《神经网络与深度学习》

神经网络优化

https://nndl.github.io/

深度学习的矛与盾

优化
经验风险最小

正则化
降低模型复杂度

网络优化

参数学习

- 给定训练集为$D = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，将每个样本$\boldsymbol{x}^{(n)}$输入给前馈神经网络，得到网络输出为$\hat{y}^{(n)}$，其在数据集D上的结构化风险函数为

$$\mathcal{R}(W, \mathbf{b}) = \frac{1}{N}\sum_{n=1}^{N}\mathcal{L}(\mathbf{y}^{(n)}, \hat{\mathbf{y}}^{(n)}) + \frac{1}{2}\lambda\|W\|_F^2$$

- 梯度下降

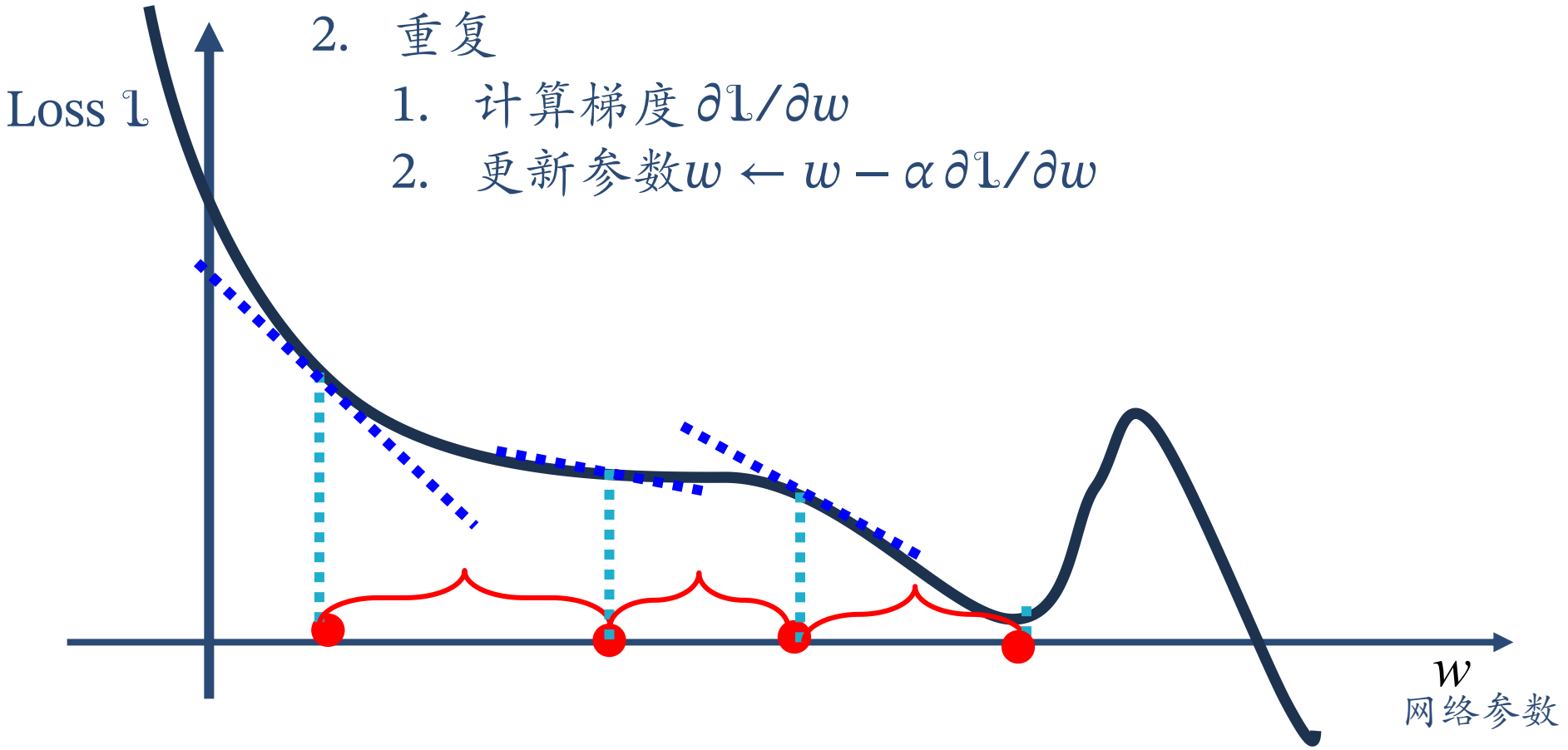
$$W^{(l)} \leftarrow W^{(l)} - \alpha\frac{\partial\mathcal{R}(W, \mathbf{b})}{\partial W^{(l)}}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha\frac{\partial\mathcal{R}(W, \mathbf{b})}{\partial \mathbf{b}^{(l)}}$$

梯度下降

梯度：$\frac{\partial f(w)}{\partial w} = \lim_{\Delta w \to 0} \frac{f(w+\Delta w)}{\Delta w}$

1. 初始化w
2. 重复
   1. 计算梯度$\partial \mathbb{L} / \partial w$
   2. 更新参数$w \leftarrow w - \alpha \partial \mathbb{L} / \partial w$



如何计算梯度？

- 神经网络为一个复杂的复合函数
  - 链式法则

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

- 反向传播算法
  - 根据前馈网络的特点而设计的高效方法

- 一个更加通用的计算方法
  - 自动微分（Automatic Differentiation，AD）

链式法则

- 链式法则是在微积分中求复合函数导数的一种常用方法

(1)若 $x \in \mathbb{R}, \boldsymbol{u} = u(x) \in \mathbb{R}^s, \boldsymbol{g} = g(\boldsymbol{u}) \in \mathbb{R}^t$, 则

$$\frac{\partial \boldsymbol{g}}{\partial x} = \frac{\partial \boldsymbol{u}}{\partial x} \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{u}} \quad \in \mathbb{R}^{1 \times t}.$$

(2)若 $\boldsymbol{x} \in \mathbb{R}^p, \boldsymbol{y} = g(\boldsymbol{x}) \in \mathbb{R}^s, \boldsymbol{z} = f(\boldsymbol{y}) \in \mathbb{R}^t$, 则

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}} \quad \in \mathbb{R}^{p \times t}.$$

(3)若 $X \in \mathbb{R}^{p \times q}$ 为矩阵, $\boldsymbol{y} = g(X) \in \mathbb{R}^s, z = f(\boldsymbol{y}) \in \mathbb{R}$, 则

$$\frac{\partial z}{\partial X_{ij}} = \frac{\partial \boldsymbol{y}}{\partial X_{ij}} \frac{\partial z}{\partial \boldsymbol{y}} \quad \in \mathbb{R}.$$

反向传播算法

$$\mathbf{z}^{(l)} = W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}$$

$$\frac{\partial \mathbf{z}^{(l)}}{\partial w_{ij}^{(l)}} = \left[\frac{\partial z_1^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}, \cdots, \frac{\partial z_{m^{(l)}}^{(l)}}{\partial w_{ij}^{(l)}}\right]$$

$$= \left[0, \cdots, \frac{\partial(\mathbf{w}_{i:}^{(l)}\mathbf{a}^{(l-1)} + b_i^{(l)})}{\partial w_{ij}^{(l)}}, \cdots, 0\right]$$

$$= \left[0, \cdots, a_j^{(l-1)}, \cdots, 0\right]$$

$$\triangleq \mathbb{I}_i(a_j^{(l-1)}) \quad \in \mathbb{R}^{m^{(l)}},$$

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

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

$$\delta^{(l)} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \quad \in \mathbb{R}^{m^{(l)}}$$

误差项

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{b}^{(l)}} = \mathbf{I}_{m^{(l)}} \quad \in \mathbb{R}^{m^{(l)} \times m^{(l)}}$$

计算 $\delta^{(l)} = \dfrac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}} \in \mathbb{R}^{m^{(l)}}$

$$\mathbf{a}^{(l)} = f_l(\mathbf{z}^{(l)})$$

$$\mathbf{z}^{(l+1)} = W^{(l+1)}\mathbf{a}^{(l)} + \mathbf{b}^{(l+1)}$$

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} = \frac{\partial f_l(\mathbf{z}^{(l)})}{\partial \mathbf{z}^{(l)}} = \mathrm{diag}(f_l'(\mathbf{z}^{(l)}))$$

$$\frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} = (W^{(l+1)})^{\mathrm{T}}$$

$$\delta^{(l)} \triangleq \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l)}}$$

$$= \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l)}} \cdot \frac{\partial \mathbf{z}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \cdot \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}^{(l+1)}}$$

$$= \mathrm{diag}(f_l'(\mathbf{z}^{(l)})) \cdot (W^{(l+1)})^{\mathrm{T}} \cdot \delta^{(l+1)}$$

$$= f_l'(\mathbf{z}^{(l)}) \odot \big((W^{(l+1)})^{\mathrm{T}} \delta^{(l+1)}\big),$$

反向传播算法

$$W^{(l)} \leftarrow W^{(l)} - \alpha \frac{\partial \mathcal{R}(W, \mathbf{b})}{\partial W^{(l)}}$$

$$\mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \alpha \frac{\partial \mathcal{R}(W, \mathbf{b})}{\partial \mathbf{b}^{(l)}}$$

在计算出上面三个偏导数之后，公式(4.49)可以写为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{(l)}} = \mathbb{I}_i(a_j^{(l-1)})\delta^{(l)} = \delta_i^{(l)} a_j^{(l-1)}.$$

进一步，$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$关于第$l$层权重$W^{(l)}$的梯度为

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial W^{(l)}} = \delta^{(l)}(\mathbf{a}^{(l-1)})^{\mathrm{T}}.$$

同理，$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$关于第$l$层偏置$\mathbf{b}^{(l)}$的梯度为

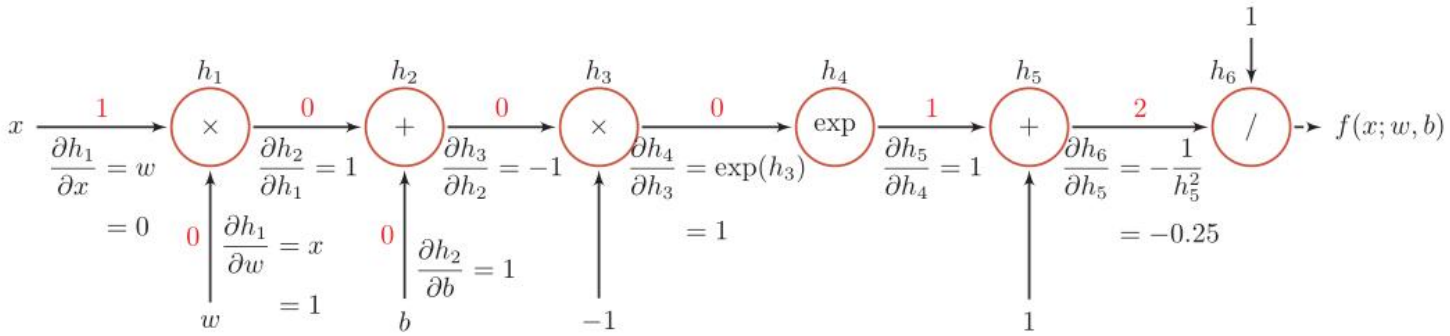
$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{b}^{(l)}} = \delta^{(l)}.$$

计算复杂，能否自动梯度计算？

计算图与自动微分

- 自动微分是利用链式法则来自动计算一个复合函数的梯度

$$f(x;w,b)=\frac{1}{\exp(-(wx+b))+1}$$



函数	导数	
$h_1 = x \times w$	$\frac{\partial h_1}{\partial w} = x$	$\frac{\partial h_1}{\partial x} = w$
$h_2 = h_1 + b$	$\frac{\partial h_2}{\partial h_1} = 1$	$\frac{\partial h_2}{\partial b} = 1$
$h_3 = h_2 \times -1$	$\frac{\partial h_3}{\partial h_2} = -1$	
$h_4 = \exp(h_3)$	$\frac{\partial h_4}{\partial h_3} = \exp(h_3)$	
$h_5 = h_4 + 1$	$\frac{\partial h_5}{\partial h_4} = 1$	
$h_6 = 1/h_5$	$\frac{\partial h_6}{\partial h_5} = -\frac{1}{h_5^2}$	

当x = 1,w = 0,b = 0时，可以得到

$$\begin{aligned}\frac{\partial f(x;w,b)}{\partial w}|_{x=1,w=0,b=0} &= \frac{\partial f(x;w,b)}{\partial h_6}\frac{\partial h_6}{\partial h_5}\frac{\partial h_5}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w}\\ &= 1\times -0.25\times 1\times 1\times -1\times 1\times 1\\ &= 0.25.\end{aligned}$$

自动微分

- 前向模式和反向模式
  - 反向模式和反向传播的计算梯度的方式相同

- 前馈神经网络的训练过程可以分为以下三步
  - 前向计算每一层的状态和激活值，直到最后一层
  - 反向计算每一层的参数的偏导数
  - 更新参数

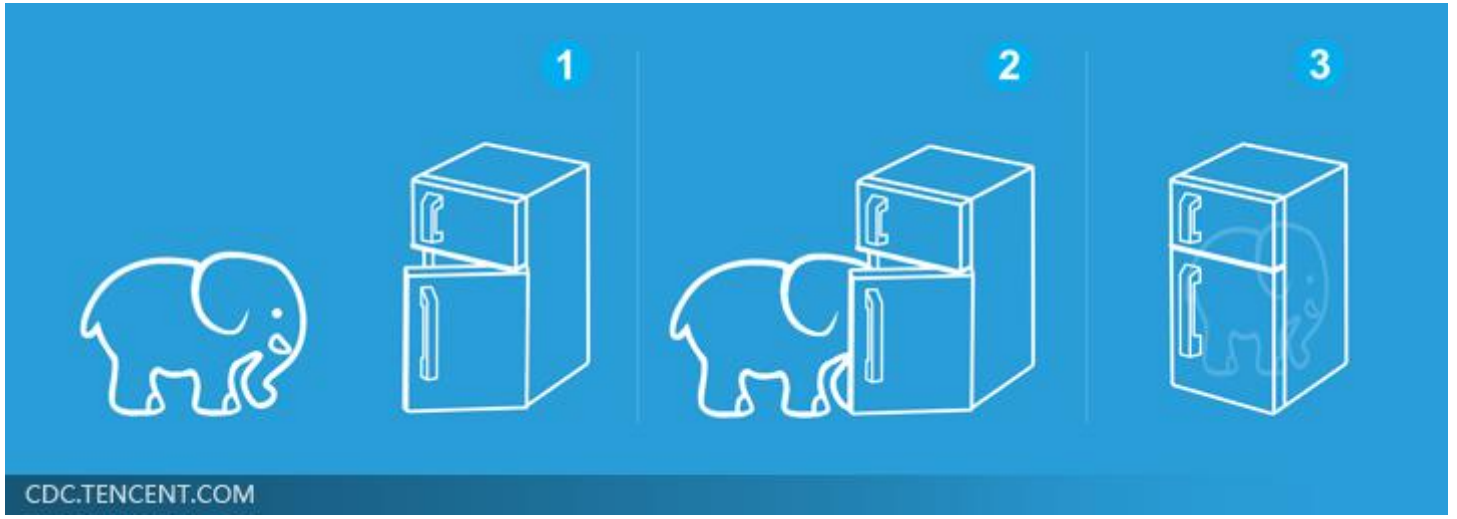
静态计算图和动态计算图

- 静态计算图是在编译时构建计算图，计算图构建好之后在程序运行时不能改变
  - Theano
  - Tensorflow
- 动态计算图是在程序运行时动态构建
  - DyNet
  - PyTorch

- 静态计算图在构建时可以进行优化，并行能力强，但灵活性比较差
- 动态计算图则不容易优化，当不同输入的网络结构不一致时，难以并行计算，但是灵活性比较高

深度学习的三个步骤

Deep Learning is so simple ……



难点

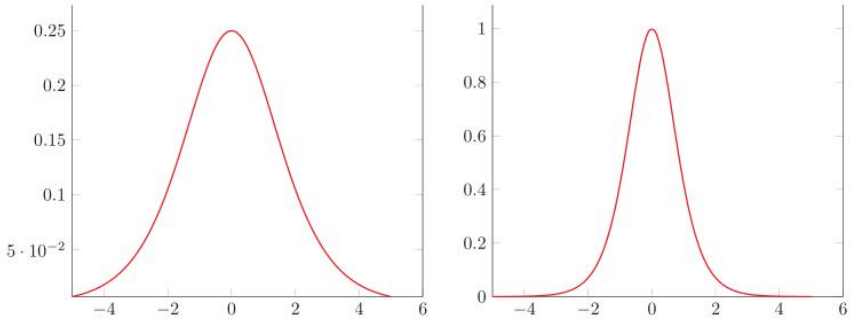
- 结构差异大
  - 没有通用的优化算法
  - 超参数多

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

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

- 非凸优化问题
  - 参数初始化
  - 逃离局部最优

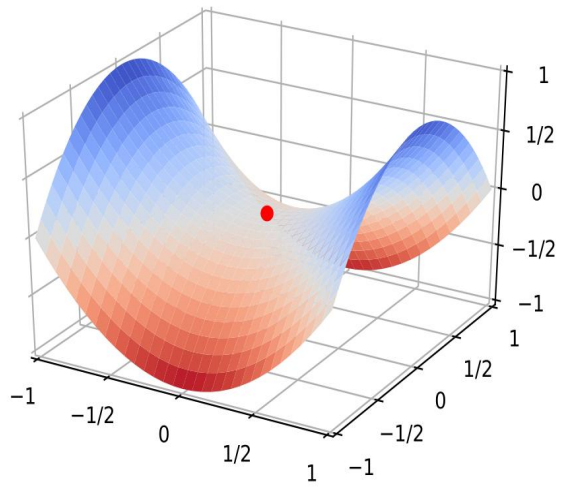
- 梯度消失（爆炸）问题



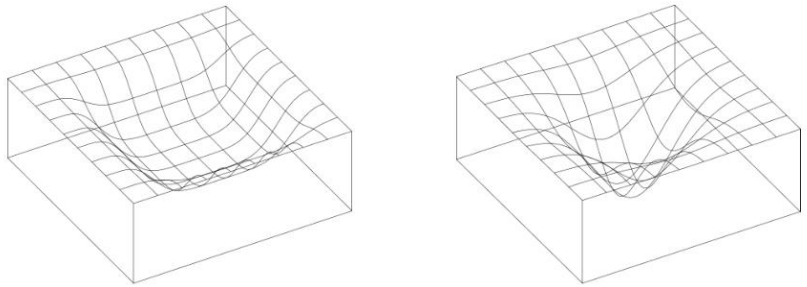
(a) logistic 函数的导数　(b) tanh 函数的导数

高维空间的非凸优化问题

- 鞍点（Saddle Point）
  - 驻点（Stationary Point）：梯度为 0 的点



- 平坦最小值（Flat Minima）
  - 一个平坦最小值的邻域内，所有点对应的训练损失都比较接近
  - 大部分的局部最小解是等价的
  - 局部最小解对应的训练损失都可能非常接近于全局最小解对应的训练损失



(a) 平坦最小值　　(b) 尖锐最小值

神经网络优化的改善方法

- 更有效的优化算法来提高优化方法的效率和稳定性
  - 动态学习率调整
  - 梯度估计修正
- 更好的参数初始化方法、数据预处理方法来提高优化效率
- 修改网络结构来得到更好的优化地形
  - 优化地形（Optimization Landscape）指在高维空间中损失函数的曲面形状
  - 好的优化地形通常比较平滑
  - 使用 ReLU 激活函数、残差连接、逐层归一化等
- 使用更好的超参数优化方法

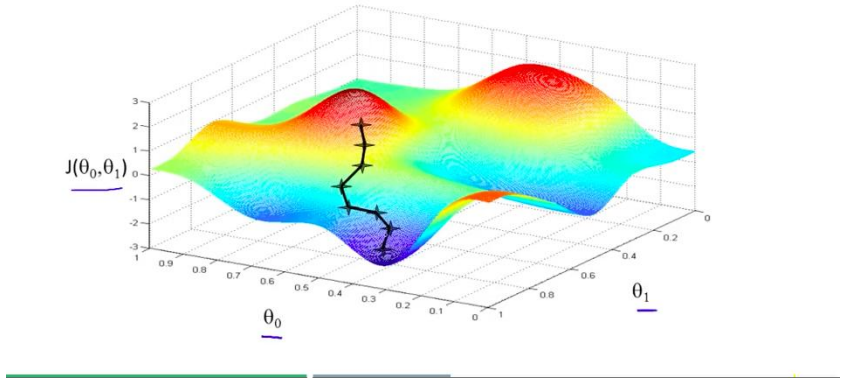
优化算法改进

优化算法：随机梯度下降

算法 2.1 随机梯度下降法

输入: 训练集 $\mathcal{D}=\{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$，验证集 $\mathcal{V}$，学习率 α

随机初始化 θ;
repeat
  对训练集 $\mathcal{D}$ 中的样本随机排序;
  for $n=1 \cdots N$ **do**
    从训练集 $\mathcal{D}$ 中选取样本 $(\boldsymbol{x}^{(n)}, y^{(n)})$;
    $\theta \leftarrow \theta-\alpha \dfrac{\partial \mathcal{L}\left(\theta; x^{(n)}, y^{(n)}\right)}{\partial \theta}$; // 更新参数
  end
until 模型 $f(\boldsymbol{x};\theta)$ 在验证集 $\mathcal{V}$ 上的错误率不再下降;
输出: θ



优化算法：小批量随机梯度下降 MiniBatch

- 选取K个训练样本$\{\mathbf{x}^{(k)}, \mathbf{y}^{(k)}\}_{k=1}^{K}$，计算偏导数

$$\mathcal{g}_t(\theta) = \frac{1}{K} \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{S}_t} \frac{\partial \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{x};\theta))}{\partial \theta}$$

- 定义梯度

$$\mathbf{g}_t \triangleq \mathcal{g}_t(\theta_{t-1})$$

- 更新参数

$$\theta_t \leftarrow \theta_{t-1} - \alpha \mathbf{g}_t$$

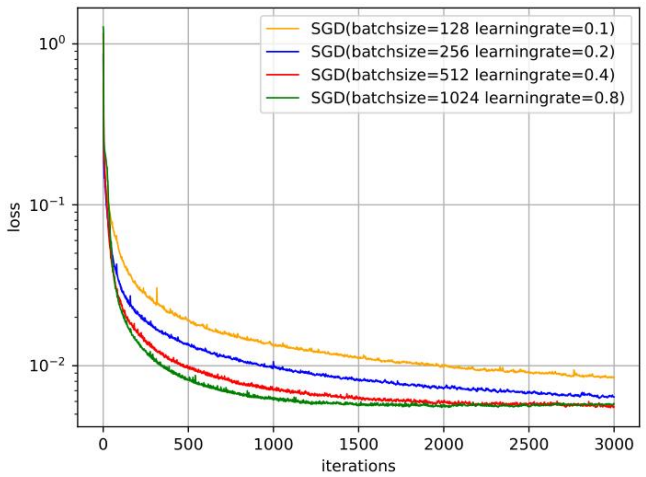
  - 其中$\alpha > 0$为学习率

几个关键因素：

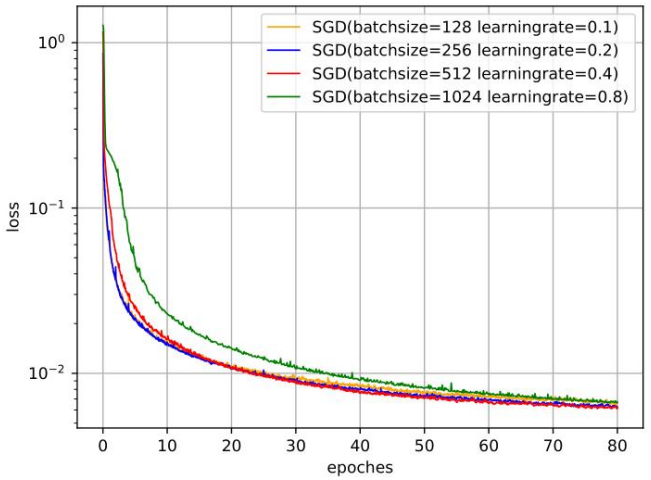
- 小批量样本数量
- 梯度
- 学习率

批量大小的影响

- 批量大小不影响随机梯度的期望，但是会影响随机梯度的方差
  - 批量越大，随机梯度的方差越小，引入的噪声也越小，训练也越稳定，因此可以设置较大的学习率
  - 而批量较小时，需要设置较小的学习率，否则模型会不收敛



(a) 按 Iteration 的损失变化



(b) 按 Epoch 的损失变化

4 种批量大小对应的学习率设置不同，因此并不是严格对比.

如何改进？

Reference:
1. An overview of gradient descent optimization algorithms
2. Optimizing the Gradient Descent

- 标准的（小批量）梯度下降
- 学习率
  - 学习率衰减
  - Adagrad
  - Adadelta
  - RMSprop
- 梯度
  - Momentum
    - 计算负梯度的“加权移动平均”作为参数的更新方向
  - Nesterov accelerated gradient
  - 梯度截断

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t$$

$$\Delta\theta_t = -\alpha\mathbf{g}_t$$

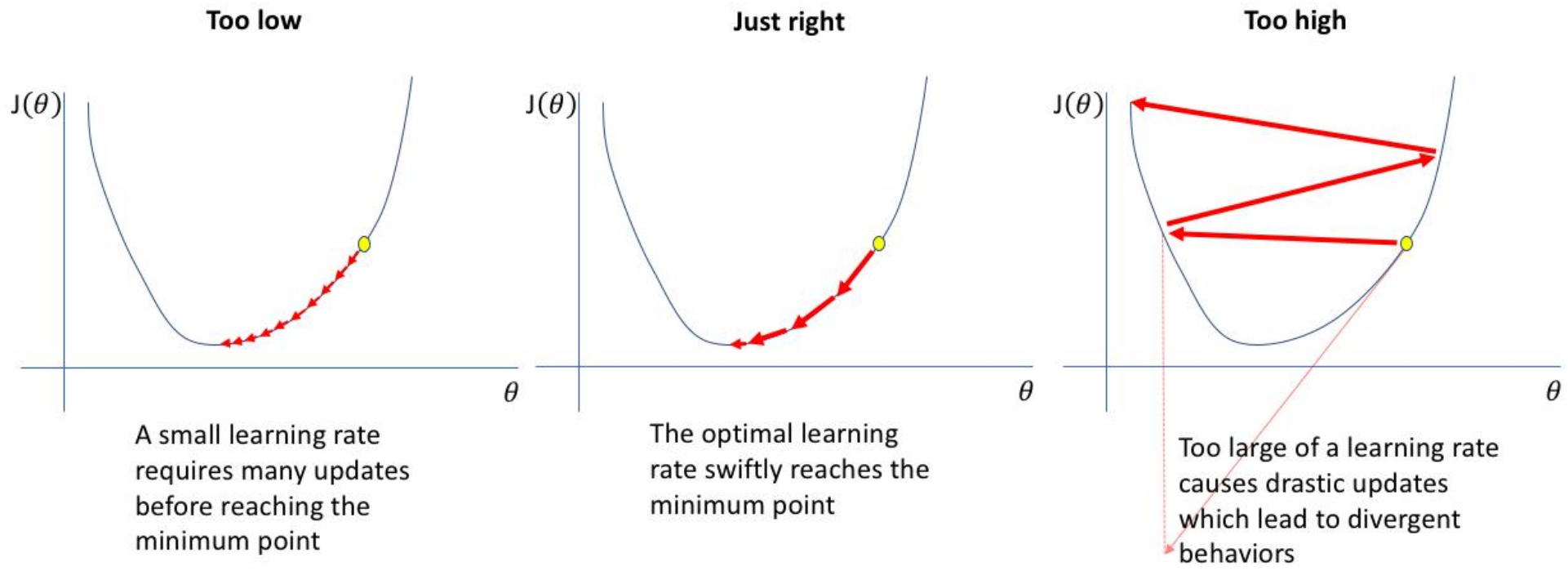
实际更新方向　　梯度方向

$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha\mathbf{g}_t$$

Adam

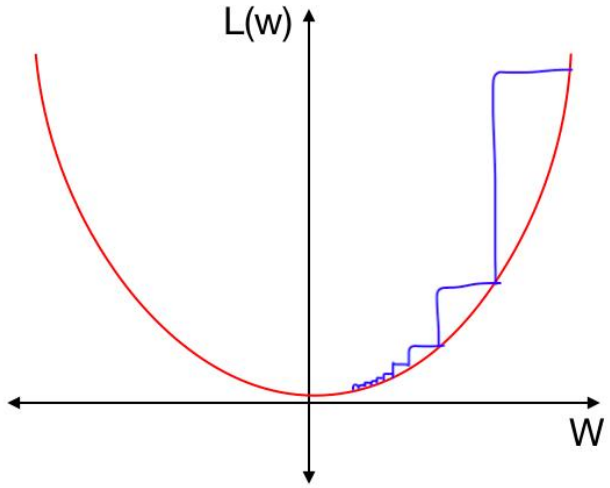
Adam is better choice!

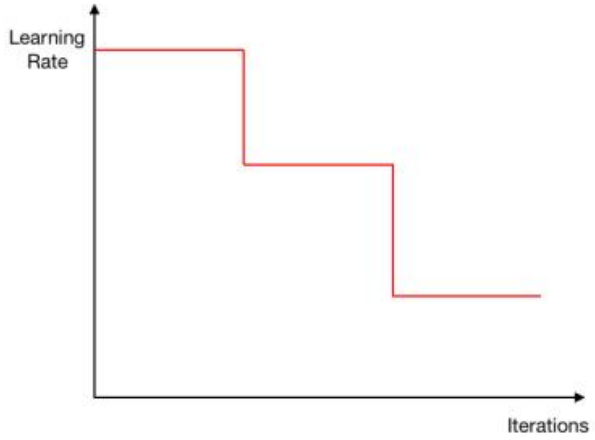
学习率的影响



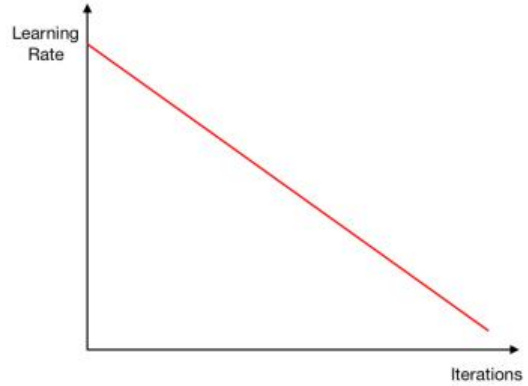
https://www.jeremyjordan.me/nn-learning-rate/

学习率衰减



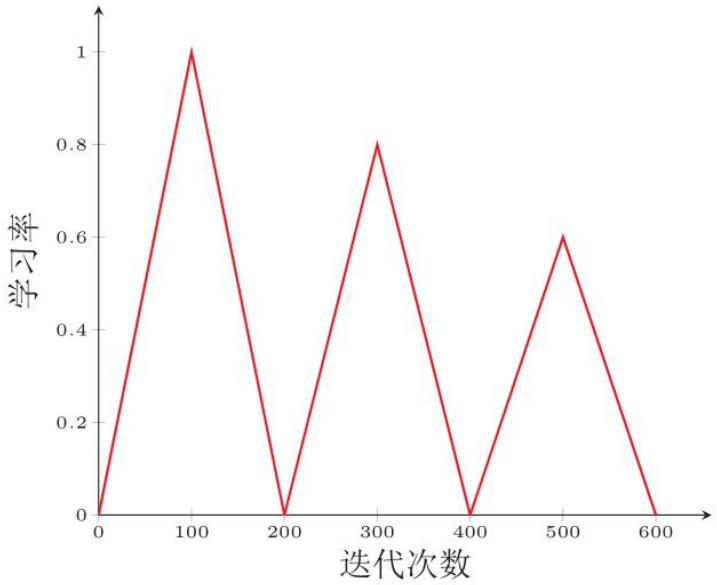


梯级衰减（step decay）

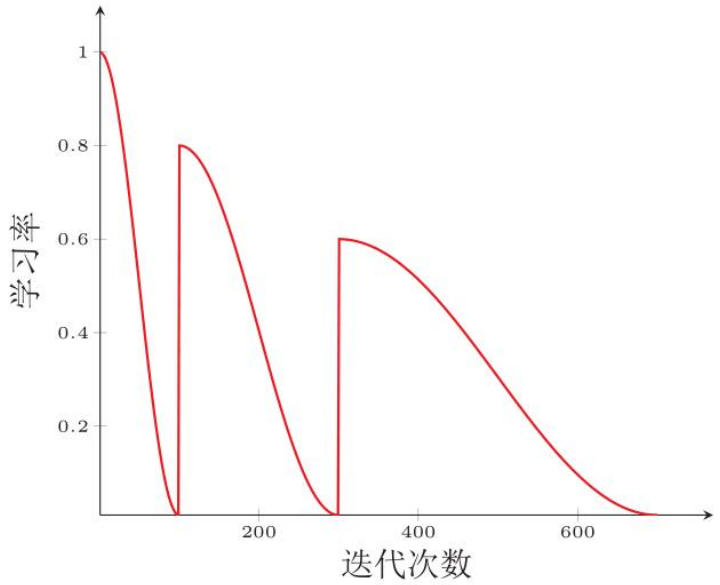


线性衰减（Linear Decay）

周期性学习率调整 Cyclical Learning Rates

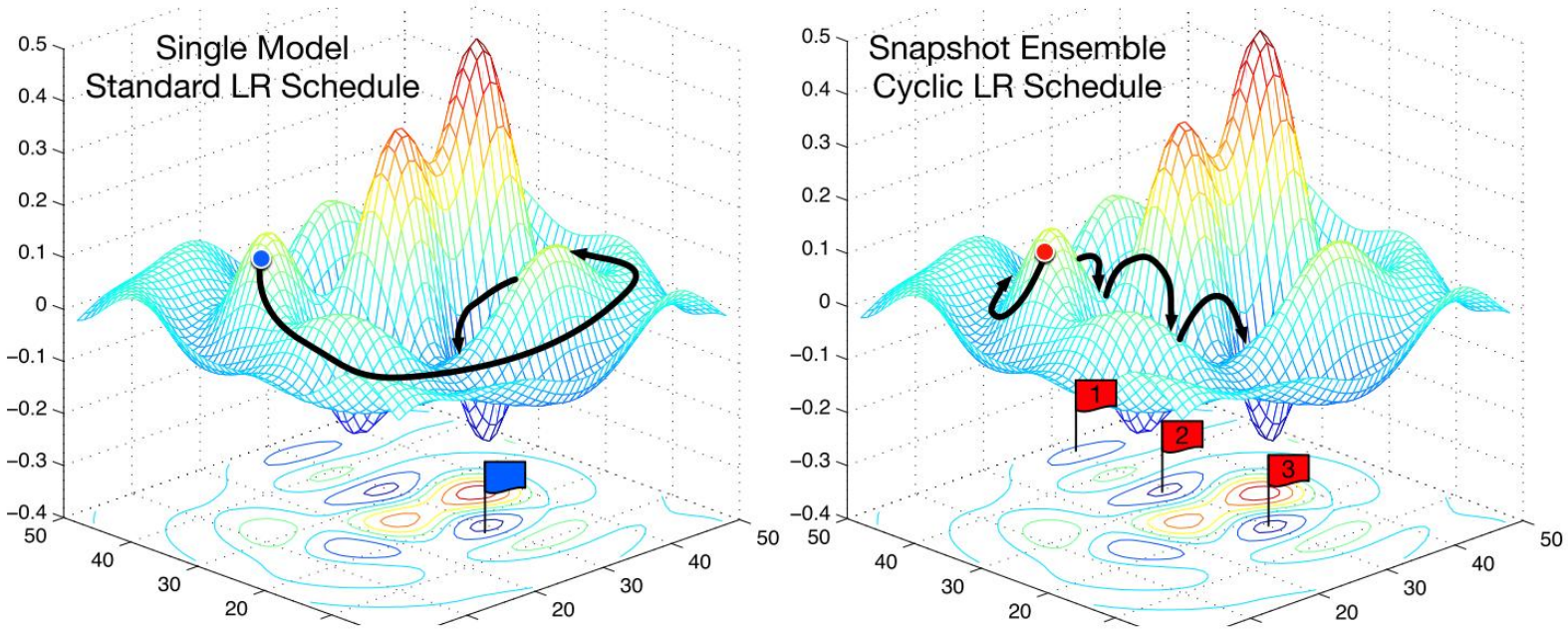


(a) 三角循环学习率

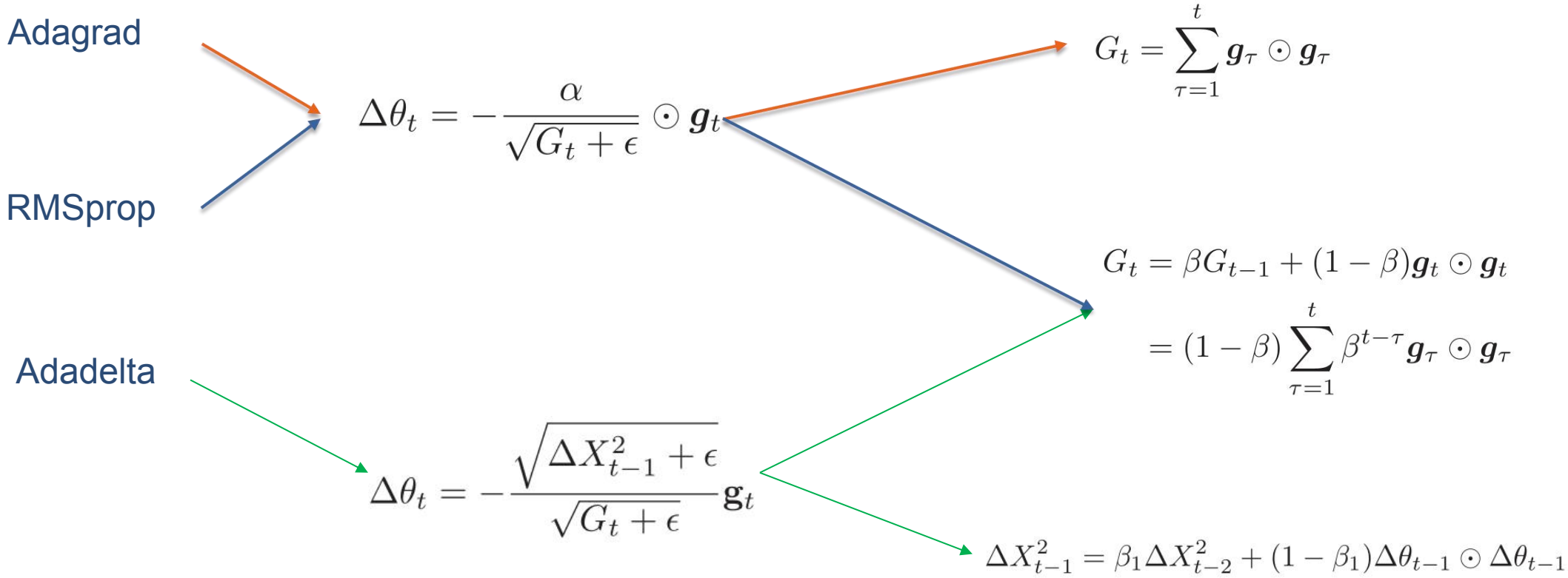


(b) 带热重启的余弦衰减

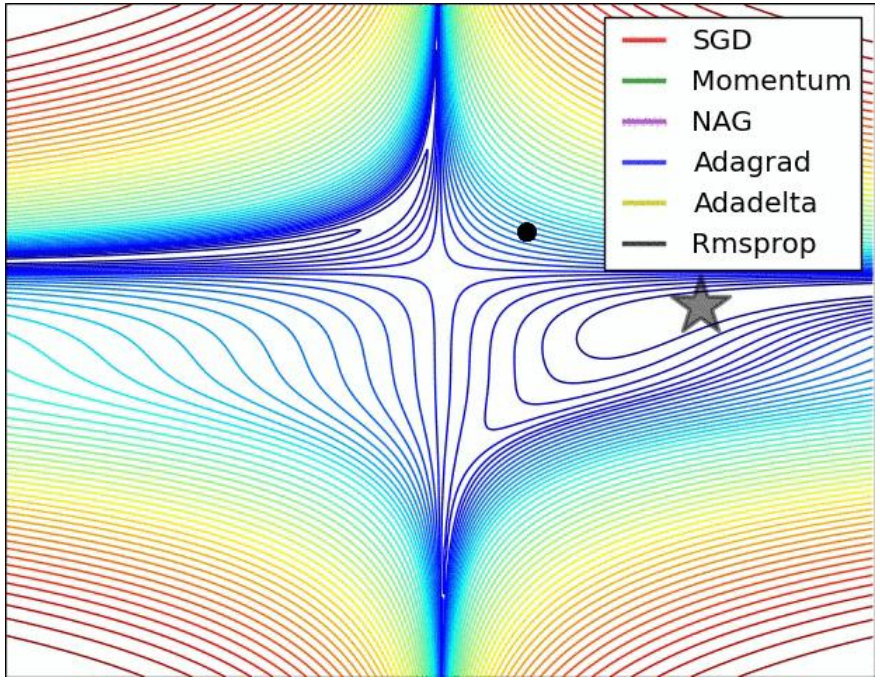
Cyclical Learning Rates

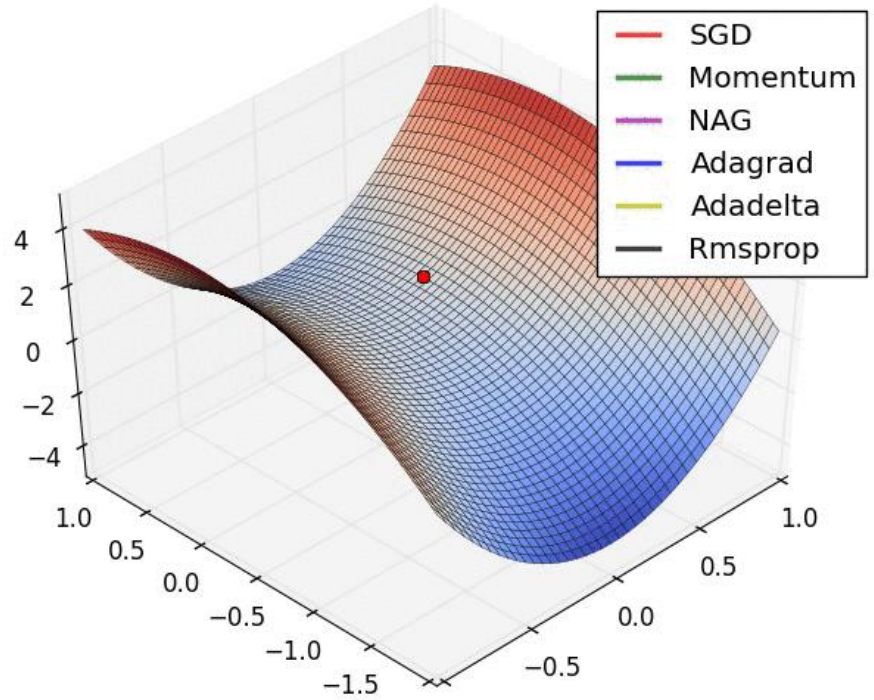


自适应学习率


Adagrad
RMSprop
Adadelta
$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \epsilon}} \odot \boldsymbol{g}_t$
$G_t = \sum_{\tau=1}^{t} \boldsymbol{g}_\tau \odot \boldsymbol{g}_\tau$
$G_t = \beta G_{t-1} + (1-\beta)\boldsymbol{g}_t \odot \boldsymbol{g}_t$
$= (1-\beta)\sum_{\tau=1}^{t} \beta^{t-\tau} \boldsymbol{g}_\tau \odot \boldsymbol{g}_\tau$
$\Delta\theta_t = -\frac{\sqrt{\Delta X_{t-1}^2 + \epsilon}}{\sqrt{G_t + \epsilon}} \mathbf{g}_t$
$\Delta X_{t-1}^2 = \beta_1 \Delta X_{t-2}^2 + (1-\beta_1)\Delta\theta_{t-1} \odot \Delta\theta_{t-1}$

优化


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

鞍点

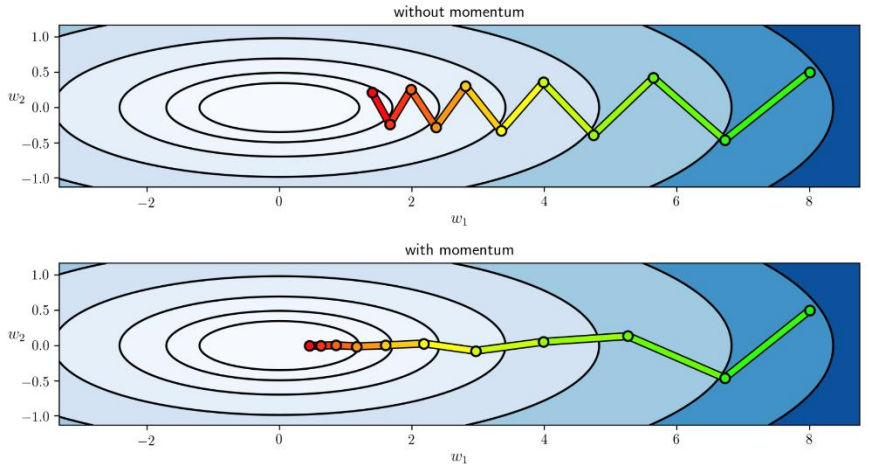
梯度方向优化

- 动量法（Momentum Method）
  - 用之前积累动量来替代真正的梯度，每次迭代的梯度可以看作是加速度

在第t次迭代时，计算负梯度的“加权移动平均”作为参数的更新方向，

$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha\mathbf{g}_t, = -\alpha\sum_{\tau=1}^{t}\beta^{t-\tau}\mathbf{g}_\tau, \tag{7.15}$$

其中ρ为动量因子，通常设为0.9，α为学习率。

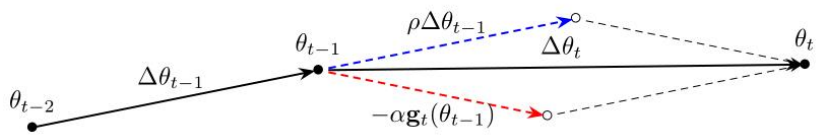


https://www.quora.com/What-exactly-is-momentum-in-machine-learning

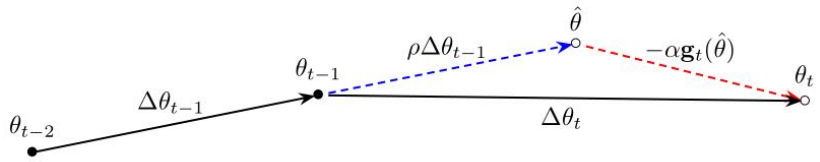
梯度方向优化

Nesterov加速梯度

$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha\mathfrak{g}_t(\theta_{t-1} + \rho\Delta\theta_{t-1})$$



(a) 动量法



(b) Nesterov 加速梯度

梯度方向优化+自适应学习率

- Adam算法≈动量法+RMSprop
  - 先计算两个移动平均

$$M_t = \beta_1 M_{t-1} + (1-\beta_1)\mathbf{g}_t,$$
$$G_t = \beta_2 G_{t-1} + (1-\beta_2)\mathbf{g}_t \odot \mathbf{g}_t,$$

  - 偏差修正

$$\hat{M}_t = \frac{M_t}{1-\beta_1^t},$$
$$\hat{G}_t = \frac{G_t}{1-\beta_2^t}.$$

  - 更新

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{\hat{G}_t + \epsilon}}\hat{M}_t$$

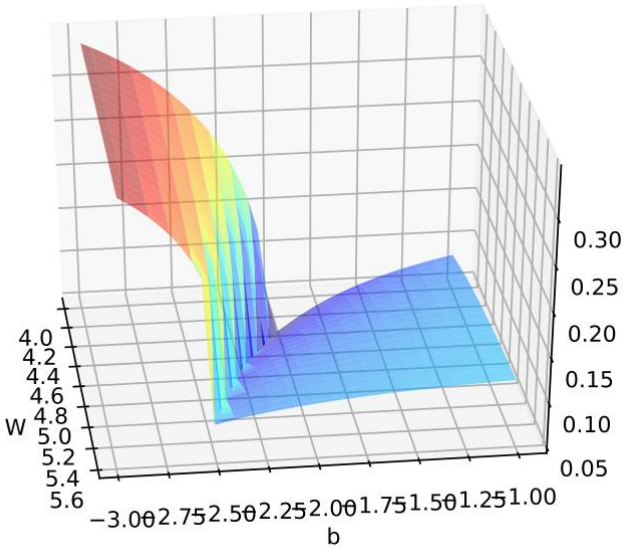
梯度截断

- 梯度截断是一种比较简单的启发式方法，把梯度的模限定在一个区间，当梯度的模小于或大于这个区间时就进行截断

- 按值截断 $\mathbf{g}_t = \max(\min(\mathbf{g}_t, b), a)$

- 按模截断

$$\mathbf{g}_t = \frac{b}{\|\mathbf{g}_t\|}\mathbf{g}_t$$



优化算法改进小结

- 大部分优化算法可以使用下面公式来统一描述概括

$$\Delta\theta_t = -\frac{\alpha_t}{\sqrt{G_t + \epsilon}} M_t,$$

$$G_t = \psi(\boldsymbol{g}_1, \cdots, \boldsymbol{g}_t),$$

$$M_t = \phi(\boldsymbol{g}_1, \cdots, \boldsymbol{g}_t),$$

类别		优化算法
学习率调整	固定衰减学习率	分段常数衰减、逆时衰减、(自然)指数衰减、余弦衰减
	周期性学习率	循环学习率、SGDR
	自适应学习率	AdaGrad、RMSprop、AdaDelta
梯度估计修正		动量法、Nesterov 加速梯度、梯度截断
综合方法		Adam≈动量法+RMSprop

g_t 为第t步的梯度

α_t 为第t步的学习率

参数初始化/数据预处理

随机初始化

- Gaussian分布初始化
  - Gaussian初始化方法是最简单的初始化方法，参数从一个固定均值（比如0）和固定方差（比如0.01）的Gaussian分布进行随机初始化

- 均匀分布初始化
  - 参数可以在区间[−r,r]内采用均匀分布进行初始化

随机初始化

- 范数保持性（Norm-Preserving）
  - 一个M层的等宽线性网络

$$\boldsymbol{y} = \boldsymbol{W}^{(L)}\boldsymbol{W}^{(L-1)}\cdots\boldsymbol{W}^{(1)}\boldsymbol{x}$$

  - 为了避免梯度消失或梯度爆炸问题，我们希望误差项

$$\|\delta^{(l-1)}\|^2 = \|\delta^{(l)}\|^2 = \|(\boldsymbol{W}^{(l)})^{\mathrm{T}}\delta^{(l)}\|^2$$

基于方差缩放的参数初始化

正交初始化

参数初始化

- 基于方差缩放的参数初始化
  - Xavier 初始化和 He 初始化

初始化方法	激活函数	均匀分布 $[-r, r]$	高斯分布 $\mathcal{N}(0, \sigma^2)$
Xavier 初始化	Logistic	$r = 4\sqrt{\frac{6}{M_{l-1}+M_l}}$	$\sigma^2 = 16 \times \frac{2}{M_{l-1}+M_l}$
Xavier 初始化	Tanh	$r = \sqrt{\frac{6}{M_{l-1}+M_l}}$	$\sigma^2 = \frac{2}{M_{l-1}+M_l}$
He 初始化	ReLU	$r = \sqrt{\frac{6}{M_{l-1}}}$	$\sigma^2 = \frac{2}{M_{l-1}}$

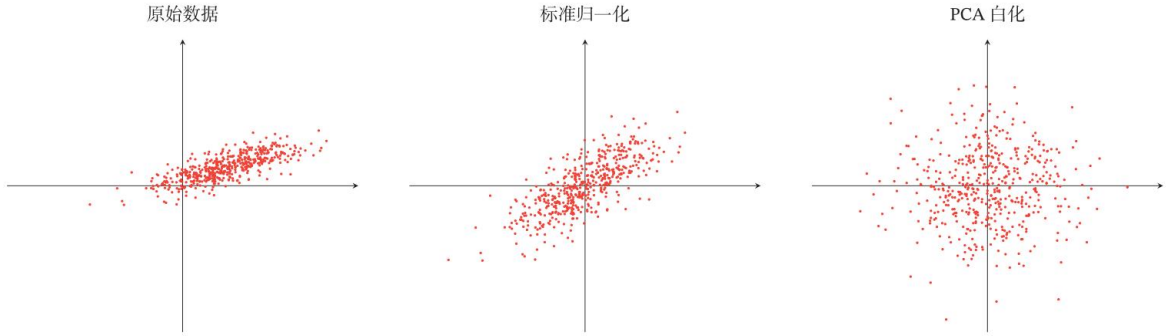
- 正交初始化
  - 用均值为 0、方差为 1 的高斯分布初始化一个矩阵
  - 将这个矩阵用奇异值分解得到两个正交矩阵，并使用其中之一作为权重矩阵

数据预处理

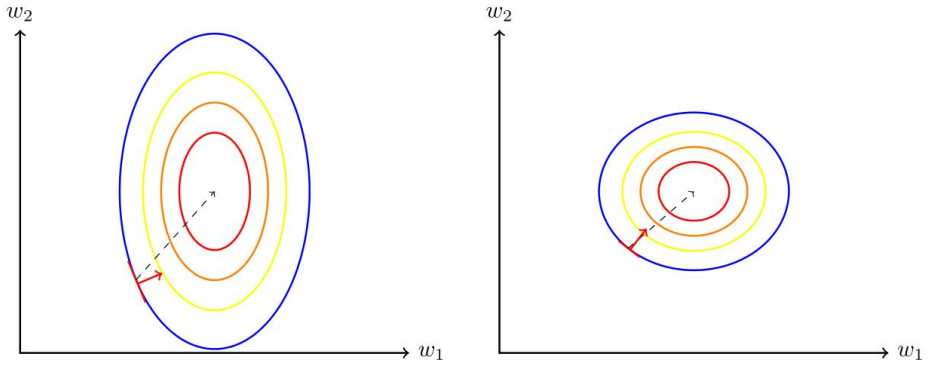
- 数据归一化
  - 最小最大值归一化
  - 标准化
  - PCA

$$\mu = \frac{1}{N}\sum_{n=1}^{N} x^{(n)},$$

$$\sigma^2 = \frac{1}{N}\sum_{n=1}^{N}(x^{(n)} - \mu)^2.$$



- 数据归一化对梯度的影响

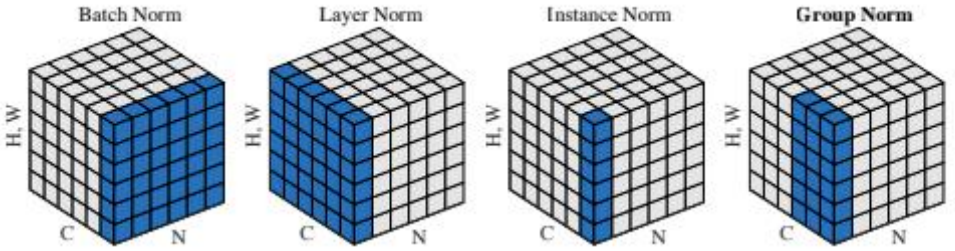


(a) 未归一化数据的梯度　　(b) 归一化数据的梯度

逐层归一化

逐层归一化

- 目的
  - 更好的尺度不变性
    - 内部协变量偏移
  - 更平滑的优化地形



- 归一化方法
  - 批量归一化（Batch Normalization，BN）
  - 层归一化（Layer Normalization）
  - 权重归一化（Weight Normalization）
  - 局部响应归一化（Local Response Normalization，LRN）

批量归一化

对于一个深层神经网络，令第l层的净输入为$\mathbf{z}^{(l)}$，神经元的输出为$\mathbf{a}^{(l)}$，即

$$\mathbf{a}^{(l)} = f(\mathbf{z}^{(l)}) = f(W\mathbf{a}^{(l-1)} + \mathbf{b}), \tag{7.42}$$

其中$f(\cdot)$是激活函数，W和$\mathbf{b}$是可学习的参数。

- 给定一个包含 K 个样本的小批量样本集合，计算均值和方差

$$\mu_{\mathcal{B}} = \frac{1}{K}\sum_{k=1}^{K}\mathbf{z}^{(k,l)},$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{K}\sum_{k=1}^{K}(\mathbf{z}^{(k,l)} - \mu_{\mathcal{B}}) \odot (\mathbf{z}^{(k,l)} - \mu_{\mathcal{B}}).$$

- 批量归一化

$$\hat{\mathbf{z}}^{(l)} = \frac{\mathbf{z}^{(l)} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \odot \gamma + \beta$$
$$\triangleq \mathrm{BN}_{\gamma,\beta}(\mathbf{z}^{(l)}),$$

层归一化

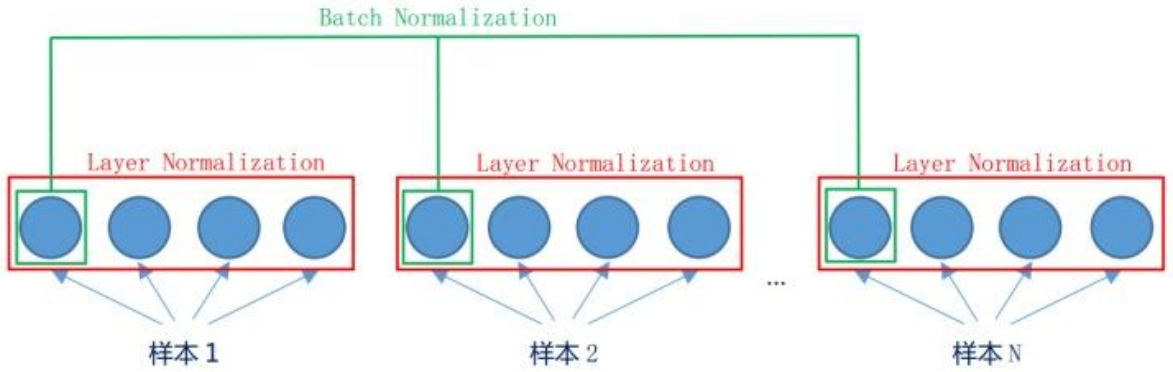
- 第l层神经元的净输入为$z^{(l)}$

$$\mu^{(l)} = \frac{1}{n^l}\sum_{i=1}^{n^l} z_i^{(l)},$$

$$\sigma^{(l)^2} = \frac{1}{n^l}\sum_{i=1}^{n^l}(z_i^{(l)} - \mu^{(l)})^2,$$

- 层归一化定义为

$$\hat{z}^{(l)} = \frac{z^{(l)} - \mu^{(l)}}{\sqrt{\sigma^{(l)^2} + \epsilon}} \odot \gamma + \beta$$
$$\triangleq \mathrm{LN}_{\gamma,\beta}(z^{(l)}),$$



超参数优化

超参数优化

- 超参数
  - 层数
  - 每层神经元个数
  - 激活函数
  - 学习率（以及动态调整算法）
  - 正则化系数
  - mini-batch 大小
- 优化方法
  - 网格搜索
  - 随机搜索
  - 贝叶斯优化
  - 动态资源分配
  - 神经架构搜索

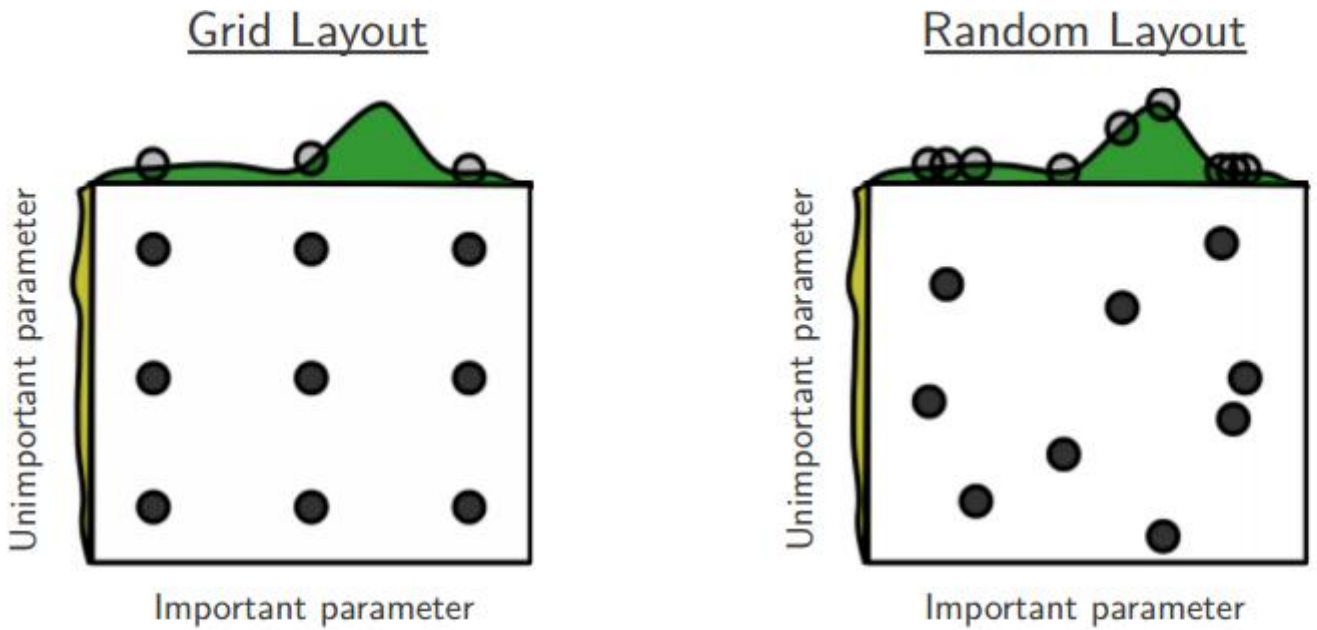
超参数优化

- 网格搜索（Grid Search）
  - 假设总共有 K 个超参数，第k个超参数的可以取m_k 个值
  - 如果参数是连续的，可以将参数离散化，选择几个“经验”值。比如学习率α，我们可以设置

$$\alpha \in \{0.01, 0.1, 0.5, 1.0\}$$

  - 这些超参数可以有 $m_1 \times m_2 \times \cdots \times m_K$ 个取值组合

超参数优化

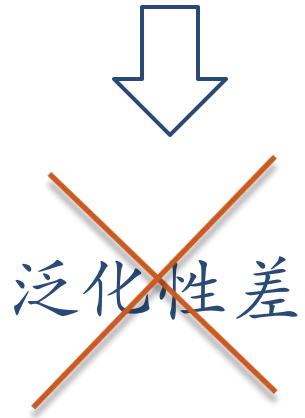


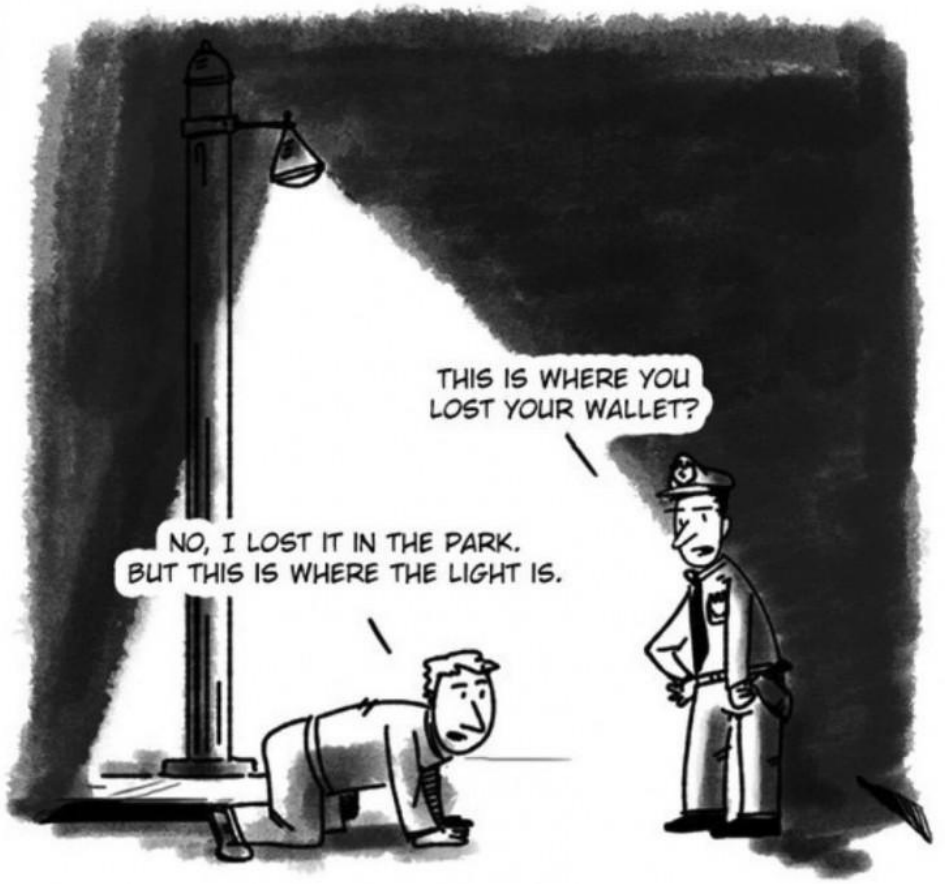
Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

网络正则化

重新思考泛化性

- 神经网络
  - 过度参数化
  - 拟合能力强



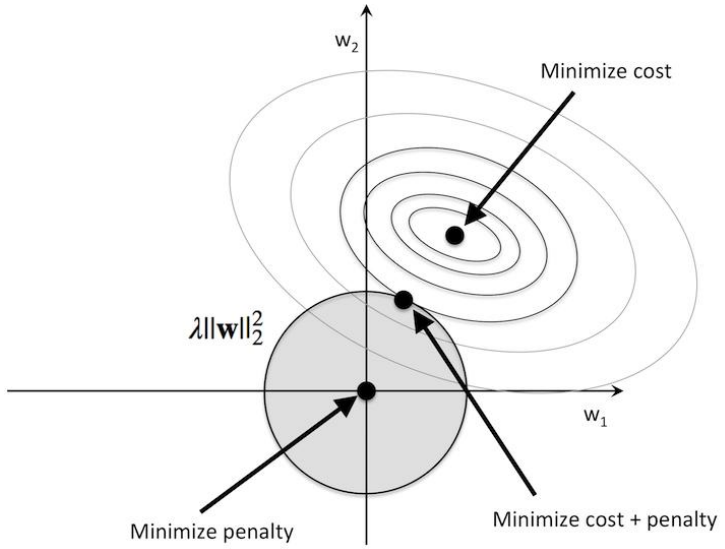


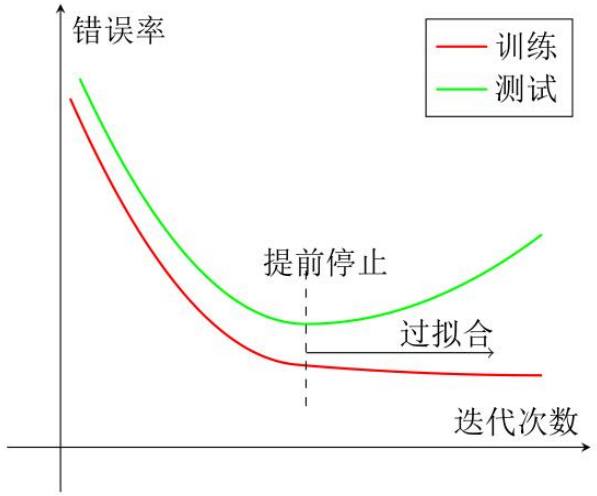
Zhang C, Bengio S, Hardt M, et al. Understanding deep learning requires rethinking generalization[J]. arXiv preprint arXiv:1611.03530, 2016.

正则化（regularization）

所有损害优化的方法都是正则化。

- 增加优化约束：L1/L2约束、数据增强
- 干扰优化过程：权重衰减、随机梯度下降、提前停止



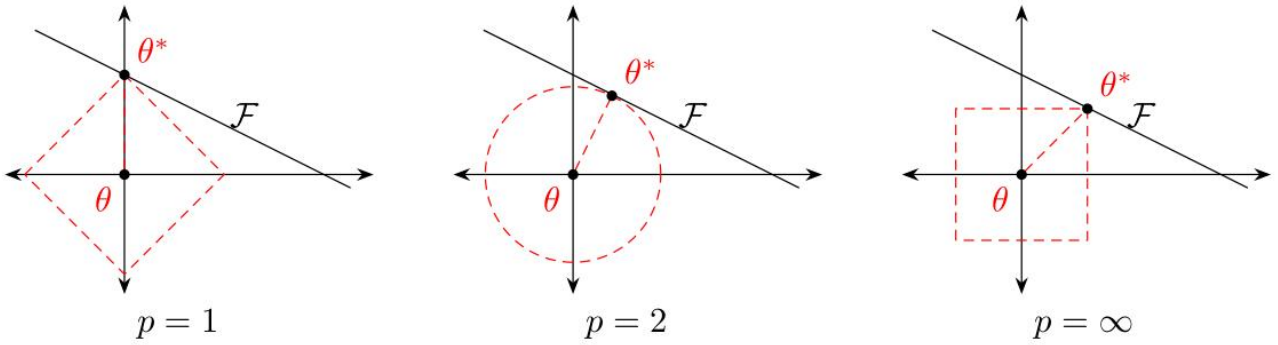


ℓ_1和ℓ_2正则化

- 优化问题可以写为

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}\big(y^{(n)}, f(\mathbf{x}^{(n)}, \theta)\big) + \lambda \ell_p(\theta)$$

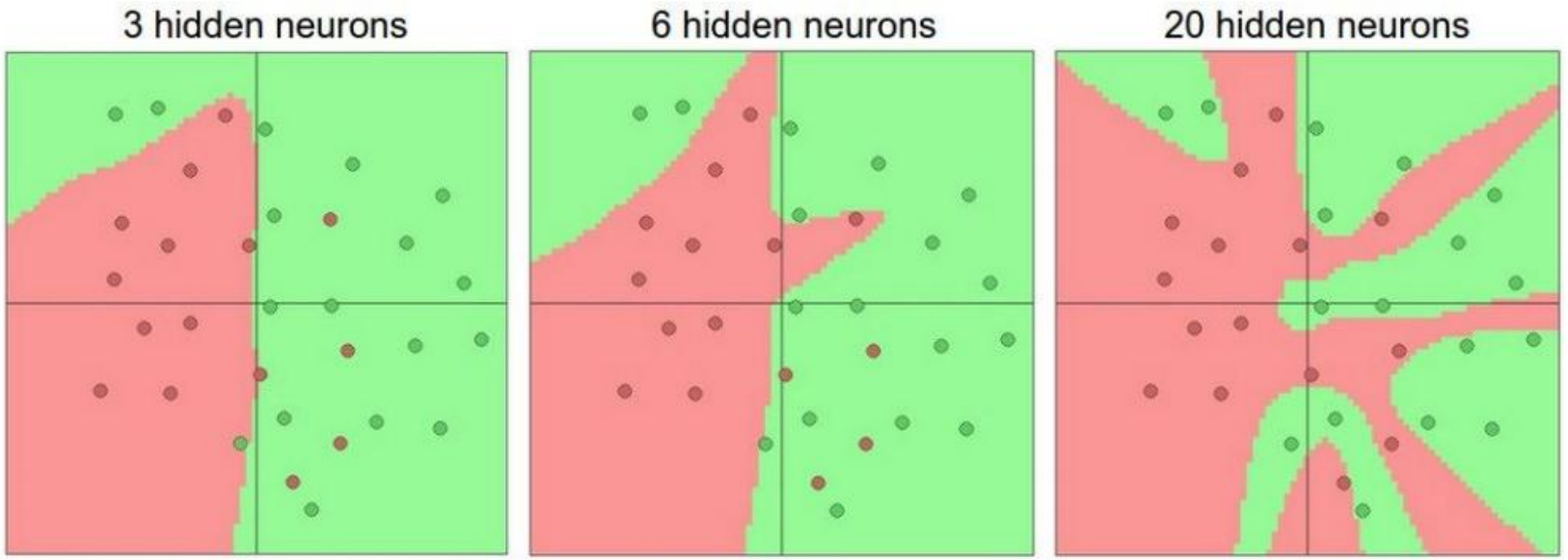
- p的取值通常为{1,2}代表ℓ_1和ℓ_2范数，λ为正则化系数



神经网络示例

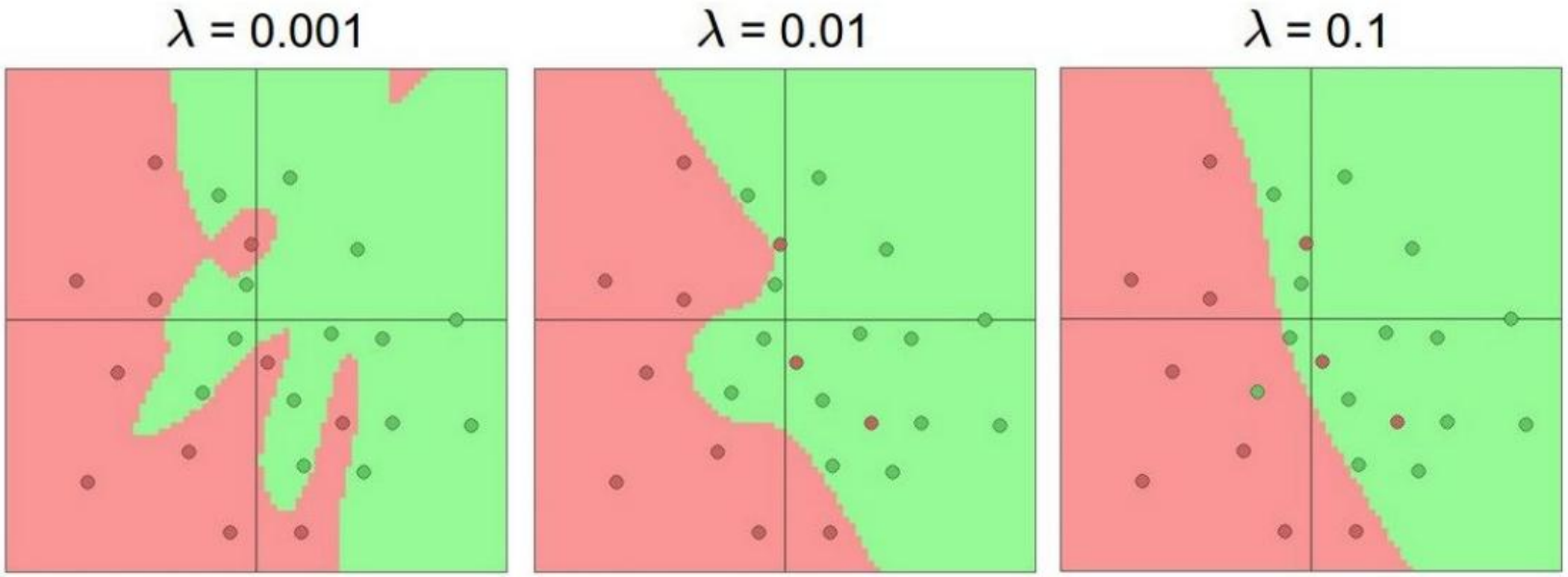
http://playground.tensorflow.org/

- 隐藏层的不同神经元个数



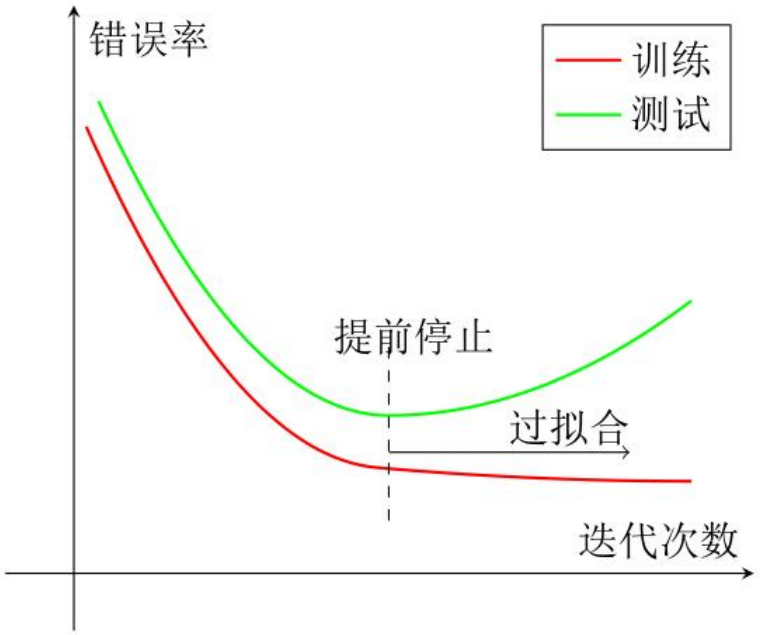
神经网络示例

- 不同的正则化系数



提前停止

- 使用一验证集（Validation Dataset）来测试每一次迭代的参数在验证集上是否最优。如果在验证集上的错误率不再下降，就停止迭代



权重衰减（Weight Decay）

- 在每次参数更新时，引入一个衰减系数w

$$\theta_t \leftarrow (1-w)\theta_{t-1} - \alpha \mathbf{g}_t$$

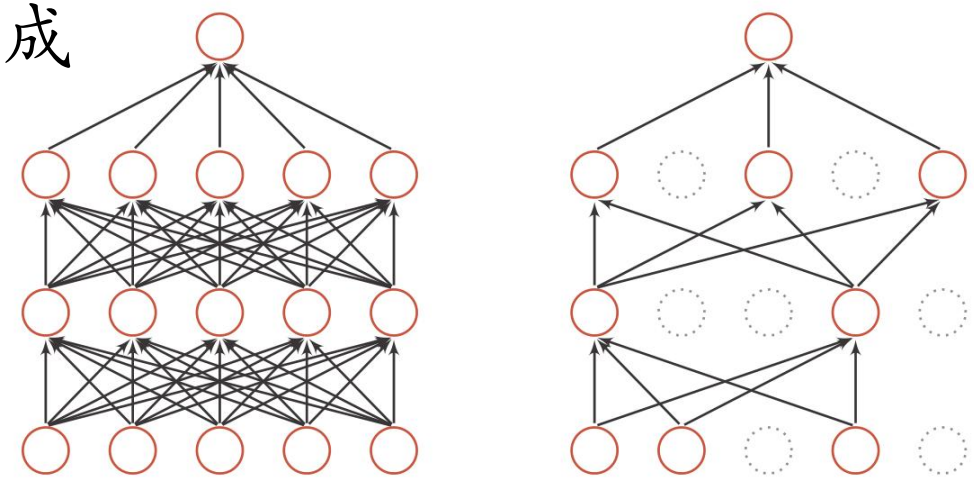
- 在标准的随机梯度下降中，权重衰减正则化和ℓ_2正则化的效果相同
- 在较为复杂的优化方法（比如Adam）中，权重衰减和ℓ_2正则化并不等价

丢弃法（Dropout Method）

- 对于一个神经层$y = f(Wx + b)$，引入一个丢弃函数$d(\cdot)$使得$y = f(Wd(x) + b)$

$$d(\mathbf{x}) = \begin{cases} \mathbf{m} \odot \mathbf{x} & \text{当训练阶段时} \\ p\mathbf{x} & \text{当测试阶段时} \end{cases}$$

- 其中$m \in \{0,1\}^d$是丢弃掩码（dropout mask），通过以概率为p的贝努力分布随机生成



(a) 标准网络　　(b) Dropout后的网络

Dropout意义

- 集成学习的解释
  - 每做一次丢弃，相当于从原始的网络中采样得到一个子网络。如果一个神经网络有n个神经元，那么总共可以采样出2n个子网络

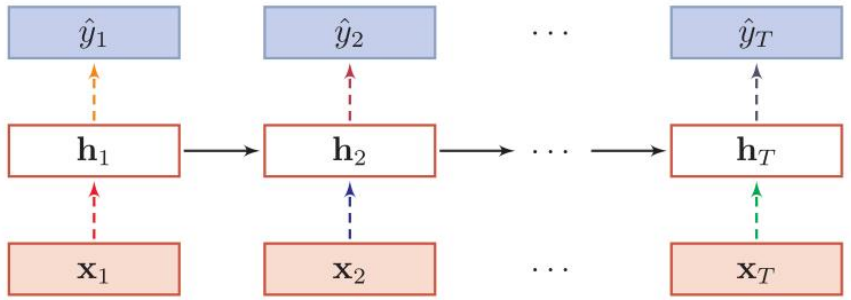
- 贝叶斯学习的解释

$$\mathbb{E}_{q(\theta)}[y] = \int_q f(\mathbf{x}, \theta) q(\theta) d\theta$$
$$\approx \frac{1}{M} \sum_{m=1}^{M} f(\mathbf{x}, \theta_m),$$

  - 其中f(x,θm)为第m次应用丢弃方法后的网络

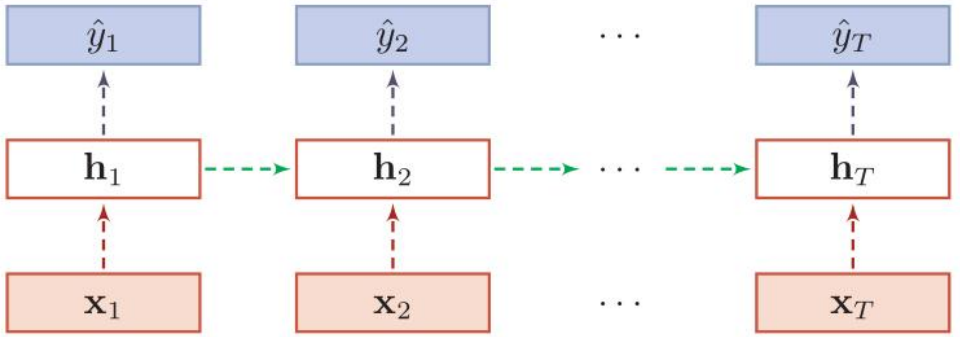
循环神经网络上的丢弃法

- 当在循环神经网络上应用丢弃法，不能直接对每个时刻的隐状态进行随机丢弃，这样会损害循环网络在时间维度上记忆能力



虚线边表示进行随机丢弃，不同的颜色表示不同的丢弃掩码

- 变分Dropout
  - 根据贝叶斯学习的解释，丢弃法是一种对参数θ的采样
    - 每次采样的参数需要在每个时刻保持不变。因此，在对循环神经网络上使用丢弃法时，需要对参数矩阵的每个元素进行随机丢弃，并在所有时刻都使用相同的丢弃掩码



相同颜色表示使用相同的丢弃掩码

数据增强（Data Augmentation）

- 图像数据的增强主要是通过算法对图像进行转变，引入噪声等方法来增加数据的多样性
  - 旋转（Rotation）：将图像按顺时针或逆时针方向随机旋转一定角度
  - 翻转（Flip）：将图像沿水平或垂直方法随机翻转一定角度
  - 缩放（Zoom In/Out）：将图像放大或缩小一定比例
  - 平移（Shift）：将图像沿水平或垂直方法平移一定步长
  - 加噪声（Noise）：加入随机噪声

标签平滑（Label Smoothing）

- 在输出标签中添加噪声来避免模型过拟合
- 一个样本x的标签一般用onehot向量表示

$$\mathbf{y} = [0, \cdots, 0, 1, 0, \cdots, 0]^{\mathrm{T}}$$

硬目标（Hard Targets）

- 引入一个噪声对标签进行平滑，即假设样本以ϵ的概率为其它类。平滑后的标签为

$$\tilde{\mathbf{y}} = [\frac{\epsilon}{K-1}, \cdots, \frac{\epsilon}{K-1}, 1-\epsilon, \frac{\epsilon}{K-1}, \cdots, \frac{\epsilon}{K-1}]^{\mathrm{T}}$$

总结

- 模型
  - 用 ReLU 作为激活函数
  - 分类时用交叉熵作为损失函数
  - 逐层归一化
- 优化
  - SDG+mini-batch（动态学习率、Adam算法优先）
  - 每次迭代都重新随机排序
  - 数据预处理（标准归一化）
  - 参数初始化
- 正则化
  - ℓ_1和ℓ_2正则化（跳过前几轮）
  - Dropout
  - Early-stop
  - 数据增强