Introduction to Computer Vision

Lecture 4 - Deep Learning I

Prof. He Wang

Logistics

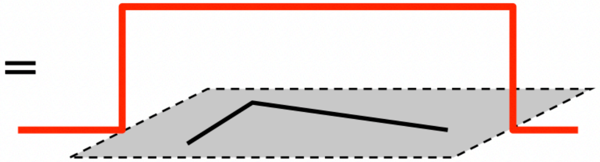
- Assignment 1: to release on 3/14 (this Friday evening), due on 3/29 11:59PM (Saturday)
  - Implementing convolution operation
  - Canny edge detector
  - Harris corner detector
  - Plane fitting using RANSAC

- 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.

Choices of Window Functions

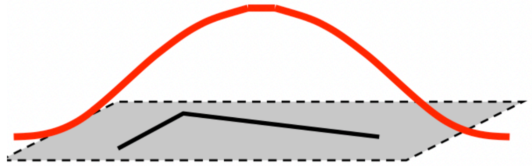
Rectangle "hard" window

$w(x,y) =$ 

1 in window, 0 outside

or

Isotropic "soft" window

$g_\sigma(x,y) =$ 

Gaussian

w is not rotation-invariant.

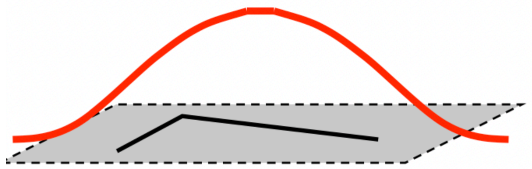
g_σ is rotation-invariant.

$$M(x_0, y_0) = \begin{bmatrix} w * I_x^2 & w * I_x I_y \\ w * I_x I_y & w * I_y^2 \end{bmatrix}$$

$$M(x_0, y_0) = \begin{bmatrix} g_\sigma * I_x^2 & g_\sigma * I_x I_y \\ g_\sigma * I_x I_y & g_\sigma * I_y^2 \end{bmatrix}$$

Using Gaussian Filter

$g_\sigma(x, y) =$



Gaussian

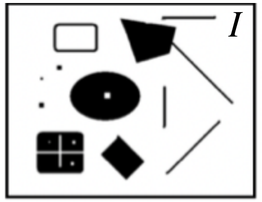
$$M(x_0, y_0) = \begin{bmatrix} g_\sigma * I_x^2 & g_\sigma * I_x I_y \\ g_\sigma * I_x I_y & g_\sigma * I_y^2 \end{bmatrix} = \begin{bmatrix} g(I_x^2) & g(I_x I_y) \\ g(I_x I_y) & g(I_y^2) \end{bmatrix}$$

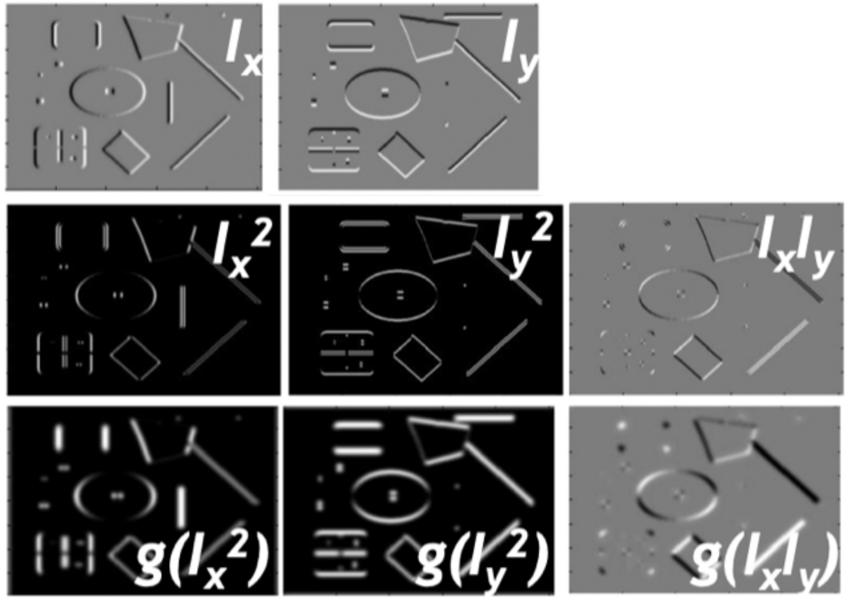
$$\therefore\ \theta(x_0, y_0) = \det(M(x_0, y_0)) - \alpha Tr(M(x_0, y_0))^2 - t$$

$$= (g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2) - \alpha[g(I_x^2) + g(I_y^2)]^2 - t$$

The Whole Process of Harris Detector

1. Image derivatives





2. Square of derivatives

3. Rectangle window or Gaussian filter

4. Corner response function

$$\theta = g(I_x^2)g(I_y^2) - [g(I_xI_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2 - t$$

5. Thresholding to obtain a binary mask $\theta(x_0, y_0) > 0$

6. Non-maximum suppression

Properties of Harris Detector

- Corner response is equivariant with both translation and image rotation.

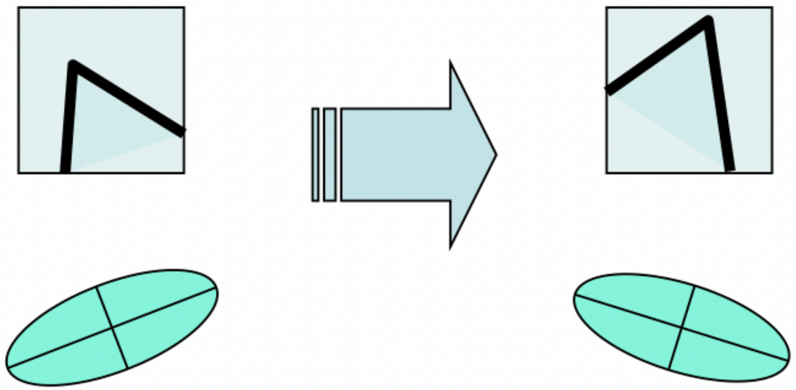


Image borrowed from Stanford CS131

Equivariance and Invariance

If $X \in V$, and $f : V \to V$ is a function,

and $T : V \to V$ is a transformation operating X, $e.g.$, translation,

Definitions:

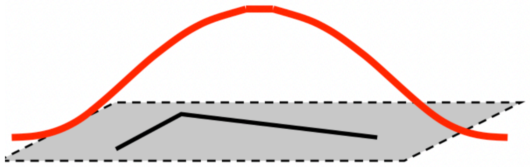
f to be **equivariant** under T if

$$T[f(X)] = f(T(X))$$

f to be **invariant** under T if

$$f(X) = f(T(X))$$

Check for Equivariance and Invariance

$g_\sigma(x, y) =$ 

Gaussian

$$M(x_0, y_0) = \begin{bmatrix} g_\sigma * I_x^2 & g_\sigma * I_x I_y \\ g_\sigma * I_x I_y & g_\sigma * I_y^2 \end{bmatrix} = \begin{bmatrix} g(I_x^2) & g(I_x I_y) \\ g(I_x I_y) & g(I_y^2) \end{bmatrix}$$

$$\theta(x_0, y_0) = (g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2) - \alpha[g(I_x^2) + g(I_y^2)]^2 - t$$

Translation operation: $T_{u,v}\theta(x_0, y_0) = \theta(x_0 + u, y_0 + v)$

Rotation operation: $R_\phi \theta(x_0, y_0) = \theta(R_\phi [x_0, y_0]^T)$

θ is **equivariant** under both rotation and translation!

