import os

from toxicity\_ml\_pipeline.settings.default\_settings\_tox import LOCAL\_DIR, MAX\_SEQ\_LENGTH

try:

from toxicity\_ml\_pipeline.optim.losses import MaskedBCE

except ImportError:

print('No MaskedBCE loss')

from toxicity\_ml\_pipeline.utils.helpers import execute\_command

import tensorflow as tf

try:

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

except ModuleNotFoundError:

print("No TextEncoder package")

try:

from transformers import TFAutoModelForSequenceClassification

except ModuleNotFoundError:

print("No HuggingFace package")

LOCAL\_MODEL\_DIR = os.path.join(LOCAL\_DIR, "models")

def reload\_model\_weights(weights\_dir, language, \*\*kwargs):

optimizer = tf.keras.optimizers.Adam(0.01)

model\_type = (

"twitter\_bert\_base\_en\_uncased\_mlm"

if language == "en"

else "twitter\_multilingual\_bert\_base\_cased\_mlm"

)

model = load(optimizer=optimizer, seed=42, model\_type=model\_type, \*\*kwargs)

model.load\_weights(weights\_dir)

return model

def \_locally\_copy\_models(model\_type):

if model\_type == "twitter\_multilingual\_bert\_base\_cased\_mlm":

preprocessor = "bert\_multi\_cased\_preprocess\_3"

elif model\_type == "twitter\_bert\_base\_en\_uncased\_mlm":

preprocessor = "bert\_en\_uncased\_preprocess\_3"

else:

raise NotImplementedError

copy\_cmd = """mkdir {local\_dir}

gsutil cp -r ...

gsutil cp -r ..."""

execute\_command(

copy\_cmd.format(model\_type=model\_type, preprocessor=preprocessor, local\_dir=LOCAL\_MODEL\_DIR)

)

return preprocessor

def load\_encoder(model\_type, trainable):

try:

model = TextEncoder(

max\_seq\_lengths=MAX\_SEQ\_LENGTH,

model\_type=model\_type,

cluster="gcp",

trainable=trainable,

enable\_dynamic\_shapes=True,

)

except (OSError, tf.errors.AbortedError) as e:

print(e)

preprocessor = \_locally\_copy\_models(model\_type)

model = TextEncoder(

max\_seq\_lengths=MAX\_SEQ\_LENGTH,

local\_model\_path=f"models/{model\_type}",

local\_preprocessor\_path=f"models/{preprocessor}",

cluster="gcp",

trainable=trainable,

enable\_dynamic\_shapes=True,

)

return model

def get\_loss(loss\_name, from\_logits, \*\*kwargs):

loss\_name = loss\_name.lower()

if loss\_name == "bce":

print("Binary CE loss")

return tf.keras.losses.BinaryCrossentropy(from\_logits=from\_logits)

if loss\_name == "cce":

print("Categorical cross-entropy loss")

return tf.keras.losses.CategoricalCrossentropy(from\_logits=from\_logits)

if loss\_name == "scce":

print("Sparse categorical cross-entropy loss")

return tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=from\_logits)

if loss\_name == "focal\_bce":

gamma = kwargs.get("gamma", 2)

print("Focal binary CE loss", gamma)

return tf.keras.losses.BinaryFocalCrossentropy(gamma=gamma, from\_logits=from\_logits)

if loss\_name == 'masked\_bce':

multitask = kwargs.get("multitask", False)

if from\_logits or multitask:

raise NotImplementedError

print(f'Masked Binary Cross Entropy')

return MaskedBCE()

if loss\_name == "inv\_kl\_loss":

raise NotImplementedError

raise ValueError(

f"This loss name is not valid: {loss\_name}. Accepted loss names: BCE, masked BCE, CCE, sCCE, "

f"Focal\_BCE, inv\_KL\_loss"

