CS4314 Project: Spoken Language Understanding

纪信佑, 陈景浩
{jerryji0414, yssycjh}@sjtu.edu.cn

1 介绍

在对话系统中，口语语义理解 (SLU) 是其中重要的一环。用户语音首先被识别并转换为文字，SLU 将这些文字作为输入，输出基于这些文字分析而成的语义信息。语义信息可以是结构化的也可以是无结构化的，这些语义信息将递交给对话系统的其他部分进行进一步的处理，进而生成对应的回复返回给用户。

因此我们可以将口语语义理解任务理解为根据输入的语音文本信息，对其进行语义的理解，进而输出其句子的完整的语义内容。SLU 任务因此有着许多的难点，首先是我们如何对其进行语义的理解，然后是我们如果对理解出来的语义进行合适的输出；对于其输入，由于是由语音识别系统生成的，更是有噪音掺杂在语音数据中，加大了语义理解的难度，即使是被正确识别的语音输入，由于在语义理解时不同人对同一语义由不同的表达习惯，这也让语义理解的难度大大的提升。

在本次 Project 中，我们对于 SLU 任务主要采用的是基于 learning 的方法，利用深度学习模型对 SLU 任务进行建模，将输入的文字进行一系列处理后输入模型，再将模型的输出进行解码进行最终的输出。Project 中 baseline 使用的是端到端的基于 BIO 的模型，使用了 Bi-LSTM[2] 加单层的 feed forward NN 对 SLU 进行建模，具体方法将在 Sec.2中进行说明。

对比 baseline，我们做了以下尝试对其进行效果的提升，具体的效果在 Sec.3中进行展示与讨论

a. 实现了其他模型，包括 Bert[1]，Bert+BiLSTM 和 seq2seq 的 focus 模型 [3](由于实现问题，focus 模型的效果较差，最终没有使用)

b. 实现了多种方式对训练集进行增强

我们的代码在 Github 上进行公开，链接是：https://github.com/Ji-Xinyou/Spoken-Language-Understanding。本次小组的分工中，纪信佑负责了代码的编写，实验，以及报告的撰写，工作量 60%-65%，陈景浩负责了部分实验和部分报告，工作量约 35%-40%。

目录

2 解决方法

2.1 数据增强

在本次 Project 中，给出了基于 train.json 的训练数据，但是训练数据存在着一些问题。首先最大的问题就是训练集的规模问题，训练数据中仅有两千多条数据，对于基于 learning 的方法来说，训练集的规模是保证模型效果的一个很大的因素，尤其是对于参数数量较大的模型来说，如果训练集规模过小，模型非常容易在训练集上过拟合。第二个问题就是噪音问题，训练集中噪音数据 (错误的语音 + 空的 semantic) 较少，这可能导致模型的稳定性低，面对噪音数据的攻击表现不够稳定。第三个问题就是 asr 和真实语音的不同，asr 语音中有错误识别的语音数据，可以视为噪音，因此在训练时需要考虑如何更好的在有噪音的数据下仍然输出稳定的结果，第二个问题和第三个问题我们都可以归类于噪音的问题。第四个问题则是 train.json 和 development.json 的数据分布都不均衡，其中缺少 act 为 deny 的数据，因此我们也对 development.json 和 train.json 添加了 deny 标签的数据。

在我们的数据增强解决方案中，使用了三种方式来缓解上面的三个问题。首先对于数据集的规模问题，我们对数据进行了等效的替换，对于每个数据，根据他的 semantic，我们对其进行等效的替换，比如 “我要找附近的加油站” 可能被增强为 “我要找附近的五家渠职业技术学校” 或者 “我要找近郊的加油站” 等。然后对于数据集的噪音问题，我们使用了添加噪音数据以及针对噪音的训练方式来缓解，我们生成一些无意义的句子，并且给予其空的 semantic，增加数据集的噪声数据，进而增强模型面对噪声的稳定性。

