# Electromagnetic-shower simulation with Graph Neural Networks

Vladislav Belavin, Andrey Ustyuzhanin

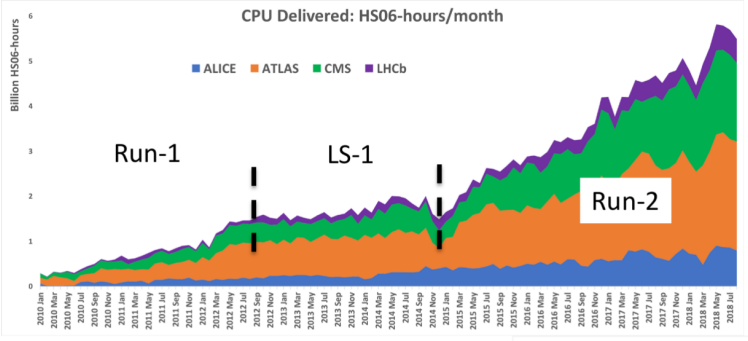
National Research University Higher School of Economics

March 12, 2019

# Motivation

# Simulation in HEP

> "Full MC simulation occupies 50-70% of the experiments worldwide computing resources, equivalent to billions of CPU hours per year" https://arxiv.org/pdf/1712.10321.pdf



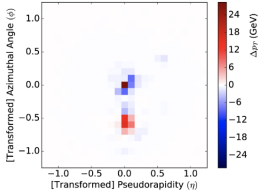

https://indico.cern.ch/event/744723/contributions/3098851/attachments/1713841/2764487/WLCG-LHCC-12-09-2018.pdf

SCHOOL OF DATA ANALYSIS

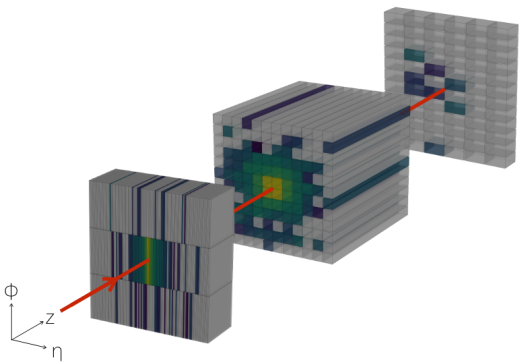
# Simulation in HEP: towards fast simulation

Handful of examples of successive examples of speed up of simulation with DL without any (serious) loss in quality.

**LAGAN for Jet Images**




**CaloGAN for calorimeter response**



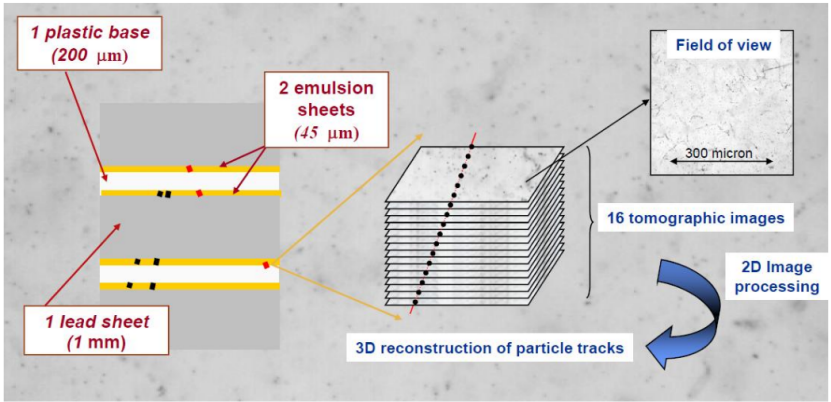

SCHOOL OF DATA ANALYSIS

# Towards fast simulation of EM showers in emulsions cloud chambers

We propose an approach for shower generation on **track level** in Emulsion Cloud Chamber target units(”ECC Brick”), a sandwitch metal-emulsion detector.

# Towards fast simulation of EM showers in emulsions cloud chambers

When particle passes through the brick it left traces in emulsion films glued on both sides of plastic base. After scanning emulsions with microscope it is possible to reconstruct these basetracks(or shower segments).




