Neural Networks for calibrating ATLAS jets

1 Introduction

The Large Hadron Collider (LHC) collides bunches of 10^9 protons at energies of 14 TeV[1] every 25 ns. These parameters lead to an average of 40 collisions per bunch crossing, but most of them won't correspond to head-on collisions, but rather a glancing collision, labeled as a pile-up (PU) event. Physicists at the LHC are interested in the high-energy or hard-scatter collision in a brunch crossing. ATLAS is one of the multi-purpose detectors at the LHC, and looks at the highly collimated streams of particles, or jets, from the collision. To reconstruct a jet's transverse momentum, p_T, it must therefore correctly disentangle the true energy depositions and pile-up contributions that can cause extra energy to leak into a jet [1]. This challenge will intensify over the next few years when the LHC plans to increase the average PU from 40 to 140.

Our goal for this project was to use supervised ML techniques to improve the reconstruction of the jet's transverse momentum, p_T. The input to our algorithm was the p_Ts cluster deposits and locations in the detector, and we used linear regression, a single layer perceptron, and a convolutional neural net to output the p_T of the jet.

2 Related Work

The current ATLAS standard for dealing with PU is

$$p_T = p_T^{deposited} - \rho A_T - \alpha(N_{PV} - 1)$$

where A_T is the transverse area of the jet, N_{PV} is the number of primary vertices from the collision with tracks from charged particles, $p_T^{deposited}$ is sum p_T from the jet definition, and ρ and α are factors set in calibration [2].

PU causes so many low energy events in the detector that it contributes to the jet-energy of the hard-scatter event with lower energy depositions. Since there are much more low energy depositions from PU in the jet than true hits from the hard scatter, the PU contribution is approximated by a low energy smear of energy in the detector [3]. Also The N_{PV} term indicates that the jet's energy will need to be corrected more aggressively for events with more PU. The 1 is subtracted in the $\alpha(N_{PV}-1)$ term since if there are N_{PV} primary vertices, one of them will be from the hard-scatter, while the rest are from PU.

PU is just from the physics processes in the collisions, but since we aren't just interested in creating the collisions, but in reconstructing them, our detector will introduce additional uncertainties. The problem is further complicated by non-uniformities in the detector. ATLAS currently accounts for the non-uniformities of the detector using numerical inversion [4].

Some other work has been done to apply supervised ML techniques to improve the jet energy resolution in the presence of pile-up. One of the CS 229 groups from last year tested linear regression, SVMs, and Boosted Decision Trees to correct for pile-up in jets [5]. They found that linear regression performed the best, motivating our choice of linear regression to optimize our choice of loss function. Expanding on this, Francesco Rubbo looked at linear regression using the p_T summed in annuli around the jet axis as features to learn the truth p_T^{jet}[6]. Both of these studies yielded promising results better than the current ATLAS standard, but each of these studies considered only truth level jets, i.e, did not account for the non-uniformities of the detector. Therefore, we sought to expand on this work by using simulations that also included detector level effects. We also used a different algorithm, a neural network to seek solutions while placing minimal requirements of the functional form of the solution.

3 Dataset and Features

Our dataset consists of $\sim$ 6 million detector level jets with p_T between 5 and 160 GeV which contain:

- The true jet p_T : the labels for our supervised learning algorithm
- The (η, ϕ) coordinates[2] with the corresponding p_T for the individual clusters making up the jet.
- A_T: jet's transverse area:
- j0: reconstructed jet p_T *with* area subtraction
- j no area sub: reconstructed jet p_T *without* area subtraction
- Number of primary vertices in event, N_{PV}
- The event weights.
- The p_T summed in 8 different disks about the jet axis with disk a radii of [0.05, 0.1, . . . , 0.4].

We divided these jets into 3 different samples, with 80% in our training sample, 10% in our cross validation (CV) sample, and 10% in a test sample. Since we had vastly more low p_T events, we revised the event weights by flattening the truth p_T distribution in 4 GeV bins to prevent our model from becoming biased toward low energy events.

