人工智能之机器学习

梯度下降

主讲人：李老师

梯度下降法

- 梯度下降法(Gradient Descent， GD)常用于求解**无约束**情况下**凸函数**(**Convex Function**)的**极小值**，是一种**迭代类型**的算法，因为凸函数只有一个极值点，故求解出来的极小值点就是函数的**最小值点**。

$$J(\theta)=\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$$

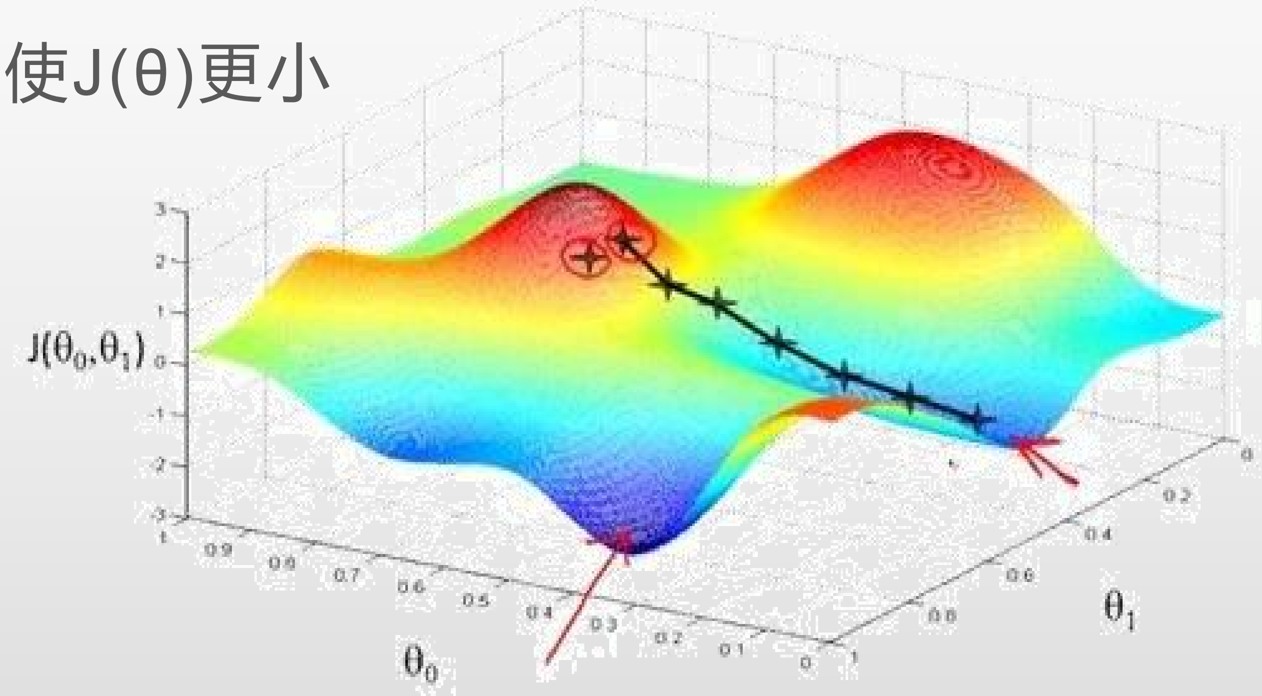
$$\theta^* = \arg\min_\theta J(\theta)$$

梯度下降算法

- 目标函数θ求解 $J(\theta)=\frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$
- 初始化θ(随机初始化，可以初始为0)
- 沿着负梯度方向迭代，更新后的θ使J(θ)更小

