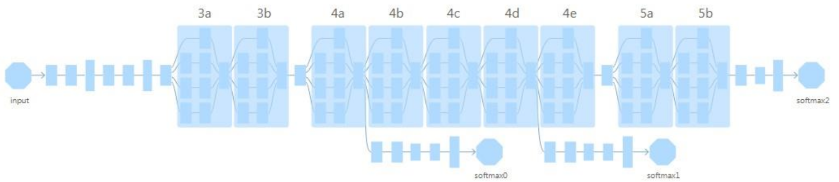


深度学习应用开发

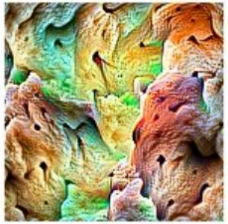
基于TensorFlow的实践

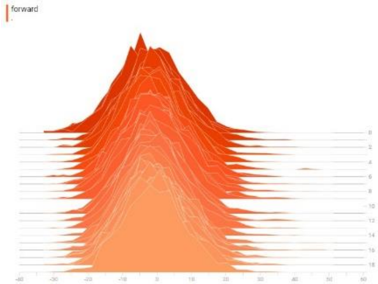
吴明晖 李卓蓉 金苍宏

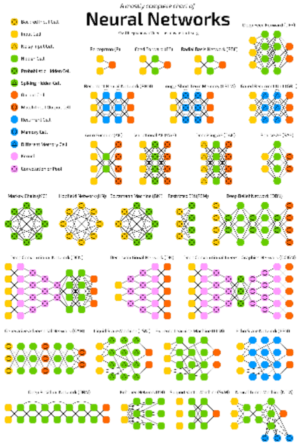
浙江大学城市学院

计算机与计算科学学院

Dept. of Computer Science
Zhejiang University City College

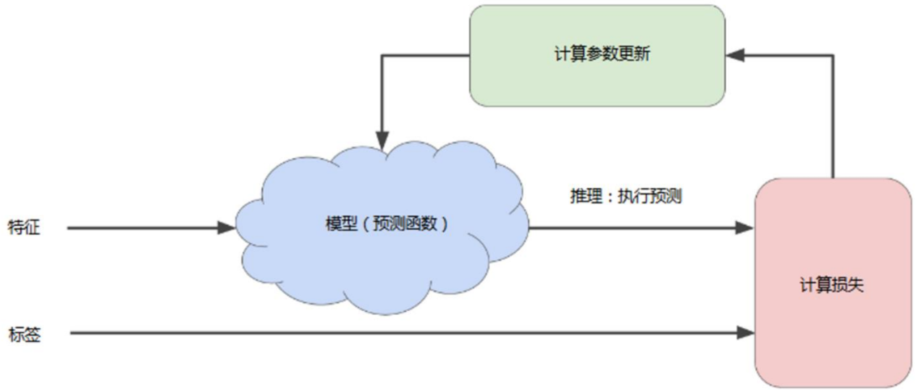


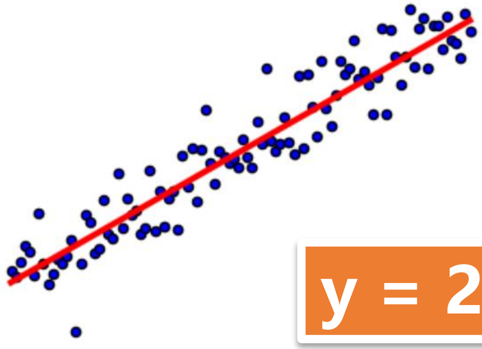




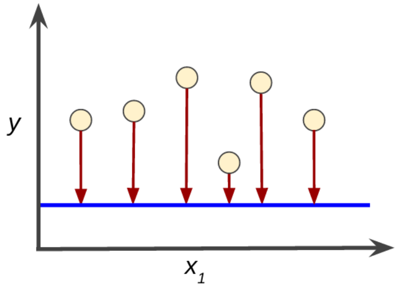
线性回归 TensorFlow实战

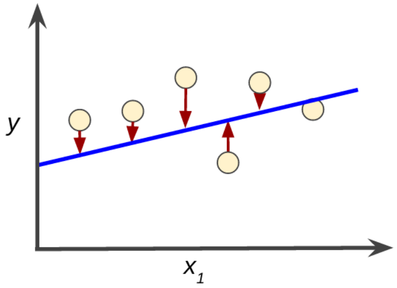
线性回归问题？用一个神经元搞定！

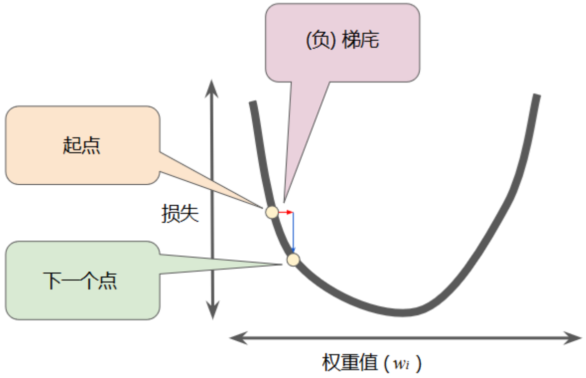




$y = 2x + 1$







监督式机器学习

监督式机器学习

机器学习系统：

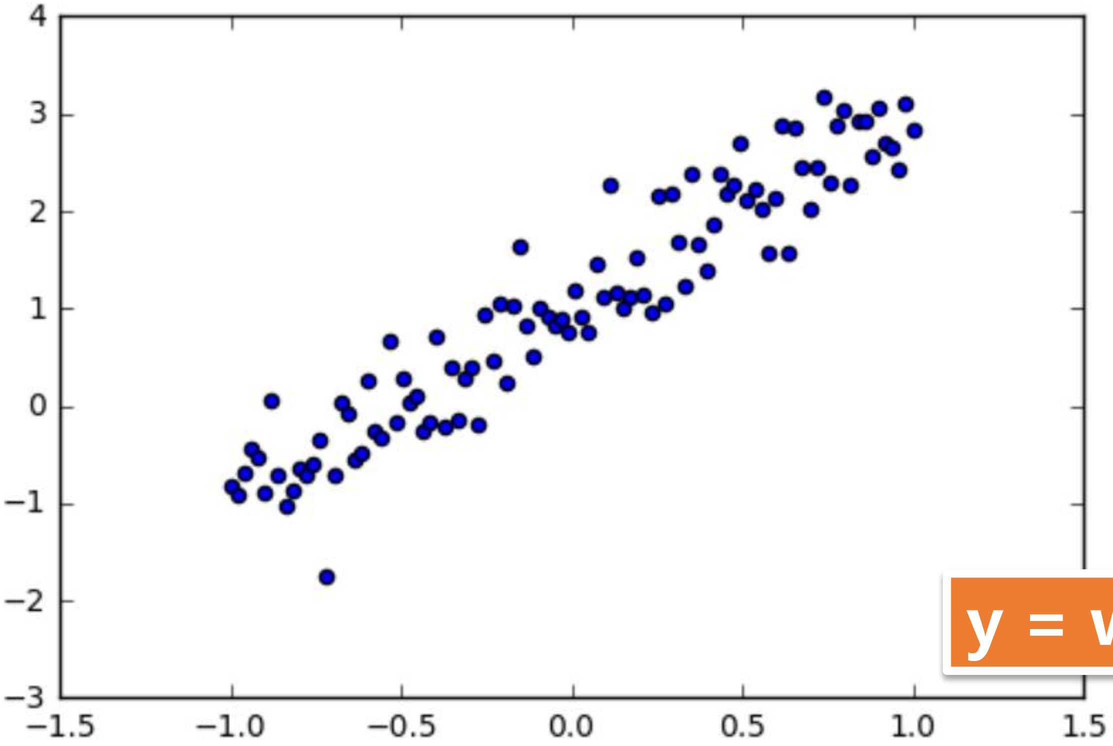
通过学习如何组合输入信息

来对未见过的数据

做出有用的预测

本讲课程部分内容基于“机器学习速成课程”
https://developers.google.cn/machine-learning/crash-course/

