Domain Adaptation via Histogram Loss Component

by Hector Castro Noguez
Mentors: Samuel May and Indara Suarez

Machine Learning in Simulation

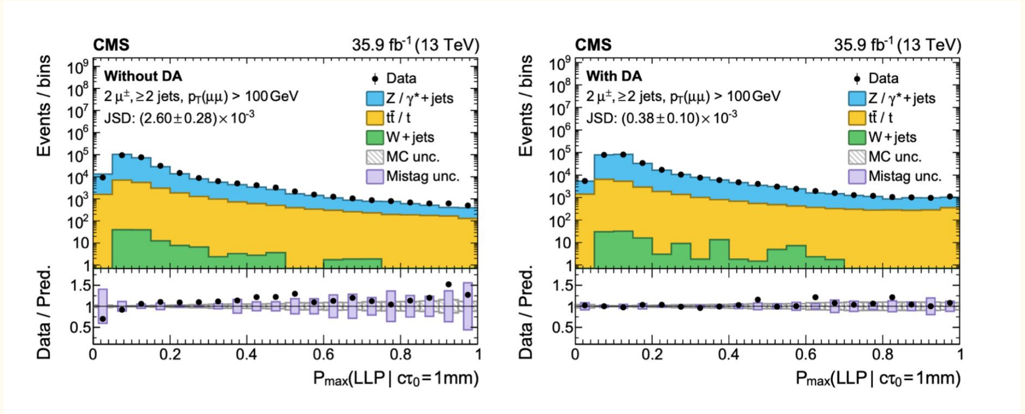
- In recent years, machine learning (**ML**) algorithms have shown tremendous promise in improving the sensitivity of results delivered by LHC experiments.
- While these algorithms are applied on data collected by LHC experiments, it is often preferable or even necessary, in the case of fully supervised algorithms, to train them on simulation
- However, a common problem that arises when applying ML algorithms on simulation is that the source domain is not fully representative of the target domain.
    - **Source domain:** the dataset used to train the ML algorithm.
    - **Target domain**: the dataset on which the ML algorithm is applied
- If the input features to the algorithm are not perfectly modeled by simulation, the output distributions may also be different between data and simulation. In a HEP analysis, these differences would typically be accounted for through a systematic uncertainty which covers the level of disagreement between data and simulation.

Domain Adaptation in HEP

- The *domain shift* between the *source domain* and *target domain* refers to the problem of the training sample not being entirely representative of the sample where the algorithm will be applied
- **Domain adaptation**, thus refers to methods which attempt to build algorithms robust to the differences between the source and target domains.
- Solutions to domain shift:
  - **Pre-training solutions**
  - **Solutions applied during training**
- In this project, we are applying a solution during training by explicitly rewarding the DNN for minimizing differences between distributions in the source and target domains

Solutions During Training: Gradient Reversal Layers

- Solutions applied during training are less common in HEP, but have been successfully applied
  - **Gradient Reversal Layers** have been included in training to minimize the DNN's ability to distinguish between events from data and events from simulation.
- Though applications of gradient reversal layers are not widespread, it has shown promise in the CMS search for LLPs
  - Gradient reversal layers are associated with only a small decrease in the algorithm's performance on the original task



Histogram Restricted Learning

- Though the gradient reversal layer has been found to improve agreement between the source and target domain, we attempted to implement a more direct solution to the issue of domain adaptation
- Instead of implementing a gradient reversal layer during the training of the DNN, we included a term to the loss function known as a *histogram loss component*
- **We constructed histograms of the output distribution for both data and simulation and minimized bin-by-bin differences by adding a histogram loss component**
- **The histogram loss component explicitly rewards agreement in the output distribution between data and simulation.**
- Instead of discouraging the DNN from learning features, as the gradient reversal layer does, our goal was to allow the DNN to distinguish between examples from the source and target domain
  - The Histogram Loss Function minimizes differences in the output distributions

