北京航空航天大學
BEIHANG UNIVERSITY

技术报告

《机器学习导论》

课程教师	黄迪、张永飞
队伍成员	19182604 宋宇奇
	19377180 王昊
	19182655 李霄龙

2021 年 12 月 26 日

1 问题重述

本次我们团队选取的题目为文本情感识别，既对于社交平台的评论回复进行情歌分析，判断其应属于的情歌类别。本次任务共有 36206 条训练数据以及 9052 条测试数据，包含有三种不同的情歌标签，分别为：negative，positive，neutral。训练集共有 num，text，labels，id 四个字段。最终任务的得分评价标准为 Micro-Fscore。

2 数据预处理

在 nlp 领域中，最常见的数据预处理方式就是 one-hot 编码也就是独热编码，我们对于本次任务同样采用了独热编码的数据预处理方式，既先对每一条文本进行分词统计，根据词频进行特征词筛选，然后对每一条文本进行分词，若对应的词在特征词中，则对应位置设为 1，不在则设为 0。由于在 nlp 领域中常用迁移学习，本次团队作业也使用了 BERT 预训练模型的迁移学习方式进行模型预测，使用的预训练模型为 bert-base-uncase，也尝试了很多其他的预训练模型，例如：cardiffnlp/twitter-roberta-base-sentiment（roberta 等系列的各种模型）。

3 模型介绍

BERT 模型介绍：BERT 模型由谷歌在 2018 年发布，发布后在当时的机器阅读理解顶级水平测试 `SQuAD1.1` 中表现出惊人的成绩：全部两个衡量指标上全面超越人类，并且还在 11 种不同`NLP`测试中创出最佳成绩。BERT 通过 **Multi-head self-attention** 有更好的上下文依赖。在训练过程中把少量词替换成 Mask 或另外一个词，以增强模型对上下文的记忆。除此之外，他参数还很多！譬如 BERT BASE 模型: `L=12, H=768, A=12`, 需要训练的模型参数总数是 `12 * 768 * 12 = 110M` 。

Bert 是通过 fill Mask 这种自监督模式来训练如此大量的参数（就是遮蔽一部分词让模型预测这些词）。BERT 是基于 Transformers 框架的，而 Transformers 模型主要包括两大部分：Encoder、Decoder。Encode 可以把自然语言翻译成高维矩阵的形式，Decoder 则可以把高维矩阵翻译回自然语言，这也和传统的 `Seq2Seq` 模型是一致的。Bert 由于是为了预训练服务，所以只有 Encoder 部分。Encoder 的编码出的高维矩阵有了自然语言的语义信息。在实践中就可以把用这个“句向量”作为下游模型的输入。

Transformers 我们就无需介绍了，Google 2019 年的神作，这里不细讲。

Bert 的 Embedding 由三种 Embedding 求和而成，分别是 Token Embeddings，Segment Embeddings，Position Embeddings。token embedding 层是要将各个词转

换成固定维度的向量。在 BERT 中，每个词会被转换成 768 维的向量表示；Segment Embeddings 用来区别两种句子，前一个句子的每个 token 都用 0 表示，后一个句子的每个 token 都用 1 表示；Position Embeddings：BERT 能够处理最长 512 个 token 的输入序列。BERT 在各个位置上学习一个向量表示来讲序列顺序的信息编码进来。这意味着 Position Embeddings layer 实际上就是一个大小为 (512, 768) 的 lookup 表，表的第一行是代表第一个序列的第一个位置，第二行代表序列的第二个位置，以此类推。

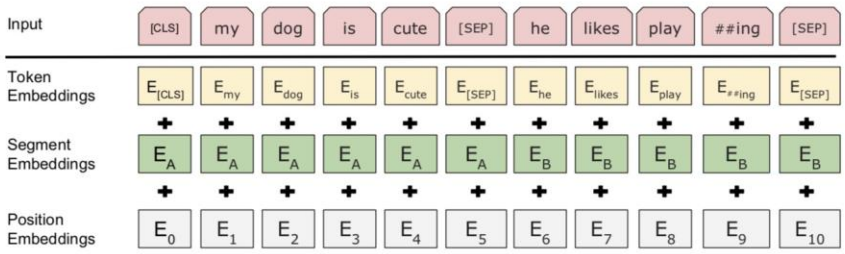

Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

最后，长度为 n 的输入序列将获得的三种不同的向量表示，分别是：

Token Embeddings， (1, n, 768) ，词的向量表示。

Segment Embeddings， (1, n, 768)，辅助 BERT 区别句子对中的两个句子的向量表示。

Position Embeddings，(1, n, 768) ，让 BERT 学习到输入的顺序属性。

这些表示会被按元素相加，得到一个大小为(1, n, 768)的合成表示。这一表示就是 BERT 编码层的输入了。

对于不同的下游任务，我们仅需要对 BERT 不同位置的输出进行处理即可，或者直接将 BERT 不同位置的输出直接输入到下游模型当中。对于情感分析等单句分类任务，可以直接输入单个句子（不需要[SEP]分隔双句），将[CLS]的输出直接输入到分类器进行分类。

