本次实验的内容为关系抽取，所用模型为论文 Relation Classification via Convolutional Deep Neural Network.(ACL 2014)所提出的模型，学习框架为 Pytorch，所用数据集为 SemEval 2010 Task8。数据集内部共有 19 种关系，8000 条训练数据与 2717 条测试数据。

关系抽取是自然语言处理的一个经典任务，目标为给定标注了两个实体的句子，返回这条句子中所能显示出来的两个实体之间的语义关系。

注：

1. 部署 Pytorch 环境时可能会出现 numpy 包版本匹配的问题，若出现这类问题可以使用 Anaconda 创建一个只装有该实验所使用包的新环境，在该新环境下进行实验。
2. 实验训练时间较长，在无 gpu 加速的情况下大概需要运行 40 分钟左右。
3. 本实验的 Lexical Level 特征中并没有论文中所提到的 L5：上位词。
4. 其实 PCNN 是这篇论文作者后面一篇 paper 提出的模型，但这篇论文提出的模型没有名字(逃

模型介绍：

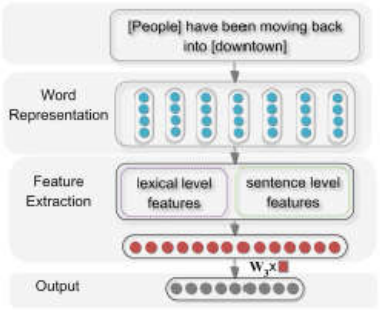

Figure 1: Architecture of the neural network used for relation classification.

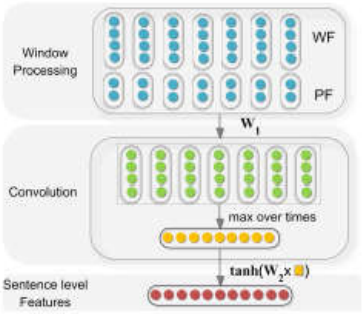

Figure 2: The framework used for extracting sentence level features.

模型的整体框架如图 1 所示，将带有实体标注的句子进行词向量化后进行特征抽取，得到词汇级别的特征和句子级别的特征，再将两个级别的特征向量拼接后经过全连接层得到维度与关系标签数相同的向量，再对得到的向量进行 soft-max 后输出。

其中词汇级别的特征结构如表 1 所示，由实体 1，实体 2，实体 1 左右的单词，实体 2 左右的单词，以及两个实体在语义网中的上位词组成，本次代码实验提供的数据中没有 L5 即两个实体在语义网中的上位词；语句级别的特征提取过程如图 2 所示，首先拼接单词与位置信息的向量化表示，然后通过卷积层与使用最大池化的池化层，再经过使用 tanh 激活函数的全连接层得到。

模型的损失函数为交叉熵损失函数，使用 Adam 方法进行参数优化。

参数设定：

词向量维度：50

位置向量维度：5
卷积窗口大小：3
通道数：200(卷积核的个数)
W2 隐层：100(卷积后的全连接层)

实验要求：

补充模型/models/PCNN.py 中的网络结构与前向函数。
其中算法的损失函数和优化方式在中附件中给出，不需要实现。
搭建并训练模型，并得到模型在测试集上的性能。

实验步骤：

网络结构：

定义并初始化以下结构：
Word Embedding：单词的向量化表示，从已有的 word embedding 中 load 出来
Position Embedding1：与句子中第一个实体的相对位置所对应的向量化表示
Position Embedding2：与句子中第二个实体的相对位置所对应的向量化表示
卷积层：结构图中的 W1，本模型中使用的是 200 个大小为(3, Word Embedding + Position Embedding1 Dim + Position Embedding2 Dim)的卷积核。在第一维设置 same padding，第二位设置 valid padding。
全连接层 1：结构图中的 W2
全连接层 2：结构图中的 W3

具体参数与模型结构可以在实验中尝试更改，若进行更改，需要在实验报告中写出。

前向传播函数：

输入为长度为 4 的列表，四个元素的尺寸分别为(batch size, 6),(batch size, word embedding dim), (batch size, position embedding dim),(batch size, position embedding dim)
输出要求为尺寸为(batch size, label num)的 Tensor
其中第一维 batch size 是批量处理数据的条数，每条数据之间相互独立，在下面的介绍中将省略这一维度。

WE，P1E，P2E 分别为单词，当前位置相对实体 1 的距离，当前位置相对实体 2 的距离的向量化表示。

WE(): word embedding

P1E(): left position embedding

P2E(): right position embedding

输入为长度为 4 的列表，其中 lexical feature 为(实体 1 的单词,实体 1 左边的单词,实体 1 右边的单词,实体 2,实体 2 左边的单词,实体 2 右边的单词)六元组。

Input = (lexical feature, sentence, left position, right position),

Shape = [(6), (max sentence length), (max sentence length), (max sentence length)]

将输入的信息向量化后得到 Lexical Level 与 Sentence Level 的向量。

Lexical1 = WE(lexical feature)

Shape=(6, word embedding dim)

Sentence1 = (WE(sentence), P1E (left position), P2E (right position))

Shape = (max sentence length, word embedding dim + left position embedding dim + right position embedding)

对 Sentence Level 的特征进行二维卷积操作，得到的结果再经过一个 max pooling 层和一个全连接层。

Sentence2 = ReLu(Conv2D(Sentence1))

Shape = (filters num, max sentence length)

Sentence3 = max-pooling(Sentence2)

Shape = (filters num)

Sentence4 = tanh(Linear(Sentence3))

Shape = (W2 hidden layer dim)

最后将处理过后的 Sentence Level Feature 与 Lexical Level Feature 拼接，经过全连接层得到维度为标签数的向量后输出。

Output = Linear([Sentence4, Lexical1])

Shape = (label num)

此时输出的结果为一个大小为标签数的向量，其中最大的是这条数据最有可能所对应的标签。若希望最后输出的向量所有维度之和为 1，可以再接上一个 soft-max 操作，但由于这份代码实现在外部代码中计算损失函数时会对输出进行一次类似的操作（Pytorch 在使用 CrossEntropyLoss 时会对输出进行一次 log_softmax 操作），故在本次实验中不需要在模型最后加上 soft-max。

可能使用到的库、对象与函数

下面将会列出可能在实验过程中使用的库、对象与函数，详细的使用方法可以自行在网上进行查询

torch.nn：包含了神经网络设计中常用到的层

- torch.nn.Embedding：embedding 层
- torch.nn.Conv2d：2 维卷积层
- torch.nn.MaxPool1d：1 维最大池化层
- torch.nn.Linear：线性变换层
- torch.nn.Tanh：Tanh 激活层，其余非线性变换层不在这里列出

torch.nn.init：常用的神经网络参数初始设置函数

- torch.nn.init.xavier_normal_：Xavier 初始化

torch.nn.functional：常用函数

- torch.nn.max_pool1d：1 维最大池化函数
- torch.nn.tanh：Tanh 激活函数

torch.cat：按指定维度拼接多个张量
[tensor].squeeze：删除 tensor 类型变量的指定维度，要求删除的维度大小为 1
[tensor].unsqueeze：为 tensor 类型变量增加一维到指定维度，增加的维度大小为 1
[tensor].type：将 tensor 类型变量转换其他数据类型的 tensor 类型变量(如将 IntTensor 转换为 LongTensor)