SCHOOL OF DATA ANALYSIS

# Towards fast simulation of EM showers in emulsions cloud chambers

Data consists of array of tracks. Each track is described by 5 variables: $x, y, z, \theta_x, \theta_y$, i.e. position and direction. **We are trying to simulate these beautiful showers!**

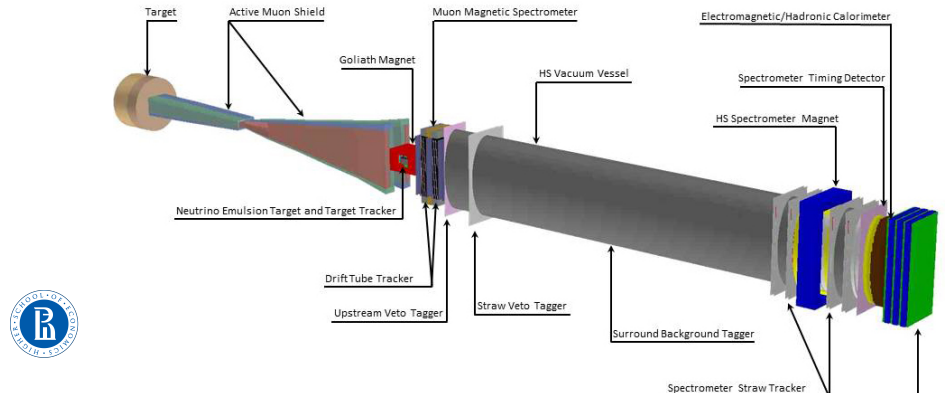


SCHOOL OF DATA ANALYSIS

# Source of data: FairShip & SHiP Experiment

All EM-showers in ECC Bricks for training were generated with open source software framework FairShip.

SHiP experiment is designed for search for hidden particles and suited to study neutrino and anti-neutrino physics.




SCHOOL OF DATA ANALYSIS

# Key differences with current works on simulation of calorimeters responses

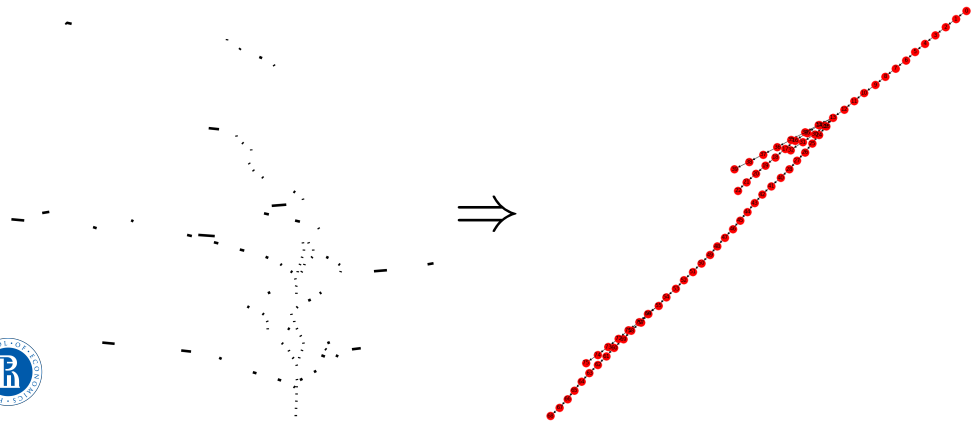
- simulation on track level instead of simulation of responses of calorimeter cells;
- a lot of degrees of freedom: even number of degrees of freedom is also a degree of freedom ;)
- variable size of output(technical difficulty).

# Proposed methodology

# Problem reformulation in ML terminology

We construct graph, where each node represents one track in emulsion based on physical history of shower development.

Then we are reformulating our problem as **graph generation** with **signal generation on graph**.



# Graph generation

# Graph generation: challenges

**Challenges:**

- graph generation requires to make $\sim O\left((\text{number\_of\_nodes})^2\right)$ operation for adjacency matrix construction;

# Graph generation: challenges

Challenges:

- graph generation requires to make $\sim O\left((\text{number\_of\_nodes})^2\right)$ operation for adjacency matrix construction;
- robustness to permutation of nodes(i.e. rows and columns in adjacency matrix);

SCHOOL OF DATA ANALYSIS

# Graph generation: challenges

Challenges:

- graph generation requires to make $\sim O\left((\text{number\_of\_nodes})^2\right)$ operation for adjacency matrix construction;
- robustness to permutation of nodes(i.e. rows and columns in adjacency matrix);
- complex(physical) dependencies in graph structure.

