Simulating Collision Events using Neural Networks

A Thesis

submitted to
Indian Institute of Science Education and Research Pune
in partial fulfillment of the requirements for the
BS-MS Dual Degree Programme

by

Aparna Jayaraj

Indian Institute of Science Education and Research Pune
Dr. Homi Bhabha Road,
Pashan, Pune 411008, INDIA.

April, 2023

Supervisor: Dr. Sourabh Dube

Certificate

This is to certify that this dissertation entitled Simulating Collision Events using Neural Networks towards the partial fulfilment of the BS-MS dual degree programme at the Indian Institute of Science Education and Research, Pune represents study/work carried out by Aparna Jayaraj at Indian Institute of Science Education and Research under the supervision of Dr. Sourabh Dube, Associate Professor, Department of Physics during the academic year 2022-2023.

Dr. Sourabh Dube

Committee:

Dr. Sourabh Dube

Dr. Arka Banerjee

This thesis is dedicated to my parents

Declaration

I hereby declare that the matter embodied in the report entitled Simulating Collision Events using Neural Networks are the results of the work carried out by me at the Department of Physics, Indian Institute of Science Education and Research, Pune, under the supervision of Dr. Sourabh Dube and the same has not been submitted elsewhere for any other degree.

Aparna Jayaraj

Acknowledgments

Firstly, I would like to express my sincere gratitude to Sourabh for his exceptional mentorship. Despite the challenging circumstances brought on by the COVID-19 pandemic, Sourabh believed in me and offered me a project that allowed me to pursue my research interests. He has been a constant source of encouragement and the best person to learn from. Sourabh always ensured that I had enough guidance for everything, understood my needs, and offered help without even being asked. I am particularly grateful for his willingness to patiently walk me through every unfamiliar concept. Sourabh always allowed me to work at my own pace, made me realise the importance of work-life balance and provided valuable feedbacks that helped me grow as a researcher.

I would like to acknowledge the crucial role played by Arnab in the success of this research project. His insightful inputs and feedback have been critical in shaping the research direction and achieving the desired results. He has always helped me understand the concepts better and was constantly available to answer my queries. I feel incredibly lucky to have worked with Angira, who has been a constant source of inspiration for me. Prachu has always been there to lend a helping hand whenever I needed it, and I am genuinely grateful for that. I would like to convey my immense gratitude to Chitrakshee, who became my close friend in a very short period of time. She always had my back whenever I felt low or needed support. Her skills to listen without prejudice have helped me get through some of the most challenging moments of my academic journey. I would also like to thank other lab members - Yash, Riya, Parijat and Soumya for their delightful company and all the healthy discussions. I feel fortunate to have worked with the group and learned from such amazing people.

I would like to express my deepest gratitude to my dad, who has been my best friend and mentor. He has always encouraged me to pursue my passions and has been my biggest supporter in every decision I have made. His unwavering belief in my abilities has helped me overcome all the chal-

lenges that came my way. I would also like to thank my mom, who has always emphasized the importance of education and instilled in me a love for learning. She has been my pillar of strength, and I owe my academic success to her. Furthermore, I would like to express my heartfelt gratitude to my brother Sarath, who has always been there to lend an ear, offer advice, and provide help whenever needed. I would also like to thank my uncle and aunt, who have been like parents to me and my brother Sajith, for their constant encouragement and presence by my side during both the good and the challenging times.

I owe a great debt of gratitude to my best friends Yadhu, Kiran, Nishanga and Sreedev from the bottom of my heart for their invaluable support throughout my Master's thesis journey. They were always there to support me in the toughest times, offering words of advice and comfort. I would like to specially mention my roommate Nila and my whole friend circle at IISER for their company, considerate actions and refreshing conversations that helped me recharge and concentrate. Last but not the least, I would like to express my appreciation for my fellow football playmates, for providing me with a much-needed break and positive environment.

Abstract

In this work, we study the use of machine learning techniques to generate simulation of pp collisions at the CMS experiment at the LHC. Standard model processes such as $W+$jets form a background for beyond standard model searches such as the search for vector-like leptons in the one lepton and two jets final state. Full GEANT4 based simulation of these backgrounds is time and compute intensive. However, if one focuses on only the quantities of interest, then the task can be done in a simpler way by using modern tools such as Generative Adversarial Networks (GANs) or variational autoencoders (VAEs). In this work, we implement a basic GAN and a basic VAE to produce event distributions for processes such as Drell-Yan and $W+$jets. We test the dependence of the GAN and the VAE on the hyperparameters of the corresponding network. We demonstrate the efficacy of using the generated distributions by training a binary classifier to distinguish the $W+$jets process from semileptonic $t\bar{t}$ production. We find that an equivalent performance to GEANT4 based simulation can be obtained by instead using the VAE generated output. This shows us that the usage of these algorithms can be used to speed up the generation of simulations for the LHC experiments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Machine Learning in High Energy Physics

Machine learning has become an essential tool in various aspects of particle physics research over the past few years [1]. The ability of machine learning algorithms to efficiently process and analyze massive amounts of data has led to numerous applications in high energy Physics, including particle identification, event reconstruction, anomaly detection, classification of signal and background events, and simulation and modelling [2]. Different neural network architectures, such as deep neural networks, convolutional neural networks, or graph neural networks, have found different applications in experiments such as CMS (Compact Muon Solenoid) at the Large Hadron Collider (LHC). Our objective is to study the use of generative adversarial networks and variational autoencoders to simulate collision events.

1.2 Background and Rationale

Previous analyses from our group performed a search for inclusive nonresonant signatures of beyond the standard model (BSM) physics on 138 fb^{-1} Run II data collected by the CMS experiment at the LHC [3]. The search was performed in multilepton final states, i.e. states with three or more charged leptons, including hadronically decaying τ leptons. Models such as vector-like leptons in their doublet and singlet scenario, type III seesaw, and scalar leptoquarks were probed, but no significant deviations from the background expectations were observed. Ongoing studies focus on the singlet model of vector-like leptons in a final state with one isolated lepton and two jets. In this

search, the primary standard model background is W+jets production. The W production has an extremely high cross-section at the LHC ($\gtrsim$ 60000 pb at $\sqrt{s} = 13\,TeV$) [4, 5]. This will increase even more after the HL-LHC upgrade [6]. Thus, predicting data with low statistical uncertainty requires simulating a large number of events [7]. The relation between uncertainty and number of events is explained in further detail below.
The number of events predicted in Monte Carlo is given by

$$N = L\sigma\epsilon \tag{1.1}$$

Here, L integrated luminosity, is the amount of data. σ is the production cross-section. It is the measure of the probability of a process taking place. It has units of area. The smaller the cross section of a process, the more rare it is to take place in a collision. ϵ is the efficiency and is defined as,

$$\epsilon = \frac{Number\ of\ events\ passing\ all\ selections}{Total\ number\ of\ events}$$

Thus, the predicted number of events N has a statistical uncertainty that depends on total number of generated events as well as number of events passing selection. Typically, to reduce background, modern analyses employ multivariate algorithms such as neural networks. These are basically designed to be binary classifiers for signal against background. The performance of these algorithms relies on large number of events for effective training. Additionally, signal optimization selection results in low acceptance for W+jets events; this hampers good training for ML algorithms meant to reduce W+jets. Thus, we need to produce more and more events as the amount of data produced at the LHC increases. But, producing large simulation samples using GEANT4 or other toolkits is computationally challenging. GEANT4 (for GEometry ANd Tracking) is a Monte Carlo- based simulation platform that simulates the interaction between elementary particles and matter [8]. It keeps track of particle trajectories and energy deposits in each cluster of the detector to help particle reconstruction. Thus, it takes a lot time to simulate millions of events [9].
In this scenario, simulating collision events using neural networks helps in reducing computational power as well as computation time. Neural networks such as Generative Adversarial Networks (GANs) or Variational Autoencoders (VAEs) require much less computational time as they generate certain numbers that follow a known distribution [10].
An additional benefit is that in certain situations, only some high-level variables are required instead of a complete event description. To address this, we intend to set up a GAN/VAE to generate only those relevant quantities of interest in W+jets production. GANs and VAEs are generative

models used in machine learning for generating new data. GANs use two neural networks, a generator and a discriminator, to learn the underlying distribution of the training data and generate new samples. VAEs use a probabilistic approach to learn the latent space representation of the data and generate new data by sampling from this space. Further details are included in the respective chapters. So far, we are able to generate five quantities of interest.

1.3 The research problem

Studies conducted on multilepton final states for beyond the Standard Model phenomena by the EHEP group has found no significant excess in the data. This paved the way to the ongoing search for BSM Physics in the singlet model of vector like leptons in final states with one isolated lepton and atleast two jets. In this search, the primary standard model background is $W+$jets events. Their respective Feynman diagrams are given below (Figure 1.1).

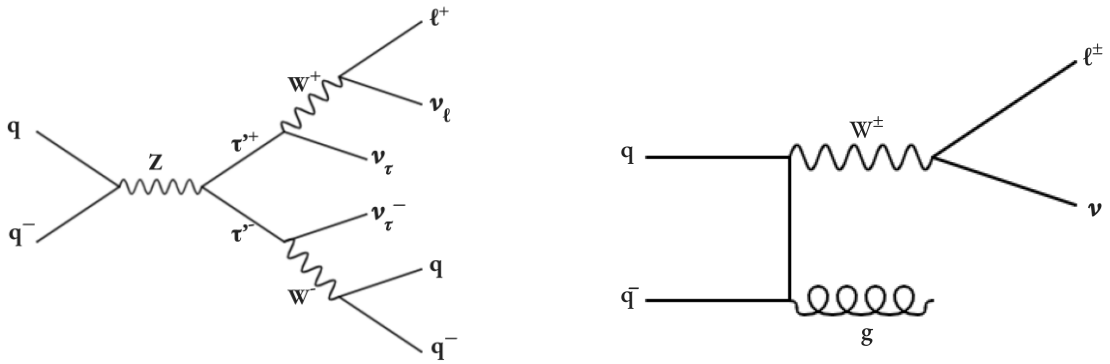


Figure 1.1: (a) Feynman diagram of VLL singlet model in their $1\ell 2$j final state and (b) Feynman diagram of $W+$jets process

Neural network classifiers can be used to distinguish signal events from background. For this, the neural network is trained on distributions of certain variables that could help distinctly identify signal from background. But, the choice of these variables are context-specific. Once trained, this model classifies similar events. In some examples, to distinguish between these events using a neural network classifier, we would only need a few high level variables that exhibit substantial differences across these events. Here, we are designing a neural network classifier that can

distinguish W+jets process from semileptonic $t\bar{t}$ production. In the case of W+jets events, such variables are dijet mass, $H_T, \Delta R$, transverse mass of lepton, Missing E_T ($E_{\mathrm{T}}^{\mathrm{miss}}$) etc. We chose to produce distributions of transverse mass of lepton, dijet mass, transverse mass of the dijet system, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T. Transverse mass of the lepton (M_T^{ℓ}) is a relevant quantity because it is invariant under Lorentz boosts along the direction of the colliding particles. The transverse mass for a single lepton is defined as,

$$M_T^{\ell} = \sqrt{2p_T^{\ell} E_{\mathrm{T}}^{\mathrm{miss}}(1 - \cos(\Delta\phi))} \tag{1.2}$$

where E_T is the transverse energy of the lepton, $E_{\mathrm{T}}^{\mathrm{miss}}$ is the missing transverse energy in the event and $\Delta\phi$ is the angle between the lepton's transverse momentum and the missing transverse energy vector. Missing transverse energy ($E_{\mathrm{T}}^{\mathrm{miss}}$) represents the momentum carried away by any undetected particles. Dijet mass refers to the invariant mass of a pair of jets produced in a particle collision. Transverse mass of the dijet system (M_T^{jj}) is the effective mass of the two jets in the transverse direction, which is perpendicular to the direction of the colliding particles. It is calculated by using p_T^{jj} instead of p_T^{ℓ} in equation. Here, p_T^{jj} is the resultant of the transverse momenta of the two jets (p_T^1 and p_T^2), and $\Delta\phi$ is the angle between the resultant transverse momentum of the two jets and $E_{\mathrm{T}}^{\mathrm{miss}}$. H_T is defined as the scalar p_T sum of all jets. The NN score output plots and ROC curve are shown in figure 1.2.

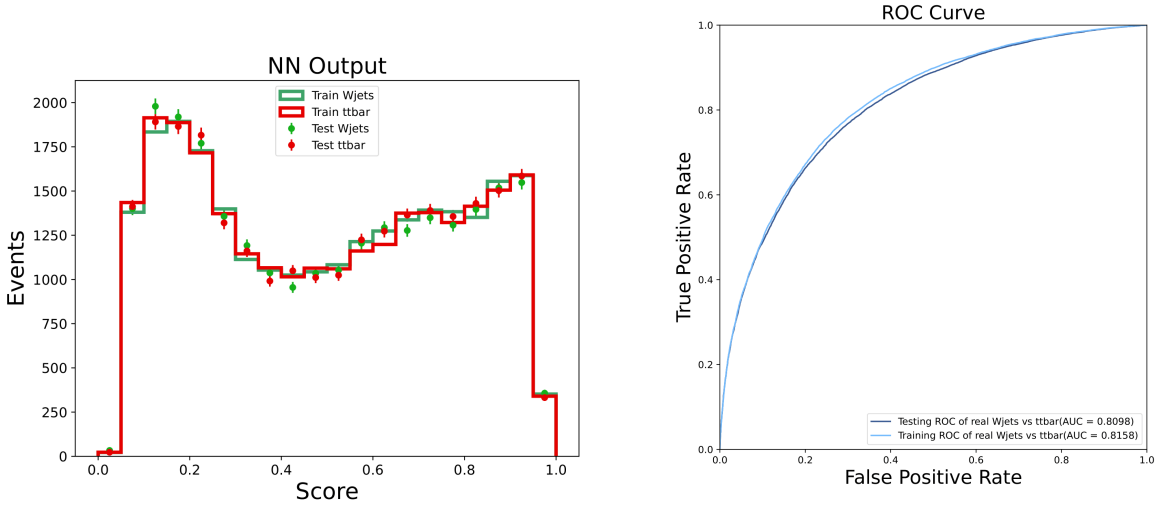


Figure 1.2: NN score output plot and ROC curve after training W+jets- classifier

1.4 Approach and Methodology

We gained a basic understanding of the underlying principles of deep learning neural networks. The fundamentals behind various neural networks and various components of deep learning neural networks are discussed in chapter 2. We started by exploring the capabilities of Generative Adversarial Networks (GANs). We produced plots of some basic functions and gradually progressed to generating real distributions of high energy physics processes. However, GANs failed to provide the results with the desired accuracy. Then, we focused on Variational Autoencoders (VAEs) and tried to perform similar tasks. The preliminary results of GANs and VAEs are provided in chapters 3 and 4, respectively. Our findings indicated that Variational Autoencoders performed more effectively than Generative Adversarial Networks in our specific context. The detailed descriptions and better results obtained through the simulation of $W+$jets events are discussed in chapter 5.

In this thesis, the whole work was performed using a computer with the following specifications; Intel® Core^{TM} i3-6006U 2.0GHz CPU , 3MB L3 Cache and 4 GB RAM.

Chapter 2

A Comprehensive Guide to Neural Networks

A neural network is a set of machine learning algorithms that imitate the functioning of biological neurons in the human brain [11]. A neural network has an input layer, one or more hidden layers, and an output layer. A basic neural network illustration is given in figure 2.1. The neural network architecture is often considered important for its optimal performance. These networks use their nodes to identify patterns and underlying relations in data. Each node, or artificial neuron, is interconnected to others; they process and transmit information. Each node has a weight associated with it. To train a neural network, a sufficiently large set of input data is provided. During training, the network adjusts its weights using backpropagation method to minimize the differences between the predicted and actual output. Once the training process is complete, the generated model can be used to test similar data.

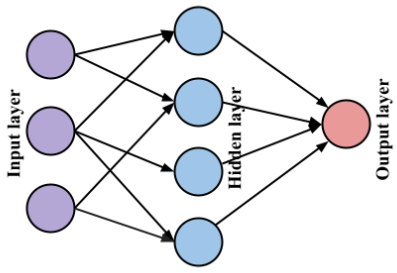


Figure 2.1: Illustration of a basic neural network.

