上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

CS3602 自然语言处理课程报告

口语语义理解模型设计与调优

王梓萌 521030910015

刘洺皓 521030910014

2024 年 1 月 14 日

目录

1 任务说明

口语语义理解（Spoken Language Understanding, SLU）是指对人类口语进行解析和理解的任务，旨在从语音中提取有意义的信息，并将其转换成计算机能够处理的形式。

口语理解任务通常包含多个阶段。首先语音识别模型被用于将输入的用户语音信号转换成文本，这个过程通常由自动语音识别系统完成，它使用声学模型和语言模型来预测说话者说的词语。识别后的文本将被用于训练语义理解模型，用来从用户的话语中提取用户的意图，为后续的对话管理模块提供基础的支持。最终的对话管理系统根据具体的下游任务不同，可以被训练成许多不同类型，比如智能助手或交互式系统等等，最终，所有任务形成了一个完整的端到端学习的过程。具体的任务图示如 图 1 所示

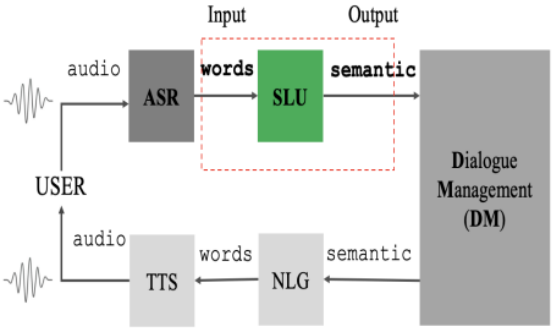


图 1: 口语语义理解任务简要图示

在本次任务中，我们主要关注如何使用 SLU 模型来提取对话中的语义。我们的任务中语义被建模成了”action-slot-value“的三元组的表达形式。其中，语义槽（slot）侧重于表现一个语义单元语义类型，比如 slot 可以是”操作“,”目的地“，它同时限定了某个 slot 下能够填写的 value 的类型。比如 slot= 操作时，value 一般则是“导航”,”打开“,”取消“等动作，而不会是”凯里大十字“,”渭南市人民政府“之类的地名。而 action 则侧重于表现语义单元的抽象动作特征，比如”confirm”，“request”，“deny”等抽象的动作。一个 action 下面可以包括多个符合的 slot。action 和 slot 由粗粒度到细粒度地为一个语句建模语义标签。因此，使用 action-slot 的对，就可以以一种序列标注的方法为语义完成建模。

在任务中，我们额外使用了 BIO 的序列标注方法为语义槽标注的语义添加分词层面的信息。具体地，B 代表一个 action-slot 语义槽中开头的单词，I 代表一个语义槽中非开始阶段的单词。O，代表不在语义槽中的单词。

因此，一个具体的语句的标注效果如图 2所示

我们具体任务中所给的数据集来自一个导航语音识别得到的数据集，包括使用语音模型识别的语音转文字文本 asr1_best，使用人工听写得到的文本 manual_transcript，以及根据 manual_transcript 标注的 semantics 标签，需要完成的任务是设计和调优模型，在导航数据集中得到更高的 accuracy 和 f1 值。

在本任务中，我们的贡献主要集中在三个方面

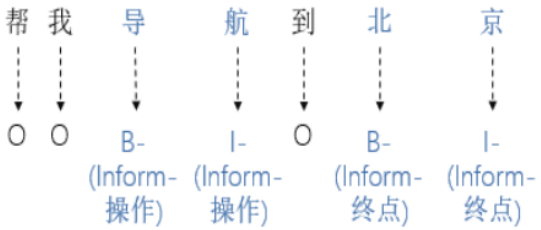


图 2: 序列标注建模语义的图示

1. 数据预处理

   - 我们观察了模型在训练集中的一些数据不均衡现象，基于提供的标签词典进行**数据增强**，比较不同增强后的结果。
   - 我们尝试使用错别字纠正的 python 模块进行**数据清洗**,使用 manual_transcript 数据进行训练。
   - 我们尝试引进带有 bigram 和更多英文词汇 embedding 的**词典增强**，复现了相关工作，使模型对词级别向量有理解能力。

2. 模型改进

   - 我们尝试使用 **Bert** 替换 LSTM encoder，收获了不错的效果。
   - 我们尝试使用 **bert + encoder + bert-decoder** 的架构进行训练，希望提高模型的大小以提升精度。
   - 我们尝试**多头训练**，针对 BIO 标签和语义槽标签设计不同输出头以及损失函数联合训练。
   - 我们**堆叠 Bert+LSTM**，该模型获得最高的准确率

3. 不同建模方式 & 相关工作的探索

   - 我们探索和实现了 **Lattice Transformer**，并比较了他和我们模型在数据集上的表现。
   - 我们探索和实现了 **DCANet**，并比较了其与我们的模型在数据集上的表现。
   - 我们讨论了**指针网络**和**阅读理解**的建模方式的实现原理，作为探索不同建模方式的一部分。

我们将所有代码上传到了github 仓库 供助教参考。

2 数据预处理

2.1 数据增强

通过分析，我们发现所给的导航数据训练集中的标签具有分布不均匀，模糊等特点，比如：

- “页码”slot 类型显著地小于“序列号”类型，因此容易对这个 slot 进行误判。
- 默写标签下的 value 出现在 lexicon 中，但训练集总几乎没有怎么出现。
- action 为 deny 的标签在数据集中出现过少，导致无法充分学习到 deny action 的情况

标签模糊的问题具体体现在，

- 比如”最近“这个 value，既可以属于路线偏好，也可以属于请求类型这两个 slot。分析训练集中出现这个东西的具体情况可以发现，二者只有细微的差别（如最近一次被标注为路线偏好时，语句中需要出现具体的路名）
- 再比如，”第一“，”第一个“，“第一家”等词语均可以被划分为序列号，且“第一个”出现的概率最高。这样可能对模型的学习造成困难

针对这些问题，我们采取数据增强的方式，缓解数据集分布不均匀和模糊的问题。**具体地，数据增广的步骤是：**

1. 首先增广数据量较少的 value 和 slot：

   我们首先在训练集中，寻找“请求类型”，“对象”，“页码”等标签中出现较少的 value，然后找到一个有该相同标签的数据中，替换原来语句中的 value，覆盖原来的 example
2. 增广数据量较少的标签

   我们对于一些出现较少的标签，比如“页码”，“路线偏好”等标签进行增广。增广的方式和和第一种大致相同，但是是复制并替换原来语句中的 value，得到一个新语句
3. 增广 deny 标签，我们对于 deny action 的增广采取的方式是，随机选取 20% 的包含“导航到 xxx”词语的语句，将其变更为“导航到不是 xxx“，同时修改 action 从 inform 变为 deny。

