import sys

sys.path.append('..')

import fewshot\_re\_kit

import torch

from torch import autograd, optim, nn

from torch.autograd import Variable

from torch.nn import functional as F

class Proto(fewshot\_re\_kit.framework.FewShotREModel):

def \_\_init\_\_(self, sentence\_encoder, dot=False):

fewshot\_re\_kit.framework.FewShotREModel.\_\_init\_\_(self, sentence\_encoder)

# self.fc = nn.Linear(hidden\_size, hidden\_size)

self.drop = nn.Dropout()

self.dot = dot

def \_\_dist\_\_(self, x, y, dim):

if self.dot:

return (x \* y).sum(dim)

else:

return -(torch.pow(x - y, 2)).sum(dim)

def \_\_batch\_dist\_\_(self, S, Q):

return self.\_\_dist\_\_(S.unsqueeze(1), Q.unsqueeze(2), 3)

def forward(self, support, query, N, K, total\_Q):

'''

support: Inputs of the support set.

query: Inputs of the query set.

N: Num of classes

K: Num of instances for each class in the support set

Q: Num of instances in the query set

'''

support\_emb = self.sentence\_encoder(support) # (B \* N \* K, D), where D is the hidden size

query\_emb = self.sentence\_encoder(query) # (B \* total\_Q, D)

hidden\_size = support\_emb.size(-1)

support = self.drop(support\_emb)

query = self.drop(query\_emb)

support = support.view(-1, N, K, hidden\_size) # (B, N, K, D)

query = query.view(-1, total\_Q, hidden\_size) # (B, total\_Q, D)

# Prototypical Networks

# Ignore NA policy

support = torch.mean(support, 2) # Calculate prototype for each class

logits = self.\_\_batch\_dist\_\_(support, query) # (B, total\_Q, N)

minn, \_ = logits.min(-1)

logits = torch.cat([logits, minn.unsqueeze(2) - 1], 2) # (B, total\_Q, N + 1)

\_, pred = torch.max(logits.view(-1, N + 1), 1)

return logits, pred