Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Computer Vision
Classification 1

WS 2019/2020

Prof. Dr. Simone Frintrop

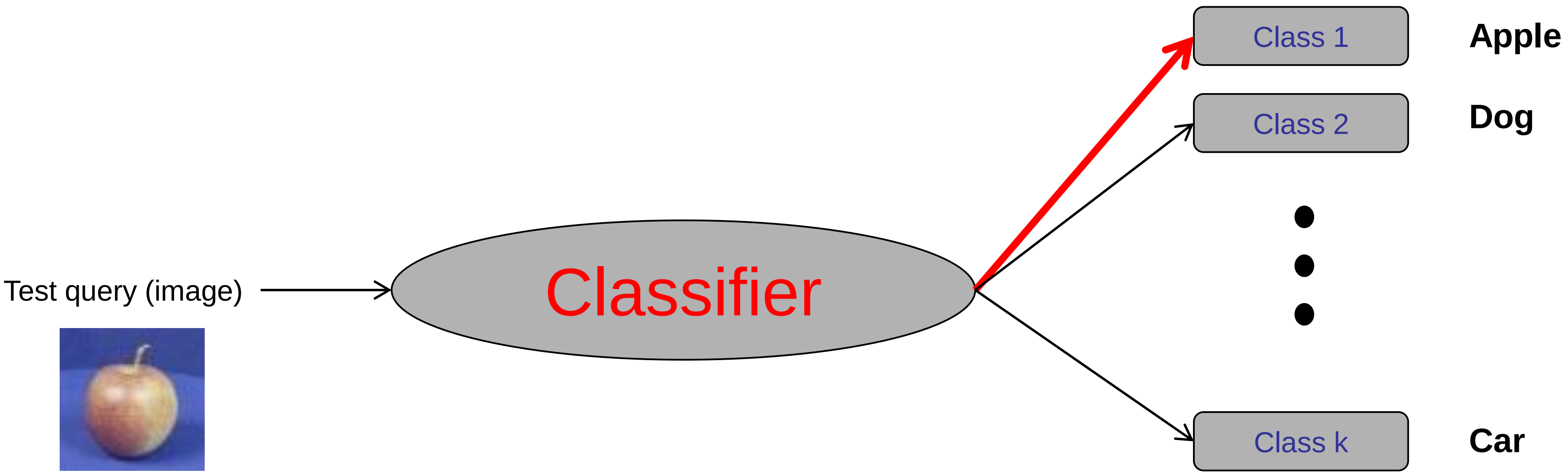
Computer Vision Group, Department of Informatics
University of Hamburg, Germany

Outline

- Classification: Problem definition and motivation
- Traditional approach versus deep learning
- Features for Recognition
- Classifier types
- Nearest Neighbor Classifier
- Linear Classification
- Loss functions
- Gradient descent

Classification

A classifier is an algorithm that assigns to each input image a class label

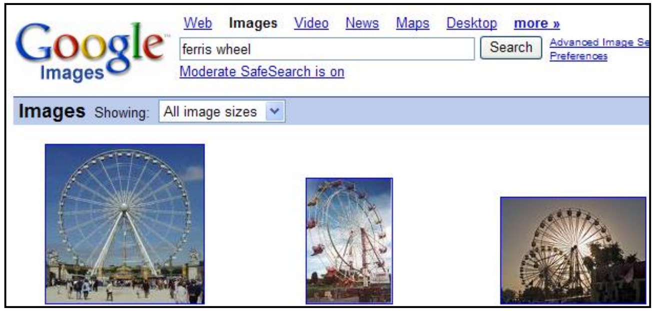


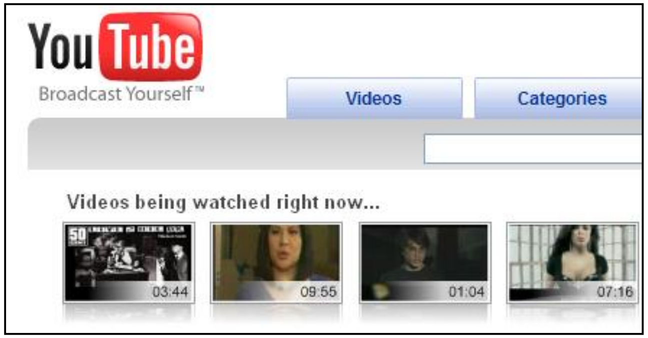
Object Categorization – Potential Applications

Autonomous robots

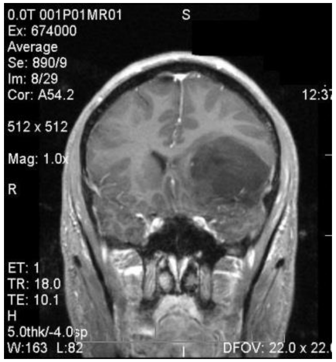
Navigation, driver safety

Consumer electronics





Content-based retrieval and analysis for images and videos



Medical image analysis

How many object categories are there?

1500 – 3000 basic categories (as chair or dog)
With subtypes: 10000 to 30000 discriminable objects (as “German shepherd dog”)

[Biederman 1987]

Identification vs. Categorization

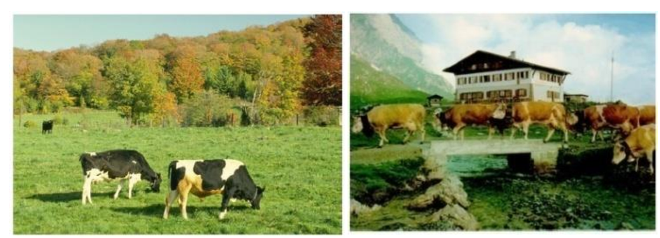
Identification/Instance Recognition

Find *this particular* object (cf. SIFT recognition)

Categorization/ Classification

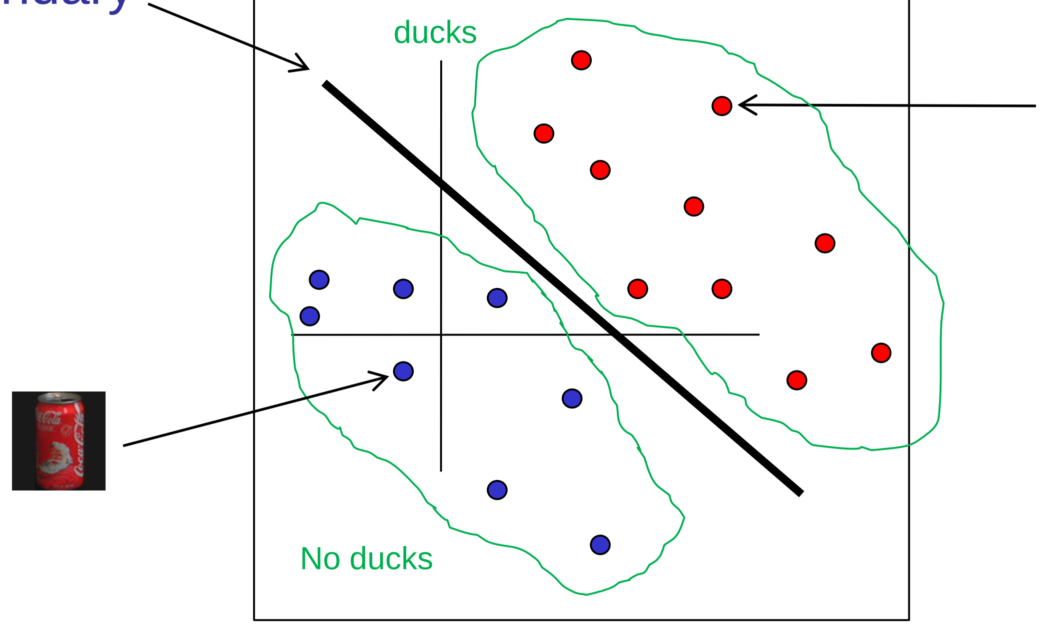
Find all instances of a category/class

Find any car/cow/…



Classification

- Data is given as vectors in a feature space
- The classifier assigns labels to data points according to a decision boundary