)

def \_add\_additional\_embedding\_layer(doc\_embedding, glorot, seed):

doc\_embedding = tf.keras.layers.Dense(768, activation="tanh", kernel\_initializer=glorot)(doc\_embedding)

doc\_embedding = tf.keras.layers.Dropout(rate=0.1, seed=seed)(doc\_embedding)

return doc\_embedding

def \_get\_bias(\*\*kwargs):

smart\_bias\_value = kwargs.get('smart\_bias\_value', 0)

print('Smart bias init to ', smart\_bias\_value)

output\_bias = tf.keras.initializers.Constant(smart\_bias\_value)

return output\_bias

def load\_inhouse\_bert(model\_type, trainable, seed, \*\*kwargs):

inputs = tf.keras.layers.Input(shape=(), dtype=tf.string)

encoder = load\_encoder(model\_type=model\_type, trainable=trainable)

doc\_embedding = encoder([inputs])["pooled\_output"]

doc\_embedding = tf.keras.layers.Dropout(rate=0.1, seed=seed)(doc\_embedding)

glorot = tf.keras.initializers.glorot\_uniform(seed=seed)

if kwargs.get("additional\_layer", False):

doc\_embedding = \_add\_additional\_embedding\_layer(doc\_embedding, glorot, seed)

if kwargs.get('content\_num\_classes', None):

probs = get\_last\_layer(glorot=glorot, last\_layer\_name='target\_output', \*\*kwargs)(doc\_embedding)

second\_probs = get\_last\_layer(num\_classes=kwargs['content\_num\_classes'],

last\_layer\_name='content\_output',

glorot=glorot)(doc\_embedding)

probs = [probs, second\_probs]

else:

probs = get\_last\_layer(glorot=glorot, \*\*kwargs)(doc\_embedding)

model = tf.keras.models.Model(inputs=inputs, outputs=probs)

return model, False

def get\_last\_layer(\*\*kwargs):

output\_bias = \_get\_bias(\*\*kwargs)

if 'glorot' in kwargs:

glorot = kwargs['glorot']

else:

glorot = tf.keras.initializers.glorot\_uniform(seed=kwargs['seed'])

layer\_name = kwargs.get('last\_layer\_name', 'dense\_1')

if kwargs.get('num\_classes', 1) > 1:

last\_layer = tf.keras.layers.Dense(

kwargs["num\_classes"], activation="softmax", kernel\_initializer=glorot,

bias\_initializer=output\_bias, name=layer\_name

)

elif kwargs.get('num\_raters', 1) > 1:

if kwargs.get('multitask', False):

raise NotImplementedError

last\_layer = tf.keras.layers.Dense(

kwargs['num\_raters'], activation="sigmoid", kernel\_initializer=glorot,

bias\_initializer=output\_bias, name='probs')

else:

last\_layer = tf.keras.layers.Dense(

1, activation="sigmoid", kernel\_initializer=glorot,

bias\_initializer=output\_bias, name=layer\_name

)

return last\_layer

def load\_bertweet(\*\*kwargs):

bert = TFAutoModelForSequenceClassification.from\_pretrained(

os.path.join(LOCAL\_MODEL\_DIR, "bertweet-base"),

num\_labels=1,

classifier\_dropout=0.1,

hidden\_size=768,

)

if "num\_classes" in kwargs and kwargs["num\_classes"] > 2:

raise NotImplementedError

return bert, True

def load(

optimizer,

seed,

model\_type="twitter\_multilingual\_bert\_base\_cased\_mlm",

loss\_name="BCE",

trainable=True,

\*\*kwargs,

):

if model\_type == "bertweet-base":

model, from\_logits = load\_bertweet()

else:

model, from\_logits = load\_inhouse\_bert(model\_type, trainable, seed, \*\*kwargs)

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

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

loss = get\_loss(loss\_name, from\_logits, \*\*kwargs)

if kwargs.get('content\_num\_classes', None):

second\_loss = get\_loss(loss\_name=kwargs['content\_loss\_name'], from\_logits=from\_logits)

loss\_weights = {'content\_output': kwargs['content\_loss\_weight'], 'target\_output': 1}

model.compile(

optimizer=optimizer,

loss={'content\_output': second\_loss, 'target\_output': loss},

loss\_weights=loss\_weights,

metrics=[pr\_auc, roc\_auc],

)

else:

model.compile(

optimizer=optimizer,

loss=loss,

metrics=[pr\_auc, roc\_auc],

)

print(model.summary(), "logits: ", from\_logits)

return model