姓名：王想
学号：2100013146

Homework 3

课程名称：自然语言处理基础
2023 年 4 月 28 日

1 任务简介

本项目基于 BiLSTM 模型和条件随机场模型完成了中文命名实体识别任务。命名实体识别是自然语言处理中的一个基础切重要的任务，目的是从给定文本中识别出具有特定意义的实体，将它们分类为预定的类别，如人名、地名、时间、组织机构名等。这项技术可以应用于很多场景，比如信息抽取、文本分类、知识图谱构建等。项目中使用的数据集是公开的若干罪名起诉意见书，包括犯罪嫌疑人、受害人、时间、地点、组织机构等十种命名实体。

2 代码说明

代码运行说明 代码运行环境为 *torch*1.8.*txt*, 训练并测试可以直接运行 *python main.py*, 参数默认值已经调好, RTX 3050 Ti 上用默认参数训练大概需要 8 分钟。由于训练数据会随机打乱并划分成训练集与验证集，随机性会导致同样的参数训练出来的模型性能略有波动，按照提供的参考评估方式计算，多次实验中在验证集上的分数在 85 附近波动 (85 ± 2). *main.py* 时有以下几个参数, 这几个参数主要用于调整训练模型的训练器, 以及是否直接加载模型进行测试; 若要更改模型结构则需要在 model.py 中更改默认值。Note: 提交的文件中并未包含预训练模型, 如果要使用load True 则必须先运行程序训练一个模型并保存。

Perparameters	Type	Default	Help
batch_size	int	128	batch size
epochs	int	30	epoch
lr	float	0.01	learning rate
load	bool	False	True: load a model; False: train a model
model_path	str	./model.pt	Path of loaded model, used only if –load is true

Tab 1: **Parameters of main.py**

数据处理 数据的预处理 (load_and_porcess in DataProcess.py) 包括以下几点：

1. 以字符串的形式读取 json 文件，再利用 json.loads 转化成字典类型，此时的数据集保存于一个列表中，列表的每一个元素对应的一条数据 {'id':...,'context':...,'entities':[...]}

2. 对字典类型的数据集进行预处理，将数据列表中的每一个元素转化成列表格式，[id, context_str, label_sequence]
3. 对每一条数据，需要在文本最后加上一个 <stop> 标记便于后续训练与维特比解码
4. 如果是加载训练数据，还需要进行以下操作：
    - 对步骤 2 得出的数据列表进行随机打乱，并以 4:1 的比例分成训练集与验证集
    - 利用划分出来的训练集构建词表

词表与标签集中都要加入四个特殊标签 <start>,<stop>,<unknow>,<pad>，分别表示句子开始、句子结束、未知汉字、补位符 (便于把长短不一的句子进行统一处理)。这个函数的返回值是一个数据列表，每一个元素都是 [id, context_str, label_sequence] 的格式。主函数中加载数据 (Load_data in main.py) 调用了上述函数，把上述函数的返回值进行重构、打包。无论是训练集、验证集还是测试集，数据均重构成 [[ids], [context_strs], [label_sequences]] 的格式。

存取、读取模型 存取、读取模型的实现可以直接调用 $torch.save()$ 与 $torch.load()$ 函数实现，需要注意的是保存和读取的模型对应的超参数需要是一致的，比如都在 GPU 上或者都在 CPU 上、embedding 维度不变等。

模型实现: Class BiLSTM_CRF 模型基于 BiLSTM 和 CRF 实现，模型结构为 Embedding Layer + BiLSTM + MLP + CRF，forward 过程中，句子经过 embedding 层对每一个字提取 word vector，输入 BiLSTM 为句子添加注意力机制，之后经过一层全连接层，最后再输入 CRF 考虑全局得分。记 x 为输入的句子，y 为命名实体的 tag 序列，则得分定义如下：

$$Score(x, y) = \sum_{i=1}^{len(x)} \log emission(y_i, x_i) + \log transition(y_{i-1}, y_i)$$

其中 $emission(y_i, x_i)$ 由 BiLSTM 与 MLP 模块计算给出，$transition(y_{i-1}, y_i)$ 由 CRF 转移矩阵给出。利用上述定义的 Score, 定义模型的损失函数定义如下：

$$Loss = Score(x, y_{predict}) - Score(x, y_{truth})$$

这个 loss 函数能要求模型的预测结果尽可能接近真实的标注，从而帮助模型学到应该如何识别命名实体。模型进行预测时，利用上述定义的 forward 过程得到一个得分矩阵，再通过

维特比算法进行解码，找到最佳 Score(x,y) 对应的标注路径 y。记 $\pi[i,j]$ 表示第 i 个词被标记为 j 的时候取得最大得分，则维特比公式如下：

$$\pi[i,j] = \max_k\{\pi[i-1,k] + emission(j,x_i) + transition(k,j)\}$$

$$\pi[0,j] = emission(j,x_i) + transition(< start >, j)$$

模型训练: Class Trainer 训练设备默认情况下是 GPU，除非 GPU 不可用。训练的相关超参数可以参考 Tab 1，由调用 main.py 的指令输入，未输入则采用默认值。训练采用的优化算法为 Adam。模型进行训练时，每训练一个 epoch 都会将训练数据随机打乱一次，再以此采样成一个个 batch 喂给模型进行训练，等效于每个 batch 的训练数据都从 epoch 中随机不重复采样。在每个 epoch 训练结束后，在验证集上对模型进行测试评估 (评估函数将在下一段中介绍)，将取得最佳性能的模型进行深度拷贝暂时保存，训练结束后将最佳模型保存到文件中。模型进行测试时，直接调用模型部分的测试函数输出预测结果。Note: 为了利用高效的 tensor 计算与追求代码实现的简洁，模型部分的实现要求传入的句子按照长度进行排序、对齐 (补 <pad> 占位)，所以在训练过程中都会先对 batch 数据进行排序再传入模型进行计算，在测试过程中则是先排序后逆排序回原顺序。

预测和输出评价: Function get_score 评估函数几乎等同于提供的参考评估函数，唯一的改动是 ground_truth_num 不再是固定的值，而是类似 prediction_num 一样通过计算得出。在训练过程中，验证集上的预测结果和原本的数据标注会被分别保存成一个 json 文件，之后直接调用这个函数对模型的性能进行评分。

3 模型评价与错误分析

对模型与训练的超参数我进行了以下几组测试, 每次改变一个超参数，其他超参数为默认值。由于相同参数下模型性能会因训练集验证集的随机划分而波动，下面几组实验中的数据都是在相应下训练 7 个模型进行平均得到。Note: COM 表示 Cuda Out of Memory

根据上述实验可以发现，模型的 embedding_dim 和 hidden_dim 越大模型性能越好，这个很容易解释，模型参数量越大，模型的拟合数据能力越强，越容易学到更多的命名实体识别知识。但是理论上当参数量达到足够大时，如果不改变其他部分，模型性能应该无法再随参数量增大而提升。但是更大的模型在我本机 (4G 显存) 已经无法运行，尚且无法确定这两个参数达到多大才无法继续让模型性能随参数量增大而提升。

embedding_dim	16	64	256	512	1024
F1-Score	80.03	83.49	85.61	**85.82**	COM
hidden_dim	16	64	256	512	1024
F1-Score	80.67	83.26	85.74	**85.82**	COM
learning rate	0.0001	0.001	0.01	0.1	1
F1-Score	84.14	85.59	**85.82**	78.31	77.27
batch_size	16	32	64	128	256
F1-Score	83.43	84.67	85.76	**85.82**	COM

Tab 2: **model performance with different Hyperparameters**

learning rate 过大会导致模型“错过”最佳点，在最佳模型参数附近徘徊却总是无法到达最佳点。而过小则一次更新的“步幅”太小，导致模型学习较慢 (lr = 1e-5 这一组在训练时保持性能递增但 30epoch 后都没能收敛)，并且可能陷入局部最优。

训练时 batch_size 过小训练得到的模型性能比 batch_size 较大的性能更低，这是因为在小的 batch 上模型更容易过拟合，导致在验证集上的平均性能相对较低。但当 batch_size 足够大时，一个 batch 的数据已经足够多样，所以模型过拟合的可能性更小。

除此之外我还做了消融实验证明我采取的随机打乱数据的有效性, 并且对比了不同数据标注格式对模型的影响。表格中 Converge speed 指训练多少 epoch 之后接近收敛, 四舍五入估计到十位；对于命名实体类型 X，数据标注格式 XO 表示直接使用 X 和其他类型 O 作为标注，而不是 B-X,I-X 等。

Shuffle or not	F1-Score	Converge speed
True	85.82	10
False	84.73	20
Label format	F1-Score	Converge speed
XO	85.82	10
BIO	85.29	10
BIOES	84.62	10

Tab 3: **Ablation Study: random shuffle**

随机打乱训练数据的策略对模型训练的收敛速度和模型的性能提升比较明显，在上面的数据处理介绍中也已经分析过原因，实验结果与理论相符。而对于数据标注的几组实验

其实并无可参考意义，因为超参数并不相同，对于 NER 任务标准的 BIO 与 BIOES 标注，由于本机显存有限，采取这两种标注后无法使用 embedding_dim=512、hiddend_dim=512 的模型参数量，但是采用更小的参数量 (256、256) 时模型的性能略有下降，所以最终我选择放弃使用这些标准的标注格式，而是采用简单粗暴的 XO 标注。

本实验的不足之处在与两方面，一方面是硬件的限制，对超参数的探索不够充分；另一方面是时间的限制，大量时间花费在 coding 和 debug 上，最后的实验时间非常有限，所以我没能进行更加细致与深入的探究，也没能想到更好的 trick 增加模型的性能。