我们选取的 value 是从 ontology.json 文件中获得的，我们确实发现 ontology 中的一些 value 存在于 development 测试集中，所以使用 ontology 中的词汇存在泄露测试集的风险。但是考虑到既然 ontology 作为文件的一部分提供给我们，应当也可以进行利用，因此我们这里没有考虑由此带来的导致数据集作弊的问题。

相关实验对比验证了我们的方法。表 6.1是使用 baseline 模型下不增强，错误增强和目前使用的增强方法的效果对比，训练的流程见 8a。

2.2 数据清洗

我们注意到 asr 文本和 manual_transcript 文本的不匹配显著影响了模型的训练。

我们将 **asr 和 manual_transcript 不同的测试案例**认为是噪声之一，具体原因为：

首先，baseline 方法中，输入预测模型的 input_id，是由 asr 文本产生的。而最终用于计算准确率的 slot-value 中的 value 是 manual_transcript 文本中的。这说明如果 asr 和 manual_transcript 存在不同，那么即使模型再 asr 输入上预测再强大，也必然在带噪声的测试案例中，无法与 manual_transcript 生成的三元组在 value 上相同，影响模型的评判

其次，我们发现在标签标注的时候，baseline 是比较 asr 中是否出现 semantics 中标注的 value，然后对出现过的 value 打上 semantics 中对应的 slot。然而，我们发现 semantics 中的 vaue 是 manual_transcript 对应的。这说明，即使一个 asr 样本的句子基本正确，只在 value 中有一个错别字，那么因为 semantics 中找不到含有错别字的 value，所以这个 asr 的对应位置的标签不会被标注上 slot。这个问题的后果较为严重，因为它可能导致很多基本正确的词语（尤其是地名），没有被打上标签就送入训练，即所以很多句子的标签最终是空的列表作为 ground label，对模型来说是一种干扰。

基于这个情况，我们提出了两种方法缓解这个问题。

首先，我们尝试使用 manuscript 文本代替 asr 文本出现在训练集的 utt 中。这基本能够让 acc 正确反映一个模型过的真实训练实力，也能让 semantic 中的标签正确地标注在句子上，使得训练集的质量更高，同时不对测试集做修改，保证算法评价的公平性。

其次，我们通过参考了开源的错别字修正模块 'pycorrector' 尝试对测试集数据进行清洗。

pycorrector [1] 的调包使用范例如：

> asr = ”到郑州日七区侯寨乡肖如何渔湾生态园 “
> manual_script = ” 到郑州二七区侯寨乡肖潞河御苑生态园”

存在噪音，我们尝试使用 'pycorrector'上表现最好的预训练模型 'macbert4csc-base-chinese' 进行处理，模型输出为”到郑州日七区侯寨乡看赏何渔湾生态园 “，并没有达到数据清洗的作用。

通过多次尝试，发现大部分的错误，pycorrector 模型都**无法将 asr 修复到 manual_transcript 的值**。我们认为这种问题来源于测试数据集中的噪声并不是常见字词之间的错误。正确和错误的词语之间，虽然属于声音近似词，但是部分专有名词很难还原，因此难以起到数据清洗的作用，因此我们没有采用 pycorrector 的方法

2.3 引入额外的词典

我们的任务中，给出的默认词典是 768 维的字符级别的词典。带有少量的英文单词和特殊字符。

我们认为使用相关工作中使用的 bigram 和 unigram 的混合词典，以及中英文混合词典，能够增加 baseline 模型对字符，词的理解，提高准确性。

具体地,我们参考了 Lattice LSTM 工作中所使用的词典生成方法,将开源的 unigram 和 bigram 词典进行精选和混合后，得到了一个处理后的 bigram unigram 的中问字词和英文字符混合词典，嵌入维度为 50.

我们尝试使用如下的方法利用引入的词典:

我们首先尝试不使用词典中的 bigram，仍然使用 unigram 的方法进行测试，可以发现使用引入词典的准确率相比 baseline 也有了一定的提升，具体的细节见实验部分。

我们接着尝试使用词典中的 bigram 进行辅助，但是经过实验发现，简单地将出现 bigram 的词语的 unigram 替换为 bigram 的嵌入（bigram 每个字符的嵌入向量相同），并且仍然按照字符的粒度进行模型的训练，效果相比 baseline 有所下降。由于时间原因，没有仔细考虑是否是实现错误，因此在报告中不作展示。但是我们成功地复现了原相关工作中的 Lattice 结构进行字符，词级别的词典融合，该工作将在 section 4.1 细介绍。

具体的实验结果见 表 5

2.4 使用 Bert tokenizer

我们也尝试使用预训练大模型 Bert 作为词嵌入的模型。但是，由于 bert 的词嵌入是和 bert 模型息息相关的，比如 bert 词嵌入中包含了开头和结尾的 ‘<cls>‘ ‘<sep>‘ 等字符。我们认为认为将 Bert 的嵌入掐头去尾作为 LSTM 的输入并不是非常合适，因此使用 Bert tokenizer 的方法和使用 Bert encoder 的方法将一起介绍，尤其是 Bert 分词中处理英文的情况，由于分词的粒度不是字符级别，导致输入时候的维度对齐出现问题，代码的实现在github 仓库源代码中

3 模型改进和调优

我们主要设计了基于 Bert 的架构尝试提高模型预测的准确率

1. BertOnly
2. Bert Encoder + Bert Decoder
3. Bert + Multihead
4. Bert + LSTM

在使用 Bert 之前，首先需要对维度进行处理，使用 Bert tokenzier 替代词典 embedding。在我们的实现中，为了实现输入 BERT 维度的对齐，我们对英文分词采用了特殊的处理手段，包括特殊字符的处理，英文单词删除首尾的 Bert 分隔符，补齐至等长等操作。

接着，我们具体介绍每个模型设计的目的，尝试解决的问题，以及最后的效果。

具体的的代码实现在源代码中以类的形式分别实现。

3.1 BertOnly

BertOnly 模型是我们设计 Bert 模型的 baseline，model 其只是在 baseline 的基础上，替换了 LSTM 为 Bert，替换了 embedding 的方法，其余的地方均未做更改。

我们在 Bert Only 模型中使用不同的预训练 Bert 模型进行测试，包括”bert-base-chinese“,” chinese-macbert-base“,” chinese-roberta-wwm-ext“和”hfl/chinese-bert-wwm” 最终我们根据性能选择了 hfl/chinese-bert-wwm 作为预训练的 bert 模型。

3.2 BertEncoder + BertDecoder

在 BertOnly 模型中，通过打印出错的一些数据样例，我们发现 BertOnly 模型在一些词语的边界识别上做的比 LSTM 更加出色，但是在一些较难的标签的选择上，仍然选择错误了标签。因此，我们希望通过增加模型容量的方式，设计新的更大的模型用于训练。