Each data point corresponds to an image or a feature representation of the image (given as a vector)

- Training a classifier means to determine a decision boundary explicitly or a rule which data point to label with which class (determines the decision boundary implicitly)

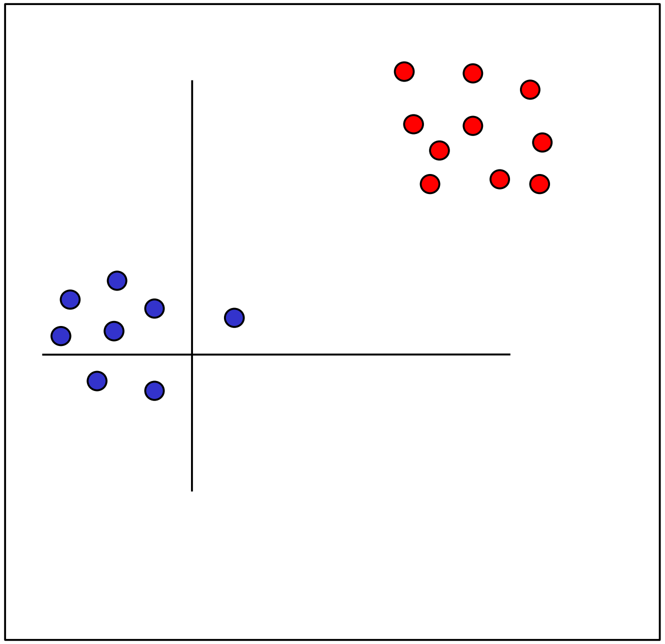
Classification

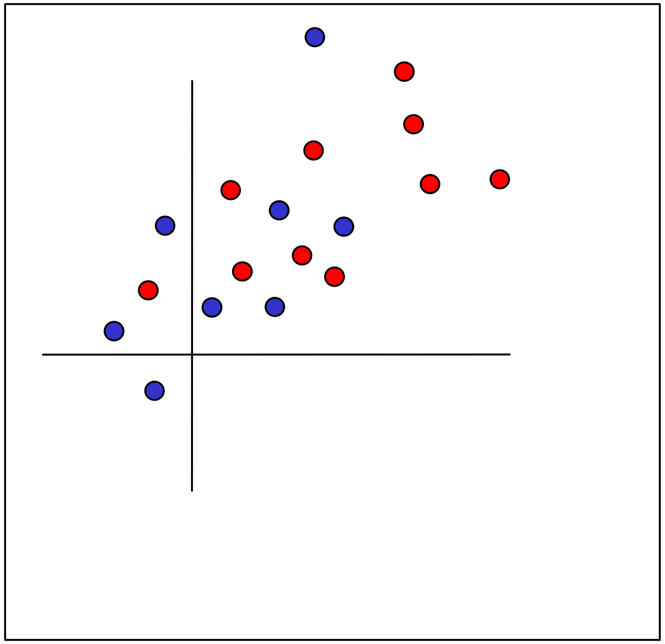
… is easy

if the items within one class are all similar (low *within class variance*) and different from the items of other classes (high *between-class-difference*) in the given feature space

… otherwise hard

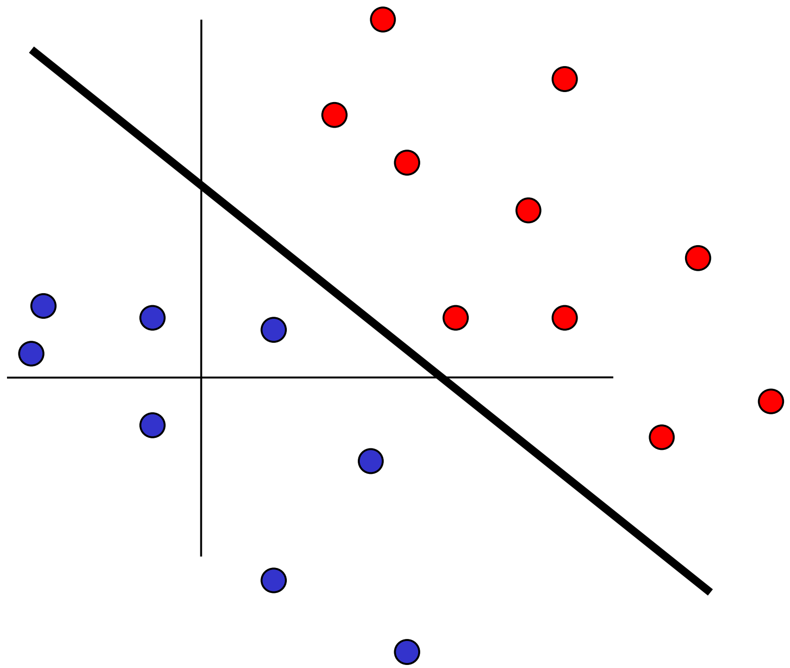
(if you are lucky,
you just chose the
wrong feature space)



