TensorFlow

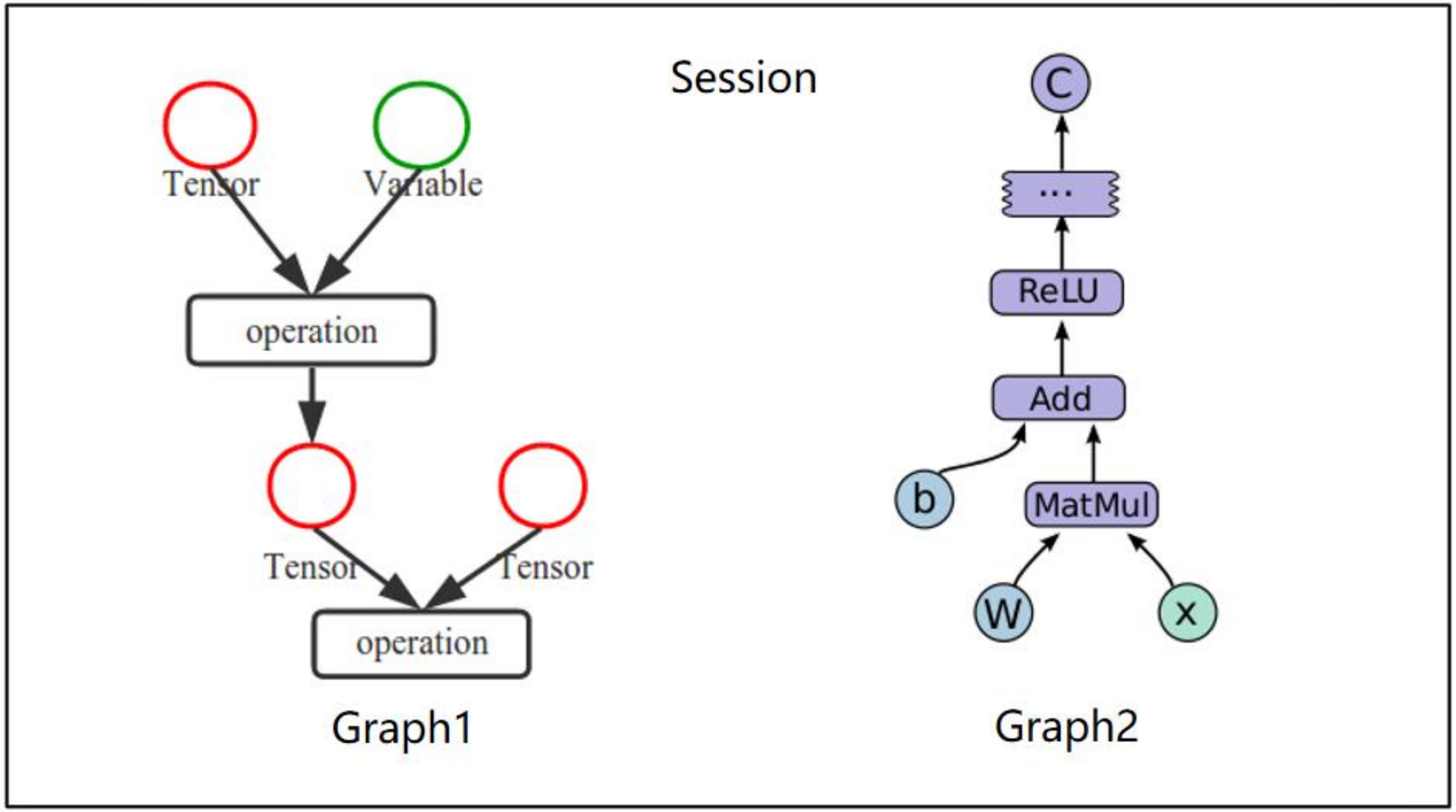
Tensorflow的基础使用与文本分类应用

Tensorboard基本概念

- 使用图（graphs）来表示计算任务
- 在被称之为会话（Session）的上下文（context）中执行图
- 使用tensor表示数据
- 通过变量（Variable）维护状态
- 使用feed和fetch可以为任意的操作赋值或者从其中获取数据

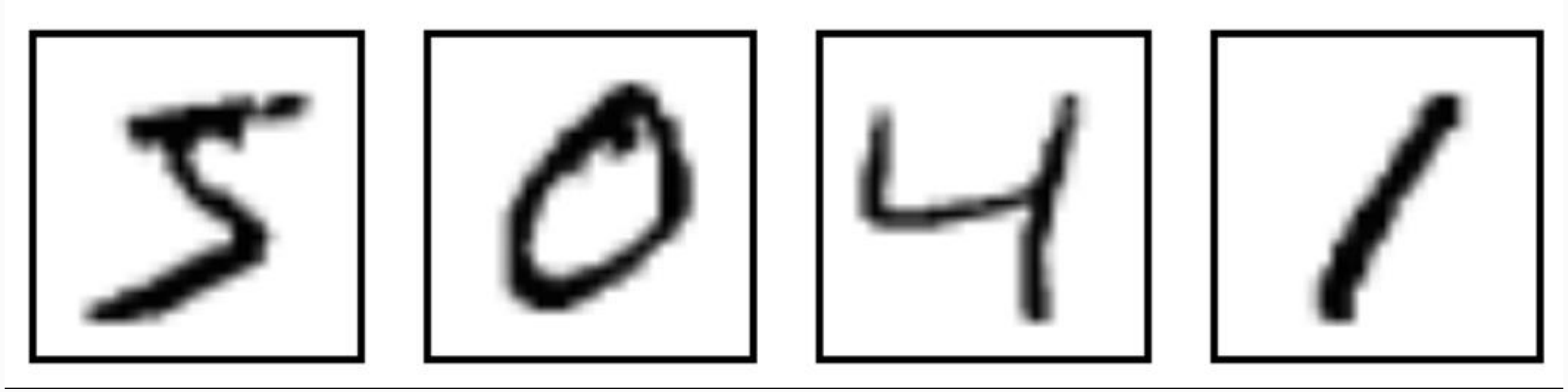
Tensorflow是一个编程系统，使用图（graphs）来表示计算任务，图（graphs）中的节点称之为op（operation），一个op获得0个或多个Tensor，执行计算，产生0个或多个Tensor。Tensor 看作是一个 n 维的数组或列表。图必须在会话（Session）里被启动。

