Lecture 1

CLASSIFICATION OF IMAGES USING A DATA DRIVEN APPROACH: INTRODUCTION

Claudia Acquistapace
Istitute for Geophysics and Meteorology
University of Cologne
email: cacquist@uni-koeln.de

UNIVERSITY
OF COLOGNE

About the lectures (and me)

Slides of the lectures and additional lectures notes with additional material can be found at:

https://tinyurl.com/teachingUnibo2024

You can always contact me at **cacquist@uni-koeln.de** for questions on the lectures or on whatever (i.e. master thesis, Erasmus, living in Germany etc), I am happy to help you

Info about me

www.claudiaacquistapace.it

and
my Junior research group EXPATS

https://expats-ideas4s.com

We are also on social media, if you want to follow us:

@EXPATS_ideas4s

@EXPATS-ideas4s

UNIVERSITY OF COLOGNE

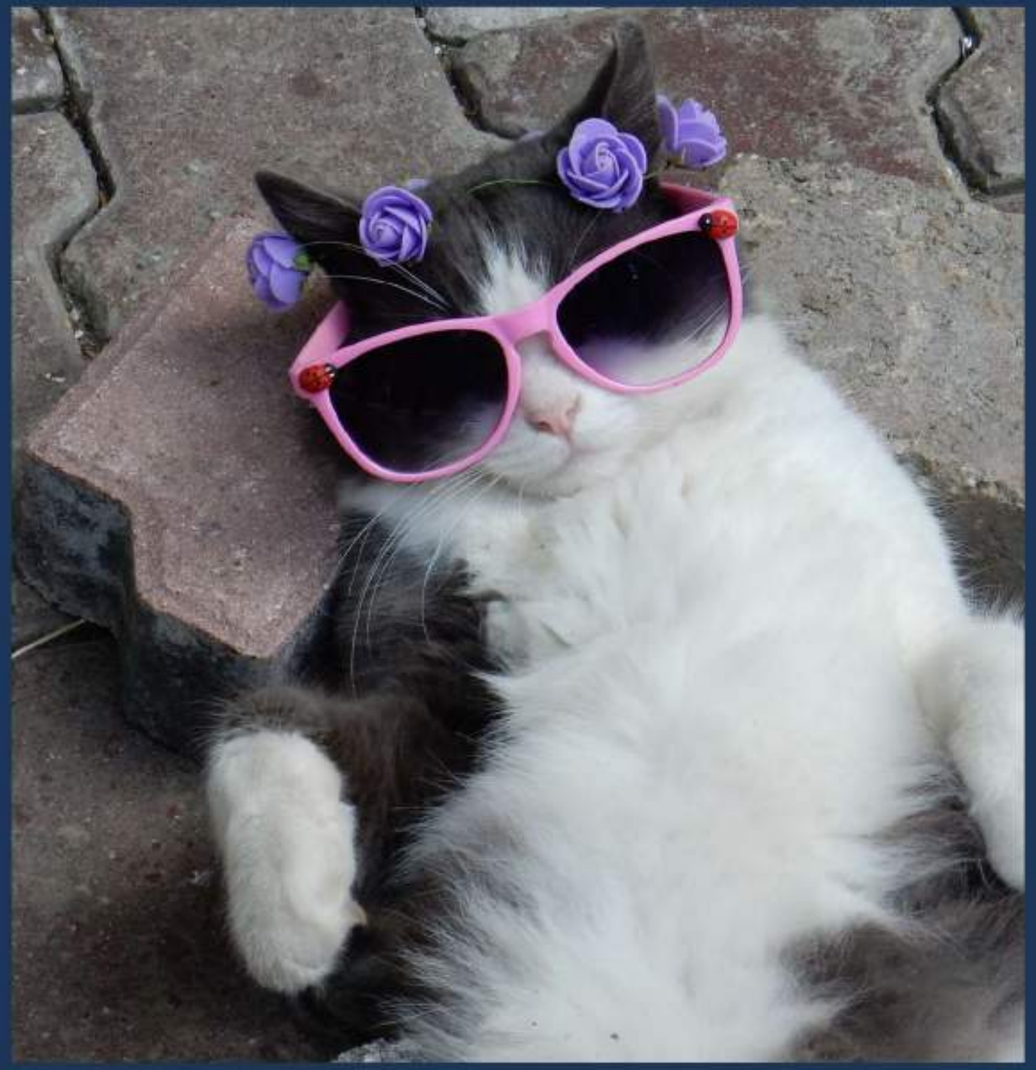
Topics for today

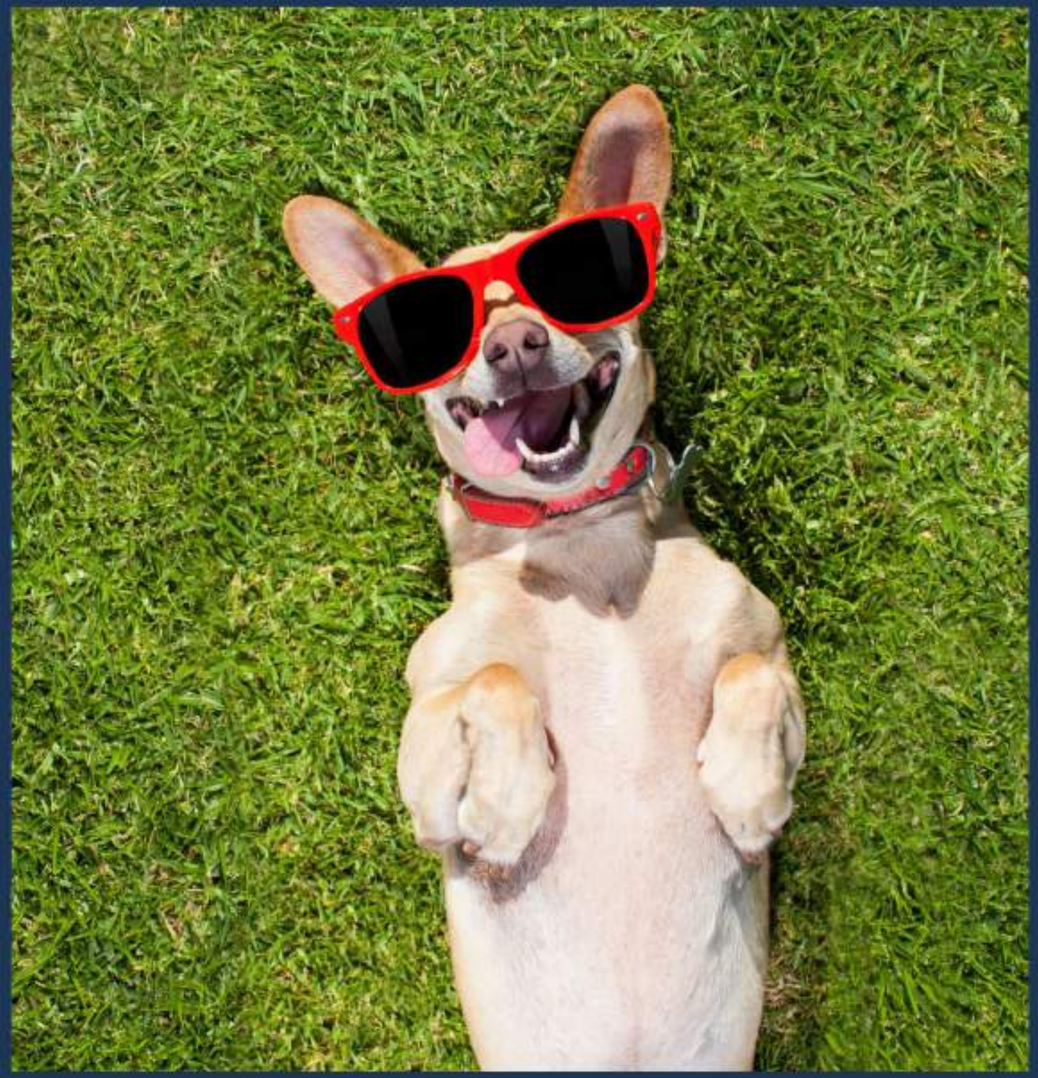
1. The problem of **image classification** or... assigning a label to an image with a computer
2. **The simplest data driven approach**: a linear classifier and a loss function
3. **Learning process via optimization** (and the various processes behind it)