# Graph generation: challenges

**Challenges:**

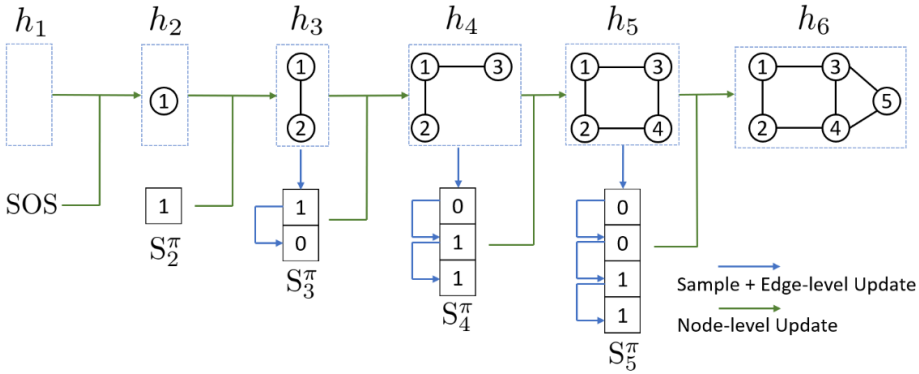
- graph generation requires to make $\sim O\left((\text{number\_of\_nodes})^2\right)$ operation for adjacency matrix construction;
- robustness to permutation of nodes(i.e. rows and columns in adjacency matrix);
- complex(physical) dependencies in graph structure.

**Solution(current):**

- GraphRNN

# GraphRNN: prediction of underlying graph

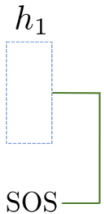
- Graph-level RNN: generates sequence of nodes;
- Edge-level RNN: generates sequence of edges for each node.



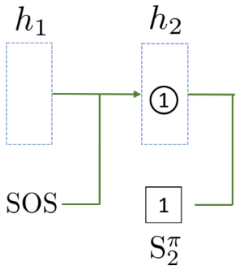

https://arxiv.org/pdf/1802.08773.pdf

SCHOOL OF DATA ANALYSIS

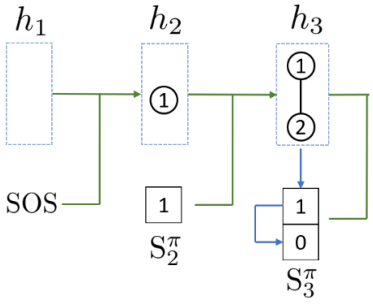
# GraphRNN: prediction of underlying graph



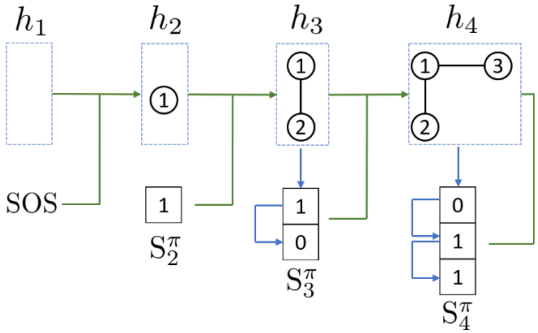
# GraphRNN: prediction of underlying graph



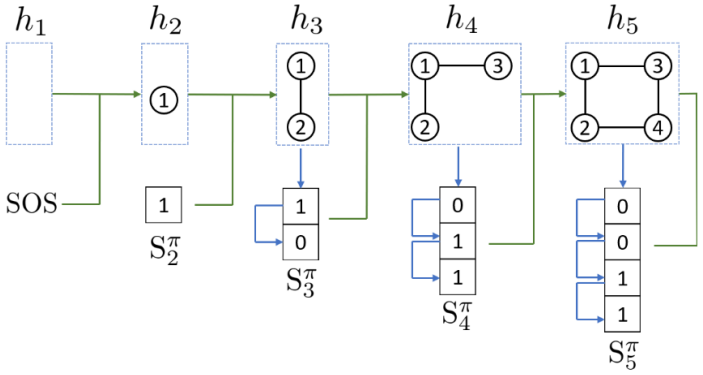
# GraphRNN: prediction of underlying graph



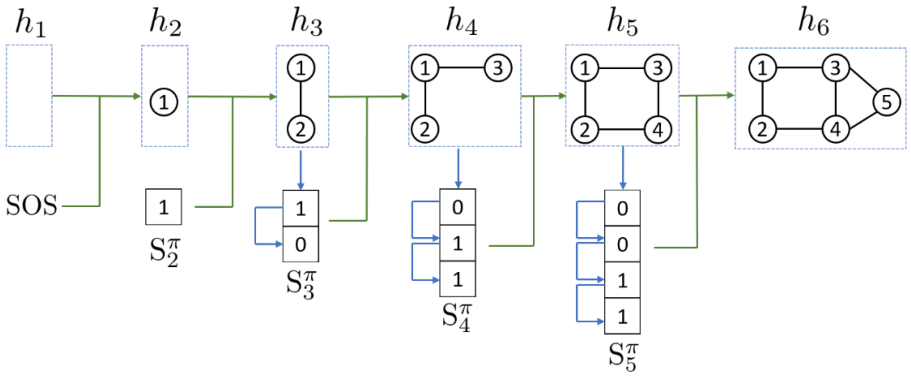
# GraphRNN: prediction of underlying graph



