



北京交通大学《深度学习》课件

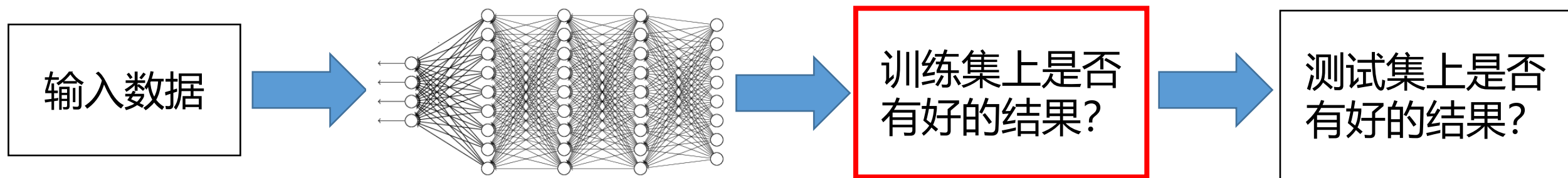
第五讲 深度模型优化与正则化



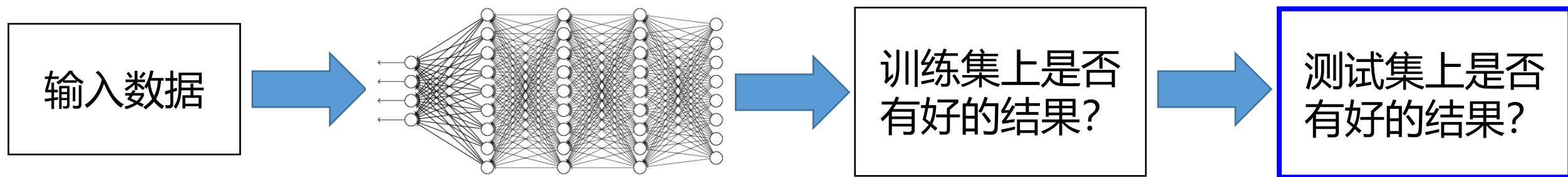
主讲教师：丛润民

北京交通大学 《深度学习》课程组





优化问题：神经网络模型是一个非凸函数，再加上在深度网络中的梯度消失问题，很难进行优化；另外，深层神经网络模型一般参数比较多，训练数据也比较大，会导致训练的效率比较低。



泛化问题（正则化）：因为神经网络的拟合能力强，反而容易在训练集上产生过拟合。因此，在训练深层神经网络时，同时也需要通过一定的正则化方法来改进网络的泛化能力。



5.1 网络优化

5.2 小批量梯度下降

5.3 学习率与梯度优化

5.4 参数初始化与数据预处理

5.5 逐层归一化

5.6 超参数优化

5.7 过拟合与正则化



5.1 网络优化

5.2 小批量梯度下降

5.3 学习率与梯度优化

5.4 参数初始化与数据预处理

5.5 逐层归一化

5.6 超参数优化

5.7 过拟合与正则化



5.1 网络优化

网络优化目标：最小化取自数据生成分布数据的预测误差期望

$$J(\theta) = E_{(\mathbf{x}, \mathbf{y}): p_{data}} [L(f(\mathbf{x}; \theta), \mathbf{y})]$$

通常 p_{data} 未知，只知道训练集中的样本： $p_{data} \approx \hat{p}_{data}$

经验风险最小化：用训练集上的经验分布替代真实分布

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



网络优化难点

➤ 网络结构多样性:

很难找到一种通用的、高效的、稳定的优化方法

超参数一般也比较多

➤ 非凸优化问题:

基于梯度下降的优化方法会陷入局部最优点，因此在低维空间中非凸优化的主要难点是如何选择初始化参数和逃离局部最优点



非凸优化问题

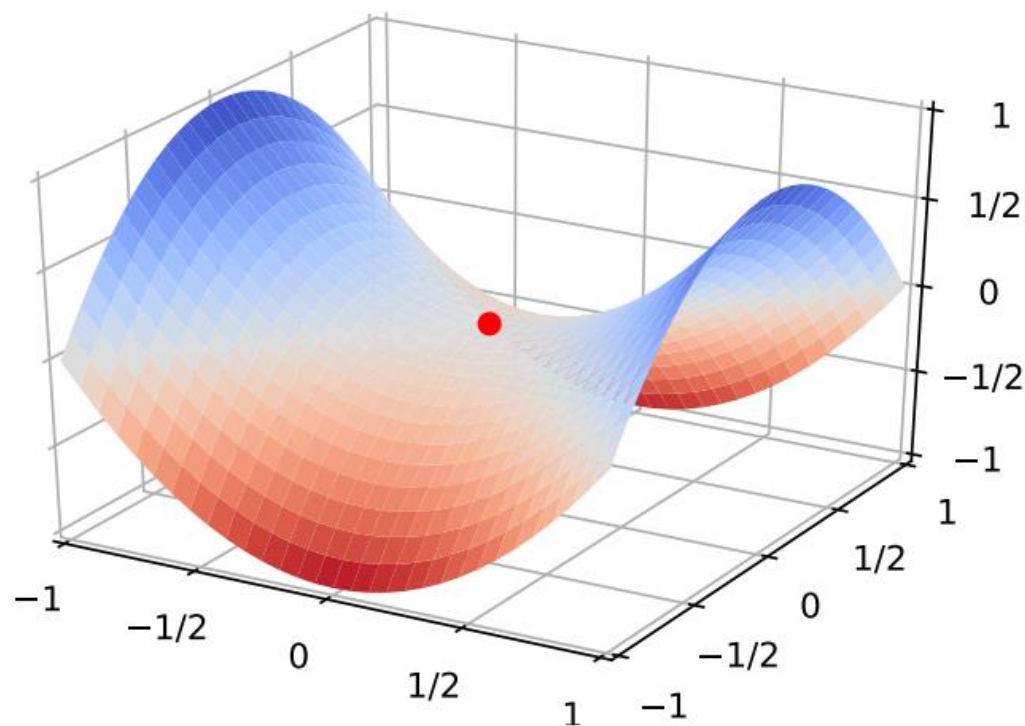
在**高维空间**中，非凸优化的难点并不在于如何逃离**局部最优点**，而是如何逃离**鞍点**。

$$\nabla L(\theta) = 0$$

局部最小点: $\nabla^2 L(\theta)$ 的所有特征值为正

局部最大点: $\nabla^2 L(\theta)$ 的所有特征值为负

鞍点: $\nabla^2 L(\theta)$ 的特征值有正有负





网络优化的改善方法

➤ 网络结构多样性:

寻找高效的、稳定的优化方法

5.2 小批量梯度下降

5.3 自适应学习率、梯度优化

5.4 参数初始化、数据预处理

5.5 逐层归一化

超参数一般也比较多

5.6 超参数优化

➤ 非凸优化问题:

逃离局部最优

5.3 自适应学习率、梯度优化



5.1 网络优化

5.2 小批量梯度下降

5.3 学习率与梯度优化

5.4 参数初始化与数据预处理

5.5 逐层归一化

5.6 超参数优化

5.7 过拟合与正则化



5.2 小批量梯度下降

批量梯度下降: batch gradient descent

随机梯度下降: stochastic gradient descent

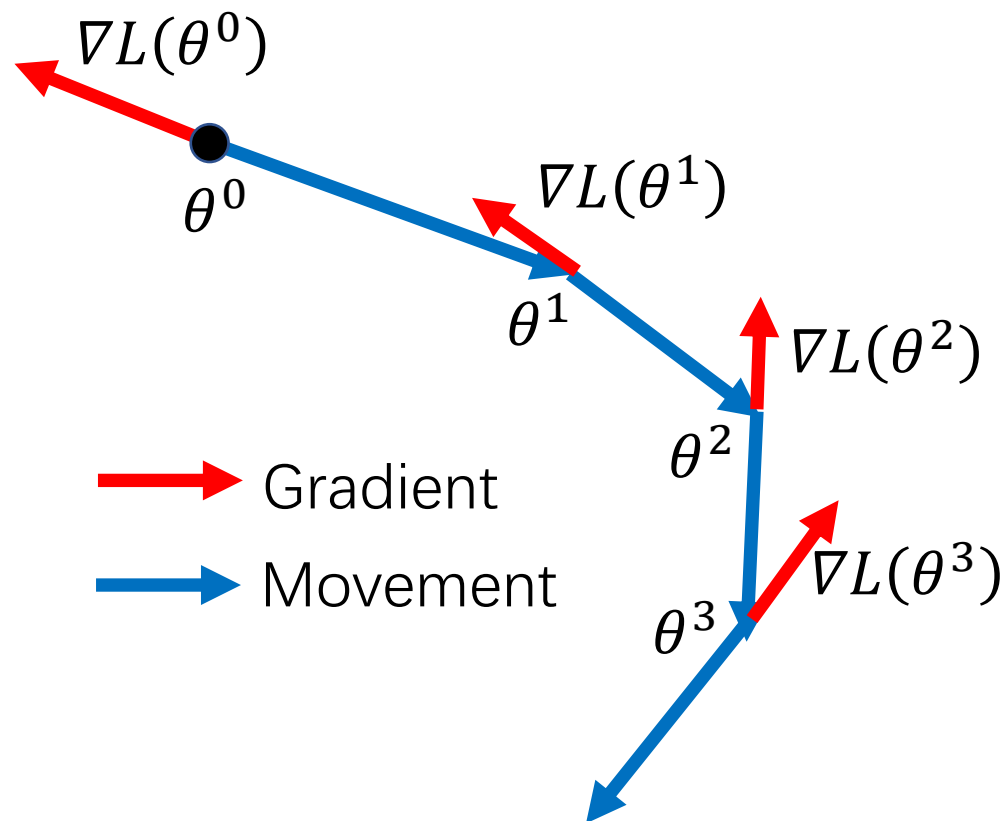
小批量梯度下降: mini-batch gradient descent

几个关键因素:

- 小批量样本数量
- 学习率
- 梯度



批量梯度下降



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

批量梯度下降：每次更新都使用整个训练集数据；梯度方差小；需要较多计算资源

从起始位置 θ^0 开始

在 θ^0 处计算梯度

应用更新 $\theta^1 = \theta^0 - \alpha \nabla L(\theta^0)$

在 θ^1 处计算梯度

应用更新 $\theta^2 = \theta^1 - \alpha \nabla L(\theta^1)$

⋮

停止迭代直至 $\nabla L(\theta^t) \approx 0$



随机梯度下降算法

Require: 学习率 α

Require: 初始参数 θ

while 停止准则未满足 **do**

 对训练集 D 中样本随机排序

for $n=1,2,\dots,N$ **do**

 更新参数 $\theta \leftarrow \theta - \alpha \nabla L(\theta)$

end for

end while

每次只取一个样本计算梯度:

$$\nabla L(\theta) = \nabla L(f(\mathbf{x}^{(n)}; \theta), y^{(n)})$$



小批量梯度下降

- 选取包含 K 个样本的 MiniBatch, 计算偏导数:

$$\nabla L(\theta) = \frac{1}{K} \sum_{(\mathbf{x}, y) \in S^t} \nabla L(f(\mathbf{x}; \theta), y)$$

- 定义第 t 次更新的梯度:

$$\mathbf{g}^t @ \nabla L(\theta^{t-1})$$

- 更新参数:

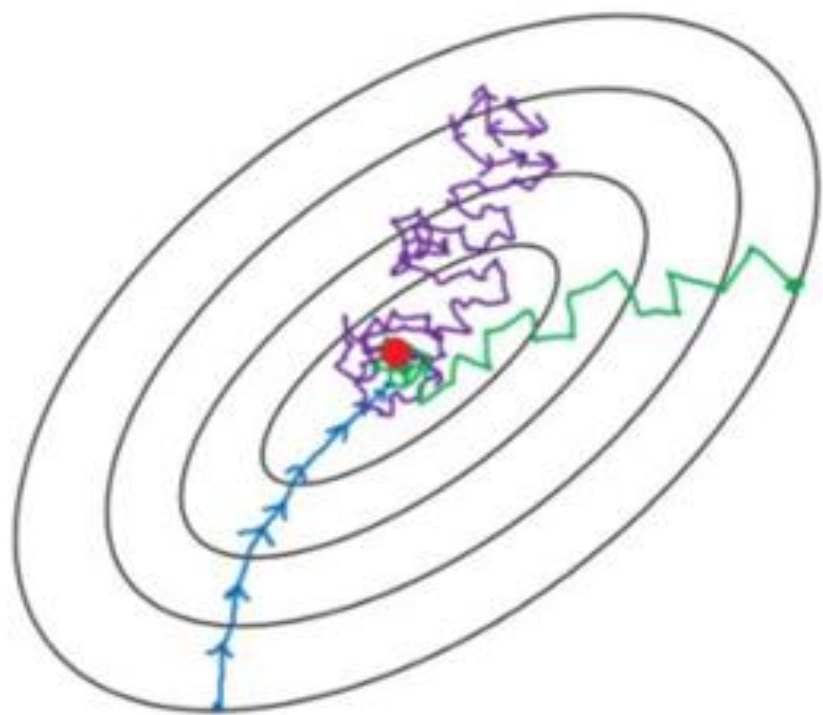
$$\theta^t = \theta^{t-1} - \alpha \mathbf{g}^t$$

几个关键因素:

- 小批量样本数量
- 梯度
- 学习率



随机梯度下降 vs 批量梯度下降 vs 小批量梯度下降



— Batch gradient descent
— Mini-batch gradient Descent
— Stochastic gradient descent

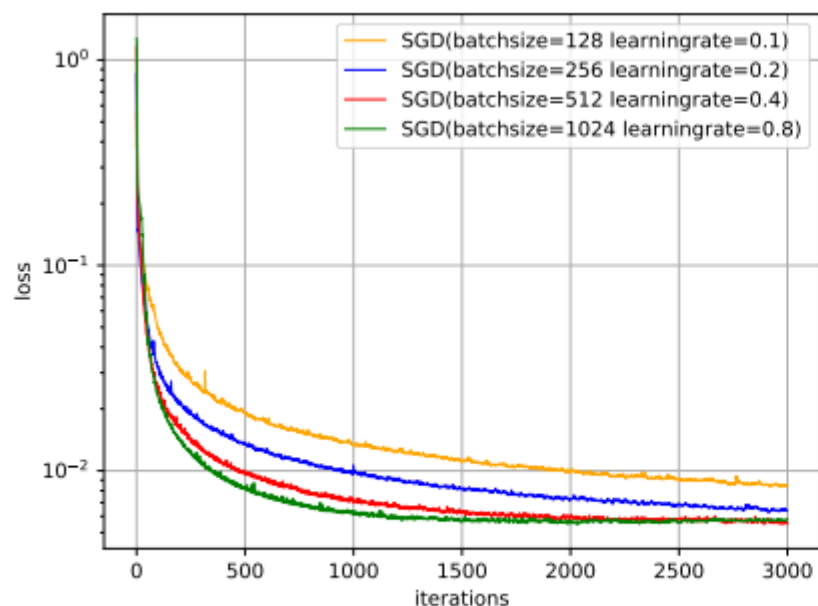
- 批量梯度下降：利于寻找全局最优解，梯度方差小；但样本数目很多时，训练过程会很慢。
- 随机梯度下降：训练速度快；准确度下降，并不是全局最优，梯度方差大。
- 小批量梯度下降法：同时兼顾两种方法的优点。



