Course number: 80240743

Deep Learning

Xiaolin Hu (胡晓林) & Jun Zhu (朱军)

Dept. of Computer Science and Technology

Tsinghua University

Topic 4: Convolutional Neural Networks-I

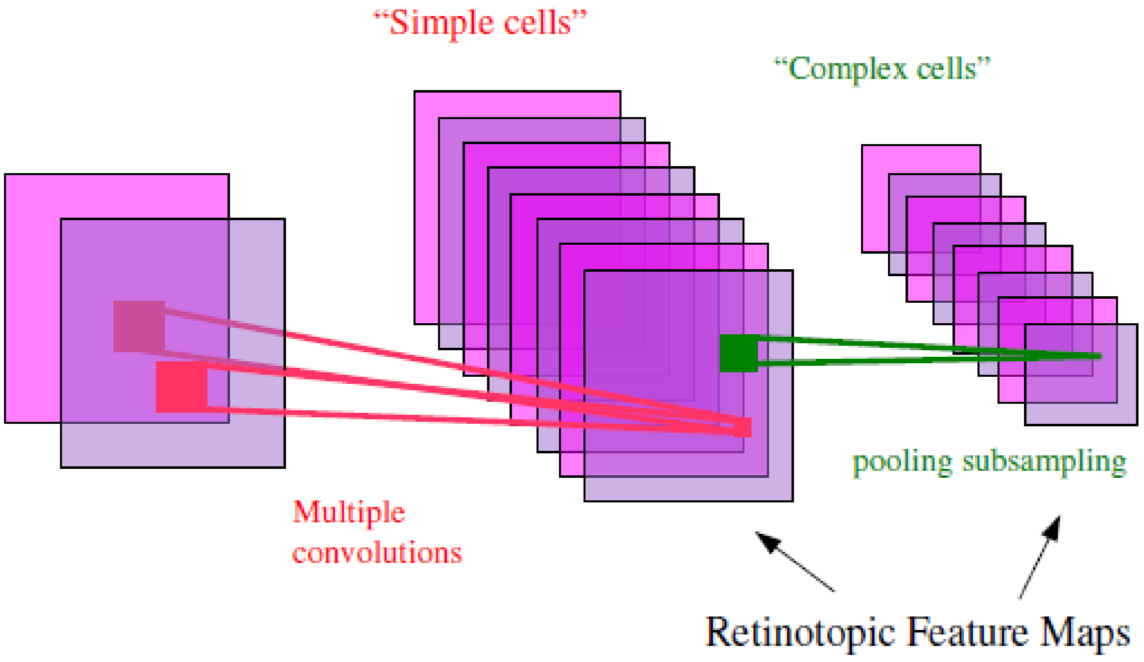
Xiaolin Hu
Dept. of Computer Science and Technology
Tsinghua University

Outline

- Introduction
- Convolution
  - Forward pass
  - Backward pass

Local detectors and shift invariance in the cortex

- (Hubel & Wiesel 1962)
  - Simple cells detect local features
  - complex cells "pool" the outputs of simple cells within a retinotopic neighborhood

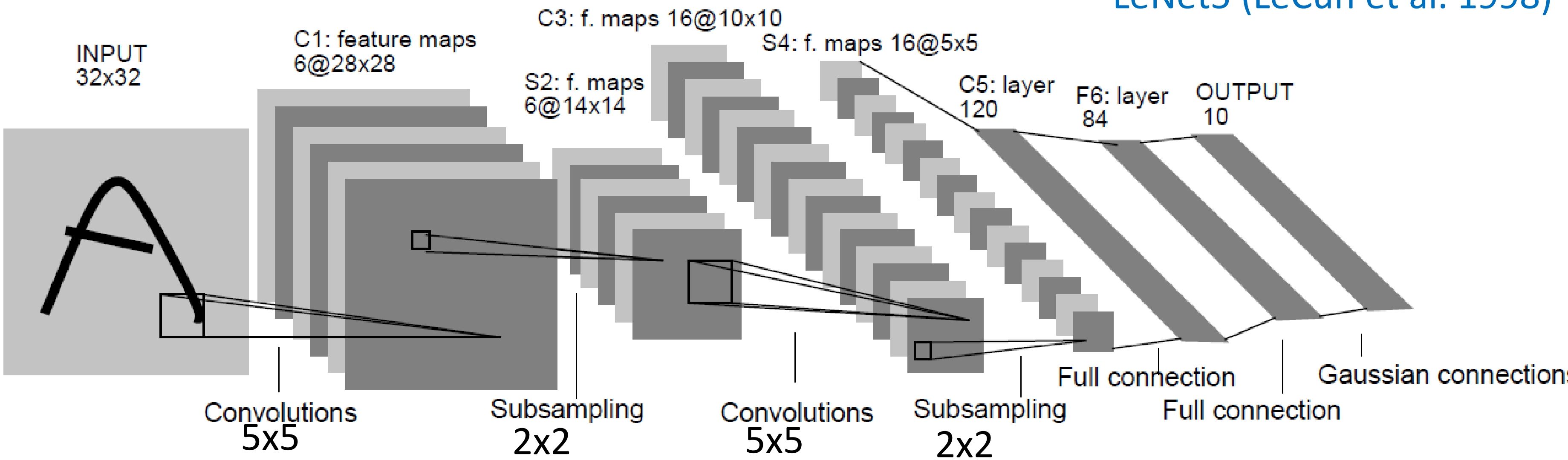


The multistage Hubel-Wiesel architecture

- Building a complete artificial vision system
  - Stack multiple stages of simple cells / complex cells layers
  - Higher stages compute more global, more invariant features
  - Stack a classification layer on top
- Models
  - Neocognitron [Fukushima 1971-1982]
  - Convolutional net [LeCun 1988]
  - HMAX [Poggio 2002-2006]
  - fragment hierarchy [Ullman 2002-2006]
  - HMAX [Lowe 2006]

Convolutional neural network (CNN)

LeNet5 (LeCun et al. 1998)

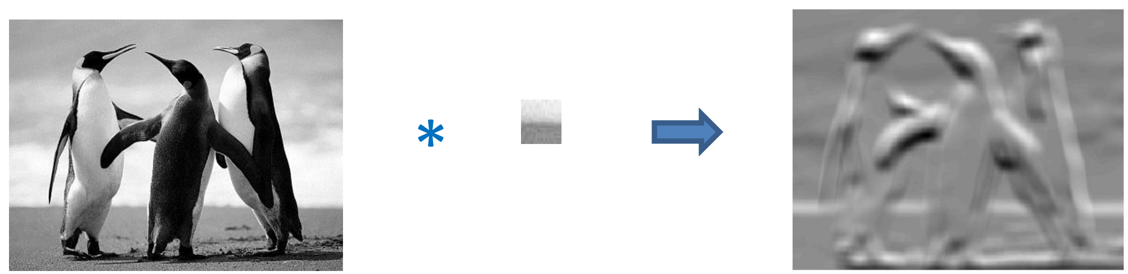


- Local connections and weight sharing
- C layers: convolution
  - Output $y_i = f(\sum_{\Omega} w_j x_j + b)$ where Ω is the patch size, $f(\cdot)$ is the sigmoid function, w and b are parameters
- S layers: subsampling (avg pooling)
  - Output $y_i = f\left(\frac{1}{|\Omega|}\sum_{\Omega} x_j\right)$ where Ω is the pooling size

Two new layers

input kernel feature map

Convolution



Pooling

poolingsize=2x2

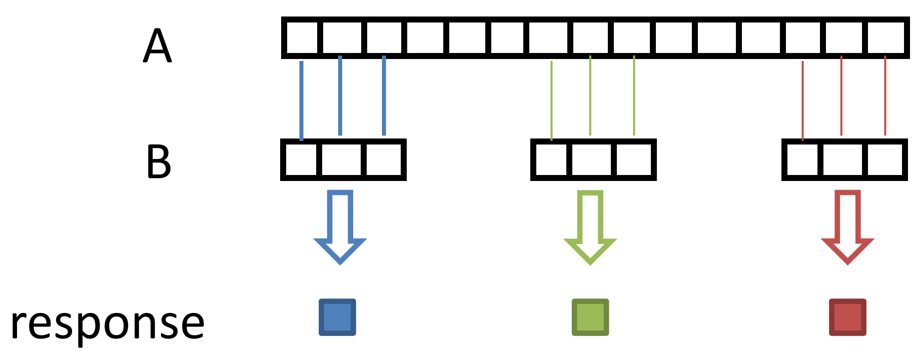
- Convolutional layer and pooling layer
  - Define two additional layers with forward computation and backward computation

Outline

- Introduction
- Convolution
  - Forward pass
  - Backward pass

Motivation

- Suppose there are two 1D sequences A and B where the length of B is smaller than that of A
- Compute the similarity between B and each part of A
- Naively, we could slide B on A and calculate the similarity one by one
  - For simplicity, we call it "correlation calculation"



But this process could be slow

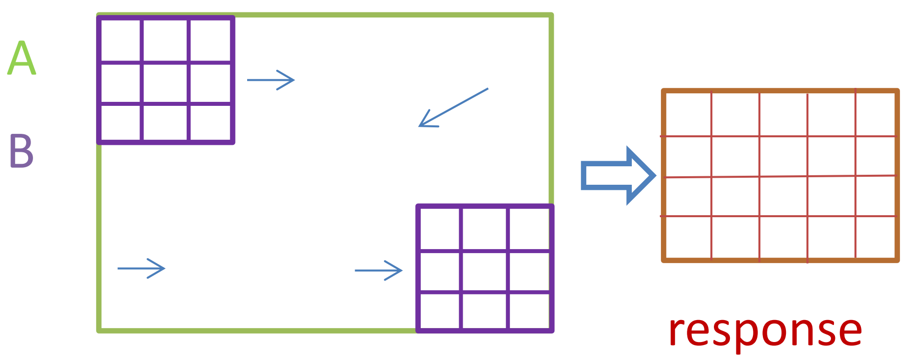
Cosine similarity between two vectors x and y:

$$s \equiv \cos\theta = \frac{x^\top y}{||x||\,||y||} = \sum_i x_i y_i$$

if the two vectors have unit length

Motivation

- Suppose there are two 2D images A and B where the size of B is smaller than that of A
- Compute the similarity between B and each part of A
- Naively, we could slide B on A and calculate the similarity one by one
  - For simplicity, we call it "correlation calculation"



Cosine similarity between two matrices x and y:

$$s = \sum_{i,j} x_{ij} y_{ij}$$

if the two matrices have unit Frobenius norm

But this process could be slow! We have other choices…

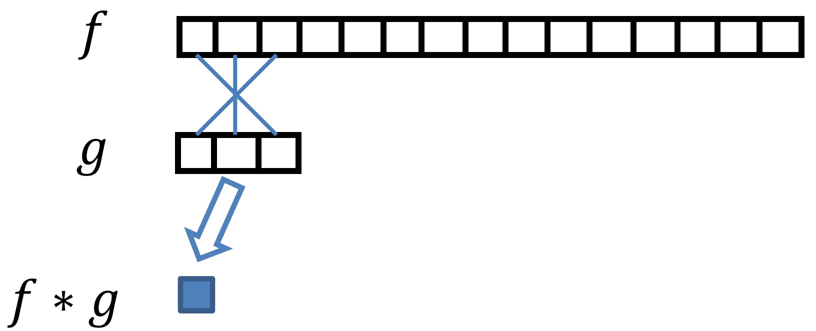
1D convolution

- Continuous convolution

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau$$

- Discrete convolution (for finite length sequences)

$$(f * g)[m] \triangleq \sum_{n=1}^{N} f[m-n]g[n]$$



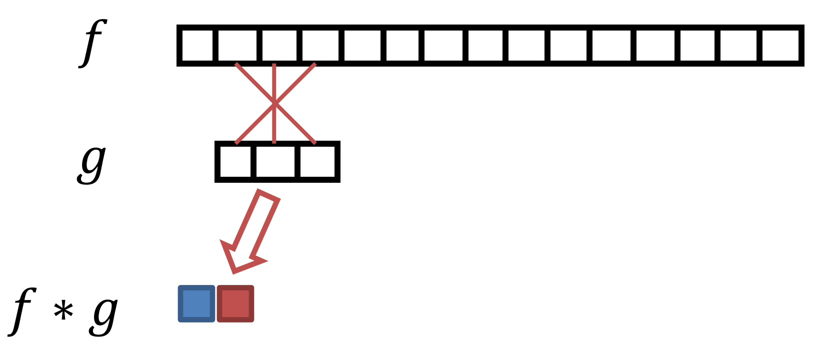
1D convolution

- Continuous convolution

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau = \int_{-\infty}^{\infty} f(t-\tau) g(\tau) d\tau$$

