情感分析

背景介绍

在自然语言处理中，情感分析一般是指判断一段文本所表达的情绪状态。其中，一段文本可以是一个句子，一个段落或一个文档。情绪状态可以是两类，如（正面，负面），（高兴，悲伤）；也可以是三类，如（积极，消极，中性）等等。情感分析的应用场景十分广泛，如把用户在购物网站（亚马逊、天猫、淘宝等）、旅游网站、电影评论网站上发表的评论分成正面评论和负面评论；或为了分析用户对于某一产品的整体使用感受，抓取产品的用户评论并进行情感分析等等。表格1展示了对电影评论进行情感分析的例子：

电影评论	**类别**
在冯小刚这几年的电影里，算最好的一部的了	正面
很不好看，好像一个地方台的电视剧	负面
圆方镜头全程炫技，色调背景美则美矣，但剧情拖沓，口音不伦不类，一直努力却始终无法入戏	负面
剧情四星。但是圆镜视角加上婺源的风景整个非常有中国写意山水画的感觉，看得实在太舒服了。。	正面

表格 1 电影评论情感分析

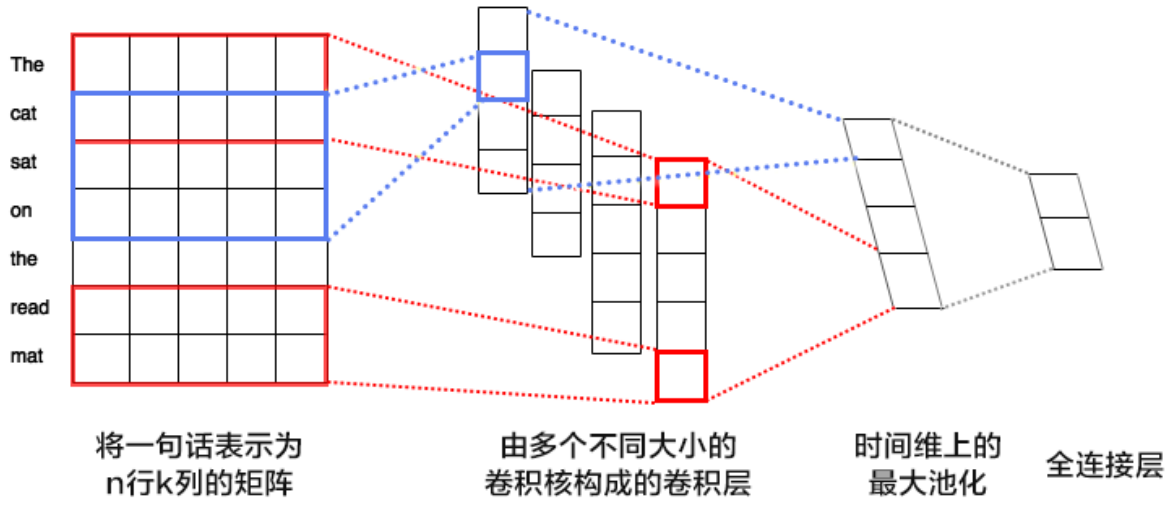

图1. 卷积神经网络文本分类模型

假设待处理句子的长度为n，其中第i个词的词向量（word embedding）为$x_i \in \mathbb{R}^k$，k为维度大小。

首先，进行词向量的拼接操作：将每h个词拼接起来形成一个大小为h的词窗口，记为$x_{i:i+h-1}$，它表示词序列$x_i, x_{i+1}, \ldots, x_{i+h-1}$的拼接，其中，$i$表示词窗口中第一个词在整个句子中的位置，取值范围从1到$n-h+1$，$x_{i:i+h-1} \in \mathbb{R}^{hk}$。

