第4章：深度学习基础

中国科学技术大学
电子工程与信息科学系

主讲教师：李厚强 (lihq@ustc.edu.cn)
周文罡 (zhwg@ustc.edu.cn)
李　礼 (lil1@ustc.edu.cn)
胡　洋 (eeyhu@ustc.edu.cn)

深度学习基础

- 深度前馈网络
- 卷积神经网络
- 循环神经网络
- Transformer网络

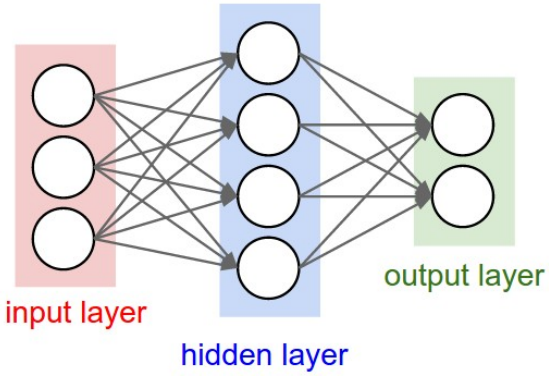
深度学习基础

- 深度前馈网络
  - 基本结构
  - 非线性激活函数
  - 反向传播算法
  - 优化算法
- 卷积神经网络
- 循环神经网络
- Transformer网络

深度前馈网络

- 深度前馈网络（多层感知机）
  - 深度前馈网络由输入层，隐藏层，输出层组成
  - 深度前馈网络本质上是在学习函数映射
  - 假设输入 $x_i \in \mathbb{R}^{D_{in}}$ ，输出 $y_i \in \mathbb{R}^{D_{out}}$

2层神经网络

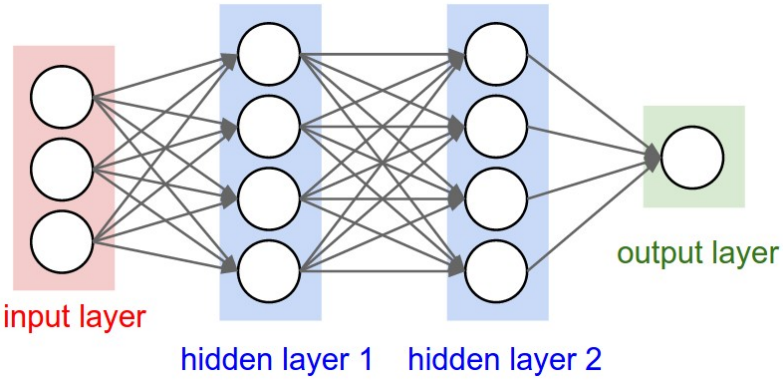


$$g(\mathbf{x}, W_1, W_2) = f(W_2 f(W_1 \mathbf{x}))$$

$$W_1 \in \mathbb{R}^{H \times D_{in}}$$

$$W_2 \in \mathbb{R}^{D_{out} \times H}$$

3层神经网络



$$g(\mathbf{x}, W_1, W_2, W_3) = f(W_3 f(W_2 f(W_1 \mathbf{x})))$$

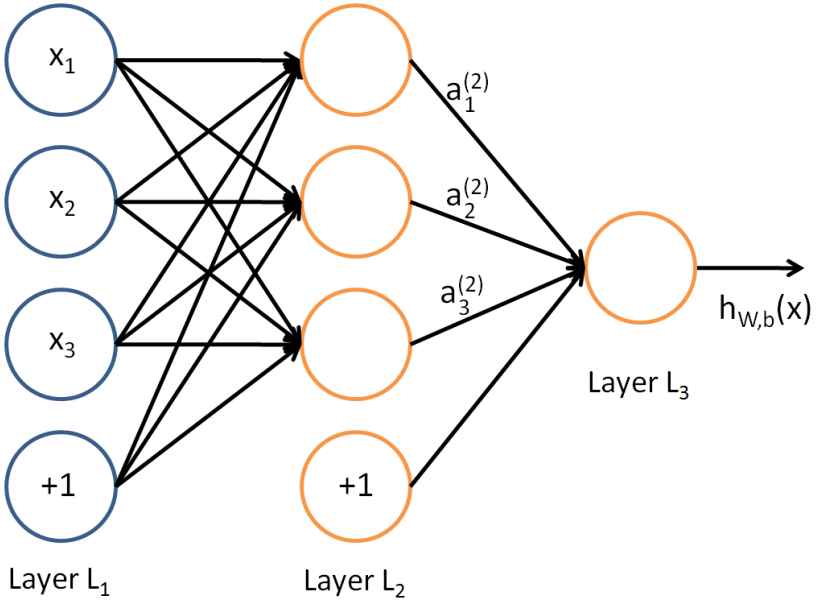
$$W_1 \in \mathbb{R}^{H_1 \times D_{in}}$$

$$W_2 \in \mathbb{R}^{H_2 \times H_1}$$

$$W_3 \in \mathbb{R}^{D_{out} \times H_2}$$

前馈网络：示例

神经网络模型



$f(\cdot)$: 激活函数（activation function）

$a_i^{(l)}$：第l层第i单元的输出值

$z_i^{(l)}$：第l层第i单元的输入加权和

$$a_1^{(2)} = f\left(z_1^{(2)}\right)$$
$$= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

$$a_2^{(2)} = f\left(z_2^{(2)}\right)$$
$$= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$a_3^{(2)} = f\left(z_3^{(2)}\right)$$
$$= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$

$$\mathrm{h}_{\mathbf{W},b}(\mathbf{x}) = a_1^{(3)} = f(z_1^{(3)})$$
$$= f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

激活函数

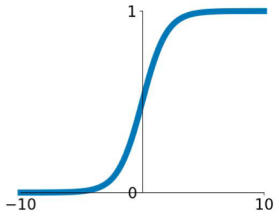
- 激活函数作用
  - 引入非线性，增强网络的表达能力
- 激活函数的性质
  - 非线性，连续可导（允许少数点上不可导）
    - ✓ 可以用数值优化的方法学习网络参数
  - 激活函数及其导数尽可能简单
    - ✓ 有利于提高网络计算效率
  - 激活函数的导函数的值域在一个合适的区间内
    - ✓ 太大或太小可能影响训练的效率和稳定性

激活函数

- 常见激活函数
  - S型函数

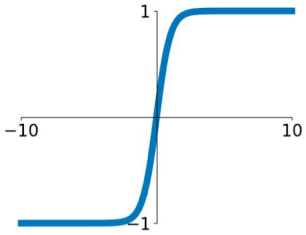
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

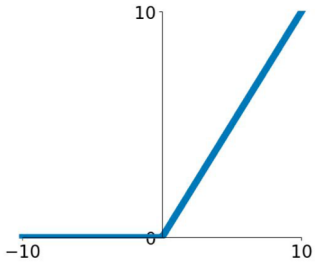


Tanh

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

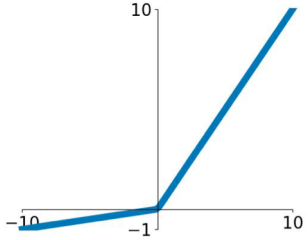


  - 斜坡函数



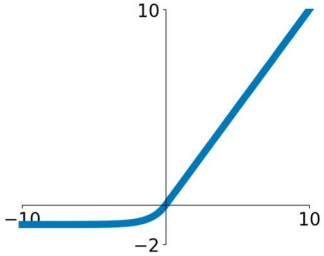
ReLU

$\max(0, x)$



Leaky ReLU

$\max(\beta x, x)$



ELU

$$\begin{cases} x & , \quad x \geq 0 \\ \alpha(e^{x}-1), & x < 0 \end{cases}$$

激活函数

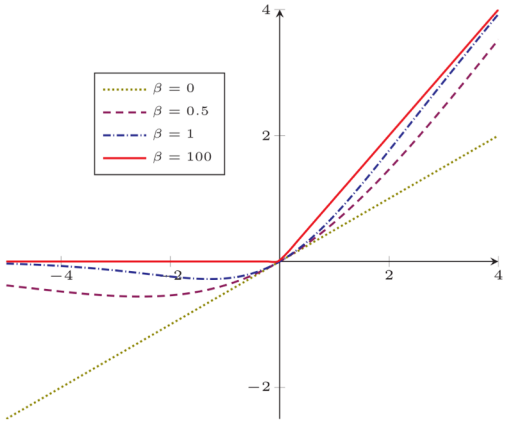
常见激活函数

复合函数

Swish(SiLU)

$$\text{Swish}(x) = x\sigma(\beta x)$$

- 一种自门控激活函数



GELU

$$\text{GELU}(x) = x\text{P}(X \leq x)$$

- $\text{P}(X \leq x)$是高斯分布的累积分布函数
- 由于$\text{P}(X \leq x)$是S型函数，可以用Sigmoid或Tanh函数近似

$$\text{GELU}(x) \approx x\sigma(1.702x)$$

或

$$\text{GELU}(x) \approx 0.5x(1 + \tanh(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)))$$

万能近似定理

- 万能近似定理（universal approximation theorem）

一个前馈神经网络如果具有线性输出层和至少一层具有任何一种“挤压”性质的激活函数（例如sigmoid激活函数）的隐藏层，只要给予网络足够数量的隐藏单元，它可以近似任何一个有限维空间到另一个有限维空间的Borel可测函数。

（定义在$\boldsymbol{R}^{\mathrm{n}}$的有界闭集上的任意连续函数是Borel可测的，因此可以用神经网络来近似）

训练前馈网络

有监督学习

损失函数

- ✓ 平方损失：$\frac{1}{2}\left\|\mathrm{h}_{\mathbf{W},b}(\mathbf{x})-y\right\|^2$
- ✓ Hinge损失：$\sum_{j\neq y}\max\left(0,s_j-s_y+1\right)$
- ✓ 交叉熵损失：$-\log\left(\frac{e^{s_y}}{\sum_j e^{s_j}}\right)$

给定训练样本集 $\{\left(\mathbf{x}^{(1)},y^{(1)}\right),\ldots,(\mathbf{x}^{(m)},y^{(m)})\}$

$$J(\mathbf{W},b;\mathbf{x},y)=\frac{1}{m}\sum_{i=1}^{m}J(\mathbf{W},b;\mathbf{x}^{(i)},y^{(i)})+\frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{H_l}\sum_{j=1}^{H_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

公式中第一项为数据项，第二项为正则化项（也叫权重衰减weight decay项），其目的是减小权重的幅度，防止过拟合。

反向传播算法

前向计算

- 参数(随机)初始化
- 进行前馈传导计算，利用前向传导公式，得到$L_2, L_3, \ldots, L_{n_l}$的输出值

$$a_i^{(n_l)} = f\left(z_i^{(n_l)}\right)$$

- 对于第n_l层（输出层）的每个输出单元i，由预测值$\mathrm{h}_{\mathbf{W},b}(\mathbf{x})$和标签$y$计算网络预测的误差

以平方误差为例 $\frac{1}{2}\left\|y - f\left(z_i^{(n_l)}\right)\right\|^2$

反向传播算法

反向传播

- 由后向前，逐层计算损失函数相对于该层输入的偏导数（误差项）
- 计算J对$z_i^{(n_l)}$的偏导数

$$\begin{aligned}\delta_i^{(n_l)} &= \frac{\partial}{\partial z_i^{(n_l)}} J(\mathbf{W}, b; \mathbf{x}, y) \\ &= \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \left\| y - f\left(z_i^{(n_l)}\right) \right\|^2 \\ &= -(y - a_i^{(n_l)}) \cdot f'\left(z_i^{(n_l)}\right)\end{aligned}$$

以平方误差为例

反向传播算法

反向传播

- 由求导的链式法则，第n_l-1层中的第i个节点的误差项可由第n_l层的误差项计算得到

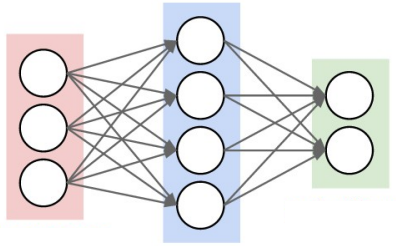
$$\delta_i^{(n_l-1)} = \frac{\partial}{\partial z_i^{(n_l-1)}} J(\mathbf{W}, b; \mathbf{x}, y) = \sum_j \frac{\partial}{\partial z_j^{(n_l)}} J(\mathbf{W}, b; \mathbf{x}, y) \cdot \frac{\partial z_j^{(n_l)}}{\partial z_i^{(n_l-1)}}$$

$$= \sum_j \delta_j^{(n_l)} \cdot \frac{\partial}{\partial z_i^{(n_l-1)}} \left[\sum_k W_{jk}^{(n_l-1)} f\left(z_k^{(n_l-1)}\right) \right]$$

$$= \sum_j \delta_j^{(n_l)} \cdot \frac{\partial}{\partial z_i^{(n_l-1)}} \left[W_{ji}^{(n_l-1)} f\left(z_i^{(n_l-1)}\right) \right]$$

$$= \sum_j \delta_j^{(n_l)} \cdot W_{ji}^{(n_l-1)} \cdot f'\left(z_i^{(n_l-1)}\right)$$

$$= f'\left(z_i^{(n_l-1)}\right) \cdot \sum_j \left(W_{ji}^{(n_l-1)} \cdot \delta_j^{(n_l)} \right)$$



上式可以一般化地表示为: $\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right) f'\left(z_i^{(l)}\right)$

反向传播算法

- 反向传播
  - 根据误差计算网络参数的梯度

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(\mathbf{W}, b; \mathbf{x}, y) = a_j^{(l)} \delta_i^{(l+1)}$$

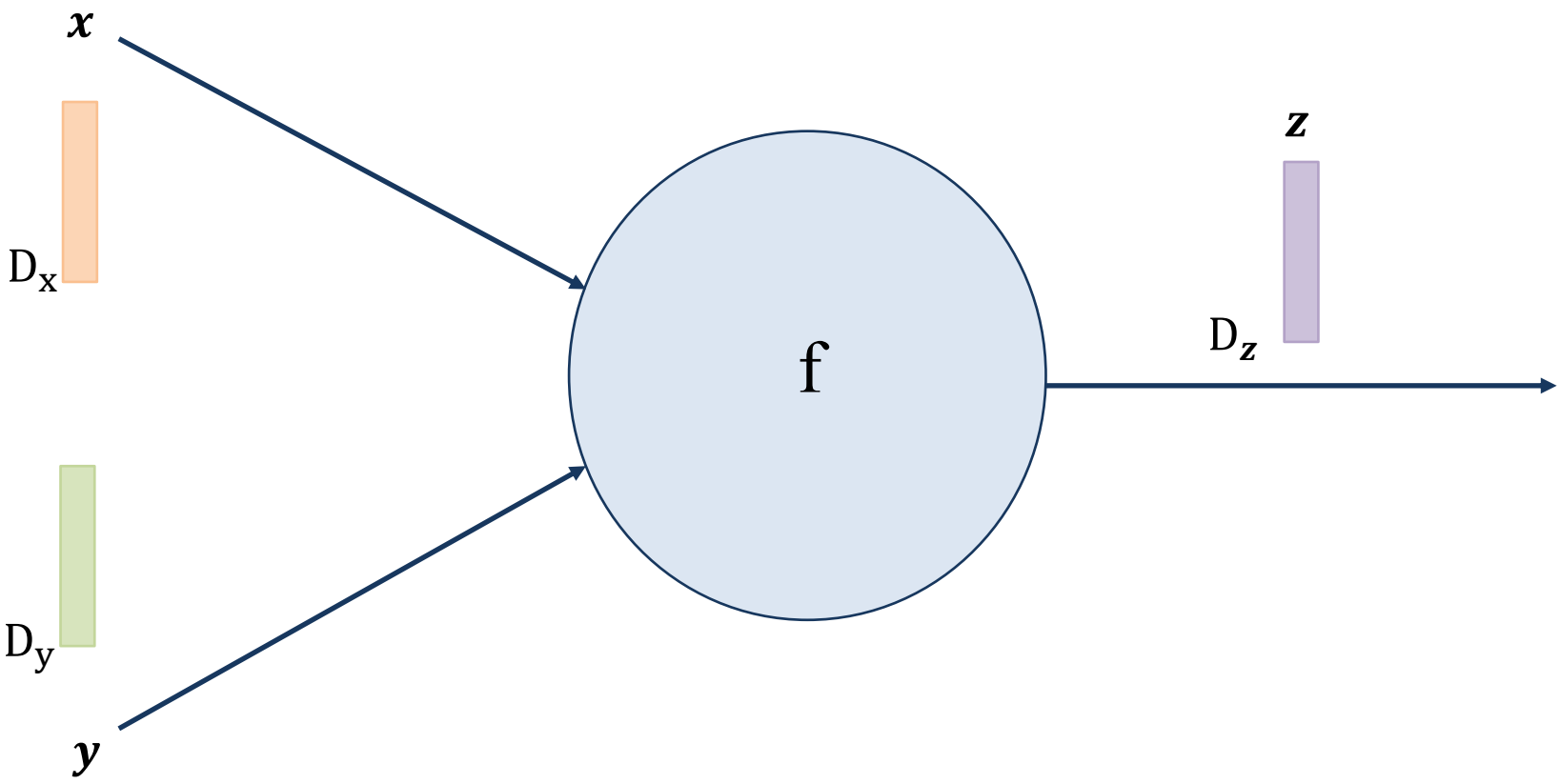
$$\frac{\partial}{\partial b_i^{(l)}} J(\mathbf{W}, b; \mathbf{x}, y) = \delta_i^{(l+1)}$$

