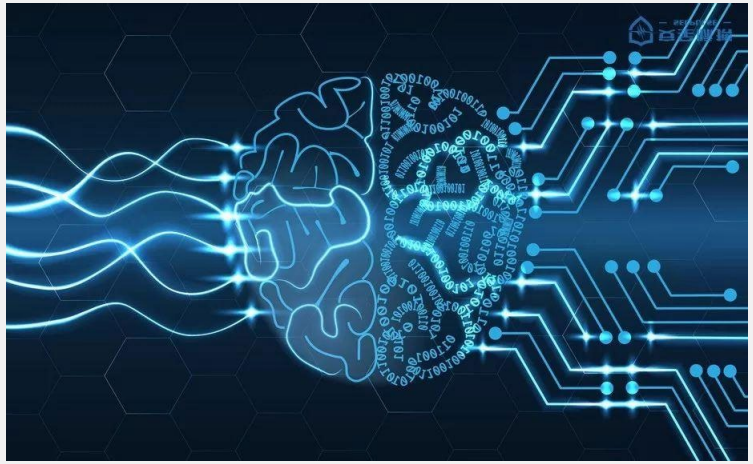


机器学习

Machine Learning

第五章：神经网络（续）

--------深度学习优化方法

目录
CONTENTS

为人师表

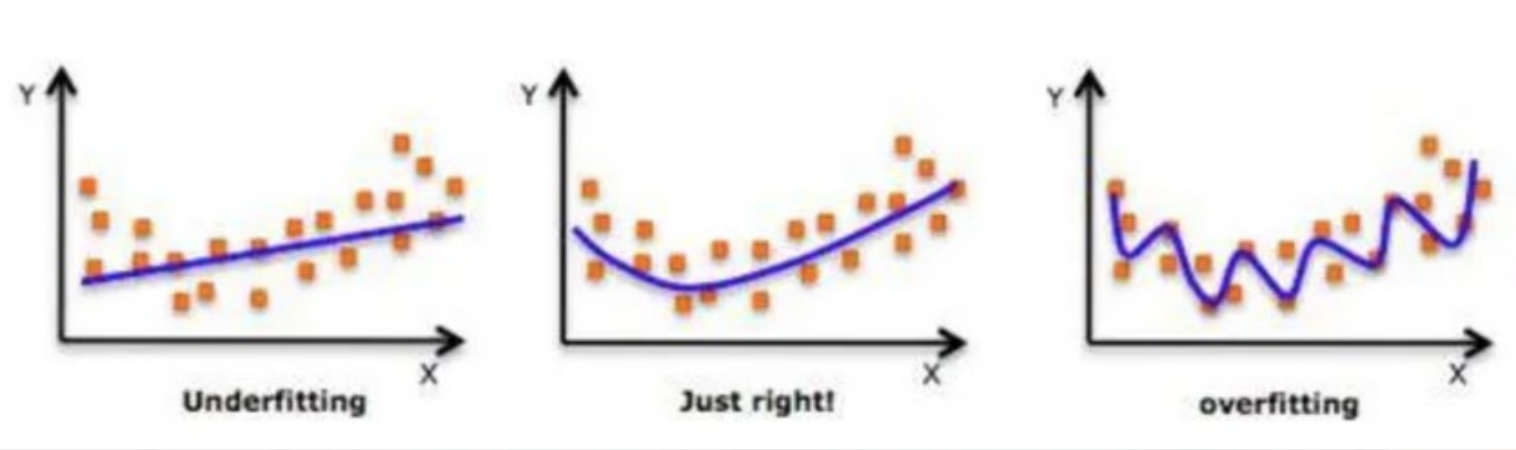
过拟合

- 深度神经网络将训练样本学得“太好”的时候，会把训练样本本身特点当成一般性质，从而导致泛化性能下降，这中现象叫做**过拟合(Overfitting)**

- 产生原因：数据集规模不够；模型太复杂，参数过多；样本里面的噪声数据干扰过大；训练集和测试集特征分布不一样

训练集上误差大， 测试集上误差大	训练集上误差小， 测试集上误差小	训练集上误差小， 测试集上误差大

1.深度学习的正则化



在设计机器学习算法时不仅要求在训练集上误差小，而且希望在新样本上的泛化能力强。许多机器学习算法都采用相关的策略来减小测试误差，这些策略被统称为正则化。因为神经网络的强大的表示能力经常遇到过拟合，所以需要使用不同形式的**正则化策略**。

正则化通过对算法的修改来减少泛化误差

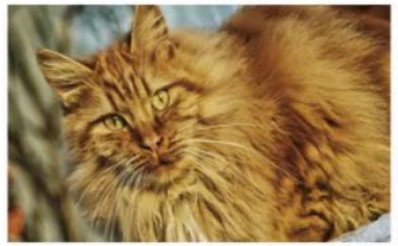
1.深度学习中常见的正则化策略

- **数据预处理：**剔除明显错误样本，通过对原始数据进行微调创造更多的训练数据
- **L1与L2正则化（参数范数惩罚）**：在误差目标函数中增加一项描述网络复杂程度的部分，例如连接权值与阈值的平方和
- **早停**：训练过程中，若训练误差降低，但验证误差升高，则停止训练
- **Dropout**：在训练中暂时丢弃一部分神经元及其连接
- **DropConnect**：将网络架构权重的一个随机选择子集设置为零
- **批标准化（Batch Normalization）**：解决数据分布不一致的问题

数据扩增

- 对已知样本进行简单处理、变换，从而增大数据集，额外生产假训练数据，减小过拟合风险
  - 和全新训练数据相比，提供的信息没那么多，且需要算法验证得到的样本依然表示原对象
  - 优点是简单，成本低

镜像/旋转截取



4

旋转扭曲

L1与L2正则化

- L1和L2是最常见的正则化方法。它们在代价函数（cost function）中增加一个正则项，由于添加了这个正则化项，权重矩阵的值减小
- 具有更小权重矩阵的神经网络导致更简单的模型。 因此它可在一定程度上减少过拟合。

L2正则化

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}||w||_2^2$$

- 这里的λ是正则化参数，它是一个需要优化的超参数

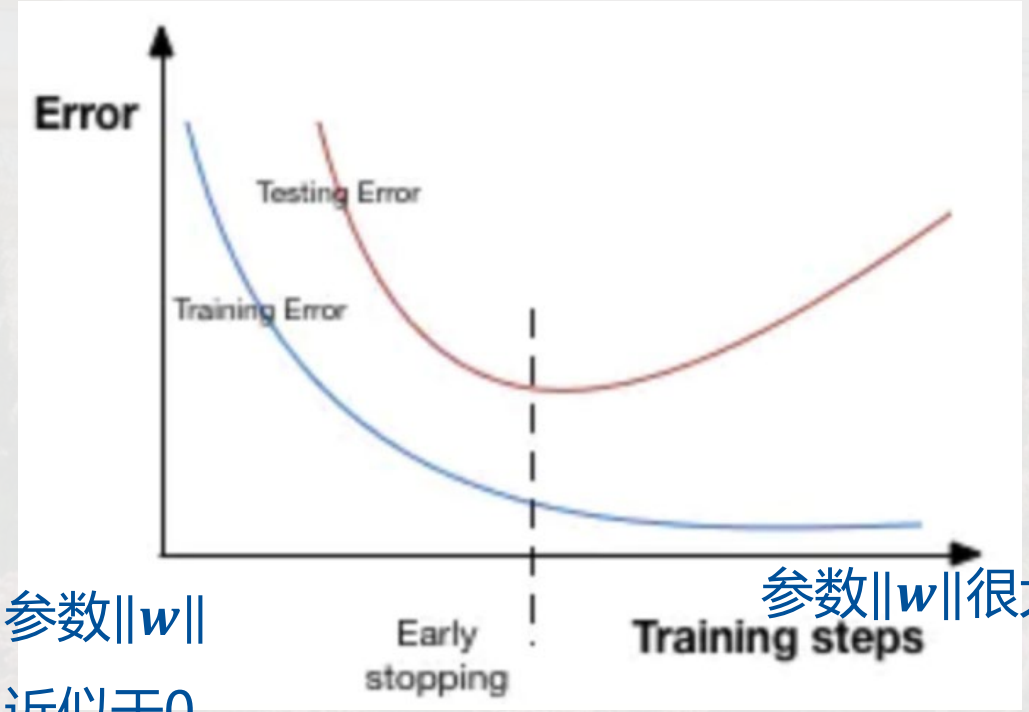
L1正则化

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}||w||_1$$

- 这里，我们惩罚权重矩阵的绝对值。L1对于压缩模型很有用。其它情况下，一般选择优先选择L2正则化

早停

- 提前停止（early stopping）是将一部分训练集作为验证集（validation set）。当验证集的性能越来越差时或者性能不再提升，则立即停止对该模型的训练
  - 提前停止，代价函数可能不够小

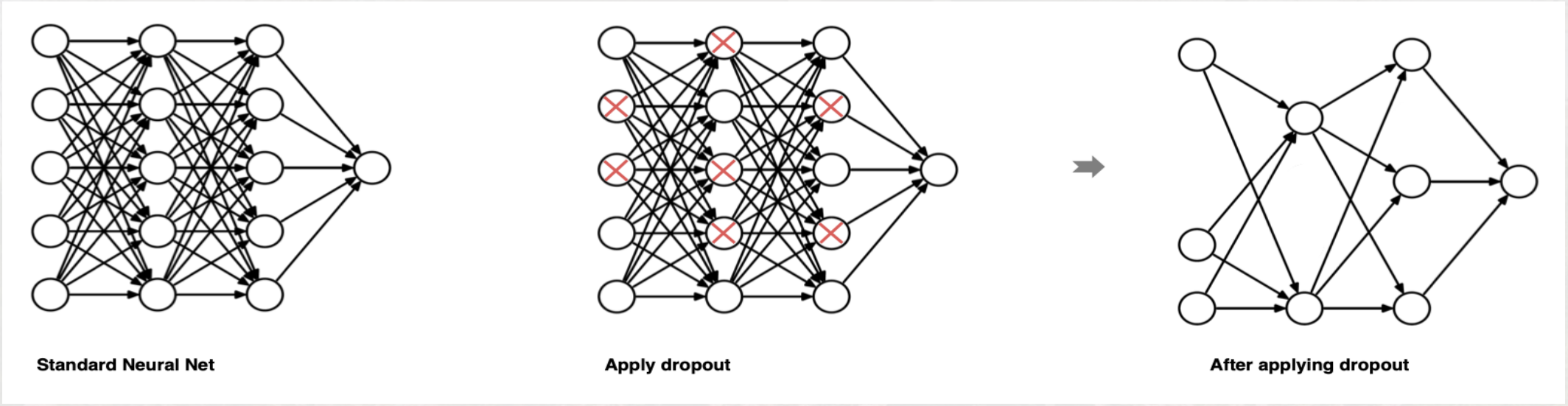


参数$\|w\|$近似于0

参数$\|w\|$很大

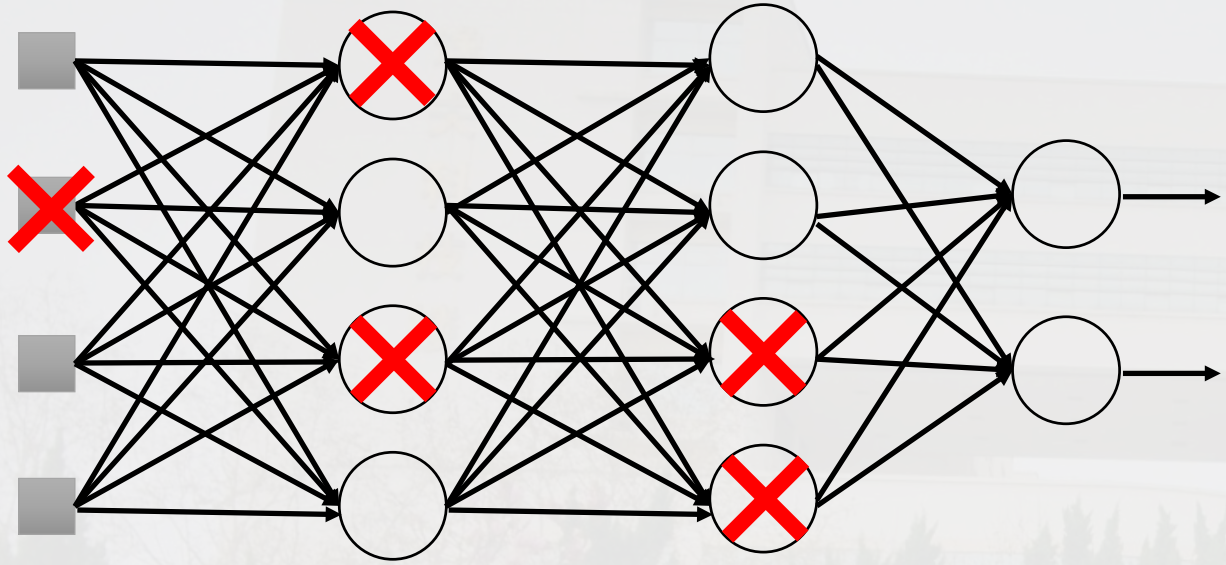
Dropout正则化

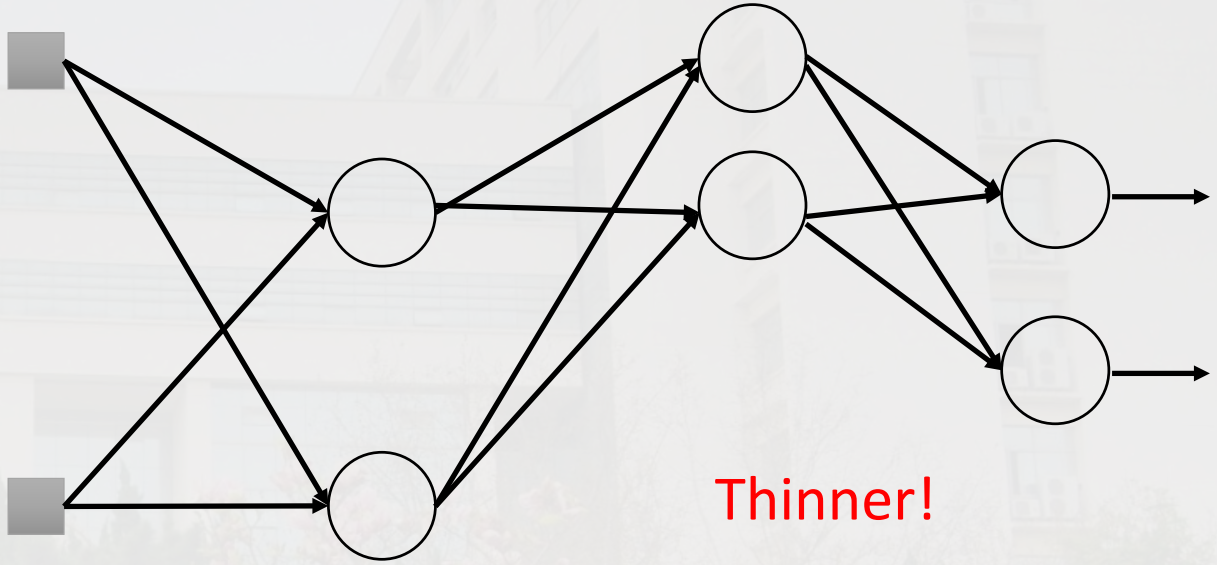
Dropout（随机失活）是在深度学习领域最常用的正则化技术。Dropout的原理很简单：假设我们的神经网络结构如下所示，在每个迭代过程中，随机选择某些节点，并且删除前向和后向连接。



