Generative Models for High Energy Physics Calorimeters

Charles Guthrie (cdg356@nyu.edu)
Israel Malkin (im965@nyu.edu)
Alex Pine (akp258@nyu.edu)
Advisor: Prof. Kyle Cranmer (kyle.cranmer@nyu.edu)

Abstract

Existing methods for simulating particle collision experiments for the planned International Linear Collider[1] are time-consuming, slow, and expensive. In these experiments, two high-energy beams of particles are collided with one another, and the resulting "particle shower" is recorded by calorimeter sensors that surround the beams. We explore using a Wasserstein Generative Adversarial Network model, and an recurrent neural network model to simulate the sensor data that is recorded from these experiments. Taking particle type (photon or pion) and beam energy as inputs, these model generate an image that represents a flattened three-dimensional array of LCD electromagnetic calorimeter sensors.

Introduction

The International Linear Collider (ILC), currently under development, will be a particle accelerator designed to collide two high-energy beams of subatomic particles and measure the resulting particle showers. The particle showers that are produced by these collision experiments will be detected and measured with a variety of sensors. Sensors in the ILC designed to measure energy levels of particles are called "calorimeters." Bendavid et al. (2016) have created a dataset containing nearly one million simulated calorimeter readings.

Existing methods to simulate calorimeter data rely on statistical models derived from physical theory. As a consequence, building a model for a novel experiment requires simulating the underlying physics, which is often computationally expensive to do. Neural network models, in contrast, require no scientific expertise to construct, and are computationally inexpensive to run once they have been trained. If a neural network model could be constructed to generate realistic calorimeter data, it would significantly speed up existing computational pipelines for high-energy physics experiments.

We experiment with two neural network models to simulate calorimeter readings: a recurrent neural network based on LSTMs (Hochreiter and Schmidhuber, 2017), and a generative

[1] https://www.linearcollider.org/ILC

adversarial network (GAN) (Goodfellow, 2014) using the “Wasserstein” training technique (Arjovsky, et al., 2017). In both models, the three-dimensional calorimeter data is projected to two-dimensions, allowing us to use model architectures designed for two dimensional data. We find that both models achieve some success, but have large shortcomings. Coincidentally, Paganini et al. (2017) recently published a paper documenting a very similar model.

Data

The ILC collision event data set was provided to us as part of the CERN Open Data Initiative (Bendavid, 2016), courtesy of Maurizio Pierini and Jean-Roch Vilmant. It contains calorimeter readings for nearly one million single-particle collision events, divided nearly evenly between two particle types: photons and neutral pions. Each collision event has its particle shower recorded by two different kinds of calorimeter sensors: an LCD electromagnetic calorimeter (ECAL) and a hadron calorimeter (HCAL). Both types of calorimeter record energy values in a three-dimensional grid. The particle beam travels through the calorimeters, along their z-axis. The ECAL consists of a 25x25x25 array of sensors, while the HCAL calorimeter is a 4x4x60 array. Each sensor records a positive number representing the energy detected at that point in space.

The data is split across 100 different files in Hierarchical Data Format (HDF5), each containing about 10,000 different collision events, split roughly equally between each particle type. Each event has a three parts: the event’s parameters, its ECAL readings, and its HCAL readings. The event parameters are given as a list of numbers containing the type of particle beam used, the initial energy of the beam, and its initial momentum vector. The ECAL sensor readings are given as a 25x25x25 tensor of floating point numbers. The figure below represents a slice, averaged across 100 events, in a sample ECAL sensor. The bottom 5 layers and the rightmost 5 layers of every sensor was exactly zero, so we truncated the ECAL data to 20x20x25 pixels.

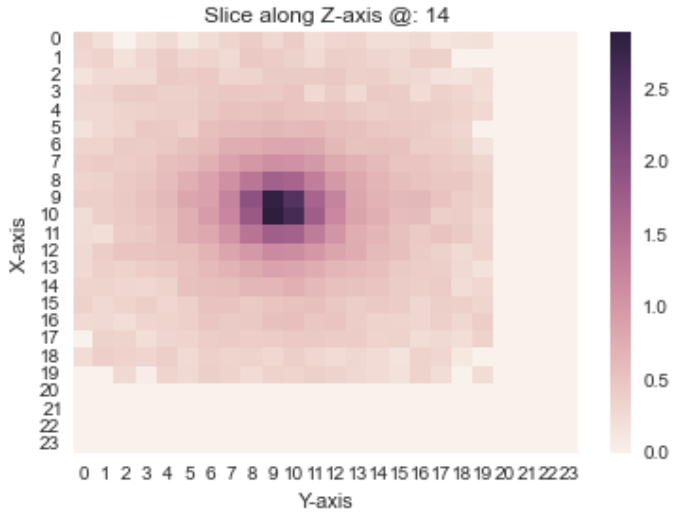

Figure 1. A 2D slice of ECAL calorimeter reading. Note that the data points from entries 20 through 24 are empty.

The primary challenges of the data were that it was highly skewed, highly sparse, and of high dimension. Generating a 20x20x25 calorimeter cube means generating an image of 10,000 pixels. And yet over 85% of pixels are exactly zero, and the majority of the remainder are close to zero. Meanwhile, the largest 15 values in any 2D slice (out of 400) account for 42% of the total energy in a typical event, and there is a small number of points that have very high energy, up to 500 GeV. See figure 2 for a histogram of pixel intensities across samples.

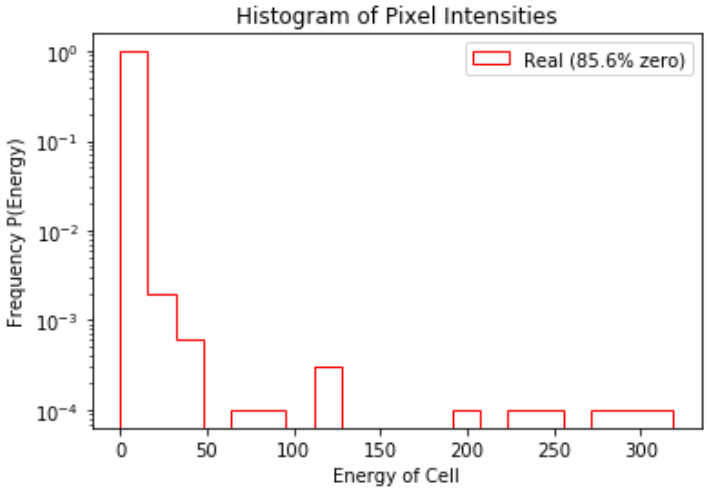

Figure 2: Histogram of the total energy in the calorimeter readings.

Since particle showers enter at the center of a calorimeter, most of the high-energy pixels of interest are in the center of the cube; you can see that in the individual axis histograms, pictured in figure 3, across the X, Y, and Z axis, respectively:

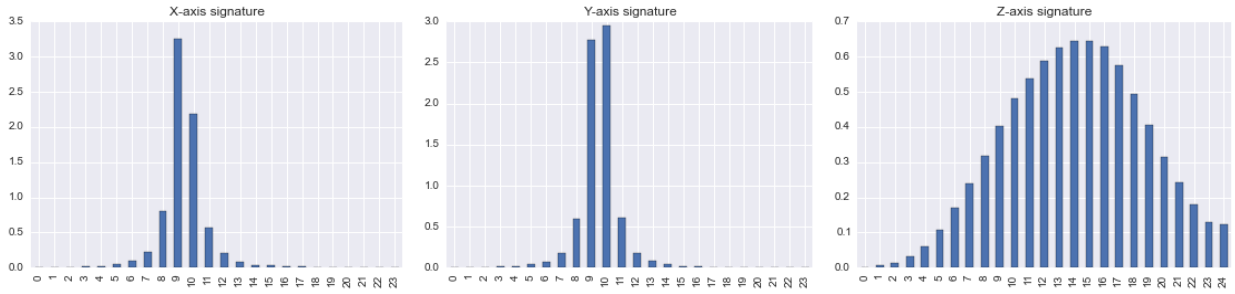

Figure 3: Histogram of the total energy in the calorimeter readings, averaged over the X, Y, and Z axes.

Unrolling to Two Dimensions

Most image-generation research is in two dimensions, rather than three. Rather than adapting 2D models to 3D, we found it easier to flatten our 3D data to 2D. But we wanted to flatten the data in a way that would preserve the proximity of the central high-energy particles to each

other. So for each calorimeter we cut a spiral into the 3D cube and unrolled it. After the transformation, the high-energy pixels from the center of the cube appear on the left side of the resulting 2D image. Then in some experiments, we discarded the low-energy area on the right side to reduce dimensionality (see figure 4).