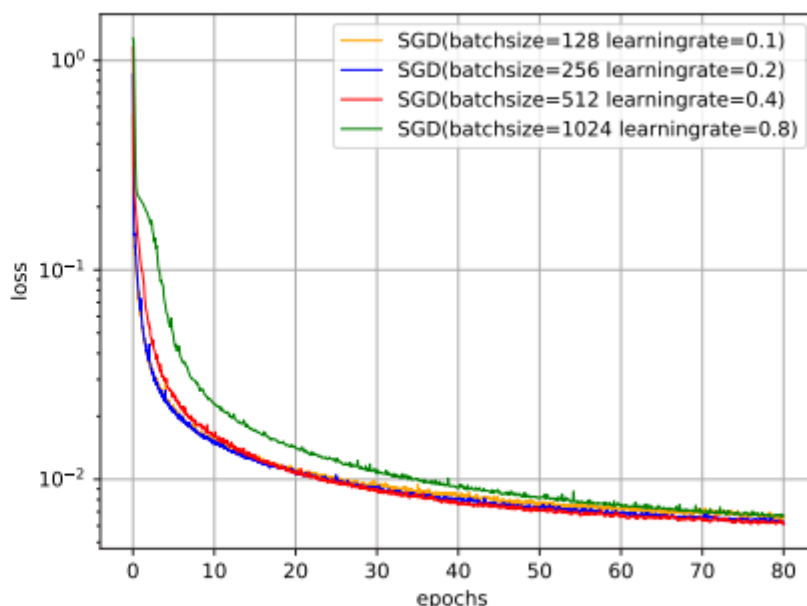
批量大小的影响

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，训练也越稳定，因此可以设置较大的学习率。
- 批量较小时，需要设置较小的学习率，否则模型会不收敛。



(a) 按迭代 (Iteration) 的损失变化



(b) 按回合 (Epoch) 的损失变化

$$\begin{aligned} &1 \text{ 个 } Epoch \text{ 包含 } Iterations \\ &= \frac{\text{训练样本的数量 } N}{\text{批量大小 } K} \end{aligned}$$

每一次小批量更新为一次迭代， 所有训练集的样本更新一遍为一个回合



5.1 网络优化

5.2 小批量梯度下降

5.3 学习率与梯度优化

5.4 参数初始化与数据预处理

5.5 逐层归一化

5.6 超参数优化

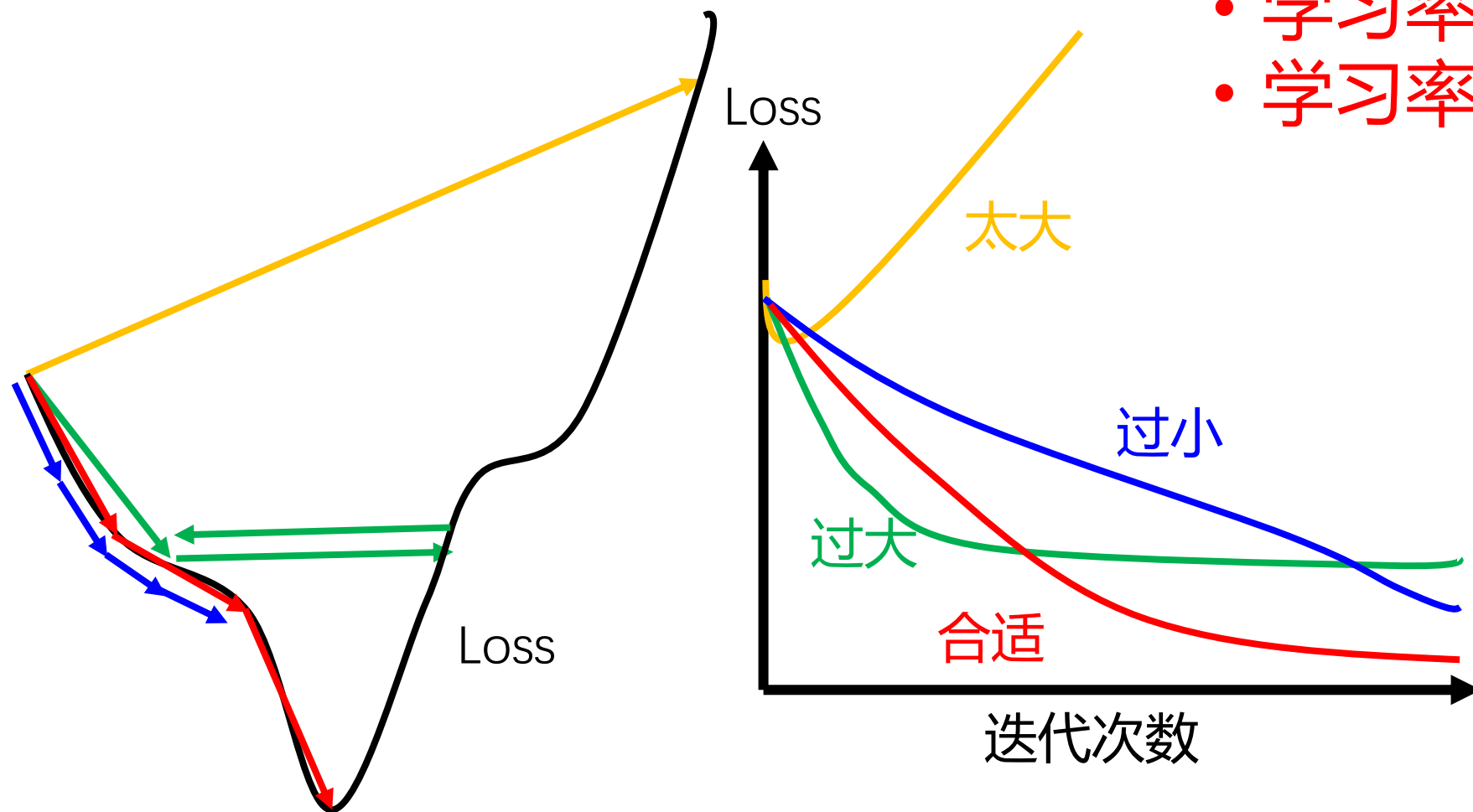
5.7 过拟合与正则化



5.3 学习率与梯度优化

学习率的影响: $\theta^t = \theta^{t-1} - \alpha \mathbf{g}^t$

- 学习率过大: 不收敛
- 学习率过小: 收敛太慢

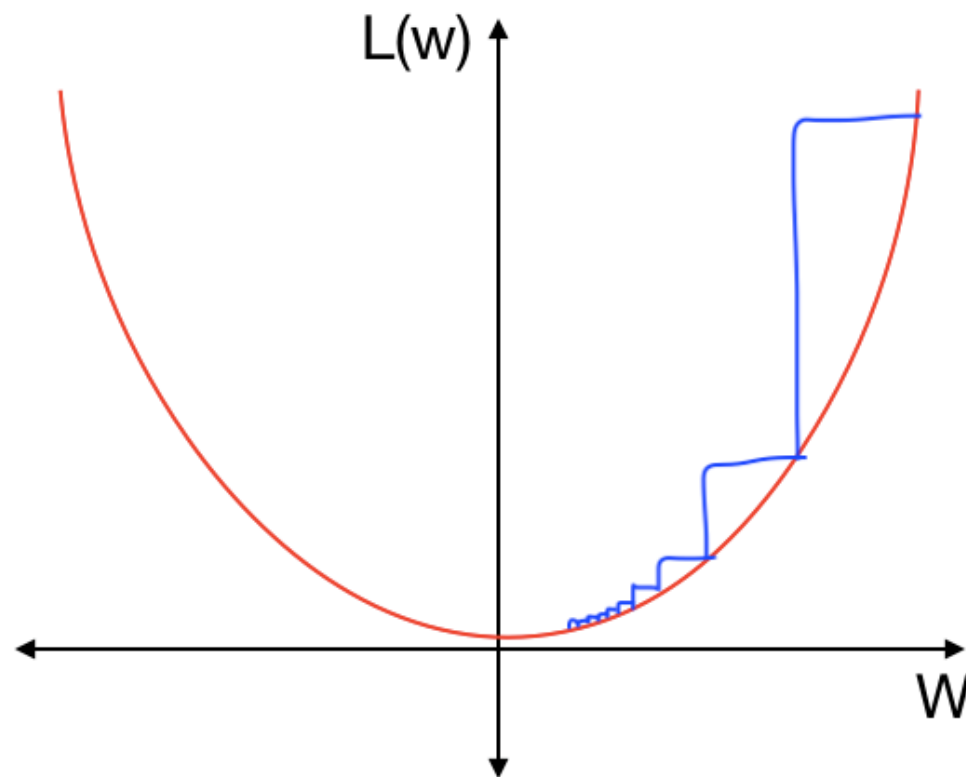




学习率衰减

经验上看，学习率在一开始要保持大些来保证收敛速度，收敛到最优点附近时要小些以避免来回震荡。

- 开始时，距离极值点处很远，采用大些的学习率，尽快接近极值点。
- 迭代多次后，接近极值点处，减小学习率，保证收敛，避免震荡。
- 多种减小学习率的方式。





学习率衰减

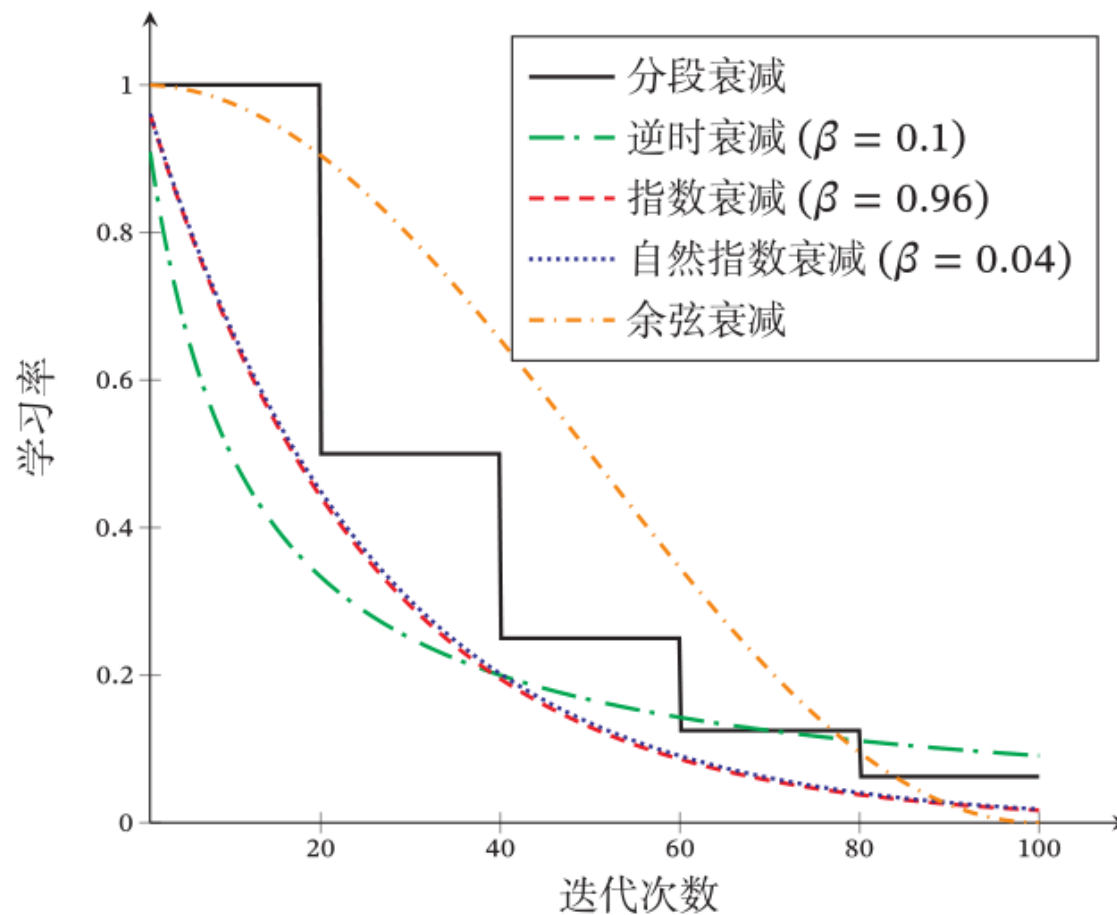
➤ 分段常数衰减：每经过 T_1, \dots, T_m 迭代，学习率衰减为原来的 β_1, \dots, β_m 倍

➤ 逆时衰减：
$$\alpha_t = \alpha_0 \frac{1}{1 + \beta \times t}$$

➤ 指数衰减：
$$\alpha_t = \alpha_0 \beta^t$$

➤ 自然指数衰减：
$$\alpha_t = \alpha_0 e^{-\beta t}$$

➤ 余弦衰减：
$$\alpha_t = \frac{1}{2} \alpha_0 (1 + \cos(\frac{t\pi}{T}))$$





学习率预热

当批量大小比较大时，需要比较大的学习率。但开始训练的参数是随机初始化的，梯度往往也比较大，加上比较大的初始学习率，会使得训练不稳定。

- **学习率预热**：为提高训练稳定性，可以在最初几轮迭代时采用较小的学习率，等梯度下降到一定程度后再恢复到初始学习率。
- **逐渐预热 (Gradual Warmup)**：假设预热的迭代次数为 T' ，预热过程中，学习率为：

$$\alpha_t = \frac{t}{T'} \alpha_0, \quad 1 \leq t \leq T'$$

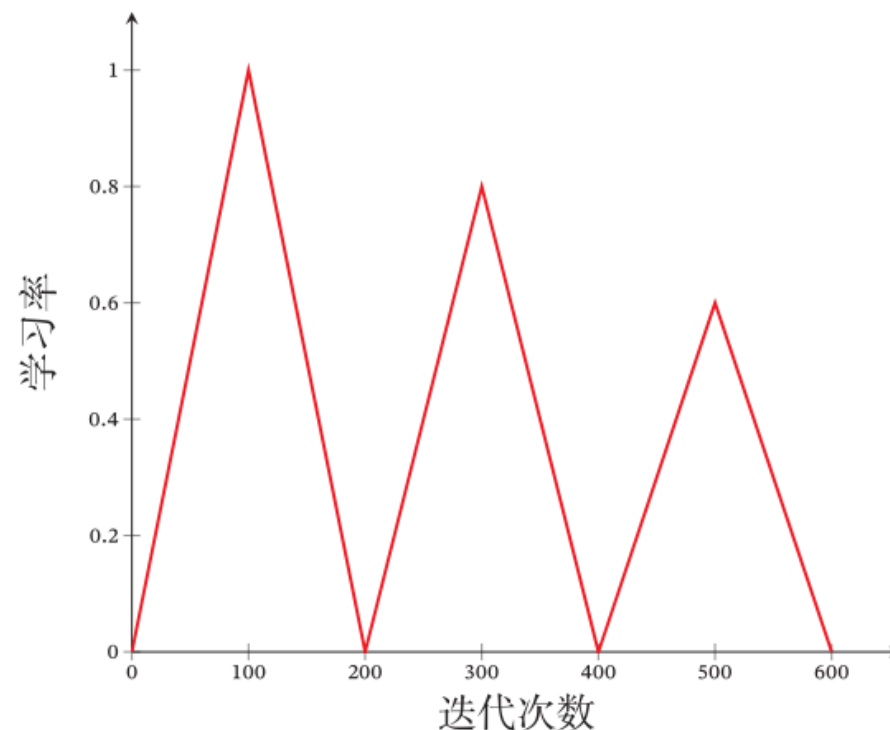


周期性学习率调整

为了逃离局部最小值或鞍点，可在训练过程中周期性地增大学习率。短期内有损收敛稳定性，长期来看有助于找到更好的局部最优解。

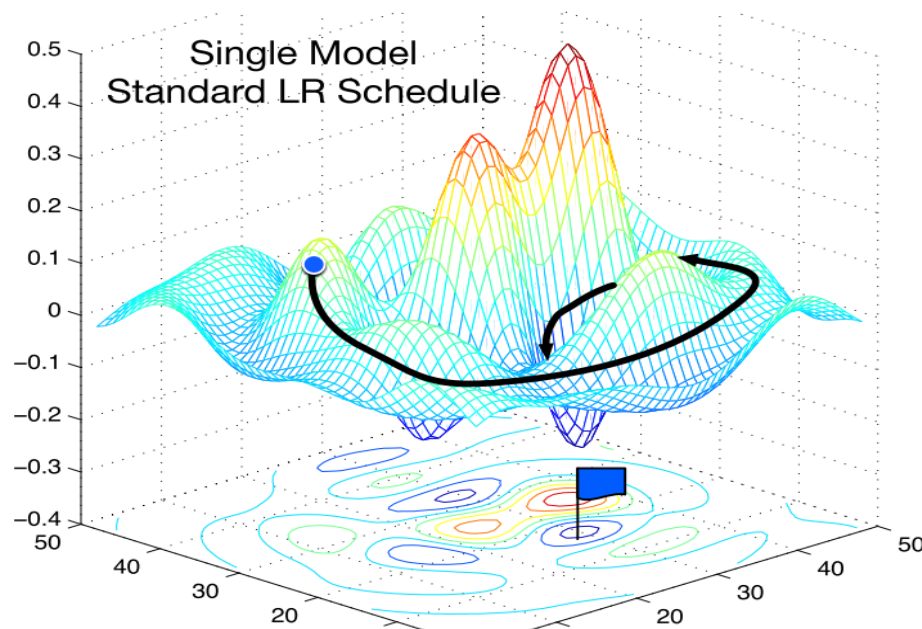
- **循环学习率**：让学习率在一个区间内周期性地增大和缩小。通常可以使用线性缩放来调整学习率，称为**三角循环学习率**。

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

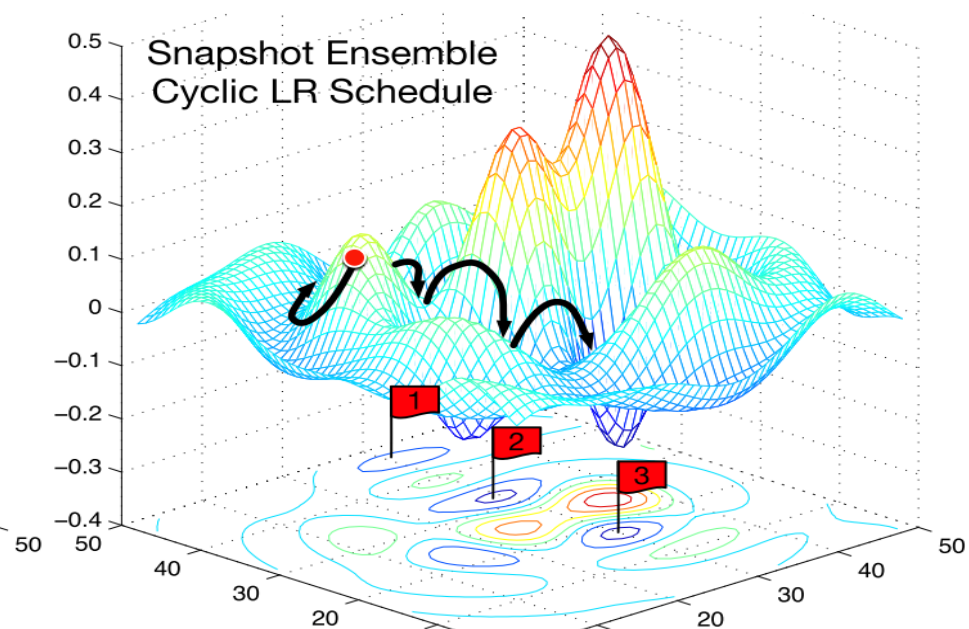




周期性学习率调整



一般的学习率调整，
容易陷入局部极值点



周期性学习率调整，有
助于逃离局部极值点



学习率衰减的局限性：

- 非自适应，不能够根据当前梯度情况做出调整
- 每个参数的维度上收敛速度都不相同，应该根据不同参数的收敛情况分别设置学习率

AdaGrad算法：借鉴 l_2 正则化的思想，每次迭代时自适应地调整每个参数的学习率。独立地适应所有模型参数的学习率，缩放每个参数反比于其所有梯度历史平方值总和的平方根。

$$G_t = \sum_{\tau=1}^t \mathbf{g}_{\tau} \odot \mathbf{g}_{\tau}, \quad \Delta \theta_t = -\frac{\alpha}{\sqrt{G_t + \epsilon}} \odot \mathbf{g}_t,$$

第 τ 次迭代时的梯度
元素乘积
参数更新差值
初始的学习率

在第 t 次迭代时，每个参数梯度平方的累计值

保持数值稳定性



AdaGrad算法

w_i 是模型的一个参数

$$\theta = \{w_1, \dots, w_i, \dots, w_{|\theta|}\}$$

$$w_i^1 \leftarrow w_i^0 - \frac{a}{\sigma_i^0} g_i^0$$

$$w_i^2 \leftarrow w_i^1 - \frac{a}{\sigma_i^1} g_i^1$$

$$w_i^3 \leftarrow w_i^2 - \frac{a}{\sigma_i^2} g_i^2$$

\vdots

$$w_i^{t+1} \leftarrow w_i^t - \frac{a}{\sigma_i^t} g_i^t$$

$$\sigma_i^0 = \sqrt{(g_i^0)^2 + \delta}$$

$$\sigma_i^1 = \sqrt{(g_i^0)^2 + (g_i^1)^2 + \delta}$$

$$\sigma_i^2 = \sqrt{(g_i^0)^2 + (g_i^1)^2 + (g_i^2)^2 + \delta}$$

$$\sigma_i^t = \sqrt{\sum_{\tau=0}^t (g_i^\tau)^2 + \delta}$$



RMSprop算法

AdaGrad局限：在经过一定次数的迭代依然没有找到最优点时，由于这时的学习率已经非常小，很难再继续找到最优点。

RMSprop：将 G_t 的计算由累积方式变成了**指数衰减移动平均**。

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

可以在有些情况下避免AdaGrad算法中学习率不断单调下降以至于过早衰减的缺点



RMSprop算法

w_i 是模型的一个参数 $\theta = \{w_1, \dots, w_i, \dots, w_{|\theta|}\}$

$$w_i^1 \leftarrow w_i^0 - \frac{a}{\sigma_i^0} g_i^0$$

$$\sigma_i^0 = g_i^0$$

$$w_i^2 \leftarrow w_i^1 - \frac{a}{\sigma_i^1} g_i^1$$

$$\sigma_i^1 = \sqrt{\beta(\sigma_i^0)^2 + (1 - \beta)(g_i^1)^2 + \delta}$$

$$w_i^3 \leftarrow w_i^2 - \frac{a}{\sigma_i^2} g_i^2$$

$$\sigma_i^2 = \sqrt{\beta(\sigma_i^1)^2 + (1 - \beta)(g_i^2)^2 + \delta}$$

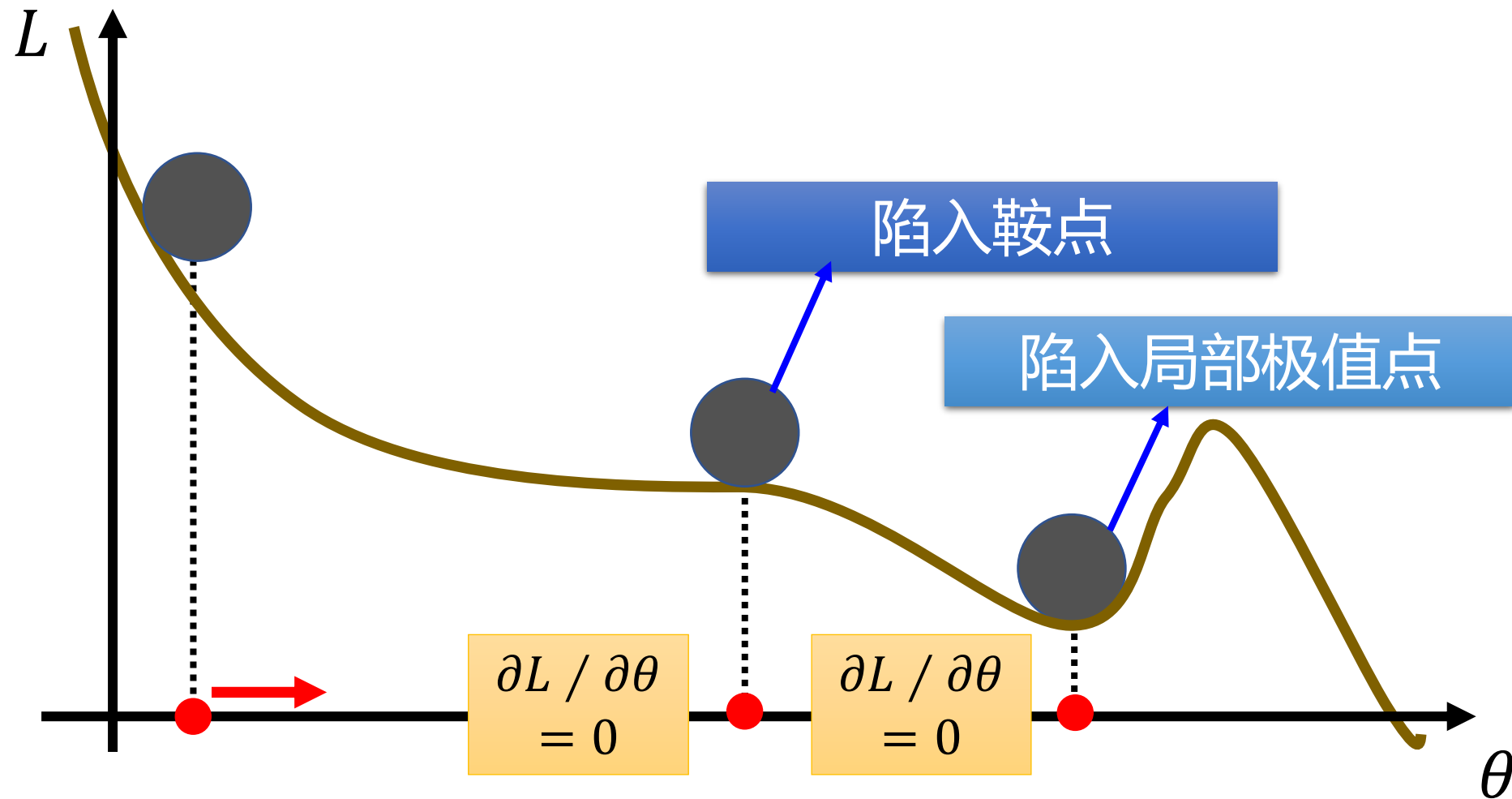
\vdots

$$w_i^{t+1} \leftarrow w_i^t - \frac{a}{\sigma_i^t} g_i^t$$

$$\sigma_i^t = \sqrt{\beta(\sigma_i^{t-1})^2 + (1 - \beta)(g_i^t)^2 + \delta}$$



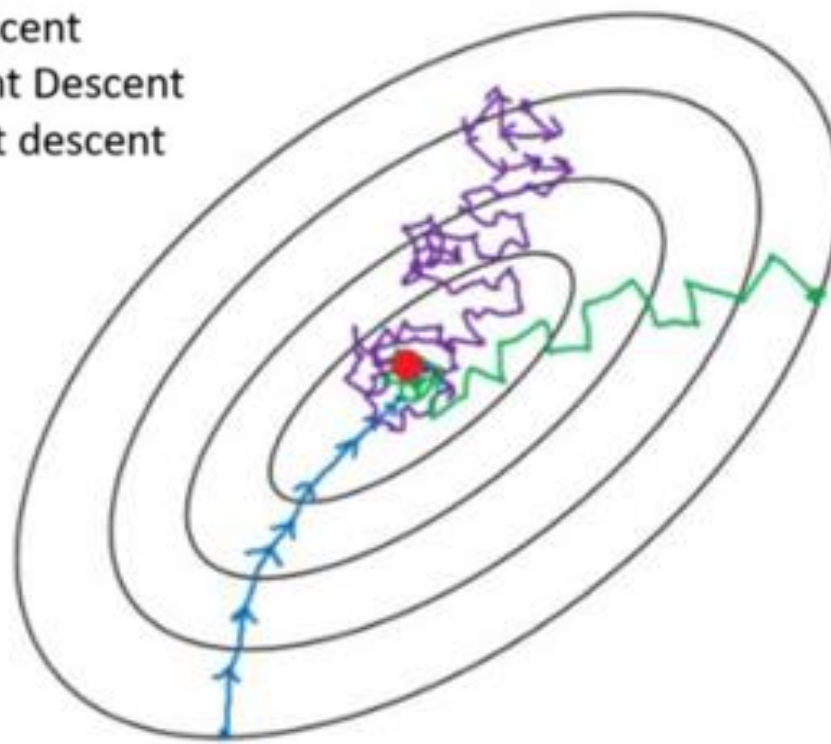
如何逃离局部极值点或鞍点？





在小批量梯度下降中，如果每次选取样本数量比较小，损失会呈现震荡的方式下降，如何缓解震荡？

— Batch gradient descent
— Mini-batch gradient Descent
— Stochastic gradient descent






动量法

一种有效地缓解梯度估计随机性的方式是通过使用最近一段时间内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，从而提高优化速度（增加批量大小也是缓解随机性的一种方式）。