Neural networks can do classification as well as regression tasks. They can be applied to various purposes, including prediction, image identification, and natural language processing. Multiple types of neural networks exist, including RNNs (Recurrent Neural Networks), CNNs (Convolutional Neural Networks), and DNNs (Deep Learning Neural Networks). Deep Neural Networks (DNNs) are a type of artificial neural network (ANN), developed to learn and model complex nonlinear relationships in input data. It consists of several layers of interconnected neurons, with the number of layers varying from a few dozens to hundreds. DNNs are implemented in many different applications, including self-driving cars, voice recognition, language translation, natural language processing, and visual recognition. Convolutional neural networks (CNNs) are mostly used for image and video processing applications. They are able to recognise, and learn features and patterns in the input data. CNNs are extensively used in fields including autonomous driving, facial recognition, and medical imaging. Recurrent Neural Networks (RNNs) are built to process sequential data, with each subsequent step's output being fed back into the network as input for the next step. They are frequently implemented for language processing tasks like speech recognition, document analysis and natural language processing.

2.1 Training and Validation

Neural network training is the process of adjusting the parameters of a neural network to minimize the difference between the predicted output and the true output. During training, the network is provided with an input data set. The cost function (also known as the loss function), is defined as a combination of weights of the network. Cost function indicates how well the model performs in terms of its ability to predict the output values given the input data. During training, a neural network tries to minimize the cost function, i.e. finding the set of weights and biases that results in the best predictions on the training data. The lower the cost, the better the prediction. The weights of the network are updated through an iterative process called backpropagation. In backpropagation, the error is propagated backwards through the network, and the weights are updated accordingly. The weights are then adjusted using an optimization algorithm, such as stochastic gradient descent (SGD). In SGD, the gradient of the cost function with respect to weights is calculated, identifying the direction in which the loss function is reducing. This process is known as gradient descent, as shown in Figure 2.2. We change the weights by some amount (learning rate), recalculate the cost, and iterate until we are satisfied with the weights. The learning rate is a hyperparameter that determines the step size at which the model's weights are updated during training. If the learning rate is too high, the optimization algorithm may surpass the optimal weights, causing the model to

diverge and perform poorly. Whereas, if the learning rate is too low, the optimization algorithm may take a long time to converge to the optimal weights, resulting in slow training and potentially getting stuck in local minima.

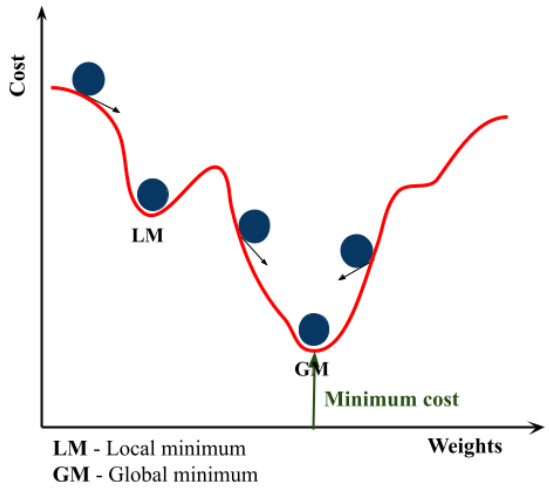


Figure 2.2: An illustration of Gradient descent method

The training process is repeated for multiple epochs, where each epoch is a complete pass through the training data. After some number of epochs (equal to the batch size), the performance of the network is evaluated on a separate validation set to prevent overfitting. If the validation error is not improving, the training is stopped, and the final weights are used for prediction on the test set.
Testing a trained neural network model involves evaluating its performance on a statistically independent dataset from the training data to avoid any potential biases or overfitting. The motivation behind testing is to evaluate how well the model generates predictions and generalizes to new data. The predicted outputs are then compared to the true outputs to calculate the model's performance metrics, such as accuracy, precision or neural network scores. The performance of the network is determined by plotting the neural network scores and ROC curve plots.

2.1.1 ROC curve

The ROC curve is a tool for evaluating the performance of classification models implemented using neural networks. It plots the relationship between TPR (True Positive Rate) and FPR (False Positive Rate), with threshold values varying along the x-axis. TPR measures the ratio of true pos-

itive predictions to the total number of positive examples, while FPR measures the ratio of false positive predictions to the total number of negative examples. A good classifier has a high TPR and a low FPR, corresponding to a point in the top-left corner of the ROC curve. AUC (Area Under the Curve) is also a useful metric for assessing neural network performance. It represents the area under the ROC curve. For a good classifier, the AUC value will be close to 1.

2.2 Components of deep learning models

2.2.1 Activation functions

The activation function decides whether or not to activate a neuron, based on the input it receives. The neural network's activation function enables it to learn and model complex relationships in the input data [12]. Commonly used activation functions and their respective plots are given below (Figure 2.3).

- **Linear activation function**
  The linear activation function is defined as $f(x) = x$; it simply returns the input value x as the output value. It is often used in the output layer of regression models, where the goal is to predict a continuous output variable. In linear activation function, output of the functions will not be confined between any range.

- **Sigmoid activation function**
  The sigmoid activation function maps the input values to the range (0, 1). It is defined as $f(x) = \frac{1}{1+exp(-x)}$. This function is commonly used in binary classification problems where the probability is predicted as an output.

- **Softmax activation function**
  The softmax function takes a vector of real numbers as input and transforms them into a probability distribution. It maps the input values into the range of [0,1], ensuring that the sum of the output values is equal to 1. The function is defined as $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{m} e^{x_j}}$ Here, x_i represents the i^{th} element of the input vector, and m is the number of elements in the vector. It is commonly used in multi-class classification problems, where the output of the neural network needs to be a probability distribution over a set of classes.

- **ReLU activation function**

  ReLU activation function performs well in convolutional neural networks or deep learning. It is a non-linear function that maps the input values to the range $(0, \infty)$. The ReLU (Rectified Linear Unit) function is defined as $f(x) = max(0, x)$, i.e. it returns the input values if it is positive and zero otherwise. Therefore, this function is piecewise linear with a slope of one for positive inputs and a slope of zero for negative inputs. Setting the negative values to zero ensures that the gradients in a deep neural network do not become too small as they propagate through the network, allowing for faster and more stable training.

- **Leaky ReLU and parametric ReLU activation functions**

  Leaky ReLU and parametric ReLU are similar to ReLU but allow a small negative slope for negative input values. They map the input values to the range (∞, ∞). Leaky ReLU is defined as $f(x) = max(\alpha x, x)$. Whereas, parametric ReLU is defined as $f(x) = max(0, x) + \alpha min(0, x)$. In leaky ReLU, the value of alpha is fixed, and usually, it is 0.01. In parametric ReLu, the value of alpha is learned along with the other network parameters through back-propagation during training. Both these functions help recover the dying ReLU problem, in which the negative inputs result in a zero output causing the neurons to be in a deactivated state.

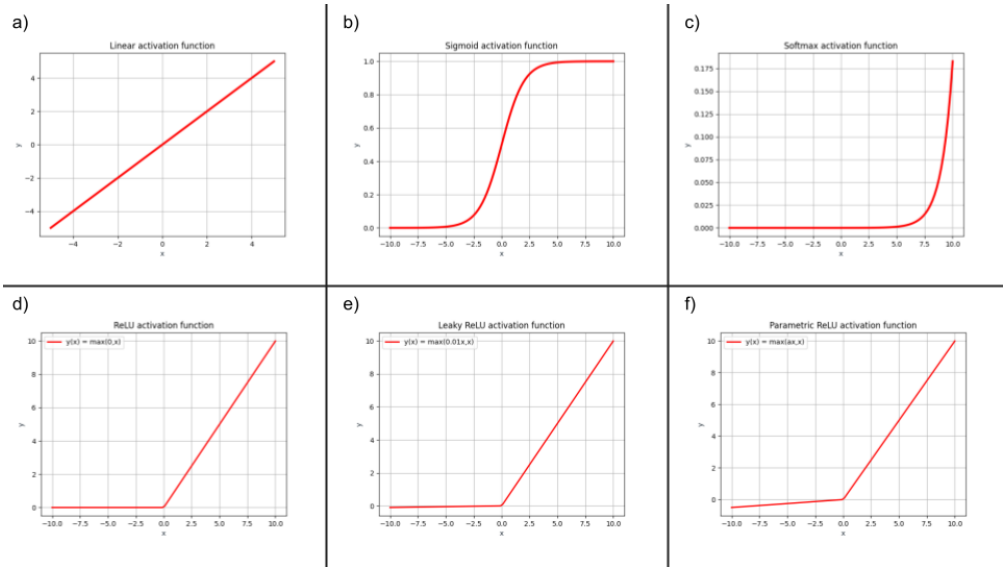


Figure 2.3: The figure shows the plots of (a) linear activation function, (b) sigmoid activation function, (c) softmax activation function, (d) ReLU activation function, (e) leaky ReLU activation function and (f) parametric ReLU function respectively.

2.2.2 Loss functions

There are various types of loss functions used in neural networks. The most commonly used loss functions are described below [13].

- **Mean Square Error**
  Mean Squared Error (MSE) is commonly used in regression problems. It measures the averaged square of the difference between the predicted and the true values of a dataset. It can be calculated using the formula,

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(Y_i - \hat{Y}_i)^2$$

  where, Y_i is the true value and $\hat{Y}_i$ is the predicted value. The lower the MSE value, the closer the predicted values are to the actual values, indicating the better neural network performance. It penalizes large errors more heavily than small errors due to the squaring of the differences. It is also differentiable, making it suitable for use with gradient-based optimization algorithms such as stochastic gradient descent (SGD) or backpropagation.

- **Binary Cross-entropy**
  Binary Cross-entropy (BCE) is widely used in binary classification tasks. It measures the difference between the predicted probabilities and the actual binary labels for a given dataset. BCE is defined as the negative mean of the log of corrected predicted probabilities for each instance in the dataset. It can be calculated as

$$BCE = -y\log(p) - (1-y)\log(1-p)$$

  The log value offers less penalty for small differences between predicted and corrected probability; when the difference is large, the penalty will be higher. Being differentiable, it allows for easy optimization using gradient descent-based algorithms such as stochastic gradient descent (SGD) or backpropagation.

2.2.3 Optimizers

Optimizers are functions or algorithms used to adjust the weights and biases of the neural network to improve accuracy and minimize the error or loss function. It allows to find the optimal set of weights and biases that will minimize the error on the training data and allows the network to gen-

eralize on testing data. There are several optimization algorithms used in neural networks. Some of them are Stochastic Gradient Descent (SGD), Adam (Adaptive Moment Estimation), AdaGrad and AdaDelta.

Adam (Adaptive moment estimation) is a popular and powerful optimization algorithm used in machine learning and deep learning. It is an extension of the stochastic gradient descent (SGD) optimization method, designed to provide faster convergence and better performance. It uses a combination of two gradient descent methods, a momentum algorithm, and an RMSP algorithm. Adam optimizes the objective function by computing individual adaptive learning rates for each parameter. The learning rate is computed using estimations of the first and second moments of the gradient, and the weights of the neural network are updated using this learning rate. It has several advantages over traditional optimization algorithms such as SGD. Adam provides faster convergence due to adaptive learning rates. It is less sensitive to the choice of hyperparameters, can handle noisy and sparse gradients, and can be easily parallelized.

2.2.4 Hyper parameters

- **Number of epochs**

  The number of epochs refers to the number of times the entire training dataset is passed through the model during training. An epoch represents a complete iteration through the entire dataset. Typically, it can range from a few hundred to several thousand, depending on the complexity of the problem, the size of the dataset and the computational power of available resources. However, it is essential to monitor the training process and stop it when the models reach a satisfactory level of performance, as continuing to train for too long can lead to overfitting or other issues. Monitoring the loss and accuracy metrics, visualizing the generated samples, and evaluating the model's performance on a validation set helps to determine the optimal value of the number of epochs for a particular GAN model.

- **Batchsize**

  Batch size is the number of samples or events processed in each iteration during the training process. The choice of batch size is crucial in neural networks as it can affect the convergence speed, memory usage and the model performance. A larger batch size can lead to more stable training and faster convergence, as more samples are processed in each iteration. But it may require more memory and computational resources, and may lead to overfitting if the

model is too complex. Whereas, a smaller batch size can help avoid overfitting, as the model updates its parameters more frequently based on smaller batches of data. But this may also result in slower convergence, as it doesn't represent the overall data distribution. The choice of optimal batch size varies depending on the dataset, model and the computational resources available. During training, different batch sizes are experimented usually and the one with the best trade-off between convergence speed and model performance is often selected.

Chapter 3

Generative Adversarial Networks (GANs)

3.1 Introduction to GANs

Generative Adversarial Networks (GANs) are a type of artificial neural network based on deep learning algorithm [14]. GANs can generate new data with the same statistics as the training set provided [15]. It was designed by Ian Goodfellow and his colleagues in June 2014. A generative adversarial network (GAN) has two parts: a generator and a discriminator. The generator generates data points in the latent space, and the discriminator learns to distinguish the generated data from real data. The two models get trained adversarially. During training, the generator generates a batch of fake data, and the discriminator receives a batch of real data from the training dataset along with a batch of fake data generated by the generator. The discriminator evaluates these batches of data and provides feedback to the generator on improving the quality of the generated data. By playing a minimax game between the generator and the discriminator, the GAN learns to generate realistic outputs that closely resemble the real data. Working of a GAN is represented in the flowchart given below(Figure 3.1) .

GANs have a wide range of applications, including generating realistic images, and videos as well as being used for data augmentation and anomaly detection. One of the strengths of GANs is their ability to learn complex and high-dimensional data distributions, which makes them suitable for generating realistic data. GANs can be trained on any combination of simulated and actual data. Training GANs can be challenging, as it requires careful tuning of hyperparameters and the choice

of loss functions used during training. But, they are extremely fast once trained. GANs have also been used for unsupervised learning, where they can generate new data without the need for labelled training data.

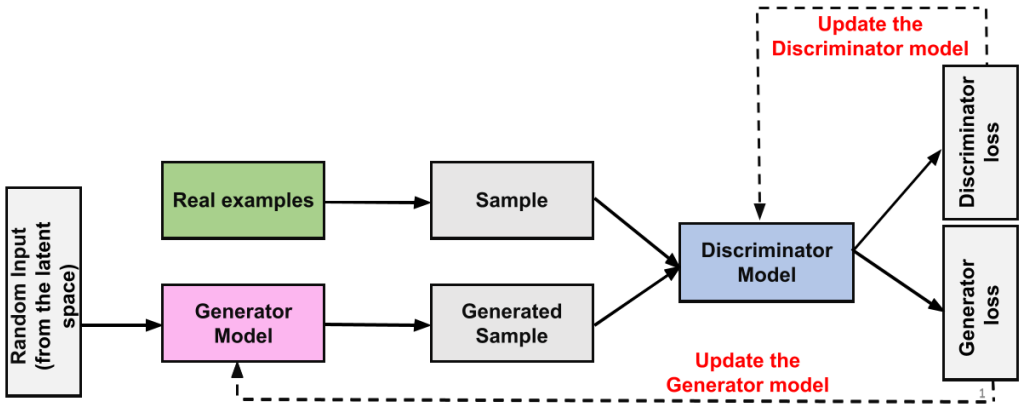


Figure 3.1: Illustration of a Generative Adversarial Network (GAN)

GAN trainings

The training procedure for the generator is to maximise the probability of the discriminator making a mistake, enabling the model to learn in an unsupervised manner. It learns to make the discriminator classify its output as real. GAN has two losses associated with it; generator loss and discriminator loss. The discriminator penalises (generator loss gets applied) the generator for producing implausible results, and the generator updates its model parameters. The discriminator uses real data instances as positive examples and fake data instances as negative examples during training [16]. When the discriminator misclassifies a real instance as fake or a fake instance as real, discriminator loss penalises the discriminator and updates its weights. As the generator improves with training, the discriminator performance gets worse. At a limit, the generator generates perfect replicas every time, and the discriminator cannot tell the difference and identifies the generated samples as real. The figure 3.2 shows different stages of an example case of GAN training.

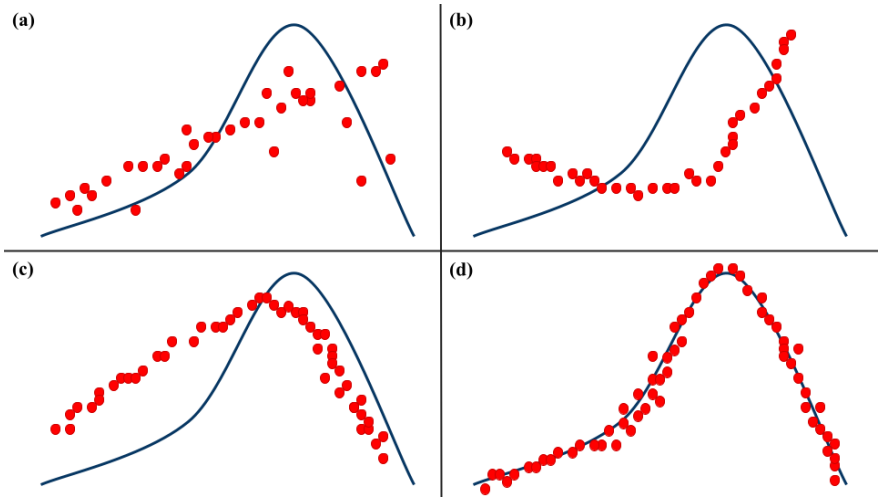


Figure 3.2: The figure shows the training of GAN. (a) When the training begins, the generator produces fake data, and the discriminator quickly identifies it's fake. (b), (c) As training advances, the generator gets closer to producing output that can fool the discriminator. (d) Finally, if the generator training goes well, the discriminator cannot differentiate between real and fake. It starts to classify fake data as real.

