Application of Machine Learning Techniques to Separate Resonance Particles From Background Processes

9th-semester report

Submitted

by

SHIVAM RAJ

1711127

Supervised by:

Dr. Prolay Kr. Mal

to the

School of Physical sciences

National Institute of Science Education and Research

Bhubaneswar, India

November 21, 2021

ACKNOWLEDGEMENTS

I would like to express my deep gratitude towards all those people without whom this project would never have been completed. I want to give special appreciation to my supervisor, Dr. Prolay Kr. Mal, to provide me with the opportunity to work on this exciting work. Also, I also want to thank my supervisor for his enthusiasm, support, encouragement, and patience, which help me to complete this project.

Furthermore, I would like to express special gratitude to Prafulla Saha, without whose assistance this work remains uncompleted. I also want to thank Chandiprashad Kar for his assistance and a healthy discussion over the doubts in the course of this project. Throughout this project, these people help me to clarify the problem and make it simpler for me to understand the reasoning behind the processes. I also want to thank other lab members, Aloke Kr. Das, Priyanka Sadangi, Lipsarani Panda, and other members for making a wonderful lab environment.

I also want to thank my friend, parents, and family for the patience and sacrifices they made during all this time.

ABSTRACT

For extracting signals, which are in small cross section compared to the large cross section of the background processes has been helped by the introduction of machine learning (ML) techniques for classification purposes. Classification algorithms in machine learning is a type of supervised learning where the outputs are constrained only to a limited set of values or classes such as signals or backgrounds. A typical example would be the Higgs analysis. In this report, the classification of the produced resonance particles from $p\bar{p}$ collision at Large Hadron Collider(LHC) is presented using machine learning technique trained with deep neural network(DNN). All the neural network training was done using Keras, TensorFlow, and Matplotlib in a Jupyter notebook. We will investigate all the challenges in the application of these novel analysis techniques in this report. This report concludes with a discussion of differences in the outcome of binary and multi-classification using DNN.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The machine learning algorithms, which is general and not task-specific, are modeled towards improving the performance on some given task by training on more and more data[**1-5**]. The training of the machine depends on the past data. The data split in training, validation set and test subsets. The first two are often combined together, where a different chunk of the data is used at each training step to estimate the predictive power of a model. The ultimate goal of the model is a generalization of its ability as to how well it performs over unseen data ot the test data, which can be real or future data. To avoid the problem of overfitting in the model, in ML approximate solutions are preferred: where the goal is to learn all the essential features of the data and further generalize the model[[**4**]].

This project report is based on case studies using simulated data for the CMS detector. Our primary objective is to make such a model search for the rare processes of the resonance particle such as Tprime using machine learning techniques. Tprime(T') is one vector-like quark, which decays into a standard model(SM) top quark and a Higgs boson with decaying into two photons. [**12**].

Compact Muon Solenoid (or CMS) detector located at one of these four collision points of the LHC[[[**15-20**]]. It is designed to observe any new physics phenomena that the LHC might reveal. CMS acts as a giant, high-speed camera, taking 3D "photographs" of particle collisions from all directions up to 40 million times per second. It is 15m in diameter. The central device around which the experiment is built is its magnet, which carries a total magnetic field of 4 Tesla(4T). The Charged particle trajectories after the collision are

measured by the silicon pixel and strip sub-detectors, covering $0 < \phi < 2\pi$ in azimuth and $|\eta| < 2.5$, where the pseudo rapidity η is defined as

$$\eta = -ln[\tan\theta/2],$$

where θ is the polar angle of trajectory of the particle with counterclockwise beam direction. Within the field volume, the silicon detectors are surrounded by a crystal electromagnetic calorimeter and a brass/scintillator hadron calorimeter that provides high resolution energy measurement of photons, electrons, and hadronic jets.

Machine learning has very vast applications in High energy physics(HEP). Earlier, a decision function often used as decision tree. These decision trees also behave like a natural tree-like model to make the decision, starting at the root, further climbing up the branches and then reaching to the leaves, where leaves behave like decision. For classification problem, each leaf represents our decisions to assign the data into classes, whether it is binary or multiclass classifications[**7-8**]. The most widely used machine learning techniques in HEP are boosted decision trees(BDT), XgBoosts, etc ...

Neural networks, also known as artificial neural networks(ANN), are structures inspired by the human brain and also mimic the connectivity of biological signals to one another. The neurons and synapses have been replaced with connected layers of nodes (neurons) and edges. A node takes inputs as real numbers (a weighted sum of the connected outputs from the previous layer) and performs a non-linear transformation to form its output. These non-linear transformation functions are known as the **activation function**. Typical activation functions are: sigmoid (logistic) and tanh where the output is limited below for any input values, and the rectified linear unit ReLU (max(0, x) or the positive part of the argument.

Each neural network consists of at least an input, an output, and one or more hidden layers. This is part of Deep learning. We represent deep NN as DNN. The learning can be supervised depending on pairs of inputs with known outputs for training, or unsupervised, or semi-supervised. A cost or loss function measures the "distance" between the current and the desired outcomes, where our main goal is to make a loss as little as possible to train the model. Classical optimization aims to minimize the cost function on the available (training) data, with or main goal in ML is to generalize, or minimize the cost best, on the unseen or the test data. At each step, the weights for all the edges can be adjusted by backpropagation based on the differentiation chain rule to reduce the cost function by small amounts. This is the stochastic gradient descent (SGD)[**4**].

Here, in this project report, we will discuss all the basic concepts related to machine learning in detail. The outline of this report are as follows: chapter 2 discusses the basic concepts related to machine learning and deep learning(DNN), and further about the simulated data sample used for the separation of signal and background is discussed in chapter 3. We will discuss the results and outputs obtained after DNN training in chapter 4, and finally, conclude this report with the conclusion in chapter 5.

Chapter 2

Machine Learning

> Machine Learning can be defined as the process of inducing intelligence into a system or machine without explicit programming
>
> -Andrew NG

The machine learning (ML) subset of artificial intelligence (AI) allows computer applications to become more accurate by predicting outcomes without being explicitly programmed(not providing commands on each step). The algorithms used by the machines depend on historical or past data as input to predict new outputs. We are surrounded by numerous applications of machine learning in our daily life such as image recognition software's, speech recognition(Voice search, voice dialing), Predictive analytics(Predicting whether a transaction is fraudulent or legitimate, Whether mail is spam or not), etc.

Machine learning is very important nowadays as it provides an overview of trends in customer behavior and operational business patterns, also supports the development of new products. The most common examples can be taken from today's leading tech companies, such as Facebook, Google, and car rental companies, Uber, which use machine learning as a central part of their operations. For handling a large amount of data, machine learning has always been an efficient method. This chapter discusses the basic concept of machine learning and how neural network functions, which are more efficient and productive compared to traditional workflow.[[**25-26**]]

2.1 Types of Machine Learning:

There are four basic approaches for machine learning: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. The type of algorithm that will be used depends on what type of data they want to predict.

2.1.1 Supervised Learning

In this type of machine learning, we supply algorithms with labeled training datasets and define the variables that we want the algorithm to assess for correlations and the outputs. Both the input and the output of the algorithm have been already specified. Supervised machine learning requires getting trained with the algorithm on both labeled inputs and desired outputs. A simple example would be the classification of datasets.

2.1.2 Unsupervised learning:

This type of machine learning involves algorithms that train on unlabeled datasets. The algorithm scans through datasets looking for any meaningful connection without any human intervention. The data on which algorithms train as well as the predict the output is predetermined. For example, unsupervised learning is used for Google news categorization, visual perception tasks such as image recognition, anomaly detection, classifying customers, etc.

2.1.3 Semi-supervised learning:

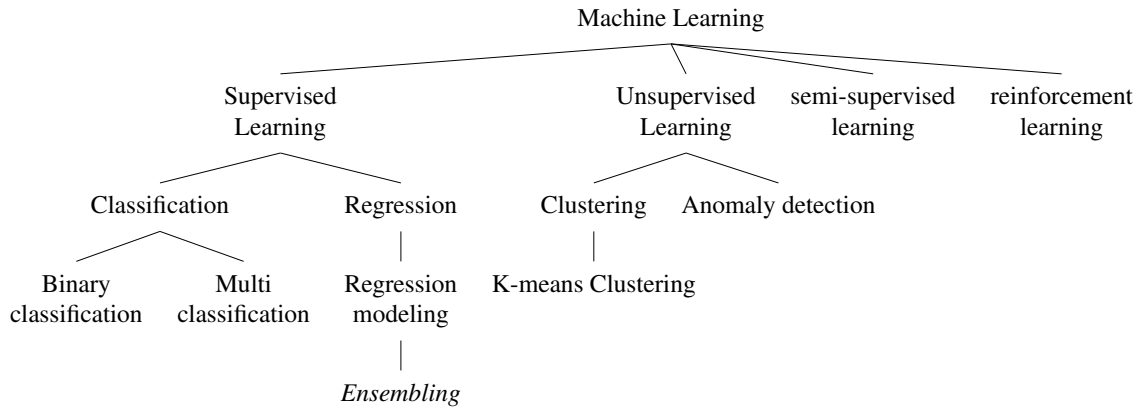
This approach to machine learning involves a mix of both the previous types of the dataset; that is algorithm is trained upon the combination of the labeled and unlabeled dataset. In this type of algorithm, the programmer just needs to cluster similar data using an unsupervised learning algorithm and further use the existing labeled data to label the rest of

the unlabeled data. A few practical examples of this type of learning are speech analysis, internet content classifications, and protein sequence classification.

2.1.4 Reinforcement learning:

Data scientists usually use reinforcement learning to teach a machine to complete a multi-step process(based on the rewarding behaviors/ or pushing the undesired one) for which there is a clearly defined set of rules. In the reinforcement learning decision is dependent on the output of the previous input sequence, so we provide labels to sequences of dependent decisions. For example, chess games or a sitting cat, the cat will only get food when she starts to walk.

A brief summary of how machine learning consists can be summarized from this tree diagram,



2.2 Evaluating models

In machine learning, the ultimate goal is to achieve models that generalize, i.e., that perform well on never-before-seen data, and overfitting is the central obstacle. To do this, splitting the available data is very crucial. Training, validation, and testing are the partitions needed, so during the training phase, the model trains with the training data and test with the validation data, and when the model is ready, it is tested one last time with the test

data, briefly explained by Figure 2.1

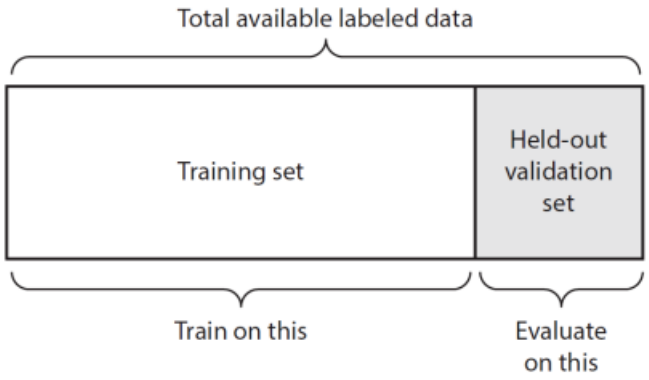


Figure 2.1: Splitting data to be used in the training phase. Image source: [Chollet, 2017]

The reason why three (and not two) datasets are used is that tuning in the model configuration is required. The parameters that can be tuned in a machine learning model are called hyperparameters. Hyperparameter values can be changed to control the learning process. Meanwhile, the value of other parameters, like node weights, are derived by training and cannot be adjusted. Hyperparameter tuning is done with the feedback of the model performance in the validation data and further applied under an unseen test dataset. This is a basic approach for each machine learning model, and we will apply the same over the neural network model in the next section.

2.3 Deep Neural Network

Machine learning works very efficiently on numerous problems but sometimes fails to excel in a few specific cases, which appears very easy for humans. For example, classifying an image as a cat or dog or distinguishing audio clips as a male or female voice,etc... The machine fails to identify this type of problem of image classification or video segregation and other unstructured data types, which are easy for us. There come to the idea of a deep

neural network(DNN), where the idea is to mimic the human brain's biological process, which is composed of billions of neurons connected to each other and to adapt and learn new things[**23-25**].

A simplified version of Deep Neural Network can be represented as a hierarchical (layered) structure of neurons (compared with the neurons in the human brain) with connections to other neurons. These neurons pass a message or signal to other neurons based on the received input and form a complex network system that learns with some feedback mechanism. The following diagram(Figure 2.2) represents an 'N' layered Deep Neural Network.