  - 使用梯度下降进行参数更新

- 重复前向和后向传播过程，直到达到停止条件

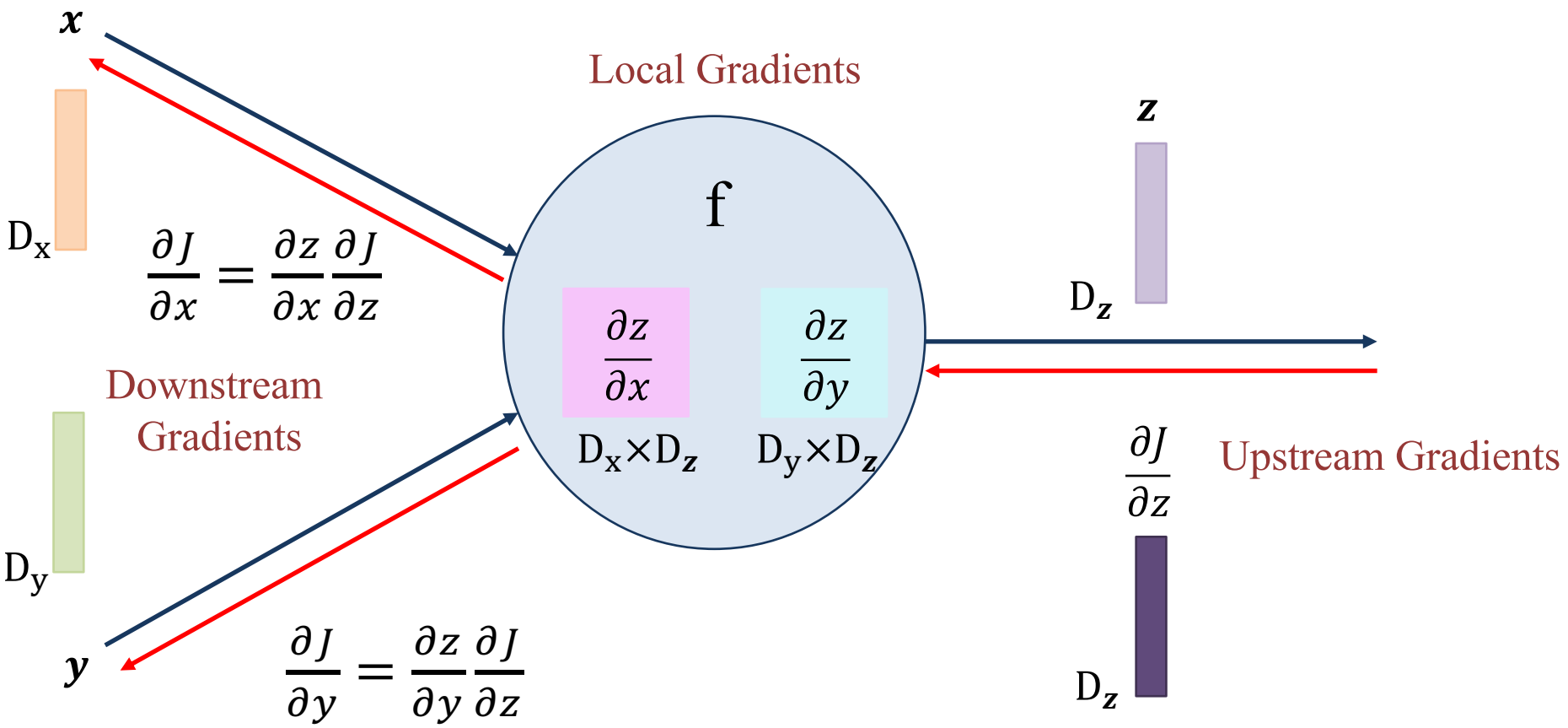
反向传播算法

- 计算图



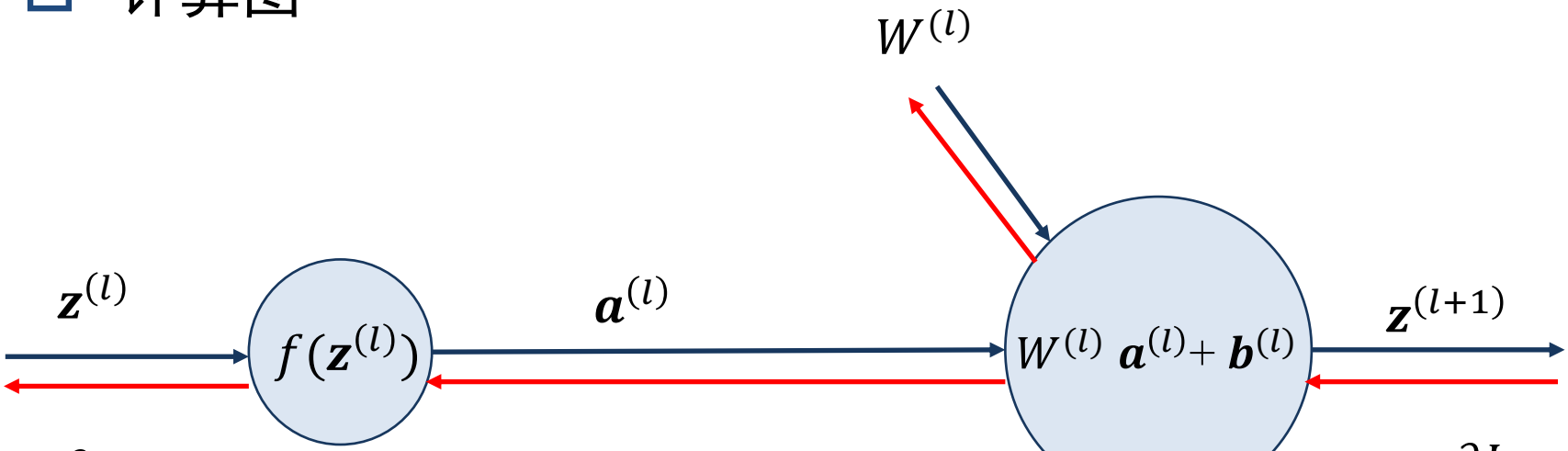
反向传播算法

计算图


x
Local Gradients
z
f
D_x
$\frac{\partial J}{\partial x} = \frac{\partial z}{\partial x}\frac{\partial J}{\partial z}$
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
D_z
Downstream Gradients
$D_x \times D_z$
$D_y \times D_z$
$\frac{\partial J}{\partial z}$
Upstream Gradients
D_y
y
$\frac{\partial J}{\partial y} = \frac{\partial z}{\partial y}\frac{\partial J}{\partial z}$
D_z

反向传播算法

计算图



$$\frac{\partial J}{\partial \boldsymbol{z}^{(l)}} = \frac{\partial \boldsymbol{a}^{(l)}}{\partial \boldsymbol{z}^{(l)}} \frac{\partial J}{\partial \boldsymbol{a}^{(l)}} = diag(f'(\boldsymbol{z}^{(l)})) \frac{\partial J}{\partial \boldsymbol{a}^{(l)}}$$

$$\frac{\partial J}{\partial \boldsymbol{a}^{(l)}} = \frac{\partial \boldsymbol{z}^{(l+1)}}{\partial \boldsymbol{a}^{(l)}} \frac{\partial J}{\partial \boldsymbol{z}^{(l+1)}} = W^{(l)\,T}\ \boldsymbol{\delta}^{(l+1)}$$

$$\frac{\partial J}{\partial \boldsymbol{z}^{(l+1)}} = \boldsymbol{\delta}^{(l+1)}$$

神经网络优化算法

- 梯度下降法

- 梯度下降法的改进
  - Momentum
  - AdaGrad
  - RMSProp
  - Adam

梯度下降法

梯度的概念

- 梯度的定义
  - 梯度是一个矢量
  - 梯度对应的方向上的方向导数最大
  - 梯度的大小即为最大的方向导数的值

$$gradf(x_0, x_1, \dots, x_n) = \left(\frac{\partial f}{\partial x_0}, \dots, \frac{\partial f}{\partial x_j}, \dots, \frac{\partial f}{\partial x_n}\right)$$

- 梯度下降法：函数沿梯度方向有最大的变化率，优化目标损失函数时，根据负梯度方向进行
- 收敛性：在凸问题上可保证收敛到全局最优解，在非凸问题上可能收敛到局部最优解

梯度下降法

- 梯度下降

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \mathcal{L}(\theta_t)$$

  - η为学习率

- 随机梯度下降

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \mathcal{L}(\theta_t; x^i, y^i)$$

- 小批量随机梯度下降

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \mathcal{L}(\theta_t; x^{i:i+n}, y^{i:i+n})$$

梯度下降法的改进

- Momentum
  - 改善SGD方法中的高方差振荡导致的收敛不稳定问题
  - 使用最近一段时间内梯度的加权平均值，加强当前更新时在相关方向的梯度下降，削弱在振荡方向的梯度下降

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla_\theta \mathcal{L}(\theta)$$

$$\theta = \theta - v_t$$

式中γ为上一步更新向量对当前步骤的影响，通常取0.9

当我们在之前一段时间一直趋向于从某个方向往下走，而当前时刻梯度告诉我们要进行很大的变化时，我们通常会有所迟疑，并根据之前的经验与当前时刻的情况作出调整。

梯度下降法的改进

- AdaGrad
  - 自适应地调整每个参数的学习率

计算梯度：$g = \nabla_\theta \mathcal{L}(\theta)$

累计平方梯度：$r = r + g \odot g$

计算更新量：$\Delta\theta = -\frac{\eta}{\epsilon + \sqrt{r}} \odot g$

参数更新：$\theta = \theta + \Delta\theta$

优势：不需要手工调整学习率

劣势：从训练开始时积累梯度平方会导致有效学习率过早和过量的减小（学习率过快接近0）

式中r为梯度累计变量，训练开始时初始化为0
ϵ为一个极小的常数

梯度下降法的改进

- RMSProp
  - 在计算累计平方梯度时加入关于历史信息的衰减系数（平方梯度的指数衰减平均值）

计算梯度：$g = \nabla_\theta \mathcal{L}(\theta)$

平方梯度的指数衰减平均值：$r = \rho \cdot r + (1-\rho) \cdot g \odot g$

计算更新量：$\Delta\theta = -\frac{\eta}{\epsilon + \sqrt{r}} \odot g$

参数更新：$\theta = \theta + \Delta\theta$

Hinton建议历史信息衰减系数ρ取0.9，学习率取0.001

梯度下降法的改进

- Adam（Adaptive Moment Estimation）
  - 不仅记录平方梯度的指数衰减平均值，还记录先前梯度的指数衰减平均值，在RMSProp的基础上加入了Momentum的思想

计算梯度：$g = \nabla_\theta \mathcal{L}(\theta)$

计算指数衰减平均值：$m_t = \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t$

$v_t = \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t \odot g_t$

一阶矩和二阶矩估计：$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$　　$\hat{v}_t = \frac{v_t}{1-\beta_2^t}$

参数更新：$\theta_{t+1} = \theta_t - \frac{\eta}{\epsilon + \sqrt{\hat{v}_t}} \hat{m}_t$

通常β_1取0.9，β_2取0.99

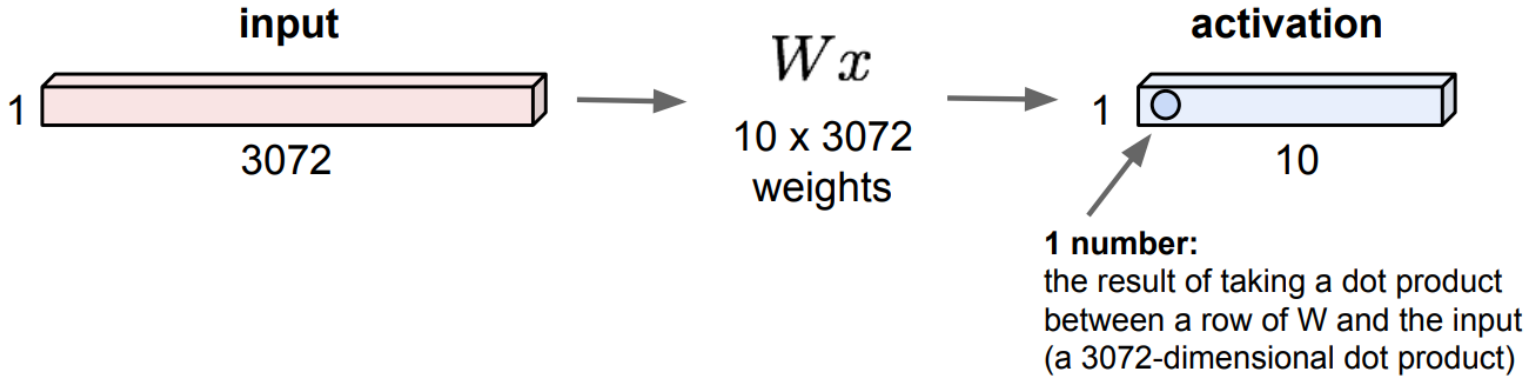
深度学习基础

- 深度前馈网络
- 卷积神经网络
  - 卷积层
  - 汇聚层
  - 归一化层
  - 现代经典卷积网络
- 循环神经网络
- Transformer网络

卷积神经网络

- 全连接层回顾
  - 在全连接下，输出层的每一个值和输入的所有值都有关系
  - 存在问题：当输入层和输出层维度较高，则参数规模庞大

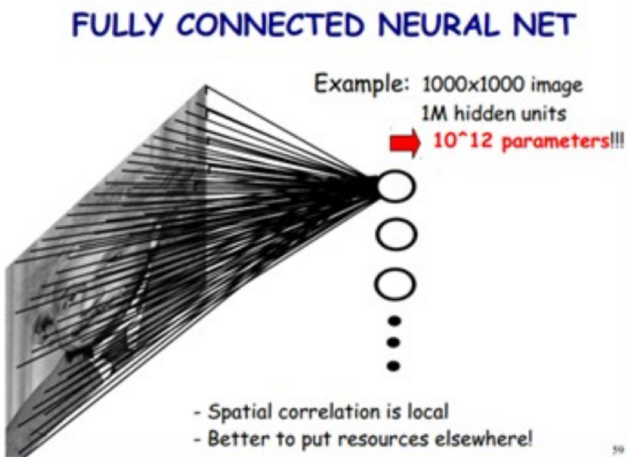
32x32x3 image -> stretch to 3072 x 1

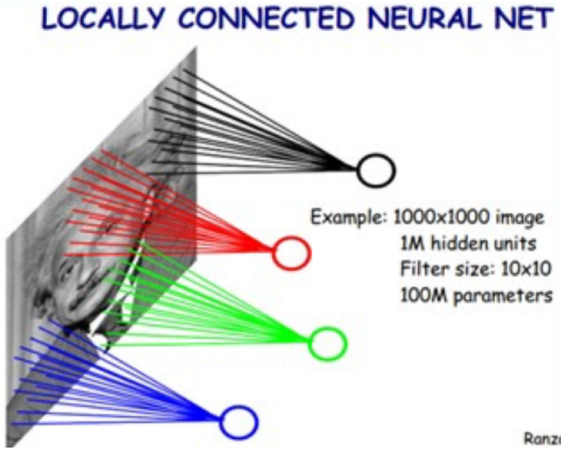


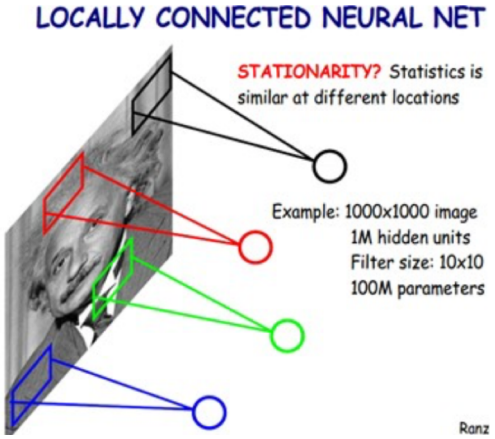
Slide credit: Stanford CS 231n

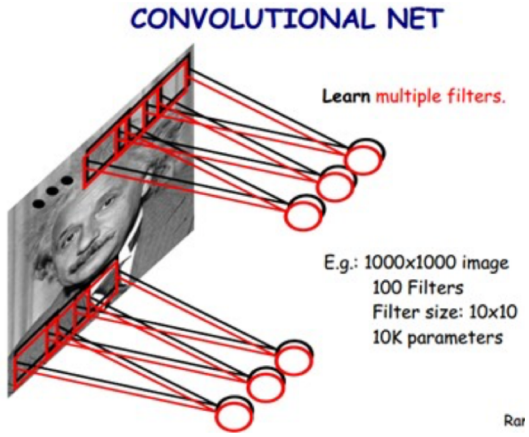
卷积神经网络

- 局部感受野（local receptive field）
- 权值共享（weight sharing）





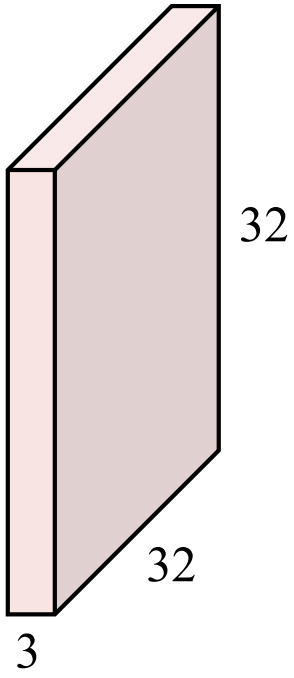




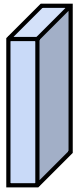
卷积神经网络

卷积层

32x32x3 image



5x5x3 filter

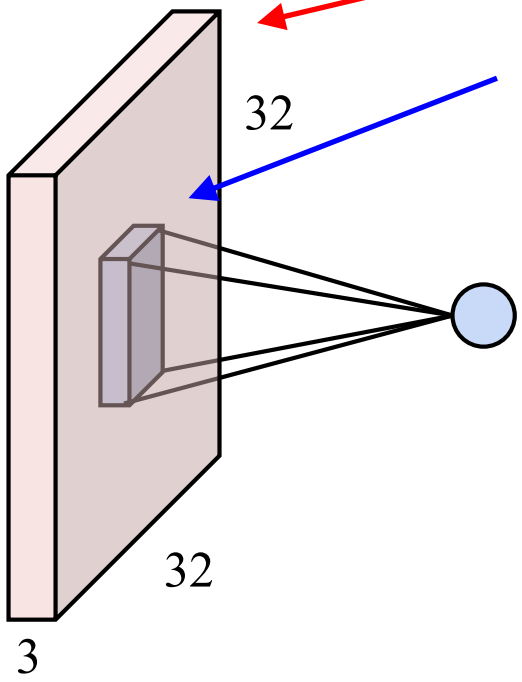


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Slide credit: Stanford CS 231n

卷积神经网络

卷积层



32x32x3 image

5x5x3 filter w

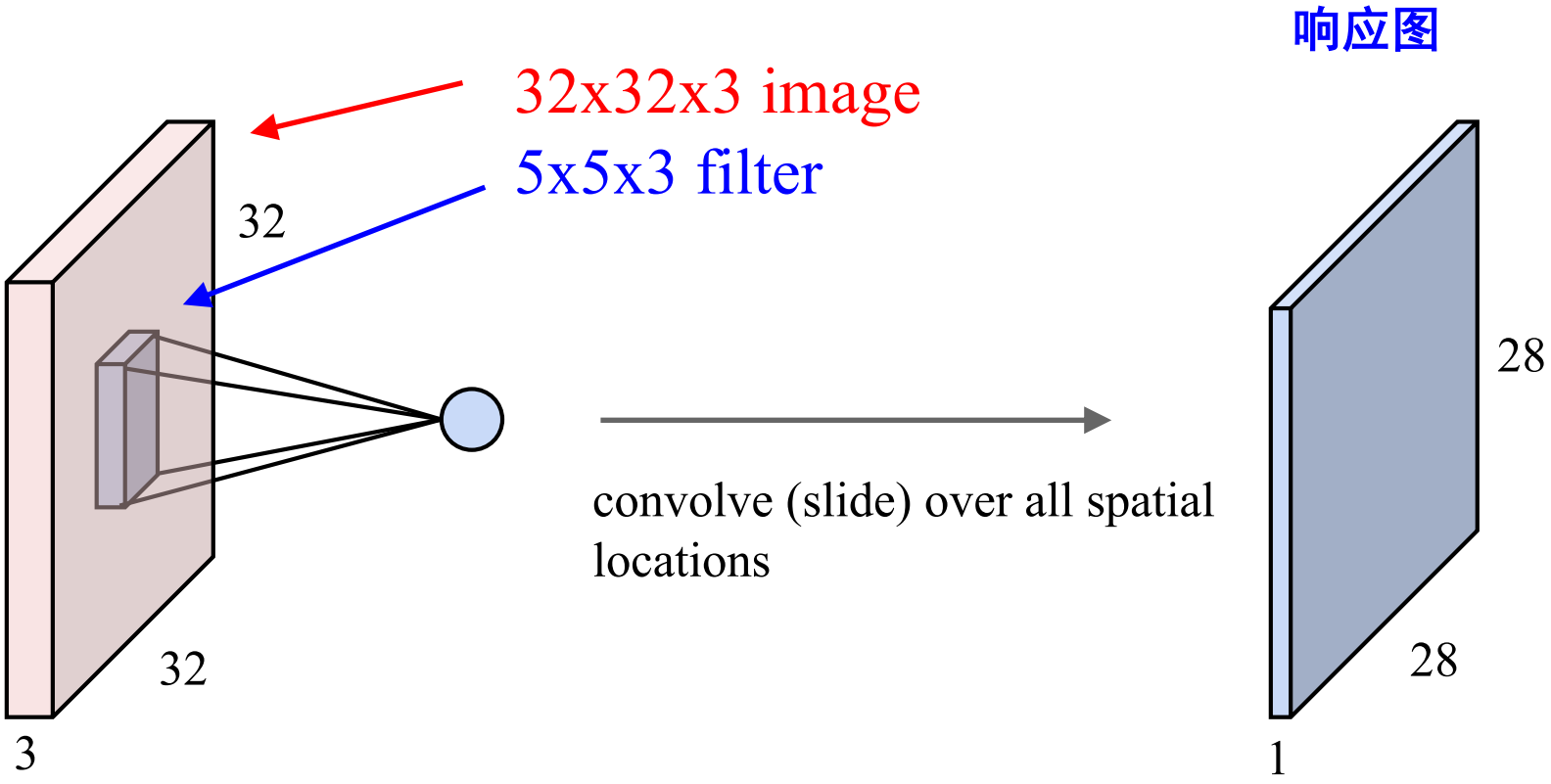
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Slide credit: Stanford CS 231n

