Loss Function Calculation in CrinGe

Mo Jia

Hit Loss

$$\text{Loss} = -\log L = \sum_{i} -\log P_{unhit}(y_i) + \sum_{i_{hit}} -\log p_{qt}(q_{i_{hit}}, t_{i_{hit}})$$

Hit_loss

All PMTs

qt_loss

Hit PMTs

```
50
51          self.bceloss = torch.nn.BCEWithLogitsLoss(reduction = "none")
52


213         hitMask = charge > 0
214         hit_loss_tensor = self.bceloss(punhit, (charge == 0).float())
215         hit_loss = hit_loss_tensor[:,mask].sum()
216         coefficients = torch.nn.functional.softmax(prediction[:, -self.N_GAUS:], dim = 1)
```

Pytorch's BCEWithLogitsLoss

- https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html

BCEWITHLOGITSLOSS

CLASS `torch.nn.BCEWithLogitsLoss(weight=None, size_average=None, reduce=None, reduction='mean', pos_weight=None)` [SOURCE]

This loss combines a *Sigmoid* layer and the *BCELoss* in one single class. This version is more numerically stable than using a plain *Sigmoid* followed by a *BCELoss* as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.

The unreduced (i.e. with `reduction` set to `'none'`) loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top, \quad l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right],$$

Our Case

where N is the batch size. If `reduction` is not `'none'` (default `'mean'`), then

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction} = \text{'mean'}; \\ \text{sum}(L), & \text{if reduction} = \text{'sum'}. \end{cases}$$

This is used for measuring the error of a reconstruction in for example an auto-encoder. Note that the targets $t[i]$ should be numbers between 0 and 1.

Hit_loss

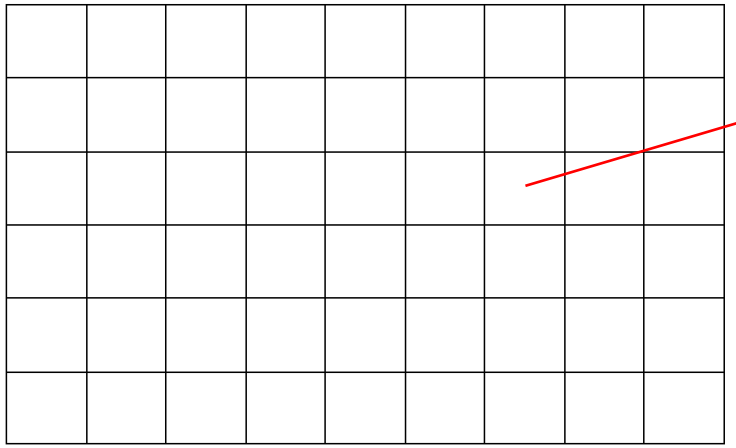
$$\ell(x, y) = L = \{l_1, \ldots, l_N\}^\top$$

a Batch

Event n

$$l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right]$$

- What we get from calling the function is a tensor/array, with results of l_n at each PMT as the elements



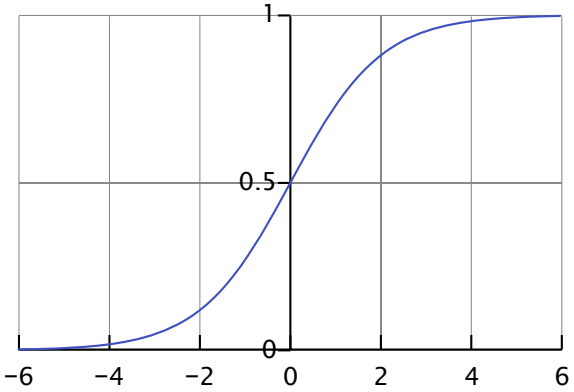
PMT i in event *n*
Prediction: x
Target: y