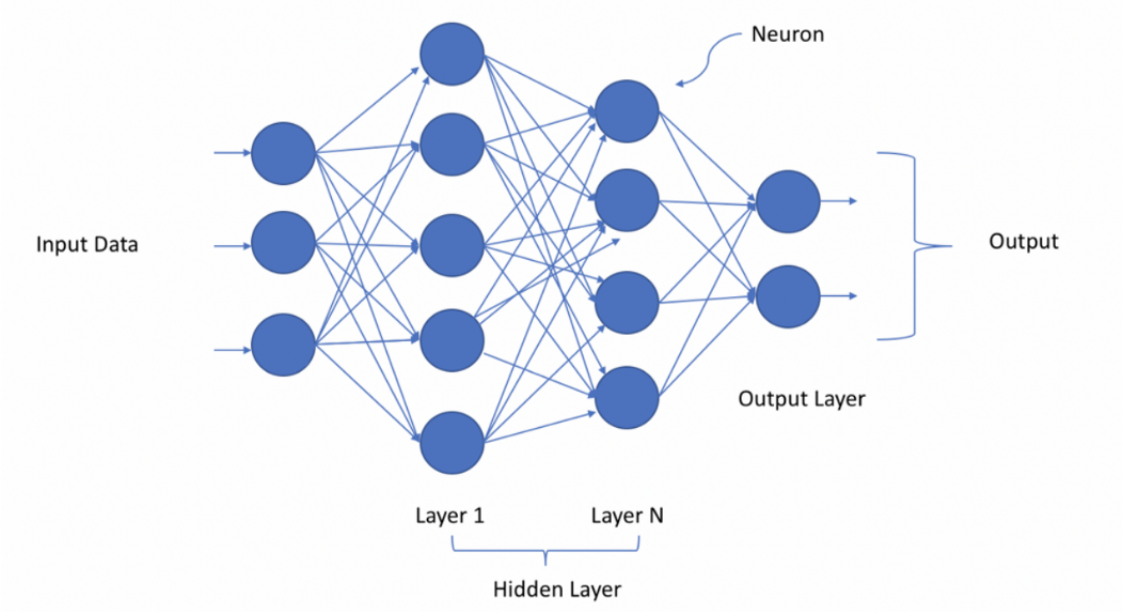


Figure 2.2: A Deep Neural Network with N hidden layers

The input data is provided to the neurons into the first layer (not hidden), which subsequently provides an output to the neurons within the next layer and so on and finally provides the final output. These outputs might be a prediction such as Yes or No (just as we represent in probability). Each layer can consist of one or many neurons, and each of them

will be computed with the help of a small function, i.e., activation function. The activation function takes the signal from the previous layers and passes it further to the next connected neurons. There is a threshold value corresponding to each activation function, where the output is passed when it is above the threshold value; else, it gets ignored. The connection between two neurons of successive layers always passes with an associated weight.

The weights have a very important role to play in the correct prediction from the model. This weight defines the influence of the input on the output for each layer. We provide initial weight to the model randomly, but during the training, these weights get updated iteratively to learn to predict a suitable output.
In a neural network, the initial weights would be provided by us as a random number, but during the model training, these weights are updated iteratively by themselves to learn to predict a correct output. The network also depends on the learning mechanism (optimizer), which helps the neural network to update its weights (that were randomly initialized) to a more suitable weight that aids in the correct prediction of the outcome. To update its weight for the connections, the mathematical algorithm used is called backpropagation. The iteration of the process several times, with more and more data, helps the model to update its weights appropriately. By iterating the process several times, with the help of more data, the networks update the weights appropriately to create a system where the system can take a decision for predicting output based on rules which the model created for itself with the help of weights and connections[**22-23**].

Deep learning is efficient to work with a large amount of data which has made it popular in the last few years; a few of the popular choices for the frameworks of deep learning in python are:-

- Low-level frameworks
  - TensorFlow
  - MxNet
  - PyTorch
- High-level frameworks
  - Keras (uses TensorFlow as a backend)
  - Gluon (uses mxnet as a backend)

Schematically, a neural network(unit, node) layer can be represented as in below Figure 2.3.

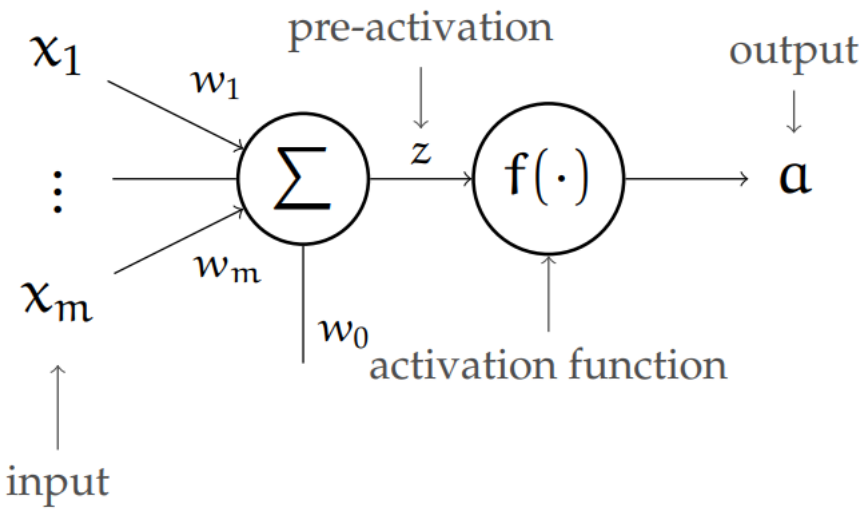


Figure 2.3: Basic structure of a neural network

The function representing the neural network can be expressed as:

$$a = f(z) = f(\sum_{j=1}^{j=m} x_j w_j + w_0) = f(w^T x + w_0) \tag{2.1}$$

A non-linear function of an input vector x $\in \mathbb{R}^m$ note to a single output value a $\in$R . It is parameterized by a vector of weights $(w_1, w_2, ...w_m) \in \mathbb{R}^m$ and an offset or threshold $w_0 \in \mathbb{R}^m$. In order for the neuron to be non-linear, we also specify an activation function $f : \mathbb{R} \to \mathbb{R}$, which can be the f(x)=x(linear function), or can also be any other non linear function, such as ReLU, tanh, etc., which is differentiable.

Before thinking about a whole network,let us consider how to train a single unit.

Given a loss function L(guess, actual) and a dataset $\{(x^{(1)}, y^{(1)}), ..., (x^{(n)}, y^{(n)})\}$, we can do (stochastic) gradient descent, adjusting the weights w, w_0 to minimize the equation

$$J(W, W_0) = \sum_{i=1}^{n} L(f(x^{(i)} : W), y^{(i)}) \tag{2.2}$$

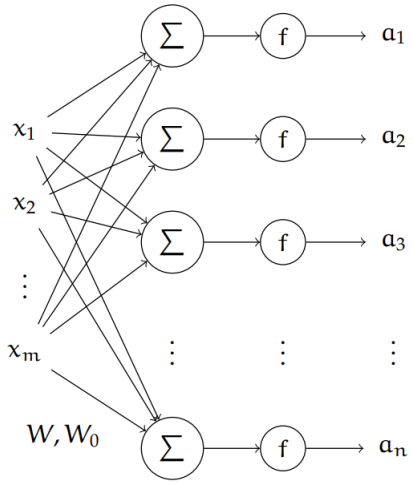
where f is the output of our neural net for a given input.

2.3.1 Networks

Now, we'll try to train the network with stacking multiple neurons together to form a network. A neural network in general takes input $x \in \mathbb{R}^m$ and generates an output $\alpha \in \mathbb{R}^n$. It is constructed with the help of multiple neurons; the inputs of each neuron might be elements of x and/or outputs of other neurons. The outputs are generated by n output units. Here, for the training of data, we will only consider feed-forward networks. In a feed-forward network, we can think of the network as defining a function-call graph that is acyclic. For the simplicity in software and analysis, we usually organize networks into layers. A **layer** can be defined as a group of neurons which are connected to each other parallelly (as in Figure 2.2): The input of a hidden layer depends on the output of previous layer(hidden or input layer); and the output from the layers are input to the neurons in the subsequent next layer. We will start to describe about the model with a single layer and further go on to the case of multiple layers.

Single-layer

A layer is a set of units that, as we have just described, are not connected to each other. The layer is called fully connected if, as in the diagram below, the inputs to each unit in the layer are the same. A layer has input x $\in \mathbb{R}^m$ and output (also known as activation) $\alpha \in \mathbb{R}^n$



Since each unit layer has a vector of weights and a single offset, we can consider the weights of the whole layer as a matrix, W, and the collection of all the offsets as a vector W_0. If we have m inputs, n units, and n outputs, then,

- W is an m $\times$ n matrix
- W_0 is an n$\times$ 1 column vector,
- X, the input, is an m $\times$ 1 column vector,
- Z = W^TX + W_0, the pre-activation, is an n$\times$ 1 column vector,

and the output vector will be

$$A = f(Z) = f(W^T x + W_0)$$

Many layers

A single neural network generally consists of multiple layers, where the output of the previous layer feeds as input to the next layer..

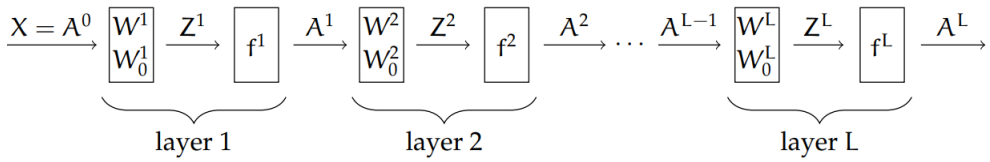
We will use l to name a layer and let m^l be the number of inputs to the layer and n^l be the number of outputs from the layer. Then, W^l and W_0^l are of shape $m^l \times n^l$, respectively. Let f^l be the activation function of layer ℓ . Then, the pre-activation outputs are the $n^l \times 1$ vector, such that,

$$Z^l = W^{l^T} A^{l-1} + W_0^l$$

and the activation function outputs are simply the $n^l \times 1$ vector

$$A^l = f^l(Z^l)$$

We will use this structural diagram to organize our algorithmic thinking and implementation different parameters in deep neural network.



As we saw here how a model of neural network function, the training and output from any model depends on the type of non linear functions Z, i.e. activation function we use in between the layers. Now, the question arises, how many types of activation functions are there?, and how we choose which one will be suitable for our model? We will addresses all these issues over the next section.

2.3.2 Activation function

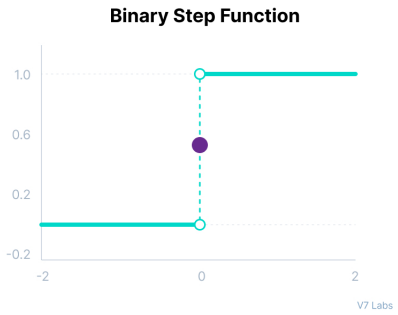
There are three types of neural networks activation functions:

Binary Step Function

Binary step function depends on a threshold value which decides whether a neuron should be activated or not. This can be represented using the equation Equation 2.3

$$f(x) = \begin{cases} 0 for x < 0 \\ 1 for x \geq 0 \end{cases} \tag{2.3}$$

The output of the equation can be represented graphically as,



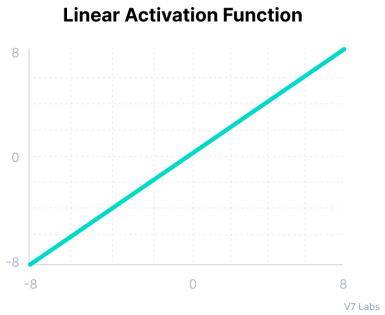
The binary activation function is not always useful; a few of its limitations are:

- We cannot use this activation function to provide us an output of multiclass problems, as it has only two labels of output.
- The gradient of the step function is zero, which causes a restriction in the back propagation process.

Linear Activation Function

Also known as identity Function, i.e. it can be represented using equation, and fig. below,

$$f(x) = x \tag{2.4}$$



However, a linear activation function also has these two major problems :

- It's impossible to use back propagation as the gradient of the function is always a constant and has no relation with the input x.
- After use of linear activation function, without having dependence on a number of layers, the last layer will still be a linear function of the first layer. A linear activation function turns the neural network model into just one layer, which leads to model collapse.

Non-Linear Activation Functions

Non-linear activation functions solve the above limitations possessed by both linear activation functions and binary activation function as the derivative are possible and also related to the inputs, thus backpropogation are allowed here.

Few non-linear activation functions are:-

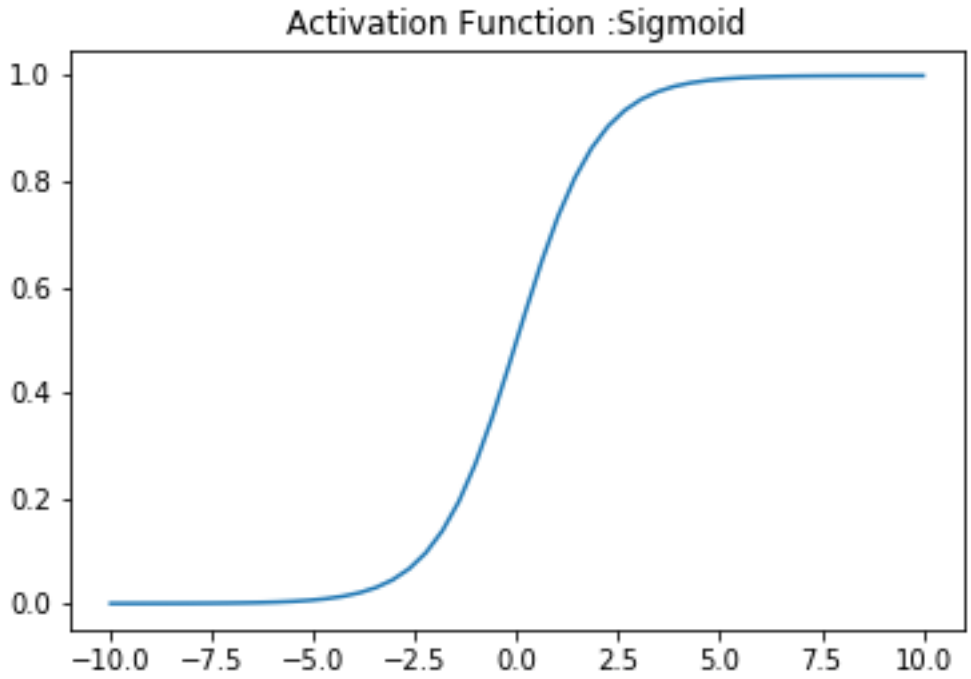
Sigmoid / Logistic Activation Function

Input is any real value(i.e. x $\in \mathbb{R}$)and outputs $\in$ [0,1].