Dropout正则化

- 训练阶段



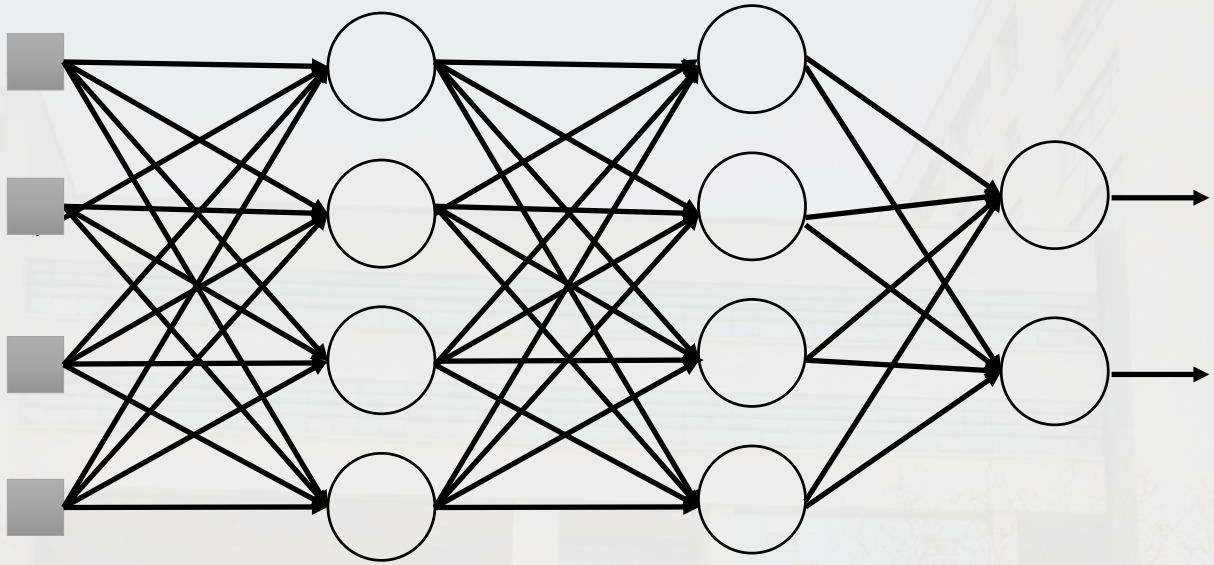


- **在学习参数开始之前**
  - 每个神经元有p的概率失活
    - **网络结构发生变化**
  - 再使用新的网络结构进行训练

对每一批数据，恢复所有神经元，
再重新选择失活的神经元

Dropout正则化

- 测试阶段



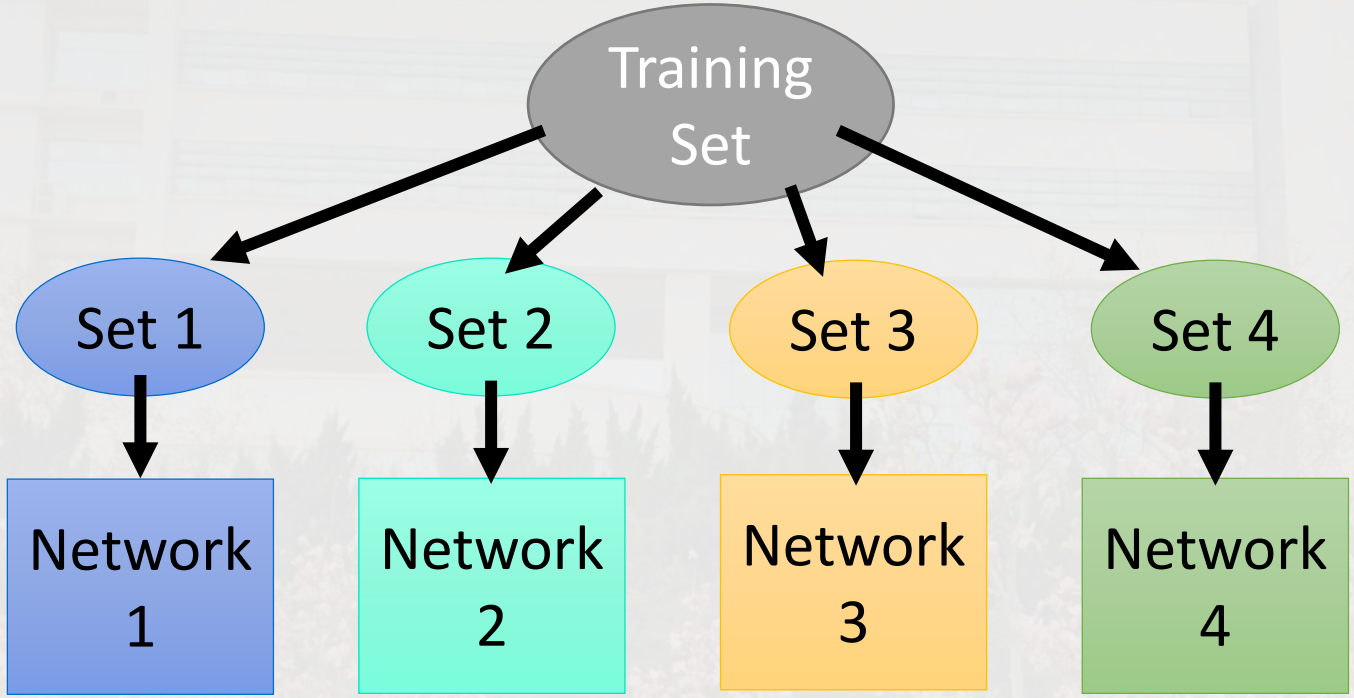
- **没有dropout**

  - 若训练阶段每个神经元失活概率为 p,
    则测试阶段所有的权重都乘上1-p

  - 假设失活概率是50%，且训练得到的所有权重$w = 1$
    则测试阶段的权重$w = 0.5$

Dropout正则化

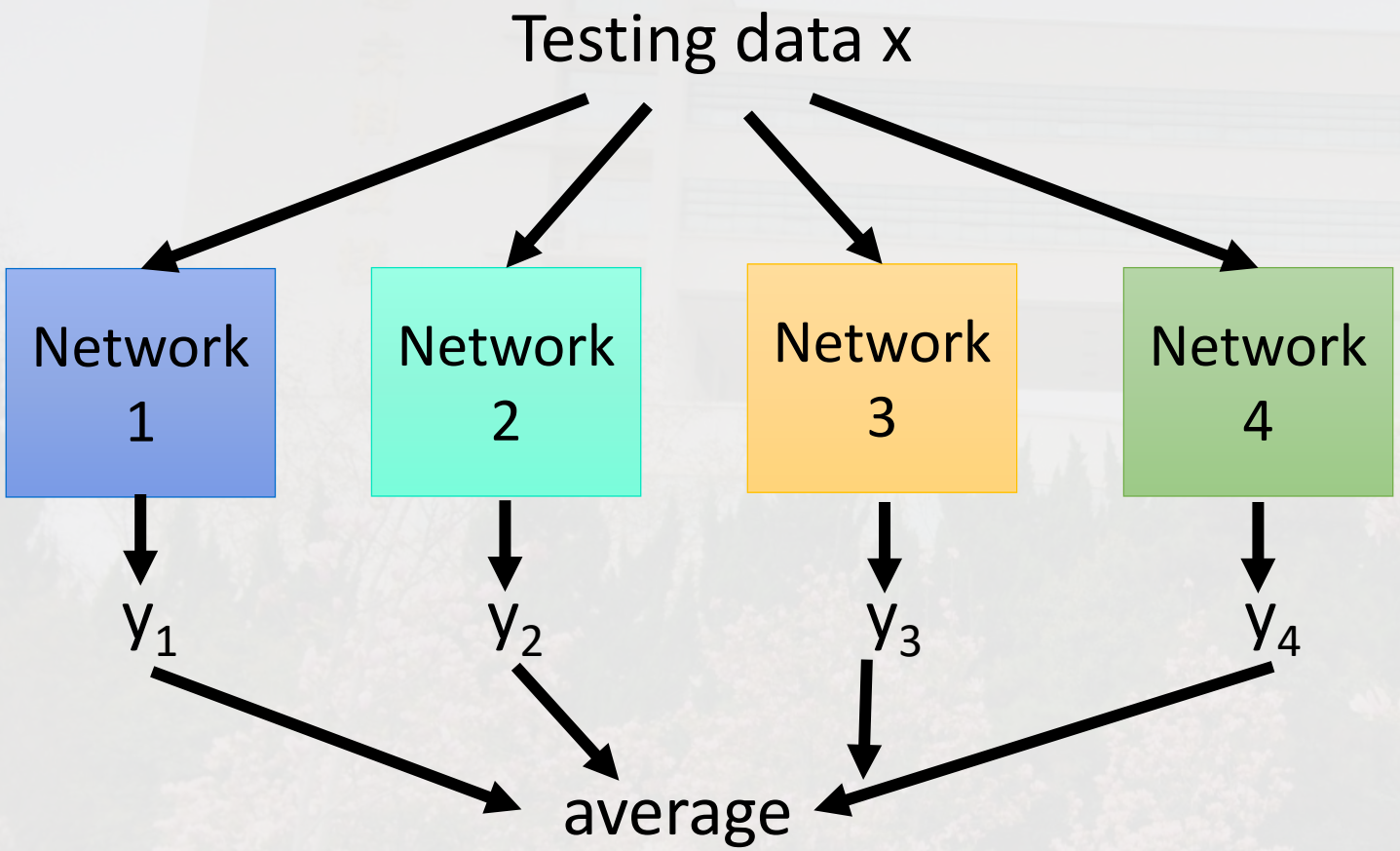
- Dropout可看成是一种集成大量神经网络的方法
  - 将训练集拆分成不同的mini-batch



训练出一批结构不同的神经网络

Dropout正则化

- 测试阶段可看成是将测试数据输入不同的网络得到输出再求平均



- 集成模型一般优于单一模型，因为它们可以捕获更多的随机性。相似地，dropout使得神经网络模型优于正常的模型。

Dropout正则化

- Dropout为什么能缓解过拟合问题?

  - **Dropout可以看作是集成了大量神经网络的方法**

    - 不同的网络可能产生不同的过拟合，取平均则有可能让一些“相反的”拟合互相抵消

  - **Dropout能够减少神经元之间复杂的共适应（co-adaptation）关系**

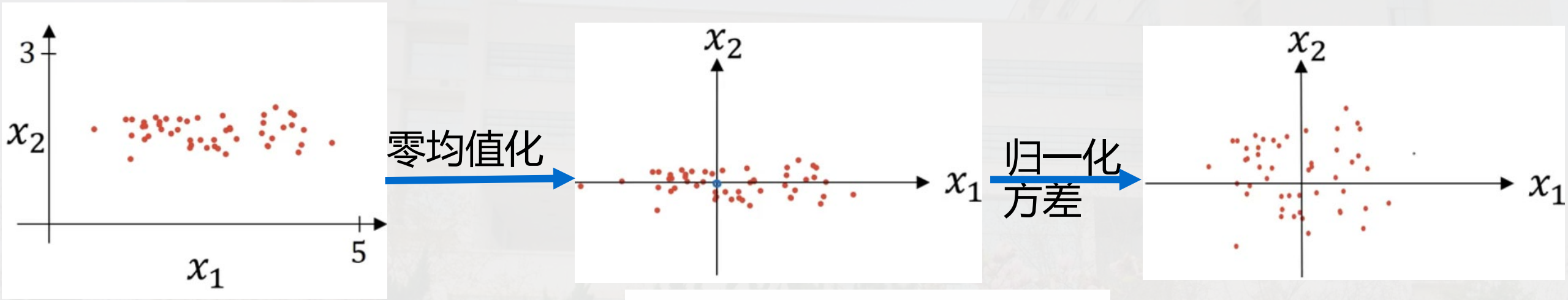
    - Dropout每次丢弃的神经元呈随机选择的，网络权值的更新不会依赖于隐节点之间的固定关系即网络中每个神经元不会对另一个特定神经元的激活非常敏感，这使得网络能够学习到一些更加泛化的特征。

Dropout正则化

- 关于dropout的其他参考文献 [Nitish Srivastava, JMLR'14] [Pierre Baldi, NIPS'13][Geoffrey E. Hinton, arXiv'12]
- Dropout 与激活函数 Maxout 一块使用效果更好 [Ian J. Goodfellow, ICML'13]
- Dropconnect [Li Wan, *ICML'13*]
  - Dropout 随机删除神经元
  - Dropconnect 随机删除神经元之间的连接
- Annealed dropout [S.J. Rennie, SLT'14]
  - 随着迭代次数的增加，失活的概率减小
- Standout [J. Ba, NISP'13]
  - 每个神经元的失活概率不一样

•批标准化

- 批标准化(BN层,Batch Normalization) 利用网络训练时一个 mini-batch 的数据来计算输入x_i的均值和方差,归一化后并重构



$$\mu = \frac{1}{m}\sum_i z^{(i)}$$

$$\bar{z}^{(i)} := z^{(i)} - \mu$$

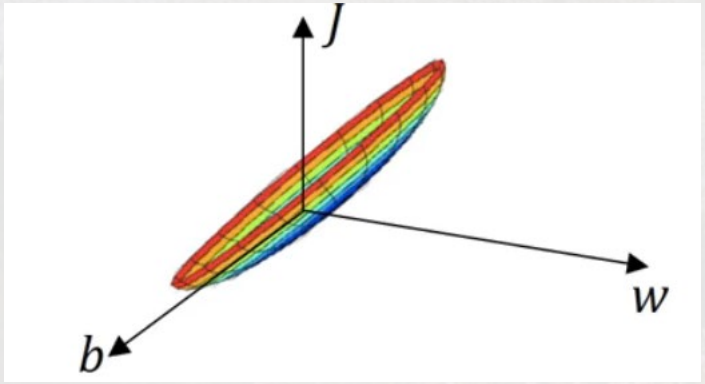
$$\sigma^2 = \frac{1}{m}\sum_i (z_i - \mu)^2$$

$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

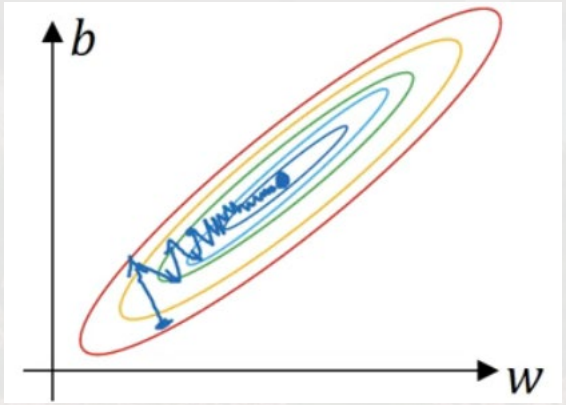
$$z_{norm}^{(i)} := \gamma z_{norm}^{(i)} + \beta$$

批标准化

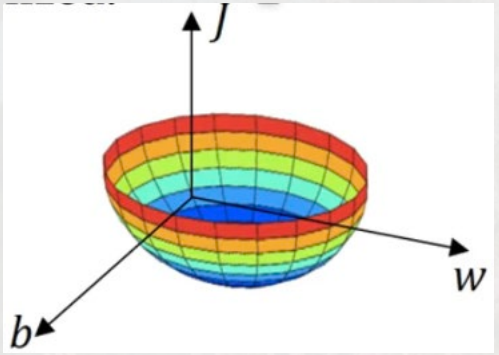
- 批标准化(BN层,Batch Normalization)是2015年提出的一种方法，在进行深度网络训练时，大多会采取这种算法，与全连接层一样，BN层也是属于网络中的一层。在每一层输入之前，将数据进行BN，然后再送入后续网络中进行学习
- 为什么需要进行批标准化?