# GraphRNN: prediction of underlying graph



# GraphRNN: prediction of underlying graph



# GraphRNN: details on implementation

How do we order nodes?

# GraphRNN: details on implementation

How do we order nodes?

It is an important question because in the worst case scenario we need to train on all possible permutations of nodes($(\text{number\_of\_nodes})!$ permutation!).

SCHOOL OF DATA ANALYSIS

# GraphRNN: details on implementation

How do we order nodes?

It is an important question because in the worst case scenario we need to train on all possible permutations of nodes($(\text{number\_of\_nodes})!$ permutation!).

- order nodes in BFS(Breadth-first search) fashion, as it was proposed by authors of the original paper;
  - adjacency matrix become more block-diagonaly-ish;
    - $\Rightarrow$ reduce # of possible node sequences;
    - $\Rightarrow$ reduce # of edges to predict for each node;

# GraphRNN: details on implementation

How do we order nodes?

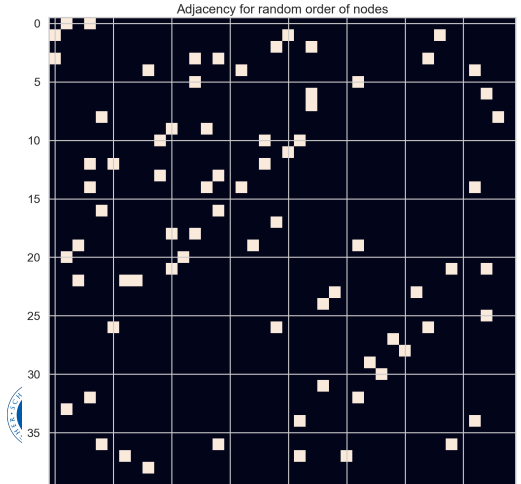
It is an important question because in the worst case scenario we need to train on all possible permutations of nodes($(\text{number\_of\_nodes})!$ permutation!).

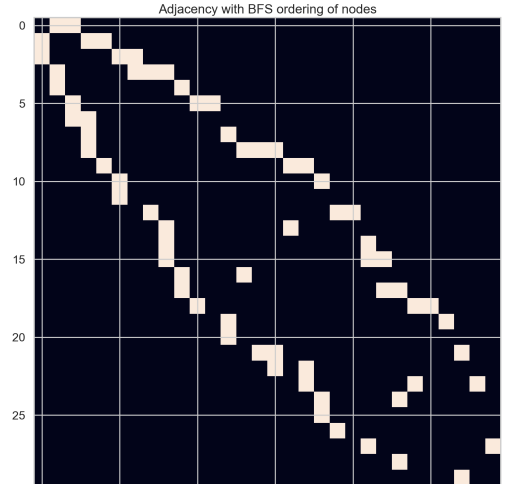
- order nodes in BFS(Breadth-first search) fashion, as it was proposed by authors of the original paper;
  - adjacency matrix become more block-diagonaly-ish;
    - $\Rightarrow$ reduce # of possible node sequences;
    - $\Rightarrow$ reduce # of edges to predict for each node;
- fix first node to be the root of the shower $\Rightarrow$ further reduce in # of node permutations;

SCHOOL OF DATA ANALYSIS

# GraphRNN: details on implementation

On left side – adjacency matrix of random graph with random node order, on the right side – adjacency matrix of this graph with nodes ordered in BFS manner.

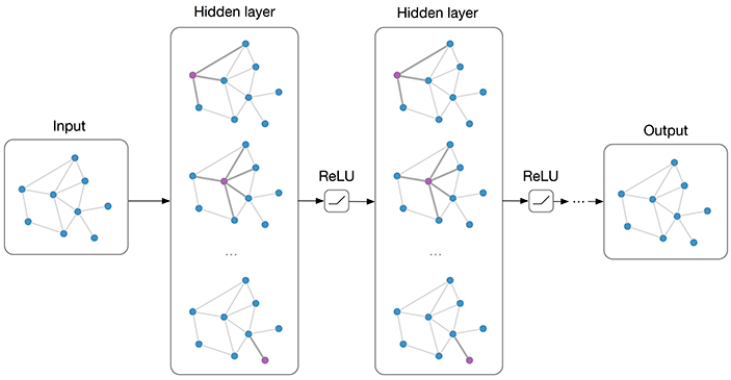


# Signal generation

# Graph Convolution Network for signal generation

Graph Convolution Network is used for generation of deep track embeddings(vector representations).