1

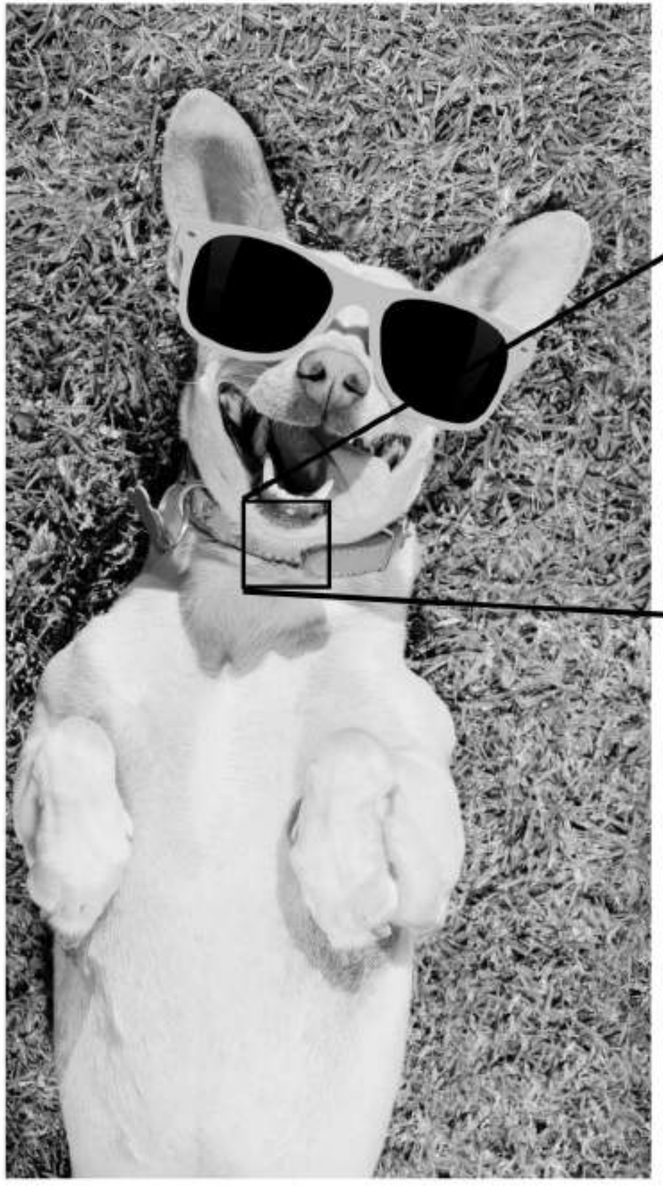
The problem of image classification or... assigning a label to an image with a computer

Assigning a label to an image with a computer





What a human sees in BN

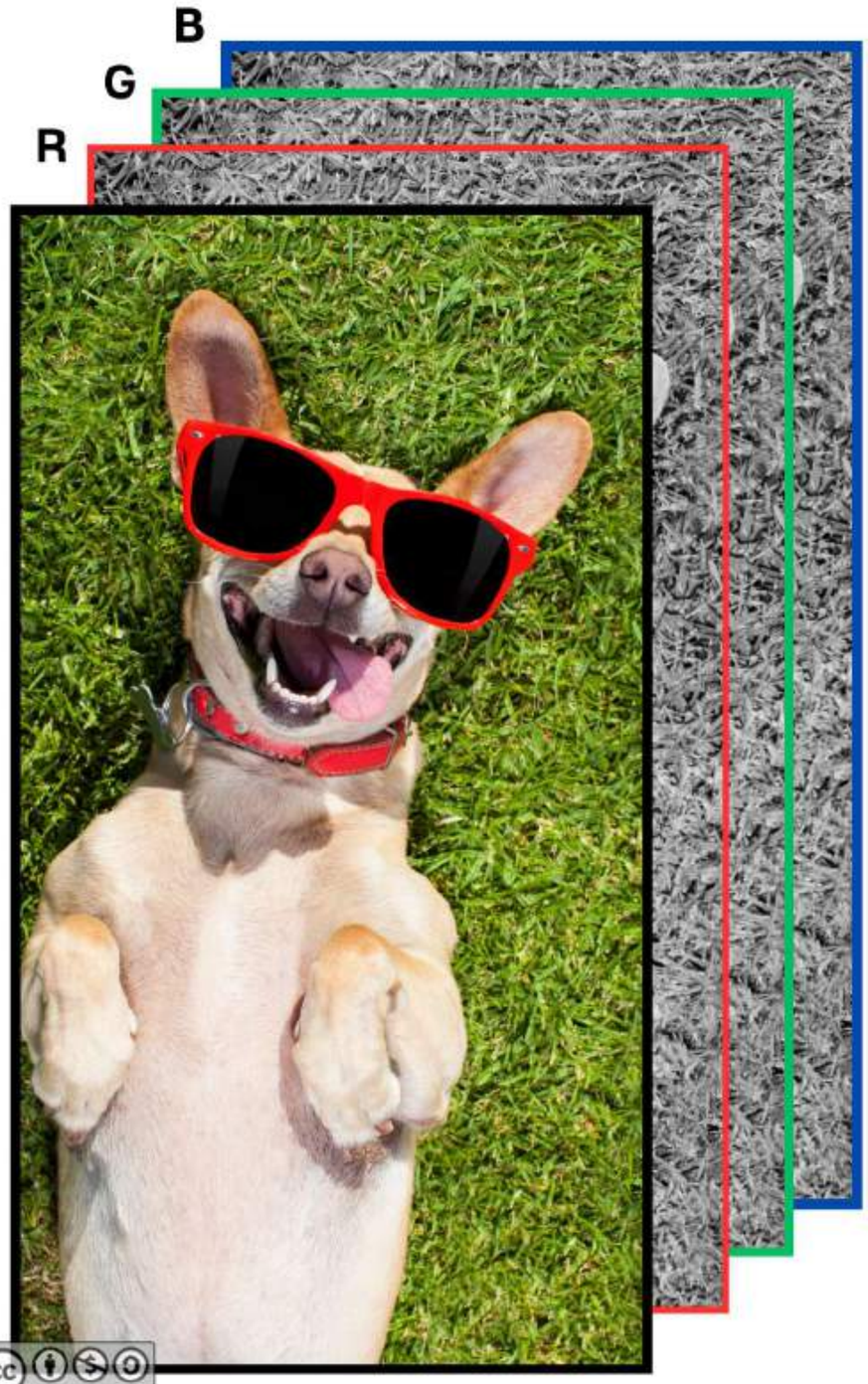


N_pixels_height

N_pixels_width

What a computer sees

5	134	26	47	102	111
1	155	57	76	34	210
95	83	82	0	51	233
159	210	249	156	196	129

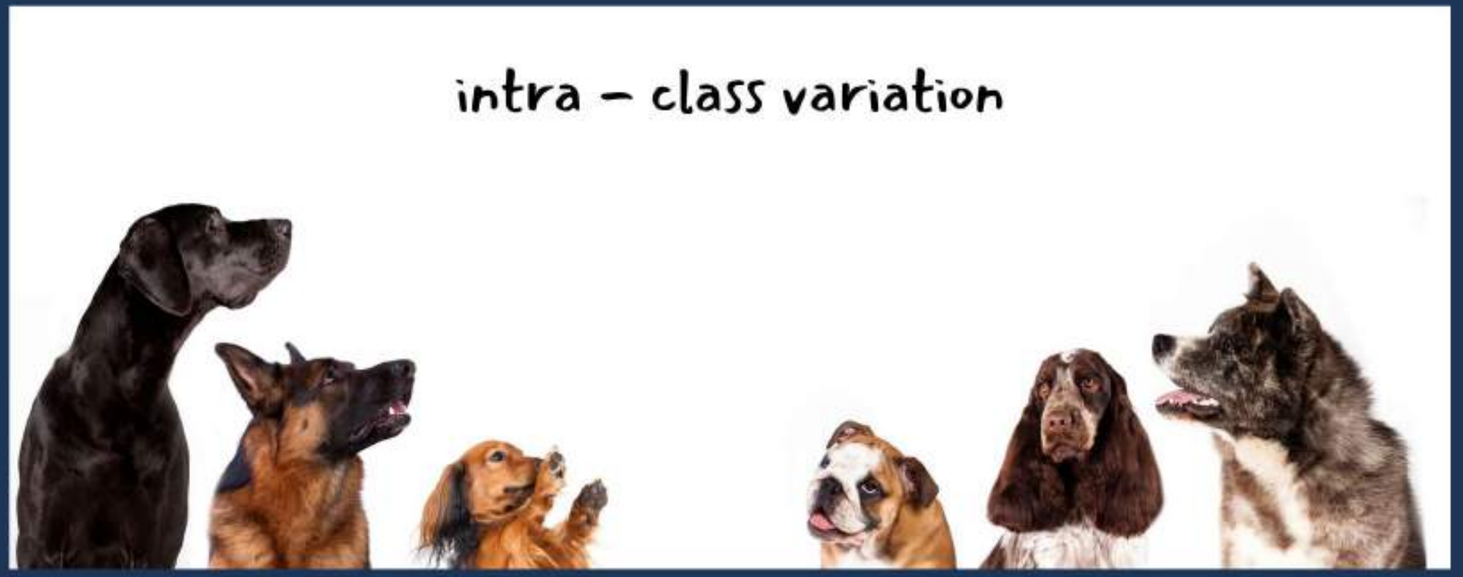


Color images are tri-dimensional arrays of integers ranging from 0 to 255,

Their dimension is: **N_pixels_width x N_pixels_height x 3**,

3 represents the three colors provided in the RGB, i.e. Red, Green and Blue.

change of viewpoint



How to write down an algorithm that classifies images?