Implementation

GANs were implemented using tensorflow algorithms in python environment [17]. TensorFlow is an open-source software library for numerical computation and machine learning [18]. It was developed by the Google Brain team and is widely used in research and industry for building and training machine learning models.

Activation functions

Generator and discriminator are both neural networks with an input layer, an output layer and several hidden layers of neurons. In the generator, all the inner layer neurons are activated by ReLU/ Leaky ReLU activation functions, whereas linear activation functions are used for the output layer. The discriminator has all the inner layer neurons activated by ReLU/ Leaky ReLU activation functions and the last hidden layer activated by the sigmoid activation function.

Loss functions

The standard GAN framework learns a data distribution as a minimax game between the generator (G) and the discriminator (D) [19]. Both the generator and the discriminator are multilayer perceptrons [20]. The discriminator ($D(x)$) takes input 'x' which is a combined set of input data and generated data. The generator ($G(z)$) takes input 'z' drawn from a random distribution. The GAN loss function is given by,

$$min_G \, max_D \, V(D, G) = E_x[\log(D(x))] + E_z[\log(1 - D(G(z)))]$$

During training, the generator tries to minimise $\log(1 - D(G(z)))$, whereas the discriminator tries to maximise it [21].

Generator loss is applied to the generator when the discriminator identifies the generated outcomes as fake. Generator loss is given by,

$$Loss_G = \nabla_{\theta_g} 1/m \sum_{i=1}^{m} \log(1 - D(G(z^{(i)})))$$

Discriminator loss penalises the discriminator when it misclassifies a real instance as generated or a generated instance as real. It is calculated as,

$$Loss_D = \nabla_{\theta_d} 1/m \sum_{i=1}^{m} [\log D(x^{(i)}) + \log(1 - D(G(z^{(i)})))]$$

The model employs the binary cross-entropy (BCE) loss function as the discriminator loss function and mean square error (MSE) as the generator loss function. GAN model was executed using Adam (Adaptive Moment Estimation) optimizer. It uses estimations of the first and second moments of the gradient to update the weight of the neural network.

Hyper parameters

- **Number of latent dimensions**
  The latent space refers to a finite dimensional space from which random vectors are drawn that is used as input to the generator network. The selection of the number of latent dimen-

sions is by the user's freedom. By changing the value of this parameter, the generator can create different outputs that correspond to different points in the latent space.

3.2 Producing simple functions using GANs

Initially, basic geometric shapes like circles and ellipses, and some simple polynomial functions were generated and examined for their reasonable match. Subsequently, the ability of Generative Adversarial Networks (GANs) to produce periodic functions was tested by plotting $sin(x)/x$. Following the training process, the GANs were found to yield satisfactory outcomes. To further improve our proficiency in utilizing Generative Adversarial Networks (GANs), we generated standard distributions such as Gaussian and various stochastic distributions. This involved creating text files that contained 10,000 data points for each function, which were then used as the truth functions for the GAN. The minimum training time required was approximately 20 minutes, with an increase up to 70-80 minutes based on the number of epochs chosen. The models were trained by varying the number of epochs, network architecture, batch size and the number of latent dimensions. However, some of the observed results were unsatisfactory. The most promising outputs and their corresponding evolution plots are tabulated below (Table 3.1).

Function	Generator	Discriminator	NEpochs	Output plots
$x^2 + y^2 = 1$	32, 16, 8	16, 8, 1	12000	Figure 3.3
$y = sin(x)/x$	256, 128, 128, 64, 8	128, 64, 32, 1	14000	Figure 3.4
$e^{-(x-5)^2/4}$	256, 128, 128, 64, 8	128, 64, 32, 1	20000	Figure 3.5
$y = x^4 - 5x^2 + cos(3x)$	256, 128, 128, 64, 8	128, 64, 32, 1	20000	Figure 3.6

Table 3.1: Training information for various functions. (NEpochs is the number of epochs during training, the number of latent dimensions = 4 and the batch size = 1024.)

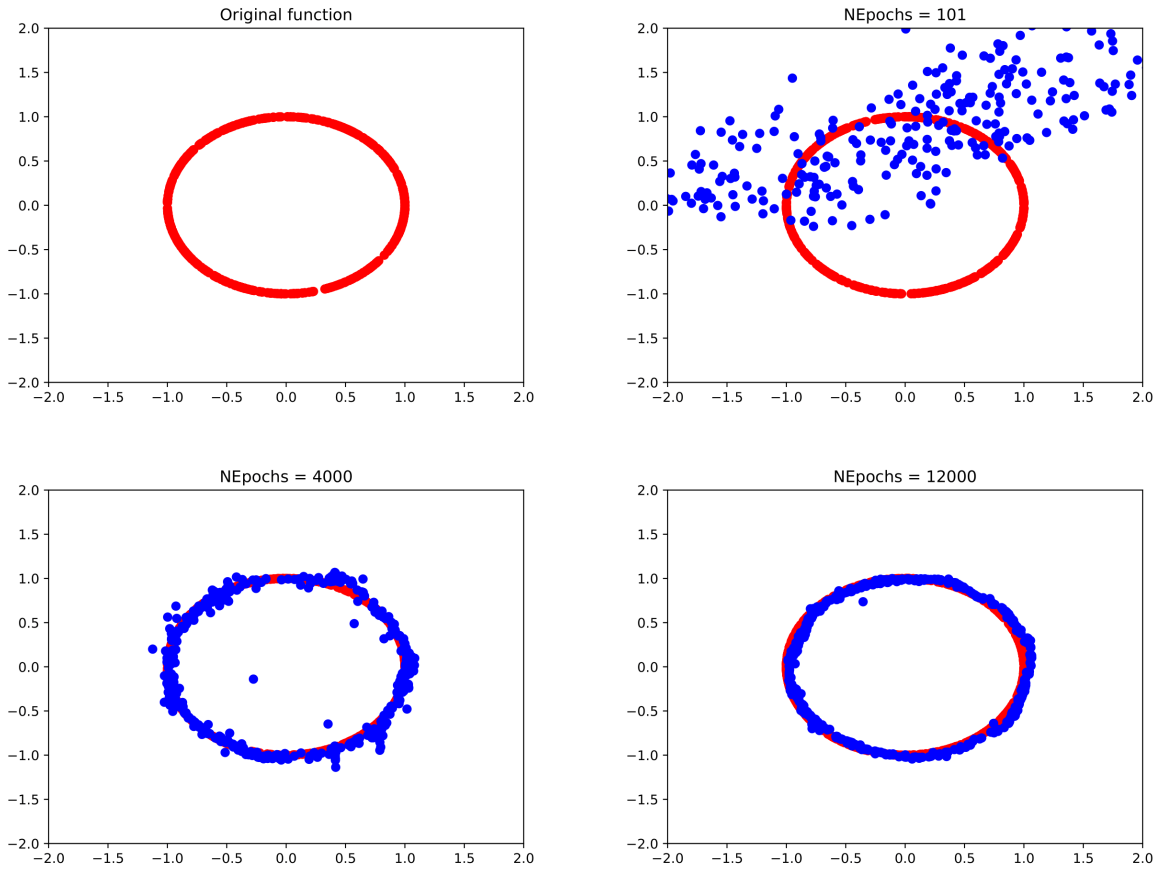


Figure 3.3: The figure shows the evolution of the GAN training for untrained, after 100 epochs, after 4000 epochs, and at end of training (12000 epochs) respectively. The target function is $x^2 + y^2 = 1$

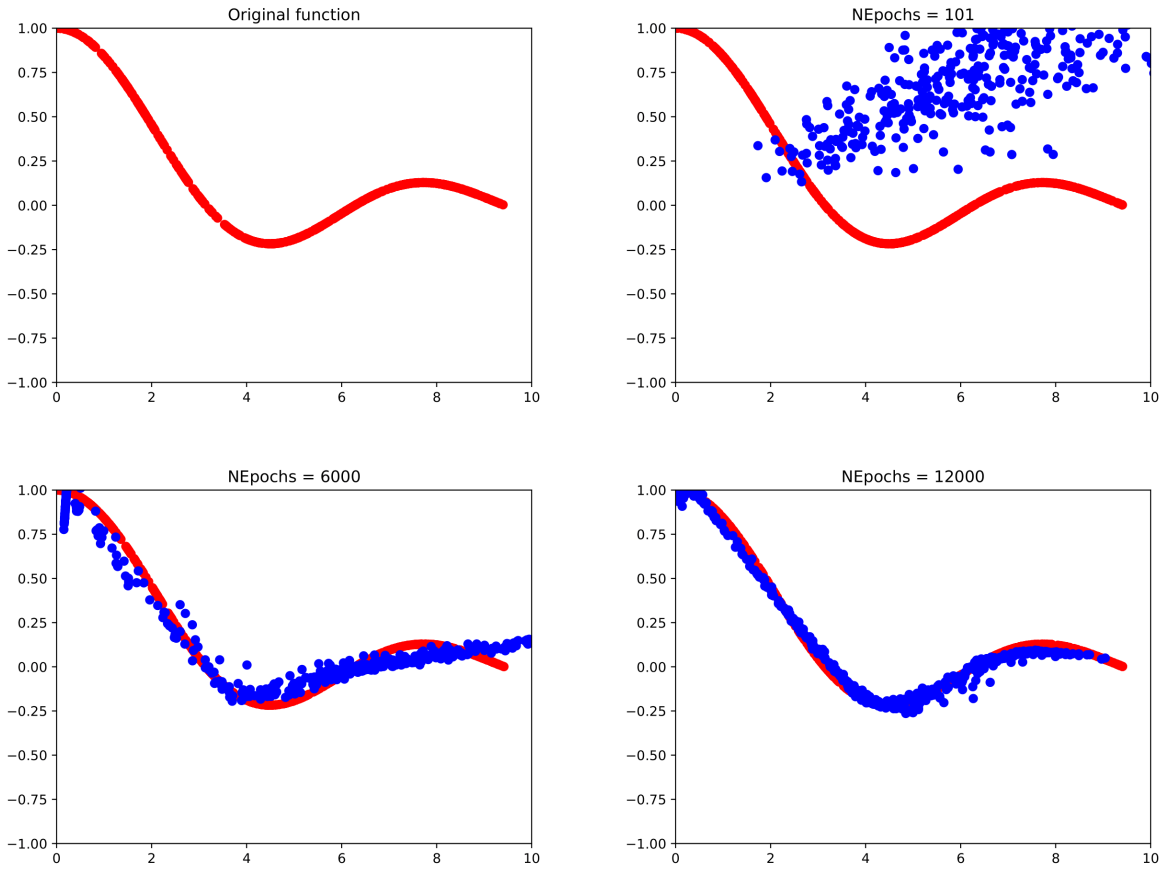


Figure 3.4: The figure shows the evolution of the GAN training for untrained, after 100 epochs, after 6000 epochs, and at end of training (12000 epochs) respectively. The target function is $y = sin(x)/x$

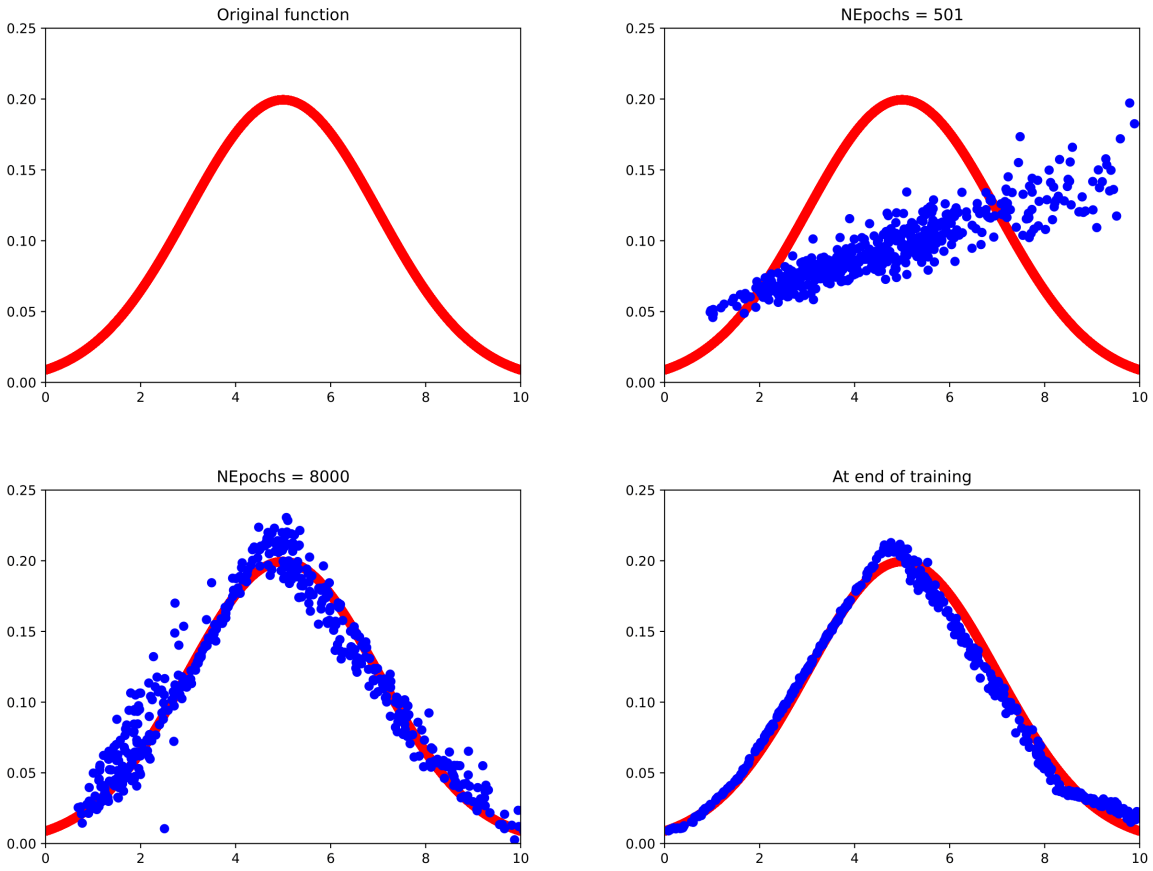


Figure 3.5: The figure shows the evolution of the GAN training for untrained(a), after 500 epochs(b), after 8000 epochs(c), and at end of training (12000 epochs) (d) respectively. The target function is $e^{-(x-5)^2/4}$

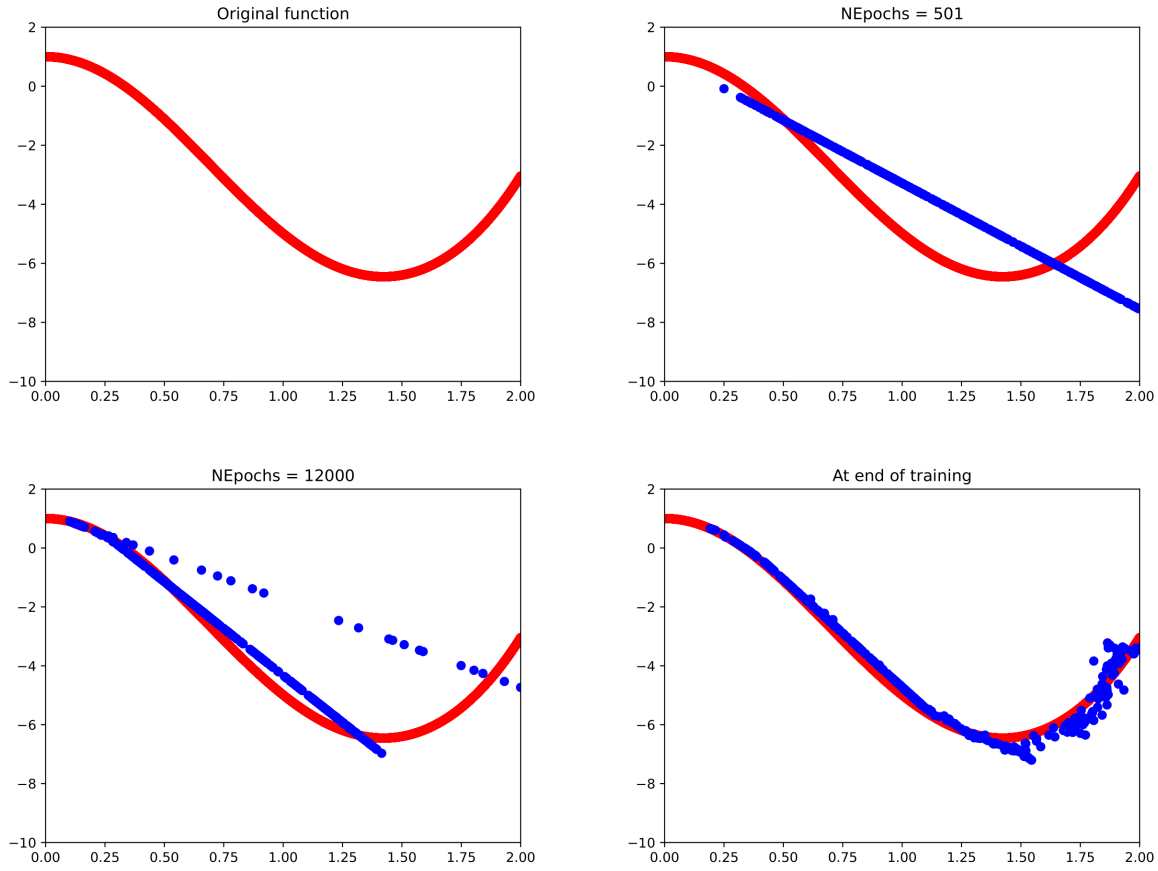


Figure 3.6: The figure shows the evolution of the GAN training for untrained, after 500 epochs, after 12000 epochs, and at end of training (20000 epochs) respectively. The target function is $y = x^4 - 5x^2 + cos(3x)$

3.3 Predicting kinematics in Drell-Yan process

Drell-Yan process

The Drell-Yan process takes place when a fermion-antifermion pair decays into a pair of fermion-antifermion via a Z boson or a virtual photon. The Feynman diagram is given below (Figure 3.7).