动量法是用之前积累动量来替代真正的梯度。每次迭代的梯度可看作加速度。第t次迭代，计算负梯度的“加权移动平均”作为参数的更新方向：

$$v^t = \lambda v^{t-1} - \alpha \mathbf{g}^{t-1}$$



动量因子 学习率

$$\theta^t \leftarrow \theta^{t-1} - v^t$$

$$v^0 = 0$$

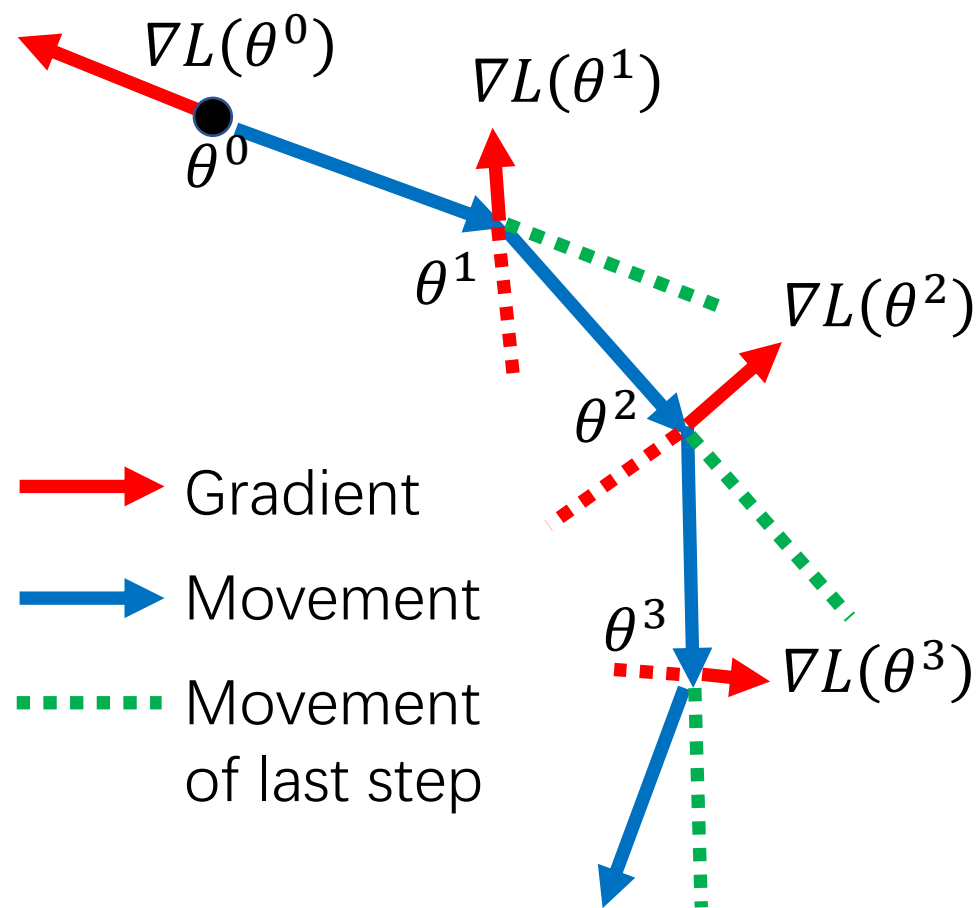
$$v^1 = -\alpha \mathbf{g}^0$$

$$v^2 = -\lambda \alpha \mathbf{g}^0 - \alpha \mathbf{g}^1$$

⋮



动量法



从起始位置 θ^0 开始

计算速度更新 $v^0=0$

在 θ^0 处计算梯度

计算速度更新 $v^1 = \lambda v^0 - \alpha \nabla L(\theta^0)$

应用更新 $\theta^1 = \theta^0 + v^1$

在 θ^1 处计算梯度

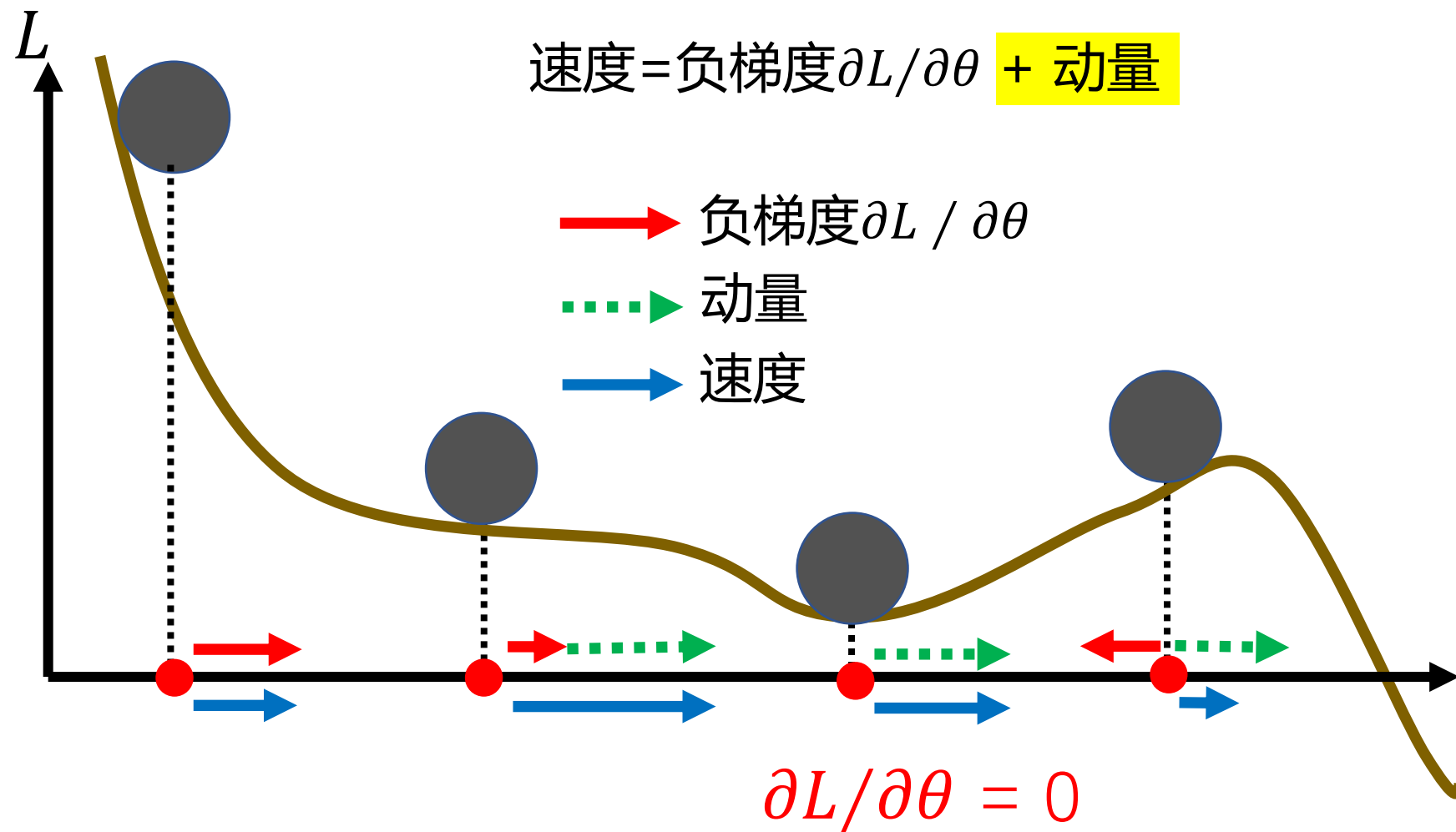
计算速度更新 $v^2 = \lambda v^1 - \alpha \nabla L(\theta^1)$

应用更新 $\theta^2 = \theta^1 + v^2$

当前时刻的速度不仅依赖于负梯度，还依赖于前序时刻的加权移动平均



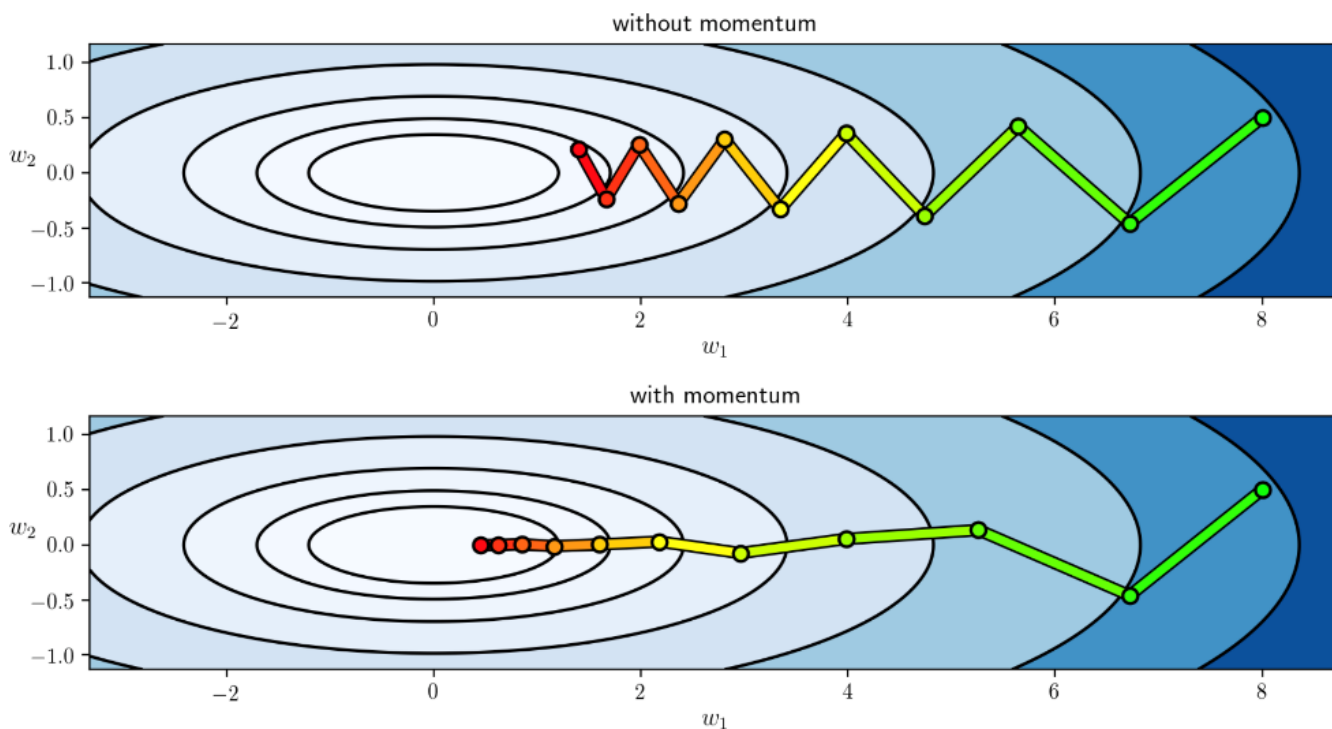
仍然不能保证收敛到全局最优，但有一定可能跳出局部极值点





动量法

一般而言，在迭代初期，梯度方向都比较一致，动量法会起到加速作用，可以更快地到达最优点。在迭代后期，梯度方向会不一致，在收敛值附近震荡，动量法会起到减速作用，增加稳定性。





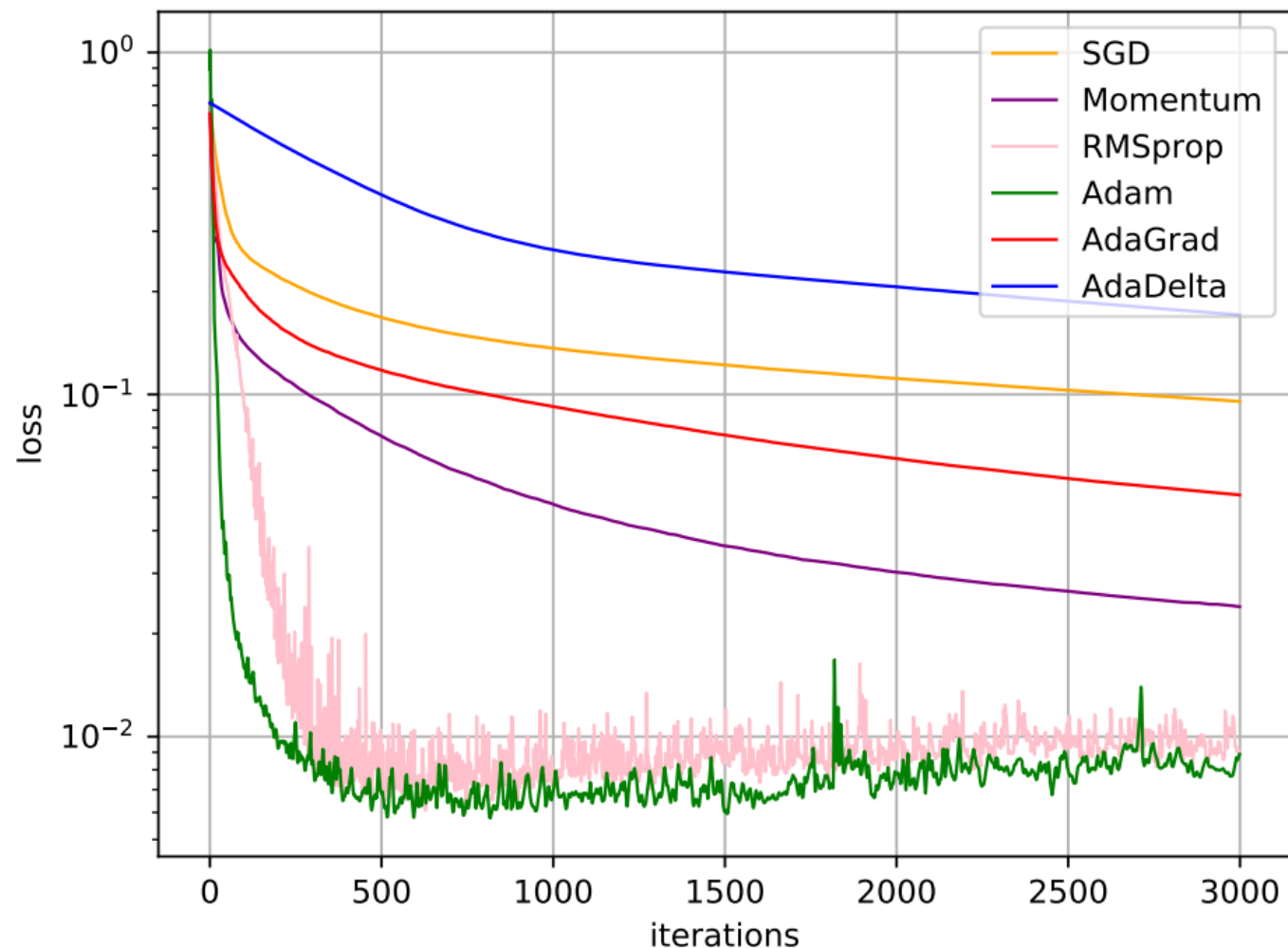
Adam算法：自适应学习率（RMSprop）+ 梯度方向优化（动量法）

- 更新有偏一阶矩估计：
$$v^t = \lambda v^{t-1} - \alpha \mathbf{g}^{t-1}$$
- 更新有偏二阶矩估计：
$$G^t = \beta G^{t-1} - (1 - \beta) \mathbf{g}^t \odot \mathbf{g}^t$$
- 修正一阶矩的偏差：
$$\hat{v}^t = \frac{v^t}{1 - \gamma_1^t}$$
- 修正二阶矩的偏差：
$$\hat{G}^t = \frac{G^t}{1 - \gamma_2^t}$$
- 更新参数：
$$\theta^t = \theta^{t-1} - \frac{\alpha}{\sqrt{\hat{G}^t + \delta}} \hat{v}^t$$



各优化算法比较

在MNIST数据集上收敛性的比较（学习率为0.001，批量大小为128）



- RMSProp和Adam更好
- 目前最流行并且使用很高的优化算法包括SGD、具动量的SGD、RMSProp、具动量的RMSProp和Adam。

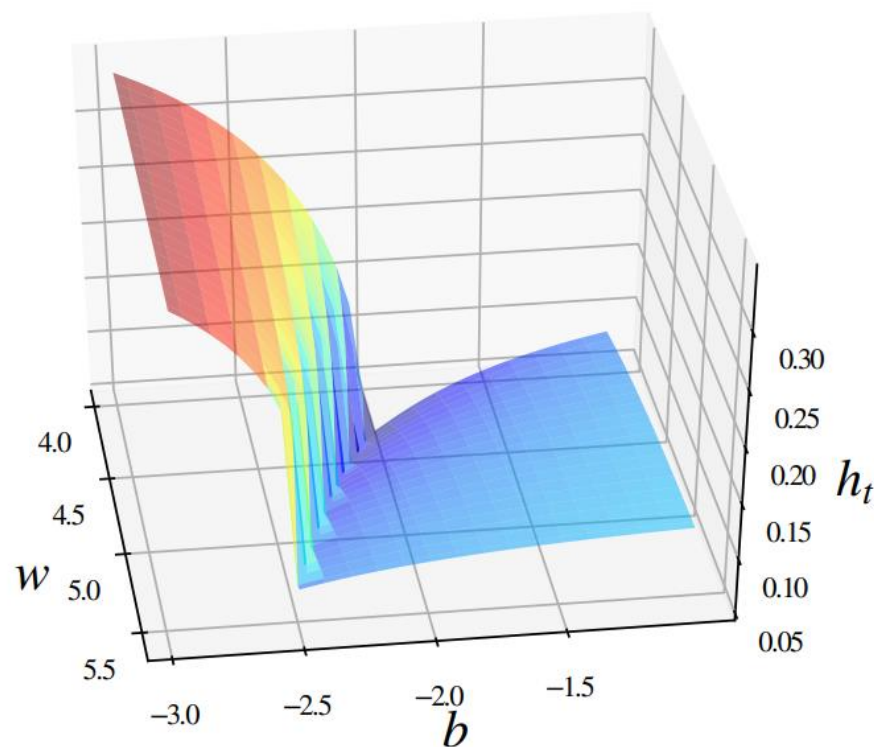


梯度截断

除了梯度消失之外，**梯度爆炸**是影响学习效率的主要因素。在基于梯度下降的优化过程中，如果梯度突然增大，用大的梯度更新参数反而会导致其远离最优点。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**

