from meantime.loggers import \*

# from config import STATE\_DICT\_KEY, OPTIMIZER\_STATE\_DICT\_KEY, TRAIN\_LOADER\_RNG\_STATE\_DICT\_KEY

from meantime.config import \*

from meantime.utils import AverageMeterSet

from meantime.utils import fix\_random\_seed\_as

from meantime.analyze\_table import find\_saturation\_point

import torch

import torch.nn as nn

import torch.optim as optim

from tqdm import tqdm

import pandas as pd

from abc import \*

from pathlib import Path

import os

import numpy as np

import time

class AbstractTrainer(metaclass=ABCMeta):

def \_\_init\_\_(self, args, model, train\_loader, val\_loader, test\_loader, local\_export\_root):

print(' ')

print('meantime / trainers / base.py / AbstractTrainer.\_\_init\_\_')

self.args = args

# My Code Start

self.input\_middle\_seq = args.input\_middle\_seq

self.input\_middle\_num = args.input\_middle\_num

self.input\_middle\_target = args.input\_middle\_target

self.input\_future\_seq = args.input\_future\_seq

# My Code End

self.device = args.device

self.model = model.to(self.device)

self.use\_parallel = args.use\_parallel

if self.use\_parallel:

self.model = nn.DataParallel(self.model)

self.train\_loader = train\_loader

self.val\_loader = val\_loader

self.test\_loader = test\_loader

self.optimizer = self.\_create\_optimizer()

self.lr\_scheduler = optim.lr\_scheduler.StepLR(self.optimizer, step\_size=args.decay\_step, gamma=args.gamma)

self.clip\_grad\_norm = args.clip\_grad\_norm

self.epoch\_start = 0

self.best\_epoch = self.epoch\_start - 1

self.best\_metric\_at\_best\_epoch = -1

self.accum\_iter\_start = 0

self.num\_epochs = args.num\_epochs

if self.num\_epochs == -1:

self.num\_epochs = 987654321 # Technically Infinite

self.metric\_ks = args.metric\_ks

self.best\_metric = args.best\_metric

self.saturation\_wait\_epochs = args.saturation\_wait\_epochs

self.pilot = args.pilot

if self.pilot:

self.num\_epochs = 1

self.pilot\_batch\_cnt = 1

self.local\_export\_root = local\_export\_root

self.train\_loggers, self.val\_loggers, self.test\_loggers = self.\_create\_loggers() if not self.pilot else (None, None, None)

self.add\_extra\_loggers()

self.logger\_service = LoggerService(args, self.train\_loggers, self.val\_loggers, self.test\_loggers)

self.log\_period\_as\_iter = args.log\_period\_as\_iter

self.resume\_training = args.resume\_training

if self.resume\_training:

print('Restoring previous training state')

self.\_restore\_training\_state()

print('Finished restoring')

@abstractmethod

def add\_extra\_loggers(self):

pass

@abstractmethod

def log\_extra\_train\_info(self, log\_data):

pass

@classmethod

@abstractmethod

def code(cls):

pass

@abstractmethod

def calculate\_loss(self, batch):

pass

@abstractmethod

def calculate\_metrics(self, batch):

pass

def train(self):

print(' ')

print('meantime / trainers / base.py / AbstractTrainer.train')

epoch = self.epoch\_start

best\_epoch = self.best\_epoch

accum\_iter = self.accum\_iter\_start

# self.validate(epoch-1, accum\_iter, self.val\_loader)

best\_metric = self.best\_metric\_at\_best\_epoch

stop\_training = False

for epoch in range(self.epoch\_start, self.num\_epochs):

if self.pilot:

print('epoch', epoch)

fix\_random\_seed\_as(epoch) # fix random seed at every epoch to make it perfectly resumable

accum\_iter = self.train\_one\_epoch(epoch, accum\_iter, self.train\_loader)

print('meantime / trainers / base.py self.train\_loader is')

#print(self.train\_loader)

print('meantime / trainers / base.py train.accum\_iter is')

#print(accum\_iter)

self.lr\_scheduler.step() # step before val because state\_dict is saved at val. it doesn't affect val result

val\_log\_data = self.validate(epoch, accum\_iter, mode='val')

metric = val\_log\_data[self.best\_metric]

if metric > best\_metric:

best\_metric = metric

best\_epoch = epoch

elif (self.saturation\_wait\_epochs is not None) and\

(epoch - best\_epoch >= self.saturation\_wait\_epochs):

stop\_training = True # stop training if val perf doesn't improve for saturation\_wait\_epochs

if stop\_training:

# load best model

best\_model\_logger = self.val\_loggers[-1]

assert isinstance(best\_model\_logger, BestModelLogger)

weight\_path = best\_model\_logger.filepath()

if self.use\_parallel:

self.model.module.load(weight\_path)

else:

self.model.load(weight\_path)

# self.validate(epoch, accum\_iter, mode='test') # test result at best model

self.validate(best\_epoch, accum\_iter, mode='test') # test result at best model

break

self.logger\_service.complete({

'state\_dict': (self.\_create\_state\_dict(epoch, accum\_iter)),

})

def just\_validate(self, mode):

print(' ')

print('meantime / trainers / base.py / AbstractTrainer.just\_validate')

### My Code Start###

if mode == 'fake\_test':

dummy\_epoch, dummy\_accum\_iter = 0, 0

self.fake\_test(dummy\_epoch, dummy\_accum\_iter, mode)

elif mode == 'tobigs\_test':

dummy\_epoch, dummy\_accum\_iter = 0, 0

self.tobigs\_test(dummy\_epoch, dummy\_accum\_iter, mode)

else:

dummy\_epoch, dummy\_accum\_iter = 0,0

self.validate(dummy\_epoch, dummy\_accum\_iter, mode)

### My Code End

def train\_one\_epoch(self, epoch, accum\_iter, train\_loader, \*\*kwargs):

print(' ')

print('meantime / trainers / base.py / AbstractTrainer.train\_one\_epoch')

#print(' what is meantime / trainers / base.py / AbstractTrainer, train\_loader? is')

#print(train\_loader)

self.model.train()

average\_meter\_set = AverageMeterSet()

num\_instance = 0

tqdm\_dataloader = tqdm(train\_loader) if not self.pilot else train\_loader

for batch\_idx, batch in enumerate(tqdm\_dataloader):

if self.pilot and batch\_idx >= self.pilot\_batch\_cnt:

# print('Break training due to pilot mode')

break

print(' ')

print(' ###')

print(' ')

print(' raw batch file in training is')

print(' ')

#print(batch)

batch\_size = next(iter(batch.values())).size(0)

batch = {k:v.to(self.device) for k, v in batch.items()}

num\_instance += batch\_size

print(' ')

print(' ')

print('###############################################')

print(' meantime / trainers / base.py / AbstractTrainer.train\_one\_epoch / batch\_idx is')

#print(batch\_idx)

print(' ')

print(' meantime / trainers / base.py / AbstractTrainer.train\_one\_epoch / batch is')

print(' ')

print('batch')

#print(batch)

print(' ')

#print("batch['logits'].shape is")

#print(batch['logits'].shape)

print(' meantime / trainers / base.py / AbstractTrainer.train\_one\_epoch / self.optimizer.zero\_grad()')

self.optimizer.zero\_grad()

print(' ')

print(' meantime / trainers / base.py / AbstractTrainer.train\_one\_epoch / self.calculate\_loss(batch)')

loss = self.calculate\_loss(batch) # 여기로 들어가서 train 과정 자세히 봐야겠다.

if isinstance(loss, tuple):

loss, extra\_info = loss

for k, v in extra\_info.items():

average\_meter\_set.update(k, v)

loss.backward()

if self.clip\_grad\_norm is not None:

torch.nn.utils.clip\_grad\_norm\_(self.model.parameters(), self.clip\_grad\_norm)

self.optimizer.step()

average\_meter\_set.update('loss', loss.item())

if not self.pilot:

tqdm\_dataloader.set\_description(

'Epoch {}, loss {:.3f} '.format(epoch, average\_meter\_set['loss'].avg))

accum\_iter += batch\_size

if self.\_needs\_to\_log(accum\_iter):

if not self.pilot:

tqdm\_dataloader.set\_description('Logging')

log\_data = {

# 'state\_dict': (self.\_create\_state\_dict()),

'epoch': epoch,

'accum\_iter': accum\_iter,

}

log\_data.update(average\_meter\_set.averages())

log\_data.update(kwargs)

self.log\_extra\_train\_info(log\_data)

self.logger\_service.log\_train(log\_data)

log\_data = {

# 'state\_dict': (self.\_create\_state\_dict()),

'epoch': epoch,

'accum\_iter': accum\_iter,

'num\_train\_instance': num\_instance,

}

log\_data.update(average\_meter\_set.averages())

log\_data.update(kwargs)

self.log\_extra\_train\_info(log\_data)

self.logger\_service.log\_train(log\_data)

return accum\_iter

def validate(self, epoch, accum\_iter, mode, doLog=True, \*\*kwargs):

print(' ')

print('meantime / trainers / base.py / AbstractTrainer.validate is')

