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

"""

Created on Wed Dec 19 14:44:01 2018

@author: win 10

"""

from model import Span\_labler,Pred\_finder

from load\_pretrained\_embedding import Glove

import json

import torch

from utils import make\_span\_candidates, to\_var, cuda\_num

import spacy

from main\_span import pred\_span\_candidates\_filter, arg\_span\_candidates\_filter, get\_span\_head

from main\_span import hidden\_dim, label\_size, n\_layers, pos\_dim, dp\_dim

import en\_core\_web\_sm

syntax\_flag = True

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

#model\_fn= "model/model.params\_span\_False"

#model\_fn = "model/model.params\_span\_last\_"+str(syntax\_flag) +"\_e0"

#pred\_model\_fn = "model/pred\_model.params\_span\_False"

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

def prepare\_sentence(sent\_spacy,glove,pos2index,dp2index):

arg\_candidates = make\_span\_candidates(len(sent\_spacy))

pred\_candidates = pred\_span\_candidates\_filter(sent\_spacy, arg\_candidates,5)

sent\_idx = [glove.get\_word\_index(word.text) for word in sent\_spacy]

pos\_idx = [pos2index[word.tag\_] for word in sent\_spacy]

dp\_idx = []

for word in sent\_spacy:

if word.dep\_ in dp2index:

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

else:

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

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

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

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

return word\_input, pos\_input, dp\_input, pred\_candidates, arg\_candidates, sent\_spacy

def is\_overlap(span1,span2):

start = [span1[0],span2[0]]

end = [span1[1],span2[1]]

if min(end)<max(start):

return False

else:

return True

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

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

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

glove = Glove("data/glove.6B.100d.txt")

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

pos2index = json.load(f)

pos\_size = len(pos2index)

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

dp2index = json.load(f)

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

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

if torch.cuda.is\_available():

model = model.cuda(cuda\_num)

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

model.load\_state\_dict(torch.load(model\_fn))

model.eval()

pred\_model.load\_state\_dict(torch.load(pred\_model\_fn))

pred\_model.eval()

sentence = "나는 밥을 먹고나서 학교에 갔다 ."

sent\_spacy = nlp(sentence)

word\_input, pos\_input, dp\_input, pred\_candidates, arg\_candidates, sent\_spacy = prepare\_sentence(sent\_spacy, glove, pos2index, dp2index)

#找出谓词

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

\_, max\_index = torch.max(pred\_out,dim=1)

gold\_pred\_all = []

for i,v in enumerate(max\_index):

if v.data.item() == 1:

pair = pred\_candidates[i]

gold\_pred\_all.append(pair)

print("---")

new\_gold\_pred\_all = []

mark = [0 for t in gold\_pred\_all]

for pred\_id, gold\_pred in enumerate(gold\_pred\_all):

if mark[pred\_id] == 0:

combine\_flag = False

for j in range(len(gold\_pred\_all)):

if mark[j] == 0:

if gold\_pred[1] + 1 == gold\_pred\_all[j][0]:

new\_gold\_pred\_all.append([gold\_pred[0], gold\_pred\_all[j][1]])

mark[pred\_id] = 1

mark[j] = 1

combine\_flag = True

break

if gold\_pred[0] - 1 == gold\_pred\_all[j][1]:

new\_gold\_pred\_all.append([gold\_pred\_all[j][0],gold\_pred[1]])

mark[pred\_id] = 1

mark[j] = 1

combine\_flag = True

break

if combine\_flag == False and gold\_pred not in new\_gold\_pred\_all:

new\_gold\_pred\_all.append(gold\_pred)

mark[pred\_id] = 1

for gold\_pred in new\_gold\_pred\_all:

do\_flag = True

for pair in new\_gold\_pred\_all:

if pair != gold\_pred:

if (pair[0] <= gold\_pred[0]) and (pair[1] >= gold\_pred[1]):

do\_flag = False

print("skip")

break

if do\_flag == False:

continue

print(sent\_spacy[gold\_pred[0]:gold\_pred[1]+1].text)

fill = [0,0,0,0]

fill\_span = [[-1,-1],[-1,-1],[-1,-1],[-1,-1]]

arg\_candidates = arg\_span\_candidates\_filter(sent\_spacy,arg\_candidates,500,gold\_pred)

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

gold\_pred\_idx = arg\_candidates.index(gold\_pred)

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

arg\_out = arg\_out.transpose(0,1) #(num\_label, num\_span)

arg\_out = arg\_out.cpu().detach().numpy()

flatten = {}

for i in range(arg\_out.shape[0]):

for j in range(arg\_out.shape[1]):

flatten[i+10\*j] = arg\_out[i][j]

flatten\_sorted = sorted(flatten.items(), key=lambda item:item[1], reverse=True)

for pair in flatten\_sorted:

if fill == [1,1,1,1]:

break

else:

label = int(pair[0] % 10)

span\_idx = int(pair[0] / 10)

if fill[label] == 1:

continue

elif span\_idx == gold\_pred\_idx:

fill[label] = 1

else:

span = arg\_candidates[span\_idx]

add\_flag = True

for selected\_span in fill\_span:

if is\_overlap(span, selected\_span):

add\_flag = False

break

if label == 0:

if span[0] >= gold\_pred[0]:

add\_flag = False

if label == 1:

if span[0] <= gold\_pred[0]:

add\_flag = False

if add\_flag:

fill[label] = 1

fill\_span[label] = span

print("arg",str(label),": ",sent\_spacy[span[0]:span[1]+1].text)

print("-----")