Introduction to Computer Vision

Lecture 4- Deep Learning I

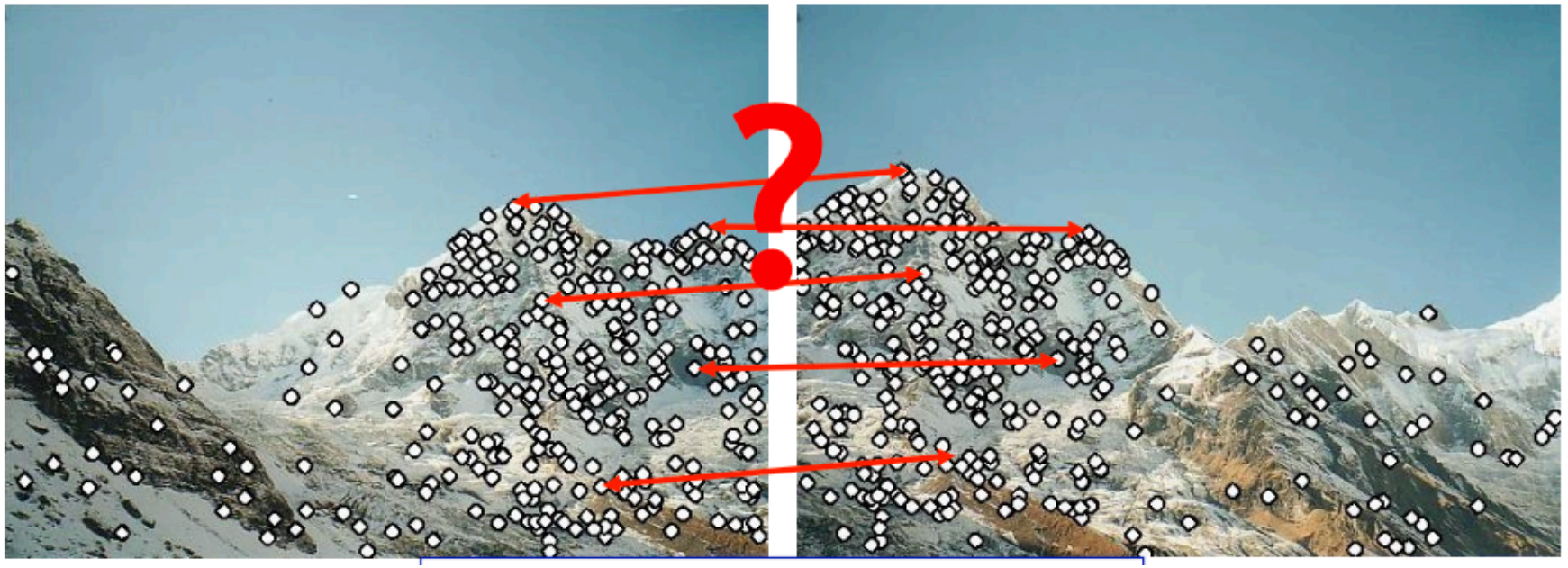
Prof. He Wang

Feature Description

Local Descriptors

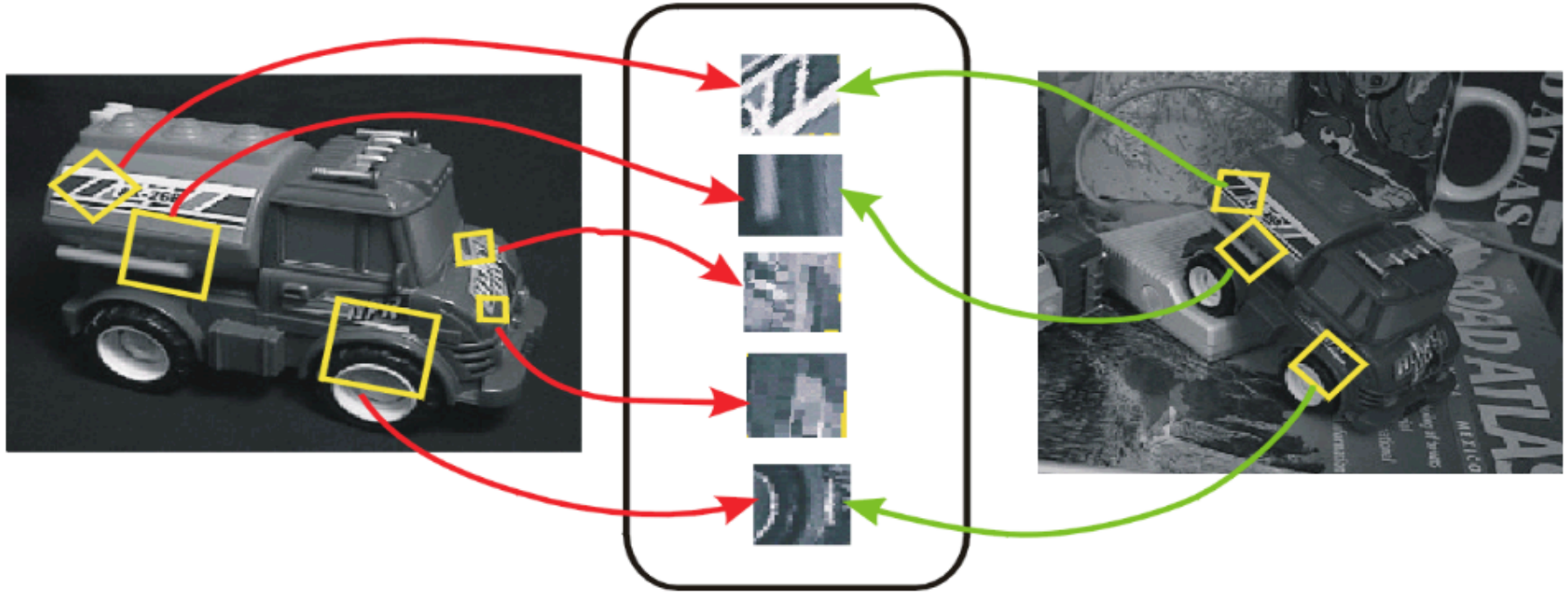
- We know how to detect points
- Next question:

How to *describe* them for matching?



Local Features

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



Feature

- A feature is any piece of information which is relevant for solving the computational task related to a certain application.

Useful features F:

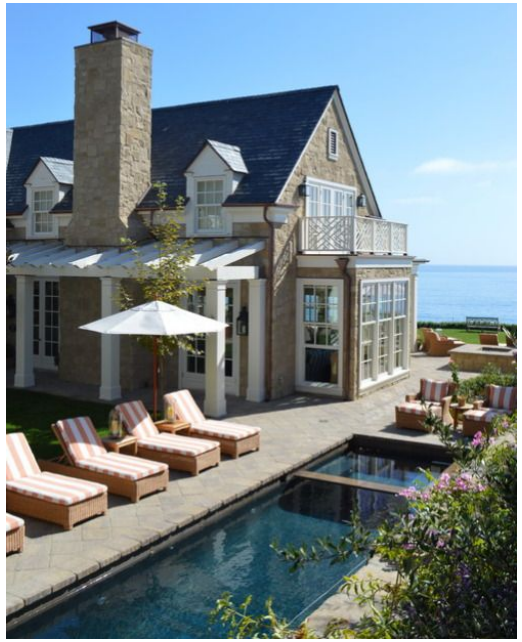
- Location
- Size
- Building time
- Current condition
- House style
- …

How much is this house?

Model

- Based on the features, we can build a model.

- Heuristic model:
  $y = (10 - \text{location}) \times \text{area}$



Useful features F:

- Location
- Size
- Building time
- Current condition
- House style
- …

How much is this house?

Model

- Based on the features, we can build a model.

- Heuristic model:
  $y = (10 - \text{location}) \times \text{area}$

- Parametric model:
  $y = \phi_\theta(F)$
  - when we have some observations, we can fit θ

Useful features F:

- Location
- Size
- Building time
- Current condition
- House style
- …

How much is this house?

Model

- Based on the features, we can build a model.

- Heuristic model:
  $y = (10 - \text{location}) \times \text{area}$

- Parametric model:
  $y = \phi_\theta(F)$

- Deep vision model $y = \phi_\theta(I)$

Useful features F:

- Location
- Size
- Building time
- Current condition
- House style
- …

How much is this house?

Topic Switch

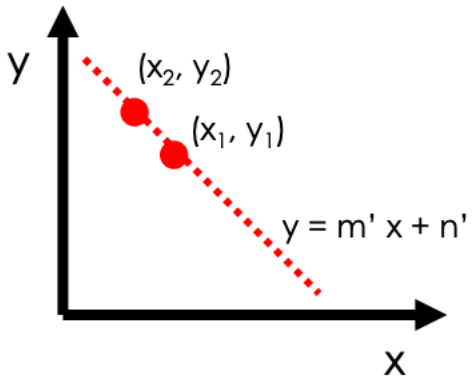
- Low-level vision
  - Image processing
  - Edge/corner detection
  - Feature extraction
- Mid-level Vision
  - Grouping
  - Inferring scene geometry (3D reconstruction)
  - Inferring camera and object motion
- **High-level vision (where deep learning wins!)**
  - Object recognition
  - Scene understanding
  - Activity understanding

Machine Learning 101

From Line Fitting to Neural Network Training

- When we have some observations $\{(x, y)\}$, we want to find the relationship behind y and x.

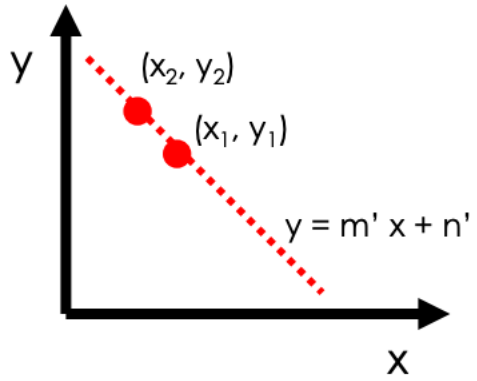
- Line fitting: we know the relationship is a line, so we use $y = mx + b$ to fit (m, n)



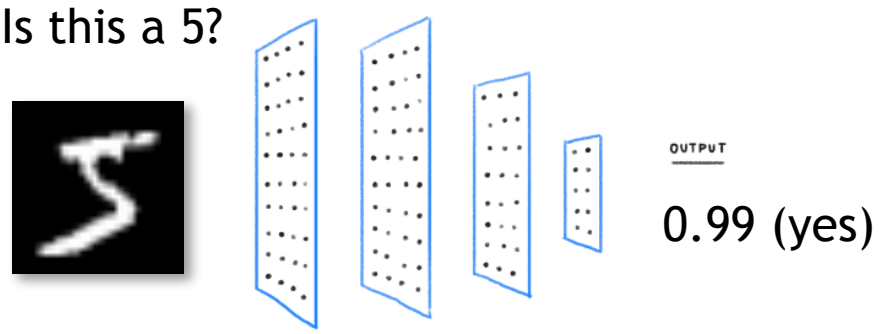
From Line Fitting to Neural Network Training

- When we have some observations $\{(x, y)\}$, we want to find the relationship behind y and x.

- Line fitting: we know the relationship is a line, so we use $y = mx + b$ to fit (m, n).



- Training neural network: similarly, we use a parametric model $y = h_\theta(x)$ to fit, however we usually have less understanding of h_θ.



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

Task: Binary Classification — Is This Digit a 5?

Data

Model

Output

$x =$ 

$h($  $; \theta)$

?

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

Parameters or weights

Yes or No

MINIST Dataset of handwritten digits

Task: Binary Classification — Is This Digit a 5?

Data

Model

Output

$x =$ 

$h($  $; \theta)$

?

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

Parameters or weights

Probability of being a 5

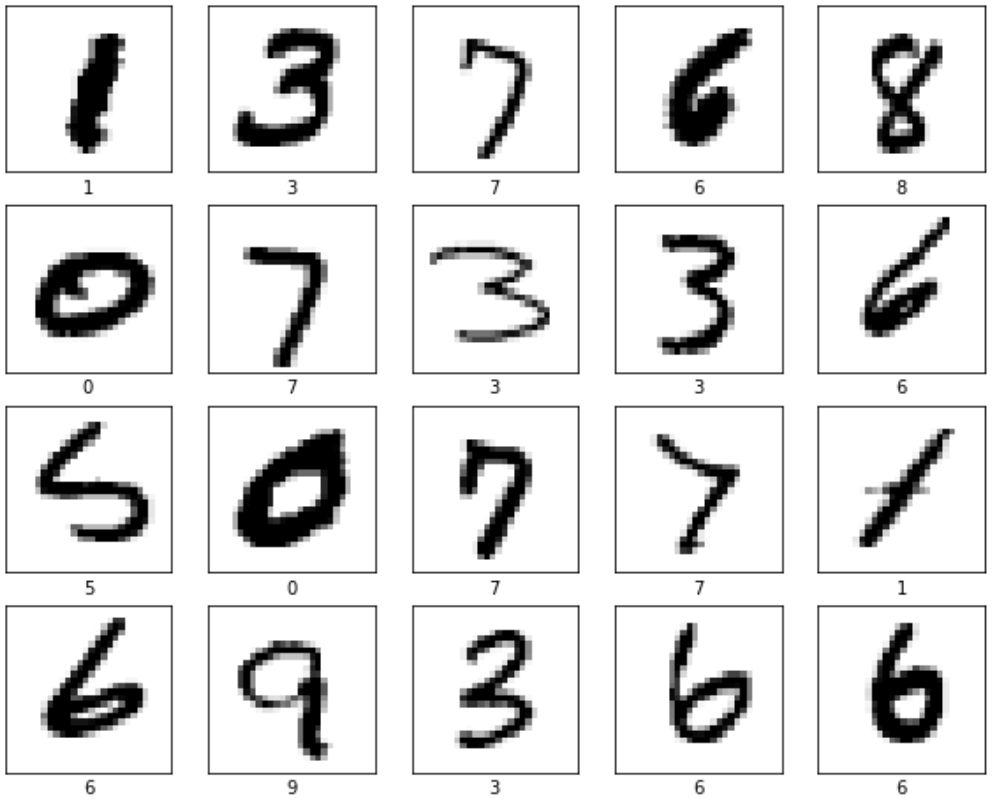
$p(y = 1 \mid x)$

MINIST Dataset of handwritten digits

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

Data



From MNIST dataset

- 70000 images in total
- Basic elements of One Data Point
  - One digit image $x^{(i)}$: 28×28 pixels
  - Paired with a label $y^{(i)} \in \{0,1\}$
- Training data X, labels Y

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

Model: Logistic Regression

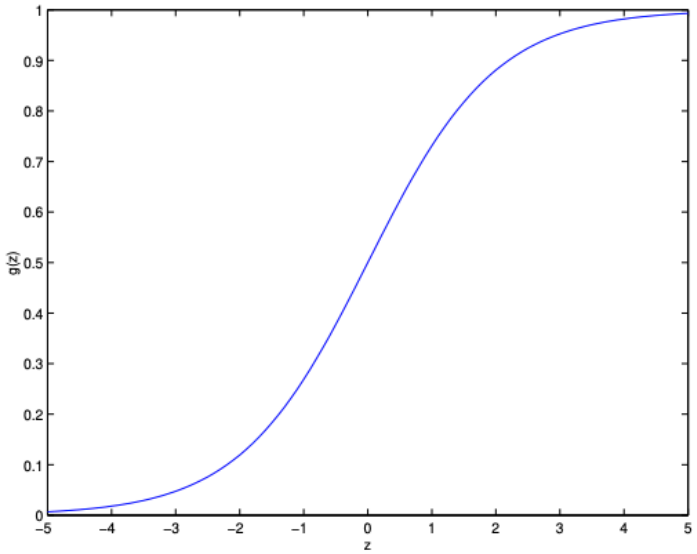
- Image 28×28: flatten to a one-dimensional vector $x \in \mathbb{R}^{784}$

- Classification function:
  - Let's assume a linear function $h(x) = g(\theta^T x)$

- Here we need a function $g(z)$ to convert $z = w^T x \in (-\infty, \infty)$ to $(0,1)$

Sigmoid Function

$$g(z) = \frac{1}{1 + e^{-z}}$$



Sigmoid function

Final model: $f(x) = g(\theta^T x)$

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

Maximum Likelihood Estimation

- In statistics, **maximum likelihood estimation (MLE)** is a method of estimating the parameters of an assumed probability distribution, given some observed data.

- This is achieved by maximizing **a likelihood function** so that, under the assumed statistical model, **the observed data is most probable.**

- The point in the parameter space (**network weight** θ) that maximizes the likelihood function is called the maximum likelihood estimate.

Task: Binary Classification — Is This Digit a 5?

Data

Model

Output

$x =$ 

$h($  $; \theta)$

?

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

Parameters or weights

$p(y = 1 \mid x)$

MINIST Dataset of handwritten digits

Probability of One Data Point

- Classification function: $h_\theta(x)$

$$p(y = 1 \mid x; \theta) = h_\theta(x)$$

$$p(y = 0 \mid x; \theta) = 1 - h_\theta(x)$$

- Writing more compactly to handle both $y = 0$ and $y = 1$,

$$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$

Probability of All Data Points

- Assume all the data points are independent, then

$$p(Y|X;\theta) = \prod_{i=1}^{n} p(y^{(i)}|x^{(i)};\theta) = \prod_{i=1}^{n} (h_\theta(x^{(i)}))^{y^{(i)}}(1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

$$\log p(Y|X;\theta) = \sum_{i=1}^{n} y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))$$

Loss: Negative Log-likelihood

- Loss: the thing you want to minimize

- Negative log-likelihood (NLL) loss

$$\mathscr{L}(\theta) = -\log p(Y|X;\theta)$$

$$= -\sum_{i=1}^{n} y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

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

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

Task: Binary Classification — Is This Digit a 5?

Data

Model

Output

$x =$ 

$h($  $; \theta)$

?

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

Parameters or weights

Probability of being a 5

$p(y = 1 \mid x)$

MINIST Dataset of handwritten digits

Loss: Negative Log-likelihood

- Loss: the thing you want to minimize

- Negative log-likelihood (NLL) loss

$$\mathscr{L}(\theta) = -\log p(Y|X;\theta)$$

$$= -\sum_{i=1}^{n} y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

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

Optimization 101

$\mathscr{L}(\theta)$

θ

How would you go to the very bottom?

More in-depth discussion, see https://web.stanford.edu/~boyd/cvxbook/.

Optimization Problems

(mathematical) optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq b_i, \quad i = 1, \ldots, m \end{array}$$

- $x = (x_1, \ldots, x_n)$: optimization variables
- $f_0 : \mathbf{R}^n \to \mathbf{R}$: objective function
- $f_i : \mathbf{R}^n \to \mathbf{R}$, $i = 1, \ldots, m$: constraint functions

optimal solution $x^\star$ has smallest value of f_0 among all vectors that satisfy the constraints

Optimization Problems

general optimization problem

- very difficult to solve
- methods involve some compromise, *e.g.*, very long computation time, or not always finding the solution

exceptions: certain problem classes can be solved efficiently and reliably

- least-squares problems
- linear programming problems
- convex optimization problems

$$\text{minimize} \quad \|Ax - b\|_2^2$$

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & a_i^T x \leq b_i, \quad i = 1, \ldots, m \end{array}$$

Convex and Non-Convex

Convex optimization problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq b_i, \quad i = 1, \ldots, m \end{array}$$

- objective and constraint functions are convex:

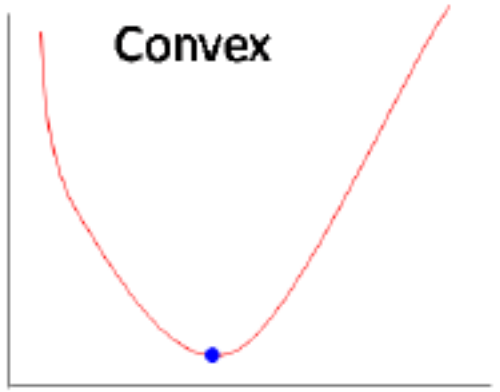
$$f_i(\alpha x + \beta y) \leq \alpha f_i(x) + \beta f_i(y)$$

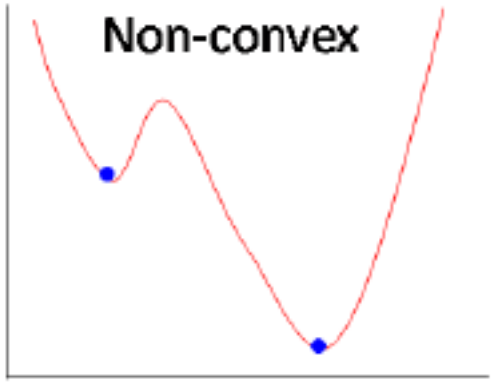
if $\alpha + \beta = 1$, $\alpha \geq 0$, $\beta \geq 0$

- includes least-squares problems and linear programs as special cases

solving convex optimization problems

- no analytical solution
- reliable and efficient algorithms





Gradient Descent

A first-order optimization method: Gradient Descent (GD)

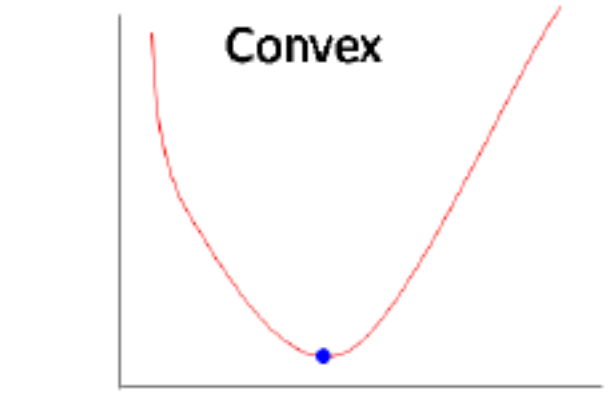
- Update rule for one iteration: $\theta := \theta - \alpha \nabla_\theta \mathscr{L}(\theta)$

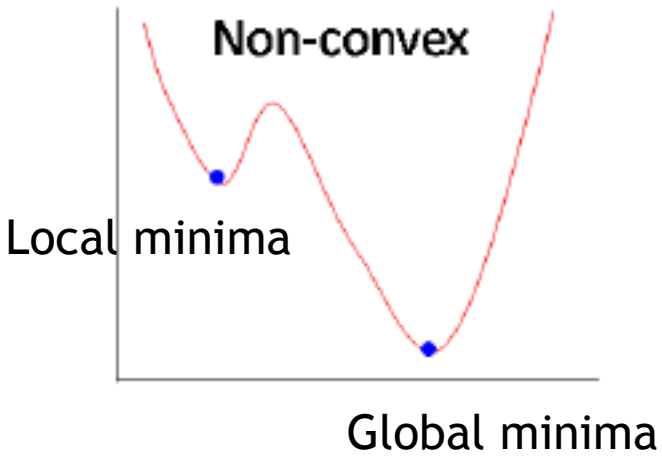
- Learning rate: α
  - If α is small enough, then GD will definitely lead to a smaller loss after the update. However, a too small α needs too many iterations to get the bottom.
  - If α is too big, overshoot! Loss not necessary to decrease.

