《人工智能数学原理与算法》
第 3 章：神经网络基础

3.3 深度神经网络

连德富
liandefu@ustc.edu.cn

目录

目录

深度模型训练的困境

- 口在卷积神经网络上不断加深网络，训练集的错误率会上升

不是过拟合导致，而是欠拟合

- 口这种现象背后的问题是优化问题，越深的模型越难训练

梯度消失 Gradient Vanishing

□ 回忆前馈神经网络反向传播梯度公式：

输入	第 1 层	第 2 层	第 L-1 层	第 L 层
$\boldsymbol{x}=\boldsymbol{a}^{(0)}$	$\to \boldsymbol{z}^{(1)} \to \boldsymbol{a}^{(1)}$	$\to \boldsymbol{z}^{(2)} \to \boldsymbol{a}^{(2)}$	$\to \cdots \to \boldsymbol{a}^{(L-1)}$	$\to \boldsymbol{z}^{(L)} \to \boldsymbol{a}^{(L)}$

$$\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{e}^{(l)}\left(\boldsymbol{a}^{(l-1)}\right)^{\mathsf{T}} \qquad \boldsymbol{e}^{(l)} = \nabla f\left(\boldsymbol{z}^{(l)}\right)\odot\left(\boldsymbol{e}^{(l+1)}\boldsymbol{W}^{(l+1)}\right)$$

梯度消失：神经网络层数过多时，由于链式法则导致反向传播过程中浅层参数的导数趋近于0，无法有效更新和学习。

$$\left\|\boldsymbol{e}^{(l)}\right\|_2 \le \left\|\nabla f\left(\boldsymbol{z}^{(l)}\right)\right\|_2 \cdot \left\|\boldsymbol{e}^{(l+1)}\boldsymbol{W}^{(l+1)}\right\|_2$$

$$\le \left\|\nabla f\left(\boldsymbol{z}^{(l)}\right)\right\|_2 \cdot \left\|\boldsymbol{W}^{(l+1)}\right\|_2 \cdot \left\|\boldsymbol{e}^{(l+1)}\right\|_2$$

例：假设每一层神经元对上一层输出的偏导数乘上权重的绝对值都小于1（假设为0.9），那么经过30层传播后$0.9^{30}=0.0424\approx 0$.

$$\left\|\boldsymbol{e}^{(l)}\right\|_2 \le \prod_{i=l}^{L-1}\left(\left\|\nabla f\left(\boldsymbol{z}^{(i)}\right)\right\|_2 \cdot \left\|\boldsymbol{W}^{(i+1)}\right\|_2\right)\cdot\left\|\boldsymbol{e}^{(L)}\right\|_2$$

$$= \prod_{i=l}^{L-1}\underbrace{\left(\left\|\nabla f\left(\boldsymbol{z}^{(i)}\right)\right\|_2 \cdot \left\|\boldsymbol{W}^{(i+1)}\right\|_2\right)}_{\text{当这一项小于1时，多层连乘趋近于0}}\cdot\left\|\frac{\partial \mathcal{L}(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \boldsymbol{z}^{(L)}}\right\|_2$$

梯度消失 Gradient Vanishing

- **梯度消失：神经网络层数过多时，由于链式法则导致反向传播过程中浅层参数的导数趋近于0，无法有效更新和学习。**
  - 例：假设每一层神经元对上一层输出的偏导数乘上权重的绝对值都小于1（假设为0.9），那么经过30层传播后$0.9^{30} = 0.0424 \approx 0$.

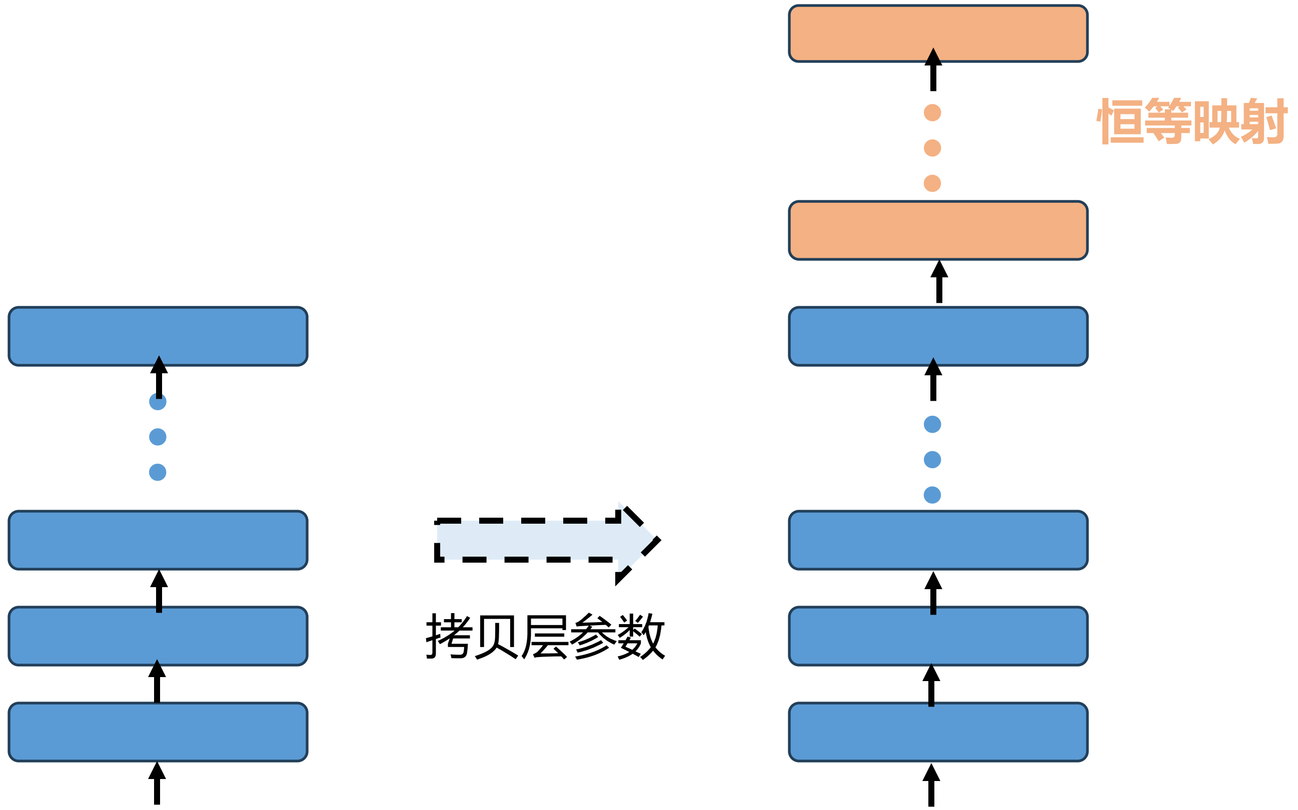
- **缓解梯度消失**
  - 更换激活函数：sigmoid 和 tanh 激活函数的导数都在[0,1]区间内，更换激活函数为 ReLU 函数 或 Leaky ReLU 函数可以有效缓解梯度消失。
  - **使用 残差连接 (Residual Connection): ResNet (He et al., 2016)。**
  - 使用归一化 (Normalization) 策略，随后讨论。
  - 使用合适的 初始化 策略，随后讨论。

目录

残差连接的动机

□直观上深层模型不会比浅层模型更差，因为可以通过如下方式构造出这样的深层模型

- 将浅层模型的所有层拷贝到深层模型的低层，其余的高层都设置为恒等映射
- 这种构造下的深层模型和浅层模型性能相当
- 为模型结构添加“直接连接”来鼓励恒等映射的学习

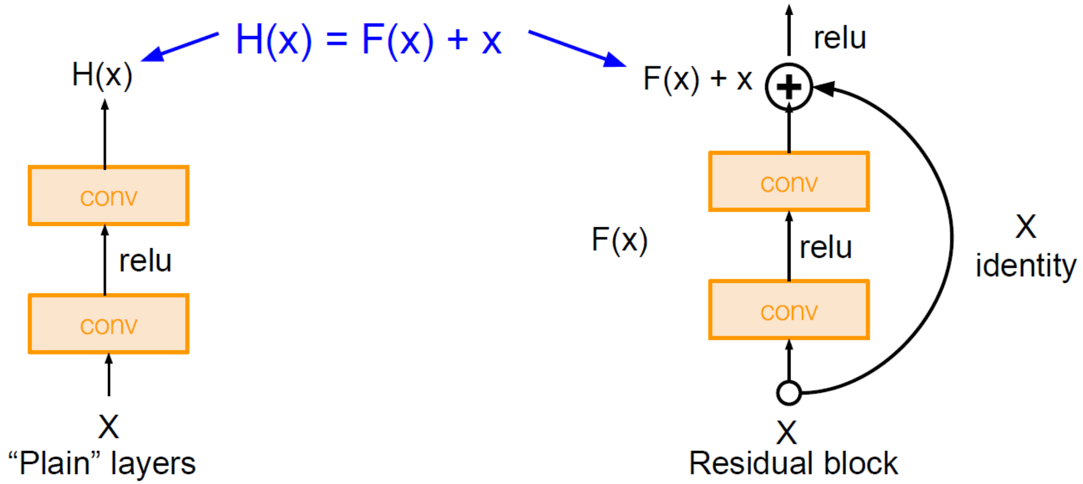


典型的卷积网络—ResNet

口残差网络（Residual Network，ResNet）是通过给非线性的卷积层增加直连边的方式来提高信息的传播效率。

- 假设在一个深度网络中，我们期望一个非线性单元（可以为一层或多层的卷积层）$F(x,\theta)$去逼近一个目标函数为$h(x)$。
- 将目标函数拆分成两部分：恒等函数和残差函数

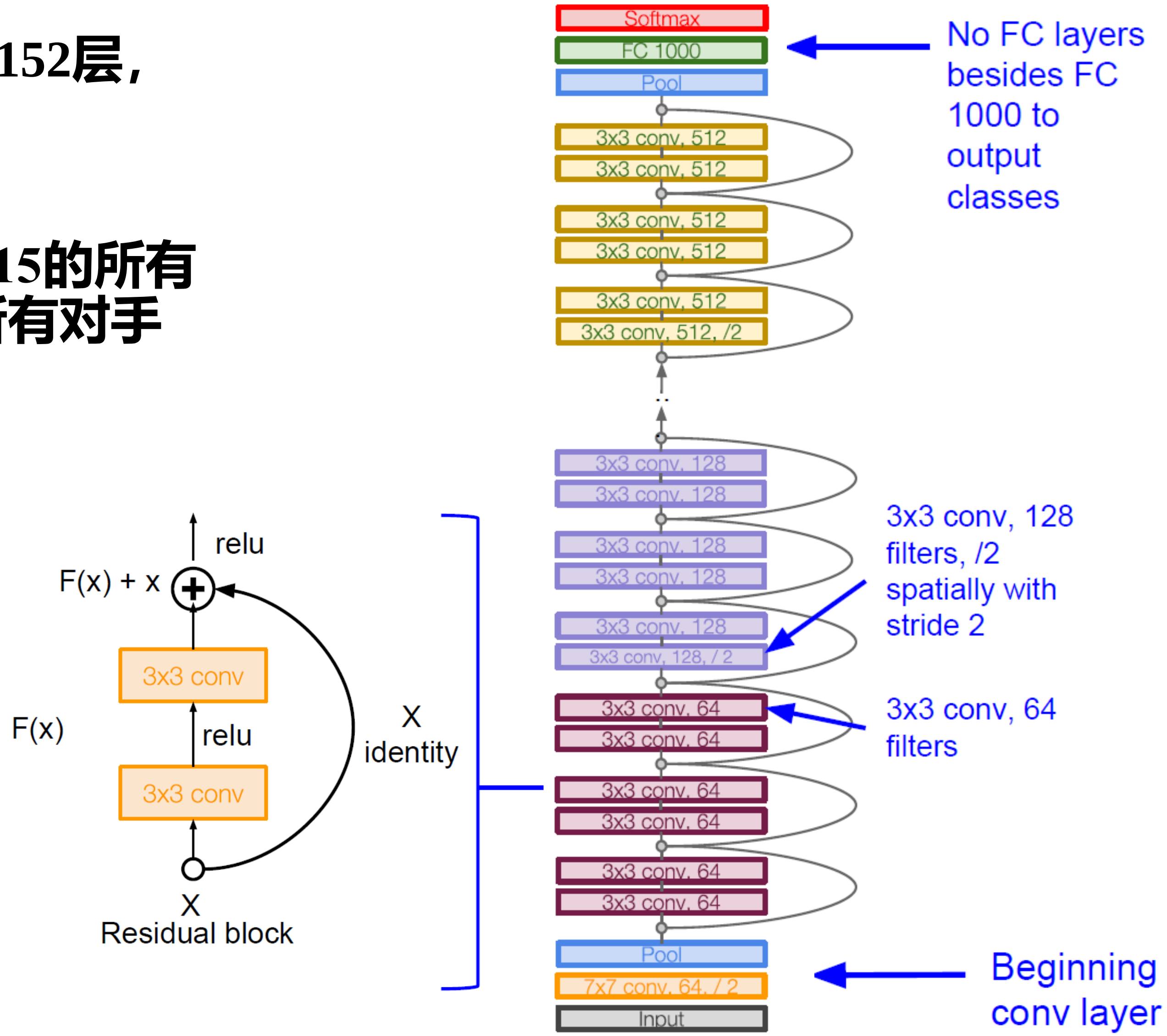
$$h(\boldsymbol{x}) = \underbrace{\boldsymbol{x}}_{\text{恒等函数}} + \underbrace{(h(\boldsymbol{x}) - \boldsymbol{x})}_{\substack{\text{残差函数} \\ \downarrow \\ F(x,\theta)}}$$



典型的卷积网络—ResNet

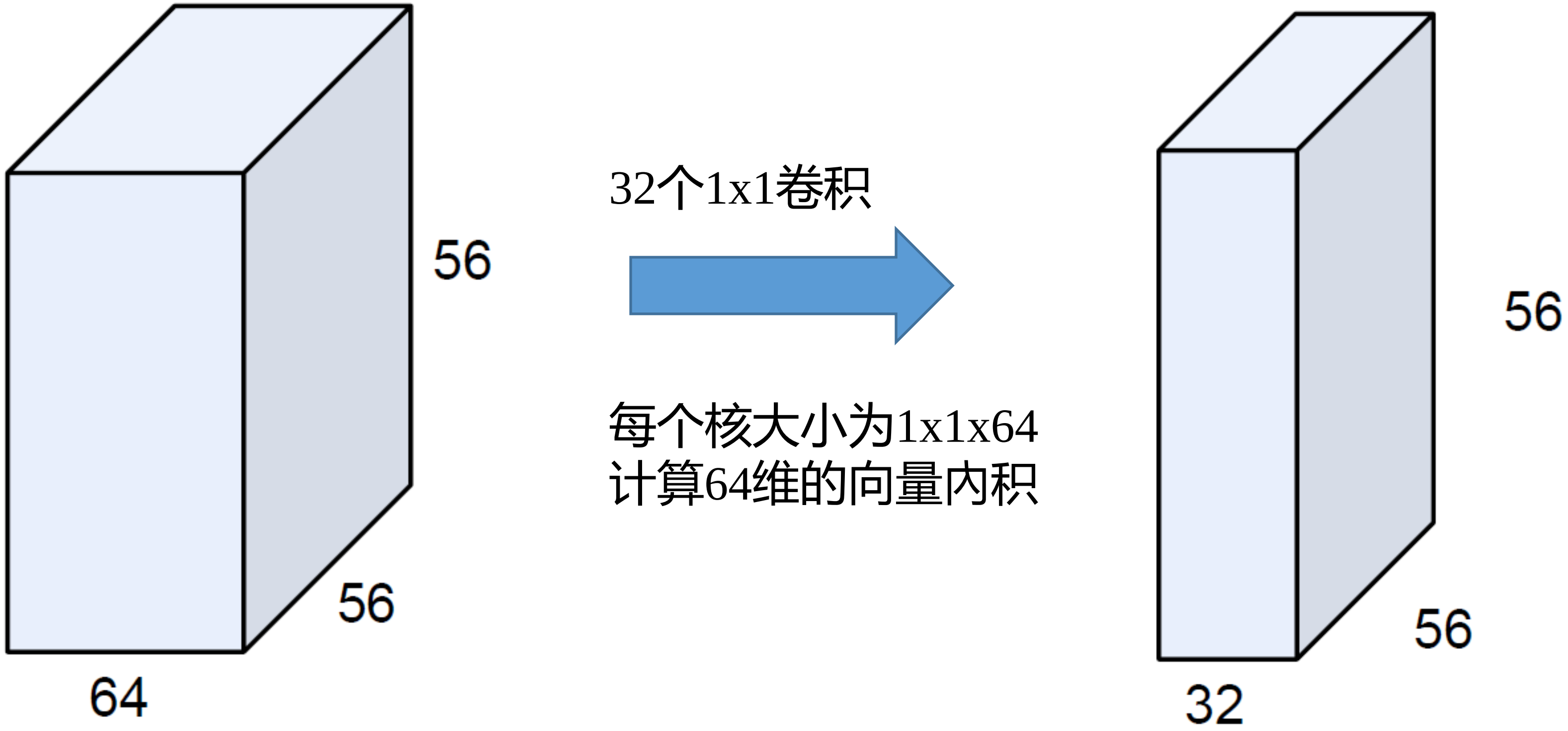
- 2015 ILSVRC winner （152层，3.57% top 5 error）
- 用残差连接的极深网络
- 在ILSVRC’15和COCO’15的所有分类和检测比赛中横扫所有对手

