ParicleNetLite++: dynamic graph convulutional neural network for signals selection at the Level-1 Trigger

Yuanhe Zhang
H.H. Wills Physics Laboratory, Department of Physics, University of Bristol.

(Dated: August 21, 2022)

In this paper, we proposed a new machine learning method for signals selection. The core of the method is to treat the collision events as a particle cloud, a disordered set of particles. And we improved the data processing method to be more explainable and natural. Based on the particle cloud representation, we introduced the ParticleNetLite++ architecture with special edegconv operation, a dynamic graph convolutional neural network architecture. The experimental results indicated that the model's accuracy and efficiency can reach 0.999 at hundred-level parameter size.

INTRODUCTION

The CMS detector [1] is one of the general detectors of the LHC. It can measure countless particles in the final state of proton-proton collisions. CMS is a closed detector with beamline coverage close to 4π, including a large phase space region to allow observation of a large number of collision event topological geometric structures, whose coordinate system is usually defined by the pseudorapidity μ and the azimuth angle ϕ[1]. Also, CMS is able to precisely measure the momentum of charged-particles.

CMS's triggering system consists of two physical layers: L1 Trigger (L1T), which is a set of algorithms that run on low-latency, highly parallel hardware to generate a provisional, accept or reject decision for a given bunch crossing. And High Level Trigger (HLT), which runs a more complex set of decision-making tools through online software [2]. In this study, we focus on the L1 Trigger. The L1T receives event inputs from various detector systems and operates on the information using algorithms (triggers) to determine whether the event is related to the physical signal of interest, or consists only the background. In this study, we focus on the di-Higgs decay $HH \to b\bar{b}b\bar{b}$, and this event is treated as a signal (interest of physics). And the goal of research is to develop a deep learning algorithm which can be efficiently used to classify signals and backgrounds. Then, this algorithm will be implemented in L1 Trigger.

[3] proposed a Convolutional Neural Networks architecture for our task which represented event as images. So it converted the task into a computer vision binary classification problem. However, the ordering of the image matrix causes loss of particle topology structures. Also, the images are 2D, which the model cannot recognize data as 3D spatial structure during training. So when the edge of the image cuts the high-energy region apart, it will lead to misclassification. So we proposed a data processing method for representing the collision events as point clouds and implement a lightweight dynamic graph convolutional neural network.

EXPERIMENTAL DETAILS

1. Data pre-processing

Converting dataset is the most important part in the whole selection process. In the raw ROOT format dataset which was generated from Monte Carlo simulations of the events at the LHC, it contained non-regular arrays which each array represented class of informations about collision events. And each array contained same number of subarrays which contained information of specific class for each single collision event. And each values in subarray represented the information for a particle which was recorded by the detector for this single collision event. For comparison with Convolutional Neural Networks for signals selection, we choose 'Particle/Particle.Phi', 'Particle/Particle.Eta' and 'Particle/Particle.PT' three classes to construct dataset used for the neural networks. Then, I will introduce data-preprocessing implementation which I take part by part.

1.1 Event as a particle cloud

We proposed to represent collision event as an unordered, permutation-invariant set of particles. This structure can flexibly contain arbitrary features of each particle (as many as we want). So we represented collision event as a particle cloud. They are actually highly similar to the point cloud representation of 3D shapes used in computer vision because they both are unordered sets of entities distributed irregularly in specific space. Also, they both represent high-level objects which have rich internal topology structures(i.e., collision particles or 3D shapes).

The idea of regarding events as unordered sets of particles was similar to the idea of [4] which represented jets as particle clouds and it got high prediction accuracy (0.940) for Top tagging and 0.840 for Quark-gluon tagging. In particle physics or heavy ion experiments, a jet is a narrow cone of hadrons and other particles created by the hadronization of a quark or gluon. In general, the collision of high-energy particles can result in the emission of jets of elementary particles. So we can regard jets as elements of collisions event results. We can expand this idea to the whole collision events by customizing the architecture in [4].

Then, the implementation of constructing particle clouds is to regard (η, ϕ) as coordinates of particles in pseudorapidity-azimuth space (2D space). Then, the set of particles' coordinates is a particle cloud and we regard each particle cloud as an event. Also, we treat (η, ϕ, pT) as features corresponding to each coordinate. So we will have two types of input data for the neural network: (η, ϕ) and (η, ϕ, pT).

