Tracking with Hashing

20th November 2024

Jeremy Couthures

Interpretability

- **Goal: Understand the model with physics**
  - Ideal: from black box (ML) to algorithm (physics)
- ***How* is the prediction done?:**
  - What are the steps taken?
- **Need to know *What* it predicts:**
  - Objective (loss function): group hits of **same particle**
    - But not necessarily what is done (poorly trained / untrained vs trained)
  - Performance plots: How good are the predictions with respect to the objective
  - Constraints: Hit by hit application → no curvature (q, pT) information

Interpretability

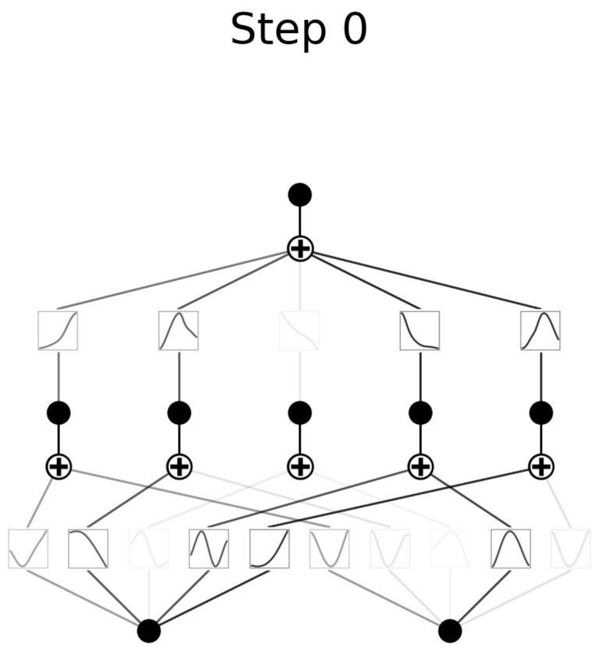
- **Extracting information:**
  - Assume the model is building an **algorithm** internally: ***mechanistic interpretability***
- **Approach:**
  - identify parts of this algorithm (relevant pieces)
  - identify known high-level features built internally

Identifying parts of the algorithm

- **Approach:**
  - Interpret relevant neurons as formulas
- **Steps:**
  1) Identify relevant neurons
  2) Symbolic regression to obtain a formula of the quantity approximated
  3) Identify relevant parts of the equation
  4) Compare with known physics high-level variables

KAN

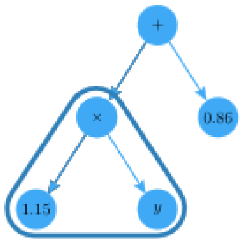
- Training Didn’t converge
- Didn’t improved after first batch
- Playing with hyper-parameters didn’t helped

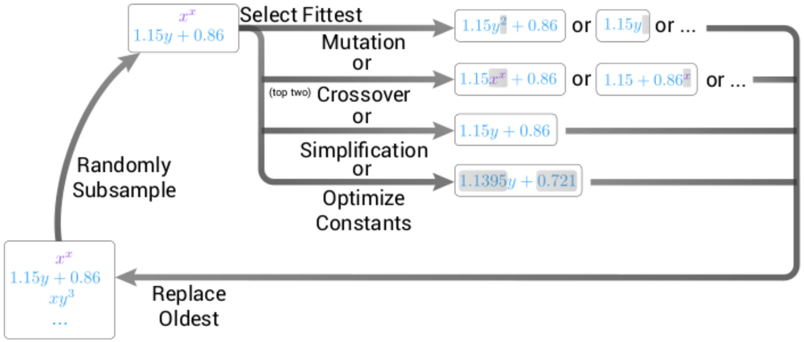


Symbolic regression

PySR

$1.15y + 0.86$





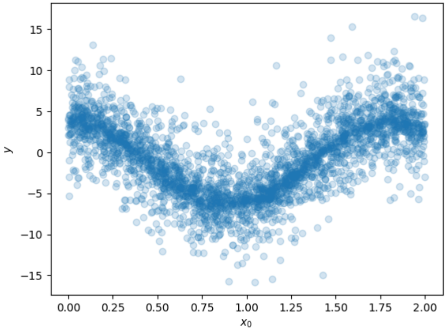
Genetic Algorithm

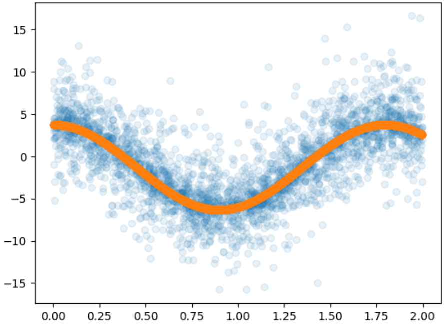
$$\sigma \sim U(0.1, 5.0)$$
$$\epsilon \sim N(0, \sigma^2)$$
$$y = 5\ \cos(3.5x_0) - 1.3 + \epsilon.$$

