第3章　文本分类

通过前面内容的实践，我们学会了如何使用 PaddlePaddle 实现一个简单的深度学习项目。本章将结合之前的内容，带领读者实现文本分类实战。文本分类是为给定的输入文本选择正确的类别标签的任务。在基本的分类任务中，每个输入被认为是与所有其他输入隔离的，并且标签集是预先定义的。这里是常见的一些分类任务：

- 判断一封电子邮件是否是垃圾邮件。
- 判断一条文本内容是否是谣言文本。
- 判断一篇新闻报道所属主题是什么，如“体育”“政治”“娱乐”等。

如果需要分类的信息中训练语料包含每个输入的正确标签被称为有监督分类，与之相反的是无监督分类。无监督分类常见的是机器学习方法，如聚类算法。在本章，我们着眼于有监督分类，用神经网络学会如何解决各种各样的分类任务。

实践十：基于 FNN 网络的电影评论情感分析

传统的文本分类模型一般根据文本的内容人工地构造特征，而人工构建特征存在考虑片面、浪费人力等现象。本实践使用基于前馈神经网络(FNN)的电影评论情感分析模型，将电影评论文本中的情感极性向量化，通过前馈神经网络的学习训练来挖掘表示文本深层的特征，避免了特征构建的问题，并能发现那些不容易被人发现的特征，从而产生更好的效果。本实践代码运行的环境配置如下：Python 版本为 3.7，PaddlePaddle 版本为 2.0.0，操作平台为 AI Studio。大部分深度学习项目都要经过以下几个过程：数据准备、模型配置、模型训练、模型评估。那下面就开始我们正式的项目实战。

步骤 1：IMDB 数据准备

电影评论情感分析是一个传统的二分类问题，创建一个分类器的第一步是决定输入的是什么样的文本数据，以及如何为这些数据进行编码。本次实践所使用的数据是 IMDB 数据集，其是一个对电影评论标注为正向评论与负向评论的数据集，共有 25 000 条文本数据作为训练集，25 000 条文本数据作为测试集。该数据集的官方地址为 http://ai.stanford.edu/~amaas/data/sentiment/。由于 IMDB 是 NLP 领域中常见的数据集，飞桨框架将其内置，路径为 paddle.text.datasets.Imdb。通过 mode 参数可以控制训练集与测试集：

```
train_dataset = paddle.text.datasets.Imdb(mode='train')
test_dataset = paddle.text.datasets.Imdb(mode='test')
```

paddle.text 目录是飞桨在文本领域的高层 API。

paddle.text.datasets.Imdb 的参数-mode(str)可设置为'train'或'test'模式。默认为'train'。-cutoff(int)为构建词典的截止大小。默认为 Default 150。

构建了训练集与测试集后,可以通过 word_idx 获取数据集的词表。在飞桨框架 2.0 版本中,推荐使用 padding 的方式来对同一个 batch 中长度不一的数据进行补齐,所以在字典中,我们还会添加一个特殊的词,用来在后续对 batch 中较短的句子进行填充。

```
word_dict = train_dataset.word_idx
word_dict['<pad>'] = len(word_dict)
```

如下所示,我们可以打印词典中前十个单词,词表大小为 5148。

```
the:0
and:1
a:2
of:3
to:4
is:5
in:6
it:7
i:8
this:9
```

为了实现词 ID 和词之间的转换,我们需要基于词典构建一个 ids_to_str()函数,实现布尔类型和字符串之间的转换,方便我们对数据的分析。

```
def ids_to_str(ids):
words = []
for k in ids:
    w = list(word_dict)[k]
    words.append(w if isinstance(w, str) else w.decode('ASCII'))
return " ".join(words)
```

电影评论文本加载完成后需要定义一个对应类别的词典,classes = ['negative', 'positive']。在这里,取出一条数据打印出来看看,可以用 docs 获取数据的 list,用 labels 获取数据的 label 值,打印出来对数据有一个初步的印象:

```
sent = train_dataset.docs[0]
label = train_dataset.labels[1]
print('sentence list id is:', sent)
print('sentence label id is:', label)
print('----------------------------')
print('sentence list is: ', ids_to_str(sent))
print('sentence label is: ', classes[label])
```

```
sentence list id is: [5146, 43, 71, 6, 1092, 14, 0, 878, 130, 151, 5146, 18, 281, 747, 0, 5146, 3, 5146, 2165, 37, 5146, 46, 5, 71, 4089,
377, 162, 46, 5, 32, 1287, 300, 35, 203, 2136, 565, 14, 2, 253, 26, 146, 61, 372, 1, 615, 5146, 5, 30, 0, 50, 3290, 6, 2148, 14, 0, 5146,
11, 17, 451, 24, 4, 127, 10, 0, 878, 130, 43, 2, 50, 5146, 751, 5146, 5, 2, 221, 3727, 6, 9, 1167, 373, 9, 5, 5146, 7, 5, 1343, 13, 2, 514
6, 1, 250, 7, 98, 4270, 56, 2316, 0, 928, 11, 11, 9, 16, 5, 5146, 5146, 6, 50, 69, 27, 280, 27, 108, 1045, 0, 2633, 4177, 3180, 17, 1675,
1, 2571]
sentence label id is: 0

--------------------------
sentence list is:  <unk> has much in common with the third man another <unk> film set among the <unk> of <unk> europe like <unk> there is
much inventive camera work there is an innocent american who gets emotionally involved with a woman he doesnt really understand and whose
<unk> is all the more striking in contrast with the <unk> br but id have to say that the third man has a more <unk> storyline <unk> is a b
it disjointed in this respect perhaps this is <unk> it is presented as a <unk> and making it too coherent would spoil the effect br br thi
s movie is <unk> <unk> in more than one sense one never sees the sun shine grim but intriguing and frightening
sentence label is:  negative
```

