Lecture 11: 其他神经网络及其应用

Kai Yu and Yanmin Qian

Cross Media Language Intelligence Lab (X-LANCE)
Department of Computer Science & Engineering
Shanghai Jiao Tong University

Spring 2021

常见的高级神经网络结构

- 卷积神经网络（Convolutional Neural Network, CNN）
- 循环神经网络（Recurrent Neural Network, RNN）
  - 长短期记忆（Long-Short Term Memory, LSTM）
- 注意力（attention）机制与 Transformer

CNN — 历史

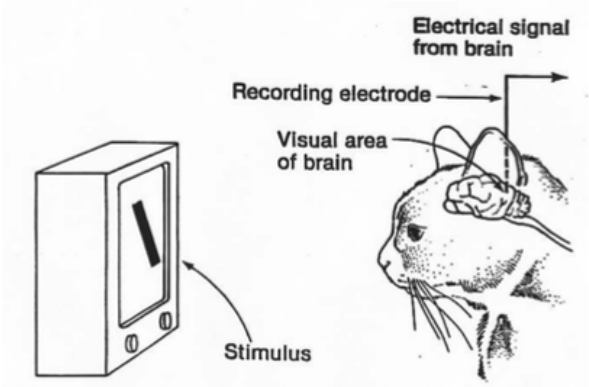
Hubel & Weisel

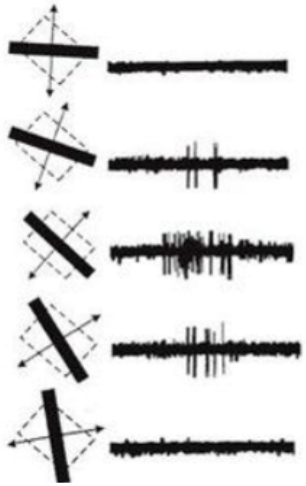
1959
猫的纹状体皮层中单个神经元的感受野

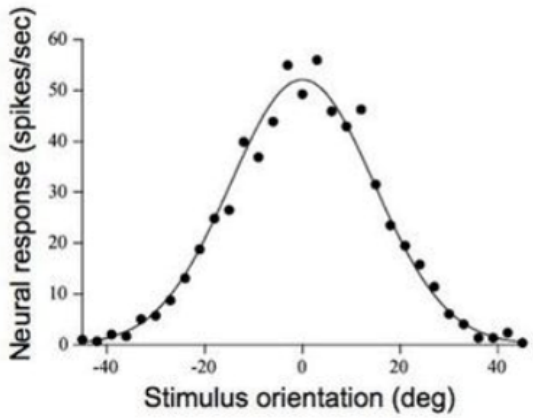
1962
猫视觉皮层的感受野、双眼相互作用和功能结构

1968

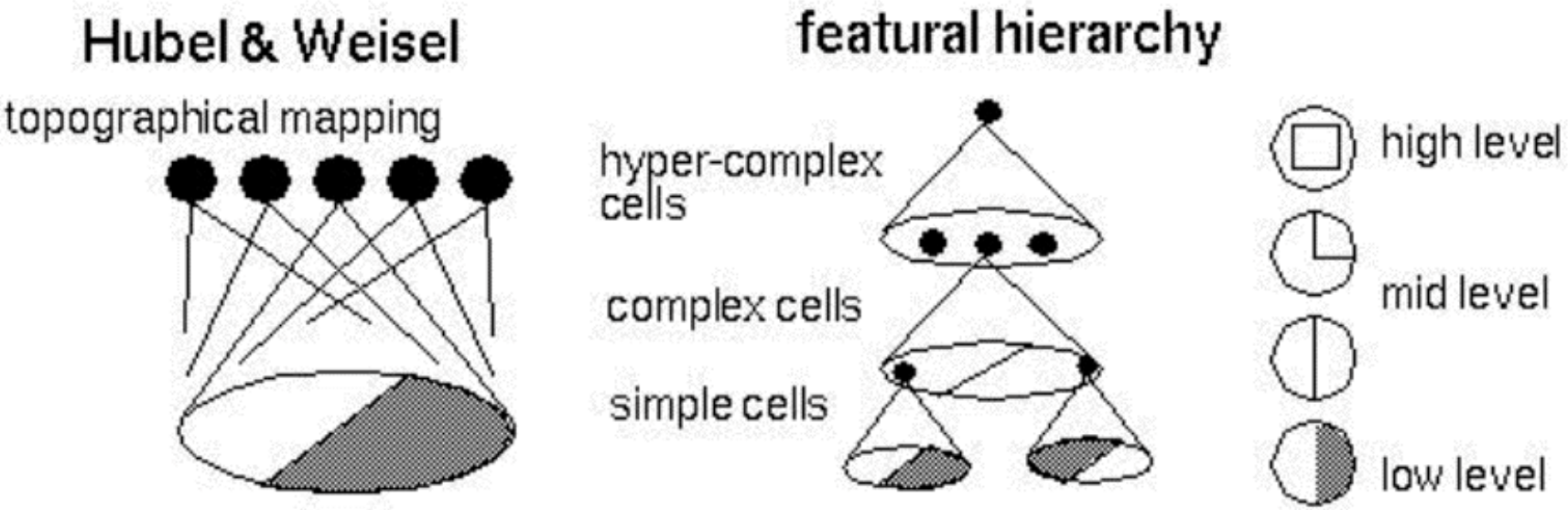
...



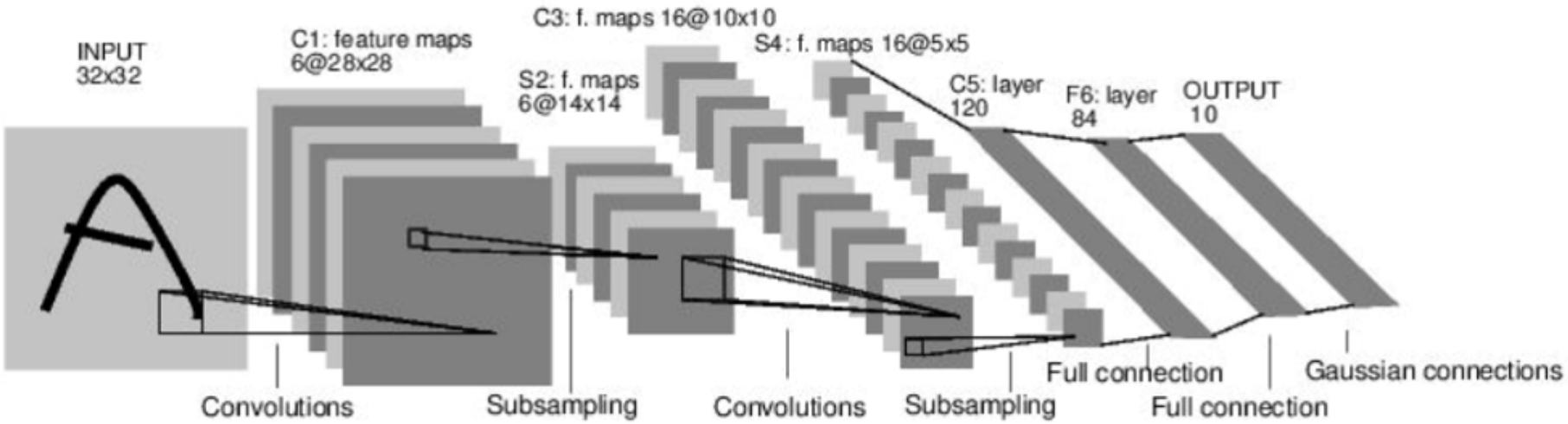




层次化的组织结构 (Hierarchical organization)

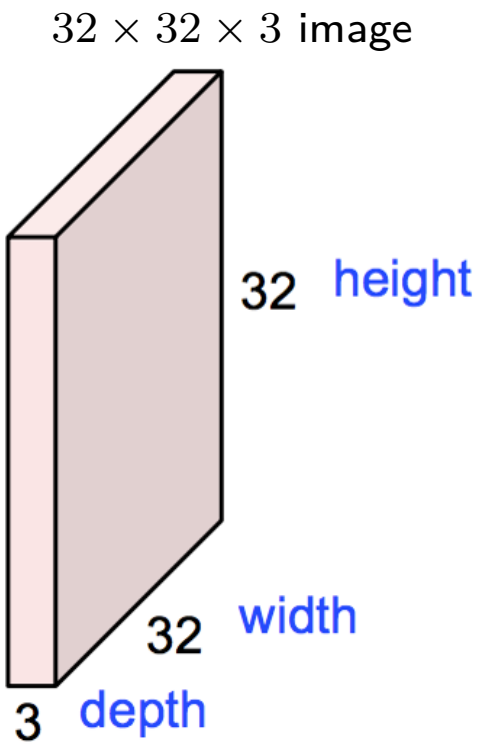


LeNet-5



LeCun, Yann; Léon Bottou; Yoshua Bengio; Patrick Haffner (1998)

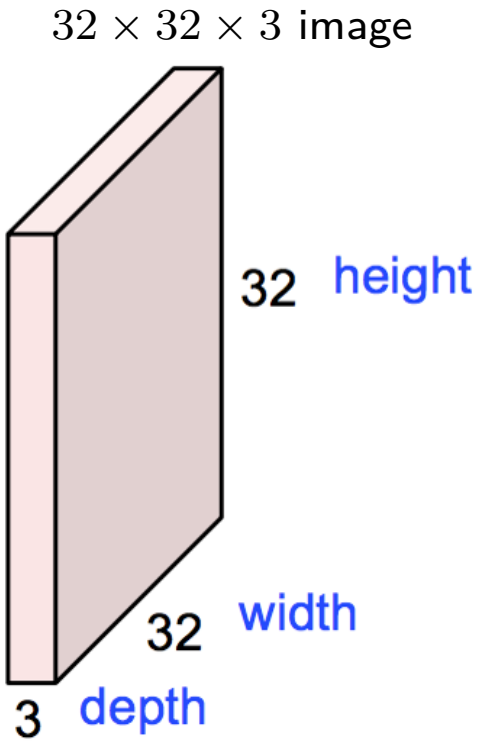
卷积层



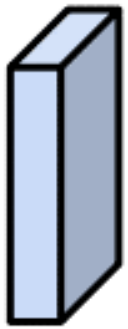
卷积层是具有‘3D’结构的网络层

在这里，按照常理，我们有一个具有 3 个特征图(feature maps)的卷积层，每个图为 32×32

卷积层



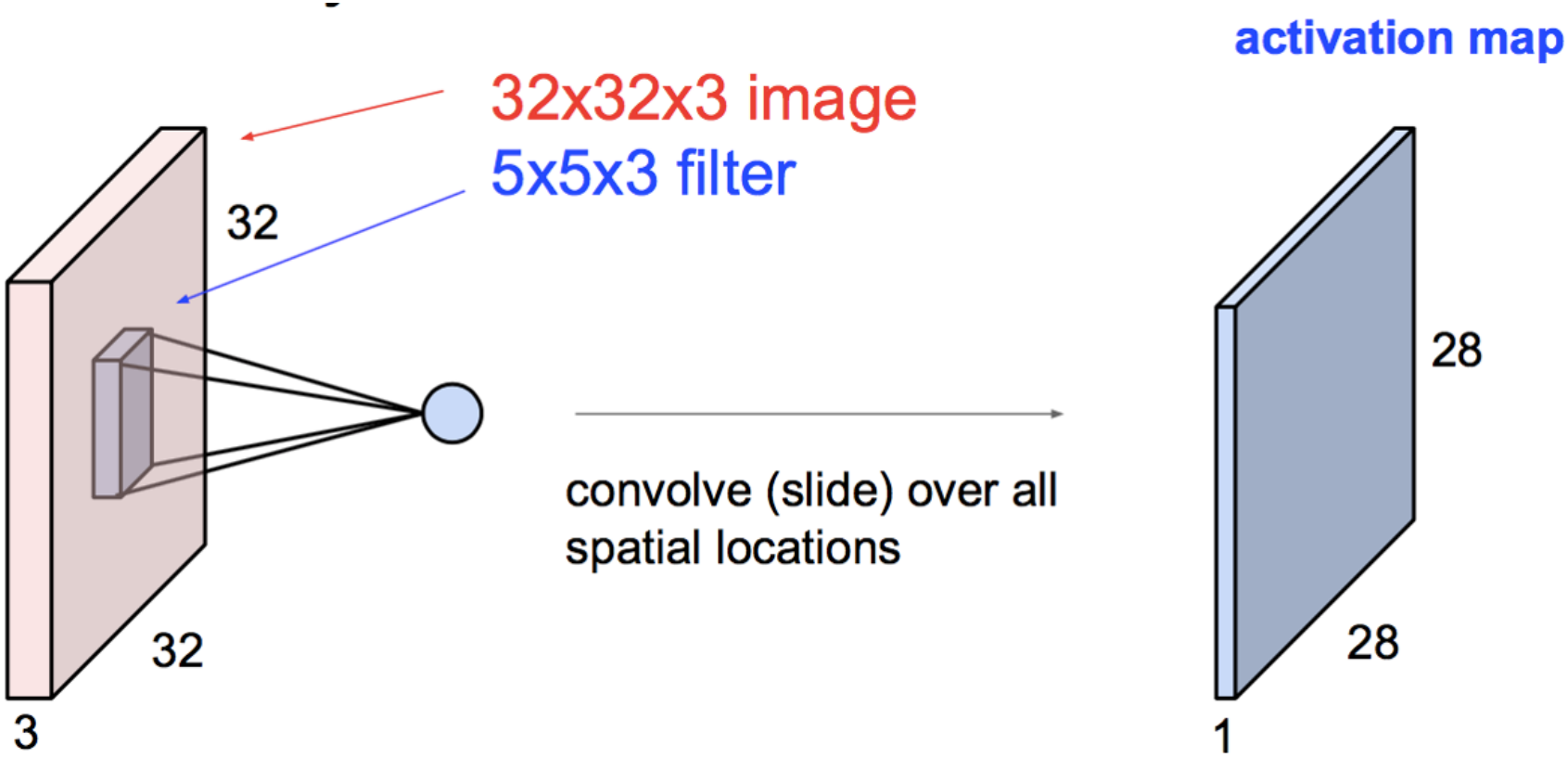
$5 \times 5 \times 3$ filter



将 filter 与图像进行**卷积 convolution**

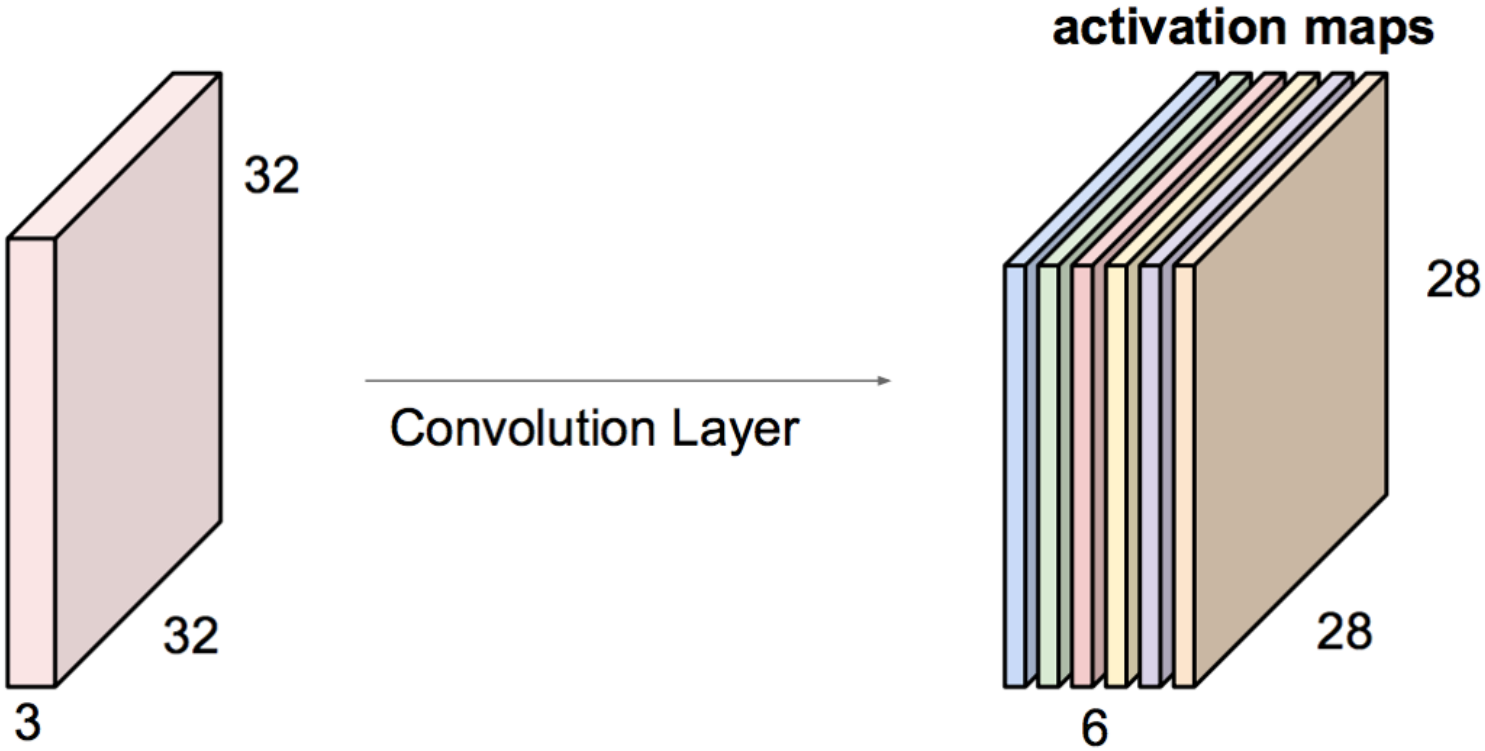
即“在图像上进行空间滑动，计算点积”

卷积层


32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
activation map
28
28
1

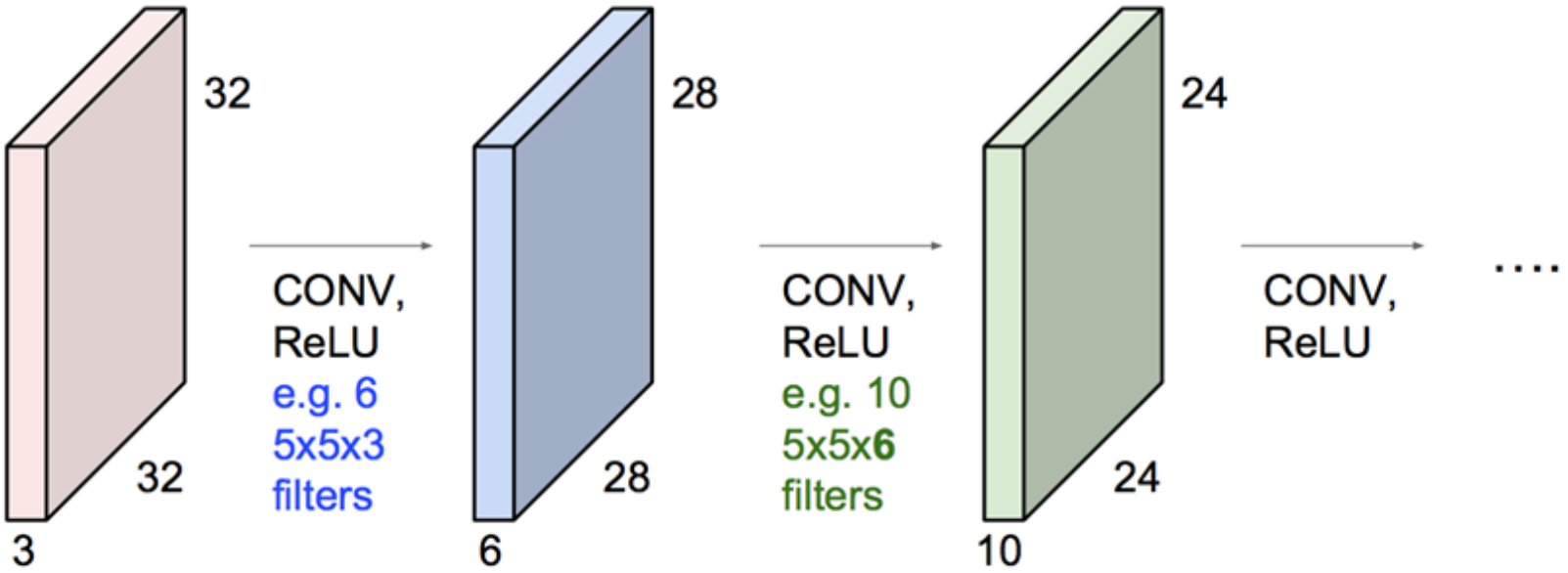
卷积层

对于每个卷积层，最终的特征图数量是个超参数，这里就是 6



卷积层

一个卷积层序列，其中卷积后通常使用 ReLU 激活函数



1st conv layer,
3 feature maps

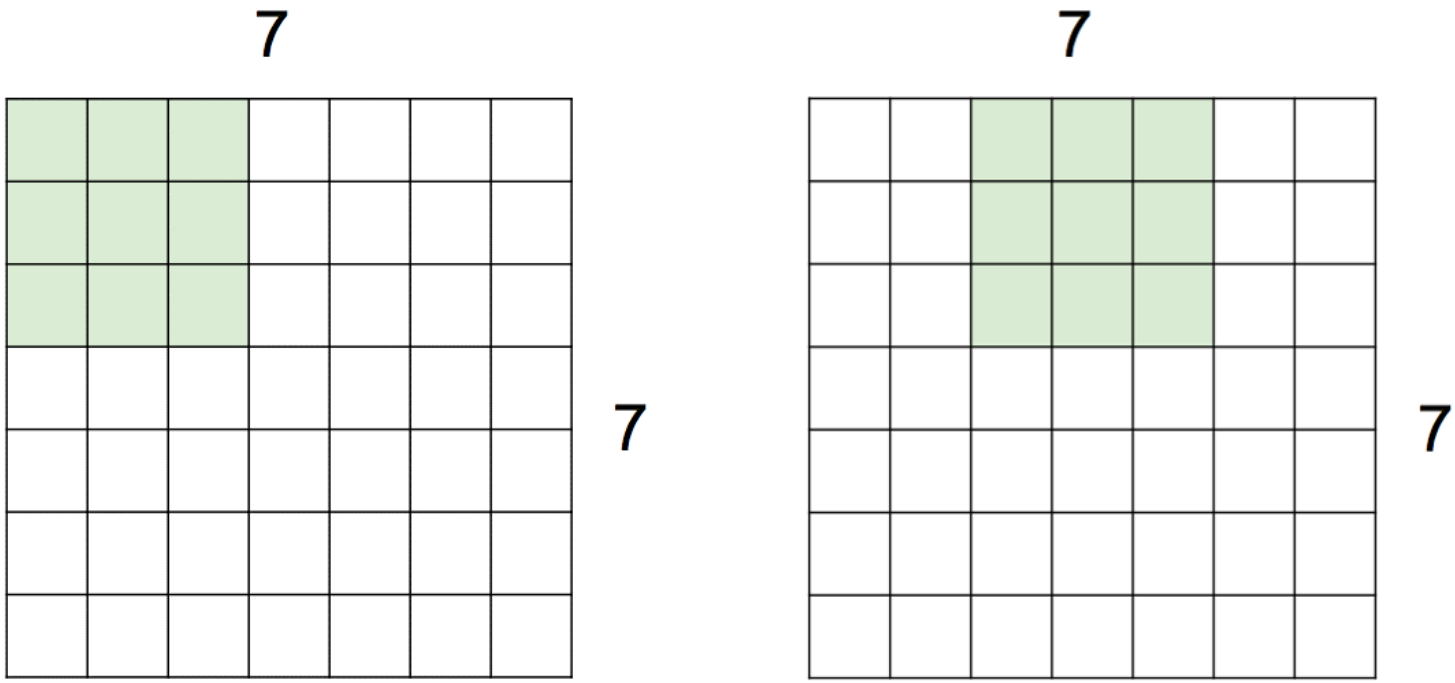
2nd conv layer,
6 feature maps

3rd conv layer,
10 feature maps

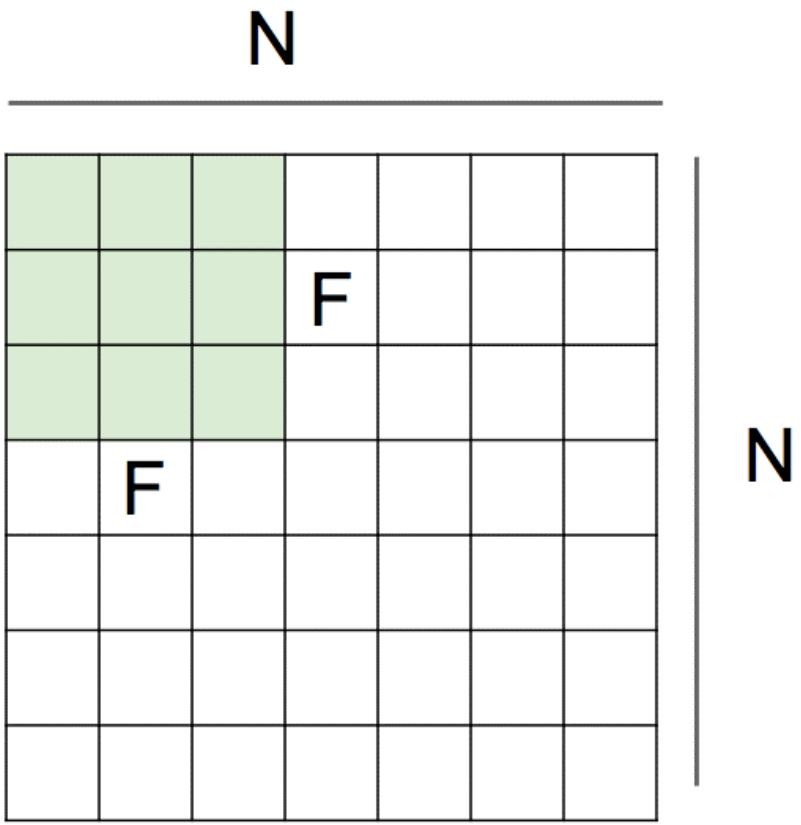
卷积层：步长（Stride）

另一个超参数：步长

下例中 7×7 的特征图，3×3 filter 按步长 $\texttt{stride} = 2$ 与之进行卷积



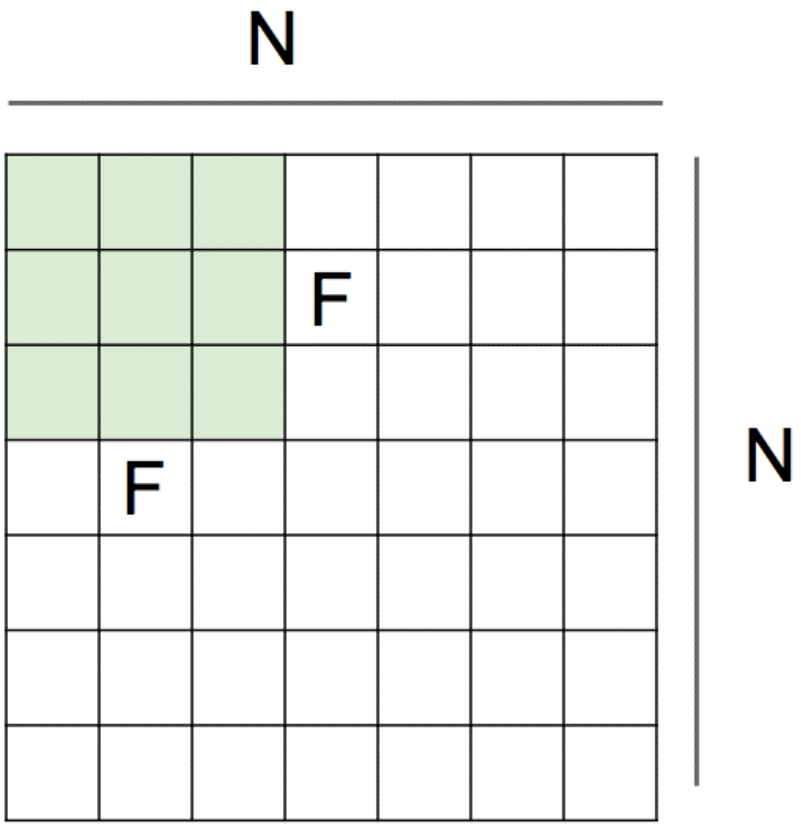
卷积层：步长（Stride）



问题：

输出的特征图大小是?

