Faculty of Physics
Warsaw University of Technology

Towards more precise correlation studies with machine learning-based particle identification with missing data

Łukasz Graczykowski
in collaboration with
M. Janik, M. Karwowska, S. Monira, K. Deja, M. Kasak,
M. Jakubowska, M. Mytkowski, M. Olędzki

Warsaw, Poland
15 December 2024

Based on:
EPJ C 84 (2024) 7, 691
JINST 19 (2024) 07, C07013

Goals

- Use ALICE and its data as a **unique environment** for **Machine Learning (ML) research**

- Identify **areas** where both ALICE (or HEP in general) and ML communities can **mutually benefit** from each other

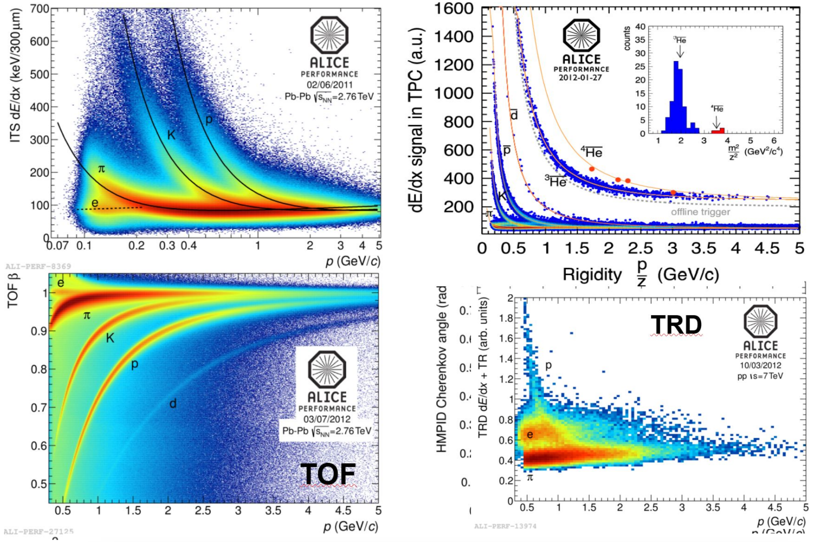
- Our solutions should be **easily applicable to other experiments** with similar PID capabilities

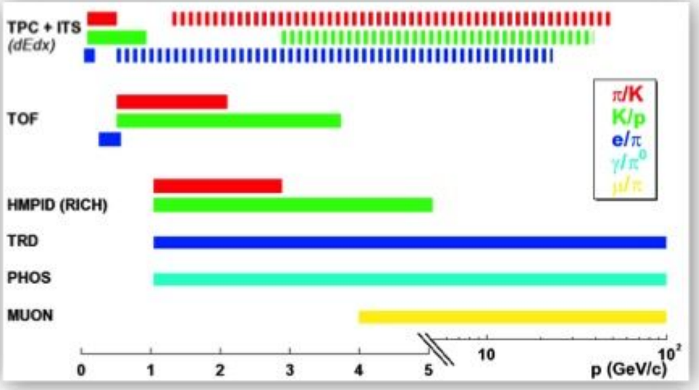
- **Disclaimer:**
    - I'm a **physicist without a big ML expertise** – just started my (human) learning of machine learning :)
    - My task is to **guide and coordinate the work of WUT ML computer scientists** within ALICE
    - The solution may be **complicated** (*shooting a sparrow with a cannon*), but the balance is to keep the project interesting for ML itself and be useful for us at the same time!

Particle identification (PID)

Aim: provide high purity samples of particles of a given type

- **an essential step** for many physics analyses, especially **correlations of identified particles**
- we use **ALICE as our R&D environment**
- **a distinguishing feature** of ALICE among the LHC experiments:
    - identification of particles of momenta in a **very wide momentum range**
    - practically **all known techniques** employed: dE/dx energy loss, time-of-flight, Cherenkov radiation for hadrons and transition radiation for electrons

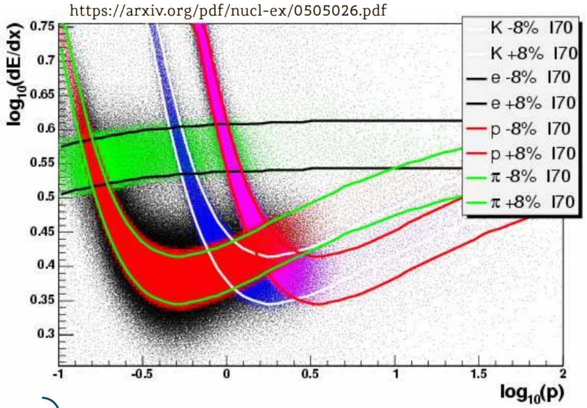




https://arxiv.org/abs/1709.00288

Present state-of-art

1. **Traditional method:**
   - hand-crafted selections of selected quantities, e.g., <u>nσ</u>
   - <u>problems:</u>
     - overlapping signals
     - <u>high purity at the cost of low efficiency</u>
     - time-consuming optimization



2. **Bayesian method** (ALICE, EPJ Plus 131 (2016) 168)**:**
   - updating probability of an hypothesis with each new evidence
   - priors = best guess of true particle yields per events
   - posteriors ~ purity
   - increased purity, results consistent with the traditional method

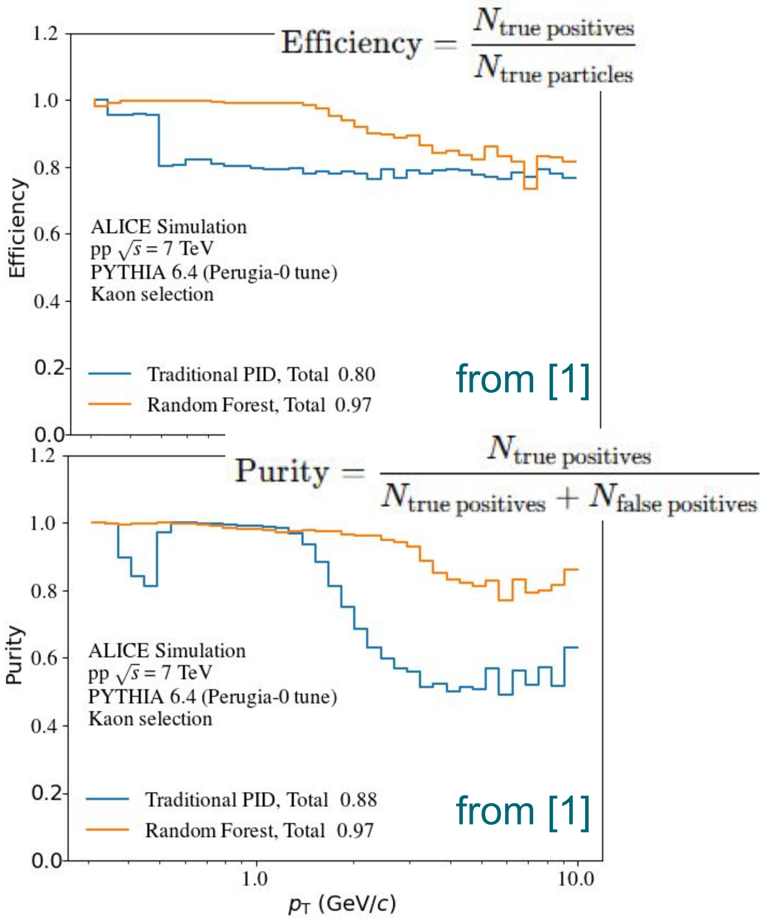
not covered in this talk
yields similar results

Both methods available in O^2 – ALICE Run 3 software

Can we do any better?

