# checkstyle: noqa

import tensorflow.compat.v1 as tf

from collections import OrderedDict

from .constants import EB\_SCORE\_IDX

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

import twml

def get\_multi\_binary\_class\_metric\_fn(metrics, classes=None, class\_dim=1):

"""

This function was copied from twml/metrics.py with the following adjustments:

- Override example weights with the ones set in graph\_output.

- Tile labels in order to support per engagement metrics for both TF and Lolly scores.

- Add lolly\_tf\_score\_MSE metric.

Note: All custom lines have a comment that starts with 'Added'

"""

# pylint: disable=invalid-name,dict-keys-not-iterating

if metrics is None:

# remove expensive metrics by default for faster eval

metrics = list(twml.metrics.SUPPORTED\_BINARY\_CLASS\_METRICS.keys())

metrics.remove('pr\_curve')

def get\_eval\_metric\_ops(graph\_output, labels, weights):

"""

graph\_output:

dict that is returned by build\_graph given input features.

labels:

target labels associated to batch.

weights:

weights of the samples..

"""

# Added to support the example weights overriding.

weights = graph\_output["weights"]

# Added to support per engagement metrics for both TF and Lolly scores.

labels = tf.tile(labels, [1, 2])

eval\_metric\_ops = OrderedDict()

preds = graph\_output['output']

threshold = graph\_output['threshold'] if 'threshold' in graph\_output else 0.5

hard\_preds = graph\_output.get('hard\_output')

if not hard\_preds:

hard\_preds = tf.greater\_equal(preds, threshold)

shape = labels.get\_shape()

# basic sanity check: multi\_metric dimension must exist

assert len(shape) > class\_dim, "Dimension specified by class\_dim does not exist."

num\_labels = shape[class\_dim]

# If we are doing multi-class / multi-label metric, the number of classes / labels must

# be know at graph construction time. This dimension cannot have size None.

assert num\_labels is not None, "The multi-metric dimension cannot be None."

assert classes is None or len(classes) == num\_labels, (

"Number of classes must match the number of labels")

weights\_shape = weights.get\_shape() if weights is not None else None

if weights\_shape is None:

num\_weights = None

elif len(weights\_shape) > 1:

num\_weights = weights\_shape[class\_dim]

else:

num\_weights = 1

for i in range(num\_labels):

# add metrics to eval\_metric\_ops dict

for metric\_name in metrics:

metric\_name = metric\_name.lower() # metric name are case insensitive.

class\_metric\_name = metric\_name + "\_" + (classes[i] if classes is not None else str(i))

if class\_metric\_name in eval\_metric\_ops:

# avoid adding duplicate metrics.

continue

class\_labels = tf.gather(labels, indices=[i], axis=class\_dim)

class\_preds = tf.gather(preds, indices=[i], axis=class\_dim)

class\_hard\_preds = tf.gather(hard\_preds, indices=[i], axis=class\_dim)

if num\_weights is None:

class\_weights = None

elif num\_weights == num\_labels:

class\_weights = tf.gather(weights, indices=[i], axis=class\_dim)

elif num\_weights == 1:

class\_weights = weights

else:

raise ValueError("num\_weights (%d) and num\_labels (%d) do not match"

% (num\_weights, num\_labels))

metric\_factory, requires\_threshold = twml.metrics.SUPPORTED\_BINARY\_CLASS\_METRICS.get(metric\_name)

if metric\_factory:

value\_op, update\_op = metric\_factory(

labels=class\_labels,

predictions=(class\_hard\_preds if requires\_threshold else class\_preds),

weights=class\_weights, name=class\_metric\_name)

eval\_metric\_ops[class\_metric\_name] = (value\_op, update\_op)

else:

raise ValueError('Cannot find the metric named ' + metric\_name)

# Added to compare TF and Lolly scores.

eval\_metric\_ops["lolly\_tf\_score\_MSE"] = get\_mse(graph\_output["output"], labels)

return eval\_metric\_ops

return get\_eval\_metric\_ops

def get\_mse(predictions, labels):

lolly\_scores = get\_lolly\_scores(labels)

tf\_scores = predictions[:, EB\_SCORE\_IDX]

squared\_lolly\_tf\_score\_diff = tf.square(tf.subtract(tf\_scores, lolly\_scores))

value\_op = tf.reduce\_mean(squared\_lolly\_tf\_score\_diff, name="value\_op")

update\_op = tf.reduce\_mean(squared\_lolly\_tf\_score\_diff, name="update\_op")

return value\_op, update\_op