The Smart Background Project

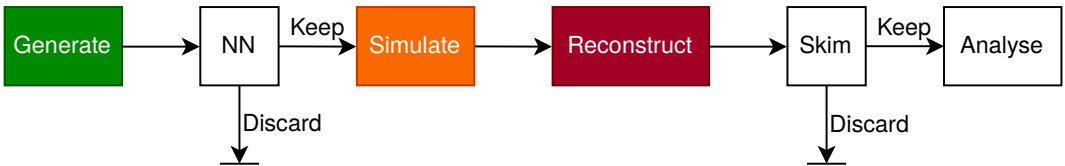
Selective Background Monte Carlo Simulation at Belle II

Boyang Yu, Nikolai Hartmann, Thomas Kuhr

KISS Annual Meeting Hamburg, February 13, 2024

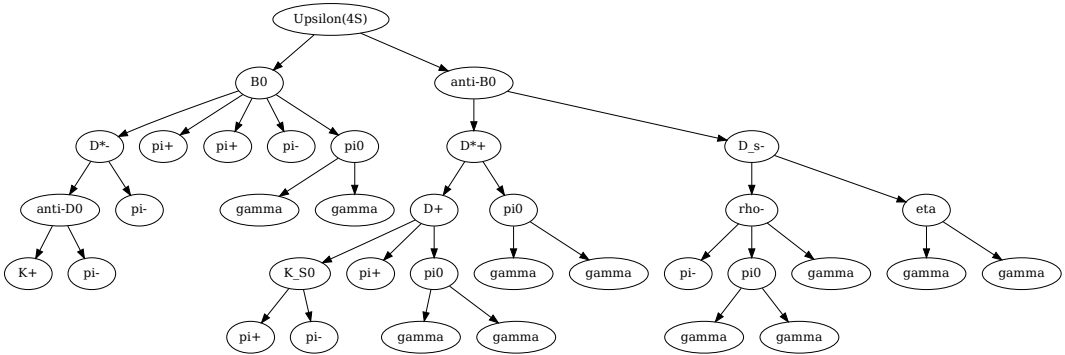
Selective/Smart background MC simulation

Introduced by James Kahn in his PhD thesis:



- Event generation takes much less computing time than detector simulation (at Belle II)
- Many events discarded (e.g. by skim)
  → try to predict which events will be discarded, already after event generation
- Not always a clearly correlated variable on generator level
  → example: skim may use involved algorithms like FEI (Full Event Interpretation)
  → train an NN to be a good filter

Graph structured data $\leftrightarrow$ Graph neural networks



Node attributes: PDG ID, 4-vector components, Vertex positions, Decay times
$\rightarrow$ usage of graph neural networks proved very useful

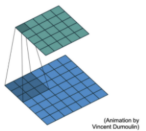
The dataset

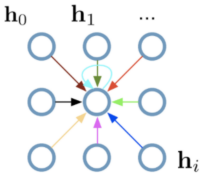
- $Y(4S) \rightarrow B^0 \bar{B}^0$ samples
- label: pass/fail FEI Hadronic B^0 skim
  $\rightarrow$ select events where at least one Hadronic B^0 reconstructed with FEI
- 900k training, 100k validation, 500k testing
- **Note:** Part of this dataset publicly available and featured in common paper "Shared Data and Algorithms for Deep Learning in Fundamental Physics"
  $\rightarrow$ arXiv:2107.00656
  $\rightarrow$ graph network model for this dataset also works well on other datasets!
  $\rightarrow$ `https://github.com/erum-data-idt/pd4ml`
  $\rightarrow$ also using this dataset for our AI Lab course at LMU

CNN vs GCN

CNN

Single CNN layer with 3x3 filter:





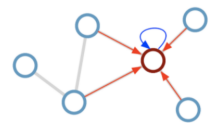
Full update:

$$\mathbf{h}_4^{(l+1)} = \sigma\left(\mathbf{W}_0^{(l)}\mathbf{h}_0^{(l)} + \mathbf{W}_1^{(l)}\mathbf{h}_1^{(l)} + \cdots + \mathbf{W}_8^{(l)}\mathbf{h}_8^{(l)}\right)$$

GCN

Consider this undirected graph:

Calculate update for node in red:

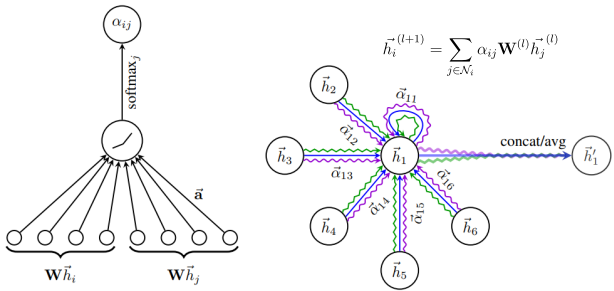


Update rule:

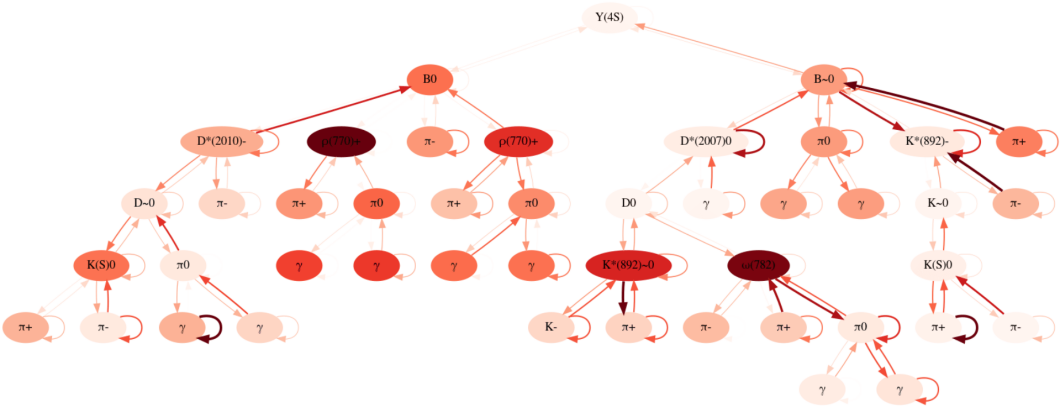
$$\mathbf{h}_i^{(l+1)} = \sigma\left(\mathbf{h}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j \in \mathcal{N}_i} \frac{1}{c_{ij}} \mathbf{h}_j^{(l)}\mathbf{W}^{(l)}\right)$$

Main limitation of plain GCN: equal contribution from each neighbor in sum

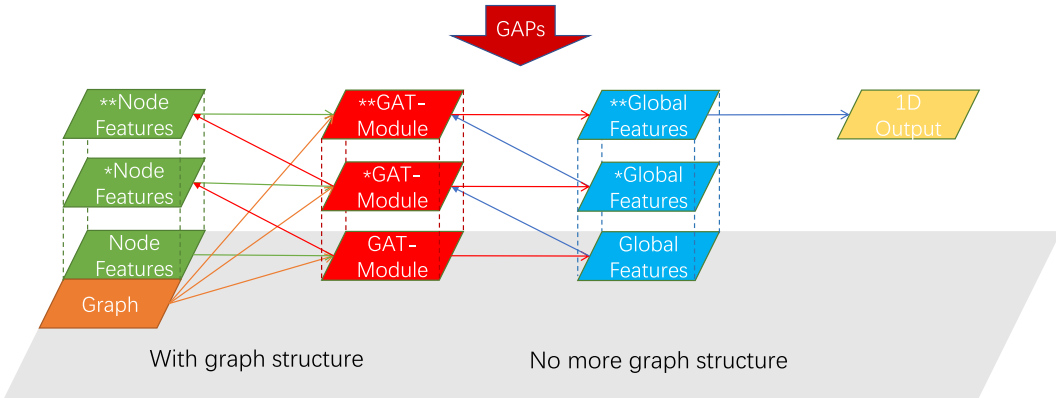
Attention mechanisms



- Graph attention networks (GAT): infer weights for neighbor aggregation from features of adjacent nodes
- Global attention pooling: infer weights for aggregation into global features from node features

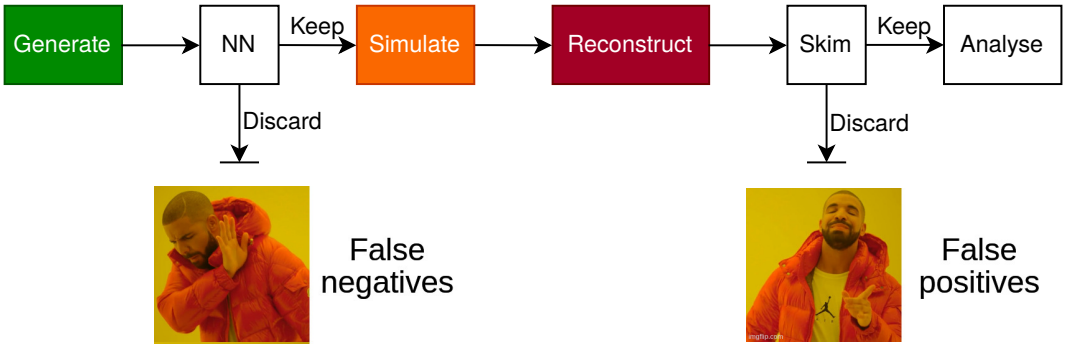

Y(4S)
B0
B~0
D*(2010)-
ρ(770)+
π-
ρ(770)+
D*(2007)0
π0
K*(892)-
π+
D~0
π-
π+
π0
π+
π0
D0
γ
γ
γ
K~0
π-
K(S)0
π0
γ
γ
γ
γ
K*(892)~0
ω(782)
K(S)0
π+
π-
γ
γ
K-
π+
π-
π+
π0
π+
π-
γ
γ

