import tensorflow as tf

physical\_devices = tf.config.list\_physical\_devices('GPU')

for device in physical\_devices:

tf.config.experimental.set\_memory\_growth(device, True)

from twitter.hmli.nimbus.modeling.model\_config import FeatureType, EncodingType, Feature, Model, LogType

from twitter.hmli.nimbus.modeling.feature\_loader import BigQueryFeatureLoader

from twitter.cuad.representation.models.text\_encoder import TextEncoder

from twitter.cuad.representation.models.optimization import create\_optimizer

from twitter.hmli.nimbus.modeling.feature\_encoder import FeatureEncoder

import numpy as np

import pandas as pd

import utils

cat\_names = [

...

]

category\_features = [Feature(name=cat\_name, ftype=FeatureType.CONTINUOUS) for cat\_name in cat\_names]

features = [

Feature(name="tweet\_text\_with\_media\_annotations", ftype=FeatureType.STRING, encoding=EncodingType.BERT),

Feature(name="precision\_nsfw", ftype=FeatureType.CONTINUOUS),

Feature(name="has\_media", ftype=FeatureType.BINARY),

Feature(name="num\_media", ftype=FeatureType.DISCRETE)

] + category\_features

ptos\_prototype = Model(

name='ptos\_prototype',

export\_path="...",

features=features,

)

print(ptos\_prototype)

cq\_loader = BigQueryFeatureLoader(gcp\_project=COMPUTE\_PROJECT)

labels = [

"has\_non\_punitive\_action",

"has\_punitive\_action",

"has\_punitive\_action\_contains\_self\_harm",

"has\_punitive\_action\_encourage\_self\_harm",

"has\_punitive\_action\_episodic",

"has\_punitive\_action\_episodic\_hateful\_conduct",

"has\_punitive\_action\_other\_abuse\_policy",

"has\_punitive\_action\_without\_self\_harm"

]

train\_query = f"""

SELECT

{{feature\_names}},

{",".join(labels)},

...

"""

val\_query = f"""

SELECT

{{feature\_names}},

{",".join(labels)},

...

"""

print(train\_query)

train = cq\_loader.load\_features(ptos\_prototype, "", "", custom\_query=train\_query)

val = cq\_loader.load\_features(ptos\_prototype, "", "", custom\_query=val\_query)

print(train.describe(model=ptos\_prototype))

params = {

'max\_seq\_lengths': 128,

'batch\_size': 196,

'lr': 1e-5,

'optimizer\_type': 'adamw',

'warmup\_steps': 0,

'cls\_dropout\_rate': 0.1,

'epochs': 30,

'steps\_per\_epoch': 5000,

'model\_type': 'twitter\_multilingual\_bert\_base\_cased\_mlm',

'mixed\_precision': True,

}

params

def parse\_labeled\_data(row\_dict):

label = [row\_dict.pop(l) for l in labels]

return row\_dict, label

mirrored\_strategy = tf.distribute.MirroredStrategy()

BATCH\_SIZE = params['batch\_size'] \* mirrored\_strategy.num\_replicas\_in\_sync

train\_ds = train.to\_tf\_dataset().map(parse\_labeled\_data).shuffle(BATCH\_SIZE\*100).batch(BATCH\_SIZE).repeat()

val\_ds = val.to\_tf\_dataset().map(parse\_labeled\_data).batch(BATCH\_SIZE)

for record in train\_ds:

tf.print(record)

break

def get\_positive\_weights():

"""Computes positive weights used for class imbalance from training data."""

label\_weights\_df = utils.get\_label\_weights(

"tos-data-media-full",

project\_id="twttr-abusive-interact-prod",

dataset\_id="tos\_policy"

)

pos\_weight\_tensor = tf.cast(

label\_weights\_df.sort\_values(by='label').positive\_class\_weight,

dtype=tf.float32

)

return pos\_weight\_tensor

pos\_weight\_tensor = get\_positive\_weights()

print(pos\_weight\_tensor)

class TextEncoderPooledOutput(TextEncoder):

def call(self, x):

return super().call([x])["pooled\_output"]

def get\_config(self):

return super().get\_config()

with mirrored\_strategy.scope():

text\_encoder\_pooled\_output = TextEncoderPooledOutput(

params['max\_seq\_lengths'],

model\_type=params['model\_type'],

trainable=True

)

fe = FeatureEncoder(train)

inputs, preprocessing\_head = fe.build\_model\_head(model=ptos\_prototype, text\_encoder=text\_encoder\_pooled\_output)

cls\_dropout = tf.keras.layers.Dropout(params['cls\_dropout\_rate'], name="cls\_dropout")

outputs = cls\_dropout(preprocessing\_head)

outputs = tf.keras.layers.Dense(8, name="output", dtype="float32")(outputs)

model = tf.keras.Model(

inputs=inputs,

outputs=outputs

)

pr\_auc = tf.keras.metrics.AUC(curve="PR", num\_thresholds=1000, multi\_label=True, from\_logits=True)

custom\_loss = lambda y\_true, y\_pred: utils.multilabel\_weighted\_loss(y\_true, y\_pred, weights=pos\_weight\_tensor)

optimizer = create\_optimizer(

init\_lr=params["lr"],

num\_train\_steps=(params["epochs"] \* params["steps\_per\_epoch"]),

num\_warmup\_steps=params["warmup\_steps"],

optimizer\_type=params["optimizer\_type"],

)

if params.get("mixed\_precision"):

optimizer = tf.train.experimental.enable\_mixed\_precision\_graph\_rewrite(optimizer)

model.compile(

optimizer=optimizer,

loss=custom\_loss,

metrics=[pr\_auc]

)

model.weights

model.summary()

pr\_auc.name

import getpass

import wandb

from wandb.keras import WandbCallback

try:

wandb\_key = ...

wandb.login(...)

run = wandb.init(project='ptos\_with\_media',

group='new-split-trains',

notes='tweet text with only (num\_media, precision\_nsfw). on full train set, new split.',

entity='absv',

config=params,

name='tweet-text-w-nsfw-1.1',

sync\_tensorboard=True)

except FileNotFoundError:

print('Wandb key not found')

run = wandb.init(mode='disabled')

import datetime

import os

start\_train\_time = datetime.datetime.now()

print(start\_train\_time.strftime("%m-%d-%Y (%H:%M:%S)"))

checkpoint\_path = os.path.join("...")

print("Saving model checkpoints here: ", checkpoint\_path)

cp\_callback = tf.keras.callbacks.ModelCheckpoint(

filepath=os.path.join(checkpoint\_path, "model.{epoch:04d}.tf"),

verbose=1,

monitor=f'val\_{pr\_auc.name}',

mode='max',

save\_freq='epoch',

save\_best\_only=True

)

early\_stopping\_callback = tf.keras.callbacks.EarlyStopping(patience=7,

monitor=f"val\_{pr\_auc.name}",

mode="max")

model.fit(train\_ds, epochs=params["epochs"], validation\_data=val\_ds, callbacks=[cp\_callback, early\_stopping\_callback],

steps\_per\_epoch=params["steps\_per\_epoch"],

verbose=2)

import tensorflow\_hub as hub

gs\_model\_path = ...

reloaded\_keras\_layer = hub.KerasLayer(gs\_model\_path)

inputs = tf.keras.layers.Input(name="tweet\_\_core\_\_tweet\_\_text", shape=(1,), dtype=tf.string)

output = reloaded\_keras\_layer(inputs)

v7\_model = tf.keras.models.Model(inputs=inputs, outputs=output)

pr\_auc = tf.keras.metrics.AUC(curve="PR", name="pr\_auc")

roc\_auc = tf.keras.metrics.AUC(curve="ROC", name="roc\_auc")

v7\_model.compile(metrics=[pr\_auc, roc\_auc])

model.load\_weights("...")

candidate\_model = model

with mirrored\_strategy.scope():

candidate\_eval = candidate\_model.evaluate(val\_ds)

test\_query = f"""

SELECT

{",".join(ptos\_prototype.feature\_names())},

has\_media,

precision\_nsfw,

{",".join(labels)},

...

"""

test = cq\_loader.load\_features(ptos\_prototype, "", "", custom\_query=test\_query)

test = test.to\_tf\_dataset().map(parse\_labeled\_data)

print(test)

test\_only\_media = test.filter(lambda x, y: tf.equal(x["has\_media"], True))

test\_only\_nsfw = test.filter(lambda x, y: tf.greater\_equal(x["precision\_nsfw"], 0.95))

test\_no\_media = test.filter(lambda x, y: tf.equal(x["has\_media"], False))

test\_media\_not\_nsfw = test.filter(lambda x, y: tf.logical\_and(tf.equal(x["has\_media"], True), tf.less(x["precision\_nsfw"], 0.95)))

for d in [test, test\_only\_media, test\_only\_nsfw, test\_no\_media, test\_media\_not\_nsfw]:

print(d.reduce(0, lambda x, \_: x + 1).numpy())

from notebook\_eval\_utils import SparseMultilabelEvaluator, EvalConfig

from dataclasses import asdict

def display\_metrics(probs, targets, labels=labels):

eval\_config = EvalConfig(prediction\_threshold=0.5, precision\_k=0.9)

for eval\_mode, y\_mask in [("implicit", np.ones(targets.shape))]:

print("Evaluation mode", eval\_mode)

metrics = SparseMultilabelEvaluator.evaluate(

targets, np.array(probs), y\_mask, classes=labels, eval\_config=eval\_config

)

metrics\_df = pd.DataFrame.from\_dict(asdict(metrics)["per\_topic\_metrics"]).transpose()

metrics\_df["pos\_to\_neg"] = metrics\_df["num\_pos\_samples"] / (metrics\_df["num\_neg\_samples"] + 1)

display(metrics\_df.median())

display(metrics\_df)

return metrics\_df

def eval\_model(model, df):

with mirrored\_strategy.scope():

targets = np.stack(list(df.map(lambda x, y: y).as\_numpy\_iterator()), axis=0)

df = df.padded\_batch(BATCH\_SIZE)

preds = model.predict(df)

return display\_metrics(preds, targets)

subsets = {"test": test,

"test\_only\_media": test\_only\_media,

"test\_only\_nsfw": test\_only\_nsfw,

"test\_no\_media": test\_no\_media,

"test\_media\_not\_nsfw": test\_media\_not\_nsfw}

metrics = {}

for name, df in subsets.items():

metrics[name] = eval\_model(candidate\_model, df)

[(name, m.pr\_auc) for name, m in metrics.items()]

for name, x in [(name, m.pr\_auc.to\_string(index=False).strip().split("\n")) for name, m in metrics.items()]:

print(name)

for y in x:

print(y.strip(), end="\t")

print(".")

for d in [test, test\_only\_media, test\_only\_nsfw, test\_no\_media, test\_media\_not\_nsfw]:

print(d.reduce(0, lambda x, \_: x + 1).numpy())