Improving $t\bar{t}H$ Detection in ATLAS Experiment Using Machine Learning Techniques

Xiang Zou,[1] Nello Bruscino,[2] Simonetta Gentile[2]

[1*] *Department of Physics, The University of Hong Kong*
[2] *Department of Physics, INFN and Sapienza Università*

[*] *x.zou@cern.ch*

Abstract: Collision data are recorded at the rate of $40MHz$ in the Large Hadron Collider (LHC) with over $60TB$ of data created every second, which contributes to over $10GB$ of data being permanently stored in various data centers after initial triggering. Analysing this huge amount of data is challenging, since traditional ways of data analysis are too slow. Luckily, with the ever-advancing computing power, machine learning techniques are now applied to a variety of tasks. Therefore, as proposed by my supervisor, Dr. Nello Bruscino, I tried to use two different machine learning algorithms, Multi-Layer Perception (MLP), or I later referred as "Ordinary Neural Network", and Graph Neural Network (GNN) to help finding Higgs boson created in proton-proton collisions in associate with a couple of top-antitop quarks.

1. Introduction

1.1. Theory of Higgs boson

The Large Hadron Collider (LHC) is performing proton-proton collisions at high energy $\sqrt{s} = 13.6TeV$. Since the start of Run 3 in July 2022 [ATLAS, 2019], collision data are collected every $25ns$, which is at the rate of $40MHz$. Higgs boson maybe created in some proton-proton collisions, but as the life-time of Higgs boson is of the magnitude $10^{-22}s$ [Collaboration, 2014],which means we need to look for the resulting decay products.

In my project, I investigated the Higgs boson production in associate with a top-antitop quark pair. Known as $t\bar{t}H$ in the multilepton final states. This is one of the most important processes to study Higgs bosons in CERN. Both top quarks and the Higgs boson can decay into different modes, therefore, there are in total six distinct final states [ATLAS, 2019]

1. $2lSS$: Two same-charge light leptons only
2. $3l$: Three light leptons only
3. $4l$: Four light leptons only
4. $1l2\tau_{had}$: one light lepton and two opposite-charge hadronically decaying τ-lepton
5. $2lSS1\tau_{had}$: two same-charge light leptons and one hadronically decaying τ-lepton
6. $3l1\tau_{had}$: three light leptons and one hadronically decaying τ-lepton

I mainly worked on the fifth channel mentioned above, which is the $2lSS1\tau_{had}$ channel. The data I used to train my machine learning model later on include two light leptons (electrons, muons and their anti-counterparts), and 1 hadronically decaying τ leptons.

1.2. Background Rejection

As mentioned, the goal of my project is to identify true $t\bar{t}H$ events from all similar background events that create two same charge light leptons and 1 hadronically decaying τ lepton.

The main backgrounds to $t\bar{t}H$ signal arises from the $t\bar{t}Z$ (top-antitop quarks pair and a Z boson), $t\bar{t}(W/\gamma^*)$ (top-antitop quarks pair and a W boson or photon), diboson production and $t\bar{t}$ (top-antitop quarks pair). The

events with extra heavy-flavour hadron decayed light leptons, jets misidentified as leptons, or photon converted electrons are together labelled as “non-prompt leptons” [ATLAS, 2019].

In my project, I focused on one particular background, $t\bar{t}$, because it is one of the most difficult background to be distinguished from the $t\bar{t}H$ signal.

1.3. Monte Carlo Simulation

When training the neural networks to perform $t\bar{t}H$ classification, it is not ideal to use pre-existing ATLAS analysed LHC collision records of $t\bar{t}H$ events in a particular channel from previous years as training data set. The reason is that the quality of those data may not be perfect, (i.e. there maybe some false positive misidentified $t\bar{t}H$ events from either backgrounds.) If non-ideal training data sets are used, the best performance of the new neural networks will be bounded by the overall accuracy of the training data sets. Correspondingly, we will use the results from Monte Carlo simulation as the training and testing data sets.

Monte Carlo simulation is a crucial part in different stages of the experimental particle physics, including the collision experiment conducted at CERN [Pia and Weidenspointner, 2012]. The state-of-the-art Monte Carlo simulation utilize the technique called ”independent particle approximation” (IPA), where it can generate simulated events of the proton-proton collision and the decay products of the Higgs boson by the best of our knowledge. We will be sure that the simulation results are very accurate $t\bar{t}H$ events or $t\bar{t}$ backgrounds, and that will in theory not limit the best accuracy of the neural network [Pia and Weidenspointner, 2012].

In my project, I have used two Monte Carlo datasets (in ROOT file format), one contains parameters and features of 19782 $t\bar{t}H$ signal events and the other contains the same sets of parameters and features of 6654 $t\bar{t}$ background events. And these 2 data files are used for the training and testing of several neural network models.

2. Ordinary Machine Learning Algorithms

Surprisingly, the term ”Machine Learning” was proposed in 1959, limited at that time by the hardware capability. Thanks to the fast-evolving computing power, we can now apply the technique to perform many tasks.

”PyTorch” is a very sophisticated and complete open sourced ”Python” library to perform machine learning tasks, which supports different kind of layers in the Multi-Layer Perception, or MLP, (including Linear Layer, Convolutional Layers, etc.), different loss function and different optimizer to perform the backward propagation. Therefore, most of my project is done with the help of ”PyTorch”.

2.1. Multi-Layer Perception (MLP)

One of the commonly used traditional way of doing machine learning is called the Multi-Layer Perception (MLP), where the architecture is like what shown in Figure 1.

The basic formulation includes an input layer for training or testing data inputs, several hidden layers for calculations and one output layer for result prediction. The hidden layers shown in Figure 1 are called linear layers, which take your inputs and connect every input with every point in the next layer with a particular adjustable weight. The value of any point in the next layer will be the sum of all inputs times its corresponding weights after passing to a chosen activation function. Possible activation functions include sigmoid function, rectified linear activation unit (ReLU) or LeakyReLu function, hyperbolic tangent function and so on, as shown in Figure 2. They are mainly used to limit the size of the value of the points in the hidden and output layers to improved the performance.

