Energy Reconstruction with CALIFA

Tobias Jenegger

CALIFA Calorimeter

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation)
under Germany's Excellence Strategy – EXC-2094 – 390783311,
BMBF 05P19WOFN1, 05P21WOFN1
and the FAIR Phase-0 program

TUM Members:
Roman Gernhäuser, Lukas Ponnath, Philipp Klenze, Tobias Jenegger

CALIFA Detector @ R^3B

CALorimeter for the **I**n **F**light detection of γ-rays and light charged p**A**rticles

Endcap:

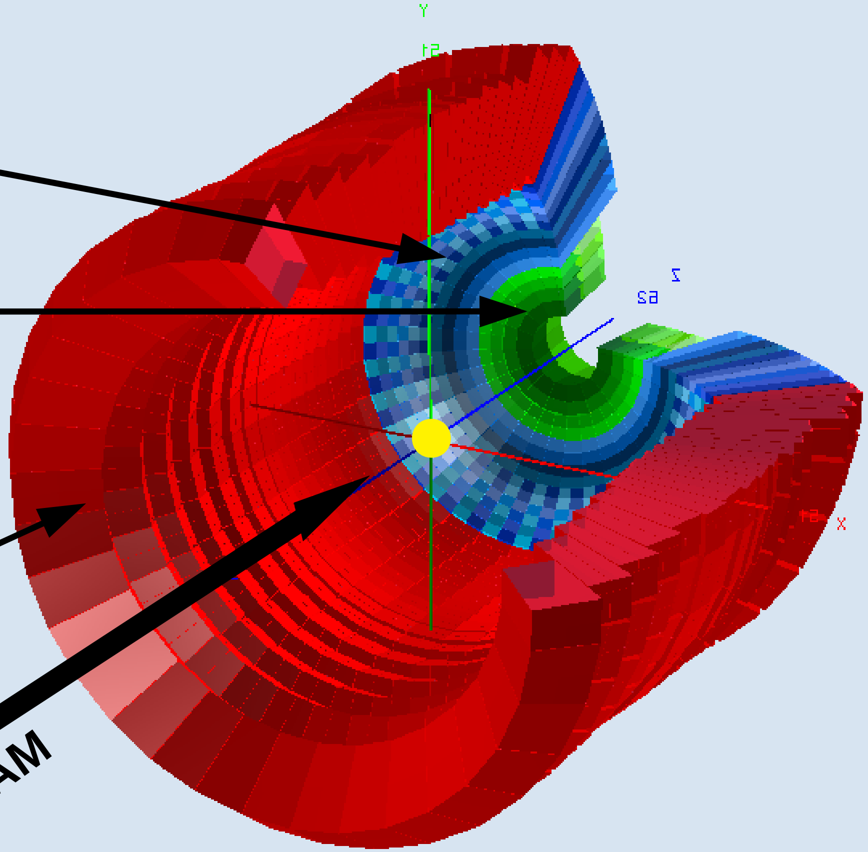
iPhos:

512 CsI(Tl) crystals

CEPA:

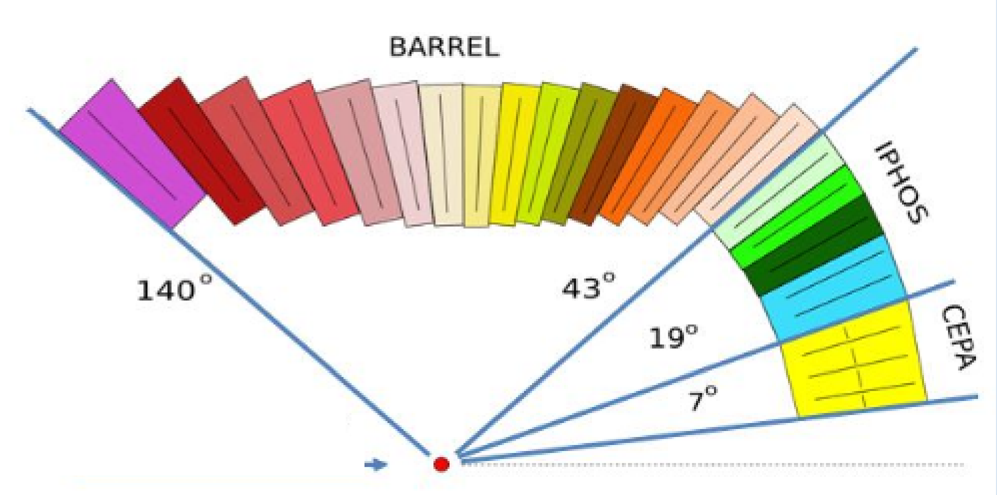
96 LaBr3 & LaCl3 crystals

Barrel:
1952 CsI(Tl) scintillator crystals



Over 2500 crystal channels!

Requirements:
- -high dynamic range:
  100 keV γ-rays – 700 AMeV charged particles
- -high efficiency
- -high granularity → Doppler correction
- -particle identification



Observables

Energy

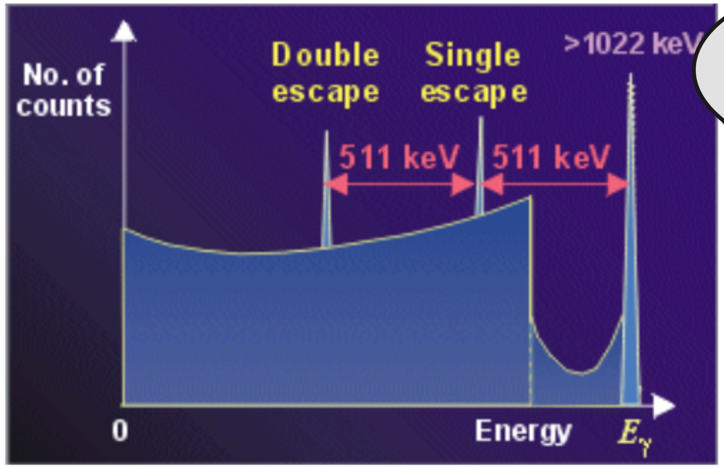
Hit Time

Spatial Position

Correlation with neighboring crystals

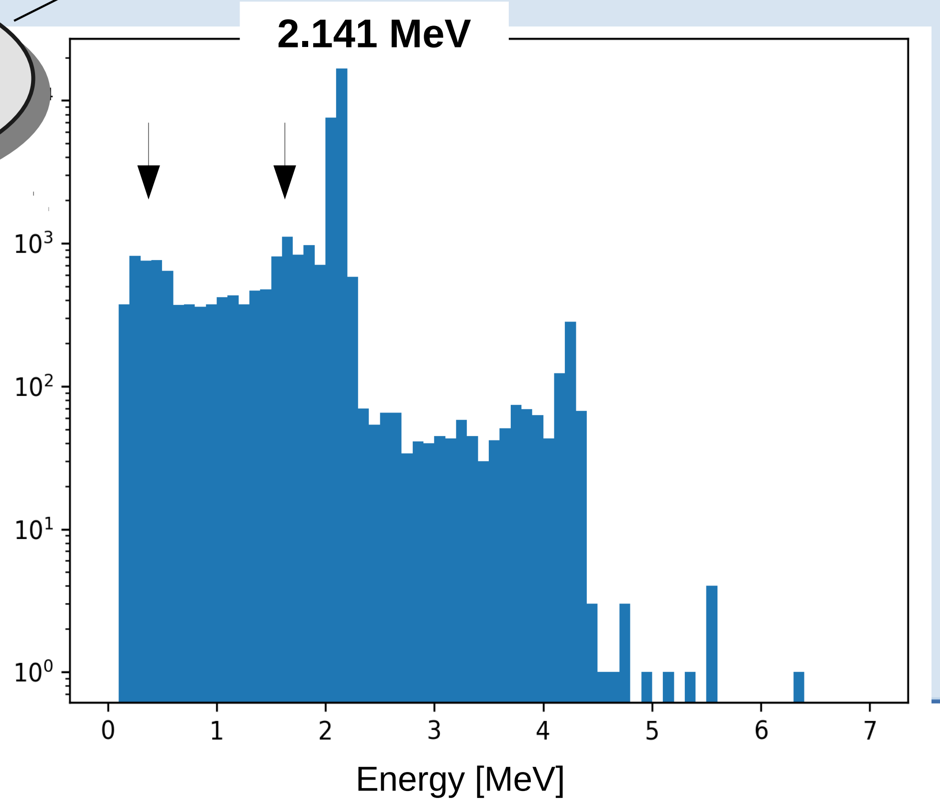
Customized Cluster

In a real detector, if the incident gamma energy is above 1022 keV, pair production events result in the production of two 511 keV annihilation gamma-rays.