文本数据中，每一句话的长度都是不一样的，为了方便后续的神经网络的计算，常见的处理方法是把数据集中的数据都统一成同样长度的数据。这包括：对于较长的数据进行截断处理，对于较短的数据用特殊的词< pad >进行填充。接下来的代码会对数据集中的数据进行这样的处理：

```
def create_padded_dataset(dataset):
    padded_sents = []
    labels = []
    for batch_id, data in enumerate(dataset): #遍历数据截断或补齐
        sent, label = data[0], data[1]
        padded_sent = np.concatenate([sent[:seq_len], [pad_id] * (seq_len - len(sent))]).
astype('int32')
        padded_sents.append(padded_sent)
        labels.append(label)
return np.array(padded_sents), np.array(labels)

train_sents, train_labels = create_padded_dataset(train_dataset)
test_sents, test_labels = create_padded_dataset(test_dataset)
```

NumPy 提供了 numpy.concatenate((a1,a2,…),axis=0)函数，可以实现多个数组的拼接。将前面准备好的训练集与测试集用 Dataset 与 DataLoader 封装后，完成数据的加载。

```
class IMDBDataset(paddle.io.Dataset):
    def __init__(self, sents, labels): #初始化
        self.sents = sents
        self.labels = labels
    def __getitem__(self, index):
        data = self.sents[index]
        label = self.labels[index]
        return data, label
    def __len__(self):
        return len(self.sents)

train_dataset = IMDBDataset(train_sents, train_labels)
test_dataset = IMDBDataset(test_sents, test_labels)
train_loader = paddle.io.DataLoader(train_dataset, return_list=True, shuffle=True,
                                    batch_size=batch_size, drop_last=True)
test_loader = paddle.io.DataLoader(test_dataset, return_list=True, shuffle=True,
                                   batch_size=batch_size, drop_last=True)
```

DataLoader 返回一个迭代器，该迭代器根据 batch_sampler 给定的顺序迭代一次给定的 dataset。DataLoader 支持单进程和多进程的数据加载方式，当 num_workers 大于 0 时，将使用多进程方式异步加载数据。数据加载时要根据内存和效率选择合适的比例。一般我们还会将训练集中的 30%作为验证集，用于检查模型训练过程的收敛情况，防止过拟合。

步骤 2：搭建前馈神经网络

数据准备的工作完成之后，接下来我们将动手来搭建一个电影评论情感分析模型，进行文本特征的提取，从而实现评论情感极性的检测。我们使用一个不考虑词的顺序的 BOW 的网络，在查找到每个词对应的 embedding 后，简单地取平均，作为一个句子的表示。然后用 Linear 进行线性变换。为了防止过拟合，还使用了 Dropout。

1. 模型定义

如图 3.1 所示，前馈神经网络就像一个复杂的函数，理论上可以完成从确定形式的输入到确定形式的输出的任何映射。然而，前馈神经网络只能完成信息的单向传递，这一特性虽然使得模型容易训练，但也在某种程度上限制了模型的能力。所以在本章的后文中，我们会引入复杂的模型结构，来学习更丰富的语义信息。

```
class MyNet(paddle.nn.Layer):
    def __init__(self):
        super(MyNet, self).__init__()
        self.emb = paddle.nn.Embedding(vocab_size, emb_size)                  #编码层
        self.fc = paddle.nn.Linear(in_features=emb_size, out_features=2)      #线性层
        self.dropout = paddle.nn.Dropout(0.5)                                 #Dropout层

    def forward(self, x):                                                     #前向计算
        x = self.emb(x)
        x = paddle.mean(x, axis=1)
        x = self.dropout(x)
        x = self.fc(x)
        return x
```

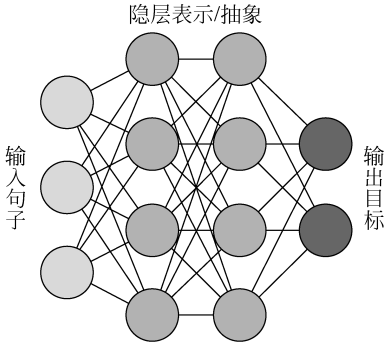

图 3.1　前馈神经网络

嵌入层（Embedding Layer）该接口用于构建 Embedding 的一个可调用对象。其根据 input 中的 id 信息从 embedding 矩阵中查询对应 embedding 信息，并根据输入的 size（num_embeddings，embedding_dim）和 weight_attr 自动构造一个二维 embedding 矩阵。输出的 Tensor 的 shape 是在输入 Tensor shape 的最后一维后面添加了 embedding_dim 的维度。