Mathematically, it can be represented as:

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.5}$$

The output of the above equation is,

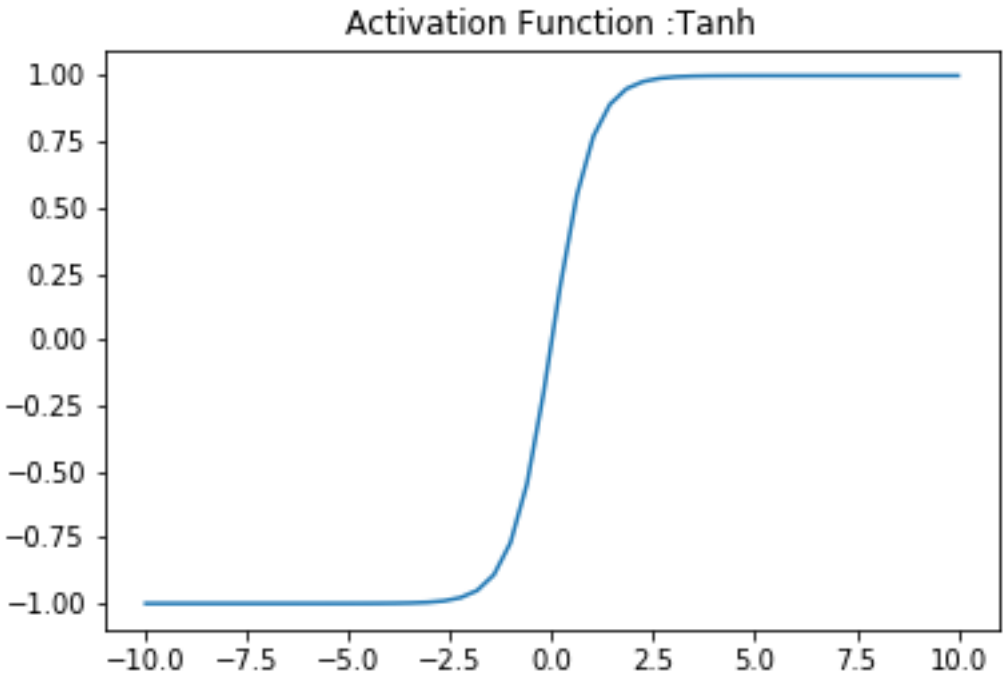


Tanh Function

Tanh function is very similar to the sigmoid/logistic activation function, with only difference in the output range of -1 to 1. Mathematically, it can be represented as;

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.6}$$

And, graphically, it can be represented as,



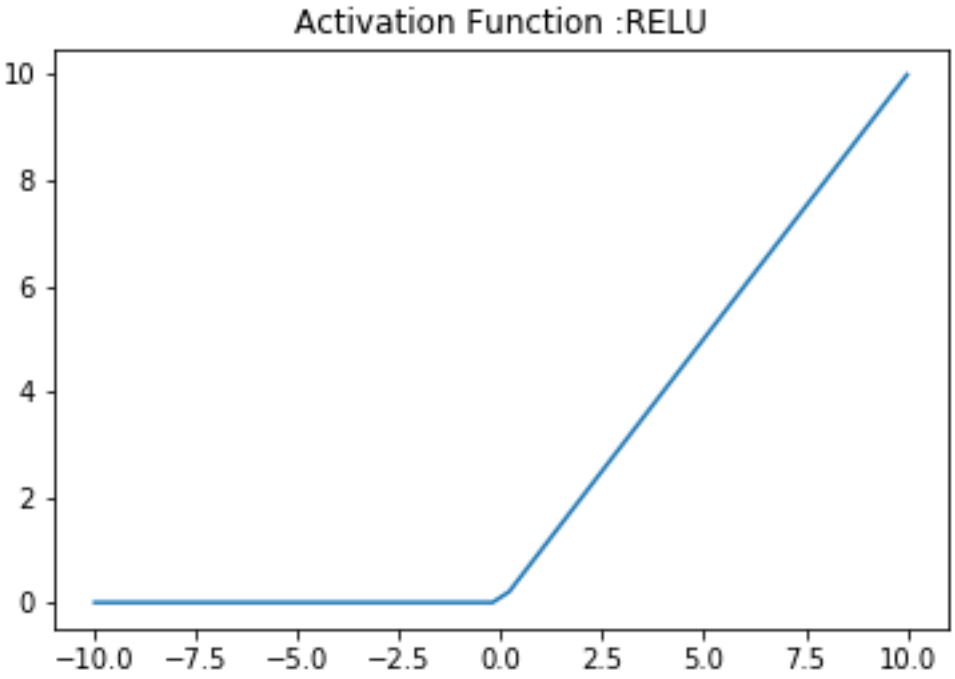
Tanh activation function gives advantages as the output of the tanh activation function is Zero centered; thus, we can easily map the output values as strongly negative, neutral, or strongly positive. Another advantage is that the value of the hidden layers in a neural network has its values between -1 to 1, and the mean for the hidden layer comes out to be 0 or very close to it. Therefore, it helps in centering the data and makes learning for the next layer much easier.

ReLU

ReLU stands for Rectified Linear Unit. ReLU has a derivative function, despite seems like linear function. It allows backpropagation and also simultaneously making it computationally efficient. The ReLU function does not activate all the neurons at the same time. The

neurons get deactivated when its output is less than 0, that is,

$$f(x) = max(0,x) = \begin{cases} 0 & z < 0 \\ z & otherwise \end{cases} \tag{2.7}$$



It is the most commonly used activation function due to following unique features. Since ReLU activation function activate only a certain number of neurons, and it help them to do computation efficiently in comparison to the sigmoid and tanh functions. The ReLU activation function also increase or decrease the rate of convergence towards global minimum of the loss function due to its linear, non saturating property[[**21**]].

Softmax

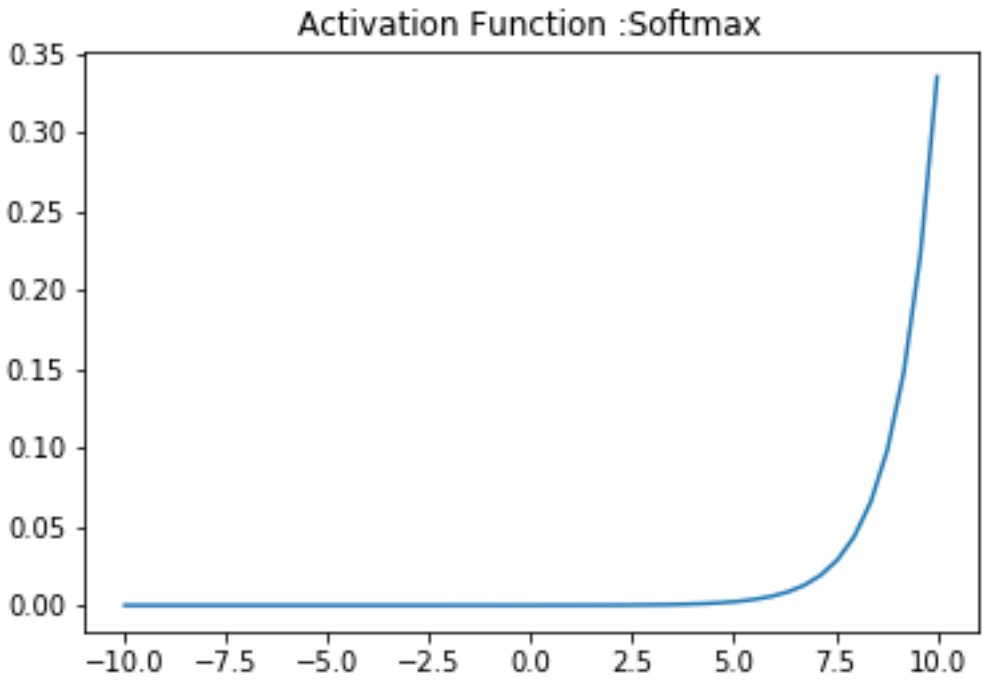
It is used to calculate the relative probabilities. Like sigmoid activation function, the Soft-Max function also returns the probability of each class.

It is the most commonly used activation function for the last or output layer of the neural

network in the case of multi-class classification.

Mathematically it can be represented as:

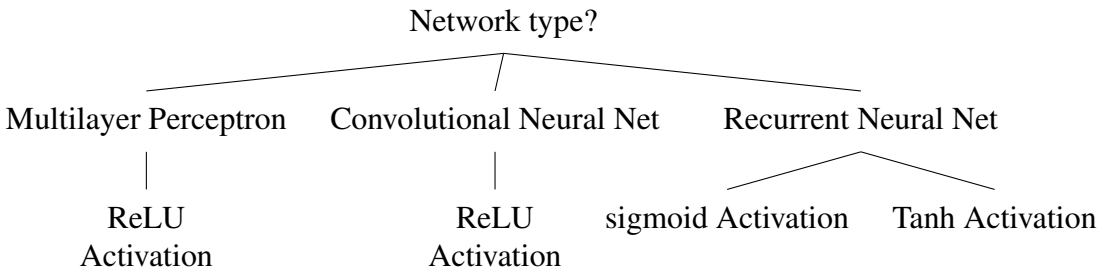
$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{2.8}$$



Here, we saw a few common activation functions, but how we can choose activation function for the given problem?

How to choose activation function for Hidden Layers

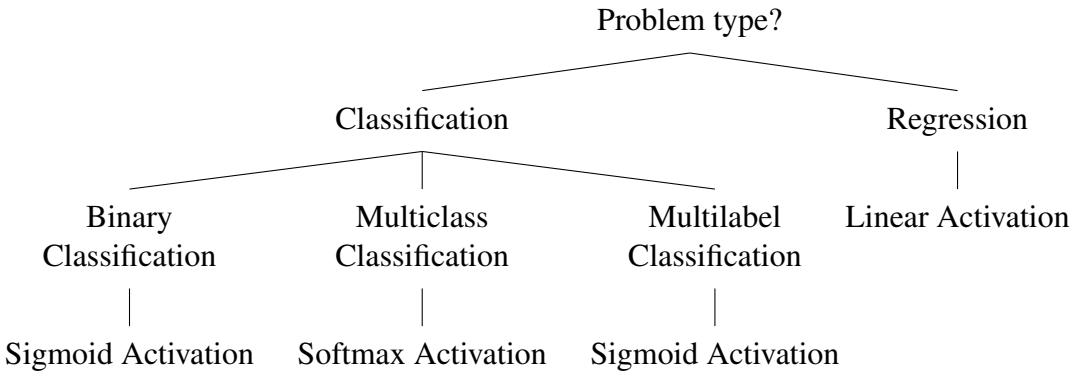
The choice of activation function in between the layers depends on the type of problem we have, as it is summarized in the tree below.



How to choose activation function for output layers

The activation function used in the final layer of the neural network depends on what type of output we need and on the type of prediction problem that we are solving, which can be summarized from the tree below.

- **Regression**: one node, Linear Activation
- **Binary Classification**:one node per class, Softmax Activation
- **Multiclass Classification**:One node per class, softmax activation
- **Multilabel Classification**: One node per class, sigmoid activation



2.4 Training of Neural Network

2.4.1 Error backpropagation

We will train our neural networks using gradient descent methods. It's possible to use batch gradient descent, in which we sum up the gradient over all the points or stochastic gradient descent (SGD), where we take a small step with respect to the gradient after considering a single point at a time[[**23**]].

we will always compute the gradient of the loss function with respect to the weights for a particular value of (x, y). That tells us how much change is needed in the weights, in order to reduce the loss experienced on this particular training. Let us understand with this example.