Local/Global Minima

For convex optimization problem, gradient descent will converge to the global minima.

For general optimization problem, gradient descent will converge to a local minima.





Analytical Gradient

- How to perform GD to minimize NLL loss?

- Derive analytical gradient:
  - For Sigmoid function

$$
\begin{aligned}
g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\
&= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\
&= \frac{1}{(1+e^{-z})}\cdot\left(1-\frac{1}{(1+e^{-z})}\right) \\
&= g(z)(1-g(z)).
\end{aligned}
$$

Analytical Gradient

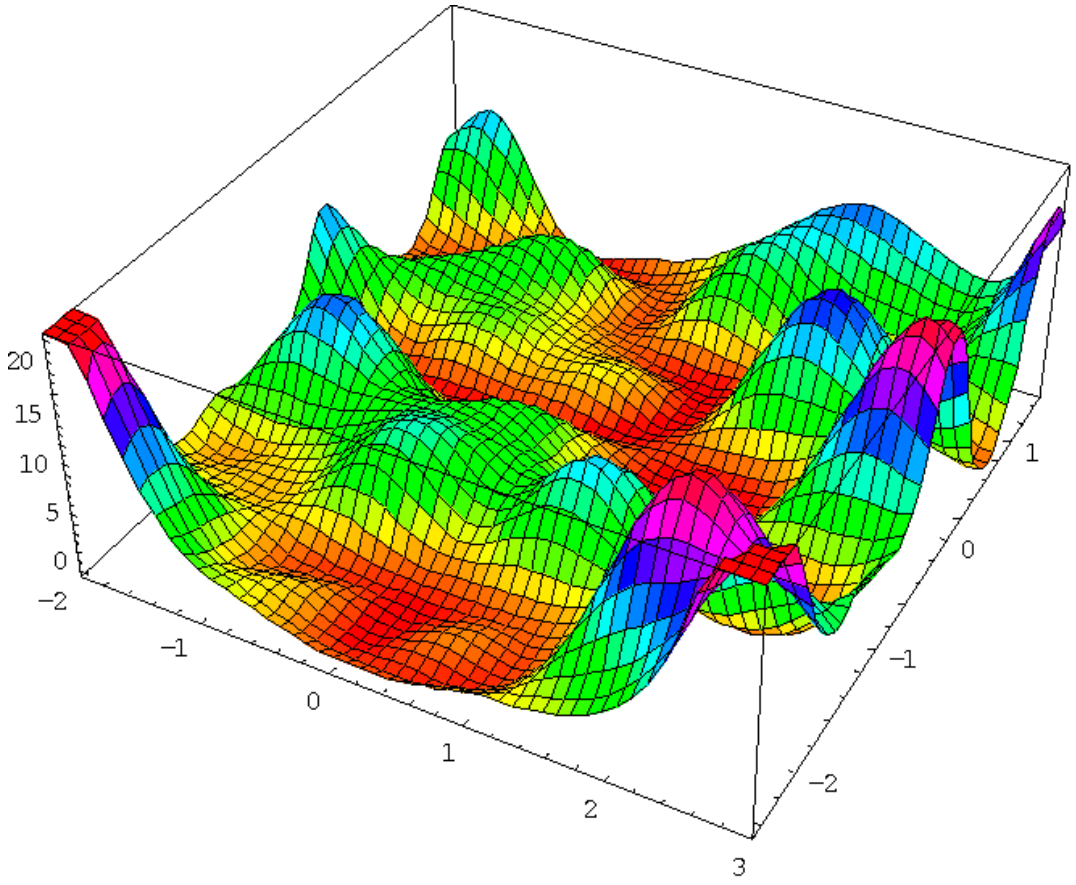
- How to perform GD to minimize NLL loss?

- Derive analytical gradient:

$$\mathscr{L} = -\sum_{i=1}^{n} y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))$$

$$\begin{aligned}
\frac{\partial \mathscr{L}}{\partial \theta_j} &= -\sum \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\
&= -\sum \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\
&= -\sum \left(y(1 - g(\theta^T x)) - (1 - y) g(\theta^T x) \right) x_j \\
&= -\sum \left(y - h_\theta(x) \right) x_j
\end{aligned}$$

Non-Linear and Non-Convex Optimization



Non-convex energy landscape

Naive gradient descent will trap at local minima.

Batch Gradient Descent vs. Stochastic Gradient Descent

- Batch Gradient Descent

Take all data and label pairs in the training set to calculate the gradient.

-: very slow

-: easily get trapped at local minima

- Stochastic Gradient Descent (SGD, or Mini-batch Gradient Descent)

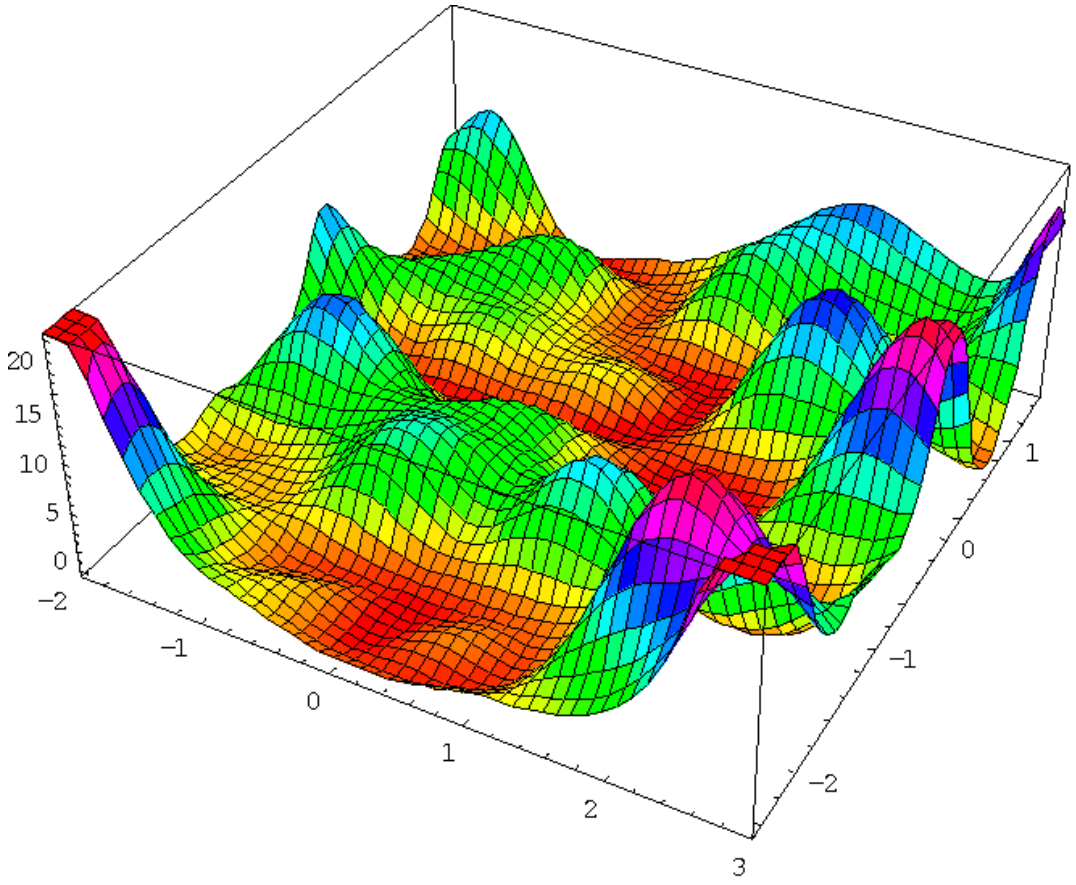
Randomly sample N pairs as a batch from the training data and then compute the average gradient from them.

+: fast

+: can get out of local minima

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W)$$

Non-Linear and Non-Convex Optimization



Non-convex energy landscape

SGD has the potential to jump out of a local minima.

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

Testing and Evaluation

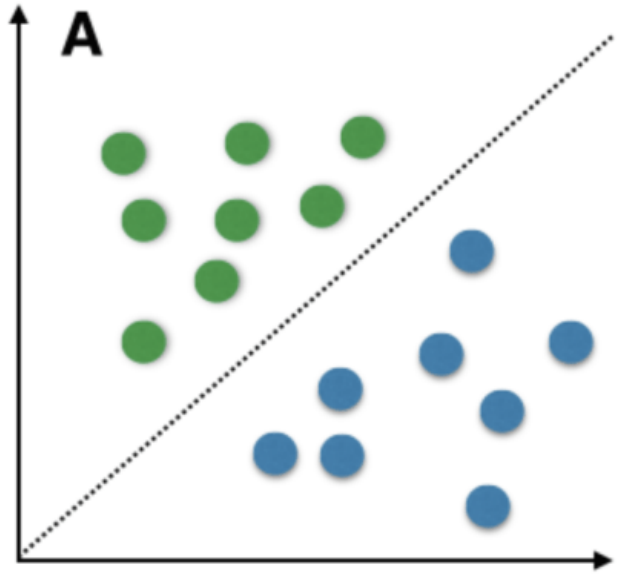
- After training, we need to know how well our model generalizes to unseen data or test data.
- Evaluate the classification accuracy on the test split.
- Will we still work well?

Generalization gap!

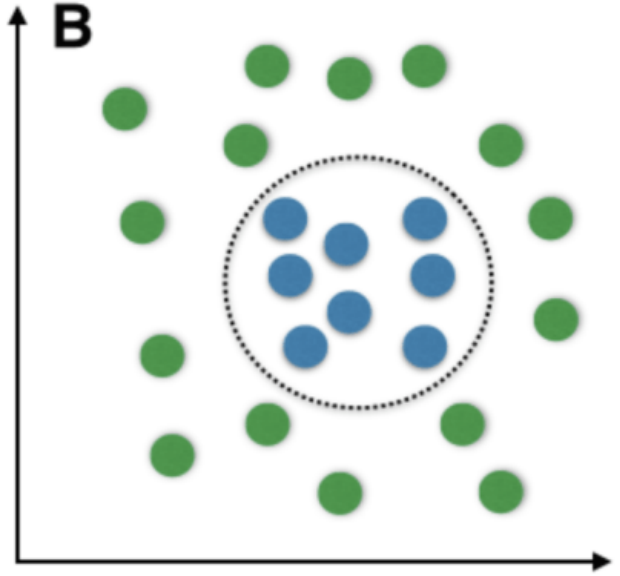
Multilayer Perceptron

Problem with Single-Layer Network

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

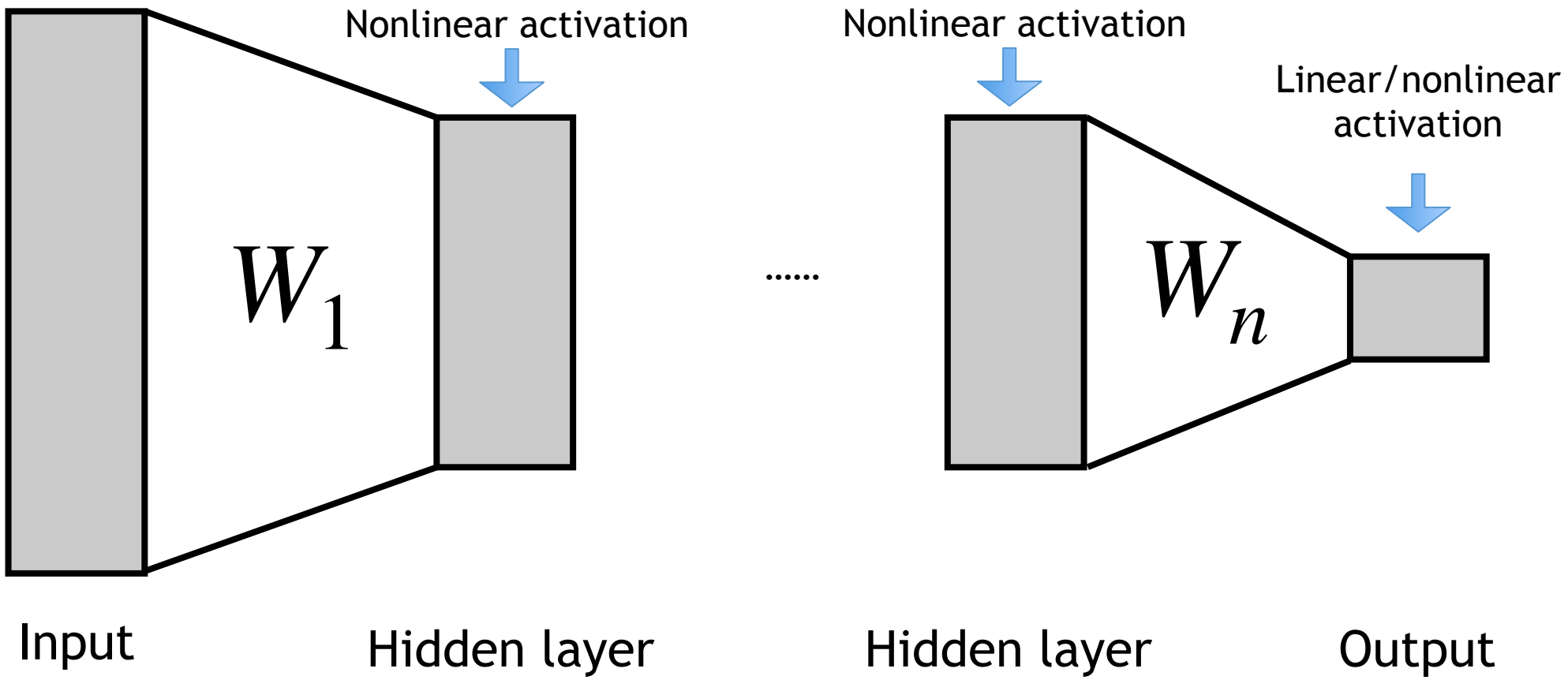


Linear separable



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

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

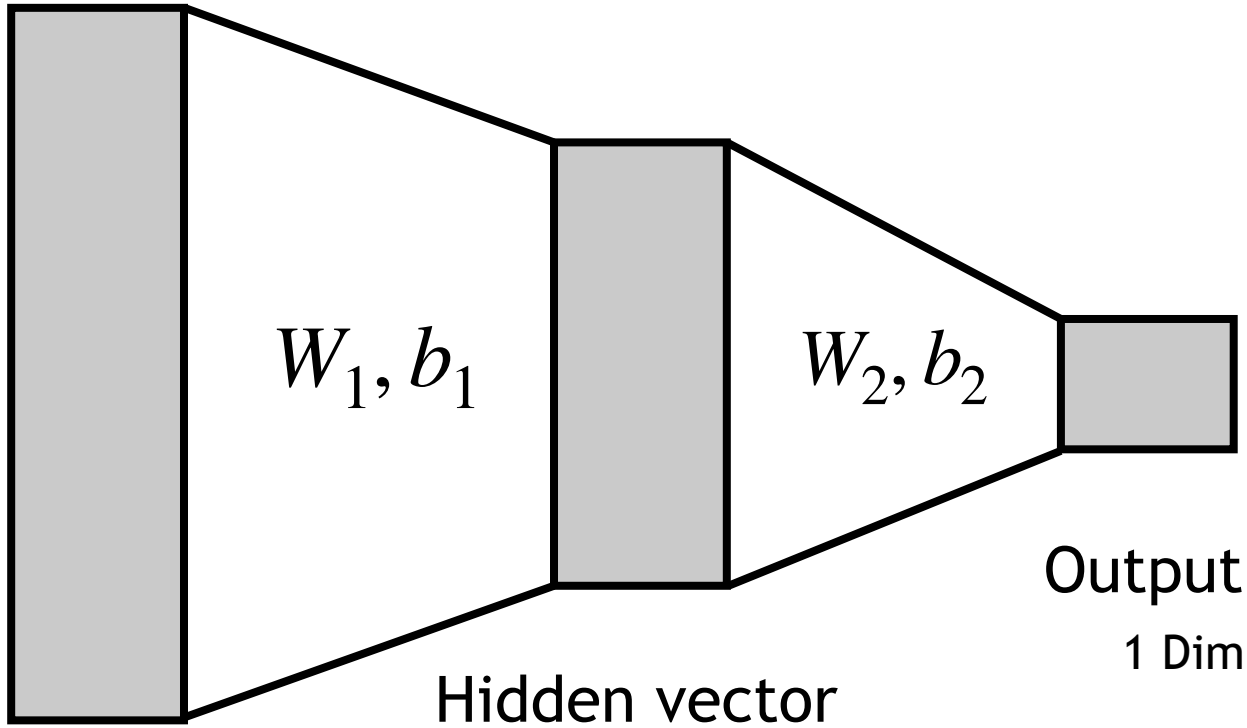
Linear function

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

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



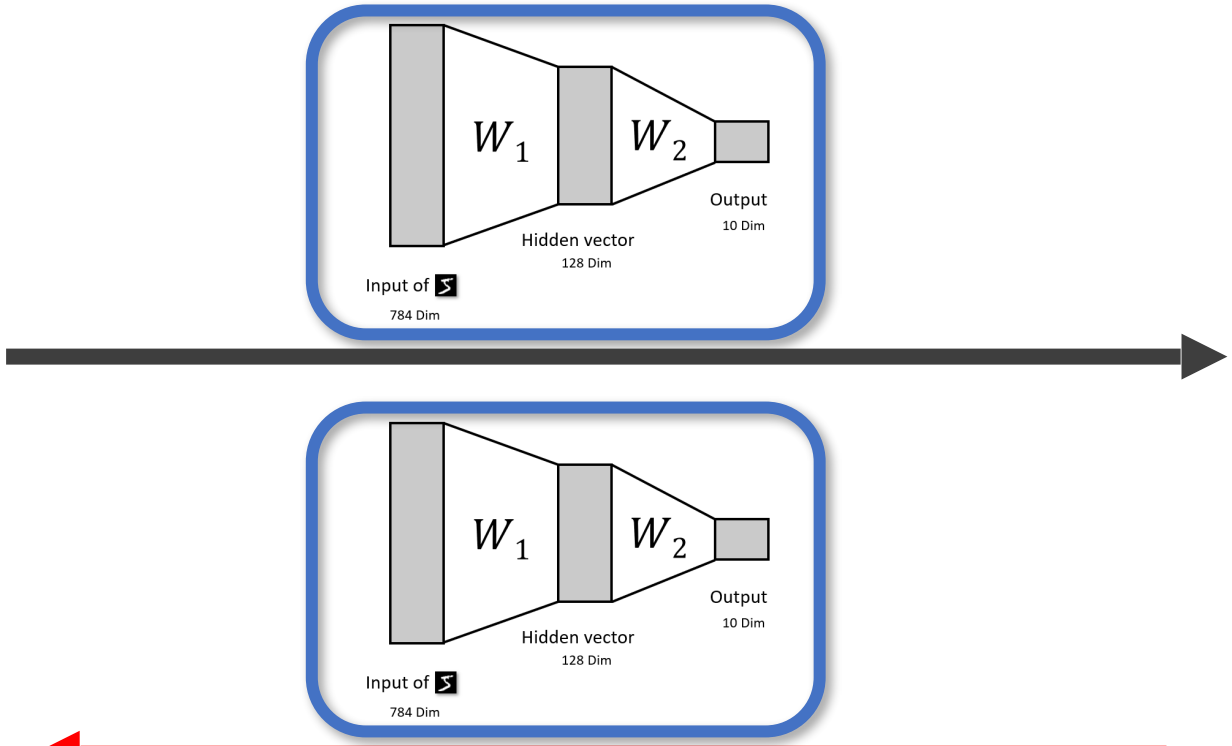
$$W_1 \in \mathbb{R}^{784\times128}, W_2 \in \mathbb{R}^{128\times1}$$
$$b_1 \in \mathbb{R}^{128}, b_2 \in \mathbb{R}$$