因此我们设计了 BertEncoder + BertDecoder 的模型，使用两个 Bert 模型，一个的 is_decoder 参数设置为 True，add_cross_attention 设置为 True，作为 decoder，一个的作为 encoder，将模型的最后一层输出的 hidden state 作为输入，和 input_id 一起输入 decoder 模型之中，构成 encoder + decoder 架构的模型。

3.3 Bert+Multi-head

在分析 BertOnly 出错的样例中，我们发现样例在一些名词上的切分并不准确，比如“卫生局”这个名词，可能会加入它前面的“那”这个字，变成“那卫生局”，再比如地名”上上足“也可能会被切分成”上“和”上足“，然后分配错误的标签，导致准确率降低。因此，我们希望新建一个子训练任务，单独预测一个词语中 B，I 和 O 的分布，希望由此模型能够通过训练，对 BIO 的分布这个比 tag_id 所需预测数量更小的任务上有一个更好的表现，由此提升模型在切分专有名词任务上的表现。

具体地，我们在 example 类构建的时候，加入了新的标签 ‘BIO_id‘，对于一个标注完 action-slot-value 的语句，我们将 B 标签的 id 记为 3，将 I 标签的 id 记为 2，将 O 标签的 id 记为 1，padding_idx 依然是 0 ，得到的序列作为新任务的 ground_label。

我们在 BertOnly 模型的基础上加以改进，将最后一层的 hidden_state 提取出来，作为分类头的输入。我们除了使用 baseline 中原有的 TaggingFNN 全连接层完成 tag_id 的预测任务外，还额外使用了一个 CRF 层作为输出层，完成 BIO_id 的预测任务。在课内介绍过 CRF 可以将标签的分布，结合手动设计的转移规则进行综合输出一个最终最好序列的评分。比如，我们不允许 O 直接转移到 I 标签，所以在 CRF 层初始的 4x4 转移概率矩阵中，将 ‘trans_matrix[1][2]‘ 的值设为负无穷。

同时存在一些更复杂的规则是，有些标签的转移需要首先判断两者是否又属于一个 action-slot 的标签，比如只有不同标签之间，才能通过 ‘I‘转移到 ‘B‘等。但是由于我们

所选择的 CRF 层的模组并不支持这种粒度的修改，因此我们通过讨论说明考虑到这种情况，并期待在未来的工作中进行补充

综上，Bert+Multi-head 模型的架构如图 6所示。我们希望通过这种多任务学习的方式，降低模型学习部分任务的难度，由此提高最终的学习成果。

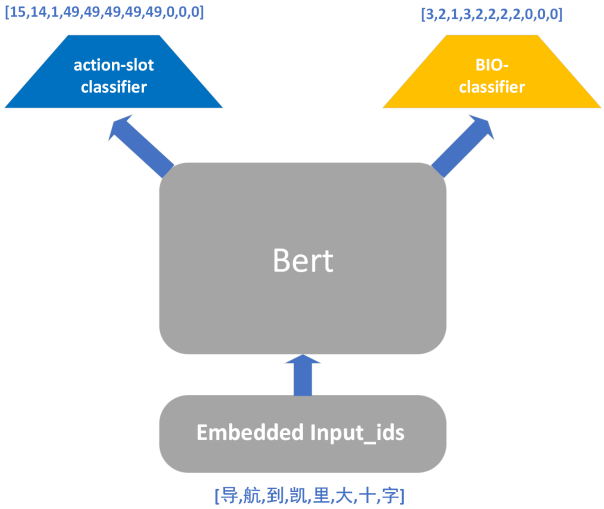


图 3: 口语语义理解任务简要图示

3.4 Bert+LSTM

经过思考，我们认为 Bert 可能并不能完全代替 LSTM 在本人物种的作用，因为 Bert 作为预训练模型，其模型参数并不会用于训练，因此虽然在训练初始的效果显著高于 LSTM，但是训练参数量并不比 LSTM 多一些，所以最终充分训练下能够拟合我们数据集的能力可能并不会很好。LSTM 完全是从头开始训练，因此能够调整的幅度也更加多，所以可能更加适合于在一个数据集上增加准确性。

基于这种考虑，我们设计了 BERT + LSTM 串联的模型。模型的设计朴实简单，即使用 Bert 首先对输入的 input token 进行 embedding 和综合，然后取最后一层的 hidden_state 作为 BiLSTM 的输入，继续训练得到的输出再进入输出头预测 tag_id

然而值得一提的是，这种改进最终使得模型的结果达到最好，具体的实验结果在第三部分进行展示

4 不同相关工作的探索

通过查阅文献和相关工作，我们实现了以下两个不同方法从不同角度去解决 SLU（NER）的问题。

4.1 Lattice

4.1.1 模型结构介绍

Lattice 是一种特殊的神经网络结构，专门针对中文命名实体识别（NER）任务而设计。这种模型结构能够有效地处理中文这种没有显而易见空格分隔的语言的特点，使得模型能够同时考虑字符级别（character-level）和词汇级别（word-level）的信息。

[2] 这篇工作首先提出了 Lattice 结构和 Lattice LSTM 模型。Lattice LSTM 的关键创新点在于其能够将预先分词的词汇信息以格子（lattice）的形式整合到序列模型中。这样可以让模型在学习序列的时候，不仅关注到单个字符，同时也关注到这些字符所组成的更长的词汇。这对于中文来说尤其重要，因为在中文里，一个词往往由两个或两个以上的字符组成，并且词的界定对理解文本的含义至关重要。相关图示见图 4

在标准的 LSTM 中，每个时间步只处理一个输入单元（在中文中即为一个字符）。然而，在 Lattice LSTM 中，每个时间步可以同时处理多个输入单元。这样的设计允许模型在每个时间步考虑多个可能的词边界，这是通过将词汇信息以格子的形式嵌入到模型中实现的。

具体来说，Lattice LSTM 在构建输入序列时，除了字符本身，还会查找一个预先训练好的词典，以确定哪些字符序列可以形成一个有效的词（bigram）。然后这些词作为额外的信息与字符一同输入到 Lattice LSTM 中。例如，对于字符序列” 北京大学”，除了每一个单独的字符” 北”、” 京”、” 大”、” 学”，词” 北京” 和” 大学” 也会被识别并加入到序列中。

Lattice LSTM 在其单元内部增加了额外的门控机制来处理这些词汇信息。它引入了一个额外的细胞状态来存储与词汇相关的信息，并使用专门设计的门控机制来控制信息的流动。这意味着在每个时间步，Lattice LSTM 的状态更新不仅依赖于前一个时间步的状态和当前的字符输入，还依赖于与当前字符匹配的词汇信息。