Yes!
With ML :)

ML for PID



Advantages of the ML approach to PID:

- **classification** – a ''standard'' ML problem
- can use **more track parameters** as input
- can learn **more complex relationships**
- many software libraries available

Note also **the limitations**:

- depends on **quality of the training data** (MC)
- hard to obtain **systematic uncertainties**
- hard to follow classifier's ''reasoning'' (**black box**)

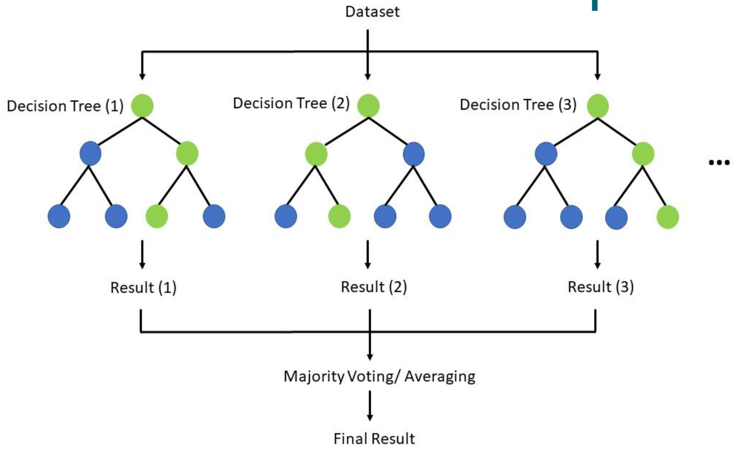
Our **first works** show ML can **greatly improve** purity and efficiency:

1. Random Forest: T. Trzciński, Ł. Graczykowski, M. Glinka, ALICE Collaboration. Using Random Forest classifier for particle identification in the ALICE experiment. Conference on Information Technology, Systems Research and Computational Physics, pp. 3-17. 2018
2. Domain Adaptation: M. Kabus, M. Jakubowska, Ł. Graczykowski, K. Deja, ALICE Collaboration. Using machine learning for particle identification in ALICE. JINST, v. 17, p. C07016. 2022

Proof-of-concept: Random Forest

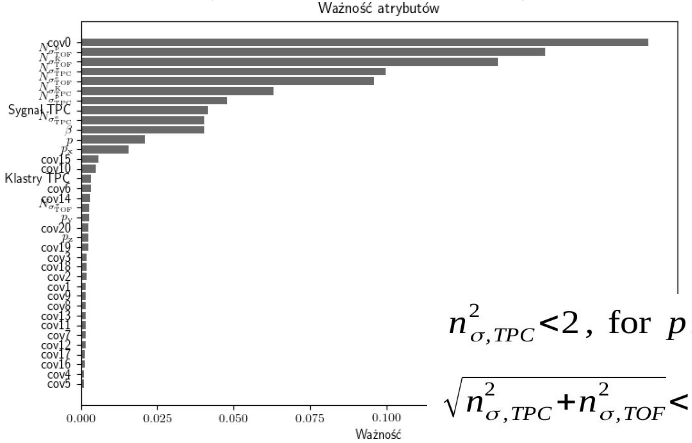
T. Trzciński, Ł. Graczykowski, M. Glinka, Conference on Information Technology, Systems Research and Computational Physics, 3-17. 2018

2018



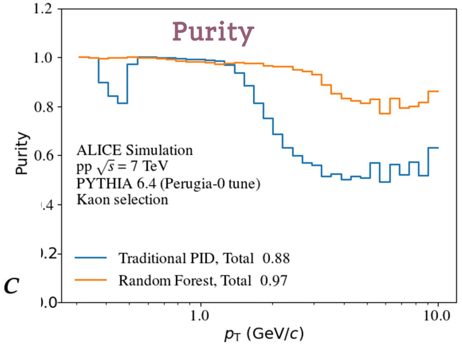
https://en.m.wikipedia.org/wiki/File:Random_forest_explain.png

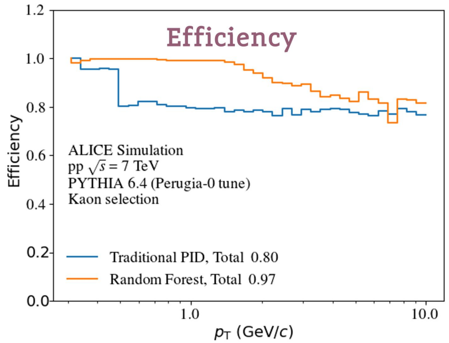
- Preliminary work with ALICE Run 2 data
- First solution - **Random Forest**
- Model works on **high-level track parameters**
- Depends on the **quality of Monte Carlo sample** and **post-processed information** (i.e. nσ calculation)
- Can be used **only for analysis-specific use-case** (concrete dataset and specific particle selection)
  - model has to be **trained by the specific end user**



$$n^2_{\sigma,TPC} < 2, \text{ for } p \leq 0.5 \text{ GeV}/c$$

$$\sqrt{n^2_{\sigma,TPC} + n^2_{\sigma,TOF}} < 2, \text{ for } p > 0.5 \text{ GeV}/c$$





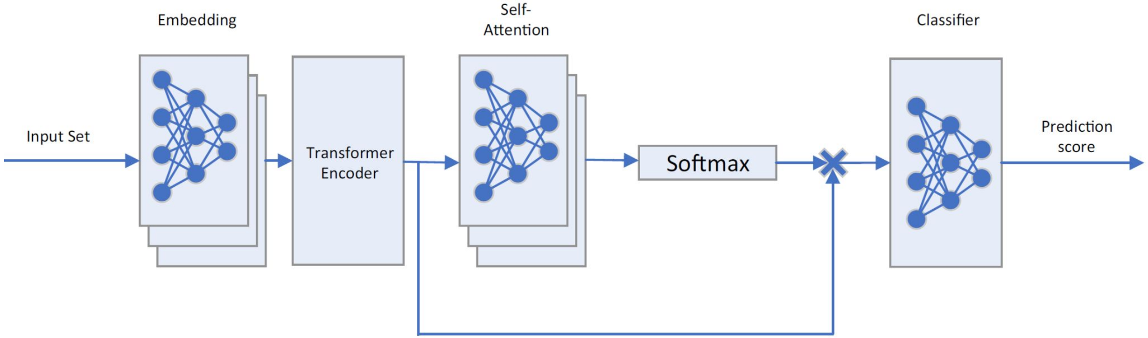
Current solution - our model

- Solution **general enough** to be used for variety of analyses
- **At present our input data has 19 features:** i.e. momentum components, charge sign, DCA_{XY}, DCA_{Z}, detector signals (TPC dE/dx, TOF time, TRD signal), etc.
- **Data might be missing** from one or more detectors due to, e.g., too small p_T
- In **“standard” ML** approaches dealing with such cases, people use **data imputation** or **case deletion** - however artificially altered data may bias the physics results!
  - **Challenge:** classify particles without making any assumptions about the missing values

- The **proposed model** is much more advanced than the proof-of-concept solution and has **4 steps** (see next slides)
- For **details**, see our **two papers**:
  - EPJ C 84 (2024) 7, 691
  - JINST 19 (2024) 07, C07013

Current solution - our model

M. Kasak, K. Deja. M. Karwowska,
M. Jakubowska, ŁG
M. Janik, EPJ C 84 (2024) 7, 691
M. Karwowska, ŁG, K. Deja, M. Kasak,
M. Jaik, JINST 19 (2024) 07, C07013