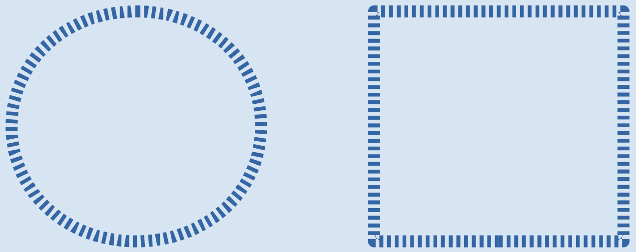
If only one of these gamma-rays escapes while the other is completely absorbed in the detector, 511 keV will be lost from the detector. This results in a separate peak in the spectrum representing Eg-511 keV, called the **single** escape peak.

If both annihilation gamma-rays escape this gives rise to the **double** escape peak at Eg-1022 keV.



Standard Cluster Algorithm

User defines shape and size of cluster:



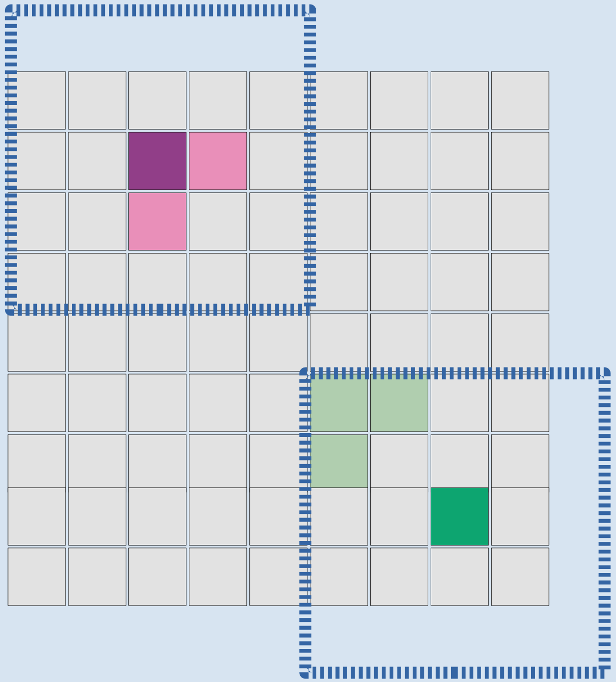
(and set energy threshold for single crystals)

Sort the hit list according to their energy

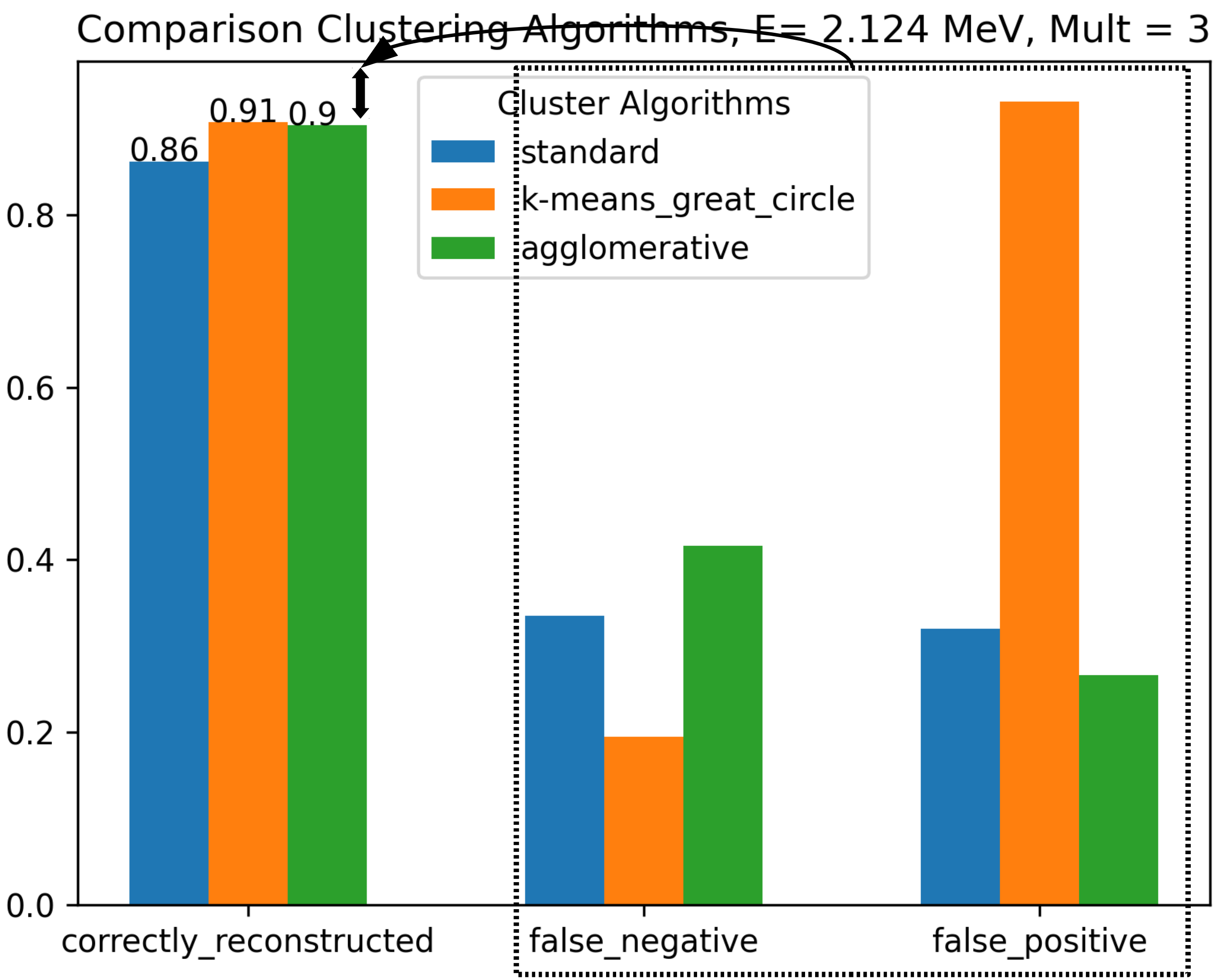
30. MeV
22. MeV
10. MeV
5. MeV
3. MeV
2.5 MeV
0.7 MeV

1. create cluster centered around first hit
2. loop over all hits in list
   → if hit inside cluster add it and remove it from the list

3. Do this procedure until list is empty



Summary Clustering Methods



false_negative:

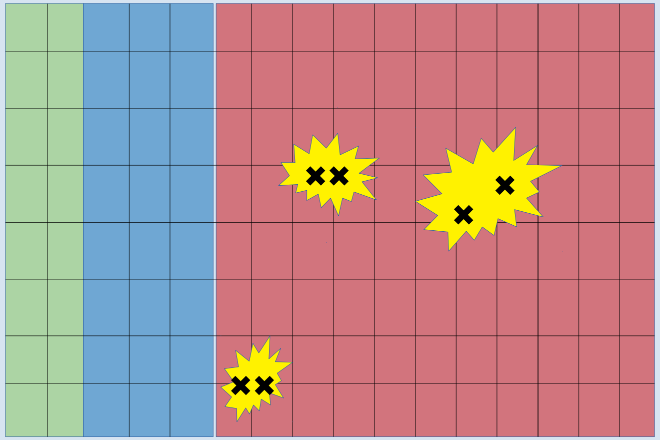
true cluster $\begin{pmatrix} 1.2 \\ 0.8 \\ \mathbf{0.124} \end{pmatrix}$ reconstructed cluster $\begin{pmatrix} 1.2 \\ 0.8 \\ 0.511 \end{pmatrix}$ $\Rightarrow$ false negative $= \dfrac{0.124}{2.124}$

false_positive:

true cluster $\begin{pmatrix} 1.2 \\ 0.8 \\ 0.124 \end{pmatrix}$ reconstructed cluster $\begin{pmatrix} 1.2 \\ 0.8 \\ \mathbf{0.511} \end{pmatrix}$ $\Rightarrow$ false positive $= \dfrac{0.511}{2.124}$