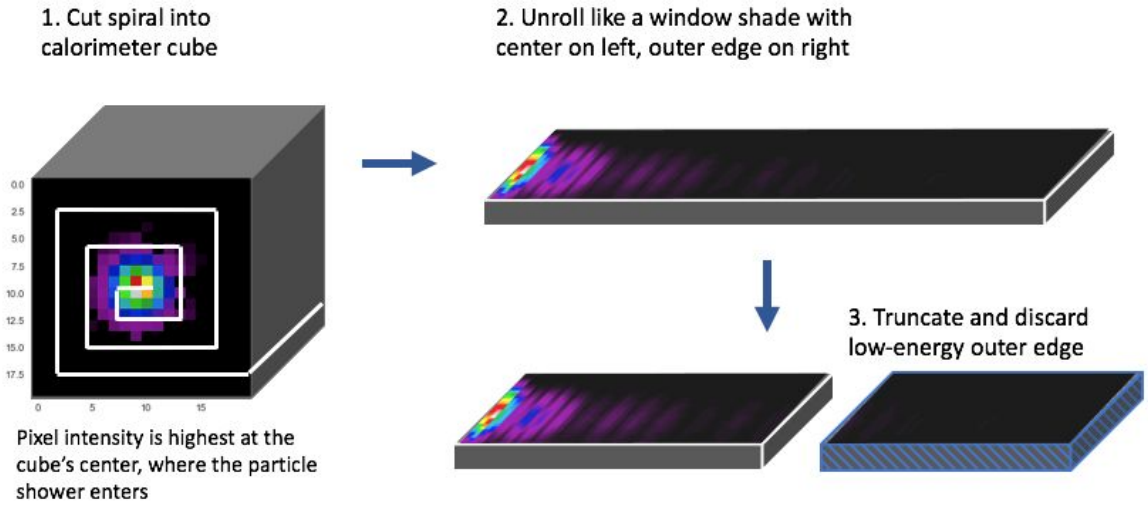

Figure 4: Demonstration of how the three-dimensional calorimeter data is projected into two dimensions.

Evaluation Methodology

As generative modeling is an unsupervised learning task, we needed some objective measures of model performance and progress. For the most part this took the form of data visualizations so that we could compare generated output with real data. For all of the charts below, the goal is for generated data to match the real data. There are many different ways to look at the real and generated images, and no single perspective tells the whole story.

Naturally the first step would be to visualize energy intensities of each calorimeter. However it was its own challenge to visualize four-dimensional data (x, y, z position and pixel intensity) in two dimensions. In order to view a whole cube at a time, we used a plot of the unrolled data. Figure 5 illustrates an example of this.

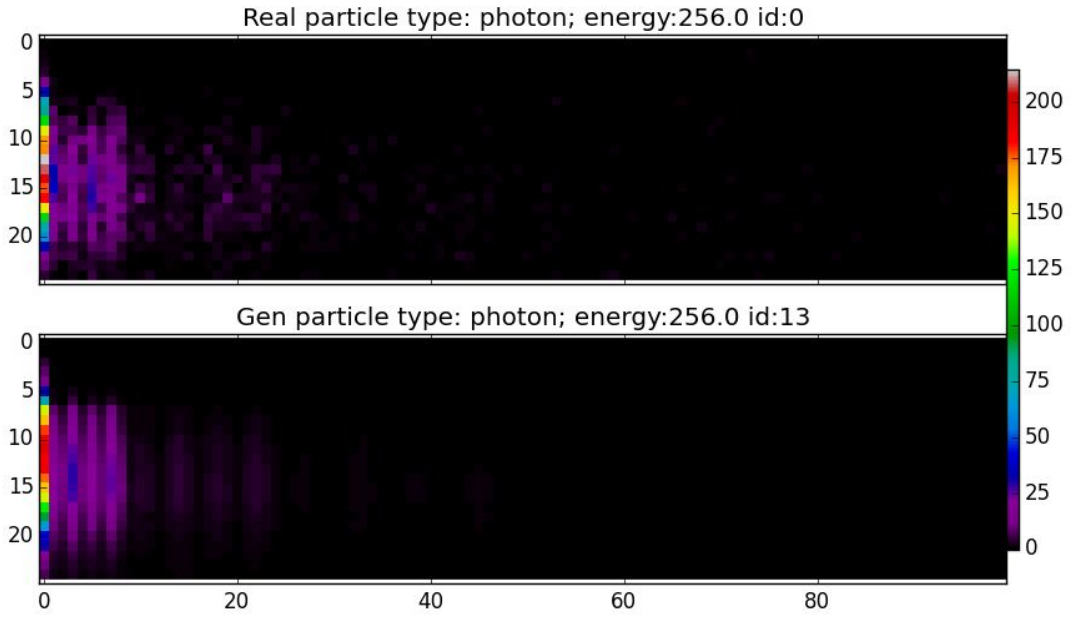

Figure 5: Sample of visualization of real and model-generated calorimeter readings.

These individual charts were useful for visual inspection, but direct comparison between the shapes of the generated distribution and real data distribution required a different approach. One metric we used was to inspect the sum of all energy in each layer of the z-axis of the data, averaged across several thousand samples. If our model is working well, the curve for the generated data should look similar to that of the real data. An example of this chart is shown in figure 6. The left side of the graph is the front of the calorimeter - where the particle shower enters - and the right side of the graph is the back.

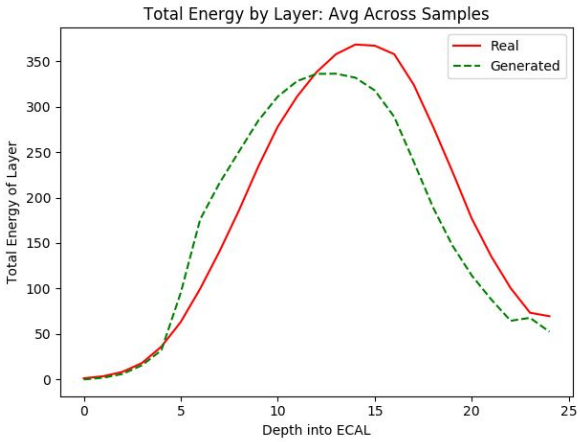

Figure 6: Sample chart comparing total energy, by layer, of real and generated images

But that chart does not say whether all of the energy of a given layer is concentrated at a single pixel, or spread out across many pixels in a given layer. The next chart below displays the

energy spread at each layer. Energy spread here is root mean squared error (sigma) of pixels, weighted by pixel intensity:

$$\sigma_z = \sqrt{\left(\sum_{i=1}^{n} w_{i,z}(x_{i,z} - \mu_z)^2\right)},$$

Where σ_z is the root mean squared error at layer z; μ_z is the position of the weighted center of mass of the pixel intensities at layer z; $x_{i,z}$ is the (x,y) position of pixel i; and $w_{i,z}$ is the intensity of pixel i, at layer z. An example plot of this metric is shown in figure 7.

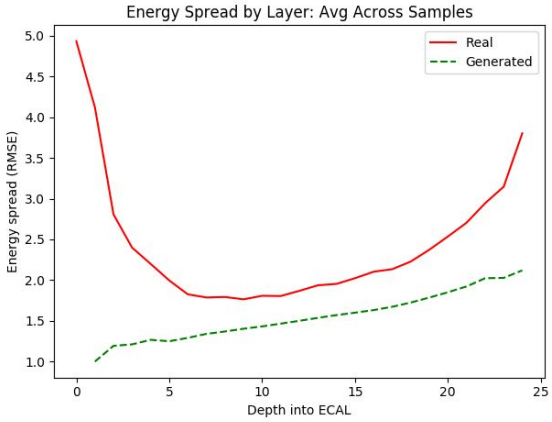

Figure 7: Sample chart comparing energy spread (root mean squared error), by layer, of real and generated images

Distributions

Abstracting a level further, we needed some charts that would give a sense of how well generated data compared to real data in terms of distribution across samples. The “Histogram of Nonzero Cells” chart plots a simple distribution across all individual pixel intensities, across all generated and real samples.

