[NLP] 新手的第一个 NLP 项目：文本分类（3）

原创 我是老宅 花解语NLP 8月19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

在前两篇文章新手的第一个 NLP 任务：文本分类（1）和新手的第一个 NLP 项目：文本分类（2）中，我们读取了数据、对数据进行了预处理和封装，并搭建了一个 CNN 模型。本文中，我们将 CNN 模型换为 RNN 模型。

数据的准备

同新手的第一个 NLP 任务：文本分类（1）一样，不再赘述。

基础 RNN 模型

有关 RNN 的知识可以参考我以前写的文章 PyTorch 折桂 11：CNN & RNN。

```
from torch import nn, optim
from torch.nn import functional as F

class RNN(nn.Module):
    def __init__(self, vocab_size, embed_dim, hidden_dim):
        super(RNN, self).__init__()

        self.embedding = nn.Embedding(vocab_size, embed_dim) # (BATCH_SIZE, SEQ_LEN, EMBED_DIM)
        self.rnn = nn.RNN(embed_dim, hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        x = self.embedding(x)

        output, hidden = self.rnn(x)
        # output: (BATCH_SIZE, SEQ_LENGTH, HIDDEN_DIM)
```

```
        # hidden: (1, BATCH_SIZE, HIDDEN_DIM)

        return self.fc(hidden.squeeze(0))
```

我们首先使用一层单向 RNN。RNN 网络生成两个张量：输出层与保存了历史信息的隐藏层。使用哪一个呢？这要具体问题具体分析。对于文本摘要类任务，一般使用保存了历史信息的隐藏层。

这里要注意隐藏层的维度：当 `batch_first=True` 时，隐藏层的维度为（`(num_layers * directions, BATCH_SIZE, HIDDEN_DIM)`）；当 `batch_first=False` 时，隐藏层的维度为（`(num_layers * directions, SEQ_LENGTH, HIDDEN_DIM)`）。

因为这是一个单词单向的 RNN，所以第 0 维为 1；在将隐藏层进行全连接处理以前，先去除无用的第 0 维。

实例化 RNN 网络：

```
EMBED_DIM = 128
HIDDEN_DIM = 256
rnn = RNN(len(vocab), EMBED_DIM, HIDDEN_DIM)
```

损失函数、优化器、训练过程与前文一致，不再赘述。训练 10 个 epoch 后的结果如下：

```
Epoch: 10 | Epoch Time: 1m 8s
 Train Loss: 0.590 | Train Acc: 68.58%
  Val. Loss: 0.682 |  Val. Acc: 61.32%
```

可以看到，模型过拟合了。下面我们改进一下这个 RNN 模型。

改进 RNN 模型

我们主要从以下两个方面进行改进：

1. 改进词嵌入；
2. 增加模型的复杂度（使用两层双向 LSTM）；
3. 增加正则化。

```
class LSTM(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, n_layers,
                 bidirectional, dropout):
        super(LSTM, self).__init__()
```

```
        self.embed = nn.Embedding(vocab_size, embedding_dim, padding_idx=0)

        self.lstm = nn.LSTM(embedding_dim,
                            hidden_dim,
                            num_layers=n_layers,
                            bidirectional=bidirectional,
                            dropout=dropout,
                            batch_first=True)

        self.dropout = nn.Dropout(dropout)
        self.num_directions = 2 if bidirectional else 1
        self.fc = nn.Linear(hidden_dim * self.num_directions, 1)

    def forward(self, x):
        embedded = self.dropout(self.embed(x)) # (BATCH_SIZE, SEQ_LEN, EMBED_DIM)

        output, (hidden, cell) = self.lstm(embedded)
        # output: (BATCH_SIZE, SEQ_LENGTH, HIDDEN_DIM)
        # hidden: (n_layers * num_directions, BATCH_SIZE, HIDDEN_DIM)
        # cell: (n_layers * num_directions, BATCH_SIZE, HIDDEN_DIM)

        hidden = self.dropout(torch.cat((hidden[-2, :, :], hidden[-1, :, :]), dim=1))
        # hidden: (BATCH_SIZE, HIDDEN_DIM * 2)
        return self.fc(hidden)
```

首先，填充 `<PAD>` 应该恒为 0，所以我们在词嵌入层中加入 `padding_idx=0` 条件。这里 `padding_idx` 为 0 是因为我们在数据准备过程中将填充占位设为 0。

其次，将 RNN 层变成 LSTM 层。LSTM 模型的输出有三个， `output, (hidden, cell)` ，隐藏层与细胞状态在一个元组内。当 `batch_first=True` 时，隐藏层与细胞状态的维度为（ `(num_layers * directions, BATCH_SIZE, HIDDEN_DIM)` ）；当 `batch_first=False` 时，隐藏层与细胞状态的维度为（ `(num_layers * directions, SEQ_LENGTH, HIDDEN_DIM)` ）。当方向为双向且层数多于 1 时，隐藏层与细胞状态的堆叠层次为：[第一层正向，第一层反向，...，最后一层正向，最后一层反向]。这里使用了两层双向 LSTM。我们需要最后一层的正向与反向隐藏层，并把它们拼接在一起。

最后，还加入了 dropout 正则化。LSTM 内部的 dropout 可以使用 `dropout` 声明，LSTM 与全连接层之间的 dropout 可以使用 `nn.Dropout` 层。

实例化这个 LSTM 模型。

```
EMBED_DIM = 128
HIDDEN_DIM = 256
N_LAYERS = 2
BIDIRECTIONAL = True
```

```
DROPOUT = 0.5

lstm = LSTM(len(vocab), EMBED_DIM, HIDDEN_DIM, N_LAYERS, BIDIRECTIONAL, DROPOUT)
```

损失函数、优化器、训练过程与前面相同。最终的训练效果为：

```
Epoch: 10 | Epoch Time: 7m 34s
 Train Loss: 0.303 | Train Acc: 87.30%
  Val. Loss: 0.412 |  Val. Acc: 83.43%
```

比前面的 CNN 效果稍好。下文中我们将使用 SOTA 的预训练模型 - BERT。

本文的代码可以在 https://github.com/vincent507cpu/nlp_project/blob/master/text%20classification/02%20RNN.ipynb 查看。

- 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