第13章 人工神经网络

—— 2. 优化算法

卿来云
lyqing@ucas.ac.cn

回忆：梯度下降

- $\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} J(\boldsymbol{\theta})$
- （随机）选择一个初始值$\boldsymbol{\theta}^{(1)}$，$t = 1$
- 计算$\boldsymbol{\theta}^{(t)}$处的梯度：$\boldsymbol{g}^{(t)} = \nabla J(\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$
- 移动方向：在负梯度方向移动一步（步长为学习率η）

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta \boldsymbol{g}^{(t)}$$

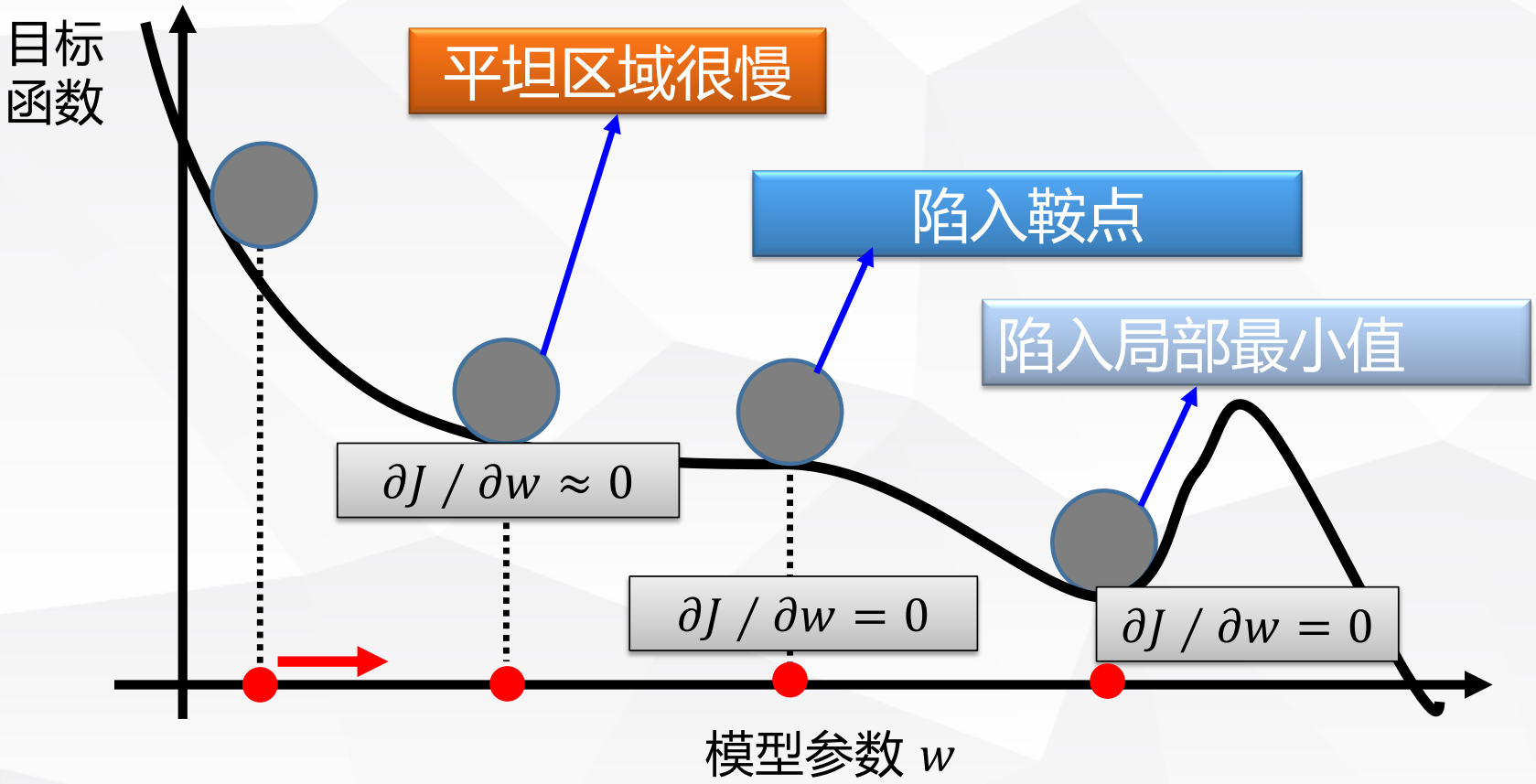
$$t + 1 \leftarrow t$$

深度模型训练

- 梯度下降的改进
  - 小批量随机梯度下降
  - 自适应梯度方向（动量法）
  - 自适应学习率
  - 参数初始化
  - 批正规化
  - 跳跃连接
- 抗过拟合策略

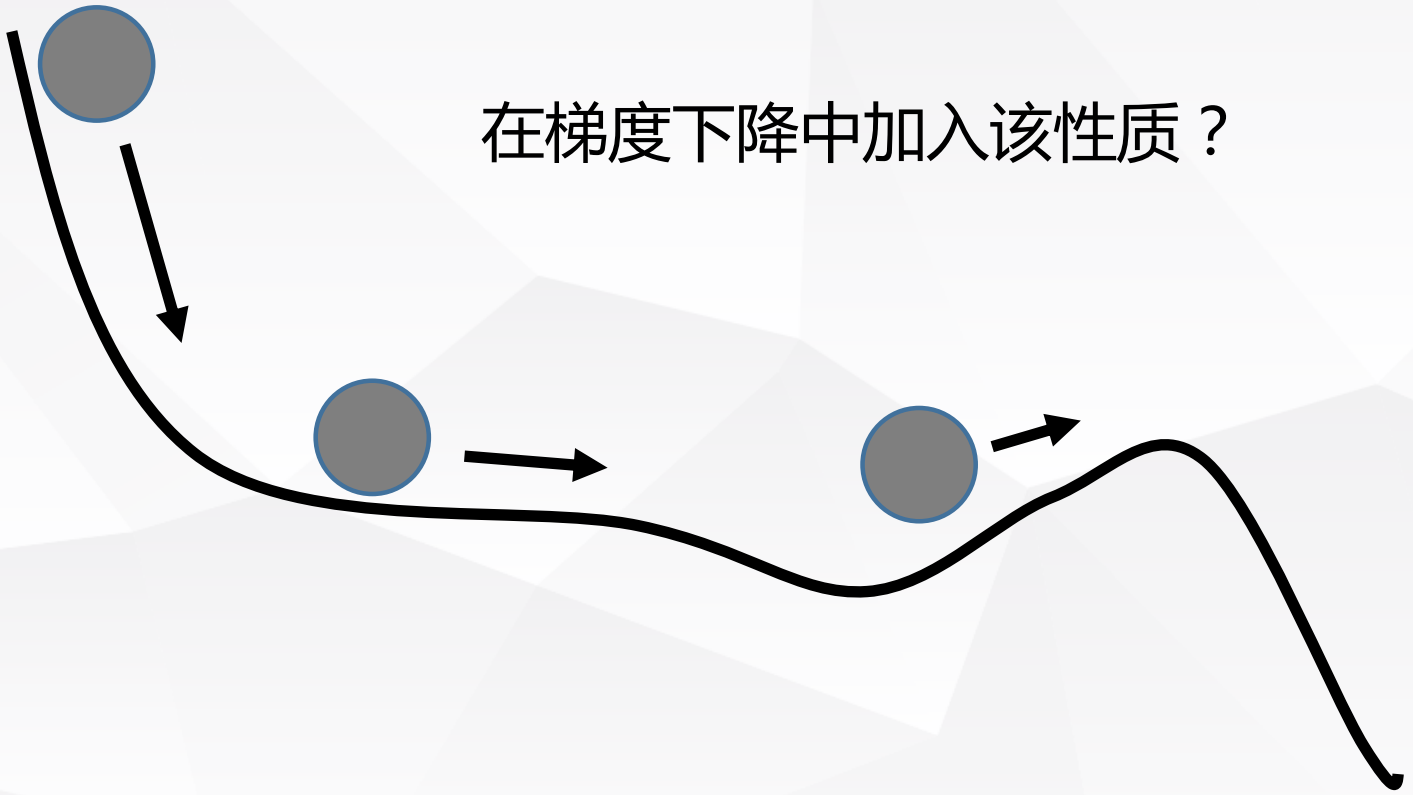
梯度下降

■梯度下降法在有些地方（Hessian矩阵病态）进展缓慢



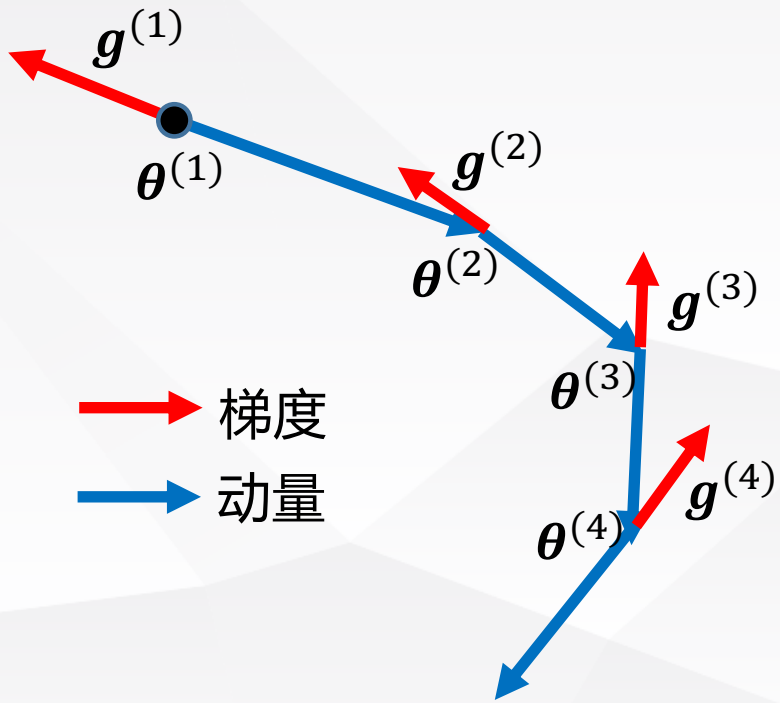
物理世界中 ……

- 动量/惯性



在梯度下降中加入该性质？

回忆：朴素梯度下降



初始位置$\boldsymbol{\theta}^{(1)}$

计算$\boldsymbol{\theta}^{(1)}$处的梯度$\boldsymbol{g}^{(1)}$

移到$\boldsymbol{\theta}^{(2)} = \boldsymbol{\theta}^{(1)} - \eta \boldsymbol{g}^{(1)}$

计算$\boldsymbol{\theta}^{(2)}$处的梯度$\boldsymbol{g}^{(2)}$

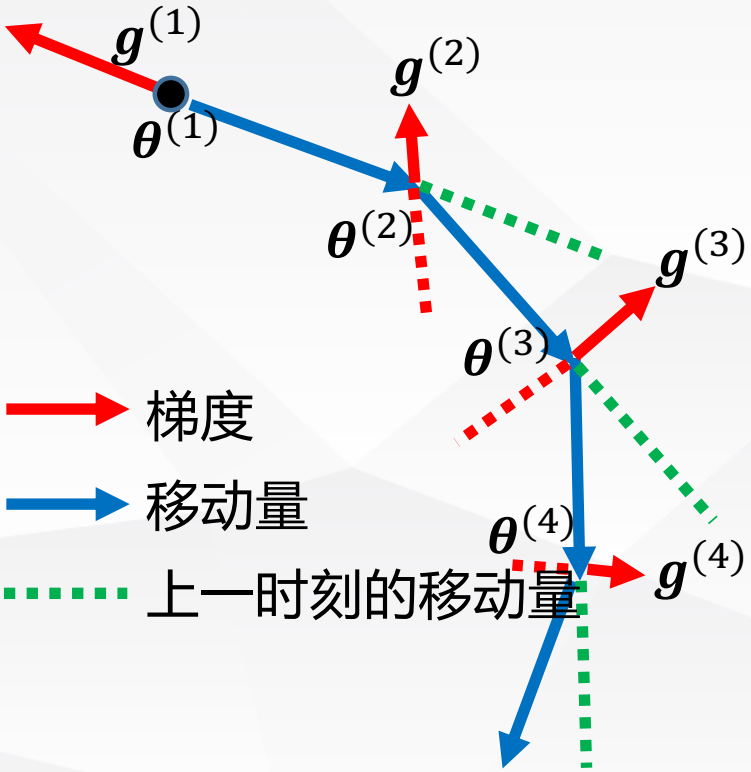
移到$\boldsymbol{\theta}^{(3)} = \boldsymbol{\theta}^{(2)} - \eta \boldsymbol{g}^{(2)}$