Histogram Loss Component

- We applied our Histogram Loss Component for a classification task which is trained on events having a label 0 or 1, and which gives a prediction for each event [0,1]
- We estimated the probability distribution functions for events from the source domain, denoted S^S (label $z = 0$), and the target domain, denoted S^T (label $z = 1$) through histograms H^S and H^T with N number of bins.
- The bins are assumed to be uniformly spaced, such that the n-th bin of H^T can be constructed as the equation below, where $\delta_{i,n}$ is a weight defined such that we linearly interpolate for each entry when constructing the histograms

$$h_n^T = \frac{1}{|S^T|} \sum_{i:z_i=1} \delta_{i,n},$$

Histogram Loss Component Continued

- A loss function rewarding agreement between the probability distributions between data and simulation can be constructed as the sum of the squares of differences between each bin of H^S and H^T, as seen below

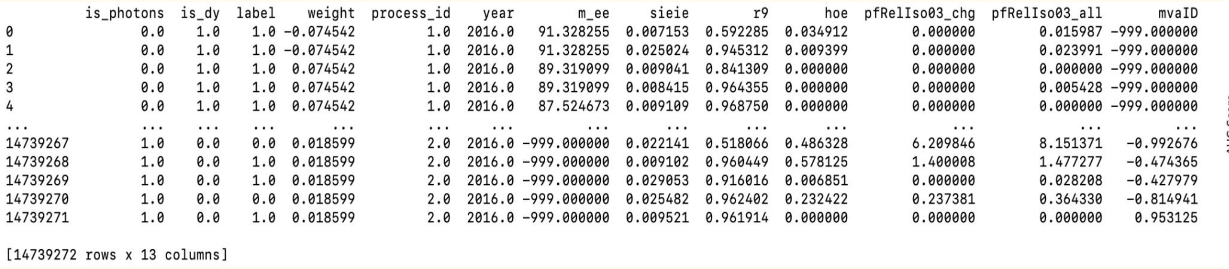
$$L_H(z, \hat{y}) = \sum_{n=1}^{N} (h_n^S - h_n^T)^2.$$

- A composite loss function which rewards performance on the original task as well as agreement between H^S and H^T takes the form below, where L_C is a cross entropy loss function and lambda is a hyperparameter that dictates the balance between rewarding classification and rewarding data and simulation histogram agreement

$$L(y, z, \hat{y}) = L_C(y, \hat{y}) + \lambda L_H(z, \hat{y}),$$

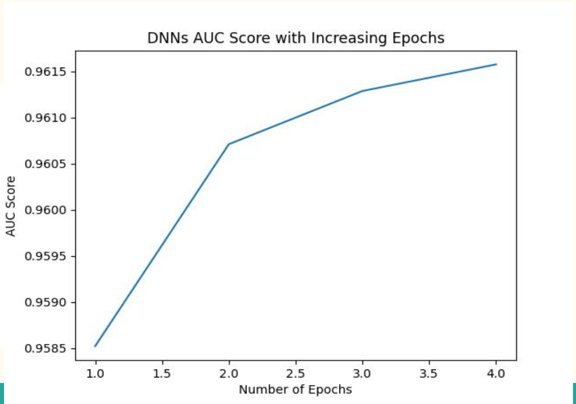
Prompt and Fake Photon Classification

- We constructed a photon ID DNN with a simple set of features describing isolation and shower shape. The classification component is trained with prompt and fake photons from gamma burst jets simulation, while the histogram loss component takes electrons from data and simulation in a control region.
- For the classification component of this project, I trained the DNN on a dataset of fake and prompt photons
- I calculated the DNN's AUC score as the number of epochs increases, and the results are as we should expect:



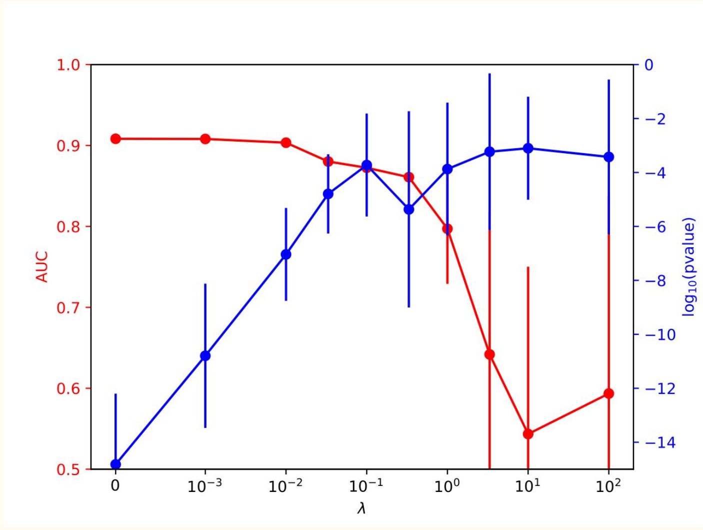
```
          is_photons  is_dy  label    weight  process_id    year        m_ee     sieie        r9       hoe  pfRelIso03_chg  pfRelIso03_all       mvaID
0                0.0    1.0    1.0 -0.074542         1.0  2016.0   91.328255  0.007153  0.592285  0.034912        0.000000        0.015987 -999.000000
1                0.0    1.0    1.0 -0.074542         1.0  2016.0   91.328255  0.025024  0.945312  0.009399        0.000000        0.023991 -999.000000
2                0.0    1.0    1.0  0.074542         1.0  2016.0   89.319099  0.009041  0.841309  0.000000        0.000000        0.000000 -999.000000
3                0.0    1.0    1.0  0.074542         1.0  2016.0   89.319099  0.008415  0.964355  0.000000        0.000000        0.005428 -999.000000
4                0.0    1.0    1.0  0.074542         1.0  2016.0   87.524673  0.009109  0.968750  0.000000        0.000000        0.000000 -999.000000
...              ...    ...    ...       ...         ...     ...         ...       ...       ...       ...             ...             ...         ...
14739267         1.0    0.0    0.0  0.018599         2.0  2016.0 -999.000000  0.022141  0.518066  0.486328        6.209846        8.151371   -0.992676
14739268         1.0    0.0    1.0  0.018599         2.0  2016.0 -999.000000  0.009102  0.960449  0.578125        1.400008        1.477277   -0.474365
14739269         1.0    0.0    1.0  0.018599         2.0  2016.0 -999.000000  0.029053  0.916016  0.006851        0.000000        0.028208   -0.427979
14739270         1.0    0.0    0.0  0.018599         2.0  2016.0 -999.000000  0.025482  0.962402  0.232422        0.237381        0.364330   -0.814941
14739271         1.0    0.0    1.0  0.018599         2.0  2016.0 -999.000000  0.009521  0.961914  0.000000        0.000000        0.000000    0.953125

[14739272 rows x 13 columns]
```