卷积神经网络

□ 卷积层



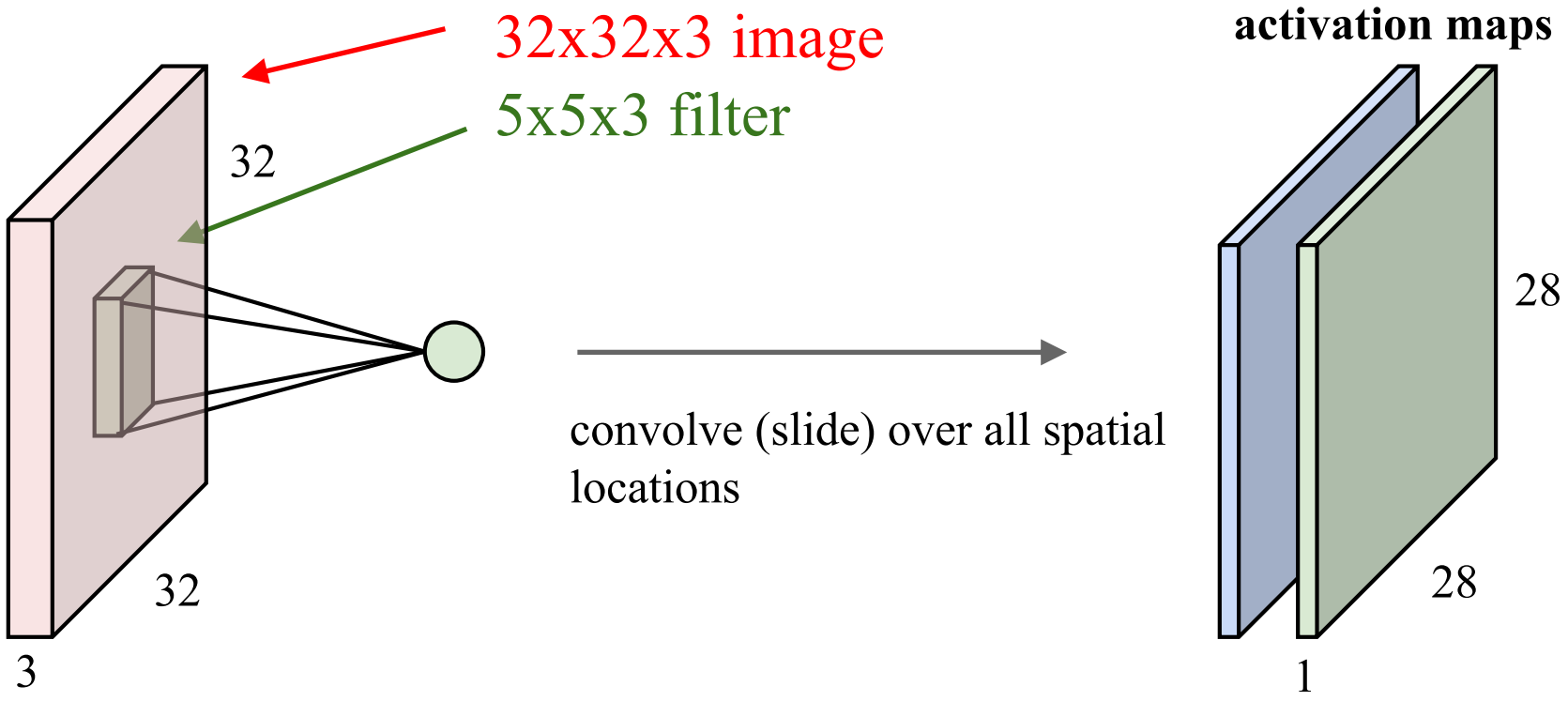
Translation Invariance!

Slide credit: Stanford CS 231n

卷积神经网络

□ 卷积层

考虑第二个滤波器（绿色）

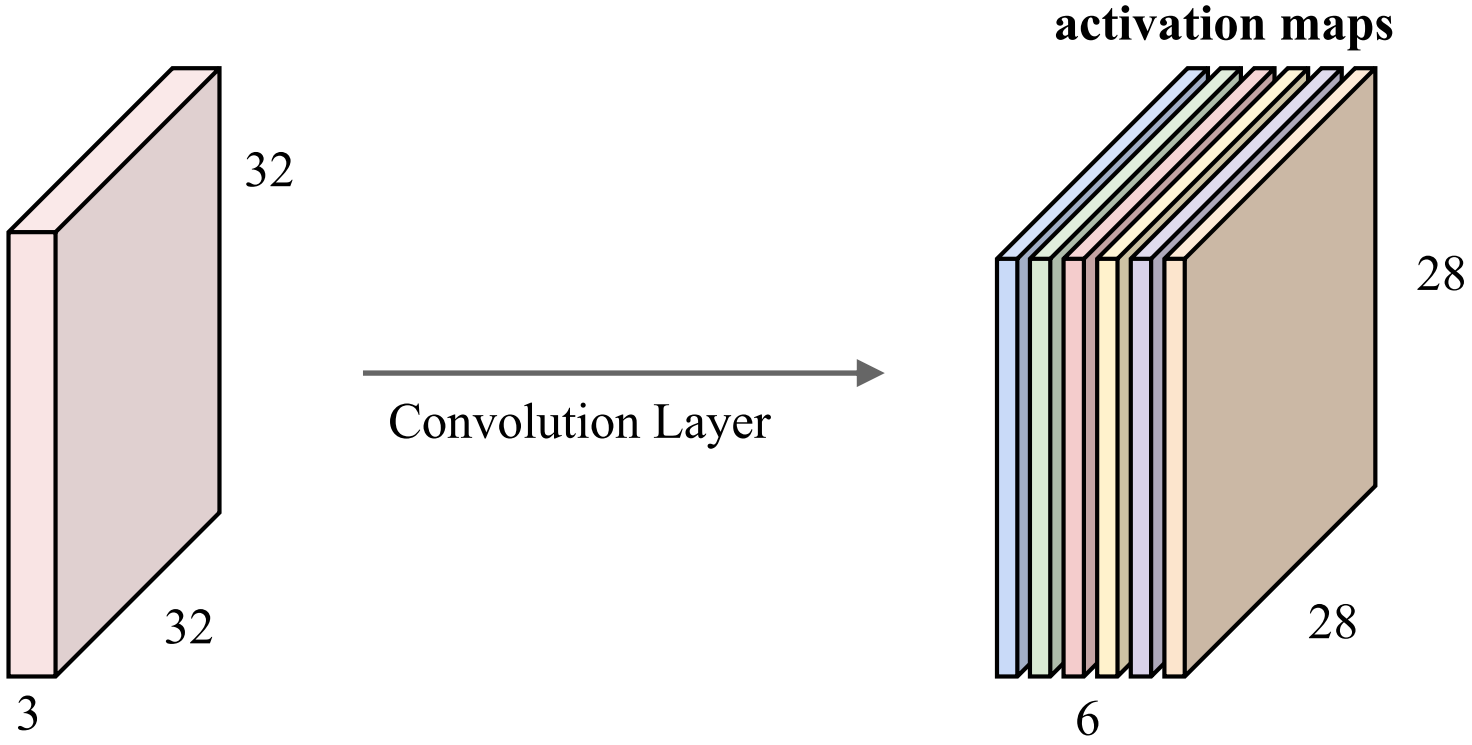


Slide credit: Stanford CS 231n

卷积神经网络

□ 卷积层

如果我们有6个5x5的滤波器, 则可以得到6个不同的响应图



We stack these up to get a “new image” of size 28x28x6!

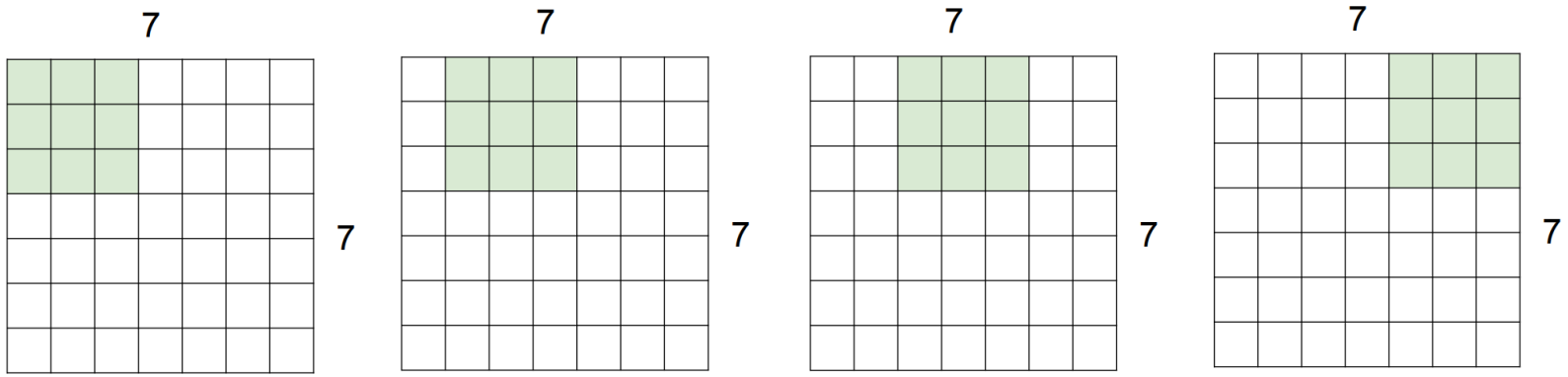
Number of parameters: F^2CK and K biases

Slide credit: Stanford CS 231n

卷积神经网络

卷积操作

7x7 input (spatially)
assume 3x3 filter



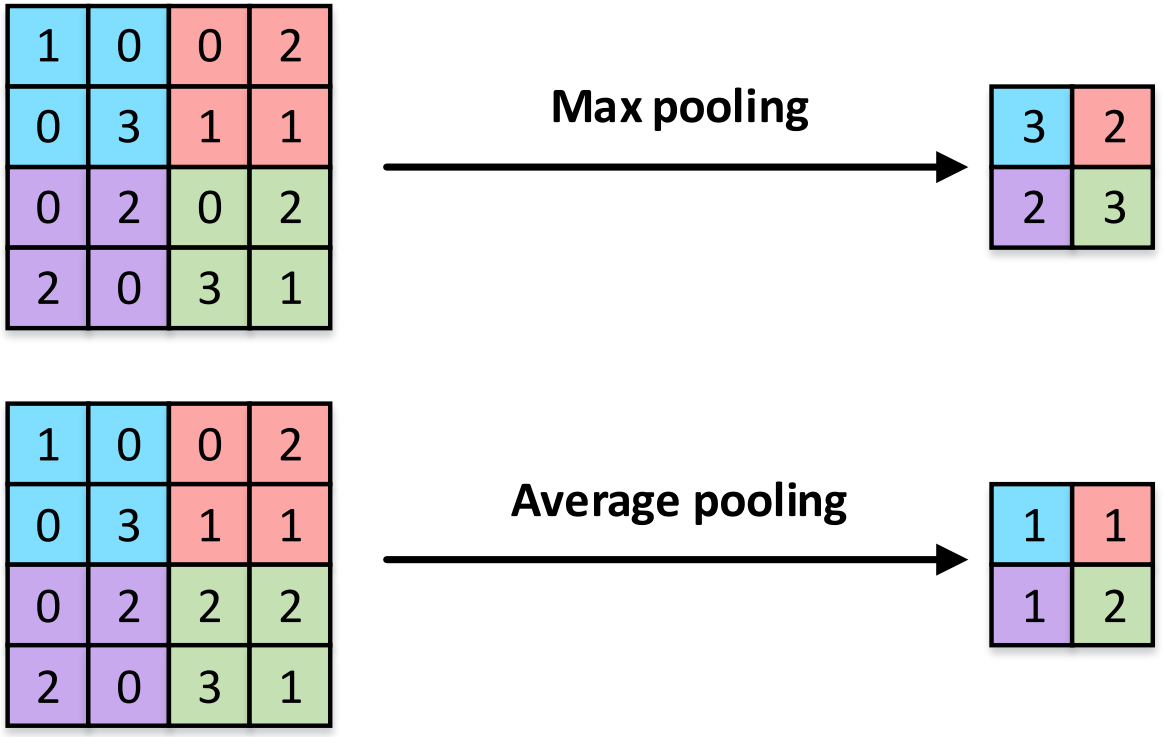
当stride为1，不做padding时，得到5×5的feature map

Output size: (N+2P-F) / stride + 1

Slide credit: Stanford CS 231n

卷积神经网络

- 汇聚层（池化层）

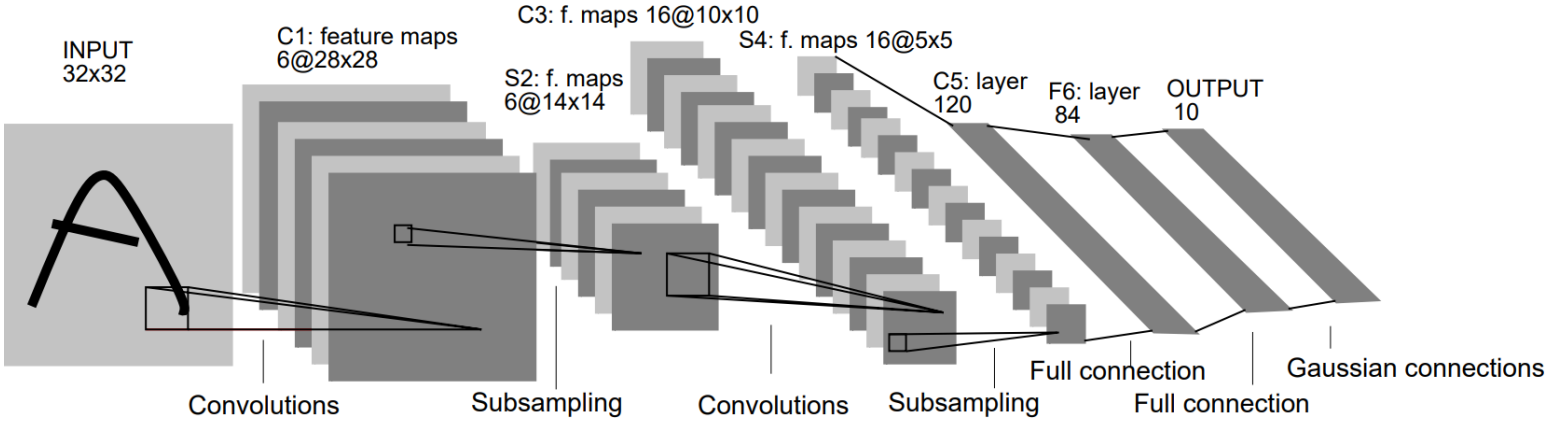


卷积神经网络

□ 实例：LeNet-5

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



与现在常见的卷积神经网络结构不同，LeNet-5的下采样层不是用池化层实现，而是先将2×2的窗口中的4个输入相加，乘以一个可训练参数，再加上一个可训练偏置，输出的结果通过sigmoid激活

- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition”. In Proceedings of the IEEE, 86(11):2278–2324, 1998.

批量归一化

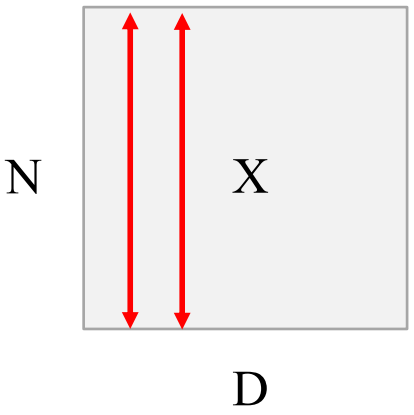
- 问题

  中间层输出数值的不稳定性，给深度神经网络的训练带来困难

- 解决办法

  利用小批量上的均值和标准差，调整中间层输出

- 假设输入$x \in \mathbb{R}^{N \times D}$



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N} \left(x_{i,j} - \mu_j\right)^2$$

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \varepsilon}}$$

$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

γ、β为可学习的参数

批量归一化

- 全连接层的批量归一化

$x: N \times D$

归一化 ↓

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

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

$$y = \frac{(x - \mu)}{\sqrt{\sigma^2 + \varepsilon}}\gamma + \beta$$

- 卷积层的批量归一化

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

归一化 ↓ ↓ ↓

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

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

$$y = \frac{(x - \mu)}{\sqrt{\sigma^2 + \varepsilon}}\gamma + \beta$$

层归一化

- 全连接层的批量归一化

$x: N \times D$

归一化 ↓

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

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

$$y = \frac{(x-\mu)}{\sqrt{\sigma^2+\varepsilon}}\gamma + \beta$$

- 层归一化（RNN, Transformers）

$x: N \times D$

归一化 ↓

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

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

$$y = \frac{(x-\mu)}{\sqrt{\sigma^2+\varepsilon}}\gamma + \beta$$

实例归一化

- 卷积层的批量归一化

$$x：N \times C \times H \times W$$

归一化

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

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

$$y = \frac{(x - \mu)}{\sqrt{\sigma^2 + \varepsilon}}\gamma + \beta$$

- 卷积层的实例归一化

$$x：N \times C \times H \times W$$

归一化

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

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

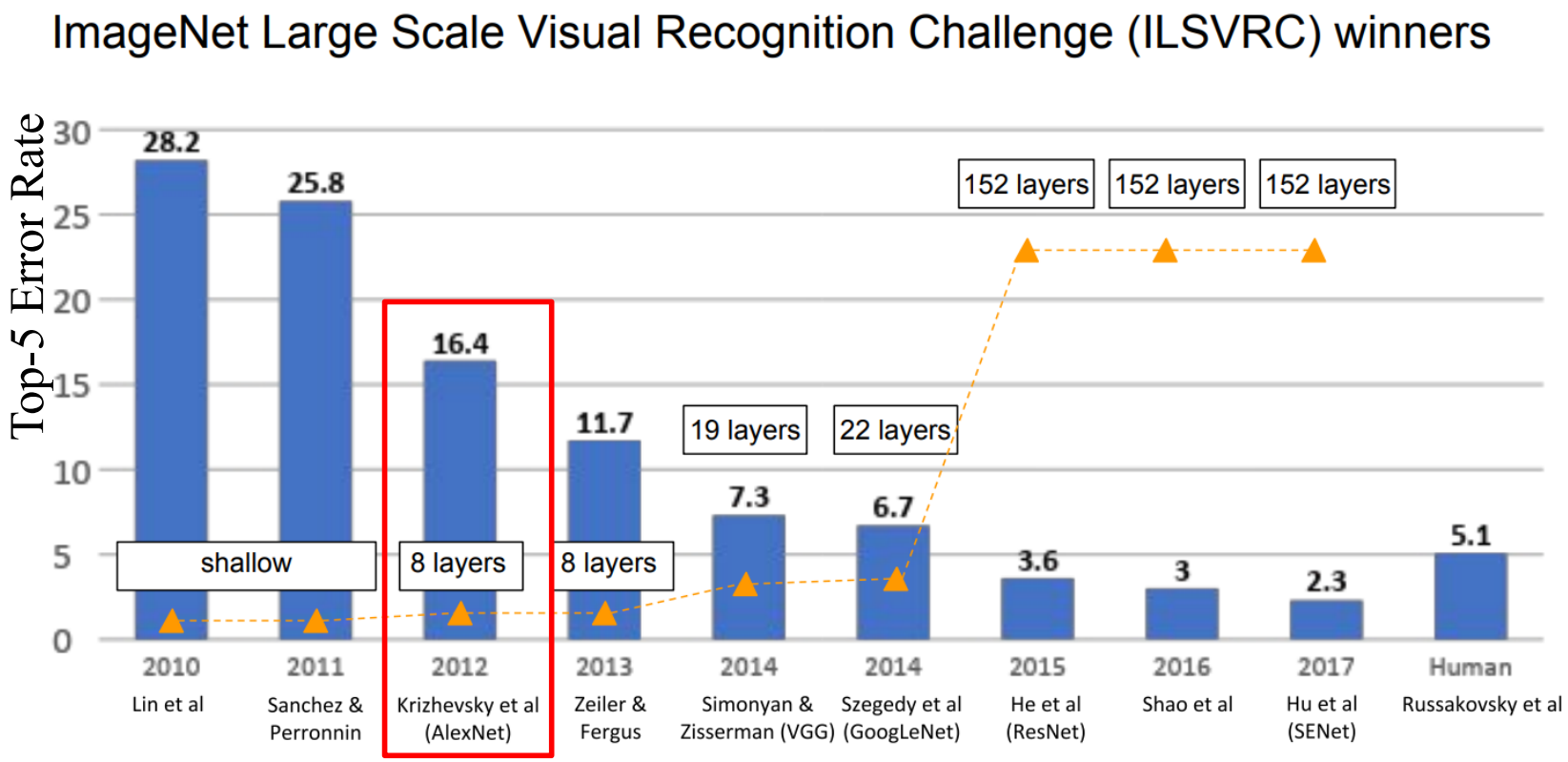
$$y = \frac{(x - \mu)}{\sqrt{\sigma^2 + \varepsilon}}\gamma + \beta$$