线性变换层（Linear Layer）只接受一个 Tensor 作为输入，形状为 [batch_size，*，in_features]，其中 * 表示可以为任意个额外的维度。该层可以计算输入 Tensor 与权重矩阵 $\boldsymbol{W}$

的乘积，然后生成形状为 [batch_size, *, out_features]的输出 Tensor。如果 bias_attr 不是 False，则将创建一个偏置参数并将其添加到输出中。

Dropout 是一种正则化手段，该算子根据给定的丢弃概率 p，在训练过程中随机将一些神经元输出设置为 0，通过阻止神经元节点间的相关性来减少过拟合。forward(self,x)函数是前向计算过程。

2. 损失函数

接着是定义损失函数，这里使用的是交叉熵损失函数，该函数在分类任务上比较常用。定义了一个损失函数之后，还要对它求平均值，因为定义的是一个 batch 的损失值。同时还可以定义一个准确率函数，可以在训练的时候输出分类的准确率。

这里使用的是 paddle.nn.functional.cross_entropy()，该 API 实现了 softmax 交叉熵损失函数。该函数会将 softmax 操作、交叉熵损失函数的计算过程进行合并，从而提供了数值上更稳定的计算。

```
# 获取损失函数和准确率
logits = model(sent)
loss = paddle.nn.functional.cross_entropy(logits, label)
acc = paddle.metric.accuracy(logits, label)
```

paddle.metric.accuracy(input,label,k=1,correct=None,total=None,name=None)使用输入和标签计算准确率。如果正确的标签在 topk 个预测值里，则计算结果加 1。注意：输出正确率的类型由 input 类型决定，input 和 label 的类型可以不一样。

3. 优化方法

接着定义优化算法，这里使用的是 Adam 优化算法，指定学习率为 0.001。paddle.optimizer.Adam()能够利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率。

```
# 定义优化方法
opt = paddle.optimizer.Adam(learning_rate = 0.001,parameters = model.parameters())
```

步骤 3：训练前馈神经网络

在步骤 2 中定义好了网络模型，即构造好了核心 train()函数，在正式进行网络训练前，首先进行参数初始化。

```
vocab_size = len(word_dict) + 1     # 词典大小
emb_size = 256                      # embedding 维度
seq_len = 200                       # 文本长度
batch_size = 32                     # batch_size 大小
epochs = 5                          # 迭代轮数
```

之后就可以进行正式的训练了，本实践中设置训练轮数 5。在每轮训练中，每 500 个 batch，打印一次训练平均误差和准确率。每轮训练完成后，使用验证集进行一次验证。

```
# 开始训练
def train(model):
    model.train()
```

```
opt = paddle.optimizer.Adam(learning_rate = 0.001, parameters = model.parameters())
steps = 0
Iters, total_loss, total_acc = [], [], []
for epoch in range(epochs):
    for batch_id, data in enumerate(train_loader):
        steps += 1
        sent = data[0]
        label = data[1]

        logits = model(sent)
        loss = paddle.nn.functional.cross_entropy(logits, label)
        acc = paddle.metric.accuracy(logits, label)

        if batch_id % 500 == 0:
            Iters.append(steps)
            total_loss.append(loss.numpy()[0])
            total_acc.append(acc.numpy()[0])
            print("epoch: {}, batch_id: {}, loss is: {}".format(epoch, batch_id, loss.numpy()))
        loss.backward()
        opt.step()
        opt.clear_grad()
```

训练完成后，对模型进行保存，使用飞桨提供的 paddle.save()进行模型保存。

```
paddle.save(model.state_dict(),str(epoch) + "_model_final.pdparams")
```

步骤 4：模型评估

通过观察训练过程中误差和准确率随着迭代次数的变化趋势，可对网络训练结果进行评估。使用 test_loader 对模型进行验证得到最终的准确率为 0.86。通过图 3.2 和图 3.3 可以观察到，在训练和验证过程中平均误差是在逐步降低的，与此同时，准确率逐步趋近于 100%。

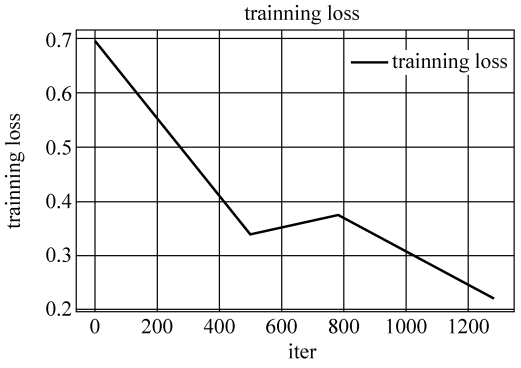

图 3.2　训练过程 loss 变化折线图

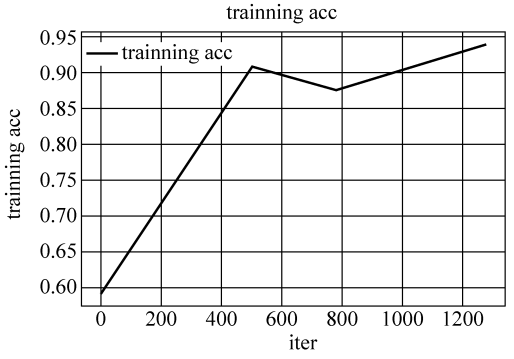

图 3.3　训练过程准确率变化折线图

```
for batch_id, data in enumerate(test_loader):

    sent = data[0]
    label = data[1]

    logits = model(sent)
    loss = paddle.nn.functional.cross_entropy(logits, label)
    acc = paddle.metric.accuracy(logits, label)

    accuracies.append(acc.numpy())
    losses.append(loss.numpy())

avg_acc, avg_loss = np.mean(accuracies), np.mean(losses)
print("[validation] accuracy: {}, loss: {}".format(avg_acc, avg_loss))
```