Step-by-Step Harris Detector

- Input: two images

Image borrowed from Stanford CS131

Step-by-Step Harris Detector

- Compute corner response θ

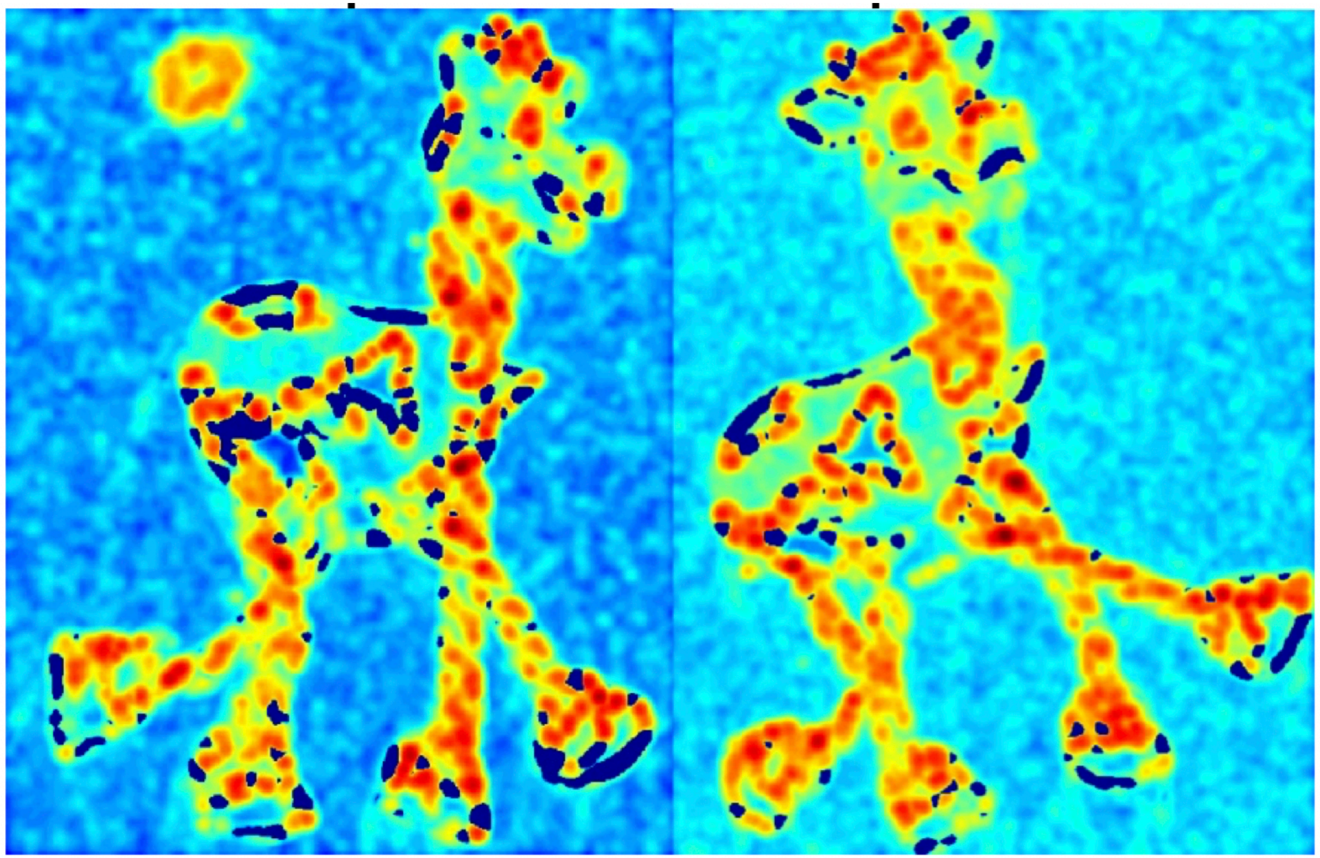


Image borrowed from Stanford CS131

Step-by-Step Harris Detector

- Thresholding and perform non-maximal suppression

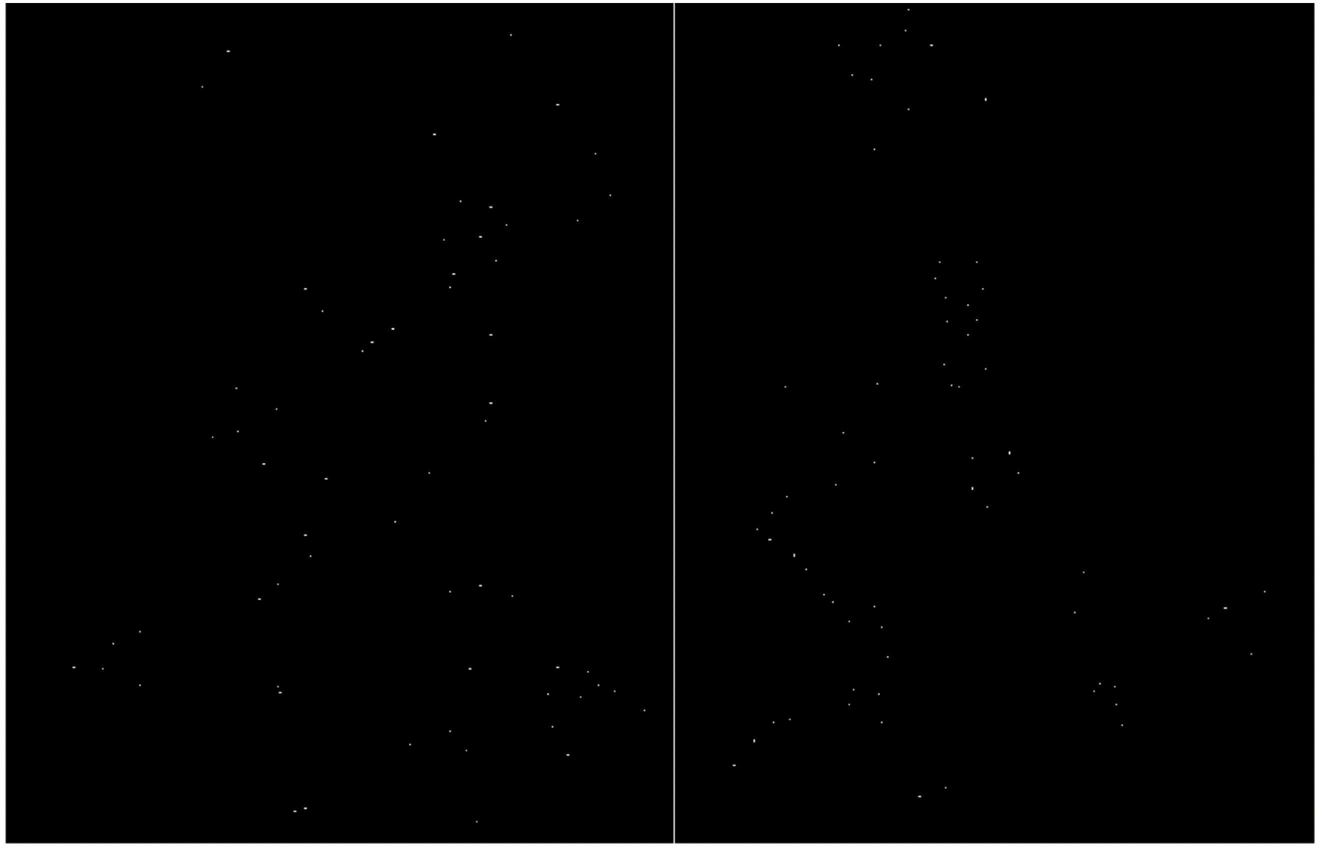


Image borrowed from Stanford CS131

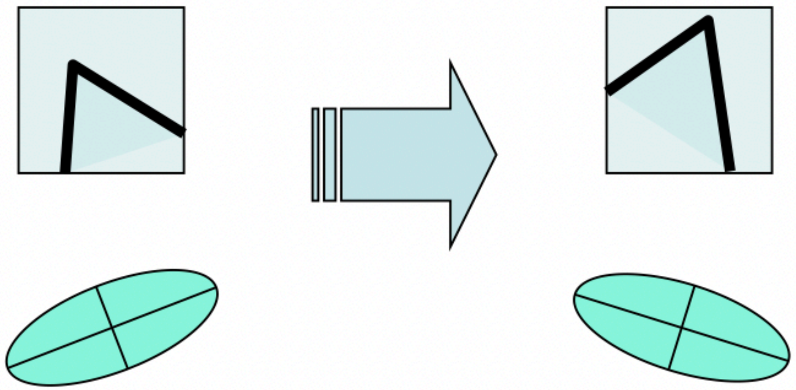
Step-by-Step Harris Detector

- Results

Image borrowed from Stanford CS131

Properties of Harris Detector

- Corner response is equivariant with both translation and image rotation.
- Not invariant to scale.



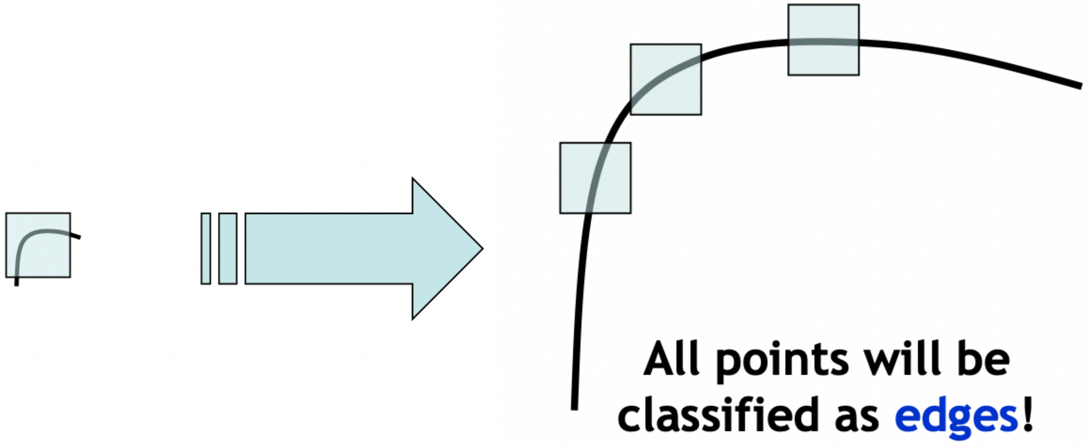
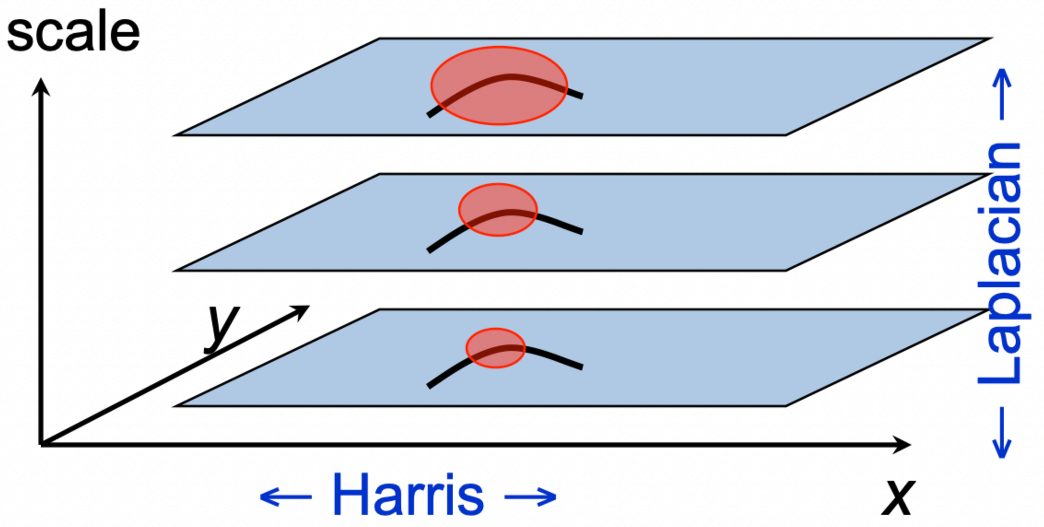


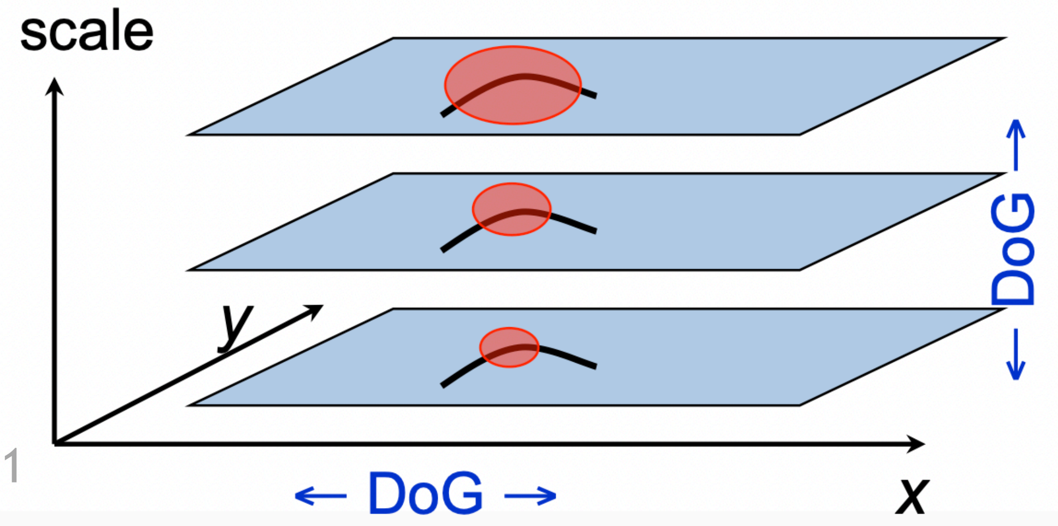
Image borrowed from Stanford CS131

Scale Invariant Detectors

- **Harris-Laplacian**[1]
  Find local maximum of:
  - Harris corner detector in space (image coordinates)
  - Laplacian in scale



- **SIFT (Lowe)**[2]
  Find local maximum of:
  - Difference of Gaussians in space and scale



Read by Yourself

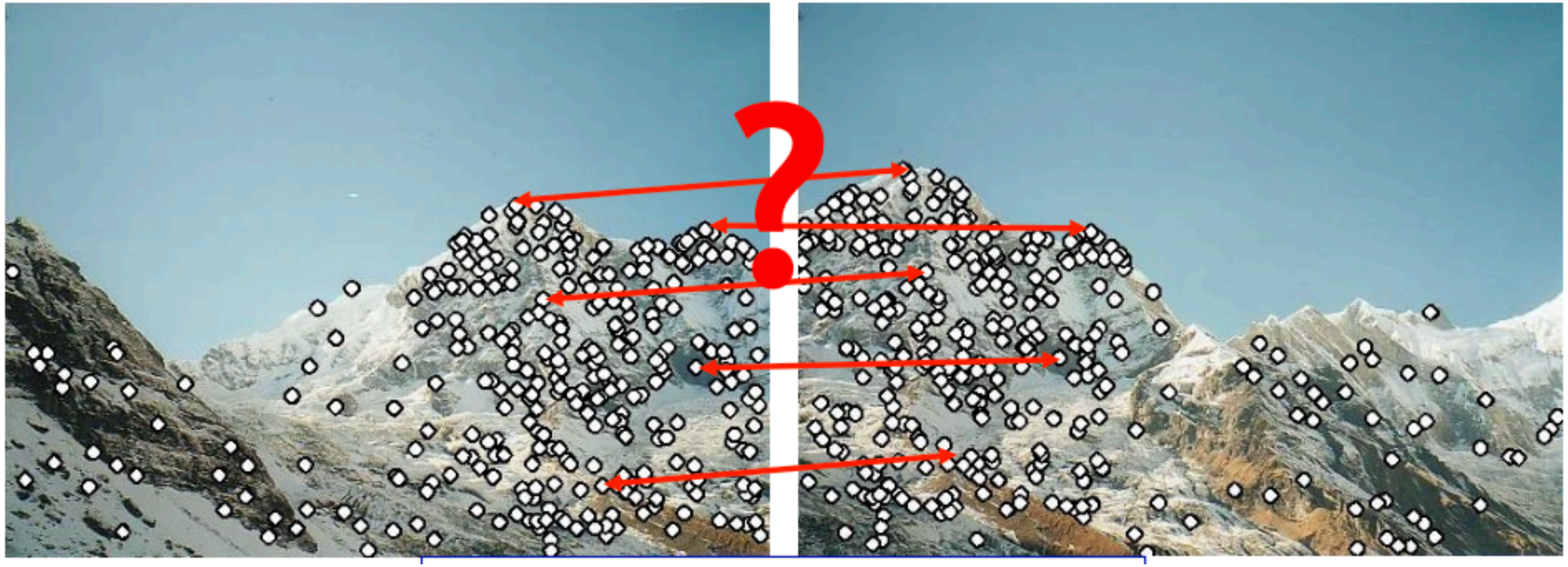
Slides borrowed from Stanford CS131

Feature Description

Local Descriptors

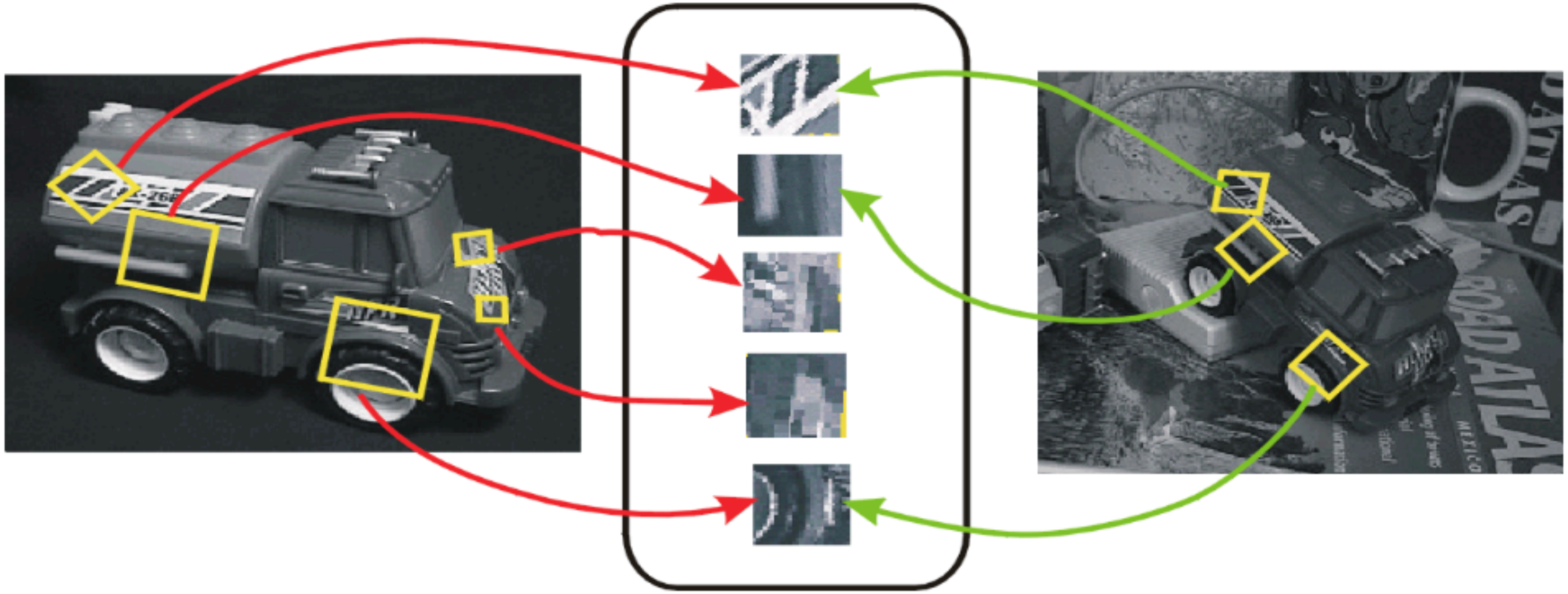
- We know how to detect points
- Next question:

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?