[1] An eV is a measure of energy, the energy gained by a particle with the same charge as an electron accelerated through a potential difference of 1 V. In particle physics, we use natural units where $c = 1$, so that mass, momentum, and energy are given in terms of eV.

[2] (η, ϕ) represent the angular coordinates on the surface of the surface of a cylinder, where ϕ is the azimuthal angle in $[0, 2pi]$, and $\eta = -\log\tan\frac{\theta}{2}$, where θ is the polar angle in $[0, \pi]$.

We trained over events for all the N_{PV} values, but for the the CV set we only considered events $5 \leq N_{PV} \leq 25$, since this was where numerical inversion was defined, and we wanted to compare our models to the ATLAS standard. This N_{PV} restriction still kept 95.6% of the CV set. Furthermore, since it was known that the jet energy scale can depend on the η position of the jet [7], we also required that jets fed into the neural nets have $|\eta| < 1.0$, which requires the jets to be in the central region of the detector where the detector non-uniformities are not as significant.

An example of the energy deposition of one of our jets is shown in Fig. 1. The jet's radius is $R = \sqrt{\eta^2 + \phi^2}$, and $R \approx 0.4$ for the jet regime we're considering. For our jet images we used a resolution of $\Delta\eta \times \Delta\phi = (0.1, 0.1)$, so the image illustrates the p_T deposited in the bins of the 8×8 matrix defining the jet images and centered at the jet axis.

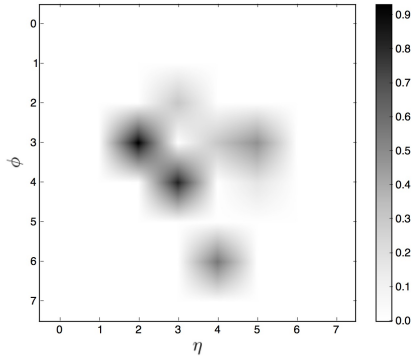

Figure 1: Sample image of a jet event in our detector with the p_T deposited in the η, ϕ bins of an 8×8 matrix.

4 Methods

4.1 Linear Regression with SGD

We wanted to investigate which version of a cost-function was best suited to our problem. To do this we looked at the CV error in a linear regression model with input $j0$, the area subtracted truth jet to predict the truth jet p_T. We considered variants of the cost function

$$J_\beta(\theta) = \frac{1}{2} \sum_{i=1}^{m} w_i \frac{(p_{T,i}^{truth} - h_\theta(j0_i))^2}{(p_{T,i}^{truth})^\beta}$$

where w_i are the event weights for each of the jets. When $\beta = 0$ this reduces to the cost function for linear regression presented in the lecture notes, and $\beta = 2$ corresponds to the cost function defined as the sum of squares of the relative errors. We then used stochastic gradient descent to minimize $J(\theta)$ over our training set.

4.2 Neural Network

We don't want to constrain ourselves to linear relationships between inputs and outputs and inputs. The universal approximation theorem says that a single layer perceptron can approximate any function to arbitrary precision [9], so this motivated us to use a neural network for creating outputs from inputs. The forward pass of a neural network is how you find the networks predictions, and then the parameters are updated based on the size of the objective function.

4.2.1 Forward Pass

To explain the algorithm, consider a NN with I inputs, a hidden layer with H nodes, and a single output for the estimated p_T, as depicted in Fig. 2. Each of the hidden layer

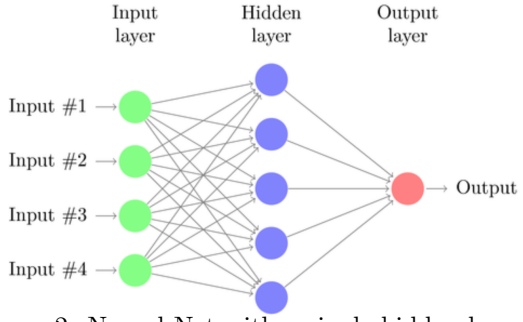

Figure 2: Neural Net with a single hidden layer [8]