### My Code Start###

my\_final\_result = -1\*torch.ones(1, 205)

my\_dtype = torch.cuda.FloatTensor if torch.cuda.is\_available() else torch.FloatTensor

my\_final\_result = my\_final\_result.to(self.device)

### My Code End###

if mode == 'val':

loader = self.val\_loader

elif mode == 'test':

loader = self.test\_loader

else:

raise ValueError

self.model.eval()

average\_meter\_set = AverageMeterSet()

num\_instance = 0

with torch.no\_grad():

tqdm\_dataloader = tqdm(loader) if not self.pilot else loader

for batch\_idx, batch in enumerate(tqdm\_dataloader):

if self.pilot and batch\_idx >= self.pilot\_batch\_cnt:

# print('Break validation due to pilot mode')

break

batch = {k:v.to(self.device) for k, v in batch.items()}

batch\_size = next(iter(batch.values())).size(0)

num\_instance += batch\_size

metrics = self.calculate\_metrics(batch)

'''

print(' ')

print(' ')

print('batch idx is')

print(batch\_idx)

print('batch : token, [Batch\_size x seq\_len]')

print(batch['tokens'])

print('batch : candidate, [Batch\_size x 100\_negative\_samples is]')

print(batch['candidates'])

print('batch : labels, [Batch\_size x (1 + 100)\_labels is]')

print(batch['labels'])

###### MY CODE ######

#print('epoch is') # 20201214

#print(epoch)

#print('batch is') ##### My code 20201119

#print(batch)

#print('true answer is')

#print(batch['candidates'][:,0])

MY\_SCORES, MY\_LABELS, MY\_CUT, MY\_HITS = self.NEW\_CODE\_PRINT\_PREDICTION(batch) ##### My code 20201119

my\_len = len(MY\_CUT)

print("MY\_SCORES is, [Batch\_size x (1 + 100)]")

print(MY\_SCORES) ##### My code 20201119

print(' ')

#print("MY\_LABELS")

#print(MY\_LABELS) ##### My code 20201119

print("MY\_CUT(prediction) is, [Batch\_size x 1]")

print(MY\_CUT) ##### My code 20201119

print(' ')

print("MY\_HITS is, [Batch\_size x 1]")

print(MY\_HITS) ##### My code 20201119

print(' ')

#print('MY\_SCORES shape')

#print(MY\_SCORES.shape)

#print(' ')

#print('MY\_LABELS shape')

#print(MY\_LABELS.shape)

#print(' ')

#print('MY\_CUT shape')

#print(MY\_CUT.shape)

#print('MY\_HITS.shape')

#print(MY\_HITS.shape)

'''

#my\_epoch = epoch

#my\_batch\_idx = batch\_idx

#my\_batch\_token = batch['tokens']

#my\_batch\_candidate = batch['candidates']

#my\_batch\_score = MY\_SCORES

#my\_batch\_cut = MY\_CUT

#my\_hit = MY\_HITS

#print('true answer is')

#print(batch['candidates'][:,0])

#my\_epoch1 = torch.Tensor([my\_epoch]\*batch\_size).reshape(batch\_size,1)

#batch\_idx1 = torch.Tensor([my\_batch\_idx]\*batch\_size).reshape(batch\_size,1)

#batch\_idx2 = torch.Tensor(range(batch\_size)).reshape(batch\_size,1)

#my\_batch\_token = my\_batch\_token.to(self.device)

#my\_candi = batch['candidates'][:,0]

#my\_candi = my\_candi.to(self.device)

#my\_cut = MY\_CUT

#my\_cut = my\_cut.to(self.device)

#my\_epoch1 = my\_epoch1.type(my\_dtype)

#batch\_idx1 = batch\_idx1.type(my\_dtype)