简单的线性回归案例



y = w*x + b

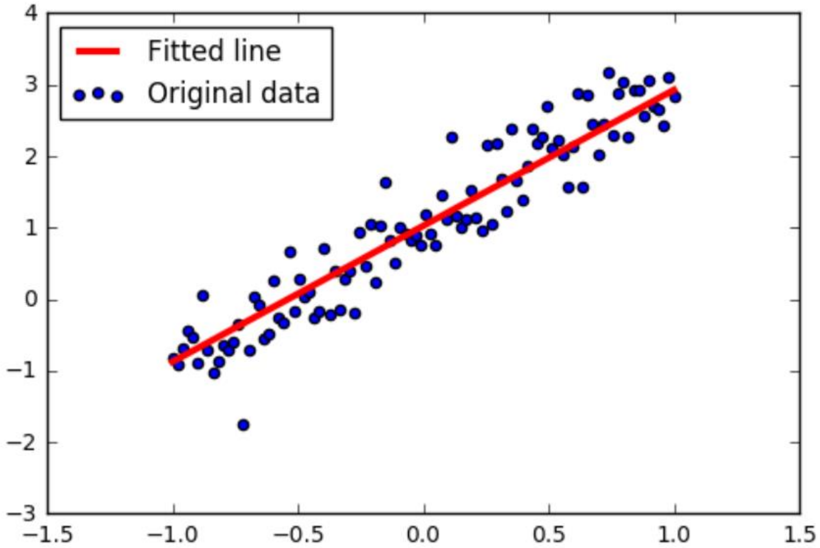
术语：标签和特征

标签是我们要预测的真实事物：y

线性回归中的y变量

特征是指用于描述数据的输入变量：x_i

线性回归中的$\{x_1, x_2, \cdots, x_n\}$变量



术语：样本和模型

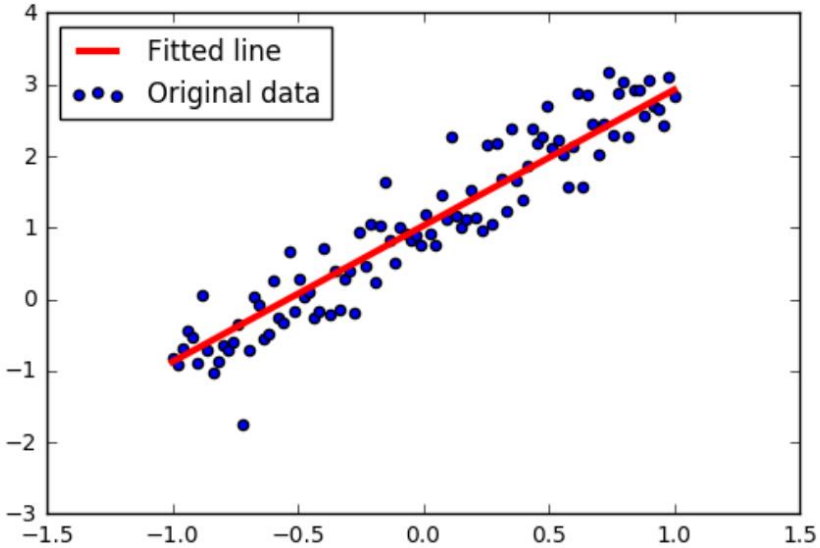
样本是指数据的特定实例：x

有标签样本具有{特征，标签}：{x，y}

用于训练模型

无标签样本具有{特征，？}：{x，?}

用于对新数据做出预测



术语：样本和模型

样本是指数据的特定实例：x

有标签样本具有{特征，标签}：{x，y}

用于训练模型

无标签样本具有{特征，？}：{x，?}

用于对新数据做出预测

模型可将样本映射到预测标签：y’

由模型的内部参数定义，这些内部参数值是通过学习得到的

术语：训练

训练模型表示通过**有标签样本**来学习（确定）所有**权重**和**偏差**的理想值

在监督式学习中，机器学习算法通过以下方式构建模型：

检查多个样本并尝试找出可最大限度地**减少损失**的模型

这一过程称为**经验风险最小化**

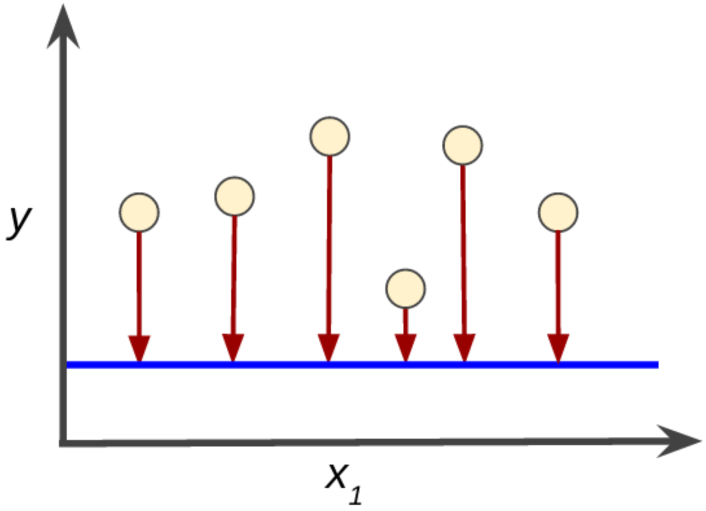
术语：损失

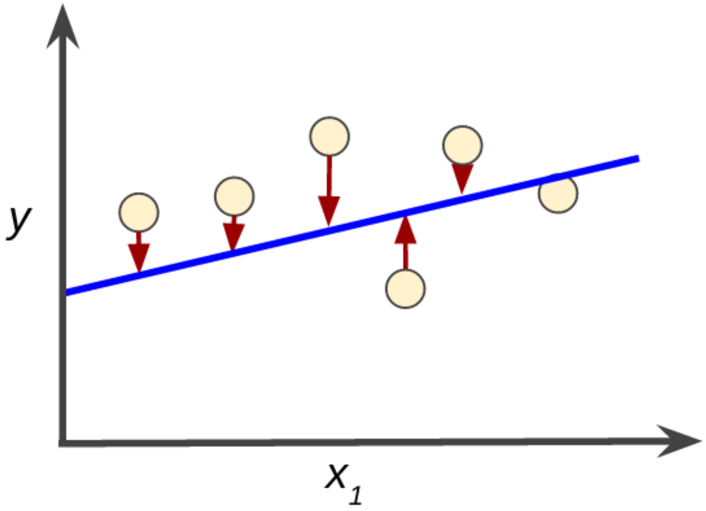
损失是对糟糕预测的惩罚：**损失**是一个数值，表示对于单个样本而言模型预测的准确程度