卷积层：步长（Stride）

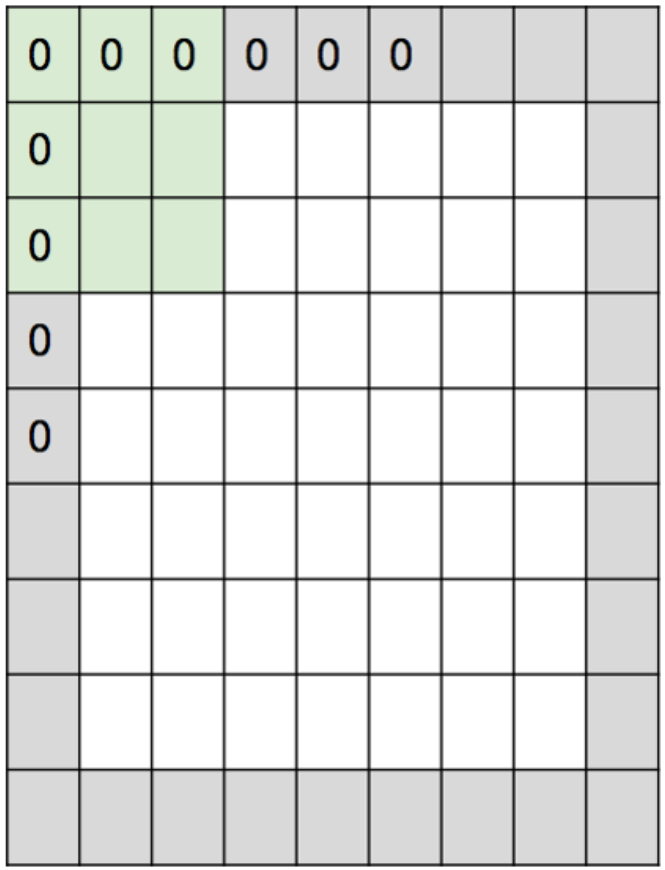


问题：
输出的特征图大小是?

$$(\mathtt{N} - \mathtt{F})/\mathtt{stride} + 1$$

不能完全整除？丢掉它们
考虑 $\mathtt{stride} = 3$
因此，请避免这种情况

卷积层：补零（Zero-Padding）



问题：

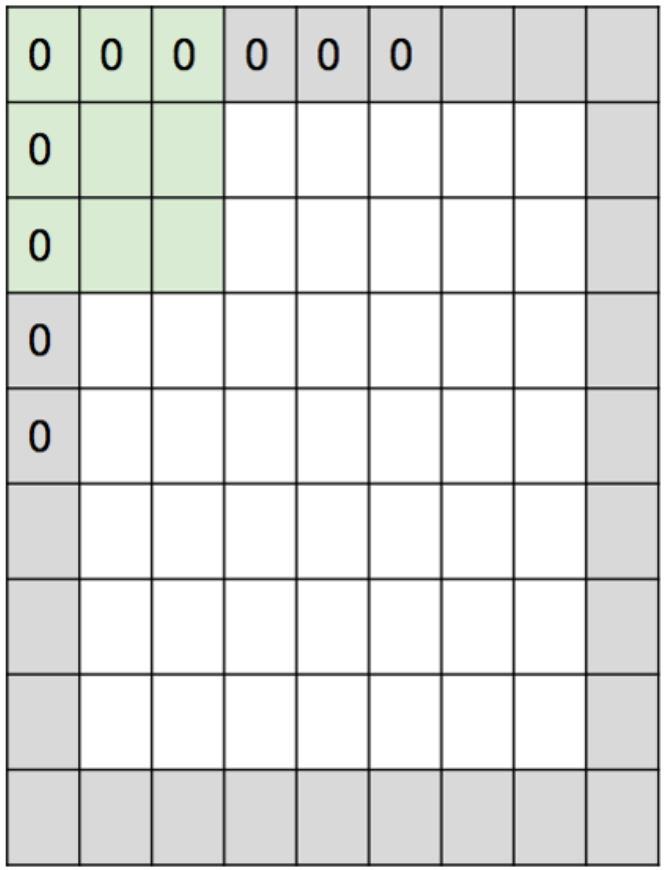
如果在每个边界处补一个 0 会发生什么？

回顾：

$(\mathrm{N}-\mathrm{F})/\mathtt{stride}+1$

这里 $\mathtt{stride}=1$

卷积层：补零（Zero-Padding）



问题：

如果在每个边界处补一个 0 会发生什么？

回顾：

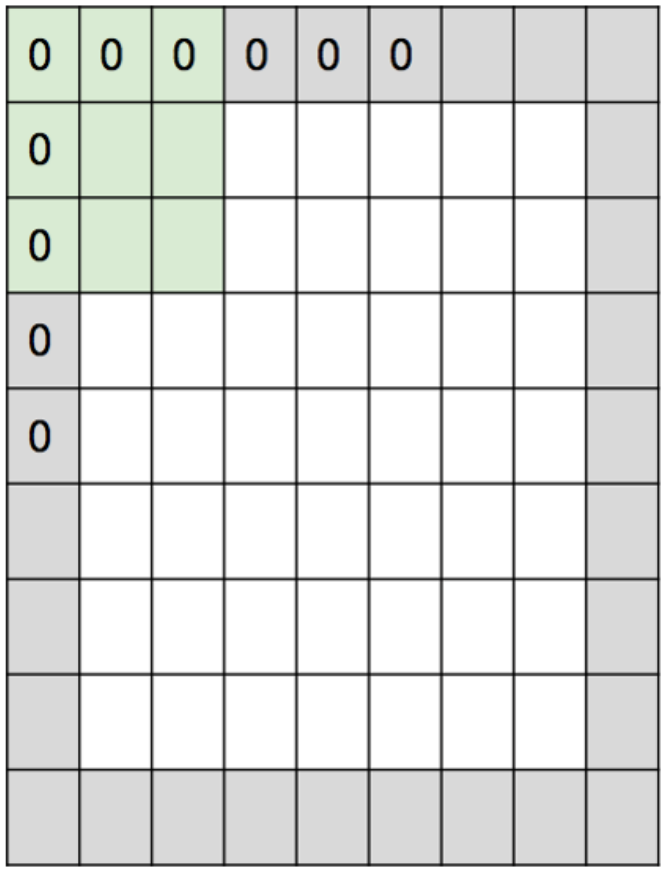
$(\mathrm{N}-\mathrm{F})/\mathtt{stride}+1$

这里 $\mathtt{stride}=1$

哇哦！

我们得到了 7×7 的输入 & 7×7 的输出！

卷积层：补零（Zero-Padding）



问题：

如果在每个边界处补一个 0 会发生什么？

回顾：

$(\mathrm{N}-\mathrm{F})/\mathtt{stride}+1$

这里 $\mathtt{stride}=1$

哇哦！

我们得到了 7×7 的输入 & 7×7 的输出！

卷积前使用补零是一种常见的做法

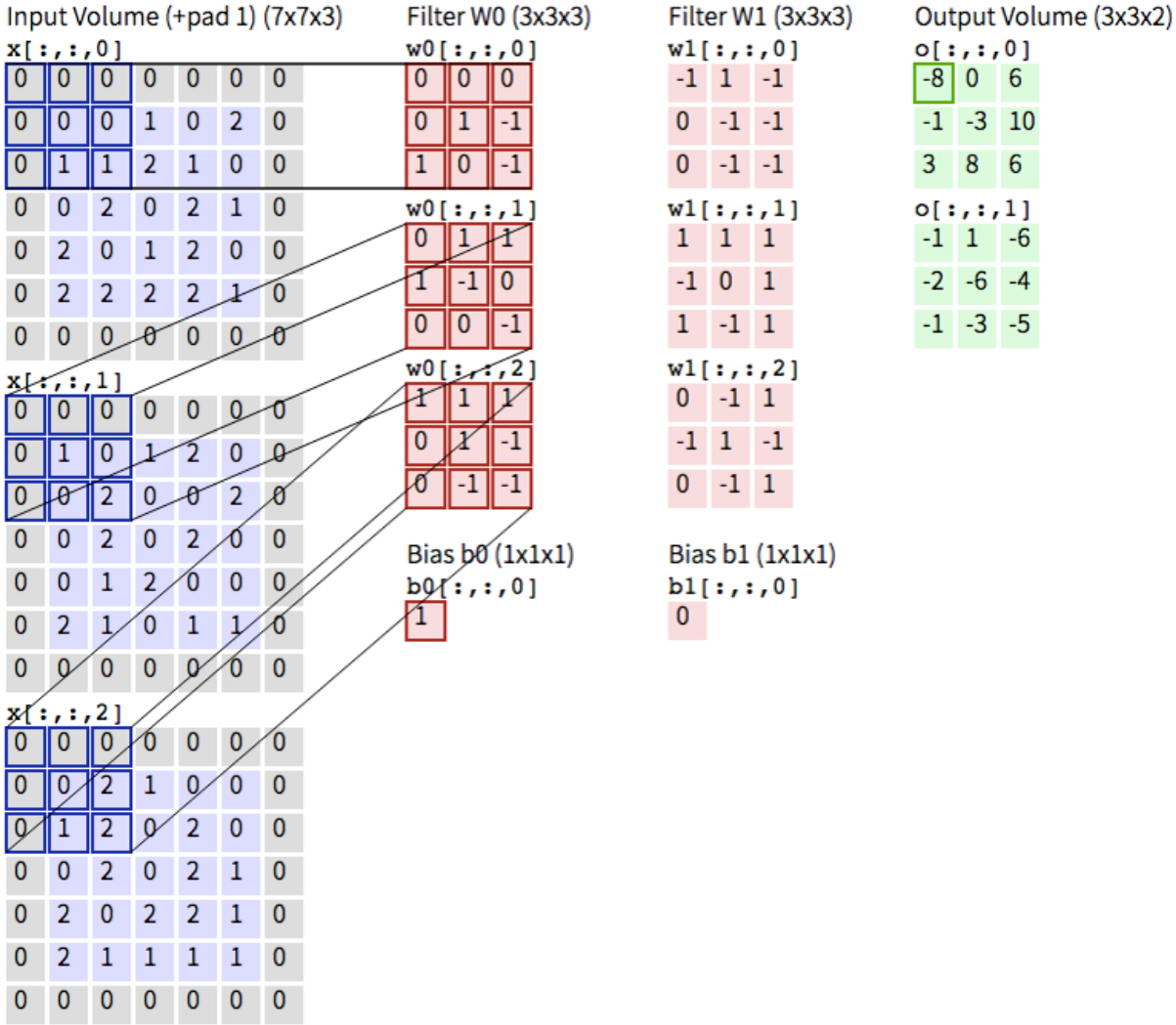
两个好处：

- 保留了特征图分辨率
- 更好地利用了边界信息，从而能提升性能

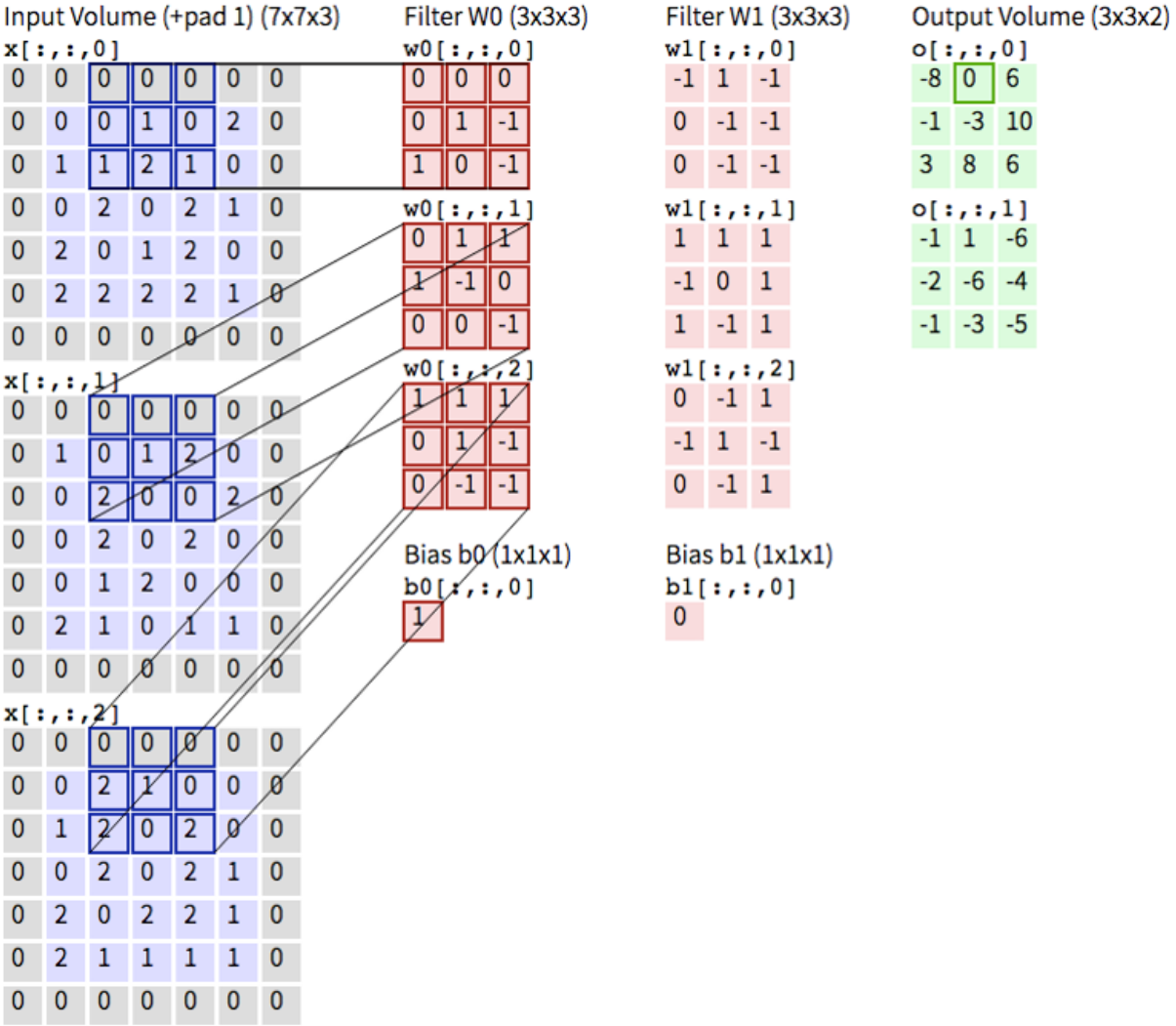
卷积层：示例

一个生动的 例子:

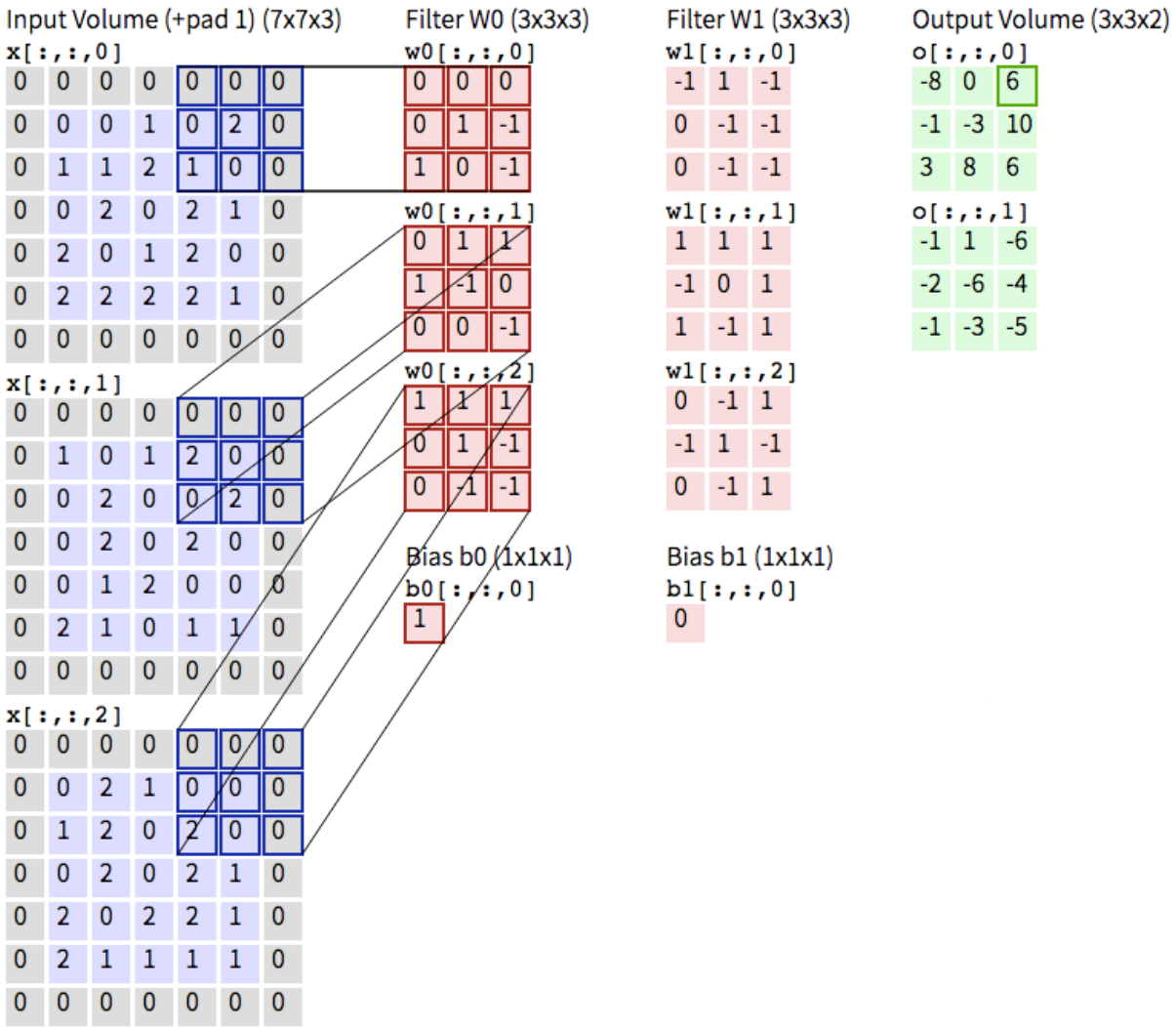
- $5 \times 5 \times 3$ 的输入图像 (3 个 5×5 的特征图)
- 和2 个 $3 \times 3 \times 3$ filter
- 那么得到2 个 3×3 的输出特征图
- $\texttt{Stride} = 2$
- $\texttt{pad} = 1$



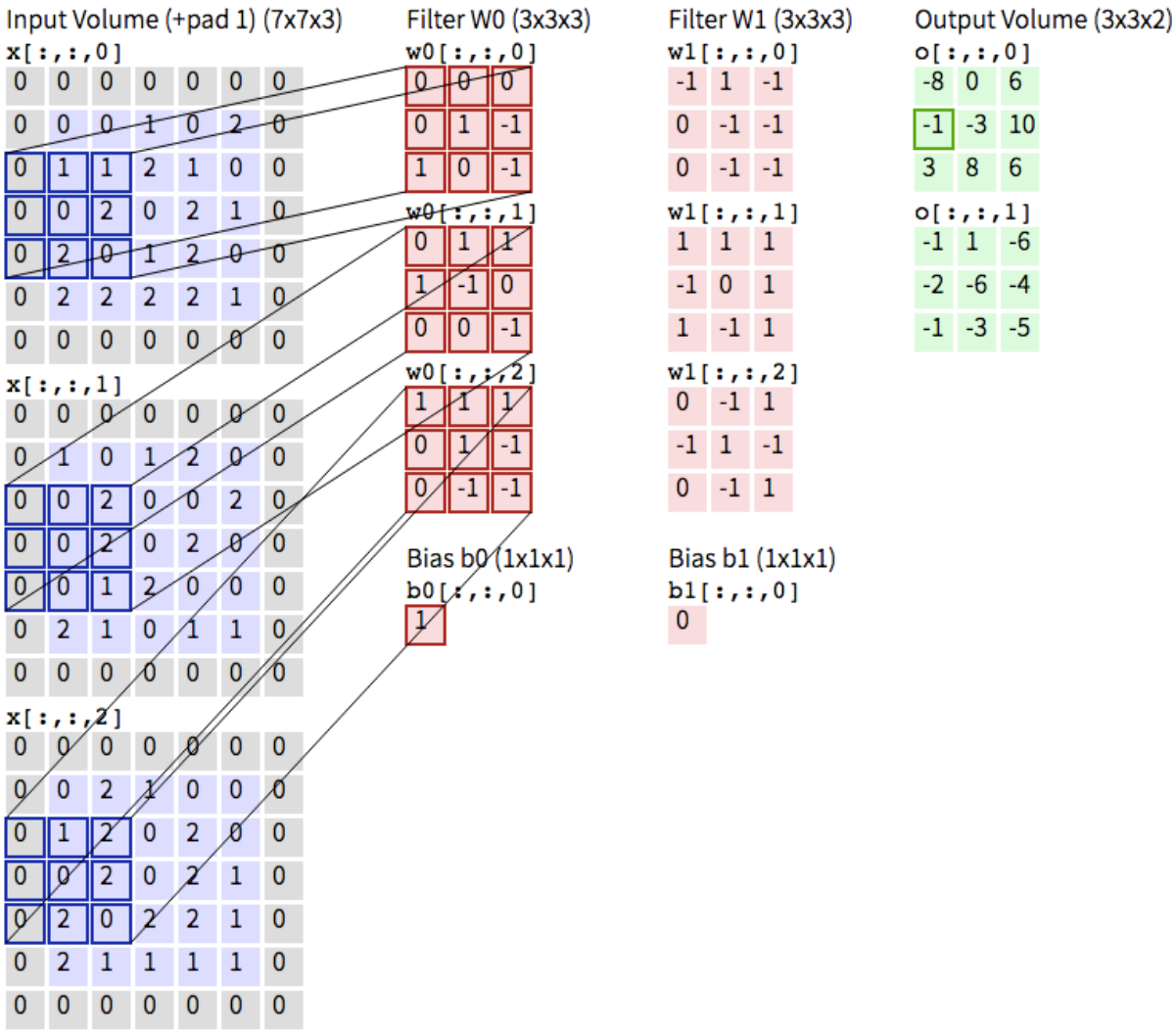
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

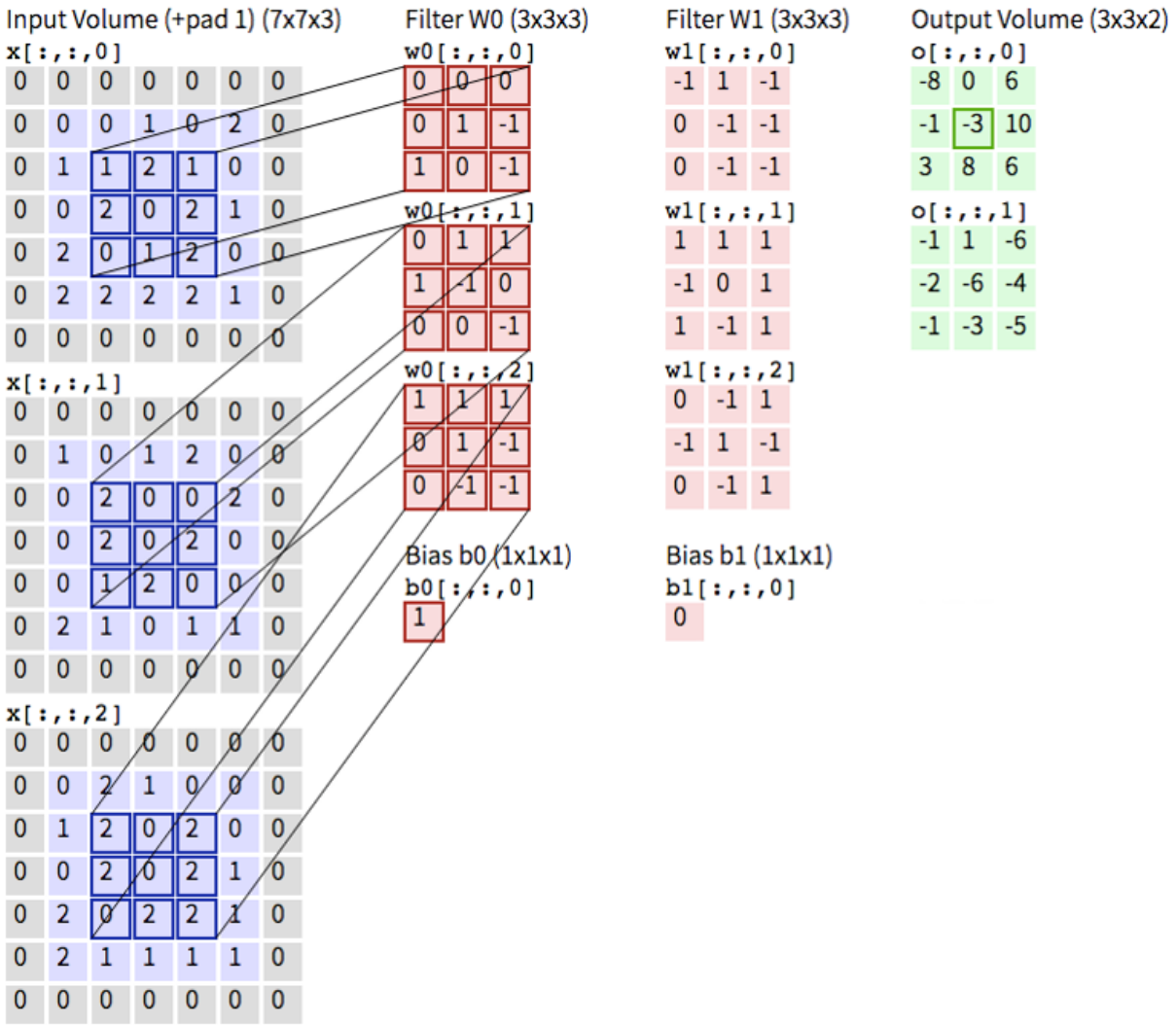
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

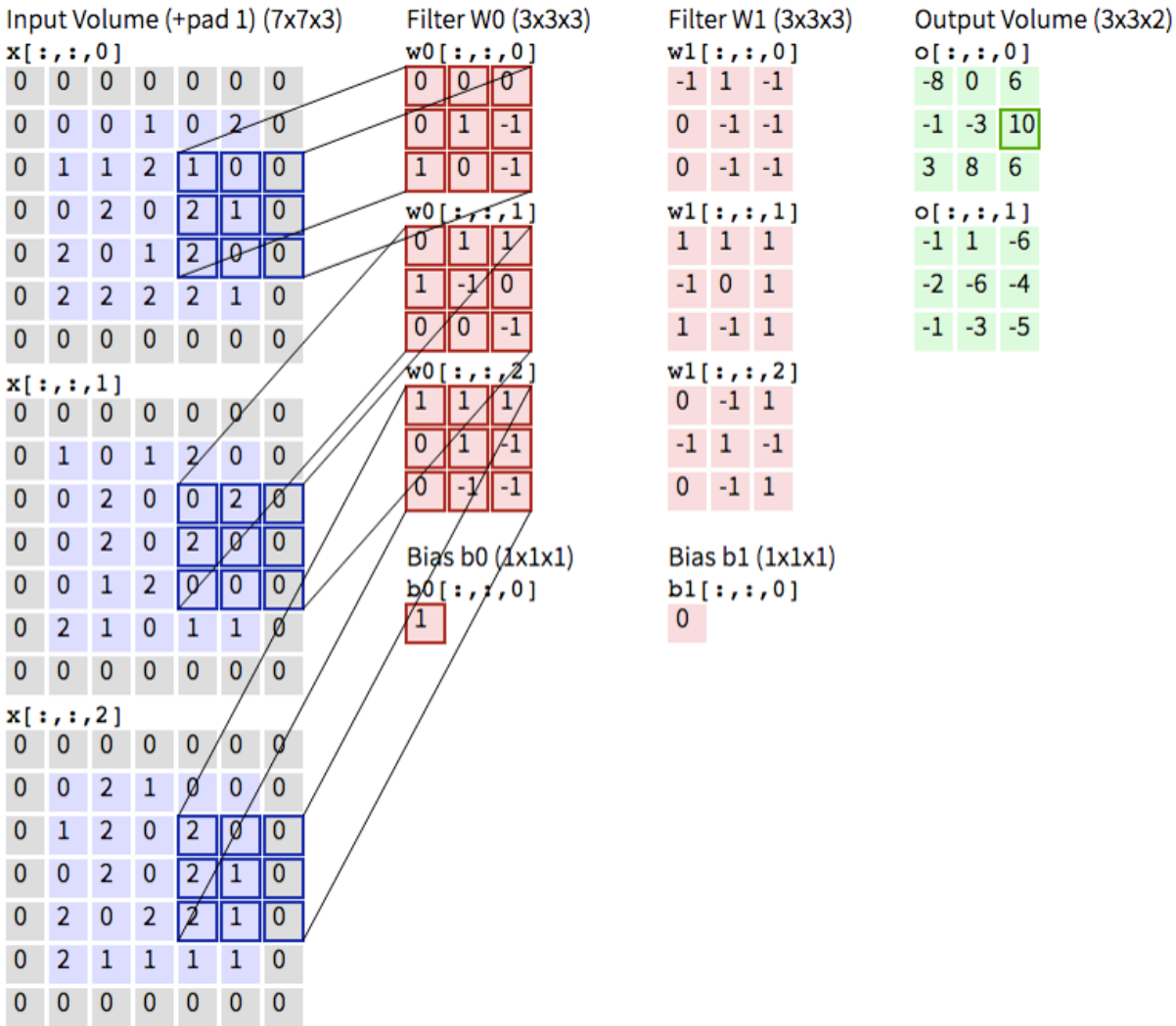
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

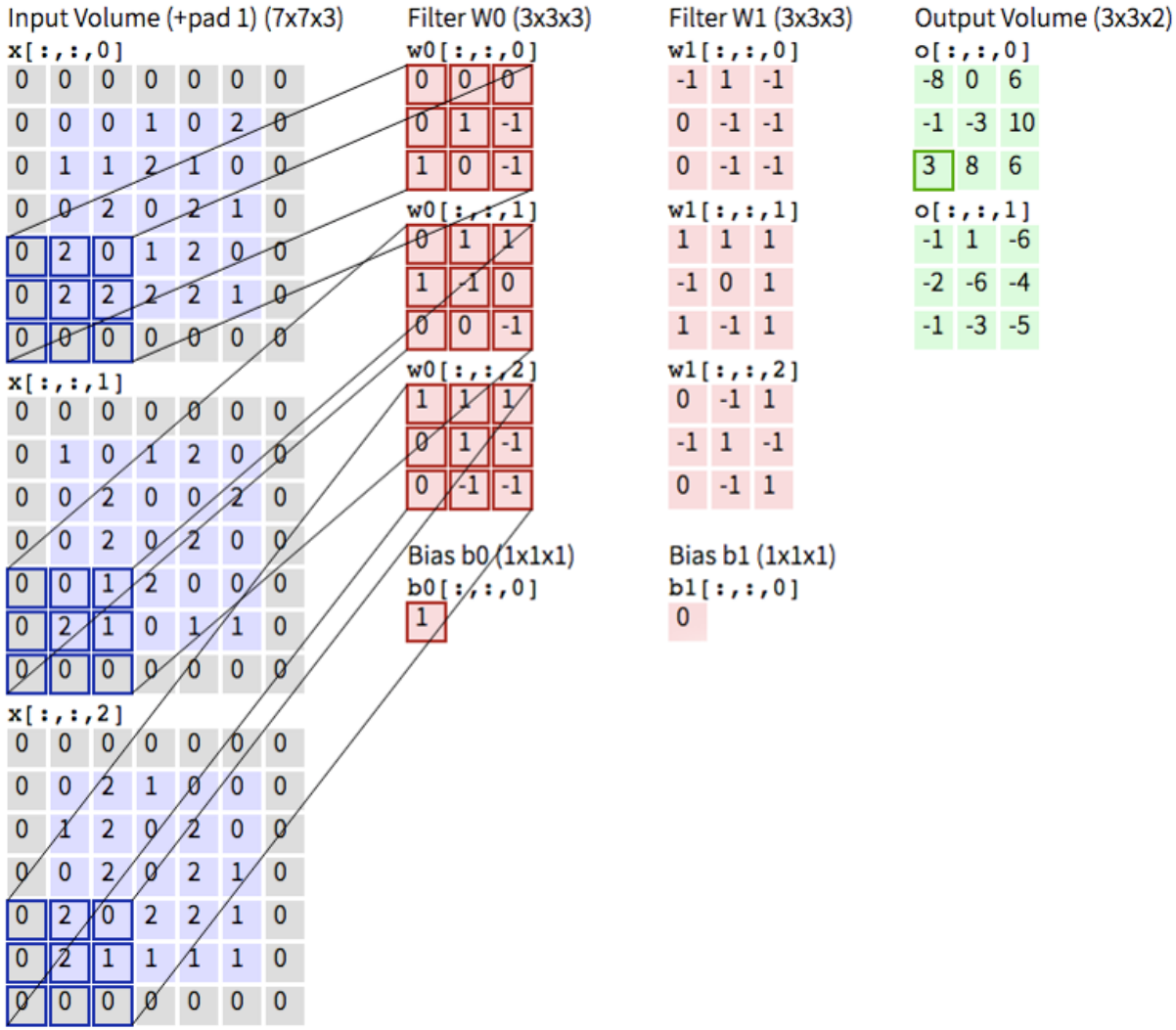
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

