"""

Module containing extra tensorflow metrics used at Twitter.

This module conforms to conventions used by tf.metrics.\*.

In particular, each metric constructs two subgraphs: value\_op and update\_op:

- The value op is used to fetch the current metric value.

- The update\_op is used to accumulate into the metric.

Note: similar to tf.metrics.\*, metrics in here do not support multi-label learning.

We will have to write wrapper classes to create one metric per label.

Note: similar to tf.metrics.\*, batches added into a metric via its update\_op are cumulative!

"""

from collections import OrderedDict

from functools import partial

import tensorflow.compat.v1 as tf

from tensorflow.python.eager import context

from tensorflow.python.framework import dtypes, ops

from tensorflow.python.ops import array\_ops, state\_ops

import twml

from twml.contrib.utils import math\_fns

def ndcg(labels, predictions,

metrics\_collections=None,

updates\_collections=None,

name=None,

top\_k\_int=1):

# pylint: disable=unused-argument

"""

Compute full normalized discounted cumulative gain (ndcg) based on predictions

ndcg = dcg\_k/idcg\_k, k is a cut off ranking postion

There are a few variants of ndcg

The dcg (discounted cumulative gain) formula used in

twml.contrib.metrics.ndcg is::

\\sum\_{i=1}^k \frac{2^{relevance\\\_score} -1}{\\log\_{2}(i + 1)}

k is the length of items to be ranked in a batch/query

Notice that whether k will be replaced with a fixed value requires discussions

The scores in predictions are transformed to order and relevance scores to calculate ndcg

A relevance score means how relevant a DataRecord is to a particular query

Arguments:

labels: the ground truth value.

predictions: the predicted values, whose shape must match labels. Ignored for CTR computation.

metrics\_collections: optional list of collections to add this metric into.

updates\_collections: optional list of collections to add the associated update\_op into.

name: an optional variable\_scope name.

Returns:

ndcg: A `Tensor` representing the ndcg score.

update\_op: A update operation used to accumulate data into this metric.

"""

with tf.variable\_scope(name, 'ndcg', (labels, predictions)):

label\_scores = tf.to\_float(labels, name='label\_to\_float')

predicted\_scores = tf.to\_float(predictions, name='predictions\_to\_float')

if context.executing\_eagerly():

raise RuntimeError('ndcg is not supported when eager execution '

'is enabled.')

total\_ndcg = \_metric\_variable([], dtypes.float32, name='total\_ndcg')

count\_query = \_metric\_variable([], dtypes.float32, name='query\_count')

# actual ndcg cutoff position top\_k\_int

max\_prediction\_size = array\_ops.size(predicted\_scores)

top\_k\_int = tf.minimum(max\_prediction\_size, top\_k\_int)

# the ndcg score of the batch

ndcg = math\_fns.cal\_ndcg(label\_scores,

predicted\_scores, top\_k\_int=top\_k\_int)

# add ndcg of the current batch to total\_ndcg

update\_total\_op = state\_ops.assign\_add(total\_ndcg, ndcg)

with ops.control\_dependencies([ndcg]):

# count\_query stores the number of queries

# count\_query increases by 1 for each batch/query

update\_count\_op = state\_ops.assign\_add(count\_query, 1)

mean\_ndcg = math\_fns.safe\_div(total\_ndcg, count\_query, 'mean\_ndcg')

update\_op = math\_fns.safe\_div(update\_total\_op, update\_count\_op, 'update\_mean\_ndcg\_op')

if metrics\_collections:

ops.add\_to\_collections(metrics\_collections, mean\_ndcg)

if updates\_collections:

ops.add\_to\_collections(updates\_collections, update\_op)

return mean\_ndcg, update\_op

# Copied from metrics\_impl.py with minor modifications.

# https://github.com/tensorflow/tensorflow/blob/v1.5.0/tensorflow/python/ops/metrics\_impl.py#L39

def \_metric\_variable(shape, dtype, validate\_shape=True, name=None):

"""Create variable in `GraphKeys.(LOCAL|METRIC\_VARIABLES`) collections."""

return tf.Variable(

lambda: tf.zeros(shape, dtype),

trainable=False,

collections=[tf.GraphKeys.LOCAL\_VARIABLES, tf.GraphKeys.METRIC\_VARIABLES],

validate\_shape=validate\_shape,

name=name)

# binary metric\_name: (metric, requires thresholded output)

SUPPORTED\_BINARY\_CLASS\_METRICS = {

# TWML binary metrics

'rce': (twml.metrics.rce, False),

'nrce': (partial(twml.metrics.rce, normalize=True), False),

# CTR measures positive sample ratio. This terminology is inherited from Ads.

'ctr': (twml.metrics.ctr, False),

# predicted CTR measures predicted positive ratio.

'predicted\_ctr': (twml.metrics.predicted\_ctr, False),

# thresholded metrics

'accuracy': (tf.metrics.accuracy, True),

'precision': (tf.metrics.precision, True),

'recall': (tf.metrics.recall, True),

# tensorflow metrics

'roc\_auc': (partial(tf.metrics.auc, curve='ROC'), False),

'pr\_auc': (partial(tf.metrics.auc, curve='PR'), False),

}

# search metric\_name: metric

SUPPORTED\_SEARCH\_METRICS = {

# TWML search metrics

# ndcg needs the raw prediction scores to sort

'ndcg': ndcg,

}

def get\_search\_metric\_fn(binary\_metrics=None, search\_metrics=None,

ndcg\_top\_ks=[1, 3, 5, 10], use\_binary\_metrics=False):

"""

Returns a function having signature:

.. code-block:: python

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

...

return eval\_metric\_ops

where the returned eval\_metric\_ops is a dict of common evaluation metric

Ops for ranking. See `tf.estimator.EstimatorSpec

<https://www.tensorflow.org/api\_docs/python/tf/estimator/EstimatorSpec>`\_

for a description of eval\_metric\_ops. The graph\_output is a the result

dict returned by build\_graph. Labels and weights are tf.Tensors.