如果模型的预测完全准确，则损失为零，否则损失会较大

训练模型的目标是从所有样本中找到一组平均损失“较小”的权重和偏差

损失





左侧模型的**损失较大**　　　　右侧模型的**损失较小**

定义损失函数

L_1损失：基于模型预测的值与标签的实际值之差的绝对值

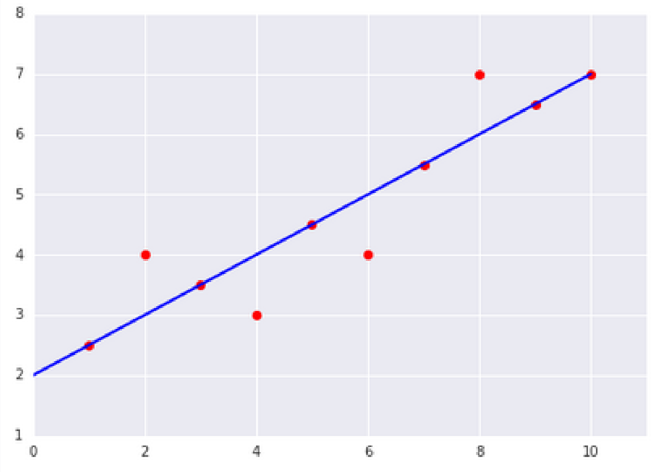
平方损失：一种常见的损失函数，又称为 L_2 损失

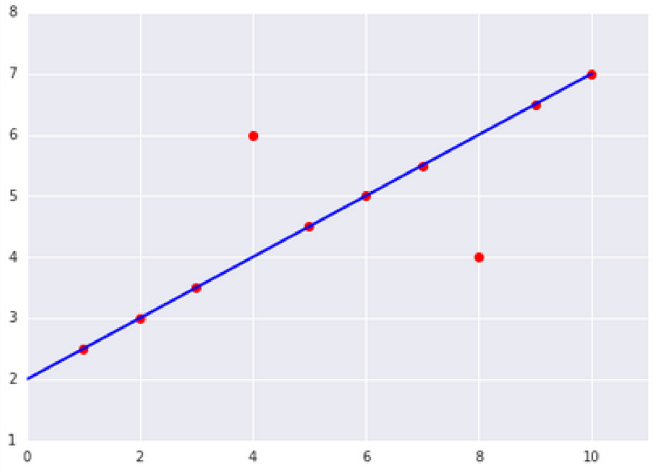
均方误差（MSE） 指的是每个样本的平均平方损失

$$MSE = \frac{1}{N} \sum_{(x,y)\in D} (y - prediction(x))^2$$

问题

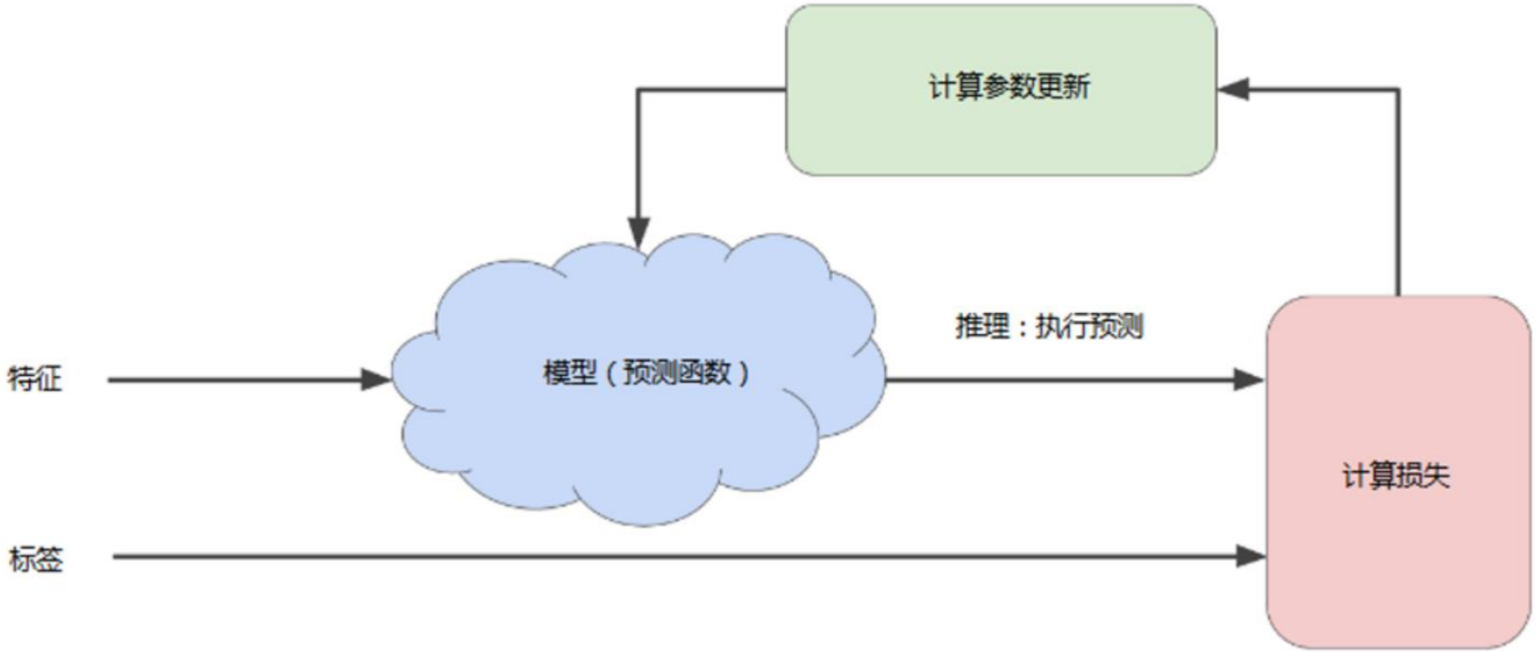
以下曲线图中显示的两个数据集，哪个数据集的均方误差（MSE）**较高**？





模型训练与降低损失

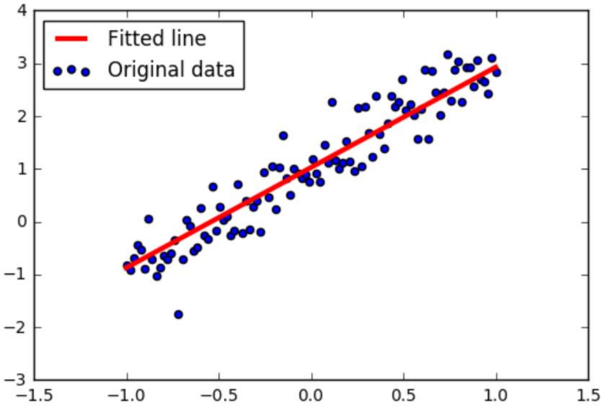
训练模型的迭代方法

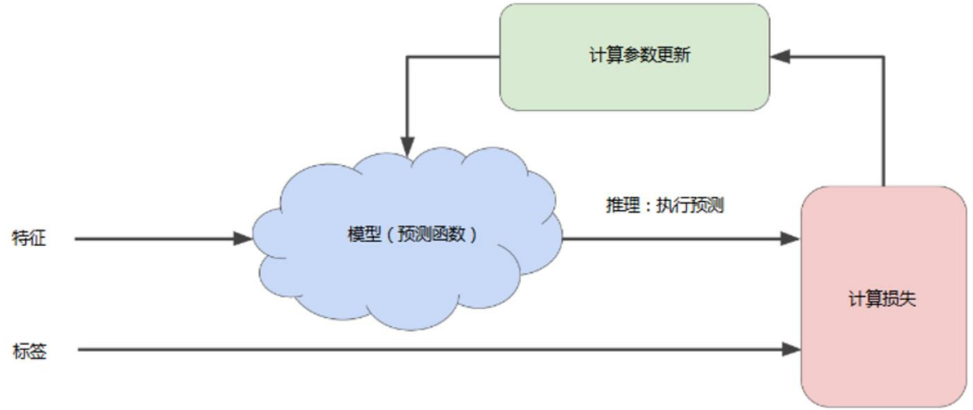


要点

模型训练要点

- 首先对权重w和偏差b进行初始猜测
- 然后反复调整这些猜测
- 直到获得损失可能最低的权重和偏差为止

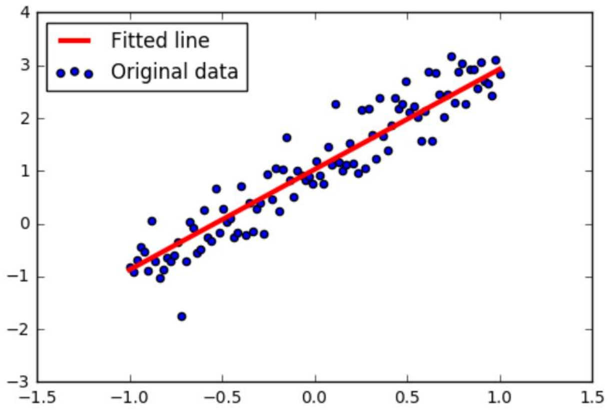


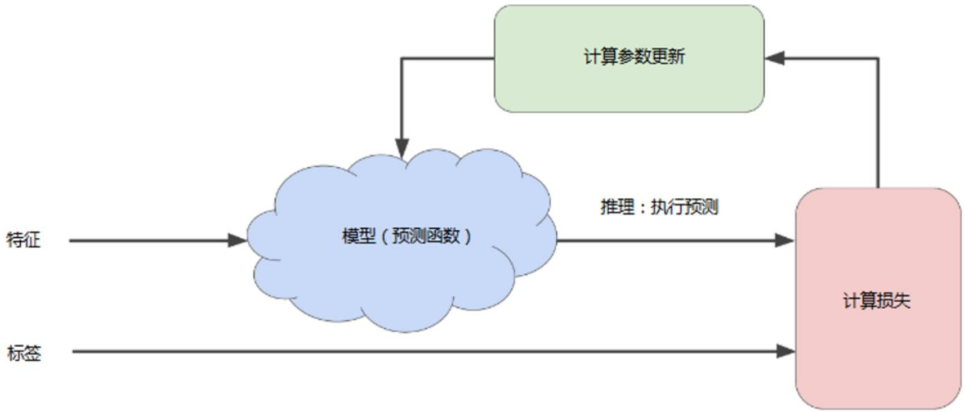


收敛

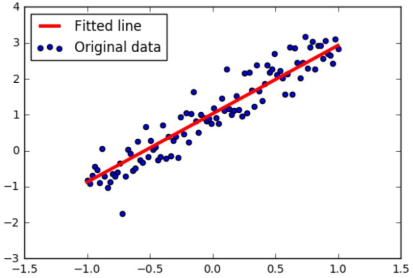
在学习优化过程中，机器学习系统将根据所有标签去重新评估所有特征，为损失函数生成一个新值，而该值又产生新的参数值。

