UNIVERSITY OF CALIFORNIA

SANTA CRUZ

AUTOMATED ANALYSIS OF SIMULATED LARGE HADRON COLLIDER EVENTS

A thesis submitted in partial satisfaction of the

requirements for the degree of

BACHELOR OF SCIENCE

in

PHYSICS

by

Yanall Boutros

1 March 2020

The thesis of Yanall Boutros is approved by:

Professor Jason Nielsen

Adviser

Professor Michael Dine

Chair, Department of Physics

Abstract

Automated Analysis of Simulated Large Hadron Collider Events

by

Yanall Boutros

The current workflow for simulating Large Hadron Collider (LHC) events is currently standardized by the ROOT Data Analysis Framework. ROOT is written in C/C++ and includes a python package to allow function calls to native C code. C/C++ is a fast language, and ROOT includes a docker image to make it portable as well. ROOT, however, is not modern. Its code base is bloated, and includes a great deal of workounds, "20 years worth"(Buckley, 2007). We aim to begin the transition of the data analysis framework from ROOT to a python environment, utilizing packages such as: *Pythia, Pyjet, Matplotlib, and TensorFlow* as the backend to simulate our data. Additionally, we dockerize our environment to further improve portability of the framework.

For proof of conecept, as well as for documentation purposes, several example programs are included which slowly introduce concepts and practices of coding common routines of particle physics in python, as well as demonstrating the effectivenes of our neural networks in analyzing the data generated in this framework. 1D histograms are generated from the cluster of particles resulting from the creation of a new quark. This cluster of particles is known as a jet, and information such as where the jet is located, how much energy and momentum it contains, and the radius of the jet are used as parameters for training an Artificial Neural Network (ANN). 2D histogrmas weighted by the energy of each particle constituting a jet is also utilized in training a different ANN structure. In either network, we find an accuracy in our validation data set of 79.9% when trained over $13,000$ events, with a validation subset reserving 10% of those events for verification.

1 Introduction

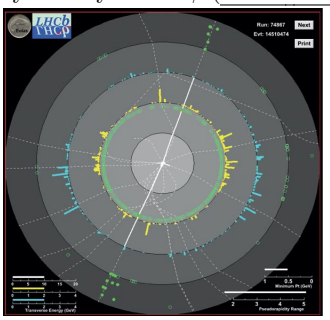
1.1 High-Energy Particle Collider

The Large Hadron Collider (LHC) is a particle accelerator which causes protons to impact at high energies, around 14 GeV in the context of this research. Hadrons are subatomic particles composed of quarks bound together by the strong nuclear force. Not only does the strong nuclear force bind quarks together to form protons and neutrons, but it also binds protons with neutrons to form the nucleus of an atom.

Collisions at high energies cause the formation of new hadrons during a process called hadronization. Since quarks can only exist in either quark and anti-quark pairs or triplets, we can only infer their existence from observing the jets produced. A jet is a shower or cluster of particles which depends on the initial properties of the collision event, such as which quarks or other fundamental particles were created from the collision (de Oliveira, Kagan, Mackey, Nachman, & Schwartzman, 2016). These particles are measured by interactions with four main detectors located around the LHC: A Large Ion Collider Experiment (ALICE), A Toroidal LHC Apparatus (ATLAS), Compact Muon Solenoid (CMS), and LHC beauty (LHCb).

Individual particles have their energy and momentum measured at a spatial coordinate relative to the azimuthal angle of the cylindrical track ϕ, as well as the pseudorapidity η, the angle of the particle relative to the beam axis (Komiske, Metodiev, & Schwartz, 2017). **Figure 1** displays features emphasized by azimuthal symmetry,while **Figure 2** shows similar information from another perspective, so the angle between the beam track and the jet η can be observed.

Figure 1: (Left) Diagram of inside of a collider track with the axis of symmetry focused on η, emphasizing azimuthal symmetry about ϕ (LHCb looks forward to electroweak physics, 2012).



