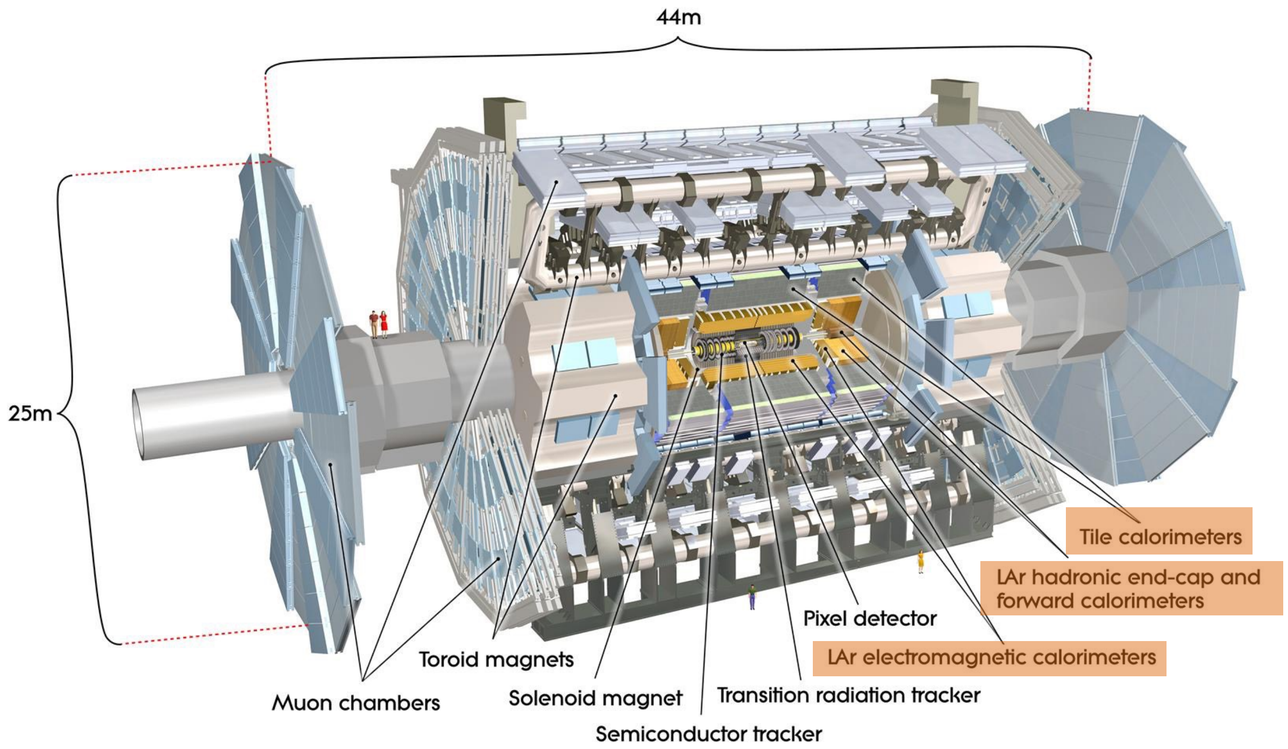


The ATLAS Experiment

General purpose detector

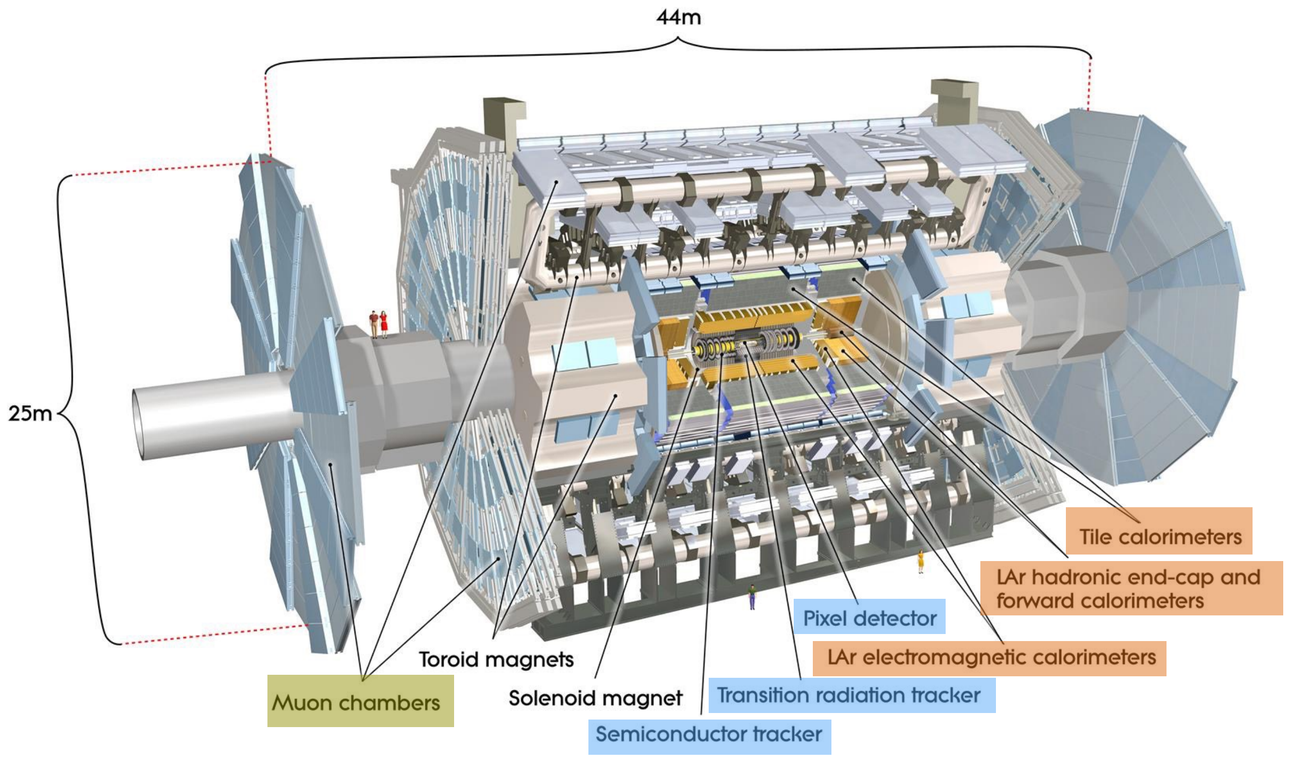
Calorimeter:
Electromagnetic (Liquid Argon), Hadronic (Liquid Argon (endcap) & Tile (barrel))

Solenoid Magnet: 2.0 T



The ATLAS Experiment

General purpose detector



Muon Spectrometer:

Four different detector technology

Calorimeter:

Electromagnetic (Liquid Argon), Hadronic (Liquid Argon (endcap) & Tile (barrel))

Solenoid Magnet: 2.0 T

Inner Detector:

Three different detector technology

1. Silicon Pixel
2. Silicon Strip
3. Straw Tubes: Transition Radiation Tracker (TRT)