通常，您可以不断迭代，直到总体损失不再变化或至少变化极其缓慢为止。这时候，我们可以说该模型已**收敛**



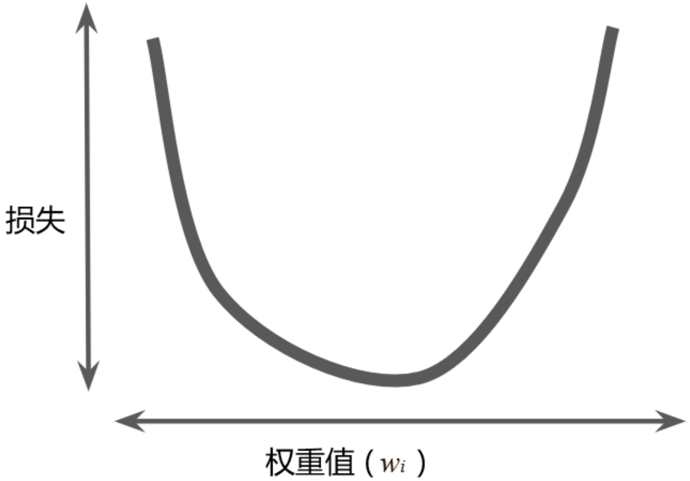


计算损失例子



$$MSE = \frac{1}{N} \sum_{(x,y)\in D} (y - prediction(x))^2$$

pred = w*x + b



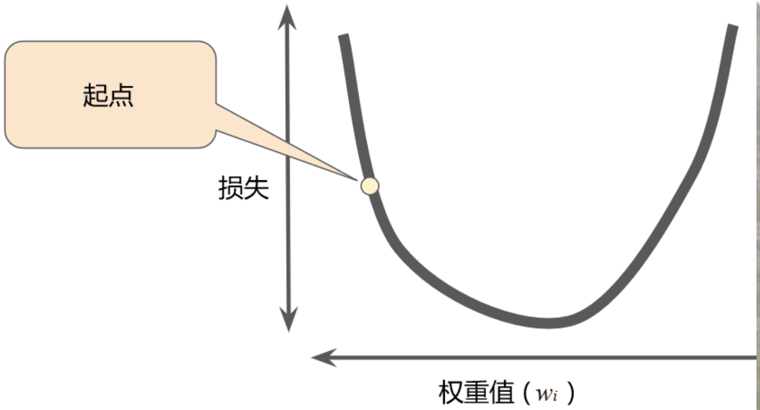
该线性回归问题产生的损失与权重图为**凸形**

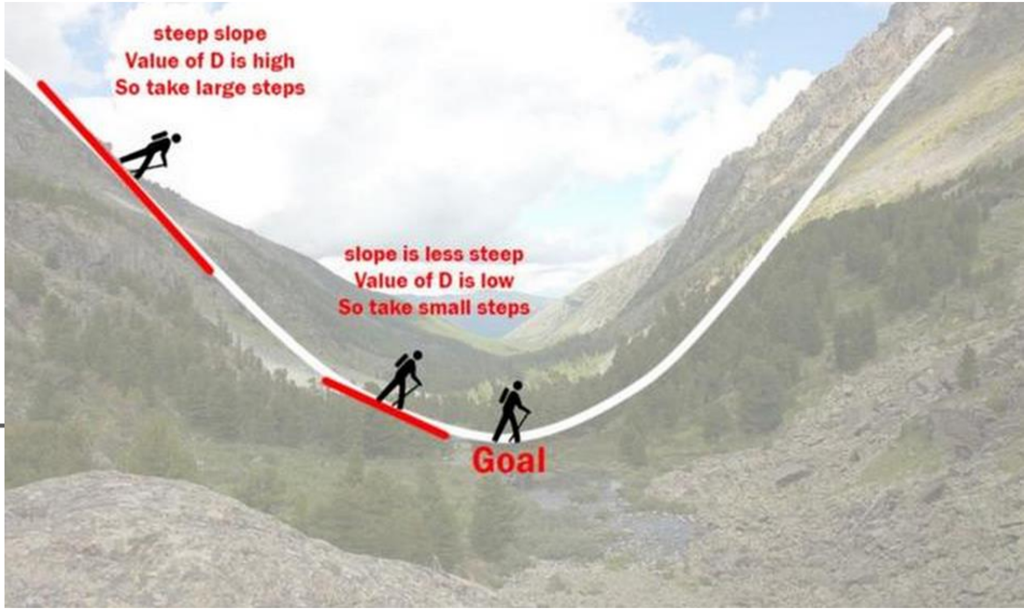
凸形问题只有**一个最低点**；即只存在一个**斜率正好为 0** 的位置

这个最小值就是损失函数收敛之处

梯度下降法

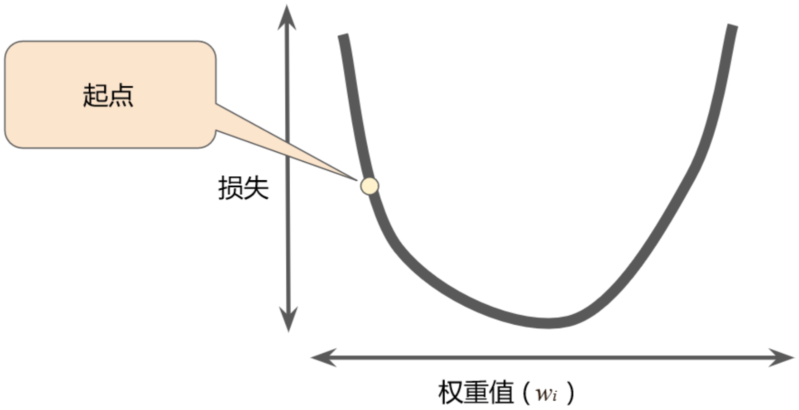
梯度下降法



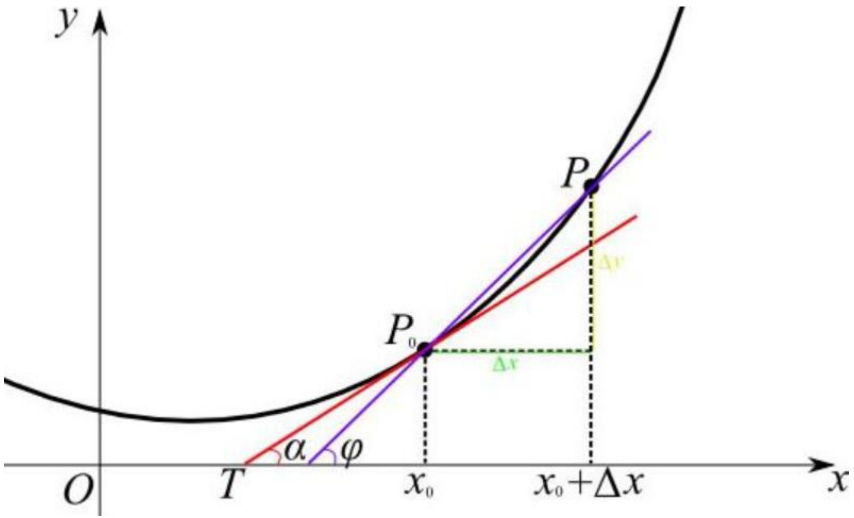


梯度：一个向量（矢量），表示某一函数在该点处的方向**导数**沿着该方向取得**最大值**，即函数在该点处沿着该方向（此梯度的方向）变化最快，变化率最大

梯度下降法



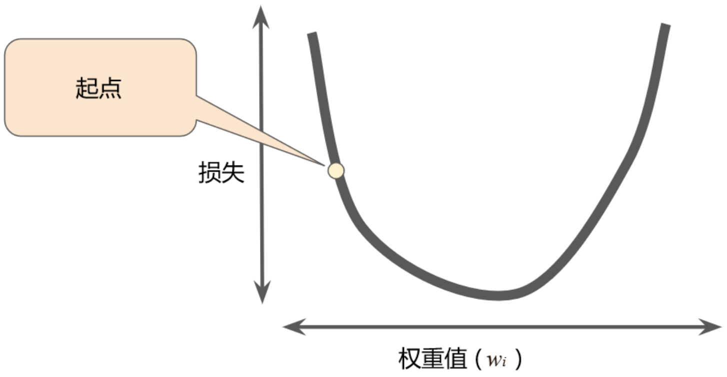
$$f'(x_0)=\lim_{\Delta x\to 0}\frac{\Delta y}{\Delta x}=\lim_{\Delta x\to 0}\frac{f(x_0+\Delta x)-f(x_0)}{\Delta x}$$