- Discrete convolution (for finite length sequences)

$$(f * g)[m] \triangleq \sum_{n=1}^{N} f[m-n] g[n]$$



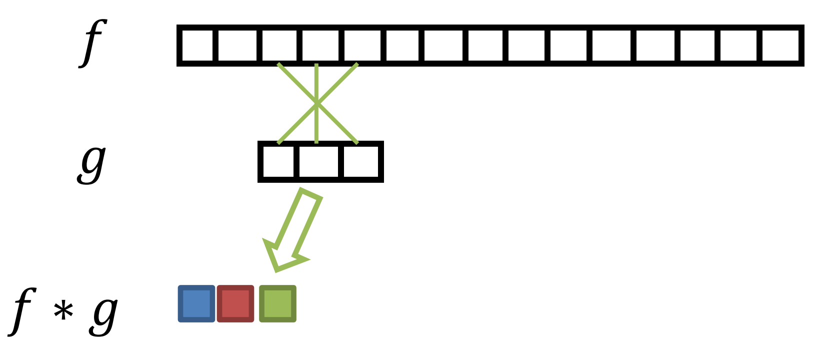
1D convolution

- Continuous convolution

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau = \int_{-\infty}^{\infty} f(t-\tau) g(\tau) d\tau$$

- Discrete convolution (for finite length sequences)

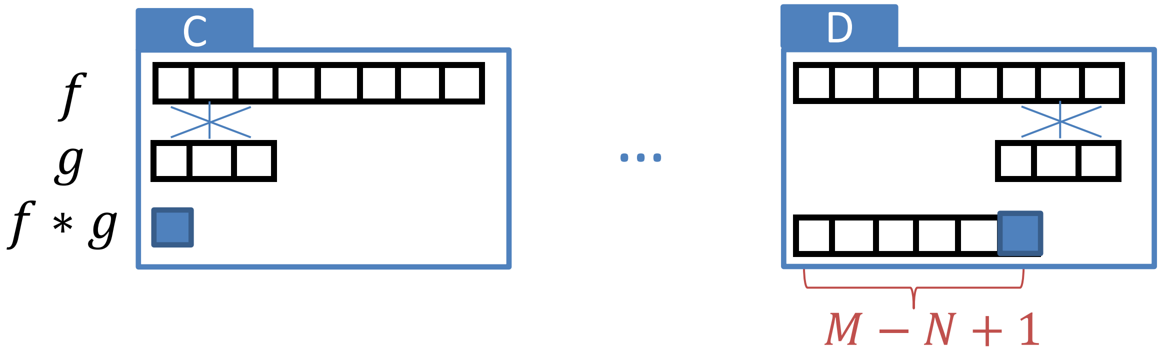
$$(f * g)[m] \triangleq \sum_{n=1}^{N} f[m-n] g[n]$$



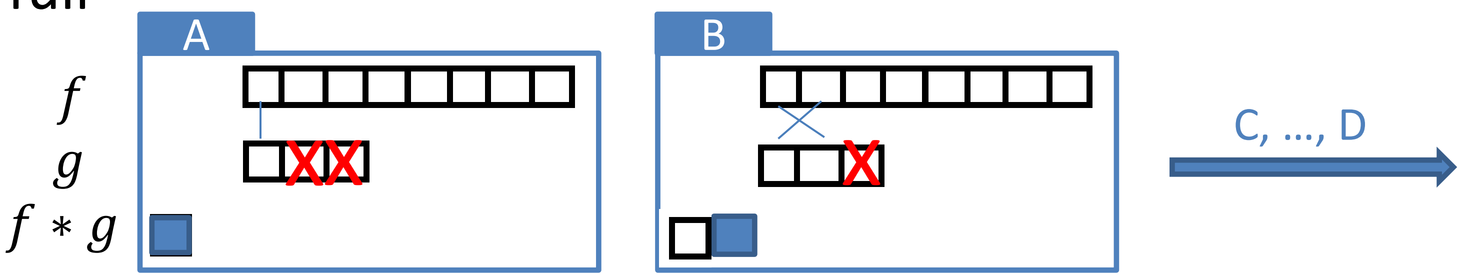
Three shapes of convolution

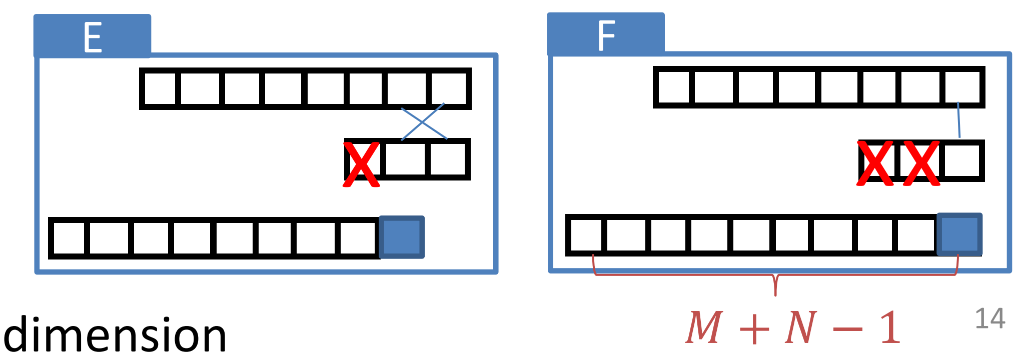
Length of f: M, length of g: N, where $M \geq N$

- valid



- full





- Same
  - truncate full result to M dimension

Example

- “Same” convolution can be also obtained by “valid” convolution of g with *zero-padded* f
- Suppose there are two sequences

$$f = [0, 1, 2, -1, 3]$$
$$g = [1, 1, 0]$$

- Then

$$(f * g)_{\text{valid}} = [3, 1, 2]$$
$$(f * g)_{\text{full}} = [0, 1, 3, 1, 2, 3, 0]$$
$$(f * g)_{\text{same}} = [1, 3, 1, 2, 3]$$

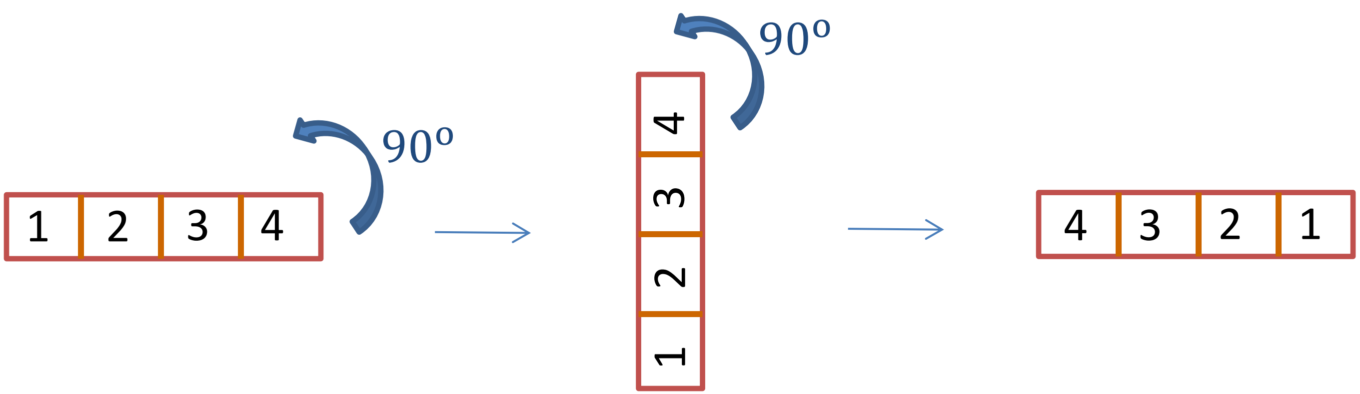
- Python commands

```
import numpy as np
from scipy import signal
```

```
f = np.array([0,1,2,-1,3])
g = np.array([1,1,0])
h = signal.convolve(f,g,mode='valid')
h = signal.convolve(f,g,mode='full')
h = signal.convolve(f,g,mode='same')
```

Relationship between similarity and convolution

- Calculating the the similarity between sequence g and each part of sequence f is equivalent to calculating $f * \tilde{g}$ where
$$\tilde{g}_1 = g_N, \tilde{g}_2 = g_{N-1}, \dots, \tilde{g}_N = g_1$$
- The above flip operation can be realized by applying the command numpy.rot90() twice (denoted by rot180() hereafter)

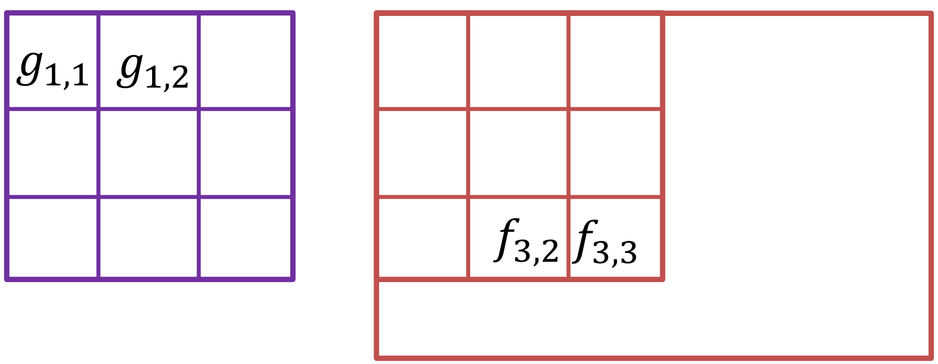


It's equivalent to *flip the vector along the axis 0*

2D convolution

- Suppose that there are two matrices f and g with sizes $M \times N$ and $K_1 \times K_2$, respectively, where $M \geq K_1, N \geq K_2$
- Discrete convolution of the two matrices

$$h[m,n] = (f * g)[m,n] \triangleq \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} f[m-k_1, n-k_2]g[k_1,k_2]$$



When $m = 4, n = 4$

$$(f * g)_{m,n} = f_{3,3}g_{1,1} + f_{3,2}g_{1,2} + f_{3,1}g_{1,3} + f_{2,3}g_{2,1} + \cdots$$

- valid shape: the size of h is $(M - K_1 + 1) \times (N - K_2 + 1)$
- full shape: the size of h is $(M + K_1 - 1) \times (N + K_2 - 1)$
- same shape: the size of h is $M \times N$