First, let's us calculate and observe how the loss depends on the weights in the final layer, W^L remembering that our output is A^L, and using the shorthand loss to stand for Loss((f(x; W), y) which is equal to Loss(A^L, y), and finally that $A^L = f^L(Z^L)$ and $Z^L = W^{L^T} A^{L-1}$, we can apply the chain rule as:

$$\frac{\partial \text{loss}}{\partial W^L} = \underbrace{\frac{\partial \text{loss}}{\partial A^L}}_{\text{depends on loss function}} \cdot \underbrace{\frac{\partial A^L}{\partial Z^L}}_{f^{L\prime}} \cdot \underbrace{\frac{\partial Z^L}{\partial W^L}}_{A^{L-1}}$$

Here, we need to be little careful with the dimensions, and here we can note that it is true for any ℓ, including $\ell = L$

$$\underbrace{\frac{\partial \text{loss}}{\partial W^l}}_{m^l \times n^l} = \underbrace{A^{l-1}}_{m^l \times 1} \underbrace{\left(\frac{\partial \text{loss}}{\partial Z^l}\right)^T}_{1 \times n^l}$$

If we can repeatedly apply the chain rule, we will obtain this expression for the gradient of the loss with respect to the pre-activation function in the first layer:

$$\frac{\partial \text{loss}}{\partial Z^1} = \underbrace{\underbrace{\frac{\partial \text{loss}}{\partial A^L} \cdot \frac{\partial A^L}{\partial Z^L} \cdot \frac{\partial Z^L}{\partial A^{L-1}} \cdot \frac{\partial A^{L-1}}{\partial Z^{L-1}} \cdot \cdots \cdot \frac{\partial A^2}{\partial Z^2}}_{\partial \text{loss}/\partial Z^2} \cdot \frac{\partial Z^2}{\partial A^1}}_{\partial \text{loss}/\partial A^1} \cdot \frac{\partial A^1}{\partial Z^1}$$

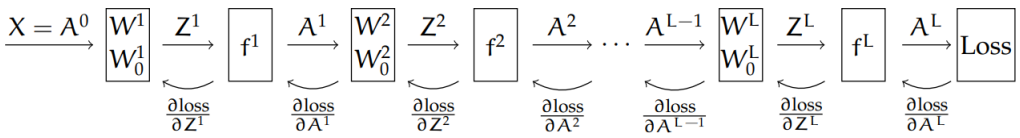
Here,

- $\frac{\partial loss}{\partial A^L}$ is $n^L \times 1$ and depends on the particular loss function you are using.
- $\frac{\partial Z^l}{\partial A^{l-1}}$ is $m^L \times n^L$ and is just W^l
- $\frac{\partial A^l}{\partial Z^l}$ is $n^L \times n^L$. Each element $\alpha_i^l = f^l(z_i^l)$. This means that $\frac{\partial \alpha_i^l}{\partial z_j^l} = 0$ whenever i$\neq$ j. So, the off-diagonal elements of $\frac{\partial A^l}{\partial Z^l}$ are all 0, and the diagonal elements are $\frac{\partial \alpha_i^l}{\partial z_j^l} = f^{l'(z_j^l)}$

We can write the above equation as,

$$\frac{\partial \text{loss}}{\partial Z^l} = \frac{\partial A^l}{\partial Z^l} \cdot W^{l+1} \cdot \frac{\partial A^{l+1}}{\partial Z^{l+1}} \cdot \ldots W^{L-1} \cdot \frac{\partial A^{L-1}}{\partial Z^{L-1}} \cdot W^L \cdot \frac{\partial A^L}{\partial Z^L} \cdot \frac{\partial \text{loss}}{A^L}$$

This general process is called error back-propagation. The general idea is that we will do a forward pass to compute all the α and z values at all the layers. Then, we can start work backward direction and compute the gradient of the loss with respect to the weights in every layer, starting at last layer L and going back to layer 1, in this way model can update its weight, as can be shown below,



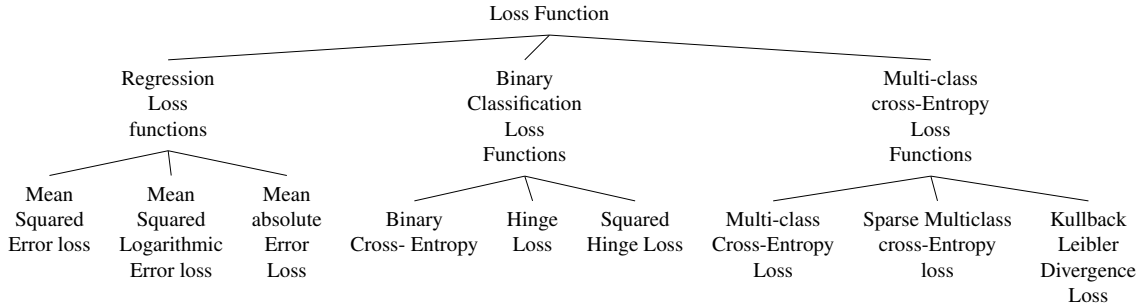
Now, How we can do stochastic gradient descent training on a feed-forward neural network. This is the pseudo code to apply stochastic gradient descent,

```
def SGD(f, theta0, alpha, num_iters):
    """
      Arguments:
      f -- the function to optimize, it takes a single argument
           and yield two outputs, a cost and the gradient
           with respect to the arguments
      theta0 -- the initial point to start SGD from
      num_iters -- total iterations to run SGD for
      Return:
      theta -- the parameter value after SGD finishes
    """
    start_iter = 0
    theta = theta0
    for iter in xrange(start_iter + 1, num_iters + 1):
        _, grad = f(theta)

        # there is NO dot product ! return theta
        theta = theta - (alpha * grad)
```

2.5 Loss functions

Now, the choice of a loss function for a particular problem is a very tedious task. We can take a rough idea about the loss function from this tree,



Like activation function, loss function also take different assumptions about the range of inputs it will take depending on the type of problem in hand. While designing the model of neural network, it become important to make things to fit well together. As, in particular, we will think about matching loss function with activation function of the last layer, f^L. This hypothesis can be infer from the table below, Table 2.1.

There are also other loss function we have used to make our model better, such as "Bi-

Loss	f^L	*Comments*
squared	Linear	–
hinge	Linear	for "maximum-margin" classification
NLL	Sigmoid	negative log likelihood loss. Useful to train classification problem with C classes.
NLLM	softmax	–

Table 2.1: Few loss functions corresponding to the last layer/ output layer activation function. There are many different types of the loss function following the kind of problem, whether it belongs to classification or regressions.

naryCrossentropy", "CategoricalCrossentropy", and "SparseCategoricalCrossentropy" belonging to probabilistic losses. Also, "mean_squared_error", "mean_absolute_error", and "MeanSquaredError" belonging to regression losses.

Two-class classification and log likelihood

For binary classification problems, which are useful in separation of signal and background, Hinge loss gives us another way, to make a smoother objective, penalizing the margins of the labeled points relative to the separator. The hinge loss is defined to be

$$L_h(guess, actual) = max(1 - guess.actual, 0)$$

when actual $\in$ {+1, -1} Using hinge loss, together with a squarednorm regularizer, forces the learning process to try and find a separator that has the maximum margin relative to the data set. This optimization set-up is called a **support vector machine**. It was popular because it has a quadratic form that makes SVM to get easily optimized.

Multi-class classification and log likelihood

Multi-class classification with total of K classes, where the training label is represented with the one-hot vector $y = [y_1, ..., y_k]^T$, where y_k= 1 if the example is of class k.Assume that our network uses softmax as the activation function (Which is most commomly used activation

function for multi classification) in the last layer, so that the output is $\alpha = [\alpha_1, ..., \alpha_k]^T$, which represents a probability distribution over the K possible classes. Then, the probability that our network predicts the correct class for this example is $\Pi_{k=1}^{N} \alpha_k^y$ and the log of the probability that it is correct is $\sum_{k=1}^{k} y_k \log \alpha_k$,so

$$L(guess, actual) = -\sum_{k=1}^{K} actual_k.log(guess_k)$$

2.6 Optimizing neural network parameters

As neural networks consists of many parameters, our ultimate goal to minimize the loss function. The optimization can be done with help of standard gradient-descent softwares, but here, we can take advantages of the structure of the loss functions to improve optimization. The structure of loss function as a sum over terms, one training data point, help us to consider stochastic gradient methods. In this section, we will try to consider some alternative strategies for organizing training, and also to make easier to handle the step-size parameter.

2.6.1 Batches

Lets us assume we have an objective function of the form

$$J(W) = \sum_{i=1}^{n} L(h(x^{(i)}; W), y^{(i)})$$

where h is the function computed by a neural network, and W stands for all the weight matrices and vectors in the network.

When we perform batch gradient descent, we use the update rule

$$W := W - \eta \nabla_W J(W),$$

which is equivalent to

$$W := W - \eta \nabla_W \sum_{i=1}^{n} L(h(x^{(i)}; W), y^{(i)})$$

Thefore, we add up the gradient of loss after each training point, with respect to W, and then take a step in the negative direction of the gradient to minimize the loss.

A more effective strategy for optimization is to take "average" between batch and stochastic gradient descent by using mini-batches. For a mini-batch of size k, we select k distinct data points uniformly at random from the data set and do the weight update based just on their contributions to the gradient.

$$W \leftarrow W - \eta \nabla_W \sum_{i=1}^{n} L(h(x^{(i)}; W), y^{(i)})$$

Most neural network software packages are set up to do mini-batches[**25**].

To select k unique data points at random from a large dataset is computationally difficult. An alternative strategy, if we have an efficient procedure for randomly shuffling the data set (or randomly shuffling a list of indices into the data set) is to operate in a loop, roughly as follows:

then, we can easily divide the dataset into k mini batches.

Algorithm 1 *MINI-BATCH-SGD(NN, data, k)*

$n = length(data)$
while not done: **do**
  $RANDOM - SHUFFLE(data)$
  for i=1 to $\frac{n}{k}$ **do**
    $BATCH - GRADIENT - UPDATE(NN, data[(i-1)k : ik])$

2.6.2 Adaptive stepsize

Picking the value for η is difficult and time-consuming. As our networks become deep (with increase in the numbers of layers) we can find that magnitude of the gradient of the loss with respect the weights in the last layer,$\frac{\partial loss}{\partial W_L}$,may have significant differences from the gradient of the loss with respect to the weights in the first layer $\frac{\partial loss}{\partial W_1}$.

The output gradient is further multiplied by all the weight matrices of the network and is "fed back" through all the derivatives of the activation functions. This can lead to a general problem of **exploding or vanishing gradients**, in which the back-propagated gradient is either too big or small.

Running averages

It is a computation strategy for estimating a weighted average for a sequence of data. Let us take data sequence be $\alpha_1, \alpha_2, ...$; then we define a sequence of running average values, $A_0, A_1, A_2, ...$ using the equations

$$A_0 = 0$$

$$A_t = \gamma_t A_{t-1} + (1-\gamma_t)\alpha_t$$

where $\gamma_t \in (0,1)$. If γ_t is a constant, then this is a moving average, in which

$$A_T = \gamma A_{T-1} + (1-\gamma)\alpha_T$$

$$= \gamma(\gamma A_{T-2} + (1-\gamma)\alpha_{T-1}) + (1-\gamma)\alpha_T$$

$$= \sum_{t=0}^{T} \gamma^{T-t}(1-\gamma)\alpha_t$$

So, you can see that inputs α_t closer to the end of the sequence have more effect on A_t than early inputs.

Momentum

we can use this method a bit like running averages to describe strategies for computing η. The simplest method is momentum, in which we try to "average" recent gradient updates so that if they have been bouncing oscillating in some direction, then we at least take out that component of the motion. For momentum, we have

$$V_0 = 0$$

$$V_t = \gamma V_{t-1} + \eta \nabla_W J(W_{t-1})$$

$$W_t = W_{t-1} - V_t$$

This does not look like an adaptive step-size method. But, if we let $\eta = \eta'(1-\gamma)$ then the rule looks exactly like doing an update with step size η' on a moving average of the gradients with parameter γ:

$$M_0 = 0$$

$$M_t = \gamma M_{t-1} + (1-\gamma)\nabla_W J(W_{t-1})$$

$$W_t = W_{t-1} - \eta' M_t$$

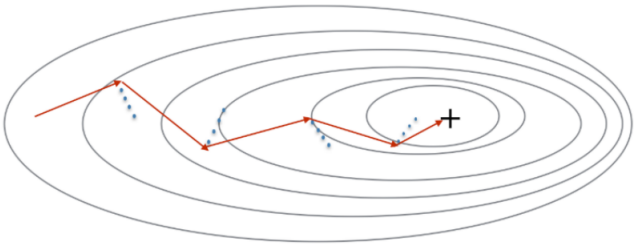


Figure 2.4: The red arrows show the change in the value of weight after one step of mini-batch gradient descent with use of momentum. The blue points show about the direction of the gradient with respect to the mini-batch at each step. The Momentum smooths the path taken towards the local minimum and further leads to faster convergence.[**21**],[**25**]

Adadelta

Here the idea is to take larger steps in parts of the space where J(W) get nearly flat (as there are no risks of taking too larger step due to the gradient being large) and smaller steps when it is steep. We'll apply this idea to each weight, and at the end we obtain a method called adadelta. Even, here our weight are indexed by layers including input and output unit, just let W_j be any weight in the network

$$g_{t,j} = \nabla_W J(W_{t-1})_j$$

$$G_{t,j} = \gamma G_{t-1,j} + (1-\gamma) g_{t,j}^2$$

$$W_{t,j} = W_{t-1,j} - \frac{\eta}{\sqrt{G_{t,j} + \varepsilon}} g_{t,j}$$

Adam

Adam has become the most common and default method of managing step sizes neural networks. The moving averages of the gradient and squared gradient, which estimates the mean and variance of the gradient for weight j:

$$g_{t,j} = \nabla_W J(W_{t-1})_j$$

$$m_{t,j} = B_1 m_{t-1,j} + (1-B_1) g_{t,j}$$

$$v_{t,j} = B_2 v_{t-1,j} + (1-B_2) g_{t,j}^2$$

If we initialize m_0 = v_0 = 0, then there will always be bias(slightly too small). So we will correct the bias by defining,

$$\hat{m}_{t,j} = \frac{m_{t,j}}{1-B_1^t}$$

$$\hat{v}_{t,j} = \frac{v_{t,j}}{1-B_2^t}$$

$$W_{t,j} = W_{t-1,j} - \frac{\eta}{\sqrt{\hat{v}_{t,j} + \varepsilon}} \hat{m}_{t,j}$$

Adam did not have a huge effect on the result after making small changes in the model, which makes it an efficient method.