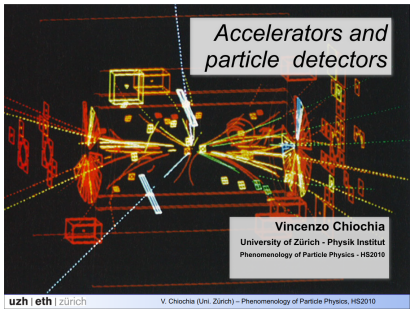


Figure 2: (Right) Diagram displaying features from both η and ϕ(Chiochia, n.d.).

Particles can then be clustered into jets, and macroscopic properties of the jets are either measured or inferred. These properties include the mass, spatial coordinates, transverse momentum and energy, and the number of particles constituting the jet. **Table 1** serves as a sample of some of these properties and how that data is structured. The decay products are detected, their physical properties measured, and that data is analyzed and used to determine what particles were initially created from the collision.

1.2 Representations of Simulated Data

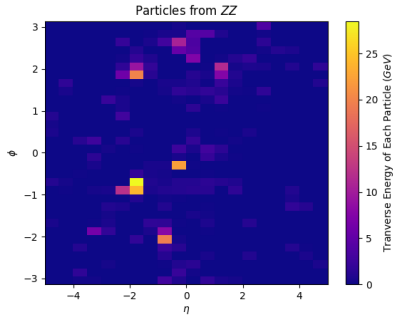
Our current theory of particle physics from the perspective of The Standard Model is well defined enough to code simulations which emulate the process and results. If there is a discrepancy between results from our simulations and results we measure experimentally, then we have an indication if our current model is not complete. Additionally, being able to simulate data allows us to train neural networks which automate the process of mapping observed jets to the parent particle they branched from. Training a neural network involves optimizing its ability to categorize a set of inputs to a particular output, by having it identify correlations from large data sets.

Table 1: Example of leading jet data per event.

Event	Leading Jets	Mass GeV	Eta η	Phi ϕ	Transverse Momentum GeV	Number of Constituents n	Effective radius R
0	0	64.9255349	1.06860603	0.560010329	44.2331188	44	0.293560737
	1	7.56656569	0.280419276	-2.41105307	330.800677	6	0.0457469783
	2	13.0619962	0.235500439	-2.84031589	139.332695	3	0.187493627
1	0	...	...	...	...	...	...
	1	...	...	...	...	...	...
	2	...	...	...	...	...	...

High-energy events in a particle collider are simulated with pythia - a python package which automates simulating such events. Particles are clustered into jets with pyjet, another python package. The Artificial Neural Network (ANN) being used determines what output is produced from pythia. Each ANN is distinct from the other by the data inputted. The two structures of data include a $2D$ weighted histogram (**Figure 3**) of how much energy is located with respect to η and ϕ, and a tuple containing the macroscopic properties of the three leading jets (**Table 1**) in an event. These structures are exported to a separate program *network.py* with binary values mapping the parent particles of the observed events decayed from, i.e. $0 \rightarrow$ zz, and $1 \rightarrow t\bar{t}$.

Figure 3: Example of 2D histogram for a single event.

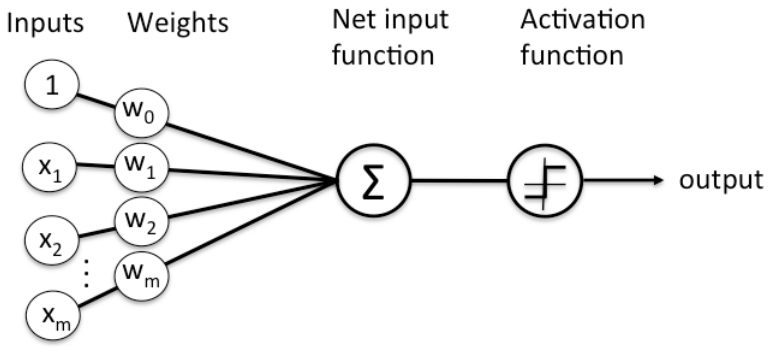


1.3 Automated Analysis of Data

The ATLAS detector at the LHC records 40 million proton collisions a second, making human curation of these events unfeasible. (Apostolatos & Bronner, n.d.). To effectively analyze the statistics within a 5-sigma confidence, machine learning algorithms must be utilized.

