

关系抽取实验

一、基于卷积神经网络的关系抽取

(1) 补充 ./CNN/model.py 的 forward 部分，跑通训练代码，给出训练结果

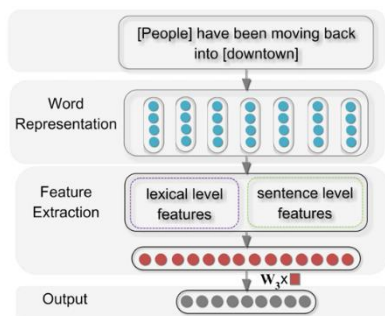


Figure 1: Architecture of the neural network used for relation classification.

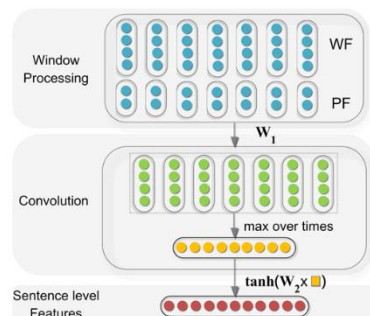


Figure 2: The framework used for extracting sentence level features.

根据论文中的图示，该网络接受一个输入句子并进行多个层次的特征提取，主要包括以下三个组件：词表示、特征提取和输出。

系统的输入是一个带有两个标记名词的句子。然后，通过查找单词嵌入将单词标记转换为向量。接着，分别提取词汇和句子级别的特征，然后直接连接以形成最终的特征向量。

- 在句子级特征提取中，每个标记进一步表示为单词特征(WF)和位置特征(PF)。然后，向量通过卷积和非线性变换获得句子级别的特征。

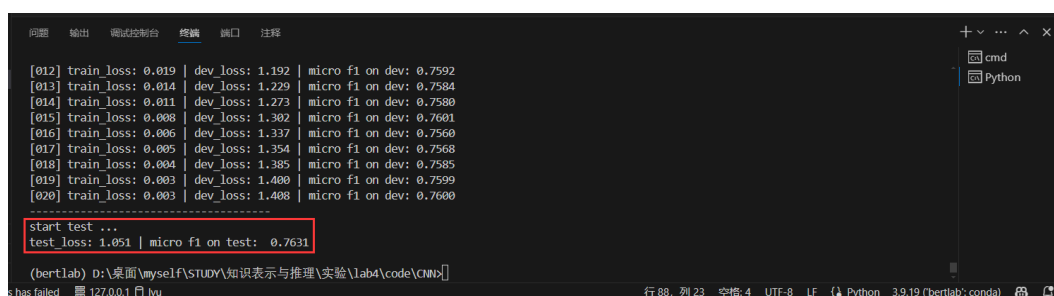
最后，为了计算每个关系的置信度，将特征向量馈送到 softmax 分类器中。分类器的输出是一个向量，其维度等于预定义的关系类型数。每个维度的值是相应关系的置信度得分。

因此可将 CNN 的推理过程 forward 函数按照上述图补充如下：

- 首先通过 **encode layer** 将 token 和位置编码混合
- 而后通过卷积层进行特征提取
- 最后通过 **Max pooling** 层和 **tanh** 激活函数的线性层

```
106 #todo 根据原论文补充
107 # Encoder Layer
108 emb = self.encoder_layer(token, pos1, pos2)
109 # Convolution Layer
110 conv = self.conv_layer(emb, mask)
111 # Maxpool Layer
112 sentence_feature = self.single_maxpool_layer(conv)
113 # Dense Layer
114 sentence_feature = self.linear(sentence_feature)
115 sentence_feature = self.tanh(sentence_feature)
116
117 sentence_feature = self.dropout(sentence_feature)
118 logits = self.dense(sentence_feature)
119 return logits
```

