Introduction to Computer Vision

Lecture 5 - Deep Learning II

Prof. He Wang

Logistics

- Assignment 1: to release on 3/14 (this Friday evening), due on 3/29 11:59PM (Saturday)
  - Implementing convolution operation
  - Canny edge detector
  - Harris corner detector
  - Plane fitting using RANSAC

- Some functions are required to be implemented without for loop.

- If 1 day (0 - 24 hours) past the deadline, 15% off
- If 2 day (24 - 48 hours) past the deadline, 30% off
- Zero credit if more than 2 days.

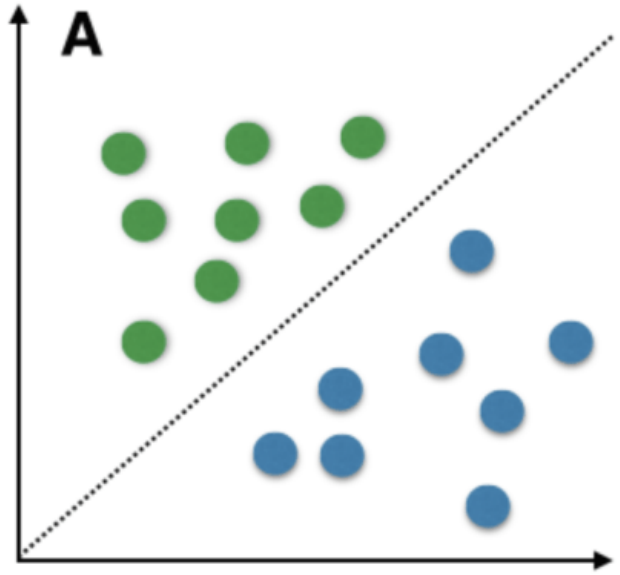
Outline

- Set up the task
- Prepare the data —> Need a labeled dataset.
- Built a model —> construct your neural network
- Decide the fitting/training objective —> Loss function
- Perform fitting —> Training by running optimization
- Testing —> Evaluating on test data

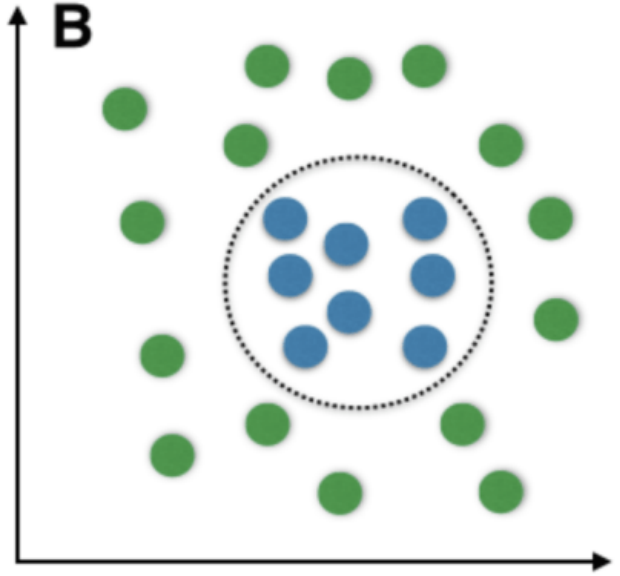
Multilayer Perceptron

Problem with Single-Layer Network

- $g(\theta^T x) = 0$ is a hyperplane in the space of x
- can only handle linear separable cases

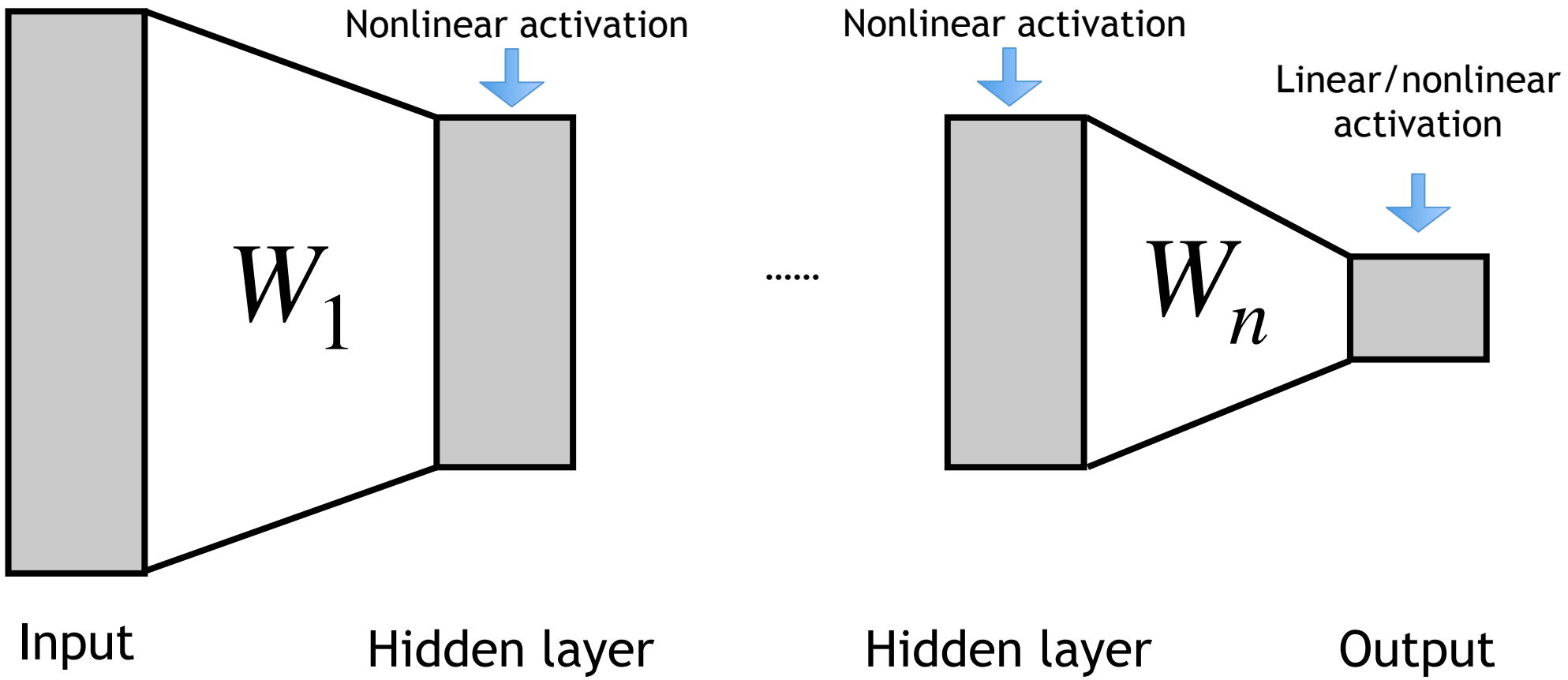


Linear separable



Linear non-separable

Multi-Layer Perceptron (MLP)



- MLP: Stacking linear layer and nonlinear activations.
- Through many non-linear layers, transform a linear non-separable problem to linear separable at the last layer

Classification function with MLP

$$f(x;\theta) = Wx + b$$

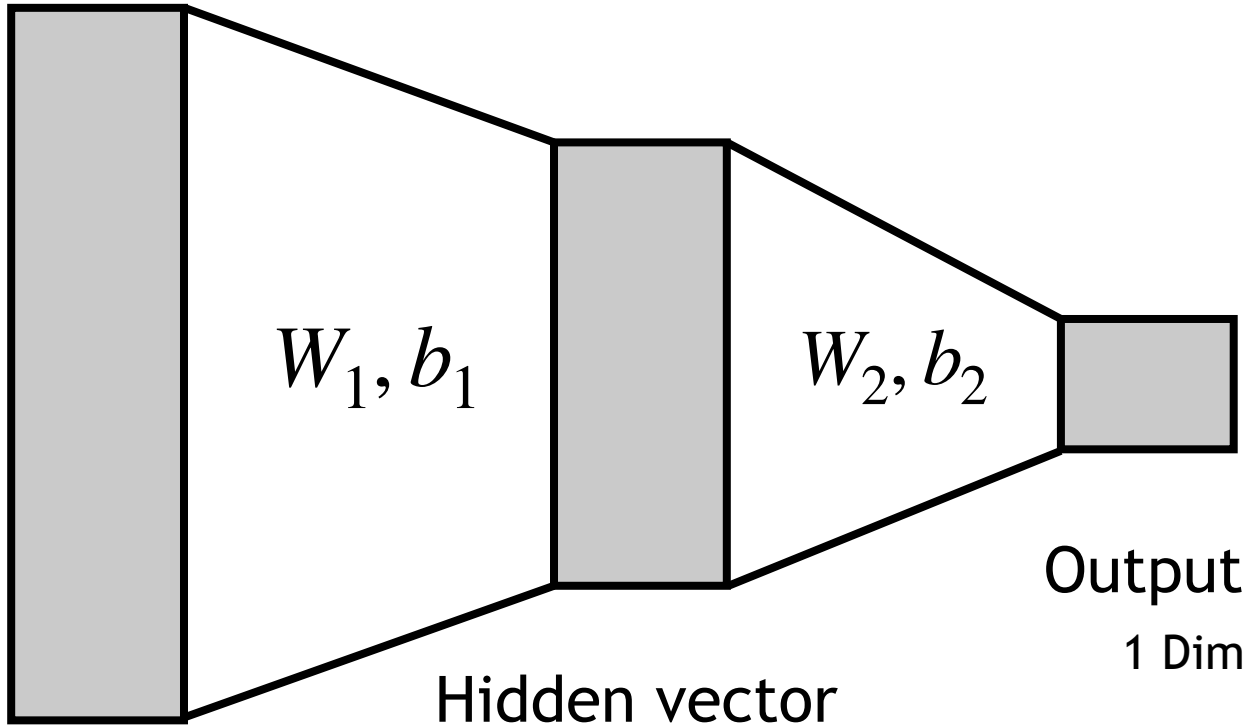
Linear function

$$f(x;\theta) = g(W_2 g(W_1 x + b_1) + b_2)$$

2-layer MLP,
or fully-conncted layers

In practice, we can concat the input variables with extract 1 for learning bias.

Classification function with MLP



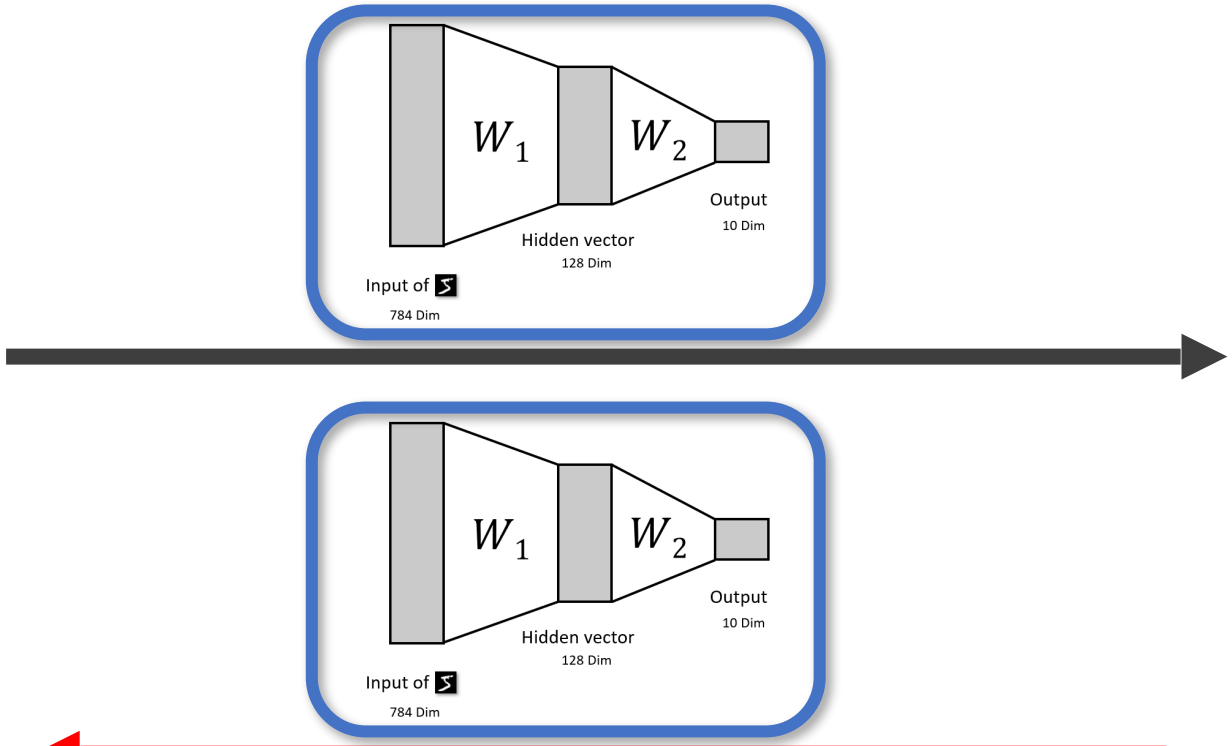
$$W_1 \in \mathbb{R}^{784\times128}, W_2 \in \mathbb{R}^{128\times1}$$
$$b_1 \in \mathbb{R}^{128}, b_2 \in \mathbb{R}$$

How can we obtain the parameters/weights of the MLP?

Classification function with MLP

1. Initialization: randomly generate the weights $\mathbf{W}_1 \in \mathbb{R}^{784\times128}, \mathbf{W}_2 \in \mathbb{R}^{128\times10}$

2. Forwarding:



$\mathbf{y}$

3. Gradient decent:

θ_{new}

Update weights

Gradient $\frac{\partial \mathcal{L}}{\partial \theta}$

$\mathcal{L}(y, y^*)$

Analytical Gradient?

So, if we can compute the gradient $\frac{\partial \mathbf{L}}{\partial \mathbf{W}_1}, \frac{\partial \mathbf{L}}{\partial \mathbf{W}_2}$, then we can update $\mathbf{W}_1, \mathbf{W}_2$

An intuitive idea : Derive $\frac{\partial \mathbf{L}}{\partial \mathbf{W}}$ by hand

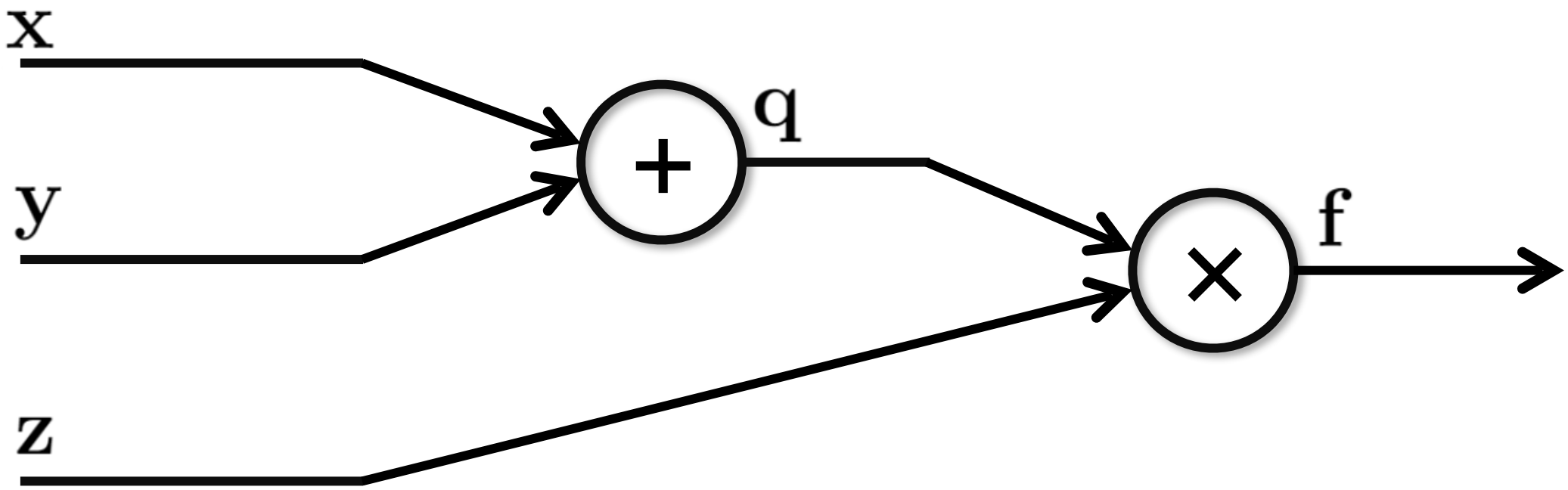
However:

- Lots of matrix calculus
- Infeasible: any modification requires re-derivation

Backpropagation with a toy example

Let we consider a toy example:
(Computational graph)

$$f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = (\mathbf{x} + \mathbf{y}) \cdot \mathbf{z}$$

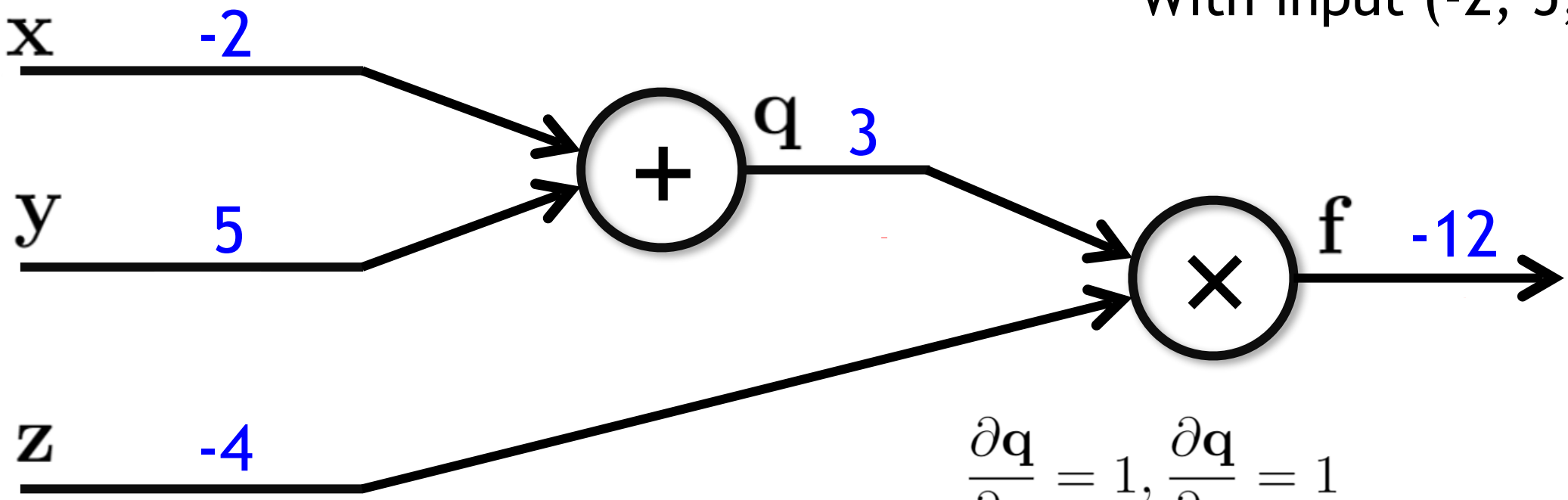


And we want $\frac{\partial f}{\partial \mathbf{x}}, \frac{\partial f}{\partial \mathbf{y}}, \frac{\partial f}{\partial \mathbf{z}}$

Backpropagation with a toy example

Let we consider a toy example: $f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = (\mathbf{x} + \mathbf{y}) \cdot \mathbf{z}$

With input (-2, 5, -4)



