ICEPP
The University of Tokyo

Event classification using Graph Neural Network

ISGC 2021 @ online

26 / 03 / 2021

International Center of Elementary Particle Physics, University of Tokyo

Masahiko Saito, Y. Iiyama. T. Kishimoto, R. Sawada, J. Tanaka, K. Terashi

Deep learning applications for particle physics

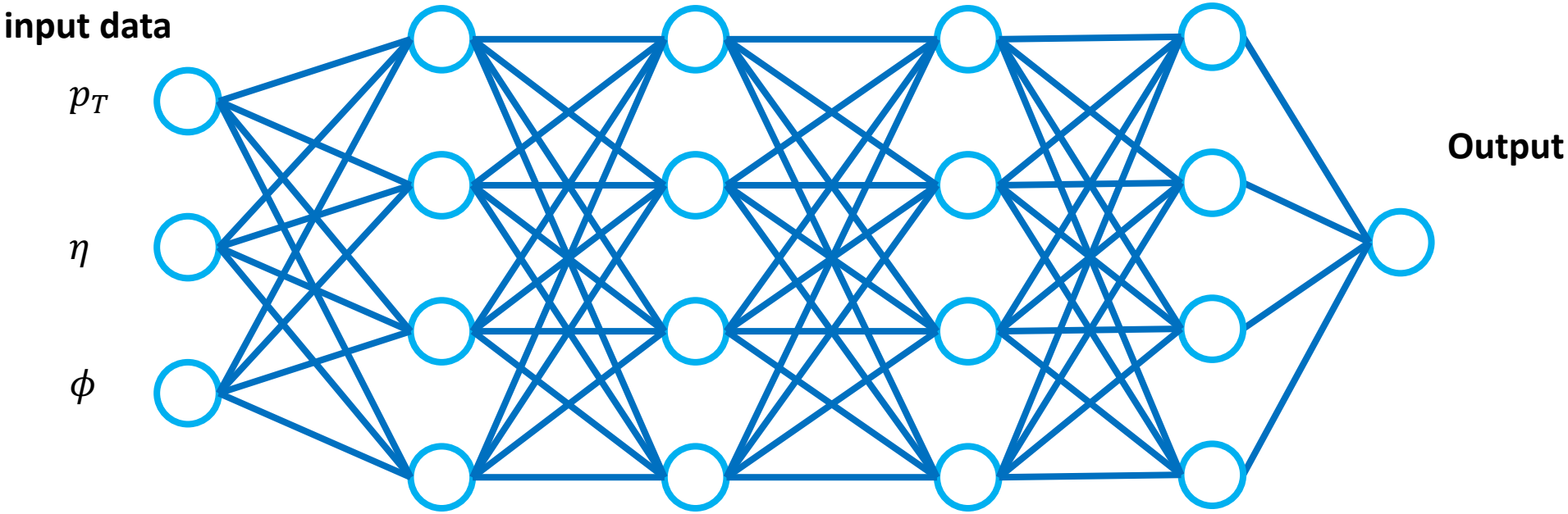
- High-luminosity LHC starts from 2027, collecting 3000 fb^{-1} pp collision events with $\sqrt{s}$ = 14 TeV.
  - Need to leverage the **quite big data** to the fullest extent
- **Machine learning** would become one of the key techniques to utilize such data in the next decade of LHC experiments.

- Machine learning, especially **deep learning**, is already used in many tasks in collider experiments.
  - jet tagging, event classification, calorimeter shower simulation, anomaly detection, …
  - HEPML-LivingReview: ~ 500 papers
- There are many types of deep learning models and techniques.

Typical deep learning model

Multi-layer perceptron (MLP)

- Fully Connected (FC), Deep neural network (DNN), Dense, …

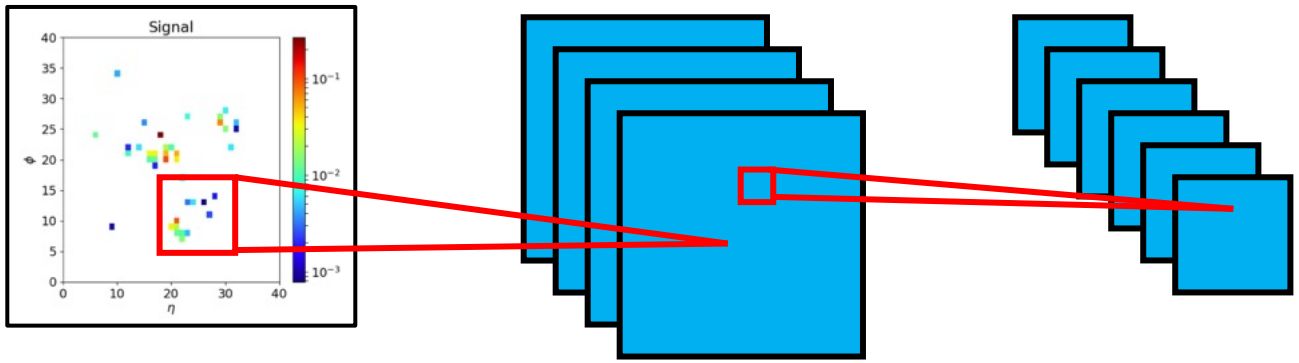


- Most basic and famous deep learning model
- All nodes are connected to all nodes in a next layer
  - Strong approximation capability
  - Many trainable parameters
- Good performance even if using low-level inputs

Typical deep learning model

Convolutional neural network (CNN)

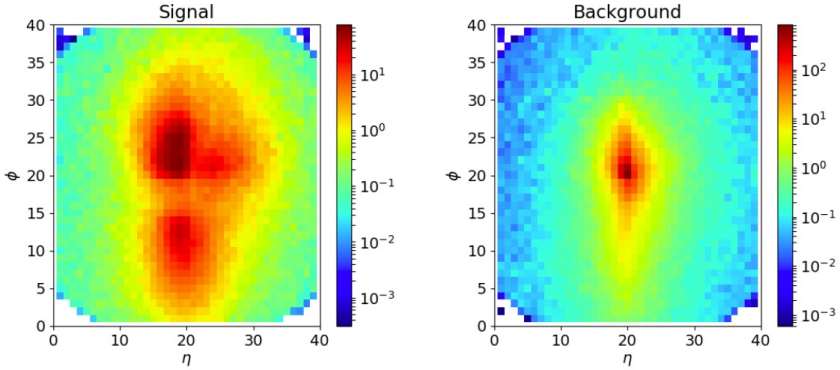
Input data



- Often used in image recognition tasks.
- Convolution with small filters enables to
  - be robust for a global shift
  - effectively learn the relationships between neighbor pixels
- e.g. CNN is used in top-tagging using calorimeter energy distributions as images

Top-tagging

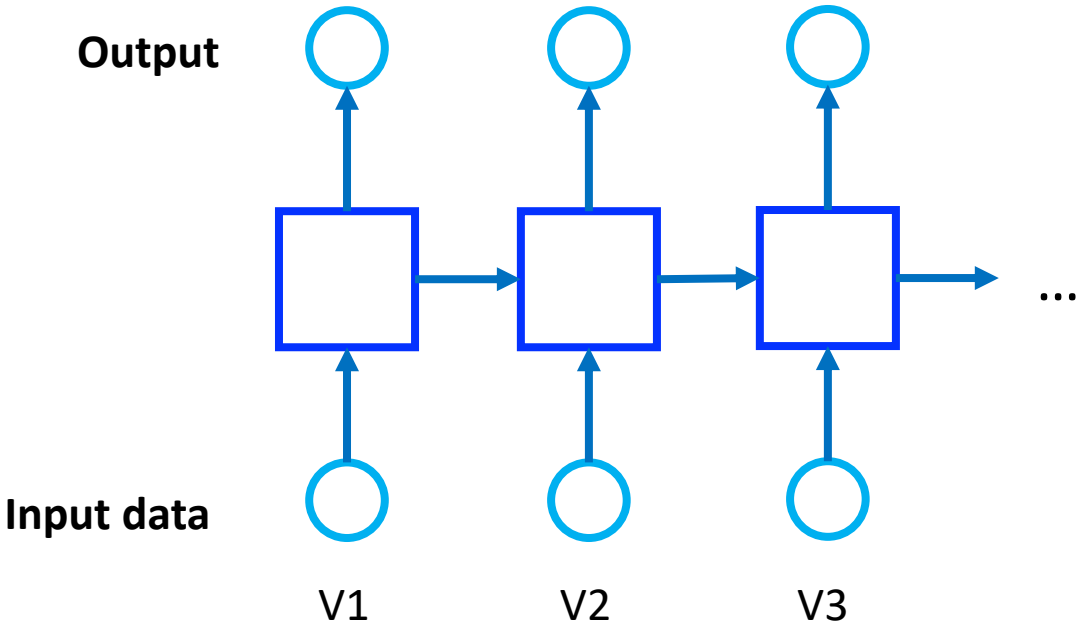
(SciPost Phys, 7, 014 (2019))