$$\theta=\theta-\alpha\bullet\frac{\partial J(\theta)}{\partial\theta}$$

  - α：学习率、步长



梯度方向

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

仅考虑单个样本的单个θ参数的梯度值

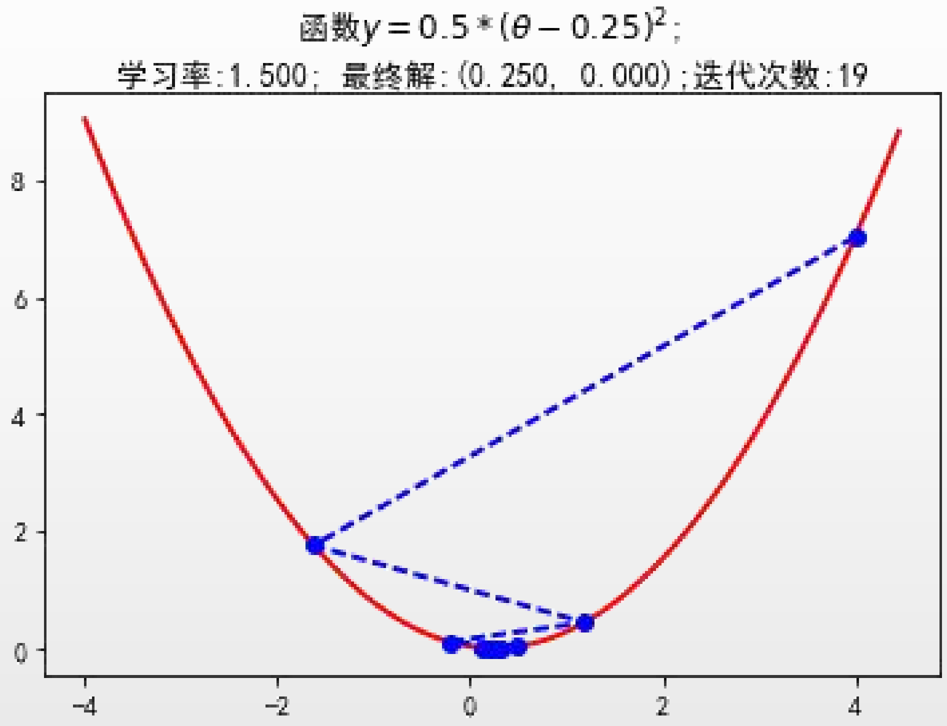
$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{\partial}{\partial\theta_j}\frac{1}{2}\left(h_\theta(x) - y\right)^2$$

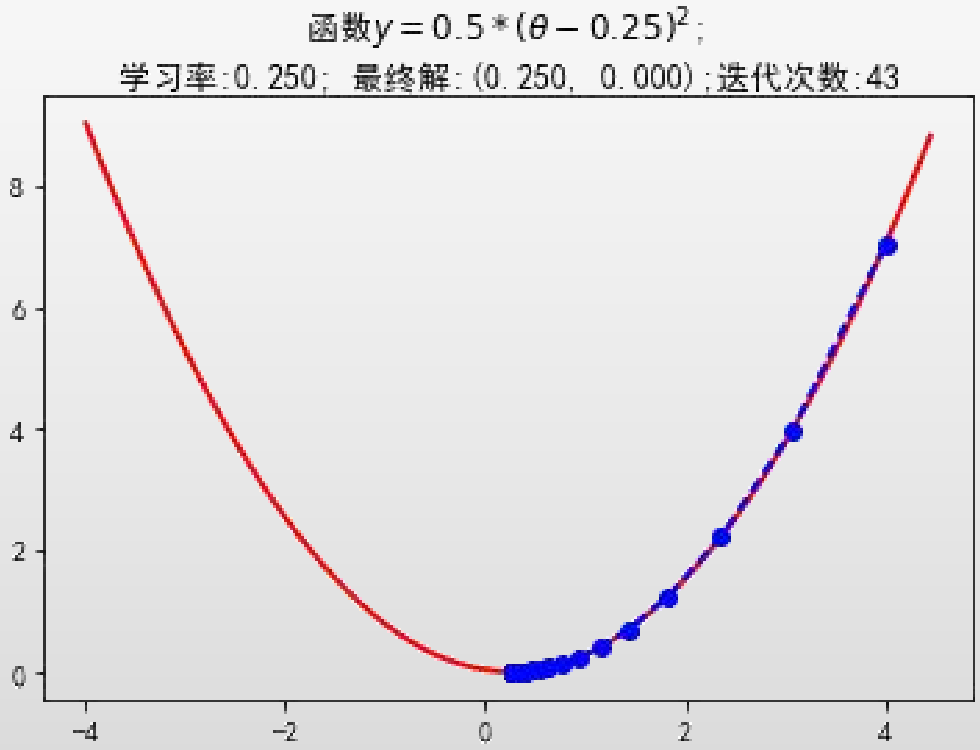
$$= 2\cdot\frac{1}{2}\left(h_\theta(x) - y\right)\cdot\frac{\partial}{\partial\theta_j}\left(h_\theta(x) - y\right)$$

$$= \left(h_\theta(x) - y\right)\frac{\partial}{\partial\theta_j}\left(\sum_{i=0}^{n}\theta_i x_i - y\right)$$

$$= \left(h_\theta(x) - y\right)x_j$$

梯度下降法案例代码


函数$y=0.5*(\theta-0.25)^2$;
学习率:1.500; 最终解:(0.250, 0.000);迭代次数:19
8
6
4
2
0
-4
-2
0
2
4