Machine learning is the specific branch of Artificial Intelligence that uses ANNs to generalize categorizations of arbitrary inputs. The most simple ANN has an input layer, with no intermediary layers, connecting to an output layer. A layer itself is a set of neurons. ANNs can be represented as mathematical graphs (not to be confused with Cartesian graphs), where the neurons are nodes containing some numerical value and the edges connecting nodes are the weight or effective impact of that value. A sample graph is shown in **Figure 4**.

Figure 4: Graph of input layer connecting to a single neuron output layer.



Each node in a layer connects to each node in the next layer. The value of the node in the next layer is the weighted sum of values from the previous layer. Let $\vec{n_p}$ be the vector of nodes in the previous layer, $\vec{w}$

be the vector of weights connected to a particular node in the next layer n_n, and b be the bias for a node in a layer. The value of the node is then

$$\vec{n_p} \cdot \vec{w} + b = n_n. \tag{1}$$

Each node has a value to feed forward or "wire" to each neuron in the next layer. Each neuron in the next layer has an activation function, which determines how much (if any) of that value is effective in causing that neuron to "fire". If there are multiple hidden layers this process repeats: neurons fire and wire together, causing further neurons down the graph to also fire and wire together. At the end of this process, the output neurons fire, indicating what value of output the network identifies to probabilistically correlate the most with the input. In this project, the input layer (in either network structure) is a set of scalar values, and the output layer is a single boolean number.

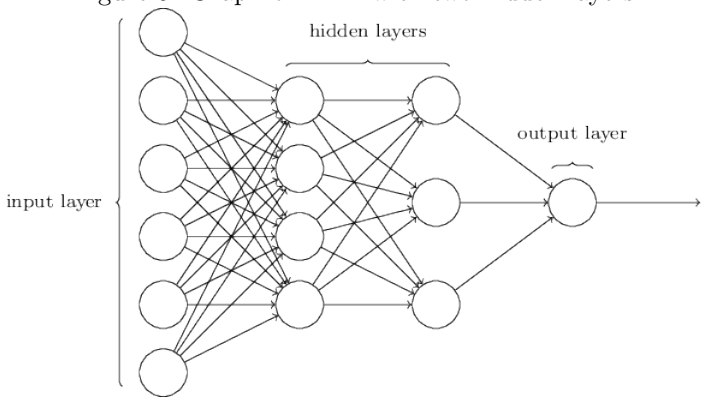
The weights and biases of the neural network are not predetermined. A single unique neural network has the values of the weights and biases for each neuron in every layer optimized to minimize the inaccuracy or loss of that network. The more neurons there are implies more parameters to optimize for. The deeper the depth of the network, the more combinations between neurons there are, which makes optimizing the network require a larger data set to be accurate.

A neural network with no hidden layers can accurately represent continuous linear functions if it has sufficiently large enough neurons. This amount is not fixed, as it is theoretical and depends on the function being analyzed. It is almost always more efficient in terms of time to train the neural network to use fewer neurons but with more depth. However, the trade-off requires a larger amount of computer memory to store all the combinations of weights and biases.

The neural network is able to pick out more complex features when more depth is added to the graph. A neural network with one hidden layer can implement boolean logic on the linear features, granting the ability to construct nonlinear outputs by piecing together linear objects. For example, assume a black and white image of a person's face is being analyzed with a neural network. The input is the brightness of each pixel in the image, so, for a 256×256 resolution image, there would be 65,536 neurons in the input layer, consisting of values from 0 to 100% brightness. A direct mapping to the output layer could pick out individual lines, such as the "slope" of that person's nose, but not the "curves" of their other features.