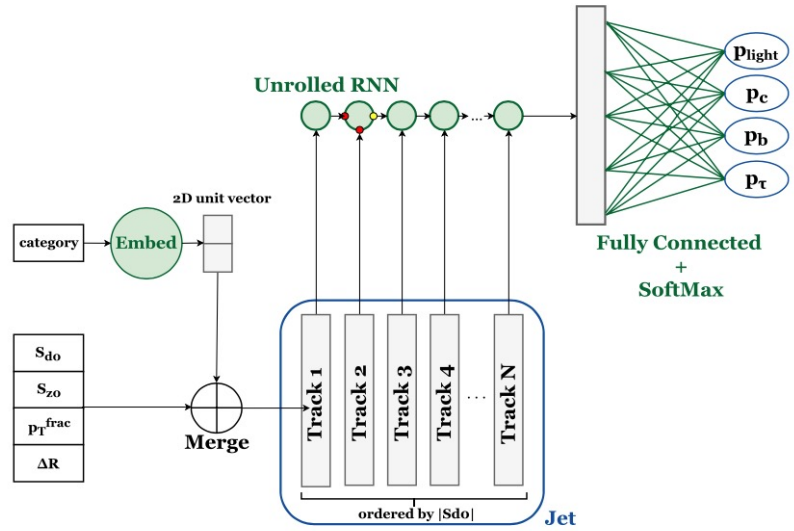
Typical deep learning model

Recurrent neural network (RNN)

- LSTM, GRU



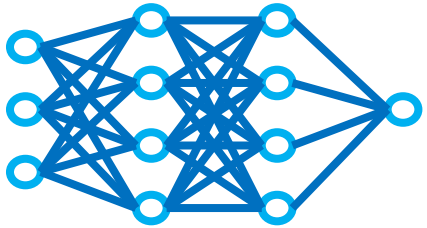
RNN b-tagging
(ATL-PHYS-PUB-2017-003)



- Used in time series analysis and natural language processing
- By using outputs recurrently, RNN can handle sequential data and variable inputs
- e.g. Tracks in jets, where the number of tracks varies from jet to jet, are processed by RNN for b-tagging

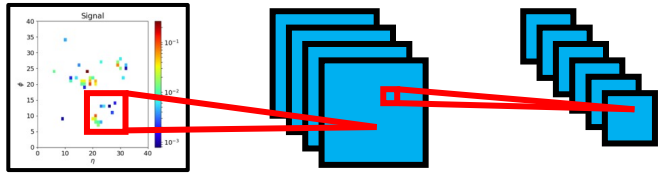
Typical deep learning model

Multilayer Perceptron(MLP)



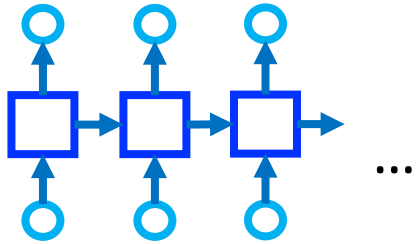
- tabular data
- All correlation between all input variables

Convolutional NN (CNN)



- Grid data
- Local correlation
- Same manipulation for different positioned data

Recurrent NN(RNN)



- Sequential data
- Time correlation
- Same manipulation for different time data

Q: What's types of deep learning model should be used ?

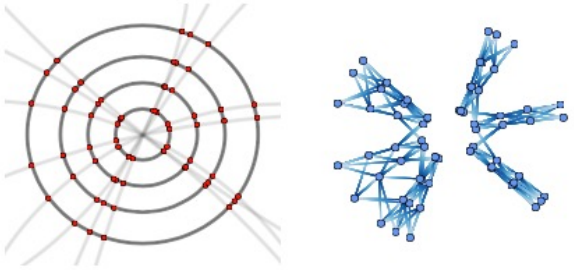
A: A deep learning model should be built considering the structure of the target task (domain knowledge, inductive bias)

- resulting in a reduction of model parameters, mitigation of over-fitting

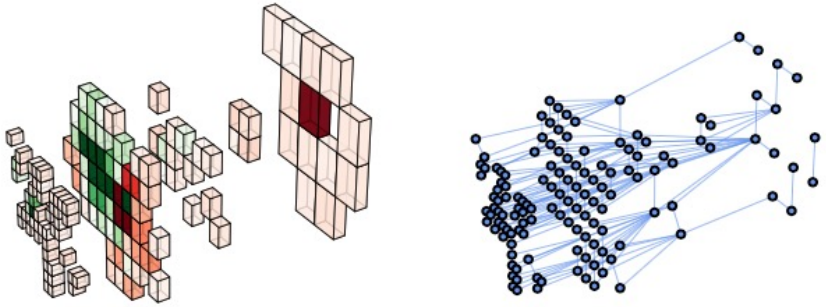
- Data structure used in HEP is often compatible with a **graph**.

Data structure in HEP

[2007.13681]



(a) **Tracking**



(b) **Calorimeter clustering**

(c) **Event classification**

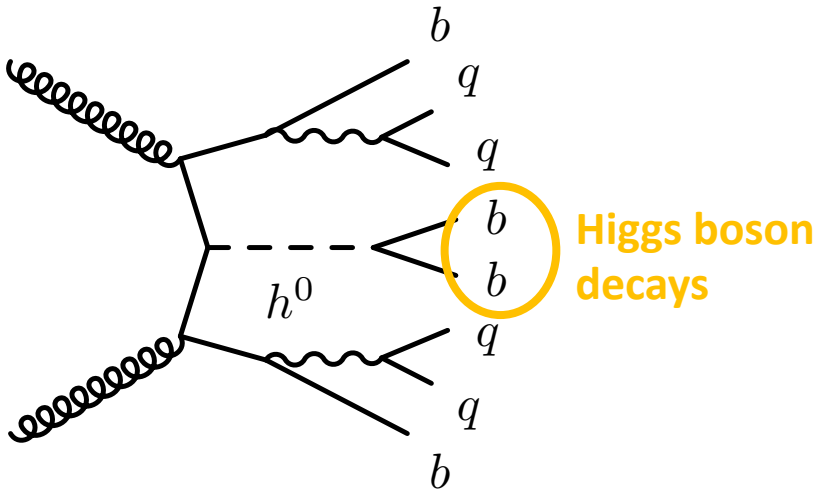
(d) **Jet classification**

- Representation as a graph has advantages for the data satisfied that
  - the number of entities varies
  - indexing (ordering) of entities is not intrinsic
  - relations between entities can be defined explicitly.
- **Graph Neural Network (GNN)** is a deep learning model that handles a graph as input data
- This talk focuses on the event classification using GNN

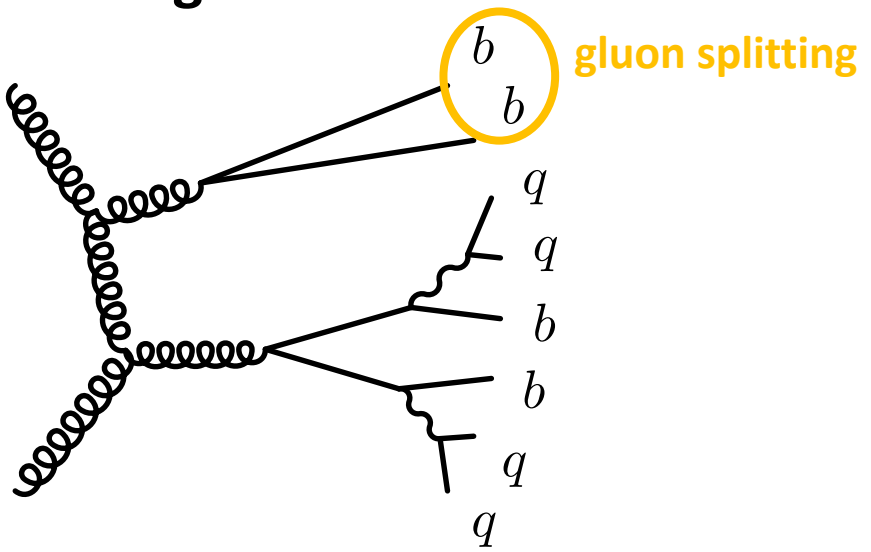
Problem/Experiments

Signal and Background

Signal: ttH(→bb)



Background: ttbb



- Classification problem of signal ($ttH(\rightarrow bb)$) and background ($ttbb$)
- Focus on no lepton channel
  - 4 bottom-quark + 4 light-quark
  - 8 jets are observed
- There are many jets, and it's hard to reconstruct/identify particle origins. This classification is challenging!

