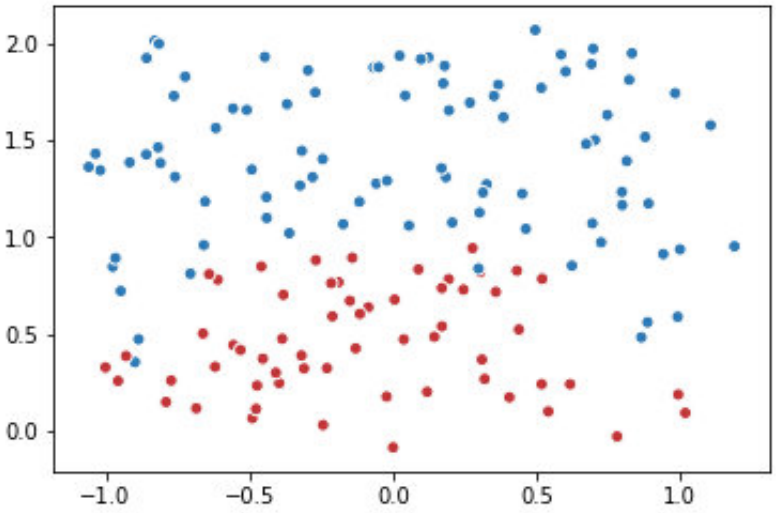
使用两个不同的学习率

```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np

data = []
label = []
np.random.seed(0)

# 以原点为圆心，半径为1的圆把散点划分成红蓝两部分，并加入随机噪音。
for i in range(150):
    x1 = np.random.uniform(-1,1)
    x2 = np.random.uniform(0,2)
    if x1**2 + x2**2 <= 1:
        data.append([np.random.normal(x1, 0.1),np.random.normal(x2,0.1)])
        label.append(0)
    else:
        data.append([np.random.normal(x1, 0.1), np.random.normal(x2, 0.1)])
        label.append(1)

data = np.hstack(data).reshape(-1,2)
label = np.hstack(label).reshape(-1, 1)
plt.scatter(data[:,0], data[:,1], c=label,
           cmap="RdBu", vmin=-.2, vmax=1.2, edgecolor="white")
plt.show()
```



```
def get_weight(shape, lambda1):
    var = tf.Variable(tf.random_normal(shape), dtype=tf.float32)
    tf.add_to_collection('losses', tf.contrib.layers.l2_regularizer(lambda1)(var))
    return var
```

```
x = tf.placeholder(tf.float32, shape=(None, 2))
y_ = tf.placeholder(tf.float32, shape=(None, 1))
sample_size = len(data)

# 每层节点的个数
layer_dimension = [2,10,5,3,1]

n_layers = len(layer_dimension)

cur_layer = x
in_dimension = layer_dimension[0]

# 循环生成网络结构
for i in range(1, n_layers):
    out_dimension = layer_dimension[i]
    weight = get_weight([in_dimension, out_dimension], 0.003)
    bias = tf.Variable(tf.constant(0.1, shape=[out_dimension]))
    cur_layer = tf.nn.elu(tf.matmul(cur_layer, weight) + bias)
    in_dimension = layer_dimension[i]

y= cur_layer

# 损失函数的定义。
mse_loss = tf.reduce_sum(tf.pow(y_ - y, 2)) / sample_size
tf.add_to_collection('losses', mse_loss)
loss = tf.add_n(tf.get_collection('losses'))
```

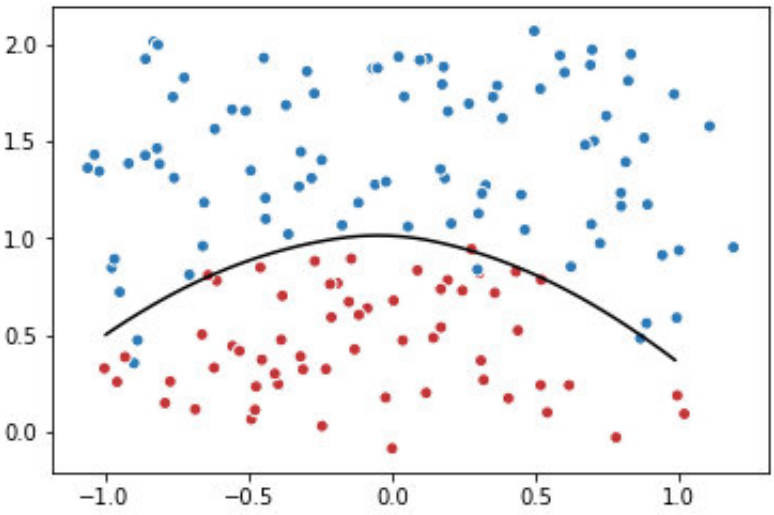
```
# 定义训练的目标函数loss，训练次数及训练模型
train_op = tf.train.AdamOptimizer(0.001).minimize(loss)
TRAINING_STEPS = 40000

with tf.Session() as sess:
    tf.global_variables_initializer().run()
    for i in range(TRAINING_STEPS):
        sess.run(train_op, feed_dict={x: data, y_: label})
        if i % 2000 == 0:
            print("After %d steps, loss: %f" % (i, sess.run(loss, feed_dict={x: da
ta, y_: label})))

    # 画出训练后的分割曲线
