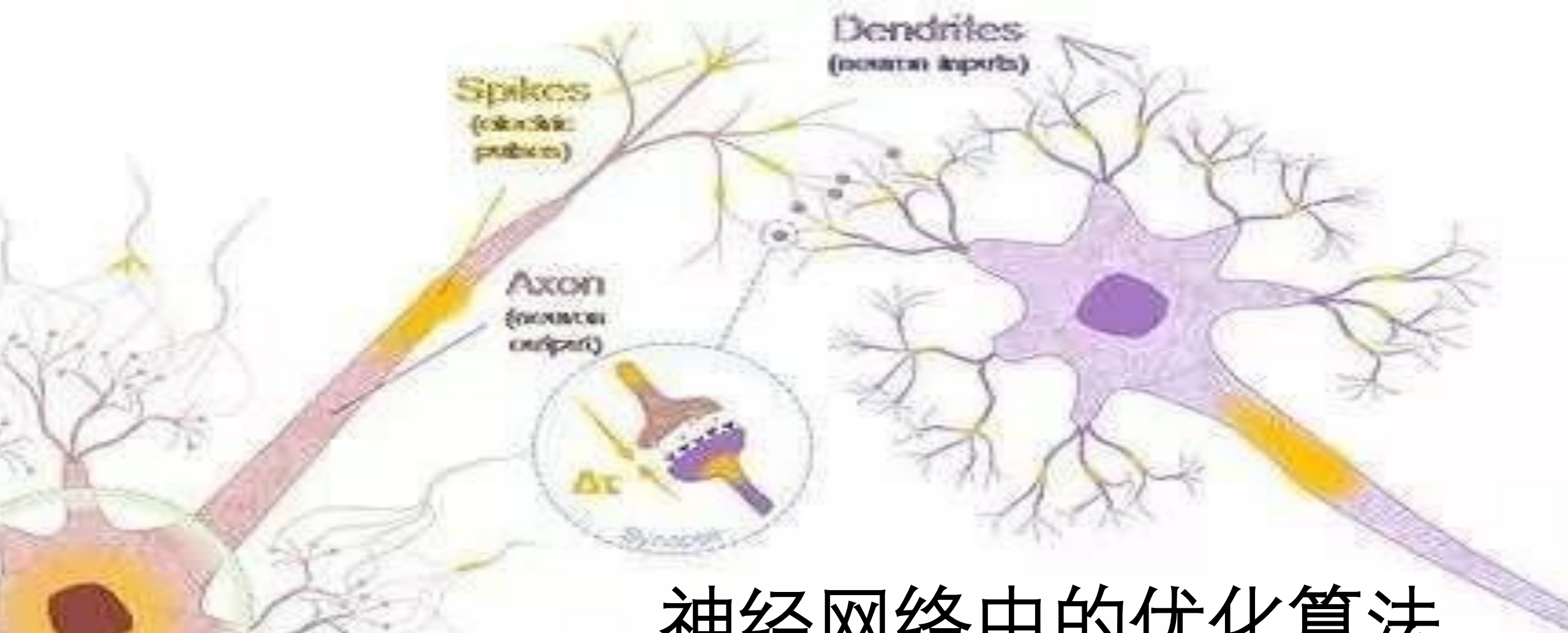


神经网络中的优化算法

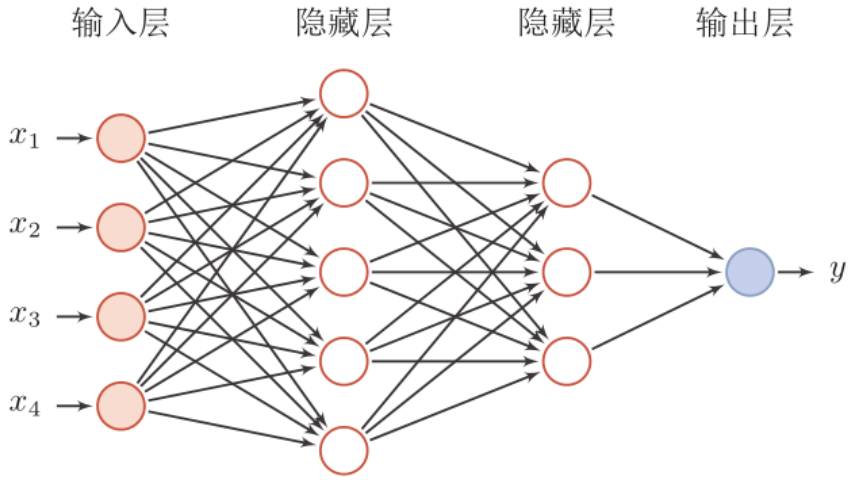
《人工神经网络》第3讲

胡贤良

2023年3月

主要内容

1. 神经网络中(一阶)优化算法
2. 梯度优化的衍生算法
3. 二阶近似方法



$$\boldsymbol{z}^{(l)} = \boldsymbol{W}^{(l)}\boldsymbol{a}^{(l-1)} + \boldsymbol{b}^{(l)},$$
$$\boldsymbol{a}^{(l)} = f_l(\boldsymbol{z}^{(l)}).$$

$$\boldsymbol{x} = \boldsymbol{a}^{(0)} \to \boldsymbol{z}^{(1)} \to \boldsymbol{a}^{(1)} \to \boldsymbol{z}^{(2)} \to \cdots \to \boldsymbol{a}^{(L-1)} \to \boldsymbol{z}^{(L)} \to \boldsymbol{a}^{(L)} = \phi(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b}))$$

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$
$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

1 神经网络中一阶优化算法

神经网络训练方法

学习和纯优化的不同

机器学习 = 优化？ NO!

- 代价函数通常可写为训练集上的平均：

$$J(\theta) = E_{(x,y)\sim p_{data}} L(f(x;\theta), y)$$

- 将机器学习问题转化成优化问题的方法是**经验风险最小化**，即用训练集上的经验分布 $\hat{p}(x,y)$ 替代真实分布：

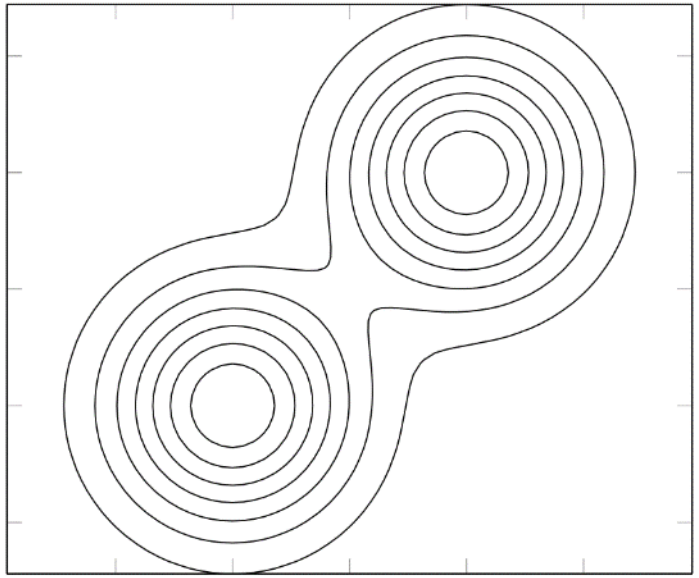
$$E_{(x,y)\sim \hat{p}_{data}}[L(f(x;\theta), y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta), y^{(i)})$$

- 其中，主要的挑战在于：
  1. 如何得到参数θ
  2. 在规模极大时如何效率地解决问题
- 基于梯度下降的优化算法：批量算法和小批量算法

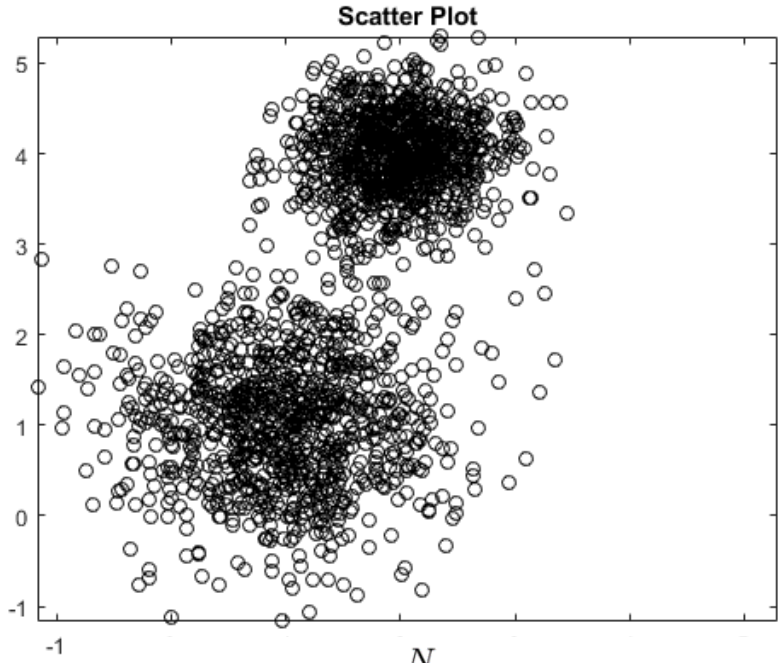
泛化错误 $\mathcal{G}_{\mathcal{D}}(f) = \mathcal{R}(f) - \mathcal{R}_{\mathcal{D}}^{emp}(f)$

- 将机器学习问题转化回一个优化问题的方法是最小化训练集上的期望损失。
- 这意味着用训练集上的经验分布替代真实分布 $E_{(x,y)\sim\hat{p}_{data}}[L(f(x;\theta),y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(x^{(i)};\theta),y^{(i)})$

真实分布 p_r



$\neq$



期望风险 $\mathcal{R}(f) = \mathbb{E}_{(\mathbf{x},y)\sim p(\mathbf{x},y)}[\mathcal{L}(f(\mathbf{x}),y)],$

$$\mathcal{R}_{\mathcal{D}}^{emp}(\theta) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(y^{(n)}, f(x^{(n)},\theta))$$

- 基于最小化这种平均训练误差的训练过程被称为**经验风险最小化（empirical risk minimization）**。

如何减少泛化错误?

优化

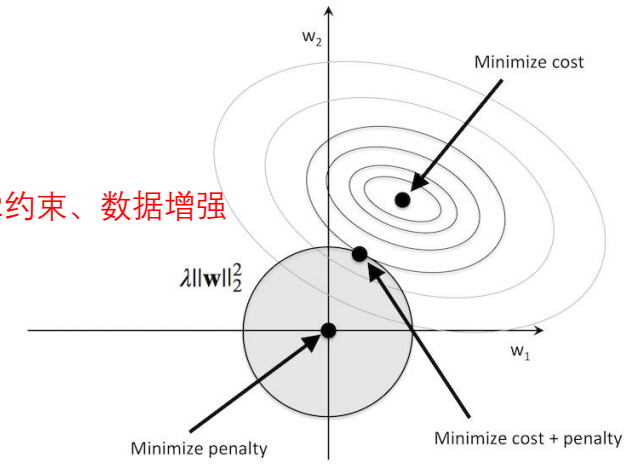
经验风险最小改善

正则化

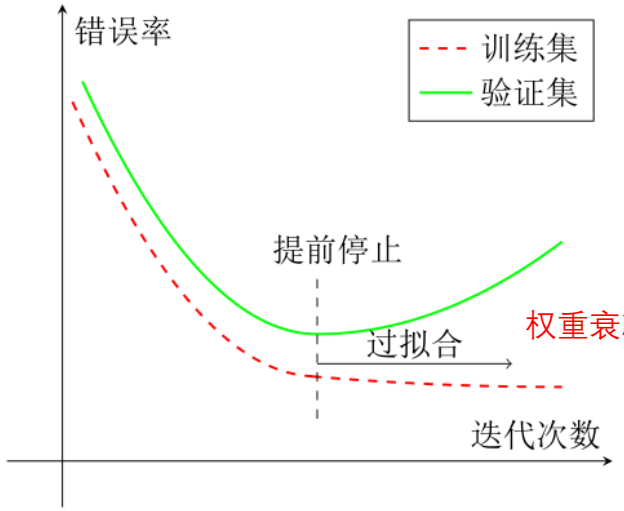
降低模型复杂度

正则化（regularization）

所有损害优化的方法都是正则化!



L1/L2约束、数据增强



权重衰减、随机梯度下降、提前停止

增加优化约束

干扰优化过程

1. 梯度下降 - 函数逼近解释

通过多项式对函数进行逼近

f(x)

$$= \frac{f(x_0)}{0!} + \frac{f'(x_0)}{1!}(x-x_0)^1 + \frac{f''(x_0)}{2!}(x-x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x-x_0)^n + R_n(x)$$

其中$R_n(x)$是余项

$$f(x+\Delta x) \simeq f(x) + \Delta x \nabla f(x)$$

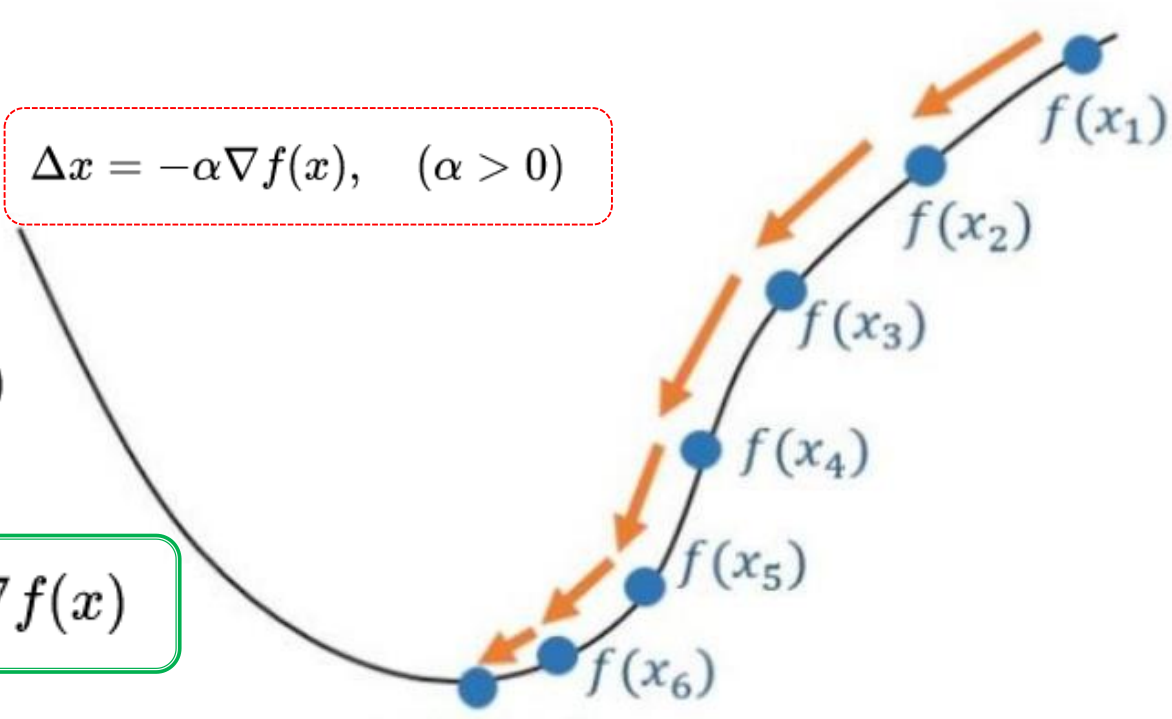
$$\Delta x = -\alpha \nabla f(x), \quad (\alpha > 0)$$

