smartBKG

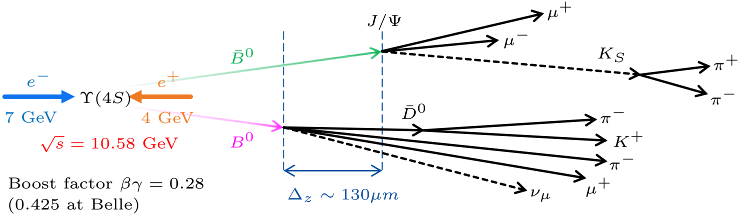
Selective Background Monte Carlo Generation

The Skimulators:

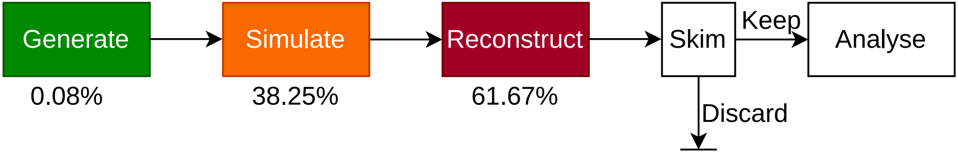
James Kahn (KIT), Kilian Lieret (LMU)
Andreas Lindner (LMU), Emilio Dorigatti (LMU)

September 16, 2019

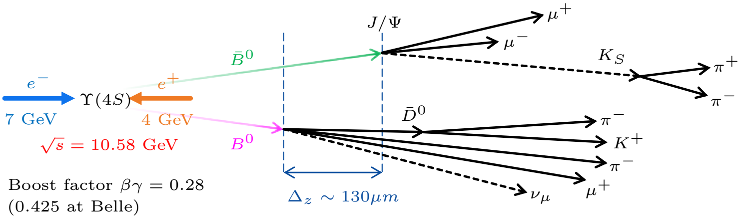
Project Overview



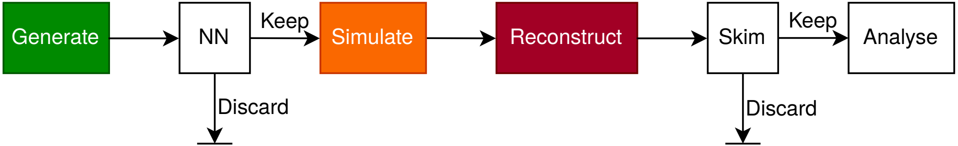
- Simulation of particle collisions computationally expensive, but most of the results are uninteresting and thrown away
- Idea: Can we figure out whether a collision is uninteresting at an early stage?



Project Overview



- Simulation of particle collisions computationally expensive, but most of the results are uninteresting and thrown away
- Idea: Can we figure out whether a collision is uninteresting at an early stage?
  - **Skip expensive steps**



Dataset

Three categories of decay data:

- Full charged B meson reconstruction
- Full neutral B meson reconstruction
- Time-dependent CP violation

For each category:

$\sim 300,000$ particle collision processes with binary classification labels

- $\Upsilon(4S)$ (300553)
  - $\bar{B}^0$ (-511)
    - J/ψ (443)
      - μ^+ (-13)
      - μ^- (13)
    - K_S^0 (310)
      - pi^- (-211)
      - pi^+ (211)
  - B^0 (511)
    - $\bar{D}^0$ (-421)
      - pi^- (-211)
      - K^+ (321)
      - pi^- (-211)
    - μ^+ (-13)
    - ν_μ (14)

Feature	Definition
PDG code	Identifier of particle type and charge.
Mother PDG code	Particle parent PDG code.
Mass	Particle mass in GeV/c^2.
Charge	Electric charge of the particle.
Energy	Particle energy in GeV.
Momentum	Three momentum of the particle in Gev/c.
Production time	Production time in ns relative to $\Upsilon(4S)$ production.
Production vertex	Coordinates of particle production vertex.
Status bit	Bitmask representing MC production conditions.