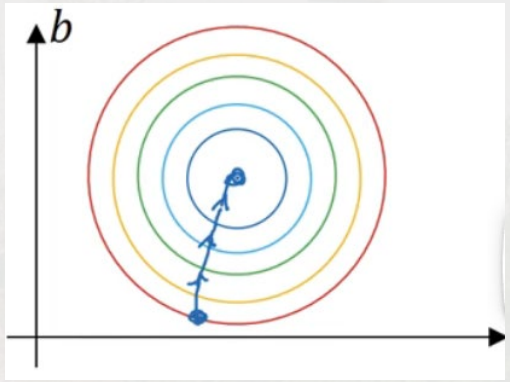
非标准化的代价函数



梯度下降过程



批标准化后的代价函数



梯度下降过程

批标准化

- 批标准化实现

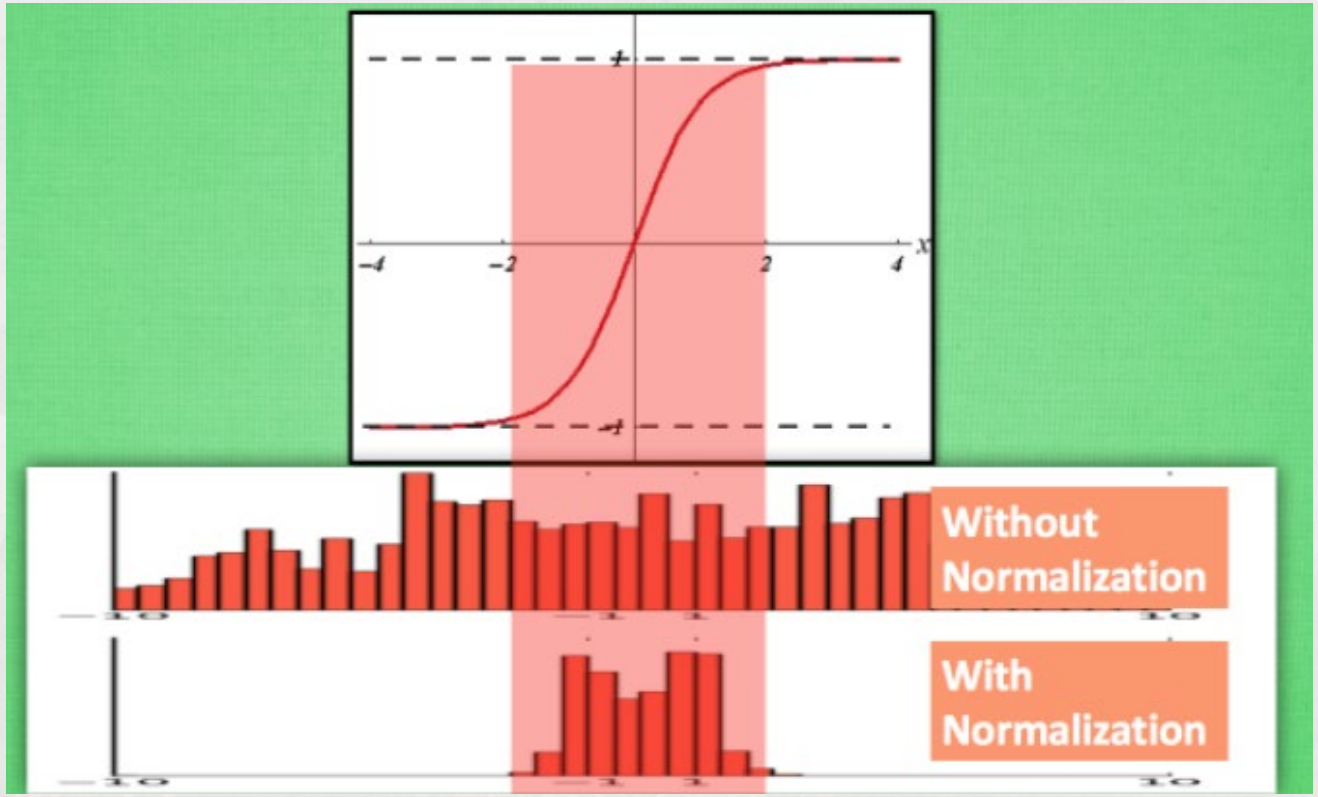
```
# 直接将其放入构建神经网络的结构中即可
tf.keras.layers.BatchNormalization(
    epsilon=0.001, center=True, scale=True,
    beta_initializer='zeros', gamma_initializer='ones',
)
```

参数	意义
epsilon	防止分母为0
center	如果为True，则将的偏移beta量添加到标准化张量。如果为False，则将beta 被忽略。
scale	如果为True，则乘以gamma。如果为False，gamma则不使用。
beta_initializer	Beta权重的初始化程序。
gamma_initializer	伽玛权重的初始化程序。

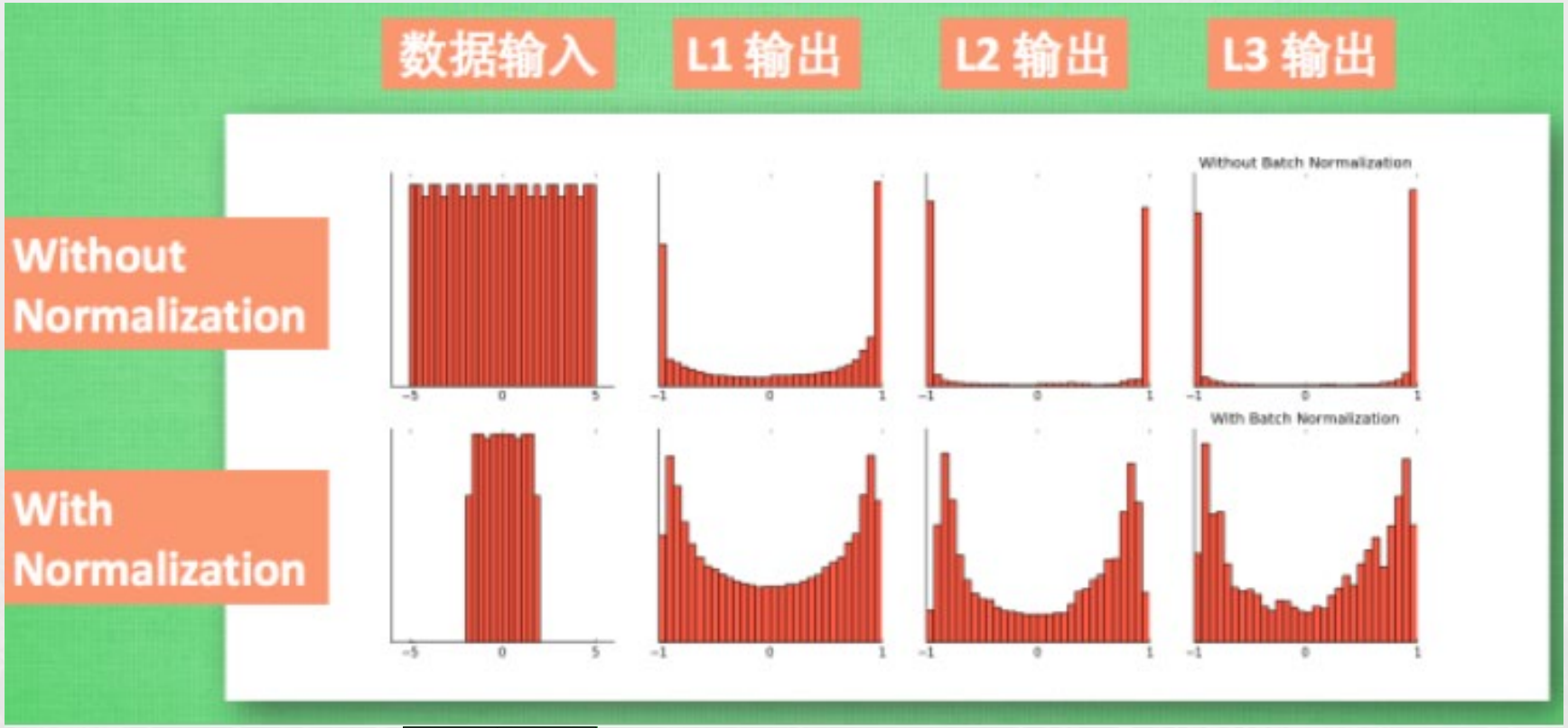
批标准化

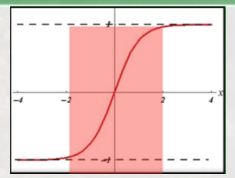
- 计算结果值的分布对于激励函数很重要。对于数据值大多分布在这个区间的数据，才能进行更有效的传递。对比这两个在激活之前的值的分布。

- 上者没有进行 normalization, 下者进行了 normalization，这样当然是下者能够更有效地利用 tanh 进行非线性化的过程。



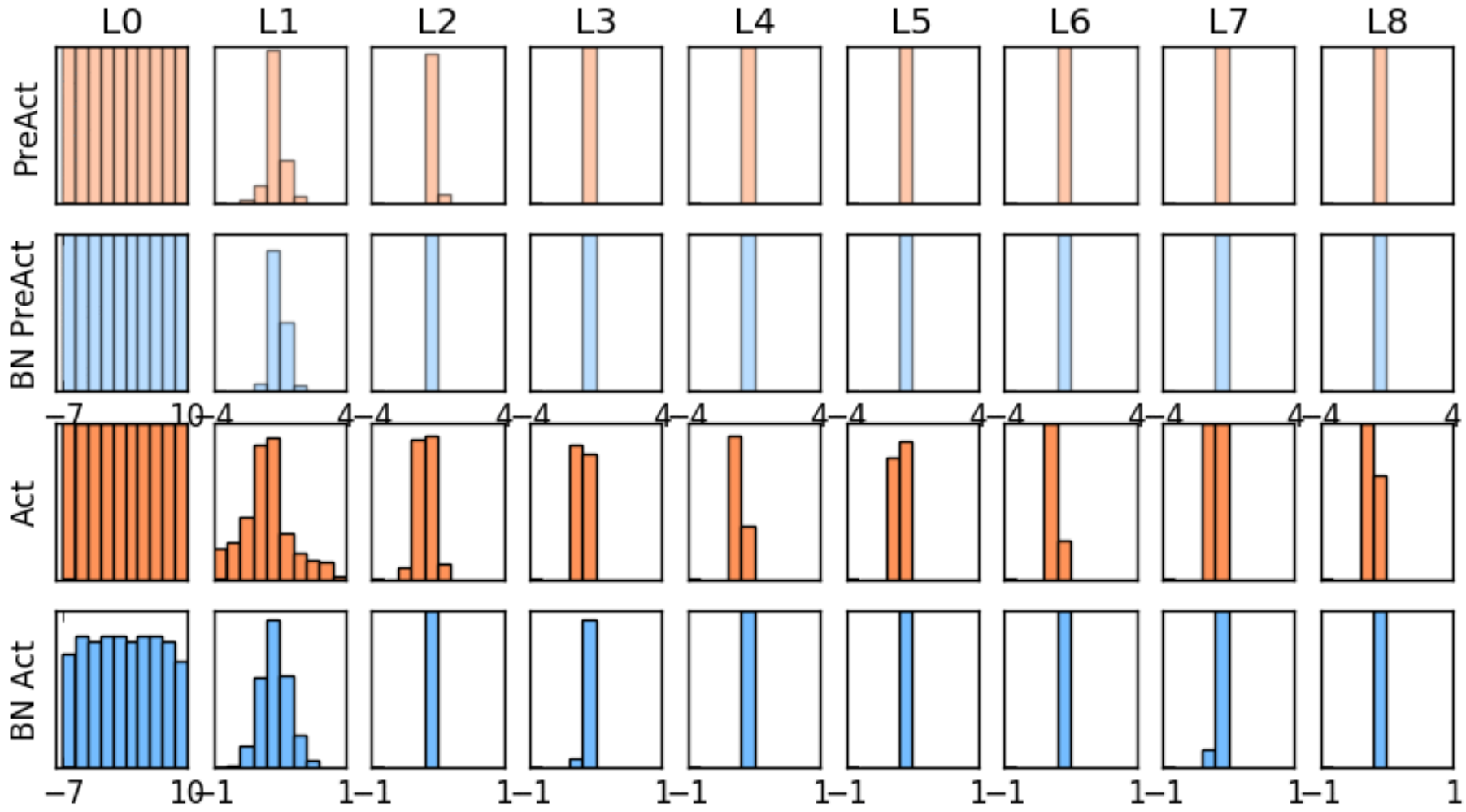
批标准化


数据输入
L1 输出
L2 输出
L3 输出
Without Normalization
With Normalization
Without Batch Normalization
With Batch Normalization



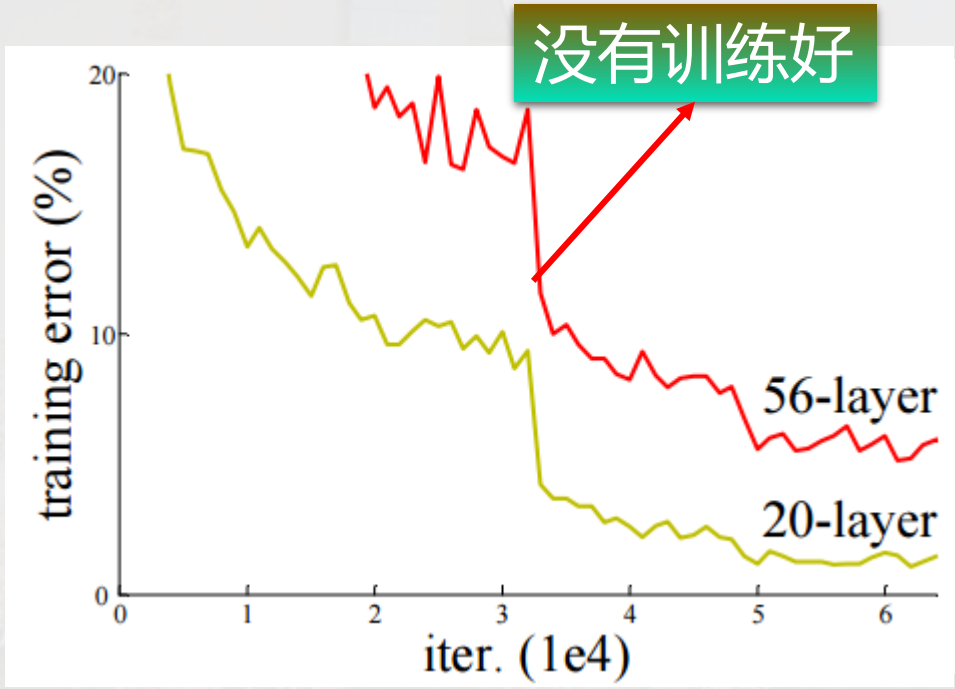
激活函数

批标准化

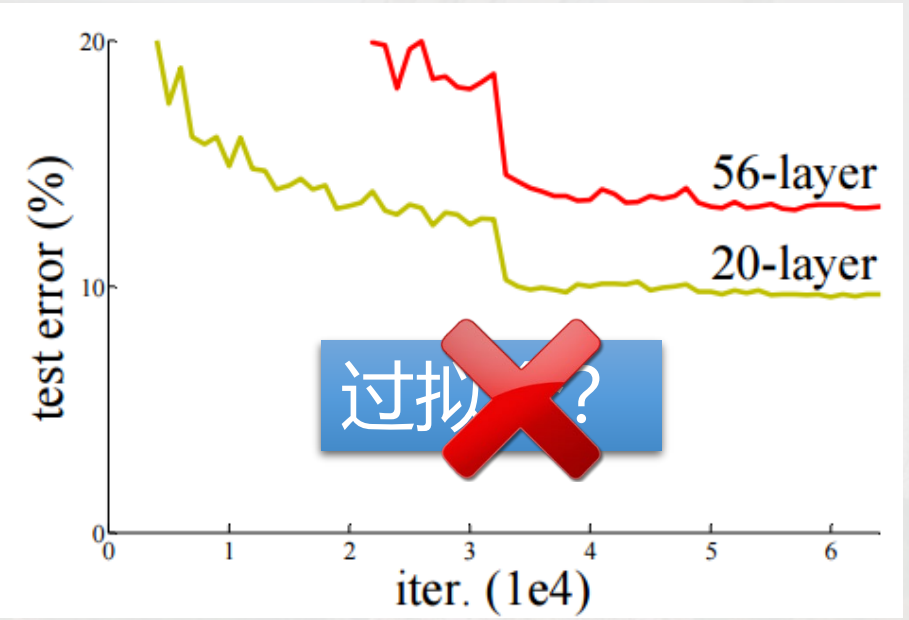


注意事项

- 过拟合常见，但并非性能不好的原因都是过拟合造成



训练数据



测试数据

Deep Residual Learning for Image Recognition
http://arxiv.org/abs/1512.03385

2 梯度下降算法

- 梯度下降法是一种寻找使损失函数最小化的方法。
- 从数学角度来看，梯度的方向是函数增长速度最快的方向，梯度的反方向就是函数减少最快的方向

$$w_{ij}^{new} = w_{ij}^{old} - \eta \frac{\partial E}{\partial w_{ij}}$$

其中，η是学习率，如果学习率太小，那么每次训练之后得到的效果都太小，增大训练的时间成本。如果，学习率太大，那就有可能直接跳过最优解，进入无限的训练中。解决的方法就是，学习率也需要随着训练的进行而变化。

2 梯度下降算法

- 一维梯度下降例题

$y = x^2$，通过梯度下降算法求y取最小值（极小值）时候的最优解x

求解过程主要通过迭代完成，迭代的方程为：　$x = x - y'(x) * \alpha$

其中x为要求的解，y'(x)为梯度（也就是导数或偏导，我们在这里用最简单的一元函数演示，所以直接写成导数）α为学习率（或称步长，是一个重要的参数，α的选择直接影响这着算法的效率）。

2 梯度下降算法

➢ 一维梯度下降例题

$y = x^2$，通过梯度下降算法求y取最小值（极小值）时候的最优解x

算法过程：

1）. 首先任取一点，我们取x = 3，计算其导数y'(x) = 6

2）. 设定学习率为α = 0.4

3）. 开始算法的迭代：

（1）x = 3，y'(x) = 6，x = x - y'(x) * α = 0.6

（2）x = 0.6，y'(x) = 1.2，x = x - y'(x) * α = 0.12

（3）x = 0.12，y'(x) = 0.24，x = x - y'(x) * α = 0.024

（4）x = 0.024，y'(x) = 0.048，x = x - y'(x) * α = 0.0048 ...