故

$$\Delta x = -\alpha \nabla f(x), \quad (\alpha > 0)$$

得梯度下降法：

$$x_{i+1} = x_i - \alpha \nabla f(x)$$



示例1：一元函数极小值

设一元函数为

$$J(\theta) = \theta^2$$

函数的微分为

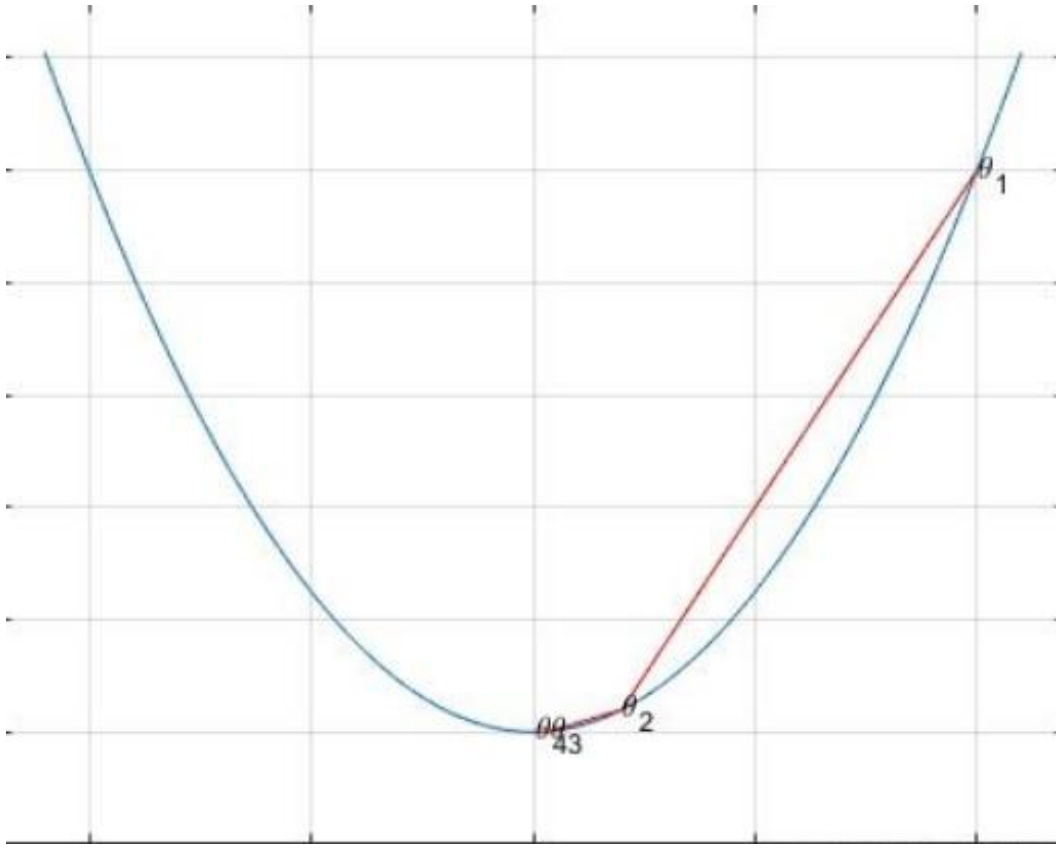
$$J^{'}(\theta) = 2\theta$$

设起点为 $J(\theta) = \theta^2$ ，步长 $alpha = 0.4$,根据梯度下降的公式

$$\theta_1 = \theta_0 - \alpha \nabla J(\theta)$$

经过4次迭代：

$$\theta_0 = 1$$
$$\theta_1 = \theta_0 - 0.4 * 2 * 1 = 0.2$$
$$\theta_2 = \theta_1 - 0.4 * 2 * 0.2 = 0.04$$
$$\theta_3 = \theta_2 - 0.4 * 2 * 0.04 = 0.008$$



示例2：二元函数极小值

设二元函数为

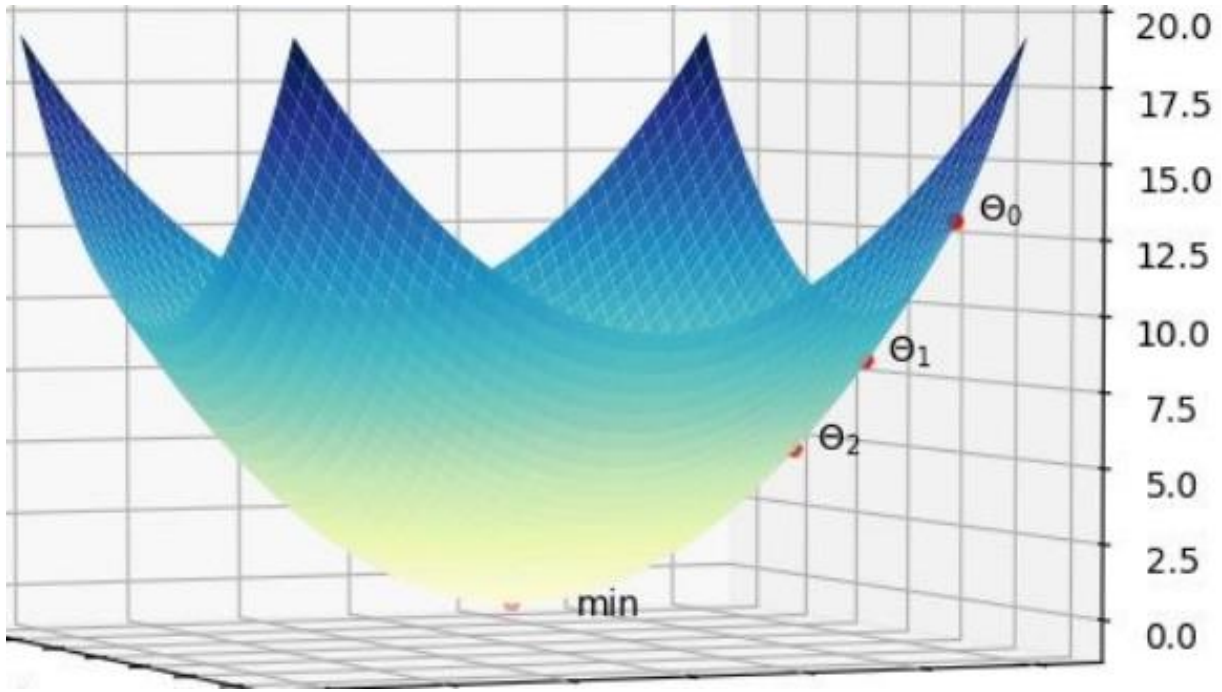
$$J(\Theta) = \theta_1^2 + \theta_2^2$$

函数的梯度为

$$\nabla J(\Theta) = (2\theta_1, 2\theta_2)$$

设起点为(2,3)，步长 $\alpha = 0.1$,根据梯度下降的公式,经过多次迭代后，

$$\Theta_0 = (2, 3)$$
$$\Theta_1 = \Theta_0 - 0.1 * (2 * 2, 2 * 3) = (1.6, 2.4)$$
$$\Theta_2 = \Theta_1 - 0.1 * (2 * 1.6, 2 * 2.4) = (1.28, 1.92)$$
$$\vdots$$
$$\Theta_{99} = \Theta_{98} - 0.1 * \Theta_{98} = (6.36e - 10, 9.55e - 10)$$
$$\Theta_{100} = \Theta_{99} - 0.1 * \Theta_{99} = (5.09e - 10, 7.64e - 10)$$



梯度下降法(一份简短的*Python*实现，同lecture 1)

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta \mathrm{f}(\theta_n)$$

```
import numpy as np
import matplotlib.pyplot as plt

def cost_function(theta, X, y):
    diff = np.dot(X, theta) - y
    return (1./2*m) * np.dot(np.transpose(diff), diff)

def gradient_function(theta, X, y):
    diff = np.dot(X, theta) - y
    return (1./m) * np.dot(np.transpose(X), diff)

def gradient_descent(X, y, eta):
    theta = np.array([1, 1]).reshape(2, 1)
    gradient = gradient_function(theta, X, y)
    while not np.all(np.absolute(gradient) <= 1e-5):
        theta = theta - eta* gradient
        gradient = gradient_function(theta, X, y)

    return theta
```

```
m = 18    # sample length
X0 = np.ones((m, 1))
X1 = np.arange(1, m+1).reshape(m, 1)
X = np.hstack((X0, X1))
# matrix y
y = np.array([2,3,3,5,8,10,10,13,15,15,16,
19,19,20,22,22,25,28])
y = y.reshape(m,1)
eta = 0.01
[theta0, theta1] = gradient_descent(X, y, eta)
plt.figure()
plt.scatter(x,y)
plt.scatter(X1,y)
plt.plot(X1, theta0 + theta1*X1, color='r')
plt.title('基于梯度下降算法的线性回归拟合')
plt.grid(True)
plt.show()
```

2. 批梯度下降法(BGD)求极小

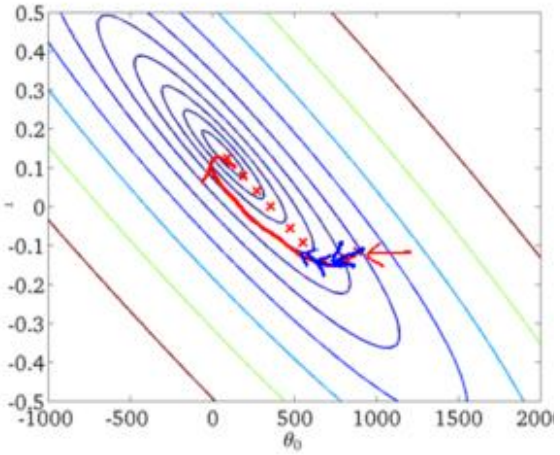
- 批梯度下降法(Batch Gradient Descent)针对的是整个数据集，通过对**所有的样本**的计算来求解梯度的方向。
- 每迭代一步，都要用到训练集所有的数据，如果样本数目很大，迭代速度就会很慢。优点是迭代次数较少
- 缺点：每更新一个参数的时候，要用到所有的样本，训练速度会随着样本数量的增加而变得非常缓慢。

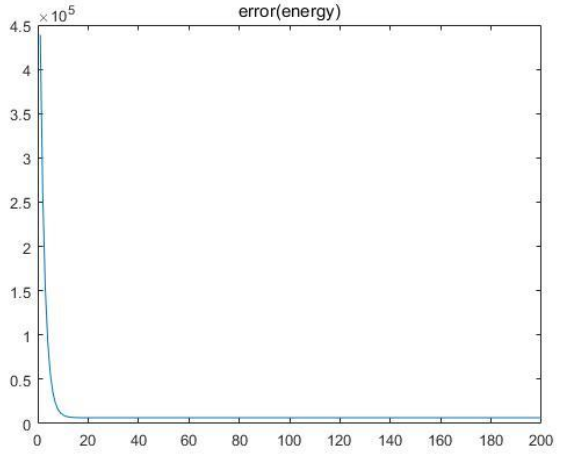
repeat{

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

(for every j=0, … , n)

}





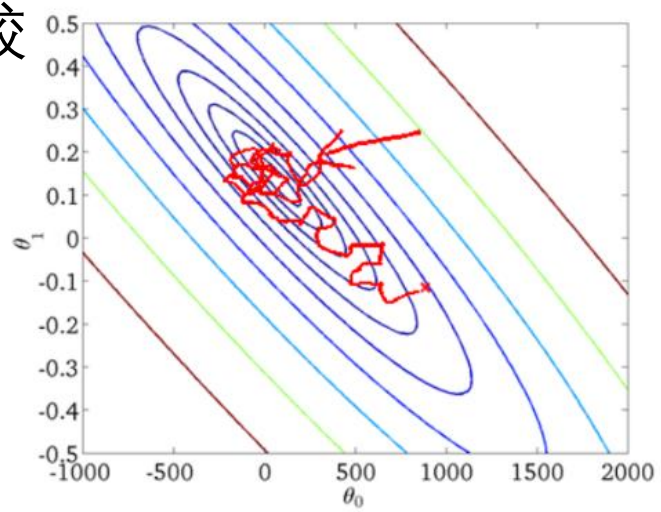
3. 随机梯度下降法(Stochastic GD)

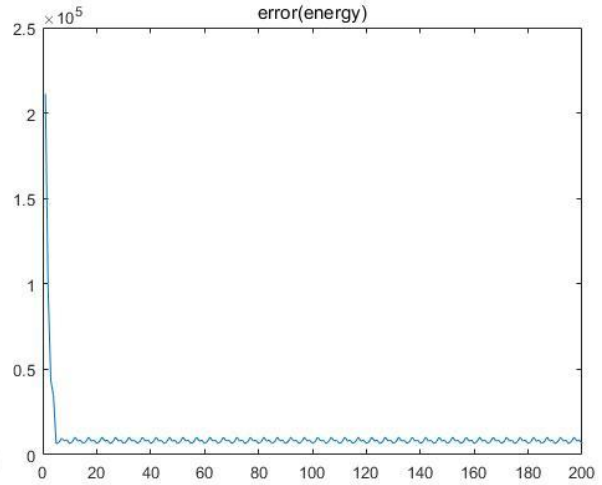
- SGD 利用**每个/逐个**样本的损失函数对θ求偏导得到对应的梯度来更新 θ 。迭代公式:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

- 缺点：更新数据频繁，方差较大，损失函数会产生波动。在解空间中体现为搜索较为盲目。但大体上仍向最优解方向移动

1. Randomly shuffle dataset ;
2. **repeat**{
   for i=1, … , m {
   $$\theta_j^{'} = \theta_j + (y^i - h_\theta(x^i))x_j^i$$
   (for j=0, … , n)
   }
}





算例展示

二维函数表达式：$f(x,y)=(1.5-xy)^2+\left(2.25-x+xy^2\right)^2+\left(2.65-x+xy^3\right)^3$