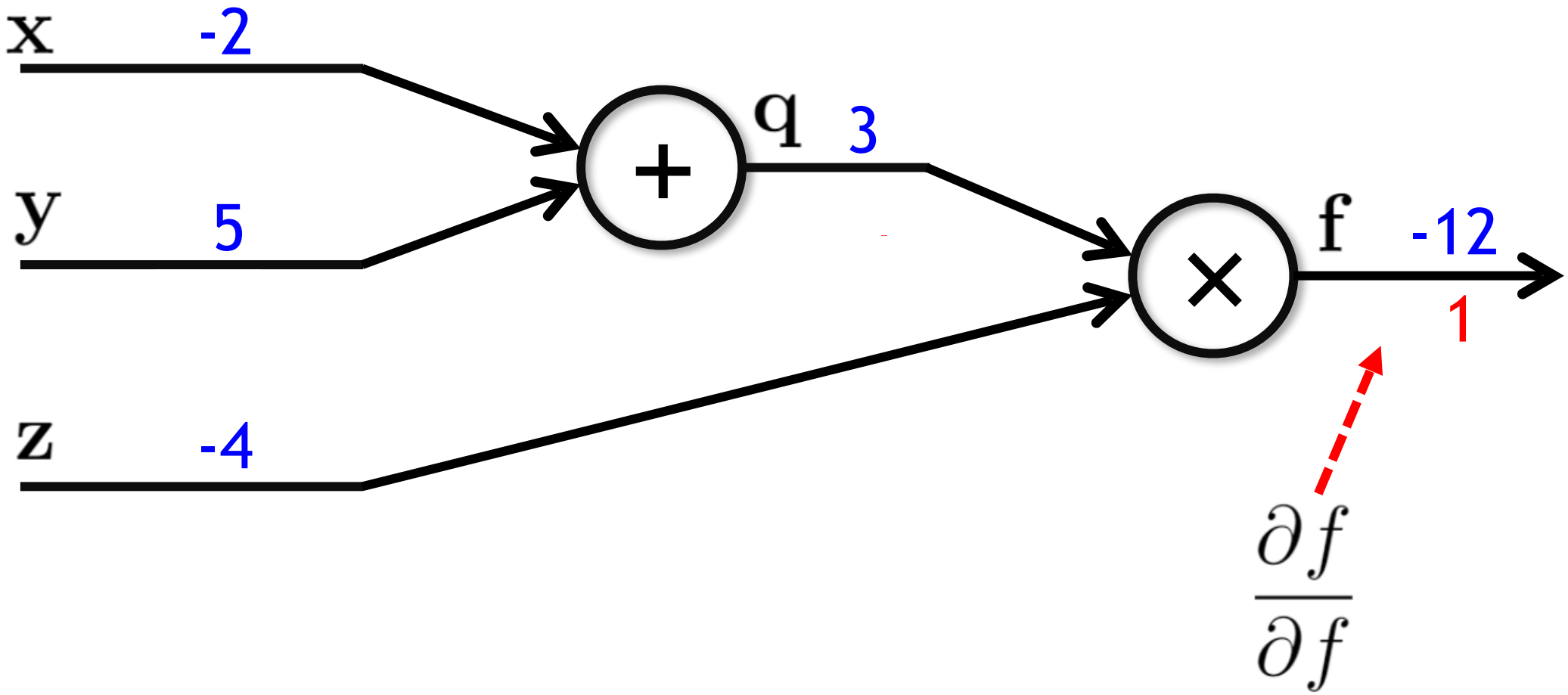
And we have the derivation

$$\frac{\partial \mathbf{q}}{\partial \mathbf{x}} = 1, \frac{\partial \mathbf{q}}{\partial \mathbf{y}} = 1$$

$$\frac{\partial f}{\partial \mathbf{q}} = \mathbf{z}, \frac{\partial f}{\partial \mathbf{z}} = \mathbf{q}$$

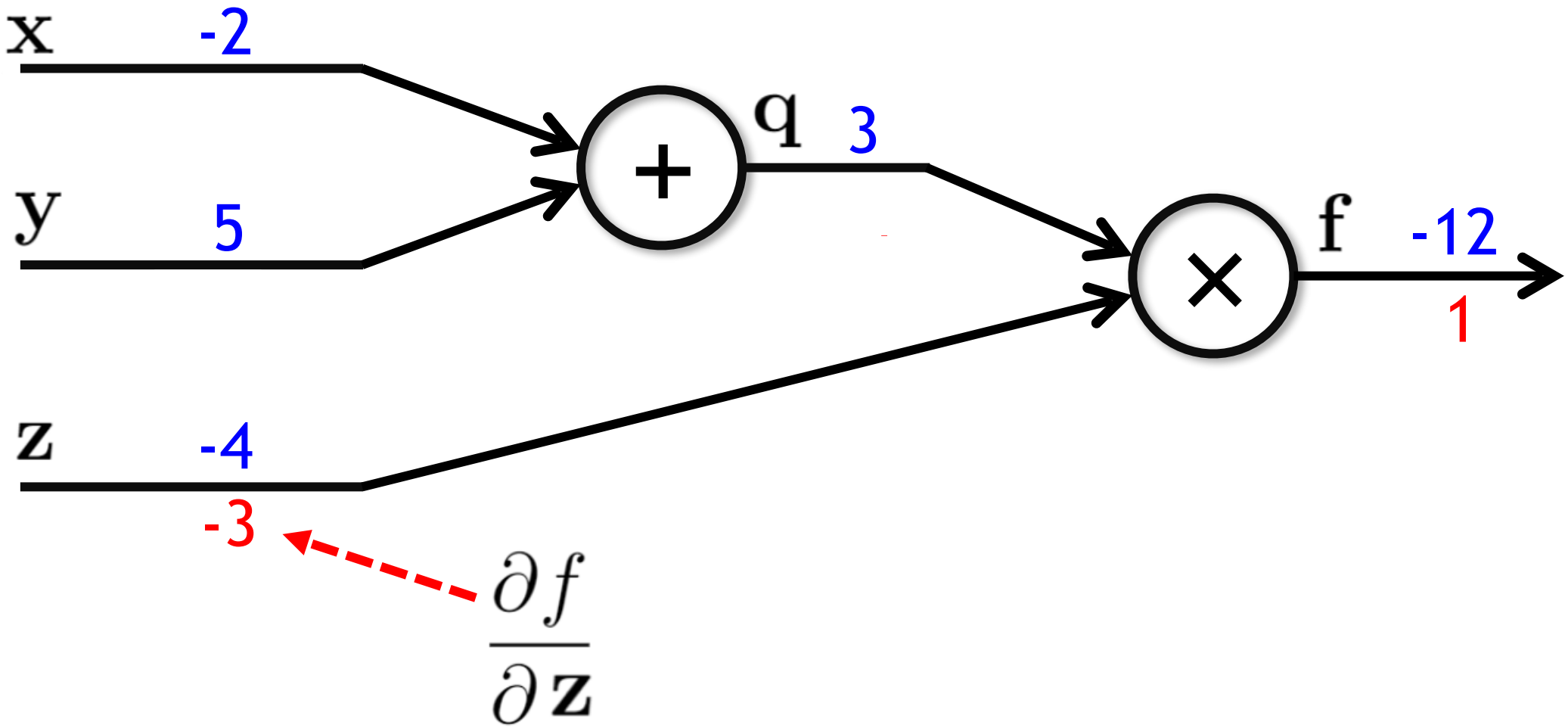
Backpropagation with a toy example

Let we consider a toy example: $f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = (\mathbf{x} + \mathbf{y}) \cdot \mathbf{z}$



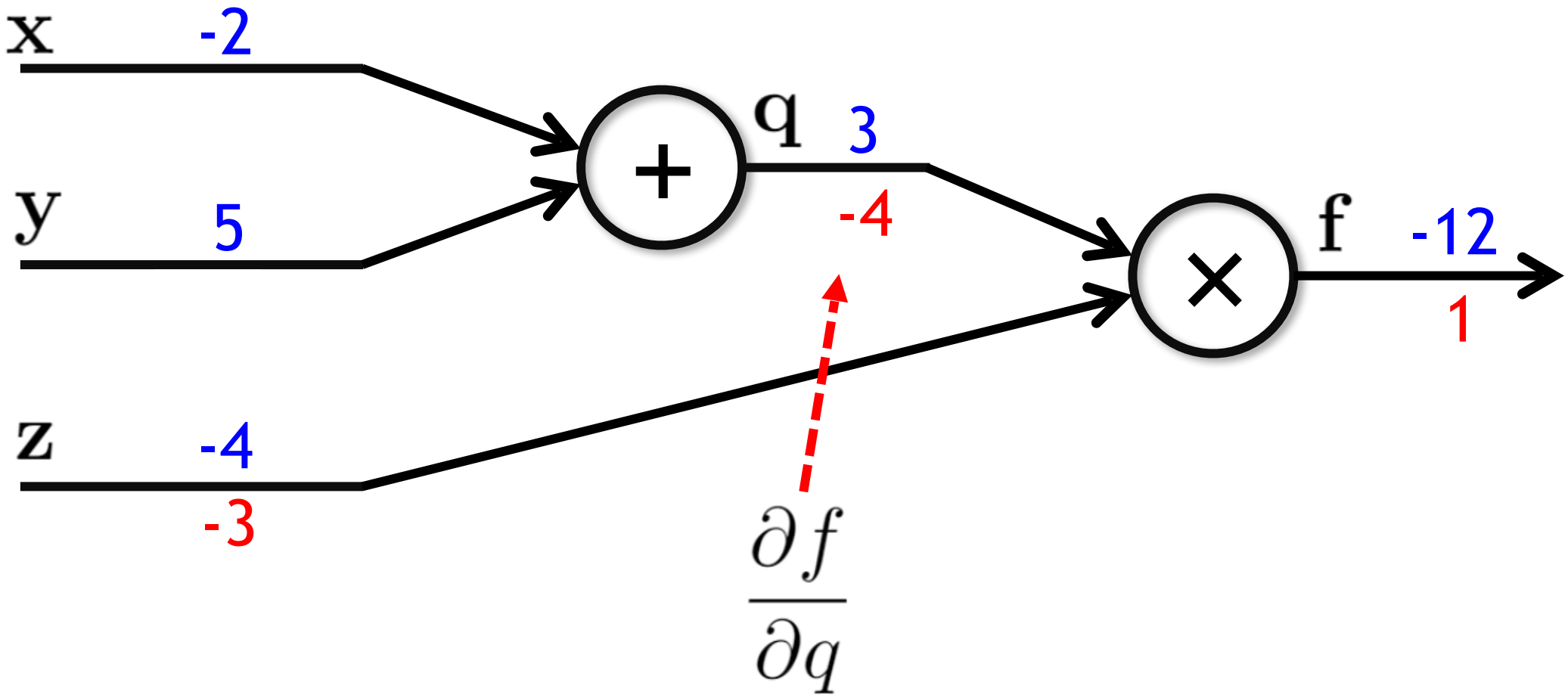
Backpropagation with a toy example

Let we consider a toy example: $f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = (\mathbf{x} + \mathbf{y}) \cdot \mathbf{z}$



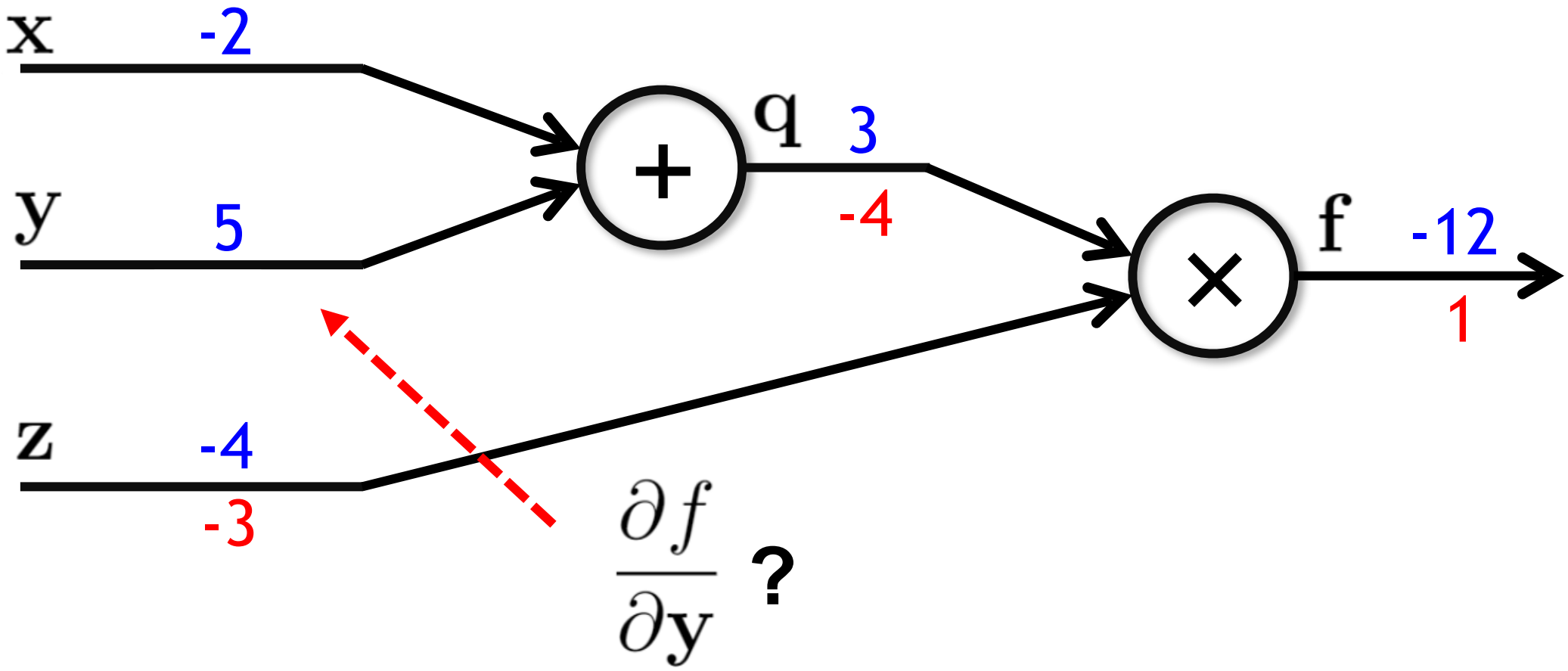
Backpropagation with a toy example

Let we consider a toy example: $f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = (\mathbf{x} + \mathbf{y}) \cdot \mathbf{z}$



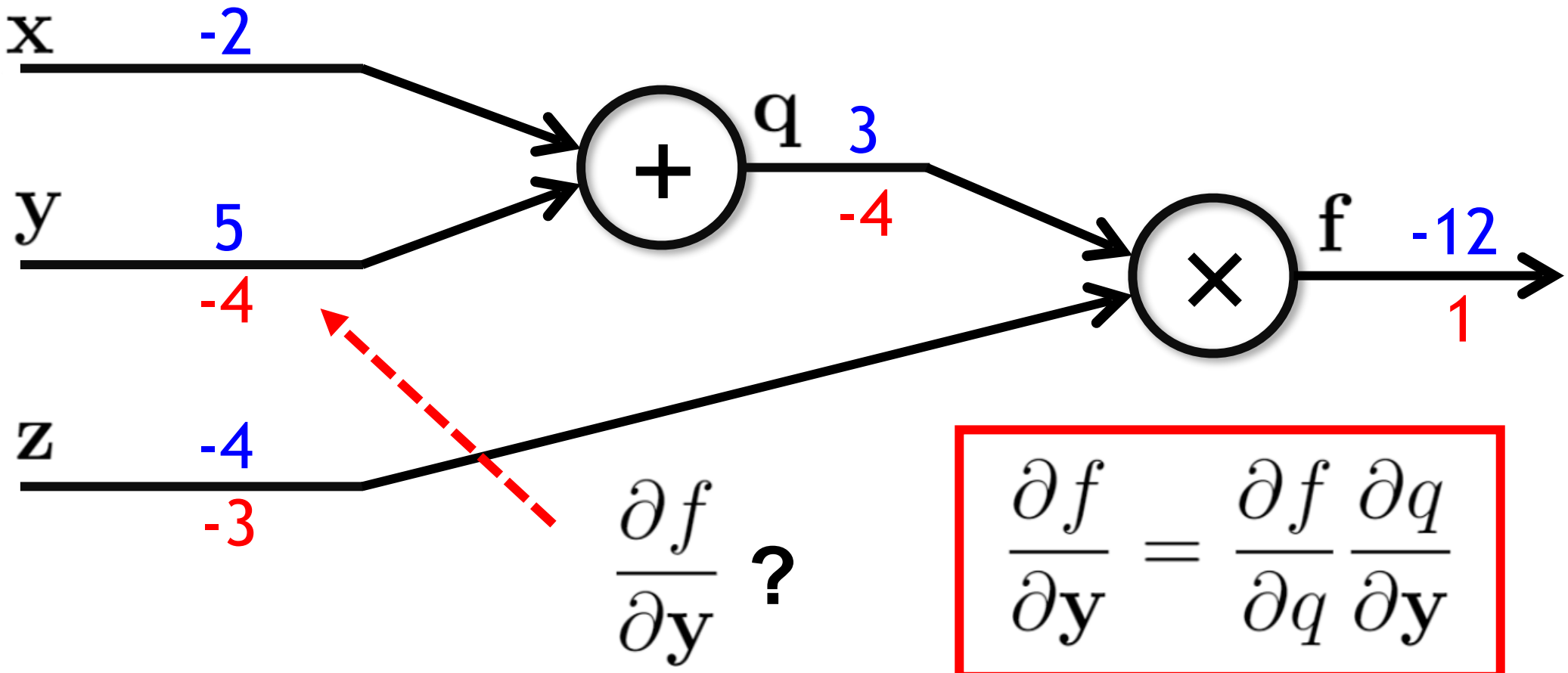
Backpropagation with a toy example

Let we consider a toy example: $f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = (\mathbf{x} + \mathbf{y}) \cdot \mathbf{z}$



Backpropagation with a toy example

Let we consider a toy example: $f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = (\mathbf{x} + \mathbf{y}) \cdot \mathbf{z}$

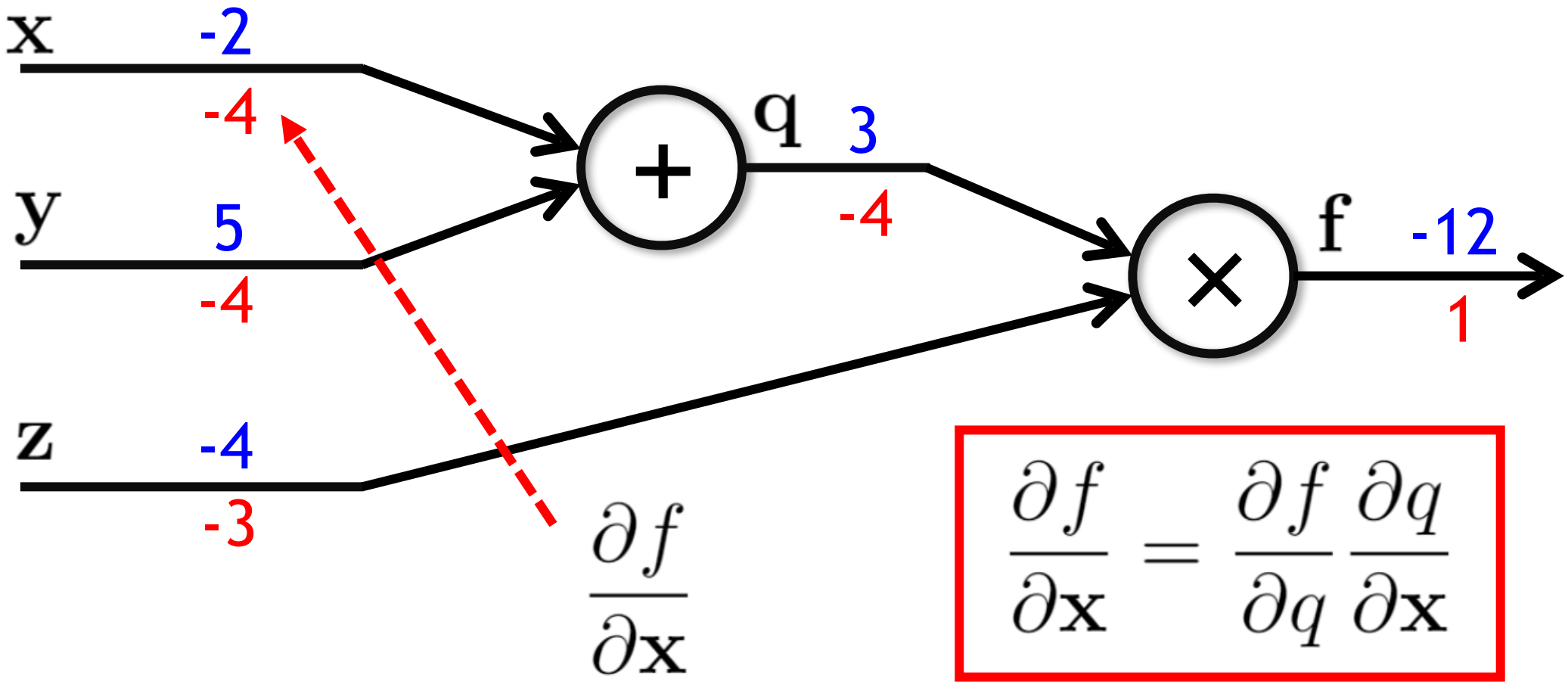


$$\frac{\partial f}{\partial \mathbf{y}} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial \mathbf{y}}$$

