ILC Weekly Meeting

05.16.2024

Who am I?

- Paul Wahlen
- Bachelor at EPFL in Switzerland
- M1 student in joint degree given by ETH Zürich and Ecole Polytechnique de Paris, specialising in High Energy Physics

Since beginning

- Learning theoretical and practical fundamentals of ML
- Catching up on Python OOP, Pytorch and awkward libraries

Then:

- Learning about Transformers and familiarising with project
- Starting to write some code

Project concept

Using a Transformer to predict to which cluster a particular hit belongs to.

	Sequence to Sequence	Physics
Input	Sentence	List of hits from 1 event
Output	Machine translation of Seq	List of clusters to which belongs the hits
token	Depends, words/ few char.	1 hit
Special tokens	bos, eos, unkwn, pad	bos, eos, sample, pad

No fxed sized vocabulary due to continous variables in each hit

Need to impose a maximum number of clusters, since fixed number of parameters

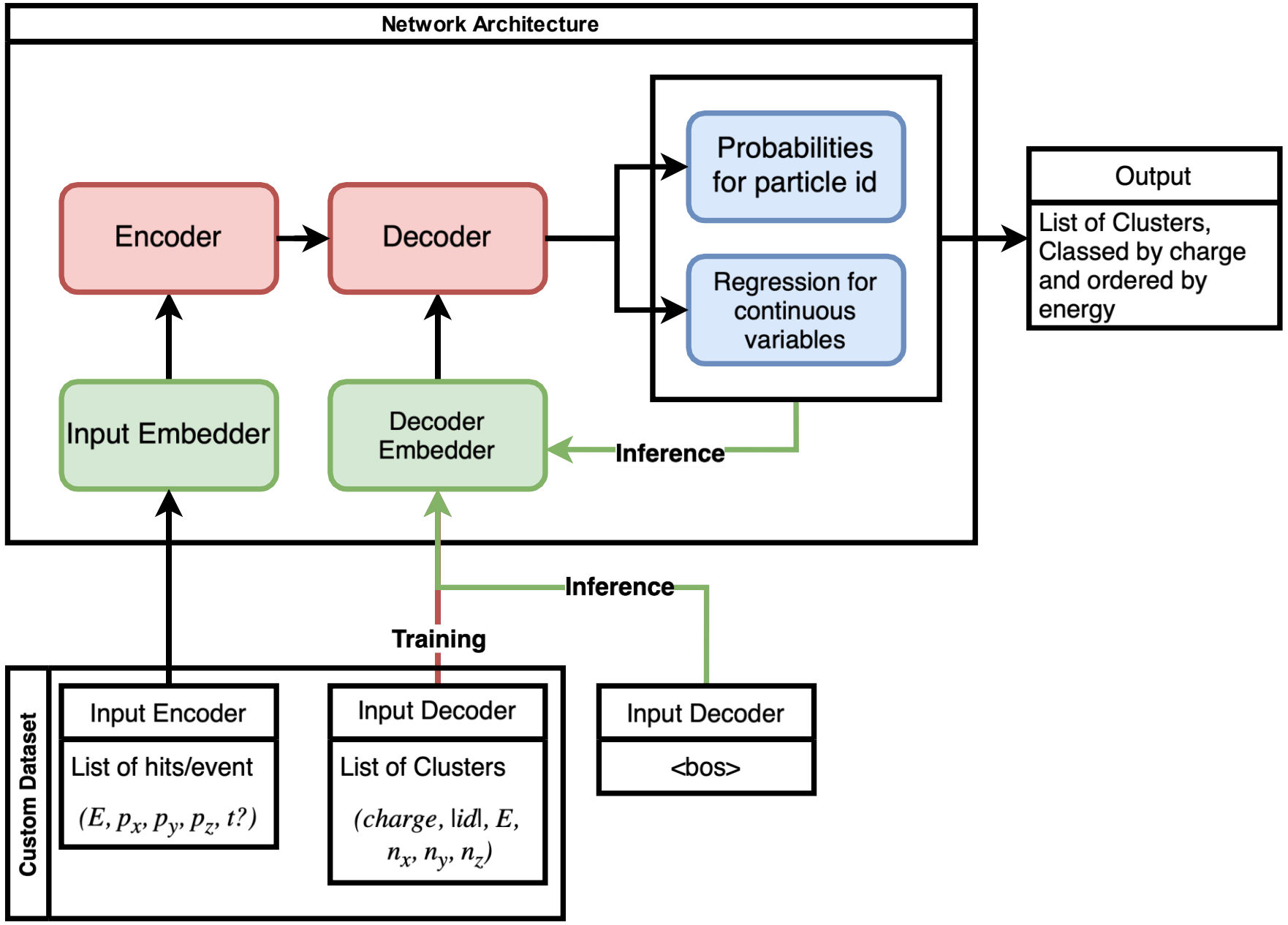
Architecture: Entire Network

Cluster information are obtained from MC Particle truth information.

2 loss functions, weighted by hyperparameters:

- Most common particle ids hot ones encoded:
  $\gamma, K_s, K_L, K^+, \mu^-, p, n, \pi^\pm, e^-$
  + antiparticles.
  Softmax then CrossEntropy for the loss function

- Continuous variables are obtained by regression. MSE for the second loss function.



Dataset

- Select the wanted features from the awkward arrays and store them
- If do_tracks is true: also puts in the information of the tracks
- If do_time is true: add the time as feature.

Still to do:

- Shrink Labels to one representative of each cluster
- Compute energy + normalising the cluster momentum and energy

```python
class CollectionHits(Dataset):
    '''params:
            dir_path: string path of directory where data is stored
            do_tracks: bool. If true, tracks are stored in the Dataset
            do_time: bool, if True, time of hits is kept'''
    def __init__(self, dir_path: str, do_tracks: bool = False, do_time: bool = False):
        super(CollectionHits,self).__init__()
        filenames = list(sorted(glob.iglob(dir_path + '/*.h5')))
        if len(filenames) == 1:
            feats, labels = la.load_awkward2(filenames) #get the events from the only file
        elif len(filenames) > 1:
            feats, labels = la.load_awkwards(filenames) #get the events from each file