nodes accepts as inputs a weighted sum of the I inputs,

$$z_j = \sum_{i=0}^{I} w_{ij} x_i$$

where z_j is the input to node j in the hidden layer, x_i are the input values, and w_i are the weights. Note, we're following the convention from class where $x_0 = 1$ so that the parameters w_{0j} correspond to the bias terms [10]. The perceptron neural network uses the a sigmoid neuron with activation function $g(z) = 1/(1 + e^{-z})$. Finally, the final output is then a weighted sum of the hidden layer outputs

$$h(x) = \sum_{j=1}^{N} v_j g(z_j)$$

where v_j are the weights. The parameters we need to determine during training are the weights w_{ij} and v_j.

4.2.2 Optimization

We initially tried stochastic gradient descent for optimizing the cost function, but failed to get very accurate convergence. This is probably because sigmoid nodes have slopes $g'(z) = g(z)[1-g(z)]$, so if your initial guess is poor, $g(z) \approx 0$ or 1, making $g'(z)$ tiny, so that it takes a large number of iterations for convergence. Instead of SGD, we used the Adam optimizer, a 1st order gradient optimization designed for large datasets with noisy data [12]. It uses estimates of the 1^{st} and 2^{nd} moments of the gradient, (i.e, $E[\nabla_\theta J]$ and $E[(\nabla_\theta J)^2]$, respectively) to optimize the objective $J(\theta)$.

4.2.3 CNNs

We also wanted to implement a convolutional neural network (CNN) using the image of the p_T clusters inside our jet. A basic CNN has three types of layers: convolutional, pooling, and fully connected layers.

The input to a CNN is an image. For a grey scale image, the image is just a 2D matrix, but a color image is represented as a 3D matrix with entries corresponding to the weights of the red, green, and blue pixels. The convolutional layers then form a series of filters that act over the depth to create an output volume also in three dimensions. The height and depth of each convolutional layer is the *receptive field*, or the size of the window over which the dot product between weights and pixels is applied on. The *stride* is then the increment by which we translate the window successively in width an height to iterate through the input image, typically chosen to be 1. Although in general the weights in a depth slice in the convolutional layer can be different for each window in the spatial dimension, this can lead to a huge number of parameters. We can tie the weights of each a depth slice in the convolution layer to be the same for each window. This helps preserve the translational invariance of the filters, and makes the dot product of the pixel windows and filter weights an actual convolution. Although the filter window is tiny in height and width, it always acts over the full depth, i.e, over all the pixel colors.

The pooling layer then is a spatially reducing to force the CNN to try to select its most relevant features. In each layer of the depth slice, it looks in a small window (typically 2×2) and outputs the maximum element it sees in the window. Typically the pooling layer uses a stride of 2, so decreases the input volume by a factor of 4, although the dimension of the depth stays the same.

The previous two layers only considered local connections of the image inputs to calculate outputs. But the last layer of the NN is a fully connected layer which takes as input the entire image (over all depths) and sends the outputs to a linear layer.

For our p_T calculation, we again used one output node with a linear neuron which implemented a weighted sum the outputs of the fully connected layer to predict the p_T.

5 Results

First we considered linear regression with the cost function $J_\beta(\theta)$ for $\beta = 2, 1, 0,$ and -1, and minimized it using stochastic gradient descent with learning rate $\alpha = 0.1$. To evaluate the effectiveness of this method, we used the closure $R = h(x)/y$, where $x = p_T^{reco}$ and $y = p_T^{truth}$, where $R = 1$ corresponds to a perfect fit between the hypothesis and the labels. For ATLAS physics analyses, it is important that the p_T reconstruction be accurate over the p_T spectrum. So to see how the closure depended on the truth p_T, we binned the closure in p_T^{truth} bins of 2 GeV, and plotted the average shown in Fig. 3.

