1 最基本的学习 - 非(not)

In [4]:

```
from __future__ import print_function, division
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline

import tensorflow as tf
```

```
C:\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the
second argument of issubdtype from `float` to `np.floating` is deprecated. In futur
e, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

In [2]:

```
import tensorflow as tf

#输入数据的处理
x = np.array([[0],
             [1]])
y = np.array([[1],[0]])
# 占位符
x_placeholder = tf.placeholder(tf.float32,shape=[None,1])
y_placeholder = tf.placeholder(tf.float32,shape=[None,1])
# Weight和bias
W = tf.Variable(tf.random_normal([1,  1], -1,  1))
b = tf.Variable(tf.random_normal([1], -1,  1))
# 预测
y_p = tf.matmul(x_placeholder,W)+b
# y_p = tf.nn.sigmoid(tf.matmul(x_placeholder,W)+b)

# 定义损失函数，训练方法，初始化变量，并进行训练
loss=tf.reduce_mean(tf.square(y_p-y_placeholder))
train=tf.train.GradientDescentOptimizer(0.1).minimize(loss)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(10000):
        sess.run(train,feed_dict={x_placeholder:x,y_placeholder:y})
        if i%1000==0:
            _loss = sess.run(loss,feed_dict={x_placeholder:x,y_placeholder:y})
            print("loss:",_loss)
    _y_p = sess.run(y_p,feed_dict={x_placeholder:x,y_placeholder:y})
    print("predict result:",_y_p.tolist())
    # 后续作图的准备
    line_x = np.linspace(-1,2,10).reshape([-1,1])
    line_y = sess.run(y_p,feed_dict={x_placeholder:line_x})
```

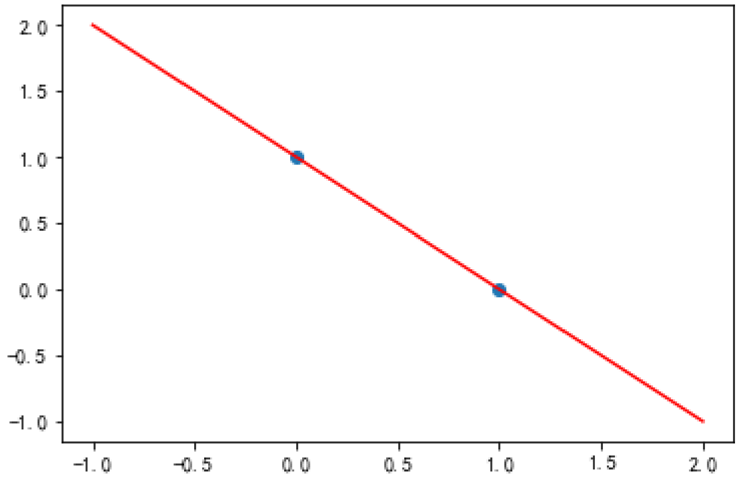
```
C:\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the
second argument of issubdtype from `float` to `np.floating` is deprecated. In futur
e, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters

loss: 5.423808
loss: 1.8829382e-13
loss: 1.8829382e-13
loss: 1.8829382e-13
loss: 1.8829382e-13
loss: 1.8829382e-13
loss: 1.8829382e-13
loss: 1.8829382e-13
loss: 1.8829382e-13
loss: 1.8829382e-13
predict result: [[0.9999994635558197], [2.980232238769531e-07]]
```

In [3]:

```
plt.scatter(x.reshape(2),y.reshape(2))
plt.plot(line_x,line_y,color='r')
```

Out[3]:

```
[<matplotlib.lines.Line2D at 0x24c9f4bec88>]
```



2 最基本的学习 - 与(and)

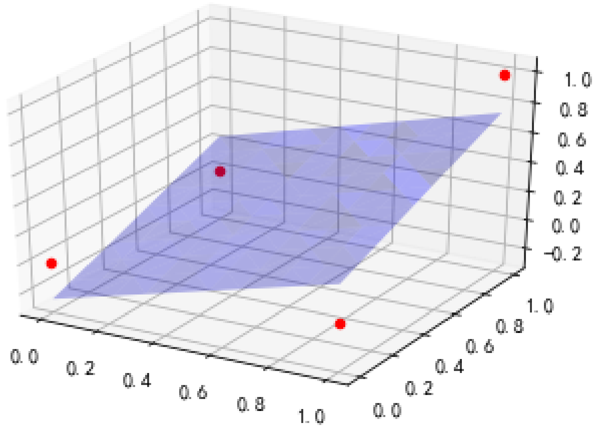
In [4]:

```
import tensorflow as tf
#输入数据的处理
x = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])
y = np.array([[0],[0],[0],[1]])
# 占位符
x_placeholder = tf.placeholder(tf.float32,shape=[None,2])
y_placeholder = tf.placeholder(tf.float32,shape=[None,1])
# Weight和bias
W = tf.Variable(tf.random_normal([2, 1], -1, 1))
b = tf.Variable(tf.random_normal([1], -1, 1))
# 预测
y_p = tf.matmul(x_placeholder,W)+b
# y_p = tf.nn.sigmoid(tf.matmul(x_placeholder,W)+b)
# 定义损失函数，训练方法，初始化变量，并进行训练
loss=tf.reduce_mean(tf.square(y_p-y_placeholder))
train=tf.train.GradientDescentOptimizer(0.1).minimize(loss)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(10000):
        sess.run(train,feed_dict={x_placeholder:x,y_placeholder:y})
        if i%1000==0:
            _loss = sess.run(loss,feed_dict={x_placeholder:x,y_placeholder:y})
            print("loss:",_loss)
    _y_p = sess.run(y_p,feed_dict={x_placeholder:x,y_placeholder:y})
    print("predict result:",_y_p.tolist())
    # 后续作图的准备
    line_y = np.linspace(0,1,10).repeat(10).reshape([10,10])
    line_x = line_y.T
    line_x = line_x.flatten()
    line_y = line_y.flatten()
    x_input = np.hstack([line_x[:,None],line_y[:,None]])
    line_z = sess.run(y_p,feed_dict={x_placeholder:x_input})
```

```
loss: 6.8259497
loss: 0.0625
loss: 0.0625
loss: 0.0625
loss: 0.0625
loss: 0.0625
loss: 0.0625
loss: 0.0625
loss: 0.0625
loss: 0.0625
predict result: [[-0.24999964237213135], [0.2500000298023224], [0.2500000298023224],
[0.7499997019767761]]
```

In [6]:

```
from mpl_toolkits.mplot3d import Axes3D,axes3d
import matplotlib.pyplot as plt
import numpy as np



x,y,z = axes3d.get_test_data(0.05)
x,y,z=x.flatten(), y.flatten(), z.flatten()
fig = plt.figure()
ax = fig.gca(projection='3d')

ax.scatter(0,0,0,color='r')
ax.scatter(0,1,0,color='r')
ax.scatter(1,0,0,color='r')
ax.scatter(1,1,1,color='r')
# ax.plot_wireframe(x, y, z, rstride=10, cstride=10)
ax.plot_trisurf(line_x,line_y,line_z.flatten(),alpha=0.3,color='b' )

plt.show()
```



3 最基本的学习 - 或(or)

In [9]:

```
#输入数据的处理
x = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])
y = np.array([[0],[1],[1],[1]])
# 占位符
x_placeholder = tf.placeholder(tf.float32,shape=[None,2])
y_placeholder = tf.placeholder(tf.float32,shape=[None,1])
# Weight和bias
W = tf.Variable(tf.random_normal([2, 1], -1, 1))
b = tf.Variable(tf.random_normal([1], -1, 1))
# 预测
y_p = tf.matmul(x_placeholder,W)+b
# y_p = tf.nn.sigmoid(tf.matmul(x_placeholder,W)+b)