2.1.1 等效替换

在等效替换的方法中，我们仍然遵循 act-slot-value 的结构化 semantic，我们通过其 semantic 的内容，结合 ontology 的 key-value 对对其进行等效替换并加入训练集，实现数据增强，具体方法如下。

首先对于每一个数据，我们每次仅替换一个 semantic，对于某一个 semantic(即 act-slot-value)，首先其 act 必然是不变的，我们根据其 slot 作为 key，在 ontology 中寻找其 value 的范围，并在范围中随机选择一个值作为 value 构成新的 act-slot-value pair，并且将原本数据的 asr_1best 和 manual_transcript 中对应 value 的文字进行替换后，加入训练集。

举例说明，对于这样一个数据 “我要找附近的加油站”，semantic 为”inform-操作-找”，”inform-poi 目标-加油站”, ”inform-请求类型-附近”, 我们选取其中的一个 semantic，假设为”inform-poi 目标-加油站”，我们按照他的 slot 也就是 poi 目标作为 key，在 ontology 中寻找对应的 value，从 value 的取值范围中随机挑选一个值，比如” 五家渠职业技术学校”，然后将其替换。替换后的 semantic 为”inform-操作-找”，”inform-poi 目标-五家渠职业技术学校”, ”inform-请求类型-附近”，并且原文也变为 “我要找附近的五家渠职业技术学校”。

对于这样的替换，我们将 key-value 按照 ontology 分为了三类，第一类为地点名称，即 poi_name.txt 中的内容，第二类为序列号与操作，即 operation_verb.txt 与 ordinal_number.txt 中的内容，第三类为 ontology 中其他的 key-value pair，并且按照不同的比例对数据进行扩充。经过实验与对数据集的考量，我们认为第一类即地点名称这一类的增强是最重要的，因为地名往往涉及到多个名词并且比较复杂，其次为操作与序列号，最后为其他的 key-value pair。因此在对数据进行等效替换的扩充时，我们对不同类型的扩充比例也是不同的。

2.1.2 噪音数据

噪音数据的生成是为了增加模型面对噪音时的稳定性，同时让模型减轻模型在训练集上的过拟合现象。首先是生成噪音数据的数量，如果噪音数据过多，则模型会倾向于拟合噪音而非真实数据分布，因此噪音数量不宜过多，因此我们对于一个扩充后的训练集，最终将生成 2% 的噪音数据加入其中。噪音数据的生成比较简单，首先设定一些在语音数据中比较常见的字作为一个集合，然后首先随机选取噪音数据的长度，然后对于噪音数据的每一个文本处，随机挑选集合中的一个字，这样就生成了一句无意义的噪音文本，同时我们将其的 semantic 置空，这样就生成了一条噪音数据。

2.1.3 deny 类型数据

在 train.json 中，仅仅有 4 条与 deny 相关的 semantic，很容易可以看出这对于模型面对否定文本的判别是十分不合适的。同时在 development.json 即验证集中，同样 deny 的数据也非常的少，因此可以猜测，在大多实验中，训练集中缺少 deny 相关的 semantic 在验证集的测试中并不会产生不良的影响。我们可以从拟合的角度分析，对于 B/I-act-slot 这样的 tag 而言，inform 的 tag 占据了 37 个，deny 的占据了 37 个，而在训练数据中，我们几乎仅仅拟合了整个 tag 空间的一半，这将导致对 deny 这种 semantic 极为脆弱的判断能力。但是对于真正应用的模型来说，对 inform 和 deny 两种 act 都需要有良好的判断能力，因此，我们首先扩充了 train.json，增加了关于 deny semantic 的数据，让模型对 deny 的语义也有着一定的适应性。除此之外，合理地添加了 deny 的数据之后，可以让模型在整个空间上去拟合 act 和 value，理论上不会对模型判断 slot-value 的能力产生负面影响。

