# checkstyle: noqa

import tensorflow.compat.v1 as tf

from tensorflow.python.estimator.export.export import build\_raw\_serving\_input\_receiver\_fn

from tensorflow.python.framework import dtypes

from tensorflow.python.ops import array\_ops

import tensorflow\_hub as hub

from datetime import datetime

from tensorflow.compat.v1 import logging

from twitter.deepbird.projects.timelines.configs import all\_configs

from twml.trainers import DataRecordTrainer

from twml.contrib.calibrators.common\_calibrators import build\_percentile\_discretizer\_graph

from twml.contrib.calibrators.common\_calibrators import calibrate\_discretizer\_and\_export

from .metrics import get\_multi\_binary\_class\_metric\_fn

from .constants import TARGET\_LABEL\_IDX, PREDICTED\_CLASSES

from .example\_weights import add\_weight\_arguments, make\_weights\_tensor

from .lolly.data\_helpers import get\_lolly\_logits

from .lolly.tf\_model\_initializer\_builder import TFModelInitializerBuilder

from .lolly.reader import LollyModelReader

from .tf\_model.discretizer\_builder import TFModelDiscretizerBuilder

from .tf\_model.weights\_initializer\_builder import TFModelWeightsInitializerBuilder

import twml

def get\_feature\_values(features\_values, params):

if params.lolly\_model\_tsv:

# The default DBv2 HashingDiscretizer bin membership interval is (a, b]

#

# The Earlybird Lolly prediction engine discretizer bin membership interval is [a, b)

#

# TFModelInitializerBuilder converts (a, b] to [a, b) by inverting the bin boundaries.

#

# Thus, invert the feature values, so that HashingDiscretizer can to find the correct bucket.

return tf.multiply(features\_values, -1.0)

else:

return features\_values

def build\_graph(features, label, mode, params, config=None):

weights = None

if "weights" in features:

weights = make\_weights\_tensor(features["weights"], label, params)

num\_bits = params.input\_size\_bits

if mode == "infer":

indices = twml.limit\_bits(features["input\_sparse\_tensor\_indices"], num\_bits)

dense\_shape = tf.stack([features["input\_sparse\_tensor\_shape"][0], 1 << num\_bits])

sparse\_tf = tf.SparseTensor(

indices=indices,

values=get\_feature\_values(features["input\_sparse\_tensor\_values"], params),

dense\_shape=dense\_shape

)

else:

features["values"] = get\_feature\_values(features["values"], params)

sparse\_tf = twml.util.convert\_to\_sparse(features, num\_bits)

if params.lolly\_model\_tsv:

tf\_model\_initializer = TFModelInitializerBuilder().build(LollyModelReader(params.lolly\_model\_tsv))

bias\_initializer, weight\_initializer = TFModelWeightsInitializerBuilder(num\_bits).build(tf\_model\_initializer)

discretizer = TFModelDiscretizerBuilder(num\_bits).build(tf\_model\_initializer)

else:

discretizer = hub.Module(params.discretizer\_save\_dir)

bias\_initializer, weight\_initializer = None, None

input\_sparse = discretizer(sparse\_tf, signature="hashing\_discretizer\_calibrator")

logits = twml.layers.full\_sparse(

inputs=input\_sparse,

output\_size=1,

bias\_initializer=bias\_initializer,

weight\_initializer=weight\_initializer,

use\_sparse\_grads=(mode == "train"),

use\_binary\_values=True,

name="full\_sparse\_1"

)

loss = None

if mode != "infer":

lolly\_activations = get\_lolly\_logits(label)

if opt.print\_data\_examples:

logits = print\_data\_example(logits, lolly\_activations, features)

if params.replicate\_lolly:

loss = tf.reduce\_mean(tf.math.squared\_difference(logits, lolly\_activations))

else:

batch\_size = tf.shape(label)[0]

target\_label = tf.reshape(tensor=label[:, TARGET\_LABEL\_IDX], shape=(batch\_size, 1))

loss = tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=target\_label, logits=logits)

loss = twml.util.weighted\_average(loss, weights)

num\_labels = tf.shape(label)[1]

eb\_scores = tf.tile(lolly\_activations, [1, num\_labels])

logits = tf.tile(logits, [1, num\_labels])

logits = tf.concat([logits, eb\_scores], axis=1)

output = tf.nn.sigmoid(logits)

return {"output": output, "loss": loss, "weights": weights}

def print\_data\_example(logits, lolly\_activations, features):

return tf.Print(

logits,

[logits, lolly\_activations, tf.reshape(features['keys'], (1, -1)), tf.reshape(tf.multiply(features['values'], -1.0), (1, -1))],

message="DATA EXAMPLE = ",

summarize=10000

)

def earlybird\_output\_fn(graph\_output):

export\_outputs = {

tf.saved\_model.signature\_constants.DEFAULT\_SERVING\_SIGNATURE\_DEF\_KEY:

tf.estimator.export.PredictOutput(

{"prediction": tf.identity(graph\_output["output"], name="output\_scores")}

)