Chain rule

Backpropagation with a toy example

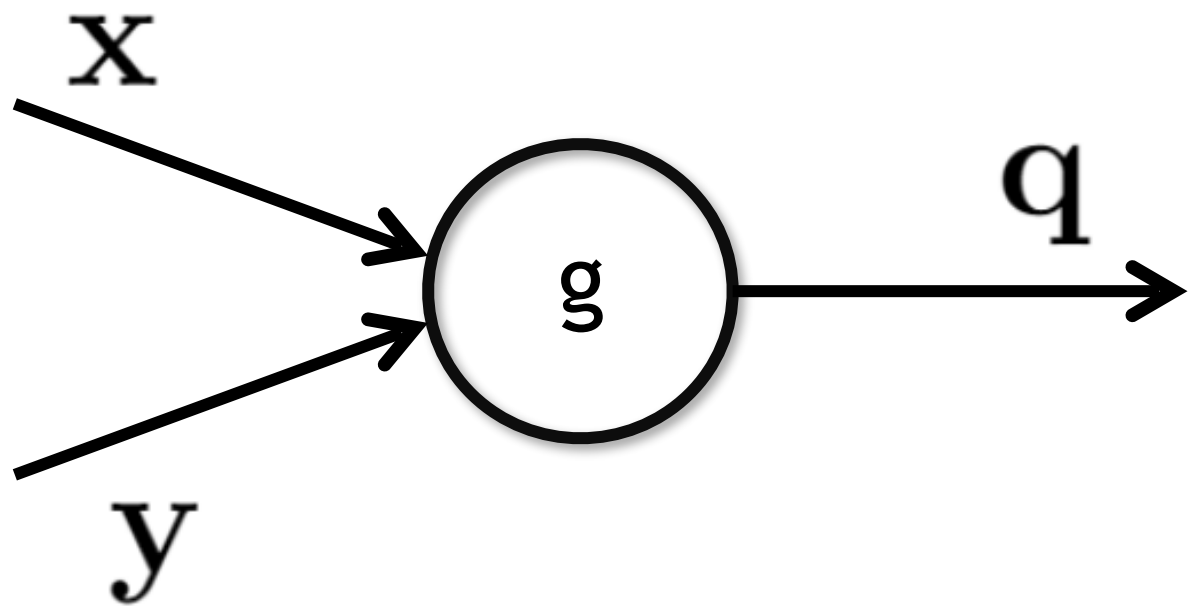
Let we consider a toy example: $f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = (\mathbf{x} + \mathbf{y}) \cdot \mathbf{z}$



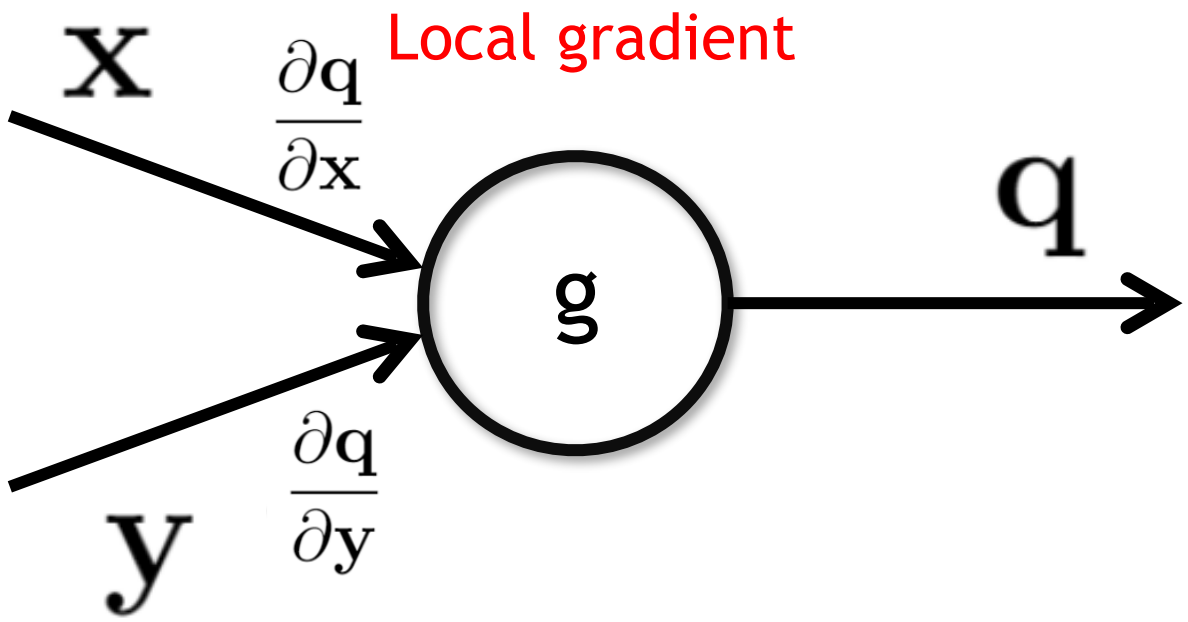
$$\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial \mathbf{x}}$$

Chain rule

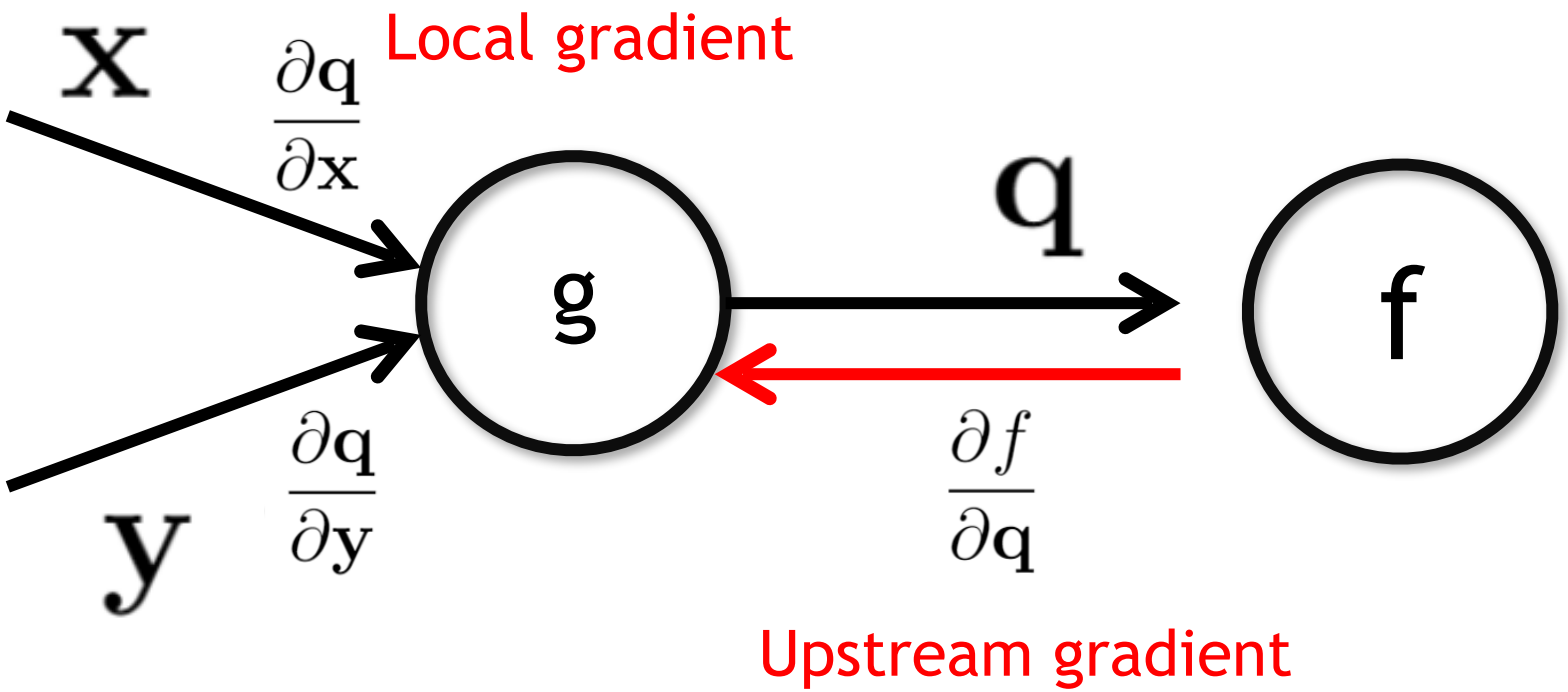
Chain rule



Chain rule

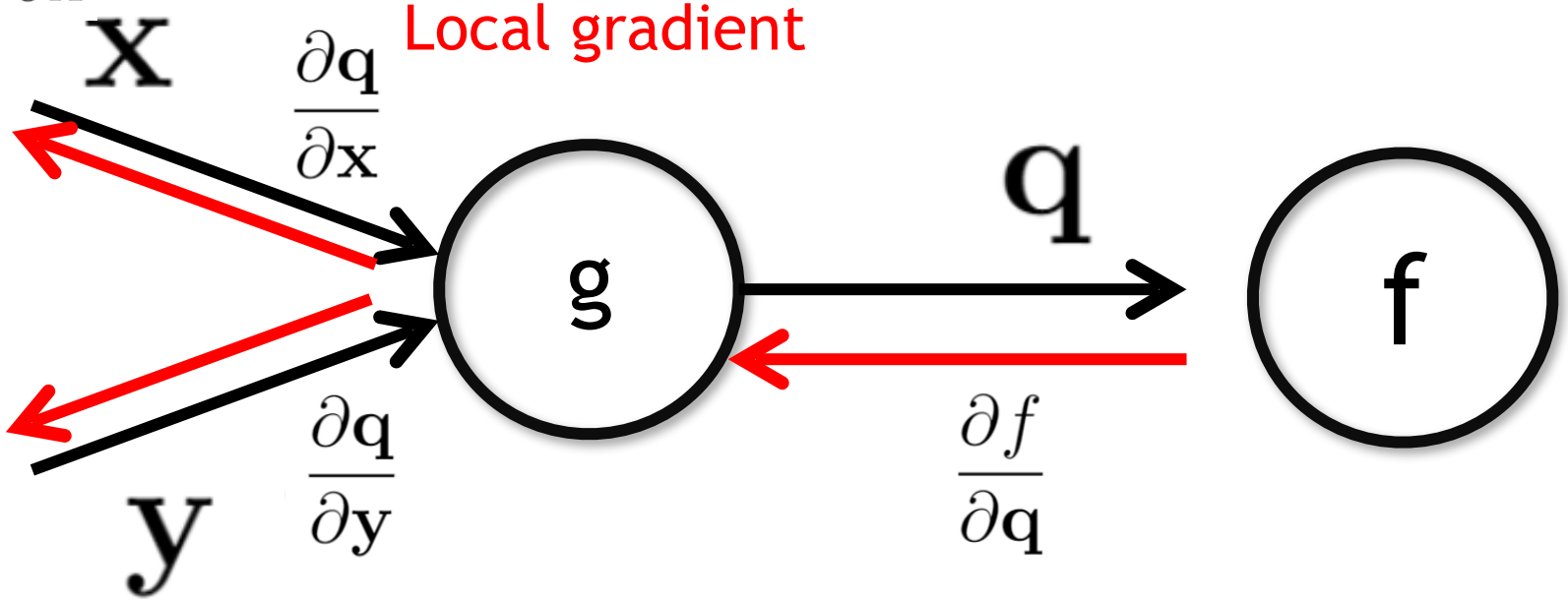

x
$\frac{\partial q}{\partial x}$
Local gradient
g
q
y
$\frac{\partial q}{\partial y}$

Chain rule


x
y
$\frac{\partial q}{\partial x}$
$\frac{\partial q}{\partial y}$
Local gradient
g
q
f
$\frac{\partial f}{\partial q}$
Upstream gradient

Chain rule

$$\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial \mathbf{x}}$$

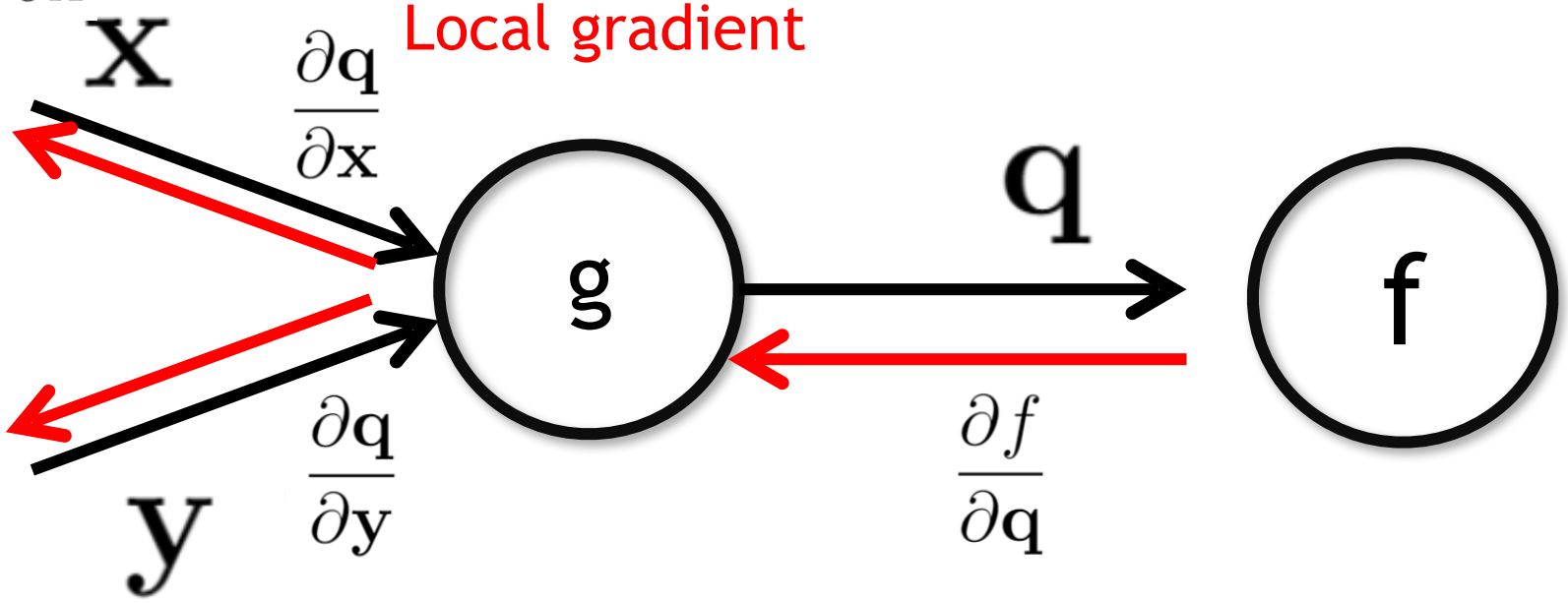


$$\frac{\partial f}{\partial \mathbf{y}} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial \mathbf{y}}$$

Downstream gradient

Chain rule

$$\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial \mathbf{x}}$$



$$\frac{\partial f}{\partial \mathbf{y}} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial \mathbf{y}}$$

Downstream gradient

Upstream gradient

The backpropagation can be efficiently implemented with simple matrix operations

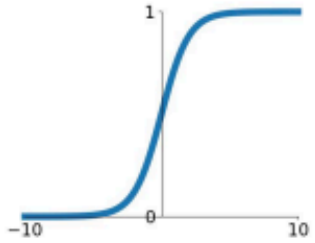
"Backpropagation for a Linear Layer" from Justin 2017

Activation Function

Activation functions

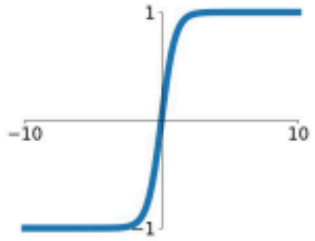
Sigmoid

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



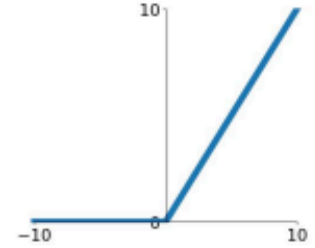
tanh

$$\tanh(x)$$



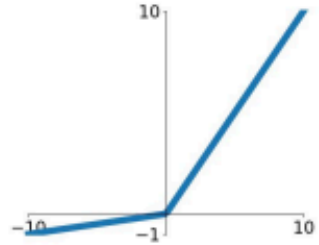
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

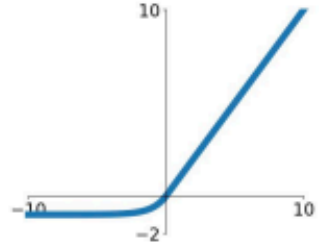


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

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



Slides credit: Stanford CS231N

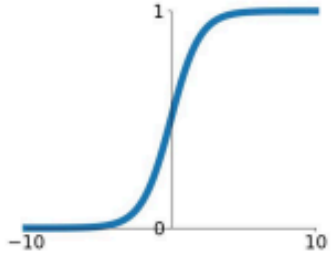
Activation Function

Activation functions

ReLU is a good default choice for most problems

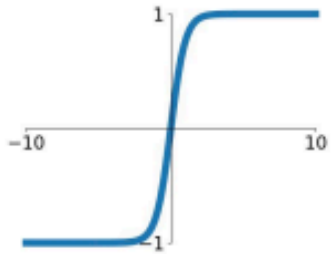
Sigmoid

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



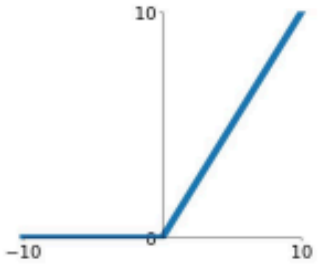
tanh

$$\tanh(x)$$



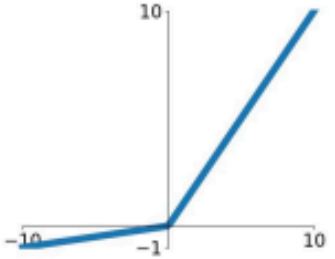
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

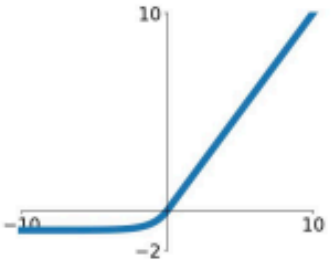


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

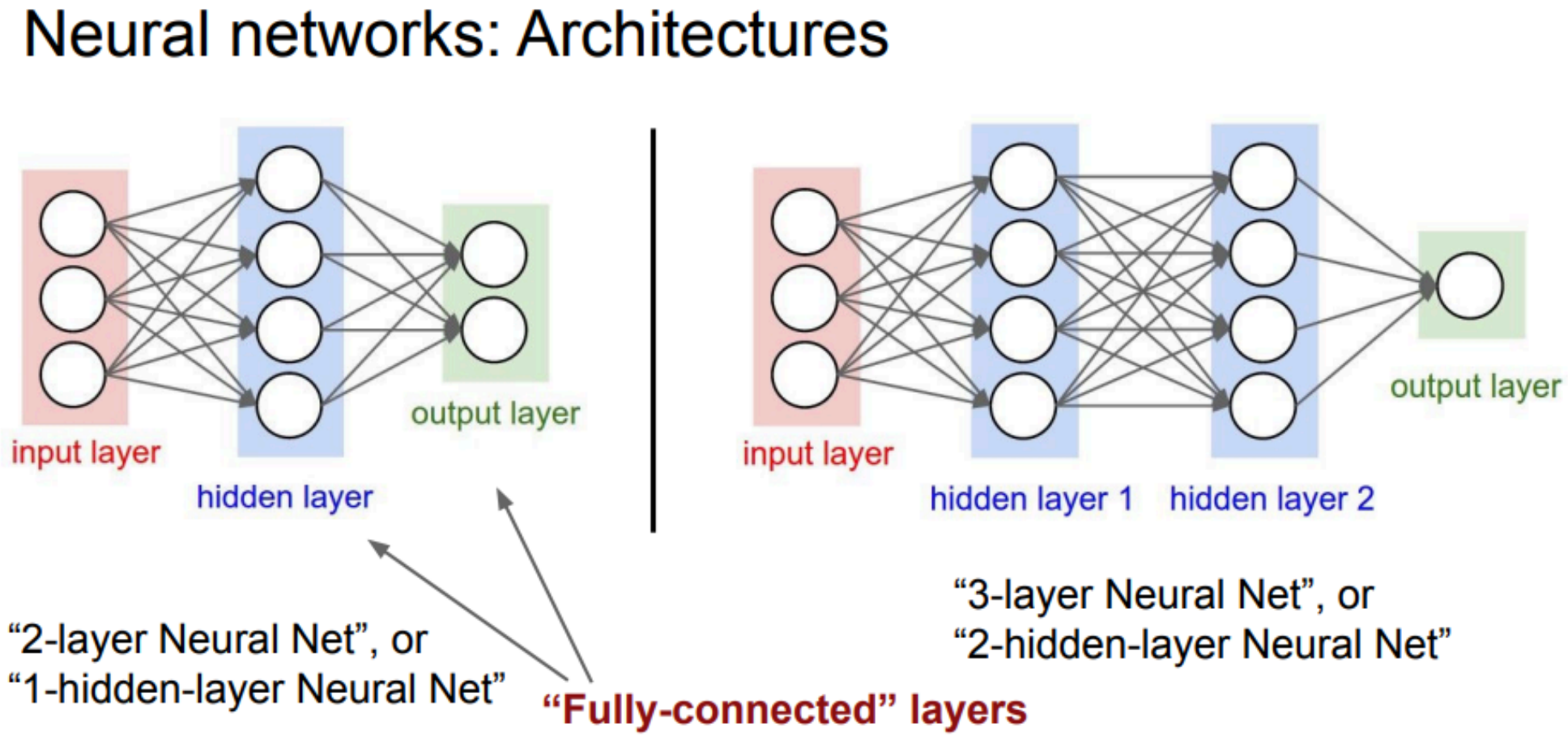
ELU

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



Slides credit: Stanford CS231N

Review



Slides credit: Stanford CS231N

Problems for Using MLP to Process Vision Signals

- Flatten an image into a vector would be very expensive for high resolution images

- Flattening operation breaks the local structure of an image.

