大作业：中文口语语义理解任务

小组成员

周易 520030910269：脚本编写+模型训练+报告
吴相佐 520030910270：脚本编写+模型训练+报告
工作贡献：周易（50%）+ 吴相佐（50%）

1.内容简介

口语语义理解是对话系统中重要的一环，其目的在于从用户的话语中提取出用户的意图，以为后续模块提供信息，最终顺利完成用户的任务。一般情况下，我们使用若干语义单元来描述用户的语义，每个语义单元由动作（act），语义槽（slot），槽值（value）三个部分构成。
任务模型：words（语句，含ASR错误）-> SLU -> semantic（语义）

2.模块介绍

data：

- lexicon：储存语义三元组的部分定义
- development.json：测试集
- ontology.json：语义三元组的定义，更详细的定义在lexicon中
- test_unlabelled.json：不含语义标记的数据
- test.json：`scripts/test.py`的输出结果
- train.json：训练集

utils：

- args.py：参数设定，包含`batch_size`，`learning_rate`，`max_epoch`等神经网络训练参数。
- batch.py：Batch类，训练批次
  Note：注意到Batch类会对传入的数据集进行sort，因而Batch中的数据与原始数据顺序不同。
- evaluator.py：Evaluator类，评估函数
- example.py：Example类，样本
- initialization.py：某些环境初始化
- vocab.py：Vocab类，词表
- word2vec.py：词和词向量间的转化

model：

- slu_baseline_tagging.py：基线模型

scripts：

- slu_baseline.py：基线模型的训练和测试脚本。默认为训练模式，传入参数`--testing`为测试模式。
- test.py：测试脚本，用本地模型对无标签语料`data/test_unlabelled.json`进行预测，结果输出到`data/test.json`中。

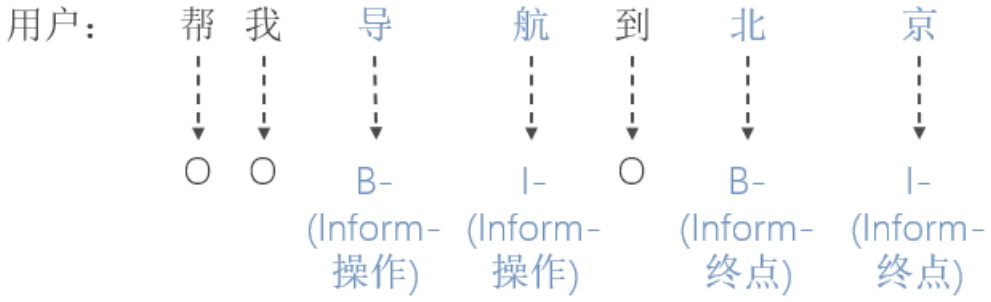
其他：

- model.bin：训练好的基线模型
- word2vec-768.txt：词向量表

3.基线模型

本次作业提供的基线模型中，我们使用序列标注BIO模型的方法，为当前对话中的每个符号（词或字）打上一个标签。标签可能为“B-X”、“I-X”或者“O”。其中，“B-X”表示该符号所在的片段属于X类型并且此符号在此片段的开头，“I-X”表示该符号所在的片段属于X类型并且此符号在此片段的中间位置，“O”表示不属于任何类型。

该方法通过预测对话中每个字的标签确定act-slot在文本中的value。对整个序列进行标注后，我们可以从中提取出最终的结果。



在上面这个例子中，该算法为对话“帮我导航到北京”里每个字打上标签“O”、“O”、“B-Infom-操作”、“I-Inform-操作”、“O”、“B-Inform-终点”、“I-Inform-终点”后，可以找出该段对话中有两个语义，并且其在语料中对应的文本为其相应的语义value。

最后的结果为：inform(操作-导航)，inform(终点-北京)

