正则化缓解过拟合

正则化在损失函数中引入模型复杂度指标，利用给W加权值，弱化了训练数据的噪声（一般不正则化b）

loss = loss(y与y_) + REGULARIZER * loss(w)

- loss(y与y_)：模型中所有参数的损失函数，如交叉熵、均方误差
- REGULARIZER：正则化权重，给出w在总loss中的比例
- loss(w)：需正则化的参数

loss(w) = tf.contrib.layers.l1_regularizer(REGULARIZER)(w)　$loss_{L1}(w) = \sum_i |w_i|$

loss(w) = tf.contrib.layers.l2_regularizer(REGULARIZER)(w)　$loss_{L2}(w) = \sum_i |w_i|^2$

tf.add_to_collection('losses', tf.contrib.layers.l2_regularizer(regularizer)(w))

loss = cem + tf.add_n(tf.get_collection('losses'))

- losses：集合
- cem：交叉熵

```
import matplotlib.pyplot as plt
plt.scatter(X坐标, y坐标, c="颜色")   # 可视化数据点
plt.show()

xx, yy = np.mgrid[起:止:步长, 起:止:步长]   # 生成网格坐标点
grid = np.c_[xx.ravel(), yy.ravel()]   # 收集该区域内所有网格坐标点
```

- np.c_：组成矩阵
- ravel()：拉直

搭建模块化的神经网络八股

前向传播 (forward.py)

```
def forward(x, regularizer):
    w=
    b=
    y=
    return y

def get_weight(shape, regularizer)
    w=tf.Variable(...)
    tf.add_to_collection('losses', tf.contrib.layers.l2_regularizer(regularizer)(w))
    return w

def get_bias(shape):
    b=tf.Variable(...)
    return b
```

反向传播:

```
def backward():
    X=tf.placeholder()
    Y_=tf.placeholder()
    y=forward.forward(x, REGULARIZER)
    global_step=tf.Variable(...)
    loss=
```

opt 4_8

正则化

指数衰减学习率

滑动平均

MNIST数据集输出识别准确率

MNIST数据集 { 6w张训练 / 1w张测试 } 28X28像素手写数字

```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('./data/', one_hot=True)
```

- 返回各子集样本数

```
print "train data size:", mnist.train.num_examples
print "validation data size:", mnist.validation.num_examples
print "test data size:", mnist.test.num_examples
```

返回标签和数据

```
mnist.train.labels[0]
mnist.train.images[0]
```

取一小撮数据，喂入神经网络

```
BATCH_SIZE = 200
xs, ys = mnist.train.next_batch(BATCH_SIZE)
```

tf.get_collection("") 从集合中取出全部变量，生成一个列表

tf.add_n([]) 列表内对应元素相加

tf.cast(x, dtype) 把x转为dtype类型

tf.argmax(x, axis) 返回最大值所在索引号； tf.argmax([1,0,0],1) ⇒ 0

os.path.join("home","name") 返回 home/name

字符串.split() 按指定拆分符对字符串切片，返回分割后的列表

with tf.Graph().as_default() as g #定义的节点在计算图g中

· 保存模型

```
saver = tf.train.Saver()
with tf.Session() as sess:
    for i in range(STEPS):
        if i % 轮数 == 0:
            saver.save(sess, os.path.join(路径名, 保存文件名), global_step=global_step)
```

加载模型

```
with tf.Session() as sess:
    ckpt = tf.train.get_checkpoint_state(存储路径)
    if ckpt and ckpt.model_checkpoint_path:
        saver.restore(sess, ckpt.model_checkpoint_path)
```

实例化可还原滑动平均值的 saver

```
ema = tf.train.ExponentialMovingAverage(滑动平均基数)
ema_restore = ema.variables_to_restore()
saver = tf.train.Saver(ema_restore)
```

准确率计算方法

```
correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

axis=0 ⇒ max per col (列)
axis=1 ⇒ max per row (行)

· 损失函数 loss 含正则化 regularization

backword.py 加入:

```
ce = tf.nn.sparse_softmax_cross_entropy_with_logits(logits=y, labels=
        tf.argmax(y_, 1))
cem = tf.reduce_mean(ce)
loss = cem + tf.add_n(tf.get_collection('losses'))
```

forward.py 加入:

```
if regularizer != None: tf.add_to_collection("losses", tf.contrib.layers
l2_regularizer(regularizer)(w))
```

学习率 learning_rate

backward.py 中加入:

```
learning_rate = tf.train.exponential_decay(
    LEARNING_RATE_BASE,
    global_step,
    LEARNING_RATE_STEP,
    LEARNING_RATE_DECAY,
    staircase=True)
```

滑动平均 ema

backward.py 加入

```
ema = tf.train.ExponentialMovingAverage(MOVING_AVERAGE_DECAY, global_step)
ema_op = ema.apply(tf.trainable_variables())
with tf.control_dependencies([train_step, ema_op]):
    train_op = tf.no_op(name='train')
```

test.py:

```
def test(mnist):
    with tf.Graph().as_default() as g:
        定义 x, y_, y
        实例化可还原滑动平均值的 saver
        计算正确率
        while True:
            with tf.Session() as sess:
                加载 ckpt 模型
                如果有模型:
                    恢复对话
                    恢复轮数
                    计算准确率
                    打印提示
                如果没有模型:
                    给出提示
```

输入手写数字图片预测结果

① 对真实图片预测结果

testPicArr = pre_pic(testPic)

preValue = restore_model(testPicArr)

② 制做数据集

tf records 进行数据读取

用 tf.train.Example 存储训练数据。训练数据的特征用键值对形式表示：

'img_raw': 值 'label': 值 值是 BytesList / FloatList / intLList

用 SerializeToString() 把数据序列化成字符串存储

mnist_backward：看 loss

mnist_test ：准确率

卷积神经网络

全连接NN：每个神经元与前后相邻层每一个神经元都元连接关系，输入是特征，输出为预测结果

待优化参数过多易产生过拟合

输出图片边长=(输入图片边长-卷积核长+1)/步长

全零填充 Padding { SAME 入长/步长 (向上取整)
VALID (不全0填充) (入长-核长+1)/步长 (向上取整)

·计算卷积

tf.nn.conv2d (输入描述，eg. [batch, 5, 5, 1] (一次喂多少数据，分辨率，通道数)

卷积核描述，eg [3, 3, 1, 16] (行列分辨率，通道，卷积核个数)

核滑动步长，eg [1, 1, 1, 1] (固定，行步长，列步长，固定)

padding = 'VALID') $\frac{5-3+1}{1}=3$

· 池化 { 减少特征数量
最大值池化可提取图片纹理，均值池化可保留背景特征

pool = tf.nn.max_pool / avg_pool (输入输入，eg. [batch, 28, 28, 6] (行列分辨率，通道数)

池化核描述(核大小)，eg [1, 2, 2, 1] (行列分辨率)

池化滑动步长，eg [1, 2, 2, 1] (行步长，列步长)

padding = 'SAME')
使用全填充