Caloutils

PointCloud-based libery for Generative Models for Calorimeter Showers

Moritz Scham*, Benno Käch,
Dirk Krücker, Simon Schnake

ML4Jets, 09.11.23

*Funded by Helmholtz AI – grant number: ZT-I-PF-5-3

The DeGeSim Project

HELMHOLTZ AI
ARTIFICIAL INTELLIGENCE
COOPERATION UNIT

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

RWTH AACHEN UNIVERSITY

JÜLICH Forschungszentrum
JÜLICH SUPERCOMPUTING CENTRE

DESY.

Why PointClouds?

Idea: Represent Calorimeter showers as 4D point clouds (PCs)

- Deals with sparsity
- Handels irregular geometry
- Data size scaling dependence from cells to hits
  (⇒ better with higher sparsity / in higher granularity)
- Makes the model architecture independent of the detector

Problems:

- Grid ↔ PC
- Metrics?
- Missing ML tools for models producing PCs

⇒ `caloutils` package

`caloutils` Package

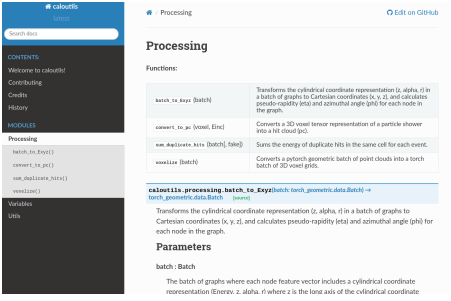
PC–based metrics and tools for generative models for calorimeter showers.

- Aims to be calorimeter independent.
  - Detector geometry specific code currently targeted at the CaloChallange Dataset, but easily extendable.
  - Reference implementations
- PCs can have arbitrary size

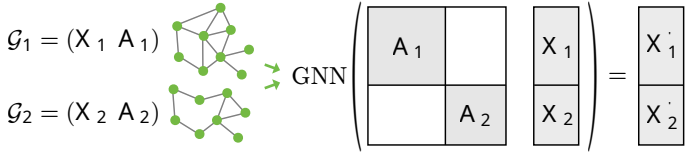
⇒ Ragged Tensors

⇒ Based on `PyTorch Geometric` and `PyTorch Scatter`

- Available on GitHub / ReadTheDocs / PyPI



PyG's Approach to Ragged Batches



http://rusty1s.github.io/pyg_slides.pdf

```
graph1 = Data(x=torch.tensor([[-1], [0.], [1.]]))
graph2 = Data(x=torch.tensor([[2.], [2.]]))
batch = Batch.from_data_list([graph1,graph2]) # Slow
print(batch)
> Batch(x=[5, 1], batch=[5], ptr=[3])
print(batch.batch)
> tensor([0, 0, 0, 1, 1])
```

Fast mapping: Grid → PC

```
showers = torch.tensor(
  [ # Events
    [ # Layers
        [[0, 7, 0, 0], [0, 0, 0, 0], [0, 0, 0, 3]],# Rows&Columns
        [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]],
    ],
    [
        [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]],
        [[0, 0, 0, 0], [0, 2, 0, 0], [0, 0, 0, 0]],
    ],
  ]
)
batch = caloutils.processing.voxel_to_pc(
    showers,
    Einc=torch.tensor([30, 10]),
)
print(batch.x, batch.y,batch.batch)
> ([[7, 0, 0, 1], ([[30, 10],[ 2,  1]]) ([0, 0, 1])
    [3, 0, 2, 3],
    [2, 1, 1, 1]])
```

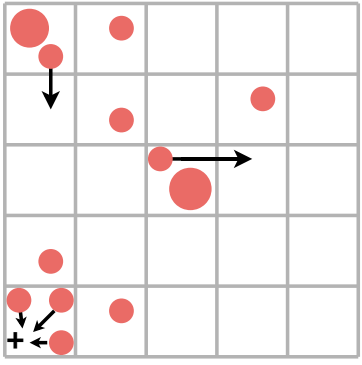
- x: Cell Energy, Index
- y: Shower energy/number of hits
- batch: What event do the hits belong to?

Fast mapping: Grid ← PC

```
caloutils.processing.pc_to_voxel(batch)
> torch.tensor(
  [ # Events
    [ # Layers
        [[0, 7, 0, 0], [0, 0, 0, 0], [0, 0, 0, 3]],# Rows&Columns
        [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]],
    ],
    [
        [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]],
        [[0, 0, 0, 0], [0, 2, 0, 0], [0, 0, 0, 0]],
    ],
  ]
)
```

