# pylint: disable=no-member, attribute-defined-outside-init, too-many-instance-attributes

"""

Implementing PercentileDiscretizer Layer

"""

import libtwml

import numpy as np

import tensorflow.compat.v1 as tf

import twml

from twml.layers import Layer

class PercentileDiscretizer(Layer):

"""

PercentileDiscretizer layer is constructed by PercentileDiscretizerCalibrator after

accumulating data and performing percentile bucket calibration.

PercentileDiscretizer takes sparse continuous features and converts then to sparse

binary features. Each binary output feature is associated to an PercentileDiscretizer bin.

Each PercentileDiscretizer input feature is converted to n\_bin bins.

Each PercentileDiscretizer calibration tries to find bin delimiters such

that the number of features values per bin is roughly equal (for

each given PercentileDiscretizer feature). In other words, bins are calibrated to be approx.

equiprobable, according to the given calibration data.

Note that if an input feature is rarely used, so will its associated output bin/features.

"""

def \_\_init\_\_(

self,

n\_feature, n\_bin, out\_bits,

bin\_values=None, hash\_keys=None, hash\_values=None,

bin\_ids=None, feature\_offsets=None, num\_parts=1, cost\_per\_unit=100, \*\*kwargs):

"""

Creates a non-initialized `PercentileDiscretizer` object.

Before using the table you will have to initialize it. After initialization

the table will be immutable.

If there are no calibrated features, then the discretizer will only apply

twml.util.limit\_bits to the the feature keys (aka "feature\_ids"). Essentially,

the discretizer will be a "no-operation", other than obeying `out\_bits`

Parent class args:

see [tf.layers.Layer](https://www.tensorflow.org/api\_docs/python/tf/layers/Layer)

for documentation of parent class arguments.

Required args:

n\_feature:

number of unique features accumulated during PercentileDiscretizer calibration.

This is the number of features in the hash map.

Used to initialize bin\_values, hash\_keys, hash\_values,

bin\_ids, bin\_values and feature\_offsets.

n\_bin:

number of PercentileDiscretizer bins used for PercentileDiscretizer calibration.

Used to initialize bin\_values, hash\_keys, hash\_values,

bin\_ids, bin\_values and feature\_offsets.

out\_bits:

Determines the maximum value for output feature IDs.

The dense\_shape of the SparseTensor returned by lookup(x)

will be [x.shape[0], 1 << output\_bits].

Optional args:

hash\_keys:

contains the features ID that PercentileDiscretizer discretizes and knows about.

The hash map (hash\_keys->hash\_values) is used for two reasons:

1. divide inputs into two feature spaces:

PercentileDiscretizer vs non-PercentileDiscretizer

2. transate the PercentileDiscretizer features into a hash\_feature ID that

PercentileDiscretizer understands.

The hash\_map is expected to contain n\_feature items.

hash\_values:

translates the feature IDs into hash\_feature IDs for PercentileDiscretizer.

bin\_ids:

a 1D Tensor of size n\_feature \* n\_bin + 1 which contains

unique IDs to which the PercentileDiscretizer features will be translated to.

For example, tf.Tensor(np.arange(n\_feature \* n\_bin)) would produce

the most efficient output space.

bin\_values:

a 1D Tensor aligned with bin\_ids.

For a given hash\_feature ID j, it's value bin's are indexed between

`j\*n\_bin` and `j\*n\_bin + n\_bin-1`.

As such, bin\_ids[j\*n\_bin+i] is translated from a hash\_feature ID of j

and a inputs value between

`bin\_values[j\*n\_bin + i]` and `bin\_values[j\*n\_bin+i+1]`.

feature\_offsets:

a 1D Tensor specifying the starting location of bins for a given feature id.

For example, tf.Tensor(np.arange(0, bin\_values.size, n\_bin, dtype='int64')).

"""

super(PercentileDiscretizer, self).\_\_init\_\_(\*\*kwargs)

if not self.built:

self.build(input\_shape=None)

max\_discretizer\_feature = n\_feature \* (n\_bin + 1)

self.\_n\_feature = n\_feature

self.\_n\_bin = n\_bin

# build variables

self.\_out\_bits = out\_bits

self.\_output\_size = tf.convert\_to\_tensor(1 << out\_bits, tf.int64)

self.\_hash\_keys = (hash\_keys if hash\_keys is not None else

np.empty(n\_feature, dtype=np.int64))

self.\_hash\_values = (hash\_values if hash\_values is not None else

np.empty(n\_feature, dtype=np.int64))

self.\_bin\_ids = (bin\_ids if bin\_ids is not None else

np.empty(max\_discretizer\_feature, dtype=np.int64))

self.\_bin\_values = (bin\_values if bin\_values is not None else

np.empty(max\_discretizer\_feature, dtype=np.float32))

self.\_feature\_offsets = (feature\_offsets if feature\_offsets is not None else

np.empty(n\_feature, dtype=np.int64))

self.num\_parts = num\_parts

self.cost\_per\_unit = cost\_per\_unit

def build(self, input\_shape): # pylint: disable=unused-argument

"""