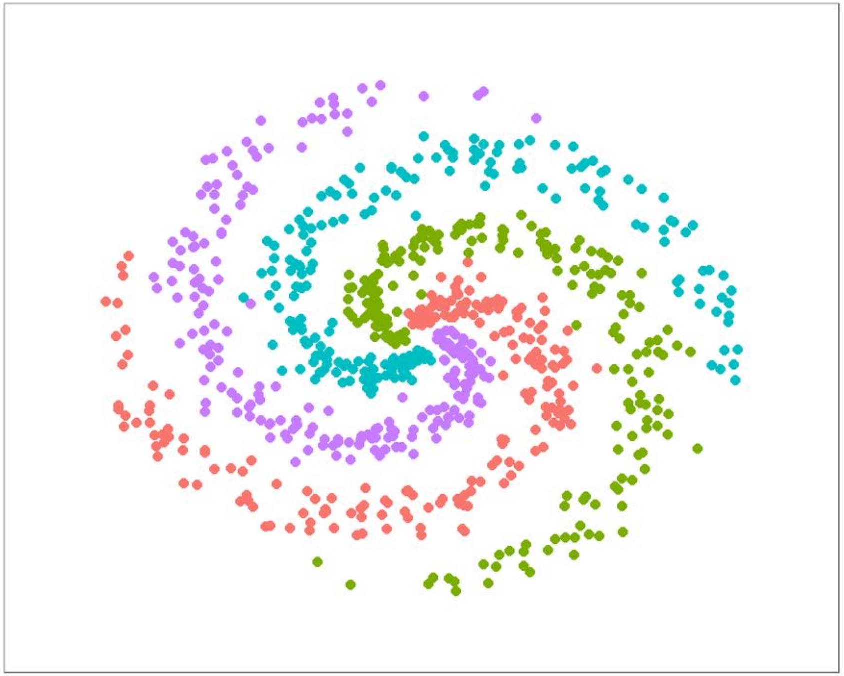


Linear vs. Non-Linear Classifiers

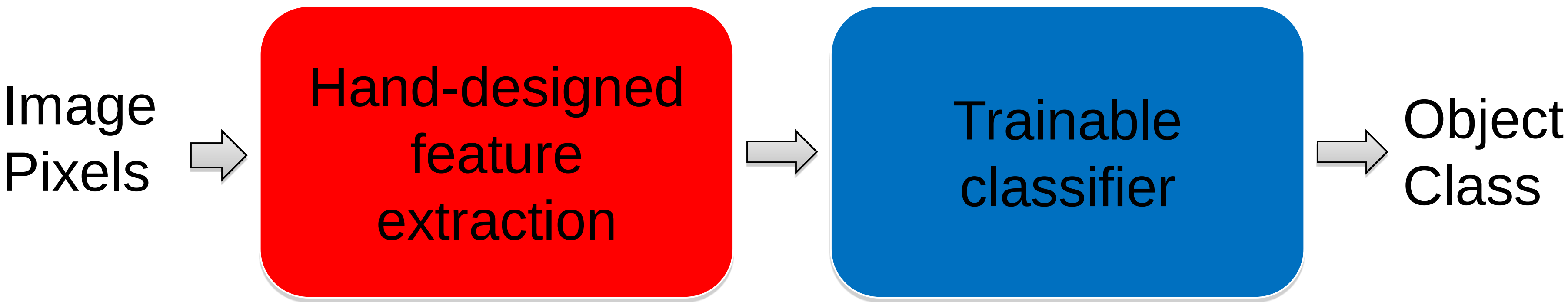
Data separable by linear classifier



Data separable only by non-linear classifier



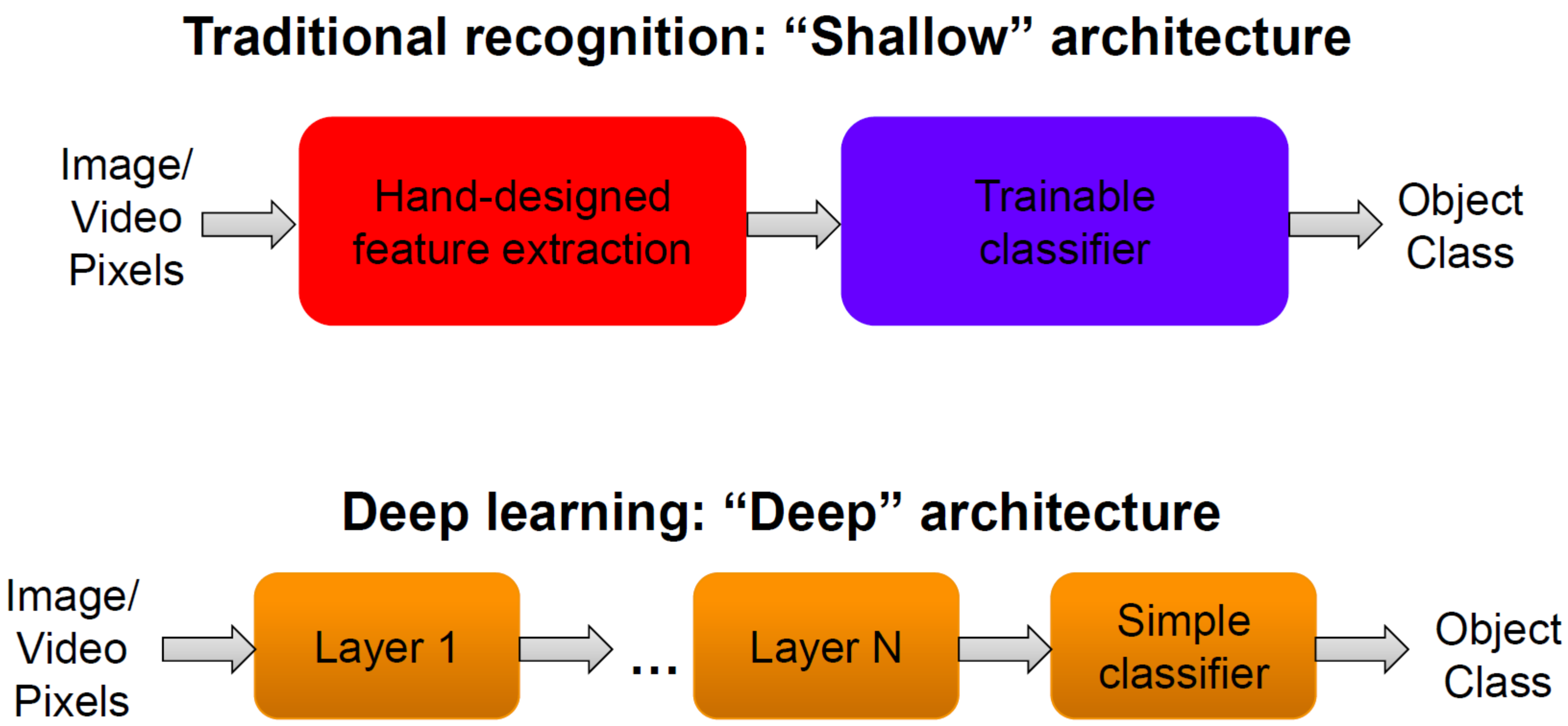
Traditional Recognition Pipeline



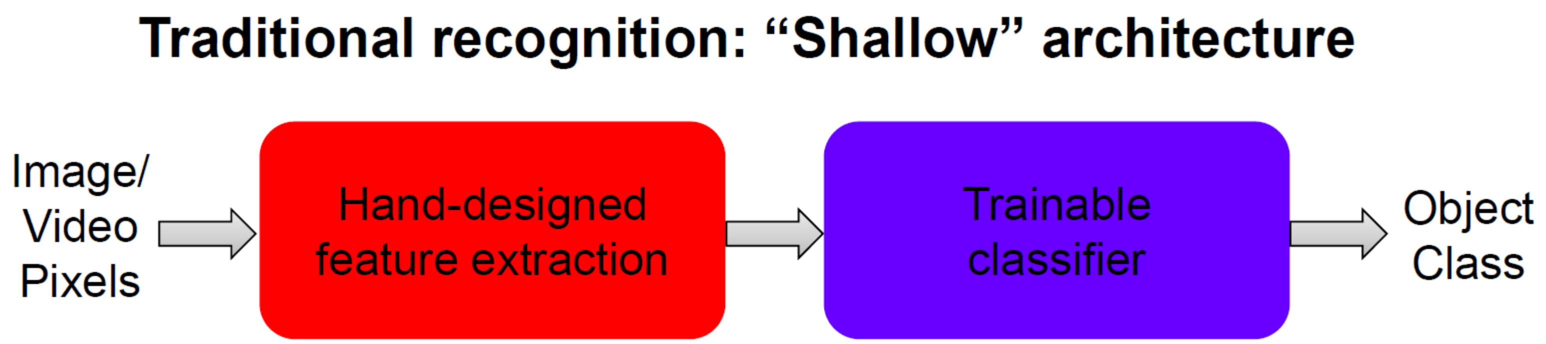
Deep learning

- Learn a *feature hierarchy* all the way from pixels to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly

Traditional vs. Deep architecture



Traditional vs. Deep architecture



Lets start with the traditional pipeline

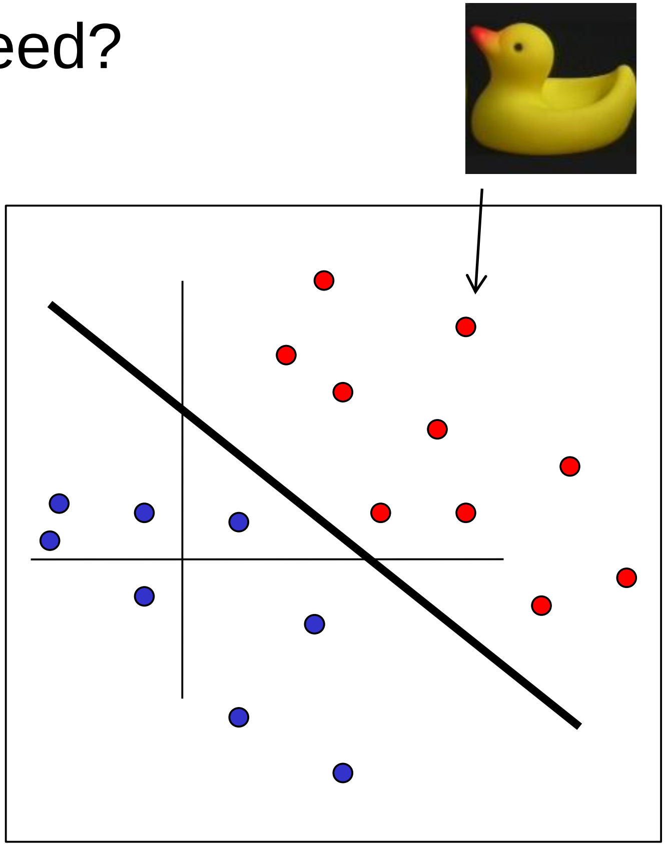
Two problems:

- Which features to select?
- Which classifier to take?

