pytorch Bert 文本分类

> SA22011050 吕瑞

环境准备

`requirements`:

```
python == 3.8.8
torch == 1.13.1+cu117
# pip3 install torch torchvision torchaudio
```

数据集准备

`prepare dataset`: 本次实验统一使用指定的 IMDB 公开数据集“Large Movie Review Dataset”。 该数据集分别包含 25,000 条电影评论作为训练集和测试集。

```
mkdir dataset
cd dataset
wget https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
tar -zxvf aclImdb_v1.tar.gz

# catalogue
dataset
    aclImdb
        test
        train
            neg
            pos
```

数据预处理

`prepare_data.py`

遍历文件夹读取数据：

```
def read_imdb(path='../dataset/aclImdb', is_train=True):
    ...
```

清洗数据集并分词，构建数据集：

```
def load_imdb(path="../dataset/aclImdb", data_path="../dataset"):
    print('load imdb dataset ...')
    ...
```

l 利用 `from torch.utils.data import Dataset` 构建 Dataset

```
class ImdbDataset(Dataset):
    def __init__(self, mode, path="../../dataset"):
        ...
    def __getitem__(self, index):
        ...
    def __len__(self):
        ...
    def clear(self, text):
        # replace the unuse tokens to " "
        ...
```

语言模型

`model.py`

搭建 BERT 模型并加载大语料库上预训练的模型参数（与实验二不同，无须 手动训练）。推荐的预训练参数来源为 huggingface

基于训练好的语言模型（固定参数），编写一个情感分类模型，包含一个 LSTM 模型和一个分类器 MLP。 首先，将一个句子中的每个单词对应的词向量输入 LSTM，得到句子的向量表征。然后将句向量作为分类器的输入，输出二元分类预测，同样进行 loss 计算和反向梯度传播训练，这里的 loss 是分类 loss，交叉熵 loss。 基于训练好的语言模型，编写一个情感分类模型，包含一个 BERT 模型和一个分类器（如 MLP）。 首先，将一个句子中的每个单词对应的词向量输入 BERT，得到句子的向量表征。然后将句向量作为分类器的输入，输出二元分类预测，同样进行 loss 计算和反向梯度传播训练，这里的 loss 是分类 loss，交叉熵 loss。

> 注意，此次实验中没有固定 BERT 模型的参数，而须随分类器一同训练，以达到微调的目的。

```
# Define model
class Bert(nn.Module):
    def __init__(self, hidden_size = 768, labels=2, max_length=256,
device='cuda'):
        super().__init__()
        # use simple bert
        print('prepare the pretrained model ... ')
        model_name = 'distilbert-base-uncased'
        # load tokenizer
        self.tokenizer =
BertTokenizer.from_pretrained(pretrained_model_name_or_path=model_name,
do_lower_case=True)
        # load pretrained model
        self.encoder =
BertModel.from_pretrained(pretrained_model_name_or_path=model_name)

        self.decoder = nn.Linear(hidden_size, labels)
        self.max_length=max_length
        self.device = device

        print('prepare model have done!')

    def forward(self, batch_sentences):
        """
        batch_sentences : [batch_size, seq]
        """
        sentence_tokenizer = self.tokenizer(batch_sentences, # sentence to
encode
                                            truncation=True, # cut off the
rest word if overlap max length
                                            padding = True,
                                            max_length = self.max_length,
                                            add_special_tokens = True
                                            )
        input_ids =
torch.tensor(sentence_tokenizer['input_ids']).to(self.device)
        attention_mask =
torch.tensor(sentence_tokenizer['attention_mask']).to(self.device)
        encoder_out = self.encoder(input_ids=input_ids,
attention_mask=attention_mask)
        """
        encoder_out <-> bert_out: tuple, includes
        last_hidden_state, pooler_output, hidden_states, attentions,
cross_attentions, past_key_values
        """
        last_hidden_state = encoder_out[1] # [batch_size,  hidden_size]
        """
        I choice the pooler_output (torch.FloatTensor of shape (batch_size,
hidden_size)) of each sentence as the representation.
        """

        output = self.decoder(last_hidden_state)

        return output
```

测试性能

```
# Set parameters
batch_size = 64 # batchsize: 16,64,128,256
length_list = [128,256] # sequence length
lr_list = [1e-5,1e-6]
epochs = 10
```

```
-------------------------------------------------------
length_list128_lr1e-05
Avg valid loss 0.460022, Accuracy: 83.4%
Training done!
Avg test loss 0.191584, Accuracy: 93.8%

-------------------------------------------------------
length_list128_lr1e-06
Avg valid loss 0.476827, Accuracy: 78.0%
Training done!
Avg test loss 0.432547, Accuracy: 80.3%

-------------------------------------------------------
length_list256_lr1e-05
Avg valid loss 0.371136, Accuracy: 85.3%
Training done!
Avg test loss 0.174399, Accuracy: 93.8%

-------------------------------------------------------
length_list256_lr1e-06
Avg valid loss 0.445700, Accuracy: 80.7%
Training done!
Avg test loss 0.401356, Accuracy: 82.6%
```

最优参数：

```
length_list = 128
lr_list = 1e-5
epochs = 10
Avg valid loss 0.460022, Accuracy: 83.4%
Avg test loss 0.191584, Accuracy: 93.8%
```

和 LSTM 实验结果对比

```
                  Accuracy | epoch | lr
word2vec + LSTM: 93.4%    |  100  | 1e-3
Bert           : 93.8%    |  10   | 1e-5
```

对比实验结果可知，Bert 预训练模型在使用较小学习率训练少量轮数即可达到与 LSTM 模型相当，甚至更优一些的效果。由此可见，预训练模型在文本分类任务上进行少量微调后的表现已经超过了传统方法下训练的模型，且使用更方便，更高效。