步骤 5：情感分析预测

前面已经进行了模型训练，并保存了训练好的模型。接下来就可以使用训练好的模型进行电影评论情感分析了。为了进行预测，我们任意选取一个评论文本。我们把文本中的每个词对应到 dict 中的 id。如果词典中没有这个词，则设为< unk >。然后我们用 model. eval()来使用模型预测结果。

```
model_state_dict = paddle.load('model_final.pdparams')
model = MyNet()
model.set_state_dict(model_state_dict)
model.eval()
sent = data[0]
results = model(sent)
```

训练集用于训练模型，验证集用于进行错误分析，测试集用于系统的最终评估，以下是我们展示的一条测试集的预测结果：

```
 数据: cheesy 80s horror <unk> genre <unk> ken <unk> and <unk> cash along with <unk> <unk> are some of the featured players in this tale ab
out a haunted health club goofy dialogue and some nasty gore effects make this movie watchable not bad but no great <unk> eitherbr br reco
mmended for the bad dialogue and acting bmovie fans <unk> br b <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pa
d> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pa
d> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pa
d> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pa
d> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pa
d> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pa
d> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
 情感: positive
```

模型结构和编码特征对于该任务非常重要，目前仍有很大的提升空间，可以通过打印一些预测错误的文本数据分析模型的进步空间。通过曲线图发现模型尚未过拟合，也可以增加迭代轮数继续训练观测指标曲线变化趋势。在后文中，我们会继续引入更适合的模型和调优策略。

实践十一：基于 LSTM 网络的谣言检测

社交媒体的发展在加速信息传播的同时，也带来了虚假谣言信息的泛滥，往往会引发诸多不安定因素，并对经济和社会产生巨大的影响。在新型冠状病毒感染的肺炎疫情防控的

关键期，网上各种有关疫情防控的谣言接连不断，从“钟南山院士被感染”到“10 万人感染肺炎”等，这些不切实际的谣言“操纵”舆论，误导了公众的判断，影响了社会稳定。因此谣言检测变得极为重要。

本次实践使用基于长短时记忆网络（LSTM）构建了谣言检测模型，将文本中的谣言事件向量化，通过 LSTM 网络的学习训练来挖掘表示文本深层的特征，避免了特征构建的问题，并能发现那些不容易被人发现的特征，从而产生更好的效果。本实践代码运行的环境配置如下：Python 版本为 3.7，PaddlePaddle 版本为 2.0.0，操作平台为 AI Studio。

步骤 1：数据准备

本实践所使用的数据是从新浪微博不实信息举报平台抓取的中文谣言数据，数据集中共包含 1538 条谣言和 1849 条非谣言。如图 3.4 所示，每条数据均为 json 格式，其中 text 字段代表微博原文的文字内容。更多数据集介绍请参考 https://github.com/thunlp/Chinese_Rumor_Dataset。


```
{
"multi":null,
"text":"【每日一书】《全球通史》[美] 斯塔夫里阿诺斯 著 北京大学出版社这部潜心力作自
"user":{
"verified":true,
"description":true,
"gender":"m",
"messages":23602,
"followers":6065984,
"location":"广东 广州",
"time":1251448522,
"friends":1550,
"verified_type":3
},
"has_url":false,
"comments":125,
"pics":1,
"source":"定时showone",
"likes":0,
"time":1333976404,
"reposts":365
}
```


图 3.4　数据格式示例

首先需要解压数据，读取并解析数据，生成数据文件 all_data.txt，根据全部文本数据生成数据字典，即 dict.txt，然后进行训练集与验证集的划分：train_list.txt 和 eval_list.txt，最后定义训练数据集提供器，方便模型训练。

```
if(not os.path.isdir(target_path)):
    z = zipfile.ZipFile(src_path, 'r')
    z.extractall(path=target_path)
    z.close()
```

为了构建数据和标签的对应关系我们需要设置两个标签，并解析数据：

```
rumor_label = "0"
non_rumor_label = "1"
```

遍历所有的数据来生成数据字典：

```
def create_dict(data_path, dict_path):
    with open(dict_path, 'w') as f:
        f.seek(0)
        f.truncate()
    dict_set = set()
    # 读取全部数据
    with open(data_path, 'r', encoding = 'utf - 8') as f:
        lines = f.readlines()
    # 把数据生成一个元组
    for line in lines:
        content = line.split('\t')[ - 1].replace('\n', '')
        for s in content:
            dict_set.add(s)
    # 把元组转换成字典,一个字对应一个数字
    dict_list = []
    i = 0
    for s in dict_set:
        dict_list.append([s, i])
        i += 1
    # 添加未知字符
    dict_txt = dict(dict_list)
    end_dict = {"< unk >": i}
    dict_txt.update(end_dict)
    end_dict = {"< pad >": i + 1}
    dict_txt.update(end_dict)
    # 把这些字典保存到本地中
    with open(dict_path, 'w', encoding = 'utf - 8') as f:
        f.write(str(dict_txt))
```

