基于 Convnet 的数据序列匹配实现

1 引言

本项目是在自然语言处理（NLP）的背景下衍生出的文本匹配的一种，本文提出了基于 Convnet（卷积神经网络，也称 CNN）的数据序列的匹配算法，训练测试结果在 80%左右。

2 算法原理介绍

卷积神经网络（CNN，有时被称为 ConvNet）是很吸引人的。在短时间内，它们变成了一种颠覆性的技术，打破了从文本、视频到语音等多个领域所有最先进的算法，远远超出了其最初在图像处理的应用范围。CNN 由许多神经网络层组成。卷积和池化这两种不同类型的层通常是交替的。网络中每个滤波器的深度从左到右增加。最后通常由一个或多个全连接的层组成：

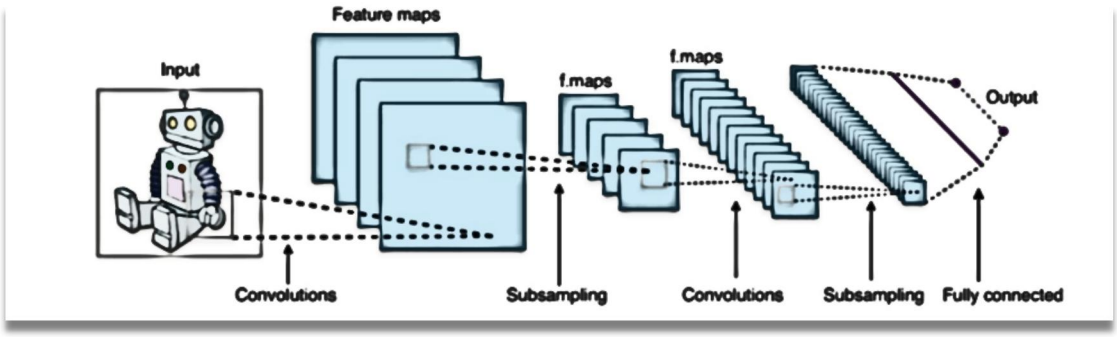

图 1 Convnet 结构示意图

Convnets 背后有三个关键动机：局部感受野、共享权重和池化。这里我主要讲一下共享权重和池化。

2.1 共享权重

假设想要从原始像素表示中获得移除与输入图像中位置信息无关的相同特征的能力。一个简单的直觉就是对隐藏层中的所有神经元使用相同的权重和偏置。通过这种方式，每层将从图像中学习到独立于位置信息的潜在特征。

下面给出卷积运算的例子：

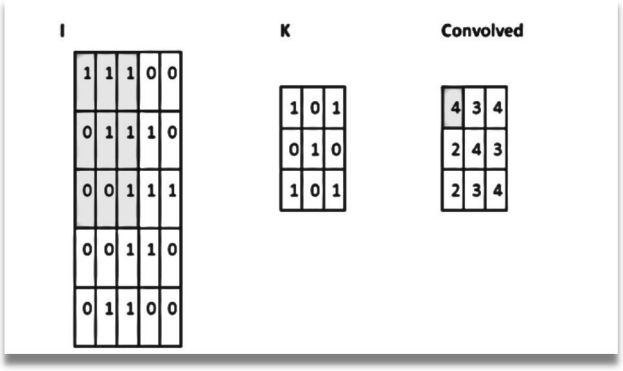

图 2 卷积运算

给定输入矩阵 I 和核 K，得到卷积输出。

2.2 池化

假设我们要总结一个特征映射的输出。我们可以使用从单个特征映射产生的输出的空间邻接性，并将子矩阵的值聚合成单个输出值，从而合成地描述与该物理区域相关联的含义。池化分为最大池化和最小池化以及均匀池化三种，本项目采用最大池化，也就是选择观察区域中的最大值作为输出：