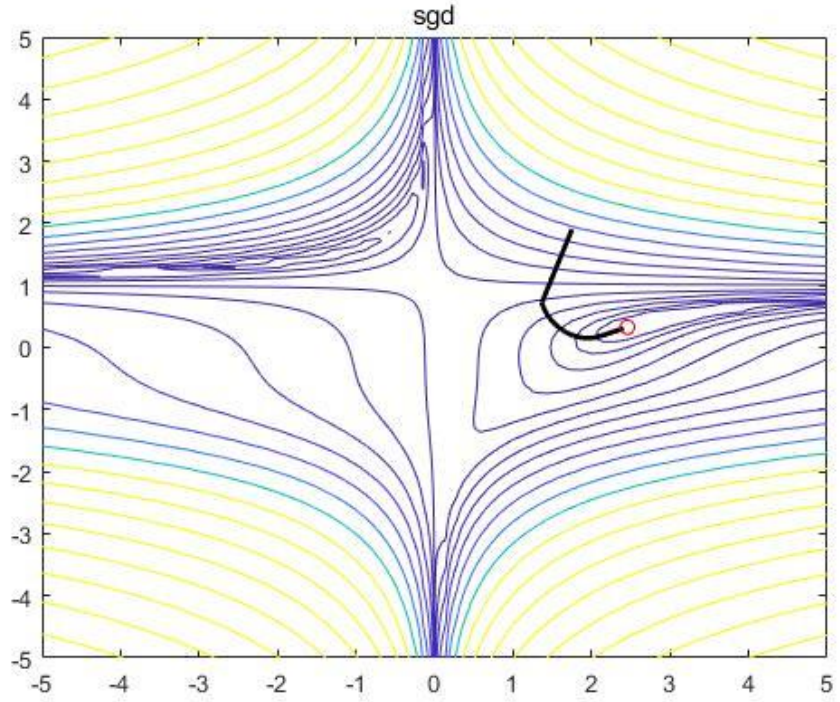
三维函数表达式：$f(x,y,z)=x^2+0.1y^2+2z^2$

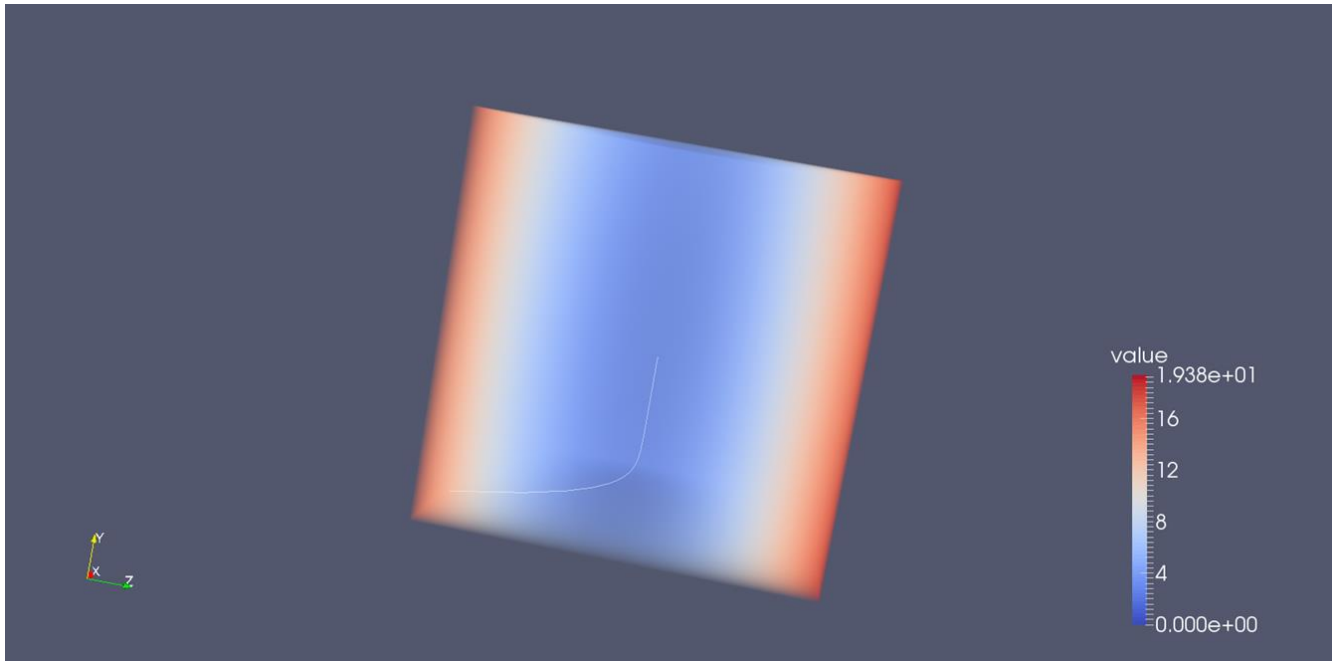
参数设置：

二维：$\eta=0.002$，起始点 [1.75 , 1.9]

三维：$\eta=0.1$，起始点 [-2,-2,-2]

- 代码参考：**gradient descent_exe2d下的sgd.m**

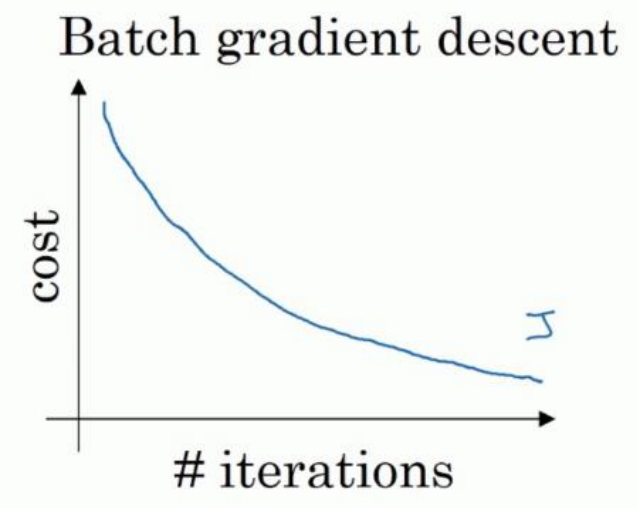


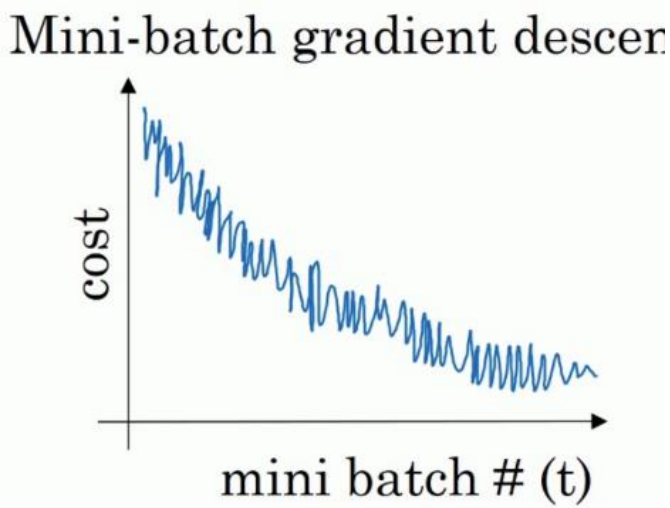


4. 改进 - 小批量(mini batch)梯度下降法

综合批梯度下降和随机梯度下降，提出小批量随机梯度下降。可以减小随机梯度下降的方差同时不至于使参数更新过慢。迭代格式：

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$





尝试基于P11或P14代码修改实现

Repeat{

for i=1, 11, 21, 31, … , 991{

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every j=0, … , n)

}

}

是朋友，就不要让自己的兄弟用大于32的mini-batch！

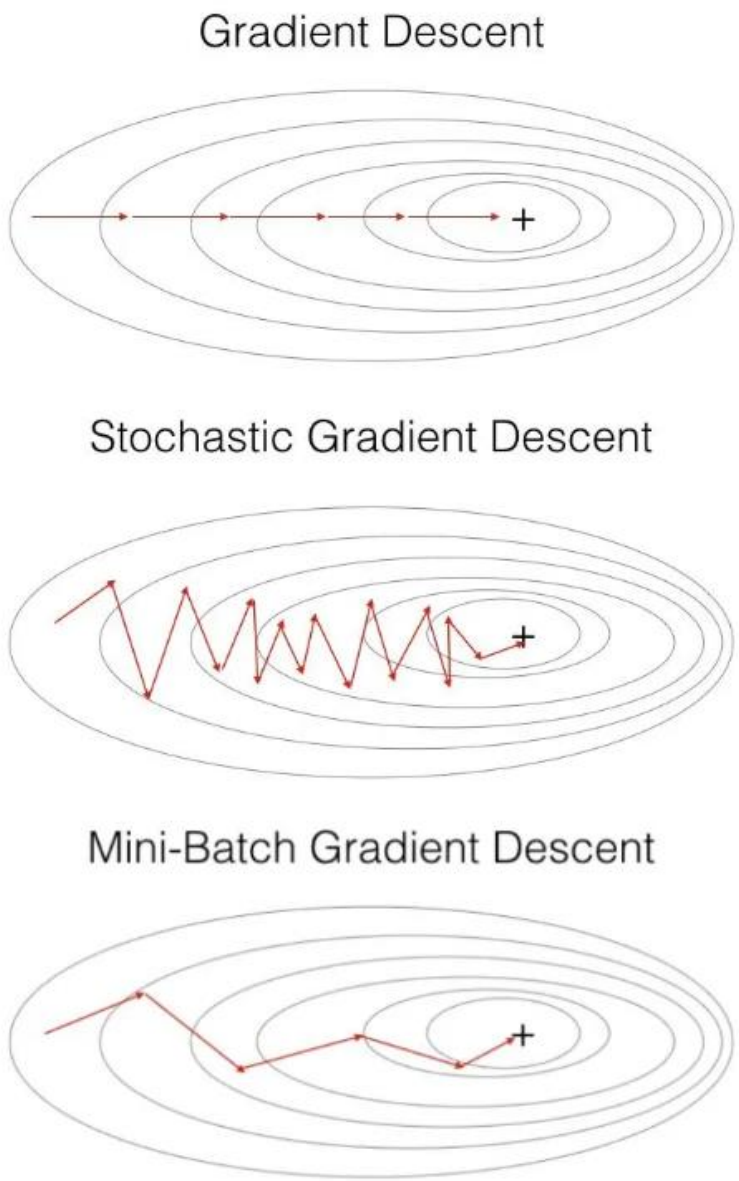
论智

365 人赞同了该文章

编者按：对于现代深度神经网络的训练来说，如果要用随机梯度下降来收敛，我们一般会选用Mini-Batch，这也是工程界最常用的做法。尽管大批量可以为并行计算提供更多算力空间，但小批量已经被证明了通用性更好，占用内存更少，而且收敛速度更快。那么，常见的mini-batch从几十到几百不等，我们又该怎么往哪个方向调试呢？近日，智能芯片创业公司Graphcore的两位工程师就在论文Revisiting Small Batch Training for Deep Neural Networks中给出了建议——**2到32之间。**

小结：批量算法 & 小批量算法

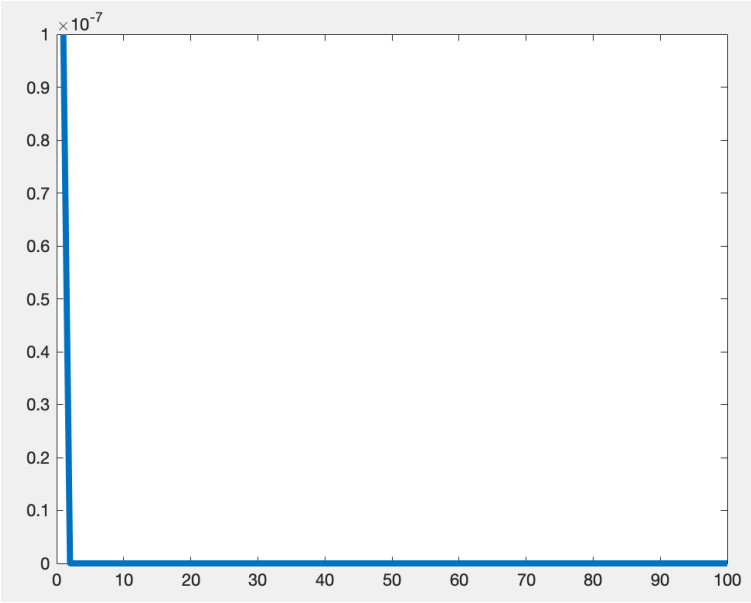
- 使用整个训练集的优化算法被称为**批量（batch）**或**确定性（deterministic）梯度算法**，因为它们会在一个大批量中同时处理所有样本。
- 每次只使用单个样本的优化算法有时被称为**随机（stochastic）**或者**在线（on-line）算法**。通常是指从连续产生样本的数据流中抽取样本的情况。
- 大多数用于深度学习的算法介于以上两者之间，称为**小批量（minibatch）方法**。



基于Matlab实现的函数逼近算例

求$f(x) = x_1^2 + x_2^2 + x_3^2$最小值

- 四层神经网络，神经元个数为 1，4，2，3
- 输入值为**1**，输出为函数取最小值的自变量值
- 主程序见下一页（完整代码参考**demo_nn_approx.m** ），输出结果如下：




```
outputs =

   1.0e-03 *

    0.1815
    0.1837
    0.1825


minfun =

   9.9969e-08
```

2. 梯度下降法推广

参考论文：An overview of gradient descent optimization algorithms
地址：https://arxiv.org/pdf/1609.04747.pdf

1. Momentum
2. Nesterov accelerated gradient
3. Adagrad
4. Adadelta
5. RMSprop
6. Adam
7. AdaMax
8. Nadam

报告专家：林宙辰 教授（北京大学）

报告时间：2022.6.17—2022.7.22，每周五20：00—21：30

报告地点：腾讯会议ID：721-6788-2449

邀请人：袁景 西安电子科技大学 浙江师范大学 教授

主持人：唐晓颖 南方科技大学 教授

王珊珊 深圳先进研究院 研究员

摘要：优化算法是机器学习的重要组成部分，但是传统的优
的参数维度很高或涉及的样本数巨大，这使得一阶优化算法

来源：http://sjxy.zjnu.edu.cn/2022/0614/c13738a398142/page.htm

1. Momentum

论文：Learning representations by back-propagating errors
地址：https://www.iro.umontreal.ca/~vincentp/ift3395/lectures/backprop_old.pdf

