"""

Functions for exporting models for different modes.

"""

from collections import OrderedDict

import os

import tensorflow.compat.v1 as tf

from tensorflow.python.estimator.export import export

import twml

import yaml

def get\_sparse\_batch\_supervised\_input\_receiver\_fn(feature\_config, keep\_fields=None):

"""Gets supervised\_input\_receiver\_fn that decodes a BatchPredictionRequest as sparse tensors

with labels and weights as defined in feature\_config.

This input\_receiver\_fn is required for exporting models with 'train' mode to be trained with

Java API

Args:

feature\_config (FeatureConfig): deepbird v2 feature config object

keep\_fields (list): list of fields to keep

Returns:

supervised\_input\_receiver\_fn: input\_receiver\_fn used for train mode

"""

def supervised\_input\_receiver\_fn():

serialized\_request = tf.placeholder(dtype=tf.uint8, name='request')

receiver\_tensors = {'request': serialized\_request}

bpr = twml.contrib.readers.HashedBatchPredictionRequest(serialized\_request, feature\_config)

features = bpr.get\_sparse\_features() if keep\_fields is None else bpr.get\_features(keep\_fields)

features['weights'] = bpr.weights

labels = bpr.labels

features, labels = bpr.apply\_filter(features, labels)

return export.SupervisedInputReceiver(features, labels, receiver\_tensors)

return supervised\_input\_receiver\_fn

def update\_build\_graph\_fn\_for\_train(build\_graph\_fn):

"""Updates a build\_graph\_fn by inserting in graph output a serialized BatchPredictionResponse

similar to the export\_output\_fns for serving.

The key difference here is that

1. We insert serialized BatchPredictionResponse in graph output with key 'prediction' instead of

creating an export\_output object. This is because of the way estimators export model in 'train'

mode doesn't take custom export\_output

2. We only do it when `mode == 'train'` to avoid altering the graph when exporting

for 'infer' mode

Args:

build\_graph\_fn (Callable): deepbird v2 build graph function

Returns:

new\_build\_graph\_fn: An updated build\_graph\_fn that inserts serialized BatchPredictResponse

to graph output when in 'train' mode

"""

def new\_build\_graph\_fn(features, label, mode, params, config=None):

output = build\_graph\_fn(features, label, mode, params, config)

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

output.update(

twml.export\_output\_fns.batch\_prediction\_continuous\_output\_fn(output)[

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

)

return output

return new\_build\_graph\_fn

def export\_model\_for\_train\_and\_infer(

trainer, feature\_config, keep\_fields, export\_dir, as\_text=False):

"""Function for exporting model with both 'train' and 'infer' mode.

This means the exported saved\_model.pb will contain two meta graphs, one with tag 'train'

and the other with tag 'serve', and it can be loaded in Java API with either tag depending on

the use case

Args:

trainer (DataRecordTrainer): deepbird v2 DataRecordTrainer

feature\_config (FeatureConfig): deepbird v2 feature config

keep\_fields (list of string): list of field keys, e.g.

('ids', 'keys', 'values', 'batch\_size', 'total\_size', 'codes')

export\_dir (str): a directory (local or hdfs) to export model to

as\_text (bool): if True, write 'saved\_model.pb' as binary file, else write

'saved\_model.pbtxt' as human readable text file. Default False

"""

train\_input\_receiver\_fn = get\_sparse\_batch\_supervised\_input\_receiver\_fn(

feature\_config, keep\_fields)

predict\_input\_receiver\_fn = twml.parsers.get\_sparse\_serving\_input\_receiver\_fn(

feature\_config, keep\_fields)

trainer.\_export\_output\_fn = twml.export\_output\_fns.batch\_prediction\_continuous\_output\_fn

trainer.\_build\_graph\_fn = update\_build\_graph\_fn\_for\_train(trainer.\_build\_graph\_fn)

trainer.\_estimator.\_export\_all\_saved\_models(

export\_dir\_base=export\_dir,

input\_receiver\_fn\_map={

tf.estimator.ModeKeys.TRAIN: train\_input\_receiver\_fn,

tf.estimator.ModeKeys.PREDICT: predict\_input\_receiver\_fn

},

as\_text=as\_text,

)

trainer.export\_model\_effects(export\_dir)

def export\_all\_models\_with\_receivers(estimator, export\_dir,

train\_input\_receiver\_fn,

eval\_input\_receiver\_fn,

predict\_input\_receiver\_fn,

export\_output\_fn,

export\_modes=('train', 'eval', 'predict'),

register\_model\_fn=None,

feature\_spec=None,

checkpoint\_path=None,

log\_features=True):

"""

Function for exporting a model with train, eval, and infer modes.

Args:

estimator:

Should be of type tf.estimator.Estimator.

You can get this from trainer using trainer.estimator

export\_dir:

Directory to export the model.

train\_input\_receiver\_fn:

Input receiver for train interface.

eval\_input\_receiver\_fn:

Input receiver for eval interface.

predict\_input\_receiver\_fn:

Input receiver for predict interface.

export\_output\_fn:

export\_output\_fn to be used for serving.

export\_modes:

A list to Specify what modes to export. Can be "train", "eval", "predict".