现代经典卷积神经网络

- 经典图像分类网络
  - AlexNet
  - VGGNet
  - GoogLeNet
  - ResNet
  - MobileNet

图像分类任务的巨大进展

- 随着深度卷积神经网络的发展，图像分类的准确率得到了极大的提升
  - 核心因素：数据、算力、算法
  - 对人工设计特征造成的巨大冲击

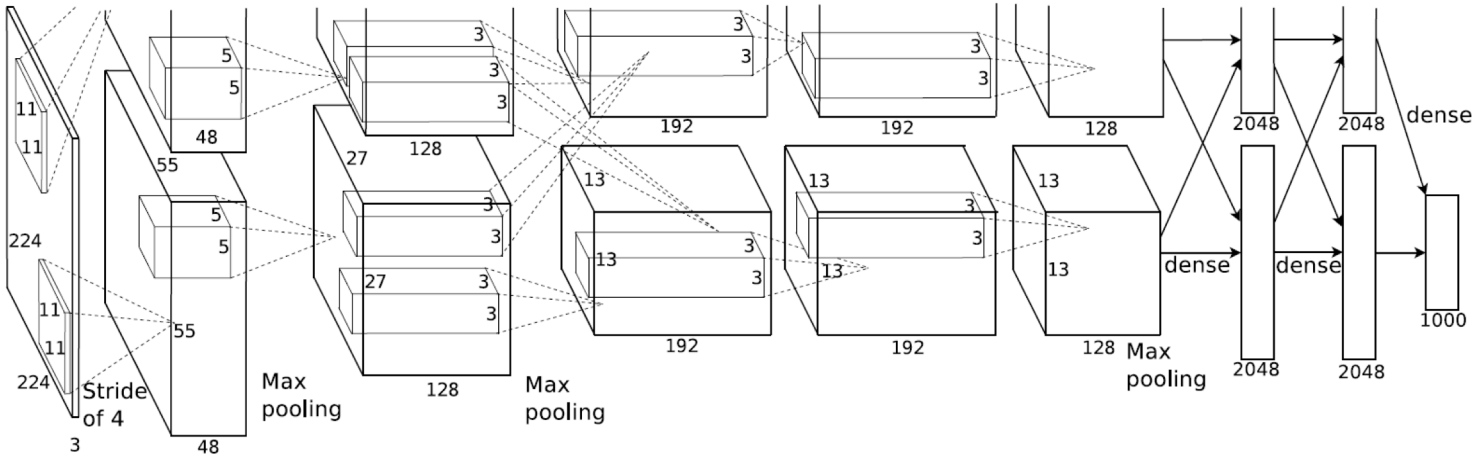


AlexNet

AlexNet

- 首次在ImageNet比赛中使用深度卷积神经网络
- 引入的一些技术被现代卷积神经网络设计一直沿用
  - ✓ ReLU
  - ✓ Data Augmentation
  - ✓ Dropout

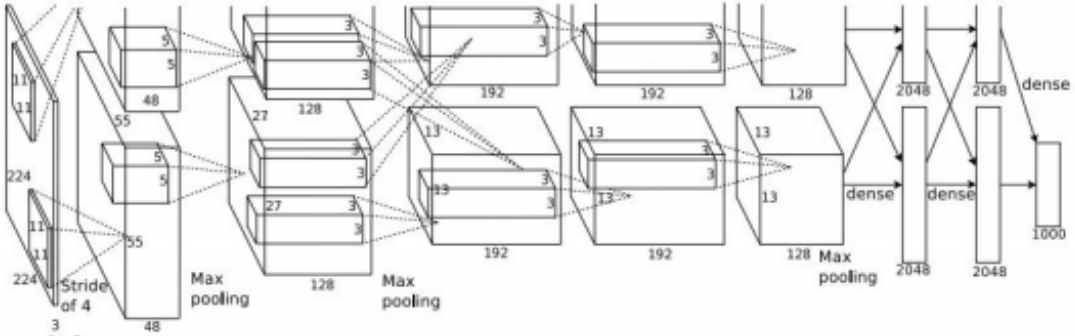
Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8



- A. Krizhevsky, I. Sutskever, and G. Hinton, “ImageNet classification with deep convolutional neural networks”. In NeurIPS, 2012.

AlexNet

- AlexNet详细信息

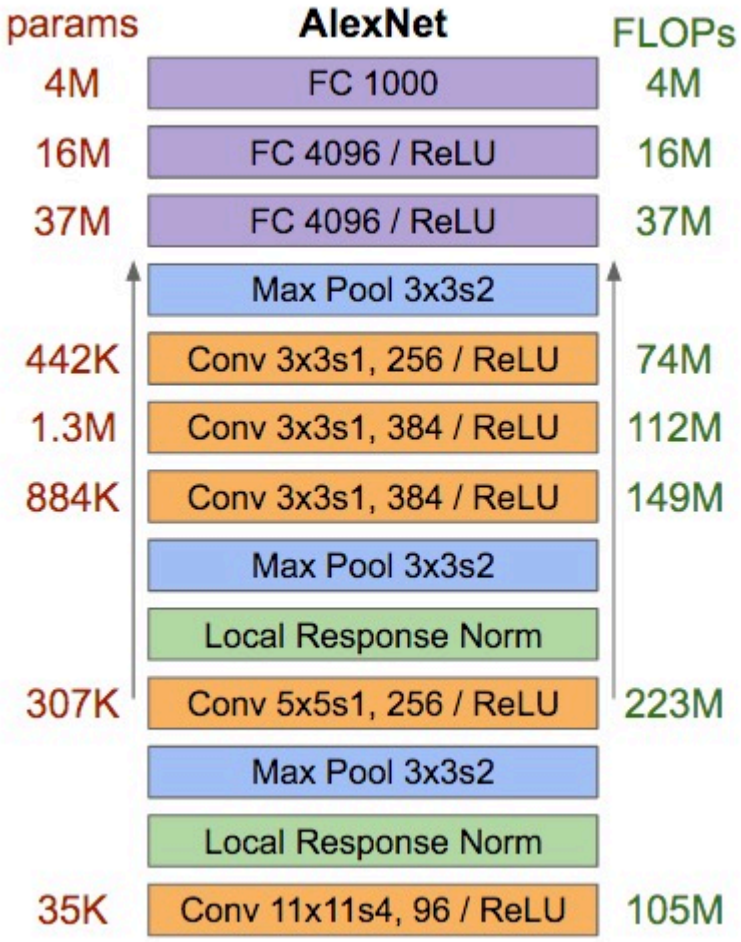


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] 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)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

AlexNet

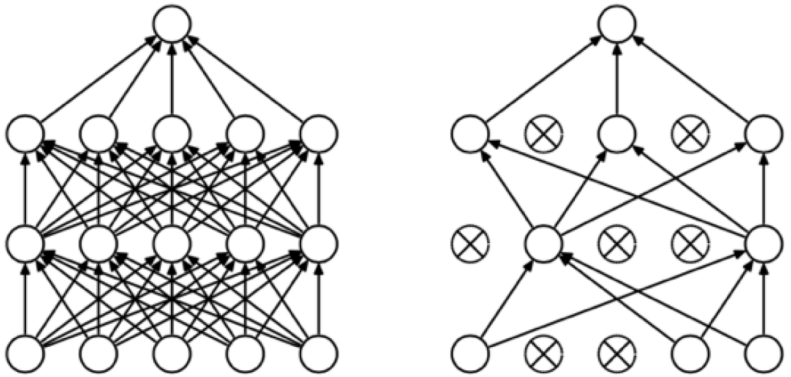
- 参数规模与计算复杂度



AlexNet

- AlexNet中使用的学习策略
  - Weight decay

$$\tilde{E}(w) = E(w) + \frac{\lambda}{2} w^T w$$



(a) Standard Neural Net　　(b) After applying dropout.

  - Data augmentation

    目的：扩充训练样本库

    方法：水平翻转，crop图中部分区域

  - Dropout

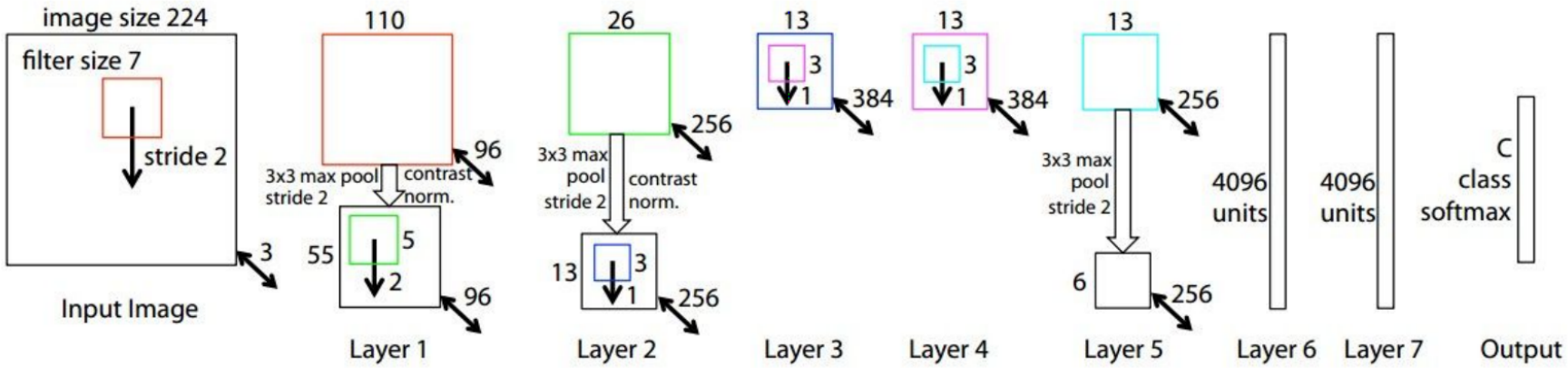
    目的：防止过拟合，相当于指数级倍网络的组合

    方法：在每次训练的迭代中，只会激活部分神经元

AlexNet网络结构改进

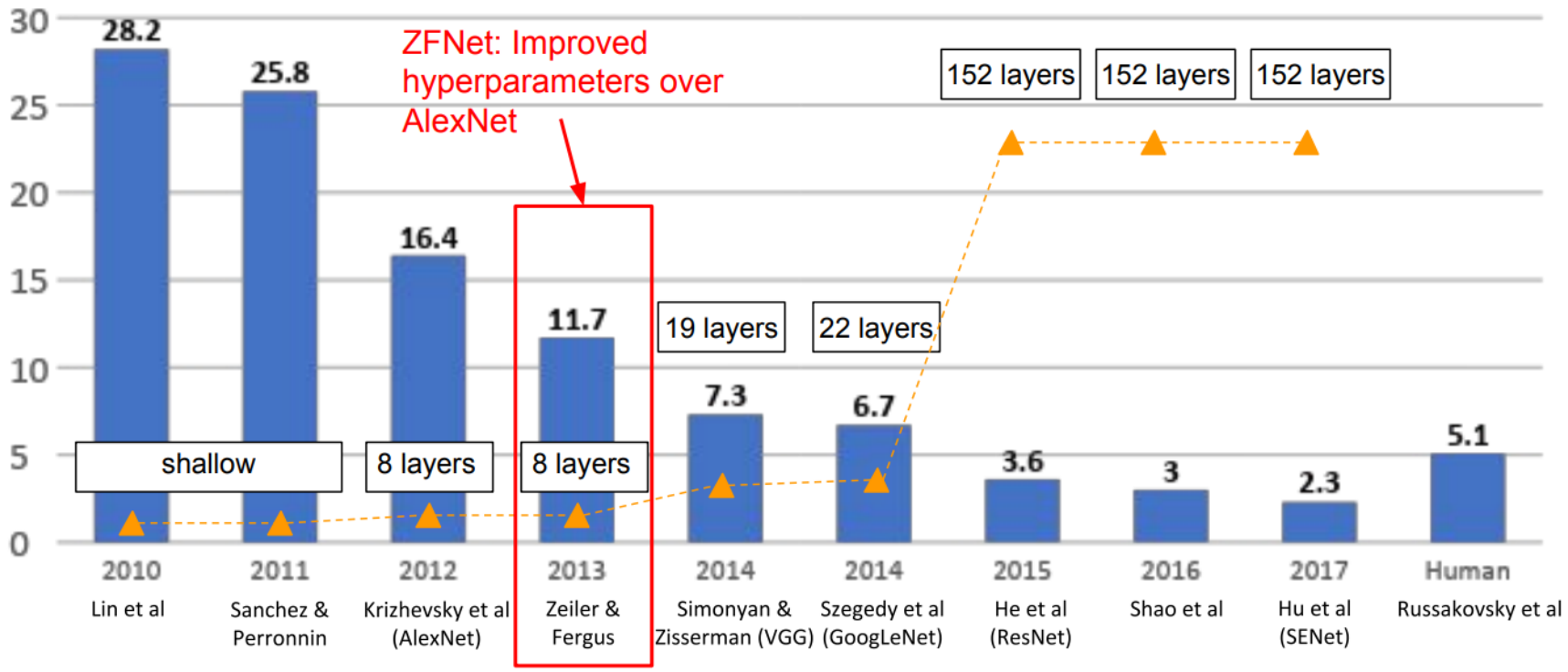
- AlexNet改进的主要方向
  - 更深：增加网络的层数
  - 更宽：增加每一级网络的卷积单元

- ZFNet：微调AlexNet结构参数
  - 将AlexNet中第一个11×11，stride为4的卷积层，换成7×7，stride为2的卷积层
  - 将Conv3,4,5中的channel数增加到512,1024,512