我们生成 deny 数据的方式基于 pattern 的，首先我们规定了一系列 pattern 表达最基本的 deny semantic，比如" 不 {op}"，" 不要 {op}"，"{op} 错了"，对应着"deny-操作-op" 这样的 semantic，同时我们还规定了一些相对复杂的 pattern，比如" 是 {poi1} 不是 {poi2} 你个笨蛋" 对应着"deny-poi 名称-poi2"，"inform-poi 名称-poi1" 这样的 semantic。简单的 deny pattern 可以让模型学习到部分 deny 的信息，而复杂的 deny pattern 可以让模型在 inform 和 deny 的两种 semantic 中进行区分，更好的拟合训练数据。

2.2 使用的模型

2.2.1 Baseline 模型

在本次使用的模型中，大多都使用的是 tagging 的方式，即将输出数据放入分类器中预测其 tag，然后结合原本对应位置的文本生成 act-slot-value pair。Baseline 模型就使用了这种方式，首先使用 word2vec 将输入语料转化为 embedding，然后将 embedding 输入 Bidirectional LSTM 生成序列的 encoding，然后将 encoding 放入分类器中给每一个 token 预测得到对应的 tag，每一个 tag 为 B-act-slot，I-act-slot 或者 O，分别表示是 semantic value 的开头，中间或者与 semantic 无关，Baseline model 的具体结构如 Fig.1。

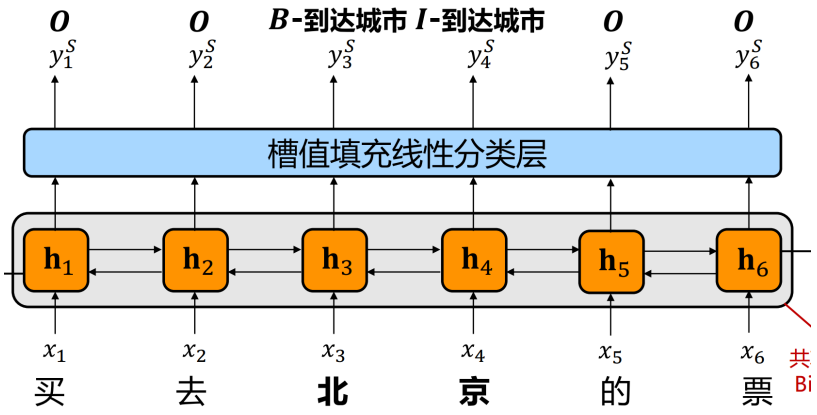

图 1: Baseline Model 结构图 (来自对话系统的 slide)

在 Baseline 模型中，分类器选用的是一个单层的 feed forward NN，考虑到当我们的数据增强后，我们应该给模型更大的 capacity，于是我们将线性层使用了一个两层的 MLP 在不同的数据规模下也进行了实验。具体的实验结果在 Sec.3 中进行展示与分析。

2.2.2 Bert 模型

Bert 全称为 Bidirectional Encoder Representations from Transformers(具体结构如 Fig2，是一个基于 Transformer 的语言模型，其中的编码层数和 self-attention head 数量是可变的。Bert 一般在上下文推断 (mask 15%)，与预测下一句的两个任务中进行 pretrain，然后在下游任务进行 finetune 即可。在本次 Project 中，我们使用的 Bert 模型即是如此，我们使用了开源的 bert-base-chinese pretrain 模型，将输入文本转化为 token 之后输入 Bert，由 Bert 生成文本对应的 encoding，再将 encoding 输入分类器得到最终的 tag。由于 Bert 已经在大规模的 corpus 上进行了预训练，因此在 finetune 的时候，我们更多的是将分类器 finetune 到我们预期的效果，因此理论上对于 bert 类型的模型，数据增强的重要性并没有 baseline 那么重要，这

一点也在 Sec.3中的实验得到了体现。在本次实验中，首先由于部分训练集数据有英文，导致 bert 在 tokenize 的时候与中文语料的 token 并不对齐 (长度小于 tag_mask)，在具体的实现中，我们将其做了 zero padding，与 mask 联合使用之后与对齐是等效的，其次由于 Bert 在 tokenize 的时候会在开头加入 CLS 在结尾加入 SEP 的 token，而在输入分类器的时候，分类器本身，tag_mask，tagid 都并不需要这些，因此我们将 bert 的输出中对应于这两个特殊 token 的 encoding 直接删去。