Tensorboard结构



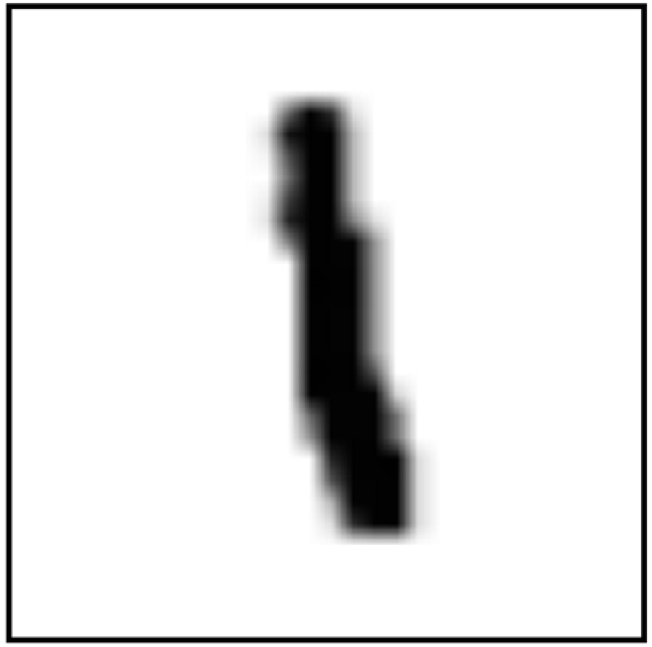
MNIST数据集

- MNIST数据集的官网：Yann LeCun's website
- 下载下来的数据集被分成两部分：60000行的训练数据集（mnist.train）和10000行的测试数据集（mnist.test）



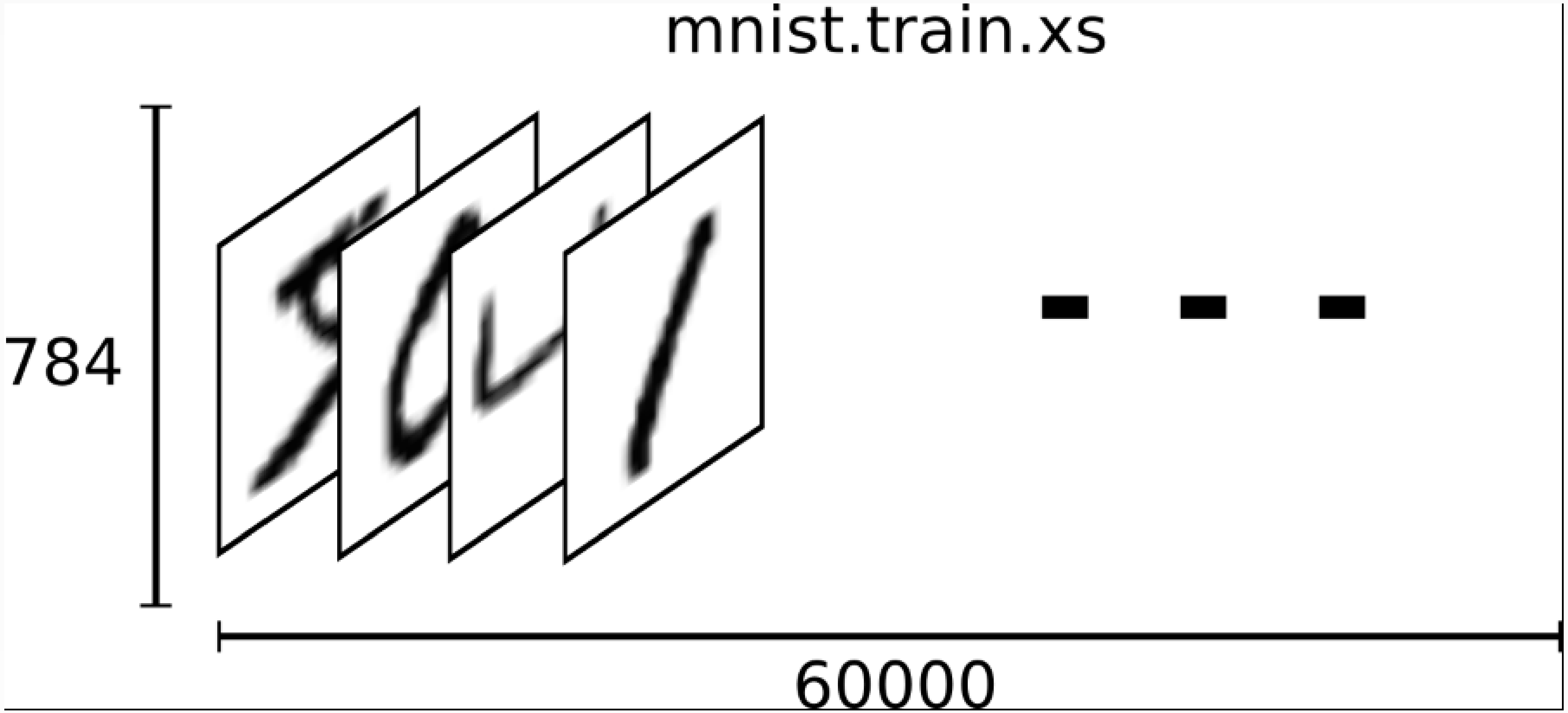
MNIST数据集

- 每一张图片包含28*28个像素，我们把这一个数组展开成一个向量，长度是28*28=784。因此在MNIST训练数据集中mnist.train.images 是一个形状为 [60000, 784] 的张量，第一个维度数字用来索引图片，第二个维度数字用来索引每张图片中的像素点。图片里的某个像素的强度值介于0-1之间。



