Course number: 80240743

Deep Learning

Xiaolin Hu (胡晓林) & Jun Zhu (朱军)
Dept. of Computer Science and Technology
Tsinghua University

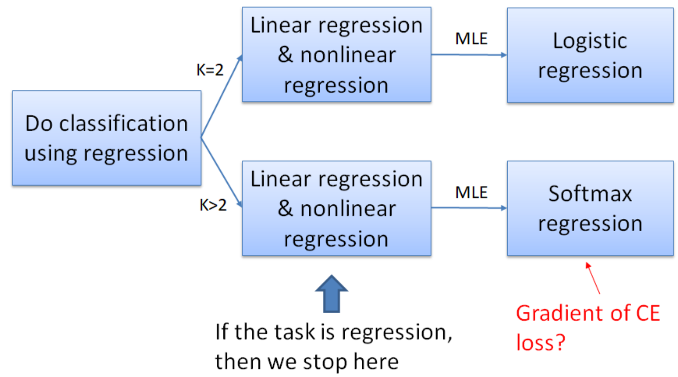
Last lecture review

1. Regression and classification (cont'd)

CE loss

$$E^{(n)} = -\left(\boldsymbol{t}^{(n)}\right)^{\top} \ln \boldsymbol{h}^{(n)}$$

$$\nabla_{\boldsymbol{\theta}} E = \left(\boldsymbol{f}\left(\boldsymbol{x}^{(n)}\right) - \boldsymbol{t}^{(n)}\right)\left(\boldsymbol{x}^{(n)}\right)^{\top}$$



2. Multi-layer perceptron
   - Forward calculation: for $l = 1, \ldots, L$

$$\boldsymbol{u}^{(l)} = \boldsymbol{W}^{(l)} \boldsymbol{y}^{(l-1)} + \boldsymbol{b}^{(l)} \text{ and } \boldsymbol{y}^{(l)} = \boldsymbol{f}(\boldsymbol{u}^{(l)})$$

   - Backward calculation:

For $l = L$: $\boldsymbol{\delta}^{(L)} = (\boldsymbol{y}^{(L)} - \boldsymbol{t}) \odot \boldsymbol{f}'(\boldsymbol{u}^{(L)})$

or $\boldsymbol{\delta}^{(L)} = \boldsymbol{y}^{(L)} - \boldsymbol{t}$

For $l = L-1, \ldots, 1$

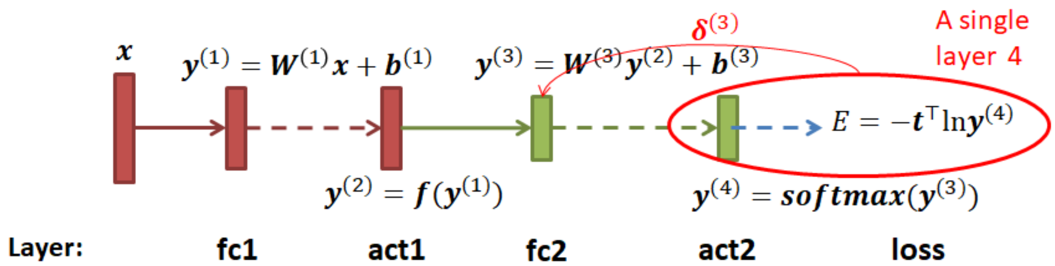
$$\boldsymbol{\delta}^{(l)} = (\boldsymbol{W}^{(l+1)})^{\top} \boldsymbol{\delta}^{(l+1)} \odot \boldsymbol{f}'(\boldsymbol{u}^{(l)})$$

$$\frac{\partial E^{(n)}}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{\delta}^{(l)} (\boldsymbol{f}(\boldsymbol{u}^{(l-1)}))^{\top},$$

$$\frac{\partial E^{(n)}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l)}$$

Last lecture review

3. Layer decomposition



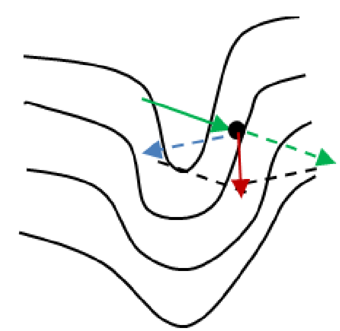
FC layer | sigmoid layer

ReLU layer | loss layer

4. Training techniques-I

Weight initialization | learning rate

order of training samples | momentum

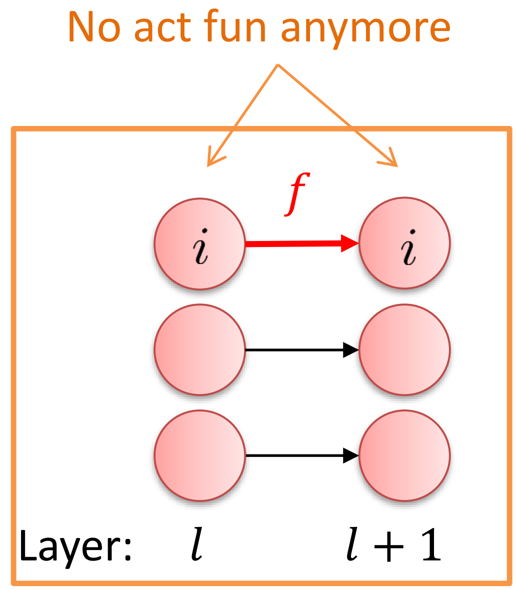


Exercise

- Derive the local sensitivity $\boldsymbol{\delta}$ and gradient $\partial E/\partial \boldsymbol{W}$ and $\partial E/\partial \boldsymbol{b}$ where applicable for
  - Euclidean loss layer: $E^{(n)} = \frac{1}{2}\left|\left|\boldsymbol{y}^{(L)} - \boldsymbol{t}\right|\right|^2$
    - Note that here we calculate $\boldsymbol{\delta}^{(L)} = \partial E^{(n)}/\partial \boldsymbol{y}^{(L)}$
  - Softmax-cross-entropy error layer $E^{(n)} = -\sum_{k=1}^{K} t_k \ln f\left(y_k^{(L)}\right)$
    - Note that here we calculate $\boldsymbol{\delta}^{(L-1)} = \partial E^{(n)}/\partial \boldsymbol{y}^{(L-1)}$
  - Fully connected layer:$y_j^{(l)} = \sum_i w_{ji}^{(l)} y_i^{(l-1)} + b_j^{(l)}$
  - Sigmoid layer: $y_j^{(l)} = f\left(y_j^{(l-1)}\right)$, where f is a sigmoid function
  - ReLU layer: $y_j^{(l)} = f\left(y_j^{(l-1)}\right)$, where f is a ReLU function

These layers are shown in the previous slides

Hint



Note that this layer doesn't have w and b

- Suppose the $(l+1)$-th layer is a sigmoid activation layer:

$$y_i^{(l+1)} = f\left(y_i^{(l)}\right)$$

where f is the sigmoid function

- Neuron i in the l-th layer only affects neuron i in the $(l+1)$-th layer, therefore

$$\delta_i^{(l)} = \frac{\partial E^{(n)}}{\partial u_i^{(l)}} = \frac{\partial E^{(n)}}{\partial y_i^{(l)}} = \frac{\partial E^{(n)}}{\partial y_i^{(l+1)}} \frac{\partial y_i^{(l+1)}}{\partial y_i^{(l)}} = \delta_i^{(l+1)} f'(y_i^{(l)})$$

- Similarly, you can derive the results for other layers.

Answers

$\boldsymbol{u}^{(l)}$ and $\boldsymbol{y}^{(l)}$ are identical in every layer l

1. Euclidean loss layer $\boldsymbol{\delta}^{(L)} = \boldsymbol{y}^{(L)} - \boldsymbol{t}$
2. Softmax-cross-entropy error layer $\boldsymbol{\delta}^{(L-1)} = \boldsymbol{y}^{(L)} - \boldsymbol{t}$

   must be softmax output
3. The l-th layer is an FC layer

$$\frac{\partial E^{(n)}}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{\delta}^{(l)} (\boldsymbol{y}^{(l-1)})^\top, \quad \frac{\partial E^{(n)}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l)}, \quad \boldsymbol{\delta}^{(l-1)} = (\boldsymbol{W}^{(l)})^\top \boldsymbol{\delta}^{(l)}$$

4. The l-th layer is a sigmoid layer

$$\boldsymbol{\delta}^{(l-1)} = \boldsymbol{\delta}^{(l)} \odot \boldsymbol{f}'(\boldsymbol{y}^{(l-1)}) \text{ where } f'(x) = f(x)(1 - f(x))$$

5. The l-th layer is a relu layer

$$\boldsymbol{\delta}^{(l-1)} = \boldsymbol{\delta}^{(l)} \odot \boldsymbol{f}'(\boldsymbol{y}^{(l-1)}) \text{ where } f'(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{else.} \end{cases}$$

BP in vector-matrix form

Feedforward: For MSE, create $L+1$ layers; for CE, create L layers $\delta_i^{(l)} \triangleq \dfrac{\partial E^{(n)}}{\partial y_i^{(l)}}$

- The last layer
  - For MSE layer, $l = L+1$, calculate $\boldsymbol{\delta}^{(L)} = \boldsymbol{y}^{(L)} - \boldsymbol{t}$
  - For Softmax-CE layer, $l = L$, calculate $\boldsymbol{\delta}^{(L-1)} = \boldsymbol{y}^{(L)} - \boldsymbol{t}$
- From the last layer to the first layer
  - The l-th layer is a FC layer

$$\frac{\partial E^{(n)}}{\partial \boldsymbol{W}^{(l)}} = \boldsymbol{\delta}^{(l)} (\boldsymbol{y}^{(l-1)})^\top, \quad \frac{\partial E^{(n)}}{\partial \boldsymbol{b}^{(l)}} = \boldsymbol{\delta}^{(l)}, \quad \boldsymbol{\delta}^{(l-1)} = (\boldsymbol{W}^{(l)})^\top \boldsymbol{\delta}^{(l)}$$

  - The l-th layer is an activation layer

$$\boldsymbol{\delta}^{(l-1)} = \boldsymbol{\delta}^{(l)} \odot \boldsymbol{f}'(\boldsymbol{y}^{(l-1)})$$

where $f(\cdot)$ is the act fun

for each sample n

- Update weights

avg over n

$$\boldsymbol{W}^{(l)} = \boldsymbol{W}^{(l)} - \frac{\alpha}{N} \sum_n \frac{\partial E^{(n)}}{\partial \boldsymbol{W}^{(l)}} - \alpha\lambda \boldsymbol{W}^{(l)}, \quad \boldsymbol{b}^{(l)} = \boldsymbol{b}^{(l)} - \frac{\alpha}{N} \sum_n \frac{\partial E^{(n)}}{\partial \boldsymbol{b}^{(l)}}$$

Lecture 4: Convolutional Neural Networks-I

Xiaolin Hu
Dept. of Computer Science and Technology
Tsinghua University

Outline

1. Introduction
2. Convolution
   - Forward pass
   - Backward pass
3. Summary

Open Question Points: 10

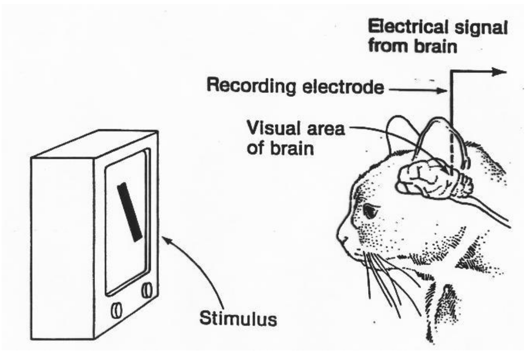
Setting

Are there any shortcomings of MLP for processing images?