#batch\_idx2 = batch\_idx2.type(my\_dtype)

#my\_batch\_token = my\_batch\_token.type(my\_dtype)

#my\_candi = my\_candi.type(my\_dtype).reshape(batch\_size,1)

#my\_hit = my\_hit.type(my\_dtype)

#my\_cut = my\_cut.type(my\_dtype)

#print('###')

#print('my batch token shape')

#print(my\_batch\_token.shape)

#print(my\_candi.shape)

#print(my\_hit.shape)

#print('batch\_idx1')

#print(batch\_idx1)

#print(batch\_idx2)

#print('my\_epoch')

#print(my\_epoch)

#my\_epoch\_result = torch.cat([my\_epoch1, batch\_idx1, batch\_idx2, my\_batch\_token, my\_candi, my\_cut], 1)

#my\_final\_result = torch.cat([my\_final\_result, my\_epoch\_result], 0)

###### MY CODE ######

for k, v in metrics.items():

average\_meter\_set.update(k, v)

if not self.pilot:

description\_metrics = ['NDCG@%d' % k for k in self.metric\_ks[:3]] +\

['Recall@%d' % k for k in self.metric\_ks[:3]]

description = '{}: '.format(mode.capitalize()) + ', '.join(s + ' {:.3f}' for s in description\_metrics)

description = description.replace('NDCG', 'N').replace('Recall', 'R')

description = description.format(\*(average\_meter\_set[k].avg for k in description\_metrics))

tqdm\_dataloader.set\_description(description)

log\_data = {

'state\_dict': (self.\_create\_state\_dict(epoch, accum\_iter)),

'epoch': epoch,

'accum\_iter': accum\_iter,

'num\_eval\_instance': num\_instance,

}

log\_data.update(average\_meter\_set.averages())

log\_data.update(kwargs)

if doLog:

if mode == 'val':

self.logger\_service.log\_val(log\_data)

elif mode == 'test':

self.logger\_service.log\_test(log\_data)

else:

raise ValueError

###### MY CODE ######

#ts = time.time()

#my\_final\_result = my\_final\_result.cpu()

#my\_final\_result\_np = my\_final\_result.numpy()

#my\_final\_result\_df = pd.DataFrame(my\_final\_result\_np)

#FILENAME = 'my\_final\_result' + mode + str(epoch) + 'time' + str(ts) + '\_' + '.csv'

#my\_final\_result\_df.to\_csv(FILENAME)

###### MY CODE ######

return log\_data

def \_create\_optimizer(self):

args = self.args

if args.optimizer.lower() == 'adam':

return optim.Adam(self.model.parameters(), lr=args.lr, weight\_decay=args.weight\_decay)

elif args.optimizer.lower() == 'sgd':

return optim.SGD(self.model.parameters(), lr=args.lr, weight\_decay=args.weight\_decay, momentum=args.momentum)

else:

raise ValueError

def \_create\_loggers(self):

train\_table\_definitions = [

('train\_log', ['epoch', 'loss'])

]

val\_table\_definitions = [

('val\_log', ['epoch'] + \

['NDCG@%d' % k for k in self.metric\_ks] +

['Recall@%d' % k for k in self.metric\_ks]),

]

test\_table\_definitions = [

('test\_log', ['epoch'] + \

['NDCG@%d' % k for k in self.metric\_ks] +

['Recall@%d' % k for k in self.metric\_ks]),

]

train\_loggers = [TableLoggersManager(args=self.args, export\_root=self.local\_export\_root, table\_definitions=train\_table\_definitions)]

val\_loggers = [TableLoggersManager(args=self.args, export\_root=self.local\_export\_root, table\_definitions=val\_table\_definitions)]

test\_loggers = [TableLoggersManager(args=self.args, export\_root=self.local\_export\_root, table\_definitions=test\_table\_definitions)]

if self.local\_export\_root is not None:

root = Path(self.local\_export\_root)

model\_checkpoint = root.joinpath('models')

val\_loggers.append(RecentModelLogger(model\_checkpoint))

val\_loggers.append(BestModelLogger(model\_checkpoint, metric\_key=self.best\_metric))

if USE\_WANDB:

train\_loggers.append(WandbLogger(table\_definitions=train\_table\_definitions))

val\_loggers.append(WandbLogger(table\_definitions=val\_table\_definitions, prefix='val\_'))

test\_loggers.append(WandbLogger(table\_definitions=test\_table\_definitions, prefix='test\_'))

return train\_loggers, val\_loggers, test\_loggers

def \_create\_state\_dict(self, epoch, accum\_iter):

return {

STATE\_DICT\_KEY: self.model.module.state\_dict() if self.use\_parallel else self.model.state\_dict(),