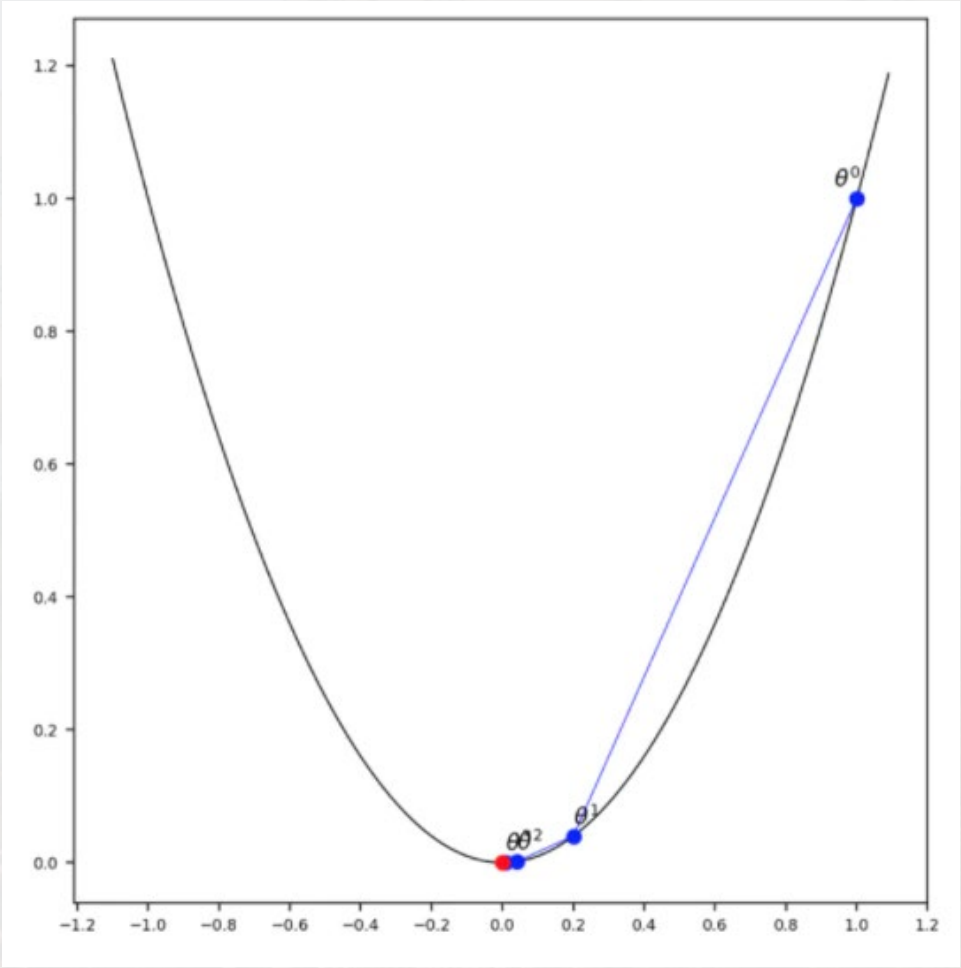
4）. 当梯度（导数）下降到很小或为0时，则求得的解x趋向最优解，比如本例中迭代到第四步时y'(x) = 0.048已经非常小了，x = 0.0048基本趋向于本例的真正解x = 0

2 梯度下降算法

- 一维梯度下降例题

假设$J(\theta)=\theta^2$，则导数为$J'(\theta) = 2\theta$，我们选择$\theta^0=1$，$\alpha=0.4$，则:

$$
\begin{aligned}
\theta^0 &= 1 \\
\theta^1 &= \theta^0 - \alpha * J'(\theta^0) \\
&= 1 - 0.4 * 2 \\
&= 0.2 \\
\theta^2 &= \theta^1 - \alpha * J'(\theta^1) \\
&= 0.04 \\
\theta^3 &= 0.008 \\
\theta^4 &= 0.0016 \\
&\cdots
\end{aligned}
$$

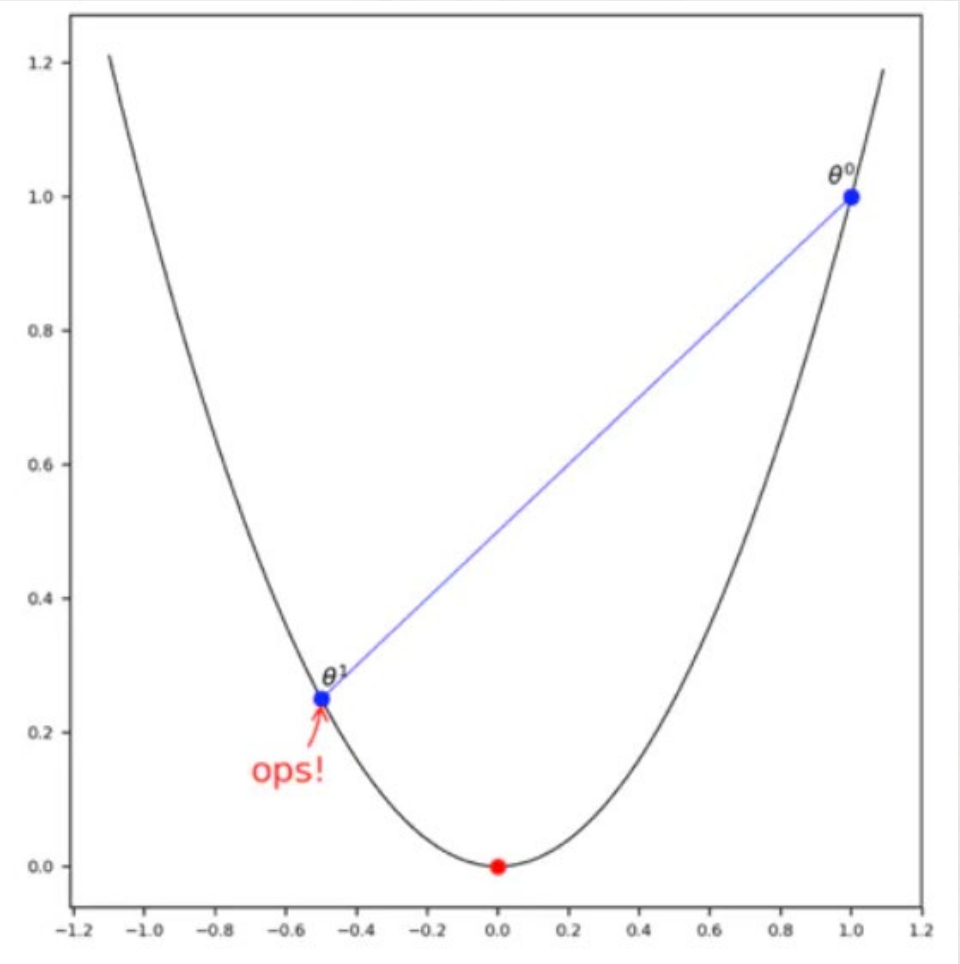


2 梯度下降算法

- 一维梯度下降例题

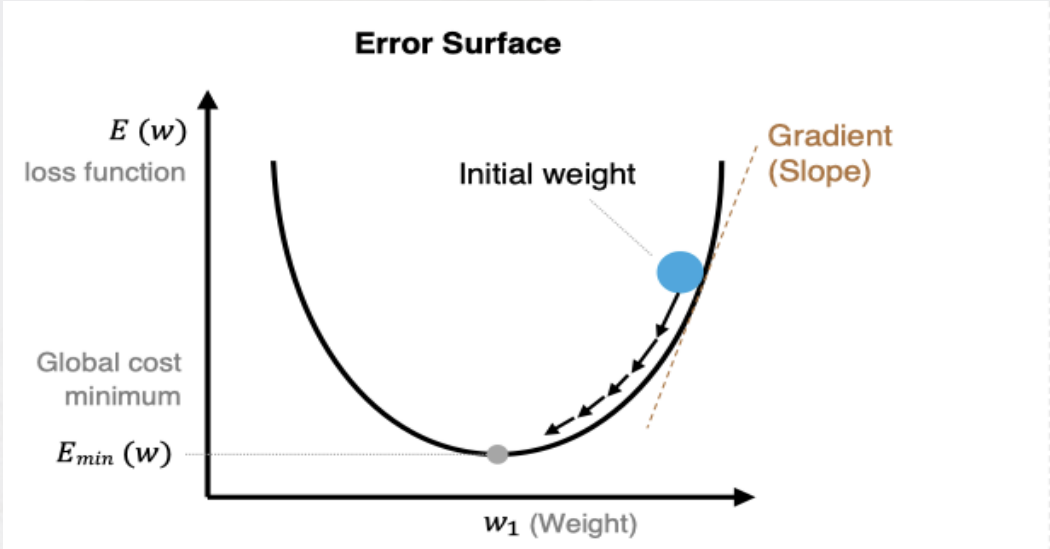
而当我们将学习率设置为α=0.75时：

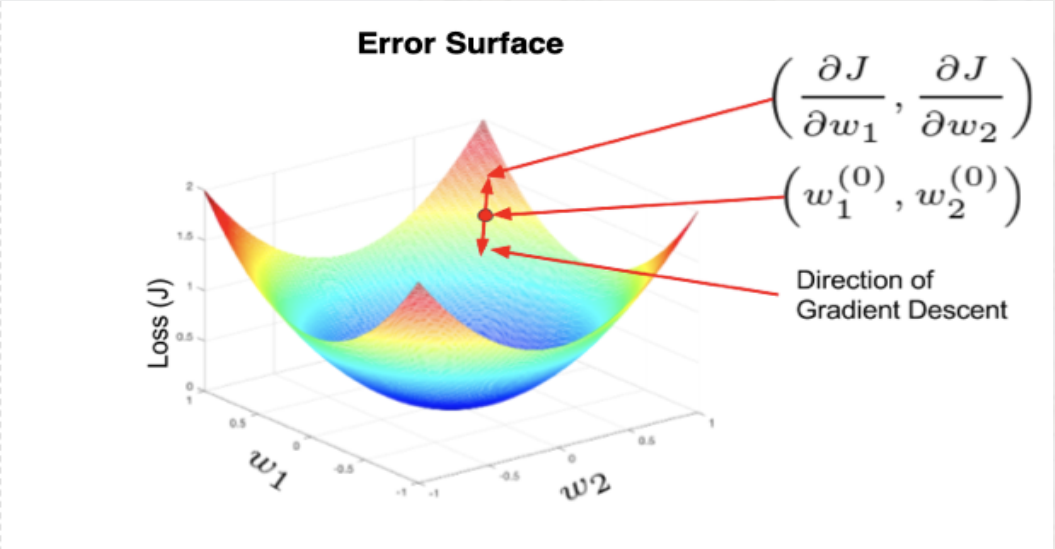
如图所示，优化后数值到了最低点左侧的位置，并且继续向左偏移，错过了最低点：



2 梯度下降算法

- 小结





在上图中我们展示了一维和多维的代价函数，代价函数呈碗状。在训练过程中代价函数对权重的偏导数就是代价函数在该位置点的梯度。我们可以看到，沿着负梯度方向移动，就可以到达代价函数底部，从而使代价函数最小化。这种利用代价函数的梯度迭代地寻找局部最小值的过程就是梯度下降的过程。

2 梯度下降算法

根据在进行迭代时使用的样本量，将梯度下降算法分为以下三类：

梯度下降算法	定义	缺点	优点
BGD（批量梯度下降）	每次迭代时需要计算每个样本上损失函数的梯度并求和	计算量大、迭代速度慢	全局最优化
SGD（随机梯度下降）	每次迭代时只 采集一个样本，计算这个样本损失函数的梯度并更新参数	准确度下降、存在噪音、非全局最优化	训练速度快、支持在线学习
MBGD（小批量梯度下降）	每次迭代时，我们随机选取一小部分训练样本来计算梯度 并更新参数	准确度不如BGD、非全局最优解	计算小批量数据的梯度更加高效、支持在线学习

3 深度学习优化器

3.1 BGD优化器

BGD 的全称是 Batch Gradient Descent，中文名称是批量梯度下降。顾名思义，BGD 根据整个训练集计算梯度进行梯度下降

$$\theta=\theta-\eta\Delta\theta L(\theta)$$

其中，$L(\theta)$ 是根据整个训练集计算出来的损失。

优点：

当损失函数是凸函数（convex）时，BGD 能收敛到全局最优；当损失函数非凸（non-convex）时，BGD 能收敛到局部最优；

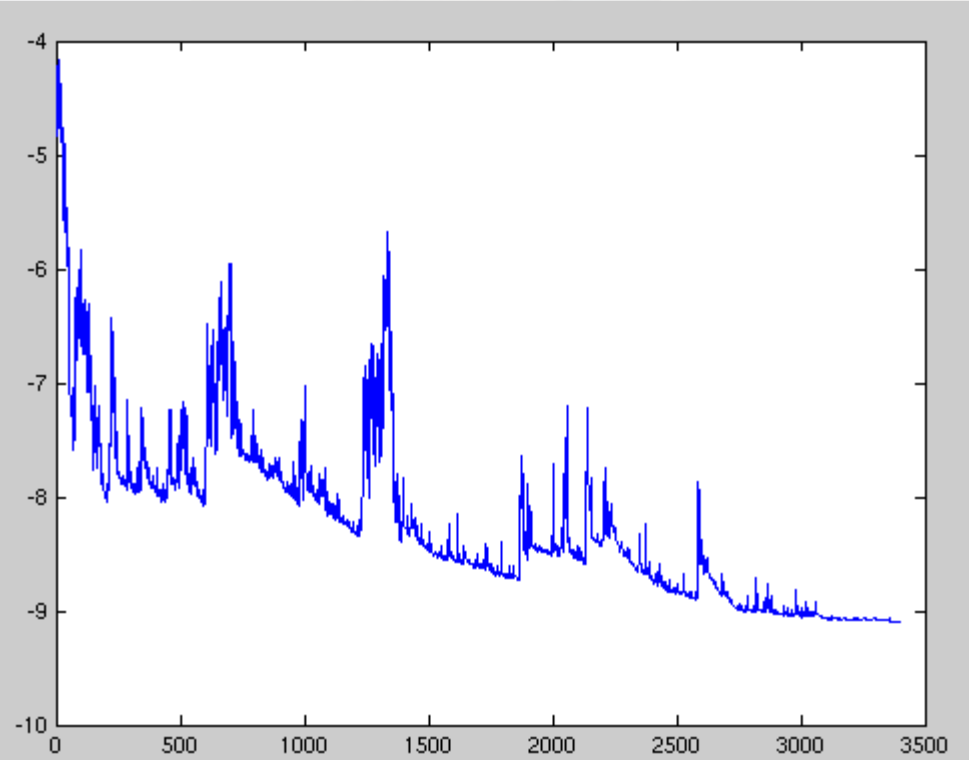
缺点：

（1）每次都要根据全部的数据来计算梯度，速度会比较慢；

（2）BGD 不能够在线训练，也就是不能根据新数据来实时更新模型；

3 深度学习优化器

3.2 SGD优化器



SGD 的全称是 Stochastic Gradient Descent，中文名称是随机梯度下降。和 BGD 相反，SGD 每次只使用一个训练样本来进行梯度更新：

$$\theta=\theta-\eta\Delta_\theta L(\theta;x^{(i)};y^{(i)})$$

其中，$L(\theta;x^{(i)};y^{(i)})$ 是只根据样本 $(x^{(i)};y^{(i)})$ 计算出的损失。

优点：

（1）SGD 每次只根据一个样本计算梯度，速度较快；

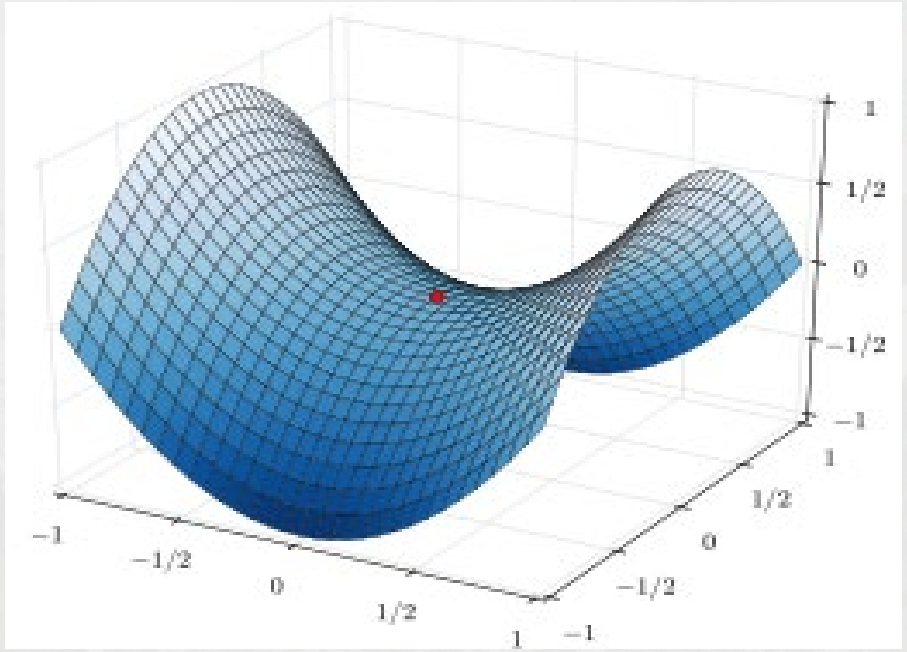
（2）SGD 可以根据新样本实时地更新模型；

缺点：

SGD 在优化的过程中损失的震荡会比较严重；

3 深度学习优化器

3.3 MBGD优化器



MBGD 的全称是 Mini-batch Gradient Descent，中文名称是小批量梯度下降。MBGD 是 BGD 和 SGD 的折中。MBGD 每次使用包含 m 个样本的小批量数据来计算梯度：