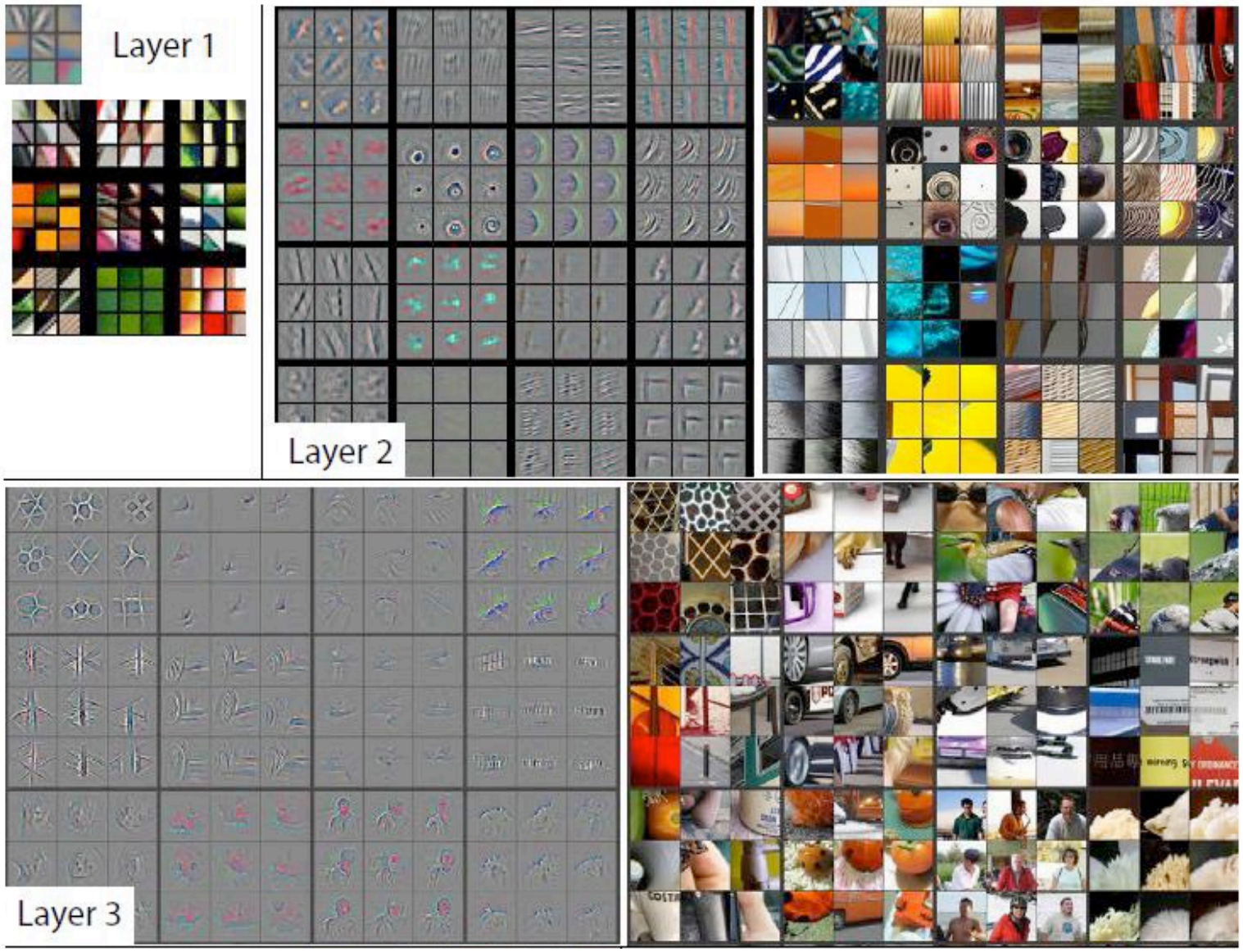


ZFNet

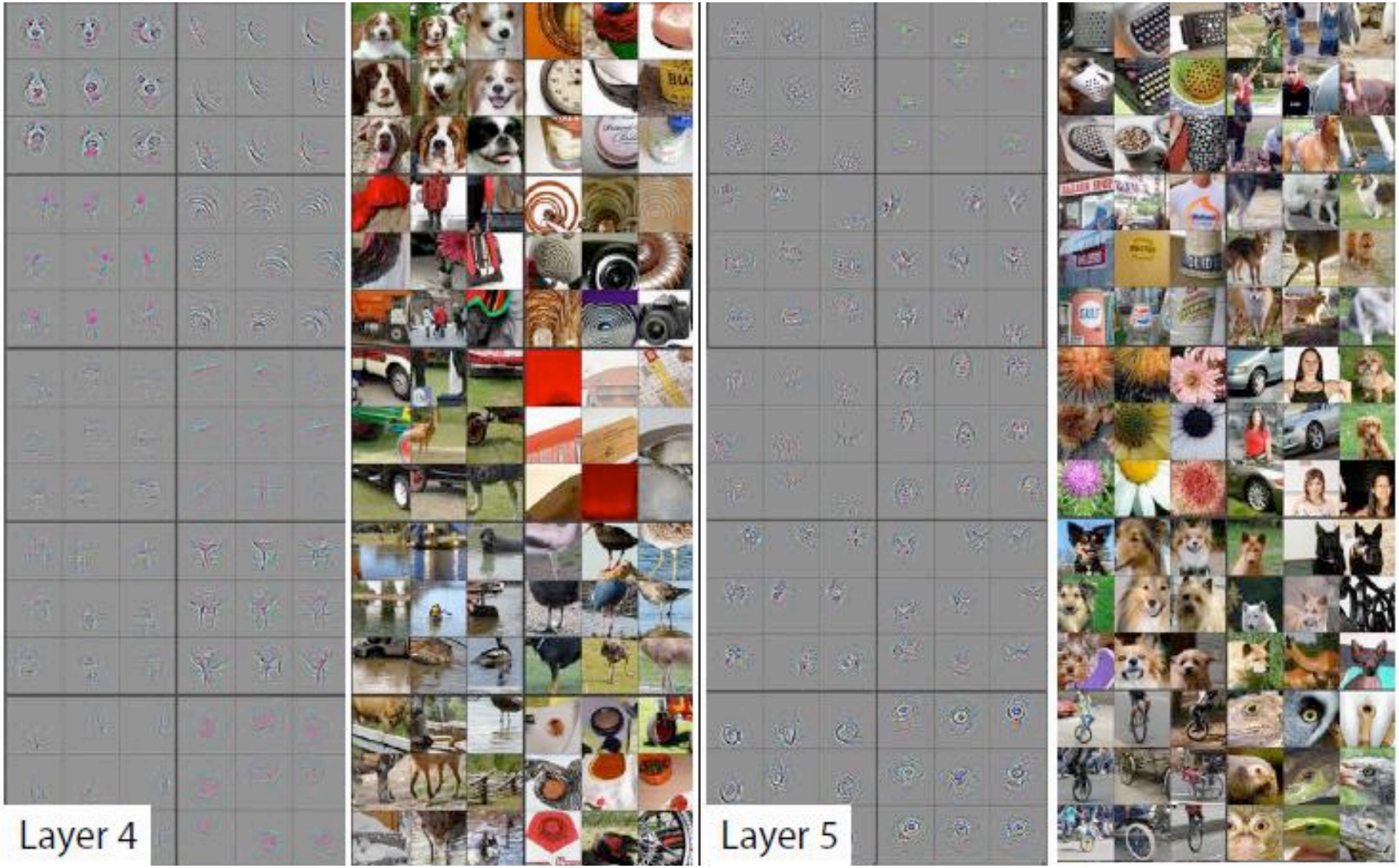
- 超参数的改进在效果上得到了明显的体现



卷积神经网络特征可视化

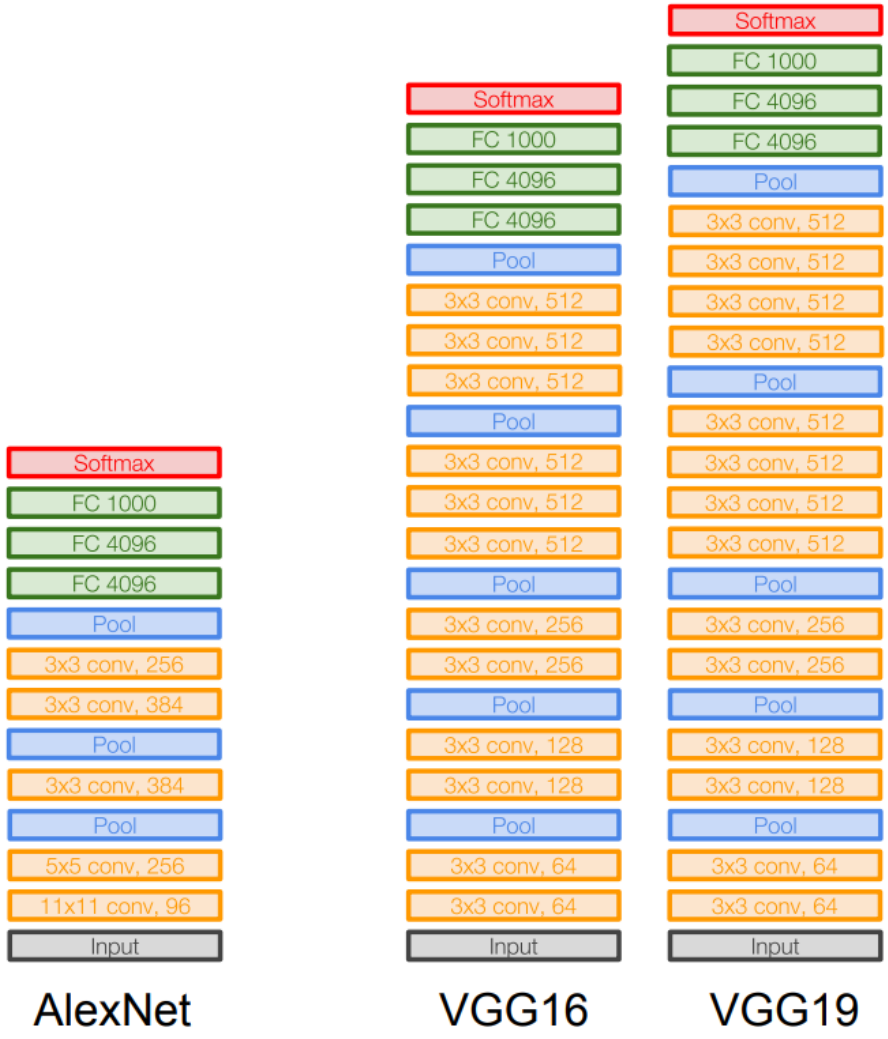


卷积神经网络特征可视化



VGGNet

- 主要思想
  - 更小的卷积核
  - 更深的网络结构
- 与AlexNet的差异
  - 从8层变为16-19层
  - 卷积层大小均为3×3，stride均为1
  - 下采样用2×2的max pooling



- K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In ICLR, 2015.

VGGNet

- 为什么使用更小的卷积层
  - 感受野的等效性
    - ✓ 3个stride为1的3×3的卷积层对应的感受野，等效于1个7×7大小的卷积层
  - 等效感受野的情况下更少的参数（假设输入和输出的channel数量均为1）
    - ✓ 3个3×3的卷积层的参数量：$3\times3^2=27$
    - ✓ 1个7×7的卷积层的参数量：$1\times7^2=49$
  - 更深的结构与更多的非线性激活层带来更好的特征表达的能力

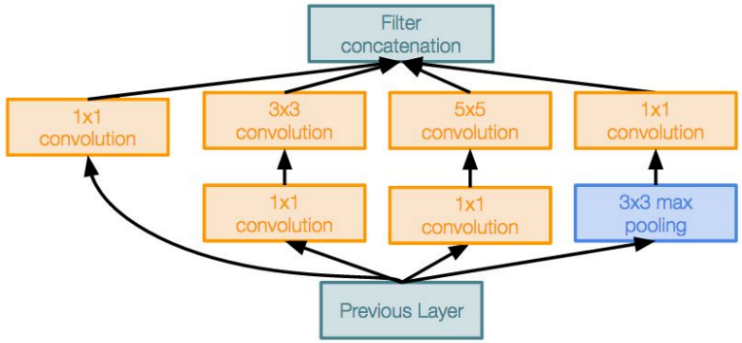
VGGNet

- VGGNet的各种变种（D和E较为常用）

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

GoogLeNet

- 更深的网络结构，更高的计算效率
  - 有22层，远深于AlexNet
  - Inception module（network within a network）
    - ✓ 不同大小的卷积层和池化层使inception结构具备不同的感受野



Inception module

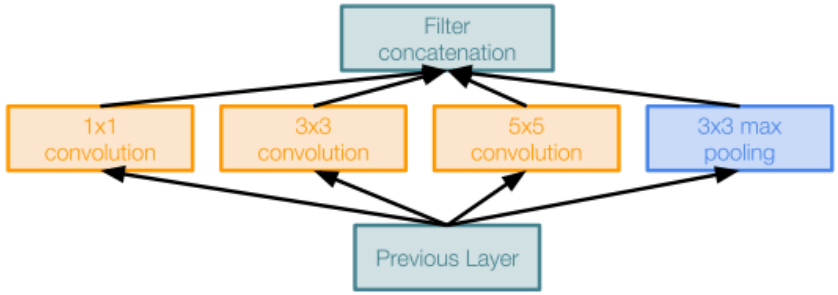
  - 没有全连接层，网络参数总量比AlexNet小12倍

GoogLeNet，VGGNet，AlexNet的对比让大家意识到，网络的学习能力不是主要取决于参数的多少，而是网络的深度

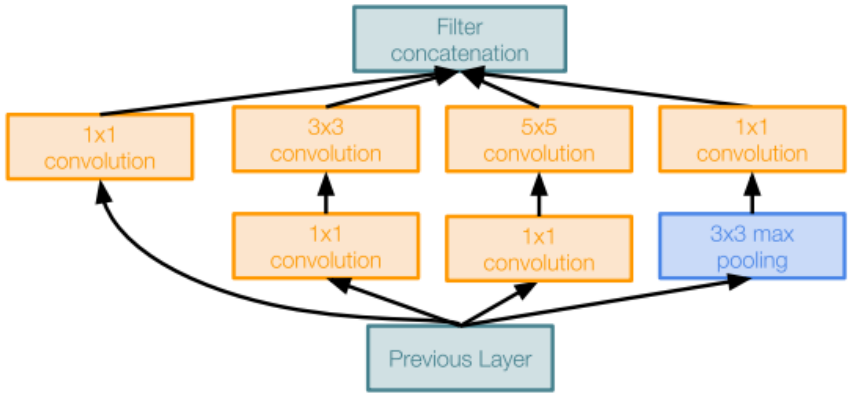
- C. Szegedy, et al., “Going deeper with convolutions”. In CVPR, 2015.

GoogLeNet

- Inception module中的1×1卷积与bottleneck
  - 直接使用下面左图的结构会带了大量的计算量
  - Inception module的实际实现中，先通过1×1卷积进行降维，再通过不同感受野的卷积核，并把结果拼接到一起
  - 这样子先通过降维进行操作，再还原回原来维度的结构我们称为bottleneck



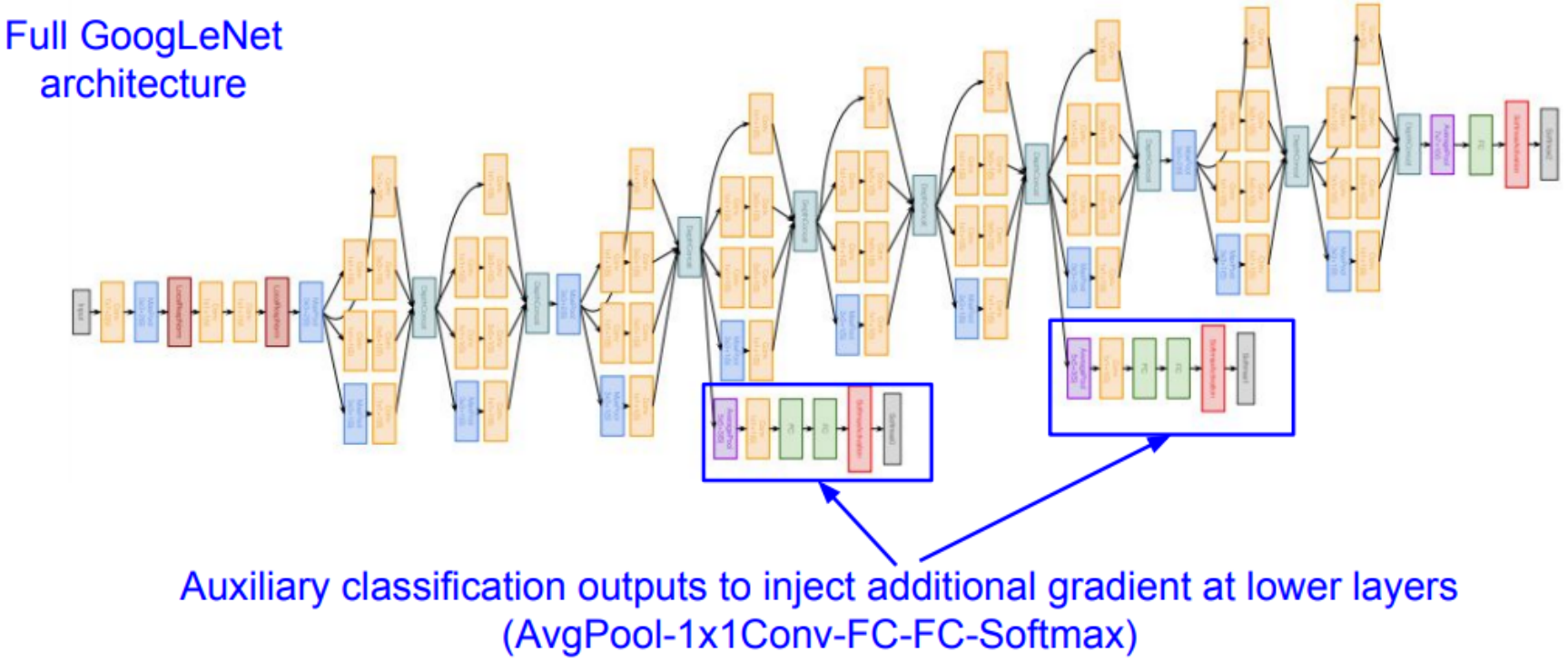
Naive Inception module



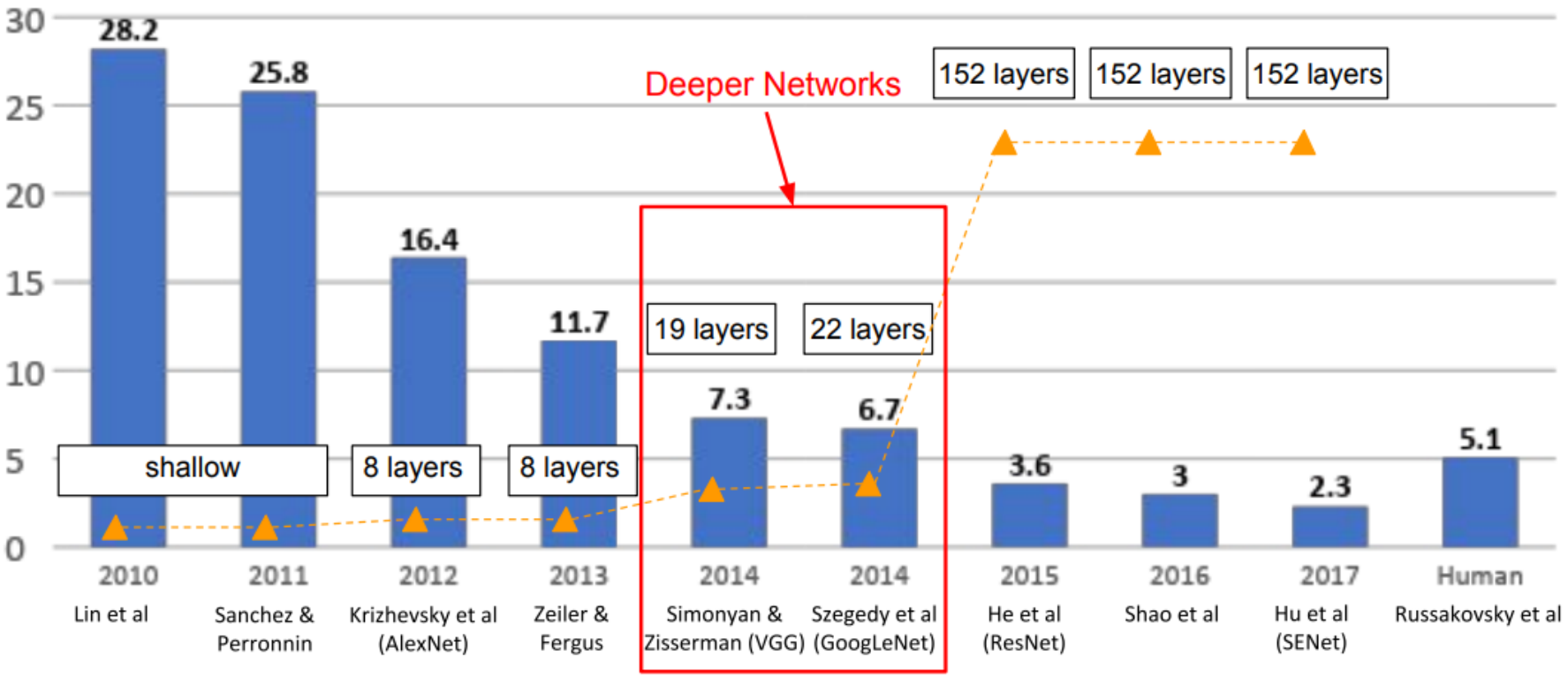
Inception module with dimension reduction

GoogLeNet

- Auxiliary classification head
  - 网络结构太深造成底层难以得到有效的训练，通过加入auxiliary classification head缓解这个问题



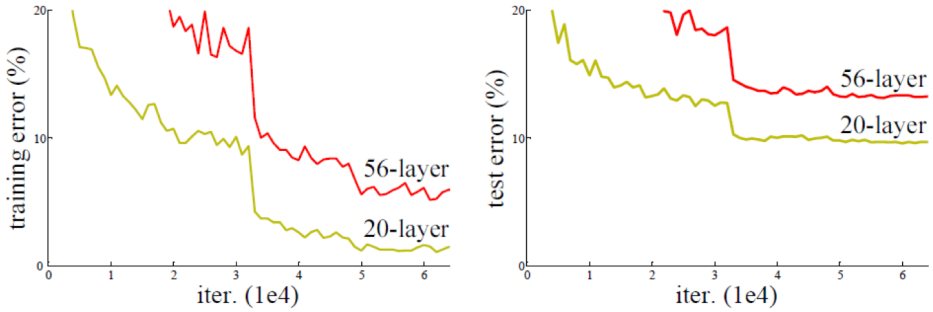
VGGNet与GoogLeNet的性能

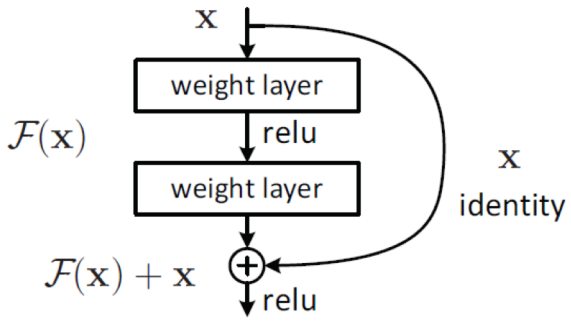


ResNet

- Motivation
  - Simply stacking more layers leads to higher training error
  - Not all systems are similarly easy to optimize
  - Hypothesis

    It is easier to optimize the residual mapping than to optimize the original, unreferenced mapping.





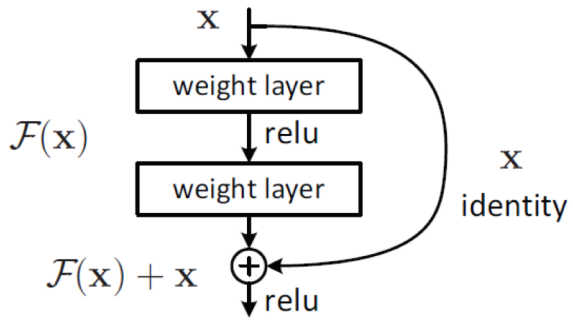
$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + \mathbf{x}.$$

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

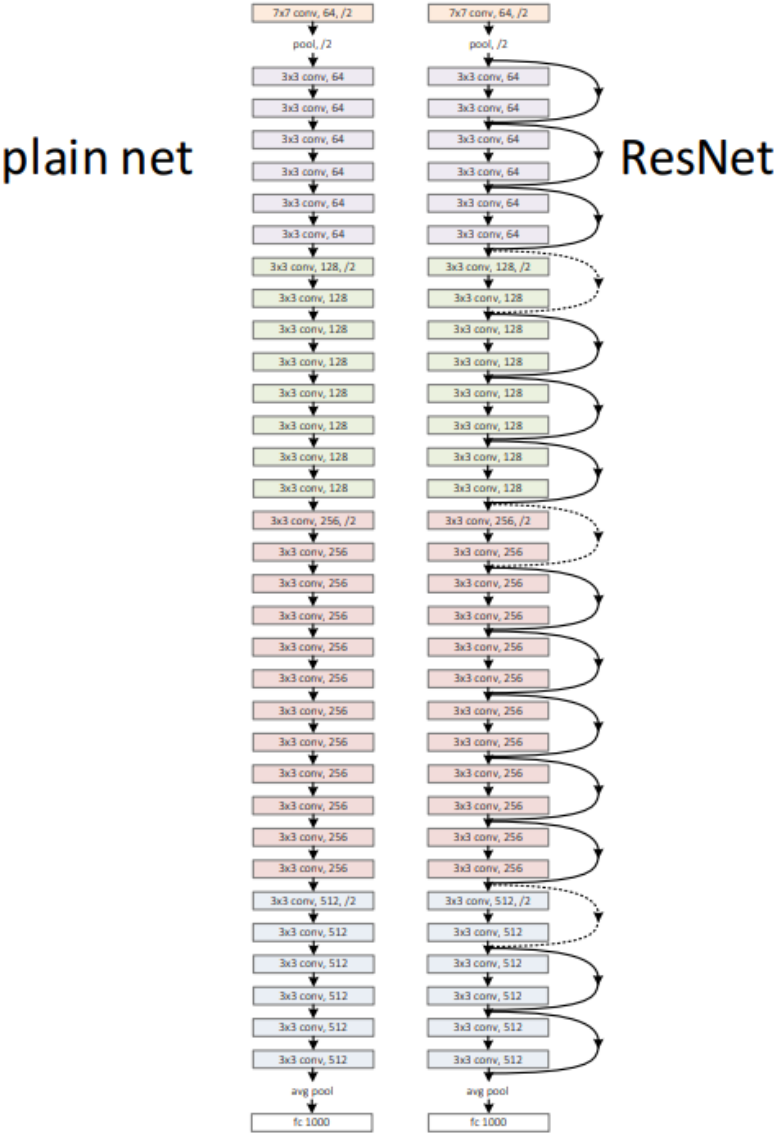
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In CVPR, 2016.

