华东师范大学数据科学与工程学院上机实践报告

课程名称：当代人工智能　　**年级**：2019级　　**上机实践成绩**：
指导教师：李翔　　**姓名**：郭腾
上机实践名称：多模态融合模型　　**学号**：10195501402　　**上机实践日期**：

一. 实验目的

1. 训练一个多模态融合模型，用于在文本和图像的数据集上实现情感分类
2. 将本次实验的代码保存在Github仓库
   （https://github.com/GuoTeng-ECNU/AI-Final-HomeWork）

二. 实验环境

1. 训练环境：Google Colab
2. 本地环境：Python 3.10.5（Windows 10）

三. 实验过程

1. 在本地上对文本数据进行处理，保存为jsonl格式（save_text_jsonl.ipynb）
   在读取训练文本数据时，每一个文本都有1/5的概率划分为验证集

```
if random.randint(0,9) < 8:
    train_index.append(strs[0])
    train_label.append(strs[1].strip())
else:
    valid_index.append(strs[0])
    valid_label.append(strs[1].strip())
```

同时将标签转为0，1，2中的一个数字，每一种标签对应一个数字

```
label_to_index = {'negative':0, 'neutral':1, 'positive':2}
index_to_label = {0:'negative', 1:'neutral', 2:'positive'}
```

在保存到jsonl前，还需对文本进行tokenize

```
def tokenizer(str):
    words = word_tokenize(str)
    wordsFiltered = [word for word in words if word not in punctuations and word not in stopWords]
    return wordsFiltered
```

保存到jsonl前，需保存成dict结构

```
if need_label:
    assert len(text) == len(label)
    for i in range(0, len(text)):
        dict_list.append({'id': id[i], 'text': tokenizer(text[i]), 'label': label[i]})
else:
    for i in range(0, len(text)):
        dict_list.append({'id': id[i], 'text': tokenizer(text[i])})
```

需注意，测试集无需label，训练验证集需要label

```
train_dict_list = [item for item in train_dict_list if len(item['text']) > 0]
valid_dict_list = [item for item in valid_dict_list if len(item['text']) > 0]
test_dict_list = [item for item in test_dict_list if len(item['text']) > 0]
```

同时注意删除文本数据为空的数据，发现只有在划分出的训练集里有空的文本

```
def save_to_jsonl(dict_list, filename):
    with jsonlines.open('jsonl/'+filename, mode='w') as writer:
        for line in dict_list:
            writer.write(line)

save_to_jsonl(train_dict_list, 'train.jsonl')
save_to_jsonl(valid_dict_list, 'valid.jsonl')
save_to_jsonl(test_dict_list, 'test.jsonl')
```

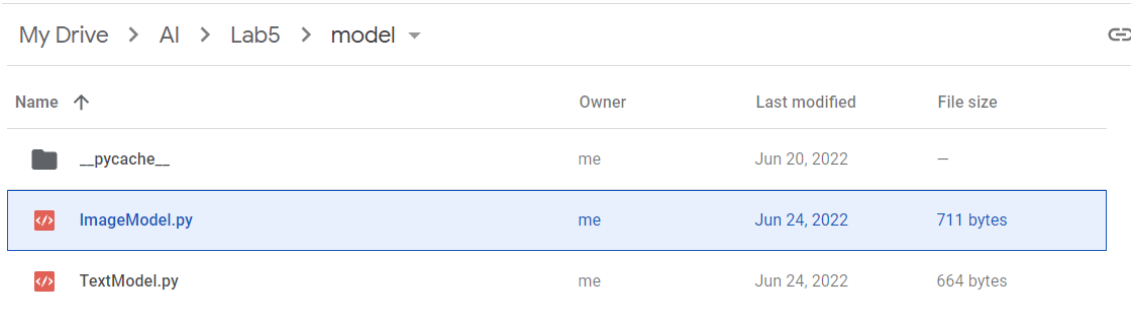
将文本写入jsonline文件

```
print(max(train_dict_len))
print(max(valid_dict_len))
print(max(test_dict_len))
```

```
29
29
29
```

发现文本序列最大的长度为29

2. 分别实现网络模型，对文本数据和图像数据实现分类



3. 对文本数据进行预处理

对于文本数据，已经保存为jisnline，使用torchtext读取jsonline格式的数据

```
fields = {'id': ('id',ID), 'text': ('text',TEXT),'label': ('label',LABEL)}
train_data, valid_data = TabularDataset.splits(
    path='jsonl',
    train='train.jsonl',
    validation='valid.jsonl',
    format='json',
    fields=fields)

test_fields = {'id': ('id',ID), 'text': ('text',TEXT)}
test_data = TabularDataset(
    path='jsonl/test.jsonl',
    format='json',
    fields=test_fields
)
```

用训练文本的数据构建词典

```
from torchtext.vocab import Vectors
from torch.utils.data import DataLoader
TEXT.build_vocab(train_data,
                 max_size=50000,
                 vectors='glove.6B.300d',
                 unk_init=torch.Tensor.normal_
)
```