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

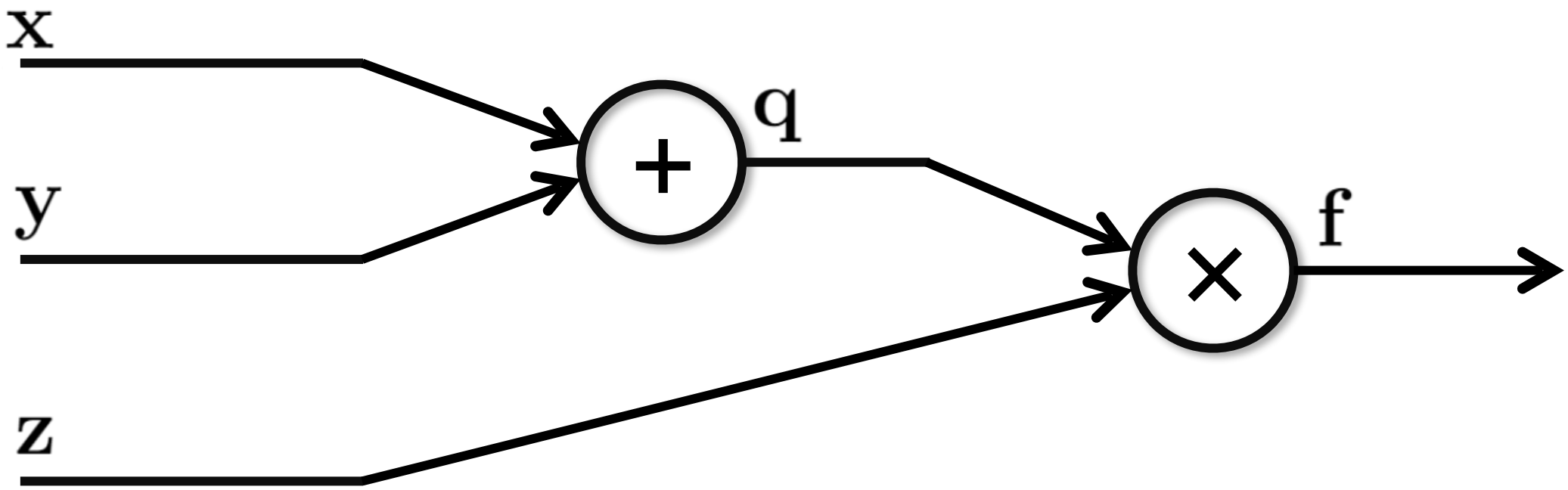
However:

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

Backpropagation with a toy example

Let we consider a toy example:
(Computational graph)

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

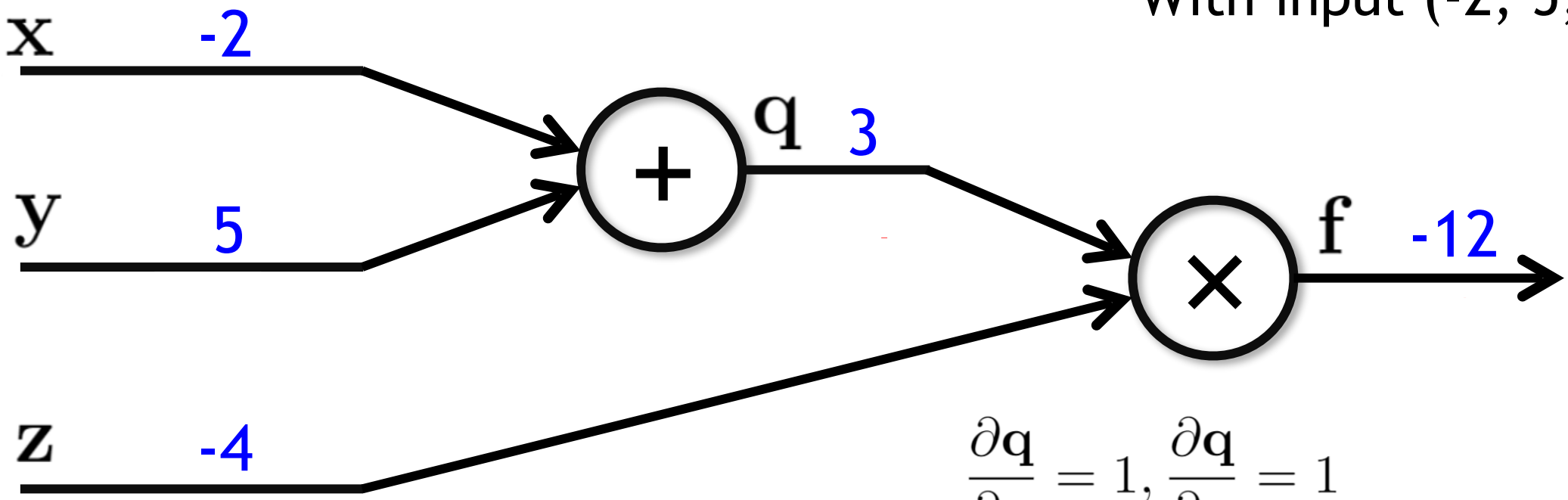


And we want $\frac{\partial f}{\partial \mathbf{x}}, \frac{\partial f}{\partial \mathbf{y}}, \frac{\partial f}{\partial \mathbf{z}}$

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)