Convolutional Neural Network

Slides are borrowed from Stanford CS231N.

Convolutional Neural Network

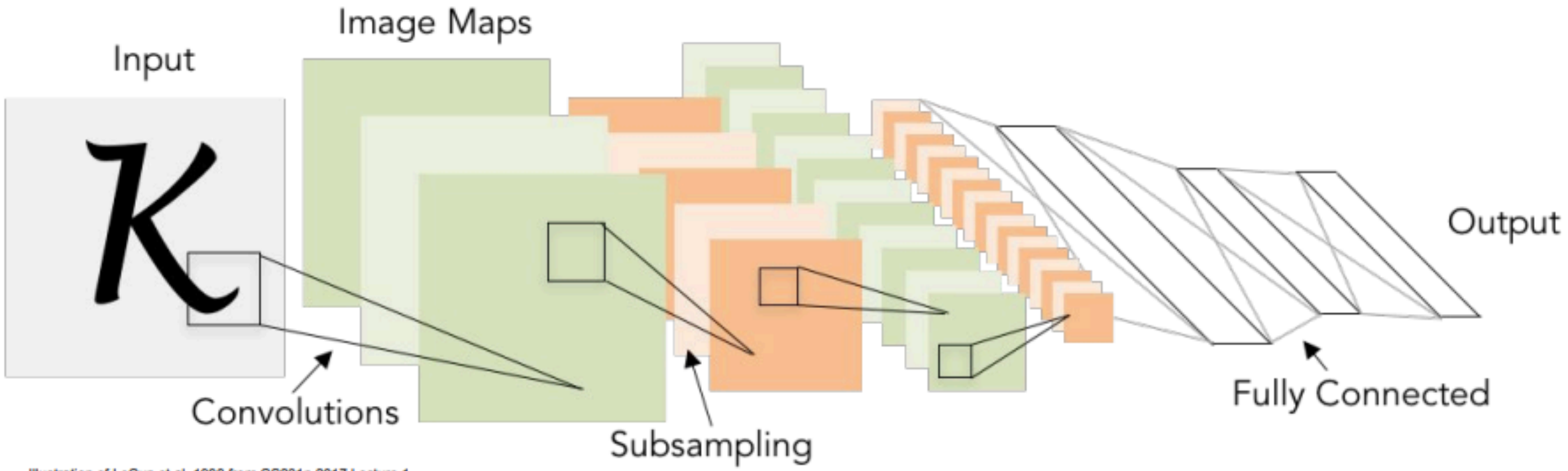
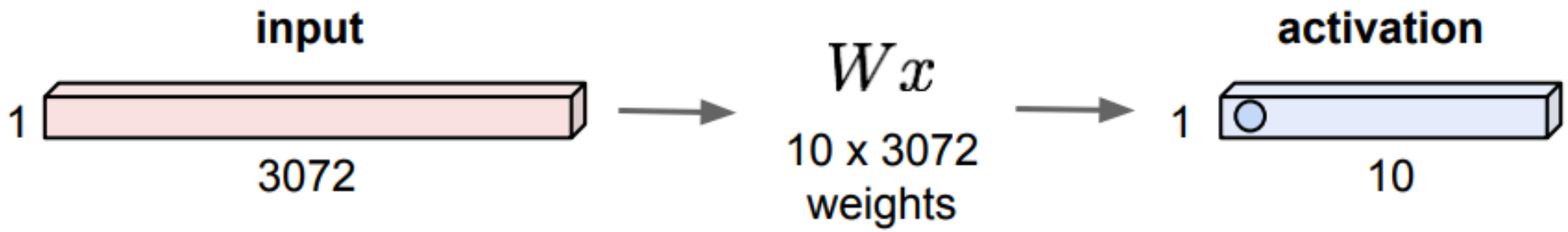


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Convolutional Neural Network

Recap: Fully Connected Layer

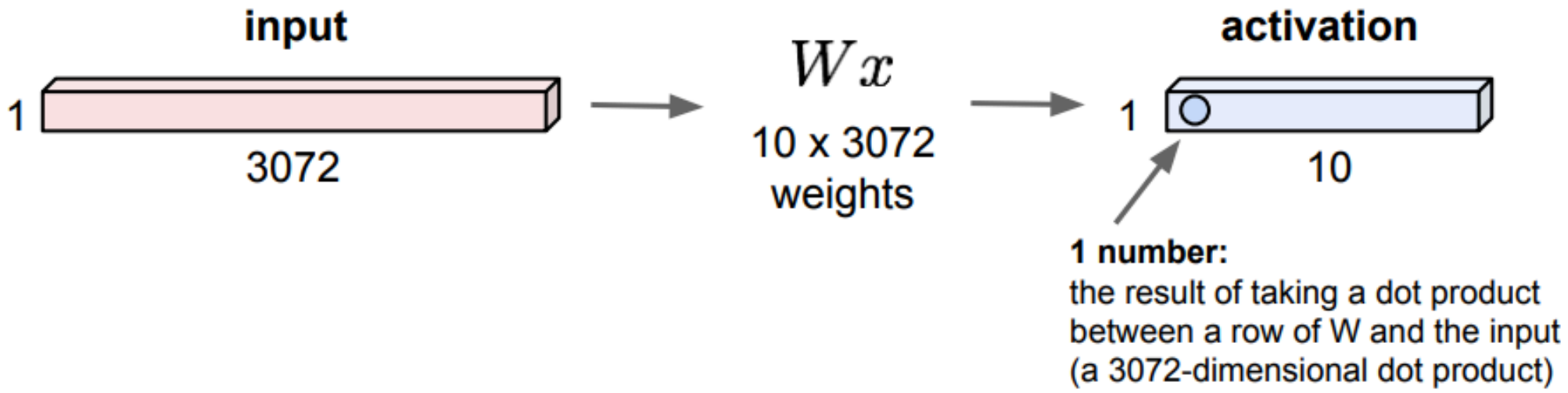
32x32x3 image -> stretch to 3072 x 1



Convolutional Neural Network

Fully Connected Layer

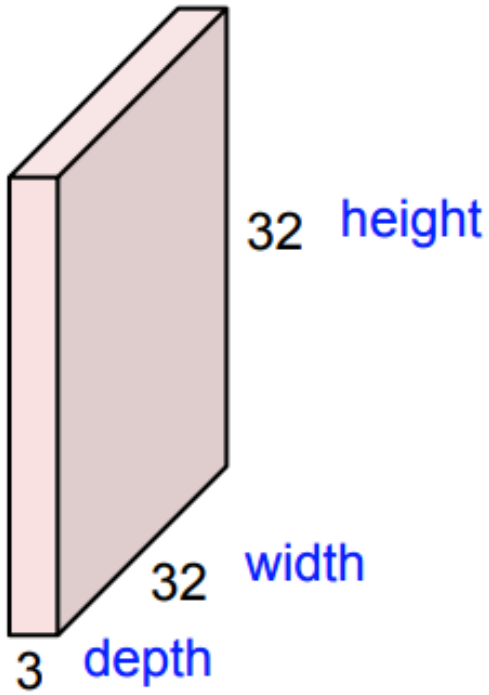
32x32x3 image -> stretch to 3072 x 1



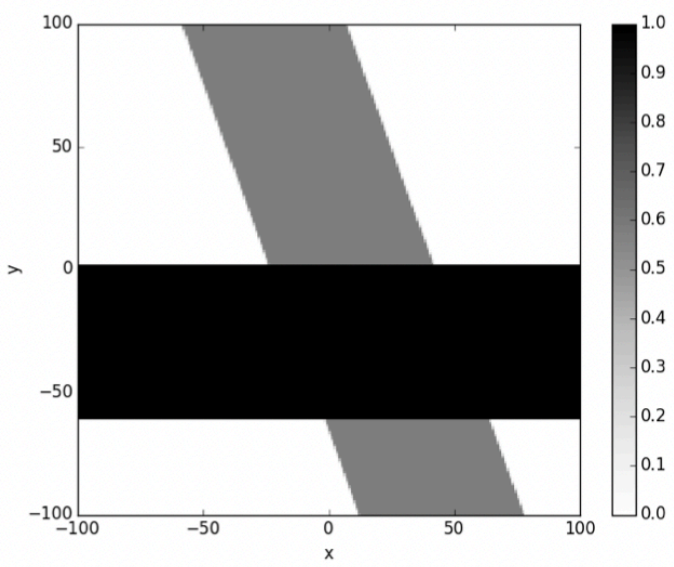
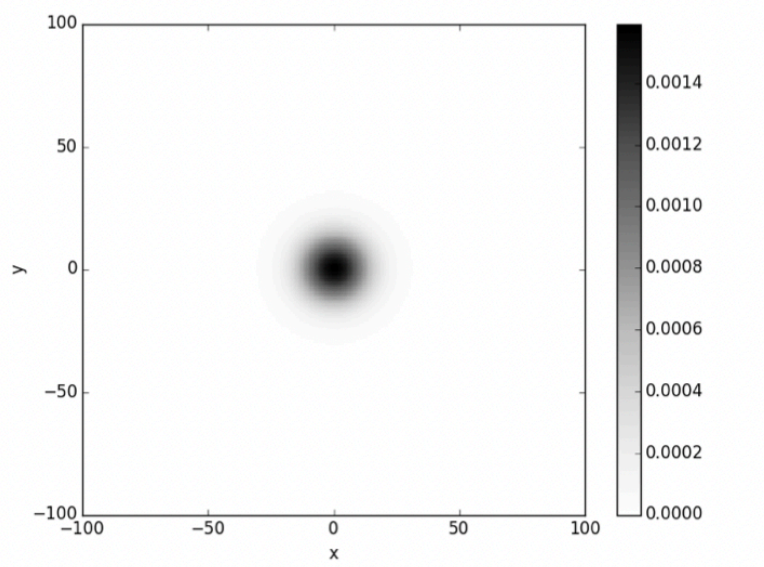
Convolutional Neural Network

Convolution Layer

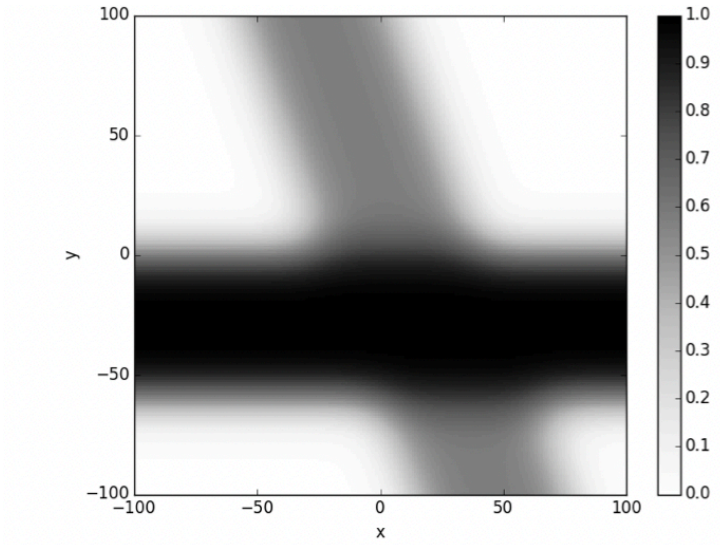
32x32x3 image -> preserve spatial structure



Two-Dimensional Convolution

f  $*$ 

$$g = \frac{1}{2\pi\sigma^2} \exp -\frac{x^2 + y^2}{2\sigma^2}$$

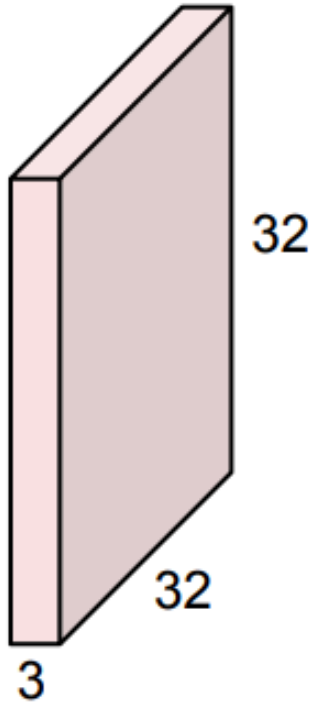
$=$ 

$$(f * g)[m, n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] g[m - k, n - l]$$

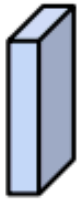
Convolutional Neural Network

Convolution Layer

32x32x3 image



5x5x3 filter



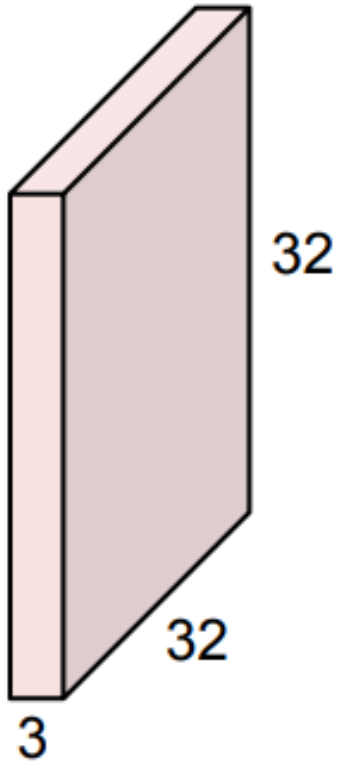
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolutional Neural Network

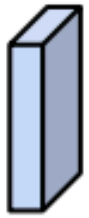
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