1. **Feature Set Embedding** to encode the inputs

2. **Transformer Encoder** to detect patterns in the input

3. Additional **self-attention** network to pool the encoder output set into a single vector

4. **Classifier** a simple neural network to classify a given particle type

Inspired by AMI-Net proposed for medical diagnosis from incomplete data (medical records)

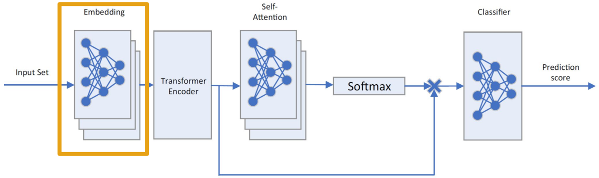
Attention-based Multi-instance Neural Network for Medical Diagnosis from Incomplete and Low Quality Data

Zeyuan Wang[1,3], Josiah Poon[1], Shiding Sun[2], Simon Poon[1*]
[1]School of Computer Science, The University of Sydney, Syndey, Australia
[2]School of Mathematics, Renmin University of China, Beijing, China
[3]Beijing Medicinovo Technology Co.,Ltd., Beijing, China
[1,3]*zwan7221@uni.sydeny.edu.au*, [1]*{josiah.poon, simon.poon}@sydney.edu.au*, [2]*sunshiding@ruc.edu.cn*

2019 International Joint Conference on Neural Networks (IJCNN)

details on slide 15

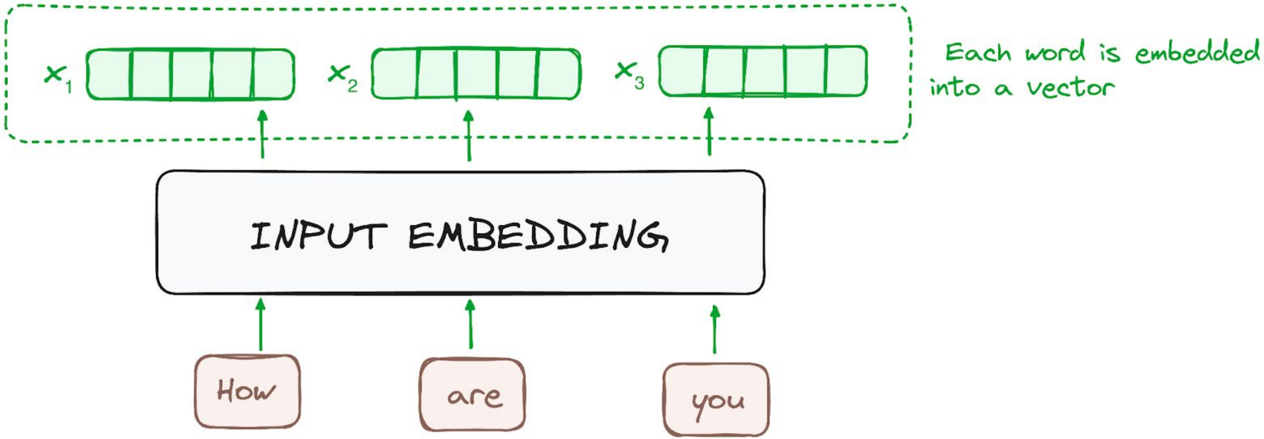
Step 1: Embedding



- **Embedding** is a technique to **handle complex data**

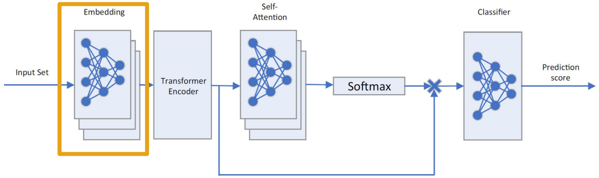
- It works by **converting high-dimensional data** (i.e. sequences of words, documents, images, etc.), **into lower-dimensional** and **abstract vector representation (embedding space)**

- It allows for capturing **meaningful relationships between data entities** (words, etc.)



https://www.datacamp.com/tutorial/how-transformers-work

Step 1: Feature Set Embedding

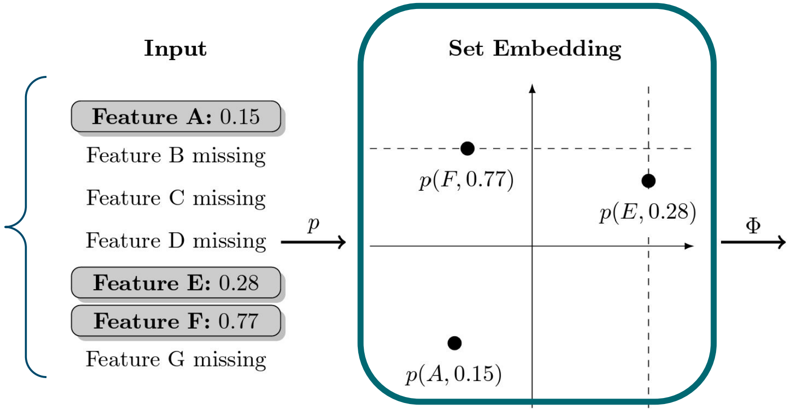


Missing data challenge: classify <u>without making any assumptions</u> about the missing values

Feature Set Embedding (<u>NIPS 2010 article</u>):

- <u>instead of vectors</u>, use <u>(feature,value) pairs</u>; no value → no pair
  - no need to model missing data (i.e. imputation)
- pairs in embedding space: <u>similar features are close to each other</u>
- pairs are then combined (by NN) into vectors (<u>embeddings</u>)

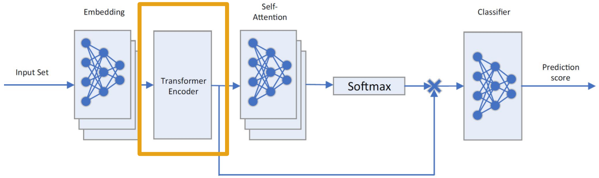
track
19 features in our case



embeddings
32 vectors in our case

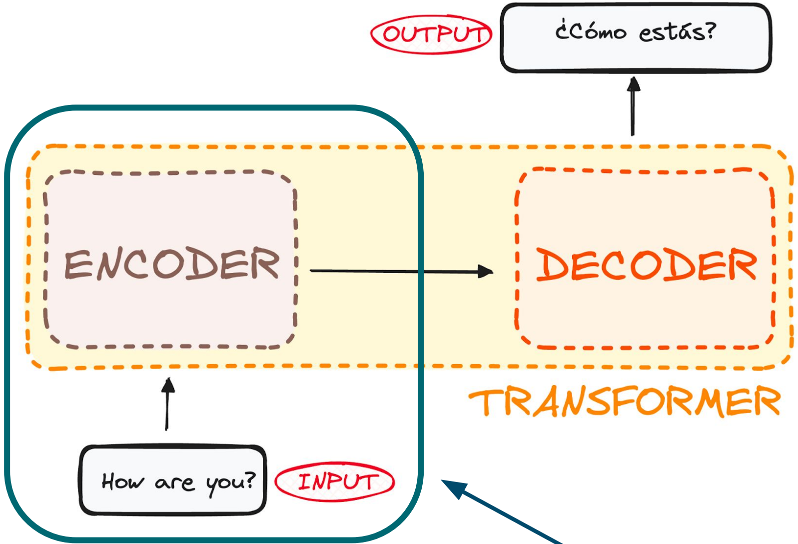
Image source: <u>NIPS 2010 article</u>

Step 2: Transformer Encoder



Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com



https://www.datacamp.com/tutorial/how-transformers-work

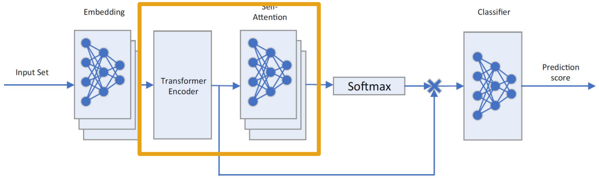
- Idea from original **Transformer** architecture proposed by Google (NIPS 2017 article)
- Developed for **transforming input data** into a **contextualized representation** on the output
- Transformer currently serves as **basis for the Natural Language Processing** tools (such as **ChatGPT**)
- In our case, **vectors from Embedding are processed by the Encoder only**
  - we do not need Decoder in our use-case