2.7 Regularization

Till now, we have only discussed how we can optimize loss on our training data. As discussed before, the risk of overfitting is still persistent. This overfitting problem can be rectified with the help of increasing our data size, which is the case nowadays where the deep neural network uses large data size. Nonetheless, there are several strategies for regularizing a neural network, and sometimes they can be important. This can be done using the implementation of early stopping, where the idea is train on the training set, and on every epochs, (pass through the whole training set, or possibly more frequently), evaluate the loss of the current W on a validation set. Here, it is observed that the loss on the training set goes down fairly consistently with each number of iteration, and the loss on the validation set will initially decrease, but then begin to increase again. Once we observe that the validation loss is systematically increasing, we can stop training the model and return the weights that had the lowest validation error.

Another simple method is to penalize the norm of all the weights,This method is known as weight decay,

$$J(W) = \sum_{i=1}^{n} Loss(NN(x^{(i)}), y^{(i)}; W) + \lambda ||W||^2$$

we end up with an update of the form

$$W_t = W_{t-1}(1 - \lambda_\eta) - \eta(\nabla_W Loss(NN(x^{(i)}), y^{(i)}; W_{t-1}))$$

This rule has the form of first "decaying" W_{t-1} by a factor of (1 -$\lambda\eta$) and then taking a gradient step.

Other few methods are:-

2.7.1 Dropout

The contains a simple idea, it suggest rather than perturbing the data every time we train, we will make changes in the network. Here, we will randomly, from each dataset, after selecting a unit layer prohibit them from participating in the training. Therefore, all of the units will have to take a kind of overall responsibility for getting the answer right, and will not be able to rely on any small subset of the weights to do all the necessary computation. This tends also to make the network more robust to data perturbations.
When we are done training and want to use the network to make predictions, we multiply all weights by p to achieve the same average activation levels. In the forward pass during training, we let

$$\alpha^l = f(z^l) * d^l \tag{2.9}$$

where * denotes component-wise product and d^l is a vector of 0's and 1's drawn randomly with probability p. The backwards pass depends on α^l so we do not need to make any further changes to the algorithm. Another modern alternative method to dropout, is batch normalization.

2.7.2 Batch Normalization

Here, idea is when training is done with mini-batches, the idea is to standardize the input values for each mini-batch, subtracting off the mean and dividing by the standard deviation of each input dimension. This gives us similar effect to adding noise and dropout. Each mini-batch of data ends up getting mildly perturbed, which prevents the network from exploiting very particular values of the data points.

2.8 ROC Curve

Another metric tom measure the output is the Receiver Operating Characteristic (ROC) curve output after training, defined with respect to a given class *C*. Given a point x and model that outputs a P(C|x) probability that x belongs to the class C. Given T , a threshold, x belongs to C if and only if P(C|x) $\geq$ T . If T = 1, a point is labeled as belonging to class, C only if the model is 100% sure. If T = 0, every point is labeled as belonging to the class C

Each value of the threshold T generates a point (False Positive, True Positive) and, then, the ROC curve is the curve formed by going through T = 0 to T = 1. A good model will have a curve that increases quickly from 0 to 1, the different ROC curve depending over output can be seen in Figure 2.5

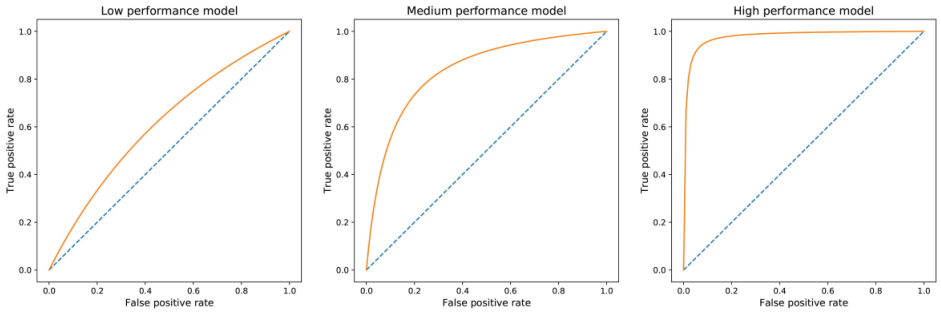


Figure 2.5: ROC curves depending on the effectiveness of the model

Chapter 3

Signal and Background

What are signal and background? The signal corresponds to the features, process, or whatever events we are interested in studying. This signal could correspond to the production of a Higgs, or maybe of a pair of W bosons, or maybe it just corresponds to events with two high energy jets or events in them. While the backgrounds are the events that may seem like signals but we are not interested in.

We took s simulated sample of the resonance particles of the top quark; Tprime, and other t-quark particle such as $\bar{t}tH$, thq, and ttgg for the training and testing of the deep neural network model. The associated production of top quarks with the Higgs boson, either in pairs ($\bar{t}tH$) or singly (tH), provides direct experimental access to the top-Higgs coupling. The $\bar{t}tH(tH)$ production mode, while proceeding at a rate of about 100(1000) times smaller than gluon fusion, bears a highly distinctive experimental signature, which includes leptons and/or jets from the decay of the two (single) top quarks as shown in Figure 3.1.

With the increase of the LHC energy from 8 to 13 TeV for Run 2, the ttH production cross section is expected to increase by a factor four. However, such process still remains rare, and searches for ttH production have been driven by the higher sensitivity achieved in Higgs decay modes with larger branching fractions.[[**14**]]

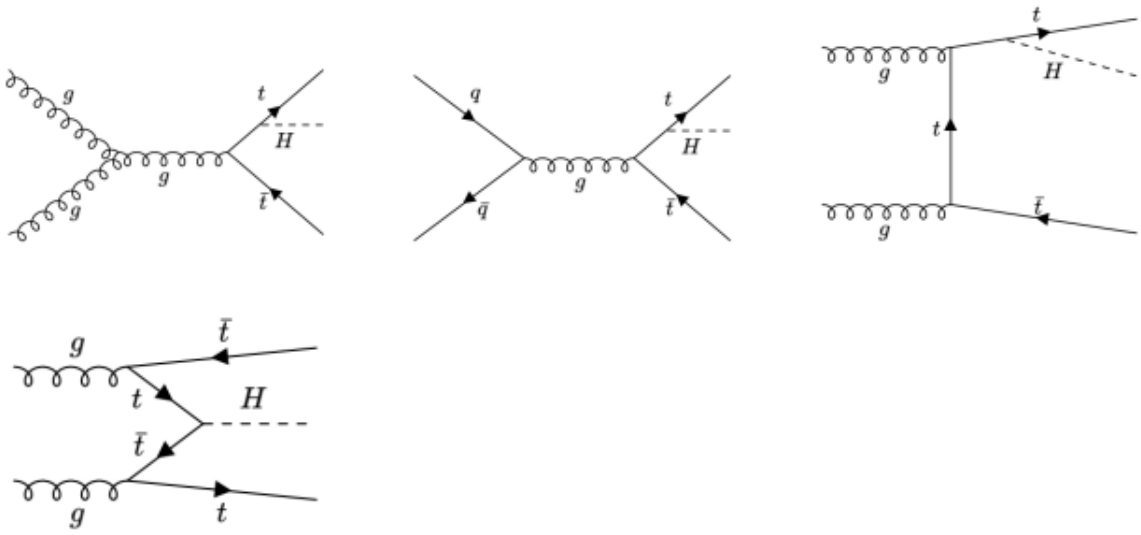


Figure 3.1: Feynman diagrams depicting $t\bar{t}H$ production modes

As the standard model(SM) is complete as a low-energy effective theory describing all known fundamental particles and their interactions. But there are also a few problems that SM cannot explain. There are various theories of new physics beyond the SM that predict the additional particles that can affect the quantum corrections to the Higgs boson mass and answer question, which is known as "hierarchy problem". One of the proposed promising new states is vector-like quarks.[**12**]

Vector-like quarks (VLQs) are hypothetical spin-1/2 coloured particles, labeled $T^{'}$ and B', with electric charges of +2e/3 and -1e/3, respectively. Their left-handed and right-handed components transform in the same way under the Standard Model gauge group. A vector-like top quark partner T' has two production modes. One is pair production through the strong interaction while the other is single production mode through the electroweak interaction[**12**]. The T' quark could couple to bW, tZ, or tH, resulting in the corresponding T' quark decays as shown with Feynman diagram in the Figure 3.2. We will use this VLQ as the signal.

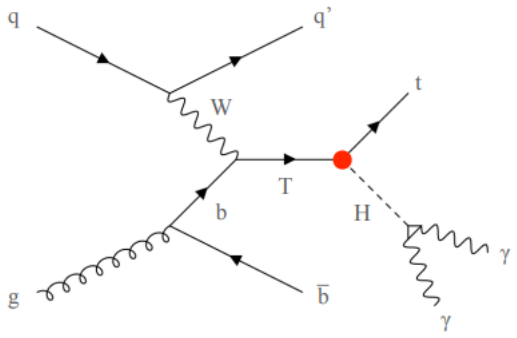


Figure 3.2: Leading-order Feynman diagram for single T' production

For the another two ntuples of thq and ttgg, the Feynman diagrams are plotted in the Figure 3.3, Figure 3.4, and Figure 3.5 respectively.

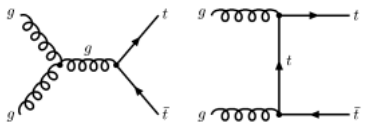


Figure 3.3: ttgg Feynman diagram

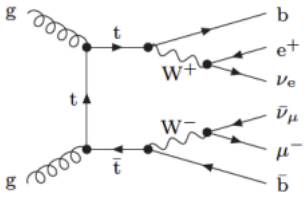


Figure 3.4: Feynman diagram for ttgg

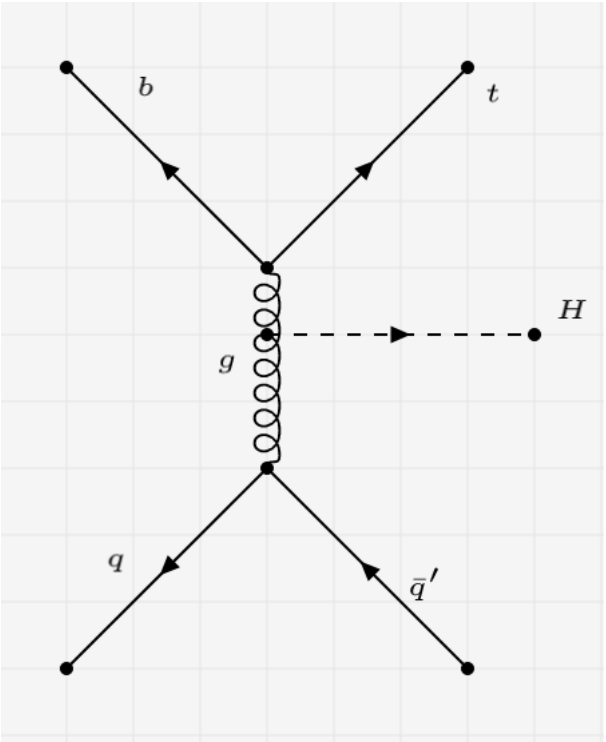


Figure 3.5: Feynman diagram of thq

Few of the plots of variables of these simulated samples are shown below,

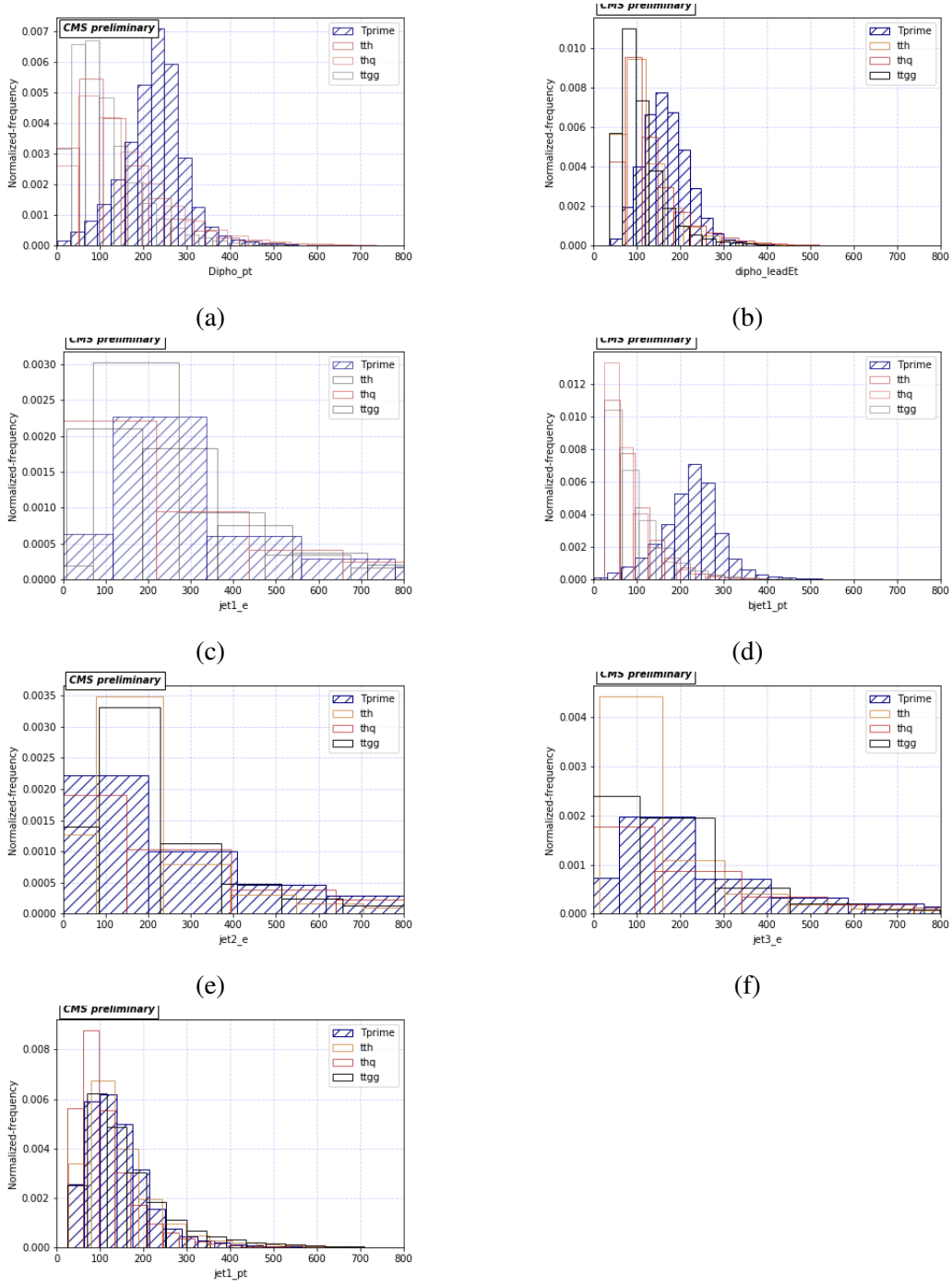


Figure 3.6: Simulated sample plot for different variables, for each figure plotted above the signal is Tprime and the background is tth, thq, and ttgg. From top to bottom, plots for different variable are as, (a) Plot for $Dipho\_P_T$, (b) Plot for $Dipho\_leadEt$, (c)Plot for $jet1\_e$, (d) Plot for $bjet1\_P_T$, (e) Plot for $jet2\_e$, (f) Plot for $jet3\_e$, and (g) Plot for $jet1\_P_T$