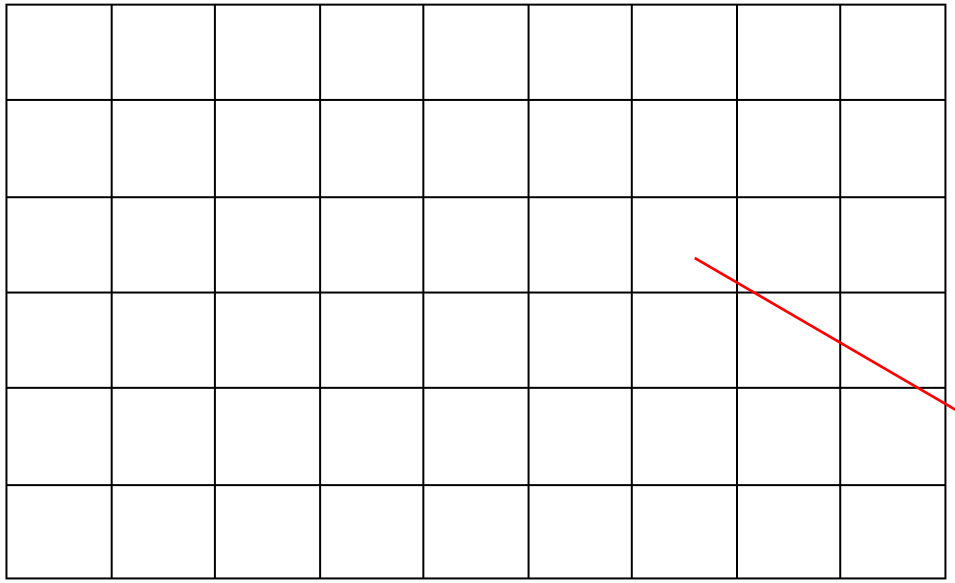
Hit_loss

$$l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right]$$

Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



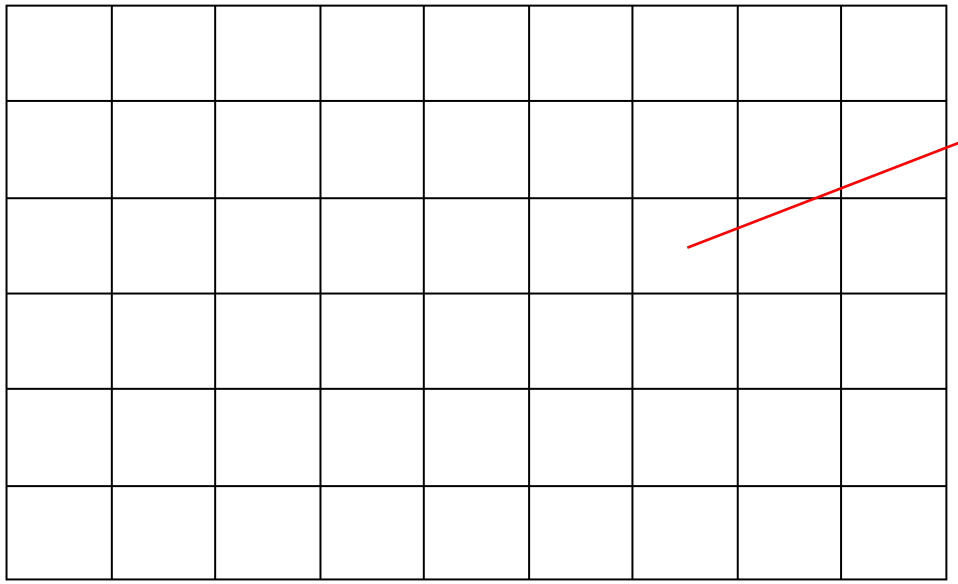


```
hitMask = charge > 0
hit_loss_tensor = self.bceloss(punhit, (charge == 0).float())
hit_loss = hit_loss_tensor[:,mask].sum()
```

- Punhit: x_n ,Output of the model
- (charge==0): y_n, from data, 1 if unhit or 0 if hit

$$\ell(x, y) = \begin{cases} -\log(1 - \sigma(x_n)), & \text{if } y_n = 0, \text{hit}; \\ -\log \sigma(x_n), & \text{if } y_n = 1, \text{unhit}. \end{cases}$$

