《自然语言处理》

-命名实体识别实验



华为技术有限公司

|  |
| --- |
| 版权所有 © 华为技术有限公司 2021。 保留一切权利。  非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。  商标声明  C:\Users\jwx341670\Desktop\华为标志 Huawei Logo 2018\竖版标志Vertical Version\PNG\HW_POS_RBG_Vertical-150ppi.png 和其他华为商标均为华为技术有限公司的商标。  本文档提及的其他所有商标或注册商标，由各自的所有人拥有。  注意  您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。  由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。 |

|  |  |
| --- | --- |
| 华为技术有限公司 | |
| 地址： | 深圳市龙岗区坂田华为总部办公楼 邮编：518129 |
| 网址： | http://[e](http://e.huawei.com/).huawei.com |

目录

[1 实验总览 2](#_Toc68648309)

[1.1 实验背景 2](#_Toc68648310)

[1.2 实验目的 2](#_Toc68648311)

[1.3 实验清单 2](#_Toc68648312)

[2 命名实体识别实验 3](#_Toc68648313)

[2.1 实验简介 3](#_Toc68648314)

[2.2实验环境 3](#_Toc68648315)

[2.3背景知识 3](#_Toc68648316)

[2.4实验步骤 5](#_Toc68648317)

[2.4.1 实验准备 5](#_Toc68648321)

[2.4.2 实验过程 9](#_Toc68648322)

[2.5实验总结 17](#_Toc68648323)

# 实验总览

## 实验背景

命名实体识别（Named Entity Recognition, NER) 是NLP领域最经典的任务之一，实体识别提取一些专有的实体，如人名，地名，机构名，公司名，药品名等，实体识别广泛应用于搜索，对话，问答，知识库构建等场景中。

## 实验目的

本章实验的主要目的是掌握命名实体识别（NER）相关基础知识点，使用开源工具以及MindSpore框架实现命名实体识别模型，加深对相关理论的理解。

## 实验清单

表格：实验、简述、难度、软件环境、硬件环境。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验 | 简述 | 难度 | 软件环境 | 开发环境 |
| 命名实体识别实验（BERT+CRF） | 使用BERT+CRF实现命名实体识别 | 高级 | Python3.7、MindSpore1.3 | ModelArts Ascend Notebook环境 |

# 命名实体识别实验

## 实验简介

命名实体识别（Named Entity Recognition, NER) 是NLP领域最经典的任务之一，实体识别提取一些专有的实体，如人名，地名，机构名，公司名，药品名等，实体识别广泛应用于搜索，对话，问答，知识库构建等场景中。基于transformer的BERT预训练模型相对于循环神经网络（Recurrent Neural Network，RNN）, 长短期记忆网络（Long Short-Term Memory, LSTM）以及传统的隐马尔科夫模型（Hidden Markov Model, HMM）、条件随机场（Conditional Random Field, CRF）能够更好地捕捉上下文语义，从而提升识别性能。

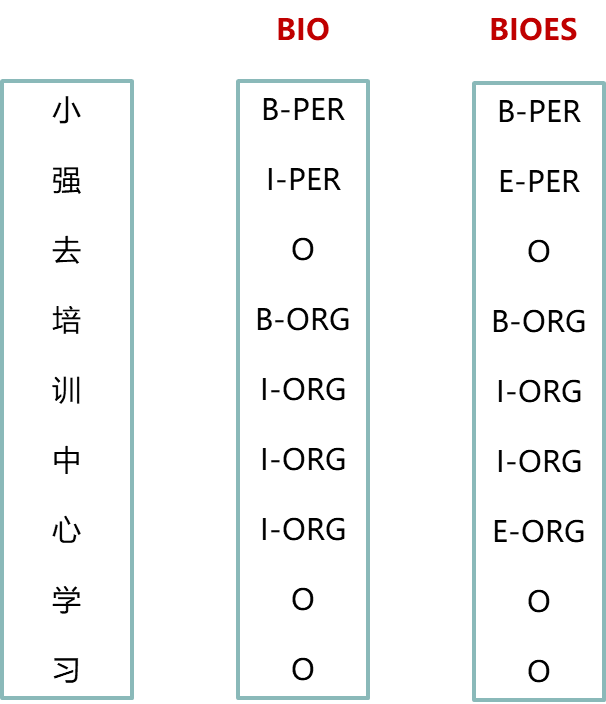
本实验在华为云ModelArts平台上使用MindSpore1.3实现BERT+CRF命名实体识别模型。