    xx, yy = np.mgrid[-1:1:.01, 0:2:.01]
    grid = np.c_[xx.ravel(), yy.ravel()]
    probs = sess.run(y, feed_dict={x:grid})
    probs = probs.reshape(xx.shape)

plt.scatter(data[:,0], data[:,1], c=label,
           cmap="RdBu", vmin=-.2, vmax=1.2, edgecolor="white")
plt.contour(xx, yy, probs, levels=[.5], cmap="Greys", vmin=0, vmax=.1)
plt.show()
```

```
After 0 steps, loss: 11.037084
After 2000 steps, loss: 0.151889
After 4000 steps, loss: 0.120565
After 6000 steps, loss: 0.092557
After 8000 steps, loss: 0.072915
After 10000 steps, loss: 0.064701
After 12000 steps, loss: 0.059759
After 14000 steps, loss: 0.058005
After 16000 steps, loss: 0.057474
After 18000 steps, loss: 0.057397
After 20000 steps, loss: 0.057354
After 22000 steps, loss: 0.057331
After 24000 steps, loss: 0.054357
After 26000 steps, loss: 0.054301
After 28000 steps, loss: 0.054291
After 30000 steps, loss: 0.054287
After 32000 steps, loss: 0.054297
After 34000 steps, loss: 0.054282
After 36000 steps, loss: 0.054280
After 38000 steps, loss: 0.054279
```



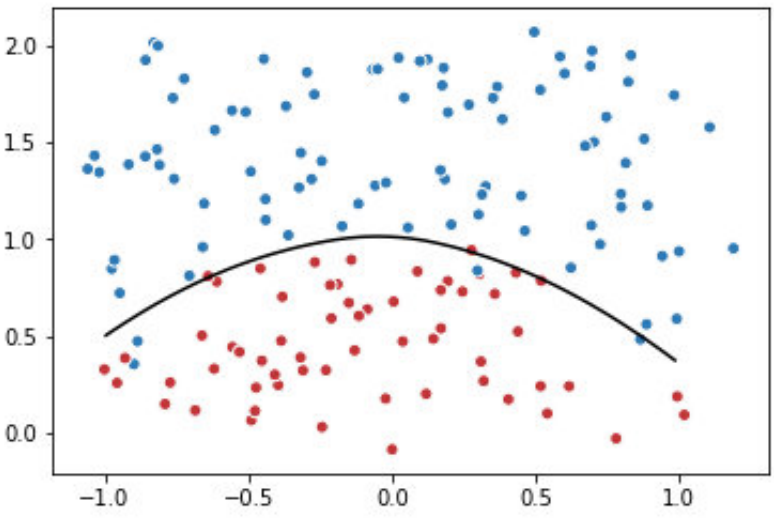

```
# 定义训练的目标函数loss，训练次数及训练模型
train_op = tf.train.AdamOptimizer(0.01).minimize(loss)
TRAINING_STEPS = 40000

with tf.Session() as sess:
    tf.global_variables_initializer().run()
    for i in range(TRAINING_STEPS):
        sess.run(train_op, feed_dict={x: data, y_: label})
        if i % 2000 == 0:
            print("After %d steps, loss: %f" % (i, sess.run(loss, feed_dict={x: da
ta, y_: label})))

    # 画出训练后的分割曲线
    xx, yy = np.mgrid[-1:1:.01, 0:2:.01]
    grid = np.c_[xx.ravel(), yy.ravel()]
    probs = sess.run(y, feed_dict={x:grid})
    probs = probs.reshape(xx.shape)

plt.scatter(data[:,0], data[:,1], c=label,
           cmap="RdBu", vmin=-.2, vmax=1.2, edgecolor="white")
plt.contour(xx, yy, probs, levels=[.5], cmap="Greys", vmin=0, vmax=.1)
plt.show()
```

```
After 0 steps, loss: 1.597551
After 2000 steps, loss: 0.055537
After 4000 steps, loss: 0.054880
After 6000 steps, loss: 0.054848
After 8000 steps, loss: 0.054836
After 10000 steps, loss: 0.054836
After 12000 steps, loss: 0.054836