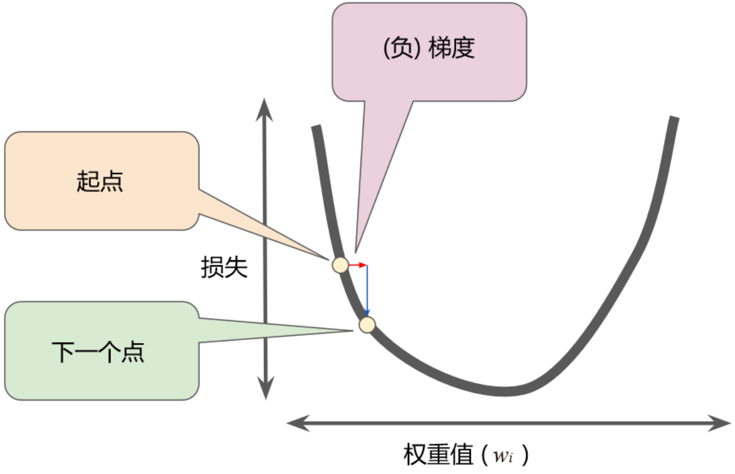


梯度：一个向量（矢量），表示某一函数在该点处的方向**导数**沿着该方向取得**最大值**，即函数在该点处沿着该方向（此梯度的方向）变化最快，变化率最大

梯度下降法

梯度是矢量：具有**方向**和**大小**





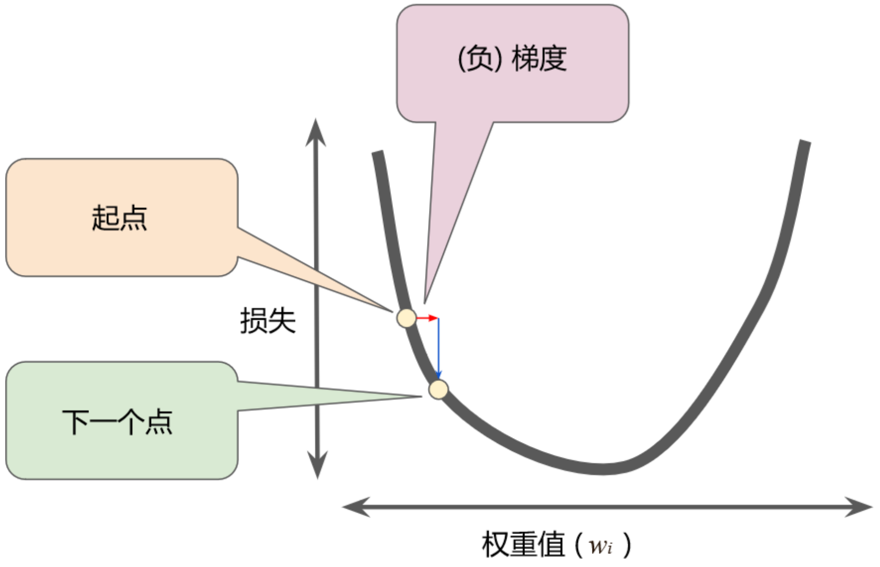
沿着**负梯度方向**进行下一步探索

学习率

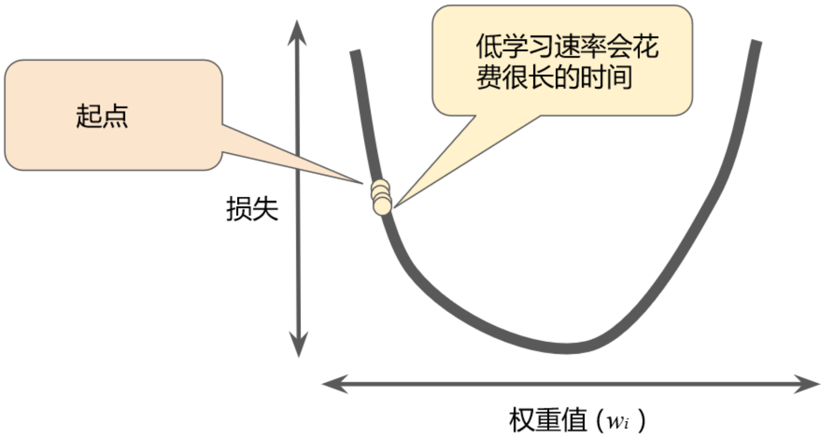
沿着负梯度方向进行下一步探索，**前进多少合适？**

用梯度度乘以一个称为**学习速率**（有时也称为**步长**）的标量，以确定下一个点的位置

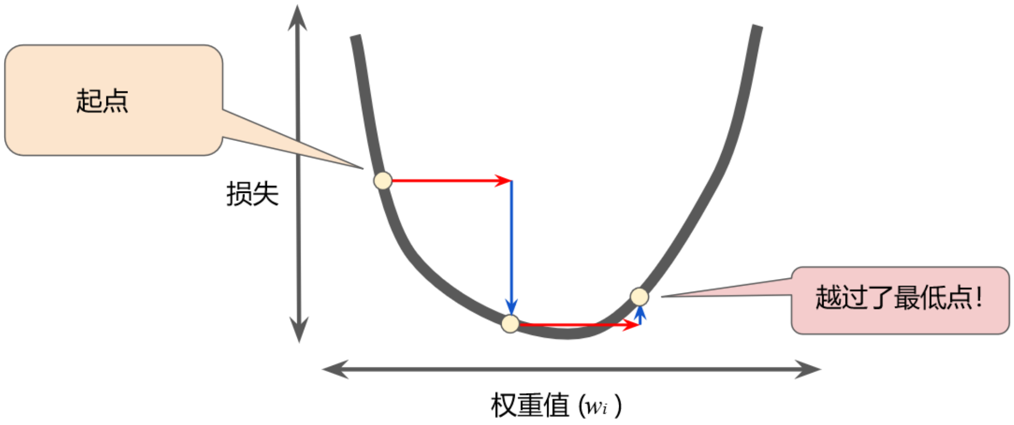
例如：如果梯度大小为 2.5，学习速率为 0.01，则梯度下降法算法会选择距离前一个点 0.025 的位置作为下一个点