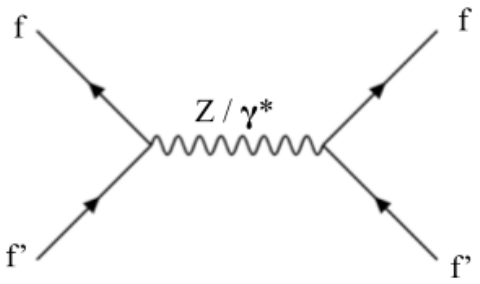


Figure 3.7: Feynman diagram for Drell- Yan process. (Where $\ell^{\pm}$ can be an $e^{\pm}$ or a $\mu^{\pm}$)

Here we are predicting the following variables; ΔR, $\Delta\phi$) and transverse momentum of fermions (p_T). The angular seperation between fermions (ΔR) between fermions is a measure of the distance between two fermions in the detector. It is defined as the square root of the sum of the differences in pseudorapidity ($\Delta\eta$) and azimuthal angle ($\Delta\phi$) between the two fermions. Pseudorapidity (η) is a measure of the angle between the trajectory of the particle and the beam axis, while azimuthal angle (ϕ) is the angle of the particle's trajectory in the plane perpendicular to the beam axis.

$$\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2}$$

The Azimuthal angle seperation between fermions ($\Delta\phi$) is defined as the difference in azimuthal angle between the two leptons in the center-of-mass frame of the collision. It can be used to study the spin alignment of the produced lepton-antilepton pair. The transverse momentum (p_T) is the momentum component that lies in the plane perpendicular to the beam axis in a particle accelerator. p_T for a particle is defined as;

$$p_T = \sqrt{p_x^2 + p_y^2}$$

Here, p_T^1 is the transverse momentum of the leading fermion and p_T^2 is the transverse momentum of the subleading fermion. Leading particle is the most energetic particle in shower. Subleading particle is the second most energetic particle in shower.

Training phase

To train the GAN, we used 101216 $Z \rightarrow \mu^+\mu^-$ events as input. The scaling of input variables to (-1,1) range was performed to ensure that they carry equal weightage during the training process. Both the generator loss function and discriminator loss function used was binary cross-entropy. After completing the training, we rescaled the variables to their original range. The trained model was then used to analyze the distributions of ΔR, $\Delta\phi$, p_T^1, and p_T^2 for the Drell-Yan process. The original plots of different variables of Drell-Yan process are given below. (Figure 3.8)

Original plots of variables made using the input file

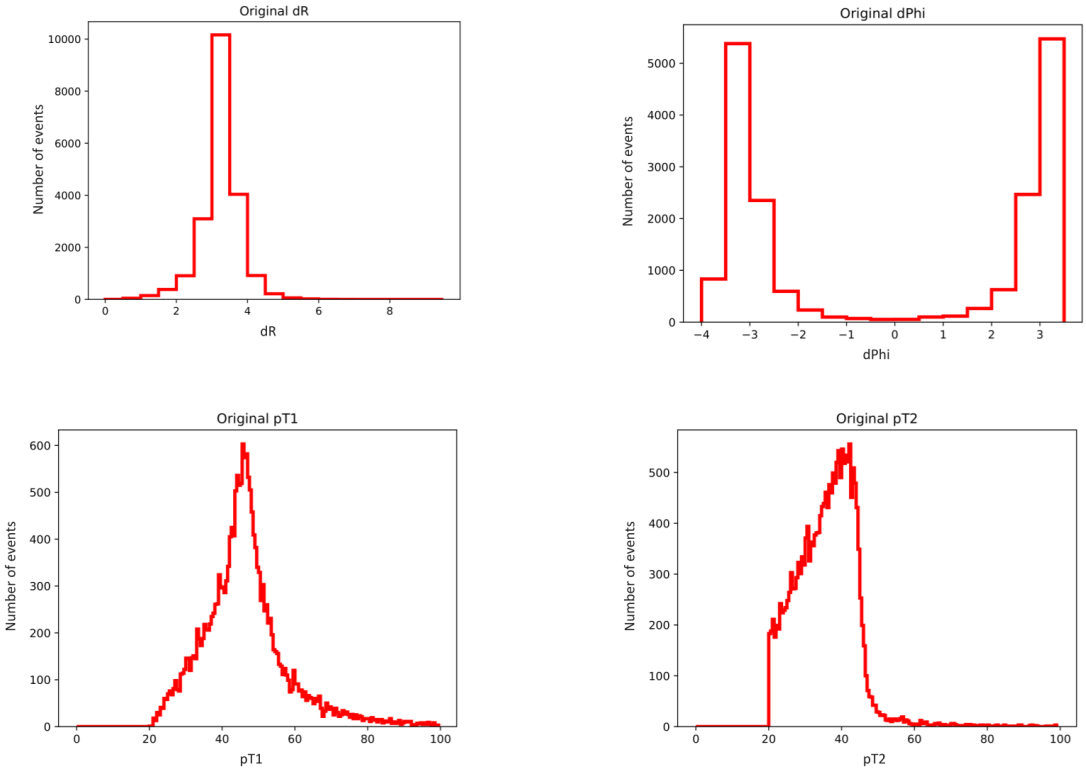


Figure 3.8: Original plots of ΔR, $\Delta\phi$, p_T^1 and p_T^2 respectively for Drell-Yan process.

After experimenting with various combinations of latent dimensions, batch sizes, epochs, and network architectures, the optimal values for each of these parameters were determined. The optimal values of various training parameters are tabulated below (Table 3.2).

Generator architecture	256, 128, 64, 64, 8, 4
Discriminator architecture	128, 64, 64, 16, 1
Number of epochs	12000
Number of latent dimensions	12
Batch size	4096
Loss function	Binary Cross-entropy

Table 3.2: Training information for optimal values of hyperparameters for DY process using GANs

The outputs which are mostly in agreement are given below (Figure 3.9). But, clearly, the agreement is not perfect.

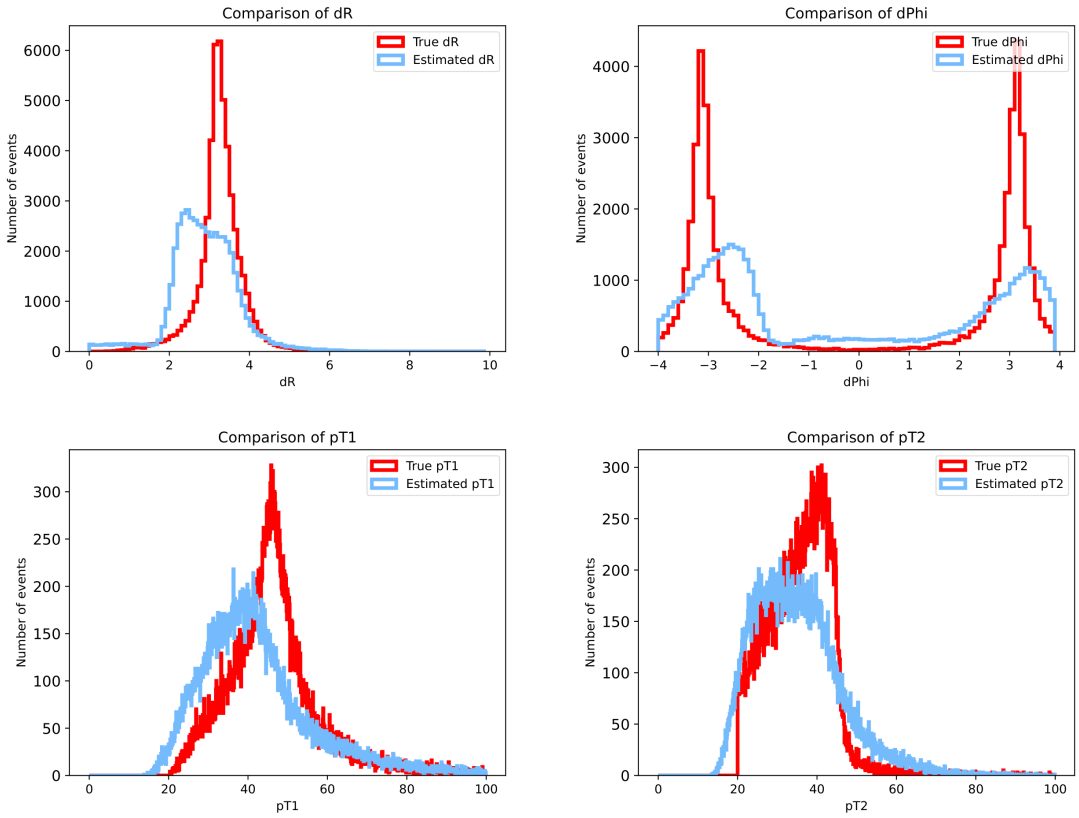


Figure 3.9: Comparison of GAN outputs of ΔR, $\Delta\phi$, p_T^1 and p_T^2 respectively for Drell-Yan process

3.4 Predicting four-vector of a single electron

Four-vector describes the energy and momentum of a particle. It has four components; the momentum of the particle in the x, y, and z directions (p_X, p_Y, and p_Z) and the energy of the particle (E). The energy-momentum four-vector is conserved in all particle interactions.

Training phase

The GAN was trained on a dataset of 70520 $Z \rightarrow e^+e^-$ events as input. To ensure equal weightage during training, all input variables were scaled to the range (-1,1), and were subsequently rescaled to their original range once the training was completed. The generator loss function employed was the mean square error and discriminator loss function was binary cross-entropy. The trained model was then used to generate the four vector distributions of Drell-Yan process. The figure 3.10 shows the original distributions of these variables.

Original plots of variables made using the input file

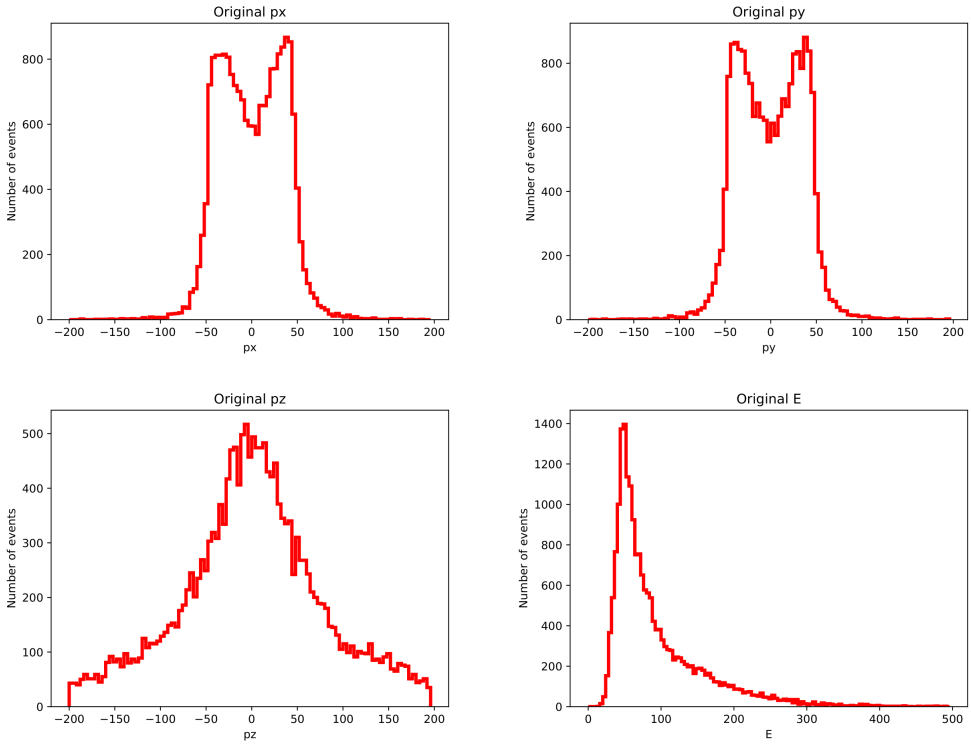


Figure 3.10: Original plots of p_X, p_Y, p_Z and E respectively for Drell-Yan process

Upon trying different combinations of latent dimensions, batch sizes, epochs, and network architectures, the most effective values for each of these parameters were identified. The optimal values of the various training parameters are tabulated below (Table 3.3).

Generator architecture	128, 64, 64, 32, 8, 4
Discriminator architecture	128, 64, 32, 8, 1
Number of epochs	12000
Number of latent dimensions	10
Batch size	4096

Table 3.3: Training information for optimal values of hyperparameters for DY process using GANs

Below are the outputs that are considered to be the most optimal (Figure 3.11). Here, the agreement is better compared to figure 3.9

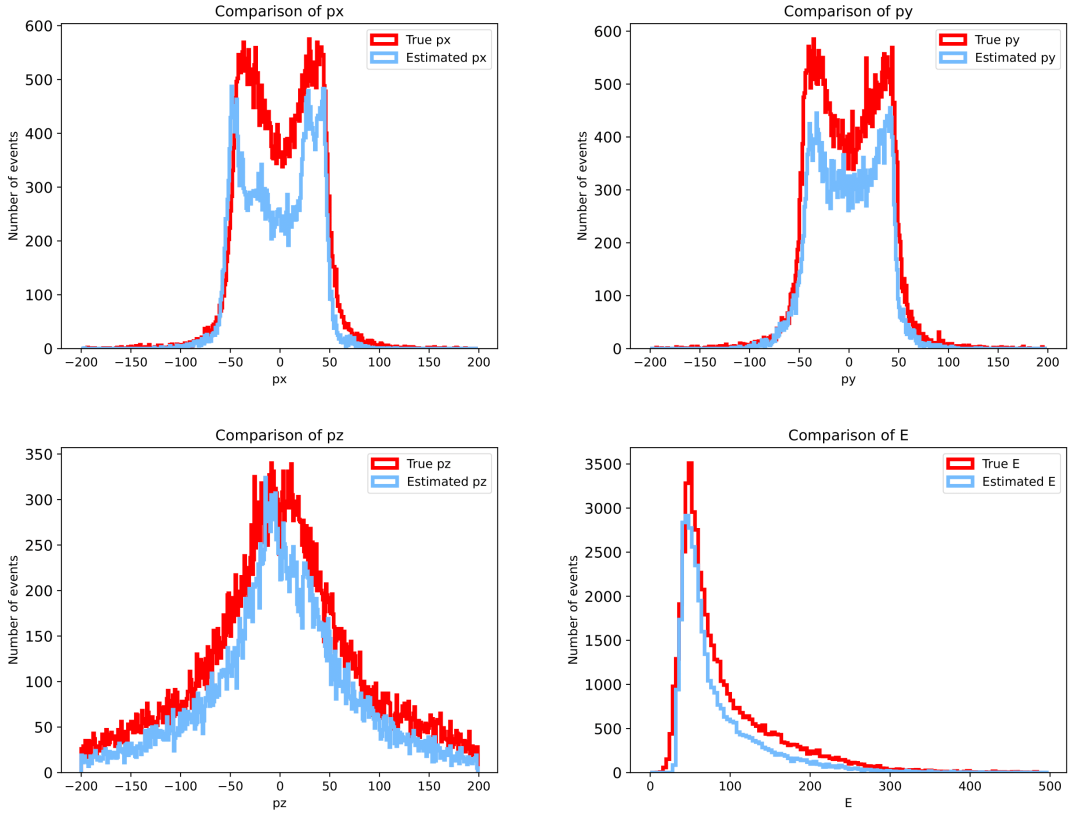


Figure 3.11: Comparison of GAN outputs of p_X, p_Y, p_Z and E distributions respectively for Drell-Yan process

3.5 Producing various distributions relevant in W+jets events

In W+jets events, we tried to produce distribution of variables like M_T of the lepton, dijet mass, M_T of the dijet, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T. These variables are defined in section 1.3.