Dataset

Three categories of decay data:

- Full charged B meson reconstruction
- Full neutral B meson reconstruction
- **Time-dependent CP violation**

For each category:
$\sim$ 300,000 particle collision processes with binary classification labels

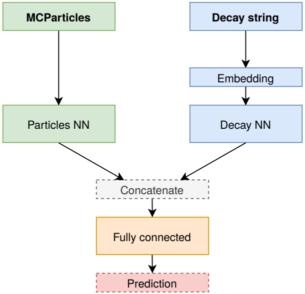
$\Upsilon(4S)$ (300553)
- $\bar{B}^0$ (-511)
  - J/ψ (443)
    - μ^+ (-13)
    - μ^- (13)
  - K_S^0 (310)
    - pi^- (-211)
    - pi^+ (211)
- B^0 (511)
  - $\bar{D}^0$ (-421)
    - pi^- (-211)
    - K^+ (321)
    - pi^- (-211)
  - μ^+ (-13)
  - ν_μ (14)

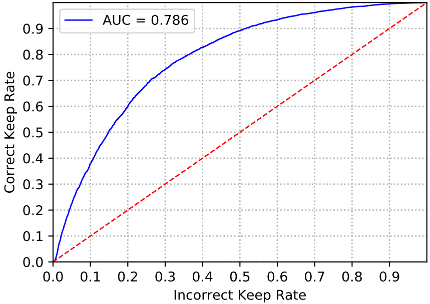
Feature	Definition
PDG code	Identifier of particle type and charge.
Mother PDG code	Particle parent PDG code.
Mass	Particle mass in GeV/c^2.
Charge	Electric charge of the particle.
Energy	Particle energy in GeV.
Momentum	Three momentum of the particle in Gev/c.
Production time	Production time in ns relative to $\Upsilon(4S)$ production.
Production vertex	Coordinates of particle production vertex.
Status bit	Bitmask representing MC production conditions.

Goals

Previous work by James Kahn:

- Convolutional neural networks: residual, recurrent, vanilla...





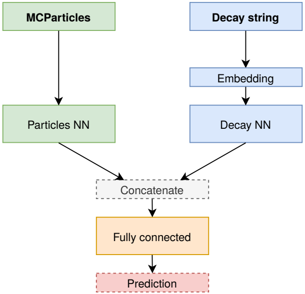
Goals for this week:

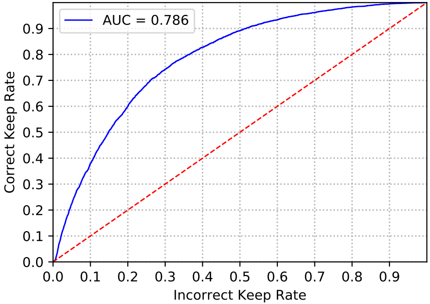
- Implement graph convolution networks
- Decorrelate network from selected kinematics

Goals

Previous work by James Kahn:

- Convolutional neural networks: residual, recurrent, vanilla...





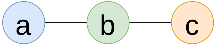
Goals for this week:

- Implement graph convolution networks
- ~~Decorrelate network from selected kinematics~~

Original Graph Convolutional Networks (GCN)

Propagation rule of layer activations $H^{(l)}$

$$H^{(l+1)} = \sigma\left(\tilde{D}^{-1}\tilde{A}\tilde{D}^{-1}\,H^{(l)}W^{(l)}\right)$$



$$\begin{aligned} H^{(0)} &= X \\ \tilde{A} &= A + I_N \\ \tilde{D}_{ii} &= \textstyle\sum_j \tilde{A}_{ij} \end{aligned}$$

$$\tilde{A}^{N\times N} = A + I = \begin{array}{c c} & \begin{array}{ccc} a & b & c \end{array} \\ \begin{array}{c} a \\ b \\ c \end{array} & \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \end{array}$$

Thomas N. Kipf, Max Welling, *Semi-Supervised Classification with Graph Convolutional Networks* (ICLR 2017)

Modified GCN

$$H^{(l+1)} = \sigma\left(H^l \left(\tilde{D}^{-1}\tilde{A}\tilde{D}^{-1} W_1^{(l)}\right)^T W_2^{(l)}\right)$$

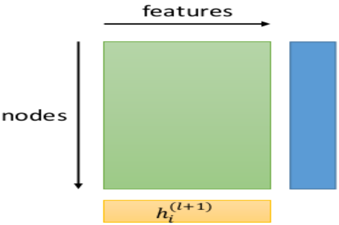
Intuition: **custom weights for each node, considering neighbors**

- Weight vector for every node: W_1 (size: $N \times F$)
- Multiply by $\tilde{D}^{-1}\tilde{A}\tilde{D}^{-1} \longrightarrow$ weight vector for every node as the average of the vectors of neighboring nodes (output size: $N \times F$ again)
- Transpose and multiply by $W_2 \longrightarrow$ custom dense layer for each node (W_2 size: $N \times U$, result size: $F \times U$)
- Multiply by H: transform every node in the previous layer with its own custom layer (H size: $N \times F$, result size: $N \times U$)

