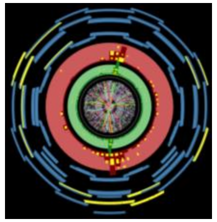
A Very Simple Tutorial on the Use of ParticleNet/ParticleTransformer

Shudong WANG

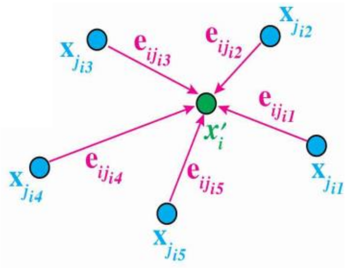
EPD, IHEP, CAS

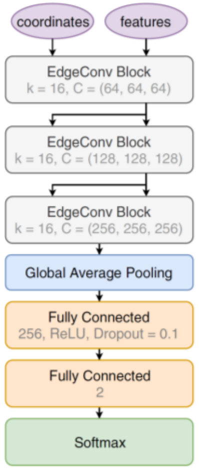
Outline

- **A little background knowledge - Jets**

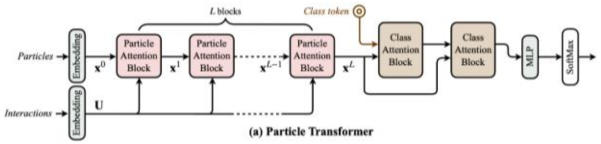


- **A Brief Introduction on ParticleNet**
- Jet/Event as a point cloud
- Point clouds VS Particle clouds
- The architecture of ParticleNet





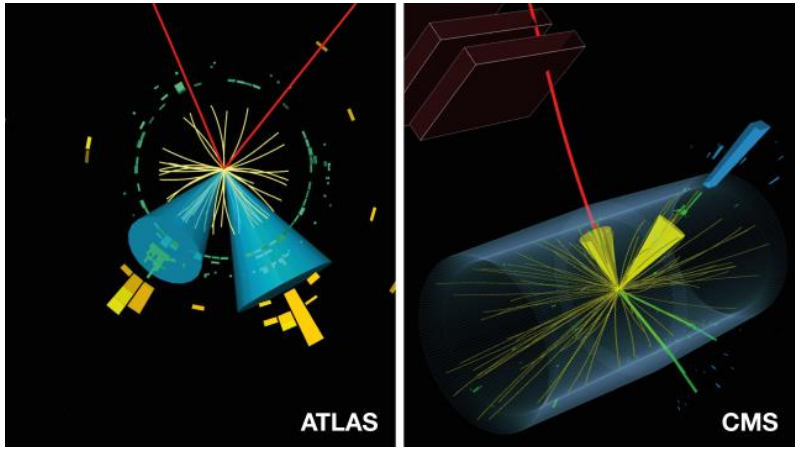
- **A Brief Introduction on ParticleTransformer**
- Attention & Self-Attention
- The architecture of ParticleTransformer



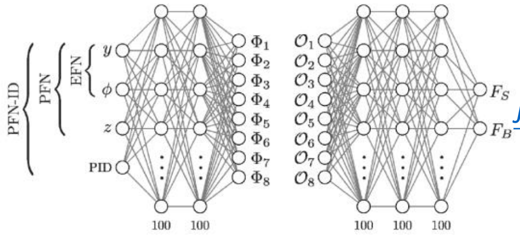
- **How to run ParticleNet/ParticleTransformer using Weaver**
- Try it yourself!

A little background knowledge-Jets

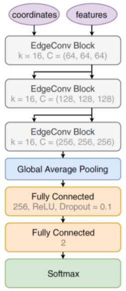
- Jets are ubiquitous at colliders, especially for hadron colliders.
- Jets are collimated sprays of particles initiated by quarks or gluons.



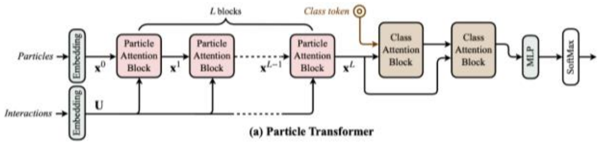
- Jet tagging: identifying the hard scattering particle that initiates the jet.
- The rise of machine learning (ML) has brought lots of new progresses to jet tagging.