Truth

$$5.0337477\cos(3.496164x_0) - 1.29099218487498$$

Learned

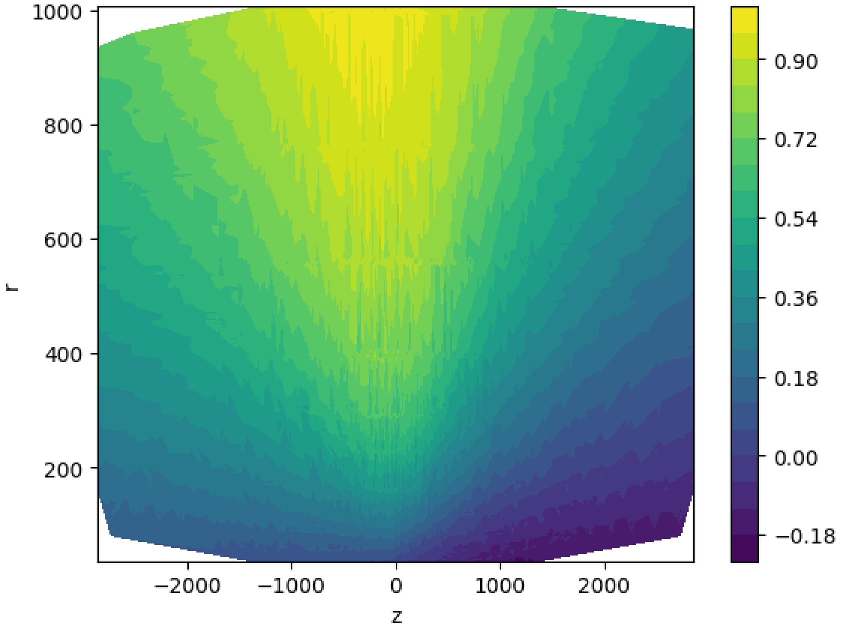




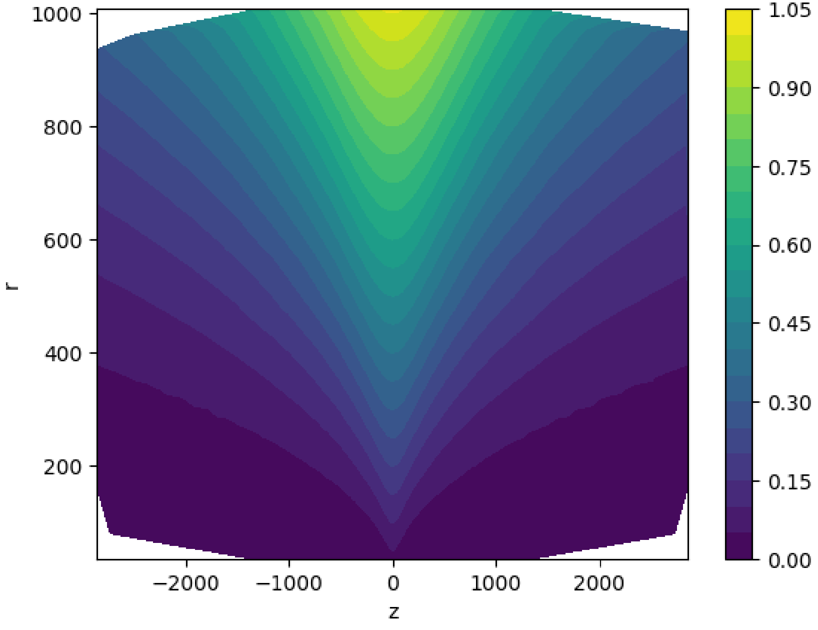
Neural Nets + Symbolic Regression

https://github.com/MilesCranmer/PySR

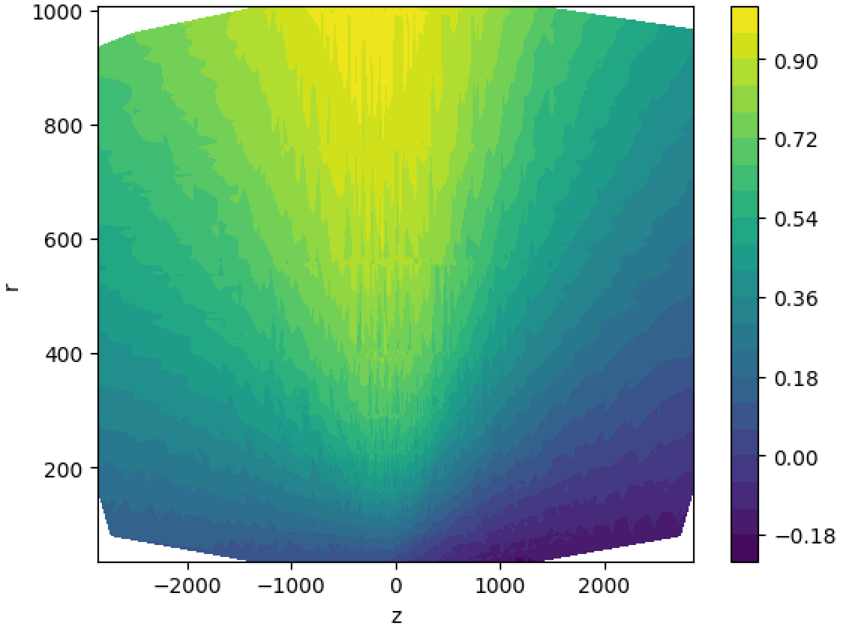
Symbolic regression



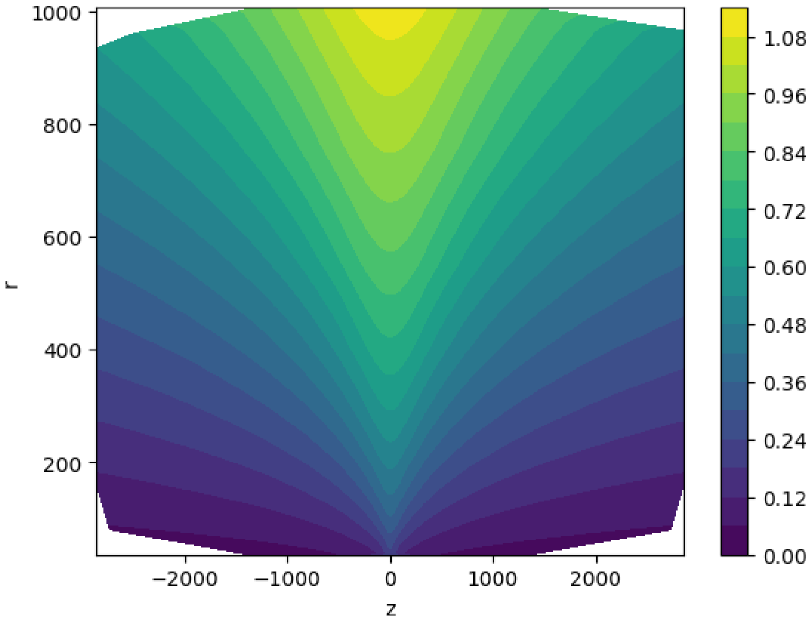
np.sin(df["theta"])*df["r"]/1000



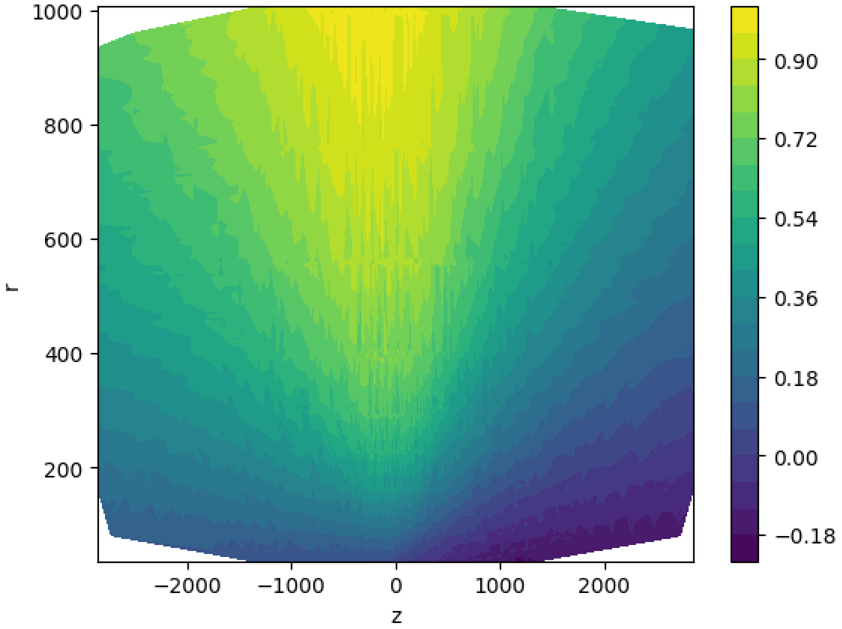
Symbolic regression



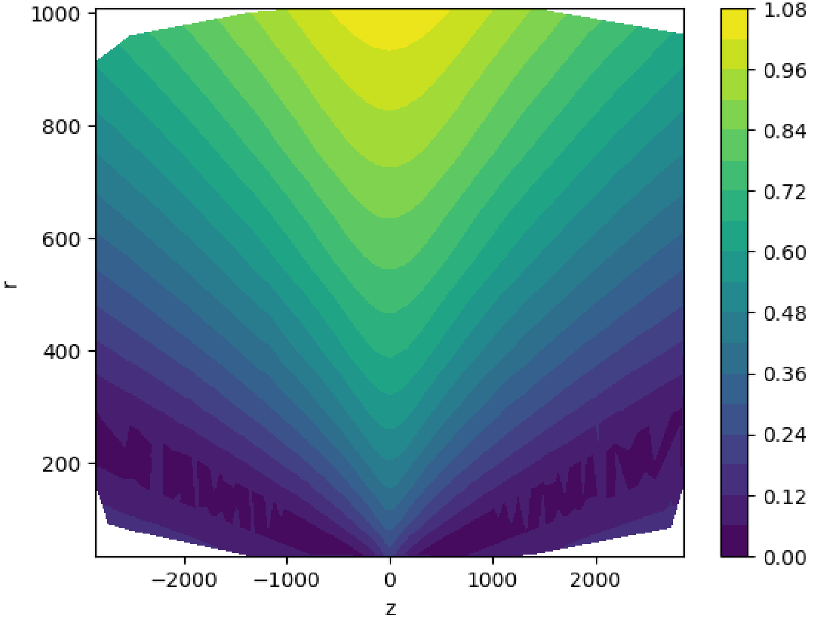
np.sin(df["theta"]) * 0.03505834 * np.sqrt(df["rho"])



Symbolic regression



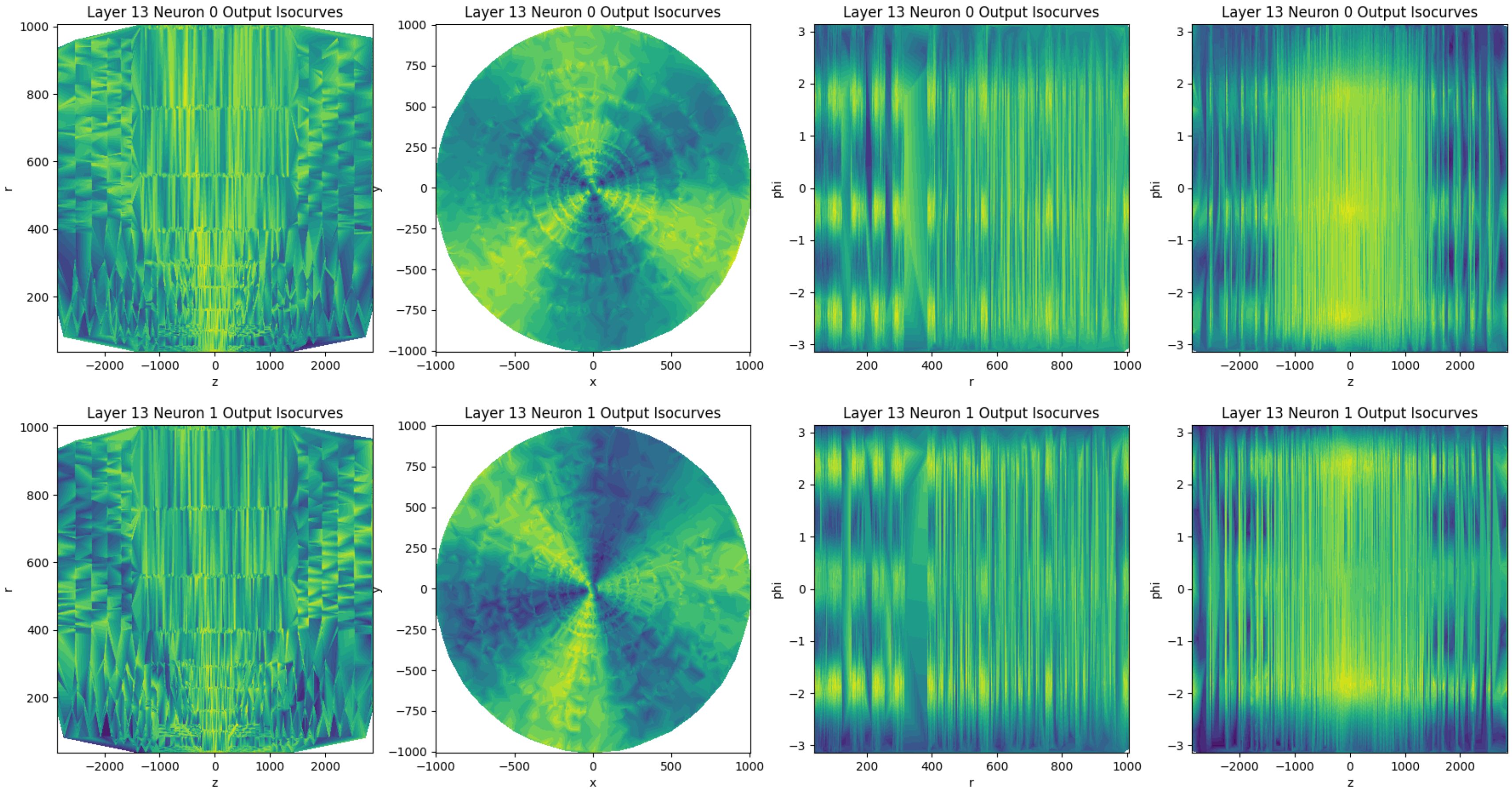
np.sqrt(0.0007077842 * (1 + np.cos(df["eta"])) * df["r"])