Matlab example

```
>> A = round(3*rand(4))

A =

     0     0     1     2
     2     2     0     0
     2     1     2     2
     3     0     1     1

>> B = round(2*rand(3))-1

B =

     0     0    -1
     1    -1     1
    -1     1     1
```

```
>> C = conv2(A,B,'full')

C =

     0     0     0     0    -1    -2
     0     0    -1    -1    -1     2
     2     0    -3     0     1     0
     0    -1     4     3    -1     1
     1    -2     5     1     4     3
    -3     3     2     0     2     1

>> D = conv2(A,B,'valid')

D =

    -3     0
     4     3
```

Matlab example

```
>> A = round(3*rand(4))

A =

    0    0    1    2
    2    2    0    0
    2    1    2    2
    3    0    1    1

>> B = round(2*rand(3))-1

B =

    0    0   -1
    1   -1    1
   -1    1    1
```

```
>> C = conv2(A,B,'full')

C =

    0    0    0    0   -1   -2
    0    0   -1   -1   -1    2
    2    0   -3    0    1    0
    0   -1    4    3   -1    1
    1   -2    5    1    4    3
   -3    3    2    0    2    1

>> D = conv2(A,B,‘same')
D =
    0   -1   -1   -1
    0   -3    0    1
   -1    4    3   -1
   -2    5    1    4
```

Python example

```
import numpy
from scipy import signal
A = numpy.array([[0,0,1,2],[2,2,0,0],[2,1,2,2],[3,0,1,1]])
B = numpy.array([[0,0,-1],[1,-1,1],[-1,1,1]])
C = signal.convolve2d(A,B,mode='full')
print(C)
C = signal.convolve2d(A,B,mode='valid')
print(C)
C = signal.convolve2d(A,B,mode=‘same')
print(C)
```

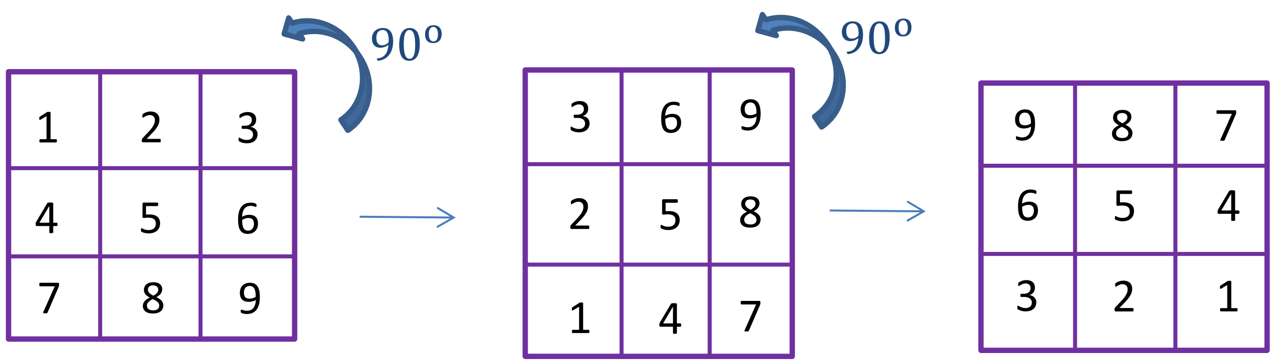
You would obtain the same results as before

Relationship between similarity and convolution

- Calculating the the similarity between matrix g and each part of matrix f is equivalent to calculating $f * \tilde{g}$ where

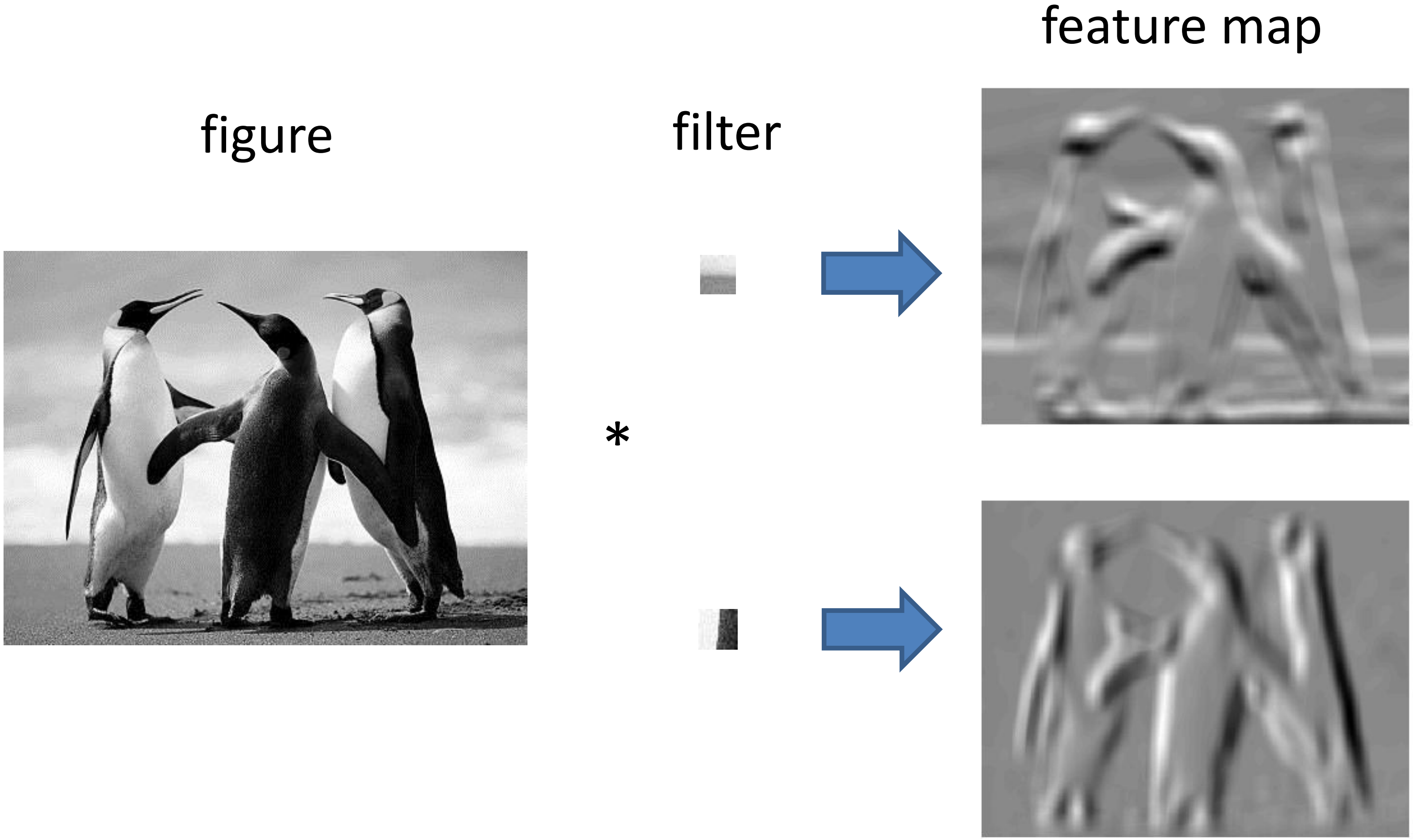
$$\begin{array}{c}
\tilde{g}_{1,1} = g_{M,N}, \tilde{g}_{1,2} = g_{M,N-1}, \cdots, \tilde{g}_{1,N} = g_{M,1} \\
\tilde{g}_{2,1} = g_{M-1,N}, \tilde{g}_{2,2} = g_{M-1,N-1}, \cdots, \tilde{g}_{2,N} = g_{M-1,1} \\
\vdots \qquad\qquad\qquad\qquad \vdots \\
\tilde{g}_{M,1} = g_{1,N}, \tilde{g}_{M,2} = g_{1,N-1}, \cdots, \tilde{g}_{M,N} = g_{1,1}
\end{array}$$

- The above operation can be realized by applying the command numpy.rot90() twice (denoted by rot180() hereafter)



It's equivalent to *flip the matrix along the axes 0 then 1*

Example

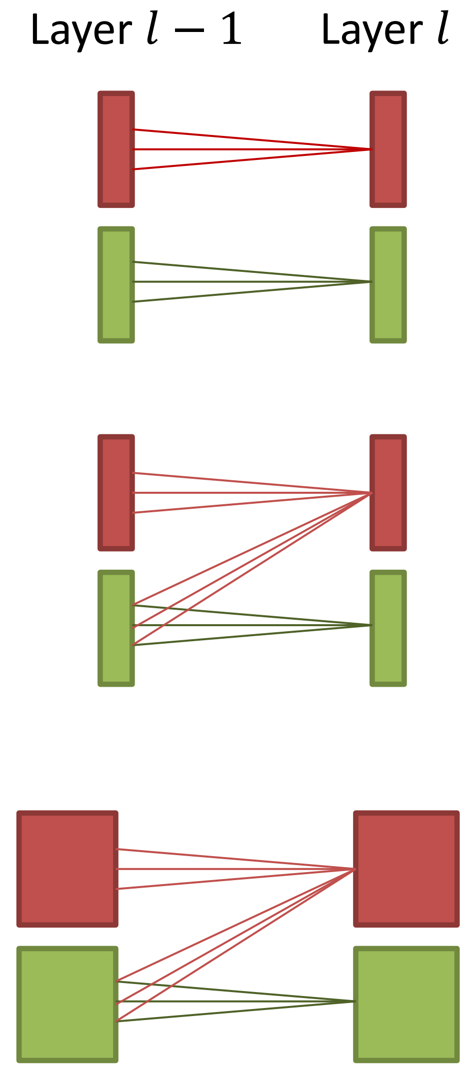


The higher a pixel value (brighter) in the feature map, the more similar between the filter and the corresponding patch in the figure

Outline

- Introduction
- Convolution
  - Forward pass
  - Backward pass

Derive BP algorithm in different cases



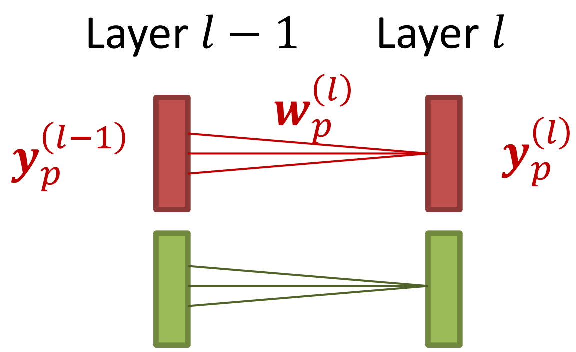
1. The 1D convolution case without feature combination

2. The 1D convolution case with feature combination

3. The 2D convolution case

Case 1: 1D convolution without feature combination

- Suppose that the l-th layer is a convolutional layer



In what follows, we drop the indexp p

- Convolve every filter $\boldsymbol{w}_p^{(l)}$ with the p-th feature map $\boldsymbol{y}_p^{(l-1)}$ in the previous layer and obtain a new feature map

$$\boldsymbol{y}_p^{(l)} = \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \operatorname{rot180}\left(\boldsymbol{w}_p^{(l)}\right) + b_p^{(l)}$$

A vector

A scalar

[We actually want to compute $\boldsymbol{y}_p^{(l)} = \boldsymbol{y}_p^{(l-1)} \operatorname{corr} \boldsymbol{w}_p^{(l)} + b_p^{(l)}$]

Recap: Derivative of two-step composition

Suppose we have:

- Independent input variables $x_1, x_2, \ldots, x_n$
- Dependent intermediate variables, $u_1, u_2, \ldots, u_m$, each of which is a function of $x_1, x_2, \ldots, x_n$
- Dependent output variables $w_1, w_2, \ldots, w_p$, each of which is a function of $u_1, u_2, \ldots, u_m$