JHEP01(2019)121



Phys.Rev.D 101 (2020) 5, 056019



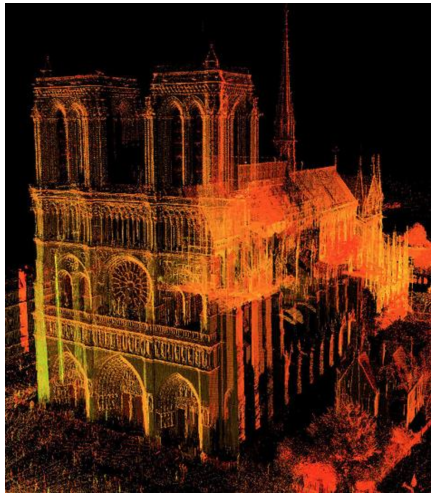
2202.03772

A Brief Introduction on ParticleNet

Based on Huilin Qu(the author of ParticleNet)'s report:

New approaches for jet tagging with machine learning (June 18, 2021) · Indico of IHEP (Indico)

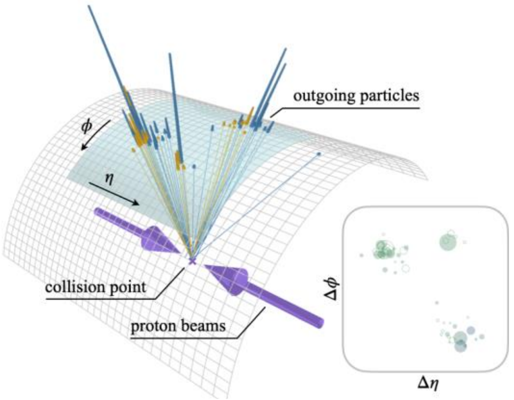
Jet/Event as a point cloud



Point cloud

From Wikipedia, the free encyclopedia

A **point cloud** is a set of data points in space. The points may represent a 3D shape or object. Each point position has its set of Cartesian coordinates (X, Y, Z).[1] Point clouds are generally produced by 3D scanners or by photogrammetry software, which measure many points on the external surfaces of objects around them. As the output of 3D

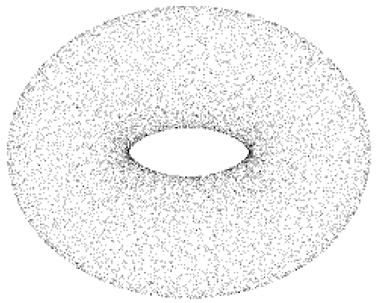


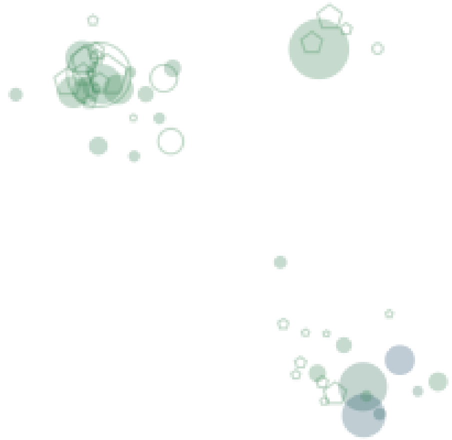
Jet (particle physics)

From Wikipedia, the free encyclopedia

A **jet** is a narrow cone of hadrons and other particles produced by the hadronization of a quark or gluon in a particle physics or heavy ion experiment. Particles

Jet/Event as a point cloud





Point cloud

From Wikipedia, the free encyclopedia

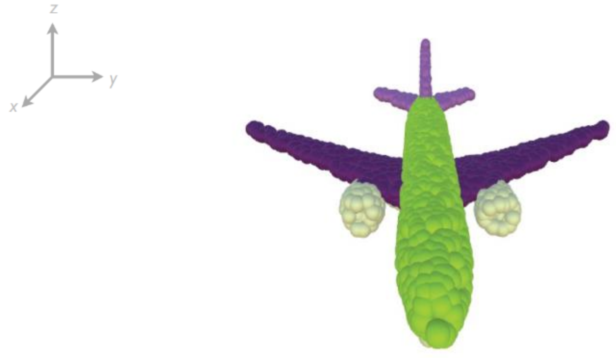
A **point cloud** is a set of data points in space. The points may represent a 3D shape or object. Each point position has its set of Cartesian coordinates (X, Y, Z).[1] Point clouds are generally produced by 3D scanners or by photogrammetry software, which measure many points on the external surfaces of objects around them. As the output of 3D

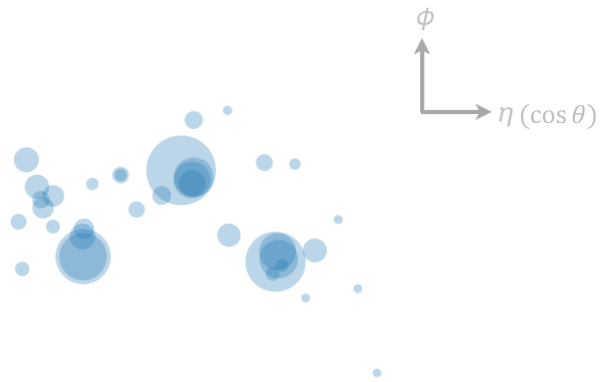
Jet (Particle cloud)