⋮

迭代，直到$\boldsymbol{g}^{(t)} \approx 0$

移动方向：动量法

动量：上一时刻的动量减去当前的梯度



初始动量$\boldsymbol{v}^{(0)} = \mathbf{0}$

初始位置$\boldsymbol{\theta}^{(1)}$

计算$\boldsymbol{\theta}^{(1)}$处的梯度$\boldsymbol{g}^{(1)}$

动量$\boldsymbol{v}^{(1)} = \rho\boldsymbol{v}^{(0)} - \eta\boldsymbol{g}^{(1)}$

移到$\boldsymbol{\theta}^{(2)} = \boldsymbol{\theta}^{(1)} + \boldsymbol{v}^{(1)}$

计算$\boldsymbol{\theta}^{(2)}$处的梯度$\boldsymbol{g}^{(2)}$

动量$\boldsymbol{v}^{(2)} = \rho\boldsymbol{v}^{(1)} - \eta\boldsymbol{g}^{(2)}$

移到$\boldsymbol{\theta}^{(3)} = \boldsymbol{\theta}^{(2)} + \boldsymbol{v}^{(2)}$

移动量不仅与梯度有关，还与前一时刻的移动量有关。

动量法

■事实上，$\boldsymbol{v}^{(t)}$ 为之前所有梯度 $\boldsymbol{g}^{(0)}, \boldsymbol{g}^{(1)}, \ldots \boldsymbol{g}^{(t)}$ 的加权和

$$\boldsymbol{v}^{(0)} = 0$$

$$\boldsymbol{v}^{(1)} = \rho \boldsymbol{v}^{(0)} - \eta \boldsymbol{g}^{(1)} = -\eta \boldsymbol{g}^{(1)}$$

$$\boldsymbol{v}^{(2)} = \rho \boldsymbol{v}^{(1)} - \eta \boldsymbol{g}^{(2)} = -\rho\eta \boldsymbol{g}^{(1)} - \eta \boldsymbol{g}^{(2)}$$

$$\vdots$$

$$\boldsymbol{v}^{(t)} = \rho \boldsymbol{v}^{(t-1)} - \eta \boldsymbol{g}^{(t)}$$

$$= \rho\left(-\rho \boldsymbol{v}^{(t-2)} - \eta \boldsymbol{g}^{(t-1)}\right) - \eta \boldsymbol{g}^{(t)}$$

$$= -\eta \sum_{j=1}^{t} \rho^{t-j} \boldsymbol{g}^{(j)}$$

负梯度的指数衰减平均

初始动量$\boldsymbol{v}^{(0)} = \boldsymbol{0}$

初始位置$\boldsymbol{\theta}^{(1)}$

计算$\boldsymbol{\theta}^{(1)}$处的梯度$\boldsymbol{g}^{(1)}$

动量$\boldsymbol{v}^{(1)} = \rho \boldsymbol{v}^{(0)} - \eta \boldsymbol{g}^{(1)}$

移到$\boldsymbol{\theta}^{(2)} = \boldsymbol{\theta}^{(1)} + \boldsymbol{v}^{(1)}$

计算$\boldsymbol{\theta}^{(2)}$处的梯度$\boldsymbol{g}^{(2)}$

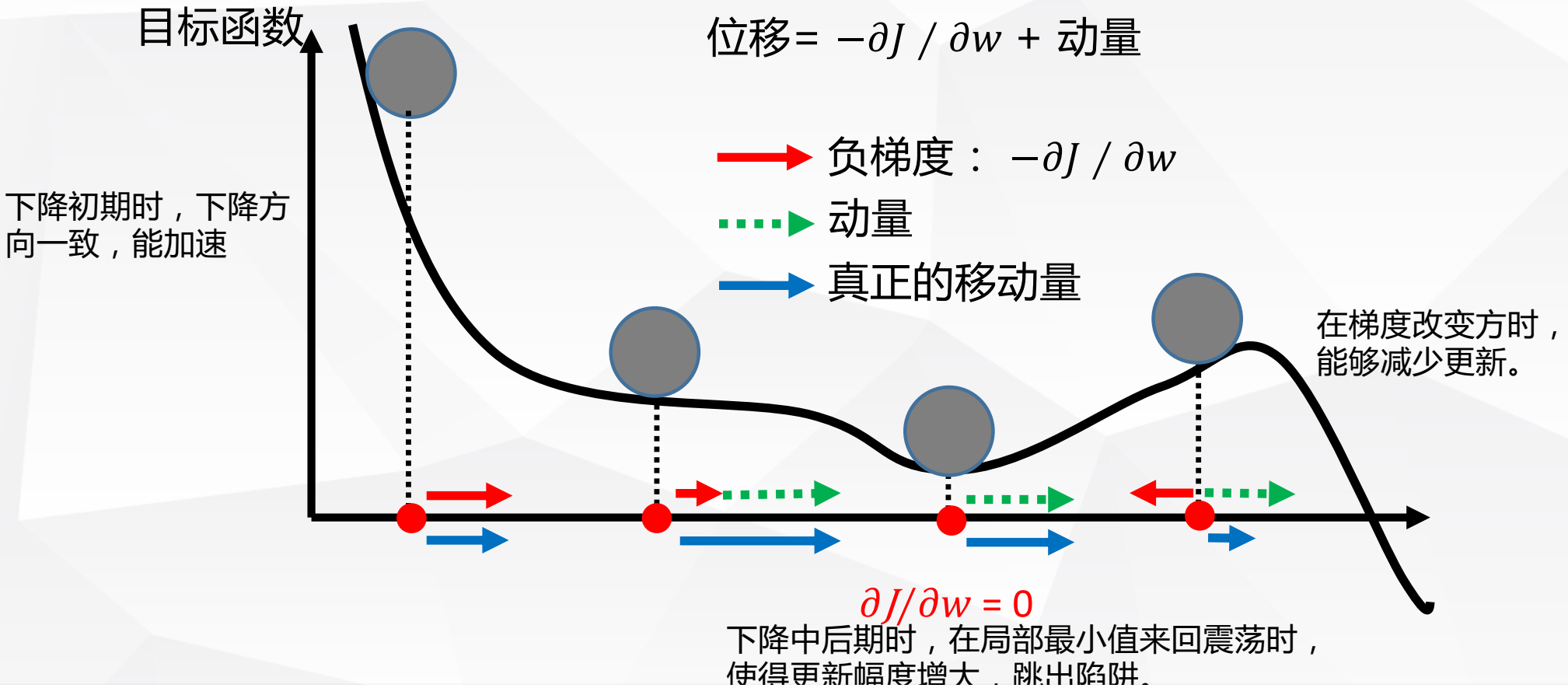
动量$\boldsymbol{v}^{(2)} = \rho \boldsymbol{v}^{(1)} - \eta \boldsymbol{g}^{(2)}$

移到$\boldsymbol{\theta}^{(3)} = \boldsymbol{\theta}^{(2)} + \boldsymbol{v}^{(2)}$

移动量不仅与梯度有关，还与前一时刻的移动量有关。

动量法

仍然不能保证到达全局最小值，
but give some hope ……



Nesterov动量法

■ SGD with Nesterov Momentum（涅斯捷罗夫动量法）

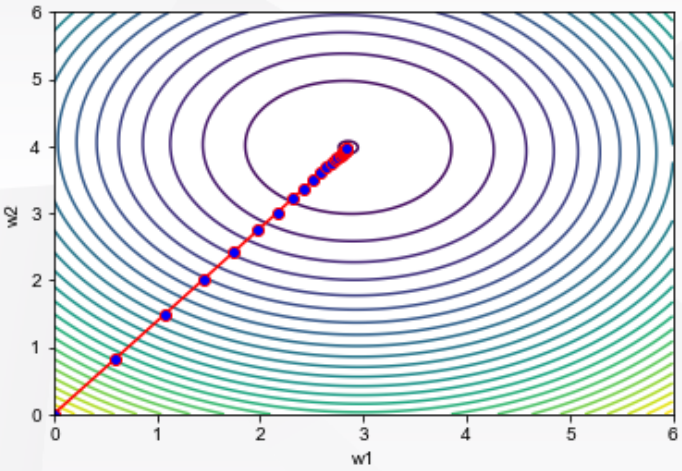
• 不计算当前位置的梯度，而是计算如果按照速度方向走了一步，那个时候的梯度，再与速度一起计算更新方向

梯度下降：$\boldsymbol{v}^{(t)} = -\eta \nabla J(\boldsymbol{\theta}) \big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$

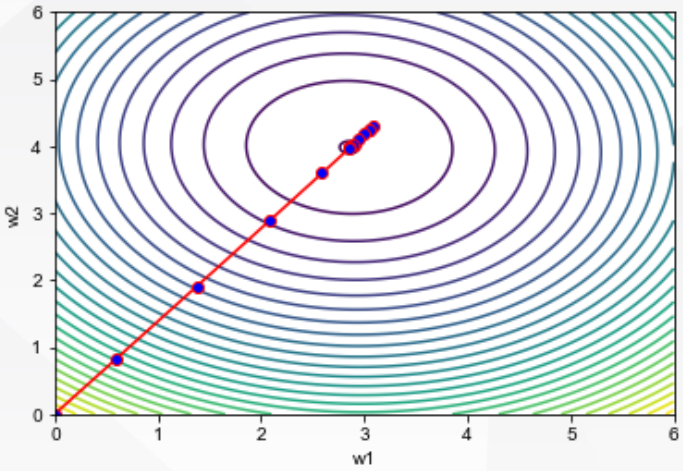
动量法：$\boldsymbol{v}^{(t)} = \rho \boldsymbol{v}^{(t-1)} - \eta \nabla J(\boldsymbol{\theta}) \big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$

NAG：$\boldsymbol{v}^{(t)} = \rho \boldsymbol{v}^{(t-1)} - \eta \nabla J \big|_{\boldsymbol{\theta}^{(t)}+\rho \boldsymbol{v}^{(t-1)}}$

动量法比较



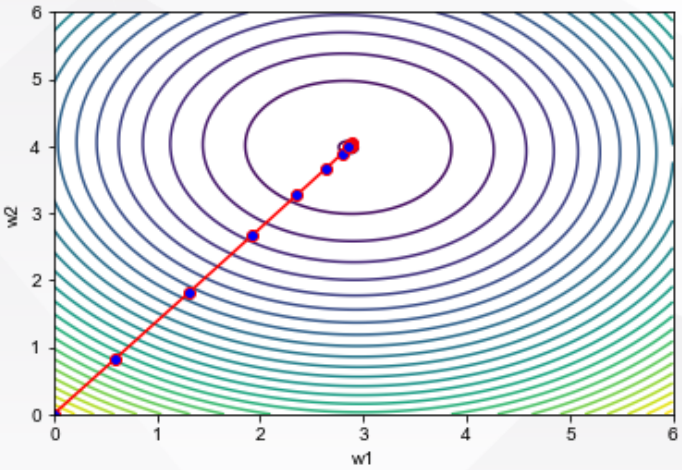
梯度下降



动量法

收敛快
迭代次数比梯度下降法少一半
尤其前几次迭代参数更新量大

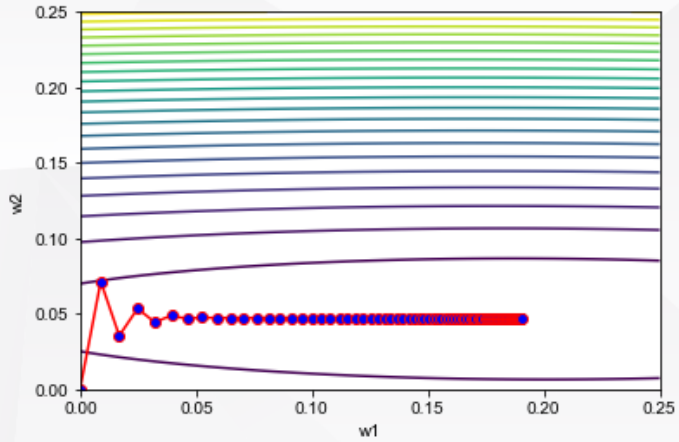
最后阶段搜索范围越过了最佳位置（学习率较大），这时两次更新方向相反，动量法会使得更新幅度减小，再慢慢回到最佳位置



NAG

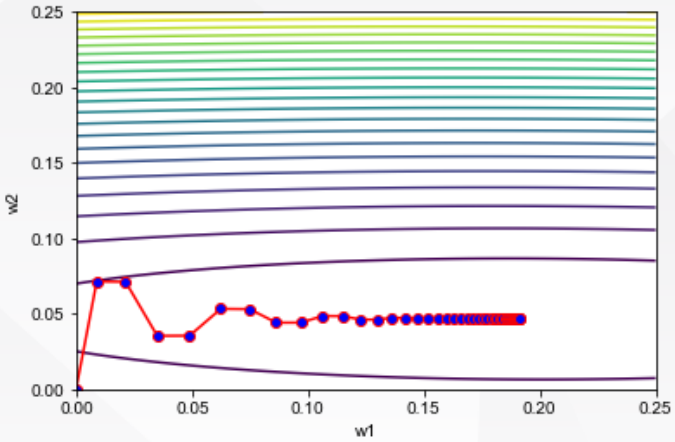
提前预知目标函数的信息，相当于多考虑了目标函数的二阶导数信息，类似牛顿法的思想，因此搜索路径更合理，收敛速度更快

动量法比较



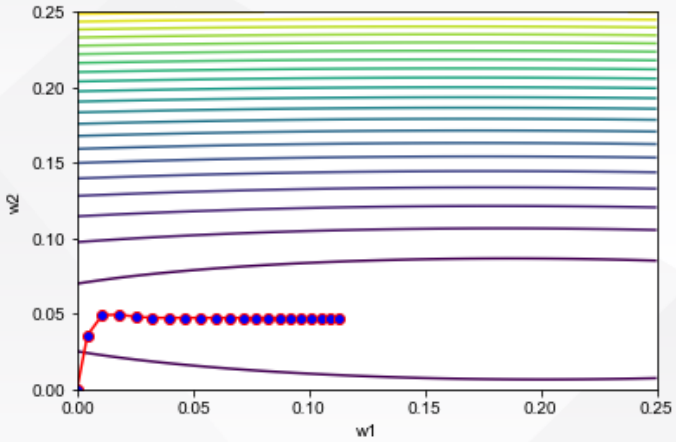
梯度下降

目标函数在竖直方向比在水平方向的斜率的绝对值更大，梯度下降法中参数在竖直方向比在水平方向移动幅度更大，在长轴上呈“之”字形反复跳跃，缓慢向最小值逼近。
（不同参数的梯度范围差异大，通常是因为特征没有去量纲）



动量法

竖直方向上的移动更加平滑，且在水平方向上更快逼近最优解，
因为此时竖直方向的当前梯度与之前的梯度方向相反相互抵消，移动的幅度小

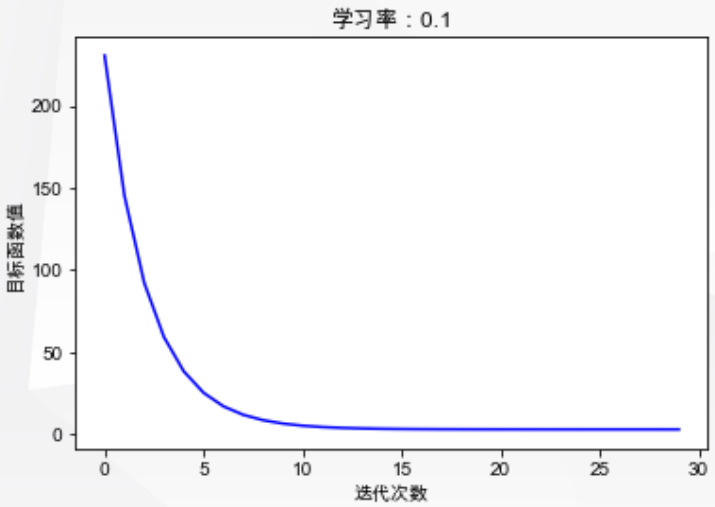


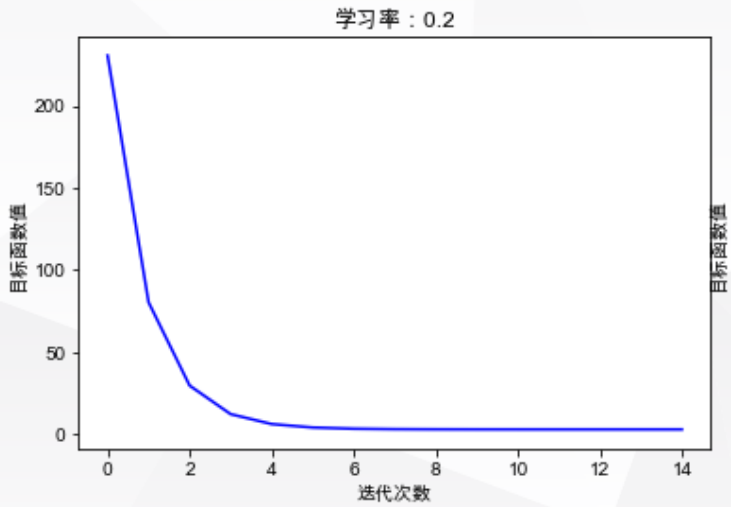
NAG

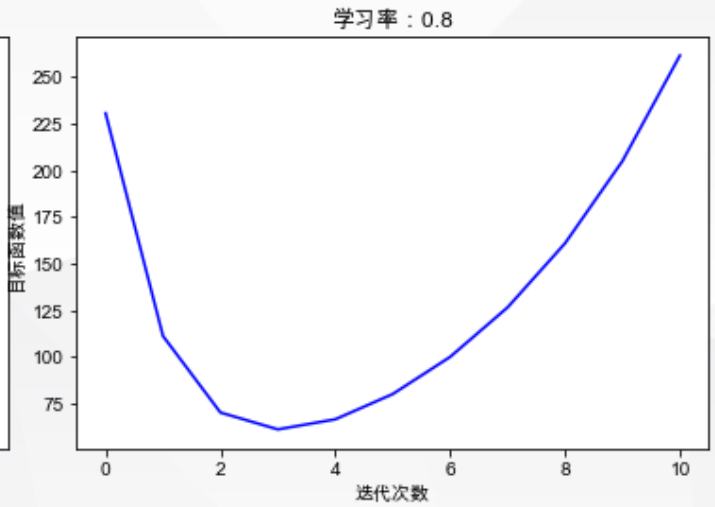
提前预知目标函数的信息，相当于多考虑了目标函数的二阶导数信息，类似牛顿法的思想，因此搜索路径更合理，收敛速度更快