Training phase

To train the GAN, we used 252587 W+jets events as input. The normalization of input variables to (-1,1) range was performed to ensure that they carry equal weightage during the training process. After completing the training, we scaled the variables back to their original range. The generator loss function employed was the mean square error and discriminator loss function was binary cross-entropy. The trained model was then used to analyze the distributions of M_T of the lepton, dijet mass, M_T of the dijet, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T distributions for W+jets process. The original distributions of these variables made using the input file provided is given below in figure 3.12.

First, we tried to produce four variables; M_T of the lepton, dijet mass, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T first. The resuls were unsatisfactory, but we decided to produce five variables also. They are M_T of the lepton, dijet mass, M_T of the dijet, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T. Upon trying different combinations of latent dimensions, batch sizes, epochs, and network architectures, comparatively better values for each of these parameters were identified. These values of various training parameters are tabulated below (Table 3.4).

Original plots of variables made using the input file

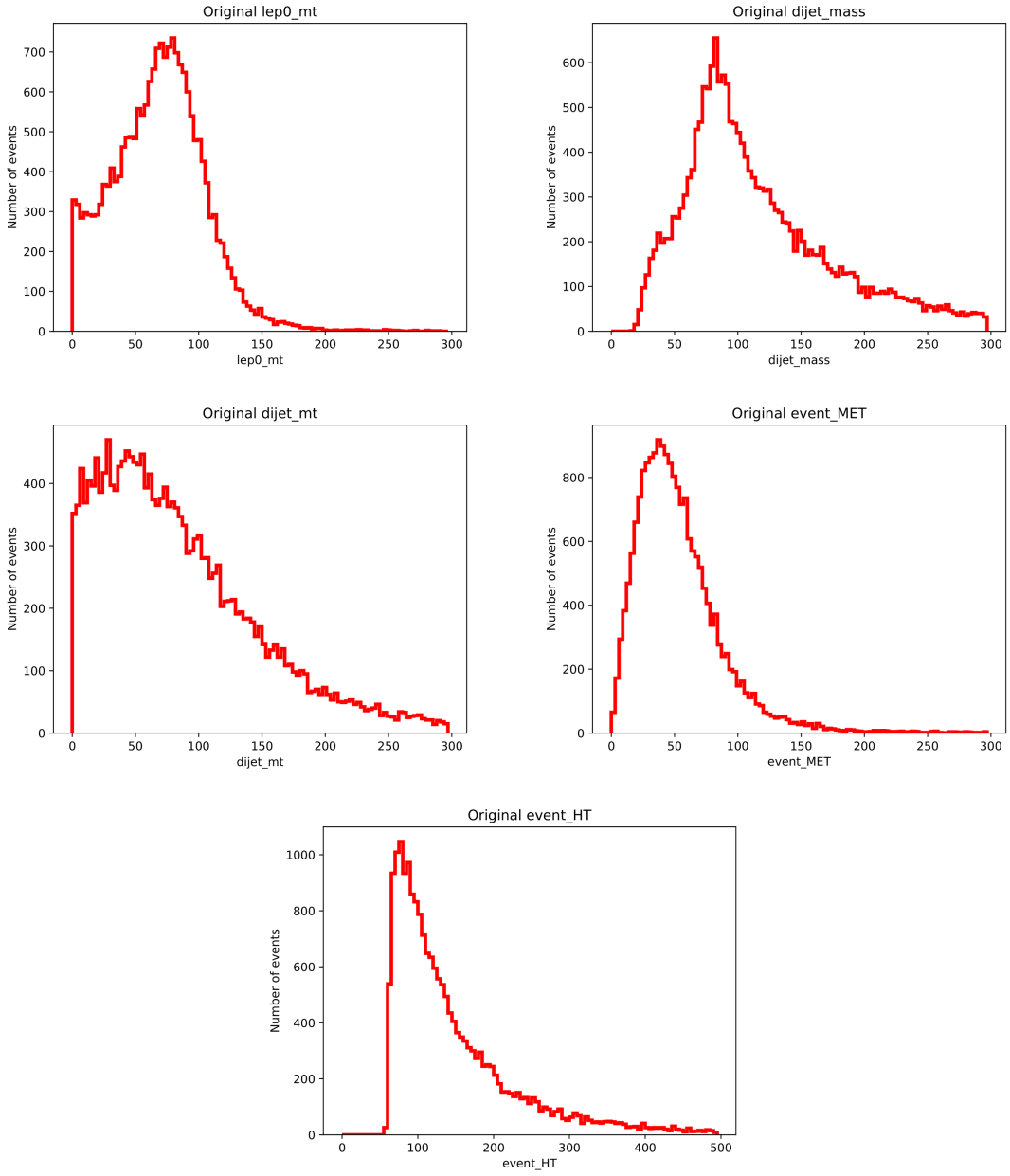


Figure 3.12: Original plots of M_T of the lepton, dijet mass, M_T of the dijet, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T respectively for Drell-Yan process.

Training parameters	For producing 4 variables	For producing 5 variables
Generator architecture	128, 128, 128, 64, 32, 16, 4	128, 128, 128, 64, 16, 8, 5
Discriminator architecture	128, 128, 128, 64, 32, 16, 1	128, 128, 64, 32, 16, 1
Number of epochs	6000	10000
Number of latent dimensions	10	10
Batch size	1024	1024

Table 3.4: Training information for optimal values of hyperparameters for W+jets process using GANs

The outputs we got are given below. Figure 3.13 shows the resulting distributions of four variables and figure 3.14 are the results of five variable problem.

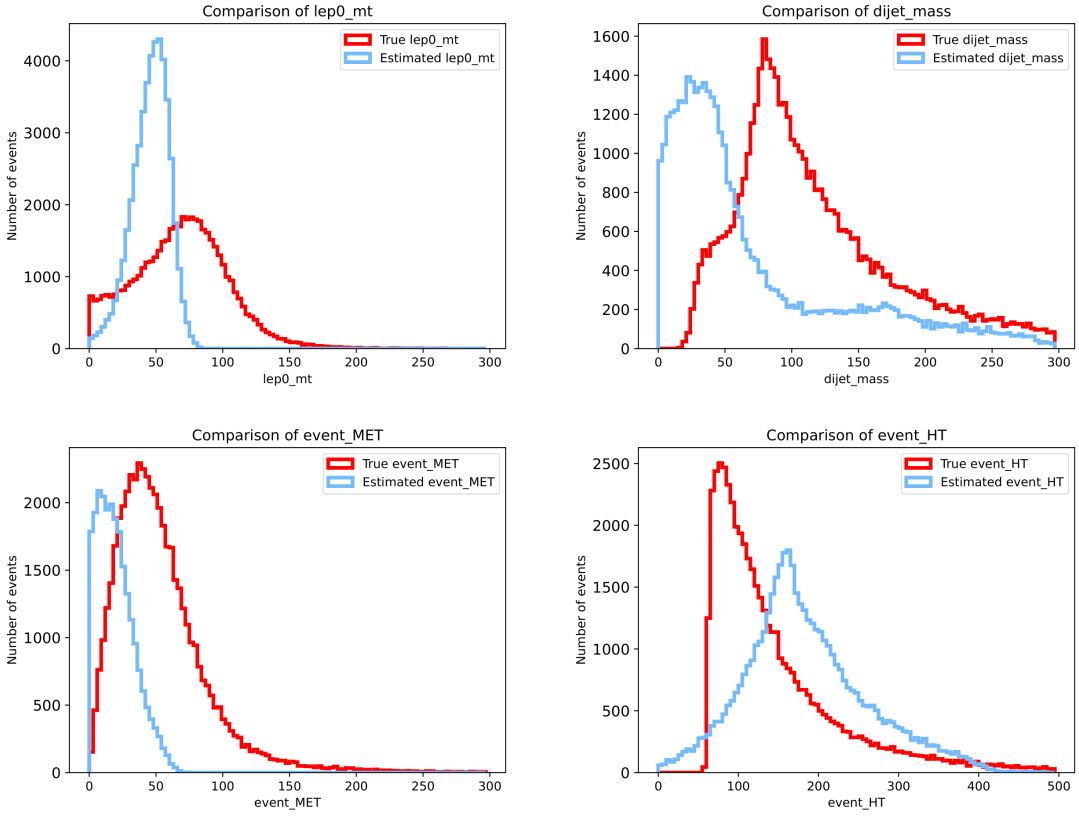


Figure 3.13: Comparison of GAN outputs of M_T of the lepton, dijet mass, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T respectively for W+jets process

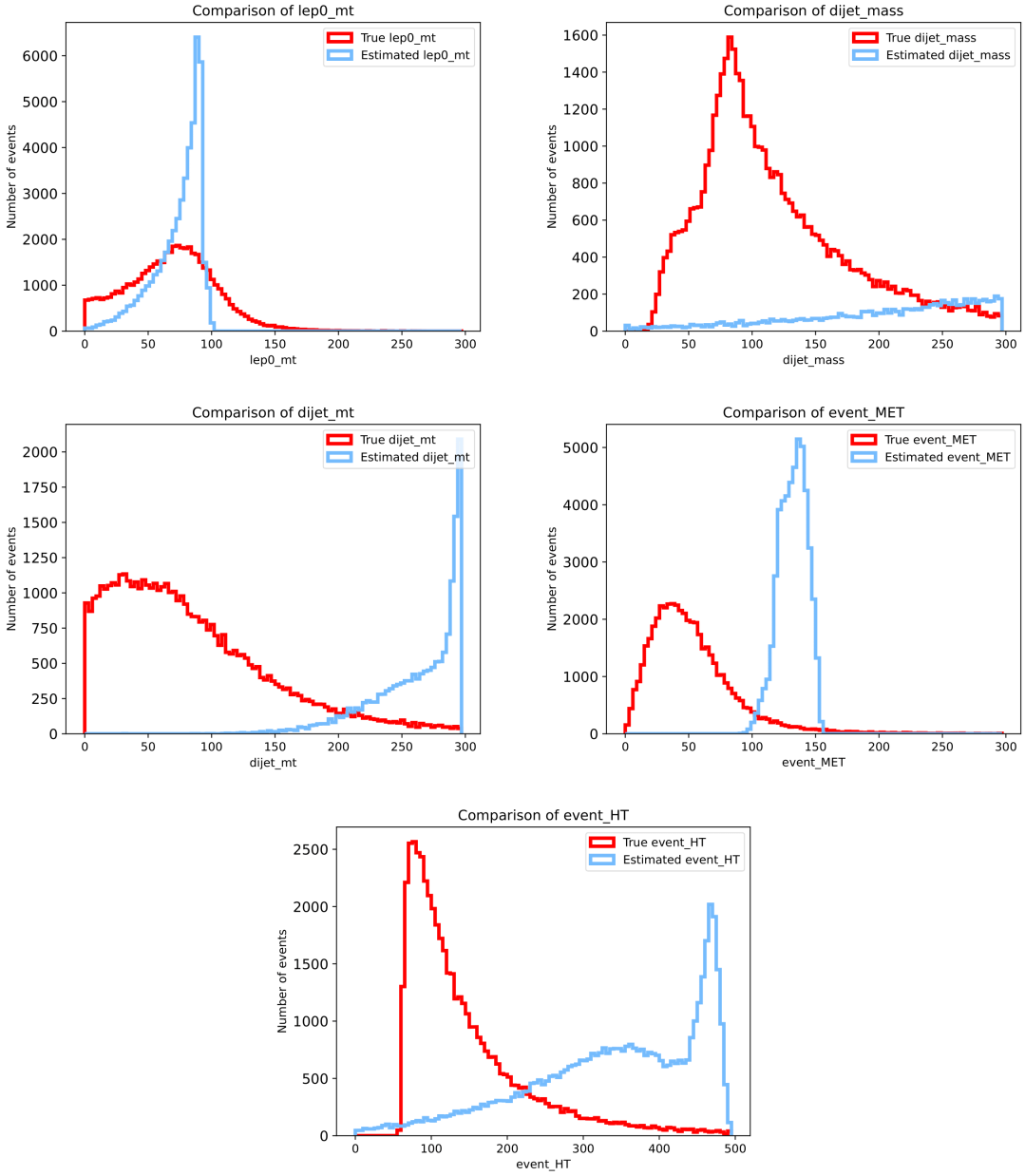


Figure 3.14: Comparison of GAN outputs of M_T of the lepton, dijet mass, M_T of the dijet, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T respectively for W+jets process process

The agreement of generated outputs is worse. GANs are not performing well in this problem, which needs further investigation. We decided to try Variational Autoencoders (VAEs), which can also be used for generating new data.

Chapter 4

Variational Autoencoders (VAEs)

Autoencoders are a kind of deep learning algorithm that contains a pair of neural networks; an encoder and a decoder [22]. The illustration of a basic autoencoder is given in Figure 4.1. The encoder receives input and compresses the data into a new lattice representation in a reduced dimension. The decoder maps the points in the latent space back to the original input space with some reconstructional error. An autoencoder learns a compressed representation of the input data in the latent space that can be used for tasks such as data compression or denoising.

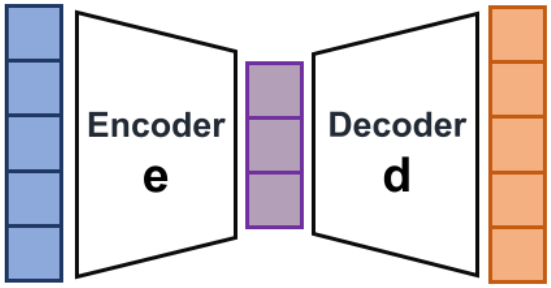


Figure 4.1: An illustration of an Autoencoder

A variational autoencoder is also an autoencoder with an additional regularisation step [23]. The origin of the term ”variational” is from the close relation between regularisation and the variational inference method in statistics. In addition to encoding and decoding data, a VAE aims to learn a probability distribution over the latent space. The regularisation process ensures that the latent

space has good properties for the generative process and avoids overfitting. This is achieved by adding a KL divergence term to the loss function, which encourages the latent space to follow a prior distribution, typically a standard normal distribution. This regularization step allows for the generation of new data by sampling from the learned probability distribution over the latent space, making the VAE a generative model. The structure of a VAE is depicted in figure 4.2.

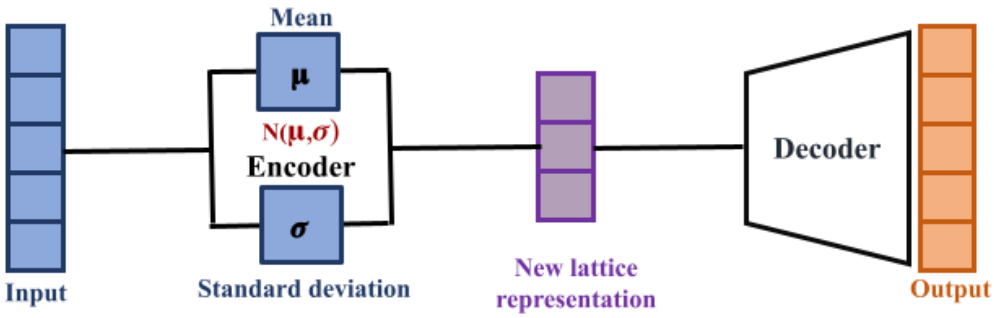


Figure 4.2: An illustration of a Variational Autoencoder (VAE)

The key difference between an autoencoder and a VAE is that the VAE explicitly models the probability distribution over the latent space, enabling it to generate new data samples. While an autoencoder can be used for data compression and denoising tasks, it is not inherently a generative model.