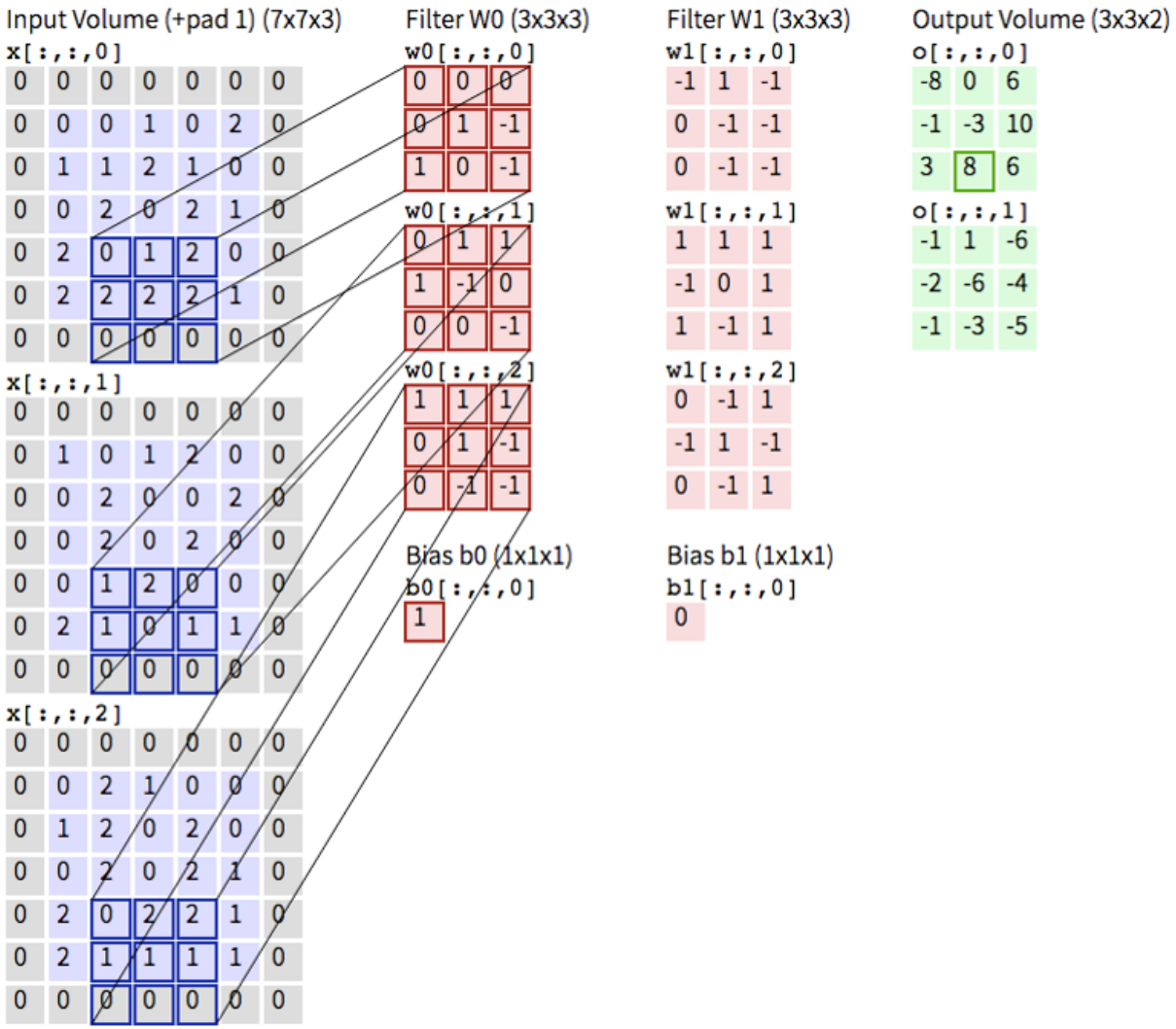
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

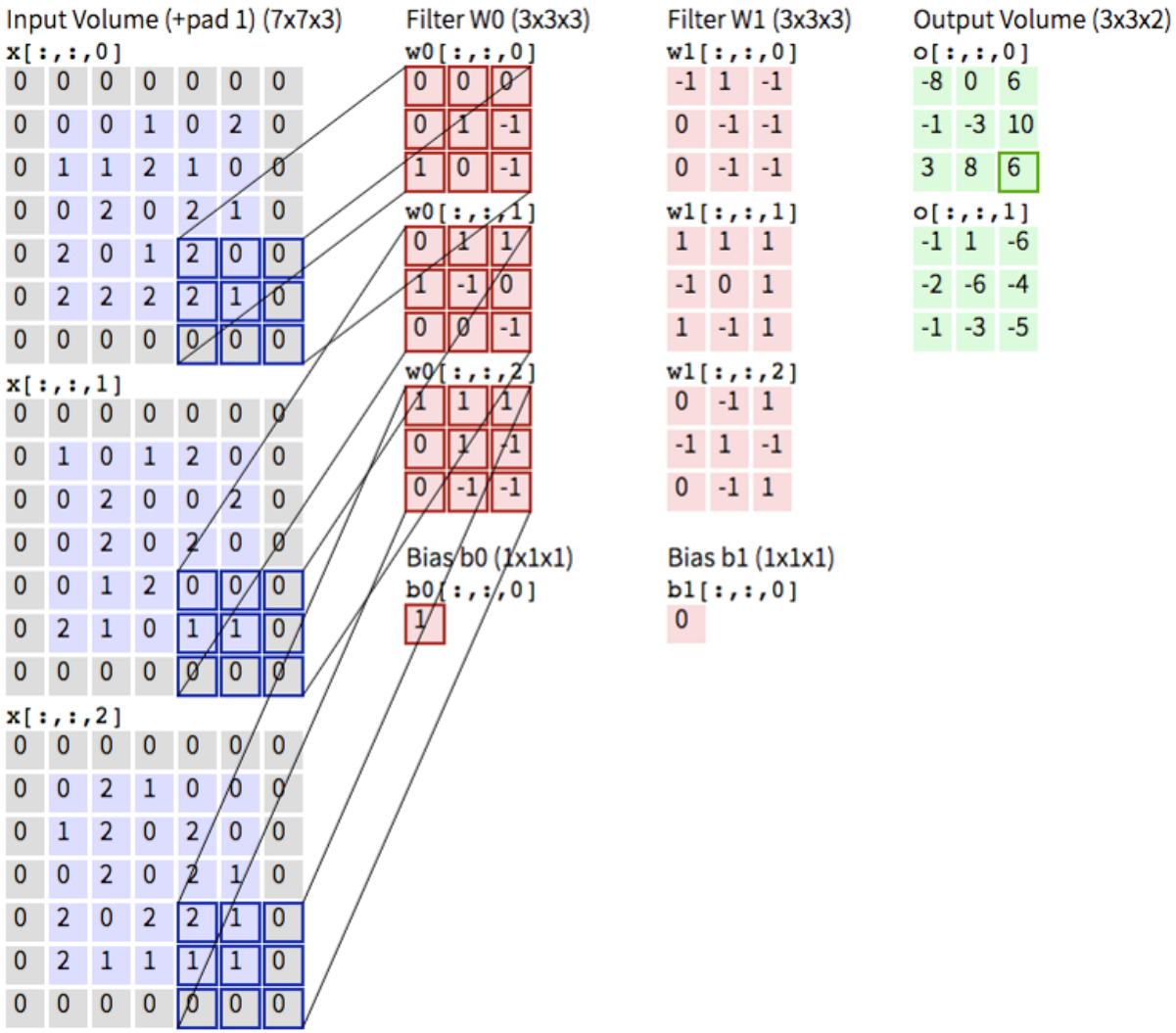
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

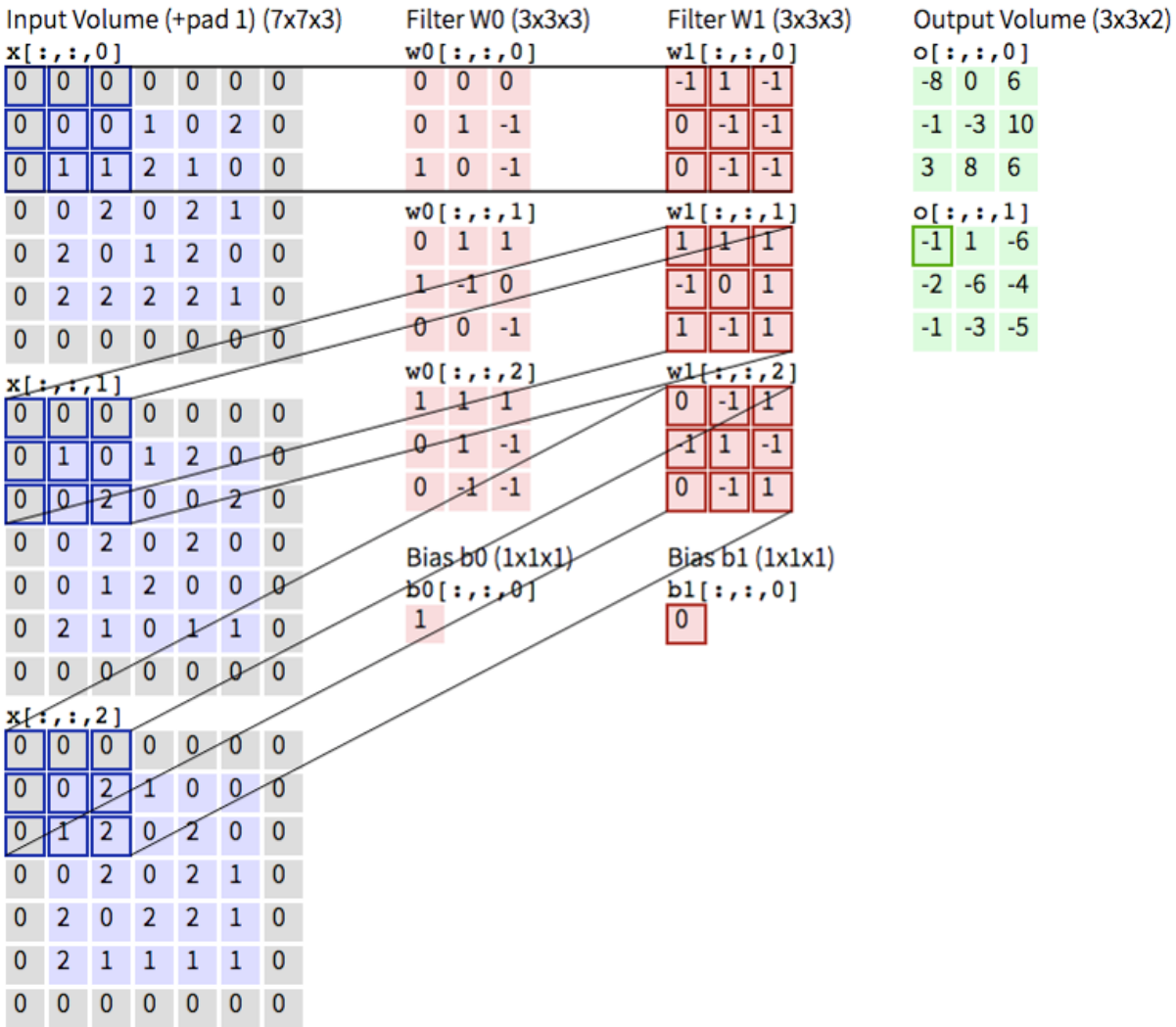
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

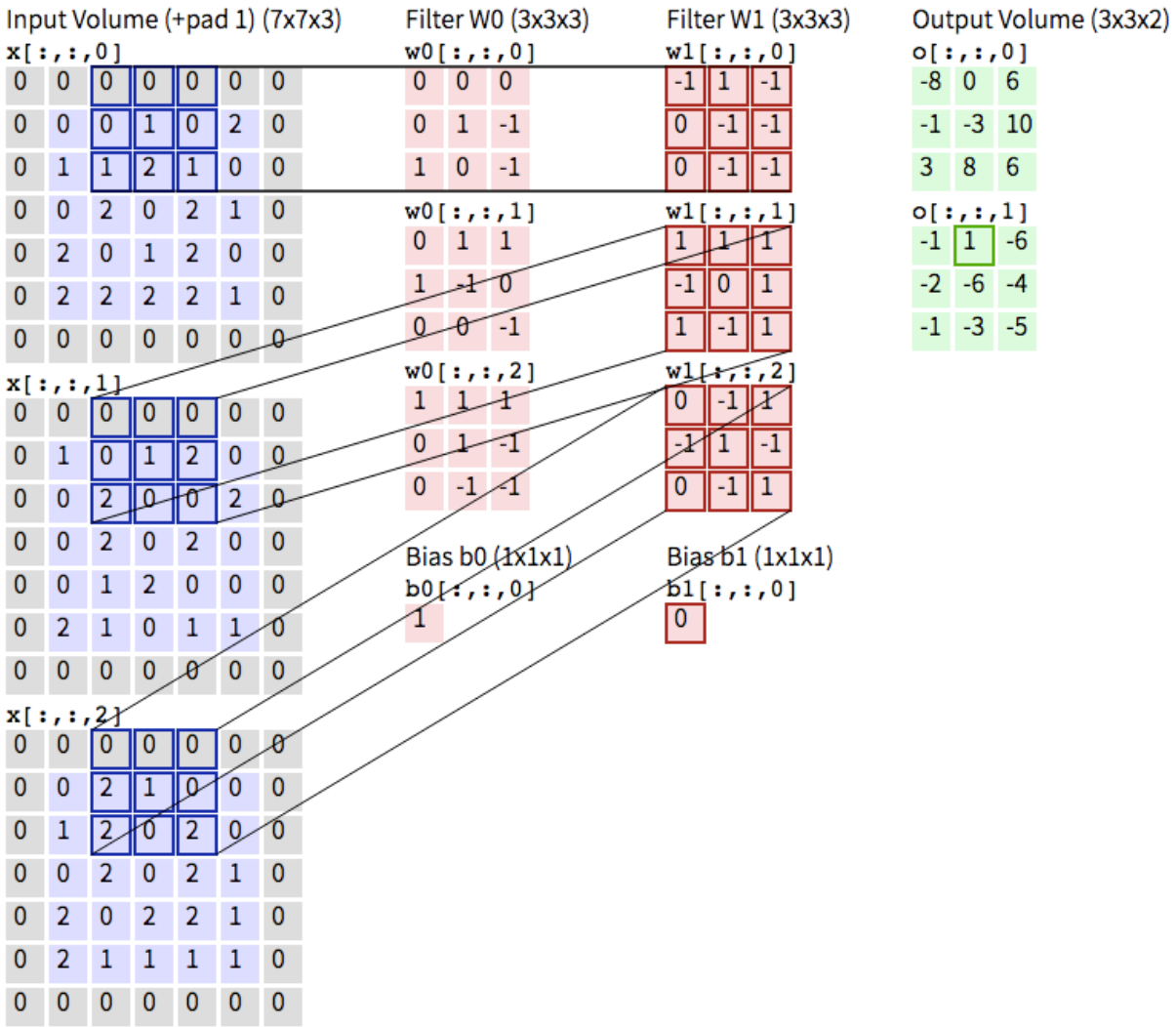
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

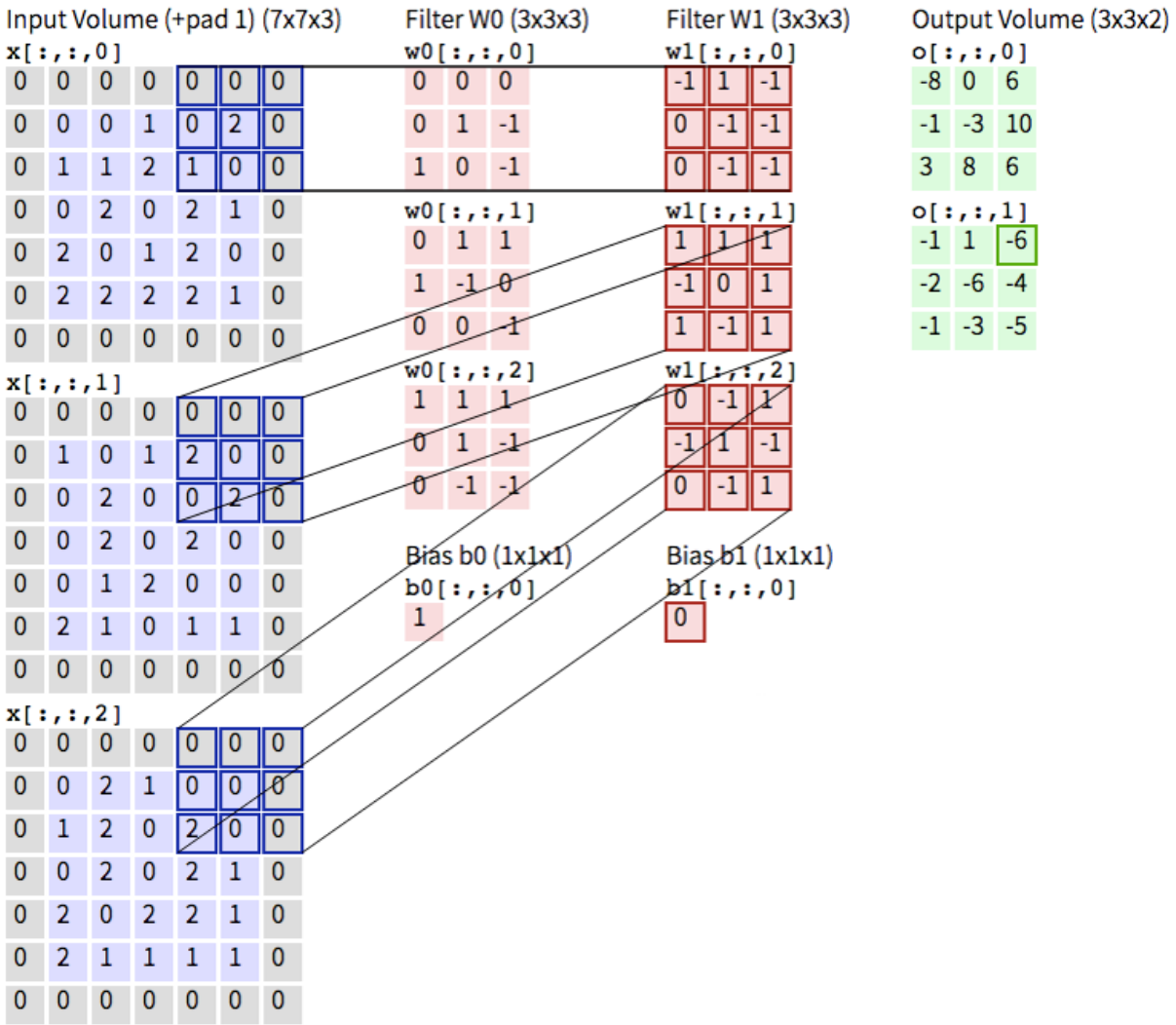
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

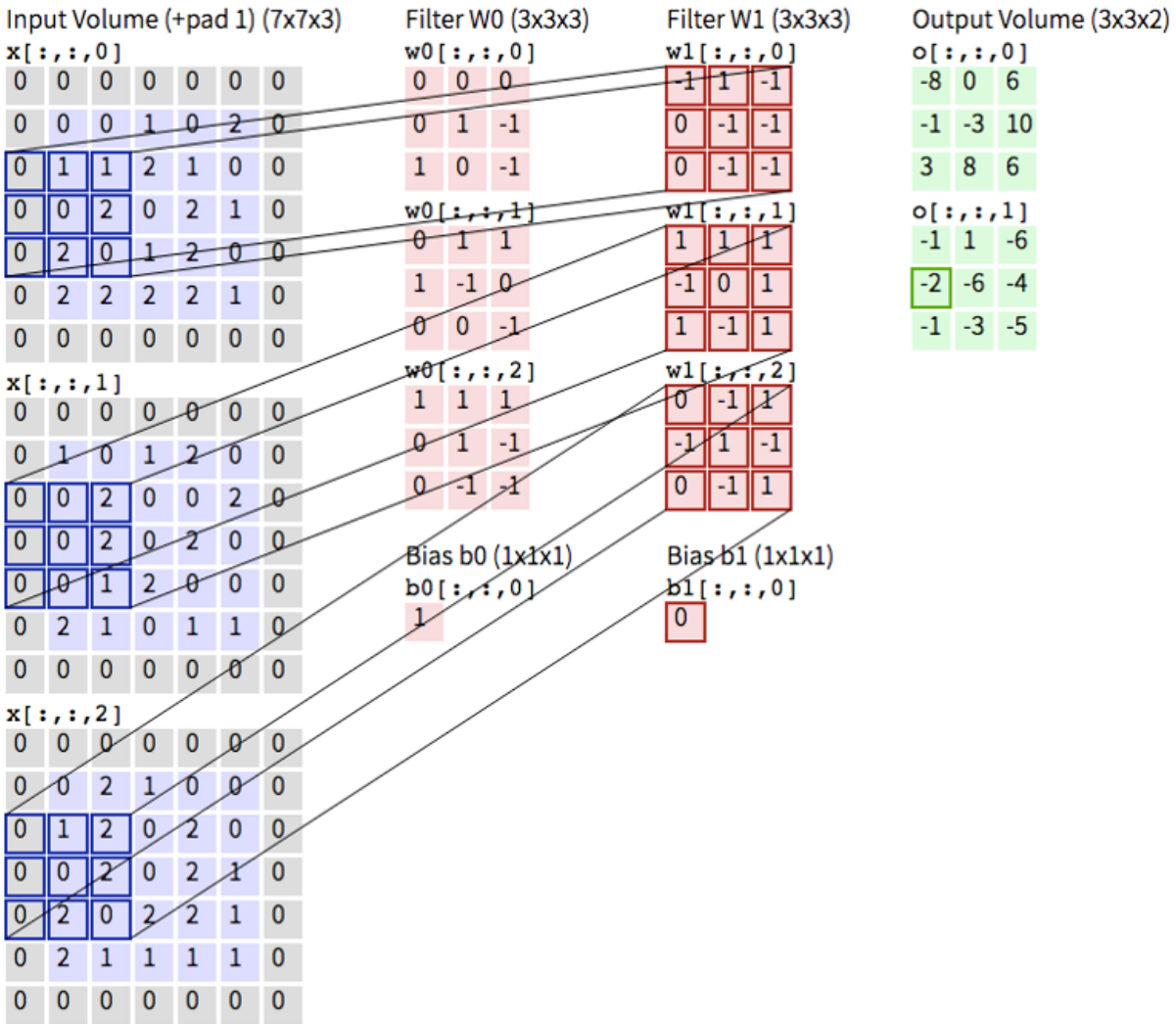
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

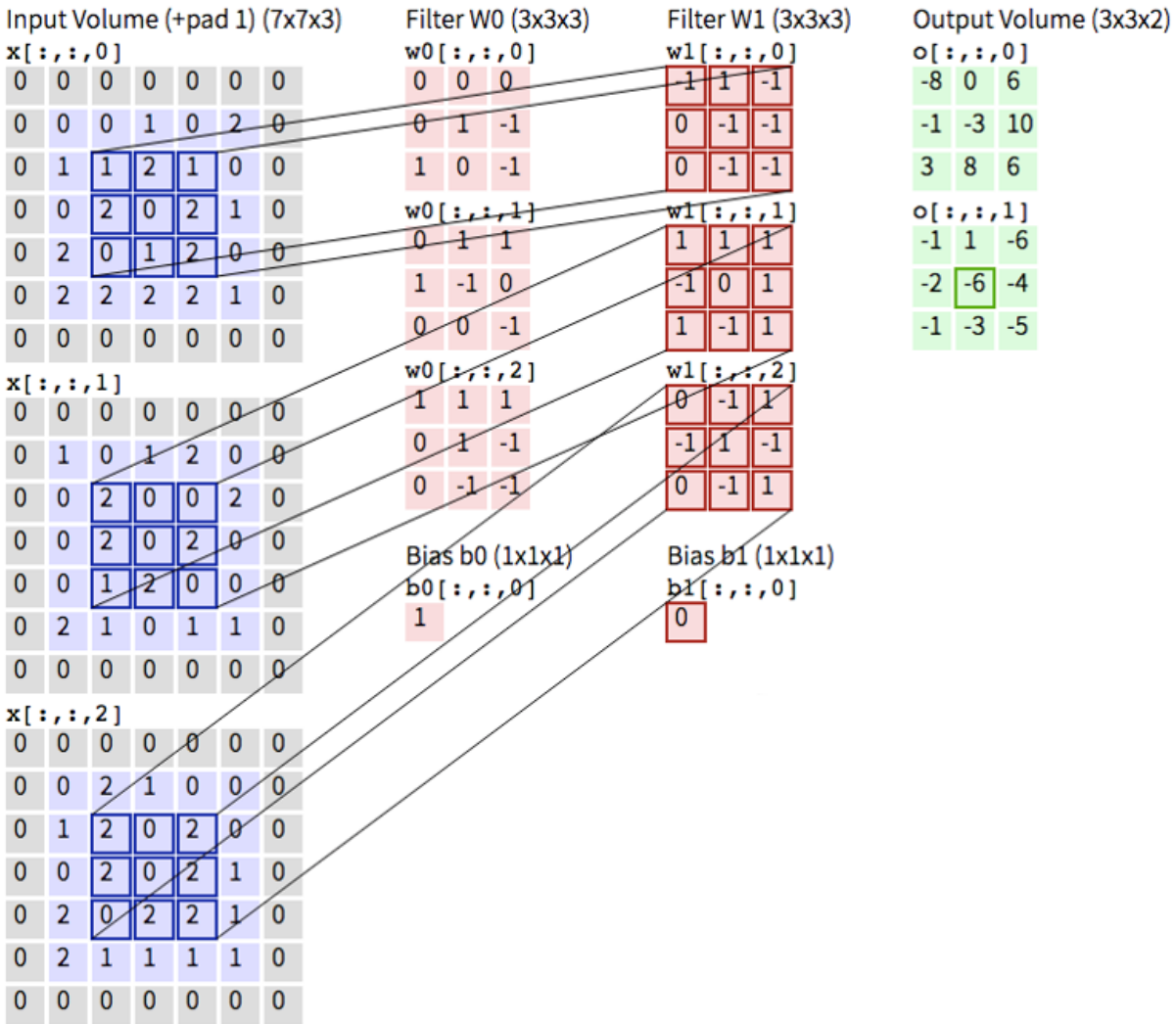
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

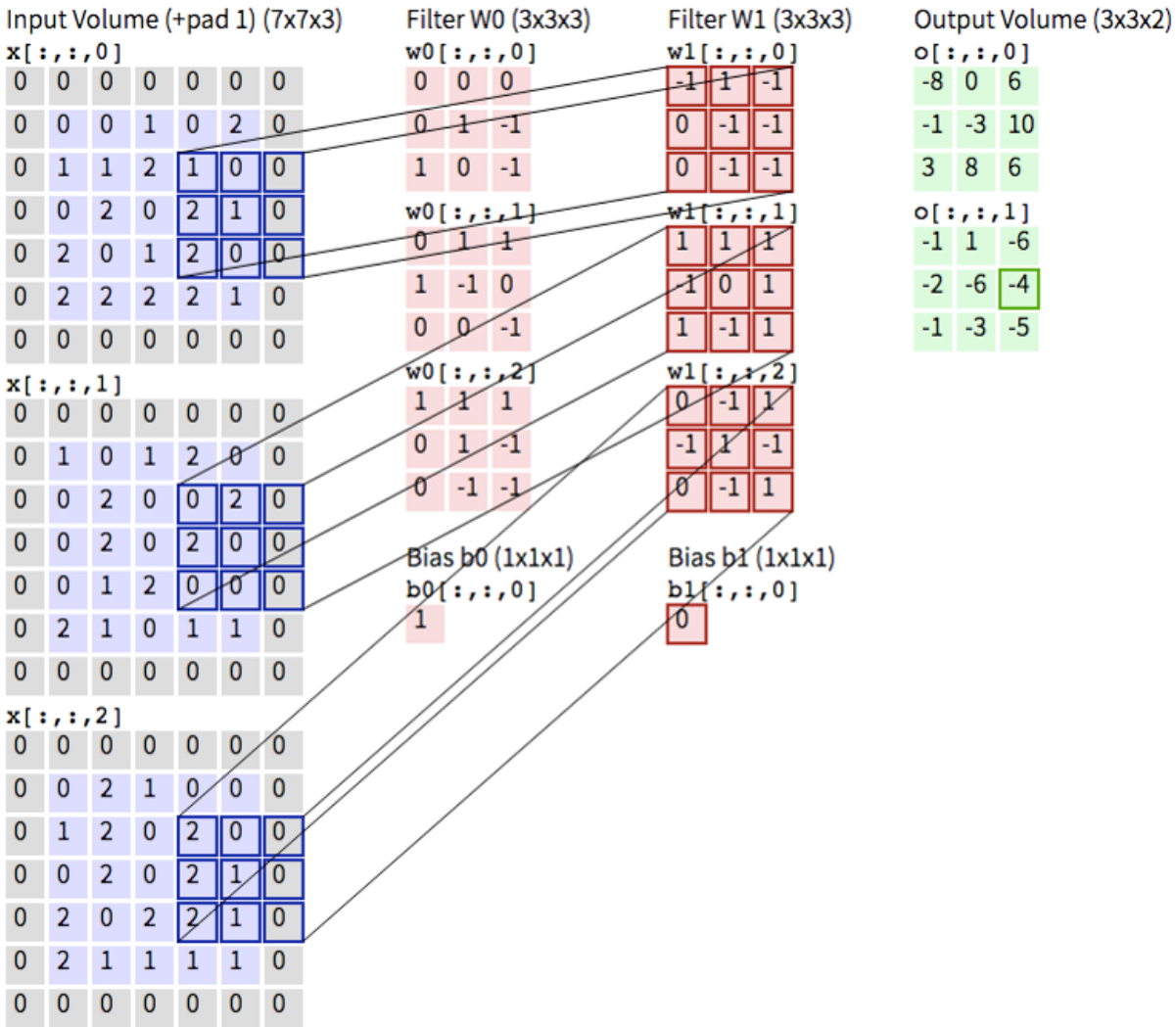
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