From Wikipedia, the free encyclopedia

A **jet (particle cloud)** is a set of particles in space. Particle clouds are generally created by clustering a large number of particles measured by particle detectors, e.g., ATLAS and CMS. which measure

Point clouds VS Particle clouds





- Point cloud
  - points are intrinsically ***unordered***
  - points are distributed in space
    - spatial coordinates (3D xyz) encode geometric structure information

- Particle cloud
  - particles are intrinsically ***unordered***
  - particles are distributed in space
    - spatial distribution (2D coordinates in the η(cos θ)-φ space) reflects radiation patterns

But particles have more features:
- energy/momenta/displacement/particle ID/etc.
- more interesting than a plain point cloud!

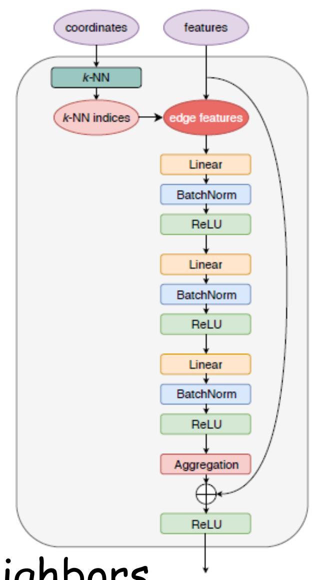
The architecture of ParticleNet

H. Qu and L. Gouskos [*Phys.Rev.D 101 (2020) 5, 056019*]

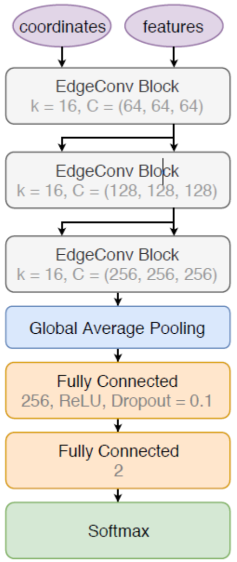
ParticleNet

- customized graph neural network architecture for jet tagging with the point cloud approach, based on Dynamic Graph CNN (DGCNN)
  [Y. Wang et al., arXiv:1801.07829]
- explicitly respects the permutation symmetry of the point cloud

EdgeConv block



ParticleNet architecture



Key building block: EdgeConv

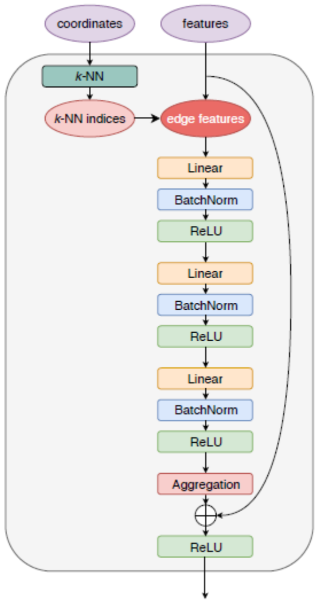
- treating a point cloud as a graph: each point is a vertex
  - for each point, a local patch is defined by finding its k-nearest neighbors
- designing a permutation-invariant "convolution" function
  - define "edge feature" for each center-neighbor pair: $e_{ij} = h_\Theta(x_i,\ x_{i_j}) = \bar{h}_\Theta(x_i,\ x_{i_j} - x_i)$
    - same h_Θ for all neighbor points, and all center points, for symmetry
  - aggregate the edge features in a symmetric way: $x_i' = \square_{j=1}^{k} h_\Theta(x_i,\ x_{i_j}) = \frac{1}{k}\sum h_\Theta(x_i,\ x_{i_j})$