在飞桨框架 2.0 版本中,推荐使用 paddle.io.DataLoader 的方式对数据进行加载。

```
class RumorDataset(paddle.io.Dataset):
    def __init__(self, data_dir):
        self.data_dir = data_dir
        self.all_data = []
        with io.open(self.data_dir, "r", encoding = 'utf8') as fin:
            for line in fin:
                cols = line.strip().split("\t")
                if len(cols) != 2:
                    sys.stderr.write("[NOTICE] Error Format Line!")
                    continue
                label = []
                label.append(int(cols[1]))
                wids = cols[0].split(",")
                if len(wids)> = 150:
                    wids = np.array(wids[:150]).astype('int64')
                else:
                    wids = np.concatenate([wids, [vocab["< pad >"]] * (150 - len(wids))]).
astype('int64')
```

```
                label = np.array(label).astype('int64')
                self.all_data.append((wids, label))
    def __getitem__(self, index):
        data, label = self.all_data[index]
        return data, label
    def __len__(self):
        return len(self.all_data)
train_dataset = RumorDataset(os.path.join(data_root_path, 'train_list.txt'))
test_dataset = RumorDataset(os.path.join(data_root_path, 'eval_list.txt'))

train_loader = paddle.io.DataLoader(train_dataset, places = paddle.CPUPlace(), return_list =
True, shuffle = True, batch_size = batch_size, drop_last = True)
test_loader = paddle.io.DataLoader(test_dataset, places = paddle.CPUPlace(), return_list =
True, shuffle = True, batch_size = batch_size, drop_last = True)
```

步骤 2：搭建长短时记忆网络模型

数据准备的工作完成之后，接下来我们将动手来搭建一个谣言检测模型，进行文本特征的提取，从而判断一个 claim（可能是一句话，可能是一个段落甚至一篇文章）是真还是假。我们使用一个循环神经网络 RNN 的变体结构 LSTM 网络构建。

1. 模型定义

1997 年，人工智能研究所的主任 Jurgen Schmidhuber 提出长短期记忆（LSTM），LSTM 使用门控单元及记忆机制缓解了早期 RNN 训练时梯度消失（gradient vanishing）及梯度爆炸（gradient exploding）的问题。长短记忆神经网络——通常称作 LSTM，是一种特殊的 RNN，能够学习长的依赖关系。它们由 Hochreiter&Schmidhuber 引入，并被许多人进行了改进和普及。它们在各种各样的问题上工作得非常好，现在被广泛使用。

LSTM 是为了避免长依赖问题而精心设计的。记住较长的历史信息实际上是它们的默认行为，而不是它们努力学习的东西。所有循环神经网络都具有神经网络的重复模块链的形式。在标准的 RNN 中，该重复模块将具有非常简单的结构，例如单个 tanh()层，如图 3.5 所示。LSTM 也拥有这种链状结构，但是重复模块则拥有不同的结构。与神经网络的简单的一层相比，LSTM 拥有四层，这四层以特殊的方式进行交互，如图 3.6 所示。