- 残差网络堆砌残差块
- 每个残差块里有两个3x3的卷积层



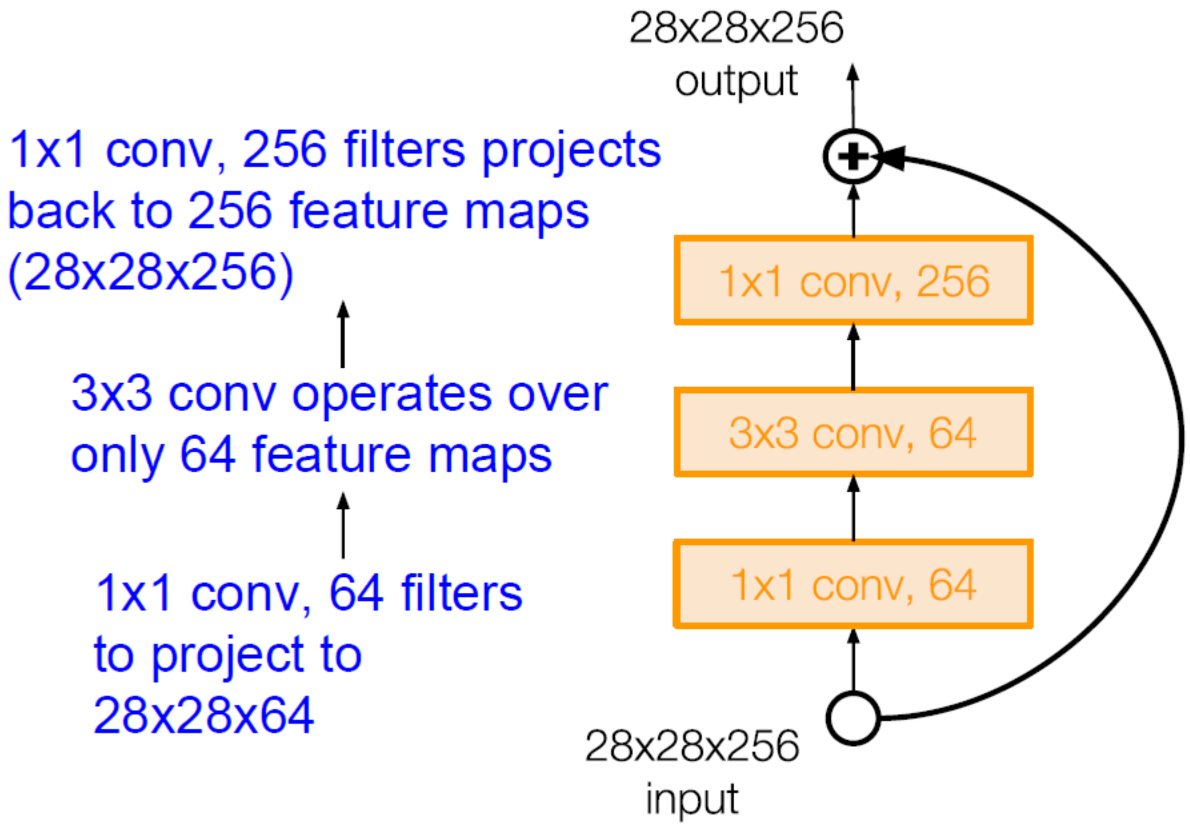
典型的卷积网络—ResNet

□利用“bottleneck”层改进效率



典型的卷积网络—ResNet

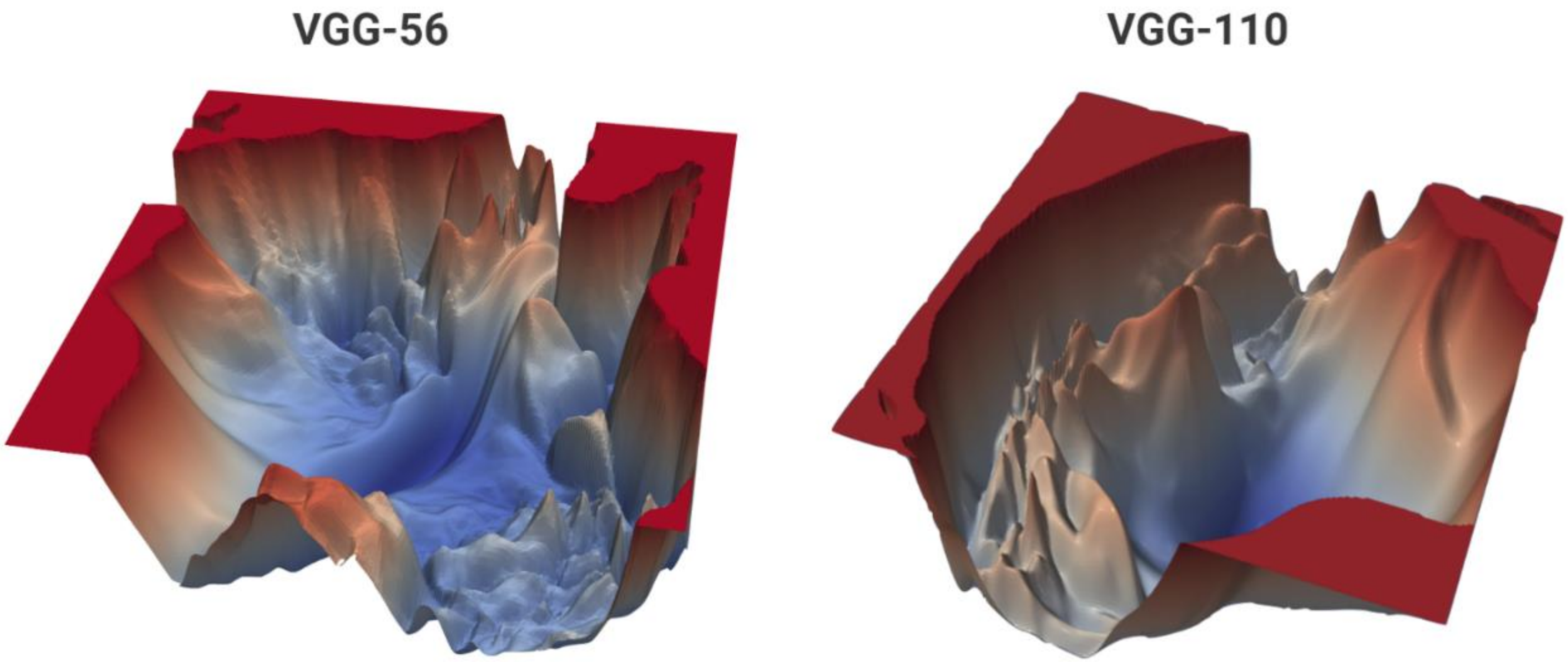
口利用“bottleneck”层改进效率



目录

神经网络优化的挑战

优化地形（Optimization Landscape）：在高维空间中损失函数的曲面形状

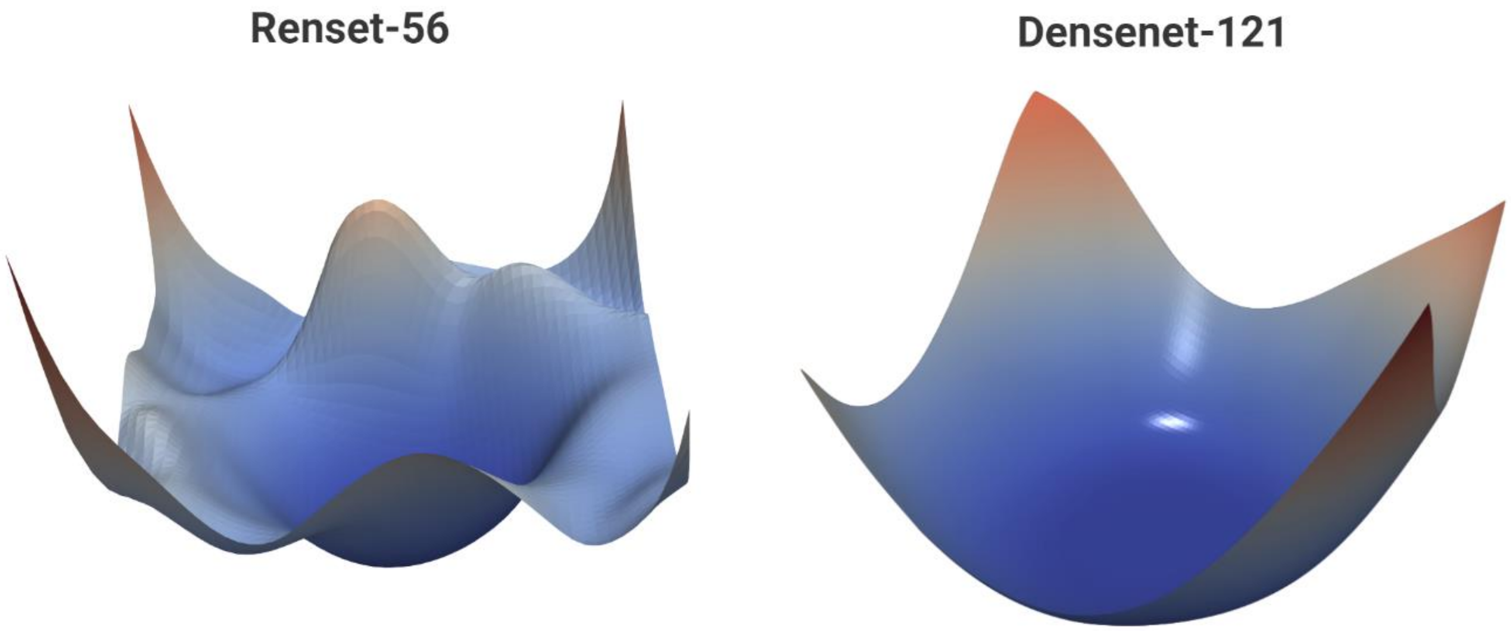


Li H, Xu Z, Taylor G, et al. Visualizing the loss landscape of neural nets[C] //Advances in Neural Information Processing Systems. 2018: 6389-6399.