Then, we can get a prediction from the output layer, to compared with a “label”, namely the expected result, and the program will compute how different is the predicted and expected result using a loss function. The program will then propagate backwards by adjusting every weight in every layer to minimize the loss. Commonly used loss functions include Mean Squared Error, Cross-Entropy Loss, Binary Cross Entropy. Different loss functions are suitable for different tasks

The number of points in the output layers should match the number of classes of our classification task. In my project, I am only distinguishing $t\bar{t}H$ signal from $t\bar{t}$ background, so I had 2 points in the final layer, one corresponding to the probability of the inputs being a $t\bar{t}H$-like and the second point corresponding to the probability of

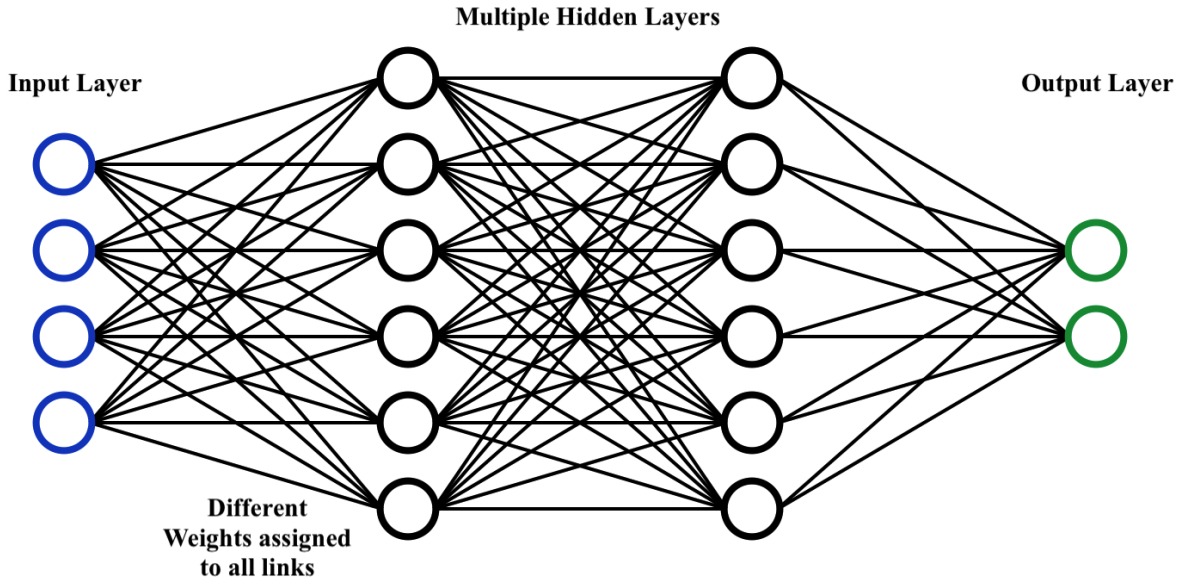


Fig. 1: This figure shows the architecture of MLP, where we will have an input layer for training or testing data inputs, several hidden layers for calculations and one output layer for result prediction. Every point in the previous layer is connected to every point in the subsequent layer with a particular weights, and in supervised machine learning, we will calculate the loss between the prediction and the labels and then modify the weights in all layers. Credit: "https://medium.com/swlh/an-introduction-to-neural-networks-de70cb4305f9"

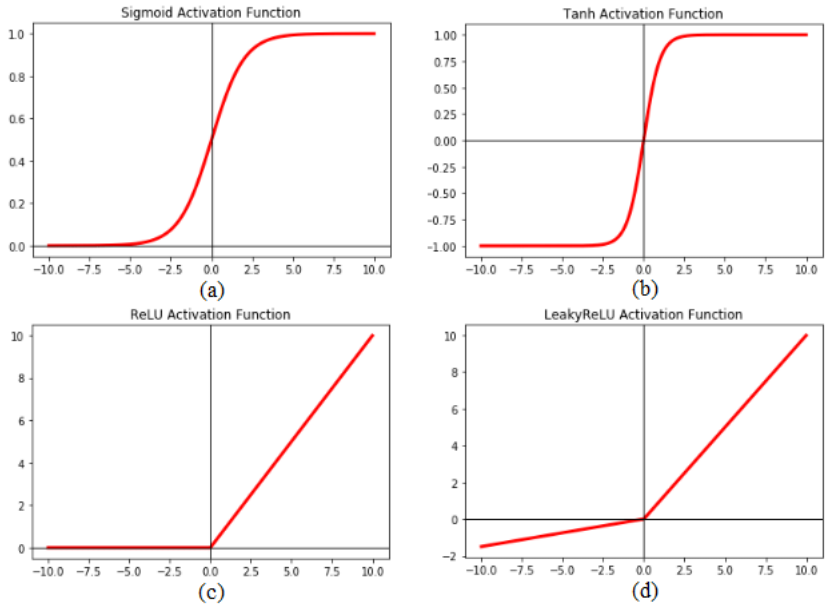


Fig. 2: This figure shows different activation functions used between layers in the neural networks, aiming to confine the value of the points in neural network to some specific value and improve the accuracy of the model. [Kandel and Castelli, 2020]

the inputs being a $t\bar{t}$-like.

2.2. *Advantages, Disadvantages and precautions of MLP*

As one of the most commonly used machine learning algorithm, MLP has several advantages:

1. It can learn and model some complex and non-linear relationships, like $t\bar{t}H$ signals
2. Programmer do not need to master everything in the field, for example, user do not have to know what every features being inputted in the neural network is, and the program will run without problem.
3. Not only can MLP be applied to different tasks in research, but it can also be used in our daily lives, for instance, Image Recognition or Natural Language Processing.