## 2.2实验环境

ModelArts Ascend Notebook环境，环境设置参考《华为云ModelArt Ascend环境配置》

## 2.3背景知识

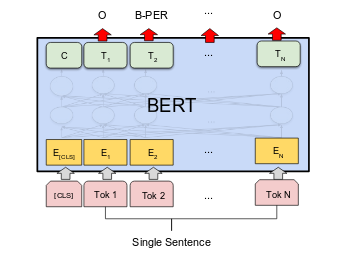
命名实体识别不仅要找出实体的位置，还要对实体进行分类。位置和类别通过标签来表达，命名实体识别数据标注格式有BIO和BIOES两种：



命名实体识别数据标注格式

上面这个例子中BIO或BIOES代表实体的位置，B代表begin, I代表inner, O代表other, E代表end, S代表single(表示这个实体只有一个词)，PER或ORG代表实体的类别，PER代表Person (人名)，ORG代表Organization （机构名）。

BERT做命名实体识别的基本原理：在每一个输入对应的位置上输出，做一个多分类，得到对应的标签。



BERT命名实体识别

## 2.4实验步骤



### 实验准备

OBS创建项目文件夹

使用OBS Brower+登录OBS



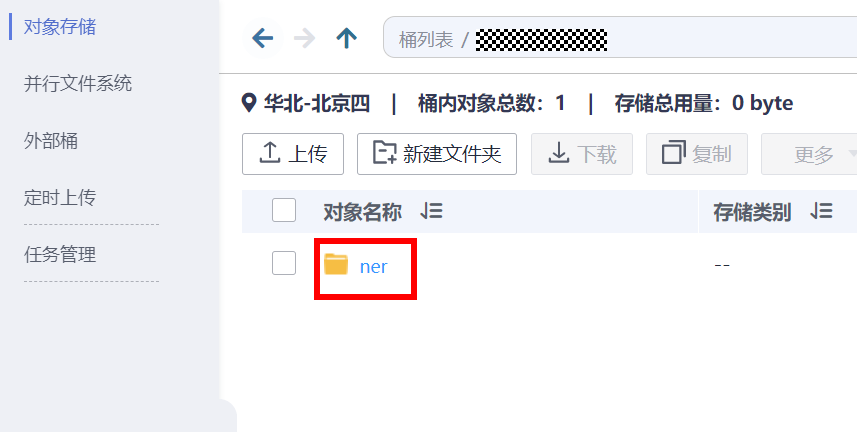
OBS Brower+登录

进入之后创建一个obs桶，用于存放实验所需的文件。然后在这个桶路径下，我们创建一个 “ner”的目录用于本实验数据的存放：



创建项目文件夹

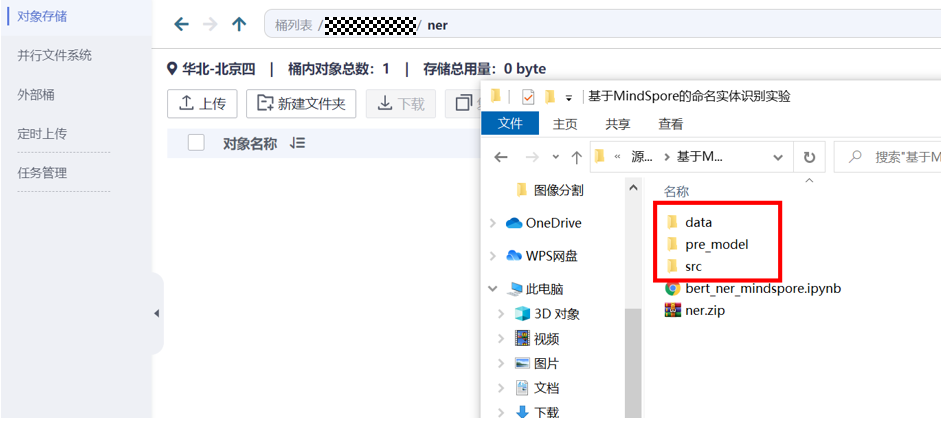
创建完如下所示：



项目文件夹创建成功

上传实验源码及数据

进入刚创建的“ner”文件夹，上传源码及数据至该目录下。直接将需要上传的文件和文件夹拖到该目录下即可。



上传实验数据

进入Modelarts，选择Notebook

在华为云主页搜索Modelarts并点击“进入控制台”，或者通过以下网址进入：

https://console.huaweicloud.com/modelarts/?region=cn-north-4#/dashboard

点击左侧导航栏的“开发环境”，选择“Notebook”