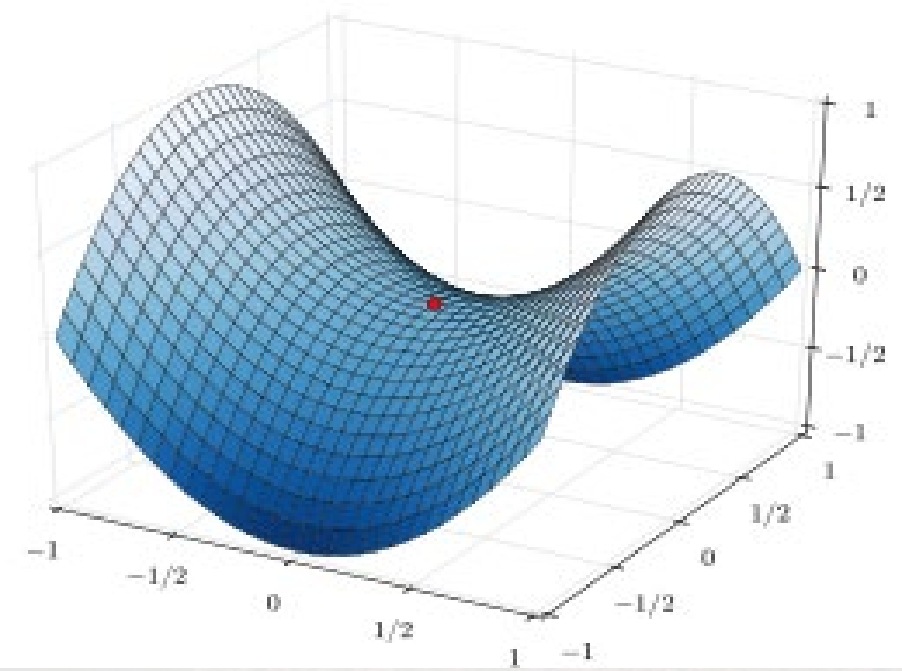
$$\theta=\theta-\eta\Delta_{\theta}L(\theta;x^{(i:i+m)};y^{(i:i+m)})$$

其中，m 为小批量的大小，范围是 [1,n]，n 为训练集的大小；$L(\theta;x^{(i:i+m)};y^{(i:i+m)})$ 是根据第 i 个样本到第 i+m 个样本计算出来的损失。

当 m==1 时，MBGD 变为 SGD；当 m==n 时，MBGD 变为 BGD。

3 深度学习优化器

3.3 MBGD优化器



优点

（1）收敛更加稳定；

（2）可以利用高度优化的矩阵库来加速计算过程；

缺点

（1）选择一个合适的学习率比较困难；

（2）相同的学习率被应用到了所有的参数，我们希望对出现频率低的特征进行大一点的更新，所以我们希望对不同的参数应用不同的学习率；

（3）容易被困在鞍点（saddle point）；

左图的红点就是一个鞍点，上面 MBGD 的 3 个缺点也是 SGD 和 BGD 的 3 个缺点。为了解决这 3 个缺点，研究人员提出了 Momentum、Adagrad、RMSprop、Adadelta、Adam 等优化器。

3 深度学习优化器

❑在SGD中，所有的参数都是用相同的学习率

SGD: $w \leftarrow w - \eta \partial L / \partial w$

❑AdaGrad可以独立的调整每一个参数的学习率

Adagrad: $w \leftarrow w - \eta_w \partial L / \partial w$

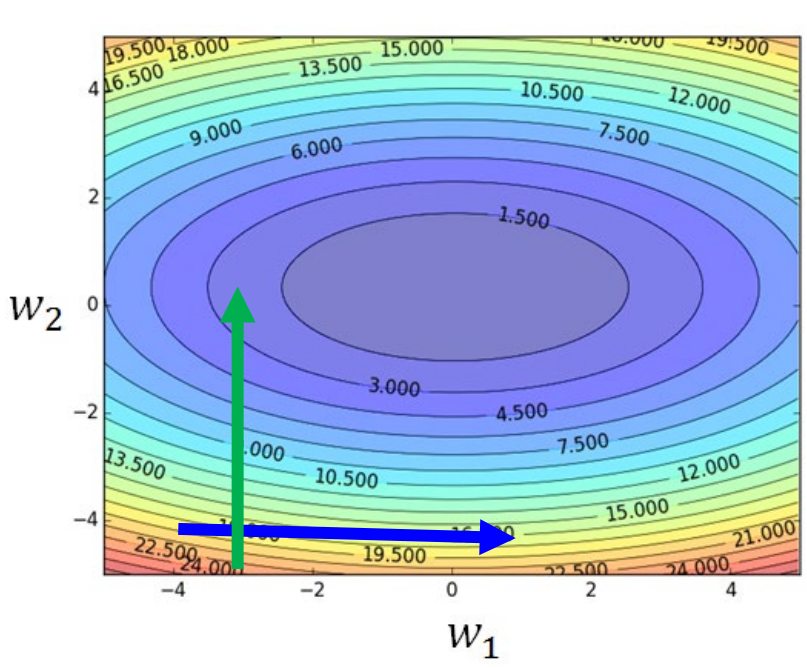
$$\eta_w = \frac{\eta}{\sqrt{\sum_{i=0}^{t} (g^i)^2}}$$

与参数相关的学习率

g^i 表示第i次迭代的梯度

$(g^i)^2 = g^i \odot g^i$

大梯度，小学习率



小梯度，大学习率

✓ 让梯度大的参数的学习率较小

✓ 梯度小的学习率较大，来加快模型的收敛速度。

3 深度学习优化器

- RMSprop
  - AdaGrad中，学习率的分母（梯度的平方和）随着迭代次数的增加，数值一定越来越大，所以这种累加的方法可能会导致学习率变得非常之小，从而参数更新停滞。

$$r^i = r^{i-1} + \left(g^i\right)^2$$

$$\eta_w = \frac{\eta}{\sqrt{\sum_{i=0}^{t}(g^i)^2}}$$

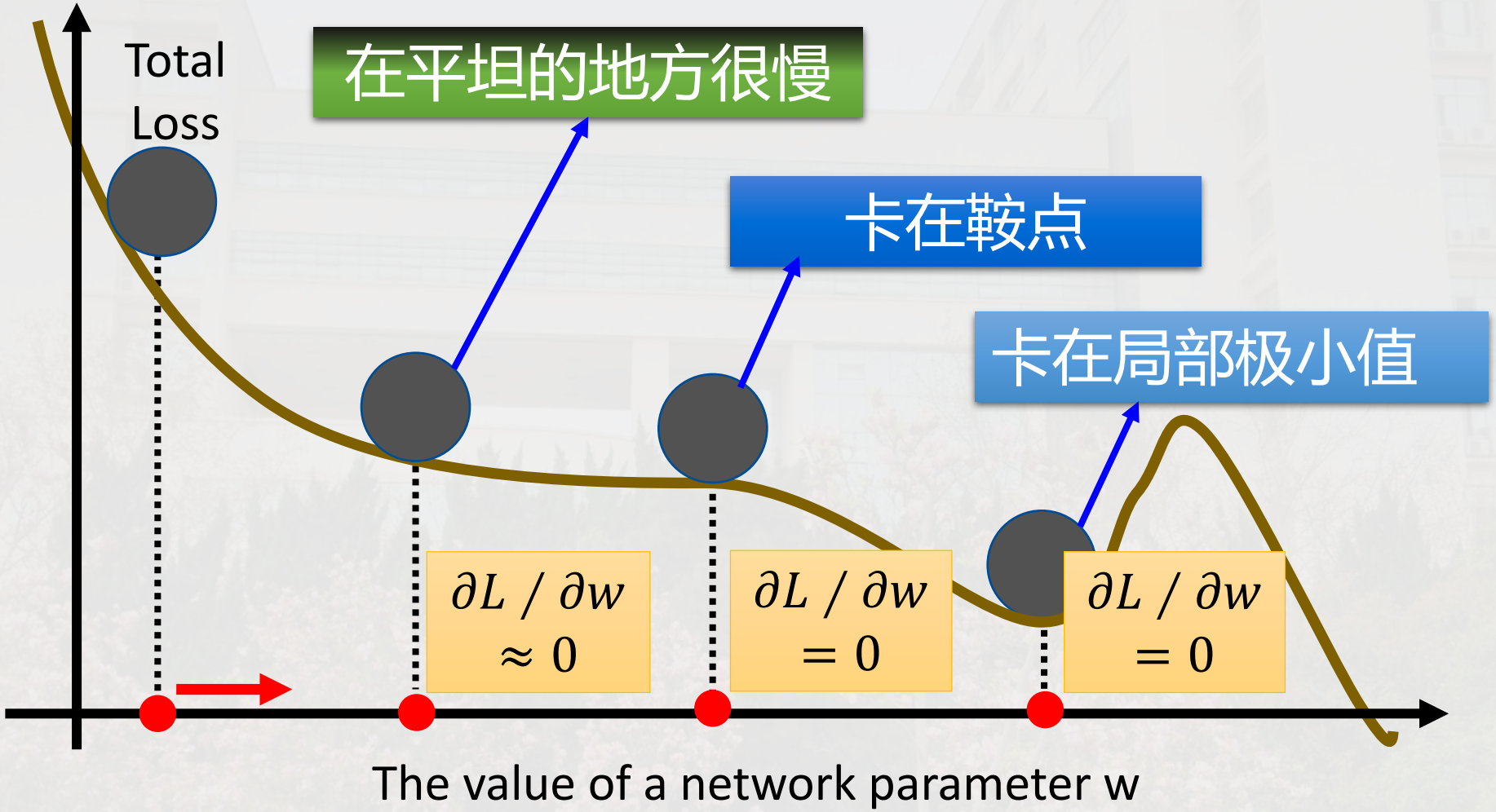
$$r^i = \rho r^{i-1} + (1-\rho)\left(g^i\right)^2$$