There's not, if you think about it, a straightforward strategy to classify images.

Data driven approach

...or learn from the data

1) **Collect** as many samples as you can of images **of each** of the selected classes

2) **Learn** visual characteristics that can be identified by looking at the pictures in each class.

Example of a classifier: the k-nearest neighbor

WHAT IS THE K-NEAREST CLASSIFIER

It is an algorithm based on proximity that classify images, also called the lazy algorithm because it does not do other operations than barely memorize

WHAT IS IT USEFUL FOR?

text mining, agriculture, finance, medical and facial recognition.
It is never used for image classification because it is very slow at test time

HOW DOES IT WORK?

training: it memorizes the labels of the images
test: derives, based on the defined distance, the closest k nearest images and identifies the most recurring label.

WHAT DOES IT NEED?

It needs to define:

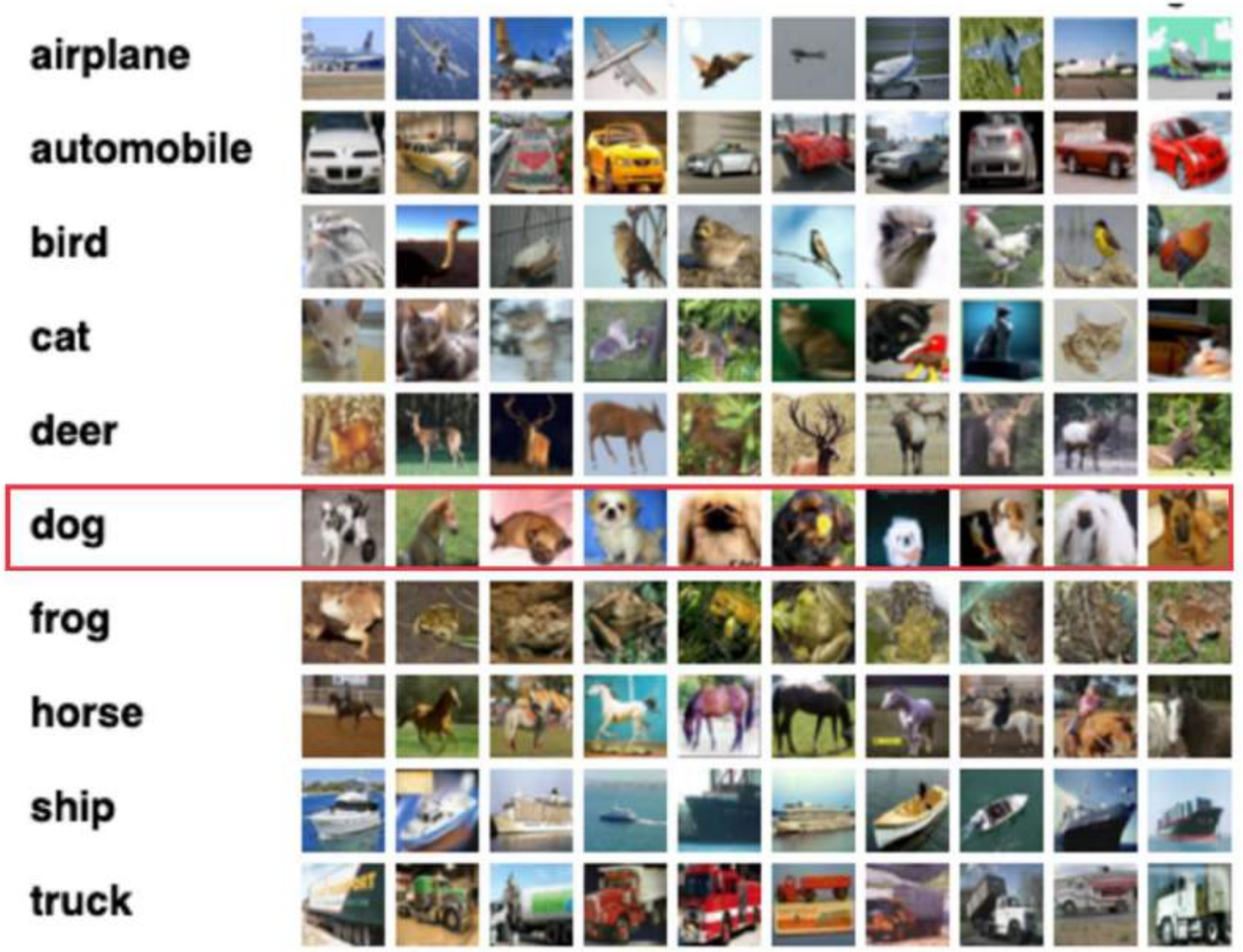
- the parameter k
- the distance to use to measure (L1 or L2)

$$d_{L_1}(I_1, I_2) = \sum_i |I_1^i - I_2^i| \qquad d_{L_2}(I_1, I_2) = \sqrt{(\sum_i (I_1^i - I_2^i)^2}$$

INPUT: N images with N labels, each in one of the D classes of the dataset. This dataset is called the training dataset.

The CIFAR10 dataset contains 60000 color images in 10 classes, with 6000 images per class. Here we can see 10 random images for each class.

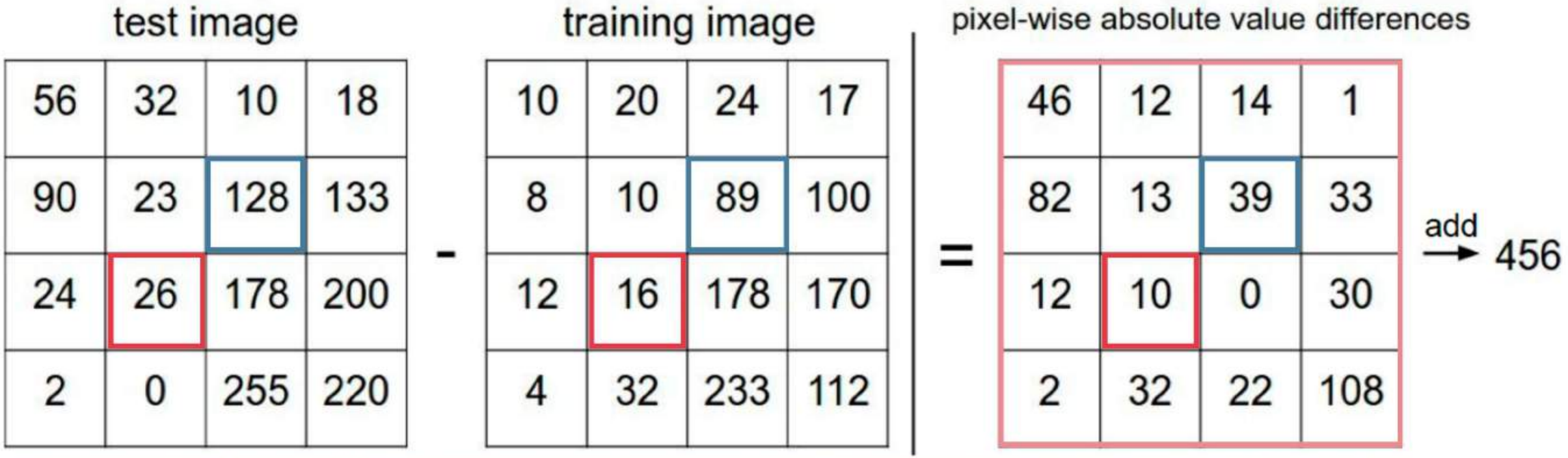
In the class dog, we can find very diverse images of dogs



Example of 10 random images for the 10 classes from the CIFAR10 dataset, available at https://www.cs.toronto.edu/%7Ekriz/cifar.html

TRAINING: To get trained, K-nn needs to memorize the labels of all images.