选择Notebook

创建Notebook

点击创建按钮来创建一个新的Notebook，选择如下配置：

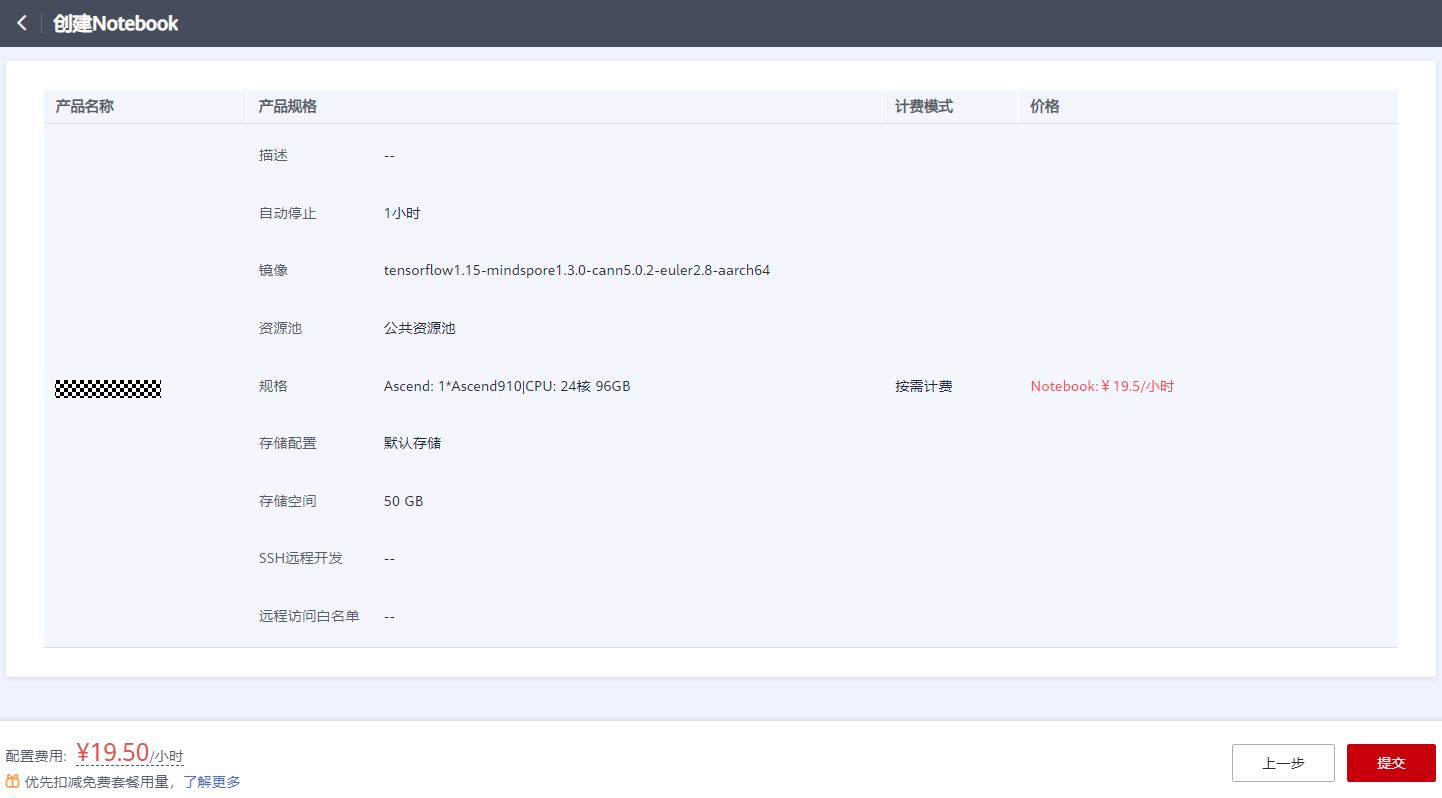
名称：建议使用ner

工作环境：选择tensorflow1.15-mindspore1.3.0-cann5.0.2-euler2.8-aarch64

规格：Ascend: 1\*Ascend910|CPU:24核96GB

存储配置：默认存储

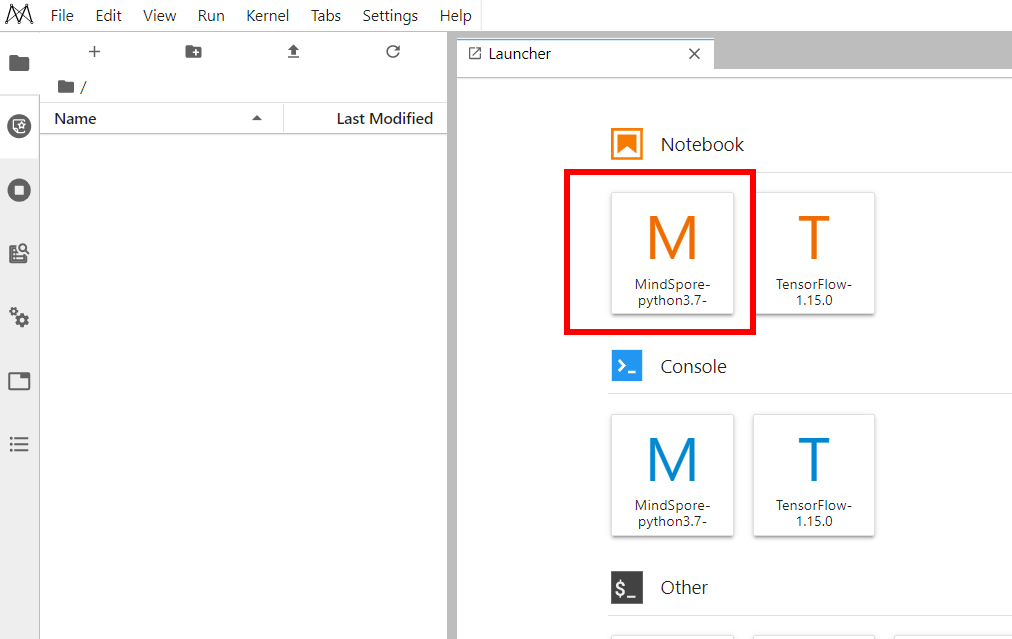
点击“下一步”，确认规格如下后选择提交：



创建Notebook

进入Notebook实验环境

当Notebook状态变为“运行中”时，点击右侧“打开”按钮打开Notebook。选择右侧“MindSpore-python3.7-aarch64”按钮，进入Notebook实验环境。



进入Notebook实验环境

### 实验过程

同步数据和源码到本地容器

进行实验前，我们需要将obs上的文件下载至容器本地环境中，此处需将obs桶名称换成自己创建的obs桶名称。

输入：

import moxing as mox

mox.file.copy\_parallel(src\_url="s3://你的obs桶名称/ner/src/", dst\_url='./src/')

mox.file.copy\_parallel(src\_url="s3://你的obs桶名称/ner/data/", dst\_url='./data/')

mox.file.copy\_parallel(src\_url="s3://你的obs桶名称/ner/pre\_model/", dst\_url='./pre\_model/')