其次，进行卷积操作：把卷积核(kernel)$w \in \mathbb{R}^{hk}$应用于包含h个词的窗口$x_{i:i+h-1}$，得到特征$c_i = f(w \cdot x_{i:i+h-1} + b)$，其中$b \in \mathbb{R}$为偏置项（bias），$f$为非线性激活函数，如$sigmoid$。将卷积核应用于句子中所有的词窗口$x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}$，产生一个特征图（feature map）：

$$c = [c_1, c_2, \ldots, c_{n-h+1}], c \in \mathbb{R}^{n-h+1}$$

接下来，对特征图采用时间维度上的最大池化（max pooling over time）操作得到此卷积核对应的整句话的特征$\hat{c}$，它是特征图中所有元素的最大值：

$$\hat{c} = max(c)$$

在实际应用中，我们会使用多个卷积核来处理句子，窗口大小相同的卷积核堆叠起来形成一个矩阵（上文中的单个卷积核参数w相当于矩阵的某一行），这样可以更高效的完成运算。另外，我们也可使用窗口大小不同的卷积核来处理句子（图1作为示意画了四个卷积核，不同颜色表示不同大小的卷积核操作）。

最后，将所有卷积核得到的特征拼接起来即为文本的定长向量表示，对于文本分类问题，将其连接至softmax即构建出完整的模型。

对于一般的短文本分类问题，上文所述的简单的文本卷积网络即可达到很高的正确率[1]。若想得到更抽象更高级的文本特征表示，可以构建深层文本卷积神经网络[2,3]。

循环神经网络（RNN）

循环神经网络是一种能对序列数据进行精确建模的有力工具。实际上，循环神经网络的理论计算能力是图灵完备的[4]。自然语言是一种典型的序列数据（词序列），近年来，循环神经网络及其变体（如long short term memory[5]等）在自然语言处理的多个领域，如语言模型、句法解析、语义角色标注（或一般的序列标注）、语义表示、图文生成、对话、机器翻译等任务上均表现优异甚至成为目前效果最好的方法。

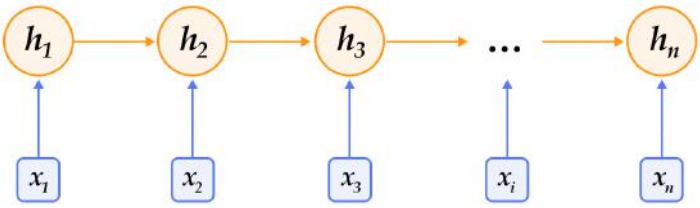

图2. 循环神经网络按时间展开的示意图

循环神经网络按时间展开后如图2所示：在第t时刻，网络读入第t个输入x_t（向量表示）及前一时刻隐层的状态值h_{t-1}（向量表示，h_0一般初始化为0向量），计算得出本时刻隐层的状态值h_t，重复这一步骤直至读完所有输入。如果将循环神经网络所表示的函数记为f，则其公式可表示为：

$$h_t = f(x_t, h_{t-1}) = \sigma(W_{xh}x_t + W_{hh}h_{h-1} + b_h)$$

其中W_{xh}是输入到隐层的矩阵参数，W_{hh}是隐层到隐层的矩阵参数，b_h为隐层的偏置向量（bias）参数，σ为

$sigmoid$函数。

在处理自然语言时，一般会先将词（one-hot表示）映射为其词向量（word embedding）表示，然后再作为循环神经网络每一时刻的输入x_t。此外，可以根据实际需要的不同在循环神经网络的隐层上连接其它层。如，可以把一个循环神经网络的隐层输出连接至下一个循环神经网络的输入构建深层（deep or stacked）循环神经网络，或者提取最后一个时刻的隐层状态作为句子表示进而使用分类模型等等。

长短期记忆网络（LSTM）

对于较长的序列数据，循环神经网络的训练过程中容易出现梯度消失或爆炸现象[6]。为了解决这一问题，Hochreiter S, Schmidhuber J. (1997)提出了LSTM(long short term memory[5])。

相比于简单的循环神经网络，LSTM增加了记忆单元c、输入门i、遗忘门f及输出门o。这些门及记忆单元组合起来大大提升了循环神经网络处理长序列数据的能力。若将基于LSTM的循环神经网络表示的函数记为F，则其公式为：

$$h_t = F(x_t, h_{t-1})$$

F由下列公式组合而成[7]：

$$\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{h-1} + W_{ci}c_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{h-1} + W_{cf}c_{t-1} + b_f) \\
c_t &= f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{h-1} + b_c) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{h-1} + W_{co}c_t + b_o) \\
h_t &= o_t \odot tanh(c_t)
\end{aligned}$$