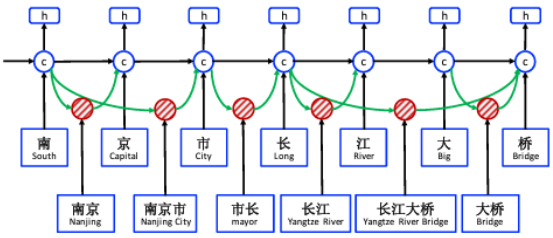


图 4: Lattice LSTM 模型架构

而最近的工作 [3] 中提出了一种全新的 Flat Lattice 结构。Flat Lattice 结构的核心思想是将词汇信息以一种扁平化（flat）的方式引入到 Transformer 模型中。这种方法不会改变原本 Transformer 输入序列的长度，也不会引入额外的复杂度。Flat Lattice 结构是通过将字符和相应的词汇信息合并在同一个序列上，而不是创建额外的层级或维度，简化了原始 Lattice 结构的复杂性。模型的示意图见

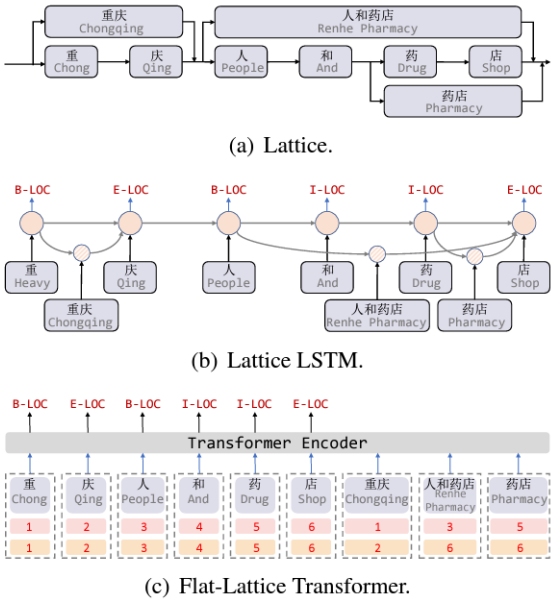


图 5: Flat Lattice Transformer 模型架构

为了适配 Flat Lattice 结构，Transformer 模型中的自注意力（self-attention）机制需要做出适当调整。在标准的 Transformer 模型中，自注意力机制将序列中任意两个位置的元素直接进行关联。然而，在 Flat Lattice Transformer 中，自注意力机制需要能够同时处理字符级别和词汇级别的信息，并确保在计算注意力时能够区分两者。

使用 Transformer 代替 LSTM 之后，同时解决的还有模型并行化输入的能力。Lattice LSTM 中由于 Lattice 是为每一个句子设计的，所以 batch_size 只能为 1，现在的 Flat Lattice 架构，成功支持并行化输入，加大了训练的速度。

4.1.2 实验结果

我们分别实验了 Lattice LSTM 和 Flat-Lattice-Transformer 两种模型。我们参考了原论文开源的模型实现，同时将输入格式，评价指标和输出形式和我们项目对齐。我们的实现发布在了项目 GitHub 仓库的 Lattice 目录下，代码可供复现。

我们在相同的实验条件下探究了两个模型的能力，它们的具体指标见表 1

模型种类	f1	Precision	Recall
`baseline`	77.62	81.44	74.13
`Lattice LSTM`	54.70	49.80	60.67
`Flat-Lattice-Transformer`	76.45	73.14	80.08

表 1: Lattice 模型的评价指标

从模型的得分和 baseline 的比较可以看出，我们复现的 lattice 方法在没有调优或进一步探究的情况下，并没有击败使用普通 LSTM 的模型。其中，Lattice LSTM 模型

相比 baseline 来说，性能相差甚远，不排除是复现过程中的一些问题导致的实现有误。Flat-Lattice-Transformer 的性能基本和 baseline 持平。如果然而可以看到 Flat-Lattice-Transformer 虽然 F1-score 值和 baseline 相差不多，但是 baseline 中 precision 表现优于 recall，而 Flat-Lattice-Transformer 中 recall 表现优于 precision。这说明 Lattice 模型确实更倾向于预测更长的更多的词语，而相对来说 baseline 对于预测更为精准，但是会因为组成的词语较短而遗漏一些专有名词。这和 Lattice 结构的特点是吻合的。

4.2 DCANet

我们选择使用 DCA-net 的主要原因是他引入了双向的 co-interactive cross-attention 机制, 可以更好的捕捉到这句句子之间的单词的长依赖关系, 下面将简要描述下 DCA-net 的架构.

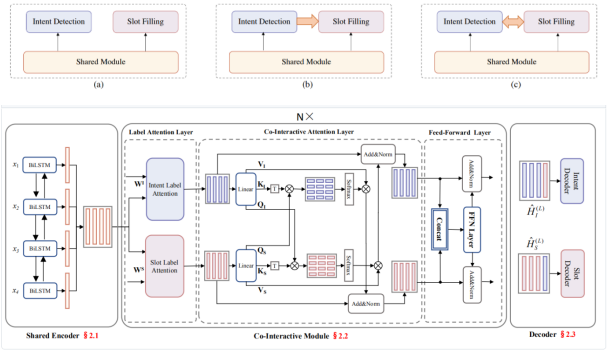


图 6: DCA-net 的模型架构总览, 和他所采用的 slot-filling 和 intent-detection 联合预测的方案. 左图表示相比传统的两个人物分开训练, 或者使用 intent detection 的结果来引导完成 slot-filling 任务, DCA-net 使用 joint-optimization scheme 和 cross-attention 方式, 意图在两个任务上都得到更佳效果, 右图展示 DCA-net 的模型架构, 左侧是共享编码器, 中间是 co-interactive cross-attention layer, 右侧的是 CRF 解码器.

4.2.1 模型结构介绍

DCA-net 使用 BiLSTM 作为共享编码器，该编码器旨在充分利用词或编码器的优势。BiLSTM 由两层 LSTM 组成。对于输入序列 $\{x_1, x_2, \cdots, x_n\}$ (n 是标记数)，BiLSTM 对输入序列 $\{x_1, x_2, \cdots, x_n\}$ 抽取出对上下文敏感的隐藏状态 $H = \{h_1, h_2, \cdots, h_n\}$。反复应用递归 $h_i = \texttt{BiLSTM}(\phi_{emb}(x_i, h_{i-1}))$，其中 $\phi_{emb}(*)$ 表示嵌入函数。DCA-net 的重点是中间的 co-interactive cross attention 模块，该模块旨在构建意图检测 (intent-detection) 与插槽填充 (slot-filling) 之间的双向联系. 在 Vanilla Transformer 中，每个子层都由一个 attention layer 和前馈网络（FFN）层组成。相比之下,DCA-net 的 co-interactive cross attention 模块中，首先应用意图和插槽标签关注层，以获得明确的意图和槽表示。然后，他们采用 co-interactive cross attention 层, 而非 self-attention 层来模拟外部的相互