➤ 按值截断 $\mathbf{g}^t = \max(\min(\mathbf{g}^t, b), a)$

➤ 按模截断 $\mathbf{g}^t = \frac{b}{\|\mathbf{g}^t\|} \mathbf{g}^t$



梯度爆炸问题示例



5.3 学习率与梯度优化





5.1 网络优化

5.2 小批量梯度下降

5.3 学习率与梯度优化

5.4 参数初始化与数据预处理

5.5 逐层归一化

5.6 超参数优化

5.7 过拟合与正则化



5.4 参数初始化与数据预处理

梯度下降法需要在开始训练时给每一个参数赋一个**初始值**。

➤ 初始化为0：对称权重问题

所有参数为0 → 神经元的输出相同 → BP梯度相同 → 参数更新相同 → 参数相同

➤ 初始化太小：梯度消失，使得Sigmoid型激活函数丢失非线性的能力

➤ 初始化太大：梯度爆炸，使得Sigmoid型激活函数变得饱和



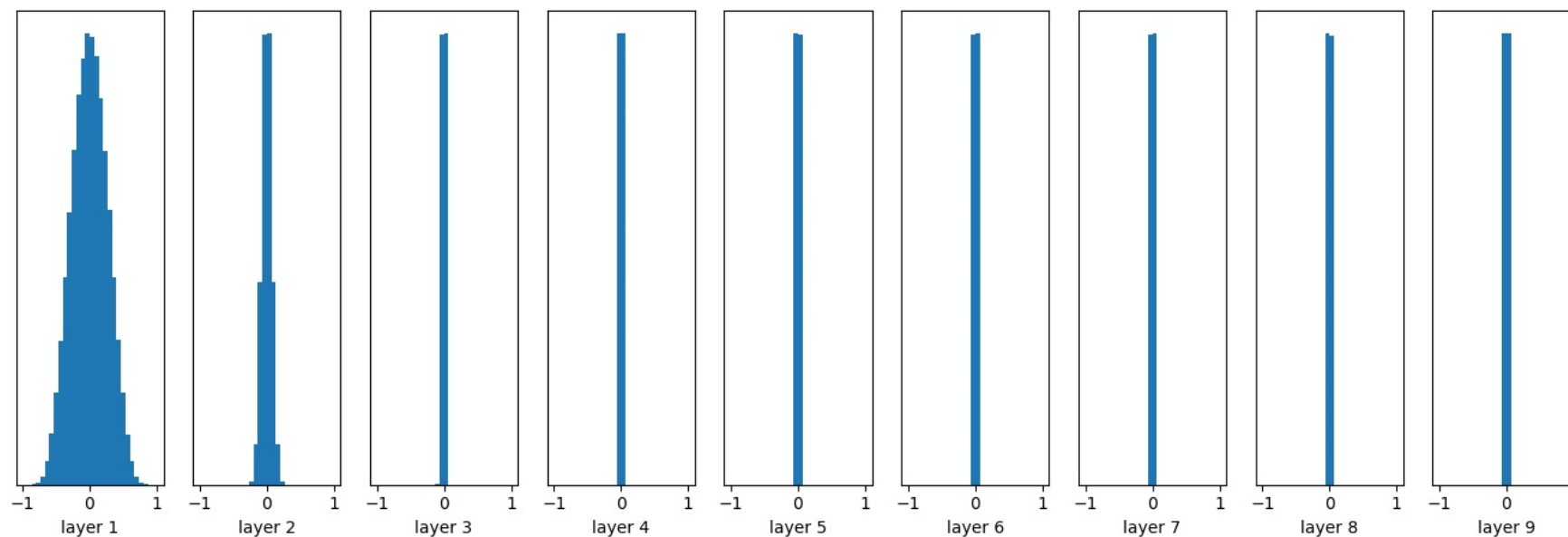
初始化方法：

- 预训练初始化：Fine-Tuning
- 固定值初始化：偏置（bias）通常用0初始化
- 随机初始化方法
 - 基于固定方差的参数初始化
 - 基于方差缩放的参数初始化
 - 正交初始化方法



基于固定方差的参数初始化

- 高斯分布初始化：从一个固定均值和方差的高斯分布进行随机初始化。
- 均匀分布初始化：在一个区间 $[-r, r]$ 内采用均匀分布来初始化参数。



输出信
号消失!

10层网络, tanh, 每一层的参数都是随机正态分布 $(0, 0.01)$ 每一层输出值分布的直方图



基于方差缩放的参数初始化

要高效地训练神经网络，给参数选取一个合适的随机初始化区间是非常重要的。一般而言，参数初始化的区间应该根据神经元的性质进行差异化的设置。如果一个神经元的输入连接很多，它的每个输入连接上的权重就应该小一些，以避免神经元的输出过大（当激活函数为 ReLU 时）或过饱和（当激活函数为 Sigmoid 函数时）。

初始化一个深度网络时，为了缓解梯度消失或爆炸问题，我们尽可能保持每个神经元的输入和输出的方差一致，根据神经元的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）。



基于方差缩放的参数初始化：Xavier初始化

假设前一层有 n^{l-1} 个神经元输出（即 l 层的净输入）：
$$z^l = \sum_{i=1}^{n^{(l-1)}} w_i^l a_i^{(l-1)}$$

为了避免激活值变得饱和，尽量使得 z 处于激活函数的线性区间： $a^l = f(z^l) \approx z^l$

a^l 均值和方差：
$$E[a^l] = E\left[\sum_{i=1}^{n^{(l-1)}} w_i^l a_i^{(l-1)}\right] = \sum_{i=1}^{n^{(l-1)}} E[w_i^l] E[a_i^{(l-1)}] = 0$$

$$\text{var}[a^l] = \sum_{i=1}^{n^{(l-1)}} \text{var}[w_i^l] \text{var}[a_i^{(l-1)}] = n^{(l-1)} \text{var}[w_i^l] \text{var}[a_i^{(l-1)}]$$

保持每个神经元的输入和输出的方差一致：
$$\text{var}[w_i^l] = \frac{1}{n^{(l-1)}}$$

同理，反向传播中，为了使误差信号也不被放大或缩小：
$$\text{var}[w_i^l] = \frac{1}{n^{(l)}}$$

前向反向传播折中：
$$\text{var}[w_i^l] = \frac{2}{n^{(l)} + n^{(l-1)}}$$



基于方差缩放的参数初始化：Xavier初始化

在计算出参数的理想方差后，可以通过高斯分布或均匀分布来随机初始化参数：

➤ 高斯分布： $N(0, \frac{2}{n^{(l)} + n^{(l-1)}})$

➤ 均匀分布： $r = \sqrt{\frac{6}{n^{(l)} + n^{(l-1)}}}$

虽然在 Xavier 初始化中我们假设激活函数为恒等函数，但是 Xavier 初始化也适用于 Logistic 函数和 Tanh 函数（处于激活函数的线性区间）

➤ Logistic 函数在线性区间的斜率约为 0.25： $N(0, 16 \times \frac{2}{n^{(l)} + n^{(l-1)}})$



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， 因此其分布的方差也近似为使用恒等函数时的一半。 这样， 只考虑前向传播时， 参数 $w_i^{(l)}$ 的理想方差为

$$\text{var}(w_i^{(l)}) = \frac{2}{M_{l-1}},$$

其中， M_{l-1} 是第 $l - 1$ 层神经元个数。

因此当使用 ReLU 激活函数时，

若采用高斯分布来初始化参数 $w_i^{(l)}$ ， 其方差为 $2/M_{l-1}$ ；

若采用区间为 $[-r, r]$ 的均分分布来初始化参数 $w_i^{(l)}$ ， 则 $r = \sqrt{6/M_{l-1}}$



基于方差缩放的参数初始化：正交初始化

- Xavier 初始化和 He 初始化的具体设置情况。

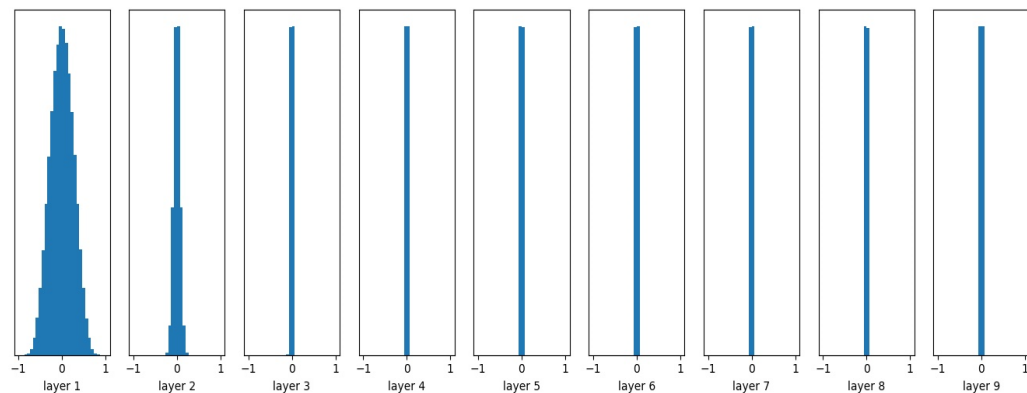
初始化方法	激活函数	均匀分布 $[-r, r]$	高斯分布 $\mathcal{N}(0, \sigma^2)$
Xavier 初始化	Logistic	$r = 4\sqrt{\frac{6}{M_{l-1}+M_l}}$	$\sigma^2 = 16 \times \frac{2}{M_{l-1}+M_l}$
Xavier 初始化	Tanh	$r = \sqrt{\frac{6}{M_{l-1}+M_l}}$	$\sigma^2 = \frac{2}{M_{l-1}+M_l}$
He 初始化	ReLU	$r = \sqrt{\frac{6}{M_{l-1}}}$	$\sigma^2 = \frac{2}{M_{l-1}}$

➤ 正交初始化

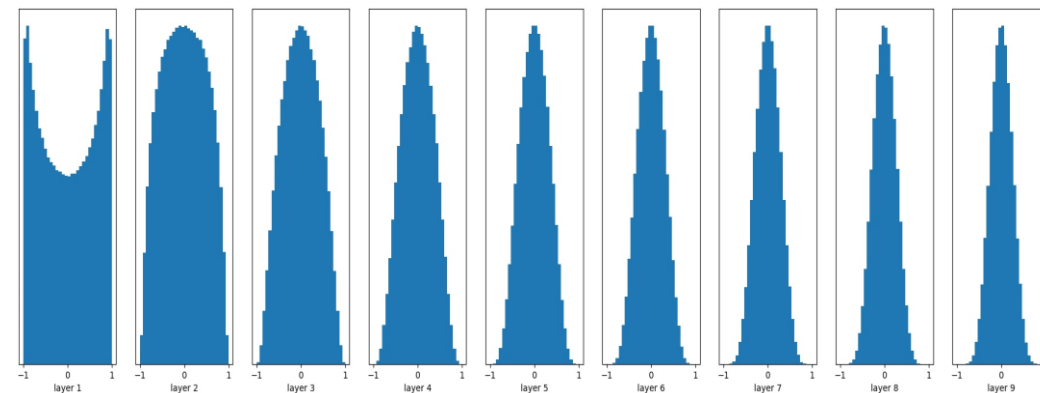
- 用均值为 0、方差为 1 的高斯分布初始化一个矩阵；
- 将这个矩阵用奇异值分解得到两个正交矩阵，并使用其中之一作为权重矩阵。



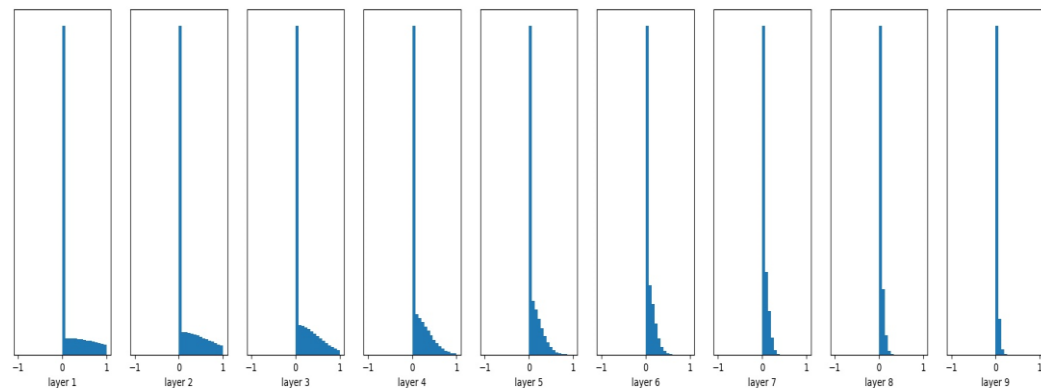
基于方差缩放的参数初始化



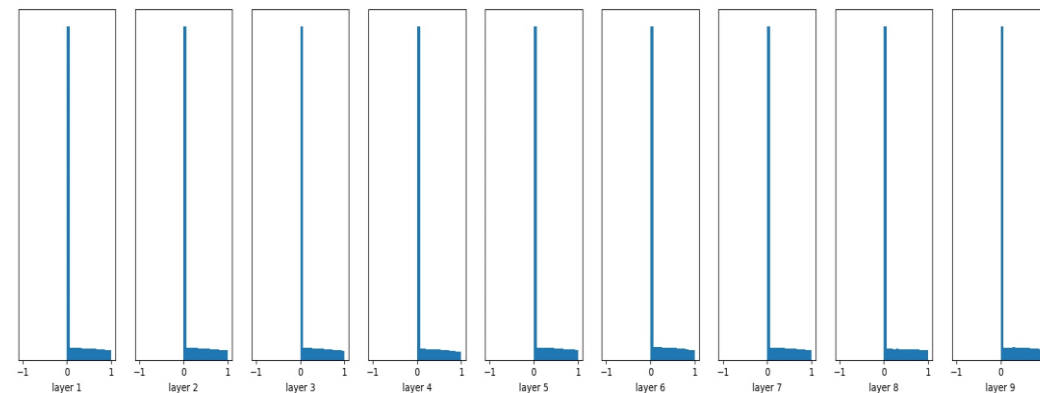
tanh, 高斯分布初始化



tanh, Xavier初始化



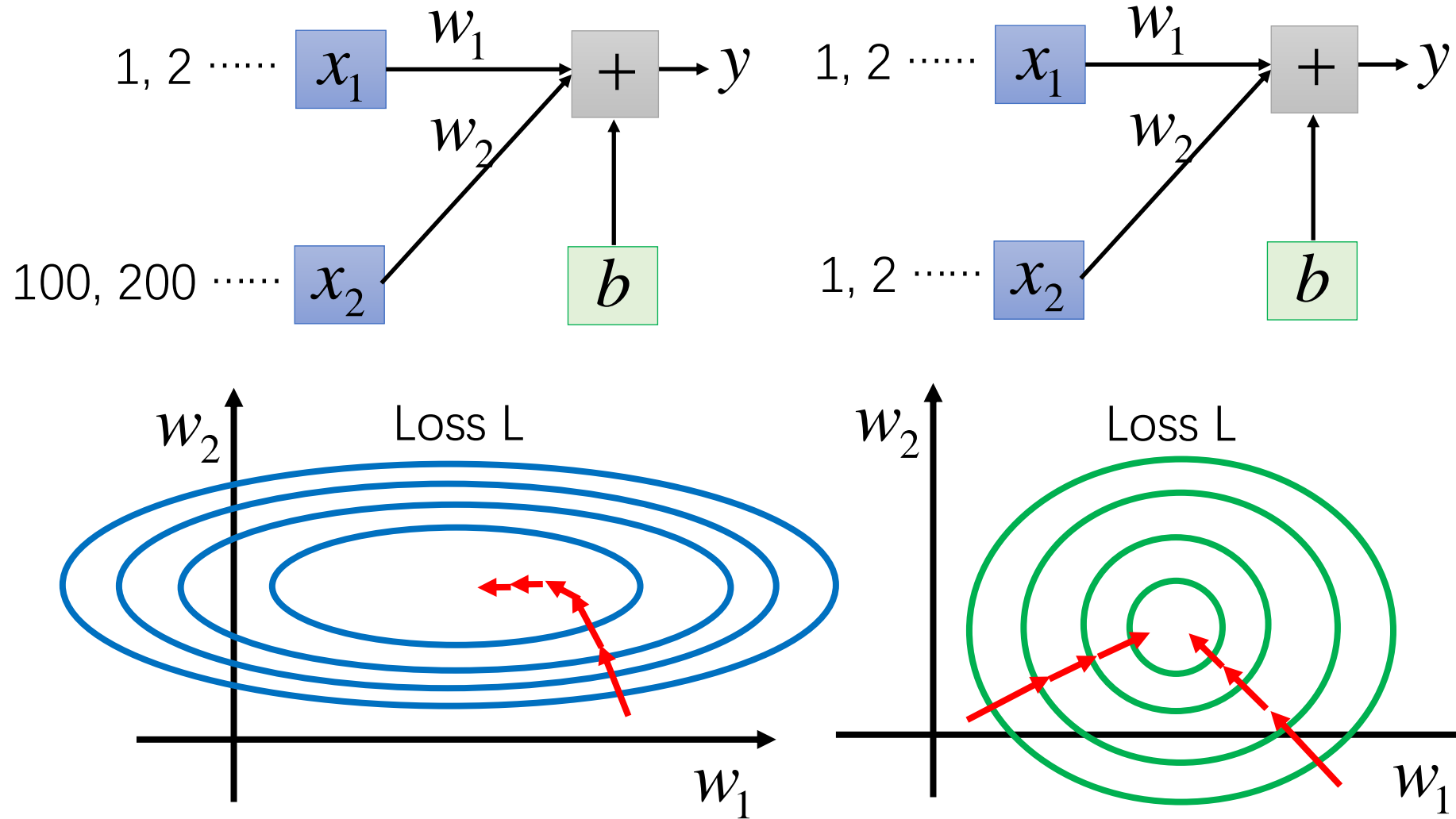
ReLU, Xavier初始化



ReLU, He初始化



► 数据尺度不一样对优化的影响





第一步 数据归一化

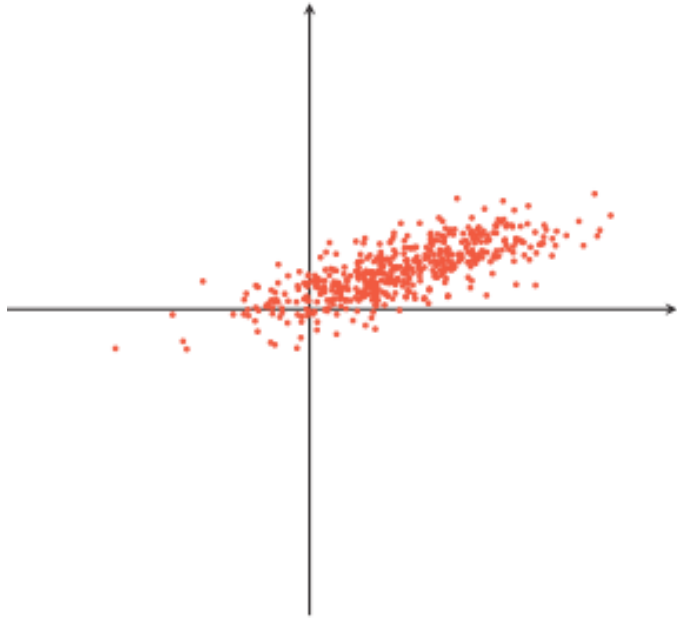
- 简单缩放 如：最小最大值归一化 $x_i \leftarrow \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$;
图像像素值除以 255;
- 逐样本均值消减 (也称为移除直流分量): 若数据是平稳的 (即数据每一个维度的统计都服从相同分布), 那么可以考虑在每个样本上减去数据的统计平均值 (逐样本计算)。
- 特征标准化: (独立地) 使得数据的每一个维度具有零均值和单位方差。