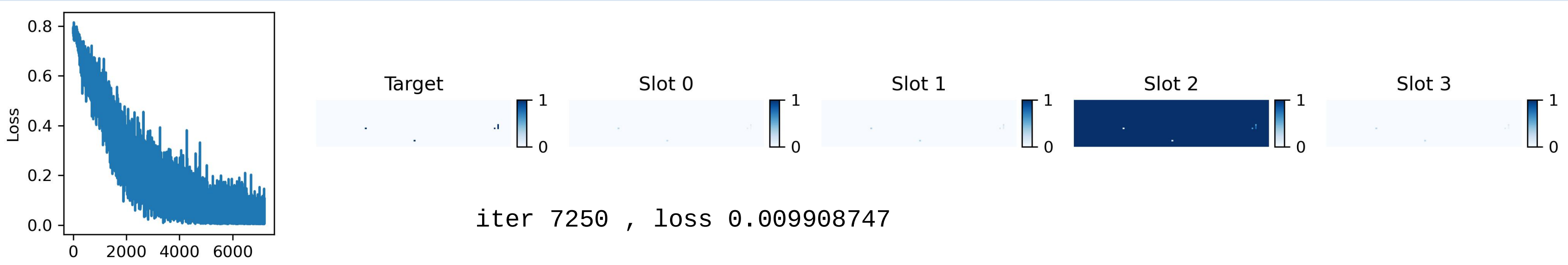
Invariant Slot Attention Model

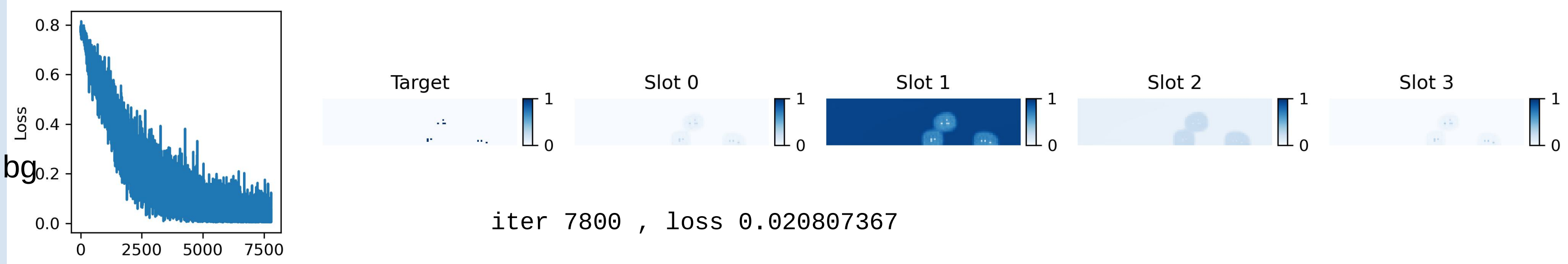
Starting with energy and position information (no time):



```
class InvariantSlotAttention(torch.nn.Module):
    def __init__(self,
                 resolution=(27,112),
                 xlow=-0.5,
                 xhigh=0.5,
                 k_slots=4,
                 num_conv_layers=3,
                 hidden_dim=16,
                 final_cnn_relu=False,
                 query_dim=32,
                 n_iter=2,
                 pixel_mult=1,
                 device='cpu'
                 ):
```

3 clusters + bg





Parameters I tuned:
Learning rate: 1e-4 to 1e-5
hidden dim: 16 – 32
query_dim: 10 – 16 – 32 - 64

Invariant Slot Attention Model – also with time info

→ Dimension of mask: → same as before! 10 x 3 x 27 x 112

→ Dimension of evt_histogram_array : 10 x **2** x 27 x 112

Lr = 5e-5

#observables:
time & energy

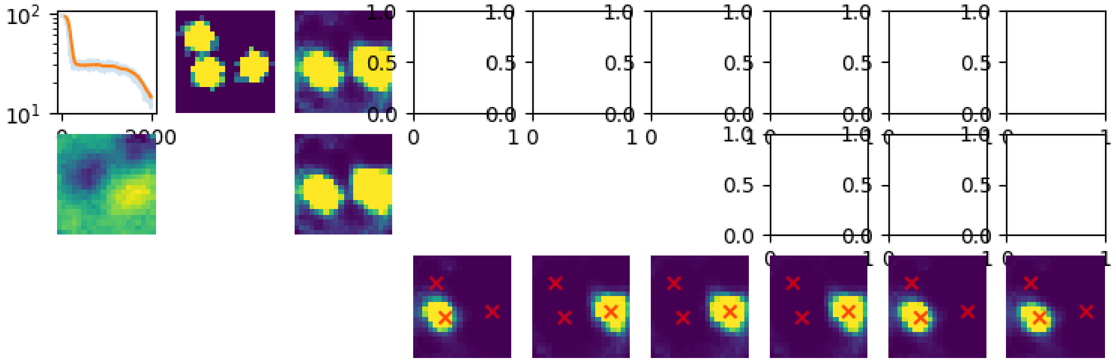
```
self.gru = torch.nn.GRUCell(self.query_dim, self.query_dim)

kwargs = {'out_channels': hidden_dim,'kernel_size': 5, 'padding':2 }
#cnn_layers = [torch.nn.Conv2d(1,**kwargs)] old cnn, with one input channel, energy
cnn_layers = [torch.nn.Conv2d(2,**kwargs)] #now also with time info

for i in range(num_conv_layers-1):
```

Loss function does not converge!

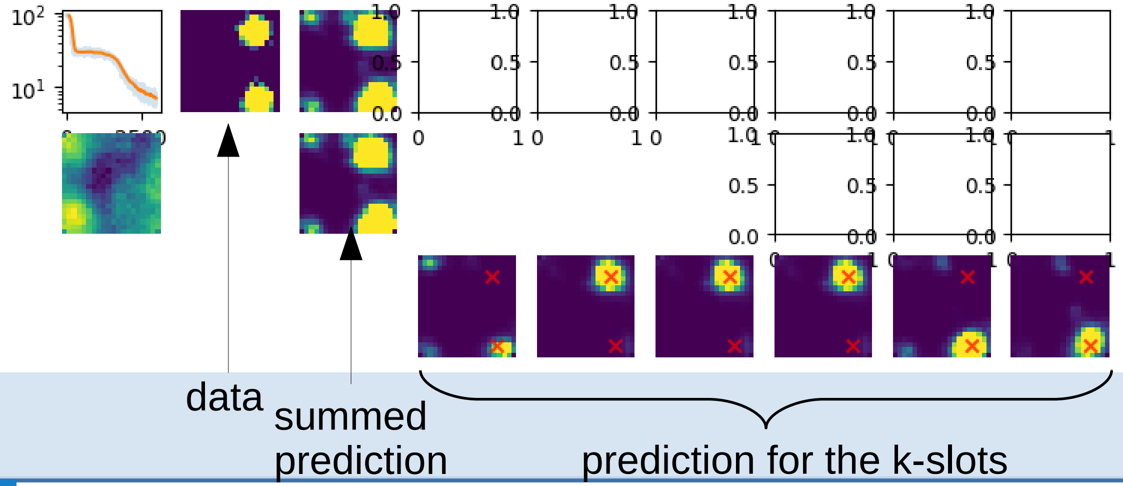
slot_and_tspn_onenotebook from Lukas Heinrich

https://gist.github.com/lukasheinrich/31d06bc4918e52d7ae3663a197b90d71

```
tensor(1.8402e-15) tensor(3.0612)
```



```
0.0003
loss 3000 tensor(8.4344, grad_fn=<MulBackward0>) 0.0003
tensor(4.0544e-09) <built-in method max of Tensor object at 0x79e592d3a2a0>
tensor(3.6479e-11) tensor(1.2241)
tensor(1.8620e-13) tensor(3.7033)
tensor(1.7014e-13) tensor(3.7114)
tensor(2.1267e-13) tensor(3.5644)
tensor(3.0657e-11) tensor(5.8000)
tensor(2.8779e-10) tensor(3.0993)
```




```python
class AttModel(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.latent_dim = 32

        # self.encoder = TSPNEncoder(n_slots = 6)
        # self.encoder = SlotAttentionEncoder(n_slots = 6)
        self.encoder = AddNoiseEncoder(n_slots = 6)
        self.decoder = torch.nn.Sequential(
            torch.nn.Linear(self.latent_dim,128),
            torch.nn.ReLU(),
            torch.nn.Linear(128,256),
            torch.nn.ReLU(),
            torch.nn.Linear(256,NBINS*NBINS),
            torch.nn.Unflatten(-1,(NBINS,NBINS))
        )


    def forward(self, data):
        Nbatch, *_ = data.shape
        positions,queries = self.encoder(data)
        decoded = self.decoder(positions).exp()/2.
        reco = decoded.sum(dim = 1)
        return reco,queries,decoded
```