The training speed is relatively slower compared to Graph Neural Networks, which will be introduced in session 3. Also, we need to be sure that the input data has the same order of magnitude. If one of the input data is a few order of magnitudes greater than the others, then, the value of all points in the next layer will be dominated by that particular input and the effect of other inputs is suppressed. This is actually a common mistake, which took me about 3 days to figure out and figure out and solve.

We can apply different method to normalize the input data size. For example, using Logarithm

$$inputs = \log(features)$$

the good side is that the size of the input will be normalized and still kept the relative size between inputs, but the drawbacks is that we need extra treatment to input data that are non-positive number, which may sometimes cause serious problems.

Another possibility is to use standard score of particular inputs

$$input = \frac{feature - mean(features)}{std(features)}$$

this algorithm works fine for non-positive inputs, but the drawbacks is that the relative size between input is lost, and every input data contribute exactly the same to the final predictions.

I applied both way to train my MLP, and the final performance of both methods are very similar.

3. Graph Neural Networks

Graph Neural Networks (GNN) are a relatively new and rapidly evolving machine learning technique, which was only brought to public attention only since 2018, and many different implemented GNN models are proved to have excellent results in various tasks [Xu et al., 2019]. And they could have even better performances than ordinary neural networks, like MLP, for a fixed amount of training data and training time.

A very sophisticated package to implement GNN is called Deep-Graph-Library, or DGL. It is an open-sourced library based on "PyTorch" that support all the GNN functions, including creating graphs and assigning nodes/edges features. It also encodes some commonly used GNN layers, including Graph Convolutional Layer (GCN), Graph Attention Networks (GAT). Therefore, I performed most of my GNN training and testing associated with DGL.

3.1. *The use of Graph Neural Networks*

Unlike ordinary neural networks, e.g. MLP, which take inputs in the form of numbers or arrays of numbers; GNN takes graphs as input. The Mathematical definition of a graph G is a ordered pair

$$G = (V,\ E),\ where\ V\ is\ a\ set\ of\ vertices,\ and\ E\ is\ a\ set\ of\ edges$$

$$E \subseteq \{(x,\ y) | x,\ y \in V\ and\ x \neq y\}$$

where each edge are directional. The GNN are capable of performing three main types of tasks.

1. **Node Classification**: Given a graph with all nodes characterised in different categories, predict the corresponding category of a newly added node.

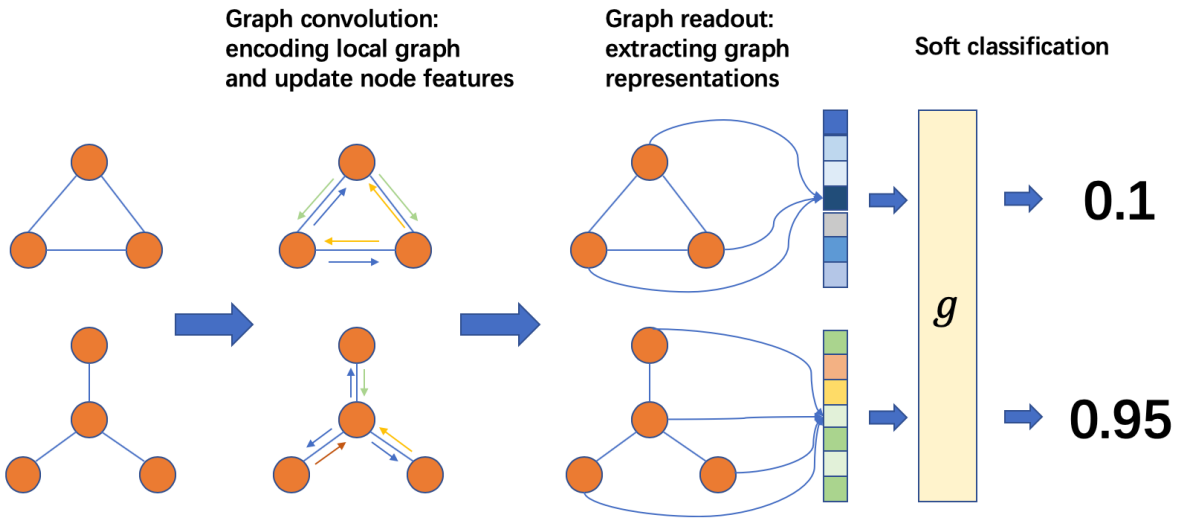


Fig. 3: This figure shows the step in GNN Graph Classification task, the input to the model are no longer number, but different graphs, and during the training phase, we allow neighbourhood aggregation between every nodes in the graph, and finally get all information from node embedding and put them in an MLP network to generate predictions. Credit: DGL Documentation

2. **Link Prediction**: Given an linked graph with different edges, predict the probability of an edge exist between any node and a newly added node.

3. **Graph Classification**: Given multiple same-shape graphs from different categories, predict the corresponding category of a new graph.

The first two type of tasks can be used commonly in our daily life, like recommending new friends for a user on social media pages, or recommending related products for a user on online-shopping sites. But in my project to distinguish $t\bar{t}H$ events from $t\bar{t}$ backgrounds, I used the third type of task of GNN, namely the "graph classification".

When we input different graphs to our GNN model, and allow the "neighbourhood aggregation", which includes message-passing between every pairs of connected nodes, the "message" that a node pass to its neighbour undergoes a process called "node embedding", which embeds nodes to a low-dimension space so that these embedding captured the essential task-specific information and use them to train the model. The embedding are automatically generated using unsupervised dimensionality reduction approaches, the detail of neighbourhood aggregation will be discuss in the next subsection. Finally, we will put the embedding of all nodes into a MLP model and ask it to generate a prediction to us, this process is the same as what discussed above.