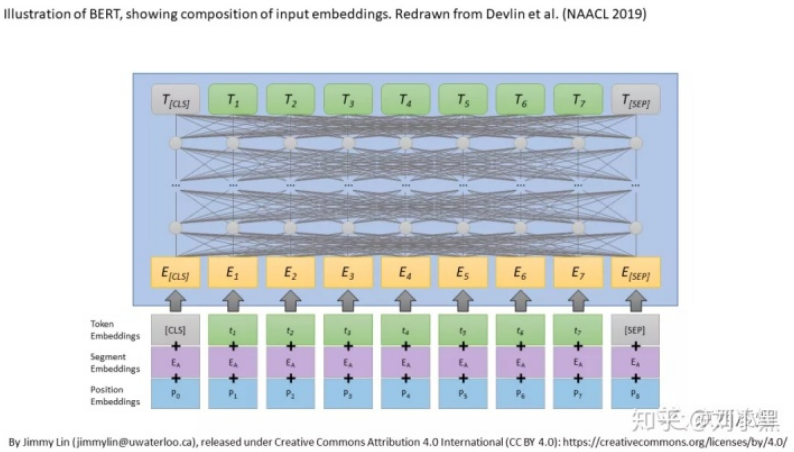

图 2: Bert 结构图

2.2.3 Bert+BiLSTM 模型

如上面所说，Bert 可以产生模型的 Encoding，而且保留了序列的顺序，于是我们尝试将其代替原本 Baseline 模型的 word2vec 的 embedding，探求其效果是否会更好，具体的实验结果在 Sec.3中进行了分析与讨论。具体的 pipeline 和 baseline 模型类似，即首先通过 pretrain 的 BertTokenizer 将文字转换为 token 序列，然后对序列进行和 Bert 模型中相同的处理，即对数据序列进行 padding 对齐并且在序列中删除 CLS 和 SEP 对应的的 encoding，然后将处理后的 embedding 数据输入 BiLSTM 产生新的 encoding 序列，最后将这个 encoding 序列送入分类器，产生预测的 tag，在我们的实验中，分类器是一个两层的 MLP。

3 实验结果与分析

下面的表格中 Train Scale 的格式为 "a, b, c, d"。其中假设原本数据集的大小为 x，那么扩充的数据集中，根据 poi_name 类型扩充的数据大小大约为 ax，根据序列号和操作扩充的数据大小大约为 bx，根据其他 slot-value 扩充的数据大小大约为 cx，deny 类型数据的大小大约为 dx。

3.1 数据增强效果分析

在本部分中，我们以 Baseline 模型为基础，分别分析数据增强的效果，包括等效替换的效果，噪音添加的效果以及 deny 类型数据的效果。并且在大致的总览即 Tab.1中给出不同模型在不同数据增强下的大致效果与实验结果。其中关于噪音数据的添加效果将在 Sec.3.1.2中单独进行讨论与分析。