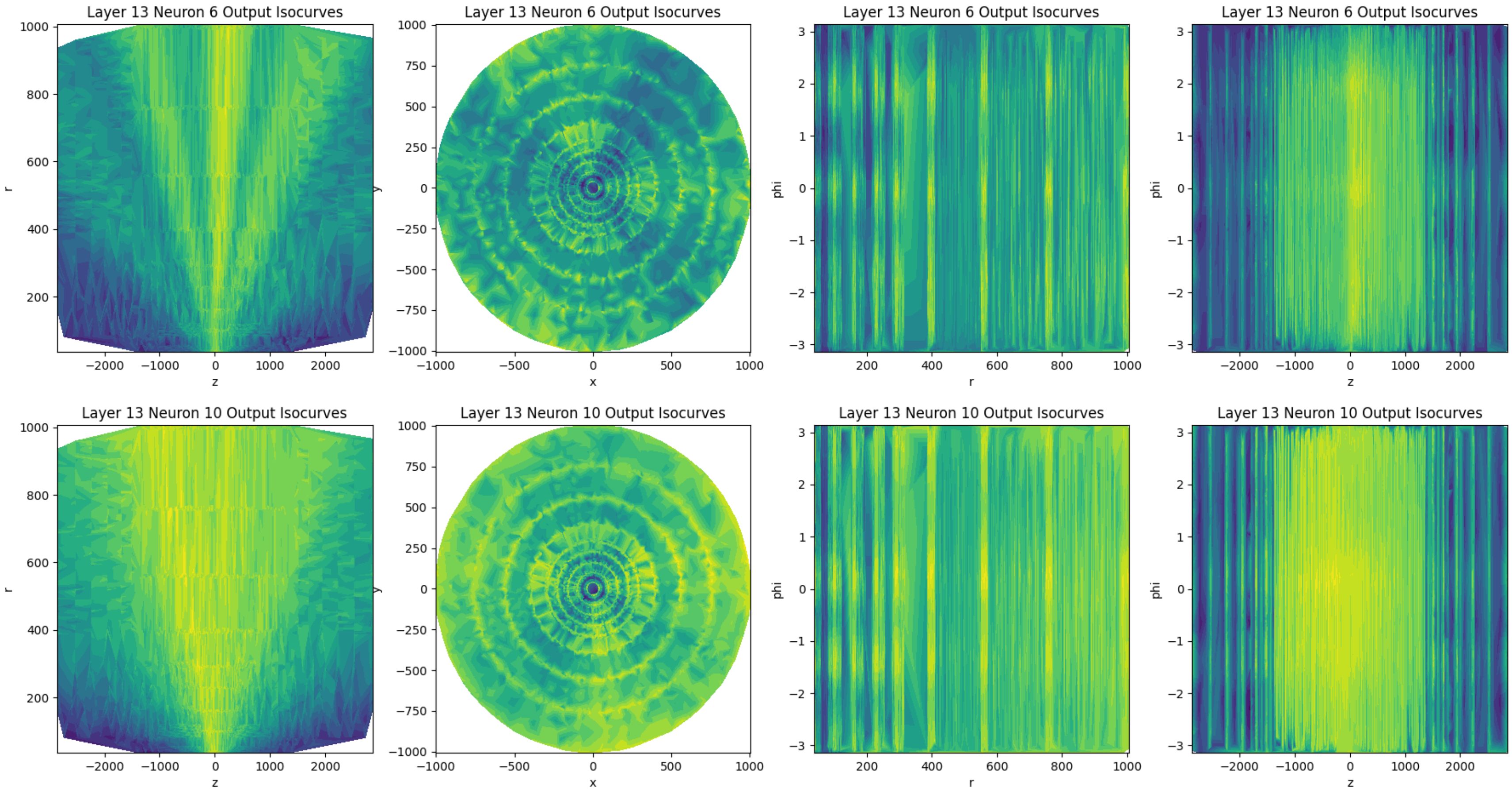
Interpretability

- ***How* is the prediction done?:**
  - What are the steps taken?
  - Does it predict track features (q/pT, eta, phi, d0, z0)?

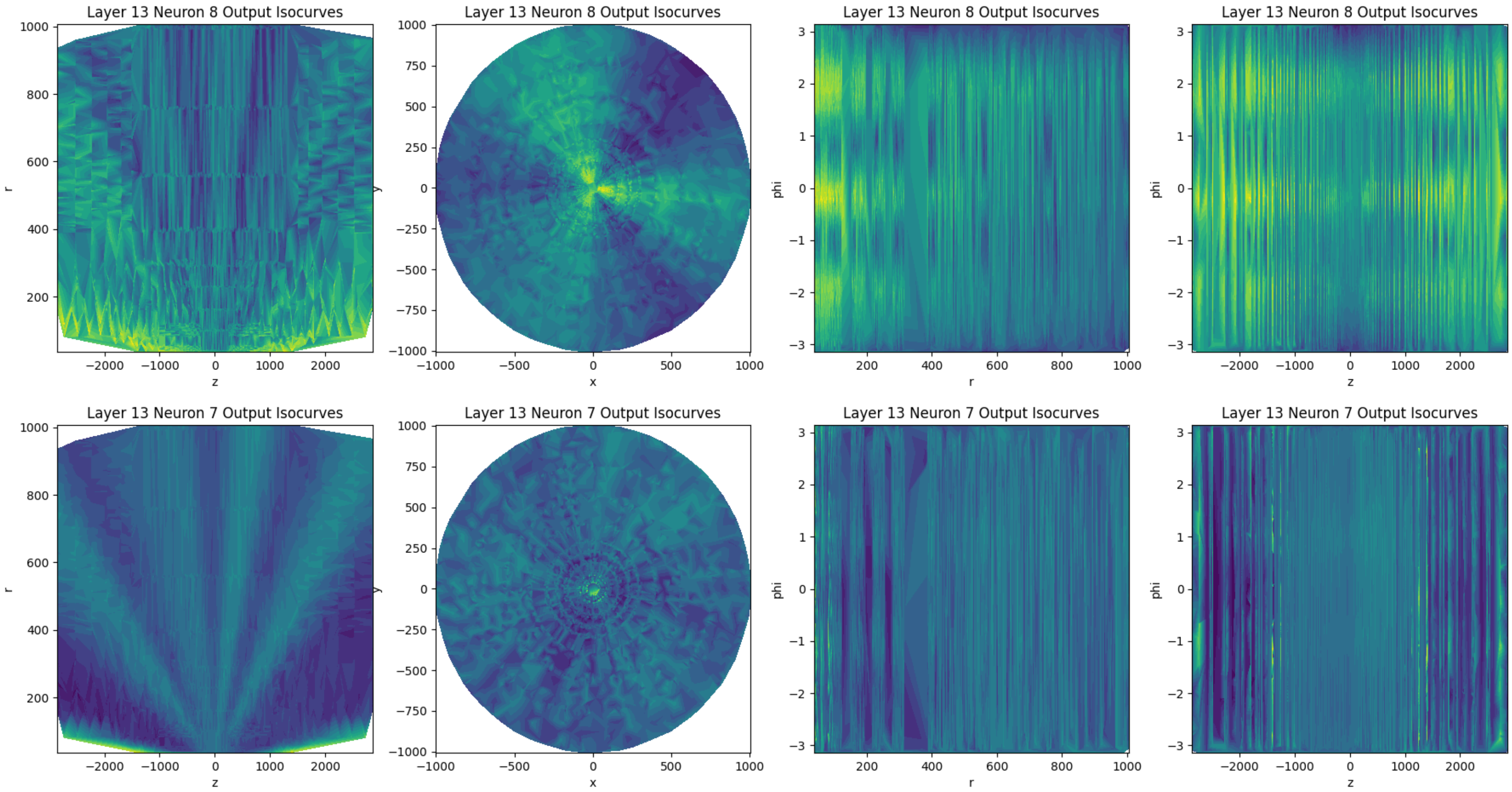
Latent space



Latent space



Latent space



Known high-level features

- **Assumption:**

  the model is using high-level features in the output latent space (12 neurons)

- **Approach:**
  - Information theory: conditional entropy of high-level features conditioned on the output latent space → gives how much of the high-level feature can be predicted from the latent space alone

Entropy

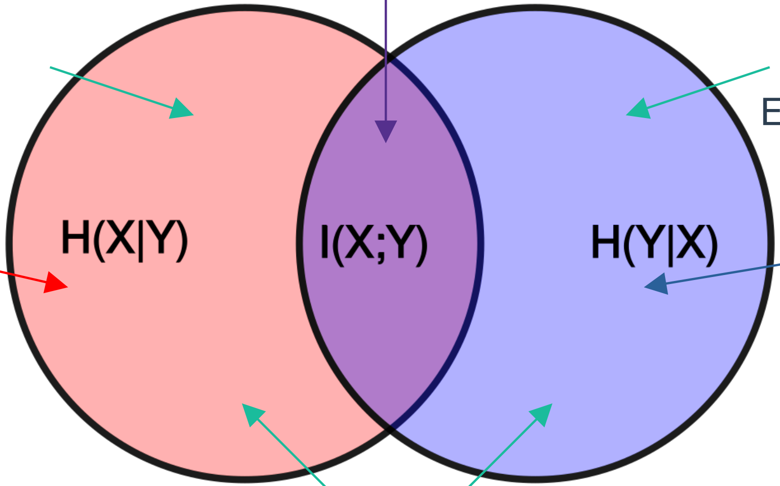
Mutual Information:
H(X) ∩ H(Y)