导入依赖库

输入：

import os

import argparse

import numpy as np

import json

from sklearn.metrics import classification\_report # 需放在前面导入

import mindspore.nn as nn

from easydict import EasyDict as edict

import mindspore.common.dtype as mstype

from mindspore import context

from mindspore import log as logger

from mindspore.common.tensor import Tensor

import mindspore.dataset as de

from mindspore.ops import operations as P

import mindspore.dataset.transforms.c\_transforms as C

from mindspore.nn.wrap.loss\_scale import DynamicLossScaleUpdateCell

from mindspore.nn.optim import AdamWeightDecay

from mindspore.train.model import Model

from mindspore.train.callback import CheckpointConfig, ModelCheckpoint, TimeMonitor, LossMonitor

from mindspore.train.serialization import load\_checkpoint, load\_param\_into\_net

from mindspore.common.initializer import TruncatedNormal

from src import tokenization

from src.CRF import CRF

from src.CRF import postprocess

from src.cluener\_evaluation import process\_one\_example\_p, label\_generation

from src.utils import BertLearningRate

from src.bert\_for\_finetune import BertFinetuneCell

from src.config import optimizer\_cfg

from src.bert\_model import BertConfig, BertModel

设置运行环境

输入：

context.set\_context(mode=context.GRAPH\_MODE, device\_target="Ascend")

定义参数配置

输入：

cfg = edict({

'is\_train': True,

'num\_labels': 41,

'schema\_file': r'./data/clue\_ner/schema.json',

'ckpt\_prefix': 'bert-ner-crf', # 'bert-ner' 'bert-ner-crf'

'train\_file': r'./data/clue\_ner/train.tf\_record',

'eval\_file': r'./data/clue\_ner/dev.tf\_record',

'use\_crf': True,

'epoch\_num': 5,

'batch\_size': 16,

'ckpt\_dir': 'ckpt',

'pre\_training\_ckpt': './pre\_model/bert\_base.ckpt',

'finetune\_ckpt': './ckpt/bert-ner-crf-5\_671.ckpt',

'label2id\_file': './data/clue\_ner/label2id.json',

'vocab\_file': './data/vocab.txt',

'eval\_out\_file': 'ner\_crf\_result.txt' # ner\_result.txt ner\_crf\_result.txt

})

bert\_net\_cfg = BertConfig(

seq\_length=128,

vocab\_size=21128,

hidden\_size=768,

num\_hidden\_layers=12,

num\_attention\_heads=12,

intermediate\_size=3072,

hidden\_act="gelu",

hidden\_dropout\_prob=0.1,

attention\_probs\_dropout\_prob=0.1,

max\_position\_embeddings=512,

type\_vocab\_size=2,

initializer\_range=0.02,

use\_relative\_positions=False,

dtype=mstype.float32,

compute\_type=mstype.float16

)

定义数据集加载函数

输入：

def get\_dataset(data\_file, schema\_file, batch\_size):

'''

get dataset

'''

ds = de.TFRecordDataset([data\_file], schema\_file, columns\_list=["input\_ids", "input\_mask","segment\_ids", "label\_ids"])

type\_cast\_op = C.TypeCast(mstype.int32)

ds = ds.map(input\_columns="segment\_ids", operations=type\_cast\_op)

ds = ds.map(input\_columns="input\_mask", operations=type\_cast\_op)

ds = ds.map(input\_columns="input\_ids", operations=type\_cast\_op)

ds = ds.map(input\_columns="label\_ids", operations=type\_cast\_op)

# apply shuffle operation

buffer\_size = 960

ds = ds.shuffle(buffer\_size=buffer\_size)

# apply batch operations

ds = ds.batch(batch\_size, drop\_remainder=True)