本次团队作业我们使用的预训练模型为 huggingface 开源的 bert-base-uncase 以及 cardiffnlp/twitter-roberta-base-sentiment。

4 训练过程

1.首先设置 max_len 为 128，利用 gpu 使用 cuda 加速：

```
max_len = 128
device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
```

2. 加载数据

```
加载数据
```

```
train = pd.read_csv('DataSet/Train.csv', index_col=False)

data = list()
for line in tqdm(train.iterrows()):
    data.append(line)
train.columns = ['num', 'text', 'emotions', 'id']
train['positive'] = str(0)
train['negative'] = str(0)
train['neutral'] = str(0)
```

```
test = pd.read_csv('DataSet/Test.csv', index_col=False)
train = train[1:-1]
```

3. 数据处理

```
for i in range(1, 36205):
    if train.loc[i, 'emotions'] == "positive":
        train.loc[i, 'positive'] = 1
    elif train.loc[i, 'emotions'] == "negative":
        train.loc[i, 'negative'] = 1
    else:
        train.loc[i, 'neutral'] = 1
    i = i + 1

test.columns = ['num', 'text', 'id']
test['positive'] = str(0)
test['negative'] = str(0)
test['neutral'] = str(0)
train.to_csv('DataSet/train_n.csv',
             columns=['num', 'text', 'emotions', 'id', 'positive',
'negative', 'neutral'],
             sep='\t',
             index=False)
test.to_csv('DataSet/test_n.csv',
            columns=['num', 'text', 'id', 'positive', 'negative',
'neutral'],
            sep='\t',
            index=False)
```

4. 定义 dataset

```
target_cols = ['positive', 'negative', 'neutral']
class RoleDataset(Dataset):
    def __init__(self, tokenizer, max_len, mode='train'):
        super(RoleDataset, self).__init__()
        if mode == 'train':
```

```
            self.data = pd.read_csv('DataSet/train_n.csv',
sep='\t')
        else:
            self.data = pd.read_csv('DataSet/test_n.csv', sep='\t')
        self.texts = self.data['text'].tolist()
        self.labels = self.data[target_cols].to_dict('records')
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __getitem__(self, index):
        text = str(self.texts[index])
        label = self.labels[index]
        encoding = self.tokenizer.encode_plus(text,
add_special_tokens=True, max_length=self.max_len,
return_token_type_ids=True, pad_to_max_length=True,
return_attention_mask=True, return_tensors='pt', )
        sample = {
            'texts': text,
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten()
        }
        for label_col in target_cols:
            sample[label_col] = torch.tensor(label[label_col] / 3.0,
dtype=torch.float)
        return sample
    def __len__(self):
        return len(self.texts)
```

5. create dataloader

```
def create_dataloader(dataset, batch_size, mode='train'):
    shuffle = True if mode == 'train' else False
    if mode == 'train':
        data_loader = DataLoader(dataset, batch_size=batch_size,
shuffle=shuffle)
    else:
        data_loader = DataLoader(dataset, batch_size=batch_size,
shuffle=shuffle)
    return data_loader
```

6. 加载预训练模型：

```
PRE_TRAINED_MODEL_NAME =
'cardiffnlp/twitter-roberta-base-sentiment'
tokenizer =
RobertaTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

```
base_model =
RobertaModel.from_pretrained(PRE_TRAINED_MODEL_NAME)  # 加载预训练
模型
```

7. 模型构建

```
def init_params(module_lst):
    for module in module_lst:
        for param in module.parameters():
            if param.dim() > 1:
                torch.nn.init.xavier_uniform_(param)
    return


class MLModelLite(nn.Module):
    def __init__(self, n_classes, model_name):
        super(MLModelLite, self).__init__()
        config = AutoConfig.from_pretrained(model_name)
        config.update({"output_hidden_states": True,
                       "hidden_dropout_prob": 0.0,
                       "layer_norm_eps": 1e-7})
        self.base = BertModel.from_pretrained(model_name,
config=config)
        dim = 1024 if 'large' in model_name else 768
        self.attention = nn.Sequential(
            nn.Linear(dim, 512),
            nn.Tanh(),
            nn.Linear(512, 1),
            nn.Softmax(dim=1)
        )
        # self.attention = AttentionHead(h_size=dim, hidden_dim=512,
w_drop=0.0, v_drop=0.0)
        self.out_pos = nn.Sequential(
            nn.Linear(dim, n_classes)
        )
        self.out_neg = nn.Sequential(
            nn.Linear(dim, n_classes)
        )
        self.out_neu = nn.Sequential(
            nn.Linear(dim, n_classes)
        )
        init_params([self.out_pos, self.out_neg, self.out_neu])
    def forward(self, input_ids, attention_mask):
        roberta_output = self.base(input_ids=input_ids,
                                   attention_mask=attention_mask)
```

```
        last_layer_hidden_states =
roberta_output.hidden_states[-1]
        weights = self.attention(last_layer_hidden_states)
        # print(weights.size())
        context_vector = torch.sum(weights *
last_layer_hidden_states, dim=1)
        # context_vector = weights
        pos = self.out_pos(context_vector)
        neg = self.out_neg(context_vector)
        neu = self.out_neu(context_vector)
        return {
            'positive': pos, 'negative': neg, 'neutral': neu
        }
```

8. 参数配置