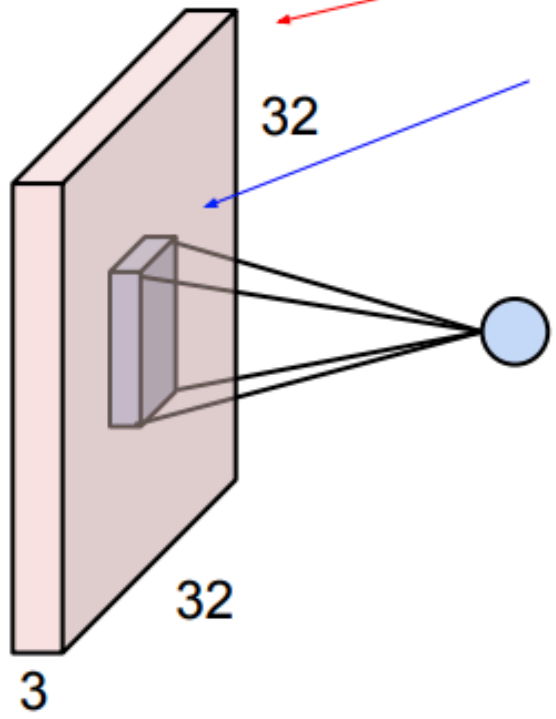
5x5x3 filter



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

Convolutional Neural Network

Convolution Layer



32x32x3 image

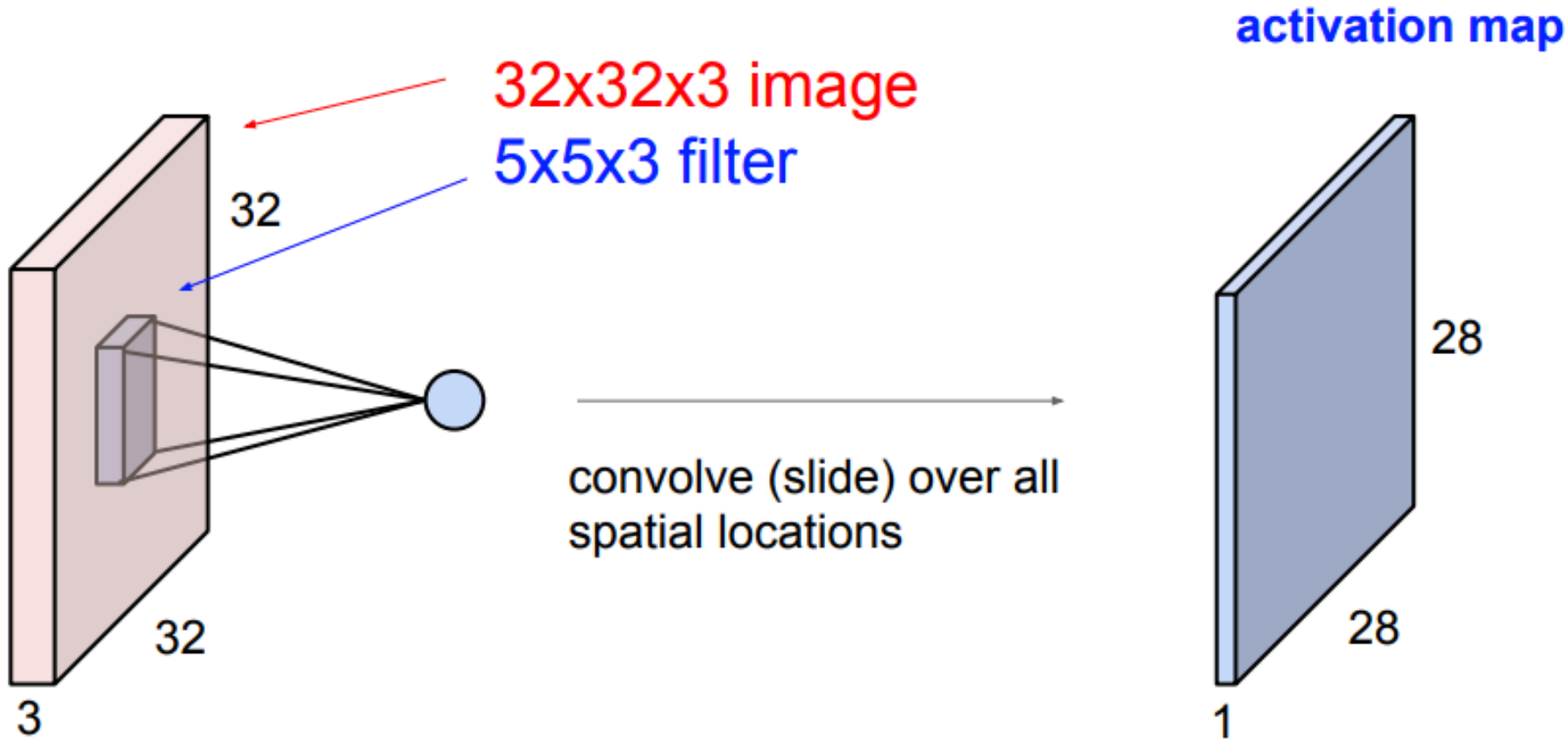
5x5x3 filter w

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

$$w^T x + b$$

Convolutional Neural Network

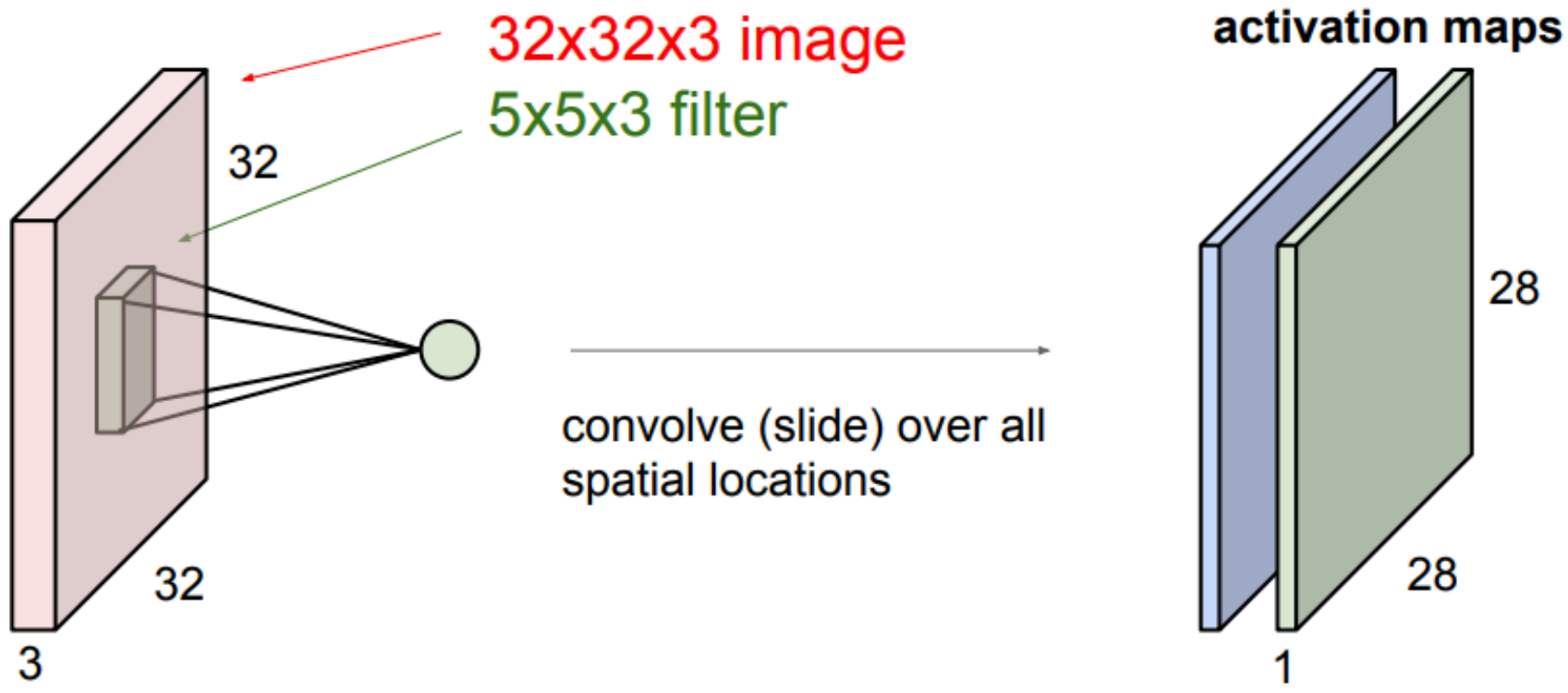
Convolution Layer



Convolutional Neural Network

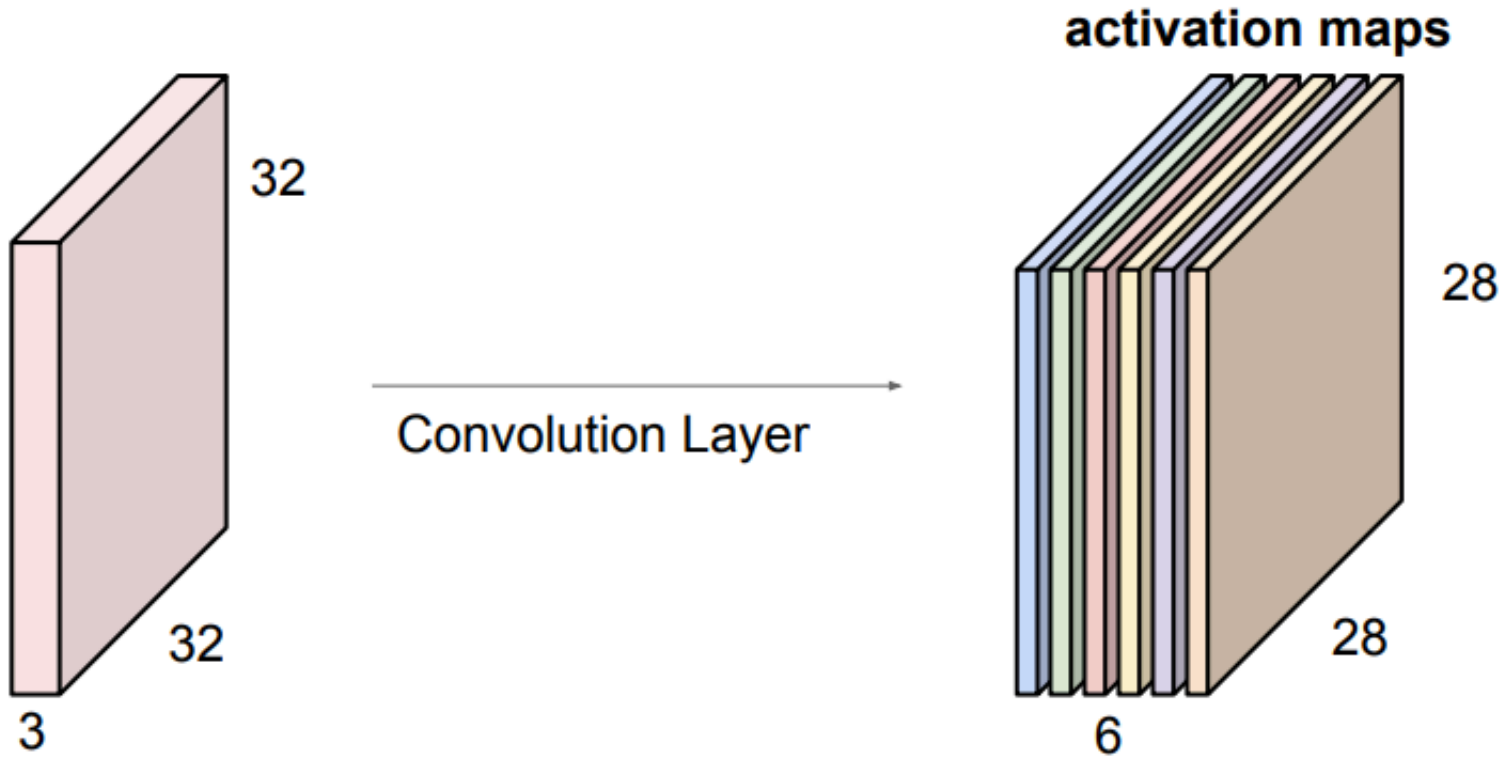
Convolution Layer

consider a second, green filter



Convolutional Neural Network

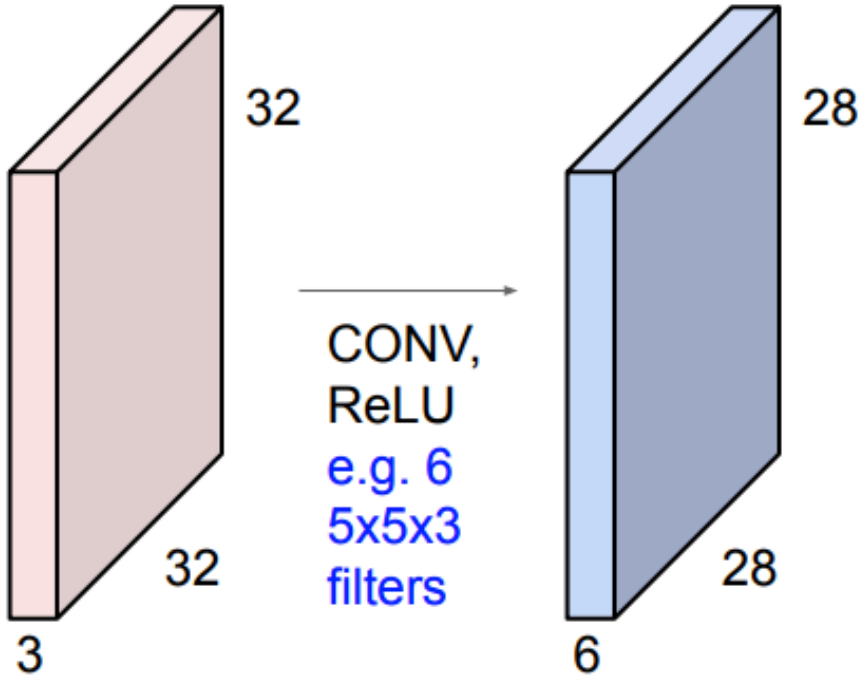
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

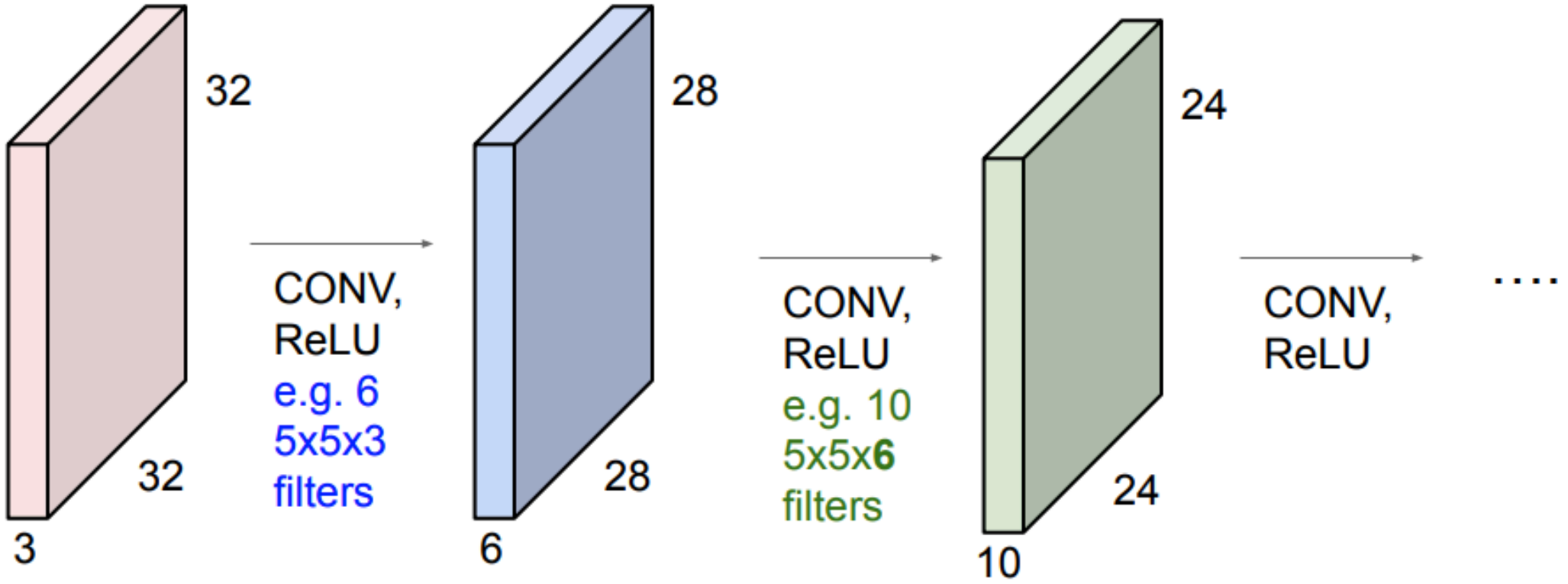
Convolutional Neural Network

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



Convolutional Neural Network

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

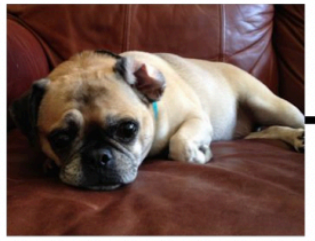


Feature Visualization

Preview

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

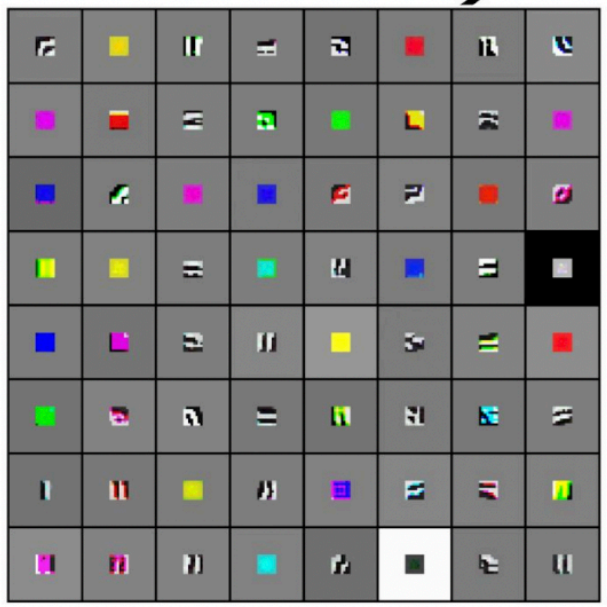


Low-level features

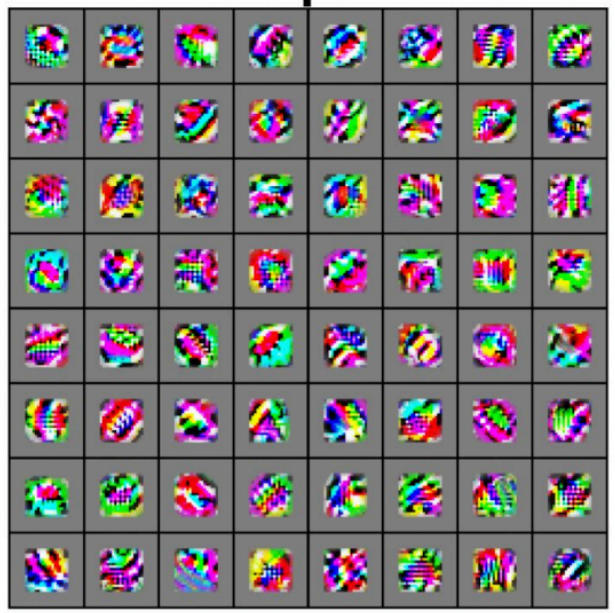
Mid-level features

High-level features

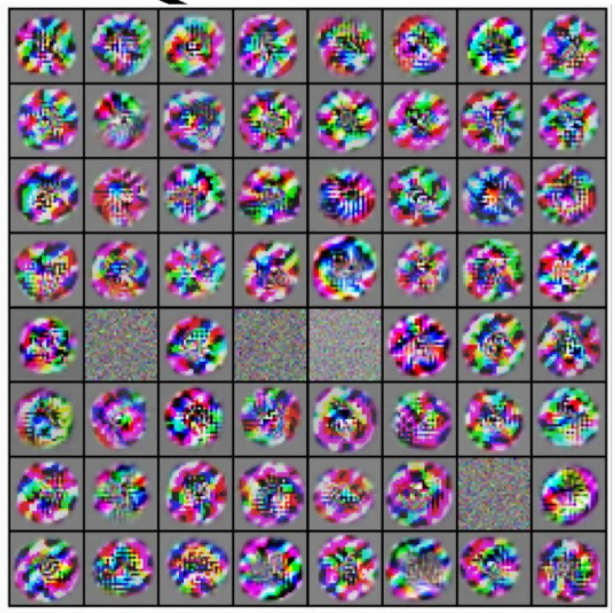
Linearly separable classifier



VGG-16 Conv1_1

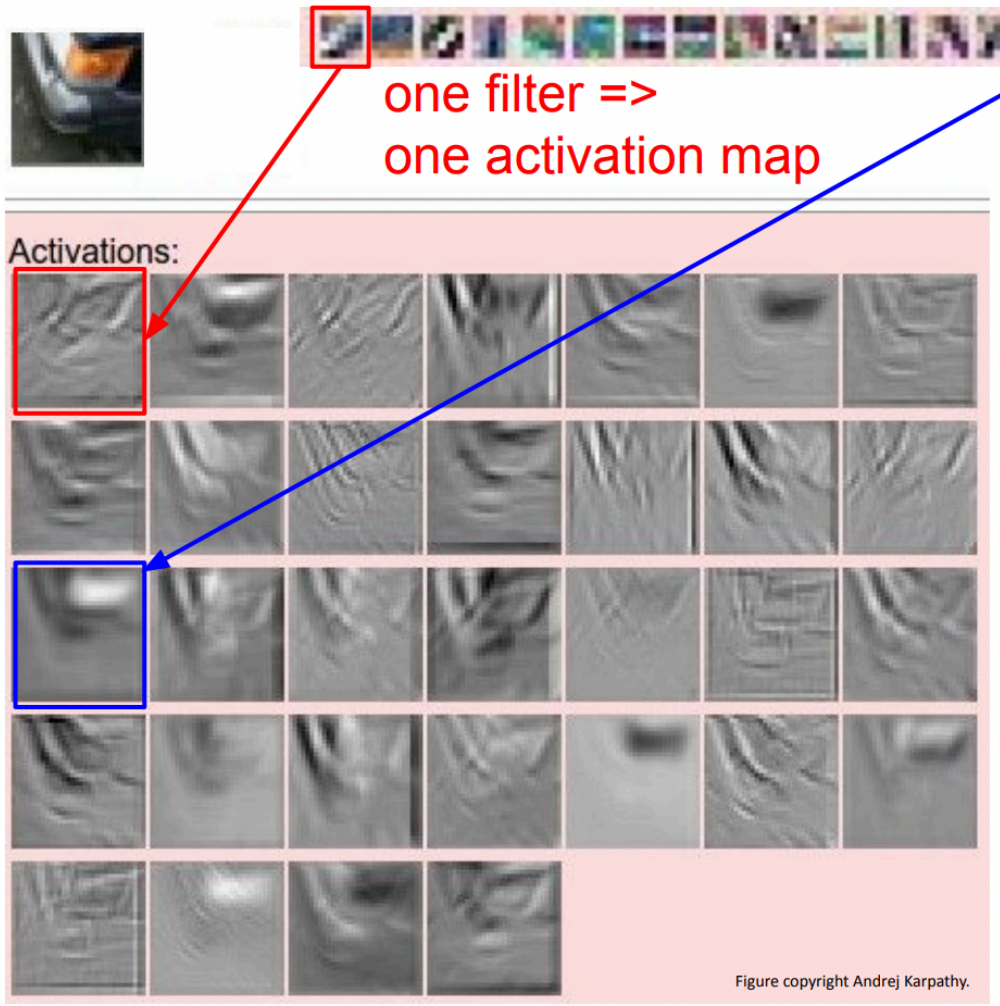


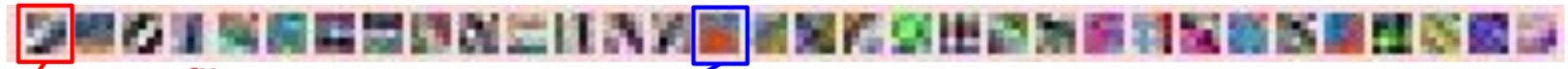
VGG-16 Conv3_2



VGG-16 Conv5_3

Feature Visualization





example 5x5 filters
(32 total)

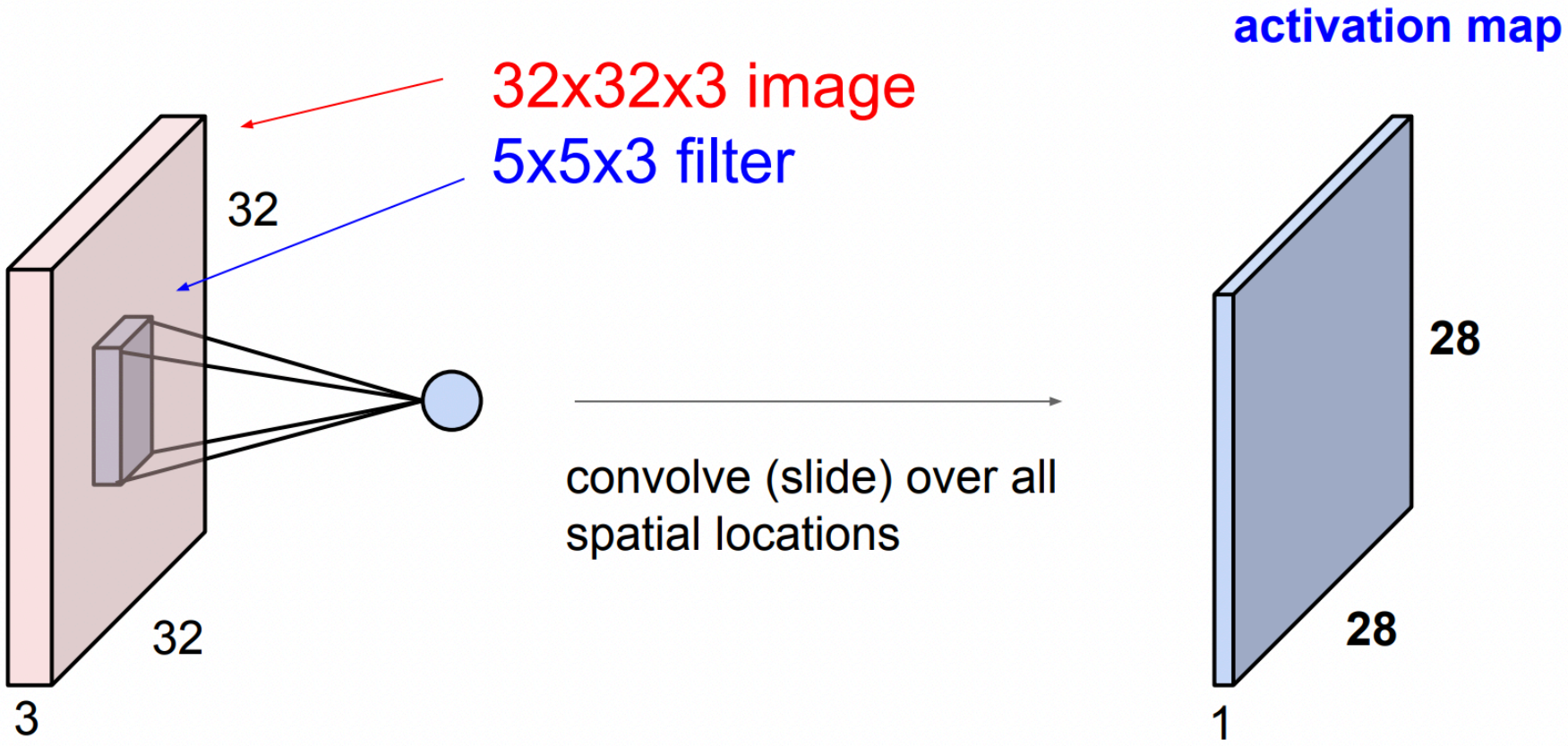
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