$$\eta_w = \frac{\eta}{\sqrt{\sum_{i=0}^{t}\rho^{t-i}(1-\rho)(g^i)^2}}$$

  - RMSprop中，通过指数衰减来丢弃很久以前的梯度信息
  - 故RMSprop也叫改进的AdaGrad（advanced AdaGrad）

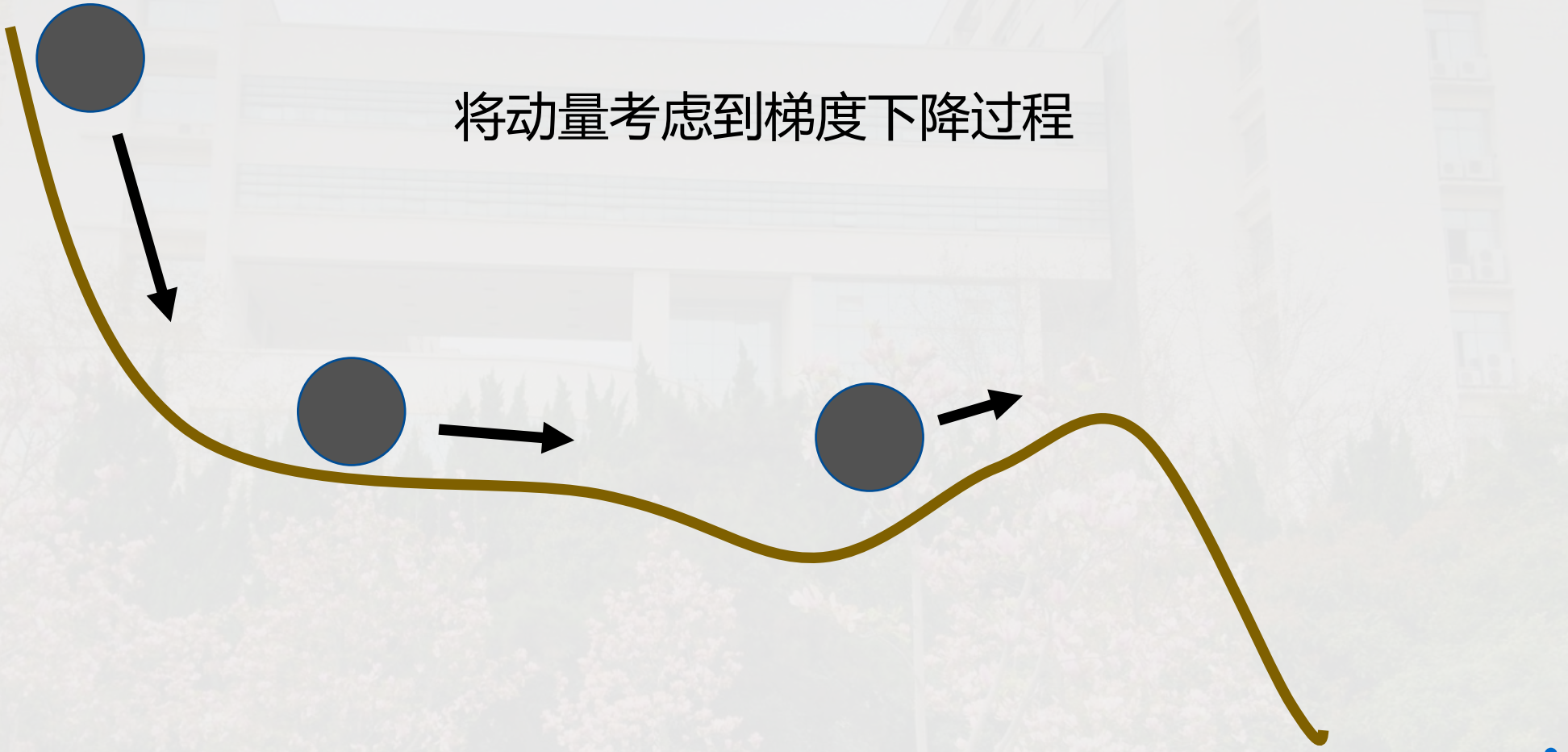
3 深度学习优化器

- 使用原始的梯度下降法，可能会出现如下问题：



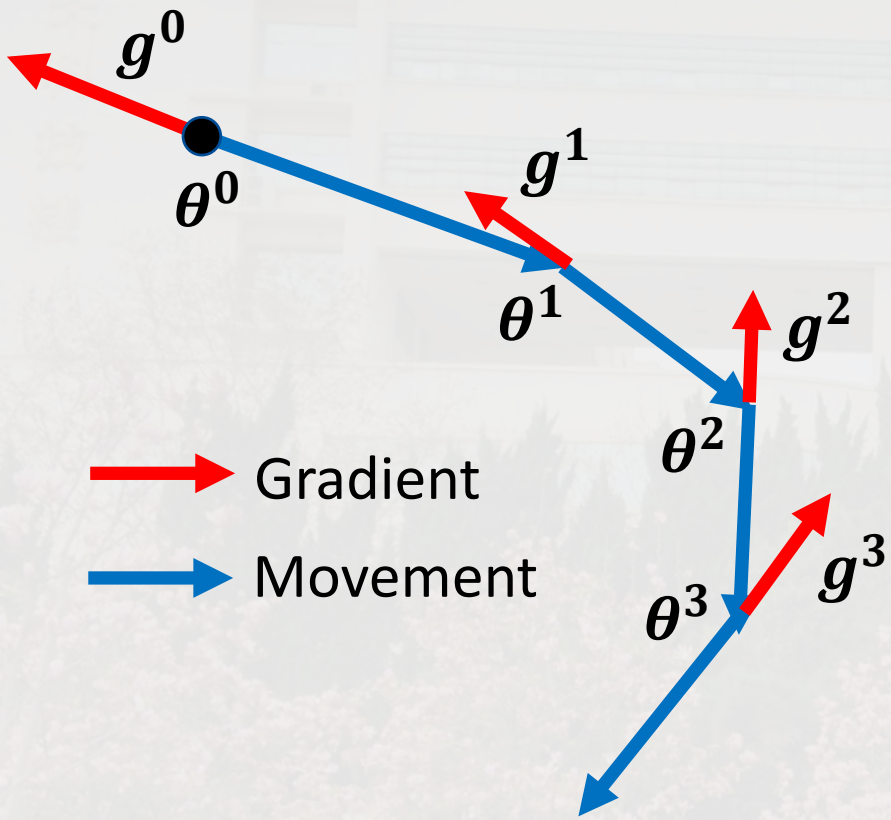
3 深度学习优化器

- Momentum （动量）



3 深度学习优化器

❑ 梯度下降过程



Starting at θ^0

Move to $\theta^1 = \theta^0 - \eta g^0$

Compute gradient g^1

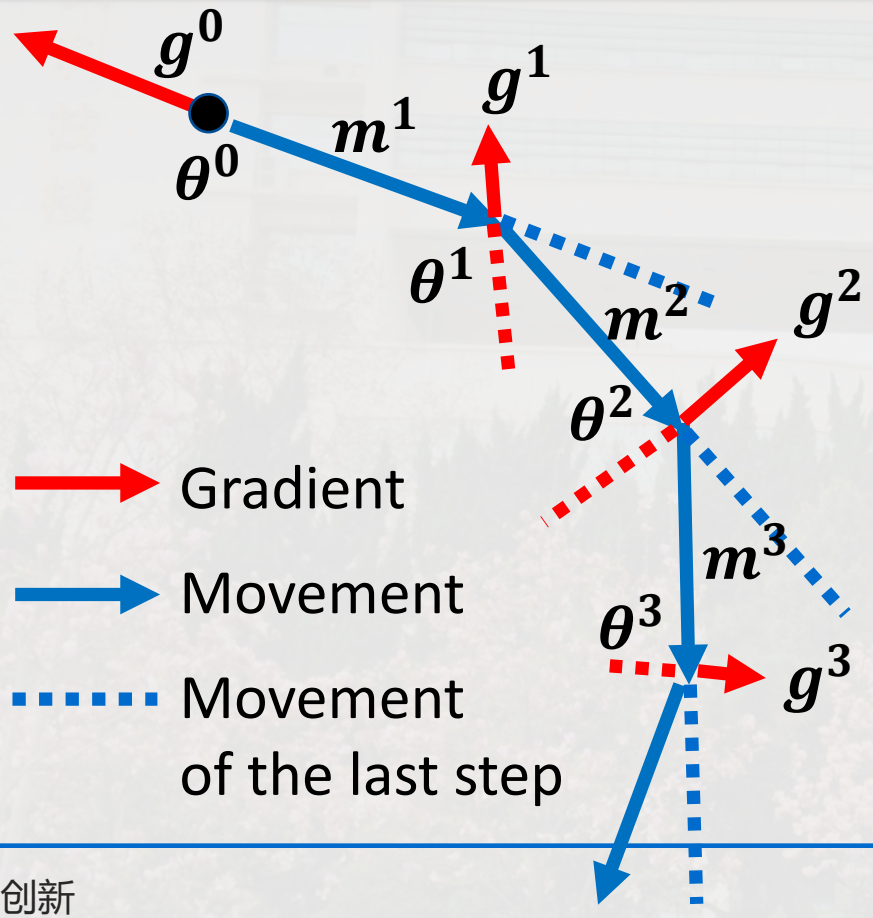
Move to $\theta^2 = \theta^1 - \eta g^1$

⋮

3 深度学习优化器

❑ 梯度下降+Momentum （动量）

Movement: **movement of last step** minus **gradient** at present



Starting at $\boldsymbol{\theta^0}$

Movement $\boldsymbol{m^0 = 0}$

Compute gradient $\boldsymbol{g^0}$

Movement $\boldsymbol{m^1 = \lambda m^0 - \eta g^0}$

Move to $\boldsymbol{\theta^1 = \theta^0 + m^1}$

Compute gradient $\boldsymbol{g^1}$

Movement $\boldsymbol{m^2 = \lambda m^1 - \eta g^1}$

Move to $\boldsymbol{\theta^2 = \theta^1 + m^2}$

Movement not just based on gradient, but previous movement.

3 深度学习优化器

❑ 梯度下降+Momentum （动量）

Movement: **movement of last step** minus **gradient** at present

$\boldsymbol{m^i}$ is the weighted sum of all the previous gradient: $\boldsymbol{g^0}, \boldsymbol{g^1}, \ldots, \boldsymbol{g^{i-1}}$

$\boldsymbol{m^0 = 0}$

$\boldsymbol{m^1} = -\eta \boldsymbol{g^0}$

⋮

Starting at $\boldsymbol{\theta^0}$

Movement $\boldsymbol{m^0 = 0}$

Compute gradient $\boldsymbol{g^0}$

Movement $\boldsymbol{m^1} = \lambda \boldsymbol{m^0} - \eta \boldsymbol{g^0}$

Move to $\boldsymbol{\theta^1 = \theta^0 + m^1}$

Compute gradient $\boldsymbol{g^1}$

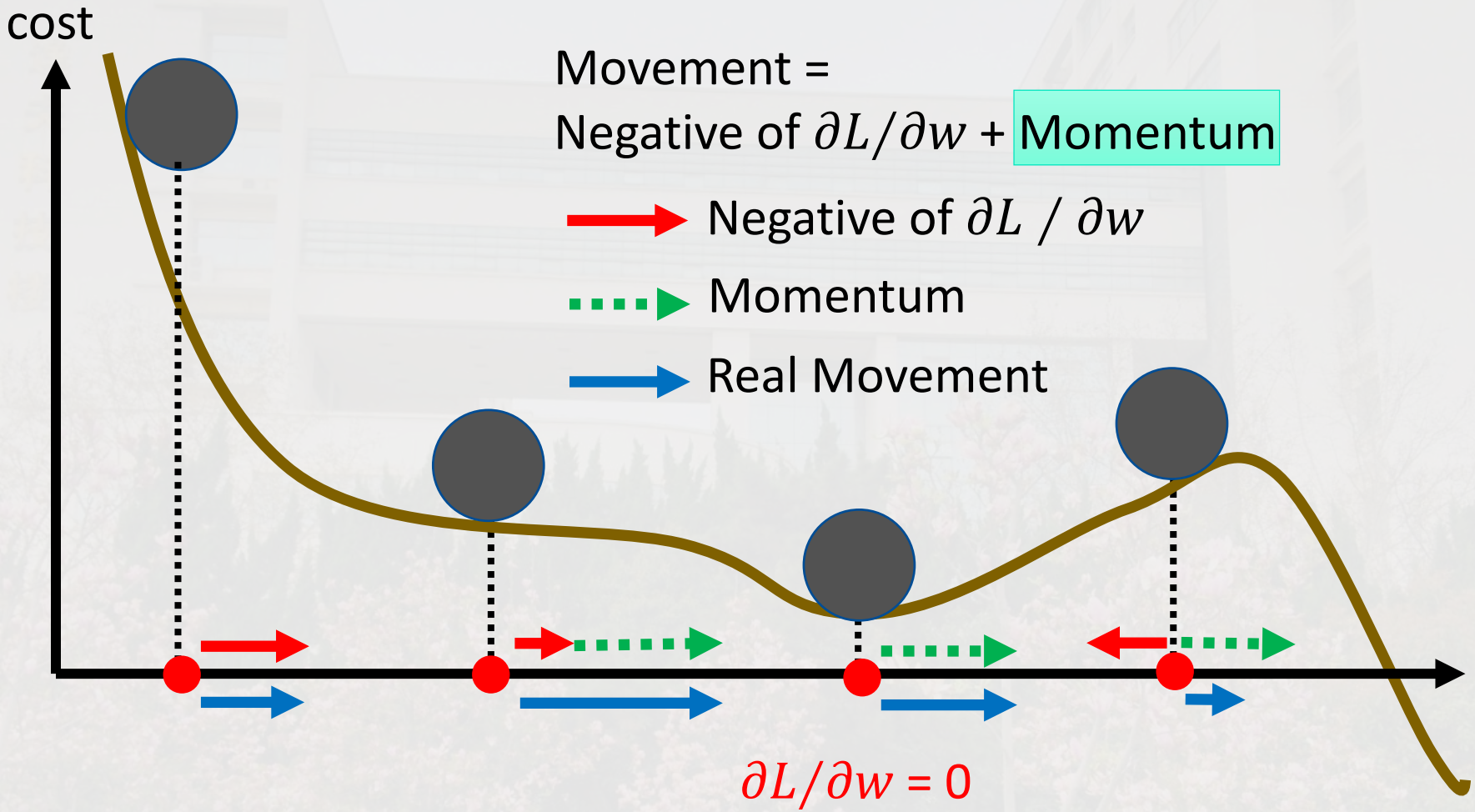
Movement $\boldsymbol{m^2} = \lambda \boldsymbol{m^1} - \eta \boldsymbol{g^1}$

Move to $\boldsymbol{\theta^2 = \theta^1 + m^2}$

Movement not just based on gradient, but previous movement.

3 深度学习优化器

- 动量随机梯度下降（SGDM）



3 深度学习优化器

- Adam

  - Adam=RMSProp (Advanced Adagrad) + Momentum

    - ✓ 可以独立的调整每一个参数的学习率
    - ✓ 结合梯度与动量来更新参数

动量:

$$m^i = \beta_1 m^{i-1} + (1-\beta_1) g^i, \quad \bar{m}^i = m^i / (1-(\beta_1)^i$$

RMSProp：

$$v^i = \beta_2 v^{i-1} + (1-\beta_2)\left(g^i\right)^2, \bar{v}^i = v^i / (1-\beta_2)$$

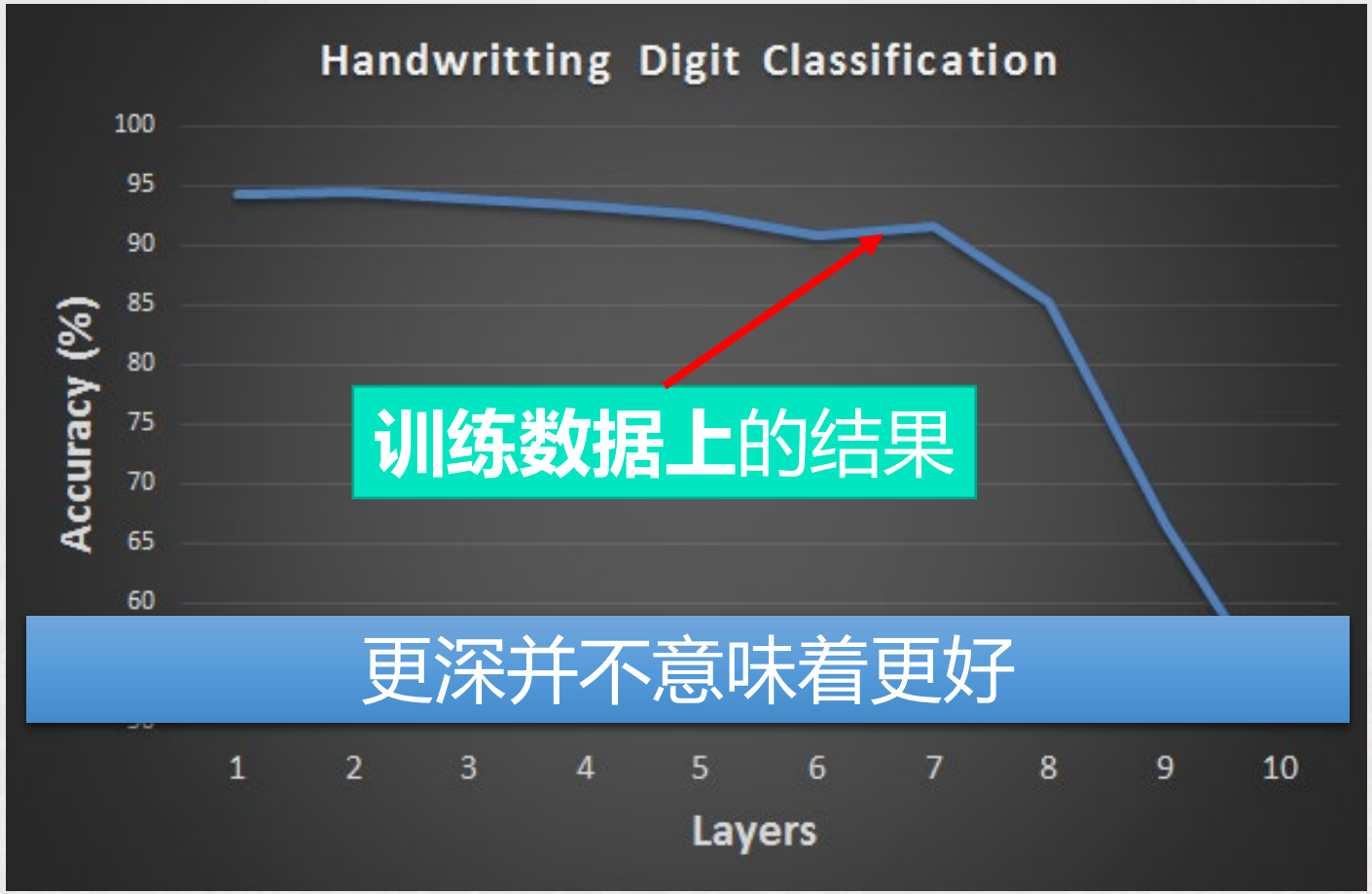
$$\boldsymbol{\theta^i = \theta^{i-1} - \alpha \bar{m}^i / (\sqrt{\bar{v}^i} + \varepsilon)}$$

  - Adam是目前常用的参数更新方法

```
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
```

4 梯度消失

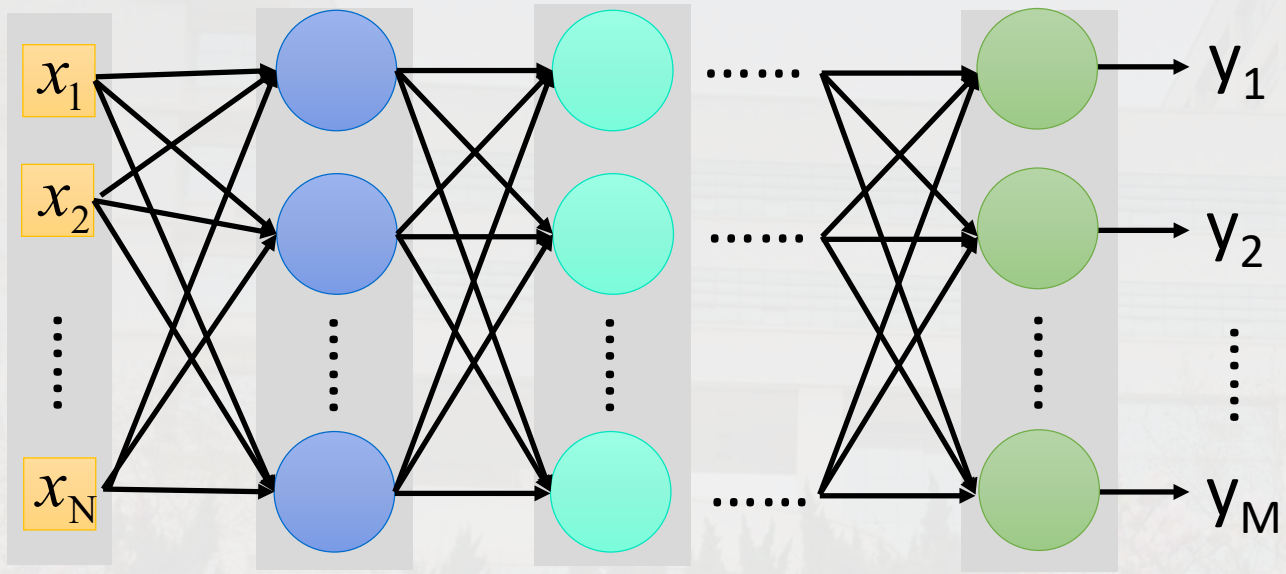
- 深度神经网络的性能

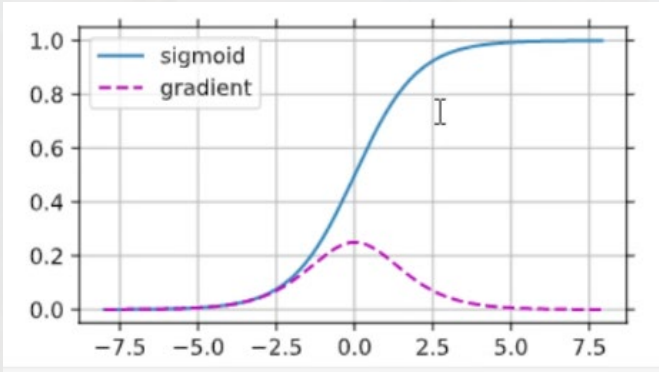


- 1980年代，激活函数一般使用sigmoid函数

4 梯度消失

➢对一个层数很深的网络





Sigmoid函数与梯度值

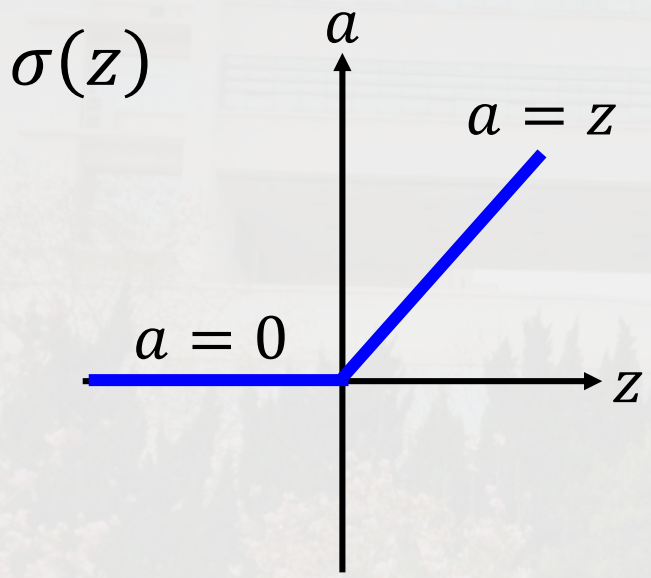
❑前面权重对应的梯度小，更新慢，后面权重对应的梯度大，更新快

❑前面的参数还是随机取值时，后面的权重可能已经收敛到某局部极值

➢产生这种梯度消失现象与选用的激活函数有关

4 梯度消失

➢选用其他激活函数： 修正线性单元 （Rectified Linear Unit, ReLU）



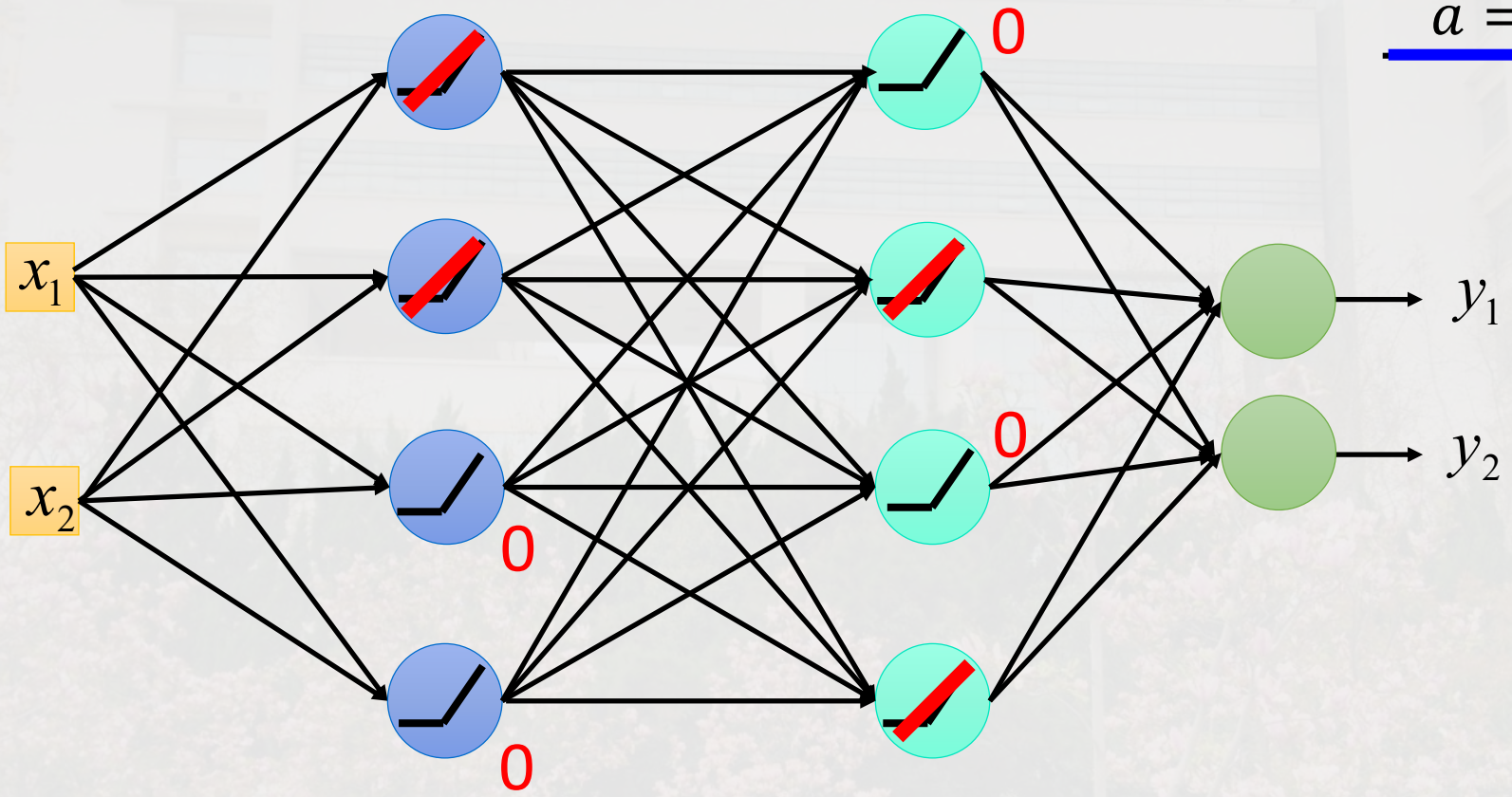
[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

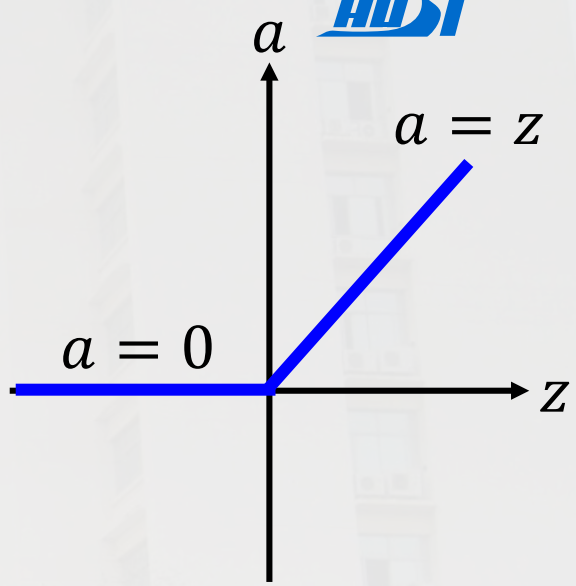
选用的原因:

1. 计算速度很快
2. 仿生物学原理
3. 可看成无数个有偏置的sigmoid函数的累加
4. 应对梯度消失问题

4 梯度消失

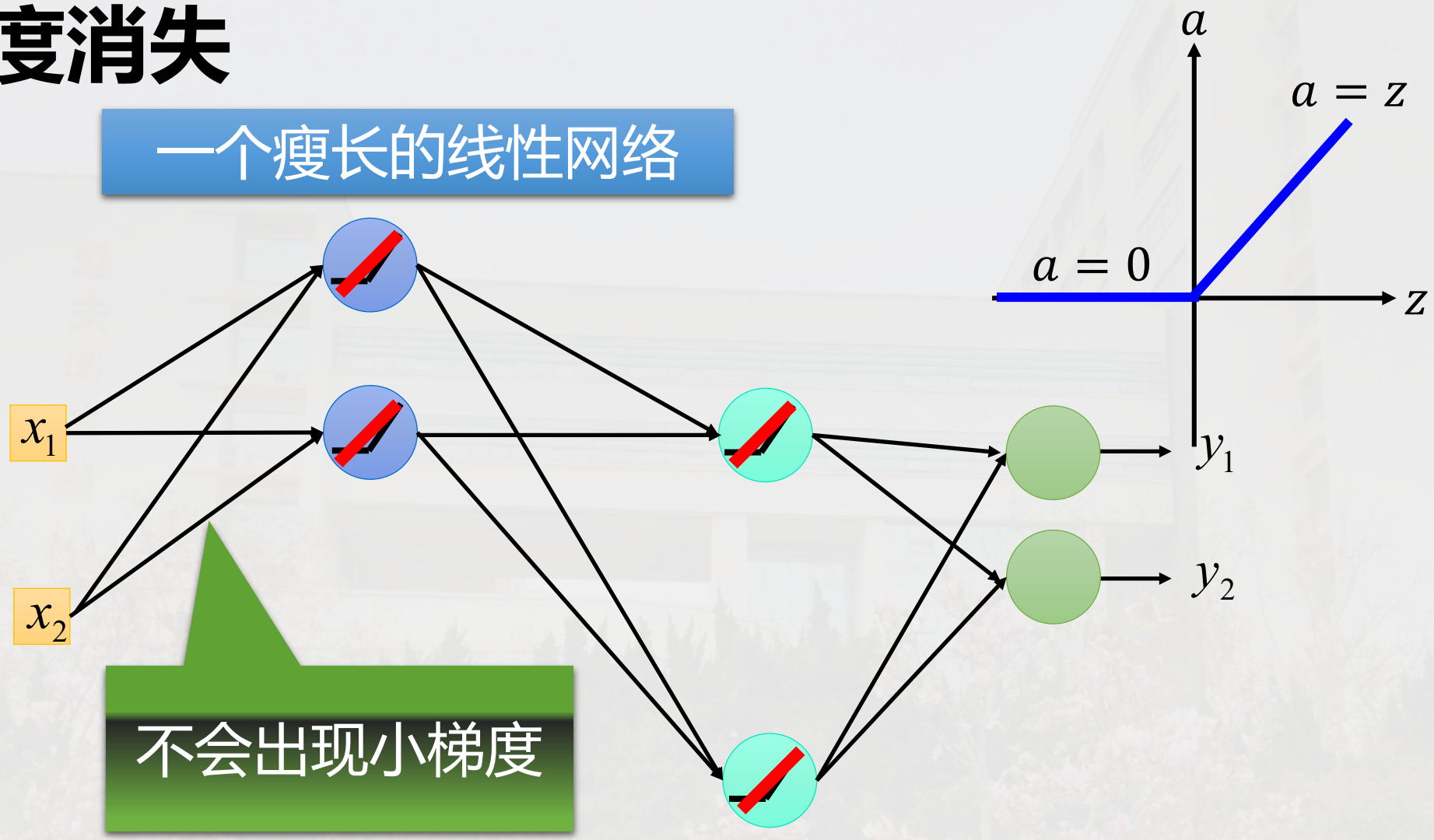
- 使用ReLU函数作为激活函数的神经网络
  - 神经元的输出要么等于输入的加权和，要么等于0





4 梯度消失

一个瘦长的线性网络

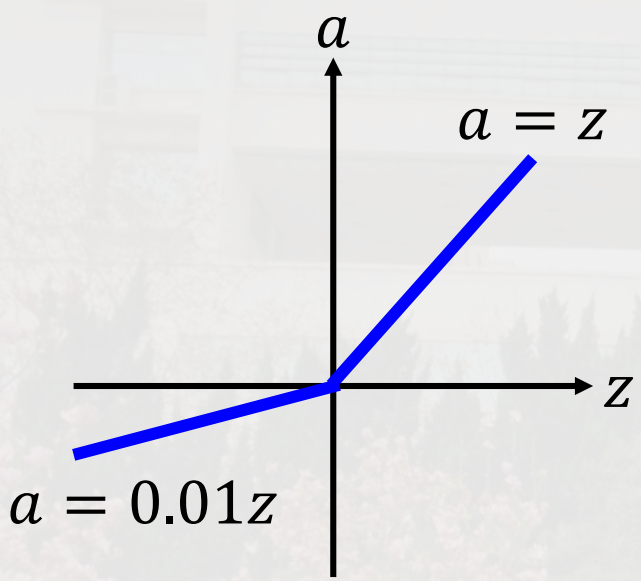


不会出现小梯度

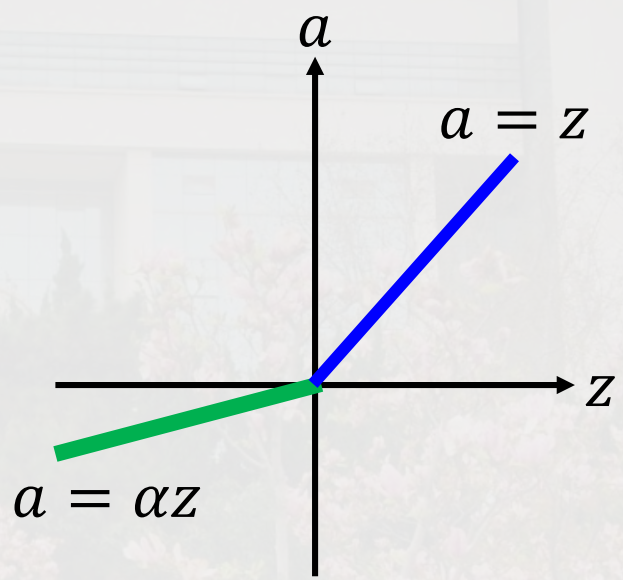
4 梯度消失

- 其他ReLU函数：

Leaky ReLU



Parametric ReLU

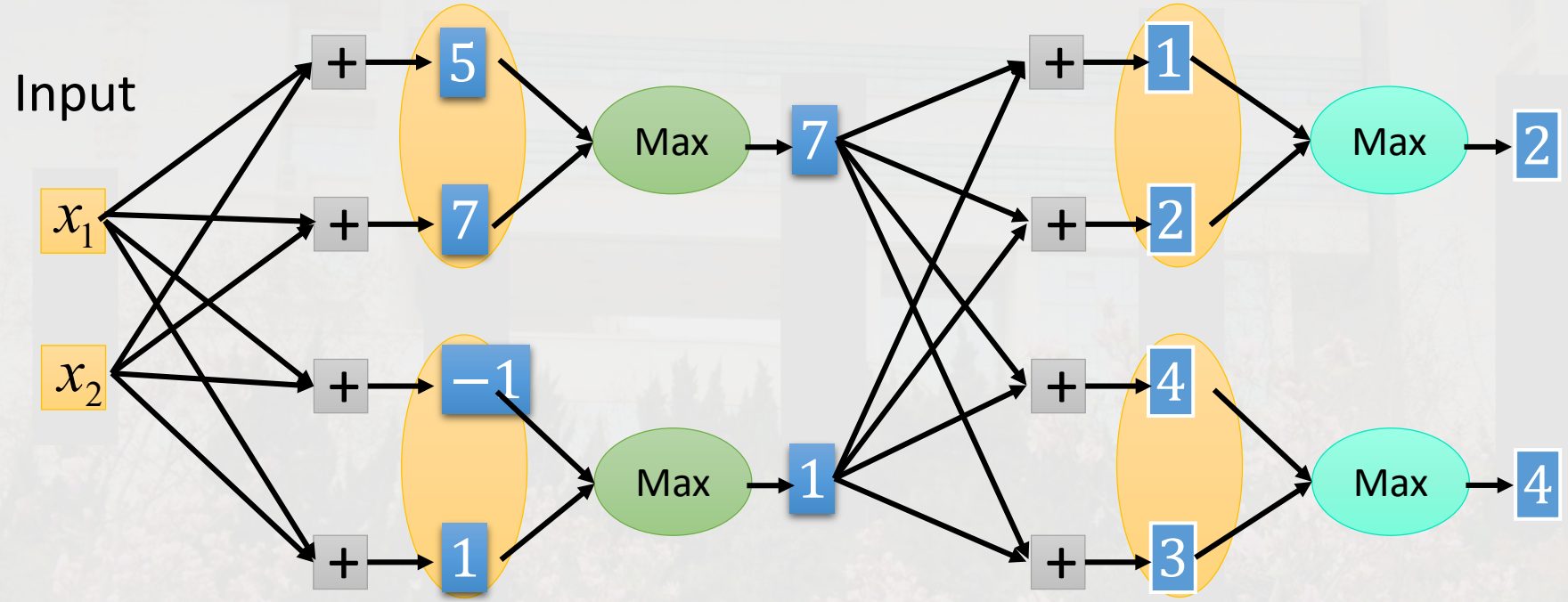


α 也是由梯度下降法学习得到

4 梯度消失

- Maxout：网络自己学习激活函数 [Ian J. Goodfellow, ICML'13]