学习率

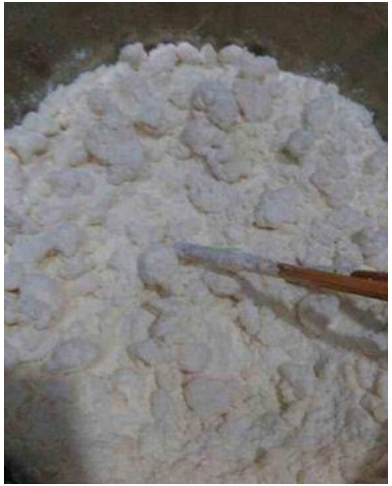


梯度学习速率过小

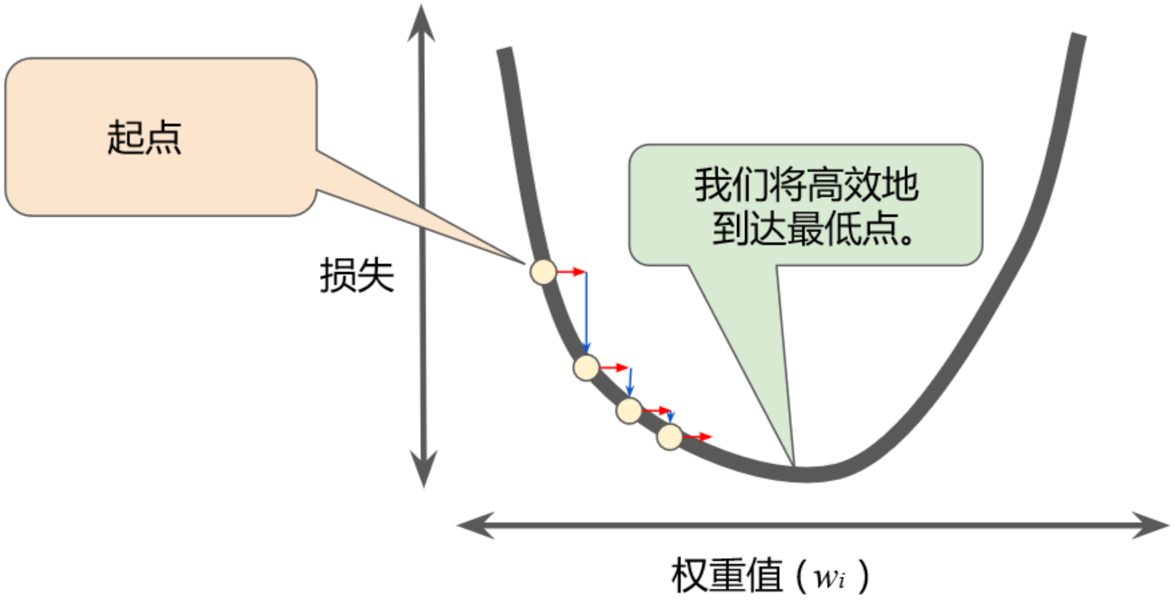


梯度学习速率过大

学习率



学习率



超参数

在机器学习中，超参数是在开始学习过程之前设置值的参数，而不是通过训练得到的参数数据

通常情况下，需要对超参数进行优化，选择一组好的超参数，可以提高学习的性能和效果

超参数是编程人员在机器学习算法中用于调整的旋钮

典型超参数：学习率、神经网络的隐含层数量……

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

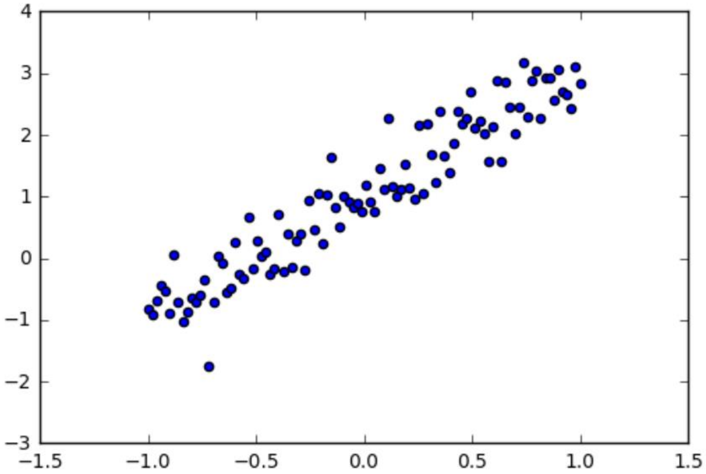
上述步骤是我们使用Tensorflow进行算法设计与训练的核心步骤，贯穿于后面介绍的具体实战中。本章用一个简单的例子来讲解这几个步骤。

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

导入相关库

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

# 在Jupyter中，使用matplotlib显示图像需要设置为 inline 模式，否则不会在网页里显示图像
%matplotlib inline
print("Tensorflow版本是：",tf.__version__) #显示当前TensorFlow版本
```

```
Tensorflow版本是： 2.0.0
```

人工数据集生成

生成数据集

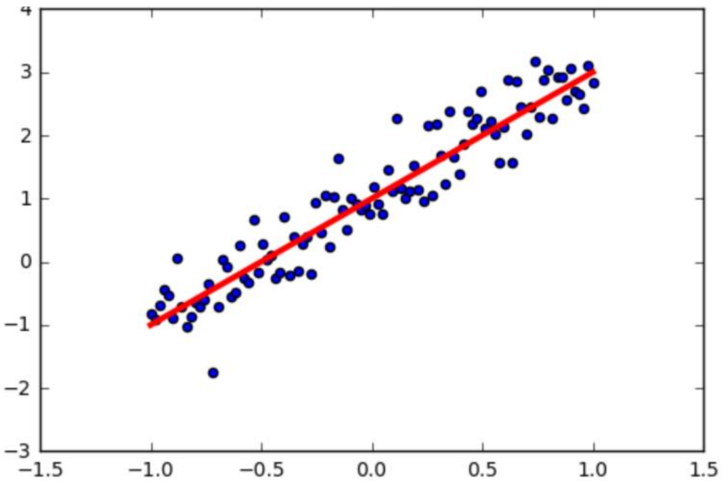
** 首先，生成输入数据。 **

我们需要构造满足这个函数的x和y同时加入一些不满足方程的噪声.

```
# 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)

np.random.seed(5)     # 设置随机数种子
# 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)

# 画出我们想要通过学习得到的目标线性函数 y = 2x +1
plt.plot (x_data, 1.0 + 2 * x_data, 'r',linewidth=3)
```



构建回归模型

```python
# 通过模型执行，将实现前向计算（预测计算）

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

创建变量

创建待优化变量

- 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

```
# 构建模型中的变量w，对应线性函数的斜率
w = tf.Variable(np.random.randn(), tf.float32)

# 构建模型中的变量b，对应线性函数的截距
b = tf.Variable(0.0, tf.float32)
```

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）

```
# 定义均方差损失函数

def loss(x, y, w, b):
    err = model(x, w, b) - y     #  计算模型预测值和标签值的差异
    squared_err = tf.square(err)     #  求平方，得出方差
    return tf.reduce_mean(squared_err)    # 求均值，得出均方差.
```

训练模型

设置训练超参数

```
training_epochs = 10      # 迭代次数（训练轮数）
learning_rate = 0.01      # 学习率
```

定义计算梯度函数

定义计算梯度函数

```
# 计算样本数据[x,y]在参数[w,b]点上的梯度
def grad(x, y, w, b):
    with tf.GradientTape() as tape:
        loss_ = loss(x, y, w, b)
    return tape.gradient(loss_, [w, b])     # 返回梯度向量
```

在 TensorFlow 2 中，使用 tf.GradientTape() 这一上下文管理器封装需要求导的计算步骤，并使用其 gradient() 方法求导。

执行训练

```
step = 0     # 记录训练步数
loss_list = []     # 用于保存loss值的列表
display_step = 10     # 控制训练过程数据显示的频率，不是超参数

for epoch in range(training_epochs):
    for xs,ys in zip(x_data, y_data):

        loss_ = loss(xs, ys, w, b)     # 计算损失
        loss_list.append(loss_)     # 保存本次损失计算结果

        delta_w, delta_b = grad(xs, ys, w, b)     # 计算该当前[w,b]点的梯度
        change_w = delta_w * learning_rate     # 计算变量w需要调整的量
        change_b = delta_b * learning_rate     # 计算变量b需要调整的量
        w.assign_sub(change_w) # 变量w值变更为减去chage_w后的值
        b.assign_sub(change_b) # 变量b值变更为减去chage_b后的值

        step=step+1     # 训练步数+1
        if step % display_step == 0:     # 显示训练过程信息
            print("Training Epoch:", '%02d' % (epoch+1), "Step: %03d" % (step),"loss=%.6f" % (loss_))
    plt.plot(x_data, w.numpy() * x_data + b.numpy())     # 完成一轮训练后，画出回归的线条
```

