[NLP] 新手的第一个 NLP 项目：文本分类（4）

原创 我是老宅 花解语NLP 8月26日

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

在之前的文章中，我们使用了 CNN 和 RNN 对 IMDB 数据集进行了分析，10 个 epoch 以后准确率不到 85%。除了使用更复杂的模型以外，我们还可以使用更好的词向量。本文中我们将使用 Bert 词向量和 GRU 层搭建另一个简单的神经网络模型。由于 transformers 涉及到大量计算，本文中将使用 Google Colab 提供的 GPU。

与前面的数据预处理流程不同，这里我们将使用 `torchtext` 来封装数据。有关 `torchtext` 的知识请看 **PyTorch 折桂 13：TorchText**。

安装所需的包：

```
!pip install -U torch # 1.7
!pip install -U torchtext # 0.7
!pip install -U transformers # 3.0.2
```

设置随机种子：

```
import torch
import random
import numpy as np

SEED = 1988

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed(SEED)
torch.backends.cudnn.deterministic = True # 这样可以稍微增加训练的速度
```

数据准备

之前的文章中，我们仅仅使用了 `<PAD>` 来填充不足的空位；而在 Bert 里，除了 `<PAD>` 还使用了 `BOS` 和 `<EOS>` 来表示句子的开始和结束以及 `<UNK>` 来表示单词表以外的单词。另外

Bert 取每句话前 512 个单词。首先载入预训练好的 Bert 分词器并以此构建分词函数。

```
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

init_token_id = tokenizer.cls_token_id # BOS
eos_token_id = tokenizer.sep_token_id # EOS
pad_token_id = tokenizer.pad_token_id # PAD
unk_token_id = tokenizer.unk_token_id # UNK

max_length_input = tokenizer.max_model_input_sizes['bert-base-uncased']

def tokenize_and_cut(sentence):
    tokens = tokenizer.tokenize(sentence)
    tokens = tokens[:max_length_input - 2]
    return tokens
```

下一步是构建数据集的域。所谓“域”指的是数据集里对文本与标签的处理方式的声明。

```
from torchtext.data import Field, LabelField

TEXT  = Field(batch_first=True,
              use_vocab=False,
              tokenize=tokenize_and_cut,
              preprocessing=tokenizer.convert_tokens_to_ids,
              init_token=init_token_id,
              eos_token=eos_token_id,
              pad_token=pad_token_id,
              unk_token=unk_token_id)

LABEL = LabelField(dtype=torch.float)
```

最后就是读取与封装数据。 `batch size` 设置为 64。因为使用 GPU 训练，数据需要转移到 GPU 上。

```
from torchtext import datasets

train, test = datasets.IMDB.splits(TEXT, LABEL)
LABEL.build_vocab(train)
from torchtext.data import BucketIterator

BATCH_SIZE = 64
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

train_iter, test_iter = BucketIterator.splits(
    (train, test),
    batch_size=BATCH_SIZE,
    device=device
)
```

模型搭建

载入 Bert 预训练模型：

```
from transformers import BertTokenizer, BertModel

bert = BertModel.from_pretrained('bert-base-uncased')
```

根据 **Bert** 论文[1]，Bert base 模型的超参数有：transformers 层数为 12，隐藏层维度为 768，self-attention head 数量为 12。我们在实际模型中只需要隐藏层维度。现在我们搭建一个在 Bert 后面连接一个双层、双向 GRU 的模型。

```
from torch import nn

class BertGRU(nn.Module):
    def __init__(self, bert, hidden_dim, n_layers, bidirectional, dropout):
        super().__init__()

        self.bert = bert

        embed_dim = bert.config.to_dict()['hidden_size']

        self.gru = nn.GRU(embed_dim, hidden_dim, num_layers=n_layers, bidirectional=bidirectional
                          batch_first=True, dropout=0 if n_layers < 2 else dropout)

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

        self.dropout = nn.Dropout(dropout)

    def forward(self, text): # text: [BATCH_SIZE, SEQ_LENGTH]
        with torch.no_grad():
            embedded = self.bert(text)[0] # embedded: [BATCH_SIZE, SEQ_LENGTH, EMBED_DIM]

        _, hidden = self.gru(embedded) # hidden: [N_LAYERS * n_driections, BATCH_SIZE, EMBED_DIM]

        if self.gru.bidirectional:
```

```
            hidden = self.dropout(torch.cat((hidden[-2, :, :], hidden[-1, :, :]), dim=1))
        else:
            hidden = self.dropout(hidden[-1, :, :])

        output = self.fc(hidden) # hidden: [BATCH_SIZE, 1]

        return output
```

首先实例化这个模型。

```
HIDDEN_DIM = 768
N_LAYERS = 2
BIDIRECTIONAL = True
DROPOUT = 0.5

model = BertGRU(bert, HIDDEN_DIM, N_LAYERS, BIDIRECTIONAL, DROPOUT)
```

因为 Bert 是已经训练好的词向量，我们不希望它被训练，也不希望它的权重被更新，所以模型里有 `with torch.no_grad()` 代码块。另外我们也手动关闭 Bert 有关的权重更新：

```
for name, param in model.named_parameters():
    if name.startswith('bert'):
        param.requires_grad = False
```

优化器和损失函数和前面一样，使用 Adam 和二分类交叉熵。同样将优化器和损失函数转移到 GPU 上。

```
from torch import optim

optimizer = optim.Adam(model.parameters())

criterion = nn.BCEWithLogitsLoss()

model = model.to(device)
criterion = criterion.to(device)
```

后面的训练和预测同以前的文章一样，不再赘述。训练 10 个 epoch 后的表现为：

```
Epoch: 10 | Epoch Time: 38m 7s
 Train Loss: 0.094 | Train Acc: 96.62%
```

```
Val. Loss: 0.243 |  Val. Acc: 92.39%
```

有了 Bert 的加持，模型的性能提高了约 10%。

本文代码可以在 https://github.com/vincent507cpu/nlp_project/blob/master/text%20classification/03%20transformers.ipynb 查看。

参考资料

[1] Bert 论文: *https://arxiv.org/pdf/1810.04805.pdf*

收录于话题 #NLP 新手的第一个项目 4个

上一篇

阅读原文