- Momentum是对梯度下降法的改进，使用了过去的梯度信息。Momentum可以在不牺牲易用性和局部性的前提下显著加速收敛。
- 迭代公式：

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$

$$\theta = \theta - v_t$$

- 参数γ通常设为0.9，η为学习率

伪代码：

f = f($x_1, x_2, \ldots, x_n$)

g = gradient(f)

初始化：起始点x，学习率η，x点负梯度d,γ

for n from 1 to N:

    d = γd- η g(x)

    x = x+d

end

- 代码参考：**gradient descent_exe2d**

算例展示

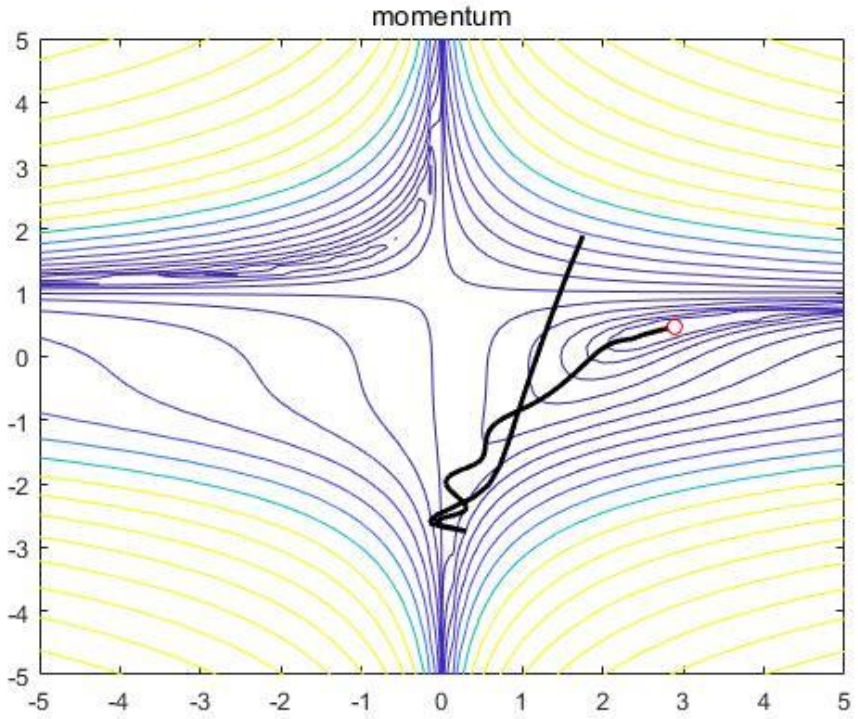
二维函数表达式：$f(x,y)=(1.5-xy)^2+\left(2.25-x+xy^2\right)^2+\left(2.65-x+xy^3\right)^3$

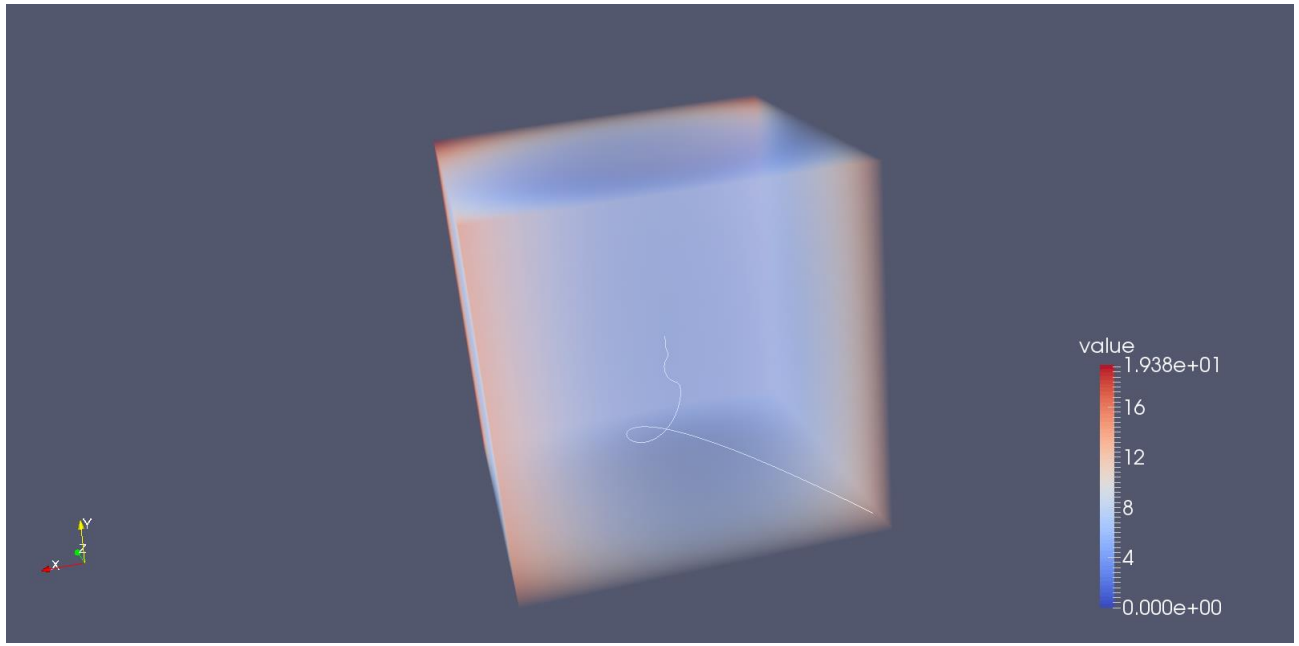
三维函数表达式：$f(x,y,z)=x^2+0.1y^2+2z^2$

参数设置：

二维：$\eta=0.001$，γ=0.9，起始点 [1.75 , 1.9]

三维：η = 0.01 ，γ=0.9，起始点 [-2,-2,-2]





2. Nesterov accelerated gradient

论文：A method of solving a convex programming problem with convergence rate O(1/k^2)
地址：http://mpawankumar.info/teaching/cdt-big-data/nesterov83.pdf

- 对momentum进行改进，每次迭代使用的梯度为未来参数位置的预测值的梯度，能够在函数值上升之前减小梯度。
- 迭代公式：

$$v_t=\gamma v_{t-1}+\eta\nabla_\theta J(\theta-\gamma v_{t-1})$$

$$\theta=\theta-v_t$$

伪代码：

f = f(x_1, x_2,…, x_n)

g = gradient(f)

初始化：起始点x，学习率η，x点负梯度d,γ

for n from 1 to N:

d = γd- η g(x+ γd)

x = x+d

end

return x

算例展示

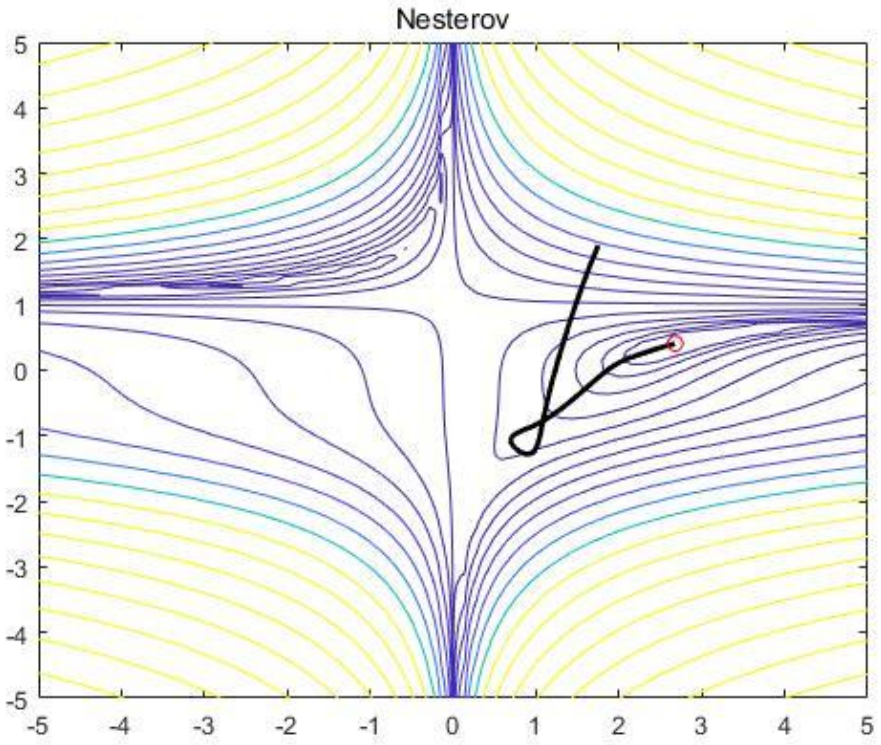
二维函数表达式： $f(x,y)=(1.5-xy)^2+\left(2.25-x+xy^2\right)^2+\left(2.65-x+xy^3\right)^3$

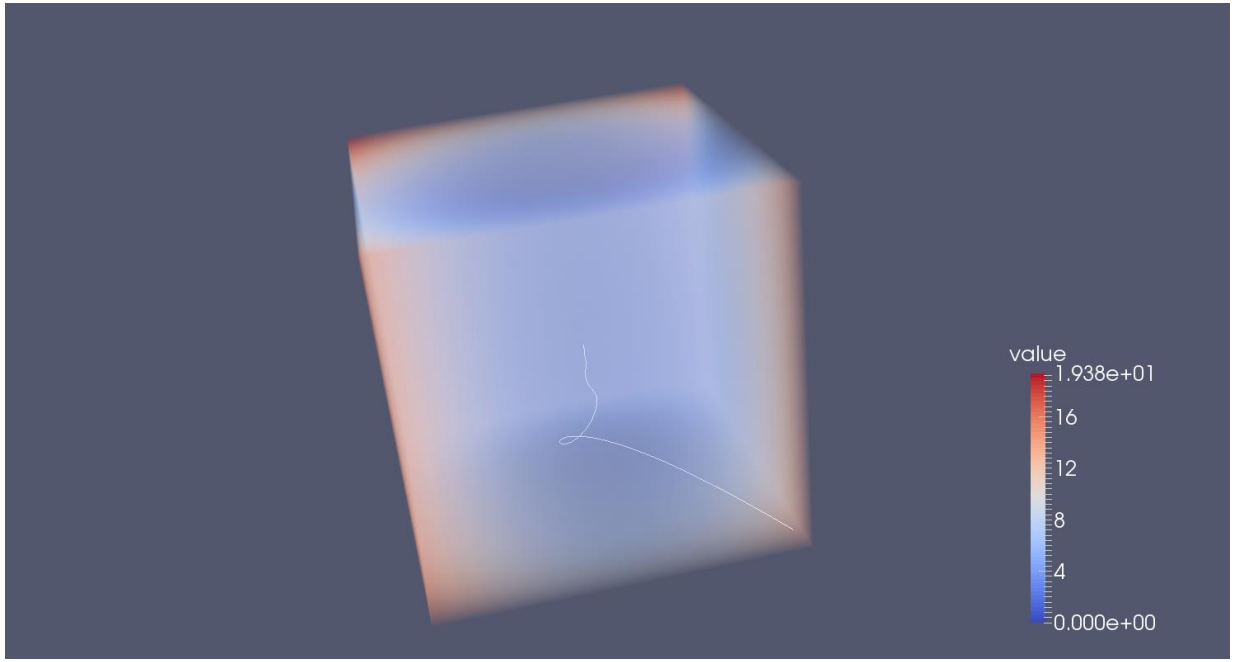
三维函数表达式： $f(x,y,z)=x^2+0.1y^2+2z^2$

参数设置：

二维：$\eta=0.0005$， γ=0.9，起始点 [1.75 , 1.9]

三维：η = 0.01 ， γ=0.9，起始点 [-2,-2,-2]





3. Adagrad

- Adagrad实现了根据参数调整学习率。
- 迭代公式：$g_{t,i} = \nabla_\theta J(\theta_{t,i})$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii}+\varepsilon}} \cdot g_{t,i}$$

- Adagrad的缺点在于分母上平方梯度的增加，导致学习速率下降。

论文： Adaptive Subgradient Methods for Online Learning and Stochastic Optimization
地址： http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

伪代码：

f = f(x_1, x_2,…, x_n)

g = gradient(f)

初始化：起始点x，学习率η，G，ε

for n from 1 to N:

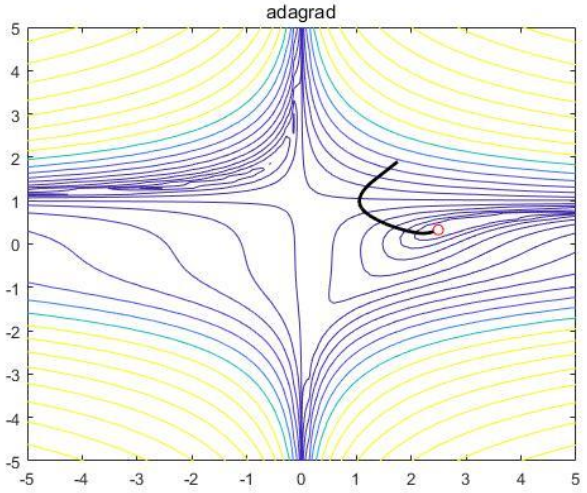
d = -g(x)

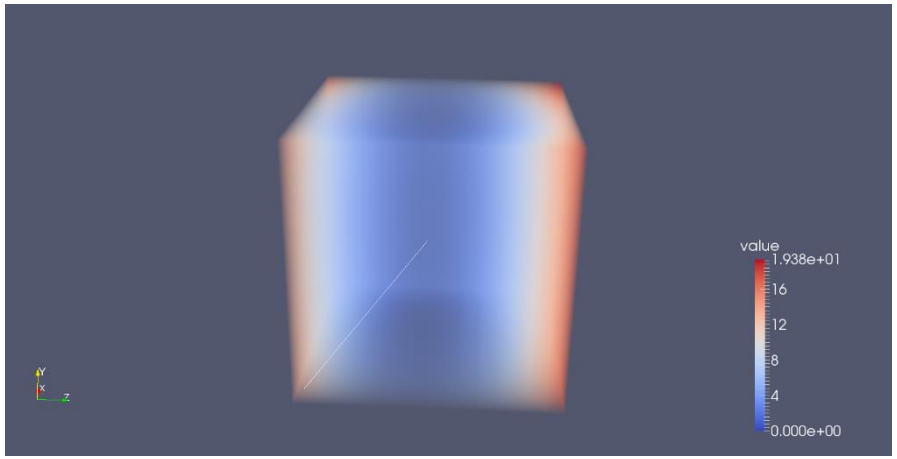
G += d^2

x = x+$\frac{\eta}{\sqrt{G+\varepsilon}}$d

end

return x





参数设置：

二维：

η 0.5

ε 1e-6

起始点 [1.75 , 1.9]