1.2 Cutting, Sorting and Padding

Particle data records are of high quality when $|\eta| < 3$ in the CMS detector. So we cut the records of each collision event according to the value of eta. The operation is: cut off the eta value whose absolute value is greater than 3, and cut off the ϕ and pT values of the corresponding particle. More explicitly, we remove (η, ϕ, pT) and (η, ϕ) depending on eta value.

Then, we sort the particles according to their pT value from largest to smallest. Because in the model, we will only apply the first hundred particles. This operation allows the model to automatically select the 100 particles with the highest pT values. High-energy particles have features that are easier to distinguish, making it easier for the model to implement learning.

Then, we need to pad the arrays of η, ϕ, pT separately. The purpose of this operation is to make the data regular so that the dataset can be readable for the model. Among all collision events, we take the number of particles of the collision event with the most particle records as the desired length. Then for all collision events that do not contain this number of particles, we will pad the arrays of pT with 0 and pad the arrays of ϕ and η with 10. I will explain the reasons for selecting 0 and 10 in the 1.3 Normalization section.

1.3 Normalization

Till now, Our dataset still contain extreme values, thus they do not refer to the same distribution. It is challenging for models to learn datasets with different scales. Therefore, normalisation is required. The Normalization statistics we used is the standard score and the formula is:

$$\frac{X-\mu}{\sigma}, \tag{1}$$

where μ is the mean and σ is the standard deviation. And this method can normalize errors when population

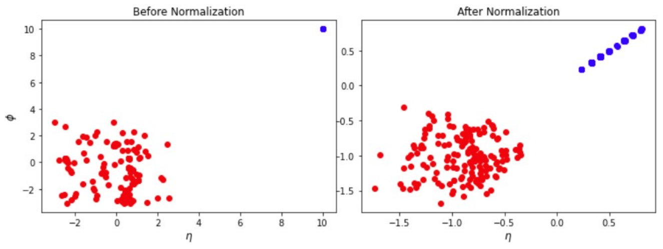

FIG. 1. In the pseudorapidity-azimuth space, the red part is the cluster of existing particles that recorded by the CMS detector, and the blue part is the cluster of non-existing particle that we pad into data. After normalization, we can clearly see that there is still a certain distance between the red and blue clusters, which is sufficient to distinguish the existing particles from those that are not existing when we apply the KNN algorithm to construct graph data.

parameters are known. It works well for populations that are normally distributed.

First, I need to explain that when we construct graph data, we need to apply the k-nearest neighbors algorithm. The KNN algorithm [5] is used to find the k closest training samples to the object in the feature space. Data normalization can greatly improve the accuracy of the algorithm. Before normalization, we pad the arrays of ϕ and η with 10 in order to ensure that existing particles will only look for nearest neighbors in clusters of existing particles and non-existing particles will only look for nearest neighbors in non-existing particle clusters. We don't set this supplementary value too large because after normalization, the existing particle clusters are pulled, too large value can cause excessive pulling, leading to distortion. Also, we reduce the impact of the non-existent particle graph data on learning features for the model by padding the arrays of pT with 0.

2. Network Architecture

The particle cloud's permutation symmetry makes it a natural and promising representation of the collision events. In order to maximize the aggregated information in the data to achieve the best performance, we designed a highly customized neural network with lightweight parameters called ParticleNetLite++, inspired by Dynamic Graph Convolutional Neural Networks. In this section, we will introduce ParticleNetLite++.

2.1 Edge Convolution

First, the convolution operation takes use of the translational symmetry of images by sharing kernels across the entire image. This not only drastically decreases the number of parameters in the neural networks, but also enables the parameters to be learned more efficiently, as each set of weights will utilise all images regions for training. CNNs utilise a hierarchical technique [64] for learning image features. Convolution operations can be stacked effectively to create a deep network. Different layers of CNNs have different receptive fields and can therefore learn features at various scales, with the shallower layers using neighbourhood information and the deeper layers learning more global structures. Such a hierarchical technique to learning images proved effective.