TEST: calculation of the distance between images (in the example with L1 distance)



Graphical representation of the calculation of the L1 pixelwise distance between an test and an training image. Image reinterpreted based on content in https://cs231n.github.io/classification/

1) pixelwise absolute value difference

2) sum of all terms (46 + 12 + 14 + 1 + 82 + 13 + ...22+108) = 456

TEST: finding the most recurrent label

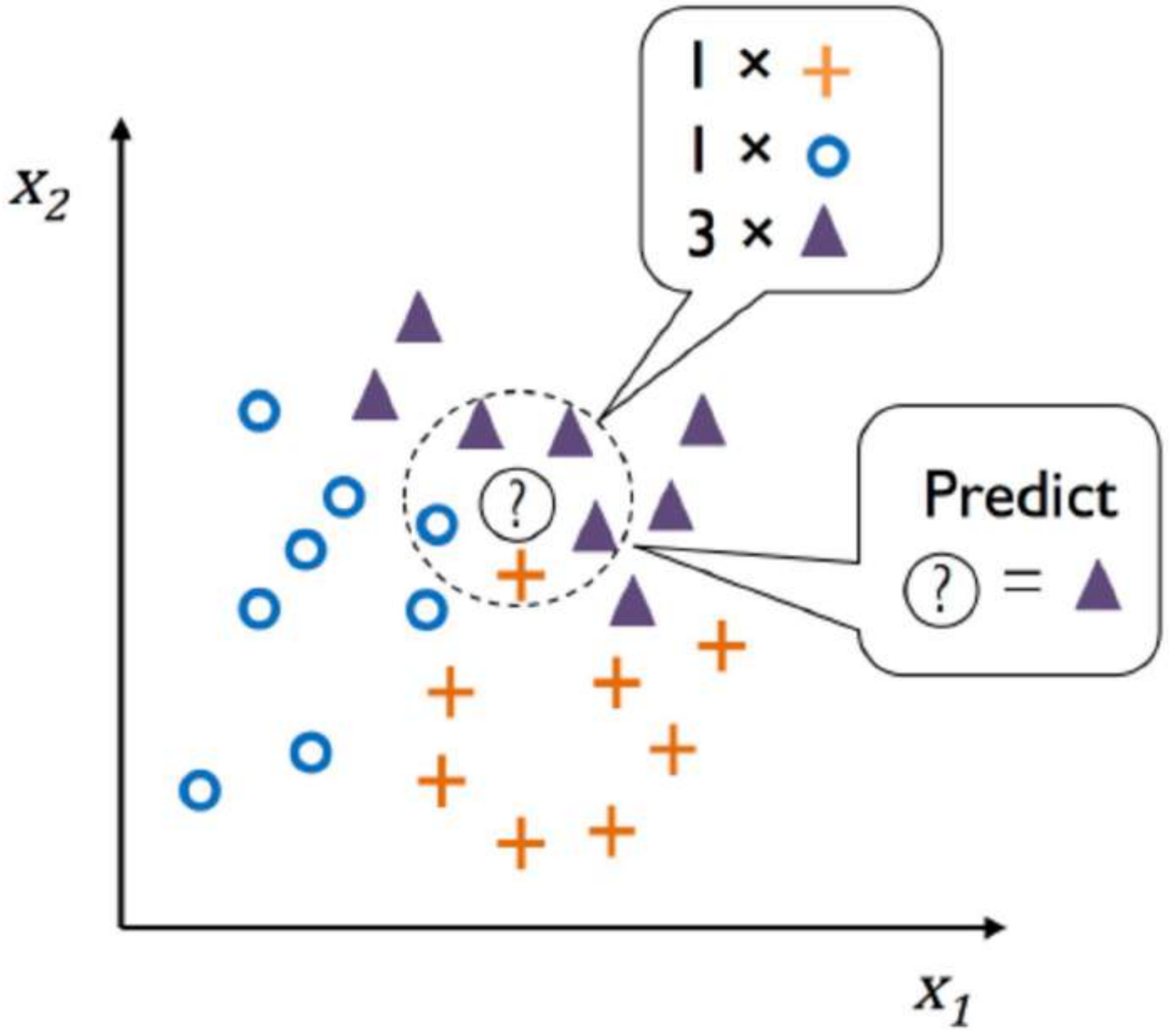
The parameter k tells how many of the nearest neihbor to consider: in the example here k =5.

The label that is picked for classifying the new image is the one that is most recurring in the selected ensemble.

In the case of the figure, the dark triangle.

k is a hyperparameter

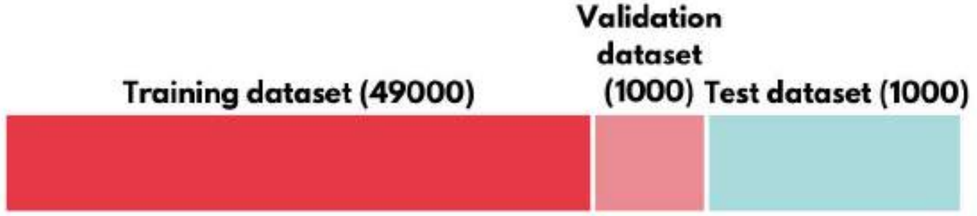
HYPERPARAMETERS: ALL THE PARAMETERS THAT ARE NEEDED TO SET UP THE CLASSIFIER, OF WHICH WE DON'T KNOW A PROPER VALUE A PRIORI



Example of K-nn algorithm with k=5 and 3 classes. In the figure, we can see that the predicted label of the grey triangle is the one that among the k-nearest neighbors, occurs more frequently, 3 times compared to 1 time for each of the other classes (Image from Sebastian Raschka's course page stat.wisc.edu/%7Esraschka/teaching/stat479-fs2018/).

On hyperparameters and overfitting

How do we determine the hyperparameters values?



By reserving part of the training dataset, the so-called **validation dataset**, for tweaking hyperparameters.

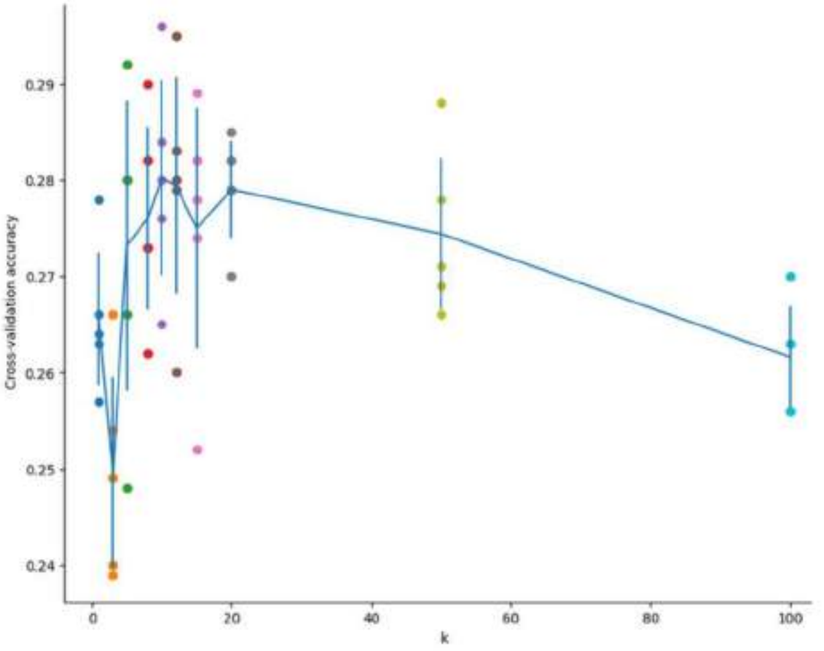
What happens if we don't do it?

Example overfitting, right fit and underfitting from matlab documentation, https://it.mathworks.com/discovery/overfitting.html

We risk to tune the parameters to work perfectly on training data but then not to be able to work on any other data input. (Overfitting)

Methods to avoid overfitting, or how to choose the best k value

Cross-validation: split the data into folds, and then try each fold as validation dataset, and then average the results.



- For each fold run the classification with all possible values of k and derive the accuracy.
- Plot the accuracy for each of the k values tested,
- Identify which is the k-value that works best for the data we are working with

Why K-nn is never used with images?