表 1: 实验结果大致总览

Model	**Train Scale**	**Noise**	**Dev Acc**	**Dev Fscore(p/r/f)**
Baseline	0,0,0,0	×	68.49	77.4487/70.9072/74.0338
Baseline	1,1,1,1	×	73.07	78.7554/76.5381/77.6309
Baseline	2,2,2,2	×	74.97	80.8396/78.3107/79.5551
Baseline	2,2,2,0	×	74.97	80.9935/78.2065/79.5756
Baseline	2,1,1,3	×	75.08	80.5763/78.7278/79.6414
Baseline	3,1,1,0	×	75.64	80.2116/79.0407/79.6218
Baseline	3,1,1,3	×	76.31	81.7112/79.6663/80.6758
Bert	0,0,0,0	×	79.22	85.8250/80.8133/83.2438
Bert	1,1,1,1	×	79.66	84.7134/83.2117/83.9558
Bert	2,2,2,2	×	79.66	85.0427/83.0031/84.0106
Bert	2,2,2,0	×	79.55	84.2887/82.7946/83.5350
Bert	2,1,1,3	×	79.78	85.8225/82.6903/84.2273
Bert	3,1,1,0	×	79.66	85.0538/82.4818/83.7480
Bert	3,1,1,3	×	80.00	84.3783/82.7946/83.5789
BertLSTM	0,0,0,0	×	78.55	83.5294/81.4390/82.4710
BertLSTM	1,1,1,1	×	79.89	84.8195/83.3160/84.0610
BertLSTM	2,2,2,2	×	79.33	83.7553/82.7946/83.2722
BertLSTM	2,2,2,0	×	79.33	83.6325/82.5860/83.1060
BertLSTM	2,1,1,3	×	79.89	85.4545/83.3160/84.3717
BertLSTM	3,1,1,0	×	79.44	83.6134/83.0031/83.3072
BertLSTM	3,1,1,3	×	**80.11**	**85.4545/83.3160/84.3717**

在上表中，我们可以看出，我们最终最优的模型是 **BertLSTM**，数据增强的 Train Scale 为 **3, 1, 1, 3**，达到了 **Dev Acc=80.11** 与 **Fscore=84.3717** 的最终效果。我们所有实验训练的命

令与最终的结果可以在 history.log 中查看，训练数据的生成方法也在 README 中给出，我们最终上交的模型也是此模型。

3.1.1 等效替换

在本小节中，我们主要测试等效替换单独作为数据增强的效果 (noise=False, Train Scale=(a,b,c,0))，将 Baseline 模型分别在没有增强 (Train Scale = (0,0,0,0))，以及不同程度的等效替换增强下进行训练，比较最后的 Dev Acc 以及 Dev FScore，实验的结果如 Tab.2.

表 2: 等效替换实验结果

Model	Train Scale	Noise	Dev Acc	Dev Fscore(p/r/f)
Baseline	0,0,0,0	×	68.49	77.4487/70.9072/74.0338
Baseline	1,1,1,0	×	72.51	78.4188/76.5381/77.4670
Baseline	2,1,1,0	×	74.97	81.0289/78.8321/79.9154
Baseline	2,2,2,0	×	74.97	80.9935/78.2065/79.5756
Baseline	3,1,1,0	×	75.64	80.2116/79.0407/79.6218
Baseline	3,3,3,0	×	76.54	81.7792/79.5620/80.6554

从上表中我们可以看出，首先，数据量增加后，Dev Acc 与 Dev FScore 两项指标是呈上升趋势的，这说明了两点，一是等效替换的数据增强方式是有效的，模型可以从等效替换的添加数据中学习到 semantic 内容，这也是符合常理的，因为我们对于训练数据的增强是几乎等效的，仅仅替换了 value，二是有效数据规模的增大对 SLU 任务的性能有着较大的提升，可以看出，我们将训练数据提升大约十一倍后，(从 2933->33307)，Baseline 模型的 Dev Acc 从 68.49 上升到了 76.54，而 Dev FScore 也从 74.03 上升到了 80.66，这个提升幅度是非常大的，并且伴随着训练数据的增加，两项指标的呈现的基本是上升趋势，可以想象当数据量继续增加时，性能应该还可以有一定的提升空间。

3.1.2 噪音添加

在本小节中，我们主要测试噪音数据对数据增强的效果，对于同一个 Train Scale，我们为其增加 2% 的噪音数据，观察其在验证集上的表现。我们分别在 Baseline 模型上，用 Train Scale=(0, 0, 0, 0), (1, 1, 1, 1), (2, 2, 2, 0)，分别添加噪音，与没有噪音的训练结果相对比，将 Dev Acc 和 Dev FScore 作为指标评价性能，实验结果如 Tab.3.