Features

What object representation do we need?

A vector that has the same length for each item/object

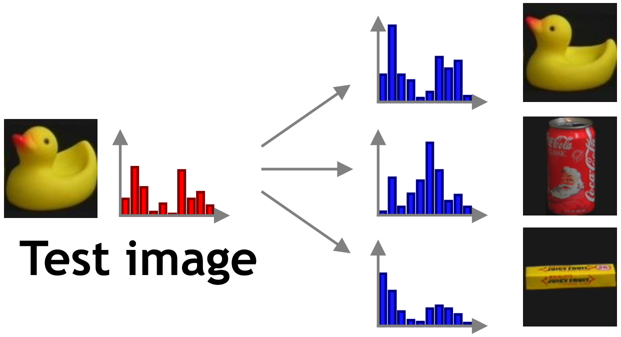


Which feature representation we covered is suitable?

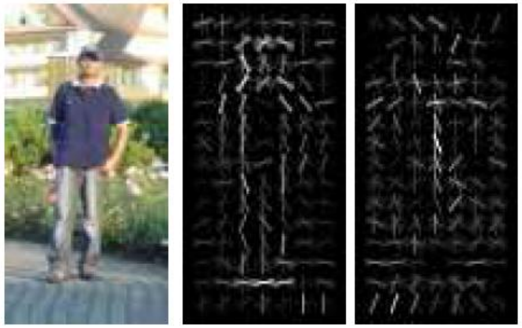
Features

Some feature representations we have seen:

- Color histograms



- HOG features



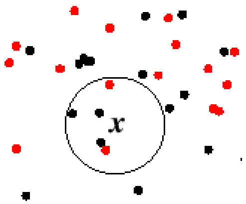
Features

- Why not SIFT?
- SIFT gives us MANY keypoints per object, and even worse: a different number for each object.
  we do not have a single vector per object.

- But: we can use DenseSIFT or SIFT in a Bag-of-Words approach (see: *Csurka, Dance, Fan, Willamowski & Bray (2004). "Visual categorization with bags of keypoints". Proc. of ECCV International Workshop on Statistical Learning in Computer Vision,*

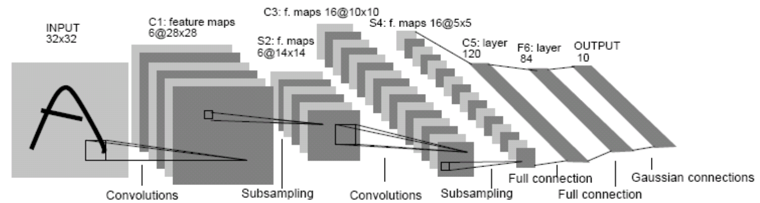
Classifiers

Nearest Neighbor



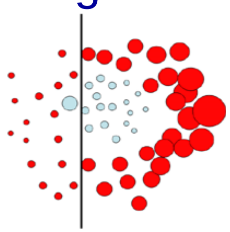
Berg, Berg, Malik 2005,
Chum, Zisserman 2007,
Boiman, Shechtman, Irani 2008, …

Neural networks



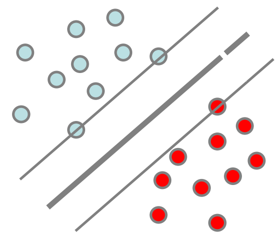
LeCun, Bottou, Bengio, Haffner 1998
Rowley, Baluja, Kanade 1998
…

Boosting



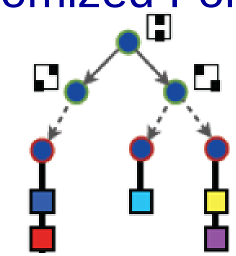
Viola, Jones 2001,
Torralba et al. 2004,
Opelt et al. 2006,
Benenson 2012, …

Support Vector Machines



Vapnik, Schölkopf 1995,
Papageorgiou, Poggio ‘01,
Dalal, Triggs 2005,
Vedaldi, Zisserman 2012

Randomized Forests



Amit, Geman 1997,
Breiman 2001,
Lepetit, Fua 2006,
Gall, Lempitsky 2009,…

The statistical learning framework

Apply a prediction function to a feature representation of the image to get the desired output:

$f($  $)$ = "apple"

$f($  $)$ = "tomato"

$f($  $)$ = "cow"

The statistical learning framework

$$\boldsymbol{y} = f(\boldsymbol{x})$$

Output
(class label/score)

prediction
function
(classifier)

Image
(or image feature
vector)

$f($$)$ = “apple”

$f($$)$ = “tomato”

$f($$)$ = “cow”

- **Training:** given a *training set* of labeled examples $\{(x_1, y_1), \ldots, (x_N, y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to an unseen *test example* x and output the predicted value $y = f(x)$

A simple linear classifier

A simple linear function (1D input):

$$y = wx + b$$

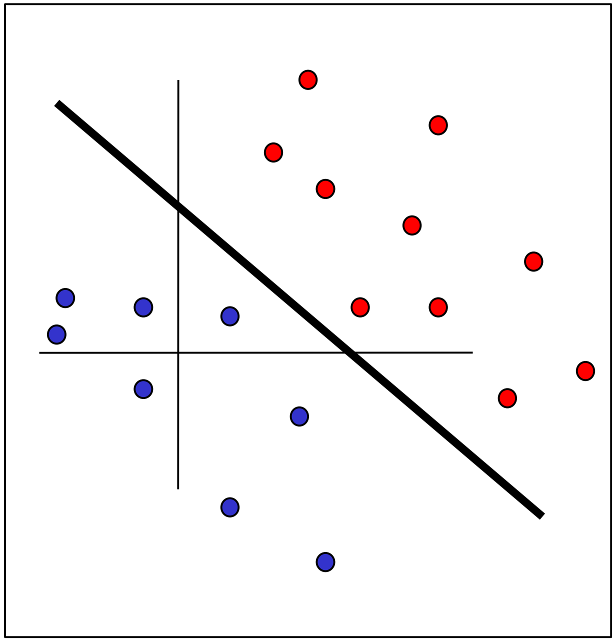
It separates two classes (binary classification)

Multi-dimensional input $\mathbf{x}$ (e.g. image with n = x*y pixels)

$$y = \mathbf{w}^T\mathbf{x} + b$$

Note: **w**, **x** are vectors now

This is the same as: $y = \sum_{i=1}^{n} w_i x_i + b$

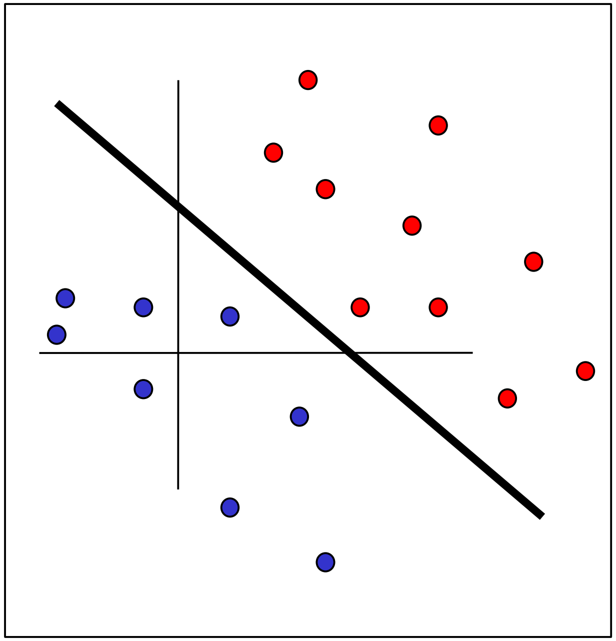


Parametric approach

Before: $y = f(x) = \mathbf{w}^T\mathbf{x} + b$

Consider now the parameters not as part of the function, but as input that can be trained (parametric approach):

$$y = f(\mathbf{x}, \mathbf{w}, b)$$



Find function f with a set of parameters $\boldsymbol{w}$ and b that maps images to classes

We want to learn the ***w*** and *b* that optimize this function.

