《神经网络与深度学习》

网络优化与正则化

周晓飞
自动化学院

将神经网络应用到机器学习时，存在两大类难点：

- **优化问题（训练）**
  - 损失函数是非凸函数
  - 参数巨多，需要的数据量较大
  - 存在梯度消失或爆炸

- **泛化问题（测试）**
  - 复杂度高，易过拟合
  - 需要正则化改进网络泛化能力

本章主要内容包括：

- **网络优化**
  - 优化算法
  - 参数初始化
  - 数据预处理
  - 逐层归一化
  - 超参数优化
- **网络正则化**
  - l1/l2正则化
  - 权重衰减
  - 提前停止
  - 丢弃法
  - 数据增强和标签平滑

1 网络优化

1 网络优化

- 网络优化目标
  - 寻找一个神经网络使得经验（或结构）风险最小化的过程

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

- 风险函数最小化是一个非凸优化问题
- 深度神经网络存在梯度消失问题

1 网络结构多样性 & 高维空间的非凸优化

- 结构多样性
  - 很难找到一种通用的优化算法
  - 超参数一般较多

- 非凸优化问题
  - 低维空间的非凸优化的难点在于参数初始化逃离局部最优
  - 神经网络的参数学习是在非常高维空间中的非凸优化问题

1 高维空间的非凸优化问题

- 鞍点（Saddle point）
  - 高位空间非凸优化不在于逃离局部最优点，而在于如何逃离**鞍点**，
  - 鞍点梯度为0，在一些维度上是最高点，在另外一些维度上是最低点；且高维空间中，大部分梯度为0的点（驻点）都是鞍点
  - 鞍点附近上升或下降；局部最小/大点附近单调
  - 基于梯度下降的优化方法会在鞍点附近接近停滞，很难从鞍点离开，因此，基于梯度下降的优化算法易陷入鞍点，引入随机性（随机梯度下降），可有效逃离鞍点

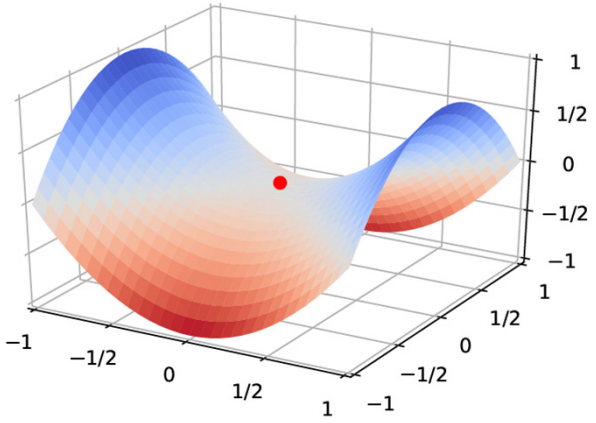


图 7.1 鞍点示例

1 高维空间的非凸优化问题

- 平坦最小值（Falt minima）
  - 参数多，存在冗余性，每个参
    数在局部最小解附近通常是一
  - 平坦最小值邻域内，损失值接
  - 平坦最小值与模型泛化能力有
    ，鲁棒性好，而当模型收敛至

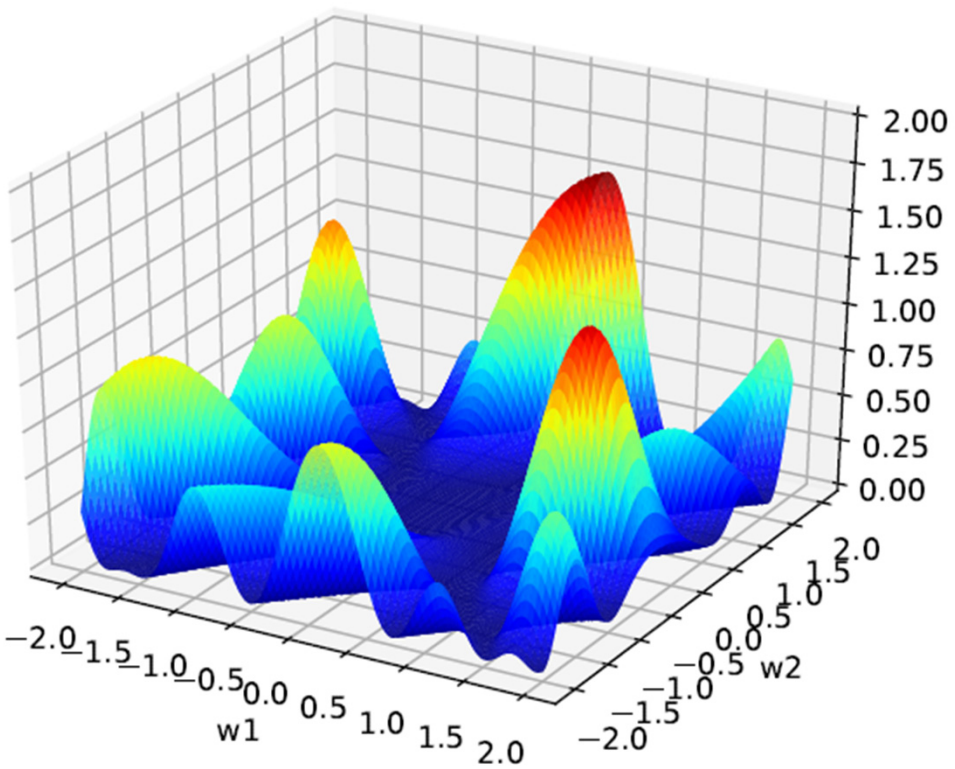
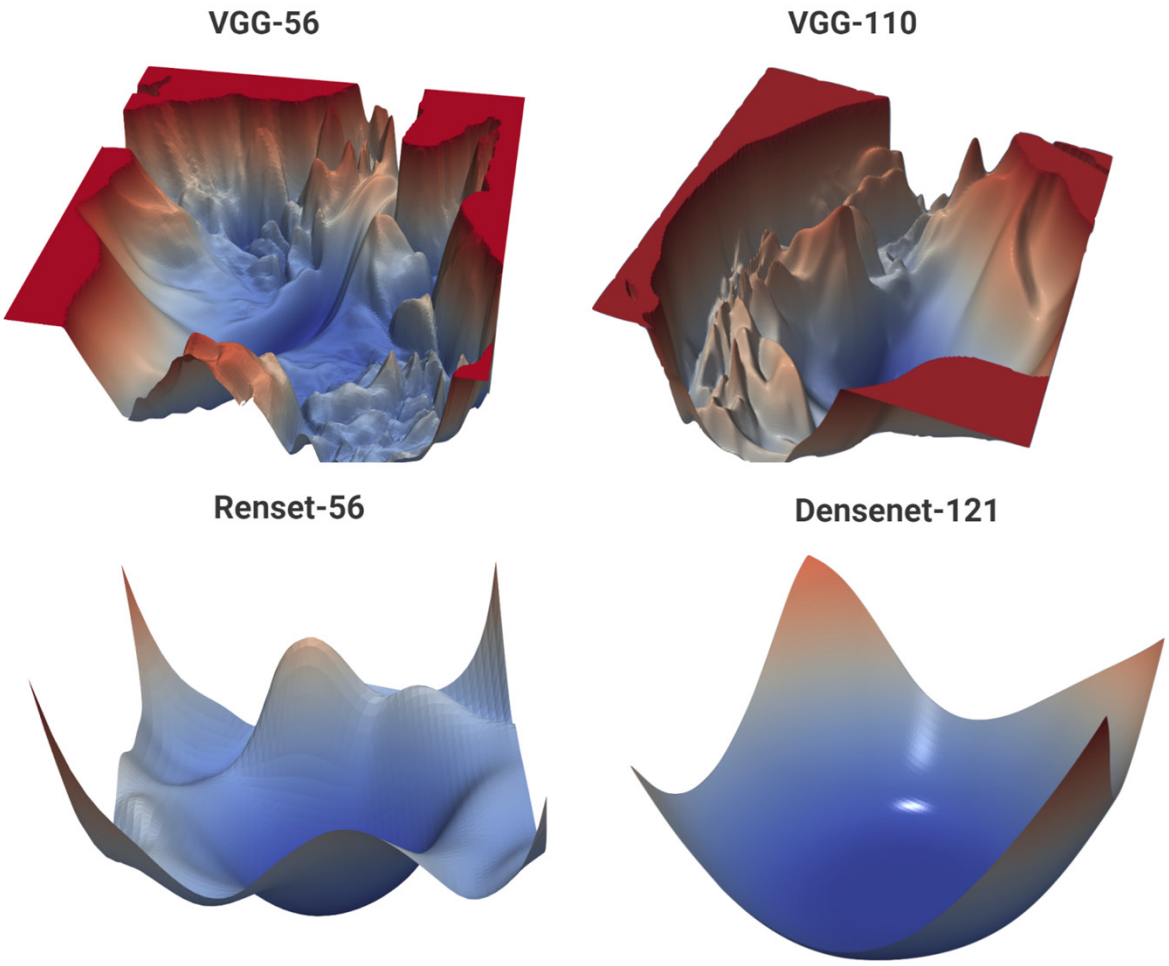


图 7.2 神经网络中的平坦底部示例

1 网络结构多样性 & 高维空间的非凸优化

- 局部最小解的等价性
  - 大部分神经网络对应的局部最小解是等价的，在测试集上性能相近
  - 局部最小解对应的损失值接近全局最小解的损失值
  - 在训练神经网络时，无需寻找全局最小解，此可能导致过拟合

VISUALIZING THE LOSS LANDSCAPE OF NEURAL NETS



without skip connections

with skip connections

2 神经网络优化的改善方法

- 目标：找到较好的局部最小值和提高优化效率
  - 有效的优化算法，以此提高梯度下降优化算法效率和稳定性，如动态学习率调整
  - 更好地参数初始化方法和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，如使用ReLU激活函数、残差连接、逐层归一化等
  - 更好的超参数优化方法

2 优化算法

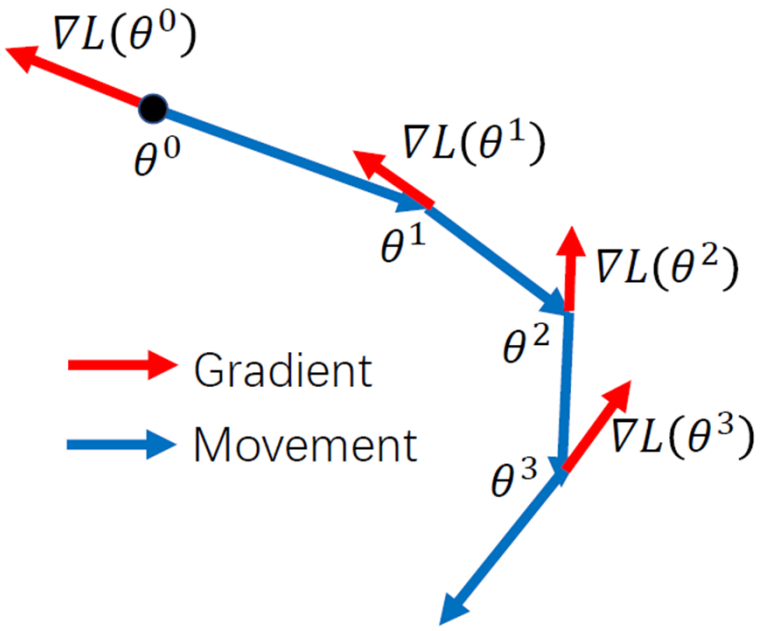
梯度下降法可分为：

- 批量梯度下降
- 随机梯度下降
- 小批量梯度下降

常用优化算法分为两类：

- 调整学习率，使得优化更稳定
- 梯度估计修正，优化训练速度

批量梯度下降:



从起始位置 θ^0 开始，在 θ^0 处计算梯度，并更新

$$\theta^1 = \theta^0 - \nabla L\left(\theta^0\right)$$

在 θ^1 处计算梯度，并更新参数

$$\theta^2 = \theta^1 - \nabla L\left(\theta^1\right)$$

.

.

停止迭代，直至 $\nabla L\left(\theta^t\right) = 0$

$$\nabla L(\theta) = \frac{1}{N}\sum_{n=1}^{N}\frac{\partial L\left(y^{(n)}, f\left(x^{(n)}, \theta\right)\right)}{\partial \theta}$$

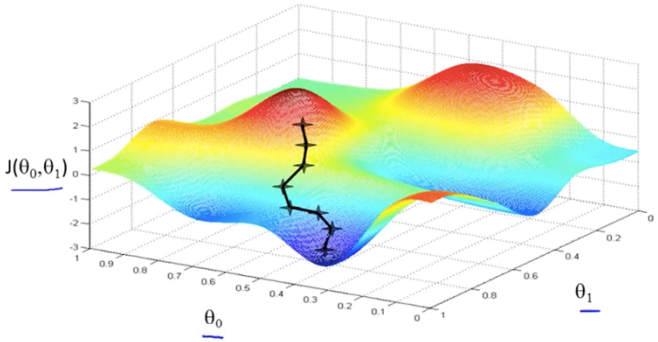
批量梯度下降：每次更新都是用整个训练集数据；梯度方差小；需要较多计算资源

随机梯度下降

算法 2.1: 随机梯度下降法

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

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



每次只取一个样本计算梯度，更新参数

1 小批量随机梯度下降 MiniBatch

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

- 定义梯度 $$\mathfrak{g}_t(\theta) = \frac{1}{K} \sum_{(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}) \in \mathcal{I}_t} \frac{\partial \mathcal{L}\left(\mathbf{y}^{(k)}, f(\mathbf{x}^{(k)}, \theta)\right)}{\partial \theta}$$

- 更新参数 $$\mathbf{g}_t \triangleq \mathfrak{g}_t(\theta_{t-1})$$

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

其中$\alpha > 0$为学习率

几个关键影响因素：
- 小批量样本数量 K
- 梯度 gt
- 学习率 a

梯度下降算法比较

批量梯度下降:

- 可寻找全局最优解，梯度方差小
- 样本数目多时，训练变慢

随机梯度下降:

- 训练速度快，准确度下降
- 非全局最优，梯度方差大

小批量梯度下降:

- 同时兼顾上述两方法优点

2 批量大小选择

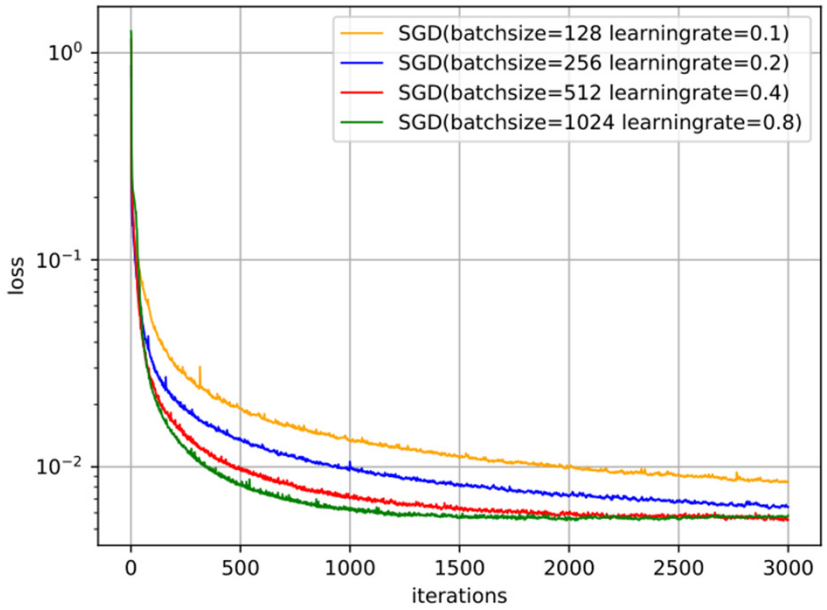
- 批量大小 Batch Size
  - 不影响随机梯度的期望，但影响随机梯度的方差
  - 批量越大，随机梯度的方差越小，可以设置较大的学习率
  - 批量越小，需要设置较小的学习率，否则模型不收敛

  - 线性缩放规则：批量大小增加m倍，则学习率也增加m倍
  - 回合 epoch 所有训练样本更新一遍
  - 迭代 iteration 每一批次小批量更新一次

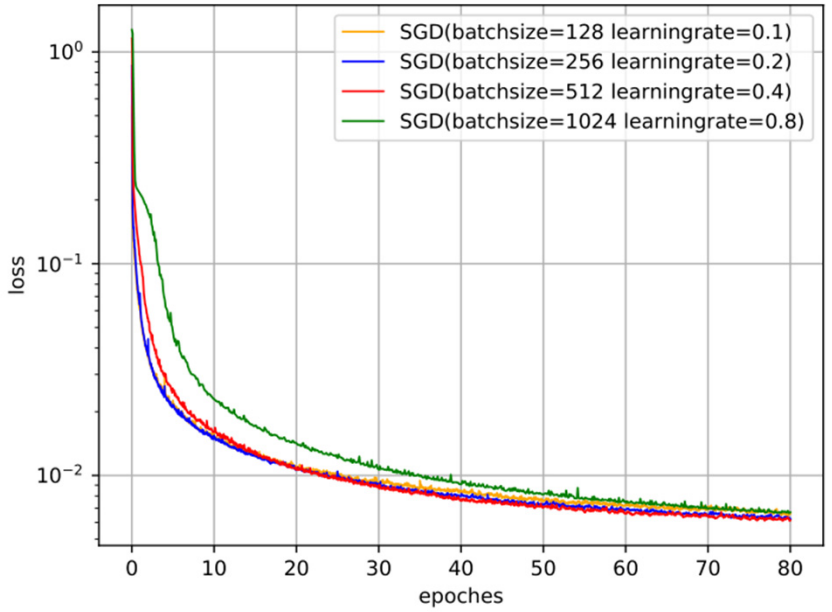
$$1\text{个回合（Epoch）：迭代}(\text{Iteration})=\left(\frac{\text{训练样本的数量N}}{\text{批量大小K}}\right)$$

- 下图给出了从epoch和iteration角度，探究批量大小对损失下降的影响

2 批量大小选择



(a) 按 Iteration 的损失变化



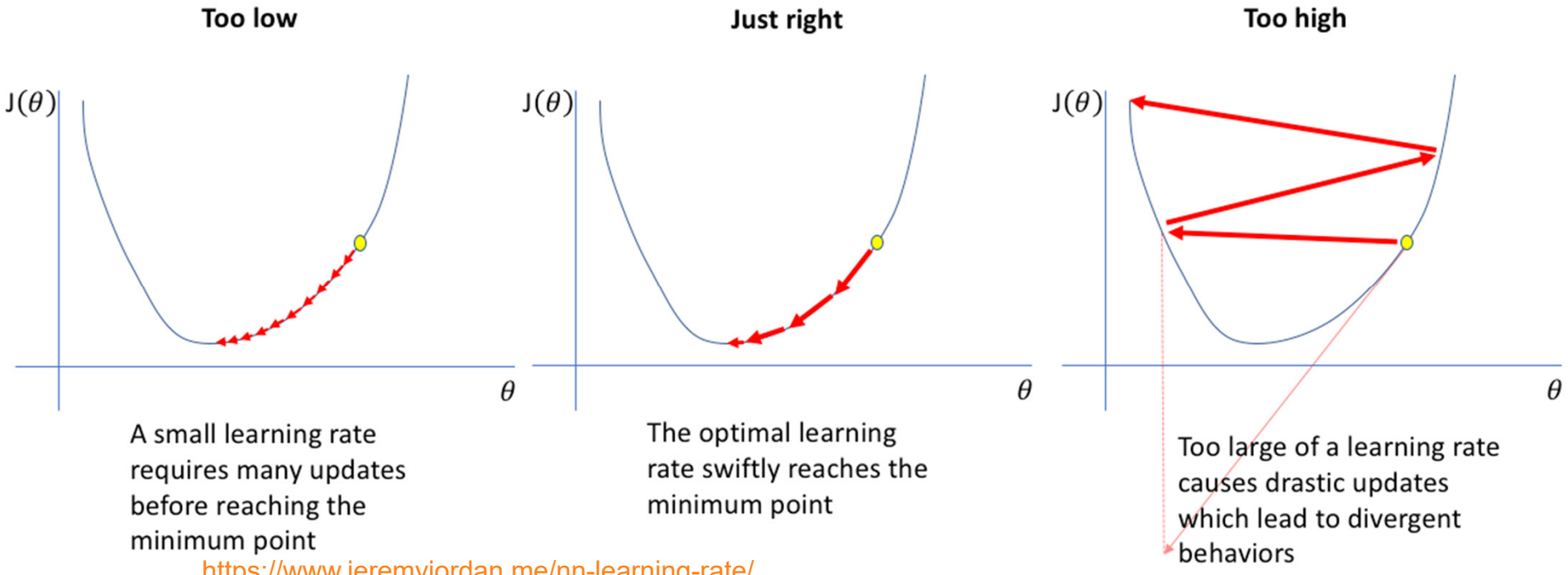
(b) 按 Epoch 的损失变化

小批量梯度下降中，每次选取样本数量对损失下降的影响。

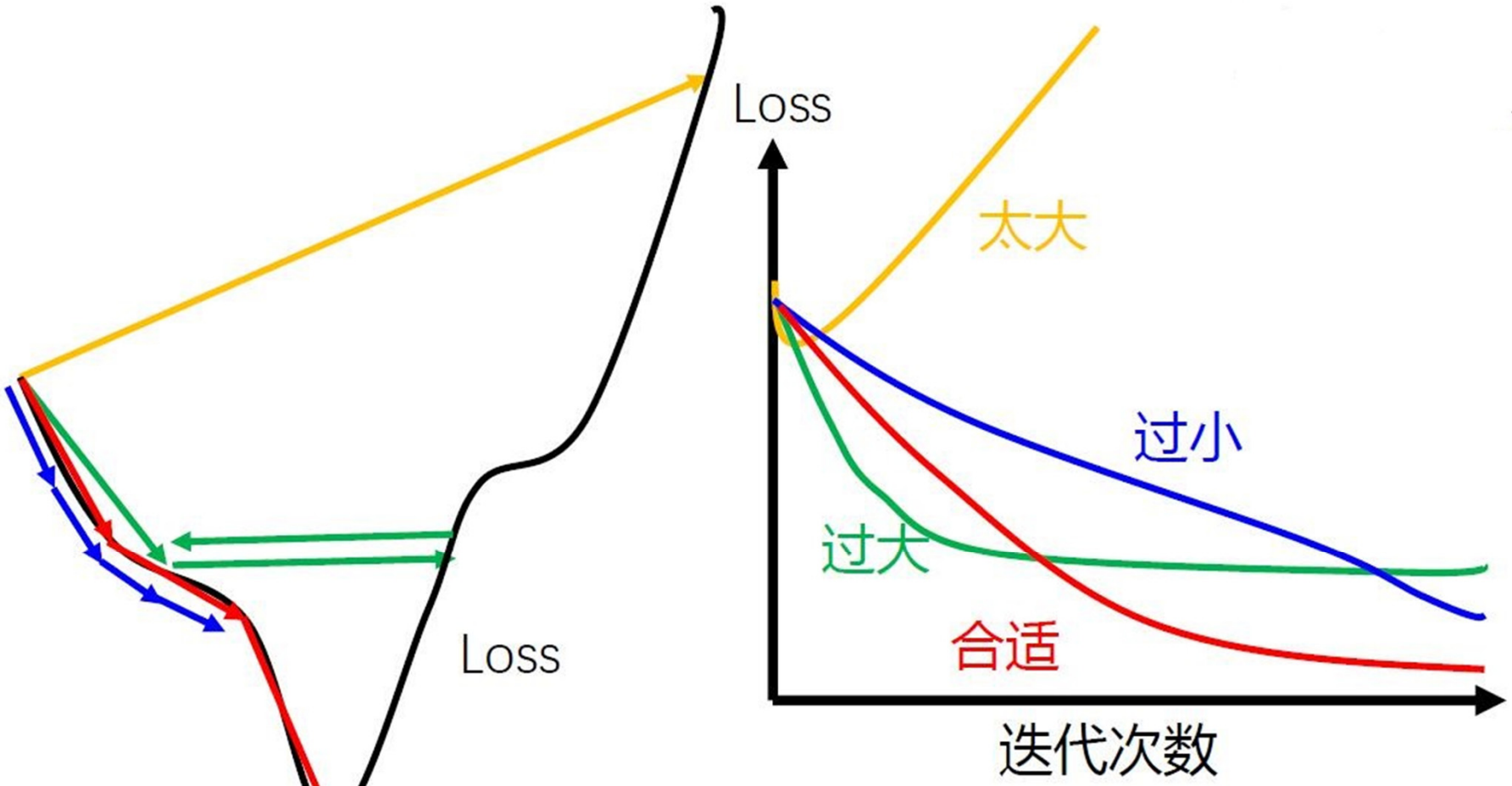
3 学习率调整

$$\theta^{t} = \theta^{t-1} - \alpha g^{t}$$

- ✓ 学习率过大，不收敛
- ✓ 学习率过小，收敛太慢



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


Loss
太大
过小
过大
合适
迭代次数
Loss

3 学习率调整---学习率衰减

- 训练开始时，学习率大些保证收敛速度；多次迭代后，收敛到最优值附近时，要小些来避免震荡
  - 假设初始化学习率为 α_0，在第t次迭代时的学习率为 α_t。常见的衰减方法有以下几种：

1）分段常数衰减（阶梯衰减 step decay），即每经过 $T_1,...,T_m$ 次迭代，学习率衰减为原来的 $\beta_1,...,\beta_m<1$ 倍，二者均为超参数

2）逆时衰减 Inverse time decay，β 是衰减率 $\alpha_t=\alpha_0\frac{1}{1+\beta\times t}$

3）指数衰减 Exponential decay $\alpha_t=\alpha_0\beta^t$

4）自然指数衰减 Natural exponential decay $\alpha_t=\alpha_0\exp(-\beta\times t)$

5）余弦衰减 Cosine decay，T为总迭代次数 $\alpha_t=\frac{1}{2}\alpha_0\left(1+\cos\left(\frac{t\pi}{T}\right)\right)$

3 学习率调整---学习率衰减

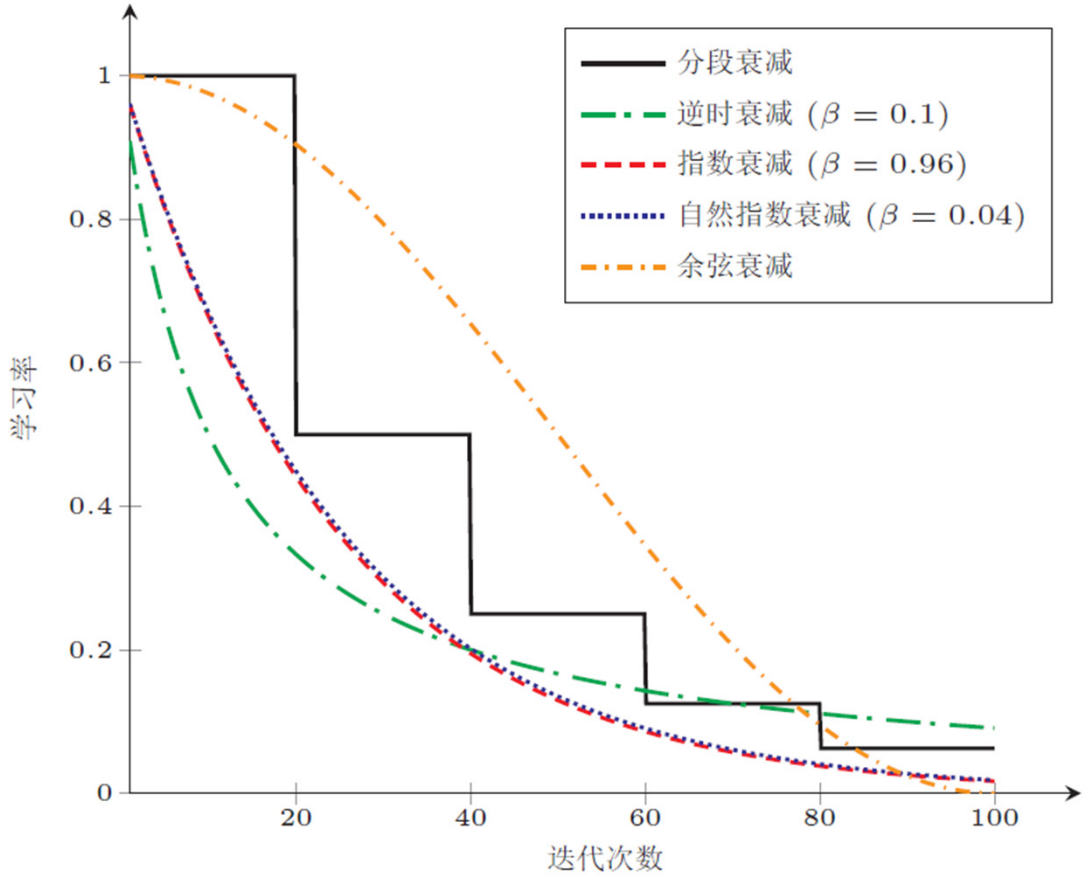


图 7.4 不同学习率衰减方法的比较

3 学习率调整---学习率预热

- 当批量大小的设置比较大时，需要比较大的学习率。但在刚开始训练时，由于参数时随机初始化的，梯度往往比较大，再加上比较大的初始学习率，会使得训练不稳定
- 为提高训练稳定性，在最初几轮迭代时，采用较小的学习率，等到梯度下降到一定程度后再恢复到初始的学习率，这种方法称为**学习率预热**（Learning rate warmup）
  - 常用的学习率预热方法是**逐渐预热**（Gradual warmup），假设预热的迭代次数为 T'，初始学习率为 α_0，在预热过程中，每次更新的学习率为

$$\alpha'_t = \frac{t}{T'}\alpha_0 \quad 1 \le t \le T'$$

学习率逐渐提高

  - 当预热结束，再选择一种学习率衰减方法来逐渐降低学习率

3 学习率调整---周期性学习率调整

- 为了使得梯度下降法能够逃离鞍点或尖锐最小值，可以在训练过程中周期性的增大学习率。
- 短期看周期性的增大学习率会损害优化过程，使得网络收敛的稳定性变差，但长期来看有助于找到更好的局部最优解
- 这里介绍两种常见的周期性调整学习率的方法：**循环学习率和带热重启随机梯度下降**

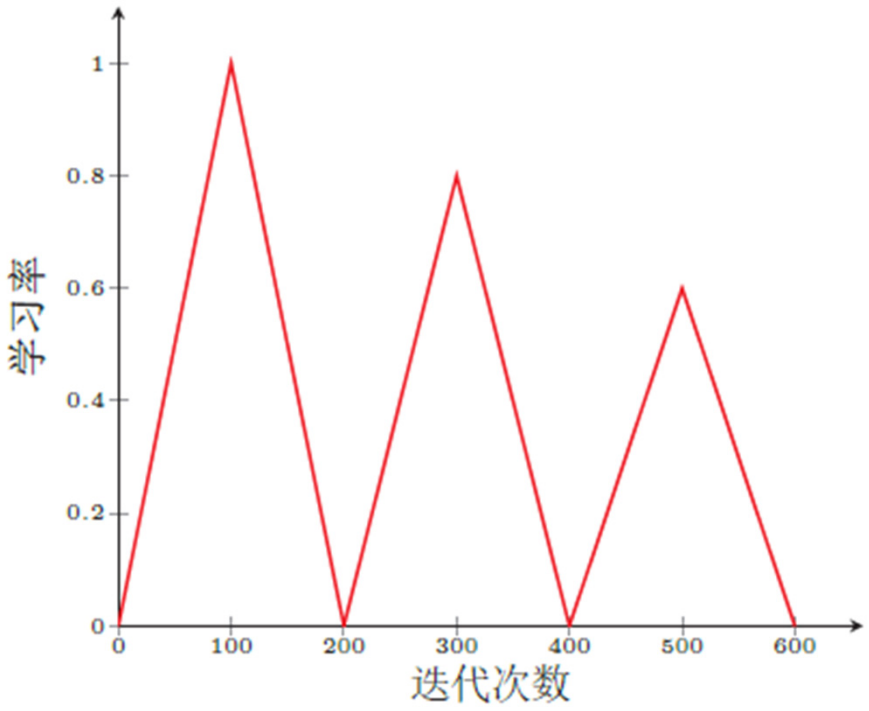
3 学习率调整---周期性学习率调整

- 循环学习率（Cyclic learning rate）
  - 让学习率在一个区间内周期性地增大或缩小
  - 三角循环学习率，即通过线性缩放来调整学习率

$$m=\left\lfloor 1+\frac{t}{2\Delta T}\right\rfloor, b=\left|\frac{t}{\Delta T}-2m+1\right|$$

$$\alpha_t=\alpha_{min}^m+\left(\alpha_{max}^m-\alpha_{min}^m\right)\left(\max\left(0,1-b\right)\right)$$

  - m为循环周期数, b介于[0,1]



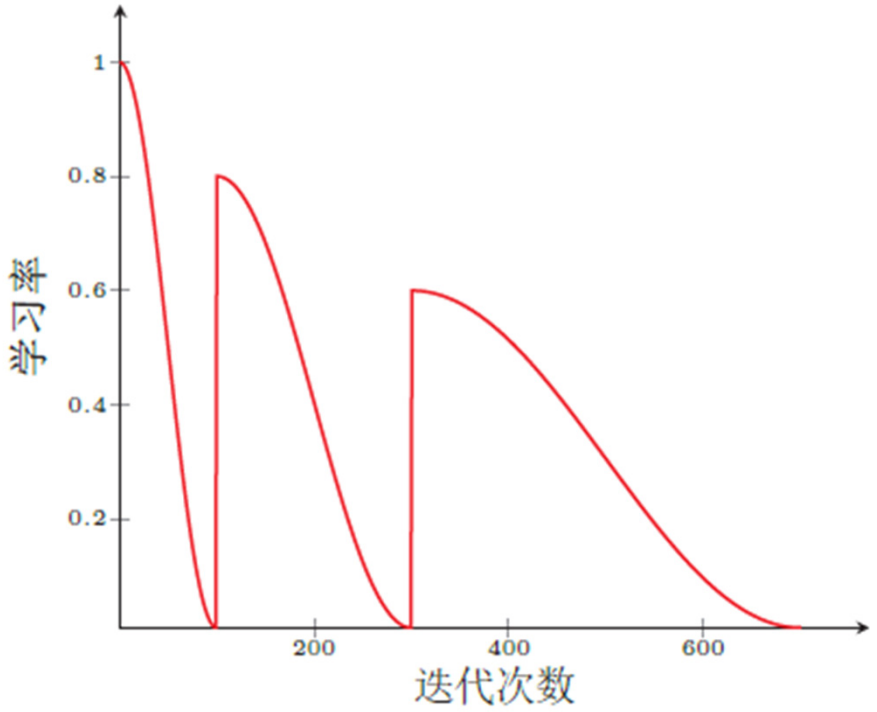
(a) 三角循环学习率

3 学习率调整---周期性学习率调整

- 带热重启的随机梯度下降 Stochastic gradient descent with warm restarts, SGDR
  - 用热重启代替学习率衰减
  - 每次重启后模型参数不是从头开始优化，而是从重启前的参数基础上继续优化
  - 假设梯度下降重启M次，第m次重启在上次重启开始第Tm个回合后进行，Tm称为重启周期。第m次重启之前，采用余弦衰减来降低学习率，第t次迭代的学习率为

$$\alpha_t = \alpha_{\min}^m + \frac{1}{2}\left(\alpha_{\max}^m - \alpha_{\min}^m\right)\left(1+\cos\left(\frac{T_{cur}}{T_m}\pi\right)\right)$$

  - Tcur为从上次重启之后的回合数



(b) 带热重启的余弦衰减

3 学习率调整---AdaGrad算法 自适应学习率

- AdaGrad算法 Adaptive gradient algorithm
  - 学习率衰减的局限：每个参数的维度上收敛速度都不相同，应根据不同参数的收敛情况分别设置学习率
  - AdaGrad借鉴了l2正则化的思想，每次迭代自适应地调整每个参数的学习率。第t次迭代时，先计算每个参数的梯度平方的累计值：

$$G_t = \sum_{\tau=1}^{t} \mathbf{g}_\tau \odot \mathbf{g}_\tau$$

⊙ 按元素进行操作

ε 保持数值稳定设置的非常小的数

  - 参数更新差值为

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

  - 缺点：在经过一定次数的迭代后，依然没有找到最优点时，由于这时的学习率非常小了，很难再继续找到最优点

3 学习率调整---RMSprop算法

- RMSprop算法
  - 由Geoff Hinton提出的一种自适应学习率方法，可在一定程度上避免AdaGrad算法中学习率不断单调下降以致于过早衰减的缺点
  - 首先计算每次迭代梯度 g_t 平方的指数衰减移动平均

$$G_t = \beta G_{t-1} + (1-\beta)\mathbf{g}_t \odot \mathbf{g}_t = (1-\beta)\sum_{\tau=1}^{t}\beta^{t-\tau}\mathbf{g}_t \odot \mathbf{g}_t$$

β 是衰减率，一般取值0.9

  - 参数更新差值为

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{G_t + \varepsilon}} e \ g_t$$

3 自适应学习率衰减

- Adagrad
- RMSprop
- Adadelta

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

$$G_t = \sum_{\tau=1}^{t} \boldsymbol{g}_\tau \odot \boldsymbol{g}_\tau$$

$$G_t = \beta G_{t-1} + (1-\beta)\boldsymbol{g}_t \odot \boldsymbol{g}_t$$

$$= (1-\beta)\sum_{\tau=1}^{t} \beta^{t-\tau} \boldsymbol{g}_\tau \odot \boldsymbol{g}_\tau$$

$$\Delta\theta_t = -\frac{\sqrt{\Delta X_{t-1}^2 + \epsilon}}{\sqrt{G_t + \epsilon}} \mathbf{g}_t$$

$$\Delta X_{t-1}^2 = \beta_1 \Delta X_{t-2}^2 + (1-\beta_1)\Delta\theta_{t-1} \odot \Delta\theta_{t-1}$$

4 梯度估计修正--- 动量法

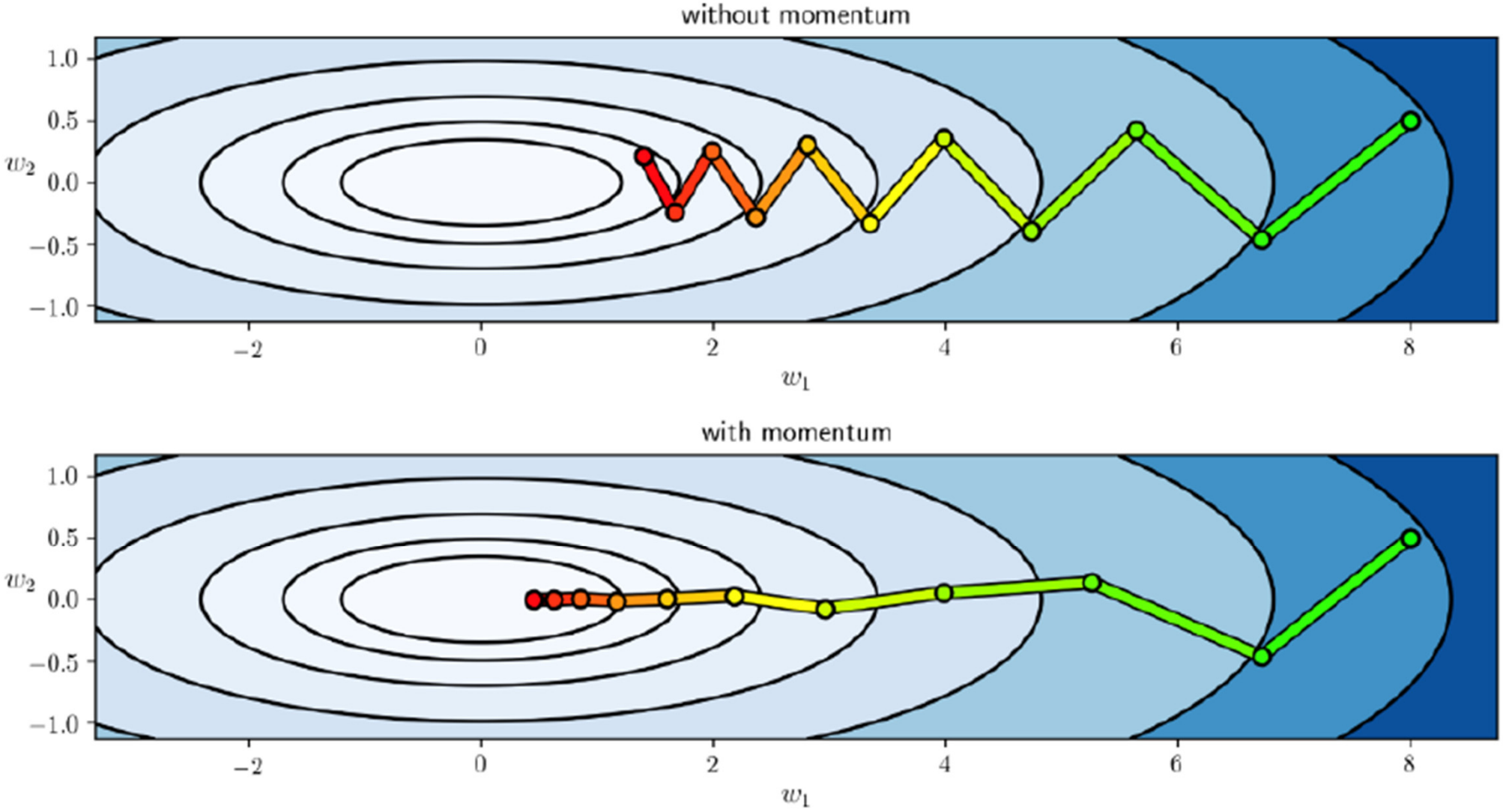
- 动量法（Momentum Method）
  - 随机（小批量）梯度下降法中，若每次选取样本数量比较小，损失则会呈现振荡的方式下降。为缓解梯度估计的随机性，可用最近一段时间的平均梯度来代替当前时刻的随机梯度来作为参数更新的方向
  - **动量法**：用之前积累动量来替代真正的梯度。每次迭代的梯度可以看作是加速度。在第t次迭代时，计算负梯度的加权移动平均作为参数的更新方向

$$\Delta\theta_t = \rho\Delta\theta_{t-1} - \alpha g_t = -\alpha\sum_{\tau=1}^{t}\rho^{t-\tau}g_\tau$$

ρ 动量因子，0.9

  - 如此，各参数的实际更新差值取决于最近一段时间内的加权平均值；
  - 一般而言，迭代初期，梯度方向较一致，动量法起到加速作用，加速收敛；迭代后期，梯度方向差异略大，会在收敛值附近震荡，动量法减速，起稳定作用

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


without momentum
1.0
0.5
0.0
−0.5
−1.0
w_2
−2
0
2
4
6
8
w_1
with momentum
1.0
0.5
0.0
−0.5
−1.0
w_2
−2
0
2
4
6
8
w_1

4 梯度估计修正--- Nesterov加速梯度

- Nesterov加速梯度（Nesterov Accelerated Gradient,NAG）
  - 是对动量法的改进，也称为Nesterov 动量法
  - 先用上一步的参数更新方向更新一次，再结合当前梯度更新一次

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

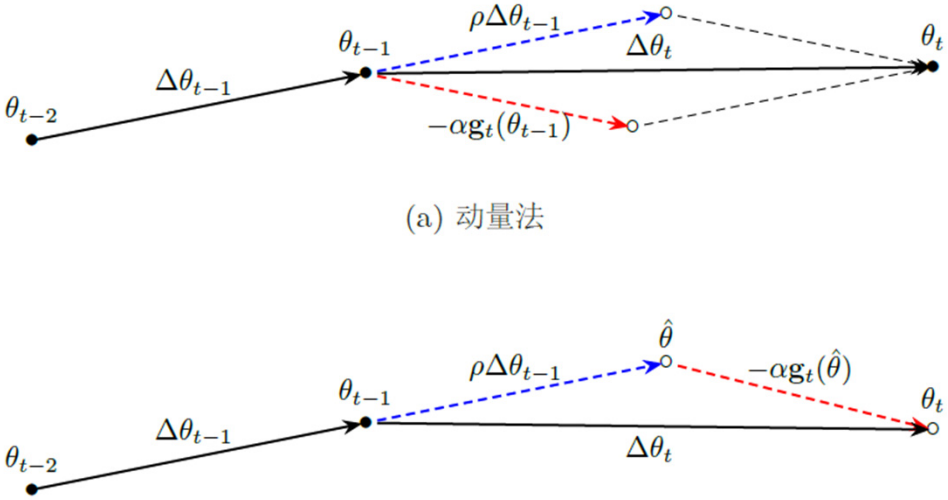
$$\therefore \theta_t = \theta_{t-1} + \Delta\theta_t$$

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

$$\therefore \hat{\theta} = \theta_{t-1} + \rho\Delta\theta_{t-1}$$

$$\therefore \theta_t = \hat{\theta} - \alpha \mathbf{g}_t(\theta_{t-1})$$

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



(a) 动量法

(b) Nesterov 加速梯度

图 7.6 动量法和 Nesterov 加速梯度的比较

4 梯度估计修正--- Adam算法

- Adam算法≈**动量法+RMSprop**
  - 先计算两个移动平均（gt指数加权平均 & gt平方的指数加权平均）

$$M_t = \beta_1 M_{t-1} + (1-\beta_1) g_t$$ 梯度指数加权平均

$$G_t = \beta_2 G_{t-1} + (1-\beta_2) g_t \odot g_t$$ 梯度平方指数加权平均

  - 偏差修正

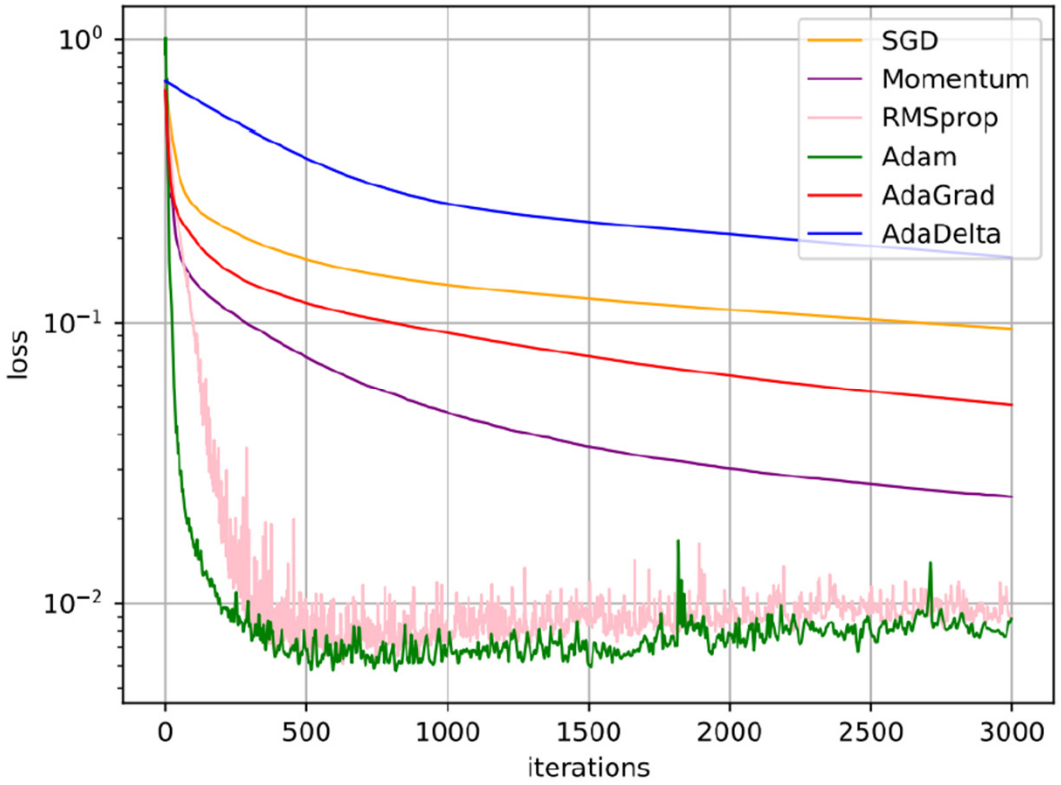
$$\hat{M}_t = \frac{M_t}{1-\beta_1^t} \quad \hat{G}_t = \frac{G_t}{1-\beta_2^t}$$

  - 更新

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

4 梯度估计修正-各优化算法比较

- 在MINIST数据集上收敛性的比较



- ✓ RMSProp和Adam表现较好
- ✓ 目前较流行的优化算法：SGD、具有动量的SGD、RMSProp、具有动量的RMSProp和Adam

4 梯度估计修正--- 梯度截断

- 神经网络中，除了梯度消失之外，梯度爆炸是影响学习效率的主要因素。
- 梯度下降优化算法中，大梯度的参数更新会导致其偏离最优点。为此，当梯度的模大于一定阈值时，就对梯度进行截断，称为梯度截断（Gradient Clipping）
  - 下图为循环神经网络的损失函数关于参数的曲面。可以看到梯度值（L关于w和b的梯度）会在某个区域突然变大

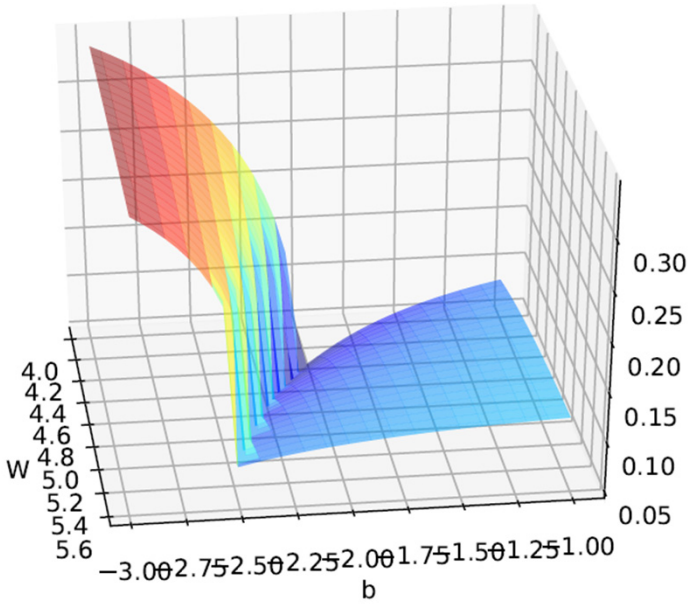


图 7.7 梯度爆炸问题示例

4 梯度估计修正--- 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

$$\mathbf{g}_t = \max\left(\min\left(\mathbf{g}_t, \mathrm{b}\right), \mathrm{a}\right)$$

  - 按模截断，即将梯度的模截断到一个给定的截断阈值b

$$\mathbf{g}_t = \begin{cases} \mathbf{g}_t & \|\mathbf{g}_t\|^2 < \mathrm{b} \\ \dfrac{\mathrm{b}}{\|\mathbf{g}_t\|}\mathbf{g}_t & \|\mathbf{g}_t\|^2 > \mathrm{b} \end{cases}$$

阈值b是超参数，可根据一段时间内的平均梯度来自动调整

5 小结

- 优化算法可以分为两类：
  - 调整学习率，使得优化更稳定
  - 梯度估计修正，优化训练速度

- 大部分优化算法可以使用下面公式来统一描述概括：

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

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

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

g_t 为第t步的梯度

α_t 为第t步的学习率

5 小结

- 神经网络常用优化方法的汇总

表 7.1 神经网络常用优化方法的汇总

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

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

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

实际更新方向　　梯度方向

- 学习率
  - 学习率衰减
  - 学习率预热
  - 周期性学习率
  - 自适应：Adagrad、Adadelta、RMSprop

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

- 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（抑制梯度爆炸）

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

Adam

Adam is better choice!

- 综合
  - Adam

3 参数初始化

- **预训练初始化** pre-trained initialization
  - 不同参数初始值会收敛到不同的局部最优解
  - 一个已在大规模数据集上训练过的模型可以提供一个良好的参数初始值
  - 预训练模型在目标任务上的学习过程称为精调Fine-tuning
- **随机初始化** Random initialization
  - 对称权重（0初始化）：参数为0，神经元输出相同，BP梯度相同，参数更新相同，参数相同
  - 每个参数随机初始化，提高神经元之间的区分性
  - 固定方差的参数初始化、方差缩放的参数初始化、正交初始化
- **固定值初始化**
  - 经验固定值来初始化，如偏置通常用0来初始化

1 基于固定方差的参数初始化

- 从固定均值和方差的分布中采样来生成参数的初始值，有以下两种
- 1）Gaussian分布初始化 $N(0, \sigma^2)$
  - 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。
- 2）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。

x在$[a,b]$内均匀分布，

则$\mathrm{var}(x)=\frac{(b-a)^2}{12}$

x在$[-r,r]$内均匀分布，$\mathrm{var}(x)=\sigma^2$，

则$r=\sqrt{3\sigma^2}$

关键在于如何设置方差。参数取值范围过小，则神经元输出小，多层之后慢慢消失；亦会使sigmoid丢失非线性；参数取值范围过大，易导致sigmoid激活值变得饱和

2 基于方差缩放的参数初始化

- 初始化网络时，为了缓解梯度消失或者梯度爆炸，要尽可能的保持每个神经元的输入和输出的方差一致，根据神经元连接数量来自适应地调整初始化分布的方差，这类方法称为方差缩放（Variance Scaling）
- Xavier初始化，由每层神经元数量自动计算初始化参数方差
  - l层神经元和l-1层神经元关系

$$a^{(l)} = f\left(\sum_{i=1}^{M_{l-1}} w_i^{(l)} a_i^{(l-1)}\right)$$

  - 假设 $w_i^{(l)}$ 和 $a_i^{(l-1)}$ 的均值为0，且相互独立，则 $a^{(l)}$ 的均值、方差为

$$E\left[a^{(l)}\right] = E\left[\sum_{i=1}^{M_{l-1}} w_i^{(l)} a_i^{(l-1)}\right]$$

$$= \sum_{i=1}^{M_{l-1}} E\left[w_i^{(l)}\right] E\left[a_i^{(l-1)}\right] = 0$$

$$\mathrm{var}\left(a^{(l)}\right) = \mathrm{var}\left(\sum_{i=1}^{M_{l-1}} w_i^{(l)} a_i^{(l-1)}\right)$$

$$= \sum_{i=1}^{M_{l-1}} \mathrm{var}\left(w_i^{(l)}\right) \mathrm{var}\left(a_i^{(l-1)}\right)$$

$$= M_{l-1}\, \mathrm{var}\left(w_i^{(l)}\right) \mathrm{var}\left(a_i^{(l-1)}\right)$$

2 基于方差缩放的参数初始化

- 由前述公式推至信输出信号方差被放大或缩小了 $M_{l-1}\,\mathrm{var}\left(w_i^{(l)}\right)$，为了保持输入输出方差一致，令 $\mathrm{var}\left(w_i^{(l)}\right)=\frac{1}{M_{l-1}}$
- 类似，反向传播时，同样令 $\mathrm{var}\left(w_i^{(l)}\right)=\frac{1}{M_l}$
- 折中，取 $\mathrm{var}\left(w_i^{(l)}\right)=\frac{2}{M_l+M_{l-1}}$
- 若采用高斯分布初始化参数，则 $w_i^{(l)}$ 可按 $N\left(0,\frac{2}{M_l+M_{l-1}}\right)$ 初始化
- 若采用均匀分布初始化参数[-r,r]，则 $r=\sqrt{\frac{6}{M_l+M_{l-1}}}$

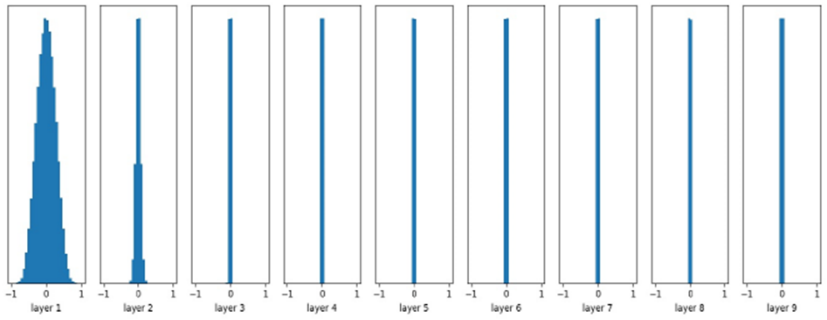
这种根据每层神经元数量来自动计算初始化参数方差的方法称为**Xavier初始化**

2 基于方差缩放的参数初始化

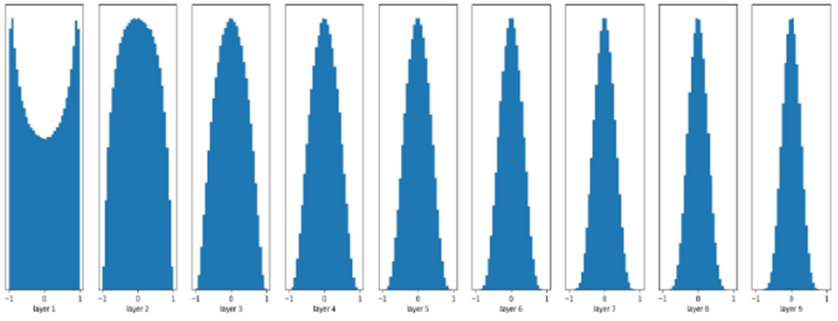
- He初始化
  - 当第l层神经元使用ReLU激活函数时，通常有一半的神经元输出为0，因此其前向输出对应权重方差变为 $\mathrm{var}\left(w_i^{(l)}\right)=\frac{2}{M_{l-1}}$
  - ReLU作为激活函数，采用高斯分布，则方差 $\frac{2}{M_{l-1}}$，若采用[-r,r]均匀分布，则 $r=\sqrt{6/M_{l-1}}$

上述初始化方法就是**He初始化方法**

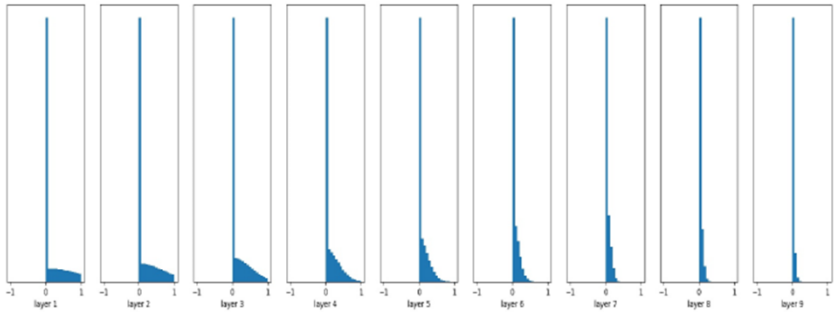
基于方差缩放的参数初始化



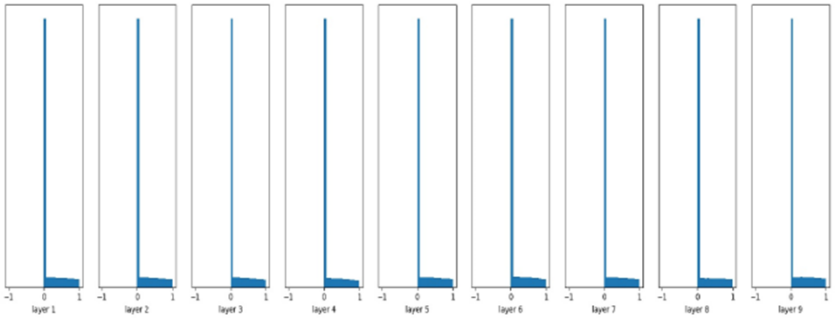
tanh，高斯分布初始化



tanh，Xavier初始化



ReLU，Xavier初始化



ReLU，He初始化

3 正交初始化

- 范数保持性（Norm-Preserving）
  - 根据误差反向传播算法，l-1层的误差项和l层的误差项之间存在如下关系

$$\delta^{(l-1)} = \left(\mathbf{W}^{(l)}\right)^{\mathrm{T}} \delta^{(l)}$$

  - 为了避免梯度消失或爆炸，提出误差项在反向传播中具有**范数保持性**，即

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

  - 若以均值0，方差1/M的高斯分布来随机生成权重矩阵 $\mathbf{W}^{(l)}$，则当 $\mathrm{M} \to \infty$ 时，范数保持成立；若M不够大，则难以保证范数保持性

3 正交初始化

- 正交初始化（Orthogonal initialization）
  - 将 $\mathbf{W}^{(l)}$ 初始化为正交矩阵，即 $\mathbf{W}^{(l)}\left(\mathbf{W}^{(l)}\right)^{\mathrm{T}}=\mathbf{I}$ 。
  - 步骤：1）均值0，方差1的高斯分布初始化一个矩阵；2）采用奇异值分解作用于此阵，得到两个正交矩阵，并将其中一个作为权重矩阵 $\mathbf{W}^{(l)}$