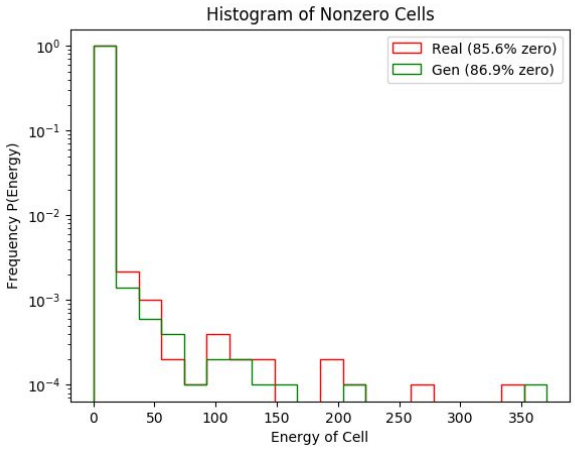

Figure 8: Sample histograms that show the number of non-zero cells in real and model-generated data.

In the next chart we look at the total input energy vs. the total output energy to ensure that the generator learns that particle showers with high energy or momentum coming in should result in higher pixel energy intensities in the calorimeters. Total output energy here is defined as the sum of energy across an entire calorimeter. Total input energy refers to the momentum of the incoming particle shower and is a known parameter in each experiment.

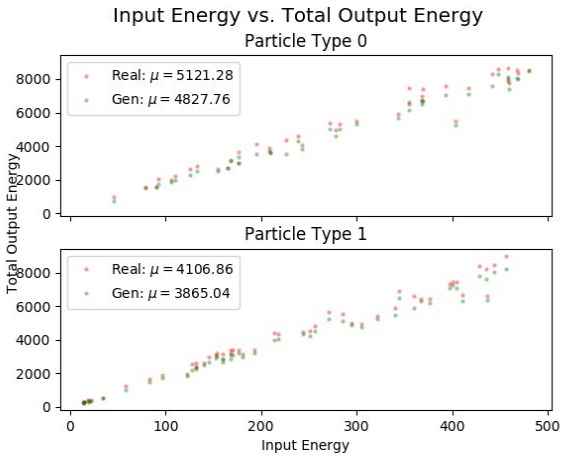

Figure 9 Sample chart demonstrating the ratio of output energy to input energy, for both particle types. A small sample of real data points are shown in red, and data generated by our models is shown in green.

For easier visual comparison, the above scatter plot is converted into a histogram: for each point, we divide the output energy by the input energy; then the distribution across these output-to-input energy ratios is plotted in a histogram.

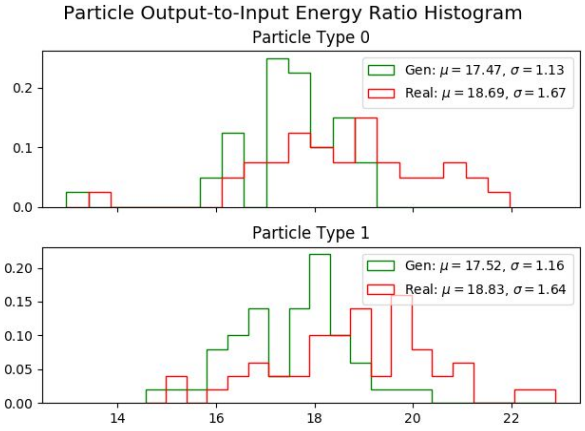

Figure 10 Sample chart of the histogram of the ratio of output energy to input energy, for both particle types. A small sample of real data points are shown in red, and data generated by our models is shown in green.

Recurrent Neural Net

The recurrent neural net (RNN) model is a natural choice for modeling the sequential nature of the event, as energy flows deeper into the calorimeter. The primary goal of generating an entire event given only the input-momentum and particle type is broken down into one-slice-ahead

predictions. The energy deposits in the following calorimeter slice are predicted using particle-type, input-momentum, and energy measured at all previous slices. In other words, P(X|M,P) factorizes into $P(X_t|X_{<t},M,P)$ with t representing the depth in the calorimeter cube. The 3-dimensional calorimeter (20x20x25) is represented as a 2-dimensional (400x25) matrix with the Z-dimension left intact, so that a 400 dimensional vector is used to represent the energy measurements at each slice. (See figure 11 below)

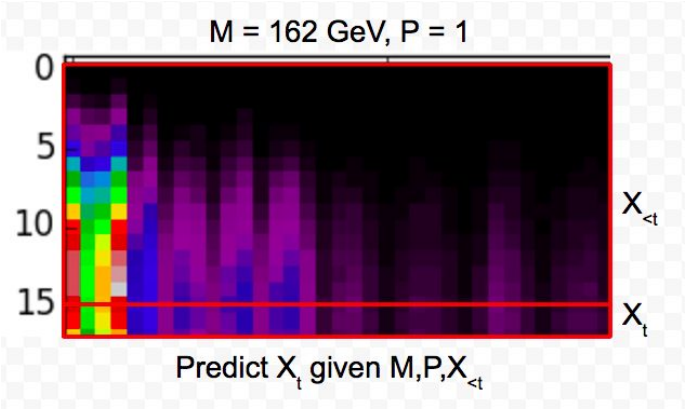

Fig. 11. The goal of the RNN model is to predict each layer given the previous predicted layers, input energy M and particle type P

The supervised (log-likelihood) approach for training the RNN gives it an advantage over the more challenging unsupervised training of the GANs. The model is trained with real data being fed in at every timestep. But when generating full samples, the initial hidden state is fed in for the first timestep (initialization) and then generated data is passed forward through the RNN.

Models

We started with a straight-forward RNN where the particle-type and momentum were embedded into 10 dimensional vectors and concatenated to the input at each time-step, resulting in a 420-dimensional recurrent input. After tuning at various cell sizes, we converged on running all of the RNN experiments with a hidden state of size 800. Both LSTM and GRU cells were trained for the early baseline models, and LSTM performed marginally better. Implementing dropout as a form of regularization (Hinton, et al., 2012) slightly improved performance and increased training time for all experiments. The suite of RNN models trained all consisted of approximately 5 million trainable parameters, and were each run for a maximum of 24 hours or until early-stopping was reached.

By-slice energy increases from the entry until around the 15th slice of the calorimeter (along the Z-axis) and then starts to decrease again (see fig. 13, lower left chart). To account for this dynamic, we added an additional 10 dimensional embedding to indicate the slice-index, as a way to condition the model on the depth within the calorimeter cube. This slightly improved performance and and the model was able to capture the hill-shape of the energy deposits much better. Additionally, we log-scaled the data which provided a slight quantitative improvement,

but did not significantly improve the quality of the generated images or the validation plots. We then attempted to include a more global representation than the pixel level by including a convolutional layer to the model. Each previous slice was put through 2 sets of 4 feature maps with 3x3 convolutions, downsampled with average pooling. This representation was concatenated the existing input, so that the representation of the previous slice consisted of [pixel-level energy, input-momentum, particle-type, convolutional-layer representation]. Surprisingly, adding the convolutional layer, and introducing a larger receptive field did not improve the results.

The initial hidden state (H_0) was also learned, so that the entire event can be generated conditional on just the particle type and momentum. Samples were also generated by introducing the first few slices of real data, and then feeding in the predicted output of the previous slice as input. These samples, which were initialized with the first few timesteps of real data produced improved results, but the results presented below were generated without any training layers fed in.

Results

The various charts below display summary statistics and simulated events generated by the RNN. Our main takeaways are that the RNN was able to match some of the high level moments, as can be seen from the plots which are averaged across samples. We notice that the single event looks far too much like that of the flattened-representation averaged across 1000 events. We hypothesize that this is due to the L2-loss function used in training, as the model as not able to pick up the event-specific signals, it basically predicts the “mean experiment” without capturing enough of the event-specific characteristics. The energy-spread-by-layer plot provides another viewpoint into the same issue, where the model does not provide enough variance in energy across the slice, focusing all of the energy close to the center of the calorimeter.