Defaults to ["train", "eval", "predict"]

register\_model\_fn:

An optional function which is called with export\_dir after models are exported.

Defaults to None.

Returns:

The timestamped directory the models are exported to.

"""

# TODO: Fix for hogwild / distributed training.

if export\_dir is None:

raise ValueError("export\_dir can not be None")

export\_dir = twml.util.sanitize\_hdfs\_path(export\_dir)

input\_receiver\_fn\_map = {}

if "train" in export\_modes:

input\_receiver\_fn\_map[tf.estimator.ModeKeys.TRAIN] = train\_input\_receiver\_fn

if "eval" in export\_modes:

input\_receiver\_fn\_map[tf.estimator.ModeKeys.EVAL] = eval\_input\_receiver\_fn

if "predict" in export\_modes:

input\_receiver\_fn\_map[tf.estimator.ModeKeys.PREDICT] = predict\_input\_receiver\_fn

export\_dir = estimator.\_export\_all\_saved\_models(

export\_dir\_base=export\_dir,

input\_receiver\_fn\_map=input\_receiver\_fn\_map,

checkpoint\_path=checkpoint\_path,

)

if register\_model\_fn is not None:

register\_model\_fn(export\_dir, feature\_spec, log\_features)

return export\_dir

def export\_all\_models(trainer,

export\_dir,

parse\_fn,

serving\_input\_receiver\_fn,

export\_output\_fn=None,

export\_modes=('train', 'eval', 'predict'),

feature\_spec=None,

checkpoint=None,

log\_features=True):

"""

Function for exporting a model with train, eval, and infer modes.

Args:

trainer:

An object of type twml.trainers.Trainer.

export\_dir:

Directory to export the model.

parse\_fn:

The parse function used parse the inputs for train and eval.

serving\_input\_receiver\_fn:

The input receiver function used during serving.

export\_output\_fn:

export\_output\_fn to be used for serving.

export\_modes:

A list to Specify what modes to export. Can be "train", "eval", "predict".

Defaults to ["train", "eval", "predict"]

feature\_spec:

A dictionary obtained from FeatureConfig.get\_feature\_spec() to serialize

as feature\_spec.yaml in export\_dir.

Defaults to None

Returns:

The timestamped directory the models are exported to.

"""

# Only export from chief in hogwild or distributed modes.

if trainer.params.get('distributed', False) and not trainer.estimator.config.is\_chief:

tf.logging.info("Trainer.export\_model ignored due to instance not being chief.")

return

if feature\_spec is None:

if getattr(trainer, '\_feature\_config') is None:

raise ValueError("feature\_spec is set to None."

"Please pass feature\_spec=feature\_config.get\_feature\_spec() to the export\_all\_model function")

else:

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

export\_dir = twml.util.sanitize\_hdfs\_path(export\_dir)

old\_export\_output\_fn = trainer.\_export\_output\_fn

trainer.\_export\_output\_fn = export\_output\_fn

supervised\_input\_receiver\_fn = twml.parsers.convert\_to\_supervised\_input\_receiver\_fn(parse\_fn)

if not checkpoint:

checkpoint = trainer.best\_or\_latest\_checkpoint

export\_dir = export\_all\_models\_with\_receivers(estimator=trainer.estimator,

export\_dir=export\_dir,

train\_input\_receiver\_fn=supervised\_input\_receiver\_fn,

eval\_input\_receiver\_fn=supervised\_input\_receiver\_fn,

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

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

export\_modes=export\_modes,

register\_model\_fn=trainer.export\_model\_effects,

feature\_spec=feature\_spec,

checkpoint\_path=checkpoint,

log\_features=log\_features)

trainer.\_export\_output\_fn = old\_export\_output\_fn

return export\_dir

def export\_feature\_spec(dir\_path, feature\_spec\_dict):

"""

Exports a FeatureConfig.get\_feature\_spec() dict to <dir\_path>/feature\_spec.yaml.

"""

def ordered\_dict\_representer(dumper, data):

return dumper.represent\_mapping('tag:yaml.org,2002:map', data.items())

try:

# needed for Python 2

yaml.add\_representer(str, yaml.representer.SafeRepresenter.represent\_str)

yaml.add\_representer(unicode, yaml.representer.SafeRepresenter.represent\_unicode)

except NameError:

# 'unicode' type doesn't exist on Python 3

# PyYAML handles unicode correctly in Python 3

pass

yaml.add\_representer(OrderedDict, ordered\_dict\_representer)

fbase = "feature\_spec.yaml"

fname = fbase.encode('utf-8') if type(dir\_path) != str else fbase

file\_path = os.path.join(dir\_path, fname)

with tf.io.gfile.GFile(file\_path, mode='w') as f:

yaml.dump(feature\_spec\_dict, f, default\_flow\_style=False, allow\_unicode=True)

tf.logging.info("Exported feature spec to %s" % file\_path)

return file\_path

# Keep the alias for compatibility.

get\_supervised\_input\_receiver\_fn = twml.parsers.convert\_to\_supervised\_input\_receiver\_fn