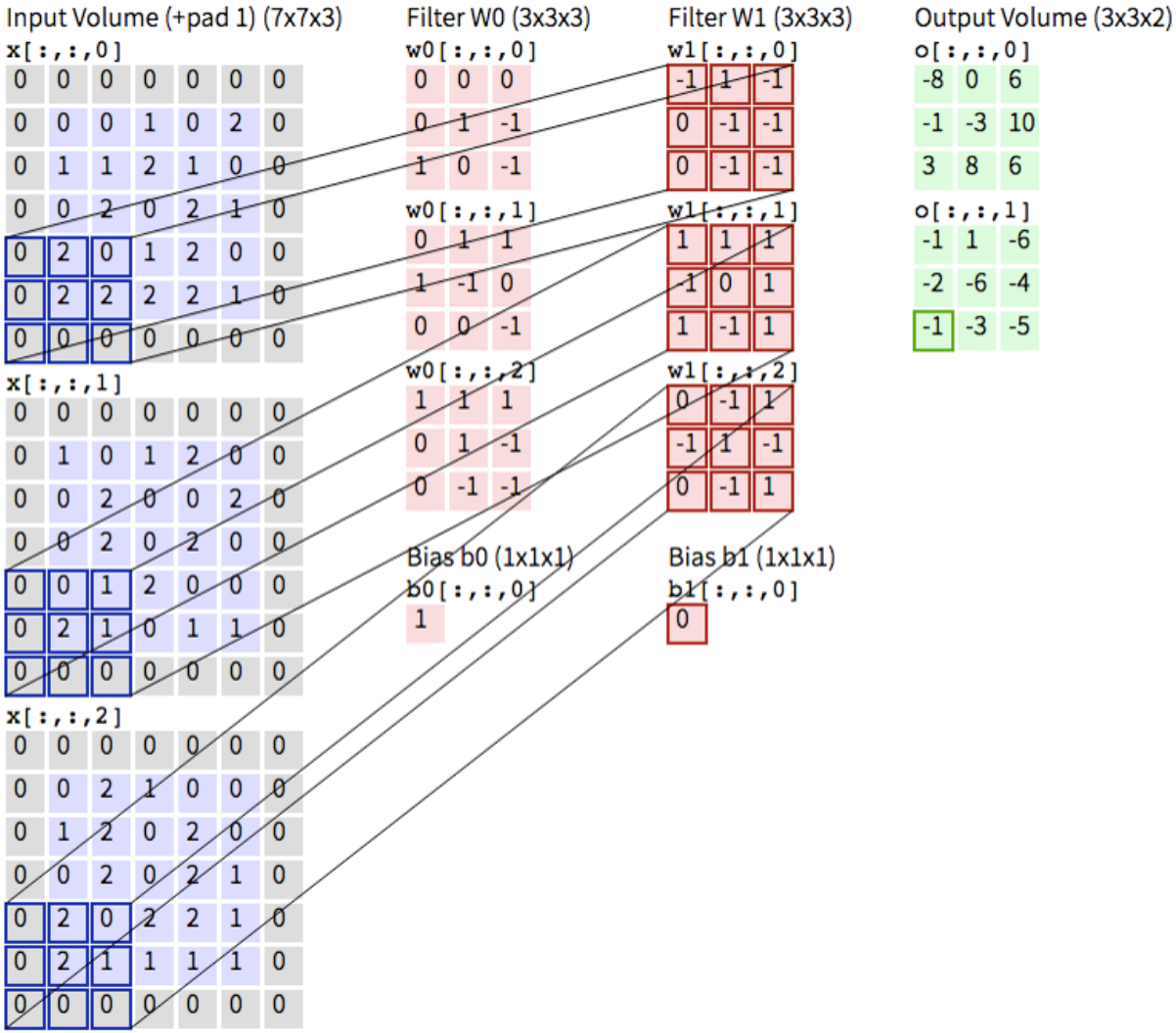
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

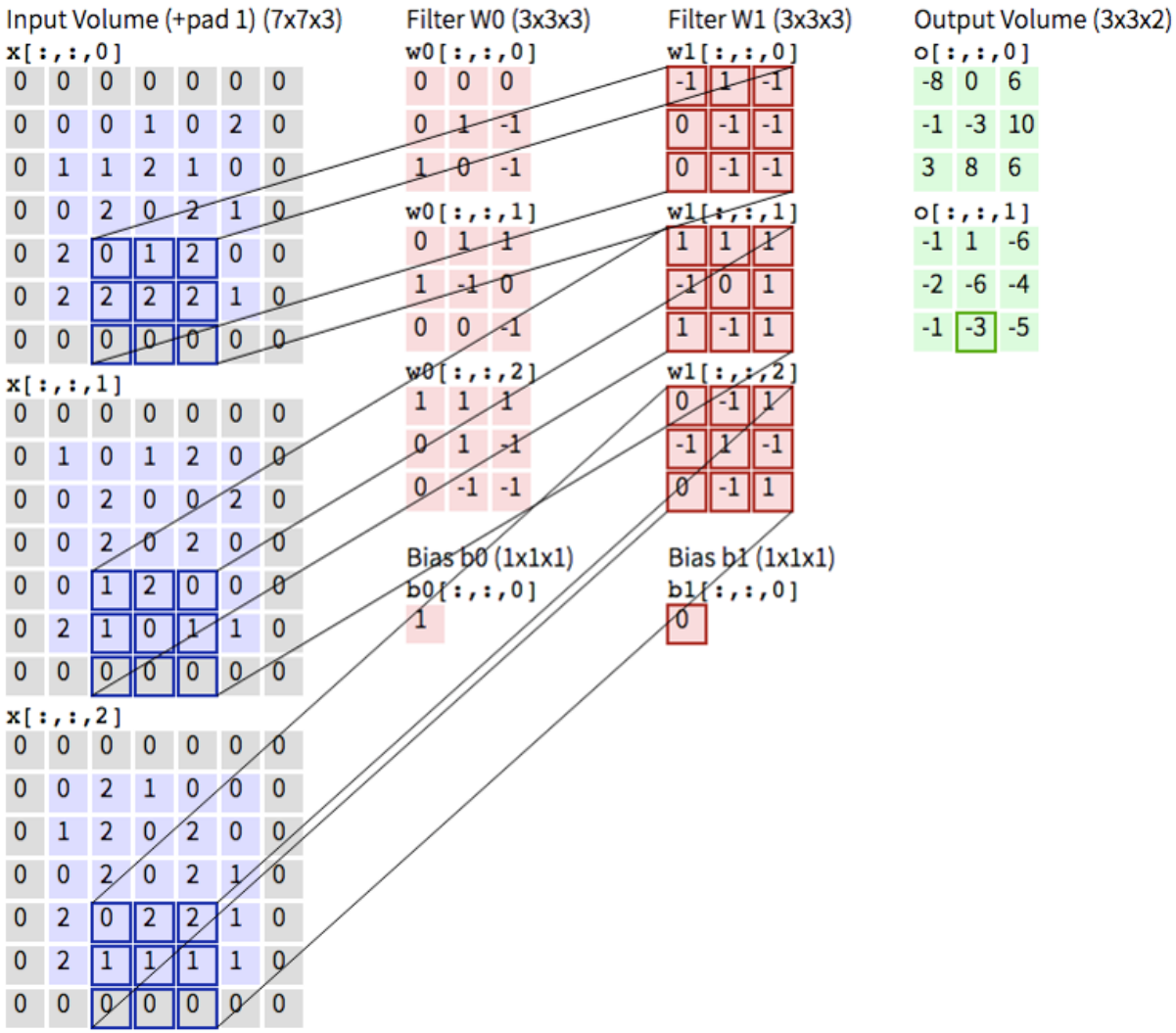
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

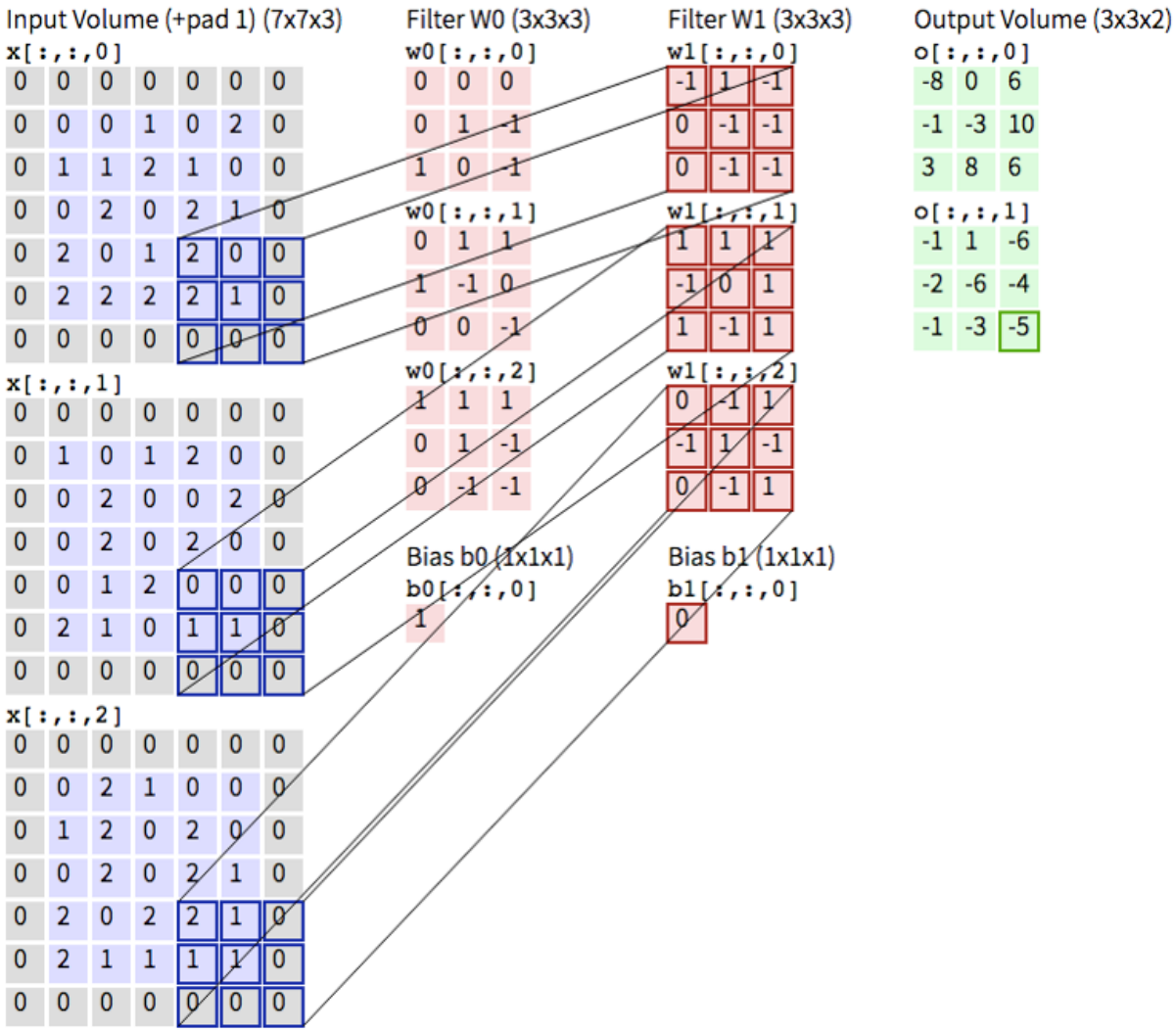
卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

卷积层：示例

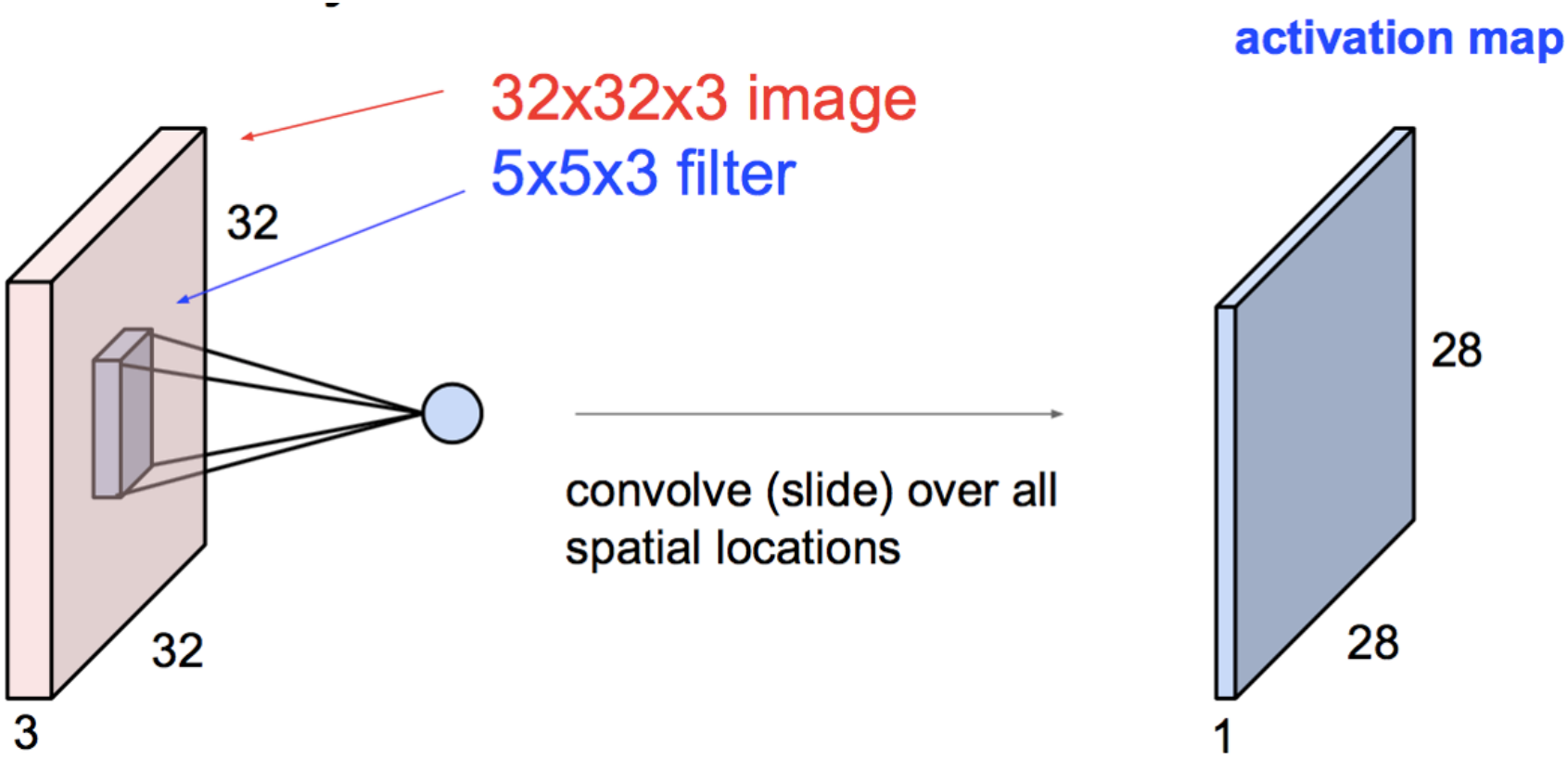

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

卷积层：示例


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 0 1 0 2 0
0 1 1 2 1 0 0
0 0 2 0 2 1 0
0 2 0 1 2 0 0
0 2 2 2 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 1 2 0 0
0 0 2 0 0 2 0
0 0 2 0 2 0 0
0 0 1 2 0 0 0
0 2 1 0 1 1 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 2 1 0 0 0
0 1 2 0 2 0 0
0 0 2 0 2 1 0
0 2 0 2 2 1 0
0 2 1 1 1 1 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 0
0 1 -1
1 0 -1
w0[:,:,1]
0 1 1
1 -1 0
0 0 -1
w0[:,:,2]
1 1 1
0 1 -1
0 -1 -1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 -1
0 -1 -1
0 -1 -1
w1[:,:,1]
1 1 1
-1 0 1
1 -1 1
w1[:,:,2]
0 -1 1
-1 1 -1
0 -1 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-8 0 6
-1 -3 10
3 8 6
o[:,:,1]
-1 1 -6
-2 -6 -4
-1 -3 -5

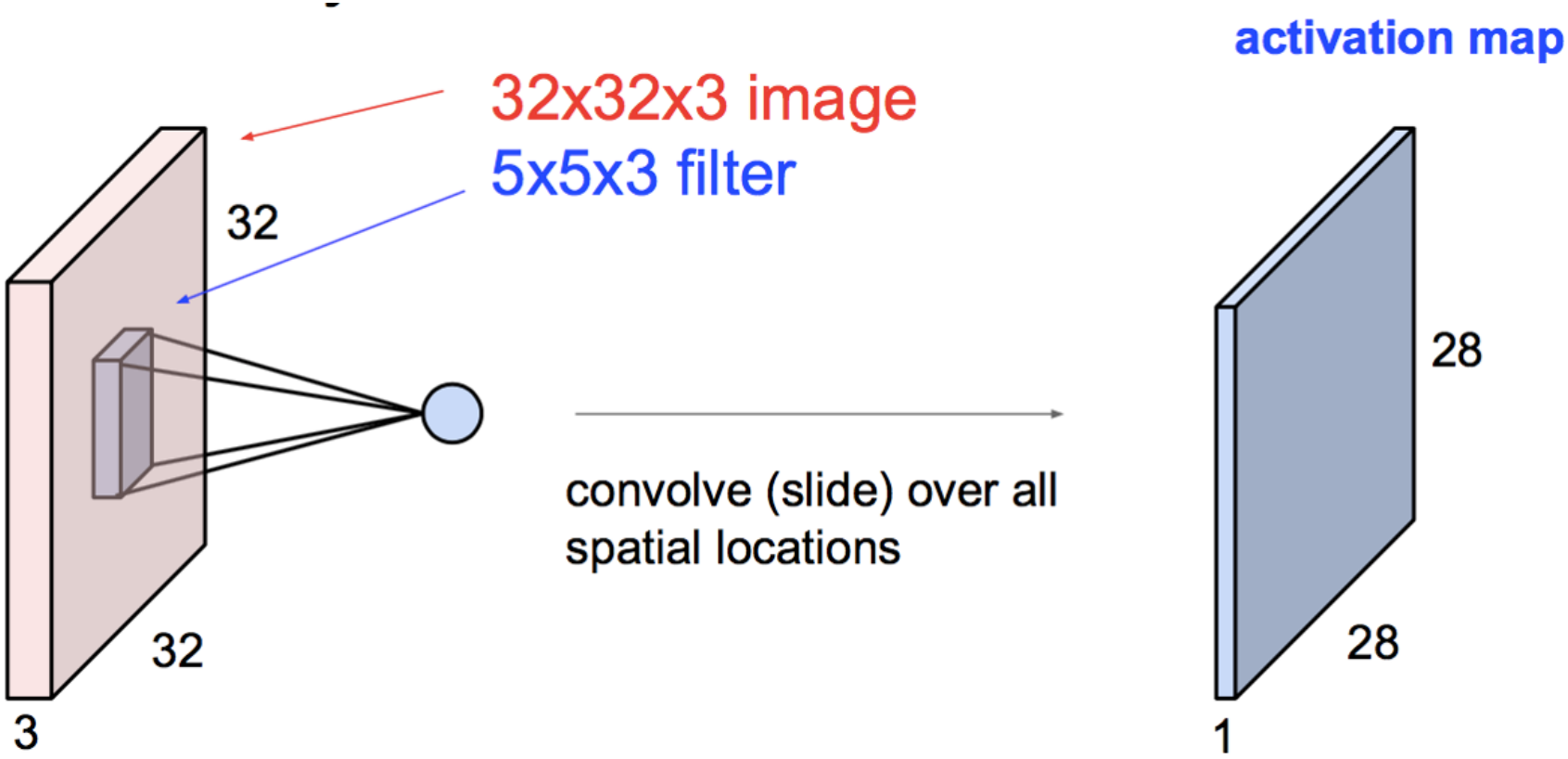
卷积层：局部连接（Local Connectivity）

每个标量输出（回顾：点积）从前一层的某个区域计算得到

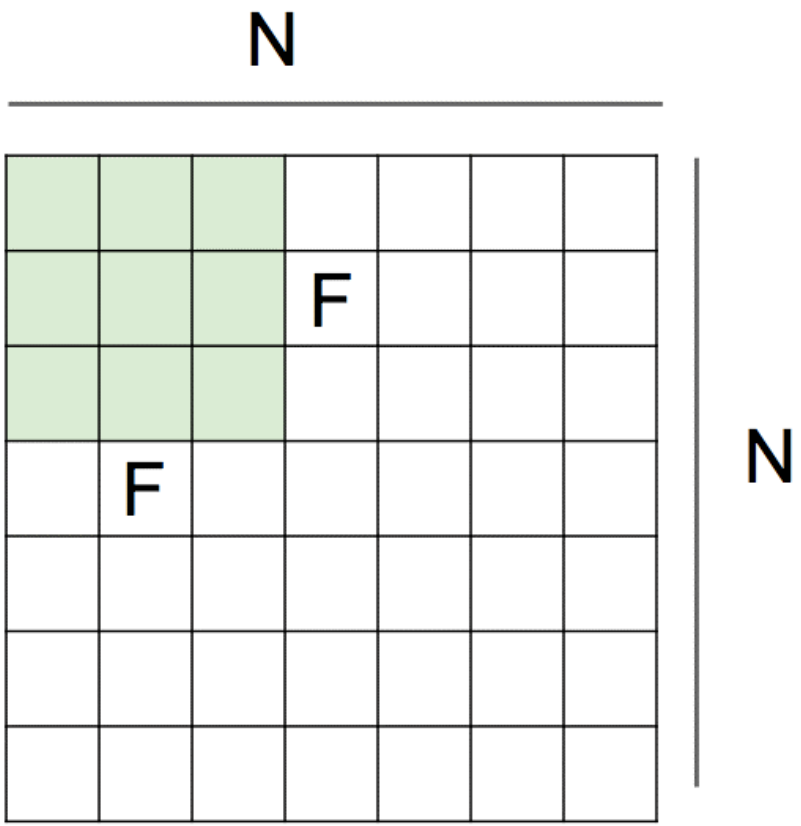


卷积层：共享权重

每个标量输出（回顾：点积）在计算时使用相同的权重 (filter)



卷积层：参数



假设：

n_1 个 $N \times N$ 特征图

n_2 个 $F \times F \times n_1$ filters

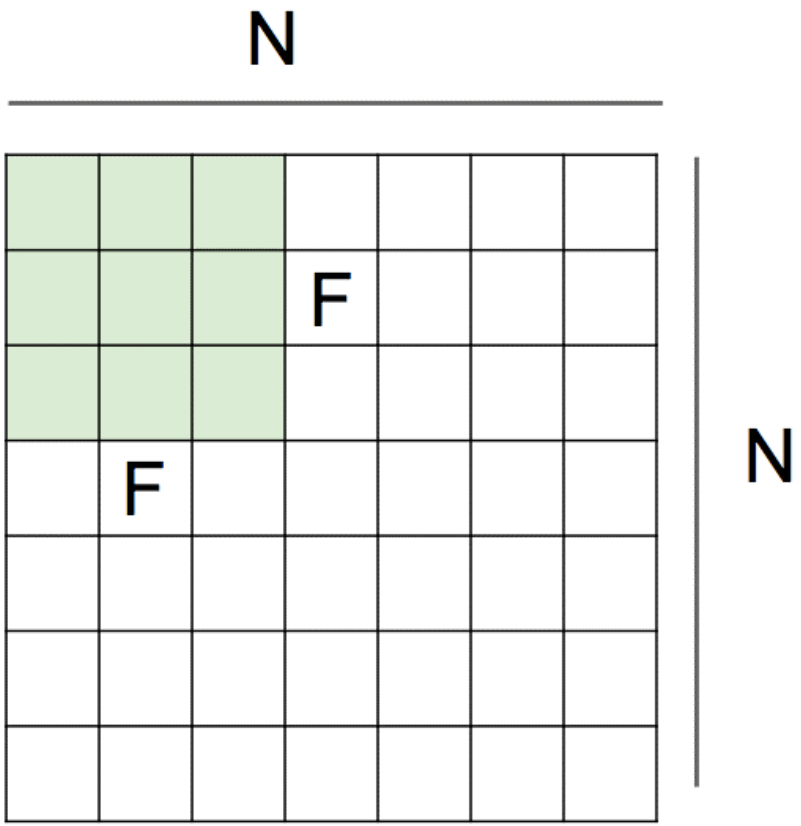
$\texttt{stride} = 2$

$\texttt{pad} = 1$

问题：

这个卷积层有多少参数？

卷积层：参数



假设：

n_1 个 $N \times N$ 特征图

n_2 个 $F \times F \times n_1$ filters

$\texttt{stride} = 2$

$\texttt{pad} = 1$

问题：

这个卷积层有多少参数？

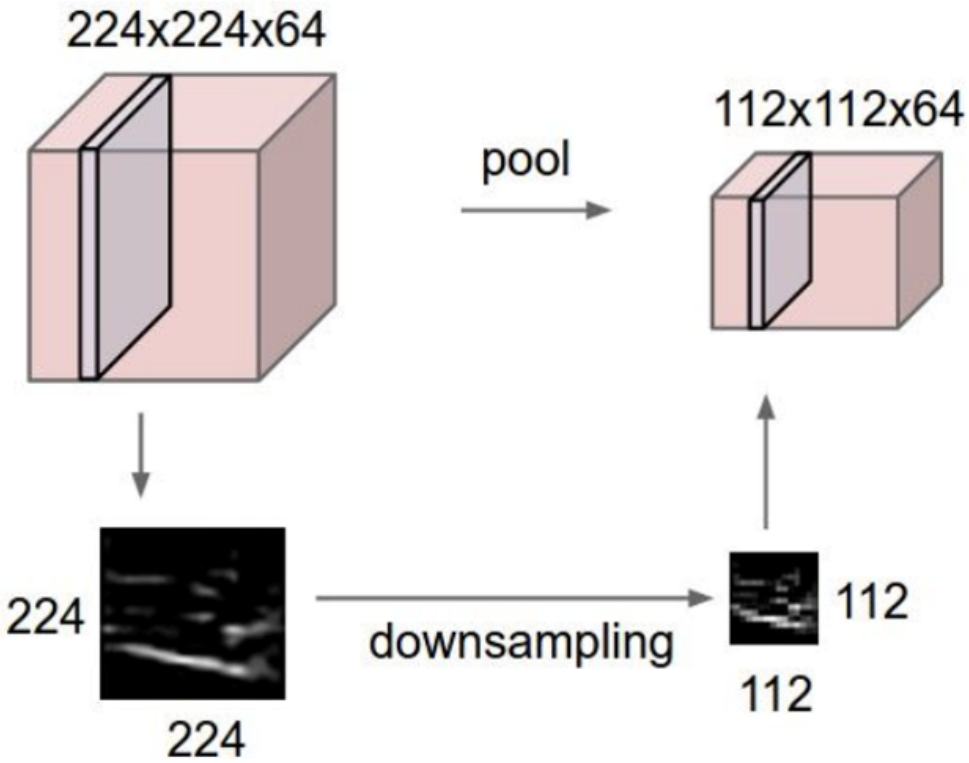
$\#\texttt{params} = n_1 \times n_2 \times F \times F$!

$\#\texttt{params}$ 和特征图大小无关!

但是请注意 $\#\texttt{computation}$ 与之有关!