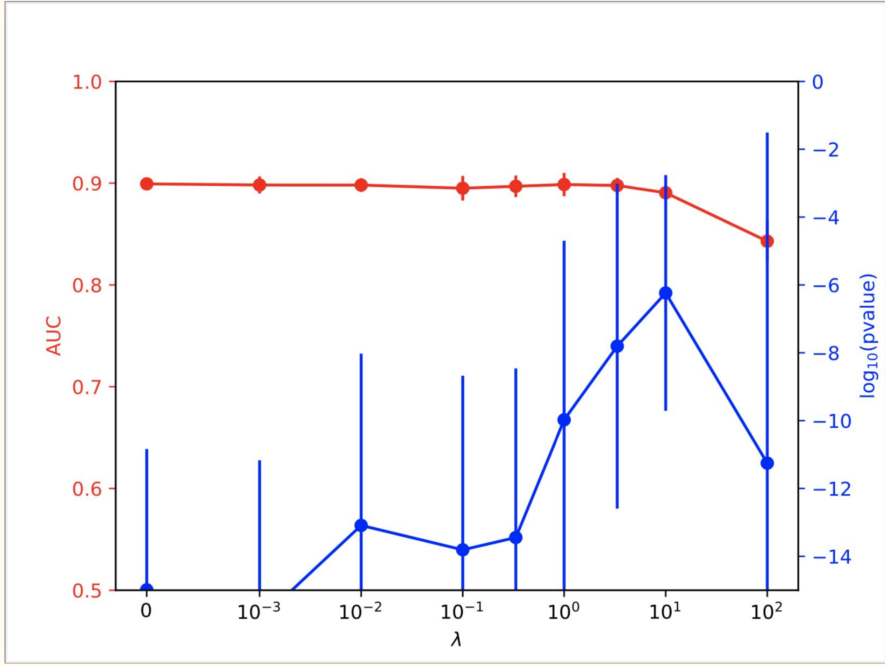
Initial Results – Histogram Loss Component

- Our main goal for this project was to quantify the performance of the DNN with a histogram loss component
- We sought to quantify its performance in terms of the original task, which was found through the AUC score, and through the data/MC agreement
- For our initial DNN, we found that as we increased the value of lambda, which represents the strength at which I implement the histogram loss component, the agreement between the distributions of the source and target domain greatly increased, but the AUC score decreased greatly as well



Final Results – Histogram Loss Component

- After updating the DNN, I recreated the previous plot and found a much more stable AUC score for increasing lambda values
- However, this new iteration of the DNN had less agreement between the source and target distributions for certain lambda values
- I wanted to explore the effects of changing the details of loss function, and the details of the DNN architecture and optimization on the DNN's performance
- This was accomplished by changing the DNN's lambda value, the learning learning rate, and the optimizer