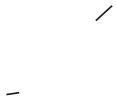

https://tkipf.github.io/graph-convolutional-networks/

SCHOOL OF DATA ANALYSIS

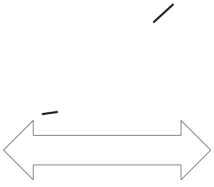
# Predictions of differences between tracks

Does it make sense to predict node/track features, i.e. $x, y, z, \theta_x, \theta_y$?

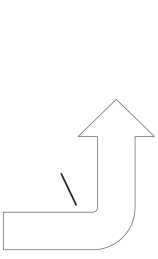
Orignal tracks



Translation **not** invariant



Rotation **not** invariant



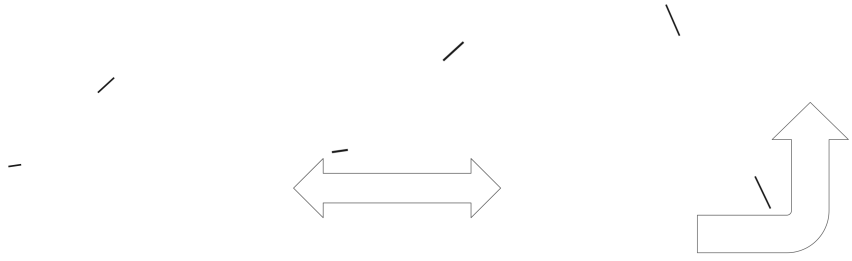
# Predictions of differences between tracks

Does it make sense to predict node/track features, i.e. $x, y, z, \theta_x, \theta_y$?

Orignal tracks

Translation **not** invariant

Rotation **not** invariant

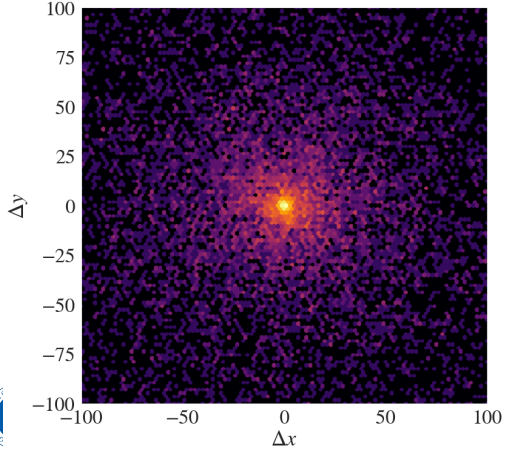


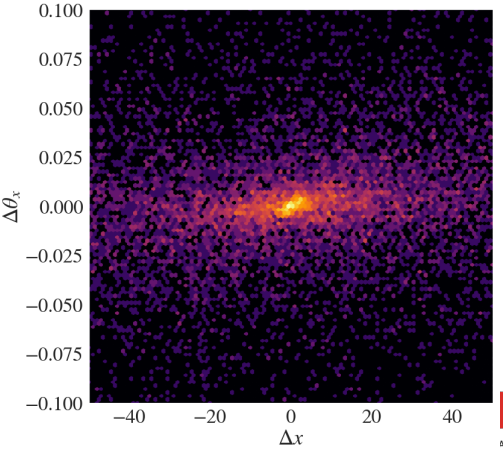
Better to predict differences in tracks coordinates, i.e. $\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y$

SCHOOL OF DATA ANALYSIS

# Predictions of distributions of differences between tracks

Even better(more natural due to stochastic nature of shower development) to predict **distributions** of differences in tracks coordinates.



