"""

This is a temporary close gap solution that allows TensorFlow users to do exploration and

experimentation using Keras models, and production training using twml Trainer.

As of now (Q4 2019), Keras model training using `model.fit()` has various issues, making it unfit

for production training:

1. `model.fit()` is slow in TF 1.14. This will be fixed with future TensorFlow updates.

2. `model.fit()` crashes during model saving or in eager mode when the input has SparseTensor.

3. Models saved using TF 2.0 API cannot be served by TensorFlow's Java API.

Until MLCE team resolves the above issues, MLCE team recommends the following:

- Please feel free to use Keras models for experimentation and exploration.

- Please stick to twml Trainer for production training & exporting,

especially if you want to serve your model using Twitter's prediction servers.

This module provide tooling for easily training keras models using twml Trainer.

This module takes a Keras model that performs binary classification, and returns a

`twml.trainers.Trainer` object performing the same task.

The common way to use the returned Trainer object is to call its

`train`, `evaluate`, `learn`, or `train\_and\_evaluate` method with an input function.

This input function can be created from the tf.data.Dataset you used with your Keras model.

.. note: this util handles the most common case. If you have cases not satisfied by this util,

consider writing your own build\_graph to wrap your keras models.

"""

from twitter.deepbird.hparam import HParams

import tensorflow # noqa: F401

import tensorflow.compat.v2 as tf

import twml

def build\_keras\_trainer(

name,

model\_factory,

save\_dir,

loss\_fn=None,

metrics\_fn=None,

\*\*kwargs):

"""

Compile the given model\_factory into a twml Trainer.

Args:

name: a string name for the returned twml Trainer.

model\_factory: a callable that returns a keras model when called.

This keras model is expected to solve a binary classification problem.

This keras model takes a dict of tensors as input, and outputs a logit or probability.

save\_dir: a directory where the trainer saves data. Can be an HDFS path.

loss\_fn: the loss function to use. Defaults to tf.keras.losses.BinaryCrossentropy.

metrics\_fn: metrics function used by TensorFlow estimators.

Defaults to twml.metrics.get\_binary\_class\_metric\_fn().

\*\*kwargs: for people familiar with twml Trainer's options, they can be passed in here

as kwargs, and they will be forwarded to Trainer as opts.

See https://cgit.twitter.biz/source/tree/twml/twml/argument\_parser.py#n43 for available args.

Returns:

a twml.trainers.Trainer object which can be used for training and exporting models.

"""

build\_graph = create\_build\_graph\_fn(model\_factory, loss\_fn)

if metrics\_fn is None:

metrics\_fn = twml.metrics.get\_binary\_class\_metric\_fn()

opts = HParams(\*\*kwargs)

opts.add\_hparam('save\_dir', save\_dir)

return twml.trainers.Trainer(

name,

opts,

build\_graph\_fn=build\_graph,

save\_dir=save\_dir,

metric\_fn=metrics\_fn)

def create\_build\_graph\_fn(model\_factory, loss\_fn=None):

"""Create a build graph function from the given keras model."""

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

# create model from model factory.

model = model\_factory()

# create loss function if the user didn't specify one.

if loss\_fn is None:

build\_graph\_loss\_fn = tf.keras.losses.BinaryCrossentropy(from\_logits=False)

else:

build\_graph\_loss\_fn = loss\_fn

output = model(features)

if mode == 'infer':

loss = None

else:

weights = features.get('weights', None)

loss = build\_graph\_loss\_fn(y\_true=label, y\_pred=output, sample\_weight=weights)

if isinstance(output, dict):

if loss is None:

return output

else:

output['loss'] = loss

return output

else:

return {'output': output, 'loss': loss}

return build\_graph