import os

from typing import Dict

from twitter.deepbird.projects.magic\_recs.libs.model\_utils import filter\_nans\_and\_infs

import twml

from twml.layers import full\_sparse, sparse\_max\_norm

from .params import FeaturesParams, GraphParams, SparseFeaturesParams

import tensorflow as tf

from tensorflow import Tensor

import tensorflow.compat.v1 as tf1

FEAT\_CONFIG\_DEFAULT\_VAL = 0

DEFAULT\_FEATURE\_LIST\_PATH = "./feature\_list\_default.yaml"

FEATURE\_LIST\_DEFAULT\_PATH = os.path.join(

os.path.dirname(os.path.realpath(\_\_file\_\_)), DEFAULT\_FEATURE\_LIST\_PATH

)

def get\_feature\_config(data\_spec\_path=None, feature\_list\_provided=[], params: GraphParams = None):

a\_string\_feat\_list = [feat for feat, feat\_type in feature\_list\_provided if feat\_type != "S"]

builder = twml.contrib.feature\_config.FeatureConfigBuilder(

data\_spec\_path=data\_spec\_path, debug=False

)

builder = builder.extract\_feature\_group(

feature\_regexes=a\_string\_feat\_list,

group\_name="continuous\_features",

default\_value=FEAT\_CONFIG\_DEFAULT\_VAL,

type\_filter=["CONTINUOUS"],

)

builder = builder.extract\_feature\_group(

feature\_regexes=a\_string\_feat\_list,

group\_name="binary\_features",

type\_filter=["BINARY"],

)

if params.model.features.sparse\_features:

builder = builder.extract\_features\_as\_hashed\_sparse(

feature\_regexes=a\_string\_feat\_list,

hash\_space\_size\_bits=params.model.features.sparse\_features.bits,

type\_filter=["DISCRETE", "STRING", "SPARSE\_BINARY"],

output\_tensor\_name="sparse\_not\_continuous",

)

builder = builder.extract\_features\_as\_hashed\_sparse(

feature\_regexes=[feat for feat, feat\_type in feature\_list\_provided if feat\_type == "S"],

hash\_space\_size\_bits=params.model.features.sparse\_features.bits,

type\_filter=["SPARSE\_CONTINUOUS"],

output\_tensor\_name="sparse\_continuous",

)

builder = builder.add\_labels([task.label for task in params.tasks] + ["label.ntabDislike"])

if params.weight:

builder = builder.define\_weight(params.weight)

return builder.build()

def dense\_features(features: Dict[str, Tensor], training: bool) -> Tensor:

"""

Performs feature transformations on the raw dense features (continuous and binary).

"""

with tf.name\_scope("dense\_features"):

x = filter\_nans\_and\_infs(features["continuous\_features"])

x = tf.sign(x) \* tf.math.log(tf.abs(x) + 1)

x = tf1.layers.batch\_normalization(

x, momentum=0.9999, training=training, renorm=training, axis=1

)

x = tf.clip\_by\_value(x, -5, 5)

transformed\_continous\_features = tf.where(tf.math.is\_nan(x), tf.zeros\_like(x), x)

binary\_features = filter\_nans\_and\_infs(features["binary\_features"])

binary\_features = tf.dtypes.cast(binary\_features, tf.float32)

output = tf.concat([transformed\_continous\_features, binary\_features], axis=1)

return output

def sparse\_features(

features: Dict[str, Tensor], training: bool, params: SparseFeaturesParams

) -> Tensor:

"""

Performs feature transformations on the raw sparse features.

"""

with tf.name\_scope("sparse\_features"):

with tf.name\_scope("sparse\_not\_continuous"):

sparse\_not\_continuous = full\_sparse(

inputs=features["sparse\_not\_continuous"],

output\_size=params.embedding\_size,

use\_sparse\_grads=training,

use\_binary\_values=False,

)

with tf.name\_scope("sparse\_continuous"):

shape\_enforced\_input = twml.util.limit\_sparse\_tensor\_size(

sparse\_tf=features["sparse\_continuous"], input\_size\_bits=params.bits, mask\_indices=False

)

normalized\_continuous\_sparse = sparse\_max\_norm(

inputs=shape\_enforced\_input, is\_training=training

)

sparse\_continuous = full\_sparse(

inputs=normalized\_continuous\_sparse,

output\_size=params.embedding\_size,

use\_sparse\_grads=training,

use\_binary\_values=False,

)

output = tf.concat([sparse\_not\_continuous, sparse\_continuous], axis=1)

return output

def get\_features(features: Dict[str, Tensor], training: bool, params: FeaturesParams) -> Tensor:

"""

Performs feature transformations on the dense and sparse features and combine the resulting

tensors into a single one.

"""

with tf.name\_scope("features"):

x = dense\_features(features, training)

tf1.logging.info(f"Dense features: {x.shape}")

if params.sparse\_features:

x = tf.concat([x, sparse\_features(features, training, params.sparse\_features)], axis=1)

return x