Adding hidden layers allows us to piece together outputs from the previous layer (Petersen & Voigtlaender, 2018). This more complex graph (in comparison to the graph in **Figure 4**) is displayed in **Figure 5**. In this example, a direct mapping to the output layer can find multiple straight lines, which the hidden layer can then use as input to identify the curves of a person's lips. In short, adding hidden layers allows us to identify more complex features. The trade-off, however, is a neural network may be so extremely capable at determining features that it is a specialist at only identifying elements in the training set correctly. In other words, it can perfectly memorize all the right answers it learns from, but is so highly specialized in this that it fails to generalize data it has not encountered before outside of the training set. This is known as overfitting.

Figure 5: Graph of ANN with two hidden layers.



1.4 Structure of This Project's Networks

Both neural networks in this project have final output layers corresponding to whether or not the parent particles were a top and anti-top quark $t\bar{t}$ or zz boson pair. The input layer for one neural network is the tensor of macroscopic properties for each jet in the event as in **Table 1**, and the input layer for the other is the tensor containing the weighted energy value in each spatial coordinate η, ϕ, as in **Figure 3**.

The neural network studying macroscopic properties of the jets has 6 properties per significant jet, and 3 significant jets arbitrarily chosen per event. Therefore, there are $6 \times 3 = 18$ values per event in our input layer mapping to either $t\bar{t}$ or zz. There are 3 hidden layers, each containing 18 nodes, and an output layer of 1 node.

The neural network analyzing the $2D$ energy weighted histograms has 64 nodes in the input layer, 25

nodes in the hidden layer, and again 1 node in the output layer.

Each network uses Rectified Linear Unit (ReLU) as the activation function in the hidden layers. ReLu is useful because it does not allow negative values to be fed forward which mimics biological neural networks. Additionally, it allows for optimizations in deeper networks to be computationally feasible without encountering overfitting. The activation function connecting to the output layer is the sigmoid function(Lin, Freytsis, Moult, & Nachman, 2018). It is common to see sigmoid used in the output layer because it treats the sum of all connections to the output layer equal to one, similar to a probability network maintaining the sum of all probabilities in the domain equal to one(Sutskever & Hinton, 2008). The loss function used by the neural network is binary cross entropy, since we're categorizing data into one of two groups: $t\bar{t}$ or *not* $t\bar{t}$(Komiske et al., 2017).

Simulated data is either streamed into the training program or sent out in batches. 1000 events with mappings to $t\bar{t}$ and zz are contained in each data stream or batch. The neural network optimizes its set of weights and biases via gradient descent to minimize its error on the training set. The effectiveness of the network is evaluated with the testing set, and the accuracy of the network is reported against a final validation set outside of the domain of the other two sets to prevent bias.

1.5 The Software Stack

A software stack is a set of programs working together to automate some logical process, with no further dependencies than what is included on the stack. Having the tools pre-packaged together will enable future generations of computational particle physicists to efficiently begin their research in a more familiar python environment.

1.5.1 A Pythonic Environment for Particle Physicists

The current standard working environment for particle physics is done in ROOT, or in a collection of various python packages. ROOT is a data analysis framework for high-energy physics (HEP). The advantages to running ROOT is its docker image, allowing anyone with docker installed to easily get started in the research. A docker image is a virtual environment that can be loaded and emulated inside of Docker,

allowing for program files and dependencies to be exported easily.

The benefit of working in python is using an easier language in terms of user-friendliness and less code overhead for tasks such as dynamic data management. The trade-off, however, is it requires the user to manually configure their environment. The files included in my research are added to a docker image containing all the dependencies required to run the code: numpythia (numpy + pythia), matplotlib, tensorflow, and pyjet. The files themselves are an example of python files that include everything from basic file input/output, to basics of simulating events with pythia, up to implementing a neural network via tensorflow.