函数$y=0.5*(\theta-0.25)^2$;
学习率:0.250; 最终解:(0.250, 0.000);迭代次数:43
8
6
4
2
0
-4
-2
0
2
4

梯度下降案例

$$z = f(x,y) = x^2 + y^2$$

```
## 原函数
def f(x, y):
    return x ** 2 + y ** 2
## 偏函数
def h(t):
    return 2 * t

X = []
Y = []
Z = []

x = 2
y = 2
f_change = x ** 2 + y ** 2
f_current = f(x, y)
step = 0.1
X.append(x)
Y.append(y)
Z.append(f_current)
while f_change > 1e-10:
    x = x - step * h(x)
    y = y - step * h(y)
    f_change = f_current - f(x, y)
    f_current = f(x, y)
    X.append(x)
    Y.append(y)
    Z.append(f_current)
print u"最终结果为:", (x, y)
```

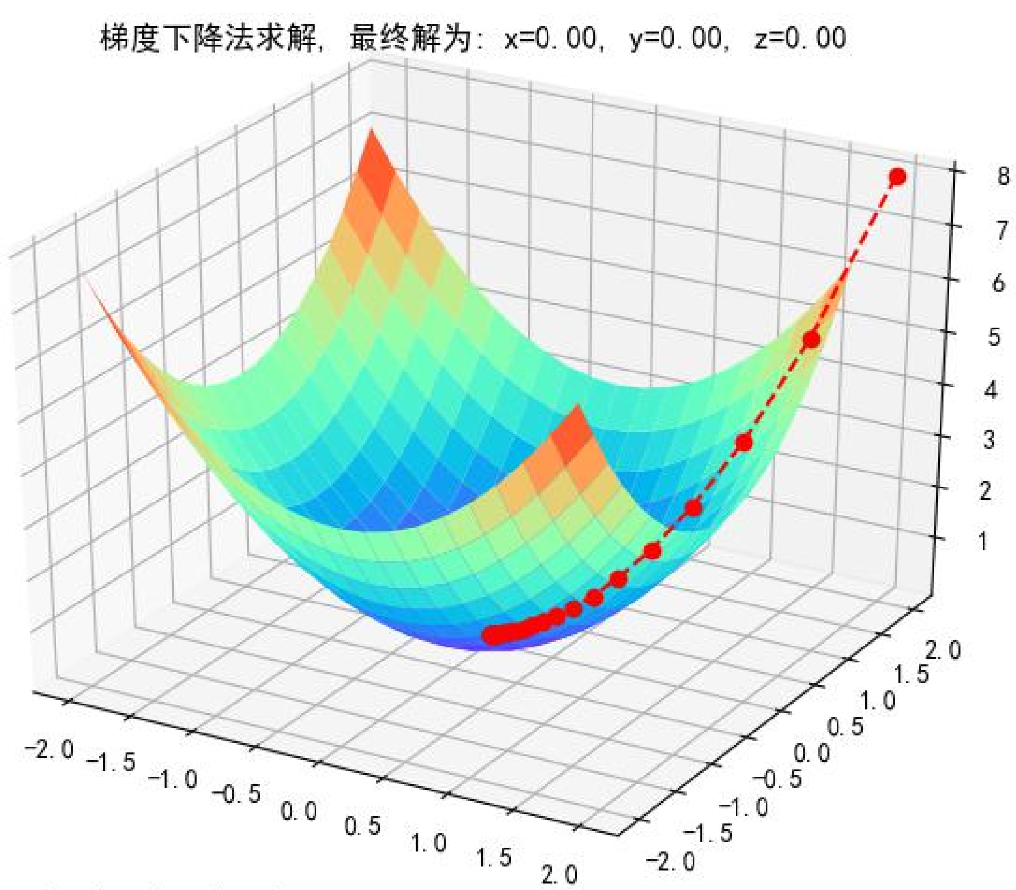
```
最终结果为：(9.353610478917782e-06, 9.
```

```
fig = plt.figure()
ax = Axes3D(fig)
X2 = np.arange(-2, 2, 0.2)
Y2 = np.arange(-2, 2, 0.2)
X2, Y2 = np.meshgrid(X2, Y2)
Z2 = X2 ** 2 + Y2 ** 2

ax.plot_surface(X2, Y2, Z2, rstride=1, cstride=1, cmap='ra
ax.plot(X, Y, Z, 'ro--')

ax.set_title(u'梯度下降法求解, 最终解为: x=%.2f, y=%.2f, z=%.2f' % (x, y, f_current))

plt.show()
```