学习率

- 学习率对训练的影响
  - 目标函数变化太慢：学习率太低
  - 目标函数出现NaN：通常意味着学习率太高
  - 建议：[1e-3 … 1e-5]







自适应学习率

- **AdaGrad**
  - 经常更新的参数学习率较小，尽量不被单个样本影响较大
  - 偶尔更新的参数学习率大一些，希望能从偶然出现的样本上多学一些
  - 使用二阶动量（迄今为止所有梯度值的平方和）来度量历史更新频率

$$\boldsymbol{s}^{(t)} = \boldsymbol{s}^{(t-1)} + \boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{s}^{(t)} + \varepsilon}} \odot \boldsymbol{g}^{(t)}$$

$\odot$：向量按元素乘

$\sqrt{()}$：对向量按元素求平方根

初始学速率η：一般设置为0.01

ϵ通常取很小的数：如10^{-6}

  - 存在问题：
    - 梯度会累加得越来越大，学习率衰减：学习速率衰减过快
    - 减缓陡峭区域的下降过程、加速平坦区域的过程

RMSProp

- 为了缓解Adagrad学习率衰减过快，RMSprop改变梯度累积为指数衰减的移动平均以丢弃遥远的历史。

AdaGrad：

$$\boldsymbol{s}^{(t)} = \boldsymbol{s}^{(t-1)} + \boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{s}^{(t)} + \varepsilon}} \odot \boldsymbol{g}^{(t)}$$

RMSprop ：

$$\boldsymbol{s}^{(t)} = \rho \boldsymbol{s}^{(t-1)} + (1-\rho)\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{s}^{(t)} + \varepsilon}} \odot \boldsymbol{g}^{(t)}$$

Adam

■ **Adam**：Adaptive + Momentum，同时利用一阶动量和二阶动量

```
## Adam
first_moment = 0
second_moment = 0

while True:
    dx = compute_gradient(x)
    first_moment = rho_1 * first_moment + (1 - rho_1)*dx
    second_moment = rho_2 * second_moment + (1 - rho_2)*dx*dx
    x -= learning_rate first_moment /np.sqrt(second_moment + 1e-7)
```

$$\boldsymbol{v}^{(t)} = \rho_1 \boldsymbol{v}^{(t-1)} - (1-\rho_1)\boldsymbol{g}^{(t)}$$

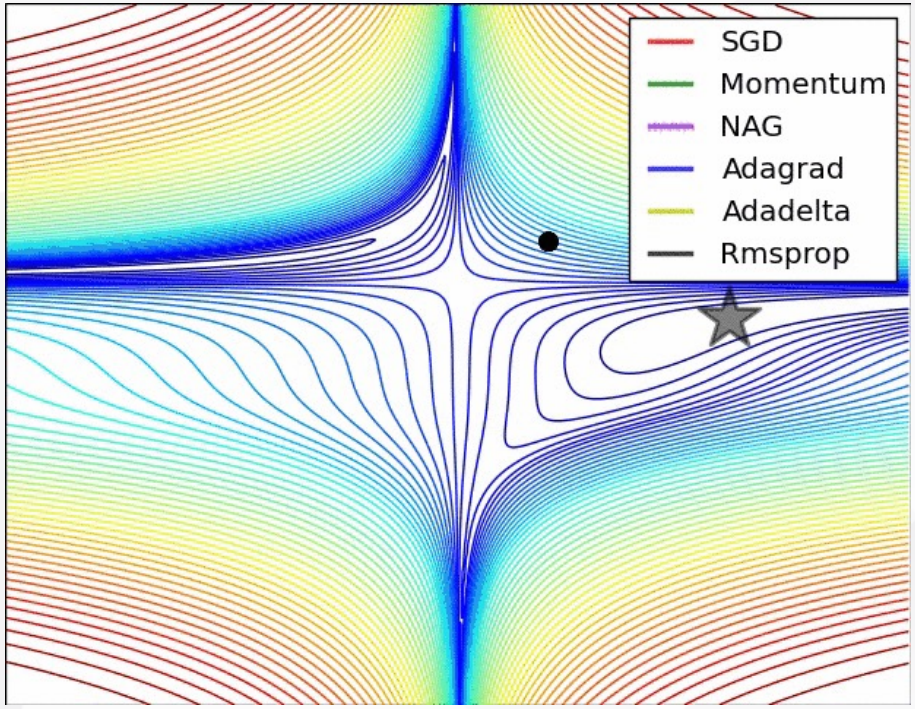
AdaGrad / RMSProp

$$\boldsymbol{s}^{(t)} = \rho_2 \boldsymbol{s}^{(t-1)} + (1-\rho_2)\boldsymbol{g}^{(i)} \odot \boldsymbol{g}^{(i)}$$

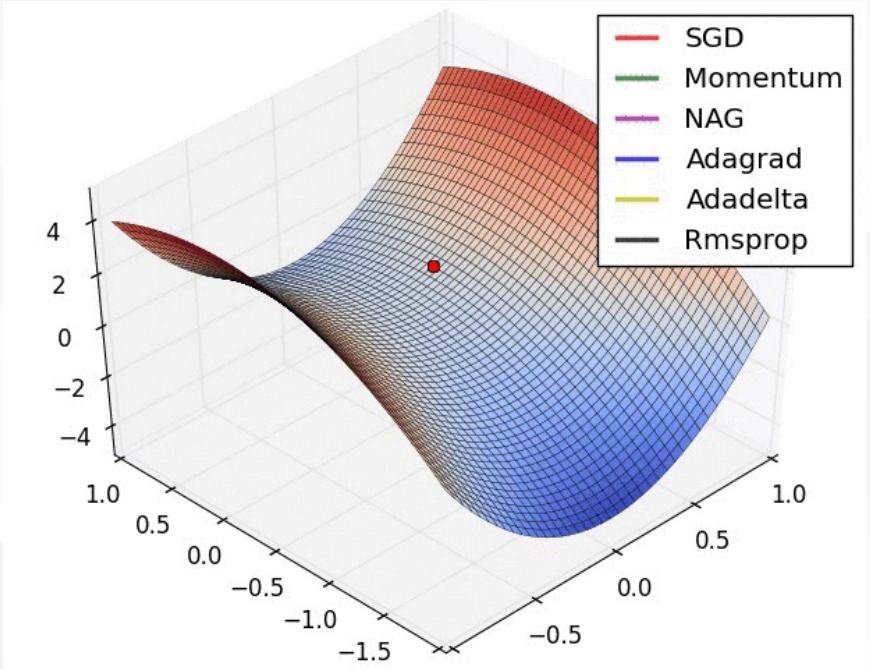
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \frac{\eta}{\sqrt{\boldsymbol{s}^{(t)}+\epsilon}} \odot \boldsymbol{v}^{(t)}$$

•超参数建议： $\rho_1 = 0.9$ ，$\rho_2 = 0.999$ ，$\eta = 1\text{e}{-3}$ 或$5\text{e}{-4}$

各种改进的梯度下降法的可视化结果



鞍点：一阶导数等于0但是海森矩阵的特征值有正有负



自适应学习率类的算法(AdaGrad, AdaDelta和RMSprop)的优化路径直接走向了右边的极小值点，
但是动量梯度法(绿色曲线)和NAG(紫色曲线)均是先走到了另一处狭长的区域，再转向极小值点，
且NAG的纠正速度快于动量梯度法。

SGD(红色曲线)在鞍点处停止
动量梯度法(绿色曲线)和NAG(紫色曲线)在鞍点处停留一会后逐渐脱离鞍点，
三种自适应学习率算法则能够快速摆脱鞍点。

优化方法

- **Adam** 通常是一个很好的选择。
- 学习率下降的**SGD+Momentum** 通常比Adam好一点点，但需要仔细调整
- 前期用Adam，享受Adam快速收敛的优势
- 后期切换到SGD，慢慢寻找最优解

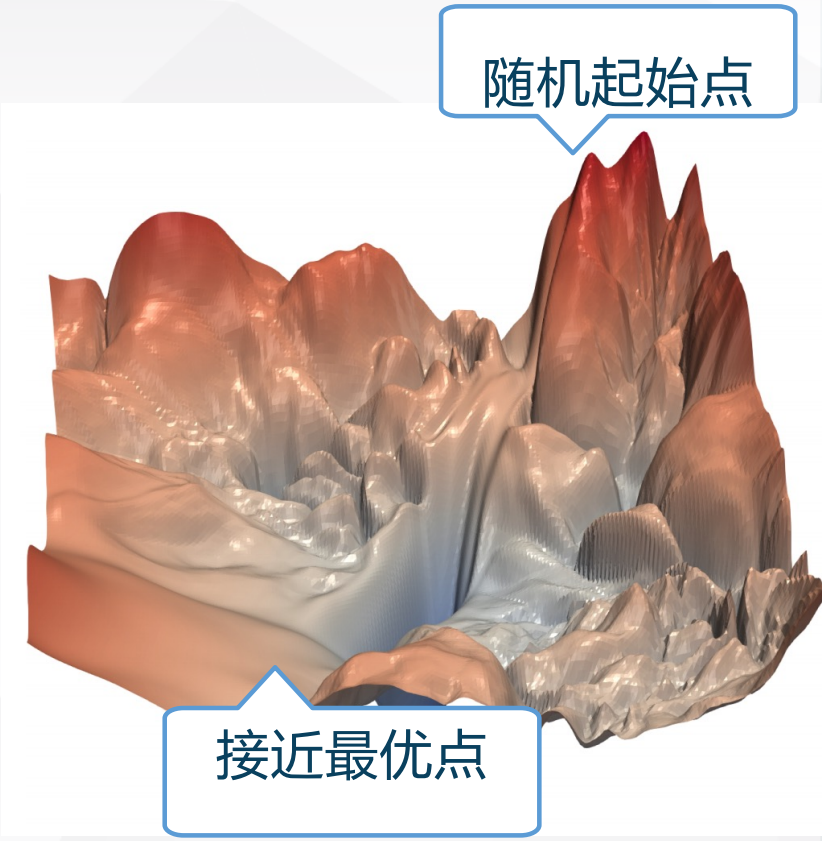
更多更新的新优化器可以参考：[An updated overview of recent gradient descent algorithms – John Chen – ML at Rice University (johnchenresearch.github.io)]

深度模型训练

- 梯度下降的改进
  - 小批量随机梯度下降
  - 自适应梯度方向（动量法）
  - 自适应学习率
  - 参数初始化
  - 批正规化
  - 跳跃连接
- 抗过拟合策略

权重初始化

- 使用适当范围内的随机值初始化权重
- 训练的开始容易受到数值不稳定性的影响
  - 远离最优点的表面可能很复杂
  - 接近最优点的表面可能更平坦



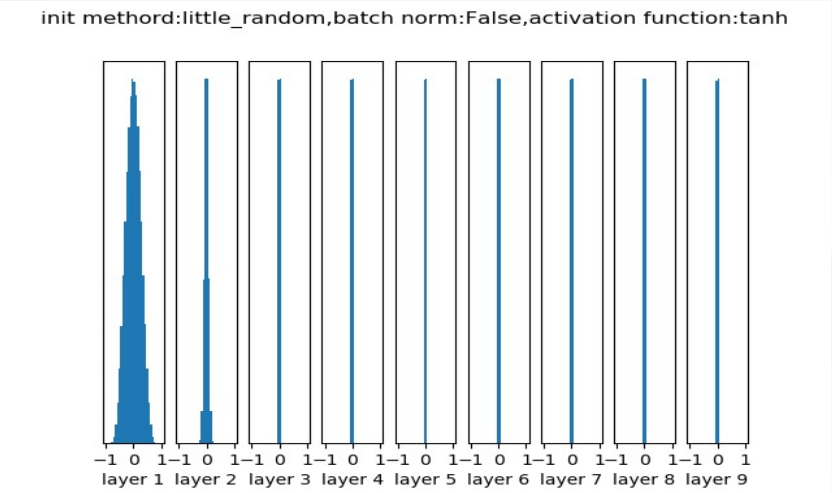
网络权重参数初始化

- 对简单的机器学习模型，如Logistic回归，简单的将模型参数初始化0或较小的随机数即可。
- 对深度模型，由于目标函数非凸，层次深，如何选择参数初始值便成为一个值得探讨的问题。
- 偏置参数b通常设置为0
- 模型权重的初始化对于网络的训练很重要：
  - 不好的初始化参数会导致梯度传播问题，降低训练速度；
  - 好的初始化参数能够加速收敛，并且更可能找到较优解。