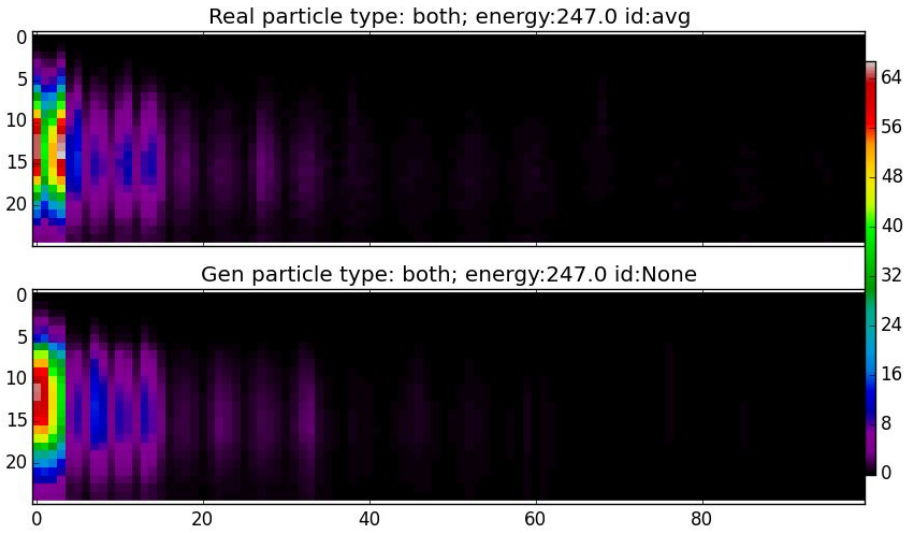

Fig 12a. Flattened representation of avg of 10,000 samples from the test data set (top) vs. from the RNN (bottom)

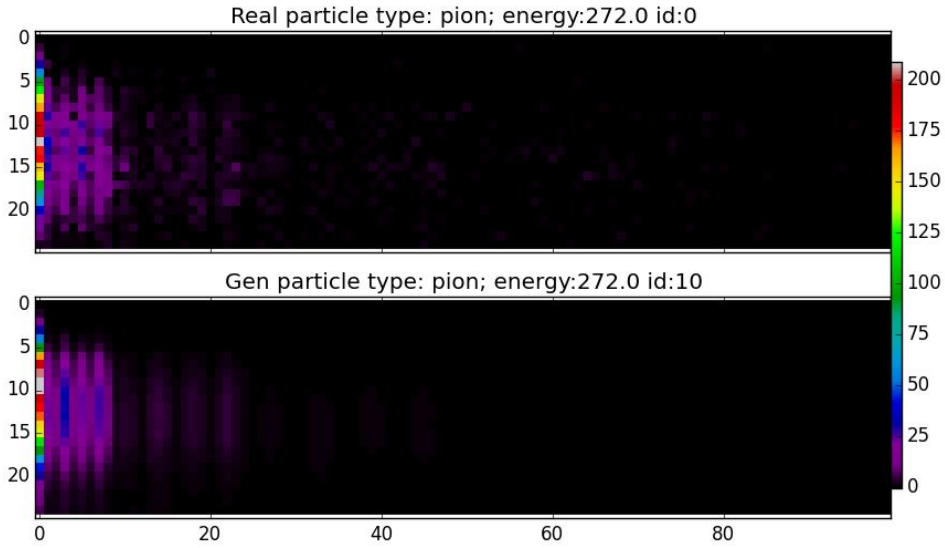

Fig 12b. Flattened representation of a photon shower with energy 272 GeV (top) vs. RNN simulation of same (bottom)

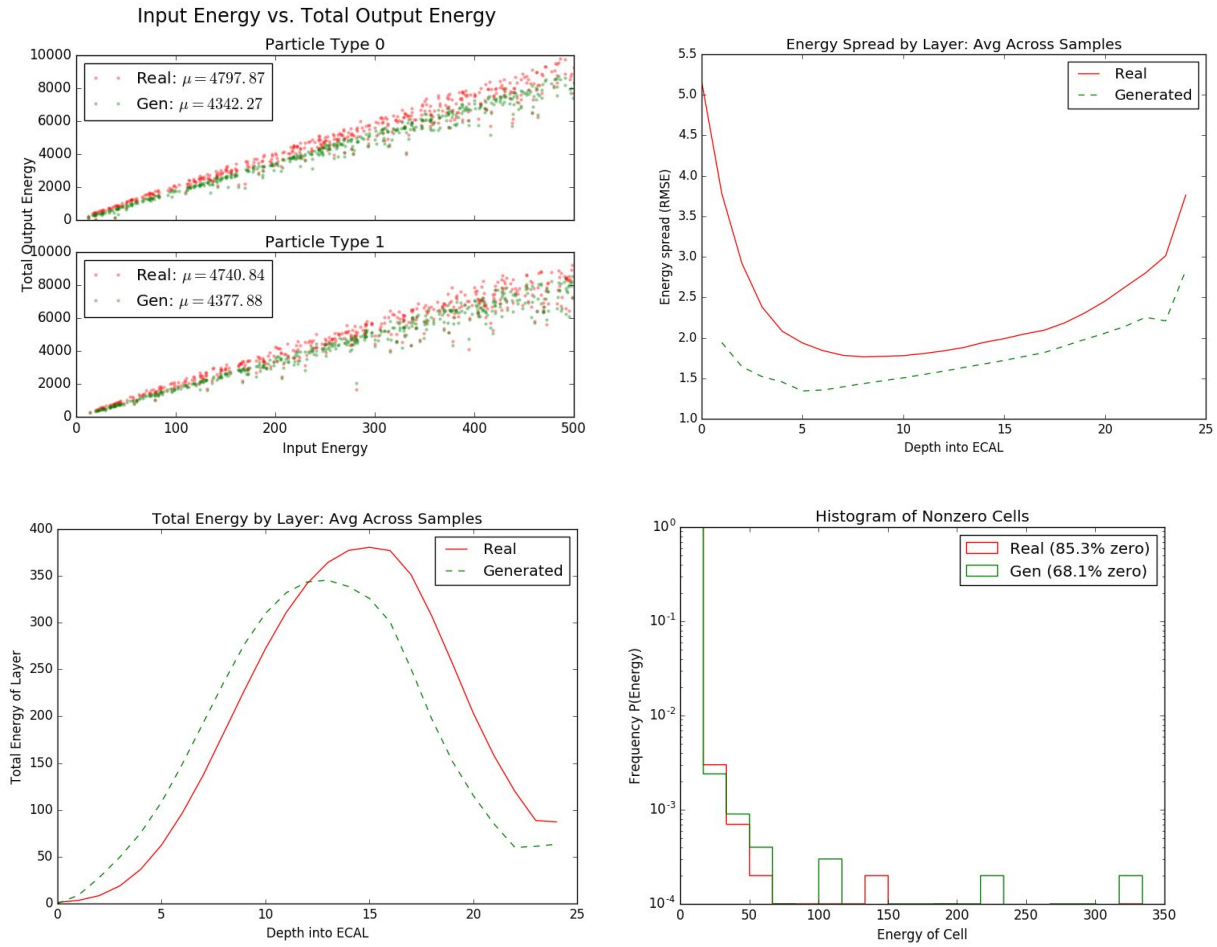

Figure 13.
Top-Left: Scatter plot of input and output energy pairs. Top-Right: Energy spread by layer.
Bottom-Left: Energy at each depth. Bottom-Right: Histogram of pixel intensities
In allcharts, red data points are real data, green data points generated by the RNN.

The central challenge of sparsity and skew in the data were not completely overcome by the RNN approach. In language modelling, which suffers from exactly the same issues, word embeddings offer dense trainable continuous representations for each word. The L2-loss function is not ideal for this type of data, and is known to yield blurry photos in image-generation tasks due to the model tending to predict towards the mean. Generative autoregressive approaches for images (van den Oord et al., 2016) combat this by treating each of the 256 RGB pixel values as categorical classes and implementing a cross-entropy loss function. This is feasible when predicting one pixel at a time in the autoregressive framework, but in the RNN framework, this would mean that the output of each hidden state would need to then be passed to 400 individual softmax layers, which would output a distribution over the number of bins the data would be split into. This is not feasible in our task, as it would require binsx400xhidden-dim parameters to be learned at the output layer, which is challenging both from a computational and modeling perspective. As mentioned above, we tried normalizing the data by applying log-transforms.

Generative Adversarial Network

We also tried using a generative adversarial network (GAN). We chose to design the GAN so that it produced a flattened, two-dimensional, representation of the the three-dimensional calorimeter data. This allowed us to build a model that could be directly drawn from two-dimensional GAN architectures designed to produce two-dimensional natural images. We built two different models: one relatively simple multilayer perceptron (MLP), and one derived from the “Location-Aware” GAN (LAGAN) designed by (de Oliveira et al., 2017).

