**姓名：杨庆 学号：201822090316**

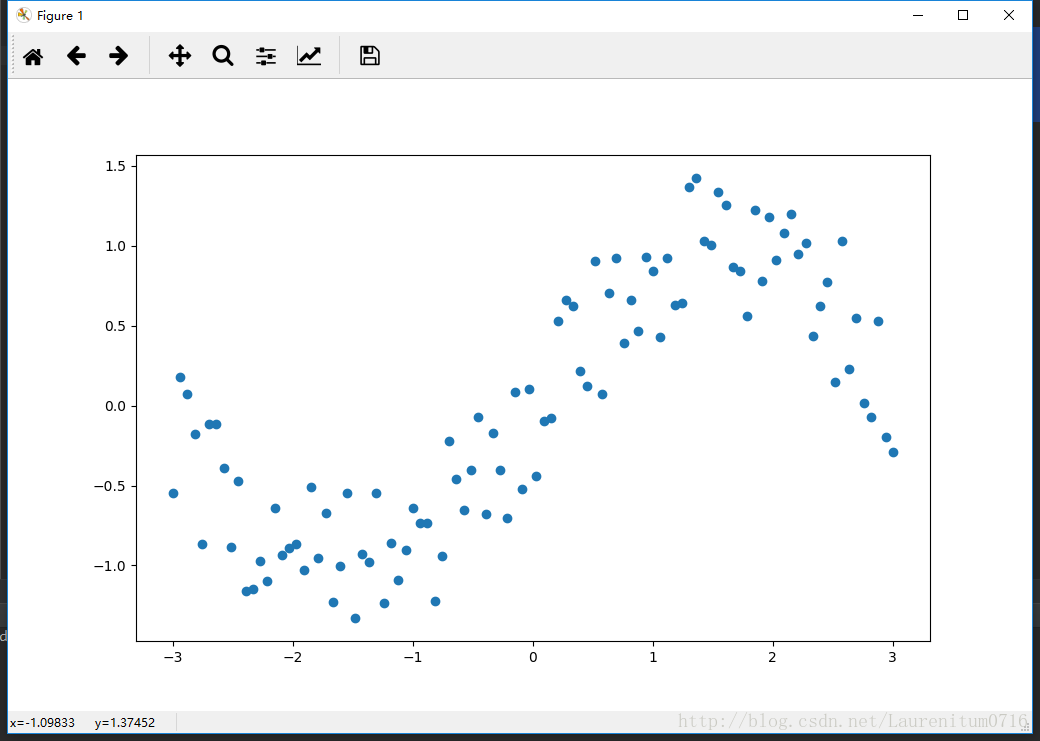
**题目一.多项式拟合正弦函数sin(x)**

线性回归并不能很好的拟合正弦的问题，把线性回归修改为多项式回归。

**（1）数据准备**

在[-3，3]内生成100个点，使用正弦函数上加上了随机噪声。

1. import numpy as np
2. import tensorflow as tf
3. import matplotlib.pyplot as plt
4. n\_observations = 100
5. xs = np.linspace(-3, 3, n\_observations)
6. ys = np.sin(xs) + np.random.uniform(-0.5, 0.5, n\_observations)
7. plt.scatter(xs, ys)
8. plt.show()



**（2）设置placeholder**

1. X = tf.placeholder(tf.float32, name='X')
2. Y = tf.placeholder(tf.float32, name='Y')

**（3）初始化权重参数**

此处使用 y = w3 x^3 + w2 x^2 + w1 x + b进行拟合，此处构造四个变量 w1、w2、w3 与 b。

1. W = tf.Variable(tf.random\_normal([1]), name='weight')
2. B = tf.Variable(tf.random\_normal([1]), name='bias')
3. W\_2 = tf.Variable(tf.random\_normal([1]), name='weight\_2')
4. W\_3 = tf.Variable(tf.random\_normal([1]), name='weight\_3')

**（4）计算预测结果**

y = w3 x^3 + w2 x^2 + w1 x + b

1. Y\_pred = tf.add(tf.multiply(X, W), B)
2. Y\_pred = tf.add(tf.multiply(tf.pow(X, 2), W\_2), Y\_pred)
3. Y\_pred = tf.add(tf.multiply(tf.pow(X, 3), W\_3), Y\_pred)

**（5）计算损失值**

1. loss = tf.reduce\_sum(tf.pow(Y\_pred - Y, 2)) / sample\_num

**（6）初始化optimizer**

梯度下降函数保持不变，使得loss函数的值取到极小值。

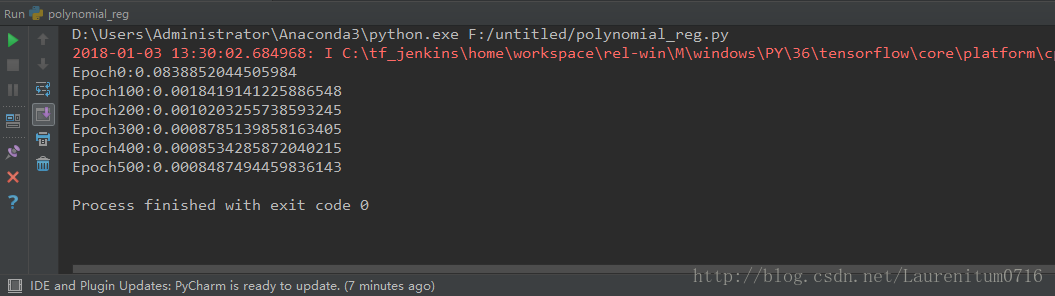
1. learning\_rate = 0.01
2. optimizer = tf.train.GradientDescentOptimizer(learning\_rate).minimize(loss)