[6] has introduced the edge convolution (”EdgeConv”) operation as a convolution-like operation for point clouds. EdgeConv begins by representing a point cloud as a graph whose vertices are the points themselves and its k nearest neighbours. In this approach, a local patch required for convolution is defined for each point as the k nearest connected neighbours. Consequently, the EdgeConv operation for each point x_i has the form

$$x_i^{'} = K_{j=1}^{k} h_{\Theta}(x_i, x_{i_j}), \tag{2}$$

where $x_i \in R^d$ denotes the feature vector of the point x_i and i_1,...,i_k are the indices of the k nearest neighbors of the point x_i. The edge function h_Θ: $R^d \times R^d \rightarrow R^{d^{'}}$ is function parametrized by a set of learnable parameters Θ and K is a channel-wise symmetric aggregation operation, e.g., max, sum, or mean. The parameters Θ of the edge function are shared for all points in the point cloud. This, together with the choice of a symmetric aggregation operation K, makes EdgeConv a permutationally symmetric operation on point clouds [7].

Choice of h and K. EdgeConv's properties are significantly impacted by the edge function and aggregate operation chosen. And in this paper, we will follow the choice in [6] to choose

$$h_{\Theta}(x_i, x_{i_j}) = \hbar_{\Theta}(x_i, x_{i_j} - x_i) = \theta_m \cdot (x_{i_j} - x_i) + \phi_m \cdot x_i, \tag{3}$$

where the feature vectors of the neighbors, x_{i_j}, are substituted by their differences from the central point x_i and $\hbar_\Theta$ can be implemented as a multilayer perception (MLP) whose parameters are shared among all edges. For the aggregation operation K, we use mean, i.e.,$\frac{1}{k}\sum$, which have been proved that has the best performance in the experiments. And $\Theta = (\theta_1,...,\theta_m,\phi_1,...,\phi_m)$ are set of parameters which are trainable for the model. . This strategy not only consider about global information x_i of local neighborhoods, but also local neighborhoods information $x_{i_j} - x_i$.

Then, we can define our updated edges features by notating

$$e_{ijm}^{'} = ReLU(\theta_m \cdot (x_{i_j} - x_i) + \phi_m \cdot x_i), \tag{4}$$

which can be implemented as a shared MLP. m is the number of neurons of MLP and $\Theta = (\theta_1,...,\theta_m,\phi_1,...,\phi_m)$ are set of parameters which are trainable for the model. Then we can update the point x_i

$$x_{im}^{'} = \frac{1}{k}\sum_{j=1}^{k} e_{ijm}^{'}, \tag{5}$$

Also, inspiring by ResNet [8], each block also has a parallel shortcut connection to the EdgeConv operation that allows the input features to pass through directly.

EdgeConv can be stacked as easily as regular convolutions. EdgeConv can be viewed as a mapping from one point cloud to another with the same number of points, only possibly changing the dimensionality of the eigenvectors of each point. Therefore, another EdgeConv operation can be applied subsequently. This enables us to use the EdgeConv operation to build a deep network that learns the features of point clouds hierarchically. The feature vector learned by EdgeConv can be regarded as the new coordinates of the original point in the latent space, and then, we can apply EdgeConv again. We can understand that the graph of the point cloud is dynamically updated to reflect changes in the edges, i.e. the neighbors of each point. [6] shows that this leads to better performance than keeping the graph static.