VAE training

During training, the encoder network maps the input data to a lower-dimensional latent space and provides the mean and variance of the latent variables as output. The latent variables are then sampled from the distribution defined by the mean and variance. These latent variables are then fed into the decoder network, which reconstructs the original input data. The loss function is then calculated, and the model weights are updated through backpropagation to minimize the loss. VAE loss function typically consists of two terms; a reconstruction loss that measures the difference

between the input data and the reconstructed output, and a KL-divergence loss that measures the difference between the distribution of the encoded latent variables and a known prior distribution.

Activation functions

Encoder and decoder are both neural networks with an input layer, an output layer and several hidden layers of neurons. In the encoder, both the inner layer neurons and output layer neurons are activated by the ReLU activation function. The decoder has all the inner layer neurons activated by ReLU activation functions and the outer layer activated by a linear activation function.

Loss functions

VAE loss function consists of two parts, reconstruction loss and the regularisation loss [24]. The reconstruction loss measures how well the VAE can reconstruct the input data from the latent space. It is typically calculated using a distance metric, such as mean squared error or binary cross-entropy, between the input and the reconstructed data. This term encourages the VAE to learn a compact representation of the input data that can be used to reconstruct the original data with minimal loss. The Kullback Leibler (KL) divergence loss or the regularisation loss measures the difference between the distribution of the encoded latent space and a prior distribution that is usually assumed to be Gaussian [25]. This term helps to regularize the latent space to be smooth and continuous, which enables interpolation and exploration of the latent space.

In this model, reconstruction loss is calculated using mean squared error (MSE) and regularisation loss is calculated by assuming the prior distribution to be $N \sim (0, 1)$. The optimizer used was adam.

Hyper parameters

- **Number of latent dimensions**
  The latent space refers to a lower-dimensional space which contains a compressed representation of the input data, where the most important features are preserved. Typically, the number of latent dimensions is lesser than the input data [26]. By optimising the dimension-

ality of the latent space, the quality of the reconstructions and the interpretability of the latent space can be improved. A smaller dimensional latent space can result in a more compact and interpretable representation of the data, while a larger dimensional one can allow for more flexibility in the reconstruction process.

- **Reconstruction loss weight**
  The reconstruction loss weight determines the importance of the reconstruction loss term in the overall VAE loss function. If this value is set too high, the VAE will focus more on reconstructing the input data accurately, but may not learn a useful latent representation. On the other hand, if the reconstruction loss weight is set too low, the VAE may learn a good latent representation but may not generate high-quality reconstructions of the input data. The optimal value of this parameter provides a balance between the reconstruction loss and the KL divergence loss term.

4.1 Producing various distributions relevant in Drell- Yan process

4.1.1 Producing ΔR, $\Delta\phi$ and mass distributions

Here, we are predicting the angular seperation between fermions (ΔR), azimuthal angle seperation between fermions ($\Delta\phi$) and mass. Here, mass is the invariant mass of fermions in the event.

Training phase

To train the VAE, we used 70520 $Z \rightarrow e^+e^-$ events as input. The normalization of input variables to (-1,1) range was performed to ensure that they carry equal weightage during the training process. After completing the training, we rescaled the variables to their original range. The loss function employed was the weighted mean square error. The trained model was then used to analyze the distributions of ΔR, $\Delta\phi$, and invariant mass for the Drell-Yan process. Figure 4.3 shows the original plots of these variables made using the input file.

Original plots of variables made using the input file

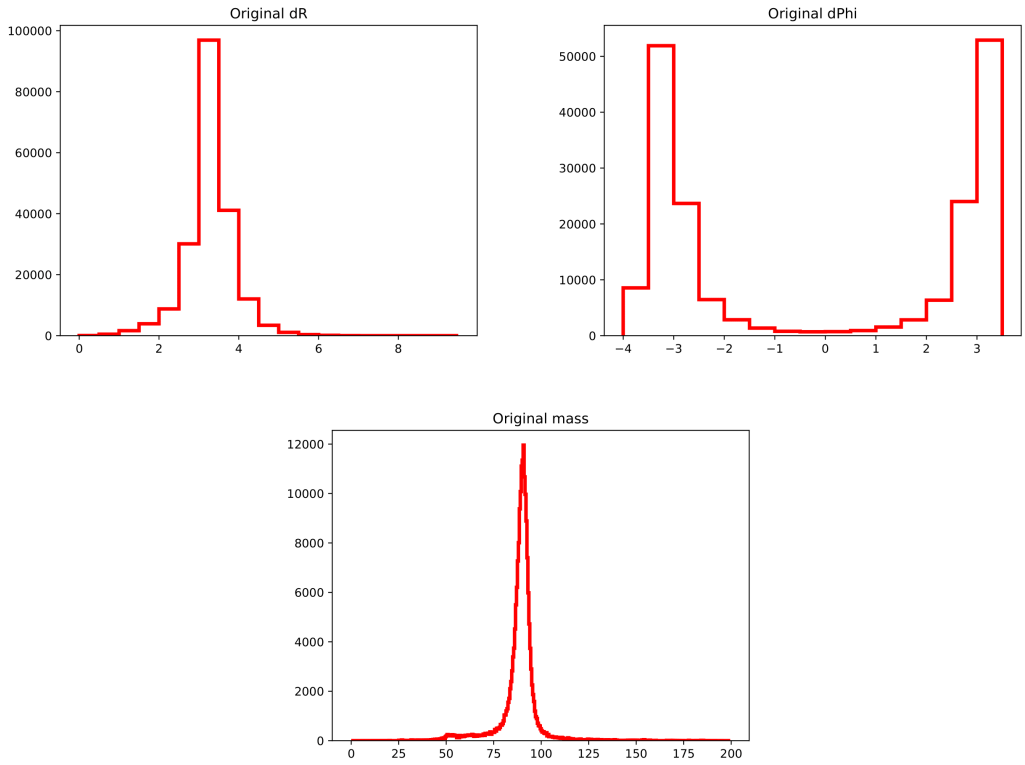


Figure 4.3: Original plots of ΔR, $\Delta\phi$, and mass respectively for Drell-Yan process

After experimenting with various combinations of latent dimensions, batch sizes, epochs, and network architectures, the optimal values for each of these parameters were determined. The optimal values of various training parameters are tabulated below (Table 4.1).

Encoder architecture	128, 64, 32, 16, 8
Decoder architecture	8, 16, 32, 64, 128
Number of epochs	10000
Number of latent dimensions	2
Batch size	1024
Test split size	0.1
Reconstruction loss factor	1
Weights of variables	0.75, 0.75, and 1 respectively for ΔR, $\Delta\phi$, and mass

Table 4.1: Training information for optimal values of hyperparameters for DY process using VAEs

The most optimal outputs are given in figure 4.4.

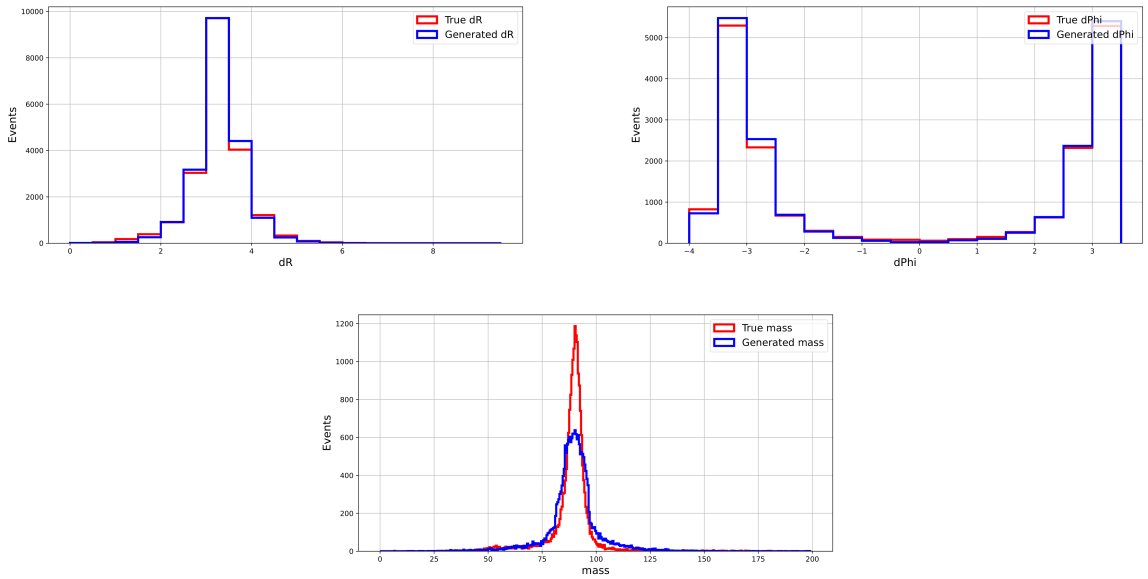


Figure 4.4: VAE outputs of ΔR, $\Delta\phi$, and mass distributions respectively for Drell-Yan process

4.1.2 Producing four vectors of a single electron

Here we are predicting p_X, p_Y, p_Z and E.

Training phase

We used 200000 Drell-Yan events as input to train the VAE. To ensure that the input variables carry equal weightage during the training process, the input variables were scaled to (-1,1) range. After completing the training, we rescaled the variables to their original range. The loss function employed was the weighted mean square error. Using the trained model, we generated p_X, p_Y, p_Z and E distributions of Drell-Yan process. The original four vector distributions are given in figure 4.5.

Original plots of variables made using the input file

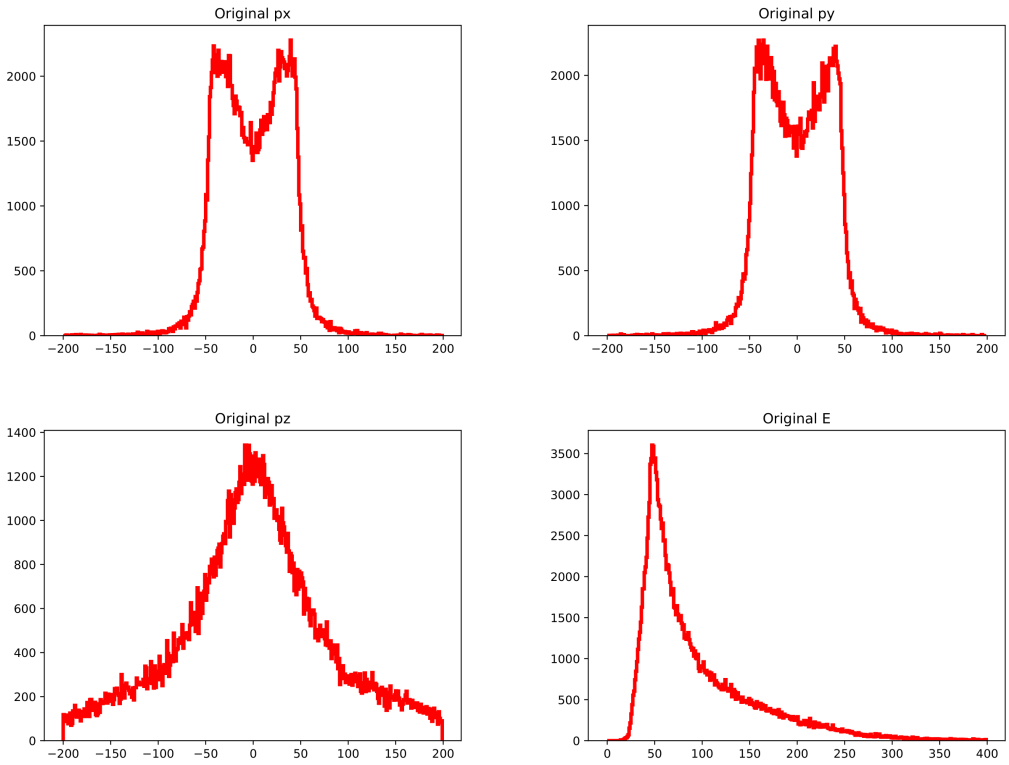


Figure 4.5: Original plots of p_X, p_Y, p_Z and E respectively for Drell-Yan process

Upon trying different combinations of latent dimensions, batch sizes, epochs, and network architectures, the most effective values for each of these parameters were identified. The optimal values of the various training parameters are tabulated below (Table 4.2).

Encoder architecture	128, 128, 128, 64, 32, 16, 8
Decoder architecture	8, 16, 32, 64, 128, 128, 128
Number of epochs	6000
Number of latent dimensions	3
Batch size	4096
Test split size	0.1
Reconstruction loss factor	1
Weights of variables	3.5, 3.5, 1.15 and 1.15 respectively for p_X, p_Y, p_Z and E

Table 4.2: Training information for optimal values of hyperparameters for DY process using VAEs

The outputs which are most in agreement are given below (Figure 4.6).

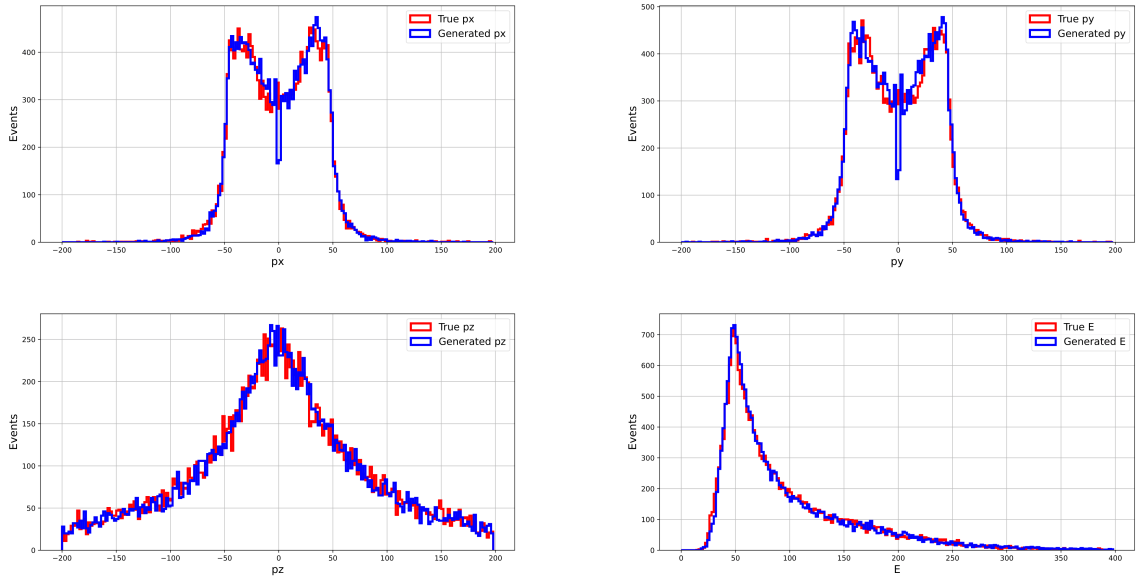


Figure 4.6: Comparison of VAE outputs of p_X, p_Y, p_Z and E distributions respectively for Drell-Yan process

4.2 Producing various distributions relevant in W+jets events

Training phase

To train the VAE, we used 252587 W+jets events as input. The normalization of input variables to (-1,1) range was performed to ensure that they carry equal weightage during the training process. After completing the training, we scaled the variables back to their original range. The loss function employed was the weighted mean square error. The trained model was then used to analyze the distributions of ΔR, $\Delta\phi$, and invariant mass for the Drell-Yan process. The original plots of these variables are given below. (Figure 4.7)

Original distributions of variables made using the input file

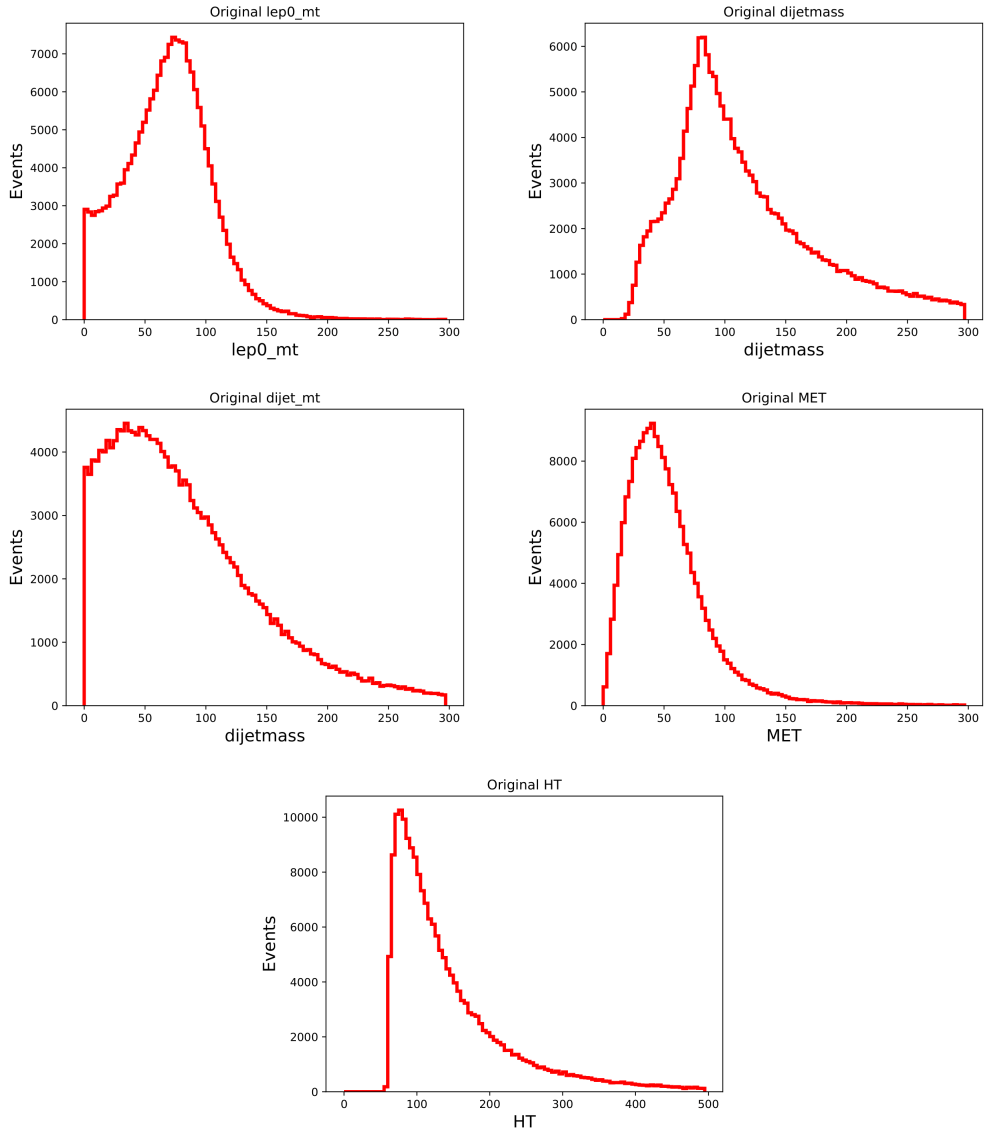


Figure 4.7: Original M_T of the lepton, Dijet mass, M_T of dijet, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T distributions respectively for W+jets events

After experimenting with various combinations of latent dimensions, batch sizes, epochs, and network architectures, the optimal values for each of these parameters were determined. The optimal values of various training parameters are tabulated below (Table 4.3). The most optimal outputs are shown in figure 4.8.