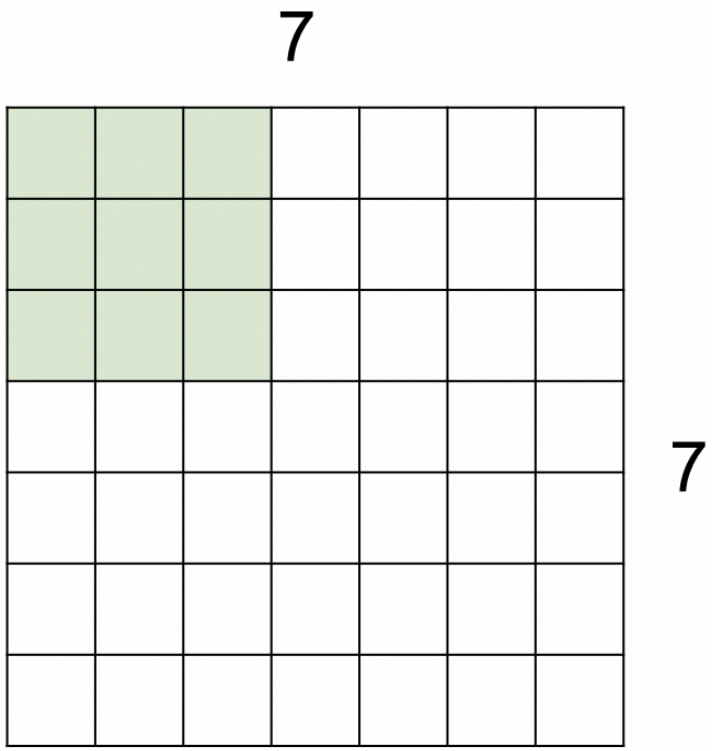
Convolutional Layer

A closer look at spatial dimensions:



Convolutional Layer

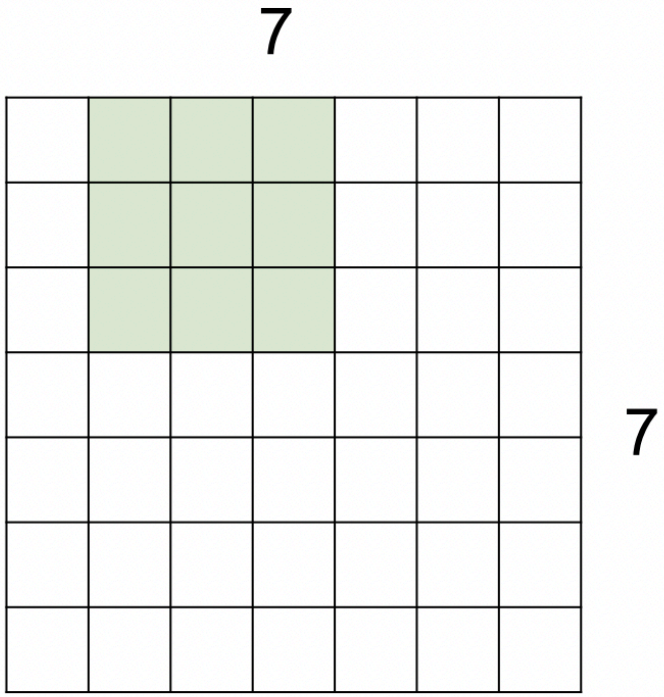
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolutional Layer

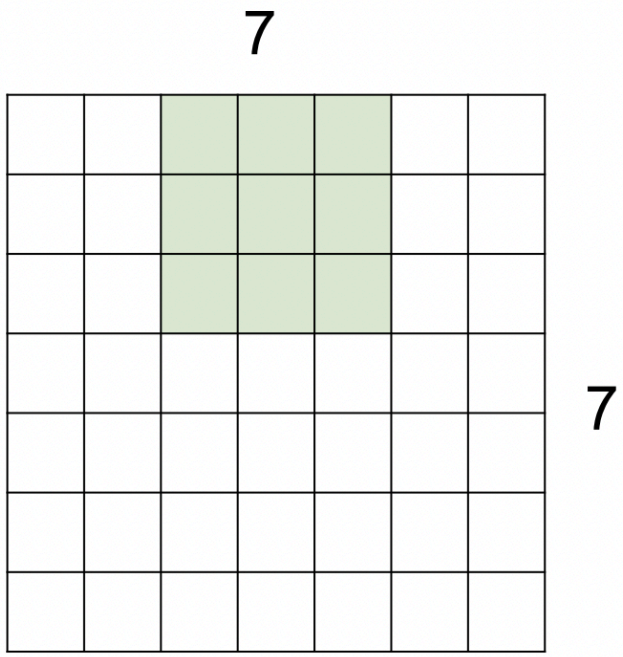
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolutional Layer

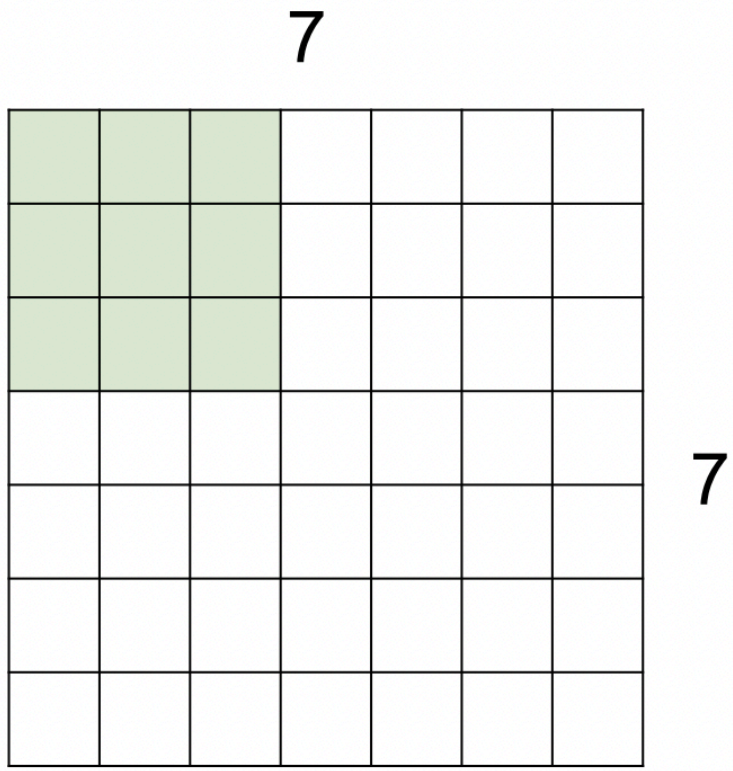
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolutional Layer

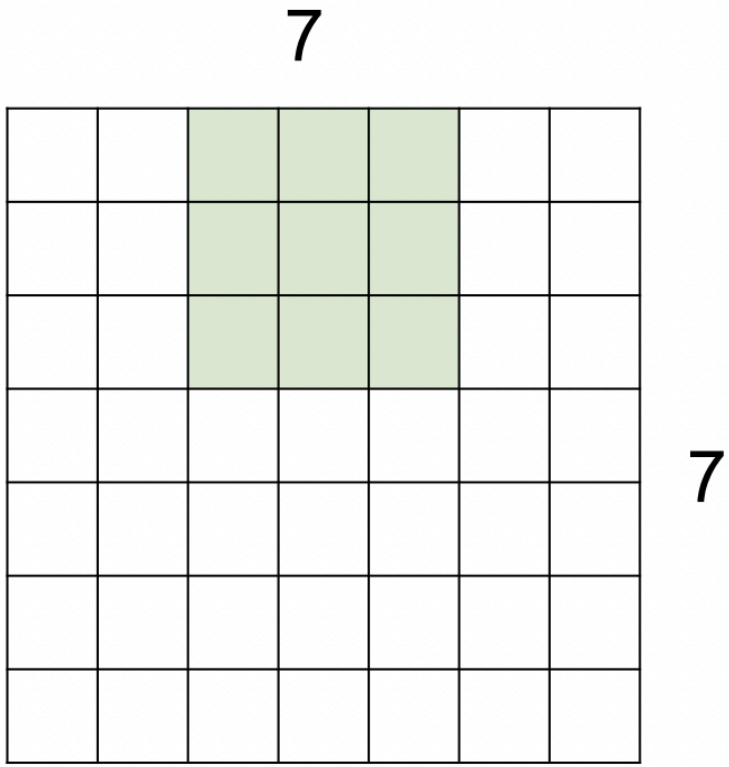
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolutional Layer

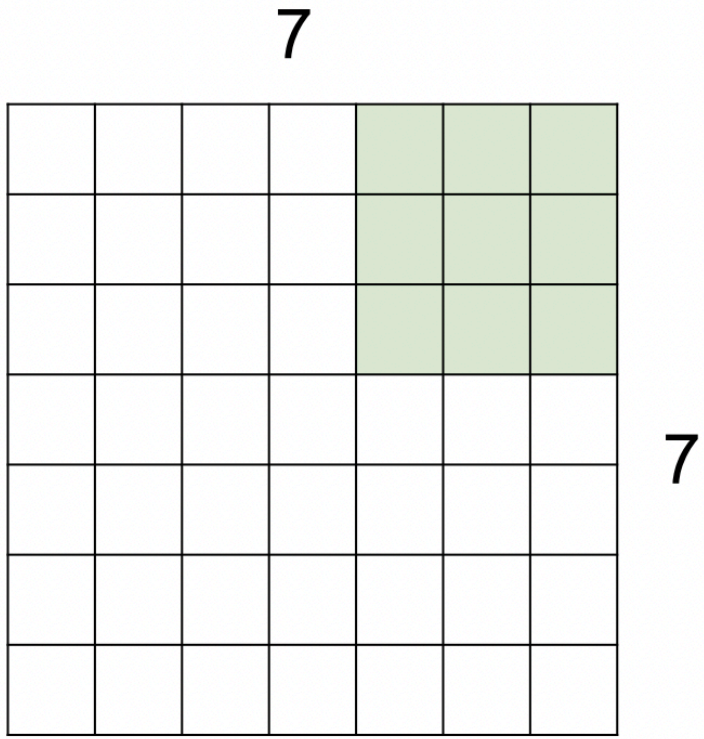
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Convolutional Layer

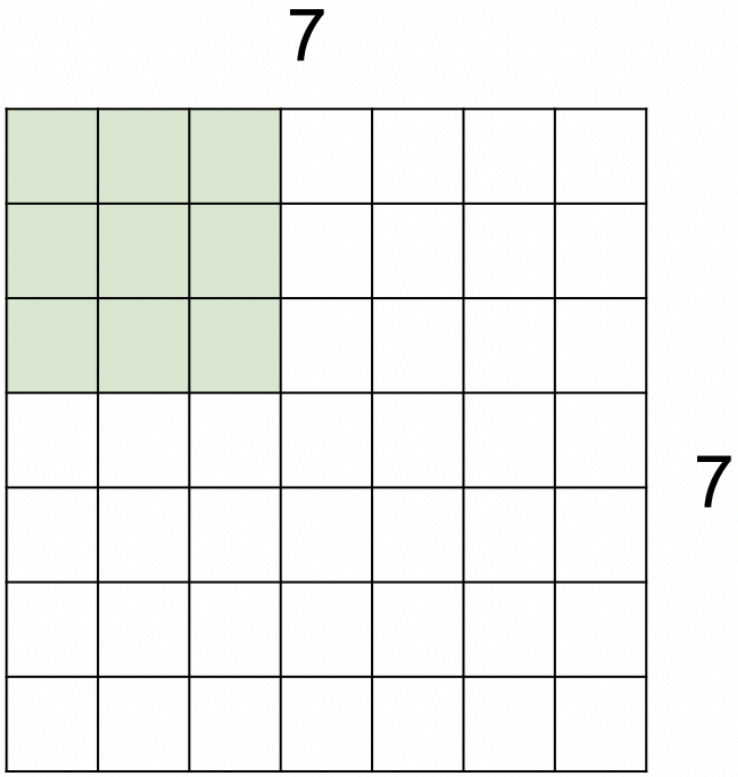
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Convolutional Layer

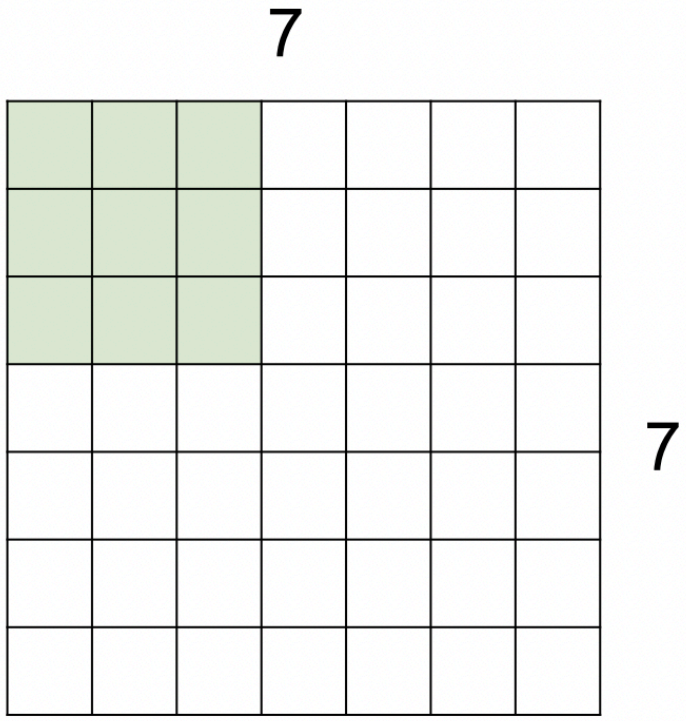
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

Convolutional Layer

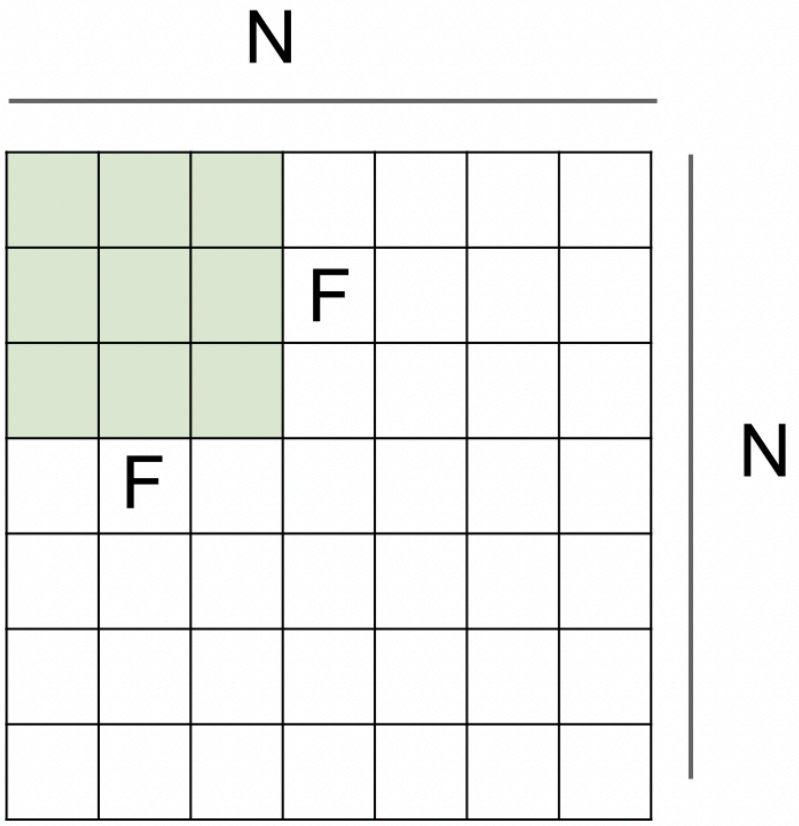
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Convolutional Layer

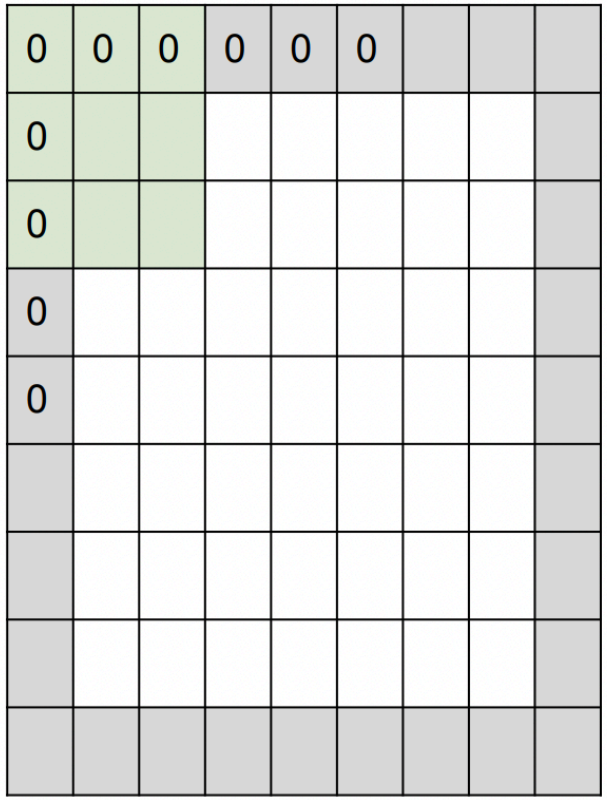


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Padding

In practice: Common to zero pad the border

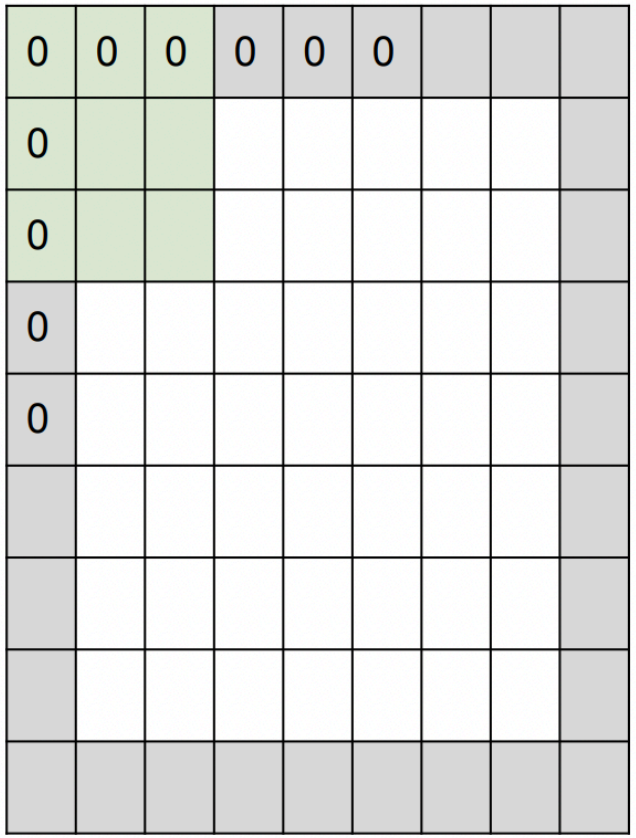


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Padding

In practice: Common to zero pad the border



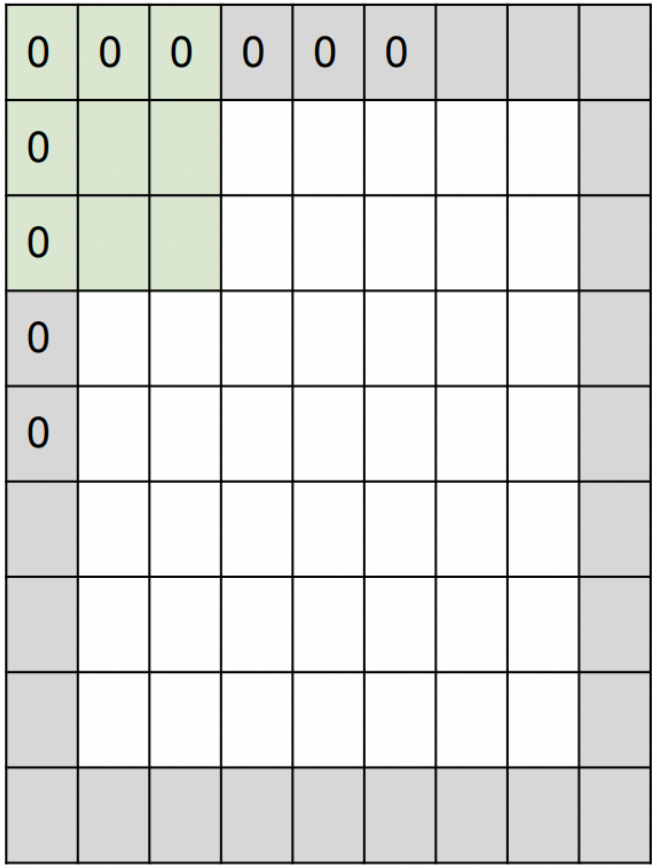
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