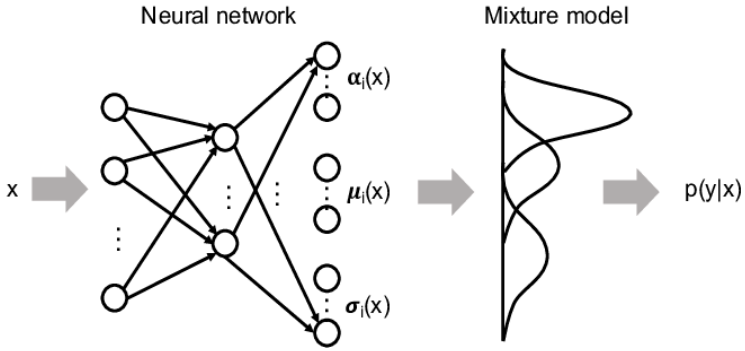




ATA ANALYSIS

# Mixture Density Network for edge predictions

At the end of GCN we have Mixture Density Networks that predicts mixture of normal distributions.

We are predicting distribution of $\prod_{(i,j)\sim\mathcal{E}} p_{ij}(\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y)$ for all edges in the graph.




https://publications.aston.ac.uk/373/1/NCRG_94_004.pdf

SCHOOL OF DATA ANALYSIS

# Training

Loss is a linear combination of

- logistic loss for edge predictions: $\sum y \log p + (1 - y) \log(1 - p)$;
- log-likelihood for MDN edge predictions: $\sum \log p_{ij}(\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y)$.

# Training

Loss is a linear combination of

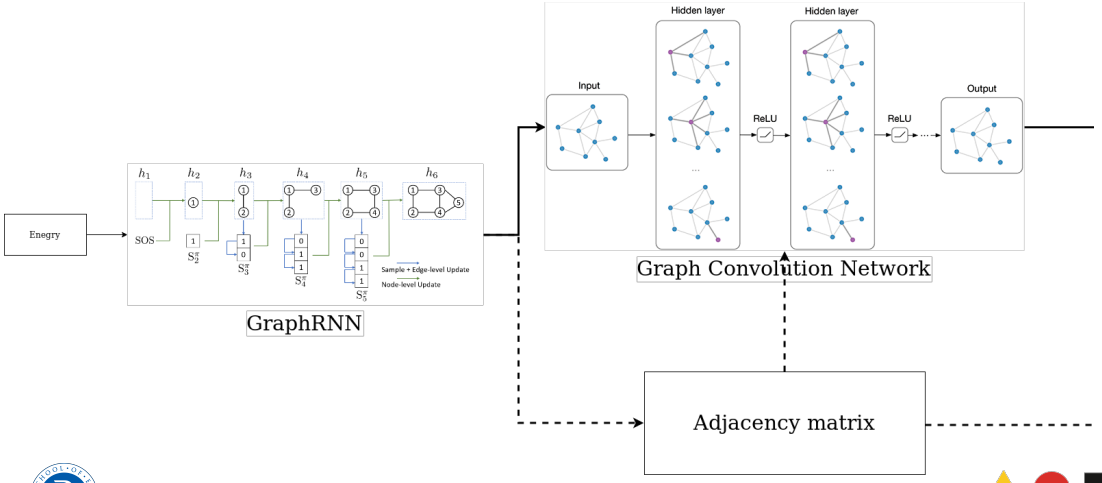
- logistic loss for edge predictions: $\sum y \log p + (1 - y) \log(1 - p)$;
- log-likelihood for MDN edge predictions: $\sum \log p_{ij}(\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y)$.

Training is done with

- 10000 showers in dataset;
- data augmentation presented by track dropout and random rotation around z-axis;
- 100(GraphRNN) + 100(GCN) epochs(trained seperately to prevent overfitting) in 4 hours.

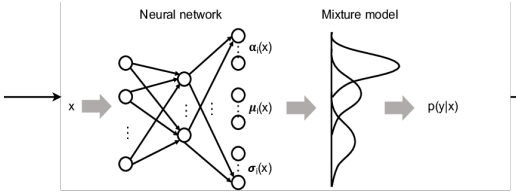
SCHOOL OF DATA ANALYSIS

# Final architecture




SCHOOL OF DATA ANALYSIS

# Final architecture



# Shower generation with trained model

$\text{track}_0 = (x_0, y_0, z_0, \theta_x^0, \theta_y^0) = (0, 0, 0, 0, 0)$

(1) Generate graph $G$ with GraphRNN;

(2) GCN with MDN predict $p_{ij}(\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y)$;

(3) $\forall$ edges $(i, j) \in G$:

  (1) sample $\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y \sim p_{ij}(\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y)$;

  (2) $\text{track}_j = \text{track}_i + [\Delta x, \Delta y, \Delta z, \Delta\theta_x, \Delta\theta_y]$;

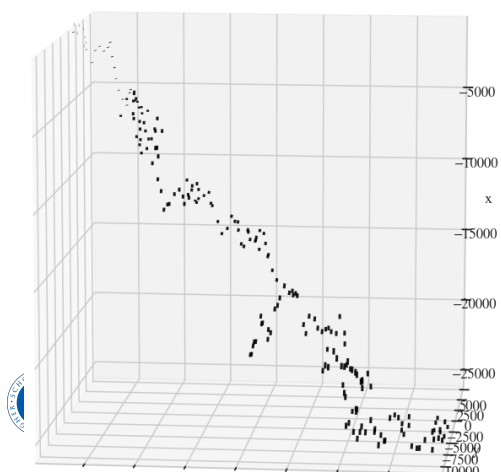
return tracks;

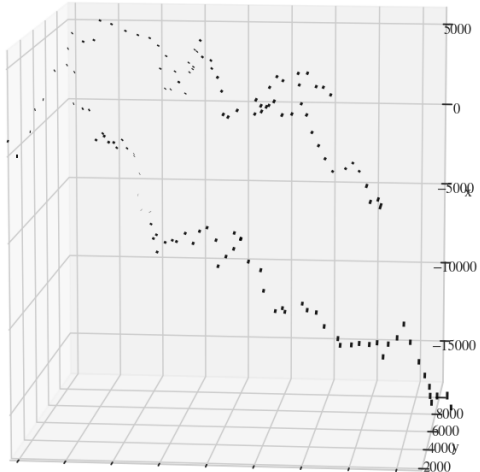
SCHOOL OF DATA ANALYSIS

# Evaluation

# Examples of generated showers

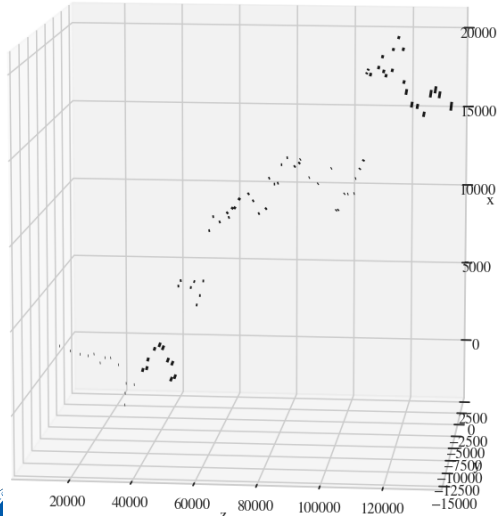
Model is able to capture shower development patterns, yet branching of the shower is suppressed(wip).