-Slow at test time

-Distance metrix on pixels are not informative (all images have same L2 distance to the leftmost one)

Example of distance metrix on images from lecture number 2 of Fei-Fei, Li and Johnson and Yeung, April 2017, Stanford course on computer vision

-Curse of dimensionality: with 3 dimensions (RGB images) it becomes soon unmanageable

KNN summary:

In image classification we start with a training set of images and labels, and must predict labels on the test set.

- K-Nearest Neighbors classifier predicts labels based on nearest training examples
- Distance metric and K are hyperparameters
- Determine hyperparameters using the validation set; run on the test set once at the very end!

2

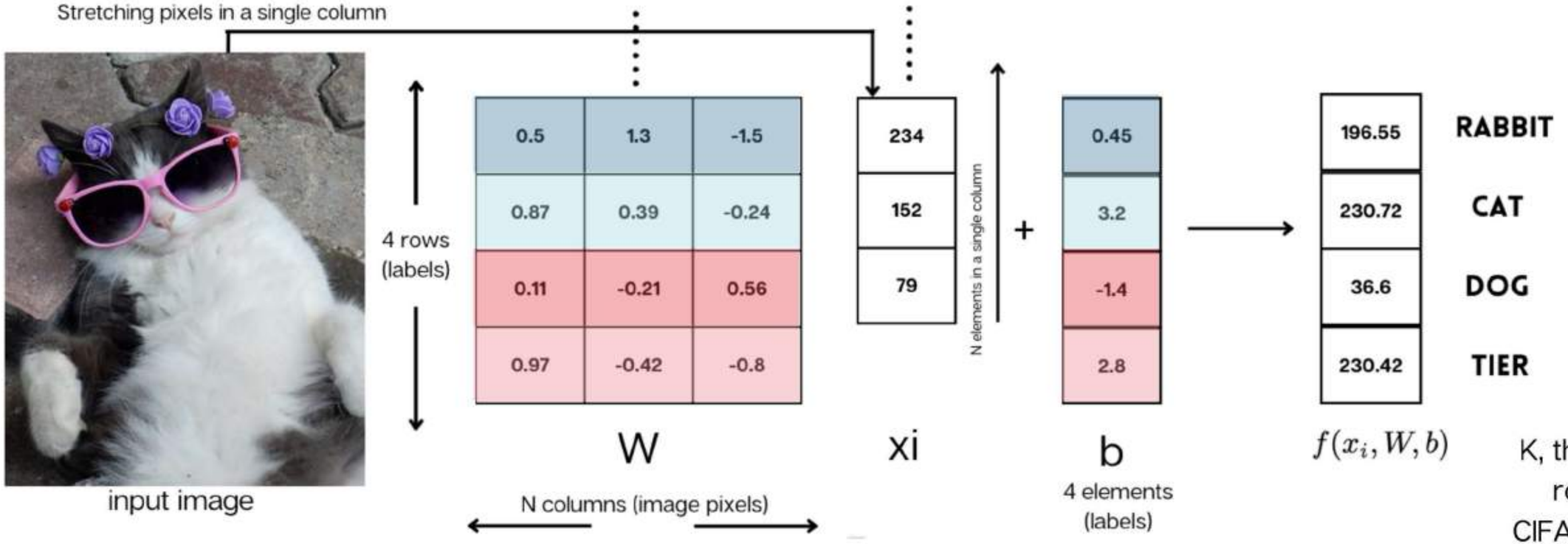
The simplest data driven approach: a linear classifier and a loss function

Parametric approach: linear classifier

f defines a function that maps the 3072 pixels of one image into one of the K labels.

$$f(x_i, W, b) = Wx_i + b$$

For every i an xi belonging to {x1,.... xN} has a label yi belonging to {y1,....yN}).



dimension of the images is D = 32 x 32 x 3 = 3072 pixels, for the example we assume D = 3

K, the number of rows, as for CIFAR10, is 10 (ten classes: dog, tier, etc..)

$$f(x_i, W, b) = Wx_i + b$$

32 x 32 x 3

3072 in total

3072 x 1

10 x 1

$f(x_i, W, b)$

10 numbers giving class scores

10 x 1

W

parameters or weights

10 x 3072

How to interpret the linear classifier?

This is an example of the weights (column) of the linear classifier trained with the CIFAR10 dataset we saw before

Example of weights optimized for the identification of the CIFAR10 classes. Figure from the course https://cs231n.github.io/classification/

The loss function

The loss function establishes how good the classifier is.
If we have N examples of a images xi and their corresponding labels yi, the loss function over this dataset of N elements is given by the sum of the losses Li over each of the single examples plus a regularization term R(W):

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

f(xi, W) is the result of the scoring function (the predicted label)
yi is the so-called "ground truth" (the label associated to the image)

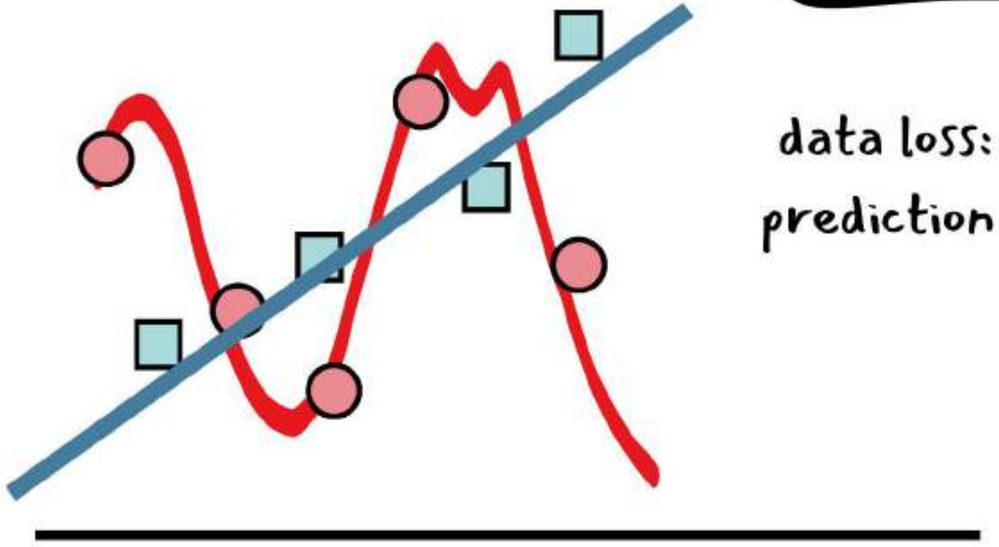
The **regularization term R(W)** is a term that helps to control the capacity of our model to generalize to unseen data. The parameter **lambda** is another hyperparameter, called the **learning rate**, that is essential for the learning task.

The loss function

The loss function establishes how good the classifier is.

If we have N examples of a images xi and their corresponding labels yi, the loss function over this dataset of N elements is given by the sum of the losses Li over each of the single examples plus a regularization term R(W):

$$L = \underbrace{\frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i)}_{} + \lambda R(W)$$



data loss: tries to make the model prediction match the training data

The loss function

The loss function establishes how good the classifier is.

If we have N examples of a images xi and their corresponding labels yi, the loss function over this dataset of N elements is given by the sum of the losses Li over each of the single examples plus a regularization term R(W):

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \underbrace{\lambda R(W)}$$

regularization: tries to make the model simple, to be able to work on test data

If we remove the regularization term from the loss function, the classifiers will tend to overfit the training data, and will loose its ability to generalize to test data or other datasets.

The loss function

The loss function establishes how good the classifier is.

If we have N examples of a images xi and their corresponding labels yi, the loss function over this dataset of N elements is given by the sum of the losses Li over each of the single examples plus a regularization term R(W):

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

What are examples of regularization terms?

L1 regularization
L2 regularization
Max norm
Dropout
Batch normalization

Which loss function will we use?