≈ Y = f(X)
X = g(Y)

H(X)

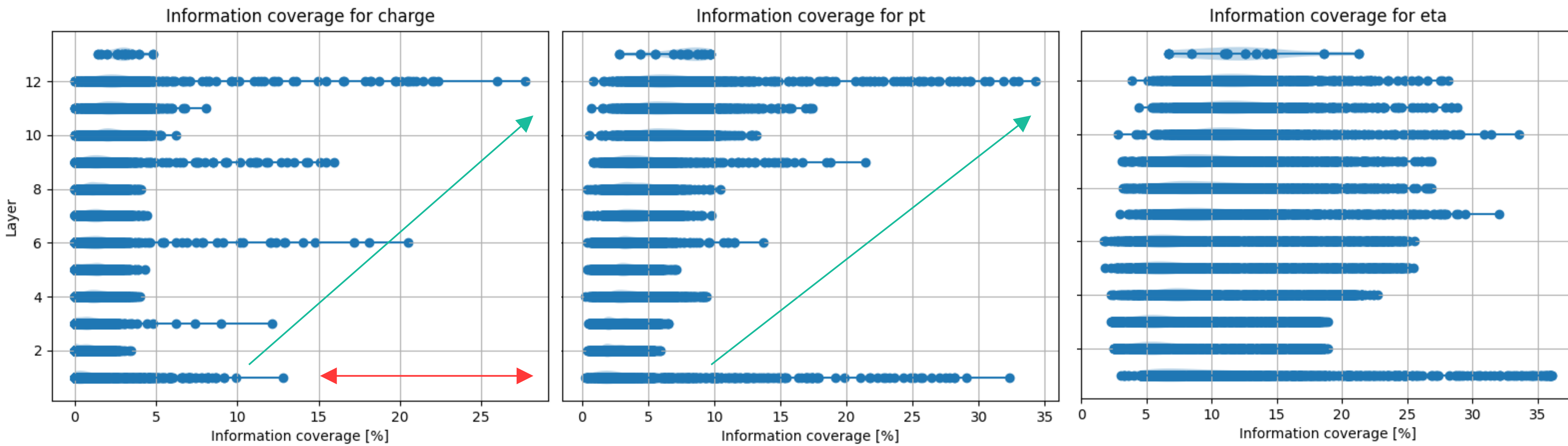
H(Y)
Entropy

Conditional entropy H(X|Y):
H(X) \ H(Y)

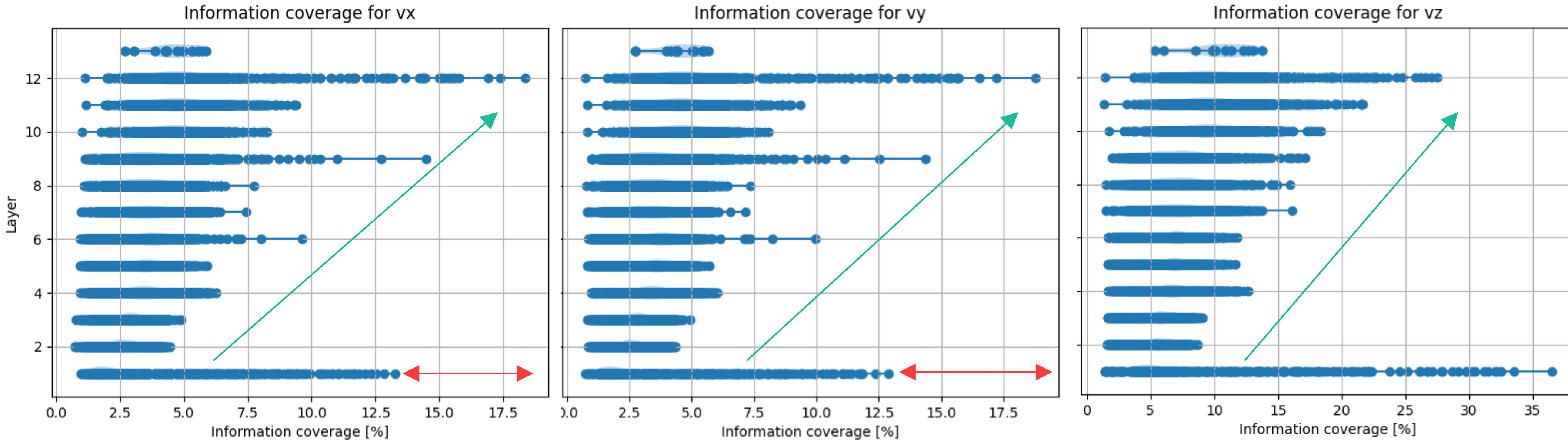


Conditional entropy H(Y|X):
H(Y) \ H(X)

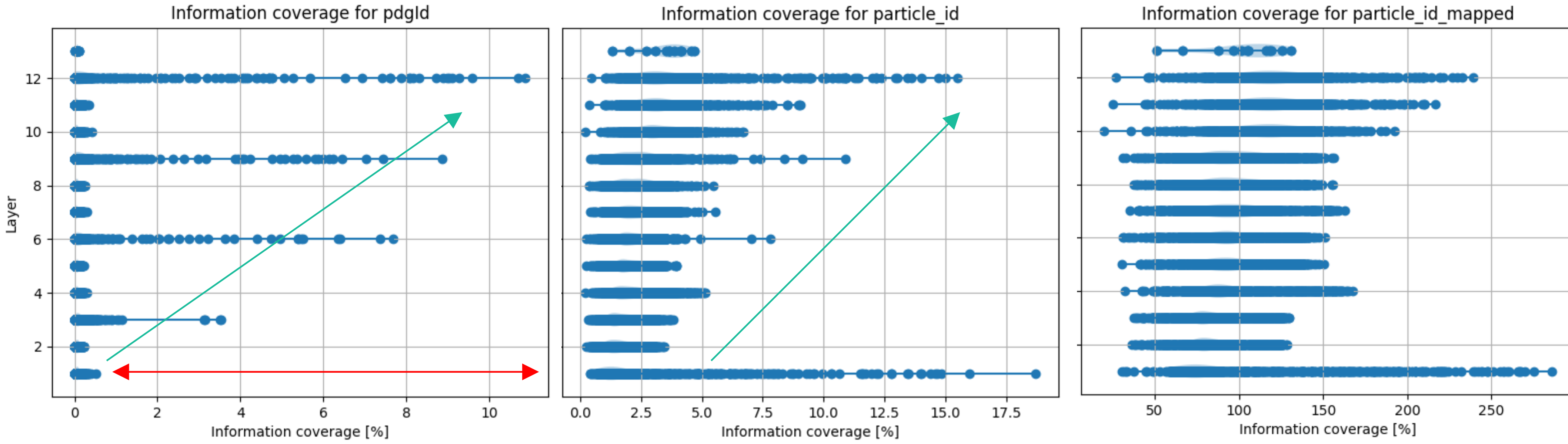
H(X,Y)
Joint entropy:
H(X) ∪ H(Y)



Information coverage for charge
Information coverage for pt
Information coverage for eta
Layer
Information coverage [%]

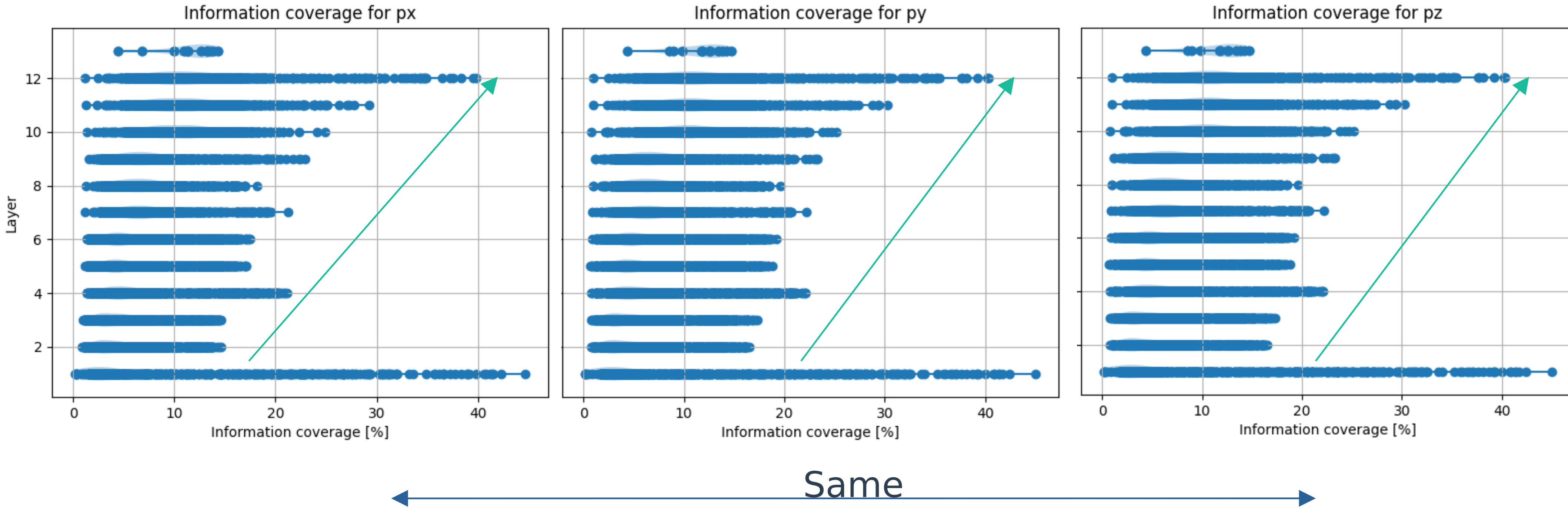

Information coverage for vx
Information coverage for vy
Information coverage for vz
Layer
Information coverage [%]
Information coverage [%]
Information coverage [%]

