from datetime import datetime

from functools import partial

import os

from twitter.cortex.ml.embeddings.common.helpers import decode\_str\_or\_unicode

import twml

from twml.trainers import DataRecordTrainer

from ..libs.get\_feat\_config import get\_feature\_config\_light\_ranking, LABELS\_LR

from ..libs.graph\_utils import get\_trainable\_variables

from ..libs.group\_metrics import (

run\_group\_metrics\_light\_ranking,

run\_group\_metrics\_light\_ranking\_in\_bq,

)

from ..libs.metric\_fn\_utils import get\_metric\_fn

from ..libs.model\_args import get\_arg\_parser\_light\_ranking

from ..libs.model\_utils import read\_config

from ..libs.warm\_start\_utils import get\_feature\_list\_for\_light\_ranking

from .model\_pools\_mlp import light\_ranking\_mlp\_ngbdt

import tensorflow.compat.v1 as tf

from tensorflow.compat.v1 import logging

# checkstyle: noqa

def build\_graph(

features, label, mode, params, config=None, run\_light\_ranking\_group\_metrics\_in\_bq=False

):

is\_training = mode == tf.estimator.ModeKeys.TRAIN

this\_model\_func = light\_ranking\_mlp\_ngbdt

model\_output = this\_model\_func(features, is\_training, params, label)

logits = model\_output["output"]

graph\_output = {}

# --------------------------------------------------------

# define graph output dict

# --------------------------------------------------------

if mode == tf.estimator.ModeKeys.PREDICT:

loss = None

output\_label = "prediction"

if params.task\_name in LABELS\_LR:

output = tf.nn.sigmoid(logits)

output = tf.clip\_by\_value(output, 0, 1)

if run\_light\_ranking\_group\_metrics\_in\_bq:

graph\_output["trace\_id"] = features["meta.trace\_id"]

graph\_output["target"] = features["meta.ranking.weighted\_oonc\_model\_score"]

else:

raise ValueError("Invalid Task Name !")

else:

output\_label = "output"

weights = tf.cast(features["weights"], dtype=tf.float32, name="RecordWeights")

if params.task\_name in LABELS\_LR:

if params.use\_record\_weight:

weights = tf.clip\_by\_value(

1.0 / (1.0 + weights + params.smooth\_weight), params.min\_record\_weight, 1.0

)

loss = tf.reduce\_sum(

tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=label, logits=logits) \* weights

) / (tf.reduce\_sum(weights))

else:

loss = tf.reduce\_mean(tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=label, logits=logits))

output = tf.nn.sigmoid(logits)

else:

raise ValueError("Invalid Task Name !")

train\_op = None

if mode == tf.estimator.ModeKeys.TRAIN:

# --------------------------------------------------------

# get train\_op

# --------------------------------------------------------

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=params.learning\_rate)

update\_ops = set(tf.get\_collection(tf.GraphKeys.UPDATE\_OPS))

variables = get\_trainable\_variables(

all\_trainable\_variables=tf.trainable\_variables(), trainable\_regexes=params.trainable\_regexes

)

with tf.control\_dependencies(update\_ops):

train\_op = twml.optimizers.optimize\_loss(

loss=loss,

variables=variables,

global\_step=tf.train.get\_global\_step(),

optimizer=optimizer,

learning\_rate=params.learning\_rate,

learning\_rate\_decay\_fn=twml.learning\_rate\_decay.get\_learning\_rate\_decay\_fn(params),

)

graph\_output[output\_label] = output

graph\_output["loss"] = loss

graph\_output["train\_op"] = train\_op

return graph\_output

def get\_params(args=None):

parser = get\_arg\_parser\_light\_ranking()

if args is None:

return parser.parse\_args()

else:

return parser.parse\_args(args)

def \_main():

opt = get\_params()

logging.info("parse is: ")

logging.info(opt)

feature\_list = read\_config(opt.feature\_list).items()

feature\_config = get\_feature\_config\_light\_ranking(

data\_spec\_path=opt.data\_spec,

feature\_list\_provided=feature\_list,

opt=opt,

add\_gbdt=opt.use\_gbdt\_features,

run\_light\_ranking\_group\_metrics\_in\_bq=opt.run\_light\_ranking\_group\_metrics\_in\_bq,