Problem/Experiments

Dataset

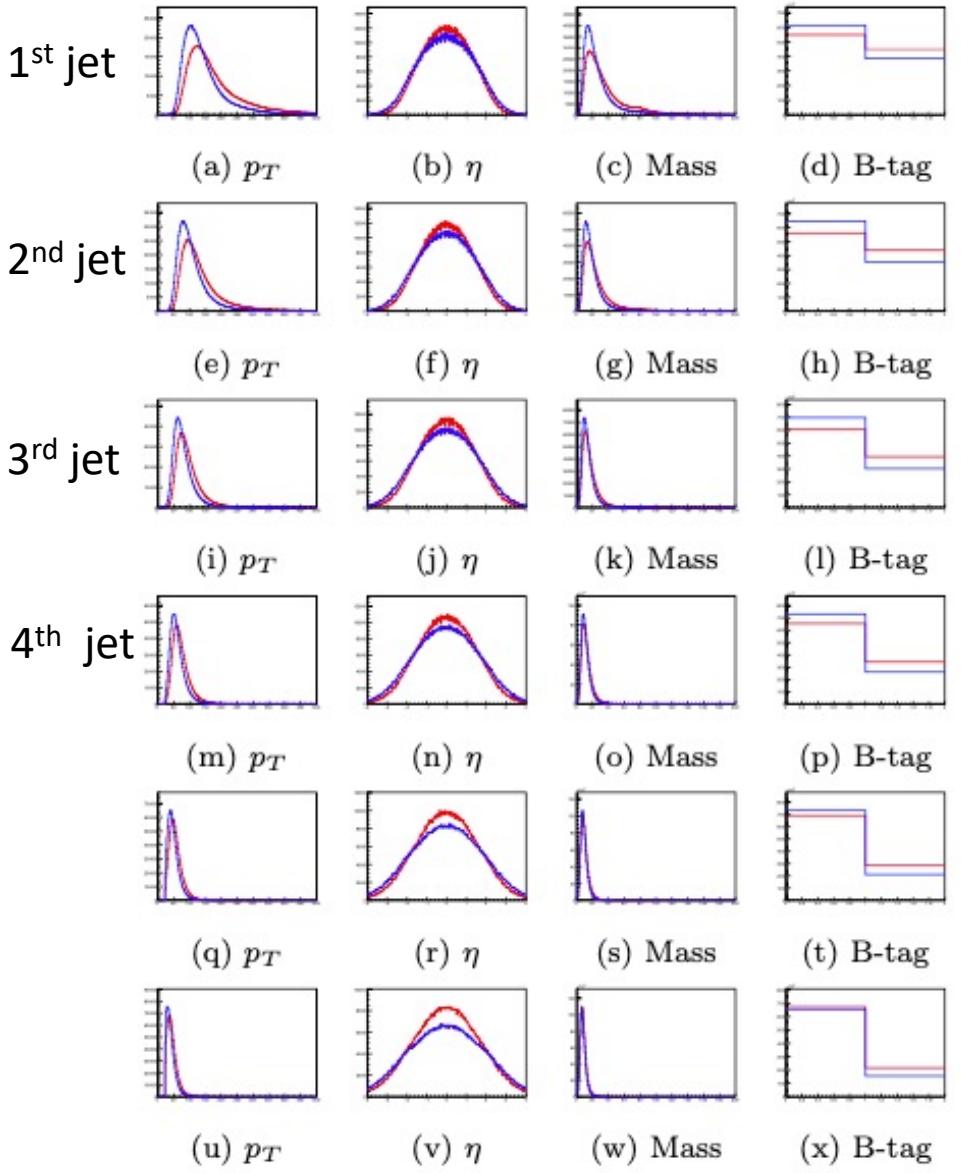
- Use simulation dataset
  - MadGraph5 + Pythia8 + Delphes
  - Emulate a response of ATLAS detector
- Generate 1M events for signal and background
  - Training / Validation / Test = 80 % : 10 % : 10 %

Input variables

- Lorentz vector (p_x, p_y, p_z, E) + b-tag label for each jets
- 8 (jets) x 5 (features/jet) = 40 features

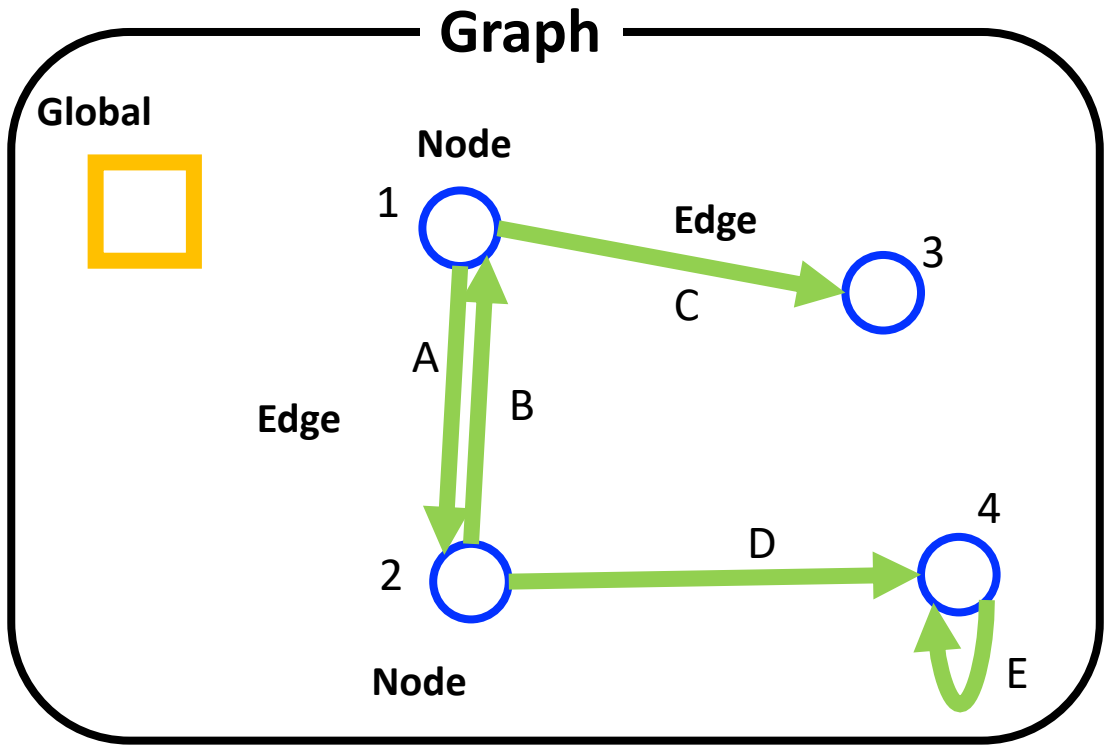
Input variables

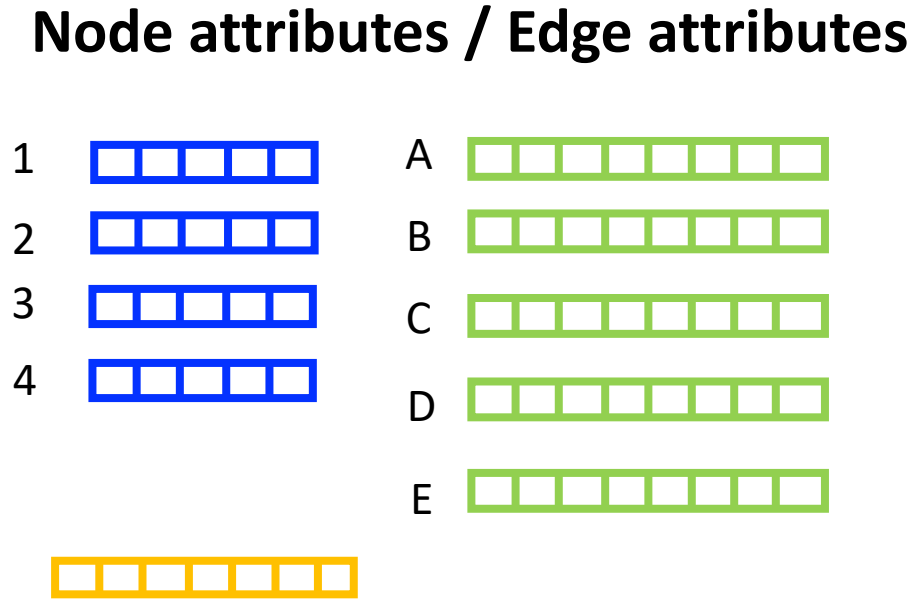
signal
Background



Overview of Graph Network

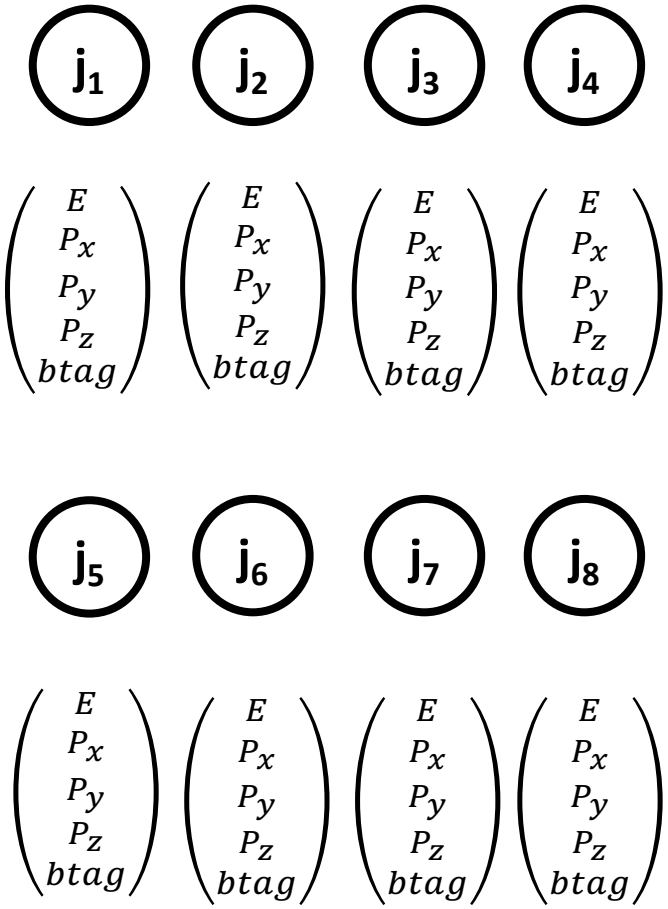
1806.01261



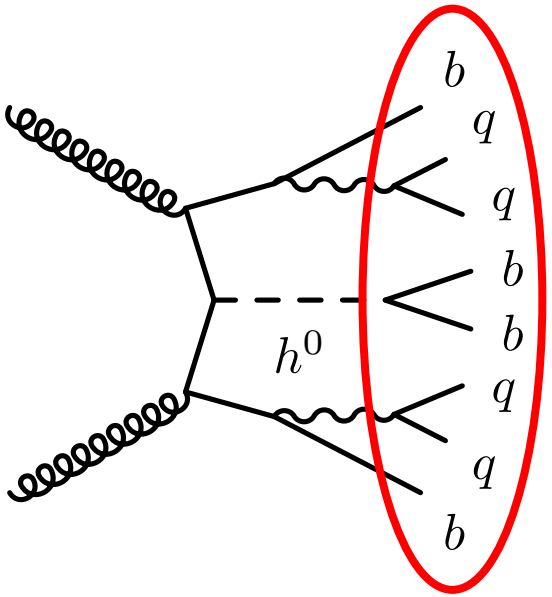


- **Graph** consists of nodes and edges
- Each node and edge has attributes
- Prediction value is calculated by iteratively updating node/edge attributes
  - Update rule is common for all nodes (edges).