其中，i_t, f_t, c_t, o_t分别表示输入门，遗忘门，记忆单元及输出门的向量值，带角标的W及b为模型参数，$tanh$为双曲正切函数，⊙表示逐元素（elementwise）的乘法操作。输入门控制着新输入进入记忆单元c的强度，遗忘门控制着记忆单元维持上一时刻值的强度，输出门控制着输出记忆单元的强度。三种门的计算方式类似，但有着完全不同的参数，它们各自以不同的方式控制着记忆单元c，如图3所示：

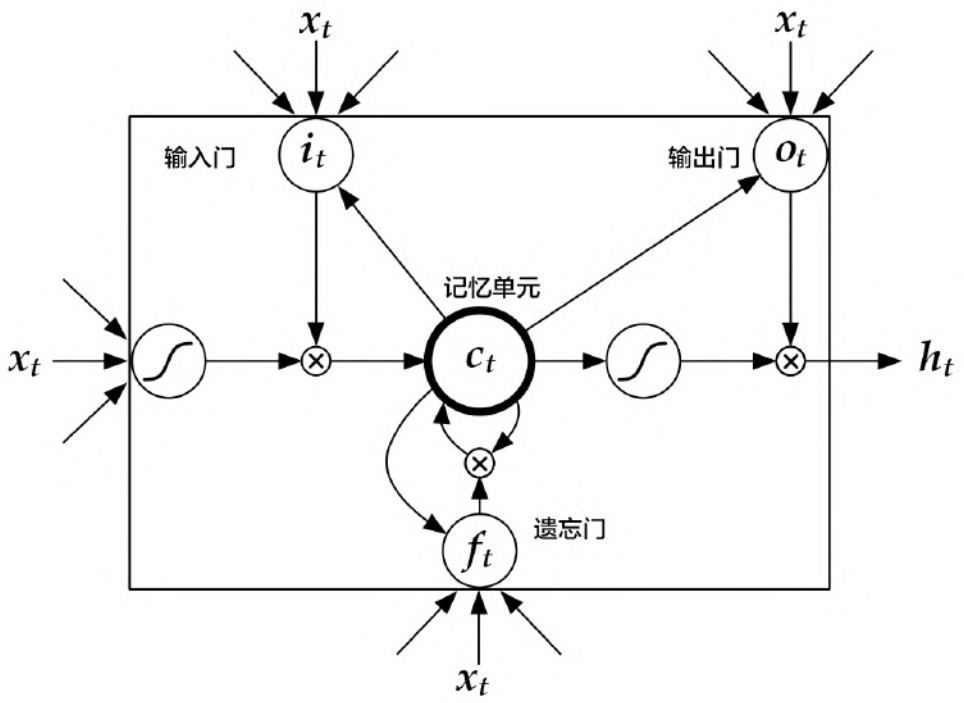

图3. 时刻t的LSTM [7]

LSTM通过给简单的循环神经网络增加记忆及控制门的方式，增强了其处理远距离依赖问题的能力。类似原理的改进还有Gated Recurrent Unit (GRU)[8]，其设计更为简洁一些。**这些改进虽然各有不同，但是它们的宏观描述却与简单的循环神经网络一样（如图2所示），即隐状态依据当前输入及前一时刻的隐状态来改变，不断地循环这一过程直至输入处理完毕：**

$$h_t = Recrurent(x_t, h_{t-1})$$

其中，$Recrurent$可以表示简单的循环神经网络、GRU或LSTM。

栈式双向LSTM（Stacked Bidirectional LSTM）

对于正常顺序的循环神经网络，h_t包含了t时刻之前的输入信息，也就是上文信息。同样，为了得到下文信息，我们可以使用反方向（将输入逆序处理）的循环神经网络。结合构建深层循环神经网络的方法（深层神经网络往往能得到更抽象和高级的特征表示），我们可以通过构建更加强有力的基于LSTM的栈式双向循环神经网络[9]，来对时序数据进行建模。

如图4所示（以三层为例），奇数层LSTM正向，偶数层LSTM反向，高一层的LSTM使用低一层LSTM及之前所有层的信息作为输入，对最高层LSTM序列使用时间维度上的最大池化即可得到文本的定长向量表示（这一表示充分融合了文本的上下文信息，并且对文本进行了深层次抽象），最后我们将文本表示连接至softmax构建分类模型。