Possibilities to Lead the Hit_loss to be Nan



$$\ell(x,y) = \begin{cases} -\log(1-\sigma(x_n)), & \text{if } y_n = 0, \text{hit;} \\ -\log\sigma(x_n), & \text{if } y_n = 1, \text{unhit.} \end{cases}$$

$$\sigma(x_n) \to 1$$

- Possible when the prediction x_n is relatively large

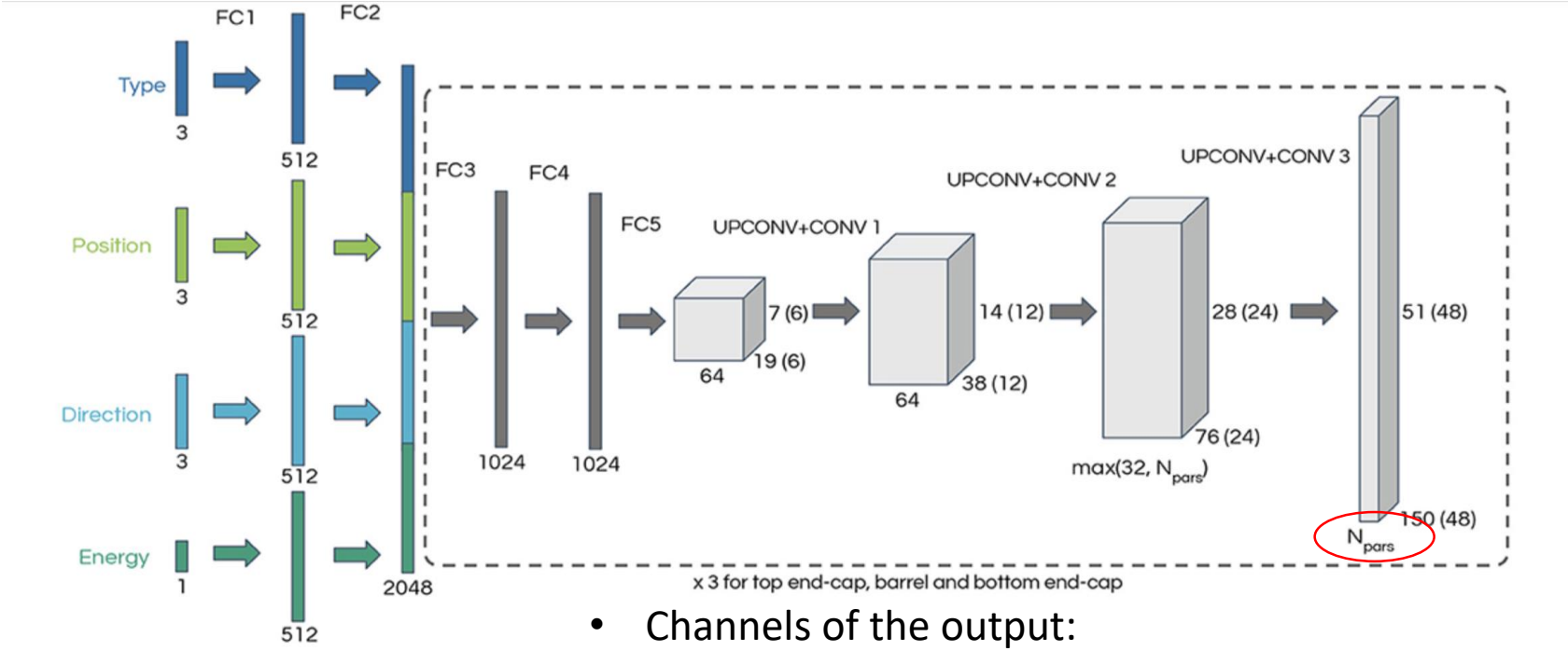
$$\sigma(x_n) \to 0$$

- Possible when the prediction x_n is negative

- One possible reason leading to a *nan* error is the input of log is considered as 0 by the computer
- Sigmoid(x) lies within {0.5, 1) if x > 0, with x as the prediction of the model, which should be our case if x is actually the probabilities made by the network
  - It is possible of the network to report a larger figure
  - A negative figure?