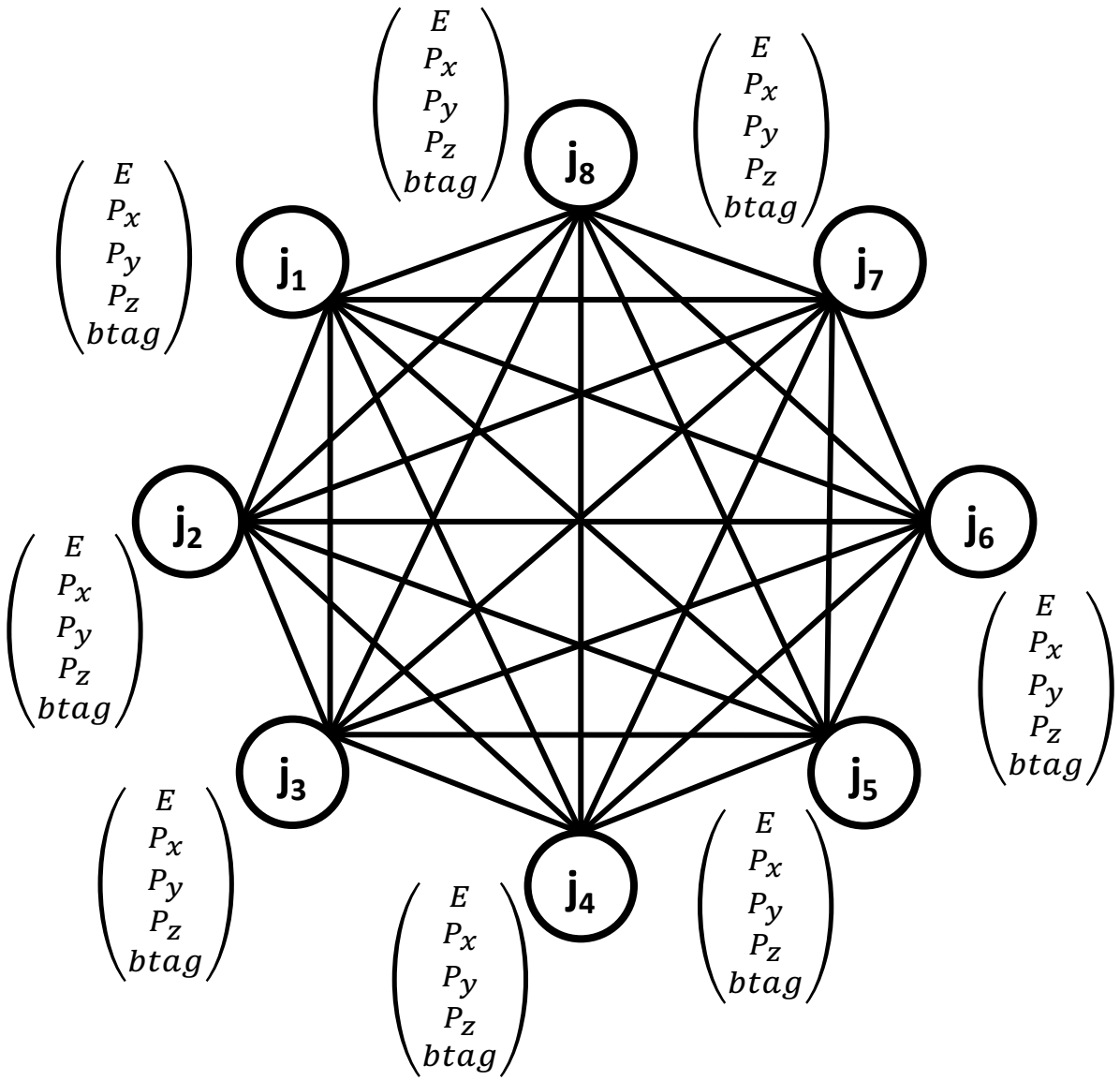
Input graphs



- There are 8 jets and 5 features for each jet, which are used as input variables

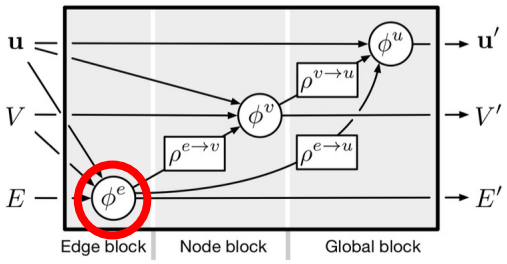


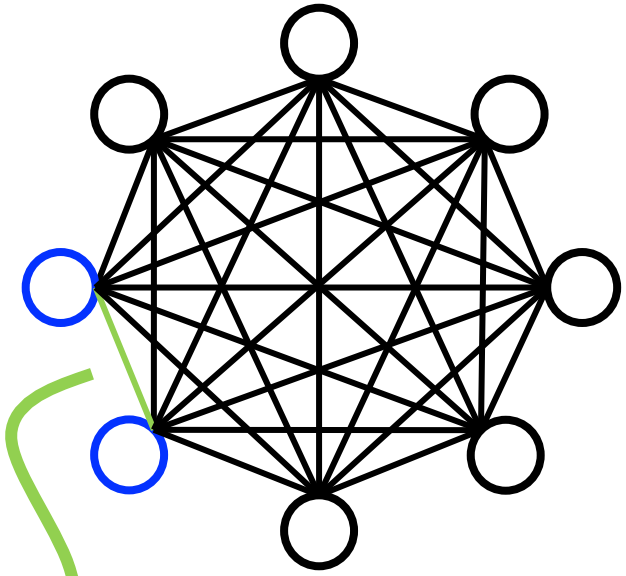
Input graphs



- There are 8 jets and 5 features for each jet, which are used as input variables
- Each jet is assigned as a node.
  - Node attribute: $(E, p_x, p_y, p_z, b\text{-tag})$ $(\mathbb{R}^5)$
- Each node connects to all other nodes (full connection)
  - Edge attribute: $\mathbb{R}^N$
    - represents a relation between two jets

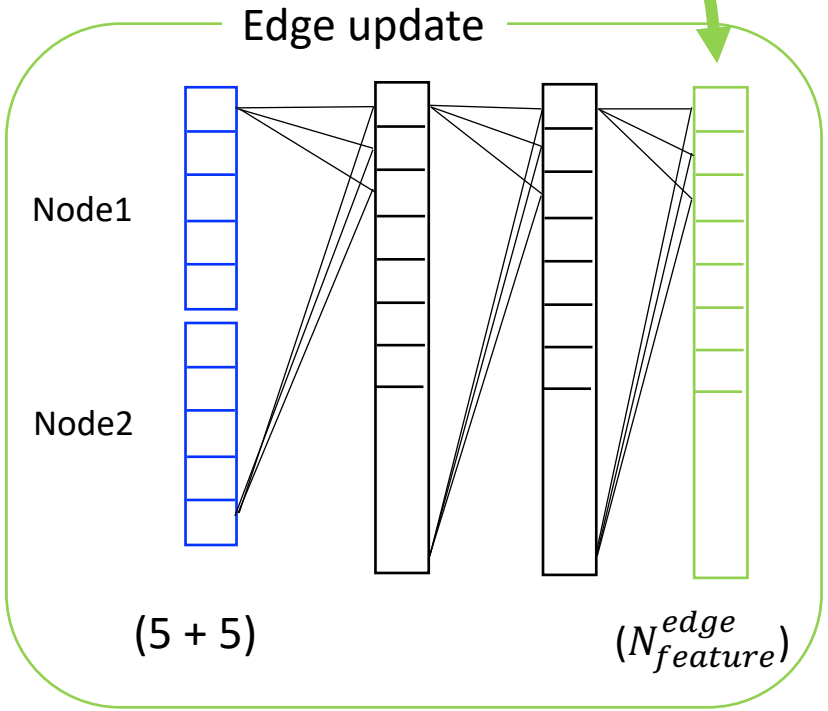
Update of graph





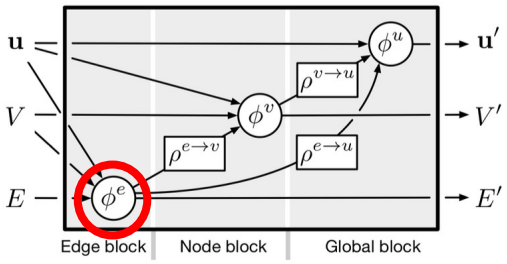
$$\begin{pmatrix} E \\ P_x \\ P_y \\ P_z \\ btag \end{pmatrix}_A$$