随机数初始化

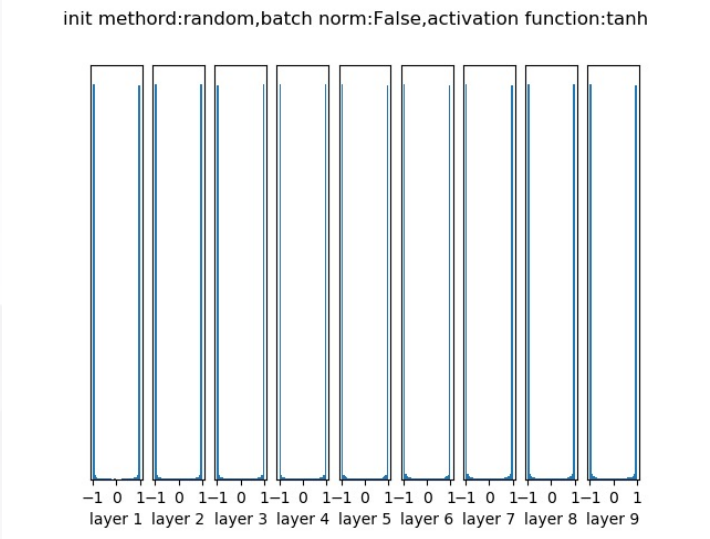
■例：10层DNN

每层权重都用$N(0,0.01)$随机初始化，
经过10层网络后各层的输出分布



各层的数据分布不一致
随着层数的增加，
几乎所有的输出值都很接近0

每层权重都用$N(0,1)$随机初始化，
经过10层网络后各层的输出分布



权重更有可能为较大的值
各层输出的几乎全部集中在-1或1附近，
神经元呈饱和状态，梯度为0，
即出现“死亡神经元”，参数难以被更新

每层输出和梯度的方差

- 略去非线性变换部分，全连接每层输出$\boldsymbol{o}$和输入$\boldsymbol{x}$之间的关系为

$$o_i = \sum_{j=1}^{N_{in}} w_{i,j}\, x_j$$

N_{in}: 神经元的输入维度

- 将每层的输出和梯度看成随机变量
- 假设所有$w_{i,j}$独立同分布，x_j独立同分布，且$w_{i,j}$和x_j独立

$$\mathbb{E}[w_{i,j}] = 0,\ \mathrm{Var}[w_{i,j}] = \sigma^2, \qquad \mathbb{E}[x_j] = 0,\ \mathrm{Var}[x_j] = \gamma^2$$

- 则输出的期望和方差为：

$$\mathbb{E}[o_i] = \mathbb{E}[\textstyle\sum_{j=1}^{N_{\mathrm{in}}} w_{i,j}\, x_j]$$

$$= \sum_{j=1}^{N_{\mathrm{in}}} \mathbb{E}[w_{i,j} x_j] = \sum_{j=1}^{N_{\mathrm{in}}} \mathbb{E}[w_{i,j}]\mathbb{E}[x_j] = 0$$

每层输出和梯度的方差

■输出的方差为：

$$\begin{aligned}\mathrm{Var}[o_i] &= \mathbb{E}\left[o_i^2\right] - (\mathbb{E}[o_i])^2 \\ &= \mathbb{E}\left[w_{i,j}^2 x_j^2\right] - 0 \\ &= \sum_{j=1}^{N_{\mathrm{in}}} \mathbb{E}\left[w_{i,j}^2\right] + \sum_{j=1}^{N_{\mathrm{in}}} \mathbb{E}\left[x_j^2\right] \\ &= N_{\mathrm{in}}\sigma^2\gamma^2\end{aligned}$$

■若要使该层输出的方差（$N_{\mathrm{in}}\sigma^2\gamma^2$）与输入的方差（$\gamma^2$）想等，则

$$N_{\mathrm{in}}\sigma^2 = 1.$$

每层输出和梯度的方差

- 类似的，反向传播时，经过一层后梯度的方差不变，得到为：

$$N_{\text{out}}\sigma^2 = 1.$$

- 因此，要使该层输出和梯度的方差都不变，

$$\left.\begin{aligned} N_{\text{in}}\sigma^2 &= 1. \\ N_{\text{out}}\sigma^2 &= 1. \end{aligned}\right\} \Longrightarrow \frac{1}{2}(N_{\text{in}}\sigma^2 + N_{\text{out}}\sigma^2) = 1$$

$$\sigma^2 = \frac{2}{N_{\text{in}} + N_{\text{out}}}$$

Xavier初始化

泽-维尔

- 保持各层的激活值的方差和梯度的方差在传播过程中保持一致
- 假设激活函数为线性函数，推导得出：

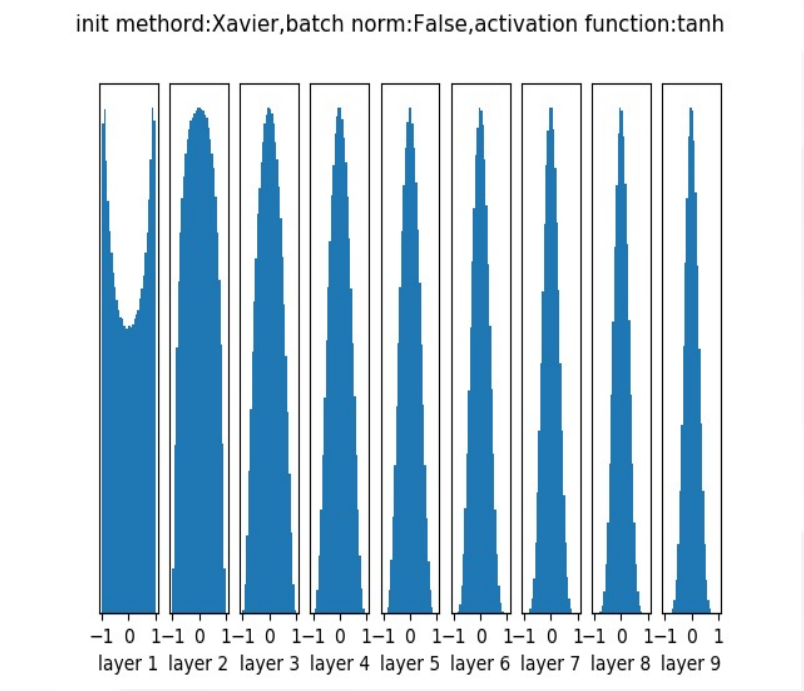
$$\mathrm{Var}[W] = \frac{2}{N_{in} + N_{out}}$$

N_{in}：该层输入的数目

N_{out}：该层输出的数目

激活函数取sigmoid或tanh，且$\boldsymbol{x}, \boldsymbol{W}$很小时，他们的乘积之和也较小，sigmoid和tanh在0附件的表现与线性相似，梯度接近 1

Xavier 初始化ReLU 激活函数无能能力，因为当输入小于0时，ReLU激活函数输出为0，不满足线性假设。

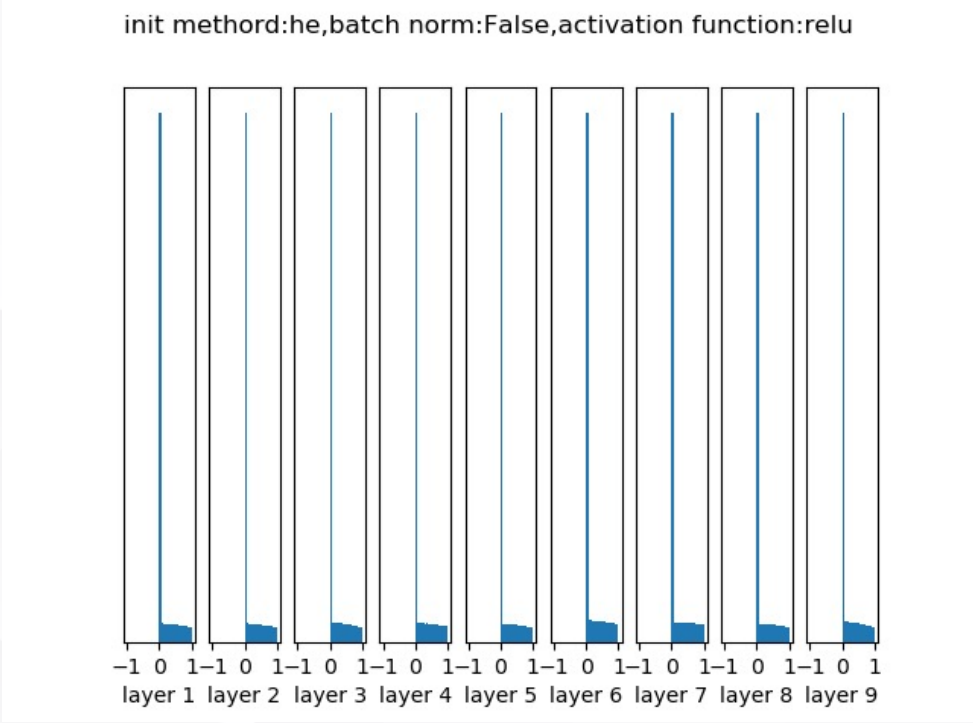


对激活函数sigmoid或tanh ，后续层的输出依然保持较好的高斯分布

He初始化

- 在ReLU网络中，假定每一层有一半的神经元被激活（输入大于0），另一半为0，所以，要保持方差不变，只需要在Xavier初始化的基础上再除以2:

$$\mathrm{Var}[W] = \frac{4}{N_{in} + N_{out}}$$



深度模型训练

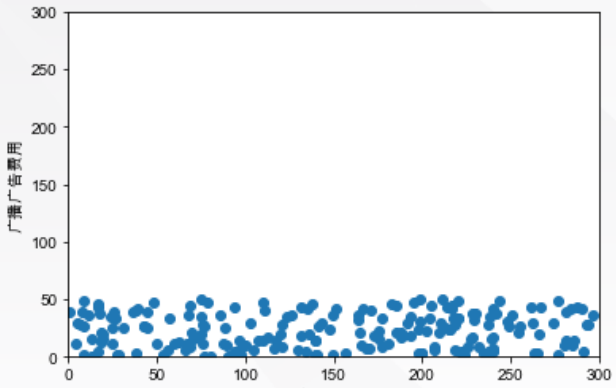
- 梯度下降的改进
  - 小批量随机梯度下降
  - 自适应梯度方向（动量法）
  - 自适应学习率
  - 参数初始化
  - 批正规化
  - 跳跃连接
- 抗过拟合策略

输入输出方差相等：将输出值强行做一次高斯归一化和线性变换

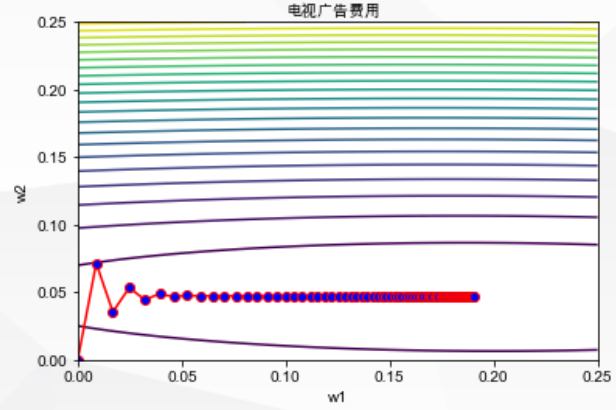
Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.

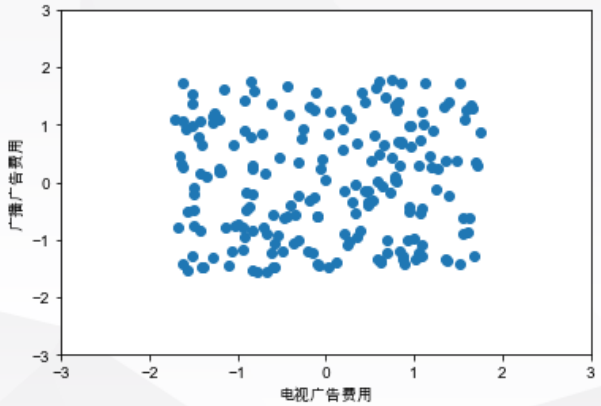
回忆：特征标准化

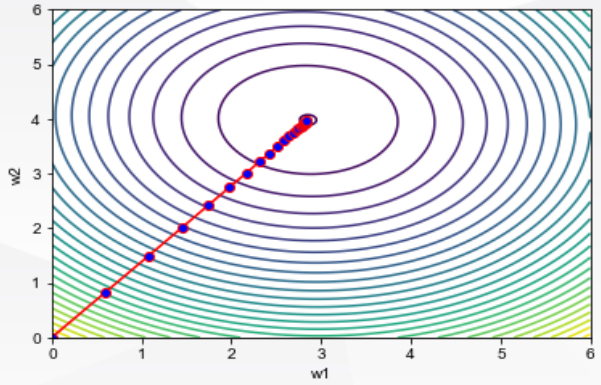
- 输入数据做标准化处理：每个特征的均值为0、标准差为1
- 标准化：各个特征的分布相近→更容易训练出有效的模型



$$x_i' = \frac{x_i - \mu}{\sigma}$$

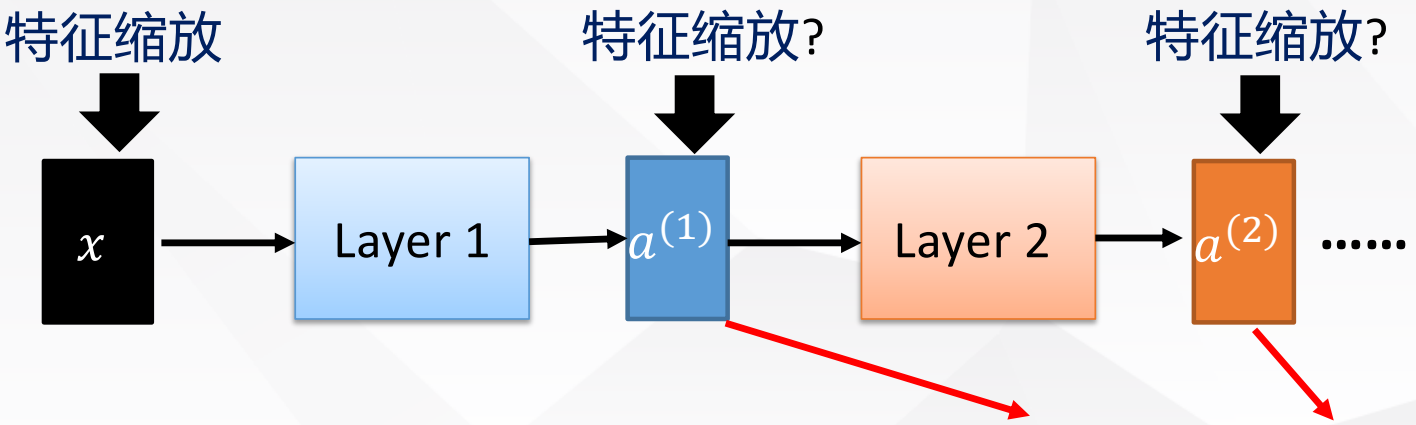






隐含层?