表 3: 噪音数据实验结果

Model	Train Scale	Noise	Dev Acc	Dev Fscore(p/r/f)
Baseline	0,0,0,0	×	68.49	77.4487/70.9072/74.0338
Baseline	0,0,0,0	✓	68.94	78.5304/71.3243/74.7541
Baseline	1,1,1,1	×	72.51	78.4188/76.5381/77.4670
Baseline	1,1,1,1	✓	73.07	78.7554/76.5381/77.6309
Baseline	2,2,2,0	×	74.97	80.9935/78.2065/79.5756
Baseline	2,2,2,0	✓	75.30	80.0650/77.0594/78.5335

首先观察实验结果，可以看出，噪音数据的增加对于 Baseline 模型在相同 Train Scale 下的训练结果 (Dev Acc 与 Dev FScore) 均是有提升的。噪音数据的生成非常的简单，并且 semantic 是置空的，我们认为噪音数据对于训练结果的提升对模型从数据角度起到了一定的正则化的效果，减少了模型对于训练数据的过拟合，并且增加了模型对于文本 semantic 的泛化性。

3.1.3 deny 类型数据

在本小节中，我们主要测试 deny 类型数据对数据增强的效果。我们分别控制了两个变量进行了实验，首先是数据的总量，当 Train Scale 为 (1, 1, 1, 0) 和 (1, 1, 0, 2) 时，与 (2, 2, 2, 0) 和 (2, 1, 1, 3) 的数据总量是基本一致的，然后是是否添加 deny 数据，即比如 (3, 1, 1, 0) 和 (3, 1, 1, 3) 这一组数据，观察其 Dev Acc 和 Dev FScore 的结果，评估性能，最终的结果如 Tab.4.

表 4: deny 类型数据实验结果

Model	Train Scale	Noise	Dev Acc	Dev Fscore(p/r/f)
Baseline	0,0,0,0	×	68.49	77.4487/70.9072/74.0338
Baseline	1,1,1,0	×	72.51	78.4188/76.5381/77.4670
Baseline	1,1,0,2	×	73.97	80.0650/77.0594/78.5335
Baseline	2,2,2,0	×	74.97	80.9935/78.2065/79.5756
Baseline	2,1,1,3	×	75.08	80.5763/78.7278/79.6414
Baseline	3,1,1,0	×	75.64	80.2116/79.0407/79.6218
Baseline	3,1,1,3	×	76.31	81.7112/79.6663/80.6758

我们在 Sec.2.1.3中对于添加 deny 数据这一增强方式给出的判断是，deny 数据可以让模型更好的拟合到 slot-value 的分布，进而提升模型在同样数据规模下的性能，在 development.json 即验证集缺少 deny 数据的情况下，模型如果学习到了正确的 semantic，也不会让模型有更差的性能，在表中观察实验结果也证实了这一点。首先，在相同的数据规模下，即 Train Scale 为 ((1, 1, 1, 0) 与 (1, 1, 0, 2)) 和 ((2, 2, 2, 0) 与 (2, 1, 1, 3)) 两组对照中，两组的 Dev Acc 和 Dev FScore 都得到了一定的提升。同时，在对某一数据规模额外添加 deny 类型的数据时，模型的性能也得到了较大的提升，即表中 Train Scale 为 (3, 1, 1, 0) 和 (3, 1, 1, 3) 的一组，可以看到 Dev Acc 提高了 0.67，Dev FScore 提升了 0.95。

3.2 模型效果分析

在本小节中，我们主要讨论不同模型在同一个 Train Scale 下的效果，对于同一个 Train Scale，我们分别使用 Baseline, Bert 和 BertLSTM 在上面训练，观察最终的 Dev Acc 与 Dev FScore 评估最终的性能，测试的结果如 Tab.5.