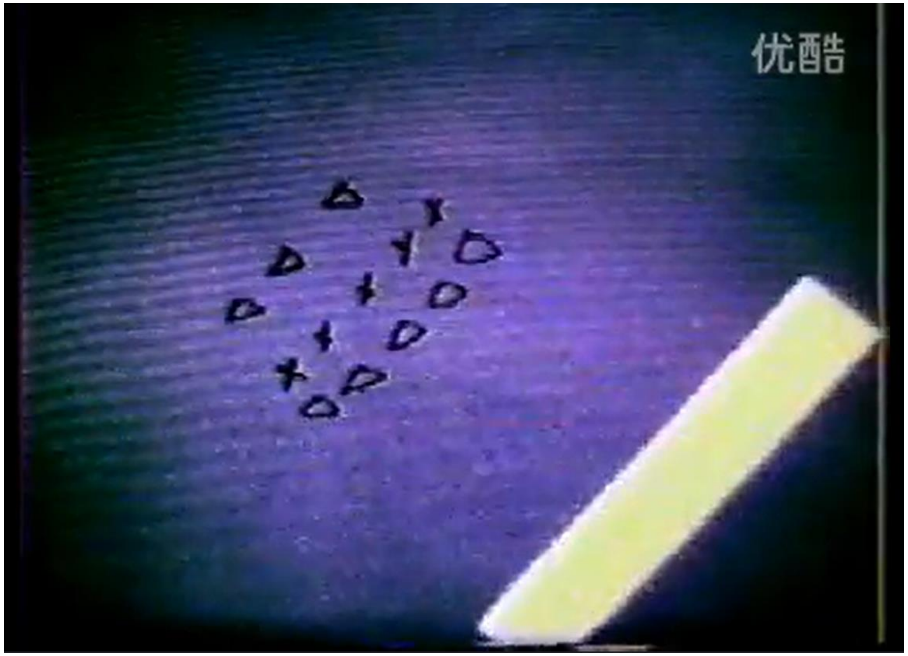
Open Question is only supported on Version 2.0 or newer.

Answer

Hubel and Wiesel’s experiment



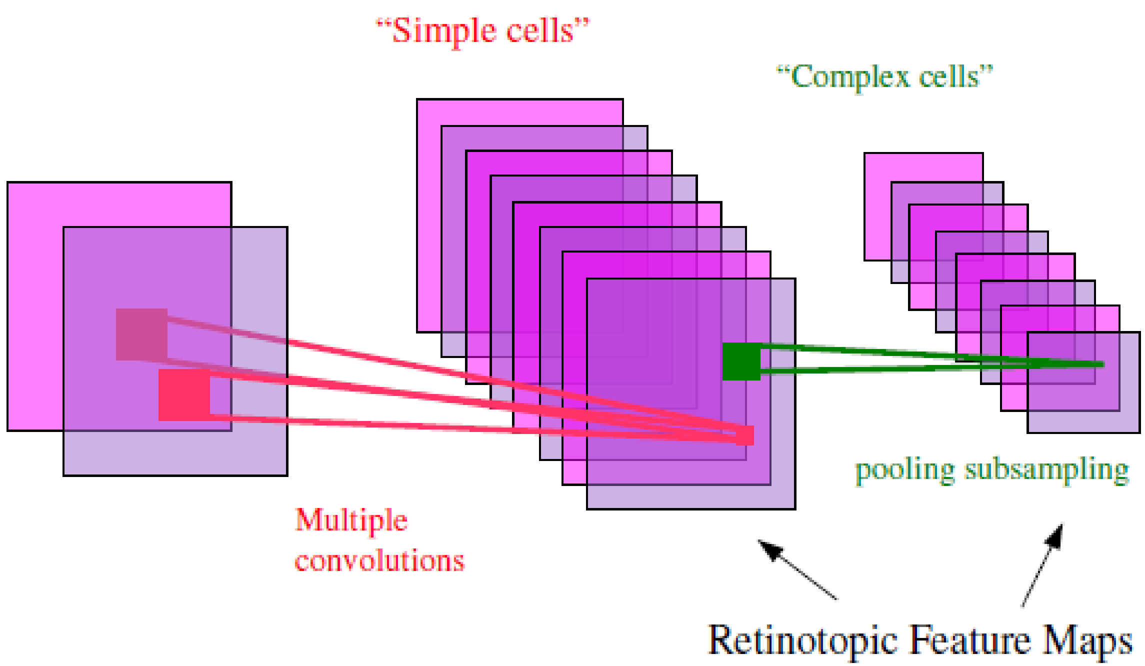
Novel Prize 1981



Youku link

Local detectors and shift invariance in the cortex

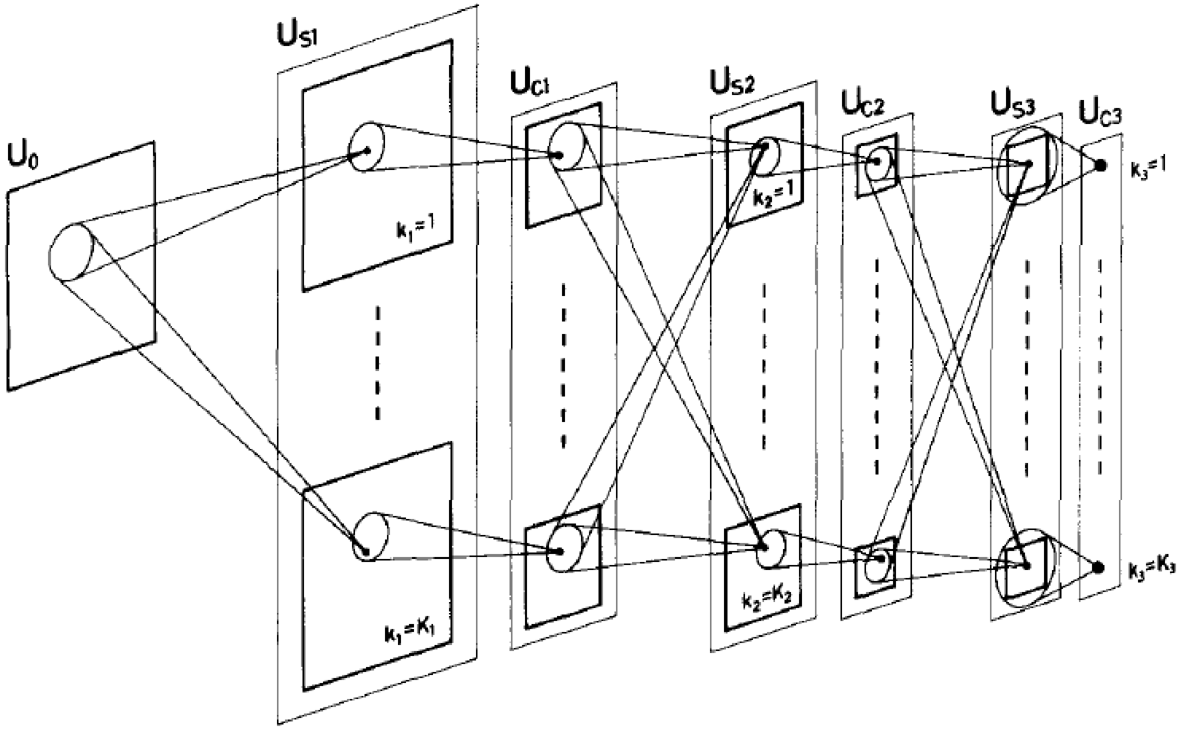
- (Hubel & Wiesel 1962)
  - Simple cells detect local features
  - complex cells "pool" the outputs of simple cells within a retinotopic neighborhood



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 1988-1989]
  - HMAX [Poggio 2002-2006]
  - fragment hierarchy [Ullman 2002-2006]
  - HMAX [Lowe 2006]

Neocognitron



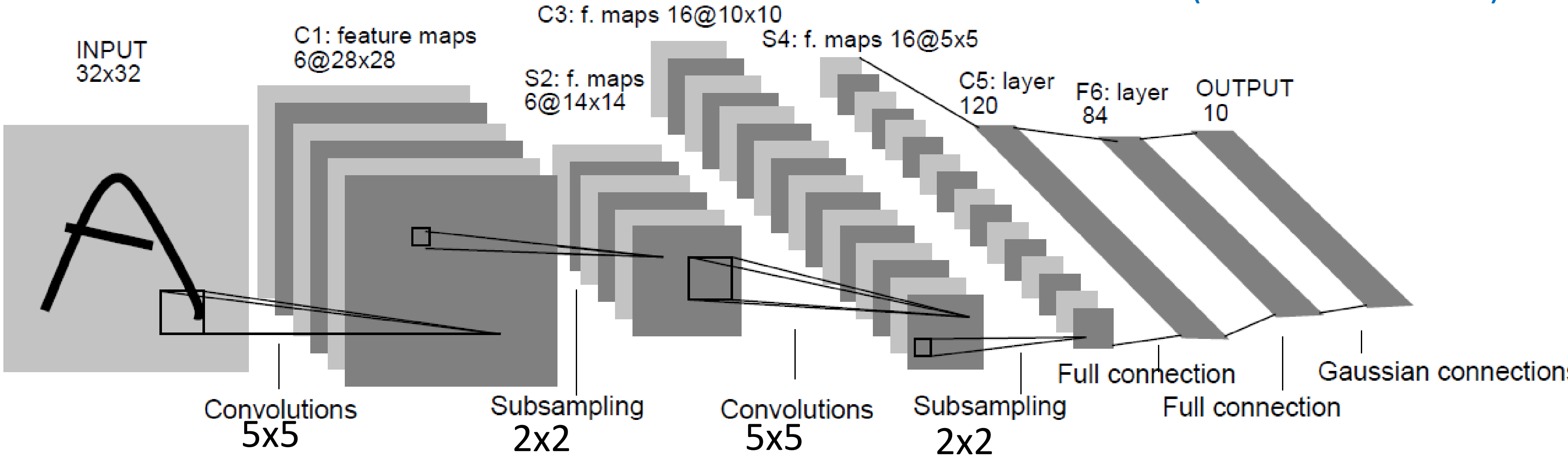
Fukushima, Biol. Cybernetics, 1980

Franklin Institute honors pioneers in the study of forest fires, longevity, eyesight

https://www.inquirer.com/science/franklin-institute-science-awards-climate-change-20200127.html

Convolutional neural network (CNN)

LeNet5 (LeCun et al. 1998)



- Convolution
  - Local connections and weight sharing
- Subsampling (pooling)
- Contribution: apply BP algorithm

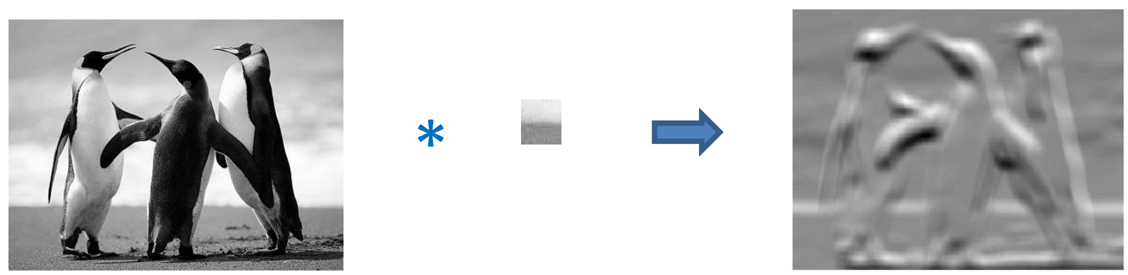
Yann LeCun (USA)
Turing award 2018

LeCun, B. Boser et al., Backpropagation Applied to Handwritten Zip Code Recognition, *Neural Computation* (1989)

Two new layers

input kernel feature map

Convolution



Pooling

poolingsize=2x2

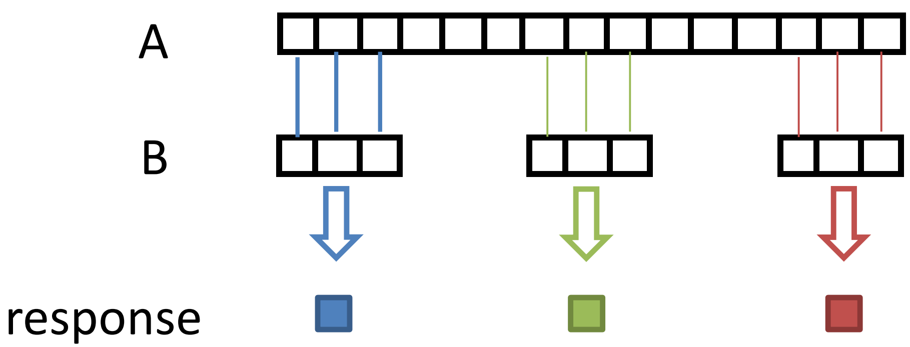
- Convolutional layer and pooling layer
  - Define two additional layers with forward computation and backward computation

Outline

1. Introduction
2. Convolution
   – Forward pass
   – Backward pass
3. Summary

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:

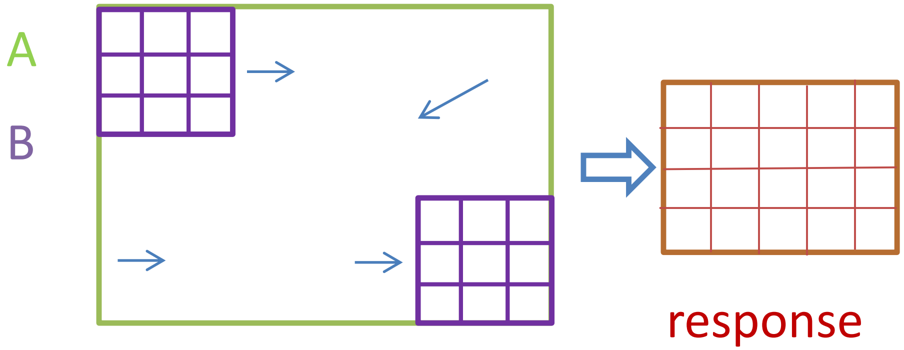
$$s \equiv \cos\theta = \frac{x^\top y}{||x||\,||y||}$$

$$= \sum_i x_i y_i$$

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

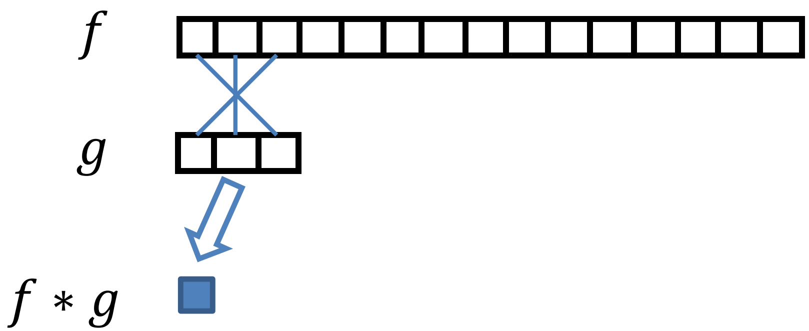
1D convolution

- 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

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

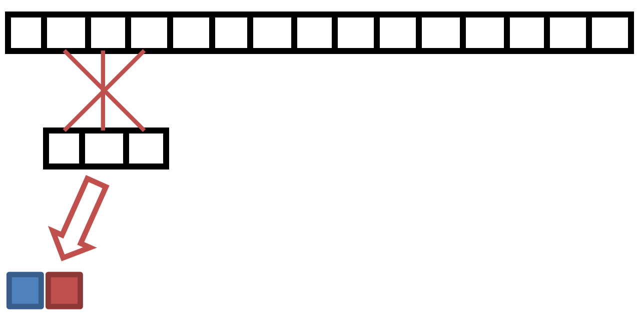
- Discrete convolution (for finite length sequences)

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

f

g

$f * g$



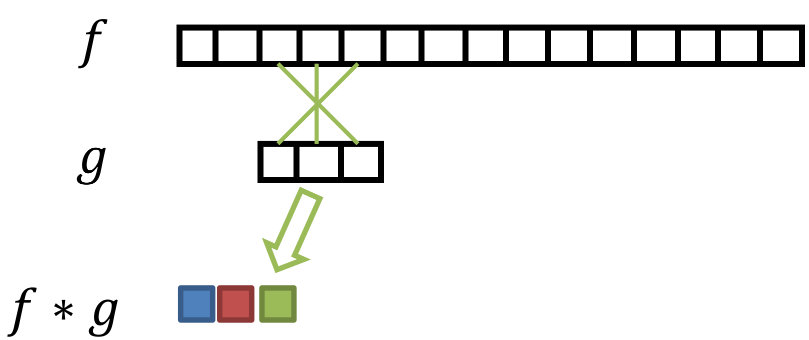
1D convolution

- 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)

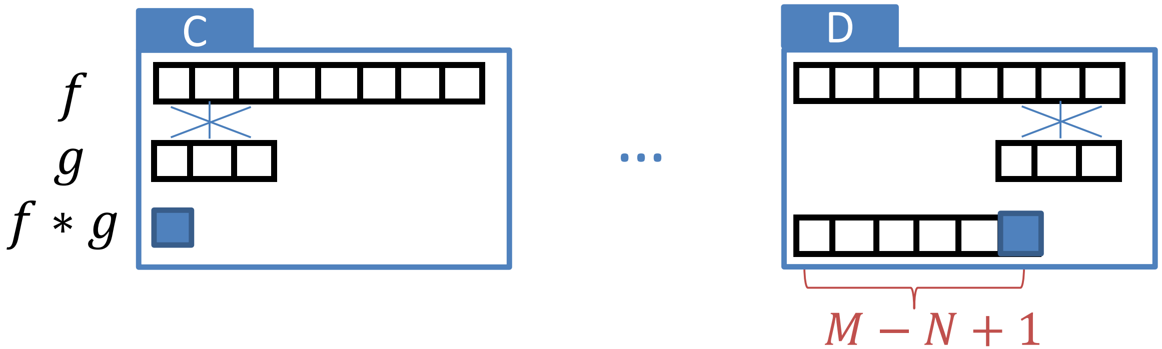
$$(f * g)[m] \triangleq \sum_{n=1}^{N} f[m-n]g[n]$$



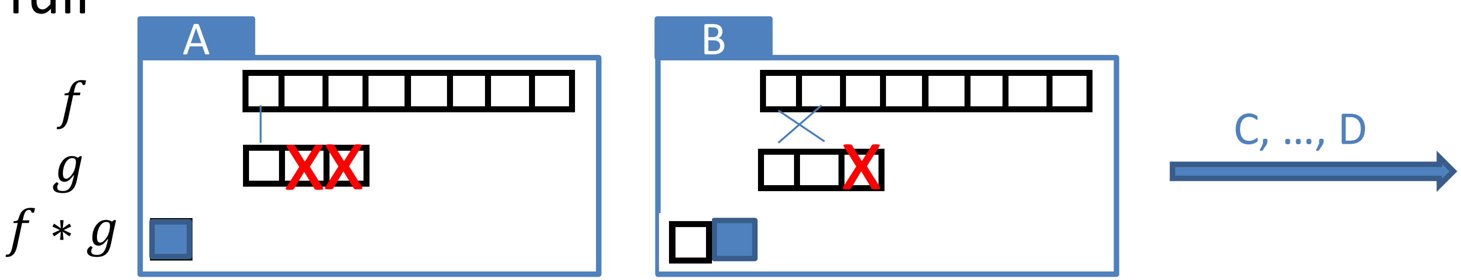
Three shapes of convolution

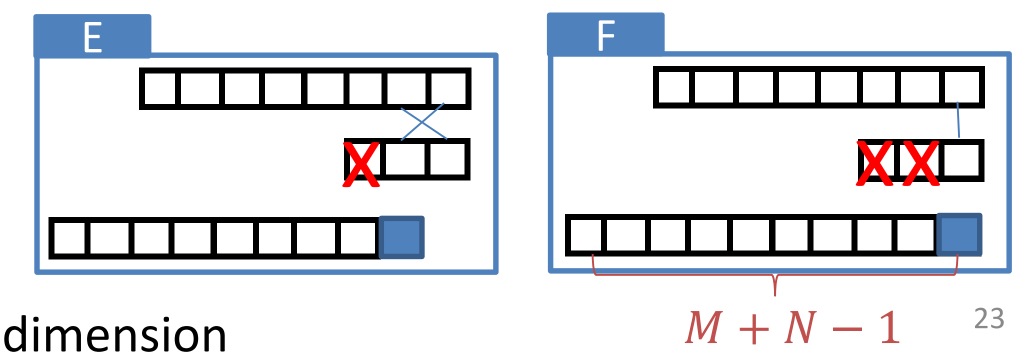
Length of f: M, length of g: N, where $M \geq N$