# 定义损失函数，训练方法，初始化变量，并进行训练
loss=tf.reduce_mean(tf.square(y_p-y_placeholder))
train=tf.train.GradientDescentOptimizer(0.1).minimize(loss)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(10000):
        sess.run(train,feed_dict={x_placeholder:x,y_placeholder:y})
        if i%1000==0:
            _loss = sess.run(loss,feed_dict={x_placeholder:x,y_placeholder:y})
            print("loss:",_loss)
    _y_p = sess.run(y_p,feed_dict={x_placeholder:x,y_placeholder:y})
    print("predict result:",_y_p.tolist())
```

```
loss: 0.75724953
loss: 0.062499996
loss: 0.062499996
loss: 0.062499996
loss: 0.062499996
loss: 0.062499996
loss: 0.062499996
loss: 0.062499996
loss: 0.062499996
loss: 0.062499996
predict result: [[0.24999937415122986], [0.75], [0.75], [1.2500005960464478]]
```

4 稍微有难度的学习：异或(xor)

——为什么深度学习会兴起

In [ ]:

```
# 一是足够多的数据，二是有足够复杂的神经网络
```

In [6]:

```
#输入数据的处理
x = np.array([[0,0],
              [0,1],
              [1,0],
              [1,1]])
y = np.array([[1],[0],[0],[1]])
# 占位符
x_placeholder = tf.placeholder(tf.float32,shape=[None,2])
y_placeholder = tf.placeholder(tf.float32,shape=[None,1])
# Weight和bias
W = tf.Variable(tf.random_normal([2,  1],  -1,  1))
b = tf.Variable(tf.random_normal([1],  -1,  1))
# 预测
y_p = tf.nn.sigmoid(tf.matmul(x_placeholder,W)+b)

# 定义损失函数，训练方法，初始化变量，并进行训练
loss=tf.reduce_mean(tf.square(y_p-y_placeholder))
train=tf.train.GradientDescentOptimizer(0.1).minimize(loss)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(10000):
        sess.run(train,feed_dict={x_placeholder:x,y_placeholder:y})
        if i%1000==0:
            _loss = sess.run(loss,feed_dict={x_placeholder:x,y_placeholder:y})
            print("loss:",_loss)
    _y_p = sess.run(y_p,feed_dict={x_placeholder:x,y_placeholder:y})
    print("predict result:",_y_p.tolist())
```

```
loss: 0.329992
loss: 0.25116795
loss: 0.2500208
loss: 0.25000042
loss: 0.25
loss: 0.25
loss: 0.25
loss: 0.24999999
loss: 0.24999999
loss: 0.25
predict result: [[0.5], [0.5], [0.5], [0.5]]
```

发生了什么？机器学习在学习异或的时候是失灵的，或者换句话说，对于异或，单层的机器学习并没有学习到任何信息。那么，多层神经网络能否学习到一些新的信息呢？

In [ ]:

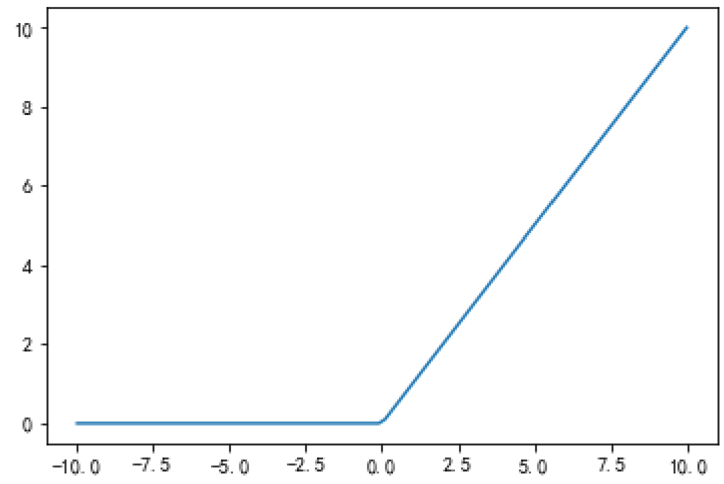
```
一个新的函数：relu函数（整流线型单元，（rectified linear unit）

                        该激活函数是被推荐用于大多数前馈神经网络的默认激活函数。将此
函数用于线性变换的输出将产生非线性变换。然而，函数仍然非常接近线性，在这种意义上它是
具有两个线性部分的分段线性函数。由于整流线性单元几乎是线性的，因此它们保留了许多使得
线性模型易于使用基于梯度的方法进行优化的属性。它们还保留了许多使得线性模型能够泛化良
好的属性。计算机科学的一个通用原则是，我们可以从最小的组件构建复杂的系统。就像图灵机
的内存只需要能够存储0 或1 的状态，我们可以从整流线性函数构建一个万能函数近似器。
```

In [7]:

```
import tensorflow as tf
import numpy as np
with tf.Session() as sess:
    x = np.linspace(-10,10,100)
    y = tf.nn.relu(x)
    res = sess.run(y)
plt.plot(x,res)
```

Out[7]:

```
[<matplotlib.lines.Line2D at 0x274fb67ba58>]
```



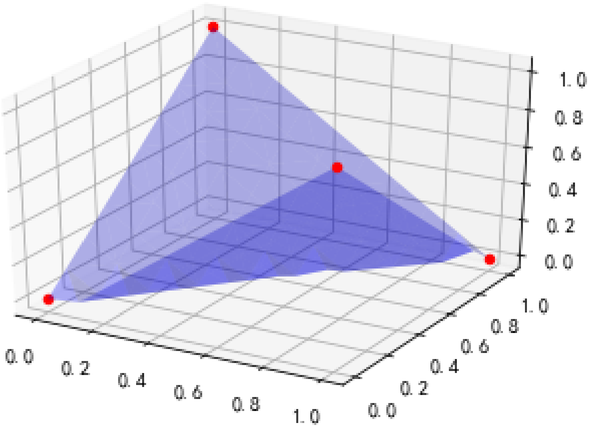
In [30]:

```
#输入数据的处理
tf.reset_default_graph()
tf.logging.set_verbosity(tf.logging.INFO)
x = np.array([[0,0],
             [0,1],
             [1,0],
             [1,1]])
y = np.array([[0],[1],[1],[0]])
# 占位符
x_placeholder = tf.placeholder(tf.float32,shape=[None,2])
y_placeholder = tf.placeholder(tf.float32,shape=[None,1])
# Weight和bias
W_1 = tf.Variable(tf.random_normal([2,2],seed=23))
b_1 = tf.Variable(tf.random_normal([2],seed=46))
# 隐藏层
# y_hidden = tf.matmul(x_placeholder,W_1)+b_1
y_hidden = tf.nn.relu(tf.matmul(x_placeholder,W_1)+b_1)
# Weight和bias
W_2 = tf.Variable(tf.random_normal([2,1],seed=100))
# 预测
y_p = tf.matmul(y_hidden,W_2)

# 定义损失函数，训练方法，初始化变量，并进行训练
loss=tf.reduce_mean(tf.square(y_p-y_placeholder))
train=tf.train.GradientDescentOptimizer(0.1).minimize(loss)
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(10000):
#         if i%1000==0:
#             _loss = sess.run([loss],feed_dict={x_placeholder:x,y_placeholder:y})
#             print("loss:",_loss)
        sess.run(train,feed_dict={x_placeholder:x,y_placeholder:y})
    _y_p = sess.run([y_p,W_1,b_1,W_2],feed_dict={x_placeholder:x,y_placeholder:y})
    print("predict result:",_y_p)
    # 后续作图的准备
    line_y = np.linspace(0,1,10).repeat(10).reshape([10,10])
    line_x = line_y.T
    line_x = line_x.flatten()
    line_y = line_y.flatten()
    x_input = np.hstack([line_x[:,None],line_y[:,None]])
    line_z = sess.run(y_p,feed_dict={x_placeholder:x_input})
```

```
predict result: [array([[4.7683739e-07],
       [9.9999952e-01],
       [9.9999994e-01],
       [0.0000000e+00]], dtype=float32), array([[ 1.0102097, -0.8916067],
       [-0.925912 ,  0.8908013]], dtype=float32), array([-9.5820270e-02,  4.2476776e
-07], dtype=float32), array([[1.0936259],
       [1.1225837]], dtype=float32)]
```

In [31]:

```
from mpl_toolkits.mplot3d import Axes3D,axes3d
import matplotlib.pyplot as plt
import numpy as np



x,y,z = axes3d.get_test_data(0.05)
x,y,z=x.flatten(), y.flatten(), z.flatten()
fig = plt.figure()
ax = fig.gca(projection='3d')

ax.scatter(0,0,0,color='r')
ax.scatter(0,1,1,color='r')
ax.scatter(1,0,1,color='r')
ax.scatter(1,1,0,color='r')
# ax.plot_wireframe(x, y, z, rstride=10, cstride=10)
ax.plot_trisurf(line_x,line_y,line_z.flatten(),alpha=0.3,color='b' )

plt.show()
```



隐藏层成功的学习到了更多的信息，帮助我们更好的完成了分类任务！让我们看看这个神经网络训练出来的三维面是什么样子的

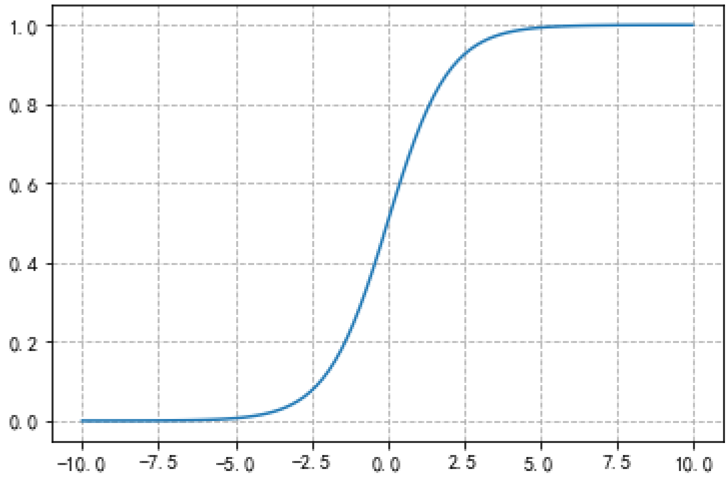
```
一个新的函数：sigmoid函数
这个函数是一个[-inf,inf]到[0,1]上的一个非线性映射，让我们用代码把它画出来看一下
```

In [1]:

```
from __future__ import print_function, division
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
```

In [6]:

```
import tensorflow as tf
import numpy as np
with tf.Session() as sess:
    x = np.linspace(-10,10,100)
    y = tf.nn.sigmoid(x)
    res = sess.run(y)
plt.plot(x,res)
plt.grid(linestyle='--')
# plt.xticks([])
# plt.yticks([])
```



$$y = a \cdot \frac{1}{1 + e^{-b(year-c)}}$$

In [70]:

```
#一个更加有用一点的loss函数:
# loss=z * -log(sigmoid(x)) + (1 - z) * -log(1 - sigmoid(x))
# 其中z是标签，x是logits（对数几率，也就是）的值
# 这个函数的名字叫做交叉熵函数，事实上，逻辑回归就是用的这个激活函数，tensorflow内部已经有了实
# tf.losses.sigmoid_cross_entropy,想了解更多的信息，可以阅读信息论中相关知识
# baike地址:
# https://baike.baidu.com/item/%E4%BA%A4%E5%8F%89%E7%86%B5
```

In [ ]:

```

```