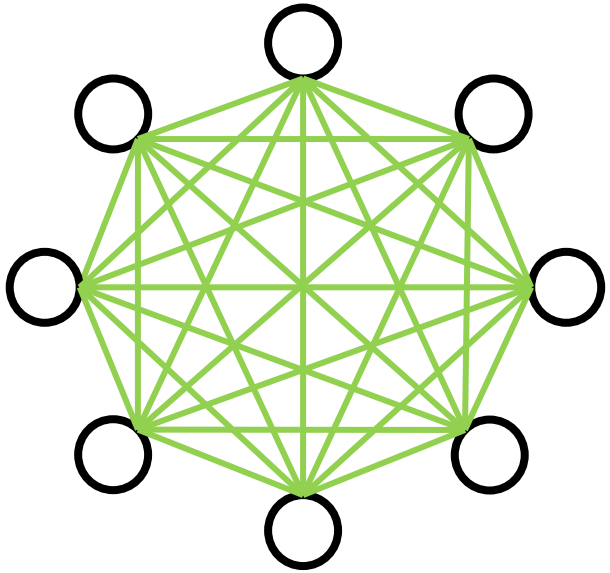
$$\begin{pmatrix} E \\ P_x \\ P_y \\ P_z \\ btag \end{pmatrix}_B$$

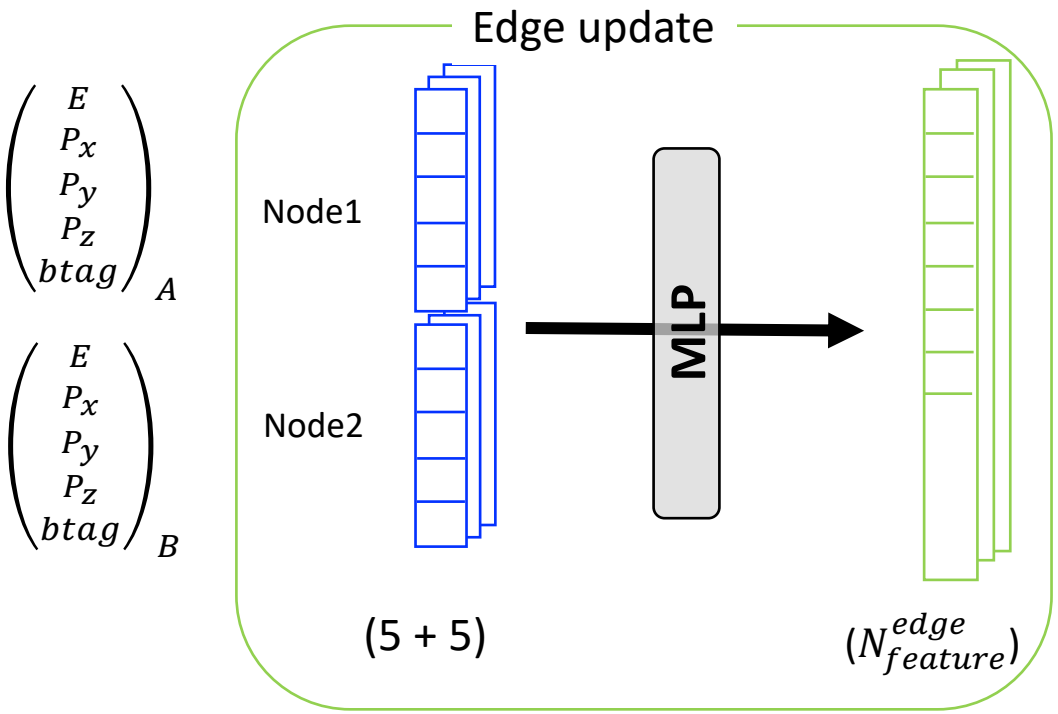


- Update edge attributes using neighbor node attributes
- Updated by multilayer perceptron (MLP)

Update of graph

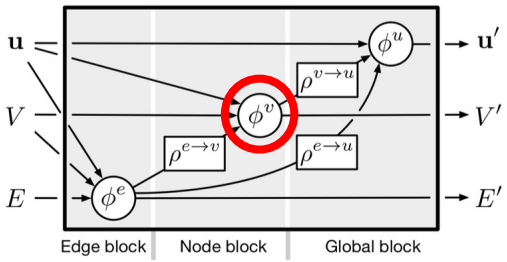


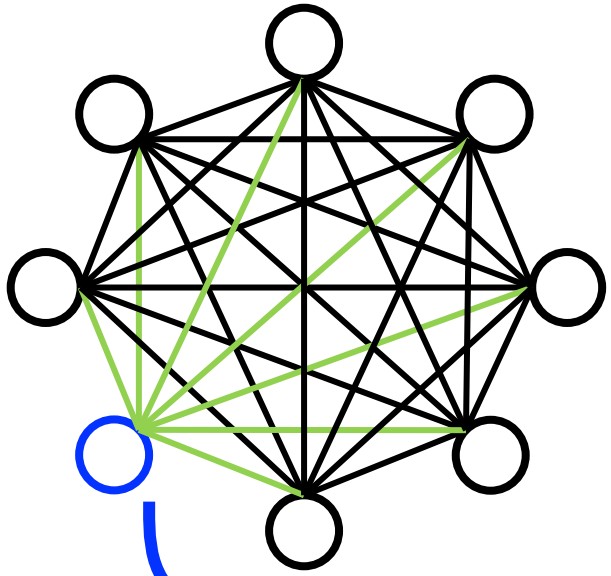




- Update edge attributes using neighbor node attributes
- Updated by multilayer perceptron (MLP)
- Update all edge attributes using the same MLP

Update of graph





- Update node attributes using neighbor edge attributes

$$\begin{pmatrix} E \\ P_x \\ P_y \\ P_z \\ btag \end{pmatrix}_A \qquad \begin{pmatrix} E \\ P_x \\ P_y \\ P_z \\ btag \end{pmatrix}_B$$

Edge update

Node1

Node2

MLP

(5 + 5)

$(N_{feature}^{edge})$

Summation of all neighbor edge attributes

Node update

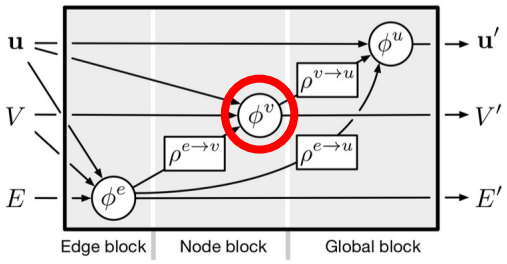
Summation of neighbor edge attributes

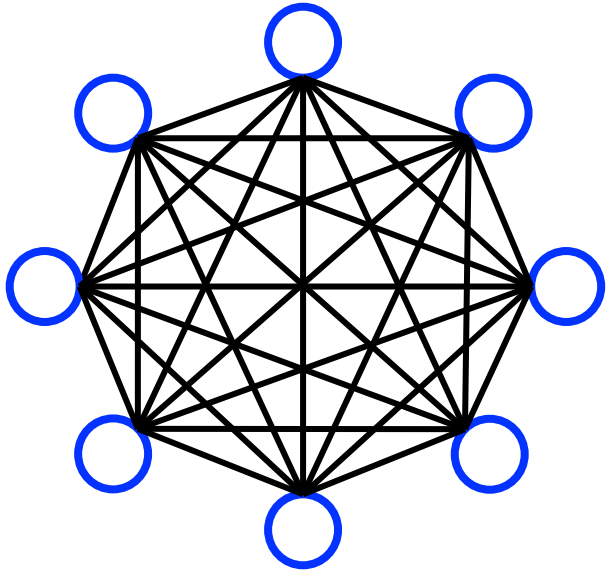
MLP

$(5+ N_{feature}^{edge})$

$(N_{feature}^{node})$

Update of graph





- Update node attributes using neighbor edge attributes
- Update all node attributes using the same MLP

$$\begin{pmatrix} E \\ P_x \\ P_y \\ P_z \\ btag \end{pmatrix}_A$$

$$\begin{pmatrix} E \\ P_x \\ P_y \\ P_z \\ btag \end{pmatrix}_B$$

Edge update

Node1

Node2

MLP

(5 + 5)

$(N_{feature}^{edge})$

Summation of all neighbor edge attributes

Node update

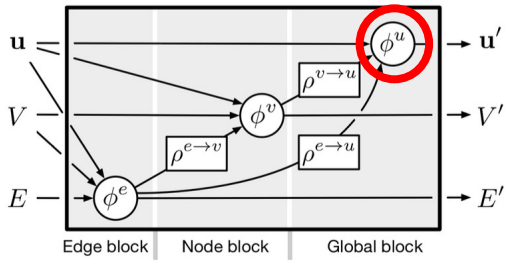
Summation of neighbor edge attributes

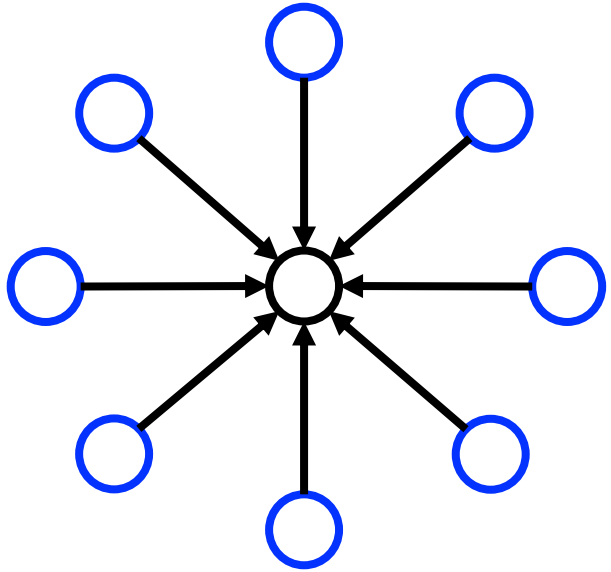
MLP

(5+ $N_{feature}^{edge}$)