三维：

η 0.5

ε 1e-6

起始点 [-2,-2,-2]

4. Adadelta

论文：ADADELTA: An Adaptive Learning Rate Method
地址：https://arxiv.org/pdf/1212.5701.pdf

- Adadelta是Adagrad的扩展，对Adagrad两处改进:

  1. 对梯度累计方式进行改进，使学习率不会趋于0
  2. 采用二阶优化的思想进行参数更新

- 迭代公式:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

with $\Delta\theta_t = -\frac{\sqrt{E[\Delta\theta^2]_{t-1}+\varepsilon}}{\sqrt{E[g^2]_t+\varepsilon}} g_t$ and $\theta_{t+1} = \theta_t + \Delta\theta_t$

伪代码:

f = f($x_1, x_2, \ldots, x_n$)

g = gradient(f)

初始化：起始点x，$E[g^2]_t$，$E[\Delta\theta^2]_t$，γ，ε

for n from 1 to N:

d = -g(x)

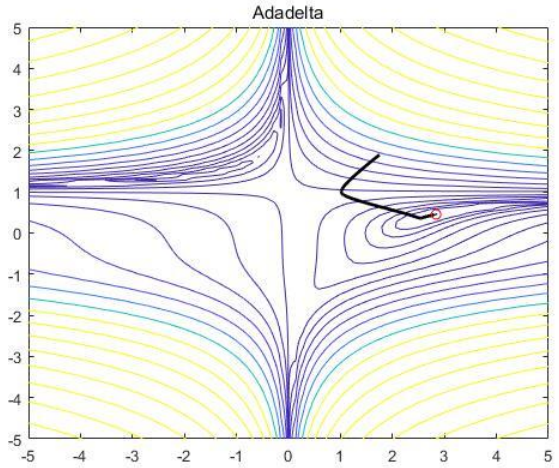
$E[g^2]_t = \gamma E[g^2]_t + (1-\gamma) d^2$

x = x+$\frac{\sqrt{E[\Delta\theta^2]_t+\varepsilon}}{\sqrt{E[g^2]_t+\varepsilon}}$d

$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_t + (1-\gamma)\left(\frac{\sqrt{E[\Delta\theta^2]_t+\varepsilon}}{\sqrt{E[g^2]_t+\varepsilon}} d\right)^2$

end

return x



二维:

γ 0.9

ε 1e-6

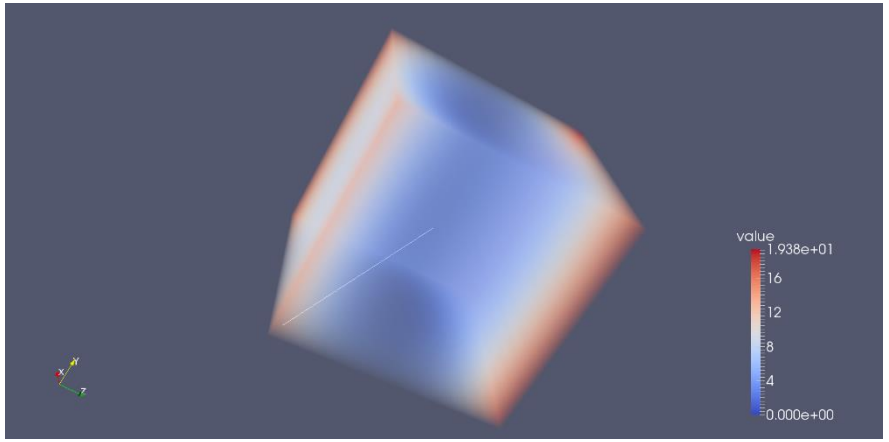
起始点 [1.75 , 1.9]

三维:

γ 0.9

ε 1e-5

起始点 [-2,-2,-2]



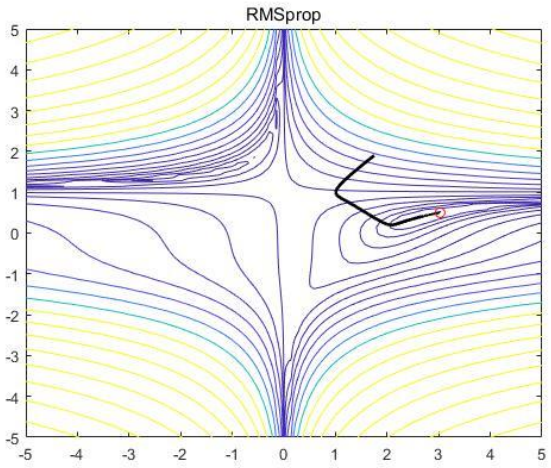
5. RMSprop

http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

RMSprop是与Adadelta同时提出的Adagrad改进算法，未以论文的形式进行发表，而是在Geoff Hinton 教授的课程中被提及，结合梯度平方的指数移动平均数来调节学习率的变化。能够在不稳定的目标函数情况下进行很好地收敛。迭代公式：

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} g_t$$



二维：

η 0.01

ε 1e-8

γ 0.9

起始点 [1.75 , 1.9]

三维：

η 0.01

ε 1e-8

γ 0.9

起始点 [-2,-2,-2]

f = f(x_1, x_2,…, x_n)

g = gradient(f)

初始化：起始点x，学习率η，$E[g^2]_t$，γ，ε

for n from 1 to N:

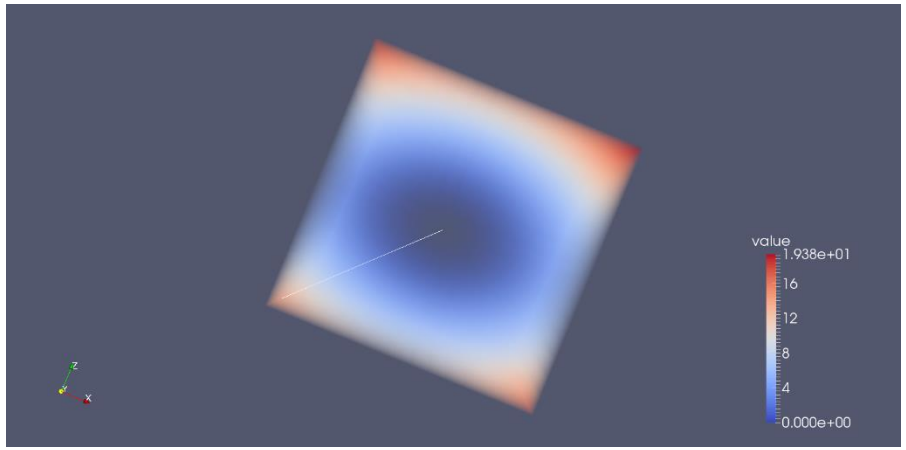
d = -g(x)

$E[g^2]_t = \gamma E[g^2]_t + (1-\gamma) d^2$

x = x+$\frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}}$d

end

return x



6. Adam

论文： Adam: A Method for Stochastic Optimization
地址： https://arxiv.org/pdf/1412.6980.pdf

2014年，Kingma和Lei Ba提出了Adam优化器，结合AdaGrad和RMSProp两种优化算法的优点。对梯度的一阶矩估计（First Moment Estimation，梯度均值）和二阶矩估计（Second Moment Estimation，梯度未中心化的方差）进行综合考虑，计算出更新步长：

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

with $\hat{m_t} = \frac{m_t}{1-\beta_2^n}, \hat{v_t} = \frac{v_t}{1-\beta_1^n}$ and $\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v_t}+\varepsilon}} \hat{m_t}$

f = f($x_1, x_2, \ldots, x_n$)

g = gradient(f)

初始化：x，η，β_1，β_2，m_t，v_t

for n from 1 to N:

$$m_t = \beta_1 m_t + (1-\beta_1) g(x)$$

$$v_t = \beta_2 v_t + (1-\beta_2) g(x)^2$$

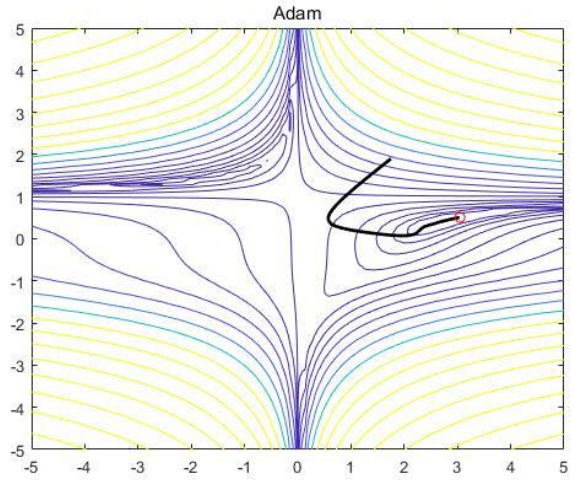
$$\hat{m_t} = \frac{m_t}{1-\beta_2^n}$$

$$\hat{v_t} = \frac{v_t}{1-\beta_1^n}$$

$$x = x - \frac{\eta}{\sqrt{\hat{v_t}+\varepsilon}} \hat{m_t}$$

end

return x



二维：

β_1 0.9， β_2 0.97

ε 1e-6

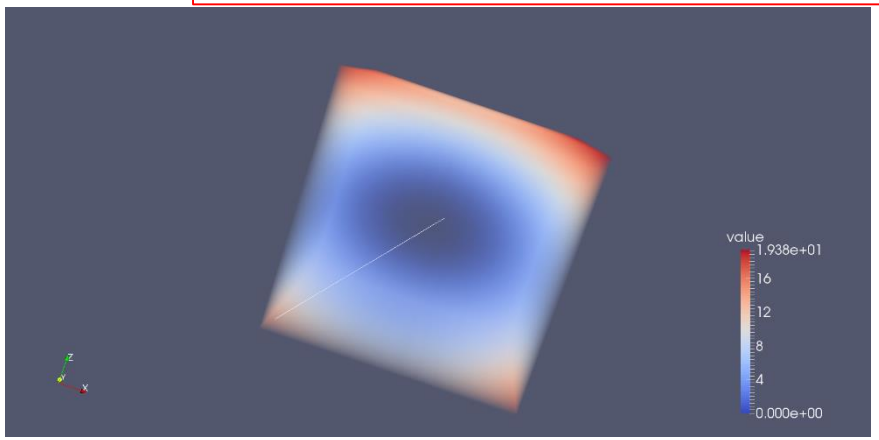
起始点 [1.75 , 1.9]

三维：

β_1 0.9， β_2 0.999

ε 1e-6

起始点 [-2,-2,-2]



7. AdaMax

论文： **Adam: A Method for Stochastic Optimization**
地址： https://arxiv.org/pdf/1412.6980.pdf

AdaMax将Adam中将基于 L_2 范数的更新规则泛化到基于 L_p范数的更新规则中。虽然这样会因为 p 的值较大而在数值上变得不稳定，令 p → ∞则会得出一个极其稳定和简单的算法：

$$v_t = \max(\beta_2 \cdot v_{t-1}, |g_t|)$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{v_t} \hat{m_t}$$

f = f(x_1, x_2,…, x_n)

g = gradient(f)

初始化：x，η，β_1，β_2，m_t，v_t

for n from 1 to N:

$$m_t = \beta_1 m_t + (1-\beta_1) g(x)$$

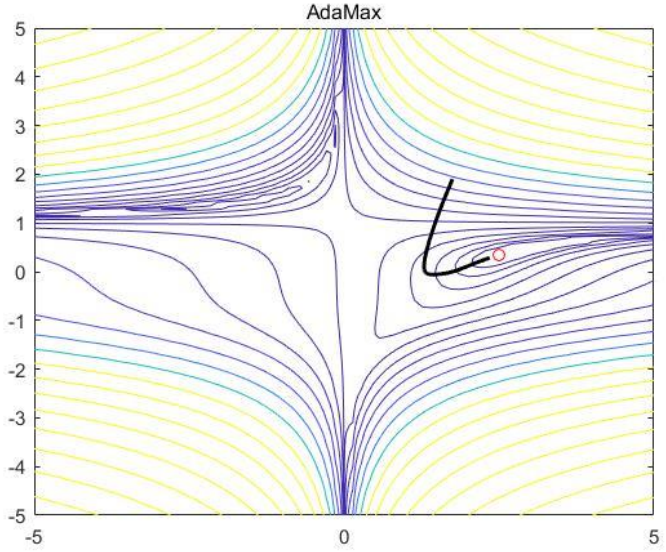
$$v_t = \max(\beta_2 \cdot v_t, |g_t|)$$

$$\hat{m_t} = \frac{m_t}{1-\beta_2^n}$$

$$x = x - \frac{\eta}{v_t} \hat{m_t}$$

end

return x



二维：

β_1 0.9, β_2 0.999

ε 1e-6, η 0.5

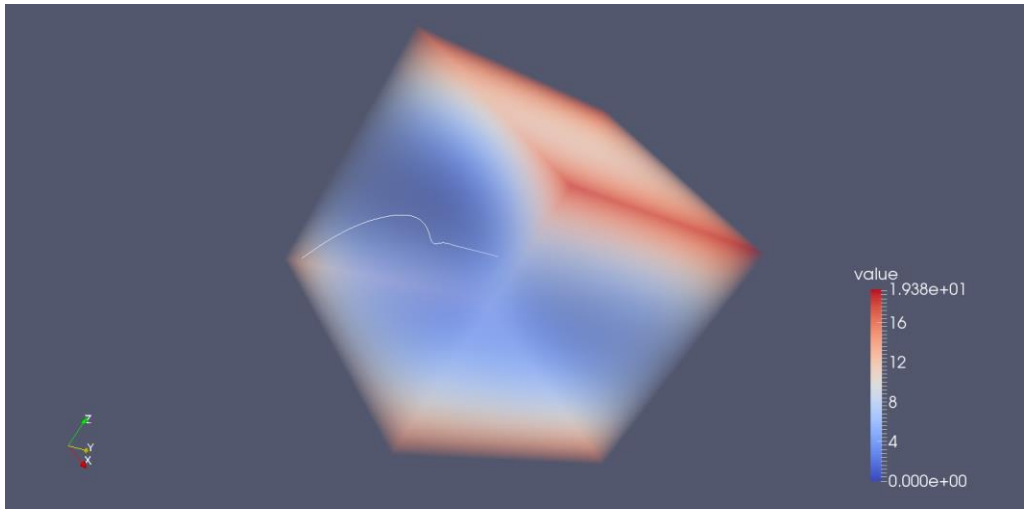
起始点 [1.75 , 1.9]

三维：

β_1 0.9, β_2 0.999

ε 1e-6, η 0.1

起始点 [-2,-2,-2]



8. Nadam

论文：Incorporating Nesterov Momentum into Adam
地址：https://openreview.net/pdf?id=OM0jvwB8jIp57ZJjtNEZ