Note：序列标注方法在进行标注时只标注act-slot信息，而value值是从文本中获取，可以解决遇见未预先定义的槽值问题。

4.测试脚本

```
model = SLUTagging(args).to(device)
Example.word2vec.load_embeddings(model.word_embed, Example.word_vocab, device=device)

def predict():
    model.eval()
    test_path = os.path.join(args.dataroot, 'test.json')
    test_dataset = Example.load_dataset(test_path)
    predictions = []
    did = []
    with torch.no_grad():
        for i in range(0, len(test_dataset), args.batch_size):
            cur_dataset = test_dataset[i: i + args.batch_size]
            current_batch = from_example_list(args, cur_dataset, device, train=False)
            pred = model.decode(Example.label_vocab, current_batch)
            predictions.extend(pred)
            did.extend(current_batch.did)
    test_json = json.load(open(test_path, 'r', encoding="utf8"))
    ptr = 0
    for example in test_json[:10]:
        for utt in example:
            utt['pred'] = [pred.split('-') for pred in predictions[did.index(ptr)]]
            ptr += 1
    json.dump(test_json, open(os.path.join(args.dataroot, 'test.json'), 'w', encoding='utf8'),

check_point = torch.load(open('model.bin', 'rb'), map_location=device)
model.load_state_dict(check_point['model'])
print("Load saved model from root path")
predict()
print("Prediction accomplished. Output saved in file 'test.json'")
```

代码详见`scripts/test.py`

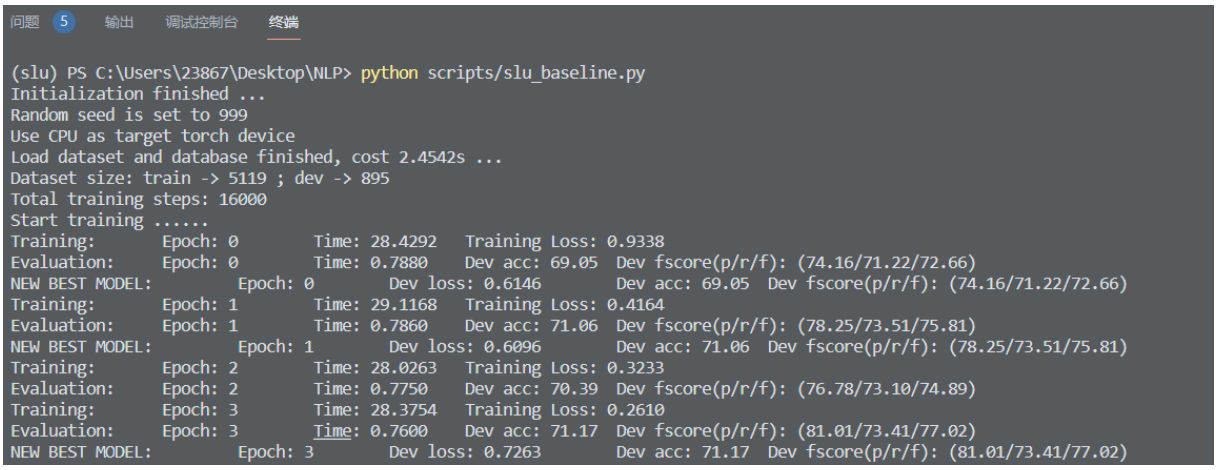
- 脚本完成对无标记语料的预测工作，从`data/test_unlabelled.json`中读取语料，预测后将结果存储到`data/test.json`目录
- `predict()`函数完成该脚本的主要功能
    - 调用`from_example_list()`生成`current_batch`时，参数`train=False`，这会导致Batch类中部分成员的值为None
    - 相应地，`model.decode()`也不需要计算loss值，只需返回pred值，需要对相关接口进行修改
    - 由于Batch类在初始化时会对传入的样本列表进行排序，最后得到的pred值顺序会与原数据集不一致
    - 为Example类和Batch类增加数据成员did用于标识最初的样本顺序， 借助did最后可将预测结果按原始顺序输出
- 脚本运行结果见后文