```
EPOCHS = 2
weight_decay = 0.0
data_path = 'data'
warmup_proportion = 0.0
batch_size = 4
lr = 1e-5

warm_up_ratio = 0.000
```

9. 模型训练

```
def do_train(model, criterion, optimizer, scheduler):
    model.train()
    global_step = 0
    tic_train = time.time()
    log_steps = 100
    for epoch in range(EPOCHS):
        losses = []
        for step, sample in enumerate(train_loader):
            input_ids = sample["input_ids"].to(device)
            attention_mask = sample["attention_mask"].to(device)
            outputs = model(input_ids=input_ids,
attention_mask=attention_mask)
            loss_pos = criterion(outputs['positive'],
sample['positive'].view(-1, 1).to(device))
            loss_neg = criterion(outputs['negative'],
sample['negative'].view(-1, 1).to(device))
            loss_neu = criterion(outputs['neutral'],
sample['neutral'].view(-1, 1).to(device))
            loss = loss_pos + loss_neg + loss_neu
```

```
            losses.append(loss.item())
            loss.backward()
            optimizer.step()
            scheduler.step()
            optimizer.zero_grad()
            global_step += 1
            if global_step % log_steps == 0:
                print("global step %d, epoch: %d, batch: %d,
loss: %.5f, speed: %.2f step/s, lr: %.10f"
                      % (global_step, epoch, step, np.mean(losses),
global_step / (time.time() - tic_train),
                         float(scheduler.get_last_lr()[0])))

do_train(model, criterion, optimizer, scheduler)
```

5 结果展示

```
PS D:\Beihang_Courses\2021_Autumn\Class\机器学习导论 张永飞 黄迪\BigWork> python .\client.py
正在提交...
{'result': 0.7823685373398144, 'id': 0, 'jsonrpc': '2.0'}
测试完成，请查看分数
0.7823685373398144
PS D:\Beihang_Courses\2021_Autumn\Class\机器学习导论 张永飞 黄迪\BigWork> python .\leaderboard.py
请查看结果
sid             |name            |score                  |rank
19182638        |刘昀鑫                 |0.7895492708793638     |0
19182605        |张瑞轩                 |0.7841361025187804     |1
19182655        |李霄龙                 |0.7823685373398144     |2
19373587        |李雨航                 |0.7729783473265576     |3
19373562        |仲书璋                 |0.7729783473265576     |4
19373026        |胡官承                 |0.7243703049049933     |5
18374162        |涂尚卿                 |0.7200618647812638     |6
18374204        |徐婧予                 |0.7190676093680954     |7
19182647        |金硕            |0.6679186920017676     |8
19373228        |罗人杰                 |0.6660406539991163     |9
19182608        |王宇            |0.6477021652673443     |10
19373027        |李雨东                 |0.6288113124171454     |11
19373733        |马介平                 |0.5546840477242598     |12
19373750        |杨宜凡                 |0.5465090587715422     |13
19373449        |马柯杰                 |0.5465090587715422     |14
19373765        |顾檬            |0.4191338930623067     |15
19373056        |王海龙                 |0.4191338930623067     |16
19182607        |张浩勋                 |0.3555015466195315     |17
19373630        |曾育群                 |0.3509721608484313     |18
```

其余的提交材料随压缩包一起放入。

6 思考与模型改进

我们在预训练模型是 bert-base-uncase，batchsize = 4，max_len = 128 的时候就可以达到 77.9，这样做是因为个人的笔记本电脑的显存实在是不够，又知道不论是 BERT 的模型训练还是迁移学习，都是非常占用显存的。所以我们换了一个支持 RTX3090 显卡的朋友的台式

机，用他们的机器跑了 cardiffnlp/twitter-roberta-base-sentiment 预训练模型，且 batchsize = 16，max_len = 256，这个时候的得分反而下降了，发现其实是预训练模型选的并不好。接下来又重新换回 bert-base-uncase，batchsize = 16，max_len = 256 的配置，结果为 0.782，效果更好了。

在改进方面，我觉得可以有以下几个点进行思考：模型投票（多模型预测结果后平均），采用模型对抗训练，删除 attention 层等等，可以预想到由于语句之间都是随机的，所以这些操作或多或少可以提升一些模型的鲁棒性。

另外还有一个提升的方面还可以有，比如说对于训练数据的增广数据集处理，对抗数据集处理等等，可以增加语言的丰富性，还可以借用一些 CV 的想法加入到下游任务中，虽然我们只是听说过这样的方式，没有自己实践过，但如果有时间的话可以试试看。

我们曾经参加过数据挖掘 CCF-BDCI 的比赛，在后期 BERT 的微调中，不同的参数对于对应的结果来说很多都是效果微乎其微，几乎看不到明显的变化。由于时间紧迫，也没有太多的时间参与到调参来，不过总体排行排在第 3 名，也是可以满足的一个成绩了。