Multiple Hits

Problem: Multiple Hits mapped to the same cell

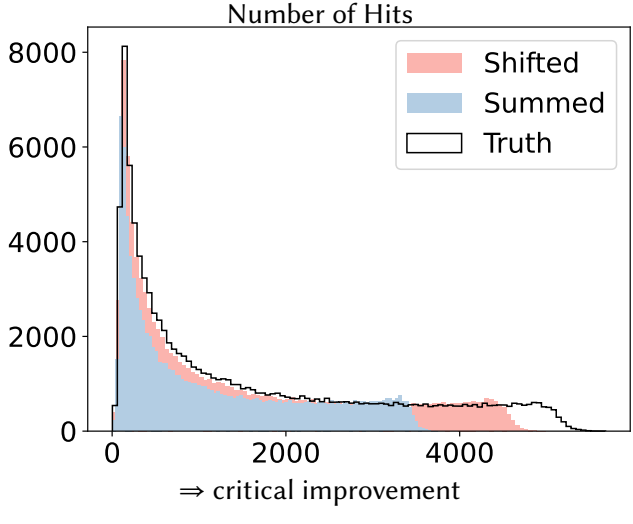


1. Select cells with multiple hits
2. Run over multihits, in order of energy, skip the highest
3. Check if the neighboring cells in random order
   ⇒ Shift if empty
4. Repeat until no multihit/no empty cells
5. Sum remaining multihits

- `sum_multi_hits` + `shift_multi_hits` = `shift_sum_multi_hits`
- Needs calorimeter dependent mapping
  - x,y,z → cell id
  - cell id → neighbors

Effect of Shifting Hits – CaloChallange Dataset 2

On Benno Käch's MDMA model



Variables + Distances = Metrics

Variables:

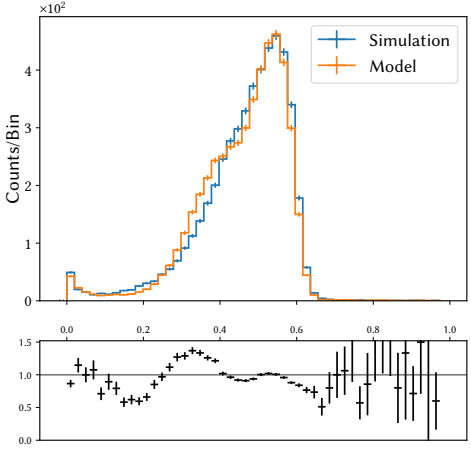
- `sphereratio` /`cyratio` : Energy content in a sphere/cylinder
- `analyze_layers` : Peak energy layer, FWHM turnon/turnoff layer
- `first_principal_components` : PCAs

Distances:

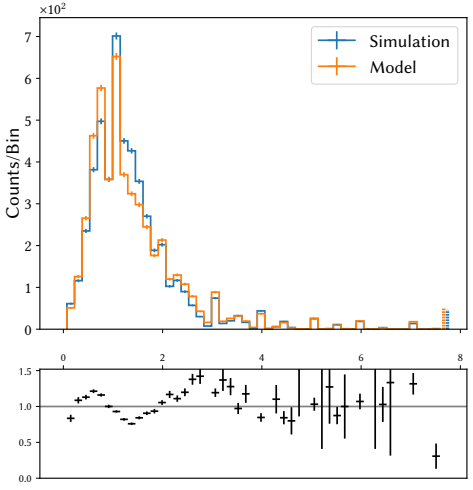
- `scaled_w1_distance` :
  1. Scale both samples to normal with μ, σ of the simulated sample
  2. Wasserstein-1 Distance
- `hist_distance` : Calculate a distance metric between two 1D samples using the histcdf function.
  1. Calculate bin ranges depending on the simulated sample
  2. Compute histogram of both samples
  3. L1 distance between the histograms
- `energy_distance` / `wasserstein_distance` (torch copies of sklearn)

Shower variables

$$\frac{\sum_i^{\text{Sphere}(\sigma=0.3)} E_i}{\sum_i^{\text{Sphere}(\sigma=0.8)} E_i}$$



$$\frac{|\text{Layer}^{\text{Peak}} - \text{Layer}^{\text{Turnoff}}| + 1}{|\text{Layer}^{\text{Peak}} - \text{Layer}^{\text{Turnon}}| + 1}$$



PyG Batch Utils

- PyG batches:
  - Creating: `Batch.from_data_list(batch_list)`
    - Ideal: Initialize the batch first, then add the features already stacked.
  - Added tensors doesn't survive indexing
  - No slicing
  - No concatenation

Extending PyG Batches

Creation ✅

```
batchidx = [0,0,0,1,1,2,2,2,2,2,3,3]
batch = init_batch(batchidx)
add_node_attr(batch, "x", torch.vstack(x_list))
```

Concatenation ✅

```
batch = Batch.from_data_list(batch_list)
batch = from_batch_list(
    Batch.from_data_list(batch_list[:3]),
    ↪ Batch.from_data_list(batch_list[3:])
)
```

Augmentation ✅

- `add_node_attr`
- `add_graph_attr`
- `set_edge_attr`
- `set_edges`

Slicing 🛠️

To be integrated in PyG, join the discussion @
`https://github.com/pyg-team/pytorch_geometric/issues/8226`

A lot of tools to make generative models on PCs more convenient!

`https://github.com/DeGeSim/caloutils`

Thank you!

DESY. Deutsches
Elektronen-Synchrotron
www.desy.de

Moritz Scham*, Benno Käch,
Dirk Krücker, Simon Schnake
moritz.scham@desy.de