For lower p_T, the $\beta = 2$ cost function performs optimally because the cost function with relative errors penalizes smaller p_T more heavily when $\beta > 0$. For higher p_T, the $\beta = 0$ cost function performs the best. Therefore, we sought to combine the strengths of these two cost functions to define our ultimate cost-function of interest,

$$J(\theta) = \sum_i w_i (y - h(x))^2 \left(1 + \frac{\lambda}{y^2}\right),$$

where the λ appropriately scales the dimensionality of J_2 to J_0. [hbt] To optimize the λ, we looked at a NN model

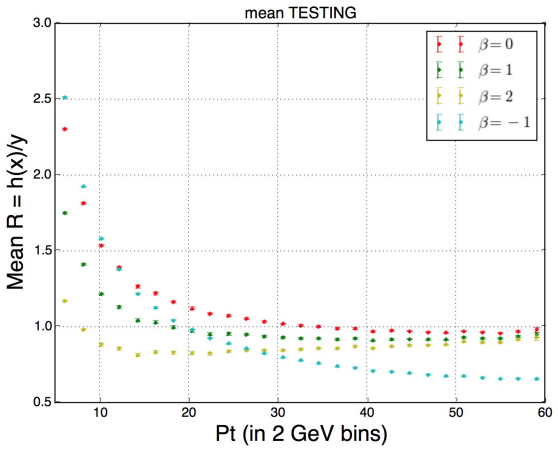

Figure 3: The average closure $\frac{h(x)}{y}$ in p_T^{truth} bins of 2 GeV.

with $j0$ as an input, and plotted the deviation of the median R from 1 for different λ values in Fig. 4. The median discrepancy from R was minimized for $\lambda = 100$ to appropriately scale the relative error to the absolute error for $5 < p_T < 160$ GeV.

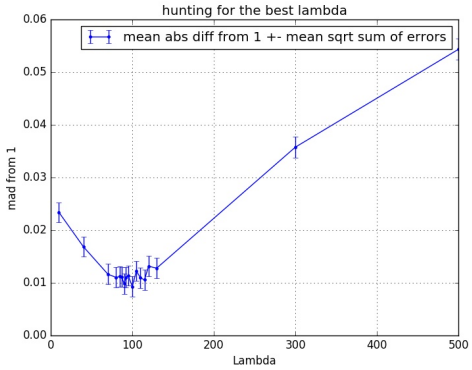

Figure 4: The λ optimization for $J(\theta) = \sum_i w_i (y - h(x))^2 \left(1 + \frac{\lambda}{y^2}\right)$ using the deviation of the median of R from 1.

We used Keras to implement the neural net [11], with the initial layer as our inputs, a hidden layer with 5 nodes, and the output layer with just one node for $h_\theta(x)$. We experimented with increasing the number of nodes in the hidden layer or increasing the number of hidden layers, but did not see a better fit with the training data. We ran over 100 timesteps, and the learning curve for the NN models we tried is shown in Fig. 5. We can see that as we continue to train on the data over more epochs, our model loss goes down and then plateaus, indicating that 100 epochs are sufficient for convergence.

We then trained the NN with this cost function over several different combinations of the inputs. First we ran it with the $j0$ and the the j *no area sub* as inputs, so essentially the same thing as linear regression, but with out requiring the hypothesis to be linear. Then we fed in

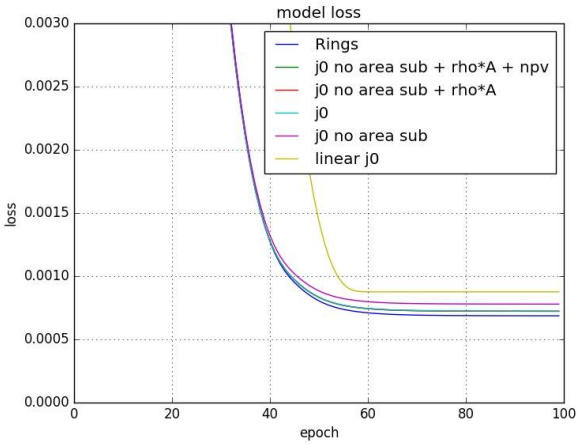

Figure 5: The learning curve for the models' losses versus the number of epochs.