- 在深层神经网络中，即使输入数据已做标准化，训练中模型参数的更新依然很容易造成输出的剧烈变化 → 难以训练出有效的模型



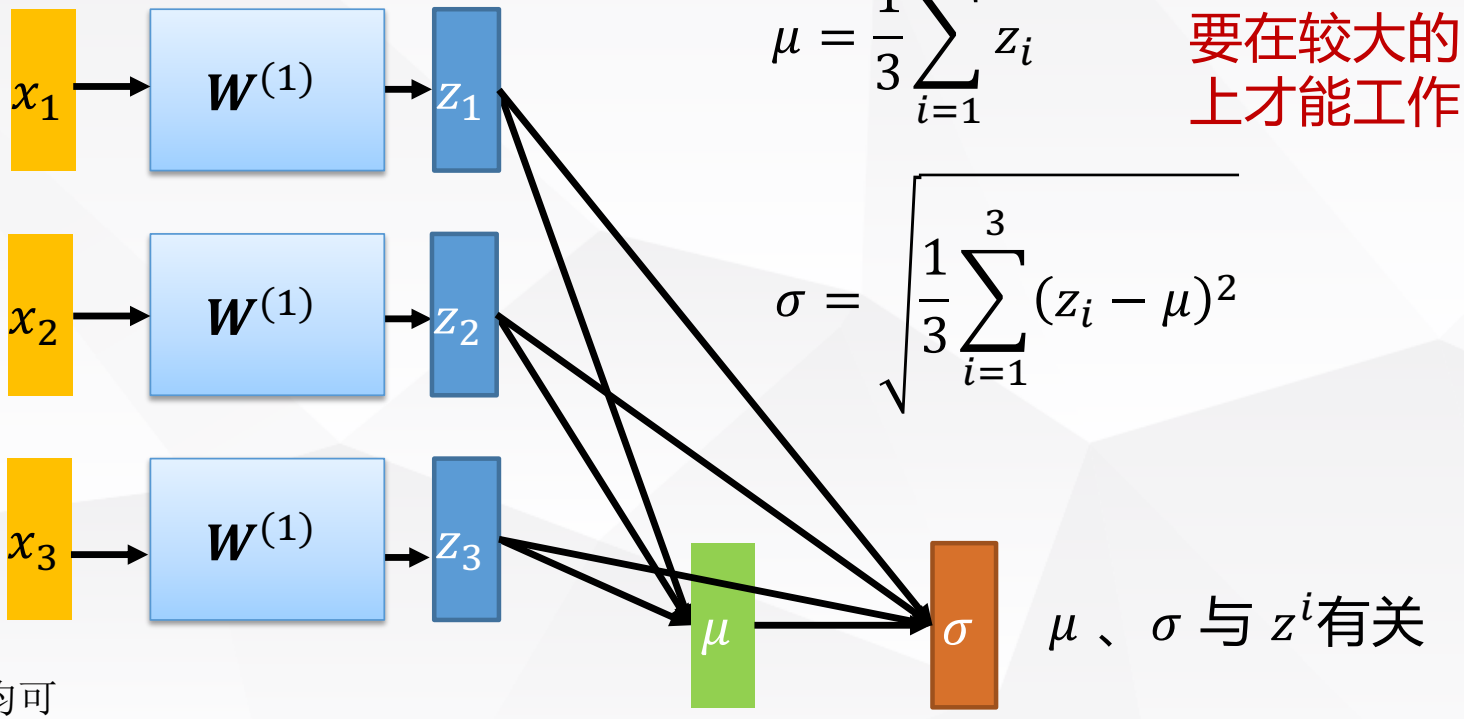
困难：统计量在训练中会变化。

Batch normalization

小的学习率也有用，但训练会更慢。

Batch normalization

- 在小批量上进行标准化（这里称为归一化normalization ），不断调整神经网络中间输出，从而使整个神经网络在各层的中间输出的数值更稳定。



$$\mu = \frac{1}{3}\sum_{i=1}^{3} z_i$$

$$\sigma = \sqrt{\frac{1}{3}\sum_{i=1}^{3}(z_i - \mu)^2}$$

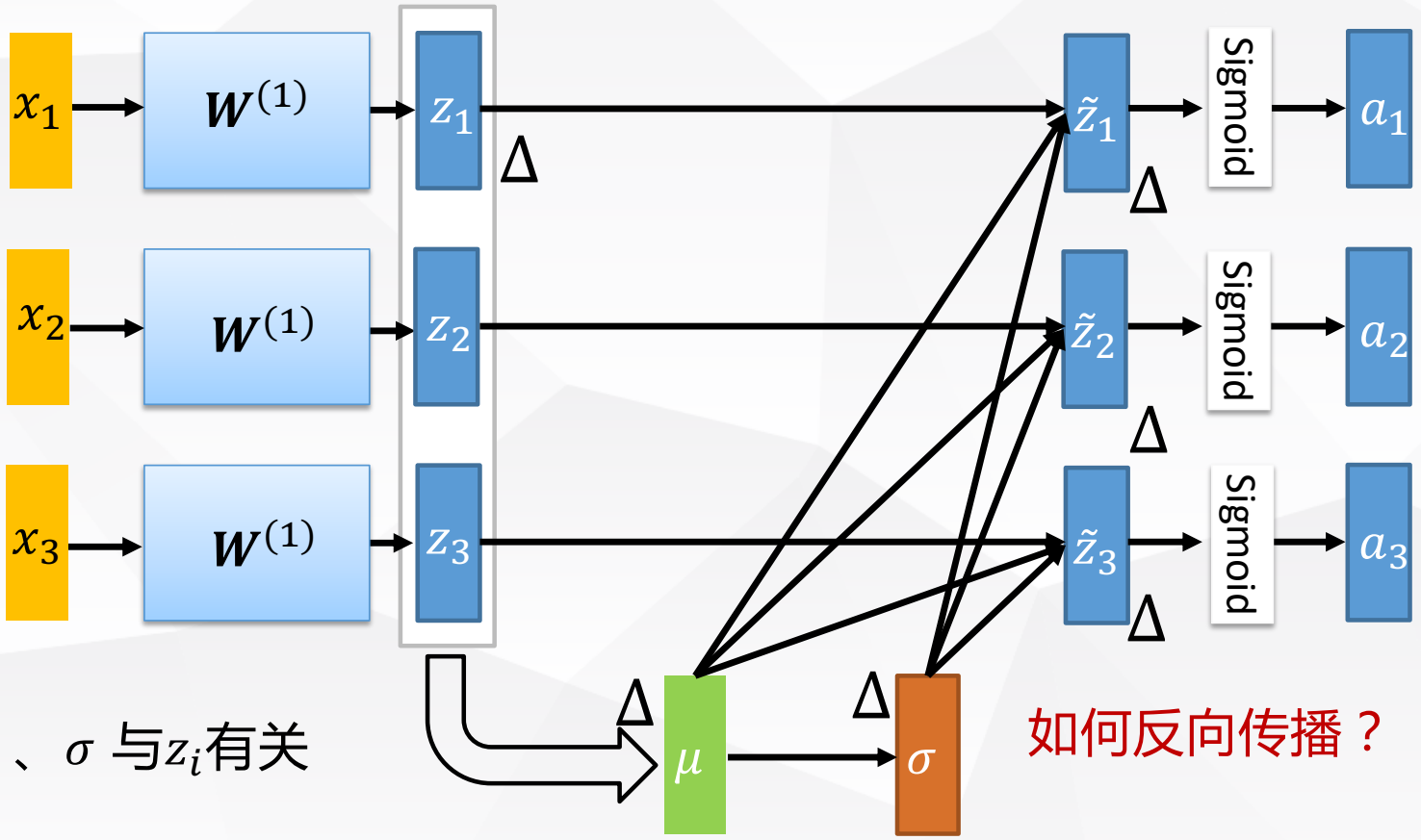
Batch normalization需要在较大的batch size上才能工作。

μ 、σ 与 z^i有关

对z、a做BN均可

Batch normalization

$$\tilde{z}_i = \frac{z_i - \mu}{\sigma}$$

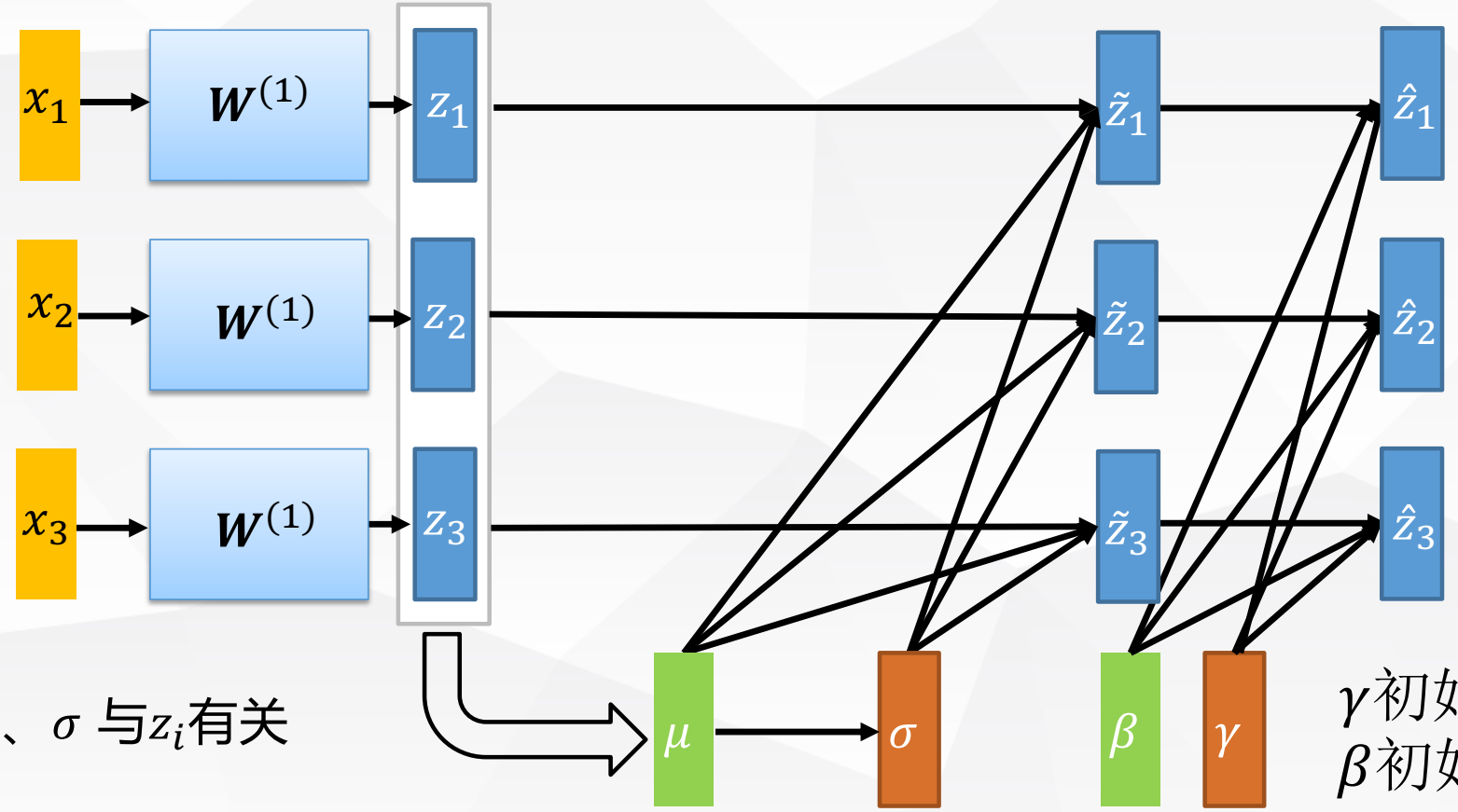


μ 、σ 与z_i有关

如何反向传播？

Batch normalization

$$\tilde{z}_i = \frac{z_i - \mu}{\sigma} \qquad \hat{z}_i = \gamma \odot \tilde{z}_i + \beta$$



μ 、σ 与z_i有关

γ初始化为1
β初始化为0

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

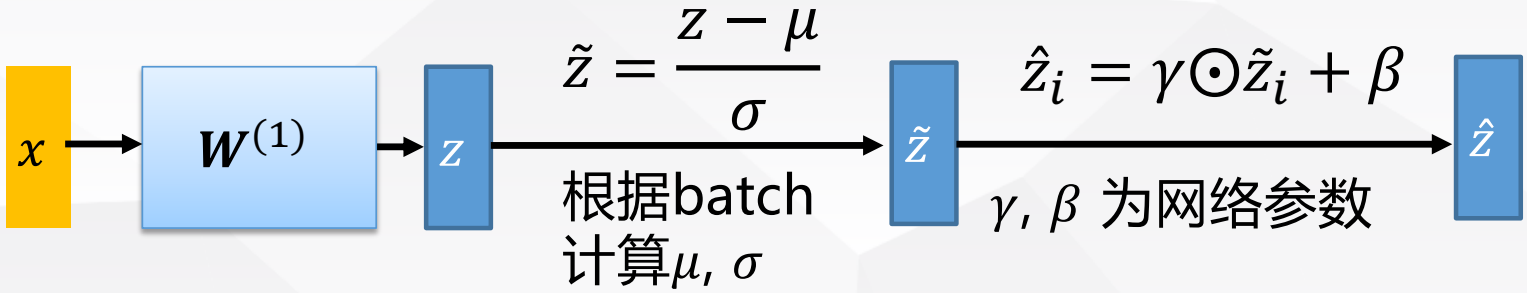
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

```
# BN model bn_model =
nn.Sequential(
nn.Linear(input_size,100), #1
myBatchNorm2d(100),
nn.Sigmoid(),

nn.Linear(100,100), #2
myBatchNorm2d(100),
nn.Sigmoid(),

nn.Linear(100,100), #3
myBatchNorm2d(100),
nn.Sigmoid(),

nn.Linear(100,10) # out )
```