3.2. Neighbourhood Aggregation

Neighbourhood aggregation is the key distinction between the GNN and ordinary neural networks, where different neighbourhood aggregation and message passing function corresponds to different GNN layers. Well-known and widely used GNN layers include Graph Convolution Networks (GCN), GraphSAGE Convolution Networks (SAGEConv), Graph Attention Network (GAT), Relational Graph Convolution Networks (RCCN) and much more. The message passing framework is described as follows,

$$m_{u\to v}^{(l)} = M^{(l)}(h_v^{(l-1)}, h_u^{(l-1)}, e_{u\to v}^{(l-1)})$$

$$m_v^{(l)} = \sum_{u\in N(v)} m_{u\to v}^{(l)}$$

$$h_v^{(l)} = U^{(l)}(h_v^{(l-1)}, m_v^{(l)})$$

where $m_{u\to v}^{(l)}$ is the l^{th} order of message passed from node u to node v, the target node. $h_v^{(l)}$ is the embedding of node v in the l^{th} order. And $M^{(l)}$ is the "message function", depending on the kind of GNN networks and takes as input the embedding of both nodes and the edge data between the nodes. $N(v)$ denotes all neighbours of node v,

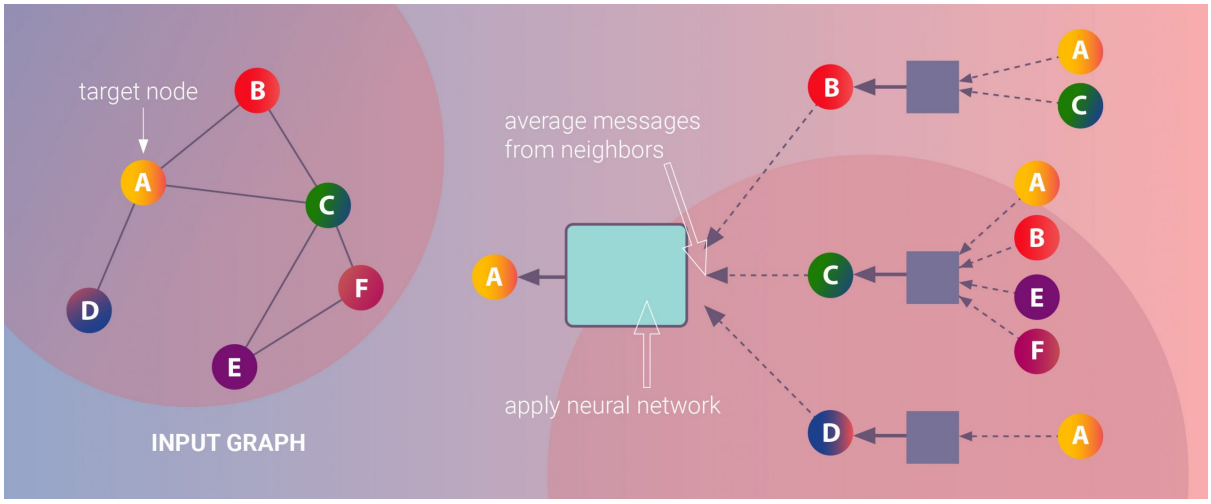


Fig. 4: This figure shows the details of neighbourhood aggregation and message passing in a general GNN layer, suppose we have the graph as shown on the left side of the graph, and node A is the targeted node. Node A has 3 neighbours, namely node B, C and D. The GNN will first embeds the information stored in node B, C and D and use the model-specific message function to pass the message to node A. Node A gather all information from its neighbour, and use the model-specific reduce and update function to modify its own data. This process is repeated for every nodes in this graph. Credit: "https://perfectial.com/blog/graph-neural-networks-and-graph-convolutional-networks/"

the target node will process messages from all its neighbours. Notice that the $\sum$, called the "reduce function", can represent any function, like mean, maximum or summation. then, the target node will invoke $U^{(l)}$, the "update function", to update its embedding, and hence update its node data.

Every nodes experiences this neighbourhood aggregation to modify its own data in one GNN layer, and the whole process is repeated for different layers. For example, the SAGEConv layer has the following architecture [Hamilton et al., 2017],

$$h^k_{N(v)} = Average\{h^{k-1}_{N(v)},\ \forall u \in N(v)\}$$

$$h^k_v = ReLU\{W^k,\ Concat(h^{k-1}_v),\ h^k_{N(v)}\}$$

Where the W is the weight matrix, used as the message function in this case. Different type of GNN layers will have various message functions, reduce function and update functions.

3.3. Advantages, Disadvantages and precautions of MLP

GNN models, which have been brought to social spotlight, have some great advantages over the normal neural networks. Firstly, the training phase with same data size is much faster in GNN than MLP, the most time consuming part in a GNN training task is actually combining graphs of different events into a large batched graph.

But in order to apply GNN to multiple tasks, programmers need to master the input features, create suitable graphs with nodes and edges and correctly assign the information to corresponding nodes and edges, I will discuss more on how the training graphs are created in the next section.

Also, GNN encounter the same issue as MLP, where user needs to ensure all nodes and edges data assigned are of the same order-of-magnitude.

4. Application of Neural Networks for $t\bar{t}H$ identification

Two different approaches are applied in the classification of $t\bar{t}H$ signal events from $t\bar{t}$ background events, MLP and GNN, the working principle of the two machine learning algorithms are discussed above, and I will now discussed the implementation part.

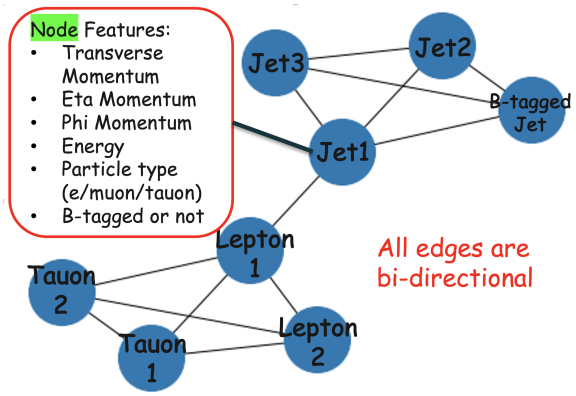


Fig. 5: The input graph to the GNN models, with every nodes corresponding to an end product of the detection, and edges between every pairs of particles and every pairs of jets.