OPTIMIZER\_STATE\_DICT\_KEY: self.optimizer.state\_dict(),

SCHEDULER\_STATE\_DICT\_KEY: self.lr\_scheduler.state\_dict(),

TRAIN\_LOADER\_DATASET\_RNG\_STATE\_DICT\_KEY: self.train\_loader.dataset.get\_rng\_state(),

TRAIN\_LOADER\_SAMPLER\_RNG\_STATE\_DICT\_KEY: self.train\_loader.sampler.get\_rng\_state(),

STEPS\_DICT\_KEY: (epoch, accum\_iter),

}

def \_restore\_best\_state(self):

### restore best epoch

df\_path = os.path.join(self.local\_export\_root, 'tables', 'val\_log.csv')

df = pd.read\_csv(df\_path)

sat, reached\_end = find\_saturation\_point(df, self.saturation\_wait\_epochs, display=False)

e = sat['epoch'].iloc[0]

self.best\_epoch = e

print('Restored best epoch:', self.best\_epoch)

###

state\_dict\_path = os.path.join(self.local\_export\_root, 'models', BEST\_STATE\_DICT\_FILENAME)

chk\_dict = torch.load(os.path.abspath(state\_dict\_path))

### sanity check

\_e, \_ = chk\_dict[STEPS\_DICT\_KEY]

assert e == \_e

### load weights

d = chk\_dict[STATE\_DICT\_KEY]

model\_state\_dict = {}

# this is for stupid reason

for k, v in d.items():

if k.startswith('model.'):

model\_state\_dict[k[6:]] = v

else:

model\_state\_dict[k] = v

if self.use\_parallel:

self.model.module.load\_state\_dict(model\_state\_dict)

else:

self.model.load\_state\_dict(model\_state\_dict)

### restore best metric

val\_log\_data = self.validate(0, 0, mode='val', doLog=False)

metric = val\_log\_data[self.best\_metric]

self.best\_metric\_at\_best\_epoch = metric

print('Restored best metric:', self.best\_metric\_at\_best\_epoch)

def \_restore\_training\_state(self):

self.\_restore\_best\_state()

###

state\_dict\_path = os.path.join(self.local\_export\_root, 'models', RECENT\_STATE\_DICT\_FILENAME)

chk\_dict = torch.load(os.path.abspath(state\_dict\_path))

### restore epoch, accum\_iter

epoch, accum\_iter = chk\_dict[STEPS\_DICT\_KEY]

self.epoch\_start = epoch + 1

self.accum\_iter\_start = accum\_iter

### restore train dataloader rngs

train\_loader\_dataset\_rng\_state = chk\_dict[TRAIN\_LOADER\_DATASET\_RNG\_STATE\_DICT\_KEY]

self.train\_loader.dataset.set\_rng\_state(train\_loader\_dataset\_rng\_state)

train\_loader\_sampler\_rng\_state = chk\_dict[TRAIN\_LOADER\_SAMPLER\_RNG\_STATE\_DICT\_KEY]

self.train\_loader.sampler.set\_rng\_state(train\_loader\_sampler\_rng\_state)

### restore model

d = chk\_dict[STATE\_DICT\_KEY]

model\_state\_dict = {}

# this is for stupid reason

for k, v in d.items():

if k.startswith('model.'):

model\_state\_dict[k[6:]] = v

else:

model\_state\_dict[k] = v

if self.use\_parallel:

self.model.module.load\_state\_dict(model\_state\_dict)

else:

self.model.load\_state\_dict(model\_state\_dict)

### restore optimizer

self.optimizer.load\_state\_dict(chk\_dict[OPTIMIZER\_STATE\_DICT\_KEY])