Batch normalization

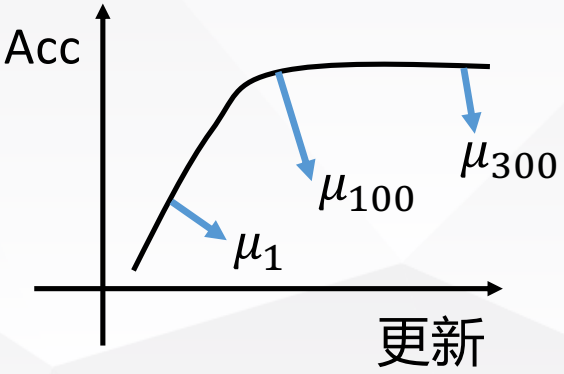
■测试



■测试时没有batch

■理想的解决方案：在整个训练集上计算μ, σ

■现实的方案：在训练时计算每个batch的μ, σ移动平均



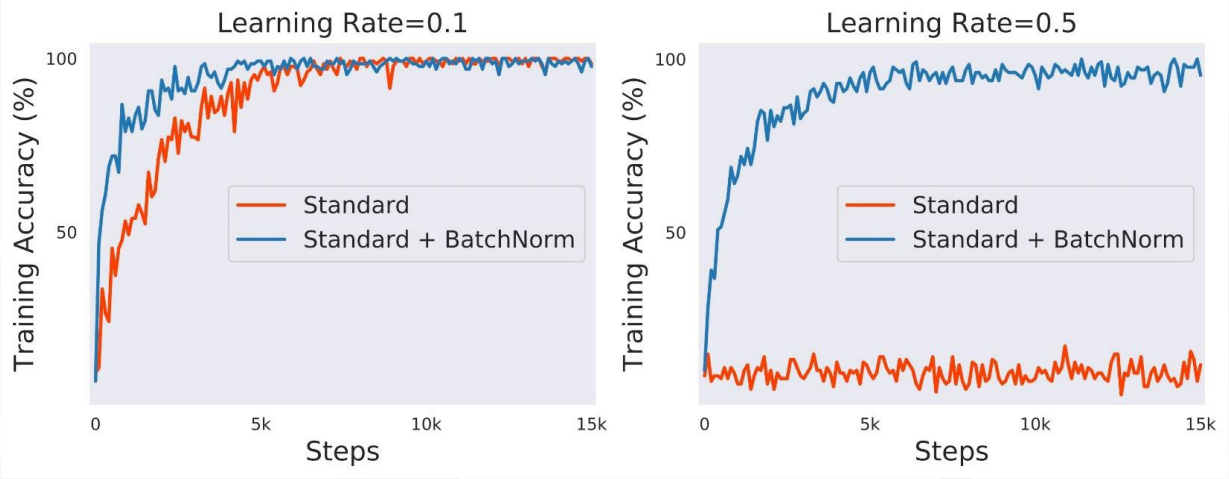
$$\boldsymbol{\mu^1} \quad \boldsymbol{\mu^2} \quad \boldsymbol{\mu^3} \quad \cdots\cdots \quad \boldsymbol{\mu^t}$$

$$\bar{\boldsymbol{\mu}} \leftarrow p\bar{\boldsymbol{\mu}} + (1-p)\boldsymbol{\mu^t}$$

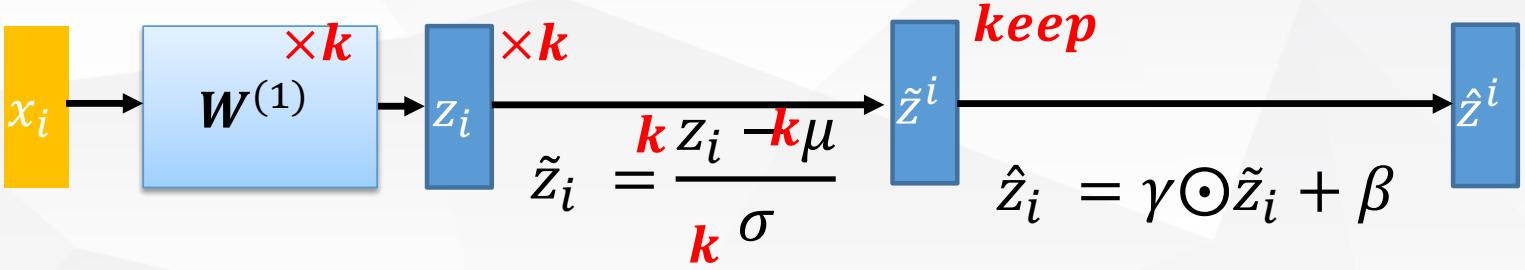
Batch normalization的好处

https://arxiv.org/abs/150203167

- 可以采用更大的学习率、收敛速度更快



- 受初始化的影响变弱
- 减少对正则化的依赖



预训练

- 预训练（pre-training）是一种非常有效的神经网络的初始化方法。
- 非监督：贪心地逐层训练自编码器得到初始权重，然后再做细调（不再常用）
- 类似问题中已经训练好的模型
  - 若数据集与预训练模型采用的训练数据集非常相似，且新数据集较小，只要在预训练模型最顶层输出特征上再训练一个线性分类器即可
  - 若新数据集较大，可以使用一个较小的学习速率对预训练模型的最后几个顶层进行调优。
  - 如果新数据集与预训练模型采用的训练数据集相差较大但拥有足够多的数据，则需要对网络的多个层进行调优，同样也要使用较小的学习速率。
  - 最坏的情况是新数据集不仅较小，而且与原始数据集相差较大，此时基于较为靠前的特征层就使用SVM分类器可能是相对较好的方案 。

模型初始化建议

- 使用 ReLU（无BN） 激活函数时，最好选用 He 初始化方法，将参数初始化为服从高斯分布或者均匀分布的较小随机数。
- 使用 BN 时，减少了网络对参数初始值尺度的依赖，此时使用较小的标准差（如0.01）的高斯分布进行初始化即可。
- 借助预训练模型中参数作为新任务参数初始化的方式也是一种简便易行且十分有效的模型参数初始化方法。

深度模型训练

- 梯度下降的改进
  - 小批量随机梯度下降
  - 自适应梯度方向（动量法）
  - 自适应学习率
  - 参数初始化
  - 批正规化
  - 跳跃连接
- 抗过拟合策略

模型的深度

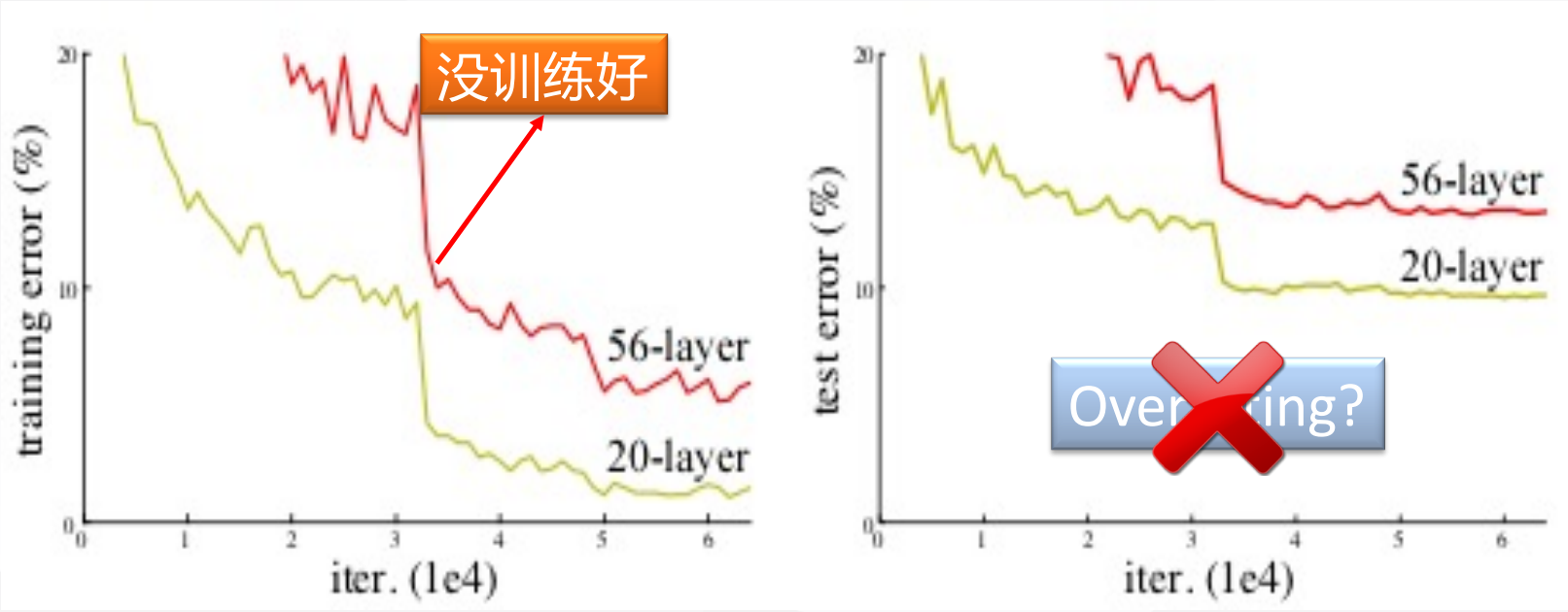
图像识别模型	面世时间	网络层次	备注
LeNet	1998	5	经典的手写数字识别模型
AlexNet	2012	8	ILSVRC 2012比赛第1名
VGG	2014	19	ILSVRC 2014比赛第2名
GoogLeNet	2014	22	ILSVRC 2014比赛第1名

注：ILSVRC是全球最权威的计算机视觉竞赛

能更深吗？

模型的深度

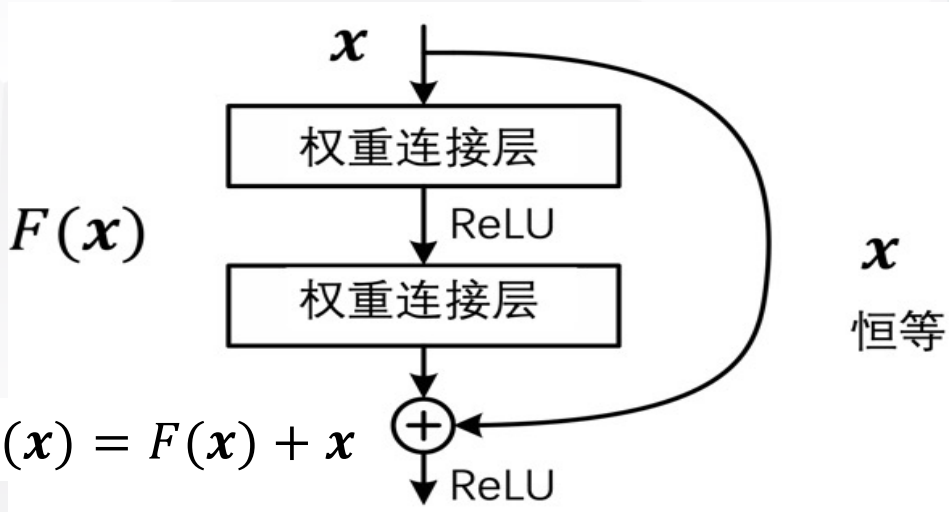
- 退化：深度CNN网络达到一定深度后再一味地增加层数并不能带来进一步的分类性能提高



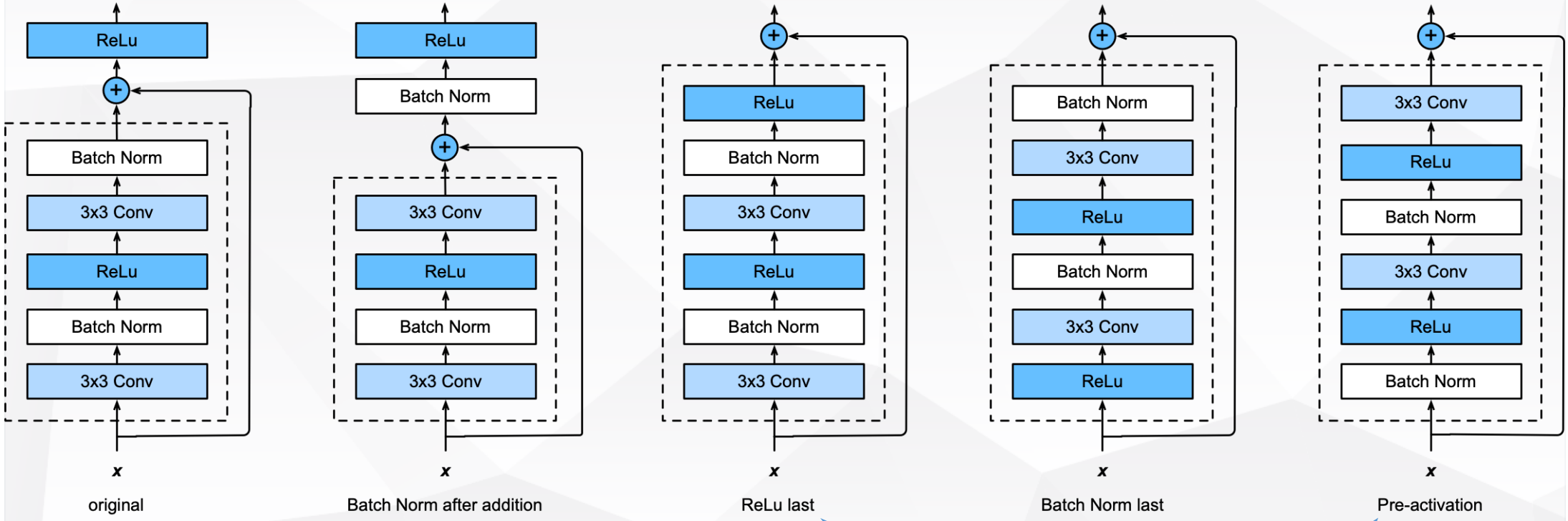
主要原因：梯度在传播过程中会逐渐消失

残差块

- 深度可达1000+层
- 短路连接（shortcut connections）
  - 优化残差映射比直接优化原始映射更容易
  - 处理梯度消失问题
  - 集成学习