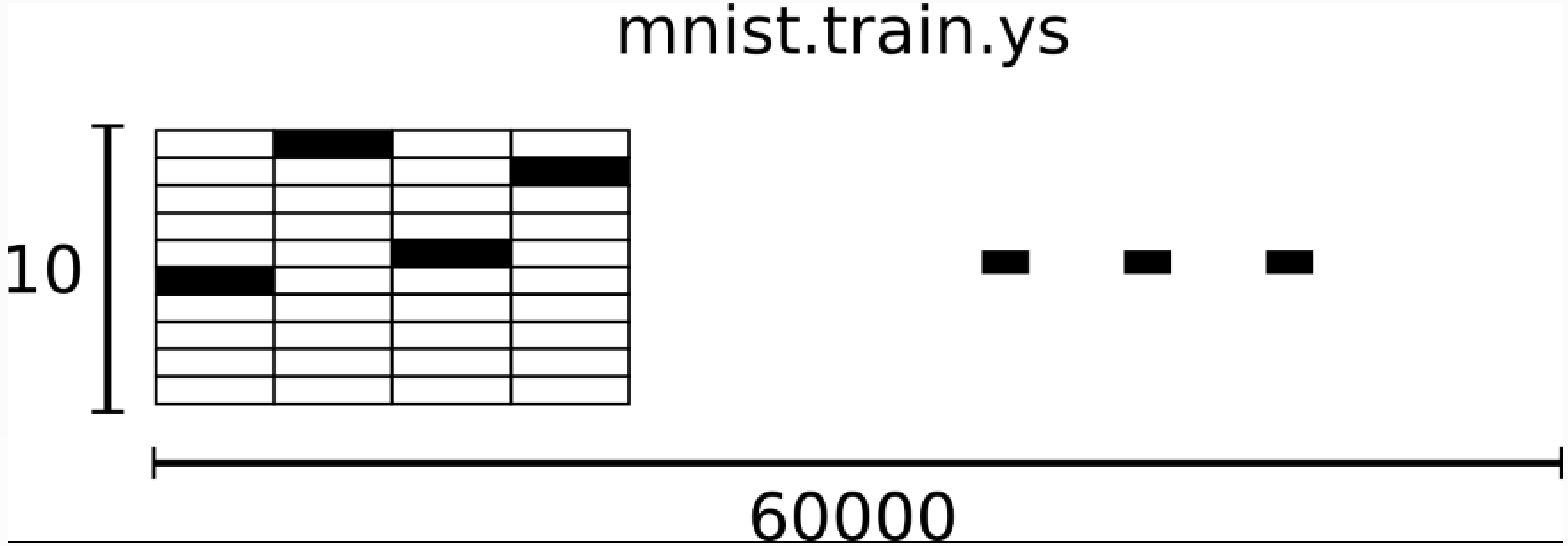
$$\simeq \begin{bmatrix}
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&.6&.8&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.7&1&0&0&0&0&0&0\\
0&0&0&0&0&0&.5&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.4&0&0&0&0&0\\
0&0&0&0&0&0&0&1&.7&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&0&.9&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&.3&1&.1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}$$

MNIST数据集

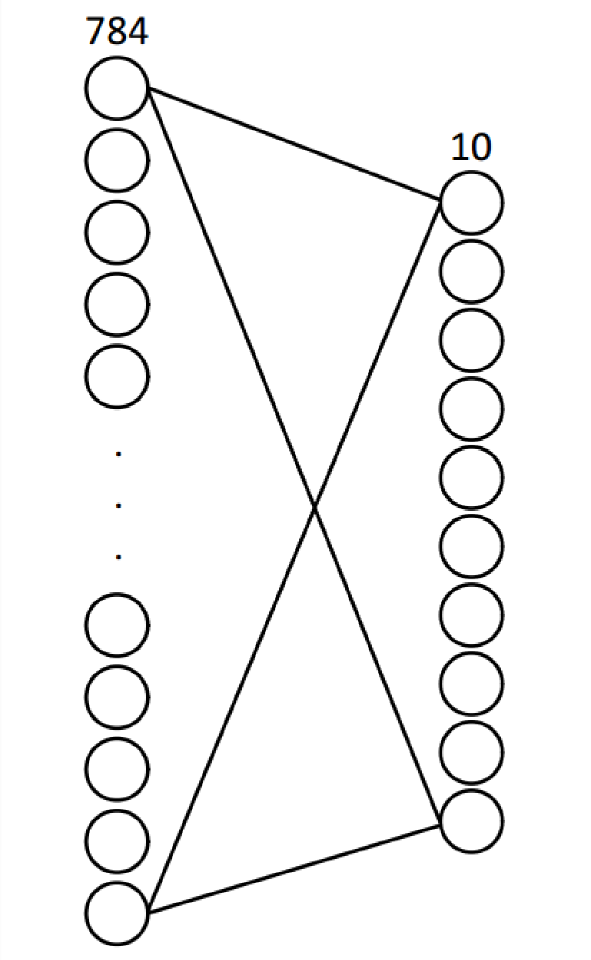


MNIST数据集

- MNIST数据集的标签是介于0-9的数字，我们要把标签转化为“one-hot vectors”。一个one-hot向量除了某一位数字是1以外，其余维度数字都是0，比如标签0将表示为([1,0,0,0,0,0,0,0,0,0])，标签3将表示为([0,0,0,1,0,0,0,0,0,0])。
- 因此， mnist.train.labels 是一个 [60000, 10] 的数字矩阵。



神经网络构建



MNIST分类（代码）

In [1]:
```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
```

In [2]:
```
#载入数据集
mnist = input_data.read_data_sets("MNIST_data",one_hot=True)

#每个批次100张照片
batch_size = 100
#计算一共有多少个批次
n_batch = mnist.train.num_examples // batch_size

#定义两个placeholder
x = tf.placeholder(tf.float32,[None,784])
y = tf.placeholder(tf.float32,[None,10])

#创建一个简单的神经网络，输入层784个神经元，输出层10个神经元
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
prediction = tf.nn.softmax(tf.matmul(x,W)+b)

#二次代价函数
#square是求平方
#reduce_mean是求平均值
loss = tf.reduce_mean(tf.square(y-prediction))

#使用梯度下降法来最小化loss，学习率是0.2
train_step = tf.train.GradientDescentOptimizer(0.2).minimize(loss)

#初始化变量
init = tf.global_variables_initializer()

#结果存放在一个布尔型列表中
correct_prediction = tf.equal(tf.argmax(y,1),tf.argmax(prediction,1))#argmax返回一维张量中最大的值所在的位置
#求准确率
accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))#cast是进行数据格式转换，把布尔型转为float32类型
```

MNIST分类（代码）

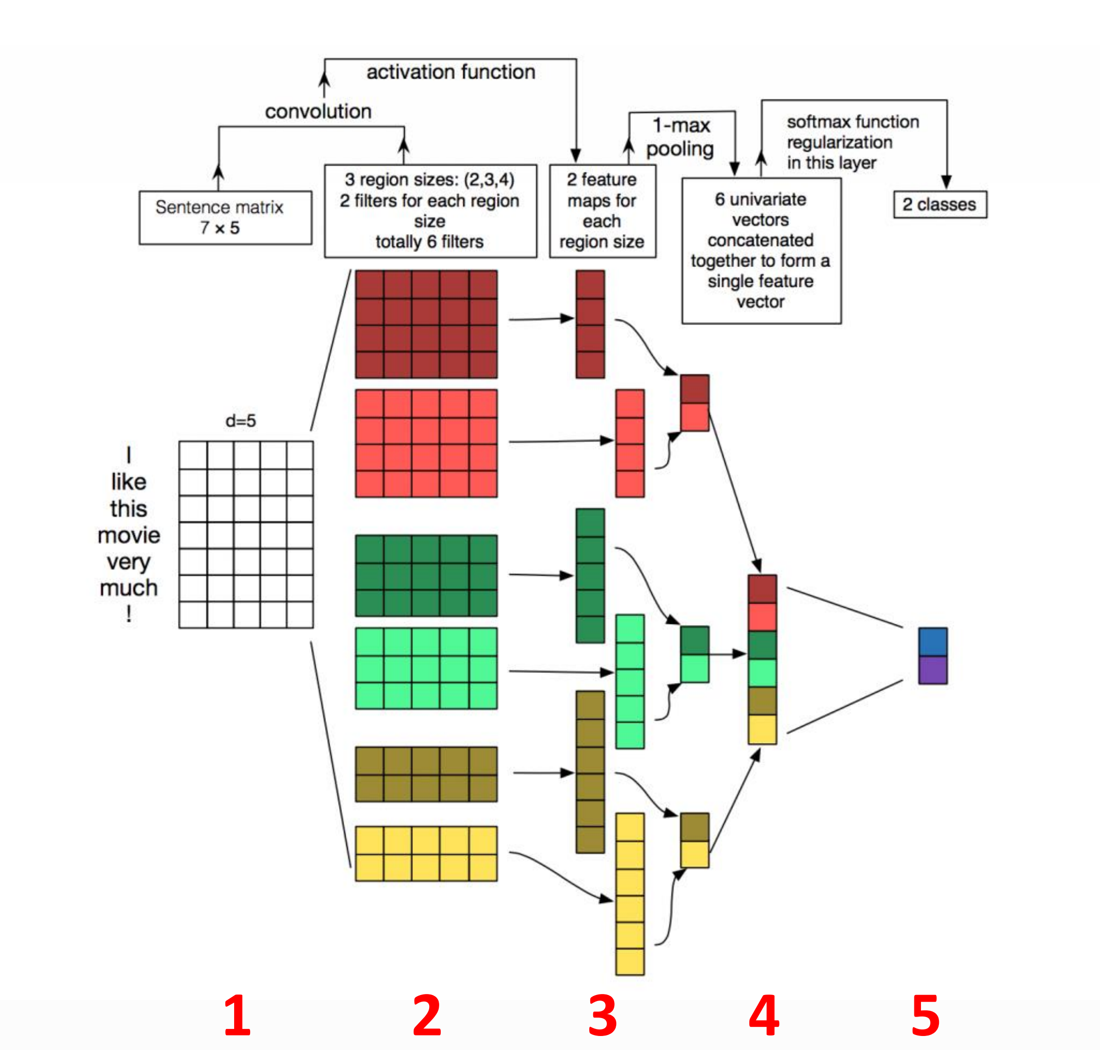
In [3]:
```
with tf.Session() as sess:
    #执行初始化
    sess.run(init)
    #迭代21个周期
    for epoch in range(21):
        #每个周期迭代n_batch个batch, 每个batch为100
        for batch in range(n_batch):
            #获得一个batch的数据和标签
            batch_xs,batch_ys =  mnist.train.next_batch(batch_size)
            #通过feed喂到模型中进行训练
            sess.run(train_step,feed_dict={x:batch_xs,y:batch_ys})

        #计算准确率
        acc = sess.run(accuracy,feed_dict={x:mnist.test.images,y:mnist.test.labels})
        print("Iter " + str(epoch) + ",Testing Accuracy " + str(acc))
```

```
Iter 0,Testing Accuracy 0.8315
Iter 1,Testing Accuracy 0.8701
Iter 2,Testing Accuracy 0.8805
Iter 3,Testing Accuracy 0.8882
Iter 4,Testing Accuracy 0.8943
Iter 5,Testing Accuracy 0.8969
Iter 6,Testing Accuracy 0.8992
Iter 7,Testing Accuracy 0.9017
Iter 8,Testing Accuracy 0.9032
Iter 9,Testing Accuracy 0.9052
Iter 10,Testing Accuracy 0.9064
Iter 11,Testing Accuracy 0.9071
Iter 12,Testing Accuracy 0.908
Iter 13,Testing Accuracy 0.9094
Iter 14,Testing Accuracy 0.9097
Iter 15,Testing Accuracy 0.9103
Iter 16,Testing Accuracy 0.9111
Iter 17,Testing Accuracy 0.912
Iter 18,Testing Accuracy 0.9128
Iter 19,Testing Accuracy 0.9133
Iter 20,Testing Accuracy 0.9132
```

CNN在文本分类中的应用

- CNN应用于NLP的任务，处理的往往是以矩阵形式表达的句子或文本。矩阵中的每一行对应于一个分词元素，一般是一个单词，也可以是一个字符。也就是说每一行都是一个词或者字符的向量（比如前面说到的word2vec）。假设我们一共有10个词，每个词都用128维的向量来表示，那么我们就可以得到一个10×128维的矩阵。这个矩阵就相当于是一副“图像”。

CNN在文本分类中的应用



CNN在文本分类中的应用

- 知乎数据集：https://biendata.com/competition/zhihu/data/

question_train_set.txt：
第一列为 问题id；
第二列为 title 的字符编号序列；
第三列为 title 的词语编号序列；
第四列为描述的字符编号序列；
第五列为描述的词语标号序列。

question_topic_train_set.txt：
第一列 问题 id；
第二列 话题 id。

topic_info.txt：
第一列为话题 id
第二列为话题的父话题 id。话题之间是有向无环图结构，一个话题可能有 0 到多个父话题；
第三列为话题名字的字符编号序列；
第四列为话题名字的词语编号序列；
第五列为话题描述的字符编号序列；
第六列为话题描述的词语编号序列。

CNN在文本分类中的应用

```
w305,w13549,w22752,w11,w7225,w2565,w1106,w16,w31389,w6,w1019,w69288,w111,w3332,w109,w11,w25,w1110,w111→3,1
w377,w54,w285,w57,w349,w54,w108215,w6,w47986,w875,w3352,w500,w21790,w12144,w111→769
w875,w15450,w42394,w15863,w6,w95421,w25,w803,w346,w6,w3763,w347,w88,w111→342
w8646,w2744,w1462,w9,w54,w138,w54,w50,w110,w140344,w111,w112,w49270,w2129,w6,w6978,w359,w10147,w111→1842,12
w380,w54,w674,w133,w54,w134,w614,w54,w929,w307,w109,w110,w19045,w6,w5830,w111→155,150,110,7,6
w133,w54,w134,w54,w518,w1133,w6,w5255,w9817,w109,w110,w111→110,11
w686,w27,w75,w1508,w11,w4668,w6,w54,w8153,w54,w1515,w1969,w90,w4699,w54,w8153,w54,w111,w109,w110,w3964,w111→351,260
w2298,w109,w110,w4253,w6,w12069,w2486,w111→197,16
w1448,w54,w644,w1231,w36910,w38972,w6,w1619,w71441,w1621,w1723,w11,w642,w3297,w76,w71441,w8582,w166,w434,w6,w1652,w25,w1370,w6,w111→600,484,38
w2218,w54,w1038,w125529,w90,w7665,w6,w929,w10147,w111→58
w9963,w9,w33263,w11,w945,w293,w1370,w6,w10351,w111→165,21
w3332,w54,w15848,w54,w305,w2429,w421,w88490,w11,w17317,w5063,w1038,w3246,w18656,w1505,w111→445,271,140
w70,w493,w16425,w75,w6,w6959,w25,w1110,w111→732,139
w102,w305,w202,w695,w644,w2169,w3587,w1110,w111→139,2
w305,w54,w935,w54,w70,w1065,w2203,w30121,w58291,w6,w5055,w63,w109,w110,w111→622,32
w305,w2351,w8926,w90,w90357,w6,w549,w11,w110,w2338,w21,w46971,w11473,w11437,w9650,w111→905,72
w7397,w637,w11716,w109,w110,w7546,w111→703,175
w140387,w5782,w65941,w5823,w11,w429,w1057,w471,w11682,w43048,w111→886,226
w14820,w54,w752,w54,w86,w84015,w7018,w17818,w6,w25,w140388,w6404,w11,w1076,w25,w18523,w111→307
w3327,w54,w140393,w57,w700,w54,w323,w110,w5779,w111,w2401,w25,w1050,w3045,w111→444,56
w825,w1638,w11,w31809,w19011,w1038,w728,w5784,w24937,w10147,w111→384,19
w6333,w54,w7456,w54,w1613,w1432,w1442,w4806,w54,w140396,w54,w14997,w54,w256,w1124,w2832,w259,w72,w10147,w111,w10717,w4333,w27582,w2021,w1442,w2
w875,w830,w723,w6407,w686,w7179,w6,w1411,w1038,w1023,w8383,w6,w1018,w1020,w111→790,218,130,120
w3332,w11586,w31303,w11,w40503,w13115,w30666,w8463,w346,w2182,w1038,w21,w11725,w4261,w111→219
w875,w1038,w109,w1619,w40744,w25,w140399,w21,w125131,w6,w13773,w1621,w3655,w6,w27125,w111,w109,w110,w1427,w111→1463,765
w62066,w62067,w6,w140402,w36885,w111→795,729
w76225,w54,w109,w1110,w3278,w21,w13472,w1107,w11569,w256,w486,w11,w1060,w6305,w259,w11,w1895,w256,w3830,w72,w1887,w20523,w644,w6,w3034,w259,w11
w4827,w54,w2830,w65,w578,w11,w928,w686,w1336,w2076,w33163,w10147,w111→1429,277,124
w87983,w54,w477,w6940,w109,w6933,w92059,w111,w677,w875,w928,w640,w24681,w13250,w11,w5791,w59,w8149,w54,w8231,w54,w1331,w0,w111→412
w10957,w6,w2292,w2299,w644,w11,w30637,w6,w224,w203,w6,w3145,w109,w110,w111→641,401,178
w133,w54,w134,w54,w307,w109,w110,w3565,w25276,w6,w19753,w628,w111→150,110
```

CNN在文本分类中的应用（代码）

```python
# Parameters
# ==================================================

# Data loading params
# validation数据集占比
tf.flags.DEFINE_float("dev_sample_percentage", .1, "Percentage of the training data to use for validation")
# 数据集
tf.flags.DEFINE_string("data_file", "./ieee_zhihu_cup/data_topic_block_0.txt", "Data source for the positive data.")

# Model Hyperparameters
# 词向量长度
tf.flags.DEFINE_integer("embedding_dim", 256, "Dimensionality of character embedding (default: 256)")
# 卷积核大小
tf.flags.DEFINE_string("filter_sizes", "3,4,5", "Comma-separated filter sizes (default: '3,4,5')")
# 每一种卷积核个数
tf.flags.DEFINE_integer("num_filters", 1024, "Number of filters per filter size (default: 1024)")
# dropout参数
tf.flags.DEFINE_float("dropout_keep_prob", 0.5, "Dropout keep probability (default: 0.5)")
# l2正则化参数
tf.flags.DEFINE_float("l2_reg_lambda", 0.0005, "L2 regularization lambda (default: 0.0005)")

# Training parameters
# 批次大小
tf.flags.DEFINE_integer("batch_size", 64, "Batch Size (default: 64)")
# 迭代周期
tf.flags.DEFINE_integer("num_epochs", 10, "Number of training epochs (default: 10)")
# 多少step测试一次
tf.flags.DEFINE_integer("evaluate_every", 50, "Evaluate model on dev set after this many steps (default: 50)")
# 多少step保存一次模型
tf.flags.DEFINE_integer("checkpoint_every", 200, "Save model after this many steps (default: 200)")
# 保存多少个模型
tf.flags.DEFINE_integer("num_checkpoints", 5, "Number of checkpoints to store (default: 5)")

# flags解析
FLAGS = tf.flags.FLAGS
FLAGS._parse_flags()

# 打印所有参数
print("\nParameters:")
for attr, value in sorted(FLAGS.__flags.items()):
    print("{}={}".format(attr.upper(), value))
print("")
```

CNN在文本分类中的应用（代码）

```
y = []
x_text = []

# 读取训练数据和标签
reader = pd.read_table(FLAGS.data_file,sep='\t',header=None)
for i in tqdm(xrange(reader.shape[0])):
    # 按','切分标签
    temp = reader.iloc[i][1].split(',')
    # 如果分类数大于5，只取前5个分类
    if (len(temp)>5):
        temp = temp[0:5]
    # 设置标签的对应位置为1，其余位置为0
    label = np.zeros(1999)
    for temp_label in temp:
        label[int(temp_label)] = 1
    y.append(label)
    x_text.append(reader.iloc[i][0])
```

```
100%|████████████████████████████████████████████████████████████████████| 300000/300000 [01:15<00:00, 3959.17it/s]
```

```
# 打印x_text和y的前5行
print(x_text[0:5])
y = np.array(y, dtype = np.float32)
print(y[0:5])
```

```
['w305,w13549,w22752,w11,w7225,w2565,w1106,w16,w31389,w6,w1019,w69288,w111,w3332,w109,w11,w25,w1110,w111', 'w377,w54,w285,w57,w349,w54,w108215,w6,w47986,w875,w3352,w500,w21790,w12144,w111', 'w875,w15450,w42394,w15863,w6,w95421,w25,w803,w346,w6,w3763,w347,w88,w111', 'w8646,w2744,w1462,w9,w54,w138,w54,w50,w110,w140344,w111,w112,w49270,w2129,w6,w6978,w359,w10147,w111', 'w380,w54,w674,w133,w54,w134,w614,w54,w929,w307,w109,w110,w19045,w6,w5830,w111']
[[ 0.  1.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]]
```

CNN在文本分类中的应用（代码）

```
# Build vocabulary
# 计算一段文本中最多的词汇数
max_document_length = max([len(x.split(",")) for x in x_text])
vocab_processor = tf.contrib.learn.preprocessing.VocabularyProcessor(max_document_length)

x = np.array(list(vocab_processor.fit_transform(x_text)))
print("x_shape:",x.shape)
print("y_shape:",y.shape)

# Split train/test set
# 数据集切分为两部分，训练集和验证集
dev_sample_index = -1 * int(FLAGS.dev_sample_percentage * float(len(y)))
x_train, x_dev = x[:dev_sample_index], x[dev_sample_index:]
y_train, y_dev = y[:dev_sample_index], y[dev_sample_index:]

print("Vocabulary Size: {:d}".format(len(vocab_processor.vocabulary_)))
print("Train/Dev split: {:d}/{:d}".format(len(y_train), len(y_dev)))
print("x:",x_train[0:5])
print("y:",y_train[0:5])
```

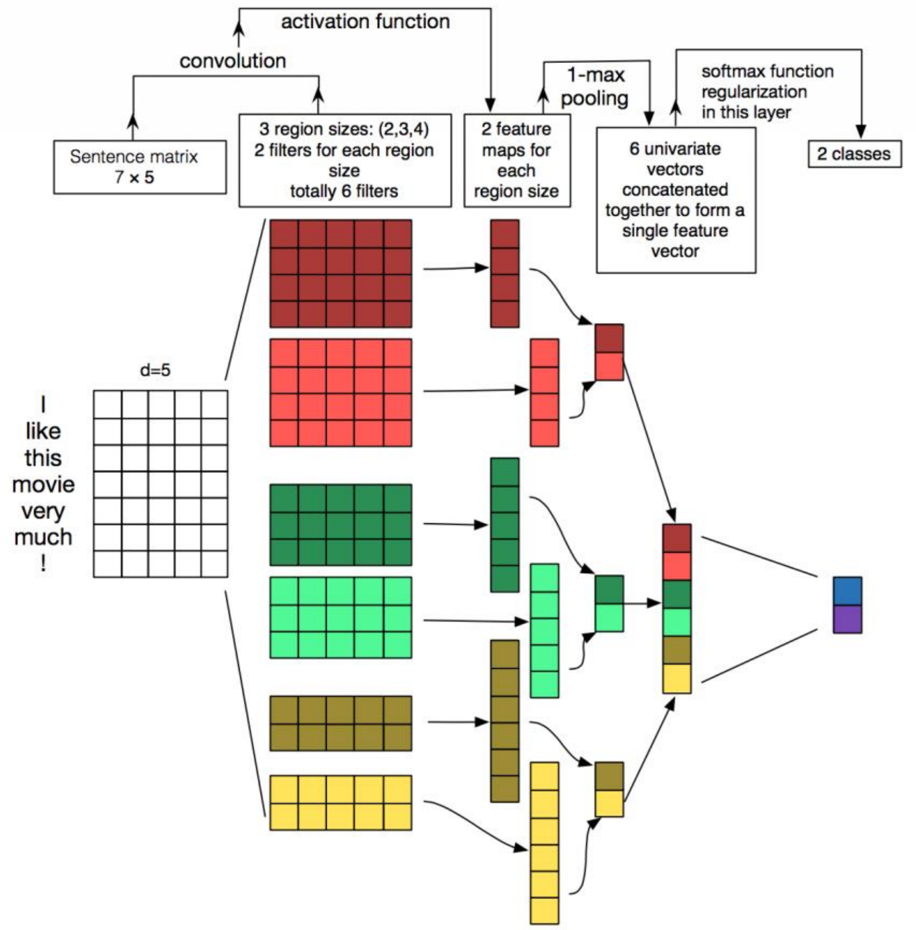
```
x_shape: (300000, 72)
y_shape: (300000, 1999)
Vocabulary Size: 131900
Train/Dev split: 270000/30000
x: [[ 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15  4 16 17 13  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [18 19 20 21 22 19 23 10 24 25 26 27 28 29 13  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [25 30 31 32 10 33 16 34 35 10 36 37 38 13  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [39 40 41 42 19 43 19 44 45 46 13 47 48 49 10 50 51 52 13  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [53 19 54 55 19 56 57 19 58 59 15 45 60 10 61 13  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
   0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]]
y: [[ 0.  1.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]]
```

CNN在文本分类中的应用（代码）

```
# 定义三个placeholder
input_x = tf.placeholder(tf.int32, [None, x_train.shape[1]], name="input_x")
input_y = tf.placeholder(tf.float32, [None, y_train.shape[1]], name="input_y")
dropout_keep_prob = tf.placeholder(tf.float32, name="dropout_keep_prob")

# sequence_length-最长词汇数
sequence_length=x_train.shape[1]
# num_classes-分类数
num_classes=y_train.shape[1]
# vocab_size-总词汇数
vocab_size=len(vocab_processor.vocabulary_)
# embedding_size-词向量长度
embedding_size=FLAGS.embedding_dim
# filter_sizes-卷积核尺寸3, 4, 5
filter_sizes=list(map(int, FLAGS.filter_sizes.split(",")))
# num_filters-卷积核数量
num_filters=FLAGS.num_filters

Weights = tf.Variable(tf.random_uniform([vocab_size, embedding_size], -1.0, 1.0), name="Weights")
# shape:[None, sequence_length, embedding_size]
embedded_chars = tf.nn.embedding_lookup(Weights, input_x)
# 添加一个维度, shape:[None, sequence_length, embedding_size, 1]
embedded_chars_expanded = tf.expand_dims(embedded_chars, -1)
```

CNN在文本分类中的应用（代码）

```python
# Create a convolution + maxpool layer for each filter size
pooled_outputs = []
for i, filter_size in enumerate(filter_sizes):
    with tf.name_scope("conv-maxpool-%s" % filter_size):
        # Convolution Layer
        filter_shape = [filter_size, embedding_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(
            embedded_chars_expanded,
            W,
            strides=[1, 1, 1, 1],
            padding="VALID",
            name="conv")
        # Apply nonlinearity
        h = tf.nn.relu(tf.nn.bias_add(conv, b), name="relu")
        # Maxpooling over the outputs
        pooled = tf.nn.max_pool(
            h,
            ksize=[1, sequence_length - filter_size + 1, 1, 1],
            strides=[1, 1, 1, 1],
            padding='VALID',
            name="pool")
        pooled_outputs.append(pooled)

# Combine all the pooled features
num_filters_total = num_filters * len(filter_sizes)
print("num_filters_total:", num_filters_total)
h_pool = tf.concat(pooled_outputs, 3)
h_pool_flat = tf.reshape(h_pool, [-1, num_filters_total])
```



CNN在文本分类中的应用（代码）

```
# Add dropout
with tf.name_scope("dropout"):h_drop = tf.nn.dropout(h_pool_flat,dropout_keep_prob)

# Final (unnormalized) scores and predictions
with tf.name_scope("output"):
    W = tf.get_variable(
        "W",
        shape=[num_filters_total, num_classes],
        initializer=tf.contrib.layers.xavier_initializer())
    b = tf.Variable(tf.constant(0.1, shape=[num_classes]), name="b")
    scores = tf.nn.xw_plus_b(h_drop, W, b, name="scores")

# 定义loss
with tf.name_scope("loss"):
    loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=scores, labels=input_y))

# 定义优化器
with tf.name_scope("optimizer"):
    optimizer = tf.train.AdamOptimizer(1e-3).minimize(loss)
```

```
num_filters_total: 3072
```

CNN在文本分类中的应用（代码）

```python
# 生成批次数据
def batch_iter(data, batch_size, num_epochs, shuffle=False):
    """
    Generates a batch iterator for a dataset.
    """
    data = np.array(data)
    data_size = len(data)
    # 每个epoch的num_batch
    num_batches_per_epoch = int((len(data) - 1) / batch_size) + 1
    print("num_batches_per_epoch:",num_batches_per_epoch)
    for epoch in range(num_epochs):
        # Shuffle the data at each epoch
        if shuffle:
            shuffle_indices = np.random.permutation(np.arange(data_size))
            shuffled_data = data[shuffle_indices]
        else:
            shuffled_data = data
        for batch_num in range(num_batches_per_epoch):
            start_index = batch_num * batch_size
            end_index = min((batch_num + 1) * batch_size, data_size)
            yield shuffled_data[start_index:end_index]
```

CNN在文本分类中的应用（代码）

```python
# 知乎提供的评测方案
def eval(predict_label_and_marked_label_list):
    """
    :param predict_label_and_marked_label_list: 一个元组列表。例如
    [ ([1, 2, 3, 4, 5], [4, 5, 6, 7]),
      ([3, 2, 1, 4, 7], [5, 7, 3])
     ]
    需要注意这里 predict_label 是去重复的，例如 [1,2,3,2,4,1,6]，去重后变成[1,2,3,4,6]

    marked_label_list 本身没有顺序性，但提交结果有，例如上例的命中情况分别为
    [0，0，0，1，1]   (4，5命中)
    [1，0，0，0，1]   (3，7命中)

    """
    right_label_num = 0  #总命中标签数量
    right_label_at_pos_num = [0, 0, 0, 0, 0]  #在各个位置上总命中数量
    sample_num = 0   #总问题数量
    all_marked_label_num = 0    #总标签数量
    for predict_labels, marked_labels in predict_label_and_marked_label_list:
        sample_num += 1
        marked_label_set = set(marked_labels)
        all_marked_label_num += len(marked_label_set)
        for pos, label in zip(range(0, min(len(predict_labels), 5)), predict_labels):
            if label in marked_label_set:     #命中
                right_label_num += 1
                right_label_at_pos_num[pos] += 1

    precision = 0.0
    for pos, right_num in zip(range(0, 5), right_label_at_pos_num):
        precision += ((right_num / float(sample_num))) / math.log(2.0 + pos)  # 下标0-4 映射到 pos1-5 + 1，所以最终+2
    recall = float(right_label_num) / all_marked_label_num

    return 2*(precision * recall) / (precision + recall )
```

CNN在文本分类中的应用（代码）

```python
# 定义saver，只保存最新的5个模型
saver = tf.train.Saver(tf.global_variables(), max_to_keep=FLAGS.num_checkpoints)

with tf.Session() as sess:
    predict_top_5 = tf.nn.top_k(scores, k=5)
    label_top_5 = tf.nn.top_k(input_y, k=5)
    sess.run(tf.global_variables_initializer())
    i = 0
    # 生成数据
    batches = batch_iter(
        list(zip(x_train, y_train)), FLAGS.batch_size, FLAGS.num_epochs)
    for batch in batches:
        i = i + 1
        # 得到一个batch的数据
        x_batch, y_batch = zip(*batch)
        # 优化模型
        sess.run([optimizer],feed_dict={input_x:x_batch, input_y:y_batch, dropout_keep_prob:FLAGS.dropout_keep_prob})

        # 每训练50次测试1次
        if (i % FLAGS.evaluate_every == 0):
            print ("Evaluation:step",i)
            predict_5, label_5, _loss = sess.run([predict_top_5,label_top_5,loss],feed_dict={input_x:x_train,
                                                                                              input_y:y_batch,
                                                                                              dropout_keep_prob:1.0})

            print ("label:",label_5[1][:5])
            print ("predict:",predict_5[1][:5])
            print ("predict:",predict_5[0][:5])
            print ("loss:",_loss)
            predict_label_and_marked_label_list = []
            for predict,label in zip(predict_5[1],label_5[1]):
                predict_label_and_marked_label_list.append((list(predict),list(label)))
            score = eval(predict_label_and_marked_label_list)
            print("score:",score)

        # 每训练200次保存1次模型
        if (i % FLAGS.checkpoint_every == 0):
            path = saver.save(sess, "models/model", global_step=i)
            print("Saved model checkpoint to {}".format(path))
```

一些优化方案

- 把所有的数据训练完
- 增加训练周期，使用动态学习率
- 使用知乎提供的word_embedding.txt文件来初始化词汇向量表
- 考虑使用更多种类的数据
- 调节网络参数
- 优化网络结构
- ……

微信公众号：深度学习与神经网络

Github

- 今天分享的内容都放到了我的github上：https://github.com/Qinbf