### restore lr\_scheduler

self.lr\_scheduler.load\_state\_dict(chk\_dict[SCHEDULER\_STATE\_DICT\_KEY])

def \_needs\_to\_log(self, accum\_iter):

return accum\_iter % self.log\_period\_as\_iter < self.args.train\_batch\_size and accum\_iter != 0

def tobigs\_test(self, epoch, accum\_iter, mode, doLog=True, \*\*kwargs):

if mode == 'val':

loader = self.val\_loader

elif mode == 'test':

loader = self.test\_loader

elif mode == 'tobigs\_test':

print('tobigs test')

import pickle

import sys, os

with open('dataset.pkl', 'rb') as fp:

data\_new = pickle.load(fp)

self.model.eval()

average\_meter\_set = AverageMeterSet()

num\_instance = 0

input\_middle\_seq = self.input\_middle\_seq

input\_middle\_num = self.input\_middle\_num

input\_middle\_target = self.input\_middle\_target

input\_future\_seq = self.input\_future\_seq

if input\_middle\_seq != None:

prediction\_mode = 'middle'

input\_middle\_num = int(input\_middle\_num[0])

elif input\_future\_seq != None:

prediction\_mode = 'future'

item\_true\_name\_dict = data\_new['smap']

inv\_map = {v: k for k, v in item\_true\_name\_dict.items()}

if prediction\_mode == 'middle':

print('##################')

print(' ')

print('##################')

print(' ')

print('##################')

print(' ')

print('input\_middle\_seq is')

print(' ')

print(input\_middle\_seq)

print(' ')

print('input\_middle\_target is')

print(' ')

print(input\_middle\_target)

new\_input\_seq = [item\_true\_name\_dict[name] for name in input\_middle\_seq]

new\_input\_target = [item\_true\_name\_dict[name] for name in input\_middle\_target]

input\_middle\_seq = list(map(int, new\_input\_seq))

input\_middle\_target = list(map(int, new\_input\_target))

pred\_seq\_len = len(input\_middle\_seq)

dummy\_tokens = torch.cat([

torch.tensor( [0]\*(12 - (pred\_seq\_len+2)) ),

torch.tensor(input\_middle\_seq),

torch.tensor([1692]),

torch.tensor(input\_middle\_target)])

dummy\_tokens = torch.cat([dummy\_tokens.reshape(1, 12), dummy\_tokens.reshape(1, 12)], dim = 0)

dummy\_labels = dummy\_tokens\*(dummy\_tokens == 1692)/1692

batch = {'tokens' : dummy\_tokens, 'labels' : dummy\_labels}

with torch.no\_grad():

test\_assign = self.calculate\_loss2(batch)

valid\_index, valid\_scores, valid\_labels = test\_assign

minimini = valid\_scores.min()

valid\_scores = valid\_scores + abs(minimini) + 0.01

scores\_all = valid\_scores

item\_type\_pickle = pd.read\_pickle('type\_dict.pkl')

career\_type\_list = [k for k, v in item\_type\_pickle.items() if v in ['중소기업', '스타트업', '대기업']]

tmp = torch.zeros(1692, requires\_grad=False).cuda()

for i in range(1, 1692):

if i in career\_type\_list:

tmp[i-1] = 1

else:

tmp[i-1] = 0

tmp = tmp.reshape(1, 1692)

tmp2 = torch.cat([tmp, tmp], dim=0)

valid\_scores = torch.mul(valid\_scores, tmp2)

tokens\_name1 = []

for key in dummy\_tokens[0]:

key = key.cpu().detach().numpy()

key = int(key)

if key == 0:

tokens\_name1.append('blank')

elif key == 1692:

tokens\_name1.append('#MASK#')

else:

tokens\_name1.append(inv\_map[key])

tokens\_name2 = []

for key in dummy\_tokens[1]:

key = key.cpu().detach().numpy()

key = int(key)

if key == 0:

tokens\_name2.append('blank')

elif key == 1692:

tokens\_name2.append('#MASK#')

else:

tokens\_name2.append(inv\_map[key])

# Scores all

\_, my\_indices = torch.max(scores\_all, 1)