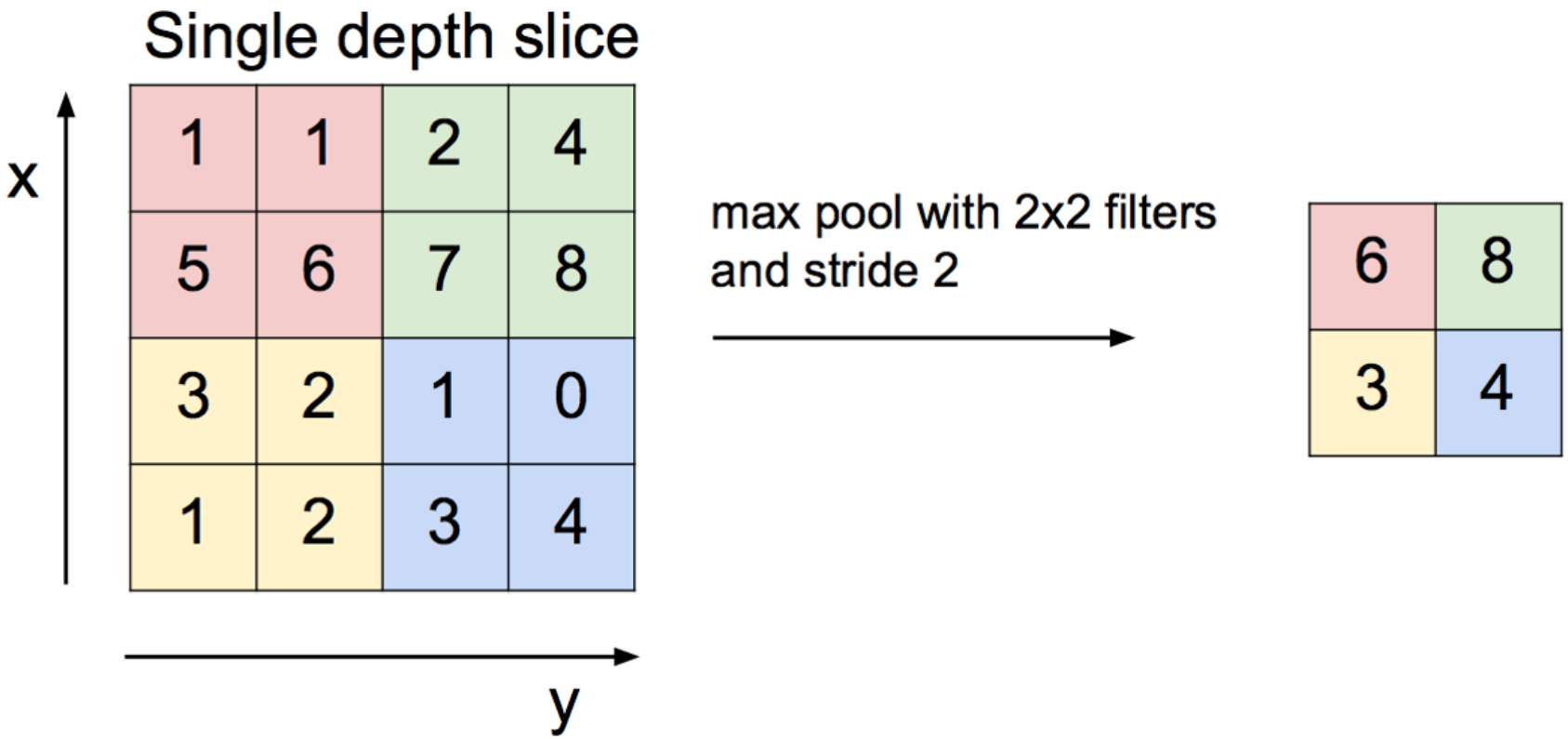
池化（Pooling）层

- 使特征表示 (representations) 更小、更容易管理
- 在每个激活图上单独运算



池化层：最大池化（Max Pooling）

这里，池化大小（filter size）为 2×2，步长（stride）也是 2×2，因此池化没有重叠（惯例），当然有重叠的池化也是可能的

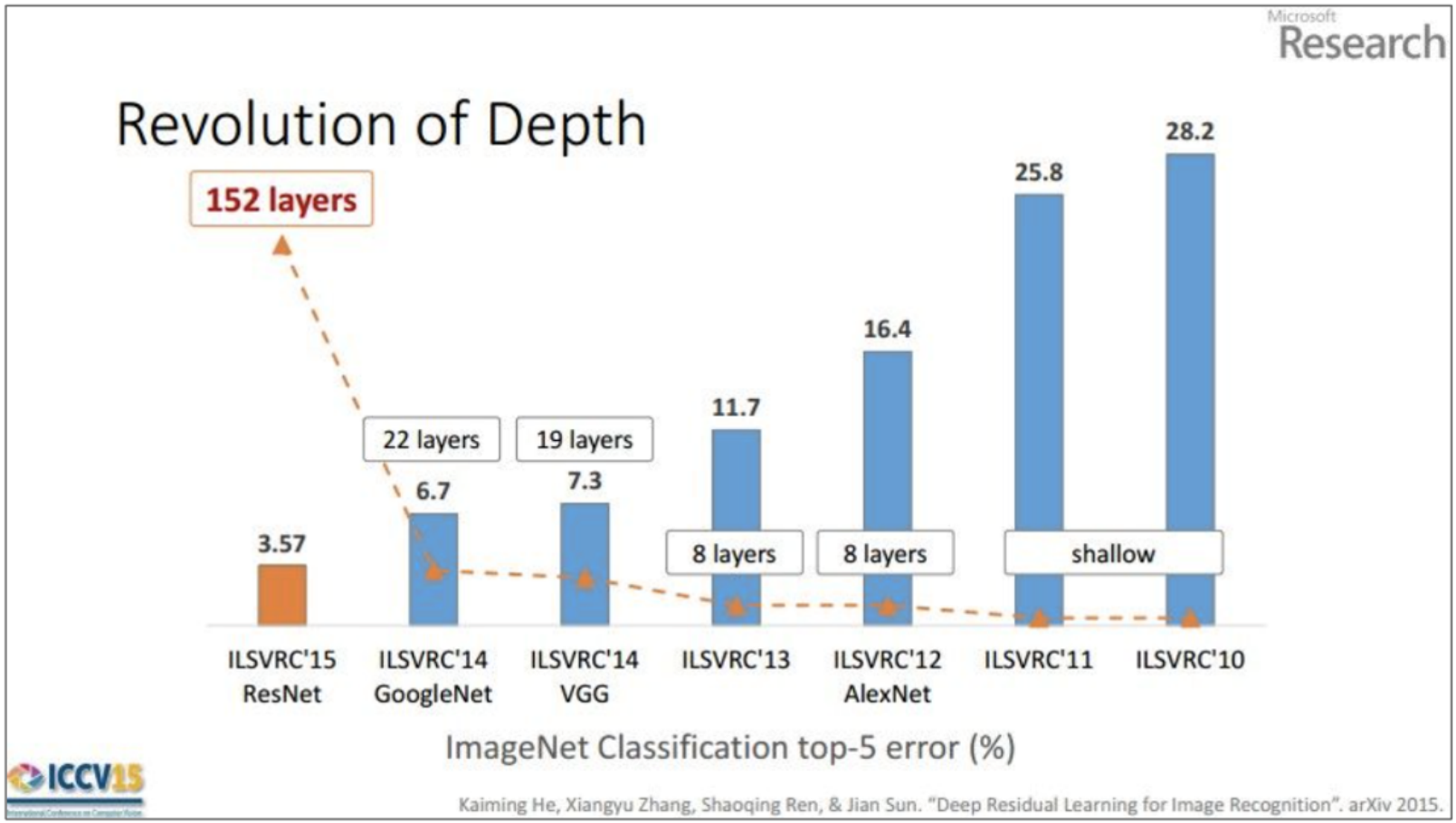


池化层：其他策略

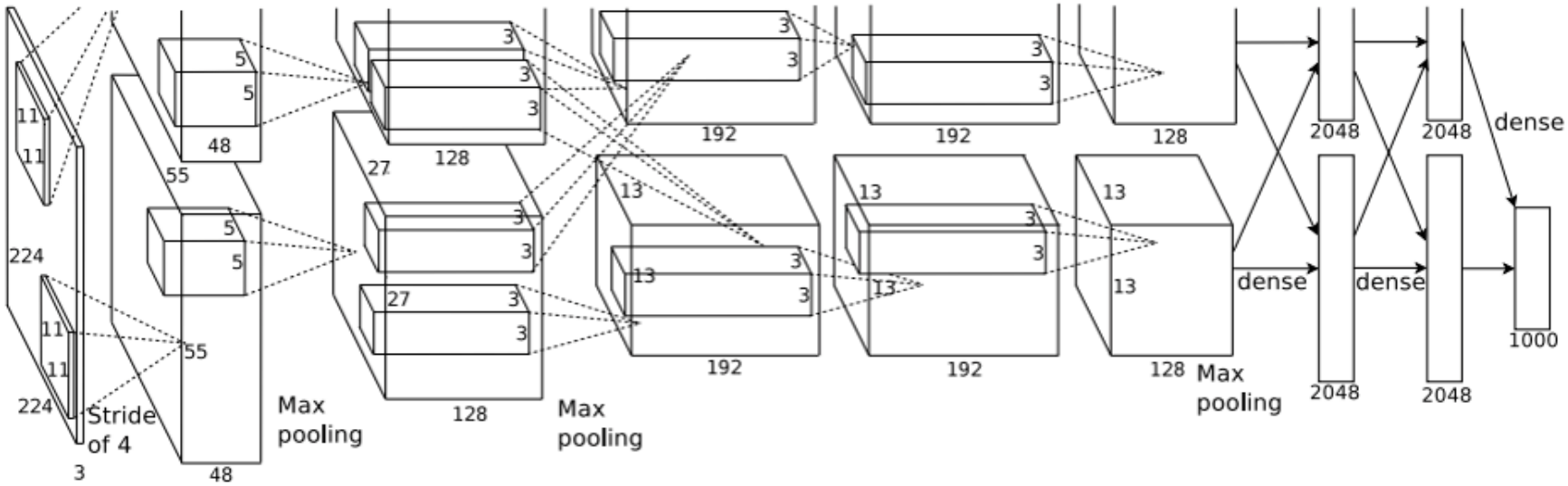
- 平均池化（Average Pooling）
  - 最大 $\rightarrow$ 平均
- 随机池化（Stochastic Pooling）
  - 最大 $\rightarrow$ 随机挑选

- 采用固定位置的卷积来代替
  - 最大 $\rightarrow$ 固定位置

几乎所有最重要的 CNN 模型都来自该比赛



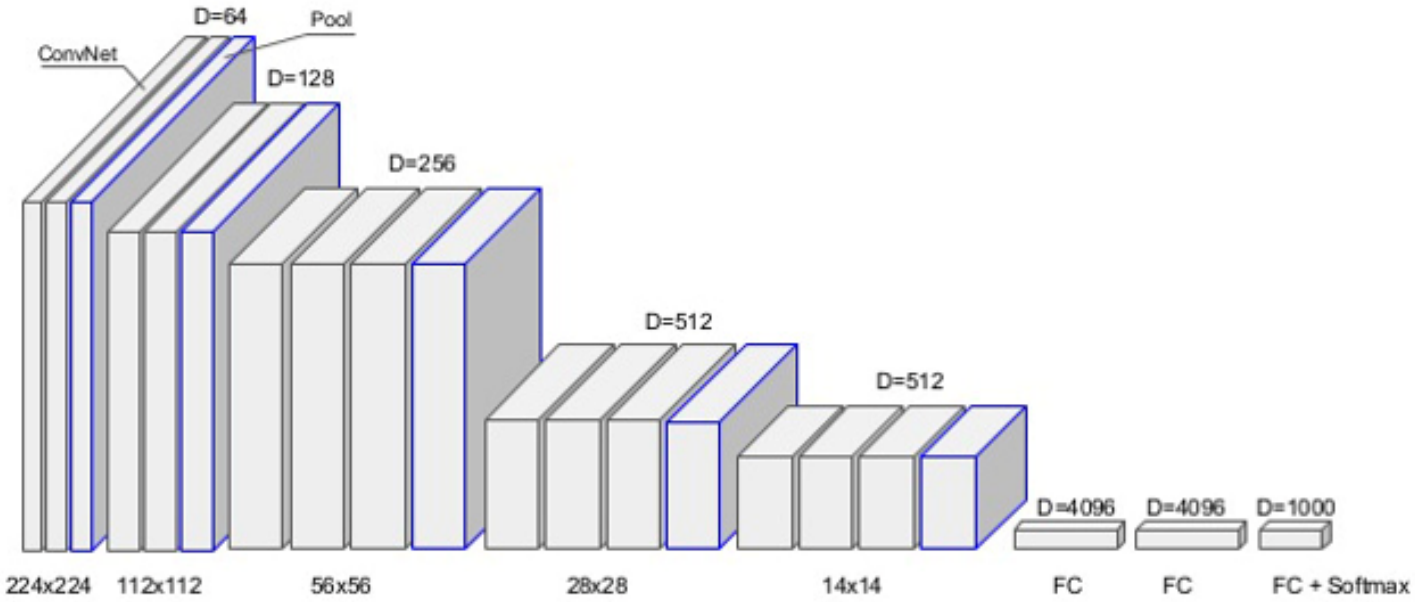
案例研究：AlexNet



Krizhevsky et al, 2012

案例研究：VGGNet

- 所有卷积层都采用 3×3 的小卷积核
- 许多卷积层堆叠到一起



Simonyan et al, 2014

案例研究：VGGNet

(not counting biases)

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 ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64	conv3-64	conv3-64	cc
conv3-64	conv3-64	conv3-64	cc
maxpool			
conv3-128	conv3-128	conv3-128	co
conv3-128	conv3-128	conv3-128	co
maxpool			
conv3-256	conv3-256	conv3-256	co
conv3-256	conv3-256	conv3-256	co
	conv1-256	**conv3-256**	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	**conv3-512**	co
			co
maxpool			
conv3-512	conv3-512	conv3-512	co
conv3-512	conv3-512	conv3-512	co
	conv1-512	**conv3-512**	co
			co
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

案例研究：VGGNet

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
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

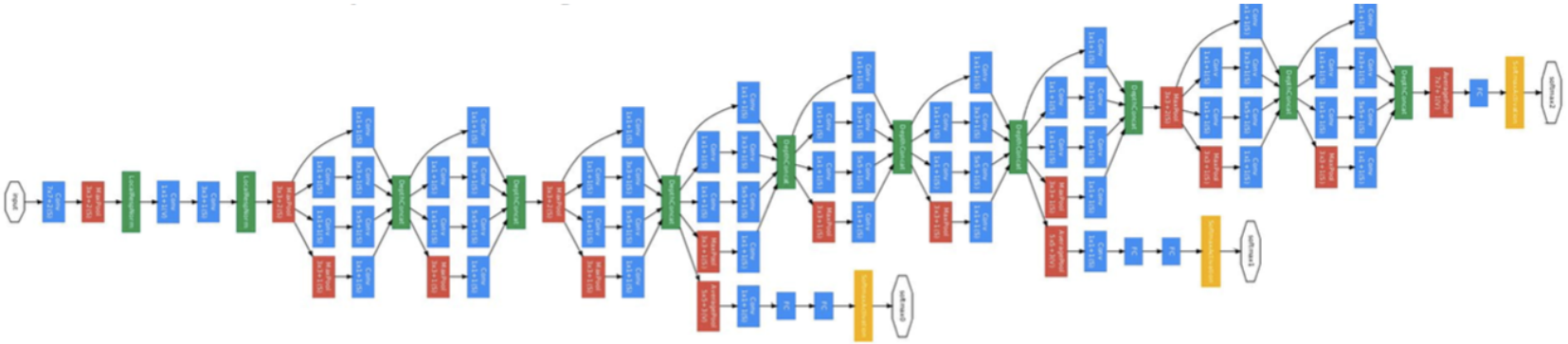
TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

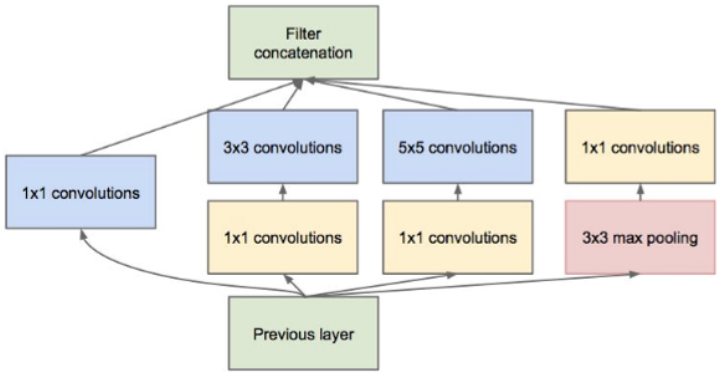
Note:

Most memory is in early CONV

Most params are in late FC

案例研究：GoogLeNet





Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Szegedy et al, 2014

案例研究：GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Fun features:

- Only 5 million params! (Removes FC layers completely)

Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

神经网络的深度

- 神经网络的深度很重要
- 然后，学习更好的网络和堆叠更多层同样容易吗？

— 并不是!

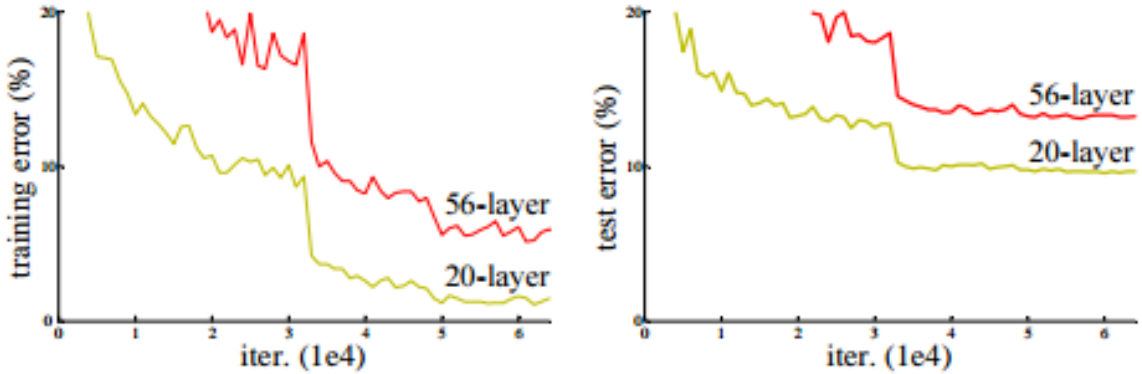
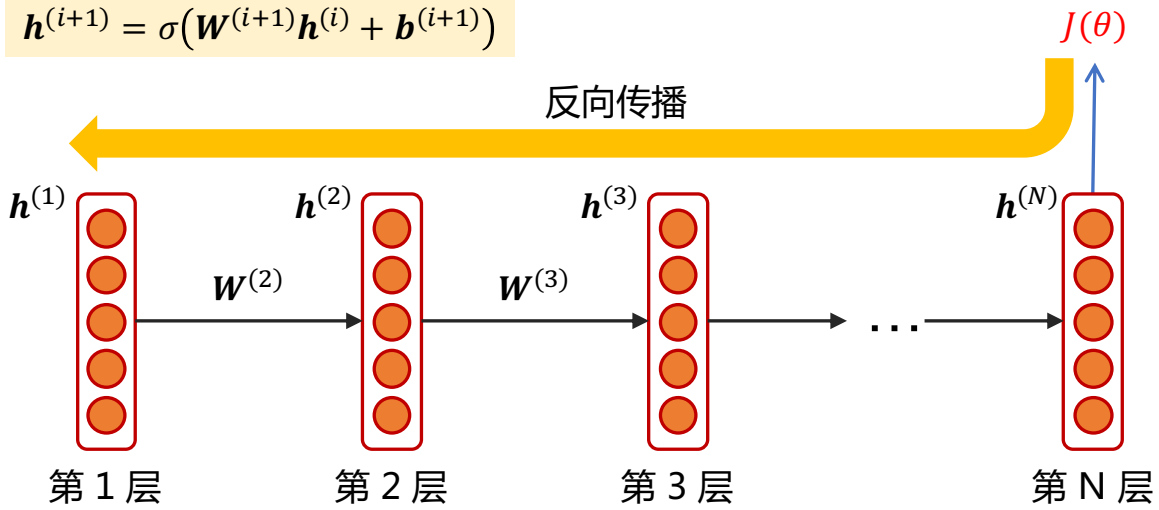


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

更深神经网络面临的挑战

- 更深的神经网络很有吸引力
  - 具有更强的表达复杂概念的能力
- 更深的模型同时会引入新的挑战
  - 反向传播中的梯度消失（Vanishing gradients）

$$\boldsymbol{h}^{(i+1)} = \sigma\left(\boldsymbol{W}^{(i+1)}\boldsymbol{h}^{(i)} + \boldsymbol{b}^{(i+1)}\right)$$



$$\frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}} = \mathrm{diag}\left(\sigma'\left(\boldsymbol{W}^{(i+1)}\boldsymbol{h}^{(i)} + \boldsymbol{b}^{(i+1)}\right)\right)\boldsymbol{W}^{(i+1)}$$

$$\frac{\partial J(\theta)}{\partial \boldsymbol{h}^{(n)}} = \frac{\partial J(\theta)}{\partial \boldsymbol{h}^{(N)}} \prod_{n \le i < N} \frac{\partial \boldsymbol{h}^{(i+1)}}{\partial \boldsymbol{h}^{(i)}}$$

更深神经网络面临的挑战

- 更深的神经网络很有吸引力
  - 具有更强的表达复杂概念的能力
- 更深的模型同时会引入新的挑战
  - 反向传播中的梯度消失（Vanishing gradients）
  - 正向传播中越来越少的特征重复使用（Diminishing feature reuse）
  - 训练时间长

更深神经网络面临的挑战

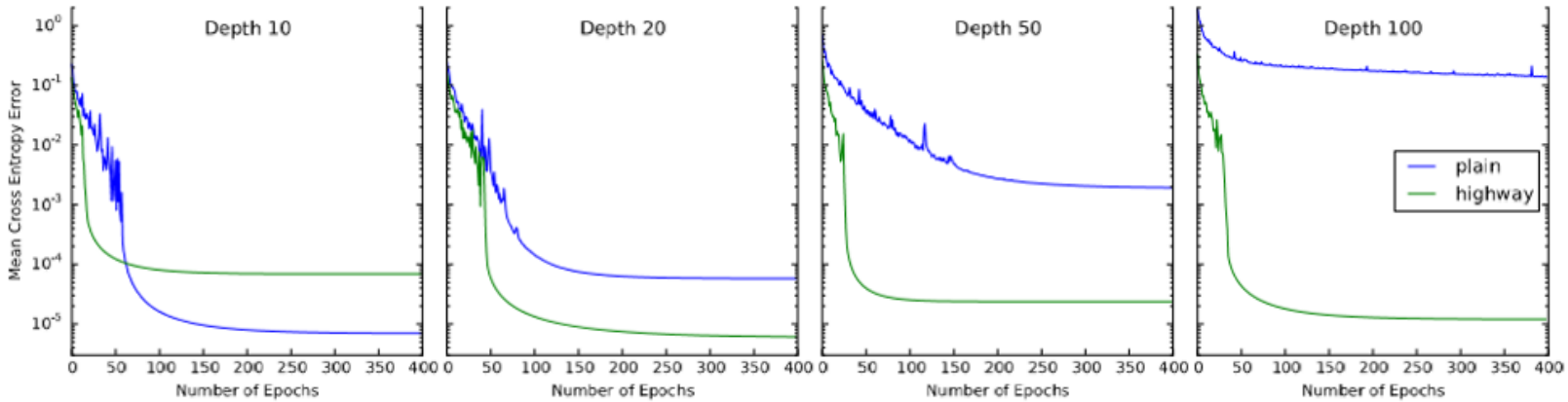
- 传统的神经网络

$$y = H(x, W_h)$$

- 高速神经网络（Highway Neural Network）

$$y = H(x, W_h)\ T(x, W_T) + x\ (1 - T(x, W_h))$$

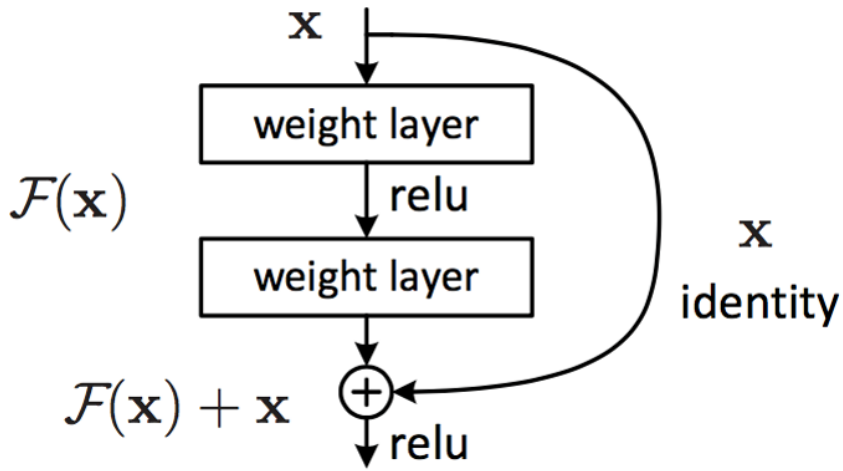
T 表示变换门（transform gate），$(1-T)$ 表示携带门（carry gate）



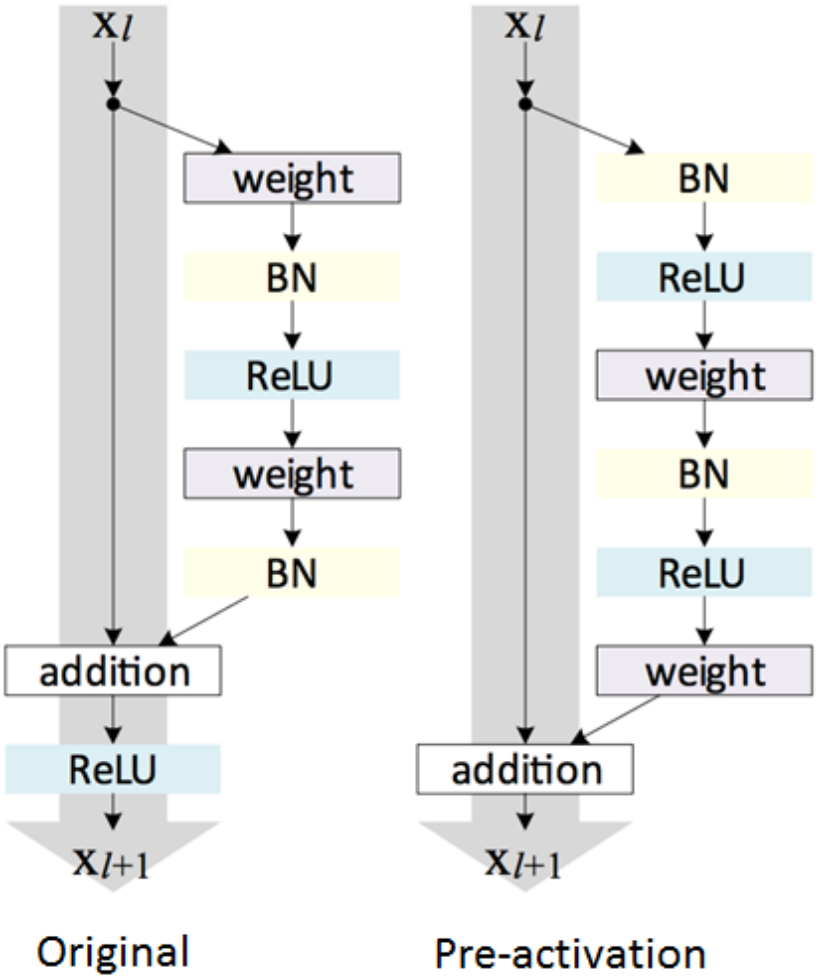
案例研究：ResNet

残差学习，解释：

- 求解器（solver）可能难以通过多个非线性层来逼近恒等映射（identity mapping）
- 恒等映射不太可能是最优的，但如果最优函数更接近于恒等映射而不是零映射，那么求解器参考恒等映射找到扰动（perturbations）应该比学习新的函数更容易。



案例研究：ResNet

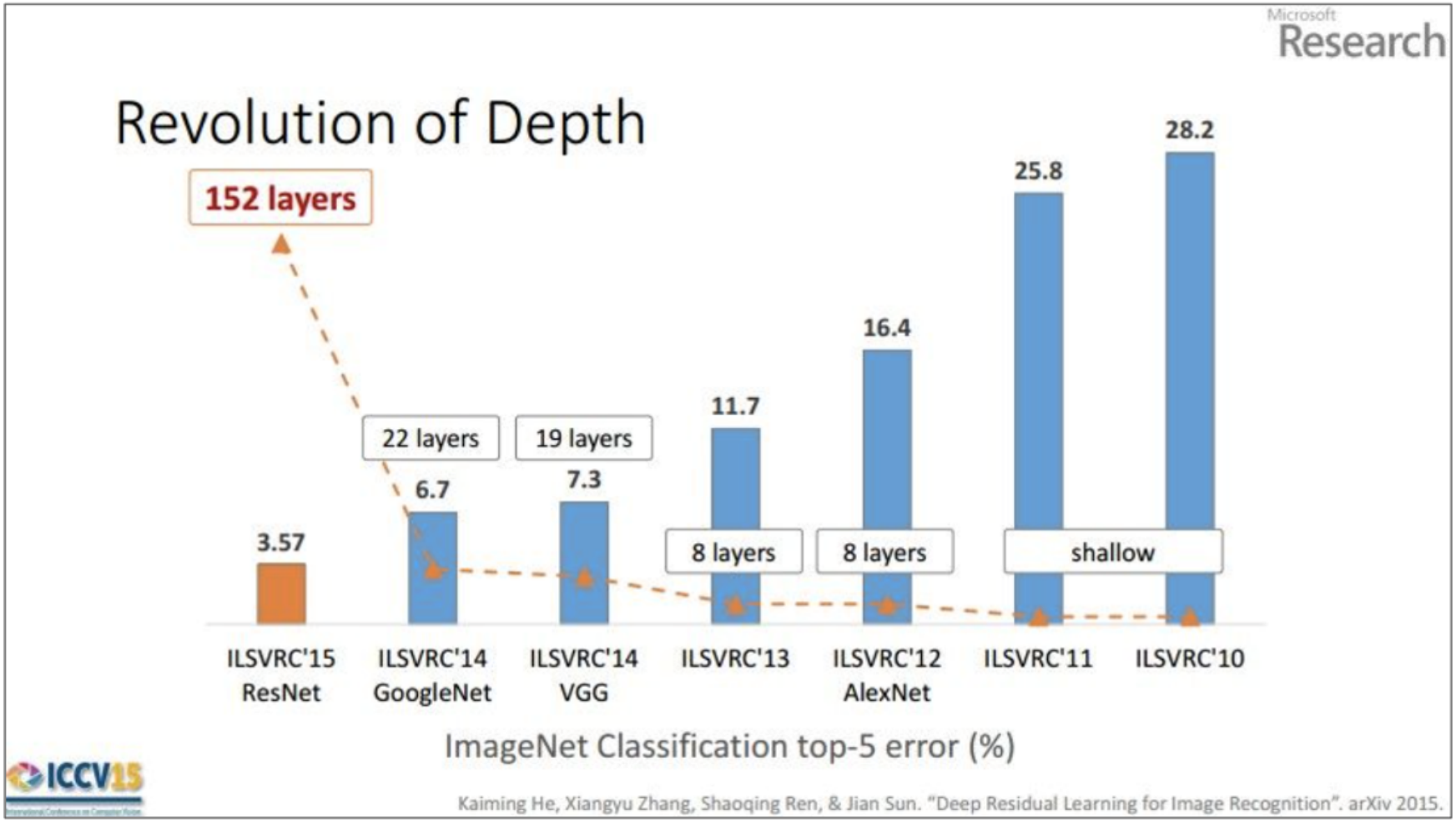


在下列工作中，残差单元被更新到“激活前”版本(Pre-activation version)。

为了建立一个“直接”路径，用于传播信息 — 不仅在一个残差单元内，而且在整个网络中。

能观察到额外的性能增益。

案例研究： ResNet


Microsoft Research
Revolution of Depth
152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10
ImageNet Classification top-5 error (%)
ICCV15
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