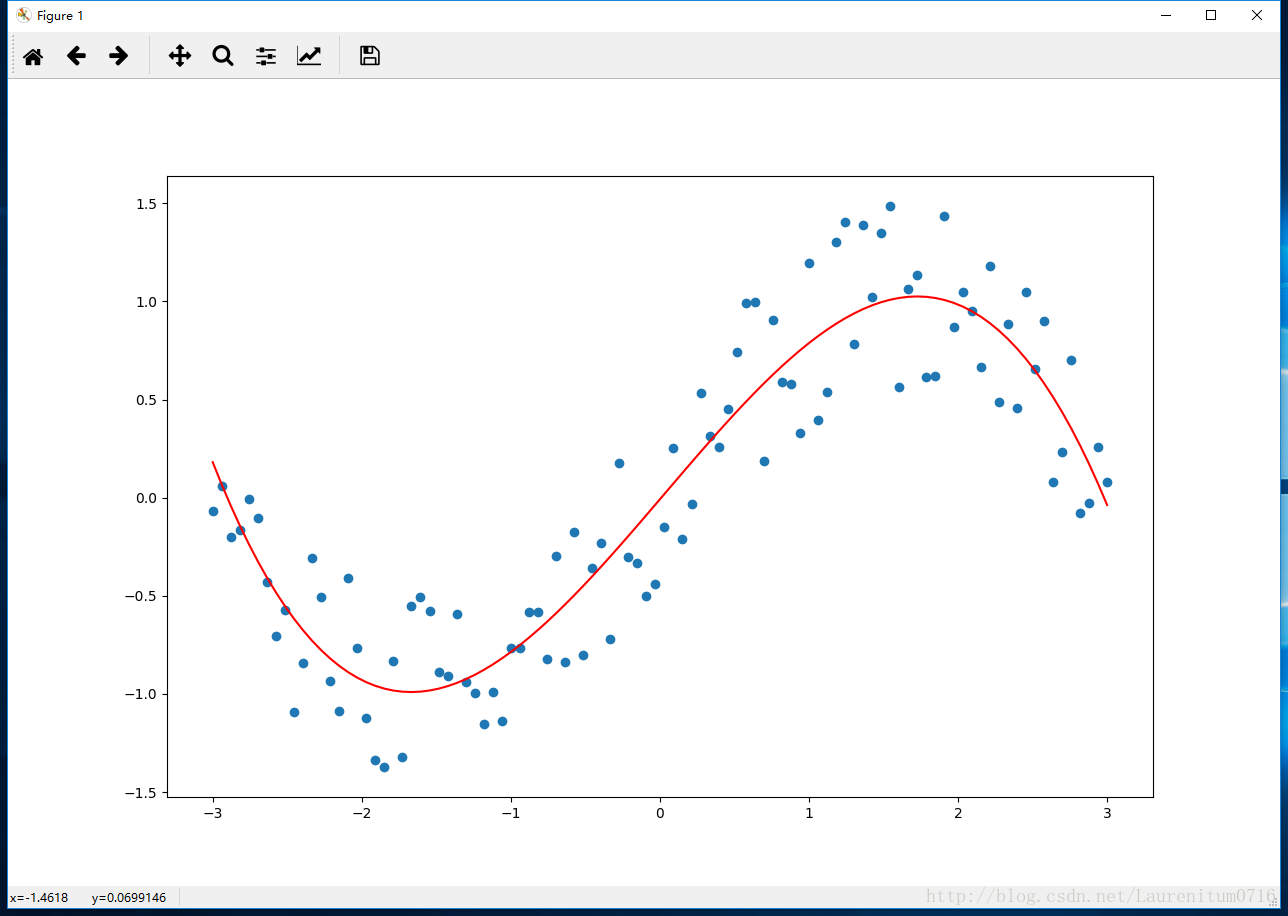
**（7）指定迭代次数，并在session执行graph**

1. n\_sample = xs.shape[0]
2. init = tf.global\_variables\_initializer()
3. with tf.Session() as sess:
4. sess.run(init)
5. for i in range(1000):
6. total\_loss = 0
7. for x, y in zip(xs, ys):
8. \_\_, l = sess.run([optimizer, loss], feed\_dict={X: x, Y: y})
9. total\_loss += l
10. if i % 20 == 0:
11. print('Epoch{0}:{1}'.format(i, total\_loss/n\_sample))
12. W, W\_2, W\_3, B = sess.run([W, W\_2, W\_3, B])
13. plt.scatter(xs, ys)
14. plt.plot(xs, xs\*\*3\*W\_3 + xs\*\*2\*W\_2 + xs\*W + B)
15. plt.show()

**（8）运行结果**

迭代计算500次，从图中可以看到，在使用了三个变量的多项式回归之后，对数据点的拟合程度达到了还不错的效果，将多项式最高次数继续提高，会出现过拟合的情况。这是因为更加灵活的高阶多项式把数据点包含的随机噪音也拟合进去了。为此我们可以增加样本空间，以减弱噪音的影响。





**题目二、TensorFlow实现多层感知机MLP**

多层感知即至少包含一个隐含层的神经网络，较为典型的就是BP神经网络。本文依旧以mnist数据集进行对MLP进行实验，实验中MLP仅包含了一个隐含层。

**（1）载入TensorFlow的并加载MNIST、数据集**

from tensorflow.examples.tutorials.mnist import input\_data

import tensorflow as tf

mnist = input\_data.read\_data\_sets('MNIST\_data/',one\_hot = True)

**（2）指定输入节点数in\_units和隐含层节点数h1\_units**

in\_units = 784 #输入维度

h1\_units = 300 #隐含层维度

**（3）权重和偏置的初始化**

W1 = tf.Variable(tf.truncated\_normal([in\_units,h1\_units],stddev = 0.1))

b1 = tf.Variable(tf.zeros([h1\_units]))

W2 = tf.Variable(tf.zeros([h1\_units,10]))

b2 = tf.Variable(tf.zeros([10]))

**（4）定义计算图的输入**

x = tf.placeholder(tf.float32,[None,in\_units])

y\_ = tf.placeholder(tf.float32,[None,10])

keep\_prob = tf.placeholder(tf.float32)

**（5）定义模型结构(网络的前向过程)**

hidden1 = tf.nn.relu(tf.matmul(x,W1) + b1)

hidden1\_drop = tf.nn.dropout(hidden1,keep\_prob)

y = tf.nn.softmax(tf.matmul(hidden1\_drop,W2) + b2)

**（6）定义损失**

loss = tf.reduce\_mean(-tf.reduce\_sum(y\_ \* tf.log(y),reduction\_indices = [1]))

**（7）定义优化器**

train\_step = tf.train.AdadeltaOptimizer(0.3).minimize(loss)

**（8）初始化各个变量**

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

**（9）网络训练**

for i in range(10000):

batch\_x,batch\_y = mnist.train.next\_batch(100)

train\_step.run(session = sess,feed\_dict = {x:batch\_x,y\_:batch\_y,keep\_prob:0.5})

**（10）模型评估**

correct\_prediction = tf.equal(tf.argmax(y,1),tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction,tf.float32))

print(accuracy.eval(session=sess,feed\_dict= {x:mnist.test.images,y\_:mnist.test.labels,keep\_prob:1.0}))

1. **运行结果**

可以看出在测试集上达到95%的准确率。

