本文主要内容

- 简略介绍卷积神经网络（CNN, Convolutional Neural Network）处理文本信息的过程
- 使用CNN进行文本分类任务，并对代码进行注释
- 本文代码【https://github.com/540117253/Chinese-Text-Classification 】

一、CNN概述

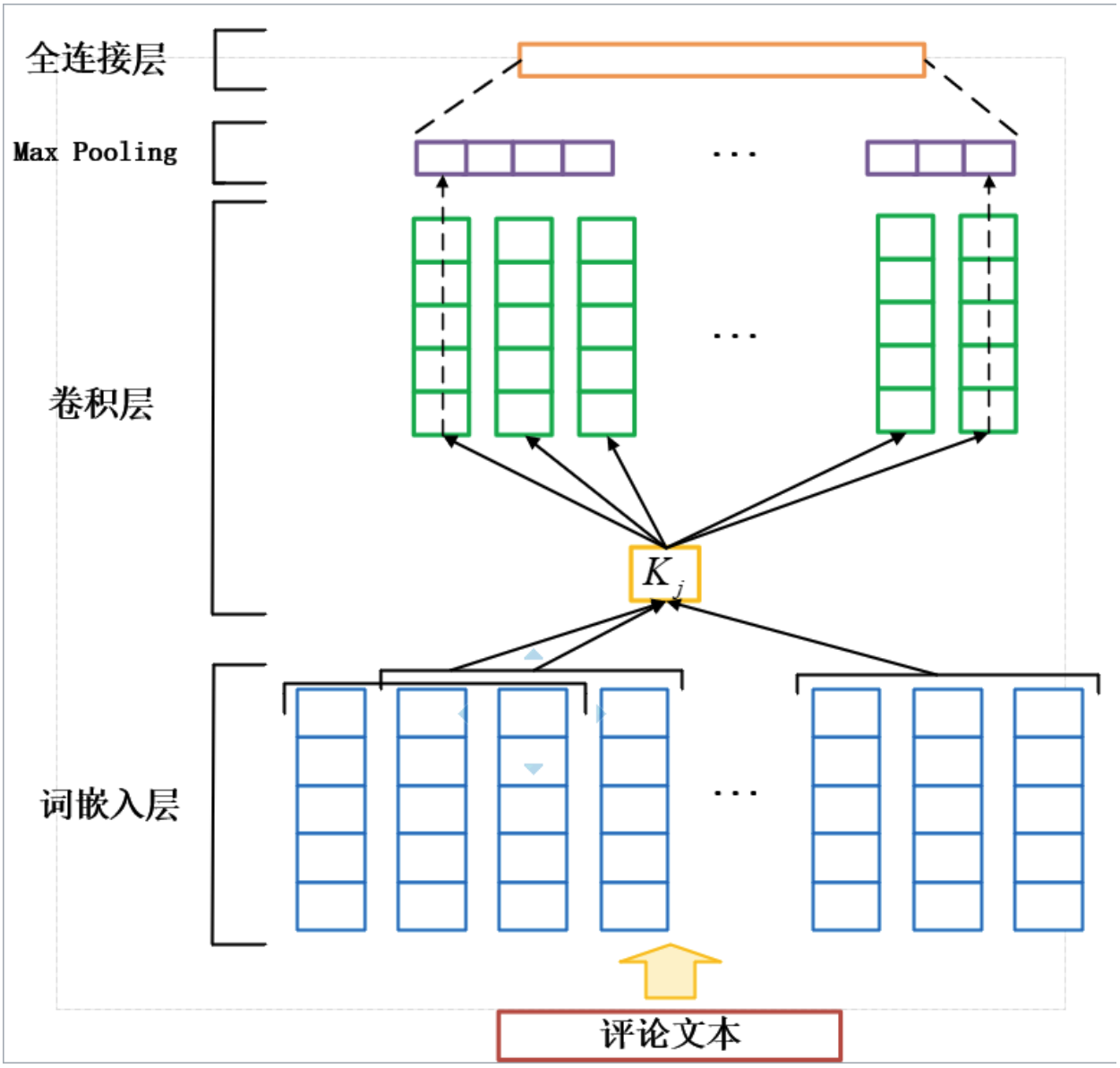

图1 CNN文本编码器

CNN文本编码器的结构如图1所示。在第一层，词映射函数 $f: M \rightarrow R^d$ 将评论的每个单词映射为 d 维向量，然后将给定的评论文本转化为长度固定为 T 的词嵌入矩阵（只截取评论文本中的前 T 个单词，如果长度不足的文本则进行填充处理）。

词映射层后的是卷积层，其包含 m 个神经元，每个神经元对应的卷积核 $K \in R^{t \times d}$ 用于对词向量进行卷积运算提取特征。假设 $V_{1:T}$ 是文本长度为 T 的词嵌入矩阵，第 j 个神经元产生的特征为：

$$z_j = ReLU(V_{1:T} * K_j + b_j)$$

其中 b_j 为偏倚项，$*$ 表示卷积运算，$ReLU$ 是非线性激活函数。

最终在滑动窗口 t 的作用下，第 j 个神经元产生的特征为 $z_1, z_2, ..., z_j^{T-t+1}$。将该特征进行max-pooling运算，其主要用于捕获拥有最大值的最重要的特征，其定义为：

$$o_j = max(z_1, z_2, ..., z_j^{T-t+1})$$

最后卷积层的最终输出为 m 个神经元输出的拼接结果，其定义为：

$$O = [o_1, o_2, ...o_m]$$

通常 O 会接着送入全连接层，其中包含权重矩阵 $W \in R^{m \times n}$ 和偏置项 $g \in R^n$，具体公式为：

$$X = ReLU(WO + g)$$

整体上，CNN的卷积核大小一般为3或者5（即一次卷积运算仅计算3个单词或5个单词的信息），其仅采用一个卷积核就能通过滑动窗口的方式来扫描整个文本，因此整个文本可以看作是共享同一个卷积核的一组参数，能很好地节省内存空间。然而一次卷积运算仅能包含卷积窗口内的单词，当输入的文本越长，卷积窗口滑动到文本的尾部时所丢失掉的前文信息就越多。因此对于文本数据，一般采用循环神经网络RNN，其比CNN的文本信息提取能力更优秀。

