线性回归问题TensorFlow实战

核心步骤

使用Tensorflow进行算法设计与训练的核心步骤

- （1）准备数据
- （2）构建模型
- （3）训练模型
- （4）进行预测

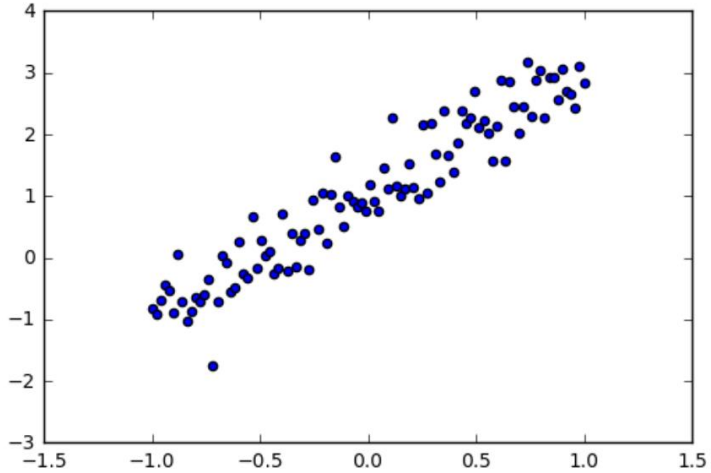
上述步骤是我们使用Tensorflow进行算法设计与训练的核心步骤，贯穿于后面介绍的具体实战中。本章用一个简单的例子来讲解这几个步骤。

线性方程

单变量的线性方程可以 表示为：

$$y = w*x + b$$

$$y = 2.0*x + 1$$



本例通过生成人工数据集。随机生成一个近似采样随机分布，使得w=2.0， b=1， 并加入一个噪声，噪声的最大振幅为0.4

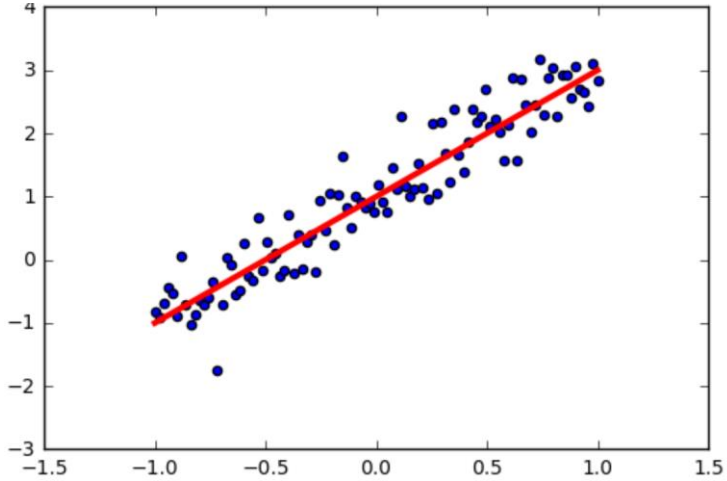
人工数据集生成

```
# 在Jupyter中，使用matplotlib显示图像需要设置为 inline 模式，否则不会现实图像
%matplotlib inline

import matplotlib.pyplot as plt # 载入matplotlib
import numpy as np # 载入numpy
import tensorflow as tf # 载入Tensorflow

# 设置随机数种子
np.random.seed(5)
```

```
# 直接采用np生成等差数列的方法，生成100个点，每个点的取值在-1~1之间

x_data = np.linspace(-1, 1, 100)

# y = 2x +1 + 噪声，  其中，噪声的维度与x_data一致

y_data = 2 * x_data + 1.0 + np.random.randn(*x_data.shape) * 0.4
```

利用matplotlib画出生成结果

```
#画出随机生成数据的散点图
plt.scatter(x_data, y_data)

# 画出我们想要学习到的线性函数 y = 2x +1

plt.plot (x_data, 2 * x_data + 1.0, color = 'red',linewidth=3)
```



构建模型

定义训练数据的占位符，x是特征值，y是标签值

```
x = tf.placeholder("float", name = "x")
y = tf.placeholder("float", name = "y")
```

定义模型函数

```
def model(x, w, b):
    return tf.multiply(x, w) + b
```

定义模型结构

创建变量

- Tensorflow变量的声明函数是tf.Variable
- tf.Variable的作用是保存和更新参数
- 变量的初始值可以是随机数、常数，或是通过其他变量的初始值计算得到

```
# 构建线性函数的斜率，变量w
w = tf.Variable(1.0, name="w0")

# 构建线性函数的截距，变量b
b = tf.Variable(0.0, name="b0")
```

```
# pred是预测值，前向计算
pred = model(x, w, b)
```

训练模型

设置训练参数

```
# 迭代次数（训练轮数）
train_epochs = 10

# 学习率
learning_rate = 0.05
```

定义损失函数

- 损失函数用于描述预测值与真实值之间的误差，从而指导模型收敛方向
- 常见损失函数：均方差（Mean Square Error, MSE）和交叉熵（cross-entropy）

L_2 损失函数

```
# 采用均方差作为损失函数

loss_function = tf.reduce_mean(tf.square(y-pred))
```

定义优化器

定义优化器Optimizer，初始化一个GradientDescentOptimizer

设置学习率和优化目标：最小化损失

```
# 梯度下降优化器

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss_function)
```