Steps 2 and 3: self-attention

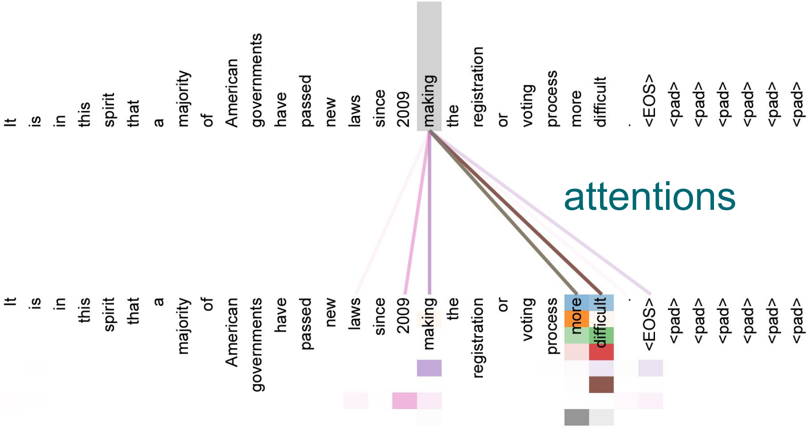
- **Attention** and **self-attention** are mechanisms used to help model focus on relevant parts of the input data
  - **self-attention** focuses on **relationships within the same input sequence**

- ***Example:*** "The cat sat on the mat"
  - when processing the word "cat," it considers other words (i.e. "the" or "mat") to understand their contribution to the meaning of "cat" (<u>in the context of the entire sentence</u>)

- Usage of **self-attention in Transformer architecture:**
  - in **single-head attention**, a <u>single set</u> of attention scores is used to focus on a particular part of the input sequence → limited ability to capture different relationships
  - **multi-headed attention** uses multiple attention heads, where each head focuses on different parts of the input <u>simultaneously</u>



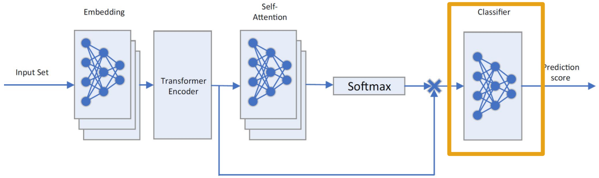
We use **self-attention twice**:
- in **Transformer Encoder**
- before **Classifier**



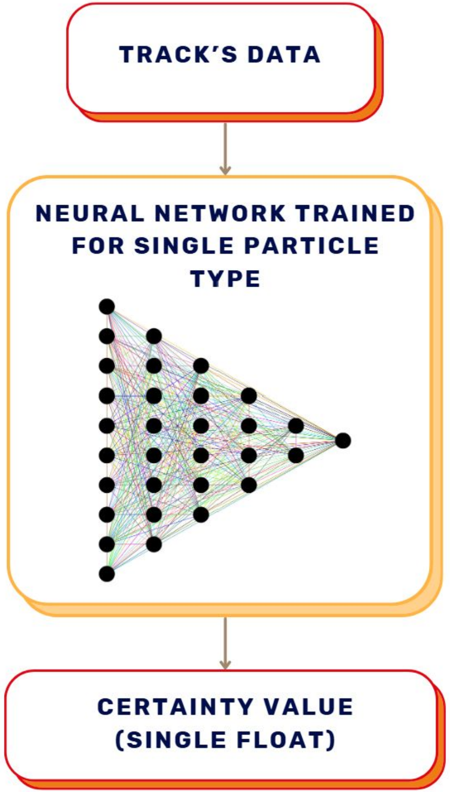
colors = attentions from different heads

NIPS 2017 article

Step 4: classification

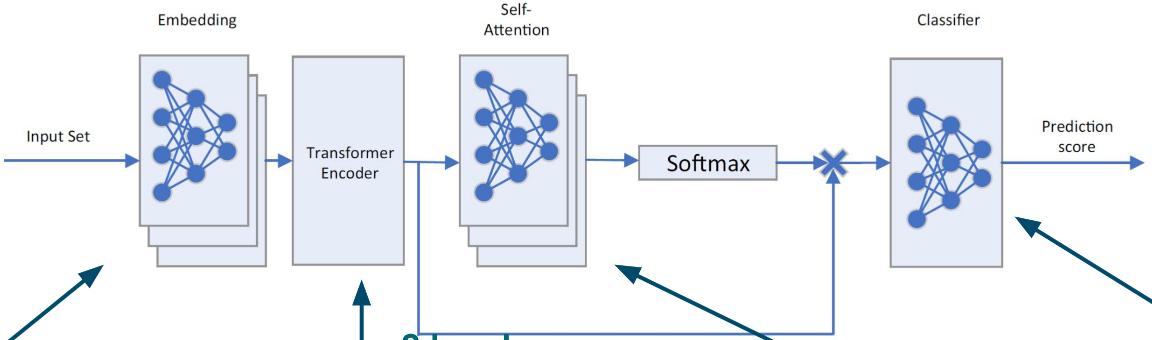


- Single **output vector** from the **Self-attention network** is propagated to the **Classifier**
- **Classifier** is represented by **one simple neural network** (one hidden layer) **per particle** (**one vs all** approach)
  - the same architecture is used **separately** for pions, kaons, protons
- **Classifier score:** logistic function $f(x) = \frac{1}{1+e^{-x}}$ in range (0, 1) represents **"certainty"** that a given particle belongs to the given particle type
  - users can still **balance the efficiency and purity** by setting their own **threshold on the "certainty" value**



Details of the architecture

M. Kasak, K. Deja. M. Karwowska,
M. Jakubowska, ŁG
M. Janik, EPJ C 84 (2024) 7, 691
M. Karwowska, ŁG, K. Deja, M. Kasak,
M. Jaik, JINST 19 (2024) 07, C07013



This model is applied **separately** for pions, kaons, protons

“certainty”

2 heads
in each head
2 layers (att.+NN)

Embedding			Transformer encoder					Self-attention			Classifier		
			Encoder layer										
			Multi-head attention		Neural network								
Input	Hidden	Output	Dimension	Heads	Input	Hidden	Output	Input	Hidden	Output	Input	Hidden	Output
19	128	32	32	2	32	128	32	32	64	32	32	64	1

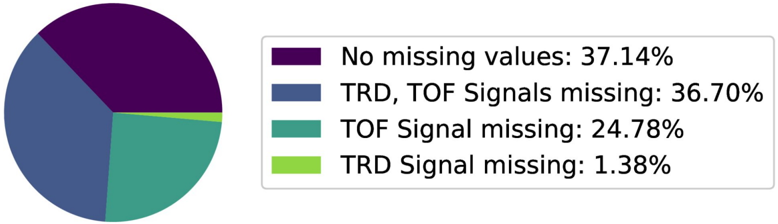
- **dropout** value 0.1 at the output of embedding and each Transformer Encoder layer (to limit overfitting)
- **softmax function** is applied to obtain weights to create a single output (weighted average) vector
- **activation function** (between neural network layers): *ReLU (Rectified Linear Unit)*
- **loss function** that is minimized is *binary cross entropy* (for *one vs all* approach)
  - to minimize differences between *predicted* and *true* values (labels from MC truth data)