Final Results Continued – Batch Sizes

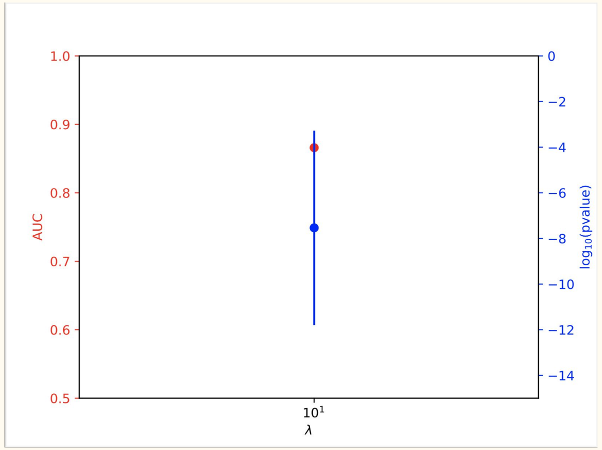


Figure 1: Batch Size: 10^6

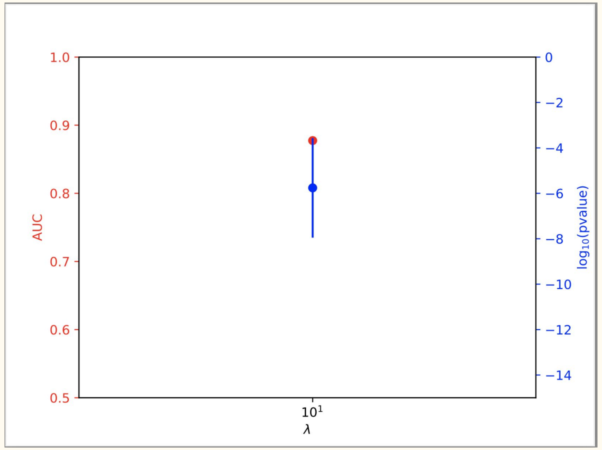


Figure 2: Batch Size: 10^5

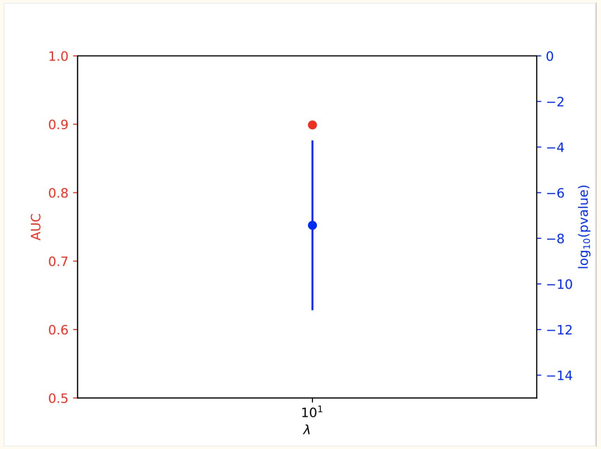


Figure 3: Batch Size: 10^4

Final Results Continued – Learning Rate

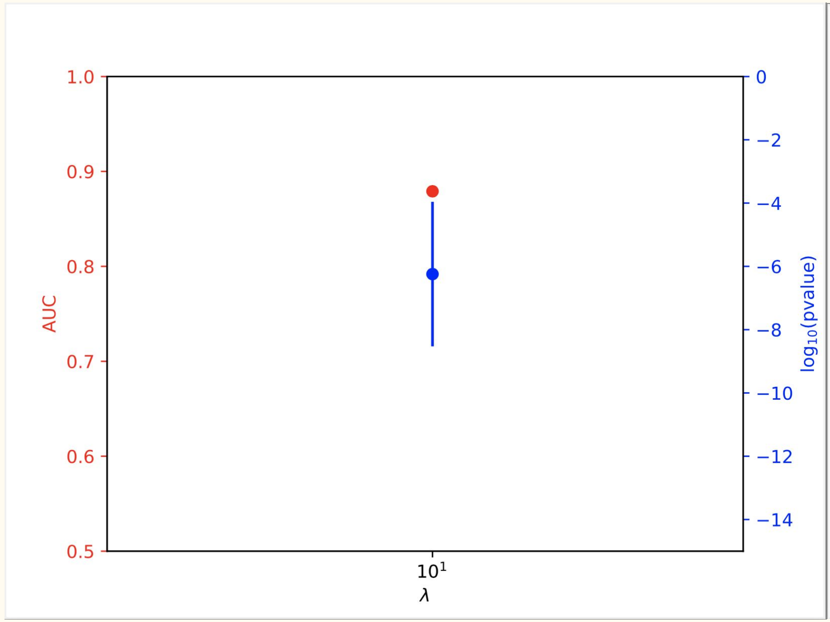


Figure 4:
Learning Rate: 3*10^-4