- 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

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

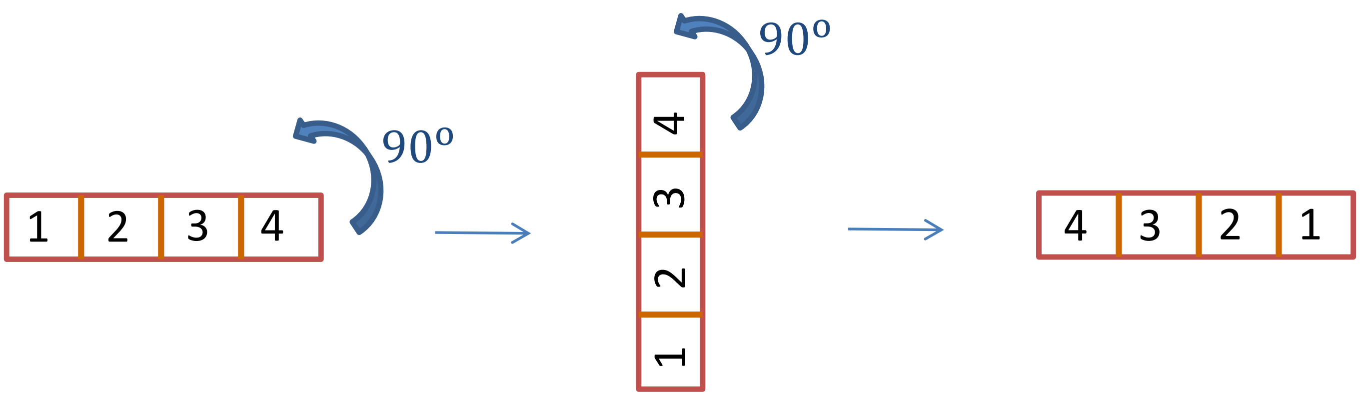
- Python commands

```
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)

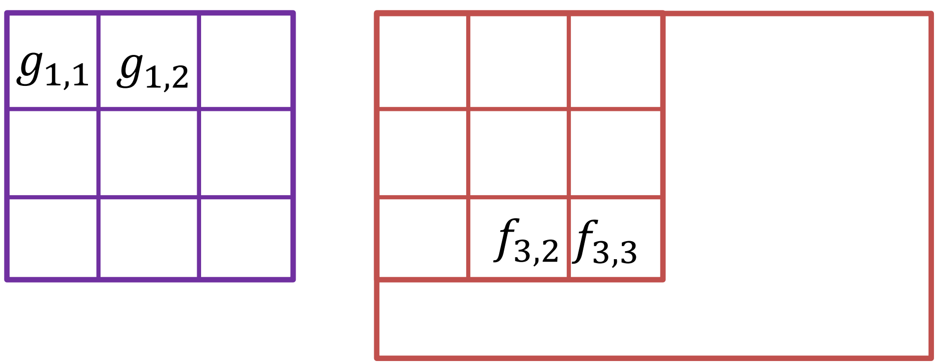


It's equivalent to *flip the vector along the axis 0*

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

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

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

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \xrightarrow{90^{o}} \begin{bmatrix} 3 & 6 & 9 \\ 2 & 5 & 8 \\ 1 & 4 & 7 \end{bmatrix} \xrightarrow{90^{o}} \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

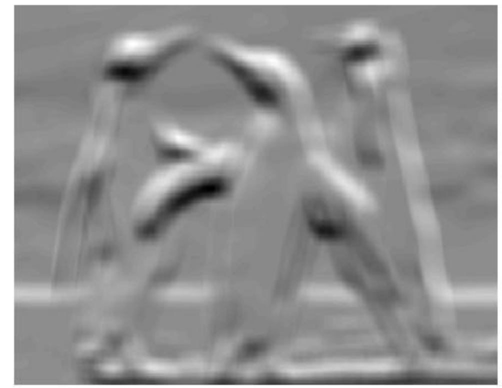
It's equivalent to *flip the matrix along the axes 0 then 1*

Example

feature map

figure

Filter
(mean subtracted)



corr

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

Multiple Choice(multiple) Points: 1

Setting

Why do we use convolution?

A Look for locally matched patterns

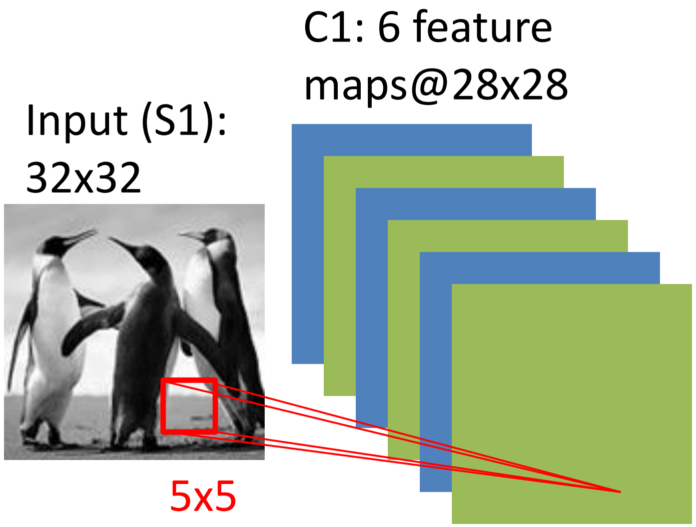
B Look for globally matched patterns

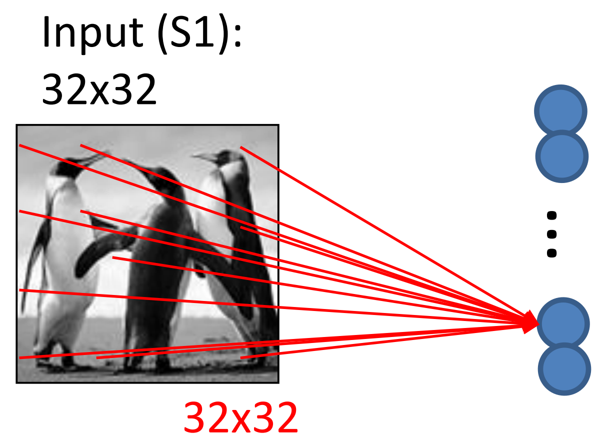
C Increase the number of parameters

D Simulate the functions of simple cells in the brain

Submit

Convolution saves the number of parameters



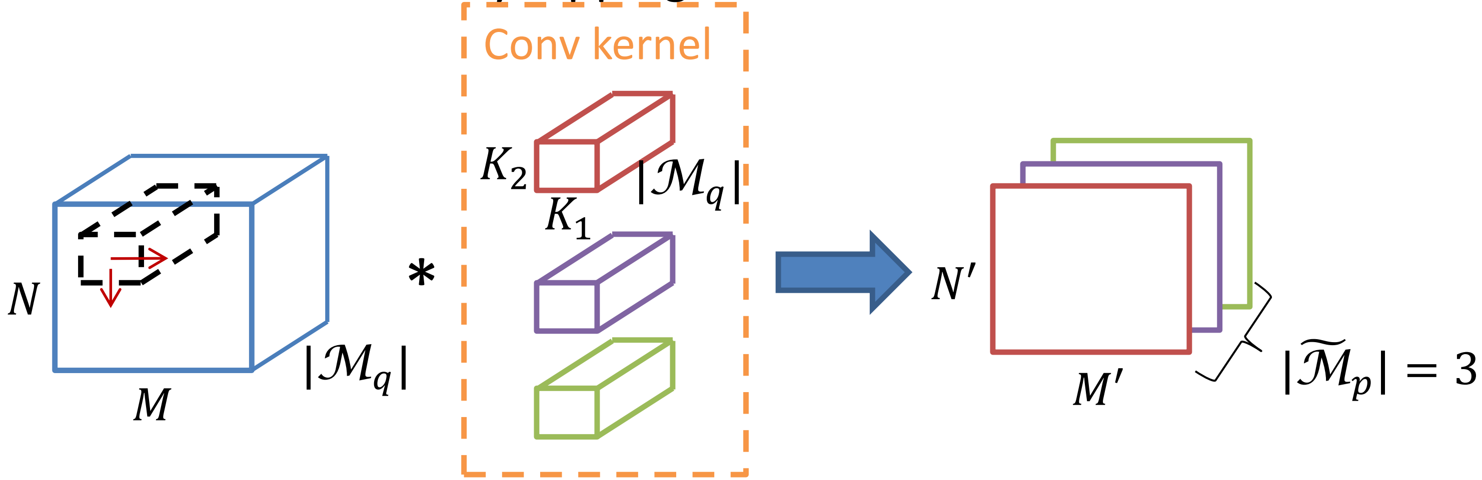


- One feature map has 25 parameters
- The total number of parameters:
  - 25x the number of feature maps

- One neuron has 1024 parameters
- The total number of parameters:
  - 1024x the number of neurons

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

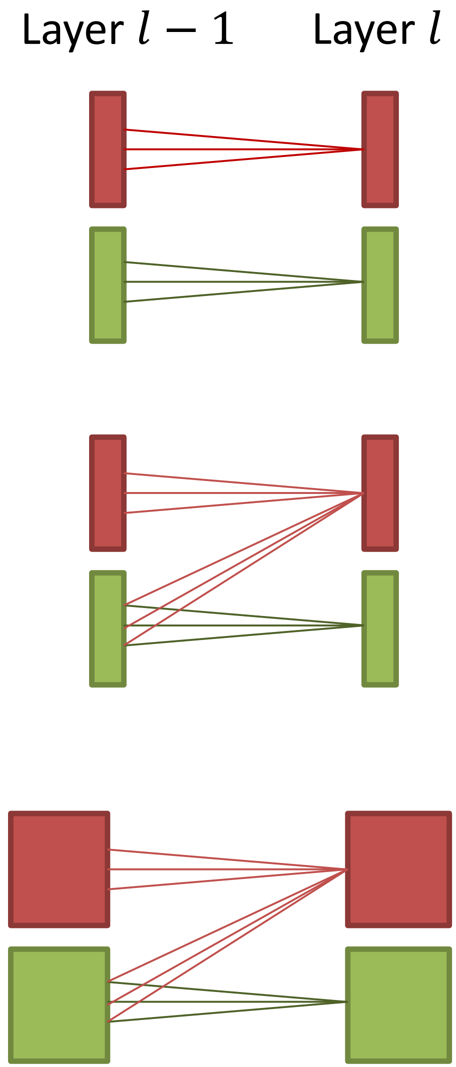


The number of parameters in this layer is $|\widetilde{\mathcal{M}}_p| \times |\mathcal{M}_q| \times K_1 \times K_2$

Outline

1. Introduction
2. Convolution
   – Forward pass
   – Backward pass
3. Summary

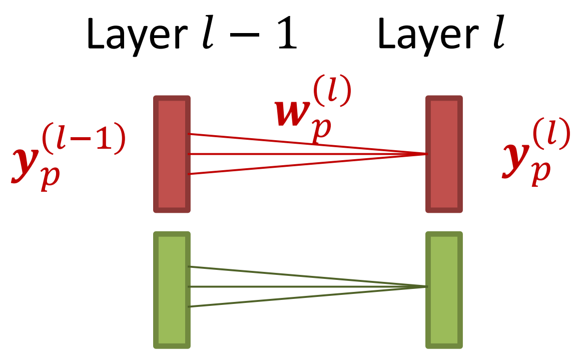
Derive BP algorithm in different cases



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

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

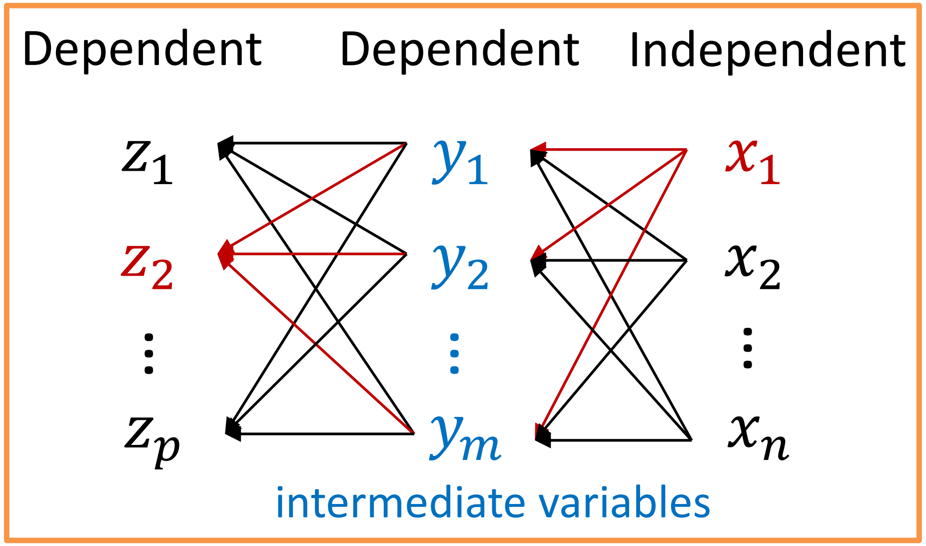
A vector

A scalar

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

Recap: Derivative of two-step composition

- Independent variables $x_1, x_2, \dots, x_n$
- Each y_i is a function of $x_1, x_2, \dots, x_n$
- Each z_i is a function of $y_1, y_2, \dots, y_m$



What's partial derivative of z_i w.r.t. x_j?

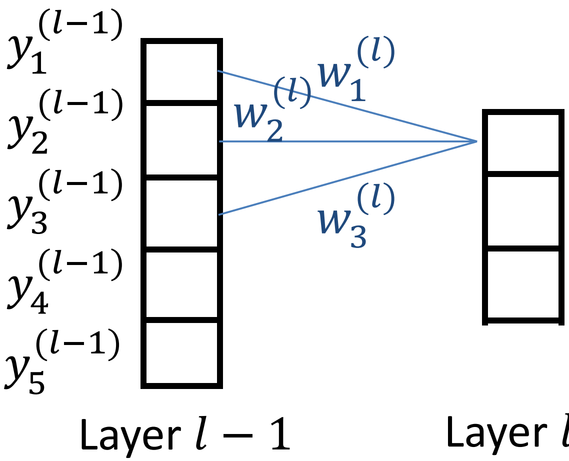
$$\frac{\partial z_i}{\partial x_j} = \sum_{k=1}^{m} \frac{\partial z_i}{\partial y_k} \frac{\partial y_k}{\partial x_j}$$

Sum over the intermediate variables

for any $i \in \{1,2, \dots, p\}$ and $j \in \{1,2, \dots, n\}$

Gradient calculation in an 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 $w_i^{(l)}$ and $E^{(n)}$

- Partial derivative w.r.t. $\boldsymbol{w}^{(l)}$: scalar form

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

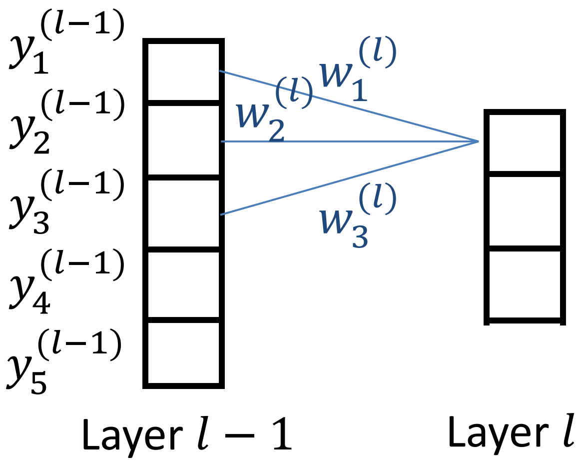
$$\frac{\partial E^{(n)}}{\partial w_2^{(l)}} = \sum_{i=1}^{3} \frac{\partial E^{(n)}}{\partial y_i^{(l)}} \frac{\partial y_i^{(l)}}{\partial w_2^{(l)}} = \delta_1^{(l)} y_2^{(l-1)} + \delta_2^{(l)} y_3^{(l-1)} + \delta_3^{(l)} y_4^{(l-1)}$$

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

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

Gradient calculation in general

Consider one single feature map in layer l



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

- Partial derivative w.r.t. $\boldsymbol{w}^{(l)}$: vector form

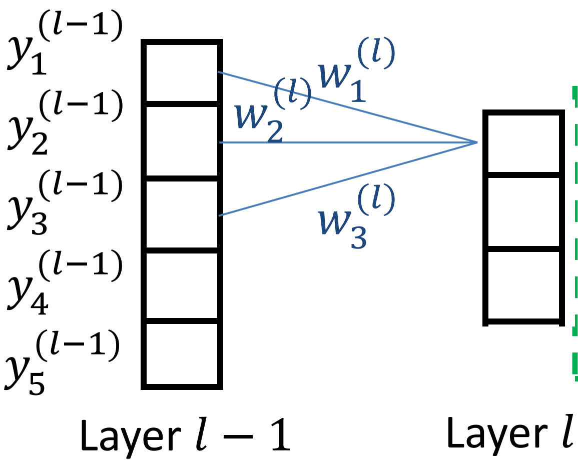
$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}^{(l)}} = \boldsymbol{y}^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{\delta}^{(l)})$$

- Partial derivative w.r.t. $b^{(l)}$

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

Local sensitivity in the example

Consider one single feature map in layer l



