

山东大学 计算机科学与技术 学院

大数据分析与实践 课程实验报告

学号：202300130005	姓名：于佳杭	班级：23 数据
实验题目：BERT (2)		
实验学时：2	实验日期：2025. 11. 4	
实验步骤与内容：		
1.		
负责数据的加载与预处理，它从指定的 TSV 文件中读取句子对和标签，利用 BERT 分词器将文本转换为模型可处理的输入 ID、注意力掩码等张量格式，并确保数据长度统一以便批量处理。		
<pre>def __init__(self, file_path, tokenizer=None, max_length=128): local_model_path = "../bert-base-uncased" self.tokenizer = tokenizer if tokenizer else BertTokenizer.from_pretrained(local_model_path) self.max_length = max_length self.data = [] with open(file_path, 'r', encoding='utf-8-sig') as f: next(f) for line in f: parts = line.strip().split('\t') if len(parts) < 5: continue label = int(parts[0]) sentence1 = parts[3] sentence2 = parts[4] self.data.append((sentence1, sentence2, label)) def __len__(self): return len(self.data) def __getitem__(self, idx): sentence1, sentence2, label = self.data[idx] assert isinstance(sentence1, str), f"Expected string but got {type(sentence1)}" assert isinstance(sentence2, str), f"Expected string but got {type(sentence2)}" encoding = self.tokenizer(sentence1, sentence2, truncation=True, padding='max_length', max_length=self.max_length, return_tensors='pt')</pre>		
2. FCModel 分类模型:简单的二元分类器，它接收 BERT 模型输出的 768 维[CLS]向量，通过两层全连接网络和非线性激活函数，最终输出一个 0 到 1 之间的概率值，用于表示两个句子语义相似的置信度。		

```
class FCModel(nn.Module): 2 用法
    def __init__(self):
        super(FCModel, self).__init__()
        self.fc1 = nn.Linear(768, 256)
        self.fc2 = nn.Linear(256, 1)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.sigmoid(x)
        return x
```

3. `Binary_accuracy` 准确率计算函数:用于评估模型性能,它将模型输出的概率值通过阈值(默认为 0.5)转换为二元预测结果,然后与真实标签进行比较,计算出当前批次的分类准确率。

```
def binary_accuracy(predictions, labels, threshold=0.5): 1 个用法
    binary_predictions = (predictions > threshold).float()
    correct = (binary_predictions == labels).float()
    accuracy = correct.sum() / len(correct)
    return accuracy
```

4. `train_model` 训练函数:管理完整的模型训练流程,它组织训练循环,在每个批次中执行前向传播计算损失和准确率,然后通过反向传播同时更新 BERT 模型和分类器的参数,并实时输出训练过程中的损失和准确率变化情况。

```
def train_model(bert_model, model, train_loader, crit, optimizer, bert_optimizer, device, num_epochs=3): 1 个用法
    bert_model.train()
    model.train()

    for epoch in range(num_epochs):
        epoch_loss, epoch_acc = 0., 0.
        total_len = 0
        progress_bar = tqdm(train_loader, desc=f'Epoch {epoch + 1}/{num_epochs}')

        for i, data in enumerate(progress_bar):
            if torch.cuda.is_available():
                print(f"当前显存使用情况: {torch.cuda.memory_allocated(device)}")

            input_ids = data['input_ids'].to(device)
            attention_mask = data['attention_mask'].to(device)
            label = data['label'].to(device)
            label = torch.clamp(label, 0, 1)

            encoding = {
                'input_ids': input_ids,
                'attention_mask': attention_mask
            }

            bert_output = bert_model(**encoding)
            pooler_output = bert_output.pooler_output
            predict = model(pooler_output).squeeze()

            loss = crit(predict, label)
            acc = binary_accuracy(predict, label)
```

结果:

```
Batch 245, Loss: 0.7275, Accuracy: 0.5196
当前显存使用情况: 1804331158
Batch 246, Loss: 0.611, Accuracy: 0.676
当前显存使用情况: 1803865814
Batch 247, Loss: 0.7785, Accuracy: 0.6172
当前显存使用情况: 1803704738
Batch 248, Loss: 0.6274, Accuracy: 0.611
当前显存使用情况: 1805907103
Batch 249, Loss: 0.6151, Accuracy: 0.6338
当前显存使用情况: 1803460001
Batch 250, Loss: 0.6212, Accuracy: 0.6324
当前显存使用情况: 1804909407
Batch 251, Loss: 0.7895, Accuracy: 0.5928
当前显存使用情况: 1802968530
Batch 252, Loss: 0.8429, Accuracy: 0.5124
当前显存使用情况: 1803125057
Batch 253, Loss: 0.6279, Accuracy: 0.5273
```

结论分析与体会:

本次实验通过构建 BERT 句子对分类任务,使我深入理解了基于 Transformer 的预训练模型在实际 NLP 任务中的应用流程。我认识到,从数据预处理、模型构建到训练优化的每一步都至关重要,尤其是 BERT 模型能够有效提取文本的深层语义特征,为下游分类任务提供强大的支持。通过亲自动手实现,我不仅掌握了如何利用现代深度学习框架搭建和训练一个完整的分类模型,更体会到了模型架构与数据预处理之间紧密配合的重要性。此次实践为今后处理更复杂的语义理解任务奠定了坚实的基础。