Nadam将Nesterov加速方法应用在Adam算法上：

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v_t} + \varepsilon}}\left(\beta_1 \hat{m_t} + \frac{(1-\beta_1)g_t}{1-\beta_1^t}\right)$$

f = f(x_1, x_2,…, x_n), g = gradient(f)

初始化：x，η，β_1，β_2，m_t，v_t，γ，ε

for n from 1 to N:

$$m_t = \beta_1 m_t + (1-\beta_1)g(x)$$

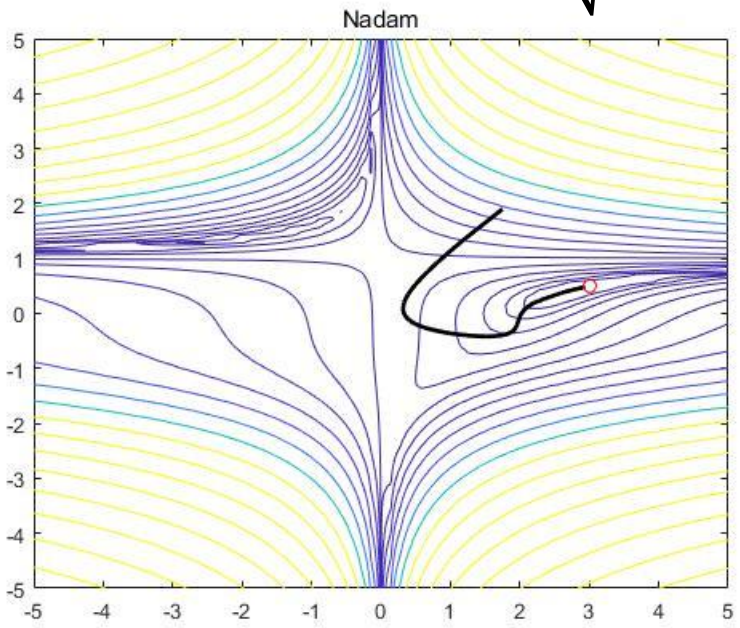
$$v_t = \beta_2 v_t + (1-\beta_2)g(x)^2$$

$$\hat{m_t} = \frac{m_t}{1-\beta_2^n}, \quad \hat{v_t} = \frac{v_t}{1-\beta_1^n}$$

$$x = x - \frac{\eta}{\sqrt{\hat{v_t}+\varepsilon}}\left(\beta_1 \hat{m_t} + \frac{(1-\beta_1)g(x)}{1-\beta_1^n}\right)$$

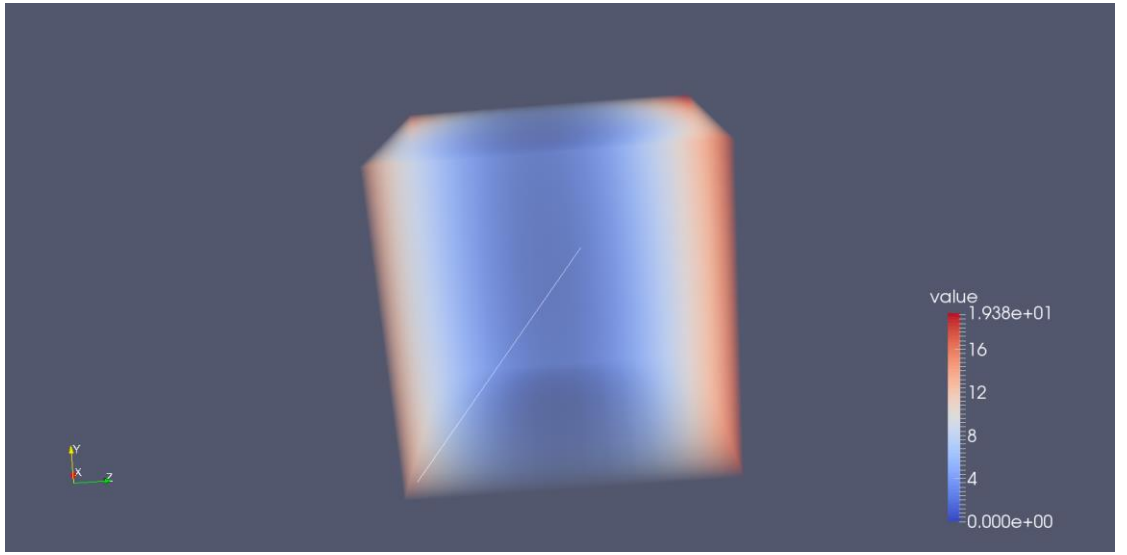
end

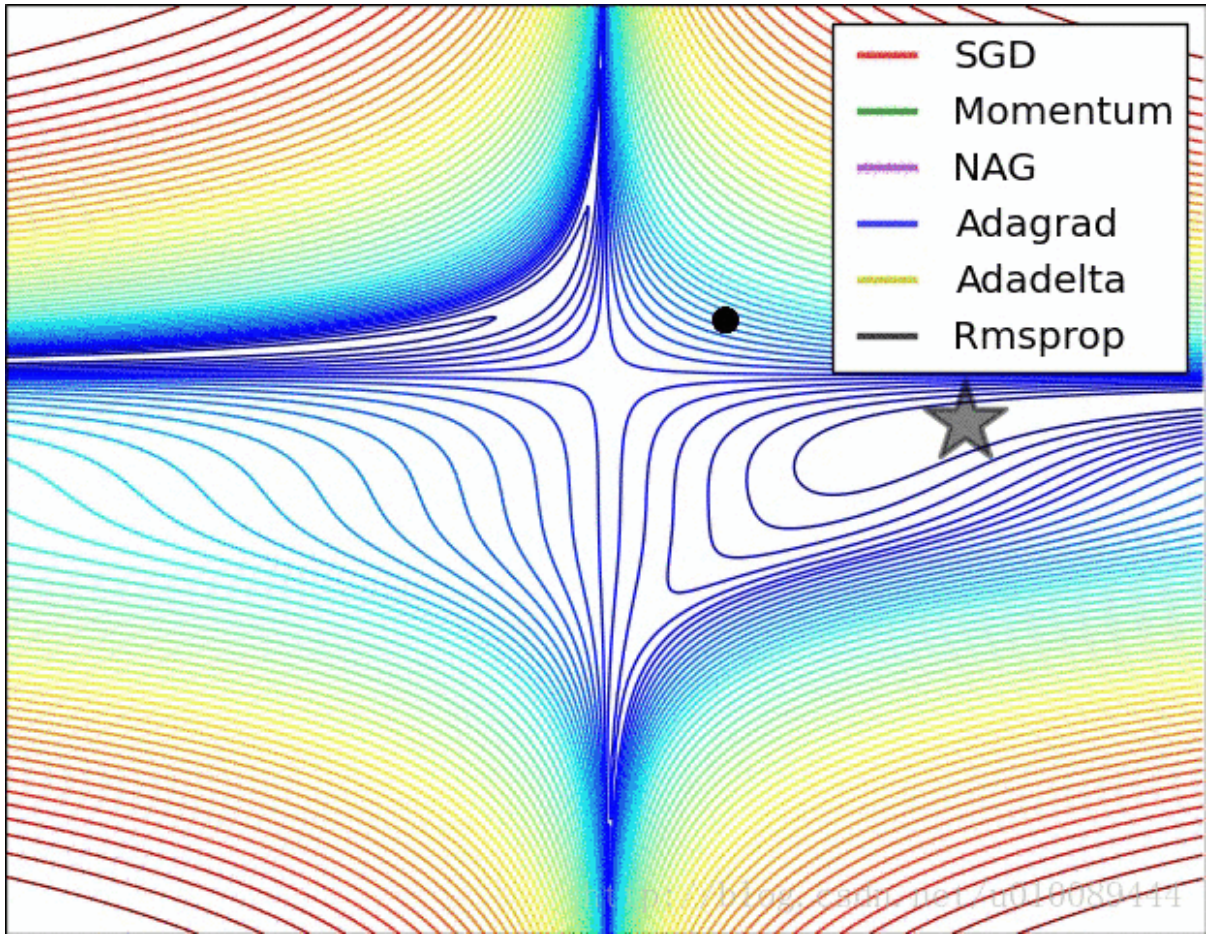
return x

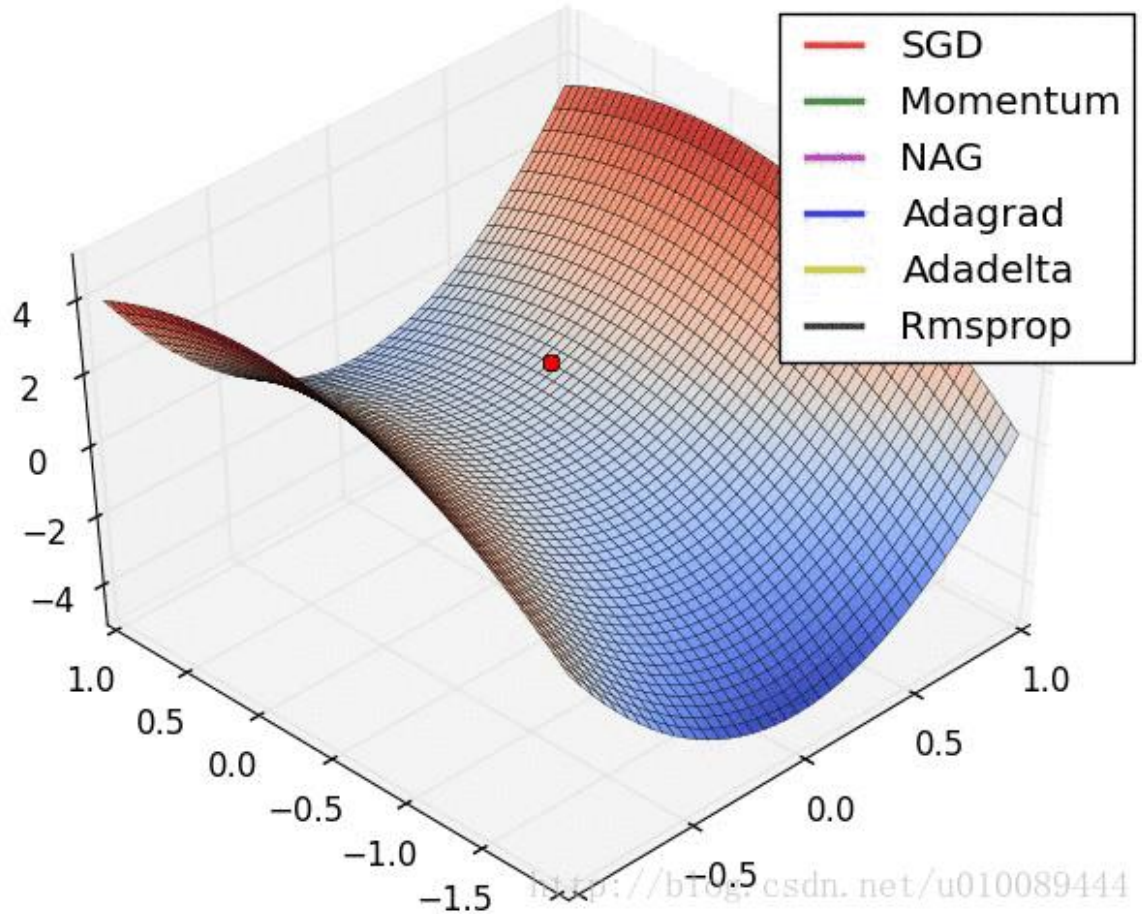


二维：
β_1 0.9, β_2 0.999
ε 1e-6, η 0.3
起始点 [1.75 , 1.9]

三维：
β_1 0.9, β_2 0.999
ε 1e-6, η 0.3
起始点 [-2,-2,-2]




SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

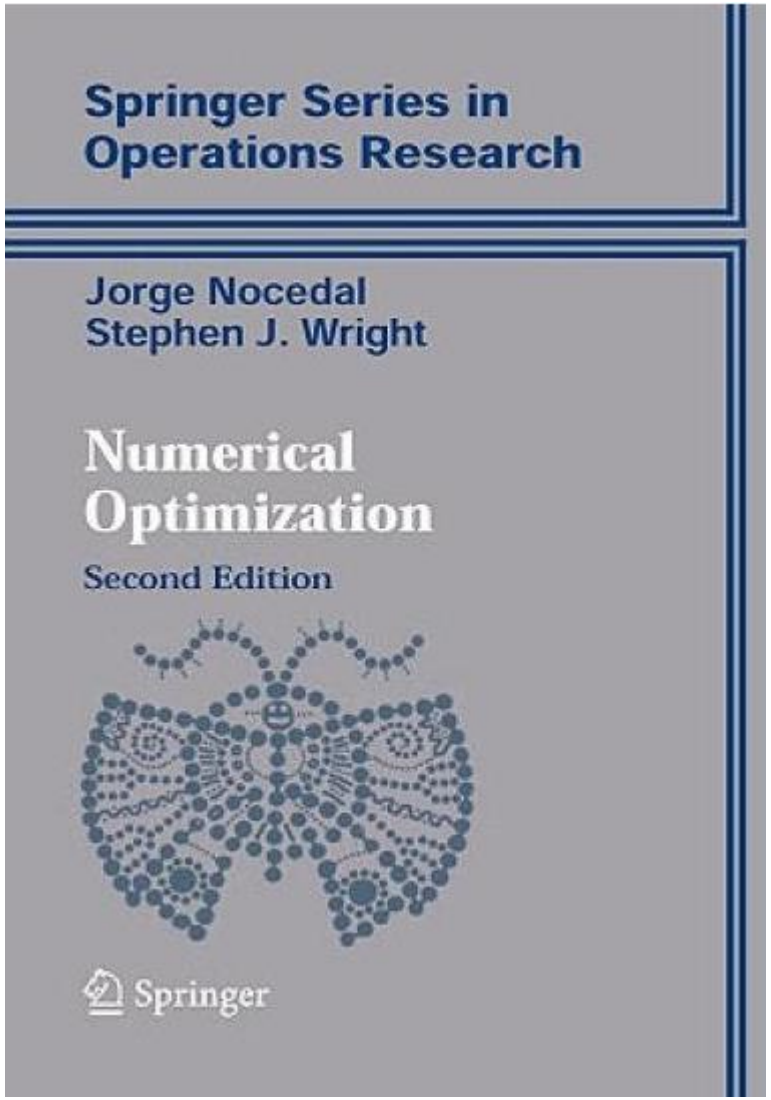

SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

3 二阶近似方法

二阶近似（数值最优化）方法

- 仅适用梯度信息的优化算法称为一阶优化算法（first-order）
- 使用Hessian矩阵的优化算法称为二阶优化算法（second-order, Nocedal and Wright,2006）
- 在本节中，我们会讨论训练深度神经网络的二阶方法。为表述简单起见，我们只考察目标函数为经验风险：

$$E_{(x,y)\sim\hat{p}_{data}}[L(f(x;\theta),y)]=\frac{1}{m}\sum_{i=1}^{m}L(f(x^{(i)};\theta),y^{(i)})$$



