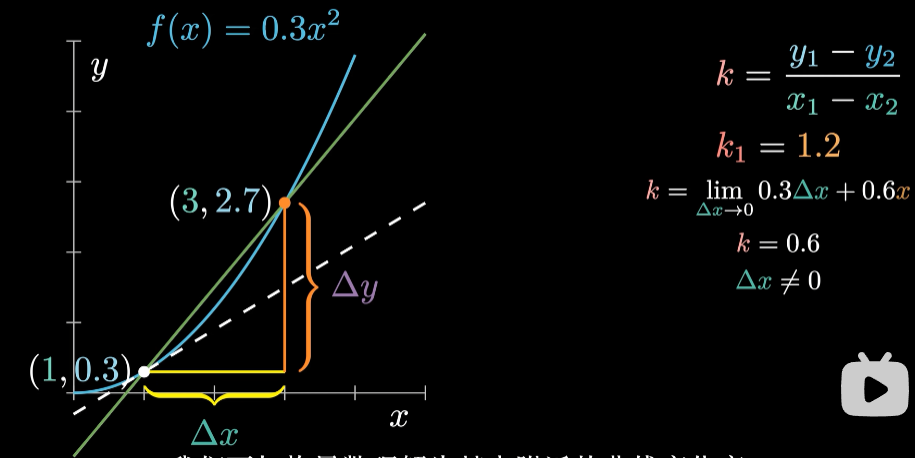
## 导数

导数：下图求的是(1, 0.3 )变化率,导数跟变化梯度有关



导数的意义

衡量变量之间的相互影响：偏导数用于衡量一个函数在某个特定点上，其中一个变量的变化对于另一个变量的影响程度。它可以揭示函数在不同变量方向上的敏感性和相关性。通过计算偏导数，可以了解每个变量对函数值的贡献程度，从而帮助我们理解变量之间的相互关系。

偏导数可以帮助我们衡量函数中不同变量之间的相互影响程度。具体来说，偏导数测量了函数在某个特定点上，沿着每个变量方向的变化速率。

假设我们有一个多元函数f(x, y)，其中x和y是函数的输入变量。我们可以通过计算偏导数来了解x和y之间的相互影响。

当计算关于变量x的偏导数 ∂f/∂x 时，我们将y视为常数，仅考虑x的变化对函数值的影响。 如果 ∂f/∂x 为正，则意味着当x增加时，f的值也会增加。反之，如果 ∂f/∂x 为负，则当x增加时，f的值会减小。通过这种方式，我们可以了解函数在x轴方向上的变化趋势。

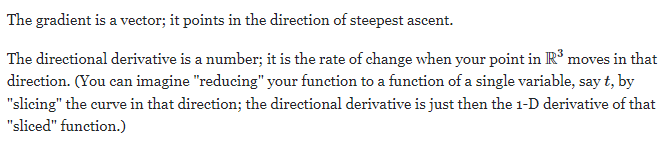
同样地，当计算关于变量y的偏导数 ∂f/∂y 时，我们将x视为常数，仅考虑y的变化对函数值的影响。类似地，如果 ∂f/∂y 为正，则意味着当y增加时，f的值也会增加；如果 ∂f/∂y 为负，则当y增加时，f的值会减小。

通过比较不同变量的偏导数值，我们可以得出结论关于变量之间的相互影响。例如，如果 ∂f/∂x 和 ∂f/∂y 同为正值，则x和y的增大都会导致f的增加，这意味着x和y之间存在正相关关系。相反，如果 ∂f/∂x 和 ∂f/∂y 在某个点上分别为正和负，那么x和y之间存在负相关关系，即当x增加时，y减小，反之亦然。

通过计算不同变量的偏导数，我们可以得到有关变量之间相互影响的信息，这对于理解函数的行为模式、分析数据之间的关系以及优化问题等非常重要。

最优化问题中的梯度信息：在最优化问题中，需要找到函数的最大值或最小值。梯度是一个向量，由各个偏导数构成，反映了函数在某个点上的最速上升或最速下降方向。通过计算偏导数，可以得到梯度信息，用于指导优化算法的迭代过程，并找到函数的极值点。

What is the difference between the gradient and the directional derivative?



