import os

import logging

from dataclasses import dataclass, field

from typing import Dict, List, Optional

import json

import torch

from transformers import T5Tokenizer, HfArgumentParser

from datasets import Dataset

import sys

logger = logging.getLogger(\_\_name\_\_)

@dataclass

class DataTrainingArguments:

"""

Arguments pertaining to what data we are going to input our model for training and eval.

"""

model\_type: Optional[str] = field(

default='QG',

metadata={"help": "if the model is trained for generation from text (QG) or generation from answer and text (QAG)"},

)

train\_file: Optional[str] = field(

default=None,

metadata={"help": "name for train dataset. Must be a .json file"},

)

valid\_file: Optional[str] = field(

default=None,

metadata={"help": "name for valid dataset. Must be a .json file"},

)

max\_source\_length: Optional[int] = field(

default=512,

metadata={"help": "Max input length for the source text"},

)

max\_target\_length: Optional[int] = field(

default=32,

metadata={"help": "Max input length for the target text"},

)

tokenizer: Optional[str] = field(

default="t5-small",

metadata={"help": "tokenizer used for the data"},

)

class DataProcessor:

def \_\_init\_\_(self, tokenizer, model\_type = "QG", max\_source\_length=512, max\_target\_length=32):

self.tokenizer = tokenizer

self.max\_source\_length = max\_source\_length

self.max\_target\_length = max\_target\_length

self.hl\_token = "<hl>"

self.sep\_token = "<sep>"

self.model\_type = model\_type

def process(self, dataset):

if self.model\_type == "QG":

dataset = dataset.map(self.\_add\_special\_tokens\_QG)

else:

dataset = dataset.map(self.\_add\_special\_tokens\_QAG)

dataset = dataset.map(self.\_convert\_to\_features, batched= True)

return dataset

def \_add\_special\_tokens\_QG(self, ex):

ex['source\_text'] = f"generate questions: {ex['source\_text']} </s>"

sub = ''

if type(ex['target\_text']) is list: #Preparing data with multiple questions per context

for k in ex['target\_text']:

sub = sub + f"{k} <sep> "

ex['target\_text'] = sub + '</s>'

else: #Preparing data with one question per context

ex['target\_text'] = f"{ex['target\_text']} </s>"

return ex

def \_add\_special\_tokens\_QAG(self,ex): #This preprocessing is going to be modified later depending on the results I got

ex['source\_text'] = f"answer: {ex['answers']} context: {ex['source\_text']}"

if type(ex['target\_text']) is list:

for k in ex['target\_text']:

sub = sub + f"{k} <sep> "

ex['target\_text'] = sub + '</s>'

else:

ex['target\_text'] = f"{ex['target\_text']} </s>"

return ex

# encode the input

def \_convert\_to\_features(self, data):

source\_encoding = self.tokenizer.batch\_encode\_plus(

data['source\_text'],

max\_length=self.max\_source\_length,

padding='max\_length',

pad\_to\_max\_length=True,

truncation=True,

)

target\_encoding = self.tokenizer.batch\_encode\_plus(

data['target\_text'],

max\_length=self.max\_target\_length,

padding='max\_length',

pad\_to\_max\_length=True,

truncation=True,

)

encodings = {

'source\_ids': source\_encoding['input\_ids'],

'target\_ids': target\_encoding['input\_ids'],

'attention\_mask': source\_encoding['attention\_mask'],

}

return encodings

def main(args\_file=None):

parser = HfArgumentParser((DataTrainingArguments,))

#Parsing the arguments

if (len(sys.argv) == 2 and sys.argv[1].endswith(".json")) or args\_file is not None:

# If we pass only one argument to the script and it's the path to a json file,

# let's parse it to get our arguments.

args\_file\_path = os.path.abspath(sys.argv[1]) if args\_file is None else args\_file

data\_args = parser.parse\_json\_file(json\_file=args\_file\_path)[0]

else:

data\_args = parser.parse\_args\_into\_dataclasses()[0]

logging.basicConfig(

format="%(asctime)s - %(levelname)s - %(name)s - %(message)s",

datefmt="%m/%d/%Y %H:%M:%S",

level=logging.INFO

)

#Loading the torch files

with open(data\_args.train\_file) as f:

train\_dataset = json.load(f)

if data\_args.valid\_file is not None:

with open(data\_args.valid\_file) as f:

valid\_dataset = json.load(f)

#Defining the tokenizer and adding the special tokens

#If the input is a dict with keys being questions and item context:

if 'source\_text' not in train\_dataset.keys():

source\_text = []

target\_text = []

for source, target in train\_dataset.items():

source\_text.append(source)

target\_text.append(target)

train\_dataset = {"source\_text":source\_text , "target\_text":target\_text}

if data\_args.valid\_file is not None:

source\_text = []

target\_text = []

for source, target in valid\_dataset.items():

source\_text.append(source)

target\_text.append(target)

valid\_dataset = {"source\_text":source\_text , "target\_text":target\_text}

tokenizer = T5Tokenizer.from\_pretrained(data\_args.tokenizer)

tokenizer.add\_tokens(['<sep>', '<hl>'])

# Processing the data

processor = DataProcessor(

tokenizer,

model\_type=data\_args.model\_type,

max\_source\_length=data\_args.max\_source\_length,

max\_target\_length=data\_args.max\_target\_length

)

train\_dataset = Dataset.from\_dict(train\_dataset)

train\_dataset = processor.process(train\_dataset)

columns = ["source\_ids", "target\_ids", "attention\_mask"]

train\_dataset.set\_format(type='torch', columns=columns)

train\_file = f"train\_data\_{data\_args.model\_type}\_T5.pt"

torch.save(train\_dataset, train\_file)

logger.info(f"saved train dataset at {train\_file}")

if data\_args.valid\_file is not None:

valid\_dataset = Dataset.from\_dict(valid\_dataset)

valid\_dataset = processor.process(valid\_dataset)

valid\_dataset.set\_format(type='torch', columns=columns)

valid\_file = f"valid\_data\_{data\_args.model\_type}\_T5.pt"

torch.save(valid\_dataset, valid\_file)

logger.info(f"saved validation dataset at {valid\_file}")

tokenizer\_path = f"{data\_args.model\_type}\_qg\_tokenizer"

if not os.path.exists(tokenizer\_path):

os.mkdir(tokenizer\_path)

tokenizer.save\_pretrained(tokenizer\_path)

logger.info(f"saved tokenizer at {tokenizer\_path}")

def prepare\_data(args\_dict):

with open("args\_temp.json", 'w') as f:

json.dump(args\_dict, f)

main(args\_file="args\_temp.json")

os.remove("args\_temp.json")

if \_\_name\_\_ == "\_\_main\_\_":

main()