}

return export\_outputs

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

parser = DataRecordTrainer.add\_parser\_arguments()

parser = twml.contrib.calibrators.add\_discretizer\_arguments(parser)

parser.add\_argument("--label", type=str, help="label for the engagement")

parser.add\_argument("--model.use\_existing\_discretizer", action="store\_true",

dest="model\_use\_existing\_discretizer",

help="Load a pre-trained calibration or train a new one")

parser.add\_argument("--input\_size\_bits", type=int)

parser.add\_argument("--export\_module\_name", type=str, default="base\_mlp", dest="export\_module\_name")

parser.add\_argument("--feature\_config", type=str)

parser.add\_argument("--replicate\_lolly", type=bool, default=False, dest="replicate\_lolly",

help="Train a regression model with MSE loss and the logged Earlybird score as a label")

parser.add\_argument("--lolly\_model\_tsv", type=str, required=False, dest="lolly\_model\_tsv",

help="Initialize with weights and discretizer bins available in the given Lolly model tsv file"

"No discretizer gets trained or loaded if set.")

parser.add\_argument("--print\_data\_examples", type=bool, default=False, dest="print\_data\_examples",

help="Prints 'DATA EXAMPLE = [[tf logit]][[logged lolly logit]][[feature ids][feature values]]'")

add\_weight\_arguments(parser)

opt = parser.parse\_args()

feature\_config\_module = all\_configs.select\_feature\_config(opt.feature\_config)

feature\_config = feature\_config\_module.get\_feature\_config(data\_spec\_path=opt.data\_spec, label=opt.label)

parse\_fn = twml.parsers.get\_sparse\_parse\_fn(

feature\_config,

keep\_fields=("ids", "keys", "values", "batch\_size", "total\_size", "codes"))

if not opt.lolly\_model\_tsv:

if opt.model\_use\_existing\_discretizer:

logging.info("Skipping discretizer calibration [model.use\_existing\_discretizer=True]")

logging.info(f"Using calibration at {opt.discretizer\_save\_dir}")

else:

logging.info("Calibrating new discretizer [model.use\_existing\_discretizer=False]")

calibrator = twml.contrib.calibrators.HashingDiscretizerCalibrator(

opt.discretizer\_num\_bins,

opt.discretizer\_output\_size\_bits

)

calibrate\_discretizer\_and\_export(name="recap\_earlybird\_hashing\_discretizer",

params=opt,

calibrator=calibrator,

build\_graph\_fn=build\_percentile\_discretizer\_graph,

feature\_config=feature\_config)

trainer = DataRecordTrainer(

name="earlybird",

params=opt,

build\_graph\_fn=build\_graph,

save\_dir=opt.save\_dir,

feature\_config=feature\_config,

metric\_fn=get\_multi\_binary\_class\_metric\_fn(

metrics=["roc\_auc"],

classes=PREDICTED\_CLASSES

),

warm\_start\_from=None

)

train\_input\_fn = trainer.get\_train\_input\_fn(parse\_fn=parse\_fn)

eval\_input\_fn = trainer.get\_eval\_input\_fn(parse\_fn=parse\_fn)

logging.info("Training and Evaluation ...")

trainingStartTime = datetime.now()

trainer.train\_and\_evaluate(train\_input\_fn=train\_input\_fn, eval\_input\_fn=eval\_input\_fn)

trainingEndTime = datetime.now()

logging.info("Training and Evaluation time: " + str(trainingEndTime - trainingStartTime))

if trainer.\_estimator.config.is\_chief:

serving\_input\_in\_earlybird = {

"input\_sparse\_tensor\_indices": array\_ops.placeholder(

name="input\_sparse\_tensor\_indices",

shape=[None, 2],

dtype=dtypes.int64),

"input\_sparse\_tensor\_values": array\_ops.placeholder(

name="input\_sparse\_tensor\_values",

shape=[None],

dtype=dtypes.float32),

"input\_sparse\_tensor\_shape": array\_ops.placeholder(

name="input\_sparse\_tensor\_shape",

shape=[2],

dtype=dtypes.int64)

}

serving\_input\_receiver\_fn = build\_raw\_serving\_input\_receiver\_fn(serving\_input\_in\_earlybird)

twml.contrib.export.export\_fn.export\_all\_models(

trainer=trainer,

export\_dir=opt.export\_dir,

parse\_fn=parse\_fn,

serving\_input\_receiver\_fn=serving\_input\_receiver\_fn,

export\_output\_fn=earlybird\_output\_fn,

feature\_spec=feature\_config.get\_feature\_spec()

)

logging.info("The export model path is: " + opt.export\_dir)