ResNet

- Plain net 与 ResNet
  - ResNet整体结构非常简洁
  - 在不同层之间加入 ”shortcut”



  - 除stem cell（7×7卷积）外均采用3×3卷积层
  - 每当特征的空间分辨率减半时，filter的数量翻倍



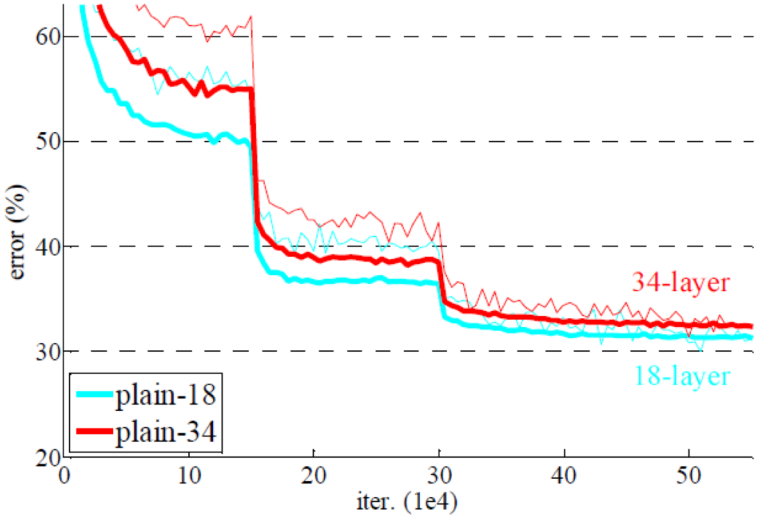
ResNet

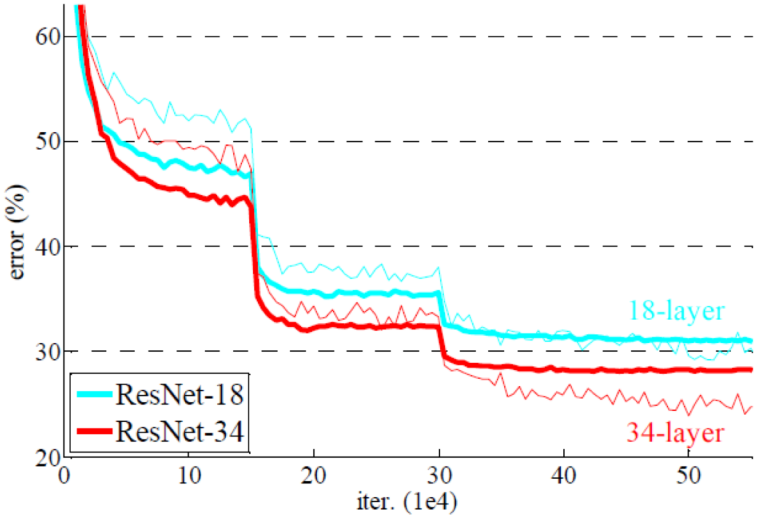
- ResNet不同深度的网络结构

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ResNet

- 在ImageNet上的训练过程





Error rates on ImageNet validation

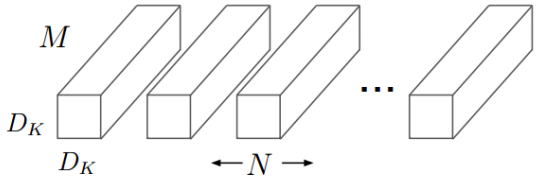
method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Error rates (%) of ensembles

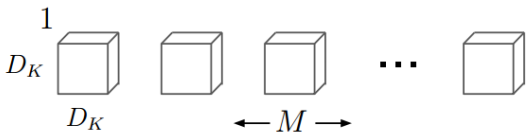
method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

MobileNet

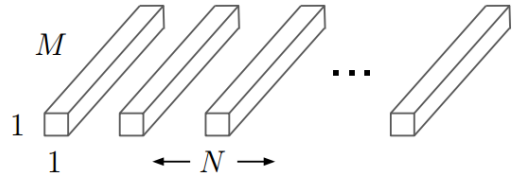
- Goal: *efficiently trade off between latency and accuracy*
  - Build very small, low latency models that can be easily matched to the design requirements for mobile and embedded vision applications
- Depthwise Separable Convolution



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 1. Standard convolution vs separable convolution.

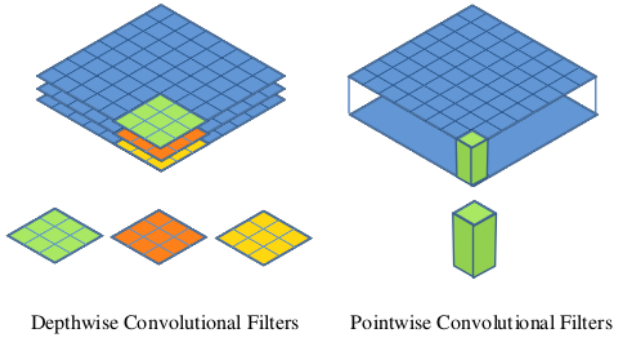


Figure 2. Depthwise separable convolution.

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2}$$

MobileNet

Architecture

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

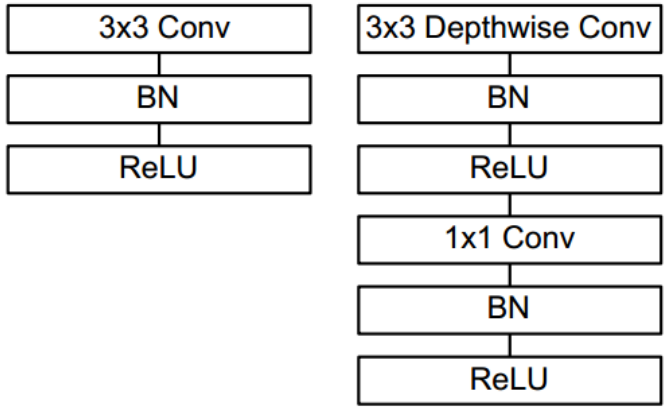


Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.

Table 4. Depthwise Separable vs Full Convolution MobileNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
Conv MobileNet	71.7%	4866	29.3
MobileNet	70.6%	569	4.2

MobileNet

- Performance

Table 8. MobileNet Comparison to Popular Models

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138

Table 9. Smaller MobileNet Comparison to Popular Models

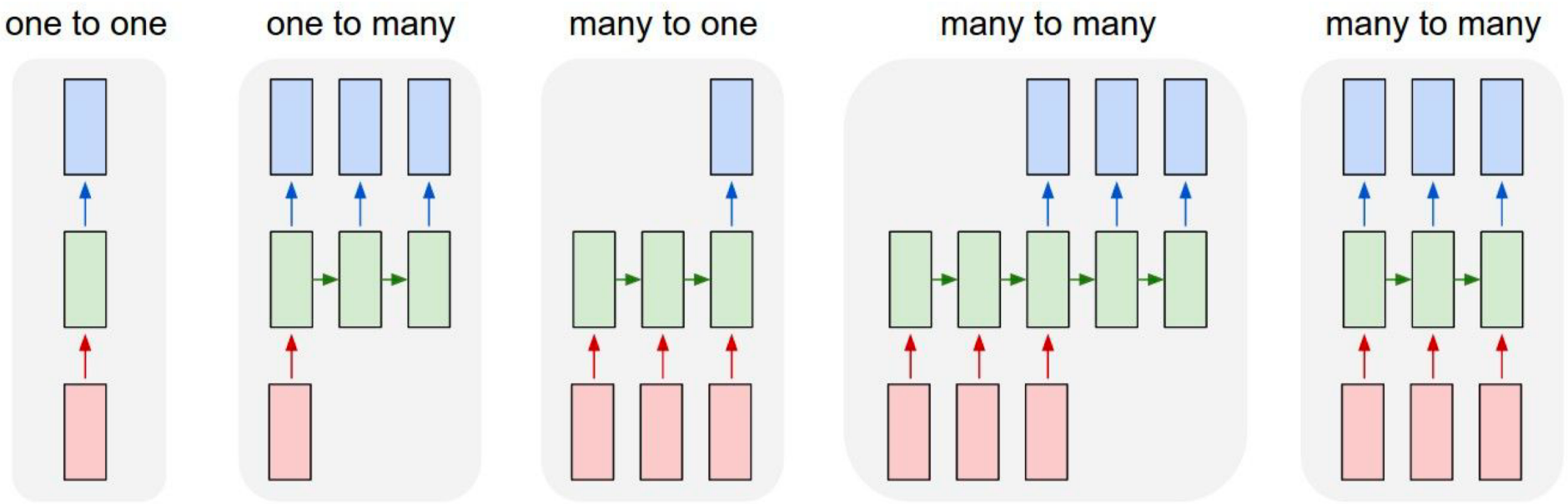
Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

深度学习基础

- 深度前馈网络
- 卷积神经网络
- 循环神经网络
  - RNN
  - LSTM
  - GRU
- Transformer网络

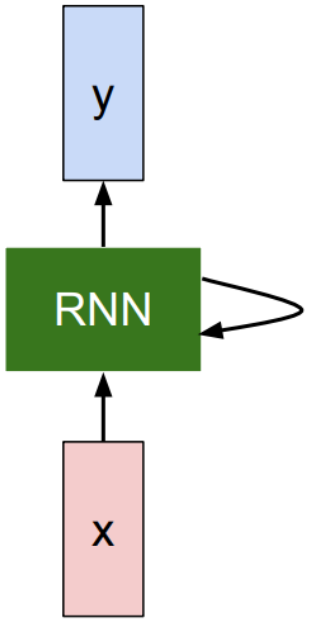
循环神经网络

- 序列处理



循环神经网络

- Recurrent neural network (RNN)
  - 输入$\boldsymbol{x} = \{x_1, x_2, \ldots, x_N\}$是一个序列
  - RNN递归地处理输入信号



$$h_t = f_W(h_{t-1}, x_t)$$

new state　old state　input vector at some time step

some function with parameters W

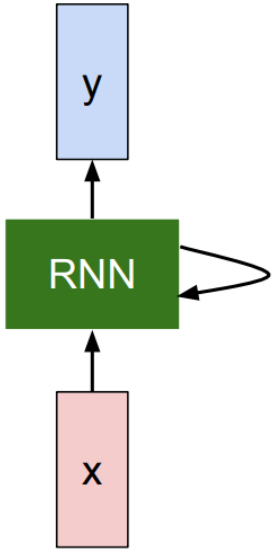
$$y_t = f_{W_{hy}}(h_t)$$

output　new state

another function with parameters W_o

循环神经网络

- Vanilla RNN
  - RNN的一种最简单的形式，具有单个隐藏单元
  - 存在的问题
    - ✓ 梯度消失和梯度爆炸
      - RNN的权值矩阵循环相乘导致的
    - ✓ 长期依赖
      - 之前比较长的时间片的特征被覆盖/遗忘
    - ✓ 难并行



$$h_t = f_W(h_{t-1}, x_t)$$

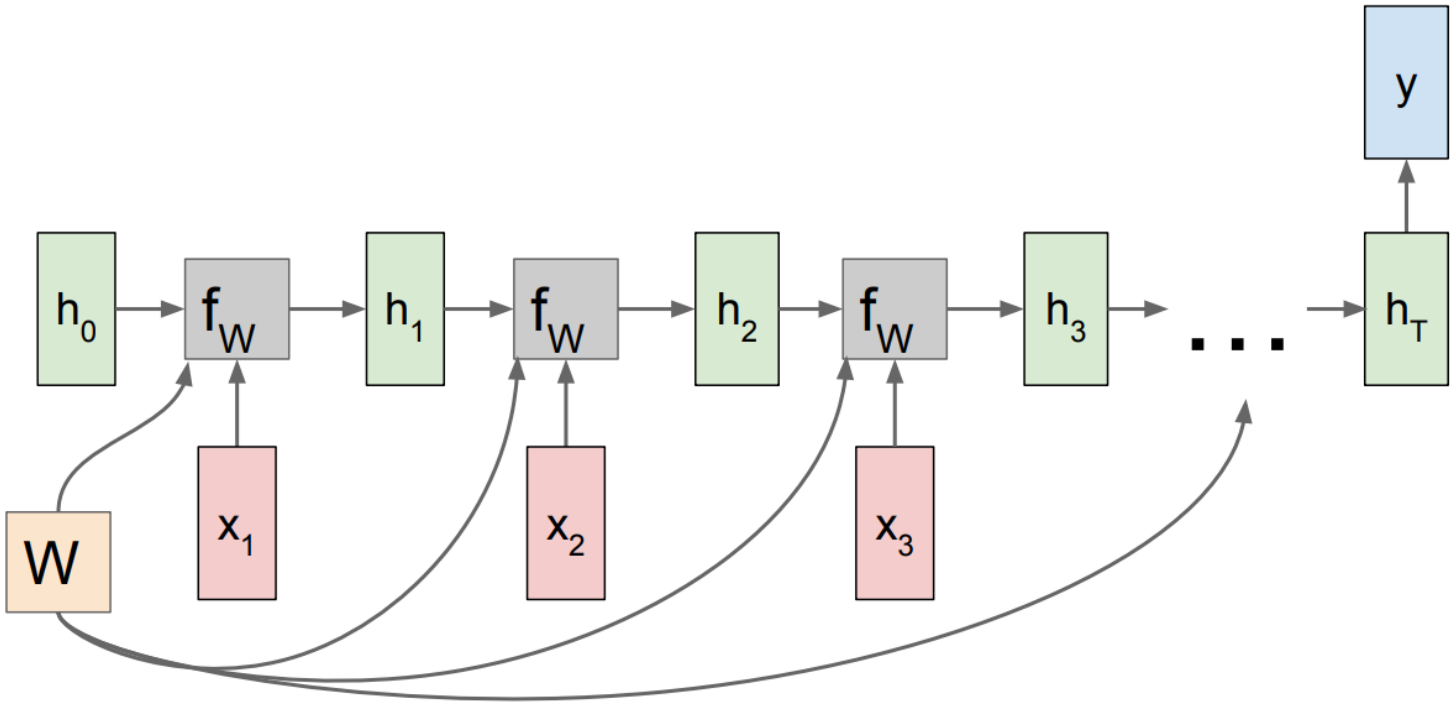
$$\downarrow$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