In the given Figure 3.6, we plotted a representation for a few variables. from this figure, we can see how our signal (Tprime) cannot get clearly separated from our background($tt\gamma\gamma$, $\bar{t}th$, and thq. Our main motivation is to make a better separation of the Tprime, signal in this case from the backgrounds i.e., $tt\gamma\gamma$, $\bar{t}th$, and thq.

Chapter 4

Results and Discussion

In the previous section, from Figure 3.6, we observed that the separation of the signal and the background events for different datasets were not clearly visible. This is a basic problem of classifications. To separate and make a proper visualization of the events as signal or background; there are several machine learning methods used traditionally, such as Boosted Decision Trees(BDTs), XGBoosts, and many other TMVA based machine learning methods. To separate signal and background, python based modules such as Keras, which is based on the high-level wrapper for the machine learning frameworks Theano (www.deeplearning.net/software/theano/) and TensorFlow (www.tensorflow.org), they were mainly used to set up deep neural networks.[**29**] [**30**] [http://tmva.sourceforge.net]

4.1 Uses of DNN

The Deep Neural network(DNN) [1] has been applied for training over the ntuple of tth, thq, ttgg and Tprime. The dataset was converted into the array using modules root2numpy(https://scikit-hep.org/root_numpy/), and further randomize it using rec2array. By using Pandas DataFrame we converted the array and further feed it into the model after splitting them as train and test data sets. The basic working model of the neural network is shown in the Figure 4.1.

[1] All the code related to this work can be found here: https://github.com/raj2022/M.Sc.-thesis/tree/main/Codes

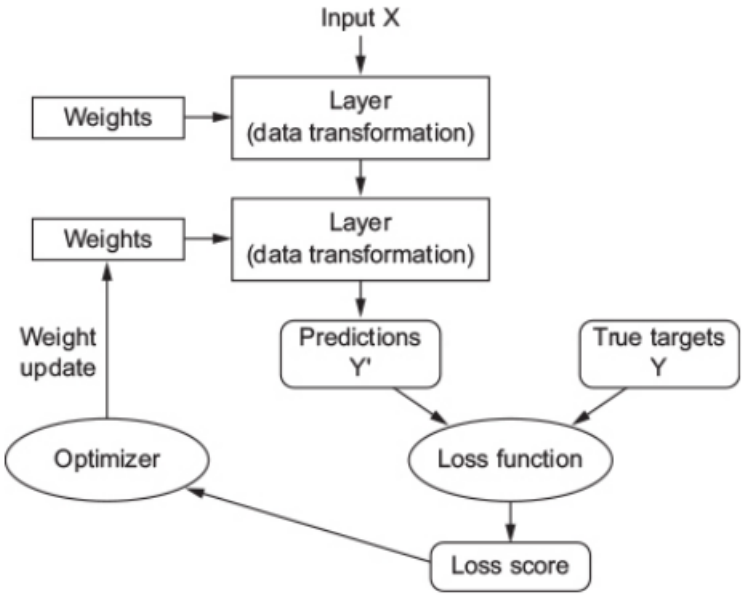


Figure 4.1: Neural network scheme. Image source: [Chollet, 2017]

4.2 DNN model

Deep neural network (DNN) is used here for training the data and further testing the model. Keras module, Sequential used to make the DNN model. The basic architecture of our model are as follow, given in Table 4.1. Our model was made to train total 5,33,457 trainable parameters,and we also have total number of datasets around this number. The model was given total of 29 different input features(Table 4.2) in the first layer. To make neural layer more deep and to get a better training output, there were 4 extra hidden layers consisting of 200, 100, 100 and 100 nodes were added to the input layer. The output of successive layers goes as input to the next layer after passing through the ReLU activation function(as shown in Figure 2.3). For each layer, we have used dropout layer to save our model from collapsing. For each 3 input model will drop a value, the model architecture is shown in the Figure 4.2.

For each input layers, we have provided weight or random state as 5, which will be getting updated after each training. The rationale behind choose these specific parameters such as the number of nodes, layers, epochs is to improve our training accuracy, and it was completely based on hit and trail method.

The output layer consist only a single node for binary classification and again changed for multi classification, depending on the number of outputs we required as the outputs. In this semester project report, the output will only behave either as signal or as background. The output layer also depend on whether the output required in binary or multiclass. For binary classification, the common activation function used is 'sigmoid', which we have already discussed in subsection 2.3.2 The model was compile with the binary cross entropy loss function & categorical_crossentropy loss function depending on the type of classifications. To avoid the model collapsing issues, we have used ADAM optimizer in the model and to prevent the training from over-fitting, the model is trained with accuracy matrices.

The model was fitted with the given dataset by splitting it into training and testing dataset in the ratio of 80:20, that is 80% of the data used for training and 20% of the data used for testing the model. The model has also provided with validation dataset, which is sample of dataset used to provide an unbiased evaluation of a model fit on training dataset while having model hyper-parameter(discussed in Table 4.1. Here the validation dataset are one-fourth of training data. This model was trained in the batch of 900, which means for every training, it would divide the dataset into 900 mini batches and run through all over the training data. The number of epochs of the model is 100, it employs the model has run through 100 times in total, after each run, model update or optimize its weight/ random_state by itself depending on the output of the error/loss and other hyper parameter. The model was further tested on the given testing dataset.

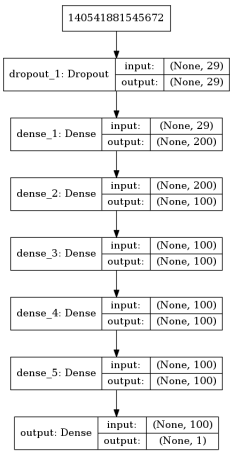


Figure 4.2: Basic architecture of our DNN Model

Options		Description
Model	sequential	-
Number of Inputs	29	as given in Table 4.2
Number of layers (Input)	200	$Dense_1$
Hidden	200	$Dense_2$
Hidden	100	$Dense_3$
Hidden	200	$Dense_4$
Hidden	200	$Dense_5$
Output	1	$Dense_6$
Activation function(Hidden Layer)	ReLU	Same for both binary classification and multiclassification
Activation layer(Output)	sigmoid	For binary classification
	Softmax	For multi classification
Loss function	Binary cross entropy	For binary classification
	categorical_crossentropy	For multi classification
Optimizer	ADAM	-
Matrices	Accuracy	
BatchSize	900	Batch size used for a single gradient step during training
NumEpochs	100	Number of training epochs
Verbose	1	Verbosity during training

Table 4.1: Configurations of the Keras model used for training and testing purposes

4.3 Correlation between signal and background variable

A correlation function represents a measurement between the strength of a linear relationship between two quantitative variables. There are two types of correlation, positive correlation and negative correlation. Positive correlation represents a relationship between two variables in which both variables move in the same direction. This means when one variable increases while the other also increases and vice versa While negative correlation is a relationship where one variable increases as the other decreases, and vice versa. A correlation of 1 or +1 shows a perfect positive correlation, which means both the variables move in the same direction. A correlation of -1 shows a perfect negative correlation, which means as one variable goes down, the other goes up [**27**]. Here for the comparison of the variables (given in Table 4.2) from two different datasets were calculated using Pearson Correlation Coefficient Formula, which is,

$$r = \frac{n(\sum xy) - \sum x \sum y}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \tag{4.1}$$

Where,

n = Quantity of Information

$\sum x$ = Total of the First Variable Value

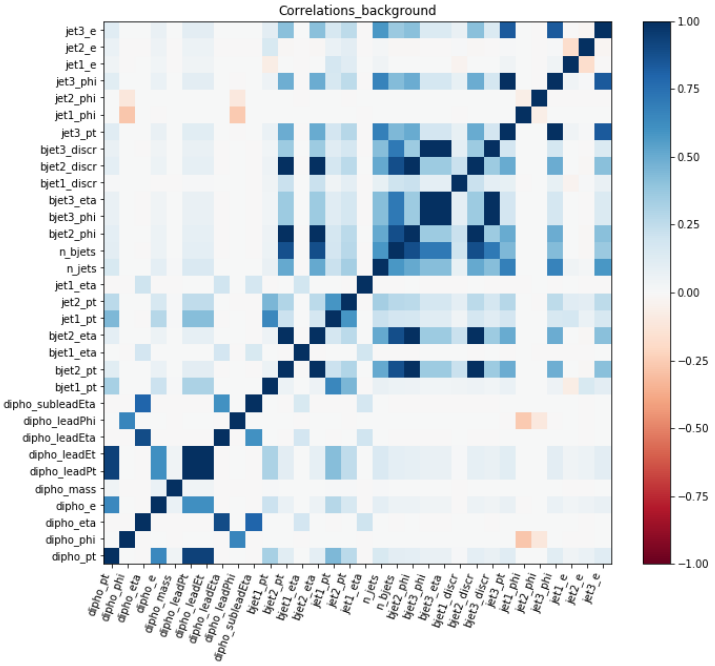
$\sum y$ = Total of the Second Variable Value

$\sum xy$ = Sum of the Product of first & Second Value

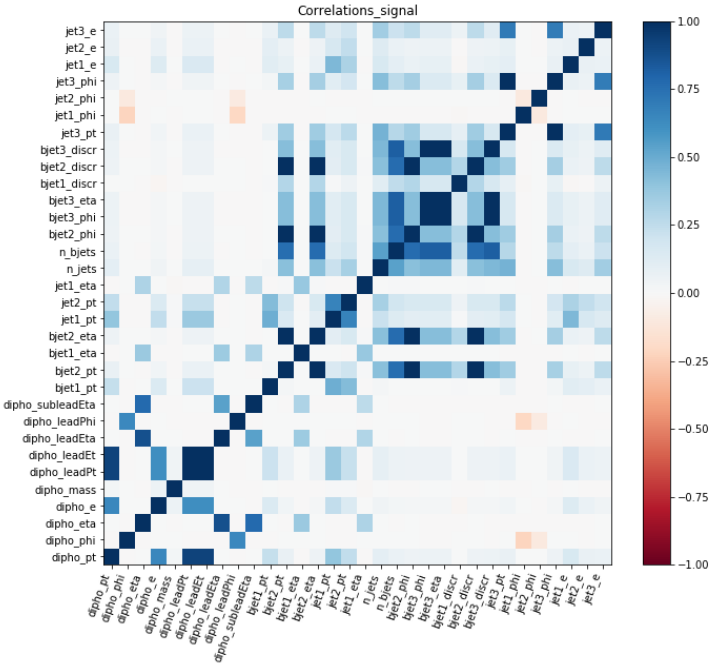
$\sum x^2$ = Sum of the Squares of the First Value

$\sum y^2$ = Sum of the Squares of the Second Value

The correlation plot for the given variables of Tprime and ttgg are plotted in the Figure 4.3.