表 5: 不同模型实验结果

Model	Train Scale	Noise	Dev Acc	Dev Fscore(p/r/f)
Baseline	0,0,0,0	×	68.49	77.4487/70.9072/74.0338
Baseline	2,2,2,0	×	74.97	80.9935/78.2065/79.5756
Baseline	2,1,1,3	×	75.08	80.5763/78.7278/79.6414
Baseline	3,1,1,0	×	75.64	80.2116/79.0407/79.6218
Baseline	3,1,1,3	×	75.64	80.2116/79.0407/79.6218
Bert	0,0,0,0	×	79.22	85.8250/80.8133/83.2438
Bert	2,2,2,0	×	79.55	84.2887/82.7946/83.5350
Bert	2,1,1,3	×	79.78	85.8225/82.6903/84.2273
Bert	3,1,1,0	×	79.66	85.0538/82.4818/83.7480
Bert	3,1,1,3	×	80.00	84.3783/82.7946/83.5789
BertLSTM	0,0,0,0	×	78.55	83.5294/81.4390/82.4710
BertLSTM	2,2,2,0	×	79.33	83.6325/82.5860/83.1060
BertLSTM	2,1,1,3	×	79.89	85.4545/83.3160/84.3717
BertLSTM	3,1,1,0	×	79.44	83.6134/83.0031/83.3072
BertLSTM	3,1,1,3	×	**80.11**	**85.4545/83.3160/84.3717**

在不同规模的数据以及模型下，我们产生的最优的结果是 BertLSTM，在 Train Scale=(3, 1, 1, 3) 的数据下训练，在验证集上达到了 80.11 的 Dev Acc 以及 84.37 的 Dev FScore。总体而言，Baseline 模型在同样数据规模的情况下性能是最差的，Bert 和 BertLSTM 的性能在同样规模的数据上大致一致；继续观察可以看出，在没有 deny 类型数据添加的情况下，Bert 模型性能要好过 BertLSTM 的性能，而在有 deny 类型数据添加的情况下，则 BertLSTM 的性能要略微好一些。Bert 和 BertLSTM 性能更好的原因也很明显，Bert 本身在极大规模的 corpus 上进行了预训练，已经有着对输入文本极好的 encoding，远远胜于原本的 word2vec 产生的 encoding，因此将其输入 LSTM 或者分类器之后，对 tag 的预测也更加准确。

4 总结

在本次 Project 中，我们实现了 Bert 模型，BertLSTM 模型和 Focus 模型 (实现可能有问题，效果极差)，并且实现了三种数据增强方式，分别为等效替换，添加噪音数据以及添加 deny 类型数据。通过不同模型以及数据增强的实现和在不同条件下的多次实验得到的数据比对，我们得到以下结论。

a. 数据规模对于 SLU 任务的效果有着至关重要的作用，对于 Bert 类模型而言，由于其在大规模 corpus 上已经进行了无监督预训练，因此其本身可以视作有着较大的数据规模，而对于 Baseline，数据规模的增大对其性能的提升也是非常明显的，数据规模扩大 4 倍，Baseline 的 Dev Acc 可以提升约 5%，数据规模扩大 11 倍，则可以提升约 8%。

b. 噪音的加入可以让模型在 SLU 任务上有着更优的性能，减少模型在训练数据上的过拟合，并且噪音的生成方式非常简单，2% 的噪音数据就可以给 Baseline 模型带来大约 0.5% 的 Dev Acc 性能提升。

c. 训练数据的合理分布对于模型的性能有着一定的影响，在原本的训练和验证数据中缺少 deny 类型的数据，在训练数据中加入一定数量的 deny 类型的数据，让训练数据分布的更均匀，可以让模型更好的学到 slot-value 的 semantic，即使在缺少 deny 类型的验证集上也表现的更好，在 5 倍等效替换扩充的数据集上加入原本 3 倍的 deny 类型数据，可以让 Dev Acc 从 79.44 提升到 80.11，性能的提升十分明显。

d. 对于模型的选择，可以选择预训练的大模型直接对产生的 encoding 进行分类，或者将 encoding 视作 word vector 然后在下游任务上 finetune，大模型在较小的数据上 finetune 就会有相对不错的结果。本文中提到的三种数据增强方式，在扩充数据规模后，仍然对模型性能有一定的提升。

参考文献

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[2] Z. Huang, W. Xu, and K. Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[3] S. Zhu and K. Yu. Encoder-decoder with focus-mechanism for sequence labelling based spoken language understanding. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5675–5679. IEEE, 2017.