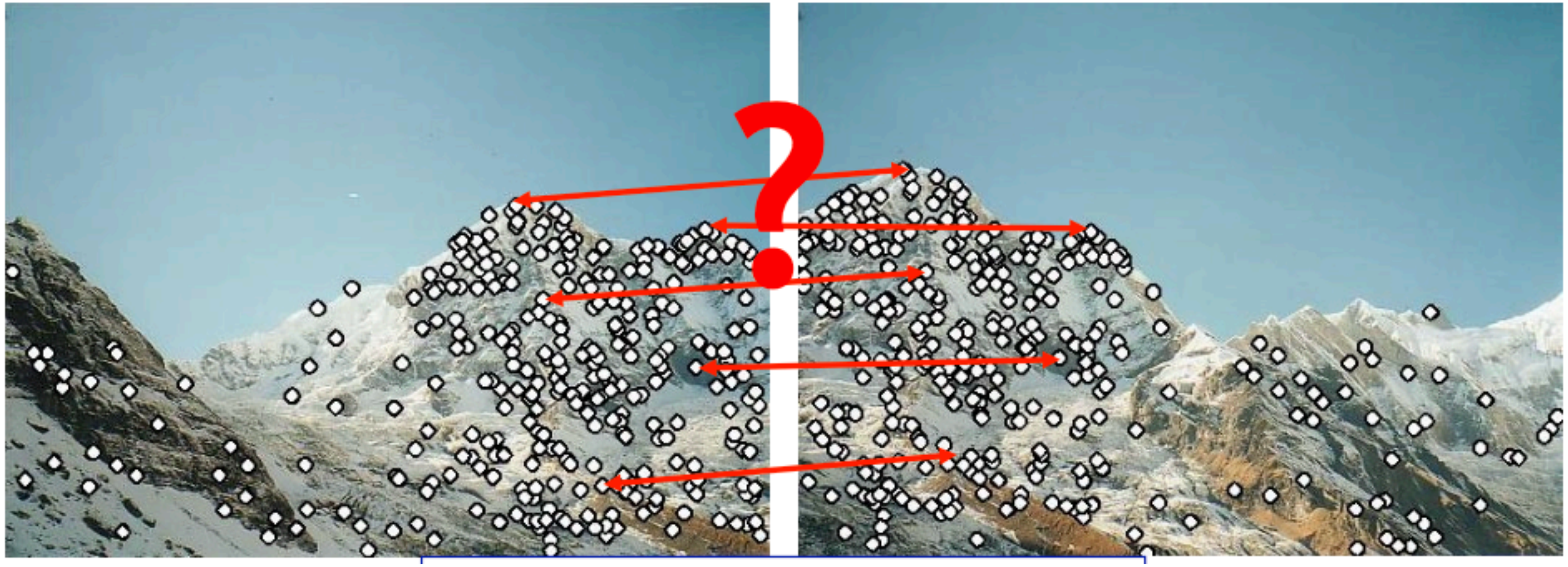
Topic Switch

- 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

Local Descriptors

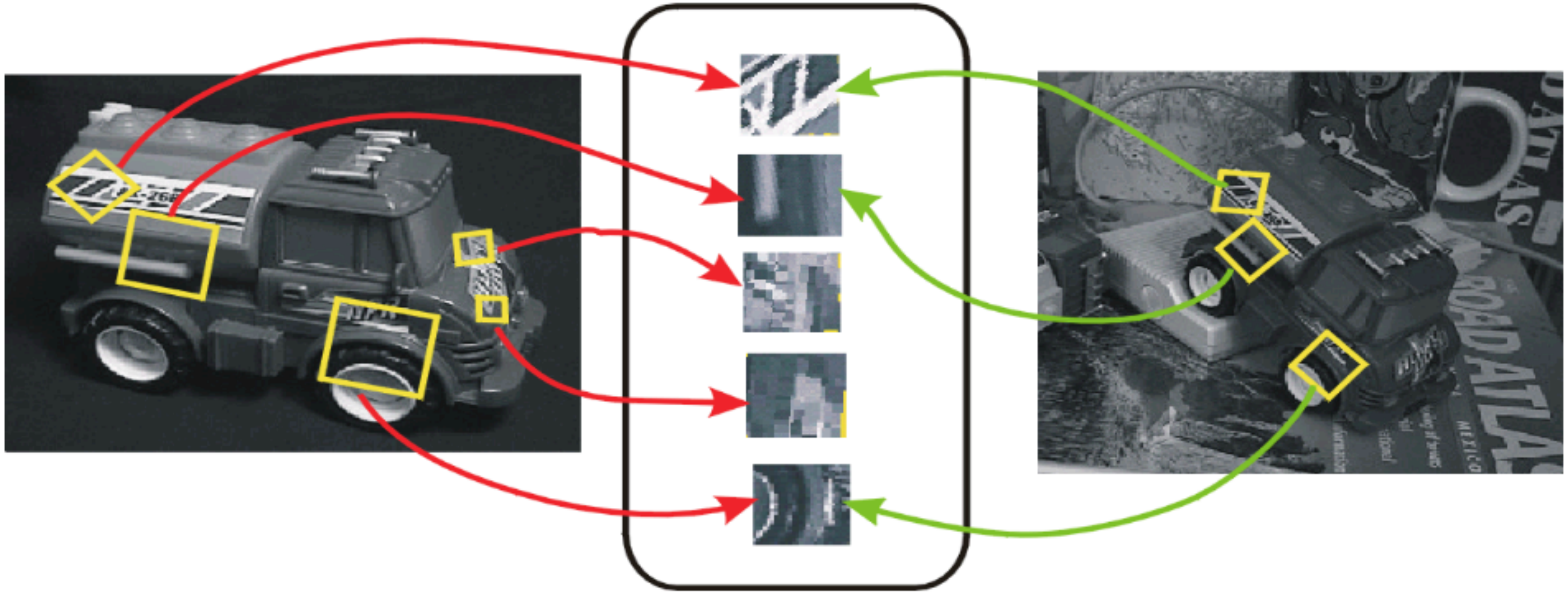
- We know how to detect points
- Next question:

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

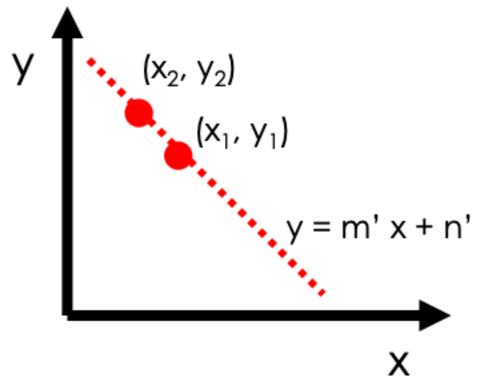
- Low-level vision
  - Image processing
  - Edge/corner detection
  - Feature extraction
- Mid-level Vision
  - Grouping
  - Inferring scene geometry (3D reconstruction)
  - Inferring camera and object motion
- **High-level vision (where deep learning wins!)**
  - Object recognition
  - Scene understanding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

Machine Learning 101

From Line Fitting to Neural Network Training

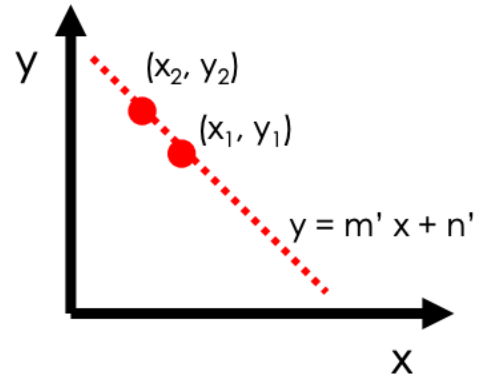
- When we have some observations $\{(x, y)\}$, we want to find the relationship behind y and x.

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



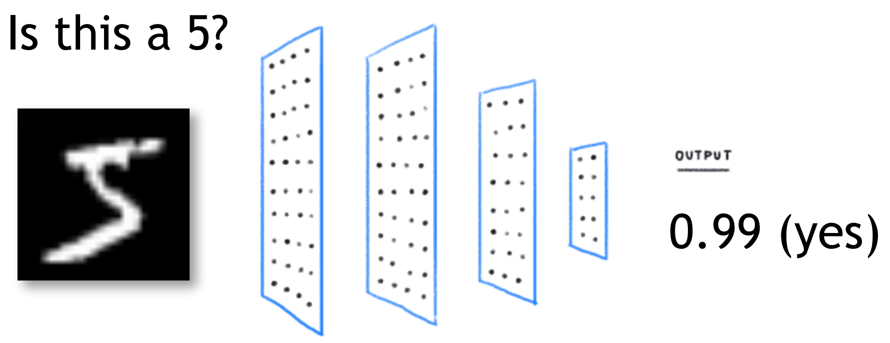
From Line Fitting to Neural Network Training

- 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

$x =$  $\longrightarrow$ $h($$; \theta)$ $\longrightarrow$?

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

Parameters or weights

Yes or No

MINIST Dataset of handwritten digits

Task: Binary Classification — Is This Digit a 5?

Data

$x =$ 

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

Model

$h($  $;\theta)$

Parameters or weights

Output

?

Probability of being a 5

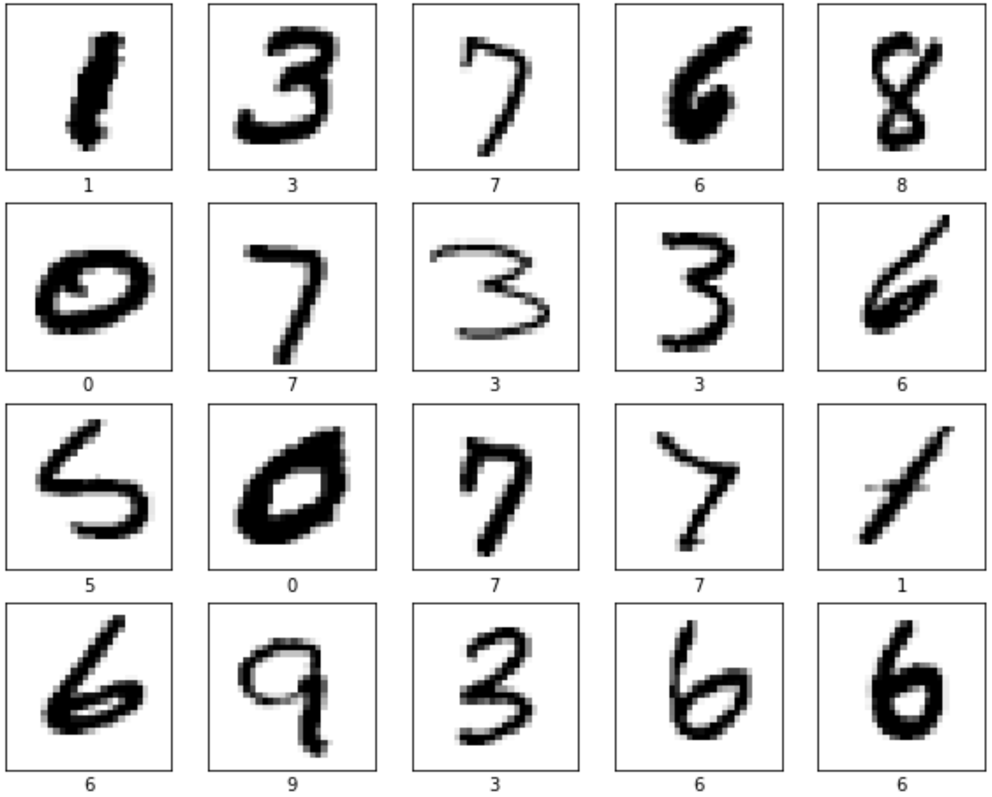
$p(y = 1 \mid x)$

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

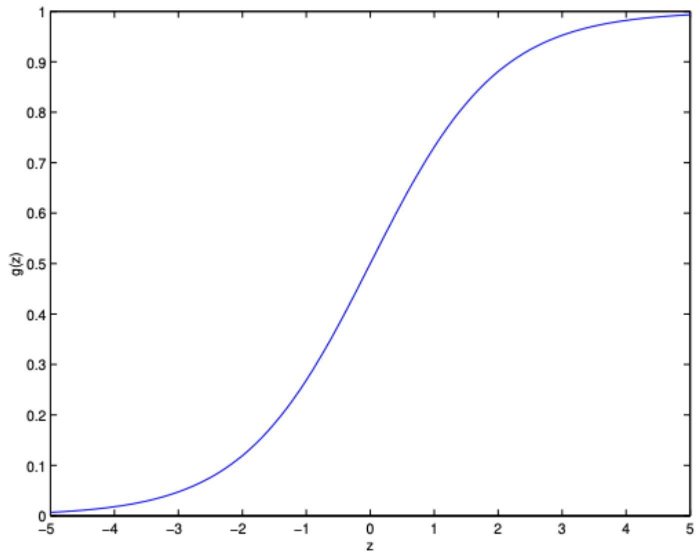
- Image 28×28: flatten to a one-dimensional vector $x \in \mathbb{R}^{784}$

- 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

$x =$  $\longrightarrow$ $h($$;\theta)$ $\longrightarrow$?