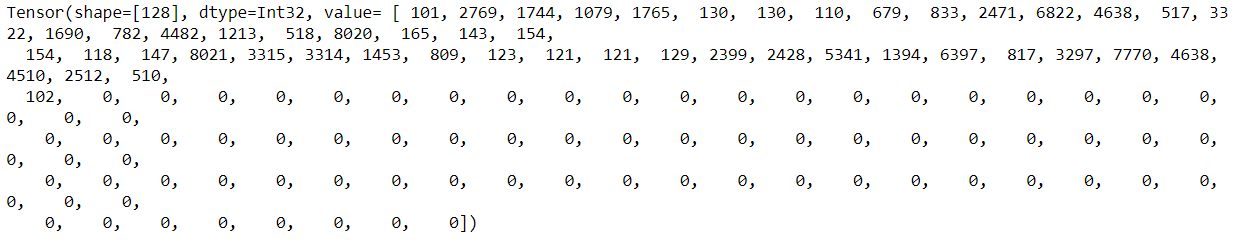
return ds

数据集测试：

输入：

next(get\_dataset(cfg.train\_file, cfg.schema\_file, batch\_size=1).create\_dict\_iterator())['input\_ids'][0]

输出：



预处理后的数据展示

定义BertNER模型

输入：

class BertNER(nn.Cell):

"""

Train interface for sequence labeling finetuning task.

"""

def \_\_init\_\_(self, config, batch\_size, is\_training, num\_labels=11, use\_crf=False, tag\_to\_index=None, dropout\_prob=0.0,

use\_one\_hot\_embeddings=False):

super(BertNER, self).\_\_init\_\_()

self.bert = BertModel(config, is\_training, use\_one\_hot\_embeddings)

self.cast = P.Cast()

self.weight\_init = TruncatedNormal(config.initializer\_range)

self.log\_softmax = P.LogSoftmax(axis=-1)

self.dtype = config.dtype

self.num\_labels = num\_labels

self.dense\_1 = nn.Dense(config.hidden\_size, self.num\_labels, weight\_init=self.weight\_init,

has\_bias=True).to\_float(config.compute\_type)

self.dropout = nn.Dropout(1 - dropout\_prob)

self.reshape = P.Reshape()

self.shape = (-1, config.hidden\_size)

self.use\_crf = use\_crf

self.origin\_shape = (batch\_size, config.seq\_length, self.num\_labels)

if use\_crf:

if not tag\_to\_index:

raise Exception("The dict for tag-index mapping should be provided for CRF.")

self.loss = CRF(tag\_to\_index, batch\_size, config.seq\_length, is\_training)

else:

self.loss = CrossEntropyCalculation(is\_training)

self.num\_labels = num\_labels

self.use\_crf = use\_crf

def construct(self, input\_ids, input\_mask, token\_type\_id, label\_ids):

sequence\_output, \_, \_ = \

self.bert(input\_ids, token\_type\_id, input\_mask)

seq = self.dropout(sequence\_output)

seq = self.reshape(seq, self.shape)

logits = self.dense\_1(seq)

logits = self.cast(logits, self.dtype)

if self.use\_crf:

return\_value = self.reshape(logits, self.origin\_shape)

loss = self.loss(return\_value, label\_ids)

else:

return\_value = self.log\_softmax(logits)

loss = self.loss(return\_value, label\_ids, self.num\_labels)

return loss

加载词汇-ID映射表

输入：

tag\_to\_index = json.loads(open(cfg.label2id\_file).read())

if cfg.use\_crf:

print(tag\_to\_index)

max\_val = len(tag\_to\_index)

tag\_to\_index["<START>"] = max\_val

tag\_to\_index["<STOP>"] = max\_val + 1

number\_labels = len(tag\_to\_index)

else:

number\_labels = cfg.num\_labels

定义训练函数

输入：

def train():

'''

finetune function

'''