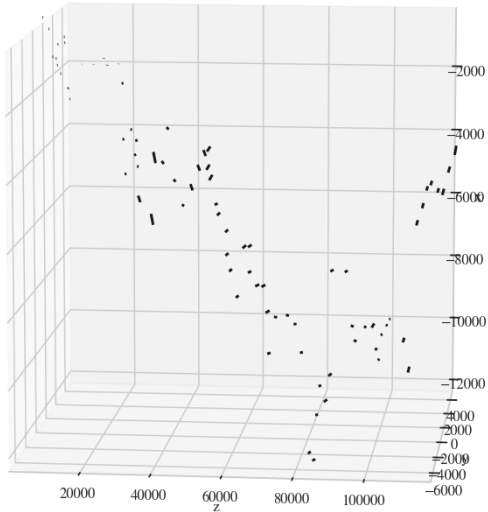



ATA ANALYSIS

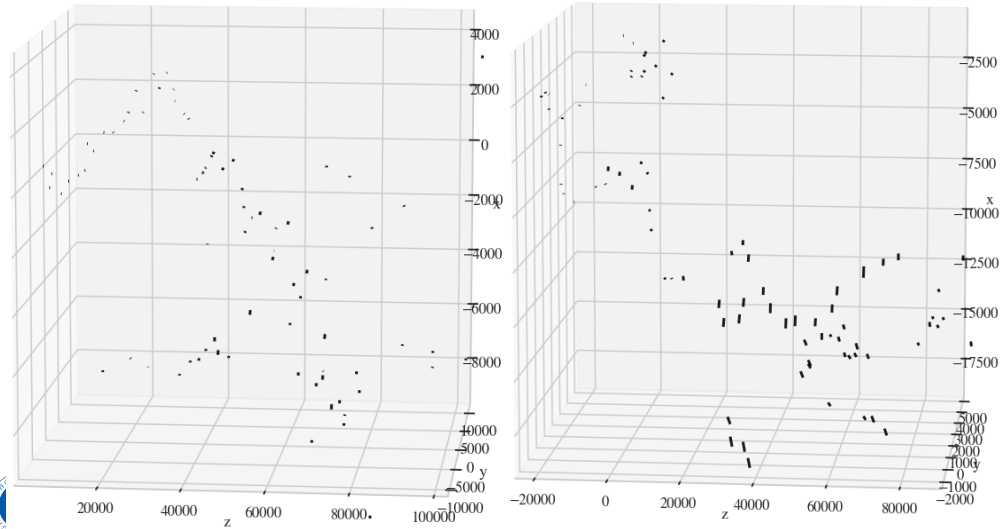
# Examples of generated showers




SCHOOL OF DATA ANALYSIS

# Examples of generated showers

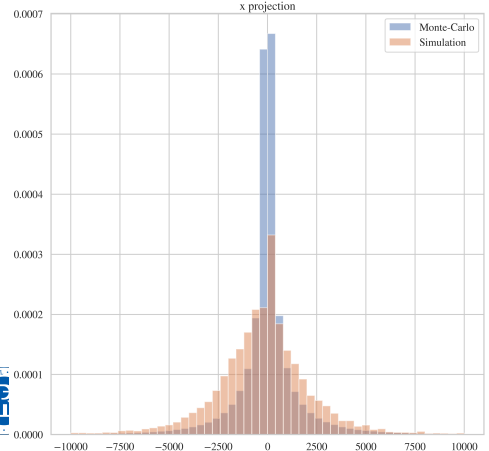



SCHOOL OF DATA ANALYSIS

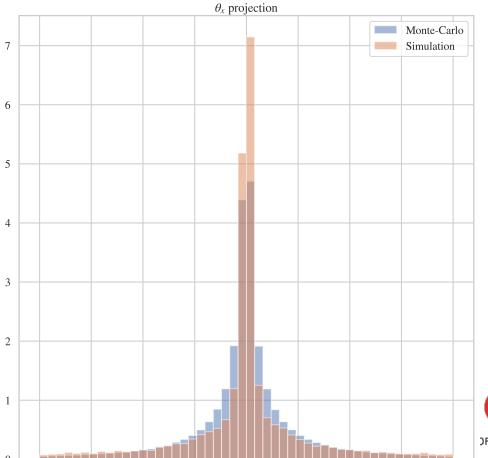
# Distributions comparison

Distributions are compared with Kolmogorov-Smirnov statistic.

$KS = 0.219$



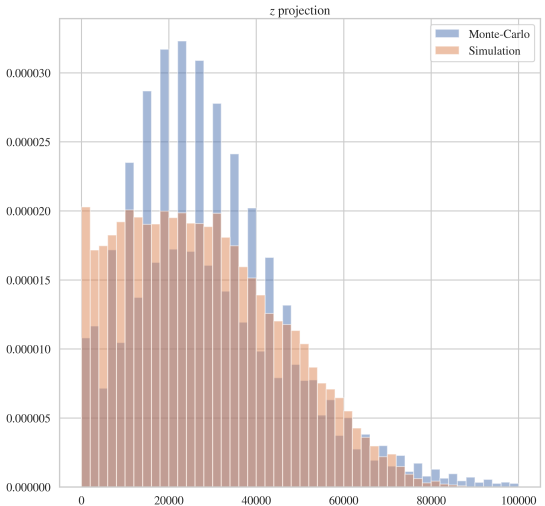

$KS = 0.177$




OF DATA ANALYSIS

# Distributions comparison

$KS = 0.154$



# Conclusion

We propose(wip) a novel approach for shower generation on track level which has following advantages:

- ready proof-of-concept model for further developments
- speed-up x10-100(depends on shower energy) in comparison with Monte-Carlo;
- scales linearly with the number of tracks in the shower.

Future plans:

- predict **conditional** distributions for track differences;
- general GCN layer $\rightarrow$ TreeRNN-layers;
- try GANs, Normalizing Flows(?) approaches.


SCHOOL OF DATA ANALYSIS

# Acknowledgements


- Andrey Ustyuzhanin, my supervisor, for his advises and fruitful discussions about this project;
- Fedor Ratnikov & Denis Derkach for detailed comments on this presentation;
- my girlfriend, Kate, for encouragement.


# Appendix

# Pros of this approach

- speed scales linearly in speed with the size of graph – thanks to GraphRNN;
- stable training – thanks to training in supervised manner;
  - training is also fast;
- itepretability – output distributions could be visualized and analysed to understand what's going on in NNs "brain".

SCHOOL OF DATA ANALYSIS