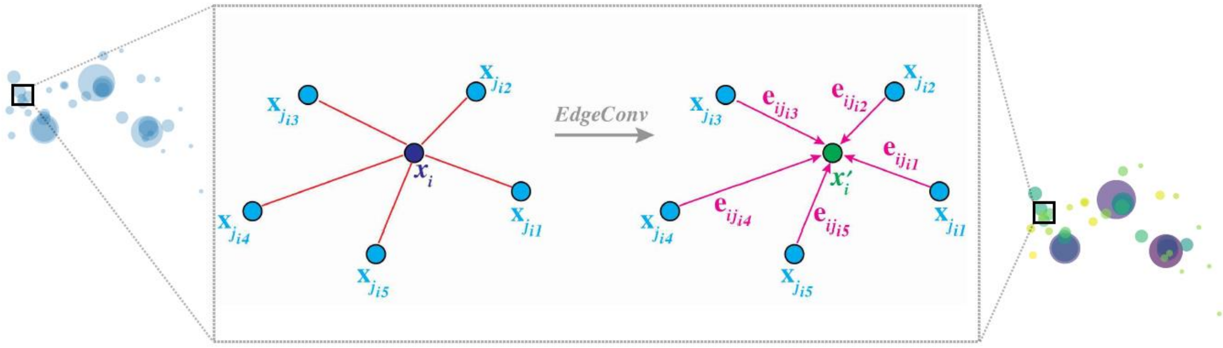
The architecture of ParticleNet

H. Qu and L. Gouskos [*Phys.Rev.D 101 (2020) 5, 056019*]

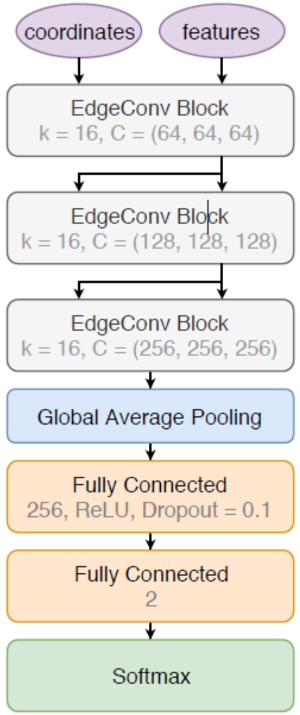
- EdgeConv can be stacked to form a deep network
  - learning both local and global structures, in a hierarchical way

EdgeConv block





ParticleNet architecture



Performance of ParticleNet

- Performance comparison on the top tagging benchmark dataset.

	Accuracy	AUC	$1/\varepsilon_b$ at $\varepsilon_s = 50\%$	$1/\varepsilon_b$ at $\varepsilon_s = 30\%$
ResNeXt-50	0.936	0.9837	302 ± 5	1147 ± 58
P-CNN	0.930	0.9803	201 ± 4	759 ± 24
PFN	...	0.9819	247 ± 3	888 ± 17
ParticleNet-Lite	0.937	0.9844	325 ± 5	1262 ± 49
ParticleNet	**0.940**	**0.9858**	$\mathbf{397 \pm 7}$	$\mathbf{1615 \pm 93}$

- Performance comparison on the quark-gluon tagging benchmark dataset.

	Accuracy	AUC	$1/\varepsilon_b$ at $\varepsilon_s = 50\%$	$1/\varepsilon_b$ at $\varepsilon_s = 30\%$
ResNeXt-50	0.821	0.8960	30.9	80.8
P-CNN	0.818	0.8915	31.0	82.3
PFN	...	0.8911	30.8 ± 0.4	...
ParticleNet-Lite	0.826	0.8993	32.8	84.6
ParticleNet	0.828	0.9014	33.7	85.4
P-CNN (w/ PID)	0.827	0.9002	34.7	91.0
PFN-Ex (w/ PID)	...	0.9005	34.7 ± 0.4	...
ParticleNet-Lite (w/ PID)	0.835	0.9079	37.1	94.5
ParticleNet (w/ PID)	**0.840**	**0.9116**	$\mathbf{39.8 \pm 0.2}$	$\mathbf{98.6 \pm 1.3}$

A Brief Introduction on ParticleTransfomer

Attention & Self-Attention

- **Attention**
- ***Attention*** is a very broad concept.
- Background: as the model grows, computing resources become increasingly strained. How to better allocate the limited computing resources by importance? -> ***Attention***

- ***Attention mechanism – general case***
- **STEP1:** Calculate the attention distribution on the input information to obtain the importance distribution of different input information, i.e. different weights
- **STEP2:** Calculate the weighted average of the current input information according to the different importance, in order to achieve for more efficient use of computing resources

Attention & Self-Attention

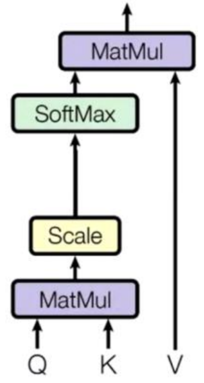
- **Calculate the attention distribution**
- For N input vectors: $[x_1, \dots, x_N]$, to pick out information that is important to one's goal, one needs to introduce a representation of the goal -> ***query vector (q)***.
- Then the problem turn into investigating correlations between different inputs and the query vector.
- A simple approach: using ***attention scoring function:***
  - additive attention: $s(x,q) = v^T tanh(Wx + Uq)$
  - dot-product attention: $s(x,q) = x^T q$
  - scaled dot-product attention: $s(x,q) = \frac{x^T q}{\sqrt{D}}$
  - bilinear attention: $s(x,q) = x^T W q$
- Weights: $softmax(s(x,q))$
- **Calculate the weighted average**
- Soft attention: $att(X,q) = \sum_{n=1}^{N} a_n x_n$
- Hard attention: focus on one input vector only –> non-differentiable (cannot use BP)
  - pick vector with the largest weight
  - random sampling on attention distribution