The calorimeter data was flattened using the spiraling technique described earlier, changing the dimensionality of the data from 20x20x25 to 400x25. We found that the large dimensionality of the data difficult made it difficult to work with. Nearly all standard model architectures became too large to fit into memory with data of this size. To combat this, we experimented with models that cropped the data around the edges, reducing its size to either 100x25 or 25x25, depending on the experiment. The energy values in these regions of the calorimeter are always very close to zero, so little information is lost. The reduced dimensionality of the data makes it simpler for a model to learn, since the output size is reduced, and the pixel distribution becomes more uniform. We also zero-centered and normalized the data by its standard deviation, ensuring the pixel energies were on roughly the same scale.

We used the Wasserstein GAN algorithm (Arjovsky et al., 2017) to train our GAN. This training method promises increased training stability, meaningful training loss plots, and a solution to the mode collapse problem, compared to the original GAN method. We adapted the Wasserstein training code from in implementation by Thibault de Boissiere[2] to our setting.

MLP GAN

This model is very simple, as was intended to serve as a baseline for more complex models. Both the generator and discriminator largely consist of a fully-connected layers.

The generator’s primary input is a “latent noise vector” with 200 elements, where each one a real number sampled from a standard Gaussian distribution. There are two “conditional” inputs that are intended to parameterize the network’s output: a bit indicating the particle type to be generated, and a single real number indicating the energy of the particle beam. We create an embedding vector for the particle type of the same size as the latent vector, and project the input energy value to 200 dimensions with a fully-connected layer. We then multiply these two vectors with the latent noise vector element-wise. The resulting 200-dimensional vector is then fed into a dense fully-connected layer, and then reshaped into the final output.

[2] https://github.com/tdeboissiere/DeepLearningImplementations/tree/master/WassersteinGAN/

The discriminator is only slightly more complex. The input image is put through a 5-by-5 convolution with 32 filters, and then flattened into a one-dimensional vector. The particle type and input energy parameters are concatenated with this vector, which is then fed to a fully-connected layer with a ReLU nonlinearity. Finally, the fully-connected layer is connected to another fully-connected layer with one node, which serves as the final output.

We ran this model on the uncropped data. Each flattened “image” has 400-by-25 pixels. The mean of the data was already zero, so the only data pre-processing we performed (other than the flattening operation) was to normalize each pixel energy value by the standard deviation of the overall pixel energy on the training data set. The high-dimensionality of the data makes the number of trainable parameters in this model very large, even though the architecture is so simple. The architecture requires 2,010,800 parameters for the generator, and 2,560,865 for the discriminator. We used the training hyperparameters from the Wasserstein GAN paper (Arjovsky, et al., 2017): a batch size of 64, learning rate of 5e-5, and we trained the discriminator on 5 batches times for every one batch we trained the generator. We trained it for five epochs, where each epoch required roughly 2.5 hours.

Experimental Results

This model performed surprisingly well, in spite of its simplistic architecture. Figure 14 illustrates that the average of 10,000 images sampled from the GAN matches the average of images taken from the real data quite closely. Averaged statistics taken on samples from the GAN match the statistics from the real data are also very similar, as shown in figure 15.

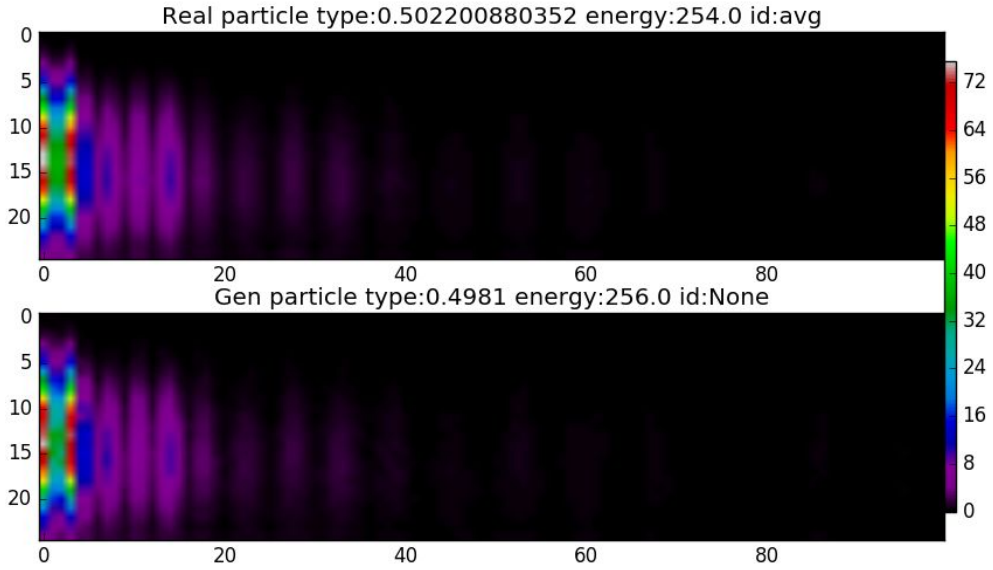

Figure 14. Flattened representation of avg of 10,000 samples from the test data set (top) vs. from the fully-connected GAN (bottom)

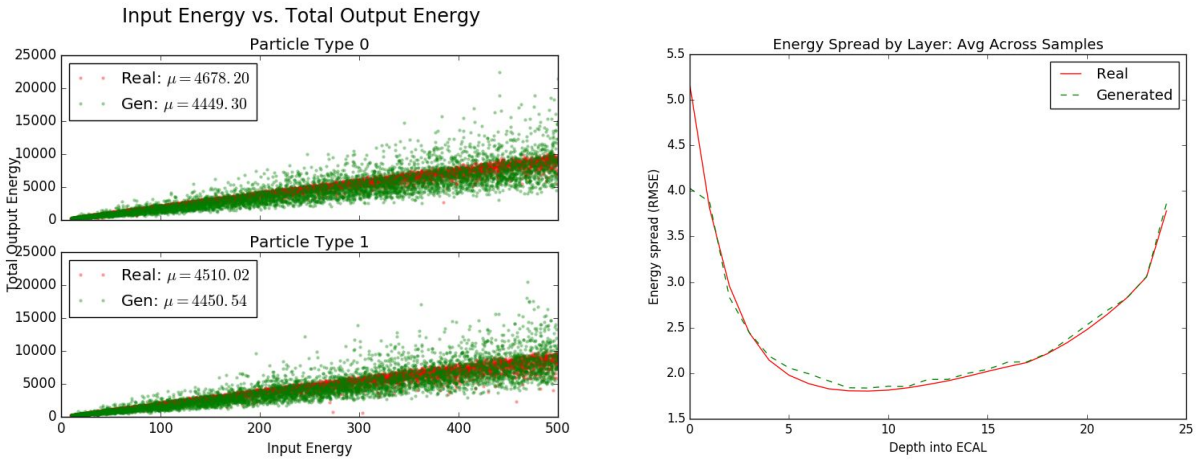

Figure 15. Left: Scatter plot of input and output energy pairs. Right: Energy spread by layer. In both charts, red data points are real data, green data points generated by the fully-connected GAN.

However, samples individual samples from the GAN do not qualitatively match the real events. The generated images lack the smooth transitions found in the real data, as illustrated in figure 16 and 17. The model was clearly able to learn the global characteristics of the data, but not the local features. The “location-aware” GAN that we discuss below was designed to fix this exact problem.

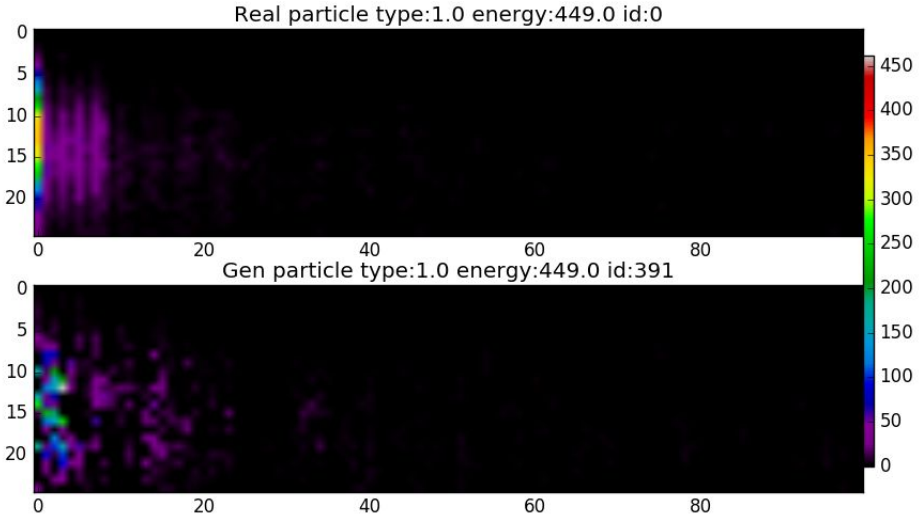

Figure 16. Flattened representation of a photon shower with energy 449 GeV. (top) vs. GAN simulation of same (bottom).

