Reinforced Sorting Networks for Particle Physics Analyses

Martin Erdmann, Benjamin Fischer, Dennis Noll, Yannik Rath, Marcel Rieger, David Schmidt, Marcus Wirtz

ACAT 2019
14 March 19
Physics analyses often highly depend on the order of their input variables

- Example: Classification - ttH vs ttbb (6 jets, lepton, neutrino, Pythia+Delphes)
- Usual approach: $p_T$ order

Motivation

<table>
<thead>
<tr>
<th>Order</th>
<th>Accuracy [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gen</td>
<td>87.3</td>
</tr>
<tr>
<td>$p_T$</td>
<td>69.5</td>
</tr>
</tbody>
</table>

Not optimal

Talk by Yannik
Idea: Train mechanism (Neural Network) which outputs the best permutation

Problem 1: Many possible permutations (n!)
Solution: Divide & conquer
1. Look at a subset of possible permutations
2. Explore subset
3. Choose next subset
=> Continue until best permutation is found

Problem 2: Best permutation generally not known
Solution: Reinforcement approach
• Simultaneously train analysis and sorting
  • Train sorting with analysis feedback
  • Train analysis with sorting feedback
• Evaluation:
  ▪ Input $p_T$ sorted events
  ▪ **Sorting** network sorts particles $S(\text{Event})$

• **Analysis** evaluates sorted events (trainable)
**Sorting Mechanism**

*Network* outputs position for every jet:
- Small position: front
- Large position: end
- Uniformly between 0 and 1

*Logic* algorithm applies the positions and rearranges the jets.
Complete Mechanism - Reinforced Training

- Training:
  - Sorting network sorts particles $S(\text{Event})$
  - Policy suggests new orderings $P_1(S(\text{Event}))$, $P_2(S(\text{Event}))$, $P_3(S(\text{Event}))$
    - Subset of possible orderings based on current ordering
  - Analysis evaluates sorted (and permuted) events
Complete Mechanism - Reinforced Training

- **Training:**
  - Sorting network sorts particles $S(\text{Event})$
  - Policy suggests new orderings $P_1(S(\text{Event}))$, $P_2(S(\text{Event}))$, $P_3(S(\text{Event}))$
    - Subset of possible orderings based on current ordering
  - Analysis evaluates sorted (and permuted) events
Permutation Policy

- Two different permutation policies:
  - Basic approach (Pool) and extension (Tree)

- All pair-wise switches
  - $N_{pool}(n) = \mathcal{O}(n^2)$
  - $N_{pool}(6) = 15$

- Policy (Pool):
  - Draw from pool $P_i \in \mathcal{P}$
Permutation Policy

- Two different permutation policies:
  - Basic approach (Pool) and extension (Tree)

- Chain permutations to reach every state:
  \[ P_{\text{any}}(S(\text{Event})) = P_z(P_y(...(P_x(S(\text{Event})))...)) \]

- Policy (Tree):
  - Draw from pool (\( \mathcal{P} \)) in every node
  - Use tree mechanics to reach valuable nodes

- All pair-wise switches
  - \( N_{\text{pool}}(n) = \mathcal{O}(n^2) \)
  - \( N_{\text{pool}}(6) = 15 \)

- Policy (Pool):
  - Draw from pool \( P_i \in \mathcal{P} \)
Train sorting to output $P_3(S(\text{Event}))$:
- **Positions** (targets) follow from $P_3$
- Gaussian smear
- Supervised training
Complete Mechanism - Reinforced Training

- Train sorting to output $P_3(S(Event))$:
  - Positions (targets) follow from $P_3$
  - Gaussian smear
  - Supervised training

$\downarrow = \text{smallest loss}$

![Diagram showing the complete mechanism with arrows indicating the flow from Event to Policy to Analysis, and decision points with $P_1(S(Event))$, $P_2(S(Event))$, and $P_3(S(Event))$.]
• Train sorting to output $P_3(S(\text{Event}))$:
  - Positions (targets) follow from $P_3$
  - Gaussian smear
  - Supervised training
Train sorting to output $P_3(S(\text{Event}))$:
- **Positions** (targets) follow from $P_3$
- Gaussian smear
- Supervised training
- Inputs for both networks: *only four-vectors* (Pythia + Delphes)

- Sorting Network:
  - Feed forward
    - 6 SeLu Layers, 256 Nodes
    - $L_2$ regularization + Layer Normalization
  - Adam Optimizer

- Analysis Network:
  - Lorentz-Boost Network (1812.09722, talk by Yannik)

- Training:
  - Schedule:
    - Batch wise
    - Epoch wise
  - Pre-training:
    - Let analysis network converge before starting to train
    - Pre-train sorting to output $p_T$
• Sorting network learns an intrinsic structure
• Performance of classifier increases
- Train analysis with sorting, evaluate on $p_T$ order
  - Accuracy drops to 50%
  - Analysis network relies on sorting
  $\rightarrow p_T$ order disfavored
Conclusion

- Reinforced input sorting may benefit various NN applications
  - **Agent:**
    - Sorting of inputs (e.g. jets)
  - **Policy:**
    - Classic permutations
    - Tree search
  - **Environment:**
    - Physics analysis (e.g. LBN)
- Especially promising in scenarios with a-priori unknown “best” order
Backup
Artificial Data - Dataset

- Build artificial dataset - with explicit sorting information:
  - Sorting information: one random number
  - „Features“: random numbers
  - Task: reconstruct weighted alternating sum over the „Features“
    - Easy task, if input order is fixed
    - Nearly impossible if input order is random
    - Regression

<table>
<thead>
<tr>
<th>Sorted: easy</th>
<th>Random: impossible</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Row1</strong> 0.3</td>
<td><strong>Row1</strong> 0.3</td>
</tr>
<tr>
<td><strong>Row2</strong> 0.8</td>
<td><strong>Row4</strong> 0.5</td>
</tr>
<tr>
<td><strong>Row3</strong> 0.2</td>
<td><strong>Row2</strong> 0.8</td>
</tr>
<tr>
<td><strong>Row4</strong> 0.5</td>
<td><strong>Row5</strong> 0.9</td>
</tr>
<tr>
<td><strong>Row5</strong> 0.9</td>
<td><strong>Row6</strong> 0.1</td>
</tr>
<tr>
<td><strong>Row6</strong> 0.1</td>
<td><strong>Row3</strong> 0.2</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
  + 0 \times 0.3 & \quad - 1 \times 0.8 & \quad + 2 \times 0.2 & \quad - 3 \times 0.5 & \quad + 4 \times 0.9 & \quad - 5 \times 0.1 \\
  & & & & & \\
  & & & & & = 1.2
\end{align*}
\]
- Sorting model converges fast
- Finds stable sorting
- Outputs gaussian