Attention & Self-Attention

- **Mutations of attention mechanism**

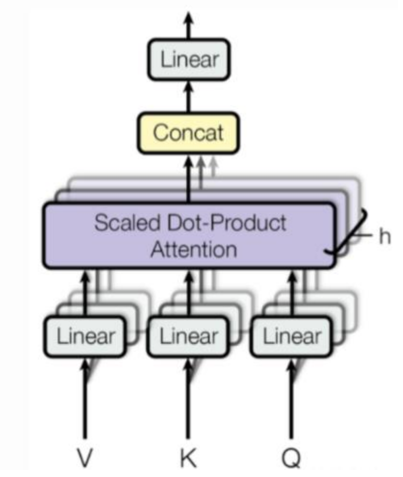
- ***Key-Value Pair Attention***
- inputs are k-v pairs $(\mathrm{K}, \mathrm{V}) = [(k_1, v_1), \ldots, (k_N, V_N)]$
- attention:

$$att((K, V), q) = \sum_{n=1}^{N} softmax(s(q, k_n))v_n$$



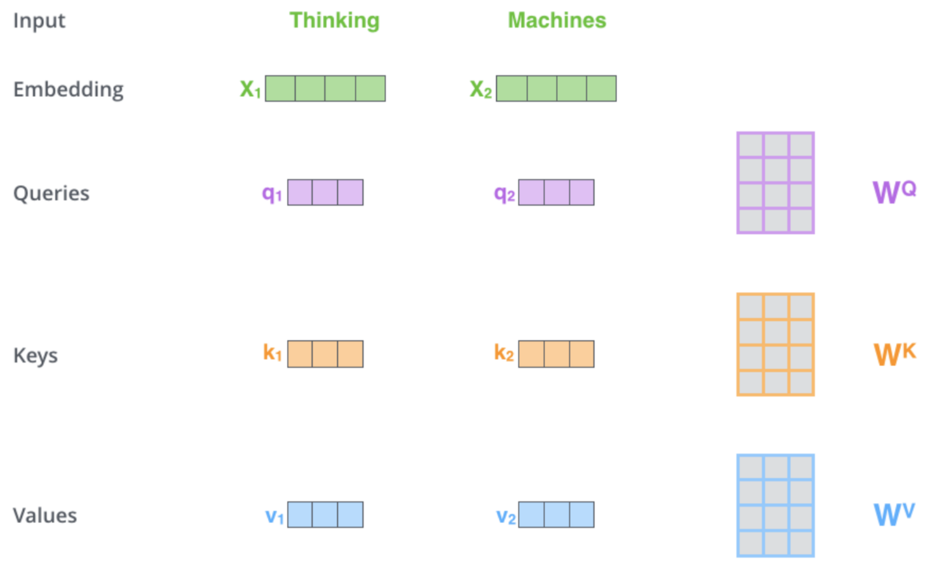
- ***Multi-Head Attention***
- multiple queries, $\mathrm{Q} = [q_1, \ldots, q_M]$
- search for information from inputs in a parallel way
- attention:

$$att((K, V), Q) = att((K, V), q_1) \oplus \ldots \oplus att((K, V), q_M)$$



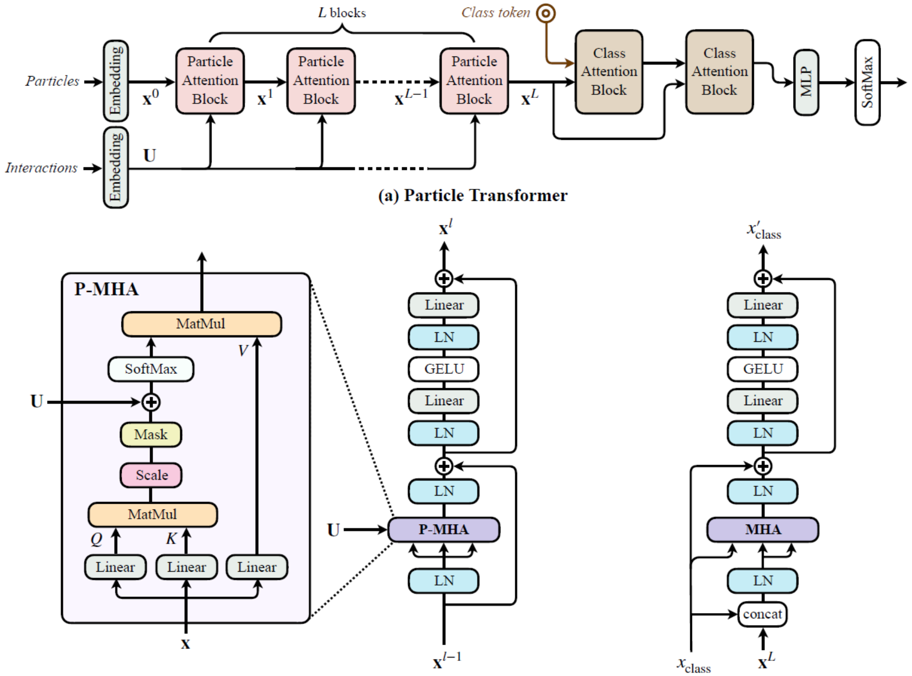
Attention & Self-Attention

- **Self-Attention**
- ***Query-Key-Value (QKV) Attention***
- Q, K, V are identical (Q=K=V=X) / from the same origin:
- input: $\boldsymbol{X} = [x_1, \ldots, x_N]$
- attention: $att(\boldsymbol{X}) = softmax(s(\boldsymbol{X}))\boldsymbol{X}$
- or
- $\boldsymbol{Q} = \boldsymbol{W}_q\boldsymbol{X}, \boldsymbol{K} = \boldsymbol{W}_k\boldsymbol{X}, \boldsymbol{V} = \boldsymbol{W}_v\boldsymbol{X}$
- attention: $att\big((\boldsymbol{K},\boldsymbol{V}),\boldsymbol{Q}\big) = softmax(s(\boldsymbol{Q},\boldsymbol{K}))\boldsymbol{V}$



The architecture of ParticleTransformer

H. Qu , C. Li, S. Qian [2202.03772]



(a) Particle Transformer

(b) Particle Attention Block

(c) Class Attention Block

image credit

- ParticleTransformer
  - Transformer designed for particle physics
  - TWO sets of inputs
    - *Particle*: Features of every single particle
    - *Interaction*: Pair-wise features
- Particle Attention Block
  - Multi-Head Attention (MHA) Module
  - Pair-wise feature are introduced as the attention mask (P-MHA)
- Class Attention Block
  - Multi-Head Attention (MHA) Module
  - Class token is used for the MHA calculation

Choice of the pair-wise features: from LundNet

$$\text{P-MHA}(Q,K,V) = \text{SoftMax}(QK^T/\sqrt{d_k} + \mathbf{U})V$$

d_k: dimension of K

$$\Delta = \sqrt{(y_a - y_b)^2 + (\phi_a - \phi_b)^2}$$
$$k_T = \min(p_{T,a}, p_{T,b}) \cdot \Delta$$
$$z = \min(p_{T,a}, p_{T,b})/(p_{T,a} + p_{T,b})$$
$$m^2 = (E_a + E_b)^2 - ||\mathbf{p}_a + \mathbf{p}_b||^2$$

$$\text{MHA}_C(Q_C, K_C, V_C) = \text{SoftMax}(Q_C K_C^T / \sqrt{d_{kC}})V_C$$

$Q_C = W_{qC} x_{\text{class}} + b_{qC}$ $K_C = W_{kC}\mathbf{z} + b_{kC}$ $V_C = W_{vC}\mathbf{z} + b_{vC}$ d_{kC}: dimension of K_C

$$\mathbf{z} = [x_{\text{class}}, \mathbf{x}^L]$$

Concatenate class information and particle embedding

S. Qian@ML4Jets2022

Official implementation of "Particle Transformer for Jet Tagging". (github.com)

Performance of ParticleNet

- Jet tagging performance on the JETCLASS dataset.

	All classes		$H \to b\bar{b}$	$H \to c\bar{c}$	$H \to gg$	$H \to 4q$	$H \to \ell\nu qq'$	$t \to bqq'$	$t \to b\ell\nu$	$W \to qq'$	$Z \to q\bar{q}$
	Accuracy	AUC	$\mathrm{Rej}_{50\%}$	$\mathrm{Rej}_{50\%}$	$\mathrm{Rej}_{50\%}$	$\mathrm{Rej}_{50\%}$	$\mathrm{Rej}_{99\%}$	$\mathrm{Rej}_{50\%}$	$\mathrm{Rej}_{99.5\%}$	$\mathrm{Rej}_{50\%}$	$\mathrm{Rej}_{50\%}$
PFN	0.772	0.9714	2924	841	75	198	265	797	721	189	159
P-CNN	0.809	0.9789	4890	1276	88	474	947	2907	2304	241	204
ParticleNet	0.844	0.9849	7634	2475	104	954	3339	10526	11173	347	283
ParT	**0.861**	**0.9877**	**10638**	**4149**	**123**	**1864**	**5479**	**32787**	**15873**	**543**	**402**
ParT (plain)	0.849	0.9859	9569	2911	112	1185	3868	17699	12987	384	311