Modified Node Aggregation

- **Soft attention** with weights given by D and values given by $\tanh(E)$
- E, D: Output of dense layers applied independently to the features of each node

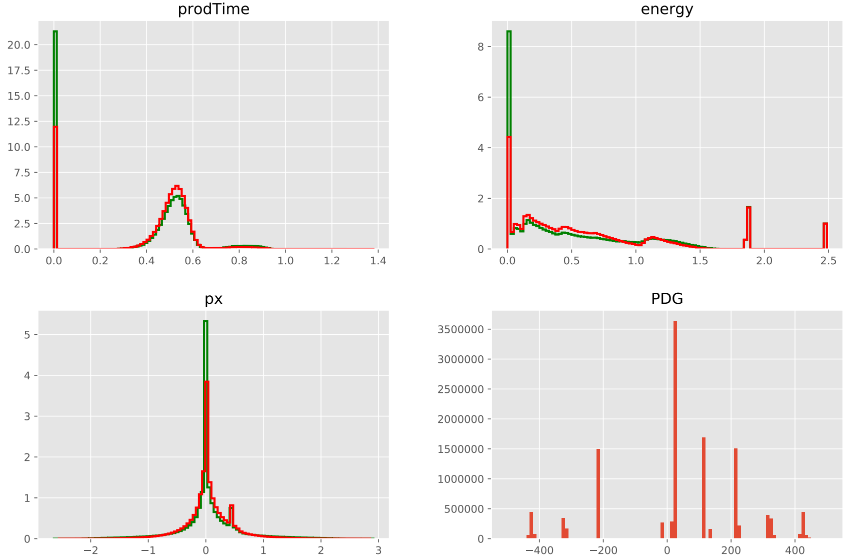
$$h_i^{(l+1)} = \sum_j \frac{\exp \sum_k D_{(jk)}^{(l)}}{\sum_l \exp \sum_k D_{(jk)}^{(l)}} \tanh E_{ji}^{(l)}$$



Adapted from: Yujia Li, Richard Zemel, Marc Brockschmidt, Daniel Tarlow, *Gated Graph Sequence Neural Networks*, ICLR 2016

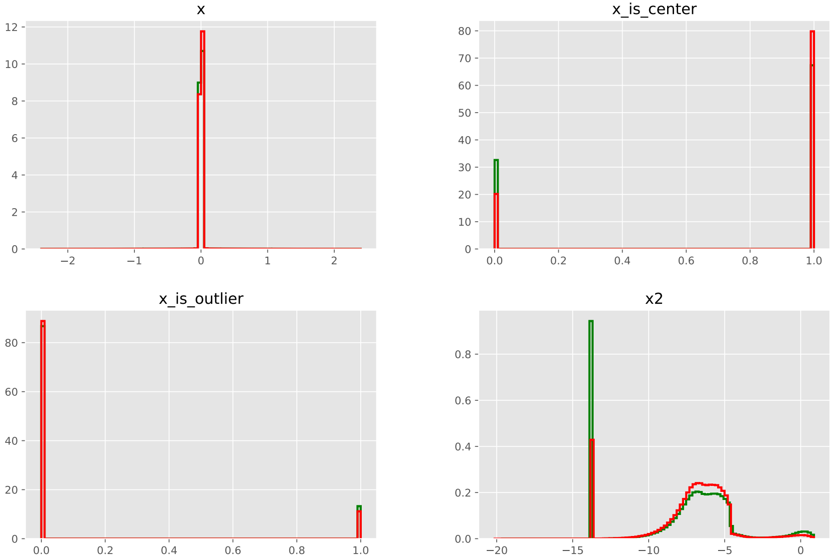
Features

Green: 'pass'. red: 'fail'