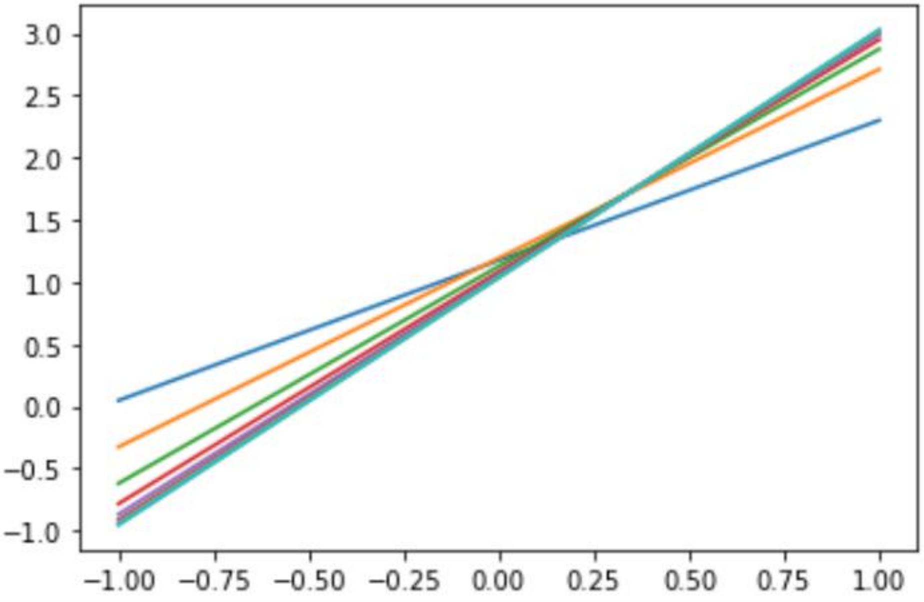
训练输出

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

```
Training Epoch: 01 Step: 010 loss=0.014715
Training Epoch: 01 Step: 020 loss=0.253290
Training Epoch: 01 Step: 030 loss=0.143054
Training Epoch: 01 Step: 040 loss=1.389589
Training Epoch: 01 Step: 050 loss=0.382215
Training Epoch: 01 Step: 060 loss=0.642117
Training Epoch: 01 Step: 070 loss=1.355959
Training Epoch: 01 Step: 080 loss=0.618715
Training Epoch: 01 Step: 090 loss=0.275236
Training Epoch: 01 Step: 100 loss=0.877006
Training Epoch: 02 Step: 110 loss=0.614993
Training Epoch: 02 Step: 120 loss=0.005396
```

```
Training Epoch: 10 Step: 970 loss=0.079192
Training Epoch: 10 Step: 980 loss=0.013070
Training Epoch: 10 Step: 990 loss=0.110992
Training Epoch: 10 Step: 1000 loss=0.032439
```

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5轮之后已经接近收敛

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

显示训练结果

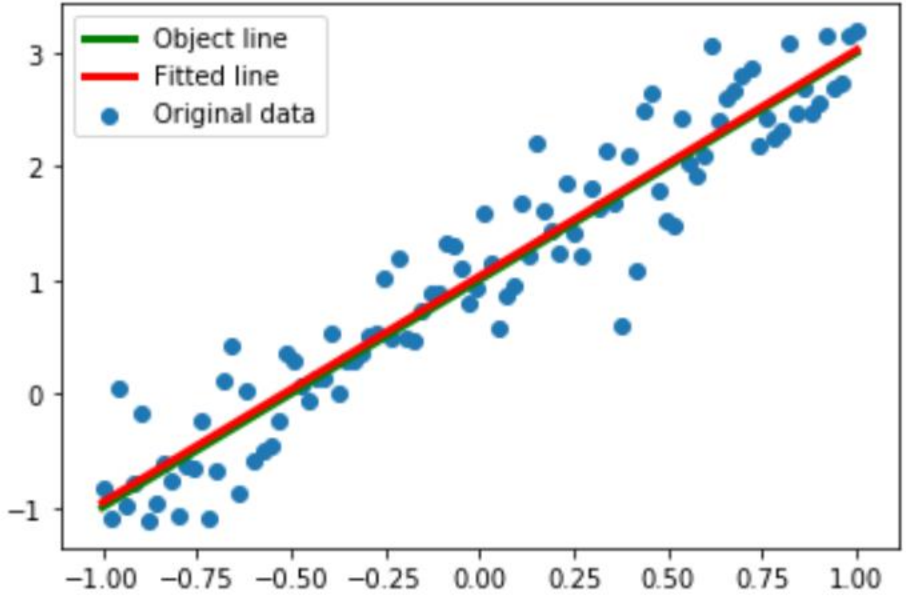
```
print ("w: ", w.numpy()) # w的值应该在2附近
print ("b: ", b.numpy()) # b的值应该在1附近
```

```
w:    1.99055
b:    1.0367402
```

＊ 数据每次运行都可能会有所不同

结果可视化

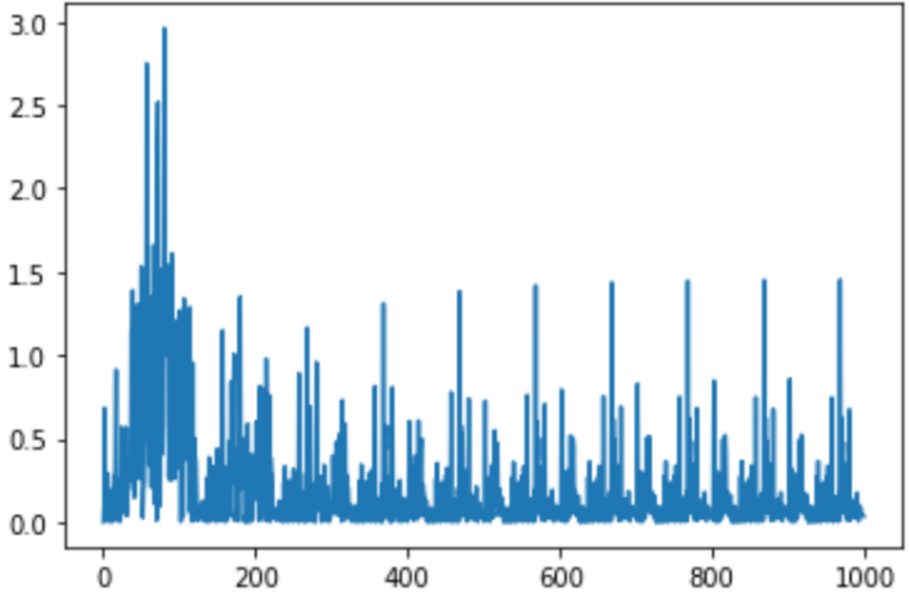
```
plt.scatter(x_data,y_data,label='Original data')
plt.plot (x_data, x_data * 2.0 + 1.0, label='Object line',color='g', linewidth=3)
plt.plot (x_data, x_data * w.numpy() + b.numpy(),label='Fitted line',color='r', linewidth=3)
plt.legend(loc=2)# 通过参数loc指定图例位置
```