影响。最后，DCA-net 扩展了基本的 FFN，以进一步融合意图和 slot 信息。在刚刚提到的意图和 slot 标签注意层中, 我们对 intent 和 slot label 进行标签关注，从而得到显式的意图表示和插槽标签, 随后, slot-label 和 intent 的隐空间表示被送入 co-interactive cross attention layer，特别来说, DCA-net 使用全连接的 slot filling 解码器层和 intent detection 解码器层的参数作为 slot embedding $W^S \in \mathbb{R}^{d\times|S^{label}|}$ | 和 intent embedding $W^I \in \mathbb{R}^{d\times|I^{label}|}$ 其中 d 代表 hidden dimension, $|S^{label}|$ 和 $|I^{label}|$ 代表槽数和意图标签)，在一定意义上可以看作是标签的分布。意图和插槽表示法, 在 DCA-net 的代码中，他们使用 $H \in \mathbb{R}^{n\times d}$ 作为 query, $W^V \in \mathbb{R}^{d\times|v^{label}|}$ 作为 key 和 value，从而获得意图表示 H_v.

$$A = \texttt{softmax}(HW^v); H_v = H + AW^v$$

其中，I 表示意图，S 表示 slot。最后，$H_I \in \mathbb{R}^{n\times d}$ 和 $H_S \in \mathbb{R}^{n\times d}$ 是得到的 intent representation 和 slot representation。intent representation 和 slot representation, 分别捕捉 intent 和 slot 语义信息。co-interactive module 当中, H_S 和 H_I 两个隐空间表示将被用来模拟两个任务之间的相互影响。这使得 intent representation 被 slot representation 更新, 而 slot representation 被 intent representation 表示, 从而实现两个任务之间的双向连接。DCA-net 和普通的 Vanilla transformer 一样, 首先将矩阵 H_S 和 H_I 线性映射到 query (Q_S , Q_I)、key (K_S , K_I) 和 value V_S V_I 矩阵。为了获得 slot representation，为了使得 co-interactive layer 可以捕捉到 intent 和 slot 之间的依赖关系, 他们把 slot 的隐空间表示和 value 的隐空间表示对齐. 如果将 Q_S 视为 query，K_I 视为 key，V_I 输出是一个 weighted sum, 表示 intent-aware slot representation:

$$C_S = \texttt{softmax}(\frac{Q_s K_I^T}{\sqrt{d_k}})V_I; H'_s = \texttt{LN}(H_S + C_S).$$

其中 LN 代表 layer normalization 函数。相似的, 为了实现双向的注意力机制, 要将 Q_I 视为 queries, 将 K_S 视为 key, 将 V_S 视为 value, 这样就可以得到 slot-aware intent representation, 最终得到的 $H'_S \in \mathbb{R}^{n\times d}$ 和 $H'_I \in \mathbb{R}^{n\times d}$ 可以认为分别使用了 slot 和 intent 的信息. 然后再将这两个隐空间表示送到 FFN 中, 最终使用标准的 CRF 来作为解码层.

在实践中, 我们选择采用 BIO 标注法得到的 act-slot-value 三元组作为 slot-filling 的目标, 选择 slot 槽值来作为 intent, 就比方在" 我去新郑市八千乡人民政府" 这句话中, 原来的 inform-终点名称-新郑市八千乡人民政府作为 slot-filling 的目标, 而将" 终点名称" 作为需要预测的 intent, 这个 adaption 也是挺 intuitive 的, 因为 slot field 的值可以明确的显示这句话所表达的一种意图, 而将 intent 和 slot-filling 结合起来则或许可以更好地利用两者信息.

4.2.2 实验结果

我们是实验结果基本和 baseline 持平, 原因可能是 (1). 我们的 DCA-net 的网络设计中将 intent-label 设成 slot field 的值, 而 slot-label 则是 BOI 标注法的结果, 两者几乎

是同样的东西, 所以 co-interactive cross-attention layer 其实退化成了类似 self-attentions layer, 比起 baseline 的用法并没有在模型架构上做出任何提升. (2). 我们认为我们指标分数的提升空间并不在更有效强大的模型结构上, 而在于数据的处理, 清洗和纠错上, 这是我们认为模型目前预测率还有很大提升空间的关键因素.

分数	acc
accuracy	70.28
precision	74.71
recall	72.99
f1 score	73.84

表 2: DCA-net 模型的分数.

5 不同建模方式的探索

5.1 阅读理解（MRC）

MRC [4] 的革命性之处在于们提出了一个能同时处理平面和嵌套 NER (flat NER and nested NER) 的统一框架, 并且使用 MRC(machine reading comprehensio) 任务来引导 slot-filling 任务 *e.g.*，提取带有 DEST(DESTINATION) 标签的实体被形式化为提取问题" 文中提到了那些可以作为终点的位置或者地方" 这个问题的阅读理解的答案的 span。这种表述方式自然而然地解决了嵌套 NER 中的实体重叠问题：提取两个不同类别的重叠实体需要回答两个独立的问题。同时，MRC 模型还可以通过对问题编码，加入一些先验知识，减小由于数据量太小带来的问题，在实际实验中，在数据量比较小的情况下，BERT-MRC 模型的效果要较其他模型要更好一点，很适合在缺乏标注数据的场景下使用。MRC 模型在很多数据集上都超越了之前的 SOTA, 在 English CONLL, English/Chinese OntoNotes, Chinese MSRA 这几个数据集上都拿到了很高的分数.

5.1.1 问题陈述

下面简要介绍下他们的模型架构, 给定一个输入序列 $X = \{x_1, x_2, ..., x_n\}$, 其中 n 代表序列长度, 需要找到 X 中所有命名实体, 并且给他指定一个标签 $y \in Y$, 其中 Y 是提前定义好的所有可能的 slot. (*e.g.*, destination, action, *etc*). 对于每一个 slot(或者是 intent), 他们选择采用 dataset annotation guidelines 来提供一种模板, dataset annotation guidelines 的关键是要设计出"as generic and precise as possible" 的问题. 在设计好每个 slot 相对应的问题后 (记问题为 q_y), 任务是在阅读理解的框架中从原来的 X 抽取出 $x_{\text{start,end}}$. 他们使用 BERT 作为他们的 backbone, 为了和 BERT 的输入模式对齐, 问题 q_y 和文章 X 可以连接起来, 产生 $\{[\text{CLS}], q_1, q_2, ..., q_m, [\text{SEP}], x_1, x_2, ..., x_n\}$, 其中 [CLS] and

实体	自然语言问题
poi 名称	在文本中找到位置, 包括任何形式或意义上的位置。
终点名称	在文本中任何可以作为终点位置的信息
操作	在文本中找到任何可以表征动作或行为的动词

表 3: 将不同实体类别转换为问题查询的示例。

[SEP] 是特殊 token. 将其输入 BERT 之后得到输出的 content-aware matrix $E \in \mathbb{R}^{n\times d}$, 其中 d 是 BERT 模型输出的最后维度.