Test setup

M. Kasak, K. Deja. M. Karwowska, M. Jakubowska, ŁG
M. Janik, EPJ C 84 (2024) 7, 691
M. Karwowska, ŁG, K. Deja, M. Kasak, M. Jaik, JINST 19 (2024) 07, C07013

- **Dataset:** Run 2 general-purpose MC (Pythia 8) pp at $\sqrt{s}$ = 13 TeV with full detector simulation with GEANT 4 (both MC truth and reconstructed data are used)
- **Standard nσ method:**
  $|n_{\sigma,\,TPC}| < 3$ for $p_T < 0.5$ GeV/c, $\sqrt{(n_{\sigma,\,TPC}^2 + n_{\sigma,\,TOF}^2)} < 3$ for $p_T \geq 0.5$ GeV/c
- **Dataset details:**
  - no. tracks: ~2.7 million
  - 30% - test dataset
  - from the 70% of the rest:
    - 70% training
    - 30% validation

Missing data distribution



Results – pions, kaons, protons

M. Kasak, K. Deja. M. Karwowska,
M. Jakubowska, ŁG
M. Janik, EPJ C 84 (2024) 7, 691
M. Karwowska, ŁG, K. Deja, M. Kasak,
M. Jaik, JINST 19 (2024) 07, C07013

F_1 = (purity x efficiency) / (purity + efficiency)

FSE + attention with **very good scores** of **F_1, purity (precision)** and **efficiency (recall)**

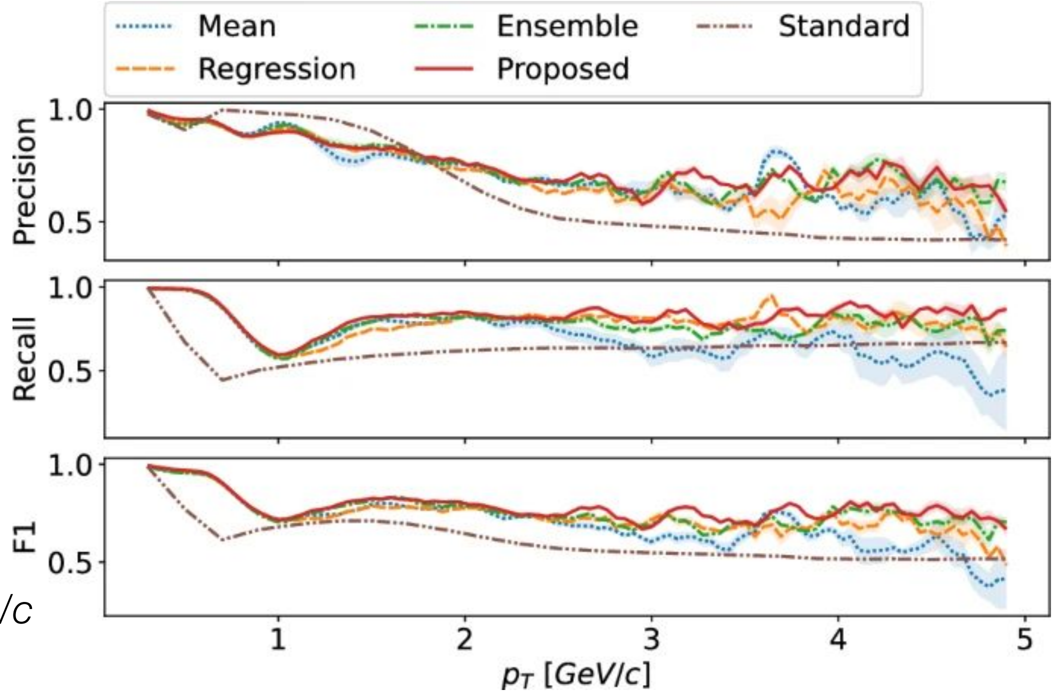
Proposed model (FSE+Attention)
compared to **other approaches:**

- **imputation:**
  artificial bias in data
  - mean
  - regression

- **NN ensemble** (4 networks)**:**
  potentially large complexity

- **standard:**
  nσ method
  $|n_{\sigma,\,TPC}| < 3$ for $p_T < 0.5$ GeV/c
  $\sqrt{(n_{\sigma,\,TPC}^2 + n_{\sigma,\,TOF}^2)} < 3$ for $p_T \geq 0.5$ GeV/c

kaon selection



Conclusions

R&D phase of the ML PID (almost) **finished!**

FSE+Attention model works well for the three basic identified hadron species (pions, kaons, protons)

Lots of work done, but still more ahead!

Plans for future:

- tests with Run 3 data with new O^2 analysis framework (*ongoing*)
- automation of model training and regular training of models for new Run 3 datasets (*implementation*)
- extending the model with domain adaptation (*still to do*)
- advertise PID ML among ALICE analyzers (*to do when fully implemented*)

The work has been carried out by an interdisciplinary team from 4 faculties of WUT:

- *Physics:* Ł. Graczykowski (*general idea, coordination, evaluation*), M. Janik (*evaluation*), M. Karwowska (*implementation*), S. Monira (*tests of implemented model*)
- *Electronics and Information Technology:* Kamil Deja, Miłosz Kasak (*ML R&D*)
- *Electrical Engineering:* Monika Jakubowska (*coordination, evaluation*)
- *Mathematics and Computer Science:* Marek Mytkowski, Mateusz Olędzki (*implementation*)

Thank you!

Run 2 results

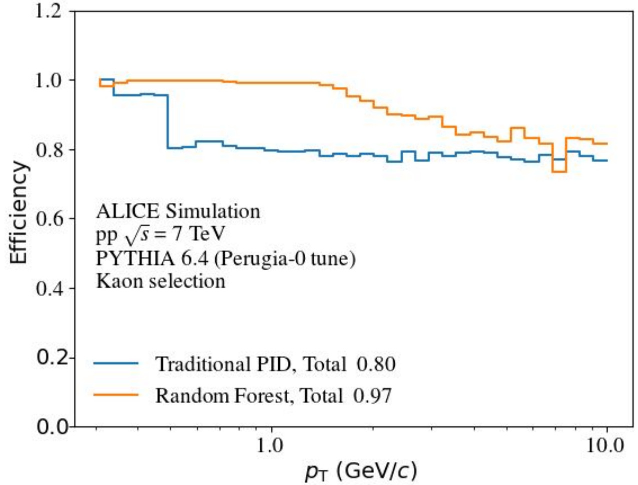
- pp at 7 TeV, Pythia 6 Perugia-0
- kaons vs other particles