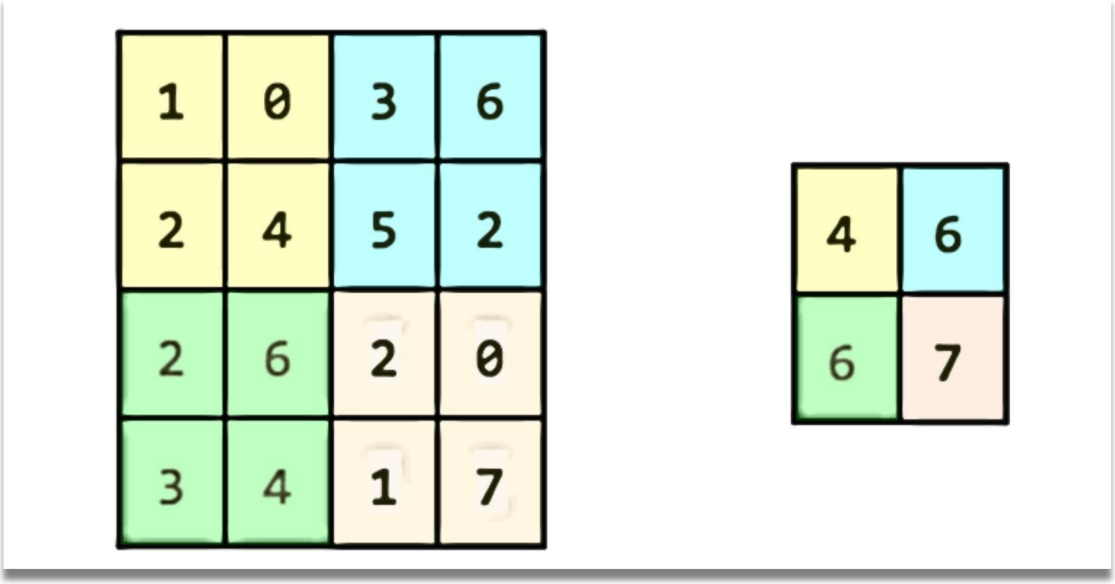

图 3 最大池化示意

3 代码思路详解

Step1：训练词向量

这里采用 python 自带的 gensim 库的 Word2vec 训练我们的数据序列生成静态的词向量：

```
df = pd.read_csv('input/data/train.csv')
p = df['sentence1'].values
h = df['sentence2'].values
p_seg = list(map(lambda x: list(jieba.cut(x)), p))
h_seg = list(map(lambda x: list(jieba.cut(x)), h))
common_texts = []
common_texts.extend(p_seg)
common_texts.extend(h_seg)
```

这里采用 jieba 模块进行切词。

```
model = Word2Vec(common_texts, size=args.char_embedding_size, window=5, min_count=5, workers=12)

model.save("output/word2vec/word2vec.model")
```

这里我们将低频词进行过滤，这里的阈值我设置为 5，表示出现次数小于 5 的则去掉。最后保存词向量的模型即可。

Step2：搭建 Convnet 网络结构

这里采用的是 tensorflow 深度学习的框架搭建。主要由输入层、卷积层、池化层、输出层构成：

```
class Graph:

    def __init__(self):
        self.p = tf.placeholder(dtype=tf.int32, shape=(None, args.seq_length), name='p')
        self.h = tf.placeholder(dtype=tf.int32, shape=(None, args.seq_length), name='h')
        self.y = tf.placeholder(dtype=tf.int32, shape=None, name='y')
        self.keep_prob = tf.placeholder(dtype=tf.float32, name='drop_rate')

        self.embedding = tf.get_variable(dtype=tf.float32, shape=(args.vocab_size, args.char_embedding_size),
                                         name='embedding')
        self.M = tf.Variable(tf.random_normal(shape=(args.n_filter, args.n_filter), mean=0, stddev=1),
                             name='M',
                             dtype=tf.float32)
        self.x_feat = tf.Variable(tf.random_normal(shape=(args.batch_size, args.n_filter), mean=0, stddev=1),
                                  name='x_feat',
                                  dtype=tf.float32)
```

这里的输入主要由 sentence1，sentence2 和 label，定义向前传播，更新权重：

```
def forward(self):
    p_embedding = tf.nn.embedding_lookup(self.embedding, self.p)
    h_embedding = tf.nn.embedding_lookup(self.embedding, self.h)

    p_embedding = tf.expand_dims(p_embedding, axis=3)
    h_embedding = tf.expand_dims(h_embedding, axis=3)
    p = tf.layers.conv2d(p_embedding,
                         filters=args.n_filter,
                         kernel_size=(args.filter_width, args.filter_height),
                         activation='relu')
    h = tf.layers.conv2d(h_embedding,
                         filters=args.n_filter,
                         kernel_size=(args.filter_width, args.filter_height),
                         activation='relu')

    pool_width = args.seq_length + 1 - args.filter_width
    p_max = tf.layers.max_pooling2d(p, pool_size=(pool_width, 1), strides=1)
    h_max = tf.layers.max_pooling2d(h, pool_size=(pool_width, 1), strides=1)
```