Generator architecture	128, 128, 128, 64, 32, 16, 8
Discriminator architecture	8, 16, 32, 64, 128, 128, 128
Number of epochs	1000
Number of latent dimensions	3
Batch size	1024
Test split size	0.1
Reconstruction loss factor	1
Weights of variables	3.55, 3.6, 3.4, 1.45 and 1.5 respectively for M_T of the lepton, Dijet mass, M_T of dijet, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T

Table 4.3: Training information for optimal values of hyperparameters for W+jets process using VAEs

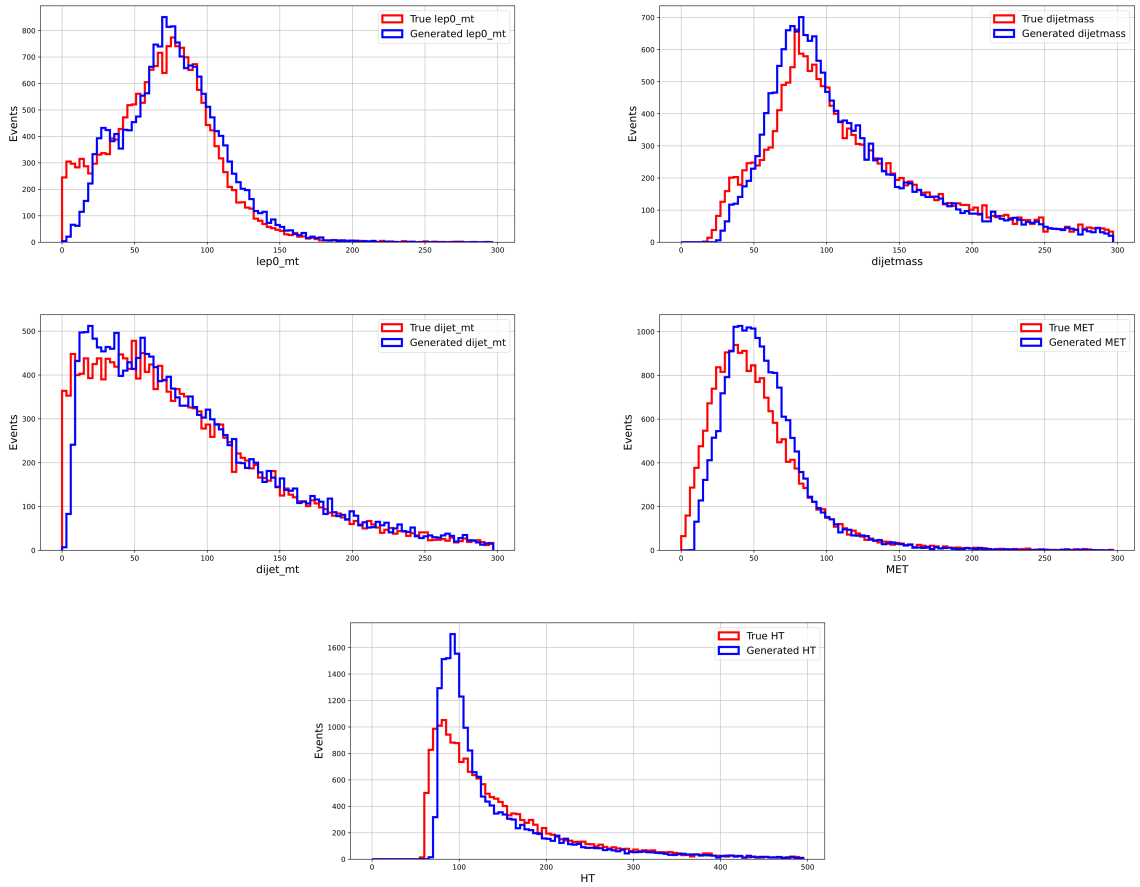


Figure 4.8: Comparison of VAE outputs of M_T of the lepton, Dijet mass, M_T of dijet, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T distributions respectively for W+jets events

4.3 Validation of generated W+jets events

In order to verify the reliability of the generated values, we developed a neural network classifier designed to differentiate between W+jets events and semileptonic $t\bar{t}$ events. We trained the neural network to distinguish between real W+jets events and semileptonic $t\bar{t}$ events. Following this, we evaluated the trained model by testing on a separate set of real W+jets, generated W+jets, and semileptonic $t\bar{t}$ events. We then plotted the neural network output scores and ROC curves to analyze the performance of the network. A flow chart depicting the entire process is given below (Figure 4.9).

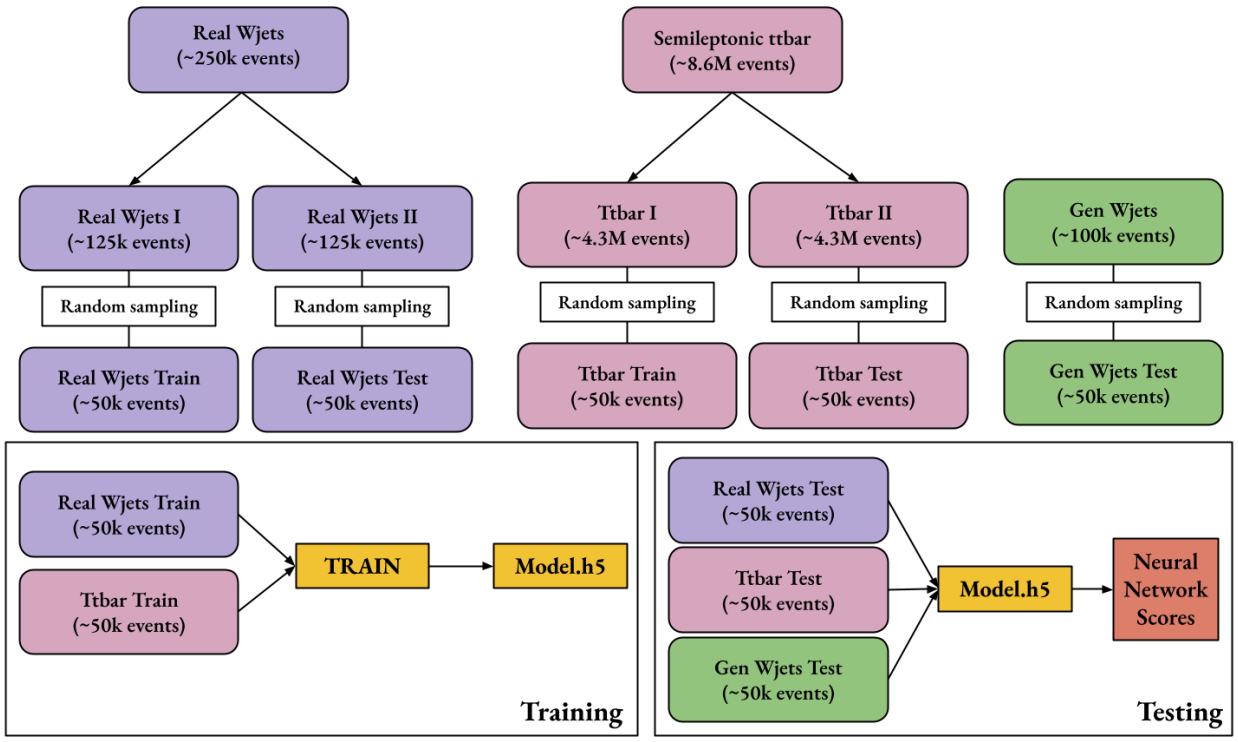


Figure 4.9: Flow chart of working of a neural network classifier that seperates W+jets events from semileptonic $t\bar{t}$

For training, we chose variables such as M_T of the lepton, dijet mass, M_T of the dijet system, $E_{\mathrm{T}}^{\mathrm{miss}}$ and H_T. The overlay plots of these variables are given in figure 4.10

Overlay plots of chosen variables of Real W+jets and Semileptonic $t\bar{t}$

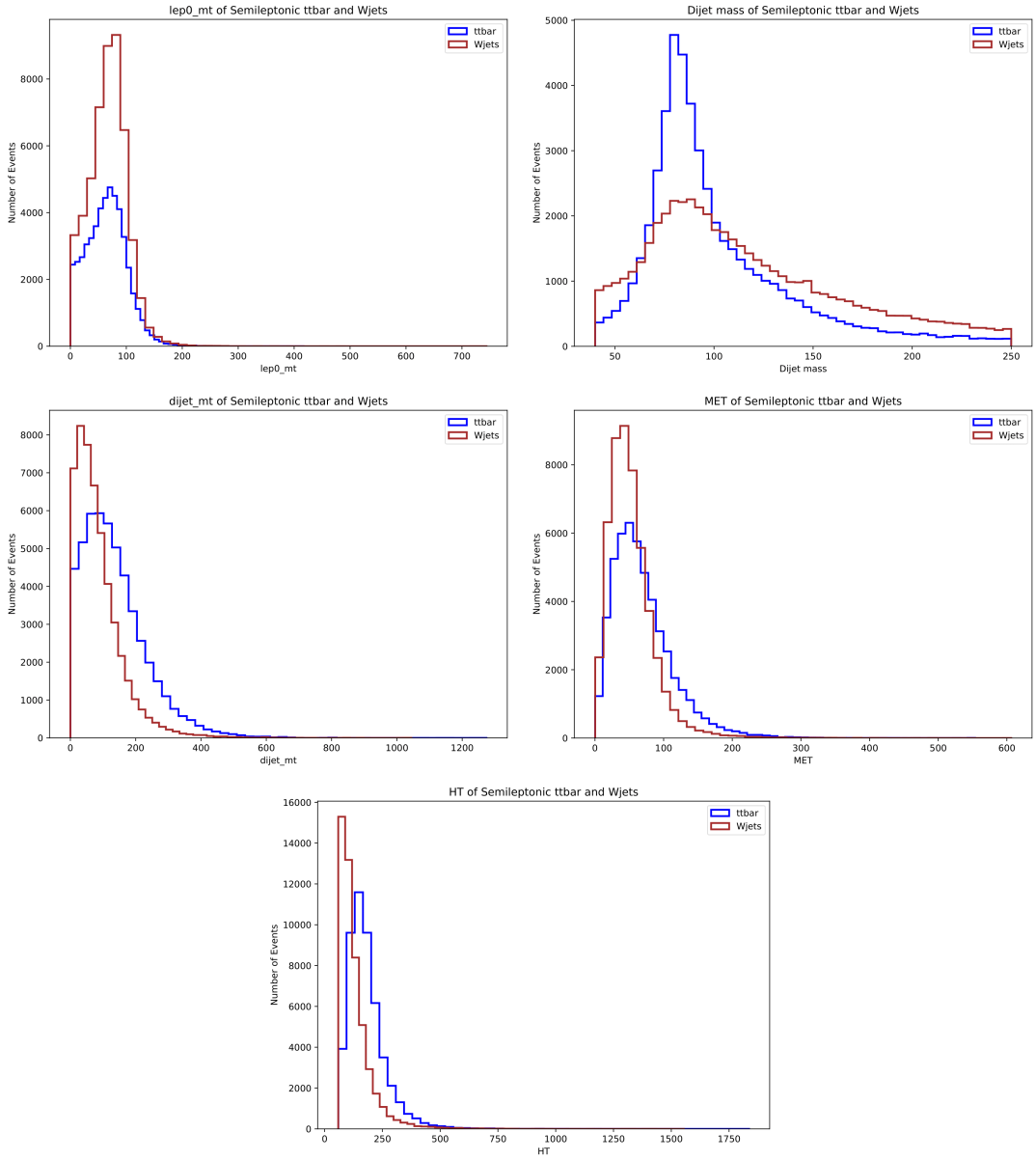


Figure 4.10: The figure shows the overlay plots of different variables for W+jets and semileptonic $t\bar{t}$ events

The NN score plots and ROC curves are included in chapter 5

Chapter 5

Results and discussion of simulation of W+jets events

While using GANs, we obtained some moderately agreeable preliminary results, but the rest were displeasing. We attempted to improve GANs, but could not find an effective solution. This requires further investigation. In the meantime, we explored VAEs and were able to produce various distributions using them. The results looked promising.

We examined the efficacy of the W+jets - semileptonic $t\bar{t}$ classifier across various sets of generated W+jets events. The training information utilized for this is outlined in the table provided below (Table 5.1).

Sl.No.	Weights of variables while calculating mean square error	NEpochs	Output plots
1	3.75, 3.6, 3.3, 1.45, 1.5	1000	Figure 5.1
2	3.6, 3.6, 3.3, 1.45, 1.5	1000	Figure 5.2
3	3.65, 3.6, 3.4, 1.45, 1.5	1000	Figure 5.3
4	3.6, 3.6, 3.4, 1.45, 1.5	1000	Figure 5.4
5	3.55, 3.6, 3.4, 1.45, 1.5	1000	Figure 5.5

Table 5.1: The table includes the training information for the variables M_T of the lepton, dijet mass, M_T of the dijet system, Missing E_T and H_T. (NEpochs is the number of epochs during training, number of latent dimensions = 3 and batch size = 1024. The network architecture is 128, 128, 128, 64, 32, 16, 8 for both encoder and decoder.)