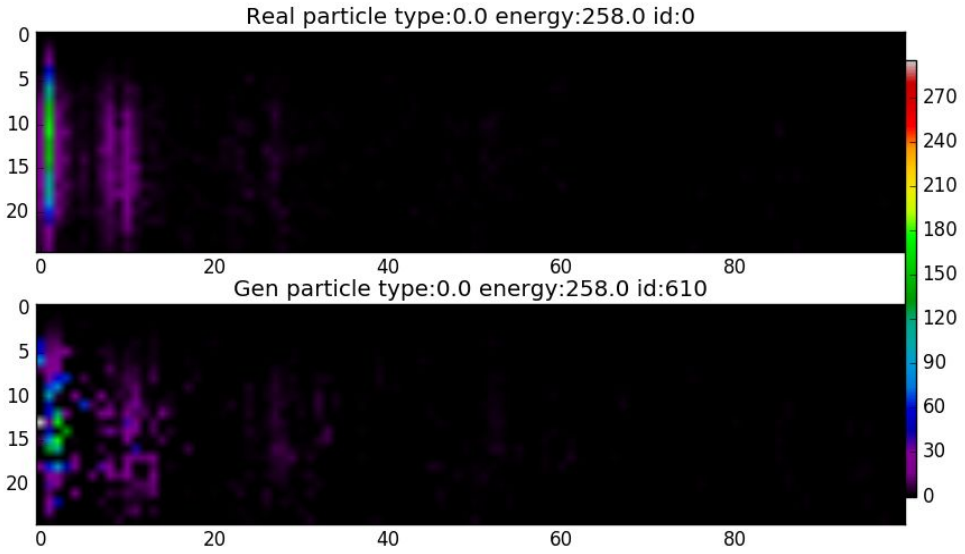

Figure 17. Flattened representation of a pion shower with energy 258 GeV. (top) vs. GAN simulation of same (bottom)

Location-Aware GAN

The architecture for this version of the GAN was based directly on the “Location-Aware GAN” (LAGAN) designed by (de Oliveira et al., 2017). That model was designed to generate “jet images”, simulations of the two-dimensional radiation patterns created by a streams of high energy hadrons. These images are very similar to our data. Like our calorimeter readings, jet-images are sparse with non-smooth features. For this reason, we decided to adapt their model, which is illustrated in figure 18, to our data.

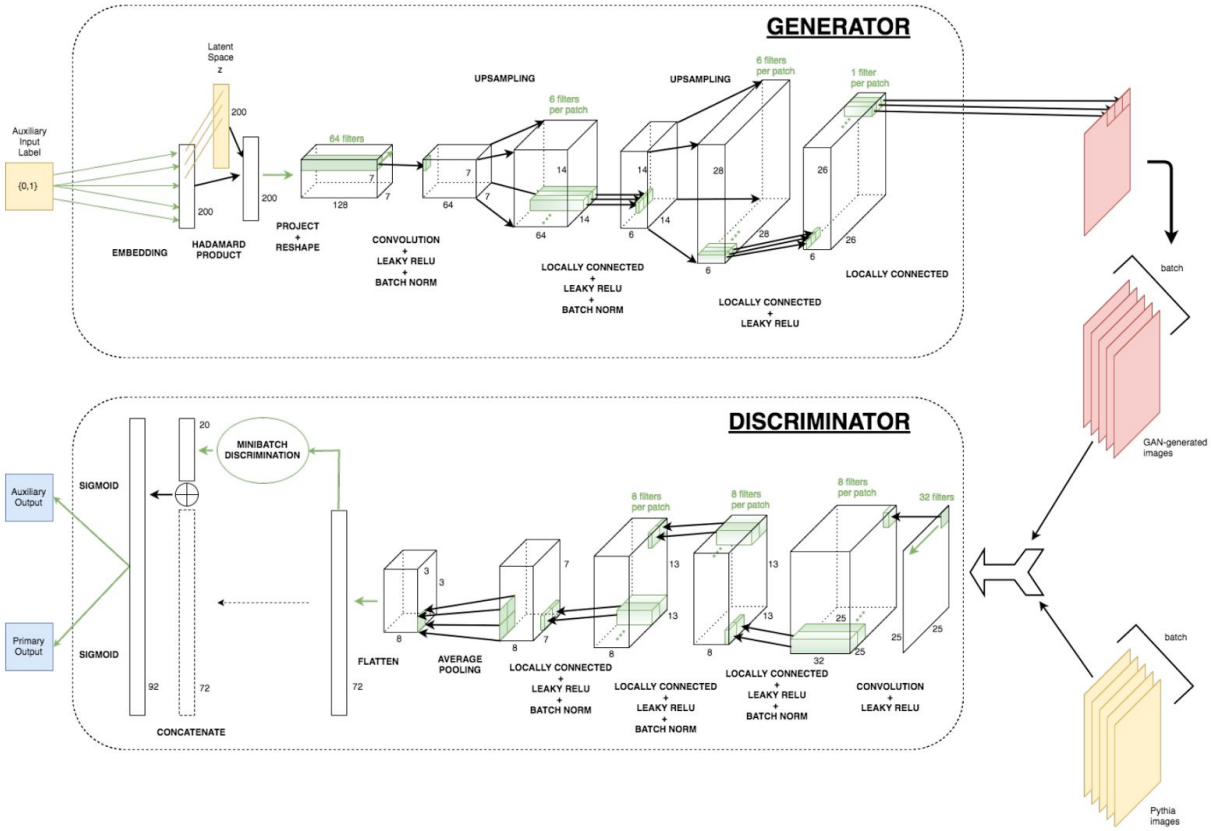

Figure 18. The original LAGAN architecture from de Oliveira et al. (2017).

The locally-connected layers require large numbers of parameters, since every two dimensional patch of the input is assigned a unique filter. As a result, the LAGAN model cannot used the 400x25 pixel images used by our MLP GAN architecture, because too much memory would be required. In order to use the LAGAN model, we cropped the calorimeter images down to 5x5x25, resulting in 25x25 pixel images when they are flattened. Coincidentally, the images used by the original LAGAN architecture were also 25x25 pixels, allowing us to adapt their model very closely.

The generator takes the “latent space” vector, a one-dimensional vector with 256 elements of Gaussian-distributed random numbers, and a single bit indicating the particle type to be generated. These inputs are merged in the same fashion as our MLP model. The resulting vector is then put through a fully-connected layer, and reshaped into two dimensions. This 2D tensor it then put through a convolutional layer, followed by two locally-connected layers. Between each of these layers is an upsampling layer that doubles the size of the 2D tensor in each dimension. The final output is produced by another locally-connected layer with a ReLU activation.

The discriminator has a similar architecture to the generator. The input image is put through a convolutional layer with 32 5x5 filters, followed by two locally-connected layers with leaky-ReLU activations. The results of the locally-connected layers are then put into an average-pooling layer and flattened. We then introduced the conditional information, the particle type and input energy, by concatenating them into this layer. This was fed into a fully-connected layer with a single node output, which served as the final output of the model. The original architecture calls for three locally-connected layers, but we only used two in order to fit the model into memory. We also used 2x2 strides in the first fully-connected layer, to reduce the number of parameters to a reasonable level.

Although this model has more layers than the MLP model, it has far fewer parameters. This generator requires 858,429 trainable parameters, and a discriminator requires 855,067. This is largely due to the fact that this model uses a cropped 5x5 image as input, while the MLP model did not crop the input images. I could have increased the number of parameters in the model, but decided against it since this model already uses 68,540 parameters per pixel, far more than the 457 parameters per pixel used by the MLP model.

In most experiments, we trained it for five epochs, and each epoch took about three and a half hours to train. We found that this model was often prone to gradient collapse, where the gradient of the discriminator and the generator flattens out to zero and ceases to change, as illustrated in figure 19. It is not clear to me why this occurred, as it only happened occasionally.