创建会话

声明会话

```
sess = tf.Session()
```

变量初始化

- 在真正执行计算之前，需将所有变量初始化
- 通过 **tf.global_variables_initializer** 函数可实现对所有变量的初始化

```
init = tf.global_variables_initializer()

sess.run(init)
```

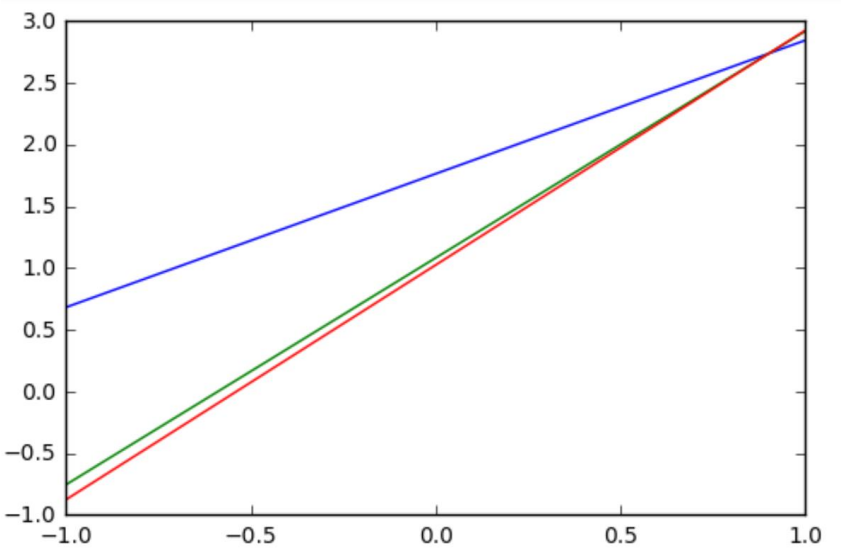
迭代训练

模型训练阶段，设置迭代轮次，每次通过将样本逐个输入模型，进行梯度下降优化操作

每轮迭代后，绘制出模型曲线

```
# 开始训练，轮数为 epoch，采用SGD随机梯度下降优化方法

for epoch in range(train_epochs):
    for xs,ys in zip(x_data, y_data):
        _, loss=sess.run([optimizer,loss_function], feed_dict={x: xs, y: ys})
    b0temp=b.eval(session=sess)
    w0temp=w.eval(session=sess)
    plt.plot (x_data, w0temp * x_data + b0temp  )# 画图
```

迭代训练结果图形



从上图可以看出，本案例所拟合的模型较简单，训练3次之后已经接近收敛
对于复杂模型，需要更多次训练才能收敛

结果查看

当训练完成后，打印查看参数

打印结果

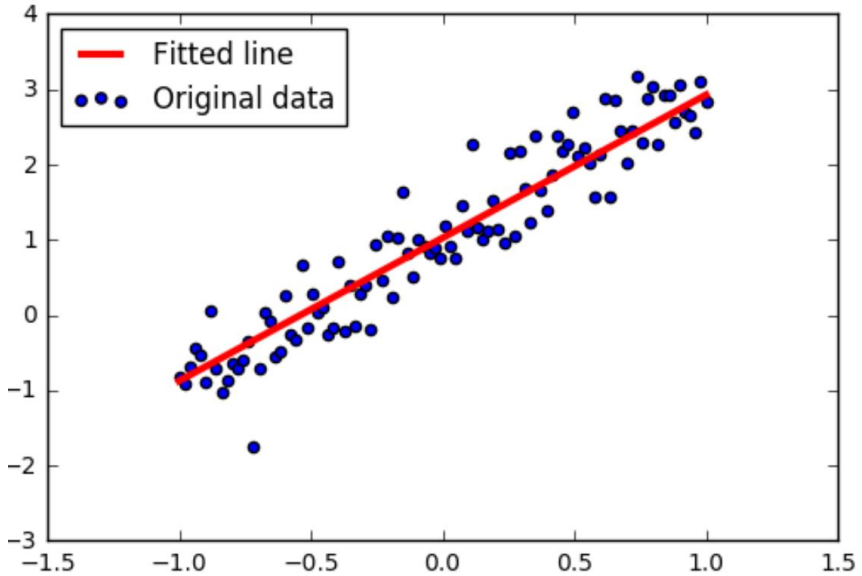
```
print ("w: ", sess.run(w)) # w的值应该在2附近
print ("b: ", sess.run(b)) # b的值应该在1附近
```

```
w:  1.90116
b:  1.02581
```

* 数据每次运行都可能会有所不同

结果可视化

```
plt.scatter(x_data,y_data,label='Original data')
plt.plot (x_data, x_data * sess.run(w) + sess.run(b),\
          label='Fitted line',color='r', linewidth=3)
plt.legend(loc=2)# 通过参数loc指定图例位置
```



利用模型 进行预测

```
x_test = 3.21

predict = sess.run(pred, feed_dict={x: x_test})
print("预测值：%f" % predict)

target = 2 * x_test + 1.0
print("目标值：%f" % target)
```

```
预测值：7.128515
目标值：7.420000
```

小结

通过一个简单的例子介绍了利用Tensorflow实现机器学习的思路，重点讲解了下述步骤：

（1）生成人工数据集及其可视化

（2）构建线性模型

（3）定义损失函数

（4）定义优化器、最小化损失函数

（5）训练结果的可视化

（6）利用学习到的模型进行预测