The statistical learning framework

Find function f with a set of parameters $\mathbf{w}$ and b that maps images to classes:

$f($  $, W, b) =$ "apple"

$f($  $, W, b) =$ "tomato"

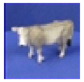
$f($  $, W, b) =$ "cow"

Classifiers

The following questions remain:

$f($  $, W, b) =$ "apple"

$f($  $, W, b) =$ "tomato"

$f($  $, W, b) =$ "cow"

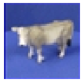
- What function can we choose for f?
  (Which classifier shall we use?)
- How can we measure how good the classifier is?
  (Define a loss function)
- How can we optimize the classifier?
  (Find better *W,b*)

Classifiers

The following questions remain:

$f($  $, W, b) =$ "apple"

$f($  $, W, b) =$ "tomato"

$f($  $, W, b) =$ "cow"

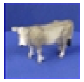
- What function can we choose for f? (Which classifier shall we use?)
- How can we measure how good the classifier is? (Define a loss function)
- How can we optimize the classifier? (Find better *W,b*)

Classifiers

- What function can we choose for f?
- Well-known classifiers:
  - Logistic regression
  - Support Vector Machine
  - Boosting
  - Random Forests
  - (Deep) Neural Networks

$f($  $, W, b) =$ "apple"

$f($  $, W, b) =$ "tomato"

$f($  $, W, b) =$ "cow"

We will cover here mainly Deep Neural Networks, but let‘s continue now with the big picture

Classifiers

- Given a function f that maps images to classes, and given a set of parameters W and b:
  What output do we get?
- Usually a classification score that measures how certain the classifier is for each class.
  The score is a vector. Size: number of classes
- Binary case:

$f($  $, \boldsymbol{w}, b)$: $\begin{pmatrix} 0.88 \\ 0.12 \end{pmatrix}$ tomato / no tomato

Classifiers

- Given a function f that maps images to classes, and given a set of parameters W and b:
  What output do we get?
- Usually a classification score that measures how certain the classifier is for each class.
  The score is a vector. Size: number of classes
- Multi-class case:

$f($  $, \boldsymbol{W}, \boldsymbol{b})$:

$$\begin{array}{cl} \begin{pmatrix} 0.54 \\ 4.78 \\ -2.56 \\ \cdot \\ \cdot \\ \cdot \end{pmatrix} & \begin{array}{l} \text{Car} \\ \text{Tomato} \\ \text{Chair} \\ \\ \\ \\ \end{array} \end{array}$$

(whether the values sum up to 1 or not depends on the classifier)

A simple linear classifier

A simple linear function (1D input):

$$y = wx + b$$

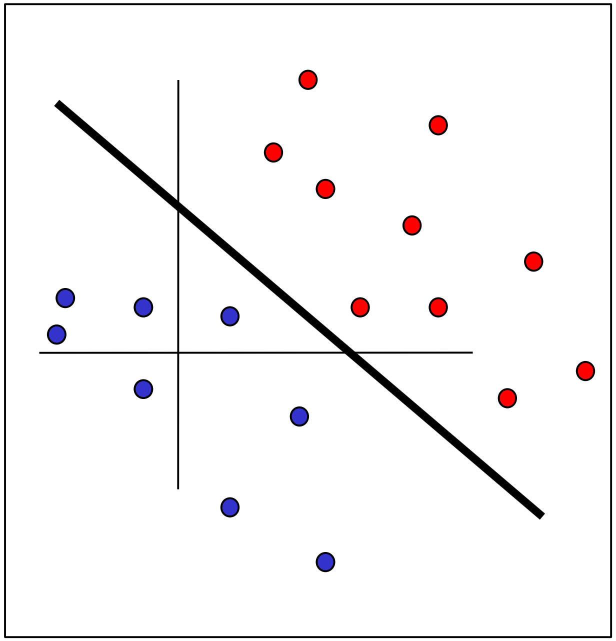
It separates two classes (binary classification)

Multi-dimensional input $\mathbf{x}$ (e.g. image with n = x*y pixels)

$$y = \mathbf{w}^T\mathbf{x} + b$$

Note: **w**, **x** are vectors now

This is the same as: $y = \sum_{i=1}^{n} w_i x_i + b$



Multi-Class Linear Classifier

Let‘s extend our linear classifier to multi-class:

Up to now we had:

$$y = \mathbf{w}^T \mathbf{x} + b$$

(multi-dimensional input x => vectors **x** and **w**)

Now we need: multi-dimensional output **y**:

$$\mathbf{y} = f(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

where **x**, **b** and **y** are vectors and **W** is a matrix

$$\begin{array}{cl} \left(\begin{array}{c} 0.54 \\ 4.78 \\ -2.56 \\ . \\ . \\ . \end{array}\right) & \begin{array}{l} \text{Car} \\ \text{Tomato} \\ \text{Chair} \\ \\ \\ \\ \end{array} \end{array}$$

Example

Image from CIFAR-10 dataset:

[32x32x3]

= 3072 pixels

What are the dimensions of the vectors/matrix?

$$\mathbf{y} = f(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

CIFAR-10 dataset: 60000 color images of 32x32 pixels in 10 classes

Example

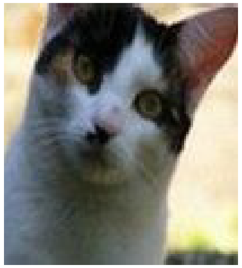
$$\boxed{f(x, W)} = \boxed{W}\boxed{x} \quad (+b)$$