Transformer model

nn.Linear(4,k)

nn.TransformerEncoder

Cosine-similarity

Source

[energy$_1$, phi$_1$,theta$_1$, time$_1$]
[energy$_2$, phi$_2$,theta$_2$, time$_2$]
....
[energy$_N$, phi$_N$,theta$_N$, time$_N$]

hitnr

k = 32
heads = 8
num_layers = 6

Upper triangle matrix
hitnr x hitnr

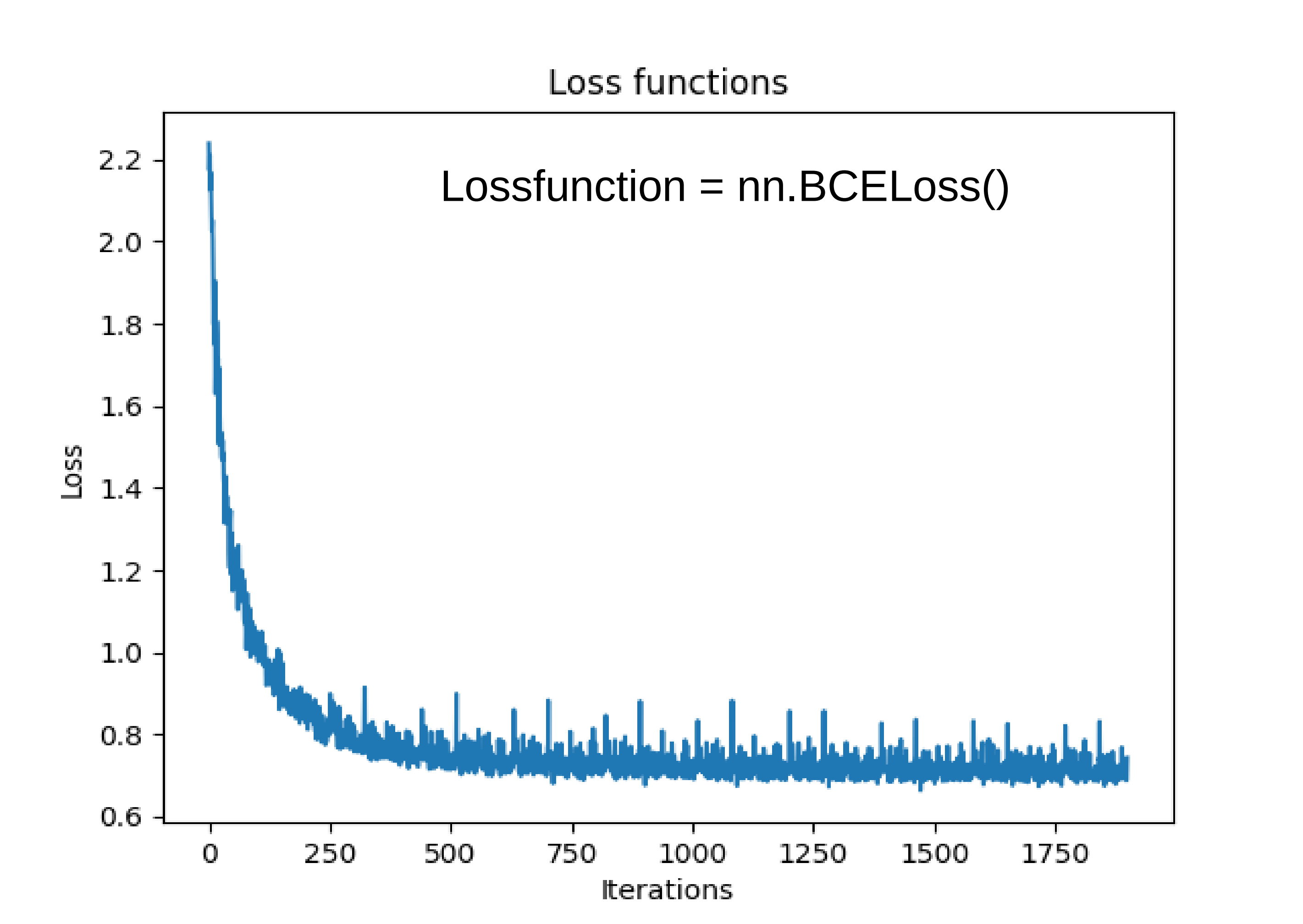
Calculate BCELoss

Target

Upper triangular matrix
(with zeros and ones)

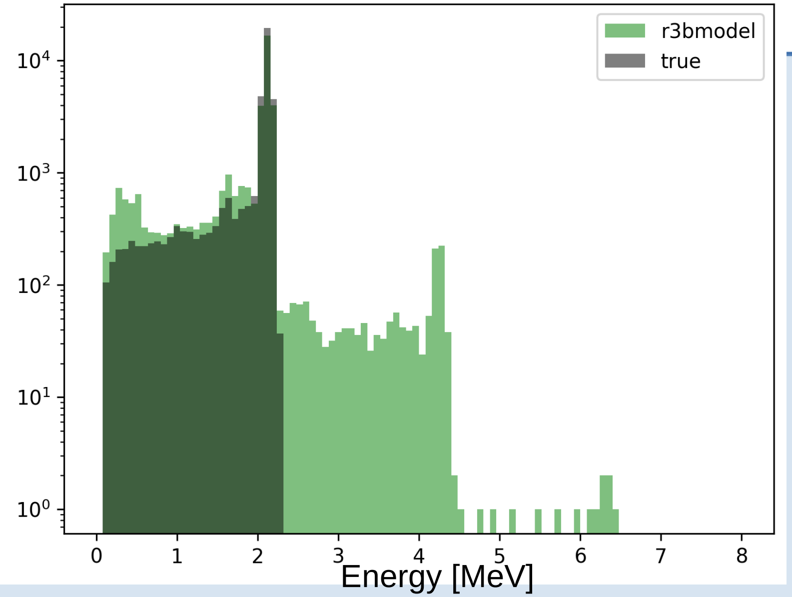
Transformer model – further parameters

Batchsize = 64
Feature number = 32
n_epochs = 10
Loss_rate = 2e-4
Loss function = nn.BCELoss()



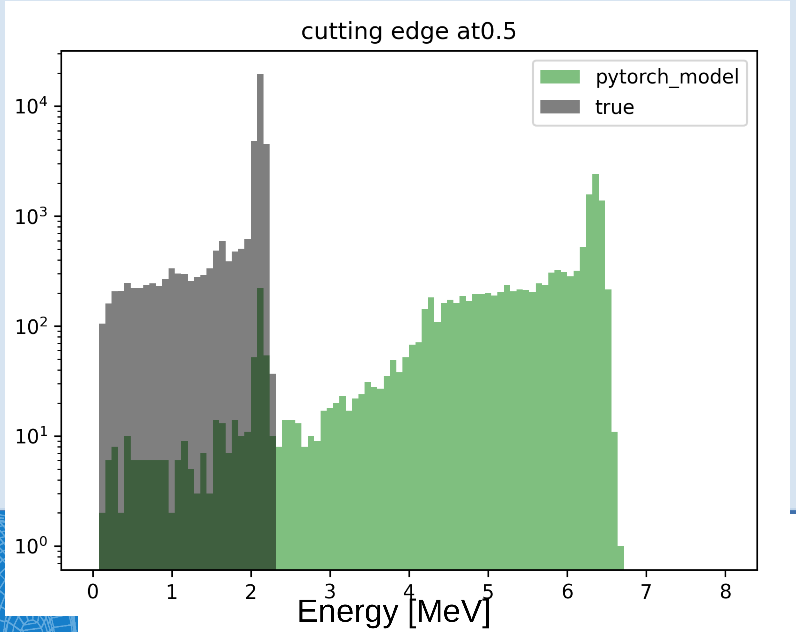
How do the energy spectra look like?

