"""

Graph class defining methods to obtain key quantities such as:

\* the logits

\* the probabilities

\* the final score

\* the loss function

\* the training operator

"""

from \_\_future\_\_ import annotations

from abc import ABC, abstractmethod

from typing import Any, Dict

from twitter.deepbird.hparam import HParams

import twml

from ..libs.model\_utils import generate\_disliked\_mask

from .params import GraphParams

import tensorflow as tf

import tensorflow.compat.v1 as tf1

class Graph(ABC):

def \_\_init\_\_(self, params: GraphParams):

self.params = params

@abstractmethod

def get\_logits(self, features: Dict[str, tf.Tensor], mode: tf.estimator.ModeKeys) -> tf.Tensor:

pass

def get\_probabilities(self, logits: tf.Tensor) -> tf.Tensor:

return tf.math.cumprod(tf.nn.sigmoid(logits), axis=1, name="probabilities")

def get\_task\_weights(self, labels: tf.Tensor) -> tf.Tensor:

oonc\_label = tf.reshape(labels[:, 0], shape=(-1, 1))

task\_weights = tf.concat([tf.ones\_like(oonc\_label), oonc\_label], axis=1)

n\_labels = len(self.params.tasks)

task\_weights = tf.reshape(task\_weights[:, 0:n\_labels], shape=(-1, n\_labels))

return task\_weights

def get\_loss(self, labels: tf.Tensor, logits: tf.Tensor, \*\*kwargs: Any) -> tf.Tensor:

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

disliked\_mask = generate\_disliked\_mask(labels)

labels = tf.reshape(labels[:, 0:2], shape=[-1, 2])

labels = labels \* tf.cast(tf.logical\_not(disliked\_mask), dtype=labels.dtype)

with tf.name\_scope("task\_weight"):

task\_weights = self.get\_task\_weights(labels)

with tf.name\_scope("batch\_size"):

batch\_size = tf.cast(tf.shape(labels)[0], dtype=tf.float32, name="batch\_size")

weights = task\_weights / batch\_size

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

loss = tf.reduce\_sum(

tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=labels, logits=logits) \* weights,

)

return loss

def get\_score(self, probabilities: tf.Tensor) -> tf.Tensor:

with tf.name\_scope("score\_weight"):

score\_weights = tf.constant([task.score\_weight for task in self.params.tasks])

score\_weights = score\_weights / tf.reduce\_sum(score\_weights, axis=0)

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

score = tf.reshape(tf.reduce\_sum(probabilities \* score\_weights, axis=1), shape=[-1, 1])

return score

def get\_train\_op(self, loss: tf.Tensor, twml\_params) -> Any:

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

learning\_rate = twml\_params.learning\_rate

optimizer = tf1.train.GradientDescentOptimizer(learning\_rate=learning\_rate)

update\_ops = set(tf1.get\_collection(tf1.GraphKeys.UPDATE\_OPS))

with tf.control\_dependencies(update\_ops):

train\_op = twml.optimizers.optimize\_loss(

loss=loss,

variables=tf1.trainable\_variables(),

global\_step=tf1.train.get\_global\_step(),

optimizer=optimizer,

learning\_rate=None,

)

return train\_op

def \_\_call\_\_(

self,

features: Dict[str, tf.Tensor],

labels: tf.Tensor,

mode: tf.estimator.ModeKeys,

params: HParams,

config=None,

) -> Dict[str, tf.Tensor]:

training = mode == tf.estimator.ModeKeys.TRAIN

logits = self.get\_logits(features=features, training=training)

probabilities = self.get\_probabilities(logits=logits)

score = None

loss = None

train\_op = None

if mode == tf.estimator.ModeKeys.PREDICT:

score = self.get\_score(probabilities=probabilities)

output = {"loss": loss, "train\_op": train\_op, "prediction": score}

elif mode in (tf.estimator.ModeKeys.TRAIN, tf.estimator.ModeKeys.EVAL):

loss = self.get\_loss(labels=labels, logits=logits)

if mode == tf.estimator.ModeKeys.TRAIN:

train\_op = self.get\_train\_op(loss=loss, twml\_params=params)

output = {"loss": loss, "train\_op": train\_op, "output": probabilities}

else:

raise ValueError(

f"""

Invalid mode. Possible values are: {tf.estimator.ModeKeys.PREDICT}, {tf.estimator.ModeKeys.TRAIN}, and {tf.estimator.ModeKeys.EVAL}

. Passed: {mode}

"""

)

return output