修改网络结构获得更好的优化地形

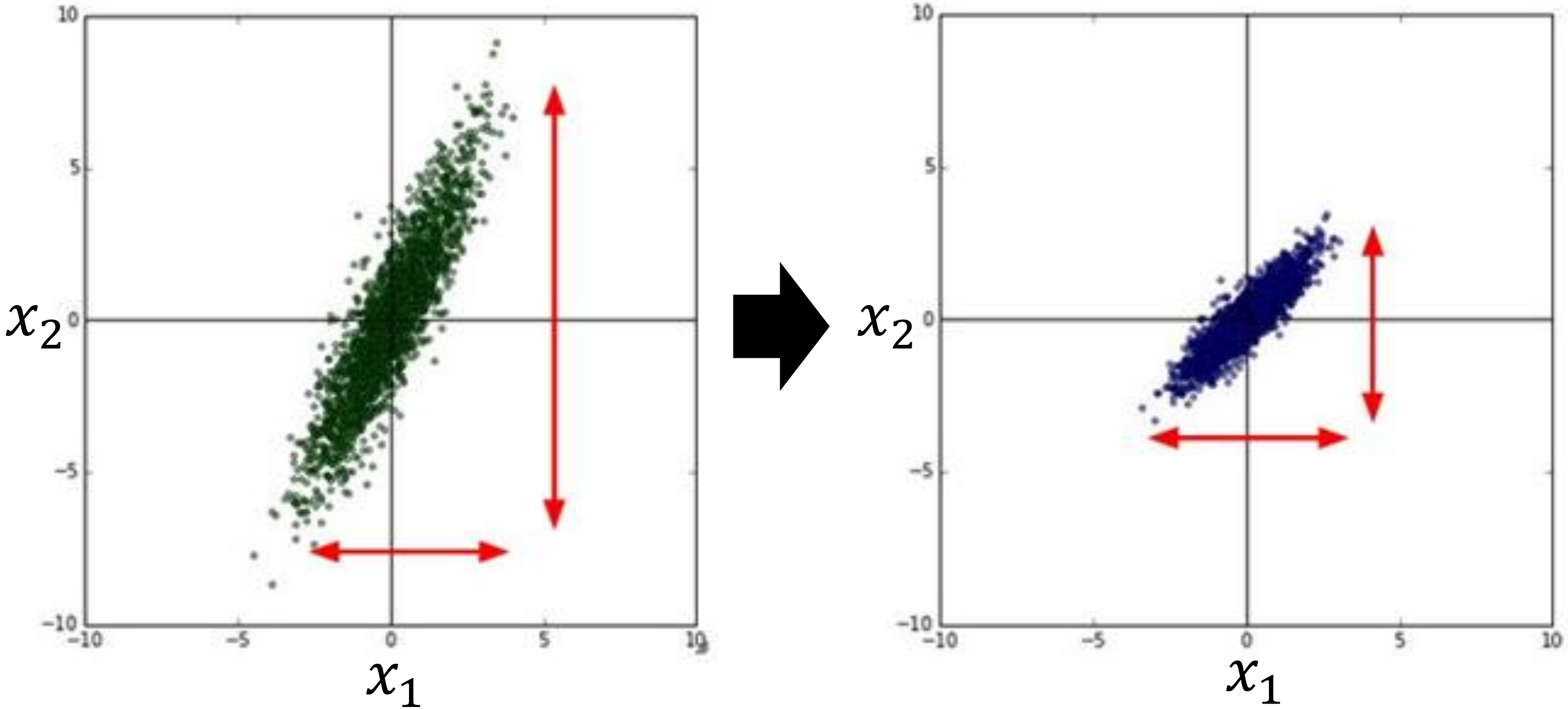
- 好的优化地形通常比较平滑，更容易优化

- 使用 ReLU 激活函数、残差连接、逐层归一化等



数据归一化

$$\hat{y} = b + w_1 x_1 + w_2 x_2$$

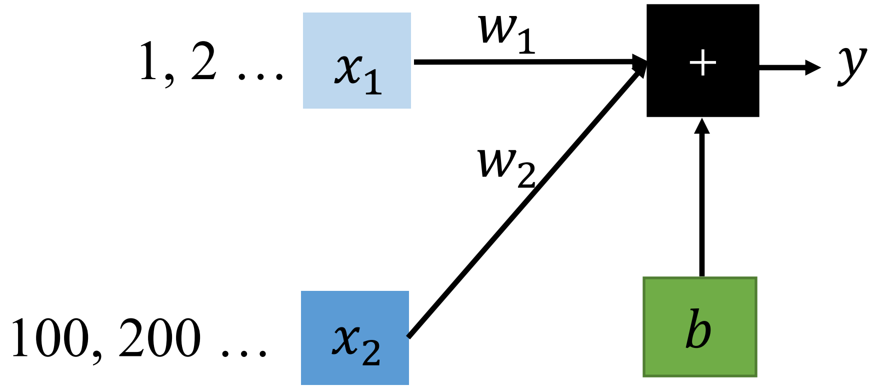


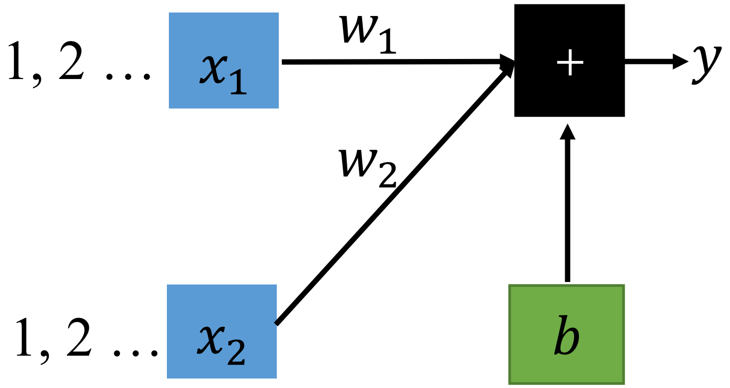
数据归一化：使得不同的特征有相同的尺度

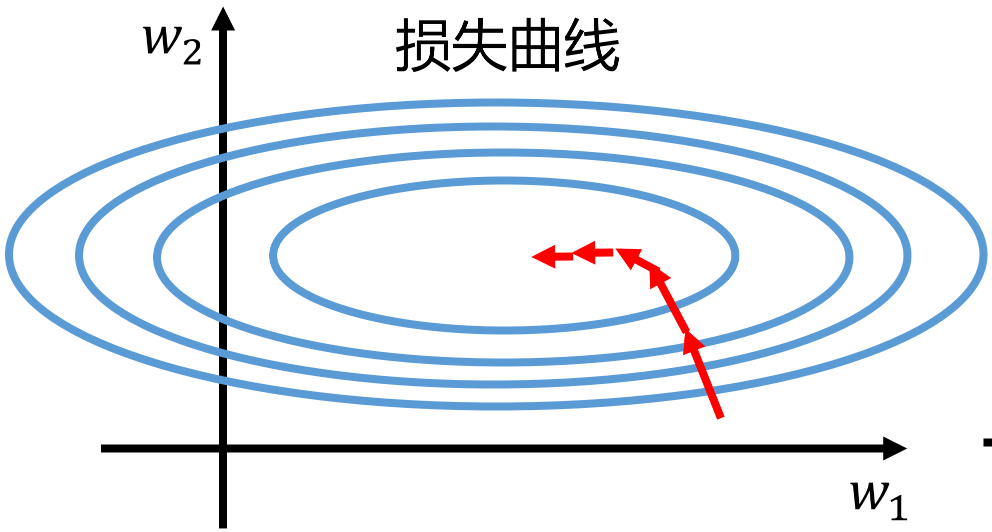
数据归一化

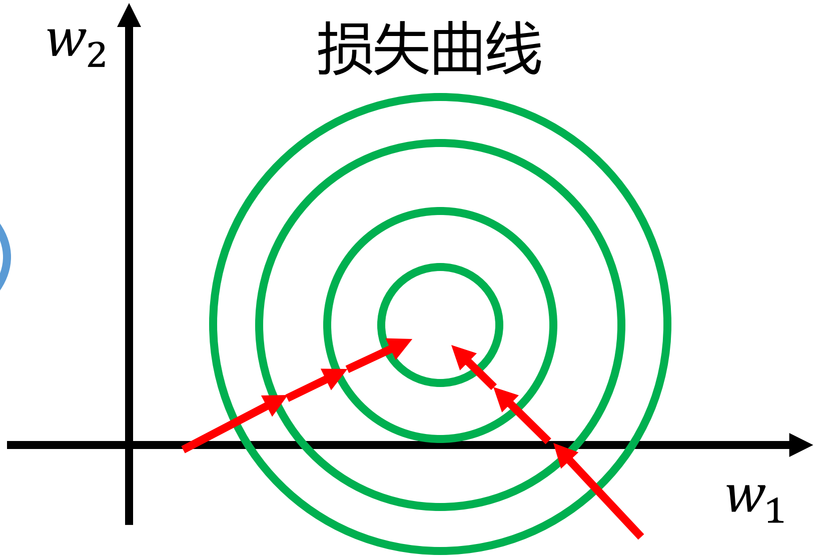
$$\hat{y} = b + w_1 x_1 + w_2 x_2$$

$$\mathcal{L} = (y - \hat{y})^2$$

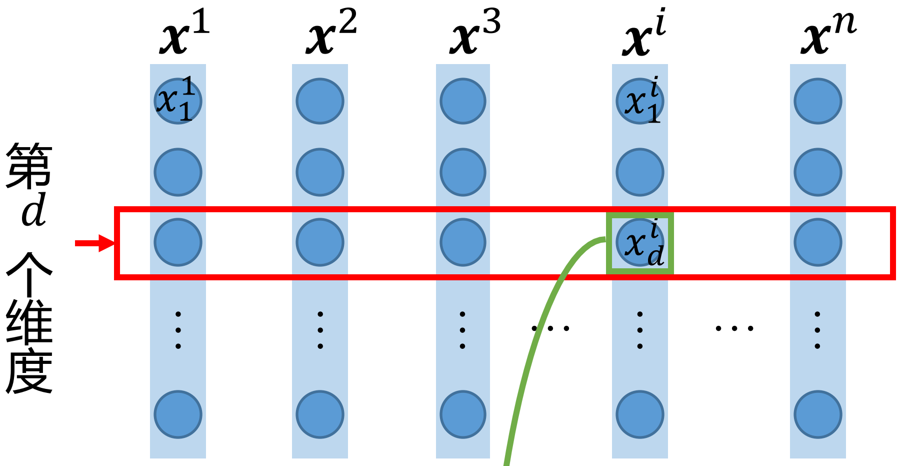








数据归一化—标准化



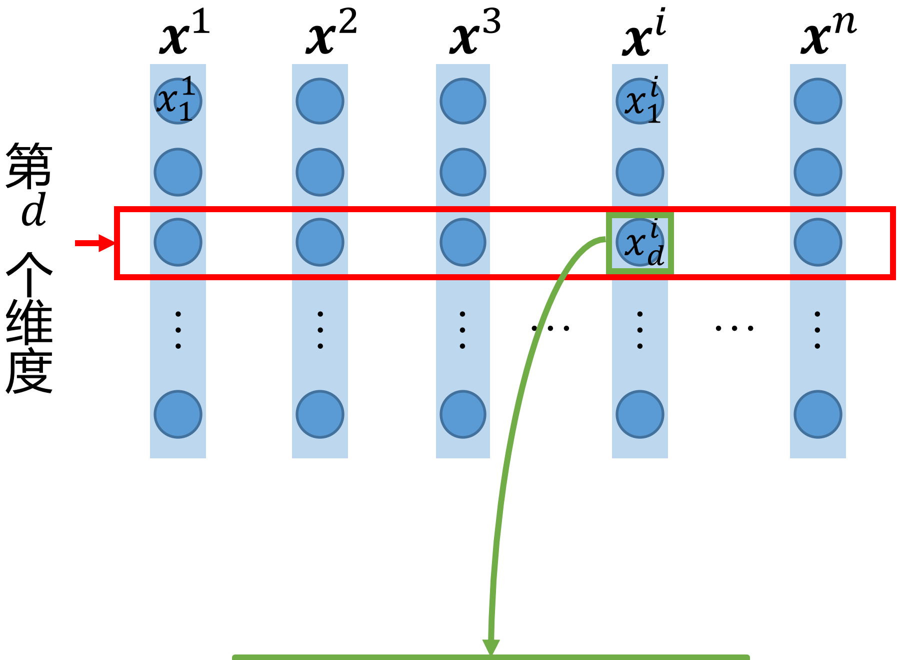
计算均值$m_d = \frac{1}{n}\sum_{i=1}^{n} x_d^i$

计算方差$\sigma_d^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_d^i - m_d\right)^2$

$$x_d^i \leftarrow \frac{x_d^i - m_d}{\sigma_d}$$

归一化后数据的所有维度均为
0均值 1方差

数据归一化—最大最小值归一化



计算最大值 $\mathrm{max_d}$

计算最小值 $\mathrm{min_d}$

$$\boxed{x_d^i \leftarrow \frac{x_d^i - \mathrm{min_d}}{\mathrm{max_d} - \mathrm{min_d}}}$$

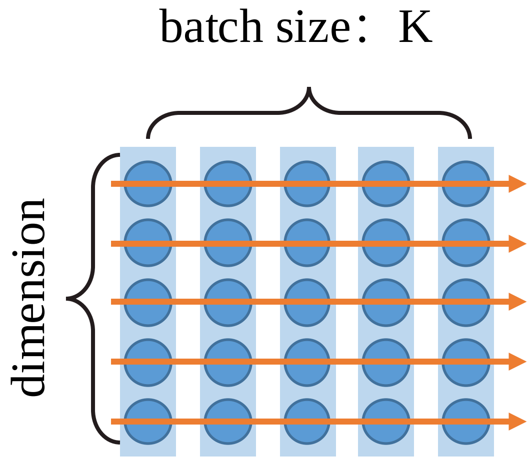
归一化后数据的所有维度均在 $[0,1]$

逐层归一化—Batch Normalization (BN)

- **内部协变量偏移**：每一层的参数在更新过程中，会改变下一层输入的分布，神经网络层数越多，表现得越明显。
  - 逐层归一化：使每一层的神经元的分布在训练过程中保持一致。

神经网络中第l层的净输入为$\boldsymbol{z}^l$，神经元的输出为$\boldsymbol{a}^l$，

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



1. **独立计算该层每个维度的均值和方差**

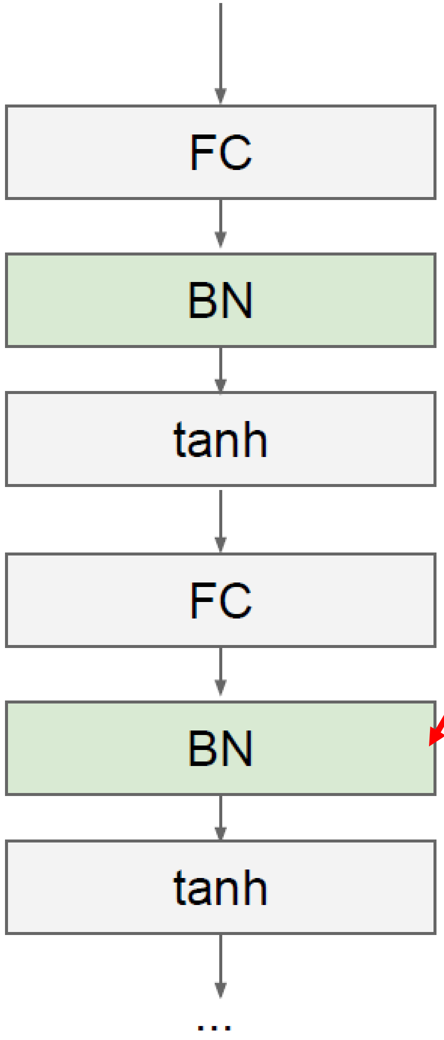
$$\boldsymbol{\mu}_{\mathcal{B}} = \frac{1}{K}\sum_{k=1}^{K}\boldsymbol{z}_k$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{K}\sum_{k=1}^{K}(\boldsymbol{z}_k - \boldsymbol{\mu}_{\mathcal{B}}) \odot (\boldsymbol{z}_k - \boldsymbol{\mu}_{\mathcal{B}})$$

2. 归一化

$$\widetilde{\boldsymbol{z}_k} = \frac{\boldsymbol{z}_k - \boldsymbol{\mu}_{\mathcal{B}}}{\boldsymbol{\sigma}_{\mathcal{B}}}$$

逐层归一化—Batch Normalization (BN)



BN层一般在非线性激活函数和仿射变换之后

- 每层的输入通常是前一层的非线性激活函数的输出
- 由于非线性激活函数，每层输入通常是非高斯的，不适合进行标准化

逐层归一化—Batch Normalization (BN)

- 标准化单元的均值和标准差会降低该单元的表达能力

- 为了使得归一化不对网络的表示能力造成负面影响，引入γ和β两个可学习参数使得各单元的净输出有任意均值和方差

$$\widetilde{z_k} = \frac{z_k - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}} \Rightarrow \widetilde{z_k} = \frac{z_k - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}} \odot \gamma + \beta$$

- 网络可能学习出这两个参数，即重新为该单元的输出添加新的均值和方差。

$$E[\widetilde{z_k}] = \beta; Var[\widetilde{z_k}] = \gamma^2$$

旧参数	新参数
• $\mu_{\mathcal{B}}$和$\sigma_{\mathcal{B}}$取决于低层神经网络的复杂关联	• γ和β解除了与下层计算的密切耦合，直接通过梯度下降来学习

逐层归一化—Batch Normalization (BN)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

BN的作用

- 改善梯度的传播
- 允许比较大的学习率
- 减小初始化的影响

batch size不能太小，否则会导致算法性能下降

逐层归一化—Batch Normalization (BN)

- 测试过程和训练过程有所不同，测试时单个示例输入
- 均值和方差不是在测试集的batch上计算，而是通过追踪训练时整个（训练）数据集上的均值和方差。
- 实践中，可以通过移动平均来计算
- 可以对单一样本评估

逐层归一化—Batch Normalization (BN)

全连接网络的BN

$x: N \times D$

Normalize (N)

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

卷积网络的BN

$x: N \times C \times H \times W$

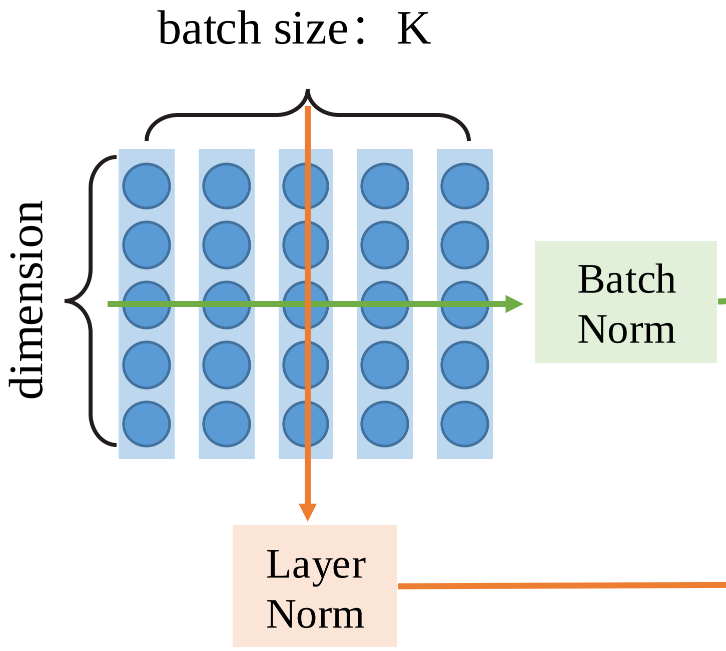
Normalize (N, H, W)

$\mu, \sigma: 1 \times C \times 1 \times 1$

$\gamma, \beta: 1 \times C \times 1 \times 1$

$y = \gamma(x-\mu)/\sigma+\beta$

逐层归一化—Layer Normalization (LN)



$x: N \times D$

Normalize

$\mu, \sigma: 1 \times D$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

全连接网络的BN

$x: N \times D$

Normalize

$\mu, \sigma: N \times 1$

$\gamma, \beta: 1 \times D$

$y = \gamma(x-\mu)/\sigma+\beta$

全连接网络的LN

训练和测试时操作一致
用在循环神经网络等模型中

逐层归一化—Layer Normalization (LN)

- 批归一化 (BN) 在处理序列数据（如文本等），由于不同文本的长度不同，导致处理某些tokens的 BN 时计算估值不合理。

  - Hello , How do you **do ?** **(7 tokens)**
  - Nice to meet you . **<pad> <pad>** **(5 tokens + 2 pad tokens)**
  - I love baseball . **<pad> <pad>** **(5 tokens + 2 pad tokens)**

- 批归一化 (BN) 在Batch Size较小的时候，估算的统计值不合理。

- 层归一化 (LN) 只关注同一层的不同神经元的归一化，因此没有上述问题。

逐层归一化—RMS Normalization

□ Root Mean Square Layer Normalization (Zhang et al., 2019)

- Layer Norm: $\bar{a}_i = \frac{a_i - \mu}{\sigma} g_i + b_i$
- RMS Norm: $\bar{a}_i = \frac{a_i}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}} g_i + b_i$

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

g_i, b_i 是可学习参数

RMS Norm 是 Layer Norm 的改进版本：**Layer Norm 的计算效率低下**。
当前的主流大语言模型（如 LLaMA2, Qwen2 等）都采用 RMS Norm.

