In [1]:
```
import plac
import logging
import argparse
import sys
import json
import os
import json
import pickle
```

In [2]:
```
def tsv_to_json_format(input_path,output_path,unknown_label):
    try:
        f=open(input_path,'r') # input file
        fp=open(output_path, 'w') # output file
        data_dict={}
        annotations =[]
        label_dict={}
        s=''
        start=0
        for line in f:
            if line[0:len(line)-1]!='.\tO':
                word,entity=line.split('\t')
                s+=word+" "
                entity=entity[:len(entity)-1]
                if entity!=unknown_label:
                    if len(entity) != 1:
                        d={}
                        d['text']=word
                        d['start']=start
                        d['end']=start+len(word)-1
                        try:
                            label_dict[entity].append(d)
                        except:
                            label_dict[entity]=[]
                            label_dict[entity].append(d)
                start+=len(word)+1
            else:
                data_dict['content']=s
                s=''
                label_list=[]
                for ents in list(label_dict.keys()):
                    for i in range(len(label_dict[ents])):
                        if(label_dict[ents][i]['text']!=''):
                            l=[ents,label_dict[ents][i]]
                            for j in range(i+1,len(label_dict[ents])):
                                if(label_dict[ents][i]['text']==label_dict[ents][j
                                    di={}
                                    di['start']=label_dict[ents][j]['start']
                                    di['end']=label_dict[ents][j]['end']
                                    di['text']=label_dict[ents][i]['text']
                                    l.append(di)
                                    label_dict[ents][j]['text']=''
                            label_list.append(l)

                for entities in label_list:
                    label={}
                    label['label']=[entities[0]]
                    label['points']=entities[1:]
                    annotations.append(label)
                data_dict['annotation']=annotations
                annotations=[]
                json.dump(data_dict, fp)
                fp.write('\n')
                data_dict={}
                start=0
                label_dict={}
    except Exception as e:
        logging.exception("Unable to process file" + "\n" + "error = " + str(e))
        return None

tsv_to_json_format("ner_corpus_260.json','abc')
```

```
  File "/tmp/ipykernel_363/2369530672.py", line 61
    tsv_to_json_format("ner_corpus_260.json','abc')
                       ^
SyntaxError: unterminated string literal (detected at line 61)
```

In [3]:
```
def main(input_file=None, output_file=None):
        training_data = []
        lines=[]
        with open(input_file, 'r') as f:
            lines = f.readlines()

        for line in lines:
            data = json.loads(line)
            text = data['content']
            entities = []
            for annotation in data['annotation']:
                point = annotation['points'][0]
                labels = annotation['label']
                if not isinstance(labels, list):
                    labels = [labels]

                for label in labels:
                    entities.append((point['start'], point['end'] + 1 ,label))


            training_data.append((text, {"entities" : entities}))

        #print(training_data)

        with open(output_file, 'w') as fp:
            json.dump(training_data, fp)

main("ner_corpus_260_training.json")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_363/1458516920.py in <cell line: 28>()
     26             json.dump(training_data, fp)
     27
---> 28 main("ner_corpus_260_training.json")

/tmp/ipykernel_363/1458516920.py in main(input_file, output_file)
      7         for line in lines:
      8             data = json.loads(line)
----> 9             text = data['content']
     10             entities = []
     11             for annotation in data['annotation']:

TypeError: list indices must be integers or slices, not str
```