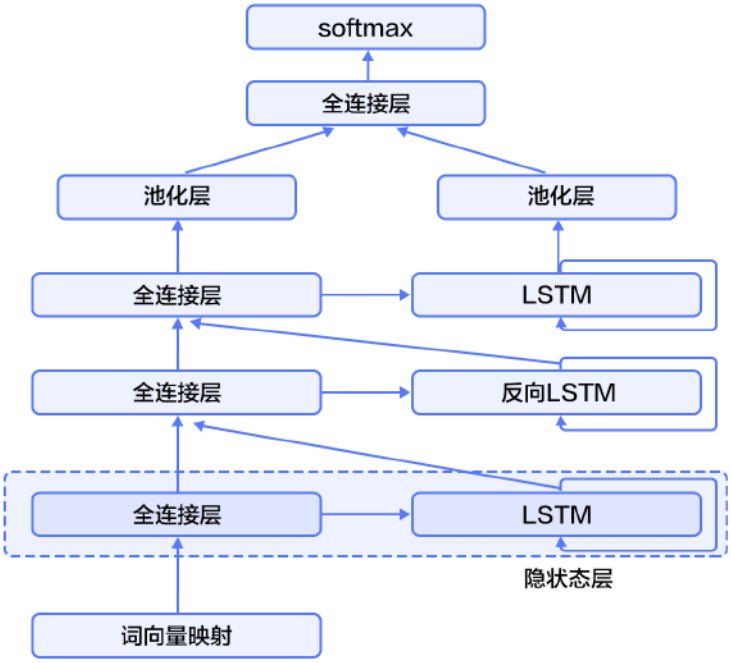

图4. 栈式双向LSTM用于文本分类

数据准备

数据介绍与下载

我们以IMDB情感分析数据集为例进行介绍。IMDB数据集的训练集和测试集分别包含25000个已标注过的电影评论。其中，负面评论的得分小于等于4，正面评论的得分大于等于7，满分10分。您可以使用下面的脚本下载IMDB 数据集和Moses工具：

```
./data/get_imdb.sh
```

如果数据获取成功，您将在目录`data`中看到下面的文件：

```
aclImdb  get_imdb.sh  imdb  mosesdecoder-master
```

- aclImdb: 从外部网站上下载的原始数据集。
- imdb: 仅包含训练和测试数据集。
- mosesdecoder-master: Moses 工具。

数据预处理

我们使用的预处理脚本为`preprocess.py`。该脚本会调用Moses工具中的`tokenizer.perl`脚本来切分单词和

标点符号，并会将训练集随机打乱排序再构建字典。注意：我们只使用已标注的训练集和测试集。执行下面的命令就可以预处理数据：

```
data_dir="./data/imdb"
python preprocess.py -i $data_dir
```

运行成功后目录`./data/pre-imdb`结构如下:

```
dict.txt  labels.list  test.list  test_part_000  train.list  train_part_000
```

- test_part_000 和 train_part_000: 所有标记的测试集和训练集，训练集已经随机打乱。
- train.list 和 test.list: 训练集和测试集文件列表。
- dict.txt: 利用训练集生成的字典。
- labels.list: 类别标签列表，标签0表示负面评论，标签1表示正面评论。

提供数据给PaddlePaddle

PaddlePaddle可以读取Python写的传输数据脚本，下面`dataprovider.py`文件给出了完整例子，主要包括两部分：

- hook： 定义文本信息、类别Id的数据类型。文本被定义为整数序列`integer_value_sequence`，类别被定义为整数`integer_value`。
- process： 按行读取以`'\t\t'`分隔的类别ID和文本信息，并用yield关键字返回。

```
from paddle.trainer.PyDataProvider2 import *

def hook(settings, dictionary, **kwargs):
    settings.word_dict = dictionary
    settings.input_types = {
        'word':  integer_value_sequence(len(settings.word_dict)),
        'label': integer_value(2)
    }
    settings.logger.info('dict len : %d' % (len(settings.word_dict)))


@provider(init_hook=hook)
def process(settings, file_name):
    with open(file_name, 'r') as fdata:
        for line_count, line in enumerate(fdata):
            label, comment = line.strip().split('\t\t')
            label = int(label)
            words = comment.split()
            word_slot = [
                settings.word_dict[w] for w in words if w in settings.word_dict
            ]
```

```
            yield {
                'word': word_slot,
                'label': label
            }
```