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

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

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

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

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

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

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

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

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

- objective and constraint functions are convex:

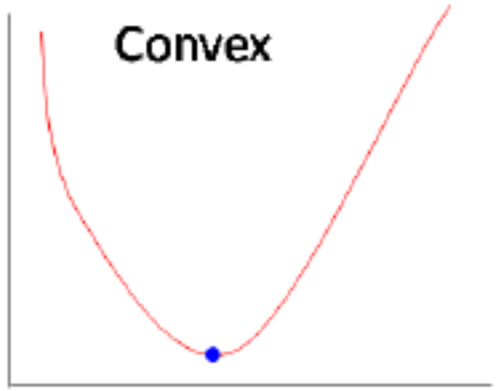
$$f_i(\alpha x + \beta y) \le \alpha f_i(x) + \beta f_i(y)$$

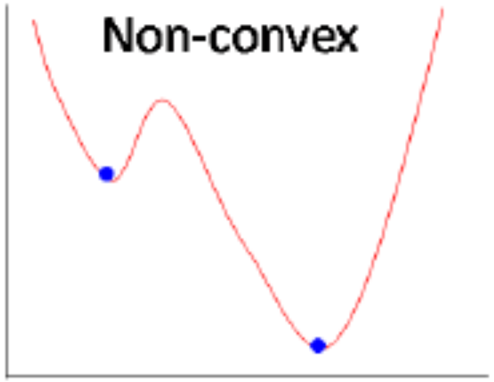
  if $\alpha + \beta = 1$, $\alpha \ge 0$, $\beta \ge 0$

- 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)

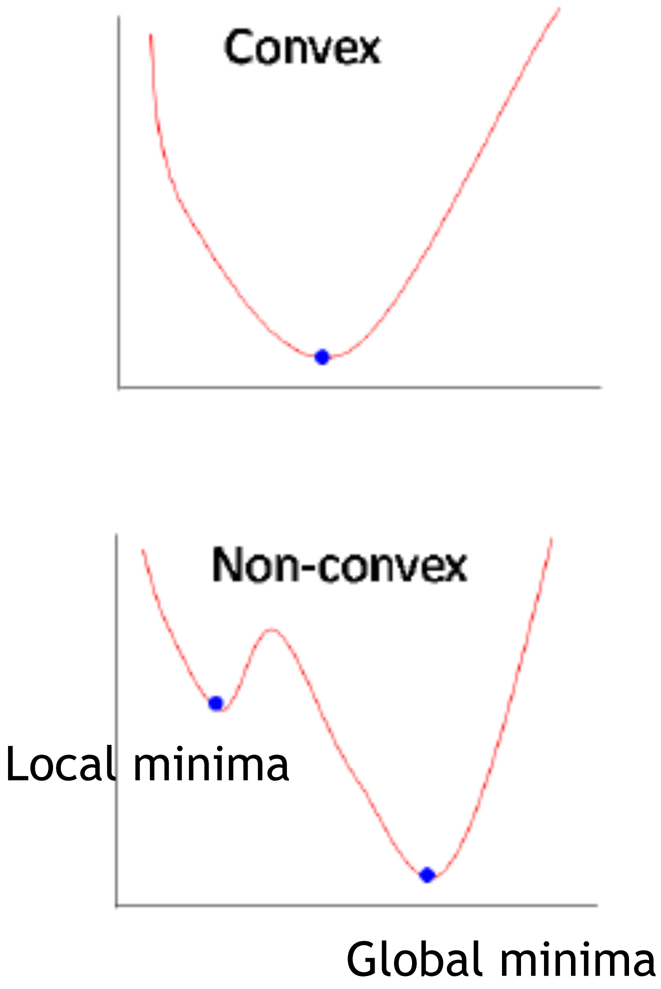
- Update rule for one iteration: $\theta := \theta - \alpha \nabla_{\theta} \mathscr{L}(\theta)$

- 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

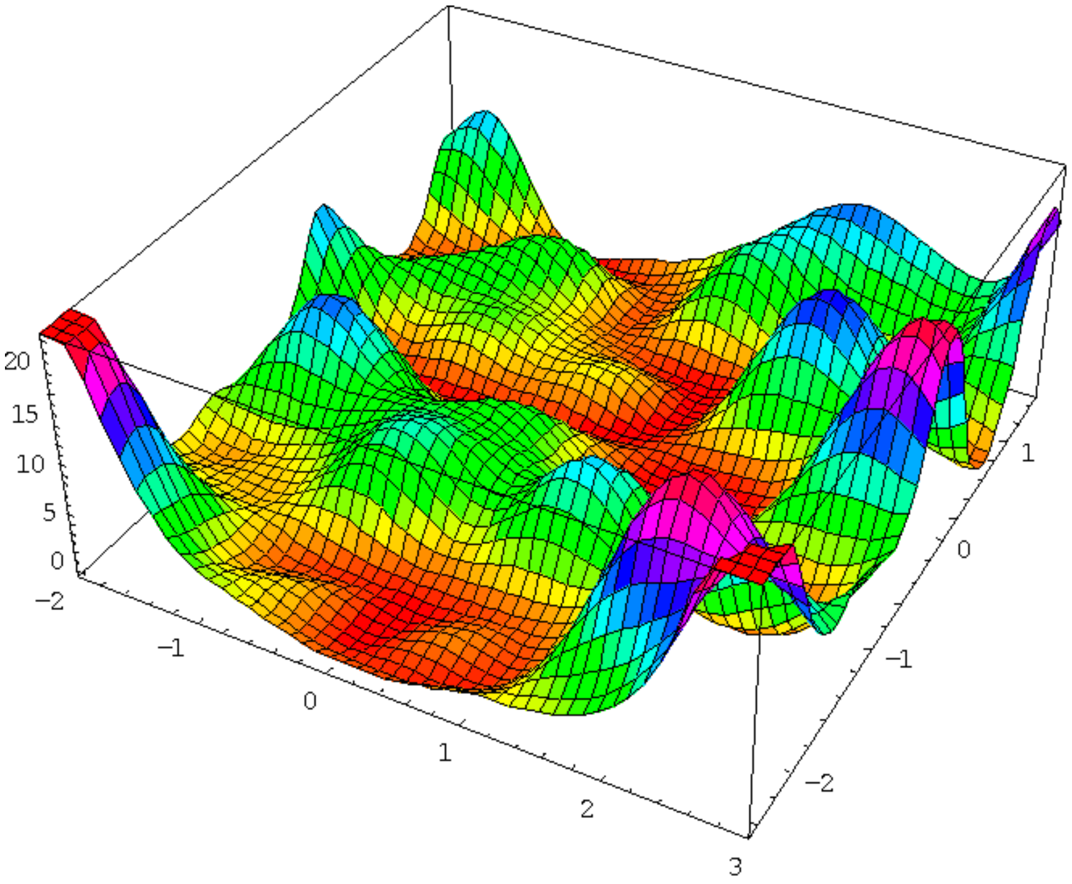
- How to perform GD to minimize NLL loss?

- 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)

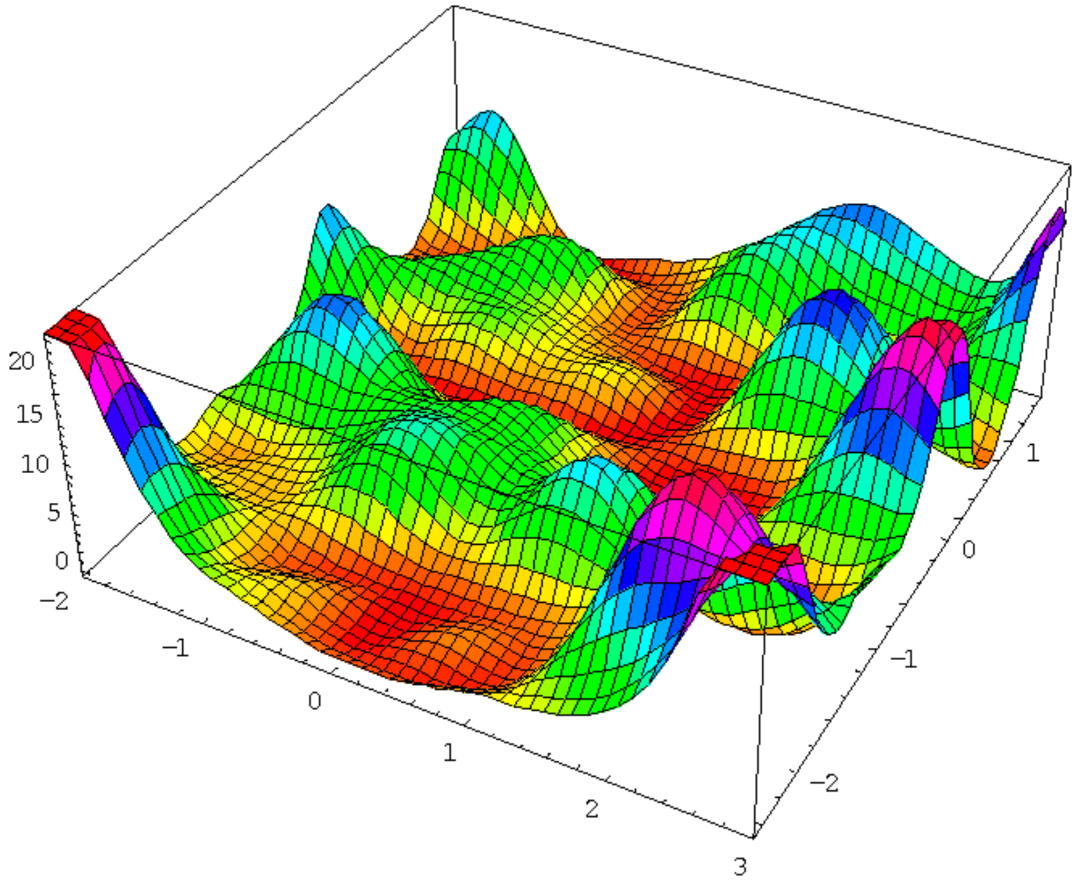
Randomly sample N pairs as a batch from the training data and then compute the average gradient from them.

+: 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.

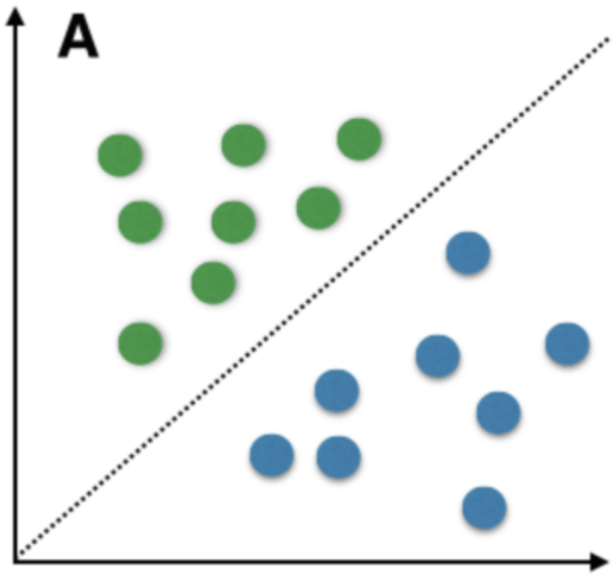
- Will we still work well?

Generalization gap!

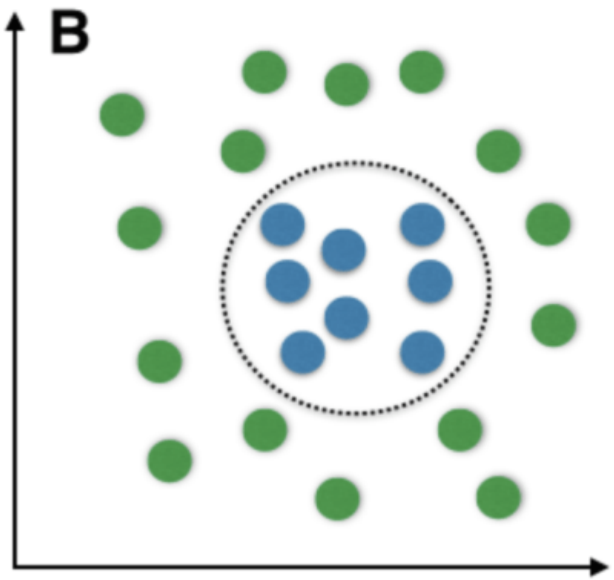
Multilayer Perceptron

Problem with Single-Layer Network

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

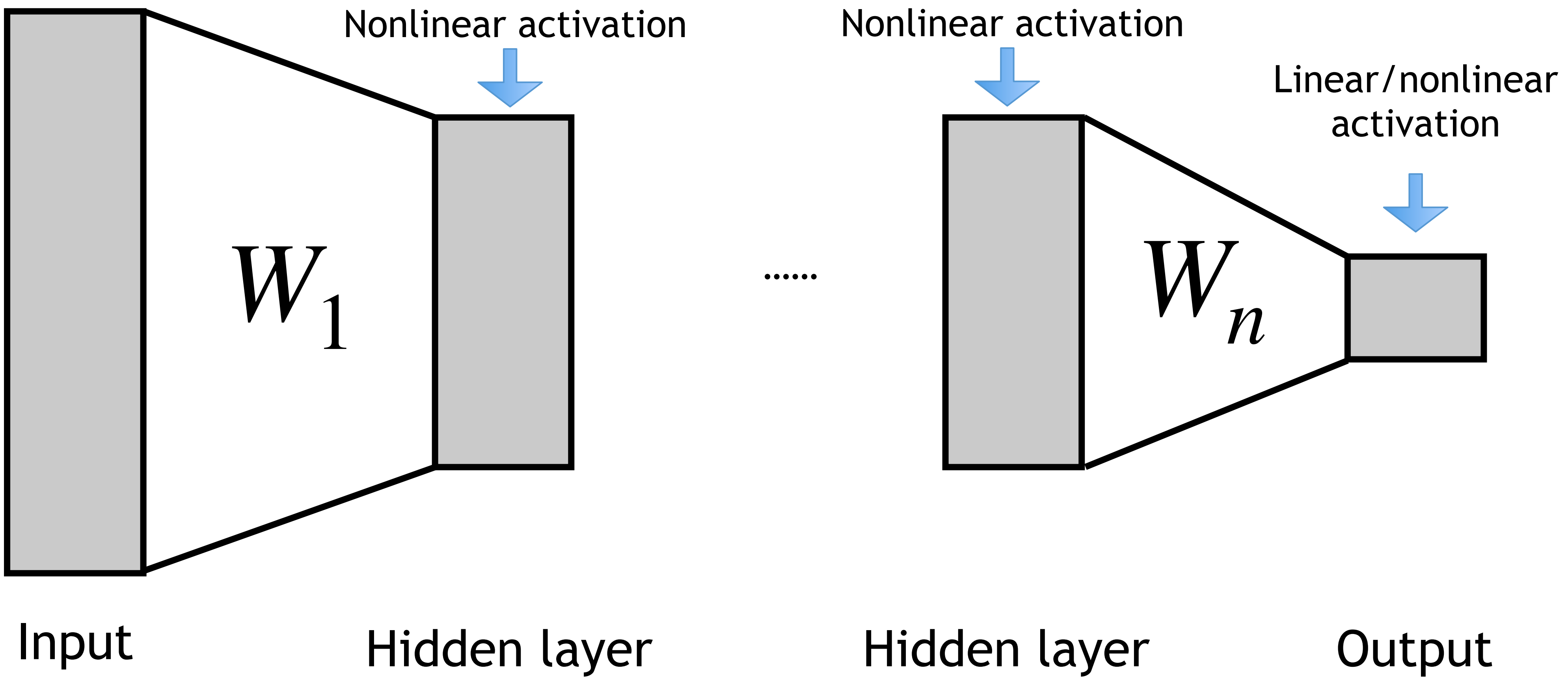


Linear separable



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

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

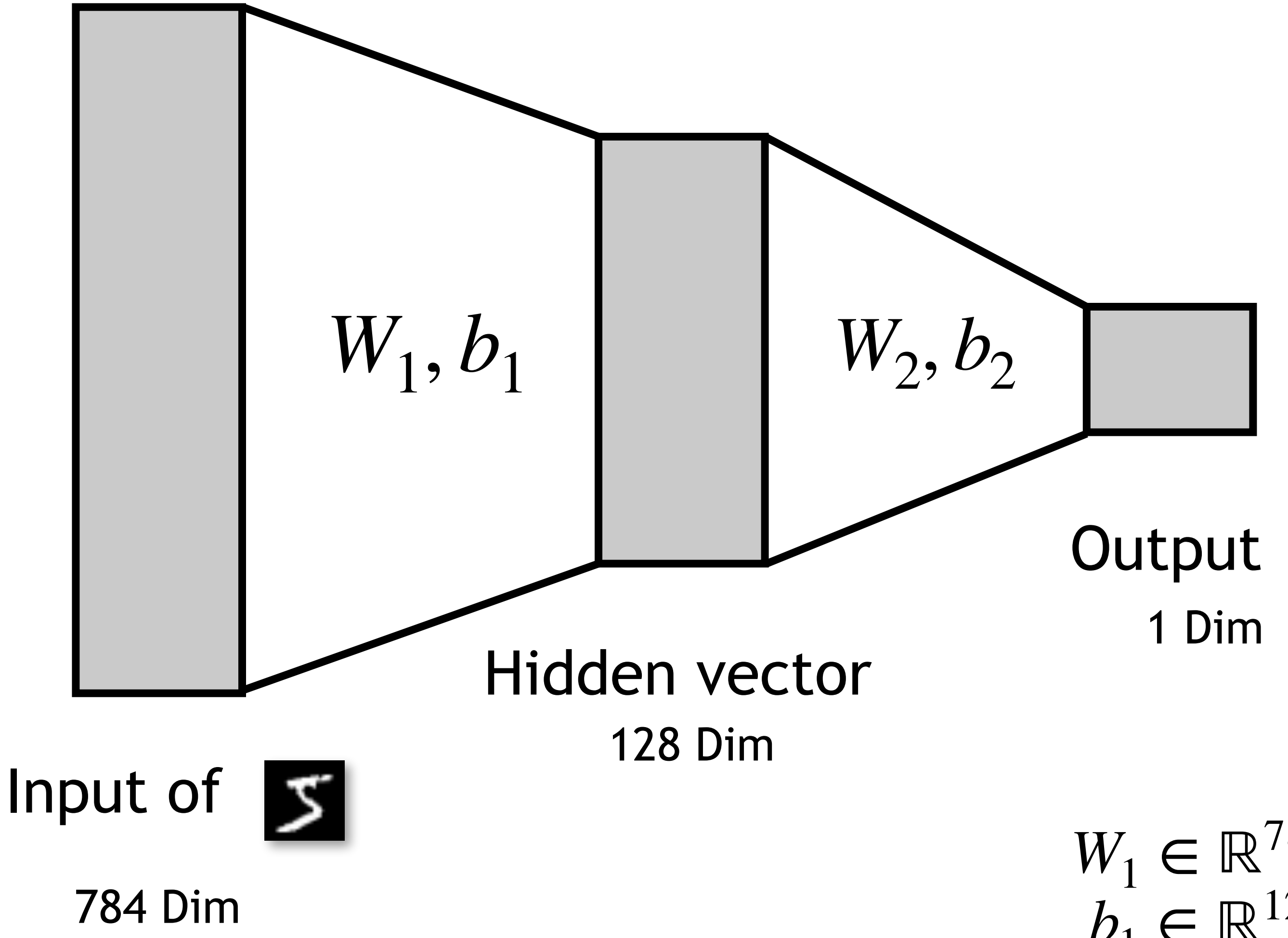
Linear function

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

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



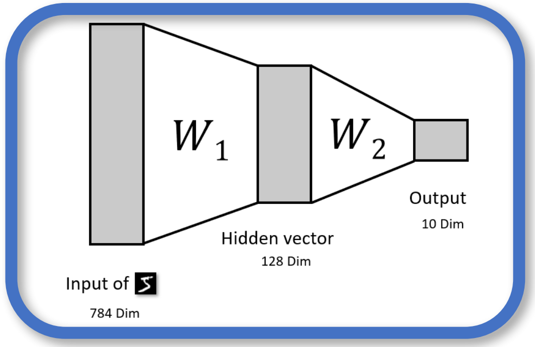
$W_1 \in \mathbb{R}^{784\times128}, W_2 \in \mathbb{R}^{128\times1}$
$b_1 \in \mathbb{R}^{128}, b_2 \in \mathbb{R}$

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

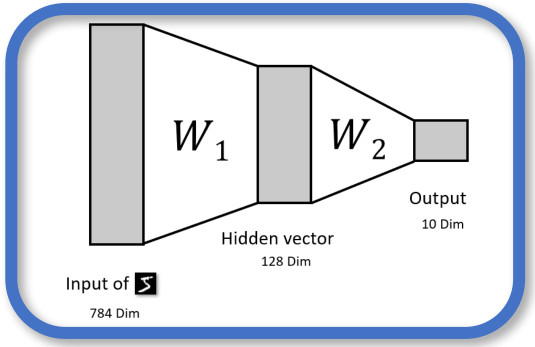
Classification function with MLP

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

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

Gradient $\dfrac{\partial \mathscr{L}}{\partial \theta}$

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

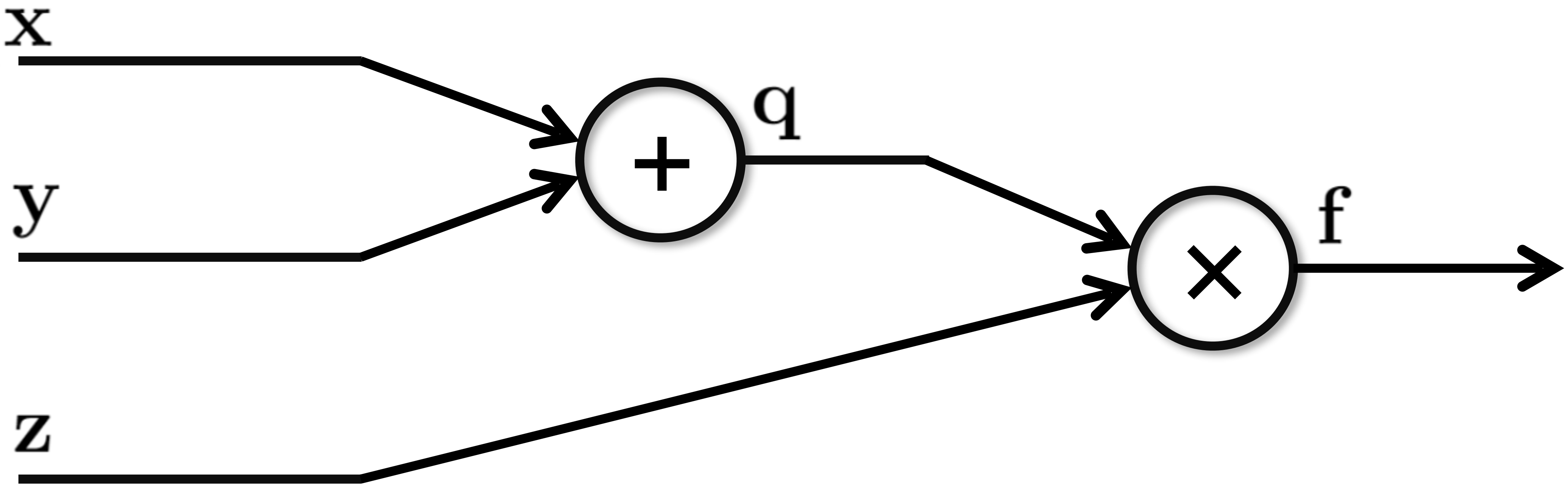
However:

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

Backpropagation with a toy example

Let we consider a toy example:
(Computational graph)

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

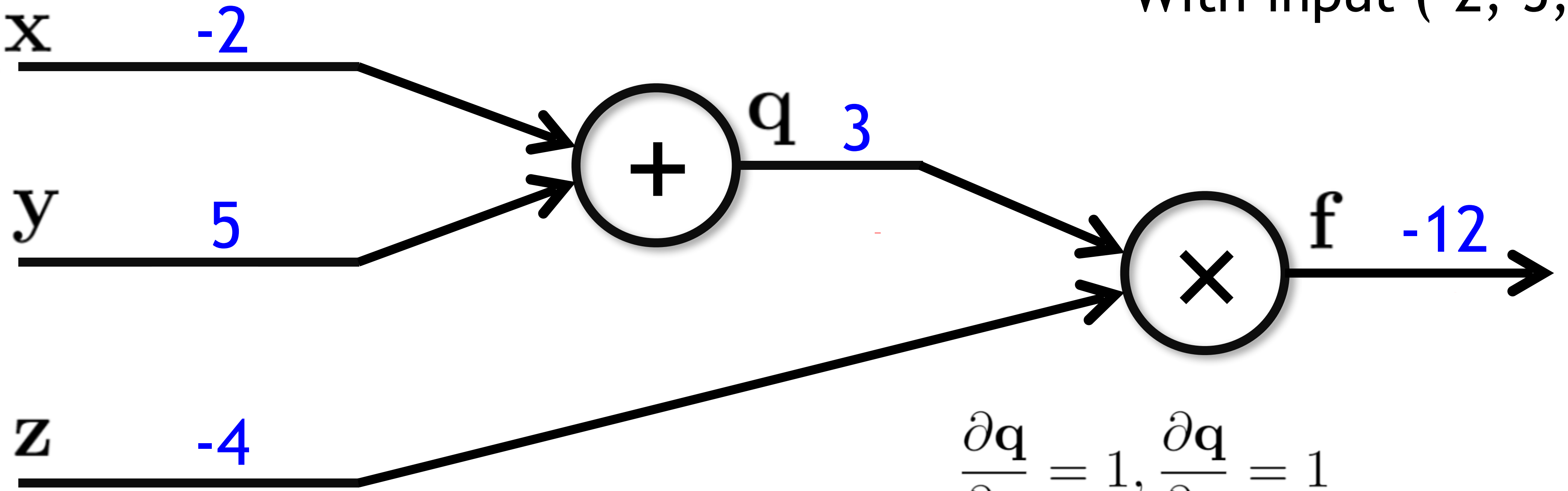


And we want $\dfrac{\partial f}{\partial \mathbf{x}}, \dfrac{\partial f}{\partial \mathbf{y}}, \dfrac{\partial f}{\partial \mathbf{z}}$

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)