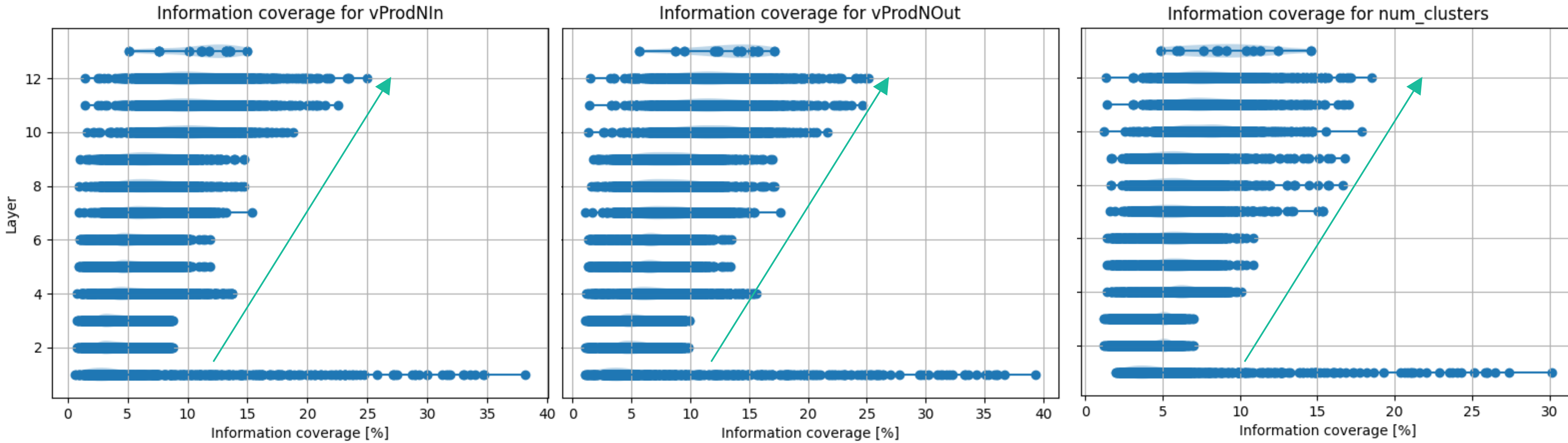
Same



Learned to isolate some particles

Broken


Information coverage for px
Information coverage for py
Information coverage for pz
Layer
12
10
8
6
4
2
0
10
20
30
40
Information coverage [%]
Same


Information coverage for vProdNIn
Information coverage for vProdNOut
Information coverage for num_clusters
Layer
Information coverage [%]
Information coverage [%]
Information coverage [%]
0
5
10
15
20
25
30
35
40
2
4
6
8
10
12

BACKUP

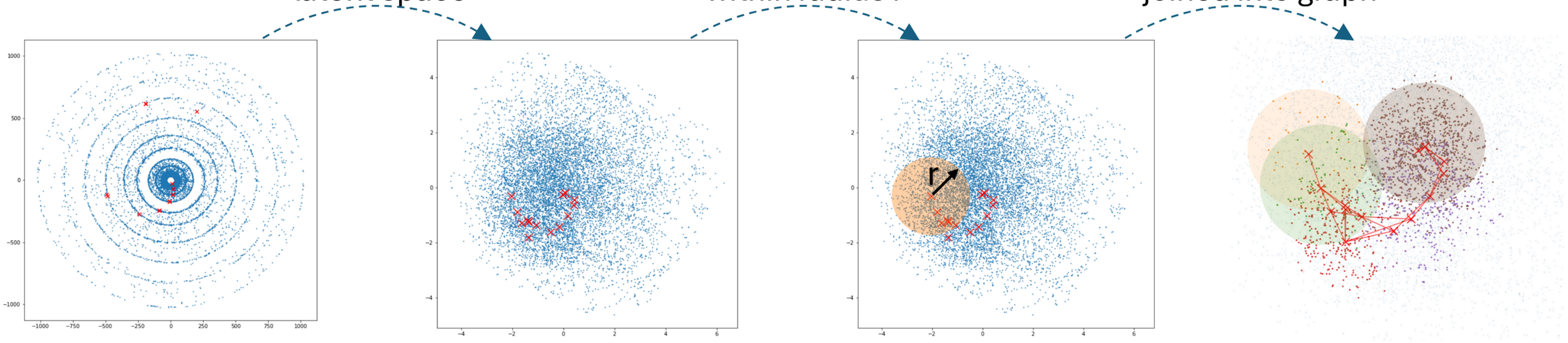
- **Plots en fonction de r-phi-z = pas la bonne approche**
- **Cherche à faciliter la reconstruction des traces → ce que les points d’une même trace ont en commun → doit regarder en fonction des paramètres des traces et comparer pour différentes traces**