最后训练得到结果：test_loss: 1.051 | micro f1 on test: 0.7631 (>82.7*0.9):



```
[012] train_loss: 0.019 | dev_loss: 1.192 | micro f1 on dev: 0.7592
[013] train_loss: 0.014 | dev_loss: 1.229 | micro f1 on dev: 0.7584
[014] train_loss: 0.011 | dev_loss: 1.273 | micro f1 on dev: 0.7580
[015] train_loss: 0.008 | dev_loss: 1.302 | micro f1 on dev: 0.7601
[016] train_loss: 0.006 | dev_loss: 1.337 | micro f1 on dev: 0.7560
[017] train_loss: 0.005 | dev_loss: 1.354 | micro f1 on dev: 0.7568
[018] train_loss: 0.004 | dev_loss: 1.385 | micro f1 on dev: 0.7585
[019] train_loss: 0.003 | dev_loss: 1.400 | micro f1 on dev: 0.7599
[020] train_loss: 0.003 | dev_loss: 1.408 | micro f1 on dev: 0.7600
-----
start test ...
test_loss: 1.051 | micro f1 on test: 0.7631
(bertlab) D:\桌面\myself\STUDY\知识表示与推理\实验\lab4\code\CNN\>
```

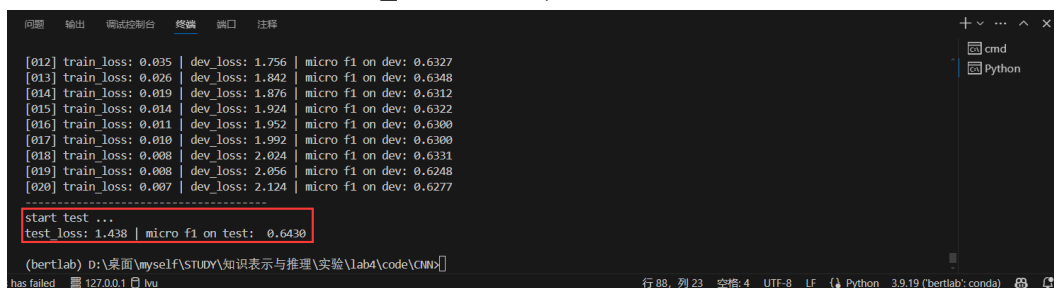
(2) 进行消融实验, 尝试去除 PF(Position features)重复实验

将(1)中 encoder_layer 中输入改为 `emb = self.encoder_layer(token, 0, 0)`, 即所有输入的位置编码都设置为 0;



```
107 # Encoder Layer
108 # # 消融实验, 去除位置信息, 填充值为0
109 pos1 = torch.zeros_like(pos1)
110 pos2 = torch.zeros_like(pos2)
111 emb = self.encoder_layer(token, pos1, pos2)
112 # emb = self.encoder_layer(token, 0, 0)
```

最后训练得到结果：test_loss: 1.438 | micro f1 on test: 0.6430



```
[012] train_loss: 0.035 | dev_loss: 1.756 | micro f1 on dev: 0.6327
[013] train_loss: 0.026 | dev_loss: 1.842 | micro f1 on dev: 0.6348
[014] train_loss: 0.019 | dev_loss: 1.876 | micro f1 on dev: 0.6312
[015] train_loss: 0.014 | dev_loss: 1.924 | micro f1 on dev: 0.6322
[016] train_loss: 0.011 | dev_loss: 1.952 | micro f1 on dev: 0.6300
[017] train_loss: 0.010 | dev_loss: 1.992 | micro f1 on dev: 0.6300
[018] train_loss: 0.008 | dev_loss: 2.024 | micro f1 on dev: 0.6331
[019] train_loss: 0.008 | dev_loss: 2.056 | micro f1 on dev: 0.6248
[020] train_loss: 0.007 | dev_loss: 2.124 | micro f1 on dev: 0.6277
-----
start test ...
test_loss: 1.438 | micro f1 on test: 0.6430
(bertlab) D:\桌面\myself\STUDY\知识表示与推理\实验\lab4\code\CNN\>
```

具体将两次实验对比如下表:

表格 1 消融实验对比

	Test loss	micro f1 on test	Dev loss	micro f1 on dev	Train loss
CNN w PF	1.051	0.7631	1.408	0.7600	0.003
CNN w/o PF	1.438	0.6430	2.124	0.6277	0.007