二、CNN文本分类实例

2.1 数据集介绍

1. 下载地址:

【https://github.com/skdjfla/toutiao-text-classfication-dataset 】

2. 格式:

```
6552431613437805063_!_102_!_news_entertainment_!_谢娜为李浩菲澄清网络谣言，之后她的两个行为给自己加分_!_佟丽娅,网络谣言,快乐大本营,李浩菲,谢娜,观众们
```

每行为一条数据，以_!_分割的个字段，从前往后分别是 新闻ID，分类code（见下文），分类名称（见下文），新闻字符串（仅含标题），新闻关键词

分类code与名称：

```
100 民生 故事 news_story
101 文化 文化 news_culture
102 娱乐 娱乐 news_entertainment
103 体育 体育 news_sports
104 财经 财经 news_finance
106 房产 房产 news_house
```

```
107 汽车 汽车 news_car
108 教育 教育 news_edu
109 科技 科技 news_tech
110 军事 军事 news_military
112 旅游 旅游 news_travel
113 国际 国际 news_world
114 证券 股票 stock
115 农业 三农 news_agriculture
116 电竞 游戏 news_game
```

2.2 预训练词向量

预训练词向量使用的是，基于ACL-2018模型在百度百科训练的词向量。

下载地址：【 https://github.com/Embedding/Chinese-Word-Vectors 】

2.3 数据预处理

1. 清除无用字符，并且进行分词处理
2. 建立整个数据集的字典,key=word, value=词语的编号
3. 对进行截断或补0处理，确保每条样本的长度为maxlen
4. 序列化样本的标签，例如“体育类新闻”的类别编号为1， “娱乐类新闻”的类别编号为2
5. 将处理好的数据转化为DataFrame格式，并保存到硬盘