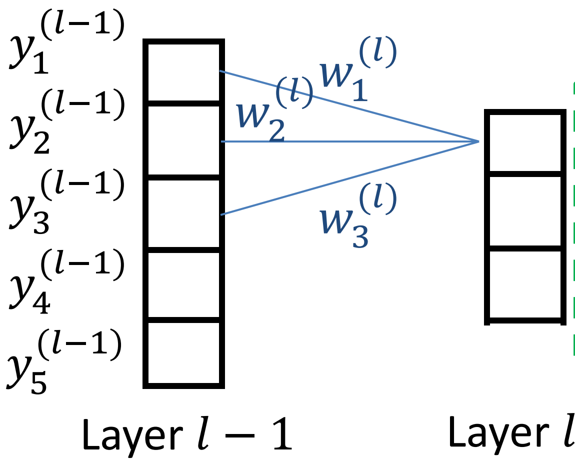
Then for any $i \in \{1,2,\ldots,p\}$ and $j \in \{1,2,\ldots,n\}$ we have

$$\frac{\partial w_i}{\partial x_j} = \sum_{k=1}^{m} \frac{\partial w_i}{\partial u_k} \frac{\partial u_k}{\partial x_j}$$

Sum over the intermediate variables

Gradient calculation in an example

Consider one single feature map in layer l



$$
\begin{aligned}
y_1^{(l)} &= w_1^{(l)} y_1^{(l-1)} + w_2^{(l)} y_2^{(l-1)} + w_3^{(l)} y_3^{(l-1)} + b^{(l)} \\
y_2^{(l)} &= w_1^{(l)} y_2^{(l-1)} + w_2^{(l)} y_3^{(l-1)} + w_3^{(l)} y_4^{(l-1)} + b^{(l)} \\
y_3^{(l)} &= w_1^{(l)} y_3^{(l-1)} + w_2^{(l)} y_4^{(l-1)} + w_3^{(l)} y_5^{(l-1)} + b^{(l)}
\end{aligned}
$$

Intermediate variable between $w_i^{(l)}$ and $E^{(n)}$

- Gradient of $\boldsymbol{w}^{(l)}$: scalar form

$$\frac{\partial E^{(n)}}{\partial w_1^{(l)}} = \sum_{i=1}^{3} \frac{\partial E^{(n)}}{\partial y_i^{(l)}} \frac{\partial y_i^{(l)}}{\partial w_1^{(l)}} = \delta_1^{(l)} y_1^{(l-1)} + \delta_2^{(l)} y_2^{(l-1)} + \delta_3^{(l)} y_3^{(l-1)}$$

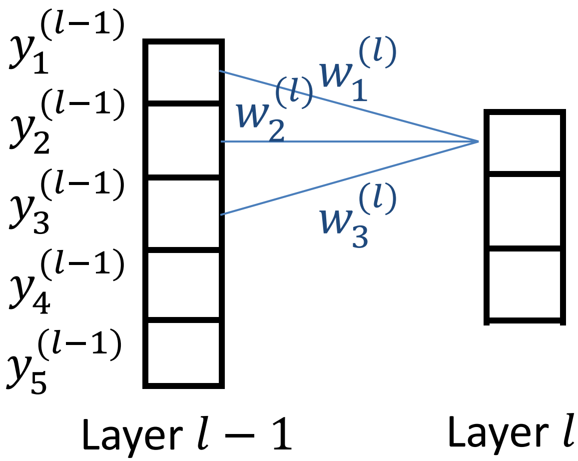
$$\frac{\partial E^{(n)}}{\partial w_2^{(l)}} = \sum_{i=1}^{3} \frac{\partial E^{(n)}}{\partial y_i^{(l)}} \frac{\partial y_i^{(l)}}{\partial w_2^{(l)}} = \delta_1^{(l)} y_2^{(l-1)} + \delta_2^{(l)} y_3^{(l-1)} + \delta_3^{(l)} y_4^{(l-1)}$$

$$\frac{\partial E^{(n)}}{\partial w_3^{(l)}} = \sum_{i=1}^{3} \frac{\partial E^{(n)}}{\partial y_i^{(l)}} \frac{\partial y_i^{(l)}}{\partial w_3^{(l)}} = \delta_1^{(l)} y_3^{(l-1)} + \delta_2^{(l)} y_4^{(l-1)} + \delta_3^{(l)} y_5^{(l-1)}$$

Note the subscripts in this slide index elements in a feature map.

Gradient calculation in general

Consider one single feature map in layer l



$$y_1^{(l)} = w_1^{(l)} y_1^{(l-1)} + w_2^{(l)} y_2^{(l-1)} + w_3^{(l)} y_3^{(l-1)} + b^{(l)}$$
$$y_2^{(l)} = w_1^{(l)} y_2^{(l-1)} + w_2^{(l)} y_3^{(l-1)} + w_3^{(l)} y_4^{(l-1)} + b^{(l)}$$
$$y_3^{(l)} = w_1^{(l)} y_3^{(l-1)} + w_2^{(l)} y_4^{(l-1)} + w_3^{(l)} y_5^{(l-1)} + b^{(l)}$$

- Gradient of $\boldsymbol{w}^{(l)}$: vector form

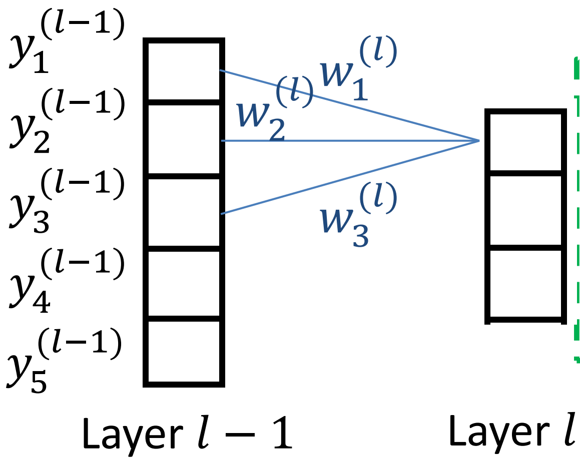
$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}^{(l)}} = \boldsymbol{y}^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{\delta}^{(l)})$$

- Gradient of $b^{(l)}$

$$\frac{\partial E^{(n)}}{\partial b^{(l)}} = \sum_{i=1}^{3} \frac{\partial E^{(n)}}{\partial y_i^{(l)}} \frac{\partial y_i^{(l)}}{\partial b^{(l)}} = \sum_i \delta_i^{(l)}$$

Local sensitivity in the example

Consider one single feature map in layer l



$$y_1^{(l)} = w_1^{(l)} y_1^{(l-1)} + w_2^{(l)} y_2^{(l-1)} + w_3^{(l)} y_3^{(l-1)} + b^{(l)}$$
$$y_2^{(l)} = w_1^{(l)} y_2^{(l-1)} + w_2^{(l)} y_3^{(l-1)} + w_3^{(l)} y_4^{(l-1)} + b^{(l)}$$
$$y_3^{(l)} = w_1^{(l)} y_3^{(l-1)} + w_2^{(l)} y_4^{(l-1)} + w_3^{(l)} y_5^{(l-1)} + b^{(l)}$$

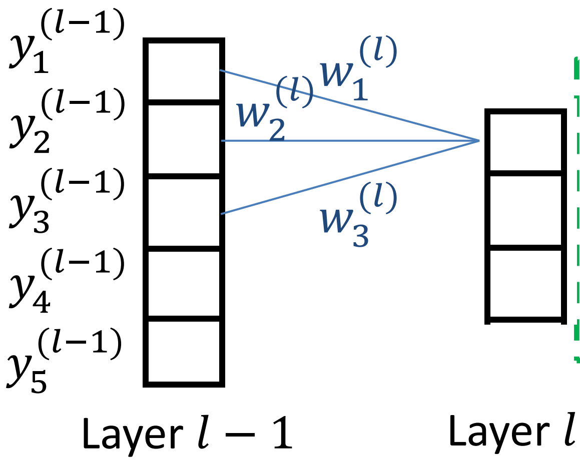
Intermediate variable between $y_1^{(l-1)}$ and $E^{(n)}$

- $y_1^{(l-1)}$ appears once in $\boldsymbol{y}^{(l)}$, and thus in the error function

$$\delta_1^{(l-1)} = \frac{\partial E^{(n)}}{\partial y_1^{(l-1)}} = \frac{\partial E^{(n)}}{\partial y_1^{(l)}} \frac{\partial y_1^{(l)}}{\partial y_1^{(l-1)}} = \delta_1^{(l)} w_1^{(l)}$$

Local sensitivity in the example

Consider one single feature map in layer l



$$
\begin{aligned}
y_1^{(l)} &= w_1^{(l)} y_1^{(l-1)} + w_2^{(l)} {\color{red} y_2^{(l-1)}} + w_3^{(l)} y_3^{(l-1)} + b^{(l)} \\
y_2^{(l)} &= w_1^{(l)} {\color{red} y_2^{(l-1)}} + w_2^{(l)} y_3^{(l-1)} + w_3^{(l)} y_4^{(l-1)} + b^{(l)} \\
y_3^{(l)} &= w_1^{(l)} y_3^{(l-1)} + w_2^{(l)} y_4^{(l-1)} + w_3^{(l)} y_5^{(l-1)} + b^{(l)}
\end{aligned}
$$

Intermediate variable between $y_2^{(l-1)}$ and $E^{(n)}$

- $y_2^{(l-1)}$ appears twice in $\boldsymbol{y}^{(l)}$, and thus in the error function

$$
\delta_2^{(l-1)} = \frac{\partial E^{(n)}}{\partial y_2^{(l-1)}} = \frac{\partial E^{(n)}}{\partial y_1^{(l)}} \frac{\partial y_1^{(l)}}{\partial y_2^{(l-1)}} + \frac{\partial E^{(n)}}{\partial y_2^{(l)}} \frac{\partial y_2^{(l)}}{\partial y_2^{(l-1)}}
$$
$$
= \delta_1^{(l)} w_2^{(l)} + \delta_2^{(l)} w_1^{(l)}
$$

- Similarly we can obtain $\delta_3^{(l)}, \delta_4^{(l)}$ and $\delta_5^{(l)}$

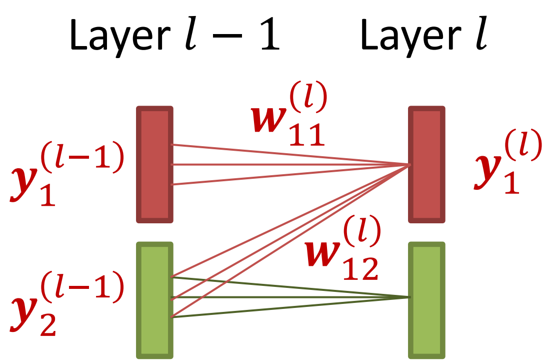
Local sensitivity in general

- Local sensitivity in the vector form

$$\boldsymbol{\delta}^{(l-1)} \triangleq \frac{\partial E^{(n)}}{\partial \boldsymbol{y}^{l-1}} = \underbrace{\begin{pmatrix} \delta_1^{(l)} w_1^{(l)} \\ \delta_1^{(l)} w_2^{(l)} + \delta_2^{(l)} w_1^{(l)} \\ \delta_1^{(l)} w_3^{(l)} + \delta_2^{(l)} w_2^{(l)} + \delta_3^{(l)} w_1^{(l)} \\ \delta_2^{(l)} w_3^{(l)} + \delta_3^{(l)} w_2^{(l)} \\ \delta_3^{(l)} w_3^{(l)} \end{pmatrix}}_{\text{Full convolution of } \boldsymbol{\delta}^{(l)} \text{ and } \boldsymbol{w}^{(l)}} = \boldsymbol{\delta}^{(l)} *_{\text{full}} \boldsymbol{w}^{(l)}$$