一些可视化

从目标检测任务的角度看，对于给定层中的一些单元，输入图像中的哪些部分（输入补丁，input patches）给出的激活值最高？

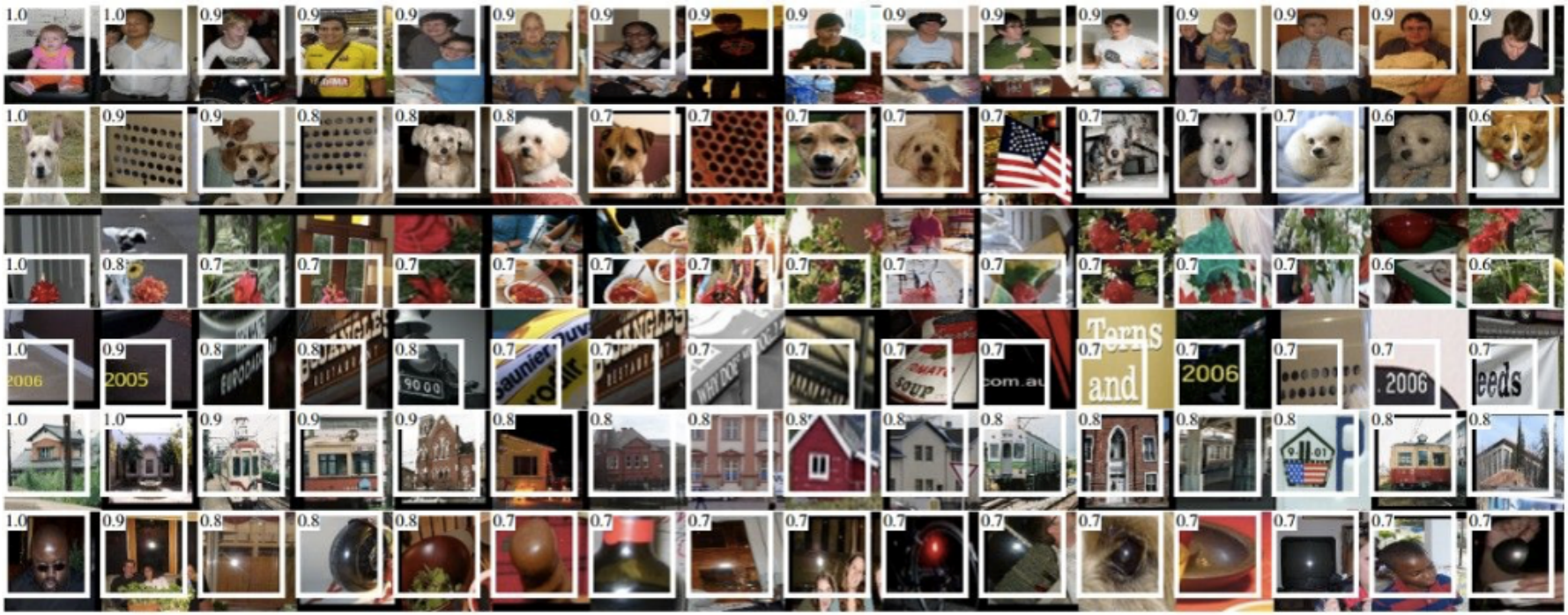
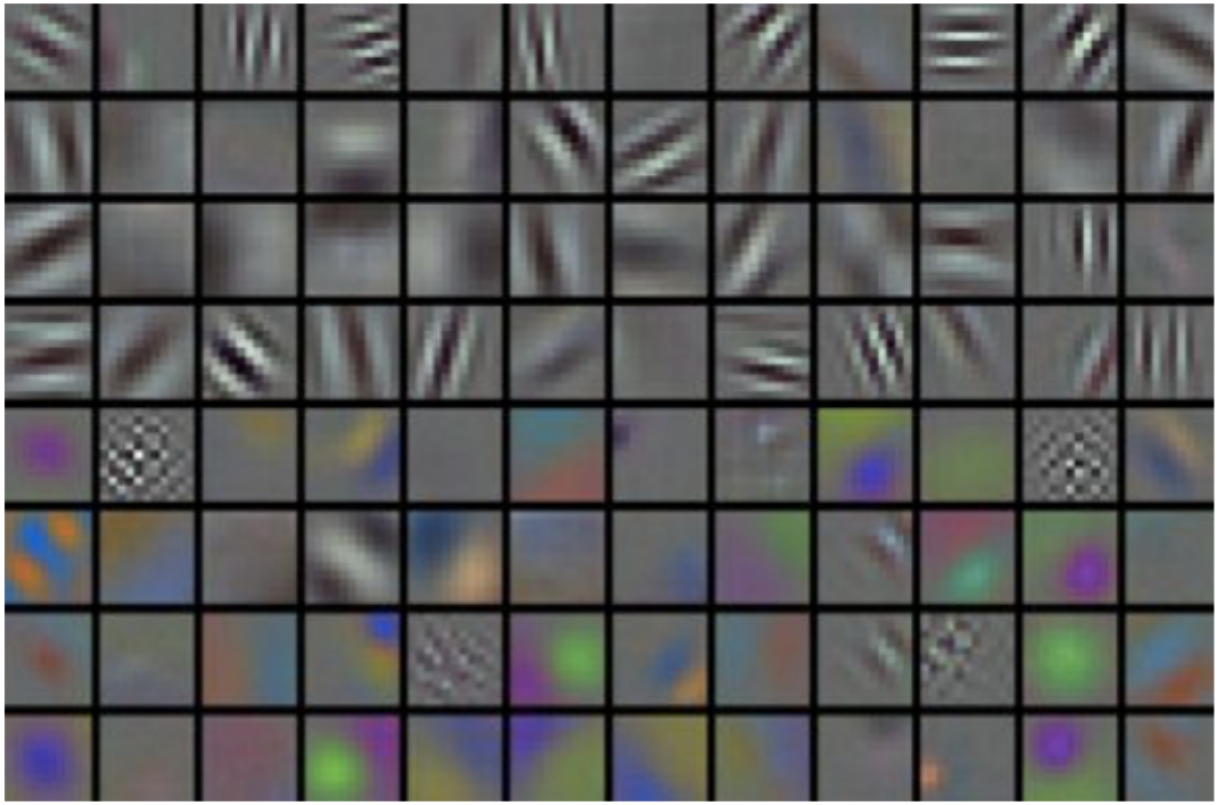


Figure 4: Top regions for six pool$_5$ units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).

Girshick et al, 2014

一些可视化

可视化卷积核，但不幸的是只有第一层的是可解释的

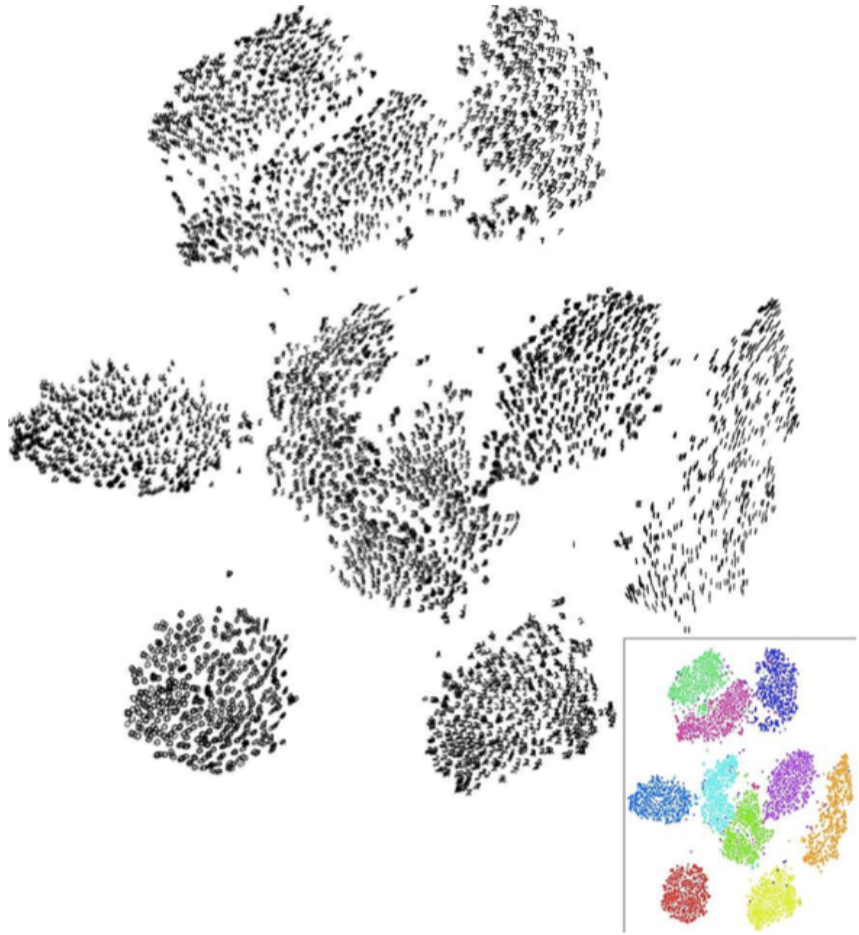


一些可视化

用t-SNE来可视化特征表示

t-SNE 是一种将高维数据点嵌入到二维平面中的方法，同时使高维空间中彼此接近的数据点在二维平面中仍然保持接近

右图：MNIST 数据集（0–9）中的 CNN 特征表示示例



如何使用： `https://scikit-learn.org/stable/modules/manifold.html#t-sne`

语音识别中的 CNN

Very Deep Convolutional Neural Networks for Noise Robust Speech Recognition [Yanmin Qian et al, TASLP 2016]

问题：
为什么 CNN 能在语音上生效？

当输入有以下两个性质时，CNN 能够很好发挥作用：

- 局部相关性（local correlations）
- 平移不变性（translational invariance）

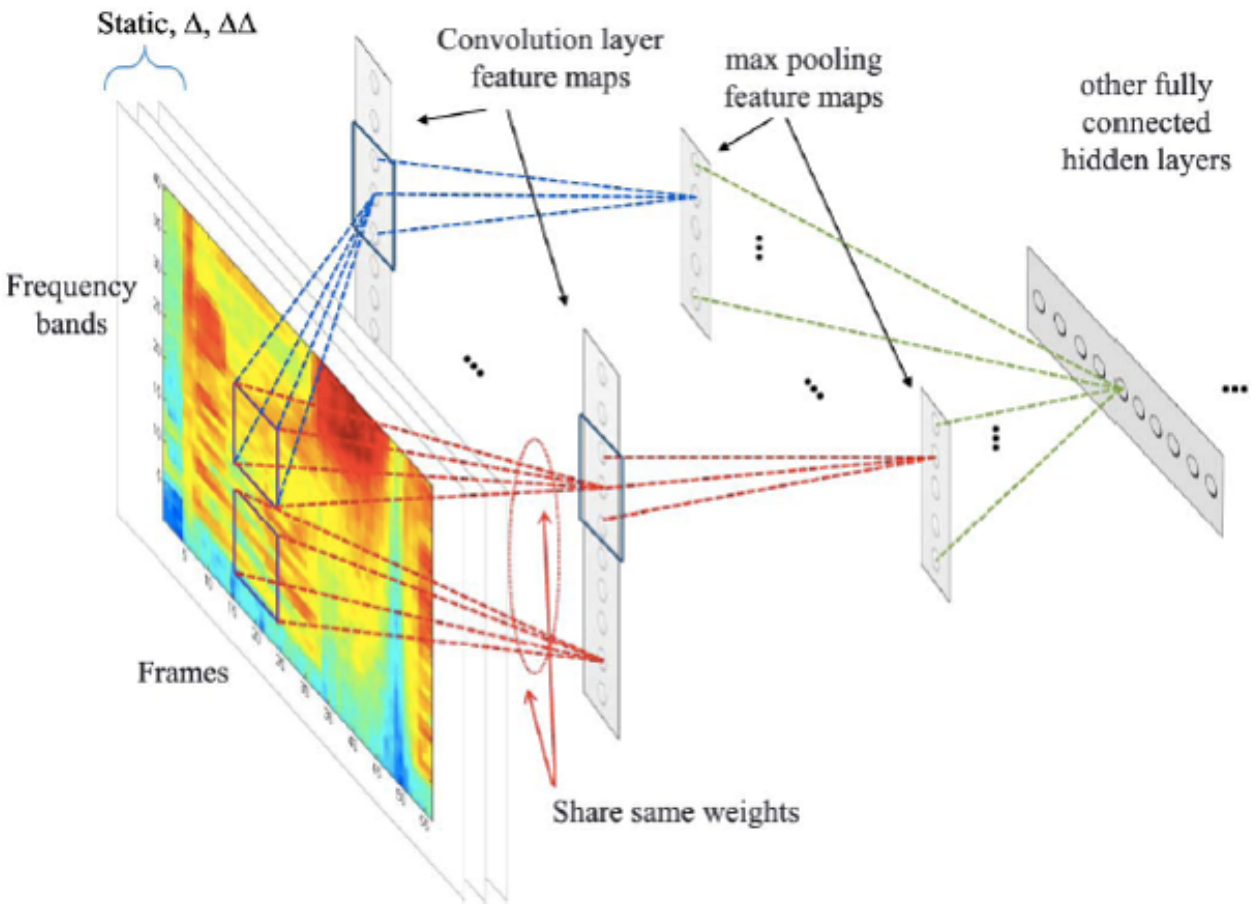
语音同时具有以上两种性质

- 语音特征的相邻维度是相关的
- 不同说话人可能在频带中引入小的偏移

ASR 中的常用结构——浅层 CNN

2 个卷积层 + 4 个全连接层

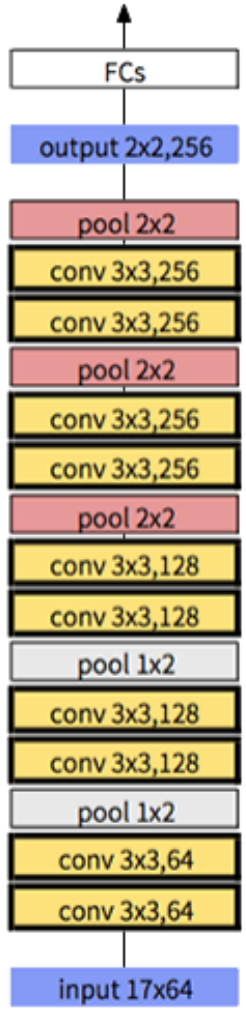
[Tara et al, ICASSP 2013]



ASR 中的高级 VDCNN

Advanced VDCNN for ASR

Very Deep Convolutional Neural Networks for Noise Robust Speech Recognition [Yanmin Qian et al, TASLP 2016]

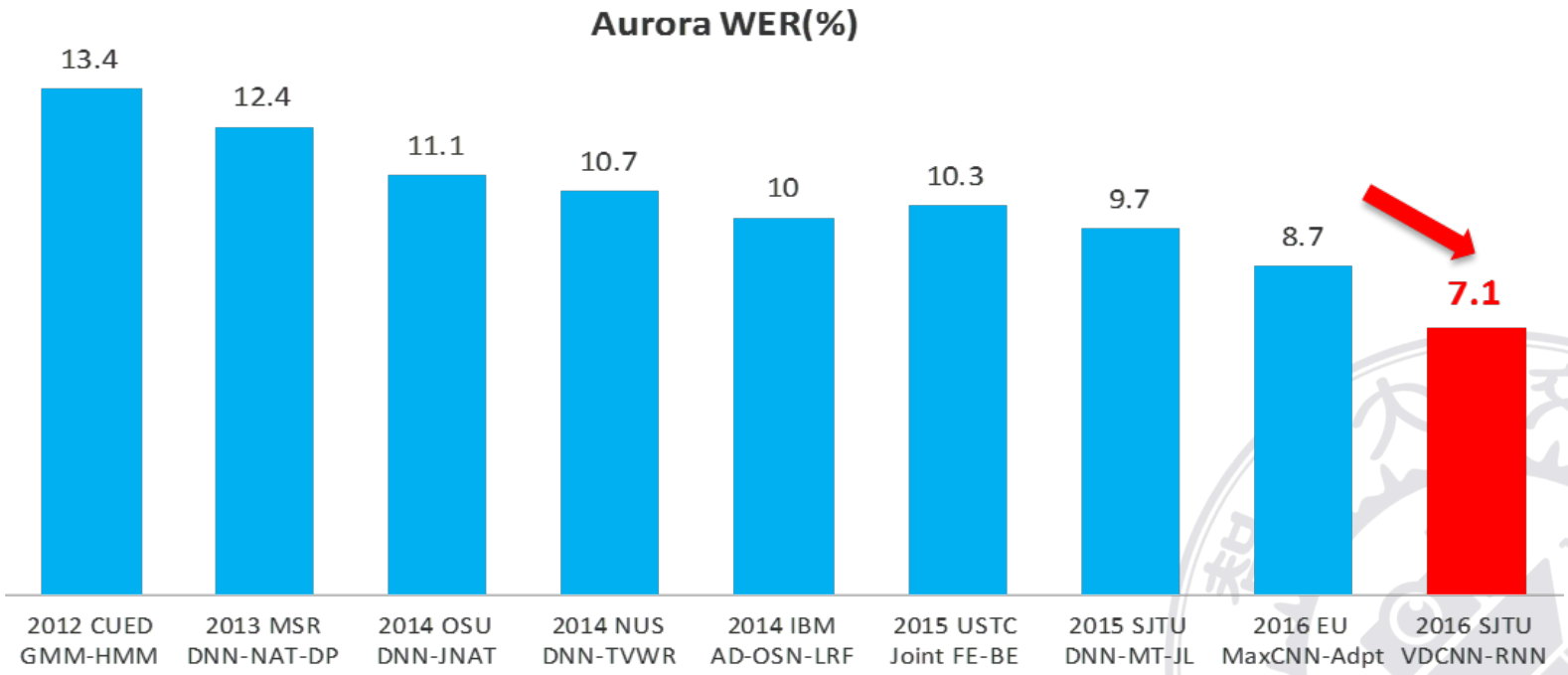


这篇工作使用了一个类似 VGG 的网络，主要的区别是，与图像分类的典型输入 224×224 不同，语音的典型输入类似于 11×39
且 CNN 的结构与输入有关

这篇工作采用了 17×64 的输入，以及 10 个卷积层的网络

让我们看一些有趣的观察：

Aurora4


Aurora WER(%)
13.4
12.4
11.1
10.7
10
10.3
9.7
8.7
7.1
2012 CUED GMM-HMM
2013 MSR DNN-NAT-DP
2014 OSU DNN-JNAT
2014 NUS DNN-TVWR
2014 IBM AD-OSN-LRF
2015 USTC Joint FE-BE
2015 SJTU DNN-MT-JL
2016 EU MaxCNN-Adpt
2016 SJTU VDCNN-RNN

语音识别中的 VDCNN

Very Deep Convolutional Neural Networks for Noise Robust Speech Recognition [Yanmin Qian et al, TASLP 2016]

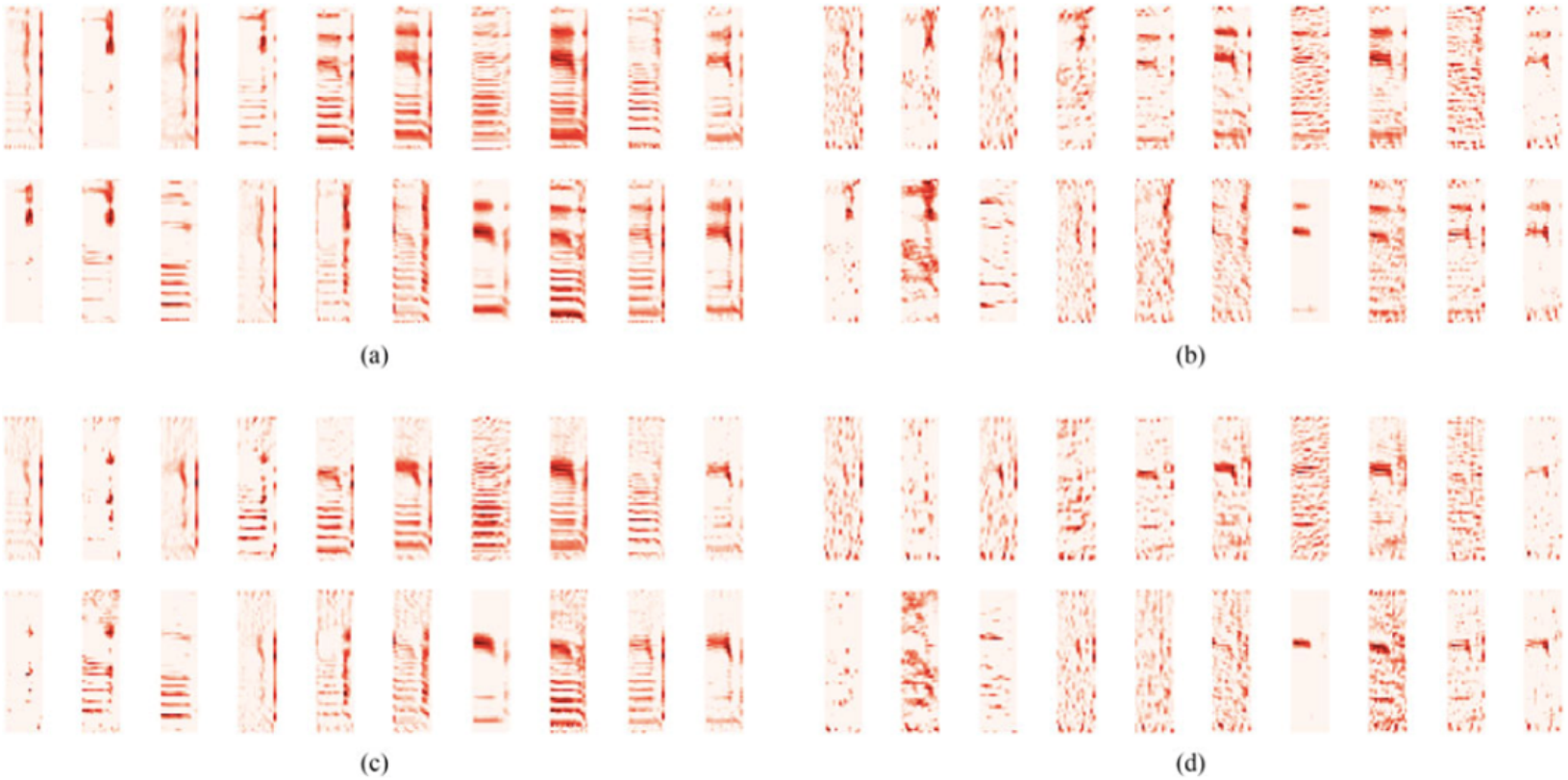


Fig. 7. Selected feature maps of the first convolutional layer using the same single frame from 4 different conditions in Aurora4.

语音识别中的 VDCNN

Very Deep Convolutional Neural Networks for Noise Robust Speech Recognition [Yanmin Qian et al, TASLP 2016]

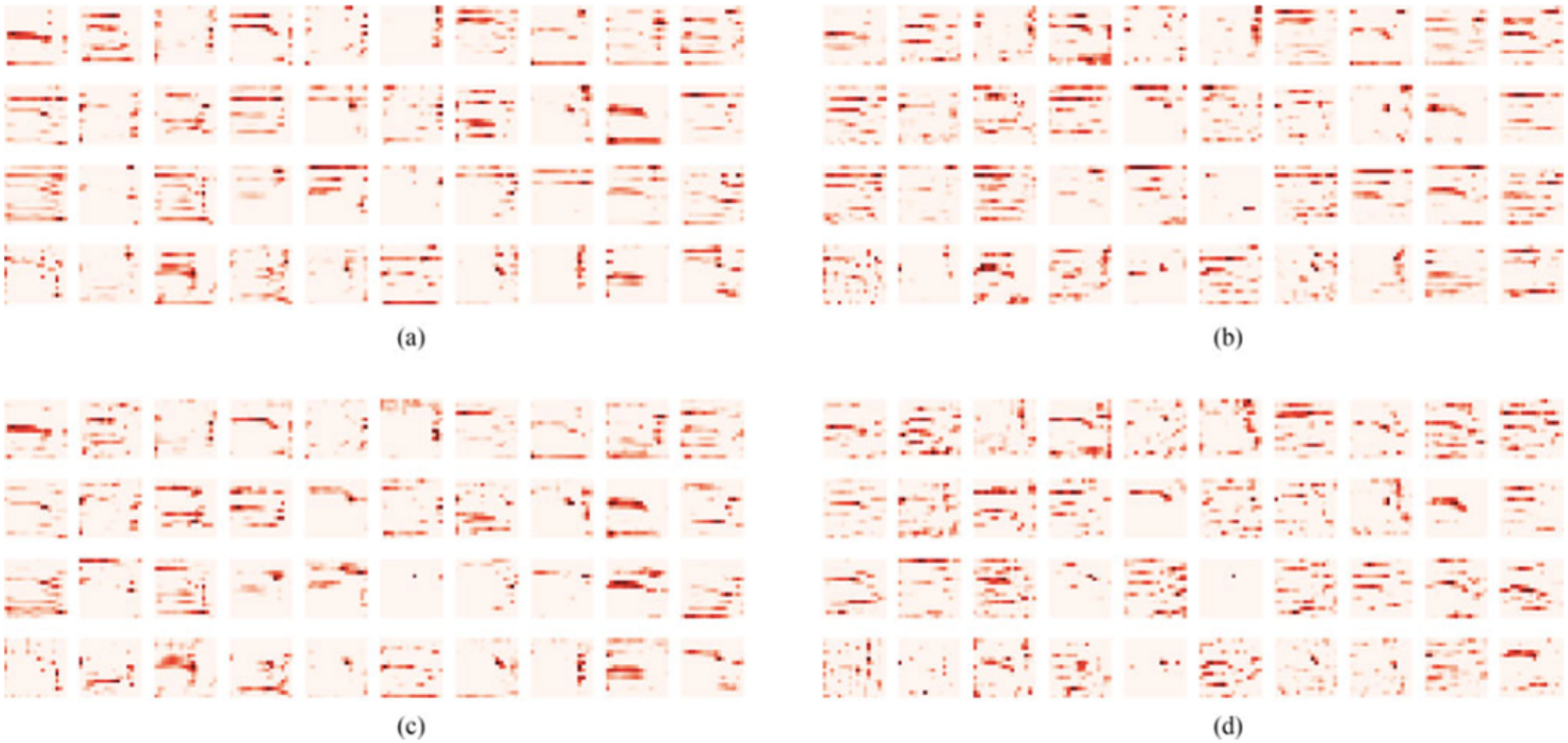
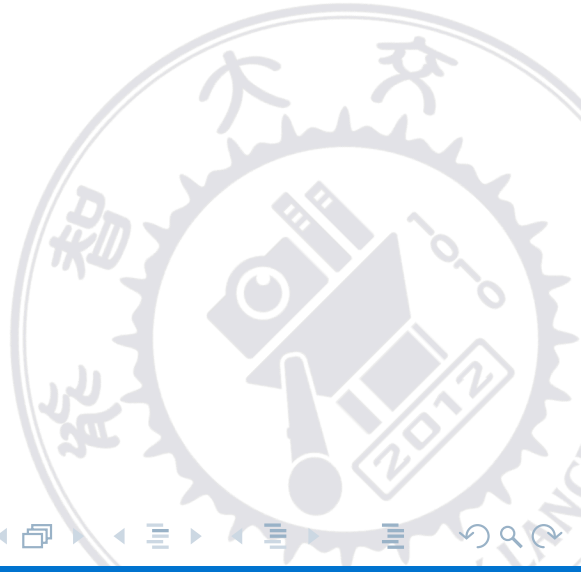


Fig. 8. Selected feature maps of the sixth convolutional layer using the same single frame from 4 different conditions in Aurora4.

