深度神经网络优化方法

PDL, School of Computer, NUDT

主要内容

- 批量梯度下降算法
  - 基本原理
  - 最速下降法
  - 批量、小批量和随机梯度下降算法
- 深度模型优化及调试策略
  - 实践流程
  - 调试策略
  - 挑战问题：病态、梯度消失、梯度爆炸
  - 参数调优
    - 参数初始化
    - 学习率、批量大小
    - 迭代次数

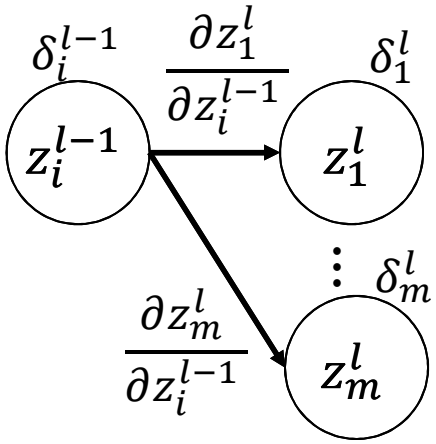
主要内容

- 其他梯度优化算法简介
  - 一阶算法
    - 基于动量的随机梯度下降算法
    - 学习率自适应随机梯度下降算法
  - 二阶算法
    - 牛顿法、共轭梯度算法

我们的反向传播总结

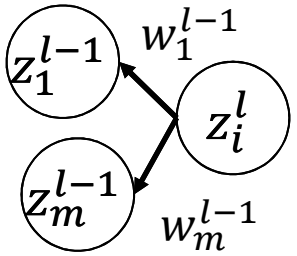
- 简单总结：
- 1. 最末节点求导：

$$\delta_j^L = \frac{\partial C}{\partial z_j^L}$$



- 2. 层间节点递推：

$$\delta_i^l = \frac{\partial C}{\partial z_i^l},\ \delta_i^{l-1} = \sum_{j=1}^{m} \frac{\partial z_j^l}{\partial z_i^{l-1}} \delta_j^l$$



w_1^{l-1}不出现在本层其他地方!

- 3. 参数偏导分离：（注意偏导分离规则只能用于W元素只出现在该节点，而不出现在其他节点！！！）

$$\frac{\partial C}{\partial W_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial W_j^l} = \delta_j^l \frac{\partial z_j^l}{\partial W_j^l}$$

怎样求w偏导？

*——卷积操作

×——矩阵乘操作

卷积核

w_{00}	w_{01}
w_{10}	w_{11}

3x3

*

输入

X_0	X_1	X_2	X_3
X_4	X_5	X_6	X_7
X_8	X_9	X_{10}	X_{11}
X_{12}	X_{13}	X_{14}	X_{15}

4x4

=

输出

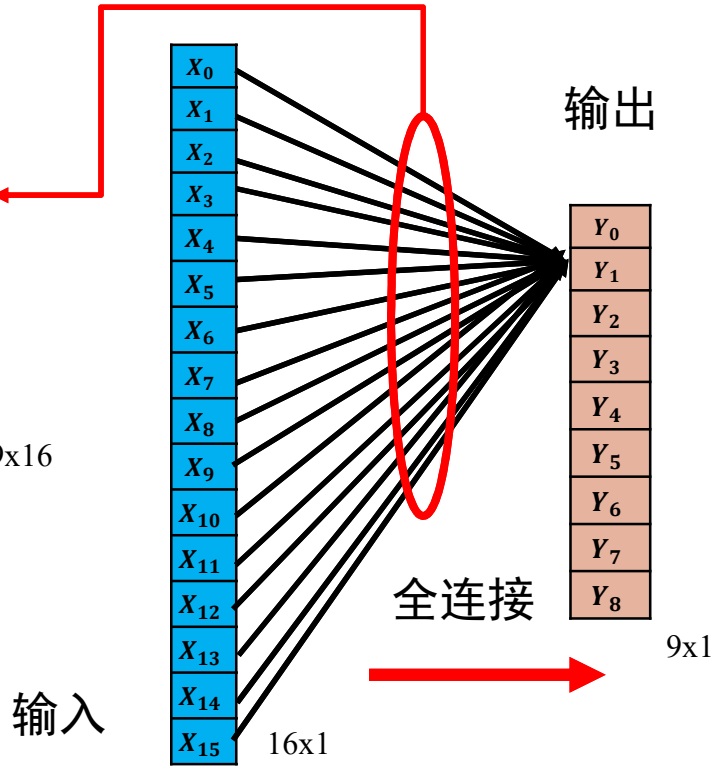
Y_0	Y_1	Y_2
Y_3	Y_4	Y_5
Y_6	Y_7	Y_8

3x3

进一步，卷积核展开矩阵可视为“全连接权值矩阵”：E(W)

w_{00}	w_{01}	0	0	w_{10}	w_{11}	0	0	0	0	0	0	0	0	0	0
0	w_{00}	w_{01}	0	0	w_{10}	w_{11}	0	0	0	0	0	0	0	0	0
0	0	w_{00}	w_{01}	0	0	w_{10}	w_{11}	0	0	0	0	0	0	0	0
0	0	0	0	w_{00}	w_{01}	0	0	w_{10}	w_{11}	0	0	0	0	0	0
0	0	0	0	0	w_{00}	w_{01}	0	0	w_{10}	w_{11}	0	0	0	0	0
0	0	0	0	0	0	w_{00}	w_{01}	0	0	w_{10}	w_{11}	0	0	0	0
0	0	0	0	0	0	0	0	w_{00}	w_{01}	0	0	w_{10}	w_{11}	0	0
0	0	0	0	0	0	0	0	0	w_{00}	w_{01}	0	0	w_{10}	w_{11}	0
0	0	0	0	0	0	0	0	0	0	w_{00}	w_{01}	0	0	w_{10}	w_{11}

9x16



$$W*X=Y$$

$$E(W)X=Y$$

W00在连线中出现多次，不能用我们的公式直接套！

怎样求w偏导？

*——卷积操作

×——矩阵乘操作

卷积核

w_{00}	w_{01}
w_{10}	w_{11}

3x3

*

输入

X_0	X_1	X_2	X_3
X_4	X_5	X_6	X_7
X_8	X_9	X_{10}	X_{11}
X_{12}	X_{13}	X_{14}	X_{15}

4x4

=

输出

Y_0	Y_1	Y_2
Y_3	Y_4	Y_5
Y_6	Y_7	Y_8

3x3

进一步，卷积核展开矩阵可视为“全连接权值矩阵”：E(X)

X_0	X_1	X_4	X_5
X_1	X_2	X_5	X_6
X_2	X_3	X_6	X_7
X_4	X_5	X_8	X_9
X_5	X_6	X_9	X_{10}
X_6	X_7	X_{10}	X_{11}
X_8	X_9	X_{12}	X_{13}
X_9	X_{10}	X_{13}	X_{14}
X_{10}	X_{11}	X_{14}	X_{15}

直接做全路径求和

$$\nabla w_{01} = \frac{\partial C}{\partial Y_0} x_1 + \frac{\partial C}{\partial Y_0} x_2 + \ldots + \frac{\partial C}{\partial Y_8} x_{15}$$

$$= \delta_0 x_1 + \delta_1 x_2 + \ldots + + \delta_8 x_{15}$$

相当于δ和$x(a^{l-1})$做卷积 9x16

$$\frac{\partial C}{\partial w^l} = \begin{pmatrix} \nabla w_{00} & \nabla w_{01} \\ \nabla w_{11} & \nabla w_{10} \end{pmatrix} = a^{l-1} * \delta^l$$

w_{00}
w_{01}
w_{10}
w_{11}

输出

δ_0
δ_1
δ_2
δ_3
δ_4
δ_5
δ_6
δ_7
δ_8

全连接

9x1

梯度下降方法：批量方法

- 梯度下降：使用所有的训练样本来计算目标函数$J(\theta)$的梯度，即：

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(f(x^{(i)};\theta), y^{(i)})$$

- 使用整个训练集的优化算法：批量（batch）或确定性（deterministic）算法【深度学习8.1.3】
- 批量算法的问题：
  - 计算代价：一次更新需要遍历所有训练样本求导
  - 数据冗余：大量样本可能都对梯度做了相似的贡献

梯度下降方法：小批量方法，随机方法

- 可以用采样的方法，对训练样本的梯度进行估计
- 按照数据生成分布抽取m个小批量（独立同分布）样本$\{x^{(1)}, x^{(2)} \ldots, x^{(m)}\}$通过计算其梯度均值可以得到梯度的无偏估计，此时

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L\left(f\left(x^{(i)};\theta\right), y^{(i)}\right)$$

- 使用部分训练样本的算法：小批量（minibatch）或小批量随机（ minibatch stocastic）方法。极端情况下，每次只用单个样本的称为随机（stocastic）或在线（online）算法。现在通常也将他们简单的称为随机（stocastic）方法【深度学习8.1.3】

随机梯度下降算法（SGD）

- SGD算法：

算法 8.1 随机梯度下降（SGD）在第 k 个训练迭代的更新

Require: 学习率 ϵ_k

Require: 初始参数 $\boldsymbol{\theta}$

while 停止准则未满足 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，其中 $\boldsymbol{x}^{(i)}$ 对应目标为 $\boldsymbol{y}^{(i)}$。

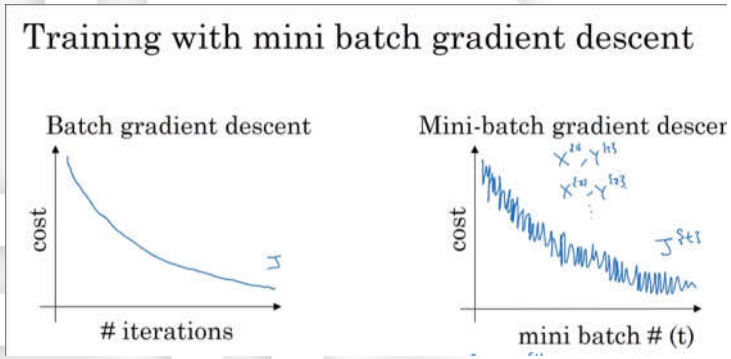
计算梯度估计：$\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

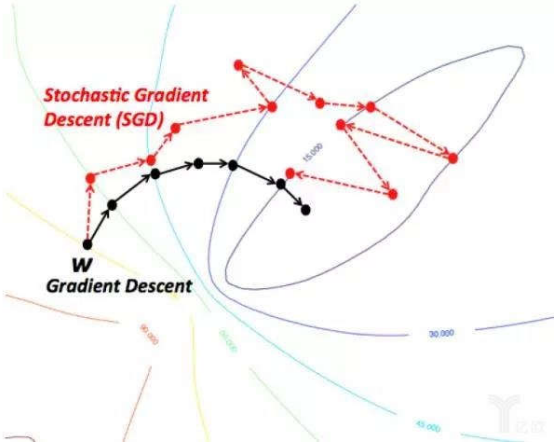
应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$

end while

随机梯度下降算法（SGD）

- 优点:
  - 训练迭代速度快,适合大数据集
- 缺点:
  - 抽样梯度存在误差，需要更多的迭代去收敛，后期需减小学习速率达到稳定收敛
  - 抽样梯度误差引起的学习曲线震荡





- 学习速率可以选常数，但更多使用动态调整：开始大（但又不能太大），快速迭代，后期小，稳定收敛

随机梯度下降算法（SGD）

- 学习速率,ϵ如何设置:
- 如果要保证SGD收敛,应该满足如下两个要求:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \qquad \sum_{k=1}^{\infty} \epsilon_k^2 < \infty.$$

- 在实际操作中,一般进行线性衰减:（开始大，后面小）

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau$$

- $\epsilon 0$是初始学习率, $\epsilon\tau$是最后一次迭代的学习率. τ代表迭代次数.一般来说,$\epsilon\tau$ 设为$\epsilon 0$的1%比较合适.
- 初始学习率$\epsilon 0$，和迭代次数τ如何设置？

优化及调试策略：实践流程

- 1. 确定目标：误差度量及期望目标
- 2. 建立端到端流程：
  - 数据读取、预处理流程
  - 端到端模型构建：基准模型（传统方法、经典模型） vs 设计模型
  - 性能度量、评测脚本
- 3. 搭建系统，确定性能瓶颈：
  - 调通系统：是否出现正确结果？
  - 预期判断：过拟合、欠拟合？
- 4. 根据观察进行增量式改进：
  - 收集新数据、调参、改进算法

优化及调试策略：调试策略

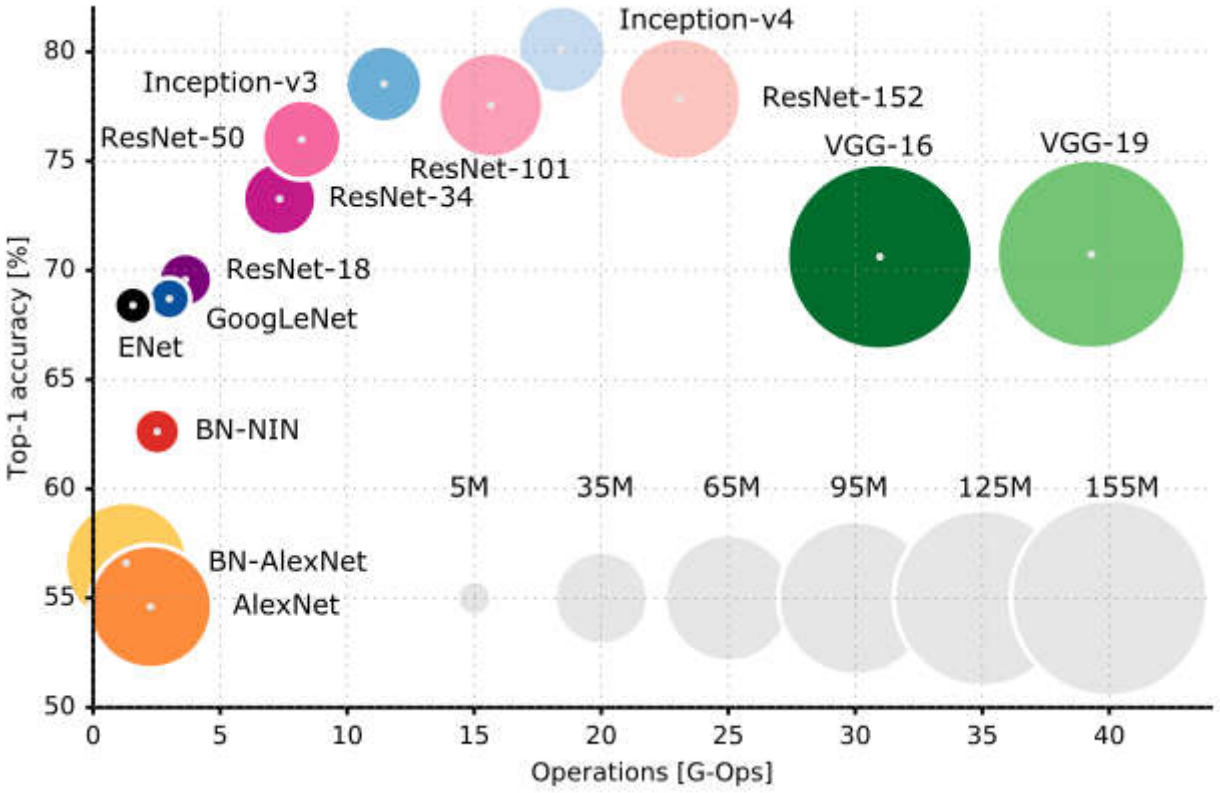
- 错误来源：算法本身还是编程实现
- 难点：1. 无法预测算法行为；2. 模型很多地方自适应
- 重要的调试检测手段：
- 1. 可视化计算中模型的行为（看中间，最后输出结果）
- 2. 可视化最严重的错误（看做错的例子）
- 3. 训练、测试误差分析：
  - 训练低，测试高：过拟合、训练模型未正确加载
  - 训练高，测试高：欠拟合、程序错误
- 4. 缩小问题规模，测试程序正确性
- 5. 可视化网络参数、梯度等

深度模型参数优化

- 参数优化：
- 1. 网络超参数
- 2. 参数初始化
- 3. 学习率
- 4. 批量大小
- 5. 正则化系数
- 6. 迭代次数

网络超参数

- 1. 根据问题的复杂度选择相应的baseline：lenet, alexnet, inception, resnet. 类比构建 (过拟合？欠拟合？参数量？计算效率？)