Multiclass SVM Loss (Hinge Loss)
Softmax Classifier (Cross entropy loss)

Multiclass SVM loss (Hinge Loss)

Indicating the result of the score function s = f(xi, W), we can write the Hinge loss for the sample {xi, yi} as:

$$L_i = \sum_{i \neq j} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1, \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$L_i = \sum_{i \neq j} max(0, s_j - s_{y_i} + 1)$$

s_{y_i}: score of the correct category

s_j: score of the incorrect category

training examples

CAT	**3.54**	-2.4	-3.1
DOG	1.23	**5.3**	1.23
TRAIN	-2.1	0.5	**2.0**

weights

L(im_1) = max(0,1.23-3.54+1) + max(-2.1-3.54+1) = 0

L(im_2) = max(-2.4-5.3+1) + max(0.5-5.3+1) = 0

L(im_3) = max(-3.1-2+1) + max(1.23-2+1) = 0.23

Loss over the full dataset:

L = L(im_1) + L(im_2) + L(im_3) = 0.23

Softmax classifier (Cross-entropy loss)

Given that the probability of a given label to be k given the input image xi is

$$P(Y = K|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

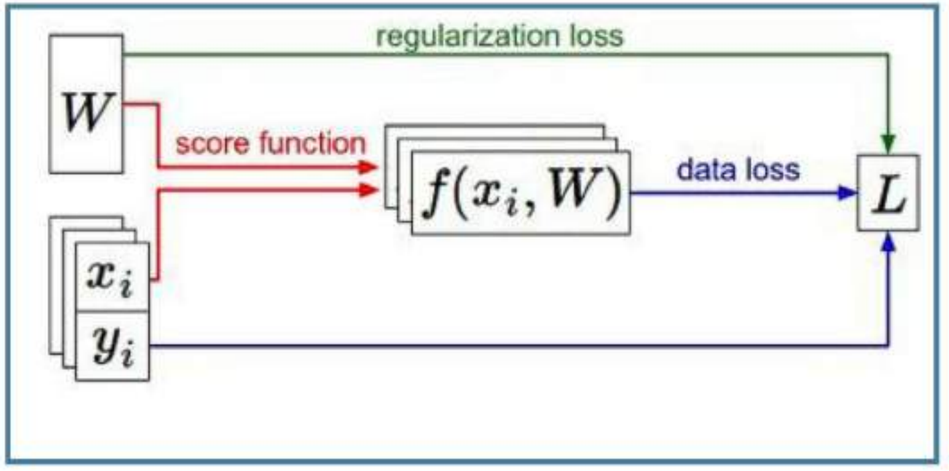
Since the loss function should minimize the negative log likelihood of the correct class, Li is:

$$L_i = -logP(Y = y_i|X = x_i) = -log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

and therefore, we can write the cross-entropy loss as:

$$L_i = -log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

Recap on linear classifier and loss + score functions



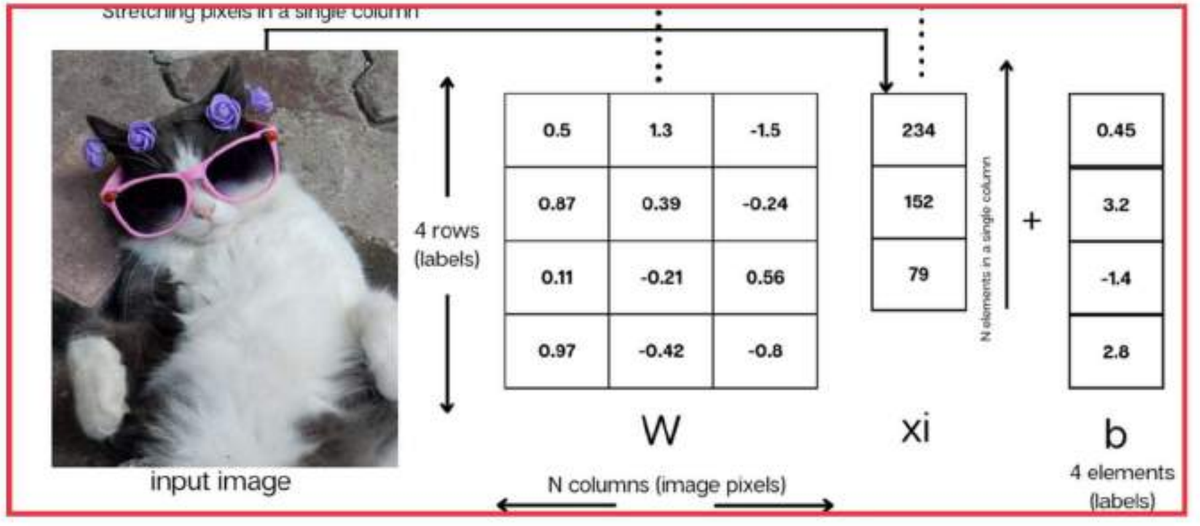
REGULARIZATION LOSS

L1 regularization
L2 regularization
Max norm
Dropout
Batch normalization

FULL LOSS

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, W), y_i) + \lambda R(W)$$

SCORE FUNCTION: MATRIX MULTIPLY + BIAS



DATA LOSS : HINGE OR SOFTMAX

$$L_i = \sum_{i \neq j} max(0, s_j - s_{y_i} + 1)$$

$$L_i = -log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$$

3

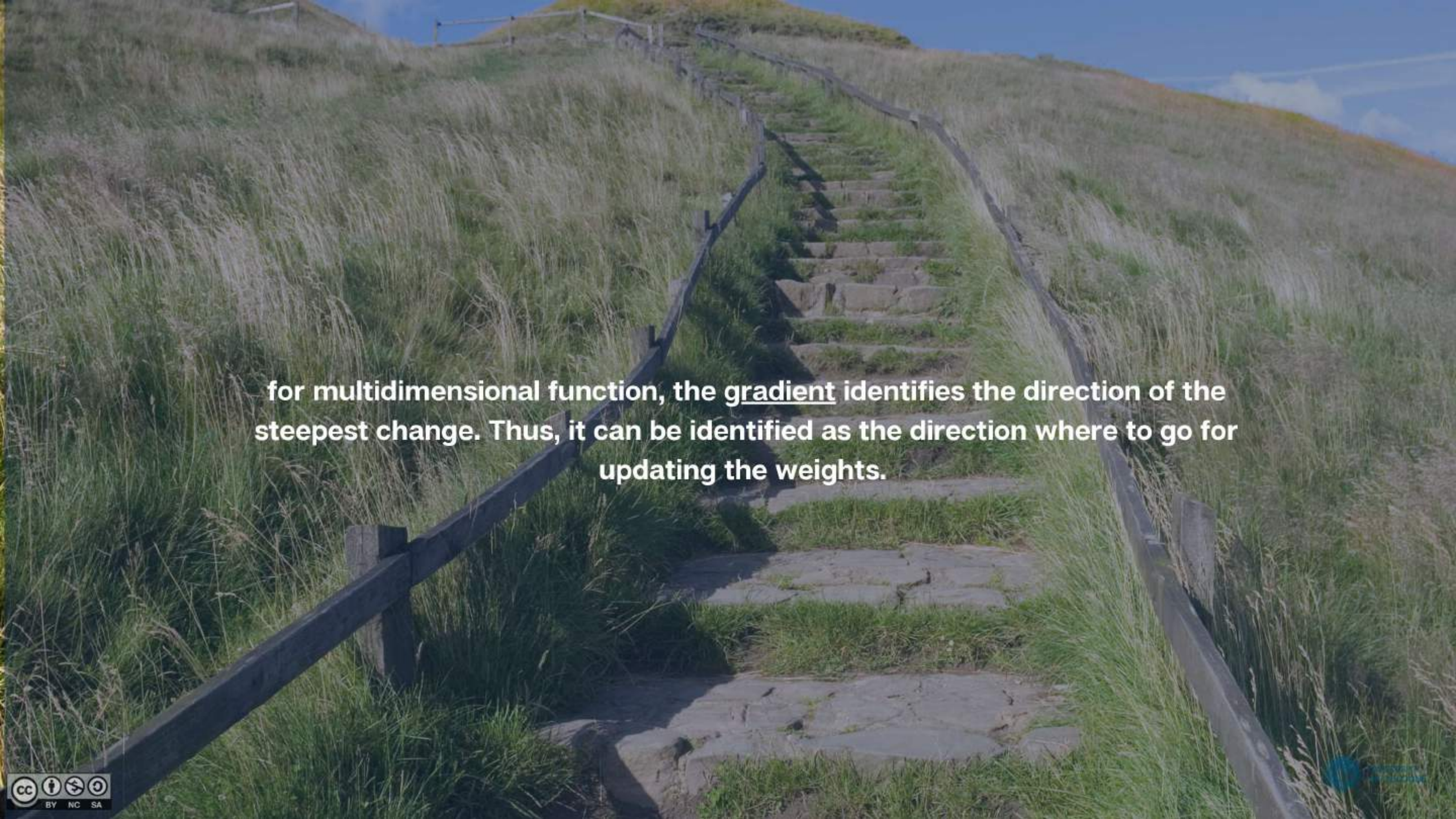
Learning process via optimization (and the various processes behind it)

You can try different random weights and see which one of them works best

(Random search, typically bad results, better for iterative refinements)

You can start with a random search and then apply perturbations to that, updating only if the loss of the perturbed state is lower than the original one.