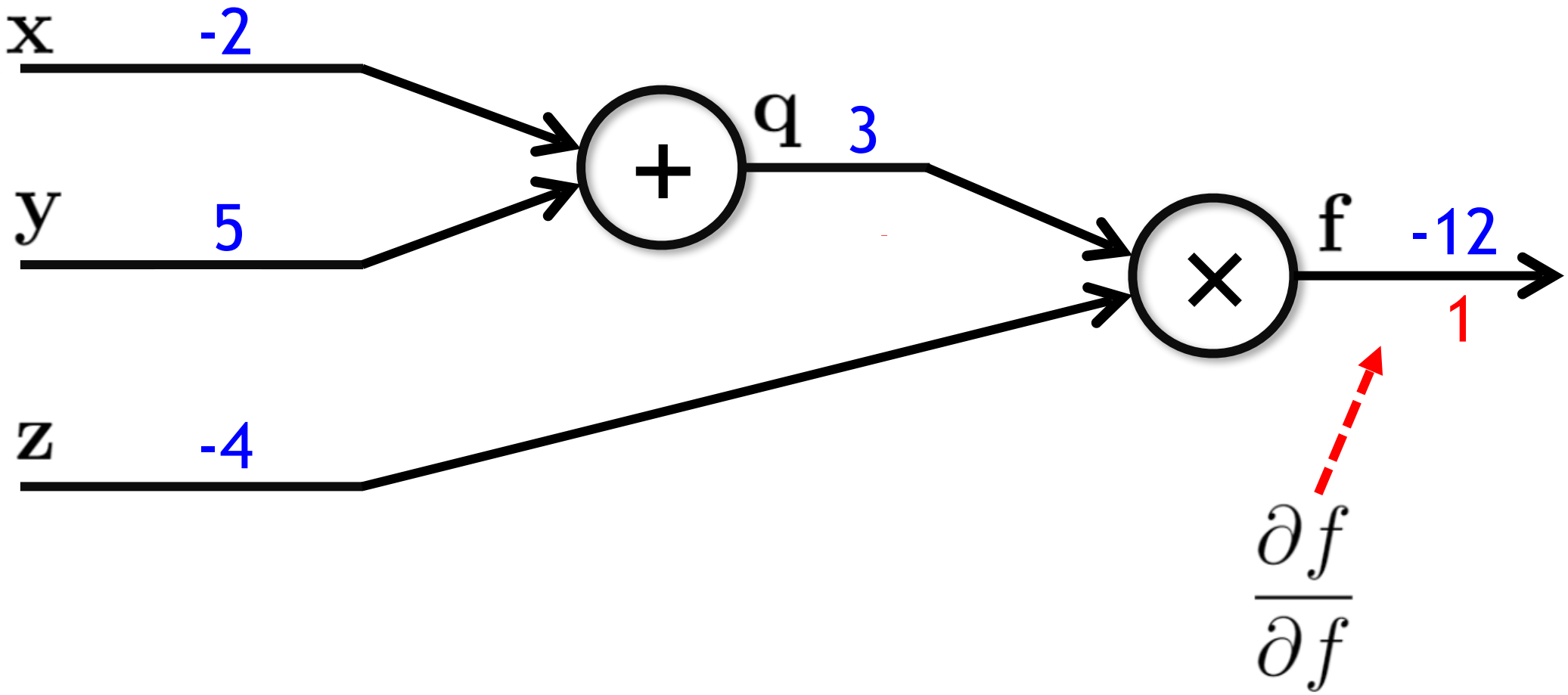
And we have the derivation

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

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

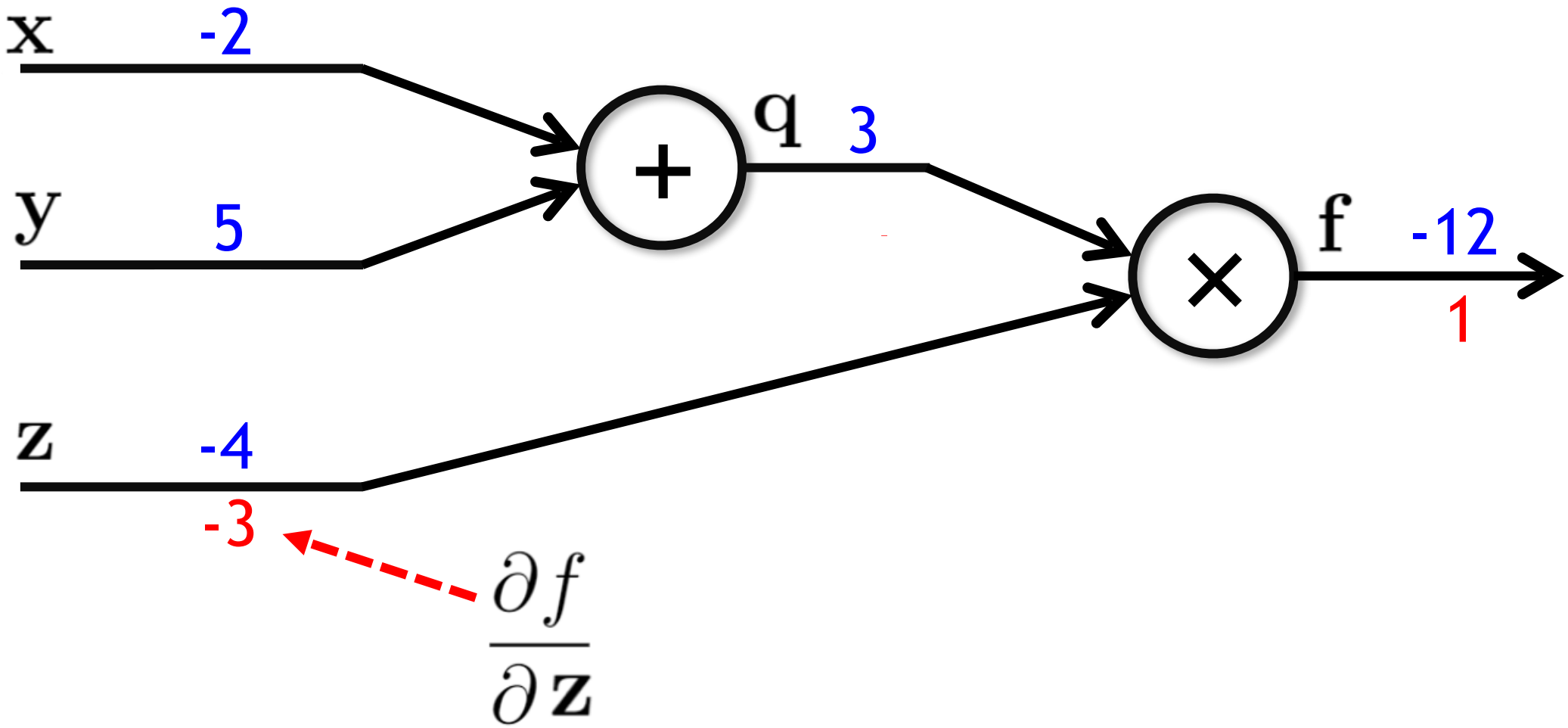
Backpropagation with a toy example

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



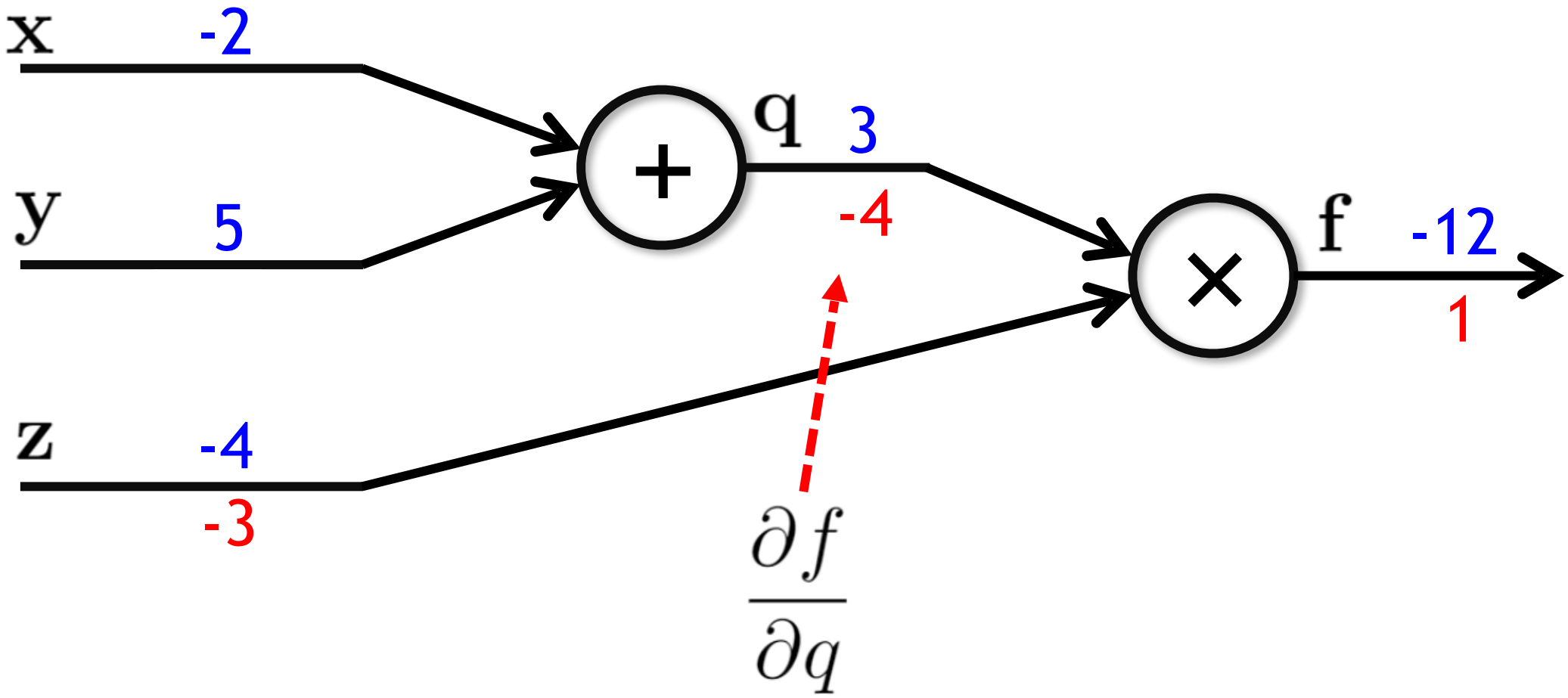
Backpropagation with a toy example

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



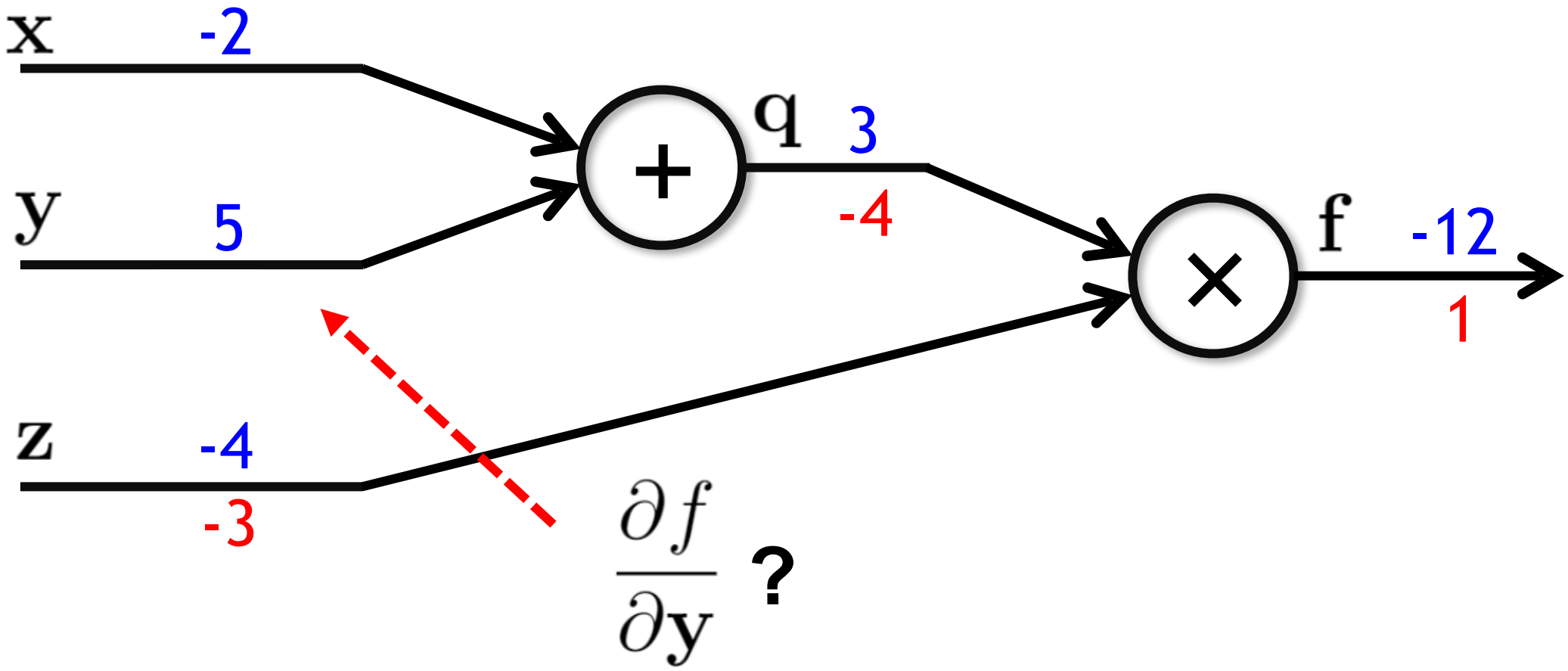
Backpropagation with a toy example

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



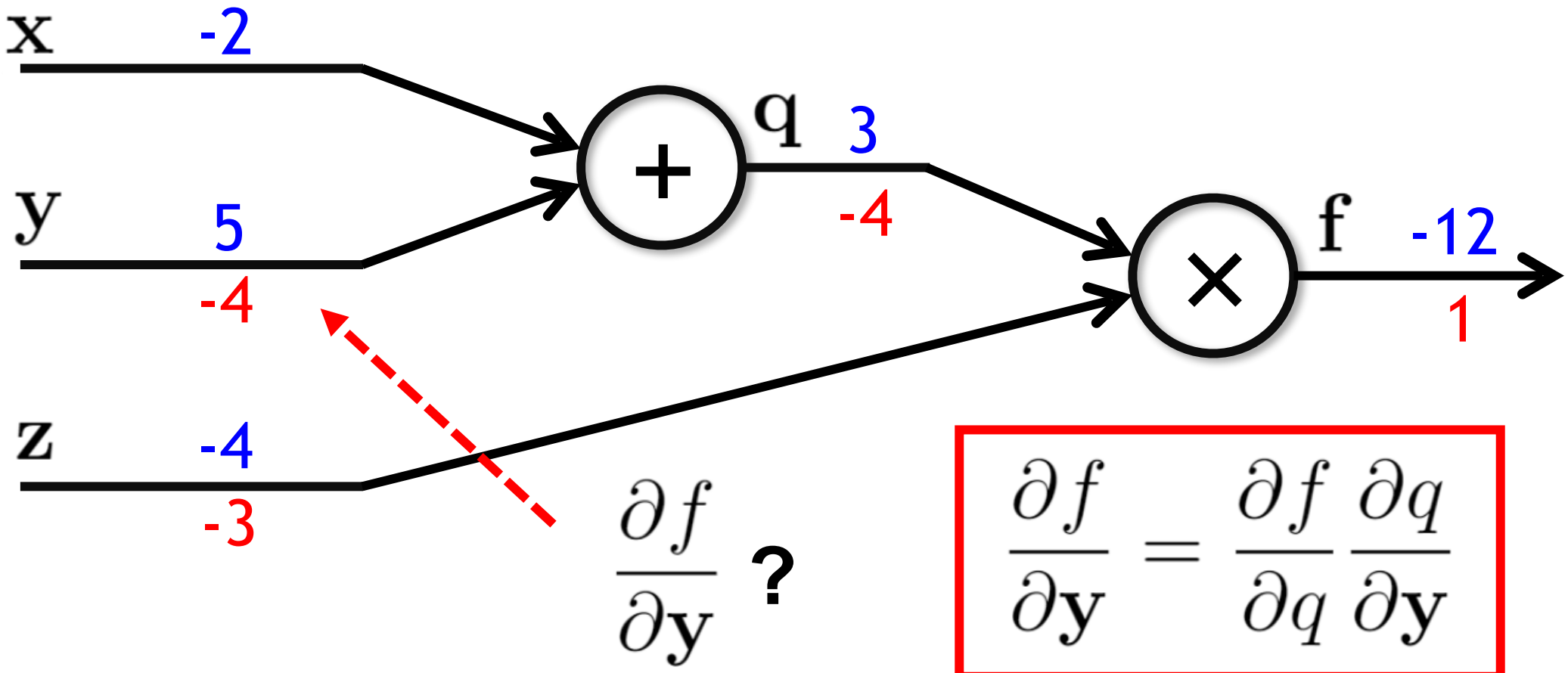
Backpropagation with a toy example

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

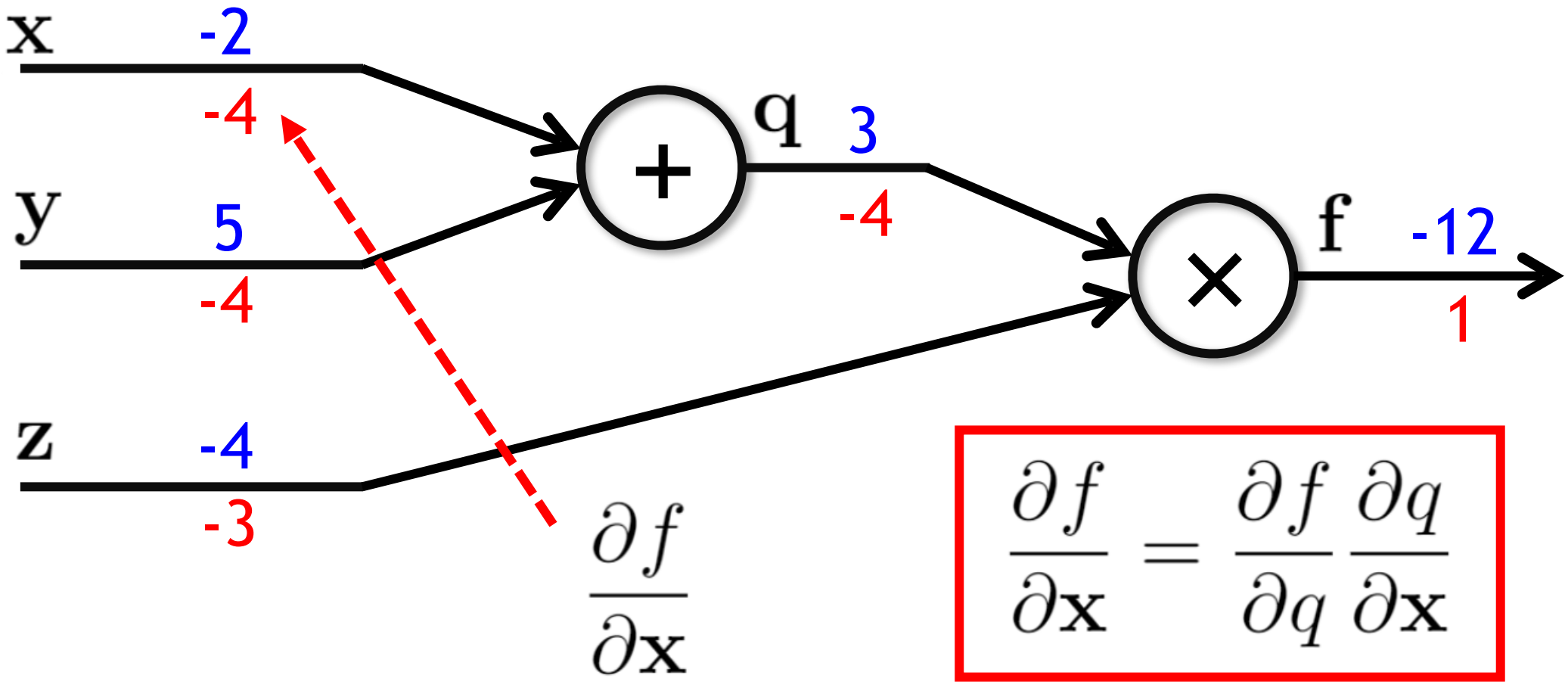


$$\frac{\partial f}{\partial \mathbf{y}} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial \mathbf{y}}$$

Chain rule

Backpropagation with a toy example

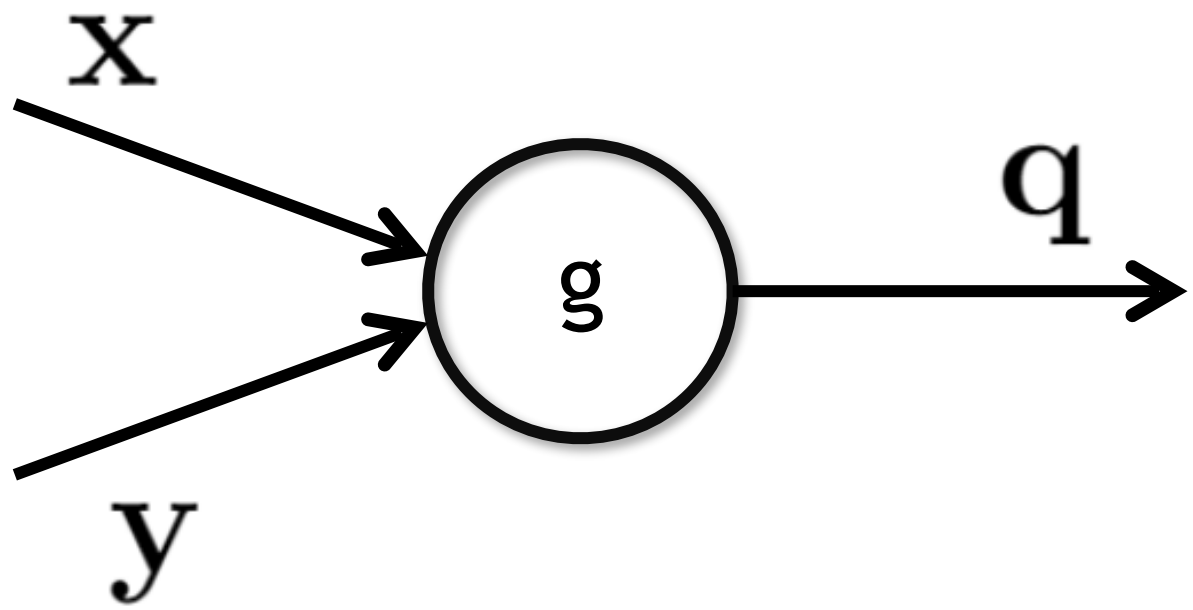
Let we consider a toy example: $f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = (\mathbf{x} + \mathbf{y}) \cdot \mathbf{z}$



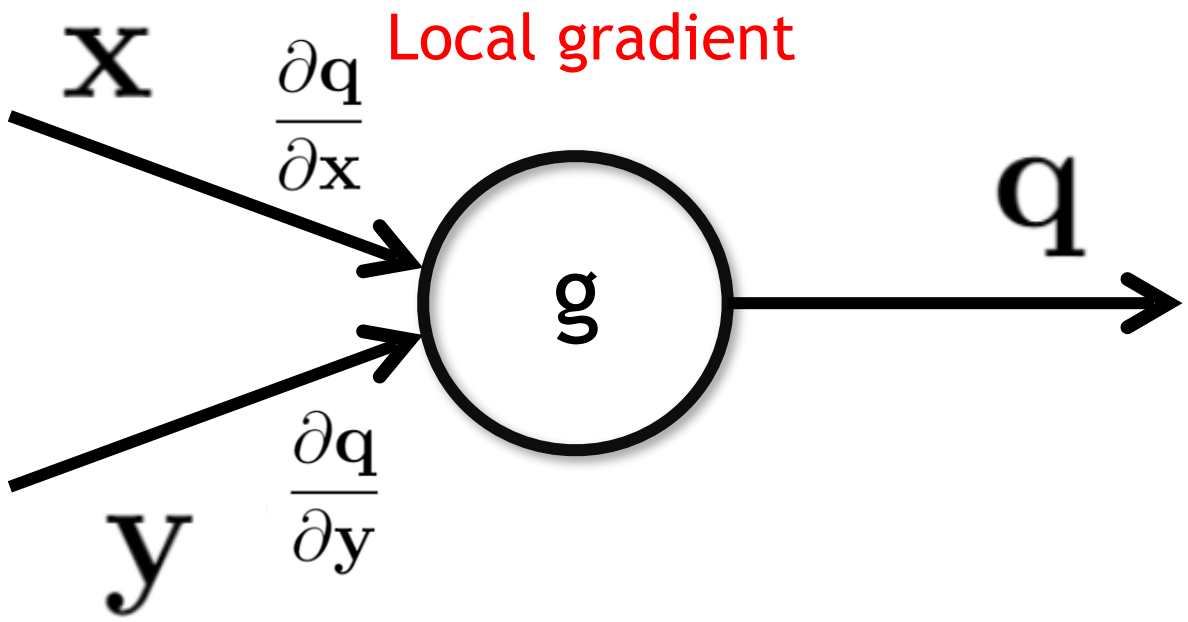
$$\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial \mathbf{x}}$$

Chain rule

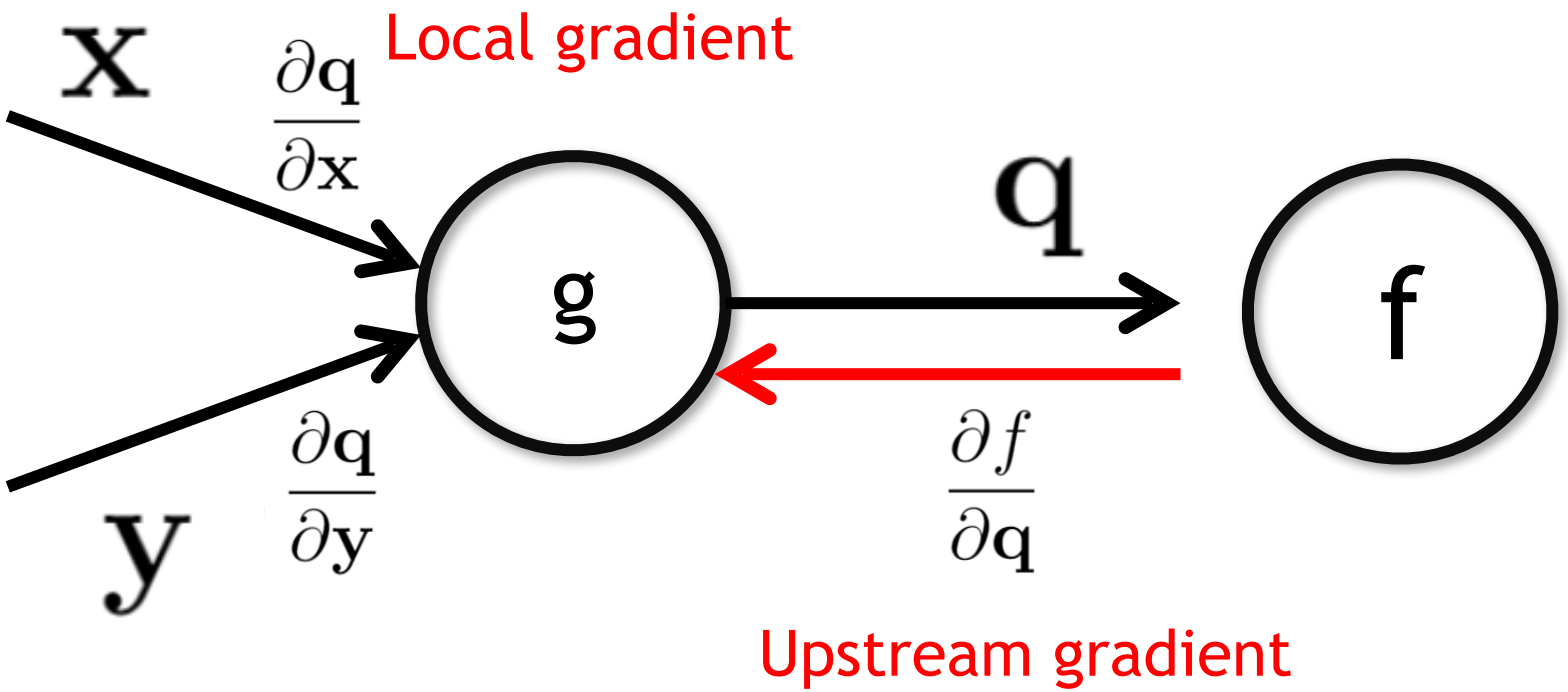
Chain rule



Chain rule

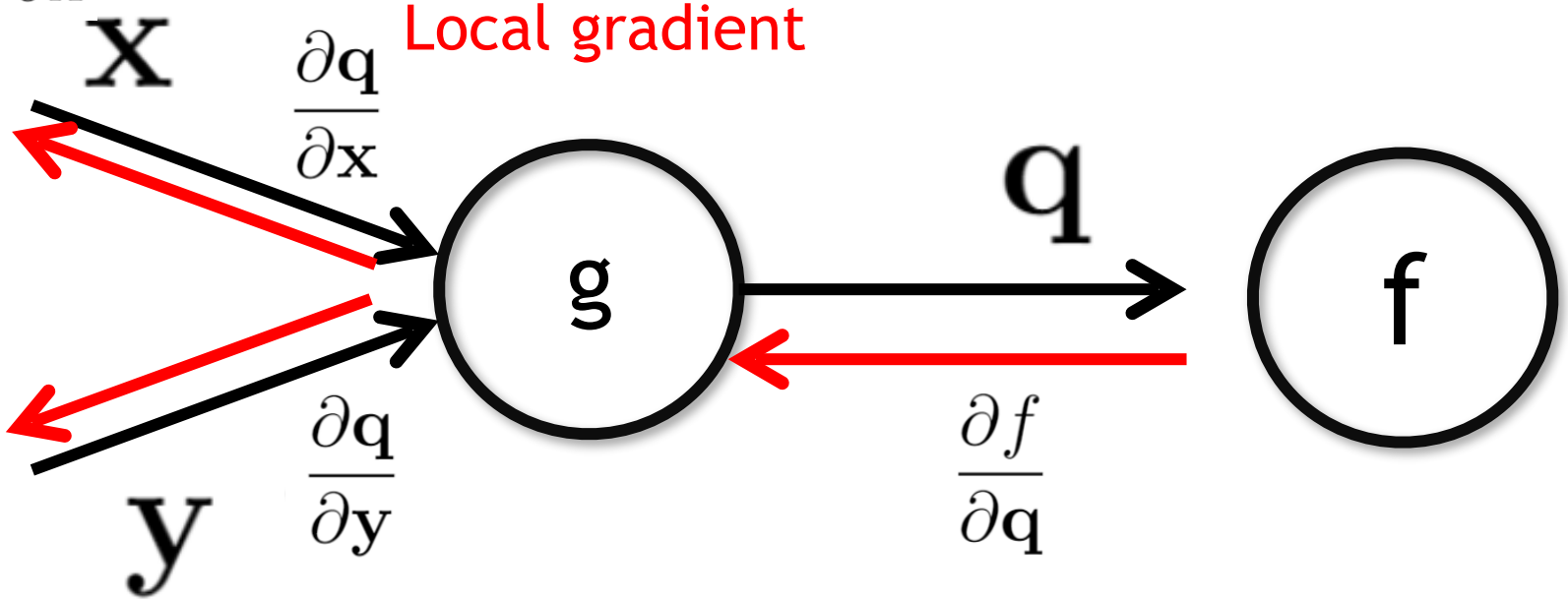

x
$\frac{\partial q}{\partial x}$
Local gradient
g
q
y
$\frac{\partial q}{\partial y}$

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

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

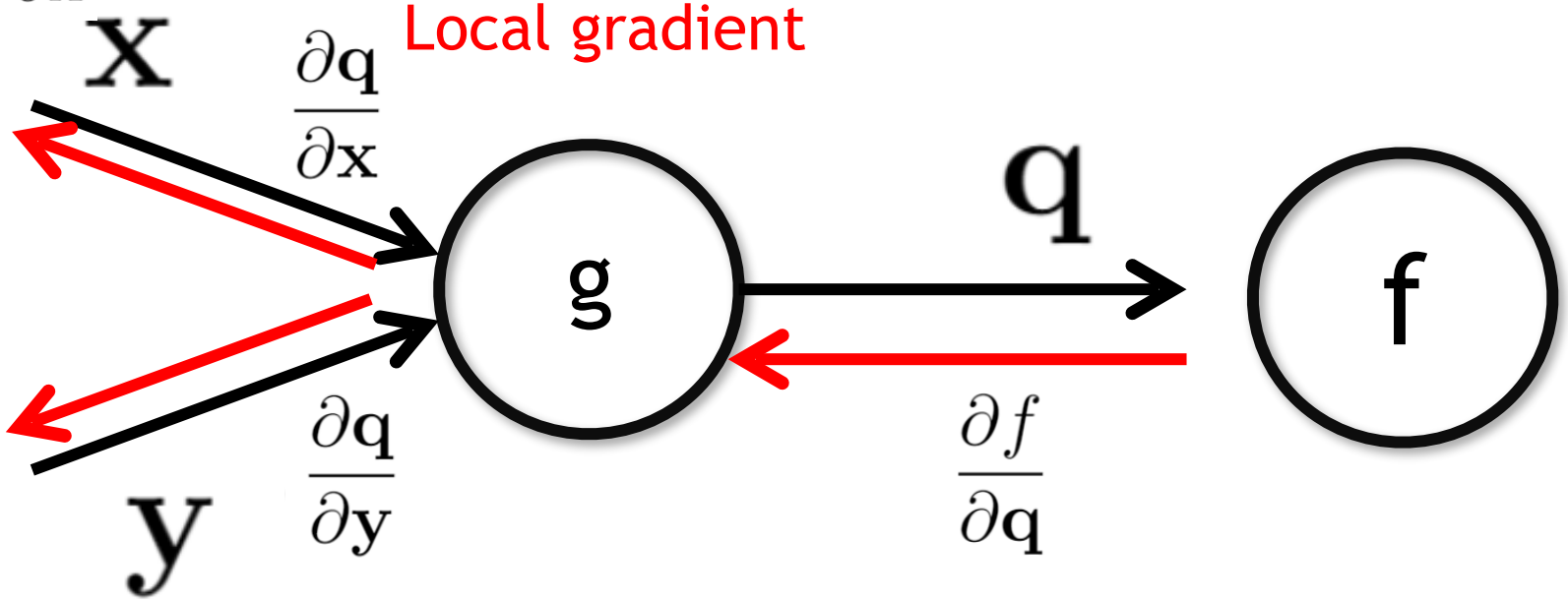


$$\frac{\partial f}{\partial \mathbf{y}} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial \mathbf{y}}$$

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

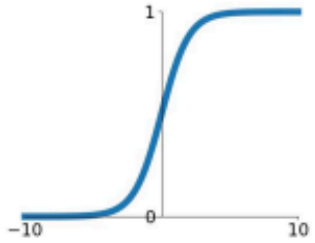
"Backpropagation for a Linear Layer" from Justin 2017