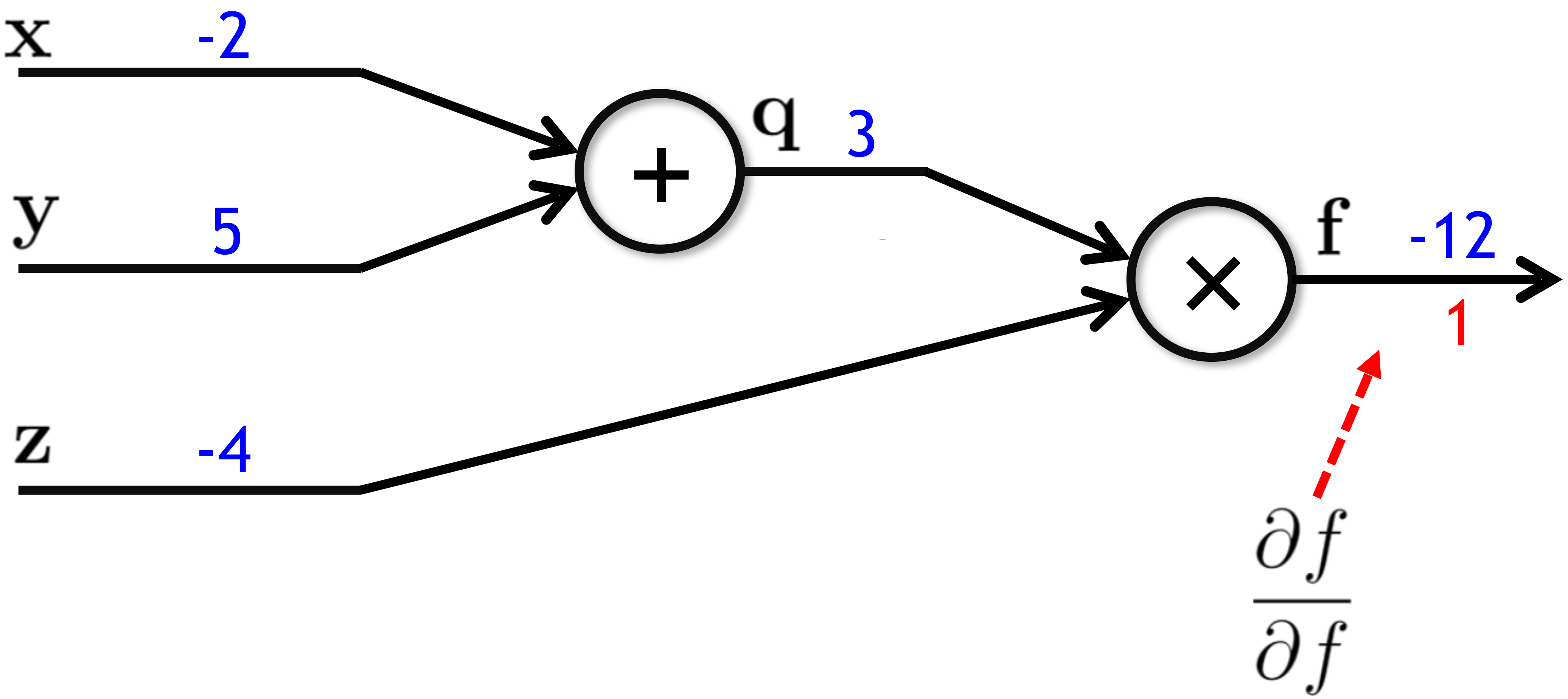
And we have the derivation

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

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

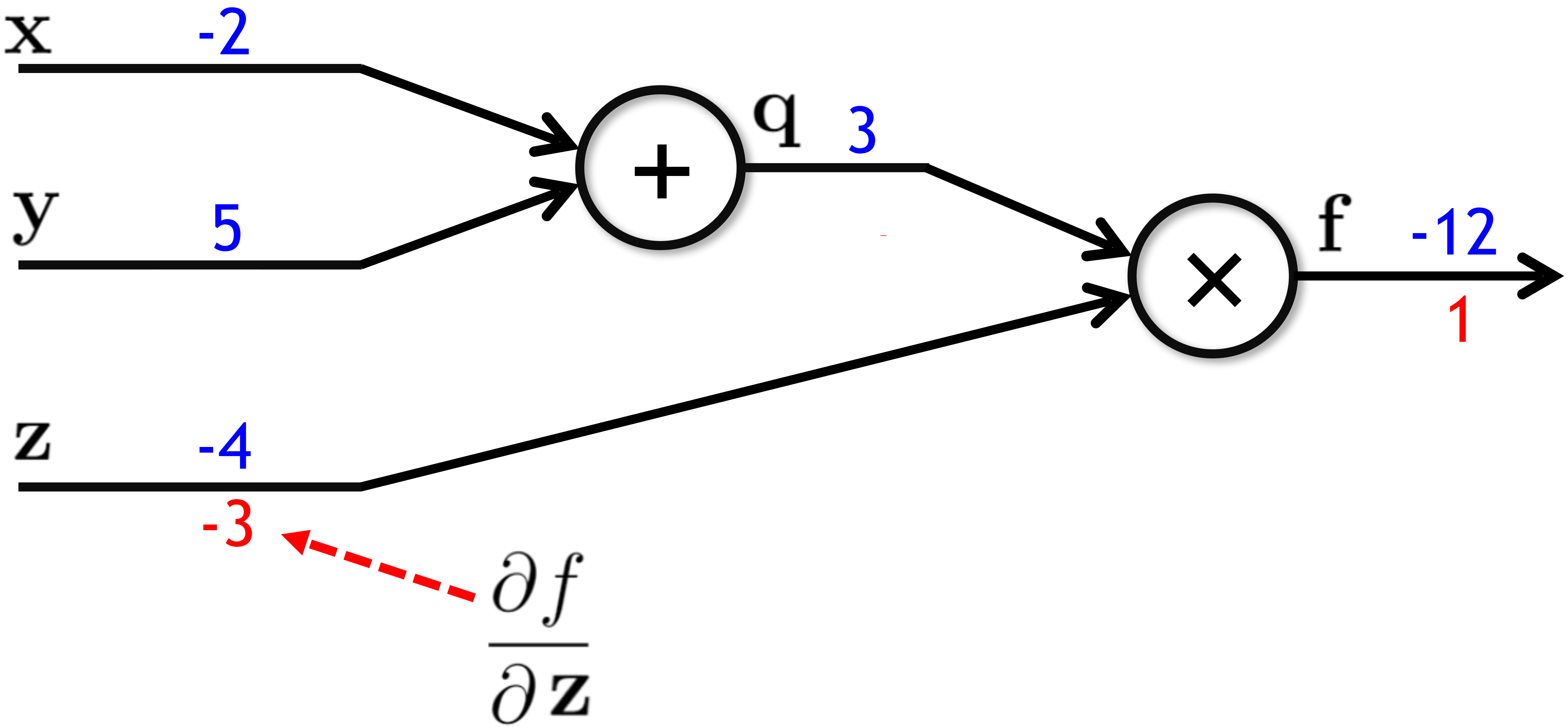
Backpropagation with a toy example

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



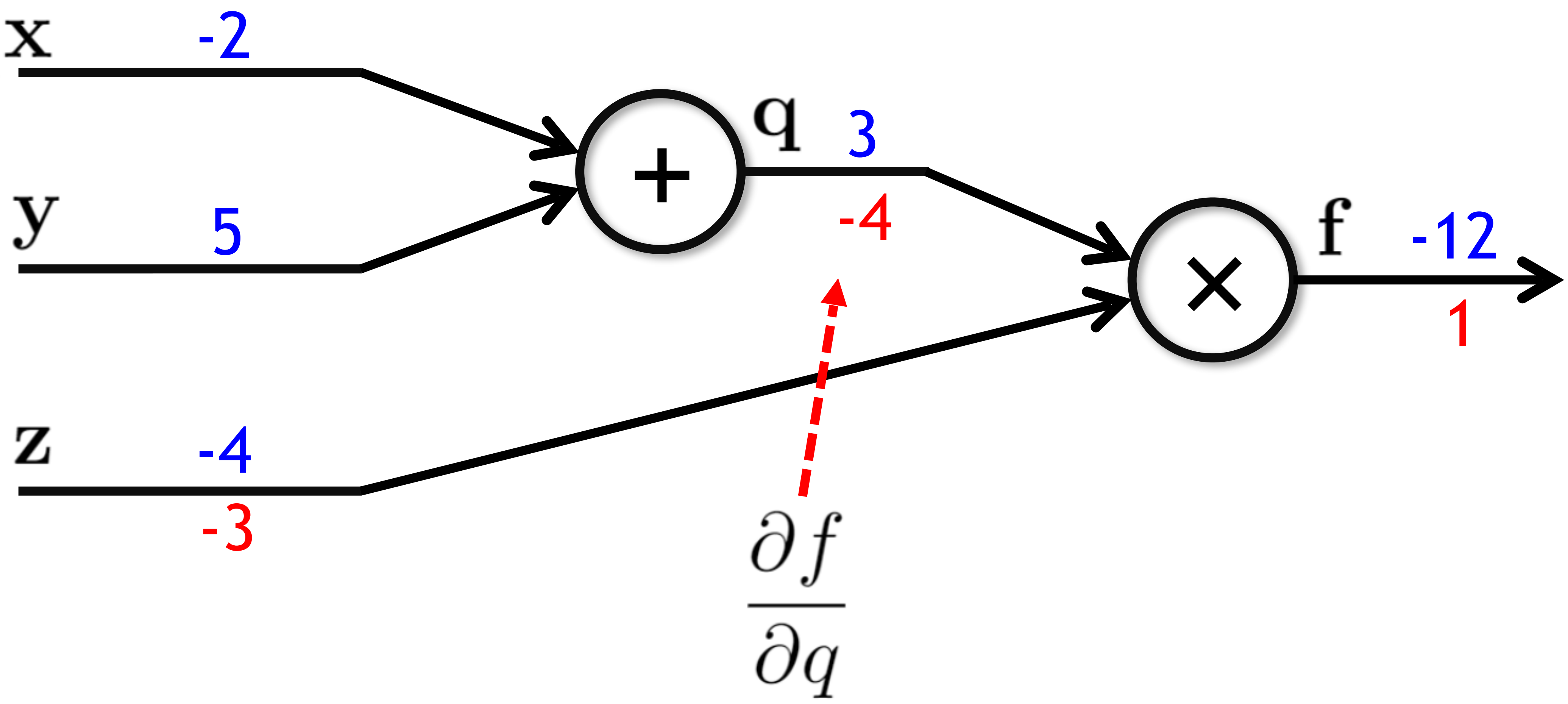
Backpropagation with a toy example

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



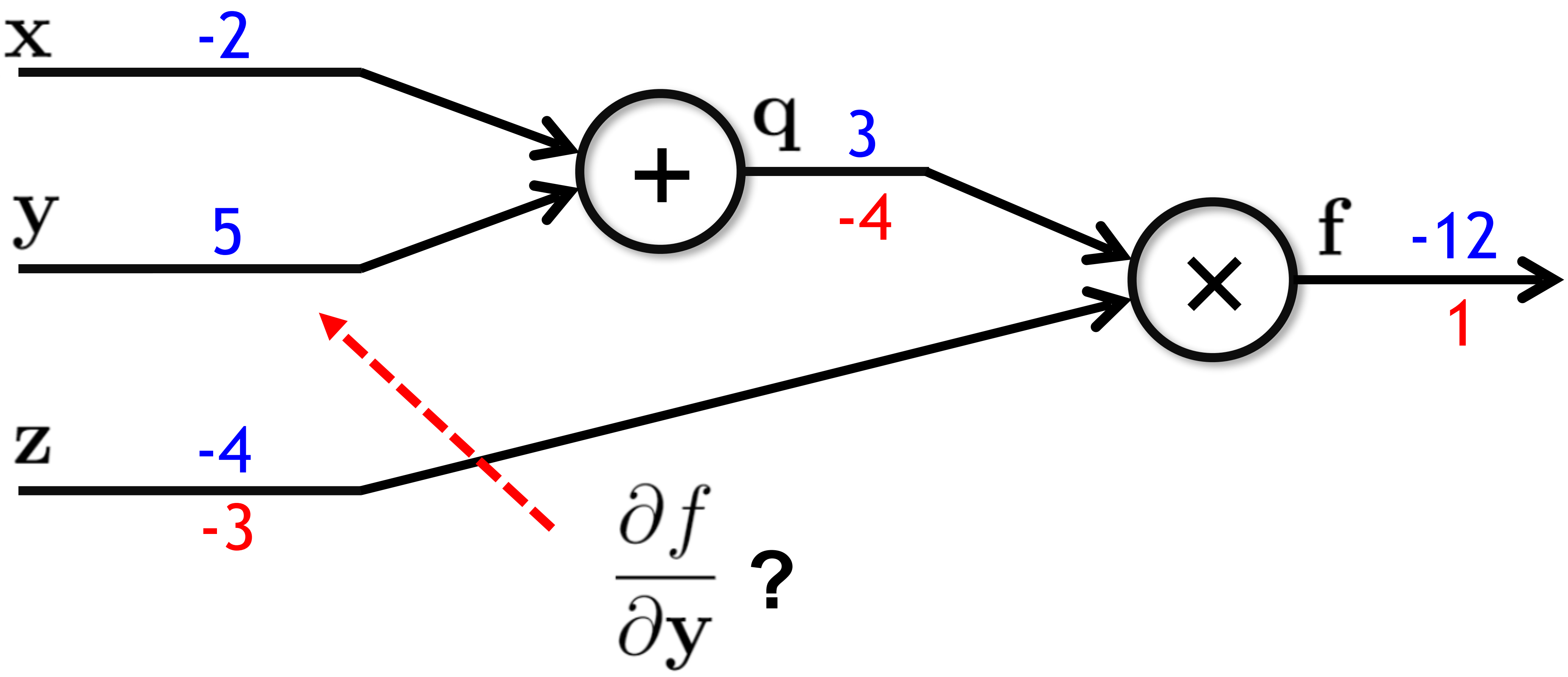
Backpropagation with a toy example

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