3072x1

10x1

10x1

10x3072



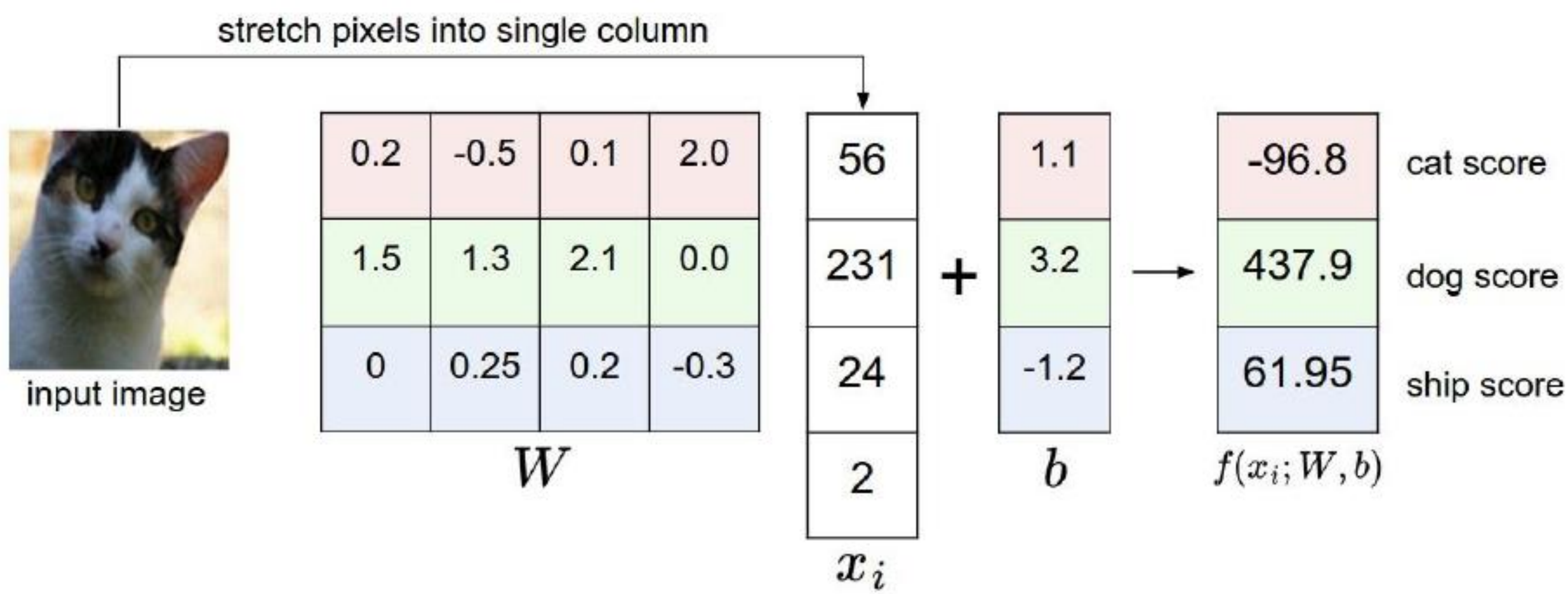
[32x32x3]
array of numbers 0...1

10 numbers, indicating class scores

parameters, or "weights"

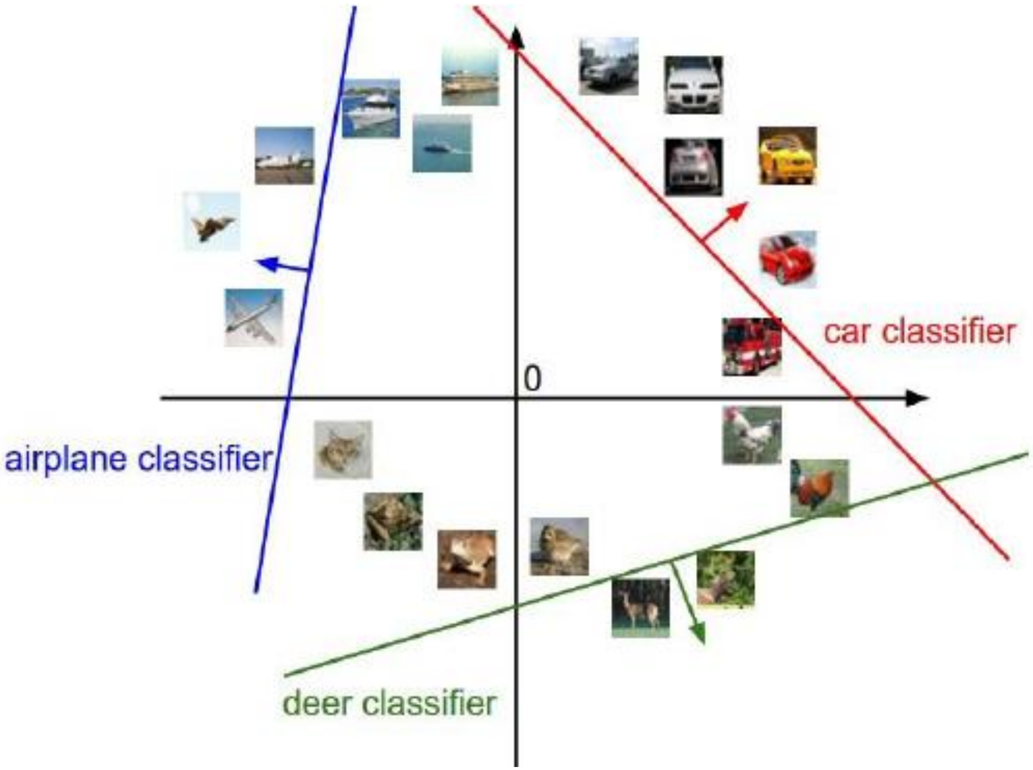
Linear classifier

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



Each row of $\boldsymbol{W}$ is a linear classifier.

Interpreting a Linear Classifier



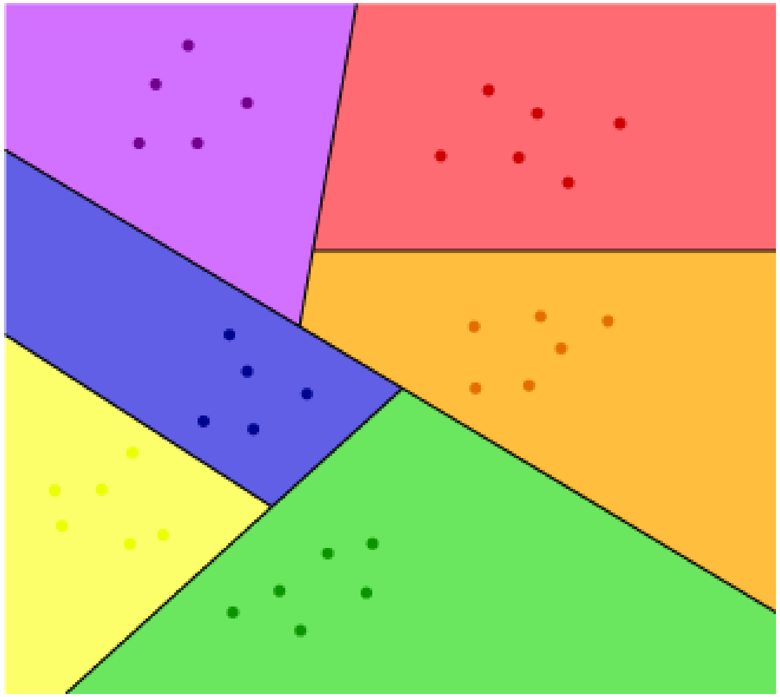
$$f(x_i, W, b) = Wx_i + b$$

[32x32x3]
array of numbers 0...1
(3072 numbers total)

Each row of **W** is a classifier for one of the classes, visualized here as line

Multi-Class Classification

Example of the decision boundaries learnable with a linear classifier for multiple classes:



(in higher dimensions, the lines are hyperplanes)

https://shapeofdata.wordpress.com/2013/06/04/multi-class-classification/

Multi-Class Linear Classifier

Up to now: equations for *one* training image x:

$$\mathbf{y} = f(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

where **x**, **b** and **y** are vectors and **W** is a matrix

Extension to multiple training images:

$$\mathbf{Y} = f(\mathbf{X}, \mathbf{W}, \mathbf{B}) = \mathbf{W}\mathbf{X} + \mathbf{B}$$

where **Y**, **X, W,** and **B** are all matrices

=> matrix-oriented programming languages like Matlab or Python are very convenient to compute this

Equation overview

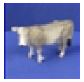
	Classification Type	Input x	Parameters w/W	Bias b	Output y
$y = wx + b$	Binary (two classes)	scalar	scalar	scalar	scalar
$y = \mathbf{w}^T\mathbf{x} + b$	Binary (two classes)	vector (e.g. image)	vector	scalar	scalar
$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$	Multi-class	vector (e.g. image)	matrix	vector	vector
$\mathbf{Y} = \mathbf{W}\mathbf{X} + \mathbf{B}$	Multi-class	matrix (several images)	matrix	matrix	matrix