\_, my\_indices = torch.topk(scores\_all, k = input\_middle\_num, dim = 1)

my\_indices = my\_indices.cpu().detach().numpy()

my\_indices = my\_indices[0]

print('new my indices is')

print(' ')

print(my\_indices)

item\_true\_name\_dict = data\_new['smap']

inv\_map = {v: k for k, v in item\_true\_name\_dict.items()}

pred\_item = []

pred\_item\_type = []

pred\_item\_name = []

for key in my\_indices:

true\_key = key + 1

pred\_item.append(true\_key)

pred\_item\_type.append(item\_type\_pickle[true\_key])

pred\_item\_name.append(inv\_map[true\_key])

print(' ')

print(' ')

print('prediction softmax : all item is')

print(' ')

print(scores\_all)

print(' ')

print('shape is')

print(scores\_all.shape)

print(' ')

print('pred\_item(all) is')

print(' ')

print(pred\_item)

print(' ')

print('pred\_item : type is')

print(' ')

print(pred\_item\_type)

print(' ')

print('pred\_item : name is')

print(' ')

print(pred\_item\_name)

print(' ')

print('The End')

print(' ')

pred\_name = pred\_item\_name

with open('pred\_middle\_name.txt', 'w') as f:

for name in pred\_name:

f.write(name+'\n')

f.close()

return torch.max(valid\_scores, 1)

if prediction\_mode == 'future':

print('##################')

print(' ')

print('##################')

print(' ')

print('##################')

print(' ')

print('input\_future\_seq is')

print(' ')

print(input\_future\_seq)

new\_input\_seq = [item\_true\_name\_dict[name] for name in input\_future\_seq]

input\_future\_seq = list(map(int, new\_input\_seq))

pred\_seq\_len = len(input\_future\_seq)

dummy\_tokens = torch.cat([

torch.tensor( [0]\*(12 - (pred\_seq\_len+1)) ),

torch.tensor(input\_future\_seq),

torch.tensor([1692])])

dummy\_tokens = torch.cat([dummy\_tokens.reshape(1, 12), dummy\_tokens.reshape(1, 12)], dim = 0)

dummy\_labels = dummy\_tokens\*(dummy\_tokens == 1692)/1692

batch = {'tokens' : dummy\_tokens, 'labels' : dummy\_labels}

with torch.no\_grad():

test\_assign = self.calculate\_loss2(batch)

valid\_index, valid\_scores, valid\_labels = test\_assign

minimini = valid\_scores.min()

valid\_scores = valid\_scores + abs(minimini) + 0.01

scores\_all = valid\_scores

item\_type\_pickle = pd.read\_pickle('type\_dict.pkl')

career\_type\_list = [k for k, v in item\_type\_pickle.items() if v in ['중소기업', '스타트업', '대기업']]

tmp = torch.zeros(1692, requires\_grad=False).cuda()

for i in range(1, 1692):

if i in career\_type\_list:

tmp[i-1] = 1

else:

tmp[i-1] = 0

tmp = tmp.reshape(1, 1692)

tmp2 = torch.cat([tmp, tmp], dim=0)

valid\_scores = torch.mul(valid\_scores, tmp2)

tokens\_name1 = []

for key in dummy\_tokens[0]:

key = key.cpu().detach().numpy()

key = int(key)

if key == 0:

tokens\_name1.append('blank')

elif key == 1692:

tokens\_name1.append('#MASK#')

else:

tokens\_name1.append(inv\_map[key])

tokens\_name2 = []

for key in dummy\_tokens[1]:

key = key.cpu().detach().numpy()

key = int(key)

if key == 0:

tokens\_name2.append('blank')

elif key == 1692:

tokens\_name2.append('#MASK#')

else:

tokens\_name2.append(inv\_map[key])

print(tokens\_name1)

print(tokens\_name2)

print(' ')

print(dummy\_tokens)

print(' ')

print(dummy\_labels)

\_, my\_indices = torch.max(valid\_scores, 1)

my\_indices = my\_indices.cpu().detach().numpy()

pred\_item = []

pred\_item\_type = []

pred\_item\_name = []

for key in my\_indices:

true\_key = key + 1

pred\_item.append(true\_key)

pred\_item\_type.append(item\_type\_pickle[true\_key])

pred\_item\_name.append(inv\_map[true\_key])

print(' ')

print('pred\_item(career) is')

print(' ')

print(pred\_item[0])

print(' ')

print('pred\_item : type is')

print(' ')

print(pred\_item\_type[0])

pred\_type = pred\_item\_type[0]

print(' ')

print('pred\_item : name is')

print(' ')

print(pred\_item\_name[0])

pred\_name = pred\_item\_name[0]

with open('pred\_future\_name.txt', 'w') as f:

f.write(pred\_name)

f.close()

print(' ')

print('The End')

print(' ')

return torch.max(valid\_scores, 1)