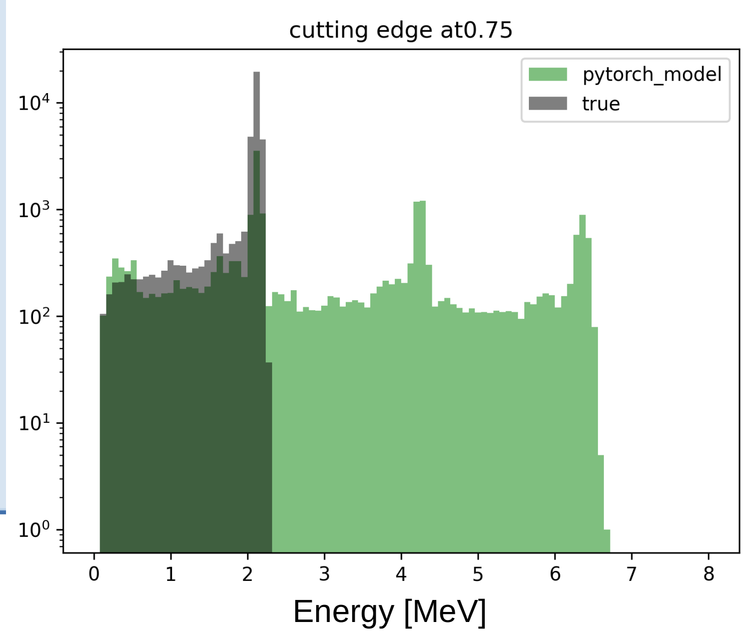
Standard Cluster vs True Clusters

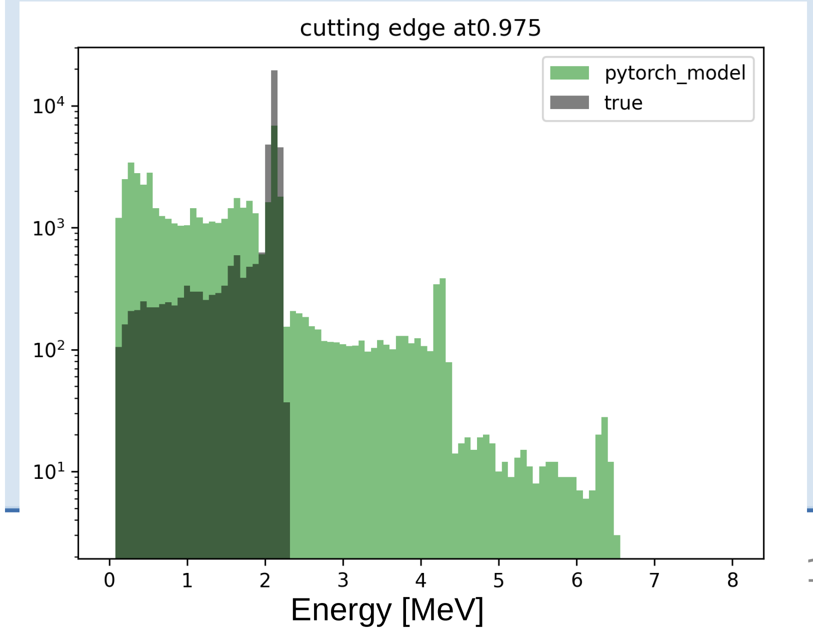


How to clusterize hits from output of transformer model:

1) Take the upper triangular matrix tri[hitnr x hitnr]
2) set “merge cut”. If tri[i,j] > “merge_cut” → hits belong to same cluster
3) do this for all combinations and merge them appropriately







Why energy spectra so bad while loss function seems to decrease ?

Most entries in model output tensor ~0.5. This diminishes the loss BCELoss function!

How to improve?

- Include some cut condition in the forward part of the transformer model

```
#out_ret_val = torch.where(ret_val > 0.7, torch.FloatTensor(1,requires_grad=True), torch.FloatTensor(0,requires_grad=True))
```

→ discontinuity of loss function → no learning!

- Use linear net instead of cosine similarity

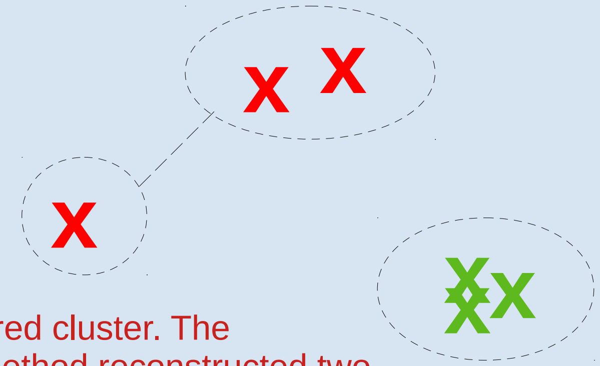
```
output_tensor = torch.cat([tensor_i.expand(-1, -1, expansion_factor, -1), tensor_j.expand(-1, expansion_factor, -1, -1)], dim=-1)
#small net:
net = nn.Sequential(
        nn.Linear(64,8),
        nn.ReLU(),
        nn.Linear(8,1),
        nn.Sigmoid()
        )
res = net(output_tensor)
temp_res = torch.squeeze(res)
upper_tri_mask = torch.triu(torch.ones((temp_res.shape[1],temp_res.shape[1])),diagonal=1).bool() #out[1] is max hit number in batch
result = temp_res[:,upper_tri_mask]
```

No improvements!

Agglomerative Model + Transformer Model

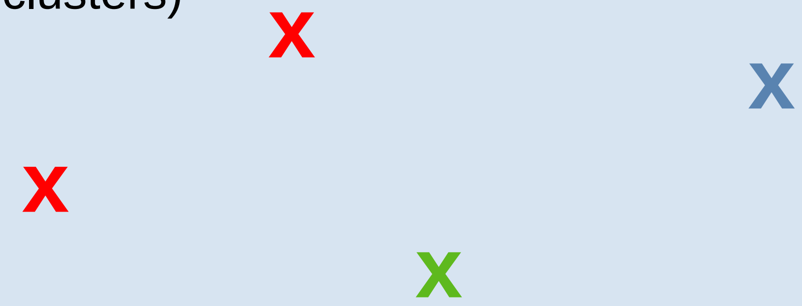
Idea:

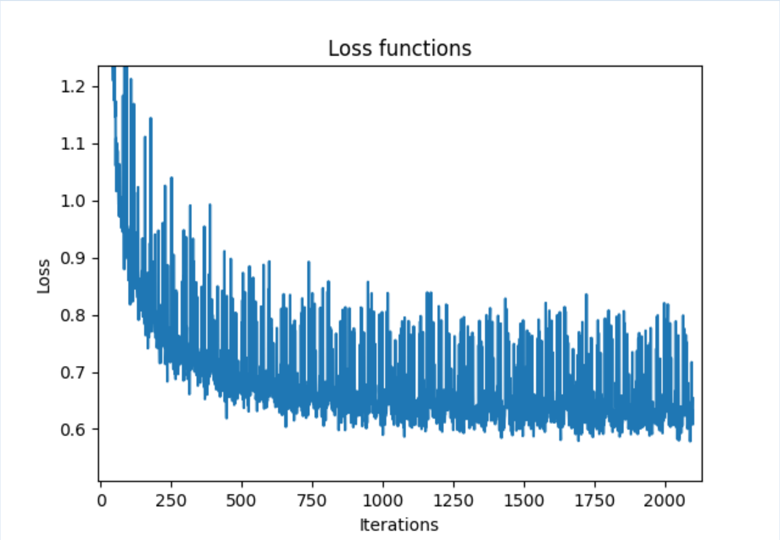
1. Use first agglomerative method to cluster
2. Select events where we have too many clusters (false negative)