## 什么是线性回归？

比如说。 身高体重预测就是一种线性回归。 身高体重预测是什么呢?就是说先给你非常非常多的这种人的身高体重信息。然后你从中可以找到身高与体重之间的一种定量关系。 比如说给你一个身高， 你乘上一个数字，再加上一个数字， 你就可以得到一个体重。 这就是一种定量关系。 一旦你找到这个定量关系之后，我再给你一个新的身高， 你就可以预测出他的体重了。这就是一个线性回归的例子，还有一种线性回归是房价预测。

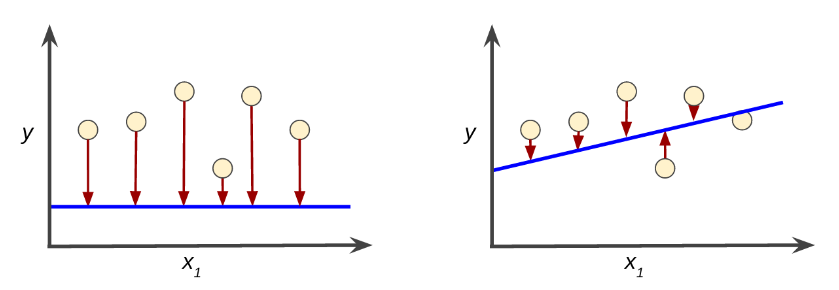
刚才那个身高体重预测， 它是两种变量之间的关系。 那么房价预测则有可能是多种变量之间的关系，比如说房子的面积以及房子里面的房屋数量， 都会影响房价。 那么就是三种变量之间的定量关系了，如果我给你海量的这种房价数据。 比如说多少平的房子，多少钱，多少个房间的房子，多少钱等等，然后你就可以得出一种定量关系。 最后我再给你一个新的房屋面积以及房间数量，你就可以预测出它的房价了

身高与体重的预测，根据大量的身高和体重数据，可以得到一个身高和体重的规律，也就是方程，这个就是回归，回归数学的意思

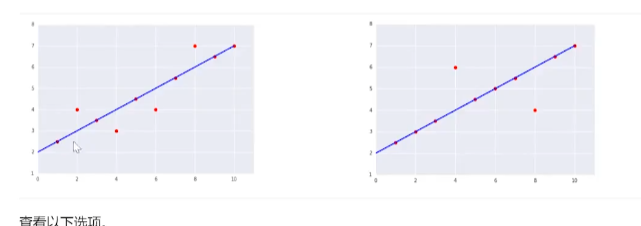
## 什么是均方误差？

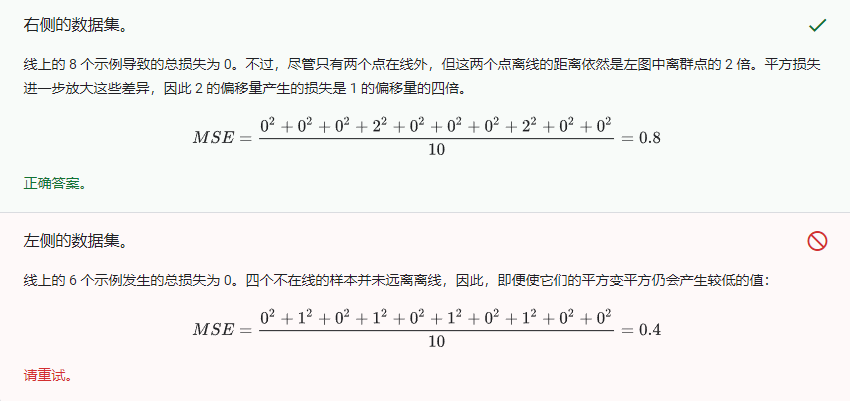
下图代表一些训练数据xy的对应关系，把它可视化成点。 然后它说红色箭头表示损失，蓝线表示预测， 我们想一下，我们训练神经网络，解决线性回归， 最终的解决的结果是什么呢? 其实无非就是找到x跟的一个对应关系，那么这个对应关系呢，其实可视化出来就是一条线。

为什么呢? 只要这条线画出来了， 你随便给我一个x误， 我一比我就能找到y了， 所以说，蓝线表示预测，也就是说我们最终回归出来的其实就是一条线而已，那么红色箭头表示损失。这又代表什么意思呢? 我们这条蓝线其实是为了能体现大部分这种训练数据的xy对应关系。



均方误差如何计算

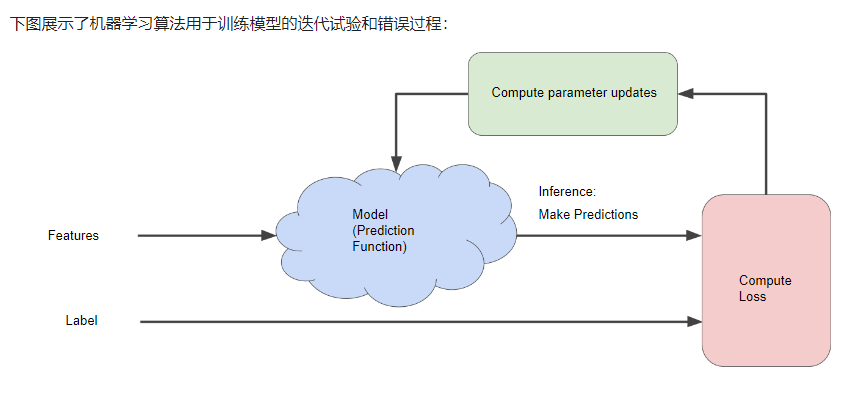




什么是梯度？

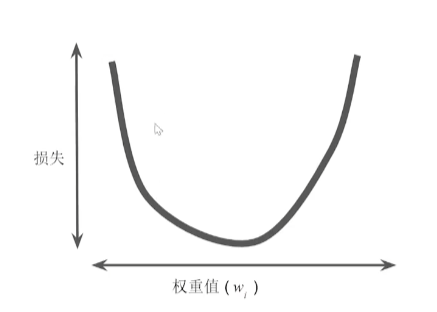
就是我刚才说的那个不断的试错，然后不断的去优化的整个过程，我们来先走一遍。 首先，我们先把特征位给这个模型，然后他瞎蒙了一个值，然后他瞎蒙的这个值跟真实的值，我们一我们算一下，哎，你这起跑到这个计算损失的这个函数里面，个跟真实的标。 标签离得也太远了吧，然后我们告诉它损失接下来它通过优化器，然后来找到这个优化的方向和大小。

然后它就再优化一下这个模型，优化完之后呢，它再重复一下刚才这个过程，它再用一个特征进来。当然，这个特征数量是根据你的样本来定的，有些样本可能多个特征在我们这个线性回归里只有一个特征，就是那个x，但是我们要把所有样本，所有训练数据的x都输入进来。然后输入到这个模型里面去计算这个损失，重复这个过程。 好，这就是一个迭代的方法，一个不断循环的过程



然后我们来看一下这个梯度下降法。 在介绍这个随机梯度下降法之前，我先给大家讲一下梯度下降法什么是梯度下降法呢?我们先来看一下它这个图，它这个图是一个损失。 与权重值关系的一个图，也就是说我给大家解释一下啊，就比如说我们知道训练神经网络，说白了就是调权重的过程。我们可以把权重想象成一个旋钮，在那调调调到你合适的就跟再加你自己的音响一样，你调音响上的各种旋钮调到一个埃不错的音质。 就跟那个道理是差不多的，它这个也是调就调权重，它就在这调

它把权重调小，发现损失变大了，它把权重又调大。 结果损失又变大了，只有权重在这个最低位置的时候，损失是最小的，然后它就算是调好了，这就是训练神经网络的过程。

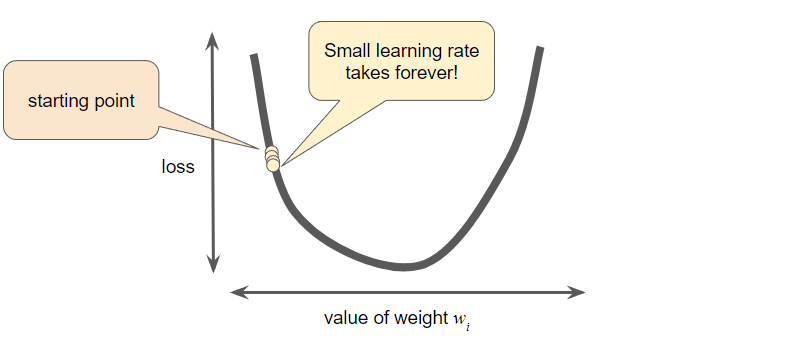


那么，这跟梯度下降法有什么关系呢? 我们先试想一下啊，假设说神经网络，它先瞎蒙了一个。权重值我们知道，它初始化的时候，

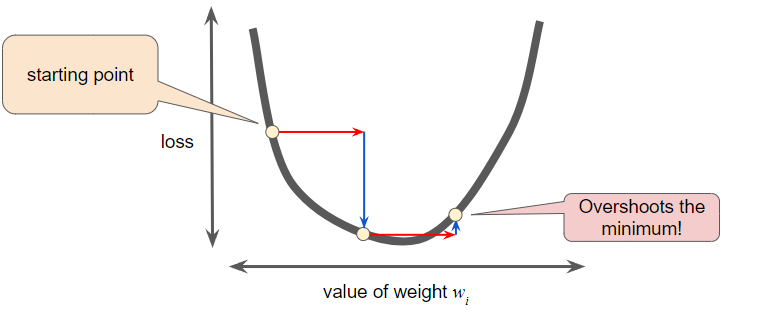
通过函数的导数可以得出斜率，知道斜率就知道往哪个方向移动，所以推导方向就是一个计算斜率的过程，但是呢。 我们在这儿这个名词啊，它叫做梯度。 因为梯度它不光是一个这个，因为斜率，它只有方向，对吧? 我们不光想知道它往哪走，我们还想知道它要走多大， 所以说梯度其实就是方向加大小。就组成了梯度，就是我刚才说的那个改进的方向和幅度就是这个意。