Activation Function

Activation functions

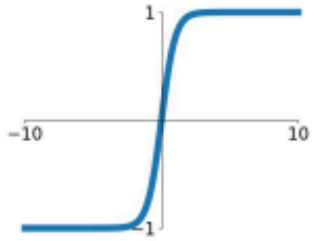
Sigmoid

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



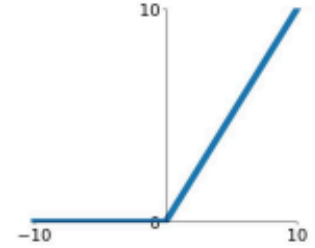
tanh

$$\tanh(x)$$



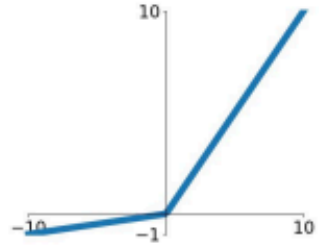
ReLU

$$\max(0, x)$$



Leaky ReLU

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

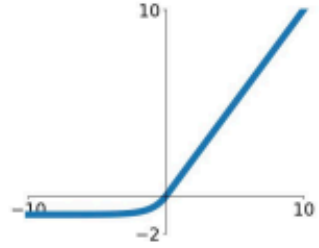


Maxout

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

ELU

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



Slides credit: Stanford CS231N

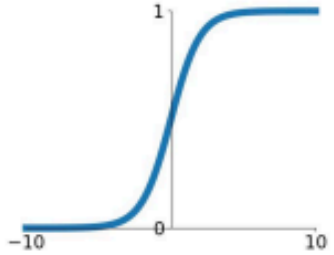
Activation Function

Activation functions

ReLU is a good default choice for most problems

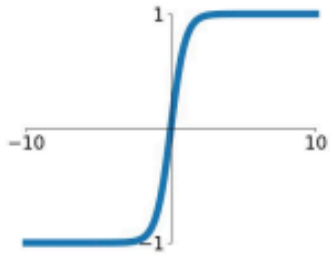
Sigmoid

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



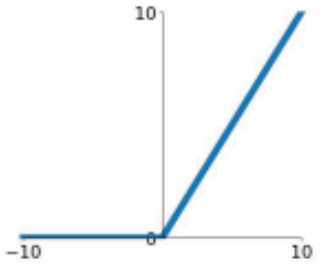
tanh

$$\tanh(x)$$



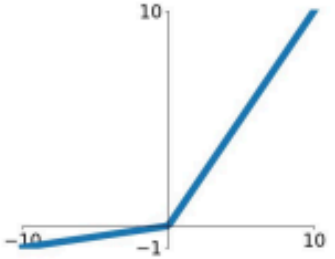
ReLU

$$\max(0, x)$$



Leaky ReLU

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

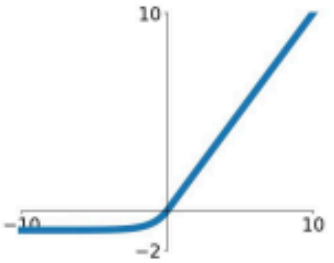


Maxout

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

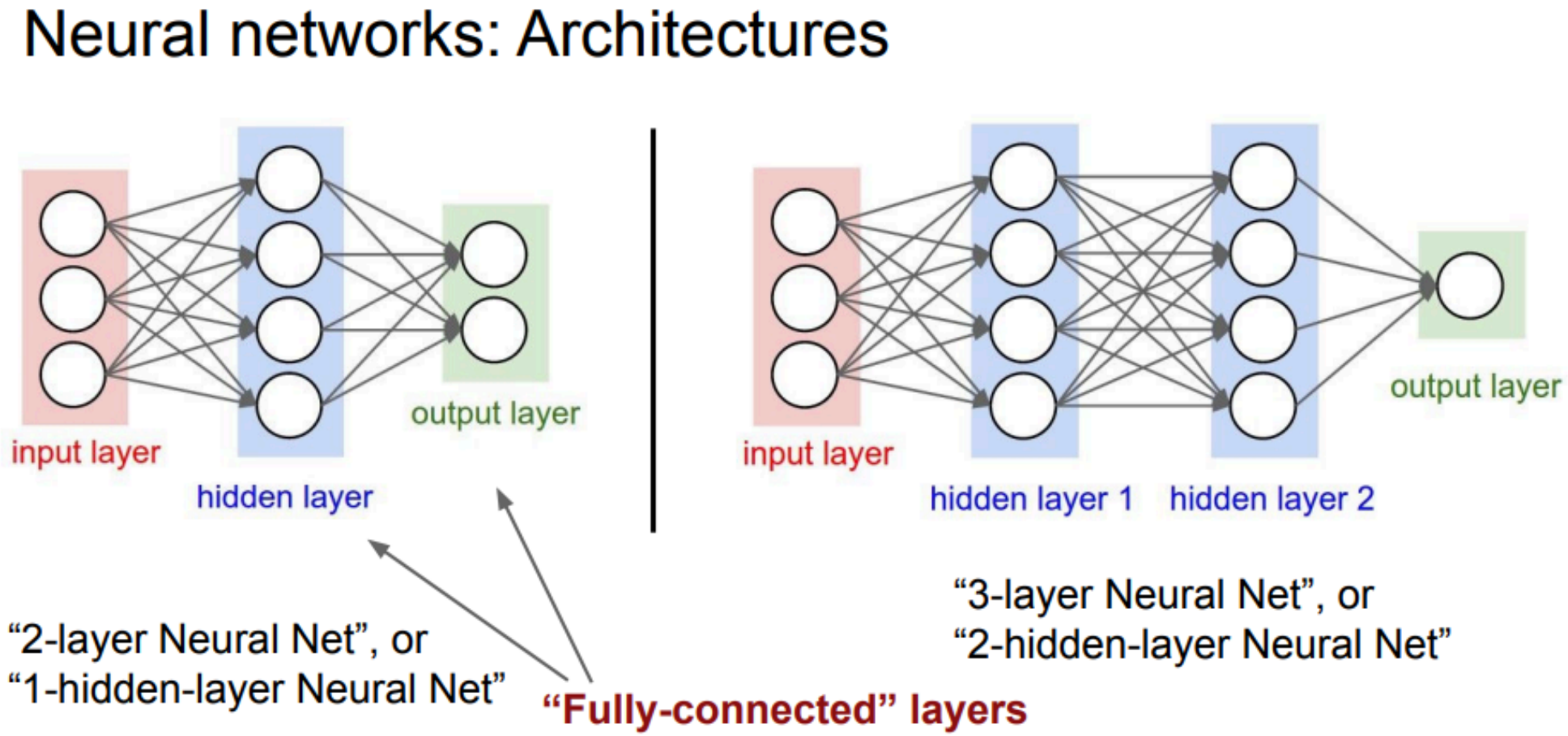
ELU

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



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

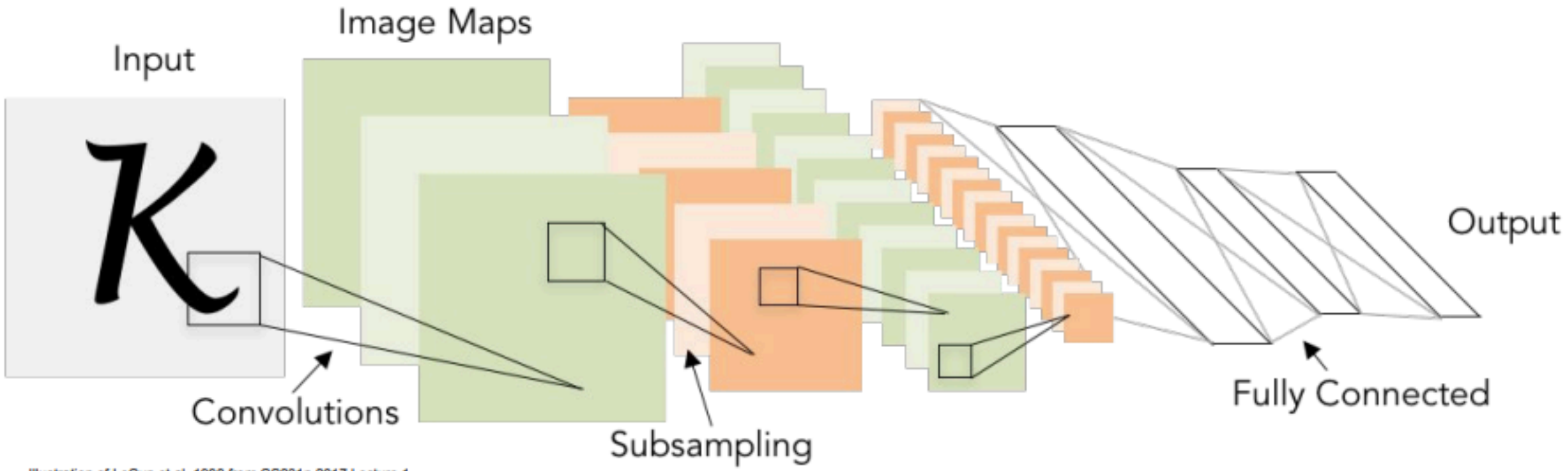
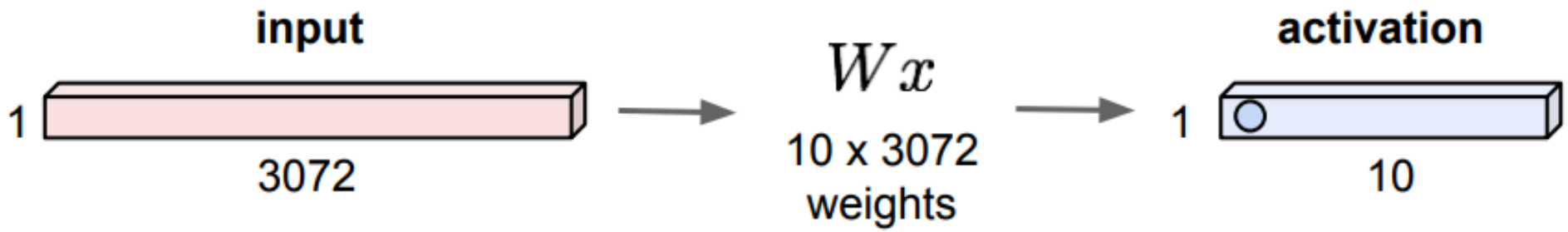


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Convolutional Neural Network

Recap: Fully Connected Layer

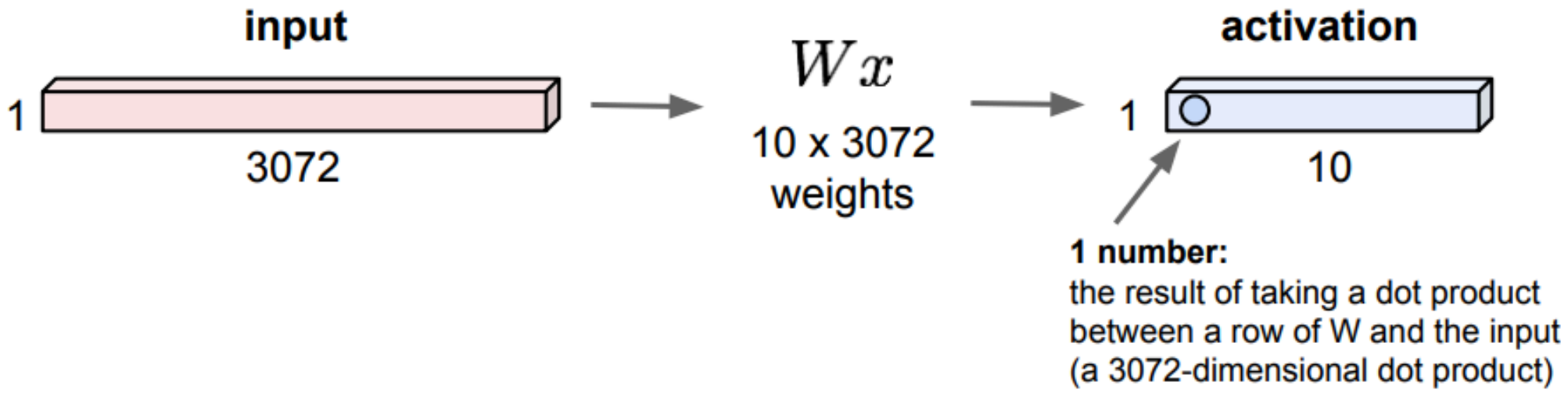
32x32x3 image -> stretch to 3072 x 1



Convolutional Neural Network

Fully Connected Layer

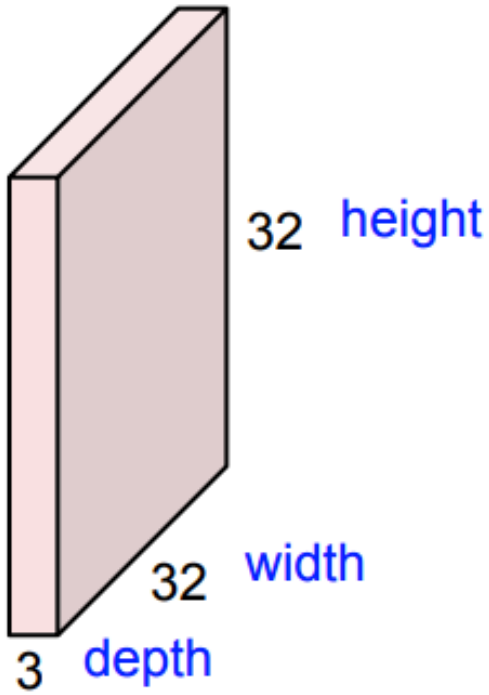
32x32x3 image -> stretch to 3072 x 1



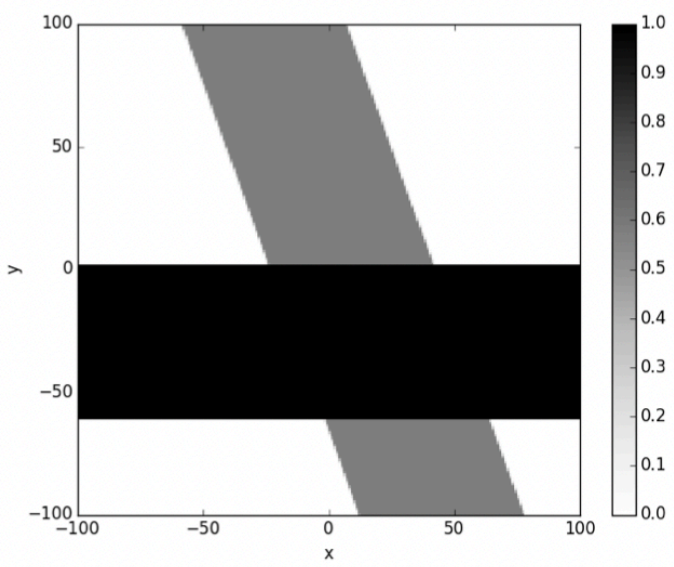
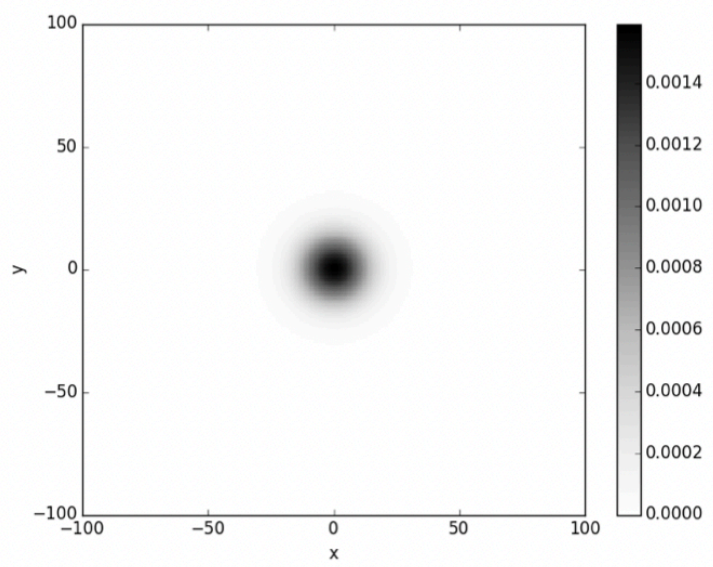
Convolutional Neural Network

Convolution Layer

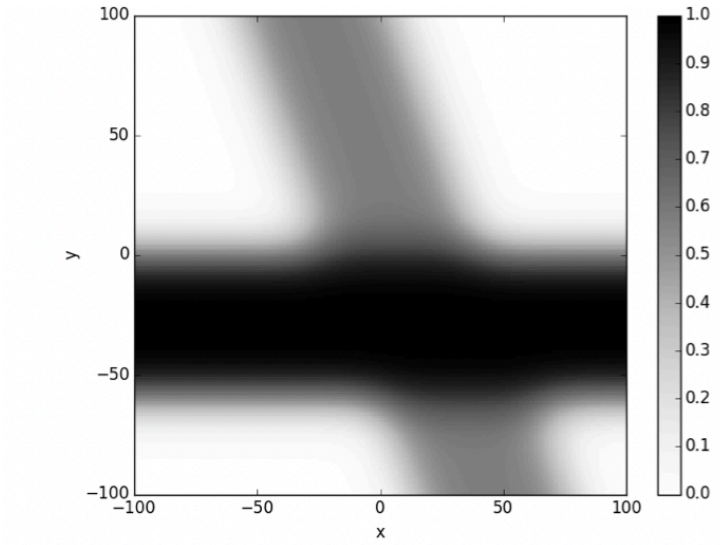
32x32x3 image -> preserve spatial structure



Two-Dimensional Convolution

f  $*$ 

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

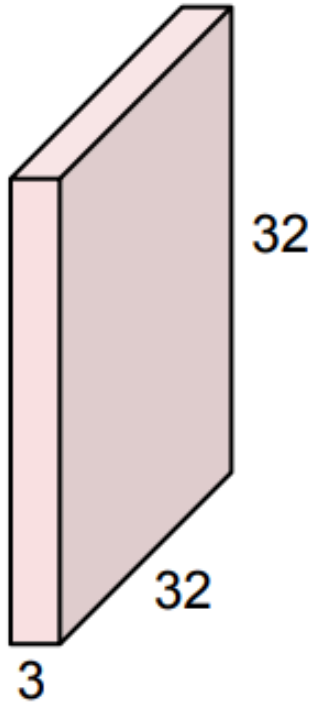
$=$ 

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

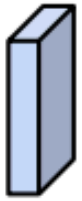
Convolutional Neural Network

Convolution Layer

32x32x3 image



5x5x3 filter



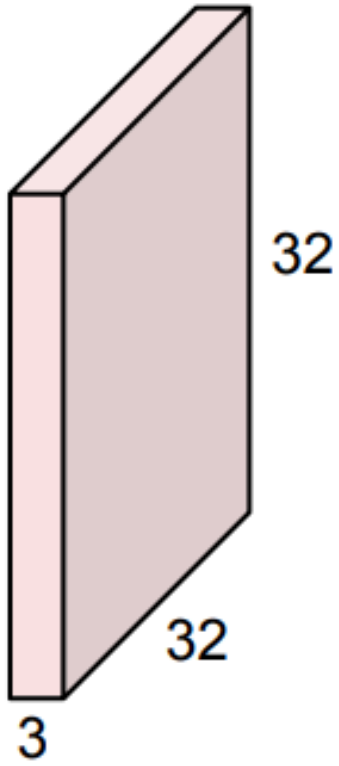
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolutional Neural Network

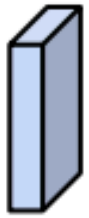
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



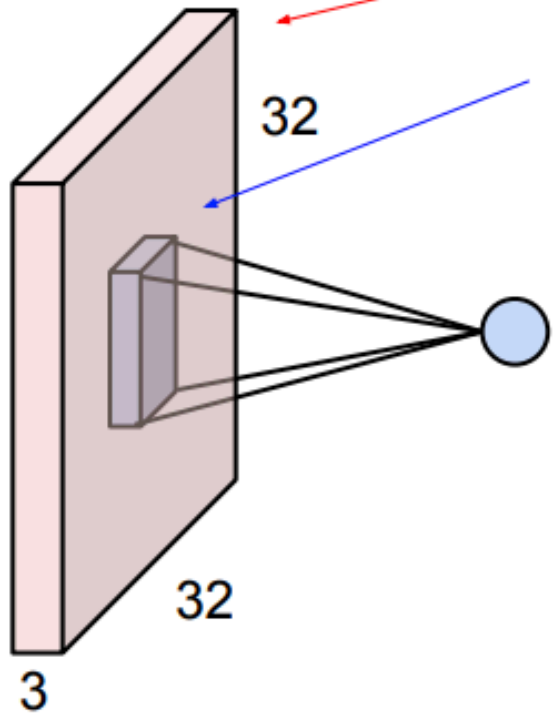
5x5x3 filter



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

Convolutional Neural Network

Convolution Layer



32x32x3 image

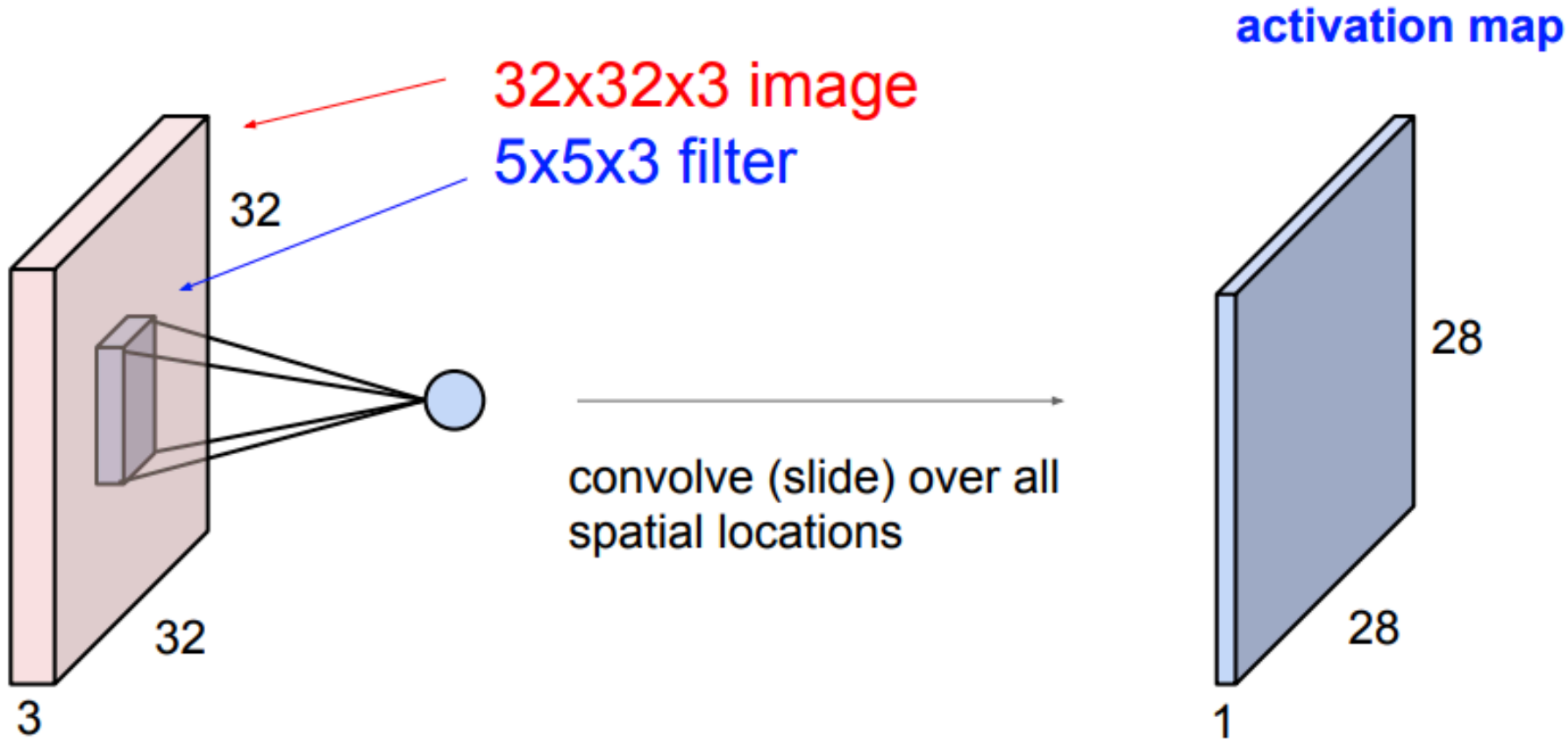
5x5x3 filter w

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

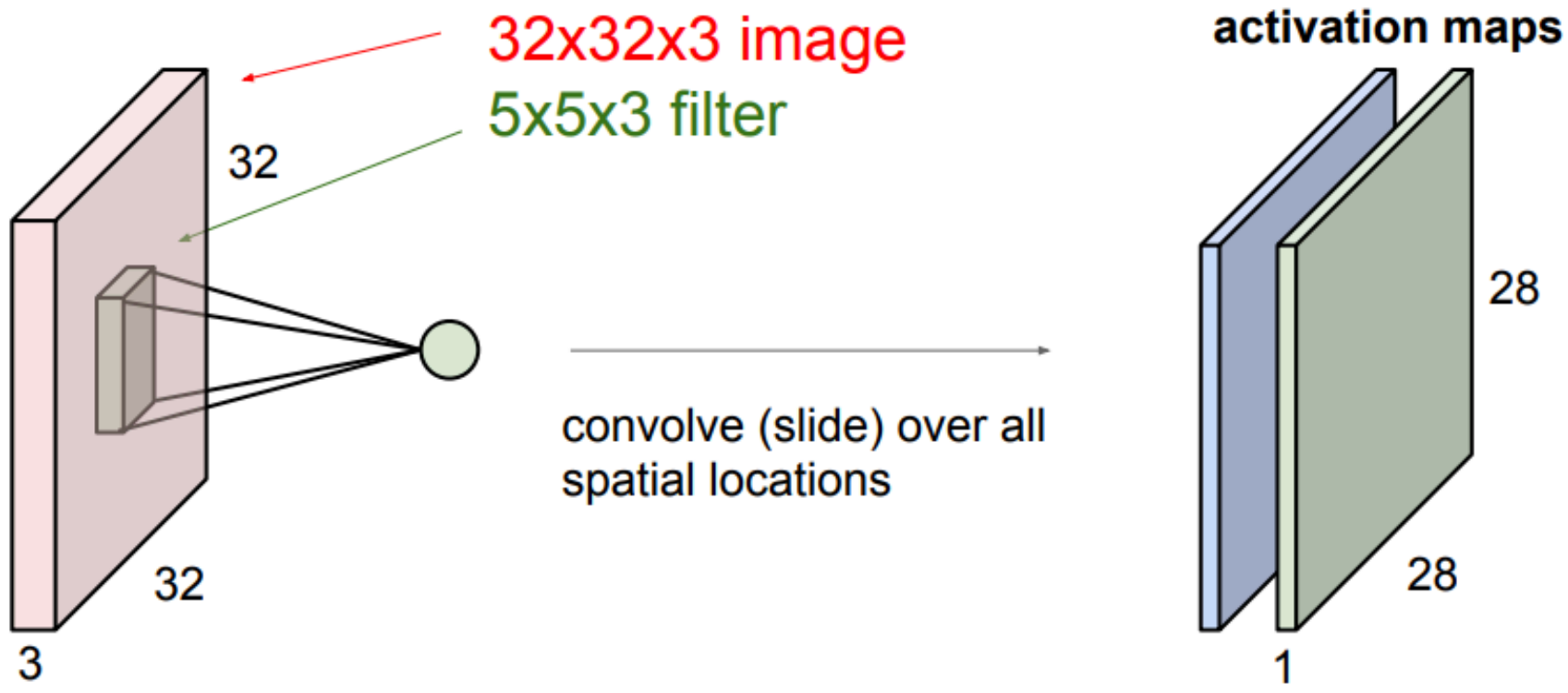
Convolution Layer



Convolutional Neural Network

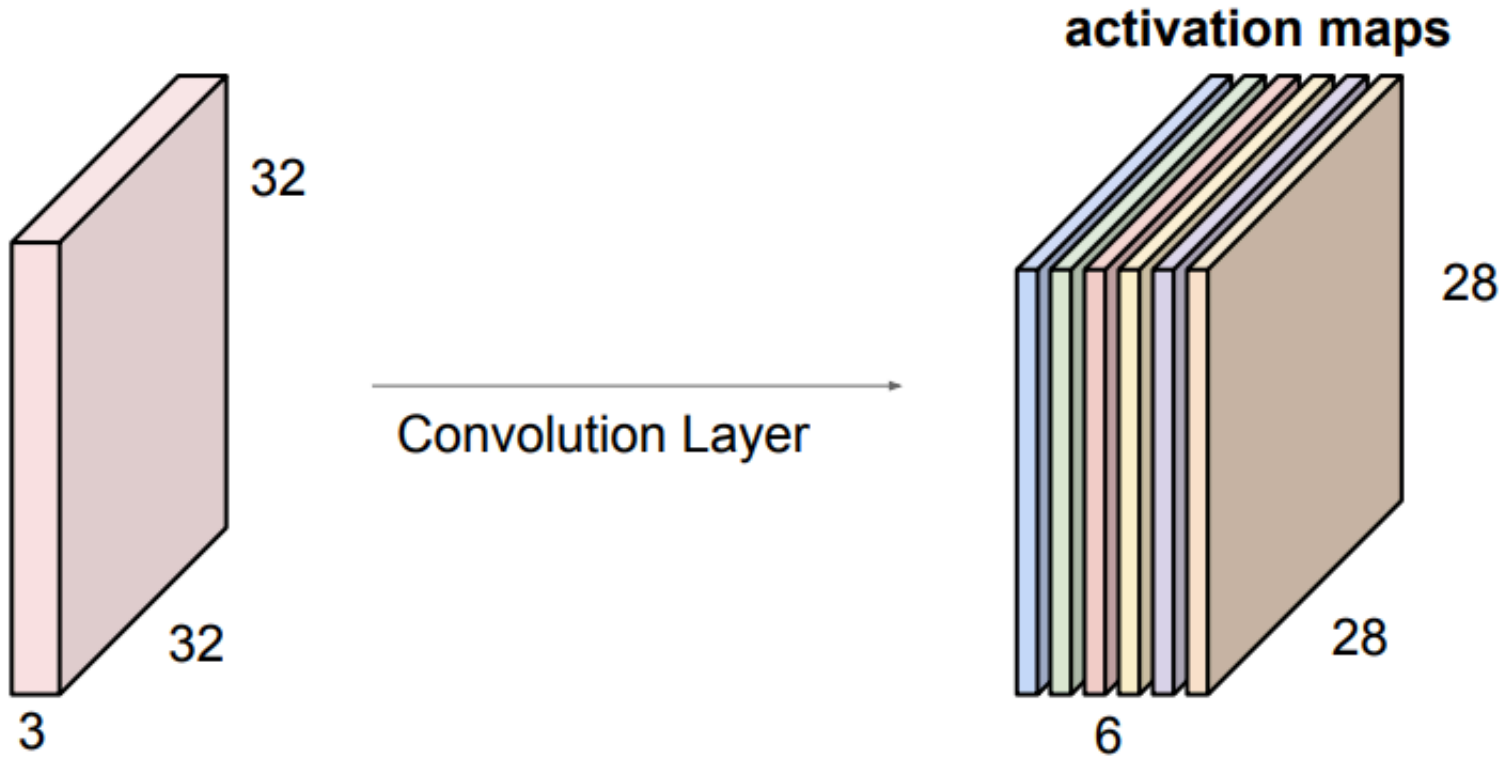
Convolution Layer

consider a second, green filter



Convolutional Neural Network

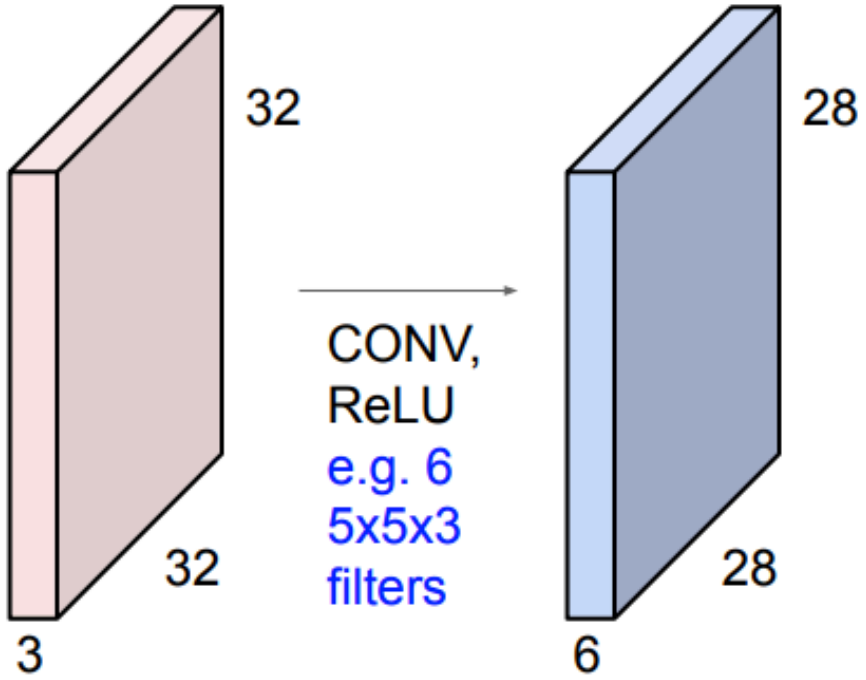
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



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

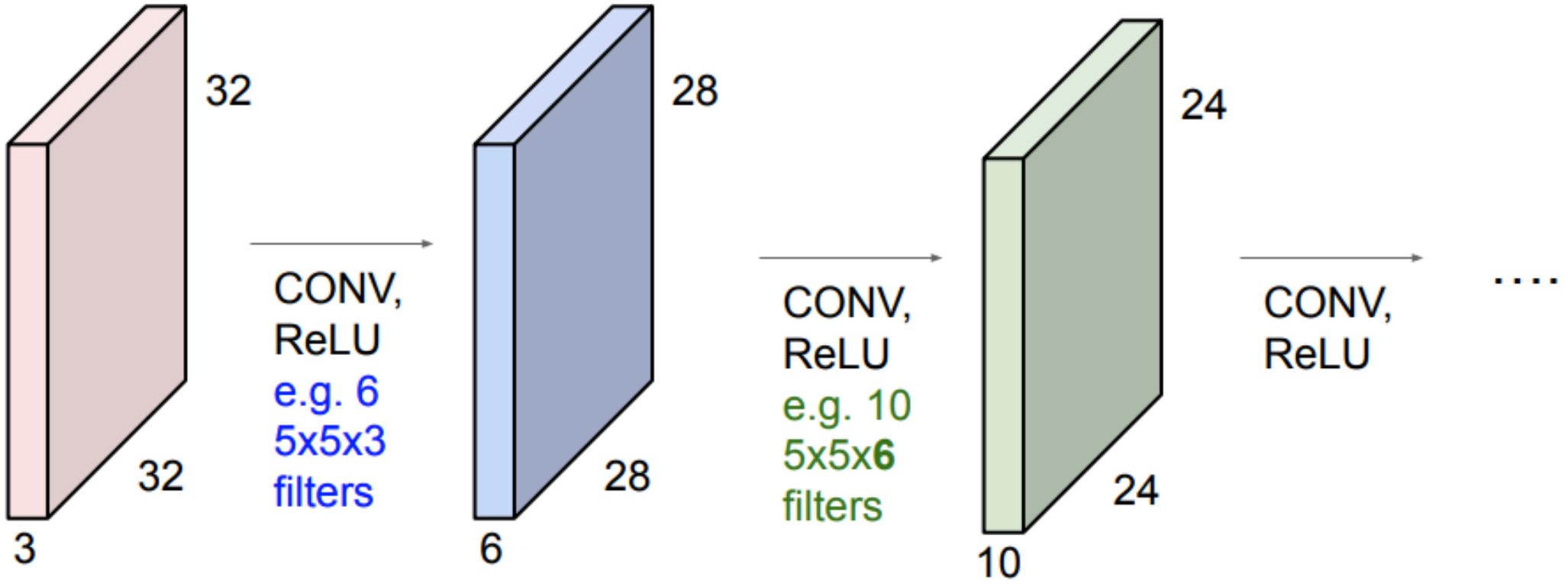
Convolutional Neural Network

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

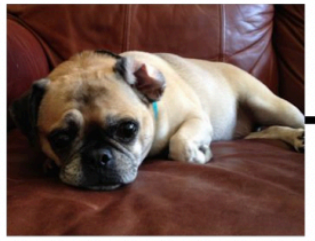


Feature Visualization

Preview

[Zeiler and Fergus 2013]

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

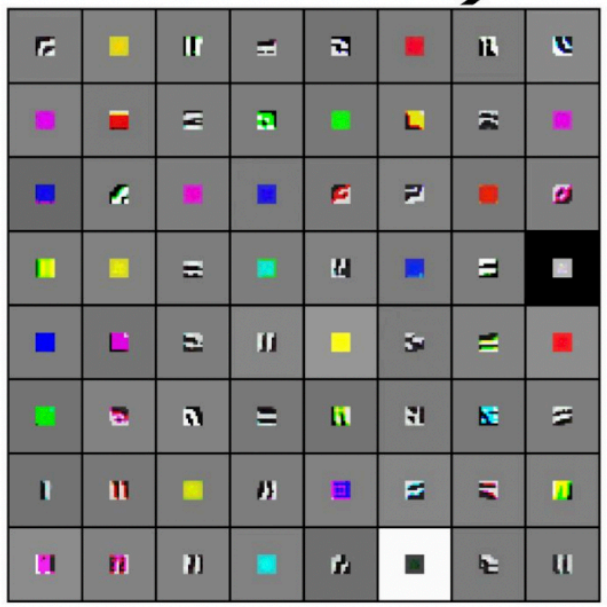


Low-level features

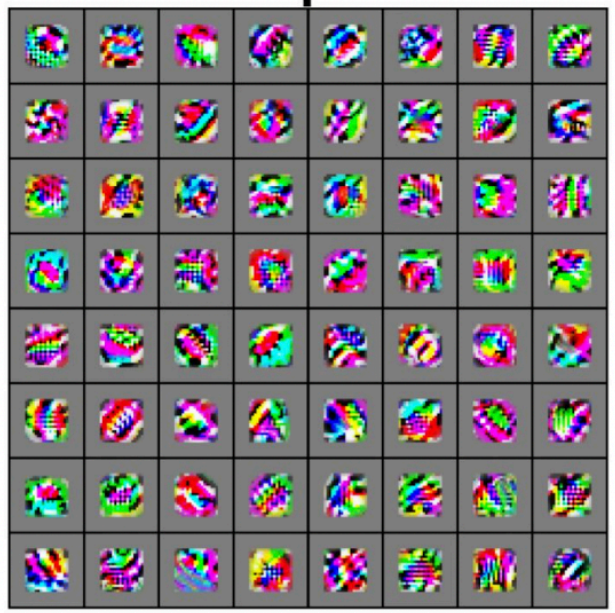
Mid-level features

High-level features

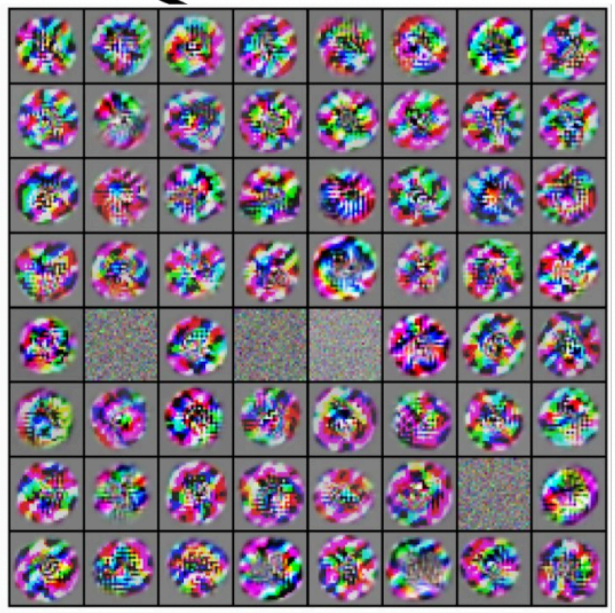
Linearly separable classifier



VGG-16 Conv1_1

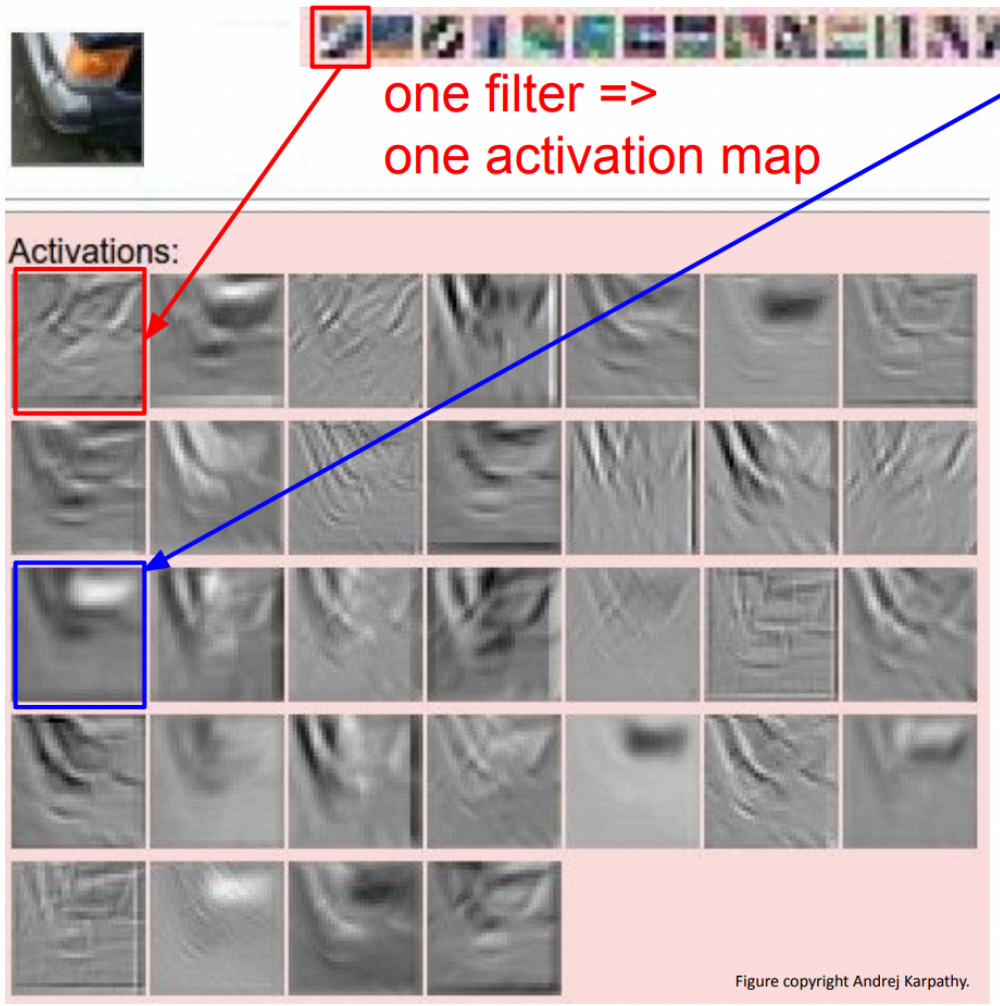


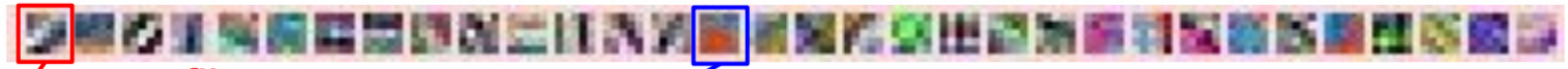
VGG-16 Conv3_2



VGG-16 Conv5_3

Feature Visualization





example 5x5 filters
(32 total)