5.实验设定及结果

根据实验结果，RNN神经网络训练速度较快但效果较差，LSTM神经网络训练速度较慢但效果较好，GRU网络训练速度介于两者中间，训练效果基本介于RNN和LSTM网络之间。为得到更好的模型，最终选用LSTM神经网络进行训练。
本次作业最终提交以下述环境训练出来的模型，存储在根目录下`model.bin`中。

实验设定

- script：`slu_baseline.py`
- model：`slu_baseline_tagging.py`（基线方法）
- utterance：ASR值
- seed：999
- batch_size：32
- learning_rate：1.4e-3
- epoch：17
- encoder_cell：LSTM
- dropout：0.2
- embed_size：768
- hidden_size：512
- num_layer：3



上述环境下，模型最终在Epoch17（MAXEpoch：100）取得最大准确值。

实验结果

对测试集（`development.json`）进行测试的效果：

- Dev acc：72.07
- Dev fscore(p/r/f)：(81.77/ 74.35/ 77.88)

```
问题 5  输出  调试控制台  终端

(slu) PS C:\Users\23867\Desktop\NLP> python scripts/slu_baseline.py --testing
Initialization finished ...
Random seed is set to 999
Use CPU as target torch device
Load dataset and database finished, cost 2.5134s ...
Dataset size: train -> 5119 ; dev -> 895
Load saved model from root path
Evaluation costs 0.79s ; Dev loss: 1.0983       Dev acc: 72.07  Dev fscore(p/r/f): (81.77/74.35/77.88)
(slu) PS C:\Users\23867\Desktop\NLP>
```

对不含语义标记文本（`unlabelled.json`）的预测：

- `取消导航`：inform（操作=取消），inform（对象=导航）
- `导航`：inform（操作=导航）
- `朝阳县农机销售有限公司导航`：inform（终点名称=朝阳县农机销售有限公司），inform（操作=导航）

详见data/test.json文件

6.分析与讨论

6.1 模型训练

实验过程中发现，在许多情况下，model总是在前几轮epoch就得到最大acc值；而后续epoch中model不再能得到更大的acc值甚至可能剧烈下滑。且经过多种参数测试后得到的最终model性能提升也极其有限。

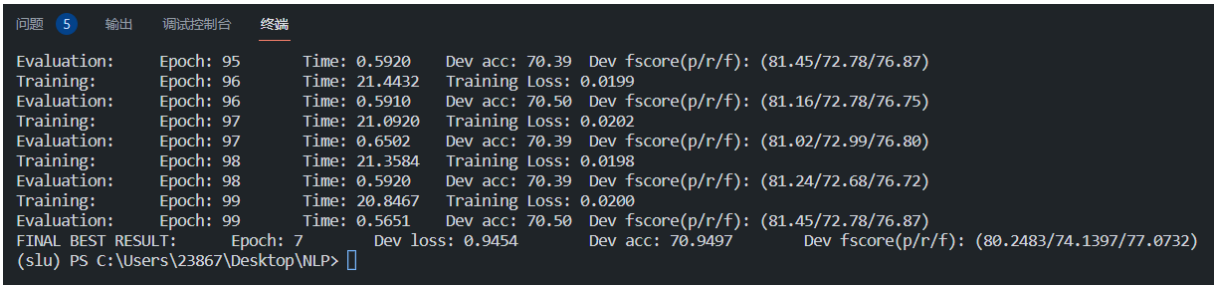
例：learning_rate=1.5e-3，num_layer=2的训练情况（其余设定同上）：