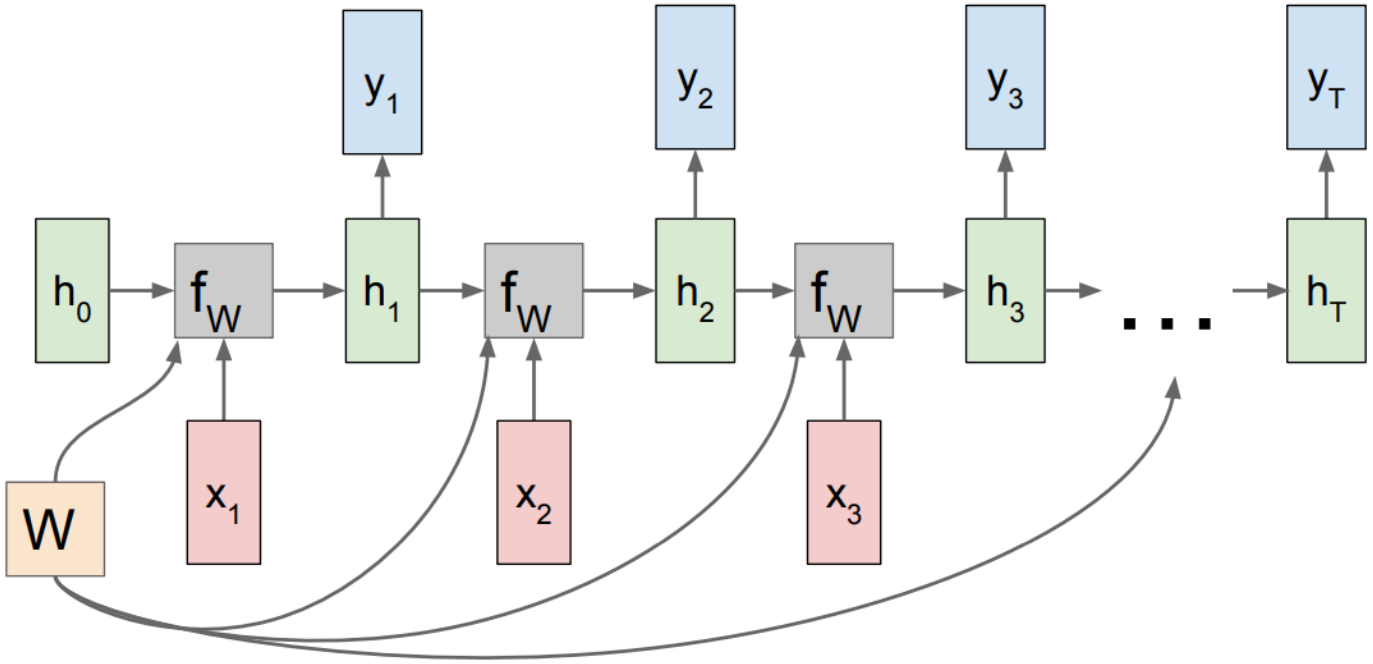
循环神经网络

- Vanilla RNN的计算图
  - 序列输入，单个输出的形式



循环神经网络

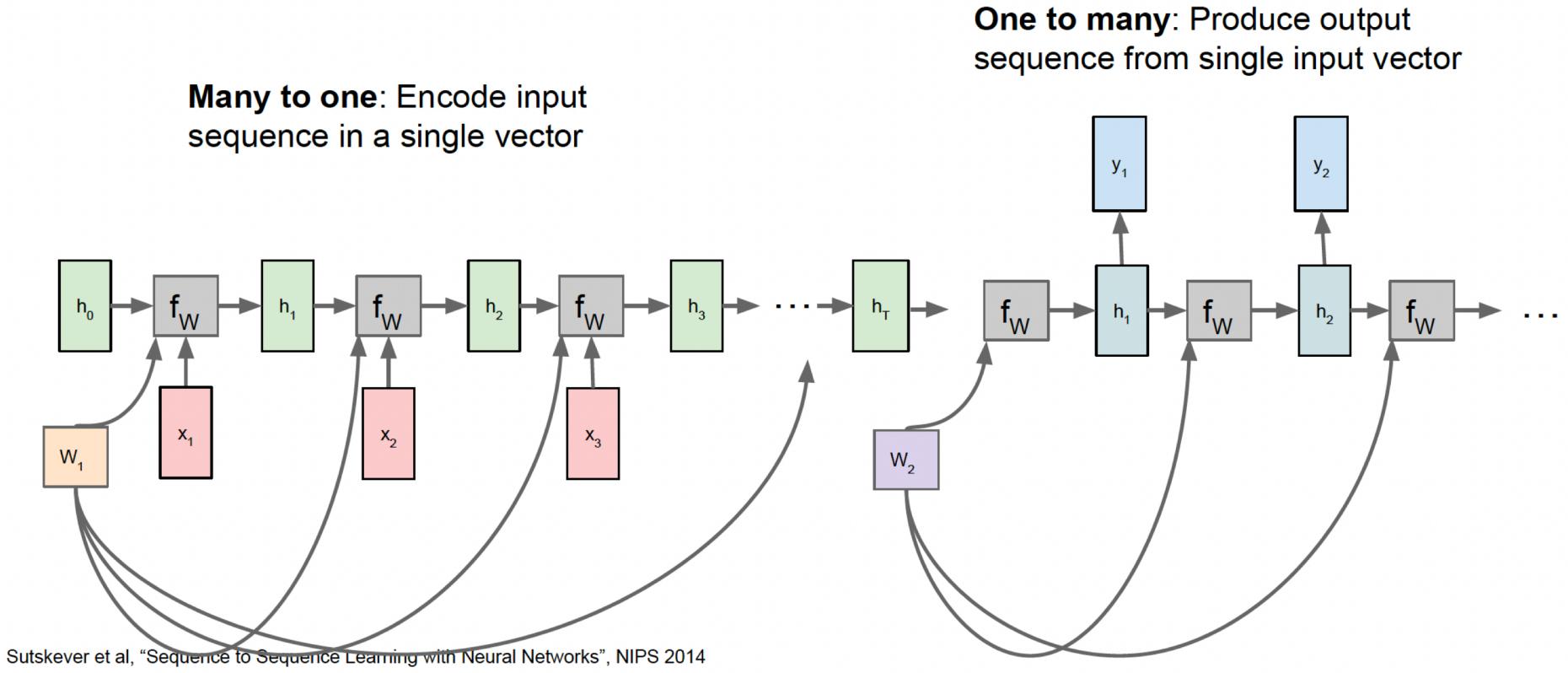
- Vanilla RNN的计算图
  - 序列输入，序列输出的形式



网络参数W在每个时刻都是共享的

循环神经网络

- Sequence to Sequence:
  - Many-to-one + One-to-many

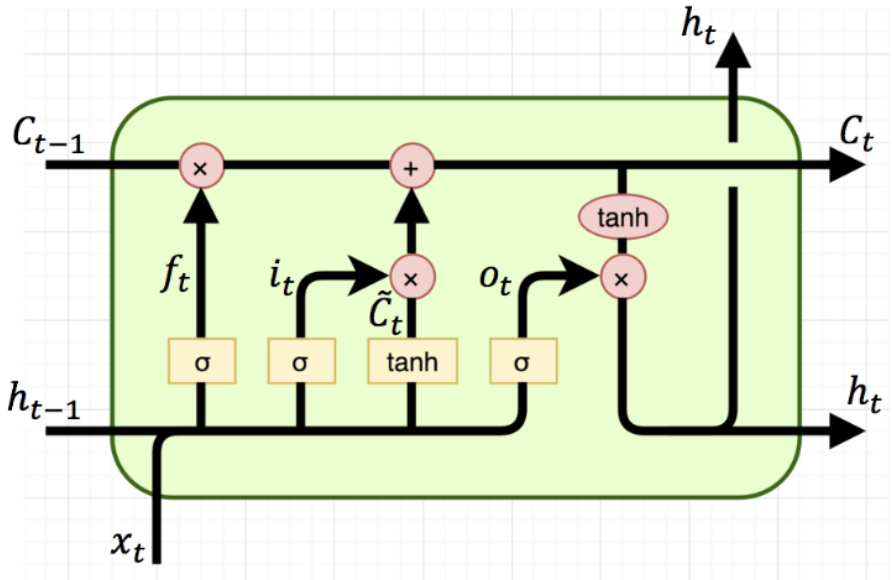


Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

循环神经网络

Variants of RNN: LSTM

- 可以解决长序列训练过程中的梯度消失和梯度爆炸问题，并缓解长时依赖问题
- Cell state（**单元状态**）：用于“记忆”当前的状态，上面包含记忆的删除、更新等
- 遗忘门 f_t ，输入门i_t，输出门o_t



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

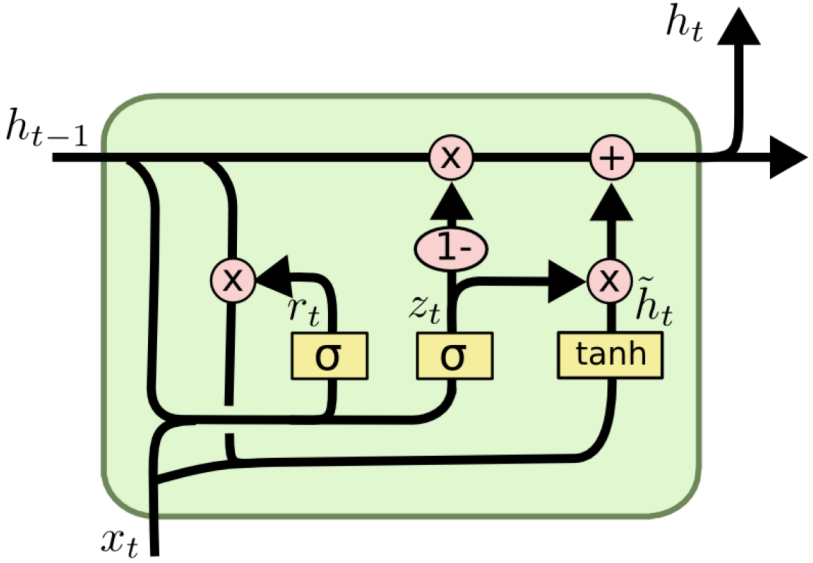
$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

$$h_t = o_t \odot \tanh(C_t)$$

循环神经网络

- Variants of RNN: GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

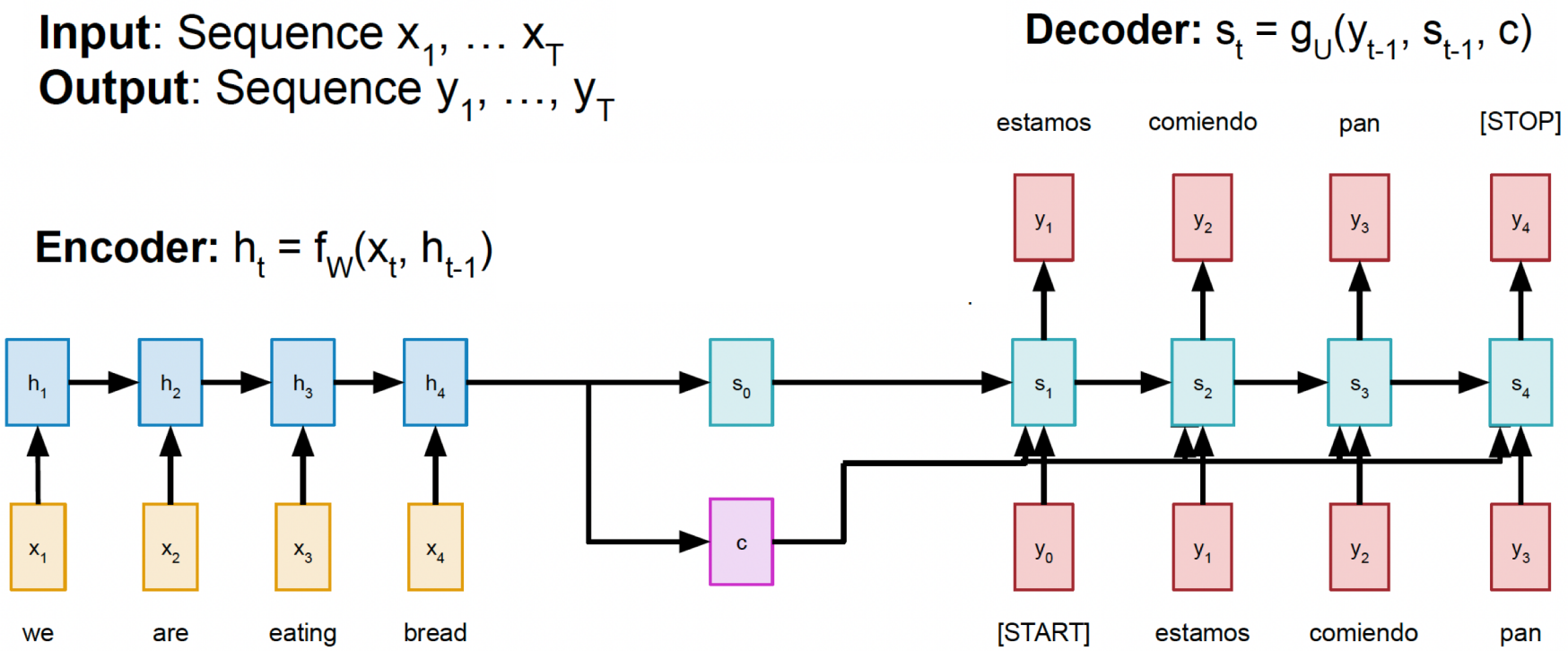
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

深度学习基础

- 深度前馈网络
- 卷积神经网络
- 循环神经网络
- Transformer网络
  - RNN+注意力
  - 注意力机制
  - Transformer

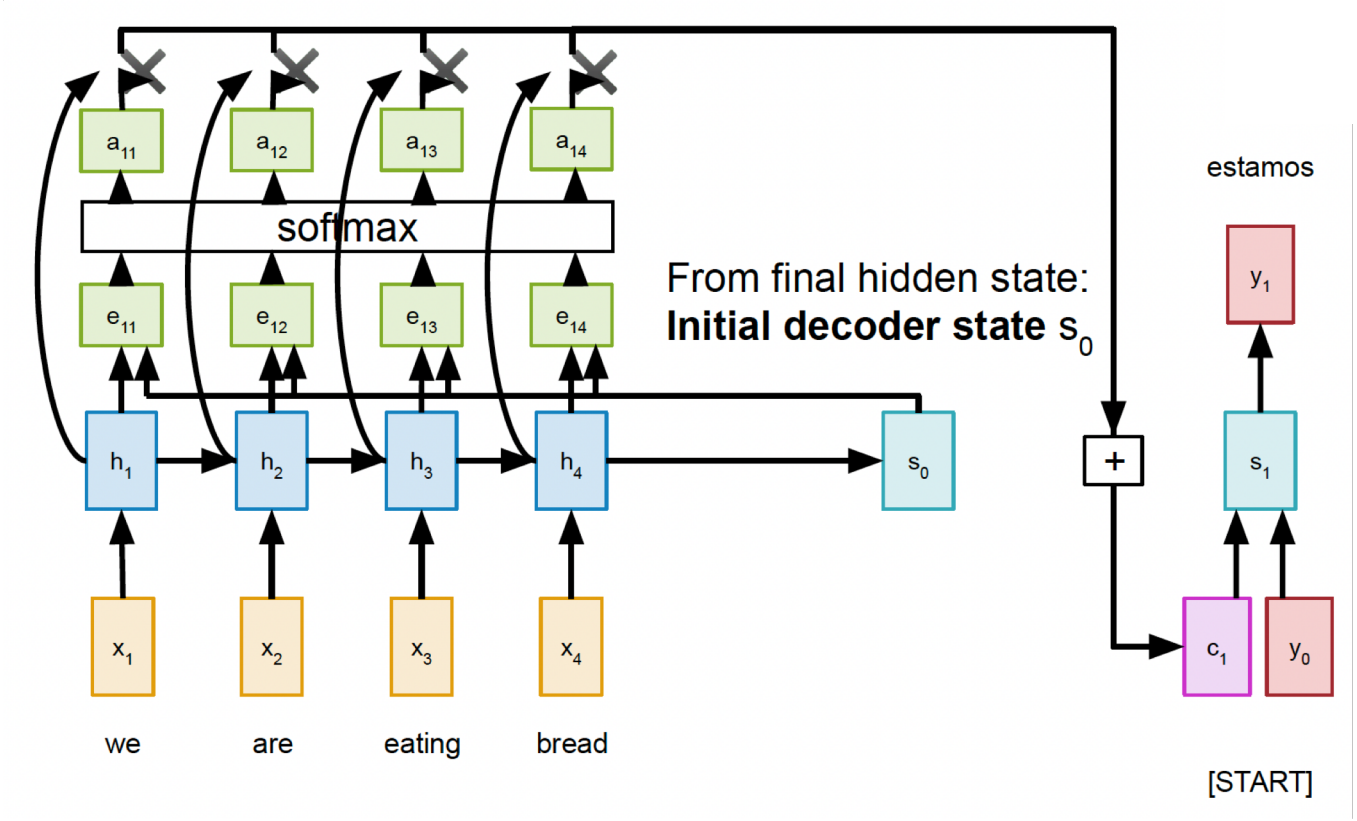
基于RNN的序列到序列模型



问题：输入序列的信息被压缩到单个向量

RNN+注意力

引入注意力机制



1. 计算对齐分数 (alignment scores)

$$e_{t,i} = f_{\text{attn}}(s_{i-1}, h_i)$$

2. 归一化得到注意力权值(attention weights)

$$0 < a_{t,i} < 1,$$

$$\sum_i a_{t,i} = 1$$

3. 计算上下文向量

$$c_t = \sum_i a_{t,i} h_i$$

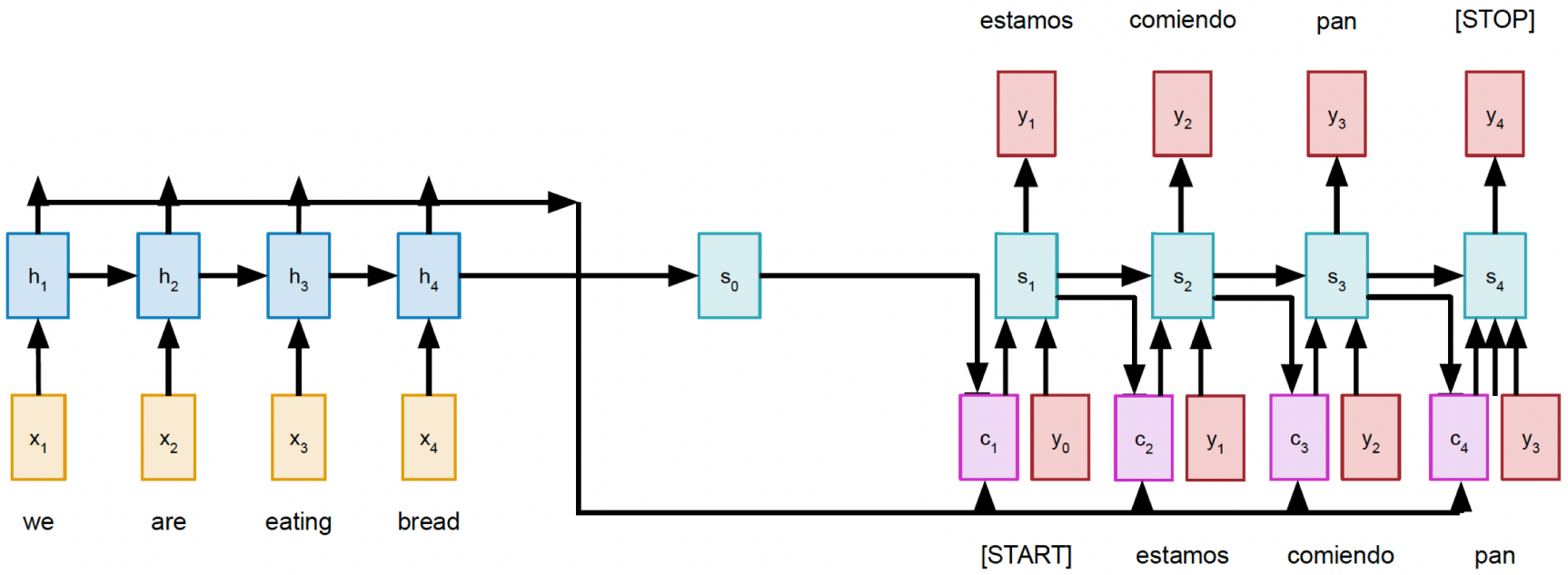
4. 更新s_t

$$s_t = g_U(y_{t-1}, s_{t-1}, c_t)$$

- Dzmitry Bahdanau, et al. “Neural machine translation by jointly learning to align and translate”. In ICLR 2015.

RNN+注意力

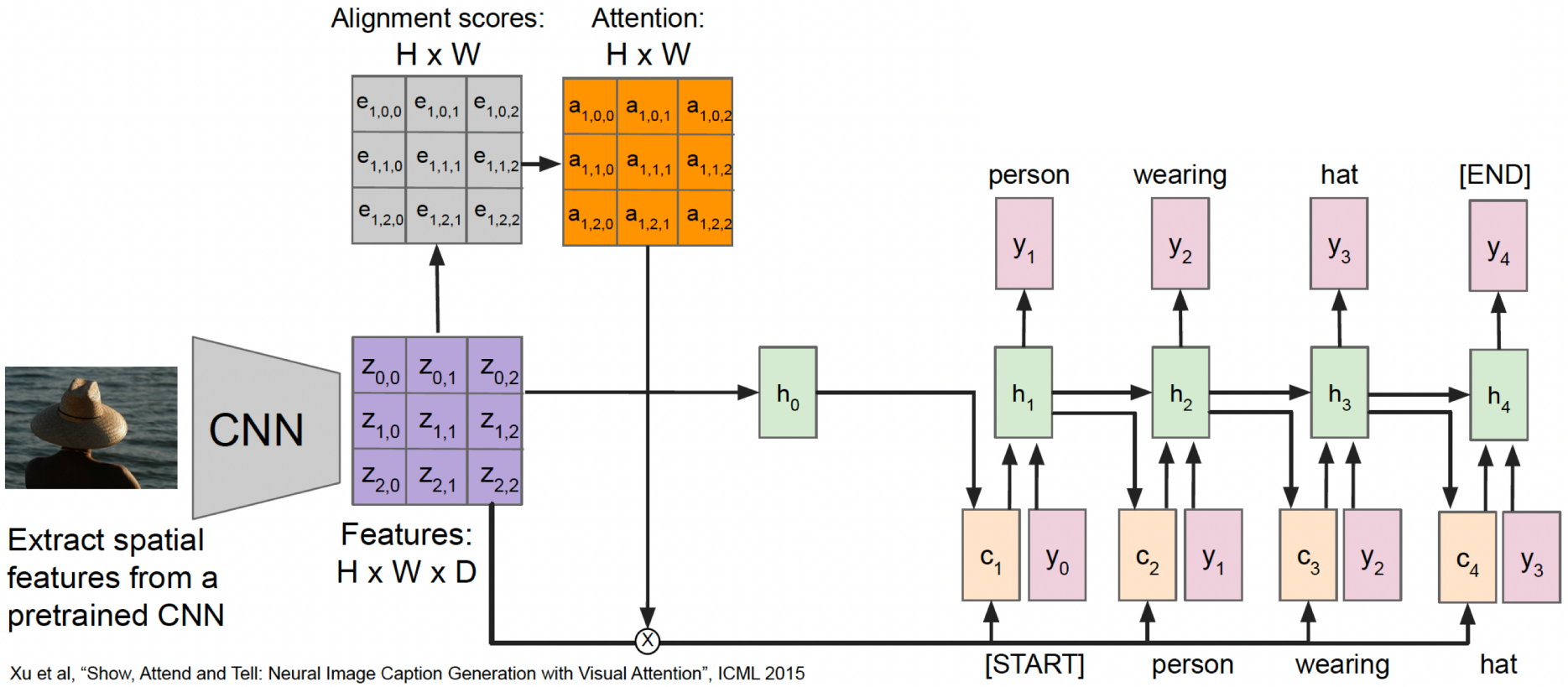
- 引入注意力机制
  - 解码器每一步使用不同的上下文向量
  - 不同时刻关注输入序列的不同部分



- Dzmitry Bahdanau, et al. “Neural machine translation by jointly learning to align and translate”. In ICLR 2015.

RNN+注意力

- Image Captioning



Xu et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In ICML, 2015.