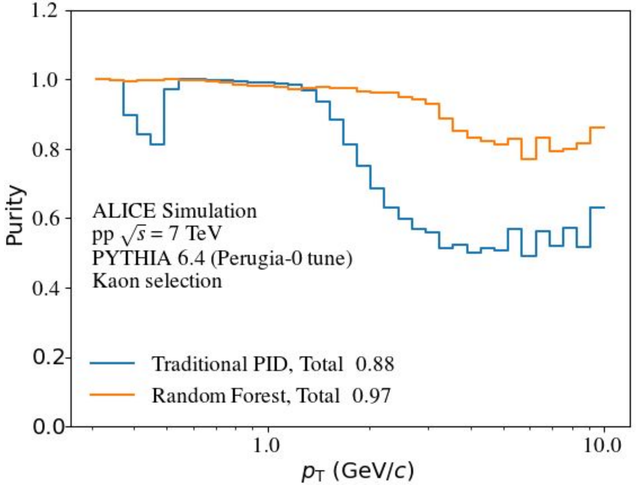
Traditional PID:

$$n^2_{\sigma,\mathrm{TPC}} \qquad p_\mathrm{T} \leq 0.5\mathrm{GeV}/c$$

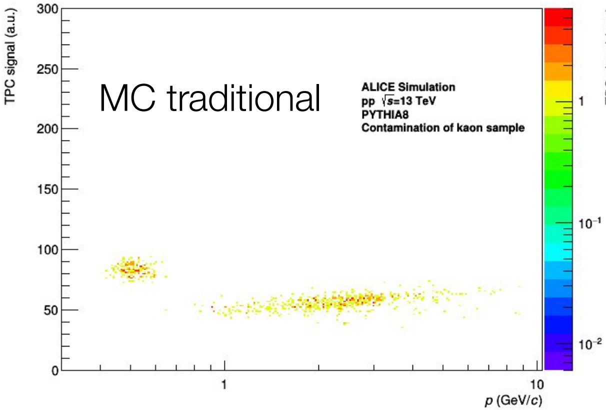
$$\sqrt{n^2_{\sigma,\mathrm{TPC}} + n^2_{\sigma,\mathrm{TOF}}} \qquad p_\mathrm{T} > 0.5\mathrm{GeV}/c$$

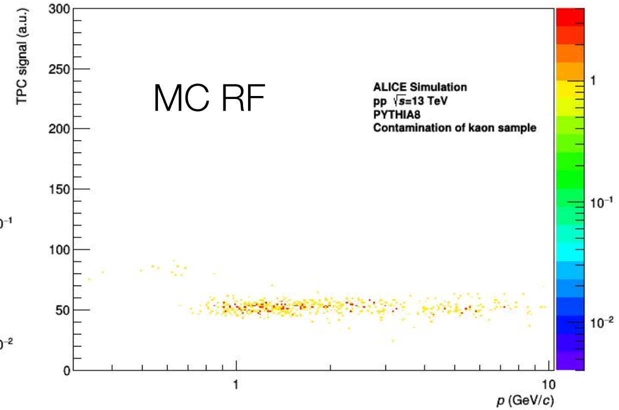
much higher efficiency and **purity** with Random Forest





Contamination of kaon samples





Results

$F_1 = 2 \times (\text{purity} \times \text{efficiency}) / (\text{purity} + \text{efficiency})$

best model, **2nd best model**

ML outperforms the standard way

FSE + attention with **very good scores** of $\mathbf{F_1}$

No flaws of other methods:

- imputation:
  artificial bias in data
- case deletion:
  no ability to analyze samples with missing detector signals
- NN ensemble:
  potentially large complexity

	π	p	K	$\bar{\pi}$	$\bar{p}$	$\bar{K}$
standard	87.87 ± 0.87	74.61 ± 1.88	73.17 ± 1.57	87.66 ± 0.87	69.12 ± 1.93	69.44 ± 1.60
NN ensemble	98.45 ± 0.04	95.42 ± 0.12	86.74 ± 0.16	98.27 ± 0.42	94.60 ± 0.10	84.91 ± 0.48
mean	98.40 ± 0.01	95.54 ± 0.06	86.36 ± 0.34	98.34 ± 0.01	94.75 ± 0.20	84.67 ± 0.38
attention + FSE	98.50 ± 0.02	95.79 ± 0.07	87.44 ± 0.14	98.44 ± 0.02	94.89 ± 0.14	86.00 ± 0.13
regression	98.40 ± 0.04	95.49 ± 0.15	86.22 ± 0.46	98.36 ± 0.03	94.57 ± 0.13	85.01 ± 0.13

	π, only complete data	p, only complete data	K, only complete data	$\bar{\pi}$, only complete data	$\bar{p}$, only complete data	$\bar{K}$, only complete data
case deletion	99.37 ± 0.01	99.43 ± 0.16	96.95 ± 0.06	99.37 ± 0.01	99.13 ± 0.26	96.33 ± 0.11
NN ensemble	99.38 ± 0.01	99.46 ± 0.13	97.23 ± 0.10	99.34 ± 0.18	99.33 ± 0.10	96.87 ± 0.09
mean	99.27 ± 0.04	99.47 ± 0.08	96.08 ± 0.36	99.27 ± 0.04	99.20 ± 0.27	95.45 ± 0.33
attention + FSE	99.36 ± 0.01	99.48 ± 0.02	97.04 ± 0.17	99.37 ± 0.03	99.44 ± 0.08	96.91 ± 0.11
regression	99.25 ± 0.07	99.37 ± 0.07	95.62 ± 0.39	99.28 ± 0.02	99.10 ± 0.13	95.11 ± 0.58

Example: FSE with one-hot encoding

M. Kasak, K. Deja. M. Karwowska,
M. Jakubowska, Ł. Graczykowski
M. Janik, EPJ C 84 (2024) 7, 691
M. Karwowska, Ł. Graczykowski, K. Deja,
M. Kasak, JINST 19 (2024) 07, C07013

Table 1: Preprocessing of data samples into feature set values – example.

(a) 3 data samples with 5 attributes with different amount of missing values.

id	momentum	TOF	TPC	TRD	ITS
1	0.1		3		5
2	7	70	24	13	88
3		78			

(b) First particle

key					value
1	0	0	0	0	0.1
0	0	1	0	0	3
0	0	0	0	1	5

(c) Second particle.

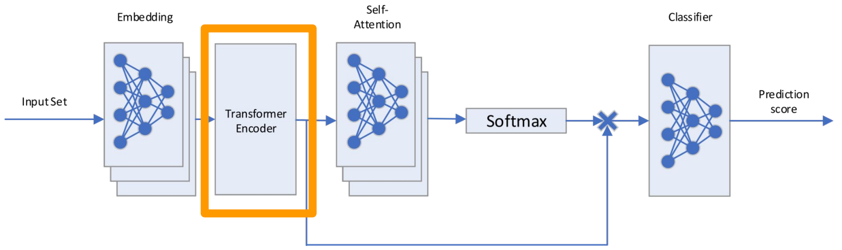
key					value
1	0	0	0	0	7
0	1	0	0	0	70
0	0	1	0	0	24
0	0	0	1	0	13
0	0	0	0	1	88

(d) Third particle.