j no area sub and ρA_T as inputs to the NN, and next we fed in N_{PV} to the NN looked at inputs *j no area sub*, ρA_T, and N_{PV}. Lastly, we used the 8 rings data + j0 as inputs to the NN. For all of these inputs, a bias input is implied, as described in the Section 4.1.2. Once we trained the NN, the closure in p_T^{truth} bins was again calculated, but this time with the bin width of 4 GeV. In Fig. 6, we see that our NN is closer to 1 away from the edges of the truth p_T distribution, and does better than current standard of numerical inversion in the current region of interest. The NN does poorly out at the edges because the network is learning from our dataset never to expect events below 5 GeV or larger than 160 GeV. However, in ATLAS physics search analyses, background mostly corresponds to low p_T jets, so in signal selection a cut of 20 or 25 GeV is generally imposed on the jet of interest, so we aren't concerned with the closure divergence at the lower p_T edge. We can account for the jet energy resolution at the high p_T end by increasing the max p_T in our dataset, because we saw when we increased our data sample from 5-60 GeV to to 5-160 GeV increased the range where our NN performed closer to 1.

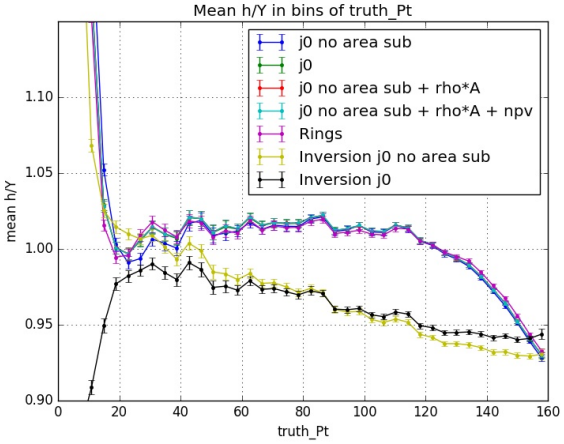

Figure 6: The mean closure in 4 GeV p_T^{truth} bins.

The standard deviation of the points inside each of the p_T^{truth} bins is shown in Fig. 7, and we can see that our NN errors perform comparably to or better than the current ATLAS standard.

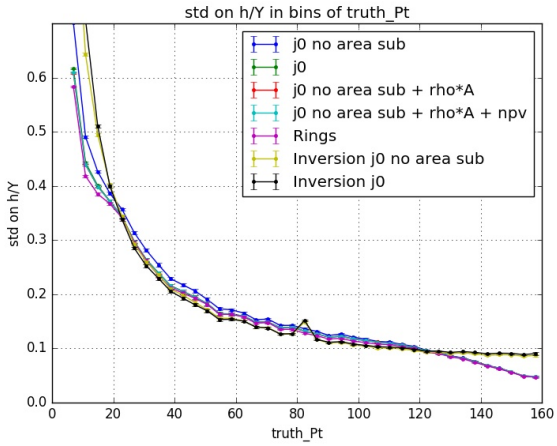

Figure 7: Standard deviation of the closure in 4 GeV p_T^{truth} bins.

To define a single metric that quantifies the performance of our algorithm, we used the CV error in a p_T range from 45-120 GeV, shown in Table 1. The inversion was calculated both on $j0$ and *j no area sub* because how detector modified the jet's p_T is dependent on the reconstructed p_T, so we considered adding in detector effects with and without the area subtraction incorporated.

Models	**CV error**
Inversion on $j0$	0.0327 ± 0.0006
Inversion on *j no area sub*	0.0311 ± 0.0006
NN with $j0$ input	0.0151 ± 0.0007
NN with *j no area sub*	0.0138 ± 0.0007
NN: *j no area sub*, ρA_T	0.0153 ± 0.0007
NN: *j no area sub*, ρA_T, N_{PV}	0.0141 ± 0.0007
NN with 8 annuli and $j0$	0.0129 ± 0.0007

Table 1: CV error for $p_T^{truth} \in [45-120]$ GeV. The NN with the annuli information does the best.