- Number of trainable parameters and FLOPs

	Accuracy	# params	FLOPs
PFN	0.772	86.1 k	4.62 M
P-CNN	0.809	354 k	15.5 M
ParticleNet	0.844	370 k	540 M
ParT	**0.861**	2.14 M	340 M
ParT (plain)	0.849	2.13 M	260 M

similar computation complexity with ParticleNet, but more performant than ParticleNet

How to run ParticleNet/ParticleTransformer using Weaver

- First thing to do:
  Login to IHEP cluster and do:

```
cp -r /scratchfs/bes/wangshudong/particle_transformer/ /PATH/TO/YOUR/SPACE/particle_transformer/
```

Try it yourself!

- **Weaver**
  - *Weaver* aims at providing a streamlined yet flexible machine learning R&D framework for high energy physics (HEP) applications. (Github: hqucms/weaver-core)
- **Set up your environment (you can use mine)**
  - Install Miniconda (if you don't already have it)
  - Set up a conda environment and install the required packages
  - On IHEP cluster, simply type commands below to use my conda env (you don't even need to do this):

```
#this conda env only support training using CPU, since most of you
#don't have access to GPU cluster
source "/cefs/higgs/wangshudong/miniconda3/etc/profile.d/conda.sh"
conda activate weaver-core
```

- **Prepare your configuration files**

  To train a neural network using *Weaver*, you need to prepare:
  - A YAML data configuration file describing how to process the input data.
  - A python model configuration file providing the neural network module and the loss function.
  - ***Let's move to codes now***

Try it yourself!

- **Start running! (general case)**
  - The `weaver` command is the top-level entry to run for training a neural net, getting prediction from trained models, and exporting trained models to ONNX for production. The corresponding script file is weaver/train.py. To check all the command-line options for `weaver`, run `weaver -h`
  - Examples for training, inference and model exportation are shown below:

- **Training**

```
weaver --data-train '/path/to/train_files/*/*/*/*/output_*.root' \
--data-test '/path/to/train_files/*/*/*/*/output_*.root' \
--data-config data/ak15_points_pf_sv.yaml \
--network-config networks/particle_net_pf_sv.py \
--model-prefix /path/to/models/prefix \
--gpus 0,1,2,3 --batch-size 512 --start-lr 5e-3 --num-epochs 20 --optimizer ranger \
--log logs/train.log
```

How to run ParticleNet

- **Prediction/Inference**

```
weaver --predict --data-test '/path/to/test_files/*/*/*/*/output_*.root' \
 --data-config data/ak15_points_pf_sv.yaml \
 --network-config networks/particle_net_pf_sv.py \
 --model-prefix /path/to/models/prefix_best_epoch_state.pt \
 --gpus 0,1,2,3 --batch-size 512 \
 --predict-output /path/to/output.root
```

- **Model exportation**

```
weaver -c data/ak15_points_pf_sv.yaml -n networks/particle_net_pf_sv.py -m
 /path/to/models/prefix_best_epoch_state.pt --export-onnx model.onnx
```

Try it yourself!

- Start running! (for this tutorial only)

```
cd /PATH/TO/YOUR/SPACE/particle_transformer/
source train_test.sh #run on login node
```

Then just wait!

Try it yourself!

- **Dataset**

  - The dataset prepared for today's tutorial:
  - MC samples: follow previous note [ATL-PHYS-PUB-2021-029]
  - Signal: boosted W/Z bosons from simulated W ’→WZ (→ $q\bar{q}q\bar{q}$) events with $m_{W'}$ = 2 TeV, Pythia8 + NNPDF2.3LO + A14 tune.
  - Bkg: QCD di-jet events @ LO, Pythia8 + NNPDF2.3LO + A14 tune.
  - Large-R jets are reconstructed from UFOs using the anti-kt algorithm implemented in the FastJet package with the radius parameter R = 1.0.
  - Jet reconstruction, grooming, and truth labeling is identical to the previous work.
  - The samples contain the flat ntuple (i.e. 1 jet / entry). 4 vector (E, p_T, η, ϕ) of the jet constituents are stored.