将文本数据用Iterator保存，batch size为16

```
train_iterator = Iterator(
                  train_data,
                  batch_size=16,
                  train=True,
                  device=device)
valid_iterator = Iterator(
                  valid_data,
                  batch_size=16,
                  train=False,
                  sort=False,
                  device=device)
test_iterator = Iterator(
                  test_data,
                  batch_size=16,
                  train=False,
                  sort=False,
                  device=device)
```

4. 对于图像数据，进行预处理

```
img = cv.resize(img, (48, 48))
img = img / 255
img = np.transpose(img, [2, 0, 1])
img = torch.tensor(img, dtype=torch.float32)
id = self.all_image_id[index]
id = torch.tensor(id)
if self.is_test:
    return img, id
label = self.all_image_labels[index]
label = torch.tensor(label)
return img, label, id
```

发现最小的图片是48x48像素，使用resize将所有图片数据保存至这一大小
同时需注意测试图片数据是不需label的

```
train_image_iterator = DataLoader(train_images, batch_size=16, shuffle=True)
valid_image_iterator = DataLoader(valid_images, batch_size=16, shuffle=False)
test_image_iterator = DataLoader(test_images, batch_size=16, shuffle=False)
```

同时设置batch size为16

5. 对两个模型进行训练，均使用交叉熵和SGD方法

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(textmodel.parameters(), lr=0.001, momentum=0.5)
num_epochs = 2

for epoch in range(num_epochs):
    running_loss = 0.0
    for batch in train_iterator:
        # RuntimeError: expected scalar type Float but found Long
        labels = batch.label.to(device=device, dtype=torch.long)
        # ValueError: Expected input batch_size (30) to match target batch_size (32).
        inputs = batch.text.t().to(device=device, dtype=torch.float)
        optimizer.zero_grad()
        outputs = textmodel(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    print('epoch {0}/{1}, training loss: {2}'.format(epoch+1, num_epochs, running_loss/len(train_iterator)))
```

```
device = 'cuda'
imagemodel = ImageModel().to(device=device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(imagemodel.parameters(), lr=0.0001, momentum=0.5)

for epoch in range(num_epochs):
    running_loss = 0.0
    for i, data in enumerate(train_image_iterator, 0):
        inputs, labels, _ = data
        inputs = torch.FloatTensor(inputs).to(device=device)
        labels = torch.LongTensor(labels).to(device=device)

        optimizer.zero_grad()

        outputs = imagemodel(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
```

6. 使用Late Fusion，将两个训练后的模型对验证集同一个ID下的数据的输出进行叠加，得到最后的结果

```
with torch.no_grad():
    for i in range(0, len(valid_texts)):
        text_inputs = valid_texts[i].text.t().to(device=device, dtype=torch.float)
        text_targets = valid_texts[i].label.to(device=device, dtype=torch.long)
        text_outputs = textmodel(text_inputs)
        image_inputs, _, _ = valid_images[i]
        image_inputs = torch.FloatTensor(image_inputs).to(device=device)
        image_outputs = imagemodel(image_inputs)

        outputs = 1 * text_outputs + 1 * image_outputs
        _, predicted = torch.max(outputs.data, 1)
        print(predicted)
        total += text_targets.size(0)
        correct += (predicted == text_targets).sum().item()
```

四. 实验结果

1. 实验bug:

   BUG1: RuntimeError: Input type (torch.cuda.FloatTensor) and weight type (torch.FloatTensor) should be the same

   解决方法：这里也需要将模型也转为cuda，需要同时保证输入数据和模型都是Cpu或者Cuda

   BUG2: RuntimeError: expected scalar type Float but found Long

   解决方法：在输入前将tensor强制转为long或float

   BUG3：ValueError: Expected input batch_size (30) to match target batch_size (32).

   解决方法：原来batchsize为32时出现的，这里将tensor转置即可

2. 设计这个模型的原因：

   这里是为了将文本和图像用不同的分类器分类，使用Late fusion将二者的预测评分融合

   这样的好处是融合后模型的错误来自不同的分类器，而来自不同分类器的错误往往互

不相关、互不影响，不会造成错误的进一步累加，这样保证了融合模型的下限，同时可以通过优化不同的分类器来提高融合模型的效果

还有一点就是，文本和图像的数据特征本身就有很大的差别，使用late fusion可以对不同的数据类型进行更符合数据特征的分类处理

3. 经多次训练，融合模型在验证集上的准确率在55%上下浮动，以下只是最后一次训练结果，为0.54：

```
tensor([2, 2, 2, 2, 2, 2, 2, 0, 2, 2], device='cuda:0')
Late Fusion, Accuracy in valid dataset: 0.5390428211586902
```

4. 最后一次训练的模型消隆实验模型的结果：

只输入文本数据，准确率0.56

```
Accuracy in valid dataset: 0.5579345088161209
```

只输入图像数据，准确率0.45

```
Accuracy in valid dataset: 0.454659949622166 27
```

可以发现，使用late fusion可以减少一个分类器的错误汇入成融合模型的错误