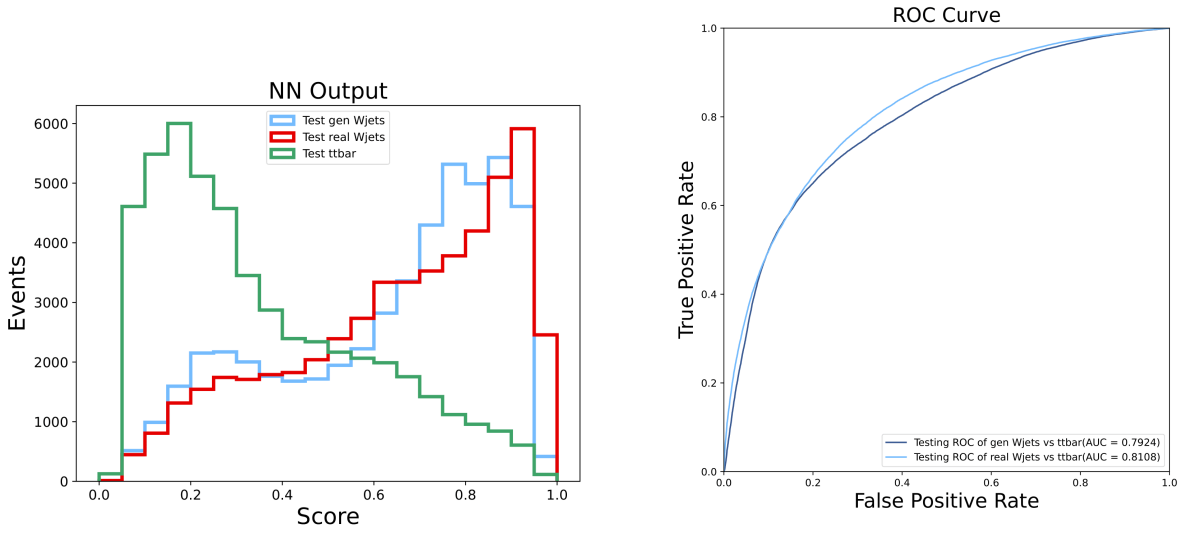


Figure 5.1: The figure shows the Neural network output score plots and ROC curve for W+jets - semileptonic $t\bar{t}$ classifier for combination 1

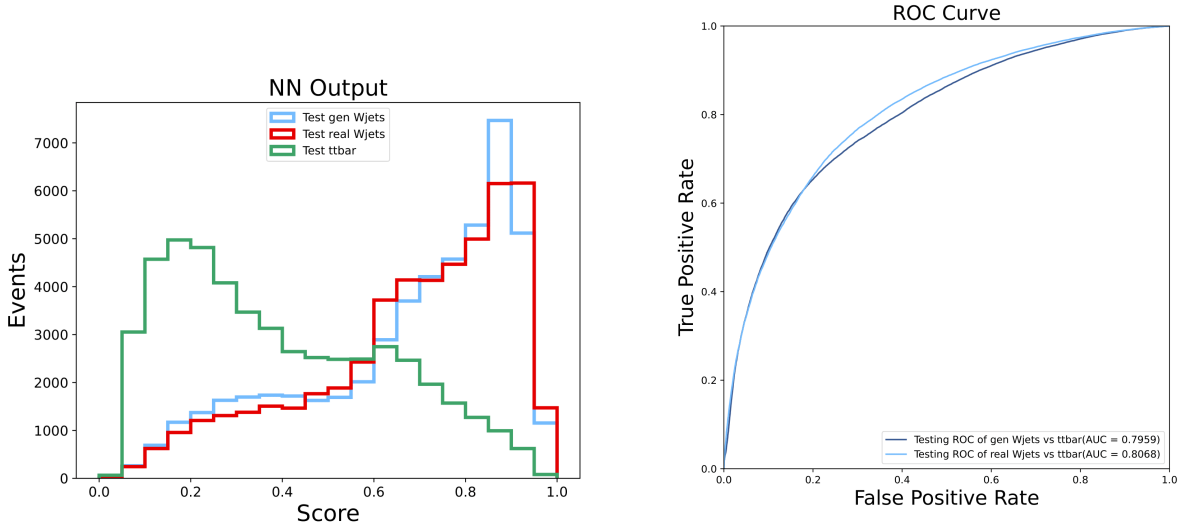


Figure 5.2: The figure shows the Neural network output score plots and ROC curve for W+jets - semileptonic $t\bar{t}$ classifier for combination 2

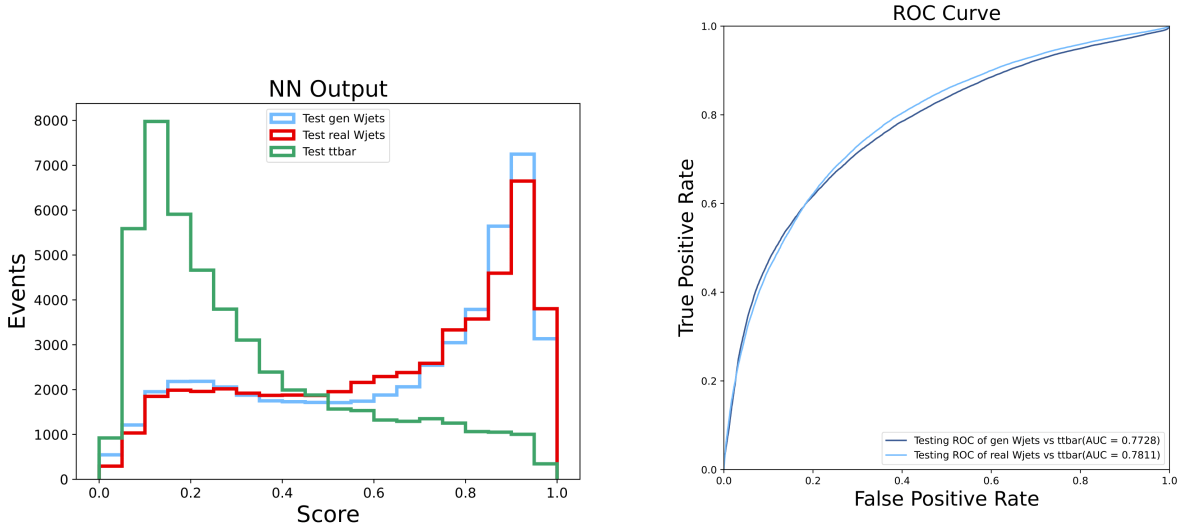


Figure 5.3: The figure shows the Neural network output score plots and ROC curve for W+jets - semileptonic $t\bar{t}$ classifier for combination 3

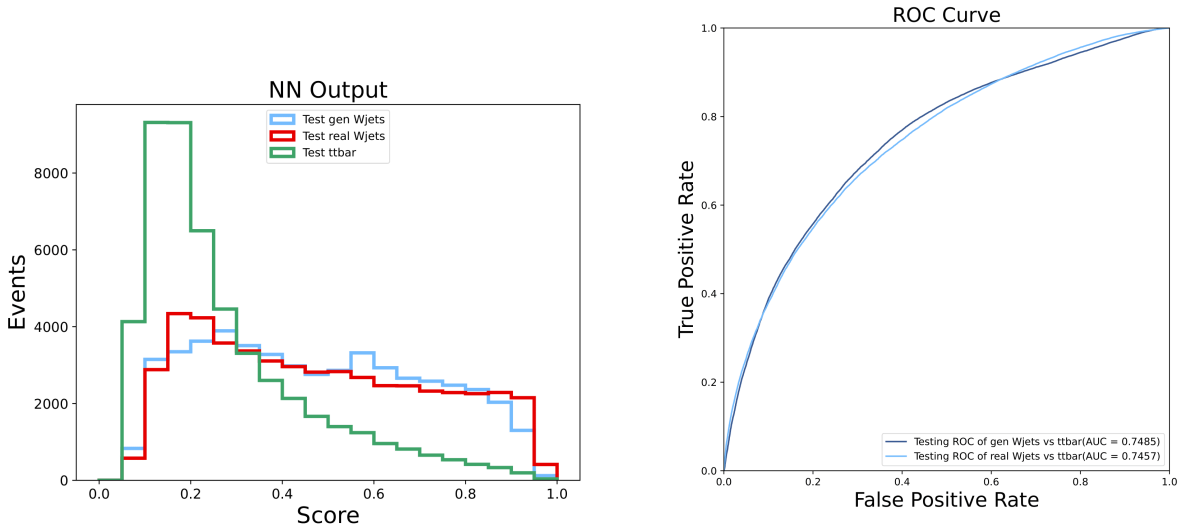
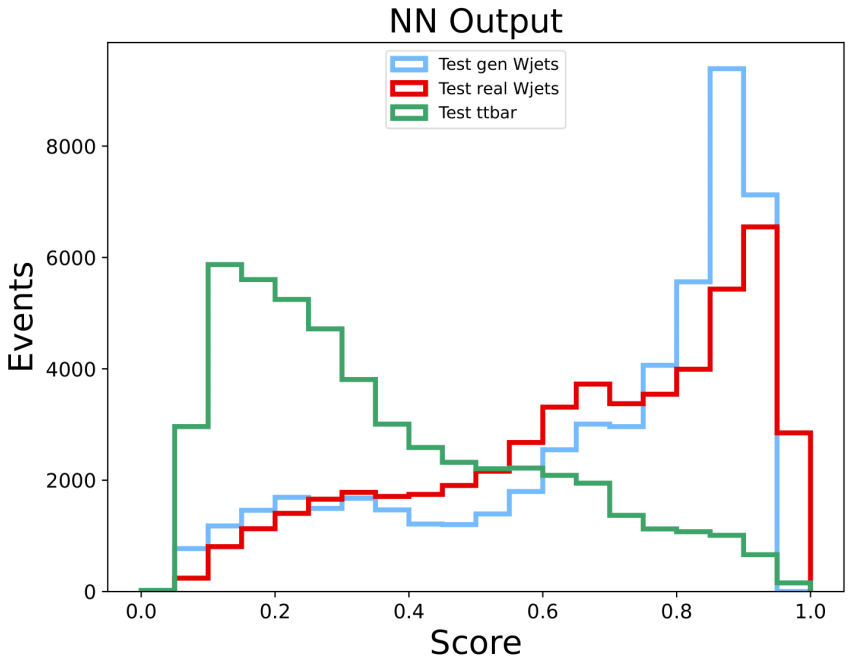


Figure 5.4: The figure shows the Neural network output score plots and ROC curve for W+jets - semileptonic $t\bar{t}$ classifier for combination 4



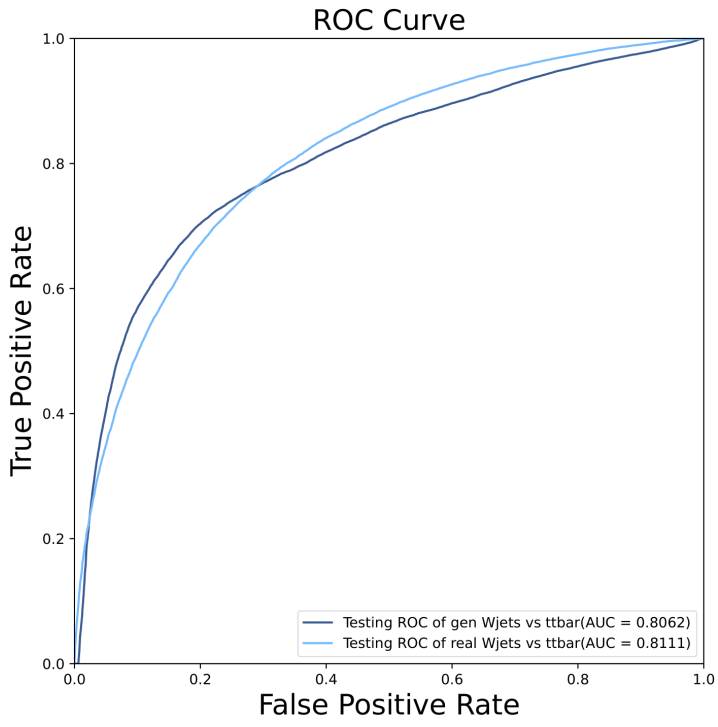


Figure 5.5: The figure shows the Neural network output score plots and ROC curve for W+jets - semileptonic $t\bar{t}$ classifier for combination 5

In our work, it was observed that VAEs outperformed GANs significantly. After testing various combinations, the fifth combination mentioned in the table 5.1 provided the highest level of accuracy.

Chapter 6

Summary

This thesis discusses an approach to simulating collision events using neural networks, specifically the Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). In the context of search for the singlet model of vector-like leptons in their $1\ell 2$j final state, the primary standard model background is $W+$jets events. The W has an extremely high cross section at the LHC. Thus, to predict data with low statistical uncertainty, as many events as possible should be simulated. While Monte Carlo platforms like GEANT4 can simulate these events, they are computationally heavy and time-consuming. Modern machine learning techniques employ multivariate algorithms such as neural networks to generate such events using less computational time and power. Thus, this study shows the efficacy of simulating collision events using deep learning neural networks such as GANs and VAEs. GANs and VAEs require much less computational time as they generate certain numbers following a known distribution.

The study first explored the capabilities of GANs by producing simple mathematical functions. Then produced the ΔR, $\Delta\phi$, p_T^1, and p_T^2 distributions of Drell-Yan events with moderate accuracy. We then moved on to produce four vectors of a single electron. GAN could produce really better results in this case. However, the trained GAN on $W+$jets events did not yield desirable results. The predicted outputs of a four-variable GAN were slightly better than those of five-variable GANs. As the GAN performance was worse, the study then explored VAEs, which can be used in similar problems. Sampling from the probability distribution $N \sim (0, 1)$, the VAE produced different distributions, such as $\Delta R, \Delta\phi$, mass, and four vectors of an object of DY process with significantly good accuracy. Moving on to the real problem of simulating $W+$jets events, the study produced

distributions of five variables: M_T of the lepton, dijet mass, M_T of the dijet system, $E_{\mathrm{T}}^{\mathrm{miss}}$, and H_T, achieving good accuracy. A W+jets-semileptonic $t\bar{t}$ classifier was then made to validate these generated values. The training ROC curve for this classifier was plotted, and the resulting AUC value was 0.8158. The generated values, which are mostly in agreement, gave a testing AUC value of 0.8111 for real W+jets vs $t\bar{t}$ and 0.8062 for generated W+jets vs $t\bar{t}$.

Once trained with optimal parameter values, these networks work extremely fast and can produce millions of events in significantly less time. VAEs provided results with the desired accuracy and were found to generate better results than GANs in the specific context of this study. This method has already been used for various purposes on CMS clusters and high-GPU devices. However, this study shows that the method can also be adopted to simulate collision events at smaller scales using the CPU. Additionally, we can generate only relevant variables instead of simulating the entire event description.

Bibliography

[1] M. Feickert and B. Nachman, [arXiv:2102.02770 [hep-ph]].

[2] Therhaag, Jan *EPJ Web Of Conferences*. **55** pp. 02003 (2013) https://doi.org/10.1051/epjconf/20135502003

[3] A. Tumasyan *et al.* [CMS], Phys. Rev. D **105** (2022) no.11, 112007 doi:10.1103/PhysRevD.105.112007 [arXiv:2202.08676 [hep-ex]].

[4] [CMS], CMS-PAS-SMP-15-004.

[5] R. L. Workman *et al.* [Particle Data Group], PTEP **2022** (2022), 083C01 doi:10.1093/ptep/ptac097

[6] B. Hashemi, N. Amin, K. Datta, D. Olivito and M. Pierini, [arXiv:1901.05282 [hep-ex]]

[7] K. T. Matchev, A. Roman and P. Shyamsundar, SciPost Phys. **12** (2022) no.3, 104 doi:10.21468/SciPostPhys.12.3.104 [arXiv:2002.06307 [hep-ph]]

[8] P. Musella and F. Pandolfi, Comput. Softw. Big Sci. **2** (2018) no.1, 8 doi:10.1007/s41781-018-0015-y [arXiv:1805.00850 [hep-ex]]

[9] [The ATLAS Collaboration], [arXiv:2210.06204 [hep-ex]]

[10] A. Butter and T. Plehn, [arXiv:2008.08558 [hep-ph]]

[11] https://www.ibm.com/in-en/topics/neural-networks

[12] https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

[13] https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9

[14] Ian J. Goodfellow *et al.*, [arXiv:1406.2661 [stat.ML]]

[15] https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/

[16] S. Li *et al.* [EXO], [arXiv:2303.06311 [hep-ex]]

[17] Developers, T. TensorFlow. (Zenodo,2023,3) https://doi.org/10.5281/zenodo.7764425

[18] Abadi *et al.* TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015) https://www.tensorflow.org/

[19] K. Datta, D. Kar and D. Roy, [arXiv:1806.00433 [physics.data-an]]

[20] S. Alonso-Monsalve and L. H. Whitehead, IEEE Trans. Neural Networks **31** (2020) no.12, 5645-5650 doi:10.1109/TNNLS.2020.2969327 [arXiv:1812.00879 [cs.CV]]

[21] Ian J. Goodfellow, [arXiv:1701.00160 [cs.LG]]

[22] J. H. Collins, Y. Huang, S. Knapen, B. Nachman and D. Whiteson, [arXiv:2210.11489 [hep-ph]]

[23] D. P. Kingma and M. Welling, Found. Trends Mach. Learn. **12** (2019) no.4, 307-392 doi:10.1561/2200000056 [arXiv:1906.02691 [cs.LG]]

[24] K. Dohi, [arXiv:2009.04842 [hep-ph]]

[25] B. Orzari, T. Tomei, M. Pierini, M. Touranakou, J. Duarte, R. Kansal, J. R. Vlimant and D. Gunopulos, [arXiv:2109.15197 [physics.data-an]]

[26] J. H. Collins, [arXiv:2109.10919 [hep-ph]]