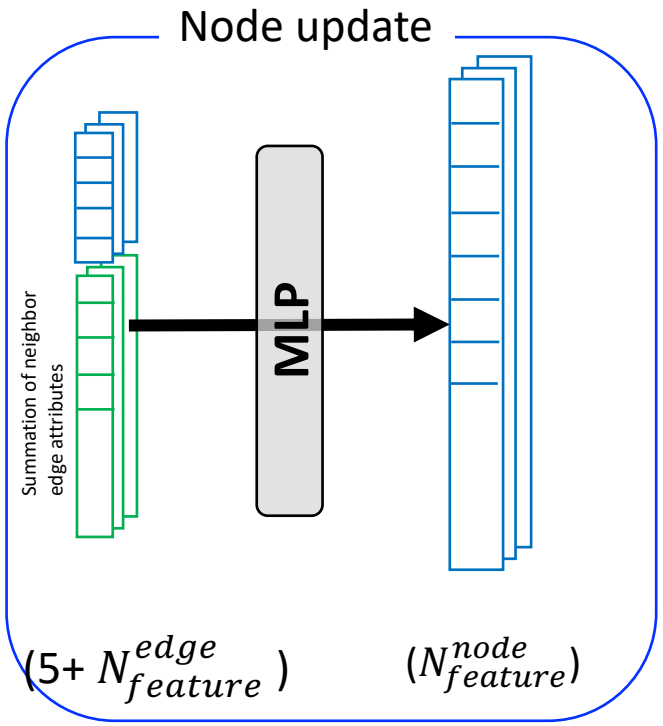
$(N_{feature}^{node})$

Update of graph

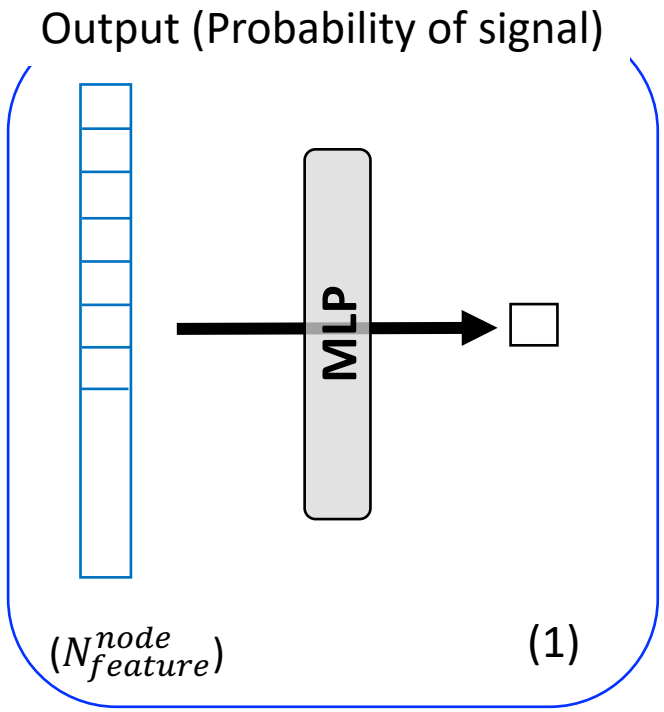




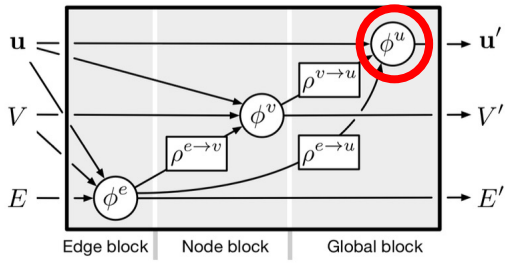
- Calculate output value using all node attributes and MLP

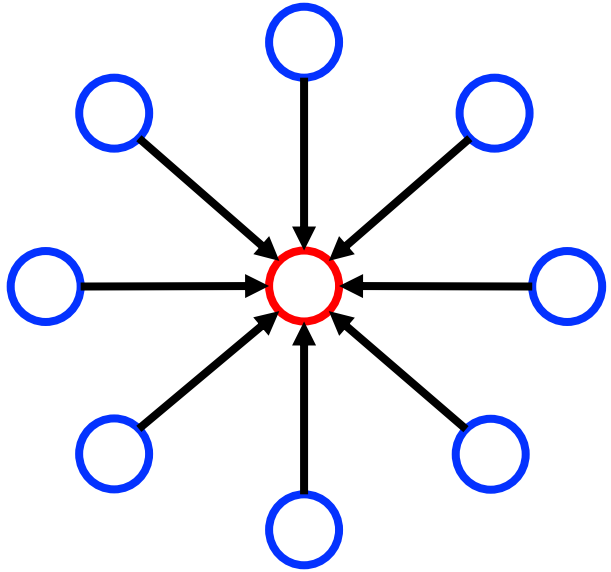


Summation of all node attributes

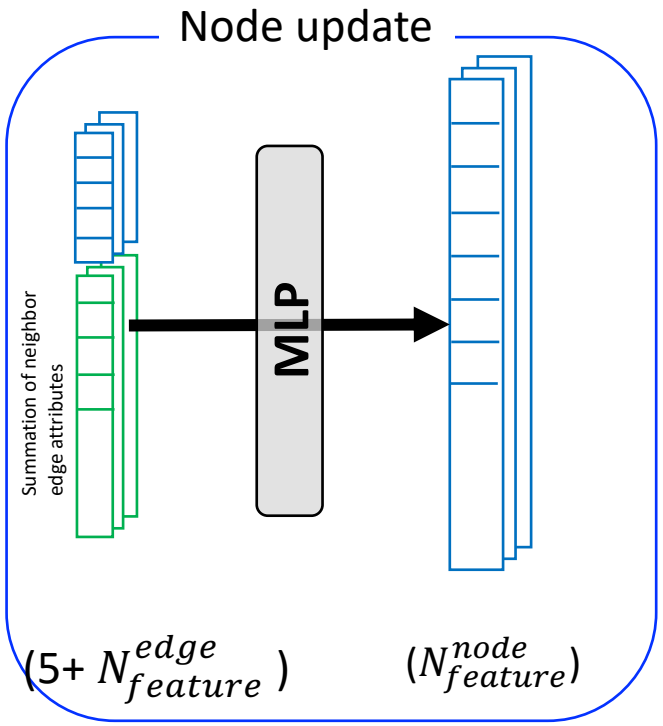


Update of graph

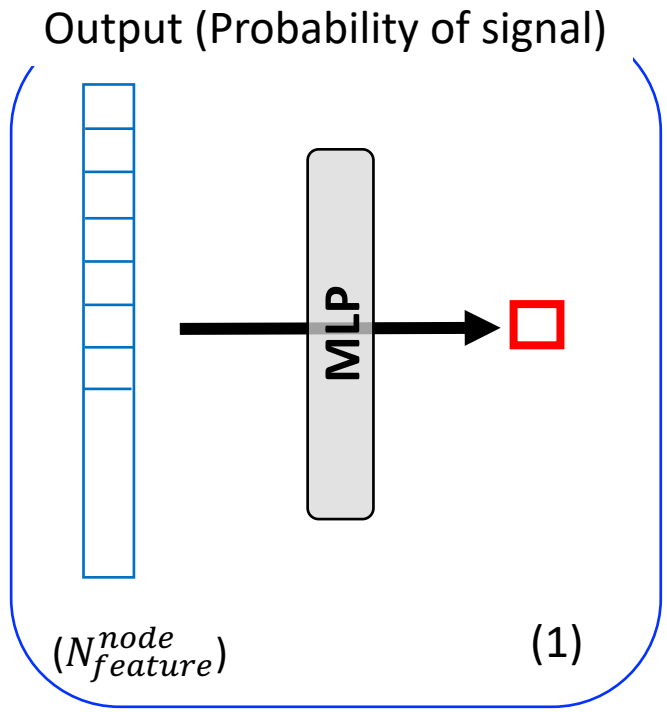




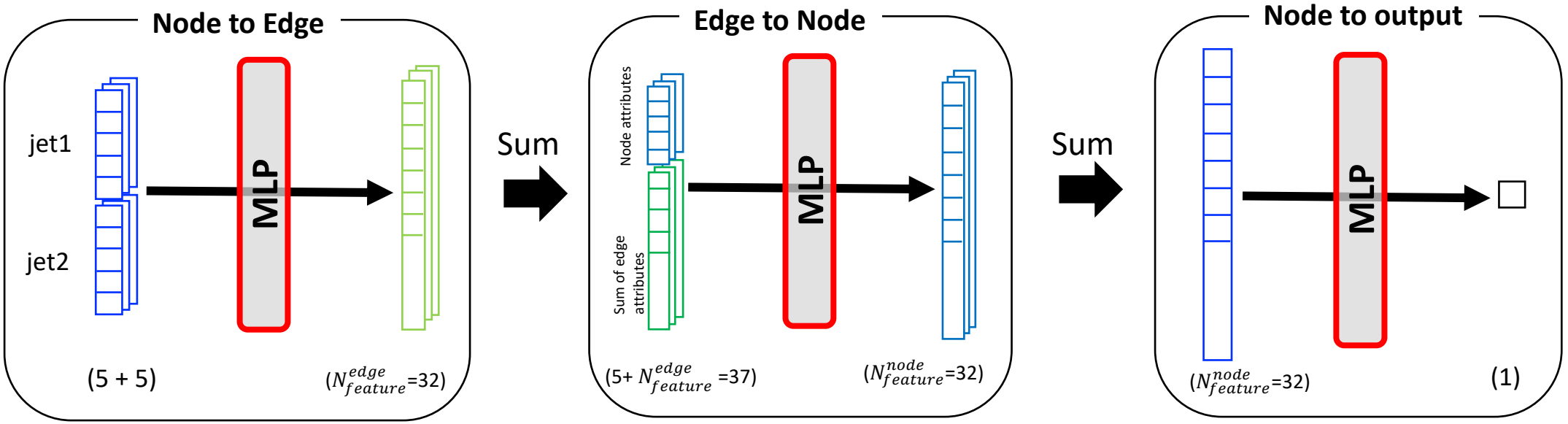
- Calculate output value using all node attributes and MLP
- The last node is used as output which is used for the calculation of loss function