Case 2: 1D convolution with feature combination---An example

- Suppose that the l-th layer is a convolutional layer



(The subscripts now index the feature maps, not elements in vectors)

- Let $\boldsymbol{w}_{qp}^{(l)}$ denote the p-th filter in layer $l-1$ to the q-th filter in layer l
- Forward pass: the first feature map in layer l combines the output of two feature maps in layer $l-1$

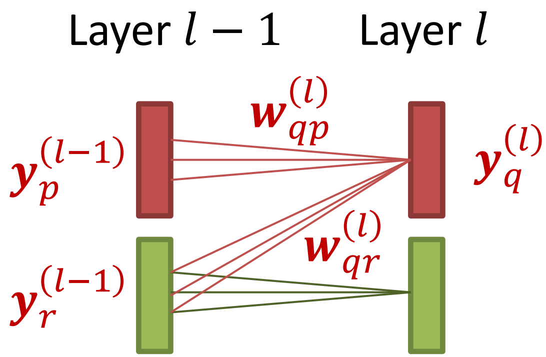
$$\boldsymbol{y}_1^{(l)} = \boldsymbol{y}_1^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_{11}^{(l)}\right) + \boldsymbol{y}_2^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_{12}^{(l)}\right) + b_1^{(l)}$$

A vector

A scalar

Forward pass in general

- Suppose that the l-th layer is a convolutional layer



- This is generalized to multiple feature maps in layer l, and each feature map is obtained by

A scalar

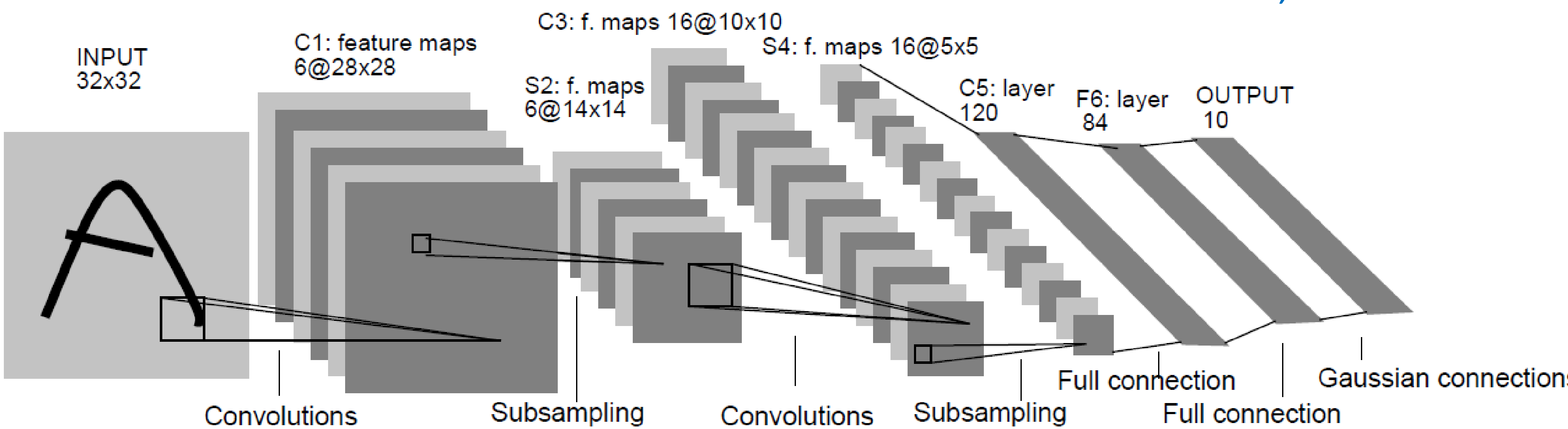
$$\boldsymbol{y}_q^{(l)} = \sum_{p \in M_q} \boldsymbol{y}_p^{(l-1)} *_{\mathrm{valid}} \mathrm{rot180}\left(\boldsymbol{w}_{qp}^{(l)}\right) + b_q^{(l)}$$

where M_q denotes the set of feature maps in layer $l-1$ connected to the q-th feature map in layer l

Feature map selection

- M_q often contains all feature maps in layer $l-1$, but sometimes it does not

LeNet5, 1998

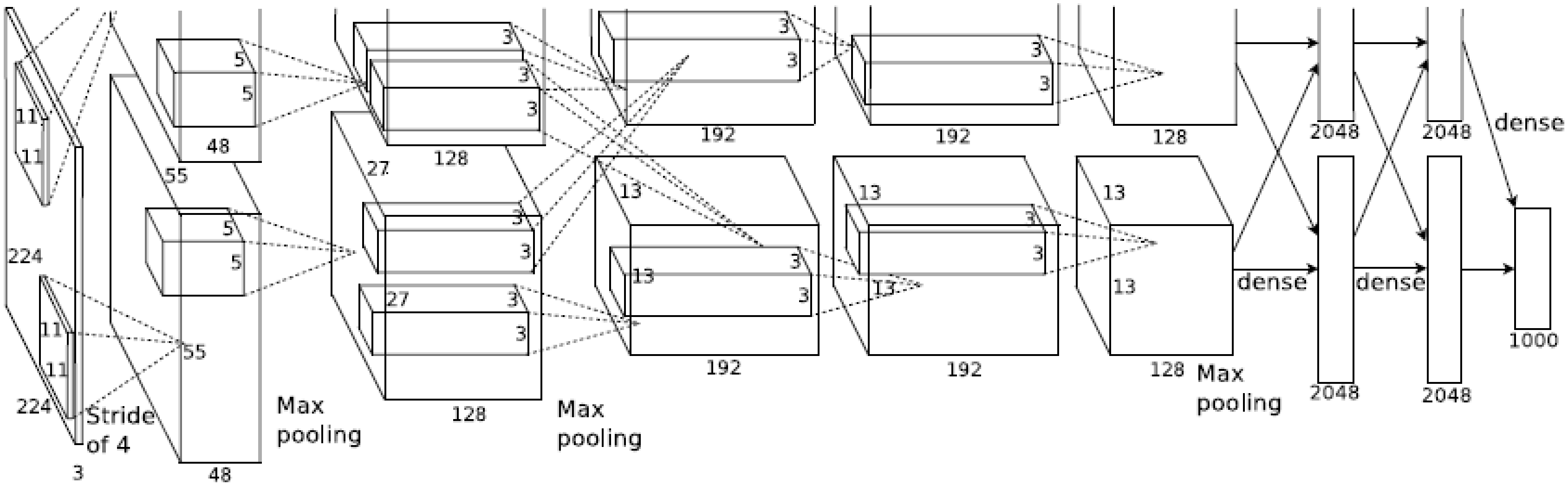


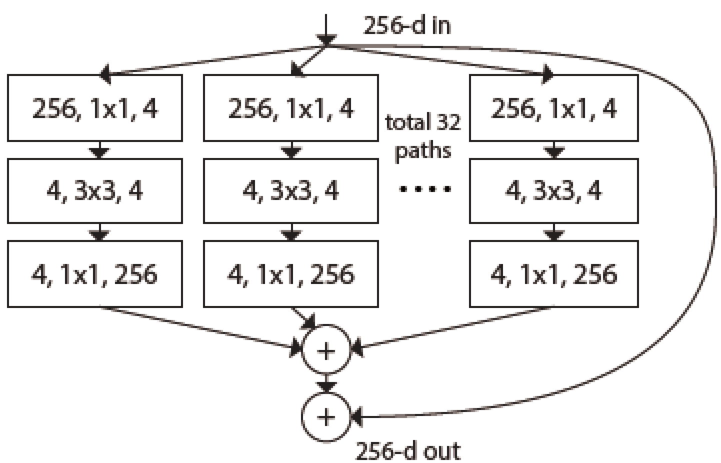
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

Each column indicates which feature map in S2 are combined to produce a particular feature map of C3

Feature map selection

AlexNet, 2012

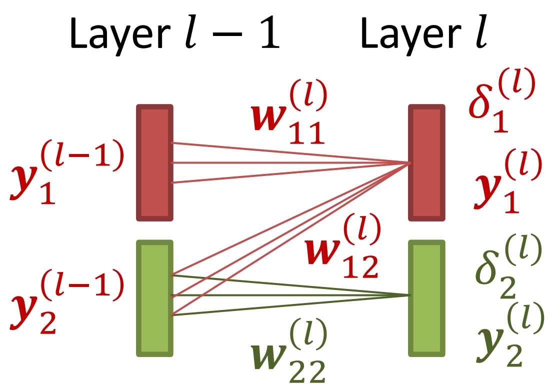




ResNeXt, 2017

Gradient calculation in the example

- In layer l, calculate gradients of parameters in this layer



- Are these eqns correct?

(A) $$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{11}^{(l)}} = \boldsymbol{y}_1^{(l-1)} *_{\text{valid}} \operatorname{rot180}(\boldsymbol{\delta}_1^{(l)}),$$

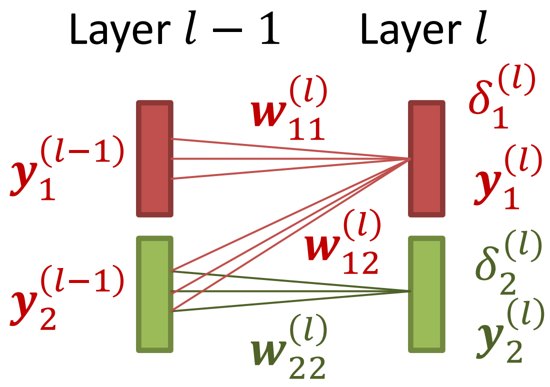
(B) $$\frac{\partial E^{(n)}}{\partial b_1^{(l)}} = \sum_i (\delta_1^{(l)})_i,$$

(C) $$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{22}^{(l)}} = \boldsymbol{y}_2^{(l-1)} *_{\text{valid}} \operatorname{rot180}(\boldsymbol{\delta}_2^{(l)}),$$

(D) $$\frac{\partial E^{(n)}}{\partial b_2^{(l)}} = \sum_i (\delta_2^{(l)})_i.$$

Gradient calculation in the example

- In layer l, calculate gradients of parameters in this layer

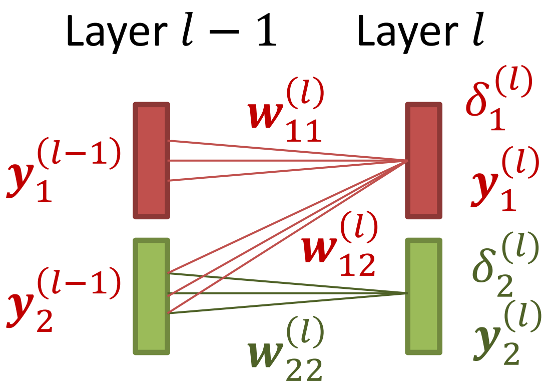


- How about $\partial E^{(n)} / \partial \boldsymbol{w}_{12}^{(l)}$?

- How about the corresponding bias term?