Meaning of the Output Channels from the Network?

- In current version, the first channel is considered as the hit probabilities.
- But the reason is not clear to me…



- Channels of the output:

Each channel represents a set of the predictions for a feature parameter in the PMT response function

- [hit_probability, GaussionComponent1_paramenters, GausComponent2_parameters, ……., normalization_paramenters]

Attempts to check hit_loss

- add a few new methods to the model:

```
310     def multiGausLoss_hitLoss_scan(self, prediction, charge, mask = None, time = None, t0 = 0.) :
311
312         charge_n = torch.stack( [ charge for i in range(self.N_GAUS) ], dim = 1 )
313         if time is not None:
314             time_n = torch.stack([ time - t0 for i in range(self.N_GAUS) ], dim = 1)
315
316         punhit = prediction[:,0]
317
318         # Hack to make this work on ivy's older torch version. Fix after upgrading ivy!
319         if torch.__version__ == '1.0.1' :
320             this_bool_dtype = torch.uint8
321         else :
322             this_bool_dtype = torch.bool
323         if mask is None :
324             mask = torch.full_like(punhit[0], True, dtype = this_bool_dtype, device = self.device)
325         else :
326             mask = torch.squeeze(torch.tensor(mask, dtype = this_bool_dtype, device = self.device), dim = 0)
327
328         hit_loss_tensor = self.bceloss(punhit, (charge == 0).float())
329
330         hit_loss_err_tensor = torch.isnan(hit_loss_tensor)
331         hit_loss_err_tensor_mask = hit_loss_err_tensor[:,mask]
332
333         punhit_mask = punhit[:,mask]
334         punhit_p = punhit_mask[hit_loss_err_tensor_mask]
335
336         charge_mask = charge[:,mask]
337         charge_p = charge_mask[hit_loss_err_tensor_mask] * self.charge_scale
338
339         err_num = hit_loss_err_tensor_mask[:].sum().item()
340
341         errors = {"num":err_num, "punhit":punhit_p.tolist(),"charge":charge_p.tolist() }
342         return errors
```

Attempts to check hit_loss

- add a few new methods to the model:

```
344     def nanLoss_diagnosis(self, Train = True, t0 = 0.) :
345         with torch.set_grad_enabled(Train) :
346             data = torch.as_tensor(self.data, device = self.device)
347 
348             prediction_barrel, prediction_bottom, prediction_top = self(data)
349 
350             if self.use_time :
351                 barrel_hitloss_errs = self.multiGausLoss_hitLoss_scan(prediction_barrel, self.charge_barrel, mask = None, time = self.time_
    barrel, t0 = t0)
352                 top_hitloss_errs = self.multiGausLoss_hitLoss_scan(prediction_top, self.charge_top, mask = self.top_mask, time = self.time_
    top, t0 = t0)
353                 bottom_hitloss_errs = self.multiGausLoss_hitLoss_scan(prediction_bottom, self.charge_bottom, mask = self.bottom_mask, time
    = self.time_bottom, t0 = t0)
354             else :
355                 barrel_hitloss_errs = self.multiGausLoss_hitLoss_scan(prediction_barrel, self.charge_barrel, mask = None)
356                 top_hitloss_errs = self.multiGausLoss_hitLoss_scan(prediction_top, self.charge_top, mask = self.top_mask)
357                 bottom_hitloss_errs = self.multiGausLoss_hitLoss_scan(prediction_bottom, self.charge_bottom, mask = self.bottom_mask)
358 
359             err_breakdown = {}
360             if barrel_hitloss_errs["num"] + top_hitloss_errs["num"] + bottom_hitloss_errs["num"] == 0:
361                 err_breakdown["noneNan"] = True
362             else:
363                 err_breakdown["noneNan"] = False
364                 err_vect = []
365                 punhit_list = []
366                 charge_list = []
367                 for err in [barrel_hitloss_errs, top_hitloss_errs, bottom_hitloss_errs]:
368                     err_vect.append(err["num"])
369                     punhit_list.append(err["punhit"])
370                     charge_list.append(err["charge"])
371 
372                 err_breakdown["err_num"] = err_vect
373                 err_breakdown["punhit_list"] = punhit_list
374                 err_breakdown["charge_list"] = charge_list
375 
376             return err_breakdown
```