模型配置说明

`trainer_config.py` 是一个配置文件的例子。

数据定义

```
from os.path import join as join_path
from paddle.trainer_config_helpers import *
# 是否是测试模式
is_test = get_config_arg('is_test', bool, False)
# 是否是预测模式
is_predict = get_config_arg('is_predict', bool, False)

# 数据路径
data_dir = "./data/pre-imdb"
# 文件名
train_list = "train.list"
test_list = "test.list"
dict_file = "dict.txt"

# 字典大小
dict_dim = len(open(join_path(data_dir, "dict.txt")).readlines())
# 类别个数
class_dim = len(open(join_path(data_dir, 'labels.list')).readlines())

if not is_predict:
    train_list = join_path(data_dir, train_list)
    test_list = join_path(data_dir, test_list)
    dict_file = join_path(data_dir, dict_file)
    train_list = train_list if not is_test else None
    # 构造字典
    word_dict = dict()
    with open(dict_file, 'r') as f:
        for i, line in enumerate(open(dict_file, 'r')):
            word_dict[line.split('\t')[0]] = i
    # 通过define_py_data_sources2函数从dataprovider.py中读取数据
    define_py_data_sources2(
        train_list,
        test_list,
        module="dataprovider",
        obj="process",  # 指定生成数据的函数。
        args={'dictionary': word_dict}) # 额外的参数，这里指定词典。
```

算法配置

```
settings(
    batch_size=128,
    learning_rate=2e-3,
    learning_method=AdamOptimizer(),
    regularization=L2Regularization(8e-4),
    gradient_clipping_threshold=25)
```

- 设置batch size大小为128。
- 设置全局学习率。
- 使用adam优化。
- 设置L2正则。
- 设置梯度截断（clipping）阈值。

模型结构

我们用PaddlePaddle实现了两种文本分类算法，分别基于上文所述的文本卷积神经网络和[栈式双向LSTM](#栈式双向LSTM（Stacked Bidirectional LSTM）)。

文本卷积神经网络的实现

```
def convolution_net(input_dim,
                    class_dim=2,
                    emb_dim=128,
                    hid_dim=128,
                    is_predict=False):
    # 网络输入：id表示的词序列，词典大小为input_dim
    data = data_layer("word", input_dim)
    # 将id表示的词序列映射为embedding序列
    emb = embedding_layer(input=data, size=emb_dim)
    # 卷积及最大化池操作，卷积核窗口大小为3
    conv_3 = sequence_conv_pool(input=emb, context_len=3, hidden_size=hid_dim)
    # 卷积及最大化池操作，卷积核窗口大小为4
    conv_4 = sequence_conv_pool(input=emb, context_len=4, hidden_size=hid_dim)
    # 将conv_3和conv_4拼接起来输入给softmax分类，类别数为class_dim
    output = fc_layer(
        input=[conv_3, conv_4], size=class_dim, act=SoftmaxActivation())

    if not is_predict:
        lbl = data_layer("label", 1)     #网络输入：类别标签
        outputs(classification_cost(input=output, label=lbl))
    else:
        outputs(output)
```