Complete architecture



- after each step update node + global features
- final linear transformation of global features into 1D output
- implemented using `pytorch` + `dgl`

The problem with naive filtering



- **false positives** are not too problematic
  (we throw them away later by running the "true" skim)
- **false negatives** may produce bias (we can't get them back)

The solution: Importance sampling

- Use NN output as **probability to keep event**
- Weight events by inverse probability
- No bias by construction
- Train NN to provide **highest speedup** $\frac{t_{\mathrm{noNN}}}{t_{\mathrm{NN}}}$
  to produce same **effective sample size** $\frac{(\sum_i w_i)^2}{\sum_i w_i^2}$ after skimming
- Very similar to slicing strategy for MC filters at LHC (ATLAS, CMS(?))
  $\rightarrow$ slicing is essentially importance sampling with discrete probabilities
  $\rightarrow$ could our method be applied there, too?
- Speedup of ≈ 2 achievable with benchmark dataset

Summary and Outlook

- Graph NN with attention works well to filter events early in simulation chain
- Importance sampling to avoid bias from false negatives
  $\rightarrow$ train to maximize speedup, considering effective sample size (weights)
- Importance sampling can achieve speedup factor of ≈ 2 on benchmark
  $\rightarrow$ could generate twice the effective samples size using same computing time
  (statistical uncertainty reduced by a factor of $\approx \sqrt{2}$)
- Initial studies suggest higher speedups for selections with lower filter efficiency
  $\rightarrow$ but also less training data available
  $\rightarrow$ investigate training on-the-fly (train while generating new MC)
- Benchmark Model and inference implemented in Belle II software
- Implementation of general training procedure ongoing

Backup

Avoid or correct bias using event weights

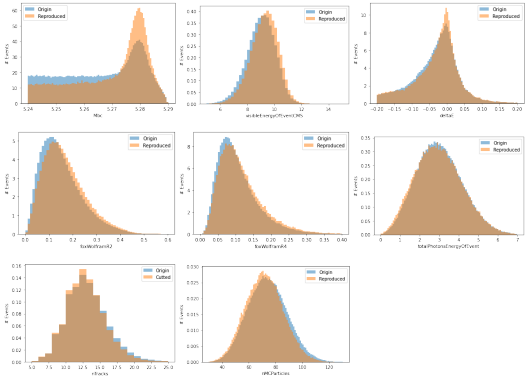
Sampling method	**Reweighting method**
Use NN output as **probability to keep event**	Use NN output as **score to cut on**
Weight events by inverse probability $\rightarrow$ like importance sampling $\rightarrow w = \frac{1}{p_{\mathrm{NNfilter}}}$	Reweight events to correct bias $\rightarrow$Gradient Boosting Decision Trees $\rightarrow w = \frac{1}{p_{\mathrm{GBDT}}}$
No bias by construction	No bias for quantities included in reweighting (if reweighting performs well) $\rightarrow$ needs validation
Use speedup as loss function in training	Use binary cross entropy in training

Metric to optimize: **Speedup**
"How much faster can i produce the same number of events (in terms of *effective sample size*) using the same computation time?"

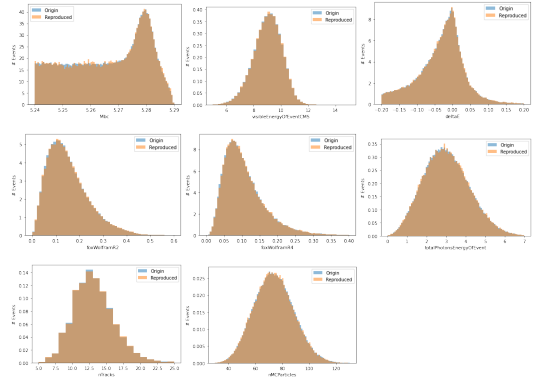
Effective sample size: $\frac{(\sum_i w_i)^2}{\sum_i w_i^2}$
$\rightarrow$ Maximum achieved for Sampling Method: ≈ 2, for Reweighting method $\approx 5 - 6$

GBDT reweighting

Original



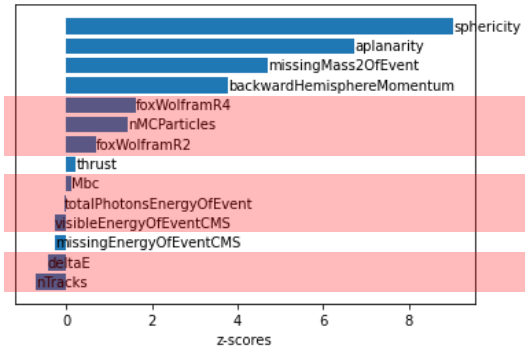
Reweighted



(8 variables above black line used to derive weights)

More quantitative: p-values from KS-Test

Marked values used to fit the GBDT (derive weights)



$\rightarrow$ still some deviations left for quantities not used in GBDT training