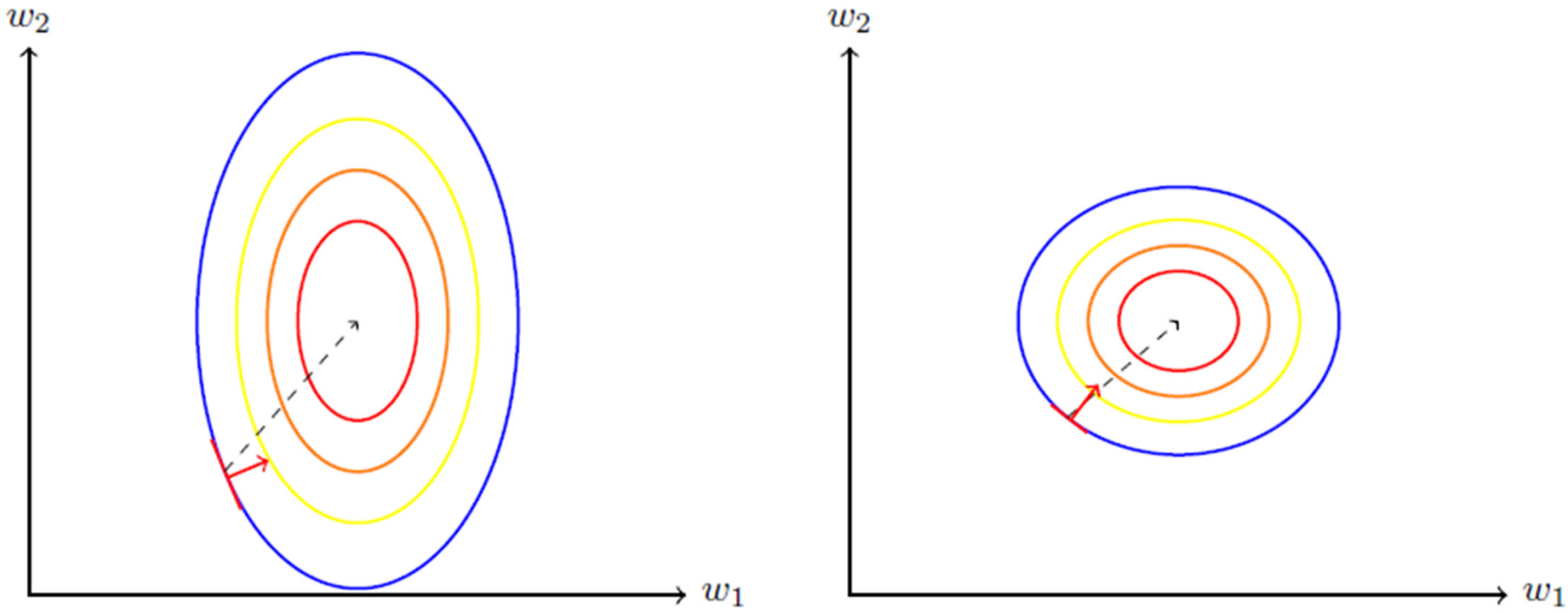
$$\mathbf{A}=\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}$$

  - 根据正交矩阵的性质，这个线性网络在信息的前向传播过程和误差的反向传播过程中，都具有范数保持性，从而可有效避免在训练开始时就出现梯度消失或梯度爆炸现象
  - 当在非线性神经网络中应用正交初始化时，通常需要将正交阵乘以一个缩放系数 ρ

4 数据预处理

数据预处理

- 尺度不变性（scale invariance）
  - 如果一个机器学习算法在缩放全部或部分特征后不影响它的学习和预测，我们称算法具有尺度不变性
  - 对尺度敏感模型，须先对样本进行预处理，将样本特征各维度转换到同一尺度，并消除彼此之间的相关性
  - 不同尺度的输入特征会增加神经网络的训练难度
- 影响梯度下降算法效率
  - 不同尺度输入特征差异较大，梯度方向并不指向最优搜索方向



(a) 未归一化数据的梯度　　(b) 归一化数据的梯度

图 7.9 数据归一化对梯度的影响

数据预处理

- 归一化（Normalization）
  - 泛指把数据特征转换为相同尺度的方法
  - 假设有N个样本 $\{x^{(n)}\}_{n=1}^{N}$，对特征的每一维进行归一化

1）缩放归一化，最大最小归一化 Min-max normalization

$$\hat{x}^{(n)} = \frac{x^{(n)} - \min_n\left(x^{(n)}\right)}{\max_n\left(x^{(n)}\right) - \min_n\left(x^{(n)}\right)}$$

2）标准化 stanardization 也叫Z值归一化 （Z-Score normalizatoin）

$$\hat{x}^{(n)} = \frac{x^{(n)} - \mu}{\sigma} \qquad \mu = \frac{1}{N}\sum_{n=1}^{N} x^{(n)}, \sigma^2 = \frac{1}{N}\sum_{n=1}^{N}\left(x^{(n)} - \mu\right)^2$$

数据预处理

- 归一化（Normalization）

3）白化 whitening 用来降低数据之间的冗余性，减少了特征之间的相关性，且所有特征具有相同方差。常采用主成分分析PCA方法去掉各成分之间的相关性

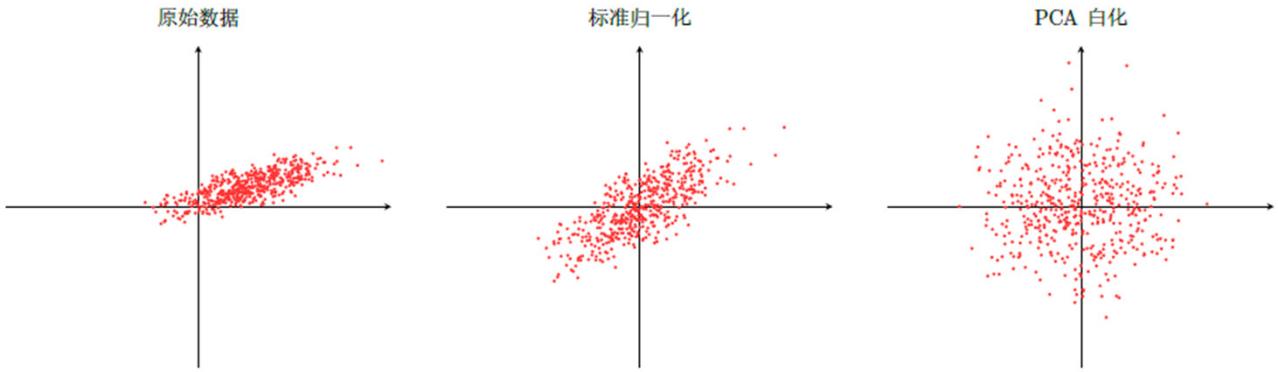


图 7.10 标准归一化和PCA 白化

5 & 6 逐层归一化和超参数优化

逐层归一化

- 逐层归一化（Layer-wise normalization）可有效提升训练效率
  - 更好地尺度不变性
    - 如果一个神经层的输入分布发生了改变，那么其他参数需要重新学习，此种现象称为内部协变量偏移（Internal covariate shift）.为此，需要对每层输入进行归一化，使得输入具有尺度不变性。
  - 更平滑的优化地形
    - 使得输入处于非饱和区，避免了梯度消失
    - 使得神经网络的优化地形（Optimization landscape）更加平滑

为了解决内部协变量偏移问题，要使得每个神经层的输入的分布在训练过程中保持一致，最直接的方法就是对每层神经元进行归一化操作

批量归一化

- 批量归一化 Batch Normalization BN, 可对神经网络中任意的中间层进行归一化操作。
  - 给定一个包含K 个样本的小批量样本集合，第l层神经元的净输入$\mathbf{z}^{(1,l)}$ ，..., $\mathbf{z}^{(K,l)}$ 计算均值和方差

$$\boldsymbol{\mu}_B = \frac{1}{K}\sum_{k=1}^{K}\mathbf{z}^{(k,l)}, \sigma_B^2 = \frac{1}{K}\sum_{k=1}^{K}\left(\mathbf{z}^{(k,l)} - \boldsymbol{\mu}_B\right)\odot\left(\mathbf{z}^{(k,l)} - \boldsymbol{\mu}_B\right)$$

  - 批量归一化(引入可学习参数 γ 和 β ，远离线性区，增加非线性)

$$\hat{\mathbf{z}}^{(l)} = \frac{\mathbf{z}^{(l)} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \varepsilon}} \Rightarrow \hat{\mathbf{z}}^{(l)} = \frac{\mathbf{z}^{(l)} - \boldsymbol{\mu}_B}{\sqrt{\boldsymbol{\sigma}_B^2 + \varepsilon}} \odot \boldsymbol{\gamma} + \boldsymbol{\beta} \triangleq \mathrm{BN}_{\gamma,\beta}\left(\mathbf{z}^{(l)}\right)$$

  - 作用于每层激活函数之前 $\mathbf{a}^{(l)} = \mathrm{f}\left(\mathrm{BN}_{\gamma,\beta}\left(\mathbf{z}^{(l)}\right)\right) = \mathrm{f}\left(\mathrm{BN}_{\gamma,\beta}\left(\mathbf{W}\mathbf{a}^{(l-1)}\right)\right)$

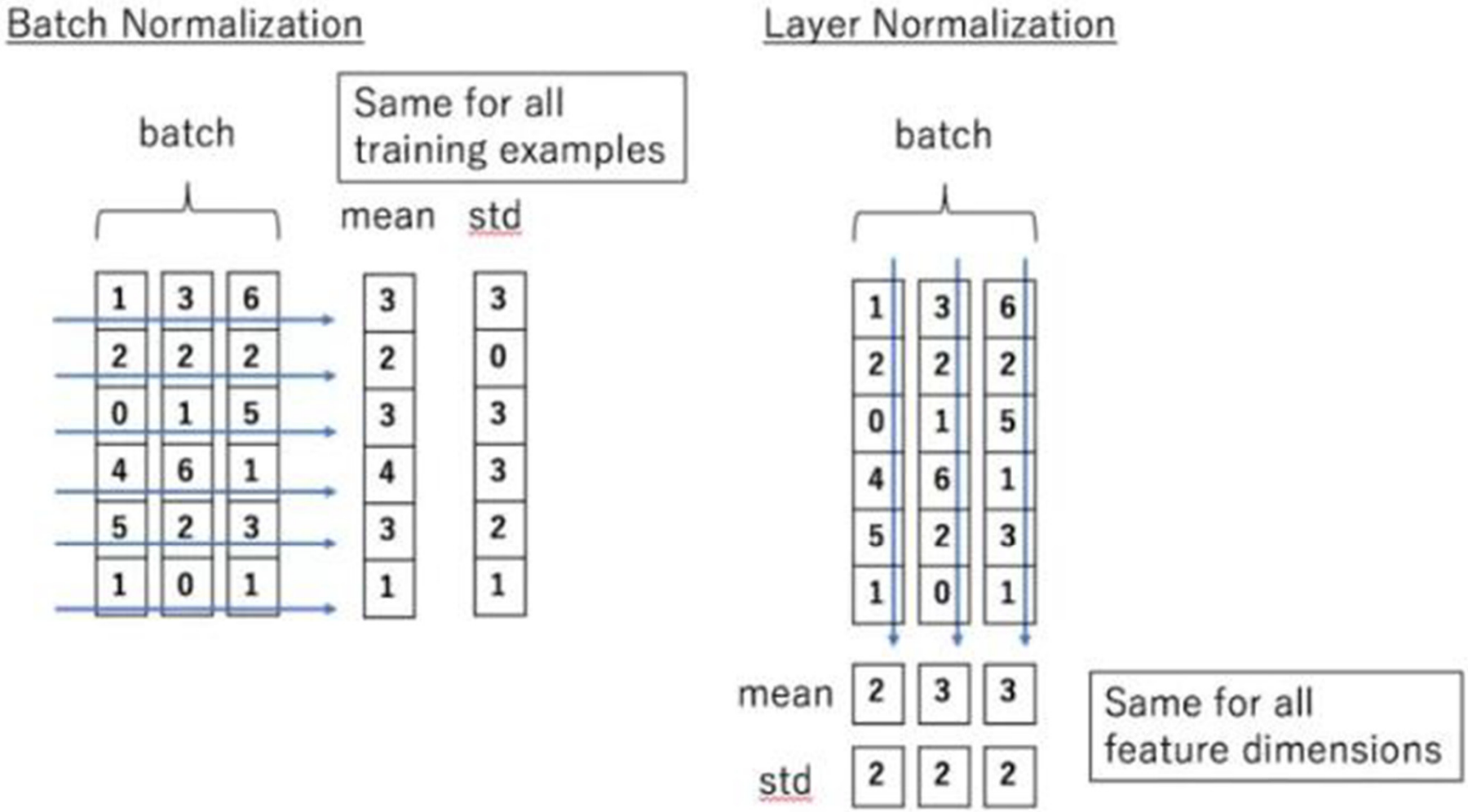
层归一化 Layer Normalization

- 小批量样本不适合BN，难以计算单个神经元的统计信息
- 对一个中间层所有神经元的归一化
  - 第l层M个神经元，净输入为 $\mathbf{z}^{(l)}$ ，则