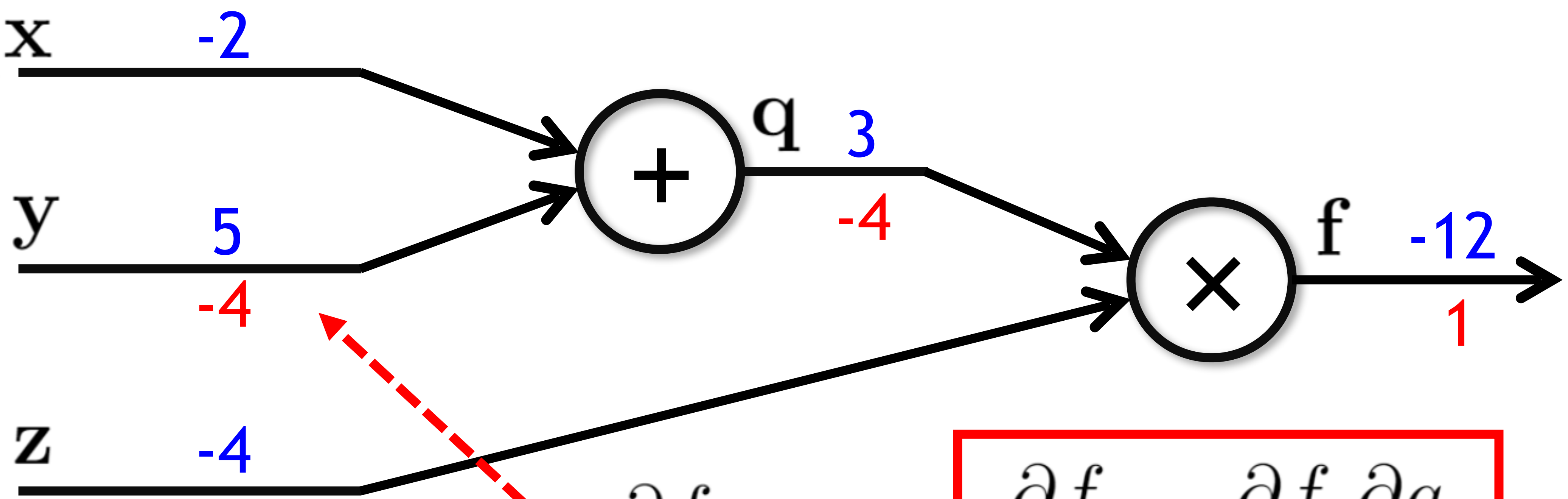
Backpropagation with a toy example

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



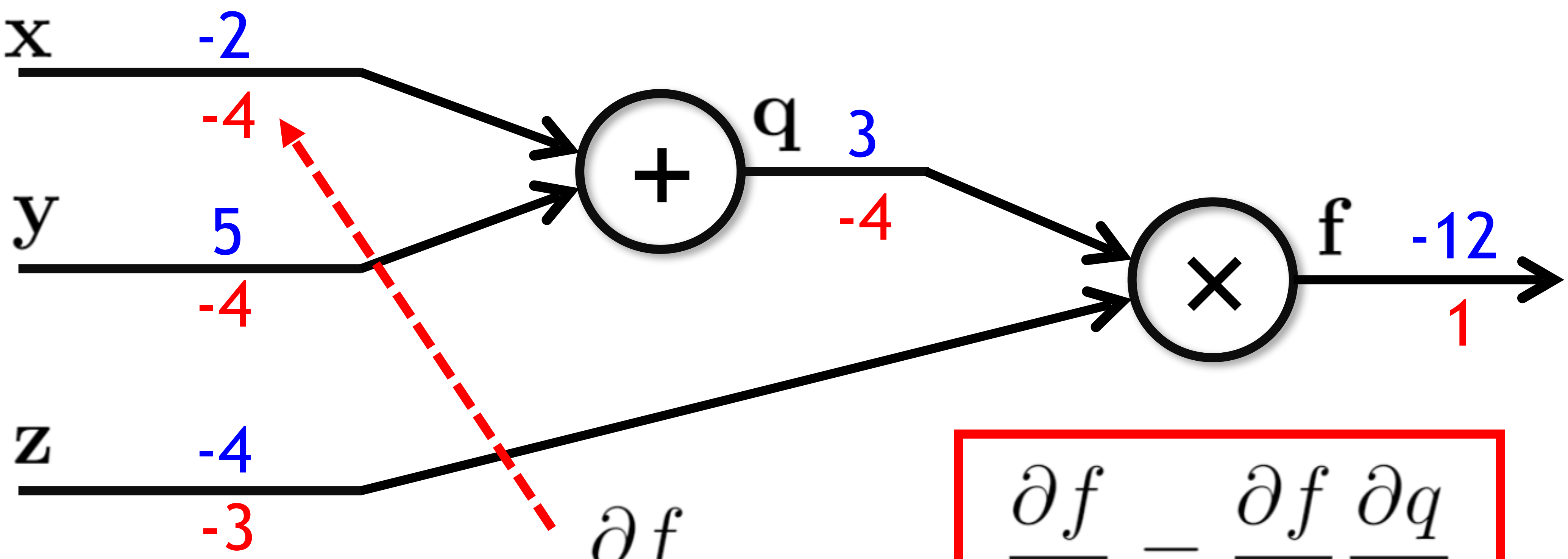
$$\frac{\partial f}{\partial \mathbf{y}} \text{ ?}$$

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

Chain rule

Backpropagation with a toy example

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

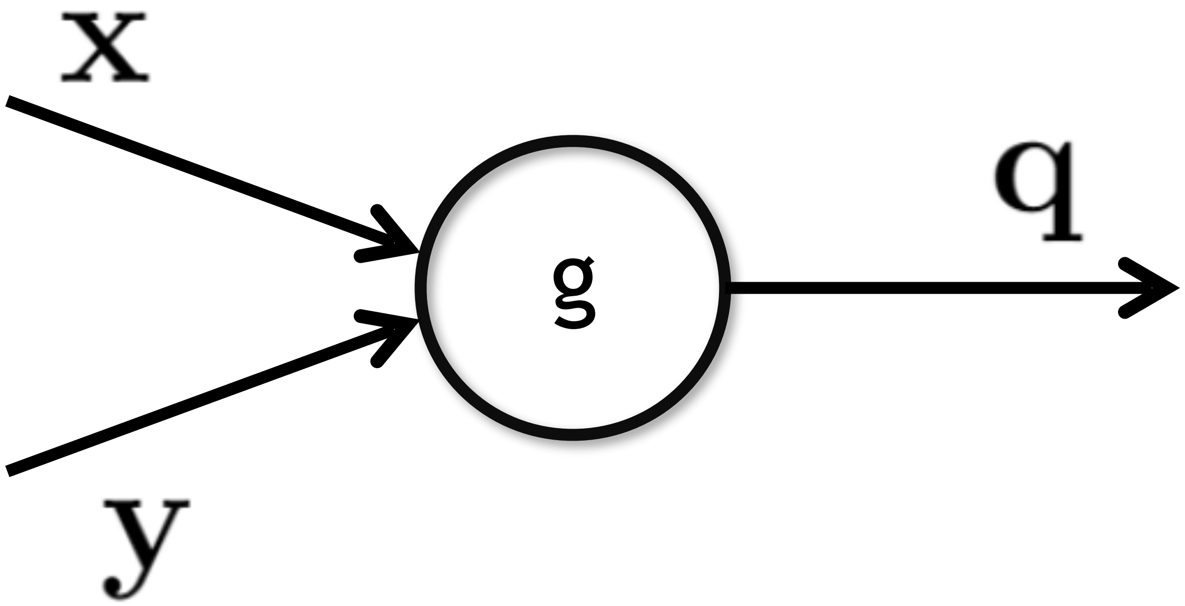


$$\frac{\partial f}{\partial \mathbf{x}}$$

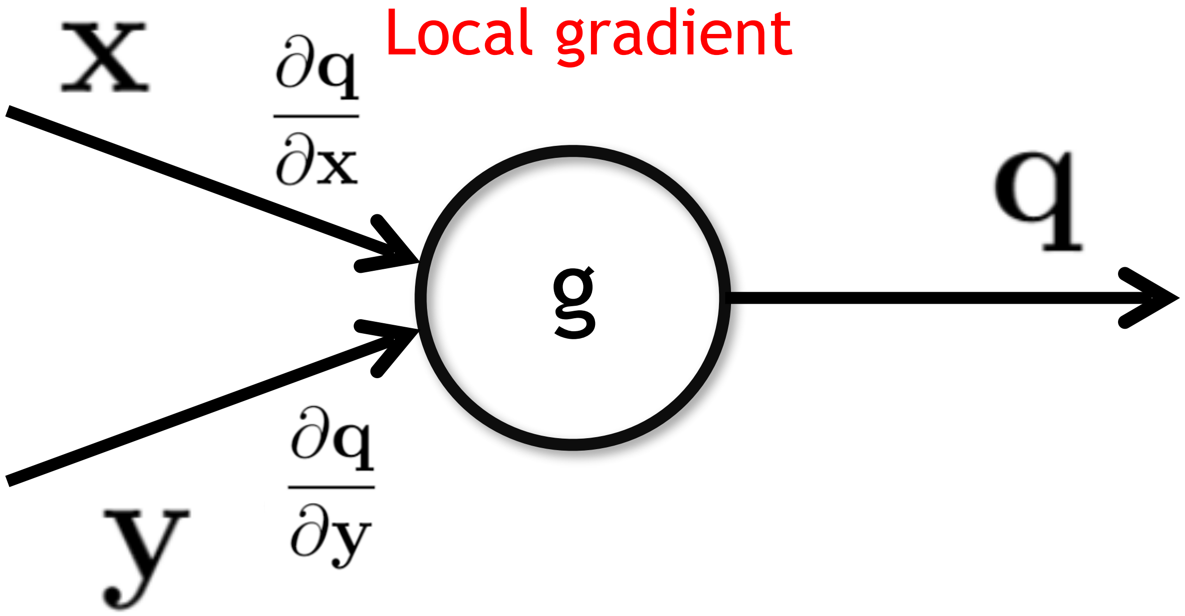
$$\boxed{\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial \mathbf{x}}}$$