$$x_i \leftarrow \frac{x_i - \mu_i}{\sigma_i}$$

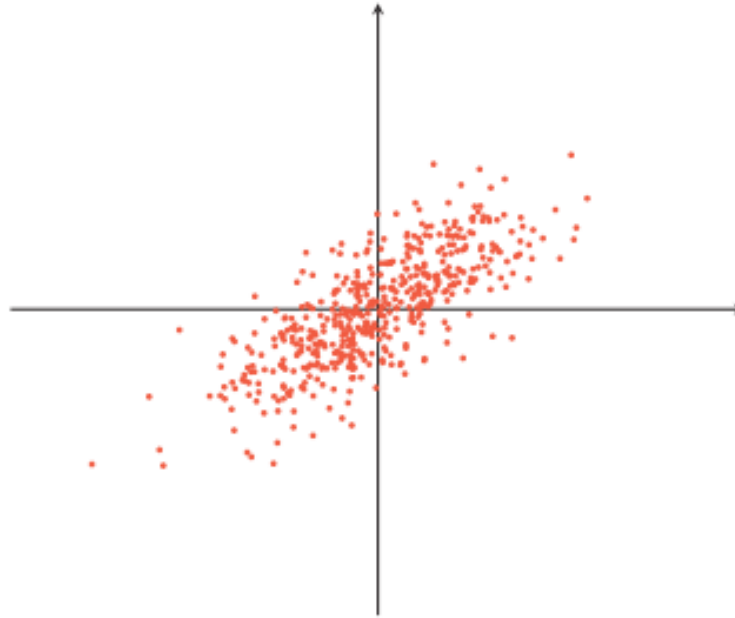


第二步 白化，降低输入的冗余性，降低特征之间的相关性，使得所有的特征具有相同的方差。

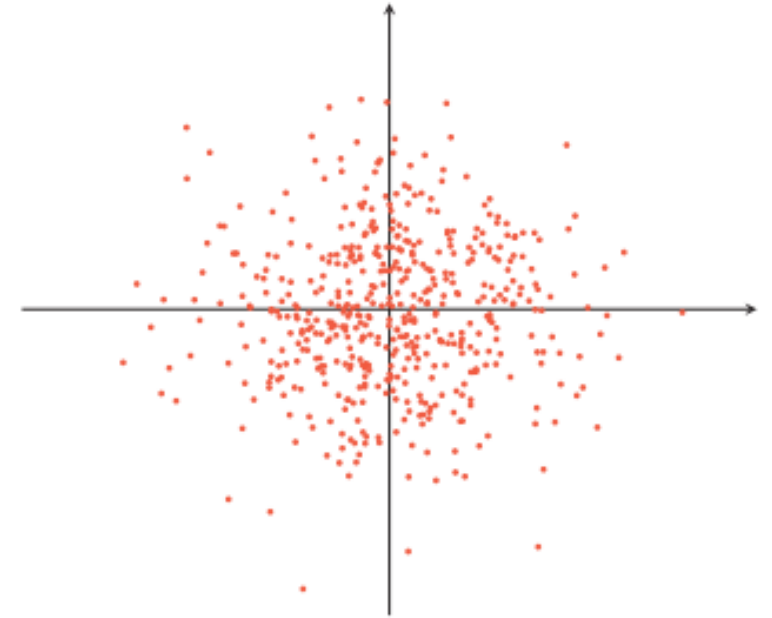
原始数据



标准归一化



PCA 白化





实际建议

- 自然灰度图像：均值消减->PCA/ZCA白化
- 彩色图像：简单缩放->PCA/ZCA白化
- 音频 (MFCC/频谱图)：特征标准化->PCA/ZCA 白化
- MNIST 手写数字：简单缩放/逐样本均值消减 (->PCA/ZCA 白化)

输入数据归一化了，中间输出数据如何？

逐层归一化



5.1 网络优化

5.2 小批量梯度下降

5.3 学习率与梯度优化

5.4 参数初始化与数据预处理

5.5 逐层归一化

5.6 超参数优化

5.7 过拟合与正则化



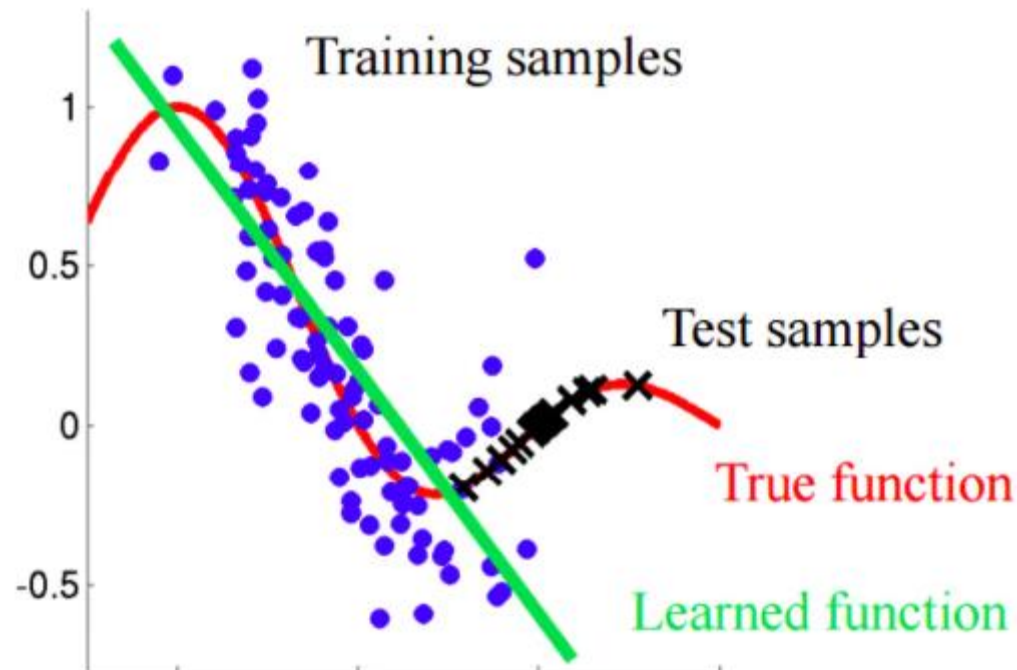
5.5 逐层归一化

目的:

- 解决内部协变量偏移问题
- 解决梯度消失、梯度爆炸
- 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)





批量归一化

批量归一化是对一个中间层的单个神经元进行归一化操作

- 第 l 层, 对于 K 个样本的一个小批量集合 $Z^{(l)} = [\mathbf{z}^{(1,l)}, \dots, \mathbf{z}^{(K,l)}]$ 其均值、方差为:

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

- 批量归一化定义为:

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

- 小批量样本的数量不能太小
- 不能处理神经元的净输入的分布在神经网络中是动态变化 (RNN)



层归一化

层归一化是对一个中间层的所有神经元进行归一化

➤ 第 l 层神经网络的净输入为 $\mathbf{z}^{(l)}$, 其均值方差为:

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

M_l 是第 l 层神经元个数

➤ 层归一化定义为:

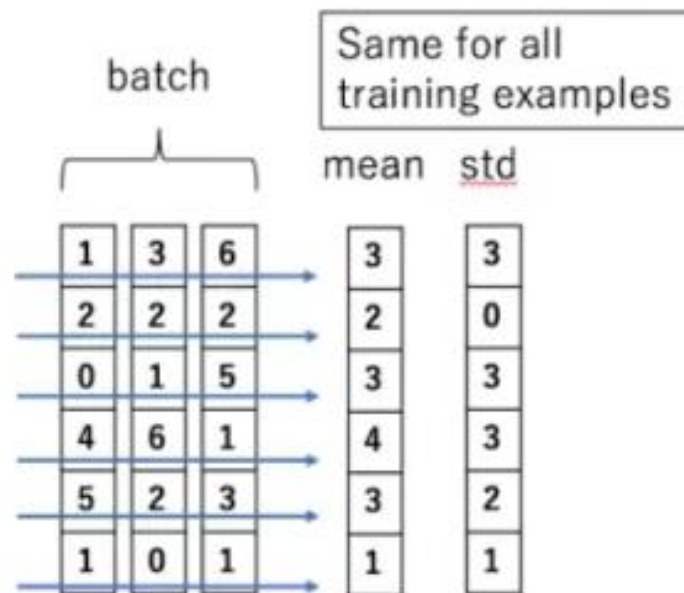
$$\hat{\mathbf{z}}^{(l)} = \frac{\mathbf{z}^{(l)} - \mu^{(l)}}{\sqrt{\sigma^{(l)^2} + \epsilon}} \odot \boldsymbol{\gamma} + \boldsymbol{\beta}$$
$$\triangleq \text{LN}_{\boldsymbol{\gamma}, \boldsymbol{\beta}}(\mathbf{z}^{(l)}),$$



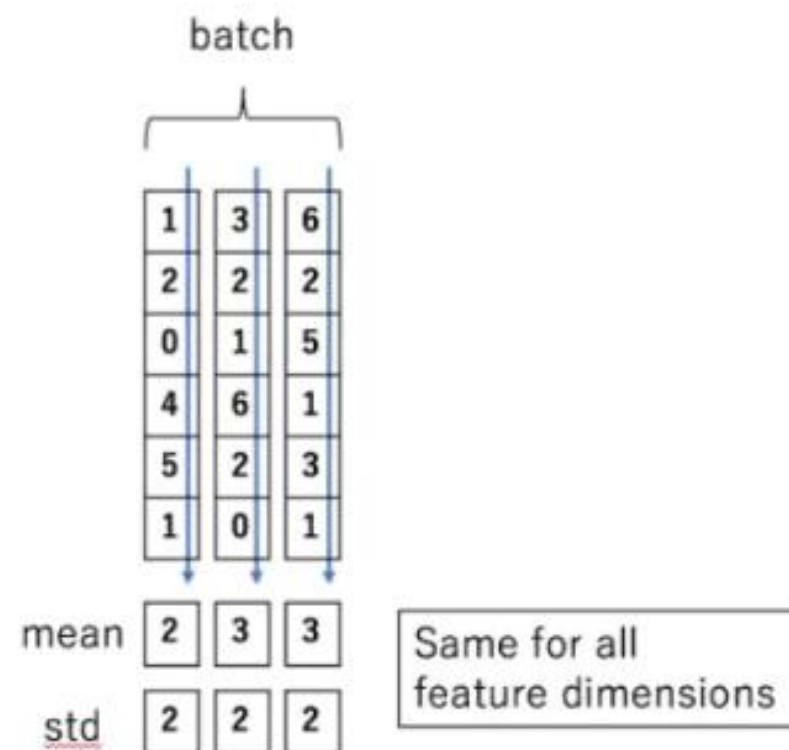
批量归一化 vs 层归一化

- 对于 K 个样本的一个小批量集合 $\mathbf{Z}^{(l)} = [\mathbf{z}^{(1,l)}; \dots; \mathbf{z}^{(K,l)}]$
- 层归一化是对其每一列进行归一化，而批量归一化是对每一行进行归一化。批量归一化是一种更好的选择。当样本数量比较小时，可选择层归一化。

Batch Normalization



Layer Normalization





5.1 网络优化

5.2 小批量梯度下降

5.3 学习率与梯度优化

5.4 参数初始化与数据预处理

5.5 逐层归一化

5.6 超参数优化

5.7 过拟合与正则化



5.6 超参数优化

神经网络中的超参数：

- 网络结构（神经元间连接关系、层数、每层神经元个数、激活函数等）
- 、优化参数（优化方法、学习率、mini-batch 大小）、正则化系数

优化难点：

- 超参数优化是一个组合优化问题：无法像用梯度下降法来优化
- 评估一组超参数配置的时间代价非常高

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



网格搜索 (Grid Search) 是一种通过尝试所有超参数的组合来寻找合适一组超参数配置的方法。

假设总共有 K 个超参数, 第 k 个超参数的可以取 m_k 个值。比如学习率 α , 可以设置: $\alpha \in \{0.01, 0.1, 0.5, 1.0\}$.

