import torch

import torch.nn as nn

class MLPEmbedding(nn.Module):

def \_\_init\_\_(self, n\_users, n\_beers, device, hidden\_size=100):

super().\_\_init\_\_()

self.users\_emb = nn.Embedding(n\_users, hidden\_size).to(device)

self.beers\_emb = nn.Embedding(n\_beers, hidden\_size).to(device)

self.mlp = nn.Sequential(

nn.Linear(hidden\_size\*2, 200),

nn.BatchNorm1d(200),

nn.ReLU(),

nn.Dropout(0.6),

nn.Linear(200, 100),

nn.BatchNorm1d(100),

nn.ReLU(),

nn.Dropout(0.6),

nn.Linear(100, 50),

nn.BatchNorm1d(50),

nn.ReLU(),

nn.Dropout(0.6),

nn.Linear(50, 25),

nn.BatchNorm1d(25),

nn.ReLU(),

nn.Dropout(0.6)

).to(device)

self.last\_layer = nn.Linear(25,1)

self.device = device

def forward(self, df):

user\_idx = torch.LongTensor(df.user\_id.to\_numpy()).to(self.device)

beer\_idx = torch.LongTensor(df.beer\_id.to\_numpy()).to(self.device)

input = torch.cat((self.users\_emb(user\_idx), self.beers\_emb(beer\_idx)), 1).to(self.device)

out = self.mlp(input)

return out

def predict(self, df):

return torch.sigmoid(self.last\_layer(self.forward(df)))

def loss(self, train\_data, loss\_fn):

y\_pred = self.last\_layer(self.forward(train\_data)).view(-1)

y\_train = torch.Tensor(train\_data.relevant.to\_numpy()).to(self.device)

return loss\_fn(y\_pred, y\_train)

class GMF(nn.Module):

def \_\_init\_\_(self, n\_users, n\_items, n\_factors=5):

super().\_\_init\_\_()

self.user\_emb = nn.Embedding(n\_users,n\_factors)

self.item\_emb = nn.Embedding(n\_items,n\_factors)

self.h\_out = nn.Linear(n\_factors,1)

def forward(self, user, item):

p = self.user\_emb(user)

q = self.item\_emb(item)

return torch.flatten(self.h\_out(p\*q))

def predict(self, user, item):

return torch.sigmoid(self.forward(user, item))

def forward\_no\_h(self, df):

user\_id = Variable(torch.LongTensor(df.user\_id.to\_numpy()))

beer\_id = Variable(torch.LongTensor(df.beer\_id.to\_numpy()))

p = self.user\_emb(user\_id)

q = self.item\_emb(beer\_id)

return p\*q