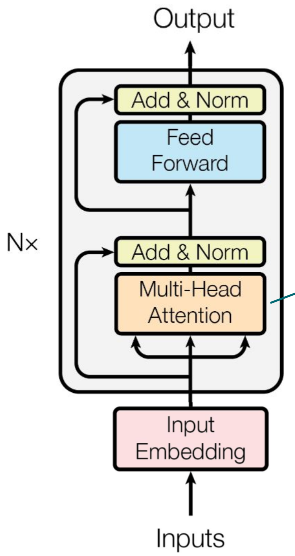
key					value
0	1	0	0	0	78

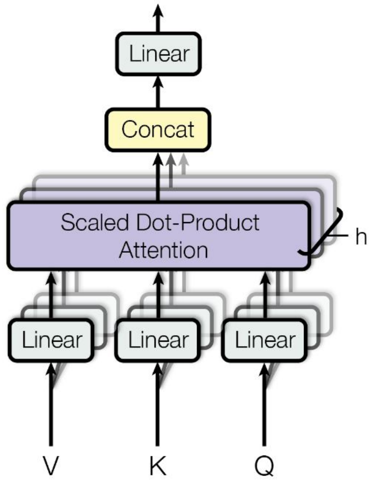
The attention continued

2. [Transformer Encoder]



N=2





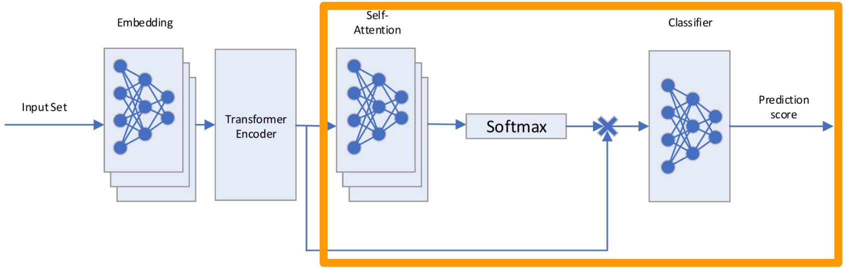
- adjusted original Transformer Encoder
- attention without convolutions and recurrence
- finding self-correlations in an instance set of vectors
- example: a specific detector signal could be used if and only if the momentum is in a specific range

$$Q, K, V \in \mathbf{R}^{n \times d_k}$$

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

modified diagram from the article

Pooling and classification



Classifier: a simple neural network
expects a single vector as an input

Solution: self-attention to pool the variable-size vector set from Transformer Encoder

$$\{v_1, v_2, ..., v_n\},\ v_i \in \mathbf{R}^{d_{model}}$$

$$e_i = NN(v_i) \qquad \forall i \in [1, n]$$ self-attention values

$$\alpha'_j = softmax(e'_j) \qquad \forall j \in [1, d_{model}]$$ self-attention weights

$$o_j = \sum_{k=1}^{n} \alpha_{kj} v_{kj} \qquad \forall j \in [1, d_{model}]$$ pooled output vector

Classifier score: logistic function $f(x) = \frac{1}{1+e^{-x}}$, range (0, 1)
"certainty" that a given particle belongs to the given type

Architecture of tested neural networks

Attention + FSE

- embedding layers: 19 – 128 – 32 neurons
- Transformer Encoder:
    - Multi-Head Attention: dimension 32, 2 heads
    - neural network layers: 32 – 128 – 32 neurons
    - 2 layers of Multi-Head Attention + neural network
- Self-Attention layers: 32 – 64 – 32 neurons
- classifier layers: 32 – 64 – 1 neurons
- dropout 0.1 at the output of embedding and each Transformer Encoder layer
- ReLU activation between neural network layers
- classifier loss function: binary cross entropy

Imputations, case deletion, and NN ensemble

- 3 hidden layers of sizes 64, 32, 16 with Leaky ReLU activation
- dropout 0.1 after each activation layer
- input size:
    - imputations and case deletion: 19 as all missing features are imputed
    - ensemble: 4 networks with input sizes 19, 17, 17, 15

Simple network implementation

PyTorch

- linear layers with ReLU, sigmoid at the end
- simple: dropout after each linear layer

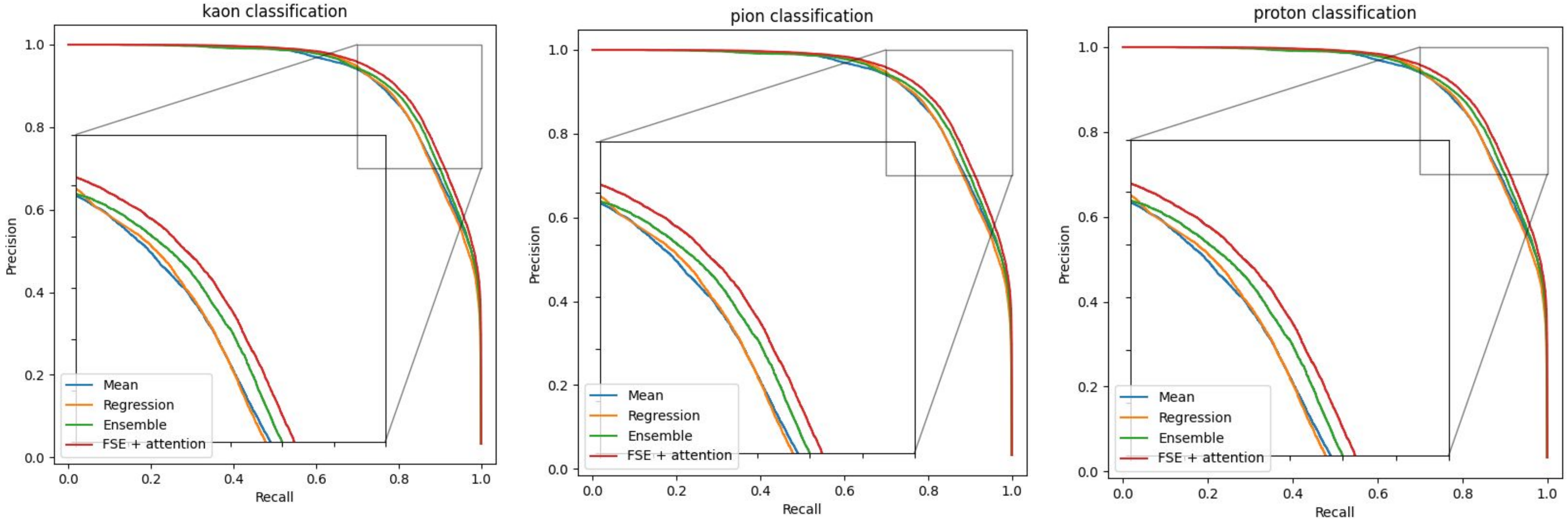
Parameters:

- optimizer: Adam
- output layer: 1 node (yes / no for a given particle)
- loss function: binary cross entropy
- scheduler: exponential with rate 0.98
- learning rate: 0.0005
- batch size: 64
- epochs: 30

Sample ROC curves

FSE+attention achieves **best results.**

Little variation between particle species.



More to go: domain adaptation

M. Kabus, M. Jakubowska,
Ł. Graczykowski, K. Deja
JINST 17, C07016. 2022

- **Monte Carlo never ideally matches the experimental data** (both physics and detector response simulation)

- **Problem:** transferring the knowledge from a **labeled source domain (MC data)** to **unlabeled target domain (experimental data)**, when both domains have different distributions of attributes

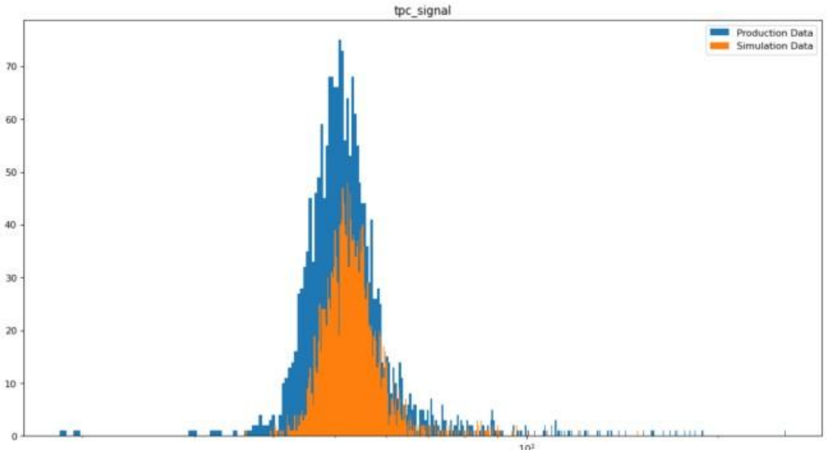
- How can we transfer the knowledge from training to inference?

