Investigation into the Efficiency and Effectiveness of Diffusion Modeling in Predicting Calorimeter Particle Showers

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements of the Degree
Bachelor of Science, School of Engineering

Luke Ostyn
Spring, 2024

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Geoffrey Fox, Department of Computer Science

Introduction

Calorimeters are a staple of particle physics, used ubiquitously to determine the energy associated with various particles. When these particles enter the calorimeter they break apart rapidly into multitudinous lower energy particles, a so-called particle shower. Traditionally, attempts to model these showers have been overly laborious and computationally expensive, meaning that the development of new, more lightweight methods is of great import (Mikuni & Nachman, 2022). For my technical project, I explored the use of deep learning methods to produce high accuracy predictions of these calorimeter showers. Specifically, I investigated the efficacy of normalizing flows models by expanding on the CaloScore diffusion model of Mikuni and Nachman (2023). Beyond just looking to increase the accuracy of the model, I also attempted changes designed to increase the speed with which the model could be trained. Although the model once trained is significantly faster than traditional methods, the training can certainly be a roadblock especially when lacking access to a distributed system. As such, I thought it pertinent to look into methods to abbreviate the training without substantially compromising model accuracy. The original model and my subsequent alterations were built to perform against the datasets available from the Fast Calorimeter Simulation Challenge 2022 hosted by Kaggle.

The Model

Broadly speaking, diffusion models are trained by taking our data and then through an iterative process adding and removing noise; we take our data, add noise to it, and then try and train a model that is able to recover that original data. When we then pass something resembling noise back through the model it should produce an output similar to the data we trained it on (Chang et al.). For the base CaloScore model, this involves training just under two and a half

million different parameters on data passed throughout a series of different networks. This data is composed of the input energies associated with the particle entering the calorimeter and then the energies associated with individual physical voxels and layers during shower generation within the calorimeter. As explained by Mikuni and Nachman, in a given training cycle, noise is added to the layer and voxel data before they are input to Resnet and U-Net architectures respectively. The Resnet architecture consists largely of a series of convolutional layers while upsampling, residual, and downsampling layers comprise most of the U-Net architecture (2023).

Methods

The methods developed for testing variations on this model were in part constrained by limitations of the system upon which they were trained. Because the original model was designed for a distributed system, utilizing parallelization techniques which were removed for the purpose of training on my personal laptop, the runtime for a single training epoch ballooned quite considerably, up to approximately two hours per epoch. With the training process taking up to three hundred epochs to complete, obviously fully training the network for multiple different variations was simply unfeasible. Instead, I opted to look at loss statistics after an epoch or two. Moreover, for similar motivations related to computational intensity, I opted to test these changes using solely the first dataset available from the Kaggle challenge.

Generally, my attempted tweaking of the model took one of two different forms: adjustments to either the architectures themselves or to the hyperparameters in charge of controlling the process by which the weights associated with these architectures are updated. In the first category are changes to both the U-Net and Resnet as well as the greater CaloScore network which frames them. For the U-Net, these changes include an increase to the numbers of residual layers and a separate decrease in the number of sampling layers (both upsampling and

downsampling). As for the Resnet, I experimented with adding a significant number of convolutional and dropout layers. Finally, I modified the process by which noise is added, generally decreasing the degree of noise appended. As for hyperparameter modifications, I tested various learning rates and data sampling time steps, observing in large part whether convergence could be faster induced.

Results and Discussion

After the completion of a training epoch, several different loss performance metrics are reported. These include loss, voxel loss, and layer loss, where loss is just the sum of the voxel loss and layer loss. As the names suggest, voxel and layer loss each report a square loss associated with the independent voxel and layer models. These statistics are reported in Table 1 for a myriad of different modifications to the default CaloScore model. Critically this data will

Table 1

Loss Statistics for Assorted Modifications to the CaloScore Normalizing Flows Model

Modification	Loss	Voxel Loss	Layer Loss
Base	1.1396	0.4949	0.6477
Removed U-Net Sampling Layers	1.2400	0.5935	0.6464
Increased U-Net Residual Layers	1.1330	0.4913	0.6417
Increased Number of Resnet Convolutional Layers	1.1354	.4971	.6383
Increased Learning Rate	1.0310	0.4286	0.6024
Fewer Time Steps	1.1396	0.4949	0.6447
Decrease Noise	2.4349	1.5328	0.9021

help us to answer the dual questions of whether these modifications were able to improve performance by either decreasing loss or keeping loss stable and increasing speed.

Let us first examine those changes made to the CaloScore architecture. More specifically, we will start with those made to the U-Net voxel model. The first alteration made was to remove the sampling layers, which reduced the number of trainable parameters in the overall network by a factor of about four. Despite this significant drop in network complexity, we failed to see any serious reduction in epoch training time. This fact, in addition to the loss and voxel loss numbers being worse than the baseline, indicates that these sampling layers are necessary and should not be excluded. To the contrary, when we increased the number of residual layers in the same U-Net, we observed a slight decrease in voxel loss and the attendant decrease in total loss. While the change obviously did not help to reduce training time, the data suggests that boosting the number of residual layers can produce small payoffs. Similarly, increasing the number of convolutional layers within the Resnet prompted the total and layer loss values to exhibit small improvements. In the same vein as the U-Net modification, this shows that an increase in trainable layers can lead to small dividends in decreasing loss. Finally, attempting to decrease the amount of noise added to the model produced loss statistics significantly worse than that of the normal model. Moreover, the training lost witnessed a significant plateau over the course of the run, suggesting that little improvement was occurring.

As for the final two changes, these changes were largely implemented with the goal of reducing training time. With the increased learning rate, we can discern that after a single epoch all of the model's loss statistics were superior to that of the base model, which is indicative of the fact that the higher learning rate induces faster convergence to a reasonable model. While the final result after hundreds of epochs might be inferior to that produced by the base model with

the stunted learning rate, for the purposes of faster training, increasing the learning rate proved effective. Secondly, I looked into drastically reducing the number of time steps involved in creating samples during the training and testing stages of the CaloScore model. Curiously, doing so resulted in no change at all to the loss statistics but decreased the runtime significantly, demonstrating that at least for the first kaggle dataset, cutting down the number of time steps involved in sample generation is a method worth exploring.

Conclusion

Calorimetry, an evergreen field within particle physics, is notoriously difficult to build models for because of the exceptional computational power needed. This is where sophisticated deep learning techniques enter the fray, with the hope that we can use them to train vast networks and unlock previously untapped modeling capabilities. One such technique, normalizing flows, was explored in this paper through the lens of the CaloScore module. The primary focuses were to both improve its effectiveness and the speed at which it could be trained by altering various parameters. The results obtained submit that greater effectiveness may be possible when both the U-Net and Resnet components of the model are expanded to include more residual and convolutional layers respectively while increasing training speed follows from a heightened learning rate or a cutback on the number of time steps during sample generation. Future research should be geared towards testing these modifications for a greater number of epochs on a more capable distributed system.

Works Cited

Chang, Z., Koulieris, G. A., & Shum, H. P. H. (2023, October 19). *On the design fundamentals of diffusion models: A survey*. arXiv. https://arxiv.org/abs/2306.04542

Mikuni, V., & Nachman, B. (2022, October 19). *Score-based generative models for calorimeter shower simulation*. arXiv. https://arxiv.org/abs/2206.11898

Mikuni, V., & Nachman, B. (2023, August 7). *CaloScore V2: Single-shot calorimeter shower simulation with diffusion models*. arXiv. https://arxiv.org/abs/2308.03847