Attempts to check hit_loss

- A script to run the nan_loss check:

```
61      network = load_model(args)
62
63      test_loader = iotools.loader_factory('H5Dataset', batch_size=1, shuffle=False, num_workers=0, pin_memory = True, data_dirs=args.data_dirs.split(","
    ), flavour=args.data_flavour, start_fraction=0.0, use_fraction=1., read_keys= ["positions","directions", "energies", "event_data_top", "event_data_bott
    om"])
64
65      # Grab end-cap masks from one of the input files
66      with h5py.File(glob(args.data_dirs+"/*"+args.data_flavour)[0], mode = "r") as f :
67          network.top_mask = f['mask'][0]
68          network.top_mask = network.top_mask.reshape(-1, network.top_mask.shape[0]*network.top_mask.shape[1])
69          network.bottom_mask = f['mask'][1]
70          network.bottom_mask = network.bottom_mask.reshape(-1, network.bottom_mask.shape[0]*network.bottom_mask.shape[1])
71
72      # Output file
73      record = []
74      for i_event, event in enumerate(test_loader) :
75          if i_event >= args.n_events :
76              break
77          print(i_event)
78
79          network.eval()
80          network.fillLabel(event)
81          network.fillData(event)
82
83          loss_diag = network.nanLoss_diagnosis(False)
84          if loss_diag["noneNan"]:
85              print("no nan loss found\n")
86              continue
87          else:
88              print("eventID ", i_event, "\n num of nans \n", loss_diag["err_num"], "\n punhit_list: \n" , loss_diag["punhit_list"], "\n charge_list: \n"
    , loss_diag["charge_list"])
89              res = {"evtID": i_event, "errVec": loss_diag["err_num"], "punhitList":loss_diag["punhit_list"], "chargeList":loss_diag["charge_list"]}
90              record.append(res)
91
92
93      with open(args.output_dir+"/"+args.model+"_nanLoss_record.p","wb") as f:
94          pickle.dump(record,f)
95
96      print("Scanning Done")
97
```

Some validation

- Do the nan_loss scan with a successfully trained model:

```
Loading model: CRinGe_SK_MultiGaus_Revised
With options:  {'N_GAUS': 5}
FILES!
['/home/mojia/HEPML/CRinGe_v1/CRinGe/storage/WCSim_gamma_npztoh5_test.h5']
0
no nan loss found

1
no nan loss found

2
no nan loss found

3
no nan loss found

4
no nan loss found

5
no nan loss found

6
no nan loss found

7
no nan loss found

8
no nan loss found

9
no nan loss found

Scanning Done
```

Some validation

- Do the nan_loss scan with a problematic trained model for the same dataset:

```
FILES!
['/home/mojia/HEPML/CRinGe_v1/CRinGe/storage/WCSim_gamma_npztoh5_test.h5']
0
eventID  0
 num of nans
 [7650, 1748, 1748]
 punhit_list:
 [[nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, nan, na
n,
```

An naive attempt to avoid loss explosion during training

- Decreasing the charge scale factor, so the charge on each PMT taken by the model is increased.
  - the reason behind this idea is to bypass potential precision issues
  - But it might also increase the number of pmts with low charges taken into consideration by the model, which isn’t what we want

- It seems working for loss with QT correlation but still not for loss with charge only