GNN Metric Learning

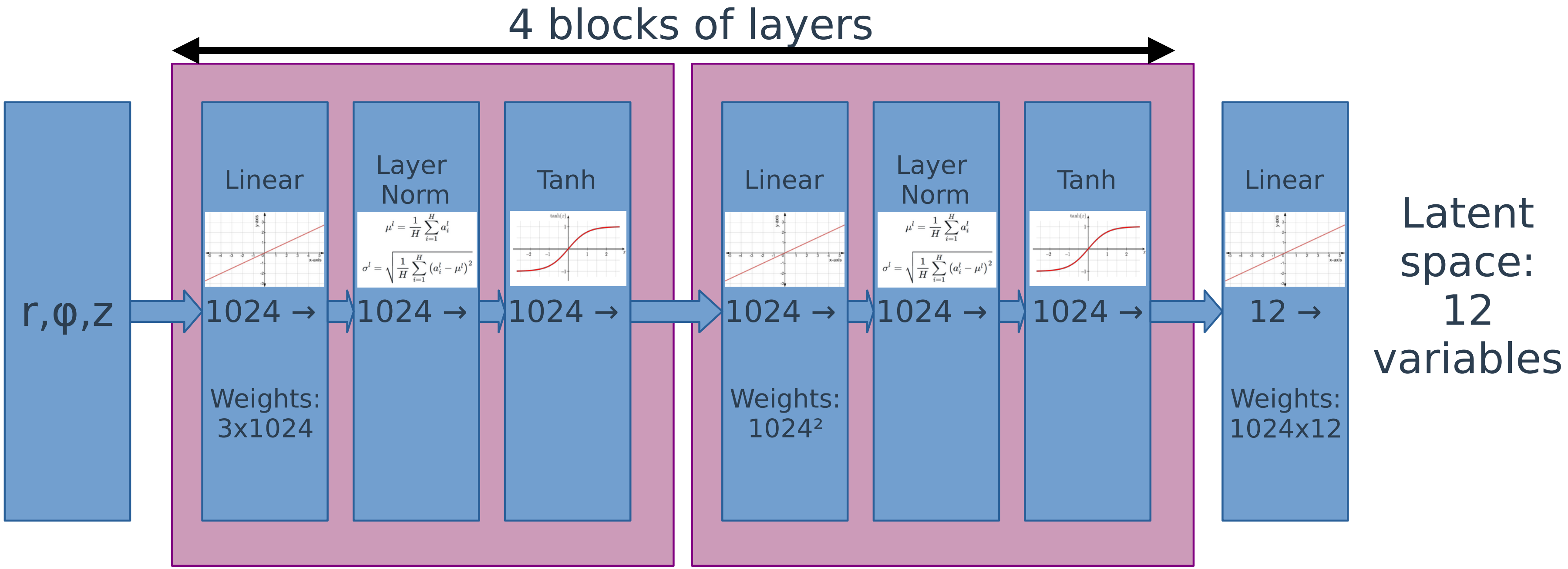
Embed into learned latent space

Connect all space points within radius r

All space point pairs joined into graph

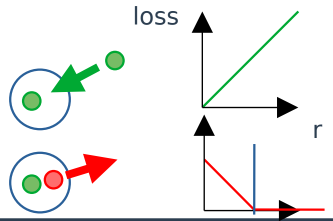


Architecture



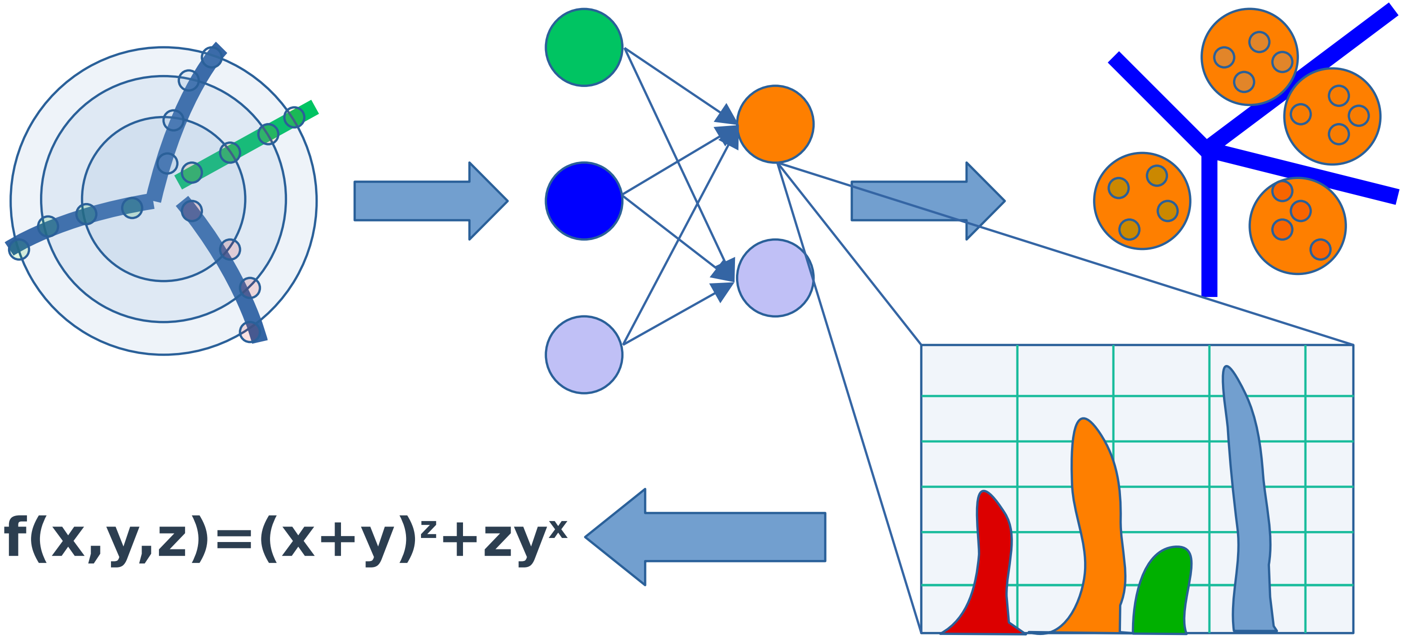
Hinge Loss

$$l_n = \begin{cases} x_n, & \text{if } y_n = 1, \\ \max\{0, margin - x_n\}, & \text{if } y_n = -1, \end{cases}$$



Interpretability

- study the internal representation of the problem by the model



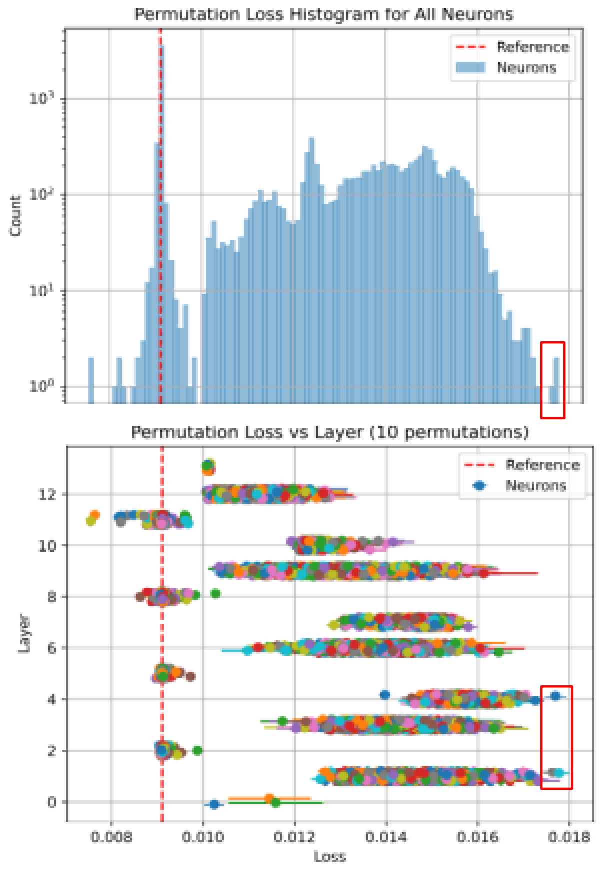
$$f(x,y,z)=(x+y)^z+zy^x$$

Software implementation

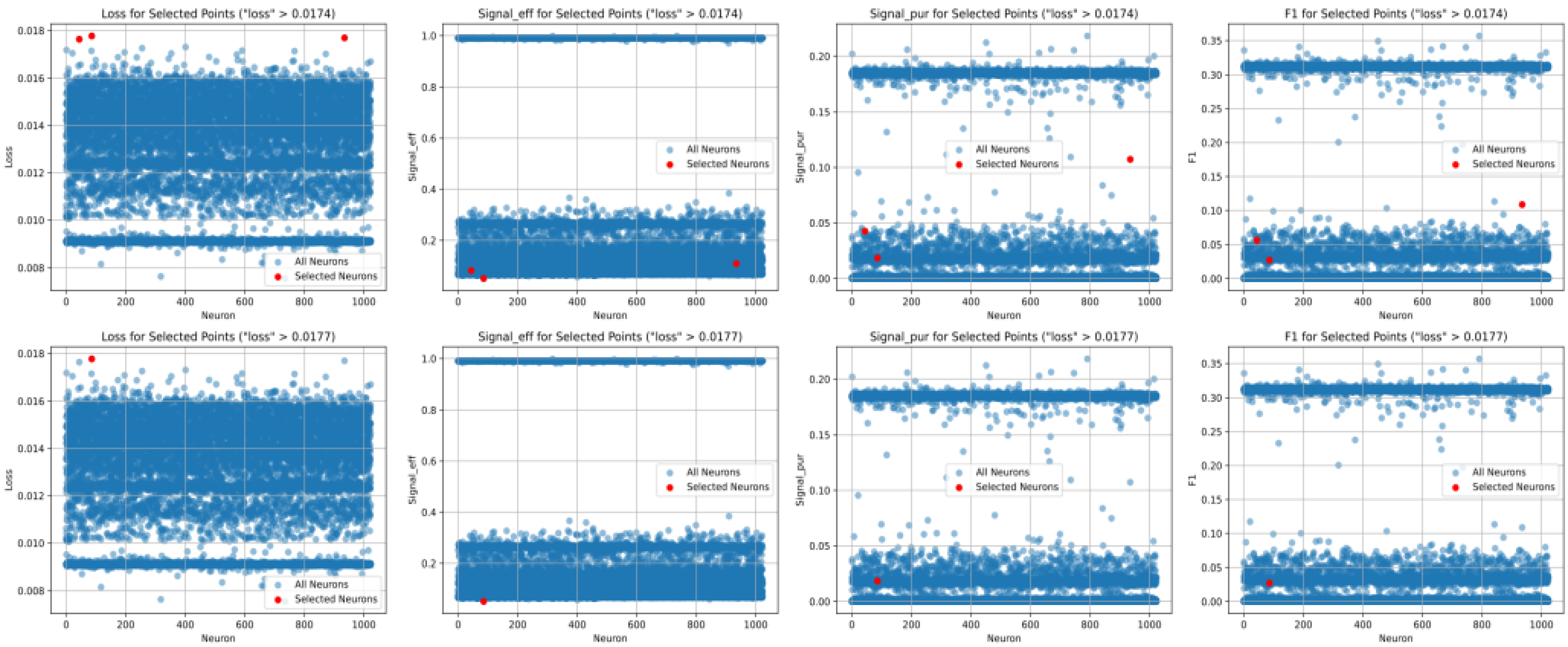
Identify High Level variables

Neuron identification: Permutation loss

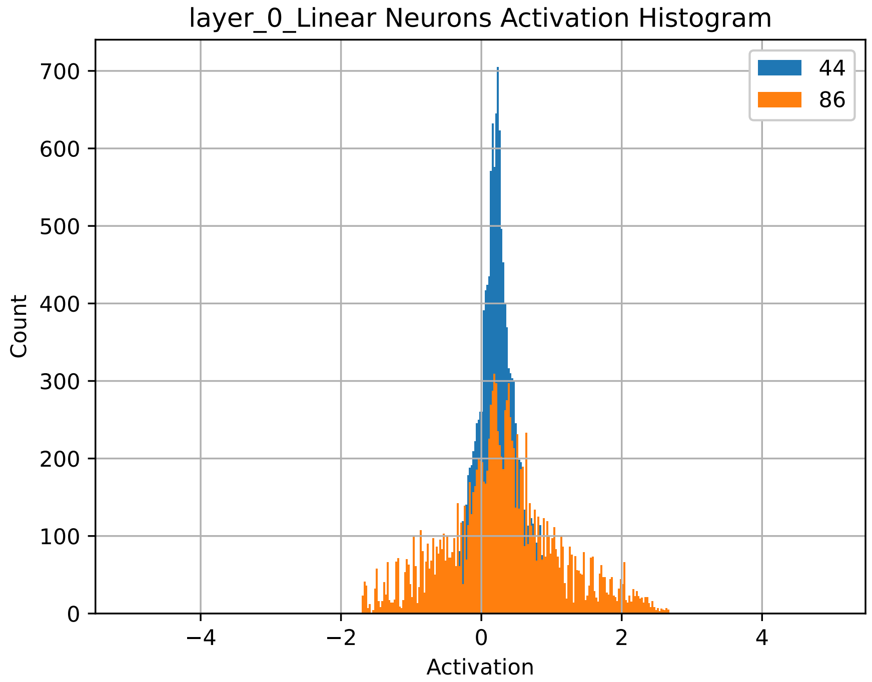
- **3 promising neurons:**
  - 2 on layer 1 (***Linear*** with input layer)
  - 1 on layer 4 (More complex)
- **Normalization Layers (3n-1) not perturbed by permutation → Information is shared among neurons**