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

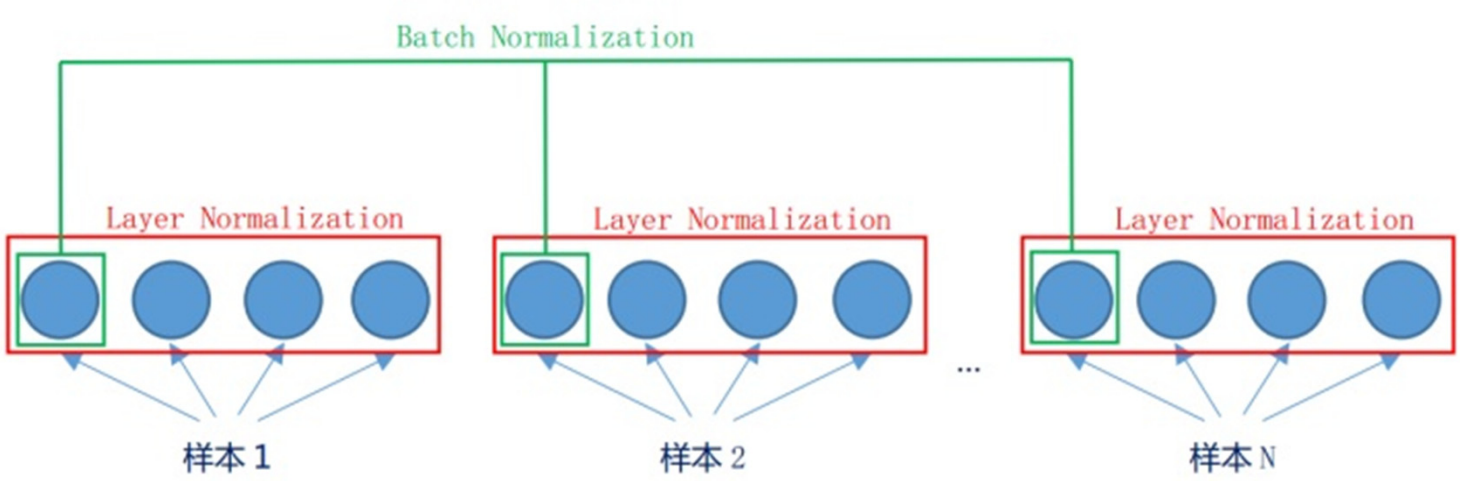
  - 层归一化 $$\hat{\mathbf{z}}^{(l)} = \frac{\mathbf{z}^{(l)} - \mu^{(l)}}{\sqrt{\sigma^{(l)^2} + \varepsilon}} \odot \boldsymbol{\gamma} + \boldsymbol{\beta} \triangleq \mathrm{LN}_{\gamma,\beta}\left(\mathbf{z}^{(l)}\right)$$

  - 可以应用在循环神经网络中，对循环神经层进行归一化操作，可有效缓解梯度消失或爆炸


Batch Normalization
batch
Same for all training examples
mean
std
1 3 6
2 2 2
0 1 5
4 6 1
5 2 3
1 0 1
3 2 3 4 3 1
3 0 3 3 2 1
Layer Normalization
batch
1 3 6
2 2 2
0 1 5
4 6 1
5 2 3
1 0 1
mean
2 3 3
std
2 2 2
Same for all feature dimensions

批量归一化VS层归一化

层归一化是对中间层的所有神经元节点进行归一化，均值方差均是标量
批量归一化则是对所有样本在中间层的归一化，均值方差是向量



更多的归一化

✓ 权重归一化 Weight Normalization

对神经连接权重进行归一化操作

$$\mathbf{W}_{i,:} = \frac{g_i}{\|\mathbf{v}_i\|}\mathbf{v}_i \quad 1 \le i \le M_l$$

✓ 局部响应归一化 Local Response Normalization LRN

一种受生物学启发的归一化方法，侧抑制

应用在激活函数之后，对同一位置邻近特征映射中的神经元进行归一化

$$\hat{Y}^p = Y^p / \left(k + \alpha \sum_{j=\max(1,p-\frac{n}{2})}^{\min(P,p+\frac{n}{2})} (Y^j)^2 \right)^{\beta}$$

$$\triangleq \mathrm{LRN}_{n,k,\alpha,\beta}(Y^p),$$

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
- 超参数优化存在的问题
  - 组合优化
  - 配置评估

- 超参数优化方法
  - 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

超参数对网络性能影响很大，常见的超参数有以下三类：

- 网络结构，包括神经元之间的连接关系、层数、每层的神经元数量、激活函数的类型等
- 优化参数，包括优化方法、学习率、小批量的样本数量等
- 正则化系数

超参数优化主要存在两方面困难：

- 优化难，超参数优化是一个组合优化问题
- 配置难，评估超参数配置的时间代价高

超参数配置比较简单的方法：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。
  - 这些超参数可以有 $m_1 \times m_2 \times \cdots \times m_K$ 个取值组合。
  - 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 $\alpha \in \{0.01, 0.1, 0.5, 1.0\}$.
  - 连续的超参数需结合自身特点进行离散化
  - 根据上述这些超参数组合分别训练一组模型，然后测试这些模型在验证集上的性能，选取性能最好的配置
- 随机搜索（Random Search）
  - 对超参数进行随机组合，选取性能最优的配置

上述两种搜索均未利用超参数彼此之间的相关性，效率较低

超参数优化

- 贝叶斯优化

根据当前已试验的超参数组合，预测下一个最优组合，常用的有时序模型优化SMBO，定义收益函数（如期望改善）

- 动态资源分配

✓最优臂问题，即在利用和探索（资源和配置）之间找到最佳平衡；

✓将有限的资源分配给更多可能带来收益的超参数组合（逐次减半）

- 神经架构搜索 Neural Architecture Search, NAS

不是在超参数空间中进行最优配置搜索；

网络架构的最优设计

7 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

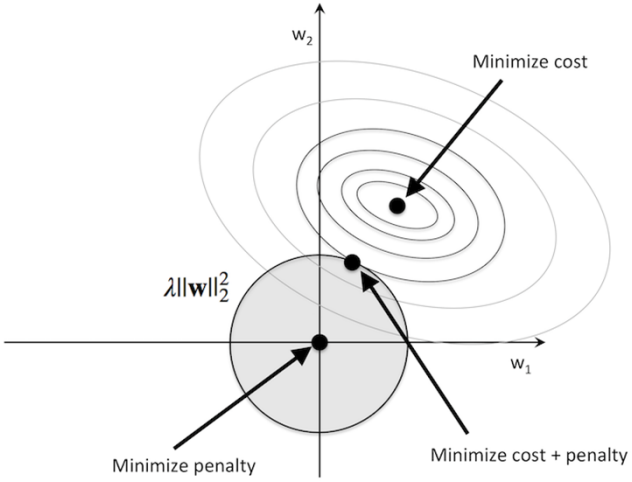
Zhang C, Bengio S, Hardt M, et al. Understanding deep learning requires rethinking generalization[J]. arXiv preprint arXiv:1611.03530, 2016.

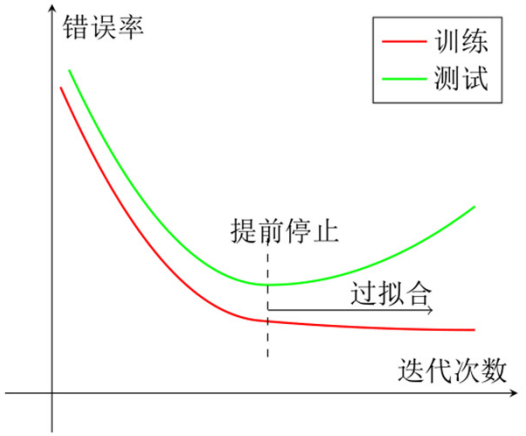
正则化（regularization）

正则化是一类通过限制模型复杂度，从而避免过拟合，提高泛化能力的方法，比如引入约束、增加先验、提前停止等；

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





正则化

- 如何提高神经网络的泛化能力
  - ℓ_1和ℓ_2正则化
  - 权重衰减
  - early stop
  - Dropout
  - 数据增强
  - 标签平滑

ℓ_1和ℓ_2正则化

- L1 & L2是最常用的正则化方法，通过约束参数的l1/l2范数，来减小模型在训练数据集上的过拟合现象；

- 优化问题可以写为
  - ℓ_p 为范数函数，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。

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

  - L1 最优解位于坐标轴上导致了稀疏性；
  - L1 & l2 弹性网络正则化

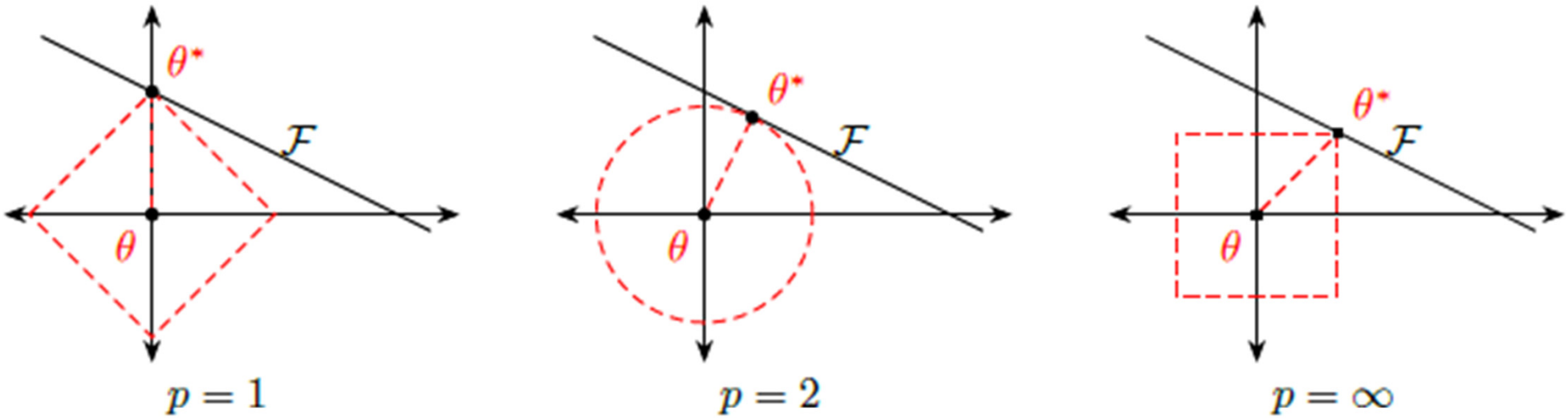
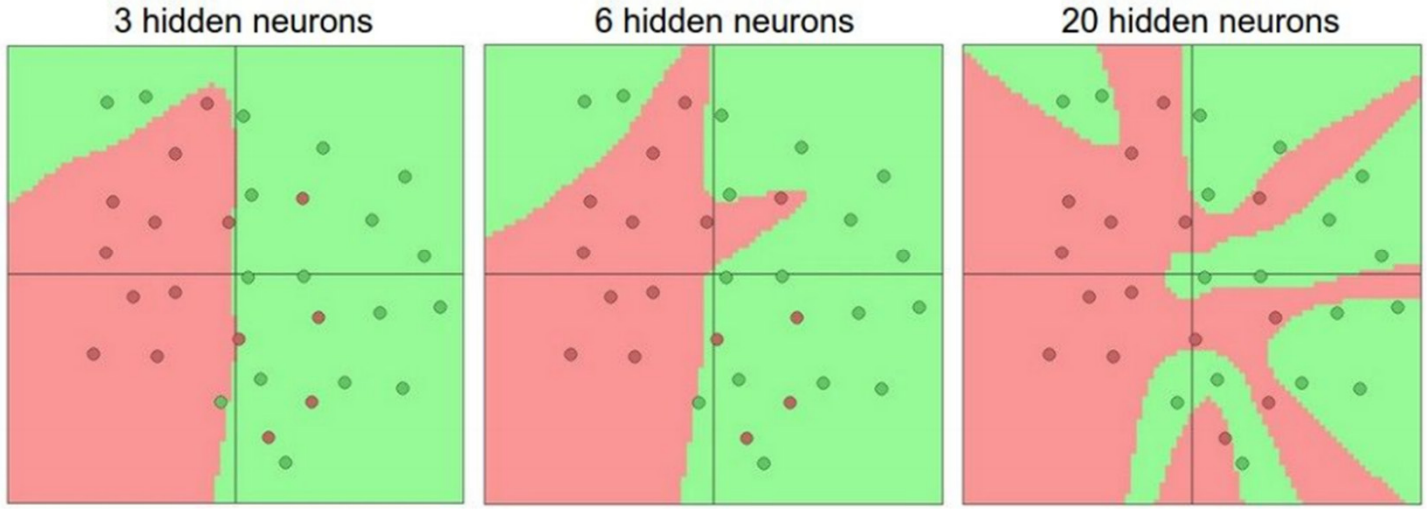