After 14000 steps, loss: 0.054836
After 16000 steps, loss: 0.054836
After 18000 steps, loss: 0.054836
After 20000 steps, loss: 0.054836
After 22000 steps, loss: 0.054836
After 24000 steps, loss: 0.054836
After 26000 steps, loss: 0.054838
After 28000 steps, loss: 0.054836
After 30000 steps, loss: 0.054836
After 32000 steps, loss: 0.054836
After 34000 steps, loss: 0.054836
After 36000 steps, loss: 0.054899
After 38000 steps, loss: 0.054836
```



使用了0.001和0.01这两个学习率，可以看出当学习率是0.01的时候模型损失值下降的更快，但是在经过40000次迭代学习之后，学习率0.001的时候模型更优。

使用两个不同的Decay值

```
v1 = tf.Variable(0, dtype=tf.float32)
step = tf.Variable(0, trainable=False)
ema = tf.train.ExponentialMovingAverage(0.99, step)
maintain_averages_op = ema.apply([v1])
```

```
with tf.Session() as sess:

    # 初始化
    init_op = tf.global_variables_initializer()
    sess.run(init_op)
    print(sess.run([v1, ema.average(v1)]))

    # 更新变量v1的取值
    sess.run(tf.assign(v1, 5))
    sess.run(maintain_averages_op)
    print(sess.run([v1, ema.average(v1)]))

    # 更新step和v1的取值
    sess.run(tf.assign(step, 10000))
    sess.run(tf.assign(v1, 10))
    sess.run(maintain_averages_op)
    print(sess.run([v1, ema.average(v1)]))

    # 更新一次v1的滑动平均值
    sess.run(maintain_averages_op)
    print(sess.run([v1, ema.average(v1)]))
```

```
[0.0, 0.0]
[5.0, 4.5]
[10.0, 4.5549998]
[10.0, 4.6094499]
```

```
v1 = tf.Variable(0, dtype=tf.float32)
step = tf.Variable(0, trainable=False)
ema = tf.train.ExponentialMovingAverage(0.80, step)
maintain_averages_op = ema.apply([v1])
with tf.Session() as sess:

    # 初始化
    init_op = tf.global_variables_initializer()
    sess.run(init_op)
    print(sess.run([v1, ema.average(v1)]))

    # 更新变量v1的取值
    sess.run(tf.assign(v1, 5))
    sess.run(maintain_averages_op)
    print(sess.run([v1, ema.average(v1)]))

    # 更新step和v1的取值
    sess.run(tf.assign(step, 10000))
    sess.run(tf.assign(v1, 10))
    sess.run(maintain_averages_op)
    print(sess.run([v1, ema.average(v1)]))

    # 更新一次v1的滑动平均值
    sess.run(maintain_averages_op)
    print(sess.run([v1, ema.average(v1)]))
```

```
[0.0, 0.0]
[5.0, 4.5]
[10.0, 5.5999999]
[10.0, 6.48]
```