情感分析

本教程源代码目录在book/understand_sentiment,初次使用请您参考Book文档使用说明。

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

在自然语言处理中，情感分析属于典型的**文本分类**问题，即把需要进行情感分析的文本划分为其所属类别。文本分类涉及文本表示和分类方法两个问题。在深度学习的方法出现之前，主流的文本表示方法为词袋模型BOW(bag of words)，话题模型等等；分类方法有SVM(support vector machine), LR(logistic regression)等等。

对于一段文本，BOW表示会忽略其词顺序、语法和句法，将这段文本仅仅看做是一个词集合，因此BOW方法并不能充分表示文本的语义信息。例如，句子"这部电影糟糕透了"和"一个乏味，空洞，没有内涵的作品"在情感分析中具有很高的语义相似度，但是它们的BOW表示的相似度为0。又如，句子"一个空洞，没有内涵的作品"和"一个不空洞而且有内涵的作品"的BOW相似度很高，但实际上它们的意思很不一样。

本章我们所要介绍的深度学习模型克服了BOW表示的上述缺陷，它在考虑词顺序的基础上把文本映射到低维度的语义空间，并且以端对端（end to end）的方式进行文本表示及分类，其性能相对于传统方法有显著的提升[1]。

说明:

1. 硬件环境要求： 本文可支持在CPU、GPU下运行
2. Docker镜像支持的CUDA/cuDNN版本： 如果使用了Docker运行Book，请注意：这里所提供的默认镜像的GPU环境为 CUDA 8/cuDNN 5，对于NVIDIA Tesla V100等要求CUDA 9的 GPU，使用该镜像可能会运行失败。
3. 文档和脚本中代码的一致性问题： 请注意：为使本文更加易读易用，我们拆分、调整了train.py的代码并放入本文。本文中代码与train.py的运行结果一致，可直接运行train.py进行验证。

模型概览

本章所使用的文本表示模型为卷积神经网络（Convolutional Neural Networks）和循环神经网络(Recurrent Neural Networks)及其扩展。下面依次介绍这几个模型。

对卷积神经网络来说，首先使用卷积处理输入的词向量序列，产生一个特征图（feature map），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，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。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，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，推荐系统一节的图3作为示意画了四个卷积核，既文本图1，不同颜色表示不同大小的卷积核操作。

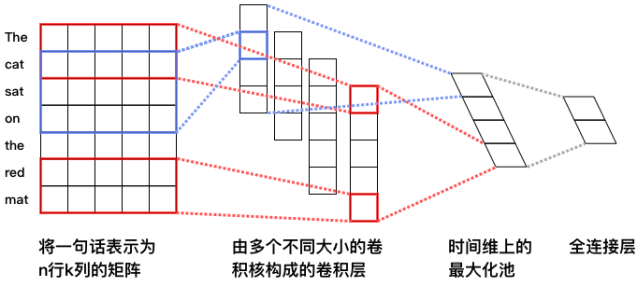

图1. 卷积神经网络文本分类模型

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。

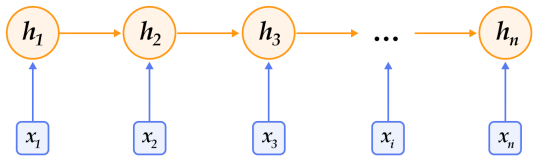

图2. 循环神经网络按时间展开的示意图

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：

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

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：

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

F由下列公式组合而成[7]：

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o)$$

$$h_t = o_t \odot tanh(c_t)$$

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：

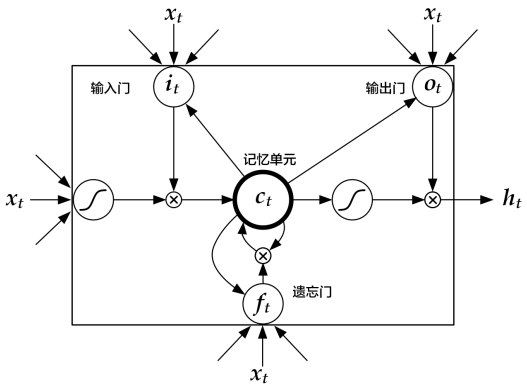

图3. 时刻t的LSTM [7]

