from datetime import datetime

from importlib import import\_module

import os

from toxicity\_ml\_pipeline.data.data\_preprocessing import (

DefaultENNoPreprocessor,

DefaultENPreprocessor,

)

from toxicity\_ml\_pipeline.data.dataframe\_loader import ENLoader, ENLoaderWithSampling

from toxicity\_ml\_pipeline.data.mb\_generator import BalancedMiniBatchLoader

from toxicity\_ml\_pipeline.load\_model import load, get\_last\_layer

from toxicity\_ml\_pipeline.optim.callbacks import (

AdditionalResultLogger,

ControlledStoppingCheckpointCallback,

GradientLoggingTensorBoard,

SyncingTensorBoard,

)

from toxicity\_ml\_pipeline.optim.schedulers import WarmUp

from toxicity\_ml\_pipeline.settings.default\_settings\_abs import GCS\_ADDRESS as ABS\_GCS

from toxicity\_ml\_pipeline.settings.default\_settings\_tox import (

GCS\_ADDRESS as TOX\_GCS,

MODEL\_DIR,

RANDOM\_SEED,

REMOTE\_LOGDIR,

WARM\_UP\_PERC,

)

from toxicity\_ml\_pipeline.utils.helpers import check\_gpu, set\_seeds, upload\_model

import numpy as np

import tensorflow as tf

try:

from tensorflow\_addons.optimizers import AdamW

except ModuleNotFoundError:

print("No TFA")

class Trainer(object):

OPTIMIZERS = ["Adam", "AdamW"]

def \_\_init\_\_(

self,

optimizer\_name,

weight\_decay,

learning\_rate,

mb\_size,

train\_epochs,

content\_loss\_weight=1,

language="en",

scope='TOX',

project=...,

experiment\_id="default",

gradient\_clipping=None,

fold="time",

seed=RANDOM\_SEED,

log\_gradients=False,

kw="",

stopping\_epoch=None,

test=False,

):

self.seed = seed

self.weight\_decay = weight\_decay

self.learning\_rate = learning\_rate

self.mb\_size = mb\_size

self.train\_epochs = train\_epochs

self.gradient\_clipping = gradient\_clipping

if optimizer\_name not in self.OPTIMIZERS:

raise ValueError(

f"Optimizer {optimizer\_name} not implemented. Accepted values {self.OPTIMIZERS}."

)

self.optimizer\_name = optimizer\_name

self.log\_gradients = log\_gradients

self.test = test

self.fold = fold

self.stopping\_epoch = stopping\_epoch

self.language = language

if scope == 'TOX':

GCS\_ADDRESS = TOX\_GCS.format(project=project)

elif scope == 'ABS':

GCS\_ADDRESS = ABS\_GCS

else:

raise ValueError

GCS\_ADDRESS = GCS\_ADDRESS.format(project=project)

try:

self.setting\_file = import\_module(f"toxicity\_ml\_pipeline.settings.{scope.lower()}{project}\_settings")

except ModuleNotFoundError:

raise ValueError(f"You need to define a setting file for your project {project}.")

experiment\_settings = self.setting\_file.experiment\_settings

self.project = project

self.remote\_logdir = REMOTE\_LOGDIR.format(GCS\_ADDRESS=GCS\_ADDRESS, project=project)

self.model\_dir = MODEL\_DIR.format(GCS\_ADDRESS=GCS\_ADDRESS, project=project)

if experiment\_id not in experiment\_settings:

raise ValueError("This is not an experiment id as defined in the settings file.")

for var, default\_value in experiment\_settings["default"].items():

override\_val = experiment\_settings[experiment\_id].get(var, default\_value)

print("Setting ", var, override\_val)

self.\_\_setattr\_\_(var, override\_val)

self.content\_loss\_weight = content\_loss\_weight if self.dual\_head else None

self.mb\_loader = BalancedMiniBatchLoader(

fold=self.fold,

seed=self.seed,

perc\_training\_tox=self.perc\_training\_tox,

mb\_size=self.mb\_size,

n\_outer\_splits="time",

scope=scope,

project=project,

dual\_head=self.dual\_head,

sample\_weights=self.sample\_weights,

huggingface=("bertweet" in self.model\_type),

)

self.\_init\_dirnames(kw=kw, experiment\_id=experiment\_id)

print("------- Checking there is a GPU")

check\_gpu()

def \_init\_dirnames(self, kw, experiment\_id):

kw = "test" if self.test else kw

hyper\_param\_kw = ""

if self.optimizer\_name == "AdamW":

hyper\_param\_kw += f"{self.weight\_decay}\_"

if self.gradient\_clipping:

hyper\_param\_kw += f"{self.gradient\_clipping}\_"

if self.content\_loss\_weight:

hyper\_param\_kw += f"{self.content\_loss\_weight}\_"

experiment\_name = (

f"{self.language}{str(datetime.now()).replace(' ', '')[:-7]}{kw}\_{experiment\_id}{self.fold}\_"

f"{self.optimizer\_name}\_"

f"{self.learning\_rate}\_"

f"{hyper\_param\_kw}"

f"{self.mb\_size}\_"

f"{self.perc\_training\_tox}\_"

f"{self.train\_epochs}\_seed{self.seed}"

)

print("------- Experiment name: ", experiment\_name)

self.logdir = (

f"..."

if self.test

else f"..."

)

self.checkpoint\_path = f"{self.model\_dir}/{experiment\_name}"

@staticmethod

def \_additional\_writers(logdir, metric\_name):

return tf.summary.create\_file\_writer(os.path.join(logdir, metric\_name))

def get\_callbacks(self, fold, val\_data, test\_data):

fold\_logdir = self.logdir + f"\_fold{fold}"

fold\_checkpoint\_path = self.checkpoint\_path + f"\_fold{fold}/{{epoch:02d}}"

tb\_args = {

"log\_dir": fold\_logdir,

"histogram\_freq": 0,

"update\_freq": 500,

"embeddings\_freq": 0,

"remote\_logdir": f"{self.remote\_logdir}\_{self.language}"

if not self.test

else f"{self.remote\_logdir}\_test",

}

tensorboard\_callback = (

GradientLoggingTensorBoard(loader=self.mb\_loader, val\_data=val\_data, freq=10, \*\*tb\_args)

if self.log\_gradients

else SyncingTensorBoard(\*\*tb\_args)

)

callbacks = [tensorboard\_callback]

if "bertweet" in self.model\_type:

from\_logits = True

dataset\_transform\_func = self.mb\_loader.make\_huggingface\_tensorflow\_ds

else:

from\_logits = False

dataset\_transform\_func = None

fixed\_recall = 0.85 if not self.dual\_head else 0.5

val\_callback = AdditionalResultLogger(

data=val\_data,

set\_="validation",

from\_logits=from\_logits,

dataset\_transform\_func=dataset\_transform\_func,

dual\_head=self.dual\_head,

fixed\_recall=fixed\_recall

)

if val\_callback is not None:

callbacks.append(val\_callback)

test\_callback = AdditionalResultLogger(

data=test\_data,

set\_="test",

from\_logits=from\_logits,

dataset\_transform\_func=dataset\_transform\_func,

dual\_head=self.dual\_head,

fixed\_recall=fixed\_recall

)

callbacks.append(test\_callback)

checkpoint\_args = {

"filepath": fold\_checkpoint\_path,

"verbose": 0,

"monitor": "val\_pr\_auc",

"save\_weights\_only": True,

"mode": "max",

"save\_freq": "epoch",

}

if self.stopping\_epoch:

checkpoint\_callback = ControlledStoppingCheckpointCallback(

\*\*checkpoint\_args,

stopping\_epoch=self.stopping\_epoch,

save\_best\_only=False,

)

callbacks.append(checkpoint\_callback)

return callbacks

def get\_lr\_schedule(self, steps\_per\_epoch):

total\_num\_steps = steps\_per\_epoch \* self.train\_epochs

warm\_up\_perc = WARM\_UP\_PERC if self.learning\_rate >= 1e-3 else 0

warm\_up\_steps = int(total\_num\_steps \* warm\_up\_perc)

if self.linear\_lr\_decay:

learning\_rate\_fn = tf.keras.optimizers.schedules.PolynomialDecay(

self.learning\_rate,

total\_num\_steps - warm\_up\_steps,

end\_learning\_rate=0.0,

power=1.0,

cycle=False,

)

else:

print('Constant learning rate')

learning\_rate\_fn = self.learning\_rate

if warm\_up\_perc > 0:

print(f".... using warm-up for {warm\_up\_steps} steps")

warm\_up\_schedule = WarmUp(

initial\_learning\_rate=self.learning\_rate,

decay\_schedule\_fn=learning\_rate\_fn,

warmup\_steps=warm\_up\_steps,

)

return warm\_up\_schedule

return learning\_rate\_fn

def get\_optimizer(self, schedule):

optim\_args = {

"learning\_rate": schedule,

"beta\_1": 0.9,

"beta\_2": 0.999,

"epsilon": 1e-6,

"amsgrad": False,

}

if self.gradient\_clipping:

optim\_args["global\_clipnorm"] = self.gradient\_clipping

print(f".... {self.optimizer\_name} w global clipnorm {self.gradient\_clipping}")

if self.optimizer\_name == "Adam":

return tf.keras.optimizers.Adam(\*\*optim\_args)

if self.optimizer\_name == "AdamW":

optim\_args["weight\_decay"] = self.weight\_decay

return AdamW(\*\*optim\_args)

raise NotImplementedError

def get\_training\_actors(self, steps\_per\_epoch, val\_data, test\_data, fold):

callbacks = self.get\_callbacks(fold=fold, val\_data=val\_data, test\_data=test\_data)

schedule = self.get\_lr\_schedule(steps\_per\_epoch=steps\_per\_epoch)

optimizer = self.get\_optimizer(schedule)

return optimizer, callbacks

def load\_data(self):

if self.project == 435 or self.project == 211:

if self.dataset\_type is None:

data\_loader = ENLoader(project=self.project, setting\_file=self.setting\_file)

dataset\_type\_args = {}

else:

data\_loader = ENLoaderWithSampling(project=self.project, setting\_file=self.setting\_file)

dataset\_type\_args = self.dataset\_type

df = data\_loader.load\_data(

language=self.language, test=self.test, reload=self.dataset\_reload, \*\*dataset\_type\_args

)

return df

def preprocess(self, df):

if self.project == 435 or self.project == 211:

if self.preprocessing is None:

data\_prepro = DefaultENNoPreprocessor()

elif self.preprocessing == "default":

data\_prepro = DefaultENPreprocessor()

else:

raise NotImplementedError

return data\_prepro(

df=df,

label\_column=self.label\_column,

class\_weight=self.perc\_training\_tox if self.sample\_weights == 'class\_weight' else None,

filter\_low\_agreements=self.filter\_low\_agreements,

num\_classes=self.num\_classes,

)

def load\_model(self, optimizer):

smart\_bias\_value = (

np.log(self.perc\_training\_tox / (1 - self.perc\_training\_tox)) if self.smart\_bias\_init else 0

)

model = load(

optimizer,

seed=self.seed,

trainable=self.trainable,

model\_type=self.model\_type,

loss\_name=self.loss\_name,

num\_classes=self.num\_classes,

additional\_layer=self.additional\_layer,

smart\_bias\_value=smart\_bias\_value,

content\_num\_classes=self.content\_num\_classes,

content\_loss\_name=self.content\_loss\_name,

content\_loss\_weight=self.content\_loss\_weight

)

if self.model\_reload is not False:

model\_folder = upload\_model(full\_gcs\_model\_path=os.path.join(self.model\_dir, self.model\_reload))

model.load\_weights(model\_folder)

if self.scratch\_last\_layer:

print('Putting the last layer back to scratch')

model.layers[-1] = get\_last\_layer(seed=self.seed,

num\_classes=self.num\_classes,

smart\_bias\_value=smart\_bias\_value)

return model

def \_train\_single\_fold(self, mb\_generator, test\_data, steps\_per\_epoch, fold, val\_data=None):

steps\_per\_epoch = 100 if self.test else steps\_per\_epoch

optimizer, callbacks = self.get\_training\_actors(

steps\_per\_epoch=steps\_per\_epoch, val\_data=val\_data, test\_data=test\_data, fold=fold

)

print("Loading model")

model = self.load\_model(optimizer)

print(f"Nb of steps per epoch: {steps\_per\_epoch} ---- launching training")

training\_args = {

"epochs": self.train\_epochs,

"steps\_per\_epoch": steps\_per\_epoch,

"batch\_size": self.mb\_size,

"callbacks": callbacks,

"verbose": 2,

}

model.fit(mb\_generator, \*\*training\_args)

return

def train\_full\_model(self):

print("Setting up random seed.")

set\_seeds(self.seed)

print(f"Loading {self.language} data")

df = self.load\_data()

df = self.preprocess(df=df)

print("Going to train on everything but the test dataset")

mini\_batches, test\_data, steps\_per\_epoch = self.mb\_loader.simple\_cv\_load(df)

self.\_train\_single\_fold(

mb\_generator=mini\_batches, test\_data=test\_data, steps\_per\_epoch=steps\_per\_epoch, fold="full"

)

def train(self):

print("Setting up random seed.")

set\_seeds(self.seed)

print(f"Loading {self.language} data")

df = self.load\_data()

df = self.preprocess(df=df)

print("Loading MB generator")

i = 0

if self.project == 435 or self.project == 211:

mb\_generator, steps\_per\_epoch, val\_data, test\_data = self.mb\_loader.no\_cv\_load(full\_df=df)

self.\_train\_single\_fold(

mb\_generator=mb\_generator,

val\_data=val\_data,

test\_data=test\_data,

steps\_per\_epoch=steps\_per\_epoch,

fold=i,

)

else:

raise ValueError("Sure you want to do multiple fold training")

for mb\_generator, steps\_per\_epoch, val\_data, test\_data in self.mb\_loader(full\_df=df):

self.\_train\_single\_fold(

mb\_generator=mb\_generator,

val\_data=val\_data,

test\_data=test\_data,

steps\_per\_epoch=steps\_per\_epoch,

fold=i,

)

i += 1

if i == 3:

break