2.4 CNN模型的定义

```
'''
    Text => CNN => Fully_Connected => Softmax

    参数:
    filter_sizes: 卷积核的大小
    num_filters: 卷积核的个数
    embedded_size: 词向量的维度
    dict_size: 数据集的单词个数
    maxlen: 每条样本的最大单词数
    label_num: 样本类别的数量
    learning_rate: 梯度优化器的初始学习率
'''
class CNN:
    def __init__(self, filter_sizes, num_filters, embedded_size,
                 dict_size, maxlen, label_num, learning_rate):

        # print('model_Name:', 'CNN')

        self.droput_rate = 0.5

        '''
            Convulutional Neural Network
        '''
        def cnn (input_emb, filter_sizes, num_filters):
            pooled_outputs = []
```

```
            for i, filter_size in enumerate(filter_sizes):
                filter_shape = [filter_size, embedded_size, 1, num_filters]
                W = tf.Variable(tf.truncated_normal(filter_shape, stddev=0.1), name="W")
                b = tf.Variable(tf.constant(0.1, shape=[num_filters]), name="b")
                conv = tf.nn.conv2d(
                    input_emb,
                    W,
                    strides=[1, 1, 1, 1],
                    padding="VALID",
                    name="conv") # shape(conv) = [None, sequence_length - filter_size +
1, 1, num_filters]
                h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
                word_num = input_emb.shape.as_list()[1]
                pooled = tf.nn.max_pool(
                    h,
                    ksize=[1, word_num - filter_size + 1, 1, 1],
                    strides=[1, 1, 1, 1],
                    padding='VALID',
                    name="pool") # shape(pooled) = [None, 1, 1, num_filters]
                pooled_outputs.append(pooled)
            num_filters_total = num_filters * len(filter_sizes)

            h_pool = tf.concat(pooled_outputs,3)

            h_pool_flat = tf.reshape(h_pool, [-1, num_filters_total])  # shape =
[None,num_filters_total]

            cnn_fea = tf.nn.dropout(h_pool_flat, keep_prob =  self.droput_rate)

            return cnn_fea

        self.X = tf.placeholder(tf.int32, [None, maxlen], name='input_x')
        self.Y = tf.placeholder(tf.int64, [None])

        self.encoder_embeddings = tf.Variable(tf.random_uniform([dict_size,
embedded_size], -1, 1), trainable=False)
        encoder_embedded = tf.nn.embedding_lookup(self.encoder_embeddings, self.X)

        # 由于conv2d需要一个四维的输入数据，因此需要手动添加一个维度。
        encoder_embedded = tf.expand_dims(encoder_embedded, -1) # shape(encoder_embedded)
= [None, user_review_num*u_n_words, embedding_size, 1]

        outputs = cnn(input_emb = encoder_embedded, filter_sizes = filter_sizes,
num_filters = num_filters)

        self.logits = keras.layers.Dense(label_num, use_bias=True)(outputs)
        self.probability = tf.nn.softmax(self.logits, name='probability')

        self.cost = tf.nn.sparse_softmax_cross_entropy_with_logits(
                                                                labels = self.Y,
                                                                logits = self.logits)
        self.cost = tf.reduce_mean(self.cost)
        self.optimizer = tf.train.AdamOptimizer(learning_rate =
learning_rate).minimize(self.cost)
        self.pre_y = tf.argmax(self.logits, 1, name='pre_y')
        correct_pred = tf.equal(self.pre_y, self.Y)
        self.accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

2.5 训练模型

1. 将预处理好的数据集切分为80%的训练集，10%作为验证集，10%作为测试集
2. 每使用一次训练集进行训练后，就使用验证集进行测试。
3. 当验证集的准确率连续下降5次，就停止步骤2，然后使用测试集的结果作为模型的最终性能。