其中，我们仅用一个`sequence_conv_pool`方法就实现了卷积和池化操作，卷积核的数量为hidden_size参数。

栈式双向LSTM的实现

```
def stacked_lstm_net(input_dim,
                     class_dim=2,
                     emb_dim=128,
                     hid_dim=512,
                     stacked_num=3,
                     is_predict=False):

    # LSTM的层数stacked_num为奇数，确保最高层LSTM正向
    assert stacked_num % 2 == 1
    # 设置神经网络层的属性
    layer_attr = ExtraLayerAttribute(drop_rate=0.5)
    # 设置参数的属性
    fc_para_attr = ParameterAttribute(learning_rate=1e-3)
    lstm_para_attr = ParameterAttribute(initial_std=0., learning_rate=1.)
    para_attr = [fc_para_attr, lstm_para_attr]
    bias_attr = ParameterAttribute(initial_std=0., l2_rate=0.)
    # 激活函数
    relu = ReluActivation()
    linear = LinearActivation()


    # 网络输入：id表示的词序列，词典大小为input_dim
    data = data_layer("word", input_dim)
    # 将id表示的词序列映射为embedding序列
    emb = embedding_layer(input=data, size=emb_dim)

    fc1 = fc_layer(input=emb, size=hid_dim, act=linear, bias_attr=bias_attr)
    # 基于LSTM的循环神经网络
    lstm1 = lstmemory(
        input=fc1, act=relu, bias_attr=bias_attr, layer_attr=layer_attr)

    # 由fc_layer和lstmemory构建深度为stacked_num的栈式双向LSTM
    inputs = [fc1, lstm1]
    for i in range(2, stacked_num + 1):
        fc = fc_layer(
            input=inputs,
            size=hid_dim,
            act=linear,
            param_attr=para_attr,
            bias_attr=bias_attr)
        lstm = lstmemory(
            input=fc,
            # 奇数层正向，偶数层反向。
            reverse=(i % 2) == 0,
            act=relu,
            bias_attr=bias_attr,
            layer_attr=layer_attr)
        inputs = [fc, lstm]

    # 对最后一层fc_layer使用时间维度上的最大池化得到定长向量
    fc_last = pooling_layer(input=inputs[0], pooling_type=MaxPooling())
```

```
    # 对最后一层lstmemory使用时间维度上的最大池化得到定长向量
    lstm_last = pooling_layer(input=inputs[1], pooling_type=MaxPooling())
    # 将fc_last和lstm_last拼接起来输入给softmax分类，类别数为class_dim
    output = fc_layer(
        input=[fc_last, lstm_last],
        size=class_dim,
        act=SoftmaxActivation(),
        bias_attr=bias_attr,
        param_attr=para_attr)

    if is_predict:
        outputs(output)
    else:
        outputs(classification_cost(input=output, label=data_layer('label', 1)))
```

我们的模型配置`trainer_config.py`默认使用`stacked_lstm_net`网络，如果要使用`convolution_net`，注释相应的行即可。

```
stacked_lstm_net(
    dict_dim, class_dim=class_dim, stacked_num=3, is_predict=is_predict)
# convolution_net(dict_dim, class_dim=class_dim, is_predict=is_predict)
```

训练模型

使用`train.sh`脚本可以开启本地的训练：

```
./train.sh
```

train.sh内容如下:

```
paddle train --config=trainer_config.py \
             --save_dir=./model_output \
             --job=train \
             --use_gpu=false \
             --trainer_count=4 \
             --num_passes=10 \
             --log_period=20 \
             --dot_period=20 \
             --show_parameter_stats_period=100 \
             --test_all_data_in_one_period=1 \
             2>&1 | tee 'train.log'
```

- --config=trainer_config.py: 设置模型配置。
- --save_dir=./model_output: 设置输出路径以保存训练完成的模型。
- --job=train: 设置工作模式为训练。

- --use_gpu=false: 使用CPU训练，如果您安装GPU版本的PaddlePaddle，并想使用GPU来训练可将此设置为true。
- --trainer_count=4:设置线程数（或GPU个数）。
- --num_passes=15: 设置pass，PaddlePaddle中的一个pass意味着对数据集中的所有样本进行一次训练。
- --log_period=20: 每20个batch打印一次日志。
- --show_parameter_stats_period=100: 每100个batch打印一次统计信息。
- --test_all_data_in_one_period=1: 每次测试都测试所有数据。

如果运行成功，输出日志保存在`train.log`中，模型保存在目录`model_output/`中。 输出日志说明如下：

```
Batch=20 samples=2560 AvgCost=0.681644 CurrentCost=0.681644 Eval:
classification_error_evaluator=0.36875  CurrentEval: classification_error_evaluator=
0.36875
...
Pass=0 Batch=196 samples=25000 AvgCost=0.418964 Eval: classification_error_evaluator
=0.1922
Test samples=24999 cost=0.39297 Eval: classification_error_evaluator=0.149406
```

- Batch=xx: 表示训练了xx个Batch。
- samples=xx: 表示训练了xx个样本。
- AvgCost=xx: 从第0个batch到当前batch的平均损失。
- CurrentCost=xx: 最新log_period个batch的损失。
- Eval: classification_error_evaluator=xx: 表示第0个batch到当前batch的分类错误。
- CurrentEval: classification_error_evaluator: 最新log_period个batch的分类错误。
- Pass=0: 通过所有训练集一次称为一个Pass。 0表示第一次经过训练集。