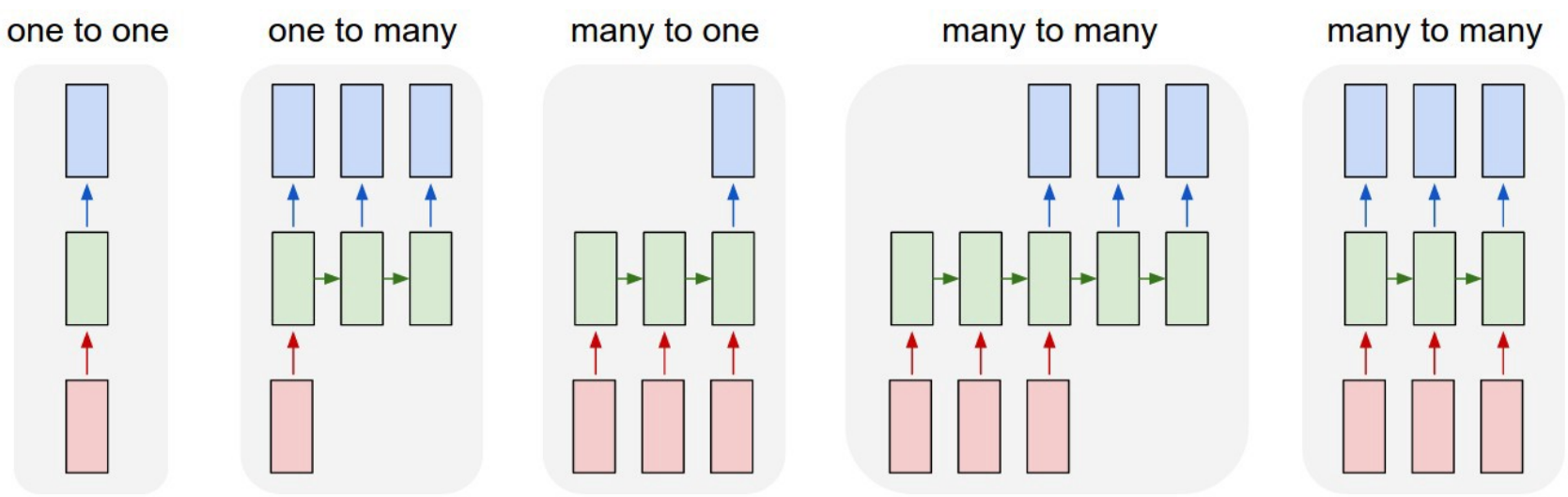
RNN — 序列建模

- 问题
  - DNN 和 CNN 对于序列到序列（Seq2Seq）问题的建模能力有限
  - 语音识别中输入和输出序列的长度通常十分不同，且不同样本的长度也各有差异，很难或不可能通过固定的输入/输出长度进行建模



RNN — 序列建模

- 循环神经网络（RNN）
  - 可以接收定长/变长序列作为输入
  - 输出可以是定长序列，也可以是变长序列



https://purnasaigudikandula.medium.com/recurrent-neural-networks-and-lstm-explained-7f51c7f6bbb9

RNN — 前向计算

- 循环神经网络（RNN）
  - 每个时刻 t 网络的输入由两个部分组成：
    - 输入序列在当前时刻的取值 $\mathbf{x}_{t-1}$
    - 上一时刻网络的输出 $\mathbf{o}_{t-1}$/隐层状态 $\mathbf{h}_{t-1}$
  - 每个时刻 t 网络的隐层状态 $\mathbf{h}_t$：

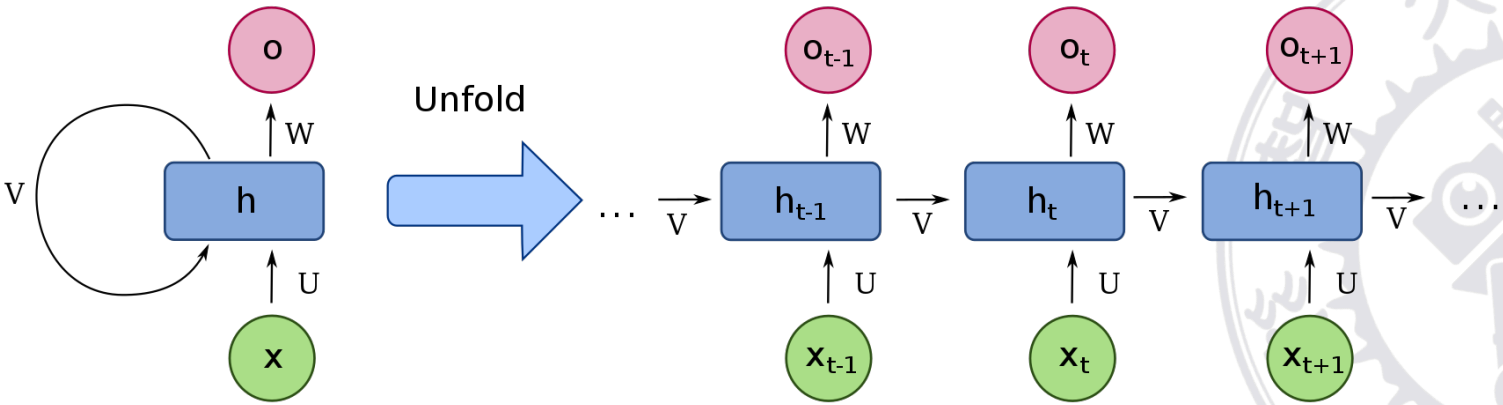
$$\mathbf{h}_t = \sigma_h(\mathbf{U}\mathbf{x}_t + \mathbf{V}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

  - 每个时刻 t 网络的输出 $\mathbf{o}_t$：

$$\mathbf{o}_t = \sigma_o(\mathbf{W}\mathbf{h}_t + \mathbf{b}_o)$$

  - 对于不同时刻，网络参数 $\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{b}_h, \mathbf{b}_o$ 均是共享的

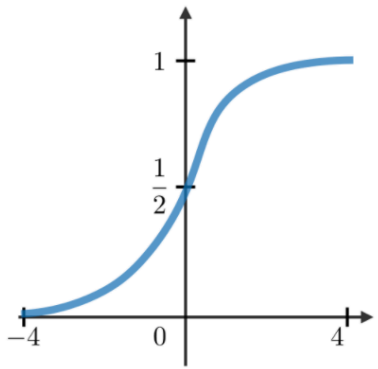
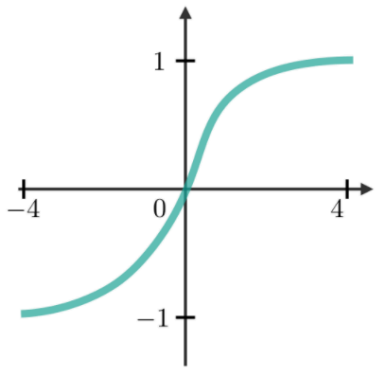
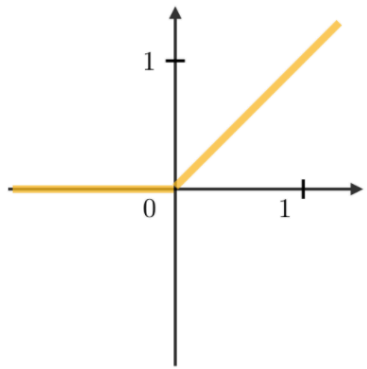
$\mathbf{W}, \mathbf{U}, \mathbf{V}$：权重矩阵参数
$\mathbf{b}_h, \mathbf{b}_o$：偏置向量参数
σ_h, σ_o：激活函数（如 tanh）



https://en.wikipedia.org/wiki/Recurrent_neural_network

RNN — 前向计算

- 循环神经网络（RNN）
  - 常用的激活函数

Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

RNN — 前向计算

- Loss 函数

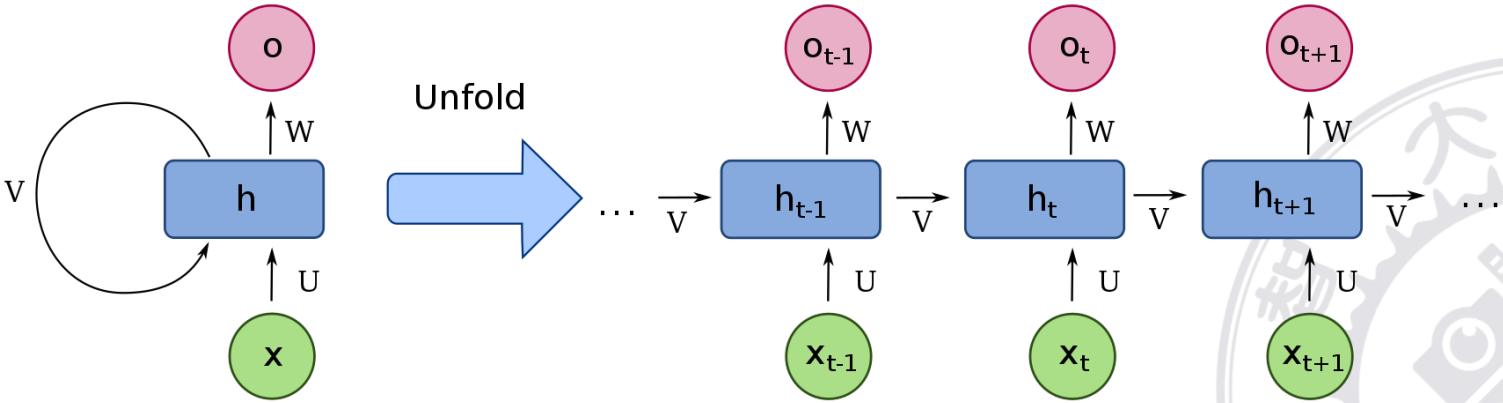
$$\mathcal{L}(\mathbf{r}, \mathbf{o}) = \sum_{t=1}^{T_r} \mathcal{L}_t(r_t, \mathbf{o}_t)$$

  - 其中 $\mathbf{o}$ 是模型输出序列，$\mathbf{r}$ 是标签序列，T_r 是标签序列的长度

- **沿时间的反向传播**（Backpropagation Through Time, BPTT）

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_t \frac{\partial \mathcal{L}_t}{\partial \mathbf{W}}$$

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{W}} = \frac{\partial \mathbf{o}_t}{\partial \mathbf{W}} \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}$$

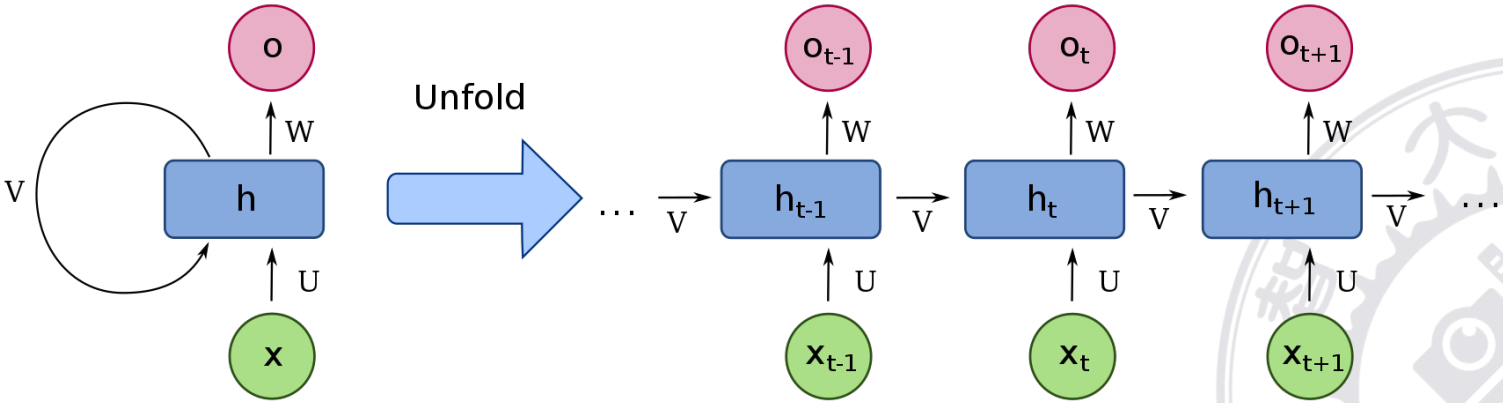


https://en.wikipedia.org/wiki/Recurrent_neural_network

- **沿时间的反向传播**（Backpropagation Through Time, BPTT）

$$\frac{\partial \mathcal{L}}{\partial \mathbf{U}} = \sum_t \frac{\partial \mathcal{L}_t}{\partial \mathbf{U}}$$

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{U}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{U}} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}$$

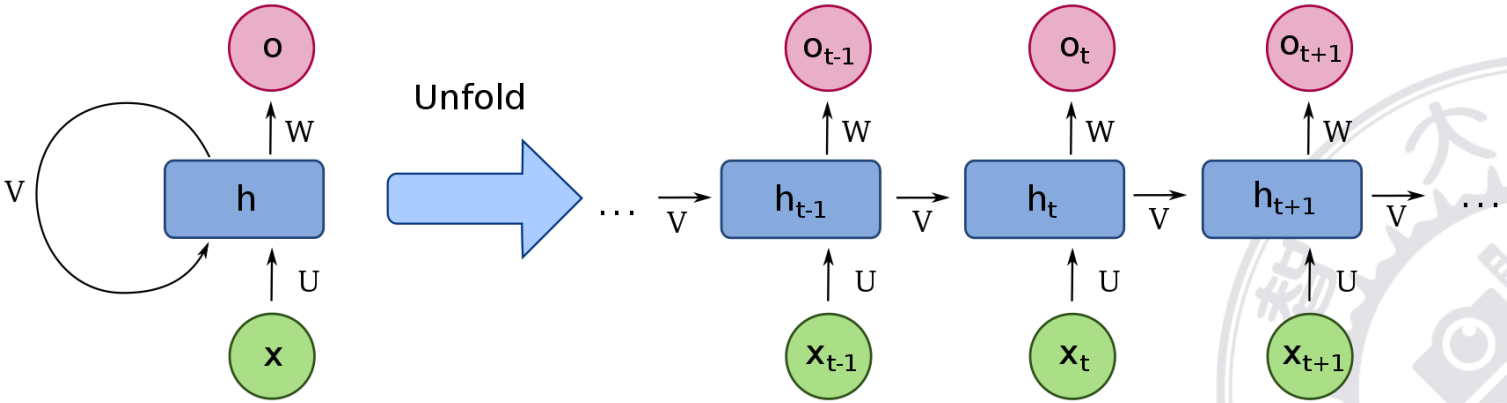


https://en.wikipedia.org/wiki/Recurrent_neural_network

- **沿时间的反向传播**（Backpropagation Through Time, BPTT）

$$\frac{\partial \mathcal{L}}{\partial \mathbf{V}} = \sum_t \frac{\partial \mathcal{L}_t}{\partial \mathbf{V}}$$

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{V}} = \frac{\partial \mathbf{h}_t}{\partial \mathbf{V}} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}$$



https://en.wikipedia.org/wiki/Recurrent_neural_network

RNN — 反向传播

```
Back_Propagation_Through_Time(x, r)    // x[t] is the input at time t. r[t] is the target label
    Unfold the network to contain k instances of hidden blocks (f) and one output block (g)
    do until stopping criteria is met:
        h := the zero-magnitude vector // initial hidden state
        for t from 0 to n - k do       // t is time. n is the length of the training sequence
            Set the network inputs to h, x[t], x[t+1], ..., x[t+k-1]
            p := forward-propagate the inputs over the whole unfolded network
            e := r[t+k] - p;           // error = target - prediction
            Back-propagate the error, e, back across the whole unfolded network
            Sum the weight changes in the k instances of f together.
            Update all the weights in f and g.
            h := f(h, x[t]);           // compute the context for the next time-step
```

https://en.wikipedia.org/wiki/Backpropagation_through_time

Homework — 推导 RNN 反向传播更新公式

- 输入序列为 $\mathbf{x}$，输出序列为 $\hat{\mathbf{r}}$，标签序列为 $\mathbf{r}$（长度为 T_r），总类别数为 C，激活函数为 $\sigma(z) = \frac{1}{1+e^{-z}}$
- 网络结构如下：
  - (1) 输入层：

$$\mathbf{a}^{(\text{in})} = \mathbf{W}^{(\text{in})}\mathbf{x} + \mathbf{b}^{(\text{in})}$$

  - (2) 隐层（RNN）：

$$\begin{aligned} \mathbf{h}_t &= \sigma(\mathbf{U}\mathbf{a}_t^{(\text{in})} + \mathbf{V}\mathbf{h}_{t-1} + \mathbf{b}_h) \\ \mathbf{o}_t &= \sigma(\mathbf{W}\mathbf{h}_t + \mathbf{b}_o) \end{aligned}$$

  - (3) 输出层：

$$\begin{aligned} \mathbf{h}^{(\text{out})} &= \mathbf{W}^{(\text{out})}\mathbf{o}_t + \mathbf{b}^{(\text{out})} \\ \hat{\mathbf{r}} &= \text{Softmax}(\mathbf{h}^{(\text{out})}) \end{aligned}$$

- Loss 函数：$\mathcal{L} = \sum_{t=1}^{T_r} \mathcal{L}_t = \sum_{t=1}^{T_r} \text{Loss}(r_t, \hat{\mathbf{r}}_t)$

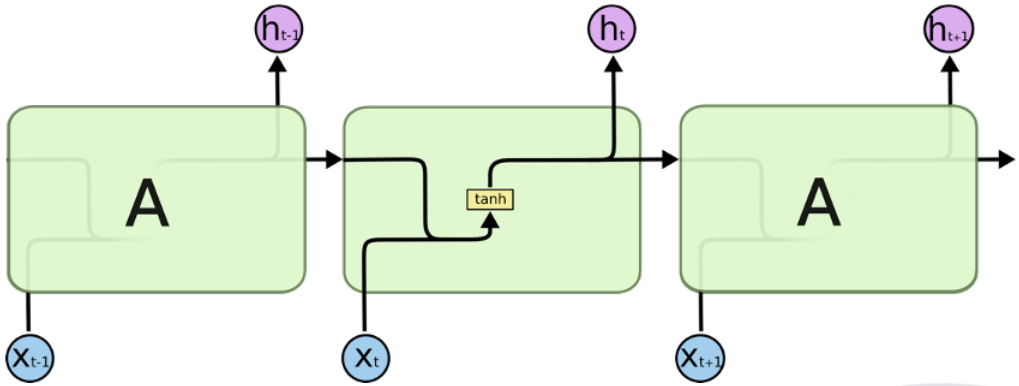
请推导上述模型的反向传播更新公式（Loss 函数假定为交叉熵 cross-entropy）。

- 循环神经网络（RNN）的问题
  - 梯度消失（Vanishing Gradient）
    - 使用 Sigmoid 作为激活函数时，随着序列长度的增长，离当前时刻较远的历史输入的梯度会呈指数级下降
    - $\Rightarrow$ 无法对长时依赖进行有效建模
  - 梯度爆炸（Exploding Gradient）
    - 由于 RNN 中后一时刻的隐层状态梯度依赖于前一时刻的隐层状态，其对应参数的梯度会随着序列长度的增长而不断累积，当每一时刻的梯度均大于 1 时，该参数的梯度将会呈指数级上升
    - $\Rightarrow$ 可通过梯度裁剪（gradient clipping）来缓解

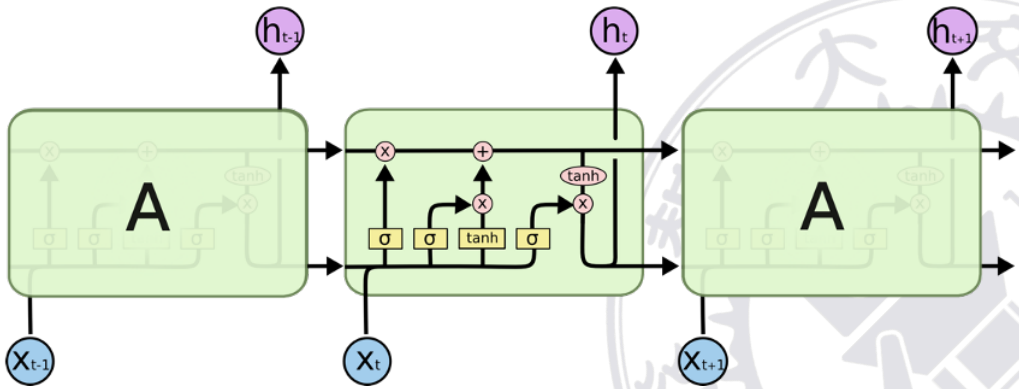
Q: 梯度消失问题在循环神经网络（RNN）中更严重，还是在前馈神经网络（FNN）中更严重？为什么？

- 长短期记忆（LSTM）
  - 缓解 RNN 中存在的梯度消失问题，从而能够建模序列的长时依赖

原始 RNN



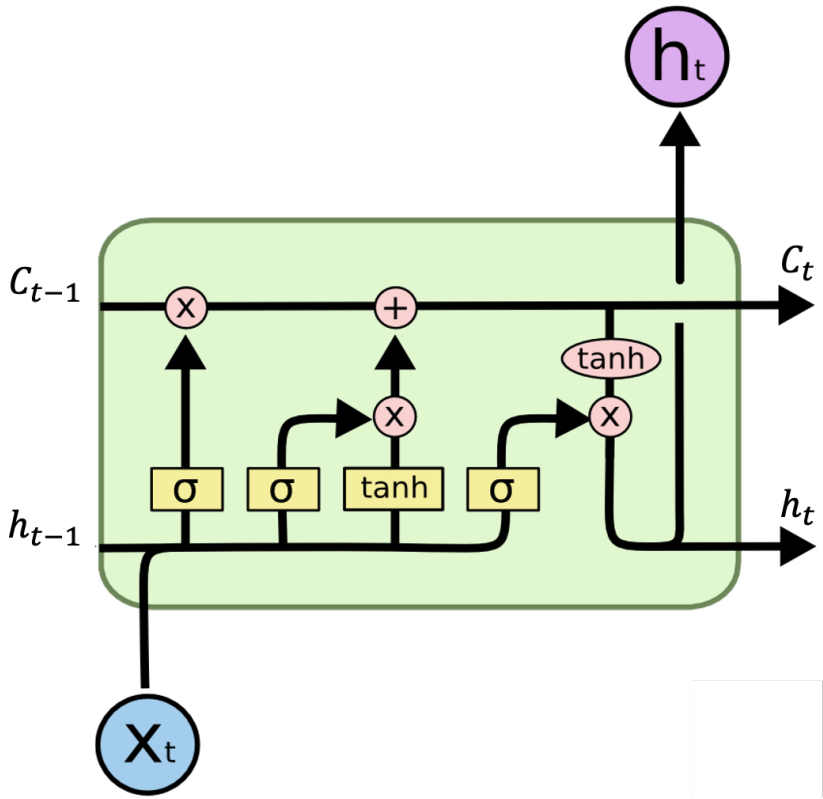
LSTM



长短期记忆（LSTM）

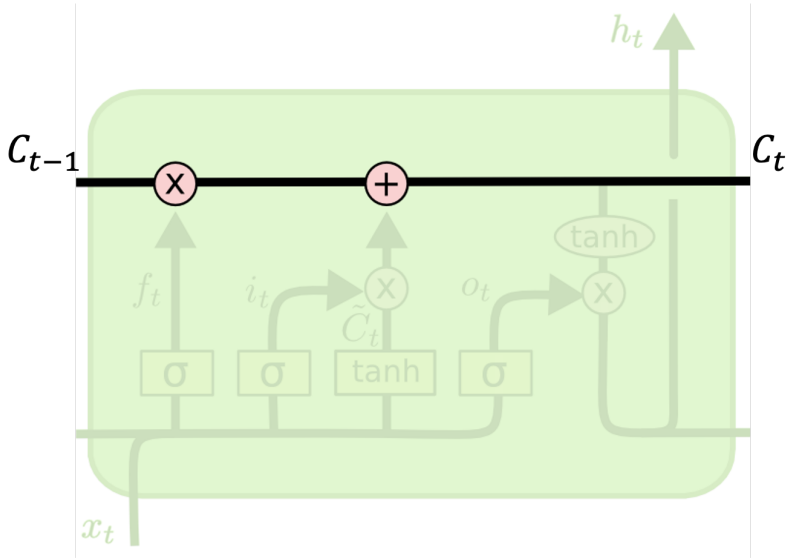
- LSTM cell 内部结构
  每个 LSTM cell 主要由遗忘门、输入门、输出门三个部分组成

  输入由前一个 cell 的“隐状态” $\mathbf{h}_{t-1}$、当前 cell 的输入 $\mathbf{x}_t$ 和前一个“cell 状态” $\mathbf{C}_{t-1}$ 组成



长短期记忆（LSTM）

- LSTM cell 内部结构
  - “cell 状态”在不同 LSTM cell 之间由一条直接路径连通，包含了历史信息

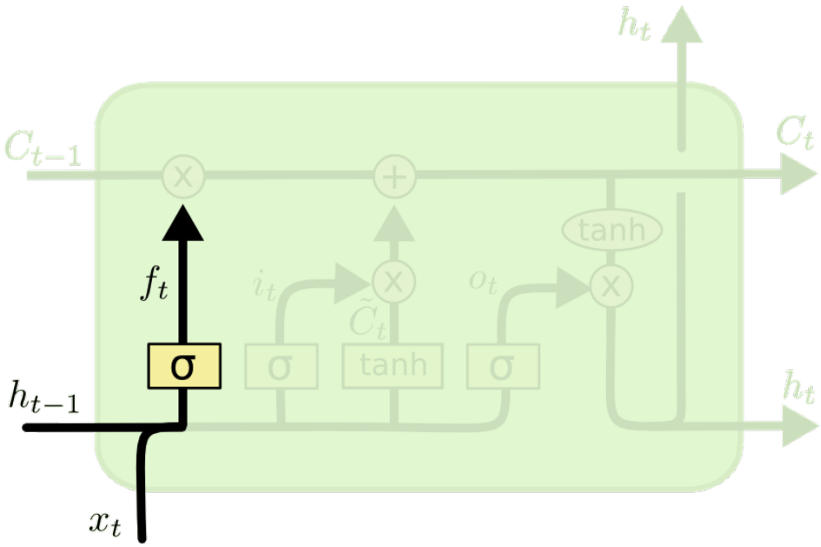


长短期记忆（LSTM）

- LSTM cell 内部结构
  - 遗忘门决定丢弃多少历史信息 C_{t-1}

$$\boldsymbol{f}_t = \sigma(\mathbf{W}_{xf}\mathbf{x}_t + \mathbf{W}_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)$$

其中 $\boldsymbol{f}_t$ 为遗忘因子，$\sigma(\cdot)$ 为 sigmoid 函数



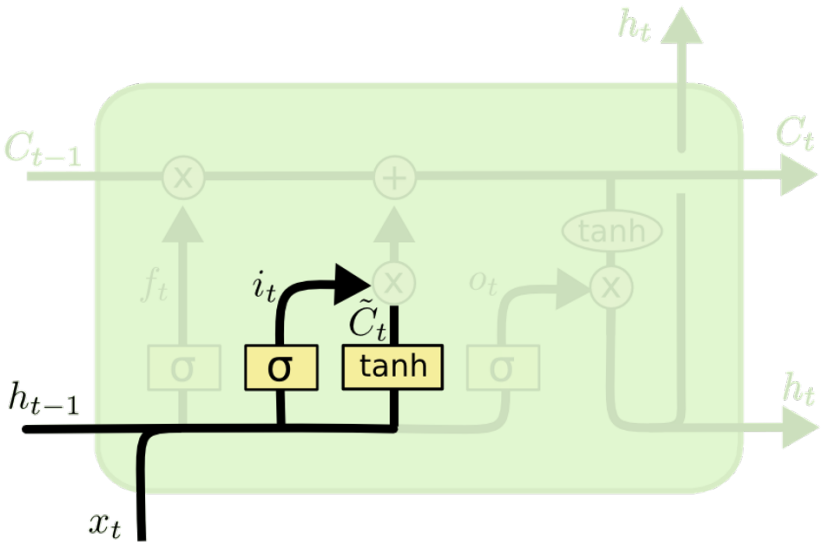
长短期记忆（LSTM）

- LSTM cell 内部结构
  - 输入门决定从前一个 cell 的“隐状态” $\mathbf{h}_{t-1}$ 和当前时刻输入 $\mathbf{x}_t$ 中保留多少信息到当前 cell 的“隐状态” $\mathbf{h}_t$ 中

$$\boldsymbol{i}_t = \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i)$$

$$\tilde{\boldsymbol{C}}_t = \tanh(\mathbf{W}_{xg}\mathbf{x}_t + \mathbf{W}_{hg}\mathbf{h}_{t-1} + \mathbf{b}_g)$$

其中 $\sigma(\cdot)$ 为 sigmoid 函数，$\tilde{C}_t$ 为当前 cell 的短期记忆



长短期记忆（LSTM）

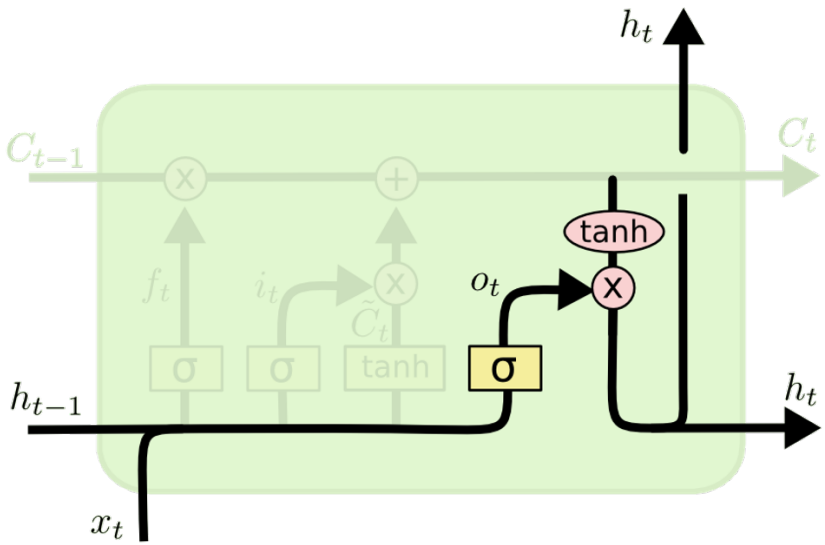
- LSTM cell 内部结构
  - 输出门决定将当前 cell 的多少信息添加到“cell 状态”中

$$\boldsymbol{o}_t = \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{C}_t)$$

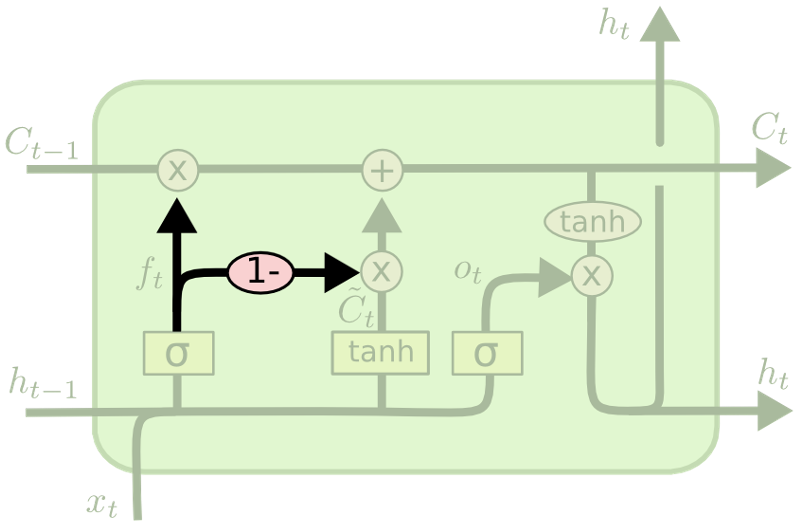
$$\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{C}}_t$$

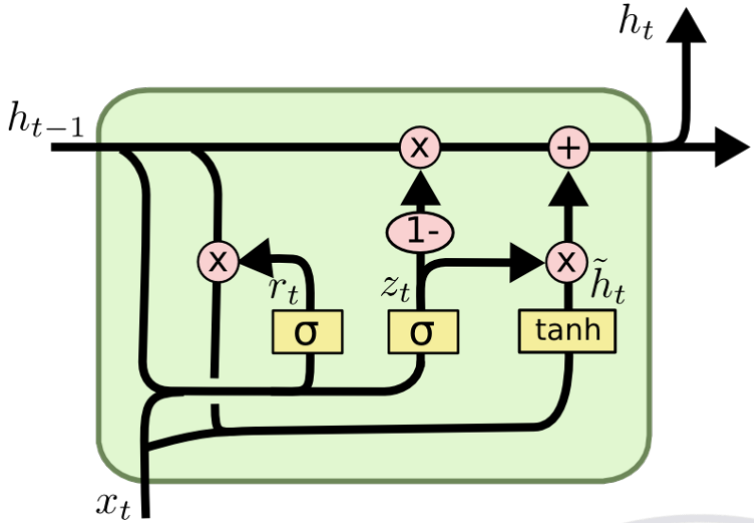
其中 $\sigma(\cdot)$ 为 sigmoid 函数，$\boldsymbol{C}_t$ 为当前“cell 状态”（“长期记忆”），$\odot$ 表示逐元素相乘



长短期记忆（LSTM）

- 其他变种





$$\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}_{t-1} + (1 - \boldsymbol{f}_t) \odot \tilde{\boldsymbol{C}}_t$$

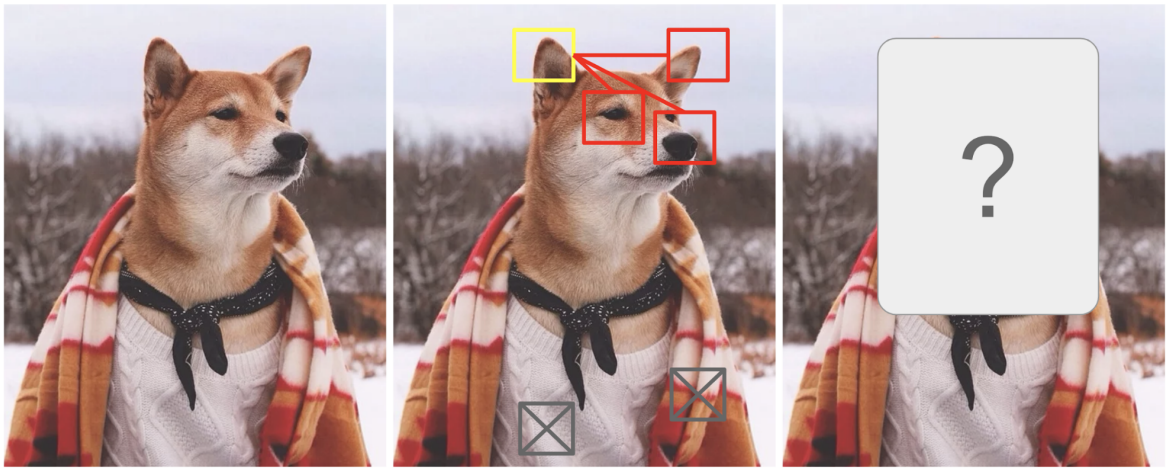
Gated Recurrent Unit (GRU)