This is one true red cluster. The agglomerative method reconstructed two

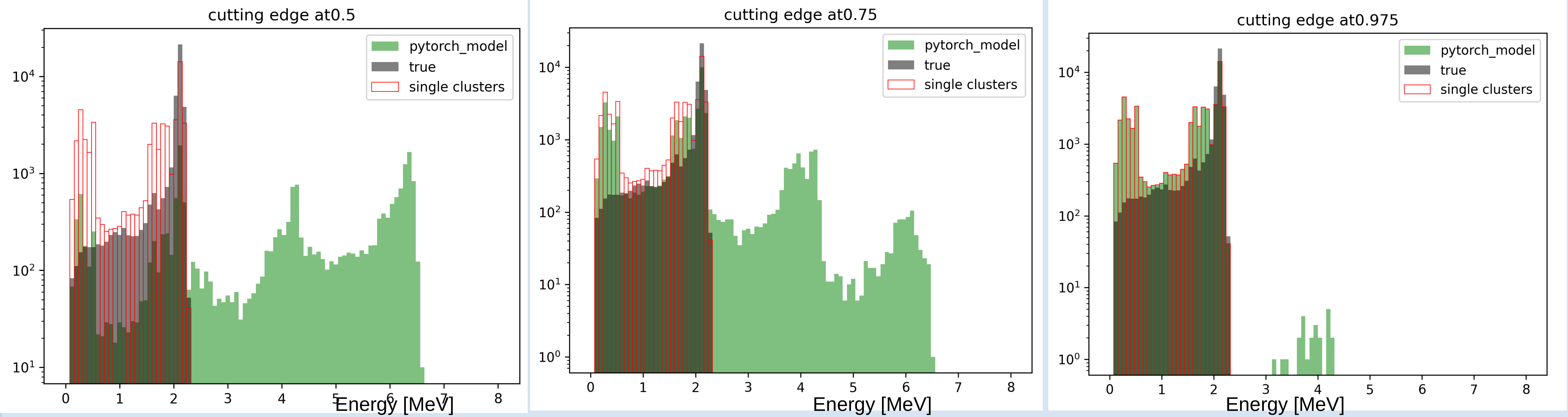
3. Feed the clusters to the transformer model (calculating cm of clusters)





Transformer Model

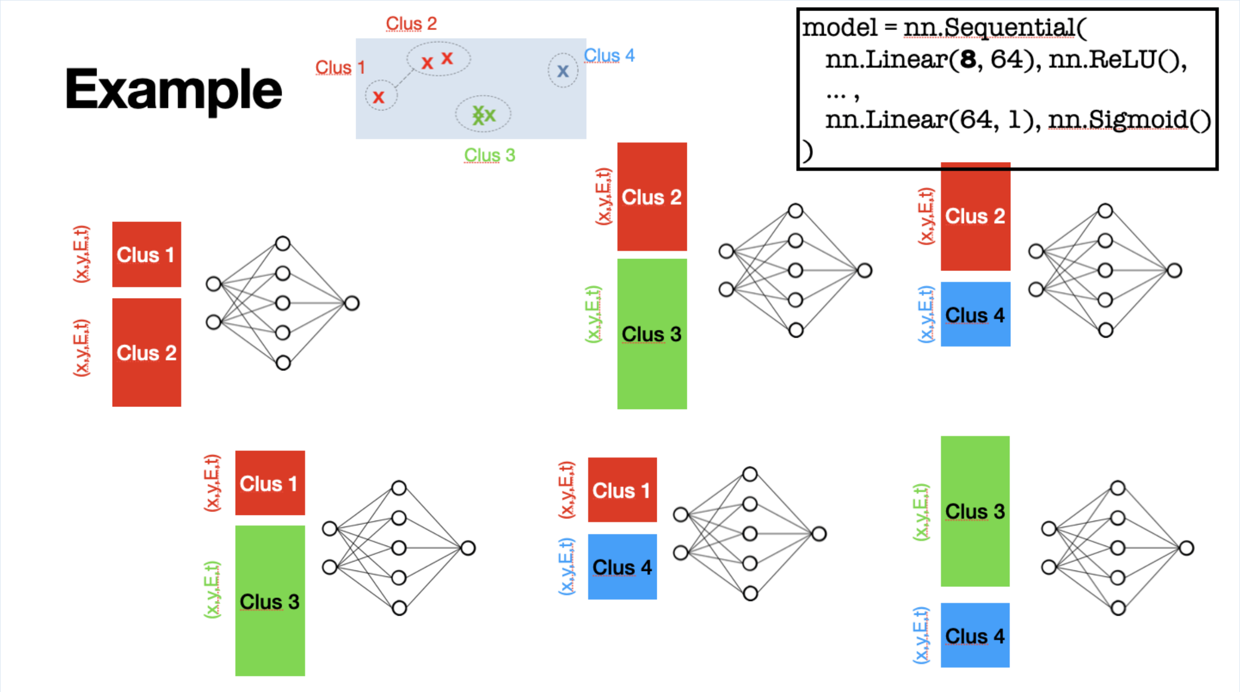
Reconstruction with transformer model (after application of agglomerative model)



Cutting edge: model give output in range [0,1]. Cutting edge is threshold:
If cutting edge > pairwise cluster output → clusters do not belong together
If cutting edge < pairwise cluster output → clusters belong together

No improvement in reconstruction,
High cutting edge → = single clusters
Low cutting edge → too many clusters are merged

Single Feed Forward NN



Since transformer method not successful, start with basic model:

Def init :

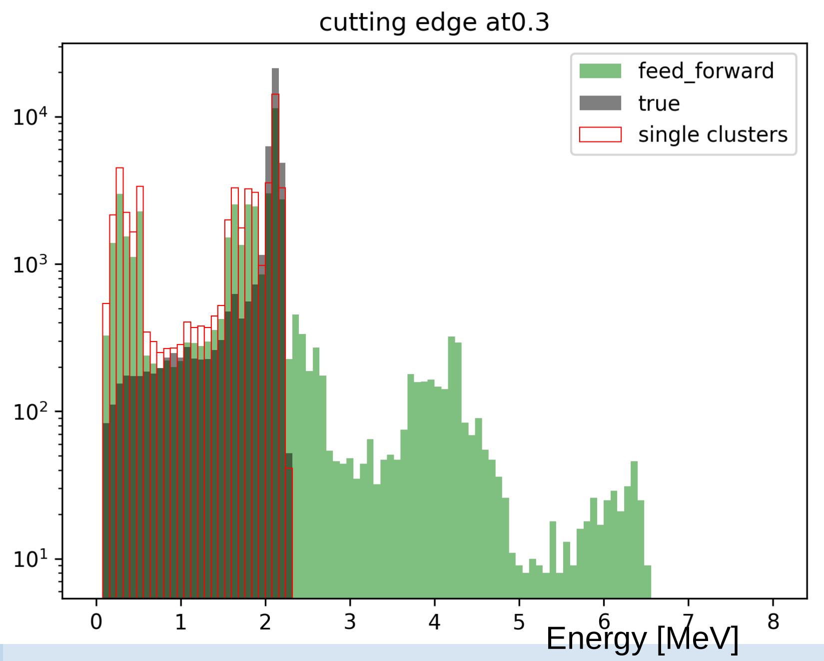
    self.linear = torch.nn.Linear(8,64)
    self.activation = torch.nn.ReLU()
    self.linear_back = torch.nn.Linear(64,1)

