[NLP] 新手的第一个 NLP 项目：文本分类（2）

原创 我是老宅 花解语NLP 8月12日

收录于话题
#深度学习 990 #自然语言处理 259 #PyTorch 64 #NLP 新手的第一个项目 4

前文回顾

在前一篇文章新手的第一个 **NLP** 任务：文本分类（1）中，我们读取了数据、对数据进行了预处理和封装，一切准备就绪。

前文更正

在数据预处理步骤中，构建词汇表一行代码应该为

```
vocab = set(['<PAD>']) # 词汇表
```

现在数据已经准备就绪，可以构建模型了。

本文的模型参考了论文 《**Convolutional Neural Networks for Sentence Classification**》[1]，原文代码在此[2]。

论文里使用了两个词嵌入：随模型进行训练的词嵌入和 Google 预训练好的 Word2Vec 词嵌入。本文没有采用预训练的词嵌入。

构建模型

论文里使用了三个卷积核分别为 3、4、5 的二维卷积层，拼接后经过一个视野为 4 的池化层。最后经过一个全连接层输出。

注意：因为下面的损失函数自带 sigmoid 运算，所以这里的输出没有经过 sigmoid 计算。在进行 inference 的时候需要加上 sigmoid 运算将输出转换为 logits。

```
from torch import nn
```

```
from torch.nn import functional as F

class CNN(nn.Module):
    def __init__(self, vocab_size, embed_size, dropout, batch_size):
        super(CNN, self).__init__()
        self.batch_size = batch_size

        self.embedding = nn.Embedding(vocab_size, embed_size) # (BATCH_SIZE, SEQ_LEN, embed_size)

        self.conv1 = nn.Conv2d(1, 1, 3)
        self.conv2 = nn.Conv2d(1, 1, 4)
        self.conv3 = nn.Conv2d(1, 1, 5)

        self.dropout = nn.Dropout(dropout)

        self.fc = nn.Linear(2232, 1)

    def forward(self, x):
        x = self.embedding(x)
        x.unsqueeze_(1) # (BATCH_SIZE, 1, SEQ_LEN, embed_size)
        output1 = self.conv1(x)
        output1 = F.max_pool2d(F.relu(output1), 4)

        output2 = self.conv2(x)
        output2 = F.max_pool2d(F.relu(output2), 4)

        output3 = self.conv3(x)
        output3 = F.max_pool2d(F.relu(output3), 4)
        output = torch.cat([output1, output2, output3], axis=1)
        output = self.dropout(output)

        return self.fc(output.view(self.batch_size, -1))
```

注意：因为经过词嵌入的张量维度为 (BATCH_SIZE, SEQ_LEN, embed_size)，而 nn.Conv2d 的输入张量的维度要求为 (BATCH_SIZE, CHANNEL, WIDTH, HEIGHT)，所以我们需要使用 x.unsqueeze_(1) 为张量添加一个维度。

我们使用 Adam 为优化器， `nn.BCEWithLogitsLoss()` 为损失函数。

```
from torch import optim

optimizer = optim.Adam(model.parameters())
criterion = nn.BCEWithLogitsLoss()
```

注意：nn.BCEWithLogitsLoss() 是先进行了 **sigmoid** 运算后再求交叉熵的损失函数，无需额外的 **sigmoid** 运算。

然后我们再定义一个求准确率的函数：

```
def binary_accuracy(preds, y):
    rounded_preds = torch.round(torch.sigmoid(preds))
    correct = (rounded_preds == y).float()
    acc = correct.sum() / len(correct)
    return acc
```

紧接着我们开始定义训练和验证的函数：

```
# 训练函数
def train(model, iterator, optimizer, criterion):
    epoch_loss = 0
    epoch_acc = 0

    model.train() # 进入训练模式

    for text, label in iterator:
        optimizer.zero_grad()
        preds = model(text)
        loss = criterion(preds.squeeze(), label.float())
        acc = binary_accuracy(preds.squeeze(), label)
        loss.backward()
        optimizer.step()

        epoch_loss += loss.item()
        epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)

# 验证函数
def evaluate(model, iterator, criterion):
    epoch_loss = 0
    epoch_acc = 0

    model.eval() #  进入验证模式

    with torch.no_grad():
        for text, label in iterator:
            preds = model(text)
            loss = criterion(preds.squeeze(), label.float())
            acc = binary_accuracy(preds.squeeze(), label)
```

```
            epoch_loss += loss.item()
            epoch_acc += acc.item()

    return epoch_loss / len(iterator), epoch_acc / len(iterator)
```

可以看到，训练函数与验证函数大同小异，主要区别在于：

1. 训练模式下权重更新，验证模式下权重不更新；
2. 训练模式下 dropout 有效，验证模式下 dropout 无效；
3. 验证模式没有优化器。

注意：在计算损失函数时，真实标签也要转换成 float 格式，否则会报错。

下面就可以构建真正的训练、评估循环了：

```
import time

def epoch_time(start_time, end_time): # 计算每一轮花费的时间
    elapsed_time = end_time - start_time
    elapsed_mins = int(elapsed_time / 60)
    elapsed_secs = int(elapsed_time - elapsed_mins * 60)
    return elapsed_mins, elapsed_secs

N_EPOCHS = 10
best_test_loss = float('inf')

for epoch in range(N_EPOCHS):
    start_time = time.time()

    train_loss, train_acc = train(model, train_iter, optimizer, criterion)
    test_loss, test_acc = evaluate(model, test_iter, criterion)

    end_time = time.time()
    epoch_mins, epoch_secs = epoch_time(start_time, end_time)

    if test_loss < best_test_loss:
        best_test_loss = test_loss
        torch.save(model.state_dict(), 'model.pt')

    print(f'Epoch: {epoch+1:02} | Epoch Time: {epoch_mins}m {epoch_secs}s')
    print(f'\tTrain Loss: {train_loss:.3f} | Train Acc: {train_acc*100:.2f}%')
    print(f'\t Val. Loss: {test_loss:.3f} |  Val. Acc: {test_acc*100:.2f}%')
```

我们进行 10 轮训练，如果验证集的准确率大于最大准确率，则保存模型。最佳结果为：

```
Epoch: 10 | Epoch Time: 2m 48s
 Train Loss: 0.218 | Train Acc: 91.25%
  Val. Loss: 0.453 |  Val. Acc: 80.52%
```

希望在后面将模型改进后，模型的表现会更好。

可以在 https://github.com/vincent507cpu/nlp_project/blob/master/text%20classification/01%20CNN.ipynb 查看全部代码。

参考资料

[1] Convolutional Neural Networks for Sentence Classification: *https://arxiv.org/pdf/1408.5882.pdf*

[2] 论文代码: *https://github.com/dennybritz/cnn-text-classification-tf*

- END -

收录于话题 #NLP 新手的第一个项目 4个

上一篇 下一篇

阅读原文