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：**

$$h_t = Recurrent\left(x_t, h_{\{t-1\}}\right)$$

其中，$Recurrent$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。

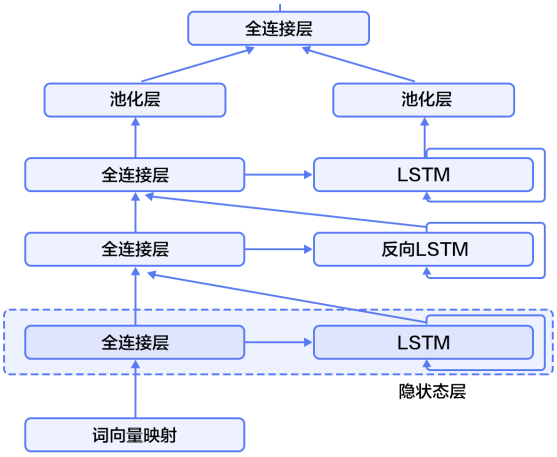

图4. 栈式双向LSTM用于文本分类

数据集介绍

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。

```
aclImdb
|- test
   |-- neg
   |-- pos
|- train
   |-- neg
   |-- pos
```

Paddle在 `dataset/imdb.py` 中提实现了imdb数据集的自动下载和读取，并提供了读取字典、训练数据、测试数据等API。

配置模型

在该示例中，我们实现了两种文本分类算法，分别基于推荐系统一节介绍过的文本卷积神经网络，以及[栈式双向LSTM](#栈式双向LSTM（Stacked Bidirectional LSTM））。我们首先引入要用到的库和定义全局变量：

```
from __future__ import print_function
import paddle
import paddle.fluid as fluid
import numpy as np
import sys
import math

CLASS_DIM = 2      #情感分类的类别数
EMB_DIM = 128      #词向量的维度
HID_DIM = 512      #隐藏层的维度
STACKED_NUM = 3    #LSTM双向栈的层数
BATCH_SIZE = 128   #batch的大小
```

文本卷积神经网络

我们构建神经网络 `convolution_net` ，示例代码如下。 需要注意的是： `fluid.nets.sequence_conv_pool` 包含卷积和池化层两个操作。

```
#文本卷积神经网络
def convolution_net(data, input_dim, class_dim, emb_dim, hid_dim):
    emb = fluid.layers.embedding(
        input=data, size=[input_dim, emb_dim], is_sparse=True)
    conv_3 = fluid.nets.sequence_conv_pool(
        input=emb,
        num_filters=hid_dim,
        filter_size=3,
        act="tanh",
        pool_type="sqrt")
    conv_4 = fluid.nets.sequence_conv_pool(
        input=emb,
```

```
        pool_type="sqrt")
    prediction = fluid.layers.fc(
        input=[conv_3, conv_4], size=class_dim, act="softmax")
    return prediction
```

网络的输入 input_dim 表示的是词典的大小， class_dim 表示类别数。这里，我们使用 sequence_conv_pool API实现了卷积和池化操作。

栈式双向LSTM

栈式双向神经网络 stacked_lstm_net 的代码片段如下：

```
#栈式双向LSTM
def stacked_lstm_net(data, input_dim, class_dim, emb_dim, hid_dim, stacked_num):

    #计算词向量
    emb = fluid.layers.embedding(
        input=data, size=[input_dim, emb_dim], is_sparse=True)

    #第一层栈
    #全连接层
    fc1 = fluid.layers.fc(input=emb, size=hid_dim)
    #lstm层
    lstm1, cell1 = fluid.layers.dynamic_lstm(input=fc1, size=hid_dim)

    inputs = [fc1, lstm1]

    #其余的所有栈结构
    for i in range(2, stacked_num + 1):
        fc = fluid.layers.fc(input=inputs, size=hid_dim)
        lstm, cell = fluid.layers.dynamic_lstm(
            input=fc, size=hid_dim, is_reverse=(i % 2) == 0)
        inputs = [fc, lstm]

    #池化层
    fc_last = fluid.layers.sequence_pool(input=inputs[0], pool_type='max')
    lstm_last = fluid.layers.sequence_pool(input=inputs[1], pool_type='max')

    #全连接层，softmax预测
    prediction = fluid.layers.fc(
        input=[fc_last, lstm_last], size=class_dim, act='softmax')
    return prediction
```