(a) Background(ttgg)



(b) Signal(Tprime)

Figure 4.3: Correlation plot of two different datasets for different variables. Above, (a) Correlation plot for background(ttgg) and below, (b) Correlation plot for signal(Tprime)

Sl. No.	Variables
1	dipho_pt
2	dipho_phi
3	dipho_eta
4	dipho_mass
5	dipho_leadPt
6	dipho_leadEt
7	dipho_leadEta
8	dipho_leadPhi
9	dipho_subleadEta
10	bjet1_pt
11	bjet2_pt
12	bjet1_eta
13	bjet2_eta
14	jet1_pt
15	jet2_pt
16	jet1_eta
17	n_jets
18	n_bjets
19	bjet2_phi
20	bjet3_phi
21	bjet1_discr
22	bjet2_discr
23	bjet3_discr
24	jet3_pt
25	jet3_phi
26	jet1_e
27	jet2_e
28	jet3_e
29	dipho_e

Table 4.2: Variables of $\bar{t}th$, Tprime(T'), thq, and ttgg used in separation of signal and Background

4.4 Binary Classifications

Using the above discussed DNN model, when fitted with batch size of 900, provided random state is 5 and epochs is 100 on the training dataset of Tprime and ttgg. We will obtain the output of signal and background as shown in the figure Figure 4.6. The training and

testing done using the deep neural network can be verified using the model accuracy and loss. The model training and testing loss and accuracy comparison have been shown in the Figure 4.4 and Figure 4.5

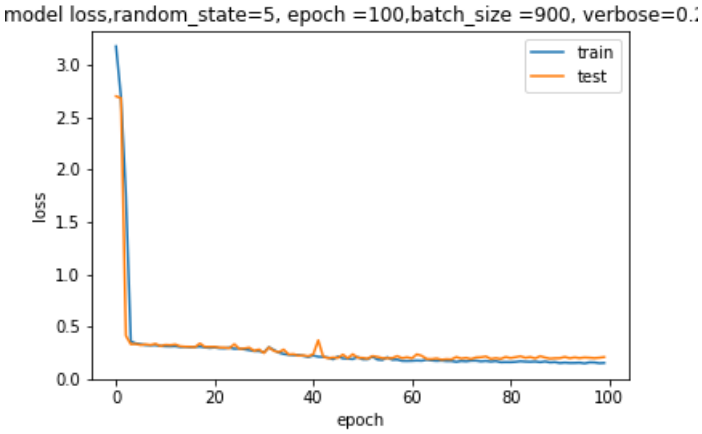


Figure 4.4: Training and testing loss when Tprime is used as signal and ttgg as the background

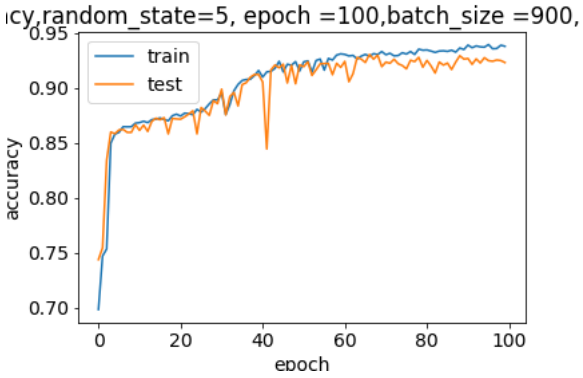


Figure 4.5: Training and testing model accuracy when Tprime is used as signal and ttgg as the background

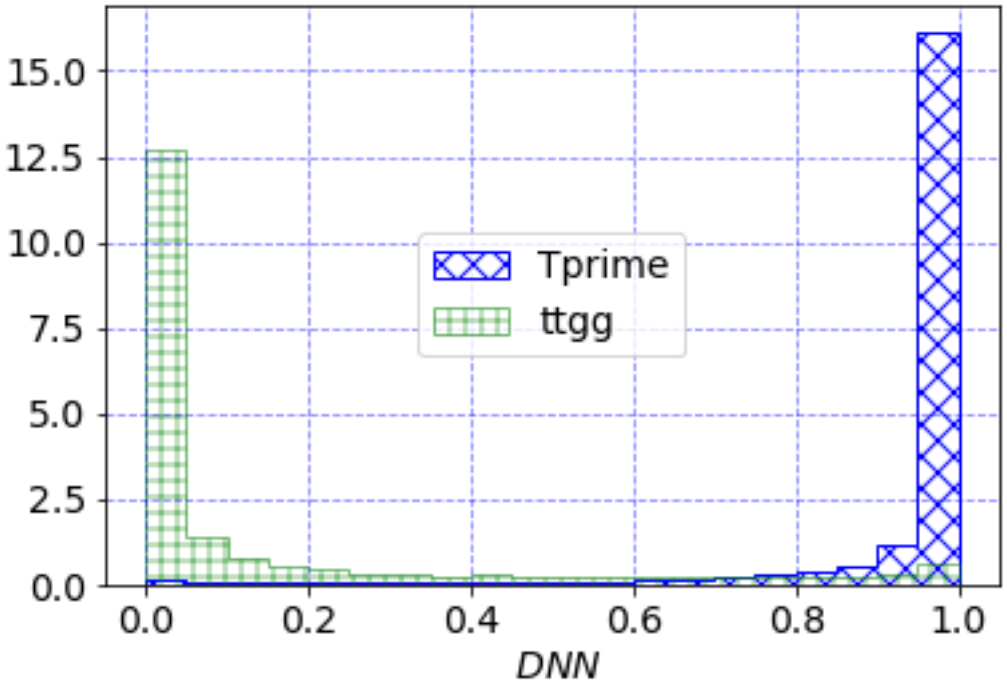


Figure 4.6: Output of training using the DNN(Deep Neural Network). Here signal(Tprime) and background(ttgg) are clearly separated with background as 0 and signal corresponds to 1.

The performance analysis of the DNN model can be done with the help of the Receiving operator Characteristic(ROC) Curve plot for both the training and testing samples. The output ROC curve are plotted in the Figure 4.7

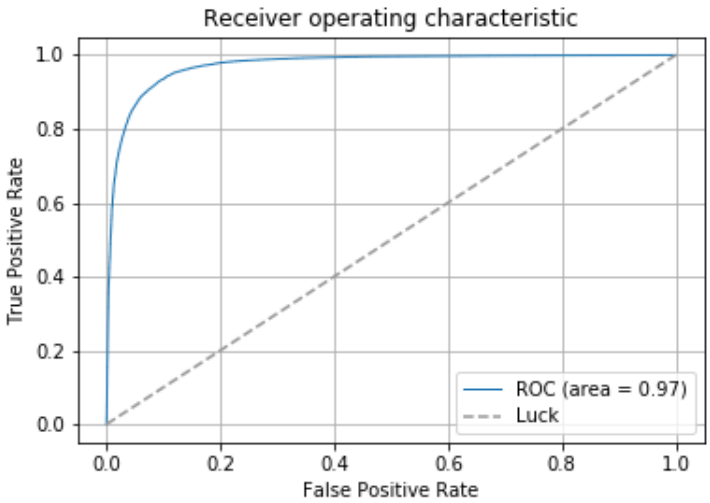


Figure 4.7: ROC curve for the training out of Tprime as signal and ttgg as the background

For the comparison of the models, we can consider any other model of machine learning. Let's take infamous Boosted Decison Tree (BDT) model, it is used very frequently for data training at LHC. The BDT which can also distinguish between signal like and background like events. The output from this model of BDT classifier is plotted in Figure 4.8. The area under the ROC Curve is 0.8863, while the training and testing output is around 82% and 83 % respectively. The input dataset for training and testing was T' and ttgg. The dataset was splitted over 33% for the testing and 67% for training.

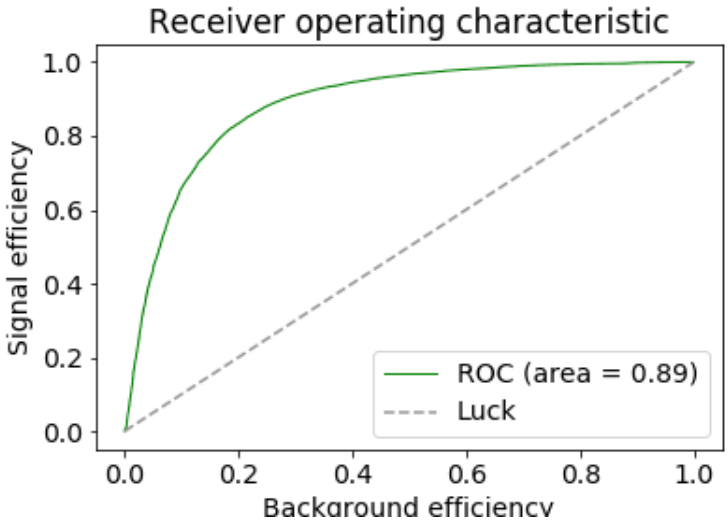


Figure 4.8: Boosted Decision Tree(BDT), ROC curve

Further, the model is used to train and test for the different combinations with the datasets. All the output scores of these training and testing are summarized in the Table 4.3. The output plot for training with Tprime as signal and $\bar{t}th$ & ttgg as background plotted in Figure 4.9. In this figure, we can see the Tprime (signal) corresponds to 1 (plotted in blue) while $\bar{t}th$ & ttgg as background corresponds to 0.

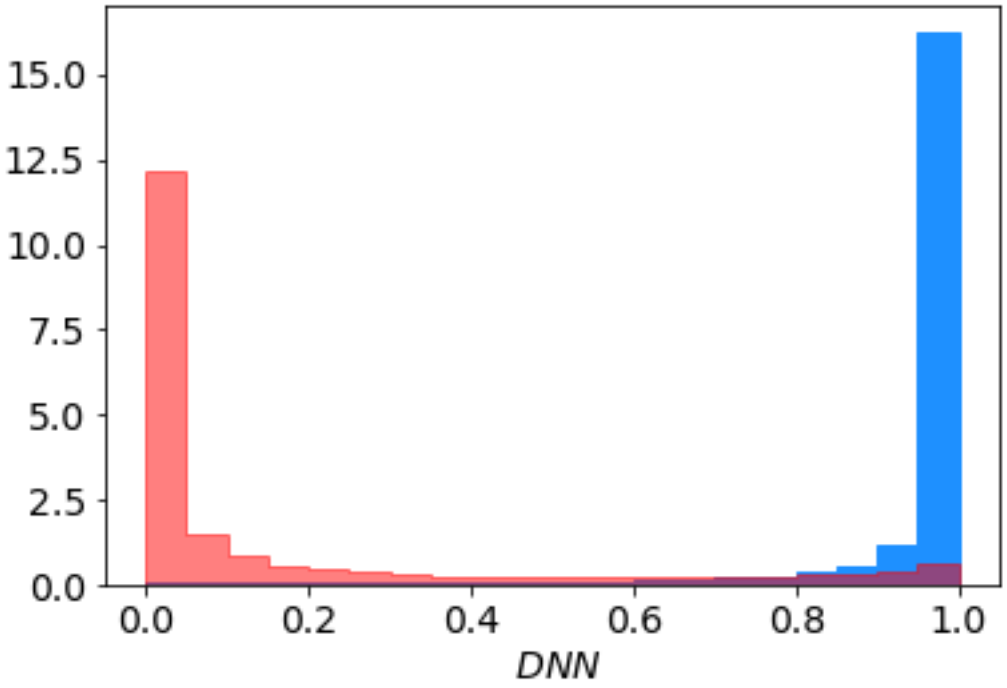


Figure 4.9: Output of training from the DNN(Deep Neural Network). Here signal(Tprime) and background(ttgg & $\bar{t}th$) are clearly separated with background as 0 and signal corresponds to 1.

With the combination of all the three datasets as the background (ttgg & $\bar{t}th$ & thq), and the Tprime as signal, we plotted the results. The ROC-curve for this setup is plotted in the Figure 4.10, and the model accuracy & loss are plotted in Figure 4.11 and Figure 4.12 respectively.

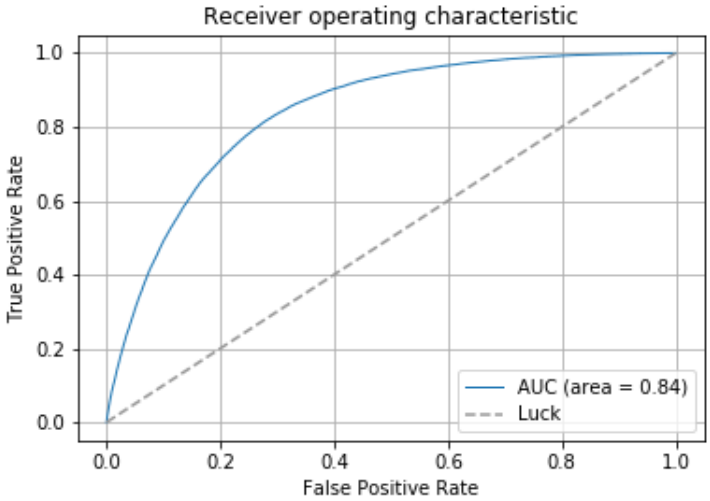


Figure 4.10: ROC curve for the training output of Tprime as signal and ttgg, tth, and thq as the background

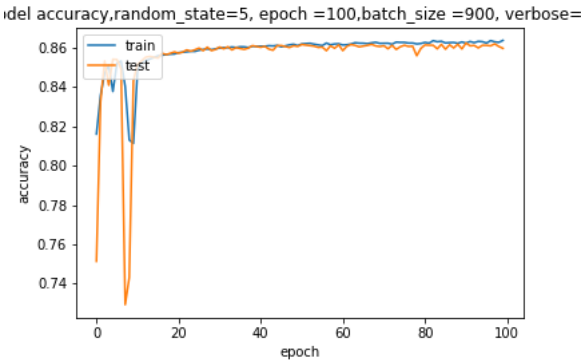


Figure 4.11: Training and testing model accuracy when Tprime is used as signal and ttgg, tth, and thq were used as the background. To train different dataset over a single dataset are known as multi-class classification.