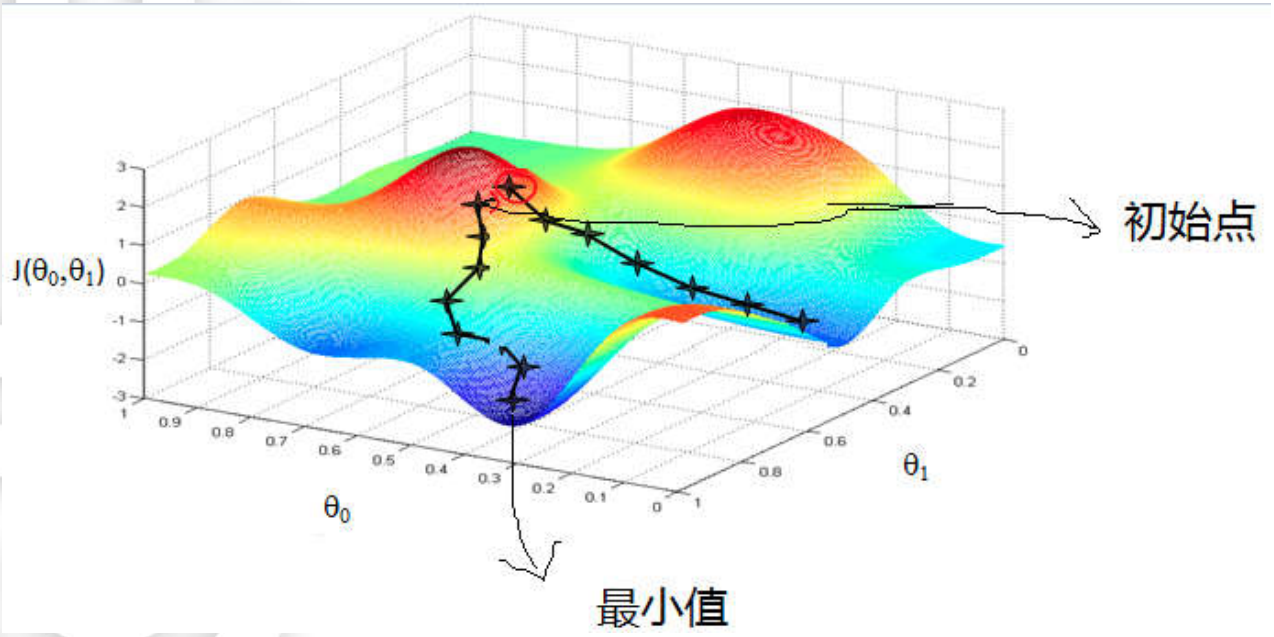


网络超参数

- 2. 卷积核大小：
- (1)小卷积核堆叠趋势（回顾典型CNN）：优：参数总数少，计算量减少，模型复杂度增。劣：网络层数增加，增加梯度弥散风险
- (2)size一般选择奇数，奇数核有”绝对中心”，方便步长、边缘填充设计：
- 原始图像n x n, 卷积核尺寸 f x f， padding p, stride s, 输出尺寸为：(n-f+2p)/s+1=(n-f+s+2p)/s 保证为整数，又要p两边对称（2p）
- 3. 卷结核个数：即是输出通道数
- (1) 越多提取特征越丰富，考虑计算量
- (2) 一般是2的幂次递增减64， 128， 256， 512， 1024 等

参数初始化

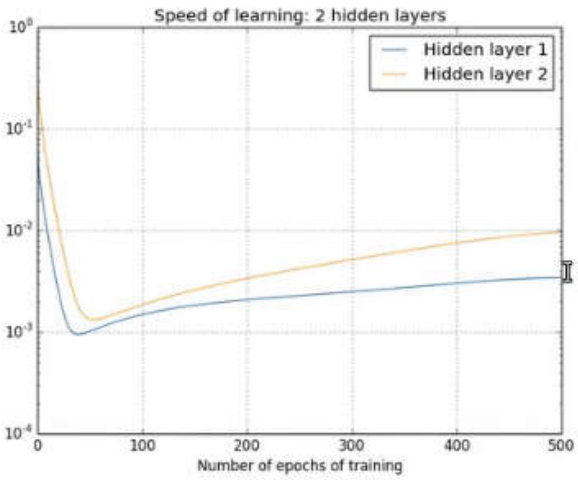
- 参数初始化：
- 1. 优化结果优劣（不只是最小值问题，梯度学习行为）
- 2. 训练是否可行（梯度消失、爆炸、训不动。。。）

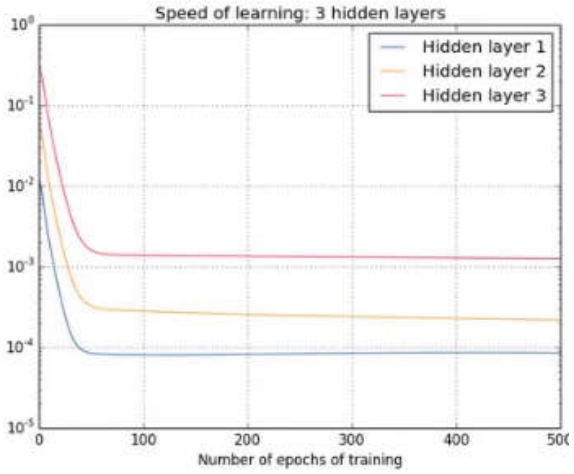


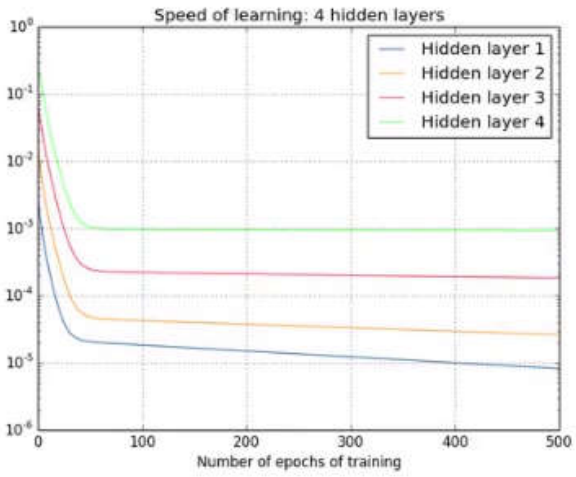
梯度下降挑战问题：梯度消失与梯度爆炸

- 梯度消失：随着网络层数的增加，网络从后往前，传回的梯度越来越小，导致之前的网络层“训不动”

另外一个例子:
[784,30,30,30,10]



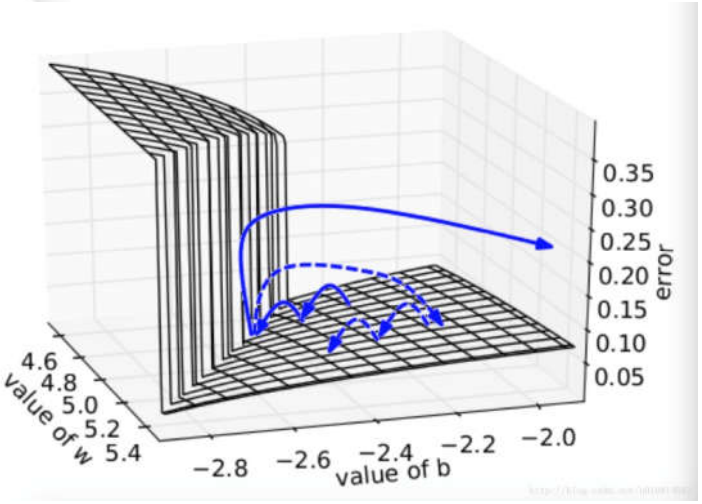




1000张训练图片，看出两层的学习率明显差异

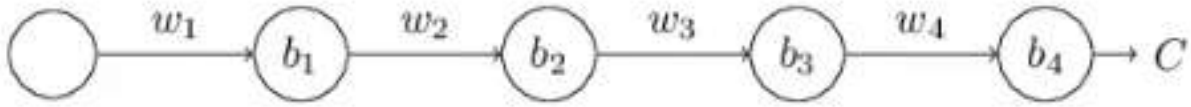
梯度下降挑战问题：梯度消失与梯度爆炸

- 梯度爆炸：梯度下降训练过程中遇到损失突变的位置（墙、悬崖等形态），梯度瞬间增大，指向某处不理想的位置。可能现象：
  - 训练无法收敛
  - 损失函数、权重：变化非常大 or 出现NaN值
  - 梯度：每层的每个节点训练时梯度一直大于1.0



梯度下降挑战问题：梯度消失与梯度爆炸

- 原因分析：
- 假设一个简单网络，每层只有一个神经元：



$\sigma(z_j)$

$z_j = w_j a_{j-1} + b_j$

梯度下降挑战问题：梯度消失与梯度爆炸

- 采用链式法则进行反向传播，获得最靠前的隐藏层神经元参数b_1的梯度形式：

$$\frac{\partial C}{\partial b_1}=\frac{\partial C}{\partial a_4}\frac{\partial a_4}{\partial z_4}\frac{\partial z_4}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial a_1}\frac{\partial a_1}{\partial z_1}\frac{\partial z_1}{\partial b_1}$$

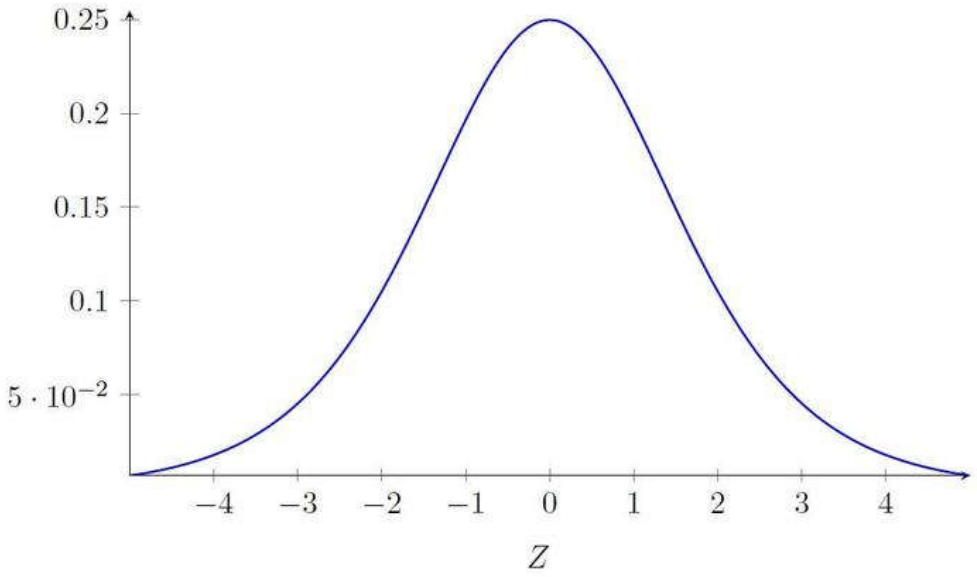
$$=\frac{\partial C}{\partial a_4}\sigma(z_4)'\frac{\partial(w_4a_3+b_1)}{\partial a_3}\sigma(z_3)'\frac{\partial(w_3a_2+b_3)}{\partial a_2}\sigma(z_2)'\frac{\partial(w_2a_1+b_2)}{\partial a_1}\sigma(z_1)'\frac{\partial(w_1a_0+b_1)}{\partial b_1}$$

$$=\frac{\partial C}{\partial a_4}\sigma(z_4)'\,w_4\,\sigma(z_3)'\,w_3\,\sigma(z_2)'\,w_2\,\sigma(z_1)'$$

- 整理得：$$\frac{\partial C}{\partial b_1}=\frac{\partial C}{\partial a_4}\sigma(z_4)'\,w_4\,\sigma(z_3)'\,w_3\,\sigma(z_2)'\,w_2\,\sigma(z_1)'$$
- 越靠近输出层的，乘性因子越多。导致神经元梯度的不稳定——容易过小或者过大，从而产生梯度消失或梯度爆炸。

梯度下降挑战问题：梯度消失与梯度爆炸

- 激活函数的影响：
- 假设网络参数初始化符合标准正态分布N(0,1)，激活函数采用Sigmoid函数，则有$|w| < 1$，而Sigmoid的导数$\sigma' = \sigma \times (1-\sigma) < 1/4$（见下面的Sigmoid导数曲线）。



梯度下降挑战问题：梯度消失与梯度爆炸

- 激活函数的影响：
- 受sigmoid连乘影响，梯度逐层递减，从而产生梯度消失

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \overbrace{\sigma(z_4)'}^{<\frac{1}{4}} w_4 \overbrace{\sigma(z_3)'}^{<\frac{1}{4}} w_3 \overbrace{\sigma(z_2)'}^{<\frac{1}{4}} w_2 \sigma(z_1)'$$

$$\frac{\partial C}{\partial b_2} = \frac{\partial C}{\partial a_4} \overbrace{\sigma(z_4)'}^{<\frac{1}{4}} w_4 \overbrace{\sigma(z_3)'}^{<\frac{1}{4}} w_3 \sigma(z_2)'$$

$$\frac{\partial C}{\partial b_3} = \frac{\partial C}{\partial a_4} \overbrace{\sigma(z_4)'}^{<\frac{1}{4}} w_4 \sigma(z_3)'$$

$$\frac{\partial C}{\partial b_4} = \frac{\partial C}{\partial a_4} \sigma(z_4)'$$

梯度下降挑战问题：梯度消失与梯度爆炸

- 权重的影响：
- 1. 初始化比较大的权重，例如w1=w2=w3=w4=100
- 2. 初始化b，使$\sigma(zj)'$不要过小，例如让z=0
- (b1=-100* a_0 $a_i=\sigma(z_i)$, z_1=w1* a_0 +b1=0)
- 此时

$$w_j\sigma'(z_j)$$

$$= 100 * 1/4 = 25$$

- 每一层梯度是前一层25倍，
- 第一层的梯度太大，出现爆炸问题

响：
比较大的权重，例如w1=w2=w3=w4=100
b，使$\sigma(zj)'$不要过小，例如让z=0
a_0 $a_i=\sigma(z_i)$, z_1=w1* a_0 +b1=0)

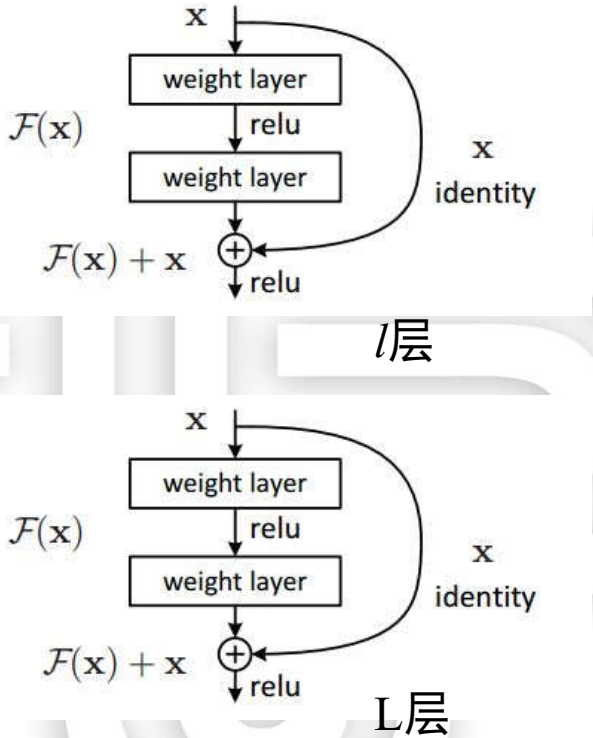
度是前一层25倍，
梯度太大，出现爆炸问题

梯度下降挑战问题：梯度消失与梯度爆炸

- 问题本质：梯度的乘积积累
- 期望效果：连续乘积都刚好平衡大约等于1，但几率很小
- 梯度爆炸更多来源于不合理的参数初始化及过大的学习率
- 梯度消失在更深的网络中更具有普遍性
  - 各层权重的分配比例，或每层用不同的学习率
  - 激活函数的问题：例如Sigmoid易饱和，考虑采用ReLU等

残差网如何缓解梯度消失

- Shortcut的加入，使得梯度可以以比例‘1’直接传给任意第l层，避免了梯度消失



$$x_{l+1} = x_l + \mathcal{F}(x_l, \mathcal{W}_l)$$

$$\begin{aligned} x_{l+2} &= x_{l+1} + \mathcal{F}(x_{l+1}, \mathcal{W}_{l+1}) \\ &= x_l + \mathcal{F}(x_l, \mathcal{W}_l) + \mathcal{F}(x_{l+1}, \mathcal{W}_{l+1}) \end{aligned}$$

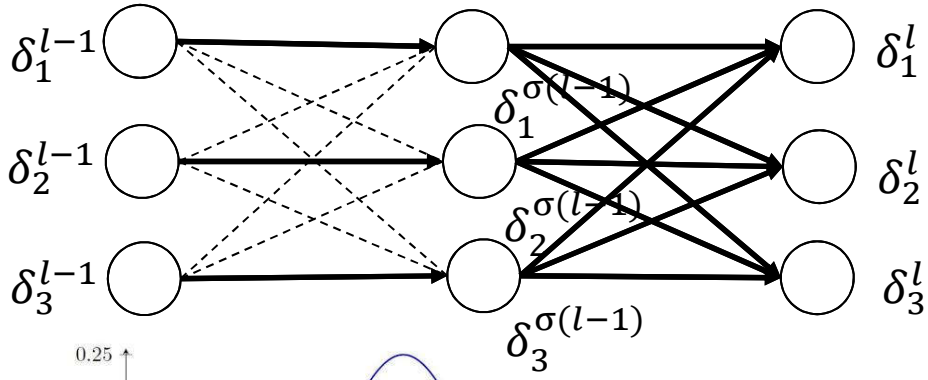
$$x_L = x_l + \sum_{i=l}^{L-1} \mathcal{F}(x_i, \mathcal{W}_i)$$

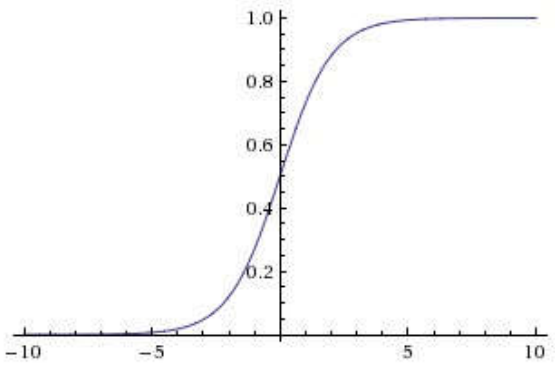
$$\frac{\partial C}{\partial X_l} = \frac{\partial C}{\partial X_L}\frac{\partial X_L}{\partial X_l} = \frac{\partial C}{\partial X_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=l}^{L-1}\mathcal{F}(x_i, \mathcal{W}_i)\right)$$

Sigmoid 激活函数问题

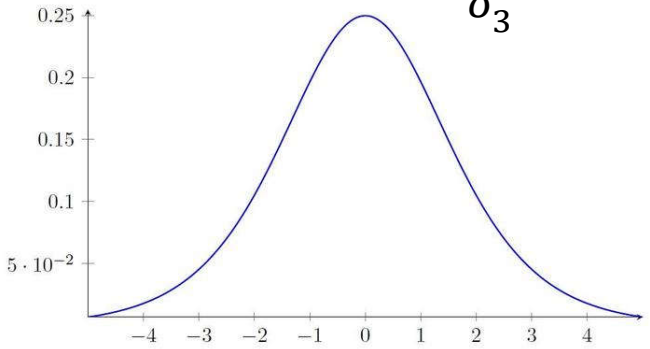
- 1. exp()计算复杂
- 2. 激活函数易饱和，梯度消失，无法继续学习：（后面权重初始化大也容易引起饱和）：输出$\sigma(z^{l-1})$为0，1的地方，导数$\sigma'(z^{l-1})$接近0。

$$\delta^{l-1} = \delta^{\sigma(l-1)} \odot \sigma'(z^{l-1})$$





Sigmoid函数曲线



Sigmoid导数曲线

Sigmoid 激活函数问题

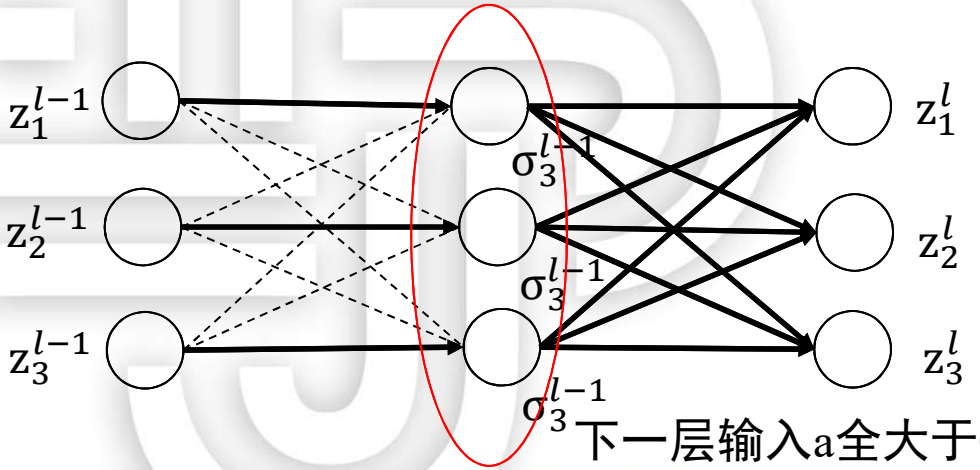
- 2. 输出(0~1)非0中心，$\sigma(z^{l-1})$ 总大于0：某节点输入全为正数，则权重w更新时呈锯齿形下降（批量算法缓解了这一问题，输入数据的归一化也缓解这个问题）

$$z_j^l = \sum_{k=1}^{K}(w_{jk}^{l-1} \cdot a_k^{l-1}) + b_j^{l-1},\ a_j^l = \sigma(z_j^l)$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l \cdot a_k^{l-1}\ \ (BP4)$$

$$\delta^{l-1} = \delta^{\sigma(l-1)} \odot \sigma'(z^{l-1})$$

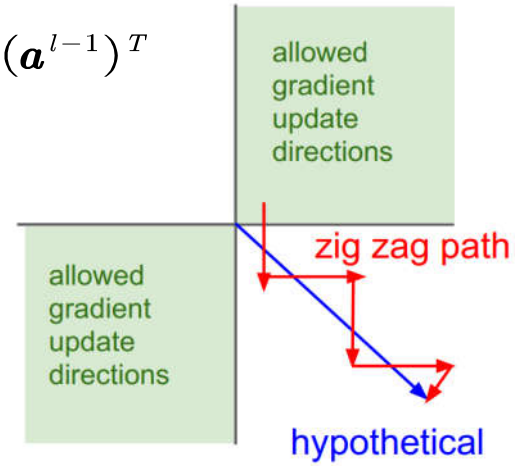
$$\frac{\partial C}{\partial w^l} = \boldsymbol{\delta}^l \cdot (\boldsymbol{a}^{l-1})^T$$



下一层输入a全大于0

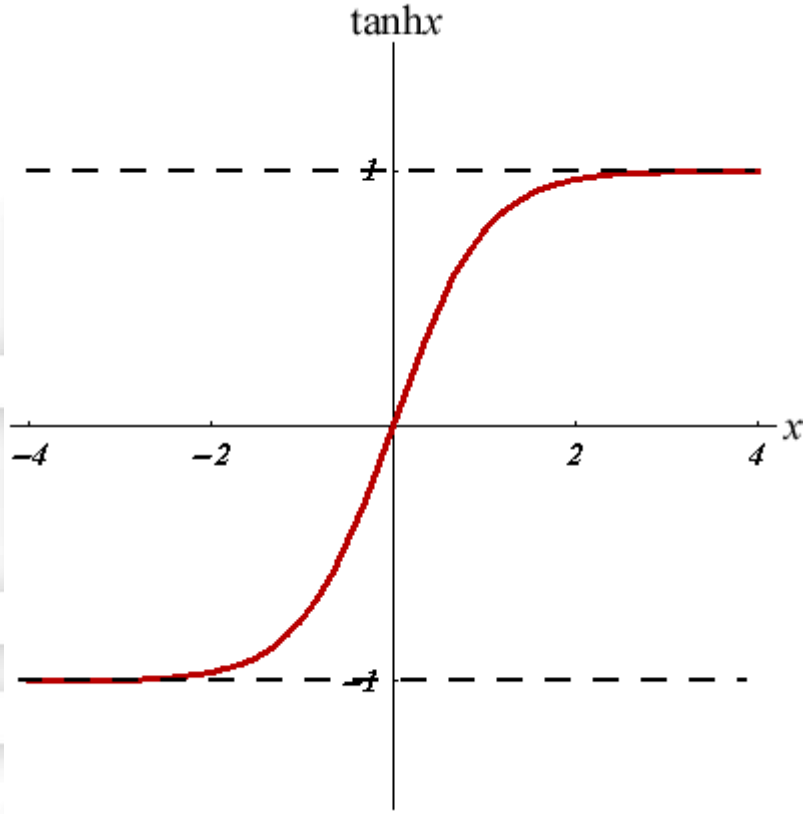
$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w_i}$$

$$= \frac{\partial C}{\partial a}\sigma(z)'\ x_i$$



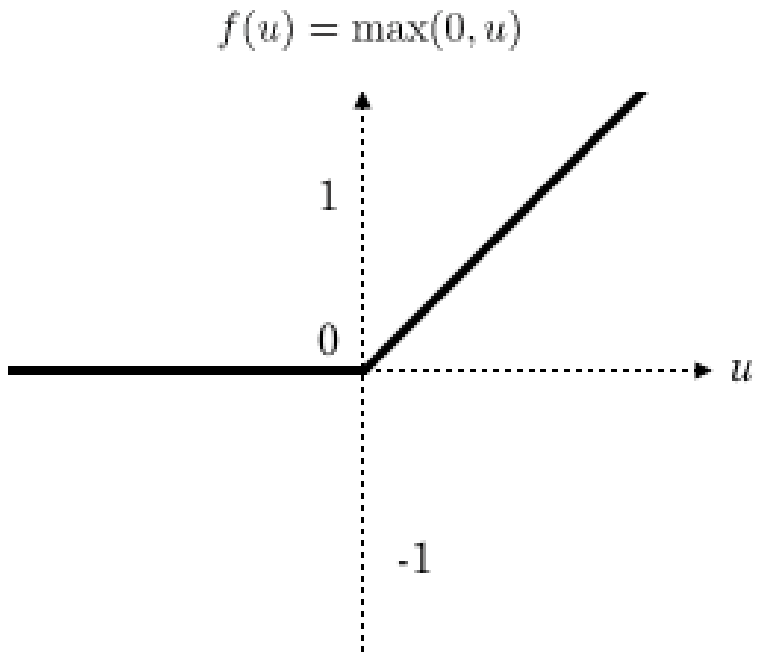
Sigmoid输入数据空间更新行为

Tanh(x)



- 输出 [-1,1]
- 是0中心化
- 依然存在激活易饱和的问题
- Tanh(x)计算也不容易

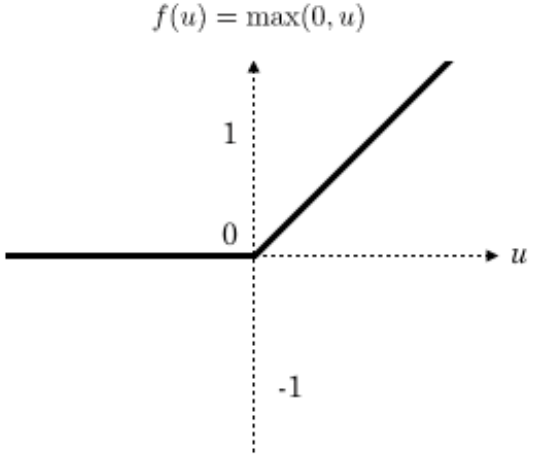
ReLU

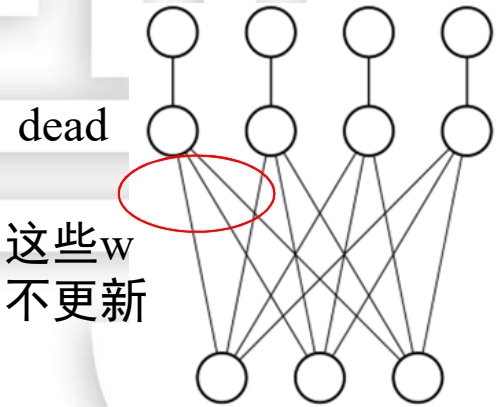


- 不会出现激活函数饱和问题
- 计算高效
- 相比sigmoid/tanh 收敛速度快

- 问题：依然是非0 中心输出，存在dying ReLU问题

Dying ReLU





$$z_n = \sum_{i=0}^{k} w_i a_i^n$$

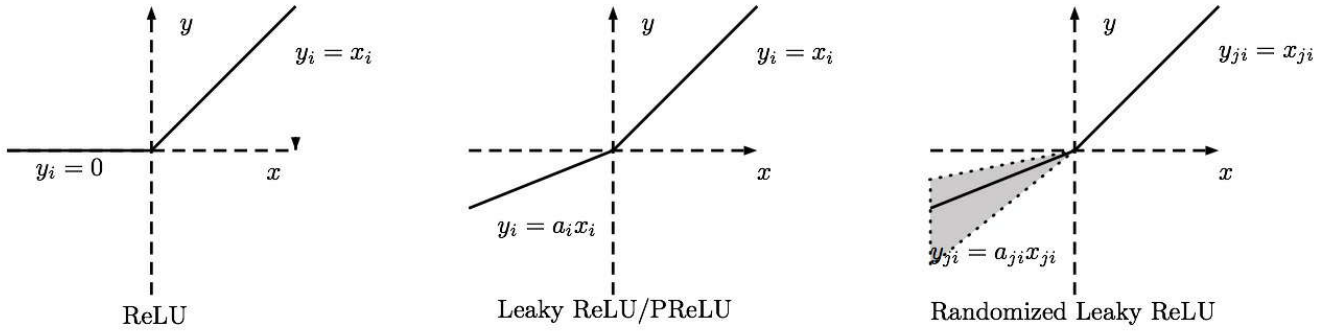
$$\frac{C}{z_n} = \delta_n = \begin{cases} 1 & z_n \geq 0 \\ 0 & z_n \leq 0 \end{cases}$$

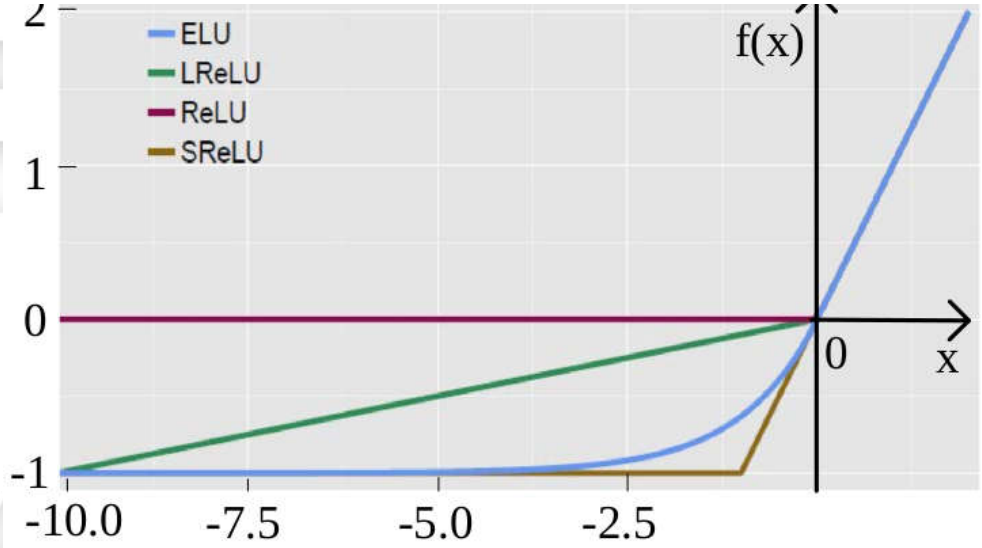
$$\frac{C}{w_j} = \frac{C}{z_n}\frac{z_n}{w_j} = \delta_n a_j^n = \begin{cases} a_j^n & z_n \geq 0 \\ 0 & z_n \leq 0 \end{cases}$$

- 激活函数输入z_n小于0，梯度也为0，z_n之前权重w不能更新
- 权重未初始化好，或经过一些调整，使得w变小，z_n一直小于0，w永远不更新，
- 总有些输入样本使得某些节点也一直输出是负数
- 卷积网也类似

其他激活函数

- 实际应用中，并没有绝对证明这些比ReLU有明显改进





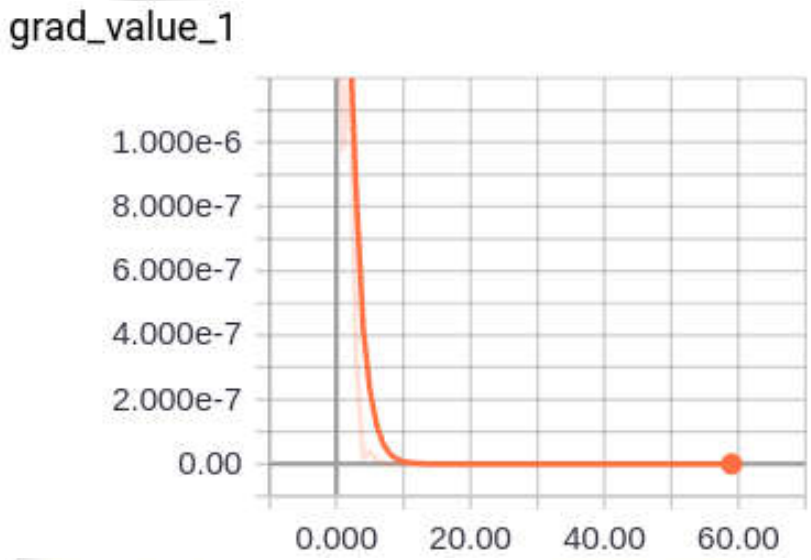
激活函数总结

- 1. 尽量使用ReLU，但注意学习率。
- 2. 可以尝试Leaky ReLU/Maxout/ELU等变体。
- 3. 可以尝试tanh，尤其在GAN中，有很多地方有好于ReLU的效果。
- 4. 尽量不用sigmoid

- 问题：为什么ReLU相对收敛快？

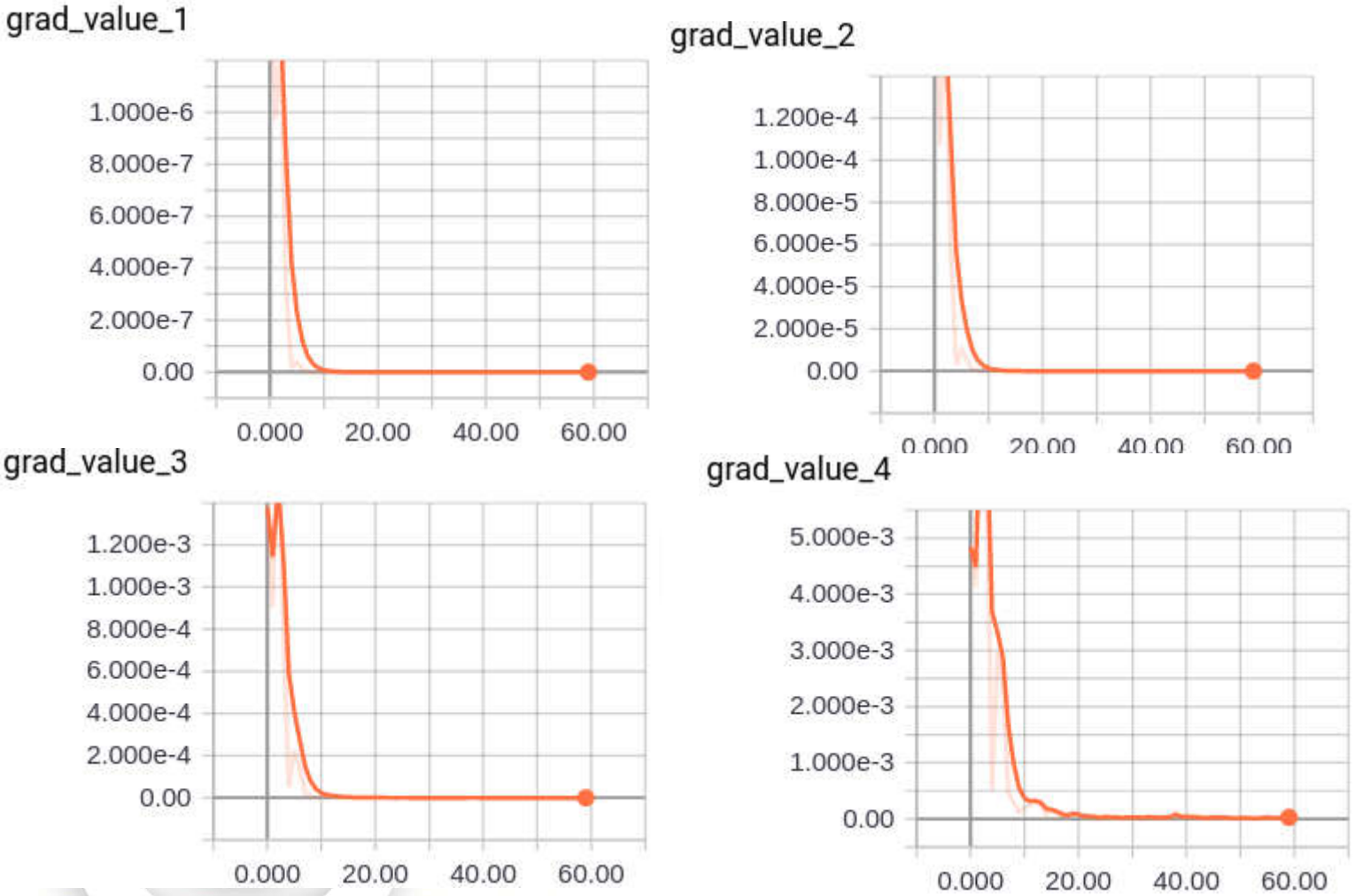
梯度下降挑战问题：梯度消失与梯度爆炸

- 梯度消失 Gradient Vanishing
  - 对前面的梯度消失例子的网络结构，进行训练迭代60次，并监测各隐藏层偏置上的梯度，会得到与理论分析相同的结果。如下图所示，从左到右，从上到下，依次是第1个到第4个隐藏层偏置bl上的梯度求模的值，曲线显示越靠前的层偏置向量b的模越小。



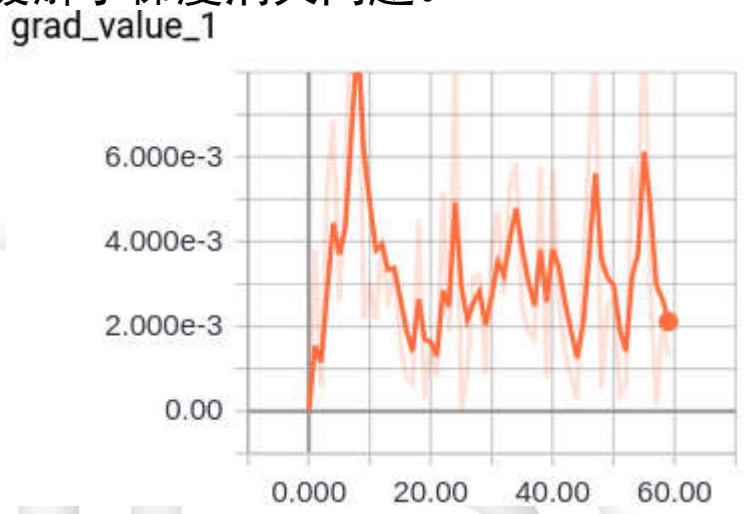
梯度下降挑战问题：梯度消失

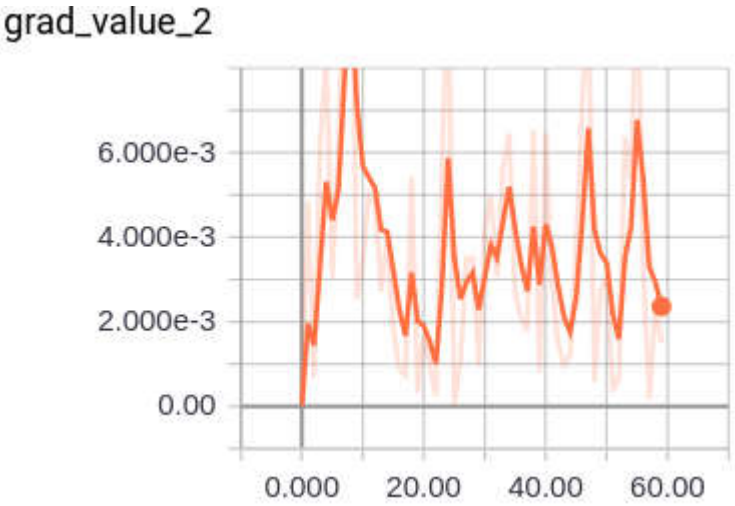
layer 1,2,3,4各层关于偏置b的梯度，使用sigmoid激活函数。

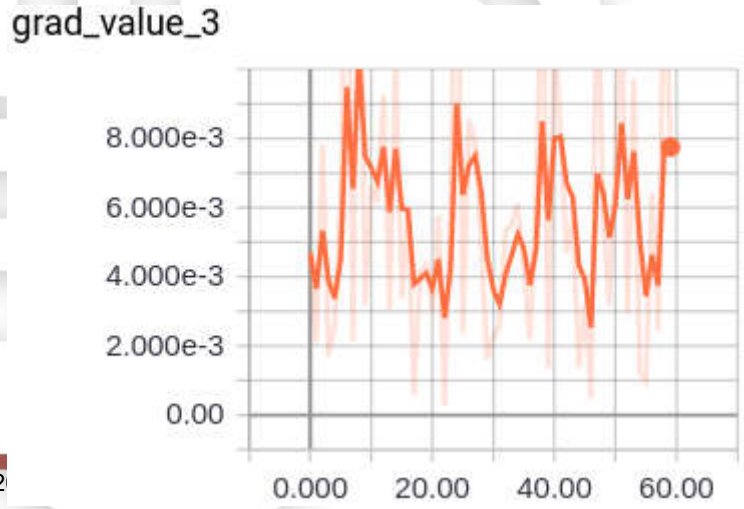


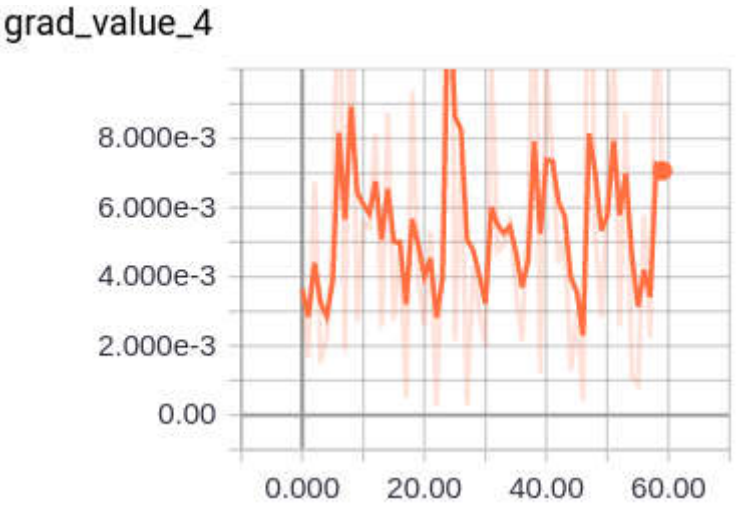
基于梯度的优化方法

layer 1,2,3,4各层关于偏置b的梯度，使用ReLU激活函数。
缓解了梯度消失问题。







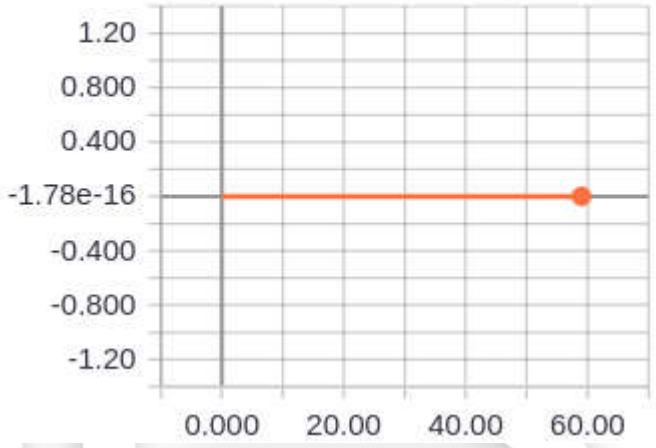


基于梯度的优化方法

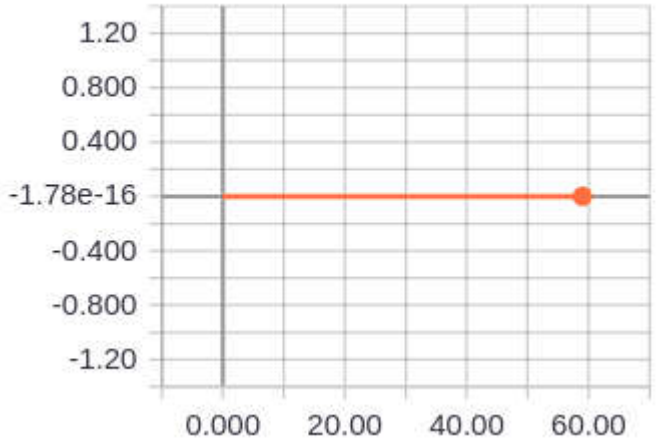
layer 1,2,3,4各层关于偏置b的梯度，使用ReLU激活函数。

ReLU依赖于初始化：较差的初始化导致前几层梯度不更新。

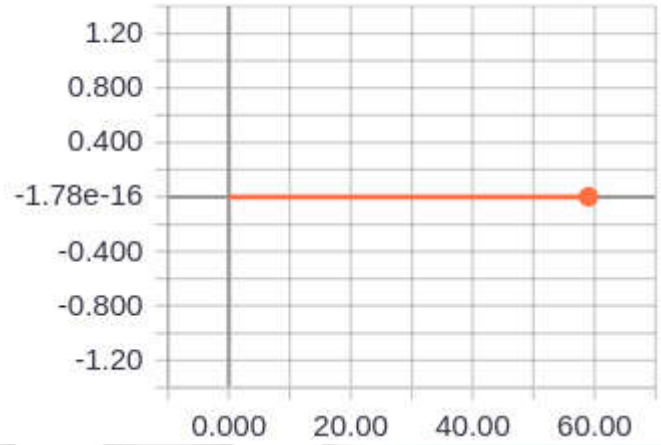
grad_value_1



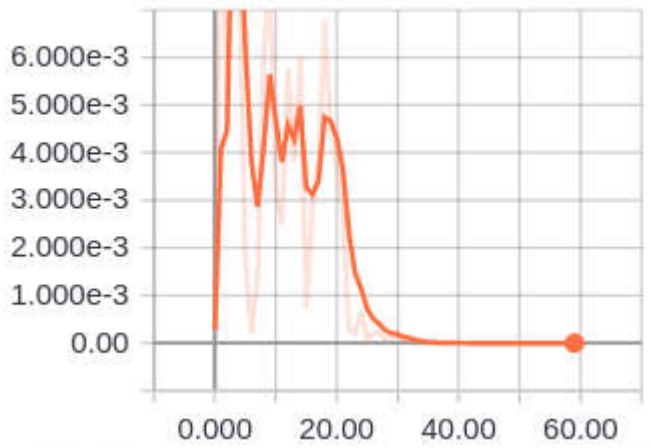
grad_value_2



grad_value_3



grad_value_4



基于梯度的优化方法

- 梯度消失 Gradient Vanishing
  - 识别minst字体多层神经网络，使用sigmoid激活函数，层数从1增加到4。

	隐层数量	每隐层神经元数	迭代次数	识别精度
1	隐层x1	100	30	95.25%
2	隐层x2	100	30	95.87%
3	隐层x3	100	30	96.3%
4	隐层x4	100	60	96.08%

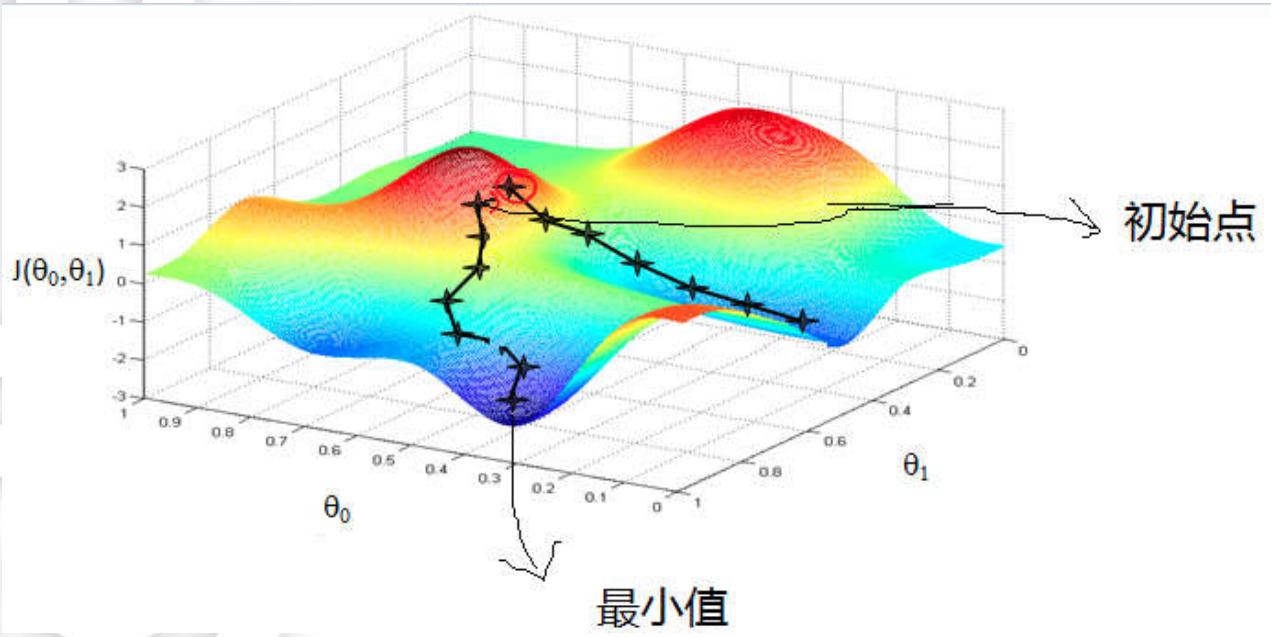
基于梯度的优化方法

- 梯度消失 Gradient Vanishing
  - 识别minst字体多层神经网络，使用ReLU激活函数，层数从1增加到4。收敛相对快，准确度提升。

	隐层数量	每隐层神经元数	迭代次数	识别精度
1	隐层x1	100	30	97.57%
2	隐层x2	100,100	30	97.92%
3	隐层x3	100,100,100	30	97.9%
4	隐层x4	100,100,100,100	60	97.81%
5	隐层x4	500,300,150,50	60	97.98%
6	隐层x4	2048,1024,512,256	60	98.07%

参数初始化

- 参数初始化：
- 1. 优化结果优劣（不只是最小值问题，梯度学习行为）
- 2. 训练是否可行（梯度消失、爆炸、训不动。。。）

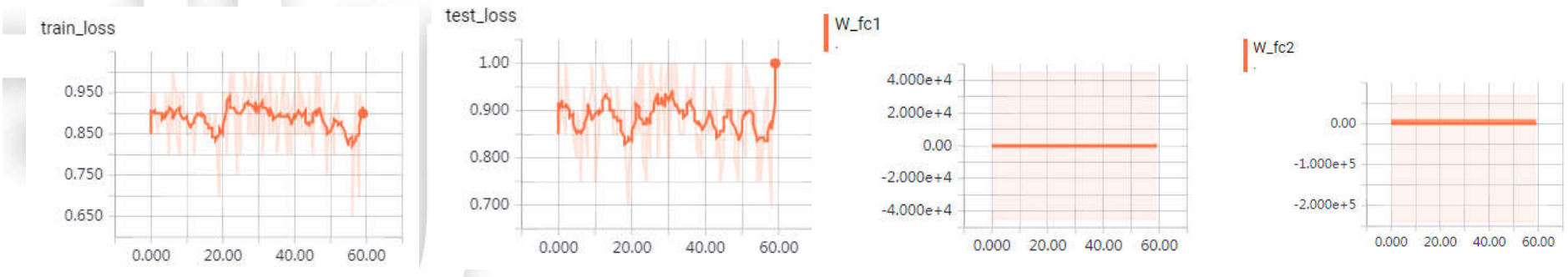


梯度优化中的参数选择

- 破坏对称性：
  - 随机初始化权重
  - 0初始化权重失败（回顾之前可视化中的例子）
- 权重大小：相对大的情况
  - 大权重破坏对称性更强+
  - 有益于缓解梯度消失+
  - 梯度爆炸等不稳定问题-
  - 激活函数易饱和-
- 不同层的缩放因子
- 偏置可以0初始化

参数初始化

- LeNet 训练 MNIST
- 案例1,送分题：全0 初始化LeNet 导致参数不更新。为什么？
- 案例2： 错误的层间参数数值范围比例：初始化最后全连接层参数均值100，方差10；其他层变量均值0， 方差0.1。导致无法训练：学习率太小，学不动；太大，爆炸，或不收敛。
- 学习率为0.1：



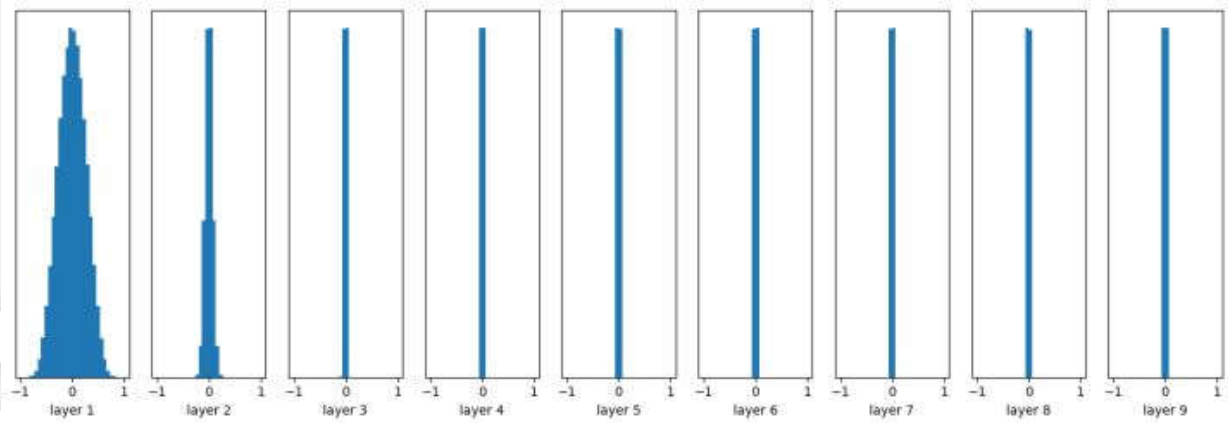
参数初始化

- 学习率为1：

- 学习率为100：
- 考察每层梯度，解释为什么

参数初始化

- 参数随机初始化的问题：
- 10层的神经网络，非线性变换为tanh，每一层的参数都是随机正态分布，均值为0，标准差为0.01
- 训练后，每一层输出值分布的直方图



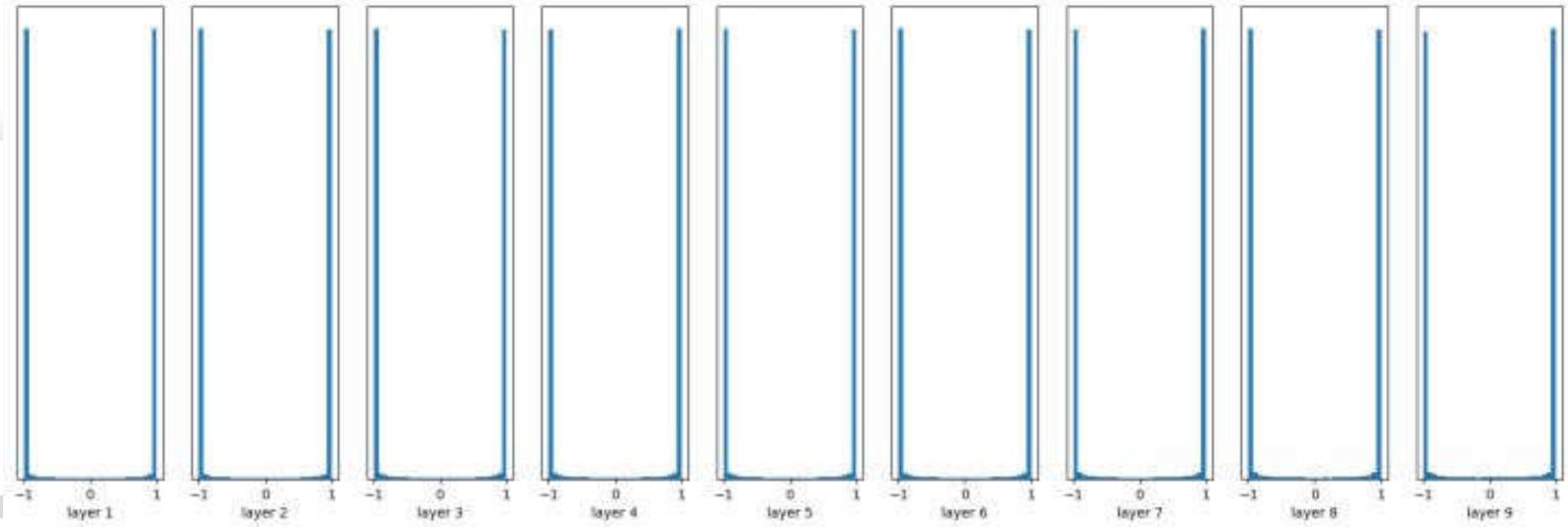
$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w_i}$$

$$= \frac{\partial C}{\partial a}\sigma(z)' x_i$$

- 输出集中在0附件，w难以更新（反向传播公式）

参数初始化

- 均值仍然为0，标准差现在变为1



- 输出集中在-1，1, tanh激活函数饱和，网络依然无法更新

参数初始化

- 问题出在哪里？
- 参数太大，饱和；太小，不激活：激活函数的特性（relu也存在相关问题）
- 如何配合激活函数特性初始化参数？
- Xavier initialization或者 Glorot initialization

Normal distribution with mean 0 and standard deviation $\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

Or a uniform distribution between -r and +r, with $r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$

- 基本思想：保持输入和输出的方差一致，避免所有输出值都趋向于0

参数初始化

- 针对不同激活函数：

Table 11-1. Initialization parameters for each type of activation function

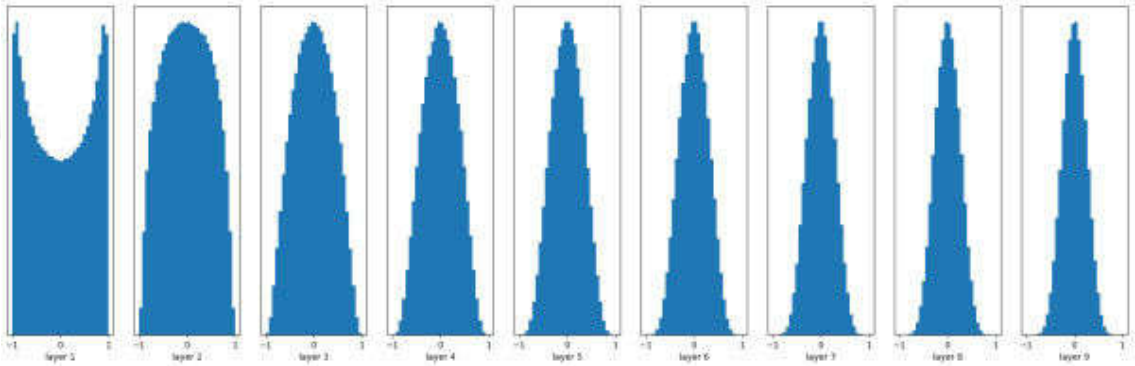
Activation function	Uniform distribution [–r, r]	Normal distribution
Logistic	$r = \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$	$\sigma = \sqrt{\frac{2}{n_{inputs} + n_{outputs}}}$
Hyperbolic tangent	$r = 4\sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$	$\sigma = 4\sqrt{\frac{2}{n_{inputs} + n_{outputs}}}$
ReLU (and its variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{inputs} + n_{outputs}}}$

- 主要区别是根号外的系数
- he_init = tf.contrib.layers.variance_scaling_initializer()
- hidden1 = fully_connected(X, n_hidden1,weights_initializer=he_init, scope="h1")

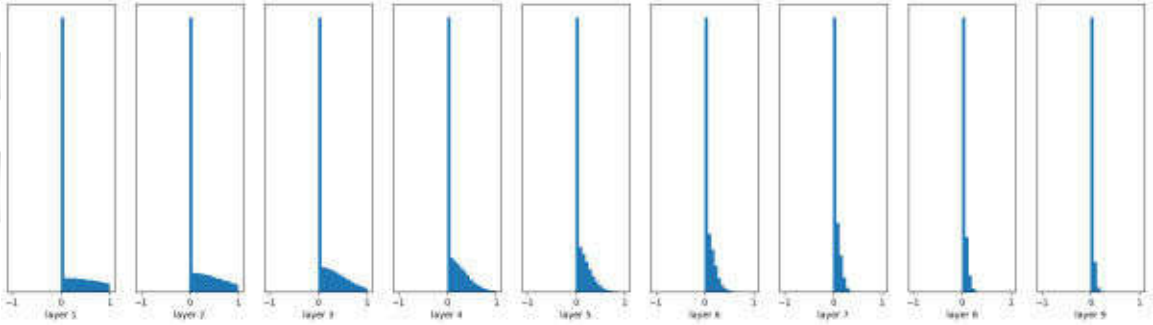
参数初始化

- ReLU实验效果：xavier初始化

```
W = tf.Variable(np.random.randn(node_in, node_out)) / np.sqrt(node_in)
```



```
W = tf.Variable(np.random.randn(node_in, node_out)) / np.sqrt(node_in)
......
fc = tf.nn.relu(fc)
```

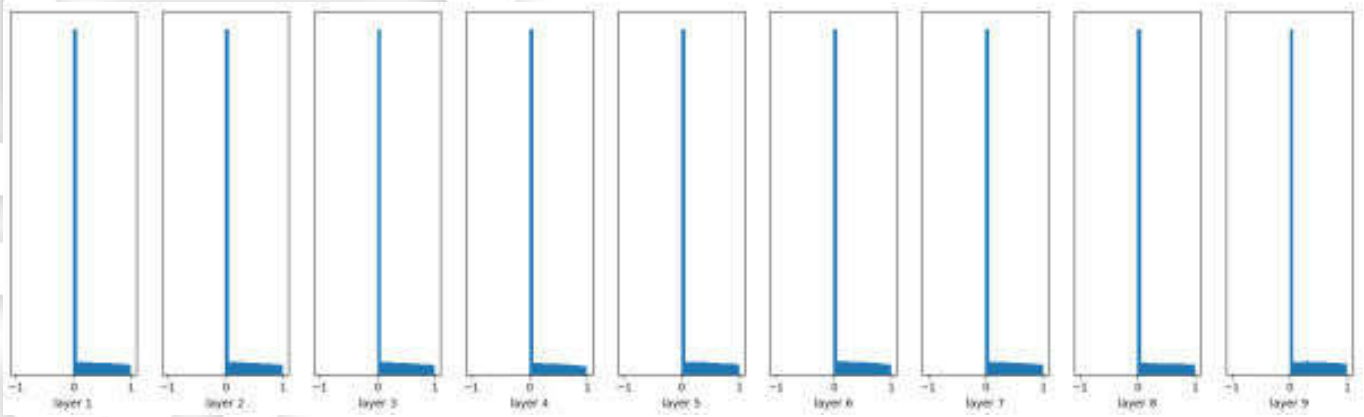


参数初始化

- **He initialization 基本思想：**ReLU网络中，假定每层一半神经元被激活，另一半为0；保持variance不变，需在Xavier的基础上再除以2

```
W = tf.Variable(np.random.randn(node_in,node_out)) / np.sqrt(node_in/2)
......
fc = tf.nn.relu(fc)
```

- 效果更好些

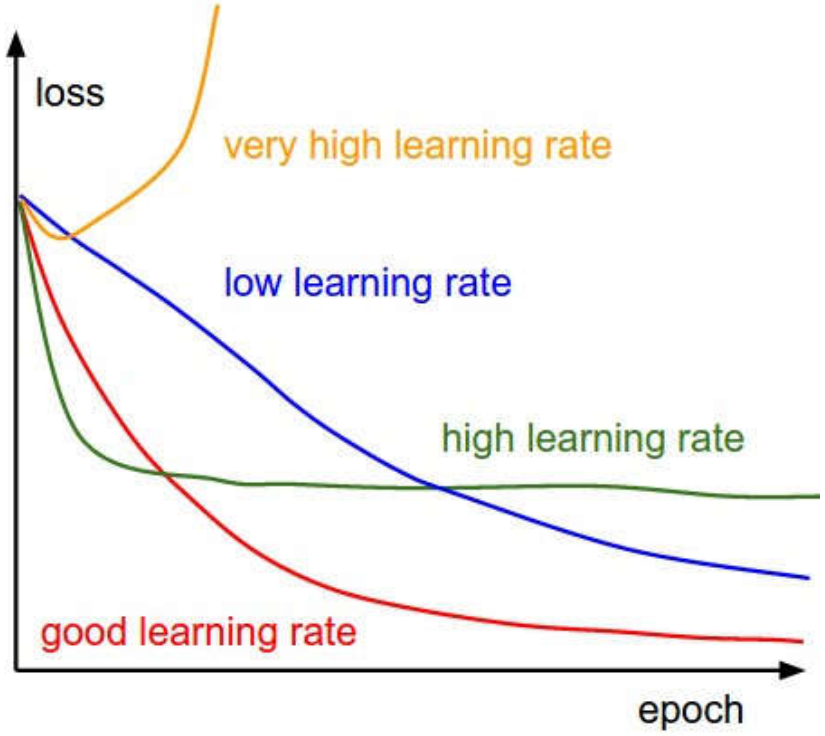


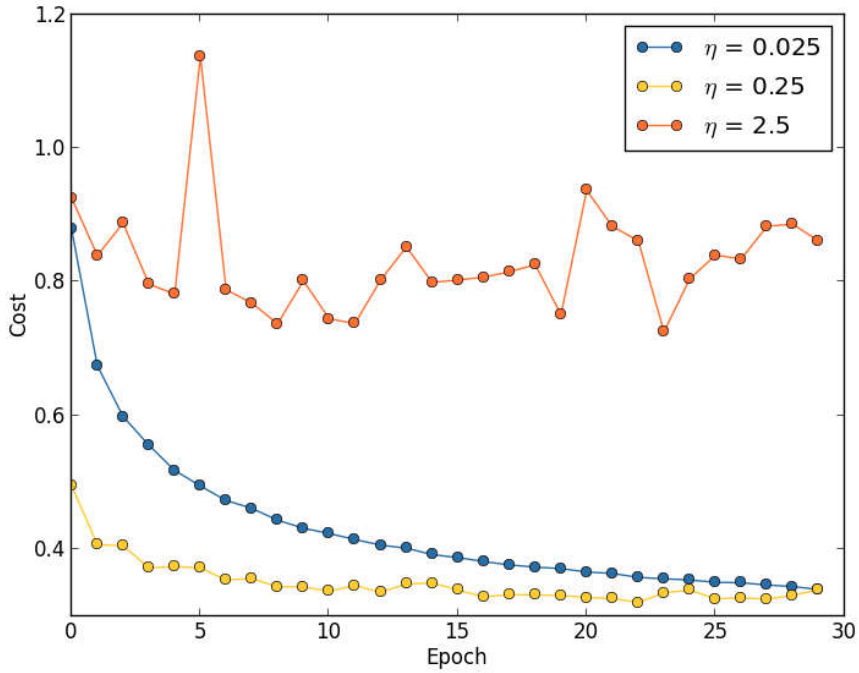
学习率

- 学习率选择
  - 过大:振动、不收敛；过小:收敛速度慢
  - 理想的学习率设计是：前期较大大学习率搜索，后期小学习率调优。以及对参数的个性化调整：优化频率高的参数小学习率调整，优化频率低的参数大学习率调整。
  - 初期选择：指数级改变学习率观察loss变化
  - 其他手段：
    - 自适应学习率算法，包括Nesterov accelerated gradient [7]，Adagrad [8], Adadelta [9], Adam[10]
    - 热启动；

学习率

- 学习率选择


loss
very high learning rate
low learning rate
high learning rate
good learning rate
epoch


1.2
1.0
0.8
0.6
0.4
Cost
0
5
10
15
20
25
30
Epoch
η = 0.025
η = 0.25
η = 2.5

学习率

- 学习率对模型训练的影响

lr = 1

lr = 0.3

lr = 0.001

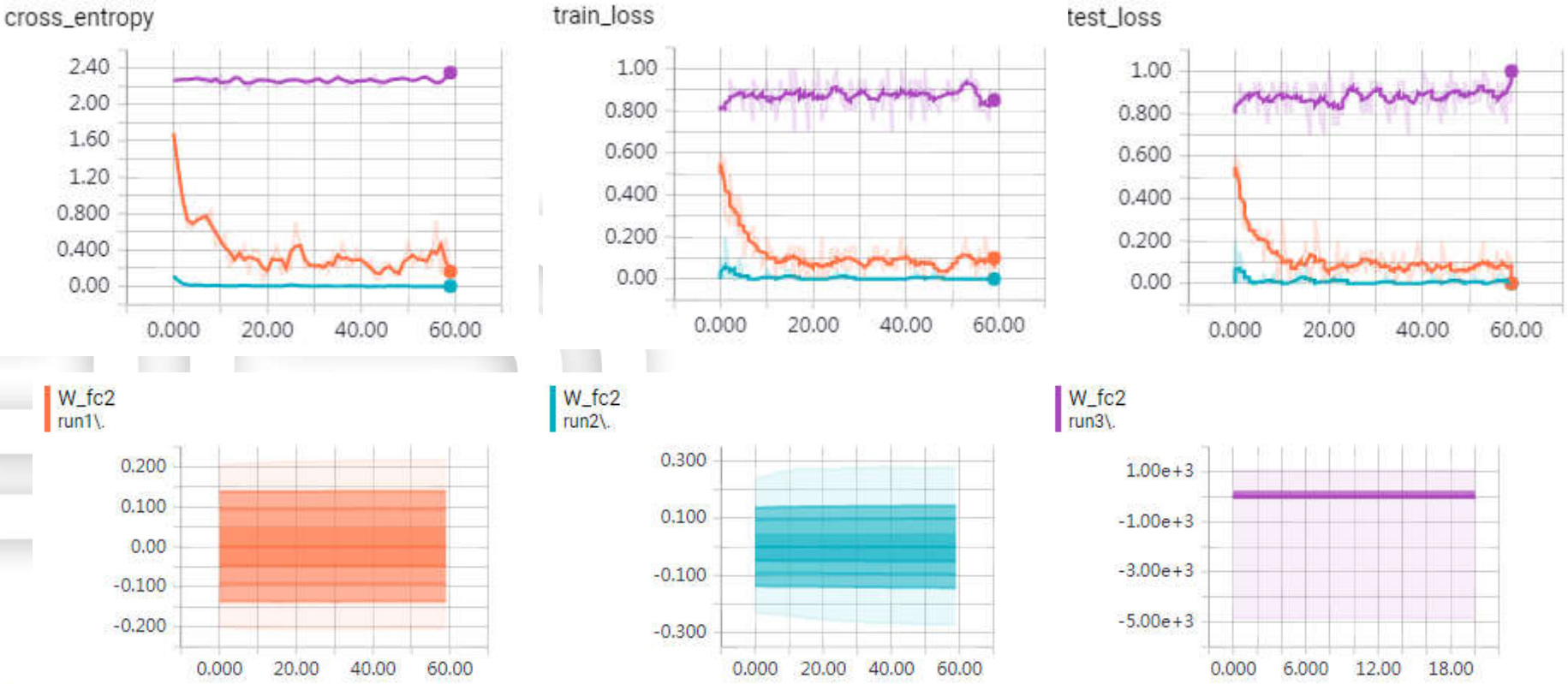
lr = 0.003

lr = 0.0001

lr = 0.00001

学习率

- LeNet 训练 MNIST
- 案例1：lr（ϵ步长）=0.0001;0.001;1 各是哪个？



学习率

- 学习率对模型训练的影响
  - 数据集是非线性可分的，增加了10个噪声点
  - 神经元使用ReLU激活函数，没有正则化项，batch size为10
  - 学习率设置过高，容易出现震荡或者陷入局部最小点的风险。

学习率	是否震荡	测试集损失（最佳）	训练集损失（最佳）	出现过拟合
1	是	0.061（或者难以收敛）	0.046	否
0.3	是	0.042	0.020	2000 Epoch
0.003	否	0.030	0.016	4000 Epoch
0.001	否	0.030	0.017	8000 Epoch
0.0001	否	0.033	0.031	否
0.00001	否	0.499（不收敛）	0.499	否

Mini Batch方差与learning rate 关系

- Batch size越大：
- 梯度的均值越接近期望，方向更准
- 梯度的方差期望越小，估计更稳
- 学习率增大会增大梯度估计方差
- 另一方面，如果batch size比较大，可以用比较大的lr而同时可以保持一定的方差，这样加快学习

$$\mathcal{V}ar(x) = \mathbb{E}\left[(x - \mathbb{E}(x))^2\right]$$

$$\mathcal{V}ar(g) = \mathcal{V}ar\left(\frac{1}{m}\sum_{i=1}^{m} g(x_i, y_i)\right)$$

$$= \frac{1}{m^2}\sum_{i=1}^{m} \mathcal{V}ar(g(x_i, y_i))$$

$$= \frac{1}{m}\mathcal{V}ar(g(x_1, y_1))$$

代入lr作为梯度比例，类似推导有：

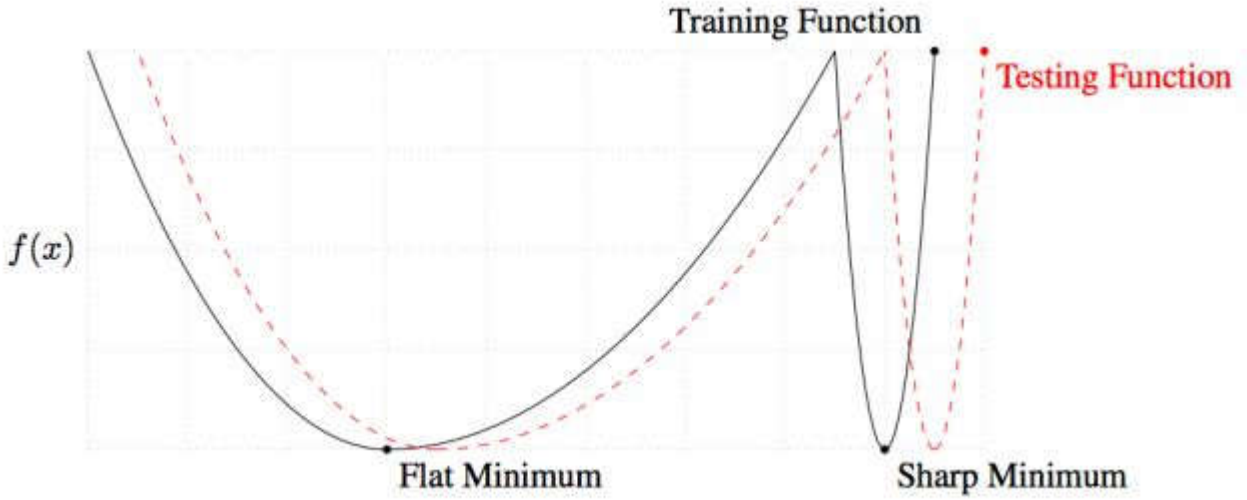
$$\mathcal{V}ar(lr*g) = \frac{lr^2}{m}\mathcal{V}ar(g(x_1, y_1))$$

Large-batch魔咒

- Large batch会有更准的梯度，方差越小，应该有更高的准确性
- 然而在深度网络中，large batch通常会降低准确率！
- ICLR 2017: ON LARGE-BATCH TRAINING FOR DEEP LEARNING: GENERALIZATION GAP AND SHARP MINIMA
- (i) LB methods over-fit the model;
- (ii) LB methods are attracted to saddle points;
- (iii) LB methods lack the explorative properties of SB methods and tend to zoom-in on the minimizer closest to the initial point;
- (iv) SB and LB methods converge to qualitatively different minimizers with differing generalization properties.

Large-batch魔咒

- LB陷入的是比较sharp的极值点，SB陷入的是比较flat的极值点，泛化性能更好



Large-batch魔咒

- **Facebook: Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour**
- LB之所以不work，不是那篇论文提到的泛化能力的问题，而主要是一个optimization issue
- 什么问题？类似我们之前的分析，学习率和batch size的关系
- Linear Scaling Learning Rate：简单点：batch size翻多少倍，learning rate就翻多少倍
- LB方法，开始用大的learning rate，效果差；只要开始把LR设小，之后逐步提高LR到正常大小，LB能到跟SB几乎一样的training curve、基本相同的准确度

Large-batch魔咒

- linear scale rule
- baeline(batch size B), large batch(batch size kB)更新公式:

$$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_{t+j}). \qquad (3)$$

$$\hat{w}_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_t). \qquad (4)$$

- （4）：过一步 相当于（3）：过k 步
- 所以（4）的学习率应该是（3）的k倍
- 在weight变化较快的时候，W(t+j) ~ W(t)不满足
- 故在开始变化很快的时候，一般不是k倍，后面逐渐放大
- lr也不能无限放大，batchsize变大后，合适的lr范围在变小

Batch size

- batch size对模型训练的影响
  - 小batch训练，受噪音影响大，不容易收敛，学习慢；泛化误差较小；
  - 大batch训练，梯度估计更准确，训练震荡越小，收敛速度快，但是容易陷入局部最小（非凸函数优化）；容易过拟合，large batch准确率会低；
  - 在实践过程中，在GPU显存容许的情况下，一般采用较大的batch size。注意和learning rate的配合

Batch size

- batch size对模型训练的影响
  - 数据集是非线性可分的，增加了10个噪声点；
  - 神经元使用ReLU激活函数，学习率0.003；
  - L2正则项，正则项系数0.003

batch size	测试集损失（最佳）	训练集损失（最佳）	拟合情况
1	0.029	0.022	拟合
2	0.031	0.022	拟合
4	0.031	0.020	拟合
8	0.030	0.020	拟合
16	0.032	0.029	拟合
30	0.035	0.035	拟合

Batch size

- batch size对模型训练的影响
  - 拓展资料：
    - [12]On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima,ICLR 2017 讨论大batch size的泛化误差不好的原因
    - [13]Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour 提出学习率的线性缩放和预热策略，使用超大batch size 1小时训练ImageNet
    - [14]Don't Decay the Learning Rate, Increase the Batch Size 讨论学习率衰减（Learning rate decay）和batch size的关系，提出不用衰减学习率，只要增大 Batch Size 就可以达到同样的精度。

梯度下降挑战问题：病态

- 考察代价函数降低效果， 令$g=\nabla_\theta J(\theta)$，代价函数泰勒展开：

$$J(\theta_{n+1}) \approx J(\theta_n) + (\theta_{n+1}-\theta_n)^T g + \frac{1}{2}(\theta_{n+1}-\theta_n)^T H(\theta_{n+1}-\theta_n)$$

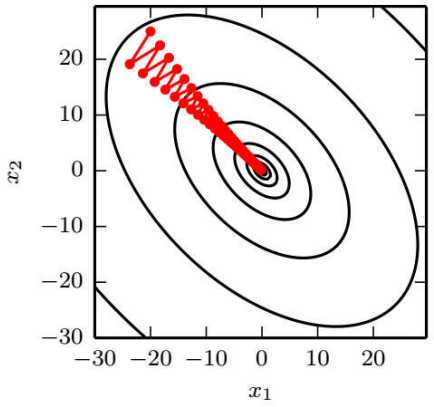
- H为hessian矩阵，由于 $\theta_{n+1}=\theta_n-\epsilon g$

$$J(\theta_{n+1}) \approx J(\theta_n) - \epsilon g^T g + \frac{1}{2} g^T H g$$

- g^THg 为0或负数时，代价函数永远下降
- g^THg为正数时，不合适的学习率ϵ有可能也会使代价函数上升。

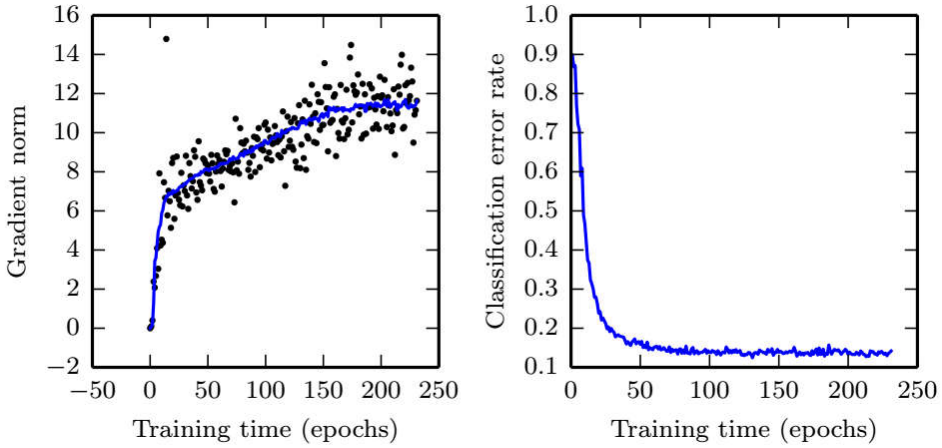
梯度下降挑战问题：病态

- Hessian矩阵反应了代价函数的曲面特征，在曲率变化大的点， Hessian矩阵往往呈现病态（ill-conditioning）
- Hessian矩阵病态标志着其条件数（最大特征值和最小特征值的比，也就意味着最大曲率和最小曲率的比）太大
- 病态影响：
  - 难以把握导数长期为负的方向，时间浪费在反复震荡中
  - 步长难以控制：要小，以免冲过，造成代价上升又要大，否则迭代过程进展缓慢
  - 训练可能会“卡”在一些地方，很小的步长也可能增加代价函数（尤其后期）



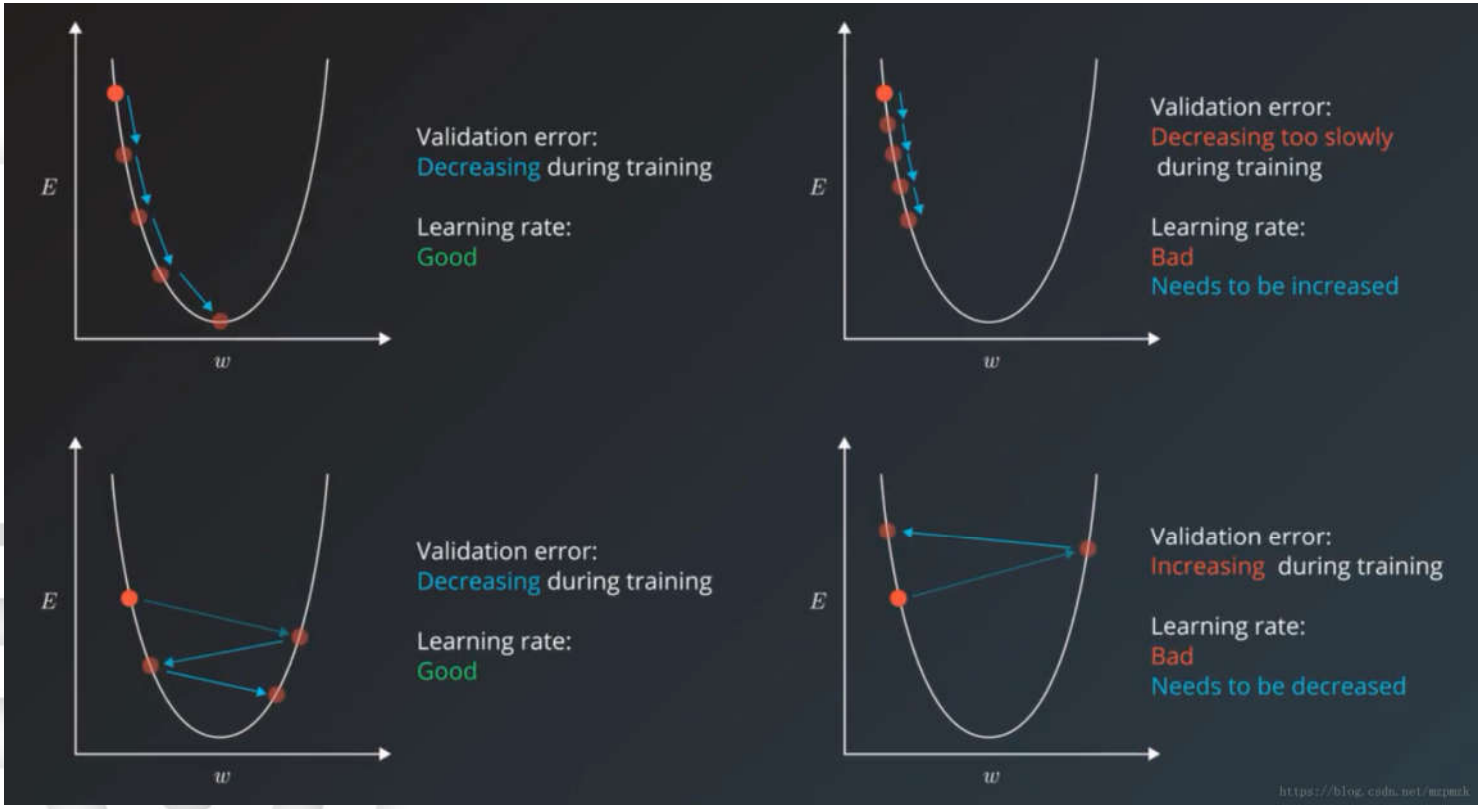
梯度下降挑战问题：病态

- 很多情况中，梯度范数不会在训练中显著减小（深度网络中，很少有极小值点，大多是鞍点，梯度下降通常不会到达任何临界点）
- $g^T Hg$会超过一个量级，回想：$J(\theta_{n+1}) \approx J(\theta_n) - \epsilon g^T g + \frac{1}{2} g^T Hg$
- 此时，尽管梯度很强，学习会非常缓慢（如果减少学习率去弥补H的强曲率，则也同时减慢了学习）



学习率衰减策略：learning rate decay

- 开始大，快速收敛，后期小，防止震荡不收敛



随机梯度下降算法（SGD）

- 学习速率,ϵ如何设置:
- 如果要保证SGD收敛,应该满足如下两个要求:

$$\sum_{k=1}^{\infty} \epsilon_k = \infty, \qquad \sum_{k=1}^{\infty} \epsilon_k^2 < \infty.$$

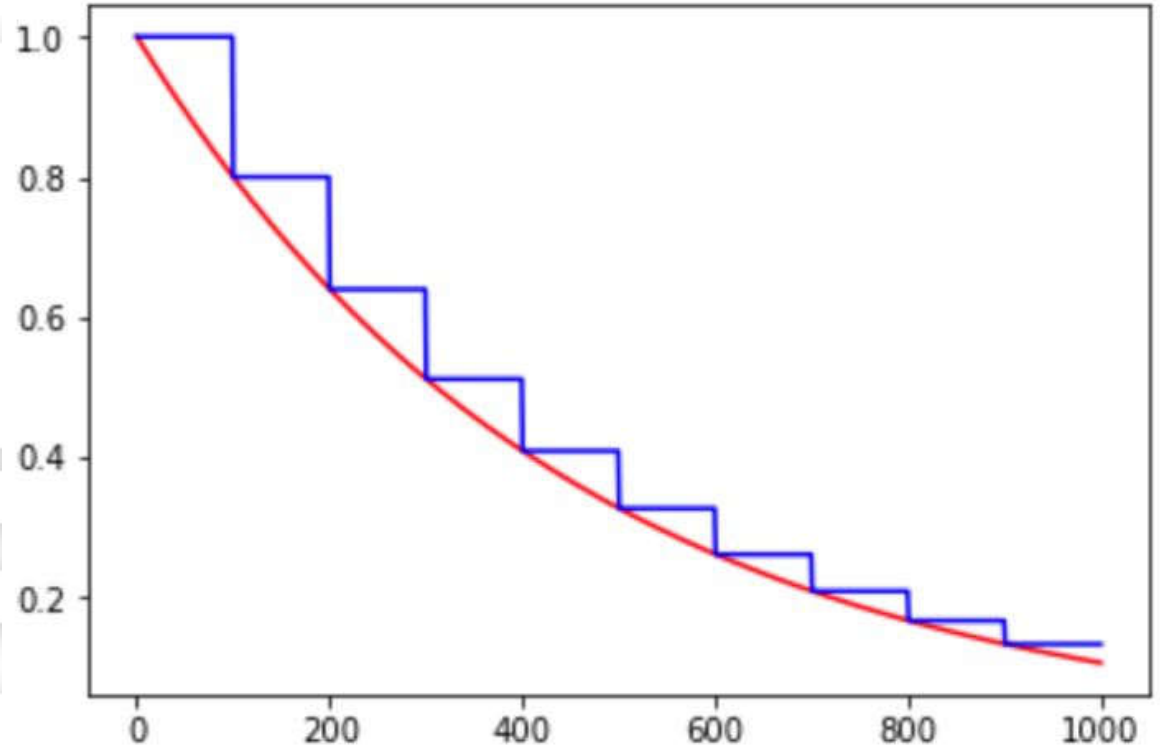
- 在实际操作中,一般进行线性衰减:（开始大，后面小）

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau$$

- $\epsilon 0$是初始学习率, $\epsilon\tau$是最后一次迭代的学习率. τ代表迭代次数.一般来说,$\epsilon\tau$ 设为$\epsilon 0$的1%比较合适.
- 初始学习率$\epsilon 0$，和迭代次数τ如何设置？

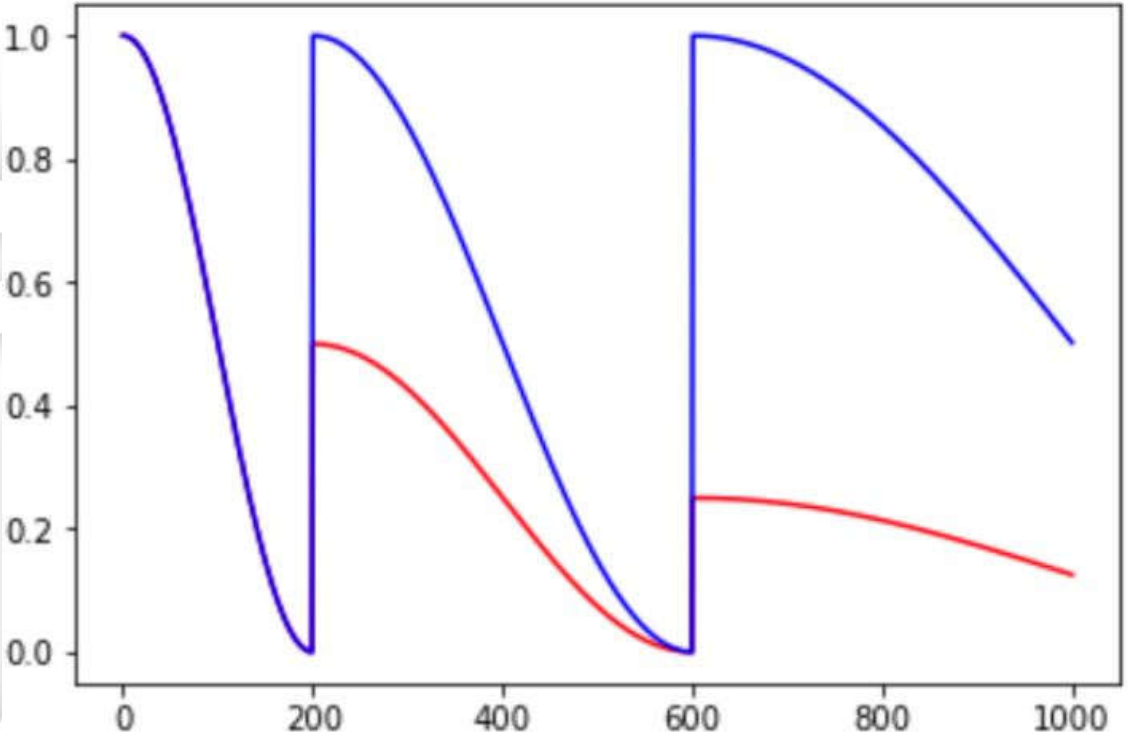
exponential_decay

- 学习率随一定步长阶段指数减少，最常用
- [tensorflow/tensorflow/python/training/learning_rate_decay.py]



Cycle learning rate

- 防止网络后期lr十分小导致一直在某个局部最小值中振荡，突然调大lr可以跳出注定不会继续增长的区域探索其他区域



正则项系数

- 正则项系数选择
  - 正则化技术令参数数量多于输入数据量的网络避免产生过拟合现象。
  - 常用的正则化方法：数据增强、L1 正则化、L2 正则化（权重衰减）、Dropout、随机池化和提前停止（Early stopping）等。
  - L1正则化产生更稀疏（sparse）的解，常用于**特征选择**；

  机器学习中最常用的正则化方法是对权重施加**L2范数约束**。[深度学习 page. 146]

  - Caffe或者Tensorflow中的权值衰减系数weight decay就是L2正则项的系数。

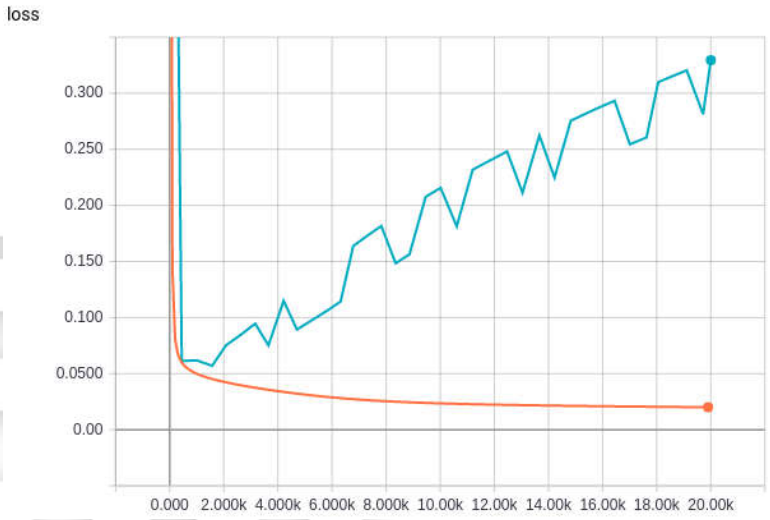
正则项系数

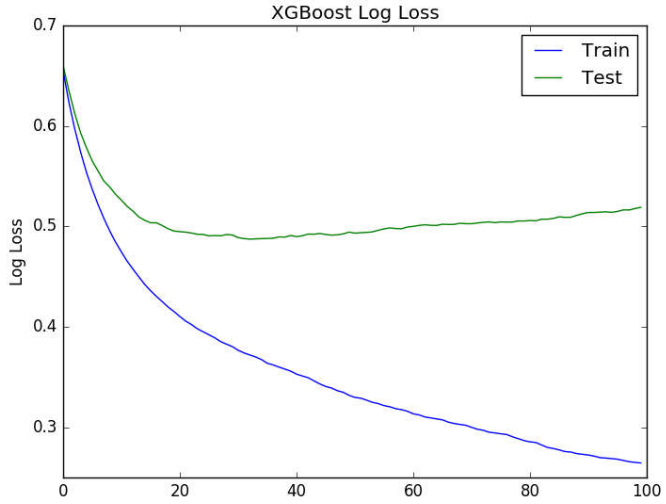
- 正则项系数对模型训练的影响
  - 数据集是非线性可分的，增加了10个噪声点；
  - 神经元使用ReLU激活函数，batch size为10；
  - 学习率0.003，L2正则项，不同正则项系数对比

正则项系数	测试集损失（最佳）	训练集损失（最佳）	拟合情况
0	0.031	0.015	过拟合
0.001	0.032	0.021	过拟合
0.003	0.031	0.026	拟合
0.01	0.031	0.033	拟合
0.03	0.044	0.046	欠拟合
0.1	0.499	0.499	不收敛

迭代次数

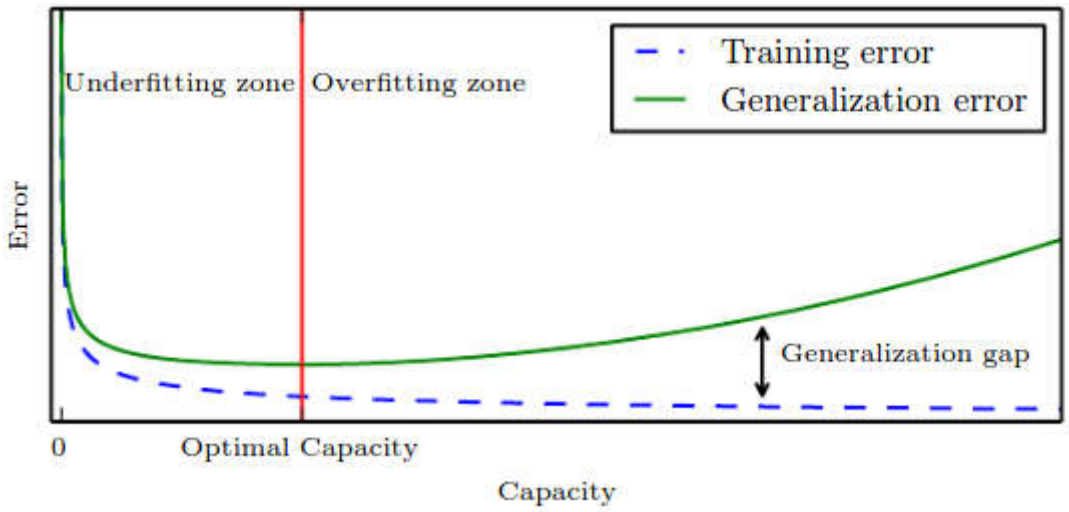
- 停止条件：
- 1. test loss 与 train loss 均缓慢下降并趋于平稳
- 2. train loss 虽然下降，但test loss 已经开始上升
- 3. train loss 与 test loss 一直上升





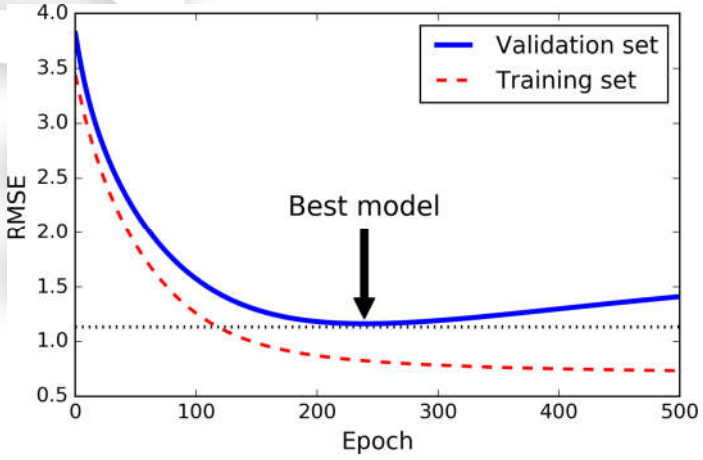
迭代次数

- 过拟合现象
  - 如图所示[深度学习 P. 73 模型容量与误差之间的典型关系]
    - 训练误差和泛化误差之间差异的上界随着模型容量增长而增长，但随着训练样本增多而下降（Vapnik and Chervonenkis,1971）
    - 当模型容量上升时，训练误差会下降，直到其渐进最小可能误差；
    - 通常，泛化误差是一个关于模型容量的U形曲线函数。



迭代次数

- 提前终止 （early stop）
  - 一种常用的正则化形式
  - 当验证集上的误差在事先指定循环次数内没有进一步改善时，停止训练算法
- 将训练步数视作超一个超参数，提前终止通过控制拟合训练集的步数来控制模型的有效容量



迭代次数

- Early stopping
  - 提前终止算法

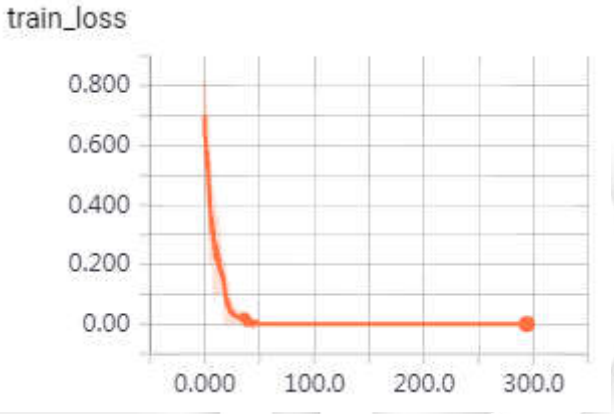
[深度学习 P.153]

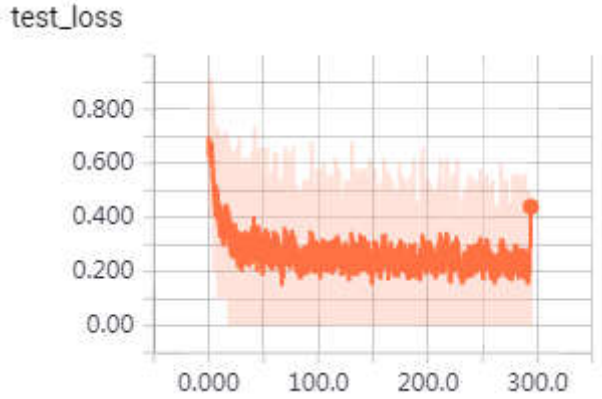
Algorithm 7.1 The early stopping meta-algorithm for determining the best amount of time to train. This meta-algorithm is a general strategy that works well with a variety of training algorithms and ways of quantifying error on the validation set.

Let n be the number of steps between evaluations.
Let p be the "patience," the number of times to observe worsening validation set error before giving up.
Let $\boldsymbol{\theta}_o$ be the initial parameters.
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_o$
$i \leftarrow 0$
$j \leftarrow 0$
$v \leftarrow \infty$
$\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
$i^* \leftarrow i$
while $j < p$ **do**
  Update $\boldsymbol{\theta}$ by running the training algorithm for n steps.
  $i \leftarrow i + n$
  $v' \leftarrow \text{ValidationSetError}(\boldsymbol{\theta})$
  if $v' < v$ **then**
    $j \leftarrow 0$
    $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}$
    $i^* \leftarrow i$
    $v \leftarrow v'$
  else
    $j \leftarrow j + 1$
  end if
end while
Best parameters are $\boldsymbol{\theta}^*$, best number of training steps is i^*.

迭代次数

- LeNet 训练 MNIST
- 案例1：总共只随机取1000个训练样本，缩小训练集的规模，人为制造过拟合

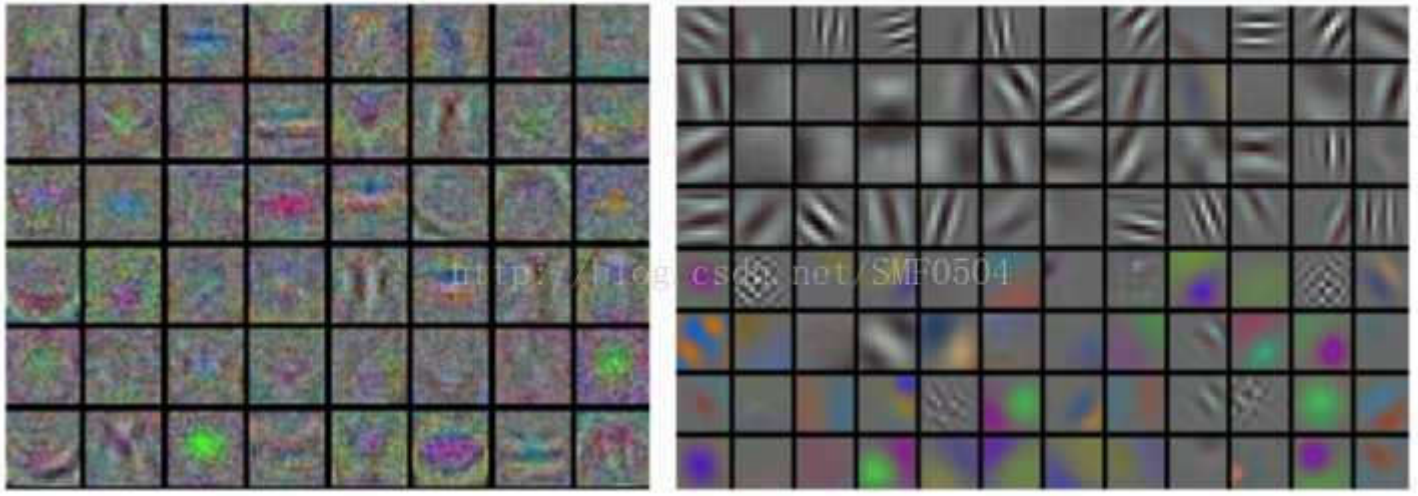




- 问题：1. 为什么说现在的训练过拟合了？
- 2. 为什么没有发生U型test上升？

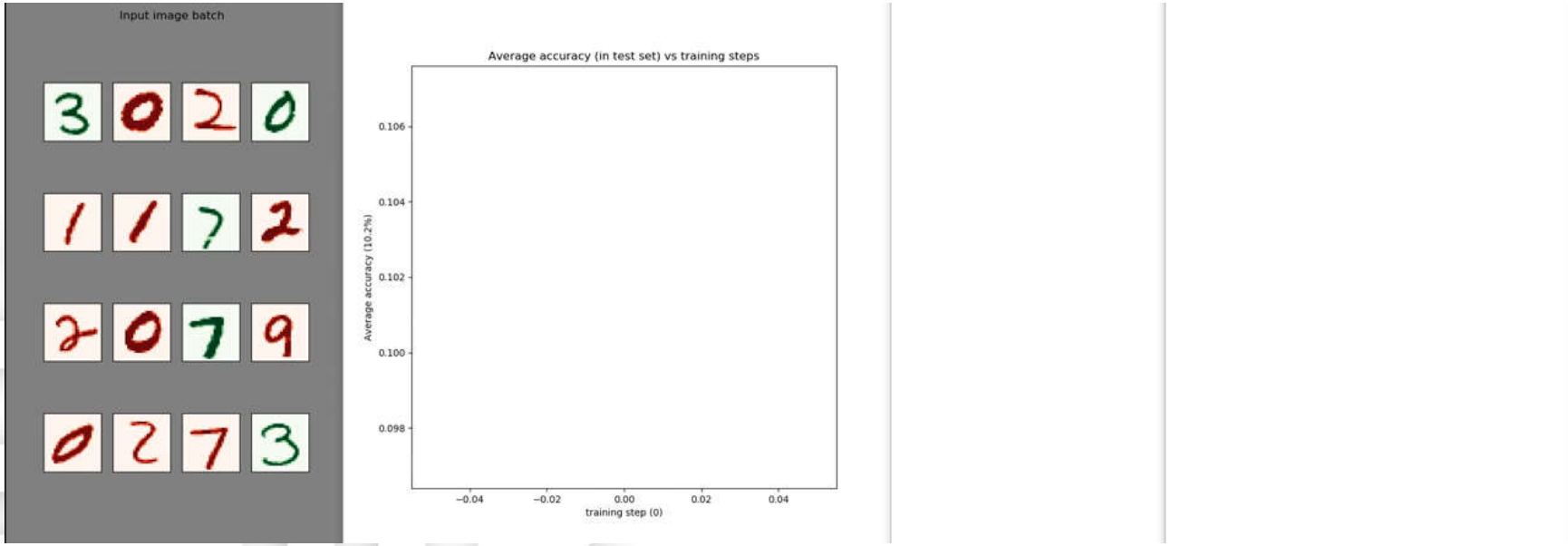
可视化评估

- 训练比较好的网络，有比较有特点且smooth的特征，没有训练好的参数看起来比较随机
- 其他之前介绍的可视化梯度、参数分布等的方法



可视化评估

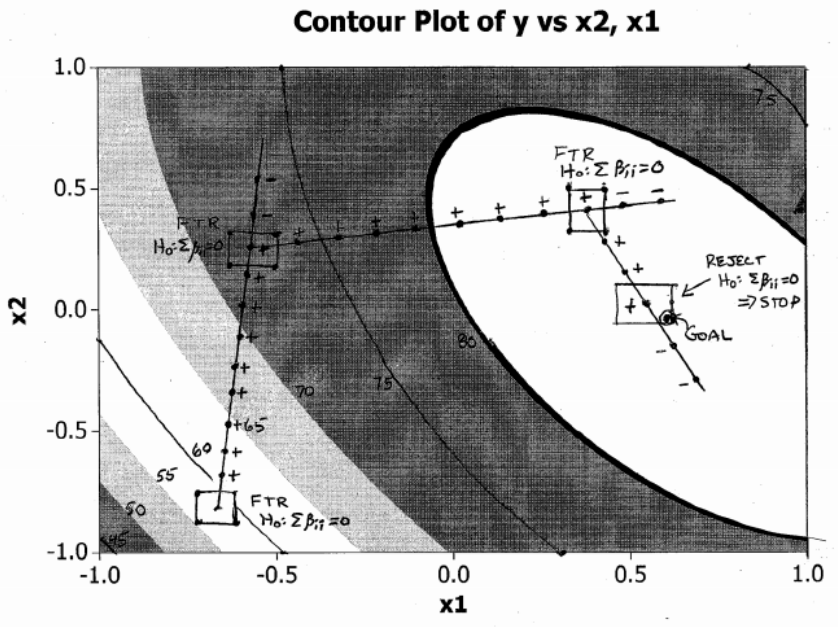
- 也有问题：现在都流行用小核了，smooth怎么看的出来



最速梯度下降法

- 梯度下降： $\theta_{n+1} = \theta_n - \epsilon \nabla_\theta J(\theta)$
- 方向已按贪心选下降最多的方向，步长ϵ如何选？
- 进一步贪心：每步都走到此方向最低的点
- 最速下降：每次迭代沿梯度方向搜索，选取该方向代价函数下降最大的点：

$$\epsilon_n = \underset{\epsilon}{arg\min} J(\theta_n - \epsilon \nabla_\theta J(\theta_n))$$