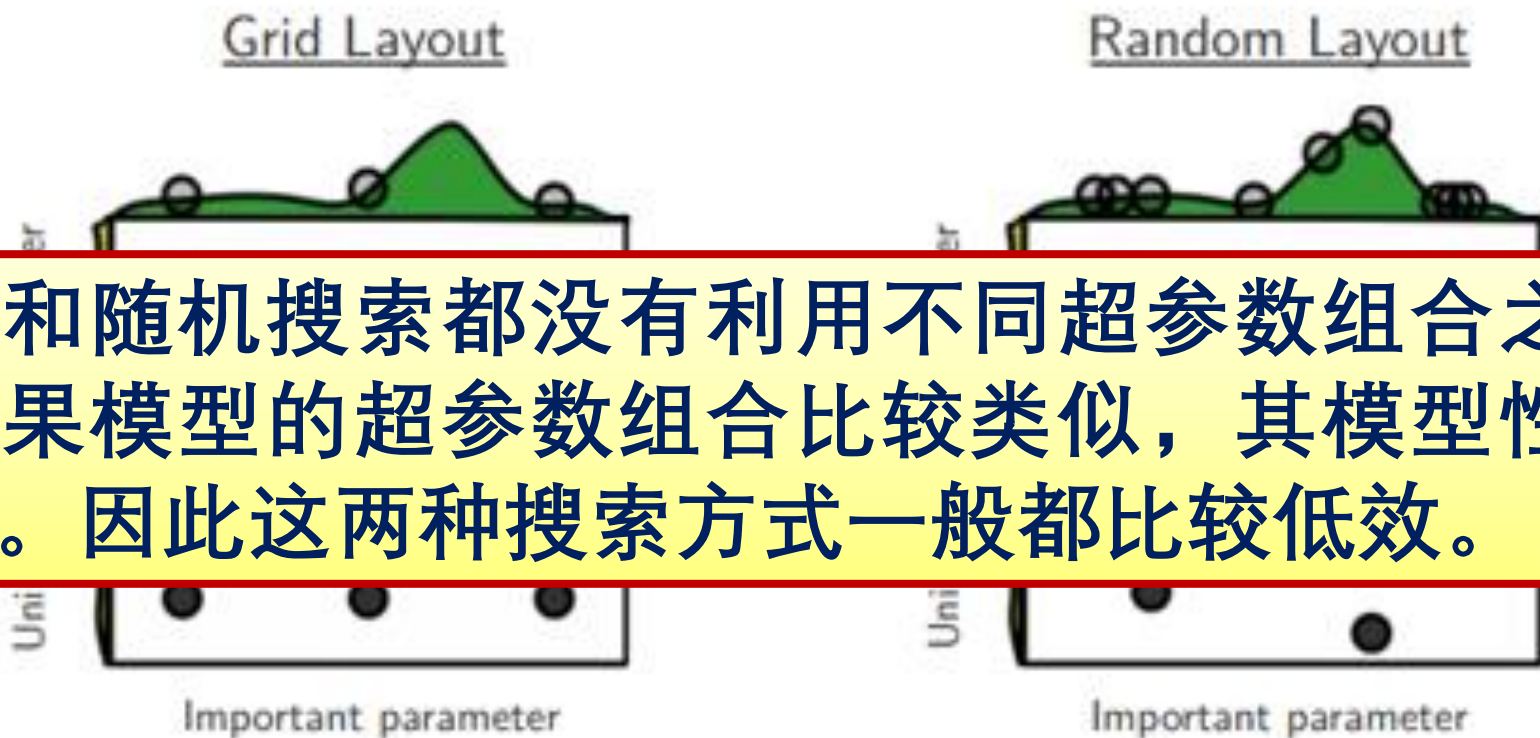
这些超参数可以有 $m_1 \times m_2 \times \cdots \times m_K$ 个取值组合。

网格搜索根据这些超参数的不同组合分别训练一个模型, 然后测试这些模型在验证集上的性能, 选取一组性能最好的配置。



随机搜索

超参数对模型性能影响程度不一样。采用网格搜索会在不重要的超参数上进行不必要的尝试。一种改进方法是对超参数进行随机组合，然后选取一个性能最好的配置，这就是随机搜索。



网格搜索和随机搜索都没有利用不同超参数组合之间的相关性，即如果模型的超参数组合比较类似，其模型性能也是比较接近的。因此这两种搜索方式一般都比较低效。



贝叶斯优化是一种自适应的超参数优化方法，根据当前已经试验的超参数组合，来预测下一个可能带来最大收益的组合。

- 假设超参数优化的函数 $f(x)$ 服从高斯过程，则 $p(f(x)|x)$ 为一个正态分布
- 贝叶斯优化过程是根据已有的 N 组试验结果 $H = \{x_n, y_n\}_{n=1}^N$ (y_n 为 $f(x_n)$ 的观测值) 来建模高斯过程，并计算 $f(x)$ 的后验分布 $p_{GP}(f(x)|x, H)$
- 需用尽可能少的样本使得后验分布接近真实分布，定义一个收益函数 $a(x, H)$
- 收益函数来判断一个样本是否能够给建模后验概率提供更多的收益，收益函数的定义有很多种方式，一个常用的是期望改善



一种比较常用的贝叶斯优化方法为时序模型优化 (SMBO)

算法 7.1 时序模型优化 (SMBO) 方法

输入: 优化目标函数 $f(\mathbf{x})$, 迭代次数 T , 收益函数 $a(x, \mathcal{H})$

- 1 $\mathcal{H} \leftarrow \emptyset$;
 - 2 随机初始化高斯过程, 并计算 $p_{\mathcal{GP}}(f(\mathbf{x})|\mathbf{x}, \mathcal{H})$;
 - 3 **for** $t \leftarrow 1$ **to** T **do**
 - 4 $\mathbf{x}' \leftarrow \arg \max_{\mathbf{x}} a(\mathbf{x}, \mathcal{H})$;
 - 5 评价 $y' = f(\mathbf{x}')$;
 - 6 $\mathcal{H} \leftarrow \mathcal{H} \cup (\mathbf{x}', y')$;
 - 7 根据 \mathcal{H} 重新建模高斯过程, 并计算 $p_{\mathcal{GP}}(f(\mathbf{x})|\mathbf{x}, \mathcal{H})$;
 - 8 **end**
- 输出:** \mathcal{H}
-



超参数优化中，每组超参数配置的评估代价比较高。如果可以在较早的阶段就可以估计出一组配置的效果会比较差，那么就可以中止这组配置的评估，将更多的资源留给其它配置。

算法 7.2 一种逐次减半的动态资源分配方法

输入: 预算 B , N 个超参数配置 $\{\mathbf{x}_n\}_{n=1}^N$

- 1 $T \leftarrow \lceil \log_2(N) \rceil - 1$;
- 2 随机初始化 $\mathcal{S}_0 = \{\mathbf{x}_n\}_{n=1}^N$;
- 3 **for** $t \leftarrow 1$ **to** T **do**
 - 4 $r_t \leftarrow \lfloor \frac{B}{|\mathcal{S}_t| \times T} \rfloor$;
 - 5 给 \mathcal{S}_t 中的每组配置分配 r_t 的资源;
 - 6 运行 \mathcal{S}_t 所有配置, 评估结果为 \mathbf{y}_t ;
 - 7 根据评估结果, 选取 $|\mathcal{S}_t|/2$ 组最优的配置 $\mathcal{S}_t \leftarrow \arg \max(\mathcal{S}_t, \mathbf{y}_t, |\mathcal{S}_t|/2)$;
// $\arg \max(\mathcal{S}, \mathbf{y}, m)$ 为从集合 \mathcal{S} 中选取 m 个元素, 对应最优的 m 个评估结果.
- 8 **end**

输出: 最优配置 \mathcal{S}_K



5.1 网络优化

5.2 小批量梯度下降

5.3 学习率与梯度优化

5.4 参数初始化与数据预处理

5.5 逐层归一化

5.6 超参数优化

5.7 过拟合与正则化

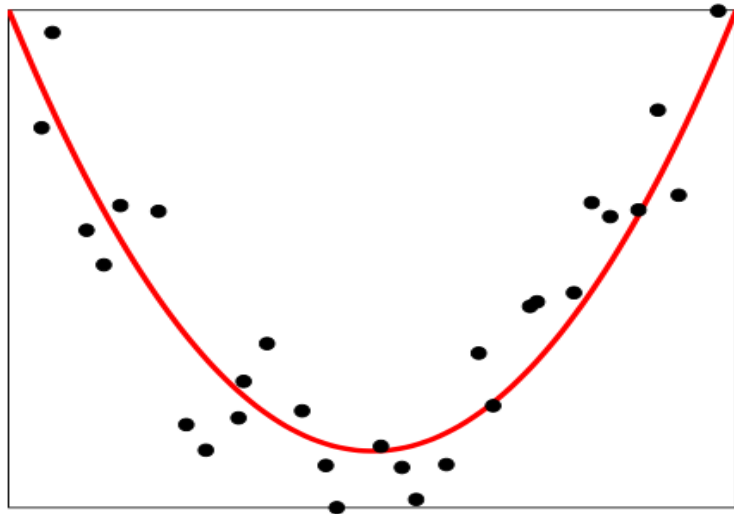


5.7 过拟合与正则化

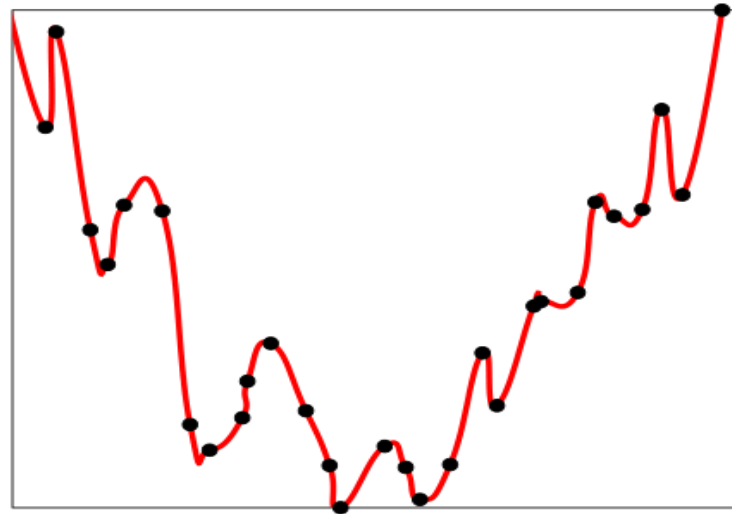
如何提高神经网络的泛化能力：

- ℓ_1 和 ℓ_2 正则化
- 提前停止
- Dropout
- 数据增强

正常



过拟合





ℓ_1 和 ℓ_2 正则化

ℓ_1 和 ℓ_2 正则化是最常用的正则化方法，通过约束参数的 ℓ_1 和 ℓ_2 范数来减小模型在训练数据集上的过拟合现象。

优化问题可以写成：

$$L'(\theta) = \underbrace{L(\theta)} + \lambda \underbrace{\ell_p(\theta)}$$

原始损失函数

范数函数， p 的取值通常为 $\{1, 2\}$ 代表 ℓ_1 和 ℓ_2 范数



ℓ_1 和 ℓ_2 正则化

ℓ_2 正则化: $L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_2^2 \quad \frac{\partial L'}{\partial \theta} = \frac{\partial L}{\partial \theta} + \lambda \theta$

$$\theta^{t+1} \rightarrow \theta^t - \alpha \frac{\partial L'}{\partial \theta} = \theta^t - \alpha \left(\frac{\partial L}{\partial \theta} + \lambda \theta^t \right) = (1 - \alpha \lambda) \theta^t - \alpha \frac{\partial L}{\partial \theta}$$

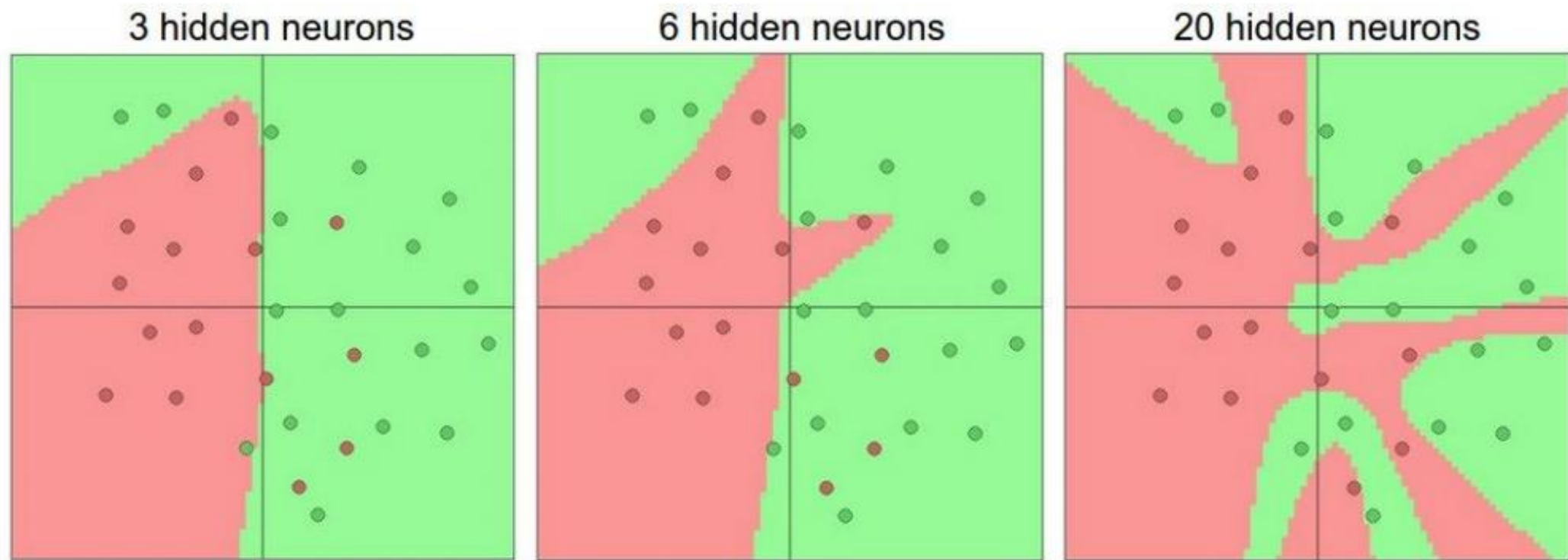
权重衰减

ℓ_1 正则化: $L'(\theta) = L(\theta) + \lambda \frac{1}{2} \|\theta\|_1 \quad \frac{\partial L'}{\partial \theta} = \frac{\partial L}{\partial \theta} + \lambda \text{sgn}(\theta)$

$$\theta^{t+1} \rightarrow \theta^t - \alpha \frac{\partial L'}{\partial \theta} = \theta^t - \alpha \left(\frac{\partial L}{\partial \theta} + \lambda \text{sgn}(\theta^t) \right) = \theta^t - \alpha \frac{\partial L}{\partial \theta} - \alpha \lambda \text{sgn}(\theta^t)$$



隐藏层的不同神经元个数:

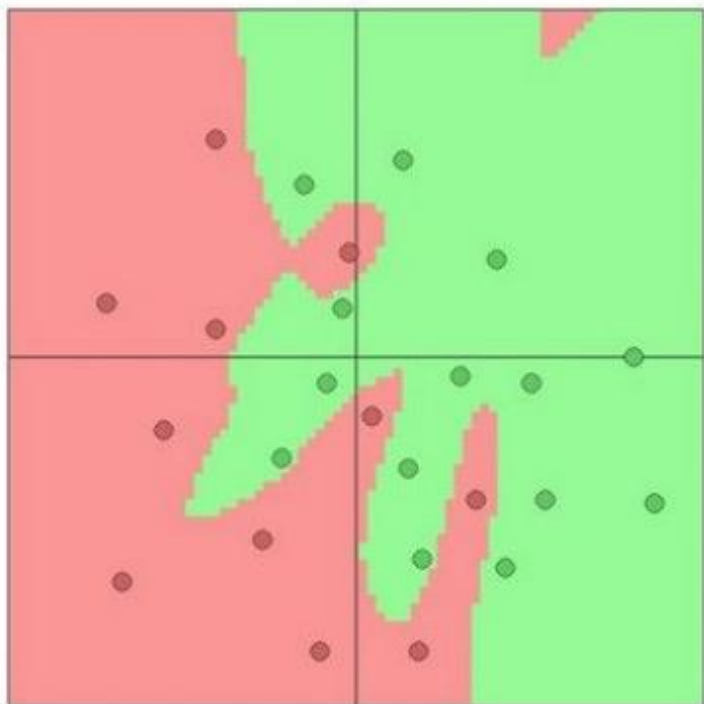




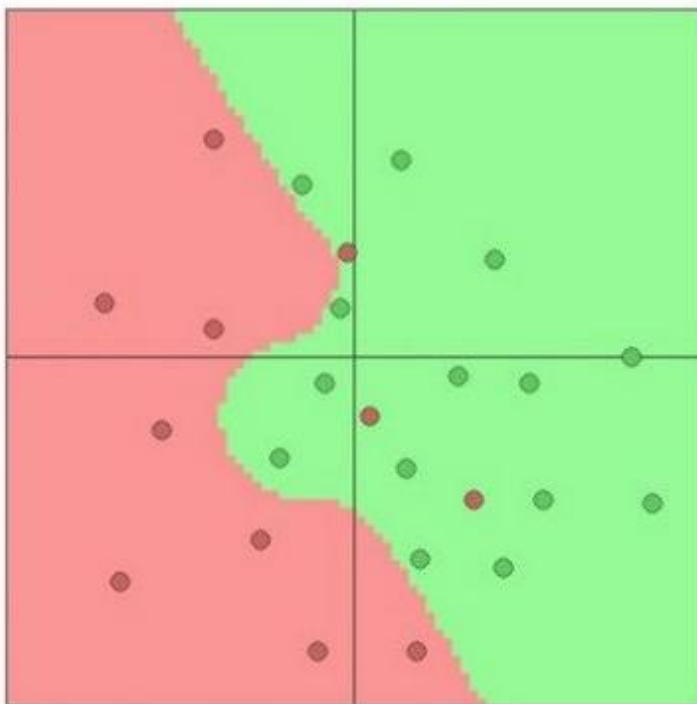
ℓ_1 和 ℓ_2 正则化

不同的正则化系数:

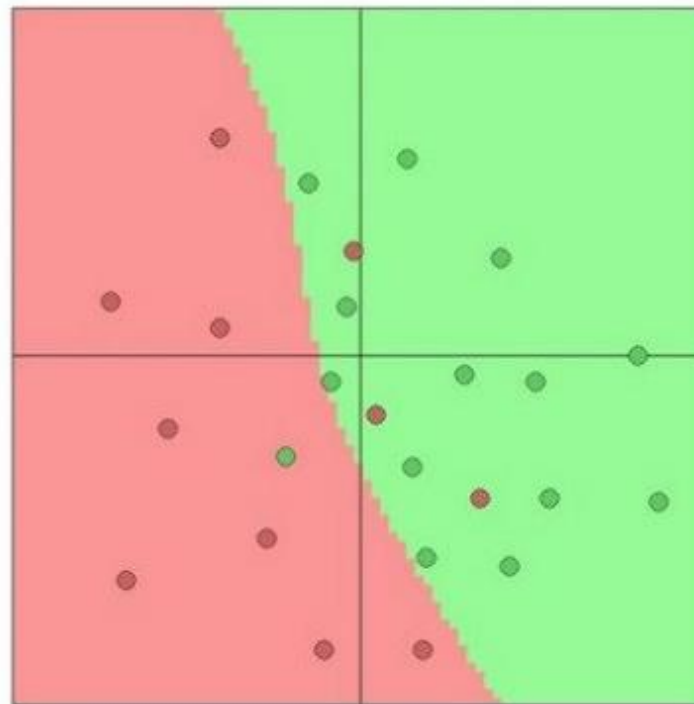
$\lambda = 0.001$



$\lambda = 0.01$



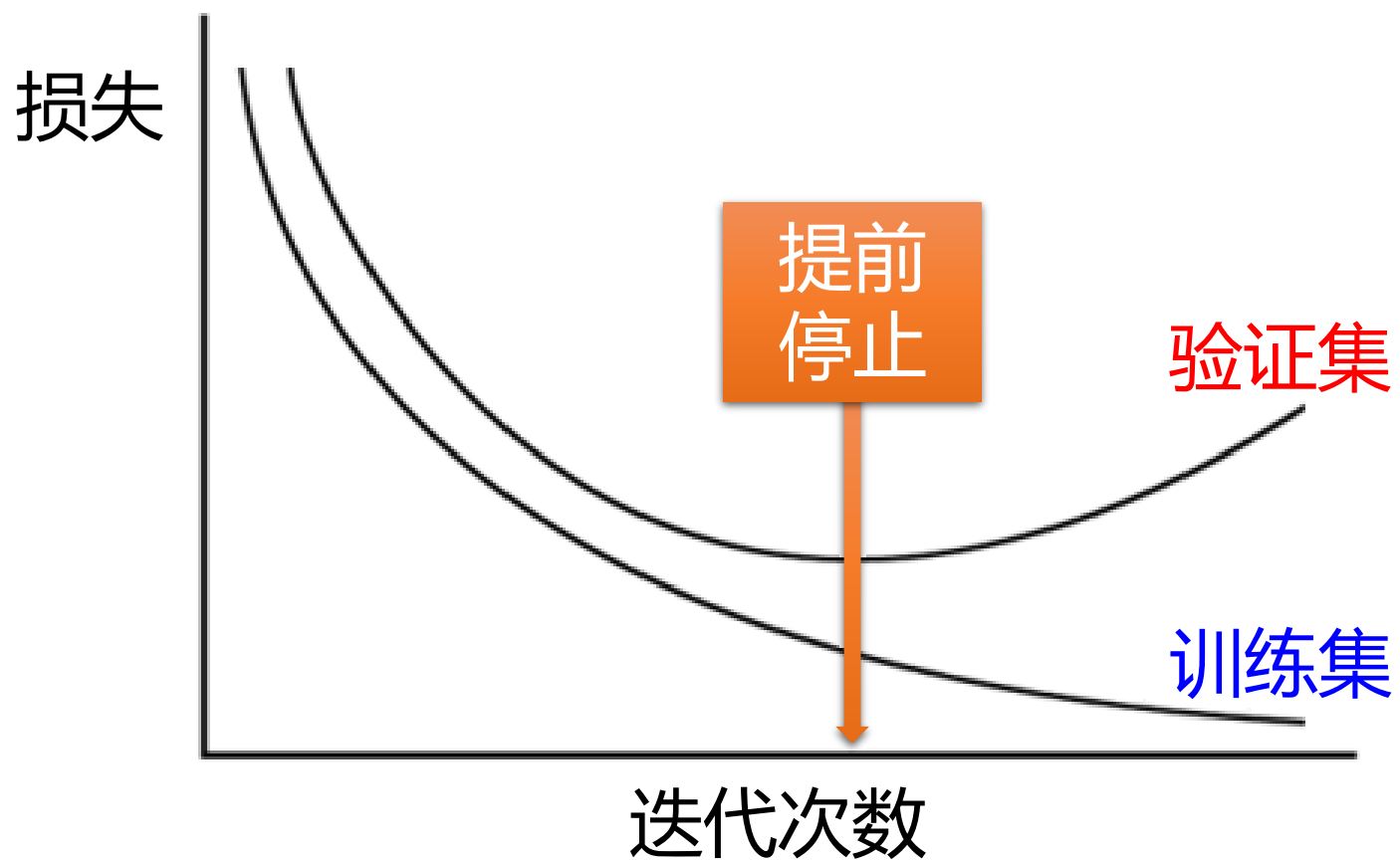
$\lambda = 0.1$





提前停止

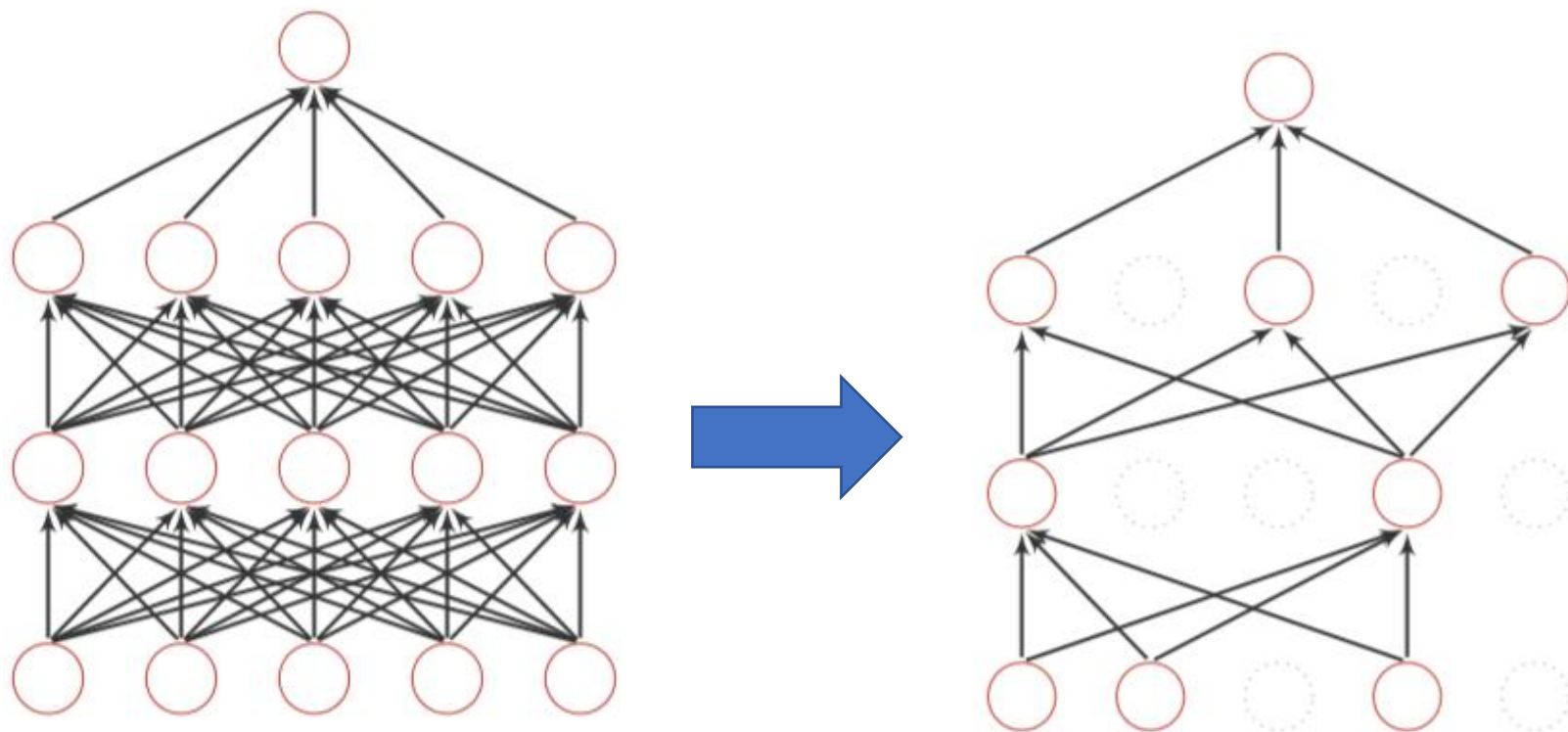
使用一个验证集来测试每一次迭代的参数在验证集上是否最优。如果在验证集上的错误率不再下降，就停止迭代。





Dropout

训练时:

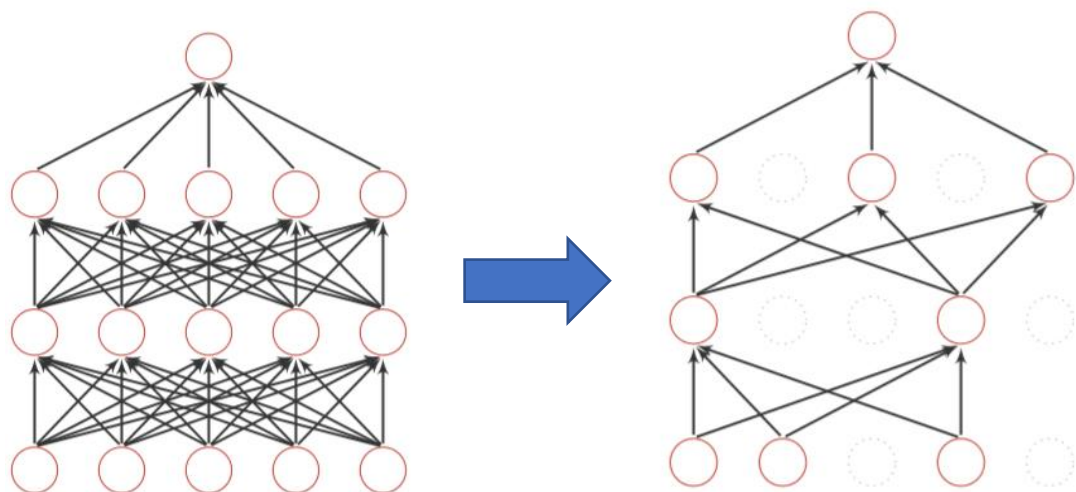


- 在每次更新参数前:
 - 每个神经元以 $p\%$ 的概率被丢弃



Dropout

训练时:



➤ 在每次更新参数前:

- 每个神经元以 $p\%$ 的概率被丢弃

➡ 网络结构发生了改变.

- 使用新的网络训练

测试时:

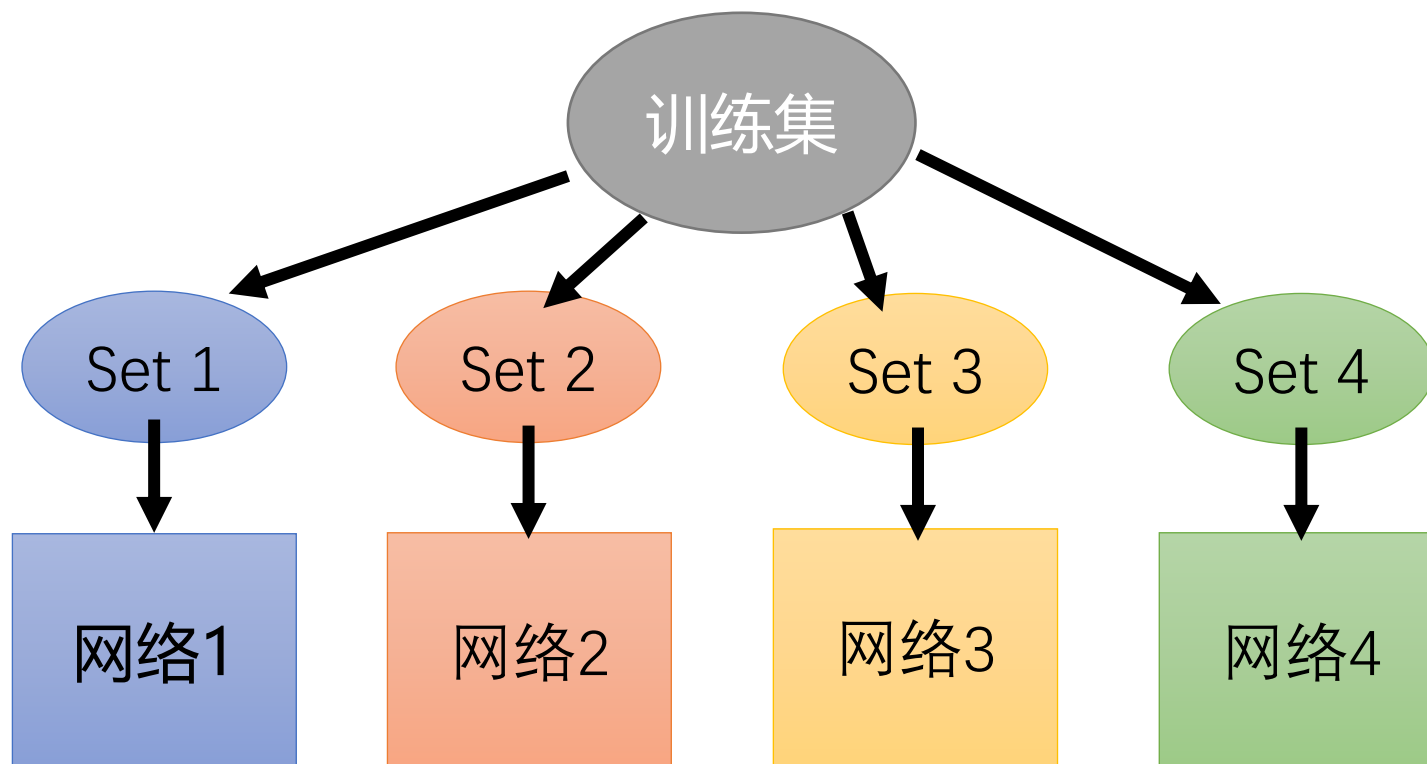
➤ 不用dropout, 每个参数乘以 $1-p\%$



Dropout

Dropout为什么会提升网络优化效果?

- Dropout简化了网络, 防止过拟合
- Dropout可看作是一种集成学习



M 神经元
↓
 2^M 可能的
网络



Dropout

Dropout为什么会提升网络优化效果？

- Dropout简化了网络，防止过拟合
- Dropout可看作是一种集成学习
- Dropout可解释为一种贝叶斯学习的近似

用 $y = f(\mathbf{x}; \theta)$ 来表示要学习的神经网络，贝叶斯学习是假设参数 θ 为随机向量，并且先验分布为 $q(\theta)$ ，贝叶斯方法的预测为：

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

其中 $f(\mathbf{x}; \theta_m)$ 为第 m 次应用dropout后的网络，其中参数 θ_m 为对全部参数 θ 的一次采样



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以及训练数据量。

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。