Chain rule

Chain rule

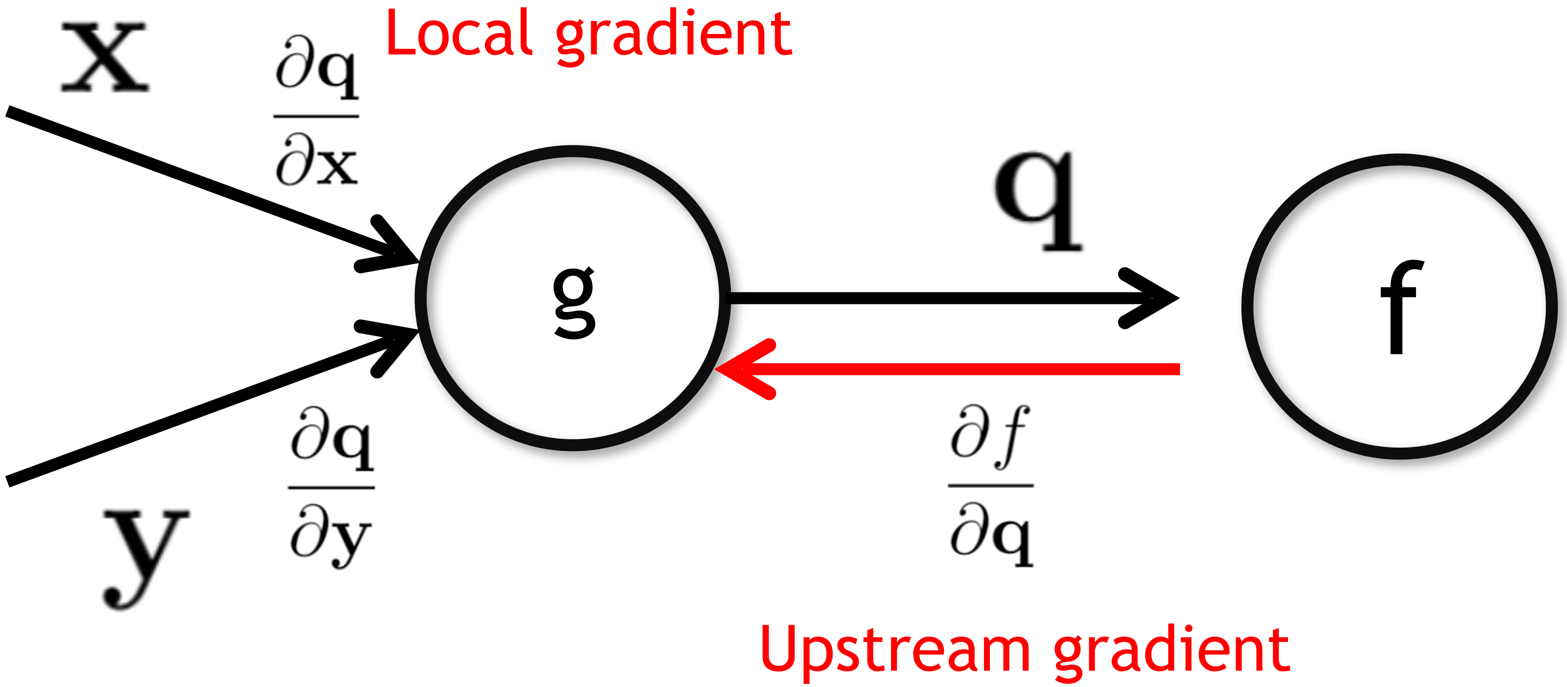


Chain rule



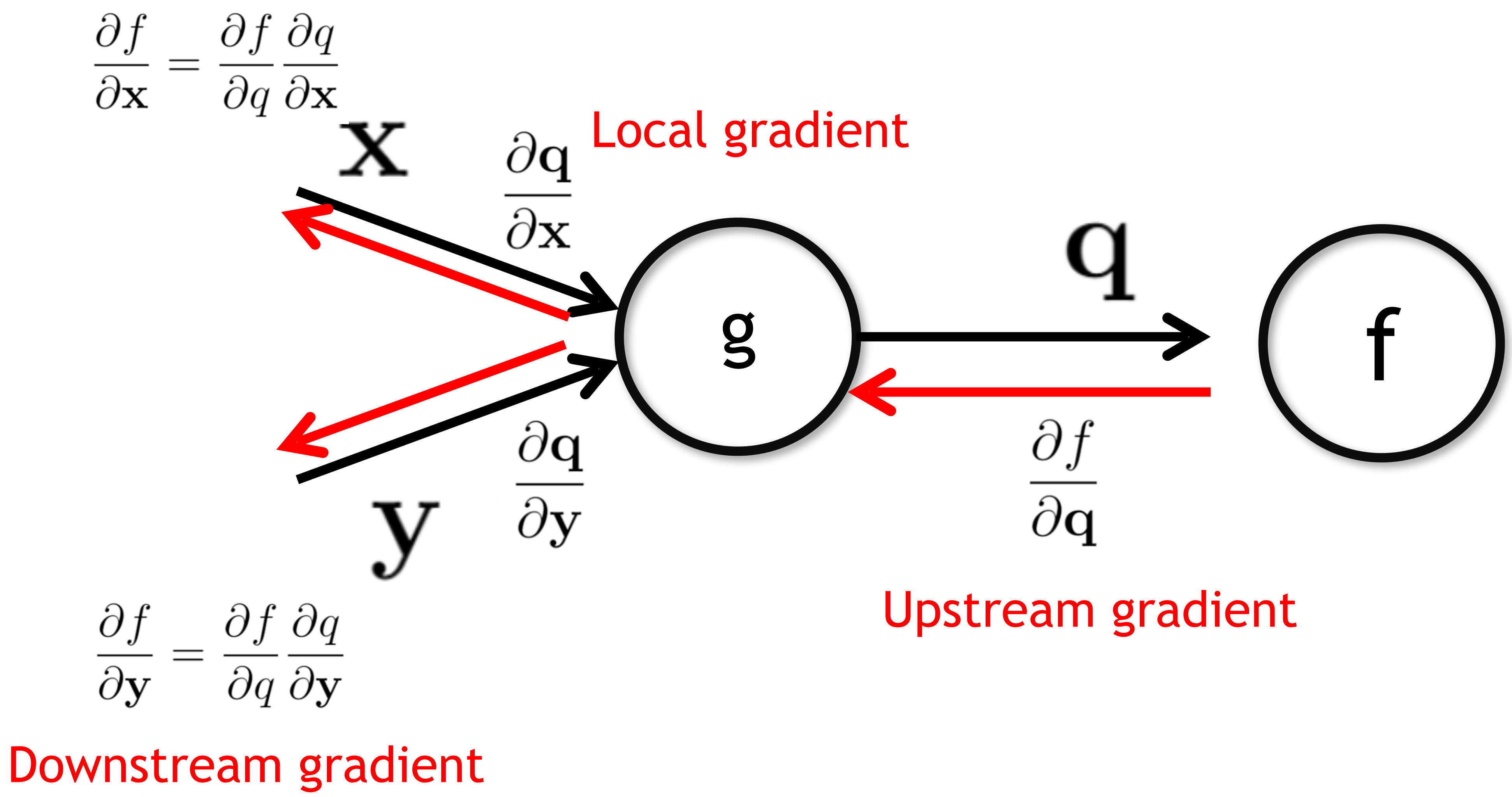
Local gradient
x
$\frac{\partial q}{\partial x}$
g
q
y
$\frac{\partial q}{\partial y}$

Chain rule



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

Chain rule

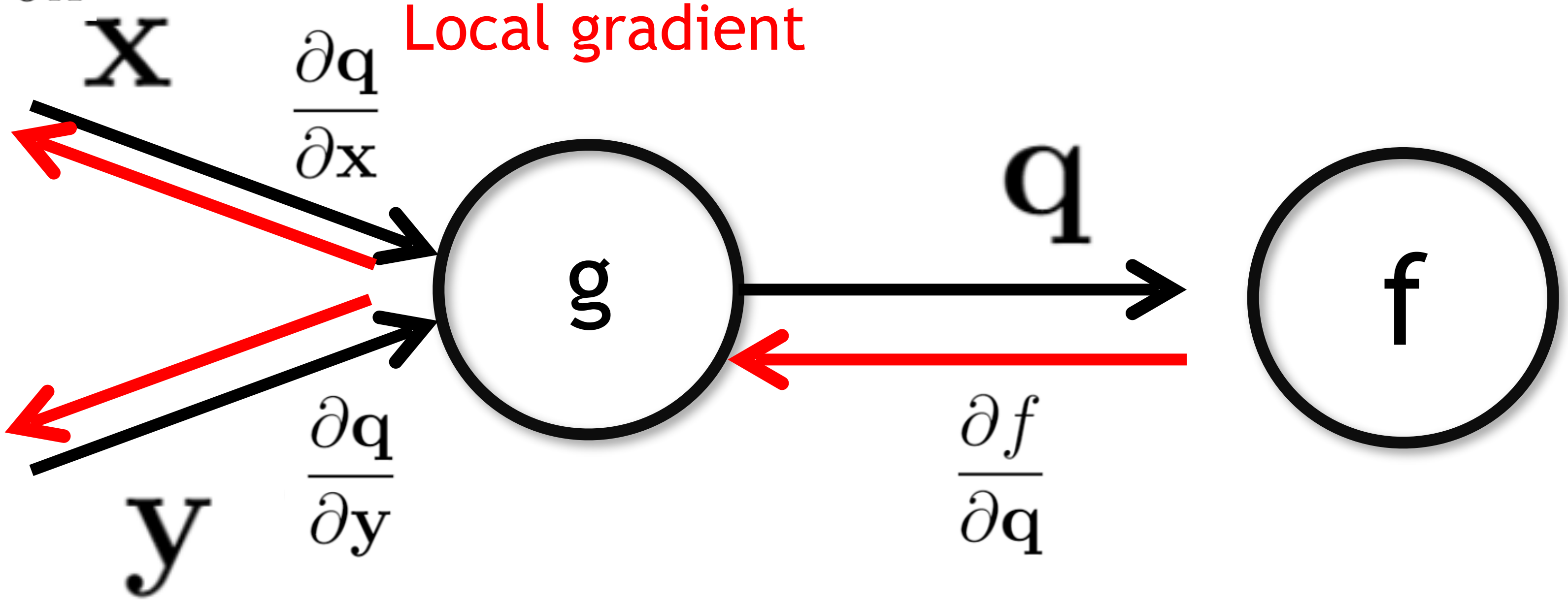


$\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial \mathbf{x}}$
$\mathbf{x}$
$\frac{\partial \mathbf{q}}{\partial \mathbf{x}}$
Local gradient
g
$\mathbf{q}$
f
$\frac{\partial \mathbf{q}}{\partial \mathbf{y}}$
$\frac{\partial f}{\partial \mathbf{q}}$
$\mathbf{y}$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

Chain rule

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

Local gradient



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

Upstream gradient

Downstream gradient

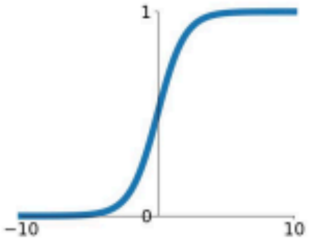
The backpropagation can be efficiently implemented with simple matrix operations

"Backpropagation for a Linear Layer" from Justin 2017

Activation functions

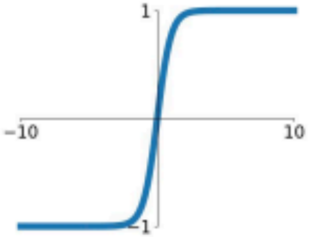
Sigmoid

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



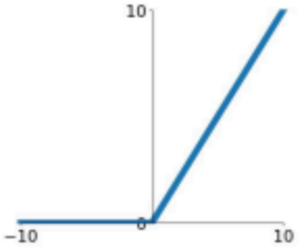
tanh

$$\tanh(x)$$



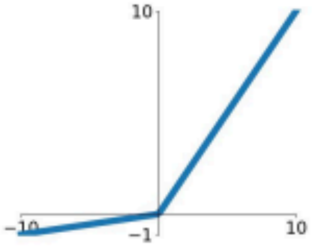
ReLU

$$\max(0, x)$$



Leaky ReLU

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

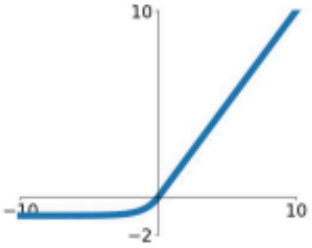


Maxout

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

ELU

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



Slides credit: Stanford CS231N

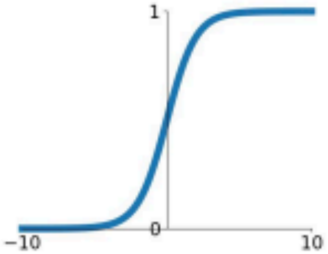
Activation Function

Activation functions

ReLU is a good default choice for most problems

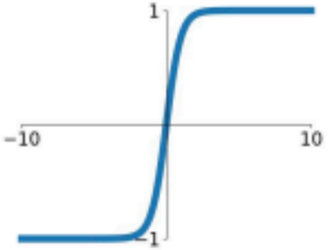
Sigmoid

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



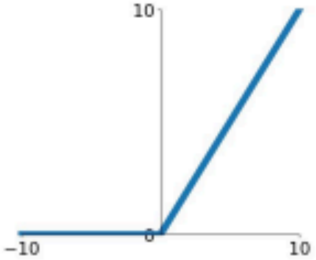
tanh

$\tanh(x)$



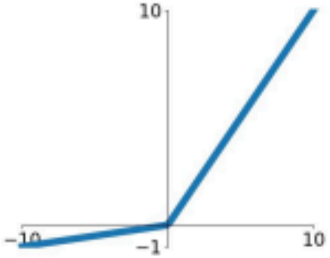
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

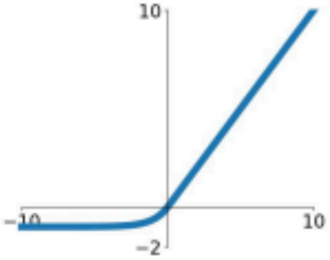


Maxout

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

ELU

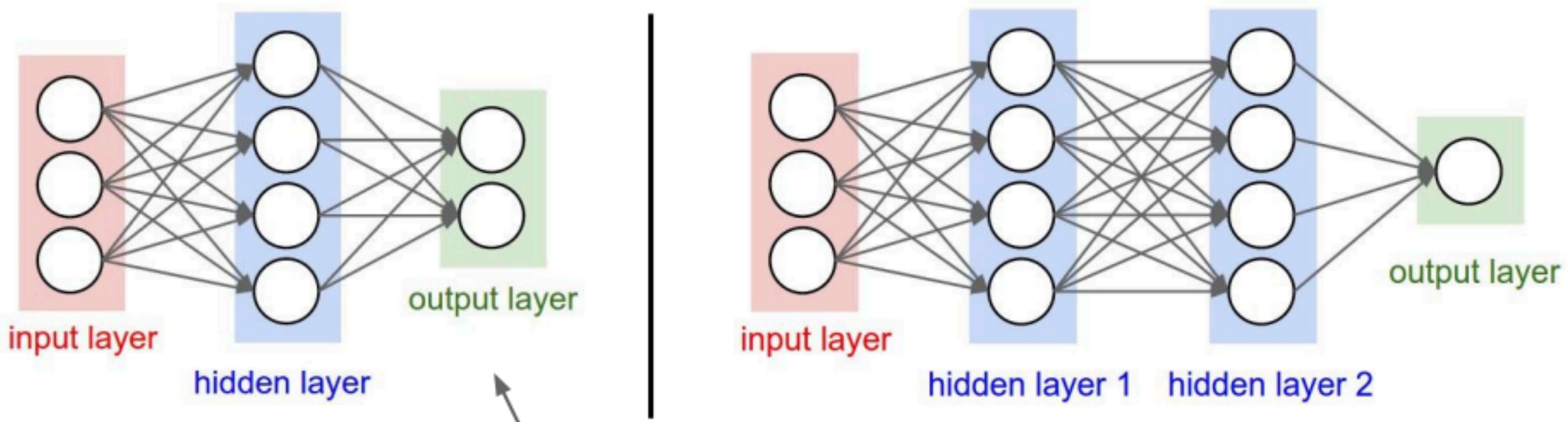
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Slides credit: Stanford CS231N

Review

Neural networks: Architectures



“2-layer Neural Net”, or
“1-hidden-layer Neural Net”

“Fully-connected” layers

“3-layer Neural Net”, or
“2-hidden-layer Neural Net”

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.

Next week: Lecture 5,
Deep Learning II