牛顿法

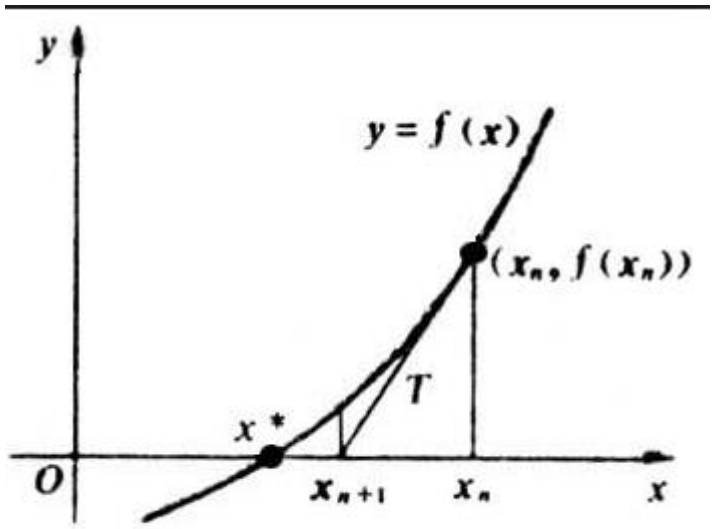
- 牛顿法基于二阶泰勒展开来近似θ_0 附近的$J(\theta)$，忽略高阶导数，有：

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_\theta J(\theta_0) + \frac{1}{2}(\theta - \theta_0)^T H(\theta - \theta_0)$$

- 其中H 是J 相对于θ的Hessian 矩阵在 θ_0 处的估计。如果我们再求解这个函数的临界点，我们将得到牛顿参数更新规则：

$$\theta^* = \theta_0 - H^{-1}\nabla_\theta J(\theta_0)$$

- 对于局部的二次函数（具有正定的H ），牛顿法会直接跳到极小值。如果目标函数是凸的但非二次的（有高阶项），该更新将会持续迭代改善。



牛顿法求实根图示

牛顿法的缺陷

- 只适用于Hessian矩阵正定的情况。目标函数非凸时，需通过正则化Hessian矩阵来避免落入鞍点:

$$\theta^* = \theta_0 - [\mathrm{H}(\mathrm{f}(\theta_0) + \alpha\mathrm{I}]^{-1}\nabla_\theta \mathrm{f}(\theta_0)$$

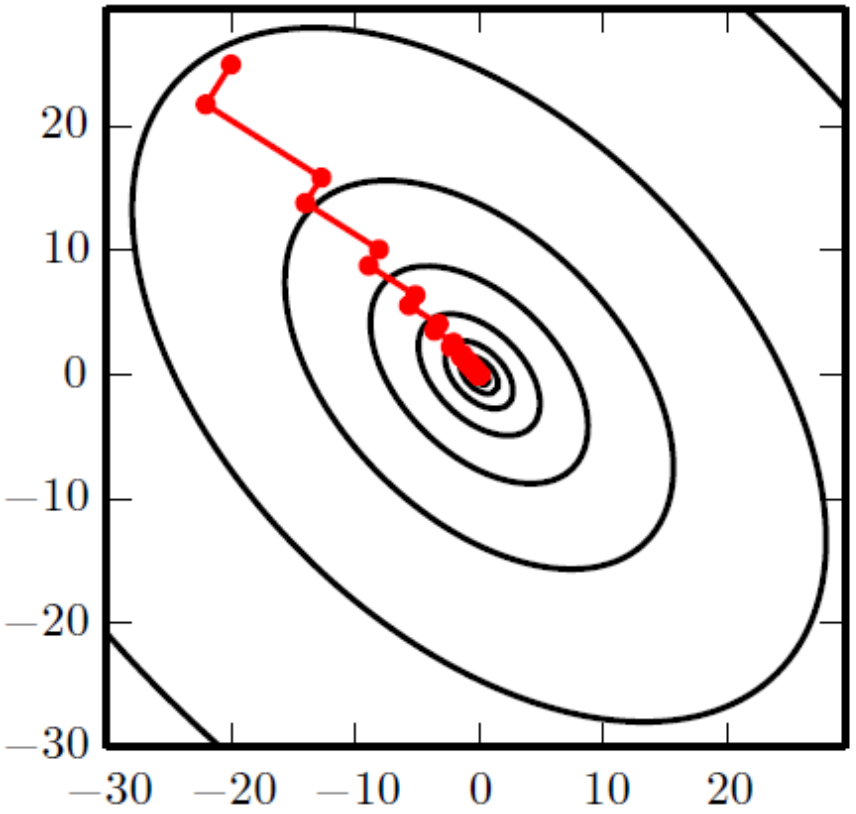
- 在曲率方向极端的情况下，α的值须足够大以抵消负特征值。但如果持续增大α，Hessian 矩阵会变得由对角矩阵主导，此时计算的方向收敛到普通梯度除以α!
- 牛顿法用于训练大型神经网络受限于其显著的计算负担。如：参数数目为k，牛顿法需要计算$k \times k$阶矩阵的逆，计算复杂度为 $\mathrm{O}(k^3)$!

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x \partial x} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial x \partial z} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y \partial y} & \frac{\partial^2 f}{\partial y \partial z} \\ \frac{\partial^2 f}{\partial z \partial x} & \frac{\partial^2 f}{\partial z \partial y} & \frac{\partial^2 f}{\partial z \partial z} \end{bmatrix}$$

共轭梯度法

Magnus R. Hestenes and Eduard Stiefel ,Methods of conjugate gradients for solving linear systems, J. Research Nat. Bur. Standards 49, 409–436. 1952.

- 共轭梯度是一种通过迭代下降的**共轭方向（conjugate directions）**以有效避免Hessian矩阵求逆计算的方法。
- 这种方法的灵感来自于对梯度下降方法弱点的仔细研究。
- 梯度下降法在二次碗型目标中如何表现为一个锯齿形模式。这是因为每一个由梯度给定的线搜索方向，都保证正交于上一个线搜索方向。

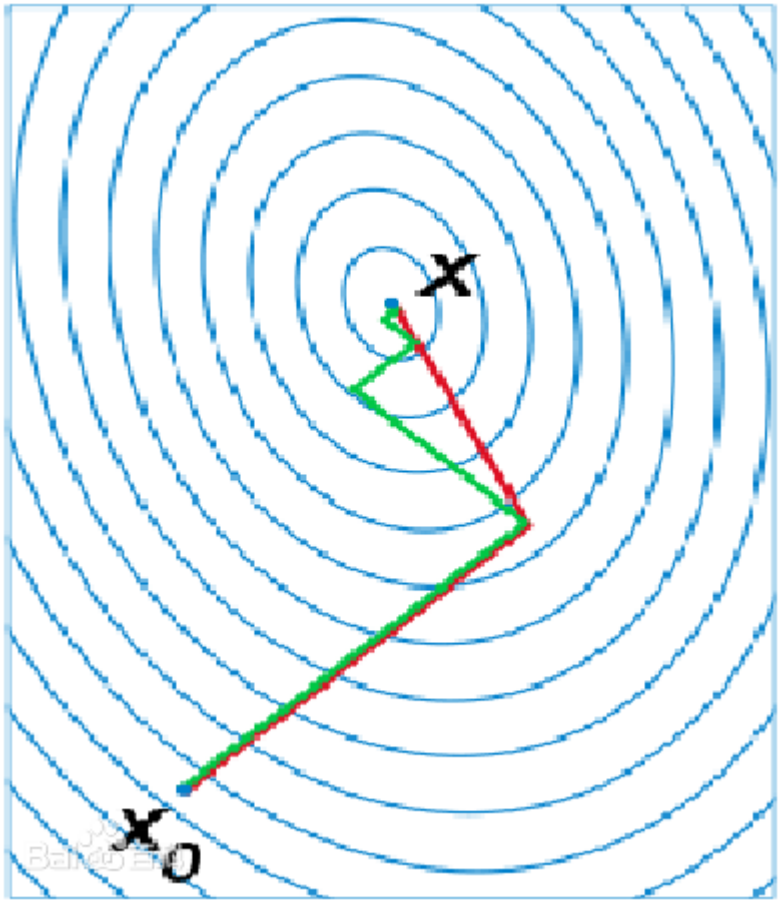


共轭梯度法

- 在共轭梯度法中，我们寻求一个和先前线搜索方向共轭(conjugate)的搜索方向，即它不会撤销该方向上的进展。在训练迭代t 时，下一步的搜索方向d_t 的形式如下：

$$d_t = \nabla_\theta J(\theta) + \beta_t d_{t-1}$$

- 如果 $d_t^T H d_{t-1} = 0$ ，其中H 是Hessian 矩阵，则两个方向被称为共轭的。适应共轭的直接方法会涉及到H 特征向量的计算以选择 β_t 。



β_t 的计算

算法 8.9 共轭梯度方法

Require: 初始参数 $\boldsymbol{\theta}_0$

Require: 包含 m 个样本的训练集

初始化 $\boldsymbol{\rho}_0 = 0$

初始化 $\boldsymbol{g}_0 = 0$

初始化 $t = 1$

while 没有达到停止准则 **do**

初始化梯度 $\boldsymbol{g}_t = 0$

计算梯度：$\boldsymbol{g}_t \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

计算 $\beta_t = \frac{(\boldsymbol{g}_t - \boldsymbol{g}_{t-1})^\top \boldsymbol{g}_t}{\boldsymbol{g}_{t-1}^\top \boldsymbol{g}_{t-1}}$ (Polak-Ribière)

(非线性共轭梯度：视情况可重置 β_t 为零，例如 t 是常数 k 的倍数时，如 $k = 5$)

计算搜索方向：$\boldsymbol{\rho}_t = -\boldsymbol{g}_t + \beta_t \boldsymbol{\rho}_{t-1}$

执行线搜索寻找：$\epsilon^* = \text{argmin}_\epsilon \frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_t + \epsilon \boldsymbol{\rho}_t), \boldsymbol{y}^{(i)})$

（对于真正二次的代价函数，存在 ϵ^* 的解析解，而无需显式地搜索）

应用更新：$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t$

$t \leftarrow t + 1$

end while

Fletcher-Reeves:

$$\beta_t = \frac{\nabla_\theta J(\theta_t)^T \nabla_\theta J(\theta_t)}{\nabla_\theta J(\theta_{t-1})^T \nabla_\theta J(\theta_{t-1})}$$

Polak-Ribière:

$$\beta_t = \frac{(\nabla_\theta J(\theta_t) - \nabla_\theta J(\theta_{t-1}))^T \nabla_\theta J(\theta_t)}{\nabla_\theta J(\theta_{t-1})^T \nabla_\theta J(\theta_{t-1})}$$

应用数学学报 ›› 2015, Vol. 38 ›› Issue (1) : 89-97. DOI: 10.12387/C2015009

论文

一个具有下降性的改进Fletcher-Reeves共轭梯度法

马国栋, 简金宝, 江羡珍

作者信息 +

An Improved Fletcher-reeves Conjugate Gradient

MA Guodong, JIAN Jinbao, JIANG Xianzhen

Author information +

History +

摘要

对无约束优化问题, 本文给出了一个改进的Fletcher-Reeves共轭梯度法. 不依赖于任何线搜
生的搜索方向均是下降的. 在标准Wolfe非精确线搜索准则下, 证明了算法的全局收敛性. 数
出的方法有效.

Abstract

In this paper, an improved Fletcher-Reeves conjugate gradient method is propose
optimization. The direction generated by the improved method provides a desce
objective function not depending on any line search. Under the standard Wolfe lir
convergence of the proposed method is proved. Some elementary numerical exper
which show that the proposed method is efficient.