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

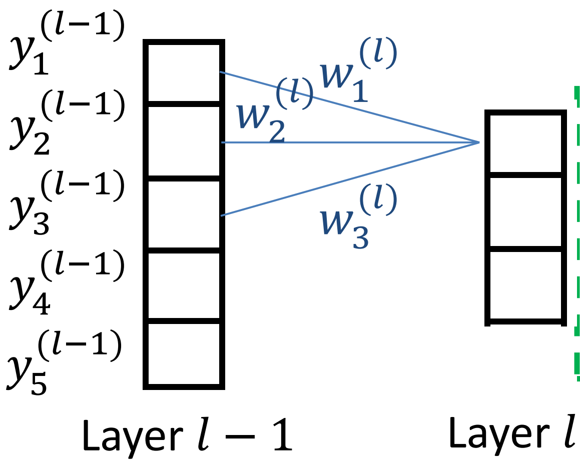
Intermediate variable between $y_i^{(l-1)}$ and $E^{(n)}$

- $y_1^{(l-1)}$ 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



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

Intermediate variable between $y_i^{(l-1)}$ 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 $\delta_3^{(l-1)}$, $\delta_4^{(l-1)}$ and $\delta_5^{(l-1)}$

Local sensitivity in general

- Local sensitivity in the vector form

$$\boldsymbol{\delta}^{(l-1)} \triangleq \frac{\partial E^{(n)}}{\partial \boldsymbol{y}^{l-1}} = \begin{pmatrix} \delta_1^{(l)} w_1^{(l)} \\ \delta_1^{(l)} w_2^{(l)} + \delta_2^{(l)} w_1^{(l)} \\ \delta_1^{(l)} w_3^{(l)} + \delta_2^{(l)} w_2^{(l)} + \delta_3^{(l)} w_1^{(l)} \\ \delta_2^{(l)} w_3^{(l)} + \delta_3^{(l)} w_2^{(l)} \\ \delta_3^{(l)} w_3^{(l)} \end{pmatrix}$$

Multiple Choice(single) Points: 1

Setting

$$\boldsymbol{\delta}^{(l-1)} \triangleq \frac{\partial E^{(n)}}{\partial \boldsymbol{y}^{l-1}} = \begin{pmatrix} \delta_1^{(l)} w_1^{(l)} \\ \delta_1^{(l)} w_2^{(l)} + \delta_2^{(l)} w_1^{(l)} \\ \delta_1^{(l)} w_3^{(l)} + \delta_2^{(l)} w_2^{(l)} + \delta_3^{(l)} w_1^{(l)} \\ \delta_2^{(l)} w_3^{(l)} + \delta_3^{(l)} w_2^{(l)} \\ \delta_3^{(l)} w_3^{(l)} \end{pmatrix} = ?$$

A $\boldsymbol{\delta}^{(l)} *_{\text{valid}} \boldsymbol{w}^{(l)}$

B $\boldsymbol{\delta}^{(l)} *_{\text{full}} \boldsymbol{w}^{(l)}$

C $\boldsymbol{\delta}^{(l)} *_{\text{full}} \text{rot180}\left(\boldsymbol{w}^{(l)}\right)$

Submit

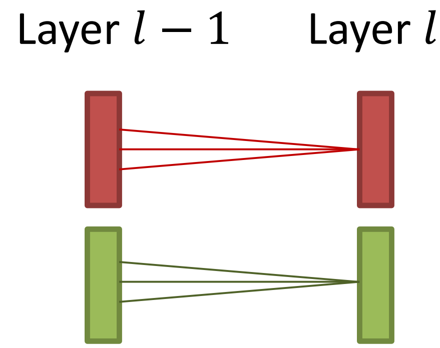
Local sensitivity in general

- Local sensitivity in the vector form

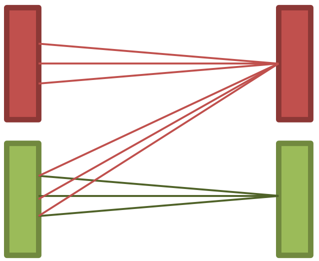
$$\boldsymbol{\delta}^{(l-1)} \triangleq \frac{\partial E^{(n)}}{\partial \boldsymbol{y}^{l-1}} = \underbrace{\begin{pmatrix} \delta_1^{(l)} w_1^{(l)} \\ \delta_1^{(l)} w_2^{(l)} + \delta_2^{(l)} w_1^{(l)} \\ \delta_1^{(l)} w_3^{(l)} + \delta_2^{(l)} w_2^{(l)} + \delta_3^{(l)} w_1^{(l)} \\ \delta_2^{(l)}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

Derive BP algorithm in different cases

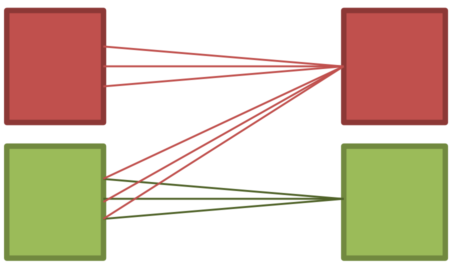
1. The 1D convolution case without feature combination



2. The 1D convolution case with feature combination

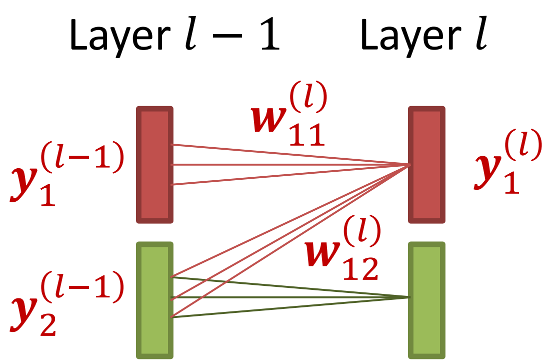


3. The 2D convolution case



Case 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 $l-1$

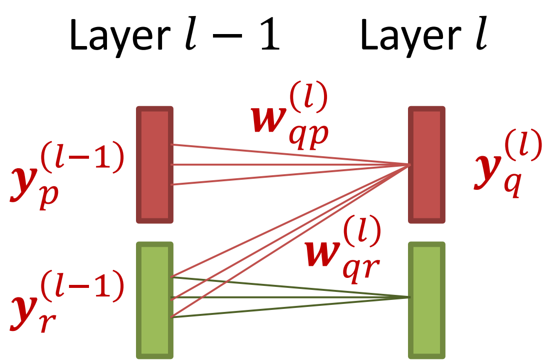
$$\boldsymbol{y}_1^{(l)} = \boldsymbol{y}_1^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_{11}^{(l)}\right) + \boldsymbol{y}_2^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{w}_{12}^{(l)}\right) + b_1^{(l)}$$

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

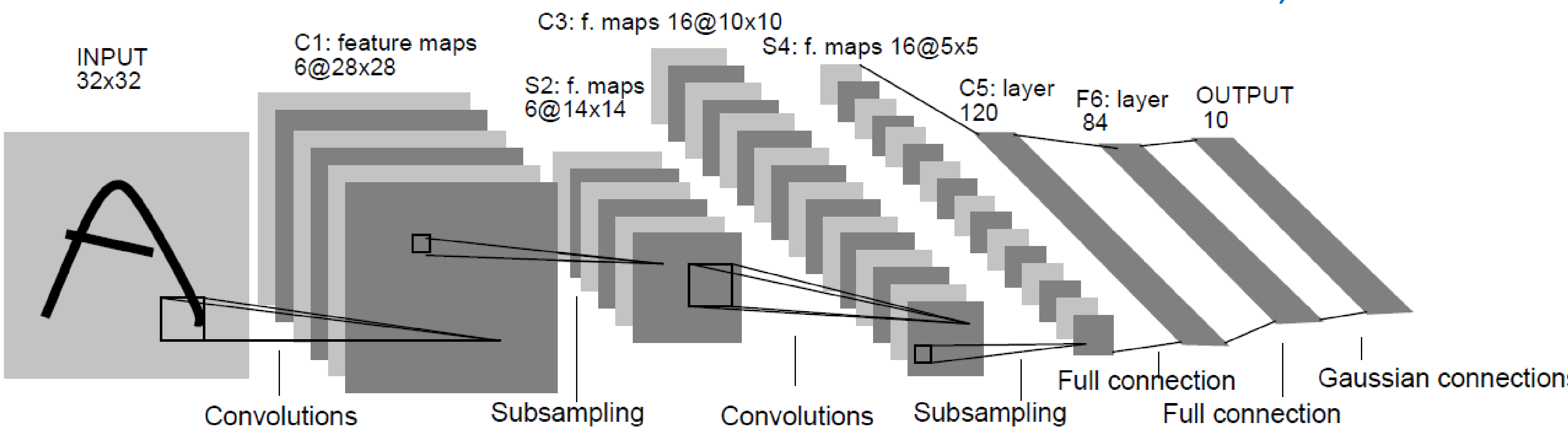
$$\boldsymbol{y}_q^{(l)} = \sum_{p \in M_q} \boldsymbol{y}_p^{(l-1)} *_{\text{valid}} \operatorname{rot180}\left(\boldsymbol{w}_{qp}^{(l)}\right) + \underbrace{b_q^{(l)}}_{\text{A scalar}}$$

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 is not the case

LeNet5, 1998

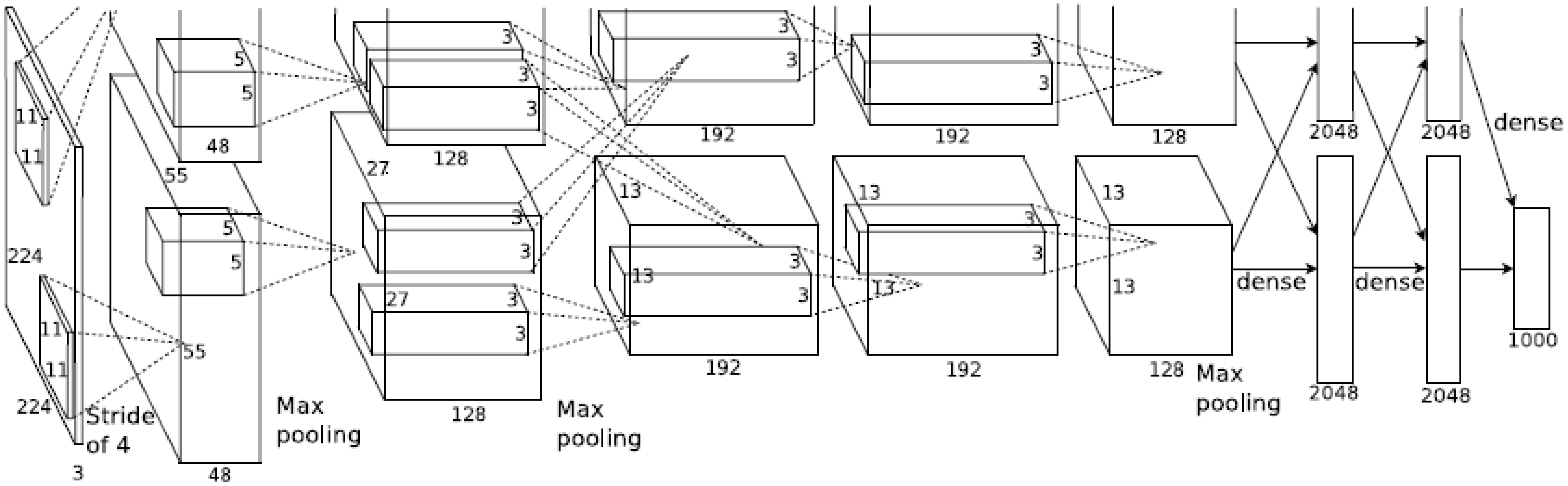


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

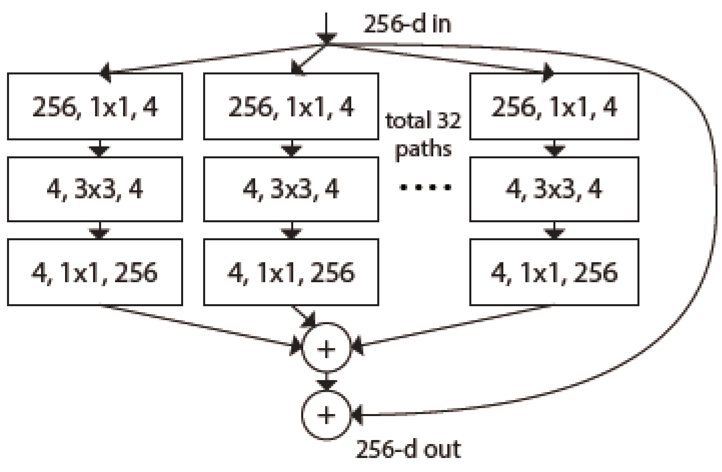
Each column indicates which feature map in S2 are combined to produce a particular feature map of C3

Feature map selection

AlexNet, 2012

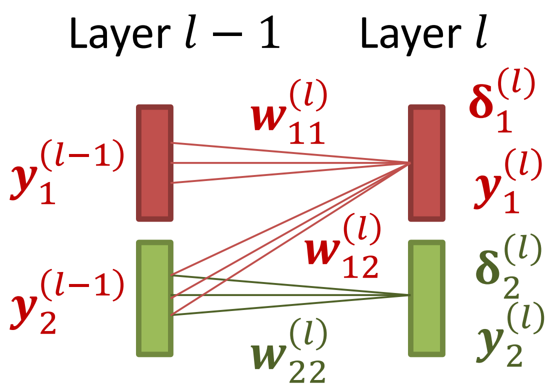


ResNeXt, 2017



Gradient calculation in the example

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



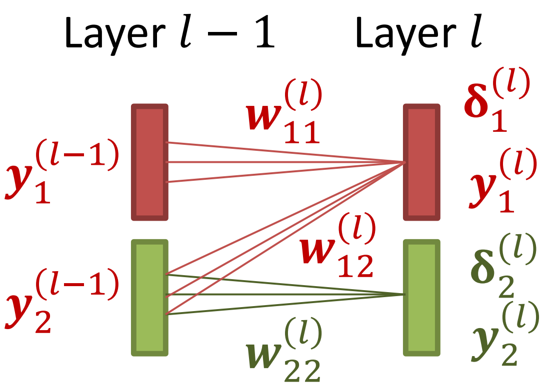
- Are these eqns correct?

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

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

Multiple Choice(multiple) Points: 1 Setting

Which are correct?



A $$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{11}^{(l)}} = \boldsymbol{y}_1^{(l-1)} *_{\text{valid}} \text{rot}180(\boldsymbol{\delta}_1^{(l)}),$$