5.1.2 span 选择

在 span 的选择上, 他们选择训练两个二元分类器, 一个用来预测开始位置, 一个用来预测终止位置. 这样的设计方式可以使得模型针对一个给定的 context 和 query 产生一些 candidates, 于是可以抽取出可能的所有命名实体.

起始索引预测 鉴于 BERT 输出的表示矩阵 E，模型首先预测每个标记作为起始索引的概率，如下所示:

$$P_{\text{start}} = \texttt{softmax}\left(\text{each row}(E \cdot T_{\text{start}}) \in \mathbb{R}^{n\times 2}\right) \tag{1}$$

其中 $T_{\text{start}} \in \mathbb{R}^{d\times 2}$ 是要学习的权重。P_{start} 的每一行展示了给定查询下，每个索引作为实体起始位置的概率分布。

结束索引预测 结束索引预测过程完全相同，只是使用另一个矩阵 T_{end} 来获取概率矩阵 $P_{\text{end}} \in \mathbb{R}^{n\times 2}$。

起始-结束匹配 在上下文 X 中，可能存在同一类别的多个实体。这意味着**起始索引预测**模型和**结束索引预测**模型可能预测出多个起始和结束索引. 通过对 P_{start} 和 P_{end} 的每一行应用 `argmax`，我们可以得到作为起始或结束位置的预测索引，即 $\hat{I}$start 和 $\hat{I}$end:

$$\begin{aligned} \hat{I}\text{start} &= i \mid \text{argmax}(P\text{start}^{(i)}) = 1, i = 1, \cdots, n \\ \hat{I}\text{end} &= j \mid \text{argmax}(P\text{end}^{(j)}) = 1, j = 1, \cdots, n \end{aligned} \tag{2}$$

其中上标 $^{(i)}$ 表示矩阵的第 i 行。对于任何起始索引 $i_{\text{start}} \in \hat{I}$start 和结束索引 $i_{\text{end}} \in \hat{I}$end, 我们又需要多训练一个二元分类模型来预测他们是否是合法的 start-end 匹配，如下所示:

$$Pi_{\text{start}}, j_{\text{end}} = \texttt{sigmoid}(m \cdot \texttt{concat}(E_{i_{\text{start}}}, E_{j_{\text{end}}})) \tag{3}$$

其中 $m \in \mathbb{R}^{1\times 2d}$ 是要学习的权重。

5.2 训练损失函数设计和实验

在训练时，X 与两个长度为 n 的标签序列 Y_{start} 和 Y_{end} 配对，分别表示每个标记 x_i 作为任何实体的起始索引或结束索引的真实标签。因此，我们有以下两个用于起始和结束索引预测的损失:

$$\begin{aligned} \mathcal{L}_{\text{start}} &= \text{CE}(P_{\text{start}}, Y_{\text{start}}) \\ \mathcal{L}_{\text{end}} &= \text{CE}(P_{\text{end}}, Y_{\text{end}}) \end{aligned} \tag{4}$$

设 $Y_{\text{start, end}}$ 表示是否应将每个起始索引与每个结束索引匹配的 GT。起始-结束索引匹配损失如下所示:

$$\mathcal{L}_{\text{span}} = \text{CE}(P_{\text{start,end}}, Y_{\text{start, end}}) \tag{5}$$

要最小化的整体训练目标如下所示:

$$\mathcal{L} = \alpha\mathcal{L}_{\text{start}} + \beta\mathcal{L}_{\text{end}} + \gamma\mathcal{L}_{\text{span}} \tag{6}$$

$\alpha, \beta, \gamma \in [0, 1]$ 是用于控制对整体训练目标的贡献的超参数。这三个损失在端到端的方式下联合训练，BERT 层共享参数。在测试时，首先根据 $\hat{I}$start 和 $\hat{I}_{\text{end}}$ 分别选择起始和结束索引。然后使用索引匹配模型将提取的起始索引与结束索引对齐，得出最终提取的答案。在我们的数据集上, 我们拟对于每一个槽值进行问题设计, 尽量满足问题精准简练又能够给模型提供足够信息的要求, 然后训练三个二元分类器, 但是因为训练上碰到的一些损失波动, 训练不成功的问题, 我们决定把这块内容放到 future work 当中.

5.3 指针网络

OOV（Out of vocabulary）的问题始终存在于我们的数据集中。由于训练词汇量有限，加之专有名词偏多，在转化为 embedding 的时候，这些词语将会被识别为 <unk>, 这些 OOV 词将被识别为 <unk>, 成为提高准确率的一个问题。

因此，指针网络 [5] 可能更适合 SLU 问题的表述。因为它直接从源文本中提取词序列，从而解决了 OOV 词的问题。

具体来说，传统的序列标注解决方法中，encoder 和 decoder 部分，通过 Attention Mechanism 将 encoder 的隐状态和 decoder 的隐状态结合成一个中间向量 C（context vector），然后使用 decoder 解码并预测，最后经由输出层（如 softmax）得到了针对词汇表的概率分布（绿色），从中选取概率最高的作为当前预测结果。这种情况会产生 OOV 问题，而且这种模型依赖于词库，对于一些词库中没有出现的关键词，往往会生成不出来，所以考虑能否从原文中进行复制。

加入 Pointer network 之后，我们希望输出的是针对输入文本序列的概率分布，而不是原输出层对词表中的词的分布。假设我们对于 encoder，decoder 部分保持不便，在输出层实现了一个 Pointer Networks，就可以得到针对输入序列的概率分布，对二者做并集就可以得到结合了输入文本中词汇和预测词汇表的一个概率分布，这样一来模型就有可能直接从输入文本中复制一些词到输出结果中。当然，直接这样操作未必会有好的

结果，因此又加入了一个 Pgen 来作为软选择的概率。Pgen 的作用可以这样理解：决定当前预测是直接从源文本中复制一个词过来还是从词汇表中生成一个词出来。[5] 中介绍的模型示意图见 图 7。