显然, 仅通过 WF 无法捕捉到这种结构信息。为此, PF (当前单词到 w_1 和 w_2 的相对距离的组合) 被提出用于关系分类。根据当前单词到 w_1 和 w_2 的相对距离得到距离向量 d_1 和 d_2 , $PF = [d_1, d_2]$ 。将 WF 和 PF 结合起来, 单词表示为 $[WF, PF]^T$, 随后将其输入到算法的卷积层中。

二、 远程监督关系抽取

在 nyt10m 数据集上分别按照下述四种方式进行训练，输入命令行和结果 AUC(accuracy)以及 F1 如下：

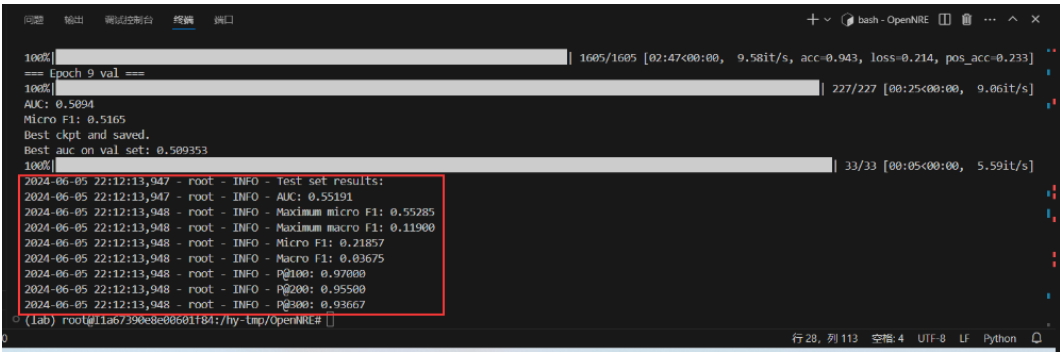
表格 2 测试集上远程监督关系抽取结果（加粗为最好结果，下划线为次好结果）

	AUC	Maximum micro F1	Maximum macro F1	Micro F1	Macro F1	P@100	P@200	P@300
cnn+att	0.55	0.55	0.12	0.22	0.04	0.97	0.95	0.93
cnn+avg	0.53	0.54	0.11	0.16	0.03	0.97	0.95	0.90
pcnn+att	0.58	0.57	0.17	0.31	0.05	0.97	0.95	0.94
pcnn+avg	<u>0.56</u>	<u>0.56</u>	<u>0.14</u>	<u>0.29</u>	<u>0.05</u>	<u>0.95</u>	<u>0.93</u>	<u>0.90</u>

详细命令行和训练截图如下：

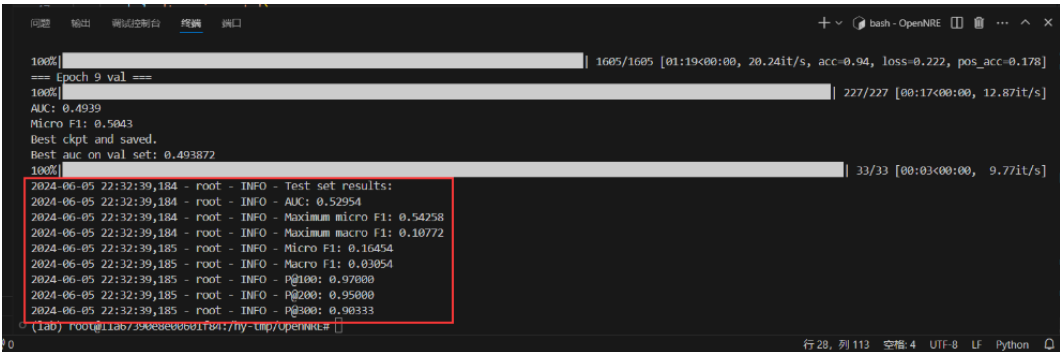
● 使用 **cnn** 作为编码器，设置--aggr 为 **att**，也就是使用句子级注意力，训练以及推理，报告 AUC 以及 F1 值

输入命令行：`python example/train_bag_cnn.py --metric auc --dataset nyt10m --batch_size 160 --lr 0.1 --weight_decay 1e-5 --max_epoch 10 --max_length 128 --seed 42 --encoder cnn --aggr att`