逐层归一化—Instance Normalization (IN)

卷积网络的BN

$$x:\ N\times C\times H\times W$$

Normalize

$$\mu,\sigma:\ 1\times C\times 1\times 1$$

$$\gamma,\beta:\ 1\times C\times 1\times 1$$

$$y = \gamma(x-\mu)/\sigma+\beta$$

卷积网络的IN

$$x:\ N\times C\times H\times W$$

Normalize

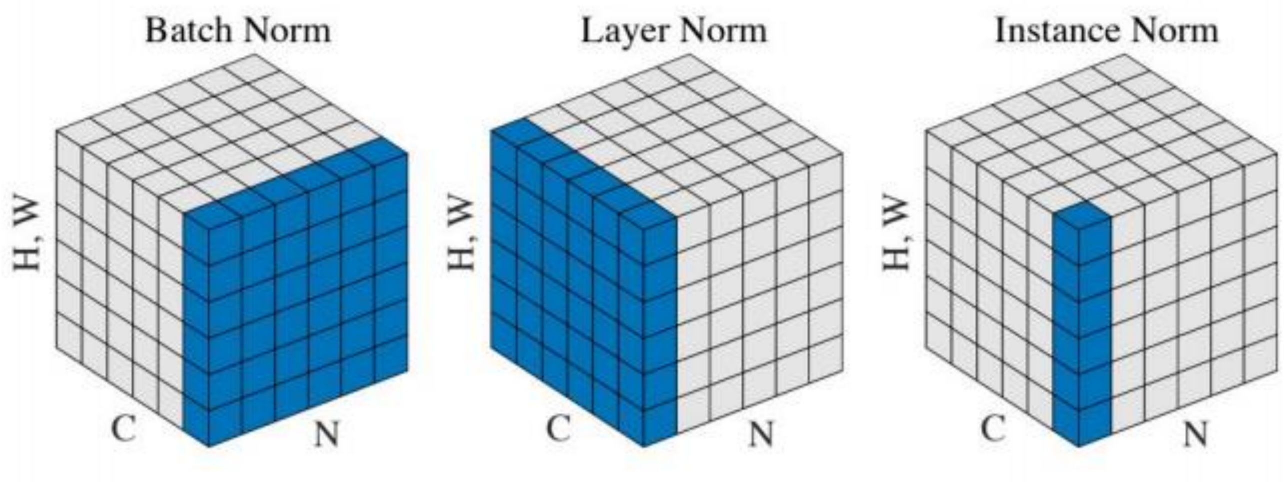
$$\mu,\sigma:\ N\times C\times 1\times 1$$

$$\gamma,\beta:\ 1\times C\times 1\times 1$$

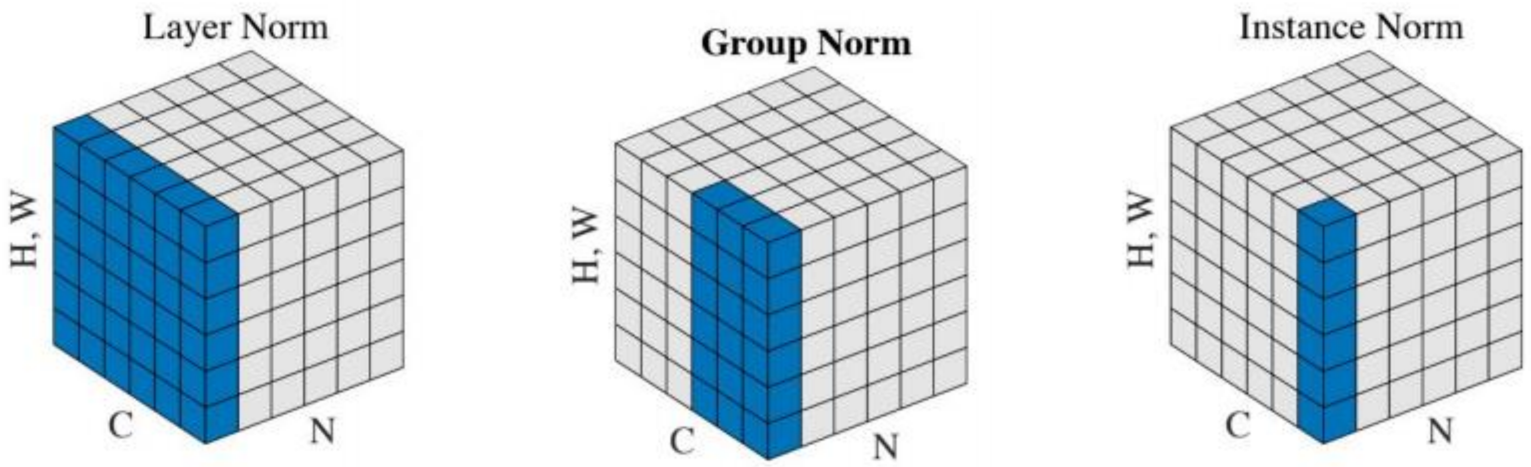
$$y = \gamma(x-\mu)/\sigma+\beta$$

Instance Normalization (IN, Ulyanov et al., 2016) 最初应用于图像的风格迁移领域。作者发现，在生成模型中， feature map 各个 channel 的均值和方差都会影响最终生成图像的风格，因此在每个 channel 中都进行归一化。

归一化层的比较



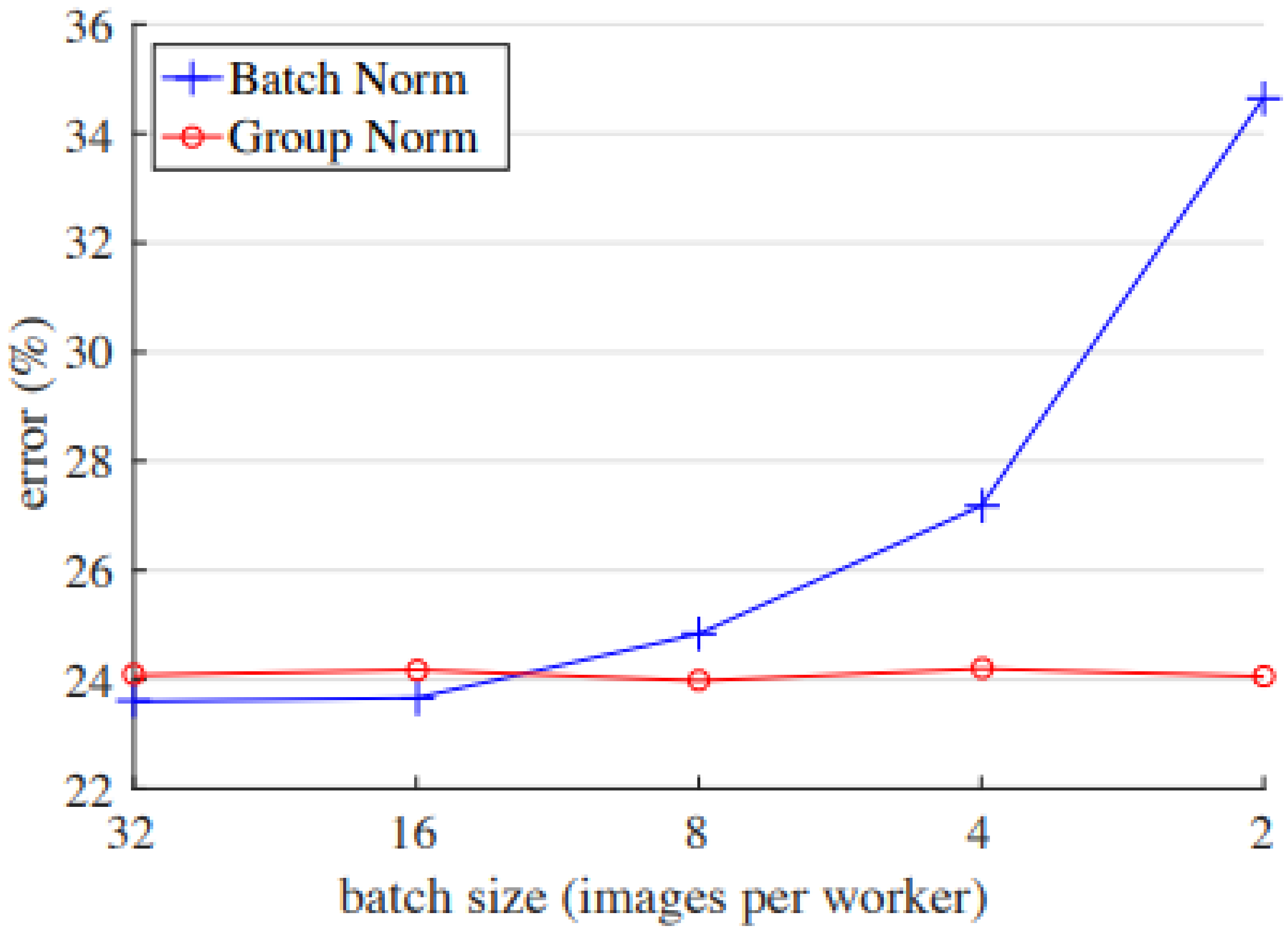
归一化层的比较



- Group Norm (GN) 适用于占用显存比较大的任务，比如图像分割。对这类任务，可能 batch size 只能是个位数。此时，BN 对数据的均值和标准差估计很不精确。GN 也是独立于 batch 的，它是 Layer Norm 和 Instance Norm 的折中。

- GN 将每一个样本 feature map 的 channel 分成 G 组，每组包含 C/G 个 channel，然后将这些 channel 中的元素求均值和标准差来进行归一化。

逐层归一化—Group Normalization (GN)



观察：

- batch size 比较大时 (batch size = 32, 16)，Batch Norm 效果略微好于 Group Norm；
- batch size 比较小时 (batch size = 8, 4, 2)，Group Norm 效果明显好于 Batch Norm。

目录

随机初始化

□Gaussian分布初始化

- 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。

$$\boldsymbol{W} = \sigma * \mathrm{np.random.randn(fan\_in, fan\_out)}$$

□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

若方差为σ^2，那么$r = \sqrt{3\sigma^2}$

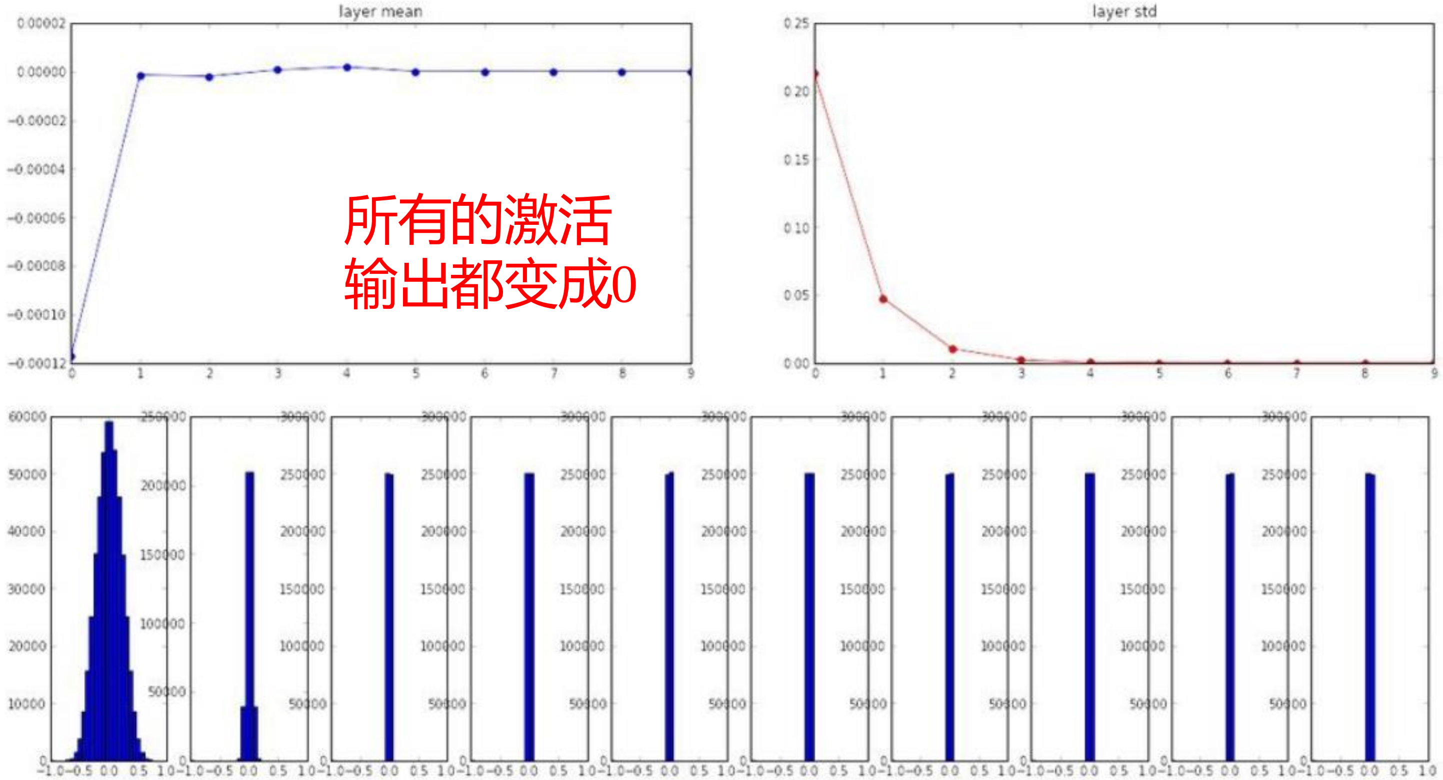
$$\boldsymbol{W} = \mathrm{r} * \mathrm{np.random.rand(fan\_in, fan\_out)}$$