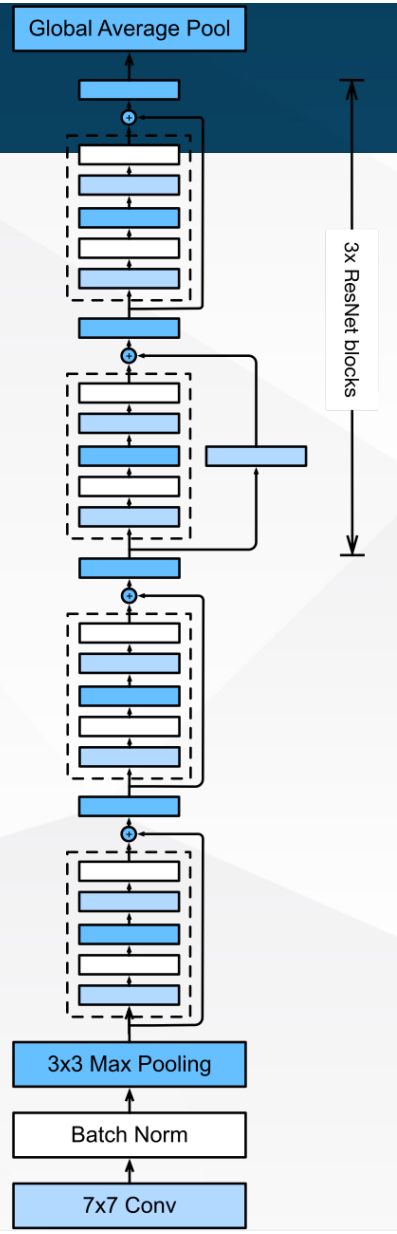


不同的残差块


ReLu
Batch Norm
3x3 Conv
ReLu
Batch Norm
3x3 Conv
x
original
ReLu
Batch Norm
3x3 Conv
ReLu
Batch Norm
3x3 Conv
x
Batch Norm after addition
ReLu
Batch Norm
3x3 Conv
ReLu
Batch Norm
3x3 Conv
x
ReLu last
Batch Norm
3x3 Conv
ReLu
Batch Norm
3x3 Conv
ReLu
x
Batch Norm last
3x3 Conv
ReLu
Batch Norm
3x3 Conv
ReLu
Batch Norm
x
Pre-activation
尝试每一个排列

残差网络（ResNet）

- 相同块结构，例如与VGG 或 GoogleNet使用块结构
- 残差块连接可增加表示能力
- 池化 / 步幅 - 减少维度
- 批量归一化



稠密连接网络（DenseNet）

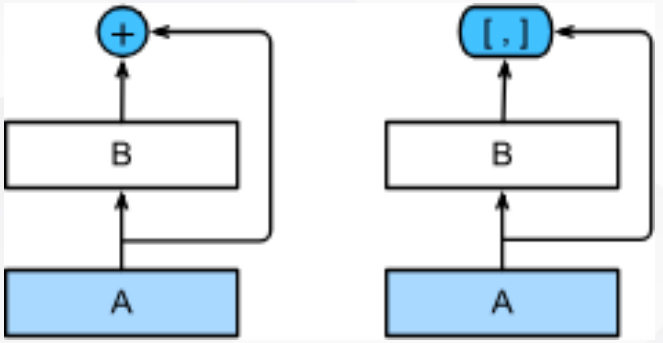
- DenseNet (Huang et al., 2016)
- ResNet结合 x 和 f(x)
- DenseNet 使用更高阶'泰勒系列'扩展

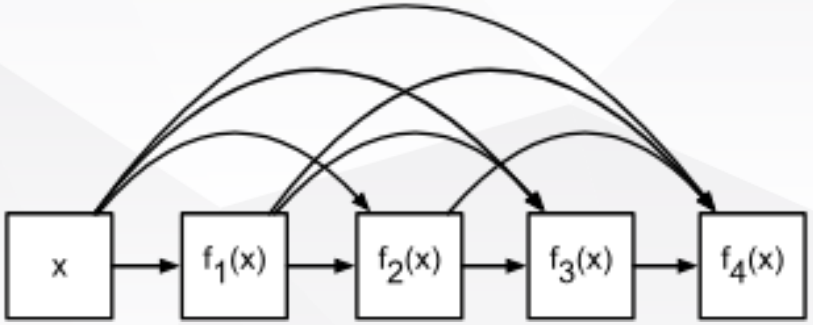
$$x_{i+1} = [x_i, f_i(x_i)]$$

$$x_1 = x$$

$$x_2 = [x, f_1(x)]$$

$$x_2 = [x, f_1(x), f_2([x, f_1(x)])]$$



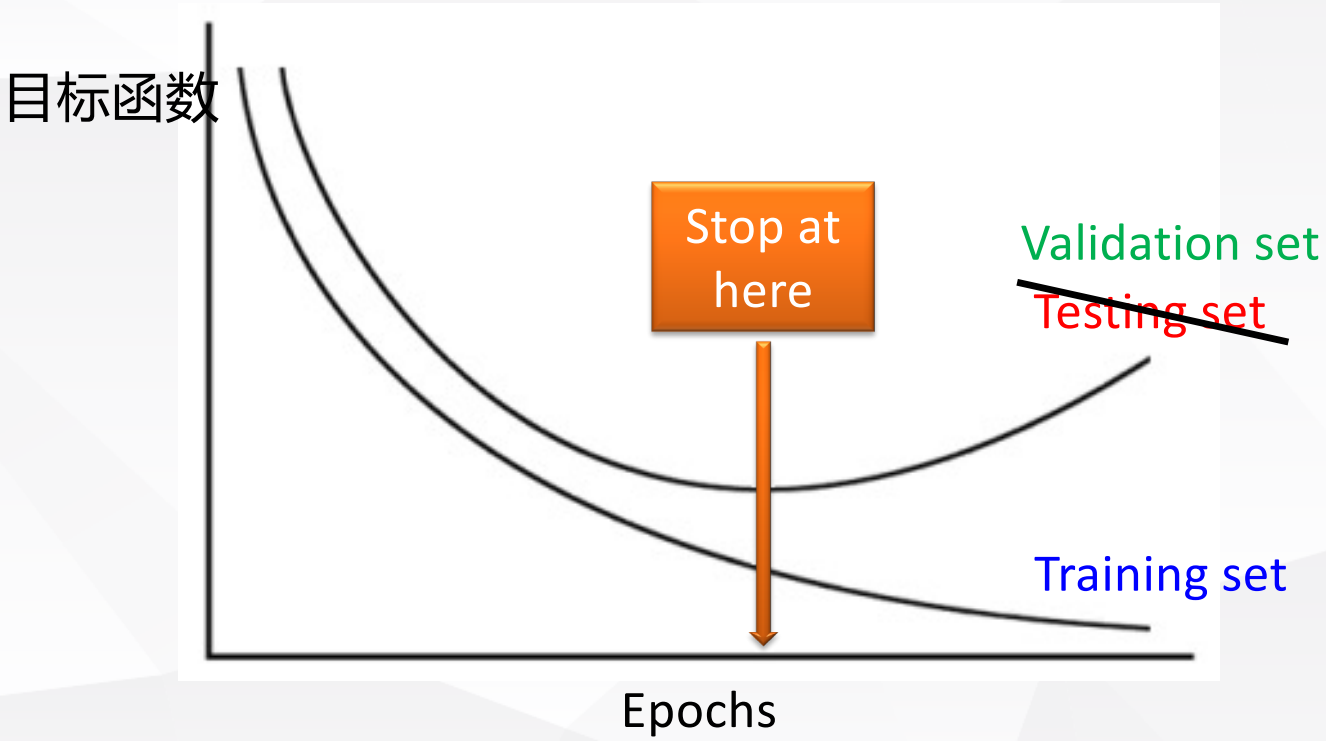


- 偶尔需要降低分辨率（过渡层）

深度模型训练

- 梯度下降的改进
  - 小批量随机梯度下降
  - 自适应学习率
  - 自适应梯度方向
  - 参数初始化
  - 批正则化
  - 跳跃连接
- 抗过拟合策略
  - 及早停止（Early Stopping）
  - 正则
  - 丢弃法（Dropout）
  - 参数初始化

Early Stopping



正则（Regularization）

■机器学习模型的目标函数通常包含两项：

- 模型不仅在训练集上的损失和要小，而且参数要尽可能接近0

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{N} L(y_i, \hat{y}_i; \boldsymbol{\theta}) + \lambda R(\boldsymbol{\theta})$$

损失函数
(如交叉熵损失，L2损失 ...)

正则项：

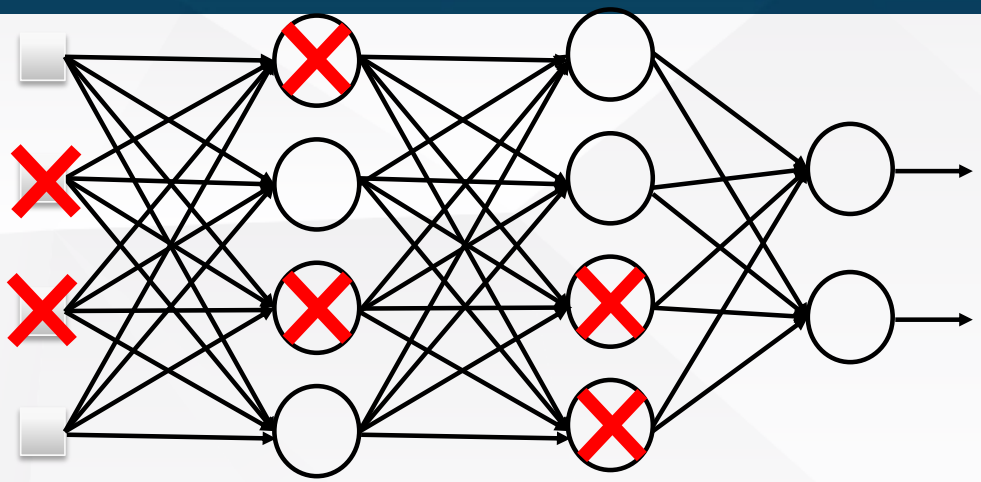
$$\boldsymbol{\theta} = [w_1, w_2, \dots,]$$

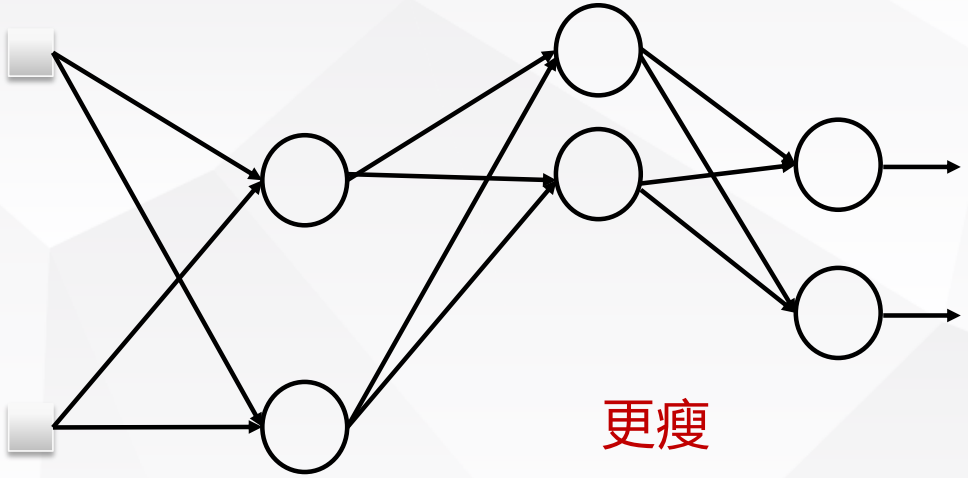
L2 正则：$\|\boldsymbol{\theta}\|_2^2 = w_1^2 + w_2^2 + \cdots$ Weight Decay

L1 正则：$\|\boldsymbol{\theta}\|_1 = |w_1| + |w_2| + \cdots$

(正则项不考虑偏置)

Dropout





在每个Mini- Batch训练时

- 随机删除一部分神经元，即随机地将输入中的每个元素以概率p置0

$$r_j^{(l)} = Bernoulli(1-p)$$

使得$\mathbb{E}[a] = \mathbb{E}[\text{dropout}(a)]$ ：对下一层网络而言，输入的幅值“看起来”一样

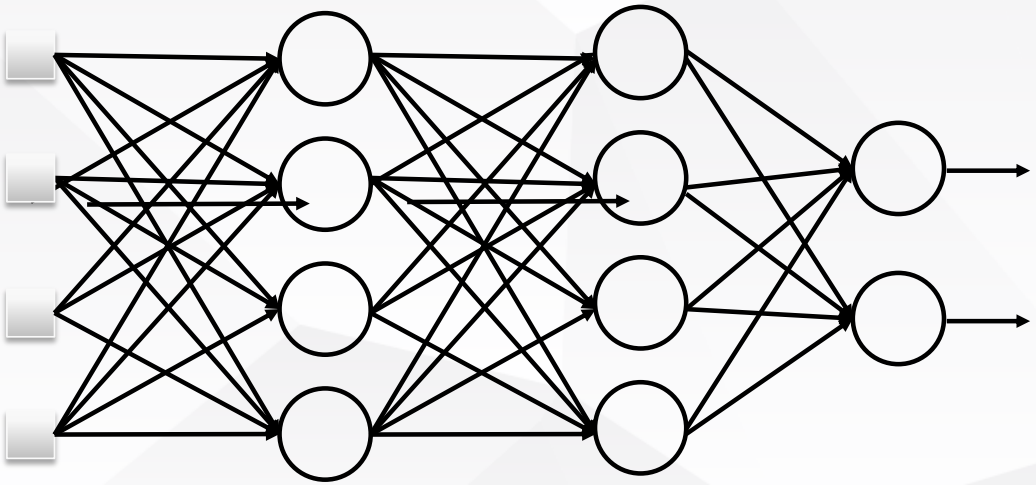
- 前向传播

$$\tilde{\boldsymbol{a}}^{(l)} = \boldsymbol{r}^{(l)} \odot \boldsymbol{a}^{(l)}, z_i^{(l+1)} = \boldsymbol{w}_i^{(l)} \tilde{\boldsymbol{a}}^{(l)} + b_i^{(l+1)}, a_i^{(l+1)} = \left(\frac{1}{1-p}\right) \sigma\left(z_i^{(l+1)}\right)$$

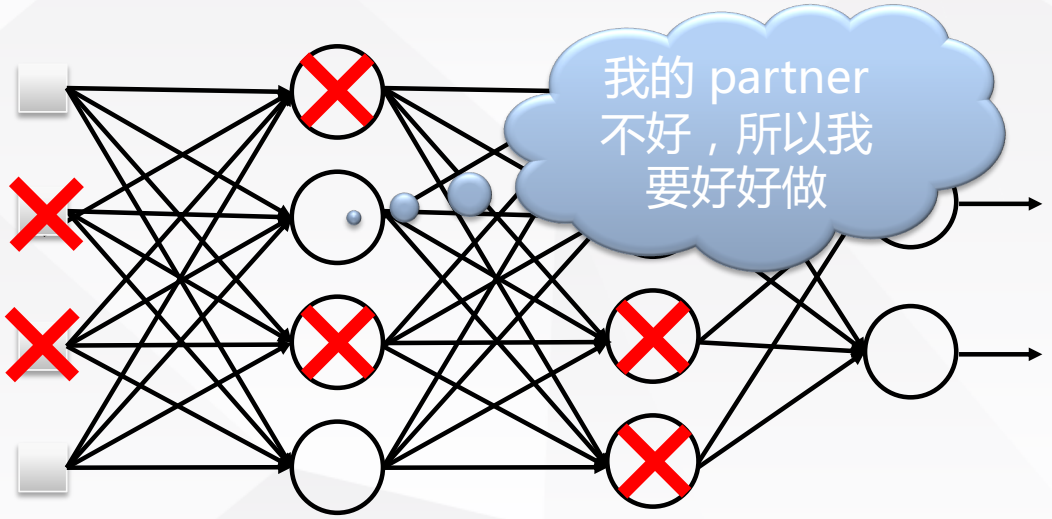
- 反向传播损失，更新参数（没有被删除的那一部分神经元的参数得到更新，删除的神经元参数保持被删除前的结果）