(slightly better than before, but still far from being acceptable)


for multidimensional function, the gradient identifies the direction of the steepest change. Thus, it can be identified as the direction where to go for updating the weights.

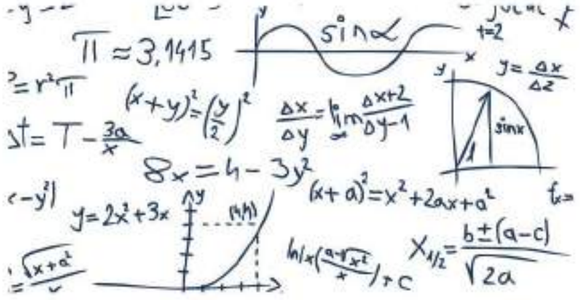
How to calculate the gradient?

Numerical calculation

$$\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Implement in a code the formulation of the gradient: where x is a vector the h is an array of increments, and f is an array of partial derivatives.

Calculus



can derive the exact expression of the gradient of the Loss function but it is more prone to errors

Usually, the way to proceed is to calculate the gradient analytically and then check with numerical gradient.

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

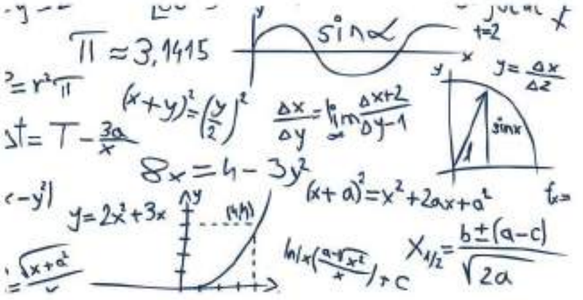
How to calculate the gradient?

Numerical calculation

$$\frac{\partial f}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Implement in a code the formulation of the gradient: where x is a vector the h is an array of increments, and f is an array of partial derivatives.

Calculus



can derive the exact expression of the gradient of the Loss function but it is more prone to errors

Usually, the way to proceed is to calculate the gradient analytically and then check with numerical gradient.

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

The procedure of iteratively calculating the gradient and then performing a parameter update is called **gradient descent**.
Calculating the gradient for the entire loss function is expensive when N is large
We approximate the sum over N to a sum over a subset of the full total of the examples, i.e. on a minibatch of 32/64/128 elements of the sum.

Backpropagation

Goal: derive analytic gradient of the loss function with respect to the weights W

$$\nabla_W L(W)$$

Backpropagation is a way to calculate gradients of given functions, by applying recursively the **chain rule.**

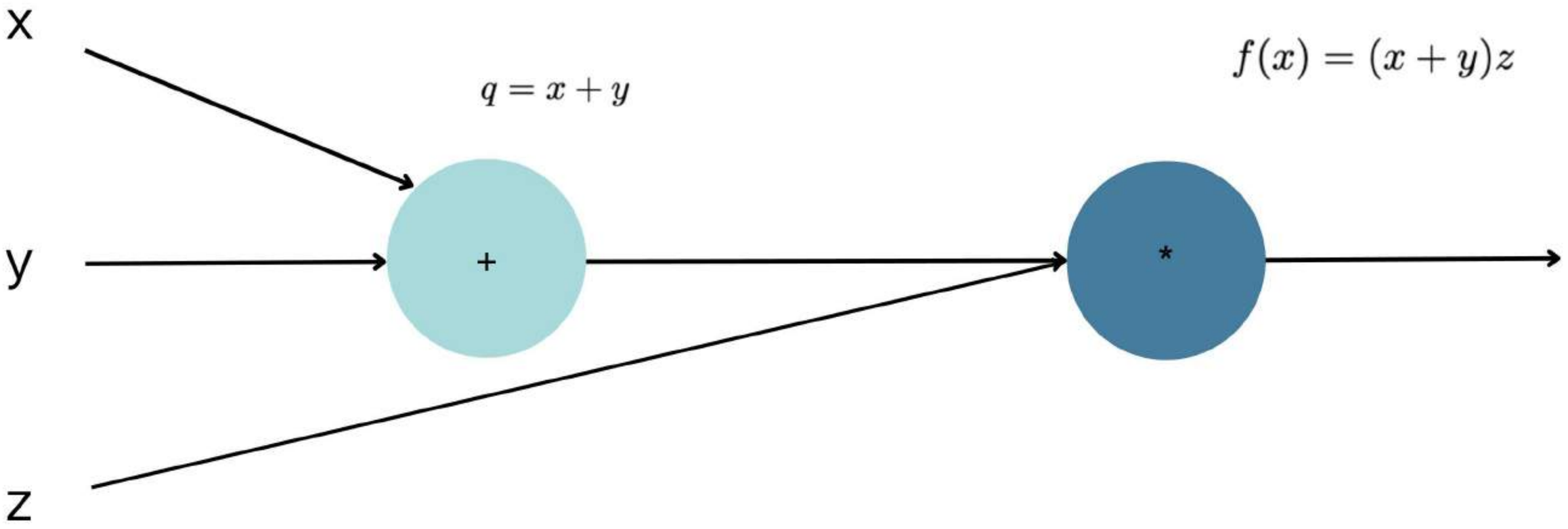
$$h(x) = f(g(x)) \qquad h'(x) = f'(g(x))g'(x)$$

Backpropagation consists of two main parts:

- **Forward pass**: the input goes through the network and provides a prediction.
- **Backward pass**: from the calculation of the gradient of the loss function at the final layer, recursively by applying the chain rule weights get updated in the network.

Example: we build an easy example, with the function $f(x) = (x + y)z$

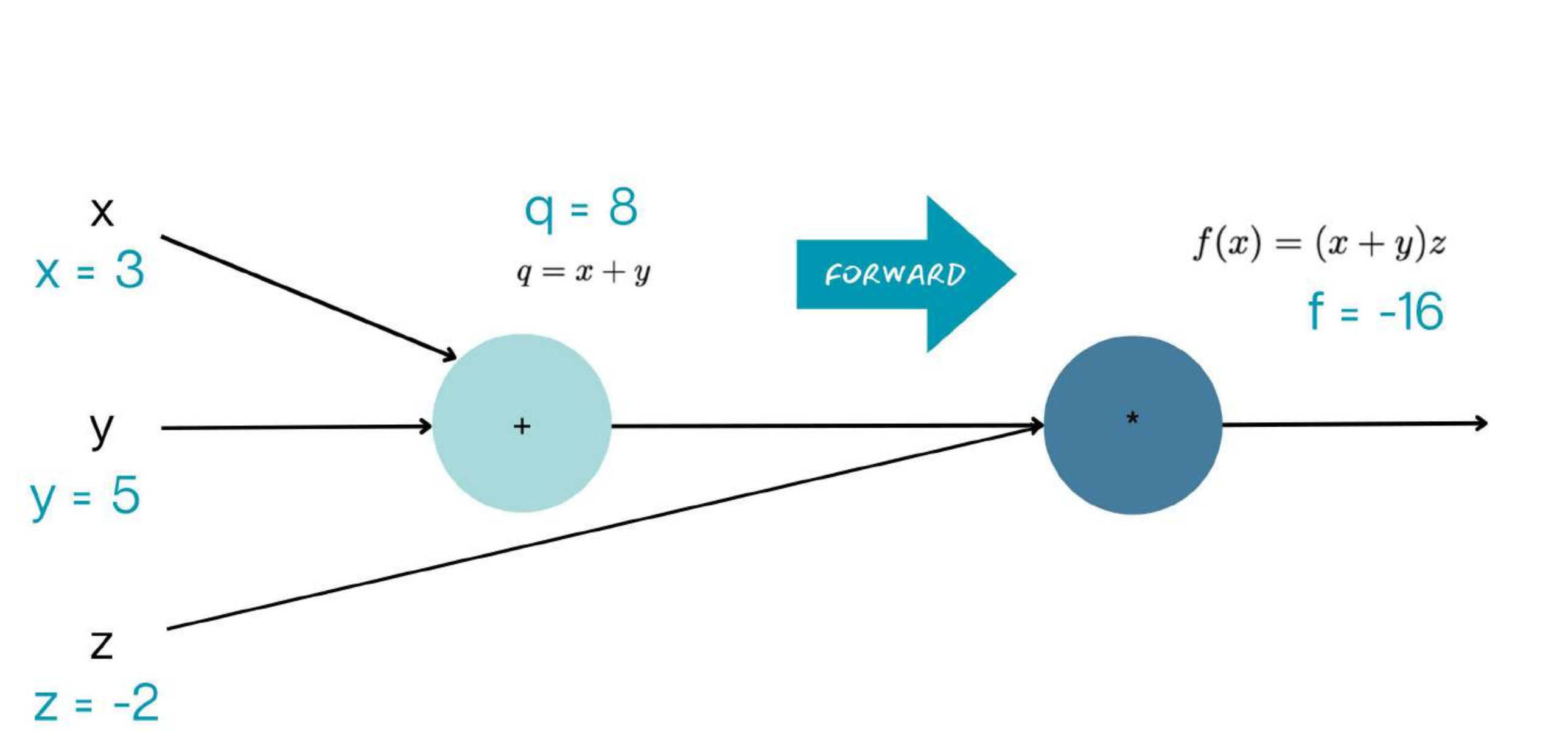
where we can define q = x + y, and it will become: $f(q, z) = qz \quad q = x + y$