在浅层网络时效果不错，但是深层网络就有较大问题

随机初始化的问题

```
input layer had mean 0.000927 and std 0.998388
hidden layer 1 had mean -0.000117 and std 0.213081
hidden layer 2 had mean -0.000001 and std 0.047551
hidden layer 3 had mean -0.000002 and std 0.010630
hidden layer 4 had mean 0.000001 and std 0.002378
hidden layer 5 had mean 0.000002 and std 0.000532
hidden layer 6 had mean -0.000000 and std 0.000119
hidden layer 7 had mean 0.000000 and std 0.000026
hidden layer 8 had mean -0.000000 and std 0.000006
hidden layer 9 had mean 0.000000 and std 0.000001
hidden layer 10 had mean -0.000000 and std 0.000000
```

- 看看激活函数的输出统计
- 10层，每层500个神经元，用tanh非线性函数，用小随机数初始化

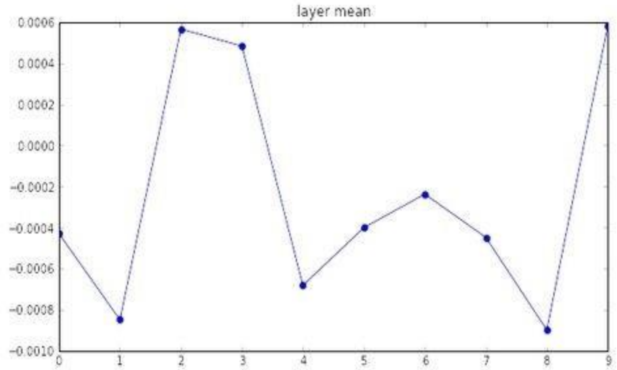


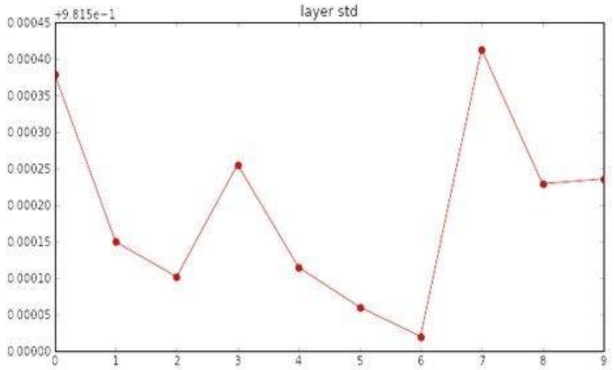
$$W = 0.01 * \mathrm{np.random.randn(fan\_in, fan\_out)}$$

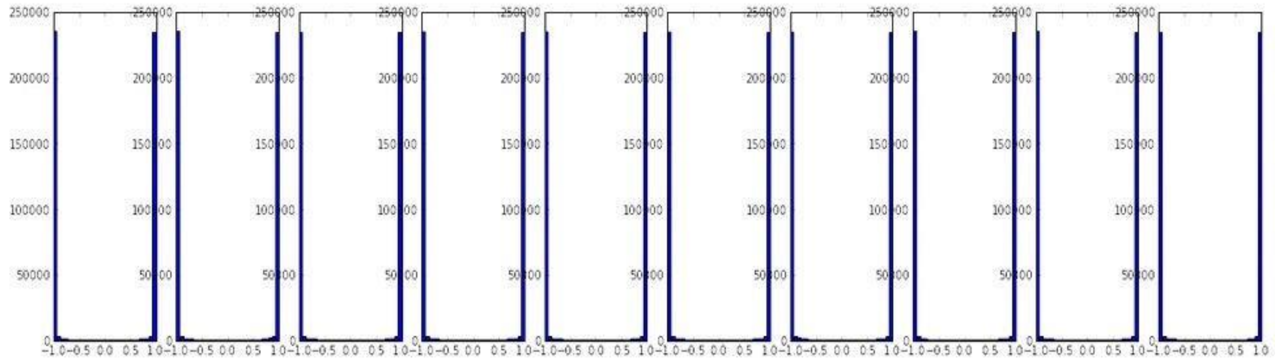
随机初始化的问题

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean -0.000430 and std 0.981879
hidden layer 2 had mean -0.000849 and std 0.981649
hidden layer 3 had mean 0.000566 and std 0.981601
hidden layer 4 had mean 0.000483 and std 0.981755
hidden layer 5 had mean -0.000682 and std 0.981614
hidden layer 6 had mean -0.000401 and std 0.981560
hidden layer 7 had mean -0.000237 and std 0.981520
hidden layer 8 had mean -0.000448 and std 0.981913
hidden layer 9 had mean -0.000899 and std 0.981728
hidden layer 10 had mean 0.000584 and std 0.981736
```

梯度为0







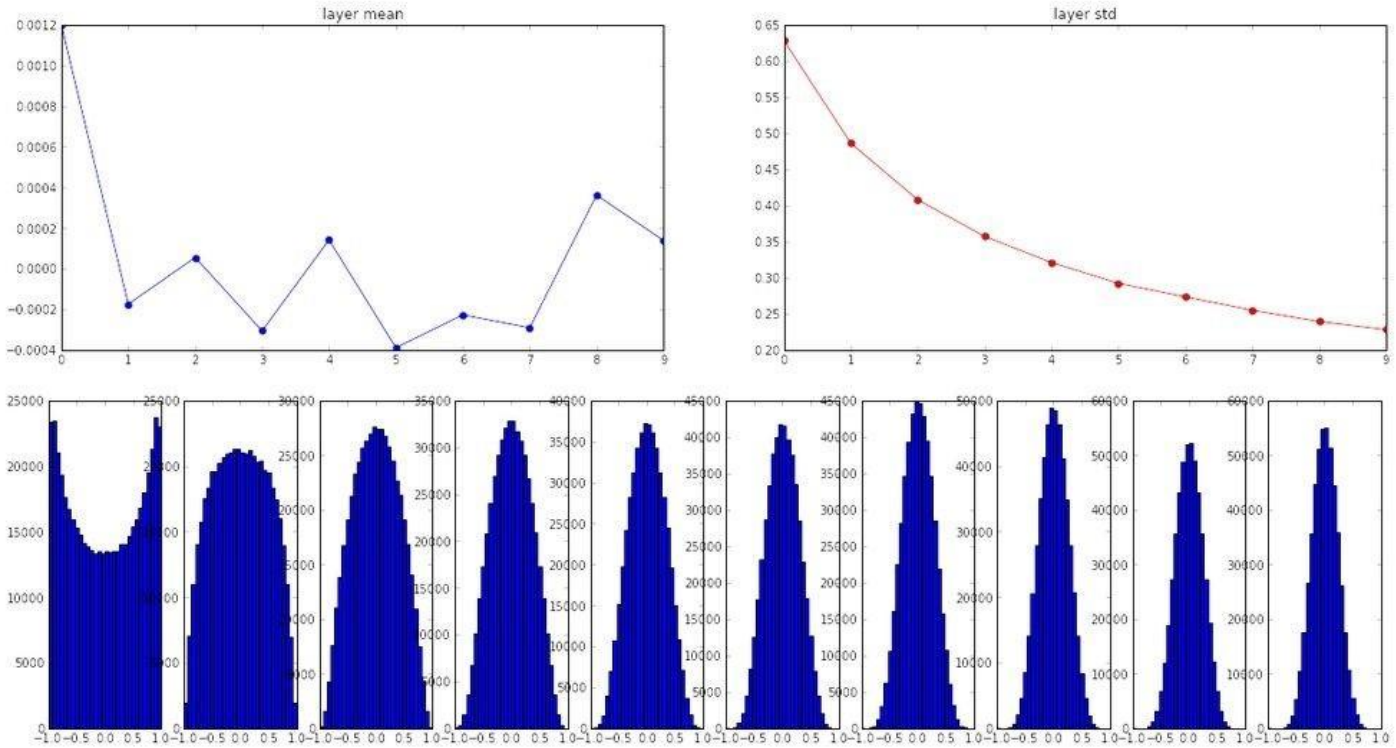
$$W = {\color{red}1} * \mathrm{np.random.randn(fan\_in, fan\_out)}$$

Xavier 初始化 (Glorot et al., 2010)

```
input layer had mean 0.001800 and std 1.001311
hidden layer 1 had mean 0.001198 and std 0.627953
hidden layer 2 had mean -0.000175 and std 0.486051
hidden layer 3 had mean 0.000055 and std 0.407723
hidden layer 4 had mean -0.000306 and std 0.357108
hidden layer 5 had mean 0.000142 and std 0.320917
hidden layer 6 had mean -0.000389 and std 0.292116
hidden layer 7 had mean -0.000228 and std 0.273387
hidden layer 8 had mean -0.000291 and std 0.254935
hidden layer 9 had mean 0.000361 and std 0.239266
hidden layer 10 had mean 0.000139 and std 0.228008
```

也可以为均匀分布

$$\mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}\right)$$



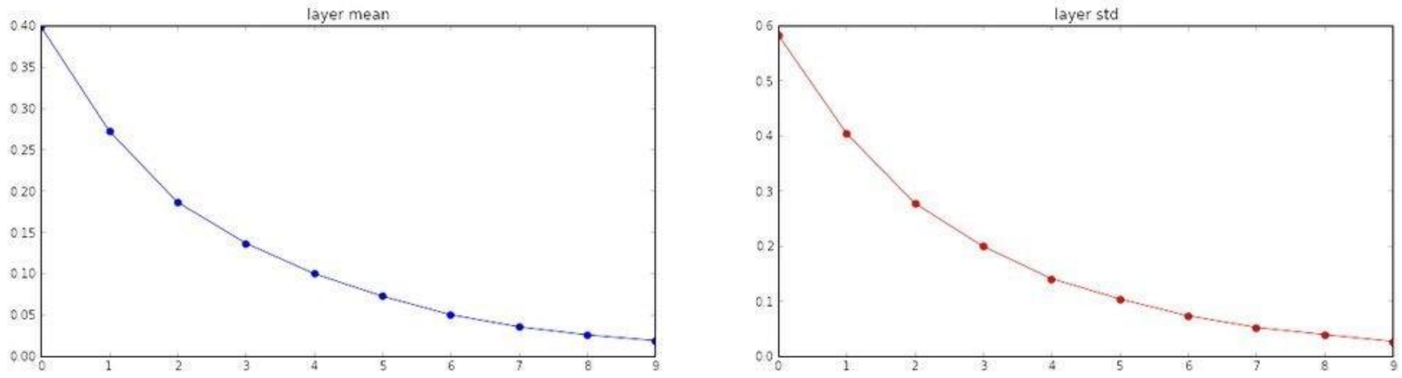
W=

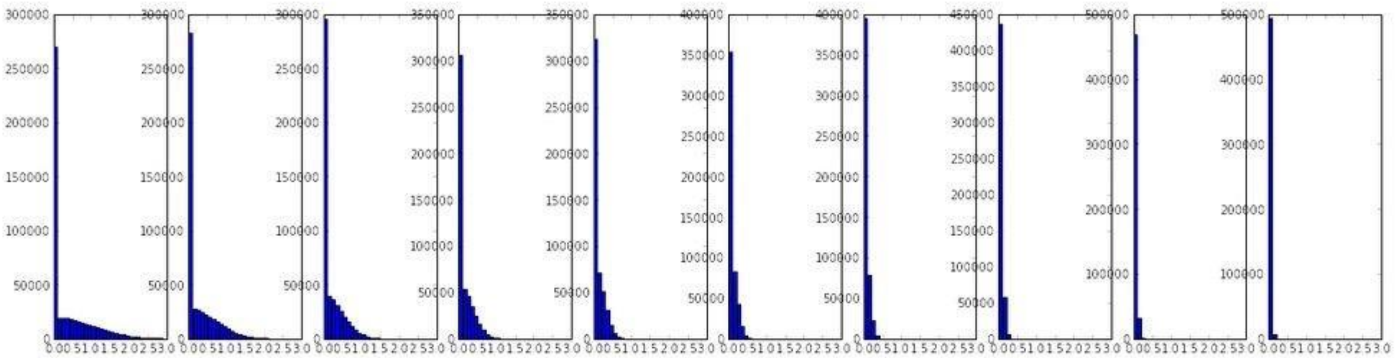
$\text{np.sqrt}(6) * \text{np.random.randn}(\text{fan\_in}, \text{fan\_out})/\text{np.sqrt}(\text{fan}_{\text{in}} + \text{fan}_{\text{out}})$

Xavier 初始化的问题

```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.398623 and std 0.582273
hidden layer 2 had mean 0.272352 and std 0.403795
hidden layer 3 had mean 0.186076 and std 0.276912
hidden layer 4 had mean 0.136442 and std 0.198685
hidden layer 5 had mean 0.099568 and std 0.140299
hidden layer 6 had mean 0.072234 and std 0.103280
hidden layer 7 had mean 0.049775 and std 0.072748
hidden layer 8 had mean 0.035138 and std 0.051572
hidden layer 9 had mean 0.025404 and std 0.038583
hidden layer 10 had mean 0.018408 and std 0.026076
```

当使用ReLU时，
方差又接近0



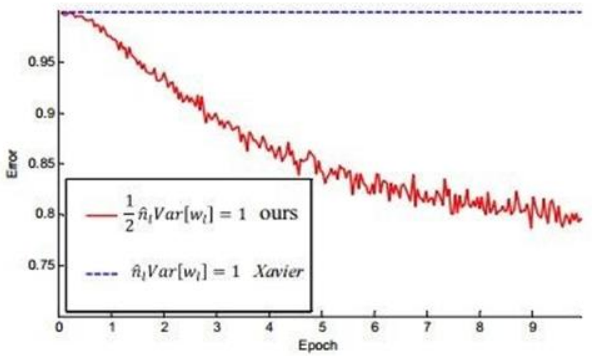


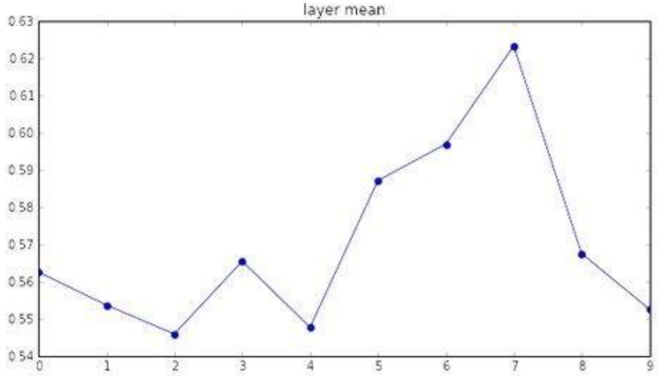
W=

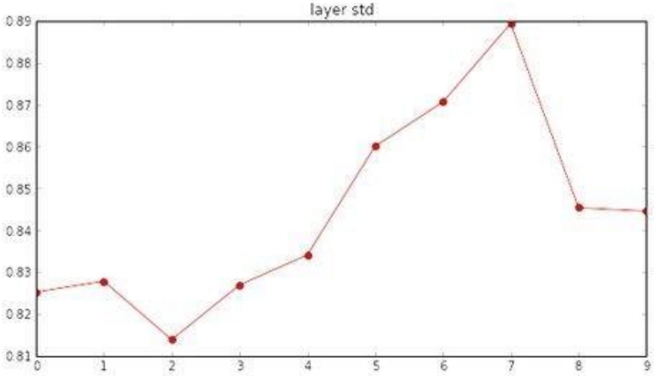
$$\mathrm{np.sqrt}(6) * \mathrm{np.random.randn}(\mathrm{fan\_in}, \mathrm{fan\_out}) / \mathrm{np.sqrt}(\mathrm{fan}_{\mathrm{in}} + \mathrm{fan}_{\mathrm{out}})$$

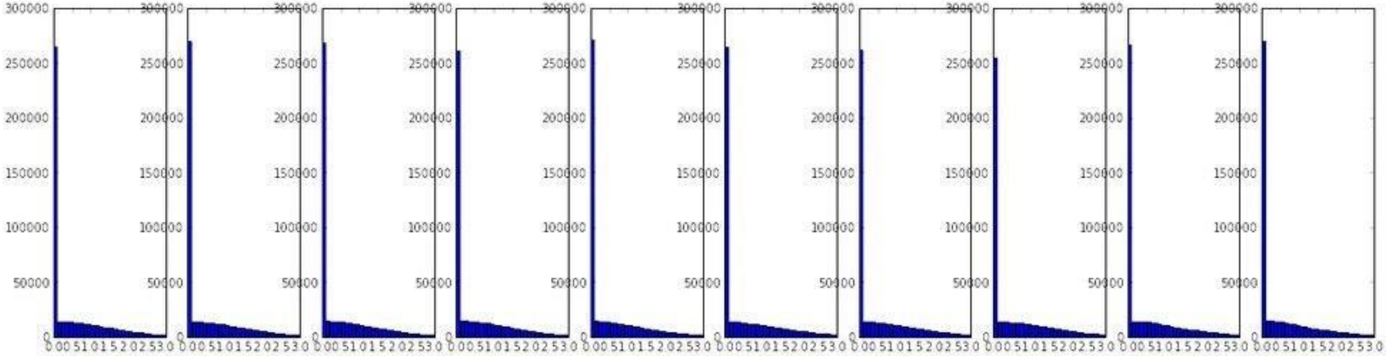
He 初始化（针对ReLU激活）

```
input layer had mean 0.000501 and std 0.999444
hidden layer 1 had mean 0.562488 and std 0.825232
hidden layer 2 had mean 0.553614 and std 0.827835
hidden layer 3 had mean 0.545867 and std 0.813855
hidden layer 4 had mean 0.565396 and std 0.826902
hidden layer 5 had mean 0.547678 and std 0.834092
hidden layer 6 had mean 0.587103 and std 0.860035
hidden layer 7 had mean 0.596867 and std 0.870610
hidden layer 8 had mean 0.623214 and std 0.889348
hidden layer 9 had mean 0.567498 and std 0.845357
hidden layer 10 had mean 0.552531 and std 0.844523
```









$$W = \text{np.random.randn(fan\_in, fan\_out)}/\text{np.sqrt}(2/\text{fan\_in})$$

[He et al., 2015]

权重初始化背后的数学 (He et al., 2015)

- 思路：保持前向传播中不同层间激活值 / 反向传播中不同层间梯度 的方差相同
- 第l层的仿射变换 $\boldsymbol{y}_l = \boldsymbol{W}_l \boldsymbol{x}_l + \boldsymbol{b}_l$ 非线性变换 $\boldsymbol{x}_l = \mathrm{ReLU}(\boldsymbol{y}_{l-1})$
- 假设W_l每个元素是独立同分布的，x_l每个元素也是独立同分布
- 假设W_l与x_l互相独立

$$
\begin{aligned}
\boldsymbol{y}_l[i] &= \boldsymbol{W}_l[i,:]^\top \boldsymbol{x}_l + \boldsymbol{b}_l[i] \\
&= \sum_d \boldsymbol{W}_l[i,d]\boldsymbol{x}_l[d] + \boldsymbol{b}_l[i]
\end{aligned}
$$

$$\mathrm{var}[y_l] = n_l \mathrm{var}[w_l x_l]$$

$\boldsymbol{y}_l$的元素 $\boldsymbol{x}_l$的维度 $\boldsymbol{W}_l$的元素 $\boldsymbol{x}_l$的元素

$$\mathrm{var}[x_1 + \cdots + x_n] = \mathrm{var}[x_1] + \cdots + \mathrm{var}[x_n]$$

权重初始化背后的数学

□设$\mathbb{E}[w_l] = 0$, 那么 $\mathbb{E}[w_l x_l] = \mathbb{E}[w_l]\mathbb{E}[x_l] = 0$

$$\mathrm{var}[y_l] = n_l \mathrm{var}[w_l x_l] \quad \Rightarrow \quad \mathrm{var}[y_l] = n_l \mathrm{var}[w_l]\mathbb{E}[x_l^2]$$

$$\begin{aligned}
\mathrm{var}[w_l x_l] &= \mathbb{E}[w_l^2 x_l^2] - (\mathbb{E}[w_l x_l])^2 \\
&= \mathbb{E}[w_l^2 x_l^2] \\
&= \mathbb{E}[w_l^2]\mathbb{E}[x_l^2] \\
&= (\mathbb{E}[w_l^2] - \mathbb{E}^2[w_l])\mathbb{E}[x_l^2] \\
&= \mathrm{var}[w_l]\mathbb{E}[x_l^2]
\end{aligned}$$

权重初始化背后的数学

- **假设** $b_{l-1} = 0$ **且** $\mathbb{E}[w_{l-1}] = 0$，**那么** $\mathbb{E}[y_{l-1}] = \mathbb{E}[w_{l-1}x_{l-1}] = 0$
- **由于** $x_l = \mathrm{ReLU}(y_{l-1})$，$\mathbb{E}[x_l^2] = \frac{1}{2}\mathbb{E}[y_{l-1}^2] = \frac{1}{2}\mathrm{var}[y_{l-1}]$

$$\mathbb{E}[x_l^2] = \mathbb{E}\left[\left(\mathrm{ReLU}(y_{l-1})\right)^2\right] = \int_{-\infty}^{+\infty}\left(\mathrm{ReLU}(y_{l-1})\right)^2 p(y_{l-1})dy_{l-1}$$

$$= \int_{0}^{+\infty}\left(\mathrm{ReLU}(y_{l-1})\right)^2 p(y_{l-1})dy_{l-1} = (\text{对称性})\frac{1}{2}\int_{-\infty}^{+\infty} y_{l-1}^2 p(y_{l-1})dy_{l-1} = \frac{1}{2}\mathbb{E}[y_{l-1}^2]$$

$$\mathrm{var}[y_l] = n_l\mathrm{var}[w_l]E\left[x_l^2\right] \Rightarrow \mathrm{var}[y_l] = \frac{n_l}{2}\mathrm{var}[w_l]\mathrm{var}[y_{l-1}]$$