B $$\frac{\partial E^{(n)}}{\partial b_1^{(l)}} = \sum_i (\boldsymbol{\delta}_1^{(l)})_i,$$

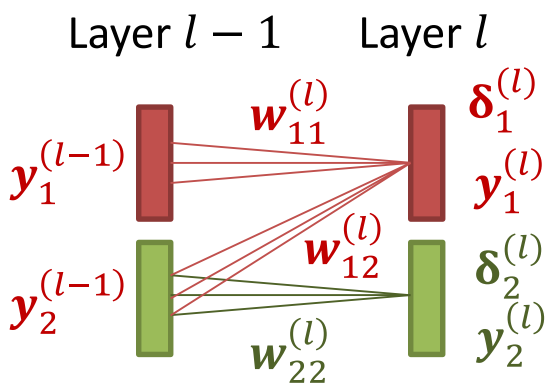
C $$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{22}^{(l)}} = \boldsymbol{y}_2^{(l-1)} *_{\text{valid}} \text{rot}180(\boldsymbol{\delta}_2^{(l)}),$$

D $$\frac{\partial E^{(n)}}{\partial b_2^{(l)}} = \sum_i (\boldsymbol{\delta}_2^{(l)})_i.$$

Submit

Gradient calculation in the example

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



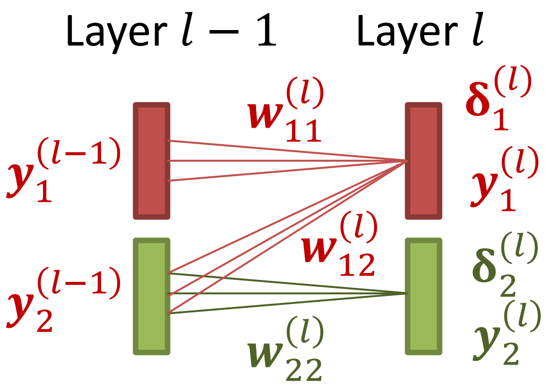
- How about $\partial E^{(n)}/\partial \boldsymbol{w}_{12}^{(l)}$?

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

- How about the corresponding bias term?
  - $\partial E^{(n)}/b_1^{(l)}$ has been calculated in the previous slide, which is shared by $\boldsymbol{w}_{11}^{(l)}$ and $\boldsymbol{w}_{12}^{(l)}$

Gradient calculation in general

- In layer l, calculate

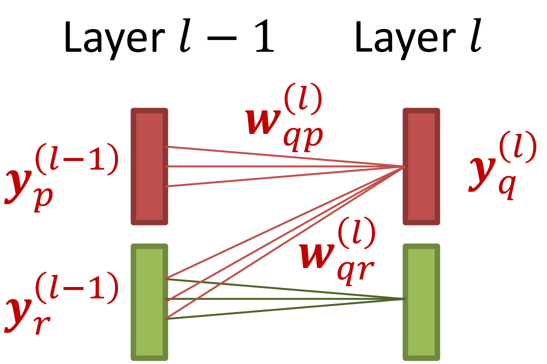


$$\frac{\partial E^{(n)}}{\partial \boldsymbol{w}_{11}^{(l)}} = \boldsymbol{y}_1^{(l-1)} *_{\text{valid}} \operatorname{rot180}(\boldsymbol{\delta}_1^{(l)}), \quad \frac{\partial E^{(n)}}{\partial b_1^{(l)}} = \sum_i (\boldsymbol{\delta}_1^{(l)})_i,$$

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

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

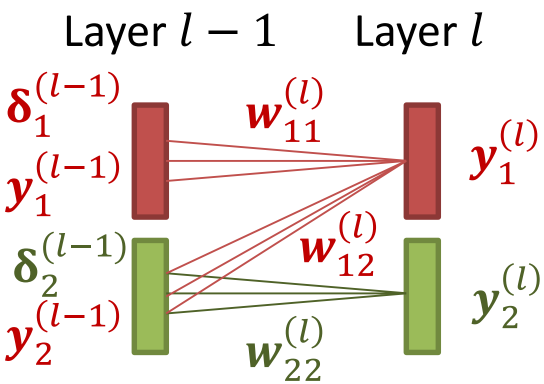
- In general



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

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,

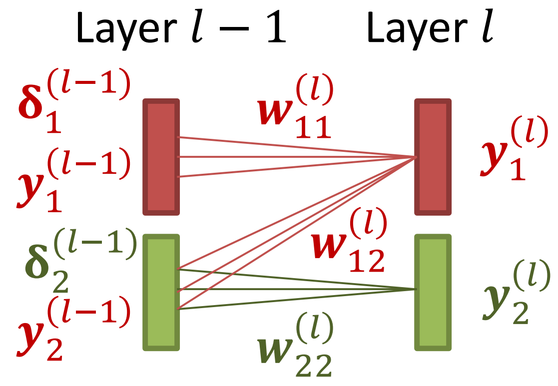
$$\boldsymbol{\delta}_1^{(l-1)} = \boldsymbol{\delta}_1^{(l)} *_{\text{full}} \boldsymbol{w}_{11}^{(l)} \quad ?$$

Multiple Choice(single) Points: 1

Setting

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,

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



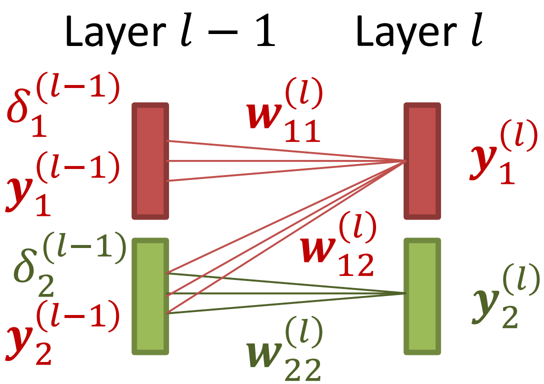
A Yes

B No

Submit

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,

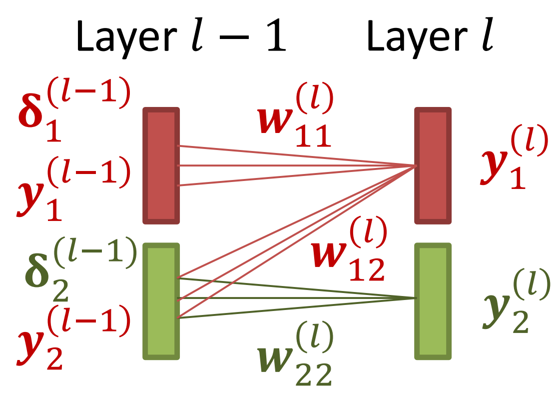
$$\boldsymbol{\delta}_2^{(l-1)} = \boldsymbol{\delta}_2^{(l)} *_{\text{full}} \boldsymbol{w}_{22}^{(l)} \quad ?$$

Multiple Choice(single) Points: 1

Setting

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,

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



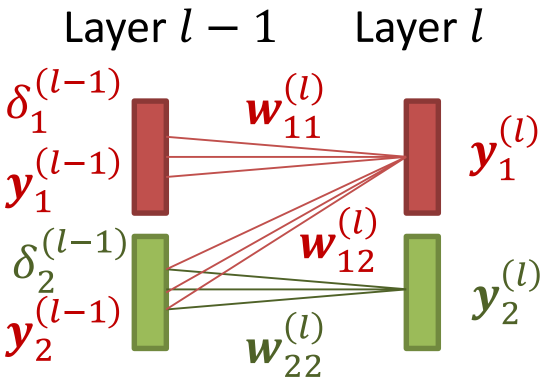
A Yes

B No

Submit

Local sensitivity in the example

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



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

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

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,

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

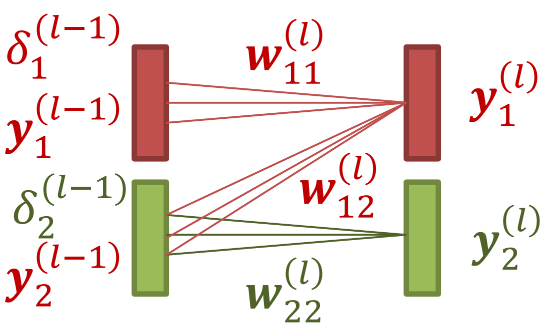
  - No. The correct answer is

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

Local sensitivity in general

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

Layer $l-1$ Layer l

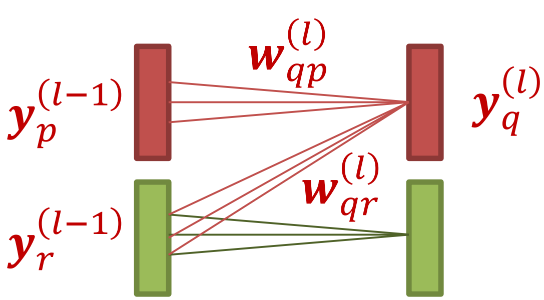


$$\boldsymbol{\delta}_1^{(l-1)} = \boldsymbol{\delta}_1^{(l)} *_{\text{full}} \boldsymbol{w}_{11}^{(l)}$$

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

- In general

Layer $l-1$ Layer l



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

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

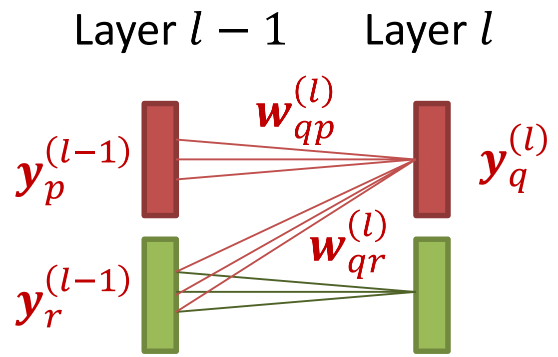
Summary for 1D convolutional layer

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

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

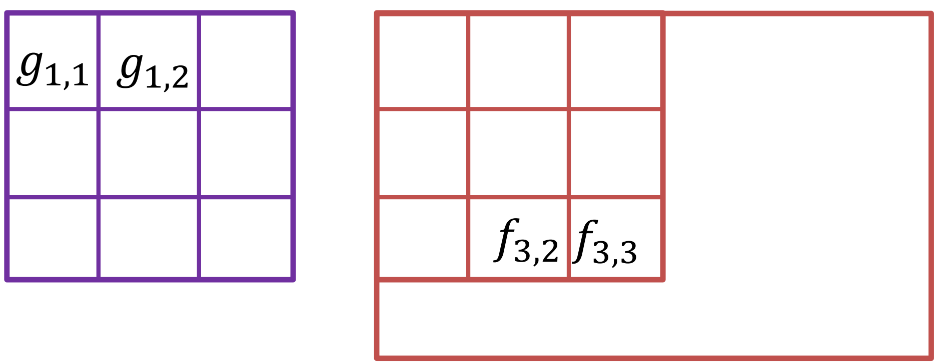
$$\boldsymbol{\delta}_p^{(l-1)} = \sum_{q \in \widetilde{M}_p} \boldsymbol{\delta}_q^{(l)} *_{\text{full}} \boldsymbol{w}_{qp}^{(l)}$$

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

Recall: 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 $m = 4, n = 4$

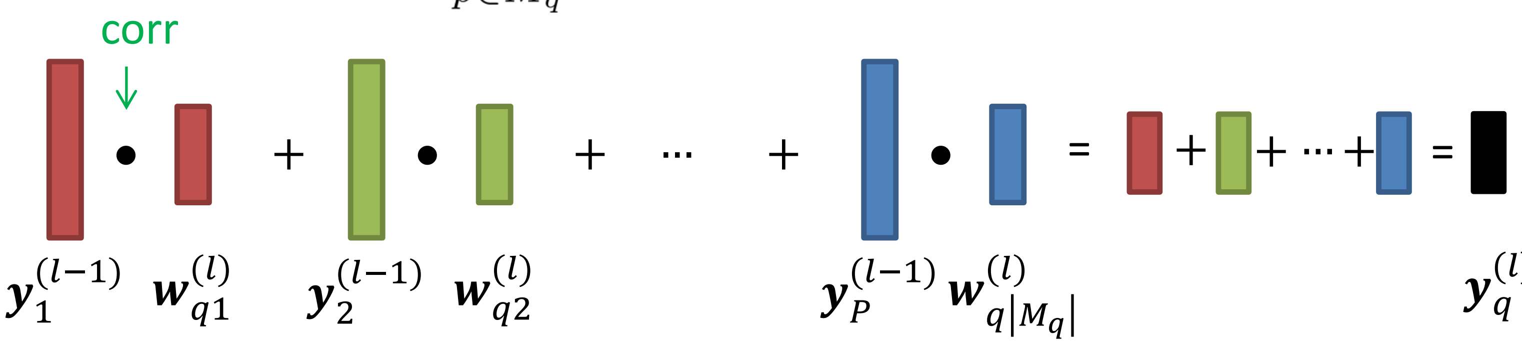
$$(f * g)_{m,n} = f_{3,3}g_{1,1} + f_{3,2}g_{1,2} + f_{3,1}g_{1,3} + f_{2,3}g_{2,1} + \cdots$$