Features II

x, y, z coordinates: Hard to normalize

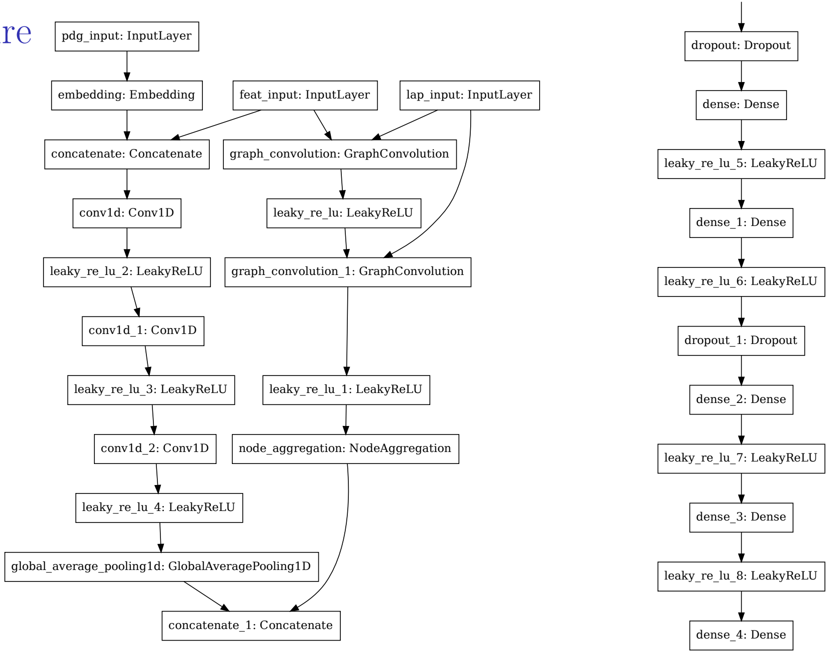


Architecture

```
p = GraphConvolution(128, kernel_regularizer=regularizers.l2(l2_strength))([feature_input, laplacian_input])
p = layers.LeakyReLU()(p)
p = GraphConvolution(128, kernel_regularizer=regularizers.l2(l2_strength))([p, laplacian_input])
p = layers.LeakyReLU()(p)
p = NodeAggregation(256, kernel_regularizer=regularizers.l2(l2_strength))(p)

m = layers.Concatenate()([pdg_l, feature_input])
m = layers.LeakyReLU()(layers.Conv1D(128, 5)(m))
m = layers.LeakyReLU()(layers.Conv1D(128, 5)(m))
m = layers.LeakyReLU()(layers.Conv1D(128, 5)(m))
m = layers.GlobalAveragePooling1D()(m)

l = layers.Concatenate()([m, p])

l = layers.Dropout(0.5)(l)
l = layers.LeakyReLU()(layers.Dense(512, kernel_regularizer=regularizers.l2(l2_strength))(l))
l = layers.LeakyReLU()(layers.Dense(256, kernel_regularizer=regularizers.l2(l2_strength))(l))
l = layers.Dropout(0.3)(l)
l = layers.LeakyReLU()(layers.Dense(128, kernel_regularizer=regularizers.l2(l2_strength))(l))
l = layers.LeakyReLU()(layers.Dense(32, kernel_regularizer=regularizers.l2(l2_strength))(l))

output_layer = layers.Dense(1, activation='sigmoid')(l)
```

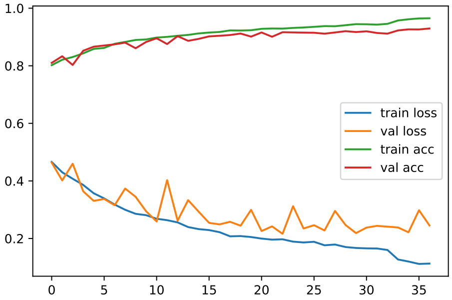
- Class weights
- Early stopping
- Reduce learning rate on plateau
- Model checkpoint – save only best

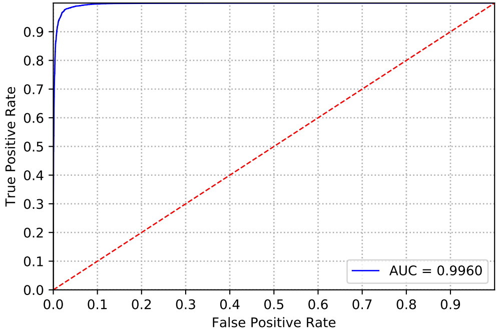
Architecture

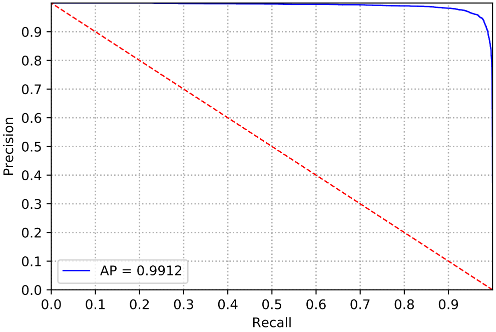


Results

- Train on random subset of 50k processes
- Test on 40k independent processes







Next steps

- Bias quantification/mitigation
- Hyperparameter optimisation
- Train on large dataset and other skims
- Talk accepted at CHEP2019 (Computing in High Energy Physics) – proceedings

Backup

Results (Smaller network)