We can see that every NN that we derived performs better than the current ATLAS standard, numerical inversion. The NN with annuli info does best, showing that jet substructure helps. All of the other NNs perform comparably, i.e, their CV errors are within their standard deviations of each other. The contrast between the NN with *j no area sub* with ρA_T versus just inputing $j0 = j$ *no area sub* $- \rho A_T$ indicates that the NN with both of the inputs is learning the linear dependance on the area subtraction. Finally, incorporating the N_{PV} variable didn't increase the performance, indicating that no new information was learned by this variable.

Finally, Fig. 8 illustrates how well the derived hypothesis reconstructs the data for the best case scenerio of the NN with the 10 rings. As we can see, the hypothesis (in red)

incorporates more variation than just a simple linear fit. It never predicts values below ∼7 GeV, which makes sense because our NN prediction in Fig. 6 performs worse at the edges of the NN.

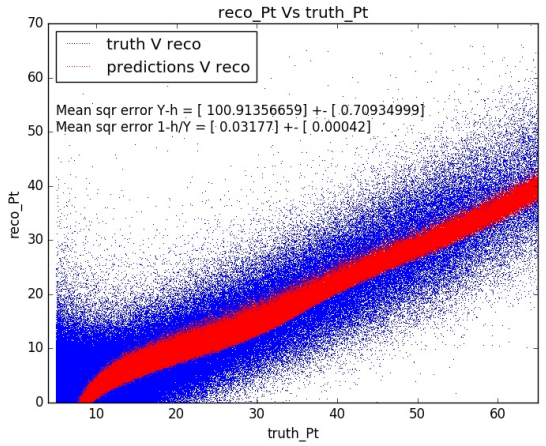

Figure 8: The blue dots show the reconstructed p_T values as a function of truth p_T, while the red points show our corrected p_T hypothesis fit.

Finally, we ran over a 50,000 event subsample of the data to train a CNN with architecture $(2 \times 8 \times 8) + (5 \times 5 \times 5) + 2 \times 2 + 5 + 1$.. For our $(2 \times 8 \times 8)$ dimensional inputs to the CNN, we used the pixel image in two "colors:" the first was the jet shown in Fig. 1, but then, to incorporate the jet's area information, we also passed an area-subtracted version of the image, where if there were N fired pixels in the image, each of these pixels had $\frac{\rho A_T}{N}$ subtracted from it. Then the $(5 \times 5 \times 5)$ factor was the convolutional layer, the (2×2) factor was the pooling layer, and there were 5 nodes in the fully-connected layer. The preliminary results show a more noisy mean closure and higher standard deviation than the NN results. The CV err in a p_T range [20-120] GeV was , a factor of 2 worse than 0.076 ± 0.005, about a factor of 2 worse than numerical inversion.

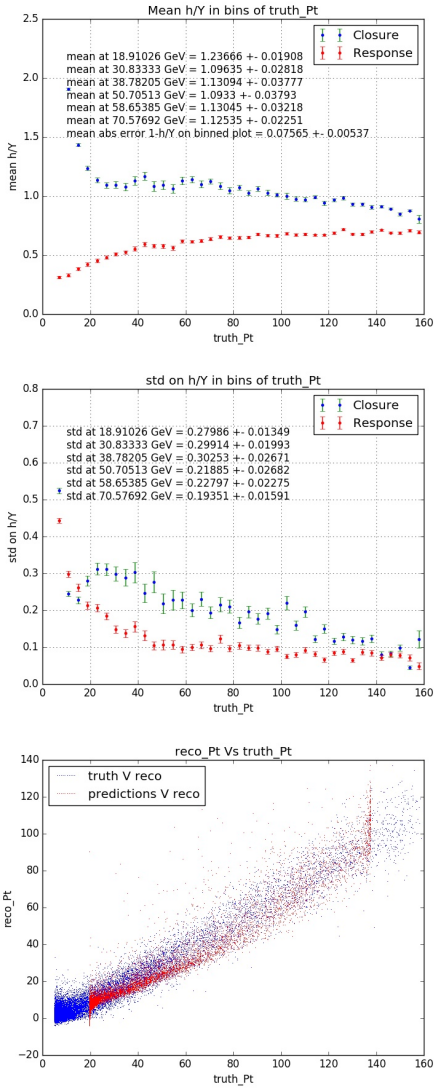

Figure 9: Preliminary results from the CNN. In the legends, the closure is the $h(x)/y$, where $h(x)$ is the output of the CNN and y is the truth p_T, and the response is $h(x)/y$, where $h(x)$ is the response of the detector from numerical inversion.