注意力机制

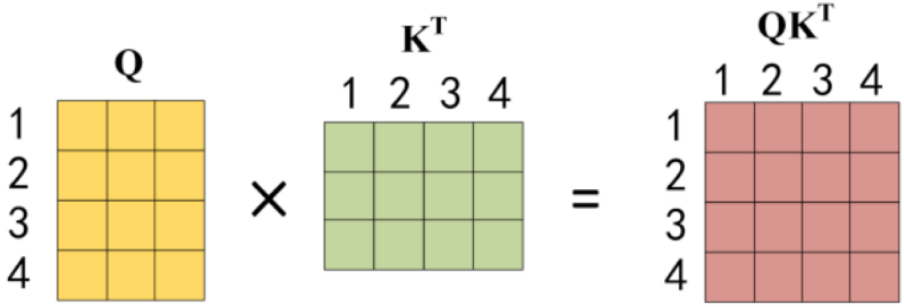
注意力机制

- 注意力模块在计算的时候需要用到矩阵Q(查询), K(键值), V(值)

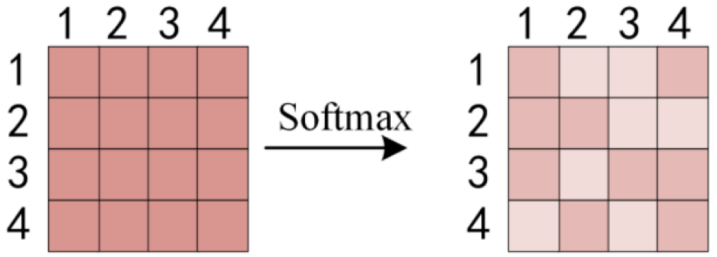
$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

d_k是Q,K矩阵的列数，即向量维度

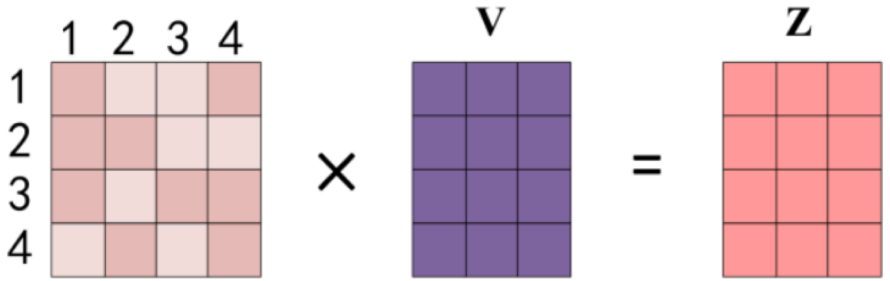
Q、K 相似性矩阵计算



相似性矩阵归一化

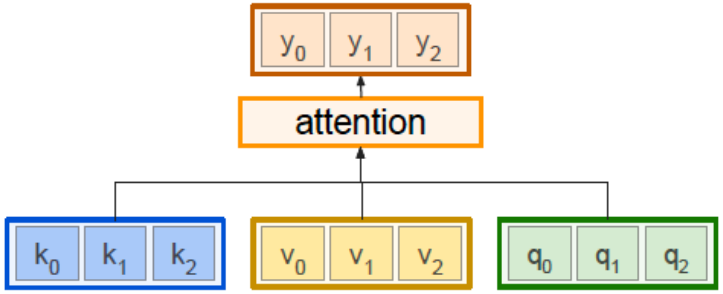


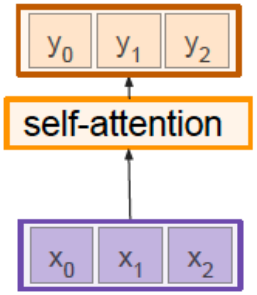
通过相似性矩阵对 V 加权求和



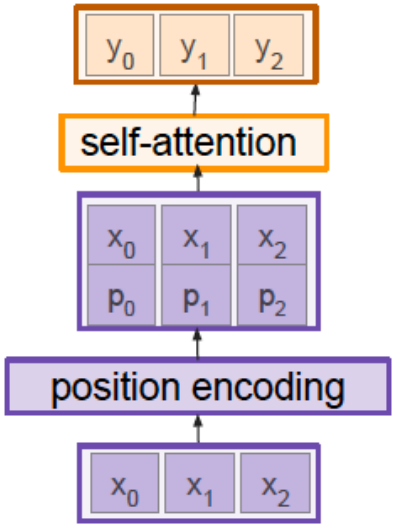
注意力机制

- 自注意力（Self Attention）





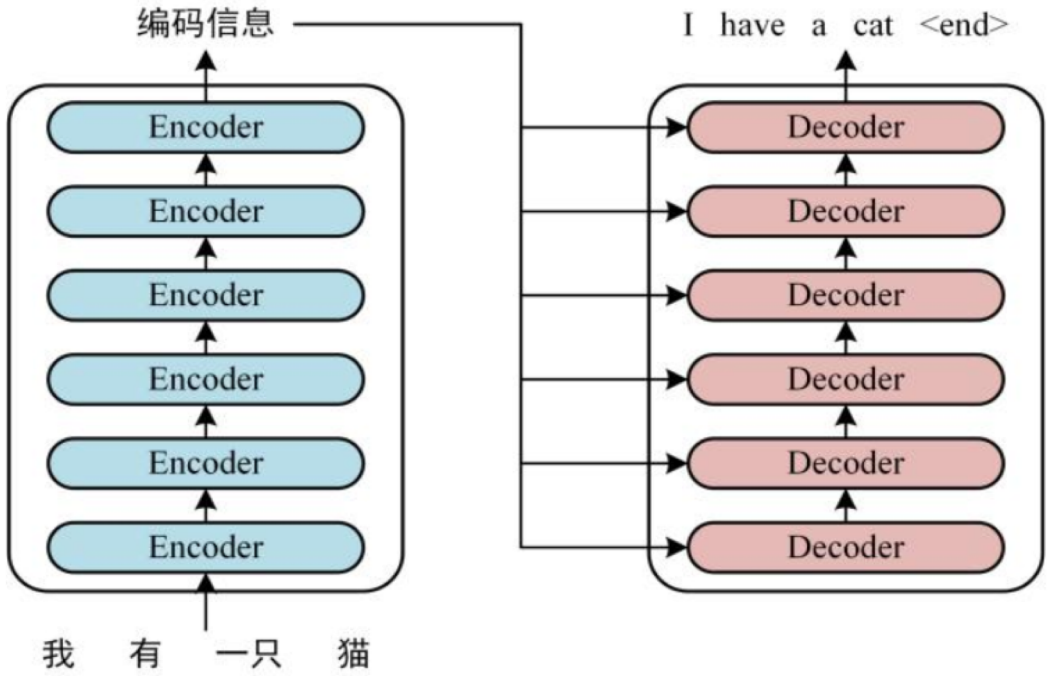
- 位置编码



Transformer

Transformer概况

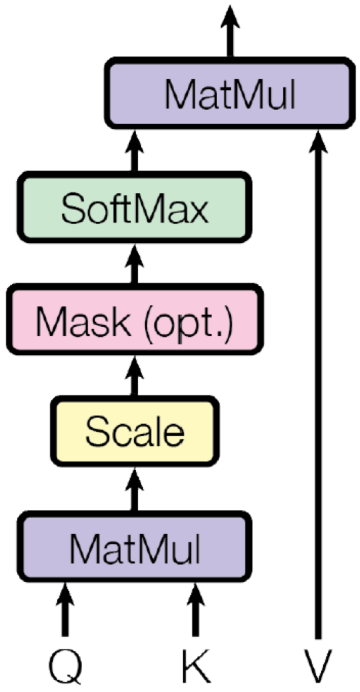
- Transformer是Google团队在2017年提出的一种自然语言处理模型
- Transformer模型使用了注意力机制，使得模型可以并行化训练，而且能够拥有全局信息
- 主要由编码器（Encoder）和解码器（Decoder）组成

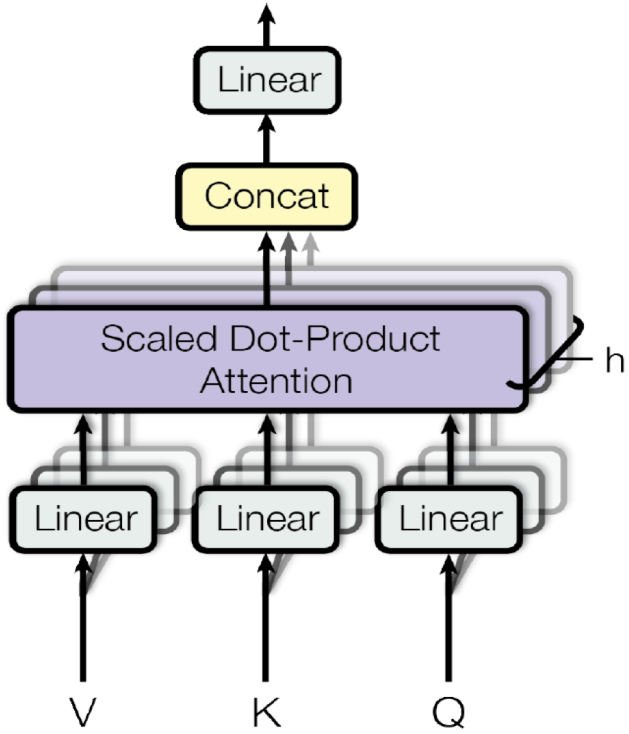


- Ashish Vaswani, et al. “Attention is all you need”. In NeurIPS, 2017.

Transformer

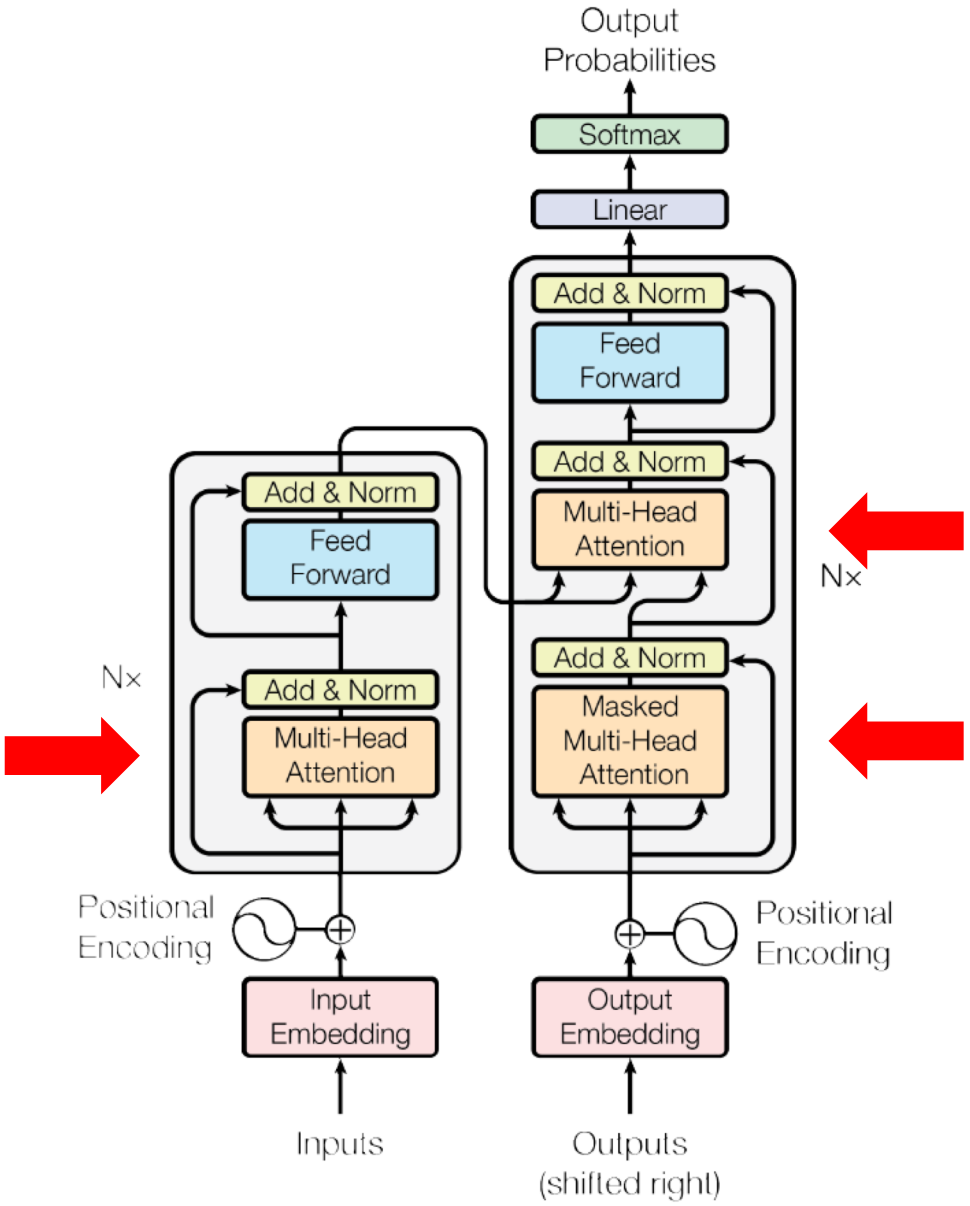
- Transformer概况
  - 注意力模块的基本形式
  - 多头注意力（Multi-head Attention）：Transformer中，通过构造一系列并列的注意力模块，组成多头注意力机制。通过将输入映射到不同的子空间，有助于学习更加丰富的注意力表达。





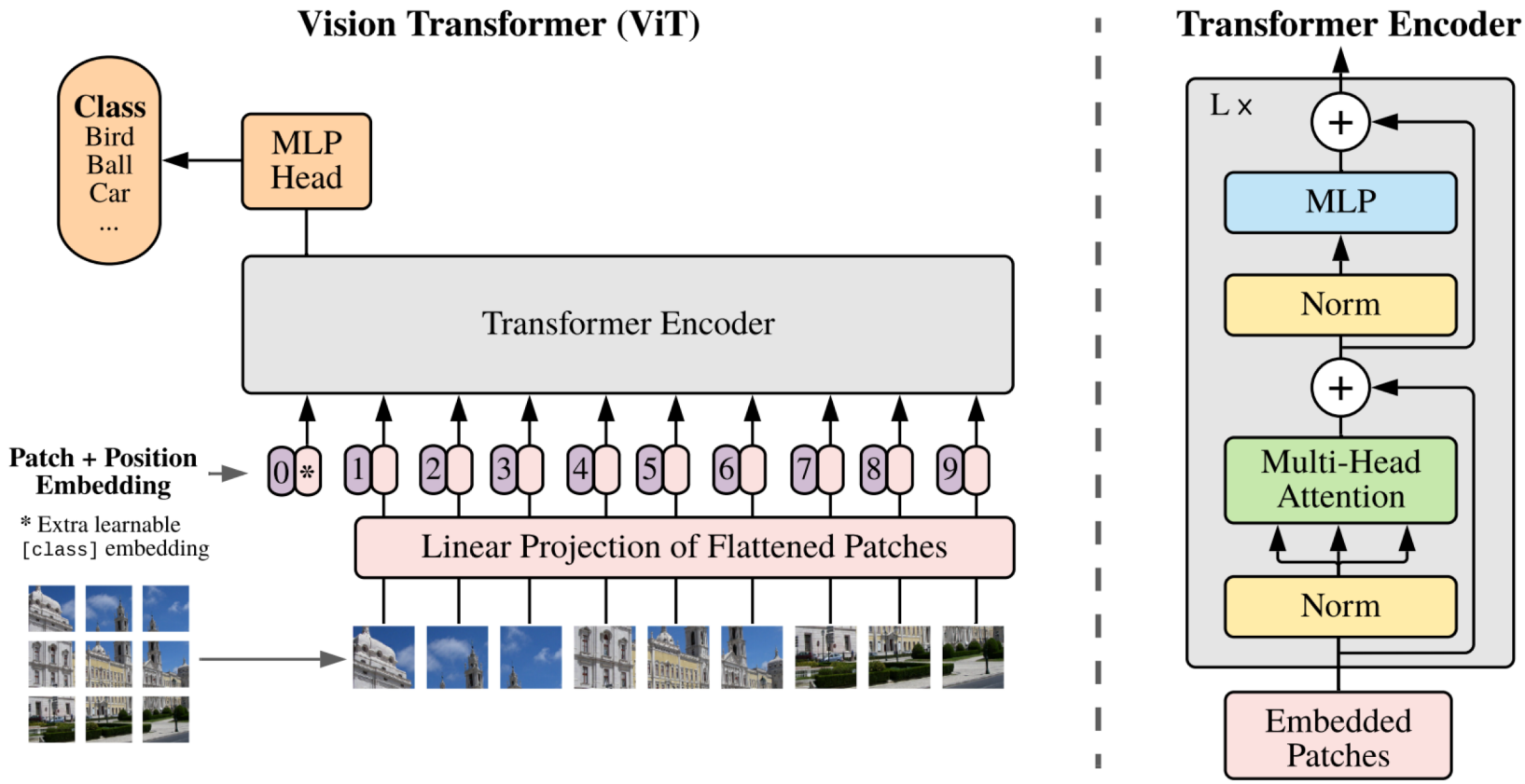
Transformer

- Transformer概况
  - 编码器（encoder）和解码器（decoder）主要由注意力模块组成
  - 编码器和解码器反复堆叠（N=6），以便更好地通过注意力机制获取全局信息



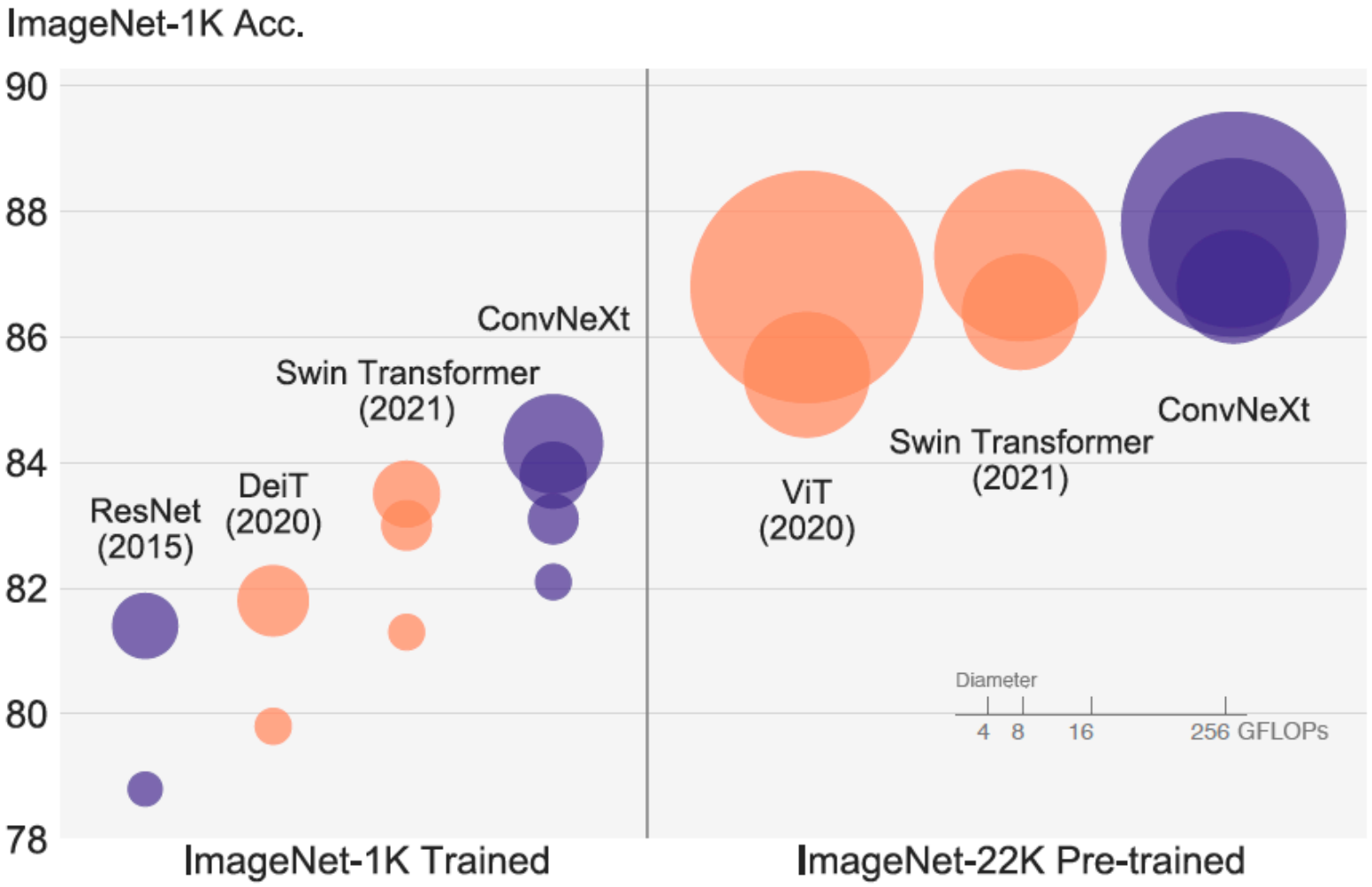
Transformer

视觉Transformer



- Alexey Dosovitskiy, et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In ICLR, 2021.

Transformer VS. CNN



- Z. Liu, et al. “A ConvNet for the 2020s”. In CVPR, 2022.