4.1. $t\bar{t}H$ classification with MLP

For each Monte Carlo event used for training, I gathered in total 91 useful features as inputs, using the standard-scoring way of normalization. Then a MLP model with 3 hidden layers, each with 50 points, is created. ReLU function is used between hidden layers and Sigmoid function is used at the final output.

There are 2 outputs as mentioned above, the first output corresponds to the probability of the input event being a $t\bar{t}H$ signal event, and the second output represents the probability of $t\bar{t}$ background event.

Cross-Entropy Loss are applied to the predictions, with the label of $t\bar{t}H$ being a pytorch tensor $[1,\ 0]$ and the label of $t\bar{t}$ is a pytorch tensor $[0,\ 1]$. The optimizer applied utilizes the Adam Algorithm with learning rate 1×10^{-4}.

During each epoch of training, I randomly selected 5000 $t\bar{t}H$ events as the positive samples, which are from the first 15000 events given in the ROOT file storing in total 19782 $t\bar{t}H$ events. Meanwhile, the first 5000 $t\bar{t}$ events from the ROOT file storing in total 6654 $t\bar{t}$ events are used as negative samples. That contributes to a total of 10000 events, which are randomly shuffled before fetching into the MLP model.

On my local machine, each epoch takes around 10 seconds to run, and each model runs for a total of 150 training epochs, with model testing done for every 3 epochs. The whole process is repeated for 9 times, and the result is shown in the next section.

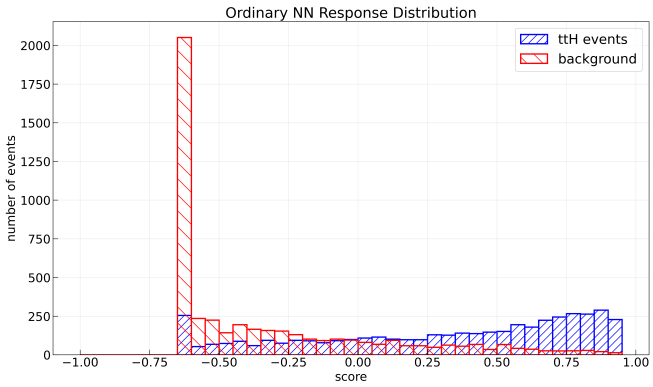
4.2. $t\bar{t}H$ classification with GNN

For each $t\bar{t}H$ event generated by Monte Carlo simulation that is used for training, 26 useful features are gathered and being assigned to a graph as inputs to the GNN, the features uses the standard-scoring way of normalization.

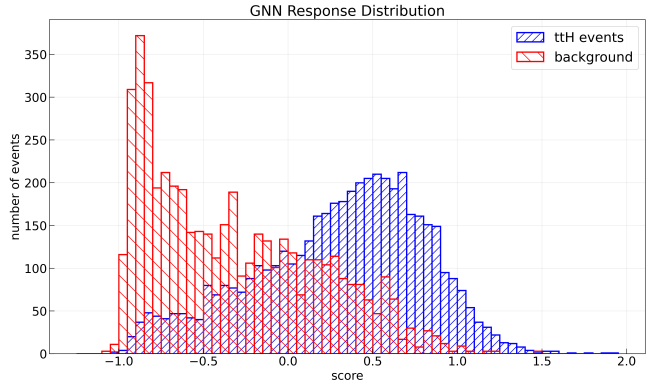
The input graph shape is as shown in Figure 5. Where a node is created for every final state object created, including jets, light leptons and τ leptons. and we have bi-directional edges connecting every pairs of particles and every pairs of jets (in practice, there are 2 edges in opposite directions connecting two nodes). Then, we assigned the features to the nodes, including the momentum in different direction, the energy, and if the node represents particle, we specify the particle type by denoting the Monte Carlo number of that particle; else if the node represents jet, we specify whether the jet is b-tagged or not.

A GNN model with in total 6 times of neighbour aggregation is used, 3 GCN and 3 SAGEConv are applied for the task, and in the final MLP of each neighbour aggregation, 50 hidden points are used to generate 2 final results.

Similar to the above-mentioned MLP model, the first output corresponds to the possibility of the input event being a $t\bar{t}H$ signal event, and the second output represents the probability of $t\bar{t}$ background event, and the same loss function and optimizer, i.e. Cross-Entropy-Loss and optimizer using Adam Algorithm with the learning rate of 5×10^{-3}, are applied with the same positive and negative labels as MLP model.



(a) MLP Response Distribution



(b) GNN Response Distribution

Fig. 6: The Response Distribution by 2 different machine learning algorithms are shown, where blue histogram symbolize the score got by $t\bar{t}H$ testing samples, while the red histogram represents the score of $t\bar{t}$ testing samples. The score is defined as $score = output(t\bar{t}H) - output(t\bar{t})$, from the 2 output classes of the neural networks.

The training events preparation is exactly the same as that in MLP model, where in total of 10000 events are used for each training epoch, I generated distinct graphs of the same shape corresponding to each event, and random shuffle them before batching them into a large single graph. The large batched graph is then put into the model for training. Each epoch takes less than 1 seconds to train, but the graph batching takes about 8 seconds.

The model is tested every 25 epochs. Then I prepare the training data again with a new data set and ordering of the input graphs. Every testing result is noted down. A total of 2000 epochs are trained for each run, and I have performed 9 runs in total, the result is presented in the next section.