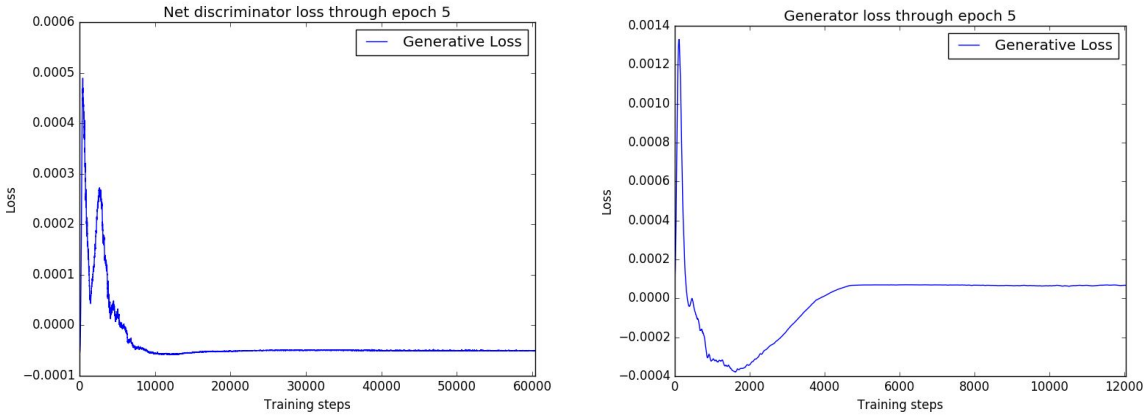

Figure 19: Training stagnation in the LAGAN model.

Differences from the original LAGAN model

There were several differences between the original LAGAN model and the one we implemented. We did not use the “mini-batch discrimination” technique (Salimans, et al., 2016) used by the LAGAN model’s discriminator to prevent the mode-collapse problem. The Wasserstein GAN method asserts that it fixes the mode collapse problem, so it we did not attempt to implement it.

The original LAGAN model had an auxiliary output, in the style of ACGAN (Odena et al., 2016) in its discriminator that predicted the particle type of the image. The additional gradient of this loss function was added with that of the normal discriminator loss. We did not use this auxiliary output, because the loss function of the Wasserstein objective is not directly compatible with it. The output of a Wasserstein GAN is an unbounded real number representing the model's judgement of the "realness" of the image, while the ACGAN was designed for the categorial output of the traditional GAN. Our model also has an additional conditioning parameter: the input momentum. We could have tried adapting the ACGAN's techniques to this model, but we decided to incorporate the conditional information directly as an input instead.

The original LAGAN model did not incorporate conditional information into the discriminator, but we did here. We tried several different methods of incorporating the conditional information, which we discuss in the next section.

Experimental Results

Our variant of the LAGAN model generally did not learn the statistics of the data set well, as illustrated in figure 21. However, the average of its generated image does qualitatively match the averaged images from the test data set quite well, as shown in figure 20. It also produces much more realistic individual images than the MLP model does.

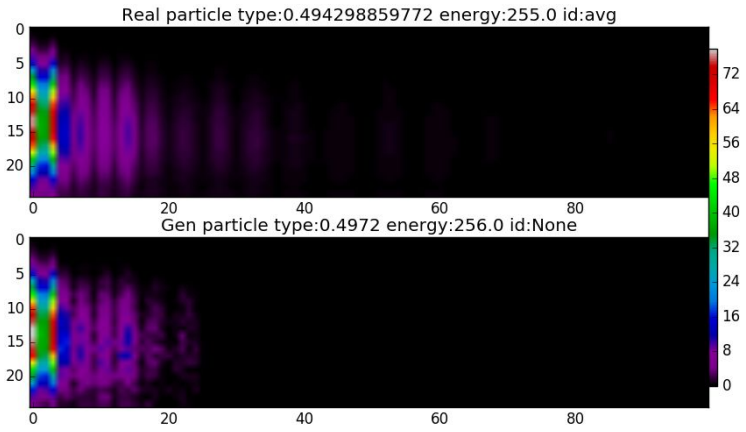

Figure 20. Above: 10,000 samples from the test data set averaged together. Below: 10,000 samples from the LAGAN averaged together.

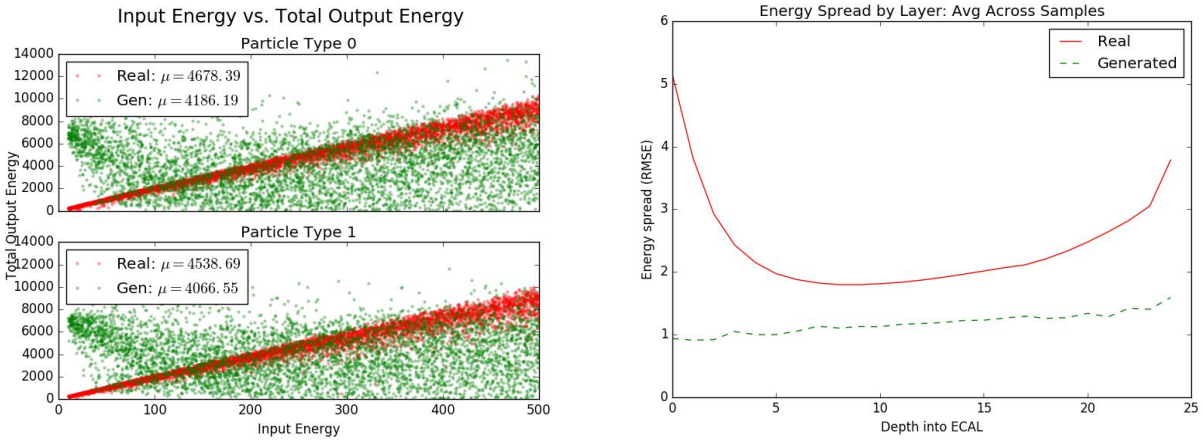

Figure 21. Left: Scatter plot of input and output energy pairs. Right: Energy spread by layer. In both charts, red data points are real data, green data points generated by the LAGAN.

If you look at figure 22, you will notice that the image has smoothly changing features that look very similar to real images. However, it did not learn the relationship between the input and total output energies at all. This is demonstrated in figure 23, where you will notice that although the input energy to the model was relatively small (94 GeV), the generated image looks more like a high-energy image.

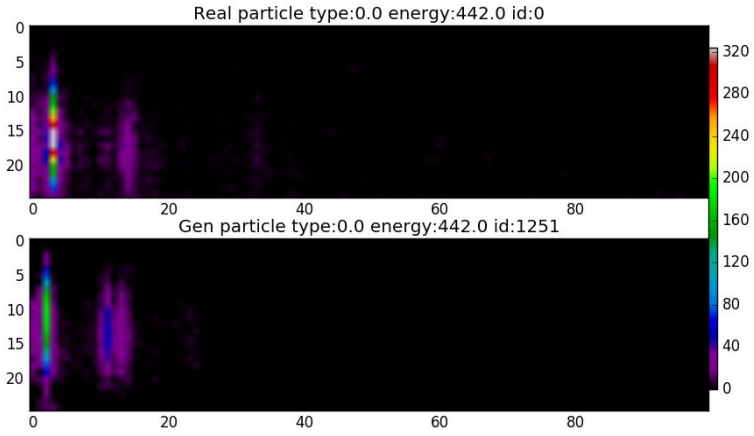

Figure 22. Above: A flattened representation of a pion shower with energy 442 GeV. Bottom: A sample from the LAGAN of a pion shower at energy 442 GeV.

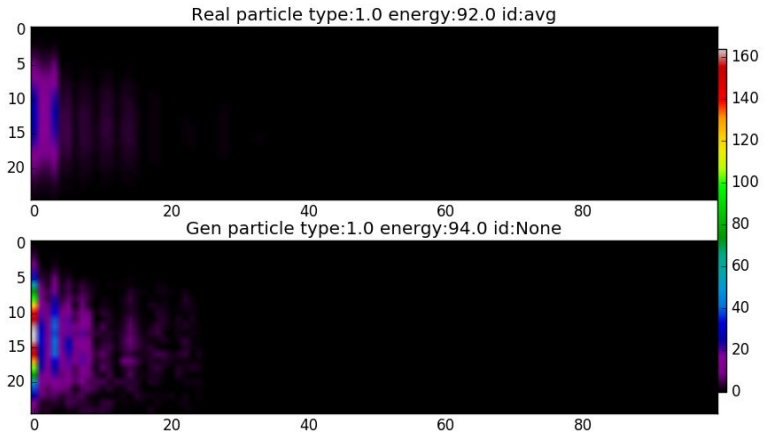

Figure 23. Above: A flattened representation of a photon shower with energy 92 GeV. Bottom: A sample from the LAGAN of a photon shower at energy 94 GeV.