In [4]:
```
from __future__ import unicode_literals, print_function
import pickle
import plac
import random
from pathlib import Path
import spacy
from spacy.util import minibatch, compounding


# New entity labels
# Specify the new entity labels which you want to add here
LABEL = ['I-geo', 'B-geo', 'I-art', 'B-art', 'B-tim', 'B-nat', 'B-eve', 'O', 'I-per

"""
geo = Geographical Entity
org = Organization
per = Person
gpe = Geopolitical Entity
tim = Time indicator
art = Artifact
eve = Event
nat = Natural Phenomenon
"""
# Loading training data
with open ('ner_corpus_260_training.json', 'rb') as fp:
    TRAIN_DATA = json.load(fp)


def main(model, new_model_name, output_dir, n_iter=10):
    """Setting up the pipeline and entity recognizer, and training the new entity.'
    if model is not None:
        nlp = spacy.load(model)  # load existing spacy model
        print("Loaded model '%s'" % model)
    else:
        nlp = spacy.blank('en')  # create blank Language class
        print("Created blank 'en' model")
    reset_weights = False
    if 'ner' not in nlp.pipe_names:
        ner = nlp.create_pipe('ner')
        nlp.add_pipe(ner)
        reset_weights = True
    else:
        ner = nlp.get_pipe('ner')

    for i in LABEL:
        ner.add_label(i)   # Add new entity labels to entity recognizer

    if model is None or reset_weights:
        optimizer = nlp.begin_training()
    else:
        optimizer = nlp.entity.create_optimizer()

    # Get names of other pipes to disable them during training to train only NER
    other_pipes = [pipe for pipe in nlp.pipe_names if pipe != 'ner']
    with nlp.disable_pipes(*other_pipes):  # only train NER
        for itn in range(n_iter):
            random.shuffle(TRAIN_DATA)
            losses = {}
            batches = minibatch(TRAIN_DATA, size=compounding(4., 32., 1.001))
            for batch in batches:
                texts, annotations = zip(*batch)
                nlp.update(texts, annotations, sgd=optimizer, drop=0.35,
                           losses=losses)
            print('Losses', losses)

    # Test the trained model
    test_text = 'Gianni Infantino is the president of FIFA.'
    doc = nlp(test_text)
    print("Entities in '%s'" % test_text)
    for ent in doc.ents:
        print(ent.label_, ent.text)

    # Save model
    if output_dir is not None:
        output_dir = Path(output_dir)
        if not output_dir.exists():
            output_dir.mkdir()
        nlp.meta['name'] = new_model_name  # rename model
        nlp.to_disk(output_dir)
        print("Saved model to", output_dir)

        # Test the saved model
        print("Loading from", output_dir)
        nlp2 = spacy.load(output_dir)
        doc2 = nlp2(test_text)
        for ent in doc2.ents:
            print(ent.label_, ent.text)


main('en',"new_model"," ",10)
```

```
/usr/local/lib/python3.10/site-packages/torch/cuda/__init__.py:503: UserWarning: C
an't initialize NVML
  warnings.warn("Can't initialize NVML")
2023-08-06 08:49:40.914980: I tensorflow/core/platform/cpu_feature_guard.cc:193] T
his TensorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDN
N) to use the following CPU instructions in performance-critical operations:  AVX2
AVX512F FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compil
er flags.
2023-08-06 08:49:43.495347: E tensorflow/compiler/xla/stream_executor/cuda/cuda_dr
iver.cc:267] failed call to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable device i
s detected
VOC-NOTICE: GPU memory for this assignment is capped at 1024MiB
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
/tmp/ipykernel_363/2425770210.py in <cell line: 90>()
     88 
     89 
---> 90 main('en',"new_model"," ",10)

/tmp/ipykernel_363/2425770210.py in main(model, new_model_name, output_dir, n_ite
r)
     30     """Setting up the pipeline and entity recognizer, and training the new
entity."""
     31     if model is not None:
---> 32         nlp = spacy.load(model)  # load existing spacy model
     33         print("Loaded model '%s'" % model)
     34     else:

/usr/local/lib/python3.10/site-packages/spacy/__init__.py in load(name, vocab, dis
able, enable, exclude, config)
     52     RETURNS (Language): The loaded nlp object.
     53     """
---> 54     return util.load_model(
     55         name,
     56         vocab=vocab,

/usr/local/lib/python3.10/site-packages/spacy/util.py in load_model(name, vocab, d
isable, enable, exclude, config)
    446         return load_model_from_path(name, **kwargs)  # type: ignore[arg-ty
pe]
    447     if name in OLD_MODEL_SHORTCUTS:
--> 448         raise IOError(Errors.E941.format(name=name, full=OLD_MODEL_SHORTCU
TS[name]))  # type: ignore[index]
    449     raise IOError(Errors.E050.format(name=name))
    450 

OSError: [E941] Can't find model 'en'. It looks like you're trying to load a model
from a shortcut, which is obsolete as of spaCy v3.0. To load the model, use its fu
ll name instead:

nlp = spacy.load("en_core_web_sm")

For more details on the available models, see the models directory: https://spacy.
io/models. If you want to create a blank model, use spacy.blank: nlp = spacy.blank
("en")
```

In [ ]:

In [ ]:

In [ ]: