# 负面评论识别

提到负面评论，我们首先要聊下水军。水军顾名思义，是指在论坛大量灌水的人员 ，他们受雇于[网络公关公司](http://baike.baidu.com/item/%E7%BD%91%E7%BB%9C%E5%85%AC%E5%85%B3%E5%85%AC%E5%8F%B8)、以发帖回帖为主要手段、为雇主进行网络造势的网络人员，有专职和兼职之分。一般来讲，发帖回帖造势常常需要成百上千个人共同完成，那些临时在网上征集来的发帖的人在行内被叫做“网络水军”。2009年12月央视报道网络水军这一新兴现象之后，受到社会广泛关注，不少长期在线的网虫们纷纷加入网络水军一族。水军的大行其道，严重影响了社会舆论，尤其是有组织的针对公众人物、企业的诽谤、攻击行为，造成严重的社会影响。大型的社交媒体以及知名论坛，早期都需要雇佣大量的运营支撑人员来人工鉴别处理。是否可以使用机器学习的技术来达到一定程度的自动化识别负面评论呢？我们利用垃圾邮件识别的技术积累，尝试来完成这一任务。



1. 网络水军

本章主要以IMDB数据集为例子介绍负面评论的识别技术。

本章1.1介绍验证码识别使用的数据集。

本章1.2介绍使用的特征提取方法，包括一维向量和二维向量。

本章1.3介绍使用的模型以及对应的验证结果，包括K近邻、支持向量机和深度学习。

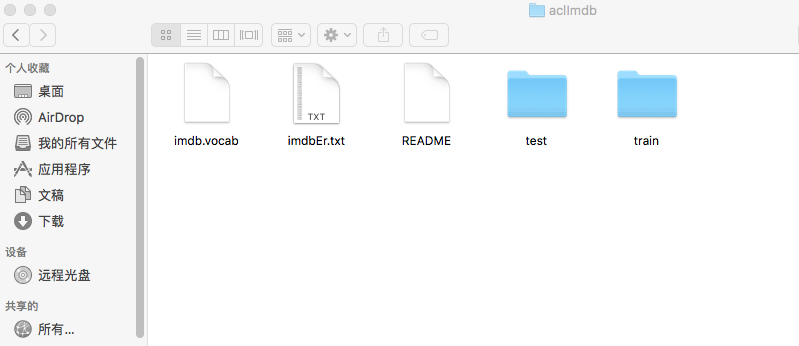
本章演示代码请参考本书Github上的review.py文件。

## 数据集

测试数据来自互联网[电影](http://baike.baidu.com/item/%E7%94%B5%E5%BD%B1/31689)资料库（Internet Movie Database，IMDB），IMDB是一个关于电影演员、电影、电视节目、电视明星和[电影制作](http://baike.baidu.com/item/%E7%94%B5%E5%BD%B1%E5%88%B6%E4%BD%9C)的[在线数据库](http://baike.baidu.com/item/%E5%9C%A8%E7%BA%BF%E6%95%B0%E6%8D%AE%E5%BA%93)。IMDb另一受欢迎的特色是其对应每个数据库条目，并且有47个主要板块的留言板系统。注册用户可以在这些留言板上分享和讨论关于电影，演员，导演的消息。至今已有超过6百万注册用户使用过留言板。我们使用标注为正面评论和负面评论的留言板数据。

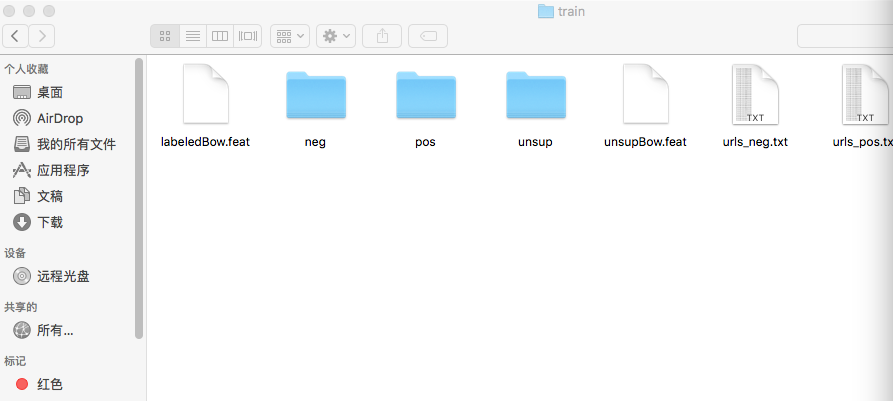
整个数据集一共10万条记录，5万做了标记，5万没有做标记。5万做了标记的数据集合被随机分配成了训练数据集和测试数据集。

IMDB文件夹下一级目录主要包含训练数据集和测试数据集两个文件夹。



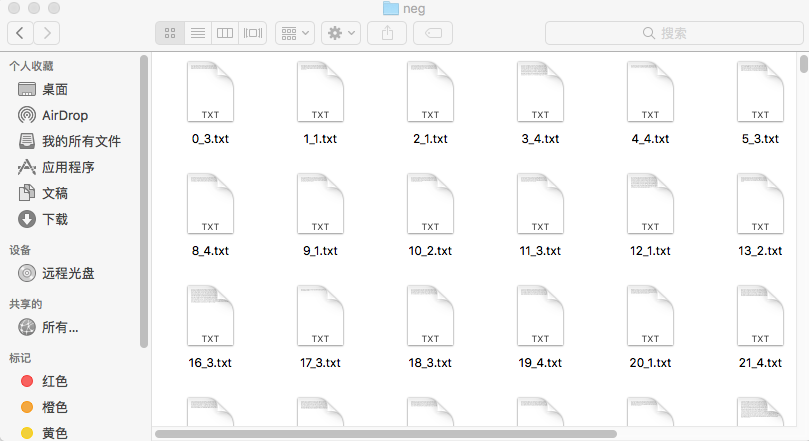
1. IMDB一级文件夹结构

以训练数据集为例，IMDB二级目录下主要包含三个目录，分别为正面评论、负面评论和未标识。



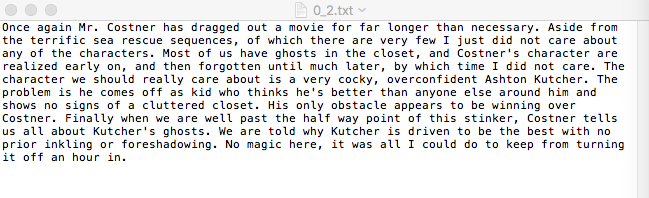
1. IMDB二级文件夹结构

以正面评论为例，IMDB三级目录下为评论文件，每个评论以单独文件形式保存。



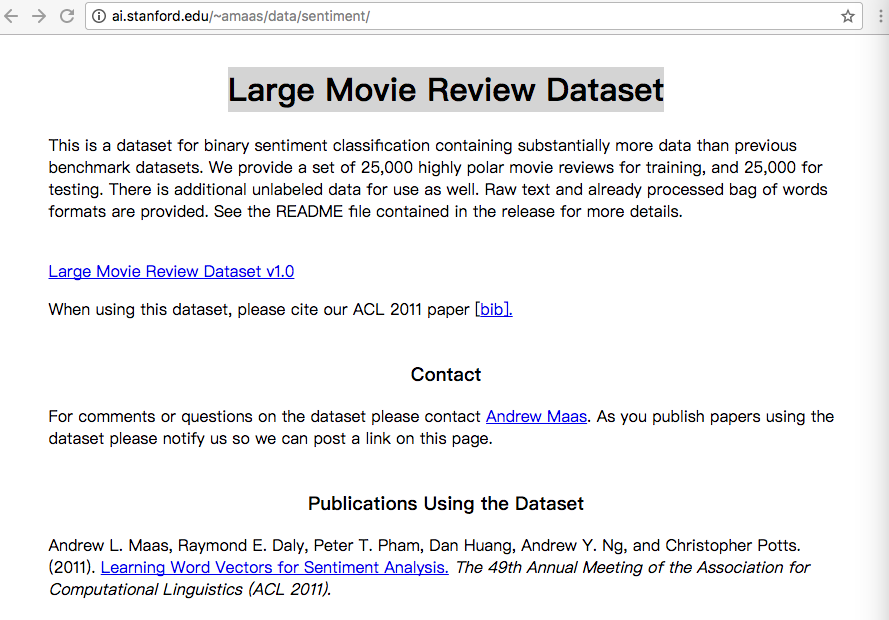
1. IMDB三级文件夹结构

评论文件记载了原始的评论内容。



1. IMDB评论文件内容

IMDB数据的下载地址为：http://ai.stanford.edu/~amaas/data/sentiment/



1. IMDB数据集下载主页

## 特征提取

### 方法一：词袋模型

我们使用最常见的词袋模型提取文件特征。

把一个评论文件作为一个完整的字符串处理，定义函数load\_one\_file加载文件到一个字符串变量中返回。

def load\_one\_file(filename):  
 x=""  
 with open(filename) as f:  
 for line in f:  
 line=line.strip('\n')  
 line = line.strip('\r')  
 x+=line  
 return x

遍历目录，加载目录下全部文件，以字符串集合的形式返回。

def load\_files\_from\_dir(rootdir):  
 x=[]  
 list = os.listdir(rootdir)  
 for i in range(0, len(list)):  
 path = os.path.join(rootdir, list[i])  
 if os.path.isfile(path):  
 v=load\_one\_file(path)  
 x.append(v)  
 return x

分别加载训练数据集目录下的正面评论和负面评论目录下全部文件，同时进行标记，正面评论为0，负面评论为1。

path="../data/review/aclImdb/train/pos/"  
print "Load %s" % path  
x\_train=load\_files\_from\_dir(path)  
y\_train=[0]\*len(x\_train)  
path="../data/review/aclImdb/train/neg/"  
print "Load %s" % path  
tmp=load\_files\_from\_dir(path)  
y\_train+=[1]\*len(tmp)

x\_train+=tmp

分别加载训练数据集目录下的正面评论和负面评论目录下全部文件，同时进行标记，正面评论为0，负面评论为1。

path="../data/review/aclImdb/test/pos/"  
print "Load %s" % path  
x\_test=load\_files\_from\_dir(path)  
y\_test=[0]\*len(x\_test)  
path="../data/review/aclImdb/test/neg/"  
print "Load %s" % path  
tmp=load\_files\_from\_dir(path)  
y\_test+=[1]\*len(tmp)

x\_test+=tmp

使用Scikit-Learn的CountVectorizer函数进行词袋化处理，同时将抽取的词汇表保存，用于词袋化测试数据集。其中非常重要的几个参数的含义为：

* decode\_error，处理解码失败的方式，分为‘strict’、‘ignore’、‘replace’三种方式
* strip\_accents，在预处理步骤中移除重音的方式
* max\_features，词袋特征个数的最大值
* stop\_words，判断word结束的方式
* max\_df，df最大值
* min\_df，df最小值
* binary，默认为False，当与TF-IDF结合使用时需要设置为True

本例中处理的数据集均为英文，所以针对解码失败直接忽略，使用ignore方式，stop\_words的方式使用english，strip\_accents方式为ascii方式。

x\_train, y\_train, x\_test, y\_test=load\_all\_files()  
vectorizer = CountVectorizer(  
 decode\_error='ignore',  
 strip\_accents='ascii',  
 max\_features=max\_features,  
 stop\_words='english',  
 max\_df=1.0,  
 min\_df=1 )  
print vectorizer  
x\_train=vectorizer.fit\_transform(x\_train)  
x\_train=x\_train.toarray()  
vocabulary=vectorizer.vocabulary\_

复用vocabulary对测试数据集进行词袋化处理，其中由于设置了vocabulary，max\_features的值会被忽略。

vectorizer = CountVectorizer(  
 decode\_error='ignore',  
 strip\_accents='ascii',  
 vocabulary=vocabulary,  
 stop\_words='english',  
 max\_df=1.0,  
 min\_df=1 )  
print vectorizer  
x\_test=vectorizer.fit\_transform(x\_test)  
x\_test=x\_test.toarray()

## 模型训练与验证

### 方法一：朴素贝叶斯算法

使用朴素贝叶斯算法，特征提取使用词袋模型。

gnb = GaussianNB()  
gnb.fit(x\_train,y\_train)  
y\_pred=gnb.predict(x\_test)

评估结果的准确度和TT、FF、TF、FT四个值。

print metrics.accuracy\_score(y\_test, y\_pred)  
print metrics.confusion\_matrix(y\_test, y\_pred)

在词袋最大特征数为5000的情况下，整个系统准确度为68.64%，TT、FF、TF、FT矩阵如表1-1 所示。

1. 基于词袋模型的朴素贝叶斯验证结果

|  |  |  |
| --- | --- | --- |
| **类型名称** | **T** | **F** |
| T | 6255 | 6245 |
| F | 1595 | 10905 |

完整输出结果为：

CountVectorizer(analyzer=u'word', binary=False, decode\_error='ignore',

        dtype=<type 'numpy.int64'>, encoding=u'utf-8', input=u'content',

        lowercase=True, max\_df=1.0, max\_features=None, min\_df=1,

        ngram\_range=(1, 1), preprocessor=None, stop\_words='english',

        strip\_accents='ascii', token\_pattern=u'(?u)\\b\\w\\w+\\b',

        tokenizer=None,

NB and wordbag

0.6864

[[ 6255  6245]

 [ 1595 10905]]

### 方法二：支持向量机算法

使用支持向量机算法，特征提取使用词袋模型。

clf = svm.SVC()  
clf.fit(x\_train, y\_train)  
y\_pred = clf.predict(x\_test)

评估结果的准确度和TT、FF、TF、FT四个值。

print metrics.accuracy\_score(y\_test, y\_pred)  
print metrics.confusion\_matrix(y\_test, y\_pred)

在词袋最大特征数为5000的情况下，整个系统准确度为%，TT、FF、TF、FT矩阵如表1-1 所示。

1. 基于词袋模型的朴素贝叶斯验证结果

|  |  |  |
| --- | --- | --- |
| **类型名称** | **T** | **F** |
| T | 6255 | 6245 |
| F | 1595 | 10905 |

完整输出结果为：

CountVectorizer(analyzer=u'word', binary=False, decode\_error='ignore',

        dtype=<type 'numpy.int64'>, encoding=u'utf-8', input=u'content',

        lowercase=True, max\_df=1.0, max\_features=None, min\_df=1,

        ngram\_range=(1, 1), preprocessor=None, stop\_words='english',

        strip\_accents='ascii', token\_pattern=u'(?u)\\b\\w\\w+\\b',

        tokenizer=None,

SVM and wordbag

### 方法三：深度学习算法之DNN

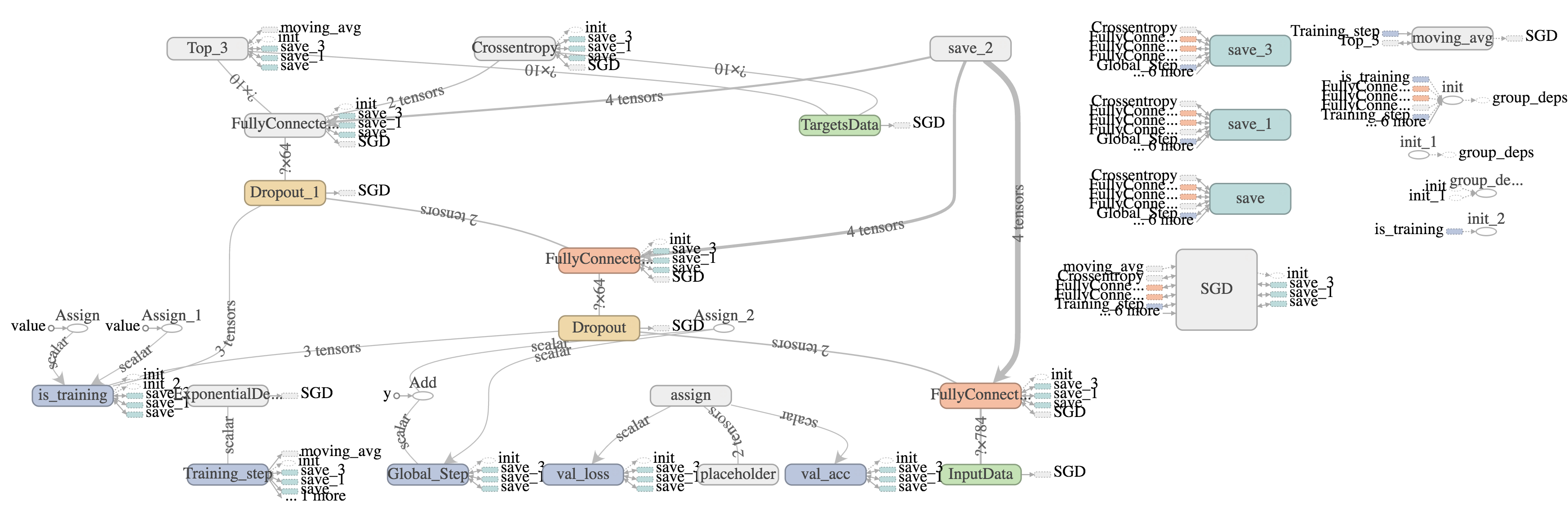
特征提取方式使用一维向量，定义输入层为维度为784的一维向量。

input\_layer = tflearn.input\_data(shape=[None, 784])

构造DNN模型，使用3个全连接层。

dense1 = tflearn.fully\_connected(input\_layer, 64, activation='tanh',  
 regularizer='L2', weight\_decay=0.001)  
dropout1 = tflearn.dropout(dense1, 0.8)  
dense2 = tflearn.fully\_connected(dropout1, 64, activation='tanh',  
 regularizer='L2', weight\_decay=0.001)  
dropout2 = tflearn.dropout(dense2, 0.8)  
softmax = tflearn.fully\_connected(dropout2, 10, activation='softmax')  
# Regression using SGD with learning rate decay and Top-3 accuracy  
sgd = tflearn.SGD(learning\_rate=0.1, lr\_decay=0.96, decay\_step=1000)  
top\_k = tflearn.metrics.Top\_k(3)  
net = tflearn.regression(softmax, optimizer=sgd, metric=top\_k,  
 loss='categorical\_crossentropy')

完整的DNN结构如图1-6 所示。



1. 用于识别MNIST的DNN结构图

实例化DNN算法并训练10轮。

# Training  
model = tflearn.DNN(net, tensorboard\_verbose=0)  
model.fit(X, Y, n\_epoch=10, validation\_set=(testX, testY),  
 show\_metric=True, run\_id="mnist")

运行程序，经过10轮训练，准确率达到了99.49%。

Training Step: 8600 | total loss: 0.28404 | time: 5.892s

| SGD | epoch: 010 | loss: 0.28404 - top3: 0.9842 | val\_loss: 0.12718 - val\_acc: 0.9949 -- iter: 55000/55000

### 方法四：深度学习算法之CNN

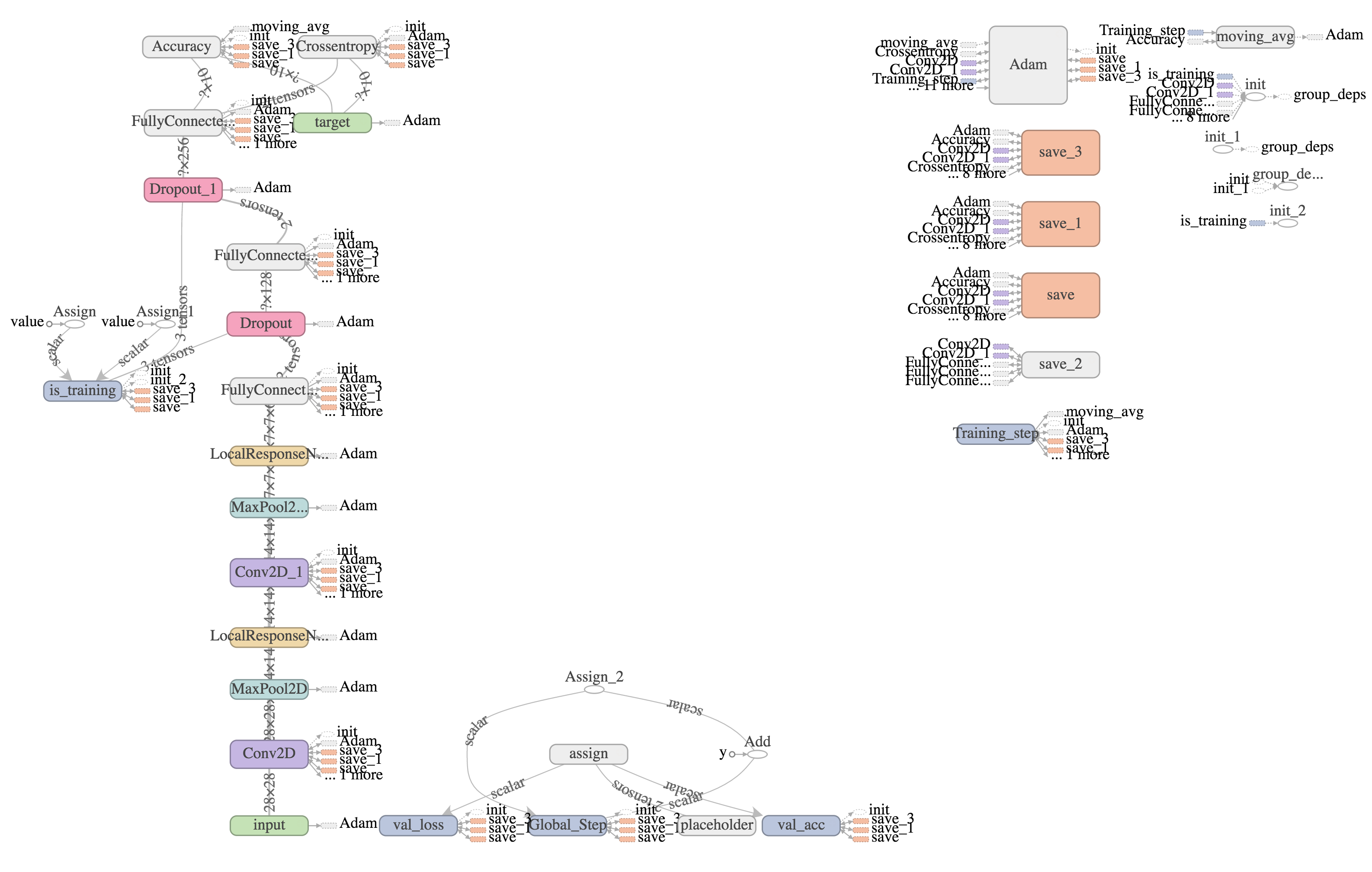
特征提取方式使用二维向量，定义输入层为28乘以28的二维向量。

network = input\_data(shape=[None, 28, 28, 1], name='input')

构造CNN模型，使用两层卷积两层全连接结构。

network = conv\_2d(network, 32, 3, activation='relu', regularizer="L2")  
network = max\_pool\_2d(network, 2)  
network = local\_response\_normalization(network)  
network = conv\_2d(network, 64, 3, activation='relu', regularizer="L2")  
network = max\_pool\_2d(network, 2)  
network = local\_response\_normalization(network)  
network = fully\_connected(network, 128, activation='tanh')  
network = dropout(network, 0.8)  
network = fully\_connected(network, 256, activation='tanh')  
network = dropout(network, 0.8)  
network = fully\_connected(network, 10, activation='softmax')  
network = regression(network, optimizer='adam', learning\_rate=0.01,  
 loss='categorical\_crossentropy', name='target')

完整的CNN结构如图1-7 所示。



1. 用于识别MNIST的CNN结构图

实例化CNN算法并训练10轮。

model = tflearn.DNN(network, tensorboard\_verbose=0)  
model.fit({'input': X}, {'target': Y}, n\_epoch=10,  
 validation\_set=({'input': testX}, {'target': testY}),  
 snapshot\_step=100, show\_metric=True, run\_id='convnet\_mnist')

运行程序，经过10轮训练，准确率达到了96.98%。

Training Step: 8600 | total loss: 0.39358 | time: 139.542s

| Adam | epoch: 010 | loss: 0.39358 - acc: 0.9366 | val\_loss: 0.12829 - val\_acc: 0.9698 -- iter: 55000/55000

## 本章小结

本章基于MNIST数据集介绍了验证码的识别方法。针对MNIST数据集，特征提取方法有一维向量和二维向量两种方法。训练模型介绍了K近邻、支持向量机以及深度学习的DNN和CNN算法，其中DNN算法表现优异，识别率达到了99%以上。

### 参考文献

* <http://china.huanqiu.com/hot/2013-09/4320971.html>
* <http://dl.sohu.com/20130911/n386353688.shtml>
* Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). [Learning Word Vectors for Sentiment Analysis.](http://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf) The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).