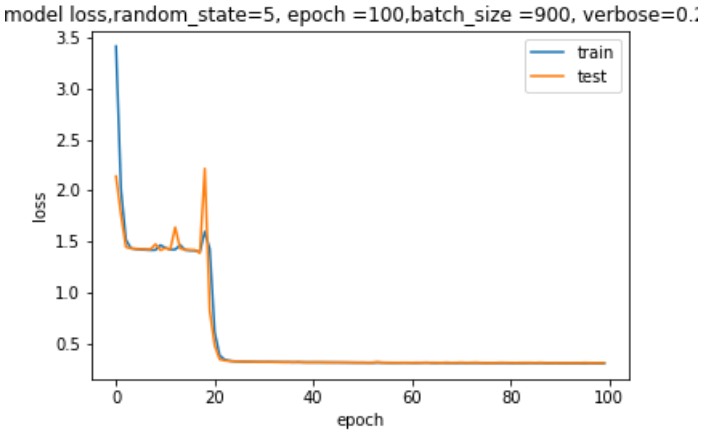


Figure 4.12: Training and testing loss when Tprime is used as signal and ttgg, tth, and thq were used as the background. To train different dataset over a single dataset are known as multi-class classification.

Table 4.3: Table with training and testing accuracy %

Signal	Background	Training Accuracy(%)	Testing Accuracy(%)
TPrime	ttgg	93.30	92.06
TPrime	ttgg& tth	89.84	89.07
TPrime	ttgg& tth& thq	86.36	86.10

If we compare out DNN outputs for each process combination from the Table 4.3, and also by comparing Figure 4.6 & Figure 4.9, we find that the training and testing accuracy are decreasing. To rectify these issues, we plan to implement multi-class classification, which we will be discussing this in the next section.

4.5 Multi class Classification

For the case of multi-class classification, we cannot use the same model of binary classifications(Table 4.1). We tried to create a different model, to avoid the frequent case of model collapsing, the model summary can be seen from Figure 4.13 we try to use batch normalization and dropout layers after each dense layers. ...

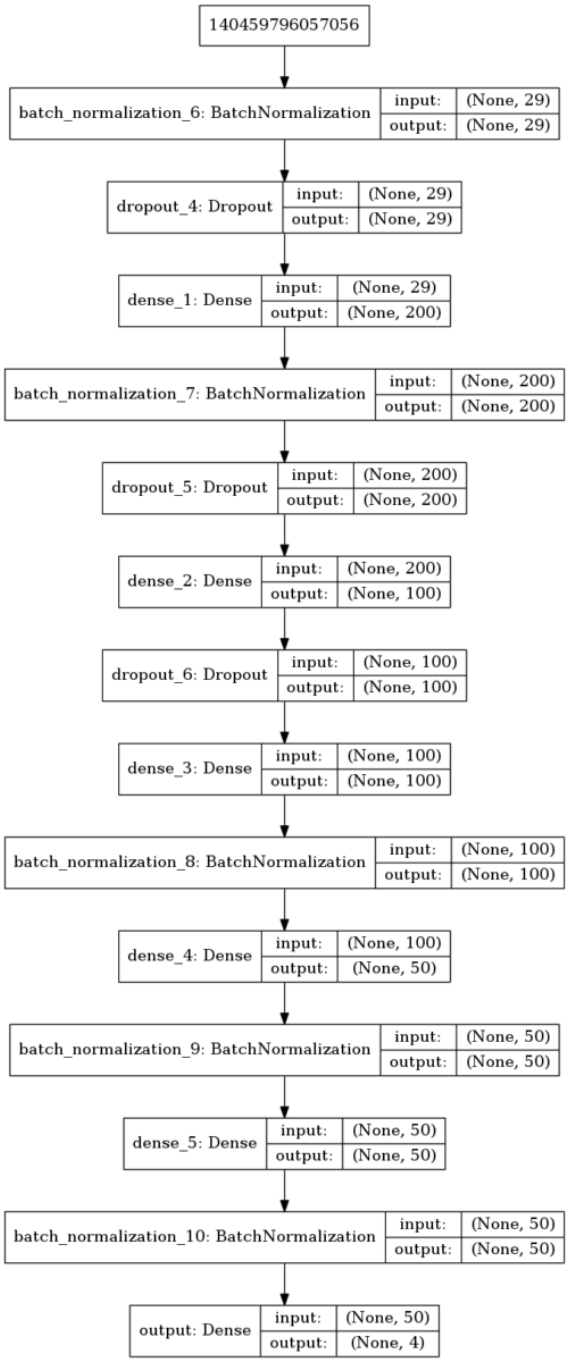


Figure 4.13: mode summary for the multi class classifications

The model obtains the same input variable, as the previous model for the binary classifications Figure 4.2. The total number of input variables are 29 as given in Table 4.2. It is a general technique that can be used to normalize the inputs to a layer. We use first layer of batch normalization and divides the layers into mini batches. The output of this layer goes as input of the subsequent layer and we again use batch normalization and this continues till the output layer.The Output layer consists of 4 different outputs as we were dividing it into 4 different classes of Tprime($T^{'}$), $\bar{t}th$, thq, and tt$\gamma\gamma$, respectively.

Deep neural networks(DNN) are sometimes very challenging to train, not least because the input from prior layers can change after weight update. Thus, by addition of batch normalization, it can be used to normalize the inputs given to a layer. The batch regularization accelerated the training rate and in some cases by halving the epochs or better. and further provides some regularization (two regularization, L1 and L2), and further reducing the generalization error.

The activation function used for inner layers were "ReLU", while the activation function for the output layer were "softmax". The corresponding loss function for multicalss training were "categorical_crossentropy" with "ADAM" optimizer.

The training output of this model is plotted in the Figure 4.14 for accuracy and Figure 4.15, for model loss.

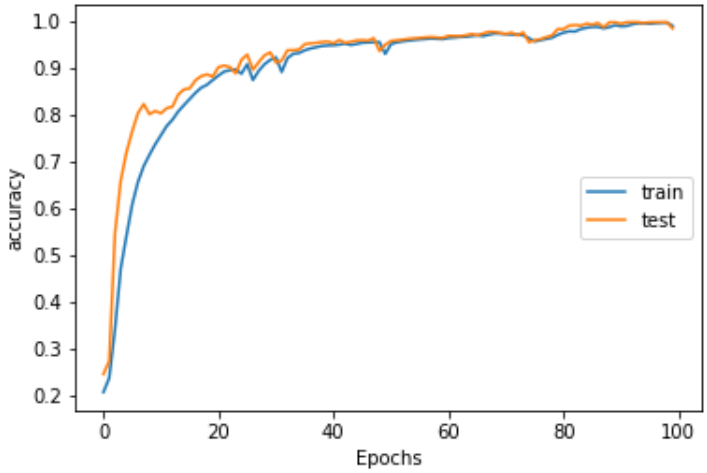


Figure 4.14: Multiclass classification output for accuracy of training and testing

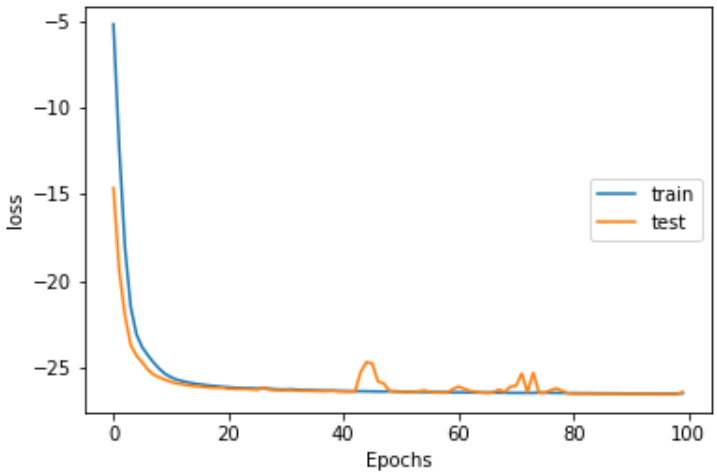


Figure 4.15: Multiclass classification output for model loss of training and testing

Chapter 5

Summary and Conclusions

In this report, we saw how we can use machine learning (Deep Neural Network) techniques to classify data from CMS, as the signal and background. We mainly focused on four simulated datasets, that was Tprime, $\bar{t}th$, thq, and tt$\gamma\gamma$. In all of this dataset, only ttgg has non Higgs background. Here, the resonant particle, Tprime was the signal and rest, $\bar{t}th$, thq, and tt$\gamma\gamma$ as the background. We also learn about deep learning techniques and its implementation. We also saw how deep neural networks (DNN) used to train and modify themselves to give a better optimized outputs. Usually, the output from this methods are better as compared to the other machine learning techniques used to separation, which is evident after the comparison of the output of two different model in Figure 4.8 for BDT and Figure 4.6 for the DNN output. The both training was done with the same dataset.

During the training through DNN, the output from the training of Tprime and ttgg was very good as expected with model accuracy of 93.30 % for training and 92.06 % for testing. ROC curve ouput is also excellent. This may be due to presence of non Higgs background, ttgg dataset as the background. When all the datasets were combined to the same DataFrame, the problem of model collapse was very feasible as also we can see from the output in Table 4.3. The training and testing accuracy started to decrease. This model problem has been rectified with the use of multi classification training of the dataset instead of the binary classification. In multi classification each dataset will perform simultaneous training with the signal dataset simultaneously. As expected the model output are much improved compared to the previous binary classification model. The training and testing

accuracy are 99.87% and 99.86% respectively.

The training over the model of multi classification can be done over any number of background. The expected resulted from this models are far better in comparison to Boosted Decision Tree(BDT), TMVA(DNN), etc.... As expected, we obtained a better classification output for multiclass compared to previous binary classification. Multiclass are a generalized way and less time consuming, whereas the binary classification are tedious and have to be done attentively.

The results of this report constitute an important development on how we can implement machine learning techniques for the search of few properties of new particles at the LHC. This result from the machine learning separation can be used to segregate the raw data from the CMS as signal and background and could be very efficient also.

Bibliography

1. M. Konishi,Associated production of top quarks with the Higgs boson at $\sqrt{s}$ =13 TeV. Paper presented at XXV International Workshop on Deep-Inelastic Scattering and Related Subjects, 3-7 April 2017, University of Birmingham, UK

2. ATLAS, CMS Collaboration, Measurements of the Higgs boson production and decay rates and constraints on its couplings from a combined ATLAS and CMS analysis of the LHC pp collision data at $\sqrt{s}$ = 7 and 8 TeV, JHEP 08 (2016) 045 [arxiv:1606.02266]

3. The CMS Collaboration, Machine learning technique for signal-background separation of nuclear interaction vertices in the CMS detector,CMS-DP-2020-036 ; CERN-CMS-DP-2020-036

4. D. Bourilkov,Machine and Deep Learning Applications in Particle Physics, [arxiv:1912.08245], https://arxiv.org/pdf/1912.08245.pdf

5. Y. LeCun, Y. Bengio and G. Hinton, Nature 521, 436 (2015). doi:10.1038/nature14539

6. D., Frédéric ; Peters, Y.et al.Top Quark Physics at the Tevatron, https://ui.adsabs.harvard.edu/abs/2013IJMPA..2830013D

7. J Mora :Thesis, Deep learning applied in the classification of events generated at the ATLAS experiment, https://hdl.handle.net/11673/49967

8. Ngairangbam, V.S., Bhardwaj, A., Konar, P. et al. Invisible Higgs search through vector boson fusion: a deep learning approach. Eur. Phys. J. C 80, 1055 (2020). https://doi.org/10.1140/epjc/s10052-020-08629-w

9. K. Albertsson et al., Machine learning in high energy physics community white paper. J. Phys. Conf. Ser. 1085(2), 022008 (2018)

10. A. Radovic, M. Williams, D. Rousseau, M. Kagan, D. Bonacorsi, A. Himmel, A. Aurisano, K. Terao, T. Wongjirad, Machine learning at the energy and intensity frontiers of particle physics. Nature 560(7716), 41–48 (2018)

11. P. Baldi, K. Bauer, C. Eng, P. Sadowski, D. Whiteson, Jet substructure classification in high-energy physics with deep neural networks. Phys. Rev. D 93(9), 094034 (2016)

12. CMS Draft Analysis Note:CMS AN-21-105, Search of a Vector Like Quark T' $\longrightarrow$ tH in diphoton final state. https://prsaha.web.cern.ch/prsaha/Tprime_analysis/AN-21-105_temp.pdf

13. He, K., Ren, S., Sun, J., & Zhang, X.Deep Residual Learning for Image Recognition (2016), CoRR, abs/1512.03385

14. G. Krintiras, ttH and tH production at 13 TeV(2017), Conference Report,CMS CR -2017/176: https://cdsweb.cern.ch/record/2272651/files/CR2017_176.pdf

15. https://cms.cern/detector

16. https://cms.cern/detector/bending-particles

17. https://cms.cern/detector/bending-particles

18. https://cms.cern/news/how-cms-detects-particles

19. https://cms.cern/detector/identifying-tracks/silicon-strips

20. https://cms.cern/detector/computing-grid

21. https://machinelearningmastery.com/

22. https://keras.io/

23. https://towardsdatascience.com/

24. https://www.thomasgmccarthy.com/an-introduction-to-collider-physics-ix

25. https://openlearninglibrary.mit.edu/courses/course-v1:MITx+6.036+1T2019/courseware/Week7/neural_networks_2/

26. https://www.phys.ufl.edu/~avery/ivdgl/itr2001/proposal_all.pdf

27. https://edusera.org/machine-learning/

28. https://www.displayr.com/what-is-correlation/

29. Guest D., Cranmer K.,Whiteson D., Deep Learning and Its Application to LHC Physics,Annu. Rev. Nucl. Part. Sci. 2018. https://doi.org/10.1146/annurev-nucl101917-021019

30. The CMS Collaboration, Machine learning technique for signal-background separation of nuclear interaction vertices in the CMS detector, CMS Performance Note(2020),CMS DP -2020/036