# BertNER train for sequence labeling

netwithloss = BertNER(bert\_net\_cfg, cfg.batch\_size, True, num\_labels=number\_labels,

use\_crf=cfg.use\_crf,

tag\_to\_index=tag\_to\_index, dropout\_prob=0.1)

dataset = get\_dataset(data\_file=cfg.train\_file, schema\_file=cfg.schema\_file, batch\_size=cfg.batch\_size)

steps\_per\_epoch = dataset.get\_dataset\_size()

print('steps\_per\_epoch:',steps\_per\_epoch)

# optimizer

steps\_per\_epoch = dataset.get\_dataset\_size()

lr\_schedule = BertLearningRate(learning\_rate=optimizer\_cfg.AdamWeightDecay.learning\_rate,

end\_learning\_rate=optimizer\_cfg.AdamWeightDecay.end\_learning\_rate,

warmup\_steps=int(steps\_per\_epoch \* cfg.epoch\_num \* 0.1),

decay\_steps=steps\_per\_epoch \* cfg.epoch\_num,

power=optimizer\_cfg.AdamWeightDecay.power)

params = netwithloss.trainable\_params()

decay\_params = list(filter(optimizer\_cfg.AdamWeightDecay.decay\_filter, params))

other\_params = list(filter(lambda x: not optimizer\_cfg.AdamWeightDecay.decay\_filter(x), params))

group\_params = [{'params': decay\_params, 'weight\_decay': optimizer\_cfg.AdamWeightDecay.weight\_decay},

{'params': other\_params, 'weight\_decay': 0.0}]

optimizer = AdamWeightDecay(group\_params, lr\_schedule, eps=optimizer\_cfg.AdamWeightDecay.eps)

# load checkpoint into network

ckpt\_config = CheckpointConfig(save\_checkpoint\_steps=steps\_per\_epoch, keep\_checkpoint\_max=1)

ckpoint\_cb = ModelCheckpoint(prefix=cfg.ckpt\_prefix, directory=cfg.ckpt\_dir, config=ckpt\_config)

param\_dict = load\_checkpoint(cfg.pre\_training\_ckpt)

load\_param\_into\_net(netwithloss, param\_dict)

update\_cell = DynamicLossScaleUpdateCell(loss\_scale\_value=2\*\*32, scale\_factor=2, scale\_window=1000)

netwithgrads = BertFinetuneCell(netwithloss, optimizer=optimizer, scale\_update\_cell=update\_cell)

model = Model(netwithgrads)

callbacks = [TimeMonitor(dataset.get\_dataset\_size()), LossMonitor(), ckpoint\_cb]

model.train(cfg.epoch\_num, dataset, callbacks=callbacks, dataset\_sink\_mode=True)

启动训练

输入：

train()

加载离线模型

输入：

netwithloss = BertNER(bert\_net\_cfg, 1, False, num\_labels=number\_labels,

use\_crf=cfg.use\_crf,

tag\_to\_index=tag\_to\_index)

netwithloss.set\_train(False)

param\_dict = load\_checkpoint(cfg.finetune\_ckpt)

load\_param\_into\_net(netwithloss, param\_dict)

model = Model(netwithloss)

tokenizer\_ = tokenization.FullTokenizer(vocab\_file=cfg.vocab\_file)

定义测试集评估函数

输入：

def eval():

'''

evaluation function

'''

dataset = get\_dataset(cfg.eval\_file, cfg.schema\_file, 1)

columns\_list = ["input\_ids", "input\_mask", "segment\_ids", "label\_ids"]

y\_true, y\_pred = [], []

for data in dataset.create\_dict\_iterator():

input\_data = []

for i in columns\_list:

input\_data.append(Tensor(data[i]))

input\_ids, input\_mask, token\_type\_id, label\_ids = input\_data

logits = model.predict(input\_ids, input\_mask, token\_type\_id, label\_ids)

if cfg.use\_crf:

backpointers, best\_tag\_id = logits

best\_path = postprocess(backpointers, best\_tag\_id)

logit\_ids = []

for ele in best\_path:

logit\_ids.append(ele)

else:

logits = logits.asnumpy()

logit\_ids = np.argmax(logits, axis=-1)

for ids in label\_ids.asnumpy():

y\_true.extend(ids)

for ids in logit\_ids:

y\_pred.extend(ids)