梯度的原理没有说

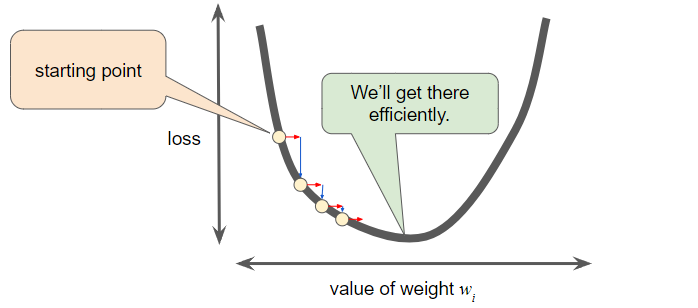
学习速率太小



学习速率太大



学习速率适中



我们如果能找到一个恰恰好的学习速率。 那么我们就可以高效的达到最低点，速度也快，也不会越过这个终点。 这就是一个非常好的学习速率，有没有什么公式可以帮我算出来一个最好的学习速率? 大概率情况下，这个学习速率不是算出来的，都是试出来的。

梯度和方向导数都是描述函数在某一点上的变化率的概念，但是两者之间存在一些区别。

定义：梯度是一个向量，表示函数在某一点上的变化率最快的方向和变化率的大小；方向导数是一个标量，表示函数在某一点上沿某一方向的变化率。

符号表示：梯度通常用符号∇f表示，其中∇是一个算子（向量），f是函数（标量）；方向导数通常用符号∂f/∂x表示，其中∂是偏导数的符号，f是函数，x是自变量。

方向：梯度是函数最快变化的方向；方向导数是沿某一特定方向的变化率。

运算对象：梯度是函数的全局性质，一般与整个函数有关；方向导数是函数的局部性质，仅与某一点附近的函数有关。

几何意义：梯度可以表示为函数在某一点上的等值线的法向量；方向导数可以表示为函数在某一点上的曲面沿给定方向的切线斜率。

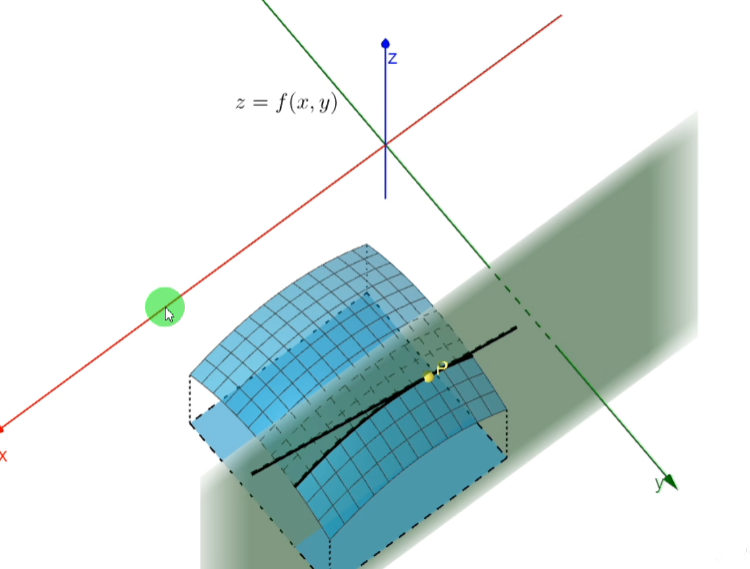
总的来说，梯度描述的是一个函数最快变化的方向和速率，而方向导数描述的是函数沿给定方向的变化率。梯度是一个向量，方向导数是一个标量。