6 Future Work

In continuing this work, we hope improve and understand CNN architecture since we expect that the additional information about the individuals clusters should help us better reconstruct the p_T, just as the information from the rings data gave us the best neural net through utilizing jet substructure information. To do this, we plan to modify our architecture to $(2 \times 8 \times 8) + [(10 \times 5 \times 5) + (10 \times 5 \times 5)] + MaxPool(2,2) + (40 \times 5 \times 5) + MaxPool(2,2) + 5 + 1$, and then train over more events, using dropout to prevent overfitting.

Another way that we could solve this problem would be instead of picking an actual value for the p_T truth, instead just predict which p_T^{truth} bin an event lies in. To do this, we would have N output nodes for our NN, and where the outputs correspond to the probabilities of soft-max for which p_T bin a jet would lie in. We then classify the jet using the bin that gets assigned the greatest probability. This has an advantage b/c we were already using a the closure in p_T bins as our metric for evaluating the model, meaning that this problem could be formulated as a classification problem quite naturally, and we have plenty of data to train all of these classes.

In conclusion, we used linear regression to ID the correct cost function, and then derived several NNs that reconstruct the jet energy better than the current ATLAS standard. The NN that incorporates jet-substructure information does the best.

We would like to thank Aviv Cukierman from the SLAC ATLAS group for his advice throughout the project, and the CS 229 TA team for all their assistance through the course.

References

[1] The ATLAS Collaboration. "Performance of pile-up mitigation techniques for jets in pp collisions at $\sqrt{s} = 8$ TeV using the ATLAS detector." arXiv 1510.03823v1.pdf (2016).

[2] Schwatzman, Ariel. "Jets and Missing ET at the LHC." Talk given at the SLAC Summer Institute on Aug 21, 2016.

[3] Matteo Cacciari, Gavin P. Salam. "Pileup subtraction using 514 jet areas." Phys. Lett. B659, pp. 119126, (2008).

[4] Cukierman, Aviv and Benjamin Nachman. "Mathematical Properties of Numerical Inversion for Jet Calibrations." arXiv:16.09.05195v1 [physics.data-an] 16 Sep 2016.

[5] Kong, Vein S, Jiankun Li, and Yujia Zhang. "Pileup Subtraction and Jet Energy Prediction Using Machine Learning." Final project report for CS 229 in Fall 2016.

[6] Francesco Rubbo, "Jet pileup and ML." Talk given in an ATLAS Collaboration Meeting, slides at https://dl.dropboxusercontent.com/u/4890393/ jet-pileup_ML20160629.pdf (2016).

[7] The ATLAS Collaboration. "A measurement of the calorimeter response to single hadrons and determination of the jet energy scale uncertainty using LHC Run-1 pp-collision data with the ATLAS detector." arXiv 1607.08842v1 (2016).

[8] WILDML: AI, Deep Learning, NLP. "Implementing a Neural Network from scratch in python - an introduction." *http://www.wildml.com/2015/09/implementing-a-neural-network-from-scratch/* Accessed Dec 16, 2016.

[9] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks (Studies in Computational Intelligence).* Springer. (2012).

[10] Geoffrey Hinton. "Machine Learning and Neural Nets." Coursera course (2003).

[11] Keras libraries, Deep Learning for Python. https://keras.io (2016).

[12] Diederik Kingma, Jimmy Ba. "Adam: A Method for Stochastic Optimization." arXiv 1412.6980 (2014).

[13] CS 231 Lecture notes. "Convolutional Neural Networks for Visual Recognition." *http://cs231n.github.io/convolutional-networks/.* Accessed Dec 15, 2016.