Classifiers

The following questions remain:

$f($  $, W, b) =$ "apple"

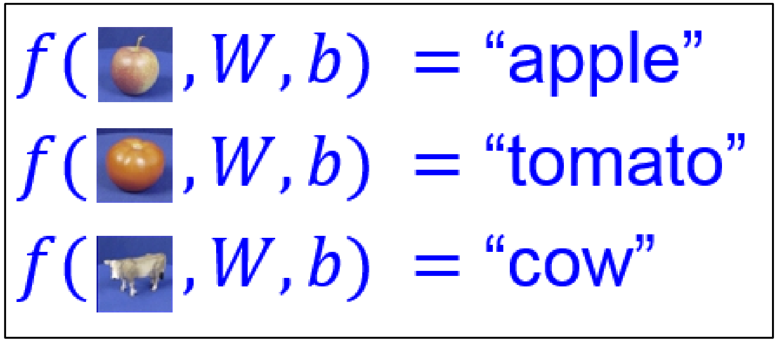
$f($  $, W, b) =$ "tomato"

$f($  $, W, b) =$ "cow"

- What function can we choose for f?
  (Which classifier shall we use?)
- How can we measure how good the classifier is?
  (Define a loss function)
- How can we optimize the classifier?
  (Find better *W,b*)

How good is our function?

- Given a function f that maps images to classes, and given a set of parameters $\boldsymbol{W}$ and $\boldsymbol{b}$.



$$y = f(\mathbf{x}, \mathbf{W}, \mathbf{b})$$

- How good is f?

- We can compute the loss over the training data

Loss functions

- A loss function (also cost function) measures how good your result is
- It compares the given output with the target output

Loss

$f($  $, W, b)$ = "cow" correct☺ 0 Don't punish the classifier for this

$f($  $, W, b)$ = "tomato" wrong☹ 1 Punish the classifier for this

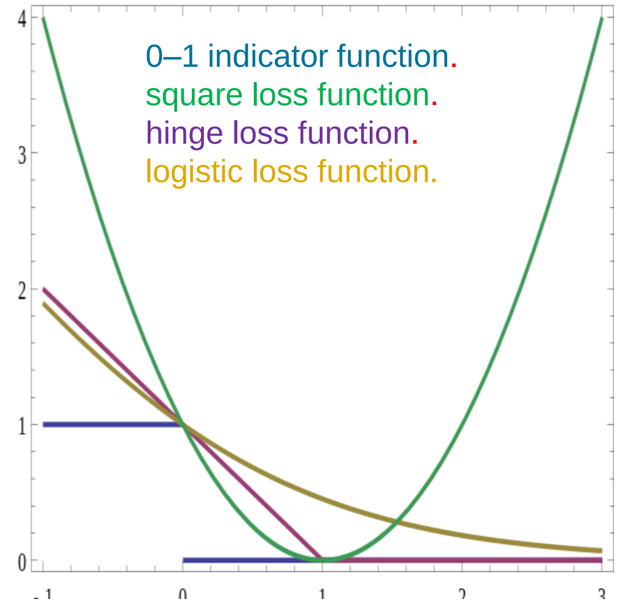
$f($  $, W, b)$ = "cow" wrong☹ 1

Average loss: 2/3

- This is the 0-1 loss. It is good for illustration purposes, but usually not useful (non-convex, non smooth)

Loss functions

- A loss function (also cost function) measures how good your result is
- It compares the given output with the target output
- Popular loss functions:
  - 0-1 loss/0-1 indicator function
  - Mean square loss
  - SVM loss (hinge loss)
  - Logistic loss
  - Cross-entropy loss



[Wikipedia: loss functions for classification]

- The choice of your loss function is not arbitrary. It has to fit to the model and the problem!

Full Loss

Given a loss function that computes the loss L_i for a given input image, we obtain the full loss L by averaging over all N training images:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$

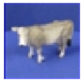
where $R(\mathbf{W})$ is a regularization term that prevents overfitting (regularization will be covered in more detail in summer term lecture „Computer Vision 2“)

Classifiers

The following questions remain:

$f($  $, W, b) =$ "apple"

$f($  $, W, b) =$ "tomato"

$f($  $, W, b) =$ "cow"

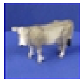
- What function can we choose for f?
  (Which classifier shall we use?)
- How can we measure how good the classifier is?
  (Define a loss function)
- How can we optimize the classifier? (Find better *W*,*b*)

Classifiers

How can we optimize the classifier?

$f($  $, W, b)$ = "apple"

$f($  $, W, b)$ = "tomato"

$f($  $, W, b)$ = "cow"

Q: what is a „better" classifier?

A: it has a lower loss => minimize the loss

Q: What can we change to obtain a lower loss?

A: the parameters *W* and *b*

Gradient descent

- Plot loss function dependent on the weights:
- A convex loss function $L(w)$ for a single weight w:

$L(w)$

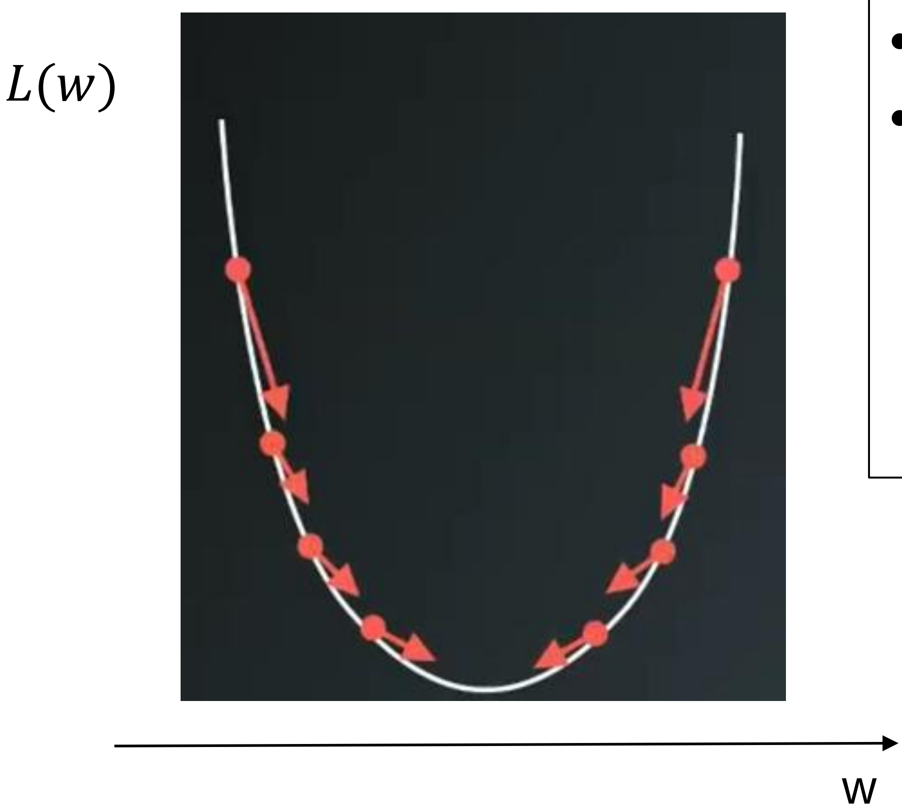
w

Find the minimum by walking downwards: *Gradient descent*

We follow the slope! The slope is given by the derivative of $L(w)$:

$$\frac{dL(w)}{dw}$$

Gradient descent



Gradient descent:

- Initialize w randomly.
- Repeat until convergence

{

$$w := w - \alpha \frac{dL(w)}{dw}$$

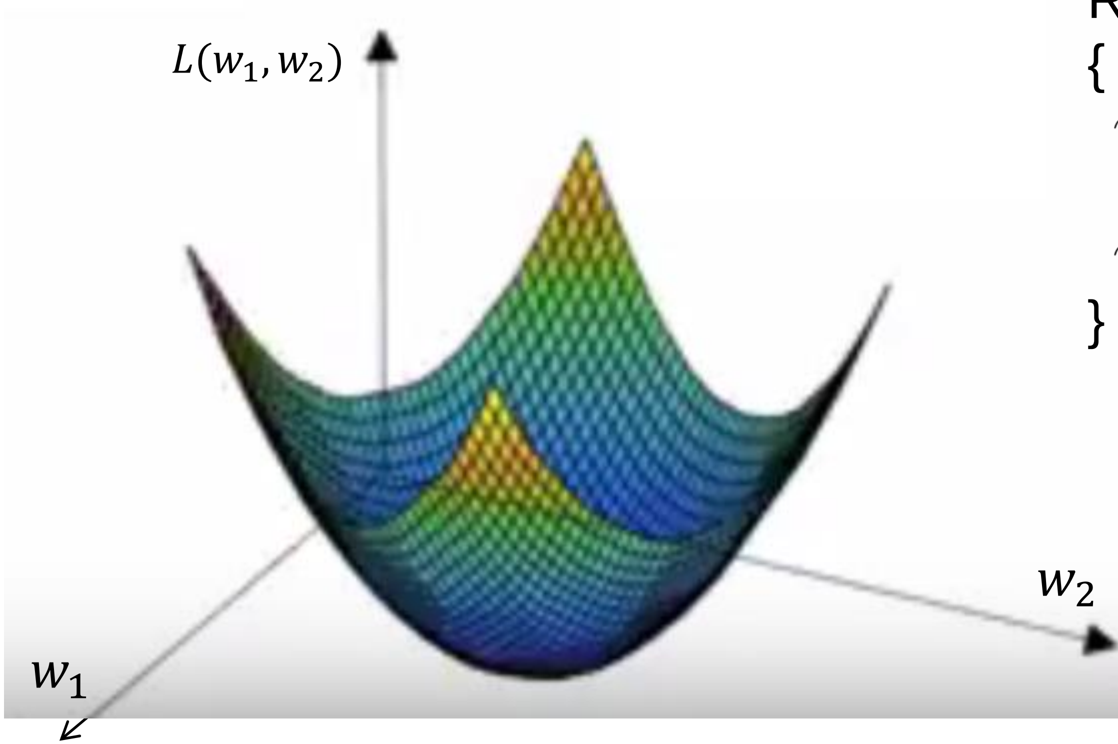
}

Learning rate

This means: update weight value w by subtracting a portion of the derivative

Gradient descent

If we have two weights with loss function $L(w_1, w_2)$:



Initialize w randomly
Repeat until convergence:
{

$$w_1 := w_1 - \alpha \frac{\partial L(w_1, w_2)}{\partial w_1}$$

$$w_2 := w_2 - \alpha \frac{\partial L(w_1, w_2)}{\partial w_2}$$

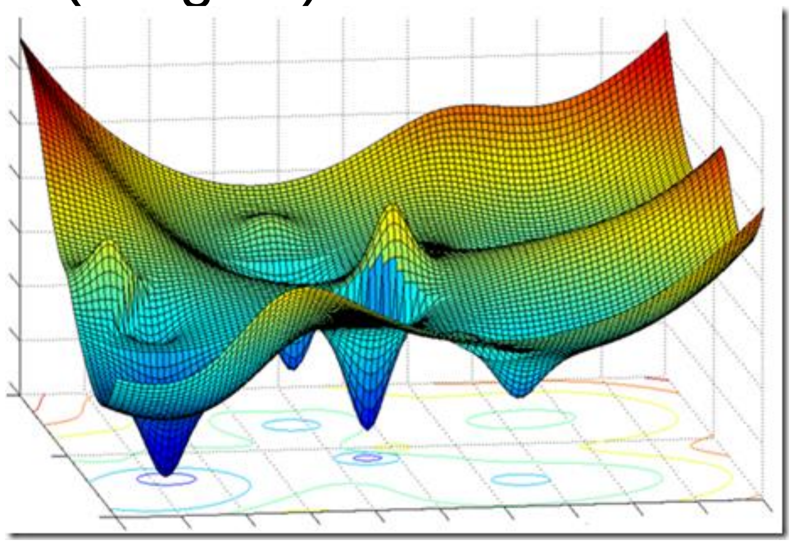
}

Note: the result of updating both weights means to step into the direction of the negative gradient!

Optimizing the loss

For a non-linear function (like a multi-layer neural network) the loss function is usually more complicated:

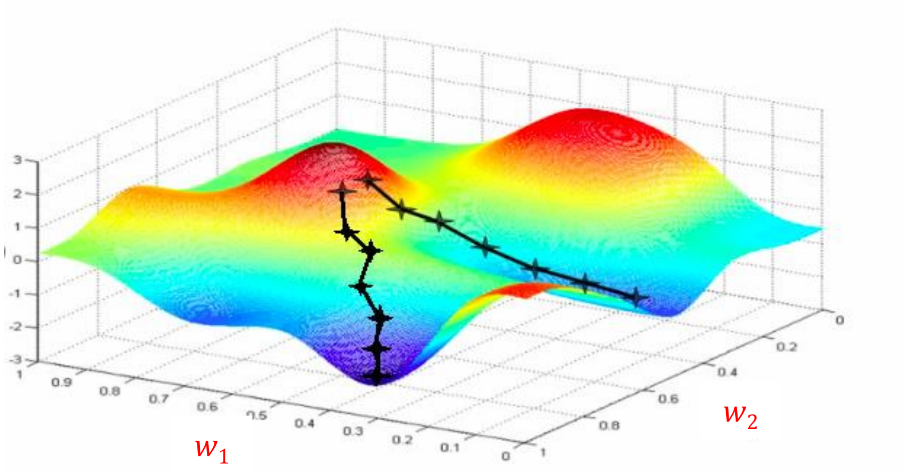
For 2 parameters (weights):



And we have usually thousands/millions of weights (dimensions)!

Optimizing the loss

- But we can still use gradient descent



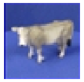
- Note: does only converge to *local* optimum
- But: it often works very well in practice (local optima often are already a good solution)
- Gradient descent is the standard optimization in deep learning

Brief summary

- Choose a data representation *x* (features or simply pixels)
- Select a classifier that computes *f(x,W,b)* (the score)
- Compute the loss
- Optimize *f* by finding better parameters *W,b* that minimize the loss

$f($  $, W, b) =$ "apple"

$f($  $, W, b) =$ "tomato"

$f($  $, W, b) =$ "cow"

- Next slide set: deep learning to compute *f*

Primary literature

You find most of these contents in most machine learning books, however not always related to images and seldomly in condensed form. Read e.g. the related parts from:

- Gonzales/Woods 2018: relevant parts from chapter 13
- Goodfellow et al, "Deep Learning", 2016: relevant parts from chapter 5 (machine learning basics)
- Course notes: CS231n Convolutional Neural Networks for Visual Recognition (lectures 1-3), from Stanford lecture of Fei Fei Li, Andrej Karpathy and Justin Johnson
  http://cs231n.github.io/
  (Corresponding video lectures on Youtube also recommended!)

Secondary literature

- Video lecture: “Deep Learning by Andrew Ng” https://www.youtube.com/playlist?list=PLBAGcD3siRDguyYYzhVwZ3tLvOyyG5k6K
- Biederman, I. (1987). Recognition-by-components: a theory of human image understanding. Psychological review, 94(2):115.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). ImageNet large scale visual recognition challenge. International Journal of Computer Vision, 115(3):211–252.
- Boiman/Shechtman/Irani: “In defense of Nearest-Neighbor Based Image Classification”, CVPR 2008