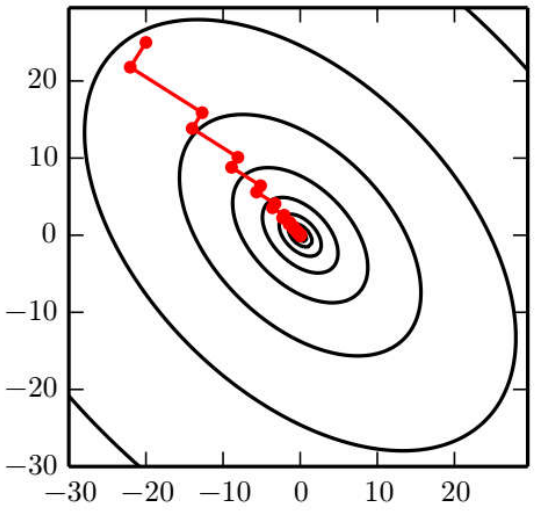


最速梯度下降法

- 由于$-\nabla_\theta J(\theta)$ 方向ϵ_n最优可知其导数为0：

$$\frac{\partial J(\theta_n - \epsilon \nabla_\theta J(\theta_n))}{\partial \epsilon} = \nabla_\theta J(\theta_n - \epsilon \nabla_\theta J(\theta_n))(0 - \nabla_\theta J(\theta_n)) = 0$$

- 即 $\nabla_\theta J(\theta_n - \epsilon_n \nabla_\theta J(\theta_n)) \nabla_\theta J(\theta_n) = \nabla_\theta J(\theta_{n+1}) \nabla_\theta J(\theta_n) = 0$
- 说明最速梯度下降每次迭代的方向互相正交
- 最速下降并不“最速”
- 锯齿梯度更新震荡
- 其在接近最优点尤其耗时



基于动量的算法

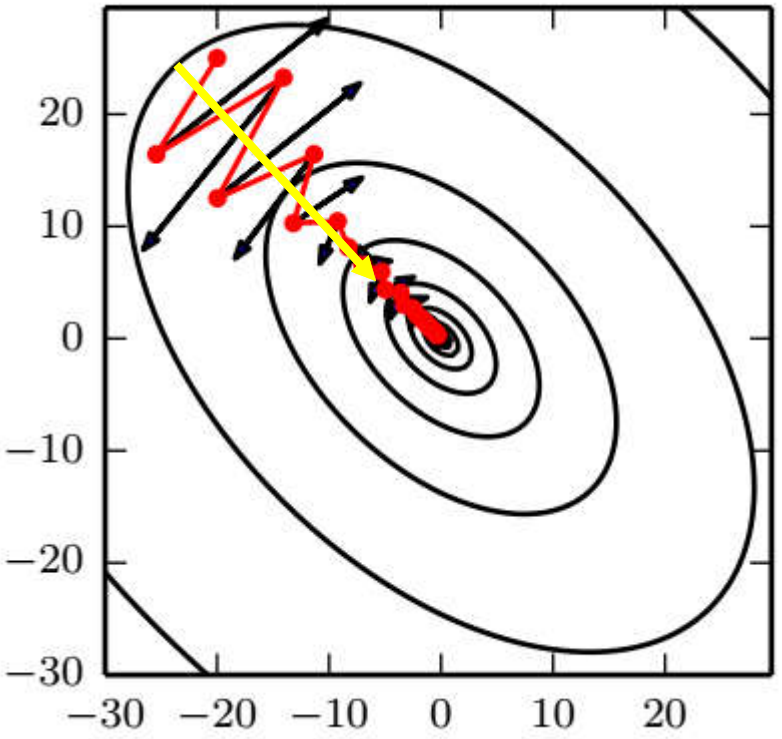
- 震荡引起梯度下降慢
- 震荡来源：病态、随机小批量采样
- 动量算法引入过往的速度积累来修正当前梯度

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right)$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}.$$

- v积加之前的梯度

$$\nabla_{\boldsymbol{\theta}}\left(\frac{1}{m}\sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})\right)$$



基于动量的算法

- α越大，之前梯度影响越大。
- 初期小，后期逐渐增大。步长 $\frac{\epsilon \|\boldsymbol{g}\|}{1-\alpha}$ 逐渐减小
- 后期调整没有ϵ重要

算法 8.2 使用动量的随机梯度下降（SGD）

Require: 学习率 ϵ，动量参数 α

Require: 初始参数 $\boldsymbol{\theta}$，初始速度 $\boldsymbol{v}$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

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

计算速度更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

基于动量的算法

- Nesterov和一般动量方法的区别：梯度计算在施加动量之后，对梯度有校正作用。
- 凸批量收敛率改进O(1/k)到O(1/k^2)。随机梯度没有改进。

算法 8.3 使用Nesterov 动量的随机梯度下降（SGD）

Require: 学习率 ϵ，动量参数 α

Require: 初始参数 $\boldsymbol{\theta}$，初始速度 $\boldsymbol{v}$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

应用临时更新：$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

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

计算速度更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

自适应学习率算法

- 回顾之前，学习率ϵ的设置在很多地方都很关键，也很难
- 考虑因素：学习速度、收敛、梯度稳定、梯度消失/爆炸。。
- 动量方法缓解了部分方向敏感问题，但引入另一个超参数
- 早期一些启发式，例如Delta-bar-delta：如果损失对于某个给定模型参数偏导保持相同符号，学习率增加，反之，减小。（用于全批量）
- 最近，一些基于小批量的算法

自适应学习率算法

- AdaGrad:
- 根据每个参数的历史梯度平方根反比，缩放每个参数
- 损失最大偏导参数学习率快速下降，否则较小下降
- 参数空间更为平缓的倾斜方向会取得更大进步
- 凸优化背景有令人满意的理论性质
- 问题：训练开始积累梯度平方，导致有效学习率过早过量减小

自适应学习率算法

算法 8.4 AdaGrad 算法

Require: 全局学习率 ϵ

Require: 初始参数 $\boldsymbol{\theta}$

Require: 小常数 δ，为了数值稳定大约设为 10^{-7}

初始化梯度累积变量 $\boldsymbol{r} = 0$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

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

累积平方梯度：$\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$

计算更新：$\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$ （逐元素地应用除和求平方根）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

自适应学习率算法

- RMSProp:
- 修改AdaGrad以在非凸设定下效果更好
- 梯度积累改为指数加权的移动平均，更远的梯度指数衰减更厉害，从而学习率衰减影响只限于近期
- 缓解了学习率过量衰减问题
- 另有结合动量的RMSProp
- 深度学习从业者常采用

自适应学习率算法

算法 8.5 RMSProp 算法

Require: 全局学习率 ϵ，衰减速率 ρ

Require: 初始参数 $\boldsymbol{\theta}$

Require: 小常数 δ，通常设为 10^{-6}（用于被小数除时的数值稳定）

初始化累积变量 $\boldsymbol{r} = 0$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

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

累积平方梯度：$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

计算参数更新：$\Delta \boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$ （$\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ 逐元素应用）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$

end while

自适应学习率算法

算法 8.6 使用 Nesterov 动量的 RMSProp 算法

Require: 全局学习率 ϵ，衰减速率 ρ，动量系数 α

Require: 初始参数 $\boldsymbol{\theta}$，初始参数 $\boldsymbol{v}$

初始化累积变量 $\boldsymbol{r} = 0$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

计算临时更新：$\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

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

累积梯度：$\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

计算速度更新：$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$ （$\frac{1}{\sqrt{\boldsymbol{r}}}$ 逐元素应用）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

自适应学习率算法

- Adam （adaptive moments）：
- 可看做RMSProp和一些动量的变种
- 1. 动量并入梯度一阶矩（指数加权）的估计。RMSProp没有把动量用在梯度校正上
- 2. Adam修正一阶和二阶矩的偏置（相当于逐渐增大α）。RMSProp没有
- Adam被认为对超参数的选择相当鲁棒

自适应学习率算法

算法 8.7 Adam 算法

Require: 步长 ϵ（建议默认为：0.001）

Require: 矩估计的指数衰减速率，ρ_1 和 ρ_2 在区间 $[0, 1)$ 内。（建议默认为：分别为 0.9 和 0.999）

Require: 用于数值稳定的小常数 δ（建议默认为：10^{-8}）

Require: 初始参数 $\boldsymbol{\theta}$

初始化一阶和二阶矩变量 $\boldsymbol{s} = 0, \boldsymbol{r} = 0$

初始化时间步 $t = 0$

while 没有达到停止准则 **do**

从训练集中采包含 m 个样本 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 的小批量，对应目标为 $\boldsymbol{y}^{(i)}$。

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

$t \leftarrow t + 1$

更新有偏一阶矩估计：$\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$

更新有偏二阶矩估计：$\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$

修正一阶矩的偏差：$\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$

修正二阶矩的偏差：$\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$

计算更新：$\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ （逐元素应用操作）

应用更新：$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

end while

二阶近似方法

- 一阶方法：
  - 只采用一阶梯度
  - 只考虑当前坡度最大的方向走一步，走一步算一步
  - 计算简单
- 二阶方法：
  - 采用二阶hessain矩阵信息来修正梯度方向
  - 不仅考虑当前坡度是否大，还考虑后面好不好走
  - 牛顿法（后面坡度是否会更大），共轭梯度（能不能少走几步）
  - 某些方法（例如牛顿法）每次迭代要计算二阶hessian矩阵，计算量增大
  - Hessian矩阵在某些情况下失效

牛顿法

- 回顾之前泰勒公式， 令$g=\nabla_\theta J(\theta)$：

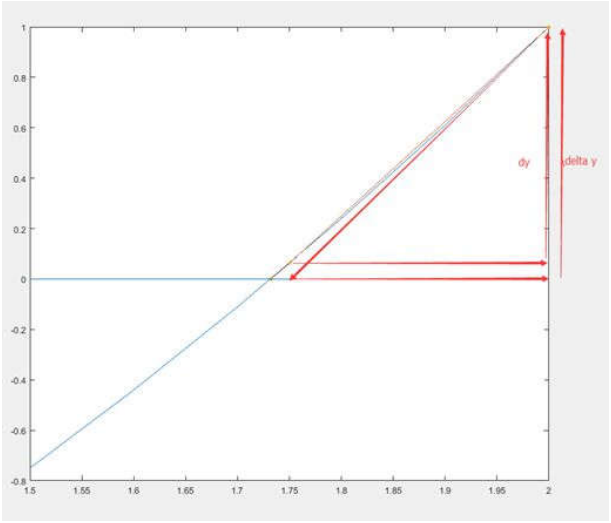
$$J(\theta) \approx J(\theta_0) + (\theta-\theta_0)^T g + \frac{1}{2}(\theta-\theta_0)^T H(\theta-\theta_0)$$

- 直接求此函数临界点（沿梯度方向的极值，对θ 求导为0）

$$\theta^* = \theta_0 - H^{-1} g$$

- H保持正定时，牛顿法能迭代应用，并一直下降
- 对比最速下降： $\theta_{n+1} = \theta_n - \epsilon g$
- 牛顿思路：极值点：一阶导为0。

选条导数一直减少到0的路。每次选导数变化最大的方向，而不是梯度方向（值变化最大的方向）。



牛顿法

算法 8.8 目标为 $J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), y^{(i)})$ 的牛顿法

Require: 初始参数 $\boldsymbol{\theta}_0$

Require: 包含 m 个样本的训练集

while 没有达到停止准则 **do**

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

计算 Hessian 矩阵：$\boldsymbol{H} \leftarrow \frac{1}{m}\nabla^2_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

计算 Hessian 逆：$\boldsymbol{H}^{-1}$

计算更新：$\Delta\boldsymbol{\theta} = -\boldsymbol{H}^{-1}\boldsymbol{g}$

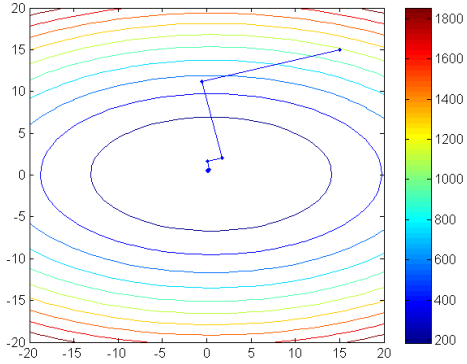
应用更新：$\boldsymbol{\theta} = \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$

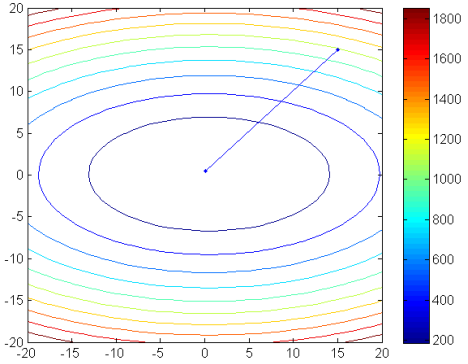
end while

- BFGS近似了牛顿法中的H求逆，并在此方向上线性搜索存储空间要求大$O(n^2)$，L-BFGS是存储的优化版本

牛顿法

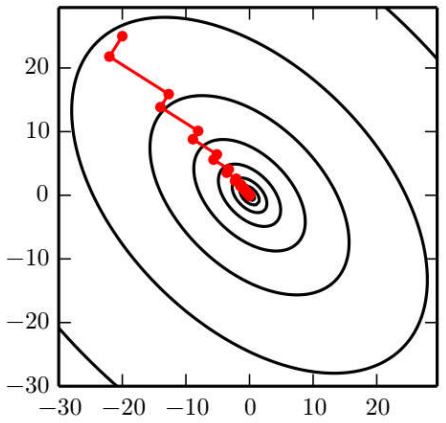
- 牛顿法：曲面拟合，二阶收敛，计算量大:H^{-1}求解$O(k^3)$
- 最速下降：平面拟合，一阶收敛
- 大部分情况下，曲面拟合更好，牛顿法更快
- 问题：H非正定情况下（鞍点等非凸点），牛顿法方向错误，常采用正则化方法修正 $\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - [H(f(\boldsymbol{\theta}_0)) + \alpha \boldsymbol{I}]^{-1} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_0).$
- 曲率更极端情况下，H会由I主导，牛顿退化为普通梯度，而过大的a导致牛顿法步长更小。





共轭梯度法

- 回想最速下降法等梯度方法中的锯齿形梯度震荡
- 反复震荡的一个原因：后期的搜索没有利用前期搜索方向上的优化进展。共轭梯度法试图保留原先搜索方向上的进展，减少迭代次数。
- 为此设计一组关于H共轭的搜索方向： $d_t^T H d_{t-1} = 0$
- 并且修正搜索方向： $d_t = \nabla_\theta J(\theta) + \beta_t d_{t-1}$
- 利用二次函数共轭性质：
  - 关于H的共轭向量线性独立
  - 则d_t构成k维空间的一组基（有限k个）
  - 可证之后迭代点的梯度g_i与之前d_j均正交
  - $g_i d_j$=0（i>j）（每个点都是该方向上极值点）
  - 和所有基都正交的向量为0，即g_k=0，即k步到极值。



共轭梯度法

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 $g_0 = 0$

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

共轭梯度法

- 共轭法的β_t可用更高效的计算给出，避免计算H
- K维参数空间中，至多k步找到最小值（二次曲面）
- 非线性共轭：目标函数非二次，共轭不能保证在以前方向上的目标仍是极小值，非线性目标函数上，会有一些修改
- 共轭梯度法是共轭性与最速下降法的结合
- 本质上，就是把目标函数分成许多方向，然后不同方向分别求出极值再综合起来