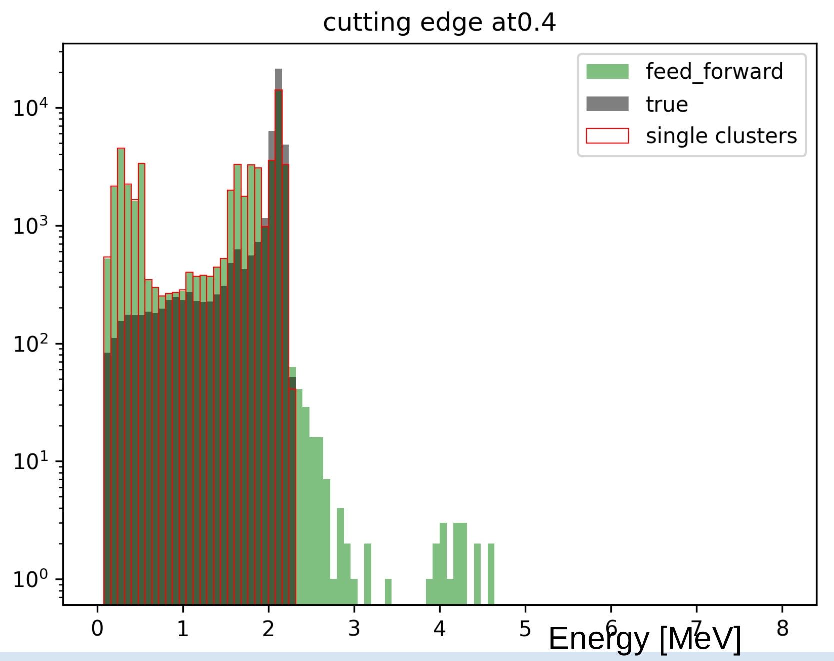
….

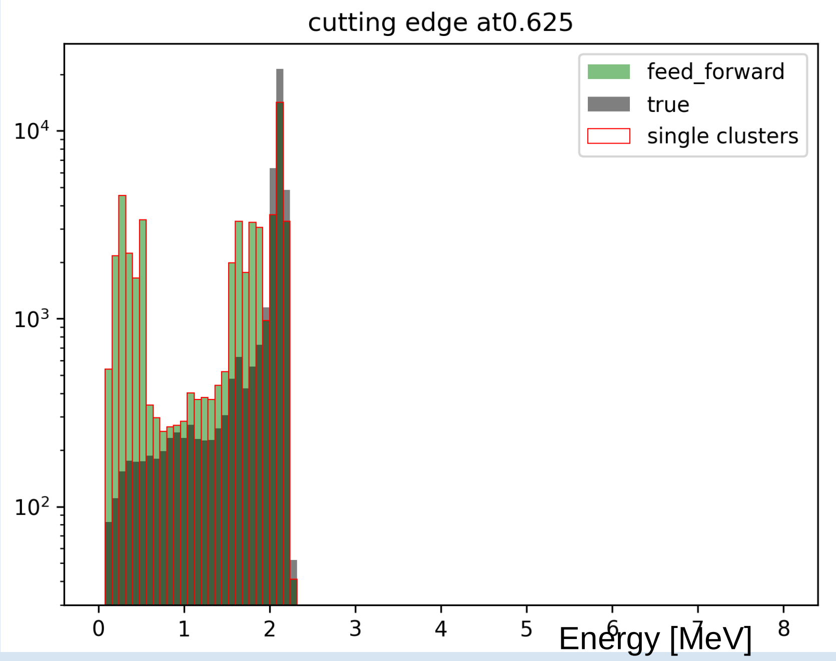
Def forward:

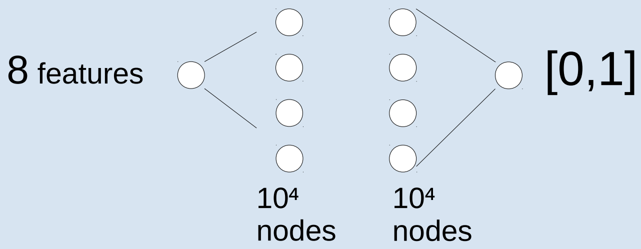
    output_tensor = self.linear(output_tensor)
    output_tensor = self.activation(output_tensor)
    output_tensor = self.linear_back(output_tensor)
    output_tensor = torch.sigmoid(output_tensor)
    output_tensor = torch.squeeze(output_tensor)

Reconstruction with feed forward (after application of agglomerative model)

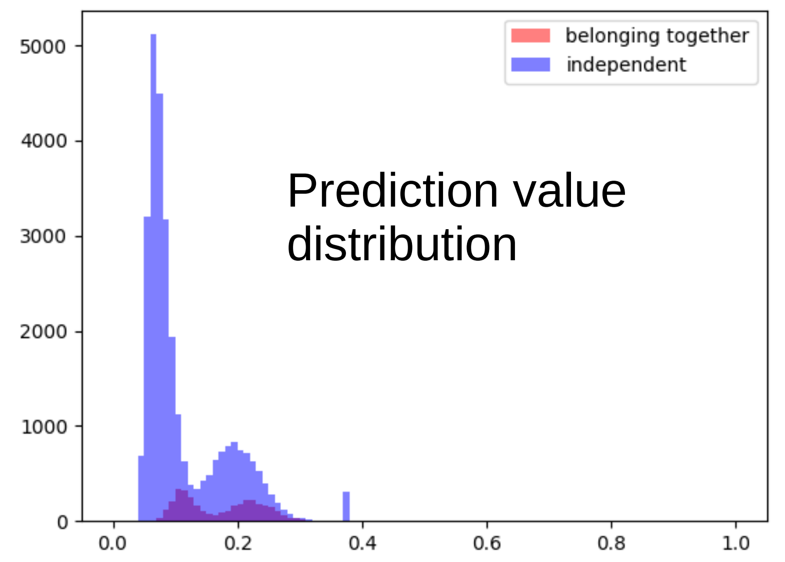








0: indep. hits
1: belonging together



Same as in transformer model. However cutting edge has to be set really low...

No improvement in reconstruction!

Thank you!

CALIFA @ Technical University of Munich (TUM)

Roman Gernhäuser, Lukas Ponnath, Philipp Klenze, Tobias Jenegger

GEFÖRDERT VOM
Bundesministerium
für Bildung
und Forschung

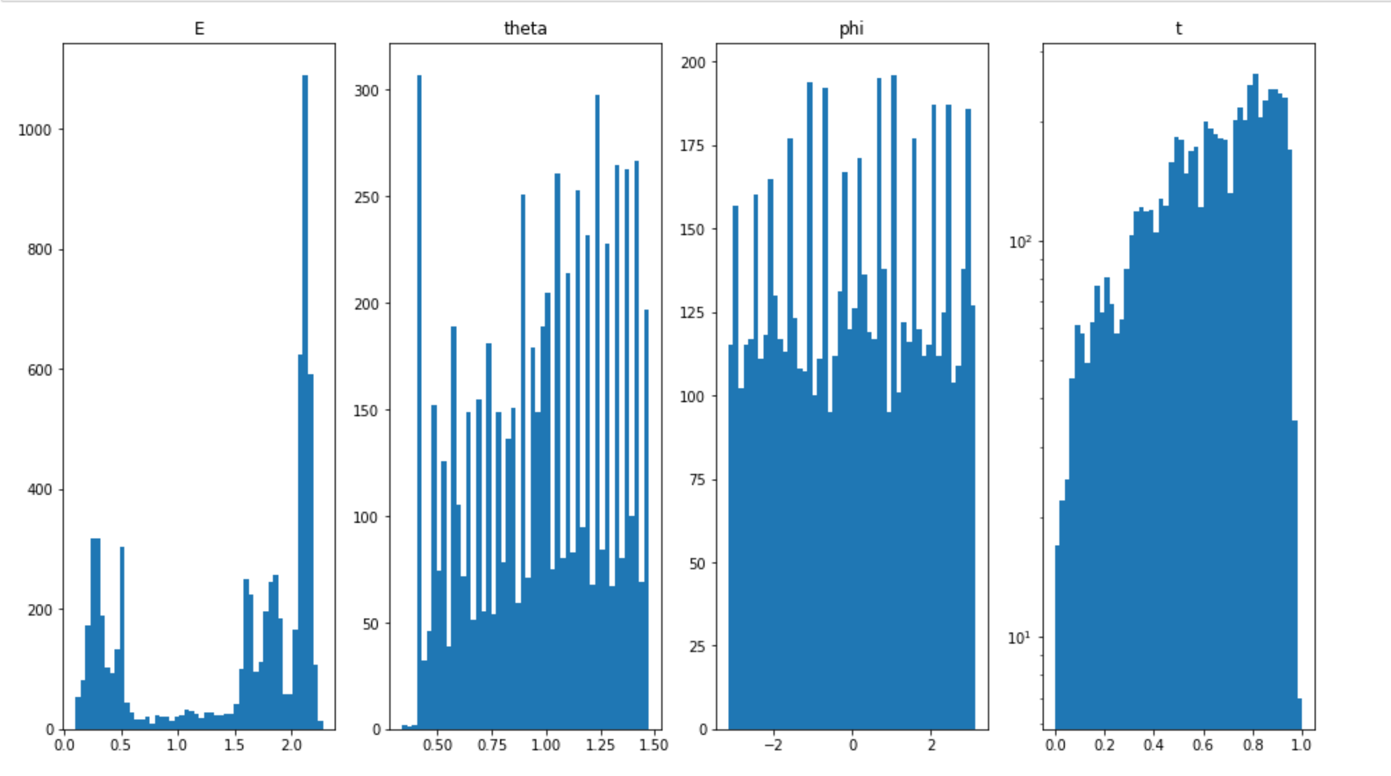
Backup

How data looks like we give to the feed forward NN

We applied agglomerative cluster
What you see here is already clustered data
BUT all events where you have too many clusters