Summation of all node attributes



Summary of Graph network



Three trainable MLP modules

Each MLP structure

- two hidden layer with 256 nodes
- Activation function: ReLU (sigmoid for the last layer)
- Applied Batch normalization for all layers

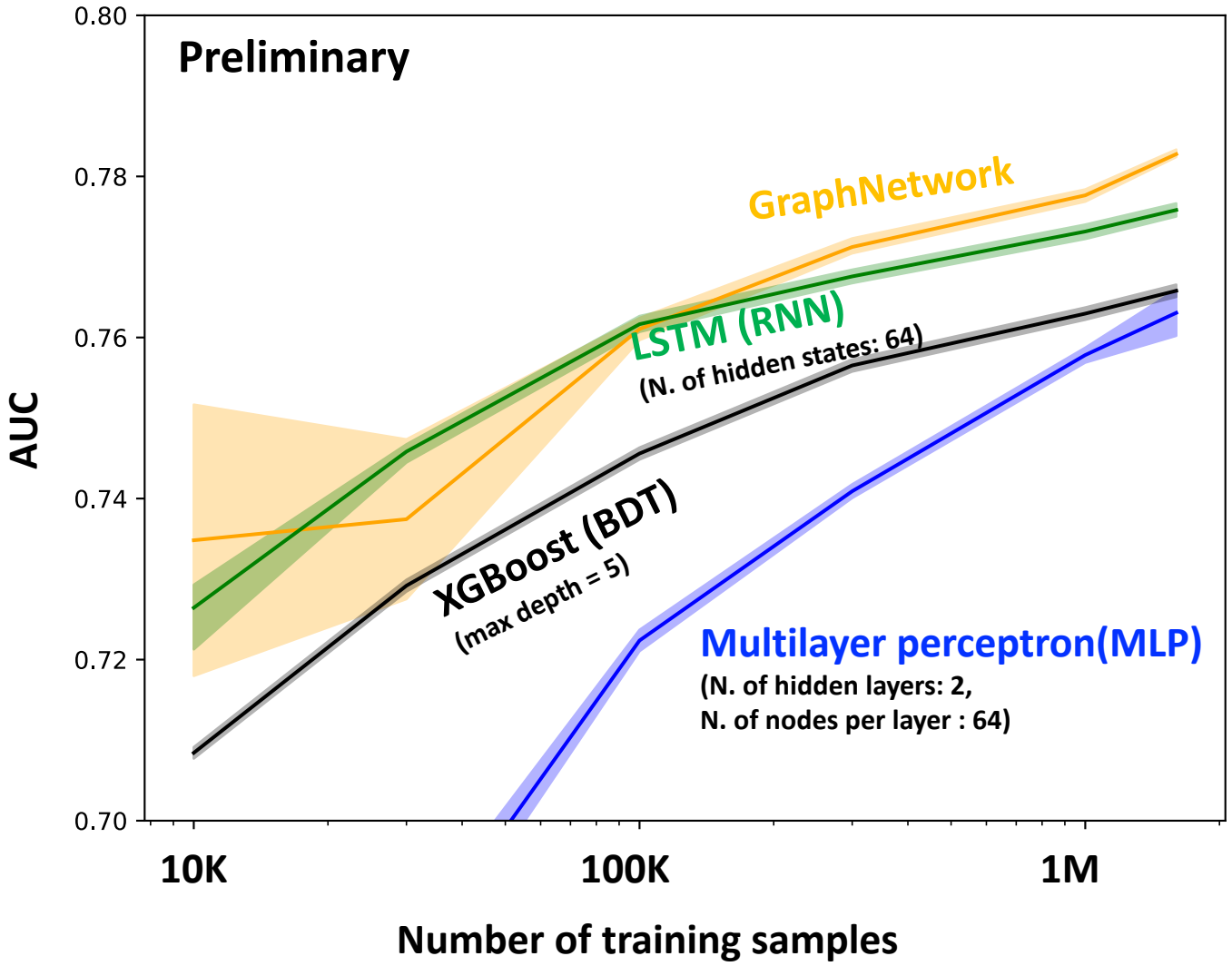
- Optimized the size of node/edge attributes and MLP layer structure to maximize AUC
- Implemented using graph_nets library that is developed by DeepMind

Reference models

- Compare with three machine learning models
  1. Multilayer perceptron (MLP)
     - Input: 40 variables in a row
     - Optimized the number of hidden layers and the number of nodes.
  2. Long short-term memory (LSTM)
     - Input: particles (5 variables, maximum 8 particles) ordering by transverse momentum (p_T)
     - Optimized the number of hidden states
  3. Boosted decision tree (BDT)
     - Input: 40 variables in a row
     - Optimized the depth of tree
     - Used XGBoost

- Hyperparameters of each model is optimized to maximize AUC, then compare the performance of the models with the best hyperparameters

Result

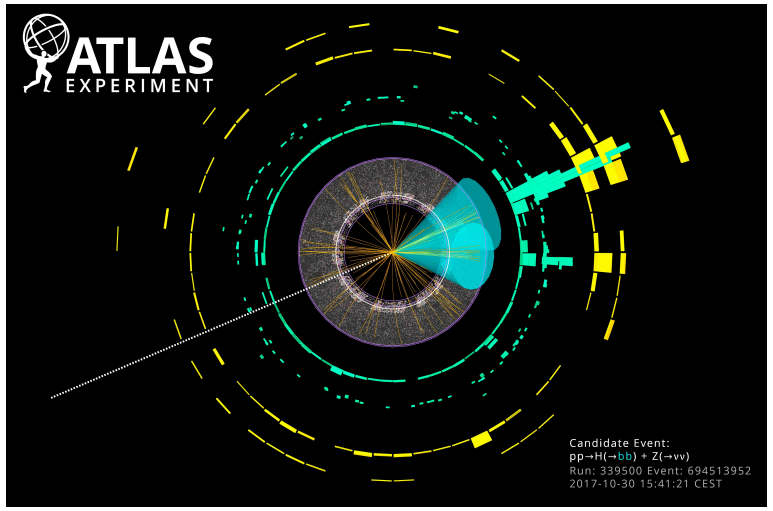


- Graph Network outperforms MLP, LSTM and XGBoost

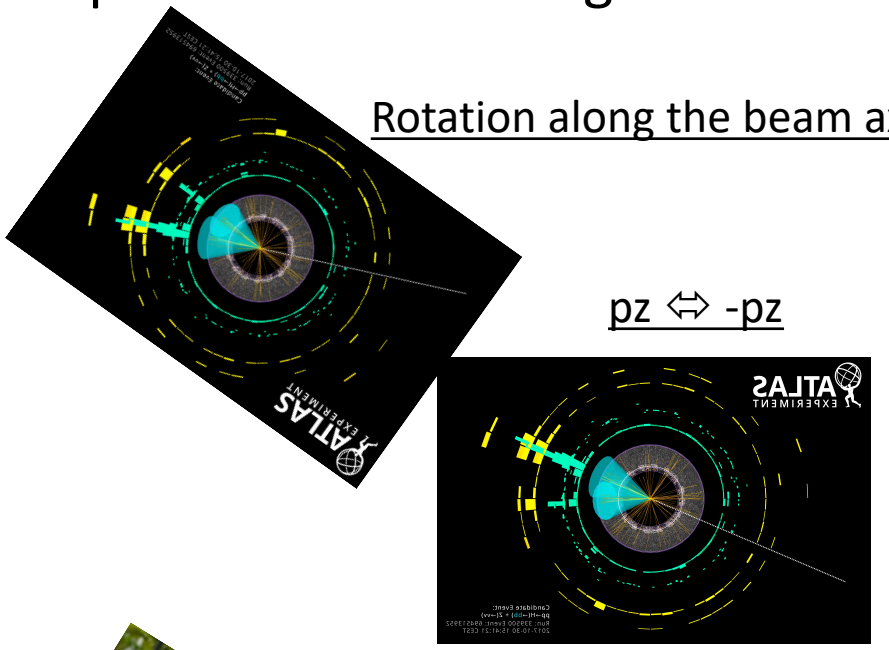
- Great performance when using the large amount of training data
- For the case of using small amount of training data, GNN has good performance compared to MLP
  - The implementation of inductive bias into deep learning models reduces redundant model parameters, resulting in improved performance.