对每一个输入进行卷积运算和池化运算。定义训练输出：

```
def train(self, logits):
    y = tf.one_hot(self.y, args.class_size)
    loss = tf.nn.softmax_cross_entropy_with_logits(labels=y, logits=logits)
    self.loss = tf.reduce_mean(loss)
    self.train_op = tf.train.AdamOptimizer(args.learning_rate).minimize(self.loss)
    prediction = tf.argmax(logits, axis=1)
    correct_prediction = tf.equal(tf.cast(prediction, tf.int32), self.y)
    self.acc = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    self.label =prediction
```

网络结构搭建完毕，开始训练参数，生成模型文件：

Step3：训练网络

```
p, h, y = load_char_data('input/data/train.csv', data_size=None)
p_eval, h_eval, y_eval = load_char_data('input/data/dev.csv', data_size=args.batch_size)

p_holder = tf.placeholder(dtype=tf.int32, shape=(None, args.seq_length), name='p')
h_holder = tf.placeholder(dtype=tf.int32, shape=(None, args.seq_length), name='h')
y_holder = tf.placeholder(dtype=tf.int32, shape=None, name='y')

dataset = tf.data.Dataset.from_tensor_slices((p_holder, h_holder, y_holder))
dataset = dataset.batch(args.batch_size).repeat(args.epochs)
iterator = dataset.make_initializable_iterator()
next_element = iterator.get_next()

model = Graph()
saver = tf.train.Saver()

config = tf.ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.per_process_gpu_memory_fraction = 0.9

with tf.Session(config=config)as sess:
    sess.run(tf.global_variables_initializer())
    sess.run(iterator.initializer, feed_dict={p_holder: p, h_holder: h, y_holder: y})
    steps = int(len(y) / args.batch_size)
    for epoch in range(args.epochs):
        for step in range(steps):
            p_batch, h_batch, y_batch = sess.run(next_element)
            _, loss, acc = sess.run([model.train_op, model.loss, model.acc],
                                    feed_dict={model.p: p_batch,
                                               model.h: h_batch,
                                               model.y: y_batch,
                                               model.keep_prob: args.keep_prob})
            print('epoch:', epoch, ' step:', step, ' loss: ', loss, ' acc:', acc)

        loss_eval, acc_eval = sess.run([model.loss, model.acc],
                                       feed_dict={model.p: p_eval,
```

这里 epoch，设置为 30，batchsize 设置为 1240，句子截取最大长度设置为 50，具体参数如下：

```
seq_length = 50

char_embedding_size = 100

learning_rate = 0.0005

keep_prob = 0.5

vocab_size = 51164

class_size = 2

epochs = 30

batch_size = 1240

filter_width = 3
filter_height = char_embedding_size
n_filter = 32
```

最后保存模型即可。

Step4:测试模型

加载测试数据集，加载上一步保存的模型，用来评估模型：

```
sys.path.append(os.path.join(os.path.dirname(__file__), '../'))

from convnet.graph import Graph
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
from utils.load_data import load_char_data

os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"

os.environ['CUDA_VISIBLE_DEVICES'] = '0,1'

p, h, y = load_char_data('input/data/test.csv', data_size=None)

model = Graph()
saver = tf.train.Saver()

with tf.Session()as sess:
    sess.run(tf.global_variables_initializer())
    saver.restore(sess, '../output/convnet/convnet_29.ckpt')
    loss, acc, label= sess.run([model.loss, model.acc,model.label],
                        feed_dict={model.p: p,
                                   model.h: h,
                                   model.y:y,
                                   model.keep_prob: 1})
    print('loss: ', loss, ' acc:', acc,'label:',label)
```

Step5:预测新数据集

```
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
os.environ['CUDA_VISIBLE_DEVICES'] = '0,1'
def predict():
 p, h, y = load_char_data('input/data/test.csv', data_size=None)
 model =Graph()
 saver = tf.train.Saver()
 with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    saver.restore(sess, 'output/convent/convnet_29.ckpt')
    label = sess.run(model.label,feed_dict={model.p:p,model.h:h,model.keep_prob:1})
    print(label)
    numpy.savetxt("predict.txt", label,fmt='%u')
```

生成 predict.txt 文件

4 运行环境说明

Python3.7 ;tensorflow2.0

需要导入的模块见 import

5 提交代码说明

1. graph.py 网络结构模型

2. train.py 训练网络

3. args.py 参数设置

4. test.py 测试模型

5. predict.py 预测