Creates the variables of the layer

"""

self.built = True

def call(self, inputs, keep\_inputs=False, \*\*kwargs):

"""Looks up `keys` in a table, outputs the corresponding values.

Implements PercentileDiscretizer inference where inputs are intersected with a hash\_map.

Input features that were not calibrated have their feature IDs truncated, so as

to be less than 1<<output\_bits, but their values remain untouched (not discretized)

If there are no calibrated features, then the discretizer will only apply

twml.util.limit\_bits to the the feature keys (aka "feature\_ids"). Essentially,

the discretizer will be a "no-operation", other than obeying `out\_bits`

Args:

inputs: A 2D SparseTensor that is input to PercentileDiscretizer for discretization.

It has a dense\_shape of [batch\_size, input\_size]

keep\_inputs:

Include the original inputs in the output.

Note - if True, undiscretized features will be passed through, but will have

their values doubled (unless there are no calibrated features to discretize).

name: A name for the operation (optional).

Returns:

A `SparseTensor` of the same type as `inputs`.

Its dense\_shape is [shape\_input.dense\_shape[0], 1 << output\_bits].

"""

if isinstance(inputs, tf.SparseTensor):

inputs = twml.SparseTensor.from\_tf(inputs)

assert(isinstance(inputs, twml.SparseTensor))

# sparse column indices

ids = inputs.ids

# sparse row indices

keys = inputs.indices

# sparse values

vals = inputs.values

if self.\_n\_feature > 0:

discretizer\_keys, discretizer\_vals = libtwml.ops.percentile\_discretizer\_v2(

input\_ids=keys, # inc key assigned to feature\_id, or -1

input\_vals=vals, # the observed feature values

bin\_ids=self.\_bin\_ids, # n\_feat X (n\_bin+1) 2D arange

bin\_vals=self.\_bin\_values, # bin boundaries

feature\_offsets=self.\_feature\_offsets, # 0 : nbin\_1 : max\_feat

output\_bits=self.\_out\_bits,

feature\_ids=tf.make\_tensor\_proto(self.\_hash\_keys), # feature ids to build internal hash map

feature\_indices=tf.make\_tensor\_proto(self.\_hash\_values), # keys associated w/ feat. indices

start\_compute=tf.constant(0, shape=[], dtype=tf.int64),

end\_compute=tf.constant(-1, shape=[], dtype=tf.int64),

cost\_per\_unit=self.cost\_per\_unit

)

else:

discretizer\_keys = twml.util.limit\_bits(keys, self.\_out\_bits)

discretizer\_vals = vals

# don't 2x the input.

keep\_inputs = False

batch\_size = tf.to\_int64(inputs.dense\_shape[0])

output\_shape = [batch\_size, self.\_output\_size]

output = twml.SparseTensor(ids, discretizer\_keys, discretizer\_vals, output\_shape).to\_tf()

if keep\_inputs:

# Note the non-discretized features will end up doubled,

# since these are already in `output`

# handle output ID conflicts

mdl\_size = self.\_n\_feature \* (self.\_n\_bin + 1)

non\_mdl\_size = tf.subtract(self.\_output\_size, mdl\_size)

input\_keys = tf.add(tf.floormod(keys, non\_mdl\_size), mdl\_size)

new\_input = twml.SparseTensor(

ids=ids, indices=input\_keys, values=vals, dense\_shape=output\_shape).to\_tf()

# concatenate discretizer output with original input

sparse\_add = tf.sparse\_add(new\_input, output)

output = tf.SparseTensor(sparse\_add.indices, sparse\_add.values, output\_shape)

return output

def compute\_output\_shape(self, input\_shape):

"""Computes the output shape of the layer given the input shape.

Args:

input\_shape: A (possibly nested tuple of) `TensorShape`. It need not

be fully defined (e.g. the batch size may be unknown).

Raises NotImplementedError.

"""

raise NotImplementedError