Data Augmentation

- Domain knowledge:
  - Physics law and our detector are symmetric for spatial rotation along the beam axis.

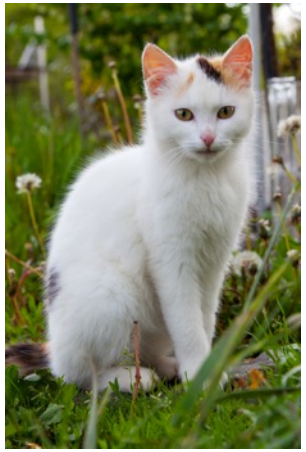


Same probability



Rotation along the beam axis

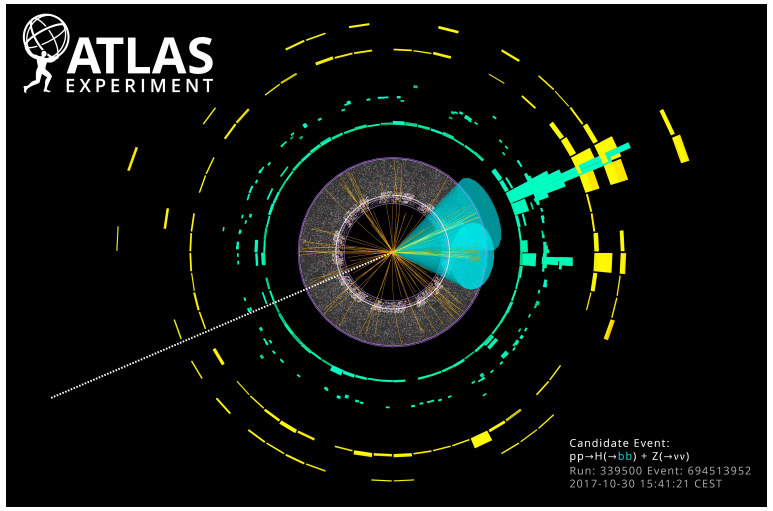
pz ⇔ -pz



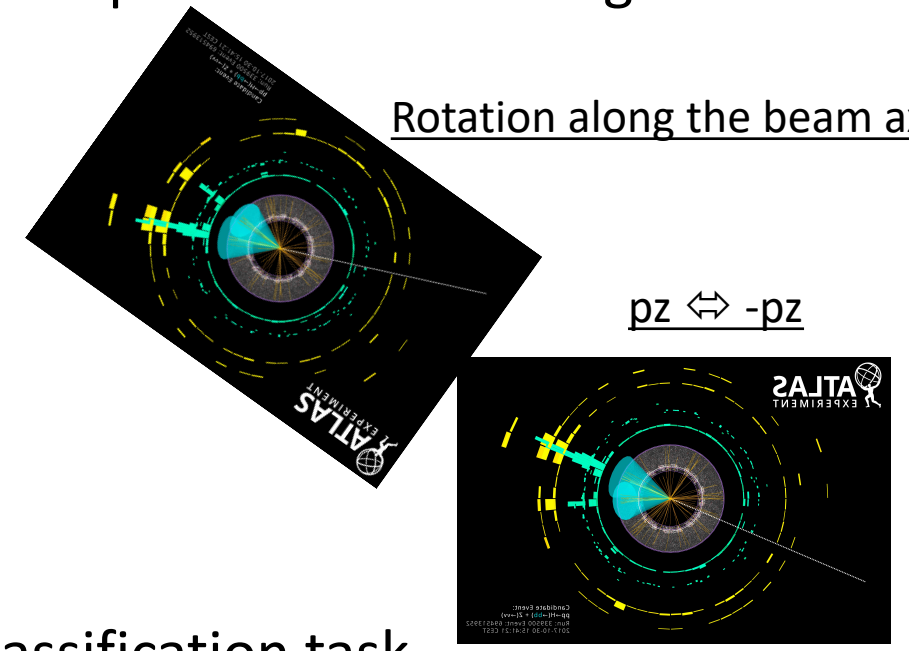
A human can recognize a target even if rotating the images

Data Augmentation

- Domain knowledge:
  - Physics law and our detector are symmetric for spatial rotation along the beam axis.



Same probability



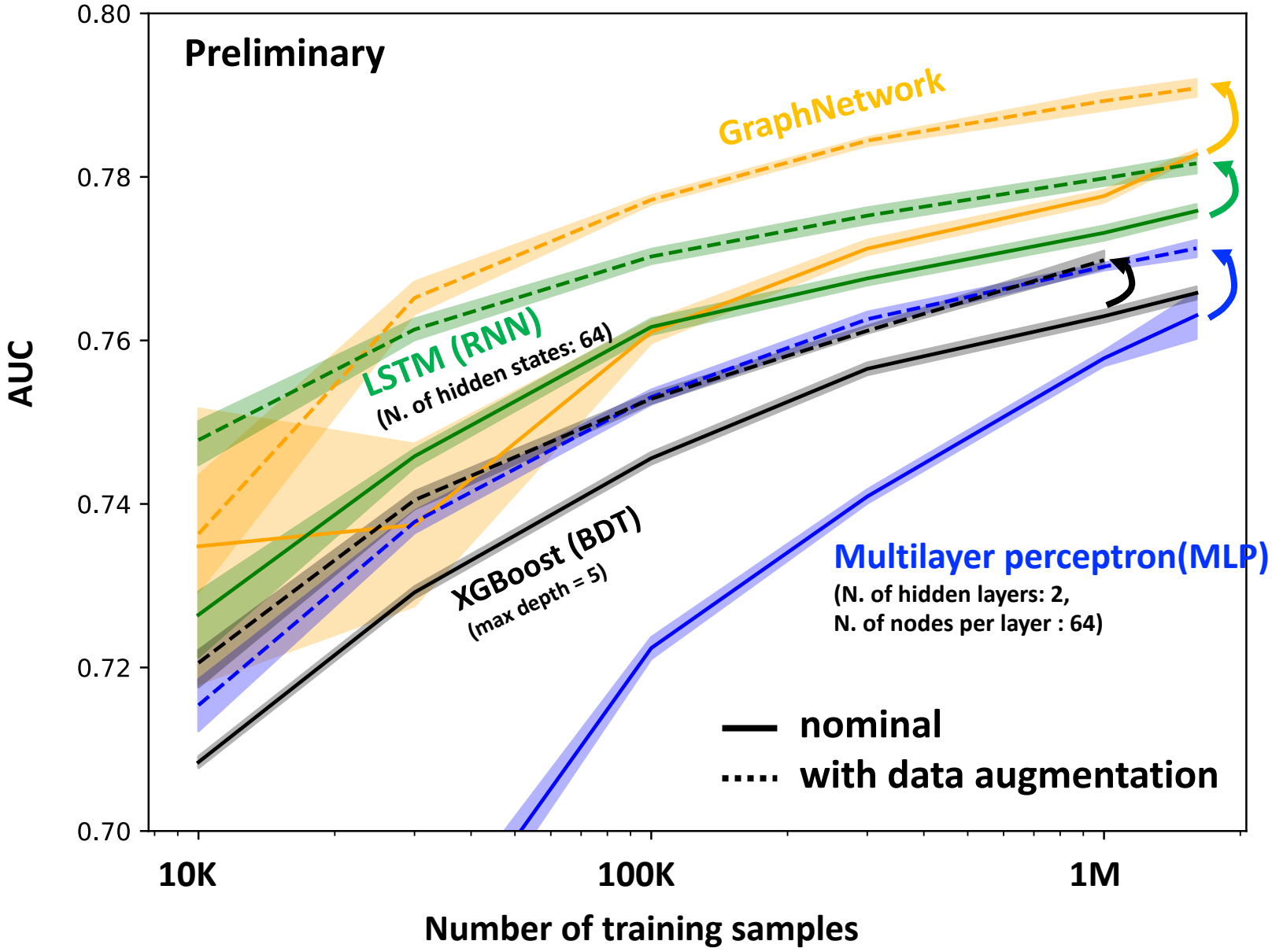
Rotation along the beam axis

pz ⇔ -pz

- As well as the data augmentation in an image classification task, we train the model by randomly rotating the events (4-vector) following the known symmetry on each epoch

$$\begin{pmatrix} E \\ P_x \\ P_y \\ P_z \end{pmatrix} \rightarrow \begin{pmatrix} E \\ P_x\cos\theta - P_y\sin\theta \\ P_x\sin\theta + P_y\cos\theta \\ P_z \end{pmatrix} \qquad \begin{pmatrix} E \\ P_x \\ P_y \\ P_z \end{pmatrix} \rightarrow \begin{pmatrix} E \\ \pm P_x \\ \pm P_y \\ \pm P_z \end{pmatrix}$$

Result with data augmentation



- **Data augmentation is effective**

  even for physical analysis data
  - Increasing effective data size
  - reduce over-fitting
  - Especially effective for MLP with the small training data case

Summary

- Proper implementation of task structure and **domain knowledge** as inductive bias is important for increasing the deep learning model's performance
- **Graph** structure often appears in HEP
  - Graph network is a deep learning model that handles graphs as input

- **Graph network** is applied for physics analysis task
  - Advantage of Graph network: input variables can be structured, it can handle variable length of particles, particle ordering is not required
  - Graph network outperforms MLP, LSTM, and BDT in our case.

- **Data augmentation** (e.g. increasing data size using spatial invariance of momentum) is also effective for physics analysis tasks