关键词

无约束优化 / 共轭梯度法 / 全局收敛性

Key words

unconstrained optimization / conjugate gradient method / global convergence

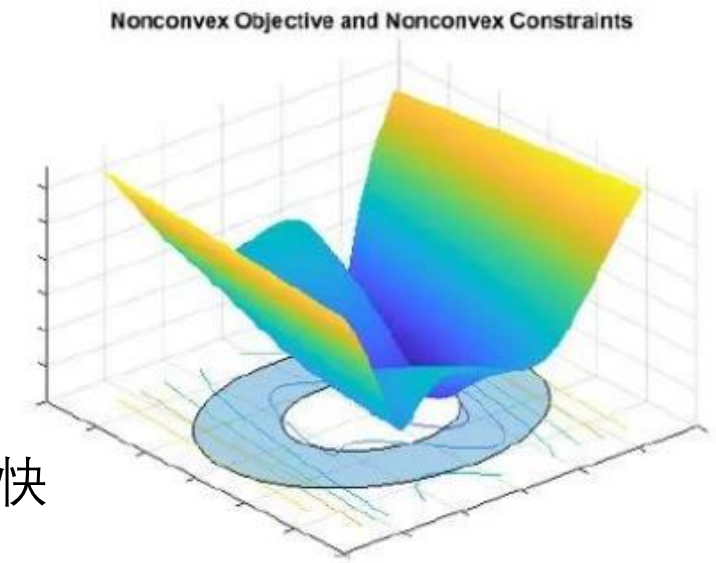
BFGS

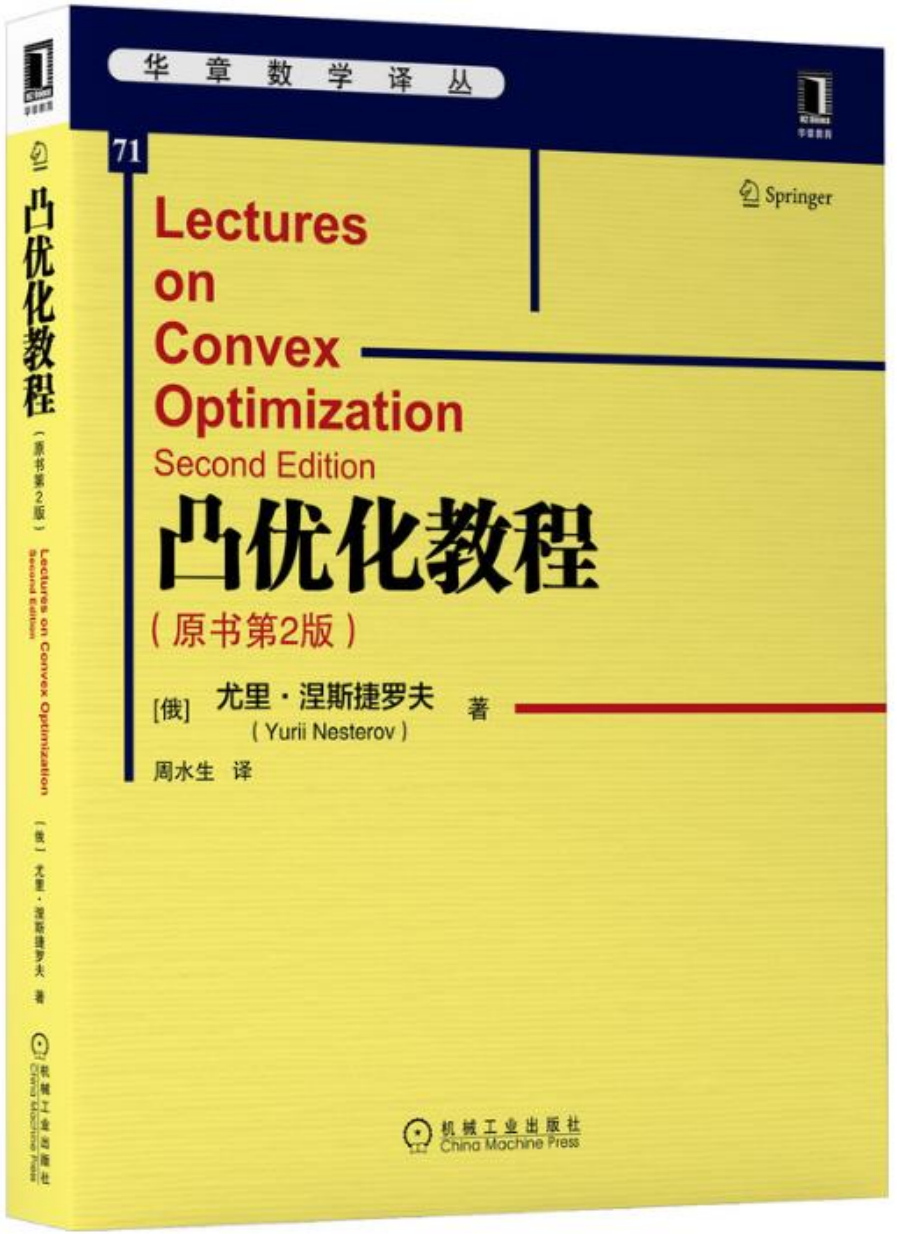
- **Broyden-Fletcher-Goldfarb-Shanno（BFGS）**算法具有牛顿法的一些优点，但没有牛顿法的计算负担。
- 运用牛顿法的主要计算难点在于计算Hessian 逆矩阵。拟牛顿法所采用的方法是使用矩阵M_t近似逆，迭代地低秩更新精度以更好地近似逆。
- 当Hessian 逆近似M_t更新时，下降方向为$\rho_t = M_t g_t$。该方向上的线搜索用于决定该方向上的步长ε^*。参数的最后更新为：

$$\theta_{t+1} = \theta_t + \varepsilon^* \rho_t$$

关于优化问题的讨论

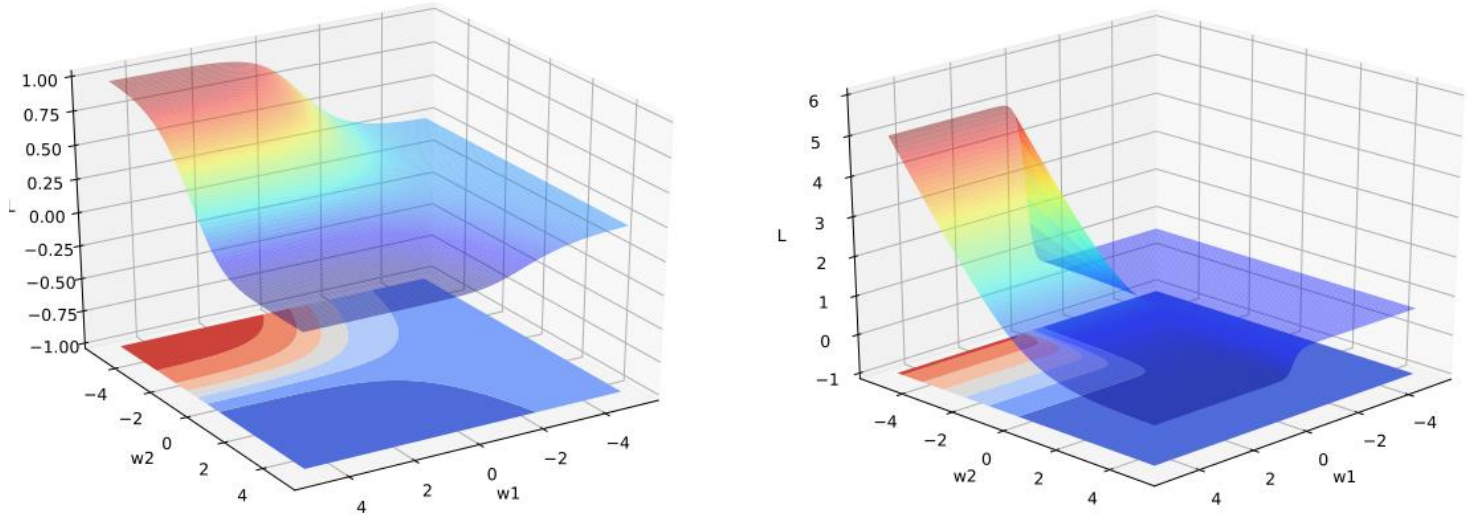
- 难点
  - 参数过多，影响训练
  - 非凸优化问题：即存在局部最优而非全局最优解
  - 梯度消失问题，下层参数比较难调
  - 参数解释起来比较困难
- 需求
  - 计算资源要大
  - 数据要多
  - 算法效率要好：即收敛快





问题：凸与非凸

- 在训练神经网络时，肯定会遇到一般的非凸情况。但经验非凸函数在极小值附近也具有局部凸的良好性质。

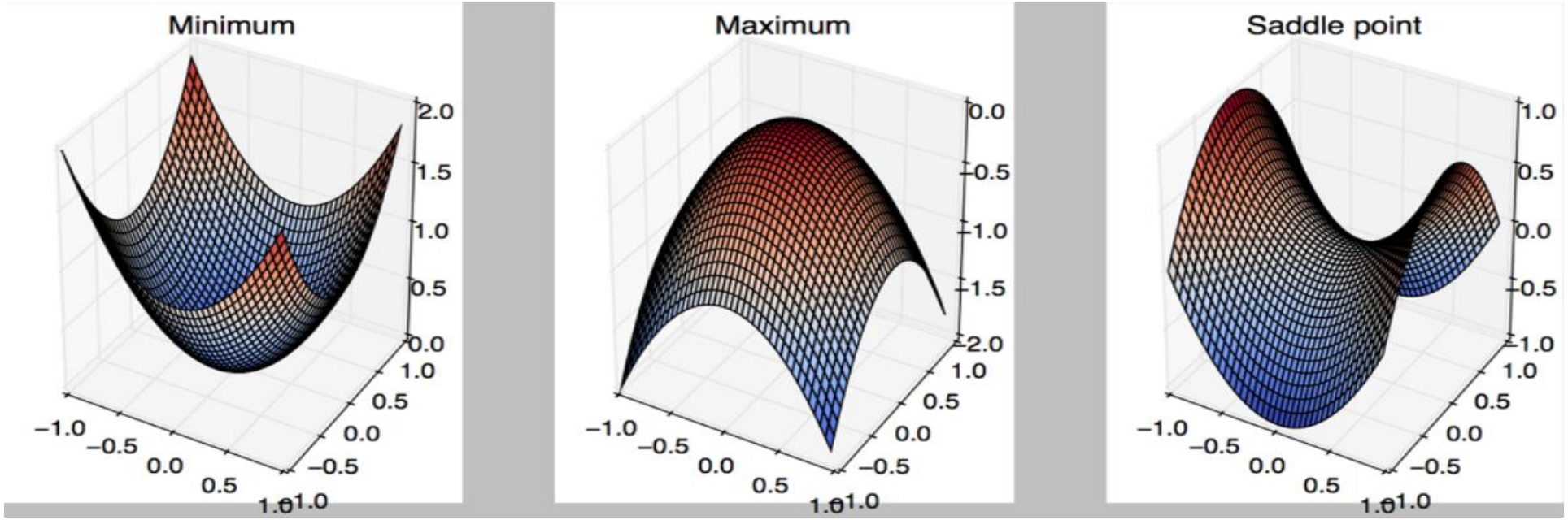


(a) 平方误差损失　　(b) 交叉熵损失

图 4.9 神经网络 $y = \sigma(w_2\sigma(w_1x))$ 的损失函数

- 但即使是凸优化，也会遇到一些挑战。其中最突出的就是Hessian矩阵的病态问题

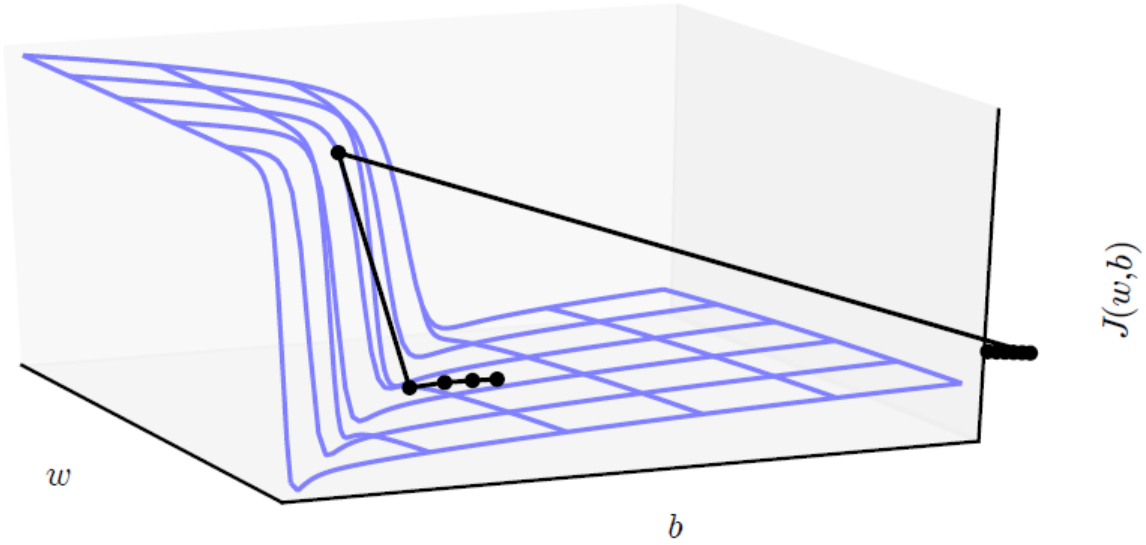
问题：高原、鞍点和其他平坦区域



- 梯度下降旨在朝“下坡”移动，而并非明确求得临界点。实验表明，梯度下降似乎可以在许多情况下逃离鞍点。Goodfellow et al(2015)也主张，应该可以通过分析来表明连续时间的梯度下降会逃离而不是吸引到鞍点。
- 对于牛顿法来说，鞍点显然是一个问题。牛顿法的目标是寻求梯度为0 的点。如果没有适当地修改，牛顿法就会跳进一个鞍点。Dauphin et al.(2014)介绍了二阶优化的无鞍牛顿法比传统算法有显著改进。

问题：梯度悬崖

- 多层神经网络通常存在像悬崖一样的斜率较大区域，如图所示。这是由于几个较大的权重相乘导致的。遇到斜率极大的悬崖结构时，梯度更新会很大程度地改变参数值，通常会完全跳过这类悬崖结构。
- 可以使用启发式**梯度截断（gradient clipping）**来避免其严重的后果。

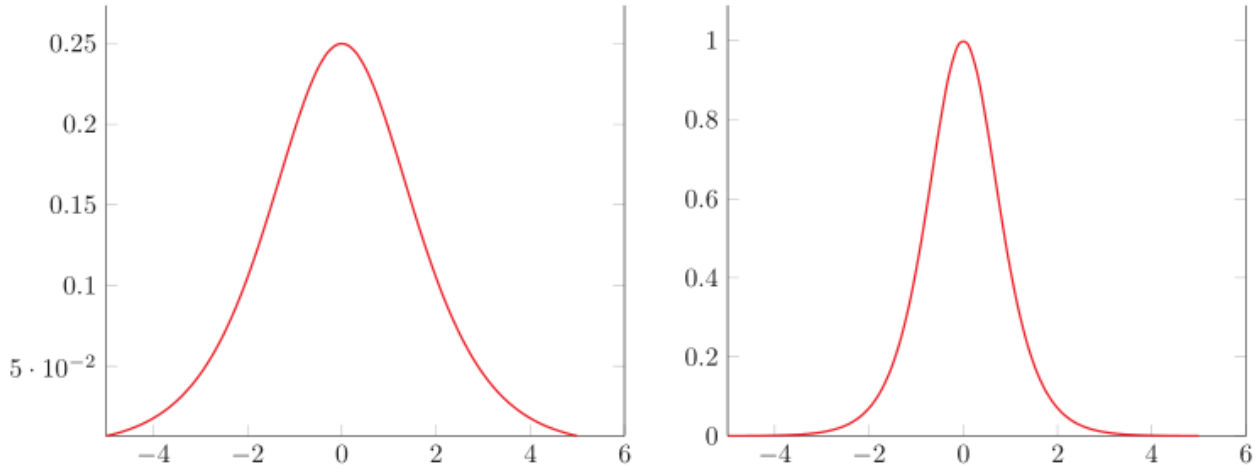


问题：长期依赖

- 当计算图变得极深时，神经网络优化算法会面临的另外一个难题就是长期依赖问题：由于变深的结构使模型丧失了学习到先前信息的能力，让优化变得极其困难。
- 特征值不在1附近时，若在量级上大于1则会爆炸；若小于1时则会消失。这就是**梯度消失与爆炸问题**
- **梯度消失**（gradient vanishing）

$$y = f^5(f^4(f^3(f^2(f^1(x)))))$$

$$\frac{\partial y}{\partial x} = \frac{\partial f^1}{\partial \mathrm{x}} \frac{\partial f^2}{\partial f^1} \frac{\partial f^3}{\partial f^2} \frac{\partial f^4}{\partial f^3} \frac{\partial f^5}{\partial f^4}$$



(a) logistic 函数的导数　　(b) tanh 函数的导数

练习：完成手写体识别的ANN程序，并比较GD，SGD以及ADAM的效率。