Standard PID example: **"tune on data"**

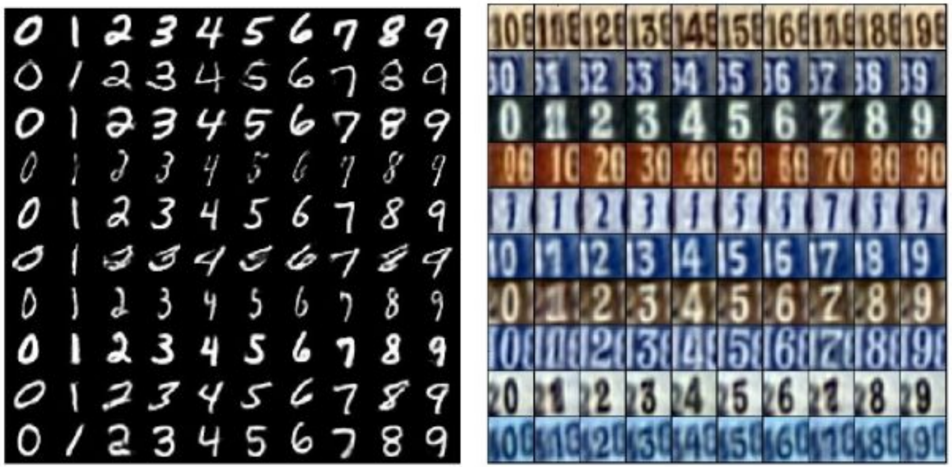
- get parametrization from data → real data
- generate a random detector signal → MC data
- equivalent distributions of real and MC samples – the differences are statistical fluctuations
- does not include correlations between attributes

Machine learning:

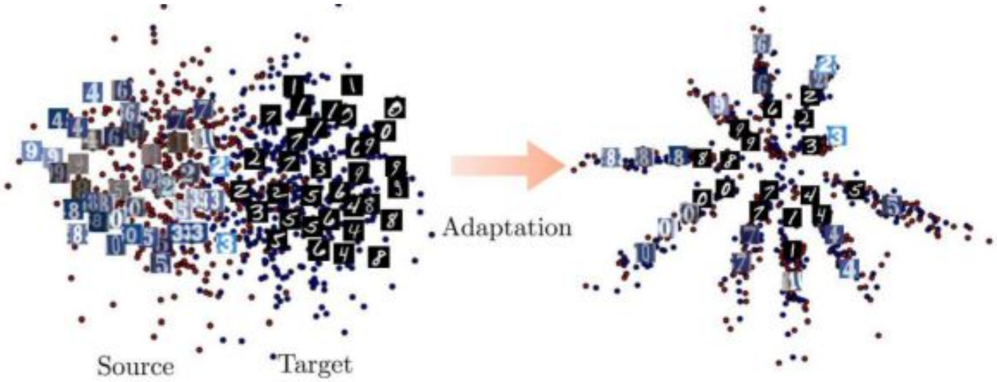
- actually **learn** the difference between data domains
- translate both data to a single common hyperspace



More to go: domain adaptation



(a) MNIST (b) SVHN



More to go: domain adaptation

M. Kabus, M. Jakubowska,
Ł. Graczykowski, K. Deja
JINST 17, C07016. 2022

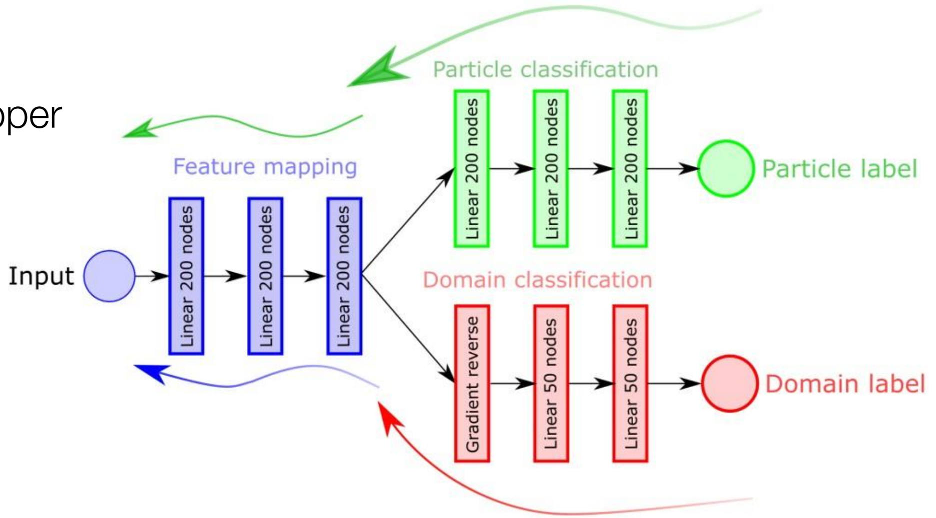
Feature mapping: input → domain invariant features

Particle classifier: recognize particles based on domain invariant latent space

Domain classifier: recognize MC vs real samples

Training more complicated:

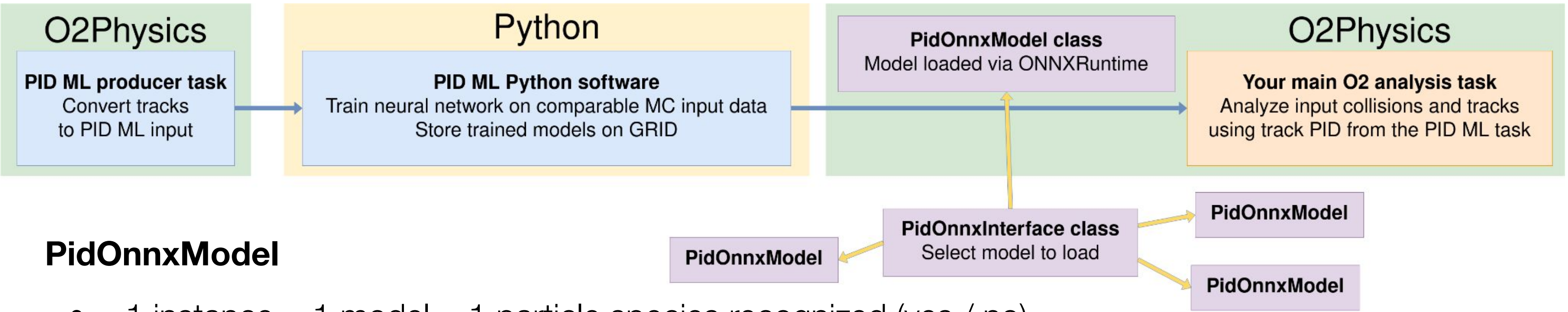
1. Train the domain classifier independently.
2. Freeze the domain classifier.
3. Train jointly particle classifier and feature mapper **adversarially** to the domain classifier.
4. Weights of the feature mapper:
   gradient from particle classifier
   + reversed gradient from domain classifier



Application time similar to a standard classifier

Our current solution still misses this step

Integration with O^2: user interface



PidOnnxModel

- 1 instance = 1 model = 1 particle species recognized (yes / no)
- **convenient interface** clearly separated from the rest of analysis
- using all capabilities of **Python ML libraries** for training
- ONNX file format and **ONNXRuntime** software used for inference in O^2 C++ environment

PidOnnxInterface

- **automatically select most suitable model** for user needs or manual mode
- as **little additional knowledge** from the analyser as possible (*"change 1 line in the code"*)

https://onnx.ai/