        else:
            raise ValueError(f"There is no h5py file in the directory {dir_path}")

        #removing tracks
        if do_tracks is False:
            tracks_mask = (feats[:,:,6] == 1)
            hits_mask = (tracks_mask != True)
            feats = feats[hits_mask]
            labels = labels[hits_mask]
        #keeping time
        if do_time:
            feats = feats[:,:,:5]
        else:
            feats = feats[:,:,:4]

        self.formatting(feats, labels)
```

Formatting dataset and batching

```python
def formatting(self, feats, labels):
    add_special_symbols = AddSpecialSymbols()
    self.feats = torch.from_numpy(ak.to_numpy(add_special_symbols(feats)))
    self.labels = torch.from_numpy(ak.to_numpy(add_special_symbols(labels)))
    self.feats[:,:,0] = np.tanh(self.feats[:,:,0])
    self.feats[:,:,1:3] /= 2000.
```

```python
class AddSpecialSymbols(object):
    def __init__(self):
        self.special_symbols = {
            "bos": [1.,1.],
            "eos": [1.,0.],
            "pad": [0.,1.],
            "sample": [0.,0.]
    }

    def __call__(self, data):
        #First adding (0,0) indicating hit/MC data at the end of features
        ones = np.array(self.special_symbols["sample"])[np.newaxis][np.newaxis]
        feat_augmented = ak.concatenate((data,ones), axis = -1)
        #Adding bos and eos at beginning and end of each event
        nfeats = int(ak.num(data, axis = -1)[0,0])
        bos = ak.Array(([0] * nfeats + self.special_symbols["bos"]))[np.newaxis]
        eos = ak.Array(([0] * nfeats + self.special_symbols["eos"]))[np.newaxis]
        feat_augmented = ak.concatenate([bos[np.newaxis], feat_augmented, eos[np.newaxis]], axis = 1)
        #Padding
        nsample_max_event = int(ak.max(ak.num(data,axis = 1))) #max number of samples in the batch
        feat_padded = ak.pad_none(feat_augmented,target = nsample_max_event, clip = True, axis = 1)
        pad = ak.Array([0]*nfeats + self.special_symbols["pad"])
        return ak.fill_none(feat_padded, value = pad, axis = None)
```

- Addition of a formatting method to the Dataset
- Callable object from a new class to do the formatting

Advantage + Flow:

1. Add the special tokens
2. Converts awk. Arrays to torch array (maybe only numpy is sufficient?), to be able to make inplace normalisation
3. Inplace normalisation.