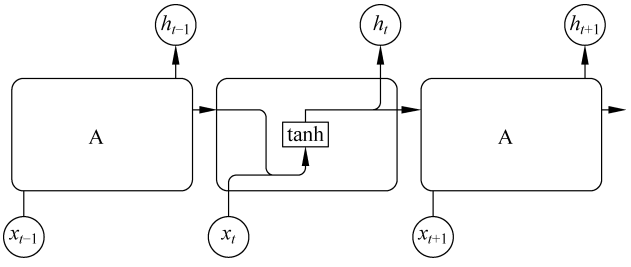

图 3.5　标准 RNN 中重复模块的单层神经网络

在了解了 LSTM 网络后，接下来就可以使用飞桨深度学习开源框架来搭建一个 LSTM 网络来解决谣言检测问题。

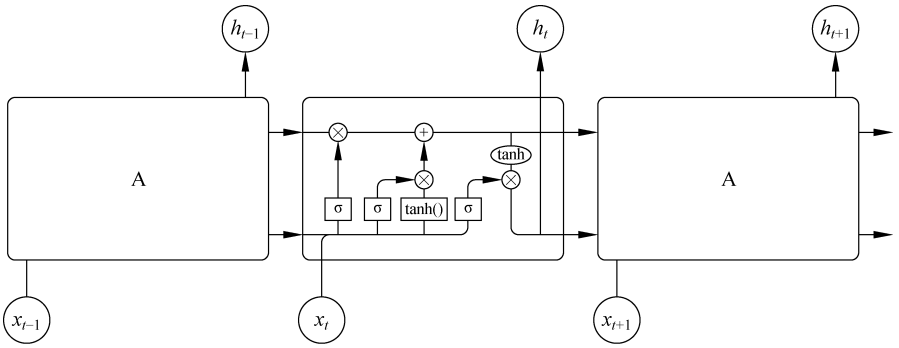

图 3.6　LSTM 中的重复模块包含的四层交互神经网络层

```
Class LSTM(paddle.nn.Layer):
    def __init__(self):                                          #参数初始化
        super(RNN, self).__init__()
        self.dict_dim = vocab["<pad>"]
        self.emb_dim = 128
        self.hid_dim = 128
        self.class_dim = 2
        self.embedding = Embedding(                              #embedding
            self.dict_dim + 1, self.emb_dim,
            sparse = False)
        self._fc1 = Linear(self.emb_dim, self.hid_dim)           #线性层
        self.lstm = paddle.nn.LSTM(self.hid_dim, self.hid_dim)   #LSTM
        self.fc2 = Linear(19200, self.class_dim)                 #线性层

    def forward(self, inputs):                                   #前向计算网络
        # [32, 150]
        emb = self.embedding(inputs)
        # [32, 150, 128]
        fc_1 = self._fc1(emb)
        # [32, 150, 128]
        x = self.lstm(fc_1)
        x = paddle.reshape(x[0], [0, -1])
        x = self.fc2(x)
        return x
```

paddle.nn.LSTM()是长短期记忆网络(LSTM),根据输出序列和给定的初始状态计算返回输出序列和最终状态。在该网络中的每一层对应输入的 step,每个 step 根据当前时刻输入 x_t 和上一时刻状态 h_{t-1},c_{t-1} 计算当前时刻输出 y_t 并更新状态 h_t,c_t。

2. 损失函数

接着是定义损失函数,这里使用的是交叉熵损失函数,该函数在分类任务上比较常用。定义了一个损失函数之后,还要对它求平均值,因为定义的是一个 Batch 的损失值。同时还可以定义一个准确率函数,可以在训练的时候输出分类的准确率。

```
# 获取损失函数和准确率
logits = model(sent)
```

```
loss = paddle.nn.functional.cross_entropy(logits, label)
acc = paddle.metric.accuracy(logits, label)
```

3. 优化方法

接着定义优化算法，这里使用的是 Adam 优化算法，指定学习率为 0.001。

```
# 定义优化方法
opt = paddle.optimizer.Adam(learning_rate = 0.001,parameters = model.parameters())
```

步骤 3：模型训练

在步骤 2 中定义好了网络模型，即构造好了核心 train()函数，在正式进行网络训练前，首先进行参数初始化。

```
vocab_size = len(word_dict) + 1
emb_size = 256
seq_len = 200
batch_size = 32
epochs = 5
```

之后就可以进行正式的训练了，本实践设置训练轮数 5。在每轮训练中，每 500 个 batch，打印一次训练平均误差和准确率。每轮训练完成后，使用验证集进行一次验证。

```
# 开始训练
def train(model):
    model.train()
    opt = paddle.optimizer.Adam(learning_rate = 0.001, parameters = model.parameters())
    steps = 0
    Iters, total_loss, total_acc = [], [], []
    for epoch in range(epochs):
        for batch_id, data in enumerate(train_loader):
            steps += 1
            sent = data[0]
            label = data[1]
            logits = model(sent)
            loss = paddle.nn.functional.cross_entropy(logits, label)
            acc = paddle.metric.accuracy(logits, label)
            if batch_id % 500 == 0:
                Iters.append(steps)
                total_loss.append(loss.numpy()[0])
                total_acc.append(acc.numpy()[0])
                print("epoch: {}, batch_id: {}, loss is: {}".format(epoch, batch_id, loss.
numpy()))
            loss.backward()
            opt.step()
            opt.clear_grad()
```

训练完成后，使用飞桨提供的 paddle.save()进行模型保存：

```
paddle.save(model.state_dict(),str(epoch) + "_model_final.pdparams")
```

步骤 4：模型评估

通过观察训练过程中误差和准确率随着迭代次数的变化趋势，可对网络训练结果进行评估。使用 test_loader 对模型进行验证得到最终的准确率为 0.89。通过图 3.7 和图 3.8 可以观察到，在训练和验证过程中平均误差是在逐步降低的，与此同时，准确率逐步趋近于 100%。

```
for batch_id, data in enumerate(test_loader):

    sent = data[0]
    label = data[1]

    logits = model(sent)
    loss = paddle.nn.functional.cross_entropy(logits, label)
    acc = paddle.metric.accuracy(logits, label)

    accuracies.append(acc.numpy())
    losses.append(loss.numpy())

avg_acc, avg_loss = np.mean(accuracies), np.mean(losses)
print("[validation] accuracy: {}, loss: {}".format(avg_acc, avg_loss))
```

图 3.7　训练过程 Loss 变化折线图

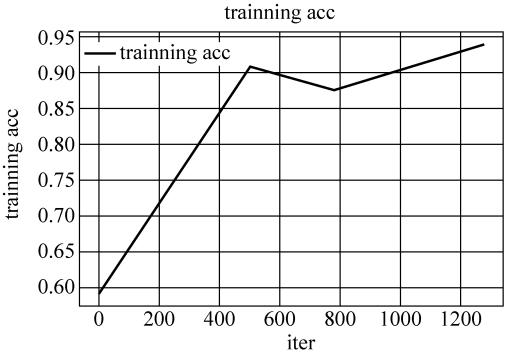

图 3.8　训练过程准确率变化折线图

步骤 5：谣言信息预测

前面已经进行了模型训练，并保存了训练好的模型。接下来就可以使用训练好的模型进行谣言预测了。为了进行预测，我们任意选取 1 个文本数据。我们把文本中的每个词对应到 dict 中的 id。如果词典中没有这个词，则设为< unk >。然后我们用 model.eval()来使用模型预测结果。

```
model_state_dict = paddle.load('model_final.pdparams')
model = RNN()
model.set_state_dict(model_state_dict)
model.eval()
```

```
sent = data[0]
results = model(sent)
```