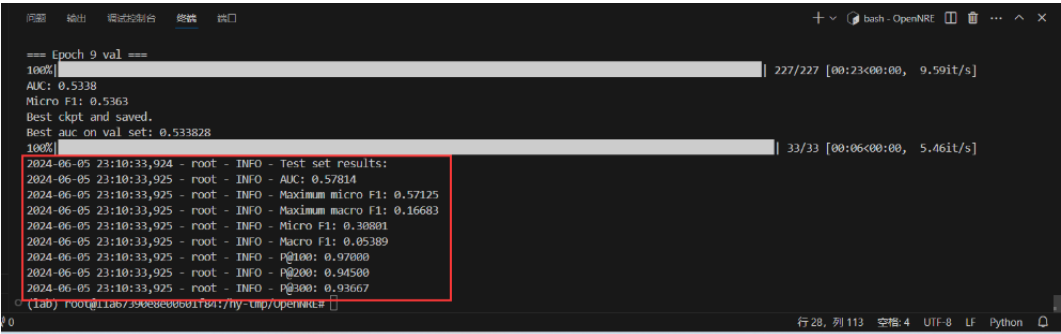
● 使用 **cnn** 作为编码器，设置--aggr 为 **avg**，也就是使用句子平均向量，训练以及推理，报告 AUC 以及 F1 值

输入命令行：`python example/train_bag_cnn.py --metric auc --dataset nyt10m --batch_size 160 --lr 0.1 --weight_decay 1e-5 --max_epoch 10 --max_length 128 --seed 42 --encoder cnn --aggr avg`



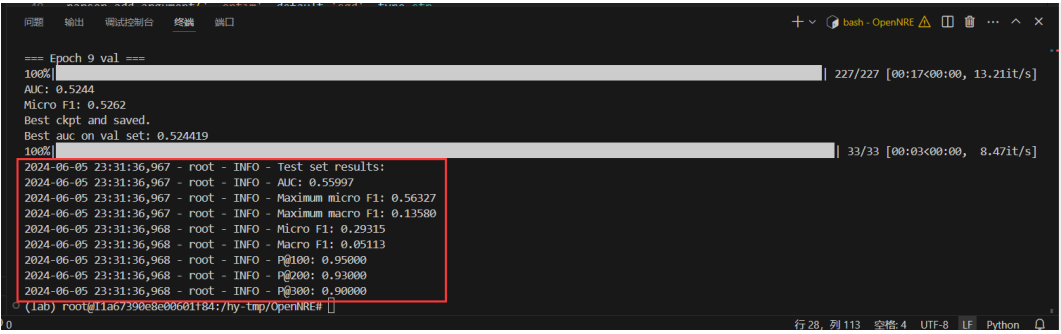
● 使用 **pcnn** 作为编码器，设置--aggr 为 **att**，也就是使用句子级注意力，训练以及推理，报告 AUC 以及 F1 值

输入命令行：`python example/train_bag_cnn.py --metric auc --dataset nyt10m --batch_size 160 --lr 0.1 --weight_decay 1e-5 --max_epoch 10 --max_length 128 --seed 42 --encoder pcnn --aggr att`

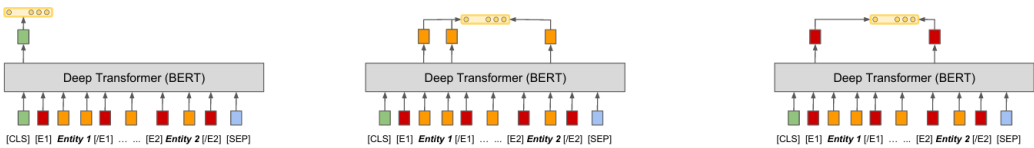


● 使用 `pcnn` 作为编码器，设置 `--aggr` 为 `avg`，也就是使用句子平均向量，训练以及推理，报告 AUC 以及 F1 值

输入命令行：
`python example/train_bag_cnn.py --metric auc --dataset nyt10m --batch_size 160 --lr 0.1 --weight_decay 1e-5 --max_epoch 10 --max_length 128 --seed 42 --encoder pcnn --aggr avg`