Agglomerative clustering has also drawbacks:
Clustering is done using E,theta,phi,time where the time is taken as radius

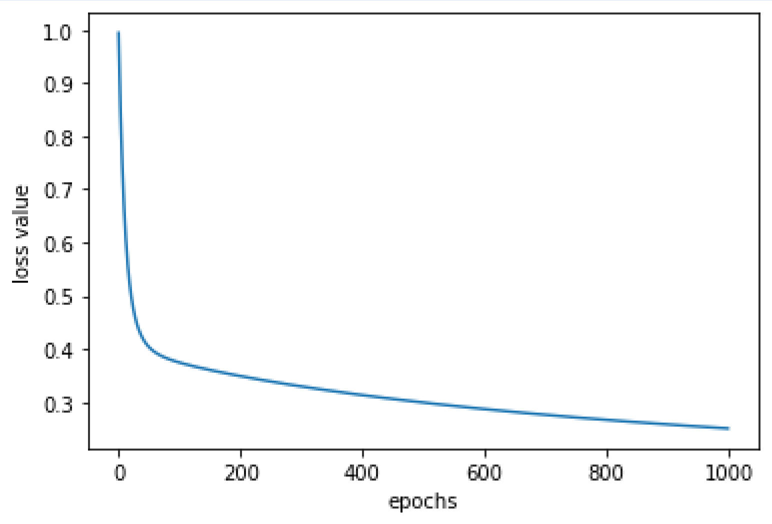
→ time starts from 0 up to…
→ for time ~0 everything is clustered togeter!



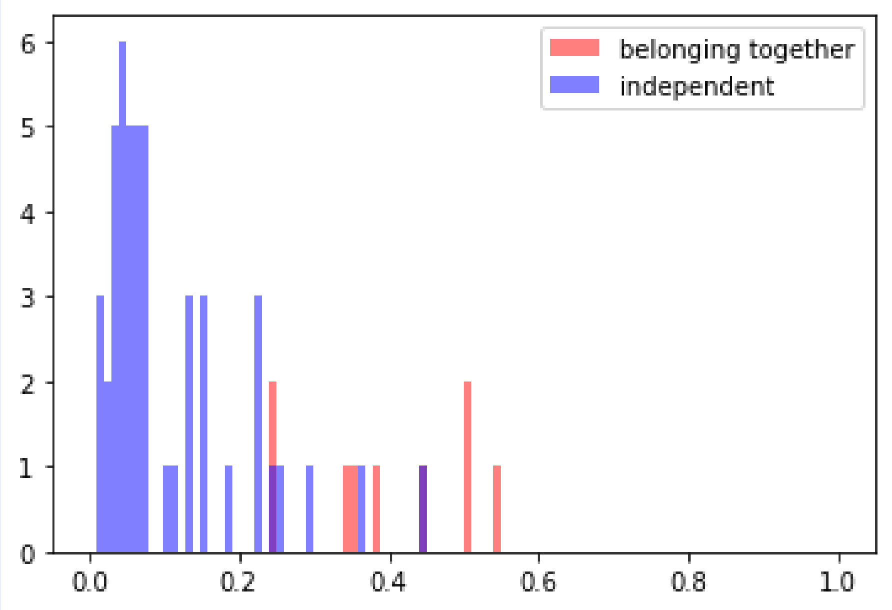
Selecting only 10 events (with 3 sub-events)

[0.	1.128016	0.913283	-2.69977	0.27121927]
[1.	0.952915	0.519144	0.878809	0.95261156]
[1.	0.434082	0.56767	-2.311936	0.93668561]
[2.	2.209619	1.105468	2.333682	0.52416023]
[3.	1.71197	1.189179	1.119164	0.78435412]
[3.	0.461599	0.777167	-0.144536	0.75403444]
[4.	2.029792	1.287946	1.30295	0.]
[5.	0.756178	1.33176	-2.309371	0.68407621]
[6.	2.1872	0.476108	2.85122	0.8842718]
[7.	1.921325	1.287711	-2.407052	1.]
[7.	0.24677	0.972258	1.512456	0.95922116]
[8.	2.12424	1.190137	0.532672	0.63181968]
[9.	2.165567	0.780356	1.333373	0.52338727]
[10.	1.927867	0.979187	-1.390222	0.83159162]
[10.	0.155214	1.058012	-2.217043	0.81032574]
[11.	2.17151	0.581699	-0.050691	0.75626808]
[12.	2.145626	0.444849	1.000789	0.69102447]
[13.	2.124236	0.907799	1.22338	0.93919196]
[14.	1.594131	0.931724	-0.470997	0.35572331]
[14.	0.518059	0.958237	1.420141	0.31123073]
[15.	1.55693	0.851083	-1.916148	0.3734817]
[15.	0.206072	1.377727	0.637334	0.43616126]
[16.	1.702479	1.014433	2.493843	0.51384286]
[17.	1.654359	1.057384	1.224166	0.95706616]
[18.	1.647063	1.212656	-0.142311	0.9388571]
[18.	0.479129	1.147077	2.998456	0.88034231]
[19.	0.887473	0.777098	-2.89615	0.68520254]
[20.	2.10968	0.773892	1.628614	0.85522686]
[21.	1.16823	1.331397	-1.120366	0.96468413]
[21.	0.287956	1.237488	-2.217654	0.97093477]
[22.	2.154496	0.828804	-0.161298	0.79557305]
[23.	1.302665	0.712428	-0.693234	0.61380342]
[24.	1.599256	0.952238	-2.448033	0.93131397]
[24.	0.505705	1.100893	-2.893951	0.92312015]
[25.	2.141112	0.839635	0.501495	0.51226942]
[26.	2.075262	1.398027	1.62944	0.71680964]
[27.	2.107771	1.101413	-2.803576	0.52078744]
[28.	2.133439	1.160672	0.212617	0.49220052]
[29.	0.432454	1.237877	-2.503905	0.94673509]

Lr = 9e-4
torch.nn.Linear(8,1000)

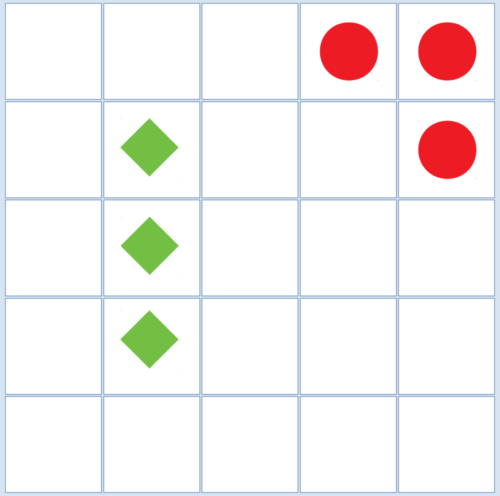


Prediction value distribution



i,i+1,i+2 belong together, in sets of 3

Generic Approach



Small grid (10x10)
Two (true clusters), with sparse data
Gaussian energy distribution of cluster hits

With machine learning tools we should get at least as good as with the standard clustering (even without considering time information)!