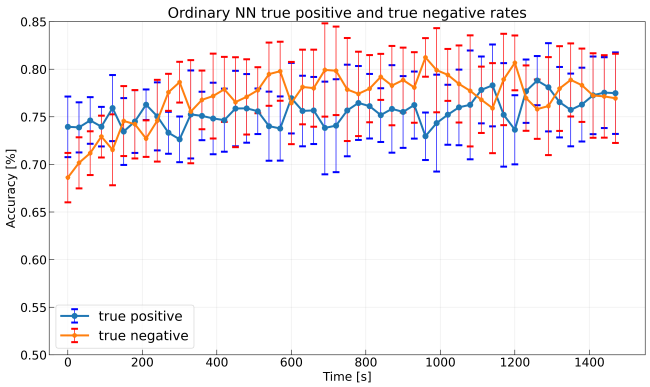
5. Graphs and Result Discussion

Both machine learning techniques achieve an acceptable performance, about 75% overall accuracy, although this accuracy is a bit less than my initial expectation. My report will focus on the comparison of the performance by the two different approaches. The comparison criteria include the response distribution, the true positive and true negative rate, the overall accuracy, the stability of the model performance, and finally, the receiver operating characteristics (ROC).

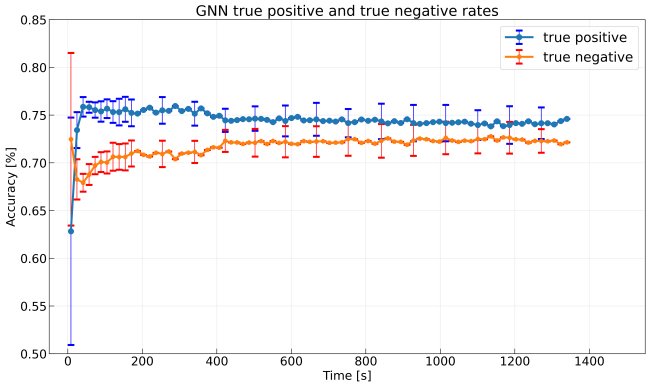
5.1. Response Distribution

Both machine learning algorithm will give 2 outputs as described above, the first output symbolizes the probability of that event being a $t\bar{t}H$ signal, and the second output corresponds to the chance that the event being a $t\bar{t}$ background. I defined the score of a particular model as

$$score = output(t\bar{t}H) - output(t\bar{t})$$



(a) MLP True Positive and True Negative Rate



(b) GNN True Positive and True Negative Rate

Fig. 7: The true positive and true negative rate of two different neural networks are shown, where the blue lines represented the true positive rate, and the orange line represented the true negative rates. 1-σ error bar is presented as well.

. For a particular testing event input, if $output(t\bar{t}H) > output(t\bar{t})$, or in another words, $score > 0$, then the event will be classify as a $t\bar{t}H$ signal, otherwise, the testing event is classified as $t\bar{t}$ background.

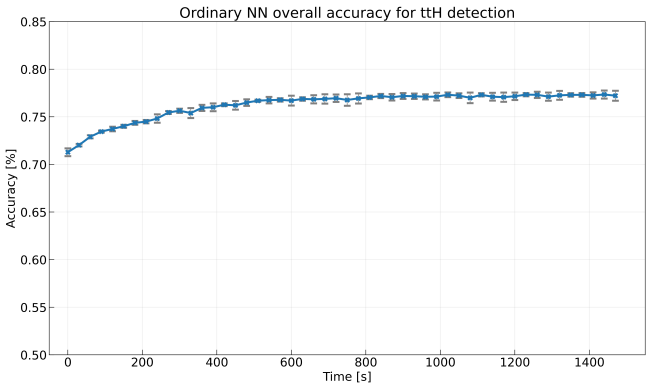
As demonstrated in Figure 6, one can see clear distinctions between the peaks of the histograms of $t\bar{t}H$ signal and $t\bar{t}$ background in the response distribution of both MLP and GNN, which means that after training, both models can at least distinguish most events of $t\bar{t}H$ signal and $t\bar{t}$ background.

A feature that one can easily spot in Figure 6 (a), the response distribution curve for MLP, is that there is a huge peak of $t\bar{t}$ background event score at the bin with boundaries -0.65 and -0.7, I guess the reason is that the application of sigmoid function at the output layer in the MLP, while I did not do the same at the output layer of GNN model.

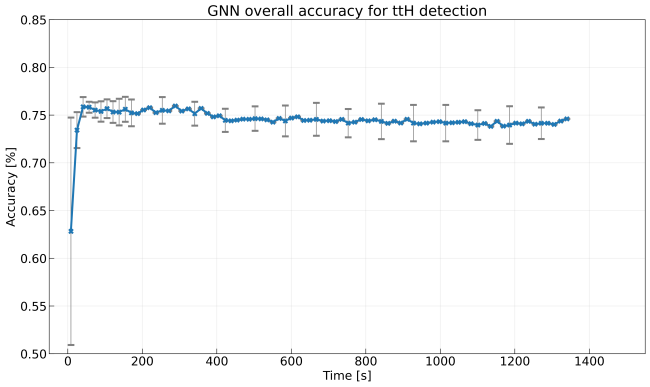
Applying sigmoid function can be benefitial, because it will normalize the output to a real number between 0 and 1. But sometimes, using sigmiod could be catastrophic, the reason is that for a number deviate a lot from 0, the sigmoid of that number will be close to either 0 or 1. To have a sense of the converging power of sigmiod function, I present the following example, where sigmoid function is represented by $\sigma()$, and we take up to four decimal numbers,

$$\sigma(1) = 0.7310,\ \sigma(3) = 0.9525,\ \sigma(5) = 0.9933,\ \sigma(7) = 0.9991,\ \sigma(9) = 0.9999,$$

Therefore, if we do not use sigmiod function carefully, it will seriously affect our loss function and the whole machine learning process.



