基于 LSTM-CRF 模型的命名实体识别实验书

一、 项目目标与总述

1.1 项目目标

实现 LSTM-CRF 模型，使用人民日报 98 版数据集[2]进行训练，输入文本输出其中实体及类别。

1.2 项目总述

本项目基于 Pytorch 框架构建，个人在参考代码实现时，发现没有一个版本是适合自己的，或多或少都不舒服。所以本项目旨在自己实现，搭建一个符合 Pytorch 规范的 lstm-CRF 实体识别模型。

这里的规范主要指的是,数据加载使用标准的 Dataset 和 DataLoader 接口，数据处理、model、以及训练代码分别各自实现，不混合使用。个人看到的都是老旧的，混合写在一起的，很不舒服。

当然，本项目有部分参考网上实现，例如数据处理的正则化部分，以及模型搭建的 CRF 部分。

1.3 项目运行

（1)运行 data_process.py,主要进行数据 BIO 标注转化,以及数据集切分，以及 char-embedding 的训练保存。

（2）运行 build_data.py, 主要进行 vocab 的构建。(vocab 内不包含[UNK]和[PAD])。

（3）运行 train.py 训练模型。

二、 项目搭建流程

2.1 数据预处理

（1）数据 BIO 化

此处本项目处理为多实体识别，因此标注时，对新闻数据中的标注规则如下，标注方法主要为正则表达式：

/nr、例如，毛，泽东，人名，标识为[B-PER],[I-PER]

/ns、例如：西安，地点名字，标识为[B-LOC],[I-LOC]

/nt、例如：中共中央，组织名，标识为[B-ORG],[I-ORG]

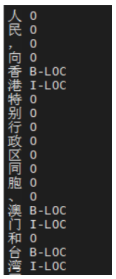

图 1　BIO 标注后结构

（2）数据切分与 vocab 构建

此处主要按照 4 ：1 划分数据集，训练集有 14964 条数据。

Vocab 主要为字库，大约有 3478 个字，此处不包含[PAD],[UNK],后续代码会加上，这里不加是为了方便。因为代码默认[PAD]为 0，[UNK]为 1，这里加上会导致和后面不对应。因为一般 vocab 是用 set 构建的,无序,所以生成 word_to_id 时会乱。

2.2 Dataset 与 DataLoader

（1）RMRB_DataSet

这是 pytorch 加载数据时需要实现的一个标准类，个人在此类内主要实现了三个部分：读取数据，去掉小于长度为 5 的数据（这里是因为后续在跑代码时发现有的句子长度过短），char 变为 id 形式。

（2）RMRB_collate_fn

pytorch 加载数据需要实现的另一个类 dataloader,此类主要用于产生 batch 大小的数据，个人实现的方法主要在 datalaoder 流程最后一步，对一批数据进行处理，此处的处理逻辑主要有：对句子按照 batch 内最大长度填充，并产生对应 mask；转化成 tensor 数据类型并返回。

个人喜爱这类加载数据方式，清晰明了。当然这类操作的缺点是对于大规模数据，Dataset 将所有数据读进内存的操作可能不友好，需要另行考虑。但是在本数据集上没有此问题。

2.3 预训练的 char embedding

这里利用了 Word2Vec 的向量，Word2Vec 具体来说是两种不同思想实现的：CBOW（Continuous Bag of Words）和 Skip-gram。CBOW 的目标是根据上下文来

预测当前词语的概率，且上下文所有的词对当前词出现概率的影响的权重是一样的，因此叫 continuous bag-of-words 模型。如在袋子中取词，取出数量足够的词就可以了，至于取出的先后顺序是无关紧要的。Skip-gram 刚好相反：根据当前词语来预测上下文的概率。

这两种方法都利用人工神经网络作为它们的分类算法。起初每个单词都是一个随机 N 维向量。经过训练之后，该算法利用 CBOW 或者 Skip-gram 的方法获得了每个单词的最优向量。

本项目在使用 lstm+crf 这类模型时，对应的 embedding 矩阵会默认随机初始化。一般而言天然认为这样可能会效果不太好，所以可以预先使用 word2vec 算法，预先训练一个 embedding 矩阵，模型构建完成后，将此矩阵按照 id 顺序调整，整体赋值给 embedding 层。当然，在实验环节个人会验证下加与不加的区别。

对应到代码部分，训练词向量本项目用的 3.6.0 版本的 gensim 训练，具体代码在 data_process.py 代码内，下面代码主要展示个人对于训练好的 embedding，加载以及实用的情况。

个人赋值 embedding 的代码主要如下：

```
# 加载转化后的文件
wvmodel = KeyedVectors.load_word2vec_format(config.filename_glove)
weight = torch.zeros(config.vocab_size, config.embedding_dim)
for i in range(len(wvmodel.vocab)):
    try:
        index = vocabs_to_id[wvmodel.index2word[i]]
    except:
        continue
    weight[index, :] = torch.from_numpy(wvmodel.get_vector(id_to_vocabs[vocabs_to_id[wvmodel.index2word[i]]]))
logger.info("Load pretrained embedding")
model.vocab_embedding.weight.data.copy_(weight)
```

2.4 Model

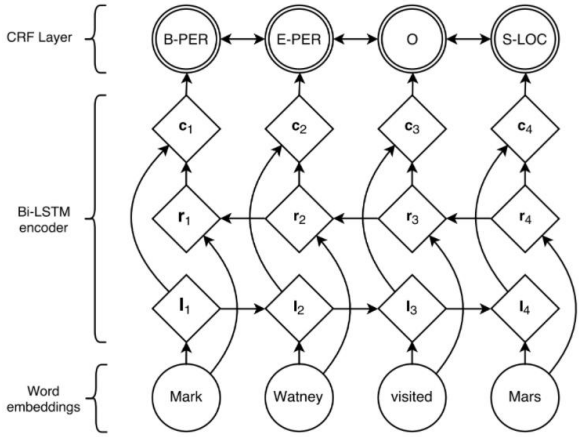

图 2　模型结构图

2.4.1 模型介绍

LSTM 的全称是 Long Short-Term Memory，它是 RNN（Recurrent Neural Network）的一种。LSTM 由于其设计的特点，非常适合用于对时序数据的建模，如文本数据。BiLSTM 是 Bi-directional Long Short-Term Memory 的缩写，是由前向 LSTM 与后向 LSTM 组合而成。两者在自然语言处理任务中都常被用来建模上下文信息。

使用 LSTM 模型可以更好的捕捉到较长距离的依赖关系。因为 LSTM 通过训练过程可以学到记忆哪些信息和遗忘哪些信息。

但是利用 LSTM 对句子进行建模还存在一个问题:无法编码从后到前的信息。在更细粒度的分类时，如对于强程度的褒义、弱程度的褒义、中性、弱程度的贬义、强程度的贬义的五分类任务需要注意情感词、程度词、否定词之间的交互。此时通过 BiLSTM 可以更好的捕捉双向的语义依赖。

加入 CRF 层的原因是，CRF 层可以加入一些约束来保证最终预测结果是有效的。这些约束可以在训练数据时被 CRF 层自动学习得到。

可能的约束条件有：句子的开头应该是“B-”或“O”，而不是“I-”。“B-label1 I-label2 I-label3…”，在该模式中，类别 1,2,3 应该是同一种实体类别。比如，“B-Person I-Person” 是正确的，而“B-Person I-Organization”则是错误的。“O I-label”是错误的，命名实体的开头应该是“B-”而不是“I-”等等。

2.4.2 细节介绍

需要讲的两点是:

（1）个人在 Lstm 与 CRF 层之间加了一层线性层，添加的经验来源于网上参考的博客。

（2）此处 CRF 使用的现成的库 pytorch-crf,具体使用可以参考其官方库，这里个人根据网上博客经验，将这个库给 download 下来放到工程里直接 import 进来使用。对于 CRF 层如果是训练，那么直接用发射矩阵和真实标签去计算 Loss，用于更新梯度。这需要用到 CRF 中的 forward 函数。

如果是预测，那么就用发射矩阵去进行维特比解码，得到最优路径（预测的标签）。这需要用到 CRF 中的 decode 函数。

这里个人通过是否传入标签来进行判断，如果传入就是要训练，没有标签就是要解码，以此来简化代码，避免代码重复书写。

2.5 Train 与 evaluate

2.5.1 Train

Train 部分代码逻辑个人将按照顺序解释：

（1） 构建 vocab 和 tag 词典，以及对应的 id 转换词典。

（2） 实例化 Dataset 以及 DataLoader。当然需要根据需要传入对应参数。

（3） 实例化 model。

（4） 构建优化器，这里个人选择的常用的 Adam 优化器。

（5） 按照 pytorch 训练流程，循环训练，计算 loss,反向传播更新参数。

（6） 当然这里个人添加了一个常见的梯度截断的 trick。防止梯度爆炸。

2.5.2 evaluate

这部分说明如下：

（1） 模型按照设置的参数，每 eval_step 在验证集上验证模型。

（2） 根据验证结果选择是否保存模型。

（3） 设置早停机制，防止模型无休止过拟合下去。

F1 指标：这里个人采取 sklearn 的 F1 计算函数，计算预测出的序列和标签序列之间的 F1 值。

由于是多分类,sklearn 函数需要选取对应参数,这里选取 micro 计算方式，通过计算总真阳性，假阴性和误报来全球计算指标。也就是把所有的类放在一起

算（具体到 precision），然后把所有类的 TP 加和，再除以所有类的 TP 和 FN 的加和。因此 micro 方法下的 precision 和 recall 都等于 accuracy。

三、 实 验

3.1 baseline 参数以及部分设置

（1）固定随机种子，降低由于随机导致实验不能复现的影响。

```
def set_manual_seed(seed):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
```

（2） 配置对应的 logger，主要方便模型日志的保存以及对于结果的查看。

（3）base-line 部分主要超参数

Num_epoch	20
Batch_size	32
dropout	0.4
Learning_rate	0.001
Lr_decay	0.9
Embedding_dim	128
lstm_hidden_size	256

3.2 实验

本章节主要是进行一些探索实验，由于代码运行时间不快，Lstm 本身的原因，所以并没有做多少，主要是进行一些简单的探索。

3.2.1 baseline 效果

这是本模型在一套 base 参数上的表现结果，其中 F1 计算可参考上文。

模型	Best_F1
baseline	0.8986

3.2.2 char embedding

该实验主要探索利用 pretrained 初始化 embedding 矩阵的影响，在本套实现代码下，是否初始化影响不大，效果并不显著。

Add_Char_Pretrained	Best_F1
True	0.8986
False	0.8951

3.2.3 LayerNorm 层

这里主要是验证，在线性层前后添加 layernrom 的效果。可以看到，效果有少许提升，说明这种方式是有效的。

Add_LayerNrom	Best_F1
False	0.8986
True	**0.9159**

3.2.4 Learning_rate

这里是对 learning rate 选取的实验，在 base 基础上，分别扩大一倍和缩小一倍，由于时间原因，这里只做一组对比试验。下述模型结果只有学习率不同，其余完全相同。

Learn_rate	Best_F1
0.001	0.8986
0.0005	0.8999
0.002	0.8989

上述表格可以看出，学习率对于现在 baseline 的影响很小，几乎没有。当然上述实验只是一次实验结果，可能会稍有偏差，但是总体结论不会有大的变化。

四、 总 结

该项目主要目的是搭建一个简单的 lstm-crf 网络，在人民日报新闻数据上自定义实体识别任务。

本项目主要提供了一套基于 pytorch 版本的，适用于该数据集上的完整代码，代码风格清新易懂。主要贡献在于区别于现有项目，本项目比较轻量化，主体训练代码并不冗余。当然，最终效果并不是很好，作者也致力于提升，无奈最后并未找到限制提升的关键因素。当然也和项目周期有关，该项目完成时间较短，可能有些地方并未考虑完整。