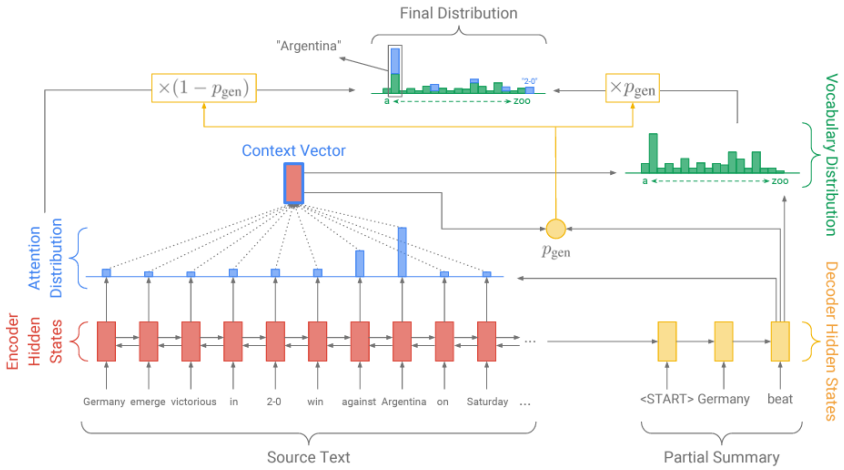


图 7: Pointer Network 模型架构

那么如何实现一个 pointer Network 呢? 查阅资料，我们发现 Pointer Network 可以理解为是 Attention Mechanism 的简化而得到的。注意力机制中所谓的针对输入序列的权重，可以把它拿出来作为指向输入序列的指针，在每次预测一个元素的时候找到输入序列中权重最大的那个元素，就是一种注意力的体现。指针网络的具体公式为:

$$u_j^i = v^T \tanh\left(W_1 e_j + W_2 d_i\right) \quad j \in (1, \ldots, n)$$
$$p\left(C_i \mid C_1, \ldots, C_{i-1}, \mathcal{P}\right) = \operatorname{softmax}\left(u^i\right)$$

第一个公式是传统注意力机制，e 和 d 分别是 encoder 和 decoder 的隐状态，得到的是一个未归一化的注意力分数。使用 softmax 函数对一次输入的所有注意力分数进行归一化，得到的就是用来对输出不同 vocab 词向量进行加权的注意力值。

但是在指针网络的第二个公式中，则是说 Pointer Networks 直接将 softmax 之后得到的概率当成了输出，将概率认为是指向输入序列特定元素的指针角色。

换句话说，传统带有注意力机制的 seq2seq 模型输出的是 encoder+decoder+ 加权后针对输出词汇表的一个概率分布，而 Pointer Networks 输出的则是针对输入文本序列的概率分布。

基于指针网络这篇论文的思想，我们设想了一种在 SLU 问题中实现指针网络的适用方法。我们使用 LSTM 作为编码器，得到最终的隐藏状态，将整个句子作为解码器的输入。作为解码器的输入，解码器也是 LSTM。这些和我们的利用注意力机制作为精确位置的指针。作为精确定位的指针。将所有 74 个不同的行为-槽对作为解码器的输入和输出 2 个指针，分别指向与给定行为槽对对应的值的开始和结束。的开始和结束。

我们认为这种指针网络可能能具有比 baseline 更好的效果，但考虑到复杂性和时间限制我们在此仅将其作为一个不同的建模方向来介绍，并欢迎在未来的工作中对模型进行改良实现。

6 实验细节和结果呈现

本节我们主要讨论实验中的具体过程和不同改进方案对提升准确性的贡献。首先说明，在我们所有的实验中，训练集作为输入模型的 utt 的语料都是 manual_transcript 中的值，我们测试发现，使用 manual_transcript 的值在所有的训练类习惯中，测试集的准确性均能有所提高，所以作为之后模型的默认的训练选项。

6.1 数据增强

在 section2.1 中，我们使用的数据增强包括不考虑增强后数据不平衡的旧增强方法，以及我们现在采用的考虑增强后数据平衡的新增强方法。它们的最终准确性效果见表 6.1

增强方法	acc	f1
`baseline`	71.62	77.62
`augment2`	76.76	80.42
`augment3`	77.09	80.61

表 4: 不同增强方法准确性对比

具体的指标在训练中的变化见 图 8

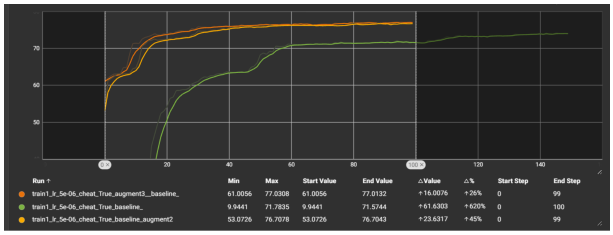
6.2 引入词典

在 section2.3 中,我们依据 [3] 的方法,使用了两种词典 `ctb.50d.vec` 和 `sgns.merge.bigram.bz2` 混合得到的 bigram+unigram 混合词典 **yj dictionary** 进行实验，最终的准确性效果见表 5, 训练图示见表 5

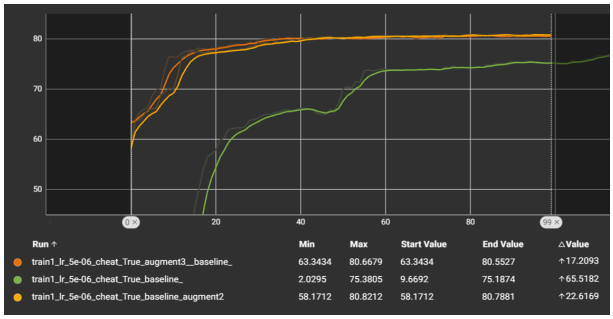
注意，我们在 baseline 上只使用了混合词典的 unigram 作为输入。使用 mix-gram 的方法是引用了开源模型 Flat Lattice 中的方法，我们仔细检查，尽量将输入输出和评价指标于我们的任务对齐。复现的方法上传到模型的 github 仓库的 Lattice 分支,

6.3 Bert 模型的性能对比

在 section3 中，我们设计了 4 种不同的 Bert 模型，包括直接使用 Bert 替换 LSTM 的 BertOnly 模型，使用两个 Bert 模型，一个作为 encoder 和一个作为 decoder 的 Bert



(a) augment acc

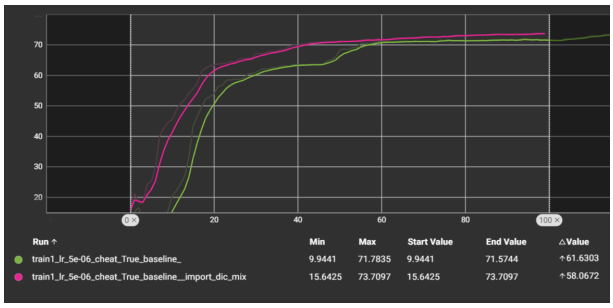


(b) augment f1

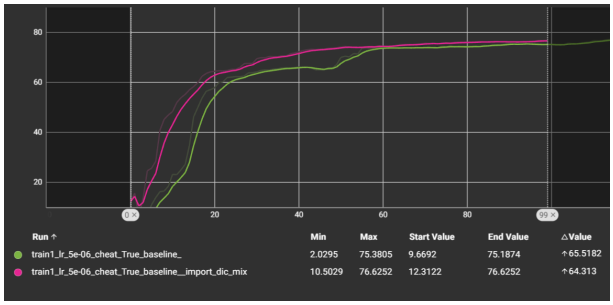
图 8: 不同增强方法下训练得到的指标

词典	acc	f1
`baseline`	71.62	77.62
`yj dic (use uni-gram)`	73.74	77.58
`Flat Lattice (use mix-gram dic)`	70.67	76.45