Jet requirements	W jet requirements	Z jet requirements
Jet $\|\eta\| < 2.0$ Jet $p_{\mathrm{T,truth}} > 200$ GeV Number of constituents $\geq$ 2 Jet mass > 40 GeV	dR(truth jet, MC truth W) < 0.75 Ungroomed truth jet mass > 50 GeV Number ghost associated b-hadrons == 0 Truth jet $\sqrt{d_{12}}$ > 55.25×exp(-2.34×10^{-3} × Jet $p_{\mathrm{T,truth}}$)	dR(truth jet, MC truth Z) < 0.75 Ungroomed truth jet mass > 50 GeV Truth jet $\sqrt{d_{12}}$ > 55.25×exp(-2.34×10^{-3} × Jet $p_{\mathrm{T,truth}}$)

Try it yourself!

- **Dataset**
  - The dataset prepared for today's tutorial:
  - **Training Variables**
  - $\Delta\eta$ Difference in pseudo-rapidity between the particle and the jet axis
  - $\Delta\phi$ Difference in azimuthal angle between the particle and the jet axis
  - $\ln p_{\mathrm{T}}$ Logarithm of the particle's $\mathrm{p_T}$
  - $\ln E$ Logarithm of the particle's energy
  - $\ln \frac{p_{\mathrm{T}}}{\sum_{\mathrm{jet}} p_{\mathrm{T}}}$ Logarithm of the particle's $\mathrm{p_T}$ relative to the total $\mathrm{p_T}$ in jet
  - $\ln \frac{E}{\sum_{\mathrm{jet}} E}$ Logarithm of the particle's energy relative to the total energy in jet
  - ΔR Angular separation between the particle and the jet axis $\sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}$
  - $(\mathrm{E}, \mathrm{p_x}, \mathrm{p_y}, \mathrm{p_z})$ 4-momentum. (only used by ParticleTransformer because it requires this certain form of input)

Try it yourself!

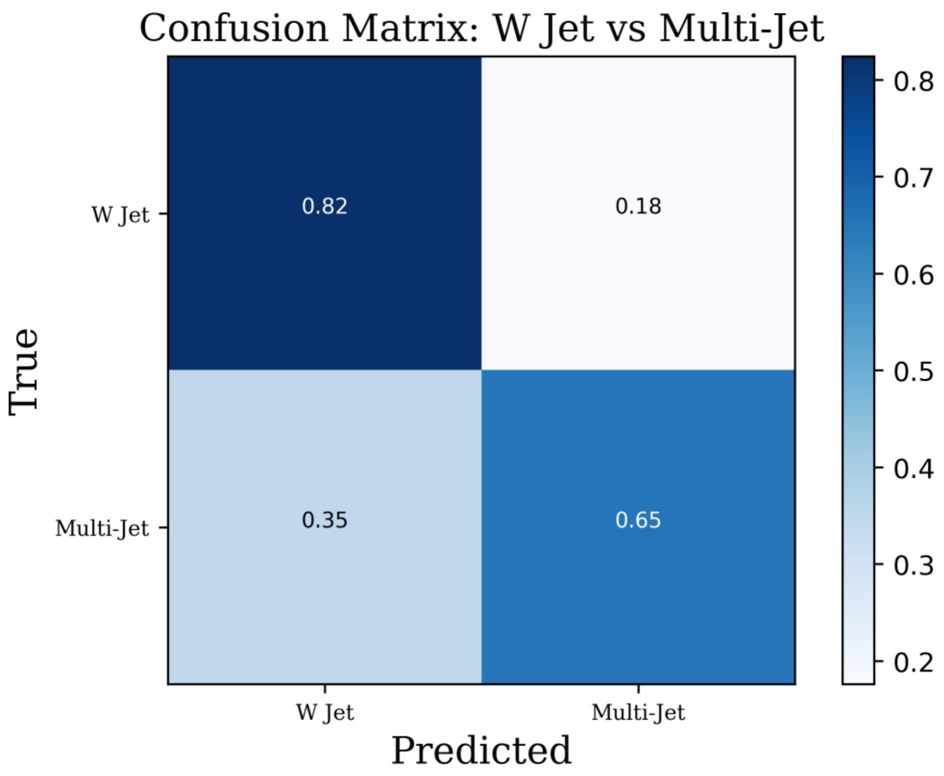
- **Plot a confusion matrix**

  - confusion matrix:

```
#in weaver conda environment
cd /PATH/TO/YOUR/SPACE/particle_transformer/pltCM
#copy confusion matrix from log and paste it in
#pltCM.py and add some commas
python plotCM.py
```

```
CM = np.array( [[0.824, 0.176],
                [0.352, 0.648]] )
```

  - Result:



Try it yourself!

- **Plot ROC curves**
  - ROC curves:

```
#deactivate conda environment
cd /PATH/TO/YOUR/SPACE/particle_transformer/pltROC
vim plotROC.C #modify input file name
#use default root environment
root plotROC.C
```

  - Result: