# -\*- coding: utf-8 -\*-

"""

Created on Fri Jan 25 18:46:40 2019

@author: zhan

"""

from model import Span\_labler, Pred\_finder

import torch.optim as optim

import torch

import torch.nn as nn

from utils import to\_var, cuda\_num

import random

import json

from load\_pretrained\_embedding import Glove

from utils import make\_span\_candidates

import spacy

import en\_core\_web\_sm

from pororo import Pororo

random.seed(10)

syntax\_flag = True

data\_fn = "data/student\_file.json"

model\_fn = "model/model.params\_span\_"+str(syntax\_flag)

pred\_model\_fn = "model/pred\_model.params\_span\_"+str(syntax\_flag)

confidence\_score = 0.1

pred\_confidence\_score = 0.8

hidden\_dim = 200

label\_size = 4

n\_layers = 2

pos\_dim = 20

dp\_dim = 20

epoches = 1

learning\_rate = 0.01

decay\_rate = 0.005

batch\_size = 20

max\_pred\_len = 5

max\_arg\_len = 10

decay\_every = 100

def lr\_decay(optimizer, epoch, batch\_num, total\_batch\_num, decay\_rate, init\_lr):

lr = init\_lr \* ((1-decay\_rate)\*\*(int((epoch\*total\_batch\_num+batch\_num)/decay\_every)))

if batch\_num % decay\_every == 0:

print(" Learning rate is setted as:", lr)

for param\_group in optimizer.param\_groups:

param\_group['lr'] = lr

return optimizer

def span\_candidates\_filter(candidates,max\_len):

new\_candidates = []

for candidate in candidates:

if candidate[1] - candidate[0] + 1 <= max\_len:

new\_candidates.append(candidate)

return new\_candidates

def syntax\_check(sent\_spacy,rel,dic, span):

parent = []

word=[]

add=0

#span[1]-> 형태소 단위로 분리한 span의 마지막 부분

for i in dic:

if span[1]!=i[0][-1]:#단어에 속한 형태소 중 마지막에 속하면

add+=1

else:# 단어 중간 부분에 속하는 형태소이면

word[1]=span[1]

if add==0:

return False

if span[1] - span[0] == 0:

return True

for i in range(span[0], span[1]+1):

word=""

for word, index in dic.items():#해당 형태소를 dic value에서 찾고 word로 바꾸기

for idx in index:

if idx==i:

word=word+1

for i in rel:

if(i[0]==word):#해당 단어의 바로 위에 연결되어 있는 노드

if(i[i]==-1):

parent.append(i[0]-1)

else:

parent.append(i[1]-1)

for i in range(span[0], span[1]+1):

if ((dic[parent[i-span[0]]][0] >= span[0]) and (dic[parent[i-span[0]]][-1] <= span[1])) or (i in parent):

pass

else:

return False

return True

def pred\_span\_candidates\_filter(sent\_spacy,rel,dic,candidates,max\_len):

candidates = span\_candidates\_filter(candidates,max\_len)

new\_candidates = []

for candidate in candidates:

if syntax\_check(sent\_spacy,rel,dic, candidate):

new\_candidates.append(candidate)

return new\_candidates

def arg\_span\_candidates\_filter(sent\_spacy,rel,dic,candidates,max\_len,pred\_span):

candidates = span\_candidates\_filter(candidates,max\_len)

new\_candidates = [pred\_span]

pred\_words = [i for i in range(pred\_span[0],pred\_span[1]+1)]

for candidate in candidates:

add\_flag = True

for word\_idx in pred\_words:

if word\_idx >= candidate[0] and word\_idx <= candidate[1]:

add\_flag = False

break

if syntax\_check(sent\_spacy,rel,dic, candidate) == False:

add\_flag = False

if add\_flag:

new\_candidates.append(candidate)

return new\_candidates

def get\_span\_head(sent\_spacy, rel,dic,candidates, dp2index, dp\_idx):

candidates\_head = []

candidates\_head\_dp = []

for span in candidates:

word\_list=[]

for word, index in dic.items():

for i in index:

if i==span[0]:

word\_list.append(word)

elif i==span[1]:

word\_list.append(word)

set\_word=set(word\_list)#중복값제거

word\_list=list(set\_word)

save\_root=[]

for i in range(len(word\_list)):

if rel[word\_list[i]][1]-1>=word\_list[0] && rel[word\_list[i]][1]-1<=word\_list[-1]:#word 중에 dep 존재한다면

save\_root.append(rel[word\_list[i]][1]-1)

if rel[word\_list[i]][1]==-1:

save\_root.append(rel[word\_list[i]][0]-1)

set\_root=set(save\_root)#중복값제거

save\_root=list(set\_root)

root=0

if len(save\_root)==1:

root=save\_root[0]

else:

for i in range(len(save\_root)):

if rel[save\_root[i]][1]-1<word\_list[0] && rel[save\_root[i]][1]-1<word\_list[-1]:#root 중복 제거

root=rel[save\_root[i]][1]-1

span\_spacy = sent\_spacy[span[0]: span[1]+1]

if(rel[root][1]==-1):

candidates\_head.append(rel[root][0]-1)

else:

candidates\_head.append(rel[root][1]-1)

if dp\_idx[root] in dp2index:

candidates\_head\_dp.append(dp2index[dp\_idx[root]])

else:

candidates\_head\_dp.append(dp2index["<UNK>"])

return candidates\_head, candidates\_head\_dp

def makeBatch(data,batch\_size):

batch\_data = []

random.shuffle(data)

batch\_num = int(len(data)/batch\_size)

print("batch\_numc=처음:",batch\_num)

for i in range(batch\_num):

batch\_data.append(data[i\*batch\_size:(i+1)\*batch\_size])

if len(data) % batch\_size != 0:

batch\_data.append(data[(i+1)\*batch\_size:])

return batch\_data

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

#nlp = spacy.load("en\_core\_web\_md")

#nlp = en\_core\_web\_sm.load()

dp= Pororo(task="dep\_parse", lang="ko") #뽀로로 dp

pos = Pororo(task="pos", lang="ko") # 뽀로로 pos tagging

with open(data\_fn) as f:

train\_data = json.load(f)

with open("data/pos2index.json") as f:

pos2index = json.load(f)

with open("data/dp2index.json") as f:

dp2index = json.load(f)

print("load data done")

glove = Glove("word2vec/kor.tsv")

model = Span\_labler(glove, label\_size, len(pos2index), pos\_dim, hidden\_dim, n\_layers, len(dp2index), dp\_dim, syntax\_flag)

pred\_model = Pred\_finder(glove, label\_size, len(pos2index), pos\_dim, hidden\_dim, n\_layers)

if torch.cuda.is\_available():

model = model.cuda(cuda\_num)

pred\_model = pred\_model.cuda(cuda\_num)

optimizer1 = optim.Adam(model.parameters(), lr=learning\_rate)

optimizer2 = optim.Adam(pred\_model.parameters(), lr=learning\_rate)

criterion1 = nn.NLLLoss()

criterion2 = nn.NLLLoss(weight=to\_var(torch.FloatTensor([1,5.5])))

for epoch in range(epoches):

print("epoch출력;",epoch)

batch\_data = makeBatch(train\_data,batch\_size)

print("batch\_data:",batch\_data)

for batch\_num,batch in enumerate(batch\_data):

print("batch num:",batch\_num,"batch;",batch)

optimizer1.zero\_grad()

optimizer2.zero\_grad()

if batch\_num % decay\_every == 0:

optimizer1 = lr\_decay(optimizer1, epoch, batch\_num, len(batch\_data),decay\_rate, learning\_rate)

optimizer2 = lr\_decay(optimizer2, epoch, batch\_num, len(batch\_data),decay\_rate, learning\_rate)

loss1 = to\_var(torch.FloatTensor([0]))

loss2 = to\_var(torch.FloatTensor([0]))

for instance in batch:

do\_pred = True

# sent\_spacy = nlp(instance["sentence"].strip())

sent\_spacy=instance["sentence"]##문장분류

sent\_idx = instance["sentence2index"]

text = text.strip()

sent\_word = text.split()#################공백 기준으로 자른거(단어)

pos\_token=pos(sent\_spacy)

dic={}########word와 형태소 딕셔너리로 매핑 인덱스

i=0

num=0

save=0

sent\_pos=[]

for word in pos\_token:

if word[1]=='SPACE':

dic[num]=list(range(save,i))

save=i

num+=1

i-=1

else:

sent\_pos.append(word[0])

i+=1

dic[num]=list(range(save,i))

if len(sent\_spacy) == len(sent\_idx):

arg\_candidates = make\_span\_candidates(len(instance["sentence2index"]))

pos\_idx = []

dp\_idx = []

#pos\_token=pos(sent\_spacy)#########포로로로 문장 pos tagging

for token in pos\_token:

if token[1] in pos2index:

pos\_idx.append(pos2index[token[1]])

else:

pos\_idx.append(pos2index["<UNK>"])

rel=[]#############dep\_parse 관계 저장한 배열

dp\_token=dp(sent\_spacy)#########포로로로 문장 dp, pos 갯수 맞춰서

for token in dp\_token:

rel.append([token[0],token[2]])###############dep\_parse 관계 저장

if token[3] in dp2index:

for a in range(len(dic)):

for b in dic[a]:

dp\_idx.append(dp2index[token[3]])

else:

for a in range(len(dic)):

for b in dic[a]:

dp\_idx.append(dp2index["<UNK>"])

# for token in sent\_spacy:

# if token.tag\_ in pos2index:

# pos\_idx.append(pos2index[token.tag\_])

# else:

# pos\_idx.append(pos2index["<UNK>"])

# if token.dep\_ in dp2index:

# dp\_idx.append(dp2index[token.dep\_])

# else:

# dp\_idx.append(dp2index["<UNK>"])

tuples = instance["tuples"]

word\_input = to\_var(torch.LongTensor(sent\_idx))

pos\_input = to\_var(torch.LongTensor(pos\_idx))

dp\_input = to\_var(torch.LongTensor(dp\_idx))

gold\_pred\_idx\_all = []

pred\_candidates = pred\_span\_candidates\_filter(sent\_spacy,rel,dic,arg\_candidates,max\_pred\_len)

gold\_pred\_idx = 0

if len(tuples) == 0:

do\_pred = False

for tuple\_ in tuples:

gold\_pred = tuple\_["rel\_pos"]

if gold\_pred[-1] == -1:

continue

if tuple\_["score"] > pred\_confidence\_score:

if gold\_pred[1] - gold\_pred[0] + 1 > max\_pred\_len or gold\_pred not in pred\_candidates:

do\_pred = False

continue

gold\_pred\_idx\_all.append(pred\_candidates.index(gold\_pred))

else:

do\_pred = False

if tuple\_["score"] > confidence\_score:

arg\_candidates = arg\_span\_candidates\_filter(sent\_spacy,arg\_candidates,max\_arg\_len,gold\_pred)

if len(tuple\_["arg0\_pos"]) == 0:

arg0\_gold = gold\_pred\_idx

else:

if tuple\_["arg0\_pos"][1] - tuple\_["arg0\_pos"][0] + 1 > max\_arg\_len:

continue

else:

if tuple\_["arg0\_pos"] in arg\_candidates:

arg0\_gold = arg\_candidates.index(tuple\_["arg0\_pos"])

else:

continue

args\_gold = [gold\_pred\_idx, gold\_pred\_idx, gold\_pred\_idx]

go\_flag = True

for i,arg in enumerate(tuple\_["args\_pos"]):

if i == 3:

break

else:

if arg[1] - arg[0] + 1 > max\_arg\_len:

go\_flag = False

break

else:

if arg in arg\_candidates:

args\_gold[i] = arg\_candidates.index(arg)

else:

go\_flag = False

break

if go\_flag:

candidates\_head, candidates\_head\_dp = get\_span\_head(sent\_spacy, arg\_candidates, dp2index, dp\_idx)

gold\_label = to\_var(torch.LongTensor([arg0\_gold, args\_gold[0], args\_gold[1], args\_gold[2]]))

arg\_out = model(word\_input, pos\_input, dp\_input, arg\_candidates, candidates\_head, candidates\_head\_dp, gold\_pred\_idx)

loss1 += tuple\_["score"] \* criterion1(arg\_out.transpose(0,1),gold\_label)

else:

do\_pred = False

if do\_pred:

gold\_pred = []

for i in range(len(pred\_candidates)):

if i in gold\_pred\_idx\_all:

gold\_pred.append(1)

else:

gold\_pred.append(0)

gold\_pred = to\_var(torch.LongTensor(gold\_pred))

pred\_out = pred\_model(word\_input, pos\_input, pred\_candidates)

loss2 += criterion2(pred\_out, gold\_pred)

print("model1 ","epoch:",epoch,"batch",batch\_num,"loss:",loss1.item()/len(batch))

print("model2 ","epoch:",epoch,"batch",batch\_num,"loss:",loss2.item()/len(batch))

if loss1.item() != 0:

loss1.backward()

optimizer1.step()

if loss2.item() != 0:

loss2.backward()

optimizer2.step()

if (batch\_num + 1) % 100 == 0:

torch.save(model.state\_dict(), model\_fn)

torch.save(pred\_model.state\_dict(), pred\_model\_fn)

print("epoch:",epoch,'done')

torch.save(model.state\_dict(), model\_fn+'\_e'+str(epoch))