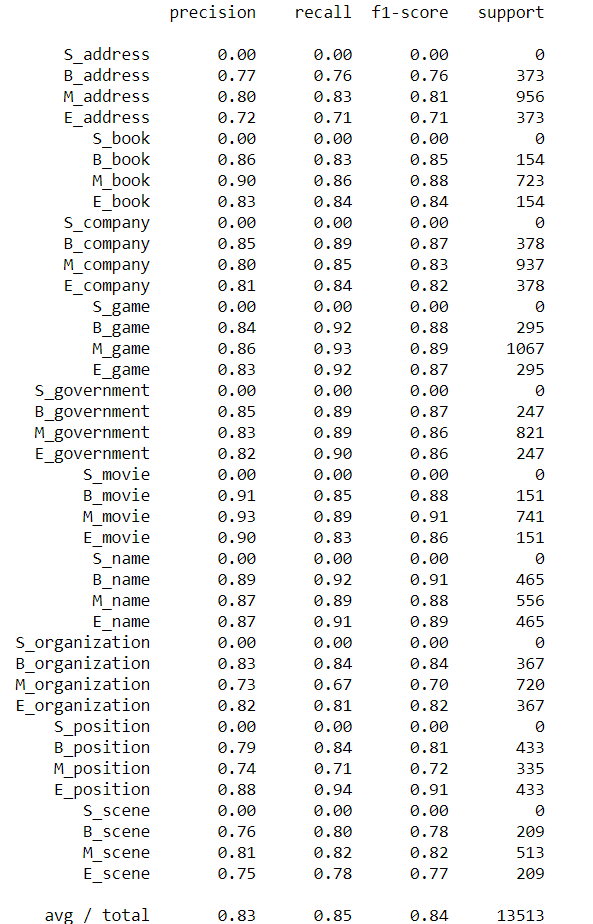
print(classification\_report(y\_true, y\_pred, labels=range(1, 41), target\_names=list(tag\_to\_index.keys())[1:41]))

启动测试集评估

输入：

eval()

输出：



验证结果展示

定义在线推理函数

输入：

def inference(text):

"""

online inference

"""

feature = process\_one\_example\_p(tokenizer\_, cfg.vocab\_file, text, max\_seq\_len=bert\_net\_cfg.seq\_length)

input\_ids, input\_mask, token\_type\_id = feature

input\_ids = Tensor(np.array(input\_ids), mstype.int32)

input\_mask = Tensor(np.array(input\_mask), mstype.int32)

token\_type\_id = Tensor(np.array(token\_type\_id), mstype.int32)

if cfg.use\_crf:

backpointers, best\_tag\_id = model.predict(input\_ids, input\_mask, token\_type\_id, Tensor(1))

best\_path = postprocess(backpointers, best\_tag\_id)

logits = []

for ele in best\_path:

logits.extend(ele)

ids = logits

else:

logits = model.predict(input\_ids, input\_mask, token\_type\_id, Tensor(1))

ids = logits.asnumpy()

ids = np.argmax(ids, axis=-1)

ids = list(ids)

res = label\_generation(text=text, probs=ids, tag\_to\_index=tag\_to\_index)

return res

在线推理测试

输入：

inference("温格的球队终于又踢了一场经典的比赛，2比1战胜曼联之后枪手仍然留在了夺冠集团之内，")

输出：



实体识别结果展示1

输入：

inference("郑阿姨就赶到文汇路排队拿钱，希望能将缴纳的一万余元学费拿回来，顺便找校方或者教委要个说法。")

输出：



图片识别结果展示2

## 2.5实验总结

本章实验在华为云ModelArts平台上使用MindSpore完整的实现了一个基于BERT+CRF的命名实体识别模型，BERT+CRF是当前性能最好的序列标注架构之一，通过实验使学员了解最前沿的算法模型，也进一步加深对命名实体识别，BERT等相关理论的理解。学员可以在该实验实验的基础上进一步开发适合自己业务场景的实体识别模型。