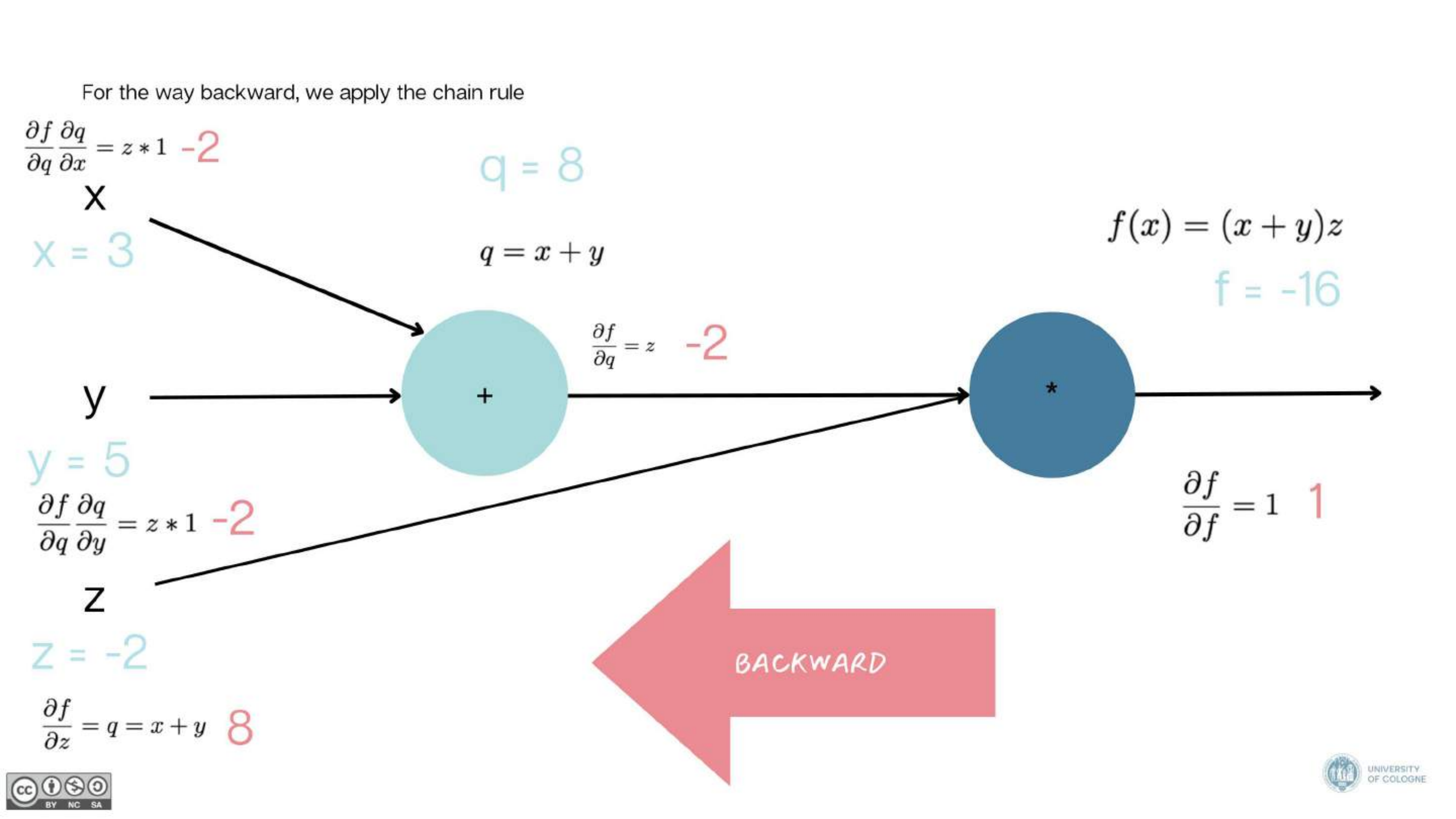
we can calculate all the partial derivatives:

$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial f}{\partial q} = z$$

$$\frac{\partial q}{\partial y} = 1 \qquad \frac{\partial f}{\partial z} = q = x + y$$

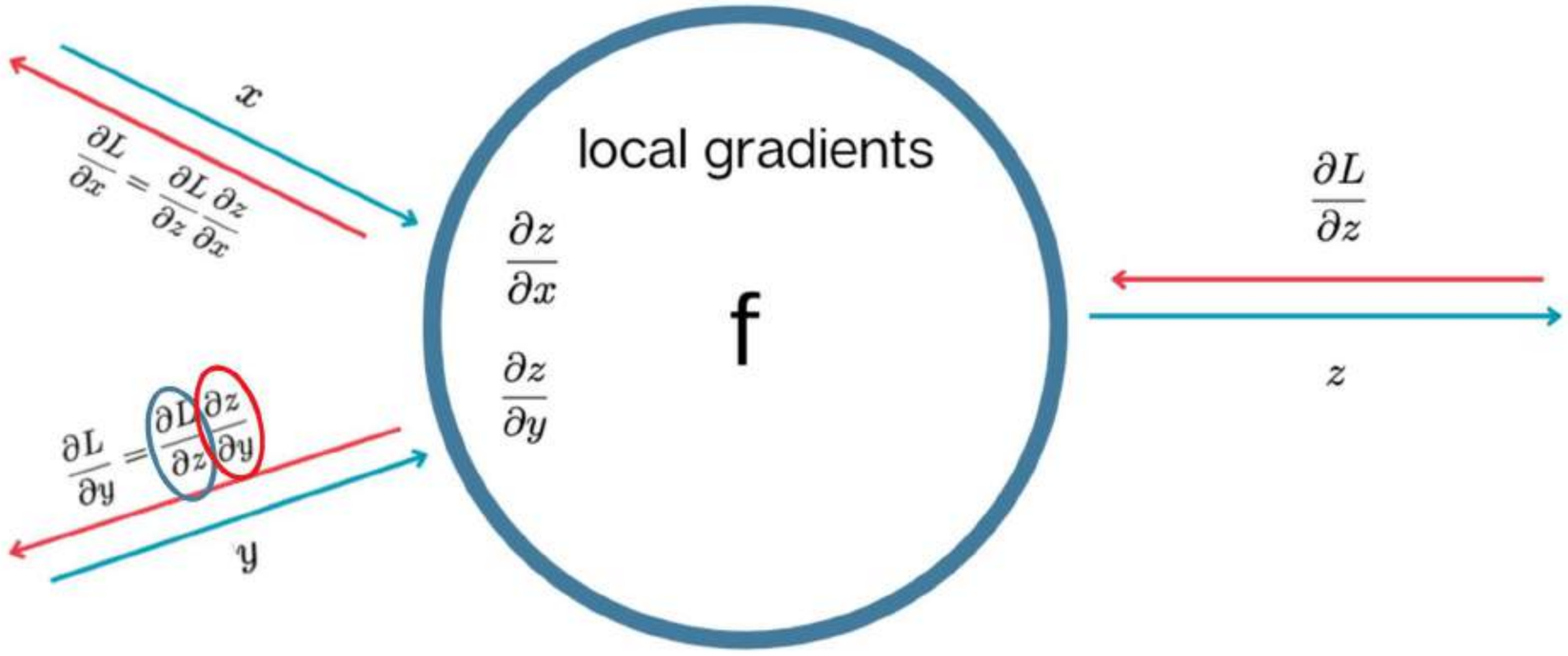


For the way backward, we apply the chain rule



Every gate in the diagram gets an input and, based on that, it can provide the output and the gradient of its output with respect to its inputs

this operation do not depend on the rest of the circuit in which the gate is located.



because of the chain rule, the gate multiplies the gradient of the loss function with respect to its output (blue circle) for the local gradients of its inputs (red circle), and passes it backwards to its inputs.

that's it for today!