$$\mathrm{var}[y_l] = \mathrm{var}[y_{l-1}] \Rightarrow \frac{n_l}{2}\mathrm{var}[w_l] = 1$$

研究反向传播时可以得出类似的结论 $\frac{n_{l+1}}{2}\mathrm{var}[w_l] = 1$

权重初始化背后的数学

□如果使用PReLU，则 $x_l = \mathrm{prelu}(y_{l-1})$，$\mathbb{E}\left[x_l^2\right] = \frac{1}{2}(1 + a^2)\mathrm{var}[y_{l-1}]$

$$f(y_i) = \begin{cases} y_i, & \text{if } y_i > 0 \\ a_i y_i, & \text{if } y_i \le 0 \end{cases}$$

$$\mathrm{var}[y_l] = n_l \mathrm{var}[w_l] E\left[x_l^2\right] \Rightarrow \mathrm{var}[y_l] = \frac{1 + a^2}{2} n_l \mathrm{var}[w_l] \mathrm{var}[y_{l-1}]$$

$$\mathrm{var}[y_l] = \mathrm{var}[y_{l-1}] \Rightarrow \frac{1 + a^2}{2} n_l \mathrm{var}[w_l] = 1$$

$$\mathrm{var}[w_l] = \frac{2}{1 + a^2} \frac{1}{n_l}$$

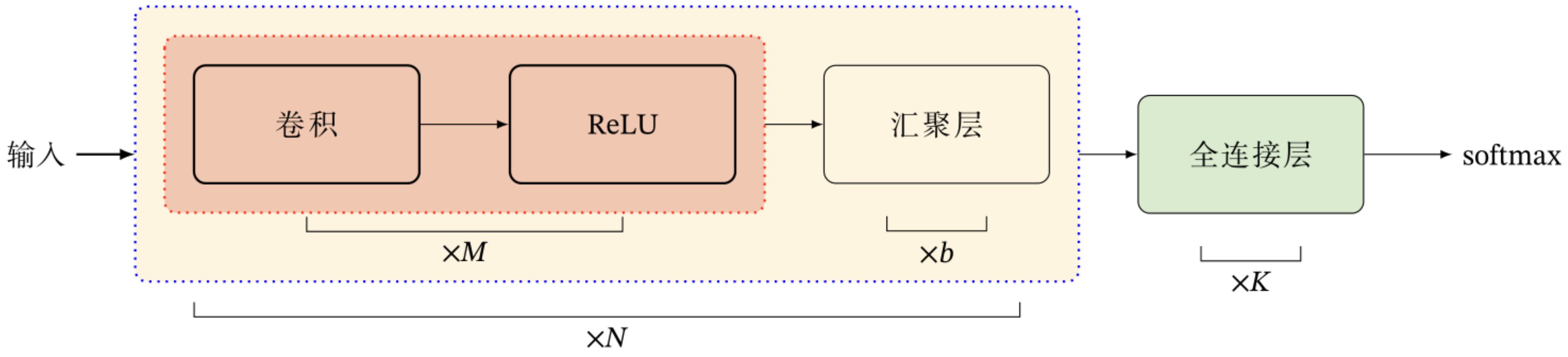
权重初始化的总结

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	$r = \sqrt{\frac{6}{M_{l-1}}}$	$\sigma^2 = \frac{2}{M_{l-1}}$

目录

卷积神经网络

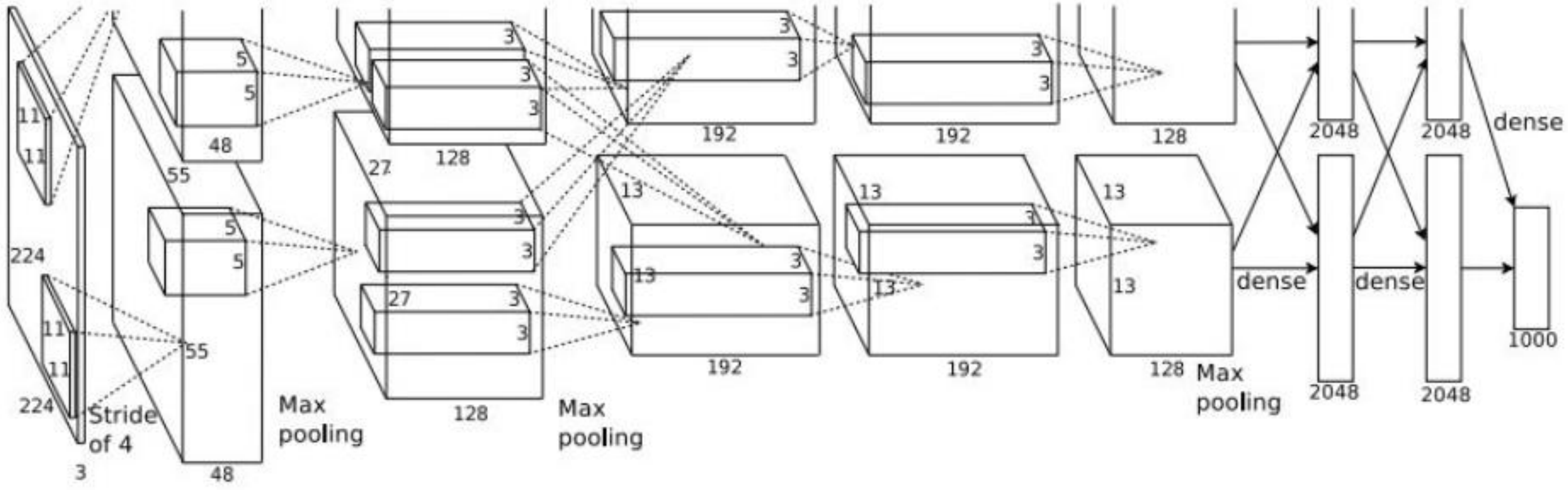
口典型结构



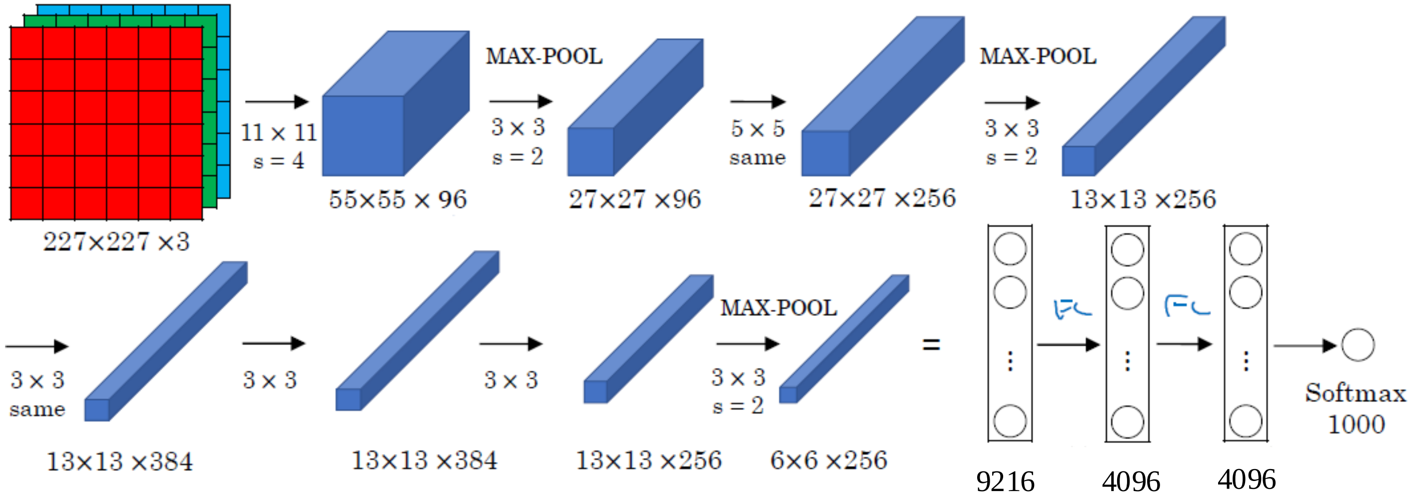
一个卷积块为连续M 个卷积层和b个汇聚层（M通常设置为2 ~ 5，b为0或1）。一个卷积网络中可以堆叠N 个连续的卷积块，然后在接着K 个全连接层（N 的取值区间比较大，比如1 ~ 100或者更大；K一般为0 ~ 2）。

第一个深度卷积网络—AlexNet

- **2012 ILSVRC winner**
  - top 5 error of 16% compared to runner-up with 26% error
- **第一个现代深度卷积网络模型**
  - 首次使用了很多现代深度卷积网络的一些技术方法
  - 使用GPU进行并行训练，采用了ReLU作为非线性激活函数，使用Dropout防止过拟合，使用数据增强
- **5个卷积层、3个汇聚层和3个全连接层**

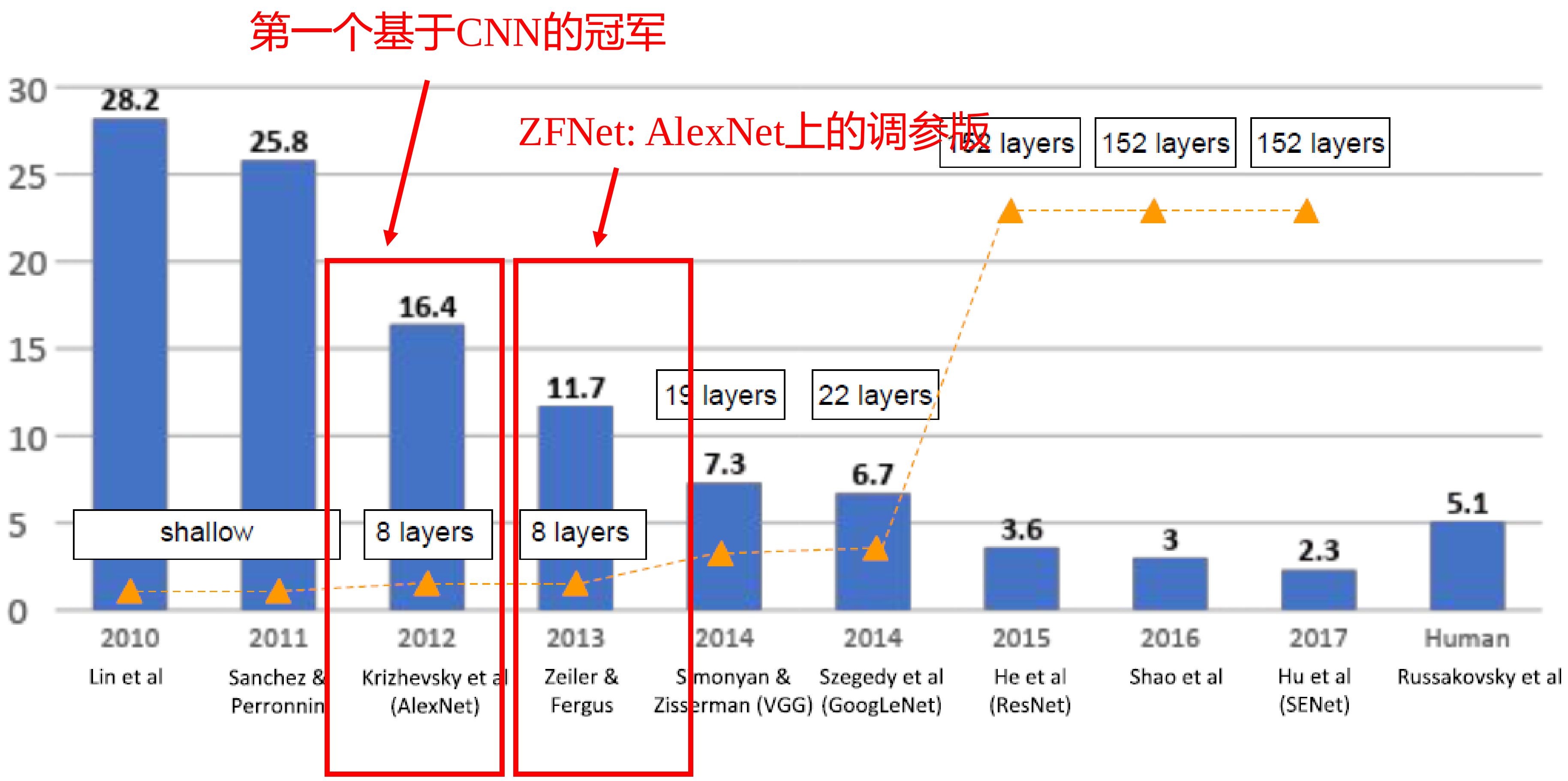


第一个深度卷积网络—AlexNet

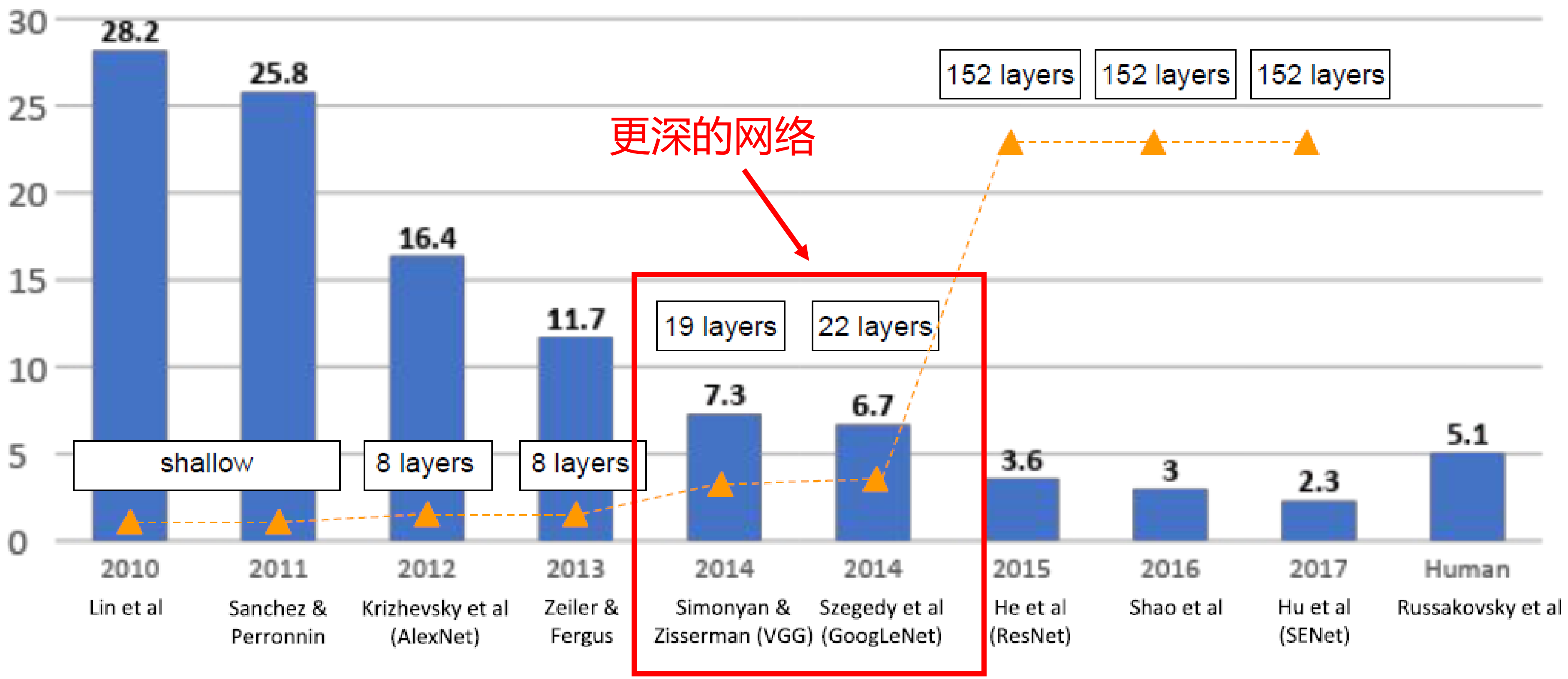


[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Large Scale Visual Recognition Challenge



Large Scale Visual Recognition Challenge



典型的卷积网络—VGG

AlexNet

11.7% top 5 error in ILSVRC’13 (ZFNet)

- **更小的核，更小的步幅，更深的网络**
- **8层 → 16-19层**
- **只有3x3的卷积核，步幅为1，补1个0**
- **2x2的池化核，步幅为2**

→ 7.3% top 5 error in ILSVRC’14

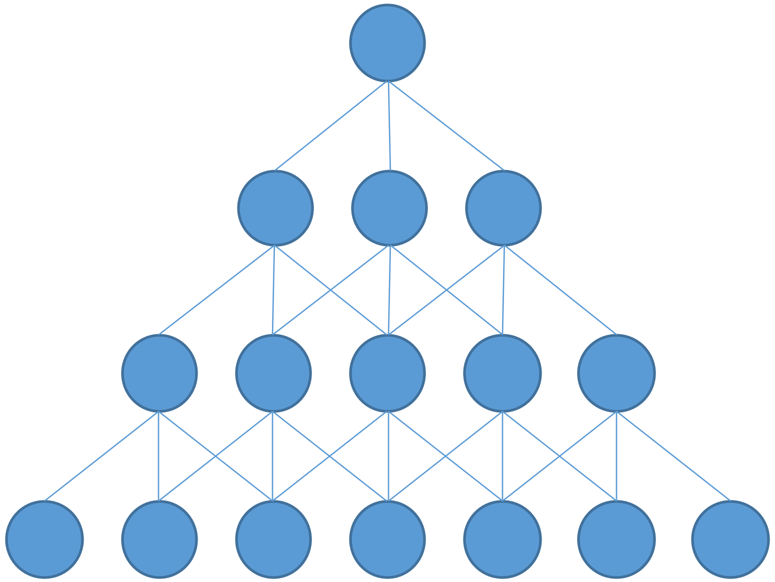
VGG16

VGG19

典型的卷积网络—VGG

口为什么更小的核（3x3）

- 3个3x3的卷积层的堆放的有效感受野和一个7x7的相同



典型的卷积网络—VGG

口为什么更小的核（3x3）

- 3个3x3的卷积层的堆放的有效感受野和一个7x7的相同
- 更深的卷积层，非线性更强
- 参数更少

假设每层C通道，参数个数

$$3(3^2C^2) \text{ vs. } 7^2C^2$$

典型的卷积网络—VGG

- INPUT: [224x224x3] memory: 224*224*3=150K params: 0
- CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
- CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
- POOL2: [112x112x64] memory: 112*112*64=800K params: 0
- CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
- CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
- POOL2: [56x56x128] memory: 56*56*128=400K params: 0
- CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
- CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
- CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
- POOL2: [28x28x256] memory: 28*28*256=200K params: 0
- CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
- CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
- CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
- POOL2: [14x14x512] memory: 14*14*512=100K params: 0
- CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
- CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
- CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
- POOL2: [7x7x512] memory: 7*7*512=25K params: 0
- FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
- FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
- FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image

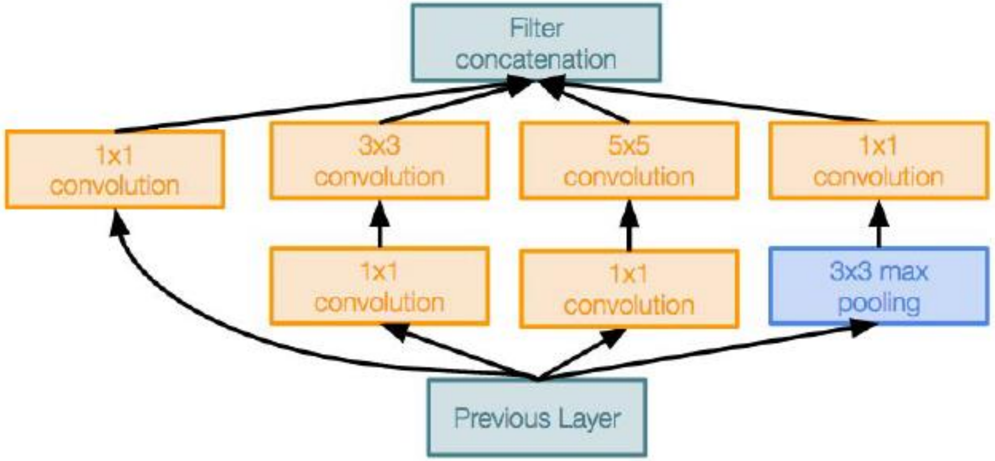
TOTAL params: 138M parameters

VGG16

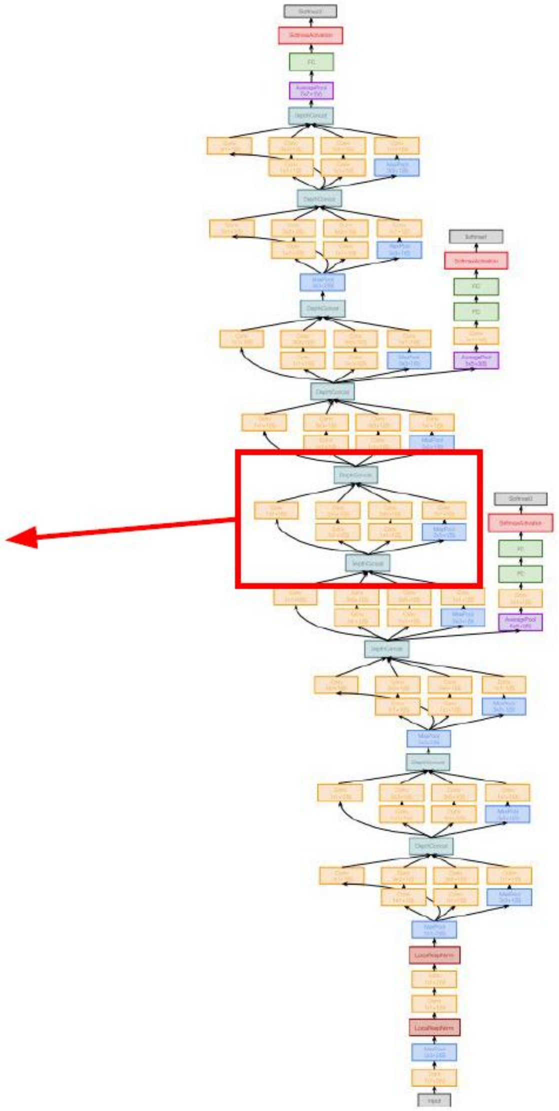
典型的卷积网络—GoogLeNet

□2014 ILSVRC winner (6.7% top 5 error)

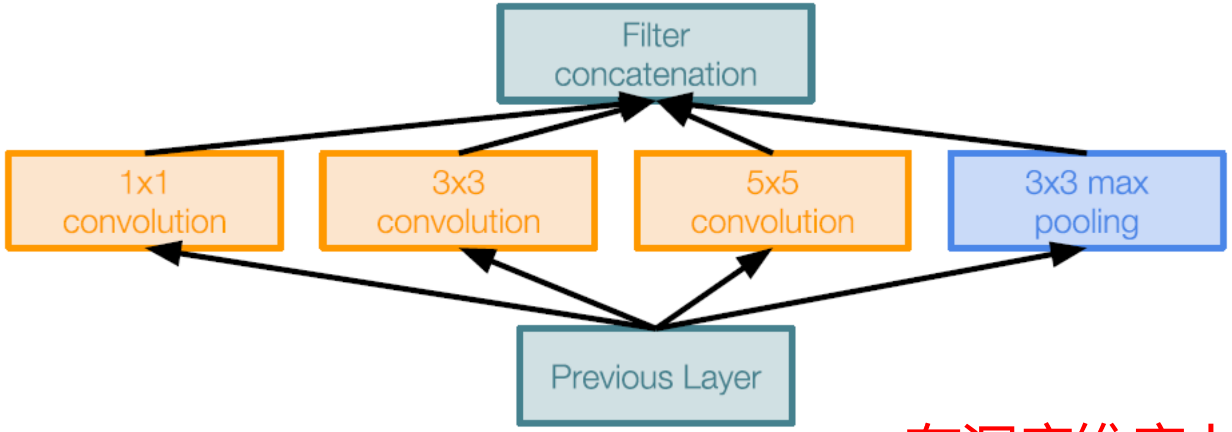
- 更深的网络（22层），没有FC层，且计算效率高
- 只有5M 参数，比AlexNet的60M参数的1/12
- 由多个inception模块和少量的汇聚层堆叠而成



Inception module



典型的卷积网络—GoogLeNet

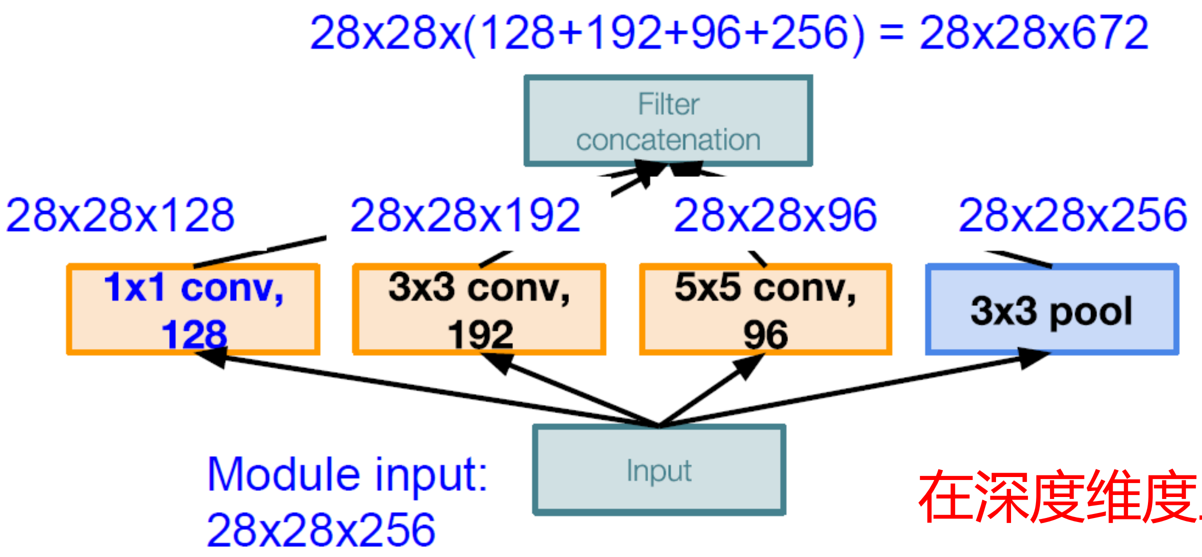


在深度维度上拼接滤波器的输出

Naive Inception module

- 在卷积网络中，如何设置卷积层的卷积核大小是一个十分关键的问题
- Inception同时使用1×1、3×3、5×5等不同大小的卷积核（对应不同感受野），将得到的特征映射在深度上拼接起来作为输出特征映射

典型的卷积网络—GoogLeNet



Naive Inception module

在深度维度上拼接滤波器的输出

计算复杂度大

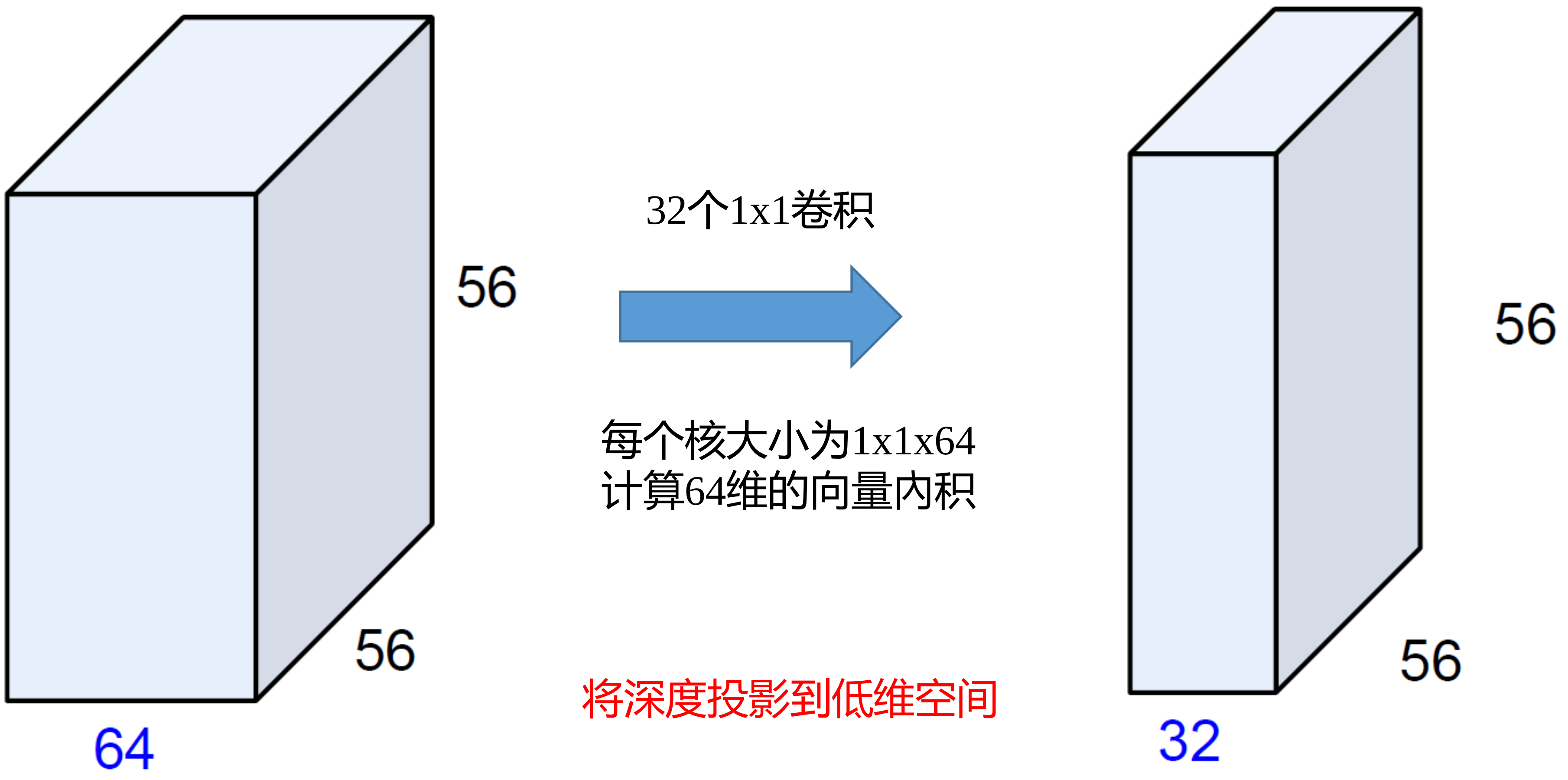
□Conv Ops:

- [1x1 conv, 128] 28x28x128x1x1x256
- [3x3 conv, 192] 28x28x192x3x3x256
- [5x5 conv, 96] 28x28x96x5x5x256

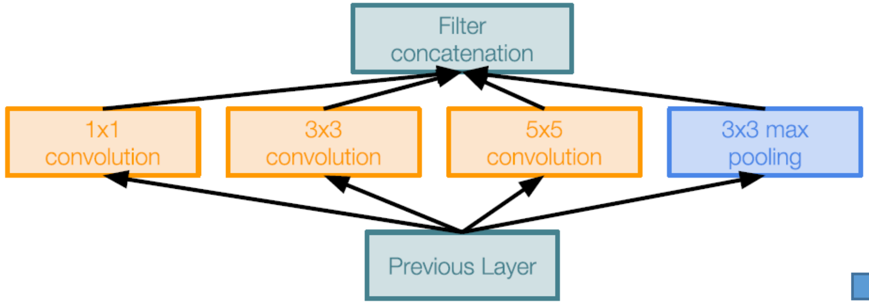
□Total: 854M ops

典型的卷积网络—GoogLeNet

口使用“bottleneck” 层（1x1卷积）来减少特征深度

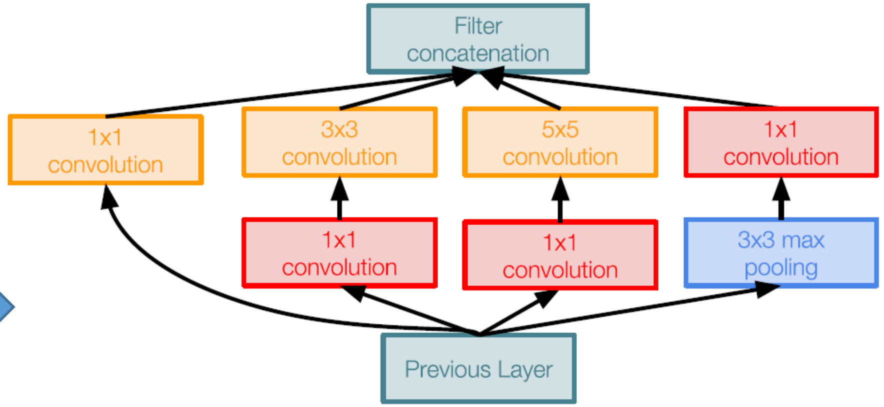


典型的卷积网络—GoogLeNet



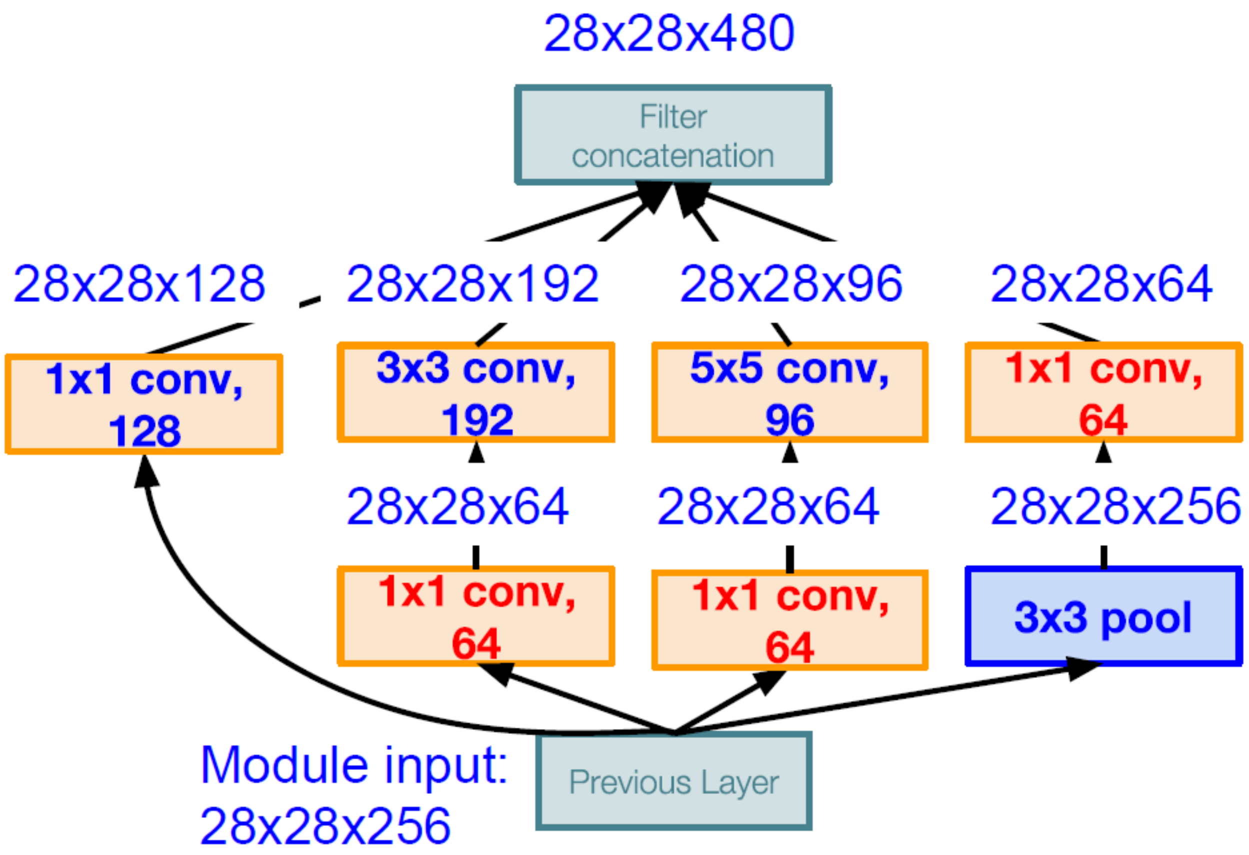
Naive Inception module

1x1 conv “bottleneck” layers



Inception module with dimension reduction

典型的卷积网络—GoogLeNet



Conv Ops:

- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 128] 28x28x128x1x1x256
- [3x3 conv, 192] 28x28x192x3x3x64
- [5x5 conv, 96] 28x28x96x5x5x64
- [1x1 conv, 64] 28x28x64x1x1x256

从Total: 854M ops降到了Total: 358M ops

Inception模块 v3

口用多层的小卷积核来替换大的卷积核，以减少计算量和参数量。

- 使用两层3x3的卷积来替换v1中的5x5的卷积
- 使用连续的nx1和1xn来替换nxn的卷积。

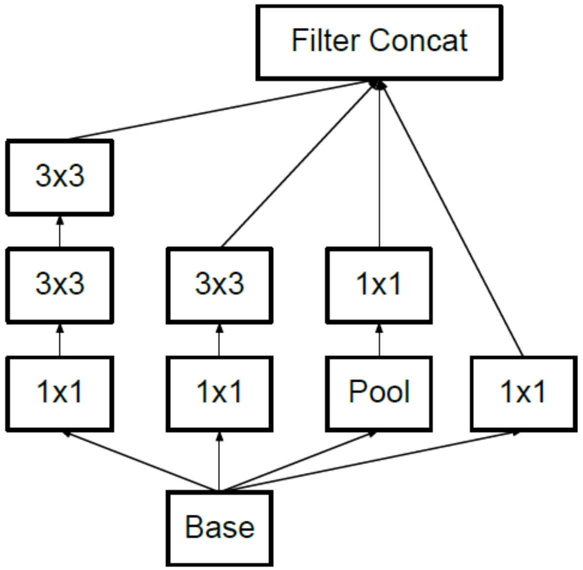


Figure 5. Inception modules where each 5×5 convolution is replaced by two 3×3 convolution, as suggested by principle 3 of Section 2.

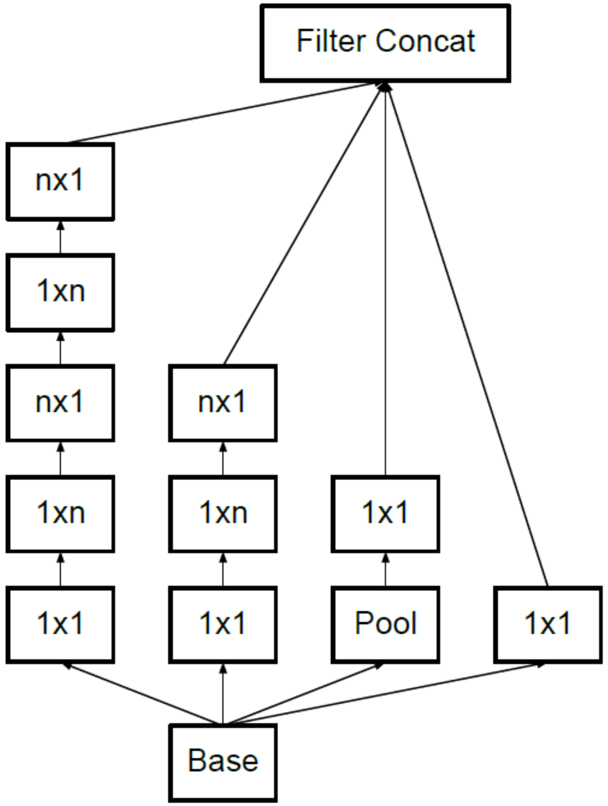


Figure 6. Inception modules after the factorization of the $n \times n$ convolutions. In our proposed architecture, we chose $n = 7$ for the 17×17 grid. (The filter sizes are picked using principle 3)

Large Scale Visual Recognition Challenge