应用模型

测试

测试是指使用训练出的模型评估已标记的数据集。

```
./test.sh
```

测试脚本`test.sh`的内容如下，其中函数`get_best_pass`通过对分类错误率进行排序来获得最佳模型：

```
function get_best_pass() {
  cat $1  | grep -Pzo 'Test .*\n.*pass-.*' | \
  sed  -r 'N;s/Test.* error=([0-9]+\.[0-9]+).*\n.*pass-([0-9]+)/\1 \2/g' | \
  sort | head -n 1
}

```

```
log=train.log
LOG=`get_best_pass $log`
LOG=(${LOG})
evaluate_pass="model_output/pass-${LOG[1]}"

echo 'evaluating from pass '$evaluate_pass

model_list=./model.list
touch $model_list | echo $evaluate_pass > $model_list
net_conf=trainer_config.py
paddle train --config=$net_conf \
             --model_list=$model_list \
             --job=test \
             --use_gpu=false \
             --trainer_count=4 \
             --config_args=is_test=1 \
             2>&1 | tee 'test.log'
```

与训练不同，测试时需要指定`--job = test`和模型路径`--model_list = $model_list`。如果测试成功，日志将保存在`test.log`中。 在我们的测试中，最好的模型是`model_output/pass-00002`，分类错误率是0.115645：

```
Pass=0 samples=24999 AvgCost=0.280471 Eval: classification_error_evaluator=0.115645
```

预测

`predict.py`脚本提供了一个预测接口。预测IMDB中未标记评论的示例如下：

```
./predict.sh
```

predict.sh的内容如下（注意应该确保默认模型路径`model_output/pass-00002`存在或更改为其它模型路径）：

```
model=model_output/pass-00002/
config=trainer_config.py
label=data/pre-imdb/labels.list
cat ./data/aclImdb/test/pos/10007_10.txt | python predict.py \
     --tconf=$config \
     --model=$model \
     --label=$label \
     --dict=./data/pre-imdb/dict.txt \
     --batch_size=1
```

- `cat ./data/aclImdb/test/pos/10007_10.txt`：输入预测样本。
- `predict.py`：预测接口脚本。
- `--tconf=$config`：设置网络配置。

- `--model=$model`：设置模型路径。
- `--label=$label`：设置标签类别字典，这个字典是整数标签和字符串标签的一个对应。
- `--dict=data/pre-imdb/dict.txt`：设置文本数据字典文件。
- `--batch_size=1`：预测时的batch size大小。

本示例的预测结果：

```
Loading parameters from model_output/pass-00002/
predicting label is pos
```

`10007_10.txt`在路径`./data/aclImdb/test/pos`下面，而这里预测的标签也是pos，说明预测正确。

总结

本章我们以情感分析为例，介绍了使用深度学习的方法进行端对端的短文本分类，并且使用PaddlePaddle完成了全部相关实验。同时，我们简要介绍了两种文本处理模型：卷积神经网络和循环神经网络。在后续的章节中我们会看到这两种基本的深度学习模型在其它任务上的应用。

参考文献

1. Kim Y. Convolutional neural networks for sentence classification[J]. arXiv preprint arXiv:1408.5882, 2014.
2. Kalchbrenner N, Grefenstette E, Blunsom P. A convolutional neural network for modelling sentences[J]. arXiv preprint arXiv:1404.2188, 2014.
3. Yann N. Dauphin, et al. Language Modeling with Gated Convolutional Networks[J] arXiv preprint arXiv:1612.08083, 2016.
4. Siegelmann H T, Sontag E D. On the computational power of neural nets[C]//Proceedings of the fifth annual workshop on Computational learning theory. ACM, 1992: 440-449.
5. Hochreiter S, Schmidhuber J. Long short-term memory[J]. Neural computation, 1997, 9(8): 1735-1780.
6. Bengio Y, Simard P, Frasconi P. Learning long-term dependencies with gradient descent is difficult[J]. IEEE transactions on neural networks, 1994, 5(2): 157-166.
7. Graves A. Generating sequences with recurrent neural networks[J]. arXiv preprint arXiv:1308.0850, 2013.
8. Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation[J]. arXiv preprint arXiv:1406.1078, 2014.
9. Zhou J, Xu W. End-to-end learning of semantic role labeling using recurrent neural networks[C]//Proceedings of the Annual Meeting of the Association for Computational Linguistics. 2015.