Other Experiments

In some experiments, we attempted to address the sparsity in the data by averaging pixels into evenly sized (particle-type, input-momentum) bins. Each average was generated by collapsing 10 samples into a single average-sample. We found that model results were not typically much better, surprisingly. In the case of the LAGAN model, we did find that the model was sometimes, although not always, able to partially learn the relationship between the input and output energy, as illustrated in figure 24.

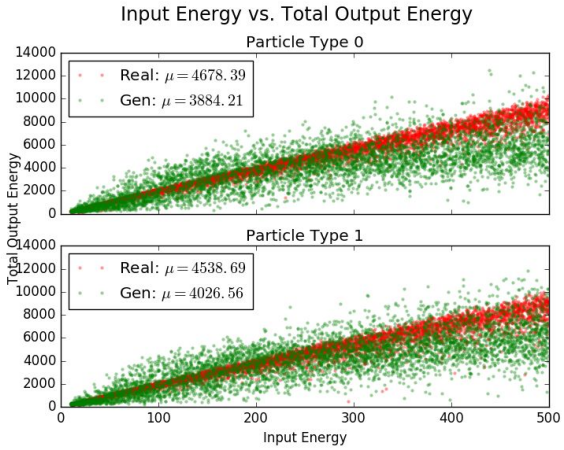

Figure 24. The output-to-input energy ratio for the LAGAN model, trained 20 epochs on the averaged dataset.

This is likely due to the fact that the decreased size of the averaged dataset allowed us to train the model on far more epochs. Figure 24 was generated when we trained the model for 20 epochs on the averaged dataset, in contrast to the 5 epochs we trained the model on the original dataset.

We also tried to introduce the conditional information by introducing it earlier in the discriminator model, similar to how we do it in the generator. In that version, the particle type bit was transformed into into a 25-element vector embedding, and the input energy was projected into a 25-element vector with a fully-connected layer. These two vectors were then reshaped to two dimensional 5-by-5 arrays and multiplied elementwise with the input image. Surprisingly, the output of this model was much worse than other variants.

We experimented with how the latent noise vector was generated. Since all the real images have a high energy center that then quickly diminishes, we tried generating noise vectors that showed a similar pattern, instead of having each pixel be generated by an independent Gaussian distribution. Surprisingly, this technique diminished the quality of the resulting images. I think this technique is worthy of further investigation, however, since the StackGAN model proved that a similar technique could result in very high quality images (Zhang et al., 2016).

Lessons Learned

We achieved more success with our RNN model than with our GAN model. Due to the opaque nature of neural networks, it's difficult to know why that might have been. It could been that the RNN was able to more readily learn the temporal patterns of the data since it processed each layer of the z-axis in-sequence. But it could have also been that the hyperparameters we used for the GAN, which we took from the Wasserstein GAN paper (Arjovsky, et al., 2017), were simply inappropriate for this setting, and a small change could have led to dramatically different results.

The only way to test these theories is through experiment, and experimenting is easier with an RNN than it is with a GAN due to the fact that RNNs are supervised models are GANs are unsupervised. The supervised loss function gives a clear signal on when training is finished, since the patterns of convergence and overfitting are easy to identify in the graph of training loss. In contrast, there is no clear way to know when an GAN has finished training. The Wasserstein GAN paper reported that the algorithm resulted in loss function curves that more directly correlated with the quality of its output (Arjovsky, et al., 2017), but we did not observe this in our experiments. As a result, when training the GAN, we simply chose an arbitrary number of steps to train, which resulted in experiments that required more than a dozen hours to train. The RNN, however, typically achieved convergence within eight hours.

Another difficulty we faced was in correctly implementing the Wasserstein training procedure. The algorithm itself is simple, but detecting errors in the algorithm was difficult, since the model was often able to learn decent-looking approximations of the underlying distribution of the training data despite errors in the algorithm's implementation. As a result, we repeatedly misdiagnosed errors in the algorithm as faults with the model architecture. Every time an algorithmic error was discovered, every previous experiment had to be re-run. Since training the

model requires more than a dozen hours for each experiment, this problem slowed down development considerably. This could have likely been partially avoided had we taken the time to write tests for our training code. Tests are difficult to write, and even more difficult to maintain, in a rapidly changing codebase, but in this case, the time saved debugging would have likely been worth the cost.

Alternative Considered: PixelRNN

PixelRNN (van den Oord et al., 2016) is another generative model designed to generate images. It tries to model the joint probability distribution of the training image set by factorizing the distribution, conditioning the distribution of each pixel on those that came before it. The the bulk of the model is an LSTM that predicts the next row of pixels given the previous rows. The weights of the model are masked to ensure each pixel only receives information from previous pixels. During training, the model is given an image from the training set, and the model tries to generate the same image, like an autoencoder. This model has been shown to generate realistic images. Additionally, it has an interpretable loss function, unlike a GAN.

However, we decided not to use this model. The model assumes that the pixels of each image are integers within a pre-specified range, since the output layer is the softmax function. This works well for actual images, since they are pixelated into a preset number of colors, but it is not appropriate for our calorimeter data. Our data has a much larger range of values, and the vast majority of “pixels” fall between zero and one. We may have been able to adapt our data to the model by pre-processing it to have a more uniform distribution. However, the resulting model would have had to have far more bins that the 256 used by the original model, which would have likely made the model less effective.

More importantly, this model generates output pixel-by-pixel, which we found to be too slow to be competitive with existing generative techniques for this data. In our tests, we found that each pixel could take more than a second to generate, requiring several minutes for a single full image. There is a paper (Ramachandran et al., 2017), however, that speeds up this performance by caching the results of previous model states, but each image still takes up to a minute to produce. Since I understand that the current Monte-Carlo-based techniques require on the order of 10 seconds to generate an image, this model seemed too slow to be useful.

Appendix

We used the Keras framework with the Tensorflow backend to build and train our models. All of the code used for this project is available at https://github.com/pinesol/lhc-calo.

Citations

Bendavid, Josh, et al., “Imaging Calorimeter Data for Machine Learning Applications in HEP.” (2016).

Hochreiter, Sepp and Schmidhuber, Jurgen. Long short-term memory. Neural computation, (1997).

Goodfellow, Ian, et al., "Generative adversarial nets," Advances in Neural Information Processing Systems, (2014).

Arjovsky, et al., “Wasserstein GAN,” arXiv preprint [arXiv:1701.07875] (2017).

Michela Paganini, Luke de Oliveira, Benjamin Nachman. “CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks.” arXiv preprint [arXiv:1705.02355] (2017).

Hinton, Geoffrey, et al., “Improving neural networks by preventing co-adaptation of feature detectors”, CoRR abs/1207.0580 (2012).

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In International Conference on Machine Learning (ICML), 2016b.

L. de Oliveira, M. Paganini and B. Nachman, “Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis,” arXiv preprint [arXiv:1701.05927] (2017).

M. Paganini, L. de Oliveira, and B. Nachman, “CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks,” [arXiv:1705.02355] (2017)

Salimans, Tim, et al. "Improved techniques for training GANs." Advances in Neural Information Processing Systems (2016).

De Boissiere, Thibault: “Generative Adversarial Networks.”. Github Repository. https://github.com/tdeboissiere/DeepLearningImplementations/tree/master/WassersteinGAN/

A. Odena, C. Olah and J. Shlens, Conditional Image Synthesis With Auxiliary Classifier GANs, arXiv preprint [1610.09585] (2016) .

Zhang, Han; Xu, Tao; Li, Hongsheng; Zhang, Shaoting; Huang, Xiaolei; Wang, Xiaogang; Metaxas, Dimitris. “StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks.” eprint arXiv:1612.03242 (2016).

Ramachandran et al. “Fast Generation for Convolutional Autoregressive Models.” eprint arXiv eprint [1704.06001] (2017).

International Linear Collider. 2013. https://www.linearcollider.org/ILC (visited May 2017)