表 5: 不同引入词典准确性对比



(a) Imported Dictionary acc



(b) Imported Dictionary f1

图 9: 引入词典下训练得到的指标（不包含 Flat Lattice 的训练图像）

Encoder + Decoder 模型，以及使用多任务学习的 Bert Multi-task 模型，和最后的 Bert + LSTM 的串联模型。

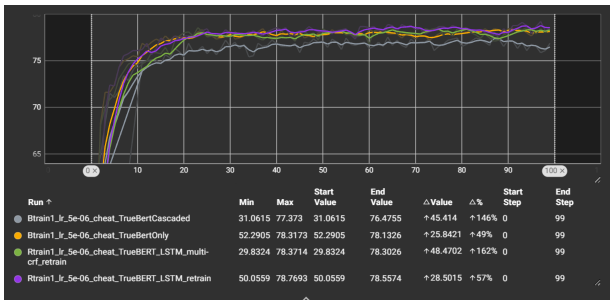
他们的性能对比见表 6 ，训练指标的图示见 图 10

模型种类	acc	f1
`baseline`	71.62	77.62
`BertOnly`	77.25	82.02
`Bert Encoder + Decoder`	77.37	80.81
`Bert Multi-task`	78.55	80.91
`Bert + LSTM`	79.22	82.17

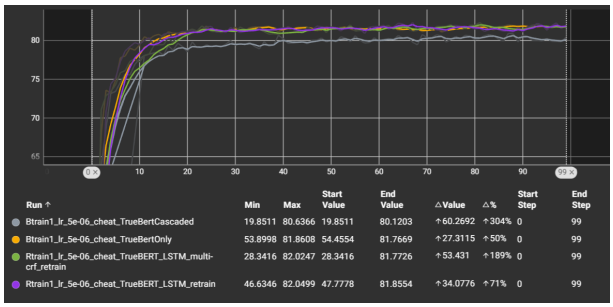
表 6: 不同模型架构的准确性对比

可以看到，在仅仅使用 Bert 的情况下，预测的准确性就已经比 baseline 模型高出许多，而且 Bert 模型由于其预训练的特点，具有一定的 zero-shot 能力，即未训练的时候的模型性能就已经达到 0.5 左右。

然而切换成 Bert+Encoder+Decoder 的架构对于模型性能的提升没有实质性的帮助，我们认为是 BertOnly 的数值代表了预训练模型充分训练后能提供的准确性提升的最大值，如果单纯再使用一个 Bert 模型，并不会对准确性由实质性的帮助，甚至可能因为模型过于复杂，发生过拟合的现象。多任务训练加上自定义输出头的方法确实在 acc 上帮助模型有所进步，提升了 0.1 左右的准确性。而 Bert + LSTM 的架构对于准确性



(a) augment acc



(b) augment f1

图 10: 不同 Bert 架构训练得到的评价指标

的帮助最大，结合之前的分析我们认为这归功于 LSTM 的加入使得可学习的参数增加，有效地增加了充分训练后模型的拟合能力。

在最终的报告中，我们只上传了 Bert+LSTM 模型的 bin 文件，但是我们提供了详细复现所有实验部分模型的重训练脚本在 github 仓库中，可供查阅。

6.3.1 错误分析

我们打印出所有不是由于测试集合噪声导致预测失败的语句, 经过统计在 bertOnly model 中，常见的错误有：

1. 空标签被错误预测为有内容
2. 有内容标签被预测为空
3. slot 预测错误，比如 poi 名称和终点名称
4. value 预测错误，比如“盖饭”预测为“吃盖饭”；“上上足”预测为“上”“上足”

我们接着查看我们的模型是否在这些错误中有所改进。我们统计了所有不是由噪音引起的模型预测错误，手动统计每个错误出现的个数（一个句子中可能出现多个错误）。相关代码集成在了模型中，可供复现。统计结果见表 7

经过统计在 Bert + Multi-head 模型中，断句错误的确被减轻了，比如错误中不再出现“上上足”和“吃德天顺盖码饭”语句的错误，证明 Multi-head 模型对于断句错误

模型	有内容识别为无内容	无内容识别为有内容	相近 slot/action 预测错	相近 value 预测错
BertOnly	3	8	7	8
Bert + Multitask	4	9	9	6
Bert + LSTM	4	7	7	4

表 7: 不同模型架构的错误类型对比

有改善的作用。但是标签预测错误的情况再 Multi-head 模型中变得更多，说明模型可能在学习 slot 的能力上受到了影响

经过统计，在 Bert + LSTM 的模型中，各项错误均有一定的减少，尤其是对于标签预测错误和空内容的预测上，进步较为明显。我们认为这说明模型的学习能力比较之前有所提升，但是对于 bertOnly model 中的问题起到的是缓解作用，没有真正地消除。

7 总结

本次组队作业的第一题中，王梓萌负责了数据预处理和模型改进和调优的工作。刘洺皓负责不同相关工作的探索和不同建模方式的探索的内容。我们共同完成了实验细节和结果呈现的部分，以及最后的总结部分。

在本项目中, 我们从数据集，模型改进以及不同建模方法和相关工作复现等多个角度，研究如何在给定的 SLU 数据集上设计模型提高准确率。我们展示了各个设计的主要模型在 development set 上的测试的结果，结果显示我们的模型能够达到 79.22 的最优准确性。

当然, 限于时间原因, 本项目未来还有很多提升空间。例如, 如果有更多的文本训练数据和更加的文本质量，设计较大的模型应该会有更好的提升，同时对于数据集的分析也应当更加地周全，在数据集预处理时考虑到更多弥补的方法。最后，限于训练能力不足，我们无法确定当前的超参数下得到的是模型的最优解，所以可能使得有些方法更加复杂但是训练的效果不如简单方法。

最后，我们也留有一些疑问尚未解决，比如说数据集增强在 baseline 模型的性能提升中具有较好的效果，但是对于 Bert 的模型，提升的效果就非常小，比如即使使用各种方法，模型在一些专有名词和一些数据集标注有歧义或者模糊的难预测的例子中的表现依然不尽如人意。如果有更多的时间，我们应当对这些细节问题进一步作探究。

参考文献

[1] Ming Xu. Pycorrector: Text error correction tool.

[2] Yue Zhang and Jie Yang. Chinese NER using lattice LSTM. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1554–1564, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[3] Xiaonan Li, Hang Yan, Xipeng Qiu, and Xuanjing Huang. FLAT: Chinese NER using flat-lattice transformer. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6836–6842, Online, July 2020. Association for Computational Linguistics.

[4] Xiaoya Li, Jingrong Feng, Yuxian Meng, Qinghong Han, Fei Wu, and Jiwei Li. A unified mrc framework for named entity recognition, 2022.

[5] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks, 2017.