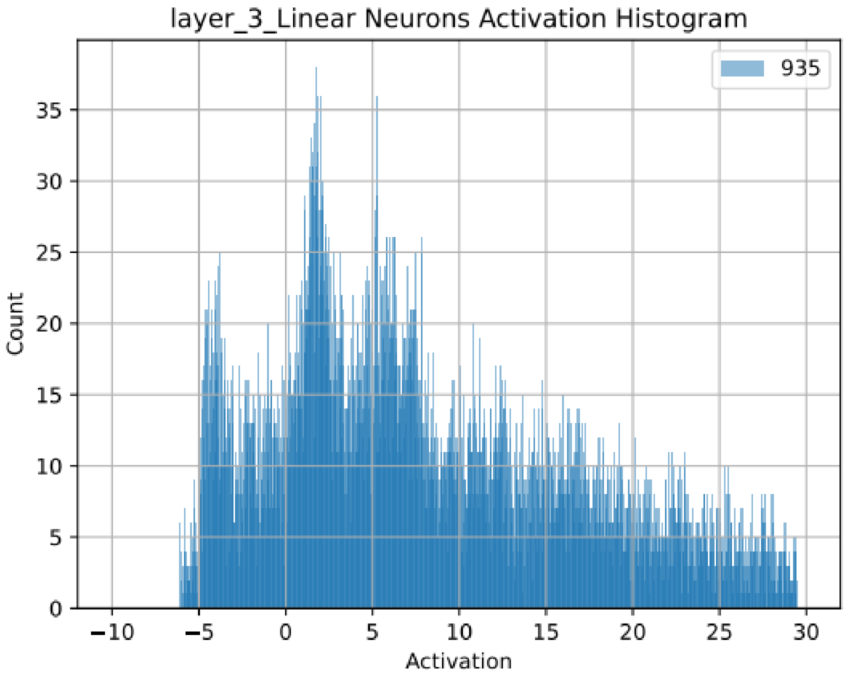


Neuron specificities

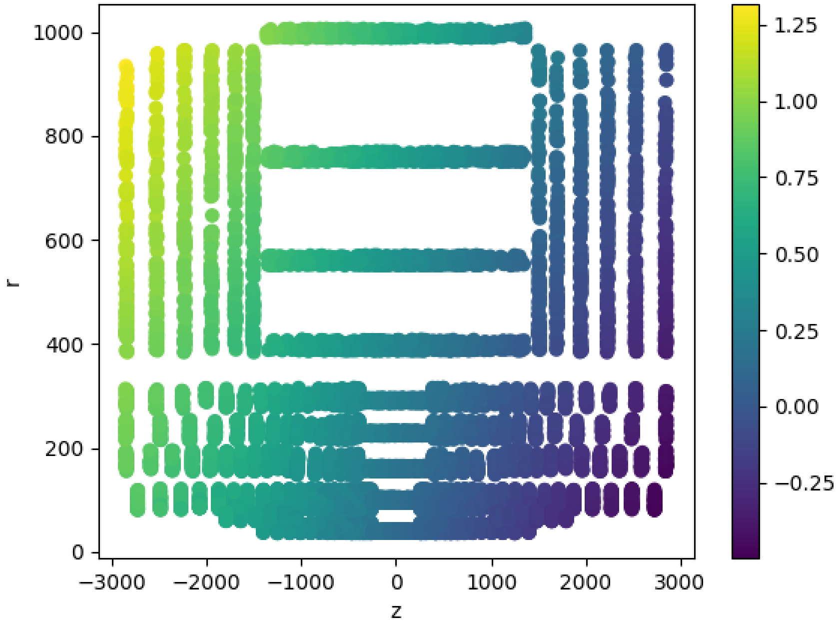


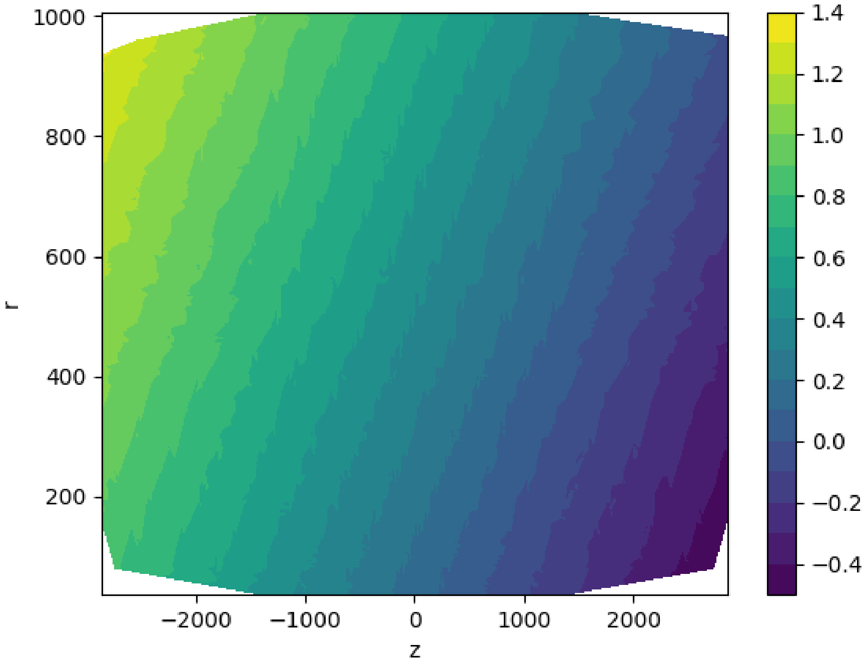
Activations



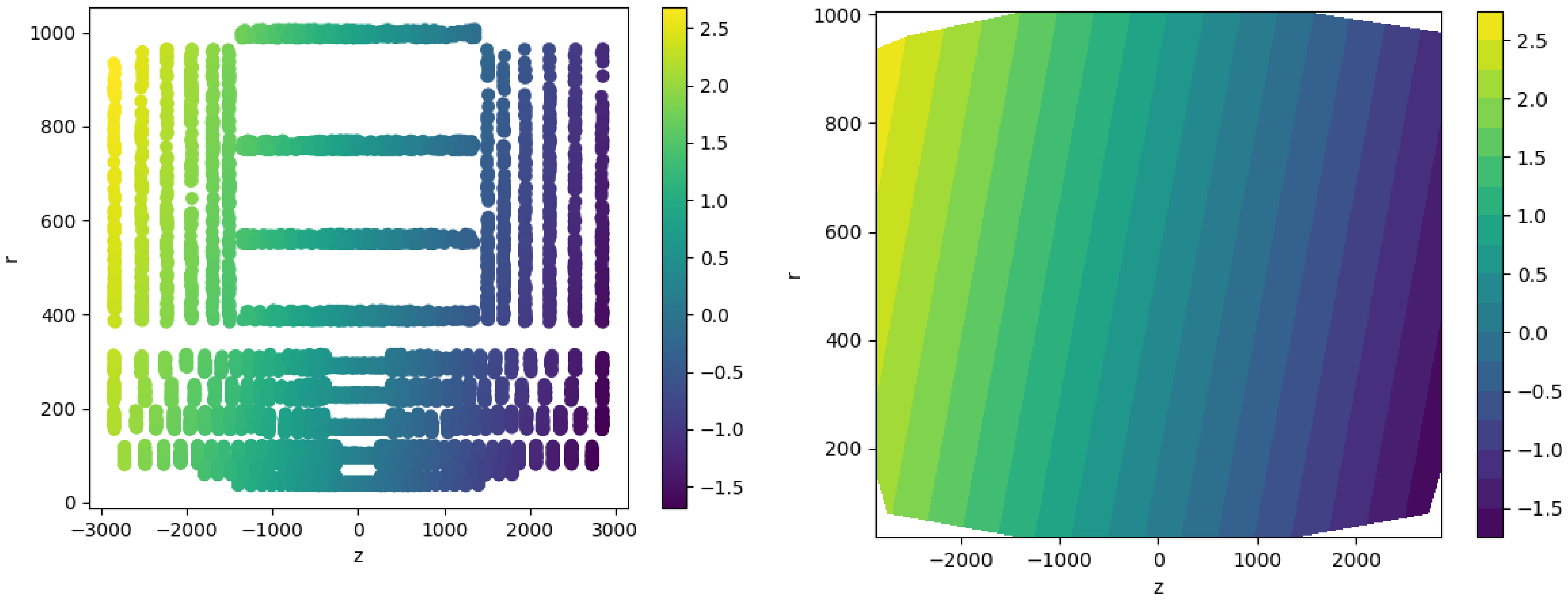


Activations r-z neuron 44



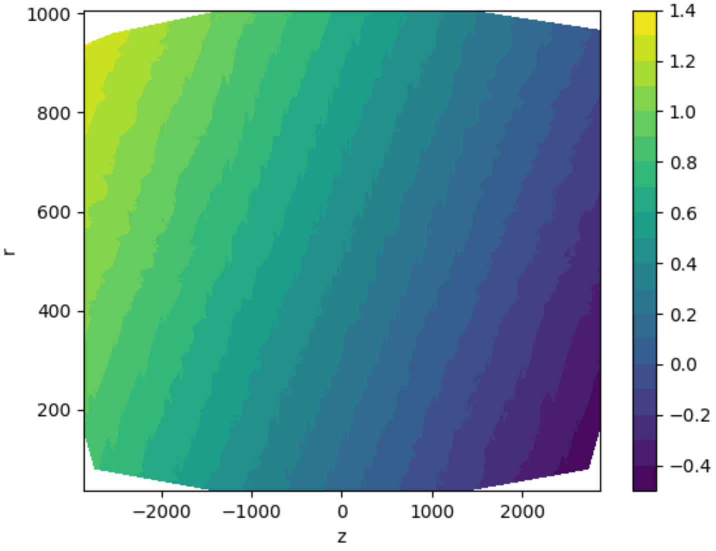


Activations r-z neuron 86

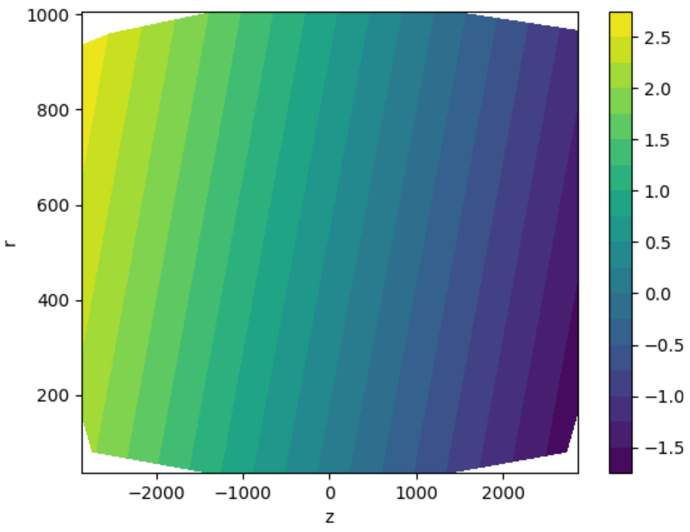


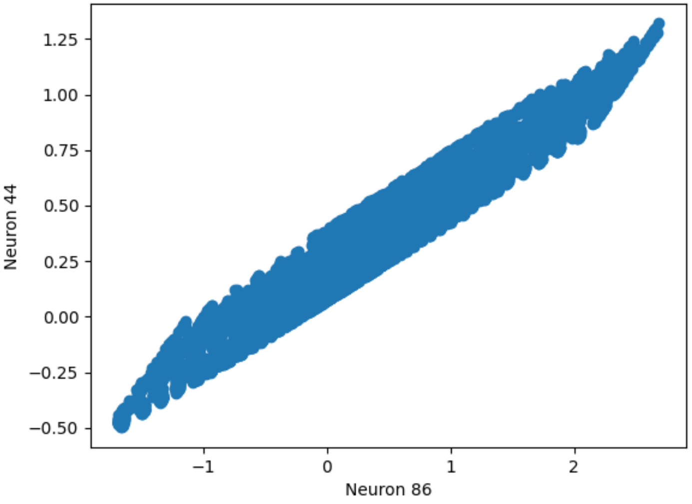
Neuron 44 vs neuron 86

Neuron 44

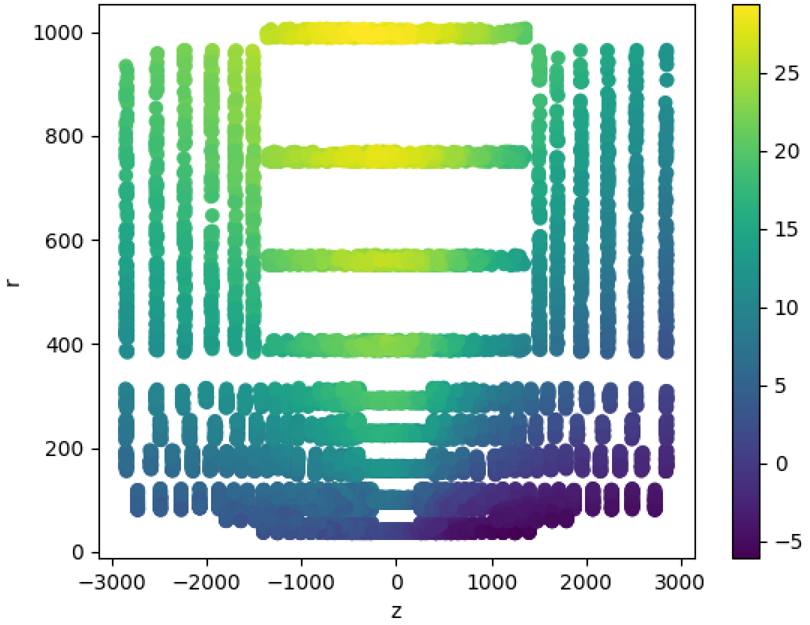


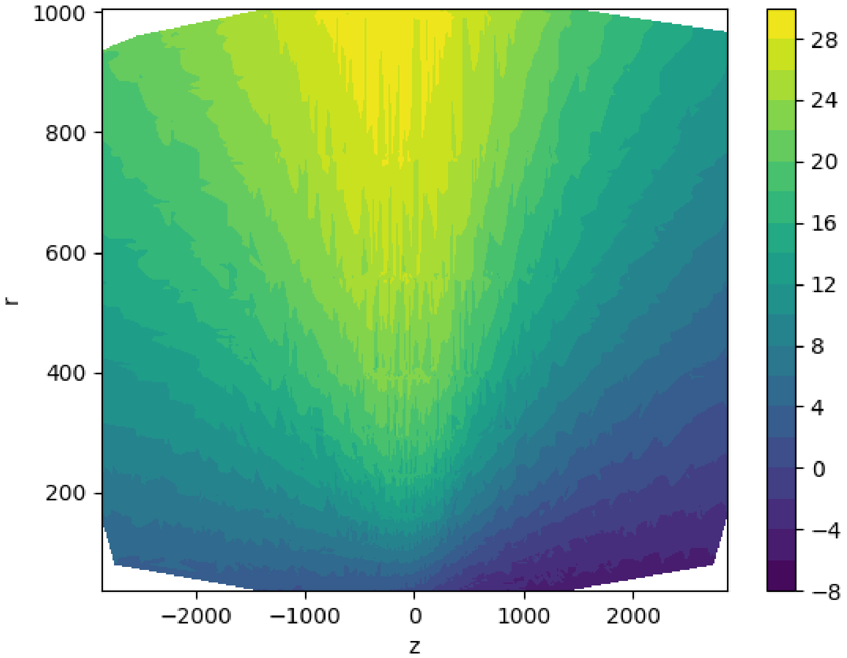
Neuron 86





Activations r-z neuron 935

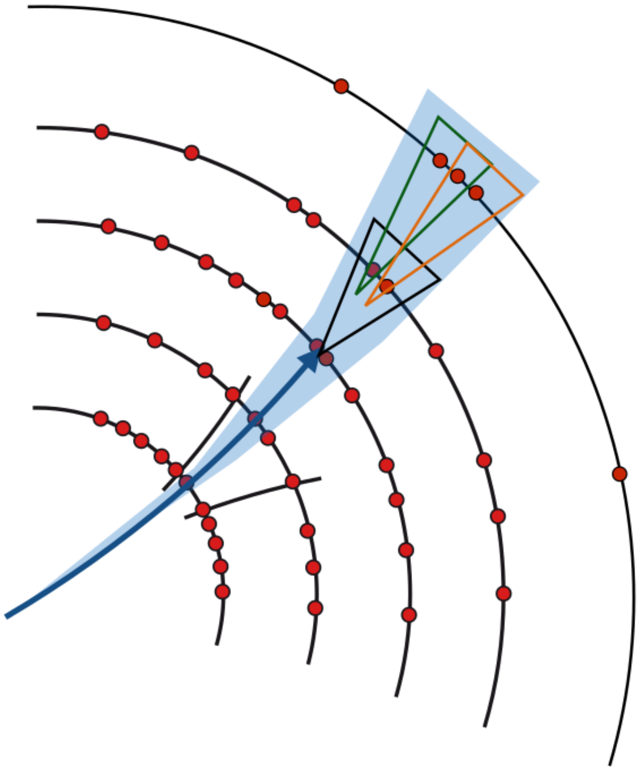




Combinatorial problem

Combinatorial Kalman Filter:

- Several possibilities of expanding the seeds at each layer → need to test them all
- Number of combinations increases exponentially with the number of layers



Model

- **Example 2**

1. First, we build our input data from the raw Athena events:

```
acorn infer data_reader.yaml
```

2. We start the graph construction by training the Metric Learning stage:

```
acorn train metric_learning_train.yaml
```

3. Then, we build graphs using the Metric Learning in inference:

```
acorn infer metric_learning_infer.yaml
```

```
# Model inference parameters
r_infer: 0.1
knn_infer: 1000
```

```
hard_cuts:
  pt: [1000, .inf]

# Model parameters
undirected: True
node_features: [r,    phi,   z]
node_scales:   [1000, 3.14, 1000]
emb_hidden: 1024
nb_layer: 4
emb_dim: 12
activation: Tanh
randomisation: 1
points_per_batch: 50000
r_train: 0.1
knn: 50
knn_val: 1000

# Training parameters
warmup: 5
margin: 0.1
lr: 0.01
factor: 0.7
patience: 10
max_epochs: 100
metric_to_monitor: f1
metric_mode: max
```

Performance