图 7.11 不同范数约束条件下的最优化问题示例

神经网络示例

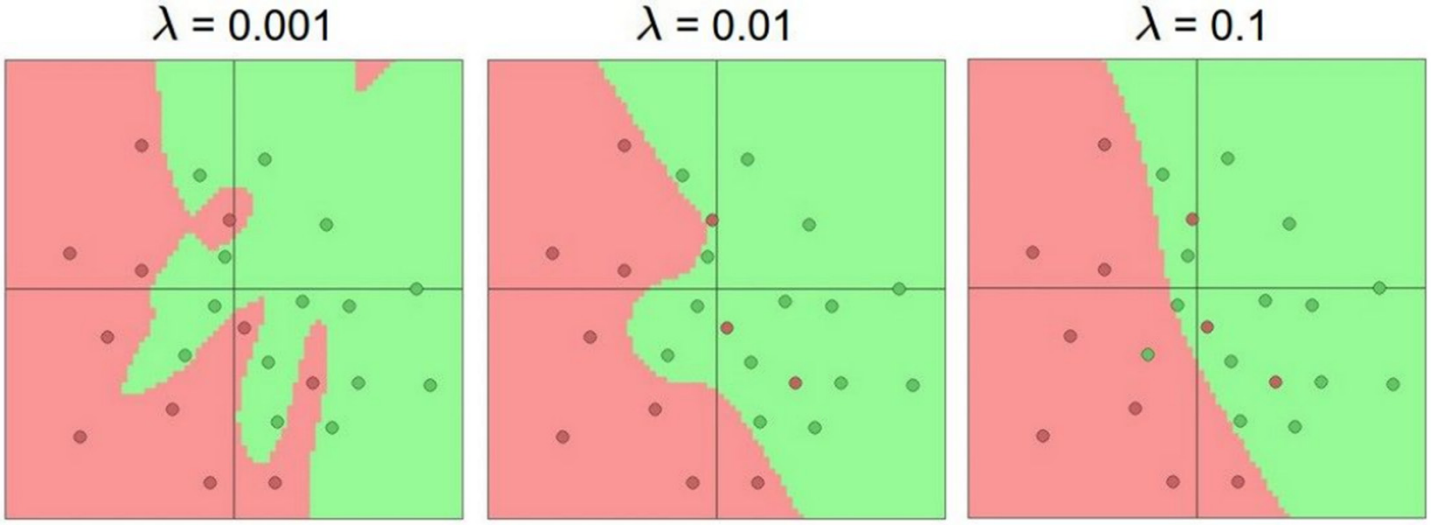
http://playground.tensorflow.org/

- 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



权重衰减（Weight Decay）

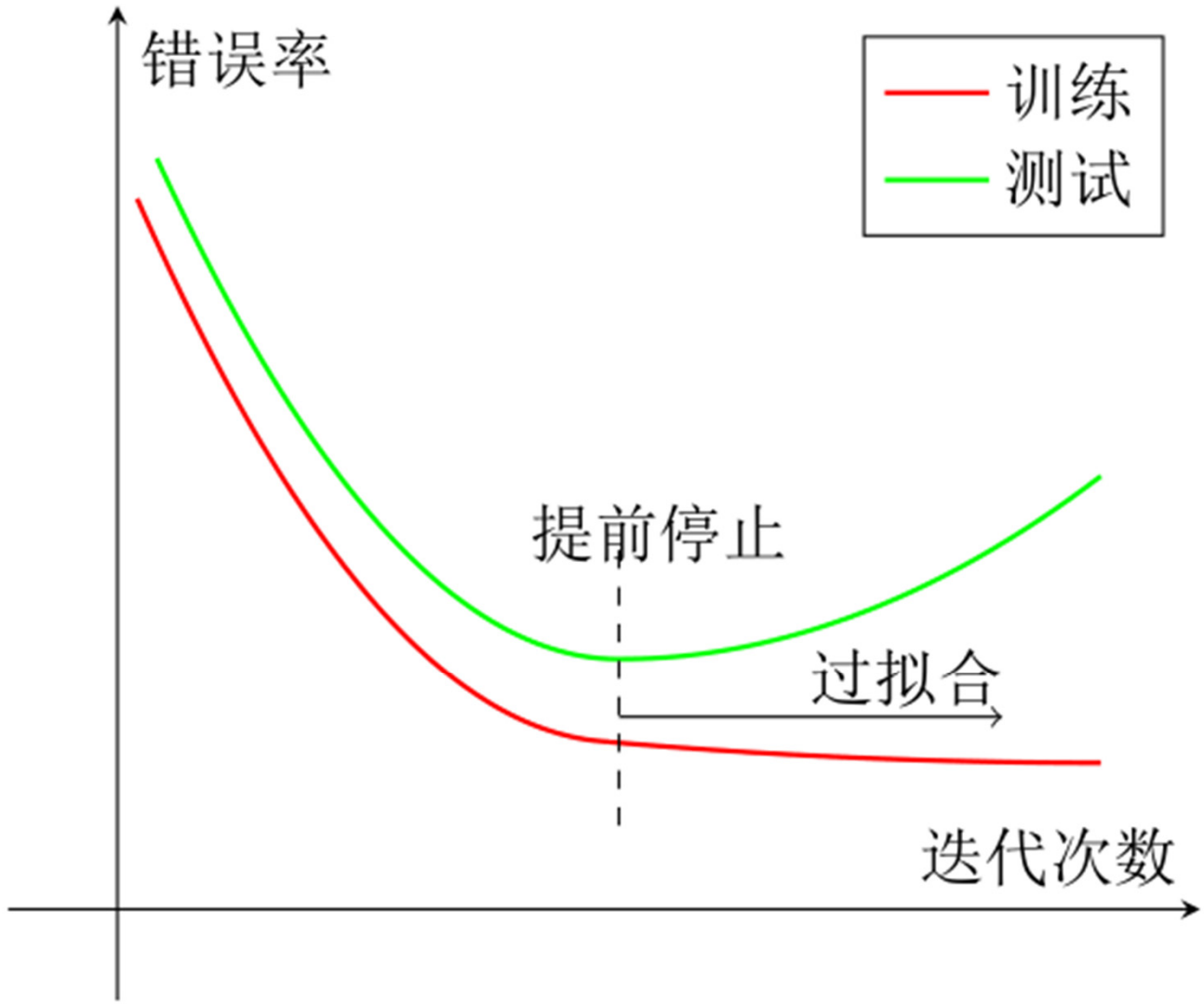
- 在每次参数更新时，引入一个衰减系数

$$\theta_t \leftarrow (1-\beta)\theta_{t-1} - \alpha g_t$$

  - gt为第t步的更新梯度，a学习率，β 为权重衰减系数
  - 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。

提前停止 Early stop

- 神经网络拟合能力强，易过拟合。训练时可以使用验证集上的错误代替期望错误。
- 我们使用一个验证集（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。

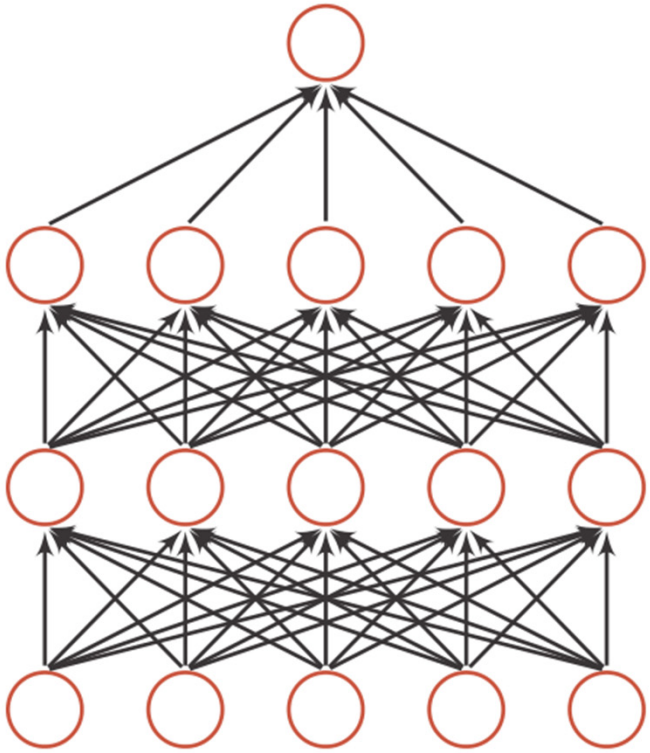

错误率
训练
测试
提前停止
过拟合
迭代次数

丢弃法（Dropout Method）

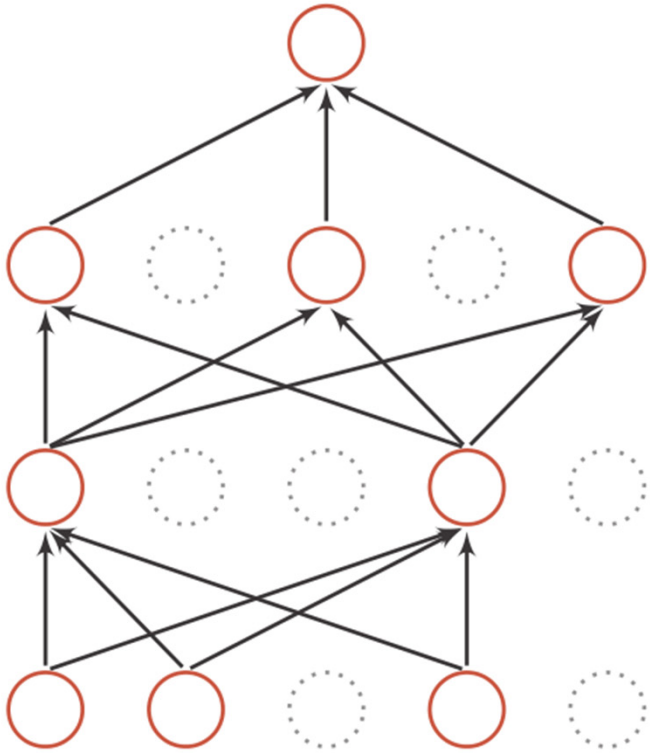
- 训练一个网络时，随机丢弃一部分神经元来避免过拟合，这种方法即是丢弃法
- 对于一个神经层$y = f(Wx + b)$，引入一个丢弃函数$d(\cdot)$使得$y = f(Wd(x) + b)$。

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

  - 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，即每个神经元以概率p来进行取舍
  - 一般针对神经元进行随机丢弃，也可以扩展至对神经元连接进行随机丢弃
  - 测试时，不用dropout，每个参数乘以p%



(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（简化网络）。如果一个神经网络有n个神经元，那么总共可以采样出2^n个子网络（集成学习）
- 贝叶斯学习的解释

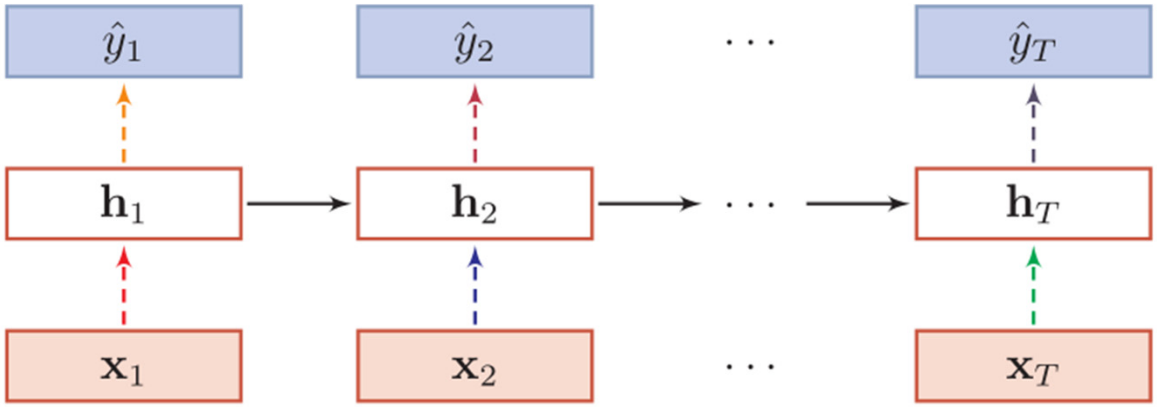
$$\mathbb{E}_{q(\theta)}[y] = \int_q f(\mathbf{x}, \theta) q(\theta) d\theta$$

$$\approx \frac{1}{M} \sum_{m=1}^{M} f(\mathbf{x}, \theta_m),$$

  - 其中$f(x,\theta_m)$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，而是在非时间维度的连接（即非循环连接）进行随机丢失。



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。

数据增强（Data Augmentation）

- 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，以此提高模型鲁棒性，避免过拟合。
  - 图像数据的增强方法：
    - 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）

- 标签平滑：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

$$\tilde{\mathbf{y}} = [\frac{\epsilon}{K-1}, \cdots, \frac{\epsilon}{K-1}, 1-\epsilon, \frac{\epsilon}{K-1}, \cdots, \frac{\epsilon}{K-1}]^{\mathrm{T}}$$ 软目标（Soft Target）

- 标签平滑可以避免模型输出过拟合至硬目标上，并且通常不会损害其分类能力

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
- 优化
  - SDG+mini-batch（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 & BN）
  - 动态学习率（越来越小）
- 正则化
  - ℓ_1和ℓ_2正则化（跳过前几轮）
  - Dropout
  - Early-stop
  - 数据增强

谢 谢