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

Padding

In practice: Common to zero pad the border

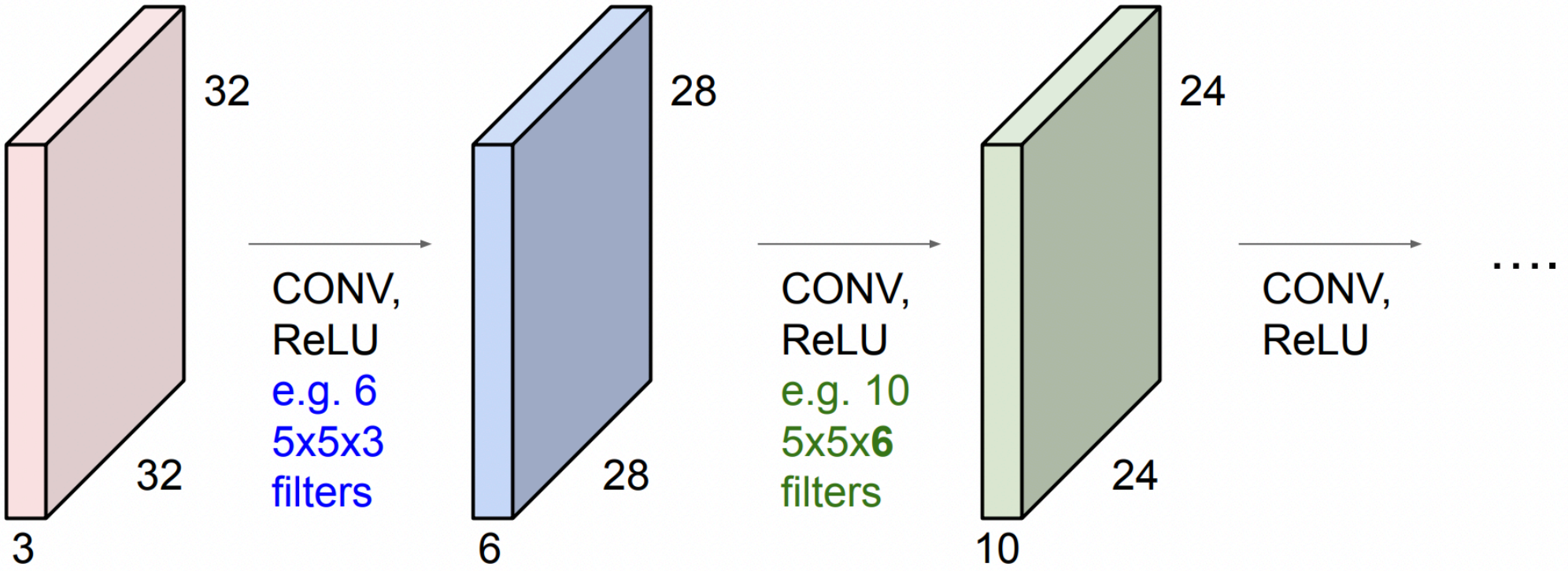


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Convolutional Layer

Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

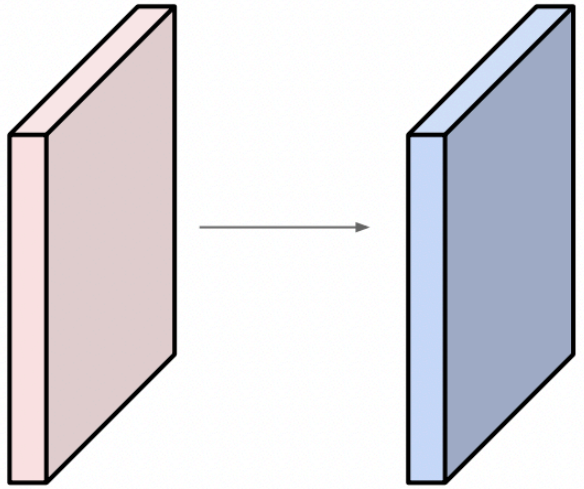


Convolutional Layer

Examples time:

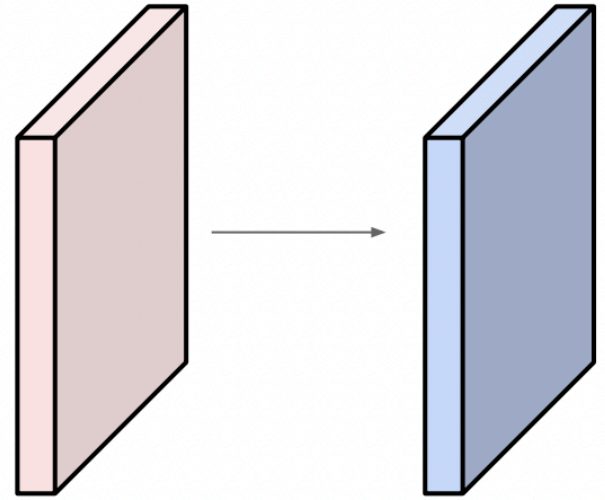
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



Convolutional Layer

Examples time:



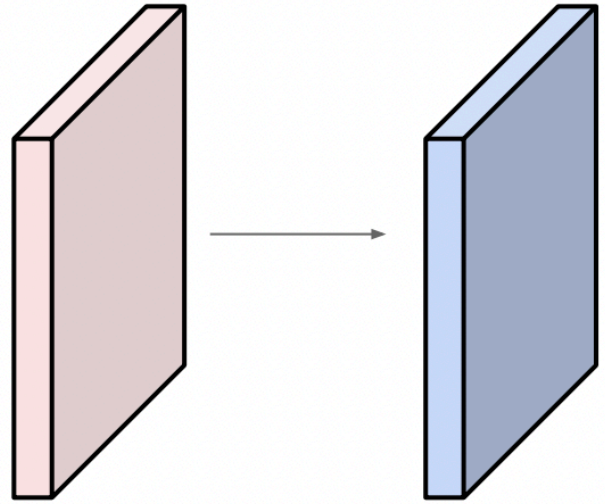
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Convolutional Layer

Examples time:

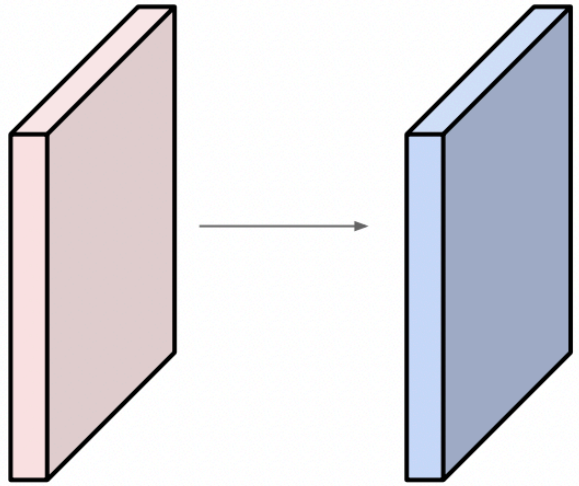
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

Convolutional Layer

Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Convolutional Layer

Convolution layer: summary

Let's assume input is W_1 x H_1 x C
Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of W_2 x H_2 x K
where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

Convolutional Layer

Convolution layer: summary

Let's assume input is W_1 x H_1 x C
Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of W_2 x H_2 x K
where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

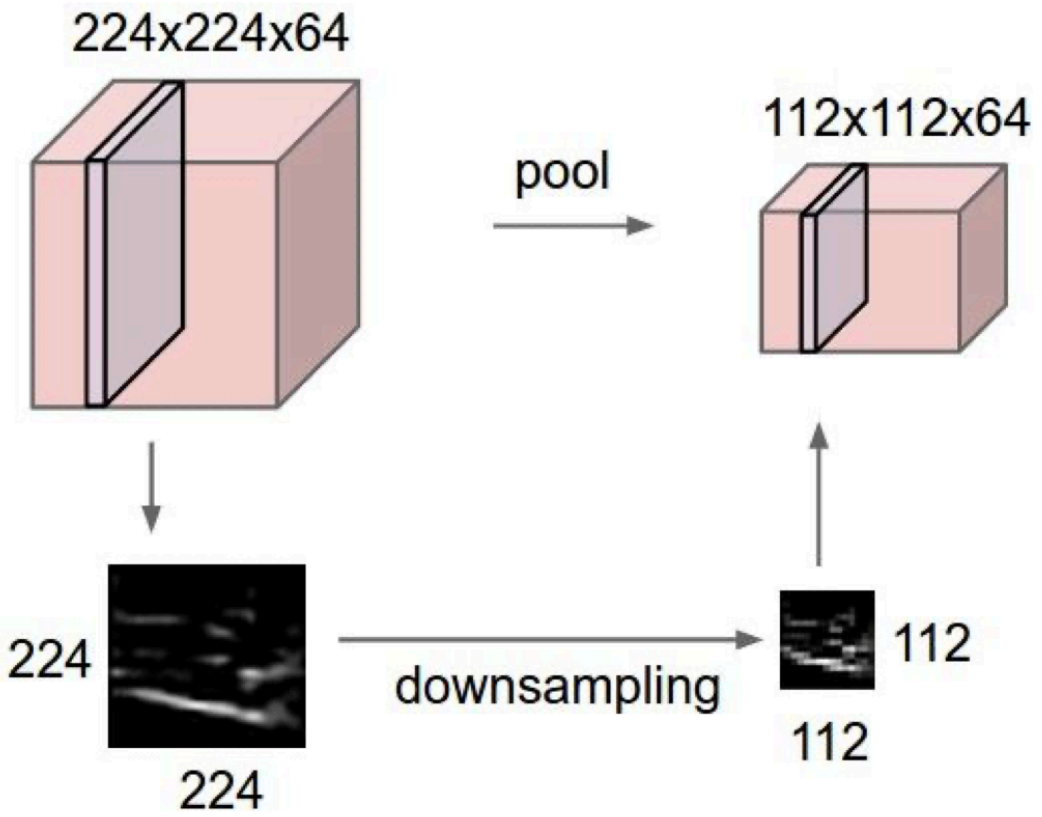
Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

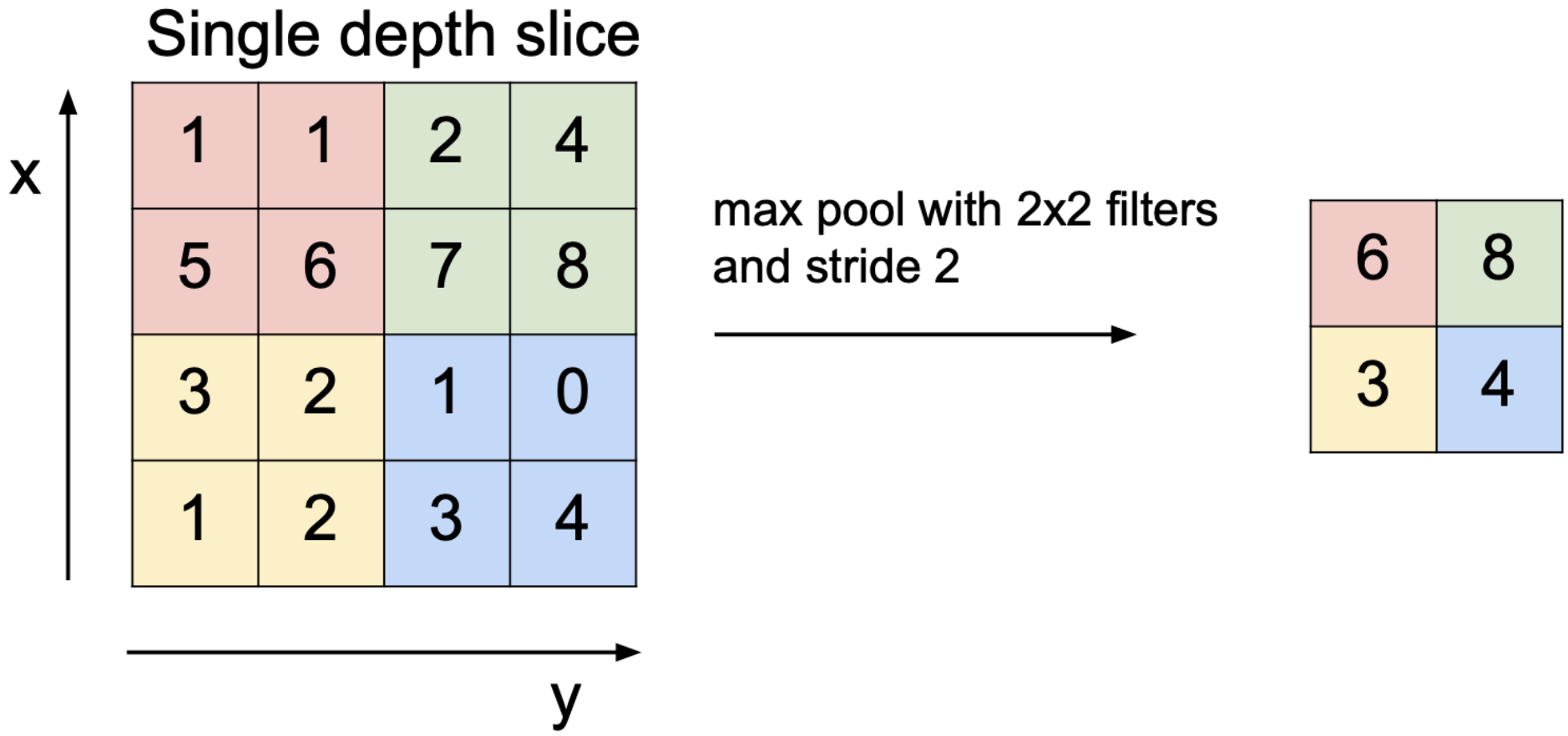
Pooling

- makes the representations smaller and more manageable
- operates over each activation map independently:



Slide credit: Stanford CS231N

Max Pooling



Slide credit: Stanford CS231N

Pooling

- Average pooling
- Max pooling
- Sum pooling (rarely)

Pooling Layer: Summary

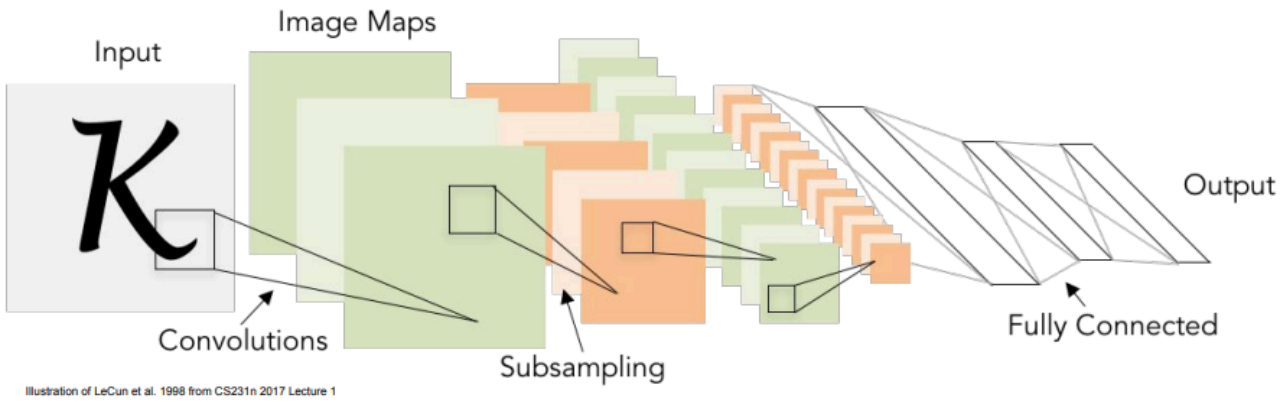
- Let's assume input is $W_1 \times H_1 \times C$
- 2×2 pooling will result in
  - $W_2 = \lceil W_1/2 \rceil$
  - $H_2 = \lceil H_2/2 \rceil$
- Number of parameters: 0

Summary of CNN-based Classification Network

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like

  [(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX

  where N is usually up to ~5, M is large, 0 <= K <= 2.
  - but recent advances such as ResNet/GoogLeNet have challenged this paradigm

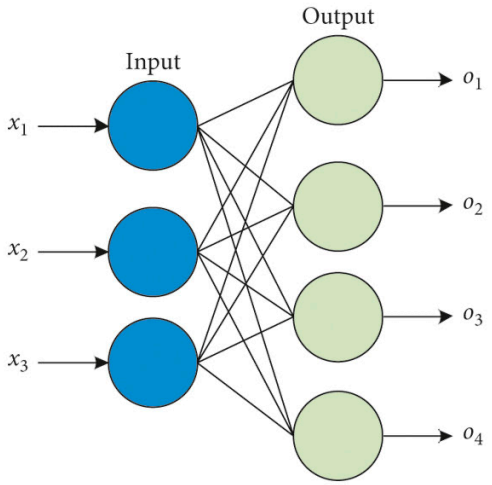


Slides are borrowed from Stanford CS231N.

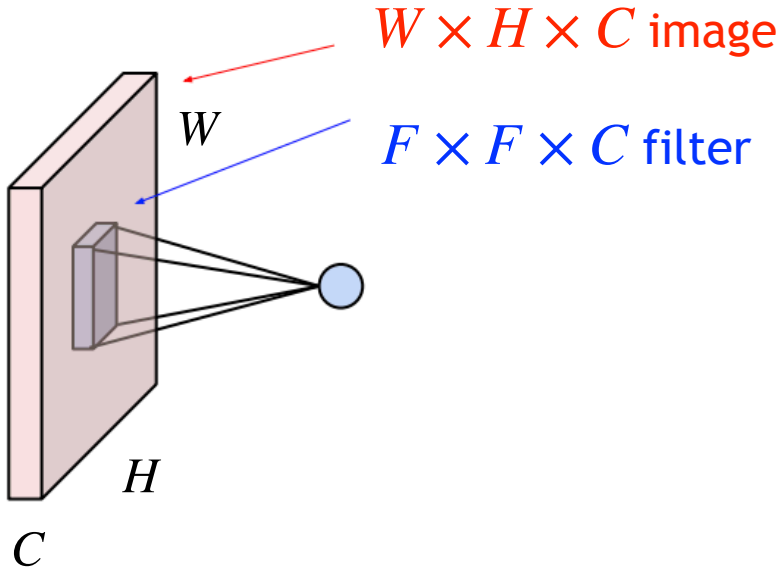
A Comparison between FC and Convolution Layers

- Input size: $W_1 \times H_1 \times C$, output size: $W_2 \times H_2 \times K$

- Fully connected layer (FC)
  - Densely connected
  - Parameters: $W_1 W_2 H_1 H_2 C K$



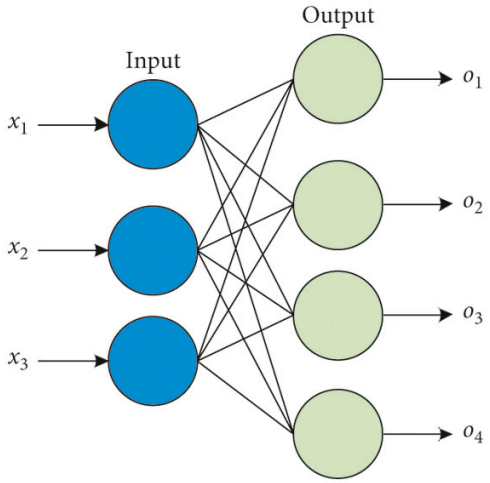
- Convolution layer
  - Filter size: F
  - Parameters: $F^2 C K$



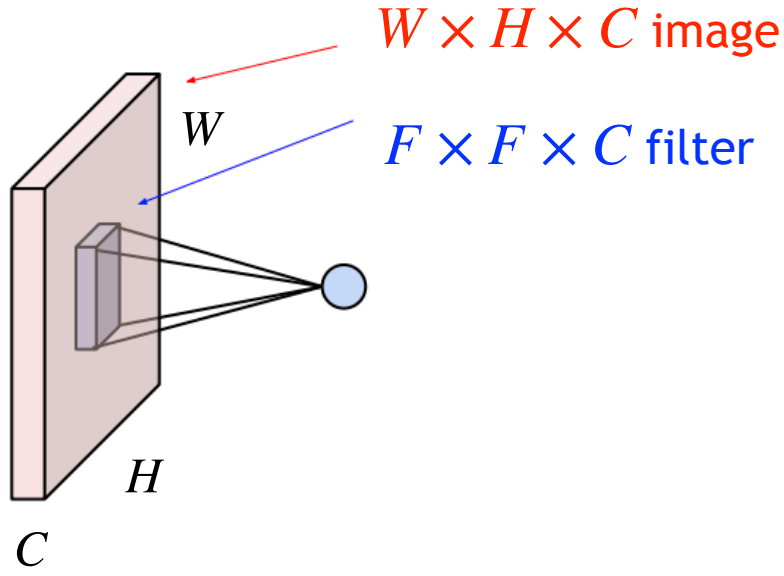
A Comparison between FC and Convolution Layers

- Input size: $W_1 \times H_1 \times C$, output size: $W_2 \times H_2 \times K$

- Fully connected layer (FC)
  - Densely connected
  - Parameters: $W_1 W_2 H_1 H_2 C K$



- Convolution layer
  - Filter size: F
  - Parameters: $F^2 CK$



If $W_1 = W_2 = H_1 = H_2 = 100$, $F = C = K = 3$, then $900M$ vs. 81!