训练集用于训练模型，验证集用于进行错误分析，测试集用于系统的最终评估，以下是我们展示的一条测试集的预测结果：

数据：

【SoLoMo】 美国 KPCB 老大约翰·杜尔今年 2 月的时候第一次提出了 SoLoMo 这个概念，他把最热的三个关键词整合到了一起：Social（社交）、Local（本地化）和 Mobile（移动）。过去几年，我一直在投资移动互联网和社区公司，也顺应了这个潮流，米聊争取成为这个潮流最火爆的产品。欢迎更多的人

是否谣言：

否

模型结构和编码特征对于该任务非常重要，目前仍有很大的提升空间，可以通过打印一些预测错误的文本数据分析模型的进步空间。通过曲线图发现模型尚未过拟合，也可以增加迭代轮数继续训练观测指标曲线变化趋势。在后文中，我们会继续引入更适合的模型和调优策略。

实践十二：基于 PaddleHub 的低俗文本审核

社交媒体色情检测模型可自动判别文本是否涉黄并给出相应的置信度，对文本中的色情描述、低俗交友、污秽文案进行识别。PaddleHub 提供了一键调用模型 porn_detection_lstm，其采用 LSTM 网络结构并按字粒度进行切词，具有较高的分类精度。该模型最大句子长度为 256 字，仅支持预测。

下面我们学习训练一个自己的文本审核模型，这里只考虑微博文本情绪数据包括消极、中性、积极，该实践需要解决的问题其实是文本分类问题。随着 2018 年 ELMo、BERT 等模型的发布，NLP 领域进入了“大力出奇迹”的时代。采用大规模语料上进行无监督预训练的深层模型，在下游任务数据上微调一下，即可达到很好的效果。曾经需要反复调参、精心设计结构的任务，现在只需简单地使用更大的预训练数据、更深层的模型便可解决。因此 NLP 比赛的入手门槛也随之变低。新手只要选择好预训练模型，在自己数据上微调，很多情况下就可以达到很好的效果。

本项目将采用百度出品的 PaddleHub 预训练模型微调工具，如图 3.9 所示，快速构建方案。模型方面则选择 ERNIE 模型。整个流程可以大致分为如图 3.10 所示的 6 步。

步骤 1：数据准备

在实践中，我们需要将数据分为训练数据和验证数据，分别用于模型的训练和测试模型的效果。因此，在这里我们遍历数据，按照比例生成 train 和 valid。train 中存储的是用于训练的数据和对应的标签，valid 中存储的是用于验证的数据和对应的标签。

PaddleHub 文本任务的输入数据格式为（见图 3.11）：

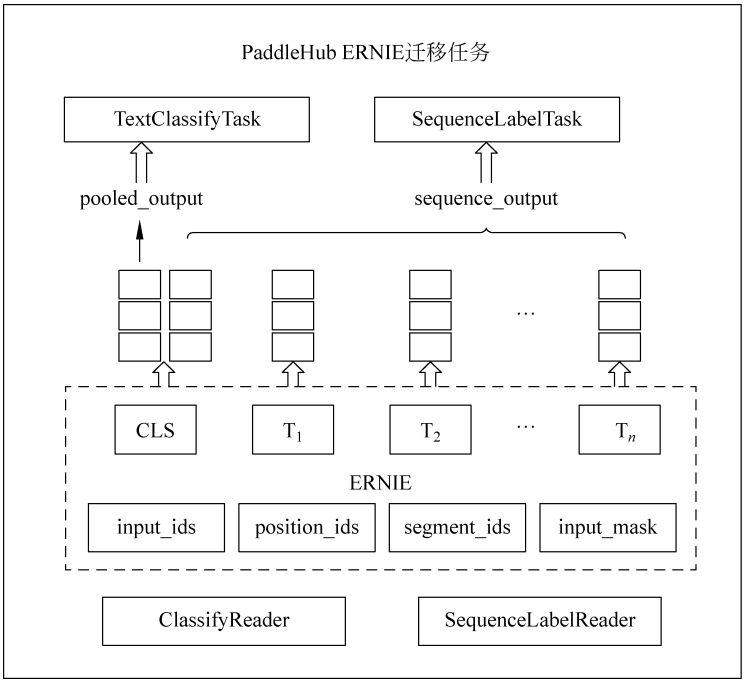

图 3.9　PaddleHub ERNIE 迁移任务

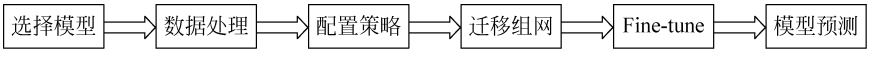

图 3.10　迁移流程

text	label
我想吃雪糕	0
…	…

图 3.11　数据格示示例

我们将数据划分为训练集和验证集，比例为 8∶2，然后保存为文本文件，两列需用 Tab 键分隔符隔开。

```
# 划分验证集,保存格式  text[\t]label
from sklearn.model_selection import train_test_split
train_labled = train_labled[['微博中文内容', '情感倾向']]
train, valid = train_test_split(train_labled, test_size=0.2, random_state=2020)
train.to_csv('/home/aistudio/data/data22724/train.txt', index=False, header=False, sep='\t')
valid.to_csv('/home/aistudio/data/data22724/valid.txt', index=False, header=False, sep='\t')
```