Gradient calculation in general

- In layer l, calculate

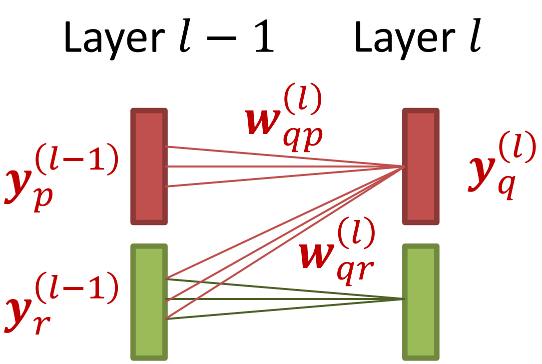


$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{11}^{(l)}} = \boldsymbol{y}_1^{(l-1)} *_{\text{valid}} \mathrm{rot180}(\boldsymbol{\delta}_1^{(l)}), \quad \frac{\partial E^{(n)}}{\partial b_1^{(l)}} = \sum_i (\delta_1^{(l)})_i,$$

$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{12}^{(l)}} = \boldsymbol{y}_2^{(l-1)} *_{\text{valid}} \mathrm{rot180}(\boldsymbol{\delta}_1^{(l)}),$$

$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{22}^{(l)}} = \boldsymbol{y}_2^{(l-1)} *_{\text{valid}} \mathrm{rot180}(\boldsymbol{\delta}_2^{(l)}), \quad \frac{\partial E^{(n)}}{\partial b_2^{(l)}} = \sum_i (\delta_2^{(l)})_i.$$

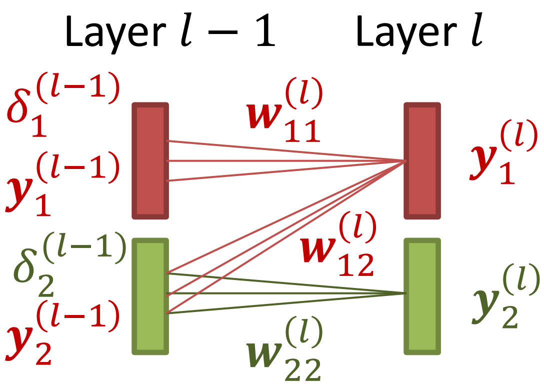
- In general



$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{qp}^{(l)}} = \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \mathrm{rot180}(\boldsymbol{\delta}_q^{(l)}), \quad \frac{\partial E^{(n)}}{\partial b_q^{(l)}} = \sum_i (\boldsymbol{\delta}_q^{(l)})_i$$

Local sensitivity in the example

- In layer l, calculate the local sensitivity in layer $l-1$



$$\boxed{\boldsymbol{y}_1^{(l)}} = \boldsymbol{y}_1^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_{11}^{(l)}\right) + \boldsymbol{y}_2^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_{12}^{(l)}\right) + b_1^{(l)}$$

$$\boldsymbol{y}_2^{(l)} = \boldsymbol{y}_2^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_{22}^{(l)}\right) + b_2^{(l)}$$

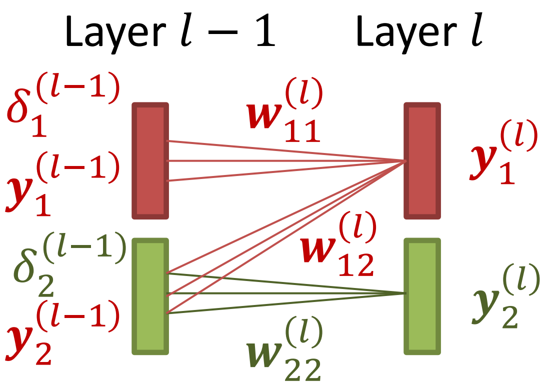
Intermediate variable between $y_1^{(l-1)}$ and $E^{(n)}$

- Is the eqn of local sensitivity $\boldsymbol{\delta}_1^{(l-1)} = \partial E^{(n)} / \partial \boldsymbol{y}_1^{(l-1)}$ the same as before, say,

$$\boldsymbol{\delta}_1^{(l-1)} = \boldsymbol{\delta}_1^{(l)} *_{\text{full}} \boldsymbol{w}_{11}^{(l)} \quad ?$$

Local sensitivity in the example

- In layer l, calculate the local sensitivity in layer $l-1$



$$\boldsymbol{y}_1^{(l)} = \boldsymbol{y}_1^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_{11}^{(l)}\right) + \boldsymbol{y}_2^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_{12}^{(l)}\right) + b_1^{(l)}$$

$$\boldsymbol{y}_2^{(l)} = \boldsymbol{y}_2^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_{22}^{(l)}\right) + b_2^{(l)}$$

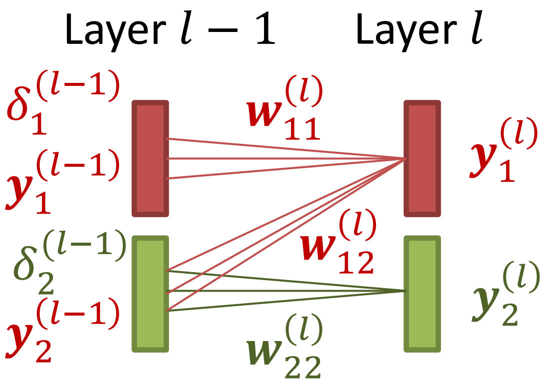
Intermediate variable between $y_2^{(l-1)}$ and $E^{(n)}$

- Is the eqn of local sensitivity $\boldsymbol{\delta}_2^{(l-1)} = \partial E^{(n)} / \partial \boldsymbol{y}_2^{(l-1)}$ the same as before, that is,

$$\boldsymbol{\delta}_2^{(l-1)} = \boldsymbol{\delta}_2^{(l)} *_{\text{full}} \boldsymbol{w}_{22}^{(l)} \quad ?$$

Local sensitivity in general

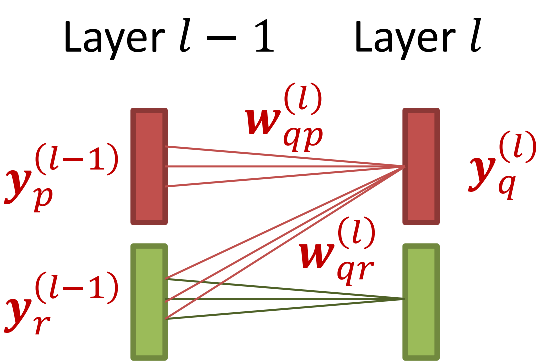
- In layer l, calculate the local sensitivity in layer $l-1$



$$\boldsymbol{\delta}_1^{(l-1)} = \boldsymbol{\delta}_1^{(l)} *_{\text{full}} \boldsymbol{w}_{11}^{(l)}$$

$$\boldsymbol{\delta}_2^{(l-1)} = \boldsymbol{\delta}_1^{(l)} *_{\text{full}} \boldsymbol{w}_{12}^{(l)} + \boldsymbol{\delta}_2^{(l)} *_{\text{full}} \boldsymbol{w}_{22}^{(l)}$$

- In general



$$\boldsymbol{\delta}_p^{(l-1)} = \sum_{q \in \widetilde{M}_p} \boldsymbol{\delta}_q^{(l)} *_{\text{full}} \boldsymbol{w}_{qp}^{(l)}$$

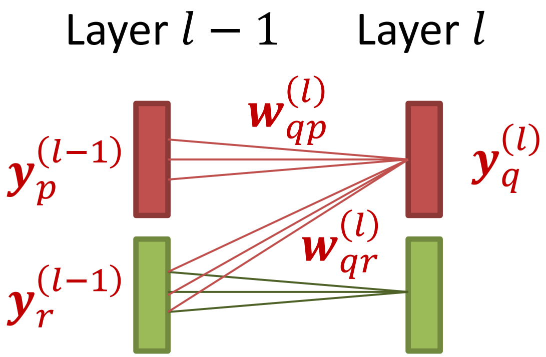
where $\widetilde{M}_p$ denotes the set of feature maps in layer l that the p-th feature map in layer $l-1$ connects to

Summary for 1D convolutional layer

Suppose that the l-th layer is a convolutional layer

- Forward pass

$$\boldsymbol{y}_q^{(l)} = \sum_{p \in M_q} \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{w}_{qp}^{(l)}) + b_q^{(l)}$$



where M_q denotes the set of feature maps in layer $l-1$ connected to the q-th feature map in layer l

- Backward pass

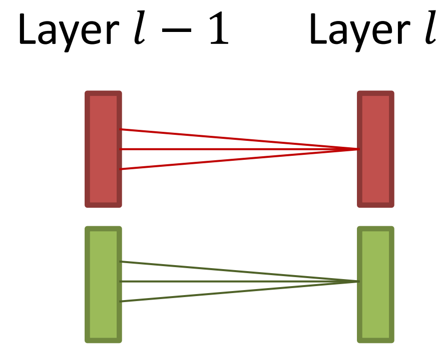
$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{qp}^{(l)}} = \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{\delta}_q^{(l)}), \quad \frac{\partial E^{(n)}}{\partial b_q^{(l)}} = \sum_i (\boldsymbol{\delta}_q^{(l)})_i$$

$$\boldsymbol{\delta}_p^{(l-1)} = \sum_{q \in \widetilde{M}_p} \boldsymbol{\delta}_q^{(l)} *_{\text{full}} \boldsymbol{w}_{qp}^{(l)}$$

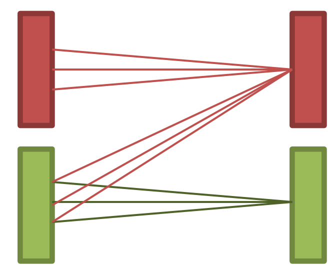
where $\widetilde{M}_p$ denotes the set of feature maps in layer l that the p-th feature map in layer $l-1$ connects to

Derive BP algorithm in different cases

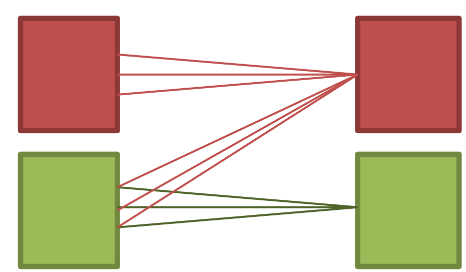
1. The 1D convolution case without feature combination



2. The 1D convolution case with feature combination



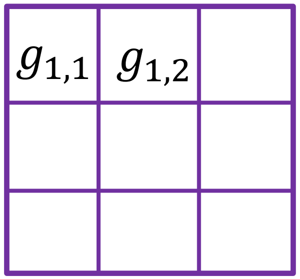
3. The 2D convolution case

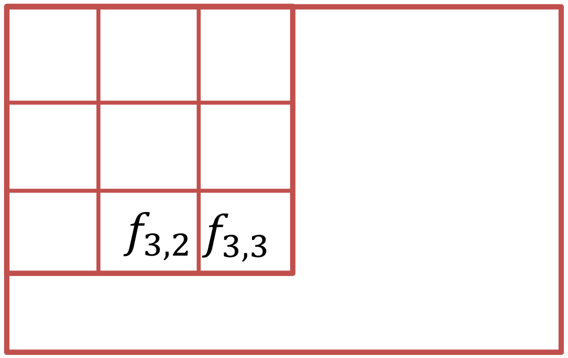


2D convolution

- Suppose that there are two matrices f and g with sizes $M \times N$ and $K_1 \times K_2$, respectively, where $M \geq K_1, N \geq K_2$
- Discrete convolution of the two matrices

$$h[m,n] = (f * g)[m,n] \triangleq \sum_{k_1=1}^{K_1} \sum_{k_2=1}^{K_2} f[m-k_1, n-k_2]g[k_1,k_2]$$