以上的栈式双向LSTM抽象出了高级特征并把其映射到和分类类别数同样大小的向量上。最后一个全连接层的'softmax'激活函数用来计算分类属于某个类别的概率。

重申一下，此处我们可以调用 convolution_net 或 stacked_lstm_net 的任何一个网络结构进行训练学习。我们以 convolution_net 为例。

接下来我们定义预测程序（ inference_program ）。预测程序使用 convolution_net 来对 fluid.layer.data 的输入进行预测。

```
def inference_program(word_dict):
    data = fluid.layers.data(
        name="words", shape=[1], dtype="int64", lod_level=1)

    dict_dim = len(word_dict)
    net = convolution_net(data, dict_dim, CLASS_DIM, EMB_DIM, HID_DIM)
    # net = stacked_lstm_net(data, dict_dim, CLASS_DIM, EMB_DIM, HID_DIM, STACKED_NUM)
    return net
```

我们这里定义了 training_program 。它使用了从 inference_program 返回的结果来计算误差。我们同时定义了优化函数 optimizer_func 。

因为是有监督的学习，训练集的标签也在 fluid.layers.data 中定义了。在训练过程中，交叉熵用来在 fluid.layer.cross_entropy 中作为损失函数。

在测试过程中，分类器会计算各个输出的概率。第一个返回的数值规定为cost。

```
def train_program(prediction):
    label = fluid.layers.data(name="label", shape=[1], dtype="int64")
    cost = fluid.layers.cross_entropy(input=prediction, label=label)
    avg_cost = fluid.layers.mean(cost)
    accuracy = fluid.layers.accuracy(input=prediction, label=label)
    return [avg_cost, accuracy]   #返回平均cost和准确率acc
```

训练模型

定义训练环境

定义您的训练是在CPU上还是在GPU上：

```
use_cuda = False  #在cpu上进行训练
place = fluid.CUDAPlace(0) if use_cuda else fluid.CPUPlace()
```

定义数据提供器

下一步是为训练和测试定义数据提供器。提供器读入一个大小为 BATCH_SIZE的数据。paddle.dataset.imdb.word_dict 每次会在乱序化后提供一个大小为BATCH_SIZE的数据，乱序化的大小为缓存大小buf_size。

注意：读取IMDB的数据可能会花费几分钟的时间，请耐心等待。

```
print("Loading IMDB word dict....")
word_dict = paddle.dataset.imdb.word_dict()

print ("Reading training data....")
train_reader = paddle.batch(
    paddle.reader.shuffle(
        paddle.dataset.imdb.train(word_dict), buf_size=25000),
    batch_size=BATCH_SIZE)
print("Reading testing data....")
test_reader = paddle.batch(
    paddle.dataset.imdb.test(word_dict), batch_size=BATCH_SIZE)
```

word_dict是一个字典序列，是词和label的对应关系，运行下一行可以看到具体内容：

```
word_dict
```

每行是如（'limited': 1726）的对应关系，该行表示单词limited所对应的label是1726。

构造训练器

训练器需要一个训练程序和一个训练优化函数。

```
exe = fluid.Executor(place)
prediction = inference_program(word_dict)
[avg_cost, accuracy] = train_program(prediction)#训练程序
sgd_optimizer = optimizer_func()#训练优化函数
sgd_optimizer.minimize(avg_cost)
```