步骤 2：自定义数据加载

加载文本类自定义数据集，用户仅需要继承基类 TextClassificationDataset，修改数据集

存放地址以及类别即可。这里我们没有带标签的测试集，所以 test_file 直接用验证集代替 "valid.txt" 。

```
# 自定义数据集
from paddlehub.datasets.base_nlp_dataset import InputExample, TextClassificationDataset
# 数据集存放位置
DATA_DIR = "/home/aistudio/data/data22724"
class News(TextClassificationDataset):
    def __init__(self, tokenizer, mode = 'train', max_seq_len = 128):
        if mode == 'train':
            data_file = 'train.txt'
        elif mode == 'test':
            data_file = 'valid.txt'
        else:
            data_file = 'valid.txt'
        super(News, self).__init__(
            base_path = DATA_DIR,
            data_file = data_file,
            tokenizer = tokenizer,
            max_seq_len = max_seq_len,
            mode = mode,
            is_file_with_header = True,
            label_list = ["-1", "0", "1"])

    # 解析文本文件里的样本
    def _read_file(self, input_file, is_file_with_header: bool = False):
        if not os.path.exists(input_file):
            raise RuntimeError("The file {} is not found.".format(input_file))
        else:
            with io.open(input_file, "r", encoding = "UTF-8") as f:
                reader = csv.reader(f, delimiter = "\t", quotechar = None)
                examples = []
                seq_id = 0
                header = next(reader) if is_file_with_header else None
                for line in reader:
                    example = InputExample(guid = seq_id, text_a = line[0], label = line[1])
                    seq_id += 1
                    examples.append(example)
                return examples
```

步骤 3：加载模型

这里我们选择 ERNIE1.0 的中文预训练模型。其他模型见：https://github.com/PaddlePaddle/PaddleHub。只需修改 name='xxx' 就可以切换不同的模型。

```
# 加载模型
model = hub.Module(name = "ernie",task = 'seq-cls',num_classes = 3) # 在多分类任务中,num_
#classes 需要显式地指定类别数,此处根据数据集设置为 3
```

步骤 4：构建 Reader

构建一个文本分类的 reader，reader 负责将 dataset 的数据进行预处理，首先对文本进行切词，接着以特定格式组织并输入给模型进行训练。通过 max_seq_len 可以修改最大序列长度，若序列长度不足，会通过 padding 方式补到 max_seq_len，若序列长度大于该值，则会以截断方式让序列长度为 max_seq_len，这里我们设置为 128。

```
train_dataset = News(model.get_tokenizer(), mode='train', max_seq_len=128)
dev_dataset = News(model.get_tokenizer(), mode='dev', max_seq_len=128)
test_dataset = News(model.get_tokenizer(), mode='test', max_seq_len=128)
```

步骤 5：finetune 策略和运行配置

选择迁移优化策略。详见 https://github.com/PaddlePaddle/PaddleHub/wiki/PaddleHub-API:-Strategy 此处我们设置最大学习率为 learning_rate=5e-5。trainer 是 fine-tune 任务的执行者。

```
optimizer = paddle.optimizer.Adam(learning_rate=5e-5, parameters=model.parameters())
# 优化器的选择和参数配置
trainer = hub.Trainer(model, optimizer, checkpoint_dir='./ckpt', use_gpu=True)
```

步骤 6：开始 finetune

我们使用 trainer.train 接口就可以开始模型训练，finetune 过程中，会周期性的进行模型效果的评估。

```
# 配置训练参数，启动训练，并指定验证集
trainer.train(train_dataset, epochs=3, batch_size=32, eval_dataset=dev_dataset, save_
interval=1)
# 在测试集上评估当前训练模型
result = trainer.evaluate(test_dataset, batch_size=32)
```

步骤 7：预测

当 Finetune 完成后，加载训练后保存的最佳模型来进行预测。

```
# 预测
data = [["文本 1"],["文本 2"],["文本 3"]]
label_list=["-1", "0", "1"])
label_map = {
    idx: label_text for idx, label_text in enumerate(label_list)
}
model = hub.Module(
    name='ernie',
    task='seq-cls',
    load_checkpoint='./ckpt/best_model/model.pdparams',
```

```
    label_map = label_map)
results = model.predict(data, max_seq_len = 128, batch_size = 1, use_gpu = True)
for idx, text in enumerate(data):
    print('Data: {} \t Lable: {}'.format(text[0], results[idx]))
```

至此我们就完成了使用 PaddleHub 语义预训练模型 ERNIE 对低俗文本进行审核完成文本分类任务。