1.5.2 Example Code Directory

There are seven example programs included in the stack, each of which progressively builds from the prior. The first example directory: `four_vec_hists`, contains code that receives a sample pythia output file as input, calculates the mass of each event, and creates a histogram of the invariant mass per event. The second directory: `pythia_prac`, is similar to the previous directory, but generates the data included in the input file itself. The third directory: `jet_prac` is similar to the previous ones, but includes methods and routines to cluster the particles into jets, and plot histograms on their macroscopic properties. `2d_hists` contains the code which makes $2D$ energy-weighted histograms, and `keras_implementation` is the first implementation to use a neural network to analyze the data.

The next two directories are the folders containing the two different neural network implementation. `data_train/choo_choo` is the directory containing the neural network which analyzes data in the form of $2D$ histograms, while `jet_data_train/choo_choo` is the network which analyzes macroscopic jet data. Each `../choo_choo` directory contains two python files - `data.py` and `network.py`. The `data.py` program in either directory runs until instructed to stop by the user, indefinitely generating the relevant data set for that directory. The `network.py` program runs until it identifies a dataset to learn from produced by `data.py`.

1.6 Overview

In Chapter 2 - Methods, the software stack control flow is defined in greater detail, as well as the structure of the neural networks. In Chapter 3 - Results, the histograms generated are provided and the accuracy of

the programs' automated analysis is reported. Lastly, Chapter 4 - Conclusion, describes shortcomings in my models and discuss what could have been improved upon to achieve a more accurate model.

References

Apostolatos, A., & Bronner, L. (n.d.). Identifying the Higgs Boson with Convolutional Neural Networks. , 9.

Buckley, A. (2007, August). The problem with ROOT (a.k.a. The ROOT of all Evil). Retrieved 2020-03-01, from http://insectnation.org/articles/problems-with-root.html

Chiochia, V. (n.d.). Accelerators and particle detectors. , 18.

de Oliveira, L., Kagan, M., Mackey, L., Nachman, B., & Schwartzman, A. (2016, July). Jet-Images – Deep Learning Edition. Journal of High Energy Physics, 2016(7), 69. Retrieved 2020-01-29, from http://arxiv.org/abs/1511.05190 (arXiv: 1511.05190) doi: 10.1007/JHEP07(2016)069

Komiske, P. T., Metodiev, E. M., & Schwartz, M. D. (2017, January). Deep learning in color: towards automated quark/gluon jet discrimination. Journal of High Energy Physics, 2017(1), 110. Retrieved 2020-01-29, from http://arxiv.org/abs/1612.01551 (arXiv: 1612.01551) doi: 10.1007/JHEP01(2017)110

LHCb looks forward to electroweak physics. (2012, March). Retrieved 2020-02-04, from https://cerncourier.com/a/lhcb-looks-forward-to-electroweak-physics/

Lin, J., Freytsis, M., Moult, I., & Nachman, B. (2018, October). Boosting $H\to b\bar b$ with Machine Learning. Journal of High Energy Physics, 2018(10), 101. Retrieved 2020-01-29, from http://arxiv.org/abs/1807.10768 (arXiv: 1807.10768) doi: 10.1007/JHEP10(2018)101

Petersen, P., & Voigtlaender, F. (2018, May). Optimal approximation of piecewise smooth functions using deep ReLU neural networks. arXiv:1709.05289 [cs, math, stat]. Retrieved 2020-02-23, from http://arxiv.org/abs/1709.05289 (arXiv: 1709.05289)

Sutskever, I., & Hinton, G. E. (2008, November). Deep, Narrow Sigmoid Belief Networks Are Universal Approximators. Neural Computation, 20(11), 2629–2636. Retrieved

2020-02-23, from http://www.mitpressjournals.org/doi/10.1162/neco.2008.12-07-661 doi: 10.1162/neco.2008.12-07-661