## 偏导数、方向导数，法向量，梯度

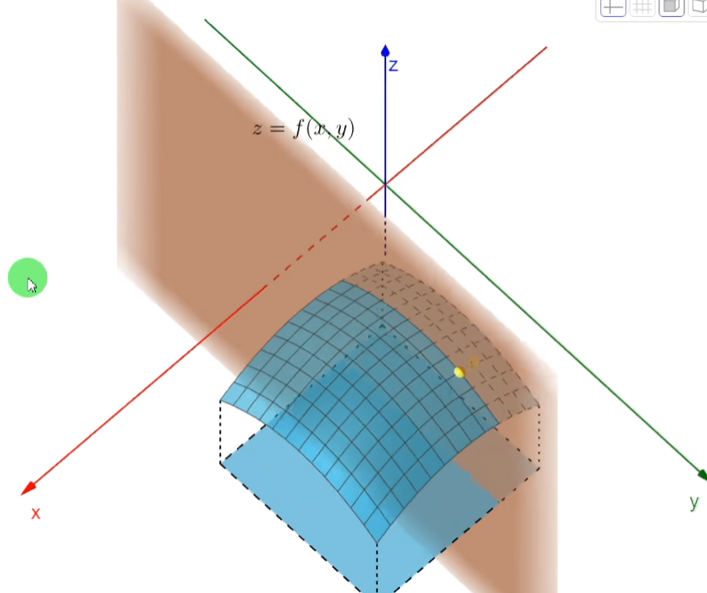
偏导数的视频

<https://www.bilibili.com/video/BV1cK411j7Ee/?spm_id_from=333.337.search-card.all.click&vd_source=73f0f43dc639135d4ea9acffa3ad6ae0>

黑色线的斜率就是f(x, y)的偏导数，下图是关于x的偏导函数



下图是关于y的偏导函数



## 归一化

为什么需要进行归一化

很多数据很大，我们需要压缩数据，例如x和y的数量级不要差别很大，都是个位数。

如果是房价跟面积的关系，就需要进行归一化训练了，因为面积是一般百位数，但是房价可能是7位数，相差较大，我们需要将房价缩小

归一化就是压缩数量级的意思。

如何进行归一化

（150-150）/20 = 0

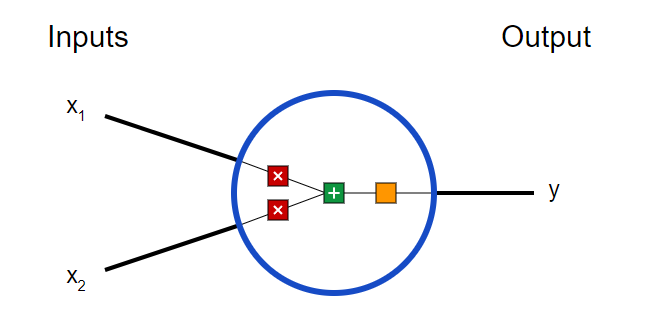
（160-150）/20 = 0.5

（170-150）/20 = 1

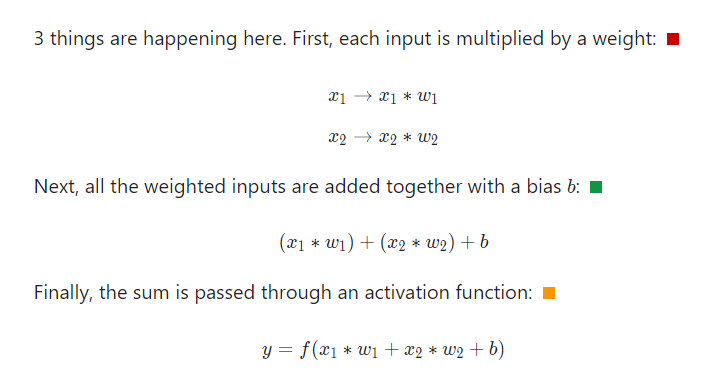
数据转化成为【0, 0.5 ,1】

## 什么是神经元？

神经网络的一个神经元



它的各个颜色的点计算方法为



对应的json格式一概是这个样子

{

  "weights": [

      0.6775066827937491,

      0.25941869760260156

  ],

  "bias": 0.2

}

对应的js代码是这个样子，bias相当于这个神经元的个性

class Neuron {

  constructor(numInputs) {

    this.weights = [];

    this.bias = 0.2; // 随机初始化偏置值

    // 随机初始化输入权重

    for (let i = 0; i < numInputs; i++) {

      this.weights.push(Math.random());

    }

  }

  // 激活函数，这里使用sigmoid函数

  activate(inputs) {

    // 省略激活函数代码

    if (inputs.length !== this.weights.length) {

      throw new Error('输入的数量与权重的数量不匹配');

    }

    let sum = 0;

    for (let i = 0; i < inputs.length; i++) {

      sum += inputs[i] \* this.weights[i];

    }

    sum += this.bias;

    return this.sigmoid(sum);

  }

  sigmoid(x) {

    return 1 / (1 + Math.exp(-x));

  }

  sigmoidDerivative(x) {

    return x \* (1 - x);

  }

}