Dropout

- 测试时没有 Dropout

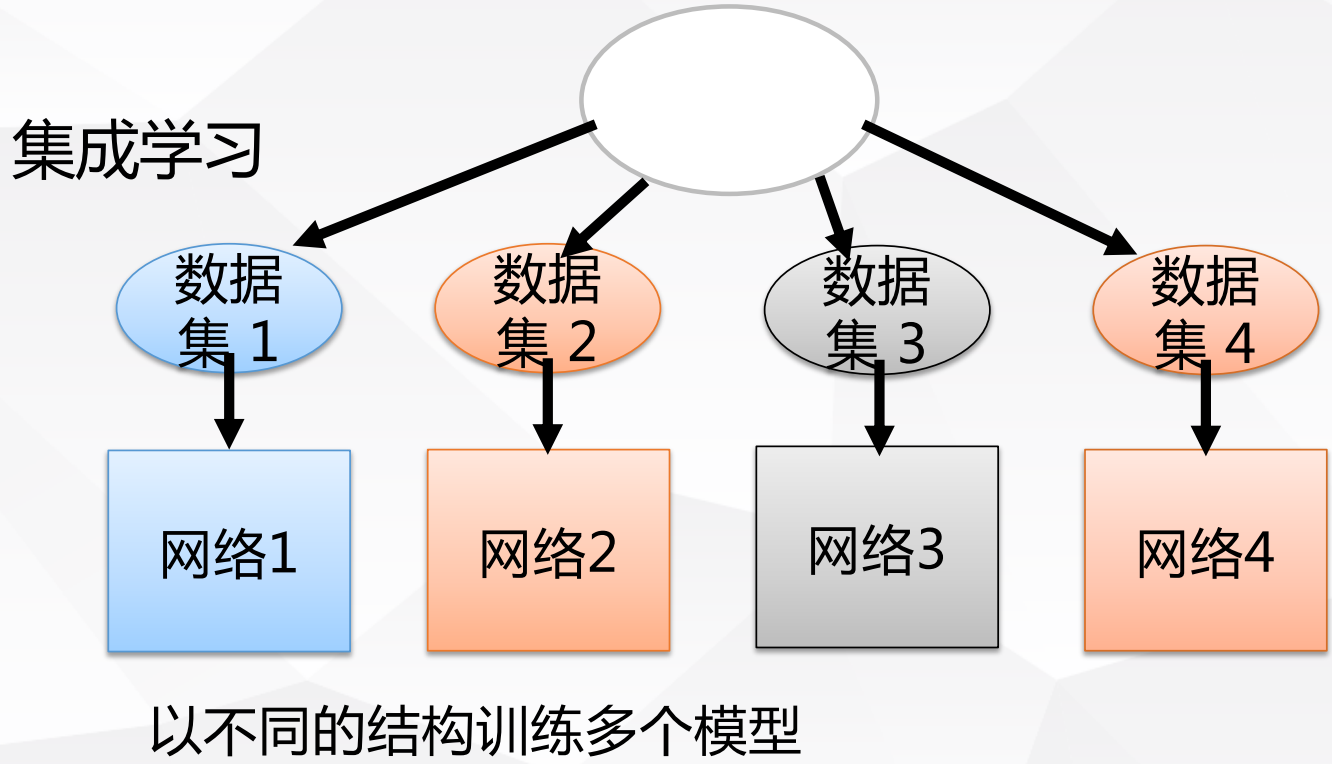


Dropout – 直觉解释



- 如果每个人都期望组里其他人好好干活，最后什么也干不好（三个和尚没水喝）。
- 但是，如果你知道你的小伙伴可能会dropout，你就会干得更好，这样才能完成任务。
- 测试时，大家都好好干，没有人dropout，因此会得到很好的结果。

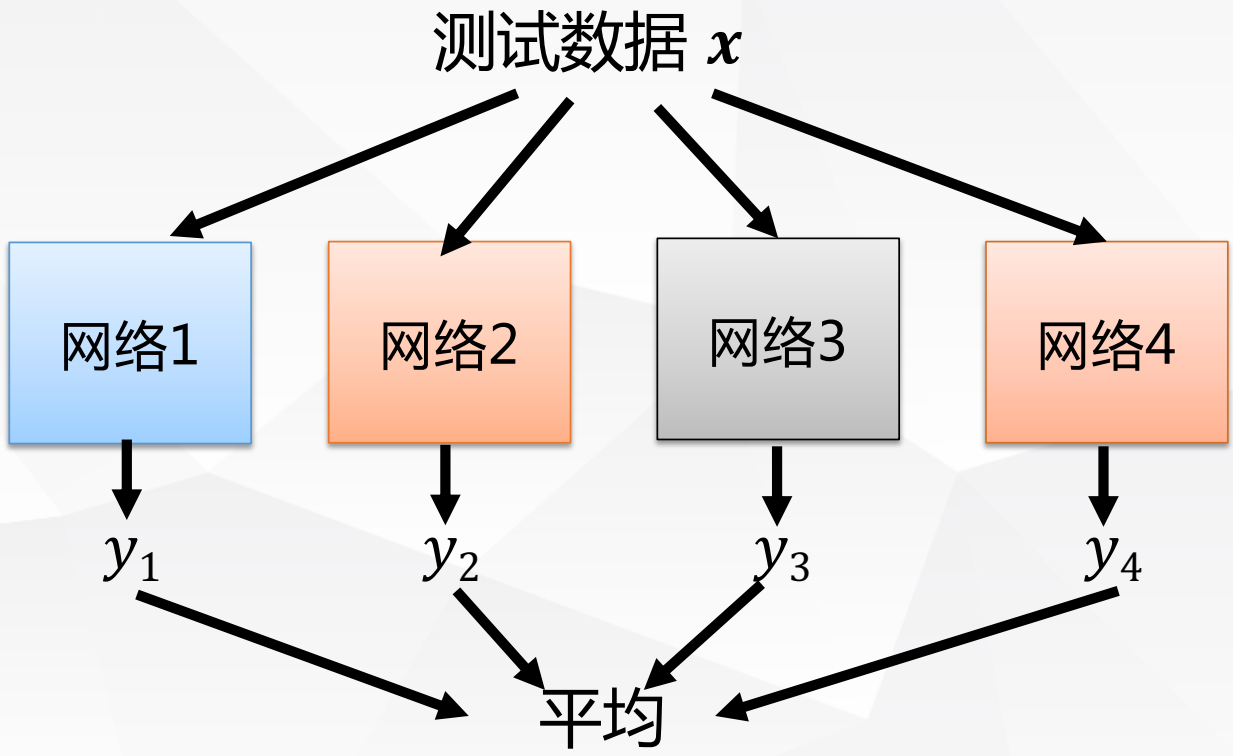
Dropout：可视为一种集成学习



以不同的结构训练多个模型

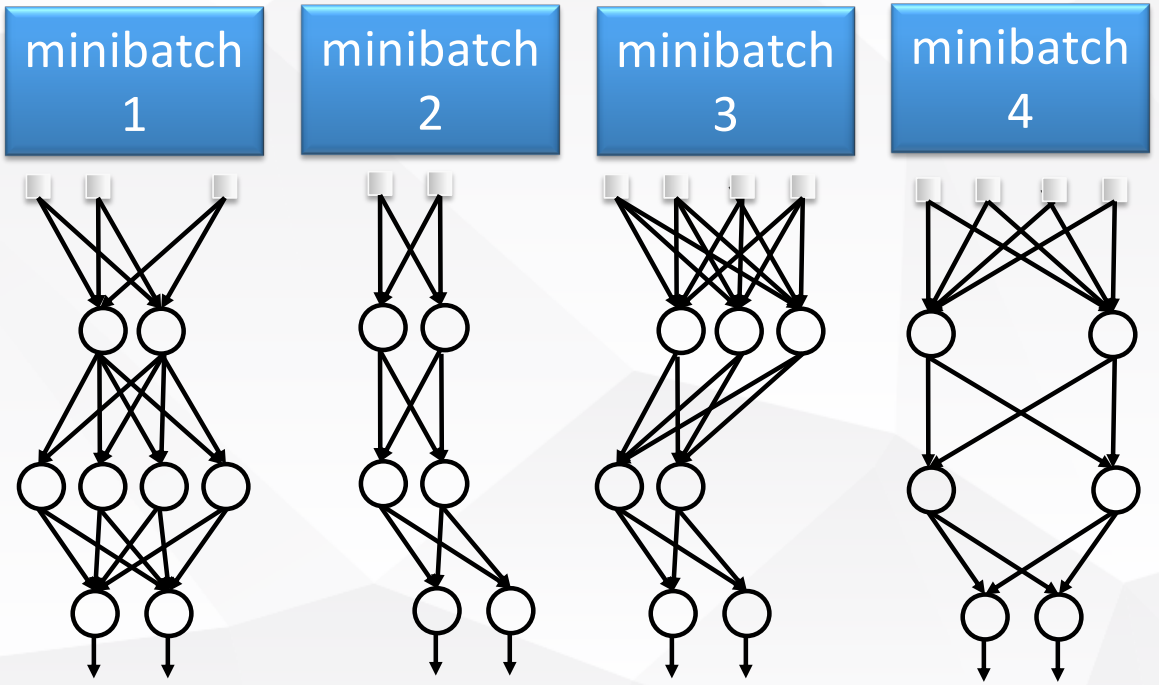
Dropout ：可视为一种集成学习

- 集成学习：结合几个模型降低泛化误差的技术
- 但训练太多模型并集成，计算和存储代价太大



Dropout ：可视为一种集成学习

- Dropout提供了一种近似的、低成本集成方法



带Dropout的训练

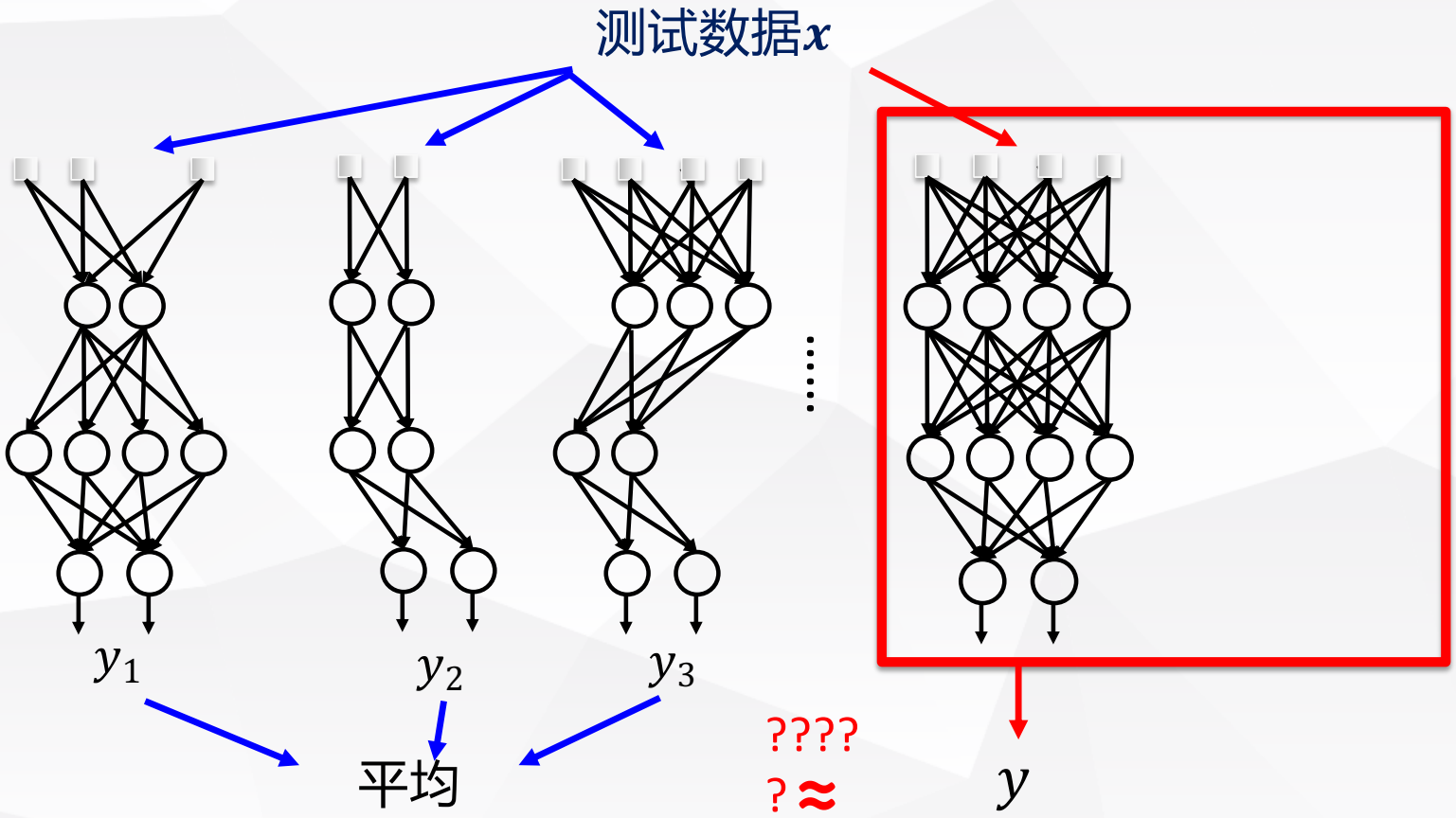
M个神经元

2^M种可能的网络

- 每次用一个mini-batch训练一个网络
- 这些网络中有些参数是共享的。

Dropout ：可视为一种集成学习

带Dropout的测试



Dropout 小结

- Dropout被大量利用于全连接网络
  - 对大型网络中间层，p=0.5是理想的
  - 对于输入层， p应保持在0.2或更低：删除输入数据会对训练产生不利影响
- 在卷积网络隐藏层中由于卷积自身的稀疏化以及稀疏化的ReLu函数的大量使用等原因，Dropout策略在卷积网络隐藏层中使用较少。