The following graph\_output keys are recognized:

output:

the raw predictions. Required.

threshold:

Only used in SUPPORTED\_BINARY\_CLASS\_METRICS

If the lables are 0s and 1s

A value between 0 and 1 used to threshold the output into a hard\_output.

Defaults to 0.5 when threshold and hard\_output are missing.

Either threshold or hard\_output can be provided, but not both.

hard\_output:

Only used in SUPPORTED\_BINARY\_CLASS\_METRICS

A thresholded output. Either threshold or hard\_output can be provided, but not both.

Arguments:

only used in pointwise learning-to-rank

binary\_metrics (list of String):

a list of metrics of interest. E.g. ['ctr', 'accuracy', 'rce']

These metrics are evaluated and reported to tensorboard \*during the eval phases only\*.

Supported metrics:

- ctr (same as positive sample ratio.)

- rce (cross entropy loss compared to the baseline model of always predicting ctr)

- nrce (normalized rce, do not use this one if you do not understand what it is)

- pr\_auc

- roc\_auc

- accuracy (percentage of predictions that are correct)

- precision (true positives) / (true positives + false positives)

- recall (true positives) / (true positives + false negatives)

NOTE: accuracy / precision / recall apply to binary classification problems only.

I.e. a prediction is only considered correct if it matches the label. E.g. if the label

is 1.0, and the prediction is 0.99, it does not get credit. If you want to use

precision / recall / accuracy metrics with soft predictions, you'll need to threshold

your predictions into hard 0/1 labels.

When binary\_metrics is None (the default), it defaults to all supported metrics

search\_metrics (list of String):

a list of metrics of interest. E.g. ['ndcg']

These metrics are evaluated and reported to tensorboard \*during the eval phases only\*.

Supported metrics:

- ndcg

NOTE: ndcg works for ranking-relatd problems.

A batch contains all DataRecords that belong to the same query

If pair\_in\_batch\_mode used in scalding -- a batch contains a pair of DataRecords

that belong to the same query and have different labels -- ndcg does not apply in here.

When search\_metrics is None (the default), it defaults to all supported search metrics

currently only 'ndcg'

ndcg\_top\_ks (list of integers):

The cut-off ranking postions for a query

When ndcg\_top\_ks is None or empty (the default), it defaults to [1, 3, 5, 10]

use\_binary\_metrics:

False (default)

Only set it to true in pointwise learning-to-rank

"""

# pylint: disable=dict-keys-not-iterating

if ndcg\_top\_ks is None or not ndcg\_top\_ks:

ndcg\_top\_ks = [1, 3, 5, 10]

if search\_metrics is None:

search\_metrics = list(SUPPORTED\_SEARCH\_METRICS.keys())

if binary\_metrics is None and use\_binary\_metrics:

# Added SUPPORTED\_BINARY\_CLASS\_METRICS in twml.metics as well

# they are only used in pointwise learing-to-rank

binary\_metrics = list(SUPPORTED\_BINARY\_CLASS\_METRICS.keys())

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..

"""

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')

# hard\_preds is a tensor

# check hard\_preds is None and then check if it is empty

if hard\_preds is None or tf.equal(tf.size(hard\_preds), 0):

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

# add search metrics to eval\_metric\_ops dict

for metric\_name in search\_metrics:

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

if metric\_name in eval\_metric\_ops:

# avoid adding duplicate metrics.

continue

search\_metric\_factory = SUPPORTED\_SEARCH\_METRICS.get(metric\_name)

if search\_metric\_factory:

if metric\_name == 'ndcg':

for top\_k in ndcg\_top\_ks:

# metric name will show as ndcg\_1, ndcg\_10, ...

metric\_name\_ndcg\_top\_k = metric\_name + '\_' + str(top\_k)

top\_k\_int = tf.constant(top\_k, dtype=tf.int32)

# Note: having weights in ndcg does not make much sense

# Because ndcg already has position weights/discounts

# Thus weights are not applied in ndcg metric

value\_op, update\_op = search\_metric\_factory(

labels=labels,

predictions=preds,

name=metric\_name\_ndcg\_top\_k,

top\_k\_int=top\_k\_int)

eval\_metric\_ops[metric\_name\_ndcg\_top\_k] = (value\_op, update\_op)

else:

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

if use\_binary\_metrics:

# add binary metrics to eval\_metric\_ops dict

for metric\_name in binary\_metrics:

if metric\_name in eval\_metric\_ops:

# avoid adding duplicate metrics.

continue

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

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

if binary\_metric\_factory:

value\_op, update\_op = binary\_metric\_factory(

labels=labels,

predictions=(hard\_preds if requires\_threshold else preds),

weights=weights,

name=metric\_name)

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

else:

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

return eval\_metric\_ops

return get\_eval\_metric\_ops