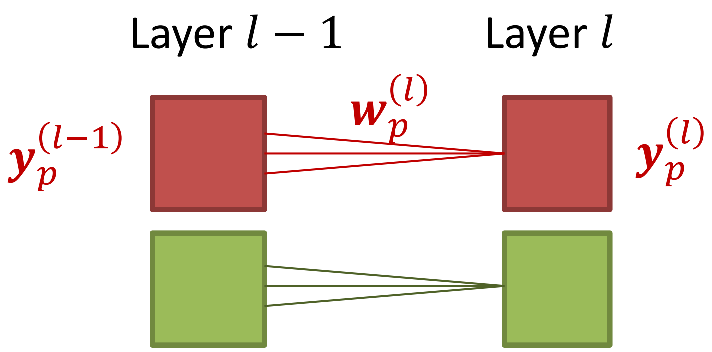
When $m = 4, n = 4$

$$(f * g)_{m,n} = f_{3,3}g_{1,1} + f_{3,2}g_{1,2} + f_{3,1}g_{1,3} + f_{2,3}g_{2,1} + \cdots$$

- valid shape: the size of h is $(M - K_1 + 1) \times (N - K_2 + 1)$
- full shape: the size of h is $(M + K_1 - 1) \times (N + K_2 - 1)$
- same shape: the size of h is $M \times N$

2D convolution without feature combination

- Suppose that the l-th layer is a convolutional layer



In what follows, we drop the index p

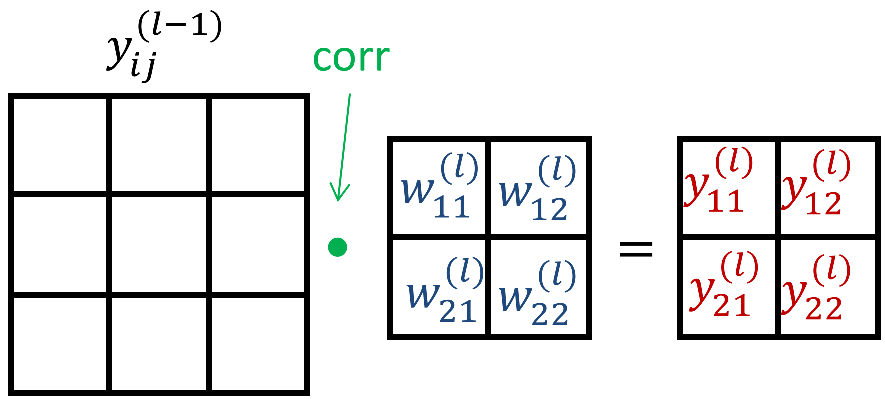
- Convolve every filter $\boldsymbol{w}_p^{(l)}$ with the p-th feature map $\boldsymbol{y}_p^{(l-1)}$ in the previous layer and obtain a new feature map

$$\boldsymbol{y}_p^{(l)} = \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_p^{(l)}\right) + b_p^{(l)}$$

[We actually want to compute $\boldsymbol{y}_p^{(l)} = \boldsymbol{y}_p^{(l-1)} \text{corr}\, \boldsymbol{w}_p^{(l)} + b_p^{(l)}$]

Forward pass in an example

Consider one single feature map in layer l



Layer $l-1$ Layer l

- The output in layer l

$$\boldsymbol{y}_p^{(l)} = \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \mathrm{rot180}\left(\boldsymbol{w}_p^{(l)}\right) + b_p^{(l)}$$

$$y_{11}^{(l)} = w_{11}^{(l)} y_{11}^{(l-1)} + w_{12}^{(l)} y_{12}^{(l-1)} + w_{21}^{(l)} y_{21}^{(l-1)} + w_{22}^{(l)} y_{22}^{(l-1)} + b^{(l)}$$

$$y_{12}^{(l)} = w_{11}^{(l)} y_{12}^{(l-1)} + w_{12}^{(l)} y_{13}^{(l-1)} + w_{21}^{(l)} y_{22}^{(l-1)} + w_{22}^{(l)} y_{23}^{(l-1)} + b^{(l)}$$

$$y_{21}^{(l)} = w_{11}^{(l)} y_{21}^{(l-1)} + w_{12}^{(l)} y_{22}^{(l-1)} + w_{21}^{(l)} y_{31}^{(l-1)} + w_{22}^{(l)} y_{32}^{(l-1)} + b^{(l)}$$

$$y_{22}^{(l)} = w_{11}^{(l)} y_{22}^{(l-1)} + w_{12}^{(l)} y_{23}^{(l-1)} + w_{21}^{(l)} y_{32}^{(l-1)} + w_{32}^{(l)} y_{33}^{(l-1)} + b^{(l)}$$

Gradient calculation in the example

$$\boldsymbol{y}_p^{(l)} = \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_p^{(l)}\right) + b_p^{(l)}$$

$$y_{11}^{(l)} = w_{11}^{(l)} y_{11}^{(l-1)} + w_{12}^{(l)} y_{12}^{(l-1)} + w_{21}^{(l)} y_{21}^{(l-1)} + w_{22}^{(l)} y_{22}^{(l-1)} + b^{(l)}$$

$$y_{12}^{(l)} = w_{11}^{(l)} y_{12}^{(l-1)} + w_{12}^{(l)} y_{13}^{(l-1)} + w_{21}^{(l)} y_{22}^{(l-1)} + w_{22}^{(l)} y_{23}^{(l-1)} + b^{(l)}$$

$$y_{21}^{(l)} = w_{11}^{(l)} y_{21}^{(l-1)} + w_{12}^{(l)} y_{22}^{(l-1)} + w_{21}^{(l)} y_{31}^{(l-1)} + w_{22}^{(l)} y_{32}^{(l-1)} + b^{(l)}$$

$$y_{22}^{(l)} = w_{11}^{(l)} y_{22}^{(l-1)} + w_{12}^{(l)} y_{23}^{(l-1)} + w_{21}^{(l)} y_{32}^{(l-1)} + w_{32}^{(l)} y_{33}^{(l-1)} + b^{(l)}$$

- Gradient of $\boldsymbol{w}^{(l)}$ and $b^{(l)}$

$$\partial E^{(n)} / \partial w_{11}^{(l)} = \delta_{11}^{(l)} y_{11}^{(l-1)} + \delta_{12}^{(l)} y_{12}^{(l-1)} + \delta_{21}^{(l)} y_{21}^{(l-1)} + \delta_{22}^{(l)} y_{22}^{(l-1)}$$

$$\partial E^{(n)} / \partial w_{12}^{(l)} = \delta_{11}^{(l)} y_{12}^{(l-1)} + \delta_{12}^{(l)} y_{13}^{(l-1)} + \delta_{21}^{(l)} y_{22}^{(l-1)} + \delta_{22}^{(l)} y_{23}^{(l-1)}$$

$$\partial E^{(n)} / \partial w_{21}^{(l)} = \delta_{11}^{(l)} y_{21}^{(l-1)} + \delta_{12}^{(l)} y_{22}^{(l-1)} + \delta_{21}^{(l)} y_{31}^{(l-1)} + \delta_{22}^{(l)} y_{32}^{(l-1)}$$

$$\partial E^{(n)} / \partial w_{22}^{(l)} = \delta_{11}^{(l)} y_{22}^{(l-1)} + \delta_{12}^{(l)} y_{23}^{(l-1)} + \delta_{21}^{(l)} y_{32}^{(l-1)} + \delta_{22}^{(l)} y_{33}^{(l-1)}$$

$$\partial E^{(n)} / \partial b^{(l)} = \delta_{11}^{(l)} + \delta_{12}^{(l)} + \delta_{21}^{(l)} + \delta_{22}^{(l)}$$

$$\boxed{\frac{\partial E^{(n)}}{\partial \boldsymbol{w}^{(l)}} = \boldsymbol{y}^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{\delta}^{(l)}), \qquad \frac{\partial E^{(n)}}{\partial b^{(l)}} = \sum_{i,j} \delta_{ij}^{(l)}}$$

General result

Local sensitivity in the example

Consider one single feature map in layer l

$$y_{ij}^{(l-1)}$$

$$\begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array} \bullet \begin{array}{|c|c|}\hline w_{11}^{(l)} & w_{12}^{(l)} \\ \hline w_{21}^{(l)} & w_{22}^{(l)} \\ \hline \end{array} = \begin{array}{|c|c|}\hline y_{11}^{(l)} & y_{12}^{(l)} \\ \hline y_{21}^{(l)} & y_{22}^{(l)} \\ \hline \end{array}$$

Layer $l-1$ Layer l

$$\delta_{ij}^{(l-1)} = \frac{\partial E^{(n)}}{\partial y_{ij}^{(l-1)}} = \sum_{m}^{2} \sum_{n}^{2} \frac{\partial E^{(n)}}{\partial y_{mn}^{(l)}} \frac{\partial y_{mn}^{(l)}}{\partial y_{ij}^{(l)}}$$

where $i, j \in \{1,2,3\}$

- Note that

$$y_{11}^{(l)} = w_{11}^{(l)} y_{11}^{(l-1)} + w_{12}^{(l)} y_{12}^{(l-1)} + w_{21}^{(l)} y_{21}^{(l-1)} + w_{22}^{(l)} y_{22}^{(l-1)} + b^{(l)}$$

$$y_{12}^{(l)} = w_{11}^{(l)} y_{12}^{(l-1)} + w_{12}^{(l)} y_{13}^{(l-1)} + w_{21}^{(l)} y_{22}^{(l-1)} + w_{22}^{(l)} y_{23}^{(l-1)} + b^{(l)}$$

$$y_{21}^{(l)} = w_{11}^{(l)} y_{21}^{(l-1)} + w_{12}^{(l)} y_{22}^{(l-1)} + w_{21}^{(l)} y_{31}^{(l-1)} + w_{22}^{(l)} y_{32}^{(l-1)} + b^{(l)}$$

$$y_{22}^{(l)} = w_{11}^{(l)} y_{22}^{(l-1)} + w_{12}^{(l)} y_{23}^{(l-1)} + w_{21}^{(l)} y_{32}^{(l-1)} + w_{32}^{(l)} y_{33}^{(l-1)} + b^{(l)}$$

Local sensitivity in the example

$$y_{11}^{(l)} = w_{11}^{(l)} y_{11}^{(l-1)} + w_{12}^{(l)} y_{12}^{(l-1)} + w_{21}^{(l)} y_{21}^{(l-1)} + w_{22}^{(l)} y_{22}^{(l-1)} + b^{(l)}$$

$$y_{12}^{(l)} = w_{11}^{(l)} y_{12}^{(l-1)} + w_{12}^{(l)} y_{13}^{(l-1)} + w_{21}^{(l)} y_{22}^{(l-1)} + w_{22}^{(l)} y_{23}^{(l-1)} + b^{(l)}$$

$$y_{21}^{(l)} = w_{11}^{(l)} y_{21}^{(l-1)} + w_{12}^{(l)} y_{22}^{(l-1)} + w_{21}^{(l)} y_{31}^{(l-1)} + w_{22}^{(l)} y_{32}^{(l-1)} + b^{(l)}$$

$$y_{22}^{(l)} = w_{11}^{(l)} y_{22}^{(l-1)} + w_{12}^{(l)} y_{23}^{(l-1)} + w_{21}^{(l)} y_{32}^{(l-1)} + w_{22}^{(l)} y_{33}^{(l-1)} + b^{(l)}$$

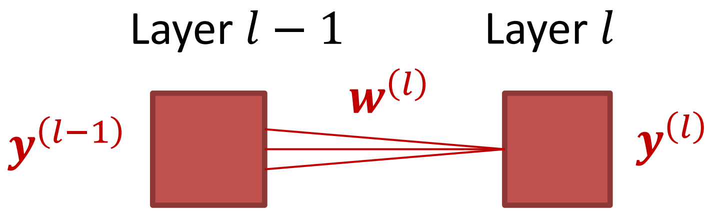
- It's easy to calculate $\delta_{ij}^{(l-1)}$
- If we define

$$\boldsymbol{\delta}^{(l)} = \begin{pmatrix} \delta_{11}^{(l)} & \delta_{12}^{(l)} \\ \delta_{21}^{(l)} & \delta_{22}^{(l)} \end{pmatrix}, \boldsymbol{w}^{(l)} = \begin{pmatrix} w_{11}^{(l)} & w_{12}^{(l)} \\ w_{21}^{(l)} & w_{22}^{(l)} \end{pmatrix}$$

What's the relationship between $\boldsymbol{\delta}^{(l)}$ and $\boldsymbol{\delta}^{(l-1)}$?