ReLU 是Maxout的一个特例

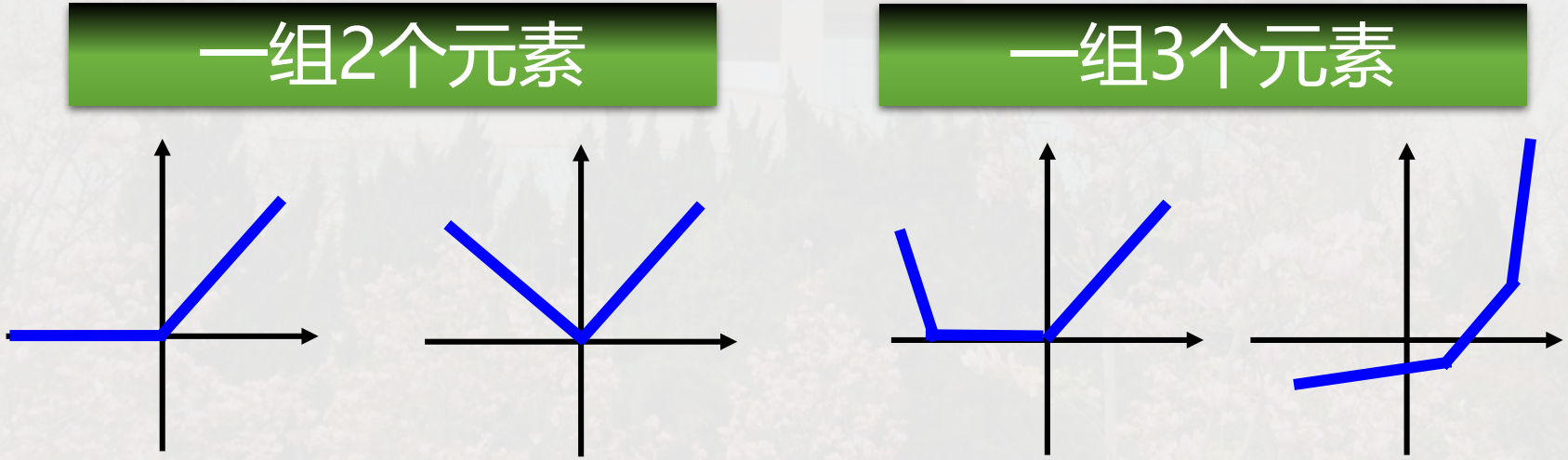


分组选最大值输出

一个组里面可以有更多的元素

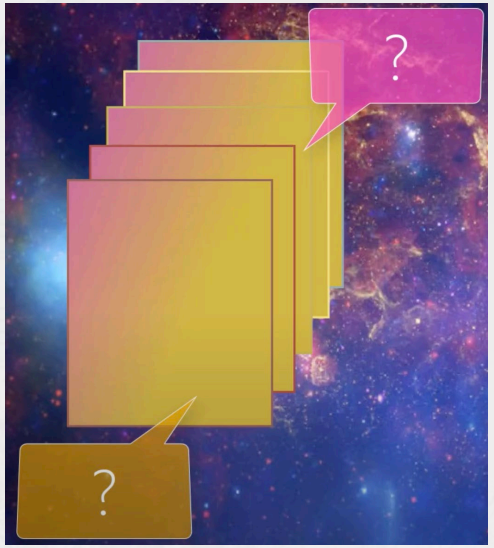
4 梯度消失

- Maxout：网络自己学习激活函数 [Ian J. Goodfellow, ICML'13]
  - Maxout网络里激活函数可能是任意的分段线性函数
  - 分段数取决于组里元素的数量



4 模型微调

- 深度神经网络里有很多的超参数需要设置，如
  - 隐含层层数与每层神经元数
  - 采用什么样的隐含层
  - 激活函数
  - 学习率
  - 梯度下降的迭代次数
  - 正则化参数
  - ……

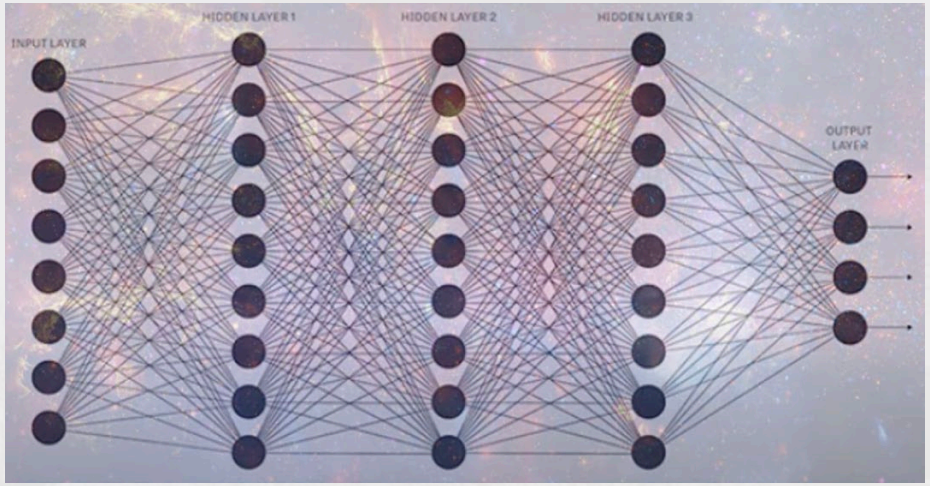


- 若从头搭建深度神经网络模型很可能会是一个艰巨的任务

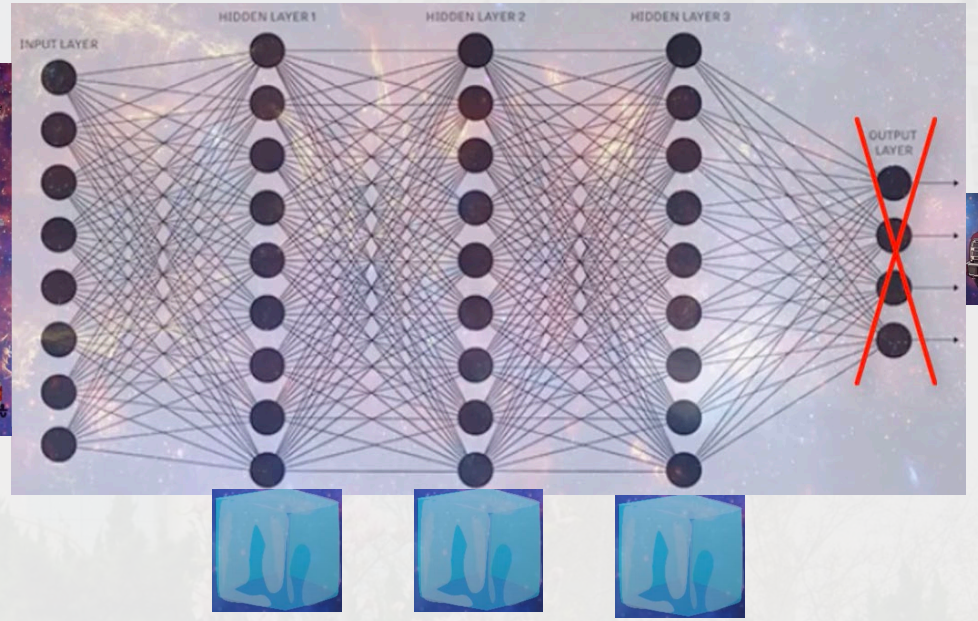
- 存在其他的模型用于处理相似的问题

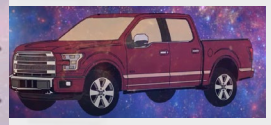
4 模型微调

- 例：在汽车识别模型上进行微调用于识别卡车



汽车识别模型





卡车识别模型

- 前面的layer的通常学习的是通用的特征，后面的layer学的是特殊的特征
  - ✓ 可采取固定前面的layer，只改变后面的输出层，再用数据训练输出层

4 模型微调

➢ 什么是预训练模型

- **预训练模型**是已经用数据集训练好了的模型，比如VGG16/19，Resnet等模型，这些通常是用大型数据集，如Imagenet, COCO等训练好模型参数
- 正常情况下，我们常用的VGG16/19等网络已经是他人调试好的优秀网络，我们无需再修改其网络结构。

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159

4 模型微调

➢ 为什么要做微调?

普通预训练模型的特点是:

用了大型数据集做训练，已经具备了提取浅层基础特征和深层抽象特征的能力。

不做微调:

（1）从头开始训练，需要大量的数据，计算时间和计算资源。

（2）存在模型不收敛，参数不够优化，准确率低，模型泛化能力低，容易过拟合等风险。

使用微调：有效避免了上述可能存在的问题。

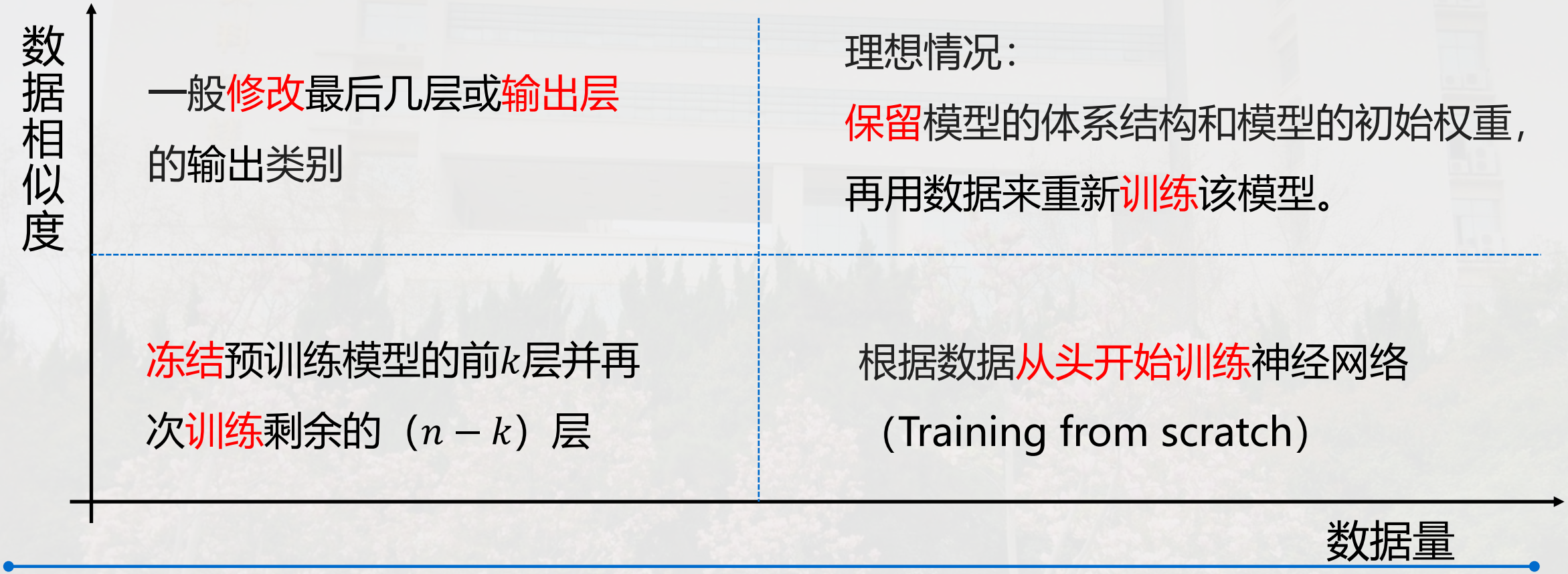
4 模型微调

➢ 什么情况下使用微调?

- ❑ 要使用的数据集和预训练模型的**数据集相似**（若不相似则效果不好，因为源模型（如自然风景识别）和目标模型（如人脸识别）特征提取不同，相应的参数也不同）
- ❑ 自己搭建的模型正确率太低
- ❑ 数据集相似，但数据集数量太少（当目标数据集远小于源数据集时，**微调**有助于提升模型的泛化能力）
- ❑ 计算资源太少。

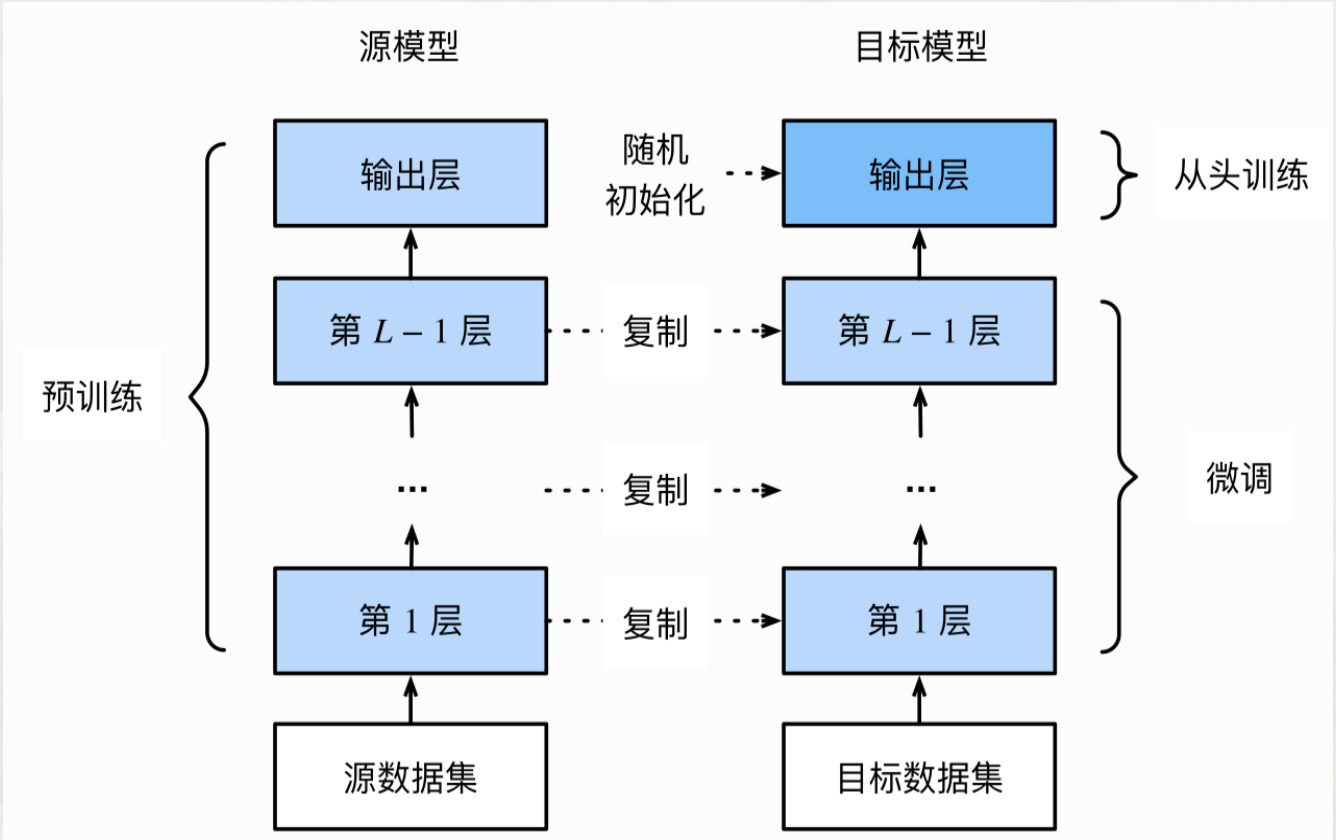
4 模型微调

- 不同数据集下使用微调的方法



模型微调的步骤

- 在源数据集（如ImageNet数据集）上预训练一个神经网络模型，即源模型
- 创建一个新的神经网络模型，即目标模型。它复制了源模型上除输出层外的所有模型设计及其参数。
- 为目标模型添加一个输出大小为目标数据集类别个数的输出层，并随机初始化该层的模型参数。
- 在目标数据集上训练目标模型，从头训练输出层，而其余层的参数都是基于源模型的参数微调得到的。



内容小结

- 深度学习中过拟合现象及正则化方法

  数据预处理 、L1与L2正则化（参数范数惩罚）、早停、Dropout、批标准化

- 梯度下降法及其优化方法

  SGD、BGD、MBGD、AdaGrad、RMSprop、Momentum、Adam

- 梯度消失与激活函数

  ReLU函数、Leaky ReLU、Parametric ReLU、Maxout

- 模型微调