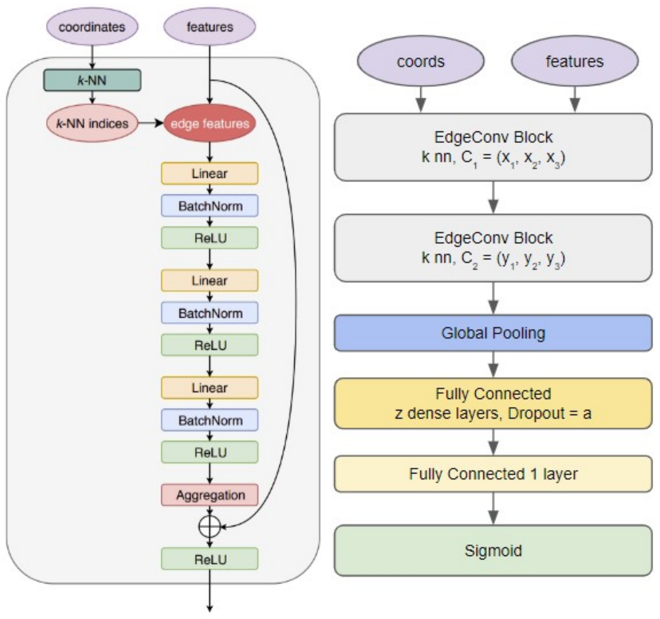

FIG. 2. [6] gave a general introduction of edge convolutional block: The EdgeConv block takes as input a tensor of shape $n \times f$, computes edge features for each point by applying a multi-layer perceptron (MLP) with the number of layer neurons defined as (C_1,C_2,C_3), and generates a tensor of shape $n \times C_3$ after pooling among neighboring edge features.

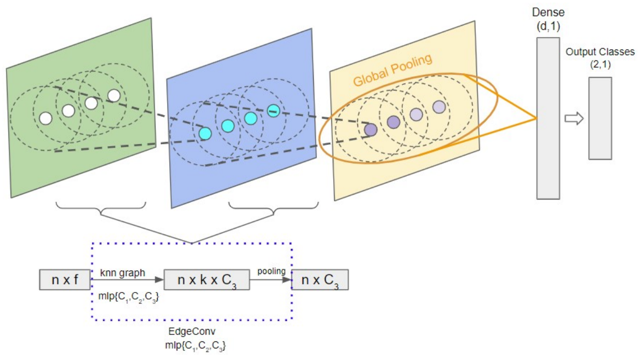

FIG. 3. ParticleNetLite++ Workflow

2.2 ParticleNetLite++

In ParticleNetLite++, we used two EdgeConv blocks, with the number of nearest neighbors k and the number of channels in EdgeConv block $C_1 = (x_1,\ x_2,\ x_3)$ and $C_2 = (y_1,\ y_2,\ y_3)$ for the two blocks, respectively. The number of units in the fully connected layer after global pooling is z. At last, we use Sigmoid function as activation function to get the final outputs. This simplified architecture is denoted as “ParticleNetLite++” and is illustrated in FIG. 2. This neural networks architecture is lightweight and highly customizable. Also it can be more suitable when computational resources are limited.

RESULTS

Our particle selection task is a typical binary classification problem. So after y-layer fully connected dense layer, we used a one-layer dense layer because we only have two classes and we use 1/0 ((1,1) dimension) to label the signal/background.

We implemented the code of ParicleNetLite++ successfully on 6 Tesla T4 gpus. Our dataset is generated by MonteCarlo simulations and included 100000 signals and 100000 backgrounds. Firstly, I concatenated signals and backgrounds (half-half distribution). Then, I splited the total dataset into training dataset and testing dataset in a 3:1 ratio and the same distribution (half-half). And when we trained the model, I use 75% training set for training and 25% training set for validation.

We performed experiments at different levels: EdgeConv Channels level, Dense level, and number of particles used for training in collision events (Particle Numbers level). For each experiment, we trained the model for 50 epochs and set learning rate to 0.001 using Adam optimizer [9].

TABLE I. We set 6 different parameters-size model with index. k(NNs) is the number of nearest neighbors, C_1 and C_2 are two channels for EdgeConv Blocks, Dense is the number of neurons for the fully connected layer, Parameters is the parameters-size. And after we will use index of each model directly.

Index	k(NNs)	C_1	C_2	Dense	Parameters
1	7	(4,4,4)	(4,4,4)	4	243
2	7	(4,4,4)	(8,8,8)	8	475
3	7	(4,4,4)	(8,8,8)	16	555
4	7	(8,8,8)	(8,8,8)	16	815
5	7	(8,8,8)	(16,16,16)	16	1583
6	7	(8,8,8)	(16,16,16)	32	1871

TABLE II. Eff is the efficiency which is the signal efficiency with trigger rate equals to 10 kHz. ACC: Accuracy = (TrueNegative + TruePositive) / (TrueNegative + TruePositive + FalseNegative + FalsePositive). Threshold is the background survival rate when trigger rate equals to 10kHz. 1/0F(0.5) is Signal/BackgroundsFalse (misclassified) when set 0.5 as standard (bkg $<$ 0.5 and signal $>$ 0.5). 1/0F(T) is Signal/BackgroundsFalse (misclassified) when set Threshold as standard (bkg $<$ Threshold and signal $>$ Threshold).