Summary for 2D convolution *without* feature combination

- Suppose that the l-th layer is a convolutional layer



- Forward pass

$$\boldsymbol{y}^{(l)} = \boldsymbol{y}^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}^{(l)}\right) + b^{(l)}$$

- Backward pass
  - Gradient:

$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}^{(l)}} = \boldsymbol{y}^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{\delta}^{(l)}), \quad \frac{\partial E^{(n)}}{\partial b^{(l)}} = \sum_{i,j} \delta_{ij}^{(l)}$$

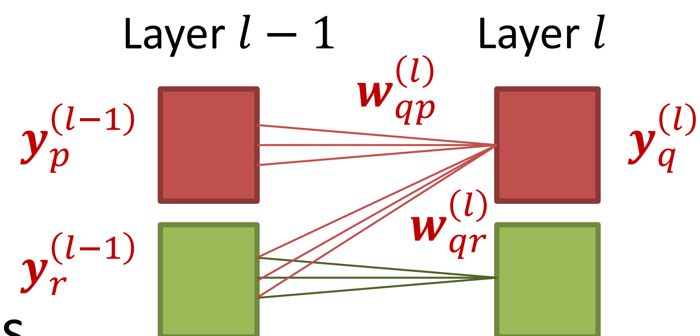
  - Local sensitivity:

$$\boldsymbol{\delta}^{(l-1)} = \boldsymbol{\delta}^{(l)} *_{\text{full}} \boldsymbol{w}^{(l)}$$

Same as 1D case

Summary for 2D convolution *with* feature combination

- Suppose that the l-th layer is a convolutional layer



(p, q, r index the feature maps)

- Forward pass

$$\boldsymbol{y}_q^{(l)} = \sum_{p \in \mathcal{M}_q} \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \operatorname{rot180}(\boldsymbol{w}_{qp}^{(l)}) + b_q^{(l)}$$

- Backward pass
  - Gradient:

$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{qp}^{(l)}} = \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \operatorname{rot180}(\boldsymbol{\delta}_q^{(l)}), \quad \frac{\partial E^{(n)}}{\partial b_q^{(l)}} = \sum_i (\boldsymbol{\delta}_q^{(l)})_{ij}$$

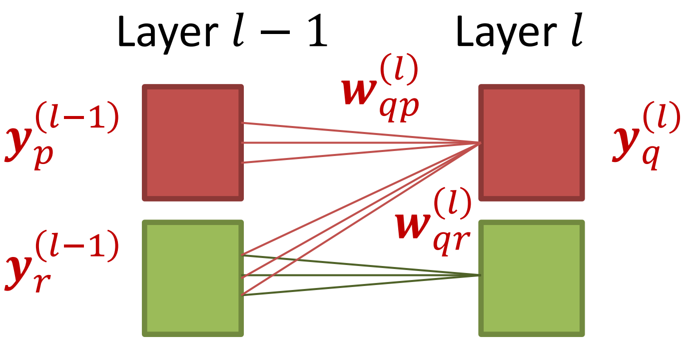
  - Local sensitivity:

$$\boldsymbol{\delta}_p^{(l-1)} = \sum_{q \in \tilde{\mathcal{M}}_p} \boldsymbol{\delta}_q^{(l)} *_{\text{full}} \boldsymbol{w}_{qp}^{(l)}$$

Same as 1D case

($\mathcal{M}_q$ and $\widetilde{\mathcal{M}}_p$ are defined before)

Replace the summation using 3D convolution



Forward pass:

$$\boldsymbol{y}_q^{(l)} = \sum_{p \in M_q} \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \text{rot}180(\boldsymbol{w}_{qp}^{(l)}) + b_q^{(l)}$$

width height

- Define 3D matrices (tensors)

$$\boldsymbol{Y}^{(l-1)} = [\boldsymbol{y}_1^{(l-1)}, \ldots, \boldsymbol{y}_p^{(l-1)}, \ldots, \boldsymbol{y}_{|\mathcal{M}_q|}^{(l-1)}] \in R^{|\mathcal{M}_q| \times M \times N}$$

$$\boldsymbol{W}_q^{(l)} = [\boldsymbol{w}_{q1}^{(l)}, \ldots, \boldsymbol{w}_{qp}^{(l)}, \ldots, \boldsymbol{w}_{q|\mathcal{M}_q|}^{(l)}] \in R^{|\mathcal{M}_q| \times K_1 \times K_2}$$

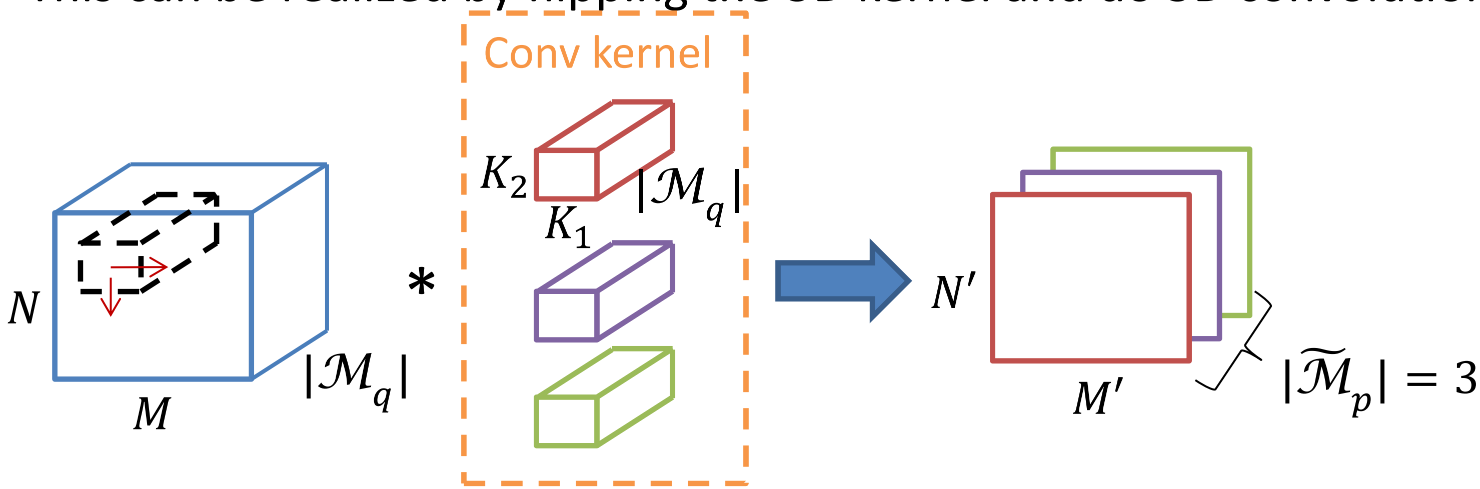
where $|\cdot|$ denotes the cardinality of a set; M, K_1: width; N, K_2: height

- The forward pass can be expressed as

$$\boldsymbol{y}_q^{(l)} = \boldsymbol{Y}^{(l-1)} *_{\text{valid}} \text{rot}180(\boldsymbol{W}_q^{(l)}) + b_q^{(l)}$$

3D convolution

- We assume the number of channels in the input is the same as that in the kernel (filter)
- Correlate a 2D feature map in the 3D input with the *corresponding* 2D section in the 3D kernel, then sum over all sections to yield one feature map
  - This can be realized by flipping the 3D kernel and do 3D convolution



The number of parameters in this layer is $|\widetilde{\mathcal{M}}_p| \times |\mathcal{M}_q| \times K_1 \times K_2$

Replace the summation using 3D convolution

Backward pass:

$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{qp}^{(l)}} = \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{\delta}_q^{(l)}), \quad \frac{\partial E^{(n)}}{\partial b_q^{(l)}} = \sum_i (\boldsymbol{\delta}_q^{(l)})_{ij}$$

- Gradient w.r.t. w

$$\begin{aligned} \frac{\partial E^{(n)}}{\partial \boldsymbol{W}_q^{(l)}} &= \boldsymbol{Y}^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{\delta}_q^{(l)}) \\ &= [\boldsymbol{y}_1^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{\delta}_q^{(l)}), \ldots, \boldsymbol{y}_{|M_q|}^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{\delta}_q^{(l)})] \end{aligned}$$

- Gradient w.r.t. b does not change
- How about the local sensitivity?

Replace the summation using 3D convolution

Backward pass:

$$\boldsymbol{\delta}_p^{(l-1)} = \sum_{q \in \tilde{M}_p} \boldsymbol{\delta}_q^{(l)} *_{\text{full}} \boldsymbol{w}_{qp}^{(l)}$$

- Define

$$\boldsymbol{\Delta}_q^{(l)} = [\boldsymbol{\delta}_{1p}^{(l)}, \ldots, \boldsymbol{\delta}_{qp}^{(l)}, \ldots, \boldsymbol{\delta}_{|\tilde{M}_p|p}^{(l)}] \in R^{|\tilde{M}_p| \times M' \times N'}$$

$$\tilde{\boldsymbol{W}}_p^{(l)} = [\boldsymbol{w}_{1p}^{(l)}, \ldots, \boldsymbol{w}_{qp}^{(l)}, \ldots, \boldsymbol{w}_{|\tilde{M}_p|p}^{(l)}] \in R^{|\tilde{M}_p| \times K_1 \times K_2}$$

width height

- Then

$$\boldsymbol{\delta}_p^{(l-1)} = \boldsymbol{\Delta}_q^{(l)} *_{\text{full}} \operatorname{flip}_0(\tilde{\boldsymbol{W}}_p^{(l)})$$

where flip_0 means flip along the first dimension

- This "full" convolution only applies in the 2nd and 3rd dimension, while in the 1st dimension (along q) the convolution type is "valid"
- $\boldsymbol{W}_q^{(l)}$ and $\tilde{\boldsymbol{W}}_p^l$ are sections of a 4D tensor $\boldsymbol{W}^{(l)} \in R^{|\tilde{M}p| \times |M_q| \times K_1 \times K_2}$

$$\boldsymbol{W}_q^{(l)} = \boldsymbol{W}_{(q,:,:,:)}^{(l)} \in R^{|M_q| \times K_1 \times K_2}, \quad \tilde{\boldsymbol{W}}_p^{(l)} = \boldsymbol{W}_{(:,p,:,:)}^{(l)} \in R^{|\tilde{M}_p| \times K_1 \times K_2}$$

Summary

- Introduction
  - Two new layers to MLP: convolution and pooling
- Convolution
  - A fast method for computing similarity
  - Akin to “simple cell”
  - With/without feature combination
  - 1D case and 2D case