(a) MLP Overall Accuracy



(b) GNN Overall Accuracy

Fig. 8: The overall accuracy of MLP and GNN models are presented, please notice that here we assume that the number of $t\bar{t}H$ signal events and $t\bar{t}$ background events are the same.

5.2. True Positive and True Negative Rate

The MLP and GNN models are separately run for 9 times, and the testing statistics is collected as mentioned in section 4. The result is shown in Figure 7. Although their accuracy is about the same, one can easily depict that GNN model is in general much more stable than the MLP model. The standard deviation of the true positive or negative rate in GNN is about 2%, while that of the MLP model is approximately 5%, which is more than a double than that of GNN model. Also, the curves of the true positive or negative rate of GNN are gradually approaching a steady value, while that of the MLP model have huge fluctuations and do not seem to have a tendency of approaching to any stable value.

Stability is a very important criterion of determining whether a model has good performance. Because in the real LHC experimental data, the background events may contribute a much higher proportion than the signal events in all collisions events recorded from the detectors. In this case, we may want to choose a model with higher true negative rate, so that we will not have a lot of mis-identified $t\bar{t}H$ signal events from backgrounds. Figure 9 (credits to one of my best colleagues at CERN, Severin König) showing the proportions of events detected in LHC with $2lSS1\tau_{had}$ channel is attached as reference, one may see that the $t\bar{t}H$ signal events only occupy about 22% of all detected events.

5.3. Overall Accuracy

This sub-session just gives an intuitive view of what the overall accuracy of our machine learning algorithms will look like. As shown in graph Figure 9, we can see the percentage of events of $t\bar{t}H$ and that of $t\bar{t}$ are 22.0% and 20.1% respectively, which are relatively close. Hence, I just assumed in my testing phases, the number of events of $t\bar{t}H$ signal and $t\bar{t}$ background is the same, and the overall accuracy is shown in Figure 8.

Please notice that the error bar of the overall accuracy can be misleading. For example, for an untrained

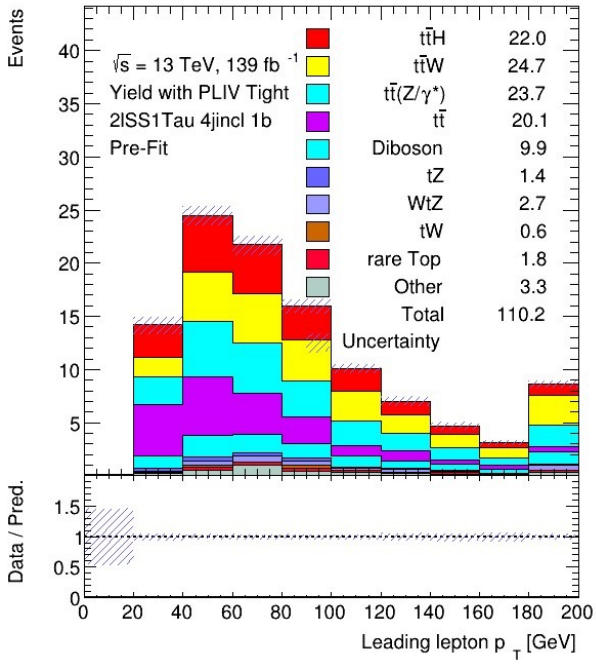


Fig. 9: This figure is from the ATLAS experiment LHC Run 2 data, and the different colour represents different events from the $2lSS1\tau_{had}$ channel. One can see that $t\bar{t}H$ signal events only contributes to 22% of all events, but the percentage of events of $t\bar{t}H$ and that of $t\bar{t}$ is roughly the same. Credit: Severin König

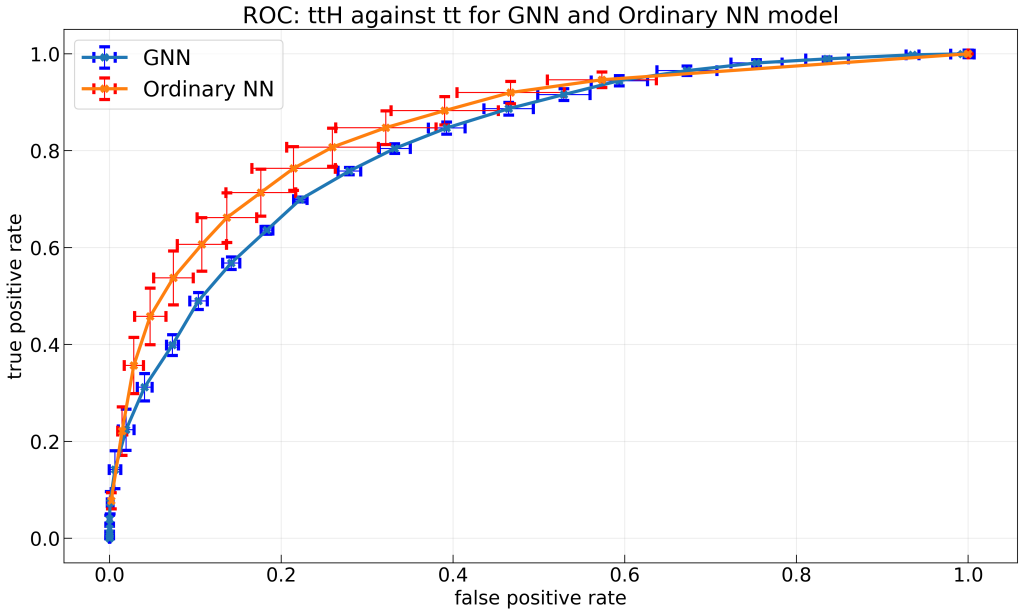


Fig. 10: This graph shows the ROC curves, a plot of true positive rate against true negative rate evaluated at all thresholds, of both models with $1-\sigma$ error bar.

model, the overall accuracy is expected to be around 50%, with small standard deviation. But due to the initial parameters in the model, the true positive and true negative rate can vary hugely, and the standard deviation of the true positive and negative rate should be large. This is indeed a phenomenon seen in the MLP model, as shown in Figure 8 (a), where the standard deviation in the true positive and true negative analysis of the MLP model is quite large, but the standard deviation of overall accuracy is very small.