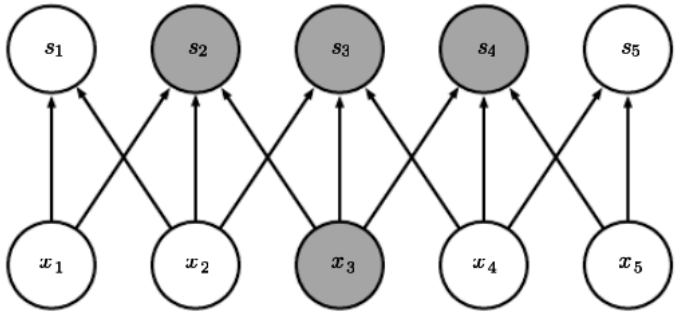
$10{,}000{,}000$ times difference!

Sparse Connectivity

- Convolutional networks, however, typically have sparse interactions (also referred to as sparse connectivity or sparse weights). This is accomplished by making the kernel smaller than the input.

CNN

1D Conv with filter size=3



FC

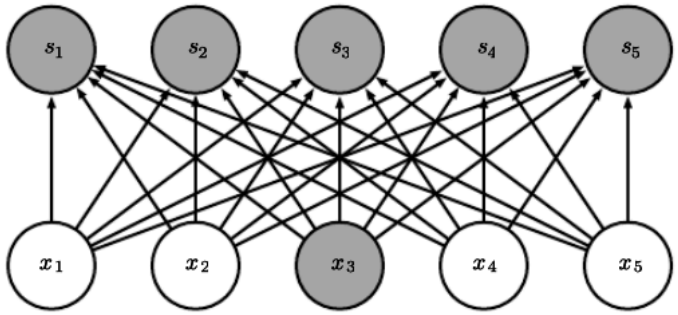


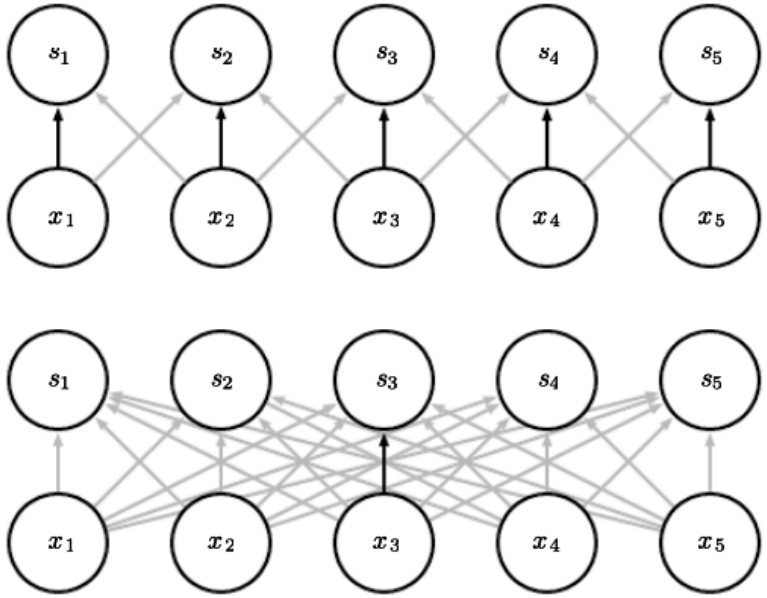
Figure from Deep learning (MIT press, 2016).

Parameter Sharing

- refers to using the same parameter for more than one function in a model.

CNN

FC



Parameter sharing. Black arrows indicate the connections that use a particular parameter in two different models.

Figure from Deep learning (MIT press, 2016).

Why CNN?

- Which is more expressive?
  - CNN
  - FC

Why CNN?

- Which is more expressive?
  - CNN
  - FC is a super set of CNN (without sparse and parameter sharing constraints.)

- What is the problem of FC/MLP?

Why CNN?

- Which is more expressive?
  - CNN
  - FC is a super set of CNN (without sparse and parameter sharing constraints.)

- What is the problem of FC/MLP?
  - FC needs too many parameters.
  - What else?

Image Classification

- Definition: image classification is to categorize an image into several known classes (N).
- Image classification is very important for semantic understanding.

Data

Model

Output: which class

$x =$ 

$y \in \{0,1\}$

$f($  $;\theta)$

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

cat

dog

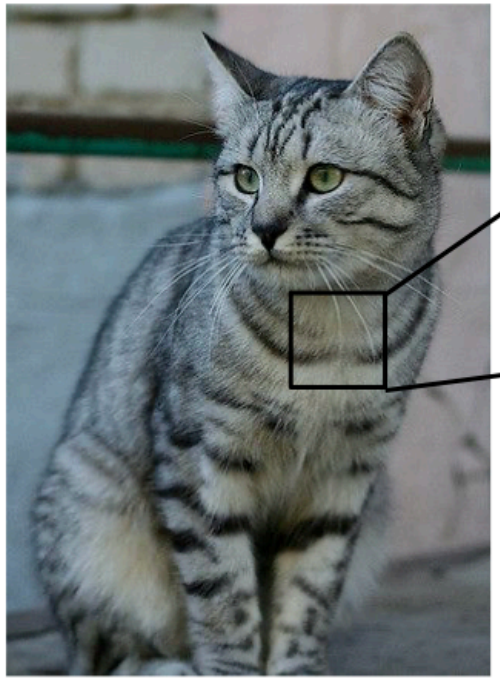
panda

elephant

Parameters or weights

Challenges

The Problem: Semantic Gap



This image by Nikita is licensed under CC-BY 2.0

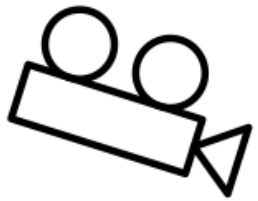
```
[[105 112 108 111 104  99 106  99  96 103 112 119 104  97  93  87]
 [ 91  98 102 106 104  79  98 103  99 105 123 136 110 105  94  85]
 [ 76  85  90 105 128 105  87  96  95  99 115 112 106 103  99  85]
 [ 99  81  81  93 120 131 127 100  95  98 102  99  96  93 101  94]
 [106  91  61  64  69  91  88  85 101 107 109  98  75  84  96  95]
 [114 108  85  55  55  69  64  54  64  87 112 129  98  74  84  91]
 [133 137 147 103  65  81  80  65  52  54  74  84 102  93  85  82]
 [128 137 144 140 109  95  86  70  62  65  63  63  60  73  86 101]
 [125 133 148 137 119 121 117  94  65  79  80  65  54  64  72  98]
 [127 125 131 147 133 127 126 131 111  96  89  75  61  64  72  84]
 [115 114 109 123 150 148 131 118 113 109 100  92  74  65  72  78]
 [ 89  93  90  97 108 147 131 118 113 114 113 109 106  95  77  80]
 [ 63  77  86  81  77  79 102 123 117 115 117 125 125 130 115  87]
 [ 62  65  82  89  78  71  80 101 124 126 119 101 107 114 131 119]
 [ 63  65  75  88  89  71  62  81 120 138 135 105  81  98 110 118]
 [ 87  65  71  87 106  95  69  45  76 130 126 107  92  94 105 112]
 [118  97  82  86 117 123 116  66  41  51  95  93  89  95 102 107]
 [164 146 112  80  82 120 124 104  76  48  45  66  88 101 102 109]
 [157 170 157 120  93  86 114 132 112  97  69  55  70  82  99  94]
 [130 128 134 161 139 100 109 118 121 134 114  87  65  53  69  86]
 [128 112  96 117 150 144 120 115 104 107 102  93  87  81  72  79]
 [123 107  96  86  83 112 153 149 122 109 104  75  80 107 112  99]
 [122 121 102  80  82  86  94 117 145 148 153 102  58  78  92 107]
 [122 164 148 103  71  56  78  83  93 103 119 139 102  61  69  84]]
```

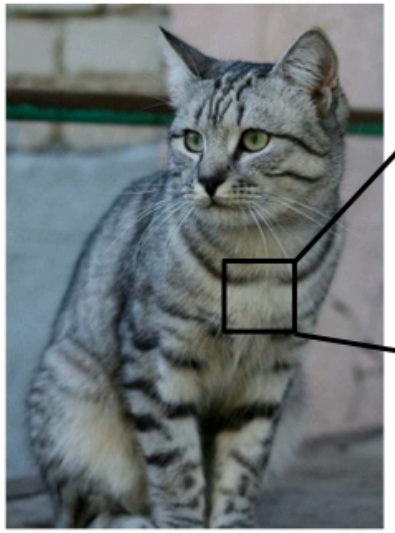
What the computer sees

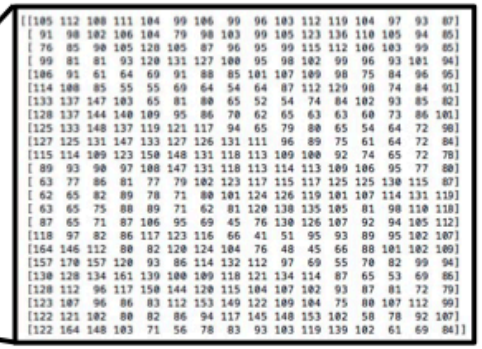
An image is a tensor of integers between [0, 255]:

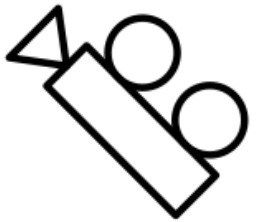
e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: Viewpoint Variation





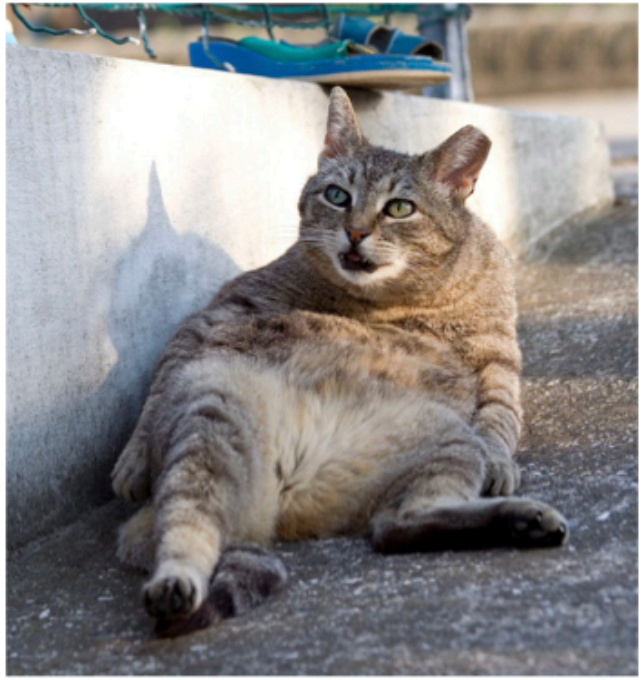




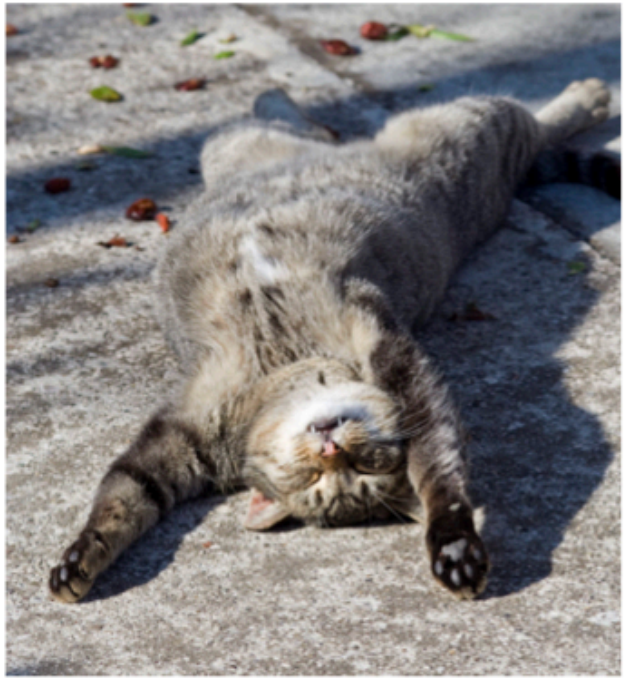
All pixels change when the camera moves!

...e by Nikita is

Challenges: Pose

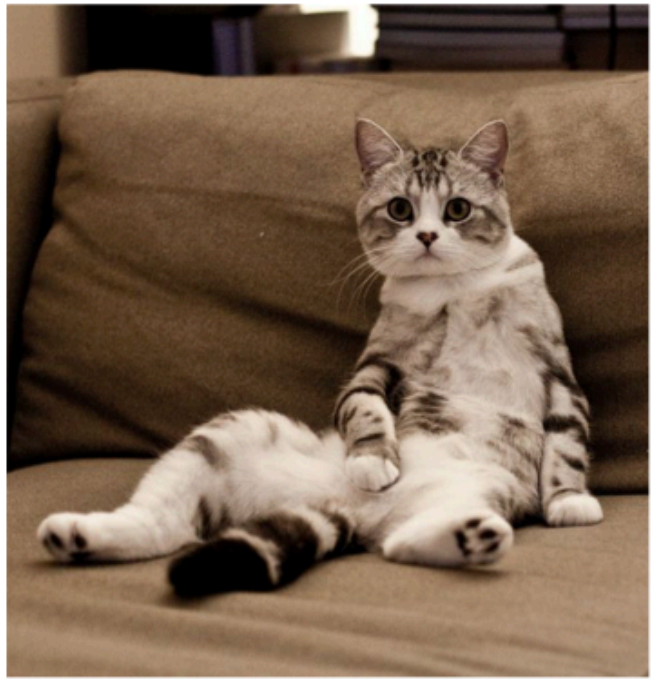


This image by Umberto Salvagnin is licensed under CC-BY 2.0



This image by Umberto Salvagnin is licensed under CC-BY 2.0

This image by sare bear is licensed under CC-BY 2.0



This image by Tom Thai is licensed under CC-BY 2.0

Challenges: Occlusion

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image by jonsson is licensed under CC-BY 2.0

Challenges: Background Variation

This image is CC0 1.0 public domain

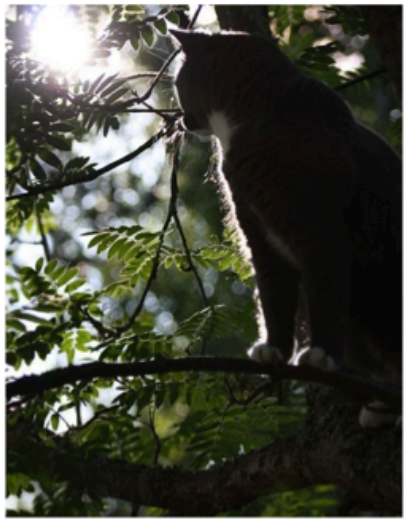
This image is CC0 1.0 public domain

Challenges: Illumination

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain

This image is CC0 1.0 public domain



This image is CC0 1.0 public domain

Challenges: Intraclass Variation

This image is CC0 1.0 public domain

The Requirement of A Good Image Classifier

- A good image classifier should
  - focus on the semantic label
  - and be immune with the following perturbations:
    - change in viewpoint
    - change in pose
    - change in illumination
    - change in background
    - …

The Requirement of A Good Image Classifier

- A good image classifier should
  - focus on the semantic label
  - and be immune with the following perturbations:
    - change in viewpoint
    - change in pose
    - change in illumination
    - change in background
    - …
- Some of these challenges can be handled by learning on big data
- However, what our neural network should behave?

Image Perturbations

Applying a small translation

https://greatergood.berkeley.edu/article/item/the_science_backed_benefits_of_being_a_dog_owner

Image Perturbations

Applying a small rotation

https://greatergood.berkeley.edu/article/item/the_science_backed_benefits_of_being_a_dog_owner

The behavior of Fully Connect Layer

- FC will change dramatically even if you just shift one row or one column. Same happens for a 5° rotation.
- A huge problem during network training or optimization!
- What about CNN?

Equivariance

General definition of equivariance:

$$S_A[\phi(X)] = \phi(T_A(X))$$

Here A is an operation, T_A and S_A are their transformation function in the space of X and $\phi(X)$.

Equivariance

General definition of equivariance:

$$S_A[\phi(X)] = \phi(T_A(X))$$

Here A is an operation, T_A and S_A are their transformation function in the space of X and $\phi(X)$.

Invariance is a special case of equivariance,
When $S_A = I, \phi(X) = \phi(T_A(X))$.

Parameter Sharing = Equivariance with Translation

- Ignoring the values at the boundary, 2D Conv is equivariant with translation.
- We use the same way to process all local regions due to parameter sharing.
- For example, when processing images, it is useful to detect edges in the first layer of a convolutional network. The same edges appear more or less everywhere in the image, so it is practical to share parameters across the entire image.

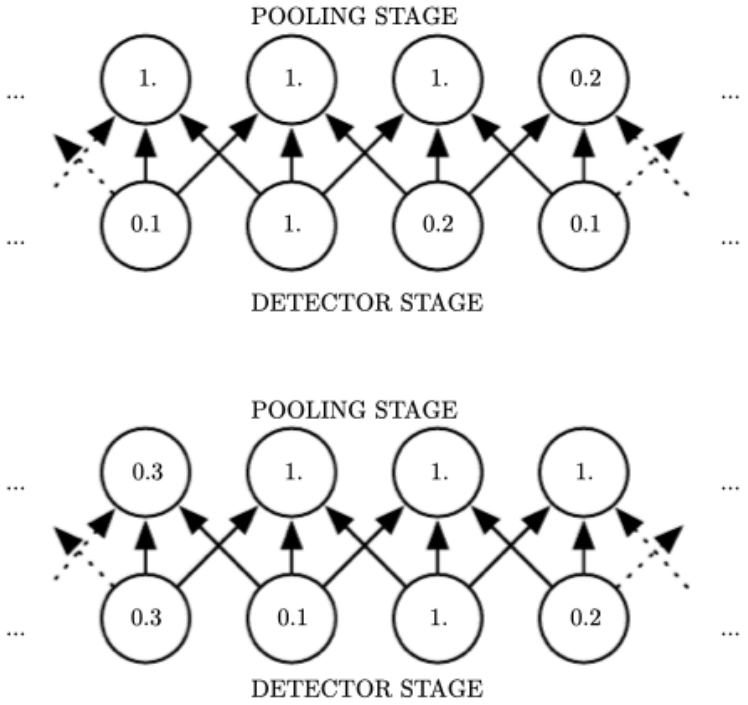
CNN

- Convolution is not naturally equivariant with some other transformations
  - changes in the scale
  - rotation of an image.

- Other mechanisms are necessary for handling these kinds of transformations.

Pooling

- Invariance to local translation can be a useful property if we care more about whether some feature is present than exactly where it is.

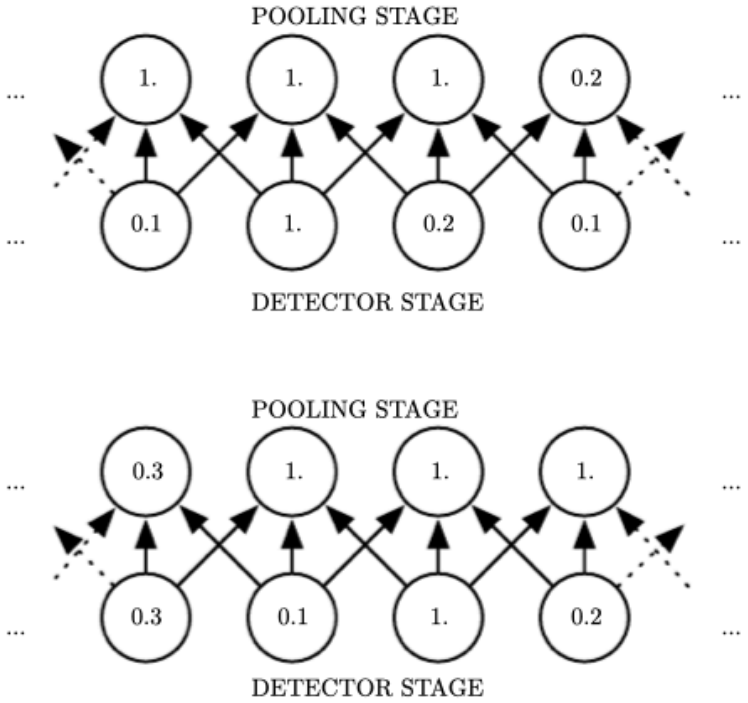


Max pooling introduces invariance.

Figure from Deep learning (MIT press, 2016).

Pooling

- Applying pooling induces invariance to small translations and rotations.



Max pooling introduces invariance.

Figure from Deep learning (MIT press, 2016).

Convolution and Pooling as a Prior

- CNN stacks Conv layer and pooling layer.
- Conv layer + pooling layer is invariant with small translation and rotation.
- Thus CNN is much easier to optimize than a FC for an image.
- Loss landscape:

Next week: Lecture 6,
Deep Learning III