)

feature\_list\_path = opt.feature\_list

# --------------------------------------------------------

# Create Trainer

# --------------------------------------------------------

trainer = DataRecordTrainer(

name=opt.model\_trainer\_name,

params=opt,

build\_graph\_fn=build\_graph,

save\_dir=opt.save\_dir,

run\_config=None,

feature\_config=feature\_config,

metric\_fn=get\_metric\_fn(opt.task\_name, use\_stratify\_metrics=False),

)

if opt.directly\_export\_best:

logging.info("Directly exporting the model without training")

else:

# ----------------------------------------------------

# Model Training & Evaluation

# ----------------------------------------------------

eval\_input\_fn = trainer.get\_eval\_input\_fn(repeat=False, shuffle=False)

train\_input\_fn = trainer.get\_train\_input\_fn(shuffle=True)

if opt.distributed or opt.num\_workers is not None:

learn = trainer.train\_and\_evaluate

else:

learn = trainer.learn

logging.info("Training...")

start = datetime.now()

early\_stop\_metric = "rce\_unweighted\_" + opt.task\_name

learn(

early\_stop\_minimize=False,

early\_stop\_metric=early\_stop\_metric,

early\_stop\_patience=opt.early\_stop\_patience,

early\_stop\_tolerance=opt.early\_stop\_tolerance,

eval\_input\_fn=eval\_input\_fn,

train\_input\_fn=train\_input\_fn,

)

end = datetime.now()

logging.info("Training time: " + str(end - start))

logging.info("Exporting the models...")

# --------------------------------------------------------

# Do the model exporting

# --------------------------------------------------------

start = datetime.now()

if not opt.export\_dir:

opt.export\_dir = os.path.join(opt.save\_dir, "exported\_models")

raw\_model\_path = twml.contrib.export.export\_fn.export\_all\_models(

trainer=trainer,

export\_dir=opt.export\_dir,

parse\_fn=feature\_config.get\_parse\_fn(),

serving\_input\_receiver\_fn=feature\_config.get\_serving\_input\_receiver\_fn(),

export\_output\_fn=twml.export\_output\_fns.batch\_prediction\_continuous\_output\_fn,

)

export\_model\_dir = decode\_str\_or\_unicode(raw\_model\_path)

logging.info("Model export time: " + str(datetime.now() - start))

logging.info("The saved model directory is: " + opt.save\_dir)

tf.logging.info("getting default continuous\_feature\_list")

continuous\_feature\_list = get\_feature\_list\_for\_light\_ranking(feature\_list\_path, opt.data\_spec)

continous\_feature\_list\_save\_path = os.path.join(opt.save\_dir, "continuous\_feature\_list.json")

twml.util.write\_file(continous\_feature\_list\_save\_path, continuous\_feature\_list, encode="json")

tf.logging.info(f"Finish writting files to {continous\_feature\_list\_save\_path}")

if opt.run\_light\_ranking\_group\_metrics:

# --------------------------------------------

# Run Light Ranking Group Metrics

# --------------------------------------------

run\_group\_metrics\_light\_ranking(

trainer=trainer,

data\_dir=os.path.join(opt.eval\_data\_dir, opt.eval\_start\_datetime),

model\_path=export\_model\_dir,

parse\_fn=feature\_config.get\_parse\_fn(),

)

if opt.run\_light\_ranking\_group\_metrics\_in\_bq:

# ----------------------------------------------------------------------------------------

# Get Light/Heavy Ranker Predictions for Light Ranking Group Metrics in BigQuery

# ----------------------------------------------------------------------------------------

trainer\_pred = DataRecordTrainer(

name=opt.model\_trainer\_name,

params=opt,

build\_graph\_fn=partial(build\_graph, run\_light\_ranking\_group\_metrics\_in\_bq=True),

save\_dir=opt.save\_dir + "/tmp/",

run\_config=None,

feature\_config=feature\_config,

metric\_fn=get\_metric\_fn(opt.task\_name, use\_stratify\_metrics=False),

)

checkpoint\_folder = os.path.join(opt.save\_dir, "best\_checkpoint")

checkpoint = tf.train.latest\_checkpoint(checkpoint\_folder, latest\_filename=None)

tf.logging.info("\n\nPrediction from Checkpoint: {:}.\n\n".format(checkpoint))

run\_group\_metrics\_light\_ranking\_in\_bq(

trainer=trainer\_pred, params=opt, checkpoint\_path=checkpoint

)

tf.logging.info("Done Training & Prediction.")

if \_\_name\_\_ == "\_\_main\_\_":

\_main()