5.4. Receiver Operating Characteristics (ROC)

An receiver operating characteristic curve, (ROC curve) is a commonly used method to present the performance of any model, whose main task if events classification at all user-defined thresholds. The graph is a plot of **true positive rate** against **false positive rate**. The diagonal line, i.e. the $y = x$ line, symbolizes that the model perform classification task at random, because the true positive rate and false positive rate are exactly the same. Similarly, if the curve is upper than the $y = x$ line, the model performs better than random, and otherwise, the model performs worse.

A ROC curve is crucial to compare the performance of the classification task from different model on the same data sets. Generally speaking, the larger the area under the ROC curve, the better the performance of the model.

The result is presented in Figure 10, both model actually perform pretty good in $t\bar{t}H$ signal identification. Where superficially one may argue that MLP model have a better performance than GNN model. However, please be reminded that the MLP model takes 91 input features, while the GNN model only takes 26 input features. And the standard deviation in the ROC curve for the MLP model is much larger than that of the GNN model.

6. Future Work on Improving the Models

The main method of my GNN model follows an suggestion presented at the 4^{th} IML Machine Learning Workshop on 21 Octobrt, 2020, by a group of researchers[1] from University of California, Berkeley. Where they metioned that the performance of GNN model are expected to out-compete the normal MLP model. However, in my implementation, the widely used MLP model seems to work better. Therefore, this section aims at proposing some possible ways to improve the performance of the GNN model, and perhaps MLP model as well.

6.1. Includes More Features

One main reason that leads to the unsatisfactory performance of my GNN model could be that only 26 features are used to train the GNN model, and a lot more features could be added. Currently, I only assigned features to nodes as shown in section 4, and additional features can be fed into the GNN model through two different ways, adding edge features or including some global nodes. Global nodes are nodes that will only pass message to other nodes while not accepting any message from other nodes.

Edge features:

1. Angular distance between 2 products
2. Invariant mass of any pair of products
3. Etc.

Global Nodes:

1. MET significance (missing Energy and momentum)
2. Number of Jets
3. Lepton flavor
4. total charge of the system
5. Etc.

[1] Shuo Han, Xiangyang Ju, Pamela Pajarillo, Ryan Roberts, Haichen Wang, Allison Xu

6.2. *Search for Better Models*

As mentioned before, the field of Machine Learning, in particular, GNN, is a rapidly improving and actively researching field in computer science. Where more advanced models are proposed from time to time, and some of those models could be used to improve the $t\bar{t}H$ detection task.

However, one should be noticed that a more complicated model does not necessarily means the model has a better performance. For example, I have tried to add over ten linear layers, or increase the size of each hidden layer in my MLP model, but the performance is hardly improved, and the training time of the model will be much longer.

Also, I also tried to use a very complicated GNN model, called the Graph Isomorphism Network (GIN), which is argued as one of the best GNN model for graph classification tasks with state-of-the-art performance [Xu et al., 2019]. However, since the idea is newly proposed, and the explanations of GIN is not adequate from sources online, I applied the GIN model to my $t\bar{t}H$ signal identification without understanding its fundamental working principle. The outcome is not satisfying at all even after I spend over one week to modify and improve the model and input graphs (its classification performance is barely better than random after many training epochs). As a result, I would recommend that any newly developed models must be understood thoroughly before applying them to any real tasks, and a more complicated model does not necessarily perform better than a simple but elegant model.

7. Conclusion

In my project, two different machine learning approaches are applied to perform the classification task for $t\bar{t}H$ signal over $t\bar{t}$ background, aiming to help with the classification in the future ATLAS LHC experimental collision data. The first method is the traditional MLP model, while the second one is the rapidly evolving GNN model. Collision data from Monte Carlo simulation is used as the training and testing data set.

The overall accuracy of both models are about 75%, while the accuracy of MLP is slightly higher than that of GNN model, but GNN model attained a much more stable performance while training the model for several times. The detailed result is described in section 5.

The reason why GNN model performance is under expectation is that it only takes one third the number of features inputs than the MLP model. We can improve the GNN model performance by incorporating more inputs as ”edge features” between pairs of nodes or introducing ”global nodes” to the input graph.

The 8-week Summer Student Program is definitely a fruitful and once-in-a-lifetime experience to me, where I learned all the essentials of machine learning, and how to apply existing packages (PyTorch and DGL) to help with the given task. Now I have a deeper look into the actual particle physics research, which will be a precious experience that will shine its light on my front path of scientific research.

References

[ATLAS, 2019] ATLAS (2019). Analysis of $t\bar{t}H$ and $t\bar{t}W$ production in multilepton final states with the ATLAS detector. *ATLAS CONF Note*, 14(3):342–351. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/CONFNOTES/ATLAS-CONF-2019-045.

[Collaboration, 2014] Collaboration, C. (2014). Evidence for the direct decay of the 125 gev higgs boson to fermions. *Nature Physics*, 10(3):557–560.

[Hamilton et al., 2017] Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive representation learning on large graphs. *31st Conference on Neural Information Processing Systems*, 1:16.

[Kandel and Castelli, 2020] Kandel, I. and Castelli, M. (2020). Transfer learning with convolutional neural networks for diabetic retinopathy image classification. a review. *Applied Sciences*, 10:2021.

[Pia and Weidenspointner, 2012] Pia, M. G. and Weidenspointner, G. (2012). Monte carlo simulation for particle detectors.

[Xu et al., 2019] Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? *International Conference on Learning Representations*, 1:17.