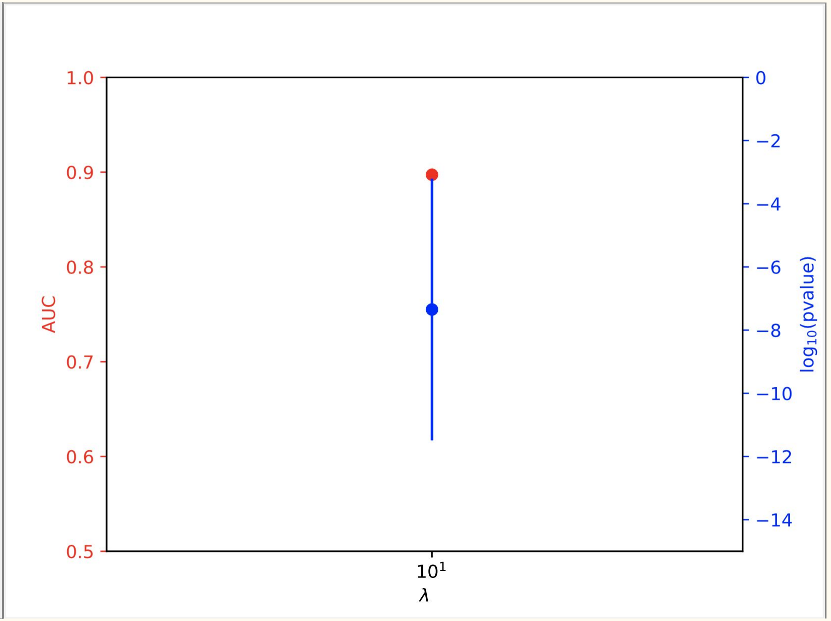


Figure 5:
Learning Rate: 3*10^-2

Final Results Continued – Optimizers

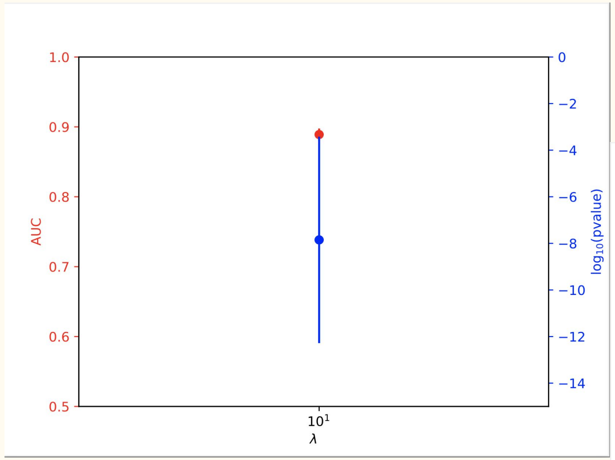
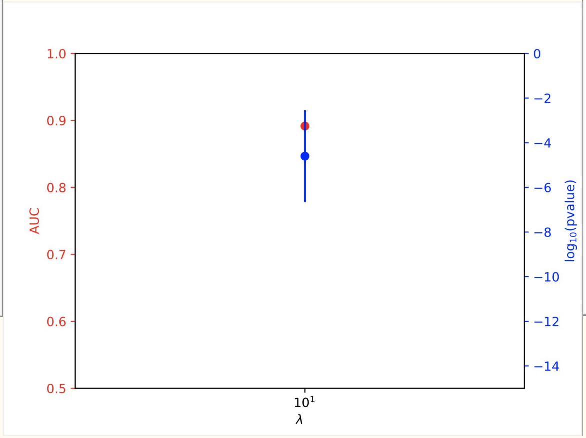


Figure 6: Adadelta Optimizer



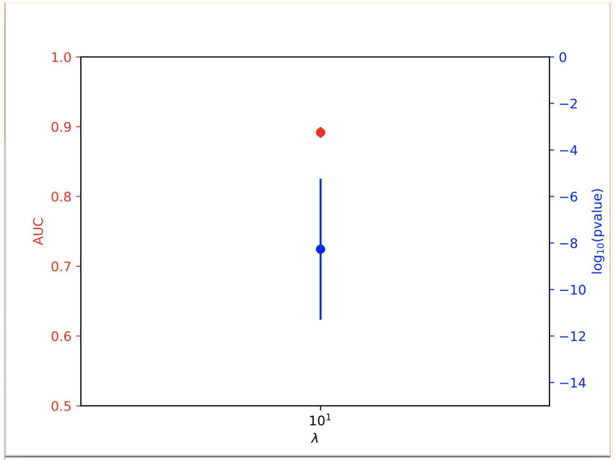


Figure 1: Adamax Optimizer

Figure 1: Adam Optimizer

Link to Github Repository

https://github.com/sam-may/HistogramRestrictedLearning