Input/Label Embedders

Input:

list of hits from one event, features:
$(E, p_x \ \ p_y, p_z, t\ ?)$

Labels:

MC Truth particle cluster information, features:

$(charge, |pdg|, E, \hat{p}_x, \hat{p}_y, \hat{p}_z)$

```
class Embedder(nn.Module):
    def __init__(self, nlayers,d_input ,d_model, act_func = nn.ReLU()):
        super().__init__()
        linear = nn.Linear(d_input,d_model)
        sequence_module = OrderedDict([("input_layer", linear)])
        sequence_module.update([("hidden_actfun1", act_func)])
        for i in range(1, nlayers):
            linear = nn.Linear(d_input,d_model) #otherwise shares same parameters
            sequence_module.update([("hidden_actfun%d"%i, act_func)])
            sequence_module.update([("hidden_linear%d"%i, linear)])

        self.model = nn.Sequential(sequence_module)

    def forward(self,src):
        return self.model(src)
```

Main Neural Network

```
class ClustersFinder(nn.Module):
    def __init__(self, nclusters_max,dmodel, nhead, nhid_ff_trsf, nlayers_encoder,
                                                    nlayers_decoder,nlayers_embder, d_input):

        super(ClustersFinder,self).__init__()
        self.input_embedder = Embedder(nlayers = nlayers_embder, d_input=d_input,d_model=dmodel)
        self.tgt_embedder = Embedder(nlayers = nlayers_embder, d_input=d_input,d_model=dmodel)
        self.transformer = nn.Transformer(d_model=dmodel, nhead = nhead, dim_feedforward= nhid_ff_trsf,
                                          num_encoder_layers=nlayers_encoder, num_decoder_layers=nlayers_decoder)
        self.lastlin = nn.Linear(dmodel,nclusters_max)

    #forward will be called when the __call__ function of nn.Module will be called.
    def forward(self, src, tgt, src_padding_mask, tgt_padding_mask, memory_padding_mask):
        src = self.input_embedder(src)
        tgt = self.tgt_embedder(tgt)
        output = self.transformer(src = src,tgt = tgt,
                                  src_key_padding_mask = src_padding_mask,
                                  tgt_key_padding_mask = tgt_padding_mask,
                                  memory_key_padding_mask = memory_padding_mask)
        output = self.lastlin(output)
        return nn.functional.softmax(output, dim = -1)
```

Train and validation functions (1 epoch)

```python
def train(model, optim,loss_fn, train_dl, special_symbols):
    model.train() #setting model into train mode
    loss_epoch = 0.0
    for src,tgt in train_dl:
        src.to(DEVICE)
        tgt.to(DEVICE)

        src_padding_mask, tgt_padding_mask = create_mask(src,tgt,special_symbols["pad"])

        logits = model(src,tgt, src_padding_mask,tgt_padding_mask,src_padding_mask)
        optim.zero_grad()
        loss = loss_fn(logits.reshape(-1,logits.shape[-1]), tgt.reshape(-1))
        loss.backward()
        optim.step()

        loss_epoch += loss.item()

    return loss_epoch / len(list(train_dl))

def validate(model,loss_fn, val_dl, special_symbols):
    model.eval() #setting model into validation mode
    for src,tgt in val_dl:
        src.to(DEVICE)
        tgt.to(DEVICE)

        src_padding_mask, tgt_padding_mask = create_mask(src,tgt,special_symbols["pad"])

        logits = model(src,tgt, src_padding_mask,tgt_padding_mask,src_padding_mask)
        loss = loss_fn(logits.reshape(-1,logits.shape[-1]), tgt.reshape(-1))
        loss_epoch += loss.item()

    return loss_epoch / len(list(train_dl))
```

TODOS:

- Coding and implementing masks for transformer, batch mask?
- Finish train and validate function
- Start coding inference function