该函数用来计算训练中模型在test数据集上的结果

```
def train_test(program, reader):
    count = 0
    feed_var_list = [
        program.global_block().var(var_name) for var_name in feed_order
    ]
    feeder_test = fluid.DataFeeder(feed_list=feed_var_list, place=place)
    test_exe = fluid.Executor(place)
    accumulated = len([avg_cost, accuracy]) * [0]
    for test_data in reader():
        avg_cost_np = test_exe.run(
            program=program,
            feed=feeder_test.feed(test_data),
            fetch_list=[avg_cost, accuracy])
        accumulated = [
            x[0] + x[1][0] for x in zip(accumulated, avg_cost_np)
        ]
        count += 1
    return [x / count for x in accumulated]
```

`feed_order` 用来定义每条产生的数据和 `fluid.layers.data` 之间的映射关系。比如， `imdb.train` 产生的第一列的数据对应的是 `words` 这个特征。

```
# Specify the directory path to save the parameters
params_dirname = "understand_sentiment_conv.inference.model"

feed_order = ['words', 'label']
pass_num = 1  #训练循环的轮数

#程序主循环部分
def train_loop(main_program):
    #启动上文构建的训练器
    exe.run(fluid.default_startup_program())

    feed_var_list_loop = [
        main_program.global_block().var(var_name) for var_name in feed_order
    ]
    feeder = fluid.DataFeeder(
        feed_list=feed_var_list_loop, place=place)

    test_program = fluid.default_main_program().clone(for_test=True)

    #训练循环
    for epoch_id in range(pass_num):
        for step_id, data in enumerate(train_reader()):
            #运行训练器
            metrics = exe.run(main_program,
                              feed=feeder.feed(data),
                              fetch_list=[avg_cost, accuracy])

            #测试结果
            avg_cost_test, acc_test = train_test(test_program, test_reader)
            print('Step {0}, Test Loss {1:0.2}, Acc {2:0.2}'.format(
                step_id, avg_cost_test, acc_test))

            print("Step {0}, Epoch {1} Metrics {2}".format(
                step_id, epoch_id, list(map(np.array,
                                            metrics))))

            if step_id == 30:
                if params_dirname is not None:
                    fluid.io.save_inference_model(params_dirname, ["words"],
                                                  prediction, exe)#保存模型
                return
```

训练过程处理

我们在训练主循环里打印了每一步输出，可以观察训练情况。

开始训练

最后，我们启动训练主循环来开始训练。训练时间较长，如果为了更快的返回结果，可以通过调整损耗值范围或者训练步数，以减少准确率的代价来缩短训练时间。

```
train_loop(fluid.default_main_program())
```

应用模型

构建预测器

和训练过程一样，我们需要创建一个预测过程，并使用训练得到的模型和参数来进行预测， `params_dirname` 用来存放训练过程中的各个参数。

```
place = fluid.CUDAPlace(0) if use_cuda else fluid.CPUPlace()
exe = fluid.Executor(place)
inference_scope = fluid.core.Scope()
```

生成测试用输入数据

```
reviews_str = [
    'read the book forget the movie', 'this is a great movie', 'this is very bad'
]
reviews = [c.split() for c in reviews_str]

UNK = word_dict['<unk>']
lod = []
for c in reviews:
    lod.append([word_dict.get(words, UNK) for words in c])

base_shape = [[len(c) for c in lod]]

tensor_words = fluid.create_lod_tensor(lod, base_shape, place)
```

应用模型并进行预测

现在我们可以对每一条评论进行正面或者负面的预测啦。

```
with fluid.scope_guard(inference_scope):

    [inferencer, feed_target_names,
     fetch_targets] = fluid.io.load_inference_model(params_dirname, exe)

    assert feed_target_names[0] == "words"
    results = exe.run(inferencer,
                      feed={feed_target_names[0]: tensor_words},
                      fetch_list=fetch_targets,
                      return_numpy=False)
    np_data = np.array(results[0])
    for i, r in enumerate(np_data):
        print("Predict probability of ", r[0], " to be positive and ", r[1],
              " to be negative for review \'", reviews_str[i], "\'")
```

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.