随机梯度下降算法(SGD)

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

使用单个样本的梯度值作为当前模型参数θ的更新

$$\frac{\partial}{\partial \theta_j} J(\theta) = \left(h_\theta(x) - y\right) x_j$$

for i= 1 to m,{

$$\theta_j = \theta_j + \alpha\left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j^{(i)}$$

}

批量梯度下降算法(BGD)

使用所有样本的梯度值作为当前模型参数θ的更新

$$\frac{\partial}{\partial \theta_j} J(\theta) = \left(h_\theta(x) - y\right) x_j$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_{i=1}^{m} \frac{\partial}{\partial \theta_j} = \sum_{i=1}^{m} \left(x_j^{(i)} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)\right) = \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

$$\theta_j = \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta\left(x^{(i)}\right)\right) x_j^{(i)}$$

小批量梯度下降法(MBGD)

- 如果即需要保证算法的训练过程比较快，又需要保证最终参数训练的准确率，而这正是小批量梯度下降法（Mini-batch Gradient Descent，简称MBGD）的初衷。MBGD中不是每拿一个样本就更新一次梯度，而且拿b个样本(b一般为10)的平均梯度作为更新方向。

for i= 1 to m/10,{

$$\theta_j = \theta_j + \alpha \sum_{k=i}^{i+10} \left(y^{(k)} - h_\theta\left(x^{(k)}\right) \right) x_j^{(k)}$$

}

BGD和SGD算法比较

- SGD速度比BGD快(整个数据集从头到尾执行的迭代次数少)
- SGD在某些情况下(全局存在多个相对最优解/J(θ)不是一个二次)，SGD有可能跳出某些小的局部最优解，所以一般情况下不会比BGD坏；SGD在收敛的位置会存在J(θ)函数波动的情况。
- BGD一定能够得到一个局部最优解(在线性回归模型中一定是得到一个全局最优解)，SGD由于随机性的存在可能导致最终结果比BGD的差
- **注意：优先选择SGD**

梯度下降法

- 由于梯度下降法中负梯度方向作为变量的变化方向，所以有可能导致最终求解的值是局部最优解，所以在使用梯度下降的时候，一般需要进行一些调优策略：
  - **学习率的选择**：学习率过大，表示每次迭代更新的时候变化比较大，有可能会跳过最优解；学习率过小，表示每次迭代更新的时候变化比较小，就会导致迭代速度过慢，很长时间都不能结束；
  - **算法初始参数值的选择**：初始值不同，最终获得的最小值也有可能不同，因为梯度下降法求解的是局部最优解，所以一般情况下，选择多次不同初始值运行算法，并最终返回损失函数最小情况下的结果值；
  - **标准化**：由于样本不同特征的取值范围不同，可能会导致在各个不同参数上迭代速度不同，为了减少特征取值的影响，可以将特征进行标准化操作。

梯度下降法

- BGD、SGD、MBGD的区别：
    - 当样本量为m的时候，每次迭代BGD算法中对于参数值更新一次，SGD算法中对于参数值更新m次，MBGD算法中对于参数值更新m/n次，相对来讲SGD算法的更新速度最快；
    - SGD算法中对于每个样本都需要更新参数值，当样本值不太正常的时候，就有可能会导致本次的参数更新会产生相反的影响，也就是说SGD算法的结果并不是完全收敛的，而是在收敛结果处波动的；
    - SGD算法是每个样本都更新一次参数值，所以SGD算法特别适合样本数据量大的情况以及在线机器学习(Online ML)。