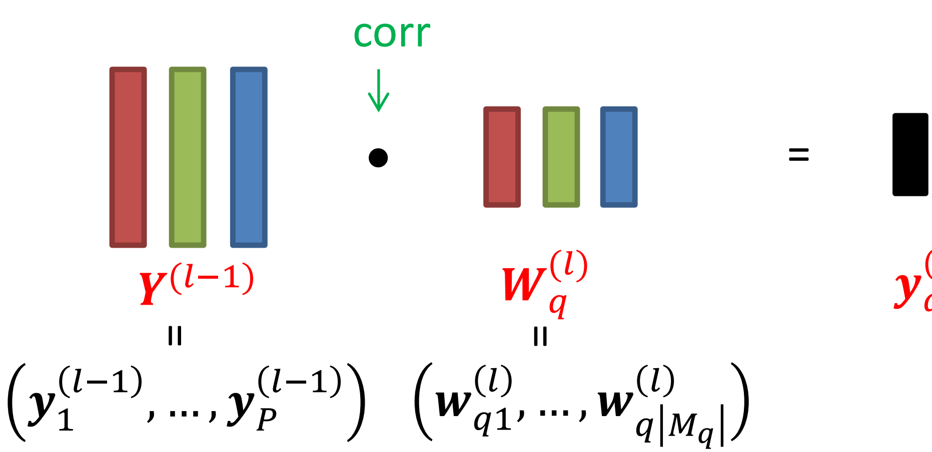
- valid shape: the size of h is $(M - K_1 + 1) \times (N - K_2 + 1)$
- full shape: the size of h is $(M + K_1 - 1) \times (N + K_2 - 1)$
- same shape: the size of h is $M \times N$

Do summation using 2D convolution

- Forward pass

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



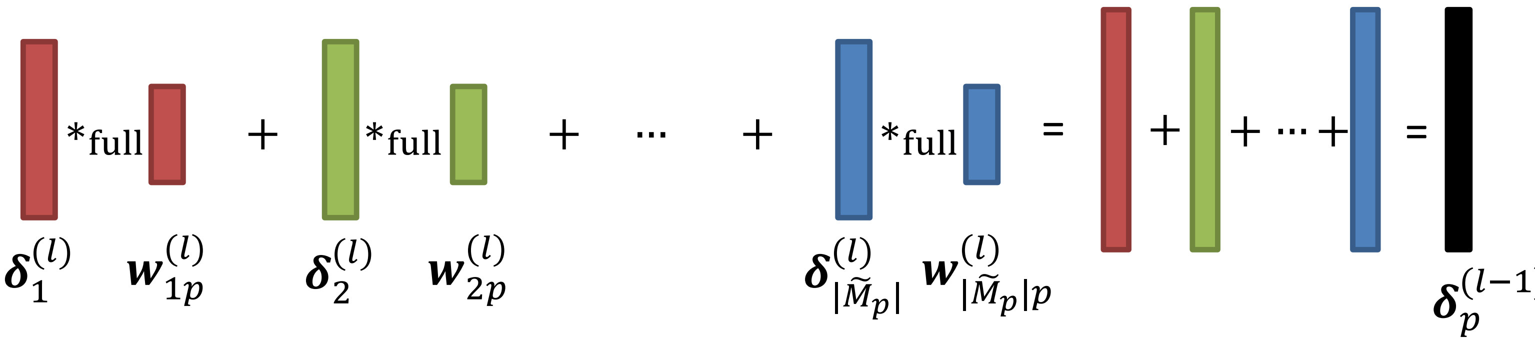


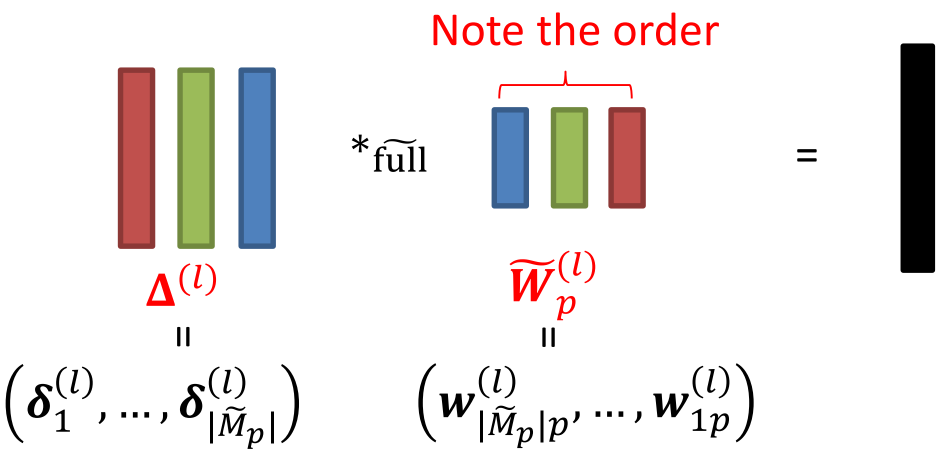
$$\boldsymbol{y}_q^{(l)} = \boldsymbol{Y}^{(l-1)} *_{\text{valid}} \text{rot180}\left(\boldsymbol{W}_q^{(l)}\right) + b_q^{(l)}$$

Do summation using 2D convolution

- Backward pass

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



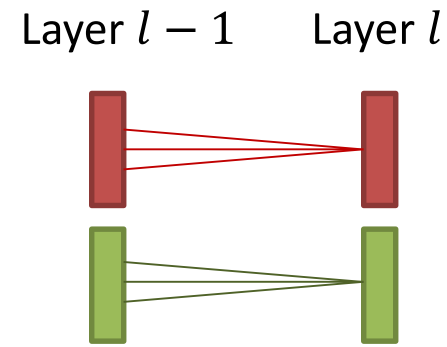


$$\boldsymbol{\delta}_p^{(l-1)} = \boldsymbol{\Delta}^{(l)} *_{\widetilde{\text{full}}} \widetilde{\boldsymbol{W}}_p^{(l)}$$

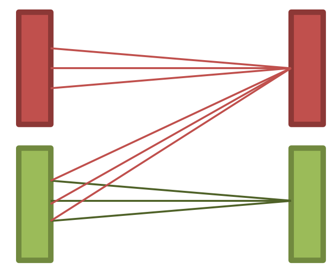
This "full" convolution only applies in the vertical dim, while in the horizontal dim (along q) the convolution type is "valid"

Derive BP algorithm in different cases

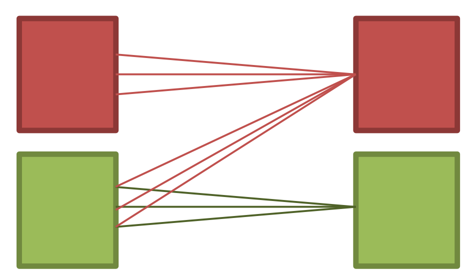
1. The 1D convolution case without feature combination



2. The 1D convolution case with feature combination

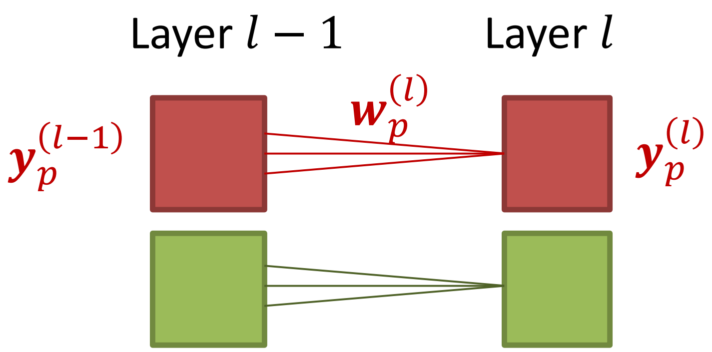


3. The 2D convolution case



2D convolution without feature combination

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



In what follows, we drop the index p

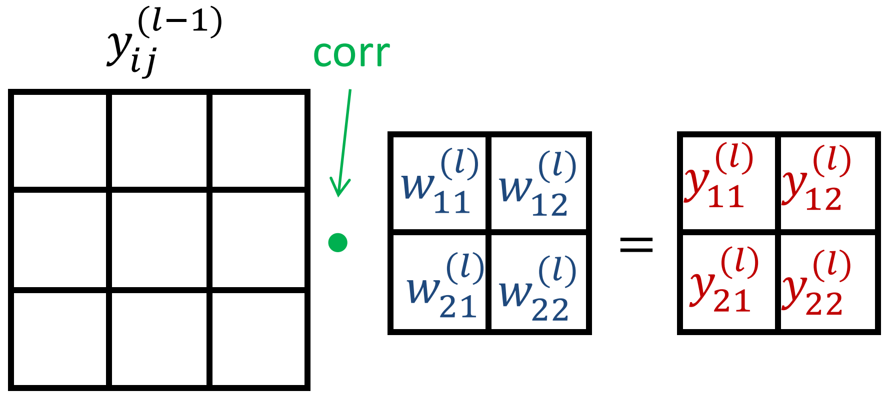
- Convolve every filter $\boldsymbol{w}_p^{(l)}$ with the p-th feature map $\boldsymbol{y}_p^{(l-1)}$ in the previous layer and obtain a new feature map

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

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

Forward pass in an example

Consider one single feature map in layer l



Layer $l-1$ Layer l

- The output in layer l

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

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

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

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

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

Gradient calculation in the example

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

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

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

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

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

- Partial derivative w.r.t. $\boldsymbol{w}^{(l)}$ and $b^{(l)}$

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

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

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

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

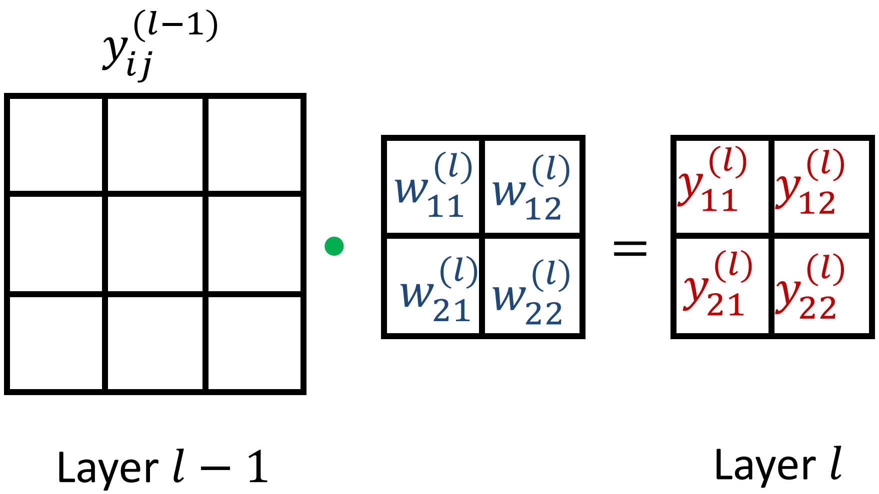
$$\partial E^{(n)} / \partial b^{(l)} = \delta_{11}^{(l)} + \delta_{12}^{(l)} + \delta_{21}^{(l)} + \delta_{22}^{(l)}$$

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

General result

Local sensitivity in the example

Consider one single feature map in layer l



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

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 show that

$$\delta_{11}^{(l-1)} = \delta_{11}^{(l)} w_{11}^{(l)}, \quad \delta_{12}^{(l-1)} = \delta_{11}^{(l)} w_{12}^{(l)} + \delta_{12}^{(l)} w_{11}^{(l)}, \quad \delta_{13}^{(l-1)} = \delta_{12}^{(l)} w_{12}^{(l)},$$

$$\delta_{21}^{(l-1)} = \delta_{11}^{(l)} w_{21}^{(l)} + \delta_{21}^{(l)} w_{11}^{(l)}, \quad \delta_{22}^{(l-1)} = \delta_{11}^{(l)} w_{22}^{(l)} + \delta_{12}^{(l)} w_{21}^{(l)} + \delta_{21}^{(l)} w_{12}^{(l)} + \delta_{22}^{(l)} w_{11}^{(l)}$$

$$\delta_{23}^{(l-1)} = \delta_{12}^{(l)} w_{22}^{(l)} + \delta_{22}^{(l)} w_{12}^{(l)},$$

$$\delta_{31}^{(l-1)} = \delta_{21}^{(l)} w_{21}^{(l)}, \quad \delta_{32}^{(l-1)} = \delta_{21}^{(l)} w_{22}^{(l)} + \delta_{22}^{(l)} w_{21}^{(l)}, \quad \delta_{33}^{(l-1)} = \delta_{22}^{(l)} w_{22}^{(l)}$$