双向 LSTM（BLSTM）：在从左向右、从右向左两个方向对输入序列进行建模

注意力机制

注意力（Attention）机制

- 对于人类视觉注意力机制的一种模仿
- 人类的视觉注意力能让我们以“较高的分辨率”主要关注某一特定区域的图像细节，而用“较低的分辨率”感知其周围的图像
- 下图中，当关注柴犬的左耳（黄色框）时，我们会期望在红色框中看到它的眼睛、鼻子、右耳等紧密相关的元素；而下半身的毛衣和毯子则不能提供同样有效的信息

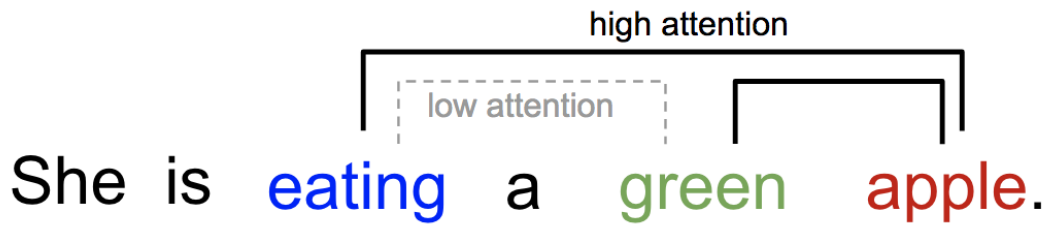


https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

注意力机制

注意力（Attention）机制

- 类似地，注意力机制也可以用于解释文本序列（如一句话）中不同单词之间的关联。



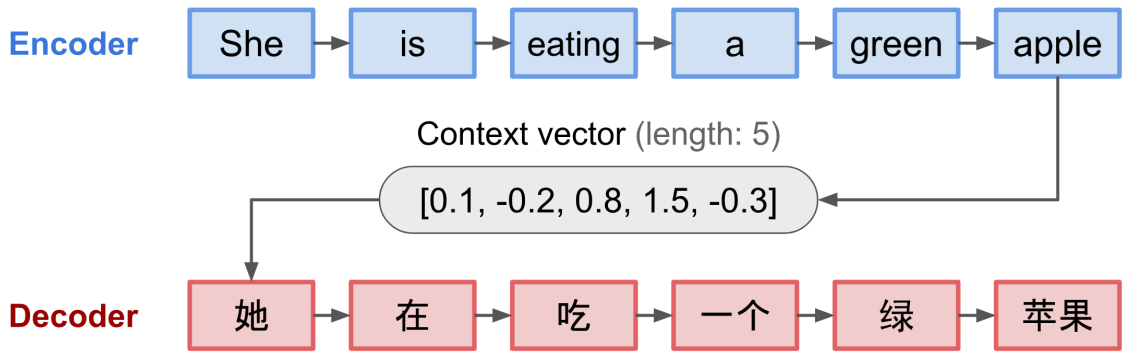
https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

- 当看到“eating”时，我们会期望其后面紧跟着一个食物的单词（上图中的“apple”），而表示颜色的单词（“green”）则主要是描述食物本身，与“eating”可能没有直接联系。

注意力机制

在注意力机制被提出之前，最常见的序列到序列（Seq2Seq）模型：编码器-解码器（Encoder-Decoder）

- 编码器：通常为 LSTM 或 BLSTM，将输入序列的信息压缩到一个定长的上下文向量（context vector）中，并期望该表示能对整个输入序列的信息具有很好的总结
- 解码器：通常为 LSTM，使用编码器输出的上下文向量作为初始输入，通过自回归（autoregressive）的方式逐步生成输出序列，直至结束（输出 <EOS>）

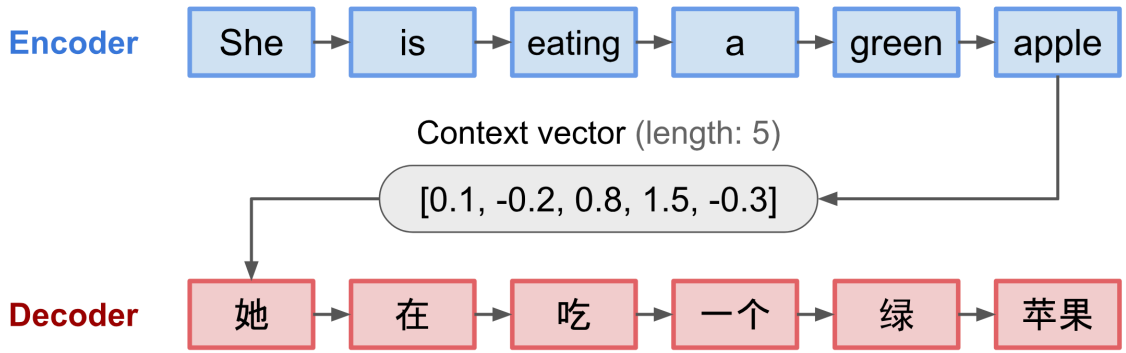


https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

注意力机制

在注意力机制被提出之前，最常见的序列到序列（Seq2Seq）模型：
编码器-解码器（Encoder-Decoder）

- 存在的问题/缺点：
  - 只使用了编码器最后一个时刻的隐层状态作为上下文向量，它依然很难记住长输入序列的所有信息
  - 先输入的序列信息会被后输入的信息稀释
  - 为解决此问题，注意力机制最初在神经机器翻译领域被提出[1]

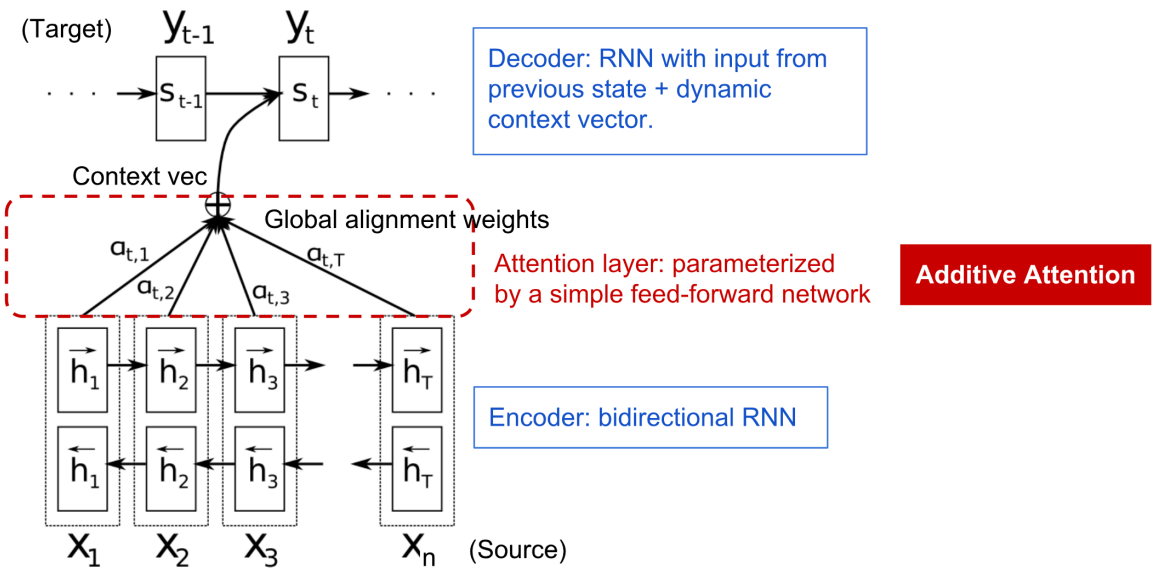


https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." in Proc. ICLR, 2015.

注意力（Attention）机制

- 主要思想：
  - 将输入序列中的每个单词都编码为一个向量（而不是只使用最后时刻的隐层状态），得到编码向量的序列
  - 在每一步解码时，将编码向量序列线性组合为一个上下文向量，其中每个编码向量的权重称为“注意力权重”



https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

注意力机制

注意力（Attention）机制

- 注意力机制的计算过程

$$Attention(Query, Key, Value) = \sum_{i=1}^{N} Similarity(Query, Key_i) \times Value_i$$

其中 N 为输入序列的长度

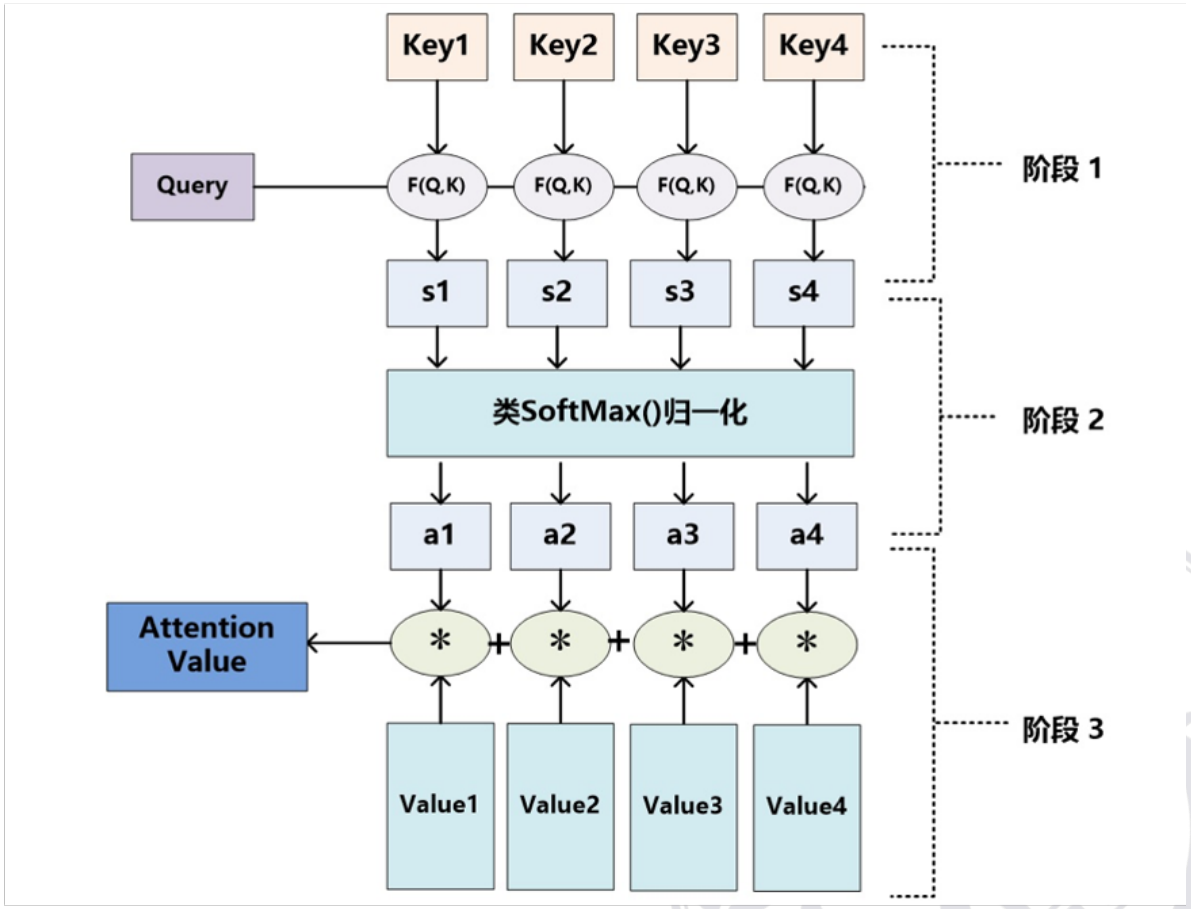
- Similarity 函数可以为 cos 距离、向量点积等，如

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \quad \in \mathbb{R}^{N \times d_v}$$

名称	描述
$\mathbf{X} \in \mathbb{R}^{N \times d}$ $\mathbf{C} \in \mathbb{R}^{M \times d}$	输入序列（长度为 N） 上下文序列（长度为 M）
$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q \in \mathbb{R}^{N \times d_k}$ $\mathbf{K} = \mathbf{C}\mathbf{W}_K \in \mathbb{R}^{M \times d_k}$ $\mathbf{V} = \mathbf{C}\mathbf{W}_V \in \mathbb{R}^{M \times d_v}$	Query Key Value

注意力（Attention）机制

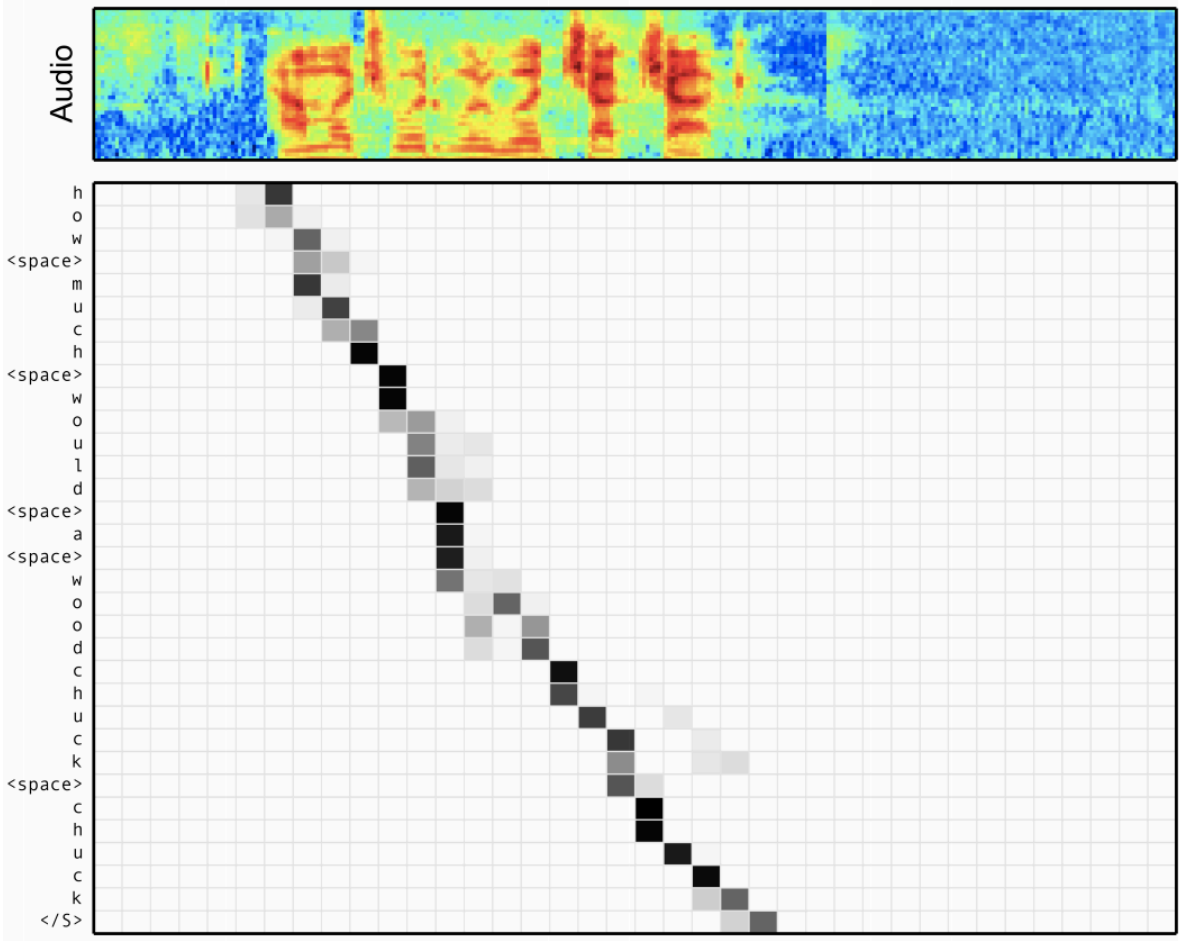
- 注意力机制的具体计算过程
  - (1) 计算 $Query$ 与每个 Key_i 的相似度 s_i；
  - (2) 对所有 s_i 进行归一化（取 Softmax），使它们的和为 1；
  - (3) 将归一化后的 s_i 与 $Value_i$ 对应相乘并求和，得到 $Attention$ 结果



注意力机制

注意力（Attention）机制

- 注意力机制也可以用于建模语音序列与文本序列之间的关系

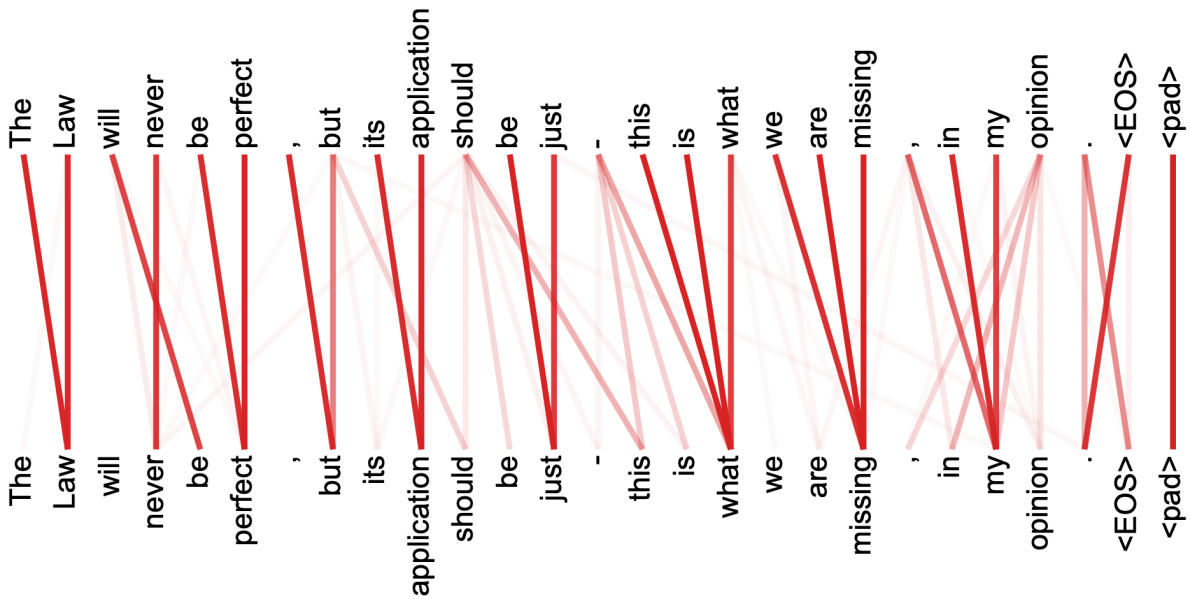


W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, Attend and Spell: A Neural Network for Large Vocabulary Conversational Speech Recognition," in Proc. IEEE ICASSP, 2016, pp. 4960–4964.

自注意力

自注意力（Self-Attention/Intra-Attention）

- 计算输入序列内部不同位置元素之间的关系
- $Query, Key, Value$ 三者均由相同的输入序列得到
- 如可以捕捉同一个句子内，不同单词之间的联系（句法特征、语义特征等）

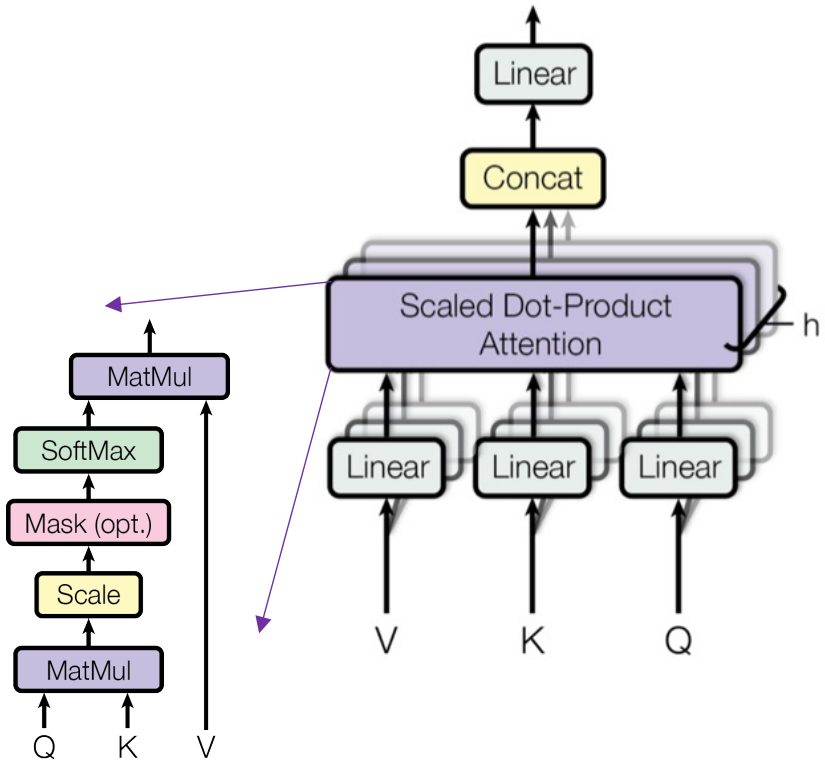


A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems, 2017.

多头注意力

多头注意力（Multi-Head Attention）

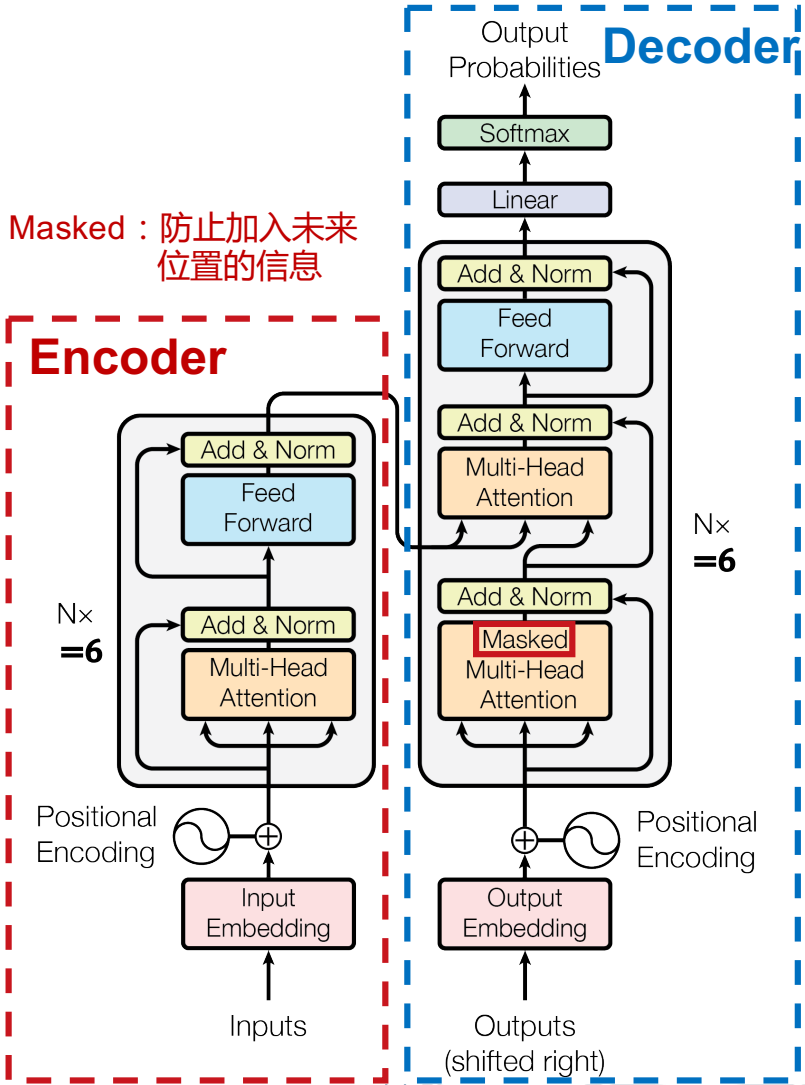
- 将 $Query, Key, Value$ 用 h 种不同投影方法，并行地计算 h 个版本的 $Attention$ 值
- 然后将这些 $Attention$ 值拼接在一起，并再次投影
- 希望能从不同的维度和表示子空间里学习到相关的信息



A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems, 2017.

Transformer: Attention is all you need

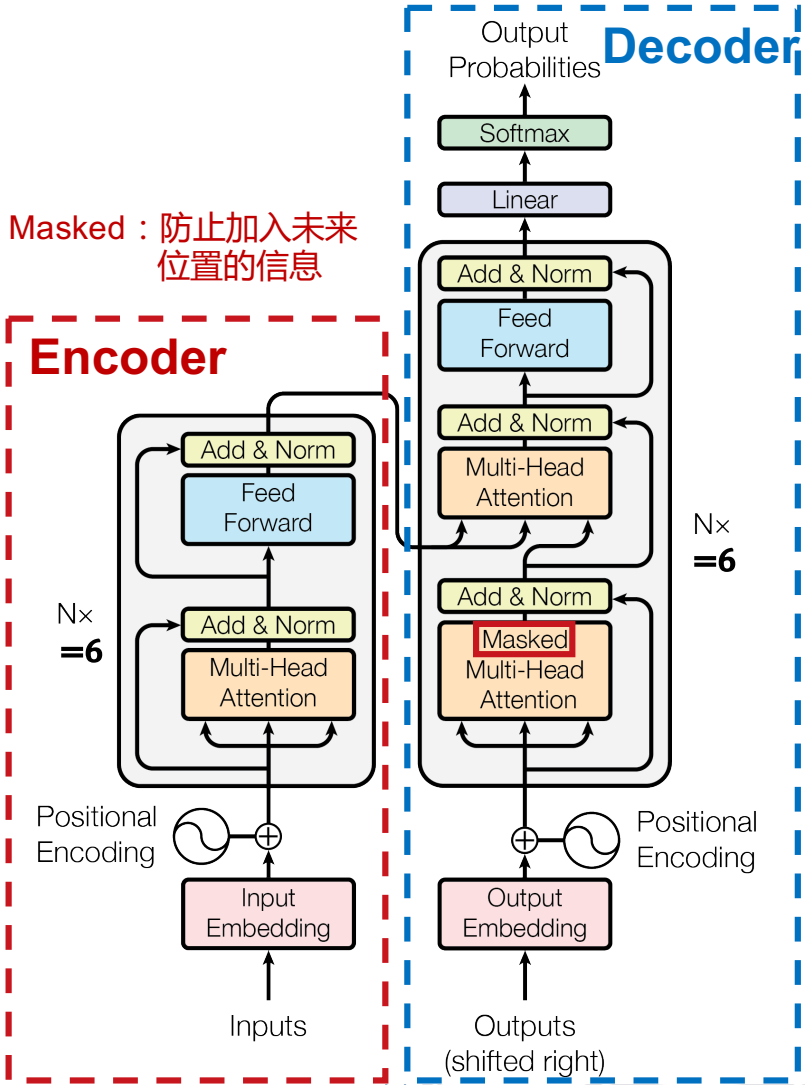
- 模型架构如右图所示
- 编码器（左半部分）
  - 输入的 embedding 中加入了位置编码，从而能够学到相对的位置信息
  - 由 N 个网络层堆叠组成
  - 每个网络层包含 2 个子网络层（均带有残差连接和层归一化）：
    1. 多头自注意力机制
    2. 全连接层
- 解码器（右半部分）



A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems, 2017.

Transformer: Attention is all you need

- 模型架构如右图所示
- 编码器（左半部分）
- 解码器（右半部分）
  - 由 N' 个网络层堆叠组成
  - 每个网络层有 3 个子网络层（均带有残差连接和层归一化）：
    1. （在输入上的）带 mask 的多头自注意力机制
    2. 全连接层
    3. （在编码器输出上的）多头注意力机制



A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems, 2017.

位置编码（Positional Encoding）

- 计算 Attention 时，由于没有采用 RNN 或卷积结构，缺少了序列的时序信息
- 为此，Transformer 引入了基于正弦/余弦函数的位置编码：

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

其中 pos 表示序列中的位置索引，i 表示编码维度，d_{model} 是编码器输入/输出维度

- 正弦/余弦函数能够较好地表示不同位置之间的相对关系
  - 对于任意固定的位置偏移 k，PE_{pos+k} 都可以表示为 PE_{pos} 的线性函数

Transformer

Transformer: Attention is all you need

- Transformer 是第一个完全基于注意力机制的序列转录模型，用多头自注意力机制取代了编码器-解码器架构中常见的 RNN 层
- Transformer 解决了 RNN 中序列建模不能并行计算的问题，训练速度比 RNN 快很多
- 在端到端语音识别（后面课程中介绍）中，基于 Transformer 结构的模型在大量基准数据集中超过了基于 RNN 的模型性能[2]

[2] S. Karita, N. Chen, T. Hayashi, T. Hori, H. Inaguma, Z. Jiang, M. Someki, N. E. Y. Soplin, R. Yamamoto, X. Wang, S. Watanabe, T. Yoshimura, and W. Zhang, "A comparative study on transformer vs RNN in speech applications," in Proc. IEEE ASRU, 2019, pp. 449–456.

深度学习开源工具介绍

常见的深度学习开源工具

- PyTorch

- TensorFlow

PyTorch

- 安装方式（Linux）：

```
# for CPU-only
conda install -y pytorch=1.6.0 cpuonly -c pytorch

# for GPU version
conda install -y pytorch=1.6.0 cudatoolkit=10.1 -c pytorch
```

- 调用方式：

```
import torch

x = torch.as_tensor([[1, -1], [0, 2]], dtype=torch.float32)
net = torch.nn.Linear(2, 1)
y = net(x)
o = torch.relu(y)
```

- 官方教程：https://pytorch.org/tutorials/

TensorFlow

- 安装方式（Linux）：

```
# for CPU-only
conda install -c tensorflow=2.4.1

# for GPU version
conda install -c tensorflow-gpu=2.4.1
```

- 调用方式：

```
import tensorflow as tf
from tensorflow.keras.layers import Dense

x = tf.convert_to_tensor([[1, -1], [0, 2]], dtype=tf.float32)
net = Dense(1, activation="relu")
o = net(x)
```

- 官方教程：https://www.tensorflow.org/tutorials