三、 预训练模型关系抽取



(d) ENTITY MARKERS – [CLS] (e) ENTITY MARKERS – MENTION POOL. (f) ENTITY MARKERS – ENTITY START

最后在两个设置下的结果如下表所示：

表格 3 预训练模型关系抽取结果

	Train accuracy	Eval accuracy	Eval f1	Eval precision	Eval recall
w ENTITY START	0.84	0.82	0.76	0.77	0.76
w [CLS]	0.80	0.80	0.75	0.77	0.72

具体运行截图如下：

(1) 运行 main_task.py 代码，要求复现 accuracy>0.74 (ENTITY MARKERS + ENTITY START)

3 个 epoch 结束后，训练结束后在训练集上的训练 accuracy 为 0.84，在验证集 eval 上 accuracy=0.82，f1=0.76，precision=0.77，recall=0.76；

```
Epoch: 3, 7200/ 8000 points] total loss, accuracy per batch: 0.453, 0.865
Epoch: 3, 8000/ 8000 points] total loss, accuracy per batch: 0.476, 0.860
06/06/2024 01:54:20 PM [INFO]: Evaluating test samples...
06/06/2024 01:54:36 PM [INFO]: ***** Eval results *****
06/06/2024 01:54:36 PM [INFO]: accuracy = 0.8192578993914807
06/06/2024 01:54:36 PM [INFO]: f1 = 0.7638230647709321
06/06/2024 01:54:36 PM [INFO]: precision = 0.7668517049960349
06/06/2024 01:54:36 PM [INFO]: recall = 0.7608182533438238
Epoch finished, took 610.37 seconds.
Losses at Epoch 3: 0.5065252
Train accuracy at Epoch 3: 0.8423750
Test f1 at Epoch 3: 0.7638231
06/06/2024 01:54:36 PM [INFO]: Finished training!
06/06/2024 01:54:40 PM [INFO]: Loading tokenizer and model...
06/06/2024 01:54:41 PM [INFO]: loading configuration file https://s3.amazonaws.com/models.huggingface.co/bert/albert-base-v2-config.json from c
```

(2) 修改成 ENTITY MARKERS+[CLS]并重复实验。

由于形状对齐，这里将[CLS]重复两遍与上述代码的维度对齐，而后按照 ENTITY MARKERS + ENTITY START 的思路仿写代码如下：

```
600 cls = sequence_output[:, 0, :].unsqueeze(1).unsqueeze(1)
601 cls = cls.repeat(1, e1_e2_start.shape[0], e1_e2_start.shape[1], 1)
602
603 buffer = []
604 for i in range(cls.shape[0]): # iterate batch & collect
605     cls_token = cls[i, i, :, :]
606     v1v2 = torch.cat((cls_token[0], cls_token[1]))
607     buffer.append(v1v2)
608 del cls
609 v1v2 = torch.stack([a for a in buffer], dim=0)
610 del buffer
```

3 个 epoch 结束后，训练结束后在训练集上的训练 accuracy 为 0.80，在验证集 eval 上 accuracy=0.80，f1=0.75，precision=0.77，recall=0.72；

```
Epoch: 3, 6400/ 8000 points] total loss, accuracy per batch: 0.633, 0.801
Epoch: 3, 7200/ 8000 points] total loss, accuracy per batch: 0.552, 0.821
Epoch: 3, 8000/ 8000 points] total loss, accuracy per batch: 0.590, 0.816
06/06/2024 08:41:21 PM [INFO]: Evaluating test samples...
06/06/2024 08:41:42 PM [INFO]: ***** Eval results *****
06/06/2024 08:41:42 PM [INFO]: accuracy = 0.7961333671399595
06/06/2024 08:41:42 PM [INFO]: f1 = 0.7462562396006656
06/06/2024 08:41:42 PM [INFO]: precision = 0.7712811693895099
06/06/2024 08:41:42 PM [INFO]: recall = 0.7228041901692184
Epoch finished, took 811.53 seconds.
Losses at Epoch 3: 0.6481456
Train accuracy at Epoch 3: 0.7967500
Test f1 at Epoch 3: 0.7462562
06/06/2024 08:41:42 PM [INFO]: Finished training!
06/06/2024 08:41:45 PM [INFO]: Loading tokenizer and model...
```