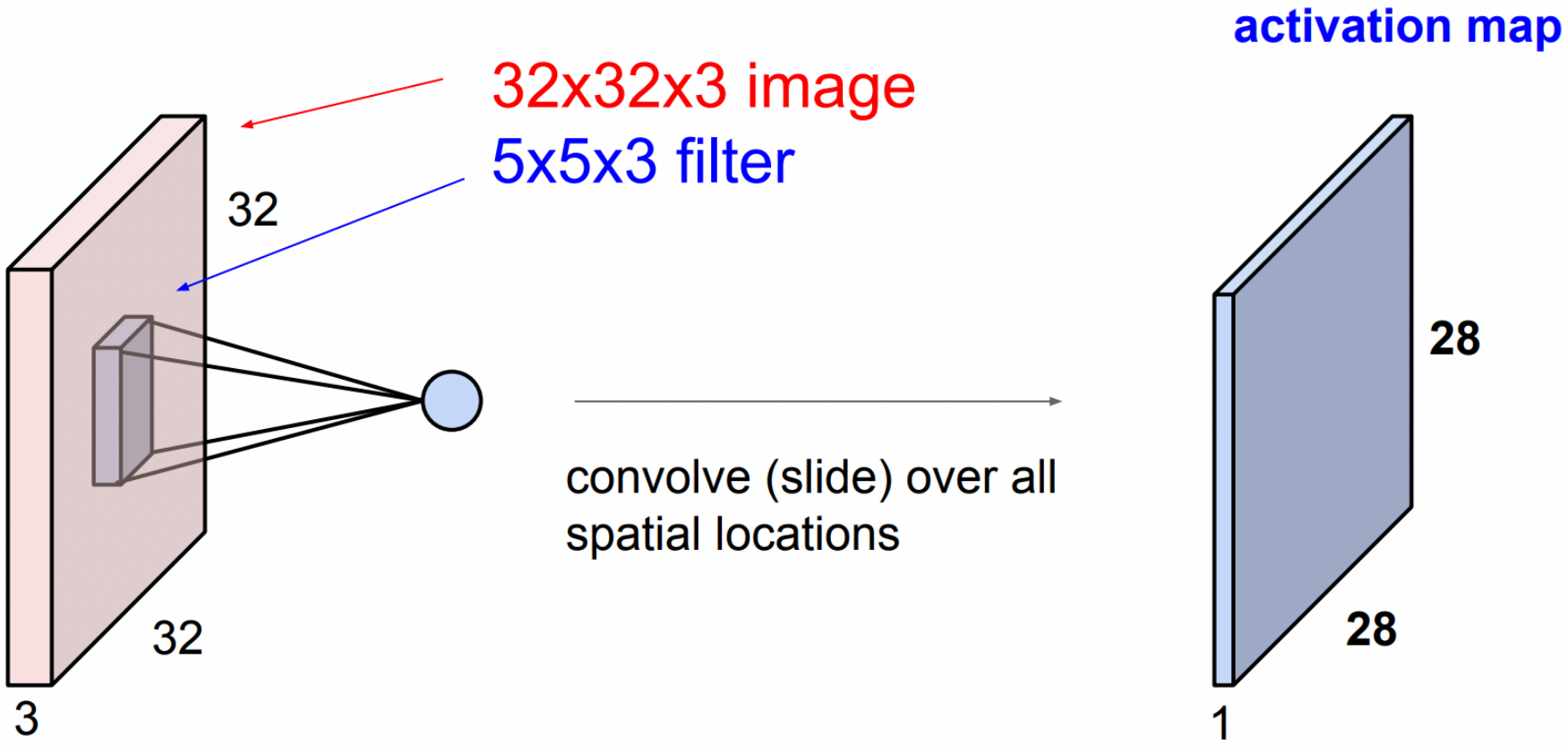
We call the layer convolutional because it is related to convolution of two signals:

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

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

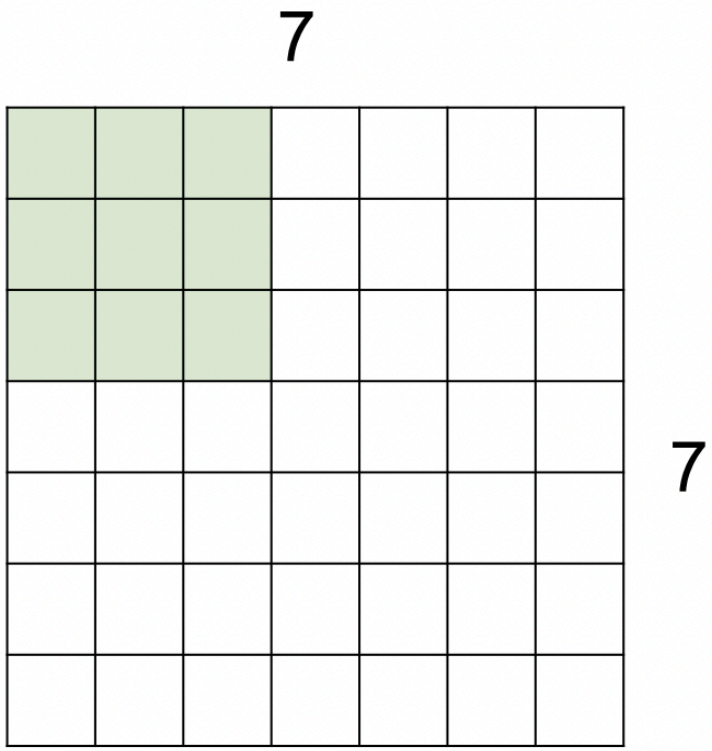
Convolutional Layer

A closer look at spatial dimensions:



Convolutional Layer

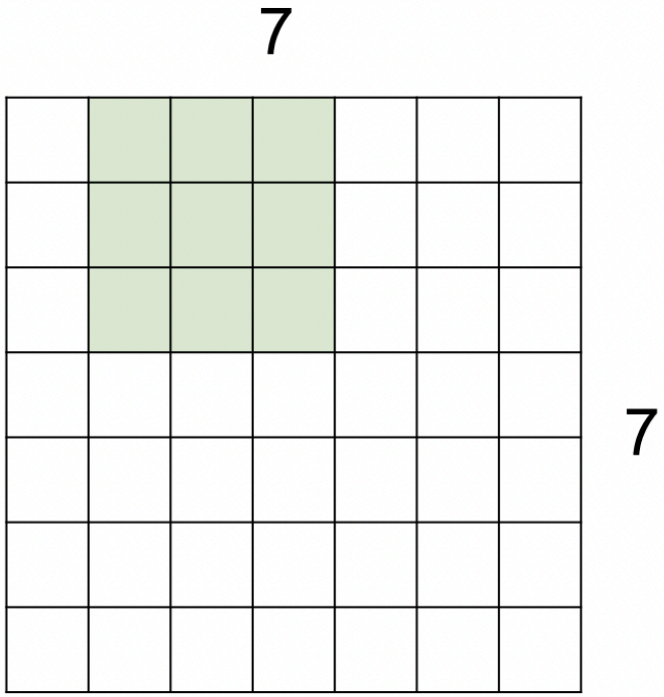
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolutional Layer

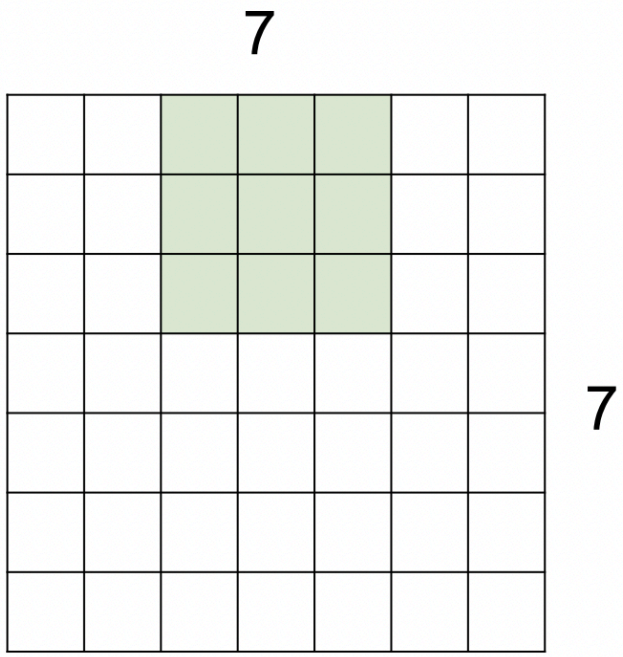
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolutional Layer

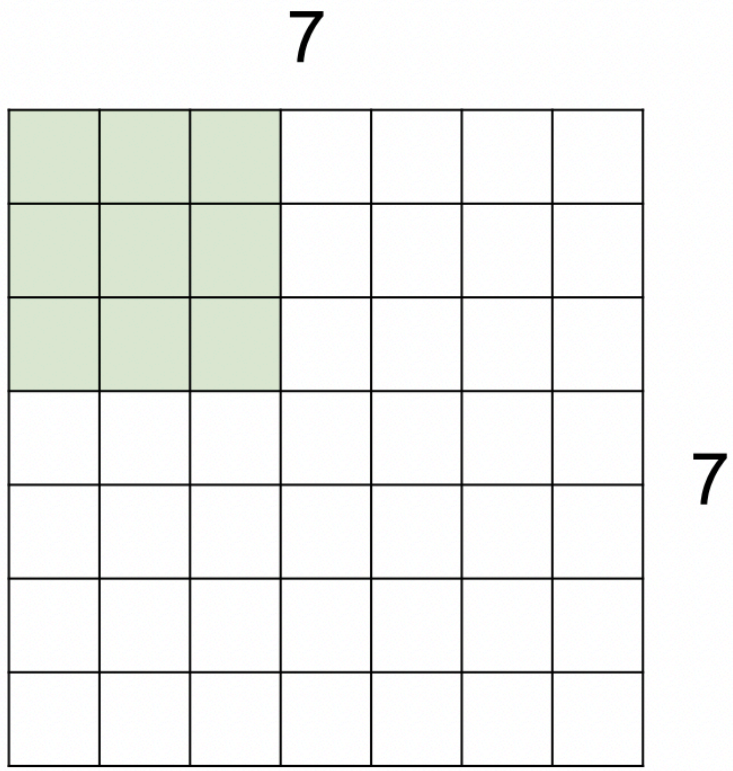
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Convolutional Layer

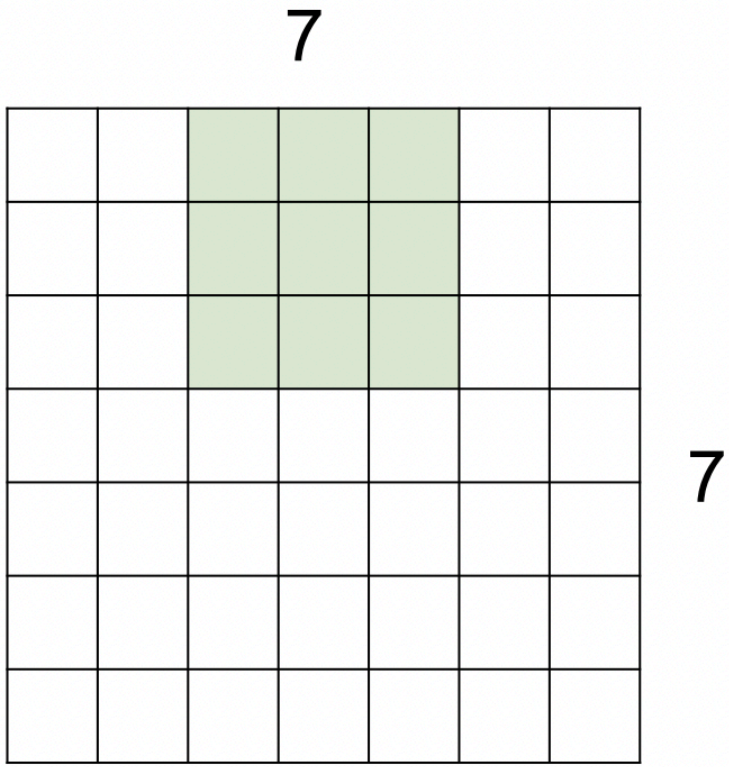
A closer look at spatial dimensions:



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

Convolutional Layer

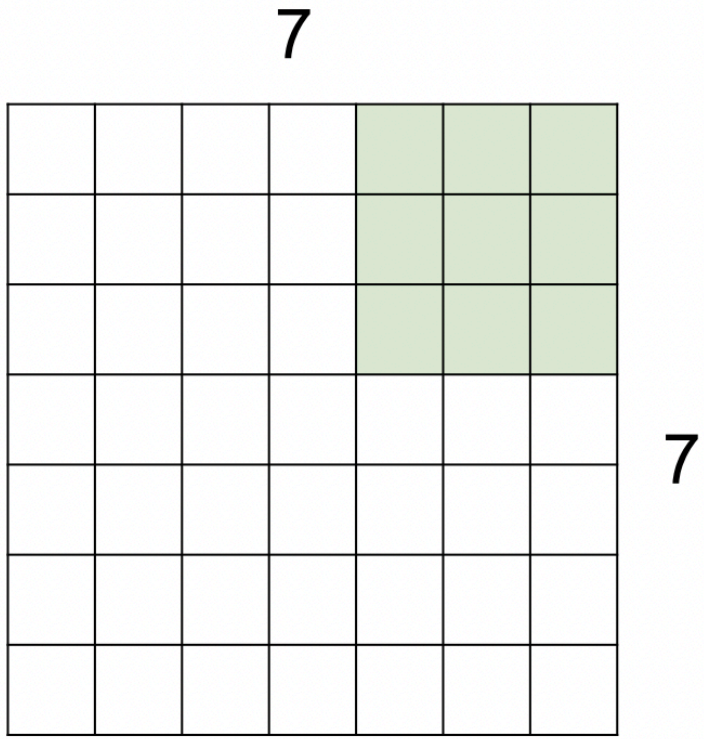
A closer look at spatial dimensions:



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

Convolutional Layer

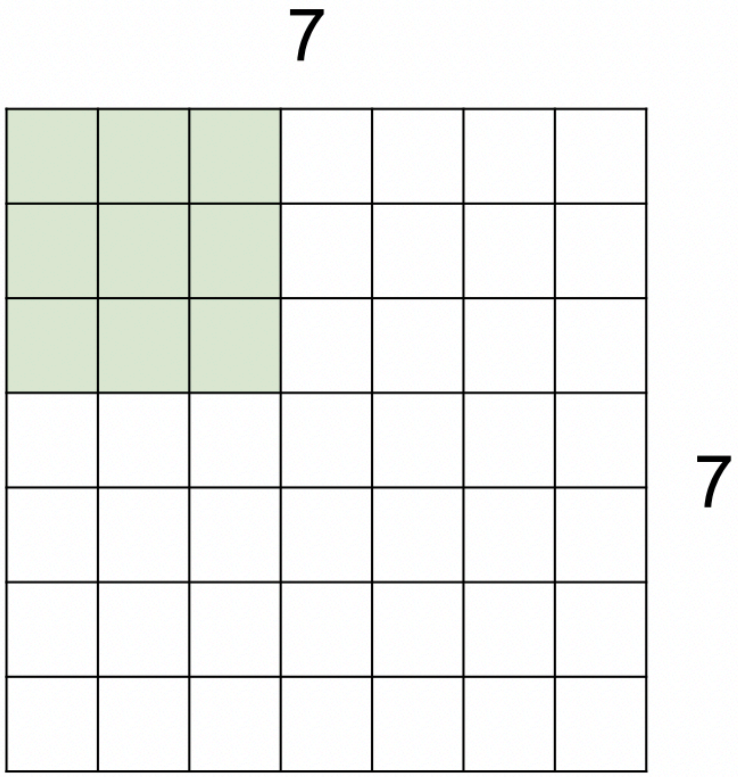
A closer look at spatial dimensions:



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

Convolutional Layer

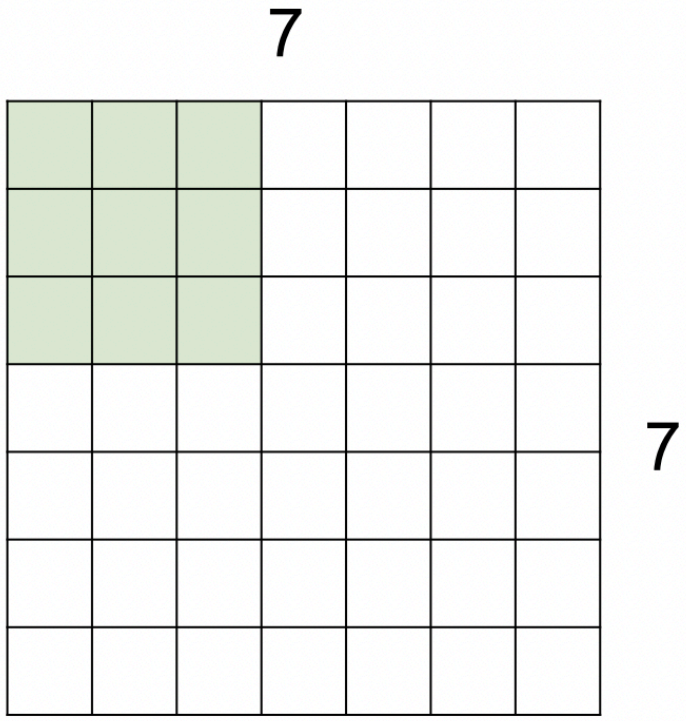
A closer look at spatial dimensions:



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

Convolutional Layer

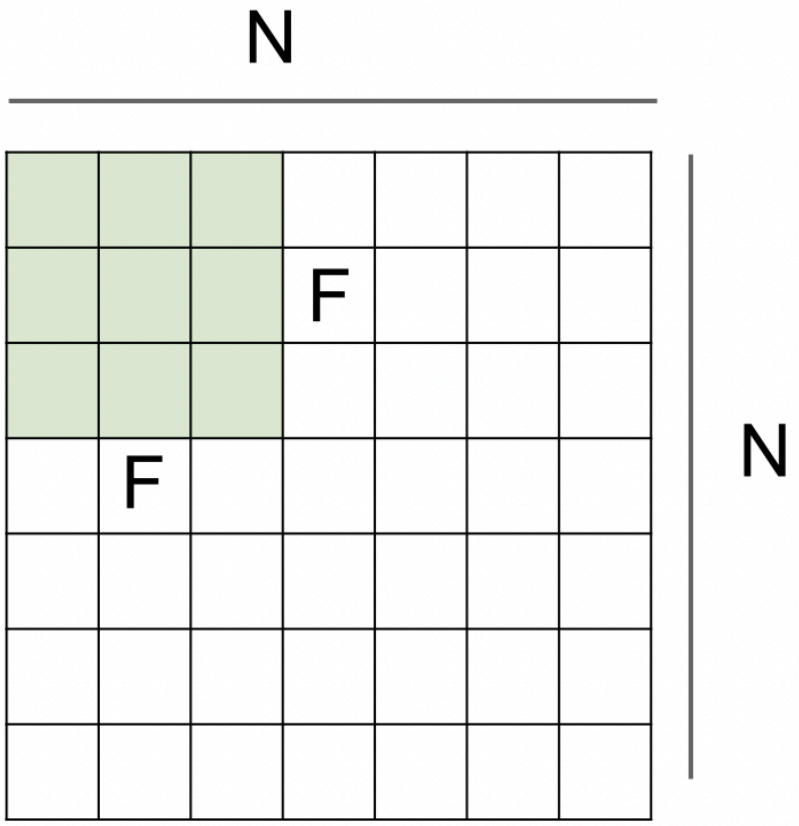
A closer look at spatial dimensions:



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

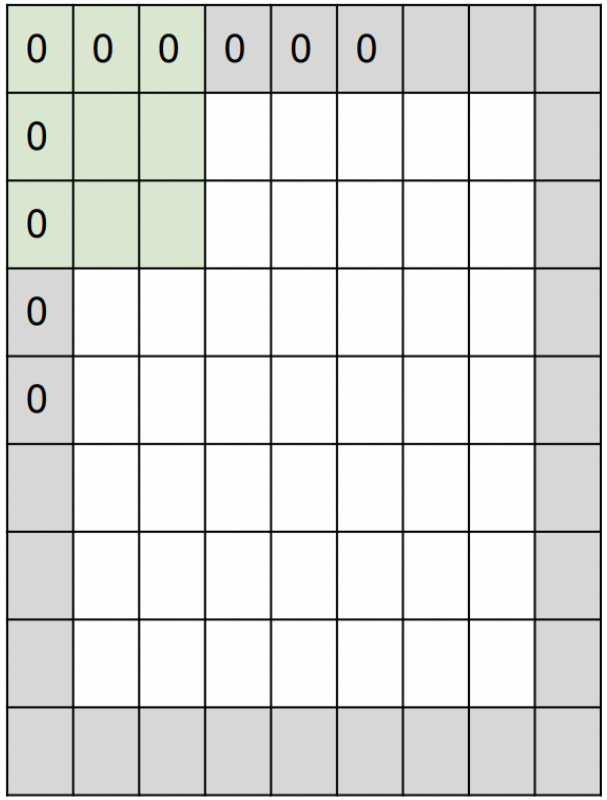


Output size:
(N - F) / stride + 1

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

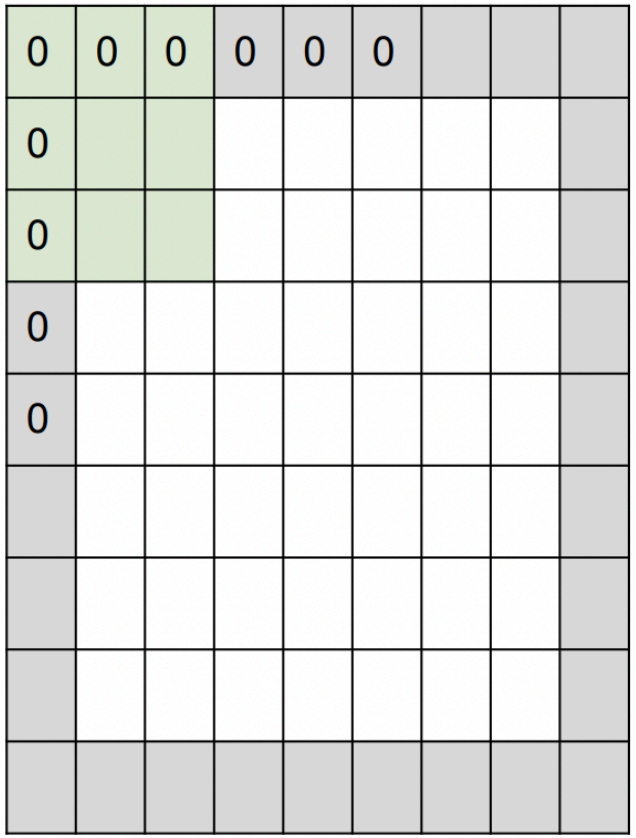


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

Padding

In practice: Common to zero pad the border



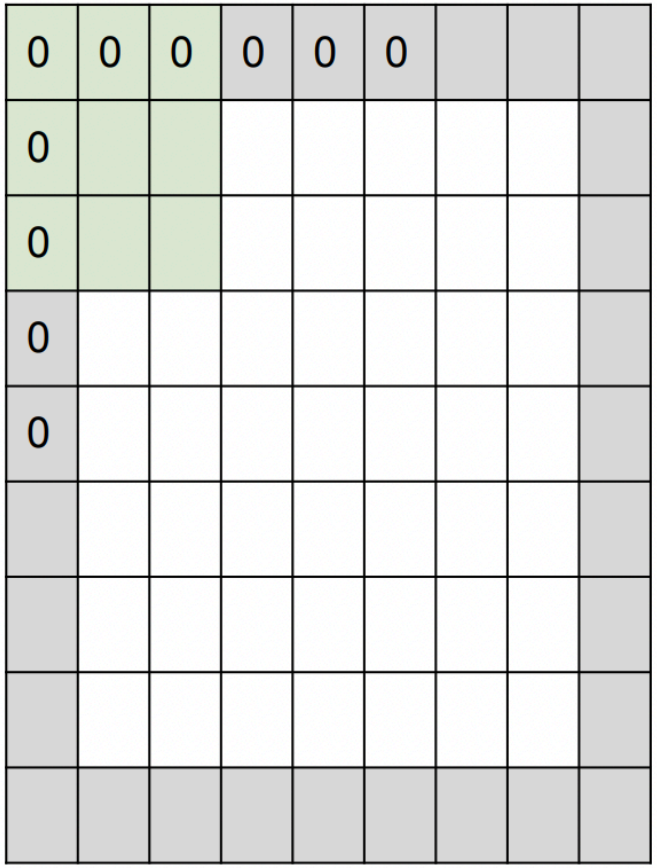
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

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

Padding

In practice: Common to zero pad the border

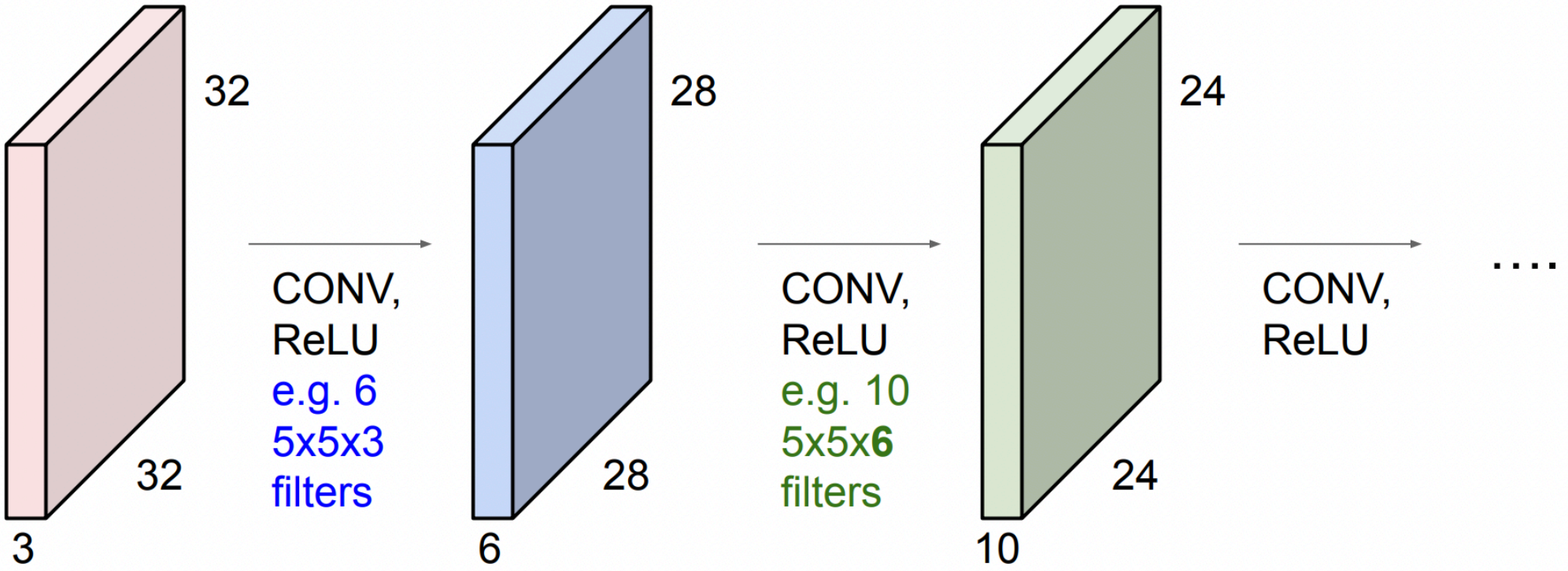


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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.

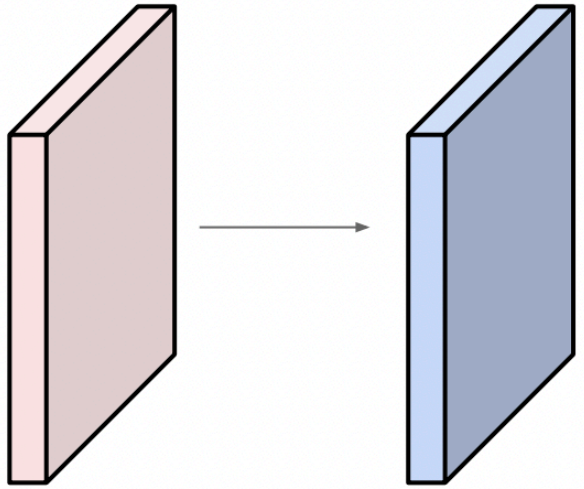


Convolutional Layer

Examples time:

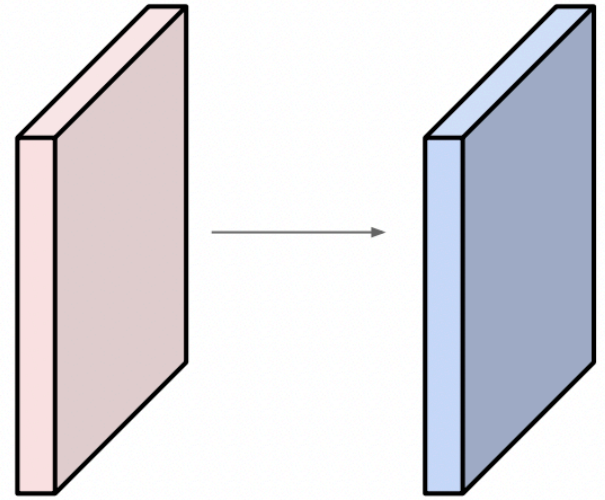
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?



Convolutional Layer

Examples time:



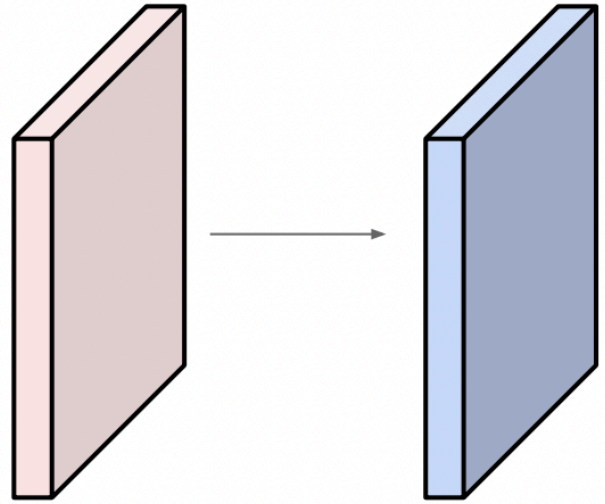
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

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

Convolutional Layer

Examples time:

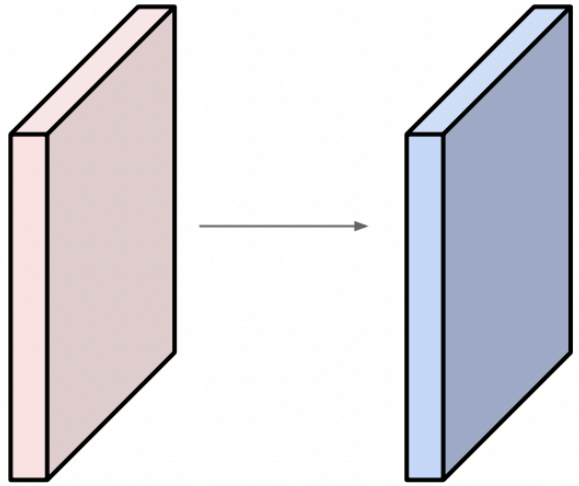
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



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

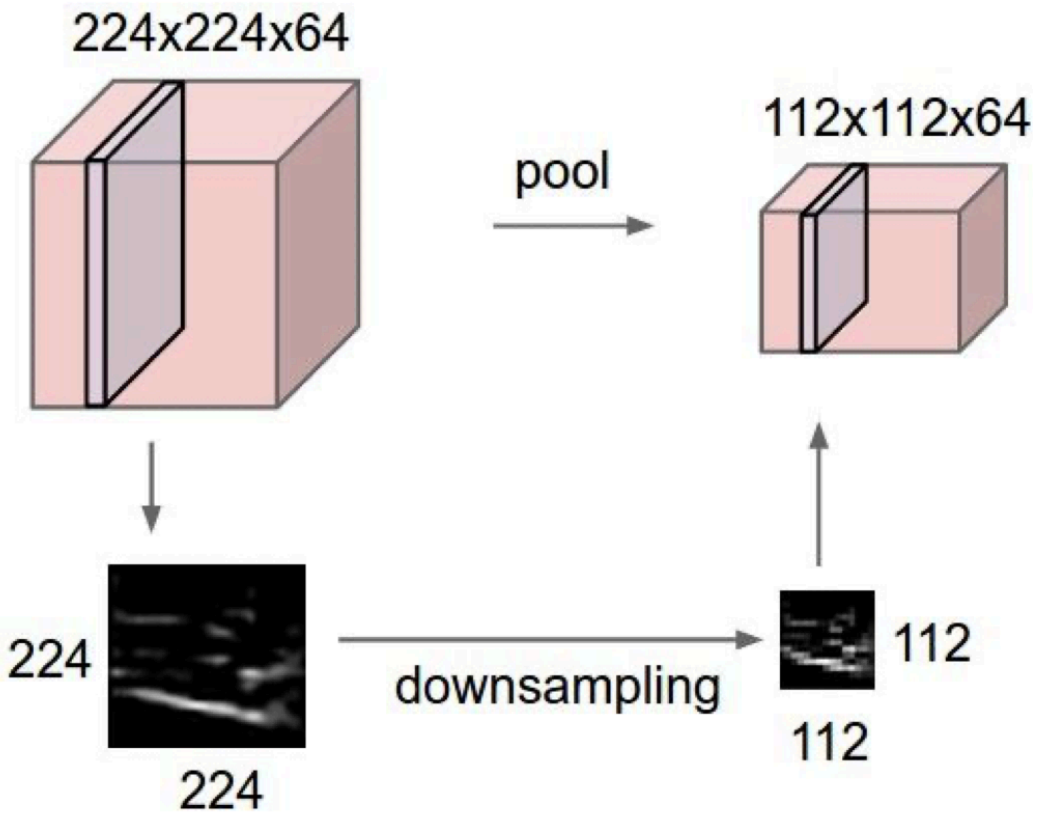
Common settings:

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

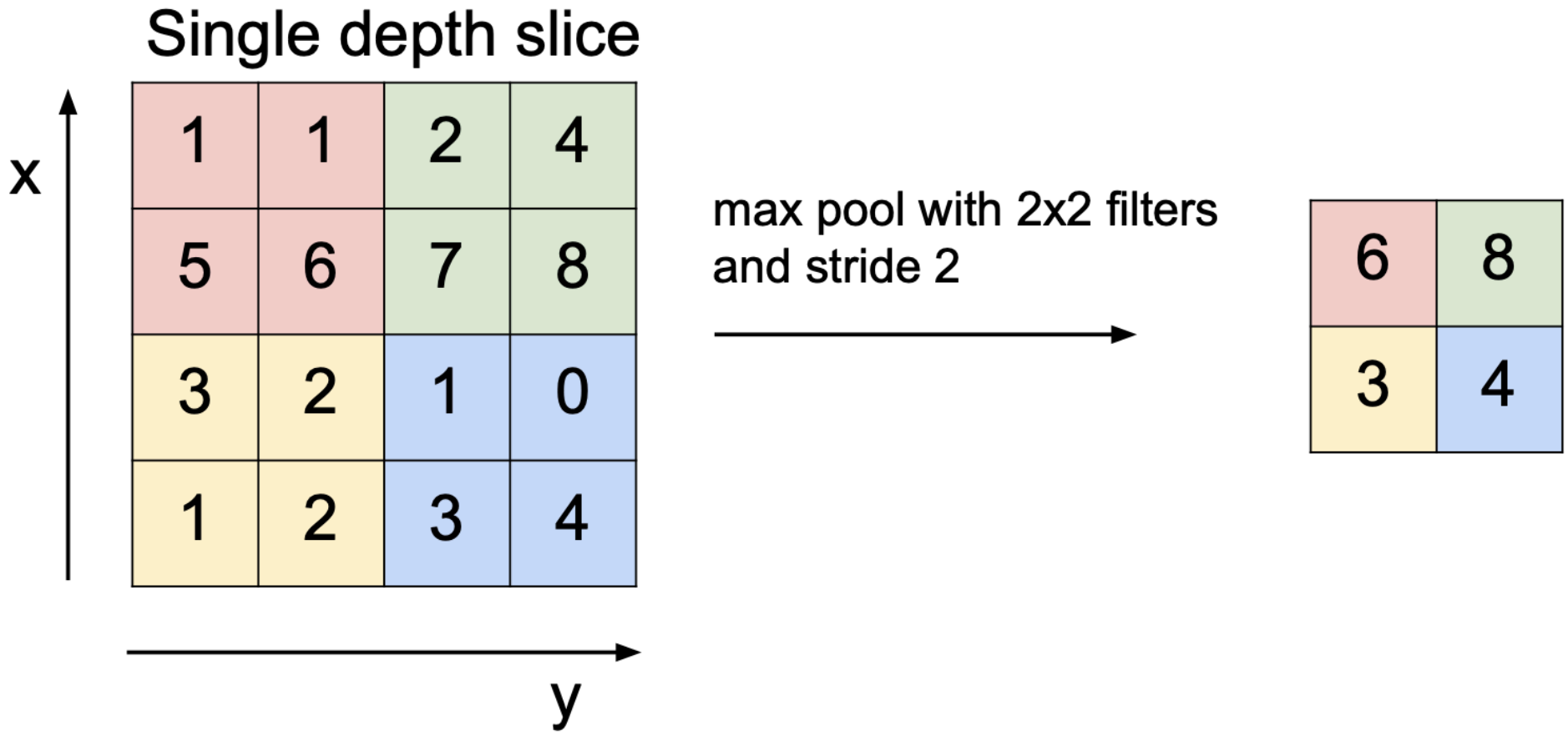
Pooling

- 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

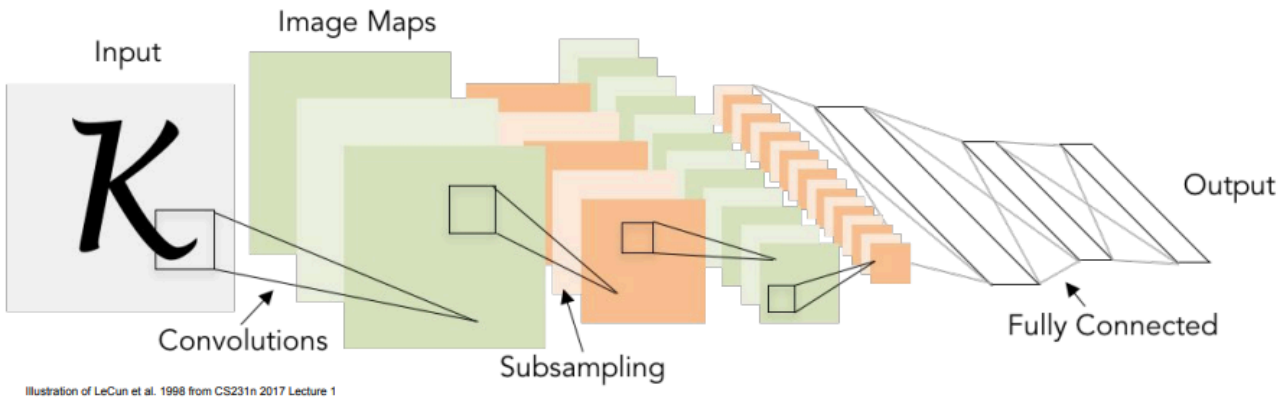
- Let's assume input is $W_1 \times H_1 \times C$
- 2×2 pooling will result in
  - $W_2 = \lceil W_1/2 \rceil$
  - $H_2 = \lceil H_2/2 \rceil$
- Number of parameters: 0

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.

Next week: Lecture 5,
Deep Learning II