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

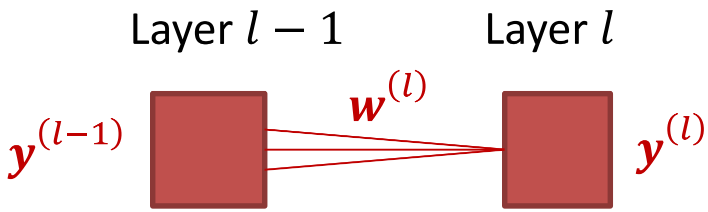
General result

where

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

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:

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

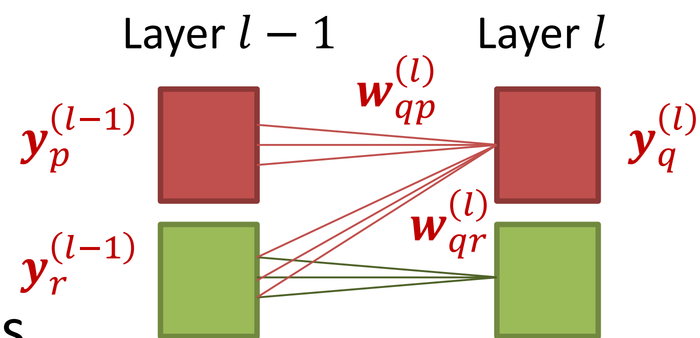
  - Local sensitivity:

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

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

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

- Backward pass
  - Gradient:

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

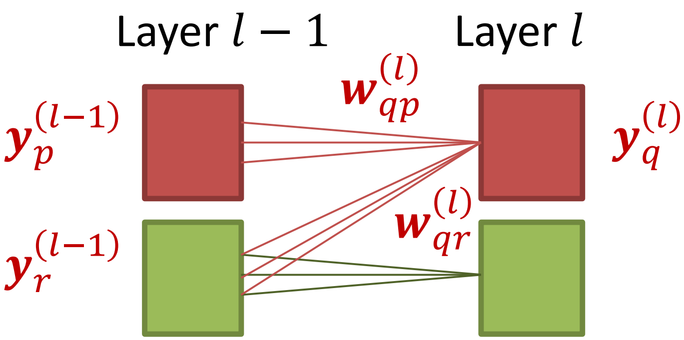
  - Local sensitivity:

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

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

Same as 1D case

Do summation using 3D convolution



Forward pass:

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

- Define 3D matrices (tensors)

width height

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

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

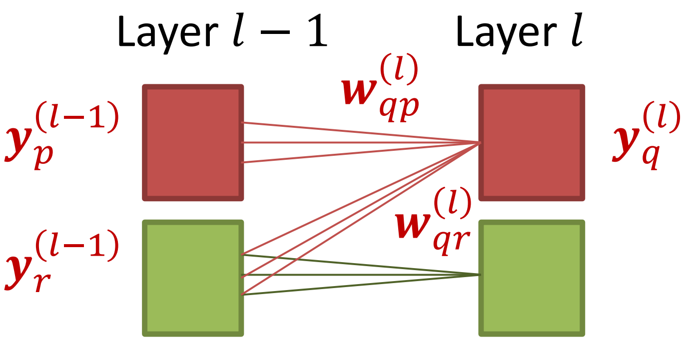
where $|\cdot|$ denotes the cardinality of a set; M, K_1: width; N, K_2: height

- The forward pass can be expressed as

$$\boldsymbol{y}_q^{(l)} = \boldsymbol{Y}^{(l-1)} *_{\text{valid}} \text{rot180}(\boldsymbol{W}_q^{(l)}) + b_q^{(l)}$$

No!

Do summation using 3D convolution



Forward pass:

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

- Define 3D matrices (tensors)

width height

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

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

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

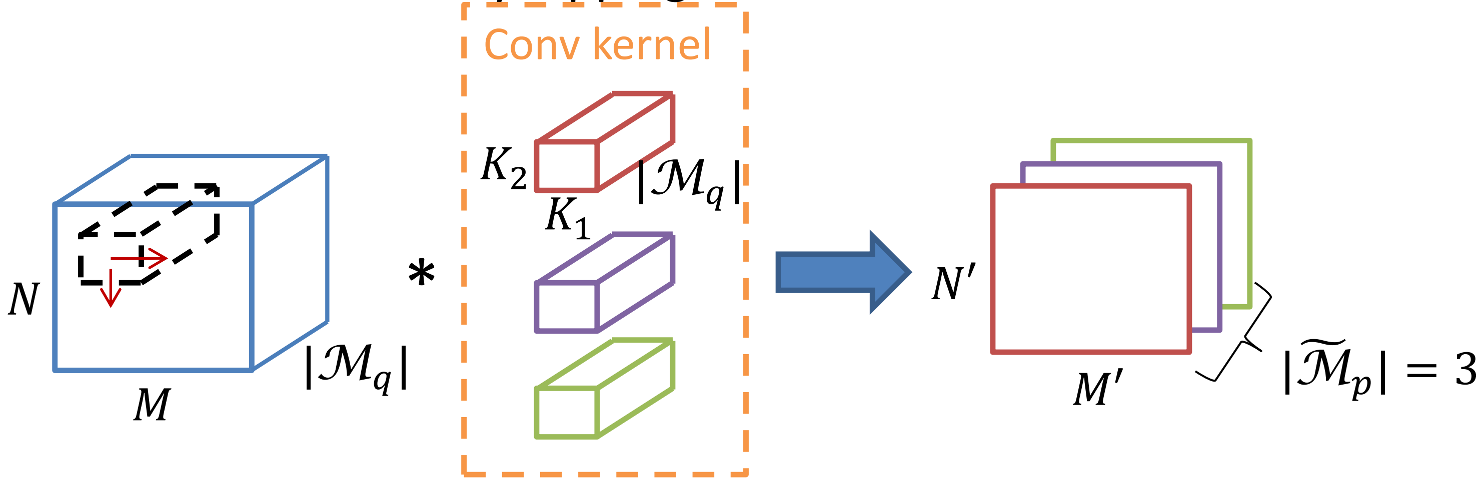
- The forward pass can be expressed as

$$\boldsymbol{y}_q^{(l)} = \boldsymbol{Y}^{(l-1)} *_{\text{valid}} \text{flip}_{012}(\boldsymbol{W}_q^{(l)}) + b_q^{(l)}$$

where flip_{012} means flip along all of the 3 dimensions

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 $|\widetilde{\mathcal{M}}_p| \times |\mathcal{M}_q| \times K_1 \times K_2$

Do 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

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

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

width height

- Then

$$\boldsymbol{\delta}_p^{(l-1)} = \boldsymbol{\Delta}^{(l)} *_{\widetilde{\text{full}}} \widetilde{\boldsymbol{W}}_p^{(l)}$$

  - This "full" convolution only applies in the 2nd and 3rd dimensions, while in the 1st dimension (along q) the convolution type is "valid"

Outline

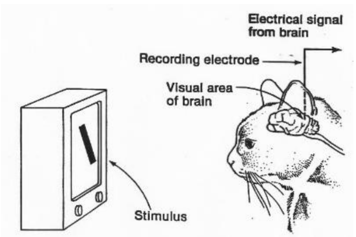
1. Introduction
2. Convolution
   – Forward pass
   – Backward pass
3. Summary

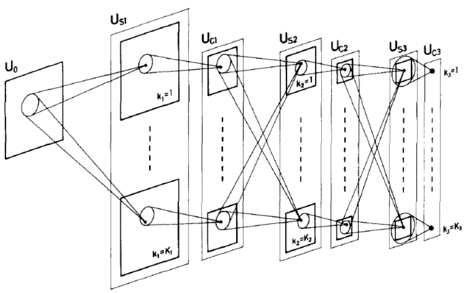
Summary of this lecture

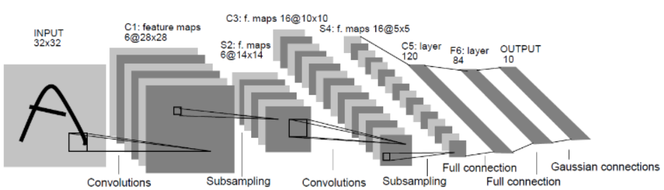
Knowledge

1. Introduction

History







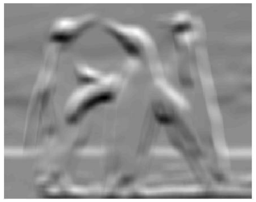
Convolution

inpu

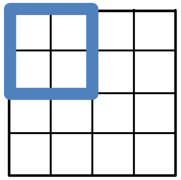
kernel

*

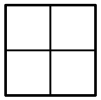
feature



Pooling



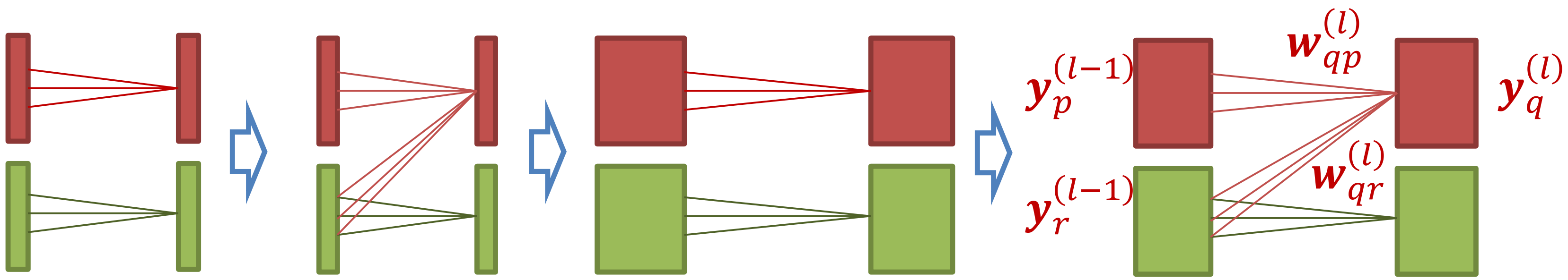
average or max



Summary of this lecture

Knowledge

- Convolutional layer



➢ Forward pass

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

➢ Backward pass

Gradient:

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

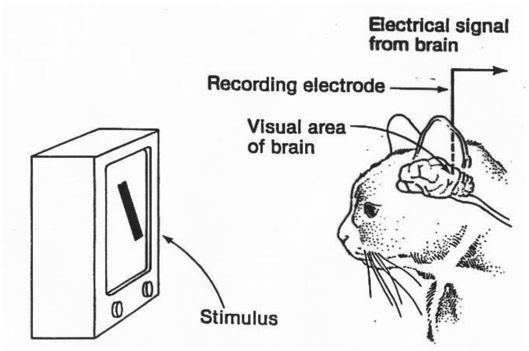
Local sensitivity:

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

Summary of this lecture

Capability and value

- Neuroscience played a significant role in CNN, and should continue to play a significant role
- The ability to extract general principle from neuroscience findings and apply to a computational model
- We have a lot of Yann LeCun's nowadays, but lack a Kunihiko Fukushima



Recommended reading

- Fukushima (1980)

  Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition

  Biological Cybernetics