训练开始：

```
问题 5  输出  调试控制台  终端

(slu) PS C:\Users\23867\Desktop\NLP> python scripts/slu_baseline.py
Initialization finished ...
Random seed is set to 999
Use CPU as target torch device
Load dataset and database finished, cost 2.4771s ...
Dataset size: train -> 5119 ; dev -> 895
Total training steps: 16000
Start training ......
Training:       Epoch: 0        Time: 19.9477   Training Loss: 0.8319
Evaluation:     Epoch: 0        Time: 0.5640    Dev acc: 69.94  Dev fscore(p/r/f): (75.08/72.89/73.97)
NEW BEST MODEL:         Epoch: 0        Dev loss: 0.5964        Dev acc: 69.94  Dev fscore(p/r/f): (75.08/72.89/73.97)
Training:       Epoch: 1        Time: 21.4650   Training Loss: 0.3941
Evaluation:     Epoch: 1        Time: 0.5870    Dev acc: 70.84  Dev fscore(p/r/f): (78.75/73.41/75.98)
NEW BEST MODEL:         Epoch: 1        Dev loss: 0.6139        Dev acc: 70.84  Dev fscore(p/r/f): (78.75/73.41/75.98)
Training:       Epoch: 2        Time: 19.3740   Training Loss: 0.3043
Evaluation:     Epoch: 2        Time: 0.5370    Dev acc: 69.50  Dev fscore(p/r/f): (77.99/72.78/75.30)
Training:       Epoch: 3        Time: 20.3160   Training Loss: 0.2381
Evaluation:     Epoch: 3        Time: 0.5950    Dev acc: 69.39  Dev fscore(p/r/f): (80.40/71.43/75.65)
Training:       Epoch: 4        Time: 20.9309   Training Loss: 0.1759
Evaluation:     Epoch: 4        Time: 0.6150    Dev acc: 69.83  Dev fscore(p/r/f): (80.30/72.68/76.30)
```

训练结束：



该轮训练仅在Epoch0、Epoch1、Epoch7中获取了更好的模型，且自Epoch7后模型再也没有取得过更好的acc值。

经讨论我们认为与以下原因相关：

- 训练集和测试集样本差距的问题（本质上讲，我们认为训练集和测试集的相似度决定了模型理论上所能取得的最好效果）
- 评估函数和选取最优模型标准的问题

其中我们认为改进模型评估标准更能优化实际训练中遇到的问题，理由如下：

- 在代码中，根据acc值选取最优模型，而acc值的评判标准相当严苛，仅模型预测的语义与预先标记的label完全相同时，acc值才会增加。
- 对于某些较难的对话，模型在训练过程中对其的预测结果可能不断逼近label值，但由于无法完全预测正确，acc值还是不会改变，模型因此也得不到有效的反馈。可能一定程度上导致了上述提到的训练性能瓶颈问题。
- 这种评估方法粒度较粗，不能很好地反映模型间的细微差异。
    - 假设一段对话中其label中包含三个语义单元，预测正确零个语义单元的模型和预测正确两个语义单元的模型在该种评估体系下不能体现出差异。
    - 可以采用更细粒度的方式，比如用单个句子作为基本单元代替整段对话计算acc值。
    - 可以将语义单元转化为词向量，最后根据词向量距离来评价该语义单元预测好坏。比如，如果label中的语义单元为 `inform(终点名称=上海市闵行区交通大学)` ，那么 `inform(终点名称=交通大学)` 应该比 `inform(终点名称=复旦大学)` 获得更高的评分。

对评估算法进行改进，能帮助模型获取更好的反馈，提升模型性能。

6.2 未见过的槽值

作业中语义三元组（act-slot-value）预先设定在文件 `data/ontology.json` 和目录 `data/lexicon` 中。其中，act和slot数量相对固定，可以通过人工的方式设定；但value值数量庞大，难以统计，且会不断发生变化（如人名、地名等专有名词），实际应用中难免会遇见未见过的槽值（没有在预定义的表中出现过的value值）。

序列标注方法能较好地解决上述问题。实验中的序列标注方法含三种标注O、B-act-slot和I-act-slot，最后的value值则是在标注完成后从语料中选取对应文本计算value值。

如若截取出的文本与已知的任何一个槽值都无法进行匹配，说明我们遇到了一个新的槽值，可以选择直接将被B、I标记的文本作为槽值，并将槽值添加到已知槽值中。