Index	Eff	ACC	0F(0.5)	1F(0.5)	Threshold	0F(T)	1F(T)
1	0.9994	0.9995	54	2	0.96	8	16
2	0.9997	0.9997	23	2	0.82	8	8
3	0.9997	0.9997	15	4	0.74	8	8
4	0.9997	0.9997	15	3	0.83	8	8
5	0.9994	0.9992	48	2	0.97	8	20
6	0.9997	0.9997	27	2	0.91	8	7

DISCUSSION

As we can see in the experiment results, the model can work very well at a very small parameter size. And before we increase size to 1k parameters, the efficiency and accuracy are increasing slightly. And the number of misclassified events is decreasing. But when we increse parameters more than 1k, the performance didn’t imporve but slightly worse than others. We can clearly see that the accuracy and efficiency are still very high but the number of misclassified events is increasing. The main reason for this phenomenon might be overfitting. When we increased parameters of the model, the training curve will overfit the training set so that the model will lose the ability of prediction and only fit very well on training set but not on testing set.

Also, I performed an experiment on Particle Numbers level. I used the No.3 model in the TABLE I. which performed very well. And then I reduced the number of particles used for training in collision events from 100 to 10. Then I trained for 50 epochs with 0.001 learning rate. Finally, the model accuracy reached 0.9998 and

TABLE III. Number of misclassified particles for the 10-particles model

	0F(0.5)	1F(0.5)	0F(0.59)	1F(0.59)
m	13	2	8	3

efficiency reached 0.999. And imporved a little bit on number misclassified particles (See TABLE III.).

CONCLUSIONS

In this paper, we propose a new machine learning method for signals selection. The core of the method is to treat the collision event as a particle cloud, a disordered set of particles. And we improved the data processing method to be more explainable and natural. Based on the particle cloud representation, we introduced the ParticleNetLite++ architecture, a dynamic graph convolutional neural network architecture. And the operation of edegconv is very powerful which has a strong ability to aggregate information. This lightweight model has huge advantages in parameter size and is particularly useful for applications in high-energy physics experiments. The experimental results are very well and show interesting phenomena in different levels, which deserve further study.

REFERENCES

[1] The CMS experiment at the CERN LHC, CMS Collaboration, JINST, 3(08):S08004–S08004, (2008).

[2] The Phase-2 Upgrade of the CMS Level-1 Trigger, CMS Collaboration, Technical Report CMS-TDR-021, CERN, Geneva, (2020).

[3] Neural network triggers for the CMS detector at the HL-LHC, Kyle Pidgeon, (2021)

[4] Jet Tagging via Particle Clouds, Qu H, Gouskos L, Physical Review D, 101.5, (2020).

[5] An Introduction to Kernel and Nearest Neighbor Non-parametric Regression, Naomi Altman, The American Statistician. 46 (3): 175–185. (1992).

[6] Dynamic graph cnn for learning on point clouds, Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, ACM Trans. Graph. 38, 146 (2019).

[7] Deep sets, M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R.Salakhutdinov, and A. J. Smola, Neural Information Processing Systems 30 pp. 3391–3401. (2017).

[8] Deep residual learning for image recognition, CVPR, pp. 770–778, (2016).

[9] Adam: A method for stochastic optimization, Kingma, Diederik P., and Jimmy Ba, arXiv preprint arXiv:1412.6980 (2014).

COVID AND STRIKE ACTION IMPACT

There is no impact caused by COVID and strike action on my experiment.

ACKNOWLEDGMENTS

I am grateful for the support which provided by Dr Sudan Paramesvaran, Mr Maciek Glowacki, Dr Emyr Clement and Mr Jeronimo Segal. Special thanks to Dr Sudan Paramesvaran for offering this research opportunity.