损失值得可视化

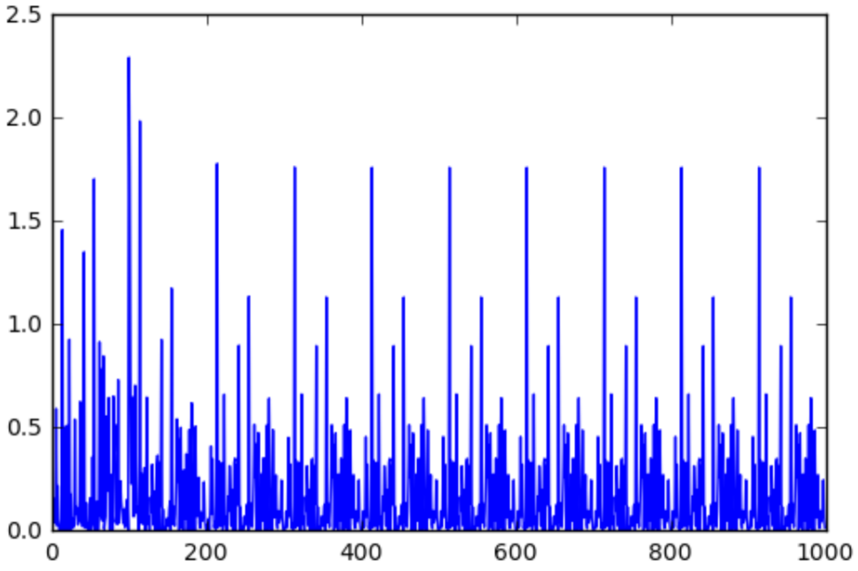
查看损失变化情况

```
plt.plot(loss_list)
```

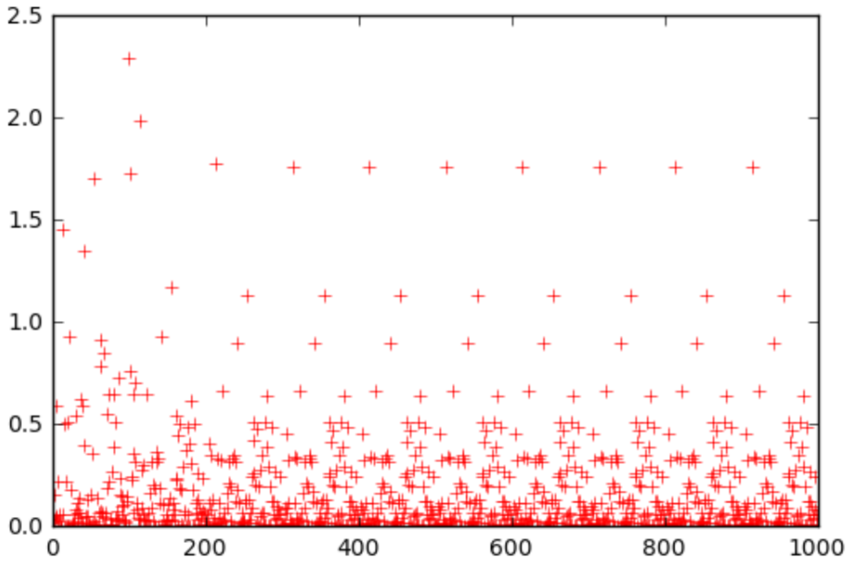


图形化显示损失值

```
plt.plot(loss_list)
```




```
plt.plot(loss_list,'r+')
```



利用模型 进行预测

浙江大学城市学院 ZHEJIANG UNIVERSITY CITY COLLEGE

进行预测

```
x_test = 3.21

predict = model(x_test,w.numpy(),b.numpy())
print("预测值: %f" % predict)

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

```
预测值: 7.426406
目标值: 7.420000
```

随机梯度下降

随机梯度下降法（SGD）每次迭代只使用一个样本（批量大小为 1），如果进行足够的迭代，SGD 也可以发挥作用。“随机”这一术语表示构成各个批量的一个样本都是随机选择的

在梯度下降法中，**批量**指的是用于在单次迭代中计算梯度的样本总数

假定批量是指整个数据集，数据集通常包含很大样本（数万甚至数千亿），此外， 数据集通常包含多个特征。因此，一个批量可能相当巨大。如果是超大批量，则单次迭代就可能要花费很长时间进行计算

小批量随机梯度下降法（小批量 SGD）是介于全批量迭代与 SGD 之间的折衷方案。小批量通常包含 10-1000 个随机选择的样本。小批量 SGD 可以减少 SGD 中的杂乱样本数量，但仍然比全批量更高效

批量梯度下降优化（BGD，Batch gradient descent）实现

修改超参数

设置训练超参数

```
training_epochs = 100     # 迭代次数（训练轮数）
learning_rate = 0.05     # 学习率
```

训练周期和学习率需要做一些调整。训练周期暂设为100，意味着所有样本要参与100次训练。学习率设置为0.05，比SGD版本的要大。

修改模型训练过程

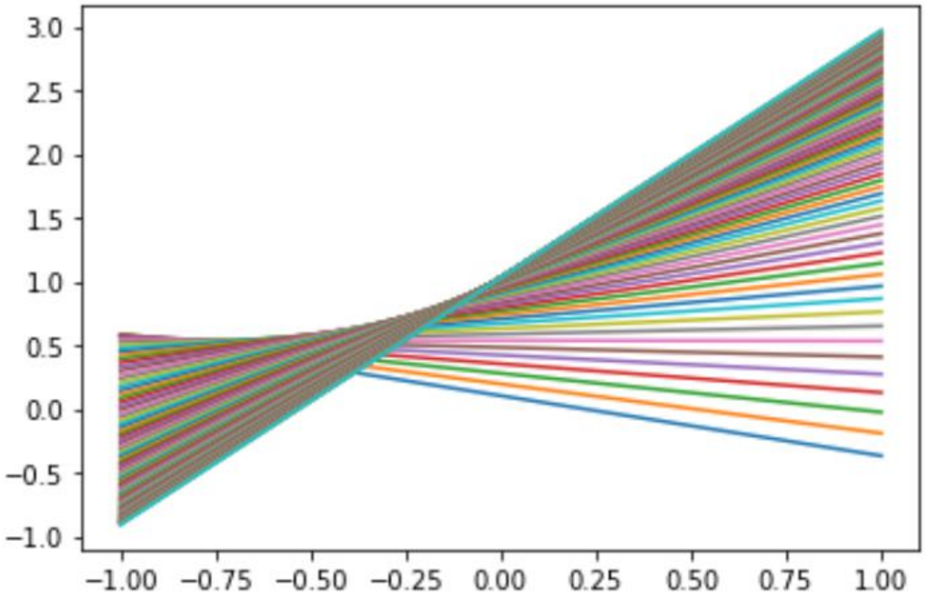
浙江大学城市学院 ZHEJIANG UNIVERSITY CITY COLLEGE

执行训练 （BGD）

```
loss_list = []    # 用于保存loss值的列表

for epoch in range(training_epochs):
    loss_ = loss(x_data, y_data, w, b)     # 计算损失，所有样本作为一个整体参与计算
    loss_list.append(loss_)     # 保存本次损失计算结果

    delta_w, delta_b = grad(x_data, y_data, w, b) # 计算该当前[w,b]点的梯度
    change_w = delta_w * learning_rate # 计算变量w需要调整的量
    change_b = delta_b * learning_rate # 计算变量b需要调整的量
    w.assign_sub(change_w) # 变量w值变更为减去chage_w后的值
    b.assign_sub(change_b) # 变量b值变更为减去chage_b后的值


    print("Training Epoch:", '%02d' % (epoch+1), "loss=%.6f" % (loss_))
    plt.plot (x_data, w.numpy() * x_data + b.numpy()  )     # 完成一轮训练后，画出回归的线条
```

训练结果可视化

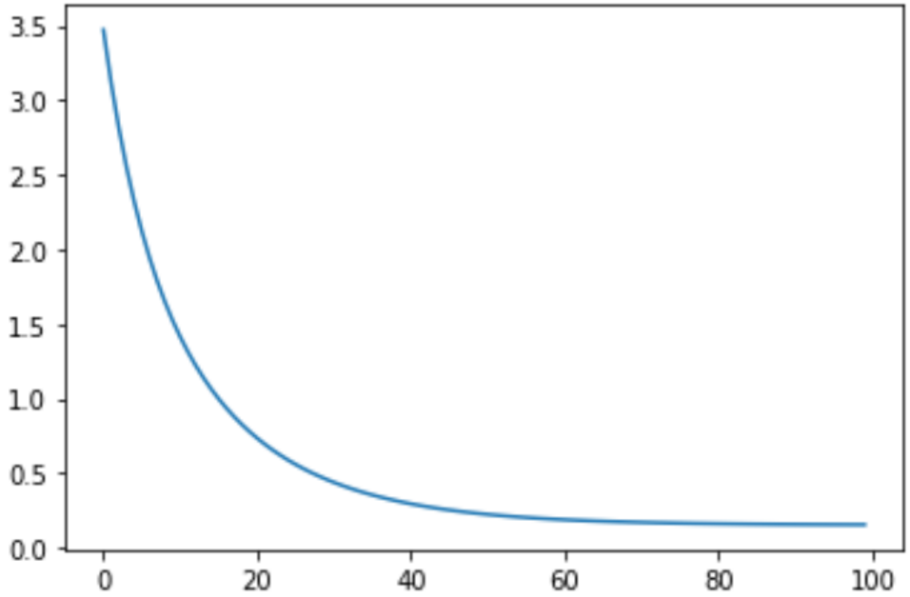
```
Train Epoch: 98 loss=0.156618
Train Epoch: 99 loss=0.156434
Train Epoch: 100 loss=0.156263
```



损失值得可视化

查看损失变化情况

```
plt.plot(loss_list)
```



梯度下降算法总结

批量梯度下降每次迭代都考虑了全部的样本，做的是全局优化，但花费的计算资源较大，如果训练数据非常大，还无法实现全部样本同步参与。

随机梯度下降每次迭代只取一条样本数据，由于单个样本的训练可能会带来很多噪声，使得SGD并不是每次迭代都向着整体最优化方向，因此在刚开始训练时可能收敛得很快，但是训练一段时间后就会变得很慢。

在SGD和BGD中间,还有一个集合了两种梯度下降法的优点的方法:**小批量梯度下降**（MBGD，Mini-batch gradient descent），每次迭代从训练样本中随机抽取一小批进行训练，这个一小批的数量取值也是一个超参数。

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）（梯度下降）优化过程

（5）训练结果的可视化

（6）利用学习到的模型进行预测